|  |
| --- |
| NetSpeed Orion NSIP IP Integration Specification  Version: ORION-NSIP-15.09  April 16, 2016 |

NetSpeed Orion NSIP IP Integration Specification

About This Document

This document describes the guidelines for seamless integration of NetSpeed Orion NSIP IP. This includes details of the IP components and instructions on how to integrate them into customer SoC. The registers in NoC RTL can be accessed via the NetSpeed configuration bus which is described in the HTML documentation generated by NocStudio for each user input configuration.

Audience

This document is intended for users of NocStudio:

* NoC Architects
* NoC Designers
* NoC Verification Engineers
* SoC Architects
* SoC Designers
* SoC Verification Engineers

Prerequisite

Before proceeding, you should generally understand:

* Basics of Network on Chip technology
* NetSpeed Streaming Interface Protocol and Bridge Spec

Related Documents

The following documents can be used as a reference to this document.

* NetSpeed NocStudio Orion NSIP User Manual
* NetSpeed Orion NSIP Physical Design Guidelines
* NetSpeed Register Bus Protocol

Customer Support

For technical support about this product, please contact [support@netspeedsystems.com](mailto:support@netspeedsystems.com)

For general information about NetSpeed products refer to: [www.netspeedsystems.com](http://www.netspeedsystems.com)

Contents

[About This Document 2](#_Toc429642464)

[Audience 2](#_Toc429642465)

[Prerequisite 2](#_Toc429642466)

[Related Documents 2](#_Toc429642467)

[Customer Support 3](#_Toc429642468)

[1 NoC IP Overview 8](#_Toc429642469)

[1.1 NoC IP Components 8](#_Toc429642470)

[1.2 Directory Structure 8](#_Toc429642471)

[1.3 Documentation 9](#_Toc429642472)

[1.4 NocStudio Flow to Generate NoC IP 10](#_Toc429642473)

[1.4.1 Generating RTL from NocStudio 11](#_Toc429642474)

[1.4.2 Sanity Testbench 14](#_Toc429642475)

[2 Integration of NoC 16](#_Toc429642476)

[2.1 Integration of NoC RTL 16](#_Toc429642477)

[2.1.1 Reset 16](#_Toc429642478)

[2.1.2 Clock 17](#_Toc429642479)

[2.1.3 Clock Gating 17](#_Toc429642480)

[2.1.4 Register Bus 19](#_Toc429642481)

[2.1.5 Interrupts 19](#_Toc429642482)

[2.1.6 Event Counters 20](#_Toc429642483)

[2.2 Integration of NoC Verification Checkers 21](#_Toc429642484)

[2.3 Integration of multiple NoCs 22](#_Toc429642485)

[2.4 Synthesis 25](#_Toc429642486)

[2.5 C++ Performance Model 25](#_Toc429642487)

[2.6 Supported Tools 25](#_Toc429642488)

[3 NoC Verification Components 26](#_Toc429642489)

[3.1 Overview of Checkers 26](#_Toc429642490)

[3.2 Environment Setup for Integration 26](#_Toc429642491)

[3.3 Usage Modes 27](#_Toc429642492)

[3.4 Checkers 28](#_Toc429642493)

[3.4.1 Terminology 28](#_Toc429642494)

[3.4.2 Streaming Bridge Checkers 28](#_Toc429642495)

[3.4.3 Router Checkers 31](#_Toc429642496)

[3.4.4 Register Bus End-to-End Checker 32](#_Toc429642497)

[3.4.5 Clock Control Signal Checks 35](#_Toc429642498)

[3.4.6 NoC End-to-End Checker 36](#_Toc429642499)

[3.4.7 NoC End-to-End Traffic Logs 36](#_Toc429642500)

List of Figures

[Figure 1 NoC IP generation flow 11](#_Toc429581871)

[Figure 2 NoC checkers binding to RTL 21](#_Toc429581872)

List of Tables

[Table 1 NoC IP directory structure 8](#_Toc429642501)

[Table 2 NoC IP document list 9](#_Toc429642502)

[Table 3 Files generated by NocStudio in project directory 12](#_Toc429642503)

[Table 4: NoC reset signals 16](#_Toc429642504)

[Table 5 NoC clock signals 17](#_Toc429642505)

[Table 6 Regbus layer clock gating signal 18](#_Toc429642506)

[Table 7 NoC interrupt signals 19](#_Toc429642507)

[Table 8 Directory and files with tag\_project\_name 22](#_Toc429642508)

[Table 9 NetSpeed checkers 26](#_Toc429642509)

[Table 10 `define variables for model build 26](#_Toc429642510)

[Table 11 Recommended checker settings 28](#_Toc429642511)

[Table 12 Streaming bridge checks 29](#_Toc429642512)

[Table 13 Fine-grained user-control of streaming bridge checkers 30](#_Toc429642513)

[Table 14 Router checks 31](#_Toc429642514)

[Table 15 Register bus end-to-end checks 32](#_Toc429642515)

[Table 16 Settings to enable register bus end-to-end traffic logs 34](#_Toc429642516)

[Table 17 Register bus end-to-end traffic logs 35](#_Toc429642517)

[Table 18 Clock control signal checks 35](#_Toc429642518)

[Table 19 NoC end-to-end checks 36](#_Toc429642519)

[Table 20 Settings to enable end-to-end traffic logs 37](#_Toc429642520)

[Table 21 Nomenclature of end-to-end traffic logs 37](#_Toc429642521)

# NoC IP Overview

## NoC IP Components

The NoC IP release package contains the following main components:

* NocStudio binary
* NocStudio usage examples
* RTL library
* Verification library
* User manuals and documentation

In addition, NocStudio generates the following for every user-specified system described in a NocStudio command script:

* NoC RTL
* NoC verification checkers
* Sanity testbench for the generated NoC
* C++ performance model
* Comprehensive html specification for the generated NoC

## Directory Structure

Table 1 NoC IP directory structure

|  |  |
| --- | --- |
| **Name** | **Description** |
| NocStudio | NocStudio executable script |
| nocinit.txt | Initial NocStudio script that gets loaded at the NocStudio startup time |
| custom\_header.txt | Custom header content, modifiable by the user, which is inserted in all auto-generated NoC files |
| examples/\*.txt | Example NocStudio Streaming NoC command scripts |
| noc\_doc\_images/\* | Support files for NoC html documentation generation |
| noc\_rtl/\* | NoC RTL library |
| noc\_modifiable\_rtl/\* | NoC RTL modules that can be replaced by customer designs |
| noc\_verif\_ip/\* | NoC verification library |
| noc\_verif\_bench/\* | NoC sanity testbench |
| noc\_verif\_cust/ns\_global\_defines.vh | Customer files for integration – defines |
| synth/\* | NoC synthesis environment. Please refer to NetSpeed Orion NSIP Physical Design Guidelines.pdf for details |
| user\_manual\_files/\* | Files for auto generation of user manual |
| lib/\* | Dynamic libraries |
| model/\* | NocStudio C++ model files |
| scripts/\* | Scripts for sanity bench |

## Documentation

Documents are provided as a separate set. The documents provided are listed in the table below.

