



SHAPING THE NEXT GENERATION OF ELECTRONICS

JUNE 23-27, 2024

MOSCONE WEST CENTER
SAN FRANCISCO, CA, USA



JUNE 23-27, 2024

MOSCONE WEST CENTER
SAN FRANCISCO, CA, USA

Verification Challenges of NoCs in complex SoCs

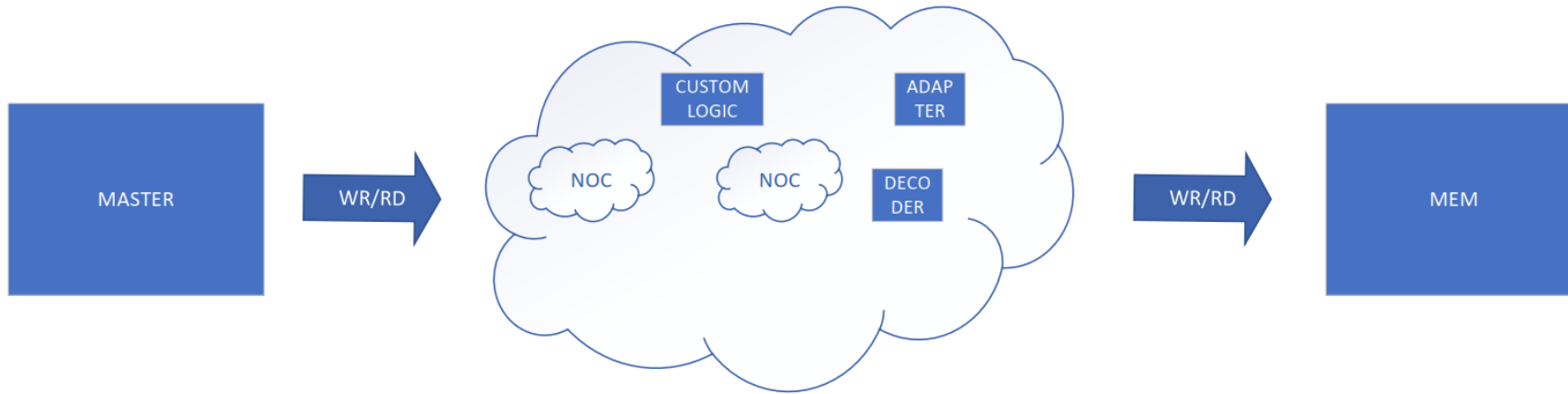
Andrea Majstorovic



Challenges

- Design complexity- many large NoC blocks with lots of features to cover
- Outside the NOC- integration
- Performance- latency, throughput
- Simulation performance
- Multiple protocols (data integrity checking)
- Coverage
- Safety features
- All of the above vs time

