

Chapter 2

Preliminaries

Network-on-Chip is the term used to describe an architecture that has maintained readily designable solutions in face of communication-centric trends. In this chapter, we will briefly review some concepts on the design of an NoC router architecture. Various flow-control mechanisms with its corresponding router architecture and design considerations will be included in this chapter.

2.1 Background Knowledge

A typical NoC architecture consists of multiple segments of wires and routers. In a tiled, city-block style of NoC layout, the wires and routers are configured much like street grids of a city, while the client (e.g., logic processor cores) are placed on city blocks separated by wires. A network interface module transforms data packets generated from the client logic (processor cores) into fixed-length flow-control digits (flits). The flits associated with a data packet consist of a header (or head) flit, a tail flit and a number of body flits in between. This array of flits will be routed toward the intended destination in a hop-by-hop manner from one router to its neighboring router.

In a city-block style NoC, each router has five input ports and five output ports corresponding to the north, east, south, and west directions as well as the local processing element. Each port will connect to another port on the neighboring router via a set of physical interconnect wires (channels). The router's function is to route flits entering from each input port to an appropriate output port, and then toward the final destinations. To realize this function, a router is equipped with an input buffer for each input port, a 5×5 cross-bar switch to re-direct traffic to the desired output port, and necessary control logic to ensure correctness of routing results.

Usually, for each data packet, the corresponding head flit specifies its intended destination. After examining the head flit, the router control logic will determine

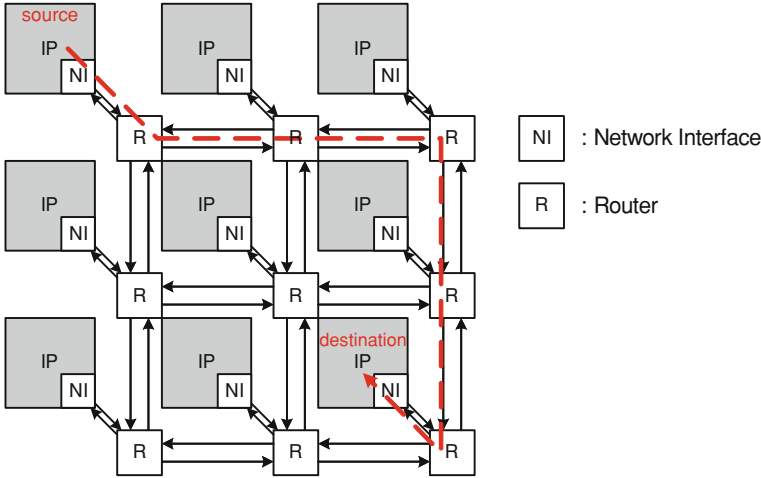


Fig. 2.1 Typical NoC architecture in a mesh topology

which output direction to route all the subsequent (body and tail) flits associated with this data packet according to the routing algorithm applied.

2.2 Conventional Network-on-Chip Architecture

A typical NoC consists of computational processing elements (PEs), network interfaces (NIs), and routers. The latter two comprise the communication architecture.

An NI is used to packetize data before using the router backbone to traverse the NoC. Each PE is attached to an NI that connects the PE to a local router. When a packet was sent from a source PE to a destination PE as shown in Fig. 2.1, the packet is forwarded hop by hop on the network via the decision made by each router. In some NoC architectures that are equipped with error control mechanisms, NIs are also used to encode and decode the data by the error control code applied.

An NoC router is composed of switches, registers, and control logic that collectively perform routing and channel arbitration to guide the flow of packets in the network as illustrated in Fig. 2.2. For each router, the packet is first received and stored in an input buffer. Then the control logic in the router is responsible to make routing decision and channel arbitration. Finally, the granted packet will traverse through a crossbar to the next router, and the process repeats until the packet arrives at its destination.

