Hierarchical Formal Verification and Progress Checking of Network-On-Chip Design

Pritam Roy[1], Ping Yeung[1], Joon Hong[1], Abhishek Desai[1], Aishwarya Raj[1],
Chirag Agarwal[2], Dhruvin Patel[2]
Nvidia Corp., [1]Santa Clara, USA, [2]Gurugram, India

Abstract - The adoption of formal verification has increased steadily thanks to the widespread adoption of formal applications, assertion-based verification, and end-to-end formal model checking. However, to sign off a design with formal model checking, we must advance formal verification beyond focusing on a handful of localized functionalities and thoroughly verify the behavior of the whole design with all the units within it. We need a comprehensive approach to conquer the complexity of a Network-on-chip (NOC) design. This paper presents a hierarchical formal verification methodology using a divide-and-conquer approach with abstraction at multiple levels. We will share our usage of abstract models, formal results, and the formal signoff process.

I. INTRODUCTION

Many companies have used formal verification to verify complex SOCs [1] and safety-critical designs [2]. It has also been used for assurance [2], bug hunting [3], and coverage closure. This paper presents the challenges of verifying a Network-on-chip (NOC) design and the solutions. The NOC layer consists of multiple switches arranged in an N x M mesh network pattern. Its primary purpose is efficiently transferring data and controlling signals between various subsystems, such as the cache sub-system, the memory sub-system, the CPU cores, and other network nodes. Each packet in the network has a source and destination, identified by X and Y coordinates for the mesh nodes and a port identifier for each type of node.

To ensure smooth data transfer, designers must prevent packets from getting lost due to starvation and deadlock. High-priority packets, like those between the CPU cores and the cache sub-system, travel through high-speed networks, while lower-priority packets use low-speed networks. The request, response, and data follow separate virtual channels (VCs).

A. Challenges

Due to the design's complexity and the capacity of formal tools, formal verification of deadlock and end-to-end checks is difficult. The complexity of mesh-level networks requires a divide-and-conquer approach, using assumed guarantee techniques to verify lower-level components with assertions and boundary boxes for the RTL.

B. Solution

To successfully replace block-level simulation with formal signoff, we introduce formal verification early in the design cycle, plan for formal-oriented design partitioning, and deploy formal checking with seasoned engineering resources. The end-to-end formal checking methodology [6] was introduced to establish a process for delivering design blocks signed off with formal verification. As project teams build confidence with formal verification of the design blocks, we leverage the design blocks to enable formal verification at a higher level as designers focus more on higher-level designs and tradeoffs. The goal is to verify the interfaces and interactions among design blocks.

Section II discusses the backgrounds of NoC design, the basics of formal verification, and the important properties needed for the signoff of NoC designs. Section III outlines the hierarchical formal verification framework. Sections IV and V discuss the details of the abstractions made for efficient verification of the switch node and router, respectively. Section VI discusses the recipe for the hierarchical formal verification process. Sections VII-VIII provide detailed results and describe a few example bugs found using the framework.

II. BACKGROUNDS

A. *Network-on-chip (NoC)*

Network-on-chip (NoC) is a common structure for moving data within a complex SoC design. NoC can oversee the communication of hundreds of cores and allow several concurrent transactions. Task allocation and scheduling are essential challenges in NoC designs, affecting application performance, latency, and power consumption.

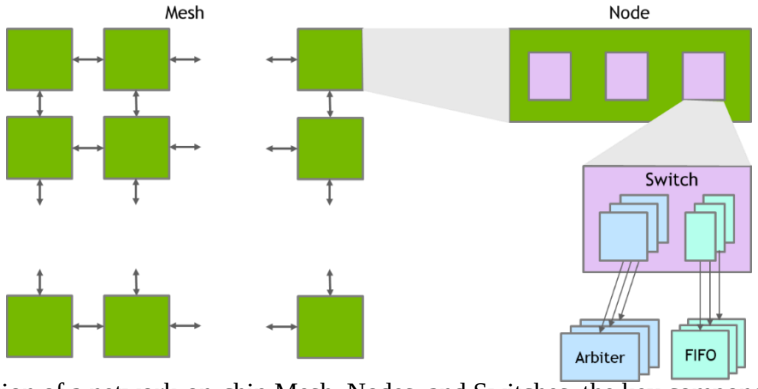

Figure 1: A representation of a network-on-chip Mesh, Nodes, and Switches, the key components of a NoC design.

