Noxim User Guide

2026-04-22

Noxim User Guide

1. Introduction and Installation

Noxim is a cycle-accurate Network-on-Chip simulator built on top of SystemC. The current tree supports:

- 2D mesh topologies
- Delta topologies: `BUTTERFLY`, `BASELINE`, and `OMEGA`
- Optional wireless extensions through radio hubs, token-based MAC policies, and a TLM-modeled wireless channel
- Performance and energy reporting
- YAML-based configuration with command-line overrides

The simulator binary is `bin/noxim`. The normal post-clone flow from the repository root is:

```
./build.sh
```

This command:

1. Populates the local third-party dependencies under `bin/libs`
2. Builds `bin/noxim`

If you only need the dependency step, run:

```
./other/setup/fix-dependencies.sh
```

Optional helper tools in `other/` can be built with:

```
make -C other
```

That produces tools such as `noxim_explorer`, `apsra2noxim`, and traffic-table helpers.

Main runtime inputs

Noxim typically consumes two YAML files:

- A simulation/configuration file, selected with `-config FILE`

- A power model file, selected with `-power FILE`

The default example configuration is:

`config_examples/default_config.yaml`

The default power file used by the existing build flow is:

`bin/power.yaml`

Reproducible verification

The repository also includes a deterministic regression suite:

`./regression.sh`

The guide for that suite is in `other/regression/README.txt`.

2. Basic Usage and Output Produced

Basic command

From the repository root:

```
./build.sh
cd bin
./noxim -config ../config_examples/default_config.yaml
```

Or directly from the root, using the built binary:

```
bin/noxim -config config_examples/default_config.yaml -power bin/power.yaml
```

A short deterministic example

```
cd bin
./noxim -config ../config_examples/default_config.yaml -seed 0 -sim 20 -warmup 0
```

Typical output:

```
Loading configuration from file "../config_examples/default_config.yaml"... Done
Loading power configurations from file "power.yaml"... Done
Reset for 1000 cycles...  done!
 Now running for 20 cycles...
Noxim simulation completed. (1020 cycles executed)

% Total received packets: 1
% Total received flits: 5
% Received/Ideal flits Ratio: 0.195312
% Average wireless utilization: 0
% Global average delay (cycles): 10
% Max delay (cycles): 10
% Network throughput (flits/cycle): 0.25
```

```
% Average IP throughput (flits/cycle/IP): 0.015625
% Total energy (J): 4.05561e-09
%   Dynamic energy (J): 1.05112e-10
%   Static energy (J): 3.9505e-09
```

Meaning of the summary lines

- `Total received packets`: total number of packets fully received during the measured interval
- `Total received flits`: total number of flits received during the measured interval
- `Received/Ideal flits Ratio`: received flits divided by the ideal injected flits implied by `packet_injection_rate`, packet size, measured cycles, and node count
- `Average wireless utilization`: ratio of wireless-received packets to total received packets
- `Global average delay (cycles)`: average packet delay, in cycles
- `Max delay (cycles)`: worst observed packet delay
- `Network throughput (flits/cycle)`: aggregated throughput over the whole NoC
- `Average IP throughput (flits/cycle/IP)`: average throughput per endpoint
- `Total energy (J)`: total estimated energy
- `Dynamic energy (J)`: activity-dependent component
- `Static energy (J)`: leakage/static component

Other runtime outputs

- `-verbose 1`, `2`, or `3` prints configuration details and affects some legacy flit-formatting paths
- `-loglevel LEVEL` enables runtime diagnostics, with optional `-logfile FILE` and `-logcomp comp1,comp2`
- `-stats_format text|csv|json -stats_file FILE` writes an additional end-of-run summary file without changing the normal console summary
- `-trace NAME -trace_scope SCOPE` generates `NAME.vcd` with selectable trace coverage
- `./visualNoxim ...` runs a traced mesh simulation and converts it into a local HTML step-by-step viewer
- `-detailed` adds per-communication statistics
- `-show_buf_stats` prints buffer occupancy statistics
- `-asciimonitor` enables an experimental textual network monitor
- `SIGQUIT` prints current statistics without waiting for the run to end

Typical debugging examples:

```
./noxim -config ../config_examples/default_config.yaml -loglevel DEBUG -logcomp router
./noxim -config ../config_examples/default_config.yaml -stats_format csv -stats_file results
```

```
./noxim -config ../config_examples/default_config.yaml -trace debug_trace -trace_scope all
./visualNoxim -sim 40 -warmup 0 -seed 0 -pir 0.3 poisson
```

