Objective: To describe a scenario, simulate with an architecture and analyze the related results

The purpose of this lab is to describe a system scenario starting from an automatic skeleton, then simulate it with a preconfigured architecture and analyze the exploration results.

Procedure:

PART 1: create and refine a scenario skeleton from an existing architecture

1. Open the Lab8.pdd file in the FlexNoC GUI by entering ñ**FlexNoC –p FlexNoC_Lab8.pdd&**ò at the command prompt.

2. In the FlexNoC project panel select the óstep1ô architecture within the Architecture folder, then from the Tools menu execute FlexExploration->Export Skeleton Script

   i. Save it with the given default name

3. Open the generated script with a text editor
4. Refine the scenario according to the following traffic description

Process	Queue depth	From	To	Address generation	Burst Size	Avg Read BW	Avg Write BW	Efficiency
1	256 B	display.scan	DRAM	random	64 bytes	80 MB/s	0	1
ó	1024B	display.rd0	óô	óô	64 bytes	320 MB/s	0	1
ó	1024B	display.rd1	óô	óô	64 bytes	320 MB/s	0	1
ó	1024B	display.wr	óô	óô	64 bytes	0	320 MB/s	1
ó	64 B	modem.cpu (*)	óô	óô	32 bytes	40 MB/s	40 MB/s	0.7
ó	1024B	modem.data	óô	óô	32 bytes	300 MB/s	0	1
ó	2048B	periph.sata	óô	óô	1024 bytes	300 MB/s	0	0
ó	2048B	periph.eth	óô	óô	128 bytes	0	300 MB/s	0
ó	16 kB	image.shade (**)	óô	óô	256 bytes	400 MB/s	400 MB/s0	0.8
ó	16 kB	image.texture (**)	óô	óô	256 bytes	250 MB/s	250 MB/s0	0.8
ó	16 kB	image.motion (**)	óô	óô	256 bytes	175 MB/s	175 MB/s0	0.8
ó	16 kB	image.raster (**)	óô	óô	256 bytes	325 MB/s	325 MB/s0	0.8
4	128 B	cpu.data (***)	óô	óô	128 bytes	(***)	(***)	0
4	64 B	cpu.inst (***)	óô	óô	64 bytes	(***)	(***)	0

i. (*) for modem.cpu create a procedure performing a write-back cache data refill (including block-write to memory) and apply a global space throughput equal to Read BW + Write BW

ii. (**) create just a single process queuing on all the image.xx sockets; it has to choose among four sub-procedures, each one sending transactions to a single/different image.xx socket; the weight of each procedure has to be set equal to Read BW + Write BW (integer positive value, no metric suffix) of the associated socket; the whole procedure has to be spaced with an average (poisson) throughput of 2.3 GBytes/s

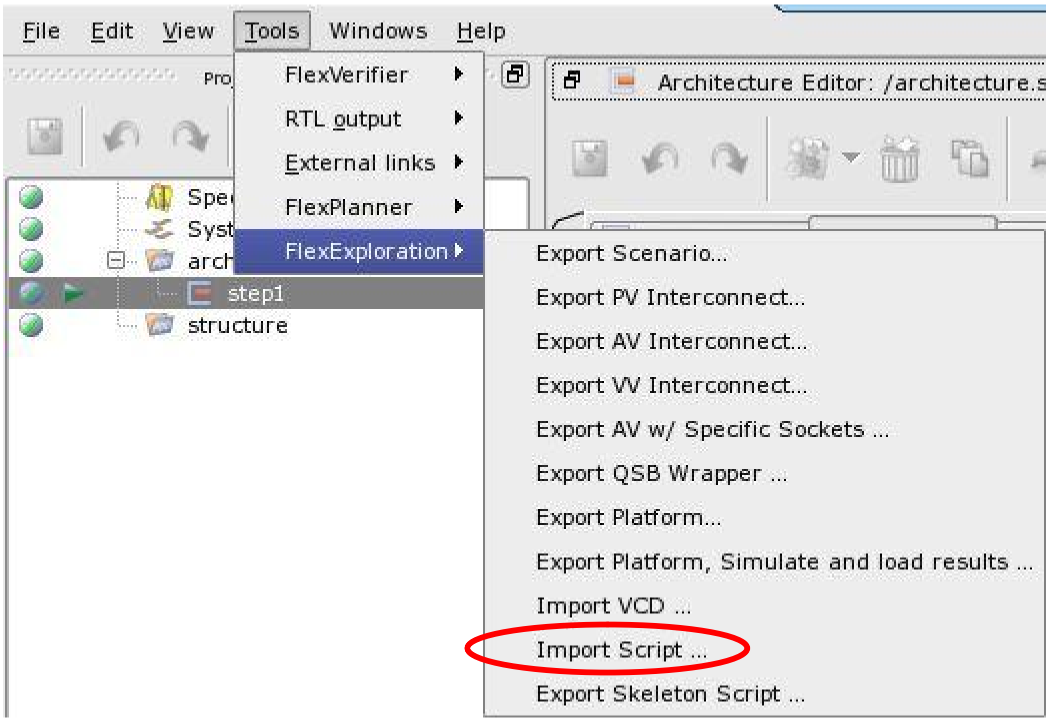
   a. Each sub-procedure has at its turn to choose beween a Read and a Write with the same weight

iii. (***) model a multi-threaded cpu using 4 processes,

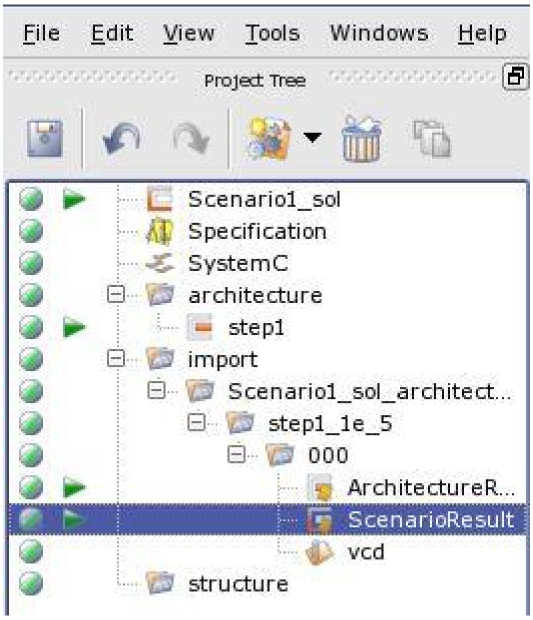
   a. each one has a couple of queues respectively associated to cpu.data and cpu.inst sockets
   b. each process chooses (with the same weight) among a data-miss on socket cpu.data (block-reading from memory only), a write-back data-miss on socket cpu.data (including block-write to memory) and a instruction-miss on socket cpu.inst(block-reading from memory only);
   c. the whole choose procedure has to be spaced with an average (poisson) throughput of 250 MBytes/s and, after that, be repeated to generate a volume of 1 MByte

PART 2: simulate the new scenario with the existing step1 architecture and inspect the results

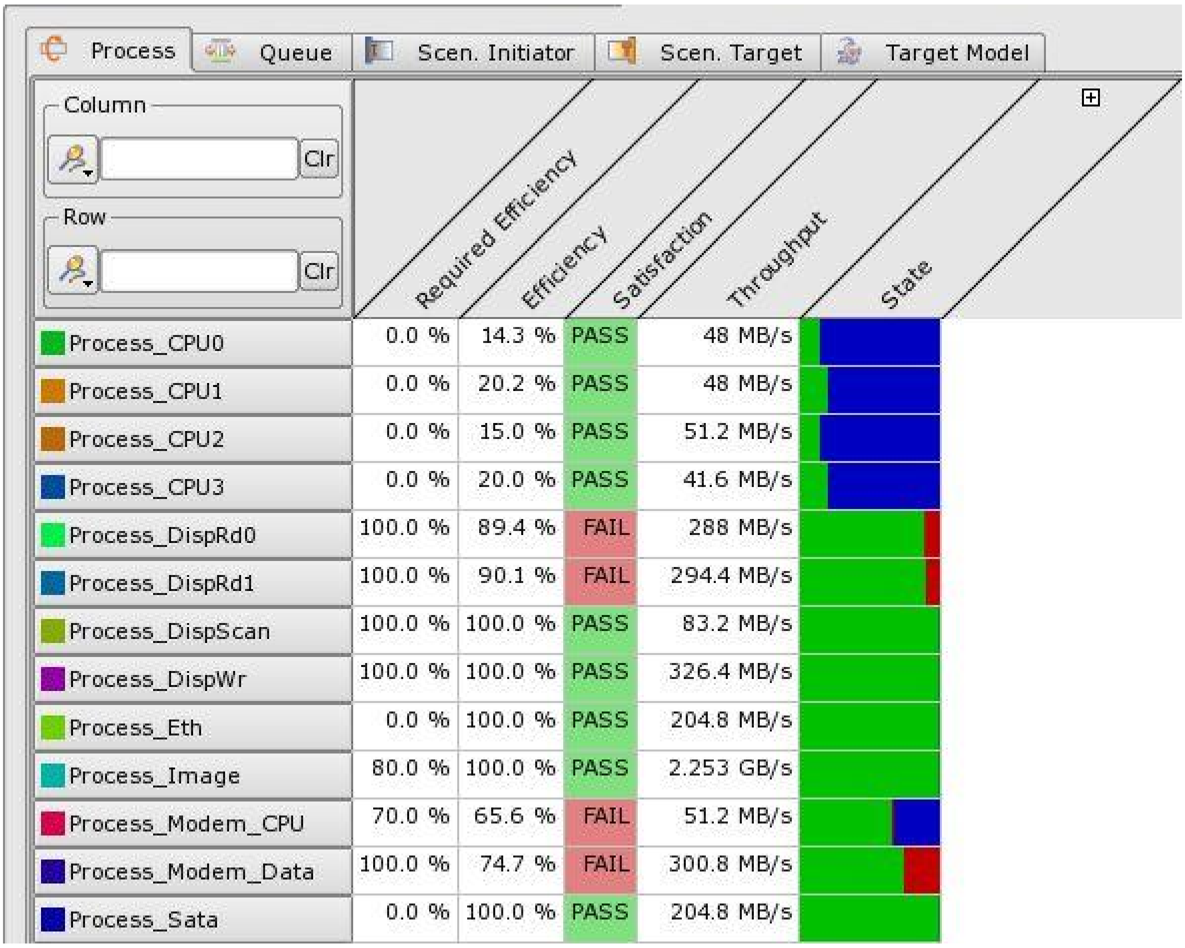
1. Import the new refined scenario
   i. From the Tools menu invoke the command FlexExploration -> Import Script and select your refined script in the popup window


File
Edit
View
Tools
Windows
Help
FlexVerifier
RTL output
External links
FlexPlanner
FlexExploration
Architecture Editor: /architecture.s
Spe
Syst
arch
step1
structure
Export Scenario...
Export PV Interconnect...
Export AV Interconnect...
Export VV Interconnect...
Export AV w/ Specific Sockets ...
Export QSB Wrapper ...
Export Platform...
Export Platform, Simulate and load results ...
Import VCD ...
Import Script ...
Export Skeleton Script ...

   ii. Once the new scenario is imported, from the Tools menu select FlexExploration -> Export Platform, Simulate and load results, select the imported scenario and the step1 architecture from the popup windows and leave the simulation time at the default value
   iii. A new ñimportò folder is created including simulation results providing details for the Scenario and the Architecture



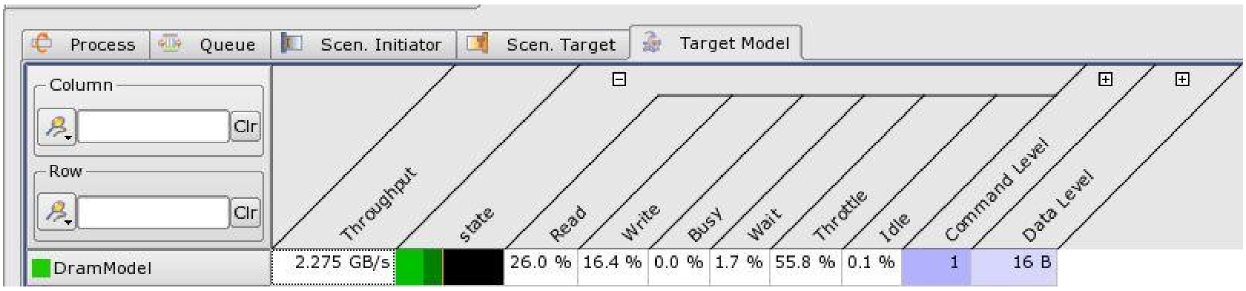
iv. Open the ScenarioResult and inspect the Process tab panel:



	Required Efficiency	Efficiency	Satisfaction	Throughput	State
🟩 Process_CPU0	0.0 %	14.3 %	PASS	48 MB/s	
🟧 Process_CPU1	0.0 %	20.2 %	PASS	48 MB/s	
🟧 Process_CPU2	0.0 %	15.0 %	PASS	51.2 MB/s	
🟦 Process_CPU3	0.0 %	20.0 %	PASS	41.6 MB/s	
🟩 Process_DispRd0	100.0 %	89.4 %	FAIL	288 MB/s	
🟦 Process_DispRd1	100.0 %	90.1 %	FAIL	294.4 MB/s	
🟩 Process_DispScan	100.0 %	100.0 %	PASS	83.2 MB/s	
🟪 Process_DispWr	100.0 %	100.0 %	PASS	326.4 MB/s	
🟩 Process_Eth	0.0 %	100.0 %	PASS	204.8 MB/s	
🟦 Process_Image	80.0 %	100.0 %	PASS	2.253 GB/s	
🟥 Process_Modem_CPU	70.0 %	65.6 %	FAIL	51.2 MB/s	
🟦 Process_Modem_Data	100.0 %	74.7 %	FAIL	300.8 MB/s	
🟦 Process_Sata	0.0 %	100.0 %	PASS	204.8 MB/s	

a. What is going wrong there?

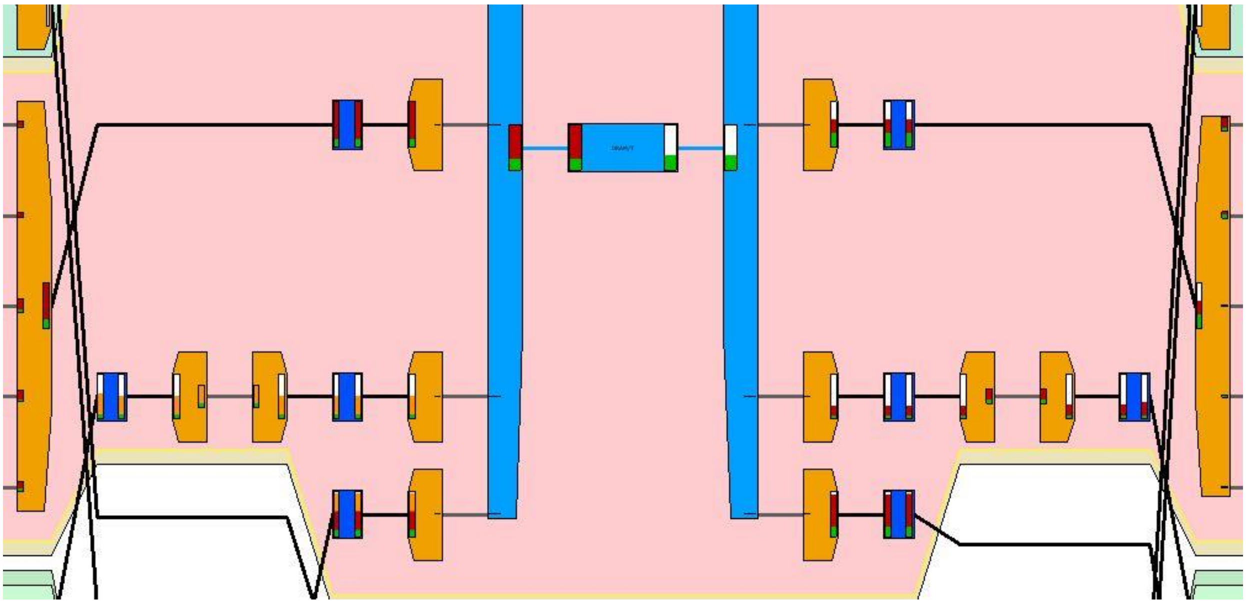
v. Inspect the Target Model tab panel and inspect the DramModel state:



	Throughput	state	Read	Write	Busy	Wait	Throttle	Idle	Command Level	Data Level
🟩 DramModel	2.275 GB/s		26.0 %	16.4 %	0.0 %	1.7 %	55.8 %	0.1 %	1	16 B

a. There is a lot of black in the state bar-graph: what is the throttle and why it is so high?

vi. Now, still within the import folder, open the ArchitectureResult and inspect the Topology panel



a. What does it mean the red color in the vertical bar-graphs

b. And the orange parts?

c. How can I refine this design to match the required efficiencies?

