On-chip Communication Architecture for OC-768 Network Processors

Faraydon Karim, Anh Nguyen
STMicroelectronics Inc.
Central R&D
San Diego, CA 92121
{faraydon.karim, anh-q.nguyen}@st.com

Sujit Dey, Ramesh Rao
University of California, San Diego
Department of Electrical Engineering
La Jolla, CA 92093
dey@ece.ucsd.edu, rrao@ucsd.edu

Abstract

The need for network processors capable of forwarding IP packets at OC-192 and higher data rates has been well established. At the same time, there is a growing need for complex tasks, like packet classification and differentiated services, to be performed by network processors. At OC-768 data rate, a network processor has 9 nanoseconds to process a minimum-size IP packet. Such ultra high-speed processing, involving complex memory-intensive tasks, can only be achieved by multi-CPU distributed memory systems, using very high performance on-chip communication architectures. In this paper, we propose a novel communication network architecture for 8-CPU distributed-memory systems that has the potential to deliver the throughput required in next generation routers. We then show that our communication architecture can easily scale to accommodate much greater number of network nodes. Our network architecture yields higher performance than the traditional bus and crossbar yet has low implementation cost. It is quite flexible and can be implemented in either packet or circuit switched mode. We will compare and contrast our proposed architecture with busses and crossbars using metrics such as throughput and physical layout cost.

1. Introduction

The next generation of Internet backbone routers must be designed to deliver ultra high performance over an optical infrastructure. This requirement is due to the growth of Internet traffic and increasing user expectation from service providers. At the current Internet traffic growth rate, it is expected that by 2003, OC-768 routers will be widely deployed. At the same time, as Internet and Application Service Providers attempt to provide better quality, diverse, and differentiated services, it will no longer be enough for routers to perform the two fundamental tasks of routing and packet forwarding.

The routing process collects information about the network topology and creates a forwarding table. The packet-forwarding process copies a packet from an input interface of the router to the proper output interface based on information contained in the forwarding table. In addition, it is expected that routers will be responsible for such tasks as packet classification that can distinguish packets and group them according to their different requirements, buffer management to determine the buffer allocation and admission control of packets, and packet scheduling to meet quality-of-service (QoS) contracts [1].

Traditionally, routers have been implemented using general-purpose RISC processors, or ASICs. While general-purpose processor-based router architectures provide Internet Service Providers the flexibility to upgrade to new router tasks and services, they will not be able to satisfy the growing speed requirements for the new complex packet processing tasks like packet classification at OC-768 data rate. On the other hand, while ASIC based router implementations can provide for speed, they cannot provide for the required flexibility that is becoming important as router tasks and services expand. Hence, the need to develop high-speed network processors, with the dual and conflicting challenges of providing for programmability at OC-768 speed [2].

At OC-768 data rate (40Gbps), the arrival rate of IP packets could reach approximately 114 x 10^6 packets per sec (assuming 44 bytes per packet). To ensure that the worst-case time to process a packet does not violate the packet arrival rate, packet-processing time can be at most 9ns/packet. It is expected that approximately 500 instructions must be performed on each arriving packet to enable packet forwarding and classification on packet flows. Hence, an OC-768 network processor will require 57 GIPS processing power. Obviously, these requirements far exceed the capacity of any central CPU/memory architectures available today and in the expected time frame. Consequently, next-generation network processors will have to be based on multi-processor/distributed memory architectures.

Now let us consider the on-chip communication requirements imposed by typical network processor applications. Consider the task of packet classification using the algorithm described in [3]. Using an estimate of 10K classification rules and 16-bit on-chip memory width, we need to perform 625 memory accesses per packet arrival or 71.3 x 10^9 memory accesses per sec (in the worst-case). A shared bus of a symmetrically distributed network of 8 nodes would need to be running at tens of GHz to support this requirement. Which is clearly not achievable. On the other hand, a crossbar could support this requirement at lower clock-speed but accompanied by a high implementation costs. Therefore, a cost-effective, scalable, and high-performance on-chip communication architecture is needed.