Practical note:

- Runtime logging has low overhead when `log_level` is `OFF`, but `DEBUG` or `TRACE` can slow down runs if many messages are emitted
- VCD tracing can slow simulations significantly, especially with `trace_scope: all`
- `./visualNoxim` is intended for short or medium debug runs, not long performance campaigns
- The standard text summary on `stdout` is intentionally kept stable for tools such as `noxim_explorer`

Configuration precedence

Noxim applies configuration in this order:

1. Built-in defaults
2. YAML configuration file
3. Command-line overrides

For boolean command-line flags such as `-winoc`, `-wirxsleep`, `-detailed`, and `-show_buf_stats`, the CLI only enables the feature. There is no paired CLI flag to force them back to `false`.

3. YAML Configuration Options and Command-Line Mapping

This section describes the supported simulation YAML keys and how they relate to the command line. If a key has no command-line equivalent, the CLI column is marked `none`. If a command-line option affects more than one YAML field, that relationship is called out explicitly.

3.1 Topology and wired-network options

YAML key	Meaning	Typical values	CLI equivalent
`topology`	NoC topology	`MESH`, `BUTTERFLY`, `BASELINE`, `OMEGA`	`-topology TYPE`
`mesh_dim_x`	Mesh width	integer > 1	`-dimx N`
`mesh_dim_y`	Mesh height	integer > 1	`-dimy N`
`n_delta_tiles`	Number of endpoints in delta topologies	power of two	`-dtiles N`
`buffer_depth`	Router input-buffer depth, in flits	integer >= 1	`-buffer N`

YAML key	Meaning	Typical values	CLI equivalent
`flit_size`	Flit width, in bits	positive integer	`-flit N`
`r2r_link_length`	Wired router-to-router link length used by the power model	real, mm	none
`r2h_link_length`	Router-to-hub link length used by the power model	real, mm	none
`n_virtual_channels`	Number of virtual channels	`1..MAX_VIRTUAL_CHANNELS`	`-vcs N`

Important constraints:

- `MESH` uses `mesh_dim_x` and `mesh_dim_y`
- `BUTTERFLY`, `BASELINE`, and `OMEGA` use `n_delta_tiles`
- `n_delta_tiles` must be a power of two
- Delta topologies currently require `routing_algorithm: DELTA`

3.2 Routing and selection

YAML key	Meaning	Typical values	CLI equivalent
`routing_algorithm`	Active routing algorithm	`XY`, `WEST_FIRST`, `NORTH_LAST`, `NEGATIVE_FIRST`, `ODD_EVEN`, `DYAD`, `DELTA`, `TABLE_BASED`	`-routing TYPE ...`
`routing_table_filename`	Routing-table file for `TABLE_BASED`	path	`-routing TABLE_BASED FILE`
`dyad_threshold`	Congestion threshold for `DYAD`	real, commonly `0..1`	`-routing DYAD T`
`selection_strategy`	Output-port selection strategy	`RANDOM`, `BUFFER_LEVEL`, `NOP`	`-sel TYPE`

Notes:

- `-routing XY`, `WEST_FIRST`, `NORTH_LAST`, `NEGATIVE_FIRST`, and `ODD_EVEN` only set `routing_algorithm`

- `-routing DYAD T` sets both `routing_algorithm` and `dyad_threshold`
- `-routing TABLE_BASED FILE` sets both `routing_algorithm` and `routing_table_filename`
- `BUFFER_LEVEL` and `NOP` are restricted to a single virtual channel

3.3 Hub configuration

The `Hubs:` map defines wireless hub topology and hub-local buffering. `defaults` provides fallback values for all hubs, and numeric hub IDs override selected fields.