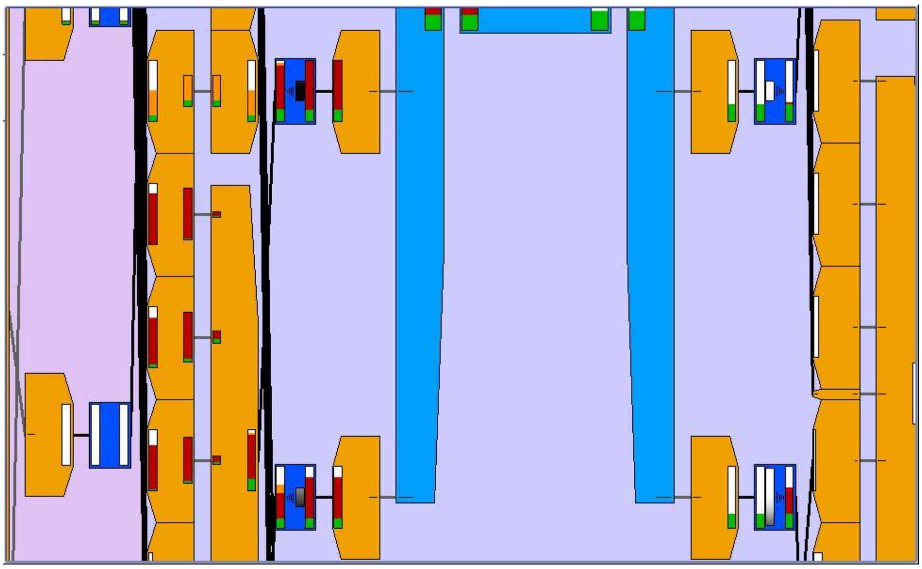
Objective: To refine a NoC architecture and match performance requirements

The purpose of this lab is to apply NoC refinement guidelines to improve the NoC bandwidth usage, increase the DRAM efficiency and guarantee the performance requirements of IPs (bandwidth and latency).

Procedure:

PART 1: create and refine step2 architecture removing congestion and burst holes

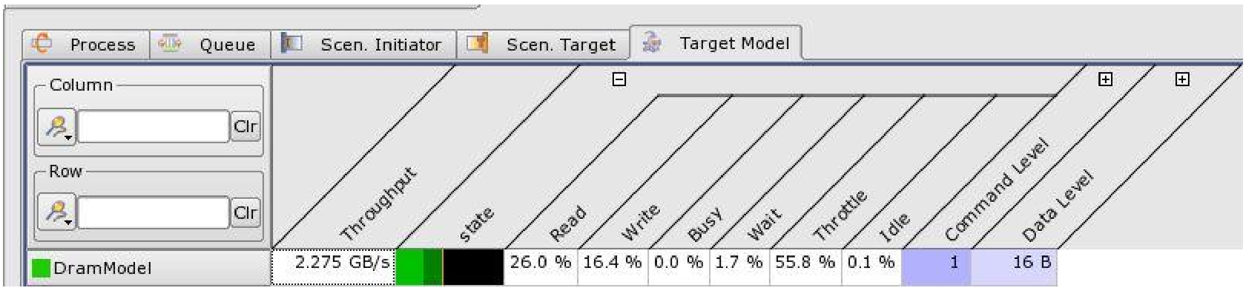
1. Re-open the Lab8 file in the FlexNoC GUI by entering ñ**FlexNoC –p FlexNoC_Lab8.pdd&**ò at the command prompt.
2. In the FlexNoC project panel select the ôstep1ö architecture and duplicate it (right_click->Duplicate)
3. Inspect the Architecture results of Lab8 simulation (regenerate it in case you do not have it anymore simulating step1 with your imported scenario)
4. In step2 architecture
   i. Remove congestion in the response network responsible of DRAM performance decrease
      a. Inspect the Architecture result topology view (step 3.) and, for each link in response network where the BUSY state (RED) is high, enable FIFO buffering in the corresponding link of ôstep2ö architecture; set each fifo size to allow storing one or two largest packets flowing thorough this link;
   ii. Remove bubbles from traffic feeding the DRAM
      a. Similar to step 4.i.a but now related to links in the request network (just in front of the scheduler) where the WAIT states (ORANGE) are present; enable RATE_ADAPTER buffering function
         1. To limit latency insertion due to rate adaptation, specify a suitable defaultThroughputRatio: this has to be set as the ratio ThroughputOut/ThroughputIn where ThroughputOut is the bandwidht peak at the rate adapter output (clk * bus_width) and ThroughputIn is the data throughput of the sending IP.
5. Simulate this new refined ôstep2ö architecture with your scenario script (Tools->FlexExploration->Export Platform, Simulate and Load Results)
   i. inspect the topology view of this new architecture results



   a. did the WAIT states disappear from the output bar-graphs of the schedulerÆs request links as shown above?
   b. did the BUSY states disappear from the input bar-graphs of the schedulerÆs response links as shown above?

PART 2: refine step2 architecture's QoS instructing the memory scheduler

6. Remember on Lab8, the first exploration was showing the Target Model suffering of a considerable throttling time (the time spent by the controller in internal DRAM penalties)



	Throughput	state	Read	Write	Busy	Wait	Throttle	Idle	Command Level	Data Level
🟩 DramModel	2.275 GB/s	🟩⬛	26.0 %	16.4 %	0.0 %	1.7 %	55.8 %	0.1 %	1	16 B

7. Letôs try to reduce as much as possible the DDR controller penalties and so enhance the DRAM efficiency suitably configuring the FlexMem scheduler
   i. Open óstep2ô architecture, go to the Performance/Scheduler tab panel, click on the criteria->customize and in the pop up menu add two new arbitration criteria in order to improve DRAM memory effciency
      a. Hint: try enabling BANK_IDLE_PAGE_HIT and/or RW criteria
   ii. Re-simulate this óstep2ô architecture with your scenario: does the target throttle decrease?
   iii. Are all the efficiencies matched in the óProcessô result panel?

PART 3: create and refine step3 architecture to improve CPU latency and match bandwidth requirements introducing QoS

8. Letôs try now to introduce a QoS scheme based on pressure/hurry priority and capable of ensuring real-time answering for traffic to/from video, guarantying long-term throughput for display and low latency figure for both cpu and modem
   i. The idea consists in classifying initiator traffics according to the following table

socket	Traffic class	QoS generator	Pressure (Max/Min)
display.scan	RT-GT		2 / 1
display.rd0	RT-GT		2 / 1
display.rd1	RT-GT		2 / 1
display.wr	RT-GT		2 / 1
image.shade	RT-GT		2 / 1
image.texture	RT-GT		2 / 1
image.motion	RT-GT		2 / 1
image.raster	RT-GT		2 / 1
cpu.inst	LL		3 / 1
cpu.data	LL		3 / 1
modem.cpu	GT		3 / 3
modem.data	GT		3 / 3

      a. Real-time guaranteed throughput sockets (RT-GT) will see their priority being increased dynamically to a maximum value whenever it is required, either directly through the hurry sideband input (for those IPs that can drive it) or through a bandwidth regulator;
      b. Modem, supposed to be guaranteed throughput (GT), will always have the highest priority;
      c. Low latency (LL) traffic IPs will have lower fixed priority so, when nobody in the other classes is raising its pressure for urgent bandwidth or accessing to the shared resource, they will be served first;
   ii. in the FlexNoC project panel create óstep3ô architecture from óstep2ô (select this last and duplicate it))
   iii. Introduce QoS generators of type FIXED for LL traffic IPs as shown in the previous table

a. Open óstep3ô architecture and in the Generic NIU tab panel enable a FIXED QoS generator for each one of the three mentioned sockets

b. Set the readUrgency to 3 and writeUrgency to 1 (higher priority for reads)

iv. Similarly introduce QoS generators of type FIXED for modem.data, modem.cpu

a. Set both the readUrgency and writeUrgency to 3 (always higher priority)

v. Assuming image.x IPs cannot directly drive the NoC with a priority signal, introduce a QoS generators of type Regulator for each one of them

a. set the priority1 (max priority) to 2 and priority0 to 1

b. set the bandwidth parameter as the 80% of the required value in order not to keep the driven pressure always to the highest value

c. leave the saturation to a value of 1Kbytes

vi. To simulate dynamic increase/decrease of display.x pressure, modify your scenario script

a. For each display queue configure the hurryThreshold parameter in order to let the hurry priority being driven at least to 1 (all the time) and directly increased to 2 whenever the threshold of half the queueôs depth is reached

vii. Finally donôt forget to add three more architecture criteria in the scheduler to take in count the dynamic pressure during arbitration

a. Try setting these three new criteria after the RW one, in the order URG3, URG2, URG1

9. Simulate the updated scenario with the óstep3ô architecture

i. Do the scenario process results show all matching efficiencies?

ii. Check the Queue Full Latency result: are all the CPU achieving the smallest latency figure compared to those of the other IPs?

Objective: to implement a power domain partitioning over a specified SoC

The purpose of this lab is to learn how to define and introduce power domains on an existing SoC and the implement a power organization scheme using power bundles, controllers and disconnect modules.

Procedure:

PART 1: Specify a power domain partitioning

Starting from the SoC/NoC designed in FlexNoC_Power_Lab.pdd file, we want to partition it according to the following power organization scheme

NIU

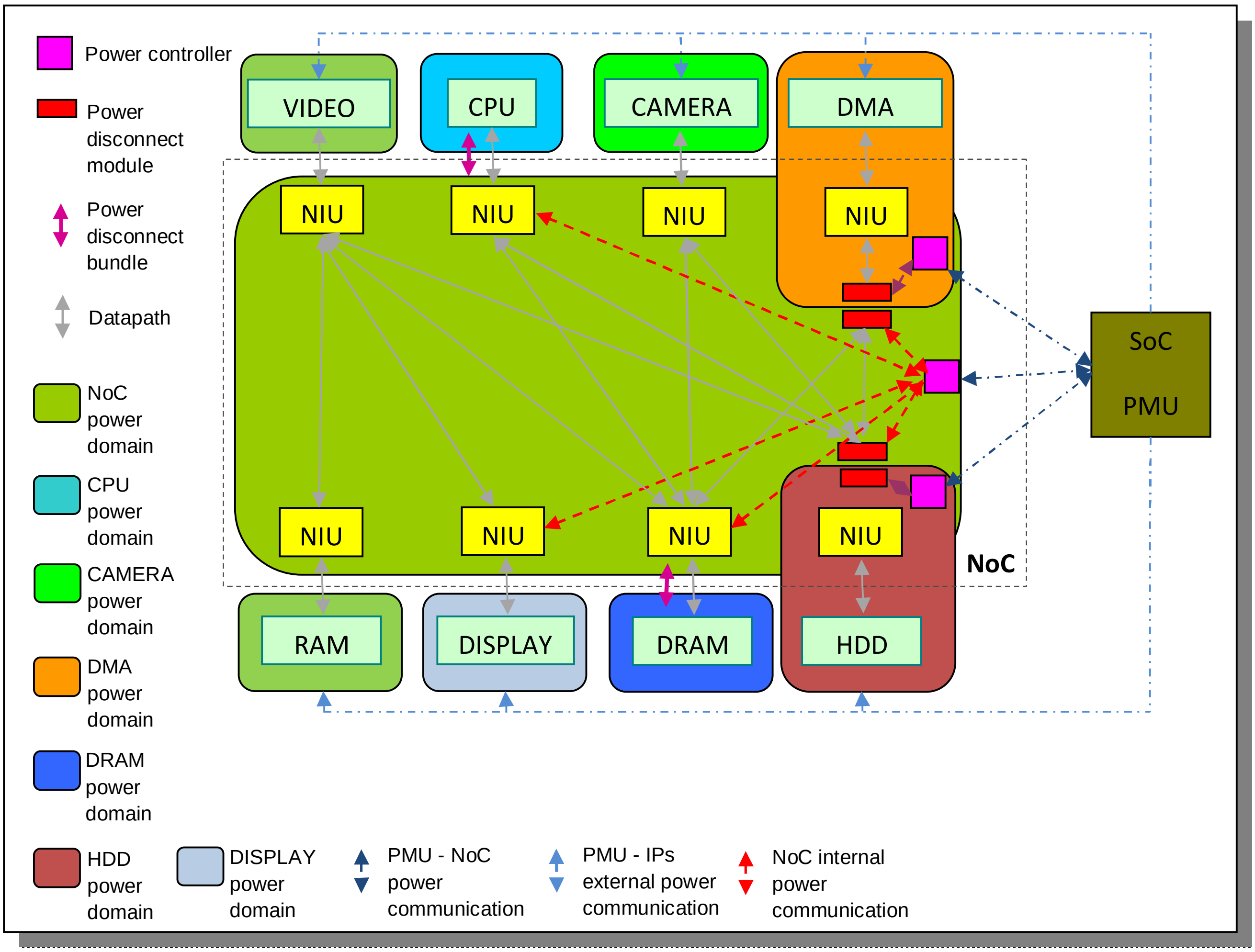


Fig.1 Power domain partitioning

We will define these 7 different power domains shown above in the existing SoC/NoC organization; power saving will occur on each one of them through either clock cut-off or power supply switch-off.

Please note that:

1. All the external IPs have their own dedicated power domain, except for VIDEO and RAM that are in the NoC power domain
2. CPU and DRAM interfaces support each one a power disconnect bundle, a dedicated communication ports allowing IP/NoC power disconnection
3. CAMERA and DISPLAY have each one its own dedicated power domain but they do not support a power disconnect bundle;
4. DMA and HDD are placed each one in a dedicated power domain including also the attached NIU; this kind of power configuration does not require a power bundle at the socket interface between the IP and its NIU
5. Finally, VIDEO and RAM are placed in the same power domain as the NoC

IMPORTANT: it is the PMU responsibility (SoC Power Management Unit, external to the NoC) to externally communicate with every IP not supporting a power disconnect bundle and ensuring

I. fencing of further transaction sending (traffic to be stopped) and draining of pending transactions (all pending responses to be received back)

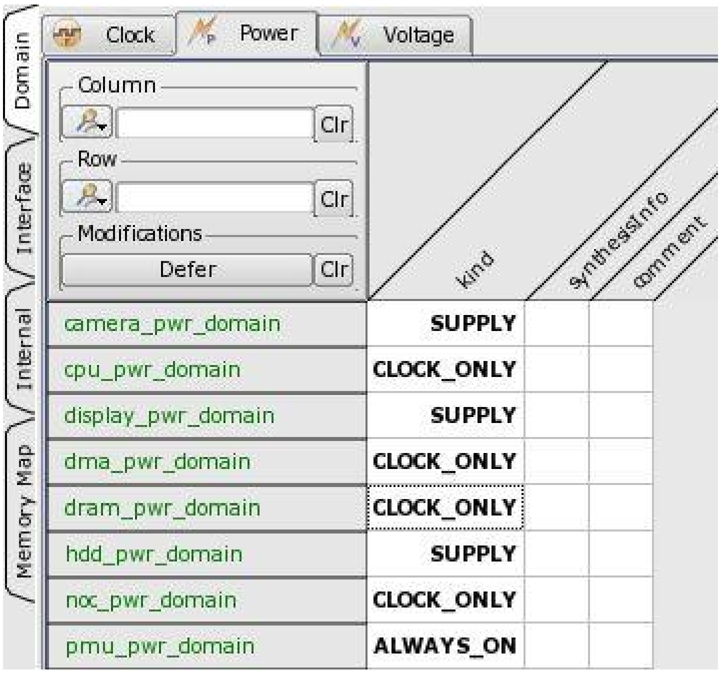
II. completion of all the pending responses within every target IP (DISPLAY,DRAM) before to be switched off

1. Open the power Lab file in the FlexNoC GUI by entering ñ**FlexNoC –p FlexNoC_Power_Lab.pdd&**ò at the command prompt.

2. Open the Specification Editor clicking twice on the ñSpecificationò item in the Project Tree

3. Specifying power domains

   i. In the ñDomainò view select the ñPowerò page
   ii. Create a new Power domain (right_click->New->Power) and give it the name of ñnoc_pwr_domainò
      a. Select ñCLOCK_ONLYò in the kind column; this will inform the tool that this domain will only implement clock cutting (no power supply cut-off)
   iii. As already done at step 3.ii, letôs now create new power domains respectively for CPU, DMA, CAMERA, DISPLAY, DRAM and HDD
   iv. Please set ñkindò to CLOCK_ONLY for all the IPs except HDD, CAMERA and DISPLAY for which SUPPLY management is required
   v. Finally letôs introduce a power domain for the PMU and letôs set it as being ñALWAYS_ONò



	kind	synthesisInfo	comment
camera_pwr_domain	SUPPLY		
cpu_pwr_domain	CLOCK_ONLY		
display_pwr_domain	SUPPLY		
dma_pwr_domain	CLOCK_ONLY		
dram_pwr_domain	CLOCK_ONLY		
hdd_pwr_domain	SUPPLY		
noc_pwr_domain	CLOCK_ONLY		
pmu_pwr_domain	ALWAYS_ON		

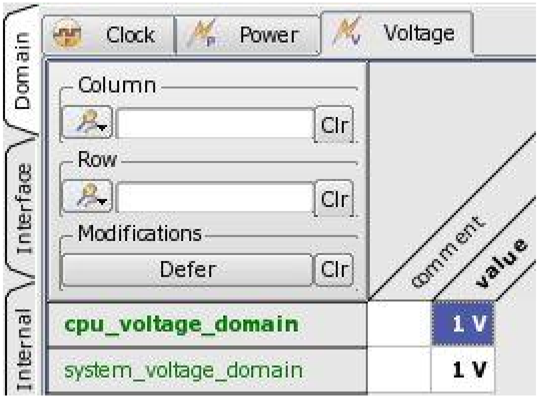
4. Defining the voltage domain

i. When power domains are specified into the NoC, they need to be associated with a voltage domain.

