The Rubber Jigsaw Puzzle
Floorplanning for Network-on-chip

Byungchul Hong (홍병철), Brian Huang (黃繼樟), Jonah Probell

Arteris, Inc.

Campbell, CA, USA

www.arteris.com

ABSTRACT

Network-on-chip fabrics have become prevalent in SoCs. They can be the savior or the bottleneck of system performance. NoCs comprise the long wires between IP macros, and can account for a significant portion of top level routing congestion. The chip floorplan has a big impact on how well the NoC meets the system performance requirements.

In this paper we explain and provide guidelines on ways to optimize a floorplan for NoC. In particular, DRAM interfaces require special considerations. A dedicated, but distributed, network of memory subsystem components minimizes wire routing congestion. With special attention to the proximity of latency-critical initiators it can give best performance.

With respect to macro placement, ones in shared domains should be placed with their interfaces close together. Interfaces with similar traffic classes should also be clustered. Pipeline stages should be placed closely to interfaces. The configuration of the interfaces can enable a clean separation of clock trees and power nets between macros and the top level.

Table of Contents

1. Introduction

This paper is applicable to floorplan creation in Synopsys DC-Graphical and IC Compiler tools, which help to achieve the goals of lower-power and smaller chips with faster time-to-market.

Most designers agree that a network-on-chip (NoC) should be designed to accommodate the chip floorplan. However, there are benefits to also designing the floorplan to accommodate the NoC. Hence we write this paper.

The hard macros of a chip floorplan must fit together like pieces of a jigsaw puzzle. However, there is space between each piece. That space is filled with the NoC. The NoC is squishy, like rubber. Its logic and wires move during place & route to surround the macros that it connects. Therefore, floorplanning for a NoC is like piecing together a rubber jigsaw puzzle.

A network-on-chip (NoC) interconnect has a large effect on the performance of a system-on-chip (SoC) due to the effects of a NoC on bandwidth, latency, and quality of service (QoS). A NoC also has a large effect on the physical design of a chip because it comprises the important long wires in the chip. An optimal topology has switches between physically close IPs, with pipelined links to other localized switches. It also has adapters on links between clock or power domains.

Commonly, a portion of the logic and interconnecting wires of the NoC are implemented at the top level in a sea of gates between macro islands. The seas have channels and fjords. Below we provide some guidelines to help the floorplan map out the routes that the NoC can navigate to move data around the chip.

Each guideline is known to yield significant benefits to certain chip designs. Not every guideline is appropriate to every project, but the wise floorplanning engineer will consider them all.

2. Types of interconnects

Crossbar interconnect

Crossbars were adopted by high bandwidth applications in the early 2000s. A crossbar is essentially an arrangement of demuxes and muxes between initiators and targets. They provide more concurrency than shared busses, and enable greater diversity of traffic QoS. The improved concurrency comes from having many parallel paths between initiators and targets. However, that comes with the cost of a large number of wires.

When the number of initiators and targets approaches 12x12, the wire tangle starts to become unroutable and causes logic timing problems. To support more IPs in chips, crossbars can be cascaded, but at the cost of significant latency and top-level logic area.

In contrast, a NoC packetizes and serializes transactions between all initiators and targets. Transactions pass across any physical distance, and across clock and power domains. The configuration of switches (demux-routers and arbiter-muxes) and their physical locations, known as the network topology, is configured to trade-off wires and traffic bandwidth.

Algorithmic NoC

This approach is commonly studied in academia. All switches are of approximately identical configurations, typically 5x5 (with 4x4 or 3x3 at edges and corners) to support north, south, east,

west, and local traffic. Typically, a physical link can carry transport layer requests and responses, and deadlock avoidance is an area of concern and study. Regular topologies such as meshes, tori, and rings are frequently studied.

Fine-grained NoC

This is the approach used in most NoC-based commercial SoCs. The NoC topology is composed of individual arbiter-muxes, demux-routers of varying size as well as individually configurable buffering, probing, and other behavioral units. There is clock and power domain adaptation as needed throughout the topology. There are variable amounts of pipelining along wires of widely varying lengths between switches. Such chips typically share an external DDR DRAM, and most topologies have distributed bandwidth that is aggregated through serialization adapters and rooted in DRAM controllers. Separate networks are used for requests and for response, making deadlocks physically impossible.

This paper addresses fine-grained NoCs.

3. DRAM-related considerations and guidelines

The DRAM subsystem has a major effect on the performance of SoCs. Consider the prototypical memory system block diagram shown in Figure 1. It supports:

- two CPU clusters with a shared address space that is interleaved between multiple memories.
- a variety of other low latency (LL) and high bandwidth (HB) initiators connected through a network of shared switches.
- shared reorder buffers (to match different memories' transaction responses with their request order), one for LL and one for HB traffic
- two interleaved last-level cache (LLC) slices
- two DRAM schedulers, each with separate LL and HB ports
- DRAM protocol controller, PHY, and pins.

Not all of these elements are present in every SoC, and some are present in different numbers, but this diagram represents a system with a bit of everything. The guidelines below are applicable to any memory subsystem configuration.

