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

For general information about NetSpeed products refer to: www.netspeedsystems.com

Contents

1 TUTORIAL AND EXAMPLE SESSION

1.1 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
```

1.2 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 .

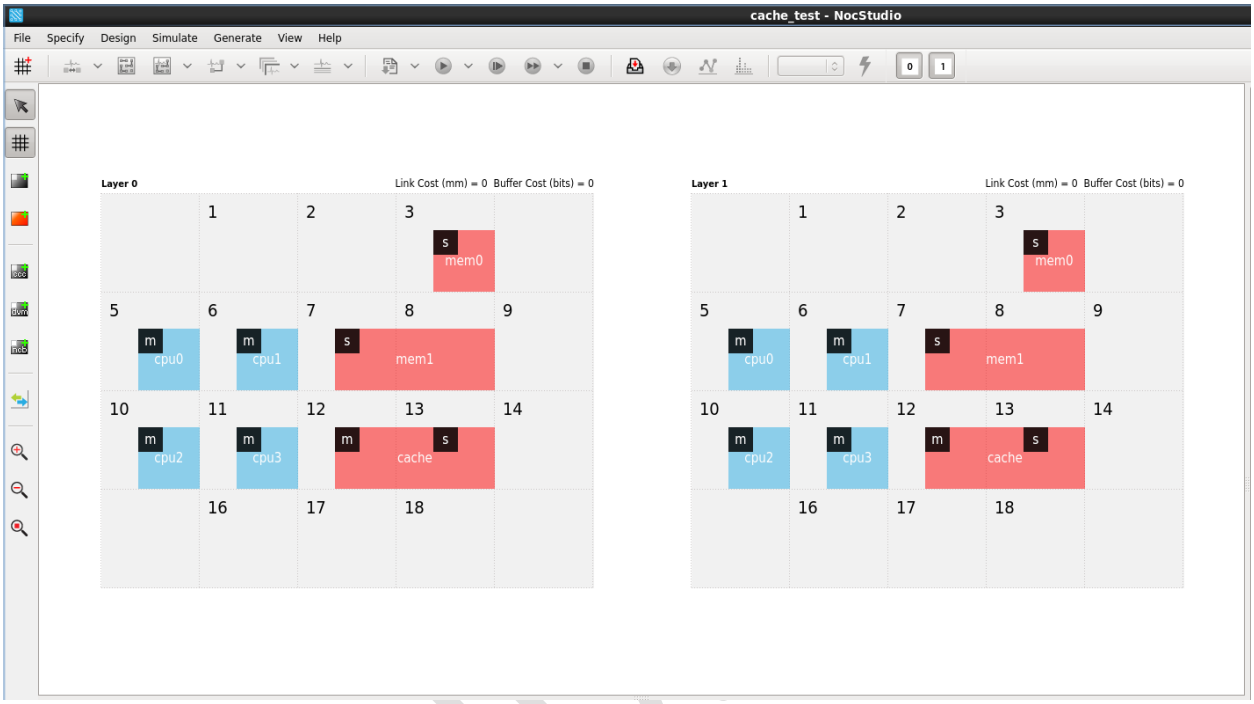


Figure 1 Tutorial 1: NocStudio display after adding all hosts

```
$ 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 .

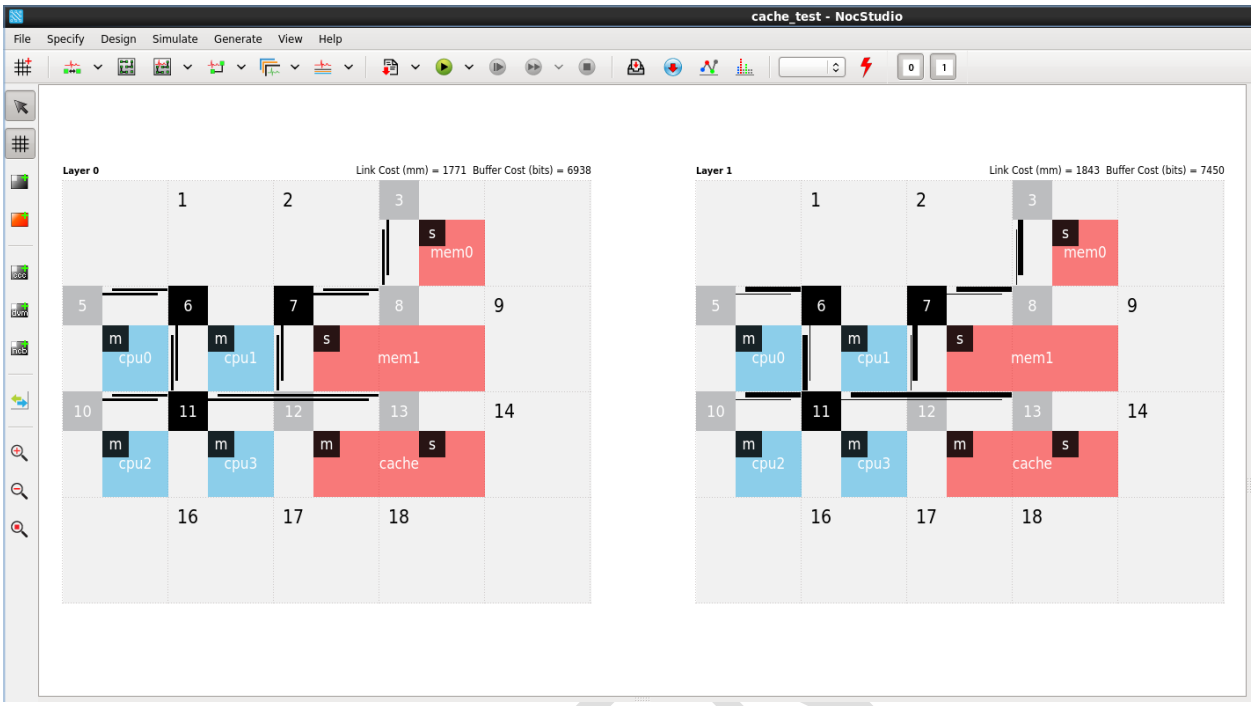


Figure 2 Tutorial 1: NocStudio display after adding traffic and map

```
$ 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.

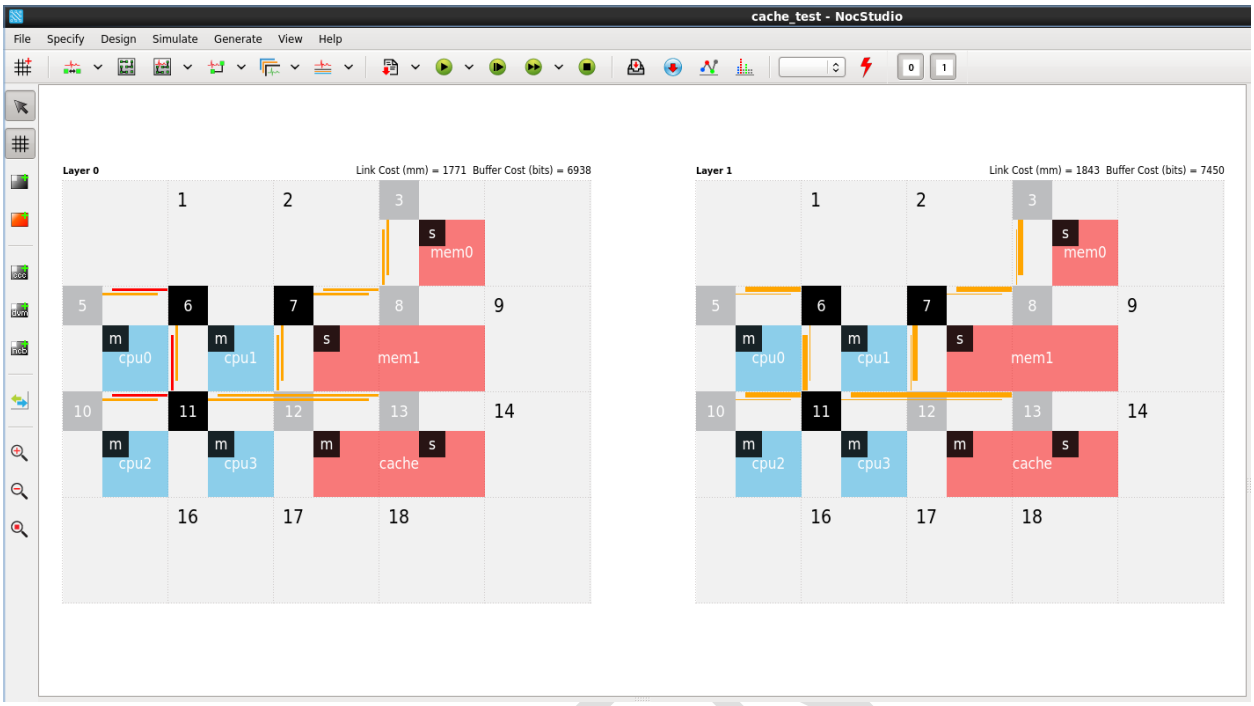


Figure 3 Tutorial 1: NocStudio display after running performance simulation

```
$ 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**.