Recent studies have shown the importance of the on-chip communication architecture used in determining the performance of the overall System-on-Chip (SoC) [4][5][6].

In [6], performance of different on-chip communication architectures, with different topologies and protocols, has been analyzed under different classes of on-chip communication traffic. It has been shown that system performance can vary by as much as 2.5 times depending on the communication architecture used. For the same on-chip communication architecture used, performance can vary by up to 6 times depending upon the nature of the communication traffic. These results point to the criticality of choosing the right on-chip communication architecture for a system, depending on the communication traffic expected in the SoC.

Techniques have been developed to design and synthesize on-chip communication to satisfy specific interface and communication needs of components in a system [7][8][9][10][11]. Recently, a reconfigurable on-chip communication architecture has been proposed in [12], which allows adaptation of the communication protocols to the on-chip communication demand. A method has been proposed in [13] to optimally map a system's communication requirements to given template communication architectures. System performance improvements of up to an order of magnitude have been reported by on-chip communication reconfiguration and mapping techniques proposed.

While there have been recent advances in the analysis and design of high-performance, configurable on-chip communication architectures, bus-based topologies and protocols are the most common forms of on-chip communication in current use in commercial SoCs [14][15][16][17][18], including efforts for bus interface standards VSIA [19]. While bus-based on-chip communication may be suitable for many applications, clearly it will not be able to satisfy an OC-768 network processor's very demanding on-chip communication needs, as indicated above.

An on-chip crossbar switch can be an alternative choice to satisfy the on-chip communication needs for an OC-768 network processor. Theoretically, the performance (throughput/delay) of crossbars is high enough to permit the development of efficient network processing tasks [20]. In reality, its implementation cost is high; crossbars require many on-chip wires and relays to minimize clock-skew across the chip. In addition, crossbars do not scale well; as the number of nodes in the network increases, the implementation cost increases as $O(n^2)$.

In this paper, we propose a novel on-chip communication strategy that has the potential to meet the performance requirements of next-generation optical routers. It is simpler to implement than a crossbar yet has much higher throughput than shared busses. Its complexity increases linearly as the number of network nodes. A node can be a stand-alone entity such as processor or memory. Alternatively, it could consist of sub-components such as processors and memory units. We shall call the basic configuration of eight nodes and associated links Octagon.

In section 2, we describe Octagon in more details. Included in the description is the routing protocol used to route packets/requests within the interconnection. We will also provide a rough physical layout comparison between Octagon and crossbar to highlight the higher cost associated with crossbar implementation. In section 3, we compare simulation results of a simple circuit switched version of Octagon with existing analytical performance results for a bus and a crossbar to justify our claim of higher performance. Section 4 deals with network scalability. Section 5 concludes the paper.

2. Description of Octagon Architecture & Routing

The basic unit of Octagon consists of eight nodes and twelve bi-directional links connected as shown in figure 1.

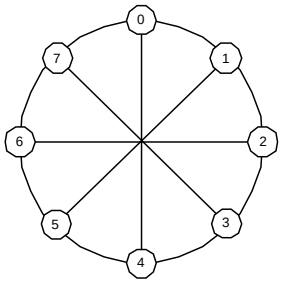


Figure 1. Basic Octagon configuration.

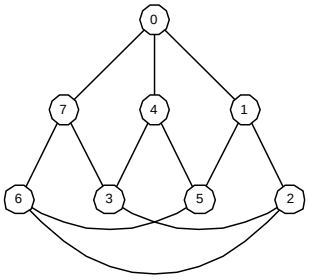


Figure 2. Basic Octagon configuration, alternate view.

The Octagon architecture has the following desirable properties:

a) Communication between any pair of nodes can be performed by at most two hops (to be shown in the Routing section)
b) Higher aggregate throughput than a shared bus or crossbar interconnect
c) Simple shortest-path routing algorithm.
d) Less wiring compared to that of a crossbar interconnect

An alternate view of Octagon is shown in figure 2. This view provides us with a second scaling option to accommodate a larger number of on-chip components. We will elaborate on this point in section 4.

