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

NocStudio Orion AMBA Tutorial

Revision History

About This Document

This document is a tutorial for NocStudio. 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](#_Toc406748514)

[Audience 2](#_Toc406748515)

[Related Documents 2](#_Toc406748516)

[Customer Support 2](#_Toc406748517)

[1 Tutorial and Example Session 4](#_Toc406748518)

[1.1 Starting NocStudio 4](#_Toc406748519)

[1.2 AXI-4 Tutorial-I and Example Interactive Session 4](#_Toc406748520)

[1.3 AMBA TUTORIAL – II 15](#_Toc406748521)

[1.3.1 System Specification : 15](#_Toc406748522)

[1.3.2 Traffic Flows 16](#_Toc406748523)

[1.3.3 Set up prop defaults and new\_mesh 16](#_Toc406748524)

[1.3.4 Adding Hosts, creating a Floorplan 17](#_Toc406748525)

[1.3.5 Bridge and Interface properties 18](#_Toc406748526)

[1.3.6 Clock Domain and Crossing set up 21](#_Toc406748527)

[1.3.7 Simulation Properties 22](#_Toc406748528)

[1.3.8 Creating the Address Map 23](#_Toc406748529)

[1.3.9 Adding traffic 24](#_Toc406748530)

[1.3.10 NoC Construction and Optimization 26](#_Toc406748531)

[1.3.11 Rebuilding the NoC with Optimizations 33](#_Toc406748532)

[1.4 Low Power Example 37](#_Toc406748533)

[1.4.1 Clock gating specification 37](#_Toc406748534)

[1.4.2 Power gating specification 38](#_Toc406748535)

[1.4.3 Creating the Power Aware NoC 40](#_Toc406748536)

# 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

## AXI-4 Tutorial-I and Example Interactive Session

$ new\_mesh 5 4 2 cache\_test

This command creates a new project directory named cache\_test to store all files related to the project. This command does not load an existing project (with the exception of traffic trace files) that may exist in that directory. This command clears any existing configuration and initializes a new mesh with 5 columns, 4 rows and 2 layers.

$ mesh\_prop

$ mesh\_prop virtual\_ok yes

The first command (**mesh\_prop**) is used to view the current properties of the mesh. The second command is used to instruct NocStudio that virtual positions are okay to use.

$ add\_host cpu0 color blue pos 0,1 bridge m axi4m

This **add\_host** command adds a new host called cpu0. This new host is colored blue, and is positioned at column 0, row 1. We give that host a single bridge called m (it is the bridge’s name) of type axi4m, or AXI-4 master.

$ add\_host cpu1 color blue pos 6 bridge m axi4m

$ add\_host cpu2 color blue pos 10 bridge m axi4m

$ add\_host cpu3 color blue pos 11 bridge m axi4m

$ add\_host mem0 color red pos 3 bridge s axi4s

$ add\_host mem1 color red pos 7 size 2 1 bridge s axi4s 7

These commands add three more cpu’s with single AXI-4 master bridges and two red mem’s with single AXI-4 slave bridges. In this set of commands, the position is specified using the grid cell number, instead of column and row. Notice that mem1 is a host of size two columns and 1 row (2x1), spanning two cells, 7 and 8, and with a bridge at cell 7.

# adding large host cache (can be added interactively in GUI too)

$ add\_host cache color red pos 12 size 2 1 bridge m axi4m 12h bridge s axi4s 13h

The first command above adds a cache controller named cache. This device has two bridges, so must have a size parameter larger than 1x1 to connect to at least two routers. For this example, the size is two columns and 1 row, or 2x1. The first bridge is called m and is a master; the second is called s and is a slave. It is necessary to specify the position of each bridge within a larger host. Large hosts may be added directly or through interactive mode; interactive mode will automatically start if bridges are not specified in the **add\_host** command. In interactive mode, the user will see a list of all bridges available for the specified host, and can enter the bridges one-by-one. After adding the bridges, the user can press Enter to exit interactive mode. The NocStudio display resulting from the commands so far is shown below, in Figure .

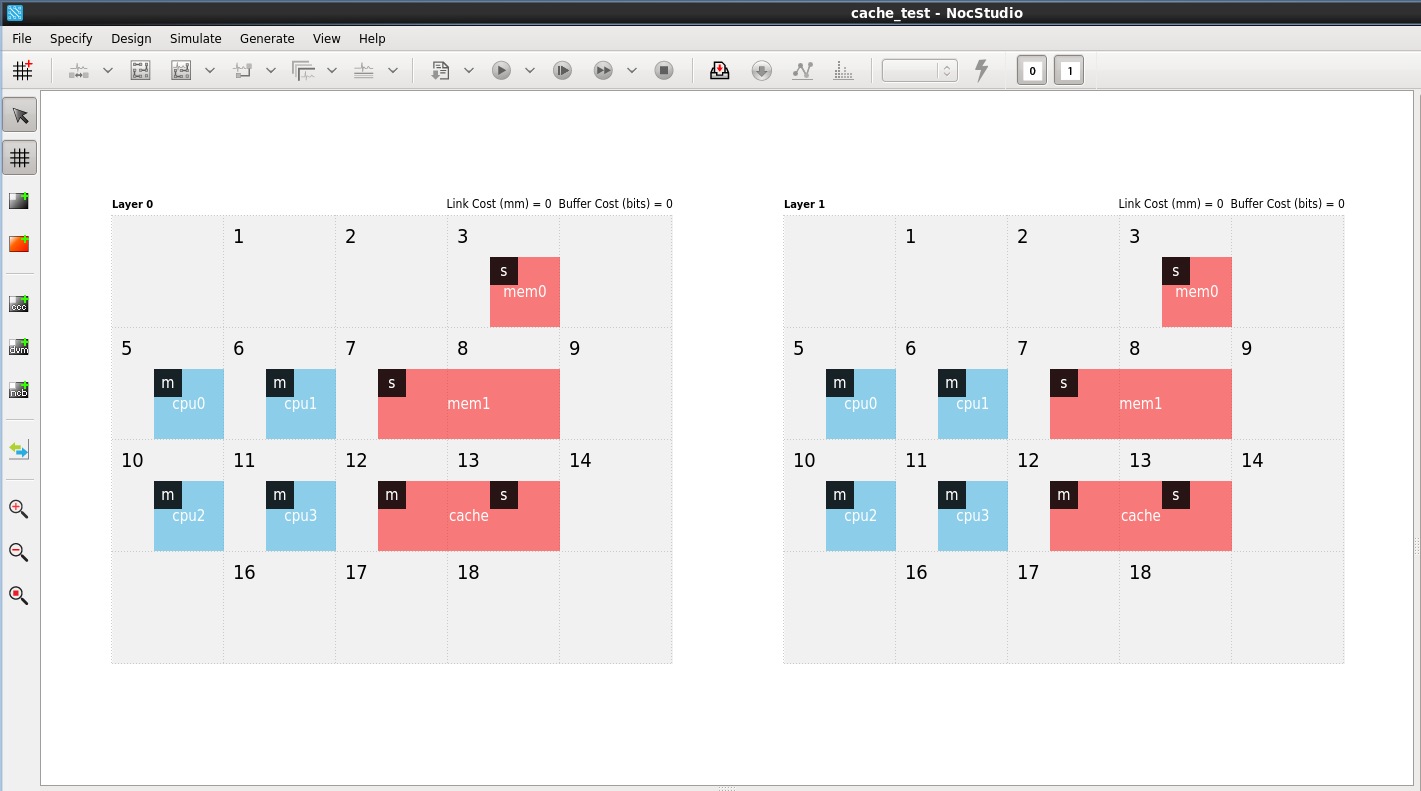


Figure 1 Tutorial 1: NocStudio display after adding all hosts

$ add\_alias cpu cpu0 cpu1 cpu2 cpu3

$ add\_alias mem mem0 mem1

To make the job of adding traffic and updating host properties easier, here user defines aliases for the CPUs and memories. It is possible to define aliases because the CPUs send to all memories equally and have similar design properties and traffic specifications. Also, the two memories are identical.

$ zoom 100

This sets the zoom level of the GUI to 100%. To adjust the size of the GUI display, increase or decrease the number; for example, set the zoom level to 150 for 50% larger display than usual, or set the zoom level to 75 for a 25% smaller display than usual.

// load hit

$ add\_traffic qos 1 cpu/m ar .1 .1 1 16 cache/s/s r .1 .1 4 16 cpu/m

This command creates a new traffic transaction between the CPUs and the cache. This transaction consists of two point-to-point AXI-4 transactions, or hops of the form <source> <message> [parameters] <destination>:

cpu/m ar .1 .1 1 16 cache/s

cache/s r .1 .1 4 16 cpu/m

The source argument is of the form <host name>/<bridge name>[/<bridge\_name>]. In the first hop each CPU master bridge, or cpu/m for short, sends an ar command to the slave bridge of the cache, or cache/s. The second hop must start from the destination of the first hop, same host, so the second hop’s source is indicated by the final /s of cache/s/s. This bridge replies back with a r message to the CPU’s master bridge and the transaction ends.

Each hop in the traffic transaction has up to 6 optional numeric properties in this format ([<avg rate> [<peak rate> [<# of beats> [<latency> [<NoC layer> [<VC id>]]]]]]). The first is the average bandwidth of that flow; it is a floating point value describing messages per cycle. Second is the maximum bandwidth of the flow, in the same units. Third is the number of beats and fourth is the desired latency for each hop in number of cycles. For this example, both hops have average and peak transmit rate of 0.1 messages per cycle, number of beats of 1 and 4 respectively, and latency target of 16 cycles for each hop message.

The fifth and sixth optional parameters are NoC layer id for the message, and the VC id for the message. With these two, users may force a message to go on a particular NoC layer and a particular VC in that layer. A VC can be specified only when a NoC layer is specified. By default (if not specified) ar and b are mapped to NoC layer 0 (left), and r and aww are mapped to NoC layer 1 (right) by NocStudio. The VC is automatically allocated by NocStudio based on the traffic specifications, priority and QoS requirements. Average and peak message rates may be specified for the entire transaction as well.

// load miss

$ add\_traffic qos 1 rates .01 .01 cpu/m ar cache/s/m ar mem/s/s r cache/m/s r cpu/m

This command adds a four-hop traffic sequence in which cpu/m sends a ar transaction to cache/s; subsequently cache/m sends a ar transaction to mem/s, and then mem/s sends a r response to cache/m and then cache/s sends a r response back to cpu/m. In this example, the **rates** command is used to specify average and peak rates of .1 and .1 for all hops in the traffic transaction. All unspecified properties will be chosen based on the default values of NocStudio. NocStudio defaults may be viewed or updated using the **prop\_default** command.

#write hit

$ add\_traffic qos 1 rates .01 .01 cpu/m aww cache/s

#write miss allocate

$ add\_traffic qos 1 cpu/m aww .1 .1 4 16 cache/s/m ar mem/s

#write back

$ add\_traffic qos 1 cache/m aww mem/s/s b cache/m

These commands create more traffic related to the write commands. In these commands, a short traffic description is used and only the request messages of multi-hop AXI-4 transactions are specified in the commands; the corresponding response hop messages of the request hops of the transaction would be automatically inferred by NocStudio. The write miss allocate traffic shows a mix of default and specified parameters for the hops in the transaction.

$ map

We then instruct the system to map the traffic flows to network channels. This is done automatically by NocStudio, based on the host positions and the routes between them, so as to avoid all deadlock scenarios and maintain isolation and QoS priority properties. The NocStudio display resulting from the commands so far is shown below in Figure .

