UNIVERSITY OF CINCINNATI

Date: September 15, 2006

I, Ming Li,

hereby submit this work as part of the requirements for the degree of:

Doctor of Philosophy

in:

Computer Science and Engineering

It is entitled:

Design and Test of A High Performance Network-on-Chip Architecture for Highly Integrated Systems

This work and its defense approved by:

Chair: Dr. Wen-Ben Jone Co-Chair: Dr. Qing-An Zeng

Dr. Hal Carter

Dr. Ranga R. Vemuri

Dr. Fred Beyette

Dr. Samuel Huang

Design and Test of A High Performance Network-on-Chip Architecture for Highly Integrated Systems

A Dissertation submitted to the
Division of Research and Advanced Studies
of the University of Cincinnati

in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in the Department of
Electrical and Computer Engineering and Computer Science
of the College of Engineering
September 2006

by

Ming Li
M.S. (Computer Engineering), University of Cincinnati, May 2004

Thesis Advisor and Committee Chair: Dr. Wen-Ben Jone
Co-Advisor: Dr. Qing-An Zeng

Abstract

A layered architecture called Network-on-Chip (NoC) has been recently proposed for global communication in a complex System-on-Chip (SoC) design to meet the performance requirements, and many new design and testing issues come up correspondingly. In this work, we aim to develop an efficient test strategy for NoC embedded core testing with a high performance router architecture which can support normal mode as well as testing mode operations.

For normal mode router design, we propose to use a novel dynamic XY (namely DyXY) routing method, which provides adaptive routing based on congestion conditions in the proximity, and ensures deadlock-free and livelock-free features at the same time. Analytical models based on queuing theory are developed for DyXY routing in two-dimensional mesh architectures, and analytical results match very much with the simulation results. It is observed that DyXY routing can achieve much better performance when compared with static XY routing and odd-even routing. Hardware is also designed to support the proposed DyXY routing method efficiently. For the embedded core testing, we propose a multiple-data-flit-format (MDFF) test data transportation concept, a heuristic wrapper scan chain configuration method, and a test scheduling algorithm which considers both channel capacity and data flit interleaving in the channels and routers. By applying the proposed test scheduling method together with the MDFF concept and the heuristic scan chain configuration method, the on-chip network channel of a NoC can be fully utilized for embedded core testing, the test time for the entire NoC can be minimized, and the test power dissipation can be controlled well. By comparing the results with other published works, it has been demonstrated that the proposed test scheduling method can achieve significant improvement on the test time for the entire NoC. To support the proposed embedded core testing strategy, the design issues for testing mode operations have also been explored, and a complete router architecture is presented to support both normal mode (DyXY) and testing mode operations. With all these works completed, we have an efficient NoC embedded core testing strategy with the support of a router architecture which provides high performances in test mode as well as normal mode operations.

Acknowledgements

I wish to express my sincere thanks to my advisor, Dr. Wen-Ben Jone, for the guidance and constructive criticism that he provided throughout my work. He was always ready to provide his guidance and help in solving problems, even during weekends. This work would never have taken this form without his encouragement and expertise.

Special thanks to my co-advisor, Dr. Qing-An Zeng, for his stimulating suggestion and discussion throughout this work.

I would also like to thank Dr. Hal Carter, Dr. Ranga Vemuri, Dr. Fred R. Beyette and Dr. Samuel Huang, for accepting to be part of my dissertaion committee and spending their valuable time reviewing this work in spite of their busy schedule.

Especially, I would like to express my deepest gratitude to my husband Chang, my parents and my sister, for their constant support, understanding and unconditional love.

Thanks all my friends for their help and the joy they bring to my life.

Contents

List of Figures

Chapter 1

Introduction

With the improvement of manufacturing process in semiconductor devices, the number of transistors fabricated onto a single chip has increased tremendously. It has paved the way for incorporating various systems on a single chip (SoC). This trend of SoC design is not only popular but also economical for integrating complex systems onto a single chip. With the feature size of 50 nm and below, a SoC consisting of 4 billion transistors running at 10 GHz will be a reality [2]. This has led to an increase in the production of intellectual property (IP) cores that can be used readily for various SoCs. Complex SoCs can be realized by integrating various IP cores from different vendors [3][4]. The major factors that are affected by this kind of component reuse are reliability in the integrated SoC and its performance. There is also a need for integrating numerous IPs to perform different functions operating at different clock frequencies on the same chip.

As the size of a chip increases, so does the complexity of the design of its interconnect resources. Some of the requirements for an interconnect architecture for future SoCs are: scalable architecture, heterogeneous architecture, reliable communication, quality of service, less power dissipation, reusability of design and reusability of interconnect resources. Traditional bus-based architecture cannot satisfy all of the above requirements. We can clearly see a necessity to make communication and computation architectures orthogonal for complex SoCs to meet the performance requirements. This would result in adding more intelligence to the communication architecture [5]. To address all the above issues, a layered approach along the lines of traditional networks has been proposed for global communication in a SoC. The interconnect architecture consists of a message passing network for on-chip communication called Network on Chip (NoC) [2].

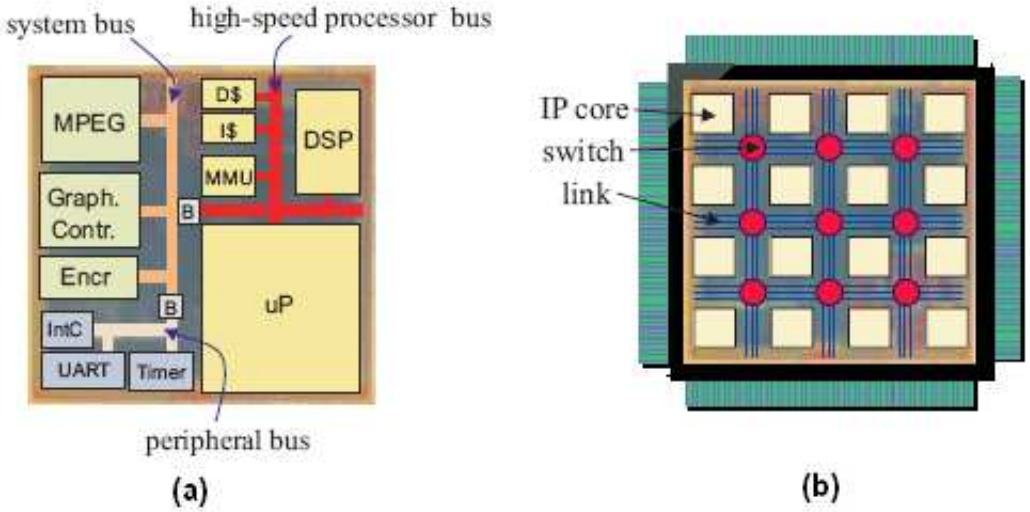


Figure 1.1: Examples of (a) typical bus-based SoC and (b) NoC architectures [1].

A NoC system consists of a number of processing elements connected to each other by a communication network. The communication network consists of a number of routers for directing the messages to appropriate processing elements. The network architecture for interconnects decouples the communication from the processing elements. Proper design of the network can make it easily scalable. The important aspect in such communication architecture is to rightly adapt its characteristics such that they meet the requirements for a complex SoC system [6] -[9]. However, many design and testing issues also come up with this new SoC architecture correspondingly.

1.1 Design and Testing issues for NoC

1.1.1 Design Issues

The main NoC design consideration is about the router architecture.

Routers are the most important components in NoC architecture. A routing algorithm is used to determine the path of a packet traverses from the source to the destination. Generally, routing algorithms can be classified as *deterministic* routing and *adaptive* routing. With deterministic routing, the path of a packet is fixed for the given source and destination addresses. The advantage of deterministic routing is its simplicity in router design. However, when the packet injection rate increases, deterministic routings are likely to suffer from throughput degradation as

they cannot respond to the network congestion dynamically. With an adaptive routing algorithm, instead of determining the path a priori, the path is determined based on the congestion conditions in the network, thus reducing the latency in the system. Since adaptiveness reduces the chance for packets to enter hot-spots or faulty components, and hence reduces the blocking probability of packets, it is an important factor for message routing.

The other important requirement of a routing algorithm is the freedom from *deadlock* and *livelock*. Livelock is a condition in which a message may never arrive at its destination, and it is possible only when message routing is adaptive and is nonminimal. Deadlock occurs when packets wait for each other in a cycle. Many works have been done to develop efficient routing methods for mesh structure computer networks and NoC architectures [10]-[20]. Although they have achieved some progress, there are still some limitations on the adaptiveness of the routing algorithms and complexity of the router architecture. A NoC does not only work under normal mode, it also has some special operation modes such as burst mode and testing mode. Complete router architecture should also be capable to support these operation modes.

1.1.2 Testing Issues

Testing of the NoC including testing the embedded cores and the on-chip network (the routers and interconnects).

Testing the embedded cores in a NoC poses considerable challenges. Reuse of the existing on-chip communication resources, such as routers and channels, is critical to avoid additional area overhead [21]. However, reusing the NoC resources efficiently is challenging because the design of routers and channels in on-chip networks is optimized for communication in mission-mode, not for test. For example, there may be a mismatch between the available network channel width and the core scan chain width (which is usually equal to the Test Access Mechanism (TAM) width for traditional SoC architectures [22][23]), and this can adversely affect test efficiency and test cost.

Most embedded cores use scan test to verify the functional and structural correctness for their random logic circuits. Scan chain configurations in SoC architectures have been fully researched, and good results have been successfully accomplished. However, the objective of scan testing in a NoC is different from that in a SoC, so the detailed configuration method is also different. The major difference comes from the following two points.

1) In a traditional SoC architecture, the width of TAM directly affects the cost of test, so each embedded core can allow only very few wrapper scan chains. The scan chain configuration has to be limited by this requirement, and the test application time for a single core has to sacrifice. However, it is no longer a bottleneck in a NoC where there is no common TAM. Instead, test patterns and output responses are transferred using the existing on-chip communication network. Each embedded core is equipped with a wrapper (i.e., network interface) for mission-mode operations to serve all I/Os of the core. These connections to all I/Os can be used in test mode as a test access port, and the number of wrapper scan chains only has a limit with the network channel width; in most common cases, it is much smaller than the network channel width. So, the minimum test application time for each core can be easily satisfied as long as we limit the length of the longest internal scan chain as the maximum length of all wrapper scan chains.

2) In a traditional SoC, wrapper scan chains are configured as balanced (i.e., equal length) as possible, and all bits of each test pattern are scanned into the scan chains simultaneously. Since the assigned TAM channel width is the same as the number of wrapper scan chains, this can minimize the waste of the channel bandwidth. However, it is not the case in a NoC. The network channel structure is fixed and designed for mission-mode operations in a NoC, so there may be a mismatch between the network channel width and the number of wrapper scan chains in a core logic. This problem will not affect the test time for a single core, but the waste of the network channel bandwidth will result in extra network traffic, and thus has a great effect on the total test time for the entire chip.

Taking these two differences into consideration, we can find that the wrapper scan chain configuration in a NoC-based system is a totally different problem from that in a traditional SoC, and thus a new method has to be developed.

With the introduction of NoC, valuable works have been done for embedded core testing based on this new architecture [21]-[28]. The works in these literatures addressed the reusing of on-chip network for embedded core testing, and proposed test scheduling methods for this new architecture. However, none of them addressed the wrapper scan chain configuration issue. Therefore, the utilization of the on-chip network is still limited, and the testing time for the entire NoC using these proposed methods is not efficient enough.

1.2 Our Work

Our aim of this research is to develop an efficient testable NoC architecture, i.e., we aim to (1) develop an efficient test strategy for NoC embedded core testing, and (2) design a router architecture which supports the testing mode and provides good performance for normal mode operations at the same time.

For the normal mode router architecture design, we propose a novel routing method, namely dynamic XY (DyXY) routing, which enables adaptive routing based on congestion conditions in the proximity, and ensures deadlock-free and livelock-free features at the same time. The adaptiveness lies in making routing decisions by monitoring congestion status in the proximity. The deadlock-free and livelock-free features are incorporated by limiting a packet to traverse the network only following one of the shortest paths between the source and the destination. Analytical models based on queuing theory are developed for both XY routing (we call it static XY in the following part of this thesis for comparison) and DyXY routing to evaluate their performance for a two-dimensional mesh NoC architecture. Extensive simulation is done to validate the analytical models, and it will be demonstrated that the simulation results match very well with the analytical results. To further evaluate the performance of DyXY, we compare it with both static XY routing and odd-even routing under different traffic patterns, and it is shown that the DyXY routing method can achieve the best performance.

For embedded core testing, we propose a new test data transportation method using *multiple data flit formats* (MDFF). With this method, a data flit can contain multiple bits for each wrapper scan chain, instead of only one bit/chain in traditional test application methods. Also, the data flits for a core can have different formats to adapt with the number of unfilled scan chains, for maximum utilization of network channels. To address the difference between traditional SoC test bus and NoC interconnect topology, we propose a new scan chain configuration concept called Factorial Scan Chain Group (FSCG), instead of equal-length configuration methods adopted in traditional bus-based SoCs. A heuristic wrapper scan chain configuration method is developed based on this new concept, to reduce both the test application time and the waste of data flits for testing cores in a NoC. A test scheduling method is also proposed to optimize the sequence of embedded core testing to reduce the test time for the entire chip. Test flit routing and flit interleaving issues are

investigated in detail. Performance evaluations of the wrapper scan chain configuration and test scheduling methods are conducted by the simulation results on the ITC'02 benchmark set.

To support the proposed embedded core testing strategy, we also explore router design issues under testing mode, and propose a complete router architecture which can support both normal mode and testing mode operations.

The dissertaion is organized as follows:

Chapter 2 reviews the basic concepts of network-on-chip, including its architecture, routing methods, and testing issues.

Chapter 3 presents the DyXY routing method for normal mode operations which can provide adaptive routing with deadlock free and livelock free features. An analytical model based on queuing theory is also developed for the DyXY routing method. Experimental results are presented to verify the analytical model and to compare the DyXY routing method with the static XY and odd-even routing methods.

Chapter 4 demonstrates the MDFF data flit transportation method and a heuristic wrapper scan chain configuration method based on the MDFF method for NoC embedded core testing. Experimental results on ITC'02 benchmark sets are presented to evaluate the wrapper scan chain configuration method.

Chapter 5 proposes a test scheduling method based on the heuristic wrapper scan chain configuration method for NoC embedded core testing. The test scheduling method takes both test resource capacity and test flit interleaving into account. The performance of the scheduling method is evaluated by the experimental results on ITC'02 benchmark sets.

Chapter 6 discusses the router design issues under testing mode, and presents a complete router architecture for both normal and testing mode operations.

Chapter 7 concludes the dissertaion and proposes future works for testing the interconnect components (including the routers and interconnects) of a NoC.

Chapter 2

Background

2.1 Emergence of NoC

Chip complexities of up to 1 billion transistors on a single piece of silicon is feasible by the end of this decade[5]. With this development, a whole system (processing elements, memories, communication infrastructure, analog I/O etc) can be virtually build on one single chip (SoC), which will bring lots of profit to the embedded system market.

In a SoC, a large part of the transistors may be used as embedded memory, however, a significant share should be used to increase the number of on-chip processing units in order to enhance system performance. Given the complexities of todays state-of-the-art embedded processors, a single chip can easily accomodate several hundreds or even thousands of embedded processors. However, no more than 10 to 15 processors can be integrated in a single chip of todays complex SoCs. Obviously, the maximum possible amount of transistors (silicon-technology-wise) per chip are unable to be exploited. This problem becomes even more evident if the number of transistors per SOC without the embedded memory is excluded. Clearly, although there is a demand for higher complexities from an application demand point of view, real-world SOCs complexities still lag a lot behind the capabilities of silicon technologies [1].

How does this happen? A possible answer can be found from Figure 2.1. In this figure, the red graph shows the number of available gates per chip for a given silicon technology for SOCs, and the blue graph showns the number of actually used gates per chip for a given silicon technology. As we can see, there is a big gap between these two graph, and the reason of existence of the gap is

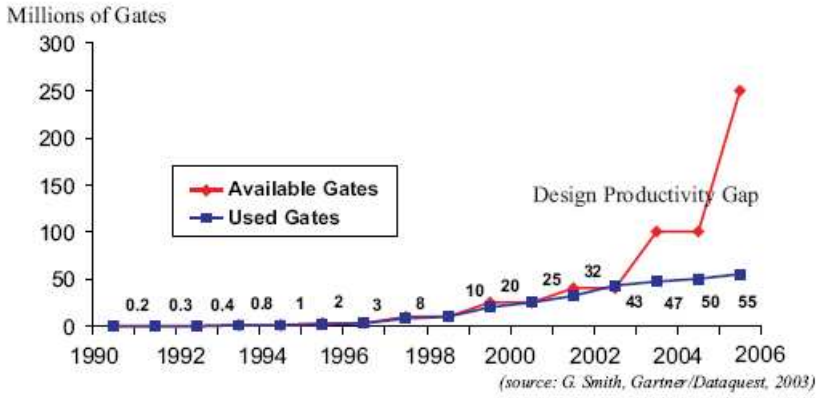


Figure 2.1: Chip complexity and design complexity crisis [1].

due to the lack of ESL (Electronic System Level Design) methodologies. Fortunately, with current developments in ESL methodologies, it can be predicted that this gap will eventually be closed, and thus it will be feasible for the system designer to integrate all billions of transistors on a single chip as silicon technology provided by the end of the decade.

Billion-transistor SoCs have manifold application areas, including security systems, control systems, individual health systems, and main stream consumer products in areas such as personal communication, personal computing, entertainment, video/photo and etc. High complexity of these devices is already a big problem today, and the problem will be even severe with the rising tendency to integrate larger functionality in new device generation. The trend of more complex SoCs will significantly change the way SoCs are designed, both from a design methodology point of view and from an architectural point of view. With hundreds or even thousands of processors to be integrated on a single chip, a sophisticated communication infrastructure is required. Bus-based communication infrastructures, even those using hierarchies of buses will not be sufficient.

The problem comes from several aspects. First, large bus lengths are prohibitive in future designs since it will lead to nonmanageable clock skews which is not tolerable for future large SoCs with high clock frequencies. Shrinking process geometries reduce the cross section of wires, which increases the wires resistance per length, while does not scale down their capacitance accordingly. Wires become slower compared to gates, as illustrated in Figure 2.2. The size of today's SoCs are already larger than 10 10 mm, which forces a tight timing constraints for bus-based, synchronous communication structures. Clock-skew control issue with slow wires in larger system becomes

increasingly difficult.

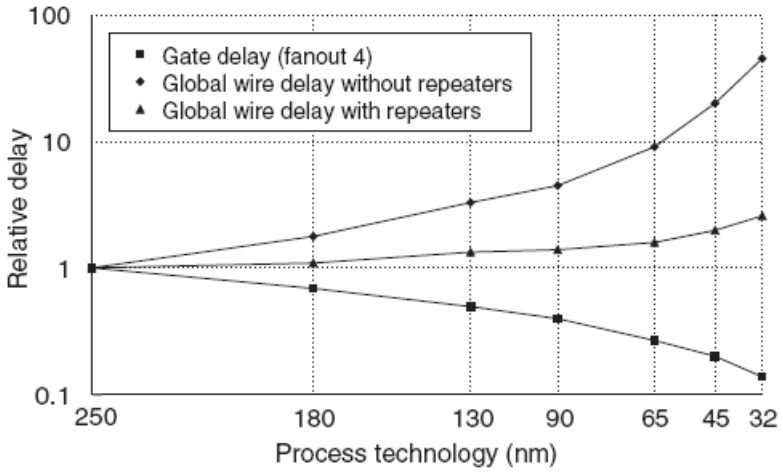


Figure 2.2: Relative evolution of wire and gate delays(source: 2003 International Technology Roadmap for Semiconductors, Sematech, 2003).

From the other aspect, maintaining globally synchronous bus protocols between cores doesn't seems practical with SoCs with multiple clocks. Many IP blocks in SoCs have their own natural clock rate, driven by the components with which they interface or by the real-time data flows they serve. And SoCs with dozens of clocks is already a reality today. Just as the on-board buses (such as PCI) have moved to a point-to-point high-speed network implementation (PCI-Express), transition of on-chip communications of SoCs to point-to-point networks on chips between locally clocked IP blocks or subsystems, is an inevitable trend.

The scalability and success of the Internet has inspired researchers to borrow the ideas of switch-based (routers) networks and packet-based communication for on-chip communication. Networks-on-Chip, NoCs, is emerging as a new design paradigm to overcome the limitations of todays bus-based communication infrastructure. NoCs use layered communication protocols, decoupling the physical transmission of bits through point-to-point wires from higher-level aspects, such as IP socket transactions. The features of NoCs includes scalability, possibility of standardization and reuse of communication architecture. These features are crucial to chip designers to lower design effort, and meet time to market constraints for new products. In the next section, we will introduce the detailed architecture and propertise of NoCs.

2.2 Basics of NoC Architecture

The concept of NoC is derived from computer networks and a typical NoC architecture is shown in Figure 2.3.

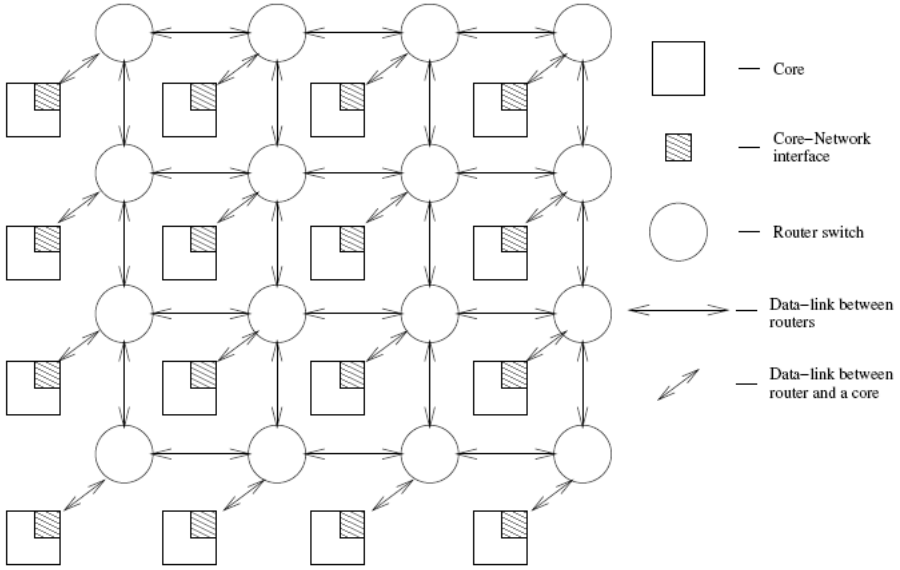


Figure 2.3: A typical NoC architecture.

A NoC architecture is mainly consists of several basic components:

- **Processing core** - a processing core can be a microprocessor, a DSP processor, an ASIC, memory or an I/O processor. Sometimes, it is also referred as a processing element (PE).
- **Network interface** - it acts as a wrapper between the processing core and the router. Network interface performs two functions. The first one is to collect information from the processing core, packetize and insert them into the network. The other one is to collect packets from the network, de-packetize them and send the information to the processin unit.
- **Interconnect** - Interconnets denote the physical wires connecting processing cores. The interconnect topology defines how the processing cores are interconnected to each other through the interconnect channels. Each topology is associated with various factors such as node degree, maximum hop distance, network diameter, and channel width. Some of the most common interconnect topologies are fat-free, 2D mesh, 2D torus and Octagon. Architecture

details and real implementations of the NoC design are presented in [29]-[32], and the basic topologies are shown in Figure 2.4. The choice in the interconnect topology depends on various factors such as performance, load distribution, power consumption, regularity, design complexity, layout compatibility, and chip area. A complete performance comparison of these interconnect topologies can be found in [33].

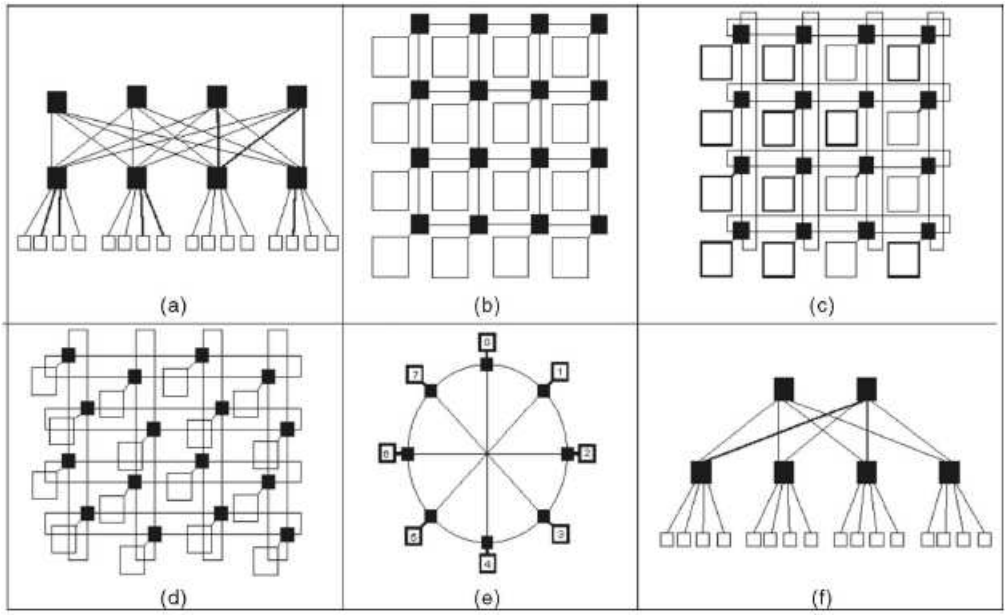


Figure 2.4: NoC interconnect topologies (a)SPIN (b)CLICHE (c)Torus (d)Folded torus (e)Octagon and (f)BFT [33].

- **Router** - routers are used to direct the flow of communication. A router receives a packet at one of its inputs and forwards the packet to one of its outputs based on the routing information. Each router is connected to a processing core, but the degree of the router depends upon the chosen communciation topology. There are mainly two switching techniques called *circuit switching* and *packet switching*. In circuit switching, the connection between the source and the destination is made prior to the communication. Once the connection is established, it stays active for the fixed time. All packets reach the destination in order, and the bandwidth offered by this switching is guaranteed and fixed. However, this switching method suffers from long setup delay, unused channel bandwidth and inflexibility. In packet switching, data is transported from the source to the destination in the form of packets. Each packet is transferred asynchronously in the network, i.e., each router makes its own decision

to send the packet to the next available router. Packet switching offers the best effort service by an efficient utilization of the resources but the order of the packets cannot maintained. This switching mechanism also suffers from congestion, long waiting time for each packet and the bandwidth is not guaranteed. NoC requires both the guaranteed throughput and best throughput services depending on the application, but incorporation of both switching mechanisms add too much complexity to the design. This problem can be solved by making use of efficient flow control mechanisms such as *store and forward*, *wormhole routing*, and *virtual channel flow control*. The details of these flow control mechanisms can be found in [34]. The choice of the flow control is dependent on the silicon cost, design complexity and performance requirements. Here, we would like to spend a little more space to descibe a popular packet switching flow control mechanism: *wormhole routing*. With wormhole routing, a packet is divided into flow control digits (or flits). The flits are routed through the network one after another in a pipeline fashion. The first flit of a packet is designated as the header flit, which contains routing information and leads the packet through the network. When the header flit is blocked from advancing due to lack of output channels, all of the flits wait at their current nodes for available channels. Each router only requires small buffer space to store the flits and communication latencies are low with wormhole routing. The routing algorithm we will present next in this disseration is based on the wormhole routing flow control mechanism.

Compared with traditional SoC architecture, the NoC architecture has some advantages:

- **Parallel Communication:** The NoC architecture can support parallel communication, and the amount of parallelism depends on the topology used.
- **Regular Structure:** The regular structure of the NoC and the relatively short interconnect help to reduce the coupling effects and thus can effectively relief the signal integrity problem.
- **Reusability:** NoC allows to reuse the already designed processing cores and interconnect resources to reduce development time and the time to market.
- **Scalability:** NoCs are totally scalable. The degree of scalability depends on various factors such as flow control, router design and topology.

- **Heterogeity:** SoCs tend to run heterogeneous applications on heterogeneous architectures. Many SoC applications perform several different types of algorithms, and the processing elements are chosen to match the computational load of each process to the processing element on which it runs. Communication loads may also vary considerably between pairs of points in a typical SoC architecture. With NoC architecture, the communication section is totally hidden from the computation section. Different processing cores can operate at different clock frequencies and to interact with the network interface for communication with each other. All communication in the network can be either synchronized to one global clock or can be totally asynchronous. This new network architecture can better match the traffic and help to reduce the network size and power consumption.

However, the NoC architecture also has some disadvantages:

- **Area:** Due to the extra communication components (wires, router and network interfaces) added to the chip, the NoC architecture consumes more area than traditional SoC architecture. A recent work [6] reported that the network logic occupies around 6-7% of the entire chip area.

- **Under Utilization:** Some resources are under utilized if the network payload is smaller than the network capacity. Careful analysis must be performed at the design time for the proper utilization of all resources.

- **Power Consumption:** Power consumption becomes a major problem due to increased clock frequencies in conjunction with high integration ratio. In NoC architecture, although the specific power consumption of a wire in an on-chip network might decrease, the sheer amount of added wires (as part of the links between routers/switches) will increase sharply due to the increasing number of IP cores residing on future SoCs. In addition, the communication will likely increase too. As a result, the power consumption of the interconnect may significantly contribute to the power budget of the entire chip. Therefore, more efforts are required for detailed analysis of interconnect network power consumption for a feasible design.