Octagon can be operated in the packet or circuit switched mode. Define an Octagon Data Unit (ODU) to be the actual data field that is transported from node to node within Octagon. An ODU could be fixed or variable length. In the packet switched mode, ODUs are buffered at intermediate nodes if there is contention at the egress link. In the circuit switched mode, the entire path between source and destination nodes is allocated to a communicating node pair for a number of clock cycles. Non-overlapping communication paths are allowed concurrently. In other words, spatial reuse is allowed.

Under this mode, system performance is a function of the chosen connection schedule. The question then is, given the set of pending communication requests, how should the connections be scheduled such that throughput (or some other metrics) is optimized? In section 3, we describe and simulate a simple schedule called the greedy algorithm.

We now consider how routing of ODUs can be implemented under the Octagon architecture.

2.1 Octagon Routing

Octagon node addresses can be coded into a three-bit field. Routing of ODU may be accomplished as follows. A three-bit tag is pre-pended to each ODU. Each node compares the tag (ODU_addr) to its own address (Node_addr) to determine the next action to take. Let the relative address of an ODU be computed according to the following equation.

$$Rel\_addr = ODU\_addr - Node\_addr \ (modulo\ 8). \quad (1)$$

At each node on the Octagon, routing of ODUs is a function of *Rel_addr* as follows.

- *Rel_addr* = 0, process at node
- *Rel_addr* = 1 or 2, route clockwise
- *Rel_addr* = 6 or 7, route counter-clockwise
- Route across otherwise.

Consequently, there is a pre-determined simple routing scheme for each ODU in the network. This enables any two nodes on the network to be separated by at most two hops. We next elaborate on the claim that Octagon has a lower on-chip implementation cost compared to a crossbar.

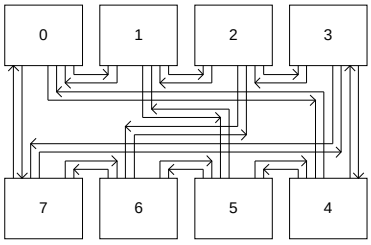


Figure 3. Octagon physical layout schematic.

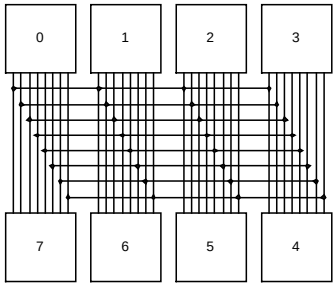


Figure 4. Crossbar physical layout schematic

2.2 Octagon Implementation Cost

Figures 3 and 4 illustrate the physical layout and wiring costs of the Octagon and crossbar interconnect architectures. In our network processor, each Octagon node consists of a processor-memory pair with an estimated size of 2mm x 2mm. Let us consider the minimum wire spacing to be 0.2μm, and the width of a 32-bit link to be 12μm (including individual wire width, spacing and shielding). As shown in figure 3, the physical layout of the Octagon architecture consists of 12 horizontal and 12 vertical 32-bit tracks. Each horizontal track is upper-bounded by 8mm, the total width of the 4 nodes. Each vertical track is upper-bounded by 0.156 μm (13 x 12μm).

As shown in Figure 4, the crossbar needs 8 horizontal and 32 vertical 32-bit tracks. While the horizontal tracks are upper-bounded by 8mm similar to the Octagon, the vertical tracks are upper-bounded by 0.108μm (9 x 12μm).

The above discussion shows that the Octagon has less wiring complexity than a crossbar. In the next section, we show that in addition to the reduced implementation cost, Octagon has significantly better performance than crossbar.

3. Performance Analysis of Bus, Crossbar and Octagon Architectures

In this section, we analyze the performance of the two existing on-chip communication architectures - bus and crossbar, and the new communication architecture proposed – the Octagon. We show that Octagon performance is orders of magnitude better than a bus-based communication, as expected, but also significantly better than that of a crossbar.