As depicted in Figure 1, the NoC design consists of a mesh of nodes, each node consisting of several switches, each switch consisting of control and data paths created with multiple arbiters, FIFOs, and RAMs. As the design is well-structured hierarchically, we can leverage the design partitions and parameterized modules to verify the functional units with hierarchical formal verification.

B. *Formal Properties for NoC Designs*

In a network-on-chip (NoC), credits are used to track available buffer/FIFO space. A credit leak might occur if the system fails to reclaim credits after data is processed, leading to the assumption that the buffer is full when it isn't.

Forward progress checks such as starvation and deadlock are critical issues in network-on-chip (NOC) designs and addressing them is essential to ensure efficient and reliable data transfer.

- **Deadlock**: Deadlock occurs when no clients can make forward progress because they are waiting for each other to release resources. This can lead to a complete halt in the system.
- **Starvation**: Starvation occurs when a client cannot make forward progress while other clients can. This can happen due to resource contention, where a high-priority client continuously preempts resources, leaving lower-priority clients starved.

The following end-to-end checkers are also verified in addition to the forward progress checks.

- **In-order Checkers: Ensure packets follow the order from the same source to the same destination, considering bypass paths and floor-sweeping**.
- **Data Consistency**: Maintain data consistency throughout the network.

III. OUTLINE OF HIERARCHICAL FORMAL VERIFICATION

To improve formal efficacy, we work with the design team early in the project to scale down the design from top to bottom. At the top level, we define a representative mesh with much smaller dimensions. At the node level, we reduce the number of ports and the number of credits. At the switch level, we reduce the number of ports, the number of credits, and the burst transfer sizes.

We leverage the NoC structure to verify it hierarchically using the following divide-and-conquer process. Usually, a bottom-up or top-down process can be used. However, a bottom-up or top-down process will introduce too much dependency on the project schedule, leading to constraints and delays. We aim to have multiple teams concurrently verify the units at each of the following levels.

- Formal verification of the Mesh
- Formal verification of the Node
- Formal verification of the Switch

Figure 2 comprehensively lists the properties at the various hierarchy levels—NoC Mesh, Node, Switches, and various control and data path levels.

Level	Design	High-level properties	Implementation
Architectural	NoC Mesh	congestions, performance, forward progress	The super-units in the sub-system are replaced with abstract models to improve formal efficacy.
Super Unit	NoC Node	request-responses, crediting, data transfers, forward progress	The units in the super-unit are replaced with abstract models to improve formal efficacy.
Unit	NoC Switches [1]	crediting, data integrity, fairness, starvation	The original RTL is used to verify the high-level properties. Various abstraction techniques are used to simplify the design for formal efficacy.
Sub-unit	Control and Data paths	functionality	The original RTL is used to verify all the functionalities of the sub-units.

Figure 2: Network-on-chip Mesh, Nodes, and Switches.

To verify a NoC node, the switches within are replaced with abstract models to verify the high-level properties, such as request responses, crediting, data transfers, and forward progress. Then, to verify a switch, several well-understood abstraction techniques are used [1] for formal efficacy.

IV. FORMAL VERIFICATION OF THE NODE

C. Formal test plan

The formal test plan of the NoC node describes the following end-to-end checking:

- Credit checking includes valid credit, max credit, steering, etc. The source and corresponding destination credit & valid signals are matched and checked to ensure no loss of credit & valid transmissions.
- Data integrity checking focuses on valid payload, payload integrity, and data consistency, as well as the steering, ordering, and bypass routing of packets among the nodes in the mesh. Within each node, the end-to-end checks ensure each packet's path from the source to the destination follows the predefined rules.
- Forward progress checking. That is to ensure packets moving from ingress to egress are subjected to deadlock and starvation.