Table 2 NoC IP document list

|  |  |
| --- | --- |
| **Name** | **Description** |
| NetSpeed NocStudio Orion NSIP User Manual.pdf | Overview, architecture, usage, examples |
| NetSpeed Orion C C++ NoC Model Spec.pdf | NetSpeed Orion C C++ generated model usage details, API descriptions |
| NetSpeed Orion NSIP NoC IP Integration Spec.pdf | How to use NocStudio-generated IP (this document) |
| NetSpeed Orion NSIP Physical Design Guidelines.pdf | Synthesis, placement recommendations |
| NetSpeed Register Bus Protocol.pdf | Protocol and usage details for the register bus interface |

NocStudio help and generated documents:

|  |  |
| --- | --- |
| **Name** | **Description** |
| NocStudio Command reference | 1. Available from NocStudio toolbar help 2. Generated HTML doc |
| Noc\_reference\_manual.html | Per project reference manual containing NoC project architecture details, registers, etc |

## NocStudio Flow to Generate NoC IP

Figure 1 describes the NoC IP generation flow using NocStudio. The user specifies a NocStudio command script that describes the user system requirements. NocStudio processes this script to construct a deadlock-free NoC that meets all the system requirements. The following files are generated by NocStudio for the NoC:

* NoC RTL
* NoC verification
* Sanity testbench
* Synthesis scripts
* HTML spec for the generated NoC (noc\_reference\_manual.html)

All the generated files are output to the project directory, whose name corresponds to the project name specified in the “new\_mesh” command in the NocStudio command script.



Figure 1 NoC IP generation flow

### Generating RTL from NocStudio

To generate NoC RTL, include “gen\_ip” command at the end of the NocStudio command script, and then process the script with NocStudio. For example, from the IP root directory, run the following command for GUI mode:

./NocStudio examples/example\_2h\_1ly.txt

Or the following command for batch mode:

./NocStudio examples/example\_2h\_1ly.txt -nogui

The last command in the above example script is “gen\_ip”. Once the command executes, a project directory called “example\_2h\_1ly/” is created which contains all the files and directories generated by NocStudio. Below is a list of key files related to RTL and verification component integration. For a complete list with detailed descriptions please refer to NetSpeed NocStudio Orion NSIP User Manual.

The files generated by default are listed in the table below.

Table 3 Files generated by NocStudio in project directory

|  |  |  |
| --- | --- | --- |
| **Name** | **Description** | **Type** |
| noc\_reference\_manual.html | HTML specification for the generated NoC, with information such as layer diagrams, traffic dependencies, full register specification and more | HTML documentation |
| doc\_files/ | Support files for HTML documentation | HTML documentation |
| transcript.log | Log file from the NocStudio run | Log file |
| commands.log | Command file from the NocStudio run | Log file |
| ns\_soc\_ip.v | Top level RTL module for the generated SoC IP | RTL |
| ns\_fabric.v | Top-level RTL module for the generated NoC | RTL |
| ns\_fabric\_modules.v | Support RTL files for the generated NoC | RTL |
| ns\_agent\_modules.v | Empty right now (for other host IP) | RTL |
| ns\_group\_modules.v | Hierarchical RTL group modules for the generated NoC. This file will be empty if user did not create any groups | RTL |
| ns\_bind\_checkers.svh | Binds file for checkers relevant to the generated ns\_fabric.v | Verification |
| ns\_node\_id\_table.sv | Support file for regbus end-to-end checker generated for the NoC by NocStudio if register bus is enabled | Verification |
| ns\_noc\_files.f | File list for integration of RTL and verification components into user environment | RTL and Verification |
| run\_test\_incisiv.sh | Run command to launch sanity bench for the generated NoC using Cadence Incisive Simulator | Sanity testbench |
| run\_test\_vcs.sh | Run command to launch sanity bench for the generated NoC using Synopsys VCS Simulator | Sanity testbench |
| noc\_rtl/\* | NoC RTL library | RTL |
| noc\_modifiable\_rtl/\* | NoC modifiable RTL | RTL |
| noc\_verif\_ip/\* | NoC verification library | Verification |
| noc\_verif\_bench/\* | NoC verification bench | Verification |
| noc\_verif\_cust/ns\_global\_defines.vh | Defines file for customer testbench integration | RTL and Verification |
| synth/\* | NoC synthesis environment. Please refer to NetSpeed Orion NSIP Physical Design Guidelines.pdf | Synthesis |
| scripts/\* | Scripts for sanity testbench | Verification |
| top.v | Support file for sanity testbench | Sanity testbench |
| system.f | Support file for sanity testbench | Sanity testbench |
| trace/ | Trace files | Sanity  testbench, performance traces |
| trace\_regbus/ | Trace files for register bus | Sanity testbench |
| protocol\_dependencies.csv | List of dependencies for the generated NoC | NoC property |
| bridge\_prop.csv | Information about the bridges in the NoC | NoC property |
| traffic/ | Traffic information specified by the user | NoC property |
| link\_costs.csv | Contains cost of all links in the NoC | NoC property |
| buffer\_costs.csv | Contains buffer cost of NoC components | NoC property |
| archive/\* | Location into which old run output is placed when a new run is started | Archive |

Optionally, a project tag can be applied for each NoC by adding ‘prop\_default tag\_project\_name yes’ to the NocStudio configuration file. This will ensure the generated RTL and verification files are tagged with the specific project name, allowing multiple NoCs to co-exist in the same design without name space collision. For more details on simulation with multiple NoCs, see section 2.3.

### Sanity Testbench

Once the NoC RTL has been generated, the next step is running the sanity testbench to perform a sanity check on the generated NoC RTL in simulation. This is a push-button method of instantiating the generated NoC RTL in a sanity testbench along with verification checkers, and running a sanity traffic pattern on the generated NoC RTL to validate basic operation of the NoC in simulation.