Let us denote requests that originate from node *i* and destined to node *j* as type *ij*. Requests of type *ij* arrive to the system following the Poisson process with parameter λ_{ij}. λ_{ij} is considered to be the arrival rate of requests of type *ij* Service time is defined to be the time required by a destination node to complete all requested tasks if the request is processed in isolation. It is exponentially distributed with parameter μ_{ij}. $1/\mu_{ij}$ is the average service time per request. These assumptions can easily be extended to accommodate memoryless discrete-timed distribution such as Bernoulli arrivals and geometric or deterministic service time. We consider a symmetric system where $\lambda_{ij} = \lambda, \forall i,j$ and $\mu_{ij} = \mu, \forall i\ j$. The utilization of requests type *ij* is denoted by $\rho_{ij} = \lambda_{ij}/\mu_{ij} = \lambda/\mu = \rho$. Utilization is the average amount of service demand arriving within one time unit. The aggregated arrival rate is $\lambda_{tot} = \Sigma_{ij}\ \lambda_{ij}$. Total utilization $\rho_{total} = \lambda_{tot}/\mu$.

3.1 Performance of Shared-Bus and Crossbar

The shared bus is modeled as a single server queue with Poisson arrivals and exponential service time. Consider the *aggregated* request arrival process to the eight nodes. Since the individual arrival process is Poisson, the superposition is also Poisson [20]. In memory access applications, the service rate corresponds to the memory access speed. We ignore all propagation delays. Queued requests are served according to the FIFO discipline. The *response time* of an arriving request is defined to be the difference between the time of arrival and time of service completion by bus. Since the server is *work-conserving* [22], the expected response time, denoted by EW_{bus}, for an arbitrary request arriving to the single-server queue is identical to that for a single-server multiple-queue system. The expected response time of a shared bus modeled by a single server queue is [23]

$$EW_{bus} = \frac{\rho_{tot}}{\lambda_{tot}(1-\rho_{tot})}. \quad (2)$$

For crossbar throughput, we use the model as presented in [20]. In that paper, Chen and Stern showed that for a large switch (number of node N ~20) with speedup factor of one, the response time EW_{xbar} could be obtained from

$$EW_{xbar} = EW_s + \frac{\lambda_{tot} E^2 W_s}{2(8-\lambda_{tot} EW_s)}, \quad (3)$$

where $$EW_s = \frac{8\lambda_{tot}}{(8\mu-\lambda_{tot})^2} p_o + \frac{1}{\mu},$$

and $$p_o = 1 - \frac{\rho_{tot}}{8}.$$

In [20], various arbitration policies were studied to investigate their impact on switch performance. It was found that these arbitration policies do not affect the throughput results since maximum throughput is obtained based on the first moment of the service time. However, different arbitration policies do result in different response time.

3.2 Performance Analysis of Octagon

We now investigate the performance of Octagon architecture by simulation, using a simple request-response traffic model. That is, a request is generated at a source node to be sent to a destination node. A connection eventually is established for the communication. For each connection, a request is sent and a response is received. The connection is then severed afterward.

Without loss of generality, we associate with each node a processor and memory module as in figure 5. Applications of this traffic model can be found in routing table lookup, IP packet classification, etc., where requests for memory access is generated at each node. If the memory location requested is attached to the local node then no Octagon communication requests are generated. Otherwise, the request must be forwarded to the appropriate node via Octagon using the routing algorithm as presented in section 2.1. At the destination node, the memory request consumes a number of clock cycles before a response is spawned, to be returned to the originating node.

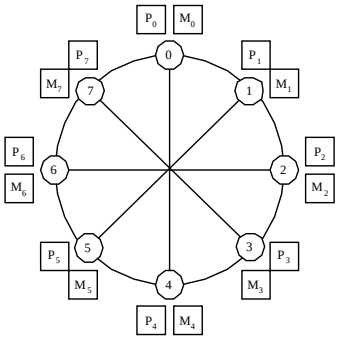


Figure 5. High-level application model.

We consider a connection oriented communication protocol. The connection scheduler follows a greedy algorithm. To elaborate, non-overlapping connections can be accommodated simultaneously. Each node maintains three queues of outstanding requests; one for each egress link. With respect to the overall network, priority is given to the requests at head-of-line in order of their arrival times (lower arrival time implies higher priority). At every service completion time, the scheduler checks to see if new connections can be made based on the previously described priority scheme. Connections are set up until no more can be accommodated without violating the non-overlapping rule. Note that we only consider head-of-line requests at each node. The scheduler is reactivated when a connection is torn down (to see if new connections can be set up).