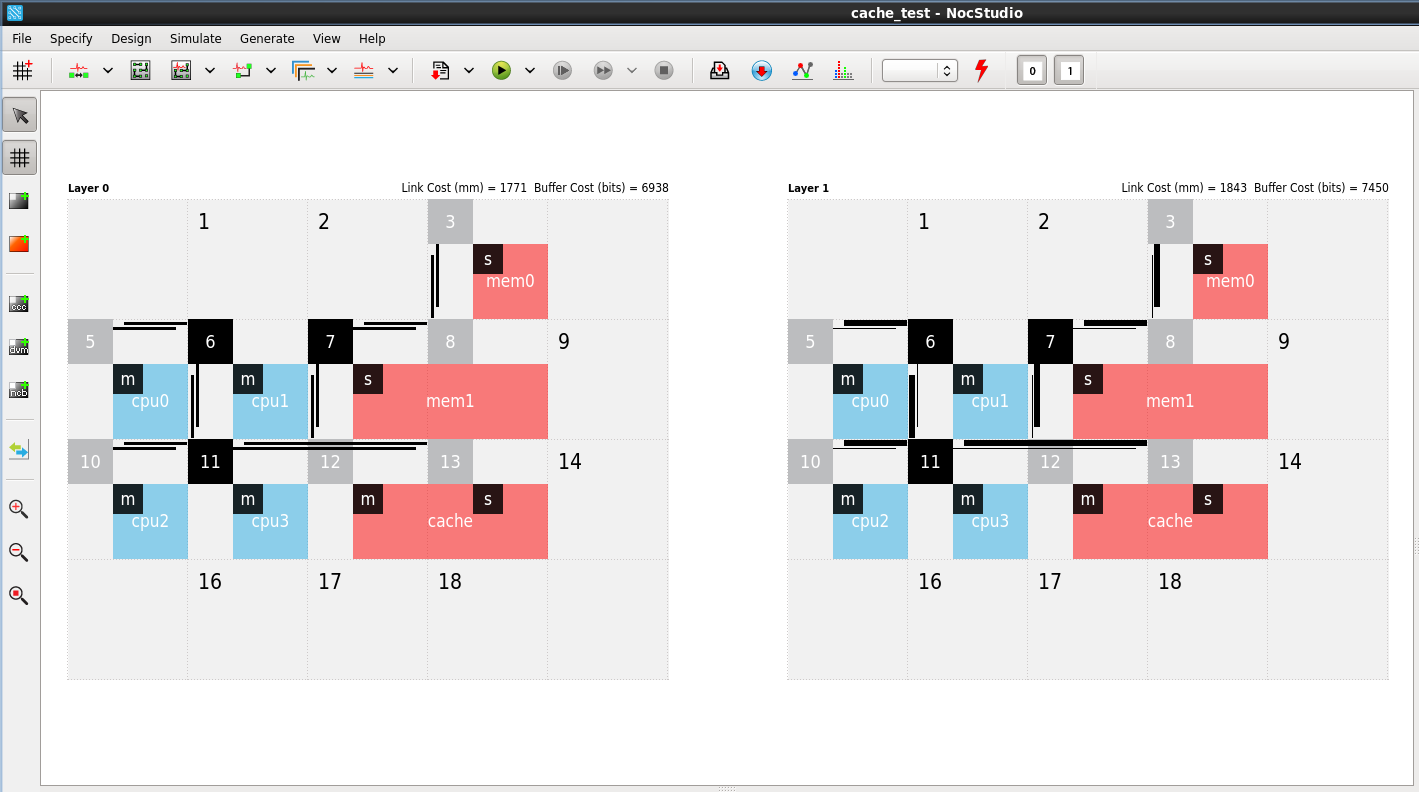


Figure 2 Tutorial 1: NocStudio display after adding traffic and map

$ create\_trace\_files avg

After adding all the traffic, user can auto-generate trace files for every AXI-4 master bridge that describes the pattern for loads and stores at the master bridge. By default, all transactions generated by a master are added to its trace file. These files are created in the project directory named “cache\_test” and can be edited; the edited version will be loaded when the simulation starts. In this example, trace files are created based on the average rate of traffic transaction specification. Trace file generation capability is fairly sophisticated in NocStudio; average, or peak rate of traffic can be used to generate trace files that will represent the behavior of the traffic specification. An additional randomization option may be used to further randomize the generated trace file. The command **create\_trace\_files** can be used to import external traffic and generate trace files based on that traffic; this allows one to generate trace files for a variety of traffic specifications which may be different from the one specified to NocStudio, and to perform simulations under these conditions.

$ run 20 trace

$ run 20 100 trace

The first command starts a simulation session, running it for 20 cycles. The next command restarts the simulation and runs for 20 cycles of warm-up period, and then continues for another 100 cycles during which performance stats are collected.

$ cont 20

We now continue the simulation for another 20 cycles.

$ sim\_stats perf1.csv

$ reset\_stats

Now user stores the performance statistics in perf1.csv file; this will contain the performance statistics for 120 cycles with 20 additional cycles of warm-up period. Then user resets the statistics in order to collect new sets of statistics.

$ cont 120

$ sim\_stats perf2.csv

Next, user runs the simulation for 120 more cycles and stores the next set of performance statistics in perf2.csv file. During the performance simulation, the NoC channels get colored to highlight the congestion level in the NoC channels. The NocStudio display resulting from the commands so far is shown below, in Figure 3.

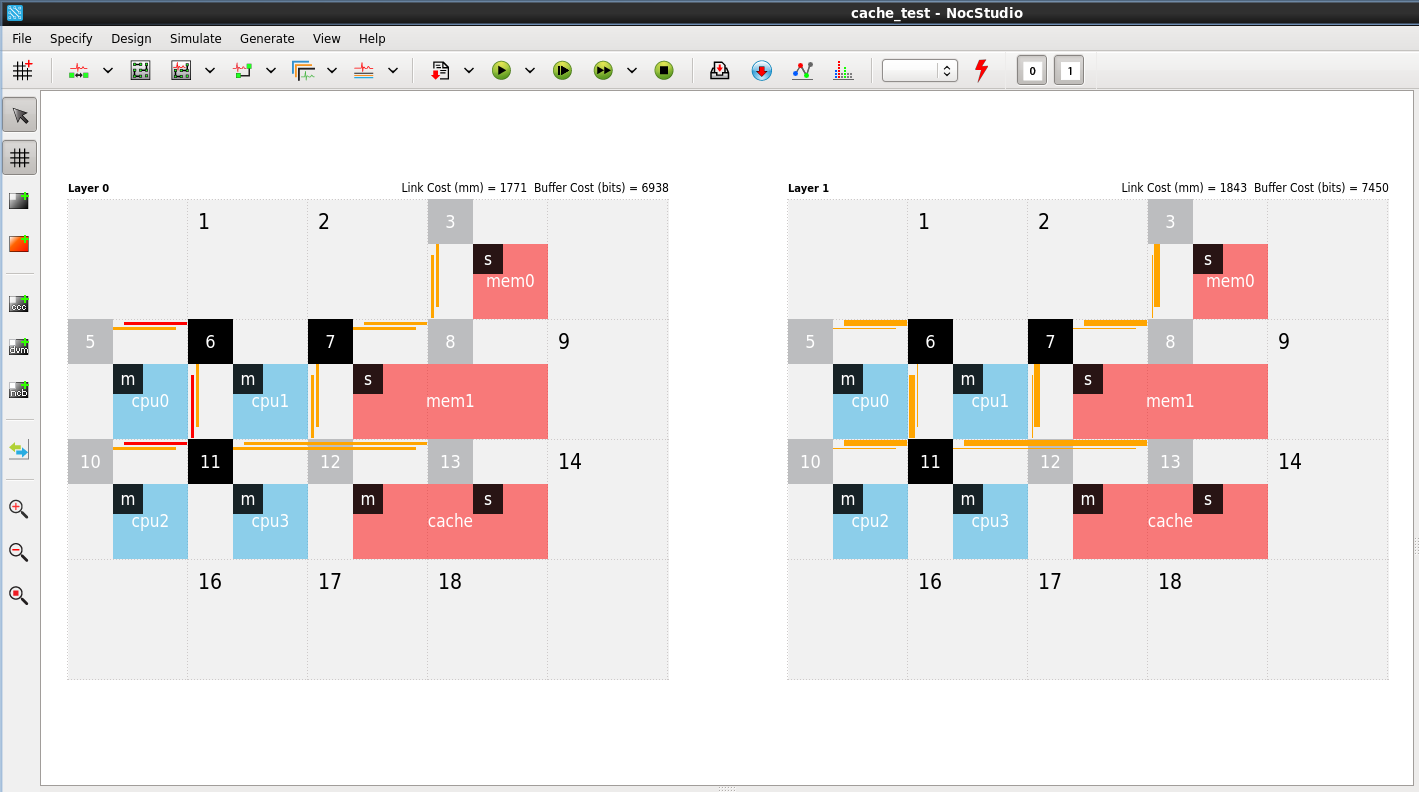


Figure 3 Tutorial 1: NocStudio display after running performance simulation

$ finalize

Finally, user finalizes the simulation (during this step, traffic stimulus is paused and simulation is continued until all outstanding messages in the NoC are drained). It is not necessary to do this for the statistics or for continuing, it simply allows us to watch the progress of the network’s buffers emptying.

$ tune\_place

This command will try to improve the performance of the network by improving placement of various hosts in the grid. Because NocStudio uses a randomized algorithm, there may be variation in the results from run to run; a seed may be used to enforce deterministic results. The updated placement after running this command is shown in Figure . Constraints on the positions of various hosts may be specified before running **tune\_place**.

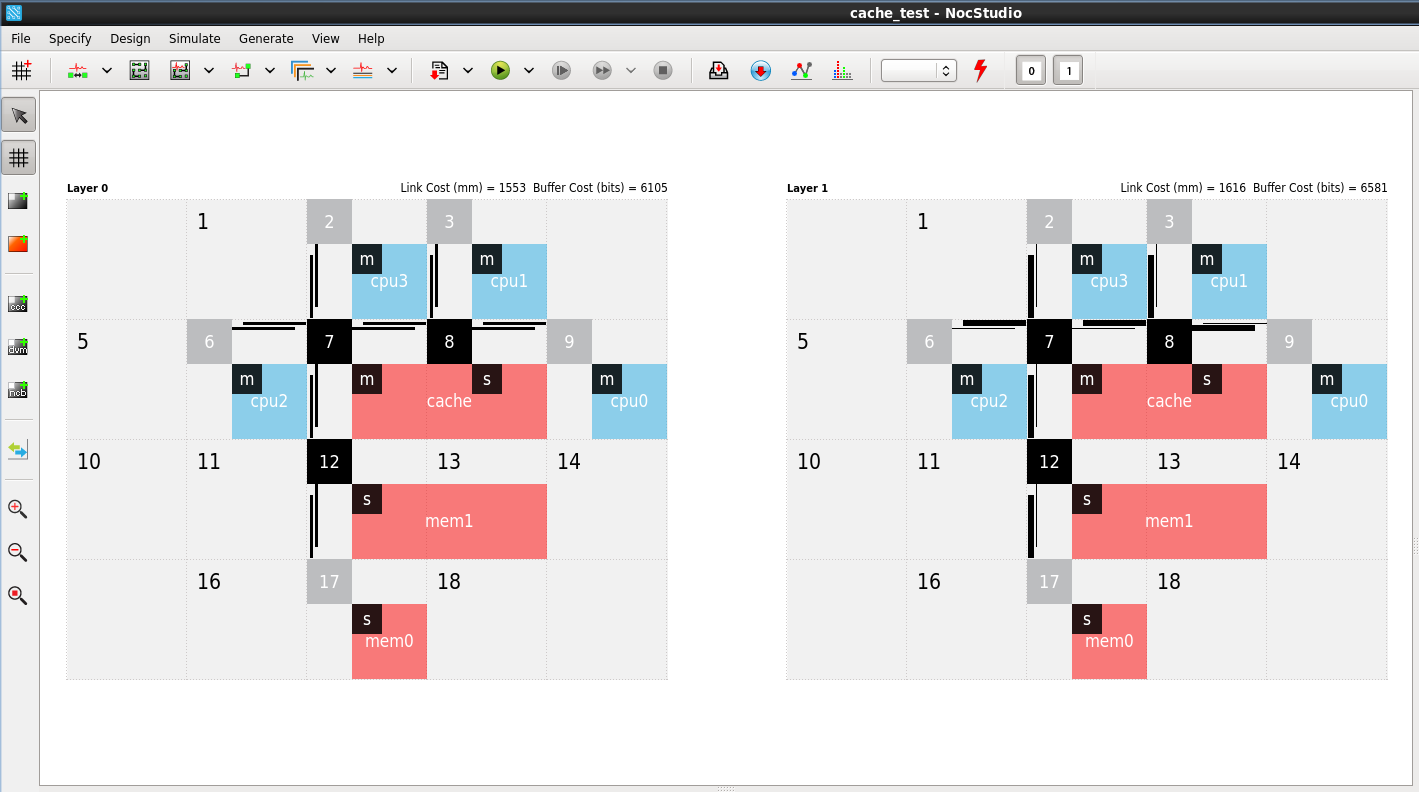


Figure 4 Tutorial 1: NocStudio display after running tune\_place

$ map

We must remap network channels after tuning the placement of hosts.

$ tune\_links peak

Next, user optimizes the link widths so that the network is guaranteed not to be a bottleneck. This algorithm can be run in three different modes: peak, average or balanced. In this example, peak mode is run, which uses peak bandwidth values to provision link bandwidth. This algorithm is not randomized and therefore, will always produce the same optimal results. The width a channel is represented by the width of the lines in the NocStudio display. The line widths in the display may become asymmetric after this optimization reflecting the different bandwidths allocated at various channels. The NocStudio display resulting from the commands so far is shown below in Figure .

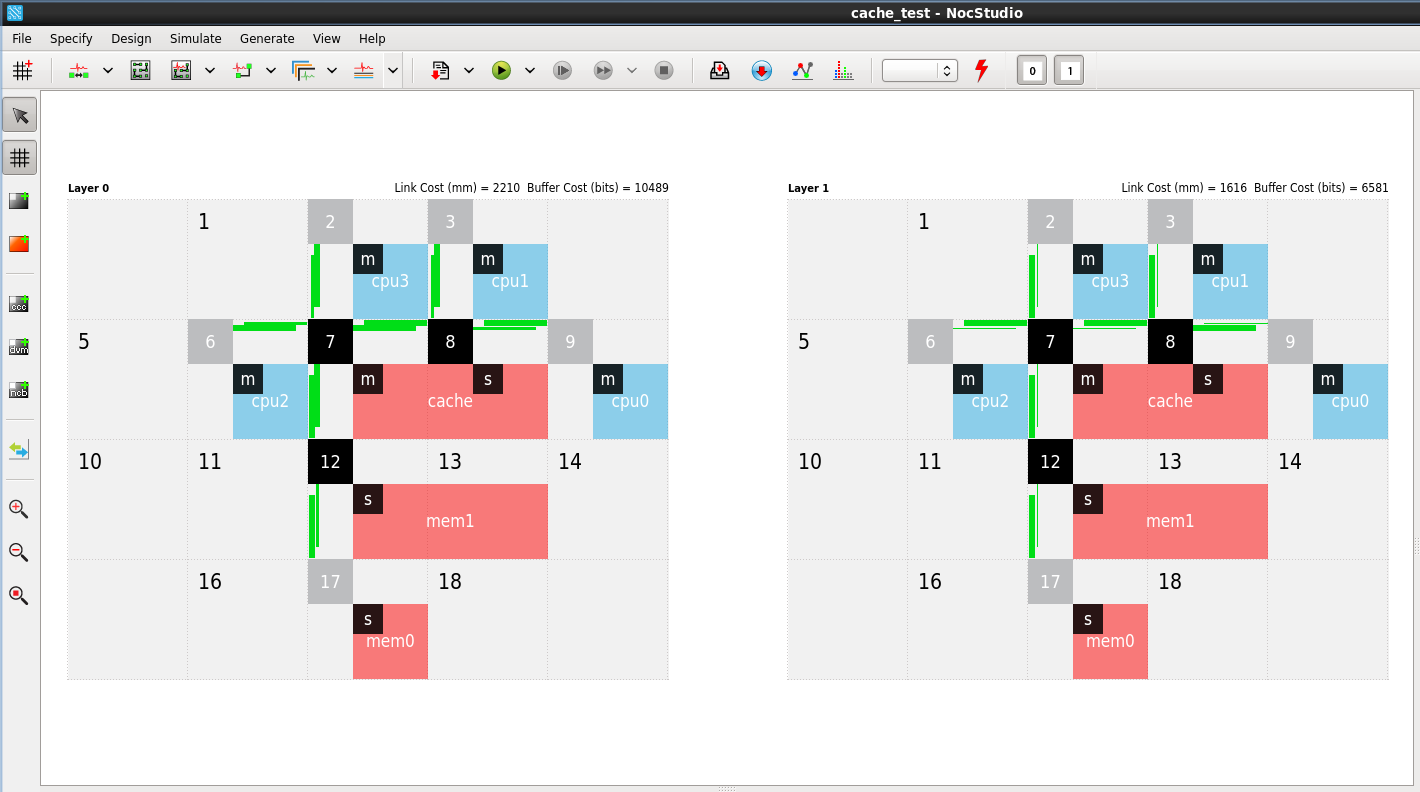


Figure 5 Tutorial 1: NocStudio display after tune\_links

$ run 40 160 trace

In a single command, user warms up the network by simulating it for 40 cycles; then clears the statistics and run for another 160 cycles.

