2026 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

SANTA CLARA, CA, USA
MARCH 2 - 5, 2026

Scalable Formal Verification Framework for NoC System Address Map Configurations

Anishmon Soosai
Shivaprasad Naranapura Chandrashekara Swamy, Kavin Rajendran , Kranthi Konganti

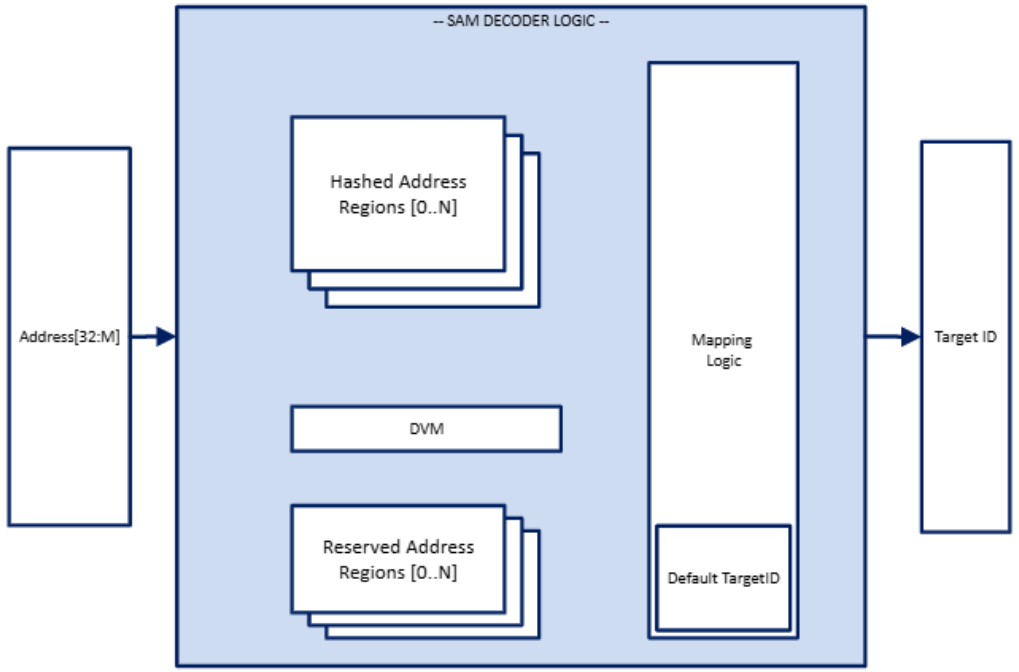
OPENEDGES

Agenda

- Design preview
- Verification scope
- Verification challenges
- Testplan
- Proposed methodology
- Formal Testbench
- Results
- Conclusion

Design preview

- The System Address Map (SAM) decoding logic is the **Target ID generator** within the **Network-on-Chip** (NoC).
- Each requester (Request Node/Home Node) uses the SAM to translate an address into a **Target ID (TgtID)** for routing.
- Look-up and Mapping Table



