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
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

**Shift Left; Formal Bug Hunting**
- Level 1: Auto Formal
  - Auto Checks
  - X-propagation
  - Unreachability
- Level 2: Formal Apps
  - RTL Assertions
  - Arbiter
  - FIFO
  - Handshake
  - Bus Protocol
- Level 3: ABV Formal
  - SVA
  - PSL
  - OVL

**Exhaustive: Formal Sign-Off**
- Level 4: Connectors
  - Block Sign-Off
  - Connectivity
  - Register checks
  - Clock gating
  - Sequential LEC
- Level 5: System Arch. Verification
  - System Deadlock
  - Cache Coherence
  - Sys-Level Security
  - Load/Store Unit
  - Warp Sequencer
  - Cache Controller
  - Multi-Lane Aligner
  - MAC Rx Block

- White-box approach
- Black-box approach (ideally)
Block-Level Formal Signoff

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

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

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

<table>
<thead>
<tr>
<th>Task</th>
<th>Planning</th>
<th>Implementation</th>
<th>Closure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Management</td>
<td>Formal expertise</td>
<td>Allocate formal engineer resources</td>
<td>Plan extra compute, vendor resources</td>
</tr>
<tr>
<td></td>
<td>Schedule &amp; milestones</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 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

<table>
<thead>
<tr>
<th>Task</th>
<th>Planning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Management</td>
<td>Formal expertise Schedule &amp; milestones</td>
</tr>
<tr>
<td>Block</td>
<td>Identify and Evaluate</td>
</tr>
<tr>
<td>Function</td>
<td>Describe and Prioritize</td>
</tr>
<tr>
<td>Complexity</td>
<td>Decompose and Map</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Task</th>
<th>Planning</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Management</td>
<td>Formal expertise</td>
<td>Allocate formal engineer resources</td>
</tr>
<tr>
<td></td>
<td>Schedule &amp; milestones</td>
<td></td>
</tr>
<tr>
<td>Block</td>
<td>Identify and Evaluate</td>
<td>Capture Interfaces</td>
</tr>
<tr>
<td>Function</td>
<td>Describe and Prioritize</td>
<td>End-to-End Checkers</td>
</tr>
<tr>
<td>Complexity</td>
<td>Decompose and Map</td>
<td>Abstraction Techniques</td>
</tr>
</tbody>
</table>
# End-to-End Checking Methodology

<table>
<thead>
<tr>
<th>Task</th>
<th>Planning</th>
<th>Implementation</th>
<th>Closure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Management</td>
<td>Formal expertise Schedule &amp; milestones</td>
<td>Allocate formal engineer resources</td>
<td>Plan extra compute, vendor resources</td>
</tr>
<tr>
<td>Block</td>
<td>Identify and Evaluate</td>
<td>Capture Interfaces</td>
<td>Validate Constraints</td>
</tr>
<tr>
<td>Function</td>
<td>Describe and Prioritize</td>
<td>End-to-End Checkers</td>
<td>Conclusiveness</td>
</tr>
<tr>
<td>Complexity</td>
<td>Decompose and Map</td>
<td>Abstraction Techniques</td>
<td>Formal Coverage</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Abstraction Technique</th>
<th>Design Complexity</th>
<th>Formal Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Case splitting</strong></td>
<td>Multiple runs with different cases reducing design complexity per run/case</td>
<td>Reduce COI, reduce state space per run/case</td>
</tr>
<tr>
<td><strong>Cut-point/ Black box</strong></td>
<td>Eliminate logic driving cut-points/inside blackbox</td>
<td>Reduce COI, state space; controlled with constraints</td>
</tr>
</tbody>
</table>
## Abstraction Techniques

<table>
<thead>
<tr>
<th>Abstraction Technique</th>
<th>Design Complexity</th>
<th>Formal Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case splitting</td>
<td>Multiple runs with different cases reducing design complexity per run/case</td>
<td>Reduce COI, reduce state space per run/case</td>
</tr>
<tr>
<td>Cut-point/ Black box</td>
<td>Eliminate logic driving cut-points/inside blackbox</td>
<td>Increase flexibility but controlled with constraints</td>
</tr>
<tr>
<td>Reset abstraction</td>
<td>n.a.</td>
<td>Reduce access depth</td>
</tr>
<tr>
<td>Counter abstraction</td>
<td>n.a.</td>
<td>Reduce the length of counting</td>
</tr>
</tbody>
</table>
## Abstraction Modeling 1