ii. Go to Domain view / Voltage page

iii. Letôs create two voltage domains, one for the cpu and one for the rest of the system. Right click, new voltage. Name it system_voltage_domain and set the value to 1 V.

iv. Select it, right click then ñduplicateò, rename it as cpu_voltage_domain and set the value to 1 V



	comment	value
cpu_voltage_domain		1 V
system_voltage_domain		1 V

5. Associating clock managers to power domains

i. Preface: clock managers distribute clocks and resets to NoC components (NIUs and transport components) within clock domains. Please note that NIUs are synchronous to their connected IPs and also that every NIU interface (socket) is driven/associated of course by/to a clock (look at ñInterfaceò view). Associating a clock manager to a power domain means that all the NoC components driven by this clock manager must belong to the specified power domain and so they will have the same power state (on-off).

ii. Letôs proceed by associating every clock manager to a power domain; this important information will tell the tool how clocks are distributed in the power domain partitioning scheme. For this reason letôs select the ñDomainò view and the ñClockò page in the Specification editor

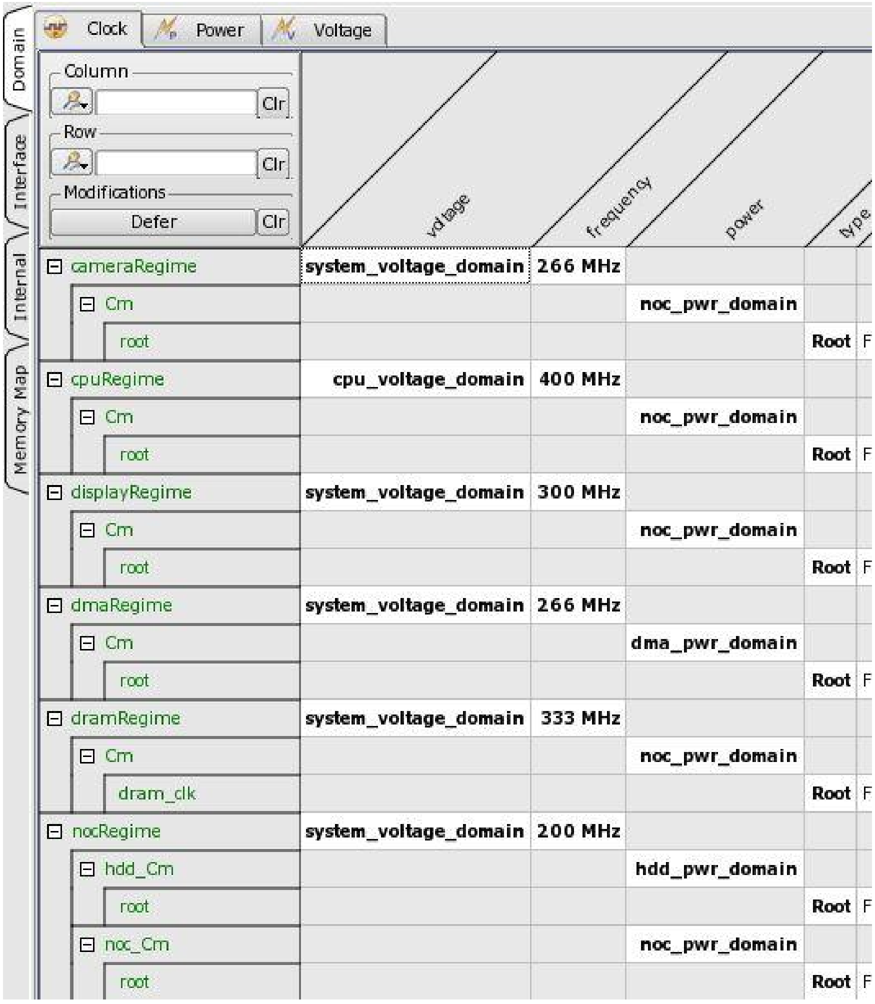
a. According to Fig.1, the CPU, DRAM, SRAM, VIDEO, DISPLAY and CAMERA **NIUs** are supposed to belong to the NoC power domain, for this reason in column ñpowerò letôs select noc_pwr_domain for all of the associated clock manager

b. Of course, NoC transport components driven by nocRegime.noc_Cm clock managerôs output belong to the NoC power domain so we can do the same setting for this clock manager

c. Conversely, the DMA and HDD NIUs belong respectively to DMA and HDD IP power domains, so dma_pwr_domain and hdd_pwr_domain must instead be selected in the power column.

d. Note VIDEO and SRAM do not have a specific clock manager as they are supposed to be in the same clock and power domain than the main NoC

iii. Finally, set the voltage domain to system_voltage_domain for all of the regimes except for the cpu one, for which cpu_voltage_domain has to be selected



	voltage	frequency	power	type	
cameraRegime	system_voltage_domain	266 MHz			
Cm			noc_pwr_domain		
root				Root	F
cpuRegime	cpu_voltage_domain	400 MHz			
Cm			noc_pwr_domain		
root				Root	F
displayRegime	system_voltage_domain	300 MHz			
Cm			noc_pwr_domain		
root				Root	F
dmaRegime	system_voltage_domain	266 MHz			
Cm			dma_pwr_domain		
root				Root	F
dramRegime	system_voltage_domain	333 MHz			
Cm			noc_pwr_domain		
dram_clk				Root	F
nocRegime	system_voltage_domain	200 MHz			
hdd_Cm			hdd_pwr_domain		
root				Root	F
noc_Cm			noc_pwr_domain		
root				Root	F

6. Associating IPs to power domains
   i. Now we need to refine our power specification by telling the tool what power domain every IP belongs to:
      a. In the ñInterfaceò view, select the Socket page and expand the ñpowerò parameter sub-group
      b. Each IP belongs to a specific power domain; letôs select the right power domain association in the ñIPpowerDomainò column (CPU to cpu_pwr_domain, DMA to dma_pwr_domain, etc)

c. setting the ñ**disconnect**ò parameter

1. This parameter can force the instance of logic for power communication/disconnection, it can have the following values

Disconnect value	Comment
None	No power bundle, no disconnect logic and active indication, no fencing/draining.
SYSTEM	Similar to previous one but required when the IP does not share the same power domain than the NIU; power communication **must** be provided externally to the NoC between the SoC Power Manager and the IP; fencing and draining **must** be performed by the initiator IP when the SoC Power Manager requests either IP or the NoC (or both) to be switched off
None with flush	Only at initiator side: no power bundle, no disconnect logic and active indication, no fencing Draining is only performed for posted writes The initiator IP **must** perform fencing and draining whenever the PMU notifies that either the IP itself or the attached NoC has to be switched off (before Idle request is sent to the NoC power controller).
SYSTEM with flush	Similar to previous one but available only when the initiator IP and its NIU do not share the same power domain
ARTERIS / OCP	A power bundle with the selected power protocol is provided at the socket interface to negotiate the power state of the socket connection ***Initiator side***: fencing and draining **must** be performed ***by the initiator* IP** when a NoC disconnection request occurs through the power bundle ***Target side***: fencing and draining are performed ***by the target NIU***; the target IP **must** guarantee response completion before to disconnect; disconnect policy (Error, Stall) can be chosen statically at configuration time or dynamically through a suitable input; Active output indication is provided to either indicate socket activity (when it is not disconnected) or presence of a pending request when Stall mode is selected and the target IP is disconnected
MULTIPORT_ARTERIS/ OCP	Same as previous but only for multi-socket target NIUs
None with protection	Only at target side: no power bundle, but fencing and draining performed ***by the target NIU*** under control of the NoC power controller The target IP **must** guarantee response completion before to be switched off
SYSTEM with protection	Similar to previous one but only when the targetIP and its NIU do not share the same power domain; active indication is available

It has to be set accordingly to the following rules

a. <u>Initiator side</u>

Conditions	Disconnect choice
the initiator NIU and the IP are in the same ClockOnly/SUPPLY power domain	***None*** or ***None with flush*** if the socket has posted writes.
the initiator NIU is on an AlwaysOn power domain	***None***
the initiator NIU and the IP are on a different ClockOnly/SUPPLY power domain	• ***ARTERIS*** • ***OCP*** • ***OCP_TI*** • ***SYSTEM*** or ***SYSTEM with flush*** if the socket has posted writes

b. <u>Target side</u>

Conditions	Disconnect choice
the target NIU and the IP are in the same ClockOnly/SUPPLY power domain	***None*** or ***None with protection***
the target NIU is on a different AlwaysOn power domain	***None***
the target NIU and the targetIP are on a different ClockOnly/SUPPLY power domain	• ***ARTERIS*** • ***OCP*** • ***OCP_TI*** • ***MULTIPORT_ARTERIS/OCP*** if the

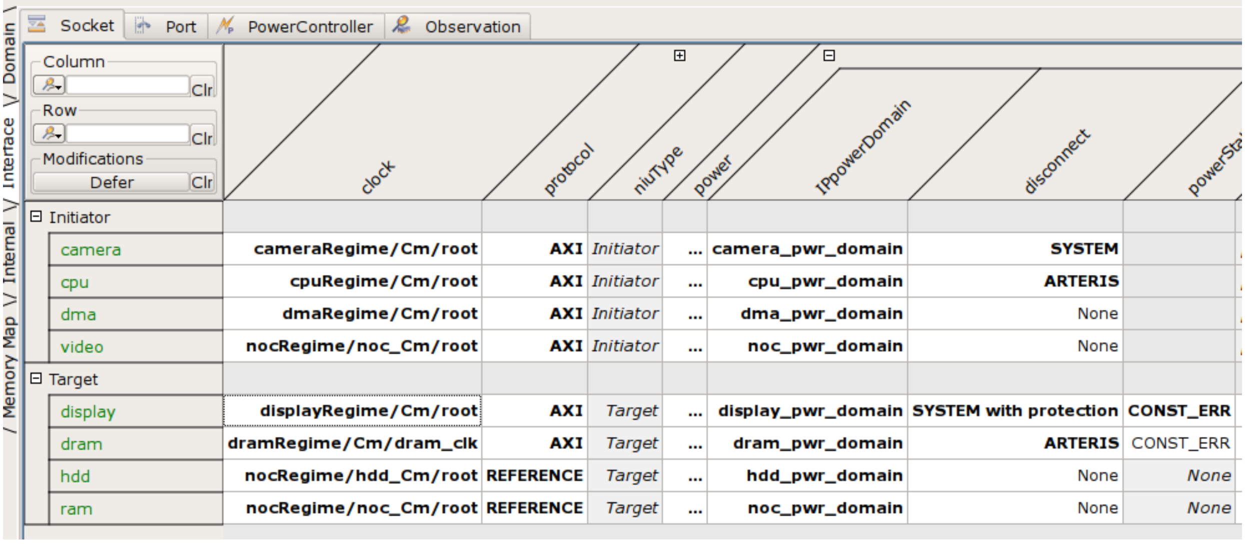
	target is a multisocket. • ***SYSTEM*** or ***SYSTEM with protection***

2. In the columns ñdisconnectò we must specify a disconnect setting for each socket; letôs select the ñARTERISò one (this will provide power bundles made of ñSlvRdyò and ñSocketConò signals) for the CPU and the DRAM, ñSYSTEMò and ñSYSTEM with protectionò respectively for the CAMERA and DISPLAY and ñNoneò for all the others

d. Finally at the ñpowerStallò column we can decide the behavior of the power disconnect module embedded in the DRAM target NIU when the DRAM is disconnected; possible values are

1. CONST_ERR: always reply with error whenever a request arrives while DRAM is off
2. CONST_STALL: always stall any incoming request to DRAM while it is off; PwrActive output is provided to signal to the PMU the controlled domain (DRAM) must wakeup
3. SYNCH: a ñPwrStallò input (to be driven by the PMU) to the target disconnect interface decides the disconnect behavior (ERR, STALL); with this setting it is supposed to be synchronous with the PMU clock domain
4. ASYNCH: similar to previous setting but now ñPwrStallò input is supposed to be asynchronous with the PMU clock domain

e. Letôs leave the default ñCONST_ERRò option.



	clock	protocol	niuType	power	IPpowerDomain	disconnect	powerSta
⊟ Initiator							
camera	cameraRegime/Cm/root	AXI	Initiator	...	camera_pwr_domain	SYSTEM	
cpu	cpuRegime/Cm/root	AXI	Initiator	...	cpu_pwr_domain	ARTERIS	
dma	dmaRegime/Cm/root	AXI	Initiator	...	dma_pwr_domain	None	
video	nocRegime/noc_Cm/root	AXI	Initiator	...	noc_pwr_domain	None	
⊟ Target							
display	displayRegime/Cm/root	AXI	Target	...	display_pwr_domain	SYSTEM with protection	CONST_ERR
dram	dramRegime/Cm/dram_clk	AXI	Target	...	dram_pwr_domain	ARTERIS	CONST_ERR
hdd	nocRegime/hdd_Cm/root	REFERENCE	Target	...	hdd_pwr_domain	None	None
ram	nocRegime/noc_Cm/root	REFERENCE	Target	...	noc_pwr_domain	None	None

7. Power controller settings

i. Each power domain which is not of ALWAYS_ON type and containing NoC units is controlled by a power controller. The controller distributes Idle requests from the PMU to the power domain disconnect components (transport disconnects and NIUs) and centralizes IdleAck from the disconnect components/interfaces to the PMU. From Fig.1 we can see that the HDD, NoC and DMA power domains includes a power controller.

ii. In the ñInterfaceò view letôs select the ñPowerControllerò page

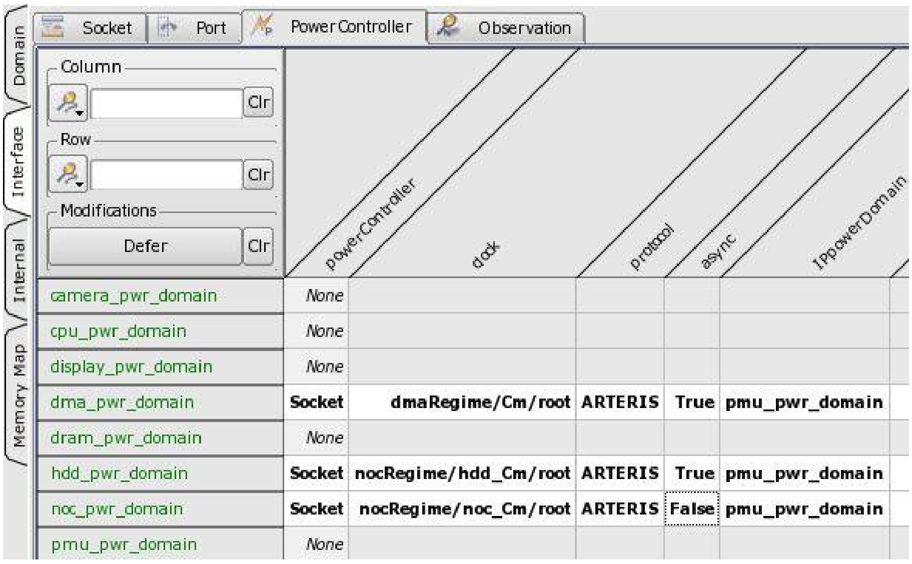
a. For each one of the DMA, HDD and NoC power domains letôs enable a power controller by setting the ñpowerControllerò parameter to Socket.
There are 2 possibilities :
   1. Socket. In this case IdleReq, IdleAck and Idle signals will be present on the NoC interface ports for connection to the power manager.
   2. Software. This option can only be used if there are only targets to be disconnected in the power domain. In this case, there will be a IdleReq/IdleAck register pair per slave. IdleReq is a software accessible writable register, and IdleAck is a readable register. When IdleReq is set to 1 and IdleAck status is 1, the corresponding target can be disconnected.

b. Letôs select the clock for each power controller unit as shown in the figure below

c. Finally letôs choose the power protocol interface for the bundle between each power controller and the PMU: please select the ñARTERISò one (protocol bundle provided with IdleReq, IdleAck and Idle signals)

d. In the ñIPpowerDomainò column letôs choose the power domain for the PMU setting ñpmu_pwr_domainò for the whole column.