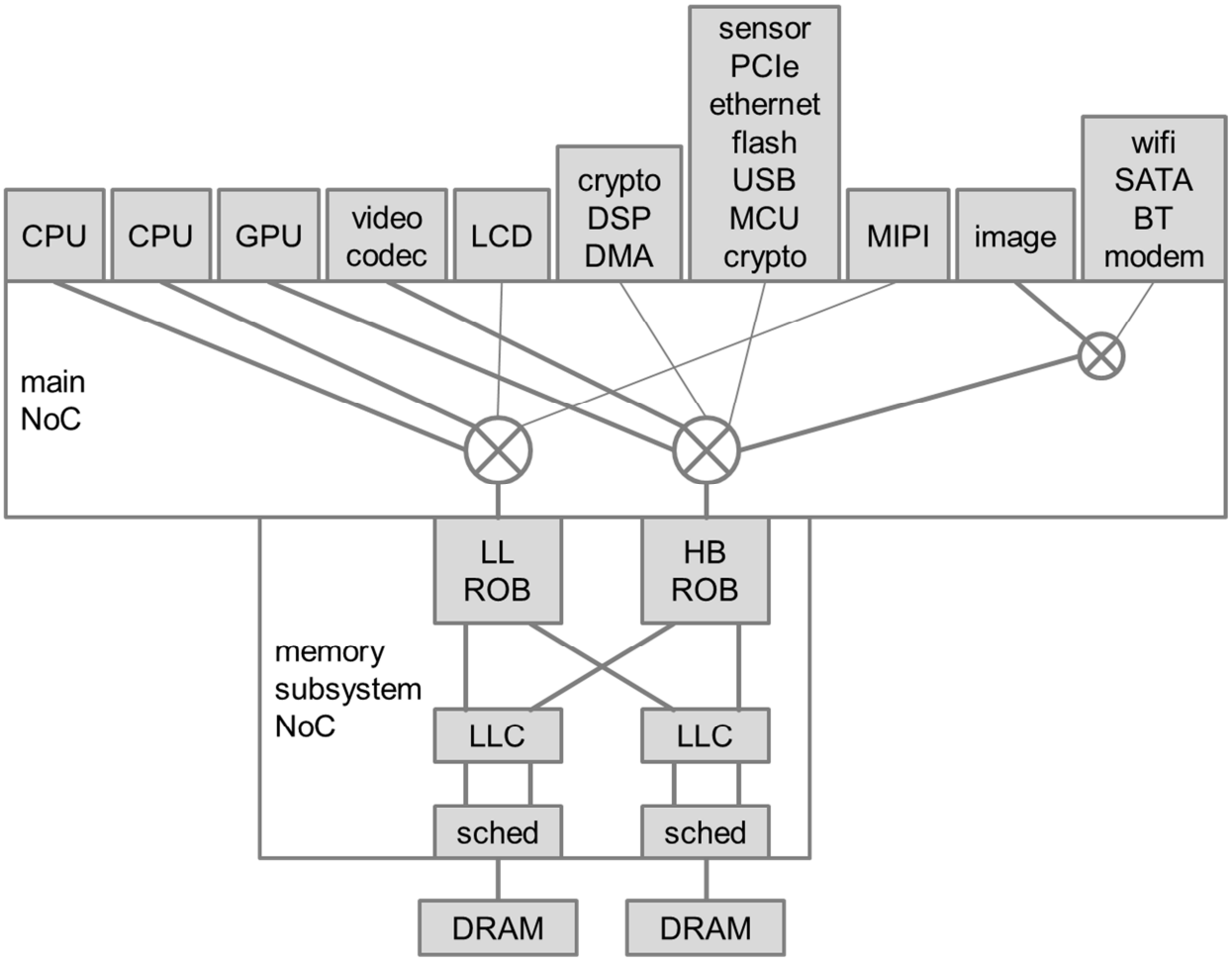

Figure 1: A memory subsystem interconnect block diagram.

That is a nice block diagram for a system architect, but it tells the floorplan designer almost nothing about how to lay out the chip. Figure 2 shows an example floorplan, corresponding to the block diagram of Figure 1. The floorplan illustrates the guidelines presented below. Any resemblance to actual chips is purely coincidental.