2.3 Review of Works on NoC Routing Algorithms and Router Architectures

Routers are the most important components in NoC architecture. A routing algorithm is used to determine the path of a packet traverses from the source to the destination. Generally, routing algorithms can be classified as *deterministic* routing and *adaptive* routing. With deterministic routing, the path of a packet is fixed for the given source and destination addresses. The advantage of deterministic routing is its simplicity in router design. However, when the packet injection rate increases, deterministic routings are likely to suffer from throughput degradation as they cannot respond to the network congestion dynamically. With an adaptive routing algorithm, instead of determining the path a priori, the path is determined based on the congestion conditions in the network, thus reducing the latency in the system. Since adaptiveness reduces the chance for packets to enter hot-spots or faulty components, and hence reduces the blocking probability of packets, it is an important factor for message routing. The other important requirement of a routing algorithm is the freedom from *deadlock* and *livelock*. Livelock is a condition in which a message may never arrive at its destination, and it is possible only when message routing is adaptive and is nonminimal. Deadlock occurs when packets wait for each other in a cycle. Deadlock-free and livelock-free are important criterias to guarantee the performance of routing algorithms.

Many routing algorithms dealing with networks with the mesh architecture have been proposed for deadlock-free and adpativeness recently. In [10]-[14], *virtual channels* are introduced to assist the design of nonadaptive and adaptive routing algorithms for a variety of network architectures. Virtual channels are abstractions that share the same physical channel. Although adding virtual channels can allow the adaptiveness of routing algorithms, it sacrifices with extra buffer space and complex router control logic, and thus may affect the network performance and router reliability [15].

In [15]-[18], routing algorithms that require no virtual channels have been proposed for networks with the mesh architecture. A static XY routing algorithm for two-dimensional meshes has been presented in [16]. With static XY routing, a packet first traverses along the x dimension and then along the y dimension. This algorithm is deadlock-free but provides no adaptiveness. The work in [15] proposed another algorithm called the *turn* model, which is a partially adaptive routing

algorithm without virtual channels. The basic idea of the turn model is to prohibit the minimum number of turns to break all of the cycles, so that deadlock can be avoided. Based on the turn model, three partially adaptive routing algorithms, namely *west-first*, *north-last*, and *negative-first* were presented for two-dimensional meshes. In [17], a routing algorithm based on the turn model was proposed. This routing algorithm is called *odd-even turn*. It restricts some locations where turns can be taken so that deadlock can be avoided. In comparison with the previous methods, the degree of routing adaptiveness provided by this model is more even for different source-destination pairs. A direction restriction model for a partially adaptive routing algorithm has been proposed in [18]. A system with this model is divided into two unidirectional networks, and message routing is done in two phases. A packet is routed adaptively to an intermediate node using one unidirectional network in the first phase, and then routed adaptively to the destination using the other network in the second phase.

With the introduction of NoC, some works about routing for NoC architectures have been done. In [19], the authors showed the benefit of using proximity congestion status to aid hot-potato routing (routing without buffers) in a deflective network. It was shown that the maximum tolerable load can increase significantly with the proximity congestion awareness technique. A routing scheme called DyAD was proposed in [20]. This algorithm is actually a the combination of a deterministic routing algorithm called *oe-fix*, and an adaptive routing algorithm called odd-even as introduced in [17], where oe-fix is the deterministic version of odd-even routing. The router can switch between these two routing modes based on the network's congestion conditions.

2.4 Review of Works on NoC Testing Methods

For traditional SoC, test stimuli and test results are transported through the TAM circuit. Hence, many research efforts for SoC testing are focused on the co-optimization of test wrapper and TAM [22]-[35]. Functional I/Os and internal scan chains of an embedded core are generally configured to a designated number of balanced wrapper scan chains whose lengths are as equal as possible. The number of wrapper scan chains is determined by some optimization algorithms (e.g., some ILP algorithms), which take both test wrapper and TAM into consideration, and try to minimize the overall test time and area overhead.

Besides the work on co-optimization of test wrapper and TAM, some other research suggested the reuse of functional connections during test to reduce test costs in terms of area and pin overhead, as proposed in literatures [36]-[39]. A core-to-core connection model is assumed in those methods. Another trend is to use a packet-switching architecture to test embedded cores, as proposed in [40] and [41]. In [40], the authors proposed a test strategy for a homogeneous system composed of an on-chip multiprocessor architecture, where the processors are connected in a network-based model. Test of the routers, the RAM blocks, and the embedded processors were all ecplored in their work. In [41], the use of a packet switching communication-based TAM was proposed for an SOC. The proposed TAM model is called NIMA and is defined to allow modularity, generality, and configurability for the test architecture. The NIMA architecture is very similar to a functional NoC, but it is specifically designed for the test task. , Power consumption during test has not been modeled in neither [40] or [41].

A number of approaches considering power constraints during SoC test scheduling have been presented in literatures [42]-[43]. All these works were based on a bus-based TAM, and the test scheduling was modeled such that two cores will not be tested concurrently whenever the sum of their power consumption during test is larger than the maximum power consumption allowed for the system. The peak power consumption for each core is assumed in all cases, however, the power consumption of the access mechanism has not been considered. The power consumption of the access mechanism can be prohibitive, and it is depended on the length and width of the set of test buses defined for the system. In [44], the author proposed a power profile manipulation approach to minimize the power dissipation during test. Such a manipulation provideed a more realistic power profile to be used by any power constrained test scheduling algorithm, and made the test time reduction possible by increasing the test concurrency.

With the introduction of NoC, valuable works have been done for embedded core testing based on this new architecture. In [21], two methods of using an on-chip network for the test of core-based systems were introduced, and advantages in reducing test time, area overhead and pin-overhead were shown by experimental results. In [24], the authors extended the results of a previous on-chip network research to a test scheduling algorithm with power constraints considered. In this algorithm, scheduling was based on every single packet and the test pipeline for a core can be interrupted. Since this is not applicable for the non-preemptive case, an improved test scheduling

algorithm with BIST and precedence constrains was further proposed in [25]. For all these test scheduling algorithms, each packet only contains a single bit for each wrapper scan chain of the embedded core. Since this may result in a huge waste of packets, a further improved test scheduling algorithm has been proposed in [26]. The algorithm allows a packet to contain multiple bits for each wrapper scan chain. To support test under this new packet format, the authors proposed to use on-chip clocking to speed up the test data transfer for certain cores by faster clocks, and use slower clocks for other cores to limit the power consumption. An algorithm was presented to determine the clock rate distribution among the cores.

A network architecture using the star-connected on-chip network was proposed in [27]. The authors implemented an example NoC and analyzed the core access time and the communication throughput. It was shown that this architecture can result in reduction of test time due to the high bandwidth and smaller area overhead due to the network reuse. Another work [28] evaluated the impact of reusing processors to test a NoC-based system. The results demonstrated that using available processors as source/sink for test can achieve better test parallelism, and hence can reduce the system test time without additional area and test pins.

Chapter 3

Normal Mode Router Design

As introduced in Chapter 2, many works have been done to develop efficient routing methods for mesh structure computer networks and NoC architectures [10]-[20]. Although they have achieved some progress, there are still limitations on the adaptiveness of the routing algorithm and complexity of the router architecture. In this Chapter, we propose a novel routing algorithm, namely dynamic XY (DyXY) routing, which provides an adaptive routing mechanism based on congestion conditions in the proximity, and ensures deadlock-free and livelock-free routing at the same time. A new router architecture is developed to support the routing algorithm. Analytical models based on queuing theory are developed for both static XY routing and DyXY routing for a two-dimensional mesh NoC architecture, and the performance of DyXY routing is compared with both static XY routing and odd-even routing.

3.1 DyXY Routing and Router Architecture

With the DyXY routing algorithm, each packet only travels along a shortest path between the source and the destination (this guarantees the deadlock-free feature of the routing algorithm). If there are multiple shortest paths available, routers will help the packet to choose one of them based on the congestion condition of the network. The detailed routing algorithm can be summarized as follows:

- Read the destination of an incoming packet.
- Compare the destination address with the current router address.

- If the destination is the local core of the current router, send the packet to the local core;
- Else
  - If the destination has the same x (or y) address as the current router, send the packet to the neighboring router on the y-axis (or x-axis) towards the destination;
  - Else, check the stress values of current router's neighbors towards the destination, and send the packet to the neighbor with the smallest stress value.

The stress value is a parameter representing the congestion condition of a router, and it can be the instant queue length (the number of occupied cells in all input buffers) of the router. Each router stores instant stress values for all neighbors. Each stress value is updated based on an event-driven mechanism, i.e., when there is a change with a router's stress value, it will send signals to all neighbors for updating.

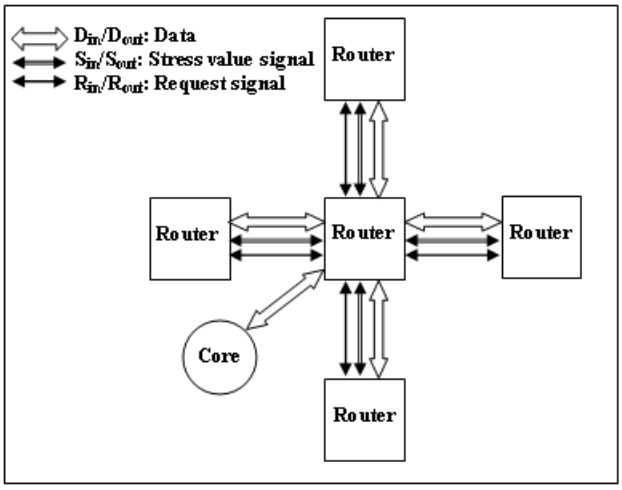


Figure 3.1: NoC interconnections under DyXY routing.

The NoC system interconnection under DyXY routing is shown in Figure 3.1, and the router architecture is shown in Figure 3.2. Each router contains a set of first-in first-out (FIFO) input buffers, an input arbiter, a history buffer, a crossbar switch circuit, a controller, and four stress value counters. The size of each input buffer is a design parameter. In Figure 3.2, D_{in0}/D_{out0} to D_{in4}/D_{out4} represent the data lines between a router and its local core, right router, up router, left router and down router, respectively. R_{in0}/R_{out0} to R_{in4}/R_{out4} represent the request signal lines between a router to its local core and all neighbor routers. S_{in1} to S_{in4} represent the input signal lines for stress value updating of the four neighboring routers, and S_{out1} to S_{out4} represent

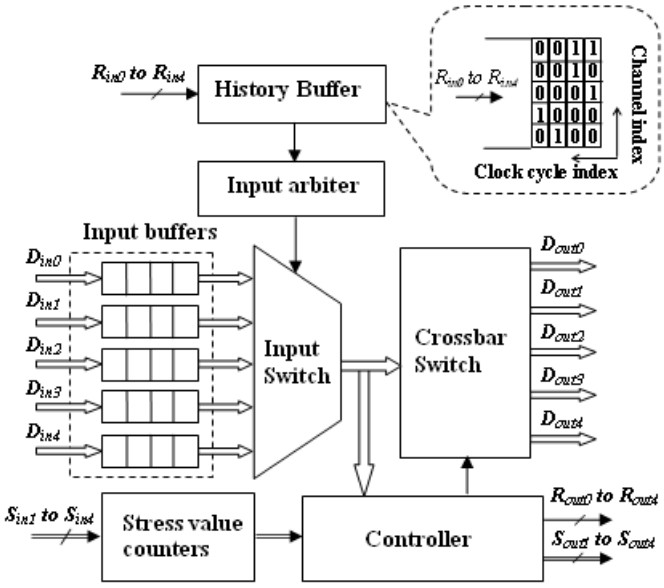


Figure 3.2: Router Architecture for DyXY routing.

the output signal lines to update the local router's stress value to its four neighbors.

At each clock cycle, the history buffer records the channels that have input requests. The input arbiter selects a request from input buffers to process based on the FIFO mechanism referring to records in the history buffer. For example, in Figure 3.2, the sequence to process received requests is channel 2, channel 4, channel 3, channel 4, channel 0, and channel 1 based on the history buffer. The main task of the controller is to determine the routing path for incoming packets, based on the routing algorithm described above. Besides this, the controller also needs to send signals to its neighbors for updating its stress value. When there are new incoming packets from neighbors or the local core, the controller will inform neighbors to increase its stress value. When the outgoing direction for a packet is determined, the controller will set a request signal to the local core or the corresponding neighbor router, and inform all neighbors to decrease its stress value.

3.2 Modeling and Performance Analysis

3.2.1 Router Modeling and Analysis

One of the best indicators for a router's performance is its mean response time. In our analysis, we model each buffer in a single router as a non-preemptive infinite buffer. Although each channel has a separate input buffer, the sequence to process all requests is based on the FIFO

mechanism. Hence, all input buffers of a router can be modeled as a single FIFO buffer for analysis. Using the infinite buffer model, we can estimate the mean waiting time of a packet in each router, and thus can use this information to estimate the required buffer size for each router for a specific traffic load. To model and analyze the mean response time of each router, we need to first analyze the traffic load of each router.

A. Traffic load

Assume that the NoC network is a two-dimensional network with $U \times V$ routers (cores). Router i (core i) has a network address (i_x, i_y) which indicates its x and y coordinates, respectively.

A packet enters a router i due to one of the following three reasons: 1) The packet from core i has to be sent out from its local router; 2) The packet whose destination is core i has to go through its local router; 3) The packet needs to go through router i to be passed to other routers. The traffic due to the first case is fixed regardless of the routing algorithm. The traffic due to the second reason is not affected by the routing algorithm, but will change with different network communication patterns, e.g., the number of packets received by a core from a given source may be different with uniform distribution and non-uniform distribution network communication patterns. However, the traffic due to the third reason will be affected by both the network communication pattern and the routing algorithm used.

Assume that each core generates packets following a random process with Poisson distribution with mean rate λ (λ is also called the average packet injection rate for the NoC). The service time of each router for all packets follows exponential distribution with mean rate μ. Let λ_i be the mean packet arrival rate of router i, λ_{s_d} be the mean rate of packets from core s to core d and $P_{s\_d\_i}$ be the probability of a packet from core s to core d via router i. The mean packet arrival rate of router i can be calculated using the following equation

$$\lambda_i = \lambda + \sum_{s=1}^{U \times V} \sum_{d=1}^{U \times V} \lambda_{s\_d} P_{s\_d\_i}, \text{ for } s \neq d. \tag{3.1}$$

The mean rate λ_{s_d} of packets from core s to core d is determined by the network communication pattern, and the probability of a packet from core s to core d via router i is determined by both the network communication pattern and the routing algorithm. Here, we use the uniformly distributed network communication pattern to model the traffic load of each router with both static

XY and DyXY routing algorithms.

With the uniformly distributed network communication pattern, λ_{s_d} can be calculated as

$$\lambda_{s\_d} = \frac{\lambda}{U \times V - 1}, \text{ for } \begin{cases} 1 \le s \le U \times V \\ 1 \le d \le U \times V \end{cases}, \text{ and } s \neq d. \tag{3.2}$$

For static XY routing, the probability of a packet from core s to core d via router i is given by

$$P_{s\_d\_i} = \begin{cases} 1, & \text{if } i_y = s_y \text{ and } i_x \in [s_x, d_x] \text{ (or } [d_x, s_x]) \\ & \text{or } i_x = d_x \text{ and } i_y \in [s_y, d_y] \text{ (or } ([d_y, s_y]), \\ 0, & \text{otherwise,} \end{cases} \tag{3.3}$$

where $z \in [l, h]$ denotes that z is a value between l and h. Hence, the mean arrival rate of each router with static XY routing can be calculated using Equations (1), (2) and (3).

For DyXY routing, the probability of a packet from core s to core d via router i is given by

$$P_{s\_d\_i} = \begin{cases} 1, & \text{if } i = s \text{ or } i = d, \\ 0, & \text{if } i_y \notin [s_y, d_y] \text{ (or } [d_y, s_y]) \\ & \text{or } i_x \notin [s_x, d_x] \text{ (or } [d_x, s_x]), \\ \sum_{j \in \psi} P_{s\_d\_j} P_{j\_i}, & \text{otherwise,} \end{cases} \tag{3.4}$$

where ψ is the set of router i's neighbors, which is located in a packet's possible routing paths from core s to core d, immediately before router i. Further, P_{j_i} is the probability that router j forwards a packet to its neighbor router i with destination core d, and it can be calculated as follows:

$$P_{j\_i} = \begin{cases} 0, & \text{if } i_y \notin (j_y, d_y] \text{ (or } [d_y, j_y)) \\ & \text{or } i_x \notin (j_x, d_x] \text{ (or } [d_x, j_x)), \\ 1, & \text{if } i_x = j_x = d_x, i_y \in (j_y, d_y] \text{ (or } [d_y, j_y)) \\ & \text{or } i_y = j_y = d_y, i_x \in (j_x, d_x] \text{ (or } [d_x, j_x)), \\ p, & \text{otherwise,} \end{cases} \tag{3.5}$$

where p is a variable depending on congestion conditions of the network. For a packet in router j whose destination is a core in the right-up direction, the packet can be forwarded to either router i ($i_x = j_x$, $i_y = j_y + 1$) or router k ($k_x = j_x + 1$, $k_y = j_y$). If the probability to forward this

packet to router i is p, the probability to forward this packet to router k is $1-p$. Since the DyXY routing algorithm chooses a path based on each possible router's stress value, the probability can be estimated by Equation (6) using the mean waiting time of a packet in the router,

$$p = \frac{W_k}{W_k + W_i}, \tag{3.6}$$

where W_k (or W_i) is the mean waiting time of router k (or router i). Fortunately, W_k can be approximated using M/M/1 queue mean waiting time equation by

$$W_k = \frac{\lambda_k}{\mu - \lambda_k}. \tag{3.7}$$

Since λ_i can be represented by $a_i + b_i \times p$ and λ_k can be represented by $a_k + b_k \times p$ (a_i, b_i, a_k and b_k are system parameters that can be calculated for each router with the given traffic pattern and routing algorithm), by combining Equations (1), (2), and (4) to (7), the value of p can be calculated.

Finally, the mean arrival rate of each router under DyXY routing can be calculated by combining Equations (1), (2), (4) and (5).

B. Mean response time

For static XY routing, the total traffic arrival process follows Poisson distribution, and hence a router can be modeled as a M/M/1 queue. The mean response time of router i can be calculated using the following equation

$$E[Tr_i] = \frac{1}{\mu - \lambda_i}, \tag{3.8}$$

where λ_i can be calculated using Equations (1), (2) and (3).

For the DyXY routing algorithm, since the traffic of each router changes dynamically with network congestion conditions, the real traffic distribution is not Poisson distribution. The mean router response time in this case can be estimated using a pair of upper bound and lower bound.

The real traffic distribution is an interrupted Poisson distribution, which is actually an optimization based on network congestion conditions, therefore, the real mean response time should be smaller than that calculated using the mean arrival rate (λ_i) and the Poisson distribution model.

Hence, the later one can be used as an upper bound of the real mean response time. The upper bound can be calculated using Equation (8) with λ_i calculated using Equations (1), (2) and (4) to (7) as described before.

The minimum traffic of each router occurs when p is set to 0 in Equation (5). In this case, the total traffic arrival process for each router follows Poisson distribution, and the mean arrival rate of each router can be calculated using Equations (1), (2), (4) and (5), by setting p to 0 in Equation (5). Hence, the lower bound of the mean response time of each router can be calculated using Equation (8) with the mean arrival rate of the minimum traffic.

After calculating the mean response time of each router, we can derive the mean waiting time of a packet in each router by $E[Tr_i] - 1/\mu$. The mean buffer size required for each router can be calculated using $E[W_i] \times \lambda_i$ by Little's law [45]. The assignment of the buffer size to each channel of a router can be determined based on the traffic load at each different direction of the router. The average mean response time of all routers, $E[Tr]$, can be calculated by

$$E[Tr] = \frac{1}{U \times V} \sum_{i=1}^{U \times V} E[Tr_i]. \tag{3.9}$$

The performance of a router with finite buffer size α can also be analyzed similarly. The derivation for the mean packet arrival rate is the same. The only difference is that when calculating the mean response time, in stead of using Equation (8), we must use the following equation

$$E[Tr_i] = \lambda_i \frac{1 - \rho_i^{\alpha}}{1 - \rho_i^{\alpha+1}}, \text{where } \rho_i = \frac{\lambda_i}{\mu}. \tag{3.10}$$

Additionally, there is one more performance indicator to check, the blocking probability of packets, which can be represented by

$$P_{B_i} = \frac{(1 - \rho_i)\rho_i^{\alpha}}{1 - \rho_i^{\alpha+1}}. \tag{3.11}$$

3.2.2 System Modeling and Analysis

A NoC system can be modeled as a queuing network. The cores generate packets and inject them into the routing network. Each packet is queued in the input buffer of the first router, and then transmitted to the next router until it reaches its destination. The performance of the system

can be evaluated by the average packet latency $E[Latency]$, which is the product of the average mean response time for all routers $E[Tr]$ and the average packet path length N. The path length of a packet denotes the number of routers that the packet traverses from the source to the destination. Thus, we can calculate the average packet latency by

$$E[Latency] = E[Tr] \times N, \tag{3.12}$$

where $E[Tr]$ can be derived directly by Equation (9), and the average packet path length N depends on the specific communication pattern and routing algorithm employed.

With static XY routing, the path traveled by packets for a given pair of source and destination is always the same. Thus, the length of a path traveled by packets for a given pair of source and destination is a constant which equals the shortest path length. For DyXY routing, although the routing path is not static, it is always a shortest path (as described in Section 3) and hence the length is still the same, i.e., the shortest path length. Therefore, the average packet path length does not change for both routing algorithms, and it is only affected by the communication pattern.

The uniform communication pattern is the most popular one used to evaluate the performance of a system. With the uniform communication pattern, when a packet is generated, the destination address is chosen randomly from all cores with equal probability. However, in reality, cores which have larger communication probabilities are often placed closer to each other to minimize the communication delay. Therefore, the communication probability between the cores may be non-uniform. Without losing the generality, we consider both uniform and non-uniform communication patterns in this proposal, and we choose Poisson distribution for the non-uniform communication pattern.

For a two-dimensional mesh network with $U \times V$ as the total number of routers in the network, let $n_{i,j}$ be the packet path length from core i to core j, and $P_c(i,j)$ be the probability of communications from core i to core j. Further, let N_i represent the average packet path length of all outgoing packets from a given core i. The average packet path length N of communications for the entire network can be calculated as follows:

$$N = \frac{1}{U \times V} \sum_{i=1}^{U \times V} N_i, \tag{3.13}$$

where

$$N_i = \sum_{j=1}^{U \times V} n_{i,j} P_c(i,j), \text{ for } i \neq j. \tag{3.14}$$

Finally, Equation (13) can be simplified as

$$N = \frac{1}{U \times V} \sum_{i=1}^{U \times V} \sum_{j=1}^{U \times V} n_{i,j} P_c(i,j), \text{ for } i \neq j. \tag{3.15}$$

A. Uniformly distributed communication pattern

If communications from a given core i to all other cores follow the uniform distribution, the probability of communications from core i to core j is given by

$$P_c(i,j) = \frac{1}{U \times V - 1}, \text{ for } i \neq j. \tag{3.16}$$

Thus, Equation (15) can be rewritten as

$$N = \frac{1}{(U \times V)(U \times V - 1)} \sum_{i=1}^{U \times V} \sum_{j=1}^{U \times V} n_{i,j}, \text{ for } i \neq j. \tag{3.17}$$

Simplifying Equation (17), if $U = V$, we have the value of N as

$$N = \frac{2U}{3}. \tag{3.18}$$

B. Poisson distributed communication pattern

If communications from a given core i to all other cores follows Poisson distribution, the probability of communications from a given core i to core j is given by

$$P_c(i,j) = \frac{m_i^{n_{i,j}} e^{-m_i}}{n_{i,j}!}, \text{ for } i \neq j, \tag{3.19}$$

where m_i is the mean communication density of all outgoing packets from core i. The average packet path length N of communications for the entire network can be calculated by

$$N = \frac{1}{U \times V} \sum_{i=1}^{U \times V} \sum_{j=1}^{U \times V} n_{i,j} \frac{m_i^{n_{i,j}} e^{-m_i}}{n_{i,j}!}, \text{ for } i \neq j. \tag{3.20}$$

3.3 Experimental Results and Discussions

To evaluate the performance of the DyXY routing algorithm and verify the correctness of our analytical models, we developed an event-driven simulator using C++ and designed three sets of experiments. The mean service rate μ of each router is set to unity for all simulations.

The first set of experiments is based on a 3×3 NoC with average packet injection rate λ increasing from 0.1 to 0.3 under both the DyXY and static XY routing algorithms. Figures 3 and 4 show the mean arrival rates of the routers in the corner (e.g., router (0,0)), routers at the edge (e.g., router (1,0)) and the router in the center (i.e., router (1,1)) of the NoC for the DyXY and static XY routing algorithms, respectively. As we can see, the simulation results precisely match with the analytical results for both routing algorithms.

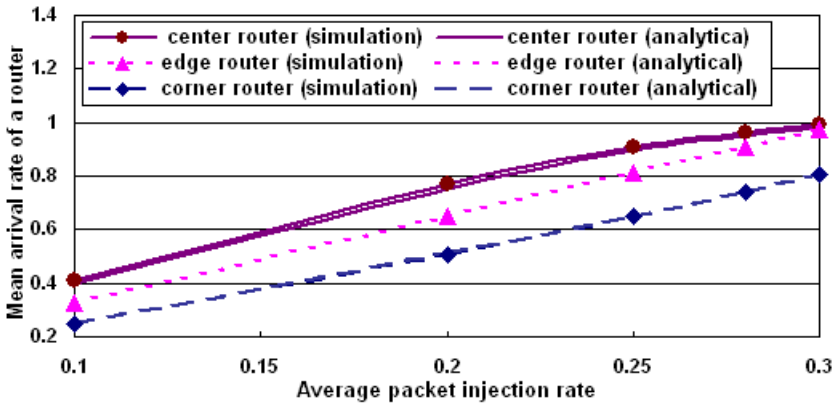


Figure 3.3: Mean arrival rate for a router in 3x3 NoC with DyXY routing.

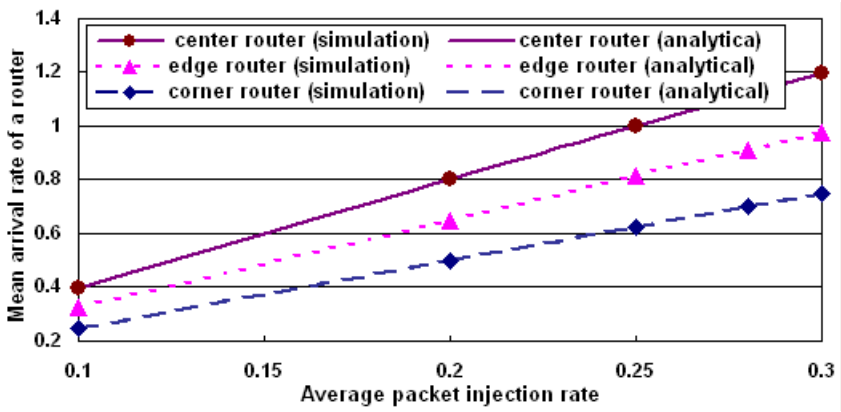


Figure 3.4: Mean arrival rate for a router in 3x3 NoC with static XY routing.

Figures 5 and 6 show the load distribution among different routers for these two routing

algorithms. As we can see, the DyXY routing algorithm achieves better balance in load distribution compared with the static XY routing algorithm, and thus it can relieve the hot-spot problem when network traffic is high.

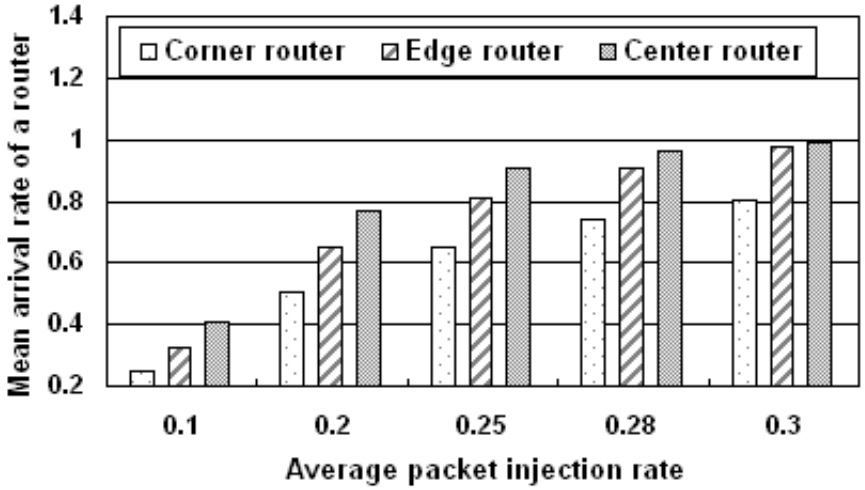


Figure 3.5: Load distribution for routers in 3x3 NoC with DyXY routing.

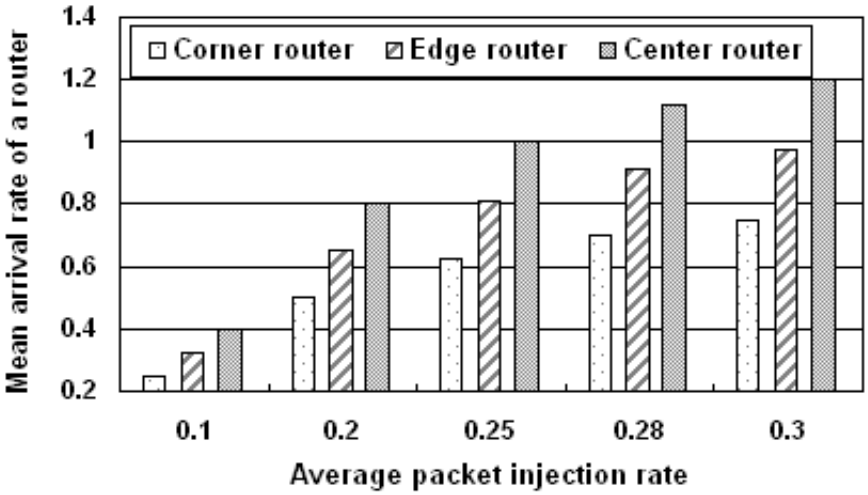


Figure 3.6: Load distribution for routers in 3x3 NoC with static XY routing.

