A Quality-Driven Design Approach for NoCs

Stephan Bourduas, Jean-Samuel Chenard, and Željko Žilić
McGill University

Editor's note:
This article advocates a systematic approach to improve NoC design quality by guiding architectural choices according to the difficulty of verification and test. The authors propose early quality metrics for added test, monitoring, and debug hardware.

—Yatin Hoskote, Intel

To achieve performance, feature, and cost goals set by the marketplace, engineers must build increasingly large and flexible systems in ever smaller and more efficient devices. This trend has led to the introduction of versatile SoCs, which can include many disparate electronic subsystems. In advancing SoCs, the emerging network-on-chip (NoC) concept plays an important role, as it will ultimately allow scalability and modularity in hardware. Architectural design and modeling of NoCs increasingly requires a cross-layer design methodology that spans multiple abstraction levels, as decisions taken at a high level inevitably have a significant impact on the final implementation's physical features. In addition to the traditional speed and area concerns, designers must model myriad other metrics, including energy consumption, noise resilience, reliable connectivity and interoperability, manufacturability, and bug-free operation across the system. All of these will ultimately determine the quality of the system during its life cycle.

Here, *quality* refers to achieving sufficiently correct operation across abstraction layers and implementation technologies. Examples include

- performing required operations under increased coupling noise or defects of new and emerging technologies;
- correctly transmitting bits at high speeds or across poor (such as wireless) communication channels; and
- being sufficiently free of logic and software bugs across increasingly large, distributed systems.

Without quality built in from the early stages of modeling, the staggering complexity and the reduced room for error are seriously affecting the ability of industry to make profitable products on time. For example, Intel and AMD have repeatedly had to recall or postpone their processors, including incidents as recent as December 2007, when the energy consumption of processors already on the market was found to be excessive in some corner cases. As a consequence, close to 25% of all design resources at Intel are now expended in post-fabrication validation.[1]

This article presents a development platform that facilitates quality-driven design via multi-abstraction-level modeling. The NoC concept not only simplifies the overall integration of cores but also provides the possibility for executing test, debug, and *CPU control* (the ability to control the execution of many CPU cores, loading, breakpoints, and so on) from a single access point. We can then monitor the transactions and highlight any deviation from the specifications at the highest possible abstraction level by incorporating assertions in the simulation and emulation platforms. This monitoring requires multiabstraction support, and we conduct it in real time using on-chip silicon-based checkers. The checkers are derived automatically from modern languages such as Property Specification Language (PSL) and SystemVerilog Assertion (SVA). In the emulation environment (or FPGA prototype), many assertion checkers can be instantiated to further improve visibility. Some of those checkers can remain in the final implementation, depending on an acceptable area overhead (that is, a reasonable cost for increased quality).

0740-7475/08/$25.00 © 2008 IEEE Copublished by the IEEE CS and the IEEE CASS

To master the NoC and SoC development challenges, we validate the behavior of complex systems, including embedded software, by using emulation and prototyping on FPGAs across abstraction levels in a cosimulation environment. Our goal is to improve quality of design (QoD). Therefore, we are developing design tools and methodologies that enhance the verification effort from the conception of the system model to the FPGA prototype.

Quality of design

The complexity of designing efficient, scalable on-chip communication interconnects will continue to grow as increasing numbers of cores are integrated onto a single chip. A major challenge in chip design will be to provide a scalable and reliable communication mechanism that will ensure correct system behavior.[2,3] Traditional SoC designs have used shared-medium (bus-based) architectures, whose limitations have now become apparent.[4,5] In fact, for systems consisting of more than 20 cores, a bus interconnect quickly becomes the system bottleneck,[6] degrading performance such that it is no longer a feasible solution to the communication requirements. The key problem with bus-based approaches is their limited scalability. To address the shortcomings of the shared-medium architecture, designers have adapted concepts from the domains of networking and parallel processing for on-chip use, giving rise to the notion of an on-chip communication network, or NoC.[2,3]

Although the NoC concept addresses the shortcomings of shared-medium architectures, the vast NoC design space adds complexity to the design flow. Specifically, because the network topology significantly impacts performance and cost, the topology selection must now be included as part of the design space exploration and high-level prototyping stages. Typically, designers start with high-level functional or transaction-level models for rapid prototyping and design space exploration. The high-level models are then refined until they can be synthesized to hardware. The traditional verification effort is usually left until the final stages of development, and is usually used to verify correct system behavior for a certain range of possible system inputs (functional coverage). Furthermore, DFT, another significant quality factor, is not integrated into architectural exploration, where there is now a possibility to reuse the NoC as a test access mechanism.[7]

Functional coverage is used to verify system behavior before the system is implemented in hardware. Although a well-tested system is of higher quality than a poorly tested one, functional coverage does not help detect error conditions in the field. Therefore, besides performing functional coverage, another way to improve system quality is to add monitoring capabilities to enable detecting and properly dealing with runtime errors after fabrication.

At first glance, it would seem that improving quality by adding test, debug, and monitoring infrastructure would be relatively straightforward. Intuitively, all that is needed is to add the infrastructure IP and possibly some redundancy to compensate against failures. The difficulty lies in integrating the quality effort into the traditional design flow in a systematic approach to improving design quality during the pre- and post-fabrication stages. There are two aspects of design that improve the overall quality: verification and the testing, monitoring, and debugging (TMD) infrastructure.

Verification can be performed at the block and system levels. Block-level verification checks that individual components conform to specifications, and system-level verification checks that the components function correctly when interacting with one another.