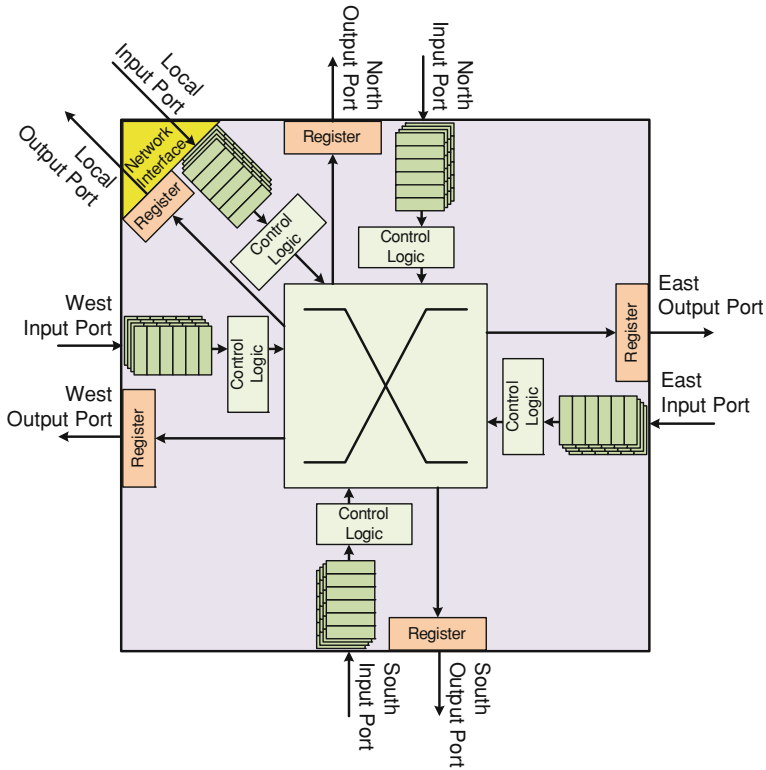


Fig. 2.2 Typical NoC router architecture

2.3 Conventional Router Architecture

Packets delivered by routers are partitioned in a flit-by-flit basis. Each flit of a packet arrives at a router and store in a memory buffer until it can traverse to the next hop of the route. The first flit in the buffer memory will be processed by the control logic to determine whether it is allowed to be forwarded and which output direction it should proceed to. The decision made by the control unit is based on the computation result of routing, arbitration, and the downstream buffer space. After the control setup is done, the flit passes through the crossbar switch to its desired output direction. Since all the decisions are settled by the control logic at the input side, flits will never be stalled at the output ports.

The input buffers in a router design are used to maintain the channel availability until the buffer spaces are exhausted. A larger buffer memory will bring a better throughput and lower latency but however result in large area overhead and power consumption. Usually, a buffer is implemented as a first-in-first-out (FIFO) queue, where data flits can be processed according to the order of their arrival time. Other mechanism such as the dynamic buffer allocation scheme proposed in [1], which

trades off complexity with performance, can achieve better buffer space utilization by using a linked-list memory structure.

The two major components in a control logic are the routing and arbitration modules. The routing module processes flits to generate the direction requests at each input direction, while the arbitration module at each output direction receives these requests and produces a grant signal to point out the winning request that can pass through the corresponding output direction. Algorithmic routing with simple logic is popular in modern on-chip router design. Note that the number of direction requests is based on the number of existing output directions in the router. In a 2-D mesh topology, there are five ports connected to the outsides, including the four neighbor routers and one local processing element. The major design concern of an arbiter is its fairness property, which means that different requestors should be provided with a reasonable amount of service from the arbiter according to their individual requirement. The logic complexity of an arbiter grows with the number of input requests. Circuit area and computation delay are major concerns in the arbiter design for an on-chip router.

The crossbar uses a multiplexer for each router output. The control input of a multiplexer is generated by the corresponding arbiter. Each output port can select at most one flit to send out in each clock cycle according to the respective arbitration result. The area and critical path delay of a crossbar, which are affected by the total number of data ports and their bus width, can occupy a significant amount of design concern in a router.

Packet transmissions in wormhole routing are segmented into three flit types: header, body, and tail flits. Head flits are responsible for initiating and reserving the channel bandwidth at each router node for their followers. Body and tail flits are guided by the routes that the header flit has created, and the reserved channel bandwidth will be released by the tail flit to finish the transit of a packet at the current node.

2.4 Flow-Control Mechanism

The performance of NoC communication architecture is dictated by its *flow-control* mechanism. Adding buffers to networks significantly improves the efficiency of a flow-control mechanism since a buffer can decouple the allocation of adjacent channels. Without a buffer, the two channels must be allocated to a packet (or flits) during consecutive cycles, or the packet must be dropped or misrouted [2]. More specifically, with *buffered flow-control*, when a packet arrives at a router, it must first occupy some resources, such as channel bandwidth and buffer capacity, depending on the flow-control methodology. Each router must juggle among multiple input data streams from multiple input ports and route them to appropriate output ports with the highest efficiency.

Buffered flow-control methods can be classified into *packet-buffer flow-control* and *flit-buffer flow-control* based on their granularity of buffer allocation and

channel bandwidth allocation [2]. Since allocating resources in unit of flit can achieve more storage utilization efficiency than in unit of packet. Two types of *flit-buffer flow-control* architectures are commonly used in NoC: the *wormhole* flow-control and the *virtual-channel* flow-control.

2.4.1 Packet-Buffer Flow-Control

Packet-buffer flow-control allocates network resources in a packet-by-packet basis. Examples are *store-and-forward flow-control* and *virtual-cut-through flow-control*.

In store-and-forward method, each node must ensure that it has already received and stored an entire packet before forwarding it to the downstream node. While the virtual-cut-through scheme can forward a packet as long as there is enough buffer space to receive a packet at the downstream node. As a result, virtual-cut-through introduces lower communication delay than store-and-forward does. However, packet-buffer flow-control needs larger size of buffer space in one node because of its inefficient use of buffer storage. In addition, allocating channels in units of packets will increase contention latency.

2.4.2 Wormhole Flow-Control Based Router

Wormhole flow-control improves performance through a finer granularity of message allocation at flit-level instead of packet-level. This technique allows more efficient use of buffer than the packet-buffer flow-control mechanism since the buffer size in each router can be reduced significantly [3, 4].

A typical three-stage pipelined NoC router architecture based on wormhole flow-control is shown in Fig. 2.3. Every input port has a FIFO-based input buffer, which can be seen as a single virtual-channel used to hold blocked flits. To facilitate wormhole flow-control based routing [2], the *routing computation* (RC) module will send a channel request signal to the *switch allocator* (SA) for data in each input buffer. If the downstream buffer at a neighboring router has vacant space, SA will allocate the channel and route the data flits through the crossbar switch toward the designated downstream router at the *switch traversal* (ST) stage.

However, wormhole flow-control based switching technique saves buffer size at the expense of throughput since the channel is owned by a packet, but buffers are allocated on a flit-by-flit basis. As such, an idle packet may continue block a channel even when another packet is ready to use the same channel, leading to inefficient resource utilization. This is the well-known *head of line* (HoL) blocking problem. Therefore, virtual-channel flow-control based router architecture was proposed to reduce blocking effect and to improve network latency.