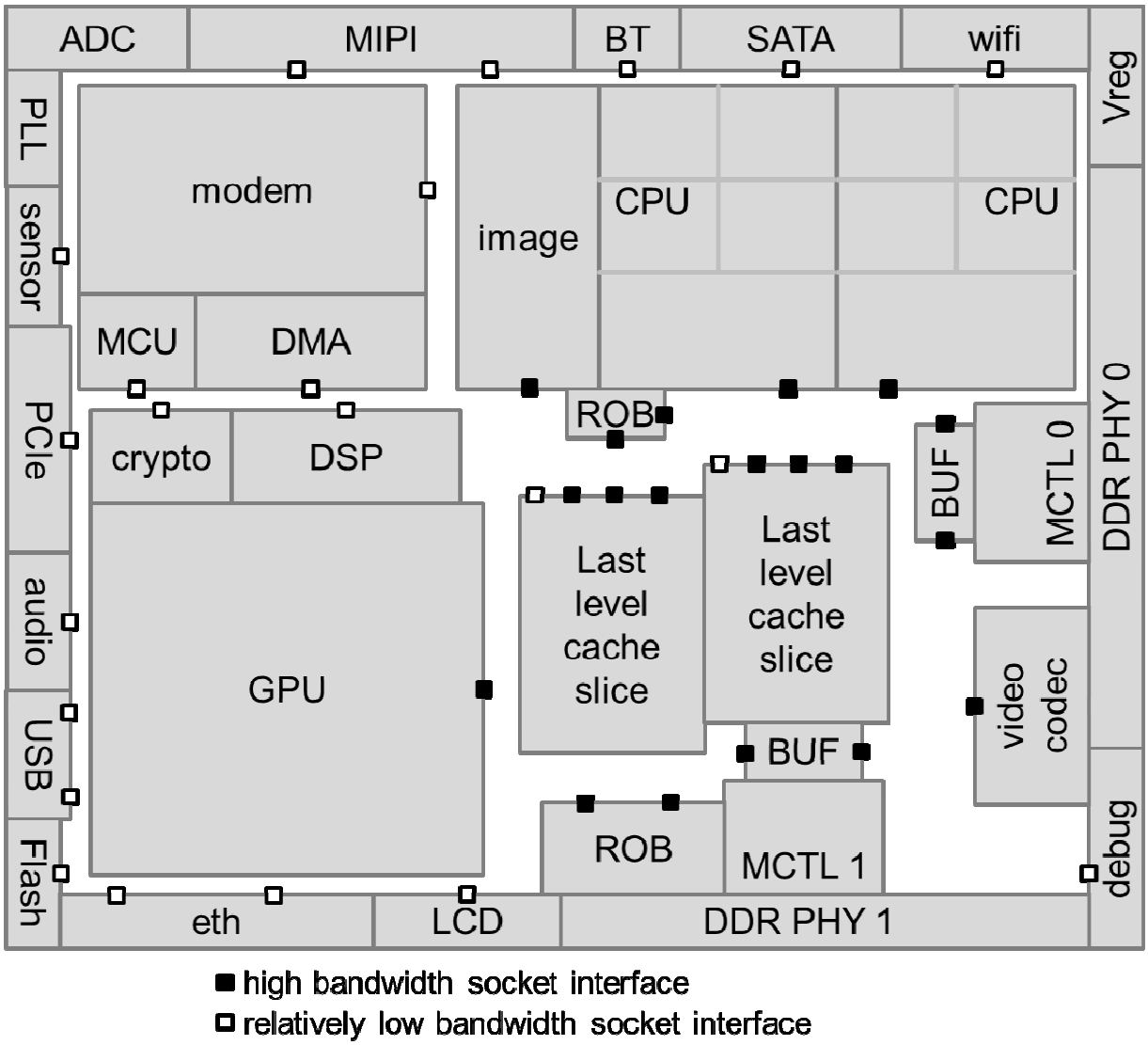

Figure 2: An example floorplan.

Guideline 1: Physically distribute the memory subsystem.

In many SoCs the memory subsystem is a large block, designed by a specialized team, and implemented as a monolithic macro. The responsibilities of the memory subsystem are:

- implement the appropriate DRAM protocol and PHY interface
- schedule commands to the DRAM chip to make best use of available banks and open pages to maximize throughput and minimize latency for critical traffic
- buffer write data until it can be sent to the DRAM
- accept and buffer read data from the DRAM until it can be accepted by the NoC
- do address interleaving for multiple DRAMs
- do reorder buffering for out-of-order interleaved responses
- implement last-level caching

It is not necessary to perform all of these functions in one physical location, and doing so creates a routing nightmare. The PHYs, protocol controller, and scheduler logic have tight timing, so they should be placed close to DRAM interface pins on the side of the chip. However, buffers for each DRAM scheduler port should be split into multiple separate smaller buffers. That is true for write and read data buffers. This allows the placer to stretch them on sides of the scheduler towards different physical IPs in order to reduce routing congestion. Three or four buffer sets is typically good for accommodating different traffic types. See Guideline 2 below.

Having a dedicated memory subsystem NoC allows a clean interface to other functions within the chip. In particular, in a system with multiple interleaved DRAMs, the NoC initiator interface units perform the DRAM address decoding and therefore the function of sending requests to the appropriate DRAM interface. Performing this at every initiator interface in the chip would require every initiator interface to be redesigned to support every change in interleaving scheme.

With interleaving comes a need for reorder buffers. Packing them close to the DRAM interfaces would create significant routing congestion hot spots unnecessarily. Splitting up the reorder buffering into multiple smaller buffers, placed close to high bandwidth initiators throughout the floorplan, reduces routing congestion. The reorder buffers are part of the memory subsystem NoC, placed at shared target sockets of the main NoC topology.

Last-level caches are often a significant portion of chip area, and serve a high bandwidth of requests coming from initiators all over the chip. Since they are big and have no direct I/O connections it makes sense for them to be central in the chip. In fact, there is no need to have a single last level cache array. Initiators can be interleaved to any number of physically distributed last level cache slices, which can serve their misses from any other number of DRAM interfaces.