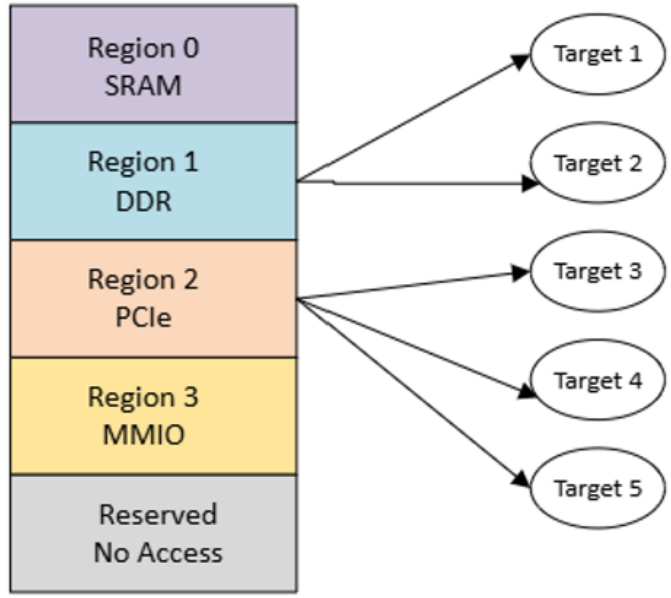
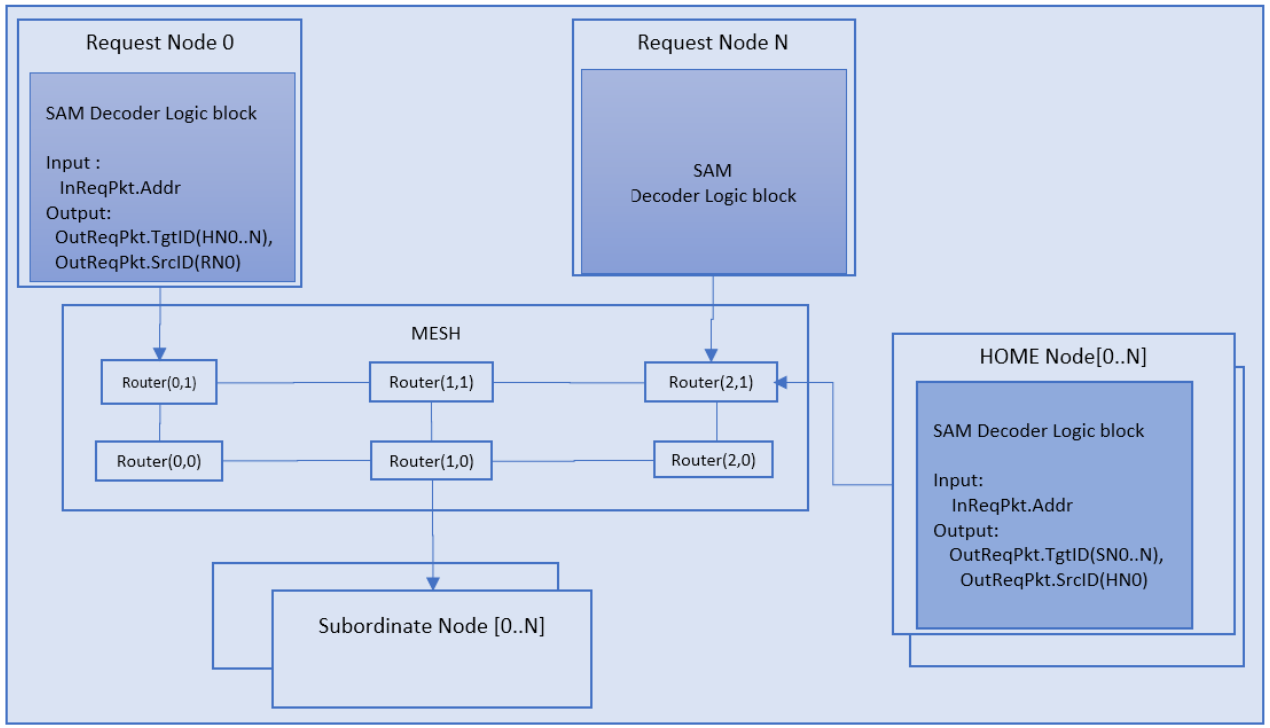


Figure 3. Example Region mapping to multiple targets.