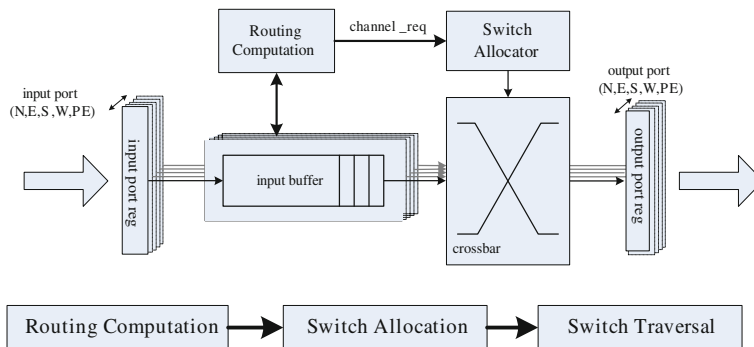


Fig. 2.3 Typical router design based on wormhole flow-control

2.4.3 Virtual-Channel Flow-Control Based Router

Virtual-channel flow-control assigns multiple virtual-paths, each with its own associated buffer queue, to the same physical channel; thus it increases throughput by up to 40% over wormhole flow-control and helps to avoid possible deadlock problems [5–7].

A virtual-channel flow-control router architecture as shown in Fig. 2.4 can be seen as a remedy to the shortcoming of the wormhole flow-control scheme. By multiplexing multiple virtual-channels into the same input buffer, an idle packet will no longer block other packets that are ready to be routed using the shared physical channel. In a typical virtual-channel flow-control based router, the flits are routed via a four-stage pipeline: *routing computation*, *virtual-channel allocation*, *switch allocator*, and *switch traversal*.

One incoming flit that arrives at a router is first written to an appropriate input virtual-channel queue and waits to be processed. When a head flit reaches the top of its virtual-channel buffer queue and enters the RC stage, it is decoded by the RC module and generates an associated direction request. The direction request of this flit is then sent to the VA module to attain virtual-channel at the downstream router. There might be some contentions among packets that request for the same virtual-channel at the downstream router. The loser packets will be stalled at the VA stage and the following flit in the previous stage will also be blocked due to this contention failure. Note that the processes of RC and VA actually take place only on the head flit. The subsequent body flits and tail flit of a packet simply accede to the routing decision acquired by the head flit and require no further processing at the RC and VA stages. Once a decision on the output virtual-channel selection is made at the VA stage, the SA module will assign physical channels to intra-router flits. Flits granted with a physical channel will traverse through the crossbar switch to the input buffer of the down-stream router during the ST stage, and the process repeats until the packet arrives at its destination.

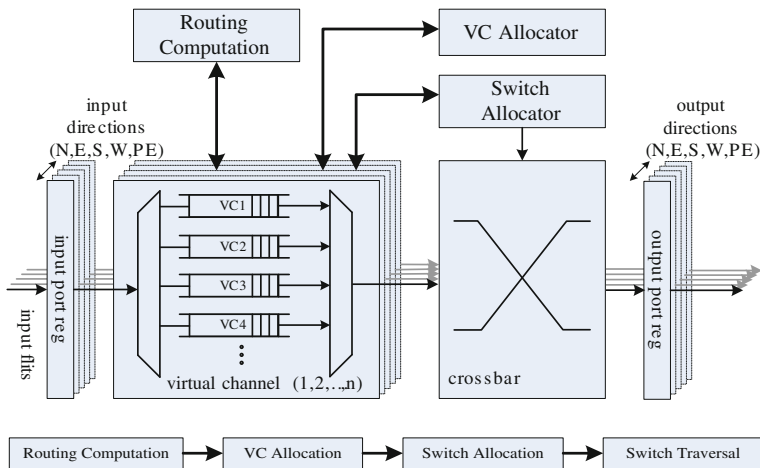


Fig. 2.4 Typical router design based on virtual-channel flow-control

The same as a wormhole flow-control based router, a typical virtual-channel flow-control based router, used by a 2-D mesh-type NoC architecture, contains ten hardwired, physical, and unidirectional data communication channels. Two unidirectional data channels are connected, each in an opposite direction as an input and an output channels, to the neighboring routers. The crossbar, which is a 5×5 switch fabric used to connect an input channel to an output channel, in these two typical router architectures can support up to ten unidirectional channels for data transmission. However, the channel bandwidth utilization is not flexible in this hardwired 5-input 5-output NoC router.

For example, while a flit f_A in the west input buffers/virtual-channels is requesting for the north output channel which is being used by another flit f_B from the east input buffer/virtual-channels. f_A has to wait for the north output channel but has no chance to use the other idle channel at the north direction because it is hardwired for input channel, which can only receive data from the neighboring router. However, if all ten unidirectional channels are replaced by bidirectional channels, the channel utilization will be more efficient. Back to the previous example, while the neighboring router is not using the other channel in north direction, the local router can dynamically self-reconfigure this channel as a second output channel and the contention at the north direction can be relieved.

2.5 Routing and Arbitration Techniques

A general problem pertaining to the routing and arbitration algorithms can be stated as: given an application graph which can be represented by a unique traffic pattern, and a communication architecture, find a decision function at each router for selecting an output port that achieves a user-defined objective function.

