CONCURRENT ONLINE TESTING FOR MANY CORE SYSTEMS-ON-CHIPS

A Dissertation

by

JASON DANIEL LEE

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

December 2010

Major Subject: Computer Engineering

Concurrent Online Testing for Many Core Systems-on-Chips

CONCURRENT ONLINE TESTING FOR

MANY CORE SYSTEMS-ON-CHIPS

A Dissertation

by

JASON DANIEL LEE

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

Chair of Committee,	Rabi N. Mahapatra
Committee Members,	Duncan M. Walker
	Eun Jung Kim
	Seong G. Choi
Head of Department,	Valerie E. Taylor

December 2010

Major Subject: Computer Engineering

ABSTRACT

Concurrent Online Testing for Many Core Systems-on-Chips. (December 2010)

Jason Daniel Lee, B.S., Texas A&M University

Chair of Advisory Committee: Dr. Rabi N. Mahapatra

Shrinking transistor sizes have introduced new challenges and opportunities for system-on-chip (SoC) design and reliability. Smaller transistors are more susceptible to early lifetime failure and electronic wear-out, greatly reducing their reliable lifetimes. However, smaller transistors will also allow SoC to contain hundreds of processing cores and other infrastructure components with the potential for increased reliability through massive structural redundancy. Concurrent online testing (COLT) can provide sufficient reliability and availability to systems with this redundancy. COLT manages the process of testing a subset of processing cores while the rest of the system remains operational. This can be considered a temporary, graceful degradation of system performance that increases reliability while maintaining availability.

In this dissertation, techniques to assist COLT are proposed and analyzed. The techniques described in this dissertation focus on two major aspects of COLT feasibility: recovery time and test delivery costs. To reduce the time between failure and recovery, and thereby increase system availability, an anomaly-based test triggering unit (ATTU) is proposed to initiate COLT when anomalous network behavior is detected. Previous COLT techniques have relied on initiating tests periodically. However, determining the

testing period is based on a device's mean time between failures (MTBF), and calculating MTBF is exceedingly difficult and imprecise.

To address the test delivery costs associated with COLT, a distributed test vector storage (DTVS) technique is proposed to eliminate the dependency of test delivery costs on core location. Previous COLT techniques have relied on a single location to store test vectors, and it has been demonstrated that centralized storage of tests scales poorly as the number of cores per SoC grows. Assuming that the SoC organizes its processing cores with a regular topology, DTVS uses an interleaving technique to optimally distribute the test vectors across the entire chip. DTVS is analyzed both empirically and analytically, and a testing protocol using DTVS is described.

COLT is only feasible if the applications running concurrently are largely unaffected. The effect of COLT on application execution time is also measured in this dissertation, and an application-aware COLT protocol is proposed and analyzed. Application interference is greatly reduced through this technique.

DEDICATION

This work is dedicated to the memory of my mother, Karen Culotta Lee.

ACKNOWLEDGEMENTS

Without the guidance and support of my advisor, Dr. Mahapatra, this work would not have been possible. The Ph.D. process is rarely a smooth endeavor; however, Dr. Mahapatra's encouragement was constant. Additionally, he provided great opportunities for my professional and personal growth beyond my own research. Highlights included a multi-year assistantship working for the FAA and various aerospace companies to make commercial avionics safer and a trip to India to discuss the need for greater American-Indian research collaboration.

The complimentary expertise of each of my committee members was also extremely valuable in creating this work. Drs. Walker, Kim and Choi each provided useful information and knowledge which improved the quality of my work greatly. They were always happy to discuss any aspect of research with me.

Maintaining a group of close friends and co-workers throughout my time here has also played a vital role in successfully completing this degree. Thank you to the following people: Praveen, Suman, Nikhil, Suneil, Aalap, Javier, Scott, Ron and Brad.

On a more personal level, my wife, Jennifer Nicks, deserves all the credit in the world for keeping me focused with a positive attitude during all of the emotional highs and lows attempting this degree has brought me. I look forward to our next challenge.

Finally, I would like to thank my father, Steven Lee, for instilling the importance of education throughout my life. From an early age, it was our plan for me to obtain a Ph.D. That goal has been fulfilled.

NOMENCLATURE

ATE	Automatic Test Equipment
ATTU	Anomaly-Based Test Triggering Unit
BIST	Built-in Self Test
COLT	Concurrent Online Testing
CMOS	Complementary Metal–Oxide–Semiconductor
CNI	Core–Network Interface
DFT	Design for Test
DTVS	Distributed Test Vector Storage
ILP	Instruction Level Parallelism
IMA	Integrated Modular Avionics
IP	Intellectual Property
MTBF	Mean Time Between Failure
MTTR	Mean Time to Repair
NoC	Network-on-Chip
SoC	System-on-Chip
TLP	Thread Level Parallelism

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

CHAPTER I

INTRODUCTION

For decades, users of any computer system have enjoyed continued, exponential performance gains thanks to the diligent work of manufacturers able to predictably shrink and place more transistors into a single device—known simply as "Moore's Law." With access to more transistors, computer architects have been able to increase clock rates and create more sophisticated techniques to optimize the execution of software. In this environment, replacing hardware without any real change to software led to significant performance improvements. However, due to approaching physical and computational constraints, processor design has been forced to retreat from "single core" complexity and speed. Instead of one large, fast and complex processor executing software, many small, slow and simple processors within a system-on-chip (SoC) must now work together to execute software [1].

Although the path to "many core" processors is relatively straightforward from a hardware perspective, software must undergo revolutionary changes in order to maintain performance improvements. Today, commercial software engineers, computer architects and researchers continue to struggle on how best to transform historically sequential tasks into parallel-friendly ones apt for many core computation. Additionally, significant effort is being spent on adding hardware and software mechanisms to simplify the analysis and debug of this new, parallel software due to its immense complexity.

This dissertation follows the style of *IEEE Transactions on Computers*.

Although the shift from the monolithic processor to many core processors and the associated revolution in software will remain the salient change in computer design for the foreseeable future, there are other, equally significant changes occurring in computer system design as the shrinking of transistors reaches its physical limits. As transistors shrink, they become more difficult to manufacture consistently and reliably—transistors across a wafer or within a single die may vary significantly, resulting in uneven performance, shortened lifetimes or immediate failure [2]. Also, smaller transistors tend to "leak" more current during periods of inactivity, increasing power consumption and negating a major advantage of Complementary Metal–Oxide–Semiconductor (CMOS) design [3].

This work focuses on the reliability aspect of shrinking transistors in many core systems. Due to the increased fragility of transistors, lifetime-awareness has emerged as a new design consideration for many core SoC. The redundancy inherent in many core SoC design must be exploited to maximize the dependable lifetime of the entire system; therefore, individual cores within the SoC must be tested and identified as functional or non-functional throughout the operational lifetime of the system. This allows the SoC to gracefully degrade as cores fail, leaving functional cores active and separated from failed components. This process of testing cores throughout the SoC operational lifetime is called online testing, and online testing may be performed concurrently with the execution of user applications or separately as a dedicated task.

Concurrent online testing (COLT) takes advantage of the availability of the many redundant processing cores within the SoC to manage the testing of a subset of those

cores while the remainder continue to execute user applications [4], [5]. By allowing most of the SoC to remain operational at all times, availability and reliability are maximized throughout the device's lifetime.

This dissertation proposes and analyzes COLT techniques specifically designed for many core SoC. Specifically, a variety of techniques to assist COLT are proposed to:

- minimize system recovery time as failures occur,
- optimize the storage / performance tradeoff of applying tests,
- minimize the interference between testing and the execution of applications, and
- simplify the process of managing test scheduling and application.

These techniques can be encapsulated as lifetime management of SoC; system designers can create a tradeoff between COLT overhead and system lifetime.

It is expected that the techniques presented in this dissertation would be most attractive to safety-critical applications using SoC. Since reliability and availability are first order requirements for these systems, the overhead required to implement COLT is justified. Additionally, safety-critical applications are expected to adopt many core SoC due to the potential for the space, weight and power savings they provide due to their high level of integration. However, the adoption of these SoC in safety-critical applications is only possible if techniques such as COLT are feasible to implement and are effective. The goal of this work is to increase the feasibility and effectiveness of COLT, thereby allowing new technical capabilities in safety-critical applications.

The remainder of this chapter introduces the reader to the preliminaries of many core SoC design, including the transition from complex processing cores to simple

processing cores and the necessity of networks-on-chip (NoC). This chapter also summarizes the contribution of this work.

A. Many Core SoC Preliminaries

The transition to many core SoC is necessitated by three major obstacles, or "walls," that preclude the continued development of the monolithic processor [6]. These walls are:

- The Instruction Level Parallelism (ILP) Wall: The hardware mechanisms required to execute more instructions in parallel are becoming prohibitively expensive in terms of area, complexity and power.
- The Memory Wall: The gap between memory speeds and processor speeds continues to grow. Eventually, a vast majority of processing time will be spent waiting for data to be retrieved and sent to memory.
- The Power Wall: Increasing the clock frequency and transistor density within processors is leading to an unsustainable, exponential increase in power consumption and density.

It is important to note that these walls interact with each other and exacerbate the underlying problems that make monolithic processing impossible to continue. For example, as the gap between memory speed and processing speed increases, processors must compensate by increasing their instruction windows, the number of instructions that a processor analyzes at any one time for potential execution, to mask memory latency [7]. These continually growing instruction windows—an effort to increase the

ILP within a processor—result in greater power consumption and are only necessary due to the memory wall.

The emergence of many core SoC follows from these obstacles. Instead of attempting to execute a single thread of instructions as quickly as possible, it is now necessary to attempt to run as many separate threads as possible simultaneously. This will alleviate the need to increase clock frequency and ILP, thereby reducing power consumption within the SoC.

1. Processor Core Simplification

Historically, processor design's primary goal was to increase ILP. Software was viewed as a single, sequential thread of execution. From this perspective, a variety of techniques were created to allow more instructions to be evaluated and executed simultaneously in order to mask the effects of memory latency. These complementary techniques are commonly called out-of-order and speculative execution, and they typically require expensive overhead to be realized [7], [8]. However, these techniques experience diminishing returns in performance, and their costs grow super-linearly with the instruction window.

In response to this trend, processor designers now seek to increase thread level parallelism (TLP) rather than ILP. Instead of trying to look at a large window of instructions in an attempt to dodge memory latency, a processor can simply switch to a new thread of execution when an instruction is blocked due to a cache miss. An example of this design is the UltraSPARC T1 SoC, which employs eight multi-threaded processors [9].

Processor designs that exploit TLP rather than ILP are attractive for a variety of reasons, including the following:

- the elimination of expensive and complex out-of-order and speculative execution components such as register renaming, instruction windows, commit queues, re-order buffers and reservation stations
- smaller processor footprint
- can be easily integrated into a many core solution
- reduces cost of design, analysis and debug of hardware design compared to out-of-order processors

2. Networks-on-Chip

As the number of processing cores per chip continues to scale, on-chip busses will be replaced with networks-on-chip (NoC). Similar to inter-computer networks, NoC contain routers, links and network interfaces. Each core within the SoC is attached to a core–network interface (CNI), and these CNI communicate with each other via links and routers. Unlike on-chip busses which use broadcast messages to communicate, NoC use point-to-point communication to deliver information. Fig. 1 illustrates a 16-core SoC using a NoC organized as a 2D-mesh. Each intellectual property (IP) core interfaces with a CNI, and each CNI is connected to a router.

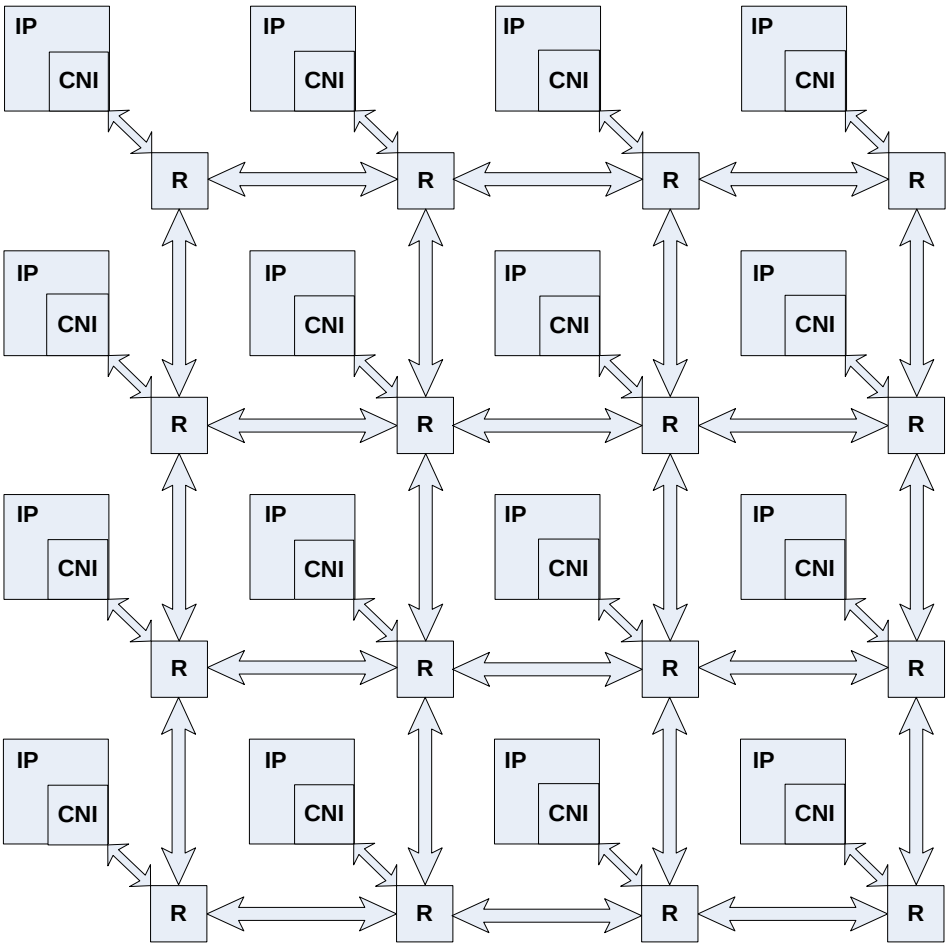

Fig. 1. Example 16-core SoC with a 2D 4x4 Mesh NoC Architecture

NoC provide many advantages over busses when the number of cores per chip exceeds more than a handful [10], [11]. These advantages include:

- Lower Contention
- Lower Power Consumption
- Standard Interfaces
- Shorter Wire Lengths
- Higher Throughput

- Fault Tolerance

Many of the COLT techniques proposed in this work depend on the existence of NoC; NoC provide a regular topology for communication between cores, and they also provide a standard protocol for communication. Additionally the NoC acts as a test access mechanism between test sources and test sinks.

B. Contributions

In previous research, COLT has relied on periodic testing to determine if a core within the SoC has failed. However, the time between failure and testing may be too great for some safety-critical applications depending on the testing period. An on-demand testing mechanism has been proposed and analyzed, and it is shown that this mechanism is effective and relatively inexpensive. This mechanism is described in Chapter III.