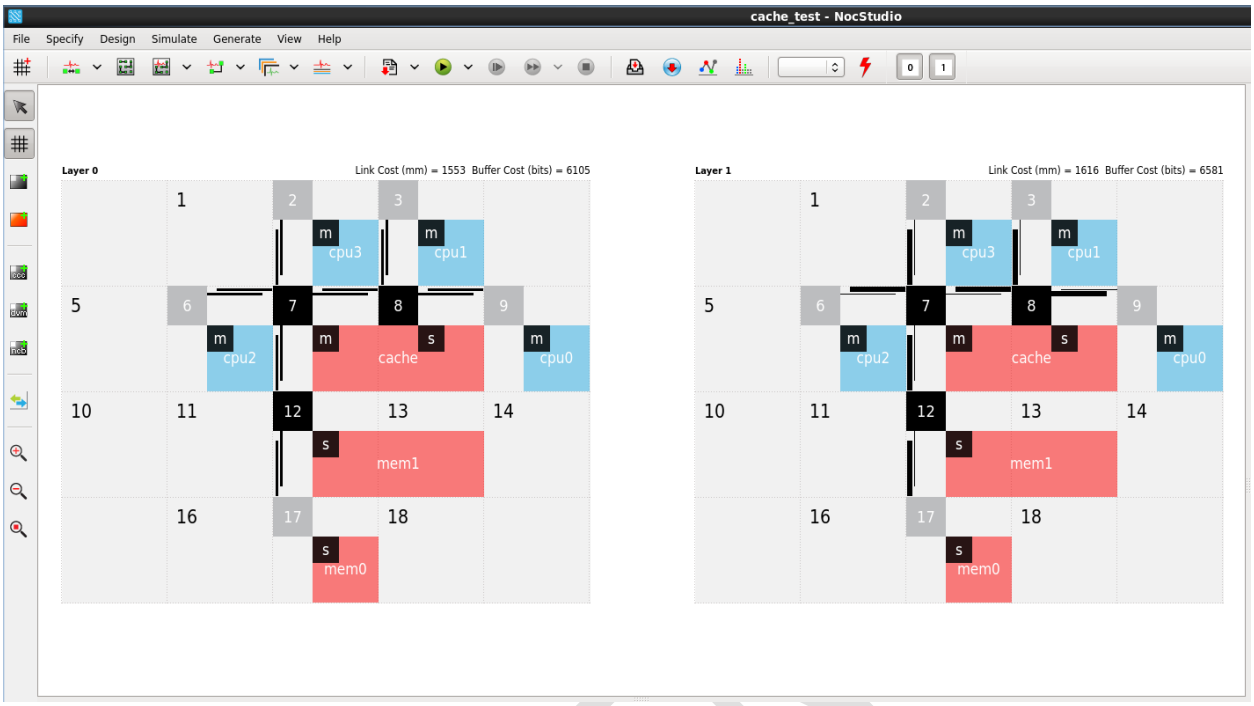


Figure 4 Tutorial 1: NocStudio display after running tune_place

```
$ 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 .

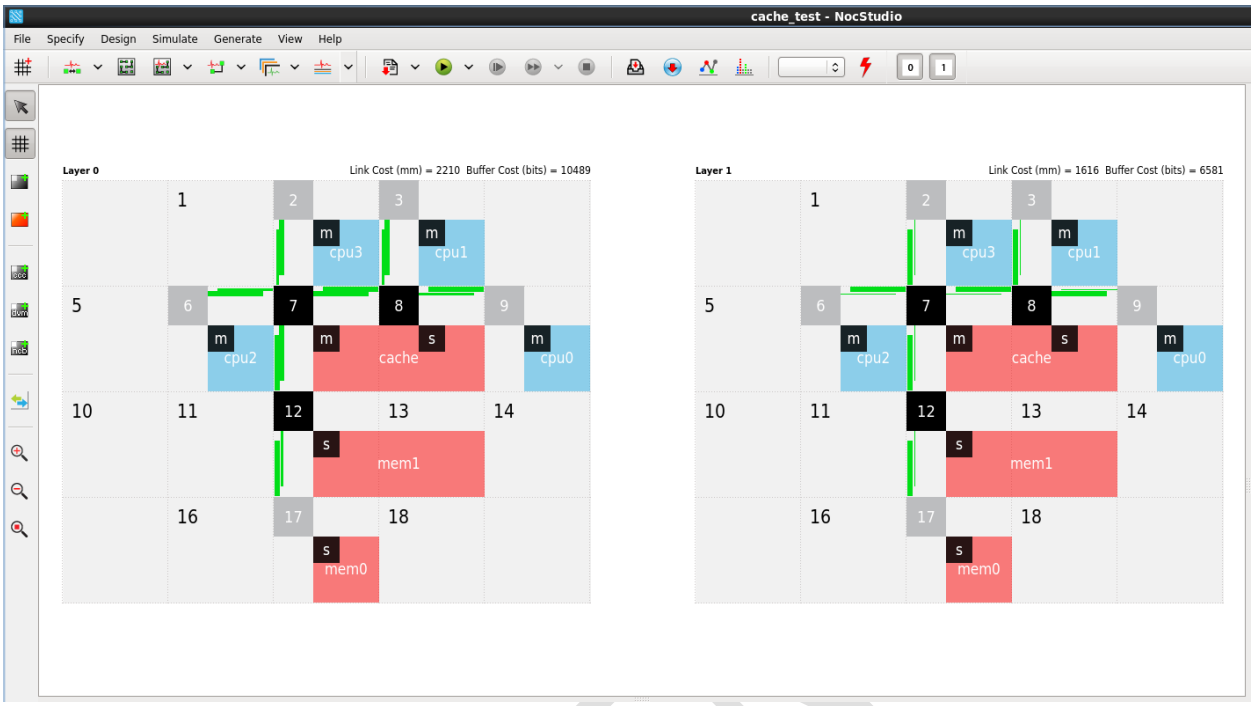


Figure 5 Tutorial 1: NocStudio display after tune_links

```
$ 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.

1.3 AMBA TUTORIAL – II

1.3.1 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

1.3.2 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	

1.3.3 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.

1.3.4 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 :

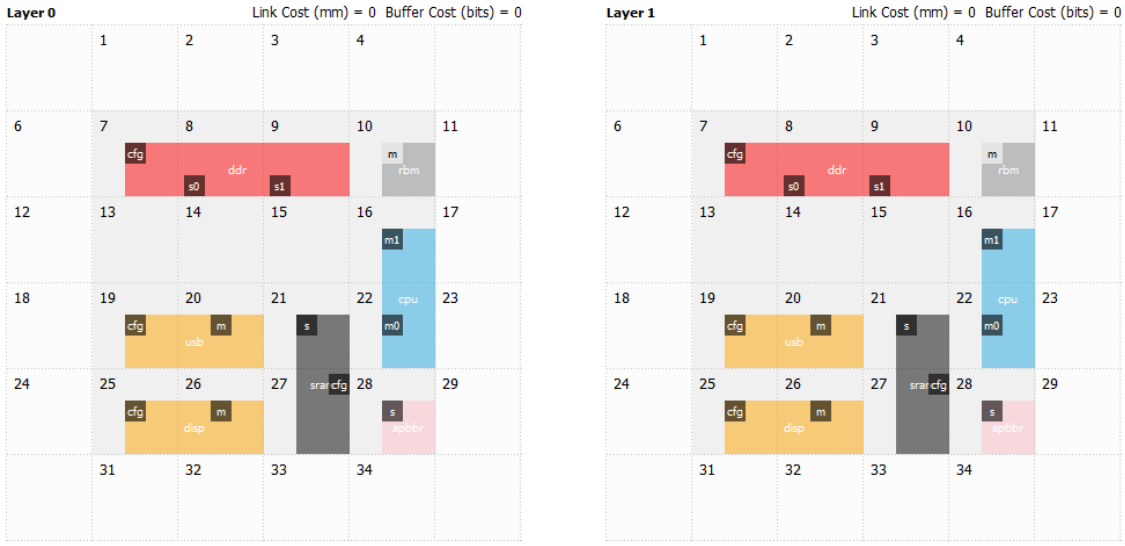


Figure 6 Tutorial 2: Floorplan

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.

1.3.5 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
```

1.3.5.1 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.

1.3.5.2 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.

1.3.5.3 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.

1.3.5.4 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
```

1.3.6 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

1.3.7 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
```

1.3.8 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
```