2.5.1 Problem Decomposition

The above problem has three main parts: a traffic pattern, an NoC communication architecture, and an algorithm which best satisfies a set of user-defined objectives. First, the traffic patterns known ahead of time can be dealt with by a scheduling algorithm. On the other hand, dynamic or stochastic traffic patterns rely on the use of a routing algorithm with a varying degree of adaptation to route packets. Our focus will be on the patterns not known ahead of time.

Second, NoC communication architectures can have different topologies. The most common one is a regular 2-D mesh, frequently used to display the behavior of adaptive routing algorithms. Other work, such as [8], deal with irregular regions in meshes. Our focus is independent of topology.

The third part deals with the algorithms themselves and the objectives to achieve. Two primary algorithms used to determine where and when a packet will move are: routing and arbitration. A routing algorithm decides which direction each input packet should travel. Arbitration is the process of deciding which input packet request should be granted when there are more than one input packet requests for the same output port.

2.5.2 State-of-the-Art

A typical router in an NoC is responsible for moving the received packets from the input buffers, with its routing and arbitration algorithms, to the output ports. The decisions which a router makes are based on the information collected from the network. Centralized decisions refer to making decisions based on the information gathered from the entire network [9]. Distributed decisions refer to making decisions based only on the information generated by the local router or nearby routers. Distributed routing, the focus of this book, allows NoCs to grow in size without worrying about the increasing order of complexity within a centralized routing unit. An example of centralized routing is the AntNet algorithm [10], which depends on global information to make routing decisions, thus needs extra ant buffers, routing tables, and arbitration mechanisms at each node.

There are some distributed routing algorithms which only rely on local information. They have been proposed as being efficient and still maintaining low overhead and high scalability. Routing algorithms in this category include deterministic and adaptive algorithms. Under realistic traffic patterns which pose the problem of hotspot traffic congestion areas, XY deterministic routing failed to avoid hotspots and resulted in high average latencies [11]. Adaptive routing guides the router to react to hotspots created by different traffic patterns, by allowing a packet at the input buffer to request more than one output port or direction [12]. While minimal routing algorithms prevent livelock from occurring, adaptive

routing introduces the possibility of deadlock, which can be prevented by applying odd–even turn model restrictions to the routing decision [13].

As presented in [11], the DyAD router dynamically switches from deterministic to adaptive routing when congestion is detected, since deterministic routing achieves low packet latency under low packet injection rates. Neighboring nodes send indication to use adaptive routing when their buffers are filled above a preset threshold. Under these conditions, the router dictates that packets are routed in the direction with more available input-buffer slots. This minimal adaptive algorithm, used in the presence of hotspots and increasing congestion rates, pushes back the saturation point of the traffic in the network. Another extension of adaptive routing is the Neighbors-on-Path (NoP) algorithm [14], which allows each router to monitor two hops away the input buffers of the routers in order to detect potential congestion earlier. By earlier detection of the buffer fill level, routes can avoid congestion better. *DyXY* is an algorithm which utilizes a history of buffer fill-levels to make decisions [15]. The algorithms presented in [16, 17] utilize variants of buffer fill-level to make decisions.

In addition to making a routing decision based on the buffer information of downstream packets, the other part of a router’s decision making is the arbitration of packets. When multiple input packets are designated to be forwarded to the same next hop destination, arbitration algorithms such as round-robin or first-come first-serve (FCFS) have been proposed to resolve the output port contention. These arbitration algorithms could be designed to relieve upstream buffers with higher congestion. Contention-Aware Input Selection (CAIS) algorithm [18] is an improved arbitration algorithm that contributes to reduce the routing congestion situation by relieving hotspots of upstream traffic, determined by requests from the upstream traffic.

More works have been proposed to deal with some variance of the routing or arbitration algorithms. Sometimes, we categorize the former ones as methods of congestion avoidance; in other words, they evaluate downstream network conditions to avoid sending packets towards the congested areas so as not to aggravate the congestion conditions. We categorize the latter as methods of congestion relief; in other words, they evaluate upstream network conditions to determine which area had the most congestion to send first in order to quickly diffuse the congested situation.

2.6 Quality-of-Service Control

There is a wide range of possibilities for implementing guaranteed services on a network. Referring to the state-of-the-art QoS mechanisms for NoCs, they can be categorized into two types of schemes: connection-oriented (circuit-switching) and connection-less (packet-switching).

2.6.1 Connection-Oriented Scheme

In connection-oriented schemes, guaranteed service packets traverse on some particular channels or buffers that were reserved for them. Specifically, the connection path between the source and destination pair of GS packets is built at the time before they are injected onto the network [19–26]. However, this kind of static pre-allocation may result in high service latency and does not consider hotspots created by temporal shifts in data requirements, thus leads to a rather unscalable NoC.

Connection-oriented QoS mechanism is reliable to achieve QoS requirement, since connections are created guaranteeing tight bounds for specific flows. Two types of the programming models for constructing the setup phase were presented: centralized programming and distributed programming. Centralized programming sets up the reservations by a configuration manager which takes over all the resources in the network. On the contrary, distributed program models let all the resource reservations to be handled by each local router. The centralized method is simpler to achieve while it is only suitable for small-size systems. Despite the hardware overhead in routers, a distributed program model has acquired popularity in a large system because of its better flexibility.

However connection-oriented QoS mechanism comes with greater hardware overhead in control and storage for resource reservations and poor scalability because complexity grows with each node added. Furthermore, bandwidth usage is inefficient and resource allocation has to be considered on a worst case basis. Moreover, the setup phase of guaranteed traffic presents a timing overhead which may result in inefficiency for non-deterministic applications.

