|  |
| --- |
| NocStudio Orion NSIP Tutorial  Revision 0.6  March 25th, 2015 |

NocStudio Orion NSIP Tutorial

Revision History

|  |  |  |
| --- | --- | --- |
| Rev 0.6 | March 25th, 2015 | Q1’15 Release |

About This Document

This document is a tutorial for NocStudio Orion Streaming IP. Using NocStudio, users can define NoC architectures, describe specifications and requirements, optimize the NoC design and finally generate the Orion NoC IP files such as RTL, testbench, C++ models, synthesis scripts, NoC IP documentation etc.

Audience

This document is intended for users of NocStudio:

* NoC Architects
* NoC Designers
* SoC Architects

Related Documents

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

* NetSpeed Orion User Manual
* NetSpeed Orion IP Integration Spec

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](#_Toc413659708)

[Audience 2](#_Toc413659709)

[Related Documents 2](#_Toc413659710)

[Customer Support 2](#_Toc413659711)

[1 Tutorial and Example Session 4](#_Toc413659712)

[1.1 Starting NocStudio 4](#_Toc413659713)

[1.2 NetSpeed Streaming Bridge Tutorial and Example Interactive Session 4](#_Toc413659714)

[1.2.1 Comparing with Map\_opt 15](#_Toc413659715)

[1.2.2 Enabling Regbus 16](#_Toc413659716)

# Tutorial and Example Session

## Starting NocStudio

NocStudio GUI can be launched by running NocStudio.sh from the command prompt in Linux or by running NocStudio-GUI.exe on Windows.

A filename containing a list of NocStudio commands (NocStudio command script) can be provided as an argument in which case upon launching NocStudio the contents of the file would be executed by NocStudio. When a filename is provided, NocStudio can also be run in a nogui mode by providing a –nogui switch at the end in which case the GUI will not be launched, and the contents of the file would be executed silently by NocStudio.

Linux$ ./NocStudio.sh

Linux$ ./NocStudio.sh commands.txt

Linux$ ./NocStudio.sh commands.txt –nogui

Win> NocStudio-GUI.exe

Win> NocStudio-GUI.exe commands.txt

Win> NocStudio-GUI.exe commands.txt –nogui

## NetSpeed Streaming Bridge Tutorial and Example Interactive Session

$ prop\_default cell\_size 8

$ prop\_default data\_width 64

$ new\_mesh 4 4 3 stream\_test

$ mesh\_prop virtual\_ok yes

First we set the default cell size and data width at 9 and 72 respectively (if user is using cell size other than 9, then it is recommended to set these two props together before creating mesh or before adding hosts so that all default interface widths will be initialized at correct values). Then we create a new project directory named “stream\_test” to store all files related to the project. This command does not load an existing project that may exist in that directory, with the exception of traffic trace files. This command also clears any existing configuration and initializes a new mesh with 3 layers 4 columns and 4 rows. The next command lets NocStudio know that virtual nodes of the NoC can be used to place a host and hostport.

$ add\_host host00 color blue bridge m stream

$ add\_host host01 color blue bridge m stream

$ add\_host host02 color blue bridge m stream

$ add\_host host03 color blue bridge m stream

$ add\_host host04 color blue bridge m stream

$ add\_host host05 color blue bridge m stream

$ add\_host host06 color blue bridge m stream

$ add\_host host07 color blue bridge m stream

$ add\_host host08 color blue bridge m stream

$ add\_host host09 color blue bridge m stream

$ add\_host host10 color blue bridge m stream

$ add\_host host11 color blue bridge m stream

Next, 12 hosts are added, each with a single port named m of type stream. In this example, the streaming hostport supports all four interfaces, i.e. interface a, b, c, and d. The position, shape and size of the hosts are not described, so they are the default single-cell-sized hosts, and NocStudio adds them at the first free position in the mesh, including the virtual node positions. The resulting display is shown below. There are three figures because three NoC layers are being built.