Figure 6 shows details of the model of each node. In addition to the request generator, processor, and memory, each node has three ingress and three egress ports labeled left (L), across (A), or right (R), consistent with its associated neighbor. Logically, the node emulates a simple 4x4 non-blocking switch (plus processor and memory). The switch has neither input nor output buffering. The central scheduler performs switch arbitration.

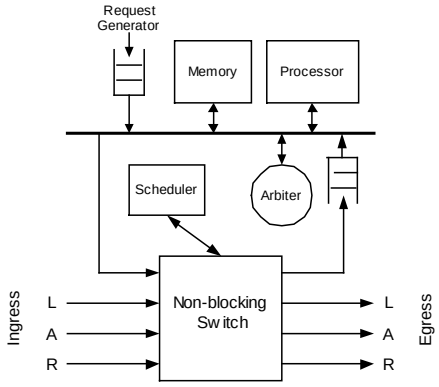


Figure 6. Node model.

3.3 Performance Results

Figure 7 shows the expected response time of the Octagon vs. bus and crossbar. We fix μ = 0.5 per clock cycle and vary λ_{tot}. The horizontal axis unit is $\rho_{tot} = \lambda_{tot}/\mu$ and the vertical axis unit is expected response time in clock. It can be seen that the Octagon has significantly higher maximum throughput than both the bus and crossbar.

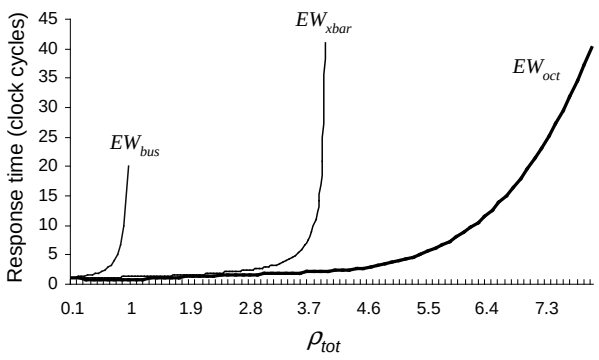


Figure 7. Throughput comparison, Octagon vs. bus and crossbar.

Let us briefly explain the performance results. The bus saturates at ρ_{tot} = 1 because a single server (the bus bandwidth) can provide at most one unit of service per unit of time. The crossbar can be modeled as a system of eight queues sharing eight servers (figure 8). Because of contention and head-of-line blocking, the eight available servers provide approximately four units of work per time unit [20]. Therefore, crossbar saturation occurs at $\rho_{tot} \approx 4$. On the other hand, the Octagon architecture can be modeled as a system of 24 queues and 24 servers (three egress queues and three outgoing links per node, figure 9). It could be argued that analogous to the model for crossbar, the maximum throughput of the Octagon should be approximately 12. However, the difference is this: each request under Octagon may require two servers simultaneously. This reflects the fact that two adjacent links are locked when a connection for a two-hop request is made. Therefore, on average, each request in the latter consumes more resources than one in the former. Consequently, saturation for Octagon occurs at $\rho_{tot} \approx 8$; still significantly higher than that for a crossbar.

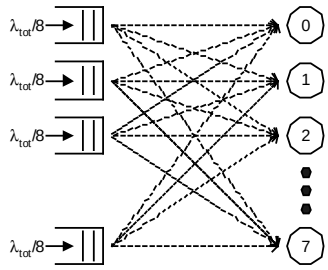


Figure 8. Queueing model of 8x8 crossbar.