Task	Planning	Implementation	Closure
Block	Divide and conquer: Use abstract models to replace switch units	Capture Interfaces: Cache, memory interfaces Bridge, IO, auxiliary interfaces Switch interfaces	Total 1300+ constraints simulation integrated
Function	Prioritize: Node-to-node interfaces Steering logic Credit flow path Data flow path	End-to-End Checking: Credit checking: Src to dst credit & valid Data integrity checking: packet transfer, parity, latency Forward progress checking	Total 1700+ properties 80+ simulation-resistant issues and bugs
Complexity	Decompose: Design scaling Testbench functionality Helper/cover assertions	Abstraction Techniques: Reset abstractions Symbolic sets for data transfers	Formal Coverage: Functional coverage Assertion COI coverage Required proof depth

Figure 3: Summary of Formal Checking Methodology for the NOC Nodes

D. Hierarchical formal testbench

Several types of nodes have different configurations, interfaces, functionality, and complexity. They are named ASN (Auxiliary Switch Node), BSN (Bridge Switch Node), CSN (Cache Switch Node), and MSN (Memory Switch Node). Although similar, we decided to build dedicated testbenches for them as the interfaces differ. It enables us to optimize performance later; as the design changes, it also turns out to be less work to update them individually. The formal testbenches are configured to manage the various nodes with their corresponding functionality and constraints. Each node consists of multiple switches, which are complex elements, especially with many ports and credits. To enable formal to focus on the node's functionality, we have created and replaced the switches with a configurable abstract model.

E. The abstract model of a Switch in a Node

Each switch determines the next direction for a packet (north, east, west, south, or terminal ports) based on the destination ID. Each switch has two main parts:

- Steering logic: Determines the next travel direction for a packet.
- Routing logic: Efficiently routes the packet to the selected destination using switch-based routers.

```
module FV_router_model #(
     parameter NUM_SRC = 3
   , parameter NUM_DST = 1
   , parameter PD_WIDTH = 100
   , parameter integer PER_SRC_CREDIT_MAX[NUM_SRC-1:0] = {4, 4, 4 }
   ,parameter integer PER_DST_CREDIT_MAX[NUM_DST-1:0] = {4, 4, 4 }
) (
     input fv_clk
   , input fv_reset_
   , input [NUM_SRC-1:0] fv_ingress_vld
   , input [NUM_SRC-1:0] fv_ingress_req_adv
   , input [NUM_SRC-1:0][PD_WIDTH-1:0] fv_ingress_pd
   , input [NUM_SRC-1:0] fv_src_credit
   , input [NUM_SRC-1:0][3:0] fv_src2dst
   , input [NUM_DST-1:0] fv_egress_vld
   , input [NUM_DST-1:0] fv_egress_adv
   , input [NUM_DST-1:0][PD_WIDTH-1:0] fv_egress_pd
   , input [NUM_DST-1:0] fv_dst_credit
   , input [NUM_DST-1:0][3:0] fv_dst2src
);
```

Figure 4: The abstract model of a switch

F. Network Router Abstraction

Each switch inside the ASN, BSN, CSN, and MSN nodes is replaced with a well-defined abstract model. The interface contains the following components.

- Inputs: Ingress valids, source side credits, source side destination, source advancement, source data.
- Outputs: Egress valids, destination side sources, destination advancement, destination data.
- Constraints/Assumptions: Source side valid credit relation, source side valid vs destination side valid relation, credit valid properties for flow control and resource management.