2.6.2 Connection-Less Scheme

The connection-less scheme is an alternative way to support different service levels in NoCs where the resource authorities are prioritized according to the QoS requirement of a traffic flow [23]. This is a distributed technique which allows traffic to be classified into different service levels. These service levels can often coincide with different virtual-channels inside the switch. As two traffic flows with different QoS requirements are presented on the same channel simultaneously, the higher prioritized flow can interrupt the lower one and traverse this channel antecedently [23, 27]. It is more adaptive to network traffic and potential hotspots and can better utilize the network.

Different from the connection-oriented schemes, connection-less schemes do not execute any resource reservation. In contrast, multiple traffic flows share the same priority or the same resource thus could cause unpredictable conditions [28]. The traffic with higher service level is guaranteed in a relative fashion in a connection-less scheme by prioritizing each type of traffic flow. However, while the

connection-less scheme provides a coarser QoS support as the connection-oriented schemes, they can offer a better adaptation of communication to the varying network traffic. Furthermore, better bandwidth utilization and less hardware cost can be achieved since the traffic is allocated with network resources dynamically. With the consideration of performance requirements for each service level, a network designer can select an appropriate bandwidth implemented in an NoC to both meet the QoS constraints and save the wiring cost [23, 29, 30].

Although connection-oriented communication guarantees tight bounds for several traffic parameters, an erroneous decision of resource reservation might cause an unexpected performance penalty. While in a connection-less network, a non-optimal priority assignment has less degradation of throughput though it provides coarse QoS support. As pointed out in [31], guaranteed services require resource reservation for the worst-case in a connection-oriented, which causes a lot of wasted resource. In addition, some quantitative modeling and comparison of these two schemes, provided in [32], has shown that under a variable-bit-rate application, connection-less technique provides a better performance in terms of the end-to-end packet delay. These comparisons can help to design an application-specific NoC using a suitable QoS scheme.

2.7 Reliability Design

The trend towards constructing large computing systems incorporated with a many-core architecture has resulted in a two-sided relationship involving reliability and fault tolerance consideration. While yield has always been a critical issue in recent high performance circuitry implementation, the document of International Technology Roadmap for Semiconductor (ITRS) [33] states that “Relaxing the requirement of 100% correctness for devices and interconnects may dramatically reduce costs of manufacturing, verification and test.” The general principle of fault tolerance for any system can be divided in two categories:

1. Employment of hardware redundancy to hide the effect of faults.
2. Self-identification of source of failure and compensate the effect by appropriate mechanism.

If we can make such a strategy work, a system will be capable of testing and reconfiguring itself, allowing it to work reliably throughout its lifetime.

2.7.1 Failure Types in NoC

Scaling chips however increase the probability of faults. Faults to be considered in an NoC architecture can be categorized into permanent (hard-fault) and transient

fault (soft-fault) [34, 35]. The former one reflects irreversible physical changes, such as electro-migration of conductor, broken wires, dielectric breakdowns, etc. In this case, permanent damages in a circuit cannot be repaired after manufacture. Therefore, the module which is suffering a permanent fault should turn off its function and inform neighboring modules of this information. Then, re-routing packets with an alternative path will be re-calculated deterministically or dynamically according to the need. However, this may induce non-minimal path routing and increase the complexity of routing decision. Hardware redundancy such as spare wire or reconfigurable circuitry can also be used to avoid using of faulty modules [36–39]. In the latter case, several phenomena, such as neutron and alpha particles, supply voltage swing, and interconnect noise, induce the packet invalid or misrouted. Usually, a transient fault is modeled with a probability of bit error rate under an adequate fault model. In an NoC system, intra-router or inter-router functionality errors may happen, to understand how to deal with the most common sources of failures in an NoC, Park et al. provided comprehensive fault tolerant solutions relevant to all stages of decision making in an NoC router [40].

2.7.2 Reliability Design in NoC

A number of fault tolerant methods were proposed in [41, 42] for large-scale communication systems. Unfortunately, these algorithms are not suitable for an NoC, because they will induce significant area and resource overhead. Dumitras et al. proposed a flood-based routing algorithm for NoC, named *stochastic communication*, which is derived from the fault tolerance mechanism used in the computer network and distributed database fields. Such stochastic-communication algorithm separates computation from communication and provides fault tolerance to on-chip failures [33, 43]. However, to eliminate the high communication overhead of flood-based fault tolerance algorithm, Pirretti et al. promoted a redundant random-walk algorithm which can significantly reduce the overhead while maintaining a useful level of fault tolerance [44]. However, the basic idea of sending redundant information via multipath to achieve fault tolerance may cause much higher traffic load in the network, and the probabilistic broadcast characteristic may also result in additional unpredictable behavior on network loading.

Therefore, in a distributed NoC router considering practical hardware implementation, the error control scheme used to detect/correct inter-router transient fault in an NoC is required to have smaller area and shorter timing delay. An error control code that adapts to different degrees of detection and correction and has a low timing overhead will ease its integration into a router. The fault-tolerant method utilizing error detection requires an additional retransmission-buffer specially designed for NoCs when the errors are detected. Error control schemes, such as the Reed-Solomon code proposed by Hoffman et al. have been used on NoCs [45]. But as their results show, the long delay would degrade the overall timing and performance of an NoC Router.