COLT depends on stored tests to achieve a sufficiently high level of coverage; however, there has been no serious analysis of how the delivery and storage of these tests scales with the number of cores per SoC. There has also been no effort in identifying techniques that can scale the storage and delivery costs of tests as SoC enter the many core era. Chapter IV details the application of a coding theory technique to the storage of test vectors, allowing for COLT to become scalable with many core SoC. A test protocol is proposed and analyzed, and the hardware overhead associated with this protocol is determined. To evaluate the proposed test storage scheme, real IP cores are used in the generation, storage and application of test vectors.

To date, there has been no analysis on the effect of COLT on the execution times of applications. Through the use of a system simulator employing a NoC as the communication infrastructure, this is the first work to measure the effect of COLT on NoC traffic and the effect of this increased traffic on software execution times. Based on these findings, an application-aware COLT protocol is proposed and analyzed. Chapter V details this application-aware COLT protocol.

CHAPTER II

RELIABILITY AND TESTING OF SAFETY-CRITICAL SOC

This work is primarily targeted at safety-critical applications using many core SoC. Although reliability and testing are important for any application, COLT is not expected to be necessary for most consumer applications due to their typical requirements: maximum performance, lowest cost and minimum power consumption. Safety-critical applications share many of the same requirements as most consumer applications; however, predictable and dependable operation are of highest importance. Therefore, it is expected that the overheads, both at design-time and during operation, are justifiable expenses for safety-critical applications.

In this chapter, safety-critical SoC design considerations are explored, a brief summary of classical electronic testing is provided, and previous work in the development COLT techniques are described. This background information is provided so that the reader may gain an appreciation for the benefits that COLT provides.

A. Design and Analysis of Safety-Critical SoC

Before any safety-critical system is certified for use, safety requirements must be established and failure possibilities must be understood. Safety requirements are typically quantified in terms of specific system dependability metrics, and failure possibilities are typically modeled as a composition of separate failure mechanisms. Therefore, it is important to understand how dependability is defined and measured, and

it is also important to understand how electronic failures mechanisms are changing as transistor sizes shrink.

1. Dependability Metrics

As defined in [12], dependability is a composition of a number of attributes: availability, reliability, safety, integrity and maintainability. Fig. 2 illustrates the relationship between these attributes and dependability. These attributes are defined as follows:

- Availability: the probability that a component is in a functional state
- Reliability: the probability that a component functions correctly
- Safety: the absence of unacceptable damage during component failure, typically defined by a regulatory agency
- Integrity: the inability of the component to be improperly modified
- Maintainability: the ability of the component to be modified and repaired

It can also be seen from Fig. 2 that dependability and security overlap. Although security is another extremely important consideration in safety-critical applications, it is outside the scope of this work.

Fig. 2. The Relationship Between Dependability and Security

This work is primarily concerned with availability and reliability. Lifetime management of safety-critical SoC aims to ensure that the system maintains maximum uptime and the highest probability of correct behavior throughout its operational lifetime.

Availability can be calculated as the ratio of mean time between failures (MTBF) and the sum of MTBF and the mean time to repair (MTTR). Therefore, as MTTR increases, availability decreases. It is for this reason that COLT must initiate tests and detect failures as quickly as possible. Fig. 3 illustrates the relationship between the components of MTTR and their associated events. Here, a fault occurs within the SoC at T_{fault}, and there is some delay between fault occurrence and the manifestation of a error at T_{error}. MTTR is composed of T_{delay}, the time between error manifestation and the beginning of testing; T_{test}, the time required to apply and determine if a fault has occurred; and $T_{reconfigure}$, the time required for the system to return to an error free state. A method to increase system availability with COLT is described in Chapter III.

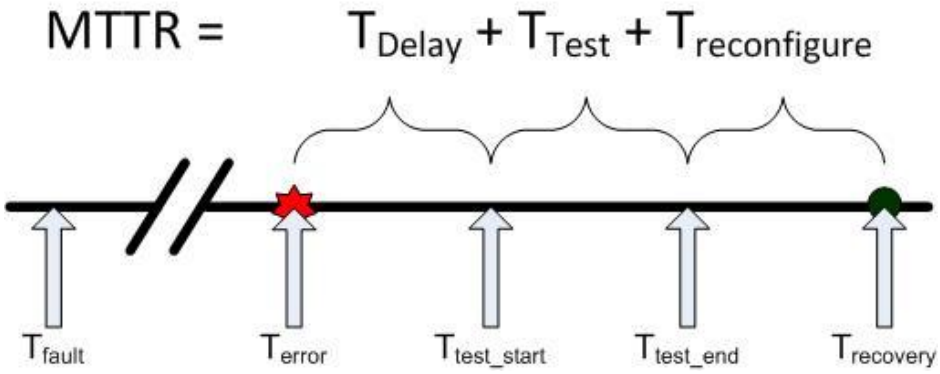

Fig. 3. The Components of Mean Time to Repair

To address reliability, it is important that the detection rate for faults is extremely high. If a fault is left undetected after error, it is extremely difficult to ensure that the system can operate correctly. Simply put, a system cannot fix a problem of which it is unaware.

2. Electronic Failure Mechanisms

There are a number of failure mechanisms that may affect transistors and interconnects throughout their lifetimes. The contribution of each failure mechanism depends on a variety of environmental, manufacturing and operational factors; therefore, it is extremely difficult to predict how transistors will fail during operation. The most well understood failure mechanisms are briefly described here, and COLT seeks to detect failures due to these mechanisms:

- Electromigration: the physical movement of material within an interconnect due to increased current density. See Fig. 4 for an illustration of electromigration.

- Hot Carrier Injection: permanent damage to the gate oxide due to highly kinetic, or "hot," electrons or holes due to the scaling of transistor geometries
- Negative Bias Temperature Instability: decreased performance affecting pMOSFET transistors, due to increased device temperatures and aggressive gate oxide manufacturing techniques
- Gate Oxide Breakdown: also known as dielectric breakdown, this is the permanent failure of the gate oxide to act as an electric barrier between source and drain in a transistor due to the decreased thickness of the gate oxide

The above list is not exhaustive. There are numerous other failure mechanisms that can potentially affect device behavior and can lead to failure; however, the above list represents the most influential failure mechanisms in typical applications. An in-depth treatment of each of the above wearout mechanisms is provided in [2], [13].

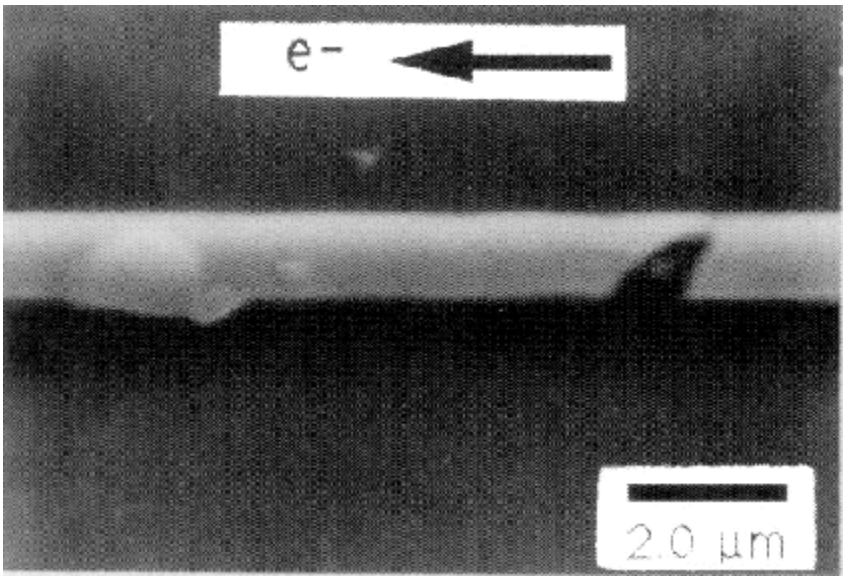

Fig. 4. Electromigration [2]

Each of these failure mechanisms contribute to the failure of transistors over time. As a whole, a population of devices will experience failure rates following a "bathtub" curve. In this curve, there are high failure rates at the beginning and the end of a device's life. These periods of high failure rates are called early lifetime failure (ELF) and electronic wearout, respectively.

Fig. 5 illustrates the bathtub curve effect for transistors with sizes of 180nm, 90nm and 65nm, based on findings reported in [14]. The largest transistors experience the longest lifetimes and lowest constant failure rate. It is interesting to note that early lifetime failures appear to be independent of transistor size; however, wearout failures are heavily dependent on transistor size. Constant failure rates, the bottom of the curve, also depend on transistor size; devices built with smaller transistors experience higher constant failure rates even during the most dependable period in their lifetime.

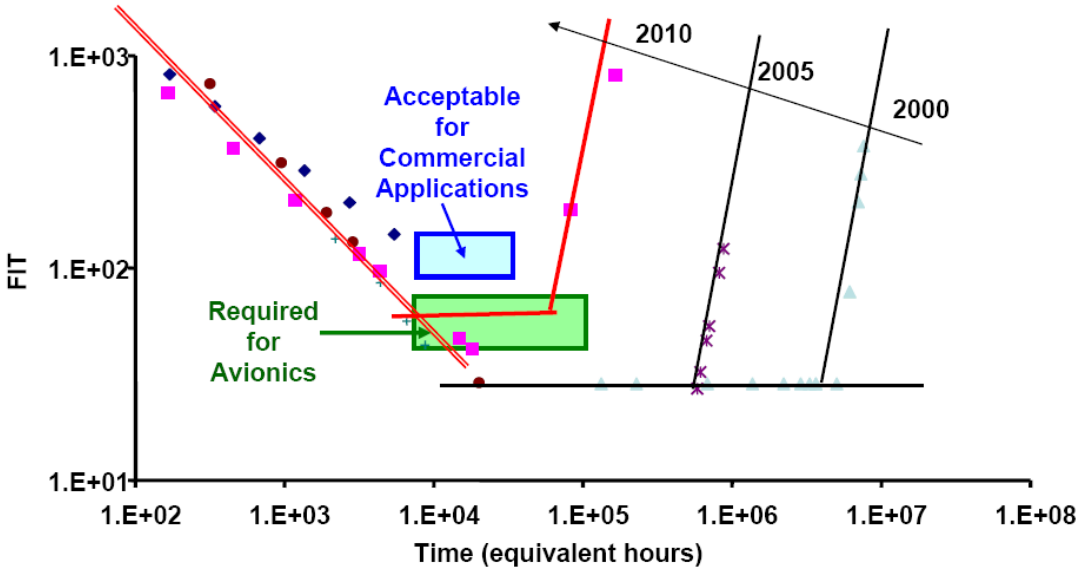

Fig. 5. The Bathtub Curve for a Variety of Transistor Sizes [14]

As [14] reports, the period of low failure rates within the bathtub curve is dramatically reducing as transistors shrink. At 65nm, it is expected that transistors will only experience a period of low failure rates lasting 10 years. Below 65nm, this period is expected to be much shorter. This will force safety-critical system designers to account for electronic wearout; many safety-critical applications depend on systems operating dependably for decades in order for a project to be economically justifiable.

B. Traditional Approaches to Testing

Electronic testing began as a discipline with the manufacturing of the first integrated circuits. In this section, a brief summary of electronic testing is provided to describe the strengths and weaknesses of the most widely adopted testing techniques in relation to COLT.

1. Manufacturer Testing

For manufacturer testing, external automatic test equipment (ATE) applies pre-generated tests at very high speeds. Each chip is typically tested in well under 10 seconds. These pre-generated tests are created from automatic test pattern generators (ATPG), which seek to build a minimum set of test vectors that achieve the highest possible fault coverage for a specified amount of fault models [15].

These fault models include the stuck-at and delay fault models which are of primary interest to this work. The delay fault model is very good at detecting electronic wearout. Electronic wearout typically manifests as increasing delay in transistors and

interconnects, making the delay fault model a natural choice for detecting this kind of failure [16], [17], [18], [19].

Manufacturer testing cannot be applied directly to COLT, since online testing must be performed in the field where no ATE exist. However, the ATPG used in manufacturer testing can be applied to COLT due to the high fault coverage achieved by these programs.

2. Built-in Self Test

Built-in Self Test (BIST) removes the need to use ATE to test the entire chip. Instead of an ATPG creating tests to be applied by an ATE, BIST is a small on-chip mechanism to generate test patterns. Test patterns are usually generated through simple means such as shift registers that can produce a very large amount of different test patterns. BIST are a good alternative to ATE-based manufacturer testing due to reduced costs; however, BIST often struggle with achieving the very high fault coverage rates produced by ATPG tests [16]. Therefore, a mixture of BIST-based and ATPG-based testing is often used for chip testing.

C. Recent Proposals in Concurrent Online Testing

1. Test Infrastructure IP Cores

In addition to processing, memory and interface IP cores being present within a many core SoC, it is expected that the SoC will also contain a variety of infrastructure IP cores to assist in the management of system operation [20]. These infrastructure IP cores can aid in system maintenance, debugging, yield increase, power management and fault

containment. Here, the role of infrastructure IP in increasing system reliability is of primary interest, and infrastructure IP that assist in COLT are described.

Using the NoC as a test delivery infrastructure, researchers have proposed reusing infrastructure IP (I-IP) designed originally for manufacturing testing as a tool for online testing of the system [20]. These online tests are mainly used for manufacturer testing to determine yields, and these tests are not meant to detect and capture wearout failures in the field.

Taking this concept a step further, [4], [5] constructed a Test Infrastructure IP (TI-IP) capable of managing test scheduling, delivery, and intrusion for concurrent on-line test (COLT) of the SoC. COLT allows cores in the SoC to be tested in the presence of normally executing applications to maximize system availability.

The original COLT scheme proposes that the test vectors are stored within the TI-IP. Due to the real-time constraints of these applications, COLT is extremely sensitive to application intrusion. The first effort to measure the effect of COLT on application intrusion is included in this dissertation, and these results are included in Chapter V.

Microprocessor pipeline on-line testing using distributed on-line BIST and periodic check-pointing was investigated and shown to be an effective technique in providing high reliability and availability at a reasonable area cost (5.8%) [22]. In this scheme, stuck-at fault test vectors are generated for processor components and stored in on-chip ROM. It should be noted that distributed BIST discussed in [22] is a different concept than DTVS discussed here. In [22], distributed BIST refers to separate BIST

mechanisms for the different pipeline components of a single microprocessor core, where distributed test vector storage refers to the separate BIST vector storage units across the entire SoC.

Yi, Makar and Mitra have proposed CASP: Concurrent Autonomous chip Self-test using stored test Patterns [16]. Similar to COLT, an on-chip test controller is proposed which manages test scheduling, test application and response comparison for processing cores of the OpenSPARC T1 chip multi-processor [23]. Their technique differs from COLT in that the test vectors are stored off-chip in a nearby flash or hard disk drive (HDD) storage system.

In [5], the authors identified an eventual problem with the proposed methods of COLT: scalability. As the number of cores per chip increases, the size of the NoC must also increase creating longer communication distances between cores. This increased distance translates into higher communication latency and power consumption. Fig. 6 illustrates the effect of hop distance and test volume on energy consumption when using COLT within a many core SoC. Hop distance produces the most profound effect on the energy consumption of test delivery. Therefore, it is extremely important to attempt to bound the distance that tests must travel during COLT. Bounding this test delivery distance is a major contribution of the work described in this dissertation, and methods to achieve this are proposed in Chapter IV.