YAML key	Meaning	CLI equivalent
`Hubs.defaults.rx_radio_channels`	Channels the hub can receive on	none
`Hubs.defaults.tx_radio_channels`	Channels the hub can transmit on	none
`Hubs.defaults.attached_tiles`	Tile IDs attached to the hub	none
`Hubs.defaults.to_tile_buffer_size`	Hub buffer depth toward tiles	`-buffer_tt N` sets this for all hubs
`Hubs.defaults.from_tile_buffer_size`	Hub buffer depth from tiles toward wireless	`-buffer_ft N` sets this for all hubs
`Hubs.defaults.rx_buffer_size`	Wireless RX antenna-buffer depth	`-buffer_antenna N` sets both RX and TX for all hubs
`Hubs.defaults.tx_buffer_size`	Wireless TX antenna-buffer depth	`-buffer_antenna N` sets both RX and TX for all hubs
`Hubs.<hub_id>.*`	Per-hub overrides of the same fields	none

Operational notes:

- `attached_nodes` is the key field that determines which tiles belong to each wireless cluster
- `TRAFFIC_LOCAL` assumes that every tile that may originate local traffic belongs to some hub
- `use_winoc: false` ignores hub/channel wireless activity at runtime, but the configuration still has to parse correctly

3.4 Radio-channel configuration

The `RadioChannels:` map defines wireless channel timing and MAC policy.

YAML key	Meaning	CLI equivalent
`RadioChannels.defaults.data_rate`	Wireless data rate in Gb/s	none
`RadioChannels.defaults.ber`	Bit-error-rate placeholder	none
`RadioChannels.defaults.mac_policy`	Token/MAC policy	none
`RadioChannels.<channel_id>`	Per-channel overrides	none

Supported MAC policies are:

- `TOKEN_PACKET`
- `TOKEN_HOLD, <cycles>`
- `TOKEN_MAX_HOLD, <cycles>`

The current codebase is stable with `TOKEN_PACKET`. The hold-based policies are implemented, but in this repository state they still expose runtime assertions in some workloads, so they are not part of the green regression baseline.

3.5 Simulation-control, logging, stats-export, and trace options

YAML key	Meaning	CLI equivalent
`clock_period_ps`	Clock period in picoseconds	none
`reset_time`	Reset length in cycles	none
`simulation_time`	Main simulation interval in cycles	`-sim N`
`stats_warm_up_time`	Cycles ignored before collecting statistics	`-warmup N`
`detailed`	Enable detailed per-communication stats	`-detailed`
`max_volume_to_be_drained`	Stop condition based on delivered flits	`-volume N`
`show_buffer_stats`	Print buffer occupancy stats	`-show_buf_stats`
`verbose_mode`	Verbosity level	`-verbose 0..3` or `-verbose VERBOSE_*`
`log_level`	Runtime logger level: `OFF`, `ERROR`, `WARN`, `INFO`, `DEBUG`, `TRACE`	`-loglevel LEVEL`
`log_file`	Runtime log file path	`-logfile FILE`
`log_to_stderr`	Also emit runtime logs to `stderr`	none

YAML key	Meaning	CLI equivalent
`log_components`	Runtime log component filter list	`-logcomp a,b,c`
`stats_format`	Optional stats-file format: `text`, `csv`, `json`	`-stats_format FORMAT`
`stats_file`	Optional stats-file path	`-stats_file FILE`
`trace_mode`	Enable VCD tracing	`-trace FILE` also sets `trace_filename`
`trace_filename`	VCD base filename	`-trace FILE`
`trace_scope`	VCD coverage selection: `basic`, `router`, `buffers`, `wireless`, `all`	`-trace_scope SCOPE`
`rnd_generator_seed`	Random seed for deterministic runs	`-seed N`

Notes:

- `clock_period_ps` affects clocking, wireless latency quantization, and the power model
- `verbose_mode` is a legacy verbosity control; the main runtime diagnostics are now driven by `log_level`
- `log_components` currently recognizes component groups such as `router`, `hub`, `channel`, `tokenring`, and `initiator`
- `stats_file` writes an extra export file and does not replace the traditional text summary printed on `stdout`
- `trace_mode` and `trace_filename` are usually controlled together through `-trace FILE`
- `trace_scope: basic` traces clock/reset plus wired handshakes; broader scopes add flits, NoP, buffer-state, and wireless/token-ring signals
- `trace_scope: all` is useful for debugging but can generate large VCD files and noticeably slow simulation
- `rnd_generator_seed` is supported by the loader even though older example YAML files may omit it

3.6 Wireless feature toggles

YAML key	Meaning	CLI equivalent
`use_winoc`	Enable wireless hub/channel behavior	`-winoc`
`winoc_dst_hops`	Butterfly relay threshold for partial wired completion after a wireless hop	`-winoc_dst_hops N`

YAML key	Meaning	CLI equivalent
`use_wirxsleep`	Enable the wireless RX/TX power-manager logic	`-wirxsleep`

Constraints:

- `winoc_dst_hops` is only valid on `BUTTERFLY`
- `winoc_dst_hops` requires `use_winoc: true`
- `use_wirxsleep` currently requires a single virtual channel
- The wireless power-manager implementation assumes `TOKEN_PACKET`

3.7 Traffic-generation and packet-size options

YAML key	Meaning	Typical values	CLI equivalent
`min_packet_size`	Minimum packet size in flits	integer >= 2	`-size Nmin Nmax`
`max_packet_size`	Maximum packet size in flits	integer >= `min_packet_size`	`-size Nmin Nmax`
`packet_injection_rate`	Base injection probability	`0 < R <= 1`	`-pir R TYPE`
`probability_of_retransmission`	Retransmission probability after a previous transmission	`0 < R <= 1`	derived from `-pir` distribution parameters
`locality`	Locality factor used by `TRAFFIC_LOCAL`	`0..1`	`-traffic local L`
`traffic_distribution`	Spatial traffic pattern	see below	`-traffic TYPE ...`
`traffic_table_filename`	File used by `TRAFFIC_TABLE_BASED`	path	`-traffic table FILE`
`traffic_hardcoded_filename`	File used by `TRAFFIC_HARDCODED`	path	`-traffic hardcoded FILE`

Supported traffic distributions:

- `TRAFFIC_RANDOM`
- `TRAFFIC_LOCAL`
- `TRAFFIC_ULOCAL`
- `TRAFFIC_TRANSPOSE1`
- `TRAFFIC_TRANSPOSE2`
- `TRAFFIC_BIT_REVERSAL`

- TRAFFIC_SHUFFLE
- TRAFFIC_BUTTERFLY
- TRAFFIC_TABLE_BASED
- TRAFFIC_HARDCODED

How CLI mapping works:

- `-pir R poisson` sets `packet_injection_rate = R` and `probability_of_retransmission = R`
- `-pir R burst B` sets `packet_injection_rate = R` and derives `probability_of_retransmission = R / (1 - B)`
- `-pir R pareto Aon Aoff r` sets `packet_injection_rate = R` and derives `probability_of_retransmission`
- `-pir R custom X` sets `packet_injection_rate = R` and `probability_of_retransmission = X`
- `-traffic local L` sets `traffic_distribution = TRAFFIC_LOCAL` and `locality = L`
- `-traffic table FILE` sets `traffic_distribution = TRAFFIC_TABLE_BASED` and `traffic_table_filename = FILE`
- `-traffic hardcoded FILE` sets `traffic_distribution = TRAFFIC_HARDCODED` and `traffic_hardcoded_filename = FILE`

Traffic file formats:

- Traffic table file: `src dst [pir [por [t_on t_off t_period]]]`
- Hardcoded traffic file: one `src dst` packet per line, with `-1` marking the end of a cycle

3.8 CLI-only options

These options are supported by the parser but do not have direct YAML fields in the main simulation file:

CLI option	Meaning
`-config FILE`	Select the main YAML configuration file
`-power FILE`	Select the power-model YAML file
`-hs ID P`	Add a hotspot destination with percentage `P`
`-asciimonitor`	Enable the experimental ASCII monitor

There is also one parser-supported option that is currently missing from `-help` output:

CLI option	Meaning
`-dtiles N`	Set `n_delta_tiles` for delta topologies

3.9 Power-model YAML loaded with `-power`

The power file is a separate YAML document, typically `bin/power.yaml`. Its main schema is:

- `Energy.Buffer`: rows of `[depth, flit_size, leakage, push, front, pop]`
- `Energy.LinkBitLine`: rows of `[length_mm, leakage, dynamic]`
- `Energy.Router.crossbar`: rows of `[ports, flit_size, static, dynamic]`
- `Energy.Router.network_interface`: rows of `[flit_size, static, dynamic]`
- `Energy.Router.routing`: per-routing-algorithm `[static, dynamic]`
- `Energy.Router.selection`: per-selection-strategy `[static, dynamic]`
- `Energy.Hub.transceiver_leakage`
- `Energy.Hub.transceiver_biasing`
- `Energy.Hub.rx_dynamic`
- `Energy.Hub.rx_snooping`
- `Energy.Hub.default_tx_energy`
- `Energy.Hub.tx_attenuation_map`: rows of `[tx_hub, rx_hub, attenuation]`

These values do not have per-field CLI overrides. The command-line control is only:

```
-power FILE
```