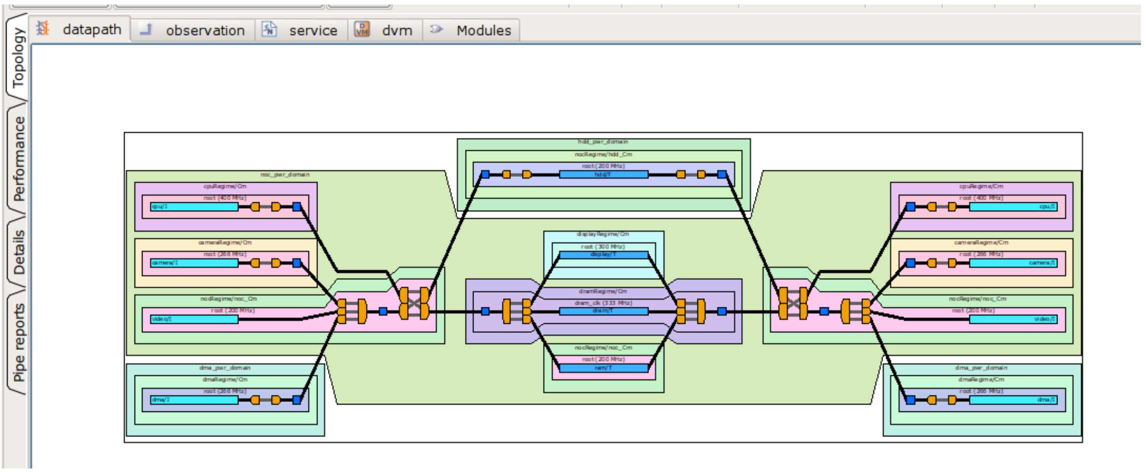
e. Finally we need to inform the tool if re-synchronization is required between power controllers and the PMU by setting the ñasynchò parameter either to True or to False. For this letôs suppose the PMU is synchronous to the NoC regime and asynchronous with all the others.



	powerController	clock	protocol	async	IPpowerDomain
camera_pwr_domain	None				
cpu_pwr_domain	None				
display_pwr_domain	None				
dma_pwr_domain	Socket	dmaRegime/Cm/root	ARTERIS	True	pmu_pwr_domain
dram_pwr_domain	None				
hdd_pwr_domain	Socket	nocRegime/hdd_Cm/root	ARTERIS	True	pmu_pwr_domain
noc_pwr_domain	Socket	nocRegime/noc_Cm/root	ARTERIS	False	pmu_pwr_domain
pmu_pwr_domain	None				

PART 2: Architecture view

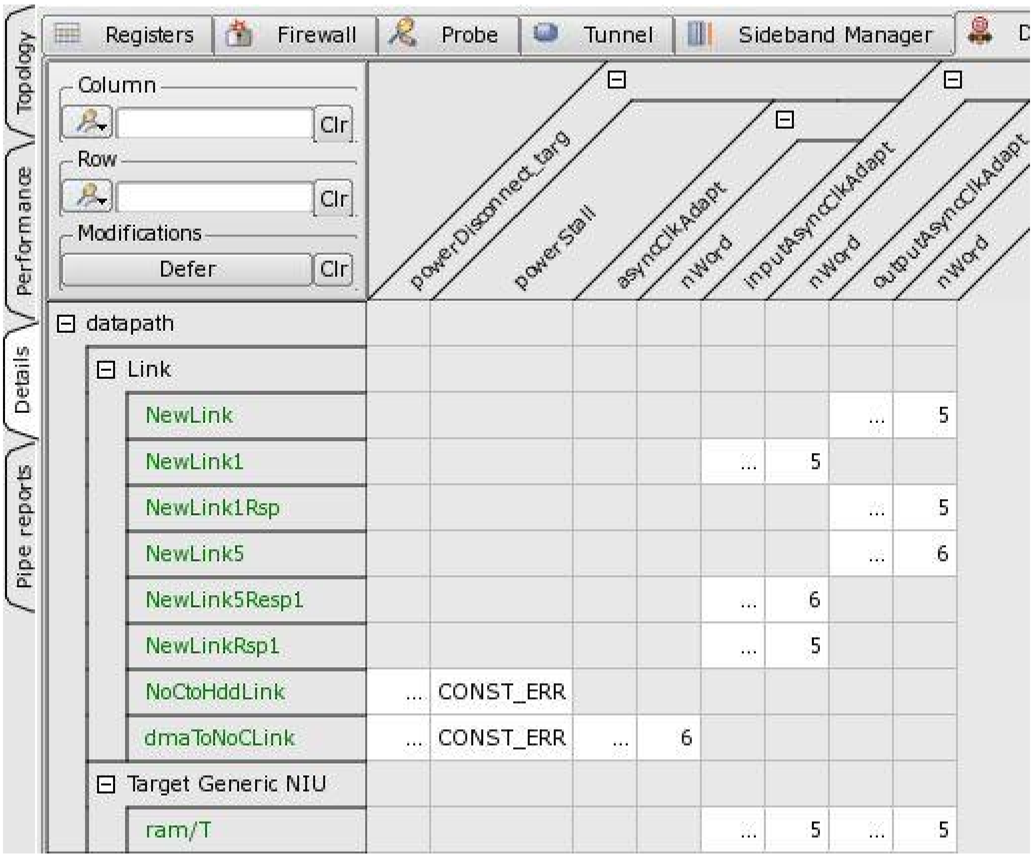
1. In the Project tree letôs open the ñArchitectureò item clicking it twice.



i. You can inspect the power domain partitioning enabling ñPowerò in the ñSector displayò menu

ii. Letôs now have a look to the architecture details and configure the behavior for transport disconnect units

a. In the Architecture browser, select the ñDetailsò major panel and the ñDomain Crossingò tab

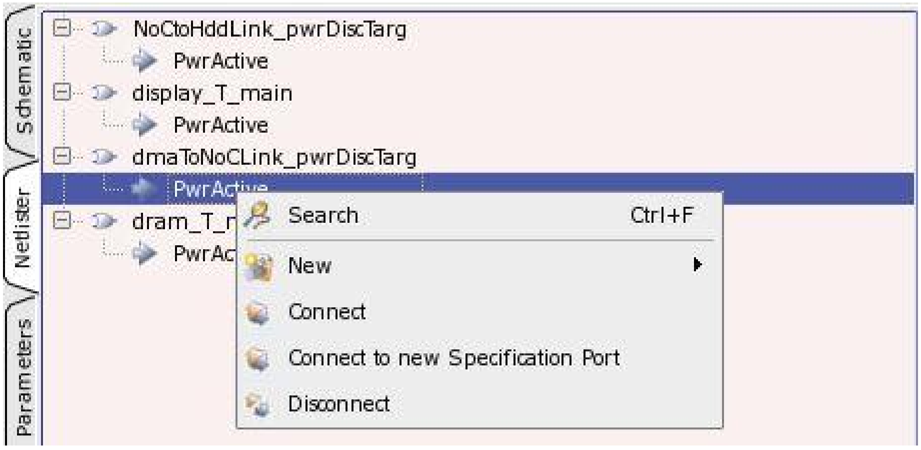


	powerDisconnect_targ	powerStall	asyncClkAdapt	nWord	inputAsyncClkAdapt	nWord	outputAsyncClkAdapt	nWord
⊟ datapath								
⊟ Link								
NewLink							...	5
NewLink1					...	5		
NewLink1Rsp							...	5
NewLink5							...	6
NewLink5Resp1					...	6		
NewLinkRsp1					...	5		
NoCtoHddLink	...	CONST_ERR						
dmaToNoCLink	...	CONST_ERR	...	6				
⊟ Target Generic NIU								
ram/T					...	5	...	5

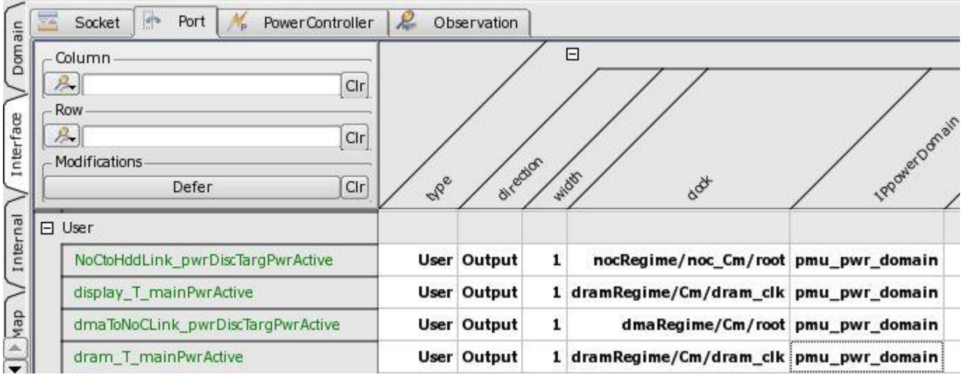
b. Here details for transport links are provided, in particular you can change ñpowerStallò setting for power disconnect modules within ñdmaToNocLinkò and ñNoCtoHddLinkò; letôs keep current setting for which error responses are generated in case of incoming requests toward a power domain that is disconnected.

PART 3: connecting tactical ports

1. Finally, we need to connect Structure tactical outputs like the ñPwrActiveò from the DRAM and the DISPLAY NIUs to the NoC top level; these signals provide activity indication when the connection is ON and a WAKEUP indication to the PMU when the target power domain is OFF.
   i. In the Project panel open the ñStructureò item
   ii. In the ñNetlisterò major panel select every PwrActive output and connect it to a port at the top level through right_click->Connect to New Specification Port



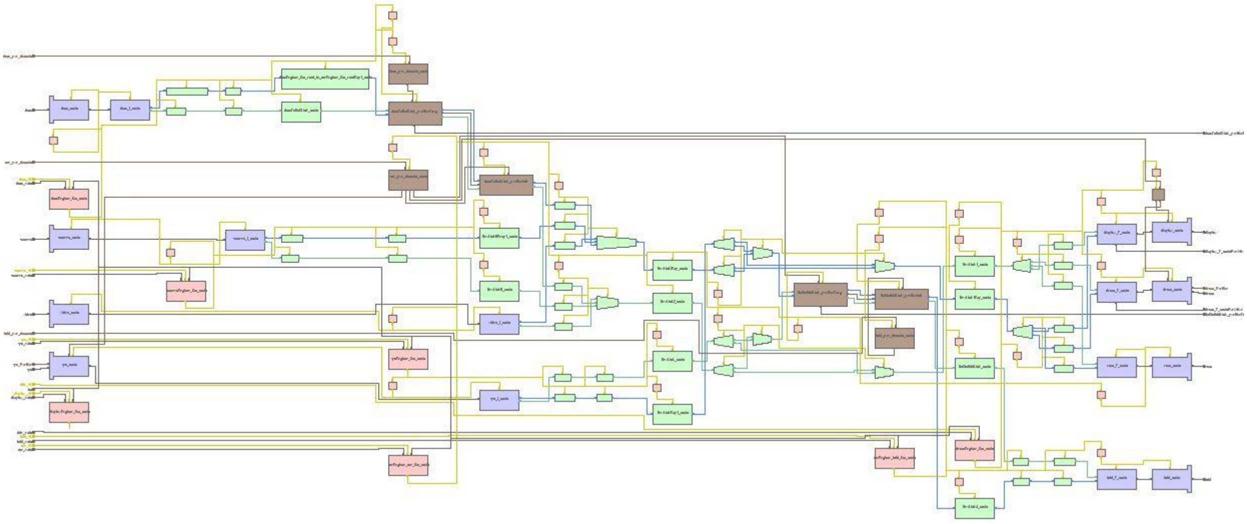
2. Some parameter setting is still missing for the new created ports
   i. In the Specification Editor, open the Interface view at the Port page
   ii. Set the IPpowerDomain parameter to pmu_pwr_domain for every new created Active output port



User	type	direction	width	clock	IPpowerDomain
NoCtoHddLink_pwrDiscTargPwrActive	User	Output	1	nocRegime/noc_Cm/root	pmu_pwr_domain
display_T_mainPwrActive	User	Output	1	dramRegime/Cm/dram_clk	pmu_pwr_domain
dmaToNoCLink_pwrDiscTargPwrActive	User	Output	1	dmaRegime/Cm/root	pmu_pwr_domain
dram_T_mainPwrActive	User	Output	1	dramRegime/Cm/dram_clk	pmu_pwr_domain

PART 4: inspecting the NoC Structure: modules

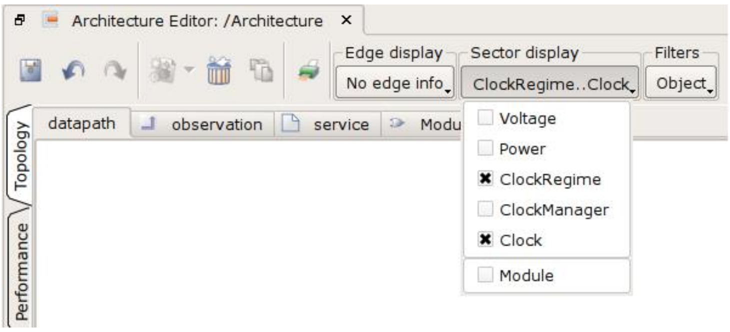
1. In the ñSchematicò major panel of the Structure view a view of the NoC structure is provided:



    i. You can identify the power disconnect modules in the brown boxes

2. For a better design organization letôs introduce one module for DMA, HDD and NoC power domains

    i. In the Architecture view, change the sector display from ñClockRegime..Clockò to ñModuleò (deselect ñClockRegimeò and ñClockò sector display and select the ñModuleò one)

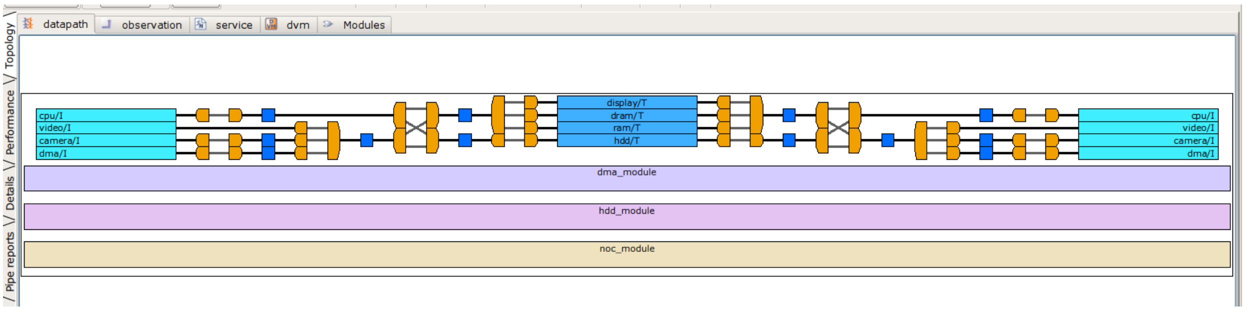


    ii. Letôs create a new module for dma_power_domain

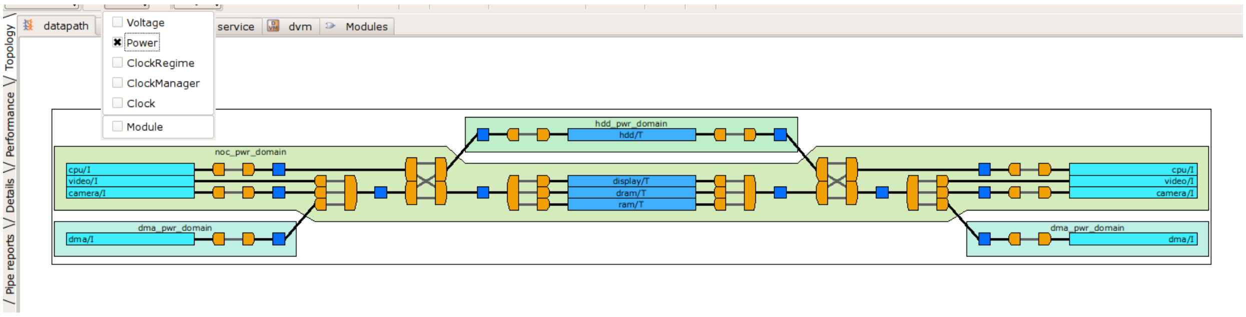
        a. Right-click->New->Module

        b. Give it a suitable name (example ñdma_modò)

    iii. repeat steps a,b to create two more modules respectively for noc_pwr_domain and hdd_power_domain



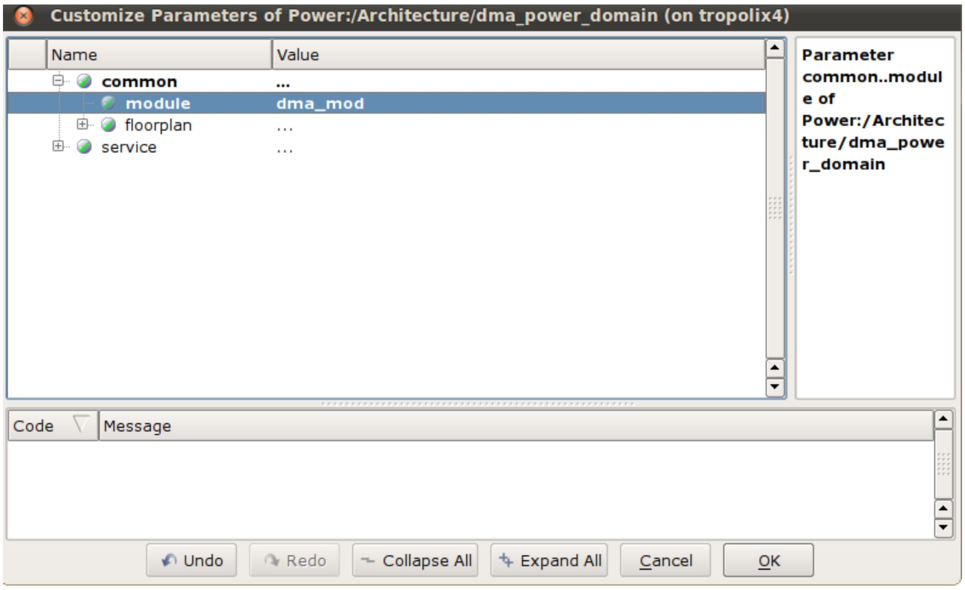
iv. Letôs now change the sector display from ñModuleò to ñPowerò



v. Letôs associate each new created module to the corresponding power domain: letôs start from dma_pwr_domain

a. Select the corresponding power domain and double click on it

1. this opens the related customizer box



Customize Parameters of Power:/Architecture/dma_power_domain (on tropolix4)

Name	Value
common	...
module	dma_mod
floorplan	...
service	...