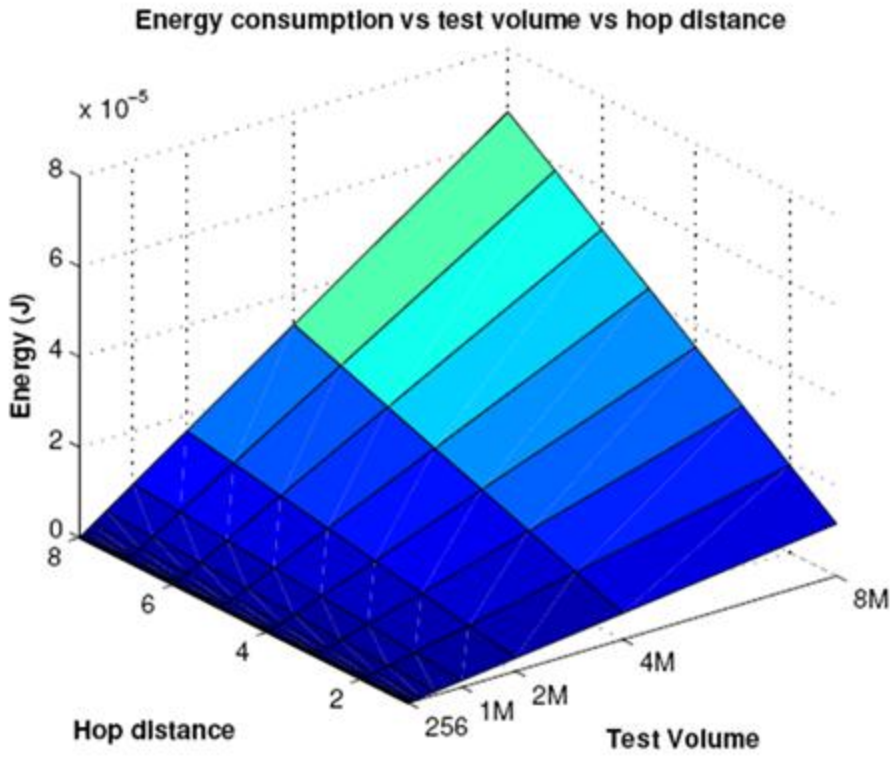

Fig. 6. The Effect of Hop Distance and Test Volume on Energy Consumption [5]

Test infrastructure IP cores have gained momentum from a variety of research efforts, and it is expected that future safety-critical SoC will employ some form of this technique to assist in extending the dependable lifetimes of these systems. Therefore, the research effort presented in this dissertation focuses on this method of COLT.

2. Software-Based Testing

Software-based self test (SBST) performs all testing completely in software. Both the test controller and actual test application are implemented by standard processor instructions. Therefore, SBST can be scheduled like any other software task,

allowing for the operating system to be aware of testing, and giving the system designer great flexibility in scheduling online tests.

SBST is completely non-intrusive in terms of hardware overhead; however, verifying that high-coverage levels of structural faults can be achieved strictly through the application of processor instructions is challenging. Several SBST techniques have been shown to achieve stuck-at fault coverage rates of 80% and above [24].

For delay fault coverage, a technique has been proposed and evaluated on a simple, custom processor [25]. SBST techniques have also been proposed for floating point units, where extremely high stuck-at fault coverage and low memory footprint have been achieved [26], [27].

Although SBST is useful for many applications, it is not expected that SBST will be a complete solution for providing sufficient reliability for safety-critical SoC. Since tests can only be generated from processor instructions, it is very difficult to achieve high fault coverage for fault models other than the stuck-at model. For models that are extremely useful in determining wearout, such as the delay fault model, hardware-based testing is expected to remain necessary. However, a combination of SBST and hardware-based COLT will probably yield an acceptable compromise between test development costs and system dependability.

CHAPTER III

ONLINE DETECTION OF CORE FAILURE WITH ANOMALY-BASED TEST TRIGGERING UNITS

A. Introduction

As performance requirements and power limitations for safety-critical applications continually increase, systems designers will be required to use more complex electronic components such as many-core systems-on-a-chip (SoC). Within the next decade, it is expected that a single SoC will contain hundreds of interdependent cores communicating across sophisticated networks-on-chip (NoC). Recently, the Tilera Corporation began shipping a 64-core SoC [28] using a NoC inspired by the MIT RAW on-chip network [29]. Although this chip does not target the safety-critical embedded system domain, it is only a matter of time before this level of technology will be required in safety-critical applications.

In addition to the aforementioned rising complexity of electronic hardware designs, operational lifetimes of these designs are threatened by a variety of factors that become more influential as feature sizes decrease to 45 nanometers and beyond. These factors include electro-migration, stress migration, time-dependant dielectric breakdown, and thermal cycling [30]. These trends present many challenges but also opportunities for system designers to create *quality electronic systems* through the use of fault-tolerance techniques such as redundancy and monitoring in both hardware and software.

Infrastructure IP cores (I-IP) have been proposed by research as a feasible approach to providing services to the SoC, including yield improvements, testing, error detection, and IP core configuration management [4], [20], [31]. Infrastructure blocks such as *Test Infrastructure IP* (TI-IP) are a clear choice to improve fault-tolerance of SoCs in hardware.

Most recently, employing TI-IPs in a NoC-enabled SoC has been shown to be an effective means for introducing concurrent on-line testing of systems [5]. *Concurrent on-line testing* (COLT) allows the various IP blocks within a SoC to be tested in-field and during normal operation. An effective and robust COLT implementation monitors IP block activity to minimize application intrusion. This manner of testing maximizes both *availability* and *reliability* of the system which are primary requirements in safety-critical applications.

IP block and NoC testing, whether done concurrently with applications or as an isolated process, is an expensive task in terms of power consumption and performance degradation [32]. Performance degradation is especially critical for concurrent testing when the applications are subject to real-time constraints. A better approach to solve this problem is to schedule testing as an *on-demand task* where components within the NoC monitor the system for possible errors and only trigger test requests from the TI-IP when errors are observed. However, triggering tests in this manner requires detection units that are accurate and inexpensive in terms of area and power consumption.

Due to intellectual property limitations, it cannot be assumed that system designers are allowed to observe the internal behaviors of the IP blocks directly.

Instead, the monitoring devices can only observe the *communication* among these IP blocks. This communication conforms to a pattern during normal operation of the applications; however, anomalies in this communication pattern will emerge when errors are present. Monitoring devices distributed across the NoC can localize the likely source of error and trigger more extensive tests from the TI-IP.

COLT services can be classified into two distinct types: periodic and event-triggered. Periodic monitoring does not consider the current state or behavior of a system, and instead relies on *a priori* knowledge of system failure rates. Establishing a suitable period for testing can be an ambiguous process when working with components whose *mean time between failure (MTBF)* is unknown. Also, testing disrupts the normal behavior of a chip, which is problematic for systems operating under real-time constraints. Therefore, testing should be reserved to times when abnormal behavior is observed.

When using COLT schemes employing a periodic testing strategy, system designers must balance two conflicting sets of constraints: system quality and reliability versus low power consumption and performance. Increasing the frequency of concurrent testing improves system quality and reliability—fault detection time is reduced, leading to quicker repair times and availability. Conversely, decreasing the frequency of concurrent testing improves power consumption and performance—performance improves because applications do not compete with tests for system resources as often, and power is conserved through the reduction of testing.

On-demand COLT is designed to minimize fault detection time and application intrusion by testing system components only when there is an indication of error. This solution accounts for both fault-tolerant and performance requirements which creates a system that maintains its quality throughout its lifetime. At the time of this research, no other such solution exists for concurrent testing.

This work introduces communication anomaly based error detection technique in NoCs with an efficient test triggering mechanism for concurrent on-line testing of SoCs. In particular, it makes the following contributions:

- Proposed an on-demand test triggering mechanism that optimizes testing overhead and intrusion by identifying potentially faulty IP blocks.
- Proposed a fault-tolerant system within the NoC which improves availability of safety-critical SoCs by reducing error detection time.
- For demonstration, SPEC CPU2006 [33] application benchmarks are executed on a NoC-enabled SoC simulator [34] with and without the presence of faults. Our experiments show that our test triggering unit detects 81% of errors and can initiate tests within 1ms of error detection, on average.

The remainder of Chapter II is organized as follows. Section B discusses previous and related research in the areas of networks-on-chip, fault-tolerance, and anomaly detection. Section C presents our proposed technique for error detection and on-demand test triggering within a NoC. Section D describes our experimental methodology and Section E presents the results of those experiments. Section F concludes with work and discusses future directions for this research.

B. Fault Tolerance and Anomaly-Based Error Detection

Current methods of fault tolerance within NoC-based SoC and anomaly-based error detection are described in this section for the purposes of background information. The anomaly-based test triggering unit presented in this chapter borrows from these previous techniques due to their effectiveness and efficiency.

1. NoC-Based Fault Tolerance

Among the many services a NoC provides, this research focuses on communication reliability and fault-tolerance support. Fault tolerant on-chip communication using stochastic flooding has been investigated in [35]. By using stochastic flooding during the transmission of messages over the NoC, spatial redundancy is introduced into the system which provides fault tolerance. However, stochastic flooding cannot overcome failures within the core or CNI, and its effectiveness depends on the NoC topology.

On-line fault detection and location in NoC interconnects were explored in [36]. In this work, faults within the NoC itself can be determined by using a coding technique to send messages. By sending messages in a constrained way, it can be determined whether a fault within the NoC has occured at a link or a switch. Debug support via NoC monitoring was introduced in [37]. The major focus of this work was to introduce debugging capabilities in NoC-based systems; however, the hardware probes required to achieve this can also be used for the purposes of fault tolerance.

Concurrent On-Line Test support for on-chip IP cores using Test Infrastructure-IPs was proposed in [4], [5]. Their technique utilized periodic test triggering for COLT initiation. However, determining the appropriate period to initiate testing is difficult since it is based on the failure rates of the device. Transistors may fail due to a variety of reasons that are extremely difficult to measure through simulation or through direct observation; transistor failures depend on process variation and on device operation. To address the inefficiencies of periodic COLT, this research proposes to use the communication behavior of the NoC to determine the optimal time to initiate testing of IP blocks.

2. Anomaly-Based Error Detection

Anomaly-based error detection has been used extensively across many application domains including distributed systems and Internet security [38]. However, to the best of our knowledge, no research has been done to investigate the effectiveness of anomaly-based error detection schemes to minimize test intrusion of COLT and to maximize availability of the SoC.

Anomaly detection belongs to the class of unsupervised learning techniques which do not rely on *a priori* information of the target system. This makes anomaly detection well suited for many-core SoCs; IP blocks provided by vendors may not provide adequate information to use an error detection technique which relies on *a priori* knowledge. Additionally, behavior may change depending on runtime characteristics or as software running on the system is modified over time. Relying on a priori knowledge under these conditions is costly and complex.

The primary challenge in using an anomaly detection based approach to error detection in SoCs is overhead management. Typical anomaly detectors require a prohibitive amount of storage and processing capabilities for an on-chip environment. The next section describes our efforts to minimize these overheads while maintaining a reasonable level of accuracy.

C. Test Triggering Mechanism

Our proposed test triggering mechanism relies on observing anomalies in the communication among the IP blocks. Anomaly detection has been used to observe a variety of network behaviors, such as *malicious intrusion*, *fault propagation*, and *congestion* [38]. It is the purpose of this research to determine the effectiveness of this technique for use in NoCs in order to efficiently trigger more robust testing of the IP blocks.

We assume that a NoC-enabled SoC runs a set of applications which generates a communication pattern for each IP block in the SoC. These communication patterns are commonly represented as a set of *task graphs* operating within the SoC [39]. When the applications are running correctly, the communication observed in the NoC conforms to the expected task graphs. When errors are present, unexpected or anomalous communication is observed. The accurate detection of these anomalies is the primary goal in the design of our test triggering mechanism.

We propose an **n-Anomaly Test Triggering Unit** (ATTU) which performs the following functions:

- Records application patterns between IP block and NoC during a non-faulty period of operation
- After pattern recording is complete, continuously observes the communication between the IP block and NoC
- Upon observing *n* anomalies in this communication, triggers a test request to the nearest TI-IP

By triggering tests based on the observation of NoC traffic, this work aims to reduce the response latency of COLT. This will increase the availability and reliability of the system.

1. ATTU Architecture

In this work, we have placed an ATTU within each CNI of the on-chip network. Fig. 7 illustrates the inclusion of an ATTU within a typical CNI architecture. Distributing these ATTU across the NoC eliminates the communication overhead associated with synchronization and management incurred by a centralized scheme, which in turn reduces latency and power consumption when determining when and where anomalies occur. Although the ATTU can be located within any component of the SoC, placing the ATTU within the CNI allows the ATTU to observe both *network layer* (source, destination, etc) and *transport layer* (address, command, etc) information contained within the messages. The CNI used in this research communicates with its IP

block via the OCP-IP 2.0 protocol standard, which includes address, data, and command fields [40]. Note that the inclusion of ATTU does not require the modification of any IP blocks.

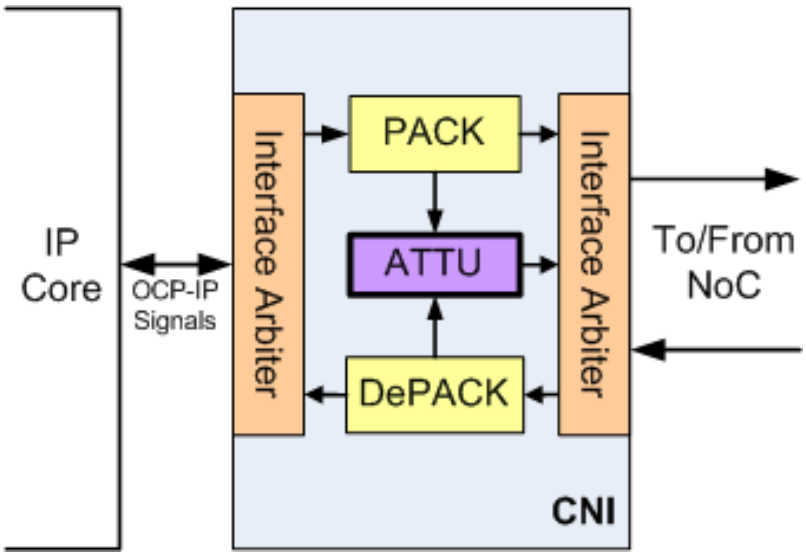

Fig. 7. CNI with ATTU Architecture

Anomaly detection can be implemented in several ways—the simplest implementations count events and establish threshold values for defining anomalous activity, while more sophisticated methods employ statistical techniques such as clustering or maximum likelihood functions [38]. Our implementation uses both simple counters and clustering when detecting anomalies in NoC communication.

2. Message Monitoring Considerations

As Fig. 8 illustrates, messages traveling across the NoC contain information stored in fields representing multiple layers of communication. The layers of interest to this research are the network and transport layers which contain the following fields:

source, destination, address, command, and data. Each of these fields can be treated as an independent dimension in terms of communication behavior. Each dimension reveals certain aspects of communication, and each dimension must be observed for anomalous behavior.