Design preview

- System Address Map(SAM) decoder logic resides in the NoC network layer.
- Paths:
  - Request Node(RNF/RNI) -> Router -> Home Node(HNF)
  - Home Node(HNF) -> Router -> Subordinate Node(SNF/SNI)



Verification Scope

- Highly scalable, configurable design(NOC/SAM)
- No fixed design
- Key multiple configuration combinations specific to SAM
  - Address width variability: 32 → M bits.
  - Scalability of targets/home agents: 2 → Nmax home agents.
  - Hashing / interleaving diversity.
  - Secure vs non-secure routing/permissions.
  - Coherent vs non-coherent requester/target types.
  - Address Map: Valid regions, reserved regions, fragmented ranges, and “holes”

Verification Challenges

- Combinations: Address Width x Home Node x Address Map…etc
- Traditional UVM based Testbench (constrained-random, scoreboards, coverage regressions) struggles to cover the exploding SAM configuration space on schedule.
- Manually adapting UVM testbenches for each SAM variant is not only labor-intensive but also error-prone
- Allowing rare bugs to escape late in the project cycle.

Testplan

Feature	Test plan
Parameters	Perform static assertion checks for parameter values, such as verifying the size of ADDR_WIDTH, NODE_ID_WIDTH…etc
TgtID, NodeID Check	Model the TgtID and Node IDs in the testbench and ensure they match the actual RTL values
Reverse Lookup logic	Ensure that the node IDs and port IDs returned from the home nodes generate the correct reverse IDs for routing packets to the Request node
Boundary Address	Ensure that address regions are properly aligned and do not overlap with other regions.
Error Scenarios	Verify that reserved regions trigger an error flag and return the default home agent ID as TgtID
Hash Function	Verify Hash equations are getting implemented correctly
…..	….
Assumptions	Review all assumptions with the design team and obtain sign-off
Formal Coverage	Review code coverage, dead code, unreachable code, stimuli, checkers and COI…etc

Proposed Methodology (Flow)

- Three Major combinations
  (Formal Property Verification(FPV),
  Python-based automations and database tracking).