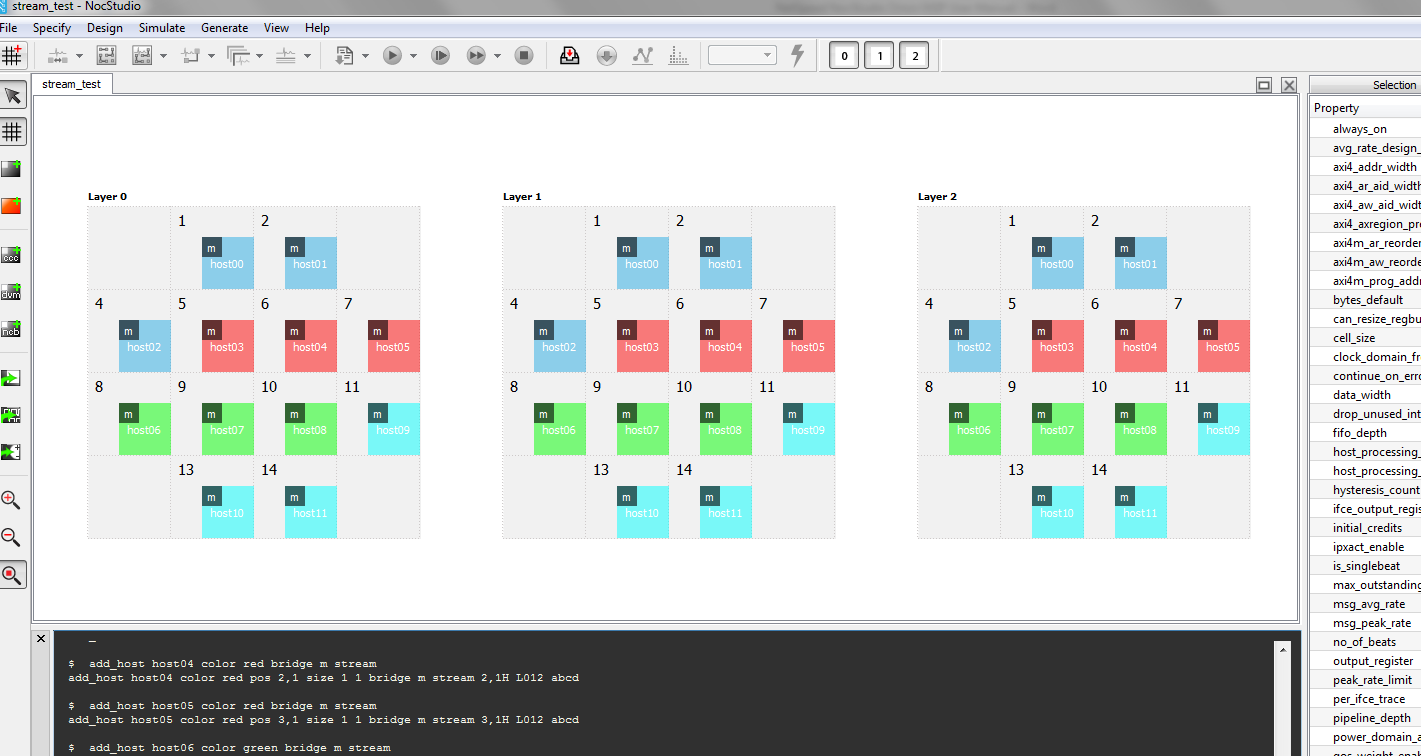


Figure 1: NoC display after adding 12 hosts with single streaming ports

$ add\_alias hosts host00 host01 host02 host03 host04 host05 host06 host07 host08 host09 host10 host11

Next, user creates an alias named hosts containing all 12 hosts.

$ add\_traffic rates 0.025 0.01 hosts/m.a <-1 -1 4 64 0> hosts/m.a

$ add\_traffic rates 0.01 0.01 hosts/m.b <-1 -1 4 64 1> hosts/m.b

$ add\_traffic rates 0.01 0.01 host00/m.a host01/m.a <-1 -1 4 64 2> host10/m.b host11/m.b

$ map

Next, user adds traffic transaction between various hostports and interfaces. The first traffic transaction is from interface a of all hostports to interface a of all hostports. This traffic is given average rate 0.025, peak rate 0.01, number of beats 4, latency requirement 64 cycles, and is mapped to NoC layer 0. Note that each initiator sends 0.025 beat rate of traffic to a target. Therefore each intiator sends n x beat rate, and each target receives n x beat rate. The second traffic transaction is from interface b of all hostports to interface b of all hostports, and is mapped to NoC layer 1. The third traffic transaction is from interface a, of hostports host00/m and host01/m to interface b of hostports host10/m and host11/m on NoC layer 2. Finally, the **map** command maps the traffic.

$ add\_traffic rates 0.025 0.01 hosts/m.b <-1 -1 1 64 0> hosts/m.a/m.a <-1 -1 4 64 2> hosts/m.b

$ map

Next, more complex traffic is added. Here there is multi-hop traffic. The first hop is from interface **b** of all host m ports to interface **a** of all host m ports; with latency constraint of 64 cycles on NoC layer 0. The second hop is from interface **a** of all host m ports back to interface **b** of all host m ports;. **Because the alias hosts contains all hosts in the system, this traffic specification, when alias is expanded, will contain transactions from a host to itself; such transactions are omitted by NocStudio.**

Upon adding the multi-hop traffic transaction and mapping, NocStudio detects a protocol level deadlock and displays following message to the user.