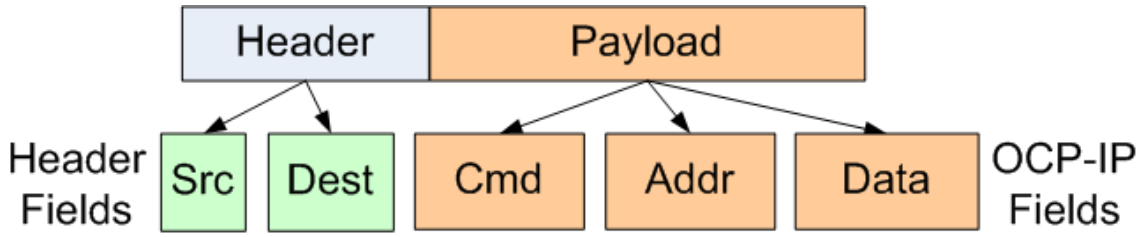

Fig. 8. Message Format

Whether to use a simple or complex detection method for a dimension depends on the cardinality, or size, of that dimension. For example, the source and destination fields have low cardinality and can only be of *n* values, where *n* is the number of IP blocks in the SoC. In SoCs which have dozens or hundreds of IP blocks, it is feasible to construct small counters for each possible value for source and destination. Alternatively, fields like data and address have high cardinality and can be one of billions of possible values – constructing counters for each possible address or data value is not possible. Therefore, statistical clustering is used for these dimensions [38].

Forming clusters in a single dimension is analogous to forming ranges, which may be implemented by storing the lower-most value and upper-most value of the ranges. To further simplify implementation, the proposed ATTU creates a constant number of clusters, regardless of application behavior. As the ATTU observes CNI communication, values are stored in empty ranges. Once all ranges are no longer empty,

nearest neighbor ranges are merged to allow for more observations to be recorded. Once these clusters are established, any observation which lies outside of these clusters is flagged as an anomaly. The initial version of the ATTU creates clusters for only the address field of the OCP-IP messages.

3. ATTU Training Period

As with most unsupervised learning techniques, the ATTU requires a "training" period in order to learn the normal communication behavior of a set of applications. The duration of this training period depends on the complexity of communication behavior exhibited by the set of applications run on the SoC. Assuming a static task set, the ATTU requires only one training period before it can effectively monitor the SoC for the remainder of its operational lifetime. Retraining is only necessary when the set of applications running on the SoC is changed. Fig. 9 illustrates a typical training phase period within the lifetime of a safety-critical system.

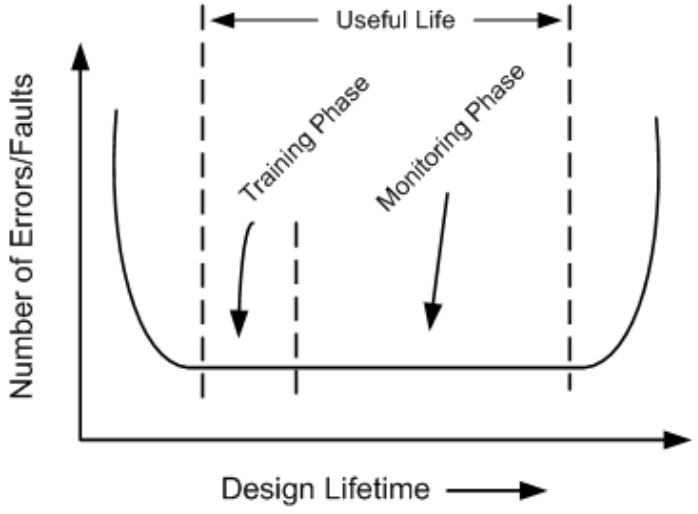

Fig. 9. Training Phase Period

It is important to note that the limitations of this proposed technique depends exclusively on the accuracy of the anomaly detector. If the ATTU triggers tests based on false positives, then the IP blocks are subject to excessive testing which increases power consumption and latency, and ultimately reduces the reliability of the system. Oppositely, if the ATTU cannot successfully detect errors, then faulty IP blocks are not sufficiently tested. Again, this leads to a system that is unreliable. Therefore, verifying that the ATTU accurately detects errors is the primary goal of our experiments.

D. Experimental Setup

1. Fault Model

As an IP block approaches the end of its operational lifetime, permanent failure is preceded by increasingly common incidences of transient and intermittent errors. The arrival rate and error pattern of these errors depend on the architecture of that particular IP block. When a fault occurs within an IP block, the architecture and application determine when that fault will **manifest** into an observable error in the system.

In the absence of this information, we explore the behavior of the ATTU by manifesting faults at the IP block-CNI interface with a variety of random distributions with respect to arrival behavior and manifestation behavior. To simulate the arrival of errors, *uniform*, *standard normal*, *Weibull*, and *Poisson* stochastic processes were used. The **fault effect**, or output deviation, was modeled with *uniform* and *normal* random distributions.

Faults within the NoC architecture – CNI, routers, and links – have been addressed by numerous researchers [4], [5], [35], [36] and are not the focus of this research. Therefore, we assume that only the IP blocks will become faulty over their operational lifetime.

Based on this fault model, we inject faults into the SoC via the IP blocks. Each injected fault is recorded to determine if the ATTU successfully identifies the fault as an anomaly. Additionally, we measure the latency between fault injection and anomaly detection within the ATTU.

2. NoC Configuration and Simulation

All NoC simulations are run on *NoCSim* [34], a SystemC-based cycle-accurate simulator [41]. NoC communication is modeled by processor-memory traffic and TI-IP tests. We implement the typical memory organization of NoC-based SoCs; processors with on-core L1 caches use the NoC to access L2 cache cores distributed throughout the network. NoC communication therefore primarily consists of memory communication resulting from L1 cache misses.

Fig. 10. NoC Topology

The SoC contains 25 tiles connected by a 5x5 2D torus (a 5-ary 2-cube topology). There are six ARM microprocessor IP blocks which transmit information across the network based on memory traces extracted from the SimIT-ARM instruction set simulator [42]. These microprocessors access data from 16 memory IP blocks. The three remaining IP blocks are TI-IP which await test requests from the ATTU and transmit test vectors upon test request arrival. Fig. 10 illustrates the NoC topology used for all experiments.

Each message is divided into three 64-bit flits and is routed across the network using a shortest path virtual wormhole switching algorithm. Each physical channel is divided into 8 virtual channels. Credit-based flow control is used for each router and CNI, and a buffer depth of 8 flights is used for each virtual channel.

3. Application Benchmarks

For each simulation, the six ARM processors execute a set of applications in order to produce a communication pattern in the NoC. Table 1 describes the application mapping for each test case used in our experiments. All applications are part of the SPEC CPU2006 suite of benchmarks [33]. For our experiments, we run the following application sets.

Table 1. SPEC CPU2006 Benchmark Test Cases

	mcf	**hmmer**	**Mix 1**	**Mix 2**	**Mix 3**
µP 0	mcf	hmmer	bzip2	mcf	hmmer
µP 1	mcf	hmmer	mcf	astar	astar
µP 2	mcf	hmmer	gcc	mcf	mcf
µP 3	mcf	hmmer	astar	mcf	gcc
µP 4	mcf	hmmer	hmmer	astar	gcc
µP 5	mcf	hmmer	hmmer	astar	astar

These applications represent a broad variety of benchmarks within the SPEC suite, and we have included three mixture cases to determine the effect of heterogeneous application load on ATTU behavior.

E. Experimental Results

1. Detection Rate

To maximally explore the behavior of the ATTU, we observe the accuracy and test-triggering latency of the ATTU under varying communication patterns, error manifestation distributions, and error effect distributions. Processor-memory communication patterns are affected by the application and cache configuration.

As described previously, the injected errors are affected by error manifestation and effect distributions. Our experiments determine the sensitivity of the ATTU with respect to each of these parameters. We also determine the effect of varying *n*, the number of anomalies to detect before triggering a test, on the test-triggering latency of the ATTU.

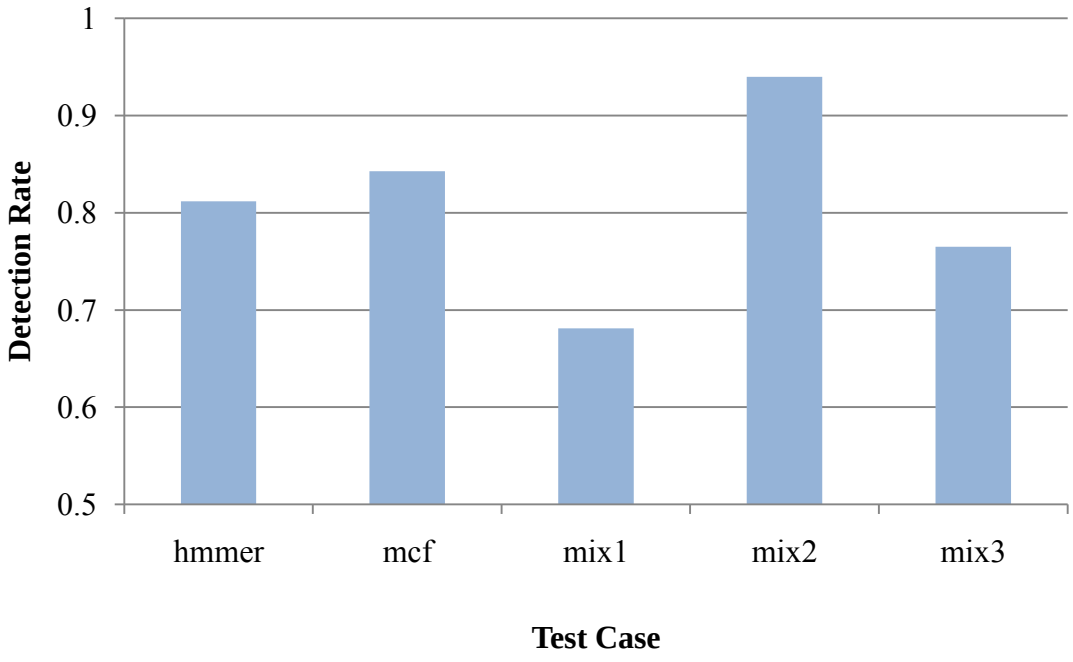

Fig. 11. ATTU Performance of Application Sets

Fig. 11 illustrates the average accuracy of the ATTU in terms of detection rate. Across the five selected application sets, the ATTU detects approximately 68% to 92% of errors manifested in IP block communication.

2. Effect of Error Distribution

Using the random distributions for fault manifestation and effects selected from the previous section, we observe ATTU accuracy averaged across the five test cases. Table 2 shows the effect of error behavior on ATTU performance.

Table 2. Effect of Error Distribution on ATTU Performance

	Fault Effect Distribution		
Manifestation Distribution		Uniform	Gaussian
	Uniform	84.2%	76.6%
	Normal	84.1%	79.7%
	Weibull	86.2%	78.5%
	Poisson	85.6%	77.5%

Here, we see little effect on ATTU behavior due to error distribution; this is primarily due to the nature of the application communication between processing and memory cores.

3. Effect of Error Rate

As the effective error rate is increased from 5E-06 to 3E-05, the ATTU detection rate is largely unaffected, varying from approximately 80% to 88% accuracy. However, test-triggering latency reduces as error rate increases. This is an expected result, as more errors manifesting in the communication between IP blocks results in more anomalies detected.

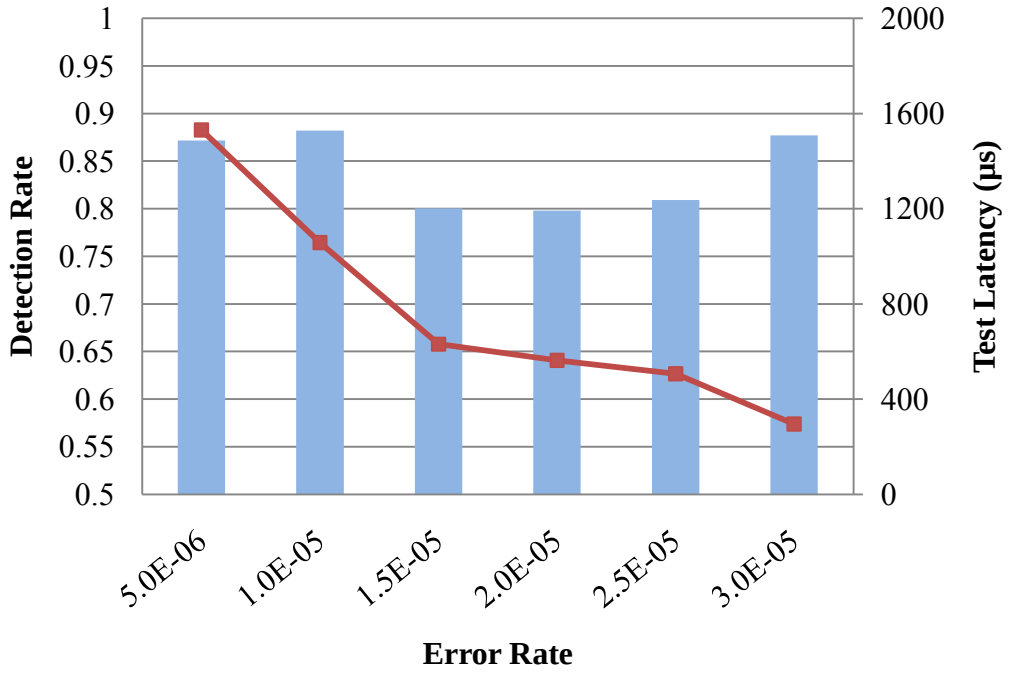

Fig. 12. Effect of Error Rate on ATTU

Fig. 12 shows that test initiation latency reduces by 33% as the error rate increases from 5.0E-06 to 3.0E-05. Test initiation latency can be further adjusted by the number of anomalous required to trigger tests as discussed later in this section.

4. Effect of L1 Cache Size

Varying the L1 cache size of each processor IP block changes the memory communication observed on the network – larger caches require less L2 cache accesses as capacity and conflict misses are reduced. This reduction in memory traffic as L1 cache sizes increase results in longer test-triggering delays as seen in Figure 8.

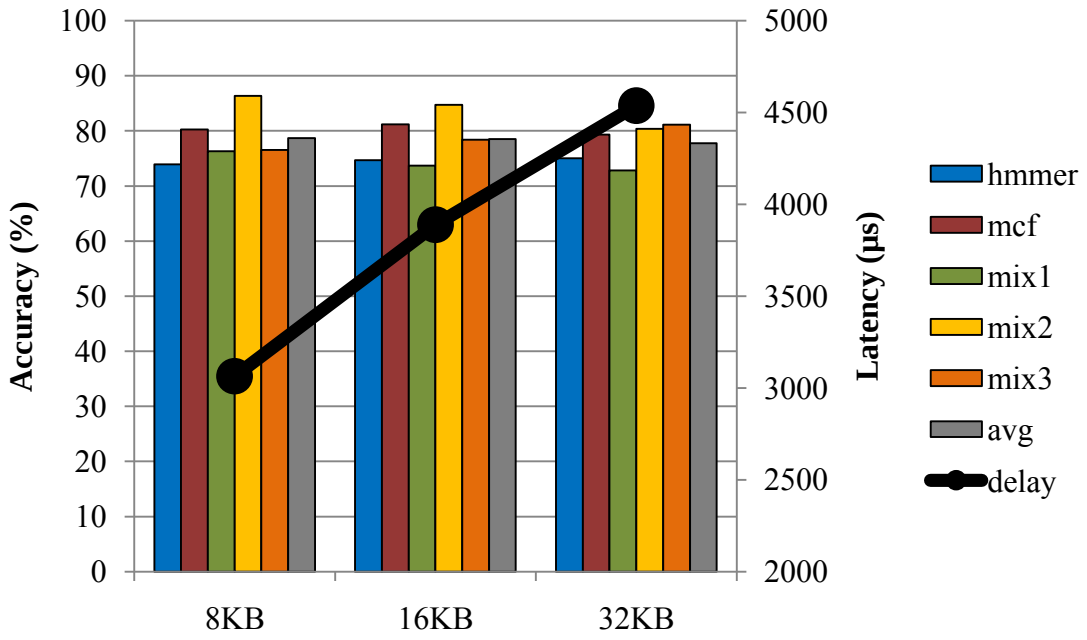

Fig. 13. Effect of L1 Cache Capacity on ATTU

Fig. 13 shows that the benchmark set used has little effect on accuracy; however, test initiation latency increases from approximately 40µs to over 80µs as the L1 cache size increases from 8KB to 32KB.