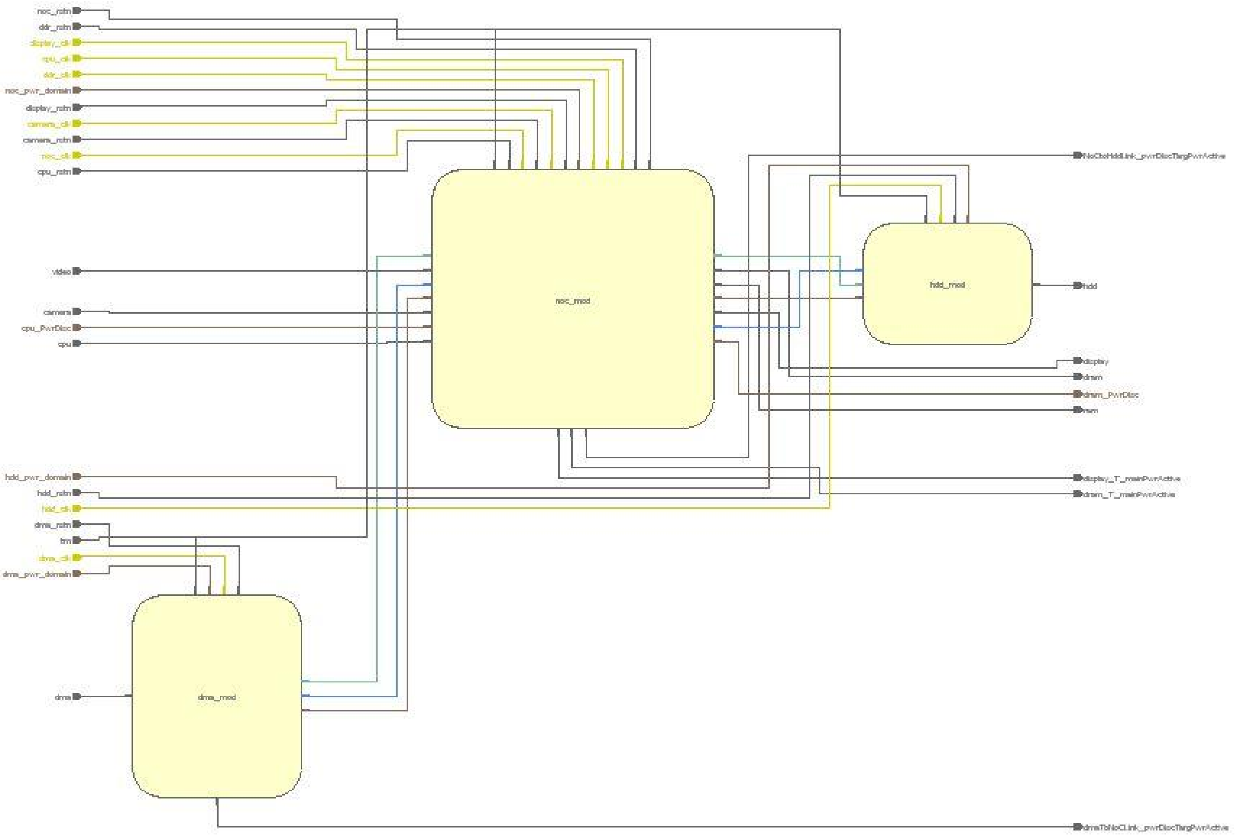
Parameter common..module of Power:/Architecture/dma_power_domain

Code	Message

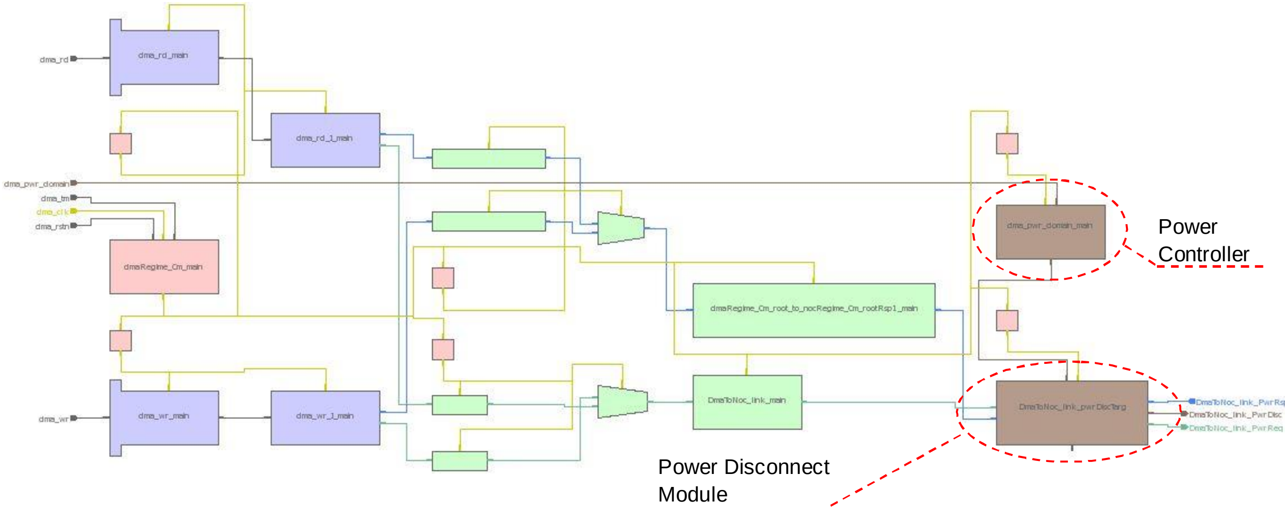
2. select the corresponding module

b. repeat steps 1,2 for noc_pwr_domain and hdd_pwr_domain too.

vi. Finally, letôs inspect the Structure view, it now shows the new three modules



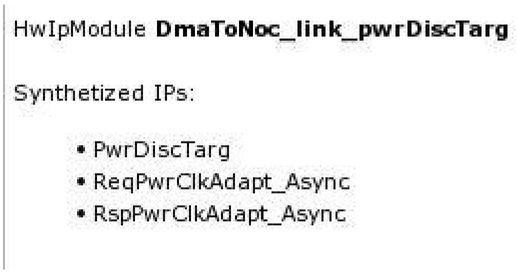
a. Double clicking on one of them, for example the DMA one, we can now easily distinguish power components like the power controller and the power target disconnect modules.



Power Controller

Power Disconnect Module

1. Selecting the target power disconnect modules in the view before and inspecting the ñInformationò panel you can see



HwIpModule **DmaToNoc_link_pwrDiscTarg**

Synthetized IPs:

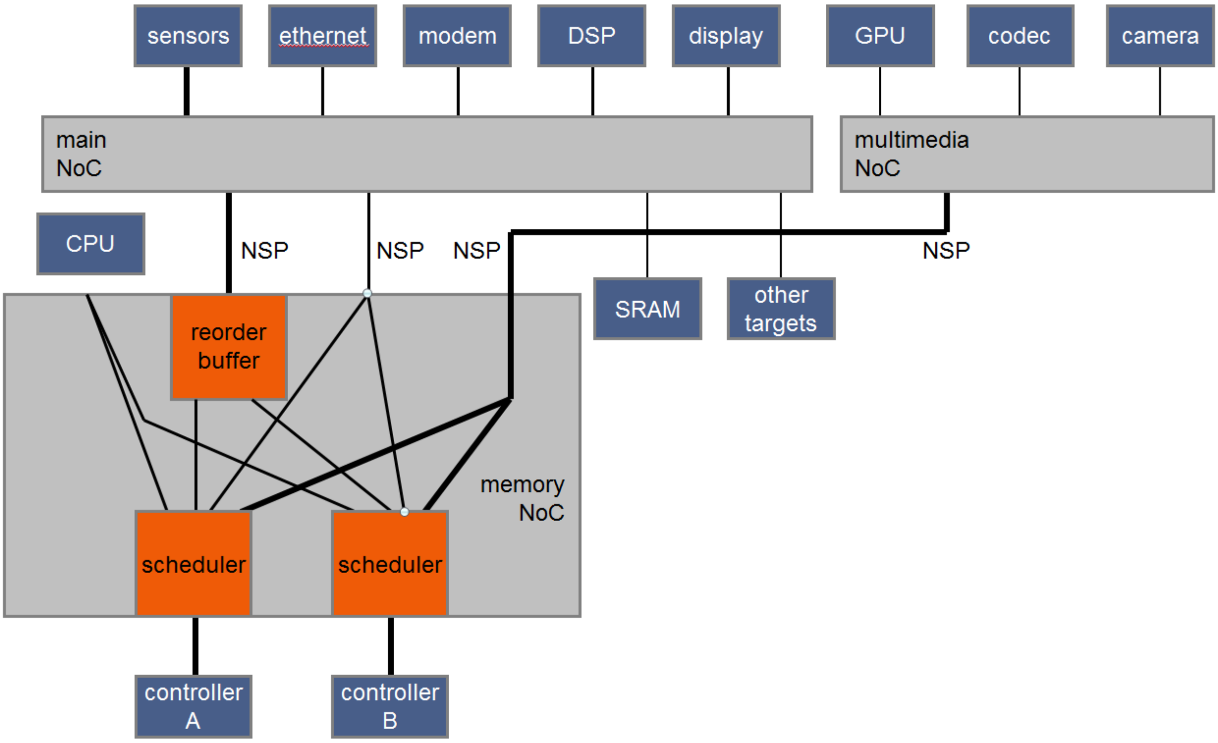
- PwrDiscTarg
- ReqPwrClkAdapt_Async
- RspPwrClkAdapt_Async

It includes the PwrDiscTarg hardware functionality as well the asynchronous clock adapters required at the boundaries of this domain.

End of Power Lab

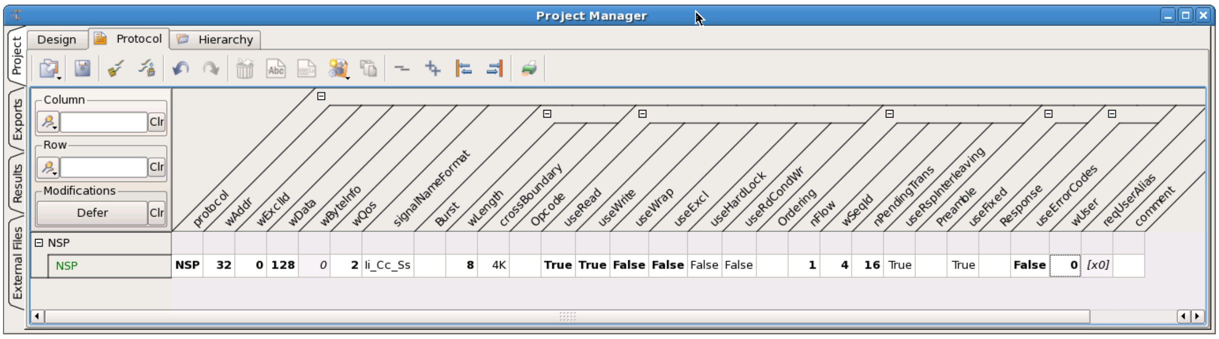
Multi-NoC lab

In this lab you will create a system NoC, multimedia NoC, and memory NoCs for a chip with two interleaved memory interfaces. You will configure a NoC composition with all three. You will connect the NoCs using NSP socket interfaces. You will create a low-latency path from CPU to DRAM.



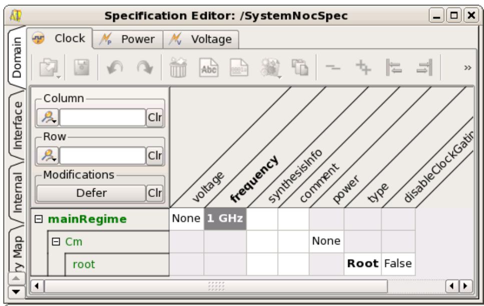
Step 1. Define a Reference Protocol for NSP with

a. 32-bit address
b. no exclusive ID support
c. 128 bit data
d. 2 bits of binary-coded QoS
e. 8 burst length bits
f. support for reads and writes
g. no support for wrapping bursts
h. no support for exclusive access
i. 1 flow
j. 4 sequence ID bits
k. 16 pending transactions
l. no error code support
m. no user bits

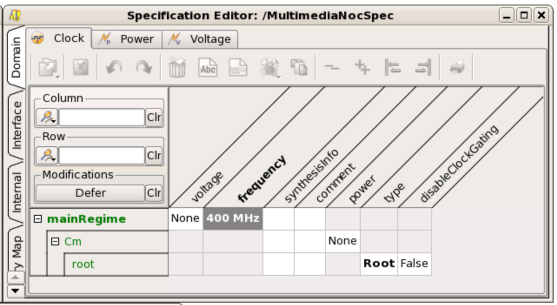


	protocol	wAddr	wExclId	wData	wByteInfo	wQos	signalNameFormat	Burst	wLength	crossBoundary	Opcode	useRead	useWrite	useWrap	useExcl	useHardLock	useRdCondWr	Ordering	nFlow	wSeqId	nPendingTrans	useRspInterleaving	Preamble	useFixed	Response	useErrorCodes	wUser	reqUserAlias	comment
⊟ NSP																													
NSP	**NSP**	**32**	**0**	**128**	*0*	**2**	li_Cc_Ss		**8**	4K		**True**	**True**	**False**	**False**	False	False		**1**	**4**	**16**	True		True		**False**	**0**	*[x0]*	

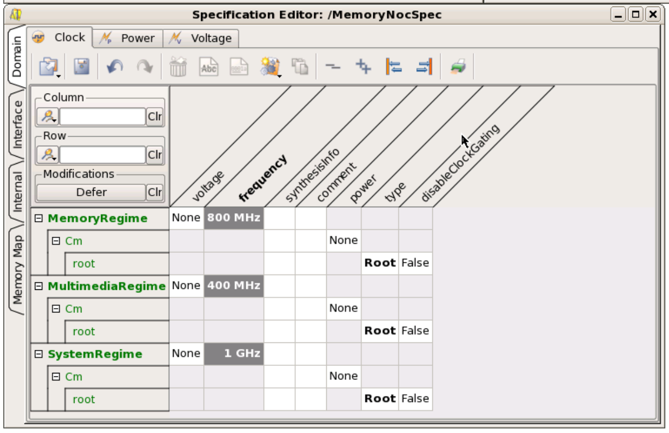
Step 2. Create specifications for a System NoC at 1 GHz, Multimedia NoC at 400 MHz, and Memory NoC with 1 GHz system, 400 MHz multimedia, and 800 MHz memory clock regimes.



	voltage	**frequency**	synthesisInfo	comment	power	type	disableClockGatin
⊟ **mainRegime**	None	**1 GHz**					
⊟ Cm					None		
root						**Root**	False

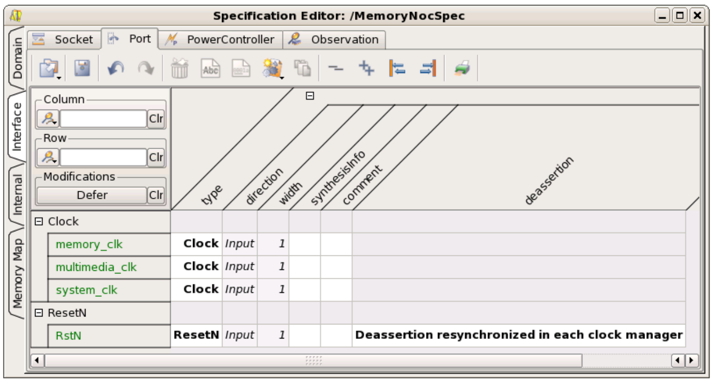


	voltage	**frequency**	synthesisInfo	comment	power	type	disableClockGating
⊟ **mainRegime**	None	**400 MHz**					
⊟ Cm					None		
root						**Root**	False



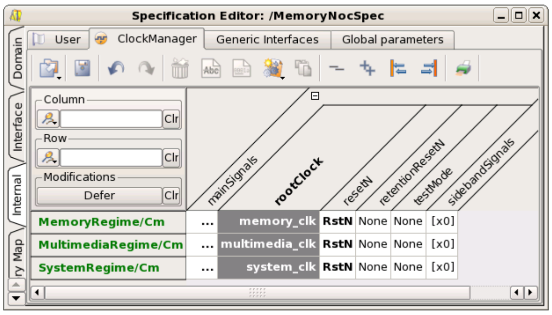
	voltage	**frequency**	synthesisInfo	comment	power	type	disableClockGating
⊟ **MemoryRegime**	None	**800 MHz**					
⊟ Cm					None		
root						**Root**	False
⊟ **MultimediaRegime**	None	**400 MHz**					
⊟ Cm					None		
root						**Root**	False
⊟ **SystemRegime**	None	**1 GHz**					
⊟ Cm					None		
root						**Root**	False

Step 3. Create separate clock ports for each clock regime in the memory NoC.


Specification Editor: /MemoryNocSpec

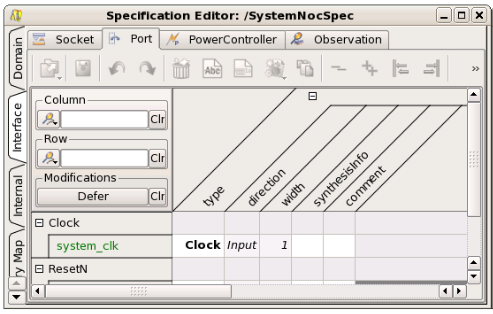
	type	direction	width	synthesisInfo	comment	deassertion
⊟ Clock						
memory_clk	Clock	Input	1			
multimedia_clk	Clock	Input	1			
system_clk	Clock	Input	1			
⊟ ResetN						
RstN	ResetN	Input	1			Deassertion resynchronized in each clock manager

and connect them to corresponding clock managers.

