Modeling Concurrency in NOC for Embedded Systems

Ankur Agarwal, Ravi Shankar
ankur@cse.fau.edu, ravi@cse.fau.edu
Dept of Computer Science and Engineering, Florida Atlantic University
Boca Raton, FL 33431

Abstract

Real Time embedded system designers are facing challenges in the selection and optimization of the system architecture. A typical system comprises of programmable, concurrent, heterogeneous multiprocessors; these are called multiprocessor system-on-chip (MPSoC). The consequence of this MPSoC trend is a shift in concern from computation and sequential algorithms to concurrency, synchronization and communication in every aspect of hardware and software co-design and development. The main problems in the deep sub-micron technologies arise from non scalable wire delays, errors in signal integrity and non-synchronized communication. These problems may be addressed by the use of packet switched Network on Chip (NOC) architecture for MPSoCs used in real-time systems. Such a NOC based system will have to support concurrent processing, in software and hardware. Concurrency issues, if not addressed, may lead the system into a deadlock or a livelock state. System design integration and verification approaches will not be cost-effective in exposing concurrency failures as they are intermittent; this can be costly (significantly increased time to market and field failures). One would have to develop abstract concurrency models and do exhaustive analysis on these models to test for concurrency problems. In this paper we present a systematic approach that models concurrency by using Finite State Process (FSP); this model is exhaustively tested with Labeled Transition State Analyzer (LTSA). We propose a set of rules that should be followed for modeling a concurrent system. We further analyze the proposed approach by modeling concurrency in the NOC backbone.

1. NOC Platform for Embedded Systems

As the technology scaling works better for transistors than for interconnecting wires, there is an increasing disparity between wires and the transistors in terms of power consumption and latency. Thus bus based communication has become a bottleneck. Moreover, in a bus based system, as the number of the processors on the bus is increased, the system performance decreases rapidly. Many innovations, such as, pipelining, split-and-retry techniques, removal of tri-state buffers and multi-phase clocks, have been introduced in bus architectures to counteract this. However, in many cases introduction of new bus architectures has required many changes in bus implementation, and more importantly bus interfaces, thus impacting IP reusability. Another reason that the buses are not scalable is that they cannot decouple the activities of the transaction, transport and physical layers. On a billion transistor chip, it would not be possible to send a global signal across the chip within real-time bounds. Moreover, a bus based SoC or MPSoC does not offer the required amount of reuse in order to meet the time-to-market requirements. This has resulted in declining productivity of system architects and designers. NOC can improve design productivity by supporting modularity and reuse of complex cores, thus enabling a higher level of abstraction in the architectural modeling of future systems.

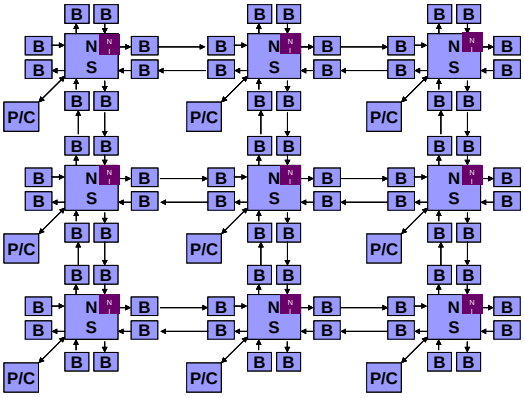

Figure 1. NOC Architecture

Figure 1 shows a 3×3 mesh based NOC architecture. B, P, C, N and S represent buffer, producer, consumer, node and scheduler respectively. Links provide connection to various buffers, node/scheduler, and buffer. The routers are connected to each other and to NI (network interface).

2. Concurrency Modeling on NOC Platform

Use of the NOC architecture implies a shift in concern from computation and sequential algorithms to modeling concurrency, synchronization and communication. In a multiprocessing environment there are various processes executing simultaneously. In NOC, at a given time, every resource may either be producing or consuming some data packets. Thus almost every element (node, link, buffer, router, scheduler, resource etc.) might be communicating with another element, perhaps concurrently, on the NOC platform. Therefore, there are several inter-processes communications in such a model. If these inter-process communications are not modeled properly then the system may fail. Such a failure may not be detected at the system integration phase. These failures are intermittent, which occur only under certain condition, but nevertheless may be catastrophic. A system may be more prone to intermittent failures if concurrency concerns are not addressed properly. These intermittent failures may result in a deadlock or a livelock state. Thus it is very important to model concurrency issues in such NOC systems.

In this paper we propose to model concurrent processes in a NOC. For this we use finite state process (FSP) language and labeled transition system analyzer (LTSA) tool developed by Jeff Magee and Jeff Kramer [2]. We first write a high level specification. This should not be platform

or data dependent. We then identify the concurrent processes in our model. As the concurrency issues arise only due to intra-process interactions, not due to (sequential) internal data processing, we model only the (external) interaction among various processes. This reduces the model complexity significantly; models execute faster due to reduced state exploration.