4. Noxim Internals: SystemC Components and Their Relationships

4.1 Startup flow

The top-level control flow is:

1. `sc_main()` prints the banner and calls `configure()`
2. `configure()` loads the YAML files, applies CLI overrides, and validates the resulting configuration
3. `sc_main()` creates the SystemC clock and reset
4. `sc_main()` instantiates `NoC`
5. `NoC` builds the selected topology and attaches the supporting global tables
6. The simulation runs for `reset_time`, then for `simulation_time`
7. `GlobalStats` is used to print final statistics

4.2 High-level structure

Noxim has two interacting data planes:

- Wired plane: signal-level SystemC modules using alternating-bit handshakes

- Wireless plane: hub logic plus TLM-modeled radio channels

At the highest level:

```
sc_main
  -> NoC
       -> Tile[] / core[]
            -> Router
            -> ProcessingElement
       -> Hub[]
            -> Initiator[]
            -> Target[]
       -> Channel[]
       -> TokenRing
       -> GlobalRoutingTable / GlobalTrafficTable / GlobalTrafficHardcoding
```

4.3 Main SystemC modules

`NoC` Role:

- Top-level SystemC testbench and topology builder

Responsibilities:

- Builds `MESH`, `BUTTERFLY`, `BASELINE`, or `OMEGA`
- Allocates all tiles, hubs, channels, and shared signals
- Loads routing and traffic tables when required
- Connects local wired ports and optional wireless links

Key relationships:

- Owns the tile matrix in mesh mode
- Owns the endpoint/switch structures for delta topologies
- Owns all `Hub`, `Channel`, and `TokenRing` instances

`Tile` Role:

- Composite module pairing one `Router` with one `ProcessingElement`

Responsibilities:

- Binds router ports to neighboring tiles
- Binds the router local port to the local processing element
- Exposes optional hub-facing ports

Key relationships:

- One tile corresponds to one network endpoint in mesh mode
- Contains exactly one `Router` and one `ProcessingElement`

`Router` Role:

- Per-node packet-switching element

Responsibilities:

- Receives flits from directional, local, and hub ports
- Stores incoming flits in per-port, per-VC buffers
- Applies routing and output-port selection
- Uses a reservation table to arbitrate the internal crossbar
- Maintains local statistics and power accounting
- Exchanges neighbor state for `NOP` and `BUFFER_LEVEL`

Interfaces:

- Wired ABP handshake on `flit`, `req`, `ack`, and `buffer_full_status`
- Additional `free_slots` and `NoP_data` ports for selection strategies

`ProcessingElement` Role:

- Traffic source and sink attached to each tile

Responsibilities:

- Generates packets according to the selected traffic model
- Turns packets into flits
- Consumes incoming flits and acknowledges them
- Uses the global traffic table or hardcoded traffic trace when configured

Important behavior:

- Random traffic generation uses the simulator random seed
- `TRAFFIC_LOCAL` and `TRAFFIC_ULOCAL` depend on hub connectivity

`Buffer` Role:

- Generic FIFO for flits

Responsibilities:

- Tracks occupancy, fullness, free slots, and simple buffer statistics
- Used in routers, hubs, antenna buffers, and wireless TX/RX queues

`Hub` Role:

- Bridge between local tile clusters and the wireless channels

Responsibilities:

- Receives flits from locally attached tiles
- Queues them in tile-to-antenna buffers
- Starts wireless transactions through per-channel `Initiator` modules
- Receives wireless traffic through per-channel `Target` modules

- Routes received wireless flits into local tile-facing buffers
- Applies wireless token/MAC behavior and optional power-manager logic

Important relationships:

- One hub has zero or more attached tiles
- One hub may transmit on multiple channels and receive on multiple channels
- Each hub attaches to the shared `TokenRing`

`Channel` Role:

- TLM-modeled wireless medium for one radio channel

Responsibilities:

- Accepts transactions from hub initiators
- Delivers them to the correct destination hub target
- Computes flit transmission latency from `flit_size`, `data_rate`, and `clock_period_ps`
- Accounts for wireless power consumption

Design note:

- The wired NoC is signal-based
- The wireless inter-hub path is transaction-level

`Initiator` Role:

- Per-hub, per-channel wireless transmitter front-end

Responsibilities:

- Pulls flits from the hub transmit buffer
- Builds TLM payloads
- Starts a wireless transaction when the token policy allows it

`Target` Role:

- Per-hub, per-channel wireless receiver front-end

Responsibilities:

- Accepts wireless TLM transactions
- Stores received flits into the hub RX antenna buffer

`TokenRing` Role:

- Shared wireless MAC arbiter

Responsibilities:

- Tracks token ownership for each radio channel

- Implements `TOKEN_PACKET`, `TOKEN_HOLD`, and `TOKEN_MAX_HOLD`
- Exposes token-holder and expiration signals to hubs

`Stats` Role:

- Per-destination-node communication statistics

Responsibilities:

- Tracks delay and throughput for each source-to-destination history
- Owned by each router

`GlobalStats` Role:

- End-of-run aggregation across the entire NoC

Responsibilities:

- Aggregates packet, flit, delay, throughput, wireless, and energy metrics
- Optionally prints buffer statistics and power-manager-related summaries

4.4 Supporting non-SystemC structures

These are not top-level SystemC modules, but they are central to the design:

- `GlobalRoutingTable`: stores routing tables loaded from file
- `LocalRoutingTable`: extracts the per-router view of the global routing table
- `GlobalTrafficTable`: stores table-driven traffic schedules
- `GlobalTrafficHardcoding`: stores cycle-by-cycle packet traces
- `ReservationTable`: tracks temporary ownership of router and hub outputs
- `RoutingAlgorithms` and `SelectionStrategies`: plugin-like registries for routing and selection implementations

4.5 Wired data path

For a normal wired packet:

1. `ProcessingElement` injects a packet
2. The local `Router` receives the flit on its local port
3. The router stores the flit in an input buffer
4. The router computes candidate outputs through the selected routing algorithm
5. The router picks one output with the selected strategy
6. The reservation table arbitrates the crossbar
7. The flit is forwarded to the neighboring router or to the destination PE

4.6 Wireless data path

For a packet that uses wireless:

1. The source tile router sends the flit toward its hub
2. The source `Hub` stores it in `buffer_from_tile`
3. The per-channel `Initiator` launches a TLM transaction if the hub owns the token
4. The `Channel` delivers the transaction to the destination hub `Target`
5. The destination `Target` stores the flit in the antenna RX buffer
6. The destination `Hub` reserves the proper local output port
7. The hub transfers the flit to `buffer_to_tile`
8. The destination tile router receives it from the hub-side port
9. The local PE consumes the packet

4.7 Timing model

- The SystemC clock period is `clock_period_ps`
- `reset_time`, `simulation_time`, and warm-up are expressed in cycles
- Wireless transmission delay is quantized in cycles from `flit_size / data_rate`
- Output metrics are printed in cycles and joules

5. Other Tools: Regression Tests, Trace Viewer, Noxim Explorer, and the Produced Matlab Files

5.1 Regression tests

The deterministic regression harness is:

```
./regression.sh
```

Useful commands:

```
./regression.sh
./regression.sh --list
./regression.sh --case mesh_8x8_buf4
./regression.sh --update
```

Directory structure:

- `other/regression/configs/`: pinned YAML configurations
- `other/regression/cases.txt`: curated regression matrix
- `other/regression/expected/`: committed golden summaries
- `other/regression/traffic/`: deterministic traffic assets
- `other/regression/generated/`: scratch output created at runtime

What the harness does:

1. Builds `bin/noxim` unless `--skip-build` is used
2. Runs the selected cases

3. Normalizes the stable summary section of the simulator output
4. Compares it against committed baselines, or refreshes them with `--update`

5.2 Trace viewer

The easiest way to inspect mesh network state cycle by cycle after a run is:

```
./visualNoxim -sim 40 -warmup 0 -seed 0 -pir 0.3 poisson
```

What it does:

1. Builds `bin/noxim` if needed
2. Runs the simulator with `-trace` and `-trace_scope router,buffers` unless you override them
3. Converts the resulting VCD file into a self-contained HTML file

Current limitation:

- `visualNoxim` supports `MESH` topology only and fails early for non-mesh configs

The generated files go by default under:

- `other/visualizer-output/noxim_trace.vcd`
- `other/visualizer-output/noxim_trace.html`

The HTML viewer is intentionally lightweight:

- Pure Python 3 stdlib on the conversion side
- No external JavaScript libraries
- Works directly from a local `file://` path in a browser

Viewer features:

- previous / next cycle navigation
- cycle slider
- go-to-cycle input
- reset-start and post-reset jump controls
- mesh-grid rendering with directional north/west/east/south placement
- per-tile inspection of directional buffers and in-flight link flits
- selection persists while you move across cycles