Make it a blackbox

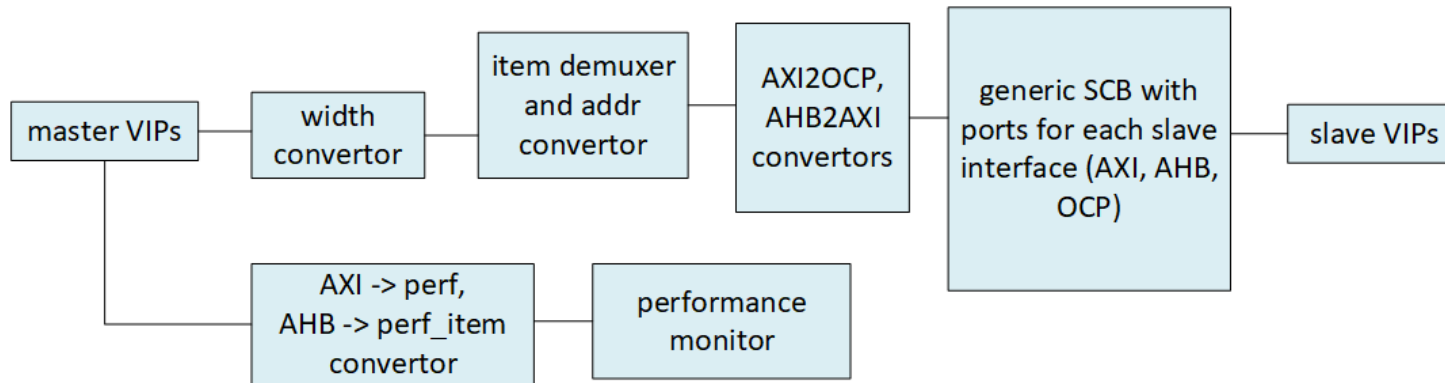


Make it a blackbox

- Provide generic sequences
- Provide generic tests (memory, register, etc.) and integration guides
- Provide generic components (latency checkers, performance monitors)
- Advantage:
 - written only once and used many times, taken from project to project
 - NOC is a kind of a blackbox in other tests, but still tested in the background
 - users don't need to dig into the NOC to implement some tests (registers, memory access, etc.)
 - adding new protocol, testing new feature is very fast
 - time to level 0 passing for all clusters is much faster
 - cluster verification owner knows some limitations better than NoC owner
- Disadvantage:
 - debug is centralized
 - mistake in a generic flow is a chip-wide bug (and you will make those)

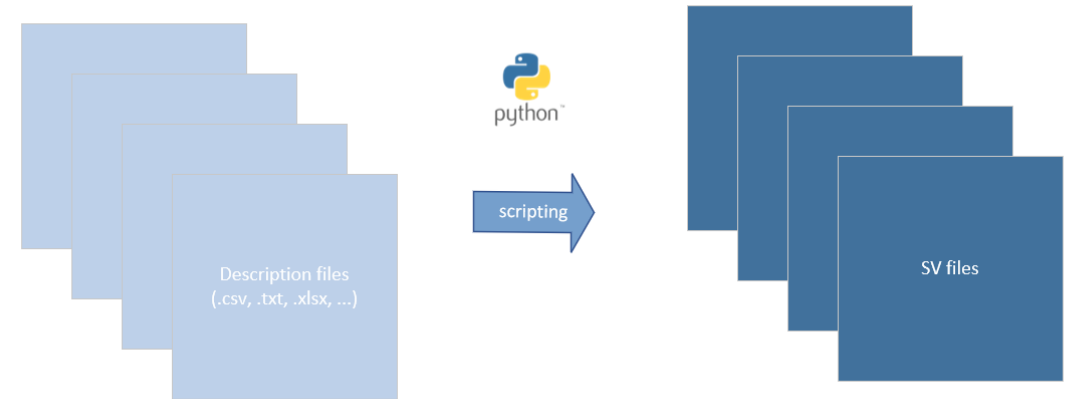
Complexity

- Many master/slave nodes, thousands of paths
- Multiple protocols – between the NoCs, from masters/ from slaves
- Reference model? Needs to be as generic as possible + relatively easy to add new protocols to it.
- Big amount of data- performance/memory issues in simulation - emulation?



Complexity

- system specifications
- design implementation specifications
- utilize Python to generate SV cfg files, test lists, etc.
- use those files for generic tests
- after an RTL change – just review the change, tests are updated automatically
- writing new generic flow test? cfg files are already there



Divide and conquer

- Network consisting of many blocks (typically many “standalone” NoC components + a lot of custom logic)
- Keep verification on 3 levels top wise:
 - Block/cluster level (to stress test each component, maximum coverage)
 - “NOC” level (to test the network, medium coverage, can be quick and dirty, ready before top level is stable)
 - Top/chip level (to test important functional scenarios, smallest coverage, ready only when chip level is stable)