The TMD infrastructure has the following characteristics:

- Reusing an NoC as a test access mechanism poses challenges in guaranteeing the bandwidth and latency for streaming the tests to all subsystems.[7] The communication bandwidth and processing capabilities of the processing elements (PEs) are needed to predictably route test data, but possibly also for the self-tests and online testing.
- Runtime monitoring can be added to detect error conditions that might be caused by unverified corner cases or by the presence of a timing fault or silicon defect that might have escaped initial testing.
- Integrated debugging hardware enables the system to diagnose the cause of errors, and to react appropriately (for example, reroute traffic around a faulty node). The visibility gained by the presence of the debugging modules facilitates the localization of the root cause of

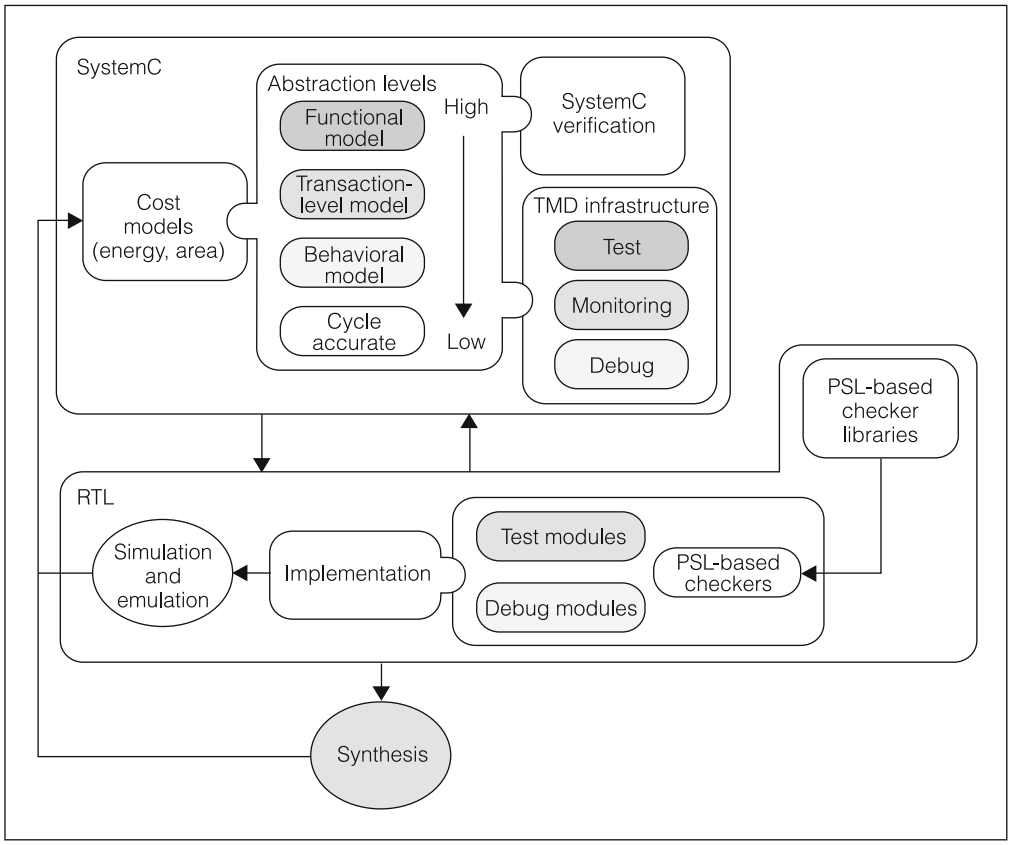

Figure 1. The quality of design (QoD) flow incorporates the system debug and monitoring infrastructure via debug and assertion modules, and reuses the network on chip (NoC) for test and verification. (PSL: Property Specification Language; TMD: testing, monitoring, and debugging.)

problems during both pre- and post-fabrication debug.

Figure 1 shows our proposed design flow, which integrates verification and the TMD infrastructure.

Usually, system-level verification is performed after block-level verification. So, in addition to verification typically taking place late in the development process, system-level verification usually occurs last. This has profound implications for NoC development because of the paradigm shift from computation-centric to communication-centric design.[3] That is, the performance of current SoCs is limited mainly by the computational cores, so the system interconnect can be selected almost as an afterthought. Conversely, tomorrow's large SoCs will be limited by the available communication bandwidth, thus making the design of the interconnect architecture critical. If system-level verification occurs at the latter stages of development, problems with a chosen interconnect architecture may be discovered too late. It is therefore necessary to begin system-level verification as early as possible, and to include it in the architectural exploration cycle. Moreover, verification components can be reused and refined during the implementation phases for regression testing.

Essential to the monitoring effort is the ability to detect temporally complex error conditions. Electronic system-level (ESL) design and RTL languages and their simulation environments support assertions in a way similar to modern programming languages. There are two problems with those simple assertion mechanisms:

- The assertions can be expressed only by using simple Boolean conditions.
- The assertion logic cannot be directly translated to hardware.

To address the problem of the limited complexity restrictions of standard assertion mechanisms, the industry introduced assertion languages such as PSL, which supports temporal expressions and thus can express complex sequences of events over time. This is an important capability for NoC verification because transactions can be monitored at varying levels of abstraction and granularity, ensuring conformance to specifications. Without a tool to translate PSL statements to hardware, there is a gap between the monitoring capabilities of the simulation environment and the final hardware implementation. In our view, system quality could be improved by incorporating the PSL-represented logic in all stages of specification and design, with some logic left in the final implementation for online monitoring. Therefore, our designs include PSL-derived hardware checkers.[8,9]

To complement the checker hardware, we added debug modules, as shown later in the article. Thus, when the monitoring hardware detects an error condition, the system diagnoses the problem and takes appropriate action.

Design space exploration is the process of comparing the properties (area, speed, power, and so forth) of several system configurations and selecting the most appropriate candidate. Figure 1 shows how high-level cost models are integrated into the SystemC prototyping phase to help drive early design space exploration. We developed the cost models by using FPGA and ASIC synthesis results of specific design instances. For example, we can use the synthesis results of a specific configuration of a hierarchical-ring interconnect to develop a cost function that takes as input certain design parameters (for example, bus width and FIFO buffer depths), and allows comparing different configurations without synthesizing each instance.[10] Using cost models early in the design phase increases the probability that the final implementation will meet resource and performance constraints, and thus is an important step in improving QoD.