$ sim\_stats perf-post-tune.csv

After this simulation, user can output the performance characteristics of this optimized network.

$ tune\_route packed

$ tune\_links balanced

Next, user optimizes the routes of the various transaction messages; NocStudio will optimize the routes for various transaction messages and attempt to balance the load on various channels to improve overall cost and performance. Two modes are supported: packed and balanced; user picks packed mode. Optionally, an initial seed value and attempt count can be provided to this command. In this example, the default seed and attempt count are used.

After route optimization, link optimization needs to be run again, so user runs it, but in balanced mode this time. In balanced mode, link bandwidths are provisioned so that the total average bandwidth requirement of all simultaneously active flows on a link is met, and the peak bandwidth requirement of each flow individually is met.

$ show\_route \* \* white

$ show\_route cpu cache red

$ show\_route cpu cache green

$ show\_route mem cache blue

The optimized routes can be highlighted in the NocStudio display using these commands. First, all routes are made white. Next, the route from cpu to cache for **ar** message is highlighted in red; the route from cpu to cache for **aww** message is highlighted in green; the route from mem to cache for all messages is highlighted in blue.

$ gen\_ip

Next, the RTL and other files of the current NoC IP is generated in the project folder.

## AMBA TUTORIAL – II

### System Specification:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| BLK | Freq (Mhz) | Init/  Targ | Ports | Add (bits) | Data (bits) | Address Range | Comments |
| CPU | 500 | I | AXI4 | 40 | 128 |  | Data port |
|  |  | I | AXI4 |  | 32 |  | Configuration / IO |
| DISP | 400 | I | AXI3 | 32 | 64 |  | Data port |
|  |  | T | AHBL |  | 32 | 0x91000000-0x91ffffff 0 | Config register |
| USB | 400 | I | AHBL | 32 | 32 |  | Data port |
|  |  | T | AHBL |  | 32 | 0x92000000-0x92ffffff 0 | Config register |
| DDR | 400 | T | AXI4 | 32 | 128 | 0x0-0x7fffffff (64B-intrlv) | Data port |
|  |  | T | AXI4 |  | 128 | 0x0-0x7fffffff (64B-intrlv) | Data port |
|  |  | T | APB |  | 32 | 0x90000000-0x90ffffff | Config register |
| SRAM | 400 | T | AHBL | 32 | 32 | 0x80000000-0x80ffffff | Data port |
|  |  | T | AHBL |  | 32 | 0x93000000-0x93ffffff 0 | Config register |
| APBBR | 200 | T | APB | 32 | 32 | 0x94000000-0x94ffffff 1 | Low BW slave |
|  |  | T | APB | 32 | 32 | 0x95000000-0x95ffffff 2 | Low BW slave |
|  |  | T | APB | 32 | 32 | 0x96000000-0x96ffffff 3 | Config registers |

### Traffic Flows

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Initiator | Target | Read BW (GBps) | Write BW (GBps) | Comments |
| CPU | DDR | 1.0 | 1.0 |  |
| CPU | SRAM | 0.5 | 0.5 |  |
| CPU | REGS / APB | small | Small |  |
| DISPLAY | DDR | 1.0 | 0 |  |
| USB | DDR | 0.2 | 0.2 |  |
| USB | SRAM | 0.1 | 0.1 |  |
|  |  |  |  |  |

### Set up prop defaults and new\_mesh

prop\_default data\_width 32

prop\_default axi4\_addr\_width 32

new\_mesh 6 6 2 simple-example regbus\_enabled

set\_clock\_domain\_freq noc 500 #set noc\_clock at 500Mhz

set\_clock\_domain\_freq regbus 200 #set regbus\_clock at 200Mhz

mesh\_prop wire\_delay\_mm\_per\_ns 4

mesh\_prop grid\_cell\_size\_mm 4

# Can have two grids between pipeline stages

mesh\_prop extra\_bandwidth\_provisioning 25

Regbus – the private register access and debug network is set up in this example. Also note the frequency domain of the NoC is set up to match CPU, and the pipe-lining and bandwidth headroom parameters.

### Adding Hosts, creating a Floorplan

add\_host cpu color blue pos 4,2 size 1 2 bridge m1 axi4m bridge m0 axi4m

add\_host usb color orange pos 1,3 size 2 1 bridge m ahblm 2,3H bridge cfg ahbls slaves 1

add\_host disp color orange pos 1,4 size 2 1 bridge m axi3m 2,4H bridge cfg ahbls slaves 1

add\_host ddr color red pos 1,1 size 3 1 bridge s0 axi4s 2,2N bridge s1 axi4s 3,2N bridge cfg apb 1,1H slaves 1

add\_host sram color black pos 3,3 size 1 2 bridge s ahbls 3,3H slaves 1 bridge cfg ahbls 4,4W slaves 1

add\_host apbbr color pink pos 4,4 size 1 1 bridge s apb slaves 3

host\_prop rbm lock no

move\_host rbm 4,1

The resulting floorplan is below :

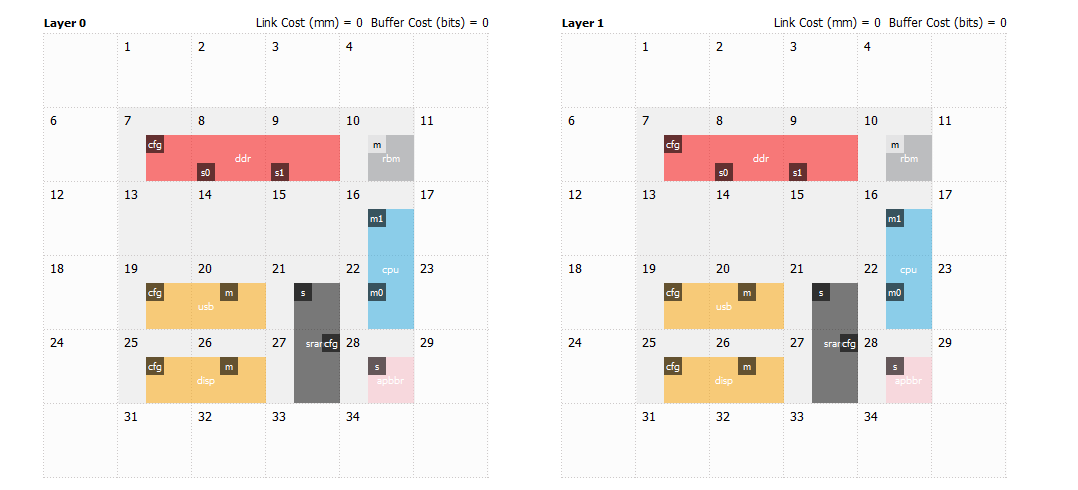


Figure 6 Tutorial 2: Floorplan

The regbus master has been moved to a real node. The CPU, Sram and DDR are kept closer together. A channel has been created where the bulk of the traffic flow is expected. 3 APB slaves are created on the single APB bridge, taking a single NoC port.

### Bridge and Interface properties

ifce\_prop cpu/m0.aww data\_width 128

ifce\_prop cpu/m0.r data\_width 128

ifce\_prop cpu/m1.aww data\_width 32

ifce\_prop cpu/m1.r data\_width 32

bridge\_prop cpu/m0 axi4m\_ar\_reorder\_enable yes

bridge\_prop cpu/m0 axi4m\_aw\_reorder\_enable yes

bridge\_prop cpu/m0 axi4\_ar\_aid\_width 4

bridge\_prop cpu/m0 axi4\_aw\_aid\_width 4

bridge\_prop cpu/m1 axi4m\_ar\_reorder\_enable no

bridge\_prop cpu/m1 axi4m\_aw\_reorder\_enable no

bridge\_prop cpu/m1 axi4\_ar\_aid\_width 4

bridge\_prop cpu/m1 axi4\_aw\_aid\_width 4

bridge\_prop cpu/m0 axi4\_addr\_width 40

bridge\_prop cpu/m1 axi4\_addr\_width 40

ifce\_prop cpu/m0.aww max\_outstanding\_requests 16

ifce\_prop cpu/m0.ar max\_outstanding\_requests 16

ifce\_prop cpu/m1.aww max\_outstanding\_requests 2

ifce\_prop cpu/m1.ar max\_outstanding\_requests 2

#### CPU

These are CPU Bridge and interface properties. Note how the port used for configuration (m1) differ from the port used for memory access (m0). Besides the data width, the higher performance m0 port is configured to have 16 multiple accesses outstanding with reordering enabled.

#### Display & USB

ifce\_prop disp/m.r data\_width 64

ifce\_prop disp/m.aww data\_width 64

ifce\_prop disp/m.ar max\_outstanding\_requests 8

bridge\_prop disp/m axi4m\_ar\_reorder\_enable yes

bridge\_prop disp/m axi4\_ar\_aid\_width 4

ifce\_prop usb/m.aww max\_outstanding\_requests 8

ifce\_prop usb/m.ar max\_outstanding\_requests 1

bridge\_prop usb/m ahblm\_force\_nonbufferable\_enable yes

Properties for the other masters (Display and USB) are specified above. Since USB has no changes in data or address widths from the default, the only statements required are to do with how many transactions to have outstanding. **An interesting feature of the AHBL master bridge is that it can force all write transactions to have early responses** **through the *ahblm\_force\_nonbufferable\_enable*** **parameter**. Performance is therefore improved though software on the CPU must guarantee that ordering can be maintained where needed.

#### DDR

ifce\_prop ddr/s0.aww data\_width 128

ifce\_prop ddr/s0.r data\_width 128

ifce\_prop ddr/s1.aww data\_width 128

ifce\_prop ddr/s1.r data\_width 128

bridge\_prop ddr/s0 axi4\_ar\_aid\_width 4

bridge\_prop ddr/s0 axi4\_aw\_aid\_width 4

bridge\_prop ddr/s1 axi4\_ar\_aid\_width 4

bridge\_prop ddr/s1 axi4\_aw\_aid\_width 4

ifce\_prop ddr/s0.aww max\_outstanding\_requests 16

ifce\_prop ddr/s0.ar max\_outstanding\_requests 16

ifce\_prop ddr/s1.aww max\_outstanding\_requests 16

ifce\_prop ddr/s1.ar max\_outstanding\_requests 16

ifce\_prop ddr/cfg.aww max\_outstanding\_requests 2

ifce\_prop ddr/cfg.ar max\_outstanding\_requests 2

The DDR slave bridges issue transactions from the NoC into the controller, and therefore need to track the outstanding transactions. Depending on the bandwidth requirements, the number of outstanding transactions to be tracked here can be large, since the large latency to memory must be covered. Typically the number of transactions buffered in the dram controller must be equal to the number of outstanding transactions programmed into the slave bridge.