2.8 Energy-Aware Task Scheduling

The availability of many cores on the same chip promises a high level of parallelism to expedite the execution of computation-intensive applications. To do so, a program must first be represented by a *task graph* where each node is a coarse-grained task (e.g., a procedure or a sub-routine). Often, a task needs to forward its intermediate results to another task for further processing. This inter-task data-dependency is represented by a directed arc from the origin task to the destination task in the task graph. Tasks that have no inter-task data dependency among themselves can be assigned to multiple processor cores to execute concurrently. As such, the total execution time can be significantly shortened.

A *real-time application* is an application which execution time must be smaller than a *deadline*. Otherwise, the computation will be deemed a failure. To implement an application on an MC-NoC platform for parallel execution, each task in the task graph will be *assigned* to a processor core. Depending on the city-block distance between two tiles, inter-task communication will take different amount of communication delay. For a particular application, proper task assignment will reduce communication delay while maximizing parallelism such that the total execution time can be minimized. For a real-time application, if the total execution time is less than the pre-defined deadline of the application, the *slacks* between them could be exploited to reduce energy consumption.

The execution time of a task may vary depending on the clock frequency the processor core is running. One technique to adjust the clock frequency of individual time on an MC-NoC is *Dynamic Voltage Scaling* (DVS). When the clock frequency slows down, often the associated energy consumed by a running task is also reduced. Hence, in addition to assigning tasks to the processor cores located at appropriate tiles, another design objective would be to use DVS to save some energy while conforming to the deadline constraint, with perhaps smaller slacks.

Previously, it has been shown that the minimum energy multi-processor task scheduling problem is NP-hard [46–48]. For real-time applications, it was proposed that execution of some tasks can be slowed down using DVS on corresponding tiles without violating the deadline timing constraint [49]. Several DVS-enabled uni-processors have been implemented. Test results running real-world applications showed significant power saving up to 10 times [50]. For multi-processor core systems implemented to execute a set of real-time dependent tasks, Schmitz et al. [51–53] presented an iterative synthesis approach for DVS-enabled processing element based on genetic algorithms (GA). They proposed a heuristic PV-DVS algorithm specifically for solving the voltage scaling. Kianzad et al. improved the previous work by combining assignment, scheduling, and power management in a single GA algorithm [54]. However, GA-based design optimization suffers slow convergence and lower desired quality. Chang et al. [55] proposed using Ant Colony Optimization (ACO) algorithm. Common to these approaches is that when PV-DVS is applied for power reduction, it is applied to one task (tile) at a time and is done after assignment and scheduling. Zhang et al.

[56] and Varatkar et al. [57] proposed using a list scheduling algorithm to find an initial task schedule, and the DVS problem was solved by integer linear programming. The idea behind these methods is to maximize the available slack in a schedule so as to enlarge the solution space of using DVS. However, the communication infrastructures used in these works are either a point-to-point interconnect, or a bus architecture. Hu et al. [58] proposed an energy-aware scheduling (EAS) algorithm that considers the communication delay on a NoC architecture. However, DVS frequency adjustment was not considered.

References

1. Y. Tamir and G. L. Frazier, "Dynamically-Allocated Multi-Queue Buffers for VLSI Communication Switches," *IEEE Transactions on Computers*, vol. 41, no. 6, pp. 725–737, June 1992
2. W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*, Morgan Kaufmann, 2004
3. W. J. Dally and C. L. Seitz, "The Torus Routing Chip," *Journal of Distributed Computing*, vol. 1, no. 4, pp. 187–196, January 1986
4. P. Kermani and L. Kleinrock, "Virtual Cut-Through: A New Computer Communication Switching Technique," *Computer Networks*, vol. 3, no. 4, pp. 267–286, September 1979
5. W. J. Dally, "Virtual Channel Flow Control," *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, no. 2, pp. 194–205, March 1992
6. L. S. Peh and W. J. Dally, "A Delay Model for Router Microarchitectures," *IEEE Micro*, vol. 21, no. 1, pp.26–34, January 2001
7. W. J. Dally and C. L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Transactions on Computers*, vol. C-36, no. 5, pp. 547–553, May 1987
8. E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "Routing Table Minimization for Irregular Mesh NoCs," in *Proceedings of the Design Automation and Test in Europe Conference*, pp. 1–6, April 2007
9. M. A. Yazdi, M. Modarressi, and H. S. Azad, "A Load-Balanced Routing Scheme for NoC-Based System-on-Chip," in *Proceedings of the Workshop on Hardware and Software Implementation and Control of Distributed MEMS*, pp. 72–77, June 2010
10. M. Daneshlatab, A. A. Kusha, A. Sobhani, Z. Navabi, M. D. Mottaghi, and O. Fatemi, "Ant Colony Based Routing Architecture for Minimizing Hot Spots in NOCs," in *Proceedings of the Annual Symposium on Integrated Circuits and System Design*, pp. 56–61, September 2006
11. J. Hu and R. Marculescu, "DyAD–Smart Routing for Networks-on-Chip," in *Proceedings of the Design Automation Conference*, pp. 260–263, June 2004
12. C. J. Glass and L.M. Ni, "The Turn Model for Adaptive Routing," *Journal of ACM*, vol. 41, no. 5, pp. 874–902, September 1994
13. G. M. Chiu, "The Odd-Even Turn Model for Adaptive Routing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 7, pp. 729–738, July 2000
14. G. Ascia, V. Catania, M. Palesi, and D. Patti, "Neighbors On-Path: A New Selection Strategy for On-Chip Networks," in *Proceedings of the IEEE Workshop on Embedded Systems for Real Time Multimedia*, pp. 79–84, October 2006
15. M. Li, Q.A. Zeng, and W. B. Jone, "DyXY - a Proximity Congestion-Aware Deadlock-Free Dynamic Routing Method for Network on Chip," in *Proceedings of the Design Automation Conference*, pp.849–852, July 2006