Although verification is an extremely important part of the usual design flow, it does not impact the resource usage of the final implementation. However, the TMD infrastructure in our QoD flow will eventually be implemented in hardware. Therefore, the overhead of the TMD infrastructure must also be characterized and included in the cost models so that its resource requirements are accounted for during the design space exploration stages. Also, TMD cost models can serve to optimize the included TMD components to meet resource constraints. For example, it is probably too expensive to include all assertion checkers or instantiate debug modules at every node in the network.

Quantifying quality

So far, the discussion has centered on the idea of improving quality by adding the TMD infrastructure to the design flow. It is obvious that adding extra diagnostic and recovery capabilities to a design will improve its quality, but how do we quantify the improvement? And, how can we compare the quality of two different designs when performing architectural exploration? A representative quantitative measure lets us evaluate the quality of several architectures and select the one with the best score. We use the following quality function to score the quality of a specific design instance d_i:

$$Q(d_i) = \lambda Q_V(d_i) + \rho Q_{TMD}(d_i) + \sigma Q_{NoC}(d_i) \quad (1)$$

where Q_V numerically represents the verification quality of the components in the design; Q_{TMD} is a measure of the quality of the TMD infrastructure hardware; Q_{NoC} is a quality measure of the network architecture and topology; and λ, ρ, and σ assign relative weights in the calculation of Q. The term Q_V can be a combination of the functional coverage of each component and the completeness of the system-level verification effort (the higher the coverage, the higher the quality score).

To facilitate the comparison of architectures with differing numbers of nodes, we can express Q_V and Q_{TMD} as an average value per node. For example, we compute the value of Q_{TMD} by solving

$$Q_{TMD}(d_i) = \frac{\alpha Q_T(d_i) + \beta Q_M(d_i) + \gamma Q_D(d_i)}{|d_i|} \quad (2)$$

where Q_T, Q_M, and Q_D are the quality scores of the test, monitoring, and debug hardware; and α, β, and γ assign relative weights in the calculation of Q. Dividing by $|d_i|$ (the number of nodes in the architecture) yields an average Q_{TMD} value per node. The calculation of Q_{TMD} described by Equation 2 assumes we can numerically represent the capabilities of the TMD infrastructure. One way to score assertions is to simply rank each one using a numeric value. (The calculation of Q_{NoC} will be discussed later.)

The cost of quality

Under ideal circumstances, we'd like to add as much extra TMD hardware as possible to maximize quality. However, the added functionality comes at the cost of extra resource requirements, which may exceed the resource budget for the design. Therefore, we must find a way to balance quality against the required overhead. We express the resource requirement of a design instance d_i as

$$R(d_i) = R_0(d_i) + R_{TMD}(d_i) \quad (3)$$

where R_0 is the resource requirement of the design without TMD infrastructure, and R_{TMD} is the resource requirement of the TMD infrastructure components.

Optimizing quality versus cost

In addition to integrating the TMD infrastructure to increase quality, we can also use our design flow (Figure 1) to support design space exploration and optimization to meet performance and resource constraints. Aided by FPGA and ASIC synthesis results, we developed cost models so that we could drive exploration at higher abstraction levels using SystemC instead of the RTL. The feedback loop in Figure 1 shows how we use synthesis results from the lower abstraction levels at the SystemC level to develop the cost models. Only a subset of the possible configuration of parameterizable components are synthesized to obtain a few data points, which can be used to infer relatively accurate estimates for the range of possible configurations. This is key to enabling rapid high-level exploration and optimization, because synthesizing every design instance and comparing resource requirements would be far too time-consuming.

Under resource constraints, optimizing quality versus cost (resource requirements) is a multiobjective optimization problem, with the goal of maximizing quality and minimizing cost. We define the objective function for optimization as the ratio Q:R. Thus, we find the optimal design d_i by solving

$$\max[Q(d_i)/R(d_i)] \quad (4)$$

The RTL box in Figure 1 shows how we select the checker and assertion hardware from large libraries. (A significant portion of this hardware comes from reusing the verification assertions.) A practical design has resource constraints, so only a subset of the TMD capabilities goes in hardware. Therefore, the challenge is to efficiently select the best subset such that the quality is maximized for the R_{TMD} resource constraint.

FPGA emulation in quality-driven architecture exploration

FPGA emulation and prototyping can provide a decisive advantage to the multiprocessor and computer architecture research. Using this platform, we have proposed and studied several NoC topologies that augment the transaction-level SystemC models with RTL modules to be run directly on FPGAs. The modeling can be made more accurate when needed. However, dealing with the inherent complexity and difficulties in validating a given proposal is possible only through accurate models executed at the speed of FPGA circuits (for an acceptable Q_V). The concrete NoC studies completed so far describe how to create the composite topologies of the popular mesh NoC using hierarchical rings in several different ways,[11] and how to evaluate them realistically. This is an extremely interesting NoC proposition, given the inherent planarity of hierarchical rings.

The hierarchical ring offers fewer physical links connecting the NoC's PEs. This is quite beneficial in any emulation platform because it reduces the use of the limited inter-FPGA links. A full crossbar switch or mesh network would overwhelm the emulation platform hardware connectivity and would require hardware workarounds such as time-division multiplexing to support the very large number of links as the design is partitioned into multiple FPGAs.

There are, however, limitations in using only direct circuit emulation without quality considerations. Therefore, we augmented the NoC architectural studies with the energy and area models, derived from more refined RTL block implementations. Our quality considerations benefit significantly from FPGA emulation to advance architectural and related research and offer a flexible testing ground to refine our quality metrics.

Networking and quality of service

Central to the theme of the NoC, the networking protocol and data structures require careful considerations to incorporate support for the TMD infrastructure. The networking header definitions must include fields so that the routers can autonomously terminate transactions that are destined for the monitor and debug units or the router itself. The NoC must support virtual channels that allow differentiated services of TMD traffic. One such service is the transport of assertion information, which is characterized by small messages (low bandwidth) requiring low latency and best-effort delivery. To support debugging, the NoC needs at least one communication channel that can bypass the flow control and compel delivery of the payload without considering the congestion level in the buffers (for example, overwrite user data to end a deadlock situation). These bypass channels are necessary for diagnosing complex error conditions in the NoC and for controlling the PEs at a very low level.