These considerations are all specific to the memory subsystem of the chip, and are best addressed with their own dedicated NoC, even though it is physically interspersed with the top-level NoC. Furthermore, having a dedicated memory subsystem NoC also makes it easy to change out the memory subsystem for derivative chips, without having to resynthesize every initiator interface unit through the top level NoC.

Guideline 2: Place low latency and high bandwidth IPs close to memory subsystem.

Certain initiators, CPUs in particular, are especially latency critical. This includes MIPI LLI interfaces carrying CPU traffic from external chips. To avoid wire and pipeline delay through the chip, these initiators should be placed relatively close to the DRAM interface, and oriented with their bus interface ports pointed towards the DRAM. If multiple DRAM interfaces are close together, such as adjacent to each other at a corner of the chip, latency critical IPs should be placed close to the corner. Every chip is different, but the specific parts of the memory subsystem that must be between the CPUs and DRAM should be scrunched in as much as possible between the CPU and DRAM interface without creating a routing nightmare.

After floorplanning to minimize NoC latency, the floorplan should consider bandwidth requirements. A NoC tends to aggregate bandwidth from distant initiators onto buses that are increasingly wide as they approach shared targets. The DRAM is the highest bandwidth shared target in many SoCs. Higher bandwidth initiators should be placed and oriented with their bus interface ports as close as possible to the shared high bandwidth target. This minimizes total wire length by ensuring that the widest links are the shortest, and also minimizes power dissipation due to wire capacitance because the largest amount of toggling happens on the shortest wires.

4. Macro and domain considerations and guidelines

Planning for power nets and clock tree insertion and balancing is a major constraint on the floorplan. Different macros exist in different clock and power domains, which are potentially different from the top level interconnect logic.

Guideline 3: Group sockets that are in shared domains close together.

This might seem obvious, but it is especially important for closing timing on the long wire routes between IPs. NoC logic tends to be spread over long distances, and span multiple domains. Communication between IPs through the NoC is through transport-level clock domain adapters and power disconnect units within the sea of gates. A large group of NoC logic clustering in one area forces logic in other power domains to be routed around. This can cause unintentional bottlenecks and routing congestion points. It is desirable to keep the locations of socket interfaces that share domains physically close to each other to avoid unintentional bottlenecks. Not only should macros in shared domains be placed close to each other, but interfaces should be placed towards the side or corner of macros that are closest to others within the same clock domain. Figure 3 shows an egregious example of IPs in domain B separating the black NoC sockets of domain A. Furthermore, the NoC sockets are central to the IPs.

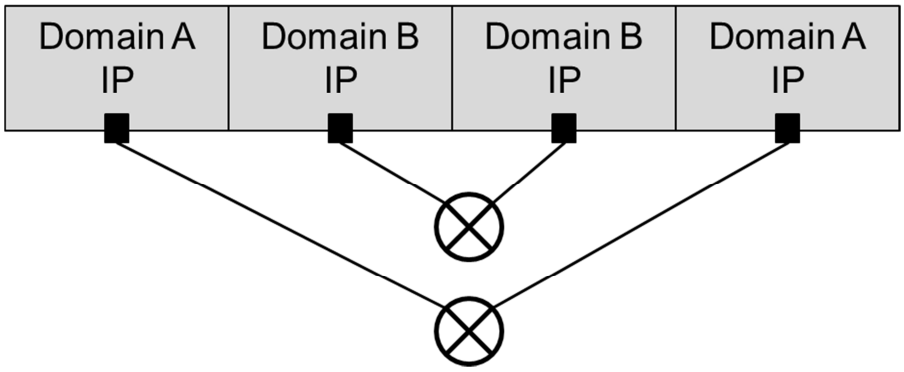

Figure 3: Bad arrangement of domain A IPs and domain B sockets.

Figure 4 shows a preferable organization because IPs of common domains are physically close to each other. Furthermore, the socket interfaces of the IPs in common domains are very close. This will make power and clock net design simpler and more compact. See Guideline 4 below.

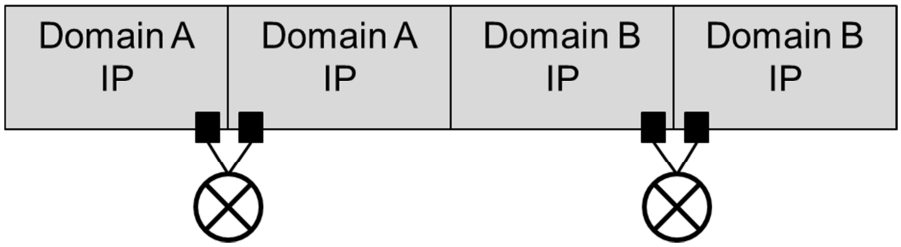

Figure 4: Good arrangement that groups IPs and sockets by domain.

Another important reason for grouping sockets that share a domain is that each switch in the network topology exists within a single domain. Having switches grouped both physically and by domain means that async clock domain adapters, which are area-expensive, can be placed along the long wire links between switches. This spreads the logic physically, and allows place & route to move the clock adapter logic along the link as needed to keep the clock tree with the tighter

constraints in a physically smaller area. Figure 5 shows a clock domain adapter placed between two switches in different clock domains so as to meet timing requirements in each domain.