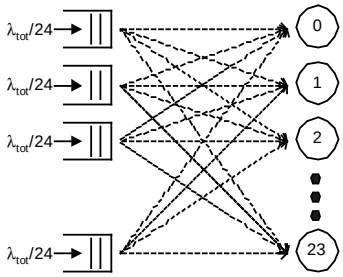
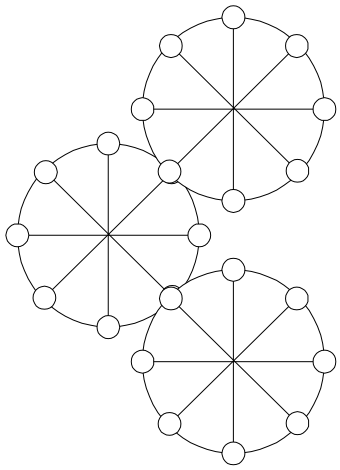


Figure 9. Queueing model of 8-node Octagon.

4. Scalability

Scalability is an important factor because of increasing performance demand from programmable network processors. Next generation network processors are expected to have 16 or more processors, and a large number of distributed memory components holding tables for IP lookup and classification. We believe that the need for interconnecting greater number of on-chip components in network processors and other system-on-chips will accelerate in the foreseeable future, hence the need for scalable on-chip communication architectures.

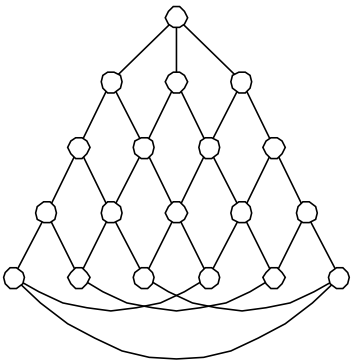
One of the strengths of the Octagon architecture is its ability to scale *linearly*. Figure 10 shows an approach that requires two different node types: *bridge* and *member* nodes. As its name implies, bridge nodes connect adjacent Octagons and perform hierarchical routing of ODUs. As an example, consider a network consisting of eight interconnected Octagons. The Octagon address field of each ODU can be six bits wide; three high-order bits to identify the local Octagon and three low-order bits to identify the node. At each node, static routing can be performed on the entire field while at each member node, routing is performed based only on the low-order bits. Figure 10 also shows the cost as a function of increasing network size for Octagon and crossbar architectures.



	Nodes	Horizontal Links #/max length(mm)	Vertical Links #/max length(mm)
Octagon	8	12/8	12/0.156
	15	24/8	24/0.156
	22 (shown)	**36/8**	**36/0.156**

	Nodes	Horizontal Links #/max length(mm)	Vertical Links #/max length(mm)
Crossbar	8	8/8	32/0.108
	15	15/16	120/0.192
	22	22/11	242/0.276

Figure 10. Scaling strategy 1.



Nodes	Links
8	12
19 (shown)	**31**
34	58

Figure 11. Scaling strategy 2.

The second approach to scalability takes advantage of the natural expansion of the Octagon architecture as shown in figure 2. Under this approach (figure 11), nodes are no longer identified as bridges and members. Instead, they should be called *core* and *edge* nodes. Core nodes are those with four bi-directional links and edge nodes have three. As shown in the table in figure 11, for certain number of nodes that need to be connected, the second scaling strategy offers the most compact architecture in terms of total number of links and physical layout area.

5. Conclusion

The motivation for this work comes from a need for high-performance on-chip communication architectures to help deliver the processing capacity required by OC-768 routers and beyond. Traditional interconnect architectures such as shared bus and crossbars will have difficulties scaling to these data rates while maintaining reasonable costs. In this paper, we proposed a communication architecture which has many desirable properties such as minimal hop counts from one node to any other node on the network, simple static routing algorithm for network traffic, implementation complexity scales linearly with network nodes, and high performance. We compared the throughput of our architecture with those of shared bus and crossbar. Our analysis shows that our network significantly outperforms the shared bus and crossbar.

6. Acknowledgement:

We would like to thank Naresh Soni for his encouragement and support, and Razak Hossain for his help on physical layout and implementation issues.

7. References:

[1] Kumar, V.P.; Lakshman, T.V.; Stiliadis, D., "Beyond best effort: router architectures for the differentiated services of tomorrow's Internet," *IEEE Communications Magazine*, Volume: 36 Issue: 5 , May 1998, pp. 152 –164.