To run the sanity test, change to the project directory and invoke the following run script

If you are using Synopsys VCS Simulator, run:

run\_test\_vcs.sh

If you are using Cadence Incisive Simulator, run:

run\_test\_incisiv.sh

The run script compiles the sanity testbench and launches the simulation.

To enable waveform dumping, use the command line option -waves=1. For example:

run\_test\_vcs.sh -waves=1

or for Cadence Incisiv Simulator,

run\_test\_incisiv.sh -waves=1

On a successful compile and simulation, the following will appear at the prompt,

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* BUILD SUCCESSFUL \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Changing directory to: trace

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* SIMULATION PASSED \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

If the NoC has register bus enabled, there will be an additional sanity test which is run as part of the same script. It will perform register transactions to verify the connectivity of the register bus. On a successful simulation, the following will be logged in “run\_test\_incisiv.log” for a Cadence Incisive simulation. Similar information is in “run\_test\_vcs.log” for Synopsys VCS simulation.

Passing to irun for build

BUILD SUCCESSFUL

Passing to irun for simulation

REGBUS SIMULATION PASSED

Passing to irun for simulation

SIMULATION PASSED

(Note: For the script run\_regtest\_incisiv.sh or run\_regtest\_vcs.sh to run successfully, the user should **not** have the “create\_trace\_files” command after the “gen\_ip” command in the NocStudio command script because the files created as a result of “create\_trace\_files” command will overwrite the required stimulus generated by “gen\_ip”)

After the completion of simulation run, the presence of a file named SIM\_FAILED indicates a failure in simulation run. Depending on the simulator, the log file “run\_test\_vcs.log” or “run\_test\_incisiv.log” will list out any errors encountered during the build and simulation phase.

Presence of file named SIM\_PASSED in the project directory after the completion of simulation run indicates a successful simulation run.

With a successful simulation from the sanity testbench, the generated NoC RTL and verification IP are ready to be integrated.

# Integration of NoC

In the NocStudio project directory, “ns\_noc\_files.f” contains all the file references for NoC RTL and verification checkers. The following sections describe the integration process in detail.

## Integration of NoC RTL

To instantiate NoC RTL into customer environment, include the following lines in the file list for customer model compilation:

Set the environment variable NS\_PROJ\_PATH to point to the project directory that was created by NocStudio.

$NS\_PROJ\_PATH = /absolute/path/of/project/created/

Include the following line in the filelist for the project which instantiates the NoC.

-f ns\_noc\_files.f

The top-level module is “ns\_soc\_ip”, specified in ns\_soc\_ip.v.

### Reset

Table 4: NoC reset signals

|  |  |
| --- | --- |
| **Signal name** | **Description** |
| reset\_n\_system | Active-low reset pin for the NoC |

The reset pin is an active low reset pin. This reset pin is distributed to the different NoC elements. All reset signals are asynchronous at the NoC level. Internally to the NoC elements, reset may be asynchronously asserted, but are synchronously de-asserted.

The customer must ensure that the active low reset pin is driven low long enough so that all NoC elements are in reset at the same time. This is equal to the number of cycles for reset to propagate across the NoC + 16 clocks (time for which reset at each NoC element is asserted. Since cold reset is typically a long operation, a safe way to do this is to assert reset for a large number of slow clocks – 100 ticks of the slowest clock in NoC system. The reset logic within each module uses an asynchronous assert, synchronous de-assert mechanism. This involves registering the reset and ORing the registered version with the reset input. Because of this, de-assertion of the reset in the NoC modules will take additional time. Traffic to the NoC should be delayed until 2 cycles of the slowest clock after reset is de-asserted.

Physical distribution of reset to the various components of the NoC is the responsibility of the customer.

Each NoC element then synchronizes the reset to its clock before using it internally. The reset is de-asserted synchronously. Each NoC element can come out of reset at different times.

Additional details on the clocks, resets and physical integration and design guidelines are available in the NetSpeed Orion AMBA Physical Design Guidelines provided with the release.

### Clock

The following table lists the clock signal in the generated RTL.

Table 5 NoC clock signals

|  |  |
| --- | --- |
| **Signal name** | **Description** |
| clk\_<clock\_domain> | Clock pins are named as per the clock domain they belong to   * Default clock domain is “noc”, and hence default clock pin is “clk\_noc” * Regbus clock domain is “regbus” and the corresponding clock pin is  “clk\_regbus” * All other clock domains are added via “add\_clock\_domain”, and are named accordingly   A certain clock domain can be defined for the host clock of a bridge with an async/clock crosser interface. The corresponding clk\_<clock\_domain> pin at the NoC level will be internally connected to the host\_clk pin of the bridge (NoC element) |

The NoC SoC IP may have different clock domains that run asynchronously to each other. Instructions for adding clock domains can be found in the NetSpeed NocStudio Orion AMBA User Manual. In addition, host interfaces can operate at a clock asynchronous to the NoC clocks. The regbus layer can also operate on a clock asynchronous to NoC clocks. The physical fan-out and distribution of the clock is the responsibility of the customer.

### Clock Gating

All NoC elements support activity-based coarse clock gating. Coarse clock gating can be enabled or disabled through NocStudio programming.

#### 2.1.3.1 Clock Gating for NoCs without Regbus

In absence of Regbus layer, after programming is done through NocStudio, there is no further option to control clock gating. Coarse clock gating is either always enabled or disabled based on NocStudio programming.

#### 2.1.3.2 Clock Gating for NoCs with Regbus

For NoCs with Regbus and coarse clock gating enabled via NocStudio, further control is provided to disable or enable clock gating at the granularity of each NoC element through register programming. There is one *system\_cg\_or* pin for each NoC element. The s*ystem\_cg\_or* pin of a NoC element allows the coarse clock gating feature implemented in hardware to be overridden by software control. This is done by writing to a dedicated register RBSLVCG (details in NetSpeed Register spec) residing in Regbus Ring Master on that node. While changing the value on *system\_cg\_or* pin of a NoC element, it should be in an idle state and there should be no traffic flowing through it. Failure to observe this restriction will result in unpredictable or unrecoverable errors at system level. Also an important requirement is to wait the requisite number of cycles to allow the value written in RBSLVCG, to propagate to the target NoC element.

The regbus ring master on each node contains 32 registers, each mapping to one of 32 slaves on that node. These register outputs will drive output pins from regbus ring master and connect to *system\_cg\_or* pins of the appropriate NoC elements on that node.