1.3.9 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
```

1.3.10 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.

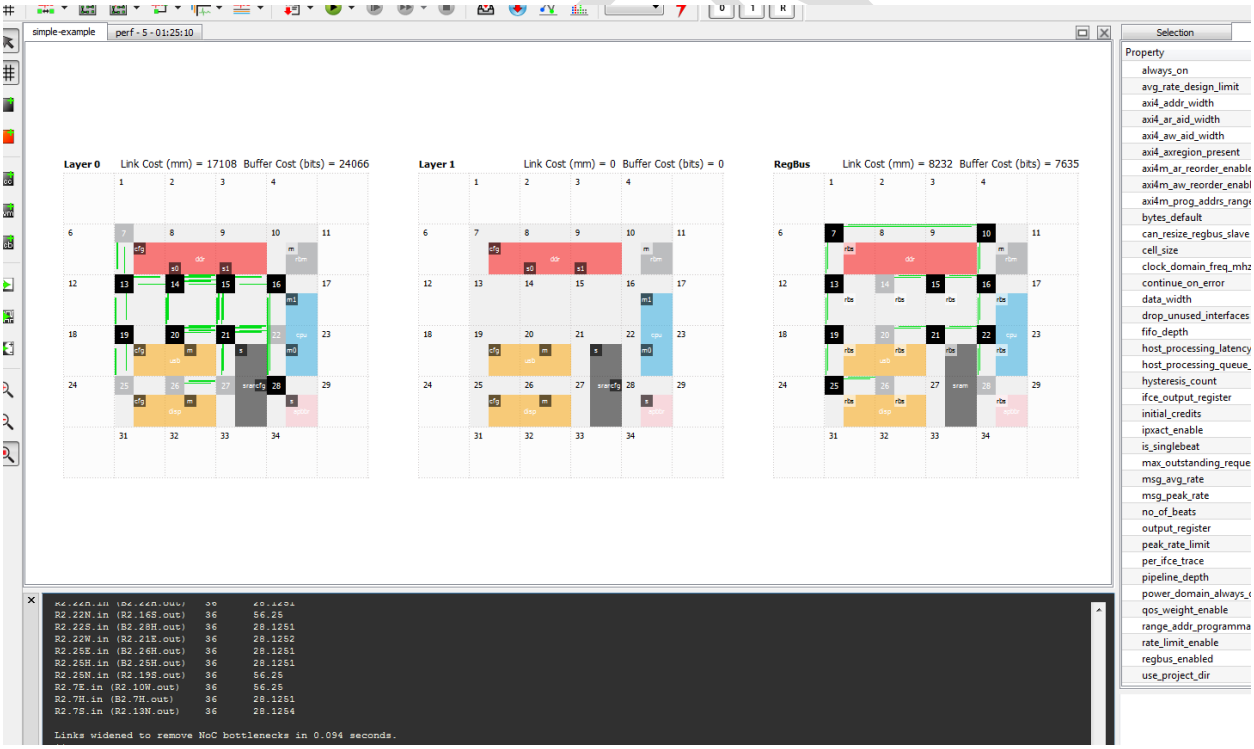


Figure 8: Regbus and Primary layers after map_opt , tune_links

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.

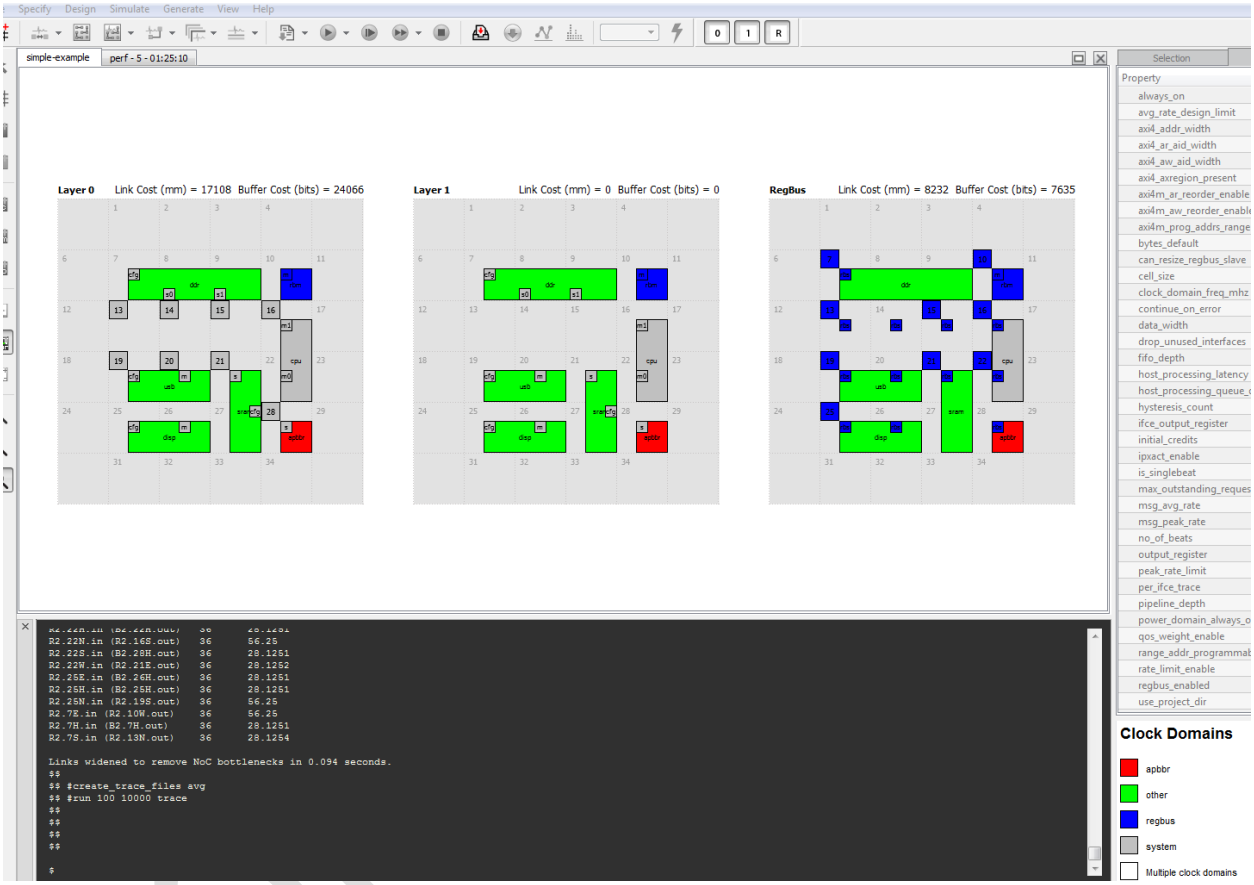
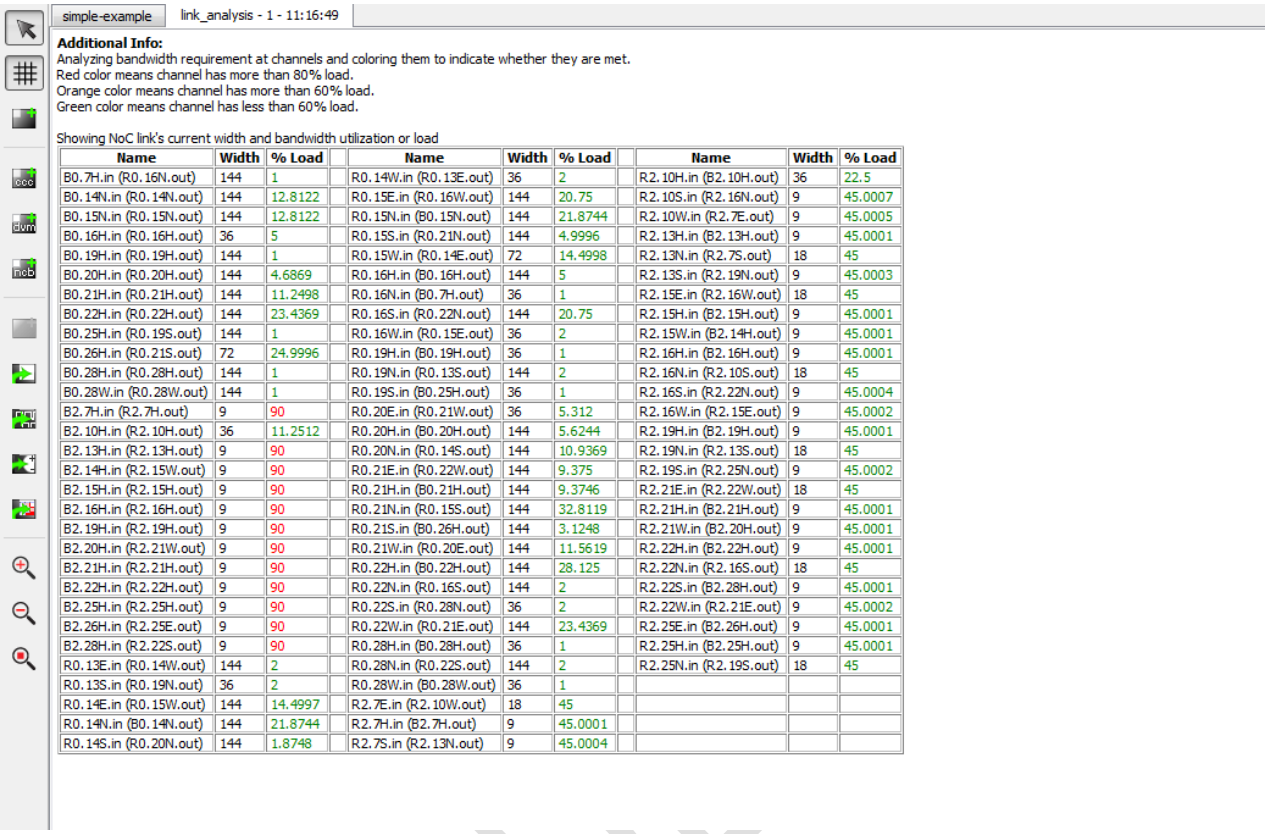


Figure 9: Clock Domain view after mapping

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.



simple-example | link_analysis - 1 - 11:16:49

Additional Info:
Analyzing bandwidth requirement at channels and coloring them to indicate whether they are met.
Red color means channel has more than 80% load.
Orange color means channel has more than 60% load.
Green color means channel has less than 60% load.

Showing NoC link's current width and bandwidth utilization or load

Name	Width	% Load	Name	Width	% Load	Name	Width	% Load
B0.7H.in (R0.16N.out)	144	1	R0.14W.in (R0.13E.out)	36	2	R2.10H.in (B2.10H.out)	36	22.5
B0.14N.in (R0.14N.out)	144	12.8122	R0.15E.in (R0.16W.out)	144	20.75	R2.10S.in (R2.16N.out)	9	45.0007
B0.15N.in (R0.15N.out)	144	12.8122	R0.15N.in (B0.15N.out)	144	21.8744	R2.10W.in (R2.7E.out)	9	45.0005
B0.16H.in (R0.16H.out)	36	5	R0.15S.in (R0.21N.out)	144	4.9996	R2.13H.in (B2.13H.out)	9	45.0001
B0.19H.in (R0.19H.out)	144	1	R0.15W.in (R0.14E.out)	72	14.4998	R2.13N.in (R2.7S.out)	18	45
B0.20H.in (R0.20H.out)	144	4.6869	R0.16H.in (B0.16H.out)	144	5	R2.13S.in (R2.19N.out)	9	45.0003
B0.21H.in (R0.21H.out)	144	11.2498	R0.16N.in (B0.7H.out)	36	1	R2.15E.in (R2.16W.out)	18	45
B0.22H.in (R0.22H.out)	144	23.4369	R0.16S.in (R0.22N.out)	144	20.75	R2.15H.in (B2.15H.out)	9	45.0001
B0.25H.in (R0.19S.out)	144	1	R0.16W.in (R0.15E.out)	36	2	R2.15W.in (B2.14H.out)	9	45.0001
B0.26H.in (R0.21S.out)	72	24.9996	R0.19H.in (B0.19H.out)	36	1	R2.16H.in (B2.16H.out)	9	45.0001
B0.28H.in (R0.28H.out)	144	1	R0.19N.in (R0.13S.out)	144	2	R2.16N.in (R2.10S.out)	18	45
B0.28W.in (R0.28W.out)	144	1	R0.19S.in (B0.25H.out)	36	1	R2.16S.in (R2.22N.out)	9	45.0004