**Below, reporting the detected cyclic dependency**

**host01/m0.a.in <- host00/m0.a.out <- host00/m0.a.in <- host01/m0.a.out <- host01/m0.a.in**

**Error: Protocol level deadlock found when mapping flow src: host01/m.a.out, dest: host00/m.b.in, qos: 0. Please correct it**

$ reset\_traffic

$ add\_traffic rates 0.025 0.01 hosts/m.a <-1 -1 4 64 0> hosts/m.a

$ add\_traffic rates 0.01 0.01 hosts/m.b <-1 -1 4 64 1> hosts/m.b

$ add\_traffic rates 0.01 0.01 host00/m.a host01/m.a <-1 -1 4 64 2> host10/m.b host11/m.b

$ map

Subsequently, the user can reset the traffic and add the first three traffic transactions again, which were mapped successfully. The resulting NocStudio display is shown below.

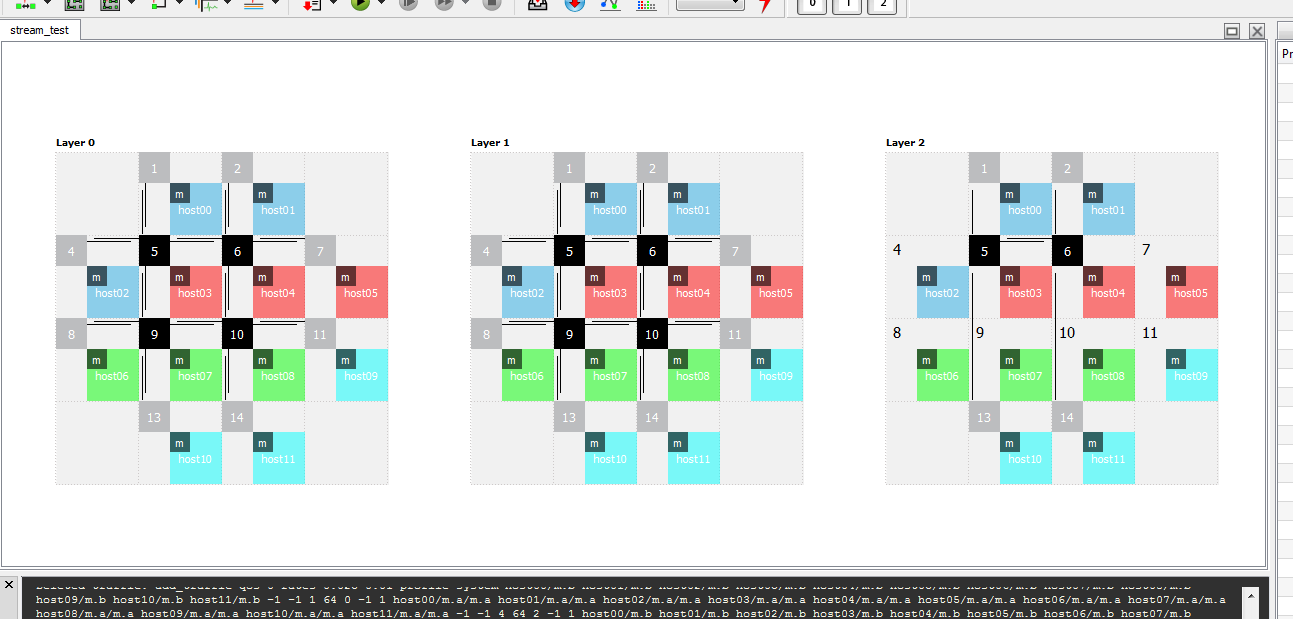


Figure 2: NoC display after adding traffic and mapping

$ tune\_route 1 10 d

$ show\_route hosts/m.\* host00/m.\* red

$ show\_route host06/m.\* host01/m.\* blue

$ grid off

Next, optimizations are carried out. First, route optimizations are performed using the **tune\_route** command. Next, the routes between all hosts’ m ports to the host00 m port are highlighted in red, and the routes from host06/m to host01/m are highlighted in blue using the **show\_route** command. The grid is also turned off for better visualization of routes. The resulting display is shown below.



Figure 3: NocStudio display highlighting certain routes in red and blue colors

$ tune\_place

$ map

Next, the position of various hosts is optimized using the **tune\_place** command; this should move the hosts that talk to each other more often closer to each other. In this example, most traffic is highly symmetric between all hosts; however, host00 and host01 communicate with host10 and host11 using an additional set of transactions in NoC layer 3, therefore these hosts are moved closer to each other when the cost function of the traffic is high enough. To illustrate, the rates between host00,01 and hosts10,11 were changed to 0.1. The resulting placement is below.