3. Concurrency Model for NOC

In this section we present our NOC concurrency model for the communication backbone layer.

3.1. Evolve High Level Specifications: These high level specifications should not contain any details. Our abbreviated specifications follow: Data will be received in serialized packet format; there will be several paths available arranged in a matrix fashion for the data packet to travel; data may be buffered at each intersection; further routing is available based on the availability & congestion of links at the destination; packet will contain destination address & Priority; and links may be unidirectional (2 links for each direction) or bi-directional. Further, we simplified our concurrency model by these assumptions: Links are unidirectional and Nodes are arranged in a 2-D mesh.

3.2. Identify Concurrency Processes: We identified concurrent processes from the above specifications. These concurrent processes identified were links, buffers, schedulers, nodes, producers and consumers. We then defined detailed specification for each of these processes: (1) Link Specification: Link collects the data from the source; and link forwards the data to the buffer. (2) Buffer Specification: Store Input Data sent by Link; forward the data to the output: to a node; inform about the buffer status: buffer is empty / full; and Forward the high priority data first. (3) Scheduler Specification: Receive request from the buffer for forwarding the data to the node; forward the request for transmitting the higher priority data first; and check the availability of data path for the data packet. (4) Node Specification: Determine the route information; get the data from buffer; and forwards the data to a buffer (5) Producer/Consumer Specification: For Producer- forward the data packet to the buffer; data packet will be forwarded based on the availability of the buffer; and For consumer - accept the data packet from the buffer

3.3. Develop Model Incrementally and Hide Internal Details: While modeling our concurrent NOC model we followed an incremental approach. We first developed the producer and link processes. We then checked for concurrency issues in these two processes before including other processes in the model. Since the link process is attached to the buffer process, we then added a buffer process to the model. If the buffer is available then producer outputs either a hi-priority data (hiPriDataOut) or a mid-priority data (midPriDataOut). The model does not reveal any details about the source or contents of the data. We have run the simulation with three priority levels and verified the absence of any concurrent violations.

3.4. Suppress Unnecessary Details from the Specification: In order to limit the number of states, it is important to hide unnecessary details from the model. In modeling the NOC, our goal was to model the system shown in Figure 1. A NOC model with 9 nodes and schedulers along with 36 buffers, 9 producers and consumers and more than 100 links would have led to tens of thousands of states. Thus it was not only difficult but almost impossible to simulate such a model with FSP and analyze with LTSA. Therefore, we abstracted this model to represent only one representative scenario of the interaction. If this interaction does not have any concurrency issues then other interactions may be replicated in a similar manner to avoid deadlocks and livelocks.

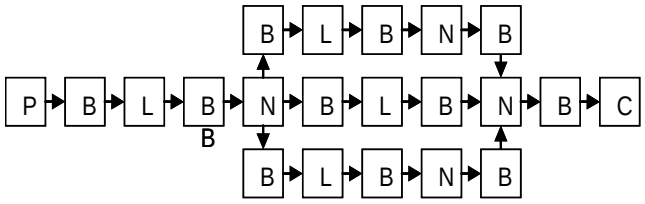

Figure 2. Abstract Model for NOC with a Single Producer/Consumer Interaction

Further, from the implementation of the link process we realized that a link is responsible for just forwarding the data to the next process. In this it does not play any role which may cause any concurrency concerns. Thus we eliminated the link process from the model. We further reduced this model into a more simplified model by removing unnecessary data paths. Instead of showing three data paths (3 buffers connected to a Node) we represented the model with two data paths (2 buffers with 1 Node). This is due to fact that synchronization issues will arise when one or more processes try to interact with a third process. But as long as the number of similar processes is more than one, we can have a general model to represent these interactions. It may be concluded that the number of buffers (as long as it is more one) will not make a difference in addressing concurrency issues. However, it should be noted that a node with one buffer will not have the same implementation as a node with two buffers. Final abstract model used for modeling concurrency is shown in Figure 3.

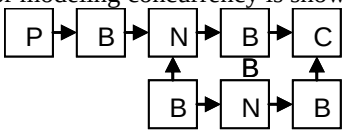

Figure 3. Abstract Model for NOC with Single Data path

Conclusion

We have proposed a methodology for exposing and evaluating concurrency issues at an abstract level for the NOC architecture. This may be adopted for use in other multiprocessor/multi-core architectures. It is our experience that this abstract modeling will reduce unnecessary interactions during the system integration phase, enhancing productivity significantly.

References

[1] Jantsch, A., and Tenhunen, H., Editors, *Networks on Chip*, Springer, 2003

[2] Magee, J., and Kramer, J., *Concurrency Modeling*, Wiley, 1999.