The clock gating of NoC elements on Regbus Layer is controlled through an external pin *system\_cg\_or\_regbus*. The pin description is given in Table 6.

Table 6 Regbus layer clock gating signal

|  |  |
| --- | --- |
| **Signal name** | * **Description** |
| system\_cg\_or\_regbus | Overrides coarse clock gating feature for Regbus Ring Master and Regbus Master Bridge (logic ‘1’ implies coarse clock gating feature will not be used by hardware). This signal originates from a system level clock controller (from external to NoC fabric) and connects to the system\_cg\_or pin of all Regbus Ring Master and Regbus Master Bridge. |

### Register Bus

The NoC has an optional, distributed register network consisting of registers used for debug visibility, performance status collection, and error logging. This register bus is built as an independent NoC layer with a single access port. This port uses a modified AXI4-Lite protocol.

Some properties of the regbus master interface are listed below:

* 32-bit data width
* Supports AxLEN 0 or 1 for accessing 32-bit and 64-bit registers, respectively
* Up to 16 outstanding read/write requests can be issued to the NoC regbus layer

NoC registers are automatically created by NocStudio and placed in a fixed register bus address map. This address map is unrelated to any address map within the main NoC design.

For details of the registers and register address map, refer to noc\_reference\_manual.html and noc\_registers.csv (which only appears if register bus is enabled) generated by NocStudio in the project directory.

There are two NocStudio configuration options through which read and write commands can be sent to the regbus layer:

* **Default option:** A master agent connects directly to the Regbus layer port for reading or writing registers of the NoC.

### Interrupts

Every NoC router and bridge has an interrupt output signal. Interrupt is asserted when a fatal error is encountered in a NoC element. The errors are also logged in interrupt status registers of the NoC element. Interrupts from different NoC elements are combined within the NoC by a specialized network. A single combined interrupt is brought out on the NoC external interface. The errors are also logged within the NoC registers, which are described in the HTML documentation generated by NocStudio.

Table 7 NoC interrupt signals

|  |  |
| --- | --- |
| **Signal name** | **Description** |
| Interrupt | Single interrupt pin which combines all interrupt events inside the NoC |

The interrupt signals are outputs of individual NoC elements, and will exist as local pins at the physical boundary of those elements. Interrupt network handles pipelined routing of the interrupt signals along NoC channels. Combining of the interrupt signals across different clock domains in the NoC is also handled internally. Currently the combined interrupt pin is transported to the register bus master’s physical grid location if register bus is enabled in a configuration. When register bus is disabled, NocStudio picks a position to transport the combined reset to.

If the register bus is instantiated in the NoC, it can be used to access interrupt control and status registers. Interrupt mask registers can be set to enable or disable some interrupts. When an interrupt occurs, a status register can be accessed to determine what the cause of the interrupt was. This status can be cleared in order to de-assert the interrupt signal. If an interrupt mask is modified to enable an event to trigger an interrupt, the status for that interrupt should be cleared by the user before changing the mask or the interrupt will trigger immediately.

If the register bus is not present in the NoC, the interrupt signals will still exist. Since there is no way to vary status or to change the interrupt mask, the mask will be set to only enable fatal error conditions. If the interrupt is ever triggered, there will be no way to de-assert the interrupt. This can still be useful to indicate a fatal error.

### Event Counters

When the register bus is present in the NoC, it is possible to configure the NoC elements to count events for either performance measurement or for debug purposes. The routers and bridges each have a set of control registers and counters. The control registers allow the user to specify which event(s) they would like to have counted. As soon as the register is programmed, the events will start to be counted. The counter register is readable and writeable. The counter can be read to see the current count. It can be written to clear the count, or to initialize the count to a specific value. The counter will keep counting even when it hits its maximum value, which will cause it to overflow and start over at count zero.

The event counters can be set up to trigger an interrupt when the counter overflows. The overflow condition updates the interrupt status register. The interrupt mask can be used to enable or disable that interrupt. Since the count can be initialized, the user can trigger an interrupt after N number of events within the count window by initializing the counter to a value where incrementing N times will cause an overflow.

The registers controlling the event counters are independent, so intelligent use of the registers is required. Before switching from one set of counts to another, it may be useful to set the controls to not count any events. At that point the counter can be cleared, and the status bit in the interrupt register can also be cleared. Programming the interrupt mask should also happen at this time. Once all of these registers are set correctly, the event control register can be programmed to start counting a specified event.

## Integration of NoC Verification Checkers

For details of the NetSpeed IP verification checkers, see Section 3. Each checker binds to the corresponding RTL instance as shown in Figure 2 NoC checkers binding to RTL.



Figure 2 NoC checkers binding to RTL

To integrate NoC checkers into user testbench, the user needs to perform the following,

* Add the following line to the user testbench:

`include "ns\_bind\_checkers.svh”

“ns\_bind\_checkers.svh” is located in project directory created by NocStudio. It binds all the checkers provided by NocStudio IP to the respective RTL instances regardless of testbench hierarchy. This file can be used without any changes.

* Set the environment variable NS\_PROJ\_PATH which points to the directory created by NocStudio.

$NS\_PROJ\_PATH = /absolute/path/of/project/created/

* Adjust `define in the following file according to the recommended usage inTable 11and Table 13,

noc\_verif\_cust/ns\_global\_defines.vh

This file contains the `define variables used by the verification checkers.

* NS\_NOC\_TOP : This signal is used to bind checkers. Please set the hierarchical path for the ns\_soc\_ip instance in the testbench where this is being simulated.
* NS\_END\_OF\_SIM: Hierarchical path to the END OF SIM signal from TB. Map to 1-bit signal with single rising edge when simulation ends. This signal is used to trigger end of test checks in checkers. Set to 1'b0 to disable exit checks.