[2] F. Karim, "Network Processors: The New Frontier in SoC Design and Validation," Presentation, *DATE conference*, 2000.

[3] Lakshman, T.V.; Stiliadis, D.; "High-Speed Policy-based Packet Forwarding Using Efficient Multi-dimensional Range Matching," *Proceedings of ACM SIGCOMM*, Sept. 1998, pp. 191-202.

[4] K.Lahiri, A.Raghunathan, S.Dey, "Fast performance analysis of bus-based system-on-chip communication architectures", in *Proc. Intl Conf. on Computer Aided Design*, pp. 566-572, Nov. 1997.

[5] K.Lahiri, A.Raghunathan, S.Dey, "Performance analysis of systems with multi-channel communication architectures", in *Proc. Int. Conf. VLSI Design*, pp. 530-537, Jan. 2000.

[6] K.Lahiri, A.Raghunathan, S.Dey, "Evaluation of the traffic performance characteristics of system-on-chip communication architectures", in *Proc. Int. Conf. VLSI Design*, pp. 29-35, Jan. 2001.

[7] T.Yen and W.Wolf, "Communication synthesis for distributed embedded systems", in *Proc. Intl Conf. on Computer Aided Design*, pp. 288-294, Nov. 1995.

[8] J.Daveau, T.B.Ismail, A.A.Jerraya,"Synthesis of system-level communication by an allocation based approach", in *Proc. Int. Symp. On System Level Synthesis*, pp. 150-155, Sept. 1995.

[9] J.A.Rowson and A.Sangiovanni-Vincentelli, "Interface based design", in *Proc. Design Automation Conf*. pp. 178-183, Jun. 1997.

[10] R.B.Ortega and G.Borriello, "Communication synthesis for distributed embedded systems", in *Proc. Intl Conf. on Computer Aided Design*, pp. 437-444, Nov. 1998.

[11] M.Gasteier and M.Glesner,"Bus-based communication synthesis on system level", *ACM Trans. Design Automation Electronic Systems*, pp. 1-11, Jan. 1999.

[12] K.Lahiri, A.Raghunathan, G.Lakhshminarayana, S.Dey, "Communication Architecture Tuners: a methodology for the design of high performance communication architectures for system-on-chips", in *Proc. Design Automation Conf*. pp. 513-518, Jun. 2000.

[13] K.Lahiri, A.Raghunathan, S.Dey, "Efficient Exploration of the SoC Communication Architecture Design Space", in *Proc. Intl Conf. on Computer Aided Design*, pp. 424-430, Nov. 2000.

[14] "IBM On-chip CoreConnect Bus Architecture", *http://www.chips.ibm.com/products/coreconnect/index.html*

[15] "Peripheral Interconnect Bus Architecture", *http://www.omimo.be*

[16] "Sonics Integration Architecture, Sonics Inc", *http://www.sonicsinc.com*

[17] B. Cordan, "An efficient bus architecture for system-on-a-chip design", *Proc. Custom Integrated Circuits Conf*., pp. 623-626, 1999.

[18] D.Flynn, "AMBA: enabling reusable on-chip designs", *IEEE Micro*, pp. 20-27, vol 7, no. 4, 1997.

[19] "On chip bus attributes specification 1 OCB 1 1.0, On-chip bus DWG", *http://www.vsi.org/library/specs/summary.htm*

[20] Chen, J. and Stern, T., "Throughput Analysis, Optimal Buffer Allocation, and Traffic Imbalance Study of a Generic Nonblocking Packet Switch, *IEEE JSAC*, Vol. 9, No. 3, 1991, pp. 439-449.

[21] Cinlar, E., *Introduction to Stochastic Processes*, pp. 87-88, Prentice Hall, 1975.

[22] Gross, D., and Harris, C., *Fundamental of Queueing Theory*, pp. 297-300, John Wiley & Sons, 3rd Edition, 1998.

[23] Gross, D., and Harris, C., *Fundamental of Queueing Theory*, pp. 61-68, John Wiley & Sons, 3rd Edition, 1998.