Figure 4: floorplan after tune\_place

$ analyze\_links p

$ tune\_links p

Next, link bandwidths are analyzed (for the original traffic spec). NocStudio prints the bandwidth of various links on the console and also highlights the link bandwidth in the display in various colors. Subsequently, the link bandwidths are optimized to meet the traffic requirements using the **tune\_links** command. The resulting NoC display after all these optimizations is shown below.

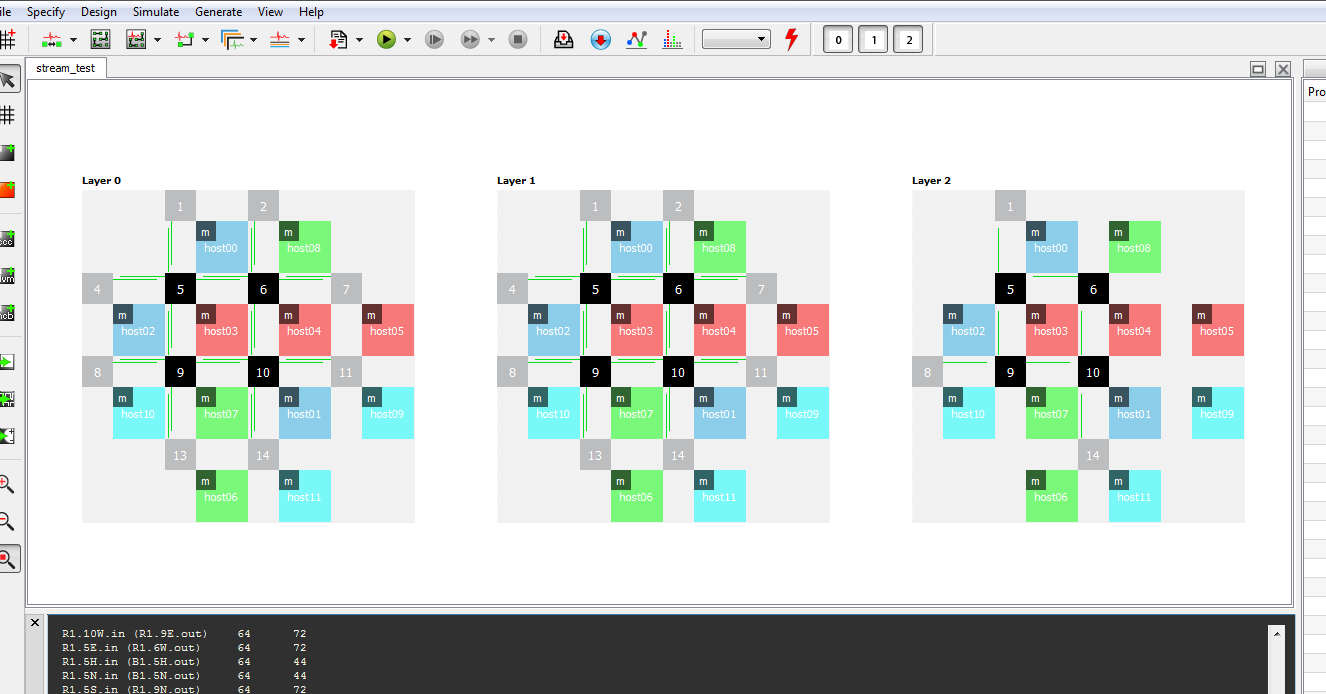


Figure 5: NocStudio display after link optimization

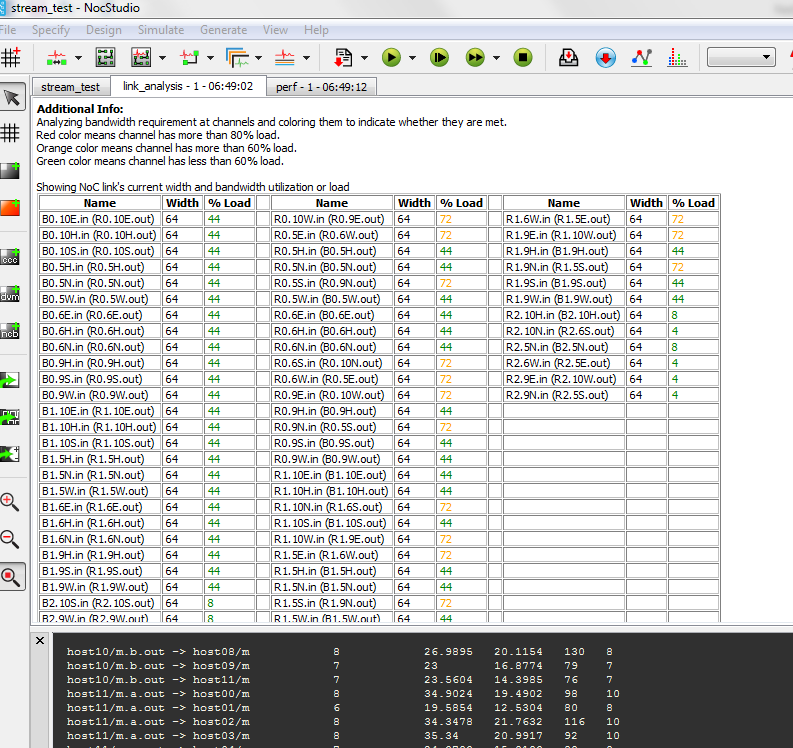


Figure 6 Results of link analysis showing loads

These windows are obtained by clicking o the link\_analysis button at the top right of the GUI screen. The window is persistent and it is possible to do