## Integration of multiple NoCs

To integrate RTL from multiple NoCs into the same design for simulation, the following steps must be followed:

* The NocStudio configuration file for each NoC must contain the command ‘prop\_default tag\_project\_name yes’. This allows each NoC to have unique top-level module names and log file names.
* For each configuration file, NocStudio generates the following,

Table 8 Directory and files with tag\_project\_name

|  |  |  |
| --- | --- | --- |
| **Name** | **Description** | **Type** |
| noc\_verif\_ip\_proj/ | Custom verification files specific to each NoC | Verification |
| ns\_<project>\_soc\_ip.v | ns\_soc\_ip.v for each NoC | RTL |
| ns\_<project>\_fabric.v | ns\_fabric.v for each NoC | RTL |
| ns\_<project>\_fabric\_modules.v | ns\_fabric\_modules.v for each NoC | RTL |
| ns\_<project>\_group\_modules.v | ns\_group\_modules.v for each NoC | RTL |
| ns\_<project>\_agent\_modules.v | ns\_agent\_modules.v for reach NoC | RTL |
| ns\_<project>\_stream\_noc\_end\_to\_end\_checker.sv | ns\_stream\_noc\_end\_to\_end\_checker.sv for each NoC | Verification |
| ns\_<project>\_stream\_struct.sv | ns\_stream\_struct.sv for each NoC | Verification |
| ns\_<project>\_bind\_checkers.svh | ns\_bind\_checkers.svh for each NoC | Verification |

* In ns\*bind\_checkers.svh for multiple NoCs, there could be bind statements for the same module in more than one file. Resolve this by removing redundant bind statements for the same module.
* Create a combined ns\_global\_defines.vh by taking noc\_verif\_cust/ns\_global\_defines.vh from the first NoC project directory and making the following updates,
  + Replace `NS\_NOC\_TOP with unique defines for each NoC,

*`define NS\_NOC\_TOP\_PROJECT1 <hierarchical\_path\_to first NoC>*

*`define NS\_NOC\_TOP\_PROJECT2 <hierarchical\_path to second NoC>*

* + Replace `NS\_NOC\_END2END\_CHECKER\_EN and `NS\_REGBUS\_END2END\_CHECKER\_EN with unique defines for each NoC,

*`define NS\_<PROJECT1>\_E2E\_CHECKER\_TOP `NS\_NOC\_TOP\_PROJECT1.u\_ns\_fabric.ns\_<project1>\_stream\_noc\_end\_to\_end\_checker*

*`define NS\_<PROJECT1>\_REGBUS\_E2E\_CHECKER\_TOP `NS\_NOC\_TOP\_PROJECT1.u\_ns\_fabric.ns\_<project1>\_regbus\_e2e\_checker*

*`define NS\_<PROJECT2>\_E2E\_CHECKER\_TOP `NS\_NOC\_TOP\_PROJECT2.u\_ns\_fabric.ns\_<project2>\_stream\_noc\_end\_to\_end\_checker*

*`define NS\_<PROJECT2>\_REGBUS\_E2E\_CHECKER\_TOP `NS\_NOC\_TOP\_PROJECT2.u\_ns\_fabric.ns\_<project2>\_regbus\_e2e\_checker*

Where,

<project1> denotes name of the project for the first NoC in lower case.

<PROJECT1> denotes name of the project for the first NoC in upper case.

<project2> denotes name of the project for the second NoC in lower case.

<PROJECT2> denotes name of the project for the second NoC in upper case.

* Consolidate ns\_noc\_files.f from each NoC to create final simulation file list for multiple NoCs. For example,

+libext+.v+.sv

+incdir+$NS\_PROJ1\_PATH

+incdir+$NS\_PROJ1\_PATH/noc\_rtl

+incdir+$NS\_PROJ1\_PATH/noc\_verif\_ip

+incdir+$COMBINED\_PATH/noc\_verif\_cust

-y $NS\_PROJ1\_PATH/noc\_rtl

-y $NS\_PROJ1\_PATH/noc\_modifiable\_rtl

-y $NS\_PROJ1\_PATH/noc\_verif\_ip

-y $NS\_PROJ1\_PATH/noc\_verif\_ip\_proj

-y $NS\_PROJ2\_PATH/noc\_verif\_ip\_proj

$NS\_PROJ1\_PATH/noc\_verif\_ip/ns\_regbus\_struct.sv

$NS\_PROJ1\_PATH/ns\_<project1>\_stream\_struct.sv

$NS\_PROJ1\_PATH/ns\_<project1>\_stream\_noc\_end\_to\_end\_checker.sv

$NS\_PROJ1\_PATH/ns\_<project1>\_fabric\_modules.v

$NS\_PROJ1\_PATH/ns\_<project1>\_fabric.v

$NS\_PROJ1\_PATH/ns\_<project1>\_group\_modules.v

$NS\_PROJ1\_PATH/ns\_<project1>\_agent\_modules.v

$NS\_PROJ1\_PATH/ns\_<project1>\_soc\_ip.v

$NS\_PROJ2\_PATH/ns\_<project2>\_stream\_struct.sv

$NS\_PROJ2\_PATH/ns\_<project2>\_stream\_noc\_end\_to\_end\_checker.sv

$NS\_PROJ2\_PATH/ns\_<project2>\_fabric\_modules.v

$NS\_PROJ2\_PATH/ns\_<project2>\_fabric.v

$NS\_PROJ2\_PATH/ns\_<project2>\_group\_modules.v

$NS\_PROJ2\_PATH/ns\_<project2>\_agent\_modules.v

$NS\_PROJ2\_PATH/ns\_<project2>\_soc\_ip.v

Where,

* $NS\_PROJ1\_PATH denotes the NocStudio project directory for the first NoC.
* $NS\_PROJ2\_PATH denotes the NocStudio project directory for the second NoC.
* $COMBINED\_PATH/verif\_cust/ denotes the path to the combined ns\_global\_defines.vh for all NoCs described above.
* For each additional NoC, simply replicate the same lines for $NS\_PROJ2\_PATH to point to the project directory of the new NoC.

This mechanism can be used to integrate more than two NoCs with no upper limit to the number of NoCs.

## Synthesis

Please refer to “NetSpeed Orion Physical Design Guidelines”.

## C++ Performance Model

Please refer to “NetSpeed Orion C++ NoC Model Spec”.

## Supported Tools

Supported versions of tools and languages:

* NoC RTL uses Verilog-2001 (IEEE Std 1364™-2001) syntax and its support must be enabled in the tool flow.
* NoC simulation environment uses SystemVerilog IEEE Std 1800-2009
* Simulator: Cadence Incisiv 13.20.004
* Simulator: Synopsys VCS J-2014.12-1\_Full64
* Synthesis: Cadence Encounter RTL Compiler version RC12.23 - v12.20-s027\_1 (64-bit)

# NoC Verification Components

The NoC IP package contains both unit-level and NoC-level checkers to ensure functional correctness of the NoC during simulation.