Figure 3.7 shows the analytical and simulation results of the average mean response time for all routers under the static XY and DyXY routing algorithms. The analytical model for static XY routing can precisely evaluate the average mean response time for all routers. For DyXY routing, the average mean response time for all routers can be effectively estimated using the analytical lower bound and upper bound.

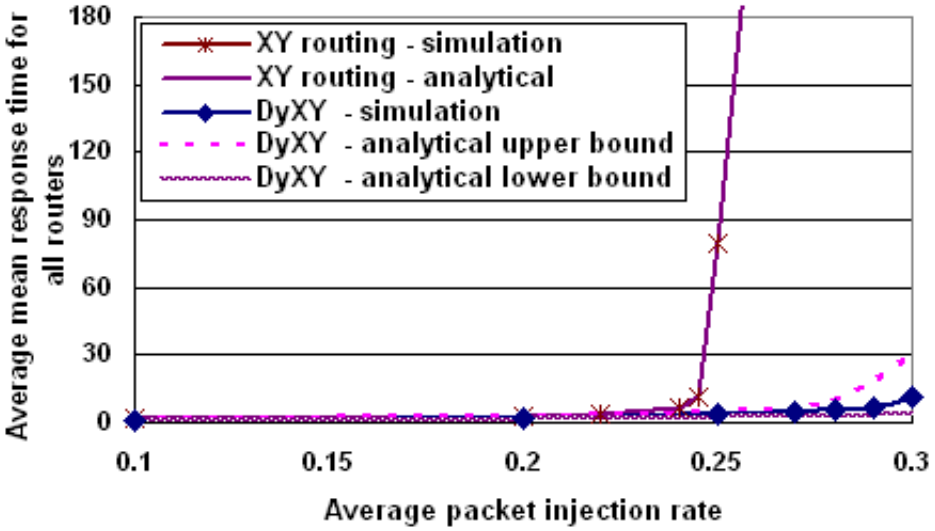


Figure 3.7: Average mean response time for all routers in 3x3 NoC.

Since DyXY routing can balance the load distribution among all routers much better than static XY routing, the average mean response time for all routers ($E[Tr]$) is smaller with DyXY routing than that with static XY routing. Further, since the average packet path length N is the same for both routing algorithms, the average packet latency ($E[Latency]$) with DyXY routing is also smaller than that with static XY routing. This is verified in Figure 3.8, where we also compared the performance of DyXY routing with that of odd-even routing. It is shown that the DyXY routing algorithm also performs better than odd-even routing.

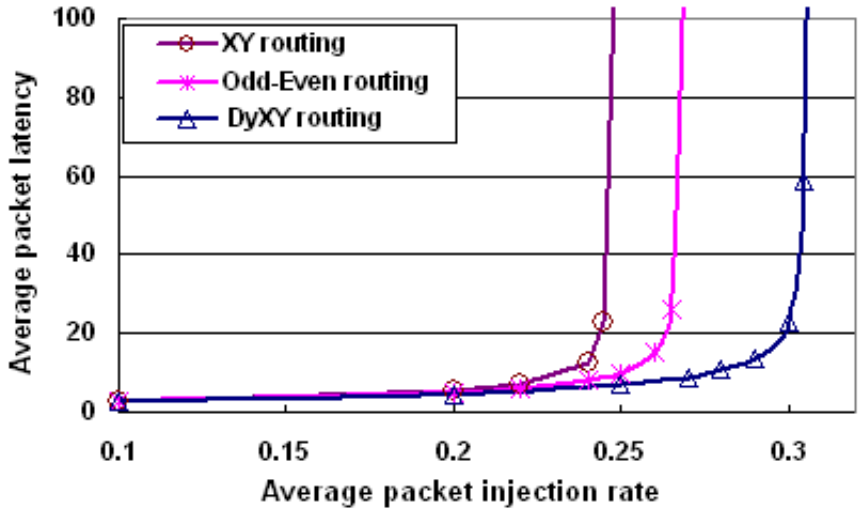


Figure 3.8: Average packet latency for 3x3 NoC with uniform network communication pattern.

To extend this verification, we conducted the second set of experiments with a Poisson distribution network communication pattern. The NoC size is also fixed to 3×3, and the average packet injection rate λ is increased from 0.1 to 0.4, The results are shown in Figure 3.9. Here, it can be observed that DyXY routing achieved the best performance in average packet latency.

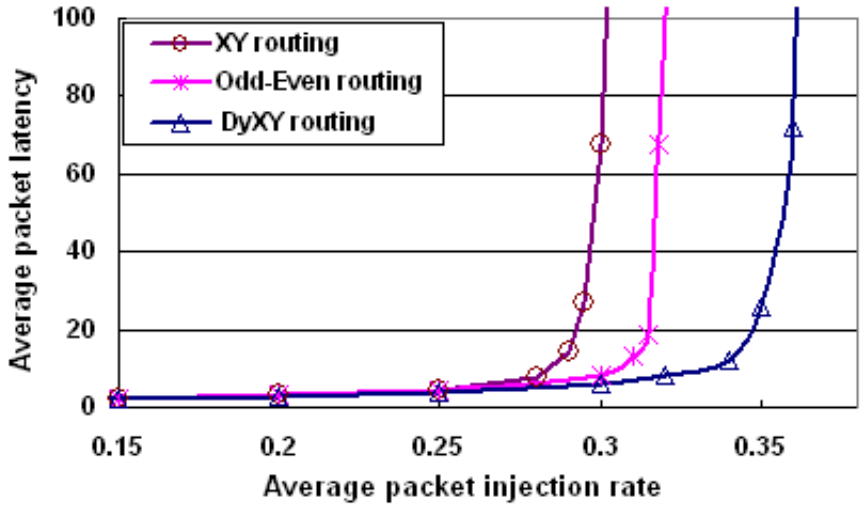


Figure 3.9: Average packet latency for 3x3 NoC with Poisson distributed network communication pattern.

The performance of a network with the Poisson distribution based communication pattern is not sensitive to the network size, however, the performance under the uniform network communication pattern is affected by the network size. Therefore, our last set of experiments changed the size of NoC from 3×3 to 9×9 with λ fixed to 0.15. The results are shown in Figure 3.10. Obviously, it can be seen that the system performs best with DyXY routing.

For each simulation experiment, more than 20,000 packets were inserted into the network, and the simulation kept running till the network remained in stable status for enough time. We have also conducted multiple simulation iterations (by changing the seed of the random number generator in the simulation engine) for each experimental setting, for the purpose of hypothesis testing. For example, to evaluate the mean packet arrival rate for each router with the DyXY routing algorithm, we have run 10 iterations for the condition: external packet injection rate $\lambda = 0.1$. The simulated mean packet arrive rates for the center router locate in the range [0.39532, 0.408416], with $mean = 0.401219$, and the variance within 1.8%. Therefore, the simulation results are accurate enough for evaluation.

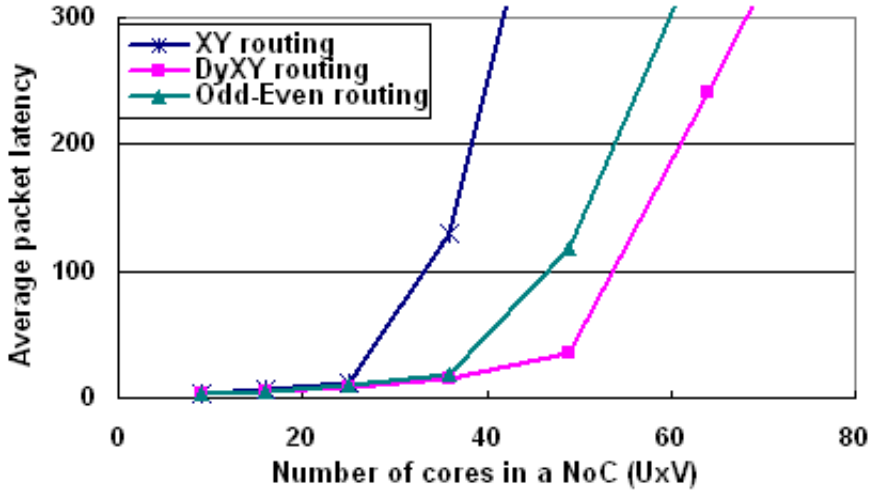


Figure 3.10: Average packet latency for 3x3 NoC with uniform network communication pattern.

Based on the simulation results, we can state that the analytical models based on queuing theory can precisely estimate the performance of static XY and DyXY routing, and DyXY routing can achieve better performance than static XY routing as well as odd-even routing.

Chapter 4

MDFF Test Data Application and Wrapper Scan Chain Configuration for NoC Embedded Core Testing

With the introduction of NoC, valuable works have been done for embedded core testing based on this new architecture [21]-[26]. In [21]-[25], the test scheduling methods only allow each data flit to contain a single bit for each wrapper scan chain of the embedded core, and this may result in a huge waste of channel capacities. An improved test scheduling algorithm has been proposed in [26], which allows a data flit to contain multiple bits for each wrapper scan chain, but the format of data flits for each embedded core is fixed once decided. To support test under this new data flit format, the authors proposed to use on-chip clocking to speed up the test data transfer for certain cores by faster clocks, and use slower clocks for other cores to limit the power consumption. Due to the limit of clock rate, there is strong constraint on the number of bits that a data flit can contain for each wrapper scan chain. Therefore, although this algorithm add more freedom for the data flit format and test scheduling, it is still not efficient enough.

In this chapter, we propose a new test data transportation method using *multiple data flit formats* (MDFF). With MDFF method, a data flit can contain multiple bits for each wrapper scan chain, instead of only 1 bit/chain in traditional test application methods. Also, the data flits for a core can have different formats to adapt with the number of unfilled scan chains, for maximum

utilization of network channels. A heuristic wrapper scan chain configuration method is developed based on the concept of MDFF, to reduce both the test application time and the waste of data flits for testing cores in a NoC. The performance of the configuration method is proved by the simulation results on the ITC'02 benchmark set.

4.1 MDFF Test Data Application

During scan test, all scan chains involved need to be filled with test patterns. To minimize the test pattern application time, data for different scan chains are scanned in simultaneously. In a NoC architecture, communication is in the format of packet-switching, and each packet is composed of a series of data flits [1], where the width of a data flit is equal to the network channel width. Since the width of the network channel may be more than the number of wrapper scan chains, each data flit can contain several bits of test data for each wrapper scan chain instead of only one bit for each. The simplest case is that there is only one data flit format. However, wrapper scan chains may have different lengths, and some of them may finish before others. In this case, after short chains are finished, their input channels are wasted. If there are significant variations between the lengths of different wrapper scan chains, the waste can be excessive. To reduce the waste of network channel due to the variation in wrapper scan chain lengths, instead of using only one data flit format, we propose to use multiple data flit formats for test data application. That is, when some wrapper scan chains finish their test applications, we will re-assign a new data flit format such that the remaining wrapper scan chains can fully use the network channel.

For example, if there are four scan chains in an embedded core of a NoC, and their lengths are 160, 160, 80, and 80 respectively. Assume each data flit can transfer 32 bits of data. By using only one fixed format of data flits to apply test patterns to the core, each data flit will include eight bits of data for each scan chain (Figure 4.1(a)). After 10 flits are transferred, both shorter scan chains are finished, and the network channel for them will be totally wasted. If variable data flit formats are allowed, after transferring 10 data flits, we can use a new flit format such that each flit carries 16 bits for each of both longer chains (Figure 4.1(b)). Thus, no network channel will be

[1] There are different definitions of a 'flit' for NoC. In this proposal, a flit means the data unit that can be sent within one clock cycle. A packet is composed of multiple flits, called head flit, body flit, and tail flit by function. 'Data flits' in this proposal are in fact the body flits that contain in-band data in stead of control information. Further, the width of a data flit indicates the effective bits in a data flit for in-band data.

wasted, and we can use fewer data flits (15 flits) than the single format option (20 flits) to finish the test data application.

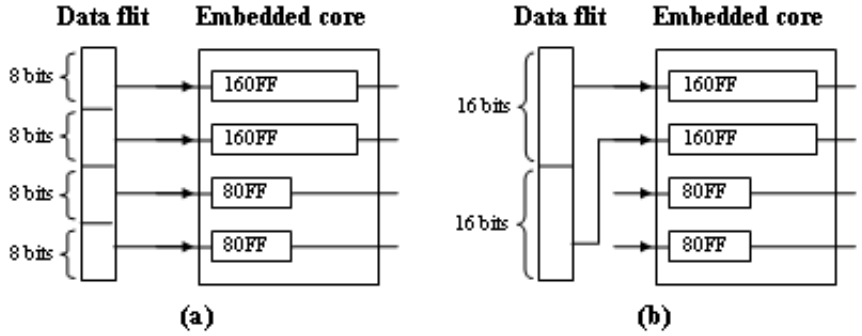


Figure 4.1: (a)Data flit format with each flit containing 8 bits/chain; (b)Data flit format with each flit containing 16 bits/chain.

Assume the width of each data flit is N, the total number of wrapper scan chains is m, and $m \leq N$. Based on the idea of multi-format test data transportation, we can calculate the total number of data flits required to finish the transfer of a test pattern as discussed below.

- At first, each flit includes data for all m chains.
  - Each chain has $x_0 = \lfloor \frac{N}{m} \rfloor$ bits where $\lfloor x \rfloor$ is the floor of x.
  - Since we have $l_0 \leq l_1 \leq ... \leq l_m$ (l_i denotes the length of wrapper scan chain i), the shortest chain with length l_0 will finish first and the number of data flits it takes is $k_0 = \lceil \frac{l_0}{x_0} \rceil$, where $\lceil y \rceil$ is the ceiling of y.
- After the shortest chain with length l_0 is finished, there are at most $m-1$ chains remaining to be filled.
  - Each chain has $x_1 = \lfloor \frac{N}{m-1} \rfloor$ bits in each flit.
  - Since we have $l_1 \leq ... \leq l_m$, the scan chain with length l_1 will finish first and this takes k_1 flits with
    $k_1 = \lceil \frac{(l_1 - k_0 * x_0)^+}{x_1} \rceil$.
    Note that $(z)^+$ returns z if z is larger than 0, otherwise, it returns 0.

- When the chain with length l_{i-1} is finished, there are at most $m-i$ chains remaining to be filled.
  - Each chain has $x_i = \lfloor \frac{N}{m-i} \rfloor$ bits in each flit.
  - To finish shifting the chain with length l_i, we need another k_i flits where $k_i = \lceil \frac{(l_i - \sum_{j=0}^{i-1} k_j * x_j)^+}{x_i} \rceil$.
- For the longest chain to finish, totally k flits are required. We have

$$\begin{aligned} K &= k_0 + ... + k_{m-1} \\ &= k_0 + \sum_{i=1}^{m-1} \lceil \frac{(l_i - \sum_{j=0}^{i-1} k_j * x_j)^+}{x_i} \rceil \quad (1). \end{aligned}$$

Consequently, it takes totally K data flits to complete the application of a single test pattern.

Let us use a core with five wrapper scan chains whose lengths equal 300, 300, 300, 120, and 90 as an example. Again, we assume the size of each data flit is 32 bits. So, we have $N = 32$ and $m = 5$ in this example. At first, each data flit will carry $\lfloor 32/5 \rfloor = 6$ bits for each chain. It takes $90/6 = 15$ flits to finish the shortest chain. After this, each data flit will carry $32/4 = 8$ bits for the remaining 4 chains. It will take $\lceil (120 - 6 * 15)/8 \rceil = 4$ flits to finish the second shortest chain. Then, each data flit will carry $\lfloor 32/3 \rfloor = 10$ bits for each remaining chain, and it will take $\lceil (300 - 8 * 4 - 6 * 15)/10 \rceil = 18$ flits to finish the 3 chains left to scan. So, the total number of data flits required to finished a test pattern application is $(15 + 4 + 18) = 37$. If only one data flit format is used, however, the total number of flits required will be $300/6 = 50$. Thus, by using 3 data flit formats, 26% of data flits can be saved in this example. This is very worthwhile if the hardware overhead for implementing multi-format data flits is tolerable.

4.2 Wrapper Scan Chain Configuration for Embedded Core Testing

4.2.1 Guidelines for Scan Chain Configuration

Assume there are two possible wrapper scan chain configurations for a given embedded core. Configuration C_1 has M_1 equal length wrapper scan chains and each chain has length L. Configuration C_2 has M_2 wrapper scan chains and the longest chain in C_2 has length equal to L. Further, assume C_1 (C_2) requires N_1 (N_2) data flits to apply a test pattern to the core. Again, N is the bit-width of each data flit.

Lemma 1: For configurations C_1 and C_2 with $N \bmod M_1 \neq 0$, $N \bmod M_2 = 0$, where M_2 is greater than M_1 and is the smallest integer factor of N between M_1 and N. We have $N_2 \leq N_1$.

Proof: Based on MDFF method, for configuration C_1, since all chains have the same length, there will be only one data flit format and the total number of data flits needed to send a test pattern is $N_1 = \lceil \frac{L}{x} \rceil$ and $x = \lfloor \frac{N}{M_1} \rfloor$. For configuration C_2, since there are different lengths of chains, we need to use different data flit formats. The data flits that contain maximum number of chains will have $x = \lfloor \frac{N}{M_1} \rfloor = \lfloor \frac{N}{M_2} \rfloor$ bits for each chain, and those contains less number of chains will have at least equal or more bits for each wrapper scan chain. Obviously, the total number of data flits needed for C_2 should be less or equal to that for C_1. **Q.E.D**

For example, let us consider an embedded core with two wrapper scan chain configurations, C_1 and C_2. Here, C_1 has 17 chains with each equal to 160 bits. C_2 has 16 chains with each equal to 160 bits, and another set of 16 chains with each equal to 10 bits. Assume each data flit can carry 32 bits of data. The number of data flits required in the first configuration is $160/1 = 160$, while that in the second configuration is $10/1 + (160 - 10)/2 = 10 + 75 = 85$. The number of data flits saved by the second configuration is $(160 - 85)/160 = 46.875\%$.

The reason behind this lemma is that, under equal length wrapper scan chain configuration, if the number of (final) wrapper scan chains is not an integer factor of the network channel width, there is fixed waste for each data flit, which is equal to $N - \lfloor \frac{N}{M_1} \rfloor * M_1 = N \bmod M_1$. Different values of M_1 result in different values of flit waste, and the relationship between the waste of data flits and the number of wrapper scan chains can be shown in Figure 4.2 (we still assume that each

data flit carries 32 bits of data).

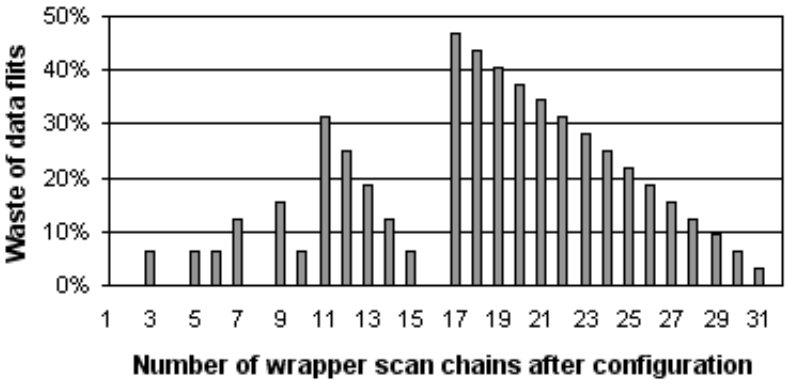


Figure 4.2: Relation between the waste of data flits and the number of wrapper scan chains.

So, in this case, the idea of equal length configuration is no longer the optimal solution as in the case of traditional SoC testing, instead, reconfiguring all scan chains in a core to M_2 wrapper scan chains may achieve some improvement. The best improvement can be estimated by the value of M_1, and it occurs when $M_1 = \frac{N}{2} + 1$, and the improvement can be up to $\frac{(N/2-1)}{N}$, as shown in the above example.

Assume l_0 and l'_0 are the shortest chains in C_1 and C_2 respectively, and l_i (l'_i) is the length of the i_{th} wrapper scan chain in C_1 (C_2).

Lemma 2: If C_1 and C_2 have the same number of wrapper scan chains (i.e., $M_1 = M_2 = M$, and N mod $M = 0$), and $l_0 \leq l'_0$, $l_i = l'_i$ for all $M/2 \leq i < M - 1$, then we have $N2 \leq N1$. C_2 requires the minimum number of data flits when $l_0 = l_1 = ... = l_{M/2-1}$.

Proof: Since C_1 and C_2 are configurations for the same core, and $l_i = l'_i$ for all $M/2 \leq i \leq M - 1$, we have $\sum_{i=0}^{M-1} l_i = \sum_{i=0}^{M-1} l'_i$, $\sum_{i=M/2}^{M-1} l_i = \sum_{i=M/2}^{M-1} l'_i$, and $\sum_{i=0}^{M/2-1} l_i = \sum_{i=0}^{M/2-1} l'_i$. Further, since $l_0 \leq l'_0$, we have $\sum_{i=1}^{M/2-1} l_i \geq \sum_{i=1}^{M/2-1} l'_i$ Since N mode $M = 0$, $M/2$ is the smallest integer factor of N among those are bigger than N. So, data flits to finish l_0 and $l_{M/2}$ to l_{m-1} are non-waste flits and data flits to finish l_1 to $l_{M/2} - 1$ are flits with waste. l_0 is less than l_0 means it takes more flits to finish l_0, thus more data are transferred using non-waste flits in C_2. Obviously, total flits to apply the same test pattern for C_1 should be no less than that for C_2. To use minimum data flits, l_0 should be as long as possible, so the configuration with minimum data flits is the one with $l_0 = l_1 = ... = l_{M/2-1}$. **Q.E.D**

For example, consider three different configurations C_1, C_2 and C_3 with $M = 4$, $N = 32$. The lengths of all wrapper scan chains in C_1 are 1600, 1600, 1200, and 400, respectively. Those in C_2 are 1600, 1600, 1000, and 600, respectively. Finally, we have the wrapper scan chain lengths in C_3 equal 1600, 1600, 800, and 800. The number of data flits required to apply a test patter by C_1 can be derived by $400/8 + (1200 - 400)/10 + (1600 - 1200)/16 = 50 + 80 + 25 = 155$. The number of data flits required by C_2 is $600/8 + (1000 - 600)/10 + \lceil(1600 - 1000)/16\rceil = 75 + 40 + 38 = 153$, and that by C_3 is $800/8 + (1600 - 800)/16 = 100 + 50 = 150$. Obviously, C_3 (C_1) requires the smallest (largest) number of data flits.

It can be observed from Lemma 2 that an embedded core will require a smaller number of data flits if: (1) the embedded core can be configured as S_1 long chains with equal length (S_1 is an integer factor of N) plus S_2 shorter chains ($S_1 + S_2$ is the minimal integer factor of N that is greater than S_1), and (2) the length of the shortest chain can be maximized. The best case occurs when the shorter S_2 chains can be configured to be of equal length.

Combining Lemma1 and Lemma2 and extending them to a general case, we can find that: the optimum configuration should result in *factorial wrapper scan chain groups* (FSCG) with balanced-length chains within each group, as shown in Figure 4.3. Each group $FSCG_{i+1}$ is composed of sorted wrapper scan chains with index $F_i + 1$ to F_{i+1}, where F_i and F_{i+1} are consecutive integer factors of N. For example, when $N = 32$, we have $F_1 = 2$, $F_2 = 4$, $F_3 = 8$, $F_4 = 16$, and $F_5 = 32$. We assign $F_0 = 0$ by default. Thus, the wrapper scan chains in $FSCG_1$ is indexed from $F_0 + 1$ to F_1, which is from wrapper scan chain 1 to wrapper scan chain 2 (Figure 4.3). The maximum wrapper scan chain length of group $FSCG_{i+1}$ is no greater than that of group $FSCG_i$. The reason for this configuration to be optimum is evident. With this configuration, there will always be y wrapper scan chains in a data flit where y is an integer factor of N, so there will be no channel loss for test data transportation.

4.2.2 A Heuristic Scan Chain Configuration Algorithm

In this section, we will introduce a heuristic wrapper scan chain configuration algorithm. The objective is to minimize the total number of data flits to apply a test pattern. Wrapper scan chain configuration has to obey a length constraint, i.e., the maximum length of all wrapper scan chains cannot be larger than the length of the longest internal scan chain of the core. The

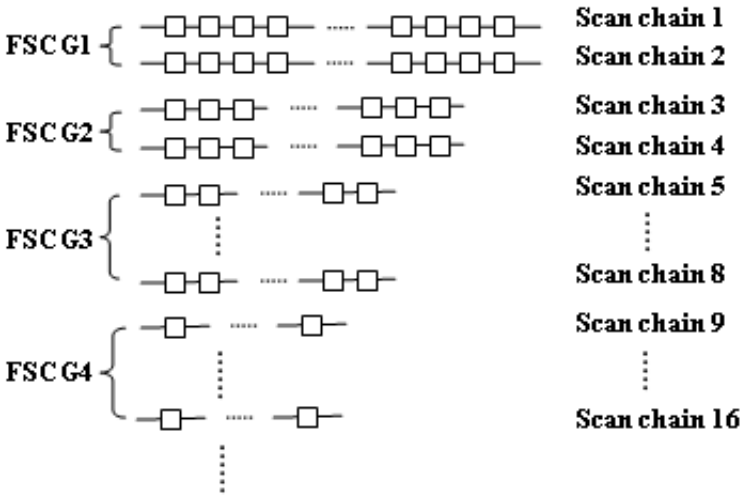


Figure 4.3: Factorial Scan Chain Groups with N=32.

performance measure of a configuration is the ratio of waste in data flits, which is given by $W = (K - K_{ideal})/K_{ideal}$. Note that K is the minimum number of data flits required to transfer a test pattern based on a specific configuration, and can be calculated using equation (1). Further, K_{ideal} is the theoretical minimum number of data flits to apply a test pattern, and can be calculated using equation (2),

$$K_{ideal} = \lceil \frac{\sum_{i=0}^{m-1} l_i + N_{io}}{N} \rceil \quad (2)$$

where l_i is the length of each internal scan chain, m is the total number of internal scan chains, N_{io} is the number of all functional I/Os, and N is the width of the network channel, i.e., the bit-width of each data flit.

Based on the MDFF test data transfer protocol and two lemmas introduced above, three guidelines are used during the wrapper scan chain configuration process:

1. Try to configure the internal scan chains and functional I/Os into M wrapper scan chains where M is an integer factor of N, since there will be no channel loss in this case (refer to lemma 1).

2. When wrapper scan chains are sorted by length, try to balance the chains within each FSCG group (refer to lemma 2).

3. Attach as many functional I/Os together as possible, since this will result in less hardware complexity.

Three configuration strategies are used in the algorithm:

- Equal-length configuration —- Assign internal scan chains or functional I/Os to wrapper scan chains and try to balance the lengths of all wrapper scan chains. This is based on guideline 2.
- Shortest-first configuration — Assign functional I/Os to existing wrapper scan chains with the shortest wrapper scan chain first. This is based on guideline 3.
- Best-fit configuration — Assign internal scan chains or functional I/Os to wrapper scan chains to allow more chains with the maximum length. This will also help to reduce the hardware complexity for configuration.

The wrapper scan chain configuration algorithm is given below.

Step 1:

- Sort all internal scan chains by length in a decreasing order.
- Apply the best-fit strategy to all internal scan chains and get M wrapper scan chains.

Step 2:

- Find the maximum integer M_1 which is no greater than M, and is an integer factor of N.
- Divide the longest M_1 wrapper scan chains into FSCG groups. If within any group, there is a chain whose length difference with the maximum length in the same group is larger than a given limit δ, fill it to the maximum length of the group using functional I/Os by the shortest-first configuration strategy. If there is no I/O left, stop. (note: since connecting a very small number of I/Os to wrapper scan chains will result in hardware complexity, if not necessary, we only connect I/Os to wrapper scan chains when more than δ I/Os can be grouped together.)

- If $M_1 = M$, and all remaining I/Os can be used to balance existing wrapper scan chains within each FSCG group without breaking the length constrain, do it using the shortest-first configuration strategy and stop. Otherwise, continue to step 3.

Step 3:

- Find a minimum integer M_2 which is larger than M and is an integer factor of N. We configure the shortest $M - M_1$ wrapper scan chains and all remaining functional I/Os into a new FSCG group with $M_2 - M_1$ wrapper scan chains, with a constraint that the maximum wrapper scan chain length of this new FSCG group is no larger than that of the closest FSCG group.

- If not all chains in this new FSCG group can be filled to the group's maximum length, try two options: 1) the equal-length configuration strategy and 2) the best-fit configuration strategy, and choose one of them based on the measure of data flit waste and the configuration complexity.

- Otherwise, fill all wrapper scan chains of this FSCG group to the group's maximum length. For extra I/Os, if they can be used to balance wrapper scan chains within each FSCG group without breaking the length constrain, do it using the shortest-first strategy. Otherwise, configure them to new FSCG groups by following the same method as step 3.