further changes to the design and come back to compare with this data stored as a timestamped tab on the GUI.

$ create\_trace\_files avg profile system

$ run 1000 10000 trace

Next, in a single command, user warms up the network by simulating it for 1000 cycles, then clears the statistics and run for another 10000 cycles.

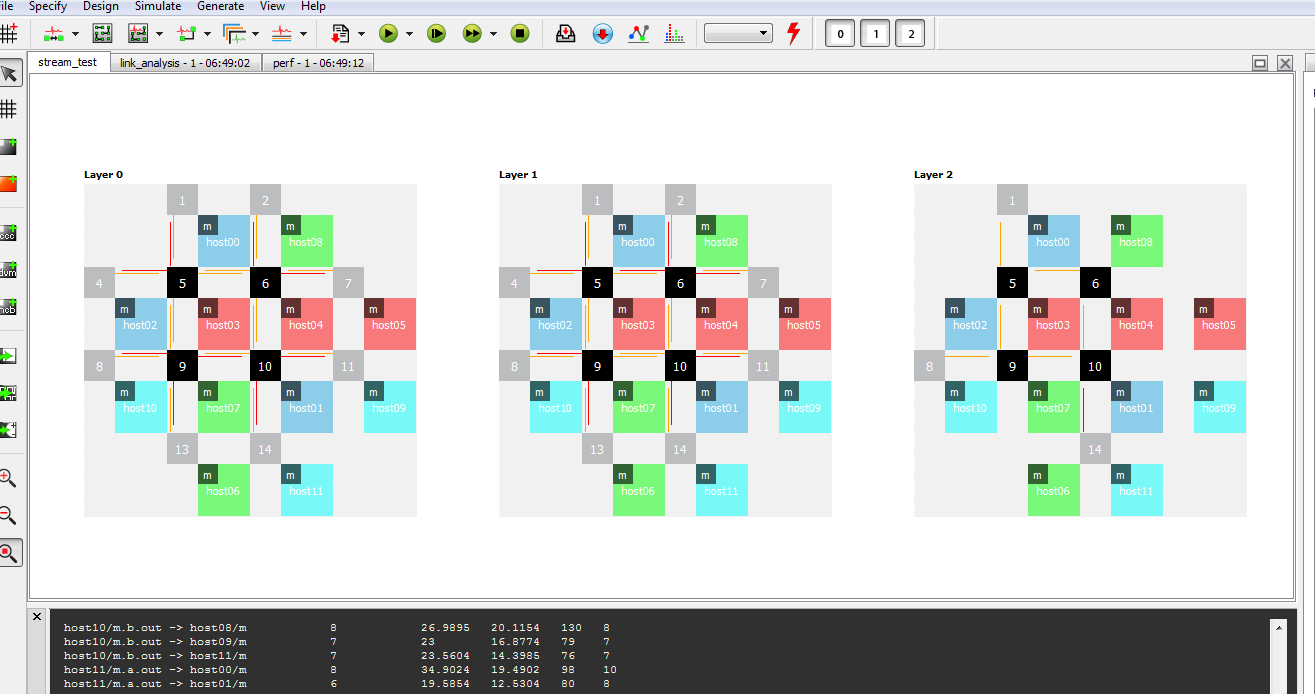
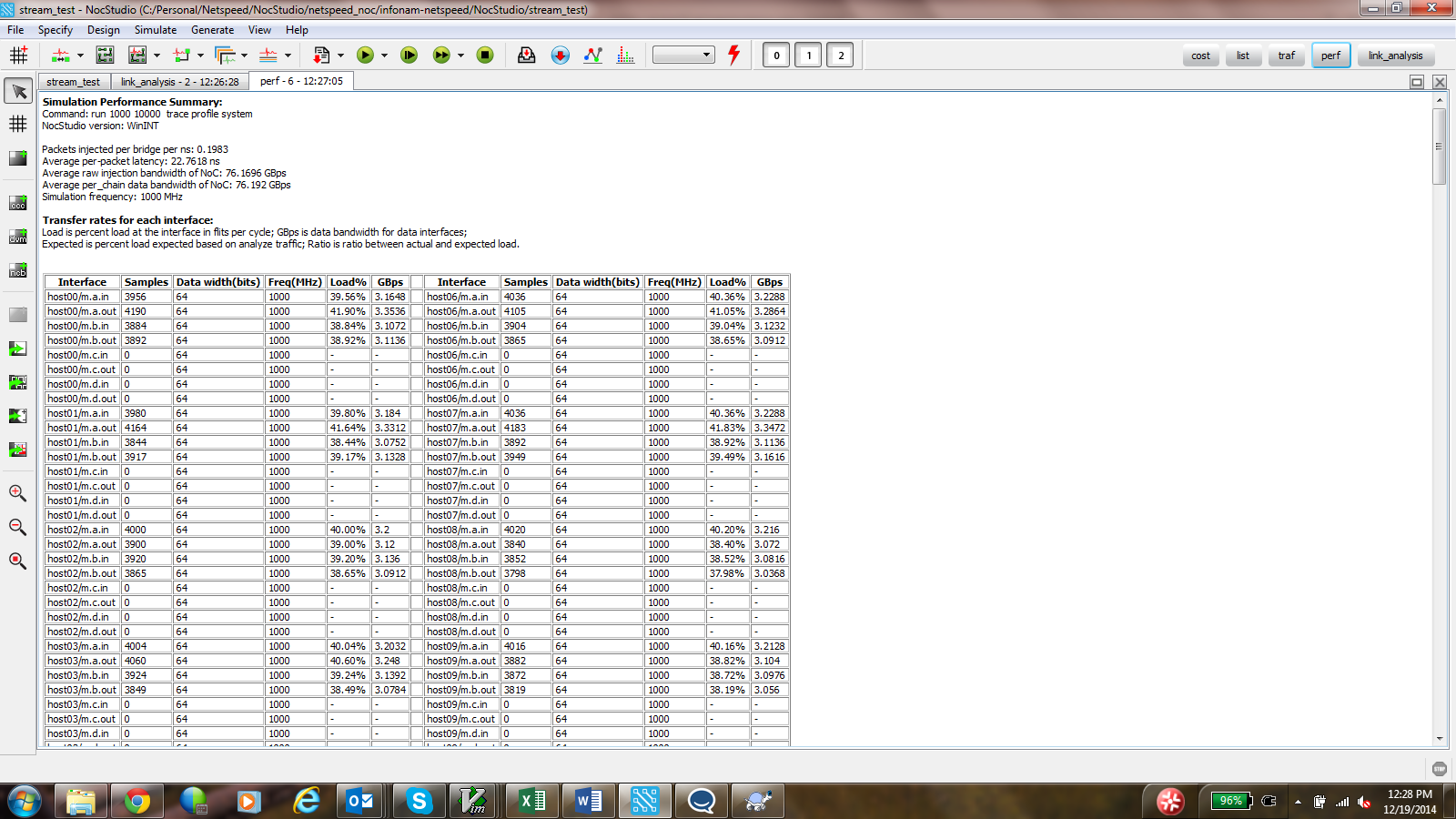