## Overview of Checkers

If NocStudio is enabled with **verification components generation privileges**, it would provide the following checker files:

Table 9 NetSpeed checkers

|  |  |
| --- | --- |
| **Checkers** | **Instantiated** |
| NoC end-to-end checker | One instance per NoC |
| Streaming bridge checker | One instance per streaming bridge RTL |
| Router checker | One instance per router RTL |
| Regbus end-to-end checker | One instance per regbus layer |

## Environment Setup for Integration

In order to integrate in NoC verification components, a list of `define variables need to be defined.

Table 10 `define variables for model build

|  |  |  |
| --- | --- | --- |
| **`define variable name** | **Description** | **Notes** |
| `NS\_NOC\_TOP | Map to hierarchical path to NoC within user testbench | None. |
| `NS\_END\_OF\_SIM | Map to a signal with one rising edge transition at end of simulation to trigger exit checks in the various NoC checkers | This signal goes high once and stays high for at least 2 cycles at the end of simulation, when all traffic in NoC is expected to have quiesced. |
| `NS\_NOC\_END2END\_CHECKER\_EN | Set to 1 to enable, 0 to disable NoC end-to-end checker | Recommend mapping to a plusarg variable to allow run-time control of value. |
| `NS\_E2E\_LOG | Set to 1 to enable, 0 to disable traffic logging by NoC end-to-end checker | Recommend mapping to a plusarg variable to allow run-time control of value. |
| `NS\_STRBRDG\_CHECKER\_EN | Set to 1 to enable, 0 to disable NoC streaming bridge checker | Recommend mapping to a plusarg variable to allow run-time control of value. |
| `NS\_ROUTER\_CHECKER\_EN | Set to 1 to enable, 0 to disable NoC router checker | Recommend mapping to a plusarg variable to allow run-time control of value. |
| `NS\_REGBUS\_END2END\_CHECKER\_EN | Set to 1 to enable, 0 to disable register bus end-to-end checker | Recommend mapping to a plusarg variable to allow run-time control of value |
| `NS\_REGBUS\_E2E\_LOG | Set to 1 to enable, 0 to disable traffic logging by register bus end-to-end checker | Recommend mapping to a plusarg variable to allow run-time control of value. |

## Usage Modes

The following table describes a set of recommended usage modes for enabling the NoC checkers. The tradeoff is made between debug visibility and simulation performance penalty for increased visibility.

Table 11 Recommended checker settings

|  |  |  |  |
| --- | --- | --- | --- |
| **Usage mode** | **Bring up**  **Mode** | **Heavy Debug**  **Mode** | **Code Stable**  **Mode** |
| `NS\_NOC\_END2END\_CHECKER\_EN | 1 | 1 | 1 |
| `NS\_E2E\_LOG | 1 | 1 | 0 |
| `NS\_REGBUS\_END2END\_CHECKER\_EN | 1 | 1 | 1 |
| `NS\_REGBUS\_E2E\_LOG | 1 | 1 | 0 |
| `NS\_STRBRDG\_CHECKER\_EN | 1 | 1 | 0 |
| `NS\_ROUTER\_CHECKER\_EN | 0 | 1 | 0 |
| `NS\_ROUTER\_CHECKER\_COMPILE\_EN | Do not define | Define | Do not define |
| `NS\_REGBUS\_CHECKER\_EN | 1 | 1 | 0 |
| `NS\_REGBUS\_E2E\_LOG | 1 | 1 | 0 |

## Checkers

### Terminology

The types of checks that are performed are divided into the following categories:

**Protocol** – These checks enforce adherence to the NoC interface protocol.

**Functional** – These checks verify the functionality of the RTL.

**Exit** – These checks are performed at the end of simulation to verify that the NoC is in a proper idle state at the end of simulation.

### Streaming Bridge Checkers

The streaming bridge checkers are responsible for monitoring streaming bridge RTL during simulation. Each instance of streaming bridge RTL has a corresponding streaming bridge checker monitoring its behavior. The streaming bridge checkers enforce adherence to interface protocol on all bridge interfaces. In addition, they also perform micro-architectural checks to ensure functional correctness of the streaming bridge RTL. At end of simulation, when there should be no traffic in the NoC, these checkers perform exit checks to ensure each instance of streaming bridge RTL is in a proper idle state. The following is a list of the checks performed by the streaming bridge checker. Violation of any one of these will trigger an error in simulation.

Table 12 Streaming bridge checks

|  |  |  |
| --- | --- | --- |
| **Description of check** | **Instantiated(per bridge or interface)** | **Type of check** |
| Each transaction begins with one and only one SOP | Host port TX interface | Protocol |
| Each transaction ends with one and only one EOP | Host port TX interface | Protocol |
| QoS field must be constant during transmission of the transaction | Host port TX interface | Protocol |
| Destination hostport id field must be constant during transmission of the transaction | Host port TX interface | Protocol |
| Destination interface id must be constant during transmission of the transaction. | Host port TX interface | Protocol |
| Credit overflow | Host port TX interface | Protocol |
| Credit underflow | Host port TX interface | Protocol |
| No unknown (x or z) control signals when not in reset | Host port TX interface | Protocol |
| No unknown (x or z) data packets | Host port TX interface | Functional |
| Payload size is no more than maximum size(64KB) | Host port TX interface | Protocol |
| Protocol violation of double EOP will raise interrupt | Host port TX interface | Functional |
| Protocol violation of packet transmission without SOP will raise interrupt | Host port TX interface | Functional |
| Packet with illegal destination | Host port TX interface | Functional |
| Packet with illegal destination will raise interrupt | Host port TX interface | Functional |
| Each transaction begins with one and only one SOP | Host port RX interface | Protocol |
| Each transaction ends with one and only one EOP | Host port RX interface | Protocol |
| Credit overflow | Host port RX interface | Protocol |
| Credit underflow | Host port RX interface | Protocol |
| Payload size is no more than maximum size | Host port RX interface | Protocol |
| No Unknown (x or z) control signals when not in reset | Host port RX interface | Protocol |
| No unknown (x or z) data packets | Host port RX interface | Functional |
| Internal micro-architectural checks | Streaming bridge | Functional |
| Should not have any interrupt | Streaming bridge | Functional |
| No packets in flight. | Host port RX interface | Exit |
| All credits have restored to initial value. | Host port RX interface | Exit |
| No packets in flight. | Host port TX interface | Exit |
| All credits have restored to initial value. | Host port TX interface | Exit |
| All bridge FIFOs in RTL are empty | Streaming bridge | Exit |

For a subset of the above checkers, fine-grained user-control is provided to individually enable or disable the checkers. For each check listed in the following table, setting the corresponding `define to 0 enables the check; setting it to 1 disables the check. They should be set to the default value in all cases except for error testing that may require these to be set otherwise.