Let us use an example to show the procedure of the proposed algorithm. Given a core with 6 internal chains where the chain lengths are 400, 398, 250, 200, 190, and 150 respectively. There are also totally 168 functional I/Os around this core. The length constrains is that the longest wrapper scan chain length cannot exceed 400. Assume each data flit can contain 32 bits of data (i.e., $N = 32$). Ideally, the minimum number of data flits required to apply one test pattern is $\lceil (400 + 398 + 250 + 200 + 190 + 150 + 168)/32 \rceil = 55$.

Now, we begin to do wrapper scan chain configuration. At step 1, six internal scan chains are first sorted and then configured using the best-fit strategy with the length constraint. The longest two scan chains are each assigned to a wrapper scan chain, the third one (250) and the shortest one (150) are connected together and assigned to one wrapper scan chain, while the remaining

two chains are also connected and assigned to another wrapper scan chain. So, now we have four wrapper scan chains with lengths: 400, 400, 398, and 390 respectively.

Assume $\delta = 5$, and we continue to step 2. In this example, the maximum integer factor of N (32) which is no larger than M (4) is 4, i.e., $M_1 = 4$. So, we first divide these four chains to two FSCG groups and try to balance chains within each group. The first FSCG group has two chains with full length, so nothing can be done further. For the second FSCG group, the difference between the shorter one and the longer one is 8, which is larger than δ, so we will assign eight I/Os to the shorter one and fill it to length 398 (based on the shortest-first strategy). Even we fill all these four chains to the maximum length, there are still I/Os left. So, we continue to step 3 without any more configuration to these four chains.

The minimum integer factor of N which is larger than M is 8 in this example, i.e., $M_2 = 8$. Now, we need to assign the remaining 160 (168-8) I/Os to a new FSCG group with 4 ($M_2 - M_1$) new wrapper scan chains. Try 2 options. With option 1, all 160 I/Os are assigned to 4 chains each with length 40. By option 2, all 160 I/Os are assigned to a single wrapper scan chain. Let us check the performance measure. The number of data flits required to apply a single test pattern with option 1 is $40/4 + (400 - 40)/8 = 55$. The total number of data flits required by option 2 is $\lceil 160/6 \rceil + \lceil \frac{400-6*\lceil 160/6 \rceil}{8} \rceil = 57$. Although the second option may result in less hardware configuration complexity, the waste of data flits by option 1 is smaller than that by option 2. So we will choose option 1 as the optimal configuration.

In the above algorithm, we only considered the case where $M \leq N$. If $M > N$, we can divide the wrapper scan chains to 2 parts. Firstly, try to fill the longest N chains to maximum length, then recursively apply the algorithm to configure the remaining $N-M$ wrapper scan chains.

For simplicity, the above algorithm and example only show the configuration of internal scan chains with functional inputs or outputs. For a core with internal scan chains, functional inputs, outputs, and bi-directional I/Os, we need to make a small adjustment to the algorithm. The first step (internal scan chain configuration) is exactly the same. Then, we fill the bi-directional I/Os to existing wrapper scan chains using the shortest-first configuration strategy, and assign the remaining bi-directional I/Os to new wrapper scan chains using the best-fit configuration strategy. After this, we continue step 2 and step 3 of the heuristic algorithm to configure the current existing wrapper scan chains and pure inputs. The same thing is done to configure pure outputs separately.

In the final configuration, all pure inputs are placed at the scan-in part of a wrapper scan chain, and all pure outputs are placed at the scan-out part of a wrapper scan chain, as shown in Figure 4.4.

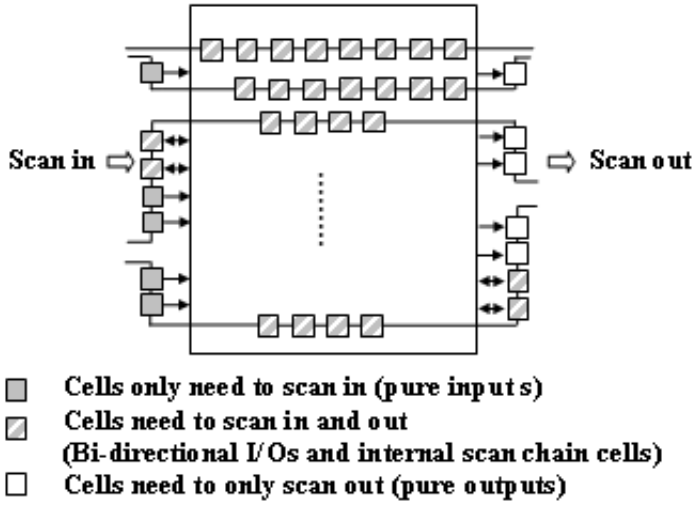


Figure 4.4: Wrapper scan chain configuration.

4.2.3 Experimental Result

We have applied our heuristic configuration algorithm to 40 cores from ITC'02 SoC benchmark set [46]. Since traditional SoC test wrapper optimization algorithms prefer to configure internal scan chains and functional I/Os to balanced wrapper scan chains, we also applied this equal-length configuration strategy to the benchmark set for comparison. In this algorithm (used for comparison), we first sort the internal scan chains, and then use the best-fit strategy to configure them into wrapper scan chains with the length constraint. After that, we use the best-fit strategy to assign functional I/Os to fill existing wrapper scan chains to maximum length, without any δ limit as in our heuristic algorithm. If there are extra I/Os left, we further use the best-fit strategy to assign them to new wrapper scan chains with the length constraint. The results obtained by these two configuration algorithms are shown in Figure 4.5.

As we can see, among 40 benchmark cores, 20 of them can be solved by using the simple equal-length configuration method without any waste, but the other 20 of them have significant waste. For the 20 remaining cores, our configuration algorithm can reduce the data flit waste to 0

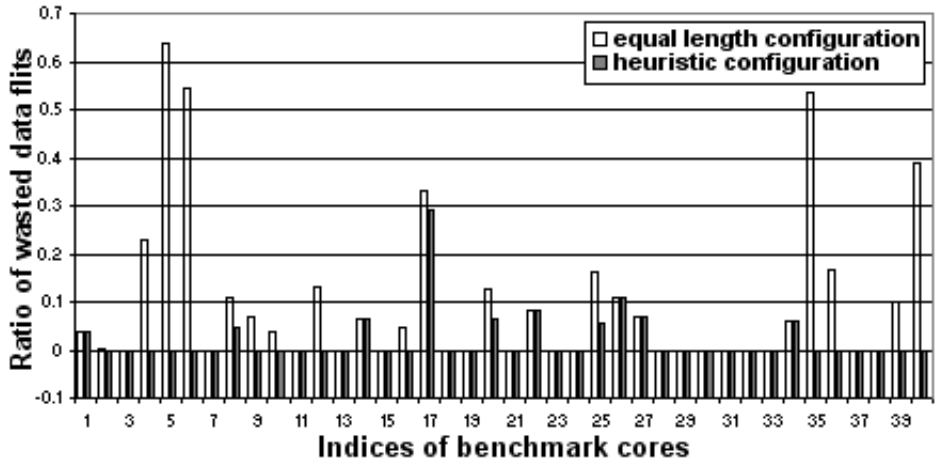


Figure 4.5: Comparison of data flit waste between equal length configuration and our algorithm.

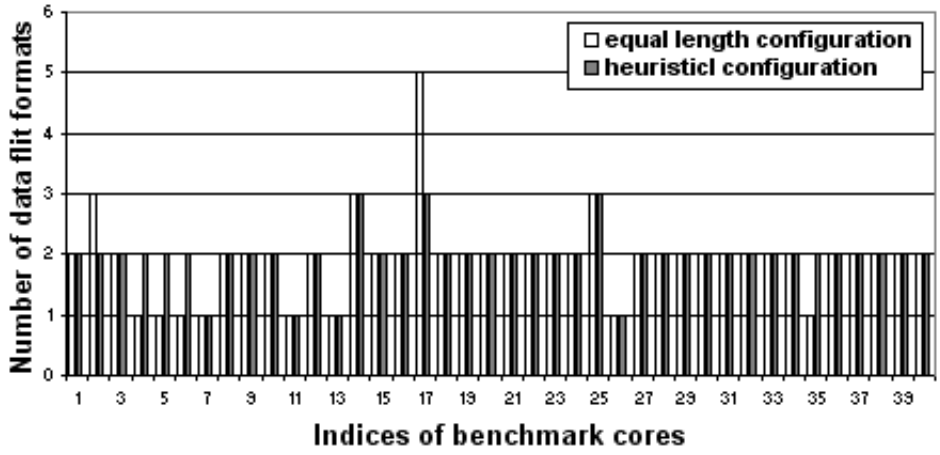


Figure 4.6: Number of data flit formats.

for 14 cores, and have slight improvement for the other 6 cores, as shown in Figure 4.5. For the cores that have significant improvement using our configuration, only 2 or 3 data flit formats are required (as shown in Figure 4.6), so the hardware overhead is quite small.

For a core with special internal scan chain structure, there is very little space for any configuration method to reduce its data flit waste without increasing the test time of the core. But, when testing a NoC-based system, although the test time for a single core is increased, the total test time for the system can also be reduced using a good scheduling method based on the concept of MDFF to interleave the test data application of different cores and test them in parallel. The

details of parallel core testing based on MDFF test data application protocol are presented in next section.

4.3 Test Wrapper Architecture for MDFF Test Data Application

The test wrapper architecture for MDFF is slightly more complex than that for a single data flit format application. The sketch of the architecture is shown in Figure 4.14. The test wrapper is located between the Network Interface(NI) unit and the embedded core as shown in Figure 4.14(a). In normal mode, data from the network will firstly pass the network interface (for synchronization, error check and etc.), go through the decoder and finally reach the I/O pins of the embedded core. In test mode, data from network interface will directly go to the test wrapper (instead of the decoder), and then scanned into the wrapper scan chains of the embedded core.

The test wrapper shown in Figure 4.14(b) is composed of a load-shift register, a test pattern distributor, a controller and several counters. The load-shift register will load a data flit from the network (when the control signal 'load' equals '1'), and will operate as a shift register to pass different bits to the test pattern distributor (when control signal 'shift' equals '1'). The test pattern distributor is used to select different bits from a data flit to the wrapper scan chains of the embedded core. The controller receives input signals from NI and counters, and sets control signals to load-shift-registers, test pattern distributor, counters and embedded core for testing.

This architecture is suitable for test data transportation where each data flit contains one or more bit of information for each wrapper scan chain. The only difference between single data flit format and multiple data flit formats is the number of counters and internal complexity of the controller. As shown in our experimental results, using 2 or 3 data flit formats can significantly reduce the waste of data flits in most cases. The small number of data flits formats ensures tolerable hardware overhead. The functional correctness of this architecture has been verified using VHDL simulation.

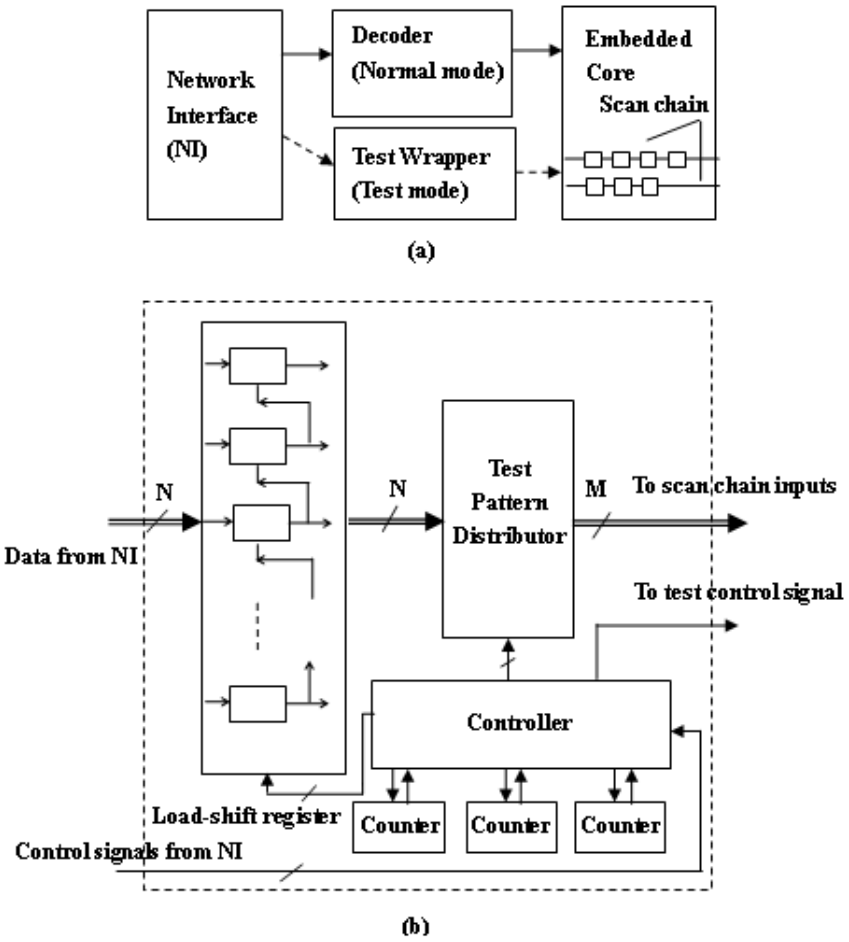


Figure 4.7: (a)Location of the test wrapper; (b)Test wrapper architecture.

Chapter 5

Test Scheduling for Embedded Core Testing

In Chapter 4, we have proposed a MDFF test data application concept and developed a heuristic wrapper scan chain configuration algorithm based. The wrapper chain configuration method with MDFF can reduce the waste of data flits for testing cores in a NoC, and formed a good foundation for efficient embedded core testing. However, an appropriate test scheduling method is needed to realize the reduction of total test application time of NoCs.

In this paper, a dedicated test scheduling algorithm is proposed to work together with the wrapper scan chain configuration method and MDFF concept, which can realize the test pattern application interleaving for different cores, and testing of different cores of a NoC in parallel. Instead of considering channel capacity only as most traditional methods, the proposed test scheduling method takes into account the data flit interleaving issues, such that flits (of test patterns or test responses) for different cores can be well interleaved without conflicts in channels and routers of the NoC. Therefore, test time of the whole system can be minimized with a specified test power consumption. By comparing the results with other published works, it has been demonstrated that the proposed test scheduling method can achieve significant improvement on the test time for the entire NoC. An optimal test scheduling algorithm is also developed based on MDFF, which can achieve the maximum utilization of the test resources and reduce the test time for the entire NoC.

5.1 Basic Concept

With the MDFF test data application method, each data flit can contain multiple bits for a single wrapper scan chain, and the data flits for a core can have different formats. Test scheduling makes use of the flexibility of MDFF to interleave the test data application of different cores through the on-chip network and test the cores in parallel. Assume the clock frequency of the on-chip network is the same as that of each embedded core. The basic concept is: for a data flit containing multiple bits (x bits) of each wrapper scan chain, we only need to send one such a flit to the core every x clock cycles, and the other $(x-1)$ cycles during the x-cycle period can be utilized to send data flits to other cores. A NoC may also contain multiple clock domains. In this case, the on-chip network and embedded cores may work under different clock frequencies, and all cores can also work under different clock frequencies. The idea of the proposed test scheduling method can be easily extended to a NoC with multiple clock domains, and the details will be explained in Sub-section 5.2.

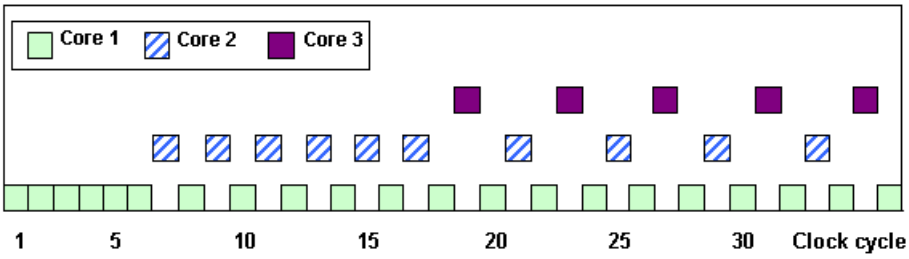


Figure 5.1: Test scheduling based on MDFF for 3 example cores.

Let us look at an example. Assume there are three cores in a NoC-based system, and all of them need the same resource (I/O ports and network channels) for testing. After configuration, core 1 has 16 wrapper scan chains with length equal to 55, and the other 16 chains with length equal to 5. Core 2 has 8 chains with length 50 and the other 8 chains with length 10. Core 3 has 8 chains with length 40. Based on the MDFF method, core 1 needs two data-flit formats. Each of the first 5 data flits contains 1 bit for each of all 32 chains, while each of the following 25 data flits contains 2 bits for each of the longer 16 chains. Core 2 also needs two formats of data flits. Each of the first 5 flits contains 2 bits for each of all 16 chains, and the following 10 flits each contains 4 bits for each of the longer 8 chains. Core 3 needs 10 flits each containing 4 bits for every chain.

To efficiently use the network channels, we schedule to send test data to core 1 first. It will take 5 clock cycles to finish the first 5 flits as shown in Figure 5.1. Then, we begin to send data flits to the remaining 16 chains of core 1 using the second format. Since the data flits of the second format only need to be sent every 2 clock cycles, we schedule to send the first data flit set of core 2 to fill the empty time slots (Figure 6). After 10 clock cycles, the first data flit set of core 2 is also finished. Since the data flits of core 2 with the second format only need to be sent every 4 clock cycles, there are more empty time slots available. So, we schedule to send test data to core 3 using the empty time slots. As we can see from Figure 5.1, the network channels are fully utilized.

5.2 Power-Aware Test Scheduling Algorithm

5.2.1 The Algorithm

The main advantage of on-chip network reuse is the availability of parallel accesses to each core, depending on the number of the interface I/Os available during test. Therefore, the reduction in system test time is highly related to test parallelization. However, test parallelization implies more power consumption, since more network structures are active simultaneously. It is necessary to consider power consumption during testing to ensure the feasibility of a test plan. The power-aware test scheduling algorithm is shown below.

Notation:

C_i: an embedded core in a NoC under test.

Rs_i: an I/O resource for testing.

L_i: the test time of core C_i.

T_{Rs_i}: the starting time for resource Rs_i to become available.

T_{C_i}: the scheduled starting test time of core C_i.

$U\{Rs_i\}$: the utilization rate of I/O resource Rs_i and related routing channels.

$U\{C_i\}$: the I/O resource utilization rate of core C_i.

IO_found: the flag to indicate whether an I/O resource is found for a core.

Procedure path_finding:

1. Find a XY path from the input of test resource Rs_j to core C_i and the other XY path from C_i to the output of Rs_j;

2. Check the availability of all the channels on the routing path

/* check whether $U\{Rs_j\}$+$U\{C_i\}$ is larger than 1 */

3. If the XY paths are blocked

4. {

5. Find the blocked segment closest to the source of the path;

6. Group all routers on the XY path before this blocked segment in a set in order;

7. While ((the shortest path is not found) and (not all routers have been tried))

8. {

9. Find the last router in the set;

10. Find all available routing paths from this router to the destination of the path without exceeding the maximum allowed path length;

11. Remove this router from the set;

12. }

13. If one or more paths can be found, select a shortest one and return $True$;

14. Otherwise, return $False$;

15. }

16. Otherwise, mark the XY routing paths and return $True$. End of $path\_finding$.

Procedure NoC_power_aware_scheduling:

1. Calculate the test time L_i and I/O resource utilization rate $U\{C_i\}$ for each core C_i;

2. Sort cores in decreasing order of L_i;

3. Set $U\{Rs_i\}$=0, $T_{Rs_i} = 0$ and $T_{C_i} = infinity$;

4. While (there are unscheduled cores)

5. {

6. For (each unscheduled core C_i) /*with the one with the highest test time first */

7. {

8. Set $IO\_found = False$;

9. While ($IO\_found = False$ and there exists an untried I/O resource Rs_a with $U\{Rs_a\} < 1$)

10. {

11. Find the next resource Rs_j with smallest T_{Rs_j};

12. Call path_finding();

13. If (path_finding() = $True$)

14. {

15. Calculate the sum of power for all cores scheduled to test concurrently;

16. If (Power constraints are met)
17. {
18. Assign resource Rs_j to core C_i;
19. Update T_{C_i} and $U\{Rs_j\}$;
20. Set $IO\_found = True$;
21. }
22. }
23. }
24. }
25. Update T_{Rs_k} and $U\{Rs_k\}$ for each resource Rs_k ;
26. }
27. For (each core C_i)
28. {
29. If (C_i shares test I/O resource with any other cores)
29. {
29. Check C_i with Rule 1;
30. Add appropriate delay logics to C_i if Rule 1 cannot be met;
31. Check Rule 2 for C_i and each C_j which uses the same test I/Os but some different routers with C_i;
32. Insert appropriate delay logics if Rule 2 cannot be met;
33. }
34. }

We note here that Rule 1 and Rule 2 in the algorithm are two rules to guarantee data flits interleaving between cores using the same test I/O resource. The details of data interleaving issues will be discussed in Sub-section 5.3.

An example NoC under test is shown in Figure 5.2 where there are totally 16 embedded cores and 3 pairs of I/O ports for test pattern application. An example to illustrate the test scheduling algorithm is shown in Figure 5.3. In the beginning, all I/O resources are available. Core C_1 has the longest test time L_1, so it is scheduled to resource Rs_1. T_{C_1} is set to 0 and the utilization rate $U\{Rs_1\}$ is updated to $U\{C_1\}$, which is equal to 1. Core C_3 is the next to be scheduled since its test time L_3 is the longest among all remaining cores. Since Rs_1 is full at this time, Rs_2 is assigned to C_3. T_{C_3} is set to 0 and $U\{Rs_2\}$ is updated to $U\{C_3\}$, which is equal to 1/2. Similarly, core

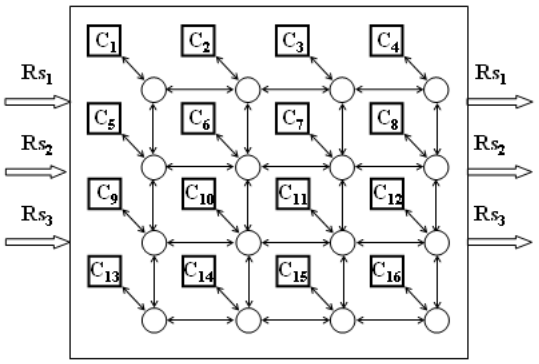


Figure 5.2: An example of NoC under test.

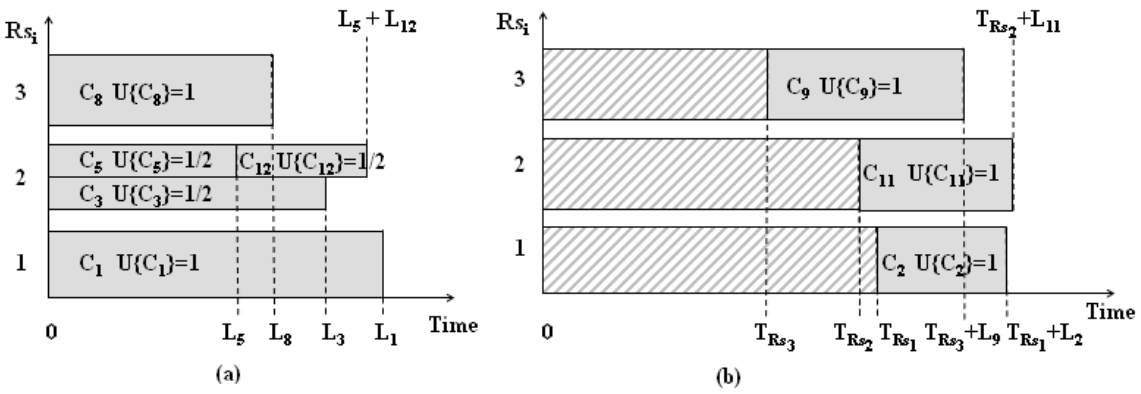


Figure 5.3: An illustration of the test scheduling algorithm.

C_8 is scheduled to Rs_3 with T_{C_8}=0 and $U\{Rs_3\}$=$U\{C_8\}$=1. Now, C_5 is to be scheduled. Since both $U\{Rs_2\}$ and $U\{C_5\}$ equal to 1/2, C_5 is scheduled to Rs_2. Similarly, C_{12} is also scheduled to Rs_2. These processes are shown in Figure 5.3(a). Since all remaining cores have utilization rates equal to 1, no core can be scheduled under the current condition. Therefore, we update $T_{Rs_1} = L_1$, $T_{Rs_2} = L_5 + L_{12}$, $T_{Rs_3} = L_8$, and set the utilization rates of all resources to 0 (line 25 of the algorithm), and then begin a new iteration as shown in Figure 5.3(b).

5.2.2 Path Finding Issue

We emphasize here that $U\{Rs_i\}$ means the utilization rate of the test I/O resource Rs_i and the related routing channels. To simplify the description, we did not mention the details of the path finding process in the above example. Basically, we apply XY routing to transfer test

patterns and test results. However, XY routing may be too restrictive to fully utilize the routing resources, and thus the total test time cannot be minimized. Therefore, when a XY routing path is not available, we allow to use other paths with small additional path length to efficiently use the network resources and reduce the total test time. An example of XY path blocking and alternative path finding is shown in Figure 5.4.

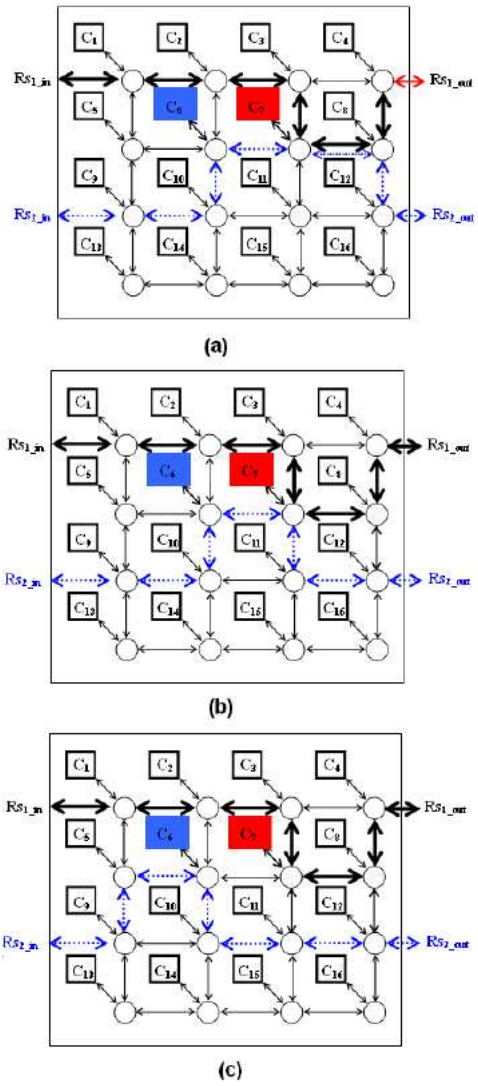


Figure 5.4: An example of routing path finding.

In this example, there are two pairs of test I/O resources Rs_{1in}/Rs_{1out} and Rs_{2in}/Rs_{2out} for a 4×4 NoC. Assume that Rs_{1in}/Rs_{1out} is assigned to test core C_7. Based on XY routing algorithm, the path for test pattern input and test result output is marked in Figure 5.4(a) using

the dark lines. Assume C_6 is the next core to be tested, and it cannot use Rs_{1in}/Rs_{1out} since $U\{Rs_1\}+U\{C_6\} > 1$. So, we assign Rs_{2in}/Rs_{2out} to C_6. The XY path is marked using the dashed lines in Figure 5.4(a). If the capacity of channel between the local router of C_7 (R_7) and that of C_8 cannot afford to transfer data flits of C_6, C_6's XY path is blocked. In this case, we find the starting router of this blocked channel, which is R_7, and check whether there is any other free path from here to the destination. A minimum routing path is then found and marked using dashed lines in Figure 5.4(b). Using this new routing path, C_7 and C_6 can be tested in parallel. To implement this routing configuration, only one routing item is needed to be inserted in the routing table of R_7, to enforce R_7 to forward the test data flits of C_6 to the south direction, and all other routers can keep working with normal XY routing algorithm. We note here, in this example, we assume that a router is able to process multiple data flits simultaneously. If this assumption cannot be met, the assigned paths for C_6 and C_7 will also block at R_7. In this case, another path (as shown in Figure 5.4(c)) should be used for C_6. To implement this new routing path, two special routing items are needed to be inserted in the routing tables of R_9 and R_6 separately, to enforce the test data flits of C_6 to be routed along the right direction. Details of path finding and checking process are described in procedure *path_finding()*.

5.2.3 Power Calculation

If there is a power constraint for test, before scheduling a core to an I/O resource, we also need to check the total power first. The power check process is simple. Assume each core C_i has test power P_{C_i}, each router Rs_i consumes power P_R, each interconnect segment consumes power P_{inc}, and the overall test power of the entire NoC cannot exceed a limit P_{max}. Assume that we plan to assign core C_j to an I/O resource and start test from time t. Before this assignment, all cores that have been scheduled and whose test periods have overlaps with that of C_j will be identified. The sum of test power for C_j, test power for all the identified cores, and power for the involved routers and interconnect segments will be calculated and compared with P_{max} to check whether the power constraint can be satisfied. For example, in Figure 5.3(a), before assigning C_{12} to Rs_2, we need to calculate the sum of test power for C_1, C_3, C_{12}, C_8, and involved routers and interconnects, and compare the result with the maximum test power limit.