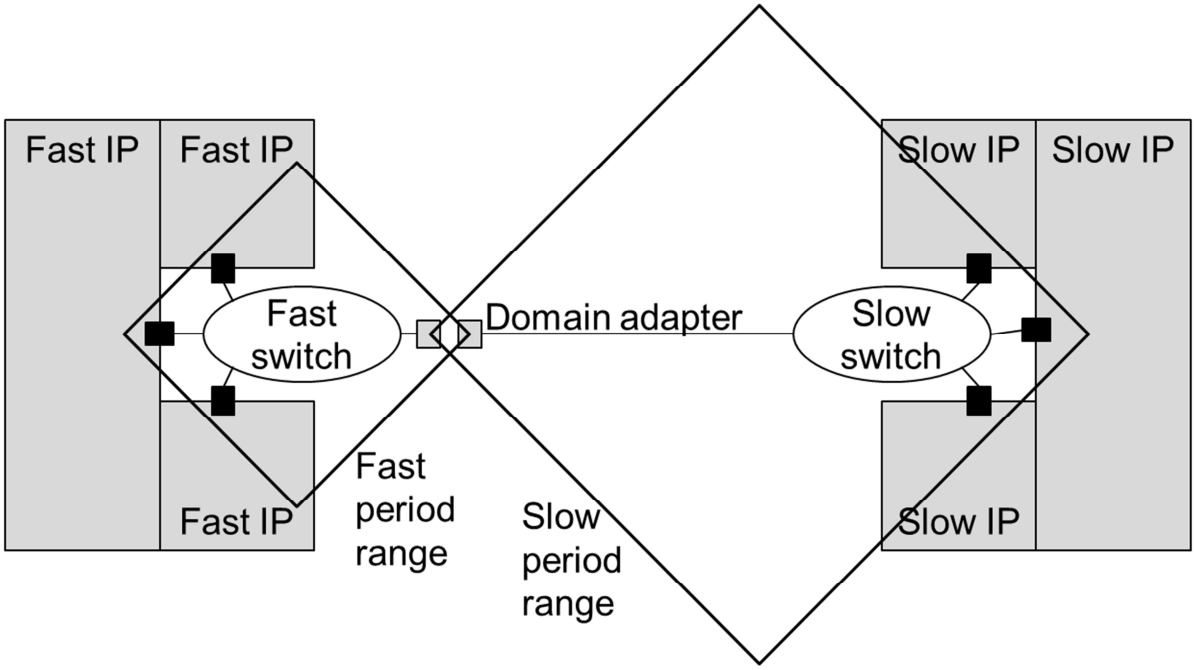

Figure 5: Domain adapter is located closer to faster domain logic, and so meets timing in both domains

Note that the range of signal propagation within a constant period of time along Manhattan routed wires is not a circle, but a diamond!

Guideline 4: Cross macro boundaries on packetized links.

This is actually a front-end system architecture decision, but it has big consequences on the planning of power networks and clock trees. Consider the following four approaches to interfacing macros to the top-level NoC.

Transaction protocol interface

A typical socket connection between a macro and top level NoC uses a transaction protocol such as AMBA. This is shown in Figure 6. An IP block, with logic implementing the AMBA protocol, is synthesized and hardened into a macro. The NoC is implemented in the top-level sea of gates, and includes a network interface unit (NIU), which packetizes and depacketizes AMBA transactions from the IP logic across transport links and through NoC switches.

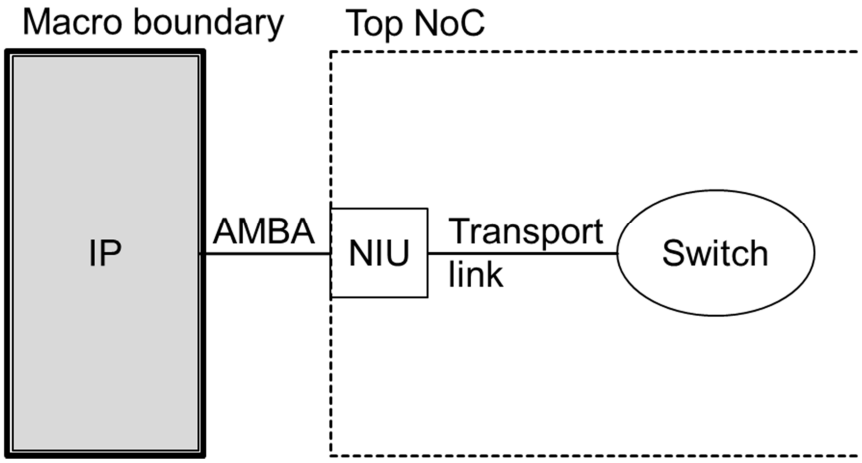

Figure 6: A typical interface between a macro and top level NoC

NoC socket protocol interface

In some cases a macro comprises a subsystem of multiple IPs. Such a macro is shown in Figure 7. Using a NoC for connectivity between smaller IPs within a macro can be beneficial. A subsystem team can use a NoC just as well as the top-level team. Industry-standard transaction interface protocols, such as AMBA and OCP, support a wide range of applications at the cost of a lot more wires than theoretically necessary to meet data transfer requirements. Low-pin count transaction interface protocols exist that are well suited to the goal of minimizing macro ports. Subsystems within NoC should be configured to group cores in macros when possible and use a low pin count protocol to cross the boundary. This gives more flexibility to macro dimensions and more opportunity to locate the socket interface pins in a part of the macro that is better conducive to top-level routing.