Table 13 Fine-grained user-control of streaming bridge checkers

|  |  |  |
| --- | --- | --- |
| **Description of check** | **`define to control** | **Default value** |
| Packet with illegal destination | `NS\_STRBRDG\_INVALID\_DEST\_CHECK\_DISABLE | 0 |
| Should not have any interrupt | `NS\_STRBRDG\_INTERRUPT\_CHECK\_DISABLE | 0 |
| No unknown (x or z) data packets | `NS\_STRBRDG\_DATA\_XZ\_CHECK\_DISABLE | 0 |

### Router Checkers

The router checkers are responsible for monitoring router RTL during simulation. Each instance of router RTL has a corresponding router checker monitoring its behavior. The router checkers enforce adherence to protocol on all ports. In addition, they ensure functional correctness of router RTL during simulation. At the end of simulation, when there should be no traffic in the NoC, these checkers perform exit checks to ensure each instance of router RTL is in a proper idle state.

Table 14 Router checks

|  |  |  |
| --- | --- | --- |
| **Description of check** | **Instantiated** | **Type of check** |
| No Unknown (x or z) control signals when not in reset | Each Router Input Port | Protocol |
| Reset has to be active for at least 16 clocks | Each Router Input Port | Protocol |
| VC valid is one-hot. | Each Router Input Port | Protocol |
| Output port should be 0-4 | Each Router Input Port | Protocol |
| Output port should not be the same as input port | Each Router Input Port | Protocol |
| Credit overflow per VC | Each Router Input Port | Protocol |
| Credit underflow per VC | Each Router Input Port | Protocol |
| Packets received on disabled VC | Each Router Input Port | Protocol |
| SB info should remain constant throughout the packet. | Each Router Input Port | Protocol |
| Packet Type should remain constant throughout the packet. | Each Router Input Port | Protocol |
| Output port should remain constant throughout the packet. | Each Router Input Port | Protocol |
| 2 SOPs without EOP in between for a VC | Each Router Input Port | Protocol |
| 2 EOPs without SOP in between for a VC | Each Router Input Port | Protocol |
| BV is non-zero only at EOP | Each Router Input Port | Protocol |
| No Unknown (x or z) control signals when not in reset | Each Router Output Port | Protocol |
| Reset has to be active for at least 16 clocks | Each Router Output Port | Protocol |
| VC valid is one-hot. | Each Router Output Port | Protocol |
| Output port should be 0-4 | Each Router Output Port | Protocol |
| Credit overflow per VC | Each Router Output Port | Protocol |
| Credit underflow per VC | Each Router Output Port | Protocol |
| Packets received on disabled VC | Each Router Output Port | Protocol |
| SB info should remain constant throughout the packet. | Each Router Output Port | Protocol |
| Packet Type should remain constant throughout the packet. | Each Router Output Port | Protocol |
| Output port should remain constant throughout the packet. | Each Router Output Port | Protocol |
| 2 SOPs without EOP in between for a VC | Each Router Output Port | Protocol |
| 2 EOPs without SOP in between for a VC | Each Router Output Port | Protocol |
| BV is non-zero only at EOP | Each Router Output Port | Protocol |
| All credits have restored to initial value. | Each Router | Exit |
| All router FIFOs are empty | Each Router | Exit |

### Register Bus End-to-End Checker

The register bus end-to-end checker is a scoreboard that tracks register traffic on the register bus layer to ensure every register access is properly routed to the correct destination register ring, and every response is propagated back to the master in the correct order with the correct data content. The following checks are performed.

Table 15 Register bus end-to-end checks

|  |  |  |
| --- | --- | --- |
| **Description of check** | **Instantiated** | **Type of check** |
| ARVALID, AWVALID, WVALID, RREADY are never x or z | Regbus Master Bridge | Protocol |
| ARLEN and AWLEN are either 0 or 1 | Regbus Master Bridge | Protocol |
| Every AR request that enters the regbus master bridge is tracked with its corresponding R response for the roundtrip check to ensure correct propagation of command fields, propagation of content and ordering within the regbus layer | NoC | Functional |
| Every AR request that enters the regbus master bridge arrives at the correct destination regbus ring master, with the correct ARADDR, ARPROT, ARLEN, node\_id, ring\_id, seqnum, in the correct order | NoC | Functional |
| Every read request packet that enters the regbus ring master is tracked with its corresponding response packet for the roundtrip check to ensure correct propagation of command fields, propagation of content and ordering within each regbus ring master | Regbus Ring Master | Functional |
| Every R response that leaves the regbus ring master arrives at the regbus master bridge with the correct RDATA, RRESP, RLAST, in the correct order | NoC | Functional |
| Every pair of AW and W requests that enters the regbus master bridge is tracked together with the corresponding B response for roundtrip check to ensure correct propagation of command fields, propagation of data and ordering within the regbus layer | NoC | Functional |
| Every AW and W request that enters the regbus master bridge arrives at the correct destination regbus ring master with the correct AWADDR, AWPROT, AWLEN, WDATA, WSTRB and WLAST, node\_id, ring\_id, seqnum, in the correct order | NoC | Functional |
| Every write request packet that enters the regbus ring master is tracked with its corresponding response packet for the roundtrip check to ensure correct propagation of command fields, propagation of content and ordering within each regbus ring master | Regbus Ring Master | Functional |
| Every B response that leaves a regbus ring master arrives at the regbus master bridge with the correct BRESP, in the correct order | NoC | Functional |
| There is one and only one SOP per regbus ring master packet | Regbus Ring Master | Protocol |
| There is one and only one EOP per regbus ring master packet | Regbus Ring Master | Protocol |
| Valid is high only when a regbus ring master packet is in-flight | Regbus Ring Master | Protocol |
| No regbus request or response is in-flight | NoC | Exit |
| All regbus requests and responses during the simulation are accounted for | NoC | Exit |

The register bus end-to-end checker has the capability of generating a set of traffic log files during the simulation to provide visibility of the traffic on the regbus master bridge and on each regbus ring master connected to the regbus master bridge. The following table lists the settings required to enable the traffic logs.