5.2.4 Multi-Clock Domain Application

The algorithm is applicable to both single-clock-domain NoCs and multi-clock-domain NoCs. The only difference of these two conditions lies in the calculation of test time (L_i), network utilization rate $U\{C_i\}$ and test power for a core C_i. Assume that the on-chip network clock frequency is f, and the clock frequency of an embedded core C_i is $k_i \times f$. In the single-clock-domain case, all k_i's are equal to 1. However, in the multi-clock-domain case, the values of k_i's can be larger than 1, less than 1, or equal to 1. If each test data flit of core C_i contains x bits of test pattern for each wrapper scan chain, one test data flit of core C_i needs to be sent every x/k_i clock cycles, thus $U\{C_i\} = k_i/x$. And, the test time L_i should be updated to $1/k_i$ times the value of the single-clock-domain case. The test power of C_i should also be updated to k_i times the value in single-clock-domain case. We note here that, no matter how large k_i can be, $U\{C_i\}$ is a value no larger than 1. So, if k_i/x is larger than 1, an on-chip slower-clock generation circuit is needed to adjust $U\{C_i\}$ to a value which is at most 1. The test time and test power of C_i should be adjusted correspondingly.

5.2.5 Test Clock Frequency

Sometimes, among all embedded cores in a NoC, there exist cores whose required test times are much longer than those of others. These cores become the bottleneck of the entire NoC test process, and only little improvement can be achieved by test scheduling or adding additional I/O ports. For example, in benchmark circuit g1023 (a benchmark circuit in ITC'02 benchmark set), assuming the single-clock-domain case, core 4 has the longest test time which is 14794 clock cycles, while the second longest test time is only 6374 clock cycles. After applying the wrapper scan chain configuration algorithm and the test scheduling method, the test time for the entire circuit is 14794 clock cycles with no power constraint, when two pairs of I/O ports are available for testing. When we increase the I/O ports to 3, the test time for the entire chip is still 14794, and the same result remains even when we increase the number of I/O ports further. The reason is that the wrapper scan chain configuration and test scheduling methods can only help to fully utilize the network channels, and thus allow more parallelisms by testing different cores simultaneously. However, the test time for a single core is fixed by the internal scan chain length which cannot be modified.

If the channel utilization rates of cores ($U\{C_i\}$'s) are less than 1, we can keep the test frequency of the network while speeding up the test clock frequencies of the corresponding cores to reduce their test times. However, this solution will add some hardware overhead (e.g., on-chip clock generation circuit), and will consume more power for the corresponding cores. Therefore, only very limited number of cores can be chosen to perform this optimization.

In our test scheduling algorithm, when this problem occurs, we identify the core with the longest test time and channel utilization rate less than 1, and increase its test clock frequency to twice of the original frequency. The test time of this core is reduced to half, and its channel utilization rate and test power are both doubled. Then, the whole scheduling process is performed with this modification. If the test time for the entire NoC is still too long, we find the next core with the longest test time and utilization rate less than 1, and then repeat the same clock rate doubling process for the core. This process is repeated until the test time for the entire NoC satisfies the requirement, or the maximal number of cores allowed to accelerate frequencies has been attempted. For g1023, by doubling the test clock frequency of core 4 and by applying the wrapper scan chain configuration and test scheduling methods, the test time for the entire chip is reduced from 14794 to 12441 clock cycles with no power constraint, when two pairs of I/O ports are available. When increasing the number of I/O ports to 3, the test time for the entire chip becomes 7626 clock cycles, and it becomes 7397 clock cycles when the number of I/O ports is increased to 4.

5.3 Data Flits Interleaving Issues

The basic idea of our test scheduling method is to interleave data flits from different cores for parallel test. In the test scheduling algorithm presented above, we use condition $U\{Rs_j\}+U\{C_i\} \leq 1$ to check whether the channel capacity of test I/O resource Rs_j can further afford to transfer data flits for core C_i. This guarantees that the channel capacity of each test I/O resource can support parallel test of multiple cores assigned to it. However, this is just the basic requirement. Of course, cores using different I/O resources can be tested in parallel without any question. However, for cores using the same test channel, their data flits have to be aligned well to avoid any transfer conflict. Data flits can come from both test patterns and test results. Also, there may exist latency differences between flits for different cores, if they traverse the network through different

paths. Therefore, even flits are well scheduled to be interleaved at the starting point, it is not guaranteed that they can keep interleaving without any conflict in the following channels and routers. Instead, some rules have to be satisfied to guarantee the interleaving throughout the entire flit transfer process. In this sub-section, we analyze this question and derive the rules from two aspects: interleaving between pattern and result flits, and interleaving between flits with different paths.

5.3.1 Interleaving between pattern and result flits

Assume each data flit of core C_i contain X_i bits for each wrapper scan chain. The time for a data flit to pass a router is T_R cycles, and the time for a data flit to pass the interconnect between two adjacent routers is T_{inc1} cycles (Figure 5.5). Further, assume the time for a data flit to transfer from a router to its local core is T_{inc2} cycles, and the time for a core to perform testing after receiving a full test pattern is T_{test} cycles.

Rule 1: If the test pattern flits of C_i can be well interleaved with data flits for other cores sharing exactly the same test I/O and routing channels, to guarantee the test result flits of C_i to continue interleaving with other data flits without any conflict, there must exist an integer k_i to satisfy the equation: $T_R + T_{test} + 2 \cdot T_{inc2} = k_i \cdot X_i$.

We will use an example shown in Figure 5.5 to derive this rule. In this example, each data flit of core C_1 (C_2) contain X_1 (X_2) bits for each wrapper scan chain, i.e., the channel utilization rate of C_1 (C_2) is $1/X_1$ ($1/X_2$). Assume $(1/X_1)+(1/X_2)$ is less than 1, and C_1 and C_2 are scheduled to be tested in parallel using the same test I/O resource and routing channels. Therefore, the test pattern flits and result flits of C_1 and C_2 should be well interleaved to be transferred through the routing channels. Let F_{1i} represent the i_{th} pattern flit of C_1, F_{2j} represent the j_{th} pattern flit of C_2, F'_{1m} represent the m_{th} result flit of C_1, and F'_{2n} represent the n_{th} result flit of C_2. The number of data flits for a single test pattern of C_1 (C_2) is N_1 (N_2). Assume the first pattern flit of C_1 (C_2) reaches the local router of C_1 (called R_1) at time T_1 (T_2). Since each data flit of C_1 (C_2) contains X_1 (X_2) bits for each wrapper scan chain, the time interval between two consecutive data flits of C_1 (C_2) should be X_1 (X_2) cycles. So, F_{1i} (F_{2j}) reaches R_1 at time $T_{F_{1i}\_R_1} = T_1 + i \cdot X_1$ ($T_{F_{2j}\_R_1} = T_2 + j \cdot X_2$).

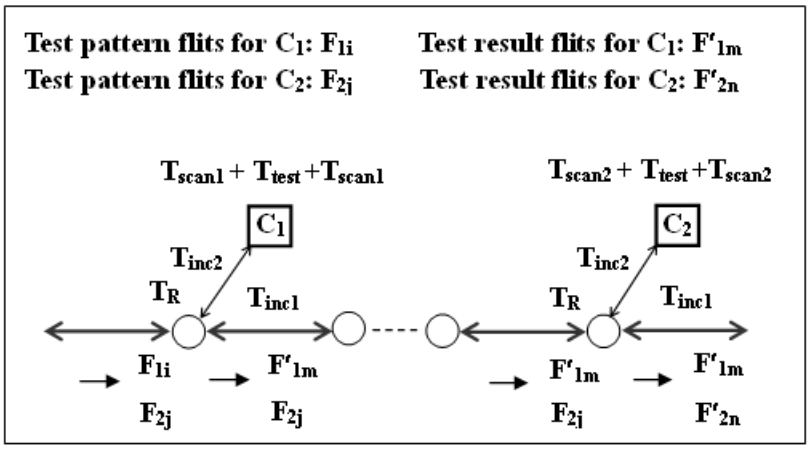


Figure 5.5: An example of test pattern and result flits interleaving.

Instead of continuing to R_2, each flit F_{1i} stops at R_1 and is forwarded to C_1 for testing. The last pattern flit of C_1 ($F_{1(N_1-1)}$) reach R_1 at time $T_{F_{1(N_1-1)}\_R_1} = T_1 + (N_1 - 1) \cdot X_1$. Then, it is forwarded to C_1, which takes $T_R + T_{inc2}$. It takes T_{scan1} (equals to X_1) for this last flit to be scanned into the wrapper scan chains of C_1, and finally C_1 is tested for T_{test} ($T_{test} = 1$ for most embedded cores using scan-based testing). After another T_{scan1} cycles, the first result flit is scanned out of the wrapper scan chains. It takes T_{inc2} clock cycles for the first result flit reaches router R_1. Therefore, the time interval between the first result flit and last pattern flit of C_1 to reach R_1 is: $T_{delay1} = T_R + T_{test} + 2 \cdot T_{scan1} + 2 \cdot T_{inc2}$. Thus, the time for the m_{th} result flit of C_1 (F'_{1m}) to reach R_1 is: $T_{F'_{1m}\_R_1} = T_{F_{1(N_1-1)}\_R_1} + T_{delay1} + m \cdot X_1$.

Since F_{1i}'s and F_{2j}'s are well interleaved in R_1, we have $T_{F_{1i}\_R_1} \neq T_{F_{2j}\_R_1}$, where i and j can be any set of integers. To keep interleaving F'_{1m}'s and F_{2j}'s without conflict, we also need $T_{F'_{1m}\_R_1} \neq T_{F_{2j}\_R_1}$ for integers j and m. As can be easily observed from Figure 5.6, F'_{1m}'s should take the same time slots originally allocated to F_{1i}'s to be well interleaved with F_{2j}'s. Thus, the requirement becomes:

$T_{F'_{1m}\_R_1} = T_{F_{1i}\_R_1} + k \cdot X_1$, where k can be any positive integer.

Applying the values of $T_{F'_{1m}\_R_1}$ and $T_{F_{1i}\_R_1}$ in the above equation, we have:

$T_1 + (N_1 - 1) \cdot X_1 + T_{delay1} + m \cdot X_1 = (T_1 + i \cdot X_1) + k \cdot X_1$,

and this leads to:

$T_{delay1} = (k + i - N_1 - m + 1) \cdot X_1$

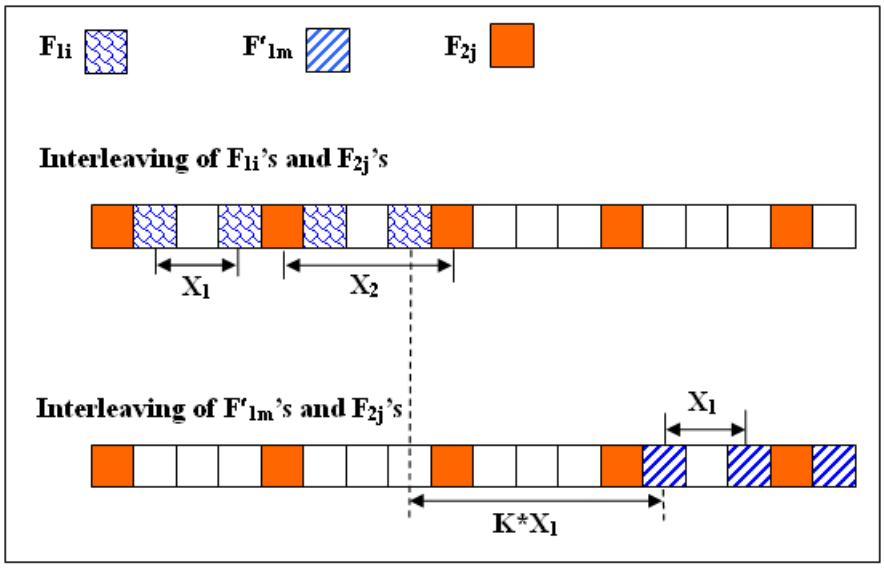


Figure 5.6: Test pattern and result flits interleaving.

Applying the value of T_{delay1}, we have:

$$T_R + T_{test} + 2 \cdot T_{scan1} + 2 \cdot T_{inc2} = (k + i - N_1 - m + 1) \cdot X_1.$$

Since $T_{scan1} = X_1$, we get:

$$T_R + T_{test} + 2 \cdot X_1 + 2 \cdot T_{inc2} = (k + i - N_1 - m + 1) \cdot X_1$$

Further simplifying the equation, we have:

$$T_R + T_{test} + 2 \cdot T_{inc2} = (k + i - N_1 - m - 1) \cdot X_1$$

By assigning $k_1 = k + i - N_1 - m - 1$, the equation can be written as:

$$T_R + T_{test} + 2 \cdot T_{inc2} = k_1 \cdot X_1. \tag{5.1}$$

Therefore, as long as we can find an integer k_1 to satisfy the above equation, result flits of C_1 (F'_{1m}) can be interleaved with pattern flits of C_2 (F_{2j}) without conflict.

Assume after passing through P routers (Figure 5.5), F'_{1m} and F_{2j} reach the local router of C_2 (called R_2). Thus, F_{2j} should reach R_2 at time $T_{F_{2j}\_R_2} = T_{F_{2j}\_R_1} + P \cdot (T_R + T_{inc1})$, and F'_{1m} reach R_2 at time $T_{F'_{1m}\_R_2} = T_{F'_{1m}\_R_1} + P \cdot (T_R + T_{inc1})$. The last pattern flit of C_2 ($F_{2(N_2-1)}$) reach R_2 at time $T_{F_{2(N_2-1)}\_R_2} = T_2 + (N_2 - 1) \cdot X_2 + P \cdot (T_R + T_{inc1})$. Similarly, we can derive the time for the n_{th} result flit of C_2 (F'_{2n}) to reach R_2 by $T_{F'_{2n}\_R_2} = T_{F_{2(N_2-1)}\_R_2} + T_{delay2} + n \cdot X_2$, where $T_{delay2} = T_R + T_{test} + 2 \cdot T_{scan2} + 2 \cdot T_{inc2}$.

Since F'_{1m}'s and F_{2j}'s are well interleaved in R_2, we have $T_{F'_{1m}\_R_2} \neq T_{F_{2j}\_R_2}$, where m and j can be any set of integers. To keep interleaving F'_{1m}'s and F'_{2n}'s without conflict, we also need $T_{F'_{1m}\_R_2} \neq T_{F'_{2n}\_R_2}$ for integers m and n. To satisfy the requirement, we need:

$T_{F'_{2n}\_R_2} = T_{F_{2j}\_R_2} + k' \cdot X_2$, where k' can be any integer.

Applying the values of $T_{F'_{2n}\_R_2}$ and $T_{F_{2j}\_R_2}$, after simplifications, we get:

$T_R + T_{test} + 2 \cdot T_{inc2} = (k' + j - N_2 - n - 1) \cdot X_2$

By assigning $k_2 = k' + j - N_2 - n - 1$, the equation can be written as:

$$T_R + T_{test} + 2 \cdot T_{inc2} = k_2 \cdot X_2. \tag{5.2}$$

Finally, Equations 5.1 and 5.2 can be extended to the general case: as long as we can find an integer k_i' to satisfy $T_R + T_{test} + 2 \cdot T_{inc2} = k_i \cdot X_i$ for every C_i, there will be no problem to interleave the test pattern and result flits. Thus, the rule is derived.

For most NoCs, the clock frequencies of the on-chip network and the embedded cores are different, and synchronization is needed between a core and its local router. In this case, the time for a data flit to transfer between a core and its local router is not only the time to pass the interconnect (T_{inc2}), but also including the the time for synchronization. So, the delay between the first result flit and last pattern flit of C_i becomes $T_{delayi} = T_R + T_{test} + 2 \cdot T_{scani} + 2 \cdot T_{inc2} + T_{synci}$, where T_{synci} is the synchronization time between C_i and R_i. Therefore, the requirement in Rule 1 should be extended to a more general form, which is : $T_R + T_{test} + 2 \cdot T_{inc2} + T_{synci} = k_i \cdot X_i$. Generally, the time for synchronization between different clock domains is not a constant, but a distribution with a mean value and a variance. Fortunately, routers in this case are usually able to work together with the synchronization circuit to adjust each arrival flit for interleaving. That is, when two flits for two different cores lose their interleaving alignment at a router R_i, the buffer of R_i can work as a regulator to resume the interleaving alignment. Thus, the synchronization problem for different clock domains seems to be a minor problem for NoCs, though details still need to be taken into account.

5.3.2 Interleaving between flits with different paths

Assume F_{1i}'s and F_{2j}'s are two sets of data flits for different cores, and they share the same test I/O resource. F_{1i} (F_{2j}) reach a router R_1 at time $T_{F_{1i}\_R_1}$ ($T_{F_{2j}\_R_1}$). The time interval between F_{1i} and $F_{1(i+1)}$ (F_{2j} and $F_{2(j+1)}$) is X_1 (X_2) cycles. After passing through N_1 (N_2) routers, F_{1i} (F_{2j}) reach router R_2 at time $T_{F_{1i}\_R_2}$ ($T_{F_{2j}\_R_2}$). By assuming $X_1 \leq X_2$ and using the same notation as in Sub-section 5.3.1, the rule shown below for interleaving flits with different paths can be derived.

Rule 2: If F_{1i}'s and F_{2j}'s can be well interleaved in the starting router R_1 but pass through different paths after R_1 (Figure 5.7), to guarantee the interleaving to continue without any conflict in another router R_2, there must exist an integer k to satisfy the equation: $(T_R + T_{inc1}) \cdot (N_1 - N_2) = k \cdot X_1$.

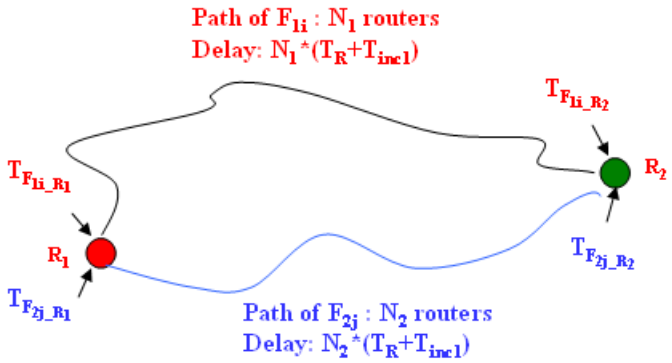


Figure 5.7: An example of flits interleaving with different paths.

We will also use an example (shown in Figure 5.7) to analyze how this rule is derived. Since F_{1i}'s and F_{2j}'s are well interleaved at R_1, we have $T_{F_{1i}\_R_1} - T_{F_{2j}\_R_1} \neq 0$ for integers i and j. Let d_0 be the delay between two closest data flits from F_{1i}'s and F_{2j}'s at R_1 (as shown in Figure 5.8). Since the time interval between any pair of two flits of F_{1i}'s is a multiple of X_1, we have:

$$T_{F_{1i}\_R_1} - T_{F_{2j}\_R_1} = d_0 + k_1 \cdot X_1. \tag{5.3}$$

where k_1 can be any integer. The interleaving example at R_1 of Fig. 13 has $d_0 = 1$, $X_1 = 2$, and $K_1 = 2$.

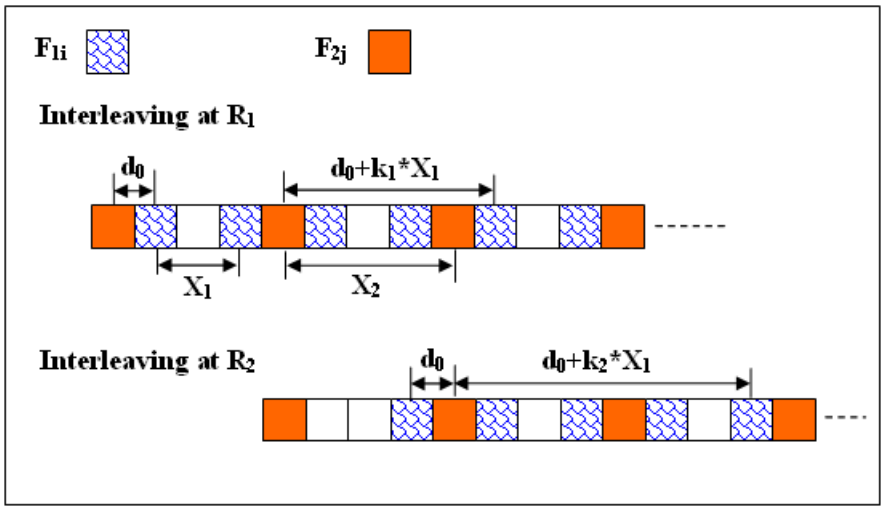


Figure 5.8: Flits interleaving with different paths.

To keep interleaving F_{1i}'s and F_{2j}'s at R_2, the timing relationship between F_{1i}'s and F_{2j}'s at R_1 should continue, i.e., for any set of i and j, we have:

$$T_{F_{1i}\_R_2} - T_{F_{2j}\_R_2} = d_0 + k_2 \cdot X_1 \tag{5.4}$$

where k_2 can be any integer. Combining Equations 5.3 and 5.4, we have:

$$(T_{F_{1i}\_R_2} - T_{F_{2j}\_R_2}) - (T_{F_{1i}\_R_1} - T_{F_{2j}\_R_1}) = (k_2 - k_1) \cdot X_1.$$

Since $T_{F_{1i}\_R_2} = T_{F_{1i}\_R_1} + N_1 \cdot (T_R + T_{inc1})$ and $T_{F_{2j}\_R_2} = T_{F_{2j}\_R_1} + N_2 \cdot (T_R + T_{inc1})$, let $k = k_2 - k_1$, simplifying both equations, we have:

$$(T_R + T_{inc1}) \cdot (N_1 - N_2) = k \cdot X_1,$$

where k (equal $k_2 - k_1$) can be any integer. Thus, the rule is derived.

For a 2-D mesh structure, we observe that: If two flits F_1 and F_2 both depart from the same router R_1 and arrive at the other router R_2, with N_1 (N_2) routers passed by F_1 (F_2) during this process, the difference between N_1 and N_2 must be an even integer. This can be easily verified as shown in Figure 5.9.

Given a pair of routers in a 2-D mesh network and a path between the source and the destination, no matter how the path is routed, it can be divided into 10 segments (Figure 5.9) and the total path length is:

$$L = |X_d - X_s| + |Y_d - Y_s| + X_{s+} + X_{s-} + Y_{s+} + Y_{s-} + X_{d+} + X_{d-} + Y_{d+} + Y_{d-}.$$

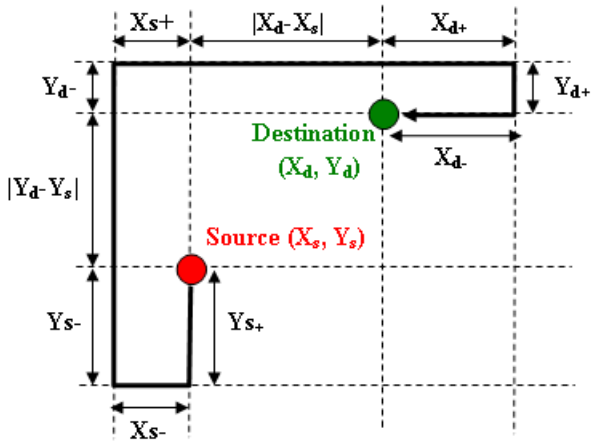


Figure 5.9: Routing path length in a 2-D mesh structure.

We emphasize here that the routing path shown in Figure 5.9 is just a general case, i.e., the shape of an actual routing path may not be the same as in this figure. However, any real routing path can be represented using this general case by setting different values to each segment (sometimes, value 0 is assigned). Easily, we find the relations: $X_{s+} = X_{s-}$, $Y_{s+} = Y_{s-}$, $X_{d+} = X_{d-}$ and $Y_{d+} = Y_{d-}$. Also, we know that $|X_d - X_s| + |Y_d - Y_s|$ is equal to L_{min} which is the minimum path length between the source and destination. Therefore, the length of any path between this router pair can be represented as:

$$L = L_{min} + 2 \cdot (X_{s+} + Y_{s+} + X_{d+} + Y_{d+}).$$

noindent Obviously, the difference in length between any two paths of this router pair is an even number. Thus, the difference between the number of routers traversed (N_1 and N_2) by any pair of two paths between routers R_1 and R_2 is an even number.

Based on this observation, $N_1 - N_2$ can be represented as $2 \cdot p$ (p is an integer), and Rule 2 can be written as $(T_R + T_{inc1}) \cdot 2 \cdot p = k \cdot X_1$. For most NoCs, we have $T_R = T_{inc1} = 1$. Therefore, from this equation, as long as X_1 is an integer factor of 4, Rule 2 can be satisfied and interleaving of data flits with different paths is not a problem. Similar to Rule 1, if Rule 2 cannot be directly satisfied for some cores, the interleaving can still be implemented by inserting simple logic to compensate the delay.

5.3.3 Checking of Interleaving Capability

As stated in the beginning of this sub-section, to enable parallel test of multiple embedded cores using the same test I/O resource, data flits of these cores need to be interleaved without any conflict. To guarantee the interleaving, two rules presented above need to be satisfied. Therefore, after all cores are scheduled to appropriate test I/O resources, an extra interleaving checking process is required. If a core C_i is not interleaved with other cores for testing, it does not need to be checked. Otherwise, C_i needs to be checked with Rule 1. If Rule 1 cannot be met, appropriate delay logic circuits need to be inserted as we stated in 5.3.1. After this, we also need to check Rule 2 for this core and all other cores using the same test I/O resource as C_i but different routers. Similarly, if Rule 2 cannot be satisfied, appropriate delay circuits are needed to accomplish the interleaving. It may seem complicated to add extra logic circuits to satisfy these two rules. However, the case is not so difficult to achieve most of the time. For example, generally, we have $T_R = T_{test} = T_{inc2} = T_{inc1} = 1$ and X_i equals to 2 or 4, so Rule 1 and Rule 2 can be satisfied very easily.

5.4 Experimental Results

We have applied the heuristic configuration algorithm to 40 cores from ITC'02 SoC benchmark set [46]. Since traditional SoC test wrapper optimization algorithms prefer to configure internal scan chains and functional I/Os to balanced wrapper scan chains, we also applied this equal-length configuration strategy to the benchmark set for comparison. In this algorithm (used for comparison), we first sort the internal scan chains, and then use the best-fit strategy to configure them into wrapper scan chains with the length constraint. After that, we use the best-fit strategy to assign functional I/Os to fill existing wrapper scan chains to maximum length, without any δ limit as in our heuristic algorithm. If there are extra I/Os left, we further use the best-fit strategy to assign them to new wrapper scan chains with the length constraint. The results obtained by these two configuration algorithms are shown in Figure 5.10.

As we can see, among 40 benchmark cores, 20 of them can be solved by using the simple equal-length configuration method without any waste, but the other 20 of them have significant waste. For the 20 remaining cores, our configuration algorithm can reduce the data flit waste to 0 for 14 cores, and have slight improvement for the other 6 cores, as shown in Figure 5.10. For the

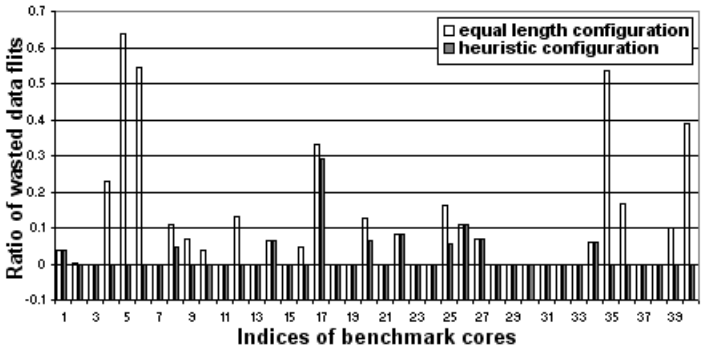


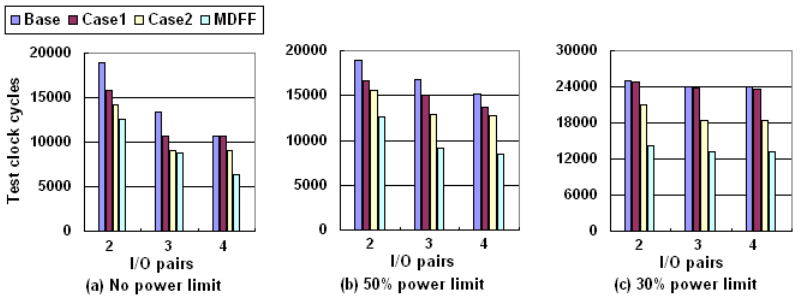
Figure 5.10: Comparison of data flit waste between equal length configuration and our algorithm.

cores that have significant improvement using our configuration, only 2 or 3 data flit formats are required, so the hardware overhead is quite small.

To evaluate the performance of our test scheduling method, we applied the heuristic wrapper scan chain configuration algorithm and test scheduling method to the ITC'02 benchmark set, and compare the results with the published works [25][26]. The selected benchmark circuits are d695, g1023 and p22810 for easy comparison with the results of [25][26]. All the results are based on the assumption that the width of each data flit is equal to 32 bits and all NoCs originally work in a single clock domain. The results of test time for these chips after test scheduling are shown in Figure 5.11 to Figure 5.13. In these figures, *Base* denotes the test scheduling method proposed in [25], which only uses one test clock frequency and allows each data flit to contain only a single bit for each wrapper scan chain of a core. *Case1* and *Case2* denote the test scheduling methods proposed in [26], which allow each data flit to contain multiple bits for each wrapper scan chain of a core, but only allow a single data flit format for all data flits of a core. Case1 only allows normal and faster test clock frequencies for some embedded cores, while Case2 allows each embedded core to choose from a set of test clock frequencies, including slower, normal and faster frequencies. *MDFF* denotes our proposed test scheduling method based on the optimal wrapper scan chain configuration algorithm. We want to note that, different topologies of a NoC greatly affect the efficient utilization of network resources, and thus affect the final results of test scheduling. Since no topologies were available from the benchmark set description or the other published works, we choose the optimum topologies for our test scheduling method in this paper.