Figure 4 shows the interface for the abstract model. We can divide the interface input/output and behavioral constraints into 3 parts. 1. Credit-valid handshake behavior (no overflow/underflow in the valid input and credit return) 2. Ingress valid vs egress valid behavior (network requests are getting granted and egressing the correct output based on steering hints src2dst) 3. Data consistency of the router (no duplication/corruption/dropping) in the data path.

```
generate
  for (genvar i=0; i<NUM_SRC; i++) begin: per_src
    for (genvar j=0; j<NUM_DST; j++) begin: per_dst

      wire fv_incoming_valid.
      wire fv_outgoing_valid;

      assign fv_incoming_valid = (fv_src2dst[i] == j) && fv_ingress_vld[i];
      assign fv_outgoing_valid =  fv_egress_vld[j] && (fv_dst2src[j] == i);
      assign fv_token_count_model_nxt = fv_token_count_model+fv_incoming_valid-fv_outgoing_valid;
      `FFER(fv_token_count_model, fv_token_count_model_nxt, fv_clk, 1'b1, fv_reset_, 0)

      asum_FV_router_model_no_outgoing_valid:
         assert property (@(posedge fv_clk)
            fv_outgoing_valid |-> |fv_token_count_model);

      asum_FV_router_model_no_overflow :
         assert property (@(posedge fv_clk)
            fv_token_count_model_nxt <= PER_SRC_CREDIT_MAX[i]);


      asum_FV_router_model_no_underflow :
         assert property (@(posedge fv_clk)
           fv_token_count_model + fv_incoming_valid >= fv_outgoing_valid);

      asum_FV_router_model_fwd_progress_liveness:
         assert property (@(posedge fv_clk)
           (fv_token_count_model != 0) |-> ##[0:$] (fv_outgoing_valid && !fv_incoming_valid));

      data_consitency_vip #(.max_outstanding(PER_SRC_CREDIT_MAX[i]), .id_width(PD_WIDTH))
         asum_FV_router_model_inorder_checker
           (fv_clk, fv_reset_,
               fv_incoming_valid, fv_ingress_pd[i], fv_outgoing_valid, fv_egress_pd[j]);
    end
  end
```

Figure 5: Code snippet for control and data checks for abstract model

G. How to validate the abstract model

In this project, we validated the constraints in the model in two ways.

- The constraints are added in the top-level simulation (not only switch level or mesh level). The assumptions in the simulation are treated as assertions and evaluated with existing dynamic simulation test suites/regressions.
- Also, we use lower-level and top-level testbenches (with original RTL) where the formal tools apply the properties as assertions. (in their default format, i.e., assertion)

These two methods are beneficial and help us correct the constraints. Initially, there were a few failures due to wrong constraints; we fixed the constraints. We found various issues in both processes. Simulation-based processes found falsifications in the first few iterations but could not prove them. The formal process could prove them, but it could be bounded as the model's router logic is still present. These two can help in complementary ways.

The beauty of hierarchical assume-guarantee reasoning helps us use the constraints at the top level. Only in the top-level testbenches, when routers are boundary-boxed, does name-based matching create the assumptions from assertions.

```
assume -from_assert  {^.*asum_FV_router_model.*((:assert)?(:assume:assert)*)?$} -regexp
```

V. FORMAL VERIFICATION OF THE SWITCH

A. Formal test plan

The formal test plan of the switch focuses on verifying design integrity, forward progress, prioritization, deadlock, and starvation. One crucial decision is to partition the design into interfaces, data paths, and control paths early in the design process. It enables the effective deployment of different "design for formal verification" techniques. For the interfaces, we unified a switch interface that can be used for all the switch instances in the design. It is a success as it allows design, simulation, and formal verification to be performed concurrently at different hierarchical levels.

Task	Planning	Implementation	Closure
Block	Divide and conquer: Interfaces, data paths, and control paths	Capture Interfaces: Client inputs/outputs Target inputs/outputs	Validate Constraints: Simulation integrated
Function	Prioritize: Design integrity, forward progress, prioritization, deadlock, and starvation	End-to-End Checkers: Target arbitration Data integrity (Wolper) Forward progress checks	Issues found when stressed under different traffic distribution and arbitration schemes
Complexity	Design scaling: Reduce sizes of storage elements, ports, bursts, and transfer credits.	Abstraction Techniques: Use symmetric elements and symbolic variables on client and target pairs	Formal Coverage: Line: 100% Condition: 100%

Figure 6: Summary of Formal Checking Methodology for parameterized NoC Switches

B. Hierarchical formal testbench

As many switches have different configurations, the formal testbench must be highly parameterized and configurable. We start with a library of unique checkers supporting distinctive features in different switch configurations. Then, the corresponding checkers are deployed based on the features used for the data path and control logic. The checkers are instantiated parametrically in the formal testbenches for the different sub-modules below:

- Interface checkers: source-ingress, data path interface, control logic interface, egress-destination, etc.
- Datapath checkers: data integrity, burst handling, clock gating, several types of storage elements, etc.
- Control logic checkers: prioritization, credit handling, several arbitration schemes, etc.

