Designing a topology DSL for NoC synthesis

2026-03-14

Network-on-chip topologies are usually hand-written RTL or vendor GUI exports. Both make early exploration and global validation difficult.

The problem

Mesh topology in SystemVerilog tangles instantiation with routing and timing:

```
genvar i, j;
generate
  for (i = 0; i < ROWS; i++) begin
    for (j = 0; j < COLS; j++) begin
      router #(.ID(i*COLS+j)) r (.clk, .rst);
    end
  end
endgenerate
```

A declarative DSL lets the compiler own the lowering. The designer specifies *what*, the topology, routing function, and timing constraints, and the compiler generates the structural RTL.

Spec surface (sketch)

A minimal topology DSL needs to express:

- **Topology family**, mesh, torus, tree, butterfly, custom graph
- **Link parameters**, width, clock domain, flow control
- **Routing function**, deterministic or adaptive, with deadlock constraints
- **QoS requirements**, latency bounds, bandwidth guarantees

Compiler outputs: RTL structure, SDC timing constraints, deadlock freedom proof.

Open questions

- Minimal calculus for topology + routing that remains synthesizable?
- Static verification of deadlock freedom at spec time, not post-layout?

- How much of the routing function can be verified by type?

This connects to the Hermes accelerator generator project, both share the idea of hardware as a compiler target.