16. E. Nilsson, M. Millberg, J. Oberg, and A. Jantsch, "Load Distribution with the Proximity Congestion Awareness in a Network-on-Chip," in *Proceedings of the Design Automation and Test in Europe Conference*, pp.1126–1127, December 2003
17. J. Kim, D. Park, T. Theocharides, N. Vijaykrishnan, and C. R. Das, "A Low Latency Router Supporting Adaptivity for on-Chip Interconnects," in *Proceedings of the Design Automation Conference*, pp. 559–564, June 2005
18. D. Wu, B. M. Al-Hashimi, and M. T. Schmitz, "Improving Routing Efficiency for Network-on-Chip through Contention-Aware Input Selection," in *Proceedings of the Asia and South Pacific Design Automation Conference*, pp. 36–41, January 2006
19. M. Millberg, E. Nilsson, R. Thid and A. Jantsch, "Guaranteed Bandwidth using Looped Containers in Temporally Disjoint Networks within the Nostrum Network on Chip," in *Proceedings of the Design Automation and Test in Europe Conference*, pp. 890–895, February 2004
20. K. Goossens, J. Dielissen, and A. Radulescu, "The Æthereal Network on Chip: Concepts, Architectures, and Implementations," *IEEE Design & Test of Computers*, vol. 22, no. 5, pp. 414–421, October 2005
21. P. Vellanki, N. Banerjee, and K. S. Chatha, "Quality-of-Service and Error Control Techniques for Mesh-Based Network-on-Chip Architectures," *ACM Very Large Scale Integration Journal*, vol. 38, no. 3, pp. 353–382, January 2005
22. N. Kavaldjiev, G. J. M. Smit, P. G. Jansen, and P. T. Wolkotte, "A Virtual-channel Network-on-Chip for GT and BE Traffic," in *Proceedings of the Annual Symposium on Emerging VLSI Technologies and Architectures*, pp. 211–216, March 2006
23. E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "QNoC: QoS Architecture and Design Process for Network-on-Chip," *Elsevier Journal of System Architecture*, vol. 50, no.2–3, pp. 105–128, February 2004
24. M. Dall'osso, G. Biccari, L. Giovannini, D. Bertozzi, and L. Benini, "Xpipes: a Latency Insensitive Parameterized Network-on-Chip Architecture for Multiprocessor SoCs," in *Proceedings of the International Conference on Computer Design*, pp. 536–539, October 2003
25. D. Bertozzi and L. Benini, "Xpipes: a Network-on-Chip Architecture for Gigascale System-on-Chip," *IEEE Circuits and Systems Magazine*, vol. 4, no. 2, pp. 18–31, April 2004
26. T. Bjerregaard and J. Sparso, "A Router Architecture for Connection-Oriented Service Guarantees in the MANGO Clockless Network-on-Chip," in *Proceedings of the Design Automation and Test in Europe Conference*, pp. 1226–1231, March 2005
27. M. D. Harmanci, N. P. Escudero, Y. Leblebici, and P. Ienne, "Providing QoS to Connection-Less Packet-Switched NoC by Implementing DiffServ Functionalities," in *Proceedings of the International Symposium on System-on-Chip*, pp. 37–40, November 2004
28. A. Mello, L. Tedesco, N. Calazans, and F. Moraes, "Evaluation of Current QoS Mechanisms in Networks-on-Chip," in *Proceedings of the International Symposium on System-on-Chip*, pp. 1–4, November 2006
29. Z. Guz, E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "Efficient Link Capacity and QoS Design for Network-on-Chip," in *Proceedings of the Design Automation and Test in Europe Conference*, pp. 1–6, March 2006
30. P. Vellanki, N. Banerjee, and K. S. Chatha, "Quality-of-Service and Error Control Techniques for Network-on-Chip Architecture," in *Proceedings of the Great Lakes Symposium on VLSI*, pp. 45–50, April 2004
31. E. Rijpkema, K. G. W. Goossens, A. Radulescu, J. Dielissen, J. V. Meerbergen, P. Wielage, and E. Waterlander, "Trade-offs in the Design of a Router with Both Guaranteed and Best-Effort Services for Networks-on-Chip," in *Proceedings of the Design Automation and Test in Europe Conference*, pp. 350–355, March 2003
32. M. D. Harmanci, N. P. Escudero, Y. Leblebici, and P. Ienne, "Quantitative Modeling and Comparison of Communication Schemes to Guarantee Quality-of-Service in Networks-on-Chip," in *Proceedings of the International Symposium on Circuits and Systems*, pp. 1782–1785, May 2005