The ability to completely bypass the router logic in the presence of a physical defect in either the router or the PEs could help increase the yield if the full NoC

capabilities are not required (say, for reduced functionality or a lower-cost device). A time-to-live (TTL) counter is also necessary in the networking payload so that packets destined to a disabled endpoint don't endlessly loop in the network. For an arbitrary NoC architecture, it's possible to predetermine the maximum number of hops, and then adjust the TTL counter size accordingly. Built-in performance monitors report expired packets to the software layers in order to detect incorrect routing tables. Because the network includes the necessary interfaces to assume control of the PE resources (mainly the CPU), debugging and diagnostic transactions can be initiated while the NoC is still operating with one (or more) of the PEs in bypass mode. In many applications, this will mean a reduced level of service to the user, but not a complete failure of the NoC.

Architectural exploration

During the initial development stages, designers must select an NoC architecture that meets the performance and cost requirements of the application. Using the high-level models, the designers can discard unsuitable candidates before adding TMD hardware and performing the quality optimization discussed earlier.

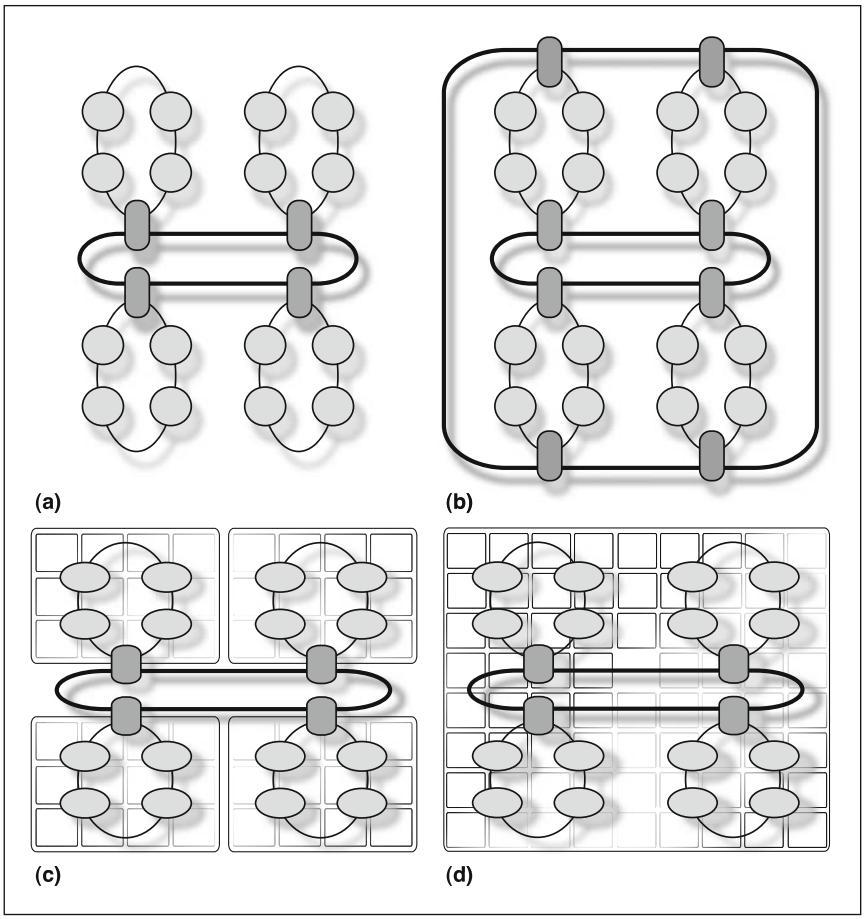

Figure 2. Examples of topologies modeled using our platform: two-level hierarchical rings (a); 2D hyper-ring (basically a hierarchical-ring architecture with a second global ring added) (b); hybrid architecture connecting several mesh networks together (c); and augmented network, routing global traffic through the hierarchical rings to reduce latencies and hop counts (d).

To facilitate architectural exploration, we developed our own NoC simulation platform, written in SystemC. This platform contains a library of parameterizable components for constructing arbitrary network topologies. We've integrated previously developed energy models, as well as area estimates derived from FPGA synthesis and more recent ASIC synthesis results.[10] The augmented high-level models are useful for comparing the performance characteristics of arbitrary network topologies for the target application, letting designers quickly discard unsuitable candidates early.

Figure 2 shows examples of topologies we have modeled using our platform. Figure 2a shows a two-level hierarchical-ring topology, and Figure 2b shows a 2D hyper-ring topology. Figure 2c shows our hybrid ring-mesh architecture,[11] which uses a two-level hierarchical-ring interconnect to route global traffic between disconnected mesh networks. The augmented ring-mesh architecture shown in Figure 2d uses the same hierarchical-ring interconnect to augment a single large mesh network; the ring interconnect reduces the latencies associated with long-distance traffic by skipping over portions of the mesh network. The inherent planarity and energy consumption benefits of hierarchical rings make these topologies very attractive. Furthermore, using RTL models, we've demonstrated that the hierarchical rings are fast and area efficient,[10] making them well-suited for NoC implementation.

Using our QoD flow, we can evaluate different topologies. Here, we focus on the hierarchical-ring and

hyper-ring topologies, but other topologies could be compared as well. The benefit of comparing two ring architectures is that we have high-level (ESL) and low-level (RTL) implementations of each, so we can use the full QoD flow to select a final implementation.

Hierarchical-ring topology

We constructed the hierarchical-ring topology in Figure 2a by arranging several unidirectional rings around a central global ring to form a two-level hierarchy. The unidirectional ring is the simplest form of a point-to-point interconnection, resulting in a minimum number of links per node and simple interface hardware. The simplicity of the rings requires a straightforward routing mechanism so that the only decision remaining is whether to remove a flit from the ring or forward it to the next node. Hence, each switch's buffer requirements are reduced, the maximum speed at which each switch can operate is increased, latencies are reduced, and the point-to-point links between nodes are better utilized.