B2.7H.in (R2.7H.out)	9	90	R0.20E.in (R0.21W.out)	36	5.312	R2.16W.in (R2.15E.out)	9	45.0002
B2.10H.in (R2.10H.out)	36	11.2512	R0.20H.in (B0.20H.out)	144	5.6244	R2.19H.in (B2.19H.out)	9	45.0001
B2.13H.in (R2.13H.out)	9	90	R0.20N.in (R0.14S.out)	144	10.9369	R2.19N.in (R2.13S.out)	18	45
B2.14H.in (R2.15W.out)	9	90	R0.21E.in (R0.22W.out)	144	9.375	R2.19S.in (R2.25N.out)	9	45.0002
B2.15H.in (R2.15H.out)	9	90	R0.21H.in (B0.21H.out)	144	9.3746	R2.21E.in (R2.22W.out)	18	45
B2.16H.in (R2.16H.out)	9	90	R0.21N.in (R0.15S.out)	144	32.8119	R2.21H.in (B2.21H.out)	9	45.0001
B2.19H.in (R2.19H.out)	9	90	R0.21S.in (B0.26H.out)	144	3.1248	R2.21W.in (B2.20H.out)	9	45.0001
B2.20H.in (R2.21W.out)	9	90	R0.21W.in (R0.20E.out)	144	11.5619	R2.22H.in (B2.22H.out)	9	45.0001
B2.21H.in (R2.21H.out)	9	90	R0.22H.in (B0.22H.out)	144	28.125	R2.22N.in (R2.16S.out)	18	45
B2.22H.in (R2.22H.out)	9	90	R0.22N.in (R0.16S.out)	144	2	R2.22S.in (B2.28H.out)	9	45.0001
B2.25H.in (R2.25H.out)	9	90	R0.22S.in (R0.28N.out)	36	2	R2.22W.in (R2.21E.out)	9	45.0002
B2.26H.in (R2.25E.out)	9	90	R0.22W.in (R0.21E.out)	144	23.4369	R2.25E.in (B2.26H.out)	9	45.0001
B2.28H.in (R2.22S.out)	9	90	R0.28H.in (B0.28H.out)	36	1	R2.25H.in (B2.25H.out)	9	45.0001
R0.13E.in (R0.14W.out)	144	2	R0.28N.in (R0.22S.out)	144	2	R2.25N.in (R2.19S.out)	18	45
R0.13S.in (R0.19N.out)	36	2	R0.28W.in (B0.28W.out)	36	1			
R0.14E.in (R0.15W.out)	144	14.4997	R2.7E.in (R2.10W.out)	18	45			
R0.14N.in (B0.14N.out)	144	21.8744	R2.7H.in (B2.7H.out)	9	45.0001			
R0.14S.in (R0.20N.out)	144	1.8748	R2.7S.in (R2.13N.out)	9	45.0004			

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.

1.3.10.1 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.

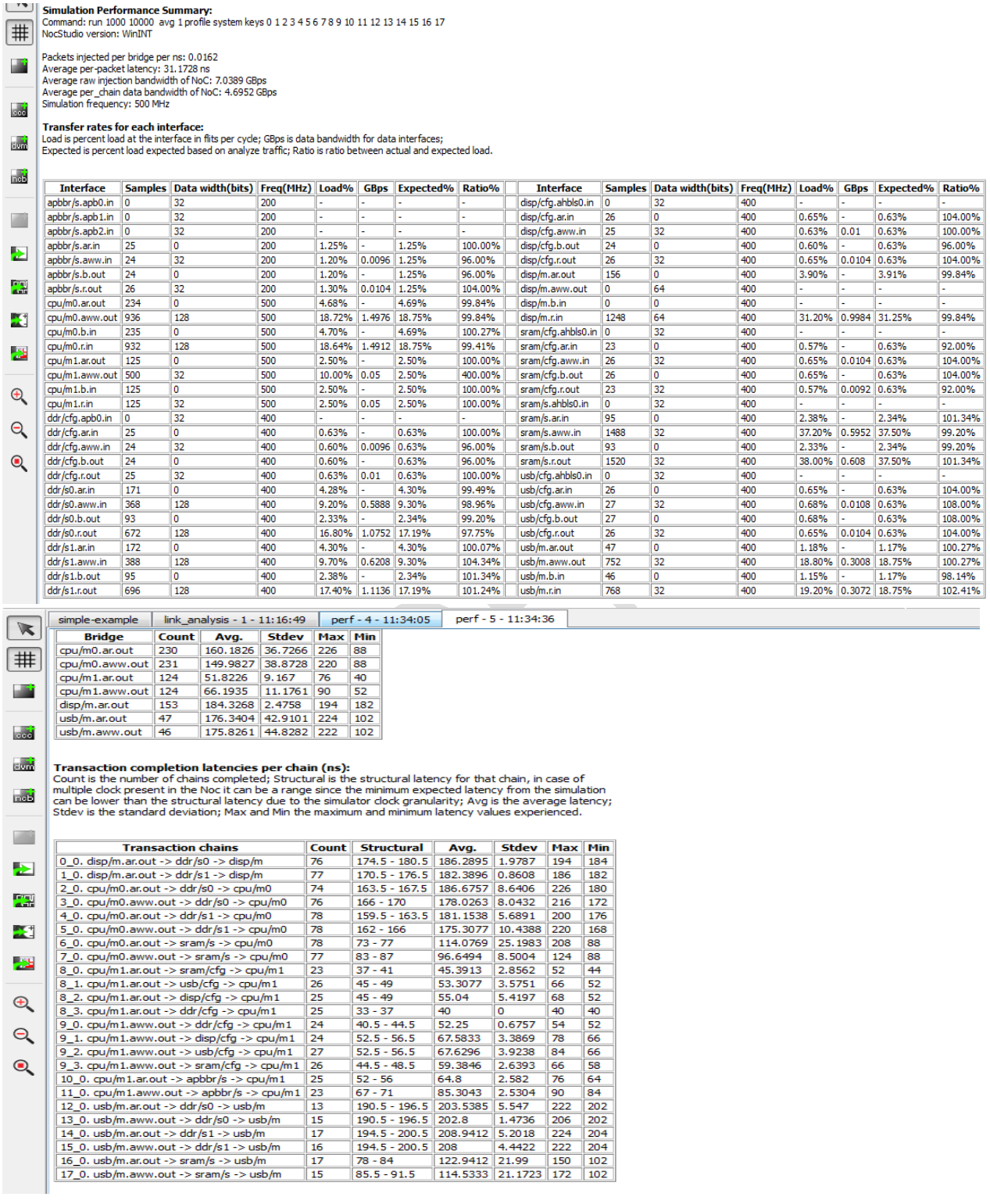
The results expected from simulation can be obtained by using the 'analyze_traffic' command. This shows the result of add traffic : looking at the 'Data' column shows the traffic spec is as expected. NocStudio will flag warnings if the specified traffic exceeds the interface capacity.