<table>
<thead>
<tr>
<th>Abstraction Model</th>
<th>Design Complexity</th>
<th>Formal Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symmetric data elements [7]</td>
<td>Eliminate multiple dimensional data elements; add single dimension abstraction model</td>
<td>Reduce COI and state space with symmetry</td>
</tr>
</tbody>
</table>

© Accellera Systems Initiative
## Abstraction Modeling 1

<table>
<thead>
<tr>
<th>Abstraction Model</th>
<th>Design Complexity</th>
<th>Formal Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symmetric data elements [7]</td>
<td>Eliminate multiple dimensional data elements; add single dimension abstraction model</td>
<td>Reduce COI and state space with symmetry</td>
</tr>
</tbody>
</table>

### RTL model

```c
element_type [SIZE-1:0] element;

element [addr] = wr_data;
rd_data = element [addr];
```

### Abstraction model

```c
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

<table>
<thead>
<tr>
<th>Abstraction Model</th>
<th>Design Complexity</th>
<th>Formal Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory abstraction [7]</td>
<td>Represent one location instead of the full size of the memory</td>
<td>Reduce COI and state space with symmetry</td>
</tr>
</tbody>
</table>

### 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

<table>
<thead>
<tr>
<th>Abstraction Model</th>
<th>Design Complexity</th>
<th>Formal Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIFO [7]</td>
<td>Eliminate logic before cut-points; add abstraction model</td>
<td>Reduce the depth of the FIFO</td>
</tr>
</tbody>
</table>

```verilog
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

<table>
<thead>
<tr>
<th>Abstraction Model</th>
<th>Design Complexity</th>
<th>Formal Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data independence (Wolper Coloring) [6]</td>
<td>Eliminate all storage elements; add Wolper FSMs</td>
<td>Reduce COI with pattern</td>
</tr>
</tbody>
</table>

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

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

© 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
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

© Accellera Systems Initiative
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
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

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

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
## GPU Level 2 Cache Request Coalescer (LRC) unit

<table>
<thead>
<tr>
<th>Task</th>
<th>Planning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Management</td>
<td>Formal expert (9+ yr) (20% time)</td>
</tr>
<tr>
<td>Block</td>
<td>Divide and conquer: Submodules: Req, Rsp</td>
</tr>
<tr>
<td>Function</td>
<td>Prioritize: All IP block, all checks are important</td>
</tr>
<tr>
<td>Complexity</td>
<td>Decompose: ILC (submodule) blackbox Design Shrinking (FIFO and CAM) Partition VC path to reduce latency</td>
</tr>
</tbody>
</table>
## GPU Level 2 Cache Request Coalescer (LRC) unit

<table>
<thead>
<tr>
<th>Task</th>
<th>Planning</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Management</td>
<td>Formal expert (9+ yr) (20% time)</td>
<td>Formal engineer (1+ yr) Schedule: 6.5 months</td>
</tr>
<tr>
<td>Block</td>
<td>Divide and conquer: Submodules: Req, Rsp</td>
<td>Capture Interfaces: Xbar Interface, L2 interface</td>
</tr>
<tr>
<td>Function</td>
<td>Prioritize: All IP block, all checks are important</td>
<td>End-to-End Checkers: Request coalescing, Data integrity, Response replay, Forward progress</td>
</tr>
<tr>
<td>Complexity</td>
<td>Decompose: ILC (submodule) blackbox Design Shrinking (FIFO and CAM) Partition VC path to reduce latency</td>
<td>Abstraction Techniques: Counter abstraction, Wolper’s method for data consistency, Symbolic address/CAM ID modeling</td>
</tr>
</tbody>
</table>
## GPU Level 2 Cache Request Coalescer (LRC) unit

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

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

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

<table>
<thead>
<tr>
<th>Task</th>
<th>Planning</th>
<th>Implementation</th>
<th>Closure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Management</td>
<td>Formal expertise</td>
<td>Allocate formal engineer resources</td>
<td>Plan extra compute, vendor resources</td>
</tr>
<tr>
<td></td>
<td>Schedule &amp; milestones</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Block</td>
<td>Identify and Evaluate</td>
<td>Capture Interfaces</td>
<td>Validate Constraints</td>
</tr>
<tr>
<td>Function</td>
<td>Describe and Prioritize</td>
<td>End-to-End Checkers</td>
<td>Conclusiveness</td>
</tr>
<tr>
<td>Complexity</td>
<td>Decompose and Map</td>
<td>Abstraction Techniques</td>
<td>Formal Coverage</td>
</tr>
</tbody>
</table>

Inconclusives

Diagram: Path through the methodology with arrows indicating the flow between tasks.
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.