Finally, the switch's top-level formal testbench will instantiate the interface, data path, and control logic testbench.

C. The abstractions applied in the work.

Firstly, we can scale down the data widths and burst buffer sizes for the data path while reducing the number of ports and transfer credits for the control logic. Since FIFOs and arbiters are common design elements in the switches, well-defined abstraction techniques [1][7], such as symbolic FIFO depth abstraction, memory abstraction, data coloring, etc., are adopted extensively. In addition, most end-to-end checkers are implemented for the symbolic client and target pair. As formal will examine all possible values of the symbolic variable to ensure none of the checkers show failure, all client and target pairs will be verified [1].

D. Deadlock and starvation.

We have efficiently partitioned the switch based on functionality and used case splitting to decompose it into multiple formal runs. Each run focuses on a particular configuration with a few concerned cases, reducing the cone-of-influence (COI) and potential state space. For the control logic, we focus extensively on deadlock and starvation verification. Various forward progress and credit tracking checkers have been developed to ensure the switch is free of deadlock and starvation.

VI. General Strategy

A. *Hierarchical assume-guarantee*

We have established a verification process for a top-down formal verification approach, incorporating hierarchical abstraction and refinement. These steps use hierarchical assume-guarantee reasoning and abstract models to focus verification on complex components.

- Starting from the high-level model, create a formal testbench for the entire design (the NOC model at the system level), including relevant properties such as guaranteed data delivery and forward progress checking (deadlock and starvation freedom).
    - Verify all formal properties (assertions) at this level:
    - If all properties are proven or falsified, verification is complete.
    - if undetermined properties remain, move to the next step.
- Identifying the complex blocks by analyzing the bounded properties at the top level.
    - Blocks causing undetermined results (e.g., routers in the NOC).
    - Modules with high complexity (e.g., fifos, arbiters, and data paths in routers).
- Turing the identified complex blocks into boundary boxes and creating abstract models.
    - Replace their detailed RTL implementations with simpler models or assumptions.
    - Use proven assertions from the lower level to create assumptions for the boundary boxes.
- The abstract model can use a small set of properties/invariants from the boundary-boxed module, e.g., fifos should never overflow, and arbiters should grant requests within bounded cycles.
    - Create assumptions based on validated data flow, latency, or grant behaviors.
    - Validate assumptions at the lower level as assertions or in the full simulation at the top level.
- The abstract model is often written to cover a superset of functionalities than the original RTL, e.g., the arbiter abstract model will grant with any latency.
- Once a block is fully verified, its abstract model can be used at a higher level
- The verification is completed once the properties are proven or falsifications are validated on the original RTL. Otherwise, continue to debug the false failure and refine the abstract model by adding constraints around the behavior of the boundary-boxed module.

B. *Benefits of this approach*

- Hierarchical Refinement:
    - Simplifies complex designs by focusing formal verification on critical blocks.
    - Reduces undetermined properties step by step.
- Assume-Guarantee Efficiency:
    - Proven properties from higher levels reduce complexity for lower-level verification.
    - Ensures logical consistency across abstraction levels.
- Focus on Problematic Blocks:
    - By iteratively targeting bounded properties, effort is concentrated on the most complex or problematic areas.
- Verification of Data Integrity Across Levels
    - Ensure data integrity and functional equivalence at each boundary:
    - Cross-check inputs and outputs between boundary-boxed blocks and original RTL.
    - Validate assumptions by verifying end-to-end integrity.

C. *The abstractions applied*

A few more abstraction strategies[6] are also used in the hierarchical verification process to combat the complexity.