5. Effect of ATTU Memory

Fig. 14 shows the effect of ATTU memory on detection accuracy and hardware overhead. As the amount of memory dedicated to recording NoC traffic grows, the ATTU is able to detect finer-grained anomalies within traffic. This translates into a 14% increase in detection accuracy as the number of ternary content addressable memory (TCAM) rows increases from 5 to 20.

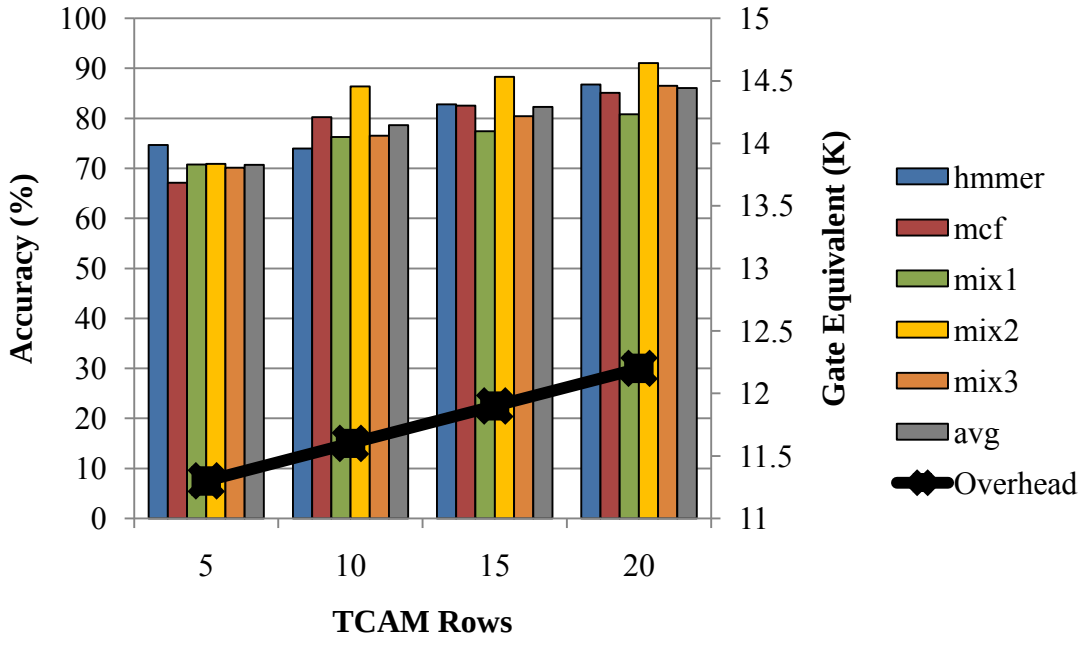

Fig. 14. Effect of Memory Size on ATTU

Additionally, it is observed that the overhead due to increased memory usage by the ATTU increases from 11.25K gate equivalents to 12.25K gate equivalents as the number of TCAM rows is varied from 5 to 20.

6. Effect of Trigger Sensitivity

Increasing the number of anomalies an ATTU observes before triggering a test from the TI-IP increases the test-triggering latency, shown in Figure 15. A designer including the ATTU should choose *n* such that the expected test-triggering latency meets the reliability and availability requirements of the system.

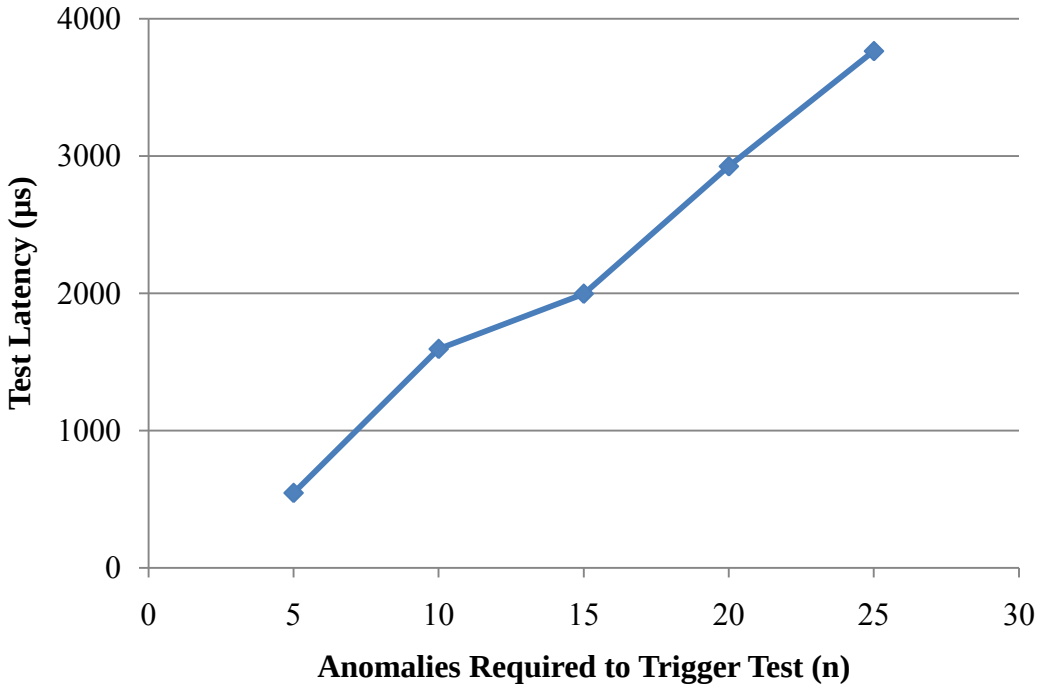

Fig. 15. ATTU Behavior for Various Values of n

Here, it is observed that the test initiation latency increases linearly with *n* from approximately 500µs to nearly 4000µs.

7. ATTU Overhead

Overhead costs in terms of area and power were measured by synthesizing the ATTU using Synopsys Design Compiler [43] and the Virginia Tech VLSI for Telecommunications TSMC-0.25µm, 2.5V standard cell library [44], [45]. Realizing an ATTU configured for this experimental setup would require an estimated 11.6K gates while consuming an estimated 50.8mW of power (leakage power of 4.64µW).

F. Conclusion

This work proposed an on-demand test triggering mechanism, the ATTU, for concurrent on-line test of SoCs. On-demand COLT of NoC-based safety-critical systems offers several advantages over traditional periodic COLT. Testing of system components, an expensive and time consuming task, is reserved to times when indications of error within the system exist. Our exploration of the ATTU under varying fault manifestation and effect distributions shows that including this mechanism promotes electronic design quality throughout its lifetime reduces fault detection time, thereby maximizing system availability. Future work includes enhancing ATTU and TI-IP coordination to autonomously diagnose system errors to assist in self-healing capabilities.

CHAPTER IV

DISTRIBUTED TEST VECTOR STORAGE

A. Introduction

Multi-core systems-on-chip (SoC), with a handful of complex processing cores and integrated peripheral components, are expected to be replaced by many core SoC that contain hundreds or thousands of light-weight processing cores, memory and I/O subsystems within the next decade. This transition marks a fundamental shift in how reliability, availability and testability for these systems will be achieved due to their massive redundancy and physical characteristics.

These many core SoC will use packet switched networks-on-chip (NoC) for inter-core communication, as opposed to the current standard of on-chip busses [1], [10]. Notable examples of this architecture include the 100-core TILE-Gx100 from TILERA [28], the 80-core Intel Terascale SoC [46], and the 48-core Intel SCC [47] organized as 24-tiles connected via a NoC mesh. On a smaller scale, Freescale Semiconductor has recently introduced an 8-core device for embedded systems. This device, the QorIQ P4080, uses a propriety switch fabric called CoreNet, which is a NoC providing cache coherence and quality-of-service capabilities [48], [49].

As technology scaling has provided new opportunities for massively parallel and distributed computation to be performed on a single chip, new reliability challenges have also emerged. In addition to the well-understood circuit failures due to manufacturing imperfections, SoC components are also more susceptible to *early-life failures* (ELF)

and *electronic wear-out*—permanent failures that emerge during use – as feature sizes scale below 65nm [1], [13], [50].

Electronic wear-out is a combination of several physical degradation mechanisms, including electro-migration (EM), hot carrier injection (HCI) and negative bias temperature instability (NBTI), that are intensified by smaller feature sizes, higher current and power densities, and higher operating temperatures [13]. Because the most significant electronic wear-out mechanisms manifest as an increasingly severe delay fault at the circuit level, many researchers have proposed the use of SCAN-based delay testing for detecting this type of error [4], [16], [51], [52]. Built-in self-test (BIST) architectures using pseudo-randomly generated test vectors are effective in discovering wear-out in memory elements. Unfortunately, BIST techniques cannot be applied to control portions of logic components, such as intellectual property (IP) cores, due to the low fault coverage levels provided by logic BIST [16].

Therefore, researchers have noted the need to apply production-quality, high-coverage SCAN delay test vectors to the cores within SoC once the system has been deployed in the field [4], [16]. These tests can be applied as an isolated process – the entire system is taken off-line periodically or triggered by an event so that all cores can be tested for wear-out [16]. Alternatively, core testing can be performed *concurrently* with normally executing applications [4], [5], [16], [52].

Concurrent on-line test (COLT) exploits the massive structural redundancy of multi- and many-core architectures by shutting down some subset of cores within the SoC for testing while the remaining cores run user applications as normal. This allows

the system to achieve its reliability requirements and maintain an extremely high level of availability. The amount of overhead in terms of power consumption, resource and network congestion, and chip area costs incurred by testing system components during normal operation is known as *application intrusion*. Application intrusion must be minimized in order for COLT strategies to operate feasibly.

In the COLT strategies currently proposed in research, the delivery of test vectors from the test source (on-chip test storage or off-chip memory) to the individual cores within the SoC is the most significant contributor to testing costs in terms of test latency and energy consumption [4], [16]. This problem will only become more critical as the number of cores per SoC increases over time. As the distance between test source and sink increases for deeply embedded cores, application intrusion in the form of test latency, energy and NoC congestion also increases.

In this research, we present a distributed test vector storage (DTVS) technique for safety-critical SoC that can accommodate storage, power, latency, and availability constraints in an optimal fashion. The on-chip networks we study in this work focus on the 2D-torus topology—a popular topology in research and in practice due to its efficient hardware implementation, small diameter, and simple routing, but this technique can be used in any regular topology.

The main contributions of this research are the following:

- Proposes the use of a formalized, distributed storage technique (t-interleaving on tori) to bound the distance test vectors travel within the network, reducing online test intrusion

- Demonstrates that this scheme can reduce test delivery latency and energy consumption by approximately 75% and total network load by 76% for the moderately sized NoC simulated in our experiments
- Proposes a fully distributed COLT scheduling protocol that addresses the scalable test vector delivery issues present in current COLT research.
- Validates the proposed protocol and supporting architecture through three case study cores: the OpenRISC processor, an ARM7 processor, and an open-source NoC router as the cores to be tested across the SoC
- Presents an optimal core test ordering, called Code-Division Core Test Scheduling, when using distributed test vector storage. By simply ordering which subset of cores is tested simultaneously, system test delivery latency and energy consumption is reduced by 40% when compared to other core test orderings.

The remainder of Chapter IV is organized as follows. Section B describes related work in online testing techniques, specifically focusing on efforts in concurrent online testing. Section C briefly motivates the use of distributed test vector storage for online testing based on observations of previous work. Section D describes our proposed use of t-interleaving on tori to optimally distribute test vectors across a many-core SoC such that test vector delivery distances are bounded. Section E describes the proposed distributed concurrent online testing protocol. Section F presents our analysis of distributed test vector storage in terms of network traffic load, test delivery latency, required energy consumption, and on-chip storage requirements for various SoC configurations. Section G describes the experimental setup used to measure the

performance of delivering test vectors using the standard single-source approach and the described distributed storage technique. Section H presents the results of our experiments in terms of test delivery latency, energy consumption, and overheads. Finally, Section I summaries this work and describes future directions for research regarding this technique.

B. Mechanisms of Concurrent Online Test

Concurrent online testing is a method of testing SoC components in the field such that user-level applications are unaware of this activity. Researchers have proposed various mechanisms to implement COLT in SoC. In this section, the three most popular approaches are described. In order to provide the high-coverage test delivery, test application, response retrieval and comparison traditionally provided by external automated test equipment (ATE), on-chip hardware, software and co-designed approaches have been evaluated.

1. On-Chip Test Controllers

In [32], the researchers analyzed the costs and benefits of reusing the NoC as a test access mechanism (TAM) for each core in the network. This was proposed as a response to the increasing difficulty of accessing deeply embedded cores within a SoC. It was shown that a NoC is indeed a feasible TAM due to minimal overheads; however, it was noted that test delivery times (and power consumption) depend on the distance between the test source and the core to be tested. Additionally, this variability in test

delivery time may not be fully understood, since relatively small 5x7 and 4x8 2D-torus topologies were studied in that research.

Using the NoC as a test delivery infrastructure, researchers have proposed reusing infrastructure IP (I-IP) designed originally for manufacturing testing as a tool for on-line testing of the system [20].

Bhojwani [4], [5] constructed a Test Infrastructure IP (TI-IP) capable of managing test scheduling, delivery, and intrusion for concurrent on-line test (COLT) of the SoC. COLT allows cores in the SoC to be tested in the presence of normally executing applications to maximize system availability.

The original COLT scheme proposes that the test vectors are stored within the TI-IP. Due to the real-time constraints of these applications, COLT is extremely sensitive to application intrusion.

Microprocessor pipeline on-line testing using distributed on-line BIST and periodic check-pointing was investigated and shown to be an effective technique in providing high reliability and availability at a reasonable area cost (5.8%) [22]. In this scheme, stuck-at fault test vectors are generated for processor components and stored in on-chip ROM. It should be noted that distributed BIST discussed in [22] is a different concept than DTVS discussed here. In [22], distributed BIST refers to separate BIST mechanisms for the different pipeline components of a single microprocessor core, where distributed test vector storage refers to the separate BIST vector storage units across the entire SoC.

Yi, Makar and Mitra have proposed CASP: Concurrent Autonomous chip Self-test using stored test Patterns [16]. Similar to COLT, an on-chip test controller is proposed which manages test scheduling, test application and response comparison for processing cores of the OpenSPARC T1 chip multi-processor [23]. Their technique differs from COLT in that the test vectors are stored off-chip in a nearby flash or hard disk drive (HDD) storage system.