33. P. Bogdan, T. Dumitras, and R. Marculescu, "Stochastic Communication: A New Paradigm for Fault Tolerant Networks on Chip," *VLSI Design*, vol. 2007, Article ID 95348, pp. 1–17, 2007
34. R. Marculescu, U. Y. Ogras, L. S. Peh, N. E. Jerger, and Y. Hoskote, "Outstanding Research Problems in NoC Design: System, Microarchitecture, and Circuit Perspectives," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 1, pp. 3–21, January 2009
35. M. Ali, M. Welzl, and S. Hessler, "A Fault Tolerant Mechanism for Handling Permanent and Transient Failures in Network on Chip," in *Proceeding of the International Conference on Information Technology*, pp. 1027–1032, April 2007
36. M. Yang, T. Li, Y. Jiang, and Y. Yang, "Fault-Tolerant Routing Schemes in RDT(2,2,1)/ α -Based Interconnection Network for Network-on-Chip Designs," in *Proceedings of the International Symposium on Pervasive Systems, Algorithms and Networks*, pp. 1–6, December 2005
37. T. Lehtonen, P. Liljeberg, and J. Plosila, "Online Reconfigurable Self-Timed Links for Fault Tolerant NoC," *VLSI Design*, vol. 2007, Article ID 94676, pp. 1–13, 2007
38. K. Kariniemi and J. Nurmi, "Fault Tolerant XGFT Network on Chip for Multi-Processor System on Chip Circuit," in *Proceedings of the International Conference on Field Programmable Logic and Applications*, pp. 203–210, August 2005
39. T. Schonwald, J. Zimmermann, O. Bringmann, and W. Rosentiel, "Fully Adaptive Fault-Tolerant Routing Algorithm for Network-on-Chip Architectures," in *Proceedings of the Euromicro Conference on Digital System Design*, pp. 527–534, August 2007
40. D. Park, C. Nicopoulos, J. Kim, N. Vijaykrishnan, and C. R. Das, "Exploring Fault-Tolerant Network-on-Chip Architectures," in *Proceedings of the Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 93–104, June 2006
41. Y. Hatanaka, M. Nakamura, Y. Kakuda, and T. Kikuno, "A Synthesis Method for Fault-Tolerant and Flexible Multipath Routing Protocols," in *Proceedings of the International Conference on Engineering of Complex Computer Systems*, pp. 96–105, September 1997
42. W. Stallings, *Data and Computer Communications*, Prentice Hall, 2007
43. T. Dumitras, S. Kerner, and R. Marculescu, "Towards On-Chip Fault-Tolerant Communication," in *Proceedings of the Asia and South Pacific Design Automation Conference*, pp. 225–232, January 2003
44. M. Pirretti, G. M. Link, R. R. Brooks, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "Fault Tolerant Algorithms for Network-on-Chip Interconnect," in *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, pp. 46–51, February 2004
45. J. Hoffman, D.A. Ilitzky, A. Chun, and A. Chapyzenka, "Architecture of the Scalable Communications Core," in *Proceedings of the International Symposium on Networks-on-Chip*, pp. 40–52, May 2007
46. E. S. H. Hou, N. Ansari, and H. Ren, "A Genetic Algorithm for Multiprocessor Scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 2, pp. 113–120, February 1994
47. C. M. Krishna and K. G. Shin, *Real-Time Systems*, WCB/McGraw Hill, 1997
48. H. El-Rewini, H. H. Ali, and T. Lewis, "Task Scheduling in Multiprocessor Systems," *IEEE Computer*, vol. 28, no. 12, pp. 27–37, December 1995
49. T. Burd and R. W. Brodersen, "Energy Efficient CMOS Microprocessor Design," in *Proceeding of the Hawaii International Conference on System Sciences*, pp. 288–297, January 1995
50. G. Quan and X. Hu, "Energy Efficient Fixed-Priority Scheduling for Real-Time Systems on Voltage Variable Processors," in *Proceedings of the Design Automation Conference*, pp. 828–833, June 2001
51. M. T. Schmitz and B. M. Al-Hashimi, "Considering Power Variations of DVS Processing Elements for Energy Minimization in Distributed Systems," in *Proceedings of the International Symposium on Systems Synthesis*, pp. 250–255, October 2001

52. M. T. Schmitz, B. M. Al-Hashimi, and P. Eles, "Energy-Efficient Mapping and Scheduling for DVS Enabled Distributed Embedded Systems," in *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 514–521, March 2002
53. M. T. Schmitz, B. M. Al-Hashimi, and P. Eles, "Iterative Schedule Optimization for Voltage Scalable Distributed Embedded Systems," *ACM TECS*, vol. 3, no. 1, pp. 182-217, February 2004
54. V. Kianzad, S. S. Bhattacharyya, and G. Qu, "CASPER: An Integrated Energy-Driven Approach for Task Graph Scheduling on Distributed Embedded Systems," in *Proceedings of the International Conference on Application-Specific Systems, Architectures and Processors*, pp. 191–197, July 2005
55. P. C. Chang, I. W. Wu, J. J. Shann, and C. P. Chung, "ETAHM: An Energy-Aware Task Allocation Algorithm for Heterogeneous Multiprocessor," in *Proceedings of the Design Automation Conference*, pp. 776–779 June 2008
56. Y. Zhang, X. Hu, and D. Z. Chen, "Task Scheduling and Voltage Selection for Energy Minimization" in *Proceedings of the Design Automation Conference*, pp. 183–188, June 2002
57. G. Varatkar and R. Marculescu, "Communication-Aware Task Scheduling and Voltage Selection for Total Systems Energy Minimization," in *Proceedings of the International Conference on Computer-Aided Design*, pp. 510–517, November 2003
58. J. Hu and R. Marculescu, "Energy-Aware Communication and Task Scheduling for Network-on-Chip Architectures under Real-Time Constraints," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, pp. 234–239, February 2004



<http://www.springer.com/978-1-4419-9340-3>

Reconfigurable Networks-on-Chip

Chen, S.-J.; Ying-Cherng, L.; Tsai, W.-C.; Hu, Y.-H.

2012, XIV, 206 p., Hardcover

ISBN: 978-1-4419-9340-3