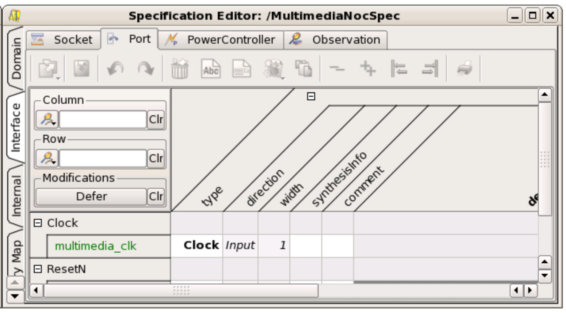

Specification Editor: /MemoryNocSpec

	mainSignals	rootClock	resetN	retentionResetN	testMode	sidebandSignals
MemoryRegime/Cm	...	memory_clk	RstN	None	None	[x0]
MultimediaRegime/Cm	...	multimedia_clk	RstN	None	None	[x0]
SystemRegime/Cm	...	system_clk	RstN	None	None	[x0]

Create similarly named clock ports in the other two NoCs to avoid confusion.

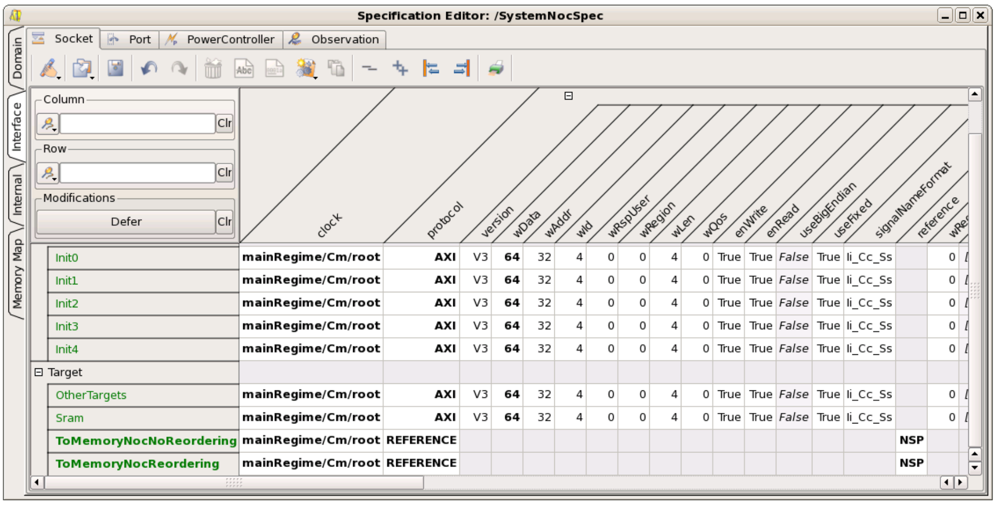

Specification Editor: /SystemNocSpec

	type	direction	width	synthesisInfo	comment
⊟ Clock					
system_clk	Clock	Input	1		
⊟ ResetN					


Specification Editor: /MultimediaNocSpec

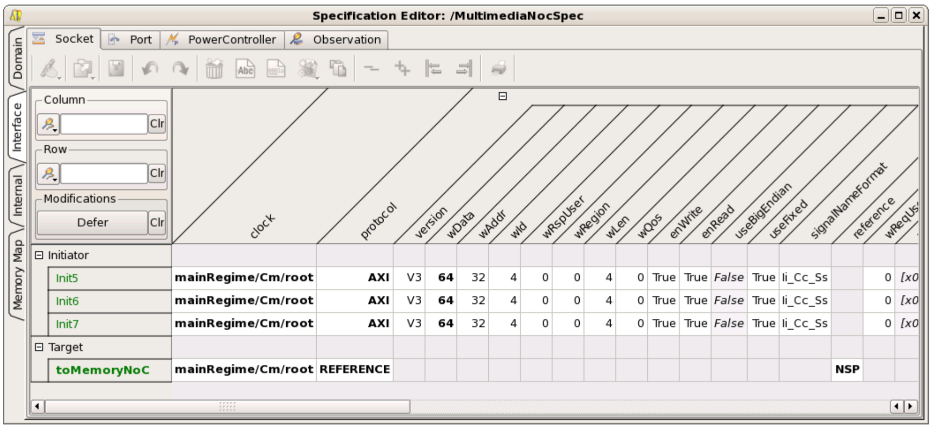
	type	direction	width	synthesisInfo	comment
⊟ Clock					
multimedia_clk	Clock	Input	1		
⊟ ResetN					

Step 4. In the system NoC create five initiator and four target sockets. Give all initiator sockets 64 bit AXI interfaces. Two targets sockets are for SRAM and for other interfaces. The other two target sockets should reference the NSP protocol. One will be connected to a reorder buffer and the other will have a direct connection to memory.


Specification Editor: /SystemNocSpec

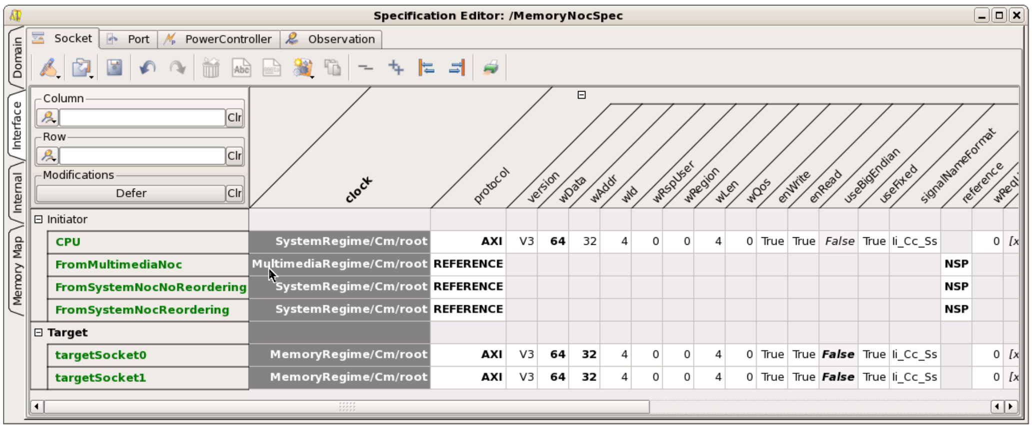
	clock	protocol	version	wData	wAddr	wId	wRspUser	wRegion	wLen	wQos	enWrite	enRead	useBigEndian	useFixed	signalNameFormat	reference	wRe
Init0	mainRegime/Cm/root	AXI	V3	64	32	4	0	0	4	0	True	True	*False*	True	li_Cc_Ss		0
Init1	mainRegime/Cm/root	AXI	V3	64	32	4	0	0	4	0	True	True	*False*	True	li_Cc_Ss		0
Init2	mainRegime/Cm/root	AXI	V3	64	32	4	0	0	4	0	True	True	*False*	True	li_Cc_Ss		0
Init3	mainRegime/Cm/root	AXI	V3	64	32	4	0	0	4	0	True	True	*False*	True	li_Cc_Ss		0
Init4	mainRegime/Cm/root	AXI	V3	64	32	4	0	0	4	0	True	True	*False*	True	li_Cc_Ss		0
⊟ Target																	
OtherTargets	mainRegime/Cm/root	AXI	V3	64	32	4	0	0	4	0	True	True	*False*	True	li_Cc_Ss		0
Sram	mainRegime/Cm/root	AXI	V3	64	32	4	0	0	4	0	True	True	*False*	True	li_Cc_Ss		0
ToMemoryNocNoReordering	mainRegime/Cm/root	REFERENCE														NSP	
ToMemoryNocReordering	mainRegime/Cm/root	REFERENCE														NSP	

In the multimedia NoC create three initiator and one target sockets. Give all initiator sockets 64 bit AXI interfaces. The target socket should reference the NSP protocol.


Specification Editor: /MultimediaNocSpec

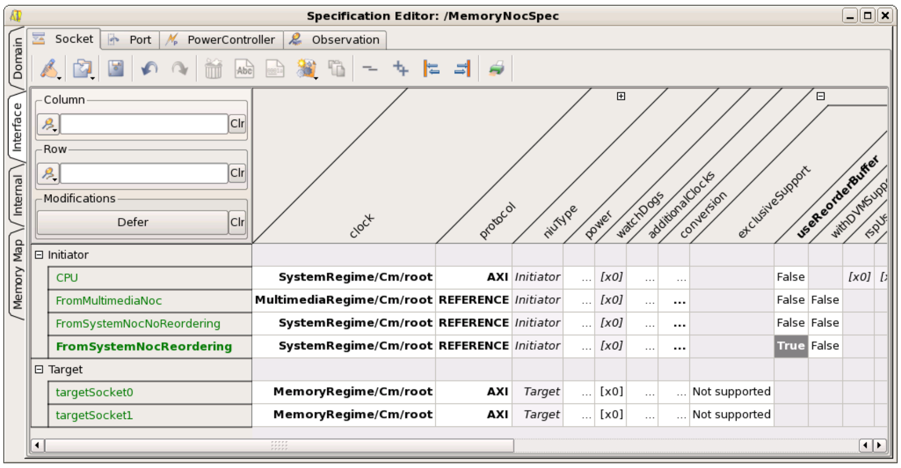
	clock	protocol	version	wData	wAddr	wId	wRspUser	wRegion	wLen	wQos	enWrite	enRead	useBigEndian	useFixed	signalNameFormat	reference	wReqUs	
⊟ Initiator																		
Init5	mainRegime/Cm/root	AXI	V3	64	32	4	0	0	4	0	True	True	*False*	True	li_Cc_Ss		0	*[x0*
Init6	mainRegime/Cm/root	AXI	V3	64	32	4	0	0	4	0	True	True	*False*	True	li_Cc_Ss		0	*[x0*
Init7	mainRegime/Cm/root	AXI	V3	64	32	4	0	0	4	0	True	True	*False*	True	li_Cc_Ss		0	*[x0*
⊟ Target																		
toMemoryNoC	mainRegime/Cm/root	REFERENCE														NSP		

In the memory NoC create four initiator and two target sockets. Give one initiator socket a 64 bit AXI interface for a CPU in the system clock regime. The other three initiator sockets should reference the NSP protocol, one for the multimedia NoC connection in the multimedia clock regime and the other two for two system NoC connections in the system clock regime. Give the two target sockets 64 bit AXI interfaces for DRAM controllers in the memory regime.


Specification Editor: /MemoryNocSpec

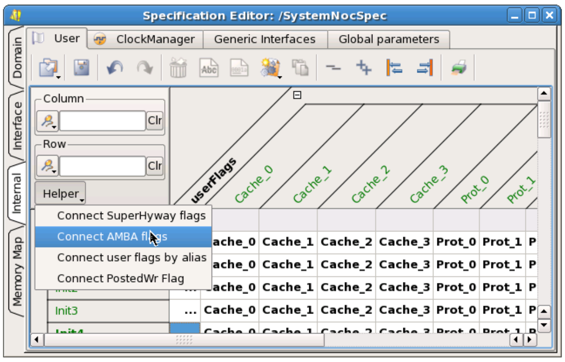
	clock	protocol	version	wData	wAddr	wId	wRspUser	wRegion	wLen	wQos	enWrite	enRead	useBigEndian	useFixed	signalNameFormat	reference	wReq	
⊟ Initiator																		
CPU	SystemRegime/Cm/root	AXI	V3	64	32	4	0	0	4	0	True	True	*False*	True	li_Cc_Ss		0	*[x*
FromMultimediaNoc	MultimediaRegime/Cm/root	REFERENCE														NSP		
FromSystemNocNoReordering	SystemRegime/Cm/root	REFERENCE														NSP		
FromSystemNocReordering	SystemRegime/Cm/root	REFERENCE														NSP		
⊟ **Target**																		
targetSocket0	MemoryRegime/Cm/root	AXI	V3	64	**32**	4	0	0	4	0	True	True	***False***	True	li_Cc_Ss		0	*[x*
targetSocket1	MemoryRegime/Cm/root	AXI	V3	64	**32**	4	0	0	4	0	True	True	***False***	True	li_Cc_Ss		0	*[x*

Enable a reorder buffer on the Reordering socket of the memory NoC.



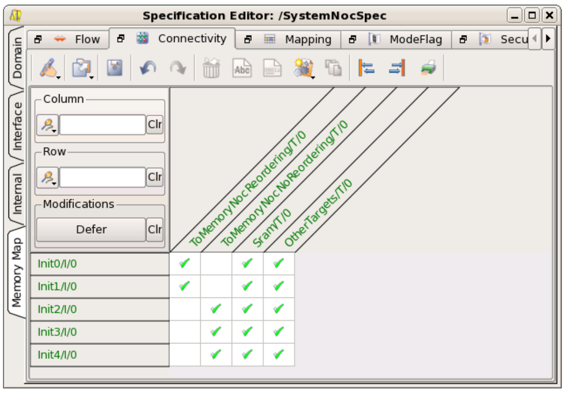
	clock	protocol	niuType	power	watchDogs	additionalClocks	conversion	exclusiveSupport	useReorderBuffer	withDVMSupp	rspUs	
⊟ Initiator												
CPU	SystemRegime/Cm/root	AXI	Initiator	…	[x0]	…	…		False		[x0]	[x
FromMultimediaNoc	MultimediaRegime/Cm/root	REFERENCE	Initiator	…	[x0]	…	…		False	False		
FromSystemNocNoReordering	SystemRegime/Cm/root	REFERENCE	Initiator	…	[x0]	…	…		False	False		
FromSystemNocReordering	SystemRegime/Cm/root	REFERENCE	Initiator	…	[x0]	…	…		**True**	False		
⊟ Target												
targetSocket0	MemoryRegime/Cm/root	AXI	Target	…	[x0]	…	…	Not supported				
targetSocket1	MemoryRegime/Cm/root	AXI	Target	…	[x0]	…	…	Not supported				

Step 5. Connect AMBA flags in the specifications that have both AXI initiators and AXI targets.



	userFlags	Cache_0	Cache_1	Cache_2	Cache_3	Prot_0	Prot_1	
		ache_0	Cache_1	Cache_2	Cache_3	Prot_0	Prot_1	P
		ache_0	Cache_1	Cache_2	Cache_3	Prot_0	Prot_1	P
	…	ache_0	Cache_1	Cache_2	Cache_3	Prot_0	Prot_1	P
Init3	…	Cache_0	Cache_1	Cache_2	Cache_3	Prot_0	Prot_1	P
Init4		Cache_0	Cache_1	Cache_2	Cache_3	Prot_0	Prot_1	P

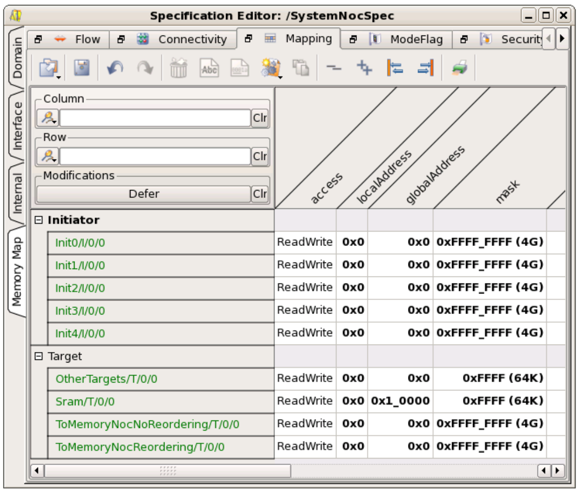
Step 6. In the system NoC connect only initiators 0 and 1 to the reordering NSP target. Connect only the other initiators to the no reordering NSP target.



	ToMemoryNocReordering/T/0	ToMemoryNocNoReordering/T/0	SramT/0	OtherTargets/T/0
Init0/I/0	✓		✓	✓
Init1/I/0	✓		✓	✓
Init2/I/0		✓	✓	✓
Init3/I/0		✓	✓	✓
Init4/I/0		✓	✓	✓

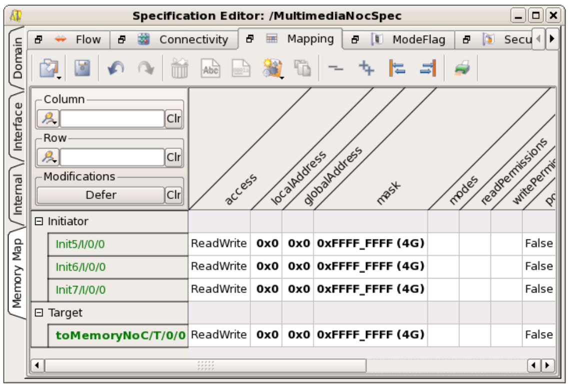
Step 7. Give all initiators a view of a full 4GB address space.

In the system NoC give the SRAM and other targets each a 64KB address space. Give the NSP memory target interfaces all of the remaining 4GB address space.