Consistent with previous research, the time required to test a core was almost completely dominated by test vector delivery latency. When using off-chip flash memory, 83% of the total test time was consumed by transferring test vectors to the core under test. Delivering test vectors via off-chip HDD memory accounted for over 99% of the total test time [16]. However, it was successfully demonstrated that CASP is a feasible solution to ensure lifetime reliability for certain applications.

Operating system and hardware virtualization support is required for any COLT scheme to operate invisibly to the system user. The OS/virtualization layer must track which cores are available to the operating system for normal applications and to initiate task migration when necessary during online test. This topic is addressed in [17], [52] and is outside the scope of this work.

2. ISA Testing Extensions

In [51], a hardware/software co-designed approach is presented in order to provide COLT capabilities to a SoC. Instead of a dedicated on-chip test controller determining when and how IP cores are tested, the authors extend the ISA of the processor to give software access to the core's SCAN chains. These additional

instructions, called Access-Control Extension (ACE) instructions, allow software to control the online testing of cores as a separate process. Software-controlled testing benefits from increased flexibility; modifying the testing strategy is as simple as changing the controller software.

Although software manages the test infrastructure, hardware modification is required to implement ACE instructions. SCAN chains are expected to already be present due to design-for-test (DFT) requirements; however, the control structure of the core must be modified to accept these additional instructions, and extra datapaths are required to connect the SCAN chains to the instruction pipeline.

3. Software-Based Self Test

Software-based self test (SBST) performs all testing completely in software. Both the test controller and actual test application are implemented by standard processor instructions. Therefore, SBST can be scheduled like any other software task, allowing for the operating system to be aware of testing, and giving the system designer great flexibility in scheduling online tests.

SBST is completely non-intrusive in terms of hardware overhead; however, verifying that high-coverage levels of structural faults can be achieved strictly through the application of processor instructions is challenging. Several SBST techniques have been shown to achieve stuck-at fault coverage rates of 80% and above [24].

For delay fault coverage, a technique has been proposed and evaluated on a simple, custom processor [25]. SBST techniques have also been proposed for floating

point units, where extremely high stuck-at fault coverage and low memory footprint have been achieved [26], [27].

C. Motivation for Distributed Test Vector Storage

Regardless of the mechanisms used to provide COLT, test data storage, delivery and retrieval is a major consideration in system design. Each of these proposed COLT designs require some central test data repository. As integration increases, the distances between test sources and sinks also increase.

Additionally, it is expected that online testing must occur very frequently for each core to predict or detect early-life failures – the frequency of core testing is on the order of seconds [17]. This high frequency testing requires that online testing consume as little time as possible.

1. Test Delivery Costs Relative to Distance

As a demonstration of the growing contribution of test delivery to total test time as integration increases, we evaluate the ratio of test delivery time to test application time of the OpenRISC CPU core [53] as the delivery distance increases.

The four cases illustrated in Fig. 16 correspond to test vectors travelling 4 hops, 8 hops, 10 hops, and 12 hops in a 2D-torus based SoC. The test application time for transition fault testing of the OpenRISC CPU is 1.186ms; however, as the number of hops increases to deliver these test vectors, the test delivery time increases from 1ms to 2.3ms. At 4 hops, the test delivery time (47%) is roughly equal to test application time (53%). At 12 hops, 66% of the time required to test the core is dedicated solely to

delivering the test vectors to the core under test. This demonstrates the increasing importance of test delivery costs of COLT as the number of cores per chip scales within an SoC.

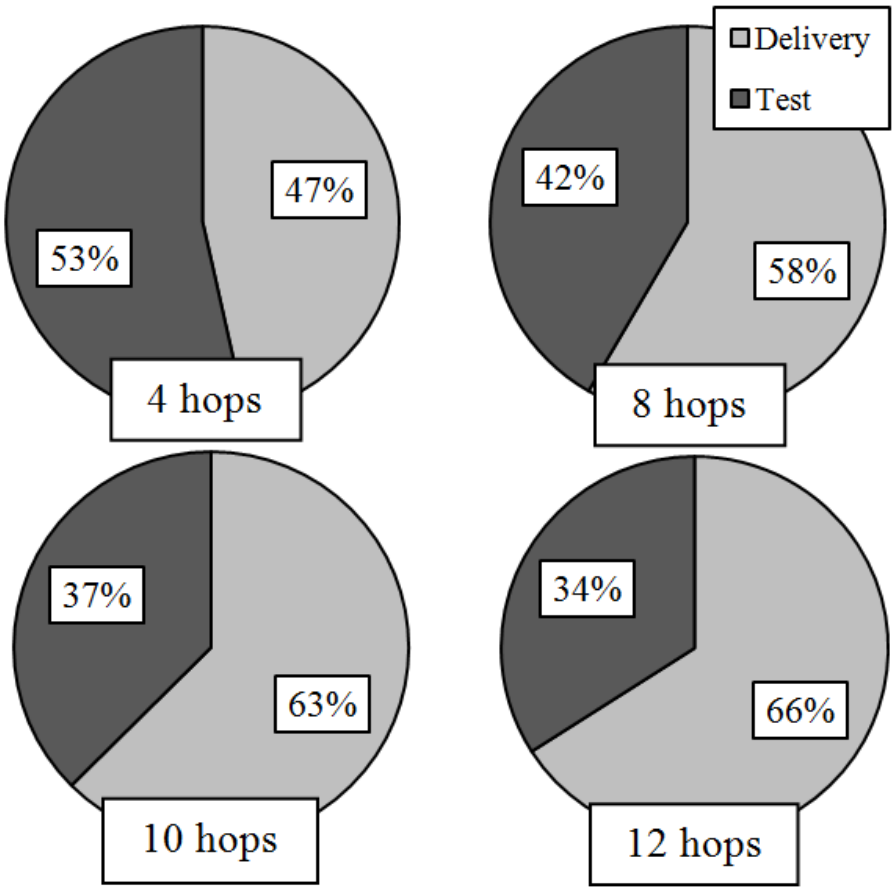

Fig 16. Growth of Test Delivery Time

Based on these observations, a distributed approach to test data storage may provide a better solution to the performance-storage tradeoff of test data storage for online testing. Instead of a single source of test data sending information to each core across the entire chip, storage redundancy can be used to shorten the effective distance between test sources and sinks. This research presents a formal approach to assessing

this tradeoff – taking advantage of the regular structure of many-core SoCs, this storage redundancy can be optimally placed to maximize the benefits of distributed test vector storage.

Aside from these scalability issues, longer test delivery and response distances increase the complexity (and potentially decreases the accuracy) of calculating intrusion costs of on-line testing by the test controller, such as the previously described TI-IP. Ultimately, this may negatively affect the availability of the system. This motivates the need to bound test delivery distances for all cores in the SoC.

D. Distributed Test Vector Storage Analysis

1. Interleaving on Tori

This research focuses on NoC based on the 2D-torus topology, also known as a k-ary 2-cube (Q_2^k). As the Hamming metric has been shown to efficiently describe relationships between nodes in hypercubes, the Lee metric has been shown to be a natural description of distances between nodes in tori [54], [55], [56].

The Lee metric for a 2D-torus of size n x n can be described by (1) and (2) as follows:

$$w_L(A) = \min(a_0, n - a_0) + \min(a_1, n - a_1) \quad (1)$$

$$d_L(A, B) = d_L(A - B) \quad (2)$$

where w_L is the Lee weight of a node A within the 2D-torus, and a_0 and a_1 are the node's positions in the two dimensions. d_L is the Lee distance between two nodes in a torus, and

like the Hamming distance, it is simply the weight of the difference between the two nodes. The Lee distance between two nodes can be intuitively described as the number of hops between two nodes in a torus topology. For a more in-depth discussion of the torus topology and their properties, please refer to [54], [55].

Understanding the role of Lee metrics in tori is important when considering how to optimally distribute a file across a 2D-torus topology. Any test vector distribution scheme that bounds test vector delivery to each core must ensure that the complete set of test vectors are within a certain Lee distance to each core. In other words, the delivery of test vectors must be bounded within a certain number of hops within the torus topology. We will now discuss how to distribute test vectors across a torus in more detail.

A file F (in our discussion, F is a set of test vectors) can be evenly divided into a set of file segments: $F = (F_0,F_1,\ldots,F_m)$. Each core can reconstruct the file F by retrieving the file segments from each of its neighbors within a radius r. It is necessary that for each core in the 2D-torus, all cores within that radius store a distinct file segment in order for the complete file F to be reconstructed.

2. Interleaving Example

As an example, Fig. 17 depicts an arbitrary core in a large 2D-torus which is storing file segment F_6 of 13 total file segments. The neighboring cores within the radius $r = 2$ store the remaining 12 file segments. When all 13 nodes are combined, a Lee sphere is created, and each node within that Lee sphere contains a distinct file segment. It can be clearly seen that the core at F_6 can access the entire file F by

retrieving data from its neighbors within the radius *r*. Likewise, it can be imagined that these 13 segments are replicated across the entire torus in a symmetric fashion; therefore, any core within the torus can access all of file F within the radius *r*.

In order to guarantee every node in the 2D-torus can also access the entire file within the same radius, the Lee sphere centered at each node must contain nodes with distinct file segments. The file F must therefore be replicated across the entire torus in a spherical manner such that this constraint is satisfied. Replicating the file in this manner is a straightforward process and can be done during design time as a one-time cost. Once the placement of file segments is made, the process does not need to be repeated.

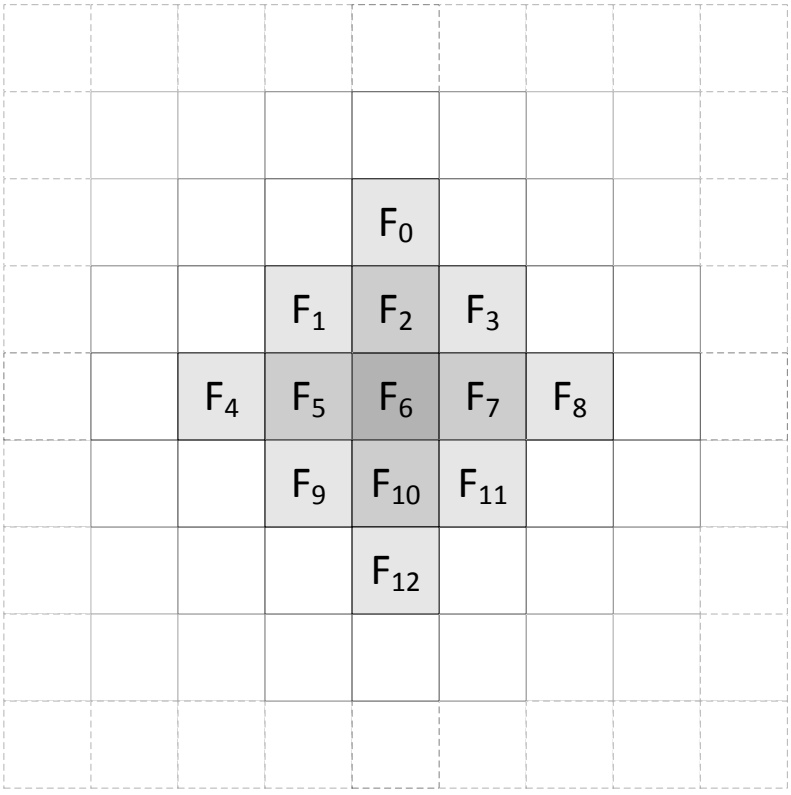

Fig. 17. A Core at F6 Retrieving All File Segments Within Radius 2

An alternative way of viewing this problem is to realize that replications of the same file segment F_i can be no closer than 2r + 1 hops apart. Viewing the problem in this way relates file placement to coding theory, which ensures that codewords are a minimum distance apart. In this case, that minimum distance is 2r+1, and the codewords can be viewed as tiles within the torus.

This problem has been solved through *t-interleaving on tori*. t-interleaving on tori is formally defined by Dr. Jiang as:

> Let G be a graph. By an interleaving, we will mean a vertex coloring, as follows. We say that G is interleaved (or there is an interleaving on G) if each vertex of G is assigned one of a finite number of distinct colors. We say that G is t-interleaved (or there is a t-interleaving on G) if every set of t vertices, forming a connected subgraph of G, is colored by t distinct colors. [56]

For 2D-tori, the minimum number of file segments required to achieve a t-interleaving on a 2D-torus is $|S_t|$, the number of nodes within an interleaving sphere, described in (3).

$$|S_t| = \begin{cases} \frac{t^2+1}{2} & , t\ odd \\ \frac{t^2}{2} & , t\ even \end{cases} \quad [56] \quad (3)$$

In [56], $|S_t|$ has been proven to be the lower bound on how many file segments must be used to create a t-interleaving on a torus, and $|S_t|$ must divide *n* for a *n* x *n* 2D-

torus to be a perfect, or optimal, t-interleaving. This work studies perfect t-interleavings over tori; however, non-perfect t-interleavings can still achieve tight bounds on test delivery distances.

Looking back at Fig. 2, we see that the core storing file segment F_6 is at the center of a Lee sphere of radius r = 2, created by a 5-interleaving on that torus ($|S_5|$ = 13).

In [56], $|S_t|$, the size of a Lee sphere with diameter *t*, has been proven to be the lower bound on how many file segments must be used to create a t-interleaving on a torus, and $|S_t|$ must divide *n* for a *n* x *n* 2D-torus to be a perfect, or optimal, t-interleaving. This work studies perfect t-interleavings over tori; however, non-perfect t-interleavings can still achieve tight bounds on test delivery distances.

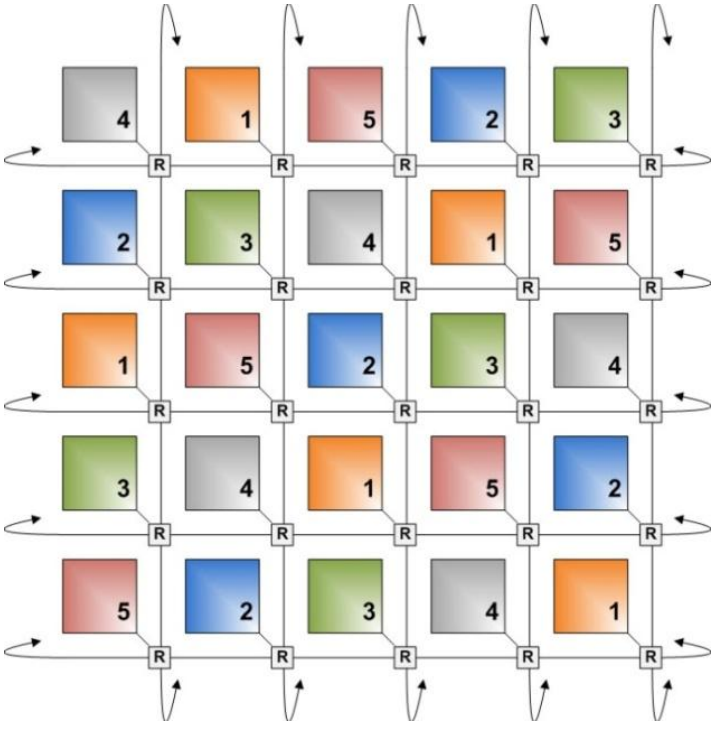

Fig. 18. 3-interleaving on 5x5 2D-torus

Fig. 18 depicts another example of t-interleaving on a smaller torus, where each numbered tile is a core, connected by routers *R*. Specifically, it depicts a perfect 3-interleaving on a 5x5 2D-torus. By observation, this example shows that any node within the graph can reach all colors within 1 hop. It also shows that nodes of the same color (or number label) are no closer than 3 hops apart. Another way to view this interleaving is to see that every node is the center of a Lee sphere of radius 1, and each Lee sphere contains 5 nodes with distinct colors (or numbers).