#### APB

The 3 APB slaves of 3 different APB version types are on the single bridge. They are all 32b wide for both address and data.

ifce\_prop apbbr/s.apb0.in apb\_slave\_version 2

ifce\_prop apbbr/s.apb1.in apb\_slave\_version 3

ifce\_prop apbbr/s.apb2.in apb\_slave\_version 4

### Clock Domain and Crossing set up

Clock domains can be set up and visualized quite nicely in NocStudio. During the mesh setup stage, we had created the regbus (200Mhz) and the noc clock (500Mhz) domains. This sets up the domains for the other hosts. Async clock crossings are also created in the bridge at the host interface.

add\_clock\_domain apbbr 200

add\_clock\_domain other 400

bridge\_prop disp/\* clock\_cross async

bridge\_prop disp/\* clock\_domain\_host other

bridge\_prop usb/\* clock\_cross async

bridge\_prop usb/\* clock\_domain\_host other

bridge\_prop ddr/\* clock\_cross async

bridge\_prop ddr/\* clock\_domain\_host other

bridge\_prop sram/\* clock\_cross async

bridge\_prop sram/\* clock\_domain\_host other

bridge\_prop apbbr/\* clock\_cross async

bridge\_prop apbbr/\* clock\_domain\_host apbbr

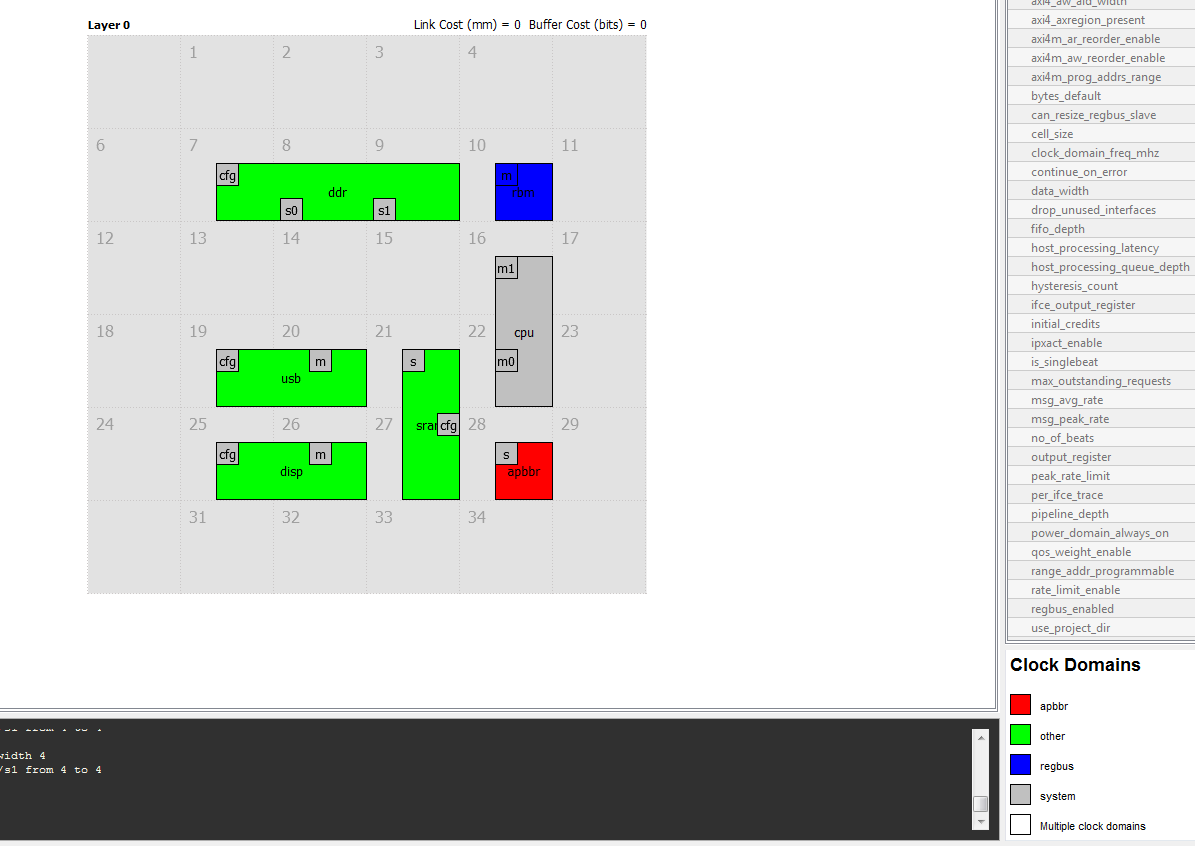


Figure 7 Tutorial 2: Clock domains

### Simulation Properties

Simulation properties do not directly affect NoC construction. However they do influence evaluation of the NoC dynamic behaviour during performance simulation.

For example the properties set below add a constant latency between messages received on the DDR read (ar) ports to their response port ( r). Similarly for writes. The numbers expressed are in cycles or dram clock – in this case the ‘other’ clock domain set to 400Mhz.

When transaction delays are reported, these 50 cycles are included in the structural latency of the path. From a NoC simulation perspective this allows us to see the effects of host latencies on NoC congestion. It also influences decisions on how many transactions to have outstanding, fifo depths etc.

set\_host\_processing\_latency ddr/s0.ar ddr/s0.r 50

set\_host\_processing\_latency ddr/s0.aww ddr/s0.b 50

set\_host\_processing\_latency ddr/s1.ar ddr/s1.r 50

set\_host\_processing\_latency ddr/s1.aww ddr/s1.b 50

set\_host\_processing\_latency sram/s.ar sram/s.r 5

set\_host\_processing\_latency sram/s.aww sram/s.b 5

### Creating the Address Map

NocStudio has two powerful methods to specify address ranges, and secure access regions for these addresses. The base:mask method is good for interleaved ranges, and the lo-hi method works well for contiguous ranges. In this example interleaved and contiguous ranges are demonstrated, and further information on creating secure ranges, read and write only range, etc are available on the NocStudio help menu.

Once the address ranges are created, they can be selectively added to each master using the add\_range\_to\_master command. If not specified, NocStudio will by default add all ranges to all masters.

add\_range ddr/s0 DDR\_S0 0x00000000:0xfff0000040 #adds range from 0x0-0x7fffffff

add\_range ddr/s1 DDR\_S1 0x00000040:0xfff0000040 #interleaved at 64B boundary

## The above uses the base : mask method. Other ranges below uses lo-hi

add\_range sram/s SRAM 0x80000000-0x80ffffff 0

add\_range ddr/cfg DDR\_REGS 0x90000000-0x90ffffff 0

add\_range disp/cfg DISP 0x91000000-0x91ffffff 0

add\_range usb/cfg USB 0x92000000-0x92ffffff 0

add\_range sram/cfg SRAM\_REGS 0x93000000-0x93ffffff 0

add\_range apbbr/s APB\_SLV0 0x94000000-0x94ffffff 0

add\_range apbbr/s APB\_SLV1 0x95000000-0x95ffffff 1

add\_range apbbr/s APB\_SLV2 0x96000000-0x96ffffff 2

add\_range\_to\_master cpu/m0 DDR\_S0 DDR\_S1 SRAM

add\_range\_to\_master cpu/m1 DDR\_REGS SRAM\_REGS DISP USB APB\_SLV0 APB\_SLV1 APB\_SLV2

add\_range\_to\_master usb/m DDR\_S0 DDR\_S1 SRAM

add\_range\_to\_master disp/m DDR\_S0 DDR\_S1

### Adding traffic

The final portion of the specification is the addition of traffic. The add\_traffic\_b statement is used here to create average traffic flows as in the system specification. Peak traffic is specified as a higher number, as we plan to build the network using ‘balanced mode’. Some headroom for a host hitting peak traffic while others are at average is thus built in. The DISPLAY unit has been given a higher QoS value than the others – as the isochronous nature of the traffic requires a higher priority. As the display traffic is not large enough to swamp out everything else, it is safe to give it this higher fixed priority. It is also worth mentioning that the configuration register access flows are specified to be 4B sized transfers, as this is likely to be the common case. The larger flows to memory are specified at a 64B block size. 64B is overwhelmingly appearing as the cache line size of choice in processors. In addition, with DDR3 burst length limitations a 64B size is also natural to Dram. An alias command is used to efficiently specify all register traffic in a single traffic statement.

add\_alias allcfgports ddr sram usb disp #alias for system config traffic

#DISPLAY TRAFFIC : read only to dram

add\_traffic\_b qos 1 disp/m.ar bw 0.5 peak 1 bytes 64 ddr/s0.ar

add\_traffic\_b qos 1 disp/m.ar bw 0.5 peak 1 bytes 64 ddr/s1.ar

#CPU TRAFFIC :

#to memory

add\_traffic\_b qos 0 cpu/m0.ar bw 0.5 peak 1 bytes 64 ddr/s0.ar

add\_traffic\_b qos 0 cpu/m0.aww bw 0.5 peak 1 bytes 64 ddr/s0.aww

add\_traffic\_b qos 0 cpu/m0.ar bw 0.5 peak 1 bytes 64 ddr/s1.ar

add\_traffic\_b qos 0 cpu/m0.aww bw 0.5 peak 1 bytes 64 ddr/s1.aww

#to sram memory

add\_traffic\_b qos 0 cpu/m0.ar bw 0.5 peak 1 bytes 64 sram/s.ar

add\_traffic\_b qos 0 cpu/m0.aww bw 0.5 peak 1 bytes 64 sram/s.aww

#to reg configuration, all register sized accesses (32b)

add\_traffic\_b qos 0 cpu/m1.ar bw 0.01 peak 0.05 bytes 4 allcfgports/cfg.ar

add\_traffic\_b qos 0 cpu/m1.aww bw 0.01 peak 0.05 bytes 4 allcfgports/cfg.aww

#to APB slaves, all 32b accesses

add\_traffic\_b qos 0 cpu/m1.ar bw 0.01 peak 0.05 bytes 4 apbbr/s.ar

add\_traffic\_b qos 0 cpu/m1.aww bw 0.01 peak 0.05 bytes 4 apbbr/s.aww

#USB TRAFFIC :

#to memory

add\_traffic\_b qos 0 usb/m.ar bw 0.1 peak 0.25 bytes 64 ddr/s0.ar

add\_traffic\_b qos 0 usb/m.aww bw 0.1 peak 0.25 bytes 64 ddr/s0.aww

add\_traffic\_b qos 0 usb/m.ar bw 0.1 peak 0.25 bytes 64 ddr/s1.ar

add\_traffic\_b qos 0 usb/m.aww bw 0.1 peak 0.25 bytes 64 ddr/s1.aww

#to sram memory

add\_traffic\_b qos 0 usb/m.ar bw 0.1 peak 0.25 bytes 64 sram/s.ar

add\_traffic\_b qos 0 usb/m.aww bw 0.1 peak 0.25 bytes 64 sram/s.aww

### NoC Construction and Optimization

We use the following two commands to build the NoC

map\_opt 1 20 d b

tune\_links b

This uses balanced mode optimization with the distributed options. Together with the 25% extra bandwidth headroom, it’s expected to provide sufficient capacity to handle any dynamic effects. The resultant NoC is pictured below, along with the Regbus layer. NocStudio is able to map the network using a single layer and the second layer is completely unused.

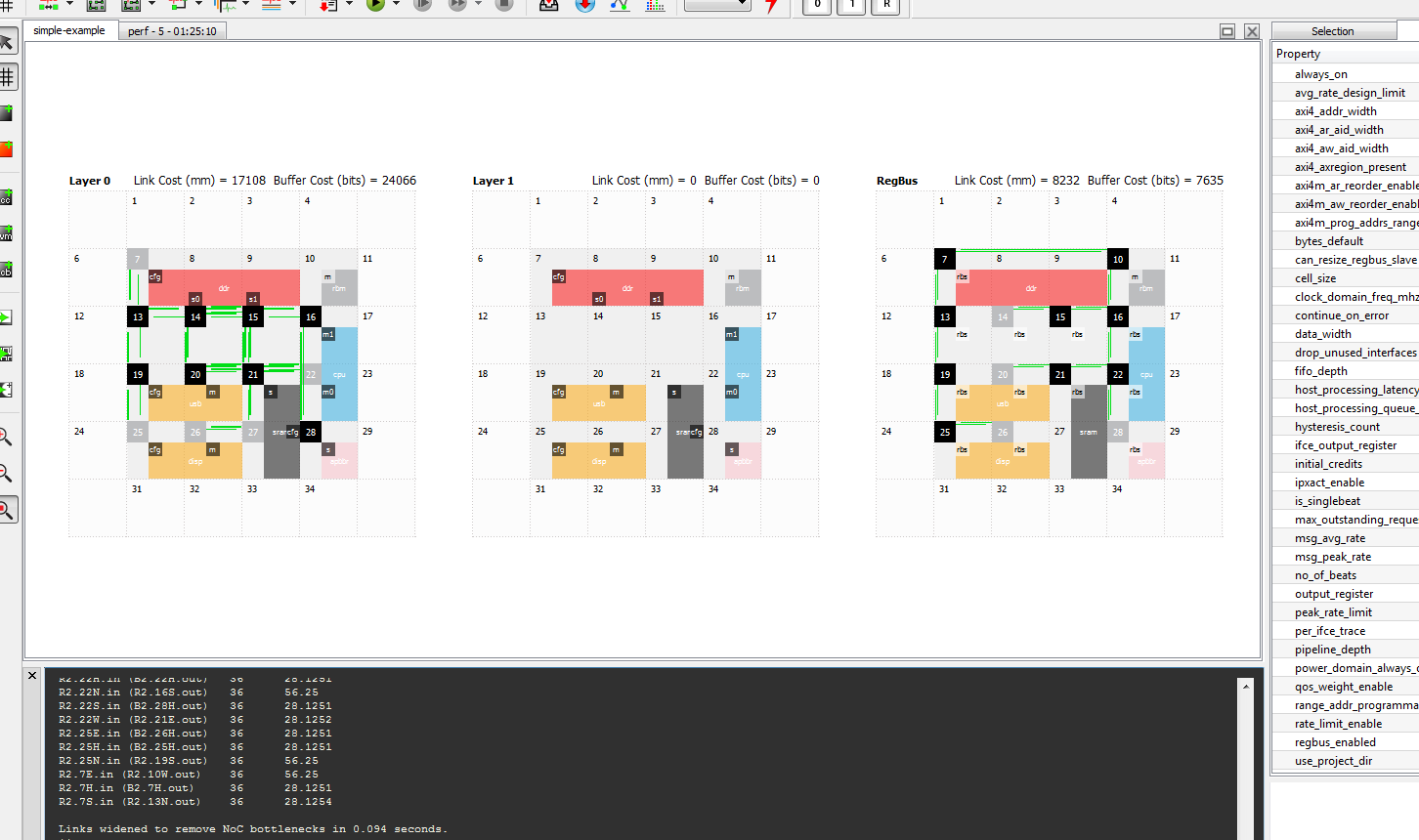


Figure 8: Regbus and Primary layers after map\_opt , tune\_links

Routers which appear gray in colour are skipped; a router may have been needed in the position, but NocStudio has found a way to optimize it away. In case the resulting link after skipping the router spans more than two grids, a pipeline stage is inserted and fifos are sized up to maintain throughput. This is both a latency and area optimization.

A clock domain view of the various layers, by clicking on the ‘clock visualization’button on the left margin of the NocStudio GUI is also interesting. The entire Regbus layer runs on a slow clock which is asynchronous to the primary NoC clock.