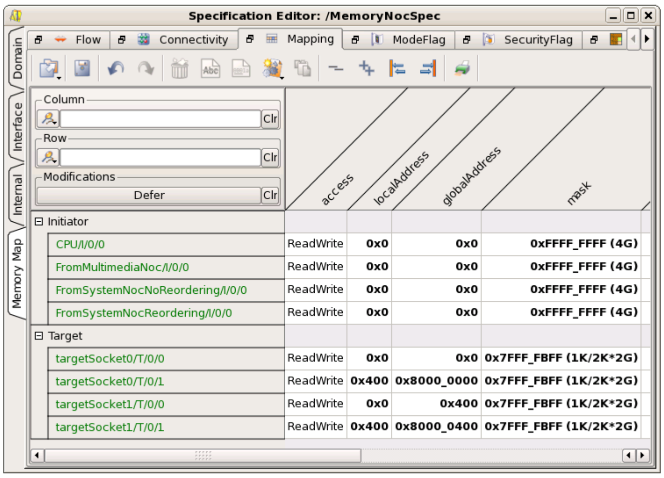
	access	localAddress	globalAddress	mask
Initiator				
Init0/I/0/0	ReadWrite	0x0	0x0	0xFFFF_FFFF (4G)
Init1/I/0/0	ReadWrite	0x0	0x0	0xFFFF_FFFF (4G)
Init2/I/0/0	ReadWrite	0x0	0x0	0xFFFF_FFFF (4G)
Init3/I/0/0	ReadWrite	0x0	0x0	0xFFFF_FFFF (4G)
Init4/I/0/0	ReadWrite	0x0	0x0	0xFFFF_FFFF (4G)
Target				
OtherTargets/T/0/0	ReadWrite	0x0	0x0	0xFFFF (64K)
Sram/T/0/0	ReadWrite	0x0	0x1_0000	0xFFFF (64K)
ToMemoryNocNoReordering/T/0/0	ReadWrite	0x0	0x0	0xFFFF_FFFF (4G)
ToMemoryNocReordering/T/0/0	ReadWrite	0x0	0x0	0xFFFF_FFFF (4G)

In the multimedia NoC give the one target the full 4G address range.



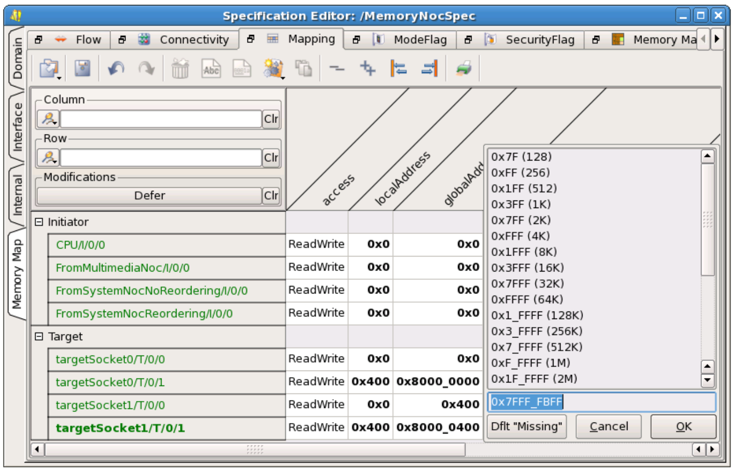
	access	localAddress	globalAddress	mask	modes	readPermissions	writePermis	po
Initiator								
Init5/I/0/0	ReadWrite	0x0	0x0	0xFFFF_FFFF (4G)				False
Init6/I/0/0	ReadWrite	0x0	0x0	0xFFFF_FFFF (4G)				False
Init7/I/0/0	ReadWrite	0x0	0x0	0xFFFF_FFFF (4G)				False
Target								
toMemoryNoC/T/0/0	ReadWrite	0x0	0x0	0xFFFF_FFFF (4G)				False

In the memory NoC set up interleaved memory address ranges for two 2GB memories interleaved on 1KB stripes.



	access	localAddress	globalAddress	mask
Initiator				
CPU/I/0/0	ReadWrite	0x0	0x0	0xFFFF_FFFF (4G)
FromMultimediaNoc/I/0/0	ReadWrite	0x0	0x0	0xFFFF_FFFF (4G)
FromSystemNocNoReordering/I/0/0	ReadWrite	0x0	0x0	0xFFFF_FFFF (4G)
FromSystemNocReordering/I/0/0	ReadWrite	0x0	0x0	0xFFFF_FFFF (4G)
Target				
targetSocket0/T/0/0	ReadWrite	0x0	0x0	0x7FFF_FBFF (1K/2K*2G)
targetSocket0/T/0/1	ReadWrite	0x400	0x8000_0000	0x7FFF_FBFF (1K/2K*2G)
targetSocket1/T/0/0	ReadWrite	0x0	0x400	0x7FFF_FBFF (1K/2K*2G)
targetSocket1/T/0/1	ReadWrite	0x400	0x8000_0400	0x7FFF_FBFF (1K/2K*2G)

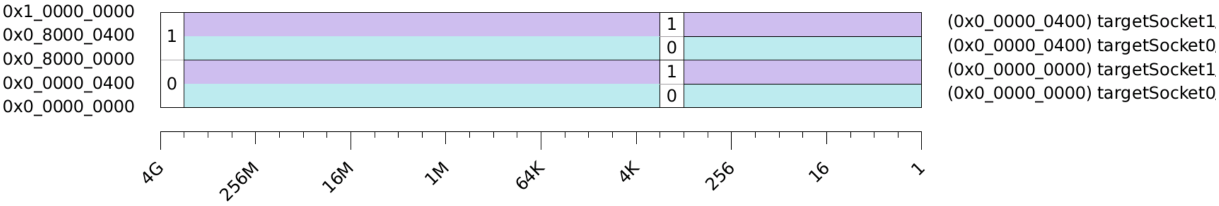
Do this by editing the address range masks for each mapping to leave address bit 10 masked to a zero.



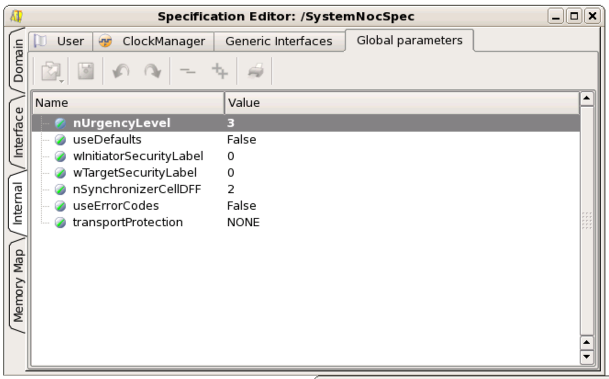
	access	localAddress	globalAdd
⊟ Initiator			
CPU/I/0/0	ReadWrite	0x0	0x0
FromMultimediaNoc/I/0/0	ReadWrite	0x0	0x0
FromSystemNocNoReordering/I/0/0	ReadWrite	0x0	0x0
FromSystemNocReordering/I/0/0	ReadWrite	0x0	0x0
⊟ Target			
targetSocket0/T/0/0	ReadWrite	0x0	0x0
targetSocket0/T/0/1	ReadWrite	0x400	0x8000_0000
targetSocket1/T/0/0	ReadWrite	0x0	0x400
targetSocket1/T/0/1	ReadWrite	0x400	0x8000_0400

0x7FFF_FBFF

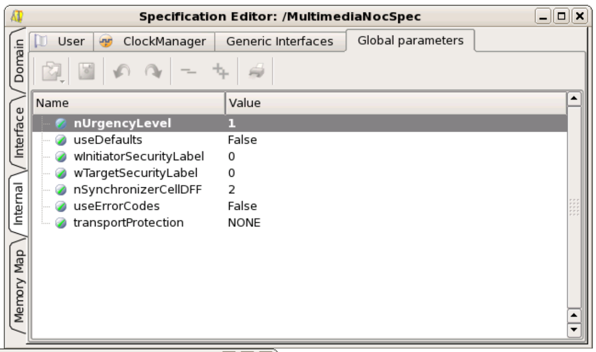
Note that there must be two mappings for each target, one in the upper and one in the lower half of the address space. Check this in the memory map viewer.



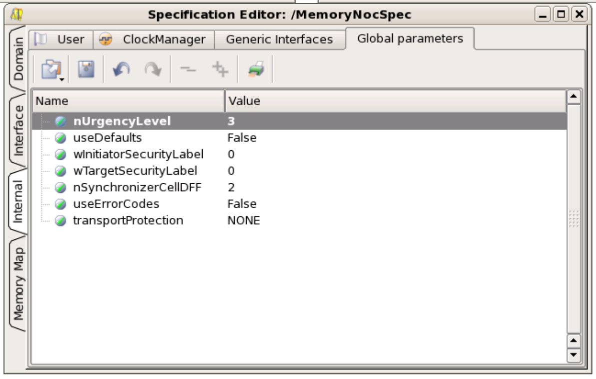
Step 8. Set four levels of urgency (nUrgencyLevel = 3) in the system and memory NoCs. Set two levels of urgency in the multimedia NoC.



Name	Value
nUrgencyLevel	3
useDefaults	False
wInitiatorSecurityLabel	0
wTargetSecurityLabel	0
nSynchronizerCellDFF	2
useErrorCodes	False
transportProtection	NONE

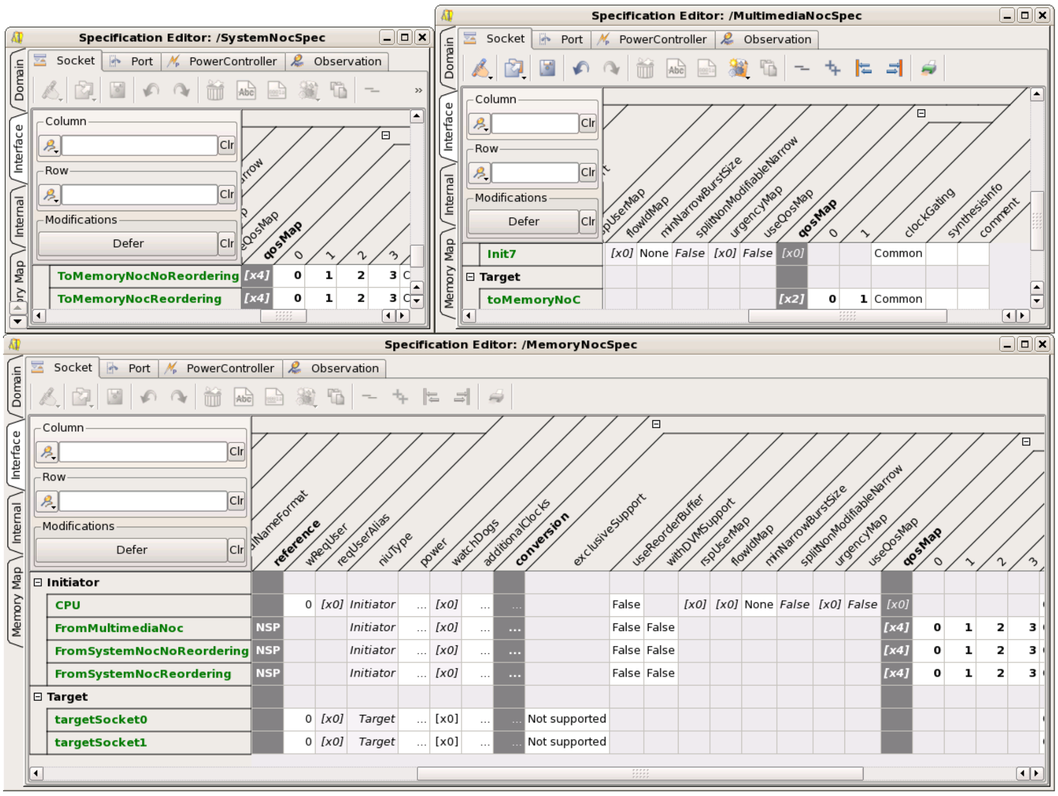


Name	Value
nUrgencyLevel	1
useDefaults	False
wInitiatorSecurityLabel	0
wTargetSecurityLabel	0
nSynchronizerCellDFF	2
useErrorCodes	False
transportProtection	NONE



Name	Value
nUrgencyLevel	3
useDefaults	False
wInitiatorSecurityLabel	0
wTargetSecurityLabel	0
nSynchronizerCellDFF	2
useErrorCodes	False
transportProtection	NONE

Step 9. It is necessary to map the QoS signals of the NSP interface to the QoS signals within the NoC. This enables NoCs with different numbers of QoS levels to be connected. Do this in the Socket interface conversion parameters.

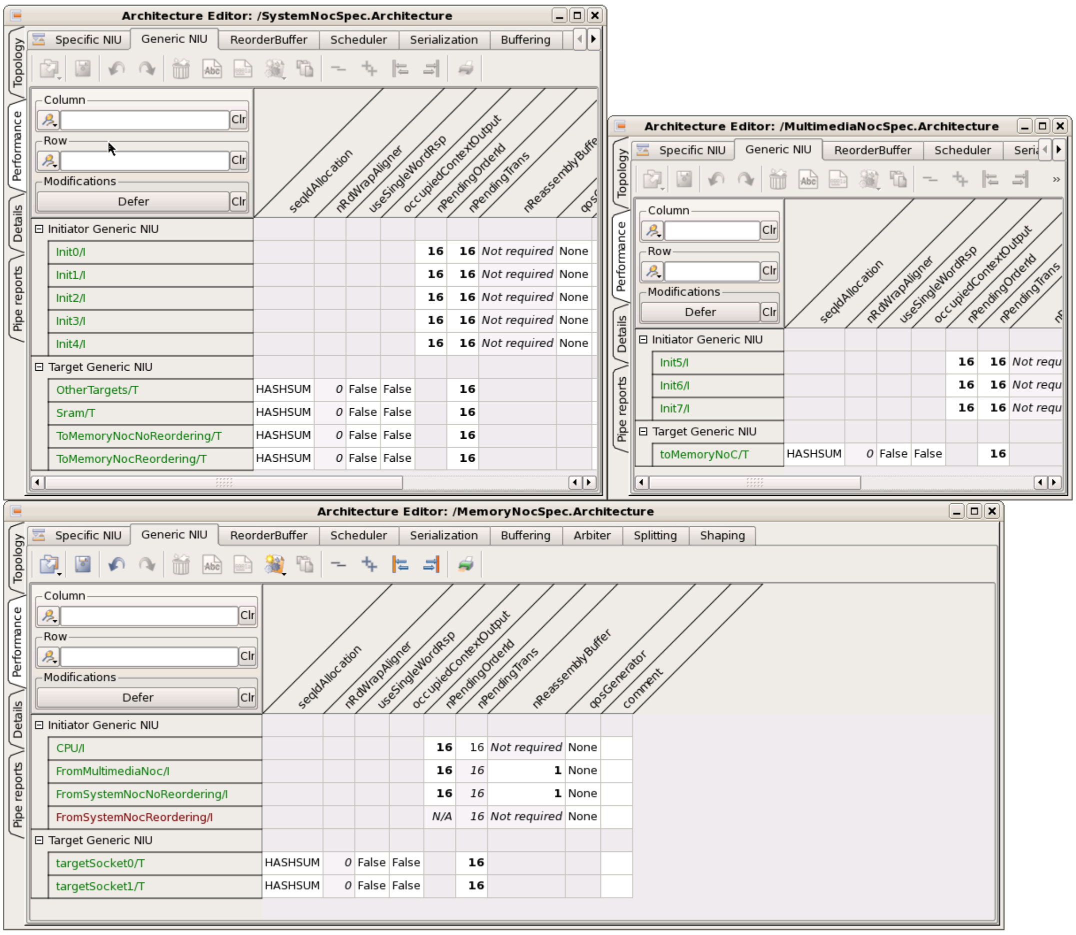


	qosMap	0	1	2	3	
ToMemoryNocNoReordering	[x4]	0	1	2	3	C
ToMemoryNocReordering	[x4]	0	1	2	3	C

	flowIdMap	minNarrowBurstSize	splitNonModifiableNarrow	urgencyMap	useQosMap	qosMap	0	1	clockGating	synthesisInfo	comment
Init7	[x0]	None	False	[x0]	False	[x0]			Common		
⊟ Target											
toMemoryNoC						[x2]	0	1	Common		

	reference	wReqUser	reqUserAlias	niuType	power	watchDogs	additionalClocks	conversion	exclusiveSupport	useReorderBuffer	withDVMSupport	rspUserMap	flowIdMap	minNarrowBurstSize	splitNonModifiableNarrow	urgencyMap	useQosMap	qosMap	0	1	2	3
⊟ Initiator																						
CPU		0	[x0]	Initiator	...	[x0]	...	...		False		[x0]	[x0]	None	False	[x0]	False	[x0]				
FromMultimediaNoc	NSP			Initiator	...	[x0]	...	...		False	False							[x4]	0	1	2	3
FromSystemNocNoReordering	NSP			Initiator	...	[x0]	...	...		False	False							[x4]	0	1	2	3
FromSystemNocReordering	NSP			Initiator	...	[x0]	...	...		False	False							[x4]	0	1	2	3
⊟ Target																						
targetSocket0		0	[x0]	Target	...	[x0]	...	...	Not supported													
targetSocket1		0	[x0]	Target	...	[x0]	...	...	Not supported													

Step 10. Create architectures for each specification and set 16 for all numbers of pending transactions and order IDs.