Without losing the generality, the power consumption limit during test is defined as a func-

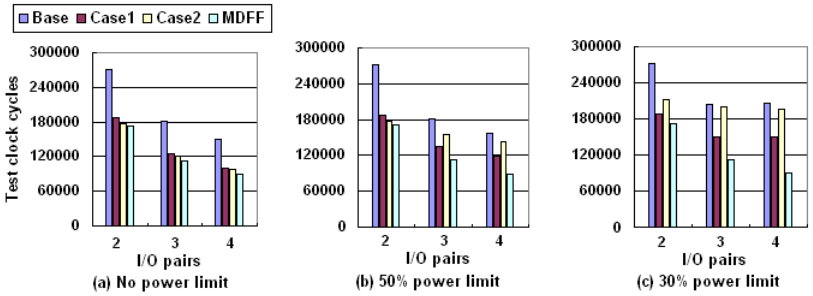
tion of the overall power consumption of the embedded cores. This function is a percentage of the sum of the power consumptions of all cores, e.g., a test power limit of 50% indicates that the power limit corresponds to half of the total power consumption of all cores. With this assumption, the result of the test scheduling algorithm can be independent of the absolute power values. However, in the real case, the designer can define any power limit according to the system requirements. In our experiments, we considered the conditions with 2, 3 and 4 test interface I/O pairs, and with no power limit, 50% power limit and 30% power limit.



d695	no power limit			50% power limit			30% power limit		
I/O resource	2	3	4	2	3	4	2	3	4
Base	18869	13412	10705	18869	16771	15156	25024	24039	24039
Case1	15770	10705	10705	16615	15048	13644	24809	23738	23518
	-16.42%	-20.18%	0.00%	-11.95%	-10.27%	-9.98%	-0.86%	-1.25%	-2.17%
Case2	14176	9073	9073	15576	12887	12696	21029	18317	18317
	-24.87%	-32.35%	-15.25%	-17.45%	-23.16%	-16.23%	-15.96%	-23.80%	-23.80%
MDFF	12576	8798	6370	12576	9188	8468	14107	13228	13228
	-33.35%	-34.40%	-40.50%	-33.35%	-45.21%	-44.13%	-43.63%	-44.97%	-44.97%

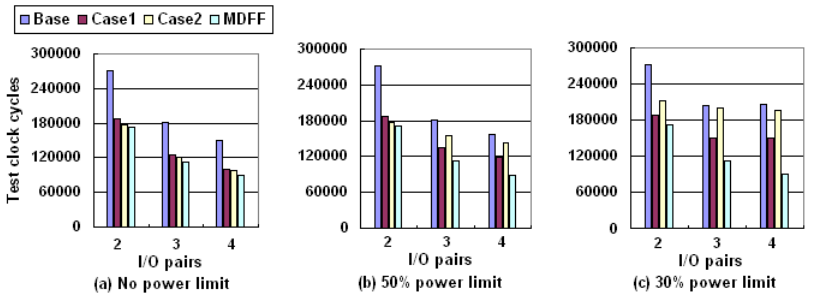
Figure 5.11: Test scheduling results on benchmark circuit d695.

As we can see, our test scheduling method based on the MDFF concept and wrapper scan chain configuration algorithm achieves the smallest test time for all these benchmark circuits. For benchmark circuit d695, by our method, all cores use the same test clock frequency f as the on-chip network. In benchmark circuit g1023, only one core needs to use higher test frequency ($2f$). And, in benchmark circuit p22810, only two cores need to use higher test clock frequency ($2f$). Thus, only very limited hardware overhead is required to achieve smaller test time by our method. In most cases, the proposed test scheduling method outperforms Base by $30\% - 50\%$, which is much better than both Case1 and Case2.



g1023	no power limit			50% power limit			30% power limit		
I/O resource	2	3	4	2	3	4	2	3	4
Base	25062	17925	16489	25062	17925	16489	25062	22847	22847
Case1	16286	10941	9163	16286	12313	12087	18355	14263	14263
	-35.02%	-38.96%	-44.43%	-35.02%	-31.31%	-26.70%	-26.76%	-37.57%	-37.57%
Case2	15662	11546	9788	17078	14310	13394	18558	17906	17646
	-37.51%	-35.59%	-40.64%	-31.86%	-20.17%	-18.77%	-25.95%	-21.63%	-22.76%
MDFF	12441	7626	7397	12441	8279	7767	13460	12825	12682
	-50.36%	-57.46%	-55.14%	-50.36%	-53.81%	-52.90%	-46.29%	-43.87%	-44.49%

Figure 5.12: Test scheduling results on benchmark circuit g1023.



p22810	no power limit			50% power limit			30% power limit		
I/O resource	2	3	4	2	3	4	2	3	4
Base	271384	180905	150921	271384	181169	156810	271384	204724	205506
Case1	187061	124787	99345	187061	134998	119202	187061	149042	149767
	-31.07%	-31.02%	-34.17%	-31.07%	-25.49%	-23.98%	-31.07%	-27.20%	-27.12%
Case2	177915	120840	98842	177915	154478	143044	211505	199151	195122
	-34.44%	-33.20%	-34.51%	-34.44%	-14.73%	-8.78%	-22.06%	-2.72%	-5.05%
MDFF	171886	112805	89504	171886	112805	89504	171886	112805	89504
	-36.66%	-37.64%	-40.69%	-36.66%	-37.73%	-42.92%	-36.66%	-44.90%	-56.45%

Figure 5.13: Test scheduling results on benchmark circuit p22810.

Chapter 6

Testing Mode Router Design and A Complete Router Architecture

In Chapter 3, we have proposed a DyXY routing algorithm for normal mode operations. Besides normal working mode, routers also need to support testing mode operations. In this section, we will discuss the router design issues for testing mode, and will present a complete router architecture to support both working modes at the end of this chapter.

6.1 Testing Mode Router Design Issues

No matter which test strategy to choose, besides the test function implementation, minimum hardware complexity and area overhead are two most important aspects to evaluate the merits of the design. Two basic questions to be considered in testing mode router design are stated below.

- **How to set a router to operate in different modes?**

  In NoC, there is no global test control signal, and all signals need to be transferred using the network in the form of packets. Therefore, the operation mode selection of a router also needs to refer the information carried by each incoming packets. To minimize the network test traffic and router complexity, an efficient method to embed operation-mode information in a packet needs to take three aspects into consideration. First, the embedding of information should add minimum overhead to the size of each packet. Second, the operation-mode control

logic in each router should be as simple as possible. And the last point, it is better to isolate this logic unit from the other functional units in each router, so the functional test of a router will not be affected by this part.

- **Which routing mechanism should be chosen to support test pattern transfer and how to implement it?**

  For embedded core testing, test patterns need to be routed following specific paths, which are dedicatedly selected to achieve maximum network utilization and minimum testing time. Traffic in test mode is pre-known once the test plan is developed. Therefore, routing in test mode should be deterministic instead of adaptive as in normal mode. As we know, there are generally two ways to implement deterministic routing in a network. First, the routing information can be embedded in each incoming packet, and the routers only need to forward the packets based on their routing information. The second way is that the packet only include source and destination addresses, and the routers make decision based on the addresses and routing algorithms (which is implemented as internal logic of each router). Either of these two methods has its advantages and disadvantages. For the first one, the router structure is very simple, however the packet size is bigger and thus induces traffic overhead. For the second one, only few extra information need to be inserted in a packet, however, routers become more complex and thus induces area overhead. Which method is the better one relies on the specific test strategy adopted.

Before presenting the details of our testing mode router design, let us first review our embedded core test strategy. Based on our test scheduling strategy, each test packet follows deterministic routing path. Paths of most test packets are XY routing path by default, however, there are also a few exceptions which need to take different routing paths to avoid blocking during parallel test. Which path needs to change and what is the new path are determined during the test scheduling process. Recall the DyXY routing algorithm for normal mode operation, we found that the basic logic to implement DyXY routing is the same as that to implement XY routing, and the only difference is that DyXY needs to consider congestion information in proximity when making routing decision. Therefore, only a little optimization on routers can help us to implement XY and DyXY routing at the same time. The question now becomes how to implement special routing paths for

the few exceptions. As we have stated in the test scheduling strategy, the special routing paths are XY routing paths for most routers, and only take special turns in very few routers. Considering the logic complexity, we plan to implement special routing paths by inserting the special routing information in the routing map of related routers.

Three questions need to be answered to implement this idea. First, how to set the special routing information into designated routers. Second, how to let a router know which packet needs to follow the special routing path as stated in the routing map. And third, what is the format of the routing map. For the first question, obviously, we need a special type of packets to set the routing map of routers, and the type of the packets should be indicated in the head flits of packets. The second question can be solved in two ways, by embedding information in the packets or saving information in the routers. Since a packet may need different actions in different routers, a better way is to embed the information in the routers' routing map. And based on the second question, we get the answer to the third question: each routing item in the routing map needs to include the ID of a test packet and the routing direction. Therefore, to implement special routing for some test packets, we define a new packet type, set packet. set packets are used to set special routing items (each item includes a Core ID and a routing direction) to related routers, such that these routers will forward test packets of specific cores based on the directions indicated in the routing items. For simplification, we apply the default XY routing for set packets.

The general structure of different flits in packets are shown in Figure 6.1. As we can see, the packet type and flit type information are clearly stated in the flits, so routers can easily make decision based on these information. The detail of routers' working flow is described in the next section.

Here, we would like to spend a few more sentences to explain the format of set packets. Let us recall the example of routing path find example in Chapter 5, as shown in Figure 6.2. In this example, to implement parallel testing of Core 6 and Core 7, we force the test packets of Core 6 to take special routing path instead of XY routing path. To realize this, we need to set special routing items to the routing maps of Router 6 and Router 9. Assume the network addresses of Core 6, Router 6 and Router 9 are (1, 2), (1, 2) and (0, 1), respectively, the set packets for Router 6 and Router 9 are shown in Figure 6.3. We note that, the 'Destination' in the Head flits (and Tail flits) means the destination router of a set packet, the 'Test Core' in the data flits means the Core

Head Flit

Flit Type (2 bits)	1 0
Packet Type (2 bits)	Set packet: 00 Normal packet: 11 Test packet with XY routing: 01 Test packet with special routing: 10
Source Address (log_2N bits)	...
Destination Address (log2N bits)	...
Other special flags	...

Tail Flit

Flit Type (2 bits)	0 1
Packet Type (2 bits)	Set packet: 00 Normal packet: 11 Test packet with XY routing: 01 Test packet with special routing: 10
Source Address (log_2N bits)	...
Destination Address (log2N bits)	...
Other special flags	...

Data Flit for Path Setting packets

Flit Type (2 bits)	0 0
Test Core ID address (log_2N bits)	...
Routing direction for this core (2 bits)	00 east 01 north 10 west 11 south
Other special flags	...

Data Flit for Normal and Test packets

Flit Type (2 bits)	1 1
Data to be sent	...
Other special flags	...

Figure 6.1: Formats of flits.

whose test packets need special routing paths, and the 'Routing direction' in the data flits means the scheduled routing direction of test packets of the 'Test Core'. Since test packets include both test pattern packets and test result packets, we use 'Other flag' to differentiate these two type of test packets, e.g., '1' mean test pattern packets and '0' means test result packets. Therefore, the set packets in Figure 6.3 mean that Router 6 should forward all test result packets of Core 6 to the 'South' direction, and Router 9 should forward all test pattern packets of Core 6 to the 'North' direction. The 'Test Core' address, the 'Routing direction', as well as the flag indicating the test packet type (test pattern or test result packets) should all be included in each routing item within routing maps.

6.2 Router Architecture

The basic router architecture is as shown in Figure 3.2 and the general working flow of router has been described in Section 3.1. In this section, we will discuss the logic of the center component of each router, the controller, in detail. The working flow of the controller is shown in

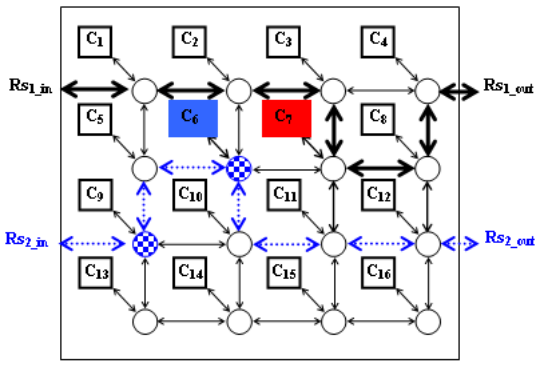


Figure 6.2: Example of Special Routing Path.

Figure 6.4.

As shown in the Figure 6.4, upon receiving a flit, the controller will first check the flit type. For head flits, the task is to make routing decision. The controller then checks the packet type. For head flits of normal packets, the controller will find a routing direction using DyXY routing algorithm. For head flits of set packets or test packets with XY routing, the controller will find a routing direction using XY routing algorithm. For head flits of special routing test packets, the controller will first look up the routing map. If the source (or destination) address of the head flit is the same as a 'Test Core' address in the routing map, the routing direction in the routing map corresponding with this 'Test Core' address is the routing decision. Otherwise, the controller will find the routing direction using XY routing algorithm. After a routing decision is made or found, the decision is saved in related item (based on the incoming direction of the flits) of the routing table for the use of following data flits. Also, if current router is not the destination of the flit, the head flit will be forwarded to the next router based on the routing decision. We note here that, test packets include both test pattern packets and test result packets. In test pattern packets, the destination address is the Test Core address, and in test result packets, the source address is the 'Test Core' address. So, when we look up the routing map, we should use the destination address in head flits of test pattern packets, and use source address in head flits of test result packets. Whether a test packet is test pattern packet or test result packet can be indicated using the 'other flag' in head flits. For example, '1' means test pattern packets and '0' mean test result packets. Let us still use the example in Figure 6.2 for demonstration. Given the head fit of test pattern packets of Core 6. When this head flit reaches Router 9, the controller will use its destination address to

Head Flit

Flit Type	10
Packet Type	00
Source Address	...
Destination Router Address	01 – X address 10 – Y address
Other flags	0 – Test result packet

Data Flit

Flit Type	00
Test Core Address	01 – X address 10 – Y address
Routing Direction	11 – South
Other flags	0 – Test result packet

Tail Flit

Flit Type	01
Packet Type	00
Source Address	...
Destination Address	01 – X address 10 – Y address
Other flags	0 – Test result packet

(a) Set packet for Router 6

Head Flit

Flit Type	10
Packet Type	00
Source Address	...
Destination Router Address	00 – X address 01 – Y address
Other flags	1 – Test pattern Packet

Data Flit

Flit Type	00
Test Core Address	01 – X address 10 – Y address
Routing Direction	01 – North
Other flags	1 – Test pattern Packet

Tail Flit

Flit Type	01
Packet Type	00
Source Address	...
Destination Address	00 – X address 01 – Y address
Other flags	1 – Test pattern Packet

(b) Set packet for Router 9

Figure 6.3: Example Set Packets.

look up the routing map. A routing item will be found, and the related routing direction with this routing item is '01', so the routing decision for this flit is 'North'. The head flit is then forwarded to Router 5. Since no related routing item can be found in Router 5's routing map, based on XY routing rule, the routing direction of this packet is 'East', so the head flit is forwarded to Router 6. Although there is a routing item in Router 6 whose Test Core address is the same as the head flit's destination address, the test packet type with this item is test result packet instead of test pattern packet, so the routing decision is '111' (local Core) based on XY routing algorithm. Readers might get confused at this point regarding the bit representation of routing decisions, since we only

use two bits to represent routing decisions in set packets, while use three bits here. We would like to spend a few sentences to explain this. There are totally five possible routing directions, 'East', 'North', 'West', 'South', and 'Local'. To represent all these five directions, we need three bits, '000' for 'East', '001' for 'North', '010' for 'West', '011' for 'South' and '111' for 'Local'. So, each routing decision in the routing table should has three bits. Let us recall the purpose of set packets. They are used to tell routers to forward some test packets to special directions instead of following XY routing path. The only possible special routing directions are 'East', 'North', 'West' and 'South', since we will never need to force a test packet to enter its destination core (which can be naturally decided by XY routing rule), and we will never allow any test packet to enter any other core which is not its destination. Therefore, the 'Local' direction will never occur in any set packet. To save space, we only use two bits to represent the four possible routing directions in set packets. Careful readers may question that head flits should stop at their destination routers, instead of being forwarded to the embedded cores. This is true. However, an important task of head flits is to find and set a routing path for following data flits of the same packet. So, the routing decision still needs to be made and stored in the routing table, such that the following data flits can directly read this routing decision and be forwarded to the embedded cores. Only when the test result packets of Core 6 reach Router 6, they will follow the direction in the routing map, and be forwarded to the 'South' direction.

For data flits, the controller will find a routing decision from the routing table based on the flits' incoming direction. This routing decision is made earlier based on the information in the head flits of the data flits. If current router is the destination of the data flits, and the data flits belongs to set packet, the controller will get the information from the data flits and save them in the routing map. For example, for the set packet of Figure 6.3(b), when the data flit reaches Router 9, the controller will get the 'Test Core Address', the flag showing the type of the test packet, and the 'Routing Direction' (which is '0110 1 01') from the data flit, and save this content as a routing item in Router 9's routing map. Otherwise, the controller will forward the data flits based on the routing decision. We note here that, we apply wormhole routing for flits within the same packet, i.e., all flits within the same packet will follow the same routing path and will have the same incoming direction for each router. Thus, each router's routing table will need to have five items, each for an incoming direction (east, west, north, south and local).

For each tail flit, the controller will find a routing decision from the routing table based on the flit's incoming direction. And then, the controller will write a special value (an invalid indication) to the related item of the routing table. The tail flit is then forwarded to the next router based on the routing decision, if the current router is not the destination. Let us use Figure 6.3(b) as an example. When the tail flit of the set packet for Core 6 reaches Router 9, the controller will look up the routing table for its routing decision based on its incoming direction. Bit pattern '001' will be found which means 'North'. Since this is not the destination of the tail flit, it will be forwarded to the 'North' direction. At the same time, a special value will be written to the related routing decision item in the routing table of Router 9, to set an invalid direction. When the tail flit reaches Router 6, the routing decision from the routing table is '111' which means 'Local'. Since it is the destination of the tail flit, the flit will not be forwarded anymore. The related routing decision item in the routing table will also be set to an invalid value.

Readers may question whether the contents in the routing map be set to invalid, together with the routing table. Before answering this question, let us clarify the purpose of routing map and routing table first. The purpose of a routing map is to store special routing decisions for some test packets which do not follow the XY routing rule. These decisions are determined by the test scheduling algorithm and are fixed. The contents in a routing map are set by set packets, and should remain unchanged once they are written. The purpose of a routing table, however, is to store the routing decisions for current incoming packets (including set packets, test packets and normal packets) for the use of data and tail flits. The routing decision in a routing table can come from either its routing map, or the result of its routing logic based on XY or DyXY routing algorithms. Since there will be many different incoming packets, the contents of the routing table change dynamically with different incoming packets. For each incoming packet, once the routing decision is made with its head flit, the corresponding item of the routing table will remain unchanged until the arrival of the tail flit. Therefore, when a tail flit arrives at a router, the related routing item of the routing table will be set to invalid since this packet is ended; however, the contents in the routing map should remain unchanged.

The structure of each router controller is shown in Figure 6.5. As we can see, a controller consists of a flit type checker, a packet type checker, a routing logic, a routing table, a write enable signal generator and a few Multiplexers. Inputs of the controller are the incoming flits, stress values

and the incoming direction information, and the outputs are signals to control the crossbar switch to forward incoming flits to correct directions.

The function of the *flit type checker* is to check the type of each incoming flit. The inputs of this function block are the first two bits of a flit: F_type_0 and F_type_1. There are three output control signals F_sel0, F_sel1 and F_sel2, each indicating the head flit, the tail flit and the data flit of a set packets respectively. The logic of this function block is shown in Figure 6.6. Similarly, the function of the *packet type checker* (Figure 6.7) is to check the packet type for each incoming head flit or tail flit. Each packet type checker has two output control signals: P_sel0 and P_sel1. P_sel0 indicates whether a flit belongs to a normal packet or not, and it is one of the inputs for the stress value comparator. P_sel1 indicates whether a flit belongs to a test packet with special routing or not, and it is a control signal for the 3-to-1 Mux in the routing logic.

Routing logic is a very important function block of a controller. It consists of a first step decision logic, a second step decision logic, a stress value comparator, a routing map logic and a few Muxs. The function of the *first step decision logic* is to make first level decision of possible routing directions for incoming flits. As shown in Figure 6.8, this function block is composed of two comparators which compare the X and Y addresses of the destination of each incoming head flit and those of the current router. The logic of this block is very simple as shown in this figure. Each comparator has two-bit output to indicate the comparison result. The output will serve as the input for the second step decision logic, as well as the control signal of the 4-to-2 Mux to select an appropriate stress value.

As we have stated in Section 3.1, there are four stress value registers in each router, each storing the stress value of a neighboring direction. Based on the outputs of the first step decision logic, the 4-to-2 Mux (shown in Figure 6.9) selects stress values of two possible routing directions as the inputs of the stress value comparator. The *stress value comparator* is shown in Figure 6.10. The basic function of this block is to compare the stress values of two possible routing directions. However, since we adopt DyXY routing only for normal packets, stress values should not affect the routing decision of other packets. Thus, one other control signal, P_sel0, is also taken as the input of this block. As shown in Figure 6.5, $P\_sel0 = And(P\_type\_0, P\_type\_1)$, i.e., $P\_sel0 = 1$ only for flits belongs to normal packets. Thus, the output of the stress value comparator only represents the actual comparison result of stress values when $P\_sel0 = 1$. Otherwise, the output is set to '0'

which indicates that X direction is preferred.

The *second step decision logic* is the function block which makes routing decision. Inputs of this block includes the outputs of the first step decision logic and the output of the stress value comparator. The decision is made based on the DyXY routing algorithm. However, since the packet type information is included in the logic of stress value comparator, the decision is actually based on XY routing for set packets and test packets. The detailed logic is shown in Figure 6.11. The output of this block has three bits, indicating five possible routing decisions.

We will use some examples to show how the routing decisions are made by the first step decision logic, stress value comparator and second step decision logic. Let us still use the NoC structure shown in Figure 6.2. Assume there is a Set packet for Router 6 entering Router 9. The first step decision logic of Router 9 compares the destination address (Router 6) with the current router address (Router 9). Since Router 6 locates in the northeast of Router 9, the output of the first step decision logic will be: $X\_comp\_0 = 0$, $X\_comp\_1 = 0$, $Y\_comp\_0 = 0$, $Y\_comp\_0 = 0$, indicating 'East' or 'North' direction. Since this is a set packet, XY routing will be chosen, and the output of the stress value comparator will be set to '0', indicating that the X direction is preferred. Refer to the truth table in Figure 6.11, the output of the second step decision logic will be '000' representing the 'East' direction. Again assume there is a normal packet for Core 6 entering Router 9. The first step decision logic of Router 9 has the same output: $X\_comp\_0 = 0$, $X\_comp\_1 = 0$, $Y\_comp\_0 = 0$, $Y\_comp\_0 = 0$, indicating 'East' or 'North' direction. Since this is a normal packet, DyXY routing will be chosen, and the output of the stress value comparator will be the true comparison result of stress values of Router 5 and Router 10. Assume Router 10 has bigger stress value than Router 5, the output of the stress value comparator will be '1', indicating the Y direction is preferred. Refer to the truth table in Figure 6.11, the output of the second step decision logic will be '001' indicating the 'North' direction.

Another important function block of the routing logic is the *routing map logic* shown in Figure 6.12. This function block is composed of a routing map, an input address Mux, an address comparator, a routing map status Register, a M-to-1 Mux and an enable signal generator. The routing map is the storing unit. A routing map consists of a number (M) of items each storing a test core address, the related test packet type and a special routing direction.

The input address Mux, the address comparator, and the M-to-1 Mux are logic blocks used

for reading the contents from the routing map.

When there is an incoming head flit of a test packet with special routing, the input address Mux will select either the source address or destination address in the head flit based on the test packet type to look up the routing map. If the source (destination) address of an incoming head flit is the same as one of the test core addresses in the routing map, and the test packet type of this head flit is the same as the type flag in the related routing item, the $Find$ signal is set to '1'; moreover, the Sel signal is set to a value to control the M-to-1 Mux to output the routing direction of the related routing item. For example, if there is an item in Router 9's routing map indicating a special routing decision for test pattern packets of Core 6. When the head flit of Core 6's test pattern packet enters Router 9, the routing map logic will find this special routing decision for the head flit.

The routing map status reg and the enable signal generator are used for writing information to the routing map. The routing map status register is essentially a self-increment register, which increases by 1 at every rising edge of the $write\_en$ signal. The function of this status register is to record how many items have been written to the routing map. The input signal of the routing map status reg is $write\_en$, which is the product of F_sel2 and Dir. $F\_sel2 = 1$ indicates a data flit of set packets. Dir is the final output of the controller logic, indicating the routing direction. $Dir = 11$ means current router is the destination of the flits. So, when $write\_en = 1$, we need to write the information from the data flit to the routing map. $write\_en$ also serves as the input for the Enable signal generator. The enable signal generator will take the status register's output and the $write\_en$ signal as inputs, to enable the information of data flits of set packets (test core address, the related test packet type and a special routing direction) to be written to the appropriate place of the routing map. For example, assume that the routing map of Router 6 has totally 5 routing items and only the first one has been filled. If the data flit of one set packet of Router 6 enters at this time, the enable signal generator will generate an enable signal to write the information of this data flit to the second routing item in Router 6's routing map, and the value of routing map status reg will also be increased by 1.

The last function block within the routing logic is the *3-to-1 Mux*. The logic of the 3 to 1 Mux is stated in Figure 6.5. Inputs of the Mux are the decision from the Second step decision logic, the output from the routing map logic and a special value. For tail flits, we need to set the

related routing item to an invalid value, so, the special value is selected. For head flits of special routing test packet, if an item is found from the routing map, the result should be taken as the routing decision. Otherwise, the decision from the Second step decision logic is selected.

Output of the 3 to 1 Mux is the input of the routing table. As stated before, *routing table* consists of 5 sets of registers each storing routing decision for an incoming direction. The Write enable signal generator (Shown in Figure 6.13) takes F_sel0 and the information of the incoming direction of flits as inputs, and generates enable signals to write the decision of the routing logic to appropriate place.

The *2-to-1 Mux* is the final stage of the controller. For head flits, the controller will select the decision of the routing logic and forward head flits based on it. And for data and tail flits, the controller will select a routing decision from the routing table based on the incoming information (the output of the 5 to 1 Mux) to forward the flits.

Besides generating routing decision, the controller also have another function: generating control signals to update the values of its stress value counter. Only two-bit outputs are required, with one for increasing the stress value and the other for decreasing it. The function is simple, whenever there is a new flit entering the input buffer of the router, the controller will generate a pulse on the signal for increasing. Whenever there is a flit leaving the input buffer, the controller will generate a pulse on the signal for decreasing. Since the design of this logic depends on the detailed structure of the input buffer, we will not go into the details at this time. However, please remember that the function does exist.

With the help of the controller design, the proposed router architecture can support both normal mode and testing mode operations. Unfortunately, it still cannot support test flit interleaving. The reason comes from the assumption of wormhole routing. With wormhole routing, a routing path is reserved for one packet at a time, i.e., once the head flit of a packet begins to travel through a routing channel, this channel is reserved for all flits of the packet until it finishes. So, no flit interleaving is possible. Both normal packets and set packets can follow wormhole routing without any problem; however, to realize efficient test scheduling, we need to interleave flits of different packets to fully utilize the network channel. Therefore, test packets cannot follow wormhole routing. What will be different with this change? The operations of each head flit and tail flit are

still the same, and only operations with data flits need to be changed. With wormhole routing, when there is an incoming data flits, the controller will look up the routing table for its routing decision based on its incoming direction. For test packets, data flits of different packets can enter the same router from the same direction alternately. Therefore, we cannot look up the routing decision merely based on a data flit's incoming direction. To solve this problem, each data flit of a test packets should also include the test core address, besides other information needed for common data flits. This can be realized by adding the test Core address to the 'Other flag' area of each data flit. And, the controller must look up the routing decision for each incoming data flit based on both the incoming direction and the test core address. For example, in Figure 6.2, assume Core 11 and Core 16 will be tested using the same Test I/O concurrently, and their flits need to interleave at Router 15. Furthermore, we assume that both of their flits enter Router 15 by the 'West' direction. When head flits (test packets) of Core 11 and Core 16 enter Router 15, the controller will make routing decisions based on their destination addresses using the XY routing algorithm. So, two routing items will be written to Router 15's routing table for the West incoming direction. One item includes '001' (North) and '1001' (the XY address of Core 11), and the other item includes '000' (East) and '1100' (the XY address of Core 16). When data flits (and tail flits) of test packets for Core 11 and Core 16 enter Router 15, the controller will find their routing decisions from the routing table based on their incoming directions and test core addresses. Therefore, flit interleaving can be easily supported.

To adapt with the proposed optimization, corresponding changes are needed at the routing table logic. Instead of only one routing item for each incoming direction of head flits, there should be more items for each direction (except the local direction). Due to the constraints of channel capacity and power consumption, generally at most three packets can be interleaved in one incoming direction. Therefore, we reserve three items for each outside incoming direction (this is based on our observation during test scheduling on the ITC'02 SoC benchmark sets). Also, each item should include both the routing direction and the test core address. The logic for each incoming direction (except the local direction) of the optimized routing table is shown in Figure 6.14. The logic for each direction includes three storing units, a status Reg, an enable signal generator, a comparator and a Mux. The function of the status Reg and the enable signal generator is to write information to the storing units, and the function of the comparator and the Mux is to read information from

the storing units. The working principles of these units are the same as the corresponding units in the routing map logic. For example, if the status reg in the routing table logic for East direction is 2, it means there is only one empty storing unit for this incoming direction. If there is a new head flit coming from the East direction, its routing decision and its test core address will be written to the last storing unit for the East incoming direction. If later, a data flit comes from the East direction, the comparator in Figure 6.14 will compare the test Core address of the data flit with the addresses in the three storing units of the routing table logic for East direction, and will generate signals to control the Mux to output a routing decision from the matched storing unit.