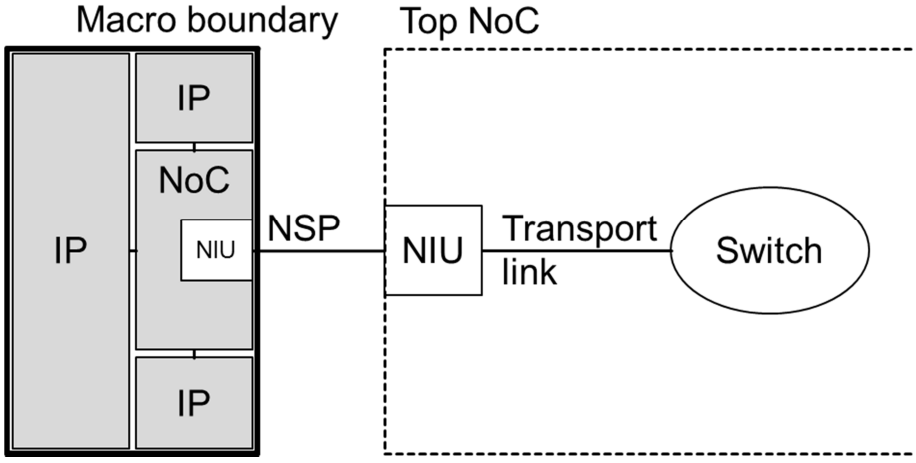

Figure 7: A subsystem NoC connected to a top level NoC through a NoC socket protocol interface.

Packetized transport protocol interface

The packetized transport links of a NoC has relatively few wires compared to any socket transaction interface protocol. Transport links are narrower because they serialize address and data information. More island port wire savings can be achieved by crossing the macro boundary

on a transport link. Such a connection is shown in Figure 8. This requires synthesizing and hardening a network interface unit, which is logically part of the NoC, physically within the macro.

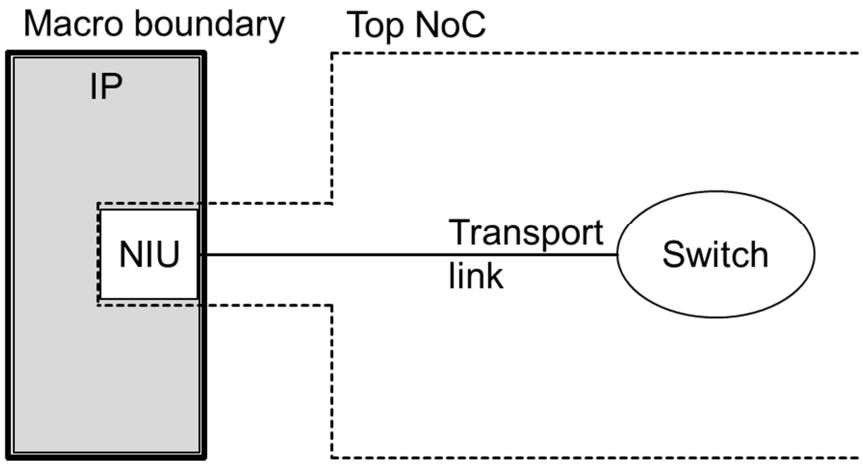

Figure 8: Include the NIU in the macro and cross the boundary on a transport link.

Including the NIU in the macro has the additional benefit of ensuring that its logic is close to the IP socket interface logic. That way the numerous wires of the AMBA socket interface won't be stretched over long distances at the top level by P&R. This also has the benefit of allowing late-in-project floorplan changes by simply adding top-level pipelining to close timing. Further still, this has the benefit of the fact that those top-level pipes are much smaller than transaction interface pipes. A simple transport pipe stage on a 64-bit link is roughly 85% smaller than an AMBA AXI register slice!

However, this approach has several caveats. Reuse of the macro will require the future user to be aware of the need for regenerating the NIU. Furthermore, macro design team to understand a bit about the NoC that is used at the top level. This requires inter-team. Another consideration is that changes to the top level NoC can cause the network interface logic to change. That means that the macro might need to be resynthesized, even up until late stages of the top-level integration process. If these organizational issues can be overcome, the physical benefits of integrating an NIU into each macro are well worth it.

Asynchronous power disconnect interface

In the case where an IP in a macro runs in a different domain than the switch logic of the top-level NoC to which it is connected, adaptation is required. For clock domains there needs to be an asynchronous clock domain adapter, including a FIFO, pointer logic, and resynchronization registers. For power domains there needs to be a power disconnect unit that monitors pending transactions, performs fencing and draining, and does so under the control of the system-level power manager. There are two approaches as to adaptation of clock and power domains between macros and the top level sea of gates.

One approach is to include the power disconnect and asynchronous clock adapter logic in the macros. In that case, a macro clock and top level clock are brought in to the macro. Each team must design and verify their macro with support for the necessary clock adaptation and power

disconnect protocol, including two clock trees and power nets. The macro floorplan must be set up to keep those nets close to its top level interface logic.