3. Applying Interleaving to Test Storage

By using t-interleaving on tori for on-chip test vector storage, it is proven that any core in the network can access the complete set of test vectors within a defined radius. Bounding this test delivery radius for each core not only guarantees scalability, but on-line test intrusion can be much more easily estimated. The level of intrusion for each core no longer depends on proximity to the test source, simplifying scheduling decisions for on-line testing.

This same technique can be applied to any topology with regular structure including meshes and rings. Other proposed NoC topologies, such as the dense Gaussian network [57], are also suitable candidates for using this technique.

E. Distributed COLT Architecture

1. System Components

Because the proposed test scheduling protocol is fully distributed, inserting dedicated TI-IP tiles into the SoC is unnecessary. Instead, each tile contains a small test controller within the Core-Network Interface (CNI) of each tile.

Fig. 19 illustrates the distributed COLT architecture for each NoC tile within the SoC. NoC tiles communicate with each other via on-chip routers (R) and CNI. In the example system, each NoC tile contains a simple CPU core, such as the OpenSPARC T1 core, surrounded by DFT SCAN chains.

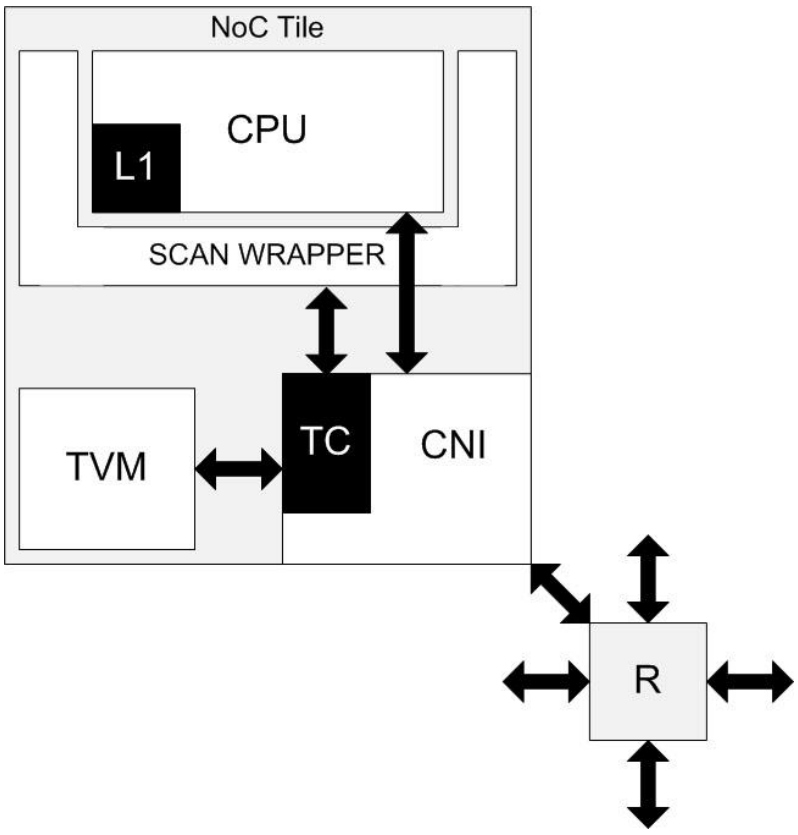

Fig. 19. Distributed COLT Architecture Within a Tile

Each CNI contains a Test Controller (TC) which implements the proposed distributed COLT protocol. The TC contains a nominal amount of buffers to temporarily store incoming test vectors and responses from neighboring cores and communicates directly with the DFT structures surrounding the core under test. Each tile also contains a dedicated Test Vector Memory (TVM) which stores some portion of the complete test vector set. The TC is responsible for sending the local test vector subset to neighboring cores upon request.

The TC is not directly responsible for managing power consumption of the core or the system during test mode. Researchers have proposed that NoC-based SoC can manage power consumption by implementing a power management (PM) unit within each CNI or router [4], [58], [59]. This PM can manage power consumption of its associated NoC tile by throttling network traffic and the TC.

Because multiple NoC tiles may initiate tests simultaneously, a token-based protocol is used to manage the distributed TC. This protocol is detailed in the next section.

2. Distributed COLT Scheduling Protocol

This research proposes a simple distributed COLT scheduling protocol that allows each tile to manage its own self-test. The TC of each CNI implements the test scheduling protocol as illustrated as a state machine in Fig. 20. We propose a token-based protocol where the individual SoC tiles are organized into one or more token rings. Any tile possessing a token can initiate self-test and respond to test results as

necessary. The rest of this section details token generation, test application and response, and token passing.

Token Generation: Depending on the operating and SCAN test characteristics of the CPU cores within the SoC, multiple cores may be tested simultaneously without violating the system power budget and application requirements. The number of cores that may be tested simultaneously is determined by the system architects and is system and application dependent. If *n* cores can be tested simultaneously, then the system is initiated with *n* tokens distributed across the NoC tiles. The cores of the SoC are divided into *n* token rings such that each ring contains one token and each core is a member of one token ring. It is possible that rings may contain an unequal number of cores depending on system characteristics. Cores without tokens remain in the *WAIT FOR TOKEN* state before initiating COLT locally.

Core Test: Once a tile possesses a token, the TC enters the *INITIATE TEST* state and may begin a local core test. Before actual testing can begin, it may be necessary to unload the core of its current task and migrate this task to an alternate core. The TC will request the other pieces of the test vector set from neighboring tiles and will begin applying SCAN tests to the test wrapper during the *TEST IN PROGRESS* state. All test vectors are applied and responses are compared until test completion. Once the TC is in the *TEST COMPLETE* state, it will determine whether the testing has passed or failed based on a standard response comparison.

Test Response: If the test has passed, the system can be configured so that the core may resume its task as described in [52]. If the testing has failed, the TC enters the

Fault-Tolerance Response (*FT RESPONSE*) state and the appropriate action is taken. For instance, an appropriate response would be to disable the faulty core and replace it with a cold spare if one is available. Alternatively, a faulty core can be disabled and system operation resumes in a degraded state.

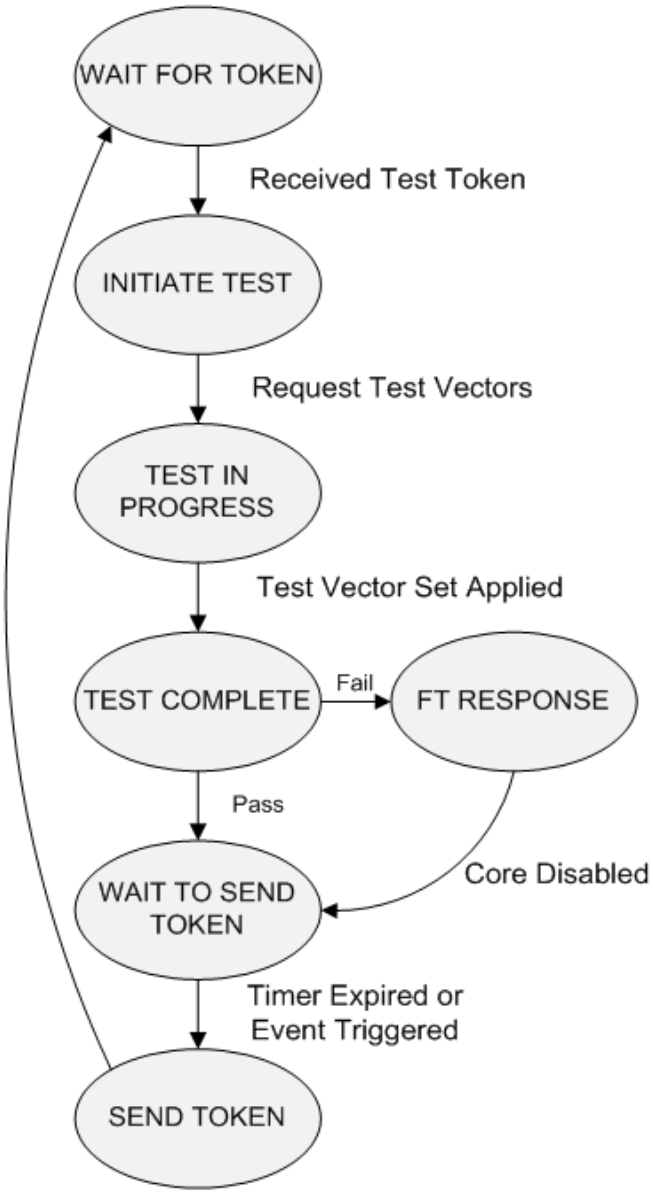

Fig. 20. Distributed COLT Protocol

Token Passing: Once this action is complete, or if the test passes, the TC enters the *WAIT TO SEND TOKEN* state. Token passing may be initiated periodically or due to specific system events. In [17], researchers have found that detecting hard failure requires very frequent testing. They note that test periods must be on the order of seconds (1-10 seconds) in order for checkpointing schemes [22] to be effective at rolling the core back to a correct state. Testing cores this frequently requires that system designers consider the impact of COLT application intrusion, including test delivery costs when applying stored test vectors.

3. Code-Division Core Test Scheduling

Given that it will be possible for some many-core SoC to test multiple cores simultaneously, optimizing which cores can be tested simultaneously for latency, network congestion, power consumption and thermal effects is a critical consideration for COLT.

As described in Section D, a many-core SoC which uses a NoC with a regular topology such as the 2D-torus, can be represented as a set of Lee-metric error correcting codes. Each code contains a set of NoC tiles located by each codeword. For example, Fig. 18 shows a 25-core SoC that contains 5 different error correcting codes. Each code is labeled by a distinct number – the tiles that are labeled 1 all belong to the same error correcting code. By the definition of error correcting codes, each tile belonging to the same code must be *t* hops apart.

Because each tile belonging to the same code—tiles that will be tested simultaneously—are t hops apart, and each core under test must access test vectors from

at most t – 1 hops away, there exist no resource conflicts due to test. In other words, any tile in the SoC will send its segment of the test vector to only one requesting core under test at any time. This design constraint greatly simplifies the testing architecture; no dual ported TVM are required, and there is no need to broadcast or multicast test vector segments across the network. Therefore, Code Division Core Test Scheduling dictates that only cores belonging to the same error correcting code may be tested simultaneously.

However, there are fundamental limits to this test ordering scheme. No more than *n* cores in an *n* x *n* 2D-torus SoC can be tested simultaneously, otherwise resource conflicts will occur. Also, if only one core can be tested at a time, the order in which cores are tested will have no effect on testing performance, neglecting user application effects.

F. Analytical Results

1. Network Load Analysis

For evaluating total network load of test vector delivery on a 2D-torus NoC-based SoC, we have developed the following equations.

The first $V_{traffic}$ equation, (4), describes the total network load for a centralized, single source test vector storage scheme, where B_{test} is the size of the test vector set in bytes, TIIP is the location of the test vector source, and $Core_{ij}$ is each core in the SoC. In centralized COLT, $V_{traffic}$ is determined by the Lee distance d_L, defined in (2), between the TIIP and each core to be tested.

$$V_{traffic} = B_{test} \sum_{i=0}^{n} \sum_{j=0}^{n} d_L(TIIP, Core_{ij}) \tag{4}$$

The second $V_{traffic}$ equation, (5), describes the total network load for a t-interleaved distributed test vector storage scheme for a 2D-torus of size n x n. Again, B_{test} is the size of the test vector set in bytes, and $|S_t|$ is the number of segments the test vector set is split into. h signifies the number of hops from each test source node to the test requesting node.

$$V_{traffic} = \begin{cases} \frac{B_{test}}{|S_t|} n^2 \sum_{h=0}^{\left|\frac{t}{2}\right|} 4h^2 & , t\ odd \\ \frac{B_{test}}{|S_t|} n^2 \left(\sum_{h=0}^{\frac{t}{2}-1} 4h^2 + \frac{t(t-1)}{2} \right) & , t\ even \end{cases} \tag{5}$$

Using these equations, Fig. 21 shows the reduction in total network load when using t-interleaving compared to single source test storage. For the 5x5 and 10x10 tori, 3-interleaving was used, whereas 4-interleaving and 5-interleaving was used for the 8x8 and 13x13 tori, respectively. The greatest network load reduction of 76% can be observed for the 13x13 torus. Greater network load reductions can be achieved for larger tori due to the ever increasing test vector delivery distances of the centralized scheme.

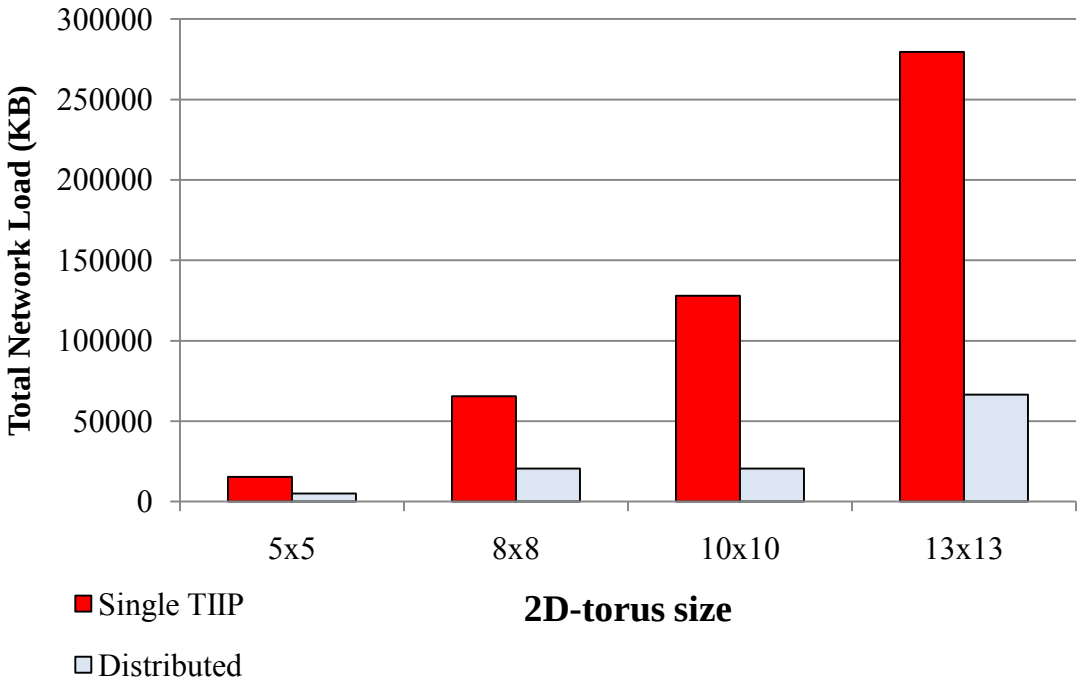

Fig. 21. Network Load Comparison as Network Size Increases

The results shown in Fig. 21 are a promising trend for using COLT with DTVS—the benefits increase with SoC size.