With this optimized controller logic, we have a high-performance router architecture which can support test mode operation, as well as provide adaptive, deadlock and livelock free DyXY routing in normal operation mode.

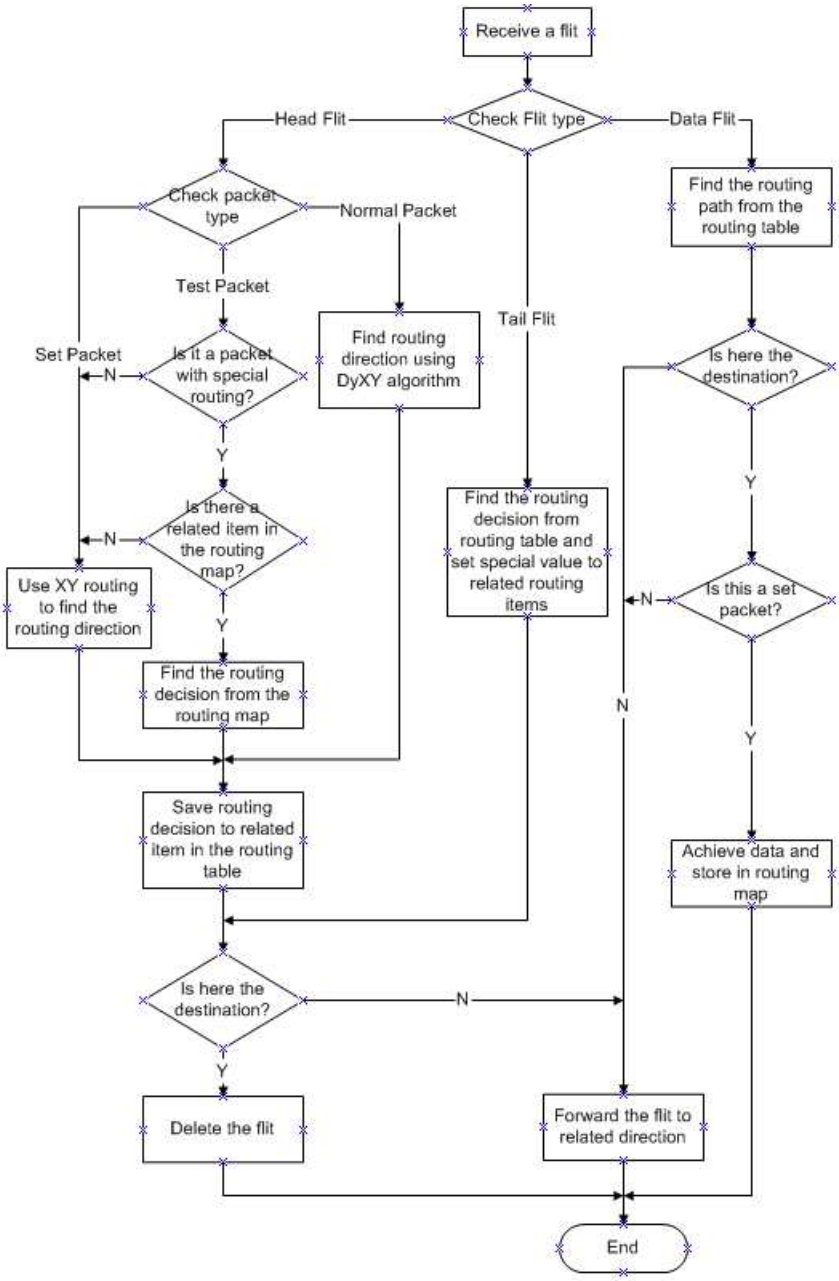


Figure 6.4: Controller working flow.

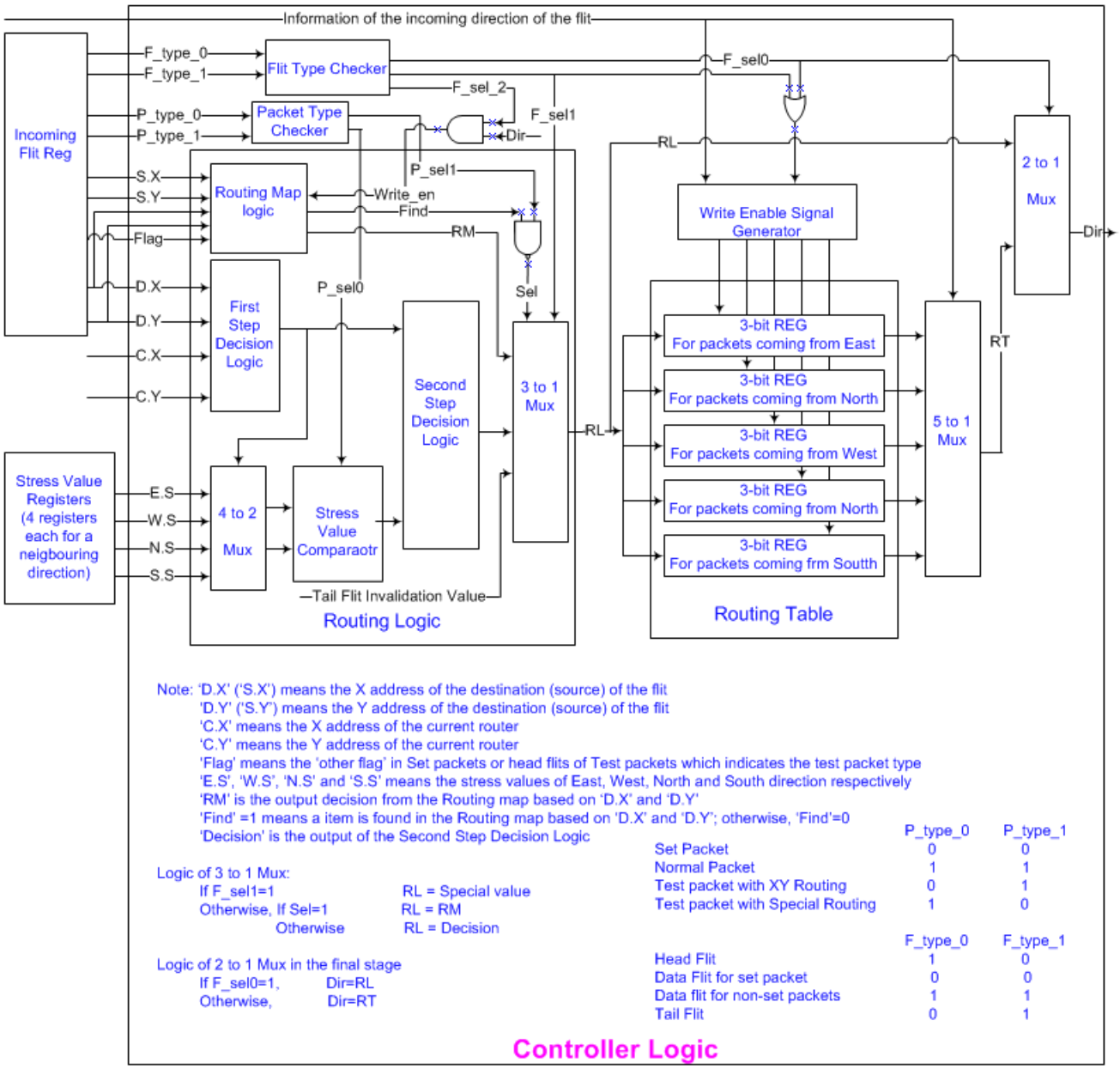


Figure 6.5: Structure of the controller.

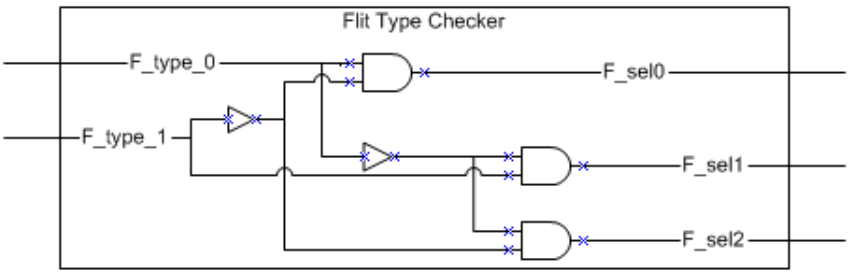


Figure 6.6: Flit type checker.

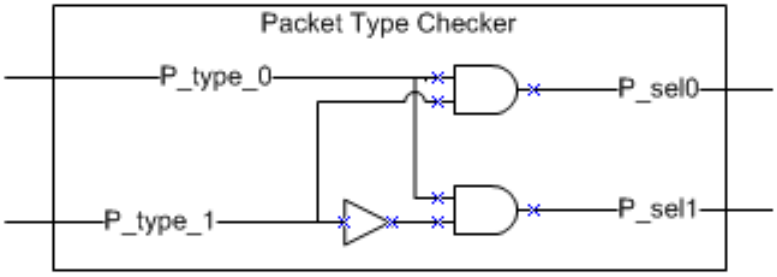


Figure 6.7: Packet type checker.

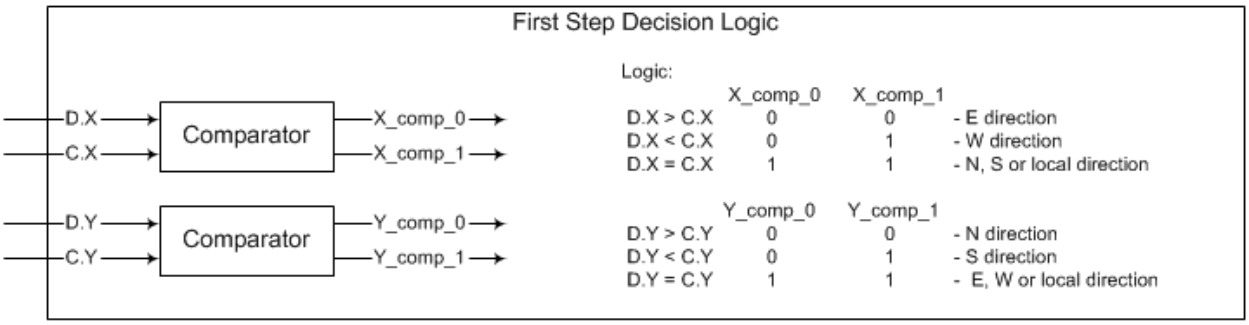


Figure 6.8: First step decision logic.

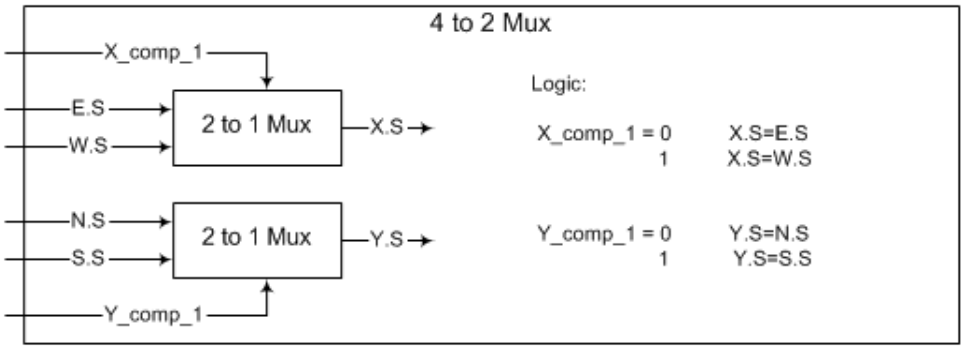


Figure 6.9: 4 to 2 Mux.

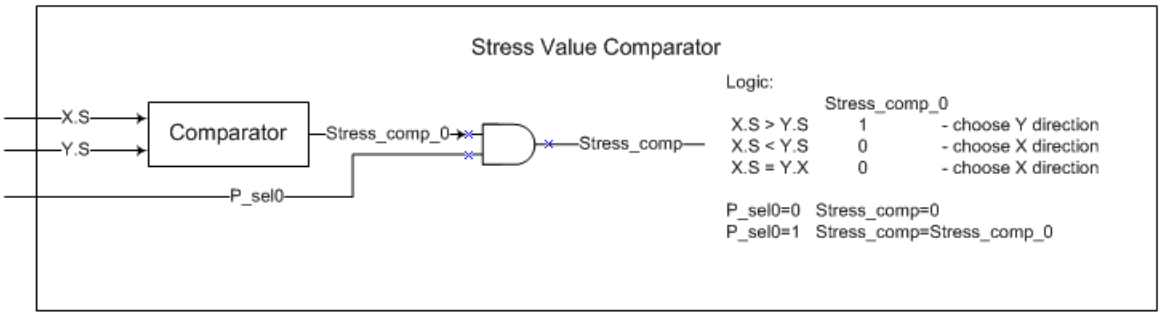
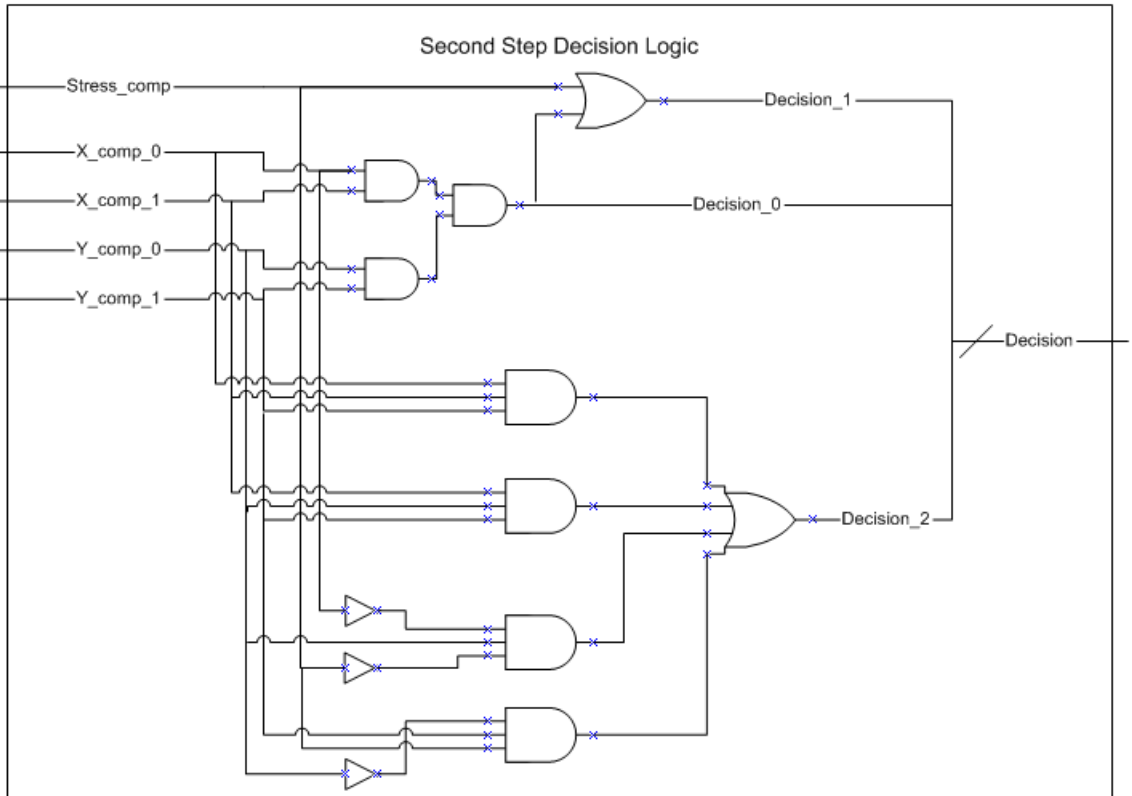


Figure 6.10: Stress value comparator.



Logic:

Inputs					Outputs			
X_comp_0	X_comp_1	Y_comp_0	Y_comp_1	Stress_comp	Decision_0	Decision_1	Decision_2	
0	0	0	0	0	0	0	0	- Choose E direction
0	0	0	1	0				
0	0	1	1	X				
0	1	0	0	0	0	0	1	- Choose W direction
0	1	0	1	0				
0	1	1	1	X				
0	0	0	0	1	0	1	0	- Choose N direction
0	1	0	0	1				
1	1	0	0	X				
0	0	0	1	1	0	1	1	- Choose S direction
0	1	0	1	1				
1	1	0	1	X				
1	1	1	1	X	1	1	1	- Choose local direction

Figure 6.11: Second step decision logic.

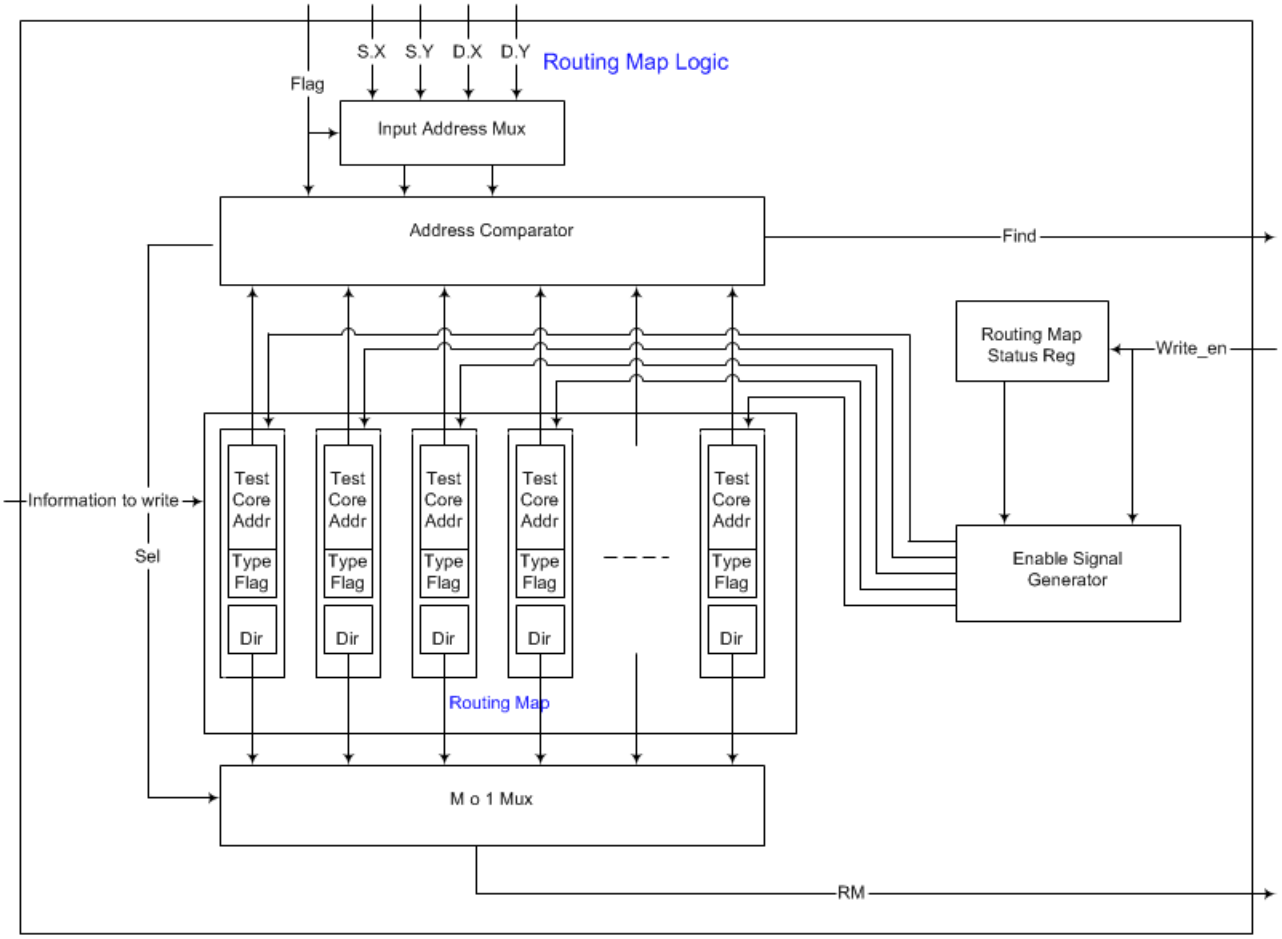


Figure 6.12: Routing map logic.

Write Enable Signal Generator

Logic:

	Inputs				Outputs				
	in_dir_0	in_dir_1	in_dir_2	F_sel0	En_E	En_W	En_N	En_S	En_L
Flit from East	0	0	0	1	1	0	0	0	0
Flit from West	0	0	1	1	0	1	0	0	0
Flit from North	0	1	0	1	0	0	1	0	0
Flit from South	0	1	1	1	0	0	0	1	0
Flit from Local	1	1	1	1	0	0	0	0	1
	x	x	x	0	0	0	0	0	0

Figure 6.13: Enable signal generator.

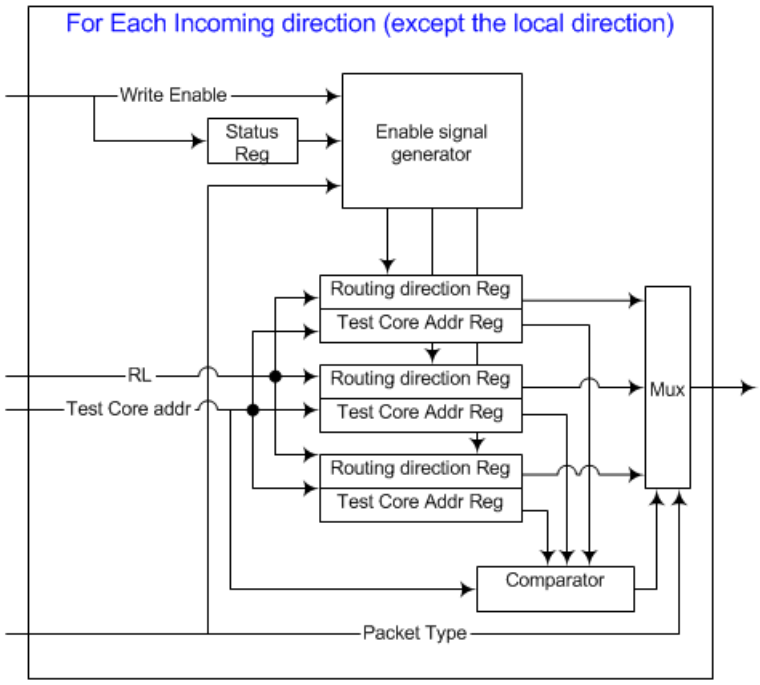


Figure 6.14: Partial Logic of the Optimizaed Routing Table.

Chapter 7

Conclusion and Future Works

7.1 Conclusions

SoC platforms based on large, complex multiprocessor are already well into existence. According to common expectations and technology roadmaps, the emergence of billion-transistor chips is also around the corner. Wireless base stations, high-definition TV, and mobile handsets are just a few applications that have arisen because of multiprocessor SoCs. With such chips, the constraints for performance, power consumption, reliability, error tolerance and recovery, cost, and so forth can be extremely severe. One design characteristic that lies at the core of all these critical specifications is the on-chip interconnection network. The NoC paradigm is one, if not the only one, to enable the integration of an exceedingly large number of computational, logic, and storage blocks in a single chip (known as a SoC).

The adoption and deployment of NoCs face important issues related to design and test methodologies. One big design challenge is on router design. Routers need to be adaptive to achieve high performance for high network traffic. Meanwhile, livelock and deadlock-free are also important to guarantee the service. In a NoC, area and logic efficiency are evenly important merits for router design. Works have been done to develop efficient routing methods for mesh structure computer networks and NoC architectures [10]-[20]. Although they have achieved some progress, there are still limitations on the adaptiveness of the routing method and complexity of the router architecture.

In Chapter 3, we have presented our works in router design. A novel routing method (DyXY)

was presented to achieve efficient network communication for normal mode NoC operations. The DyXY routing method provides adaptive routing based on congestion conditions in the proximity, and ensures deadlock-free and livelock-free features at the same time. Analytical models based on queuing theory were developed for both static XY routing and DyXY routing to evaluate their performance for a two-dimensional mesh NoC architecture. Extensive simulation was done to validate the analytical models, and it was observed that the simulation results match very well with the analytical results. To further evaluate the performance of DyXY, we compared it with both static XY routing and odd-even routing under different traffic patterns, and it was shown that the DyXY routing method can achieve the best performance.

Although many researches have worked on the design and implementation issues of NoCs, few of them paid enough attention to the testing issues. Reuse of the existing on-chip communication resources, such as routers and channels, is critical to avoid additional area overhead [21]. But reusing NoC resources efficiently is also challenging because the design of routers and channels in an on-chip network is optimized for communication in mission-mode, not for test. For example, there may be a mismatch between the available network channel width and the core scan chain width (which is usually equal to the Test Access Mechanism (TAM) width for traditional SoC architectures), and this can adversely affect test efficiency and test cost.

Some works have been done for embedded core testing based on the new NoC architecture. In [21], two methods for the test of core-based systems were introduced by using an on-chip network, and advantages in reducing test time, area overhead and pin-overhead were shown by experimental results. In [24], the authors extended the results of a previous on-chip network research to a test scheduling algorithm with power constraints considered. In this algorithm, scheduling was based on every single data flit, and the test pipeline for a core can be interrupted. Since this is not applicable for the non-preemptive case, an improved test scheduling algorithm with built-in self test (BIST) and precedence constrains was further proposed in [25]. For all these test scheduling algorithms, each data flit contains only a single bit for each wrapper scan chain of the embedded core. For example, if the bit-width of a data flit is 32 bits and there are 10 wrapper scan chains in an embedded core, then only 10 bits of each data flit are used to apply test patterns. Since this may result in a huge waste of packets, a further improved test scheduling algorithm has been proposed in [26]. The algorithm allows a data flit to contain multiple bits for each wrapper scan

chain. To support test under this new packet format, the authors proposed to use on-chip clocking to speed up the test data transfer for certain cores by faster clocks, and use slower clocks for other cores to limit the power consumption. An algorithm was presented to determine the clock rate distribution among the cores. Although these proposed test scheduling methods have achieved good performance, none of them addressed the wrapper scan chain configuration issue. Therefore, the utilization of the on-chip network is still limited, and the testing time for the entire NoC using these methods is still not efficient enough.

In Chapter 4, a heuristic wrapper scan chain configuration method has been proposed based on the concept of multiple-data-flit-format (MDFF) test data transportation concept. With MDFF, a data flit can contain multiple bits for each wrapper scan chain, instead of only 1 bit/chain in traditional test application methods [22]-[35]. Also, the data flits for a core can have several formats to adapt the entire test process to different numbers of unfilled scan chains, for the maximum utilization of network channels. Since a data flit may contain more than one bit of each test pattern (assume x bits) for each wrapper scan chain of an embedded core, one such a data flit only needs to be sent to the core every x clock cycles, instead of every cycle. Therefore, the free clock cycles can be used to apply test patterns for other embedded cores. The MDFF concept and the heuristic wrapper scan configuration method can reduce the waste of data flits for testing cores in a NoC, and formed a good foundation for efficient embedded core testing. However, an appropriate test scheduling method is needed to realize the reduction of total test application time of NoCs.

In Chapter 5, a dedicated test scheduling algorithm is proposed to work together with the wrapper scan chain configuration method and MDFF concept proposed in [47]. The test scheduling algorithm can realize test pattern application interleaving for different cores, and testing of different cores for a NoC in parallel. Instead of only considering channel capacity as most traditional methods, the proposed test scheduling method takes into account the data flit interleaving issues, such that flits (of test patterns or test responses) for different cores can be well interleaved without conflicts in channels and routers of the NoC. Therefore, test time of the whole system can be minimized with a specified test power consumption. By comparing the results with other published works, it has been demonstrated that the proposed test scheduling method can achieve significant improvement on the test time for the entire NoC.

To support testing mode operations, router design issues under testing mode were also

explored in Chapter 6, and a complete router architecture which can support both normal mode and testing mode operations has been presented. With all these works done, we finally have a complete and efficient test strategy for NoC embedded core testing, with the support of a high-performance router designed for both normal mode and testing mode operations. The total test application time of a NoC can be minimized, and the network traffic can be handled efficiently both in normal mode and testing mode.

7.2 Future Works

In this dissertation, we presented our works in NoC embedded core testing and router design. However, testing of a NoC includes not only the embedded core testing, but also testing of the on-chip network, i.e., the routers and the interconnects. In this section, we will state the research problems for on-chip network testing, review current works in this area, and propose possible solutions to solve these problems.

A. Testing of Routers

A.1 Related Works

To our best knowledge, so far, there is only one work addressing the router testing problem for a NoC architecture [48]. The testing of NoC routers is divided into two parts: the router logic blocks (RLBs) and the FIFOs [48]. For RLB testing, two methods, *unicast* and *multicast* are proposed. A test source is directly connected to a corner router (in a 2-D mesh architecture), and it stores the test patterns and expected outputs. Each router has two operation modes, the normal mode and the test mode. In normal mode, a router transfer test patterns and expected outputs to the next designated neighbor. In test mode, the router performs test using the test pattern received and the results are compared with the expected outputs locally. Using the unicast strategy, RLBs of the routers are tested in a sequential-recursive manner, one block at a time, using the upstream routers to transport test data to the RLB under test (Figure 7.1 (a)). Using the multicast strategy, the testing process can be performed on multiple routers concurrently, i.e., once a router is verified to be functional correct, it can be used to broadcast the test patterns to all adjacent neighbors as shown in Figure 7.1 (b). The test time of the entire NoC using both strategies were calculated,

and it was shown that the multicast strategy performs better than the unicast strategy especially when network size is large.