Table 16 Settings to enable register bus end-to-end traffic logs

|  |  |
| --- | --- |
| **`define to control** | **Value** |
| `NS\_REGBUS\_END2END\_CHECKER\_EN | 1 |
| `NS\_REGBUS\_E2E\_CHECKER\_LOG | 1 |

The file names of the logs are of the following format with <node\_id> corresponding to the node id of the regbus master bridge and each regbus ring master assigned by NocStudio,

Table 17 Register bus end-to-end traffic logs

|  |  |
| --- | --- |
| **File name** | **Description** |
| ns\_regbus\_mbrdg\_<node\_id>.log | Register bus master bridge traffic log for AW, W, B, AR and R channels |
| ns\_regbus\_ring\_master\_<node\_id>.log | Register bus ring master traffic log for request (regbus master bridge to regbus ring master) and response (regbus ring master to regbus master bridge) channels |

As shown in the above table, two types of log files are created for every NoC with regbus layer. The “ns\_regbus\_mbrdg<node\_id>.log” displays transactions received on the modified AXI4-Lite interface of the regbus master bridge. Each “ns\_regbus\_ring\_master\_<node\_id>.log” displays transactions received on the regbus ring master interface with the regbus master bridge.

### Clock Control Signal Checks

Table 18 Clock control signal checks

|  |  |  |
| --- | --- | --- |
| **Description of check** | **Instantiated(per bridge or interface)** | **Type of check** |
| scan\_mode pin can only toggle when streaming bridge is idle | Streaming bridge | Functional |
| system\_cg\_or pin can only toggle when streaming bridge is idle | Streaming bridge | Functional |
| system\_clk\_en pin can only toggle when streaming bridge is idle | Streaming bridge | Functional |

### NoC End-to-End Checker

NoC end-to-end checker is used to verify the correct operation of the NoC design. All packets transmitted into the NoC are tracked to construct a global reference database of expected results at the output interfaces. The packets transmitted out of each NoC interface are then compared against the global reference database to ensure packets arrive with correct content and in the correct order. Unexpected packets, packets sent to an incorrect destination, lost packets, extra packets, corrupt packets or packets received out of order would all be detected and flagged as errors. At end of simulation, when there should be no traffic in-flight in the NoC, the end-to-end checker ensures that the NoC is in a proper idle state.

Table 19 NoC end-to-end checks

|  |  |  |
| --- | --- | --- |
| **Description of check** | **Instantiated** | **Type of check** |
| Each packet is routed to the correct port with the correct content and size | Per NoC | Functional |
| Ordering of traffic is maintained for the same source, source interface, destination, destination interface and QoS value | Per NoC | Functional |
| Every packet is accounted for, no extra packet or missing packet at destinations | Per NoC | Functional |
| No traffic is in-flight within the NoC | Per NoC | Exit |
| All verification FIFOs and queues are empty | Per NoC | Exit |
| All verification checkers indicate no error | Per NoC | Exit |

### NoC End-to-End Traffic Logs

The NoC end-to-end checker has the capability of generating a set of end-to-end traffic log files as each packet successfully exits the NoC and passes all the checks in the end-to-end checker. The following table has the setting to enable the end-to-end logs.

Table 20 Settings to enable end-to-end traffic logs

|  |  |
| --- | --- |
| **`define to control** | **Value** |
| `NS\_NOC\_END2END\_CHECKER\_EN | 1 |
| `NS\_E2E\_LOG | 1 |

The file names of the logs are of the format,

ns\_noc\_packets\_to\_<*destination\_hostport*>\_<*destination\_interface*>.log

One log file is produced for each destination NoC interface. Each log file records the end-to-end information for every packet as it’s received on the destination NoC interface, in the following format:

<*sim\_time*> : pkt\_sent\_sop\_time=<*pkt\_sent\_sop\_time*>, pkt\_sent\_eop\_time=<*pkt\_sent\_eop\_time*>, pkt\_received\_sop\_time=<*pkt\_received\_sop\_time*>, pkt\_received\_eop\_time=<*pkt\_received\_eop\_time*>, pkt\_length=<*pkt\_length*>, src\_id=<*src\_id*>, src\_intf=<*src\_intf*>, qos=<*qos*>, dst\_id=<*dst\_id*>, dst\_intf=<*dst\_intf*>, pkt\_dataQ=<first\_data\_segment>...<*last\_data\_segment*>, match\_unique=<*match\_unique*>, match\_cnt=<*match\_cnt*>

Table 21 Nomenclature of end-to-end traffic logs

|  |  |
| --- | --- |
| **Field name** | **Description** |
| <sim\_time> | Simulation time when the packet passed all end-to-end checker checks at destination NoC interface |
| <pkt\_sent\_sop\_time> | Simulation time of when the packet’s SOP was sent into the NoC |
| <pkt\_sent\_eop\_time> | Simulation time of when the packet’s EOP was sent into the NoC |
| <pkt\_received\_sop\_time> | Simulation time of when the packet’s SOP exited the NoC |
| <pkt\_received\_eop\_time> | Simulation time of when the packet’s EOP exited the NoC |
| <src\_id> | Source hostport id of the streaming bridge from which the packet was sent |
| <src\_intf> | Source interface id of the streaming bridge from which the packet was sent |
| <qos> | QoS value of the packet |
| <dst\_id> | Destination hostport id of the streaming bridge to which the packet was sent |
| <dst\_intf> | Destination interface id of the streaming to which the packet was sent |
| <first\_data\_segment> | Data content at the beginning of the packet |
| <last\_data\_segment> | Data content the end of the packet |
| <match\_unique> | When the packet was received, whether the packet was a unique match in the reference database in end-to-end checker or whether there were multiple matches of identical packets. If this is 1, then the packet can be uniquely identified and has satisfied the end-to-end check. If this is 0, then this packet is not a unique packet but has satisfied one of the possible legal outcomes without violating ordering requirements |
| <match\_cnt> | If <match\_unique> is 0, this field indicates the total number of expected packets that are identical which are possible matches for the packet at the time of arrival at the destination NoC interface |

Example traffic flow from a NoC end-to-end log file:

4135: pkt\_sent\_sop\_time=3775, pkt\_sent\_eop\_time=3795, pkt\_received\_sop\_time=4125, pkt\_received\_eop\_time=4135, pkt\_length=105 bits, src\_id=0, src\_intf=0, qos=3, dst\_id=1, dst\_intf=1, pkt\_dataQ=0x705fa05fa...705fa05fa, match\_unique=1, match\_cnt=1

2670 Seely Avenue

Building 11

San Jose CA 95134

[www.netspeedsytems.com](file:///C:\Users\AE1\Downloads\www.netspeedsytems.com)