You can also convert an existing VCD without rerunning the simulation:

```
python3 other/noxim_trace_viewer.py \
  --vcd other/visualizer-output/noxim_trace.vcd \
  --output other/visualizer-output/noxim_trace.html \
  --config config_examples/default_config.yaml
```

Practical note:

- this tool is built on top of VCD tracing, so its runtime overhead is tied to the selected trace scope

- it is best used for debug runs with limited simulation length

5.3 noxim_explorer

Build it with:

```
make -C other noxim_explorer
```

Basic usage:

```
other/noxim_explorer other/sim.cfg
```

The explorer reads a configuration-space script and sweeps parameter combinations. It repeatedly launches `bin/noxim`, parses the summary metrics, and emits one MATLAB `.m` file per explored non-aggregated configuration.

The new `stats_file` export is optional and separate; `noxim_explorer` still reads the standard text summary printed on `stdout`.

5.4 Explorer configuration-file format

Explorer scripts use bracketed sections:

```
[parameter]
  value1
  value2
[/parameter]
```

Special sections:

- `[default]`: command-line options appended to every simulation
- `[aggregation]`: parameters treated as the x-axis/sweep dimension inside a generated MATLAB file
- `[explorer]`: explorer runtime options such as simulator path, repetition count, and temporary directory

Example, simplified from `other/sim.cfg`:

```
[topology]
   8x8
[/topology]

[routing]
   XY
[/routing]

[pir]
   0.010 0.015 0.001 poisson
[/pir]

[default]
```

```
   -sim 10000
   -warmup 2000
   -size 8 8
   -buffer 4
   -config ../config_examples/default_config.yaml
   -power ../bin/power.yaml
[/default]

[aggregation]
   pir
[/aggregation]

[explorer]
   simulator ../bin/noxim
   repetitions 10
[/explorer]
```

Explorer-specific parameter names:

- `topology`: mesh sizes such as `8x8`, or topology names for non-mesh use
- `dimXY`: explicit mesh dimensions
- `dtiles`: delta-topology endpoint count
- Regular keys like `routing`, `pir`, `buffer`, and `size` map directly to CLI flags

5.5 MATLAB files produced by `noxim_explorer`

For each explored configuration, `noxim_explorer` emits an `.m` file whose name is derived from the selected parameter combination.

Each generated file contains:

- A MATLAB function
- A `data` matrix with one row per repeated run
- Columns for the aggregated parameters plus:
  - `avg_delay`
  - `throughput`
  - `max_delay`
  - `total_energy`
  - `rpackets`
  - `rflits`
- Derived matrices:
  - `data_delay`
  - `data_throughput`
  - `data_maxdelay`
  - `data_totalenergy`
- Plot-generation code
- A saturation-analysis block returning:

- `max_pir`
- `max_throughput`
- `min_delay`

The explorer expects to parse the standard Noxim summary labels exactly, so changes to the console output format can affect it.

5.6 Explorer repetitions and confidence intervals

For each aggregated point, the explorer runs the simulator multiple times, computes the mean, and uses a `ttest`-based confidence interval before generating the summarized MATLAB arrays. The default repetition count is `5`, but `other/sim.cfg` sets it to `10`.

5.7 Related helper tools in `other/`

Besides `noxim_explorer`, the `other/` directory includes tools that are often useful when building workflows around Noxim:

- `noxim_trace_viewer.py`: converts a VCD trace into a local HTML cycle viewer
- `apsra2noxim`: extracts communication and routing tables from APSRA output
- `hotspot_ttable`: builds hotspot-oriented traffic tables
- `distancebased_ttable`: builds traffic tables with short/long-range mixes
- `ttable_distance_calculator`: measures the short/long-range composition of a traffic table
- `ttable_from_hub`: converts hub-to-hub traffic tables into node-to-node traffic tables

Closing Notes

For day-to-day use, the recommended loop is:

1. Build with `./build.sh`
2. Run experiments with a YAML config plus a small number of CLI overrides
3. Use `./visualNoxim` for short cycle-by-cycle mesh debug sessions
4. Use `./regression.sh` to detect unintended functional changes
5. Use `other/noxim_explorer` when you need automated sweeps and MATLAB-ready output

For deterministic runs, always set `rnd_generator_seed` in YAML or pass `-seed N` on the command line.