Although the simplicity of the unidirectional rings makes it an attractive architecture, it has the drawback of limited scalability. The diameter of the network grows linearly with network size; hence, hop counts and latencies can become unacceptably large. To attenuate the scalability issues of the single large ring, designers should consider hierarchical-ring topologies, which have greater scalability and smaller network diameters. In fact, the diameter of a hierarchical-ring topology grows logarithmically with the number of nodes.

The topology in Figure 2a consists of four local rings and one global ring for routing traffic between local rings. Each local ring consists of several ring interface (RI) components, which interface with the PEs, and an inter-ring interface (IRI) to connect the local ring to the global one. The hierarchical-ring topology shares the same characteristics of the single unidirectional ring that are important for NoC implementations. The low node degree of the switches results in simple, fast, and area-efficient routers.[10] For example, the degree of each RI is 2, and the degree of each IRI is 4. The low node degrees of the hierarchical rings result in a planar topology, which is well-suited for efficient 2D layout. In fact, the topologies discussed in this article have no global routing. Consequently, place-and-route results in a layout that has no global channels.[10] The unidirectional nature of the rings reduces the overhead associated with routing and thus results in low latencies and high throughput.

Hyper-ring topology

A drawback of the hierarchical-ring topology is that the global ring can saturate quickly under heavy load. An alternate ring topology is the *n*-dimensional hyper-ring,[12] which addresses the bisection bandwidth limitations of the hierarchical rings. The drawback of higher-dimensioned hyper-rings is that they lose the property of planarity and can thus become difficult to map efficiently onto silicon. Therefore, we've chosen the 2D hyper-ring shown in Figure 2b, because of its similarity to the two-level hierarchical-ring architecture.

In Figures 2a and 2b, the hyper-ring architectures are basically an augmented version of the hierarchical-ring architecture, with a second global ring added. Simulation results confirmed that the increased bandwidth afforded by the additional global ring lowered average latencies by 20% to 40%, depending on the simulation parameters; it also increased the saturation point for accepted traffic by 10% to 15%.

PE ring interface architecture

The RI components provide an interface to the network through which a PE can send and receive data. Figure 3 illustrates the internal blocks resulting from the design flow presented previously. Notice that the ring interface not only manages the add/drop traffic flowing in and out of the PE but also monitors and autonomously reports assertion failures and performance information. This network status reporting is decentralized and consumes only a small fraction of the bandwidth. Because the NoC transports the aggregated information like other traffic, the NoC can easily reroute this information or give it a higher priority to ensure low latencies.

We designed hardware units such as the network performance monitor using ESL design methods (or hand-coded RTL, when necessary). However, we derived the hardware assertion checkers and coverage monitors from high-level temporal statements.

Hardware assertion checkers

Recently introduced design verification languages such as IBM's PSL have allowed tremendous improvements in verification productivity, leading to a rapid

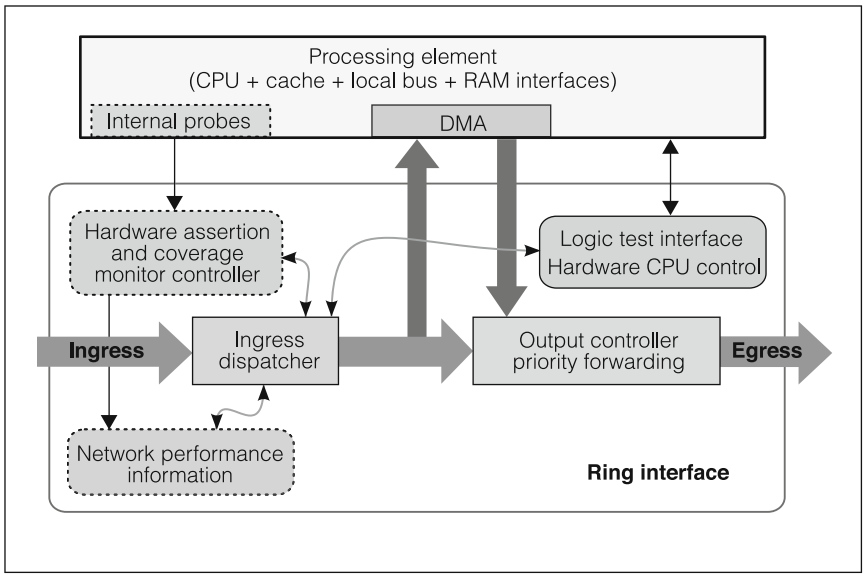

Figure 3. The processing element (PE) ring interface, which includes hardware dedicated to providing the interface between the debug control and monitors, and the networking infrastructure. (DMA: direct memory access.)

increase in the use of assertions. We use the MBAC tool to compile PSL assertions to hardware for immediate use in emulations, monitoring, and silicon debug.[8,9] Because assertions are the most suitable means of specifying complex temporal relations, this way of instrumenting circuits has far-reaching applications in monitoring, debugging, and otherwise assessing and improving the quality of digital systems. We have developed a complete new framework for debugging circuits during pre- and post-fabrication verification. To efficiently solve the synthesis problem and increase visibility, this project includes several fundamental contributions to the realization of nondeterministic acceptors of extended regular expressions, including new ways to handle nondeterminism and new automata symbol-encoding schemes.

From PSL to hardware

Each hardware assertion checker representing a PSL statement can be augmented with specialized debug enhancements,[8] such as activity monitoring or sequence completion monitoring, by passing command-line options to the assertion compiler. Designers could implement their own code for validating the internal state of the circuit, but an automated translation from PSL assertions to gates eliminates the risk of altering the behavior of the circuit with the checkers. The MBAC assertion compiler generates hardware that only monitors the circuit behavior, thus guaranteeing that the hardware behaves as originally intended. Furthermore, automatically translating PSL assertions to RTL code eliminates the risk of introducing errors into the assertion circuitry itself, which likely would occur if the translations were done manually. Any errors introduced into the assertion hardware during translation would affect the resulting quality of the hardware. Therefore, an automated approach is not only preferable but also necessary for large, complex systems such as NoCs.