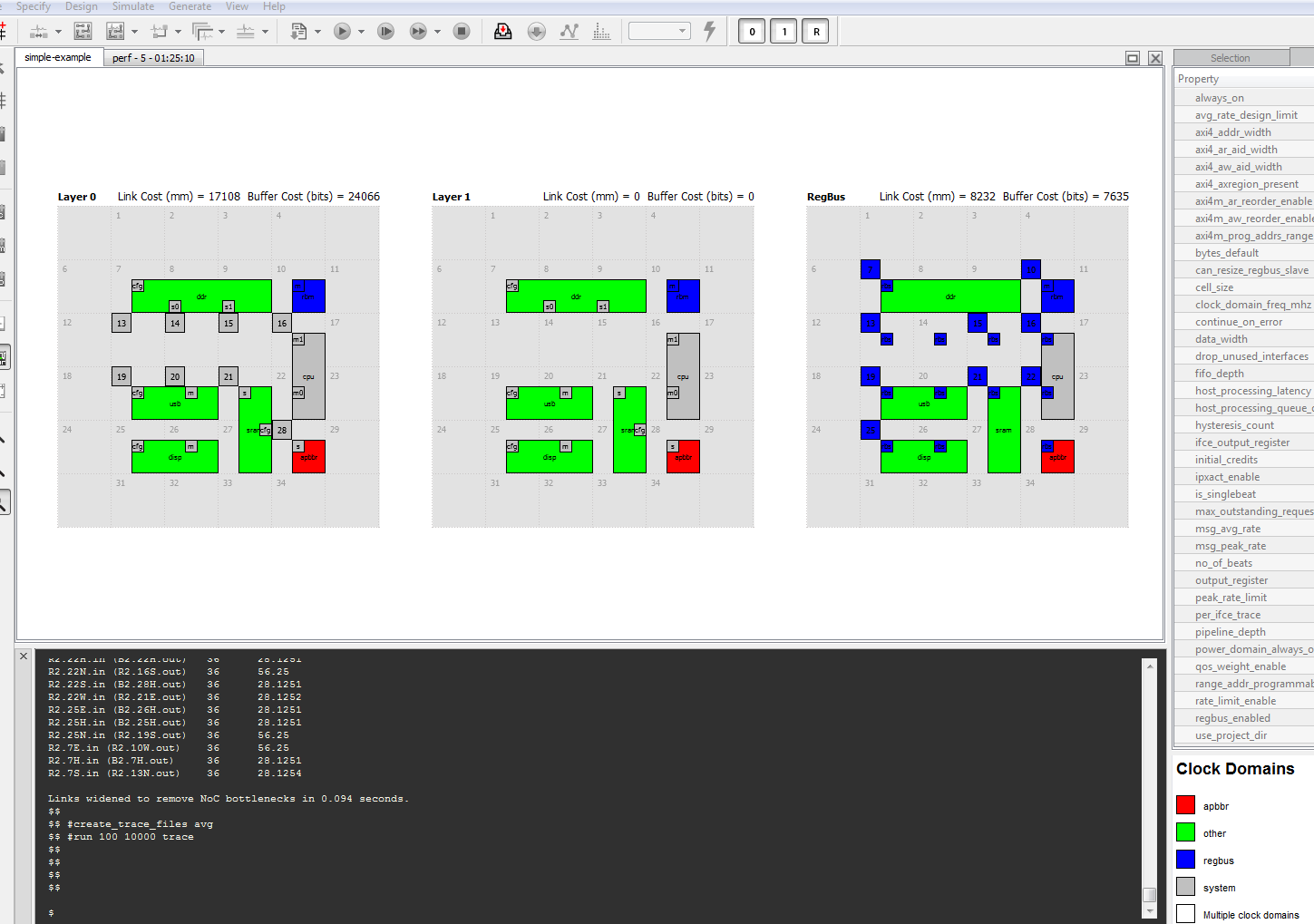


Figure 9: Clock Domain view after mapping

Clicking on the “link analysis” tab on the top right of the GUI brings up a tab which contains the load on each link. This is for the analysis done in balanced mode. The tab is persistent, and has the timestamp of generation. After further optimization is done in the NocStudio session, new results can be compared with the previous ones in a convenient manner.

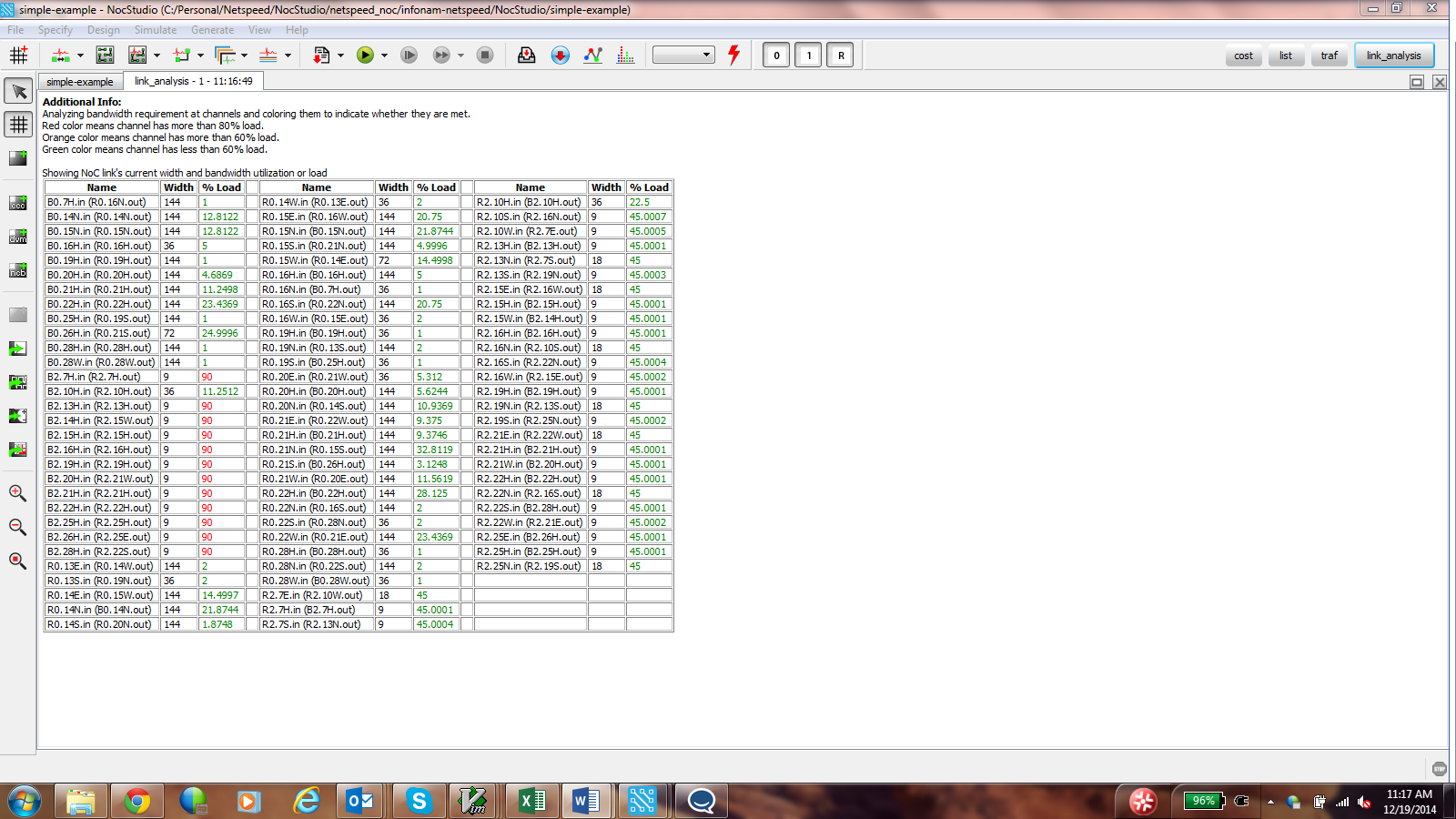


Figure 10: Output of Link load analysis in balanced mode

A relevant observation from the table above is that several links are wide but quite lightly loaded. These are likely to be the register configuration links, which are latency insensitive but sized to the interface size.

**Optimization opportunity #1: Register config buses are candidates for further area reduction using the ‘downsizing’ option in NocStudio.**

#### Performance Simulation

Before doing any optimizations is is wise to look at the dynamic performance simulation results from the NocStudio integrated simulator. The default profile tag ‘system’ is used here.

analyze\_traffic \* system \* # always use profile tag

create\_trace\_files avg profile system # always use profile tag

run 100 10000 trace

NocStudio creates traces from the add\_traffic statements and stores them in the: ./simpler-ex-axi/trace directory.

The results expected from simulation can be obtained by using the ‘analyze\_traffic’ command. This shows the result of add traffic : looking at the ‘Data’ column shows the traffic spec is as expected. NocStudio will flag warnings if the specified traffic exceeds the interface capacity.