```
$ analyze_traffic
INFO:  analyze_traffic running on the list of mapped traffic on profile system.

******************** ANALYZING TRANSACTION RATES ********************

****************** ANALYZING TOTAL TRAFFIC RATES ******************
Listing total avg and peak beat rates and NoC and data bandwidths on interfaces * based on all mapped traffic with profile system and Qo
Ifce             Profile  Avg          BW NoC   Data     Peak         BW NoC   Data
Name             Name     (Beats/ns)   (GBps)   (GBps)   (Beats/ns)   (GBps)   (GBps)
apbbr/s.ar.in    system   0.00625      0.1125   -        0.03125      0.5625   -
apbbr/s.aww.in   system   0.01         0.045    0.01     0.05         0.225    0.05
apbbr/s.b.out    system   0.001        0.00225  -        0.005        0.01125  -
apbbr/s.r.out    system   0.0025       0.01125  0.01     0.0125       0.05625  0.05
cpu/m0.ar.out    system   0.0293       0.52734  -        0.05859      1.05469  -
cpu/m0.aww.out   system   0.11719      2.10938  1.5      0.23438      4.21875  3
cpu/m0.b.in      system   0.01875      0.04219  -        0.0375       0.08438  -
cpu/m0.r.in      system   0.09375      1.6875   1.5      0.1875       3.375    3
cpu/m1.ar.out    system   0.01875      0.3375   -        0.09375      1.6875   -
cpu/m1.aww.out   system   0.05         0.225    0.05     0.25         1.125    0.25
cpu/m1.b.in      system   0.009        0.02025  -        0.045        0.10125  -
cpu/m1.r.in      system   0.0125       0.05625  0.05     0.0625       0.28125  0.25
ddr/cfg.ar.in    system   0.00312      0.05625  -        0.01563      0.28125  -
ddr/cfg.aww.in   system   0.01         0.045    0.01     0.05         0.225    0.05
ddr/cfg.b.out    system   0.002        0.0045   -        0.01         0.0225   -
ddr/cfg.r.out    system   0.0025       0.01125  0.01     0.0125       0.05625  0.05
ddr/s0.ar.in     system   0.01914      0.34453  -        0.03906      0.70313  -
ddr/s0.aww.in    system   0.04648      0.83672  0.6      0.09668      1.74023  1.25
ddr/s0.b.out     system   0.00781      0.01758  -        0.01641      0.03691  -
ddr/s0.r.out     system   0.06875      1.2375   1.1      0.14063      2.53125  2.25
ddr/s1.ar.in     system   0.01914      0.34453  -        0.03906      0.70313  -
ddr/s1.aww.in    system   0.04648      0.83672  0.6      0.09668      1.74023  1.25
ddr/s1.b.out     system   0.00781      0.01758  -        0.01641      0.03691  -
ddr/s1.r.out     system   0.06875      1.2375   1.1      0.14063      2.53125  2.25
disp/cfg.ar.in   system   0.00312      0.05625  -        0.01563      0.28125  -
disp/cfg.aww.in  system   0.01         0.045    0.01     0.05         0.225    0.05
disp/cfg.b.out   system   0.002        0.0045   -        0.01         0.0225   -
disp/cfg.r.out   system   0.0025       0.01125  0.01     0.0125       0.05625  0.05
disp/m.ar.out    system   0.01563      0.28125  -        0.03125      0.5625   -
disp/m.r.in      system   0.125        1.125    1        0.25         2.25     2
sram/cfg.ar.in   system   0.00312      0.05625  -        0.01563      0.28125  -
sram/cfg.aww.in  system   0.01         0.045    0.01     0.05         0.225    0.05
sram/cfg.b.out   system   0.002        0.0045   -        0.01         0.0225   -
sram/cfg.r.out   system   0.0025       0.01125  0.01     0.0125       0.05625  0.05
sram/s.ar.in     system   0.01133      0.20391  -        0.02344      0.42188  -
sram/s.aww.in    system   0.18594      0.83672  0.6      0.38672      1.74023  1.25
sram/s.b.out     system   0.00781      0.01758  -        0.01641      0.03691  -
sram/s.r.out     system   0.15         0.675    0.6      0.3125       1.40625  1.25
usb/cfg.ar.in    system   0.00312      0.05625  -        0.01563      0.28125  -
usb/cfg.aww.in   system   0.01         0.045    0.01     0.05         0.225    0.05
usb/cfg.b.out    system   0.002        0.0045   -        0.01         0.0225   -