Figure 7: Link congestion shown after dynamic simulation

Detailed performance statistics are obtained by clicking on the ‘perf’ button.



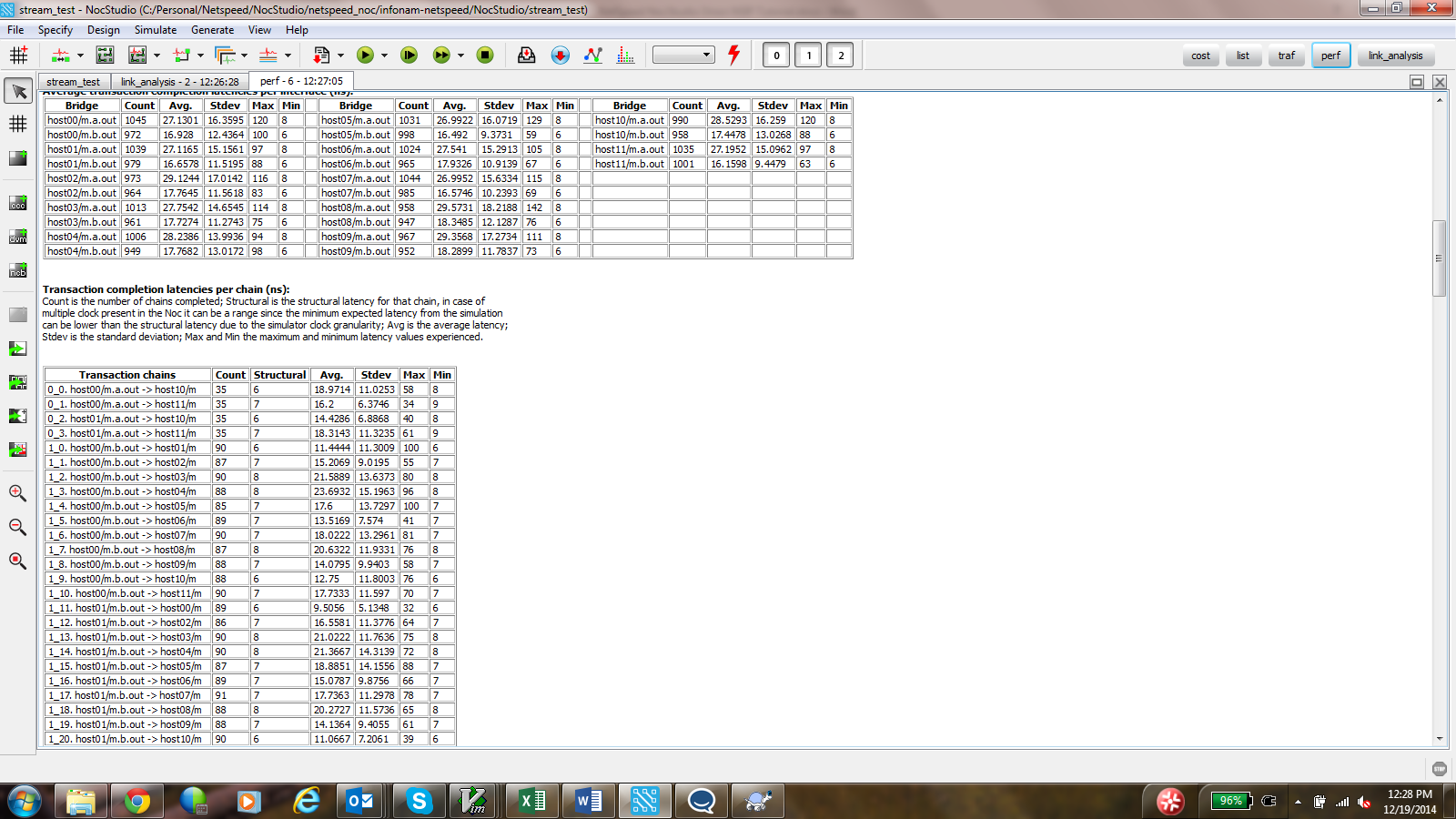


Figure 8: Performance Statistics after simulation

$ gen\_ip

Clicking on the gen-ip button generates the RTL and other IP files.

### Comparing with Map\_opt

The same configuration has been built with map\_opt below with a change to the traffic specifications, whereby explicit mapping of traffic to layers 0,1,2 has been removed.

$ add\_traffic rates 0.025 0.01 hosts/m.a <-1 -1 4 64 0> hosts/m.a

$ add\_traffic rates 0.01 0.01 hosts/m.b <-1 -1 4 64 1> hosts/m.b

$ add\_traffic rates 0.01 0.01 host00/m.a host01/m.a <-1 -1 4 64 2> host10/m.b host11/m.b

$ map\_opt 1 5 d

From the previous example, the network was seen to have been congested. In this example, map\_opt is given 8 layers to use, but ends up using 4. The resulting network is different, with traffic being spread around layers in a different way.