- FIFOs: Symbolic depth between 1 and MAX_DEPTH, stable depth during formal runs, full when size reaches depth, and cannot pop elements if size is 0.
- Credit Reduction: Ensure max_credit is within [1, MAX_CREDITS] for formal verification, finding resource issues like starvation/deadlock around the last available credit soon.
- Memories: Symbolic abstraction for reading/writing a single address location with a minimum 1-cycle delay.
- Memories with Bypass: Data sent to RAM or directly read into output with 0 or more cycle delay.
- RFC (8R/8W) Memories with Bypass: Multiple simultaneous reads/writes, exclusive and exhaustive write address ranges, and bypass saves 1 cycle if read address matches write addresses.

VII. BUG EXAMPLES

A. *Issues with steering logic*

Several issues were found related to steering logic between several types of nodes. Traffic flows smoothly when it passes through the same kind of node, but there are many corner cases when packets flow through various kinds of nodes.

The first issue shown involved the BSN response channel configuration. The BSN includes one response channel input and two response channel outputs. The intended design was to distribute response packets from the input to the two output channels evenly. However, due to an issue in the arbitration, all response packets were consistently directed to only one of the output channels.

The second issue was also observed in the arbitration process, though it was more complex since the arbitration appears to operate fairly under typical conditions.

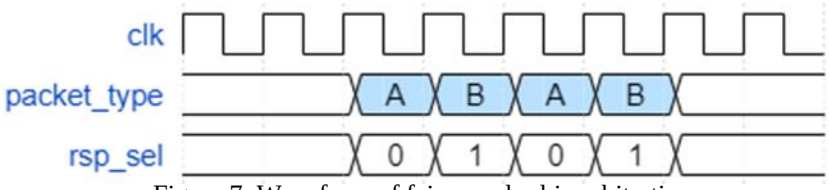

Figure 7: Waveform of fair round robin arbitration

As shown in Figure 7, response channels 0 and 1 are selected evenly when packet types A and B are transmitted from the BSN. However, when an additional packet type, C, is introduced with a target destination different from A and B, the BSN directs all type B packets through response channel 1, leaving response channel 0 unused, as shown in Figure 8.

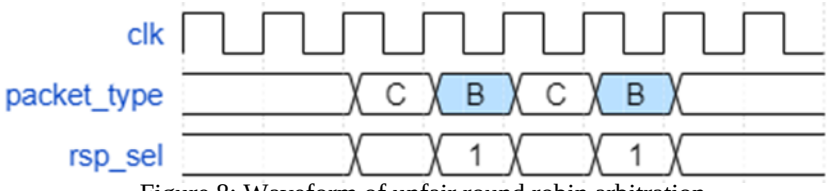

Figure 8: Waveform of unfair round robin arbitration

B. *Connectivity issues*

The NoC consists of many interconnected units. Multiple connectivity issues were found. They can be classified into these kinds: incorrect wire connections, stuck-at-signal connections, data-loss connections, timeout connections, etc. Hence, besides verifying the functionality inside each unit, we can take a higher-level point of view to ensure each packet progresses correctly from connection to connection.

C. *Crediting issues*

The availability of credits governs the flow of packets in the NoC. We must ensure that the credits in each unit are instated, consumed, retained, and returned correctly. Credit tracking and end-to-end checkers were implemented to ensure that credits were not lost and returned to the originating sources correctly. These credit checks were valuable in building confidence and finding some simulation-resisted issues.

VIII. RESULTS

Figure 9 shows the complexity and results for ASN, BSN, CSN, and MBN nodes. Full means full RTL is used for the switch module, BBX means the RTL is not used for the switch module, and AM means the switch module is replaced by abstract models (AM). Figure 10 shows the complexity and results for the switch modules corresponding to the nodes.