usb/cfg.r.out    system   0.0025       0.01125  0.01     0.0125       0.05625  0.05
usb/m.ar.out     system   0.00469      0.08438  -        0.01172      0.21094  -
```

Figure 11: Checking the Traffic Spec: analyze_traffic



Simulation Performance Summary:
Command: run 1000 10000 avg 1 profile system keys 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
NocStudio version: WinINT

Packets injected per bridge per ns: 0.0162
Average per-packet latency: 31.1728 ns
Average raw injection bandwidth of NoC: 7.0389 GBps
Average per_chain data bandwidth of NoC: 4.6952 GBps
Simulation frequency: 500 MHz

Transfer rates for each interface:
Load is percent load at the interface in flits per cycle; GBps is data bandwidth for data interfaces;
Expected is percent load expected based on analyze traffic; Ratio is ratio between actual and expected load.

Interface	Samples	Data width(bits)	Freq(MHz)	Load%	GBps	Expected%	Ratio%	Interface	Samples	Data width(bits)	Freq(MHz)	Load%	GBps	Expected%	Ratio%
apbbr/s.apb0.in	0	32	200	-	-	-	-	disp/cfg.ahbls0.in	0	32	400	-	-	-	-
apbbr/s.apb1.in	0	32	200	-	-	-	-	disp/cfg.ar.in	26	0	400	0.65%	-	0.63%	104.00%
apbbr/s.apb2.in	0	32	200	-	-	-	-	disp/cfg.aww.in	25	32	400	0.63%	0.01	0.63%	100.00%
apbbr/s.ar.in	25	0	200	1.25%	-	1.25%	100.00%	disp/cfg.b.out	24	0	400	0.60%	-	0.63%	96.00%
apbbr/s.aww.in	24	32	200	1.20%	0.0096	1.25%	96.00%	disp/cfg.r.out	26	32	400	0.65%	0.0104	0.63%	104.00%
apbbr/s.b.out	24	0	200	1.20%	-	1.25%	96.00%	disp/m.ar.out	156	0	400	3.90%	-	3.91%	99.84%
apbbr/s.r.out	26	32	200	1.30%	0.0104	1.25%	104.00%	disp/m.aww.out	0	64	400	-	-	-	-
cpu/m0.ar.out	234	0	500	4.68%	-	4.69%	99.84%	disp/m.b.in	0	0	400	-	-	-	-
cpu/m0.aww.out	936	128	500	18.72%	1.4976	18.75%	99.84%	disp/m.r.in	1248	64	400	31.20%	0.9984	31.25%	99.84%
cpu/m0.b.in	235	0	500	4.70%	-	4.69%	100.27%	sram/cfg.ahbls0.in	0	32	400	-	-	-	-
cpu/m0.r.in	932	128	500	18.64%	1.4912	18.75%	99.41%	sram/cfg.ar.in	23	0	400	0.57%	-	0.63%	92.00%
cpu/m1.ar.out	125	0	500	2.50%	-	2.50%	100.00%	sram/cfg.aww.in	26	32	400	0.65%	0.0104	0.63%	104.00%
cpu/m1.aww.out	500	32	500	10.00%	0.05	2.50%	400.00%	sram/cfg.b.out	26	0	400	0.65%	-	0.63%	104.00%
cpu/m1.b.in	125	0	500	2.50%	-	2.50%	100.00%	sram/cfg.r.out	23	32	400	0.57%	0.0092	0.63%	92.00%
cpu/m1.r.in	125	32	500	2.50%	0.05	2.50%	100.00%	sram/s.ahbls0.in	0	32	400	-	-	-	-
ddr/cfg.apb0.in	0	32	400	-	-	-	-	sram/s.ar.in	95	0	400	2.38%	-	2.34%	101.34%
ddr/cfg.ar.in	25	0	400	0.63%	-	0.63%	100.00%	sram/s.aww.in	1488	32	400	37.20%	0.5952	37.50%	99.20%
ddr/cfg.aww.in	24	32	400	0.60%	0.0096	0.63%	96.00%	sram/s.b.out	93	0	400	2.33%	-	2.34%	99.20%
ddr/cfg.b.out	24	0	400	0.60%	-	0.63%	96.00%	sram/s.r.out	1520	32	400	38.00%	0.608	37.50%	101.34%
ddr/cfg.r.out	25	32	400	0.63%	0.01	0.63%	100.00%	usb/cfg.ahbls0.in	0	32	400	-	-	-	-
ddr/s0.ar.in	171	0	400	4.28%	-	4.30%	99.49%	usb/cfg.ar.in	26	0	400	0.65%	-	0.63%	104.00%
ddr/s0.aww.in	368	128	400	9.20%	0.5888	9.30%	98.96%	usb/cfg.aww.in	27	32	400	0.68%	0.0108	0.63%	108.00%
ddr/s0.b.out	93	0	400	2.33%	-	2.34%	99.20%	usb/cfg.b.out	27	0	400	0.68%	-	0.63%	108.00%
ddr/s0.r.out	672	128	400	16.80%	1.0752	17.19%	97.75%	usb/cfg.r.out	26	32	400	0.65%	0.0104	0.63%	104.00%
ddr/s1.ar.in	172	0	400	4.30%	-	4.30%	100.07%	usb/m.ar.out	47	0	400	1.18%	-	1.17%	100.27%
ddr/s1.aww.in	388	128	400	9.70%	0.6208	9.30%	104.34%	usb/m.aww.out	752	32	400	18.80%	0.3008	18.75%	100.27%
ddr/s1.b.out	95	0	400	2.38%	-	2.34%	101.34%	usb/m.b.in	46	0	400	1.15%	-	1.17%	98.14%
ddr/s1.r.out	696	128	400	17.40%	1.1136	17.19%	101.24%	usb/m.r.in	768	32	400	19.20%	0.3072	18.75%	102.41%

simple-example | link_analysis - 1 - 11:16:49 | perf - 4 - 11:34:05 | perf - 5 - 11:34:36

Bridge	Count	Avg.	Stdev	Max	Min
cpu/m0.ar.out	230	160.1826	36.7266	226	88
cpu/m0.aww.out	231	149.9827	38.8728	220	88
cpu/m1.ar.out	124	51.8226	9.167	76	40
cpu/m1.aww.out	124	66.1935	11.1761	90	52