The second approach is to put the asynchronous clock crossing and power disconnect logic in the top level logic. This is nice because it makes that logic the responsibility of just one team, and therefore reduces the likelihood of design mistakes. However, it makes the top level design more complex because lots of little bits of clock tree and power net must be planned in appropriate regions, closely aligned to their macros.

We propose a better approach. Use an asynchronous power domain adapter that is designed in two complementary units. These units are native modules within NoCs. Such a design is shown in Figure 9. Synthesize an NIU and one part of the adapter in the macro and the other part of the adapter with the rest of the NoC logic in the top level sea of gates.

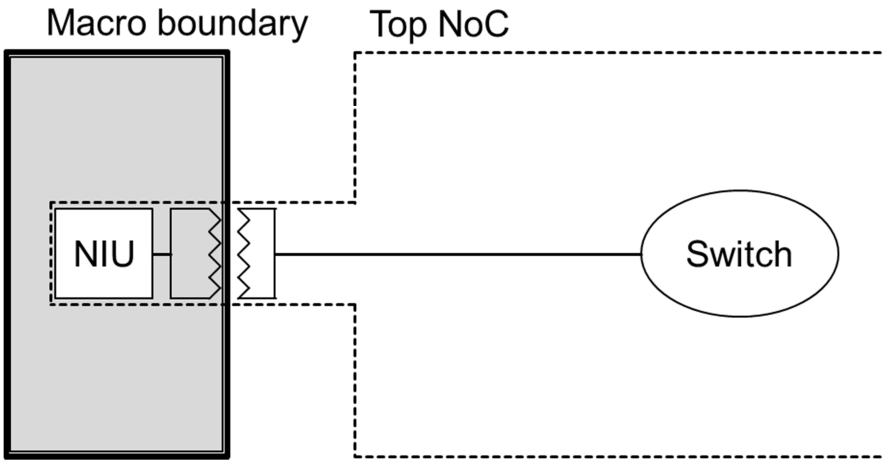

Figure 9: Cross the macro boundary at an asynchronous power domain adapter.

In this case, the macro does not need to have any top level clock input or power net and the top level logic need not have the macro clock signal or power net. As a result, the floorplan can simply assign power and clock domain regional constraints exactly matching the rectilinear shapes of macros in the floorplan. This is great! Not having to balance a clock tree across a macro boundary saves a lot of coordination effort between the top-level team and each subsystem team. A further benefit is that an asynchronous clock crossing has minimal timing constraints. By using a wider but slower interface, constraints can be further relaxed, leaving more flexibility for the placer to move logic around within the sea of gates.

Bear in mind one note of caution. Though asynchronous macro boundaries are good for DVFS, the top-level P&R team should consider the placement of the top-level half of async bridge. It should be placed close to the macro ports for the async interface. This is necessary to meet the max delay constraints on async adapter feedthrough paths. That would not be a concern if the async power domain adapter was implemented fully within the macro (with a portion of the top-level clock tree).

Guideline 5: Pipeline close to macro boundaries to separate logic delay from long wire delay.

Relative to the transport network NIUs have a large number of levels of logic, but the logic is placed physically close together. NoC transport links have relatively little logic, but span long distances.

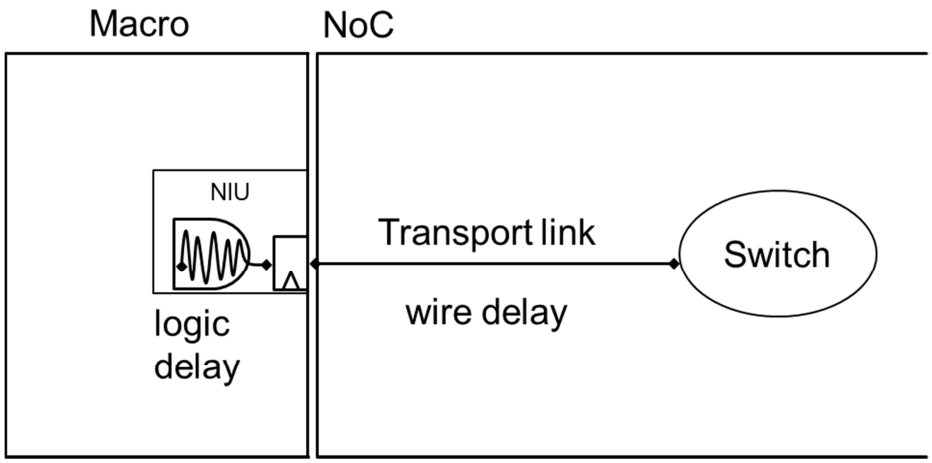

Figure 10: Localized NIU logic delay and long transport link wire delay.