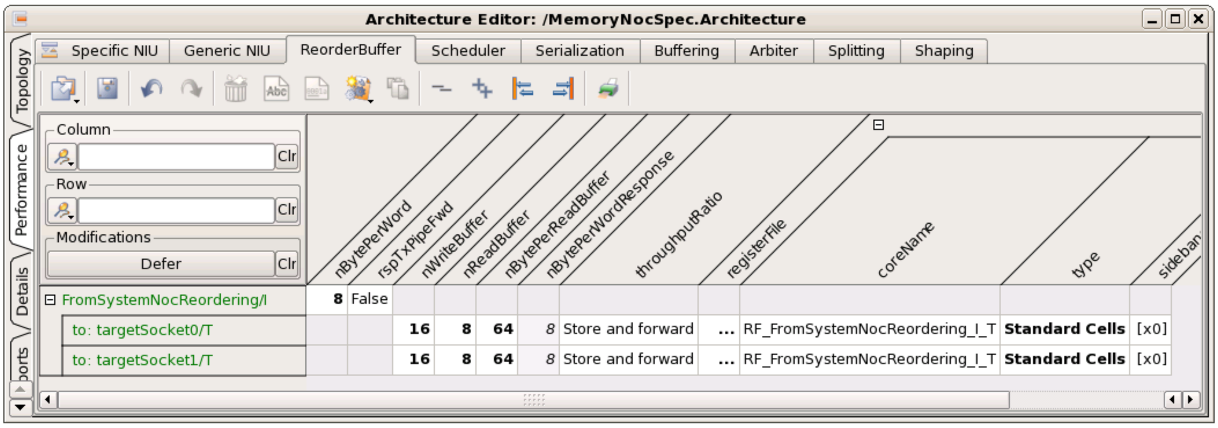


	seqIdAllocation	nRdWrapAligner	useSingleWordRsp	occupiedContextOutput	nPendingOrderId	nPendingTrans	nReassemblyBuffe	qos
⊟ Initiator Generic NIU								
Init0/I					16	16	Not required	None
Init1/I					16	16	Not required	None
Init2/I					16	16	Not required	None
Init3/I					16	16	Not required	None
Init4/I					16	16	Not required	None
⊟ Target Generic NIU								
OtherTargets/T	HASHSUM	0	False	False		16		
Sram/T	HASHSUM	0	False	False		16		
ToMemoryNocNoReordering/T	HASHSUM	0	False	False		16		
ToMemoryNocReordering/T	HASHSUM	0	False	False		16		

	seqIdAllocation	nRdWrapAligner	useSingleWordRsp	occupiedContextOutput	nPendingOrderId	nPendingTrans	nR
⊟ Initiator Generic NIU							
Init5/I					16	16	Not requ
Init6/I					16	16	Not requ
Init7/I					16	16	Not requ
⊟ Target Generic NIU							
toMemoryNoC/T	HASHSUM	0	False	False		16	

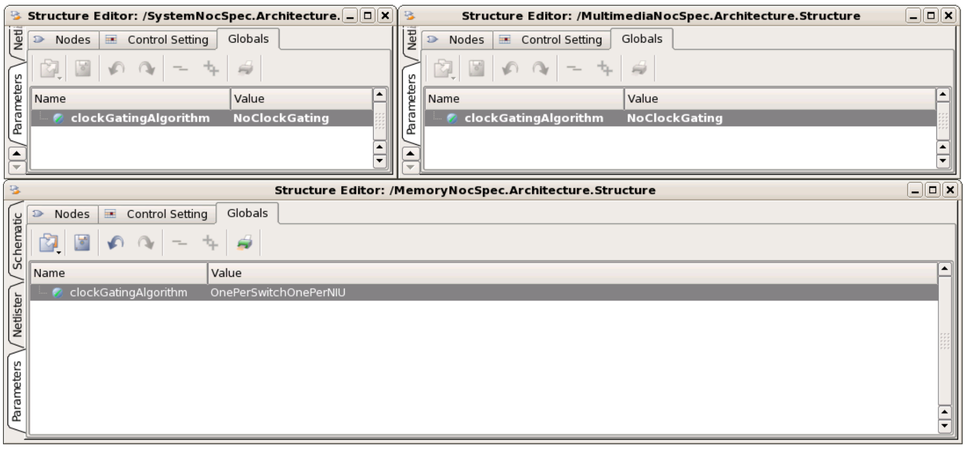
	seqIdAllocation	nRdWrapAligner	useSingleWordRsp	occupiedContextOutput	nPendingOrderId	nPendingTrans	nReassemblyBuffer	qosGenerator	comment
⊟ Initiator Generic NIU									
CPU/I					16	16	Not required	None	
FromMultimediaNoc/I					16	16	1	None	
FromSystemNocNoReordering/I					16	16	1	None	
FromSystemNocReordering/I					N/A	16	Not required	None	
⊟ Target Generic NIU									
targetSocket0/T	HASHSUM	0	False	False		16			
targetSocket1/T	HASHSUM	0	False	False		16			

Step 11. Configure the reorder buffer in the memory NoC with 8 bytes per word, 16 write buffers, 8 read buffers of 64 bytes each per channel, and use standard cells to implement the buffers.



	nBytePerWord	rspTxPipeFwd	nWriteBuffer	nReadBuffer	nBytePerReadBuffer	nBytePerWordResponse	throughputRatio	registerFile	coreName	type	sideban
⊟ FromSystemNocReordering/I	8	False									
to: targetSocket0/T			16	8	64	8	Store and forward	...	RF_FromSystemNocReordering_I_T	Standard Cells	[x0]
to: targetSocket1/T			16	8	64	8	Store and forward	...	RF_FromSystemNocReordering_I_T	Standard Cells	[x0]

Step 12. Create structures for each NoC. For simplicity, disable clock gating in all structures.



Structure Editor: /SystemNocSpec.Architecture.

Name	Value
clockGatingAlgorithm	NoClockGating

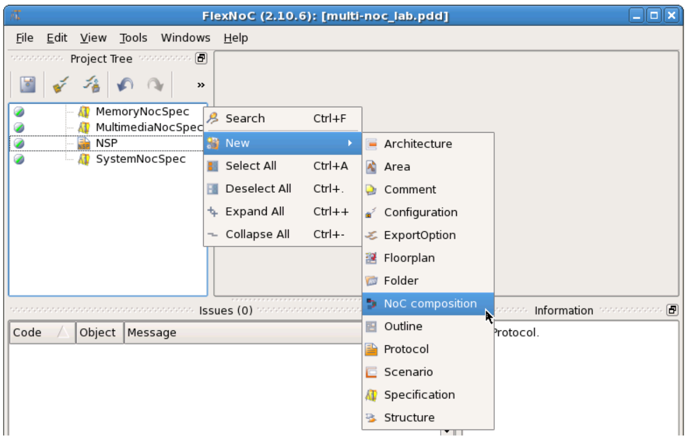
Structure Editor: /MultimediaNocSpec.Architecture.Structure

Name	Value
clockGatingAlgorithm	NoClockGating

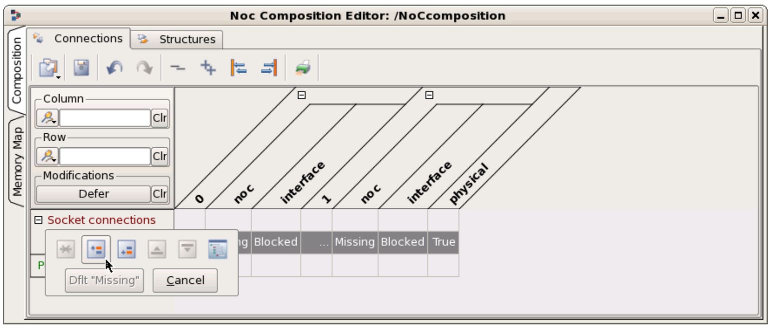
Structure Editor: /MemoryNocSpec.Architecture.Structure

Name	Value
clockGatingAlgorithm	OnePerSwitchOnePerNIU

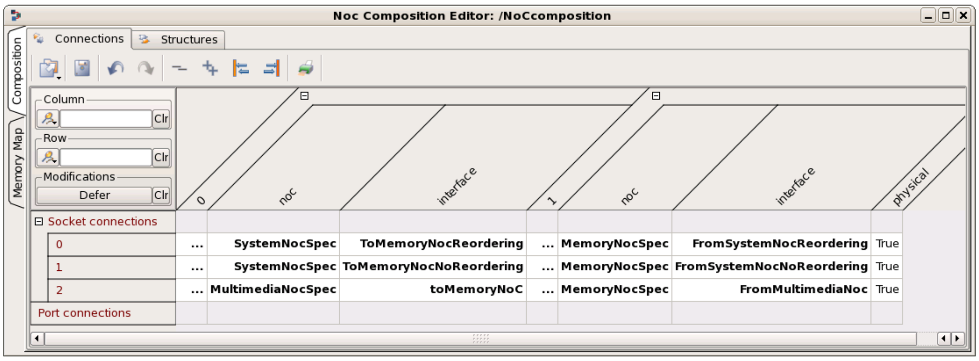
Step 13. Create a NoC composition.



Step 14. In the NoC composition editor create three connections.

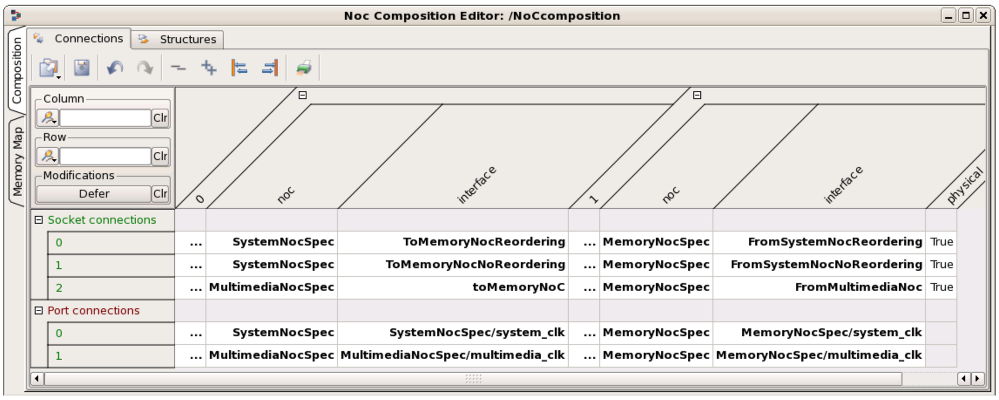


Step 15. Connect the two NSP target sockets of the system NoC to the appropriate memory NoC socket interfaces. Connect the NSP target socket of the multimedia NoC to the appropriate memory NoC initiator socket interface.



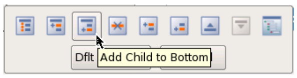
	0	noc	interface	1	noc	interface	physical
Socket connections							
0	...	SystemNocSpec	ToMemoryNocReordering	...	MemoryNocSpec	FromSystemNocReordering	True
1	...	SystemNocSpec	ToMemoryNocNoReordering	...	MemoryNocSpec	FromSystemNocNoReordering	True
2	...	MultimediaNocSpec	toMemoryNoC	...	MemoryNocSpec	FromMultimediaNoc	True
Port connections							

Step 16. Add two port connections to connect the corresponding clock signals between the system and memory NoC and between the multimedia and memory NoC.

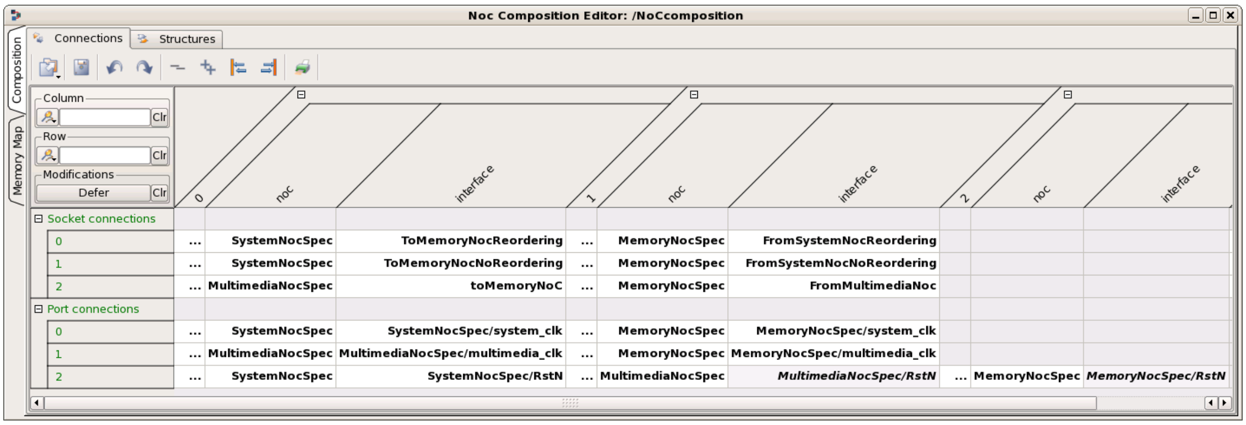


	0	noc	interface	1	noc	interface	physical
Socket connections							
0	...	SystemNocSpec	ToMemoryNocReordering	...	MemoryNocSpec	FromSystemNocReordering	True
1	...	SystemNocSpec	ToMemoryNocNoReordering	...	MemoryNocSpec	FromSystemNocNoReordering	True
2	...	MultimediaNocSpec	toMemoryNoC	...	MemoryNocSpec	FromMultimediaNoc	True
Port connections							
0	...	SystemNocSpec	SystemNocSpec/system_clk	...	MemoryNocSpec	MemoryNocSpec/system_clk	
1	...	MultimediaNocSpec	MultimediaNocSpec/multimedia_clk	...	MemoryNocSpec	MemoryNocSpec/multimedia_clk	

Step 17. Use the Add Child button

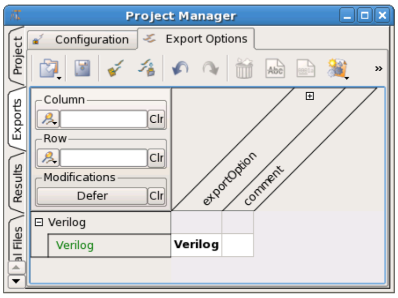


to create three connections of the reset signal between all three NoCs.

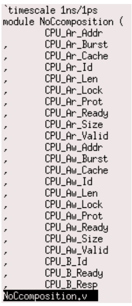


	0	noc	interface	1	noc	interface	2	noc	interface
Socket connections									
0	...	SystemNocSpec	ToMemoryNocReordering	...	MemoryNocSpec	FromSystemNocReordering			
1	...	SystemNocSpec	ToMemoryNocNoReordering	...	MemoryNocSpec	FromSystemNocNoReordering			
2	...	MultimediaNocSpec	toMemoryNoC	...	MemoryNocSpec	FromMultimediaNoc			
Port connections									
0	...	SystemNocSpec	SystemNocSpec/system_clk	...	MemoryNocSpec	MemoryNocSpec/system_clk			
1	...	MultimediaNocSpec	MultimediaNocSpec/multimedia_clk	...	MemoryNocSpec	MemoryNocSpec/multimedia_clk			
2	...	SystemNocSpec	SystemNocSpec/RstN	...	MultimediaNocSpec	*MultimediaNocSpec/RstN*	...	MemoryNocSpec	*MemoryNocSpec/RstN*

Step 18. Create a Verilog export option

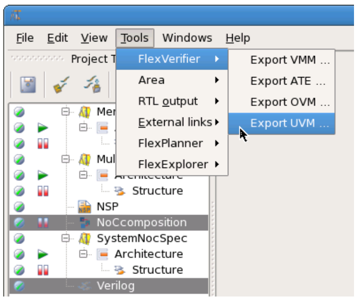


and export Verilog for the NoC composition. Review the NoCcomposition.v top level Verilog file.

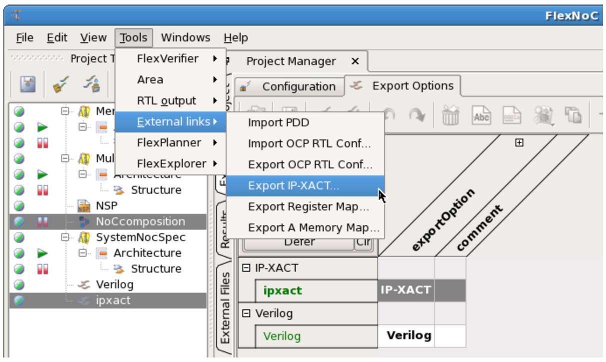


```
`timescale 1ns/1ps
module NoCcomposition (
        CPU_Ar_Addr
,       CPU_Ar_Burst
,       CPU_Ar_Cache
,       CPU_Ar_Id
,       CPU_Ar_Len
,       CPU_Ar_Lock
,       CPU_Ar_Prot
,       CPU_Ar_Ready
,       CPU_Ar_Size
,       CPU_Ar_Valid
,       CPU_Aw_Addr
,       CPU_Aw_Burst
,       CPU_Aw_Cache
,       CPU_Aw_Id
,       CPU_Aw_Len
,       CPU_Aw_Lock
,       CPU_Aw_Prot
,       CPU_Aw_Ready
,       CPU_Aw_Size
,       CPU_Aw_Valid
,       CPU_B_Id
,       CPU_B_Ready
,       CPU_B_Resp
NoCcomposition.v
```

Step 19. Export a UVM testbench.



Step 20. Create an IP-XACT export option and export an IP-XACT description of the NoC composition.



Step 21. It is also possible to create scenarios for more than one NoC specification and run a FlexExplorer simulation that models traffic between the NoCs.

Step 22. It is also possible to create a floorplan outline and run a FlexPlanner floorplan estimation that encompasses all NoCs within a NoC composition.