A. Formal Netlist Complexity of Node modules

Nodes	Interfaces	Registers	Logics	Constraints	RTL Asserts	Formal Checkers
ASN w/ Switch Full	412	19K	320K	952	1743	502
ASN w/ Switch AM	412	11K	282K	1372	527	732
ASN w/ Switch BBX	412	10K	241K	952	527	502
BSN w/ Switch Full	48	203K	751K	1150	3170	890
BSN w/ Switch AM	48	75K	661K	1360	790	890
BSN w/ Switch BBX	48	27K	419K	1150	790	890
CSN w/ Switch Full	120	69K	1890K	12108	9245	17464
CSN w/ Switch AM	120	42K	1080K	10878	4327	17464
CSN w/ Switch BBX	120	41K	1078K	9898	4327	17244
MSN w/ Switch Full	32	13K	271K	837	1565	590
MSN w/ Switch AM	32	10K	224K	1233	490	746
MSN w/ Switch BBX	32	10K	209K	837	490	590

Figure 9: Summary of formal netlist complexity and properties for the NoC nodes

B. Formal Netlist Complexity of Switch modules

Switches	Interfaces	Counters	Registers	Constraints	Asserts	Covers
ASN Switch dt	238	251	2613	98	1285	2834
ASN Switch rq	203	187	1990	84	1006	2172
ASN Switch rp	226	219	2293	85	1186	2570
BSN Switch dt	253	304	2956	91	1287	2738
BSN Switch rq	253	239	2581	94	1110	2342
BSN Switch rp	371	662	6202	128	2427	5317
CSN Switch fdt	327	541	4950	94	1911	4253
CSN Switch frq	327	523	4949	94	1863	4141
CSN Switch frp	327	523	4949	94	1863	4141
CSN Switch sdt	323	438	4574	103	1711	3789
CSN Switch srq	366	554	5752	117	2083	4666
CSN Switch srp	346	493	5173	108	1895	4218
MSN Switch dt	253	239	2549	94	1110	2342
MSN Switch rq	162	139	1275	60	698	1425
MSN Switch rp	162	139	1414	60	698	1425

Figure 10: Summary of formal netlist complexity and properties for the NoC switches

C. *Checklist for Formal signoff*

The table in Figure 11 summarizes the targeted formal signoff goals. We are close to these targets but will update them when we have the final numbers.

	ASN	BSN	CSN	MSN	
Formal Completed					
Test plan completed	100%	100%	100%	100%	Review completed
Design size reduction	Done	Done	Done	Done	Review completed
Interface constraints implemented	100%	100%	100%	100%	No failure in simulation
Formal checkers implemented	100%	100%	100%	100%	
COI coverage met	90+%	90+%	90+%	90+%	Formal coverage
RTL asserted verified	100%	100%	100%	100%	No violation
Formal checkers verified	100%	100%	100%	100%	No violation
Formal Signoff					
Advanced abstraction techniques	Done	Done	Done	Done	Described in [1][6]
RPD target met	70+%	70+%	70+%	70+%	Required Proof Depth
Forward progress implemented	100%	100%	100%	100%	
Forward progress verified	100%	100%	100%	100%	No violation
Formal coverage met	90+%	90+%	90+%	90+%	Formal coverage

Figure 11: Summary of formal signoff process and requirements

D. *Future work*

Regarding formal coverage, the coverage data is currently measured at each level (mesh, node, switch) independently and is not merged. We would explore that possibility with formal tool vendors in the future.

IX. REFERENCE

[1] Ipshita Tripathi, et al., “Process & Proof for Formal Sign-off - Live Case Study,” Oski, DVCon 2016
[2] Mandar Munishwar, et al., "Architectural Formal Verification of System-Level Deadlocks," Qualcomm, Oski, DVCon 2018.
[3] Syed Suhaib, “Architectural Formal Verification for Coherency,” Nvidia, DVCon 2018 Tutorial
[4] Vaibhav Agrawal, “Ensure Forward Progress with Formal: A Case Study of the Arm Cortex-A77 Instruction Fetch Unit”, ARM, JUG 2019
[5] Saurabh Chaurdia, et al., “Detecting Circular Dependencies in Forward Progress Checkers,” Nvidia, DVCon 2021
[6] Ping Yeung, et al, “Raising the level of Formal Signoff with End-to-End Checking Methodology,” Nvidia,DVCon 2022