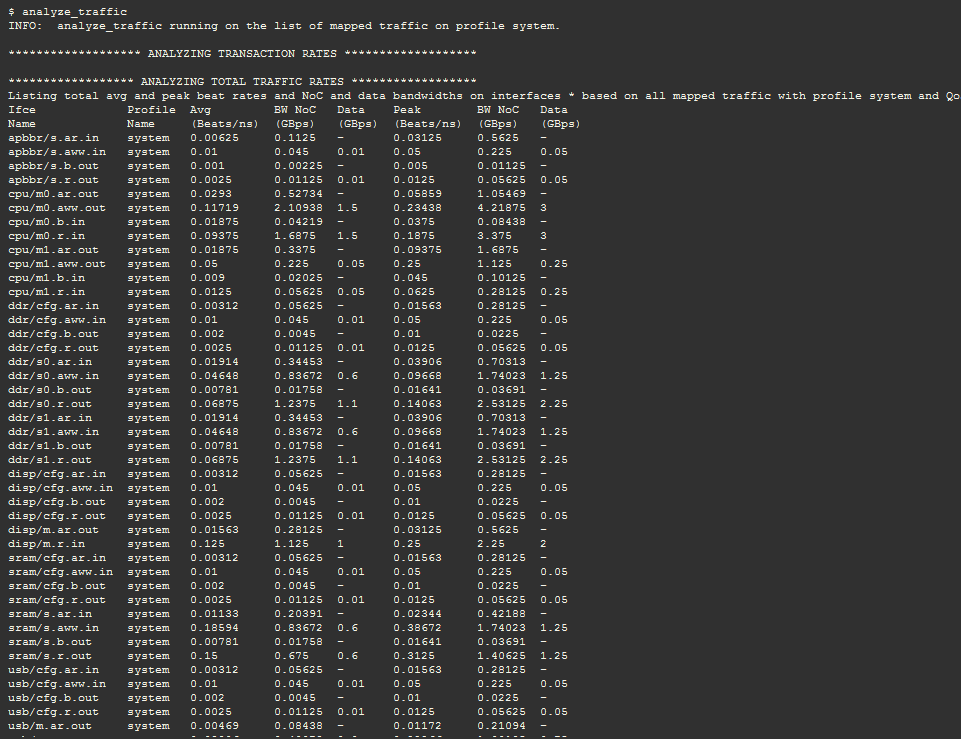


Figure 11: Checking the Traffic Spec: analyze\_traffic

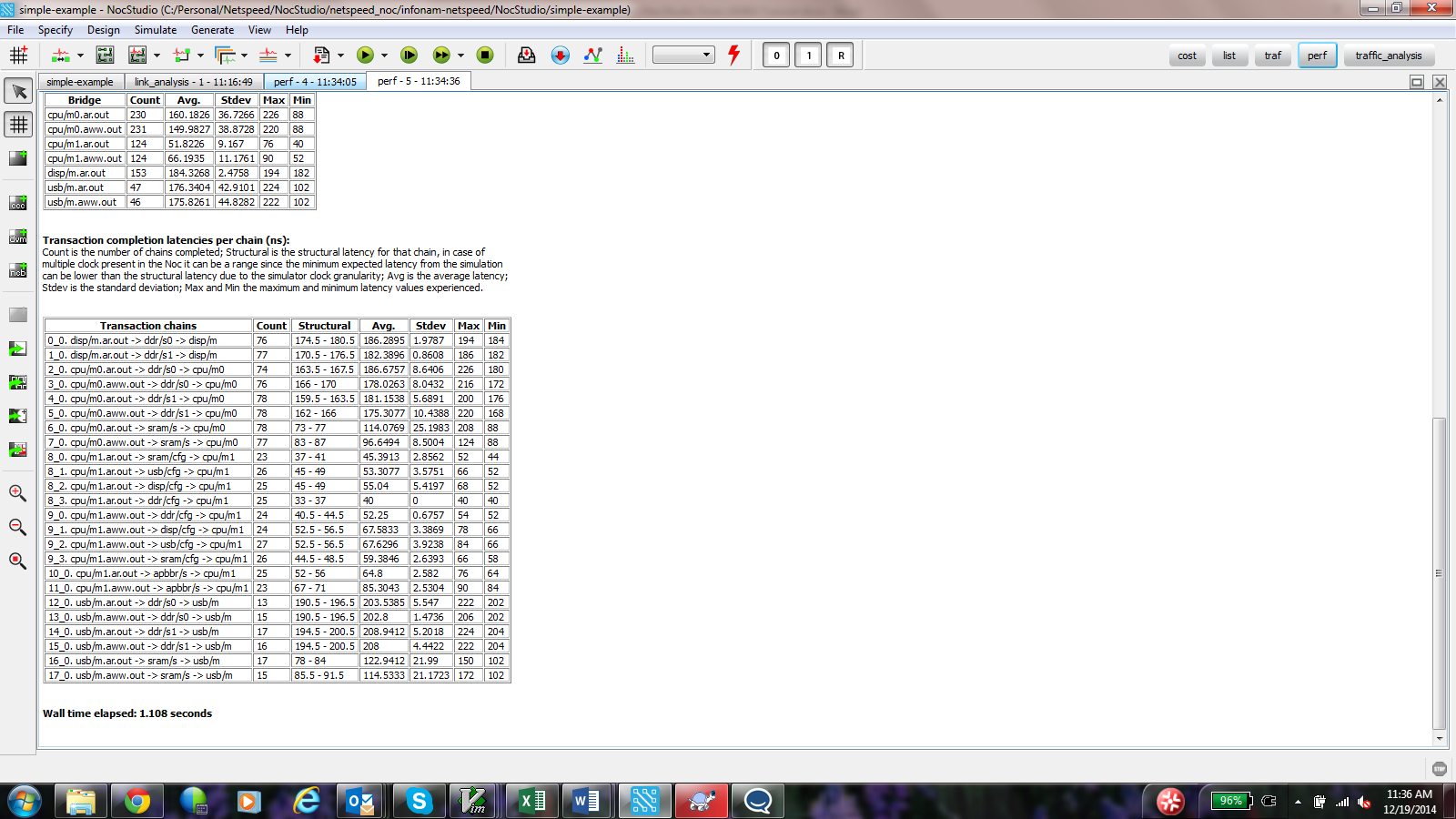
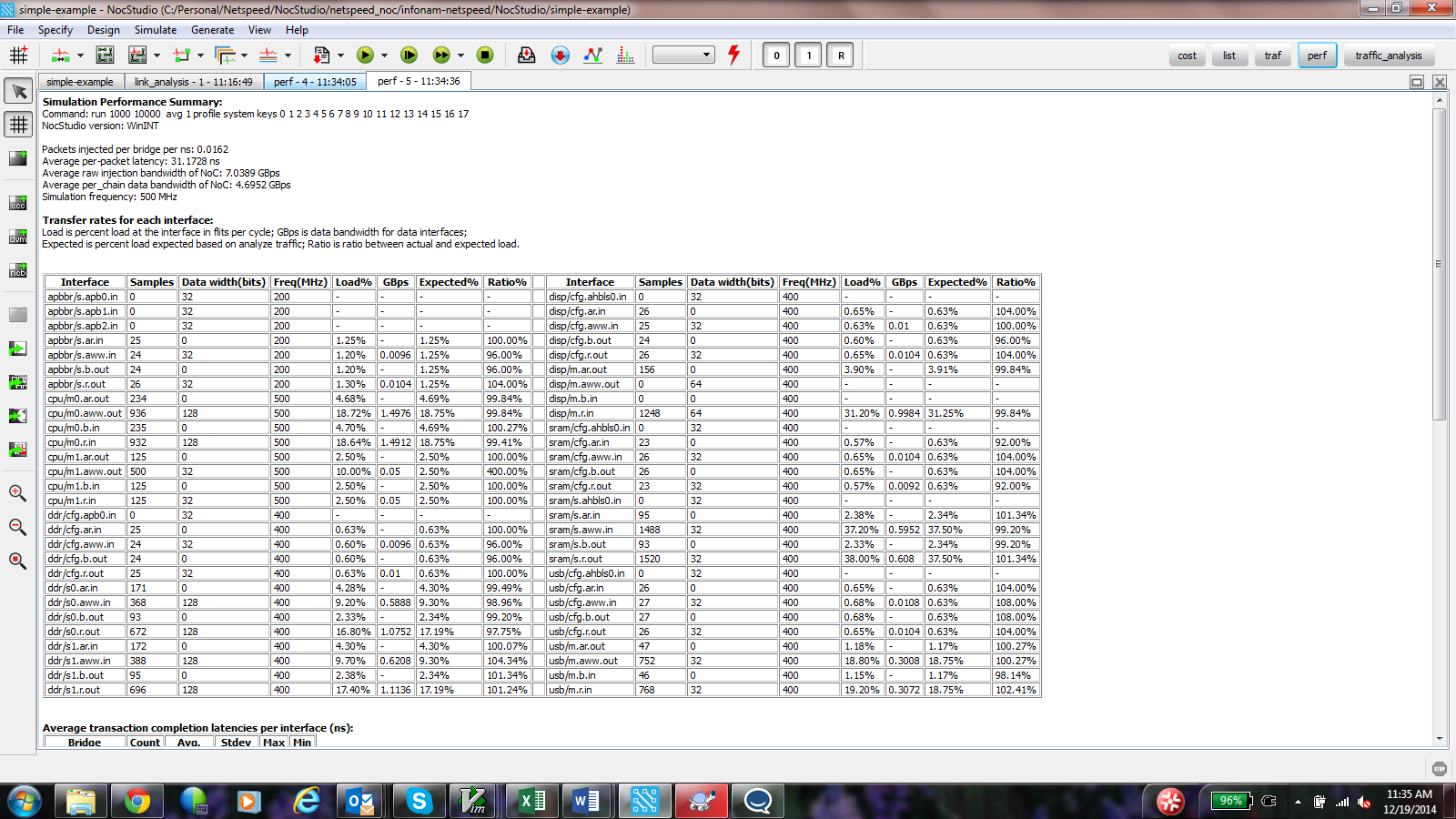


Figure 12: Performance Simulation results

The desired bandwidth is delivered using this NoC, as can be seen in the GBps columns. The structural latency is reported in ns and for DDR is dominated by the 50 cycle (@400Mhz) dram host processing latency. For SRAM access on the other hand, the latency is dominated by the 16 cycle latency (40ns) required to transmit 64B of data over the 4B sram interface. From a dynamic simulation, we expect to see queuing and arbitration delays. The average queuing delay for cpu/m0>ddr/s0 is 154.3/137.5 = 1.12 or about 12% over structural. This is quite acceptable.

NocStudio’s simulator also shades the links to indicate fifo occupancy throughout the NoC.

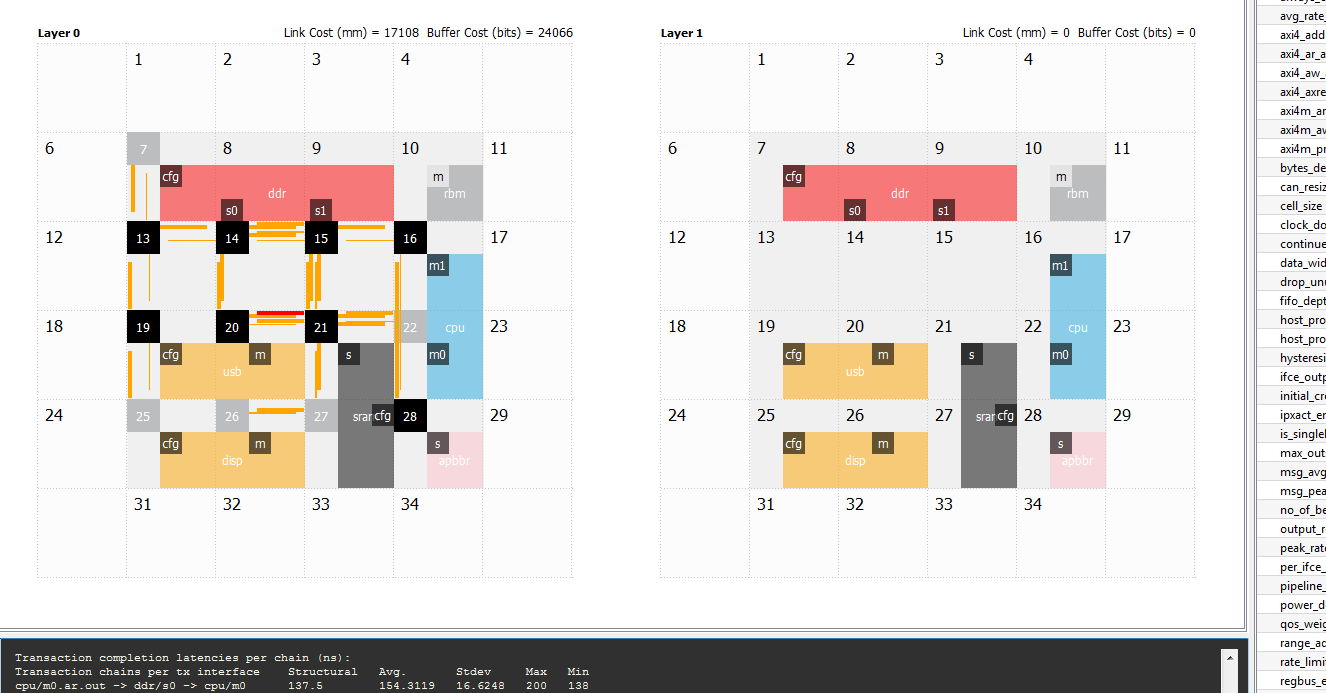


Figure 13: Performance Simulation: visually representing congestion (60-80% occupancy)

There are a couple of observations to be gleaned from this data.

First we have a channel coloured red which means it is over 80% occupied. Though it does not appear to cause a problem (from the rest of the data), it may be useful to size up that fifo in eco mode to avoid any HOL blocking effects on other traffic. **Opt #2.**

Second, we can use the per interface beat data to determine if our max outstanding on the bridges are oversized. **Opt #3**

The ddr beat rate on the ‘ar’ channels show that we get 0.02 beats/ns on that interface. To cover the 125 ns ddr host processing delay we only need 2.5 transactions outstanding. Similarly, conservatively using 200ns of CPU latency to dram for all CPU traffic, we can see that 0.29 x 200ns = support for 6 outstanding transactions is possibly sufficient. Walking through the beat rate and latency values thus gives us a way to optimize the bridge area further.

We can try this out using the NocStudio interactive GUI.

#### Exploring Optimizations with the Simulator and GUI

Using the mouse, first click on the cpu/m0 bridge to select it, and then move the mouse over to the ‘selection’ pane. There click on the cpu/m0/ar.out interface and modify max outstanding from 16 to 8. Similarly modify aww.out for that bridge. Next do the same for both dram bridges, but set the dram value to 6 (down from 16).

Now that the bridge properties for CPU and DDR are modified, we can run a simulation to determine if there’s any effect of reducing the max\_outstanding on the BW or latency.

Though it is possible to run the entire simulation again, a more controlled way to run traces to understand precisely the effect on cpu to ddr /sram traffic is described here.

Move the mouse over to the right panel and select the “sim” panel. In the filter box, type “cpu/m0”. The list of traces displayed now narrows down to just the traces from cpu/m0 to sram/ddr. At the bottom of the panel click on ‘Run’.

NocStudio now simulates ONLY these traces selectively and displays the result. There is a negligible impact on latency.

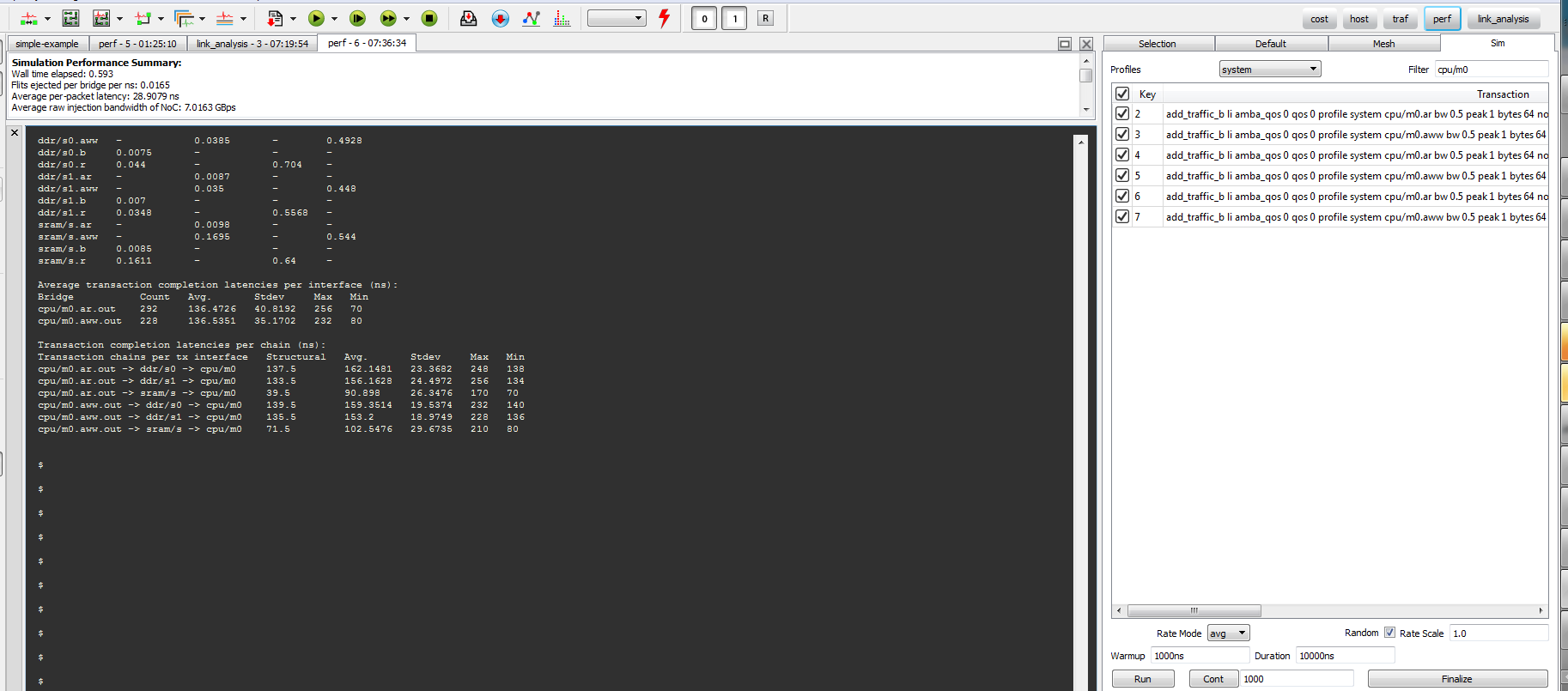


Figure 14: Selectively simulating traces using the 'Sim' panel

Another possible optimization is at the system level: given the observed load on the DDR ports, are two ports really necessary? The second port is really for system bandwidth, but represents an increase in cost due to wires, gates in the ddr controller as well as more routing to be supported on the NoC. **OPT #4.**