disp/m.ar.out	153	184.3268	2.4758	194	182
usb/m.ar.out	47	176.3404	42.9101	224	102
usb/m.aww.out	46	175.8261	44.8282	222	102

Transaction completion latencies per chain (ns):
Count is the number of chains completed; Structural is the structural latency for that chain, in case of multiple clock present in the Noc it can be a range since the minimum expected latency from the simulation can be lower than the structural latency due to the simulator clock granularity; Avg is the average latency; Stdev is the standard deviation; Max and Min the maximum and minimum latency values experienced.

Transaction chains	Count	Structural	Avg.	Stdev	Max	Min
0_0. disp/m.ar.out -> ddr/s0 -> disp/m	76	174.5 - 180.5	186.2895	1.9787	194	184
1_0. disp/m.ar.out -> ddr/s1 -> disp/m	77	170.5 - 176.5	182.3896	0.8608	186	182
2_0. cpu/m0.ar.out -> ddr/s0 -> cpu/m0	74	163.5 - 167.5	186.6757	8.6406	226	180
3_0. cpu/m0.aww.out -> ddr/s0 -> cpu/m0	76	166 - 170	178.0263	8.0432	216	172
4_0. cpu/m0.ar.out -> ddr/s1 -> cpu/m0	78	159.5 - 163.5	181.1538	5.6891	200	176
5_0. cpu/m0.aww.out -> ddr/s1 -> cpu/m0	78	162 - 166	175.3077	10.4388	220	168
6_0. cpu/m0.ar.out -> sram/s -> cpu/m0	78	73 - 77	114.0769	25.1983	208	88
7_0. cpu/m0.aww.out -> sram/s -> cpu/m0	77	83 - 87	96.6494	8.5004	124	88
8_0. cpu/m1.ar.out -> sram/cfg -> cpu/m1	23	37 - 41	45.3913	2.8562	52	44
8_1. cpu/m1.ar.out -> usb/cfg -> cpu/m1	26	45 - 49	53.3077	3.5751	66	52
8_2. cpu/m1.ar.out -> disp/cfg -> cpu/m1	25	45 - 49	55.04	5.4197	68	52
8_3. cpu/m1.ar.out -> ddr/cfg -> cpu/m1	25	33 - 37	40	0	40	40
9_0. cpu/m1.aww.out -> ddr/cfg -> cpu/m1	24	40.5 - 44.5	52.25	0.6757	54	52
9_1. cpu/m1.aww.out -> disp/cfg -> cpu/m1	24	52.5 - 56.5	67.5833	3.3869	78	66
9_2. cpu/m1.aww.out -> usb/cfg -> cpu/m1	27	52.5 - 56.5	67.6296	3.9238	84	66
9_3. cpu/m1.aww.out -> sram/cfg -> cpu/m1	26	44.5 - 48.5	59.3846	2.6393	66	58
10_0. cpu/m1.ar.out -> apbbr/s -> cpu/m1	25	52 - 56	64.8	2.582	76	64
11_0. cpu/m1.aww.out -> apbbr/s -> cpu/m1	23	67 - 71	85.3043	2.5304	90	84
12_0. usb/m.ar.out -> ddr/s0 -> usb/m	13	190.5 - 196.5	203.5385	5.547	222	202
13_0. usb/m.aww.out -> ddr/s0 -> usb/m	15	190.5 - 196.5	202.8	1.4736	206	202
14_0. usb/m.ar.out -> ddr/s1 -> usb/m	17	194.5 - 200.5	208.9412	5.2018	224	204
15_0. usb/m.aww.out -> ddr/s1 -> usb/m	16	194.5 - 200.5	208	4.4422	222	204
16_0. usb/m.ar.out -> sram/s -> usb/m	17	78 - 84	122.9412	21.99	150	102
17_0. usb/m.aww.out -> sram/s -> usb/m	15	85.5 - 91.5	114.5333	21.1723	172	102

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.

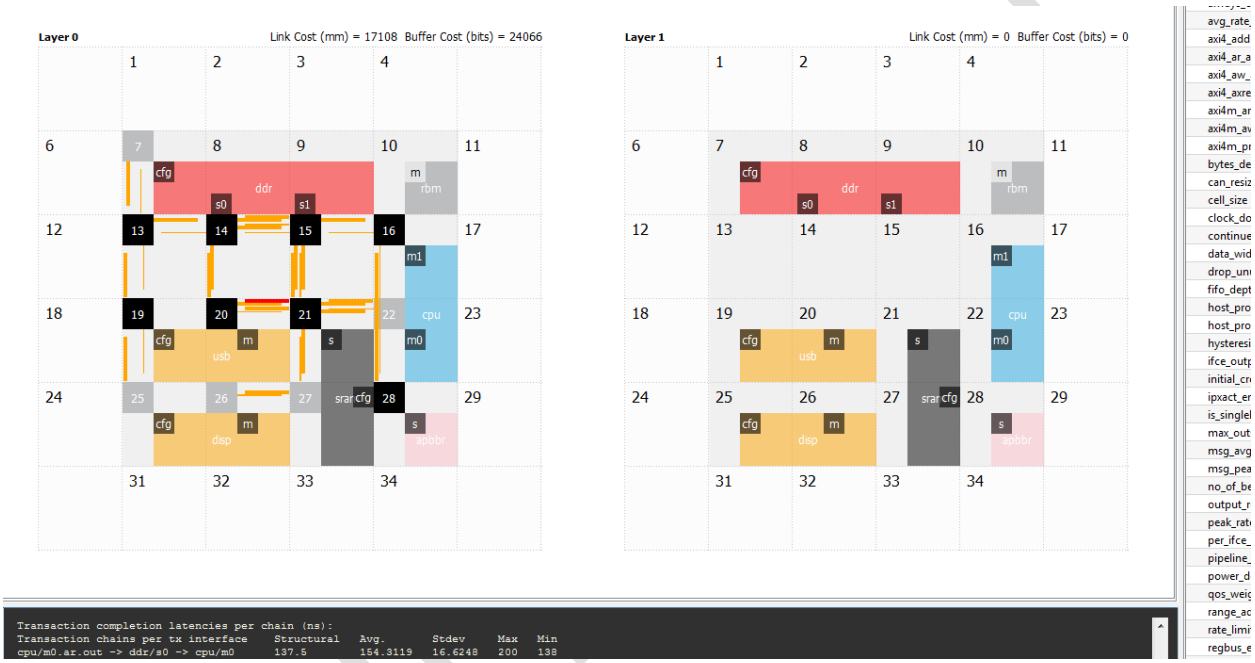


Figure 13: Performance Simulation: visually representing congestion (60-80% occupancy)

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.

1.3.10.2 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.

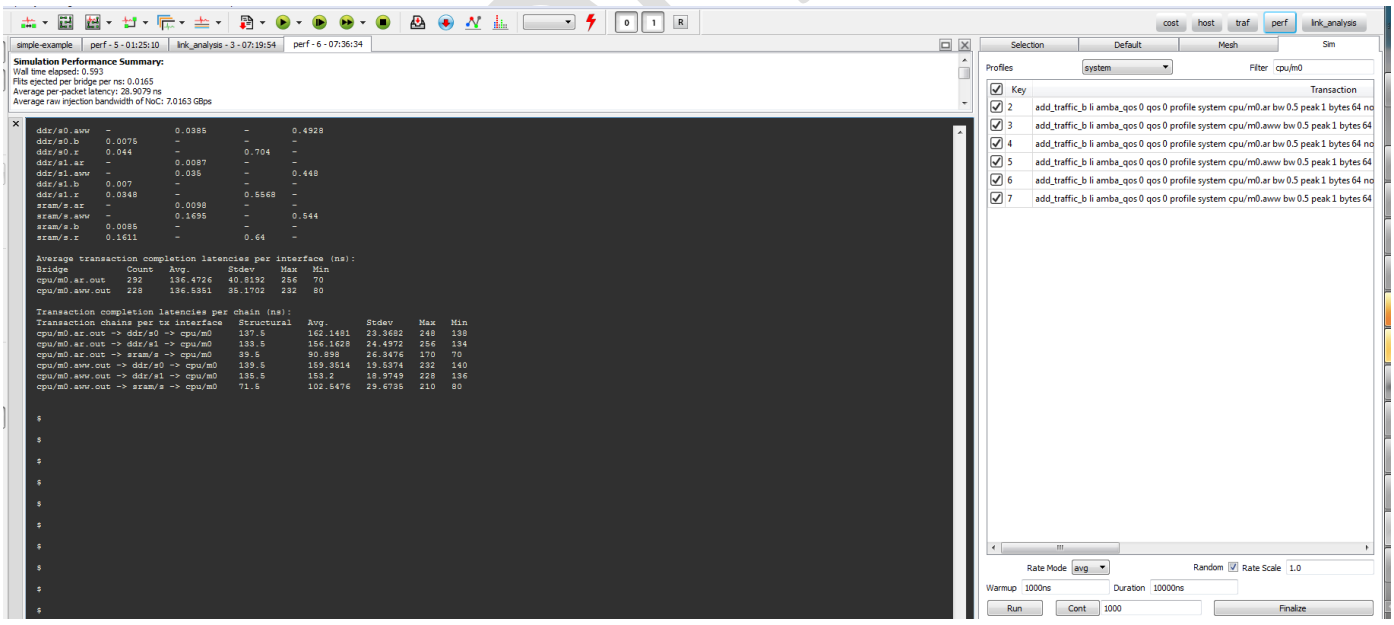


Figure 14: Selectively simulating traces using the 'Sim' panel

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.

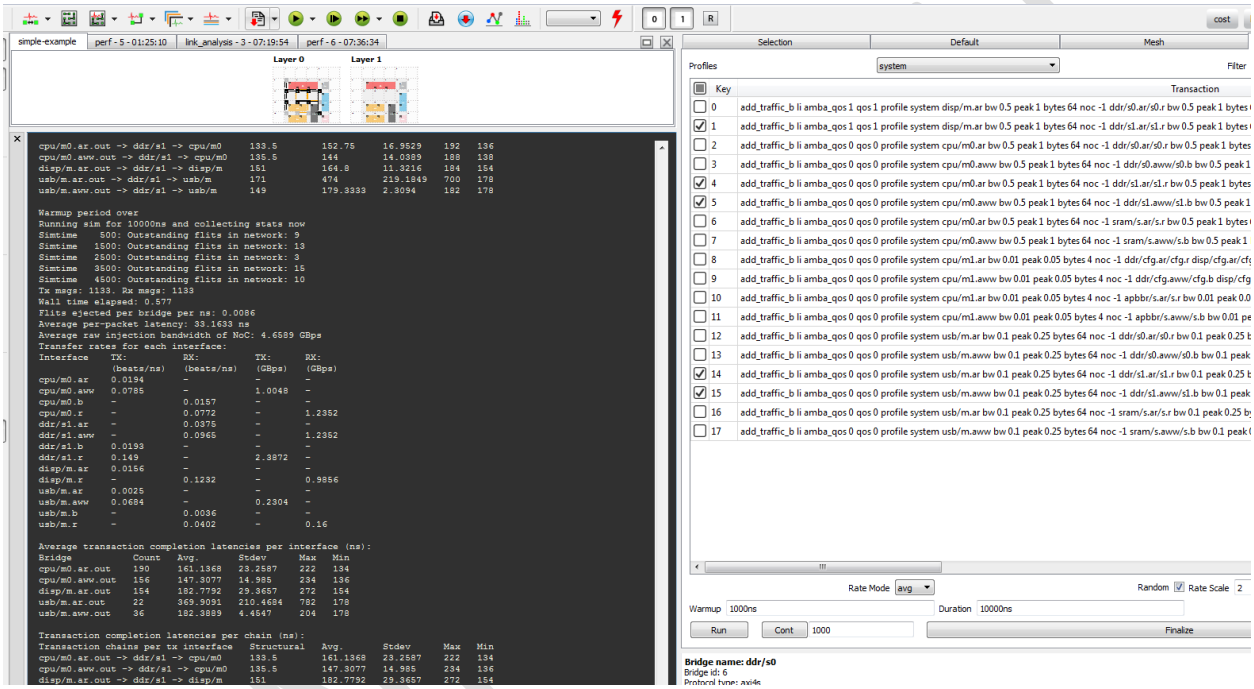


Figure 15: Rate scaling trace files to model single port ddr in the system

1.3.11 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.

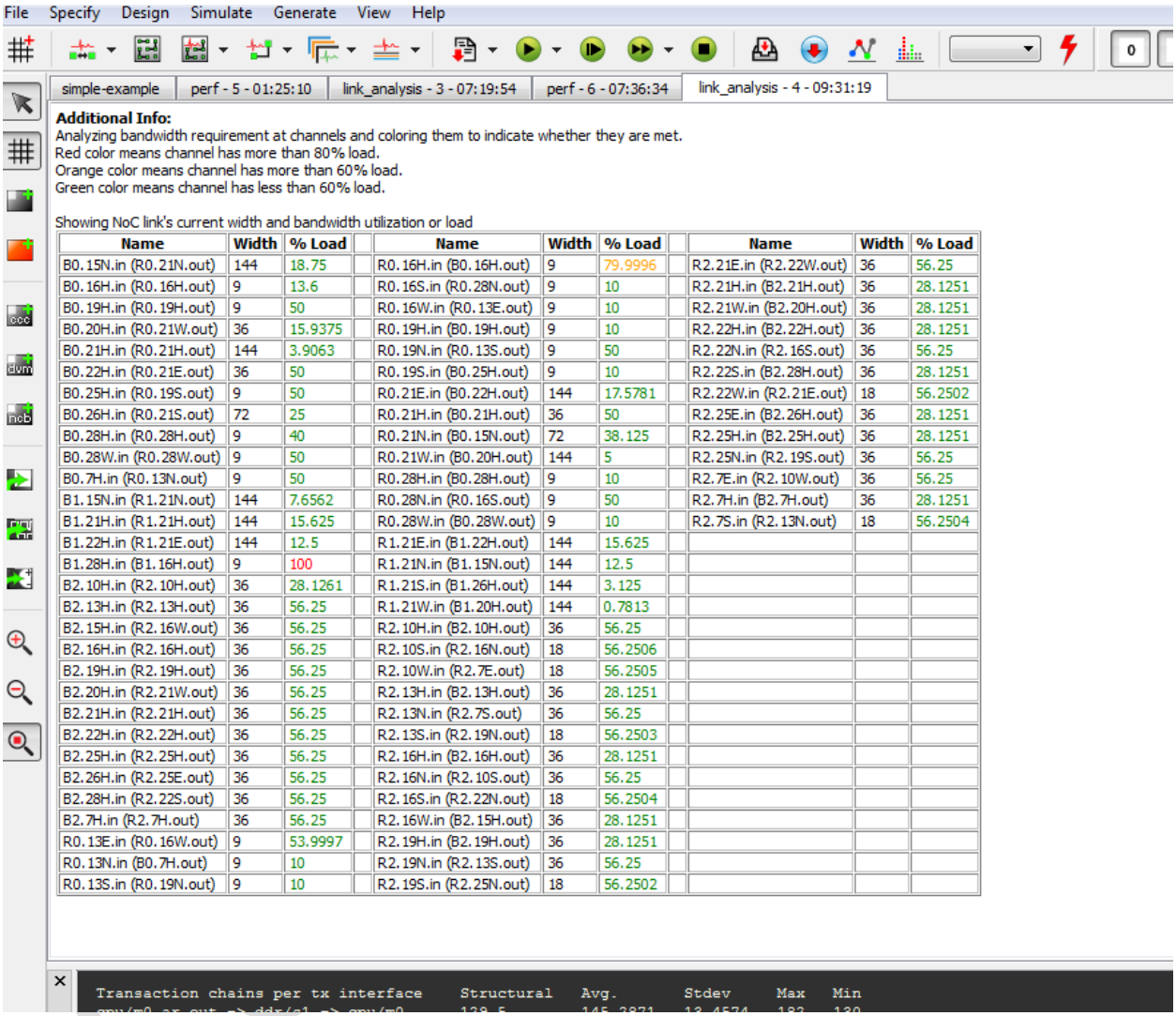


Figure 16: Link Analysis after downsizing (includes 25% extra BW provisioning)

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.

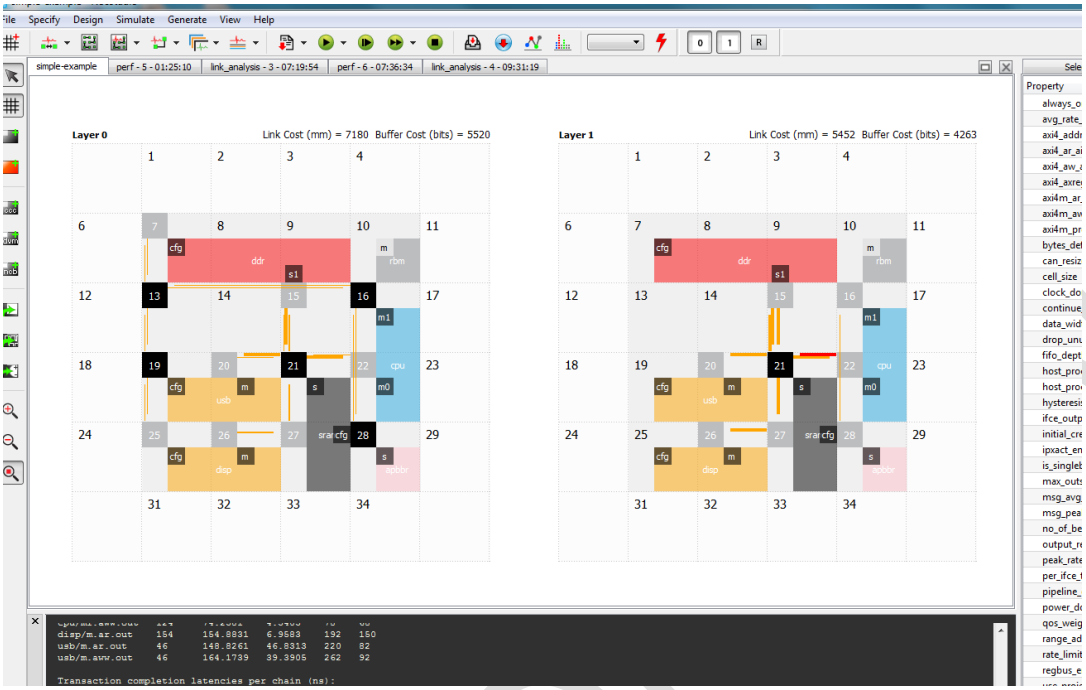


Figure 17: NoC constructed with new command file, showing links after perf simulation

1.3.11.1 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.

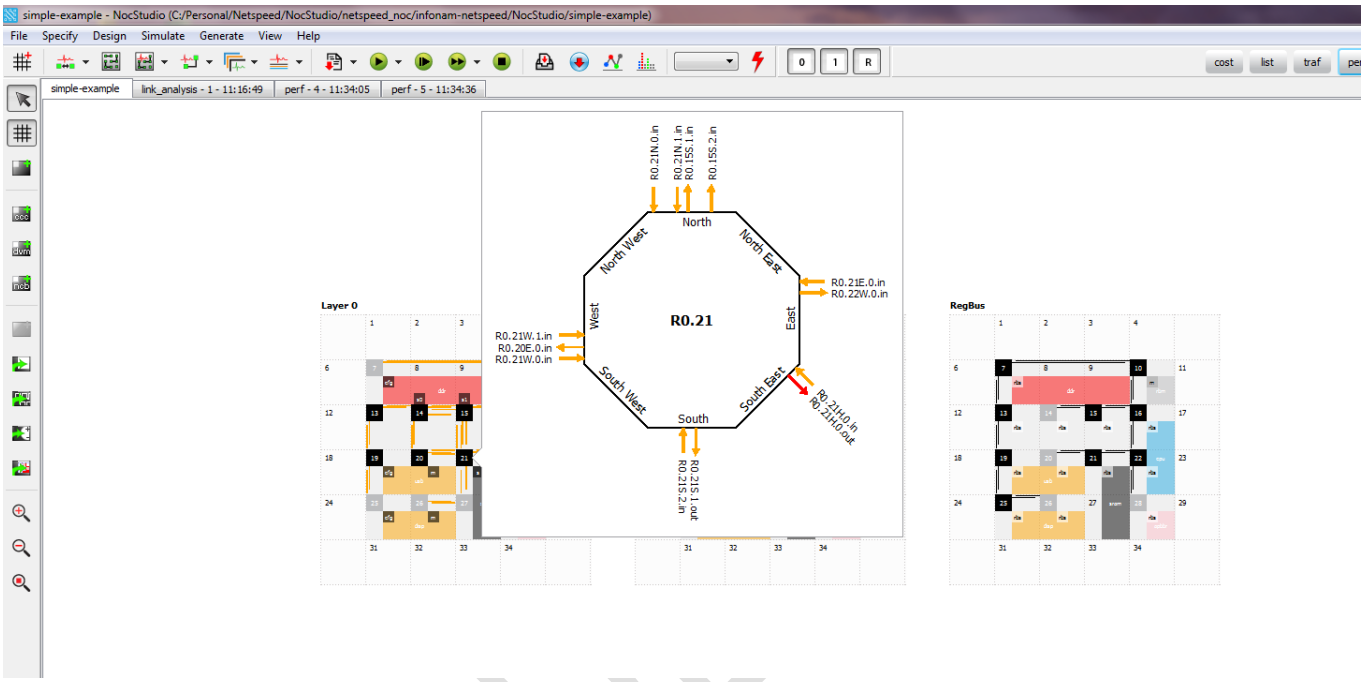


Figure 18: Performing FIFO sizing in ECO mode

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

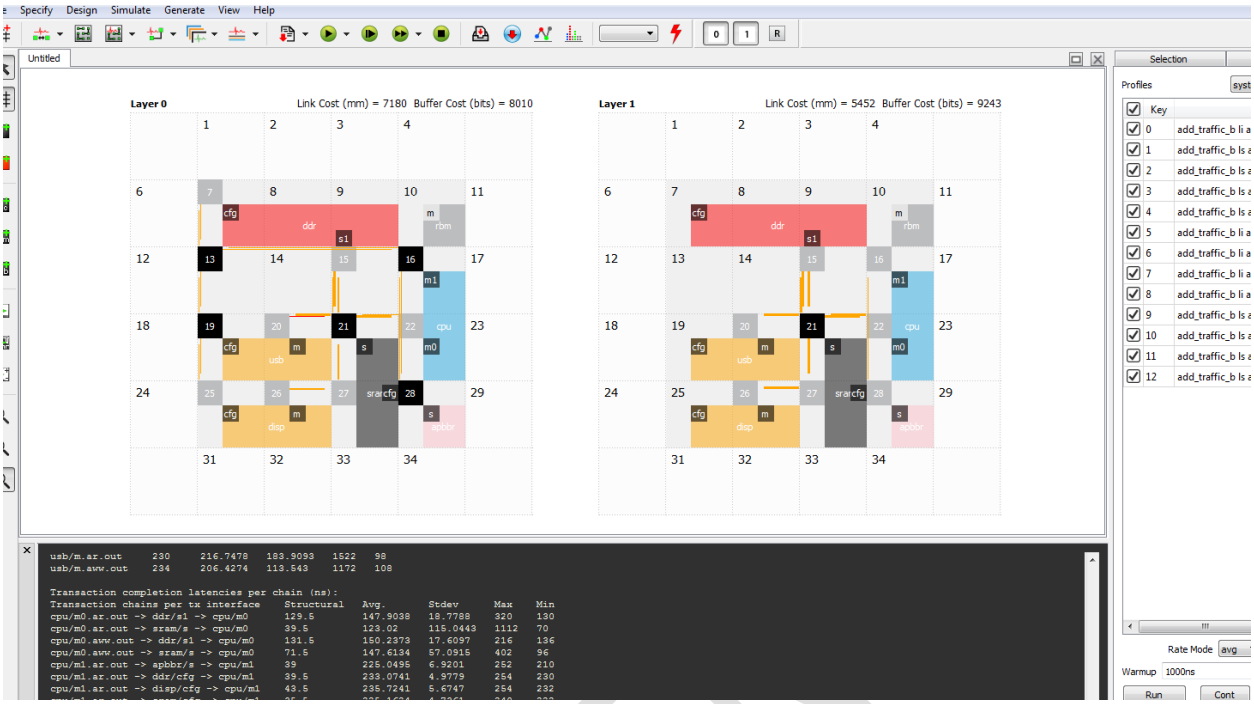


Figure 20 Simulation results after ECO fixes

Clicking on 'gen_ip' on the toolbar, now generates the IP.

1.4 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.

1.4.1 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