The methodology also benefits from the uniform representation of assertion failures and high designer productivity because the temporal languages can concisely describe complex behavior. Furthermore, the PSL statements database and NoC topology information can be combined within a more advanced debugger to automate the localization of failures.

Tool flow details

Our approach to the design of NoC diagnostic and failure detection modules includes the following steps, illustrated in Figure 4:

1. Beginning with verification units containing PSL assertions derived from the high-level modeling effort, we select those suitable for translation to hardware.
2. The hardware assertion-checker generator (MBAC) processes those assertions, then the RTL synthesizer extracts the size and timing characteristics of the hardware checker circuits.
3. The resulting information is recorded in a database along with a quality score given by the designer. The timing is checked to ensure that the assertion checkers do not interfere with the circuit timing budget.
4. New PSL statements can be written to ease debug and diagnosis of potential error condi-

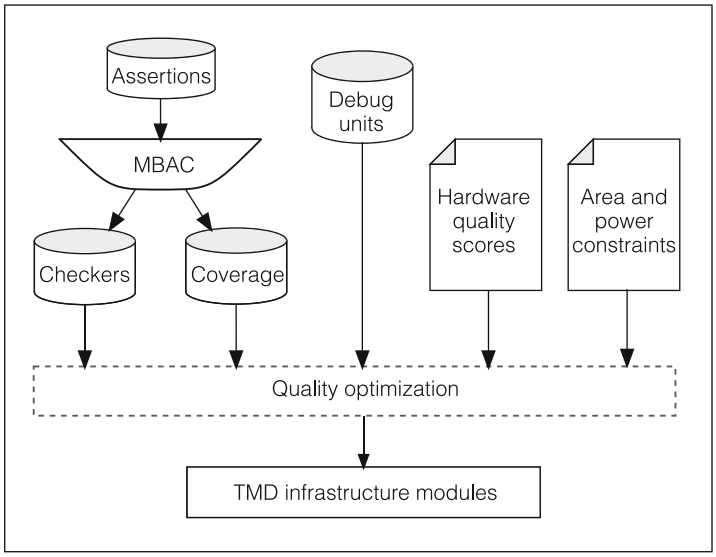

Figure 4. Quality-enhancing monitors and debug units creation are automated via the MBAC tool, which translates assertions to hardware checkers. The quality-optimizing process selects the best hardware units that can be instantiated in the design. The bases for this selection are the scores and constraints given by the designers for the PSL assertions and debug modules.

tions, and these statements go in the assertion database. A flit tracer or a checker for corner cases that are very long to get in a simulation could be added to improve visibility in the hardware implementation. Coding these additions in PSL is straightforward.

5. The interfaces to the NoC interconnect and the PEs are generated. This produces the TMD infrastructure module, including the assertion checkers and interface logic. A byproduct of this process is a database mapping the assertions' bit positions and addresses for integration with the controller and the rest of the system.

Interface generation (step 5) is performed in the following manner:

1. The hardware assertion checker outputs and monitors are packed so that they can fit in a given number of words of a certain bus width (usually 16 or 32 bits). Monitors can be set to include counters, which are placed to avoid straddling two registers.
2. A transition detector is generated for each assertion checker. It is a simple state machine that detects the transition from the idle state to a detected assertion failure. Because numerous assertions are simultaneously monitored, this method avoids flooding the network with information if assertions repeatedly fail. Only the initial failure is sent. Other firings of the same assertion checker can be counted but are not sent to the network (until the PE resets the transition detector).
3. An interface is generated to the local PE so that it can control, reset, and examine the assertion checker outputs and counters. This local interface also guarantees that the assertion failure is available in case the network is deadlocked or buggy (the PE interface can be read via the scan chain or a similar mechanism).
4. Finally, an interface to the network is generated. It consists of a hardware controller that continuously scans the assertion transition detectors and autonomously generates a worm (packet) destined to a central unit when transitions are detected.

Thus, the designer selects and ranks assertions on the basis of a subjective quality evaluation. Our tool chain can handle the transformation to hardware, optimized selection, and organization of those assertion checkers in hardware. It also provides a database that can relate a particular assertion failure to an entry that details the error back to the original assertion, thus assisting in debug.

Comparison of the hierarchical- and hyper-ring topologies

The following example illustrates our proposed quality-driven flow's usefulness for comparing the quality versus cost of the hierarchical-ring and hyper-ring topologies, and explains how we integrated the TMD infrastructure hardware into the NoC. To compare the quality of both architectures, we consider the individual terms of Equation 1.

Quality of verification

As previously discussed, the value of Q_V from Equation 1 is meant to quantify the thoroughness of the verification effort performed on each individual component as well as at the system level (that is, Q_V can be expressed as a function of block- and system-level verification). Because both architectures use the same components (RIs and IRIs), the block-level

verification value contributed to Q_V is approximately the same for each architecture. Furthermore, the similarity of the architectures lets us reuse the same application code and synthetic testbenches to perform system-level verification. Therefore, we can reason that

$$Q_V(d_{\text{hierarchical}}) \approx Q_V(d_{\text{hyper}}) \quad (5)$$

For this evaluation, we assume that they are close enough to be considered equal.

Quality of TMD infrastructure

The quality index of the TMD infrastructure is different for the two architectures because the monitoring and debugging capabilities are better in the hyper-ring architecture. The calculation of Q_{TMD} depends on several factors, such as location (or placement) and capabilities of the TMD infrastructure hardware. For example, the placement of debug and monitoring hardware at the inter-ring interfaces brings more quality to the design because these interfaces are the bridge between the global and local rings; hence, more traffic passes through those nodes. The number of network input ports for each component can serve to quantify the Q_{TMD} values of the RI and IRI components as

$$\begin{aligned} Q_{\text{TMD}}(d_{\text{RI}}) &= 1 \\ Q_{\text{TMD}}(d_{\text{IRI}}) &= 2 \end{aligned} \quad (6)$$