When setting IO constraints for NIUs in a top-level NoC at macro boundaries it is difficult to know how much wire delay will be encountered on the other side of the boundary. This is traditionally addressed by over-constraining the clock period, which leads the designer to add unnecessary pipe stages, and resulting clock cycles of latency, throughout the NoC. Alternatively, to address unexpected long paths during timing closure, a lot of cell upsizing and buffering might be necessary. As we have seen above, with smart design, NIU logic will tend to be placed close to macro boundaries. Therefore, wire capacitance will tend to be small and so synthesis constraints can be relaxed and less cell upsizing will be necessary. This is especially true if pipeline locations within the NoC are chosen in the NIU close to the macro boundary. It is wise to plan a pipe stage at the transport side of each NIU in order to separate its local logic delay from the transport wire delay.

Finally, consider the growing trend towards multi-chip packages and stacked dies. These create new floorplanning challenges. This is particularly true for through-silicon vias (TSV), in which case physical chip IO constraints are no longer only at the edges of the floorplan, but also in the middle. This will favor designs with physically smaller modules that leave more routing space between them. Some developments have been made in the area of inter-NoC connectivity that can enable efficient connectivity on links that are physically constrained within routing channels. These should be considered in the future floorplanning of chips for TSV projects.

5. Other guidelines

Guideline 6: Group sockets that will be close to each other in the logical NoC topology.

A NoC designer should define a topology that corresponds to the physical grouping of macros. Therefore, this guideline provides circular reasoning. However, the point is that you and the NoC designer should get to know each other. For various reasons, such as bandwidth and latency requirements of IPs, certain NoC topologies are preferable, and you can help make that happen. At the same time, you can tell your new friend about the floorplan, and they can insert an appropriate number of pipe stages between macros so that during your final top level integration you will have very few timing violations to fix.

In the floorplan of Figure 2 request channels from many dispersed initiators must be muxed on their way to the DRAM. As a result, the DRAM subsystem is an area of high routing congestion.

That can be alleviated by causing the mux logic to be placed closer to initiators. Long channels are thereby shared from distant regions to the DRAM subsystem.

Performance of latency-sensitive IPs, such as CPUs, depends a lot on how quickly their requests are scheduled in the DRAM scheduler. If CPUs or other latency-sensitive initiators share a port with high bandwidth traffic then the high bandwidth traffic will block low latency traffic to the detriment of quality-of-service and system performance. To reduce that problem, DRAM schedulers and last-level caches have multiple physical ports, each dedicated to traffic of different classes. Low latency initiators have their traffic muxed together separately from the traffic of high bandwidth initiators.

Muxing different classes of traffic and muxing traffic from initiators in different regions will cause interspersed wiring unless the initiators of different traffic classes are physically grouped. In other words, try to keep high bandwidth initiators close to each other and low latency initiators close to each other. Doing so minimizes routing congestion.

Guideline 7: Leave sufficient space around a switch macro

The above guidelines assume a flattened sea of gates at the top level between macros. This is currently the most common approach for SoC designs. However, some chips use a hard macro for the NoC switch logic. In such chips, essentially all hard macro ports are connected directly to other hard macros. Some space is required between the macros to allow for routing wires.

Macros connected at synchronous interfaces have a high likelihood of needing pipeline stages. Macros connected at asynchronous interfaces require more I/O ports and wires. Be aware of the overheads of the sync or async boundaries between macros, and place the connected hard macros far enough apart to accommodate pipeline registers or wider buses.

6. What not to do

Above we have discussed ways to floorplan the chip in order to get better results from a NoC. However, there are some things that NoCs do well that can make your floorplanning job easier.

Guideline 8: Don’t worry (too much) about top-level time budgeting.

We have discussed timing and how it is impacted by wire delay. One thing that NoCs do well is to allow configurable pipelining. If timing is not met on the long wires of a link, any number of pipe stages can be added at any points on the link. This means that if the synthesized hard macros exceeded their timing budgets, when top-level wire delay is included, the unexpected timing failures can be easily fixed, even late in the project. The effect on critical paths of distance between macros interface ports should be a minor concern when following the other guidelines in this paper.

However, it is necessary to leave some space available for pipeline registers to be added late in the project schedule. Ensure that the area for the sea of gates is not too dense. Pipe stages take approximately the area needed for registering one full word of data. That will typically be 32, 64, 128, or 256 DFFs.

7. Conclusion and summary of guidelines

The NoC is critical to both the system performance of the chip and to ease of late-stage top-level place & route. With careful floorplanning, the NoC can be made even more optimal. Following some simple guidelines can help a chip design team to achieve better quality results, and to finish the project more quickly and easily.

We hereby recommend the following guidelines:

1. Physically distribute the memory subsystem.
2. Place low latency and high bandwidth IPs close to memory subsystem.
3. Group sockets that are in shared domains close together.
4. Cross macro boundaries on packetized links.
5. Pipeline close to macro boundaries to separate logic delay from long wire delay.
6. Group sockets that will be close to each other in the logical NoC topology.
7. Leave sufficient space around a switch macro
8. Don't worry (too much) about top-level time budgeting.

8. Acknowledgements

This paper is the result of a great collaboration. It is based on the collective experience of the Arteris worldwide support team on more than 100 chip projects. The team is: Benoit de Lescure, Alexis Boutillier, Xavier van Ruymbeke, Jonah Probell, Monica Tang, William Tseng, Byungchul Hong, Brian Huang, Sumie Aoki, Harris Byun, Maryam Bahmani, Aimad Rhatay, Dee Lin, and Xinyu Li.