```
$ prop_default hysteresis_count 50
$ new_mesh 4 4 1 axi_m2x64_axi4s2x64_regbus_lp regbus_enabled
$ mesh_prop coarse_clock_gating_enabled yes
```

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.

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.

1.4.2 Power gating specification

The statements below cause low power options to be generated by NocStudio. It also sets up an

```
$ 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
```

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.

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.

```
$ add_power_domain pwr0
$ add_power_domain pwr1
$ add_power_domain pwr2
$ add_power_domain pwr3
```

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

```
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
```

examples.

This results in the following floorplan.

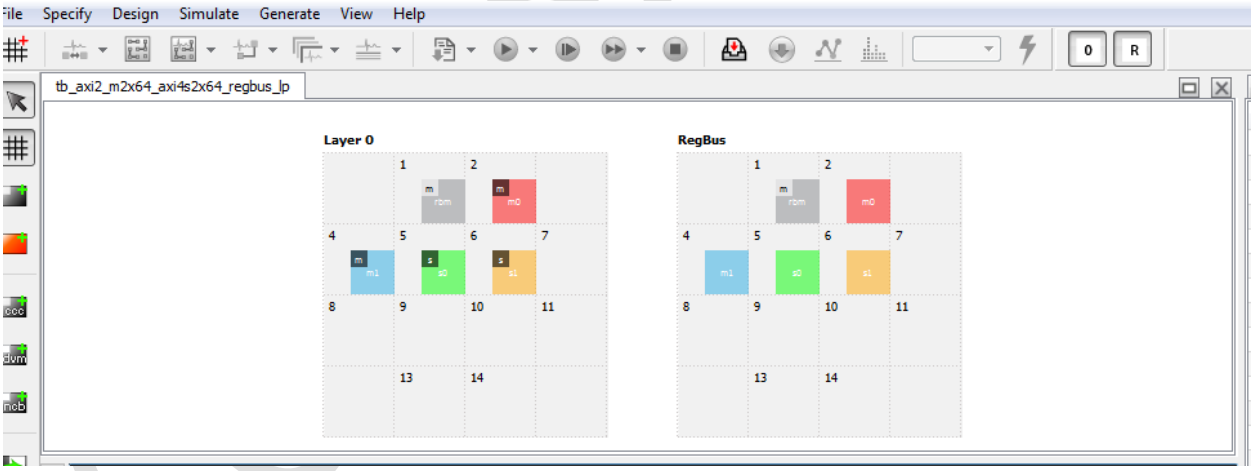


Figure 21: Addition of Hosts: note regbus layer enabled

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.

1.4.3 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.

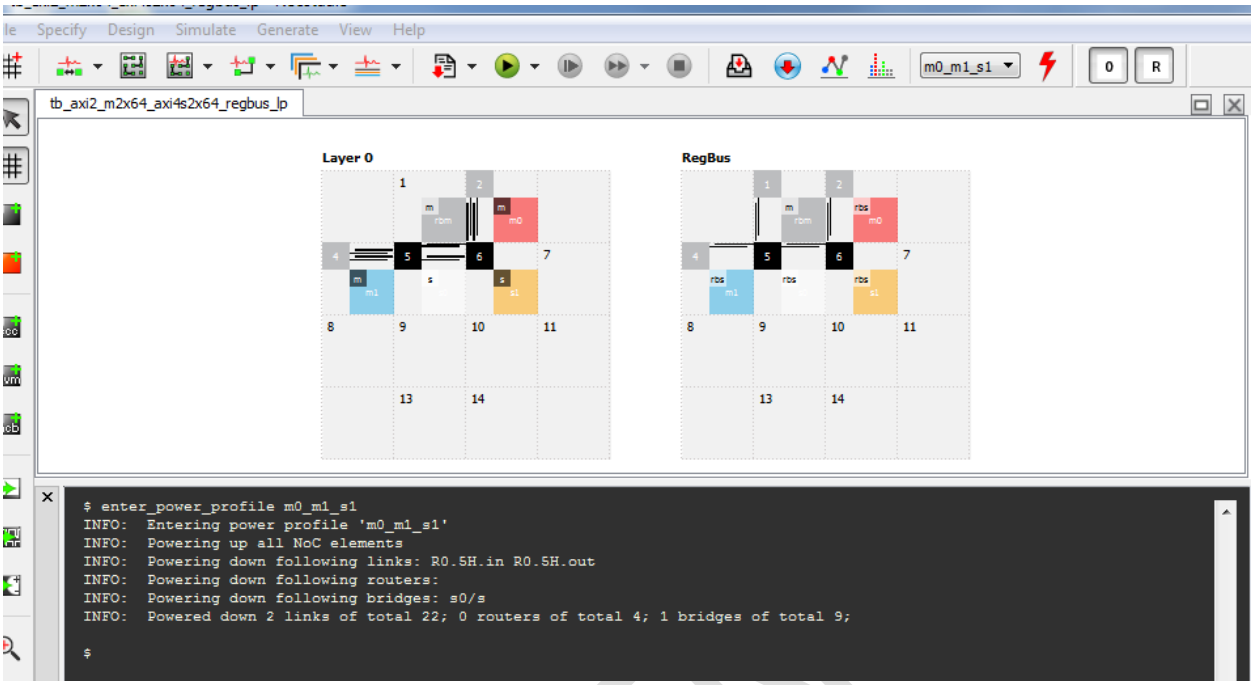


Figure 22: NoC in Power Profile view.

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