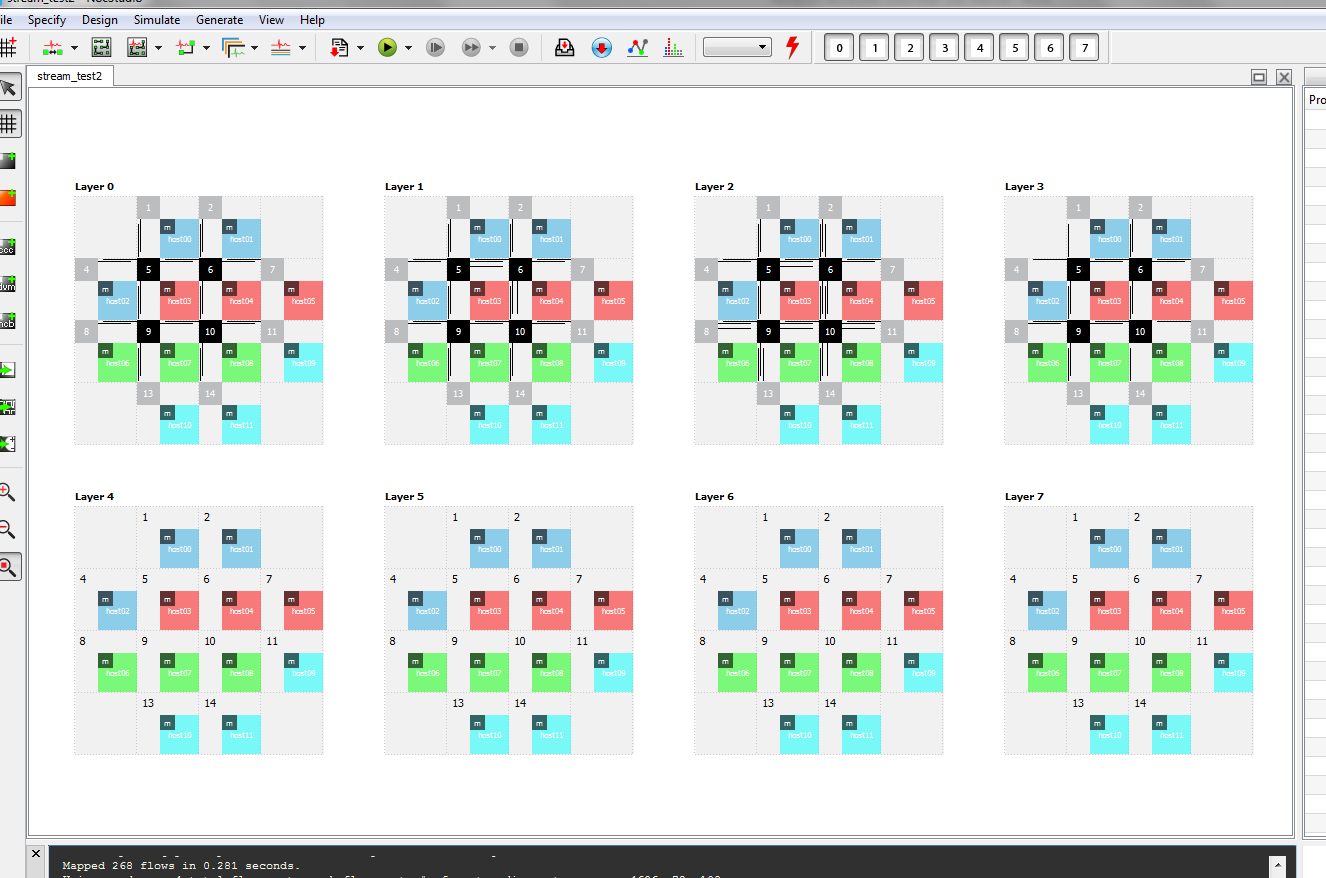


Figure 9 NoC generated using map\_opt

### Enabling Regbus

Regbus is a private layer used for accessing bridge and router registers. It has one single master device known as the Regbus Master which communicates to the regbus layer through a Regbus master bridge. The regbus master bridge speaks AXI-lite on the host side, and streaming protocol on the NoC side. Regbus is enabled by using the new\_mesh command as shown below:

$ prop\_default cell\_size 8

$ prop\_default data\_width 64

$ new\_mesh 4 4 3 stream\_test **regbus\_enabled**

$ mesh\_prop virtual\_ok yes

$ **gui on regbus**

Using the same example as before, but with one fewer host,

$ add\_host host00 color blue bridge m stream

$ add\_host host01 color blue bridge m stream

$ add\_host host02 color blue bridge m stream

$ add\_host host03 color blue bridge m stream

$ add\_host host04 color blue bridge m stream

$ add\_host host05 color blue bridge m stream

$ add\_host host06 color blue bridge m stream

$ add\_host host07 color blue bridge m stream

$ add\_host host08 color blue bridge m stream

$ add\_host host09 color blue bridge m stream

$ add\_host host10 color blue bridge m stream

$ add\_alias hosts host00 host01 host02 host03 host04 host05 host06 host07 host08 host09 host10

$ add\_traffic rates 0.025 0.01 hosts/m.a <-1 -1 4 64 0> hosts/m.a

$ add\_traffic rates 0.01 0.01 hosts/m.b <-1 -1 4 64 1> hosts/m.b

$ add\_traffic rates 0.01 0.01 host00/m.a host01/m.a <-1 -1 4 64 2> host09/m.b host10/m.b

$ map\_opt 1 5 d



Figure 10 Noc layers with regbus enabled

The figure above shows the Regbus layer in the rightmost panel. A regbus slave bridge is generated for each node that has a streaming bridge or router on it. The noc protocol used for regbus is the same for the other layers. However usage of the master bridge should be by a privileged AXI-lite host.

The regbus host can be moved to a given position on the grid by the following command sequence:

host\_prop rbm lock off

move\_host rbm 10

2670 Seely Avenue

Building 11

San Jose CA 95134

<www.netspeedsytems.com>