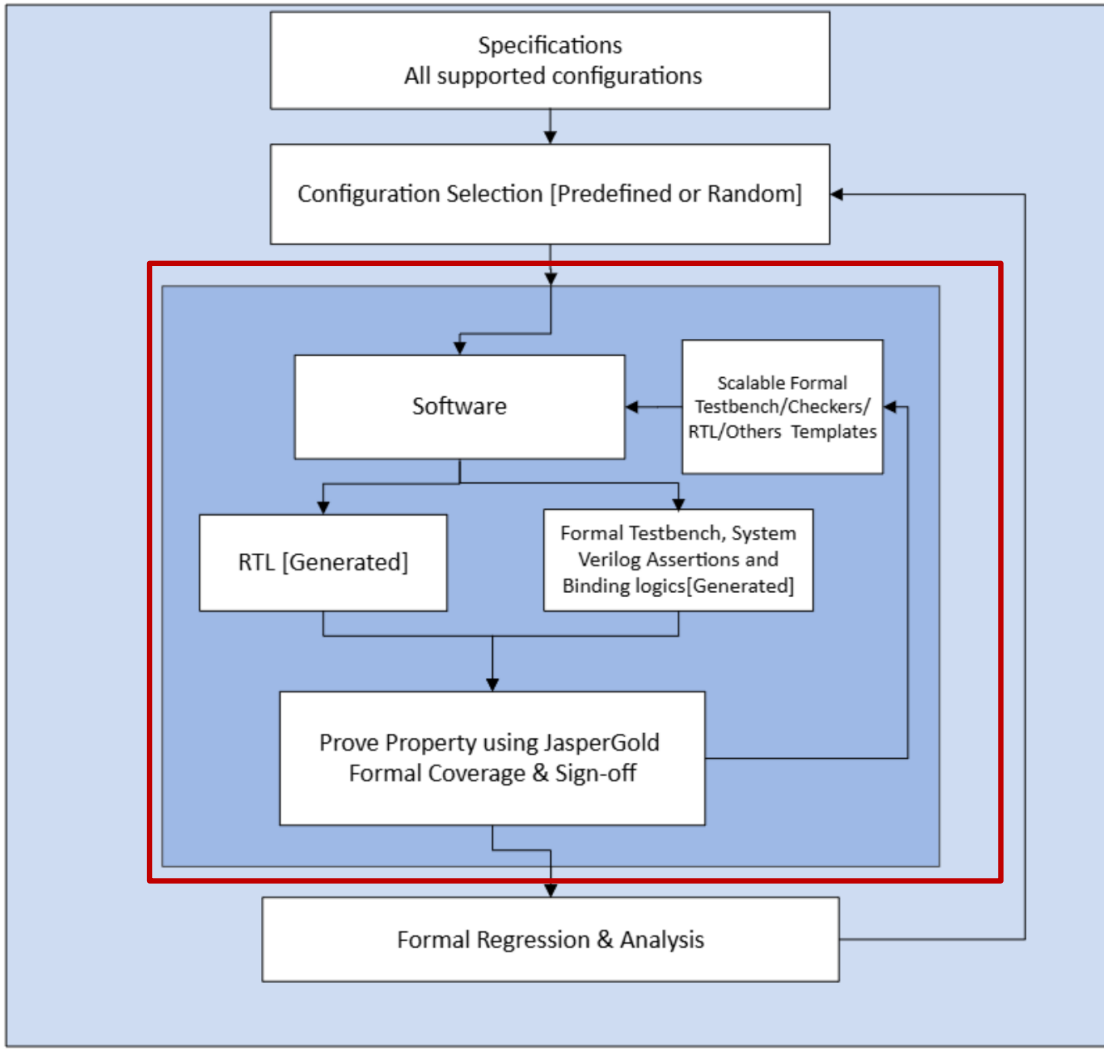
- Specification-aware flow

- Unique configuration selection generator
  (Algorithm: Specification, Milestone, Priority, Ranges, and ignore Previously Validated)

- Single configuration check flow(Inner loop)
  - Configuration-aware Formal Testbench Templates
  - Generated RTL/Tbench used for prove property
  - Failure analysis – Fix RTL/Testbench Templates

- All Targeted configuration check flow(Outer loop)
  - New Unique Configuration
  - Repeat Formal checks & regression
  - Update completion status in the database

- Cronjob & debug fails



Formal Testbench (guidelines)

- Configuration-aware Formal Testbench

- Python APIs are used to compose the FPV
  (example: Address Map, Hashing…etc)

- Parameter Overrides (example : Address Width, ID Width…etc)

- Helper functions

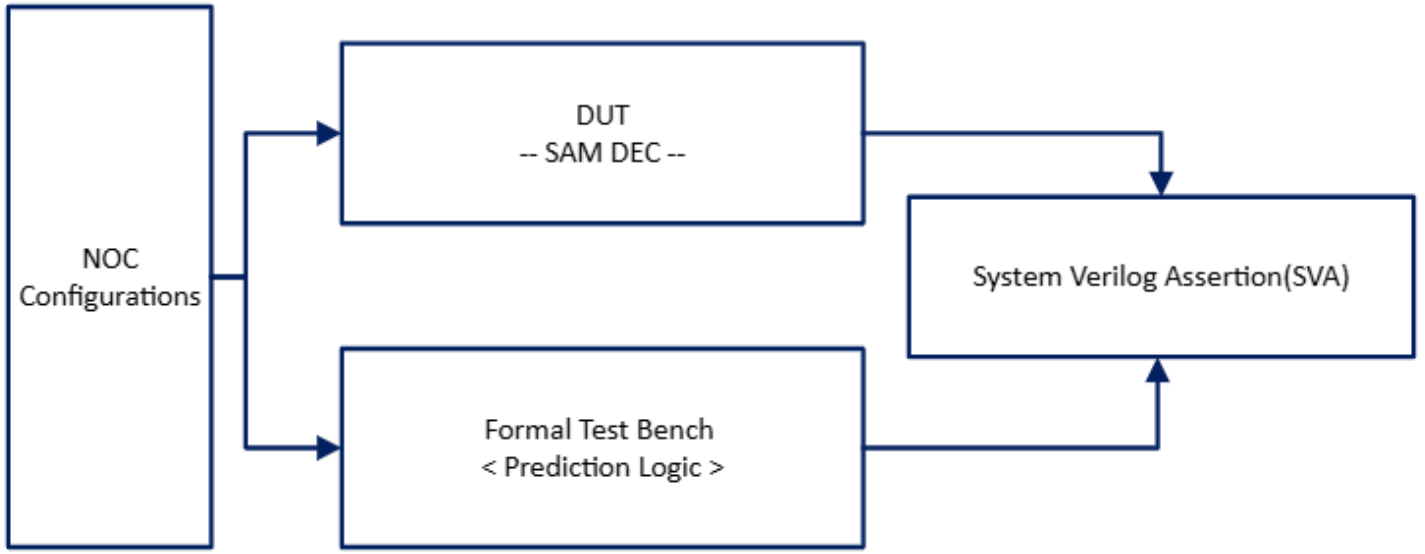
Formal Testbench (example)