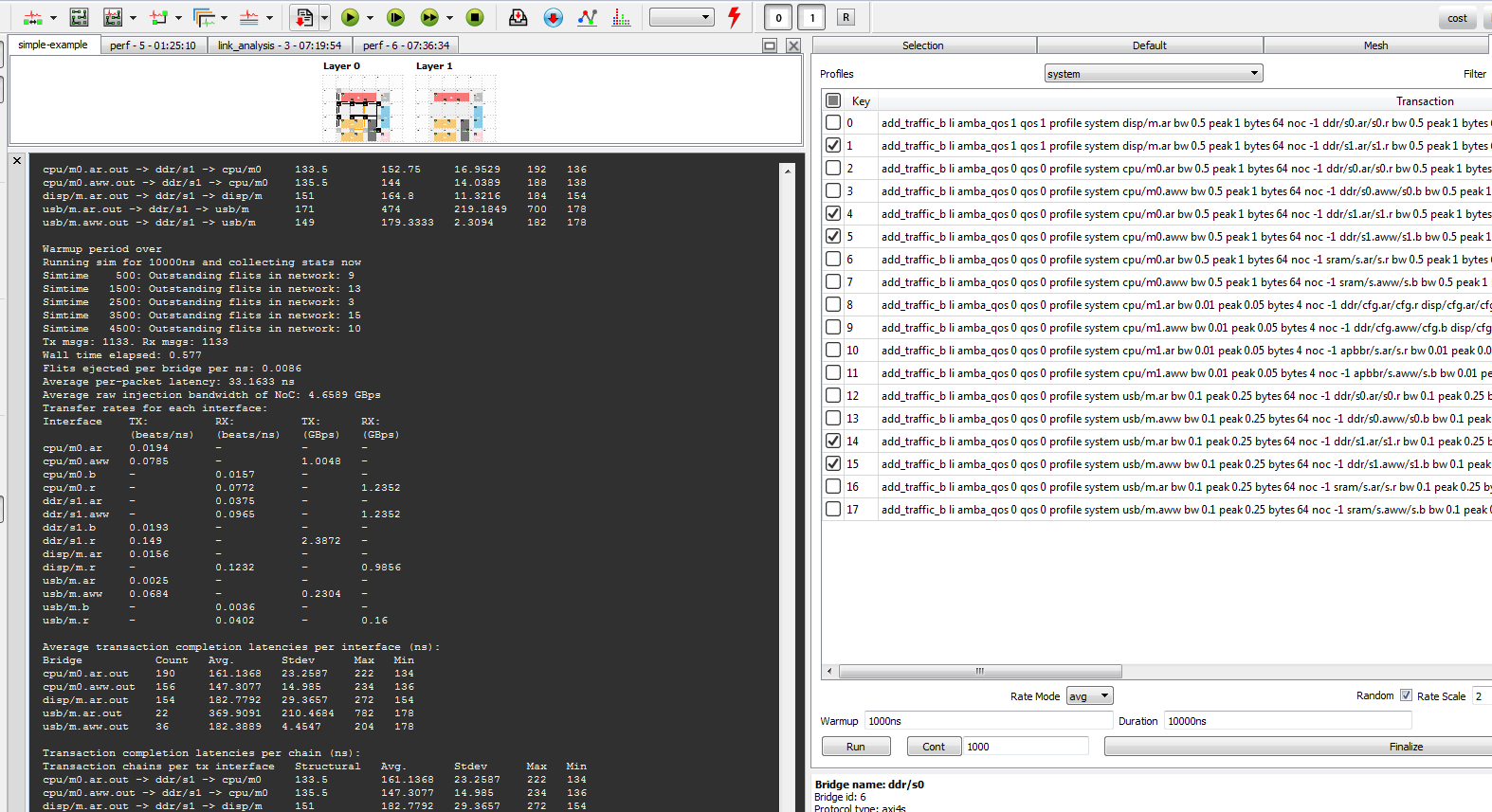
To estimate the effect of a single port dram controller, from the trace files on the ‘sim’ panel select only the flows to ddr/s1. We choose s1 as it is physically closer to the master devices. The generated traces were done assuming bandwidth interleaved between two controllers, so each master only sends half its bandwidth of traffic to a ddr slave port. In this experiment we would like to double the rate from each master to the ddr/s1 slave port. This is accomplished by selecting a ‘Rate Scale’ value of ‘2’ in the bottom right corner of the ‘Sim’panel. Next click on ‘Run ‘ to simulate the traces. The results shown in the figure below indicate that the rated bandwidth is delivered, with acceptable latency impact.

Figure 15: Rate scaling trace files to model single port ddr in the system

### Rebuilding the NoC with Optimizations

A new command file ‘simpler-ex-axi-optimized.txt’ was created with the following three optimizations:

1. Enable downsizing and declare cfg register traffic to be latency insensitive
2. Reduce max outstanding
3. Reduce DDR to one port

This file contains the original statements with comments wherever statements have been modified. To reduce the DDR to a single port, the add\_host, bridge properties, and address maps, and traffic statements must all be modified.

The link analysis report under balanced conditions is shown below. A number of links have been reduced to byte width while still being acceptably loaded, though one link is heavily loaded at 100%. Link analysis included bandwidth headroom, so 100% is including the 25% headroom specified as a mesh prop.

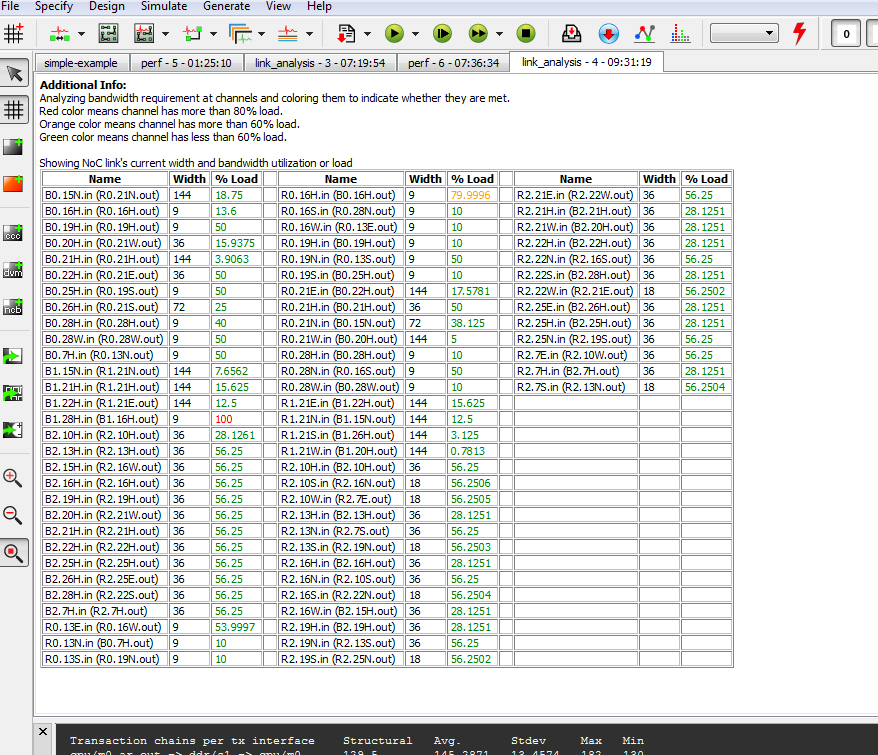


Figure 16: Link Analysis after downsizing (includes 25% extra BW provisioning)

The actual NoC constructed is shown in the next figure:

The main observation is that NocStudio has used two layers, but in these layers has been able to skip a large number of routers. The Link and Buffer cost of this NoC compare very favourably with costs of the previous NoC.

Performance results show an actual improvement in the critical cpu > ddr paths, pointing to this solution being a good one.

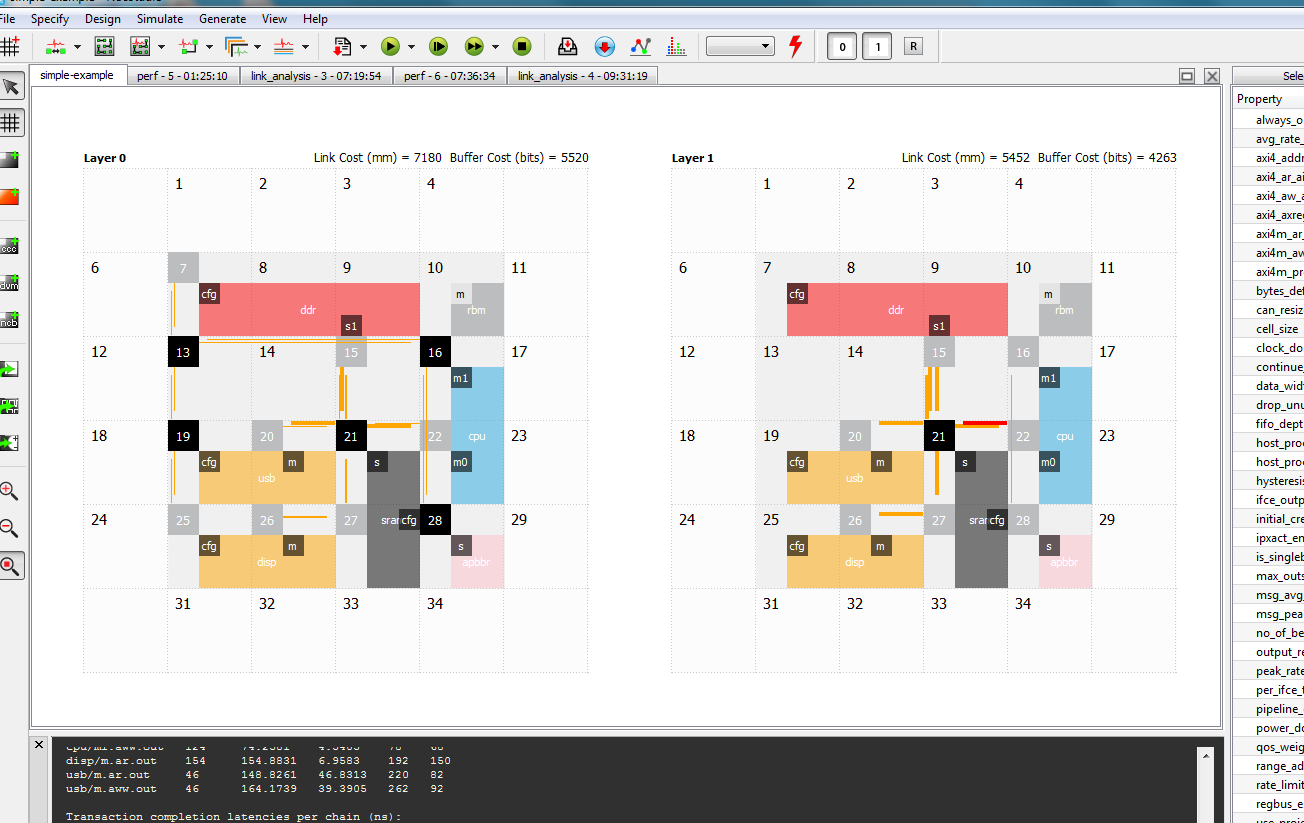


Figure 17: NoC constructed with new command file, showing links after perf simulation

#### Final Optimizations – ECO MODE

What remains now is to decide if the links shaded RED are a cause for concern. The designer has the freedom to make some design tweaks without changing the rest of the design. This is done by entering eco mode by typing ‘eco\_mode’ at the console prompt:

$ eco\_mode

INFO: Entering ECO mode

$

The congested link appears on Router 21 on layer 1. Right clicking on that router expands the router to show the various interface, of which the host interface going to sram is shaded red. Since the Sram is a 4B interface, it is very likely that the long 16 flit stream needed to transfer 64B is causing some blocking effects in R1.21. Clicking on the arrow representing the VC going to the sram host shows the VC info in the right panel. There it is possible to increase the VC fifo depth to 18 so the 16 flits can be buffered in the bridge without causing too much blocking. A similar operation can be done to size up the cpu/m0 bridge VC fifo to 18. Care must be taken to size fifos up for both layer VCs.