The hierarchical-ring architecture consists of 16 RIs and four IRIs (20 nodes), and the hyper-ring architecture has 16 RIs and eight IRIs (24 nodes). Using Equations 2 and 6, we find that the quality index for the TMD infrastructure for both architectures is

$$\begin{aligned} &Q_{\text{TMD}}(d_{\text{hierarchical}}) = \\ &\frac{16Q_{\text{TMD}}(d_{\text{RI}}) + 4Q_{\text{TMD}}(d_{\text{IRI}})}{20} = 1.20 \\ &Q_{\text{TMD}}(d_{\text{hyper}}) = \\ &\frac{16Q_{\text{TMD}}(d_{\text{RI}}) + 8Q_{\text{TMD}}(d_{\text{IRI}})}{24} = 1.33 \end{aligned} \quad (7)$$

Equation 7 assumes that all components in the architecture contain TMD infrastructure hardware, so the comparison is straightforward. However, under resource constraints, we might wish to restrict the number of IRI and/or RI components that are TMD enabled, thereby resulting in an asymmetric distribution of TMD hardware. In this case, the comparison and selection between the two architectures becomes an optimization problem with a potentially large solution space.

Quality of NoC architecture

The calculation of Q_{NoC} from Equation 1 is an open problem because there are many network characteristics that can be taken into account when comparing the quality of two NoC architectures. For example, the bisection bandwidth, an often-studied property, could be included along with the node degrees, network diameter, path diversity, and so on. For this example, it's sufficient to evaluate the relative quality of each architecture by using our understanding of the different architectures' properties. The hyper-ring architecture has a higher bisection bandwidth and greater path diversity. This is important for debug because one global ring can be reserved for debug traffic to provide quality of service. In our example, the property of path diversity is of primary concern, so we express the quality of the NoC topology as the number of paths between any two nodes in the network. For the two architectures under consideration, we define Q_{NoC} as

$$\begin{aligned} Q_{\text{NoC}}(d_{\text{hierarchical}}) &= 1 \\ Q_{\text{NoC}}(d_{\text{hyper}}) &= 2 \end{aligned} \quad (8)$$

where the hyper-ring architecture has two possible paths between any two nodes.

We've deliberately kept our representation of Q_{NoC} simple for illustration purposes, but for a real application, we would factor in other architectural characteristics such as node degree, network diameter, and so forth. The difficulty in defining a specific architecture's quality index lies in the many properties that can be taken into account. Also, the importance of each property can vary, depending on the application, problem domain, resource and packaging constraints, and so forth.

Hardware resources and quality

We distinguish between hardware checkers derived from assertions and debug units that are purposely designed. Hardware checkers derived from assertions tend to focus on a very particular aspect of the design, such as a protocol property that should remain valid at all times. One such example would be

Table 1. Area and power comparison of quality in the hierarchical- and hyper-ring topologies for operating frequencies of 500 MHz and 250 MHz.

Topology	Operating frequency (MHz)	Total cell area (mm^2)	Total power (W)	Quality, Q	Area score, $Q{:}R_A$	Power score, $Q{:}R_P$
Hierarchical-ring	500	5.10	2.26	2.20	0.43	0.97
	250	4.95	1.11	2.20	0.44	1.98
Hyper-ring	500	5.98	2.66	3.33	0.55	1.25
	250	5.82	1.31	3.33	0.57	2.54

the following PSL statements, which check for valid worm length and proper ending of worms in an NoC:

```
property ValidWormLen =
  always StartOfWorm |=>
  { [*63] ; EndOfWorm };
property ExpectEnd =
  {StartOfWorm |=> eventually!
  EndOfWorm abort AbortedWorm};
assert ValidWormLen;
assert ExpectEnd;
```

The **ValidWormLen** checker, when translated to hardware, would be quite small (comparable to a 6-bit counter). The **ExpectEnd** checker would use only one to two flip-flops (depending on MBAC output-buffering settings) and a few gates of logic. The Q_{TMD} value for those checkers could be weighted quite highly. Because their hardware overhead is small, they would be good candidates to be preserved in silicon.

Our tool flow groups the hardware assertion checker outputs into vectors at the PE interface. The hardware then sends a special worm that encodes the transitions in those assertion vectors to a designated node in the NoC for further processing and fault localization. Because the checkers are generated from PSL or SVA assertions, translated to hardware, and combined into a list of registers, fault localization takes place by correlating assertion bit position and network location with the assertion details. Those details include the original PSL expression, a reference to the specification from which the assertion was derived, and any comments left by the designer above the PSL expression. A faulty behavior in an NoC module triggers the chain of events just discussed and produces a meaningful error message. One or more assertion failures in operation might indicate, for example, that a certain part of the circuit is faulty (timing errors, defective gates, and so on). The NoC debug infrastructure can then use local scan-test access to run more-specific tests in order to help isolate the fault. Figure 3 shows the hardware modules resulting from this tool flow.

Debug units, on the other hand, are designed to provide advanced capabilities beyond what can be created from assertion statements. In an NoC, an example would be a worm back-trace buffer that can memorize the last n worms routed out of a module along with a time stamp of the interval between those worms. Such a hardware monitor can be quickly designed in an ESL flow and would imply some memory (n entries deep, and with a bit width capable of holding routing information about the worm and its time stamp). The Q_{TMD} of this debug monitor could be high, but so would its area overhead. A good compromise would be to instantiate this debug monitor in only a specific subset of stations (located at key points in the NoC) such that those stations would benefit the most in troubleshooting a problem. For the hierarchical-ring and hyper-ring topologies, this type of debug unit yields the greatest benefit when located at the IRIs, because they process more traffic.

Comparing quality-to-cost ratios

We illustrate the topology quality metric Q_{TMD} by comparing the hierarchical-ring and hyper-ring topologies. We synthesized an RTL version of the two NoC interconnects using Synopsys DC Ultra (version X-2005.09) for the TSMC 0.18-micron standard cell operating at 1.8 V. Table 1 gives a sample of the results for two different target frequencies.