- Reference Generated code

```
//---------------- Example Demo Code ------------------------------------------
// Inputs from DUT(Decoding logic)
input         dut_valid;
input  [51:0] dut_addr;
input  [15:0] dut_tgtid;

// Generated Code
bit [1:0]  NUM_SAM_REGIONS                              = 3;
bit [`ADDR_WIDTH-1:0] SAM_START_ADDR [NUM_SAM_REGIONS] = '{REGION0_START_ADDR, REGION1_START_ADDR,REGION2_ST_ADDR};
bit [`ADDR_WIDTH-1:0] SAM_END_ADDR   [NUM_SAM_REGIONS] = '{REGION0_END_ADDR,   REGION1_END_ADDR,  REGION2_END_ADDR};
bit [`TGTID_WIDTH-1:0] SAM_TGTID     [NUM_SAM_REGIONS] = '{TGTID0,TGTID1,TGTID2};

//------------------- Common functions -------------------------------
// Example : To Generate TgtID
function bit [`TGTID_WIDTH-1:0]  TbTgtIDGen(input bit [`ADDR_WIDTH-1:0] addr);
  for (int i = 0; i < NUM_SAM_REGIONS; i++) begin
   if (addr inside {[SAM_START_ADDR[i] : SAM_END_ADDR[i]]})
    return SAM_TGTID[i];
  end
  return 'habcd; //  InValid Address region
endfunction

//---------------------  Checkers ---------------------------------------
//SVA_01: TgtID compare/Check of DUT Vs Tbench
property sam_tgtid_compare_p;
  @(posedge clk) disable iff (!reset_n)
  dut_valid |-> (dut_tgtid == TbTgtIDGen(dut_addr));
endproperty
SAM_TGTID_VALUE_CHK : assert property (sam_tgtid_compare_p)

//SVA_02: Flag the fail if input dut_addr doesn't find any match
<< ..... >>

//SVA_03: Hash function check
<< ..... >>

//SVA_04: Error flag check
<< ..... >>

//SVA_05: Static/Configuration Checks
<< ..... >>
```

Formal Testbench (Integration)

- Black-box all components except SAM Decoder block
- Binding at RTL/SAM Logic boundary
- Formal Testbench/Environment Block diagram



RESULTS

- RTL Findings:
  - Boundary decoding issues, overlap related bugs.
  - Decoding errors for port_id and node_id were found across different configurations.
  - Dead code and Unreachable code analysis helped to cleanup redundant code and for adding relevant formal checks.
- SW Findings:
  - Achieved rapid turnaround and early validation of NoC design generations, enabling faster detection and resolution of potential issues.

CONCLUSION

- Fully automated, scalable verification solution
- Identified RTL and software issues early
- Validated multiple corner-case topologies early in the project cycle
- Increased overall confidence in design correctness

Thank you!

Questions ?