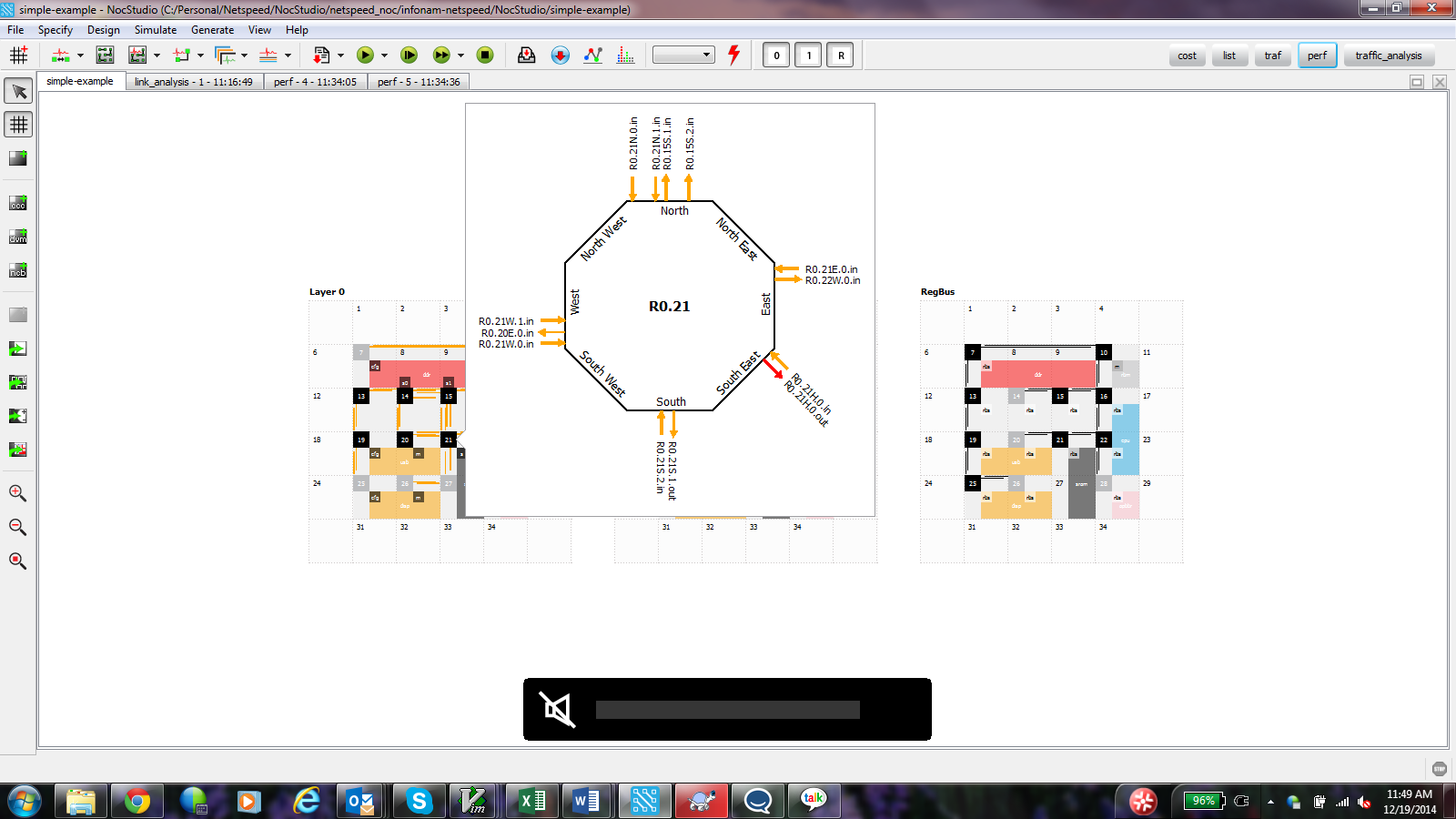


Figure 18: Performing FIFO sizing in ECO mode

Performance results are shown in the next figure after eco mode fixes have been done.

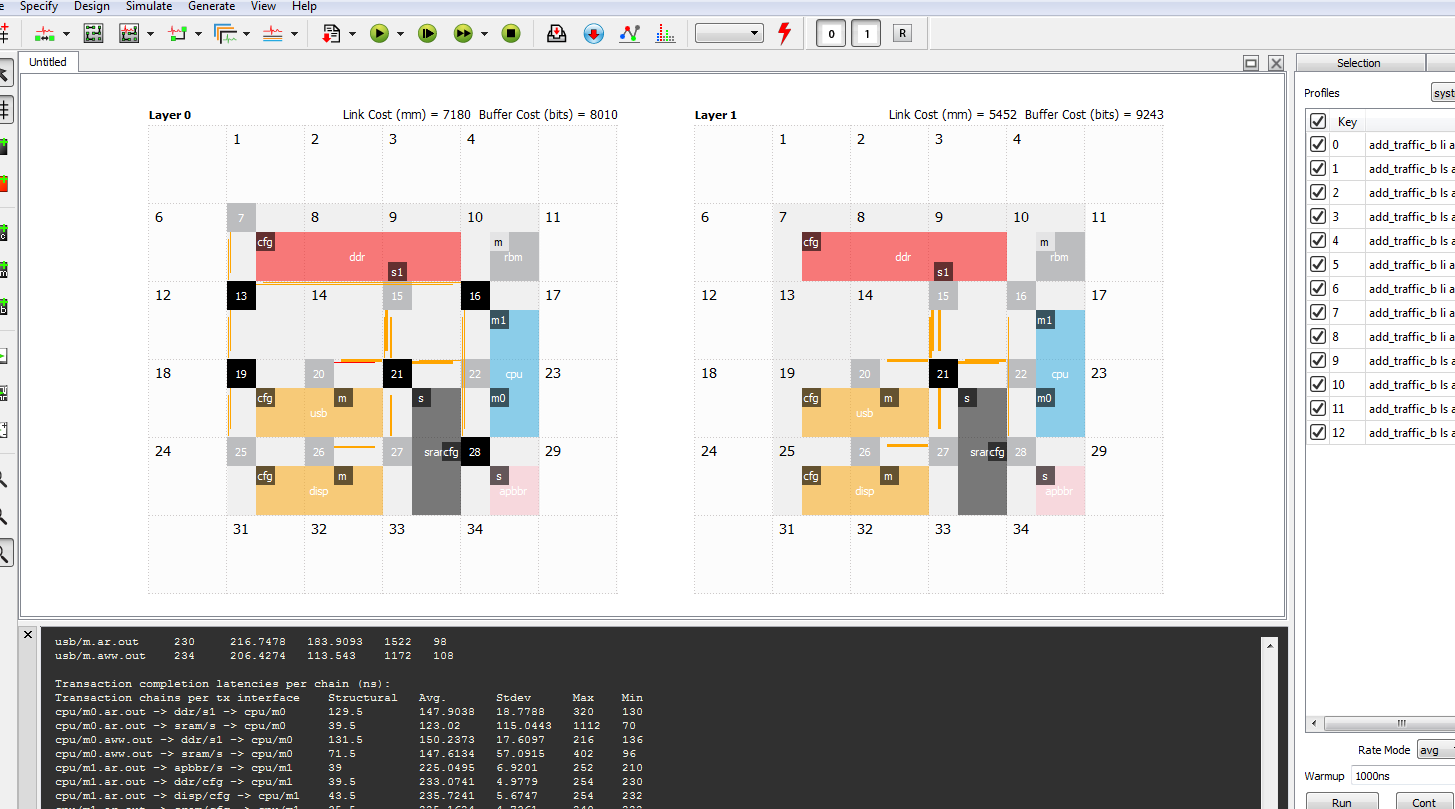


Figure 20 Simulation results after ECO fixes

Clicking on ‘gen\_ip’ on the toolbar, now generates the IP.

## Low Power Example

A simple low power example that uses both power domains and coarse clock gating is presented in this section.

This is a 2 master x 2 slave configuration, using AXI bridges, and with Regbus enabled.

### Clock gating specification

A hysteresis count value needs to be specified as default, so that it applies to all clock gateable elements in the design. Since this is done through the prop\_defaults command, it needs to be done before new\_mesh.

With the commands below a single layer grid is created, with coarse clock gating enabled for the design. This enables hardware controlled activity based clock gating of the router or bridge from the root of the clock distribution. When there is no activity in the router or bridge, and there is no flit in the neighbouring element bound for our router or bridge, the hysteresis counter begins to count down. After 50 clocks, clock will be disabled to this element. This hysteresis register is programmable on silicon, the coarse clock gating scheme is also overridable in HW on silicon. As wake up of an element has a cycle of penalty, the hysteresis value is really to reduce the average latency of the path by engaging clock gating during extended periods of inactivity.

$ prop\_default hysteresis\_count 50

$ new\_mesh 4 4 1 axi\_m2x64\_axi4s2x64\_regbus\_lp regbus\_enabled

$ mesh\_prop coarse\_clock\_gating\_enabled yes

In addition pins for control of clock gating using a clock manager are also exposed at the router and bridge boundaries.

The hysteresis value for each noc element is individually specifiable, though this is not probably necessary.

### Power gating specification

The statements below cause low power options to be generated by NocStudio. It also sets up an always on ‘system’ domain into which will fall all of the top level elements like repeaters etc which are too small and too dispersed at the top level to form a large enough area that is worthy of being power gated.

$ mesh\_prop low\_power\_enabled yes

$ mesh\_prop allow\_always\_on yes

$ set\_power\_domain\_always\_on system yes

$ mesh\_prop max\_new\_power\_domain 2

Next, we add a few power domains using the ‘add\_power\_domain command’. Elements will be assigned to these power domains at a later time.

Next is the addition of hosts and address ranges, using commands discussed in previous examples.

add\_host m0 color red bridge m axi4m

add\_host m1 color blue bridge m axi4m

add\_host s0 color green bridge s axi4s

add\_host s1 color orange bridge s axi4s

add\_range s0/s REG\_BLOCK\_0 0x10000000-0x10001FFF

add\_range s1/s REG\_BLOCK\_1 0x10002000-0x10003FFF

$ add\_power\_domain pwr0

$ add\_power\_domain pwr1

$ add\_power\_domain pwr2

$ add\_power\_domain pwr3

This results in the following floorplan.

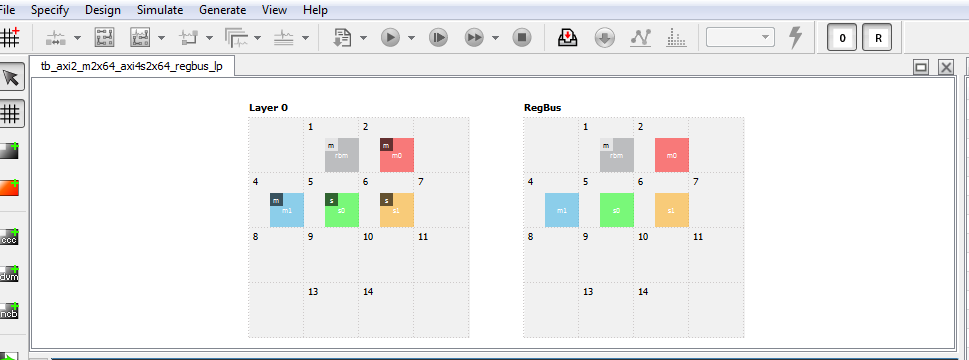


Figure 21: Addition of Hosts: note regbus layer enabled

Each bridge must now be assigned a power domain by the User. This may be done by using the GUI. The bridge is first selected with the mouse click, and then in the ‘Selection’ pane, we scroll to the ‘power\_domain’ property and select the power domain to apply from the drop down menu. Note there exists a ‘power\_domain\_host’ property as well. NocStudio supports a power domain crossing if the host and bridge power domains are required to be different.

Using command line, power domain properties to the bridge may be assigned as follows :

# Assign power domains to bridges

bridge\_prop m0/m power\_domain pwr0

bridge\_prop m1/m power\_domain pwr1

bridge\_prop s0/s power\_domain pwr2

bridge\_prop s1/s power\_domain pwr3

# Add new power profiles

add\_power\_profile m0\_m1\_s0 pwr0 pwr1 pwr2

add\_power\_profile m0\_m1\_s1 pwr0 pwr1 pwr3

add\_power\_profile m0\_s0\_s1 pwr0 pwr2 pwr3

add\_power\_profile m1\_s0\_s1 pwr1 pwr2 pwr3

Power Profile creation is also shown above ; each power profile represents an operating condition in which the power domains which are required to be ‘ON’ are listed.

### Creating the Power Aware NoC

After this, traffic is added from each master to both slaves, and followed by the map and tune\_links commands. These are omitted here in the interest of brevity. At this point the NoC is created. All routers which are created have power assigned to the system power domain. NocStudio auto-assigns routers to power domains, based on its internal algorithms to minimize hardware costs and fit within the user specified envelope of ‘max\_new\_power\_domain’. The command to do this is ‘tune\_power’.

$ tune\_power

Now the different power profiles can be chosen by picking the power profile of interest from the drop down menu, by the lightning bolt sign on the top toolbar. Note that s2 has been powered down, and therefore the elements associated with s2 are considered to be powered down. The console lists the elements which are eligible for power down.

Similarly, different power profiles may be selected.

In the gen\_ip stage, in addition to the RTL, CPF files describing the power intent and required isolation is also generated.

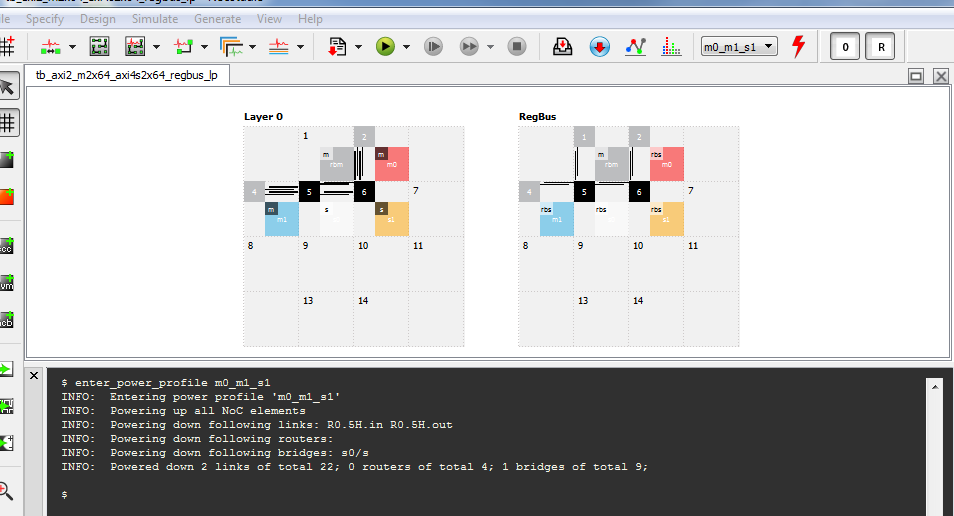


Figure 22: NoC in Power Profile view.

Lastly, the snippet below shows how hysteresis count values may be modified through NocStudio to a value different from the default of 50 cycles specified earlier.

$ router\_prop R0.6 hysteresis\_count 100

2670 Seely Avenue

Building 11

San Jose CA 95134

<www.netspeedsytems.com>