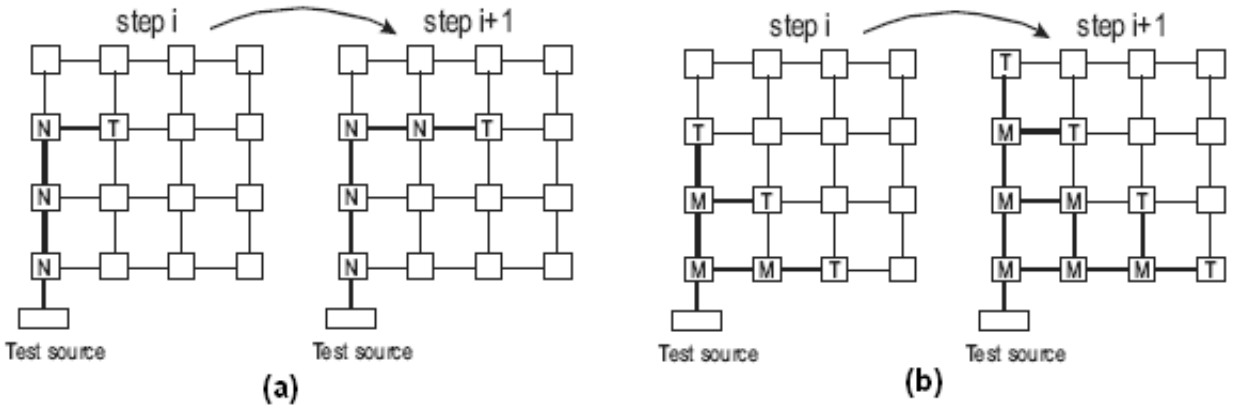


Figure 7.1: NoC router test strategies (a) unicast and (b) multicast [48].

For the test of FIFO blocks in each router, the authors in [48] proposed a distributed BIST methods as shown in Figure 7.2. In this scheme, the read/write mechanism, the control circuitry and the test data source are shared among multiple FIFO blocks, whereas the local response analyzers are distributed, one for each FIFO.

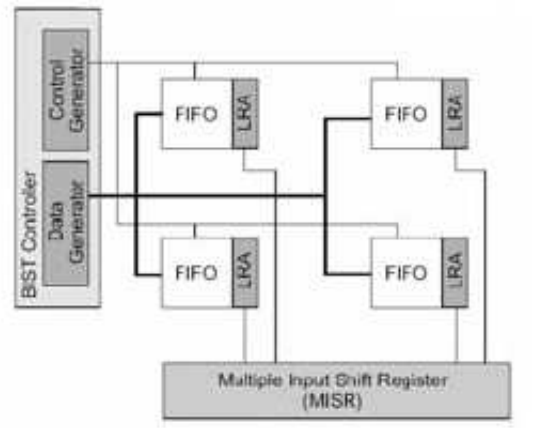


Figure 7.2: Distributed mechanism for FIFO test [48].

This work was the first one on NoC router test and stated many valuable points in this research area. However, it also has some limitations as summarized below.

- First, the testing of routers is based on the assumption that the all interconnects are in good

function which cannot be guaranteed in such a highly integrated interconnect network.

- Second, although the multicast strategy is superior to the unicast strategy, there is still space to optimize. With both strategies, to test a router, test patterns need to be transferred from the external test source to each destination router. This procedure is repeated for each router and takes a significant part in the overall testing time of all routers for the entire NoC. However, in NoC, each router has exactly the same architecture and the same RLB block, i.e., the test patterns for the RLBs of all routers are the same. Hence, there is no reason to repeat this test pattern transfer process. Also, each router may need multiple test patterns for functional test. To make use of the parallel structure of NoC, different routers should be able to conduct test concurrently. An optimized test methodology by pipelining will be proposed later in this section.

- Finally, although BIST methods for FIFO block testing is mature, there still exist some implementation issues for applying distributed BIST for FIFO testing in a NoC architecture as listed below:

  - How many BIST controllers are sufficient for a NoC?
  - How to map routers to the BIST controllers?
  - Should the interconnection between BIST controllers and routers use existing interconnects or additional wires?

  The authors in [48] did not consider these questions, therefore, more works need to be done to realize the idea of distributed BIST methodology to a practical implementation.

A.2 Possible Optimizations

To further optimize the multicast test strategy, we propose a new test methodology named *pipelined-multicast*. The main idea of the pipelined-multicast is shown in Figure 7.3.

Since each router may need multiple test patterns, once a router finishes one test pattern and is verified to be correct under this test pattern, it propagates the test pattern to all the direct downstream neighbors and let them begin to conduct testing. The testing time for all routers in a $m \times m$ 2-D mesh NoC using multicast strategy is shown in Equation 5.1, where k is the total

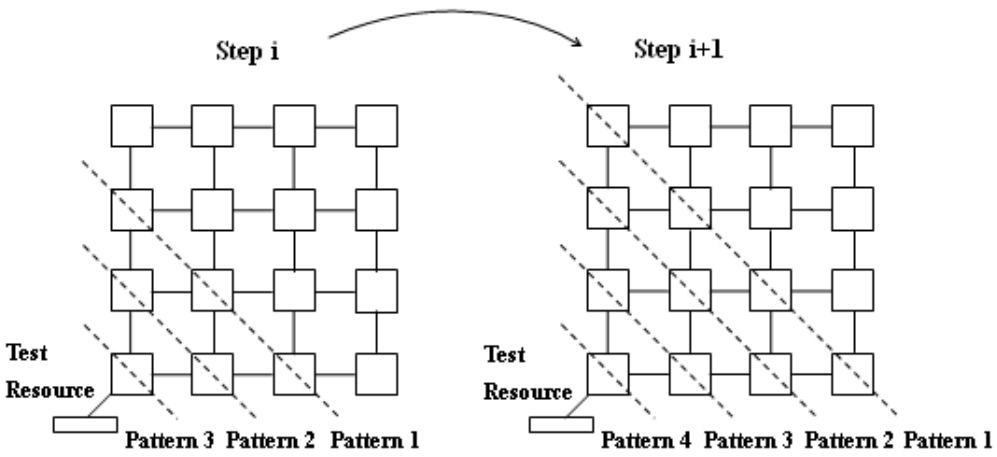


Figure 7.3: Pipelined-multicast test strategy.

number of test patterns for a RLB in each router, T_r is the testing time for a single test pattern of a standalone RLB, and N_p is the time needed for a single test pattern to traverse a router and an inter-router line.

$$T^m_{mesh} = k \times (2m-1) \times T_r + k \times (2m-1)(m-2)(N_p+1) \tag{7.1}$$

However, using pipelined-multicast, the total testing time can be reduced to

$$T^m_{mesh} = (k+1) \times T_r + (2m-2)(N_p+1). \tag{7.2}$$

Of course, to support the pipelined-multicast strategy, more control logic circuits need to be added to each router. Therefore, detailed analysis of design complexity, area overhead and test time reduction needs to be performed to verify this new test strategy.

B. Testing of Interconnects

B.1 Fault Models for Interconnects

Many works have been done for interconnect testing in traditional SoC architectures. Traditional fault models in interconnect testing includes stuck-at, open faults on a wire and short faults among wires (wired-OR or wired-AND). In [49], it was shown that $\lceil log_2 n \rceil$ test patterns are necessary and sufficient for detecting all kinds of short faults on a board with n wires. The input bit

stream proposed in [49] contains all '0's and all '1' streams that cannot detect stuck-at-0 faults and stuck-at-1 faults respectively. In [50], the test method was modified to exclude these two streams to cover the stuck-at faults, and the test pattern length becomes $\lceil log_2(n+2) \rceil$. Further, a fault diagnosis method was proposed in [51] which requires $\lceil log_2(n+2) \rceil$ test patterns following another set of $\lceil log_2(n+2) \rceil$ test patterns that are complement of the original set.

With deep sub-micron technology and high clock frequencies in the GHz range, signal integrity problems due to increasing cross-coupling capacitance and mutual inductance becomes significant, and may cause adverse effect on the proper functioning and performance of the chip. To analyze the signal integrity problem, a new fault model, named the Maximal Aggressor Fault Model (MA fault model) was proposed in [52]. In the MA model, it is assumed that the signal traveling on a wire (victim wire) may be affected by signals/transitions on other wires (aggressor wires) in its neighborhood. The coupling can be generalized by a generic coupling component, and the effect could lead to any of the following four crosstalk errors on the victim wire: positive glitch (g_p), negative glitch(g_n), rising delay(d_r) and falling delay (d_f). The transitions needed on the aggressors and victim wires to produce the maximum error effect for all four error types on the victim wire Y_i were also shown in the paper (Figure 7.4), and it was stated that these transitions constitute a necessary and sufficient MA tests to detect the corresponding four crosstalk faults for the victim wire Y_i.

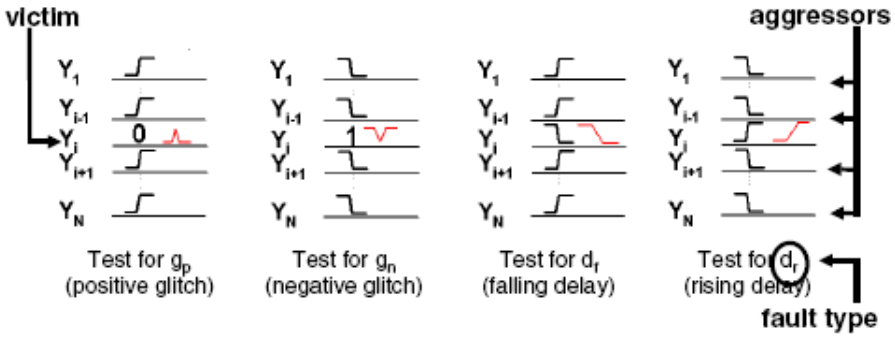


Figure 7.4: Maximal Aggressor Fault Model [53].

In the MA model, only capacitance coupling was taken into account. However, when mutual inductance comes into play, the presented MA test sets may not reflect the worst case and the test patterns may not create maximal integrity loss [54]-[56]. To solve this problem, another fault model

named Multiple Transition Model (MT) was proposed in [57]. MT essentially is a superset of MA patterns, and covers all possible transitions on the interconnects to stimulate integrity losses. The main idea of MT is having single victim, limited number of aggressors, full transition on victim, and multiple transition aggressors. An example of MT and MA test patterns for a 3-line interconnect is shown in Figure 5.5.

Quiescent at 0 x0x x0x	Transition 0→1 x0x x1x	Quiescent at 1 x1x x1x	Transition 1→0 x1x x0x
000 101	000 111	010 111	010 101
001 100	001 110	011 110	011 100
100 001	100 011	110 011	110 001
101 000	101 010	111 010	111 000

Figure 7.5: Transitions that MT and MA (shaded) models generated for a 3-line interconnect [57].

B.2 Interconnect Testing Methodologies for Traditional SoCs

Most of the works for SoC interconnect testing are aimed to develop self-test schemes, and many of them have made use of the boundary scan cells. Testing interconnect crosstalk defects using on-chip processor [58], a BIST to test long interconnects for signal integrity [59], and using boundary scan and IDDT for testing bus [60] are some of the proposed methods.

In [61], the author proposed to insert dedicated interconnect self-test structures in the SoCs to generate vectors for full coverage of crosstalk defects based on the MA model. However, this method has a prohibitively high area overhead. To reduce this overhead, a low-cost self-test scheme called LI-BIST was proposed in [53]. In LI-BIST, the existing LFSR structure was modified to generate high-quality tests for interconnect crosstalk defects, and thus the area overhead and interconnect power can be reduced.

BIST test pattern generators for board-level interconnect testing and delay testing were proposed in [62] and [63], respectively. A modified boundary scan cell using an additional level sensitive latch (called Early Capture Latch or ECL) was proposed in [63] for delay fault testing.

An extended JTAG was proposed to test SoC interconnects for signal integrity in [64] and [65]. The fault model used in [64] was the MA model, where MA test patterns are generated and applied to the interconnects by modified boundary scan cells placed at the output of a core. The other modified boundary scan cells are placed at the input of a core (at the end of interconnects) to collect the integrity loss information. Since the MA model does not count the inductance coupling effect, an optimized work based on the a fault model including inductance coupling was proposed in [65]. Similar to [64], this work also assumed that test patterns were generated externally and then scanned in. The scan-in process takes a long time and this becomes a major drawback of the work in [64] and [65]. In [57], an optimized fault model including inductance effects (MT) was introduced, and a new test pattern generation architecture was proposed to generate and apply test patterns almost at the speed of test clock.

B.3 Interconnect Testing Problems in NoC

Testing of interconnects in a NoC has similarities to that in a traditional SoC, however, it also has some significant differences summarized as following:

- Different from that in a traditional SoC, there is no global TAM in a NoC.
- Interconnects in a NoC consist of a large number of regular structured short wires. From this aspect, they are analogous to the segmented buses or the interconnects in a FPGA.
- Interconnects in a NoC include interconnects between neighboring routers and interconnects between each router and its local core. Each router in a core should have the same clock frequency, however, embedded cores in a NoC may work in different clock domains. Therefore, the clock frequency of an embedded core may be different from that of its local router, and synchronization logics are required to be inserted between them. In this case, testing of the interconnects between these core and router pairs also becomes more complex, since synchronization issues must be taken into consideration. Special cares are required to test the asynchronous interconnect fabric, e.g., insertion of scan-latches to break feedback loops [66].
- In a traditional SoC, interconnect testing can make use of the existing boundary scan cells and BIST logic circuits of the embedded cores. In a NoC, most interconnects are between

neighboring routers. Since routers are simple logic blocks, adding boundary scan cells and BIST logic to each router may cause too much area overhead, and thus may not be practical.

Due to these differences, although we can refer the fault models and some testing ideas in traditional SoC interconnect testing, no existing method can be directly applied to NoC interconnect testing.

Compared with plenty of works in embedded core testing, little research has been done in interconnect testing in a NoC architecture. The work in [67] is so far the only one we have found to address this problem. In this work, the authors stated that NoC interconnect testing is analogous to FPGA interconnect testing, and the most important thing is to configure all the inter-router links to several concatenated buses. The authors proposed a priority-based approach to derive the minimum number of configurations for testing interconnects for any 2-D mesh NoC. The approach achieves zero redundancy in testing interconnects and thus reduces test cost. By using this approach, all inter-router links in a 2-D mesh NoC can be covered by two concatenated buses C_1 and C_2 as shown in Figure 7.6, where C_1 consists of C_{11} and C_{12}, and C_2 consists of C_{21} and C_{22}.

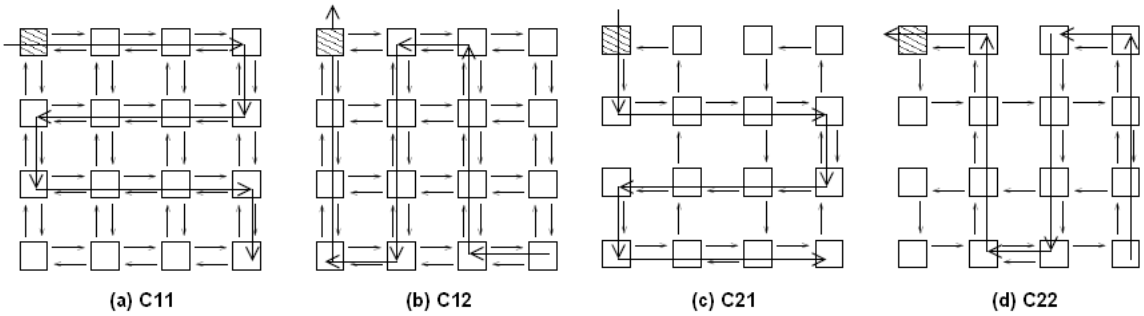


Figure 7.6: Test configuration for interconnect test [67].

Although the developed test configuration is efficient, this work only addressed the segmented property of the NoC interconnect without considering the complexity feature of the interconnect. Therefore, the testing problem was simplified too much. On the other hand, using the proposed configuration, the test is essentially a sequential test, which takes long testing time and does not take advantage of the parallel structure of NoC architectures. Further, the work is based on the assumption that the routers are functionally correct. However, the testing of routers cannot

be performed without the aid of interconnects functionally, unless we use BIST for each router which is not practical.

B.4 Possible Solutions

Based on the previous discussions, there are two possible solutions for NoC interconnect testing.

- Concurrent Distributed BIST

  As described previously, the FIFO blocks in routers need to be tested using distributed BIST method. A possible solution is to make use of these existing BIST structure and modify it to generate high-quality test patterns for interconnect testing. Thus, interconnects in a NoC can be self-tested and different part of interconnects can be tested concurrently.

- Pipelined-multicast test combined with router testing

  If a BIST method for interconnect testing is not feasible, test patterns for interconnects have to be transferred from the external test source. Since router testing needs to make use of the interconnects, the other possible solution is to combine the testing of interconnects and routers together. The test pattern application method can use the pipeline-multicast method proposed before. After testing a router, the interconnects between this router and its direct downstream neighbor are tested. Then, test patterns are transferred to the neighbors for functional test, and so on.

NoC on-chip network testing is a complicated problem, and there are many implementation issues to be considered to develop a practical and efficient solution. However, it is also a very important problem to solve. Only with the solution of this problem, a complete NoC test strategy can be developed.

Bibliography

[1] J. Henkel, W. Wolf, and S. Chakradhar. On-chip networks: a scalable, communication-centric embedded system design paradigm. In *Proc. 17th International Conference on VLSI Design*, pages 845–851, 2004.

[2] L. Benini and G. D. Micheli. Networks on chips: a new SOC paradigm. *IEEE computer*, 35:70–78, Jan 2002.

[3] Semiconductor Industry Association. The international technology roadmap for semiconductors: 2003 edition, http://public.itrs.net.

[4] Semiconductor Industry Association. The international technology roadmap for semiconductors: 2004 update, http://public.itrs.net.

[5] J. M. Rabaey, A. Chandrakasan, and B. Nikolic. *Digital Integrated Circuits: A Design Perspective*. Prentice Hall, Upper Saddle River, New Jersey, USA, 2^{nd} edition, 2003.

[6] W. J. Dally and B. Towles. Route packets, not wires: On-chip interconnection networks. In *Design Automation Conf.*, pages 684–689, 2001.

[7] P. P. Pandi, C. Grecu, A. Ivanov, R. Saleh, , and G. De Micheli. Design, synthesis, and test of network on chips. *Trans. IEEE Design and Test*, 22:404–413, Sep. 2005.

[8] C. A. Zebenes and A. A. Susin. Socin: A parametric and scalable network on chip. In *Proc. the 16th Symposium on Integrated Circuits and Systems Design*, pages 169–174, 2003.

[9] S. Kumar, A. Jantsch, and et.al. A network on chip architecture and design methodology. In *Proc. the IEEE Computer Society Annual Symposium on VLSI*, pages 105–112, 2002.

[10] W. J. Dally. Virtual-channel flow control. *IEEE Trans. Parallel and Distributed Systems*, 3:194–205, Mar 1992.

[11] Y. M. Boura and C. R. Das. Efficient fully adaptive wormhole routing in n-dimensional meshes. In *Proc. Int'l Conf. Distributed Computing Systems*, pages 589–596, 1994.

[12] A. A. Chien and J. H. Kim. Planar-adaptive routing: Low-cost adaptive networks for multiprocessors. *Journal of the ACM*, 42:91–123, Jan 1995.

[13] J. Duato. A new theory of deadlock-free adaptive routing in wormhole networks. *IEEE Trans. Parallel and Distributed Systems*, 4:1320–1331, Dec 1993.

[14] C. J. Glass and L. M. Ni. Maximally fully adaptive routing in 2D meshes. In *Proc. 1992 Int'l Conf. Parallel Processing*, pages 101–104, 1992.

[15] C. J. Glass and L. M. Ni. The turn model for adaptive routing. *Journal of the ACM*, 41:874–902, Sept 1994.

[16] Intel Corporation. A touchstone delta system description. In *Intel Advanced Information*, 1991.

[17] G. M. Chiu. The odd-even turn model for adaptive routing. *IEEE Trans. on Parallel and Distributed Systems*, 11:729 – 738, July 2000.

[18] Y. M. Boura and C. R. Das. A class of partially adaptive routing algorithms for n-dimensional meshes. In *Proc. Int'l Conf. Parallel Processing*, pages 175–182, 1993.

[19] E. Nilsson, M. Millberg, J. Oberg, and A. Jantsch. Load distribution with the proximity congestion awareness in a network on chip. In *Proc. Design, Automation and Test in Europe Conference and Exhibition*, pages 1126 – 1127, 2003.

[20] J. C. Hu and R. Marculescu. DyAD - smart routing for networks-on-chip. In *Proc. Design Automation Conference*, pages 260 – 263, 2004.

[21] E. Cota, M. Kreutz, C. A. Zeferino, L. Carro, M. Lubaszewski, and A. Susin. The impact of NoC reuse on the testing of core-based systems. In *Proc. IEEE VLSI Test Symp.*, pages 128–133, 2003.

[22] V. Iyengar, K. Chakrabarty, and E. J. Marinissen. Test wrapper and test access mechanism co-optimization for system-on-chip. In *Proc. International Test Conference*, pages 1023 – 1032, Oct 2001.

[23] V. Iyengar, K. Chakrabarty, and E. J. Marinissen. Test access mechanism optimization, test scheduling, and tester data volume reduction for system-on-chip. *IEEE tranc. on Computers*, pages 1619 – 1632, Dec 2003.

[24] E. Cota, L. Carro, F. Wagner, and M. Lubaszewski. Power-aware NoC reuse on the testing of core-based systems. In *Proc. International Test Conference*, pages 612–621, 2003.

[25] C. Liu, E. Cota, H. Sharif, and D. K. Pradhan. Test scheduling for network-on-chip with BIST and precedence constraints. In *Proc. International Test Conference*, pages 1369–1378, 2004.

[26] C. Liu, V. Iyengar, J. Shi, and E. Cota. Power-aware test scheduling in network-on-chip using variable-rate on-chip clocking. In *Proc. IEEE VLSI Test Symp.*, pages 349–354, 2005.

[27] J. S. Kim, M. S. Hwang, S. Roh, J. Y. Lee, K. Lee, S. J. Lee, and H. J. Yoo. On-chip network based embedded core testing. In *Proc. IEEE International SoC Conference*, pages 223 – 226, 2004.

[28] A. M. Amory, M. Lubaszewski, F. G. Moraes, and E. I. Moren. Test time reduction reusing multiple processors in a network-on-chip based architecture. In *Proc. Design, Automation and Test in Europe*, pages 62 – 63, 2005.

[29] P. Guerrier and A. Greiner. A generic architecture for on-chip packet-switched interconnections. In *Proc. Design and Test in Europe*, pages 250–256, 2000.

[30] S. Kumar et al. A network on chip architecture and design methodology. In *Proc. Intl Symp. VLSI (ISVLSI)*, pages 117–124, 2002.

[31] F. Karim et al. An interconnect architecture for networking systems on chips. *IEEE Micro*, 22:36–45, Sep. 2002.

[32] P. P. Pande, C. Grecu, A. Ivanov, and R. Saleh. Design of a switch for network on chip applications. In *Proc. Intl Symp. Circuits and Systems (ISCAS)*, pages 217–220, 2003.

[33] P. P. Pande and A. Ivanov. Performance evaluation and design trade-offs for network-on-chip interconnect architectures. *Trans. IEEE Computers*, 54:1025–1040, Aug. 2005.

[34] Jos Duato, Sudhakar Yalamanchili, and Lionel Ni. *Interconnection networks : an engineering approach*. IEEE Computer Society Press, Los Alamitos, CA, 1997.

[35] M. Nourani and C. Papachristou. An ILP formulation to optimize test access mechanism in system-on-chip testing. In *Proc. International Test Conference*, pages 902–1000, 2000.

[36] M. Nourani and C. Papachristou. Structural fault testing of embedded cores using pipelining. *Jouranl Of Electronic Testing: Theory and Applications*, 15(1-2):129–144, Aug-Oct 1999.

[37] E. Cota, L. Carro, A. Orailoglu, and M. Lubaszewski. Test planning and design space exploration in core-based environment. In *Proc Design Automation and Test in Europe*, pages 483 – 490, 2002.

[38] S. Ravi, G. Lakshminarayana, and N. Jha. Testing of core-based systems-on-a-chip. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 20(3):426–439, Mar 2001.

[39] P. Varma and S. Bhatia. A structured test re-use methodology for core-based system chips. In *Proc. International Test Conference*, pages 294 – 302, 1998.

[40] C. Aktouf. A complete strategy for testing an on-chip multiprocessor architecture. *IEEE Design and Test of Computers*, 19(1):18–28, Jan-Feb 2002.

[41] M. Nahvi and A. Ivanov. A packet switching communication-based test access mechanism for system chips. In *Proc IEEE European Test Workshop*, pages 483 – 490, 2001.

[42] R. M. Chou, K. K. Saluja, and V. D. Agrawal. Scheduling tests for vlsi systems under power constraints. *IEEE Trans. on VLSI Systems*, 5(2):175–185, June 1997.

[43] V. Iyengar and K. Chakrabarty. Precedence-based, preemptive, and power-constrained test scheduling for system-ona-chip. In *Proc IEEE VLSI Test Symposium*, pages 368 – 374, 2001.

[44] P. M. Rosinger, B. M. AI-Hashimi, and N. Nicolici. Power profile manipulation: A new approach for reducing test application time under power constraints. *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, 21(10):1217–1225, Oct 2002.

[45] L. Kleinrock. *Queueing Systems*. John Wiley & Sons Inc., New York, 1976.

[46] http://www.extra.research.philips.com/itc02socbenchm. ITC'02 SoC test benchmarks (online).

[47] M. Li, W. B. Jone, and Q. A. Zeng. An efficient wrapper scan chain configuration method for network-on-chip testing. In *proc. of 2006 IEEE Computer Society Annual Symposium on VLSI(ISVLSI 2006), accepted to be published*, 2006.

[48] C. Grecu, P. Pande, B. Wang, A. Ivanov, and R. Saleh. Methodologies and algorithms for testing switch-based NoC interconnects. In *Proc. Intl Symp. on Defects and Fault Tolerance in VLSI Systems*, pages 217–220, 2005.

[49] W. K. Kautz. Testing of faults in wiring interconnects. *IEEE Trans. on Comp.*, 23:358–363, 1974.

[50] P. Goel and M. T. MacMohan. Electronic chip-inplace testing. In *Proc. Int. Test Conf.*, pages 83–90, 1982.

[51] P. T. Wagner. Interconnect testing with boundary scan. In *Proc. Int. Test Conf.*, pages 52–57, 1987.

[52] M. Cuviello, S. Dey, X. Bai, and Y. Zhao. Fault modeling and simulation for crosstalk in System-on-Chip interconnects. In *Proc. Int. Conf. on Computer Aided Design*, pages 297–303, 1999.

[53] Krishna Sekar Sujit Dey. LI-BIST: A low-cost self-test scheme for SoC logic cores and interconnects. In *Proc. VLSI Test Symposium*, pages 417–422, 2002.

[54] W. Chen, S. Gupta, and M. Breuer. Test generation for crosstalk-induced delay in integrated circuits. In *Proc. Int. Test Conf.*, pages 191–200, 1999.

[55] W. Chen, S. Gupta, and M. Breuer. Test generation in vlsi circuits for crosstalk noise. In *Proc. Int. Test Conf.*, pages 641–650, 1998.

[56] Attarha and M. Nourani. Test pattern generation for signal integrity faults on long interconnects. In *Proc. VLSI Test Symp.*, pages 336–341, 2002.

[57] M. H. Tehranipour, N. Ahmed, and M. Nourani. Multiple transition model and enhanced boundary scan architecture to test interconnects for signal integrity. In *Proc. Intl. Conf. on Computer Design*, pages 554–559, 2003.

[58] L. Chen, X. Bai, and S. Dey. Testing for interconnect crosstalk defects using on-chip embedded processor cores. In *Proc. Design Automation Conf.*, pages 317–322, 2001.

[59] M. Nourani and A. Attarha. Built-in self-test for signal integrity. In *Proc. Design Automation Conf.*, pages 792–797, 2001.

[60] S. Yang, C. Papachristou, and M. Tabib-Azar. Improving bus test via iddt and boundary scan. In *Proc. Design Automation Conf.*, pages 307–312, 2001.

[61] X. Bai, S. Dey, and J. Rajski. Self-test methodology for at-speed test of crosstalk in chip interconnects. In *Proc. Design Automation Conf.*, pages 619–624, 2000.

[62] C. Chiang and S. K. Gupta. Bist tpgs for faults in board level interconnect via boundary scan. In *Proc. VLSI Test Symposium*, pages 417–422, 1997.

[63] K. Lofstrom. Early capture for boundary scan timing measurement. In *Proc. Int. Test Conf.*, pages 417–422, 1996.

[64] N. Ahmed, M. H. Tehranipour, and M. Nourani. Extending JTAG for testing signal integrity in SoCs. In *Proc. Design, Automation and Test in Europe*, pages 218–223, 2003.

[65] M. H. Tehranipour, N. Ahmed, and M. Nourani. Testing SoC interconnects for signal integrity using boundary scan. In *Proc. VLSI Test Symposium*, pages 158–163, 2003.

[66] A. Efthymiou, W. J. Bainbridge, and D. Edwards. Adding testability to an asynchronous interconnect for GALS SOC. In *Proc. Asian Test Symposium*, pages 20–23, 2004.

[67] K. Maddi. A methodology for interconnect testing of Network-on-Chip, university of cincinnati, master thesis, 2005.