For this example, we ignore verification quality (shown in Equation 5 to be equivalent) and use Q_{TMD} values from Equation 7 and Q_{NoC} values from Equation 8; and we assign equal weightings to the TMD and the NoC quality (that is, the multipliers are $\lambda = 0$, $\rho = 1$, and $\sigma = 1$ in Equation 1). With

the synthesis results obtained, we can solve for each topology's Q:R ratios (Equation 4). Table 1 summarizes the results. The higher ratios for the hyper-rings indicate that the quality increase from adding the second global ring and the TMD infrastructure comes with a relatively low cost. Thus, the Q:R scores of the hyper-ring architecture are superior to that of the hierarchical-ring architecture.

If only the Q_{TMD} scores were used, this topology would already be selected for its higher quality by allowing more traffic through the monitors. In many designs where redundancy in the center ring and higher bandwidth offset the slightly higher resource usage, the Q_{NoC} for the hyper-ring will also be higher (the exact factor is application dependent). Therefore, on the basis of those calculations, the hyper-ring architecture would provide better quality than the hierarchical-ring architecture.

This simplified example is only meant to convey the general spirit of our method. The application domain, cost constraints, and weighting coefficients for the various elements driving the overall quality scores must be carefully selected using heuristics or analysis of prototyping results. The optimization problem can then be solved using known algorithms, aiming for the best quality while meeting all constraints.

In comparing the hierarchical-ring and hyper-ring topologies, we found that the increased quality of the hyper-ring architecture came at an acceptable increase in resource requirement. Calculating the quality index accurately requires quantitatively measuring many characteristics of the architecture. This poses a difficult problem because so many variables must be taken into account. Further complicating matters is the sometimes subjective nature of design decisions, such as the selection of topology. In addition to the difficulty in quantifying quality, the addition of resource constraints makes the architecture and TMD component selection a multiobjective optimization problem. Nevertheless, although the quality index calculation remains an open problem that requires further study, we have demonstrated that the proposed QoD flow can help designers integrate TMD hardware while meeting design constraints in a systematic way. ■

Acknowledgments

This work was partially funded by the Natural Sciences and Engineering Research Council.

References

1. P. Patra, "On the Cusp of a Validation Wall," *IEEE Design & Test*, vol. 24, no. 2, Mar.-Apr. 07, pp. 193-196.
2. W. Dally and B. Towles, "Route Packets, Not Wires: On-Chip Interconnection Networks," *Proc. 38th Design Automation Conf.* (DAC 01), ACM Press, 2001, pp. 684-689.
3. L. Benini and G. De Micheli, "Networks on Chips: A New SoC Paradigm," *Computer*, vol. 35, no. 1, Jan. 2002, pp. 70-78.
4. V. Raghunathan, M.B. Srivastava, and R.K. Gupta, "A Survey of Techniques for Energy Efficient On-Chip Communication," *Proc. 40th Design Automation Conf.* (DAC 03), ACM Press, 2003, pp. 900-905.
5. C.A. Zeferino et al., "A Study on Communication Issues for Systems-on-Chip," *Proc. 15th Symp. Integrated Circuits and Systems Design*, IEEE CS Press, 2002, pp. 121-126.
6. T. Bjerregaard and S. Mahadevan, "A Survey of Research and Practices of Network-on-Chip," *ACM Computing Surveys*, vol. 38, no. 1, 2006, article 1.
7. A.M. Amory et al., "Wrapper Design for the Reuse of a Bus, Network-on-Chip, or Other Functional Interconnect as Test Access Mechanism," *IET Computers & Digital Techniques*, vol. 1, no. 3, May 2007, pp. 197-206.
8. M. Boulé, J. Chenard, and Z. Zilic, "Adding Debug Enhancements to Assertion Checkers for Hardware Emulation and Silicon Debug," *Proc. 24th IEEE Int'l Conf. Computer Design* (ICCD 06), IEEE Press, 2006, pp. 294-299.
9. M. Boulé and Z. Zilic, *Generating Hardware Assertion Checkers: For Hardware Verification, Emulation, Post-fabrication Debugging and On-line Monitoring*, Springer, 2008.
10. S. Bourduas, J.-S. Chenard, and Z. Zilic, "A RTL-Level Analysis of a Hierarchical Ring Interconnect for Network-on-Chip Multi-processors," *Proc. Int'l SoC Design Conf.* (ISOCC 06), Inst. of Electronics Engineers of Korea, 2006, pp. 379-382.
11. S. Bourduas and Z. Zilic, "A Hybrid Ring/Mesh Interconnect for Network-on-Chip Using Hierarchical Rings for Global Routing," *Proc. 1st Int'l Symp. Networks-on-Chip* (NOCS 07), IEEE CS Press, 2007, pp. 195-204.
12. F.N. Sibai, "The Hyper-ring Network: A Cost-Efficient Topology for Scalable Multicomputers," *Proc. ACM Symp. Applied Computing* (SAC 98), ACM Press, 1998, pp. 607-612.

Stephan Bourduas has completed a PhD in electrical engineering from the Integrated Microsys-

tems Laboratory in the Department of Electrical and Computer Engineering at McGill University. His research interests include embedded systems, software engineering, high-level modeling, and system-level design methodologies. He has a BEng in computer engineering and an MASc in electrical engineering from Concordia University. He is a graduate student member of the IEEE.

Jean-Samuel Chenard is pursuing a PhD at the Integrated Microsystems Laboratory in the Department of Electrical and Computer Engineering at McGill University. His research interests include embedded wireless systems, flexible radio, reprogrammable computing, and system-level debug methodologies. He has a BEng and an MEng in electrical engineering from McGill University. He is a graduate student member of the IEEE.

Željko Žilić is an associate professor in the Department of Electrical and Computer Engineering at McGill University. His research interests fall in the area of quality-driven design of integrated microsystems, including their synthesis, verification, and test. He has a Dipl Eng in electrical and computer engineering from the University of Zagreb, and an MSc and a PhD in electrical and computer engineering from the University of Toronto. He is a senior member of the IEEE and a member of the ACM.

■ Direct questions and comments about this article to Stephan Bourduas, McGill University, Dept. of Electrical and Computer Engineering, 3480 University St., Montreal, Québec, Canada H3A 2A7.