Divide and conquer

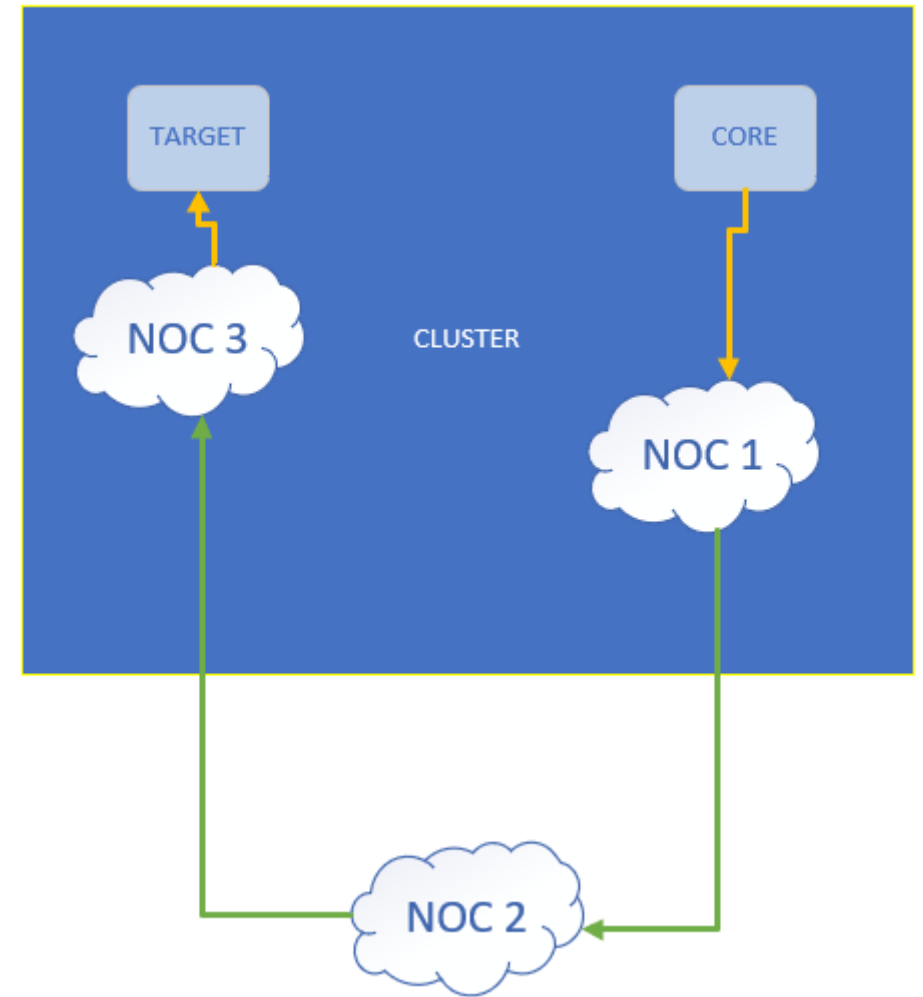
- Priority, priority, priority
- P1, P2, P3, ... , Pmaybe, Pnever
- P1: needs to be working, we will change architecture & design if not working
- P3: should be working, we will defer changes to the next project
- Debug paths are not Pnever, important for lab testing!

Outside the NoC- integration

- NoC verification can't stop on the NoC boundary
- Provide an “easy” and generic way to test inside the cluster
- Memory tests, register tests – run longer versions on smaller designs, medium on NOC level, smaller on chip level
- Path we need to cover
 - from IP (for example: CPU) boundary to the memory
 - from IP boundary to another IP boundary / chip top
- NoC to NoC connectivity – level 0 can be done by formal
- All logic between the NoCs- needs to be verified in “NOC” level

Cluster level simulation

- provide as much tools as possible
- define methodology
- allocate time for debug and support
- provide some quick and dirty workarounds
- analyze coverage, stress the NoC
- integrate full generic refmodel in all tests



Performance testing

- Latency- enable by default, in all tests + have designated tests
- Buffer sizes can't be changed towards the end of the project, verification needs to be done before final backend decisions
- Identifying main traffic patterns and assigning priorities
- Starting small – long chip level end-to-end test is hard to debug
 - run scenarios on cluster/block level
 - include few NoCs and increase as tests are more stable
 - replace complicated (run time consuming) slaves with agents
 - build the test from cluster level to multiple clusters to the top

Coverage

- Very hard to hit all protocol coverage requirements on chip level
- Cluster level: goal is to hit all
- Chip level: goal is to hit special and most important items

Safety features

- Verification of FuSa features
- End-to-end protection
- Parity, ECC
- Firewalls
- Timeout / latency interrupts
- Error scenarios – unmapped address space, error responses
- IP NoC- still need to test all error scenarios !

Change requests

- Constant and always with small deadlines
- Verification is the last test- needs to be as fast as possible
- Interface changes
- Removing masters, slaves
- Merging APRs – moving APR inside the other (verification effort 4 weeks, NoC verification effort 4 days, with 5 bugs found)
- Project -> project : around 50% of paths includes a change (minor or major)

IP from Vendor or in –house

- IP is good news?
- Yes for:
 - passing level 0
 - protocol correctness
 - time to fix bugs
- No for:
 - debug
 - identifying and defining flows
 - understanding the design

Our lessons learned

The assumption that the NOC is IP that not required deep verification **proved as mistake**.

The NoC has thousands paths (from master to slave),

There are also thousands of paths combination

- There are paths that can't work parallel
- There are paths that should work parallel but not with full BW
- There are paths that should work parallel with full BW

The Verification Owner worked very hard in time pressure

Questions?