2. Energy Consumption Analysis

Numerous studies have shown that energy consumption of a NoC is directly proportional to the amount of traffic (flits) being transmitted in that network [58], [59]. To calculate energy consumption relating to delivering test vectors across the NoC, we use a standard flit-based energy calculation adapted from [58], with 90nm NoC component energy parameters from [60]. Namely, each flit is 256 bits long, and the network operates at a frequency of 1GHz.

Equation (6) states the amount of energy consumed per flit transmitted.

$$E_{flit} = (h + 1)(E_i + E_o + E_{sw}) + h * E_{link} \quad [58] \quad (6)$$

where h is the number of hops the flit travels, E_i is the energy consumed in the tile in-port, E_o is the energy consumed in the tile out-port, E_{sw} is switching energy, and E_{link} is the energy consumed by the network links.

Table 3. Energy Consumption Over Various Tori (µJ)

	Single Source	**t-interleaved**	**Energy Reduction**
5x5	54.31	18.10	67%
8x8	231.74	72.42	69%
10x10	452.61	72.42	84%
13x13	988.50	235.36	76%

Table 3 describes the energy savings due to the limited network traffic resulting from t-interleaving. From this, we observe an average energy reduction of 74% when distributed storage is used for these four cases.

It is also observed from Table 3 that using t-interleaving to store tests for the 8x8 and 10x10 tori results in equal energy consumption for test delivery, despite their unequal network sizes. This is due to the fact that 3-interleaving is used for the 10x10 torus (Lee spheres of size 5 are created), and 4-interleaving is used for the 8x8 torus (spheres of size 8 are created). In other words, smaller but more numerous spheres are created for the 10x10 torus, reducing the distances that test vectors travel across the chip

when compared to the 8x8 torus configuration. The ultimate effect of these disparate sphere sizes is that the smaller spheres created for the 10x10 torus negate the higher energy consumption requirements induced by the larger 10x10 torus topology.

3. Storage Redundancy

For an *n* x *n* 2D-torus, the amount of redundancy *r* introduced into the network through perfect t-interleaving is described by the following equation:

$$r = \frac{n^2}{|S_t|} \tag{7}$$

This equation simply computes the number of Lee spheres embedded within the torus.

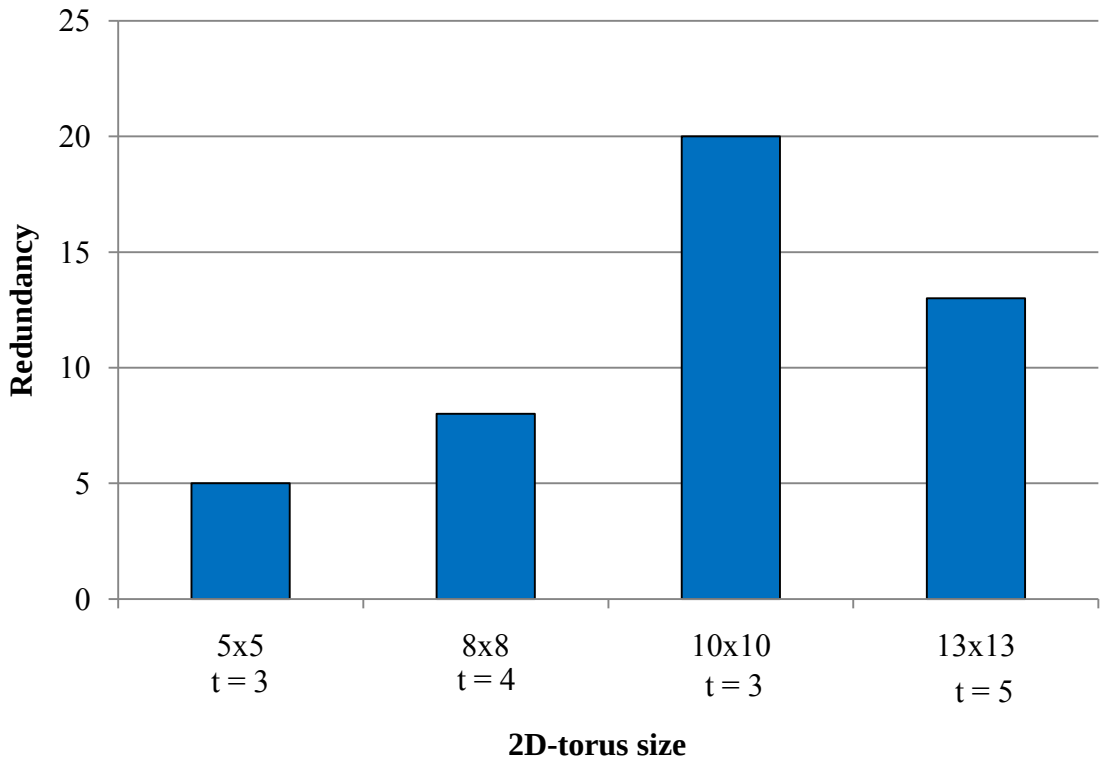

Fig. 22. Storage Redundancy for Various Tori Using DTVS

Fig. 22 shows the redundancy requirements for various t-interleaving schemes over different size 2D-tori using equation 2. In fact, for any *n* x *n* 2D-torus, the minimum required level of redundancy for a perfect t-interleaving is simply *n* since $|S_t|$ must divide *n*.

In Fig. 22, it is observed that the larger 13x13 torus requires less redundancy than the smaller 10x10 torus. This is due to the fact that 3-interleaving is used for the 10x10 torus (Lee spheres of size 5 are created), and 5-interleaving is used for the 13x13 torus (Lee spheres of size 13 are created). The 10x10 torus configuration contains 100 total nodes, and the test vectors are replicated across the 20 spheres created. The 13x13 torus configuration has 169 total nodes, and the test vectors are replicated across the 13 spheres created. Therefore, tests are replicated seven more times in the 10x10 torus configuration than the 13x13 torus configuration.

If less redundancy is desired, a designer may simply construct a sub-optimal t-interleaving over a torus. Another method to reduce redundancy requirements is to store file segments in only some nodes while still bounding test delivery distance. However, the main limitation of on-line testing noted in research literature has been test delivery distance and not storage.

G. Experimental Setup

1. NoC Simulator

To measure the performance of using this distributed test vector technique, we simulate a NoC-based SoC using the NoCSim on-chip network simulator [34]. NoCSim

is a SystemC cycle-accurate simulator which models IP cores, on-chip routers, CNI, and network links for any network topology to form a complete system. It takes as input an XML document which specifies the system topology and composition, and it reports activity latency, resource utilization and energy consumption.

Table 4 describes the baseline simulation configuration and parameters used during our experiments.

Table 4. DTVS Simulation Parameters

SoC Topology	2D Torus: 25, 64, 100, 169 cores
Test Vector Set Size	See Table 3
Network Configuration	64-bit flits, 8 flits per packet, 8 VCs per link, 8 flit buffer depth, 1 GHz, wormhole routing, credit-based flow control
Process Technology	90nm

The simulated on-chip network utilizes 64-bit links, and each IP core can transmit information in 64-byte packets. The test vector sets used in all experiments are 256KB in size; therefore, 4096 packets of information are transmitted from test source to test sink. The measurement of energy consumption is based on energy formulas developed in [58], and energy parameters developed in [60] which considers a system using 90nm technology.

2. System Architecture

For our experiments, we evaluate the performance differences between applying COLT with a centralized TI-IP architecture and the proposed distributed architecture for NoC-based SoCs organized as 2D-tori.

For the 2D-torus based systems, four systems are constructed for our experiments: a 25-node SoC in a 5x5 torus topology, a 64-node SoC in a 8x8 torus topology, a 100-node SoC in a 10x10 torus topology, and a 169-node SoC in a 13x13 torus topology. For each system, we simulate the behavior of delivering test vectors to each node in the system using a standard centralized approach described in [4], [5] and the proposed distributed protocol based on t-interleaving. We allow *n* cores to be tested simultaneously in each *n* x *n* configuration. For example, five cores are tested simultaneously in the 5x5 torus system.

For the 5x5 and 10x10 torus systems, 3-interleaving is used to distribute the test vectors across the network; the test vector set is split into 5 pieces, and each node can access all test vectors within 1 hop. For the 8x8 torus, 4-interleaving is used, therefore the test vector set is split into 8 pieces, and each node can access all test vectors within 2 hops. Finally, the 13x13 torus system uses 5-interleaving to distribute the test vectors. In this scheme, the test vector set is split into 13 pieces, and each node can access all test vectors within 2 hops.

3. System Cores

To assess the effectiveness of distributed test vector storage, this research considers the testing of the following IP cores: the OpenRISC CPU [53], the ARM7TDMI CPU [61], and a NoC router [62]. Both the OpenRISC and ARM7 cores are simple, in-order processing cores that are representative of the trend towards many, small processers in many-core SoC. The NoC router is an open-source design from the Stanford Concurrent VLSI Architecture Group which includes RTL and testbenches [62].

Each core was synthesized using Synopsys Design Compiler [43] with the Synopsys 90nm Generic Library [63], and scan chains were automatically inserted after synthesis. For each design, one scan chain is inserted. After synthesis, the ATPG tool, Synopsys Tetramax [64], was used to generate delay tests based on worst-path delay information. Fault coverage exceeded 97% for each design. To measure test time, the application of test vectors into each design was simulated at the gate level using an automatically generated testbench from the ATPG tool and the synthesized netlist. Table 5 summarizes the delay testing information for each of these cores. The size of each design is represented in gate equivalents (Gate Eq.).

Table 5. Core Test Information

Core	Gate Eq.	Vectors	Volume	Test Time
OpenRISC	773133	626	154KB	1186 µs
ARM7	34522	587	137KB	293.9 µs
Router	38617	121	65KB	60.9µs

From Table 5, it is observed that the OpenRISC core is by far the most complex IP core of the three cores analyzed for this research, and the OpenRISC core also has the longest test application time of the three cores. Although the router is almost as large as the ARM7 core in terms of gate equivalents, its test application time is much shorted due to the simplicity of its architecture.

H. Experimental Results

The goal of any concurrent online testing scheme is to minimize application intrusion. Therefore, the system impacts of COLT are measured in terms of system test latency – the amount of time required to test all cores of the SoC, and system test energy consumption, which is critical in safety-critical embedded applications. Additionally, the effect of the order of core testing is measured to determine the effect of network and resource contention on system test latency. Finally, the Test Controller (TC) portion of the CNI, which implements the proposed distributed COLT scheduling protocol is synthesized to determine area and power overhead.

1. System Test Latency

System test latency, or the amount of time required to test each core of the SoC using COLT, is a critical parameter in determining the feasibility of including a COLT scheme in a SoC.

As a baseline case, we executed the centralized COLT protocol on systems composed of the OpenRISC, ARM7, and NoC router cores described in Section G. The test delivery time for each case is the average delivery time for testing all cores in the SoC. Fig. 23 illustrates the results of this experiment.

The relative overhead of test delivery is highest in the NoC router design: in the 13x13 SoC, the average time required for delivery of test vectors to the router under test accounts for 90% of the total testing time. For the ARM7 processor, test delivery time accounts for roughly 70% of the total testing time across the SoCs of varying size. Testing the OpenRISC processors requires the longest total time on average; however, test delivery time accounts for approximately 50% of the total test time across all studied SoCs.

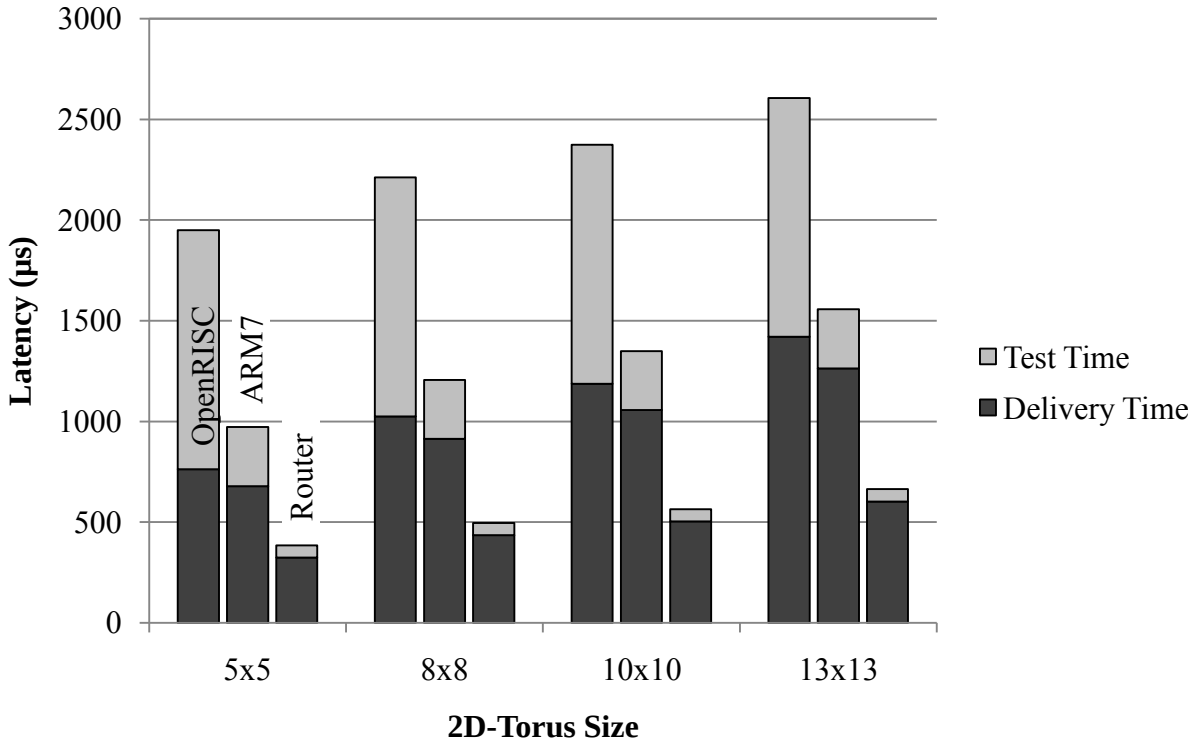

Fig. 23. Test Application and Delivery Times in Centralized COLT

Fig. 23 illustrates the performance differences between using a standard centralized COLT protocol and the proposed approach when testing the ARM7 processor core. In each n x n SoC, *n* tiles are replaced with TI-IP when using the centralized COLT scheme as described in [4] to ensure that n cores are tested simultaneously throughout the simulation for fairness.

For the 5x5 2D-torus system, distributed COLT can test each core of the system in 412µs, while the centralized COLT scheme requires 972µs to test each core of the entire system. This equates to a 58% reduction in system test time when using distributed COLT. System test latency improvements increase with SoC size as Fig. 24

shows; at the 169-core 13x13 SoC, distributed COLT can test each core in 390µs, while centralized COLT requires 1558µs. This equates to a 75% reduction in system test time.

In safety-critical applications with real-time constraints, shortening the system test time as much as possible is critical to ensuring that tests and applications can meet their deadlines.

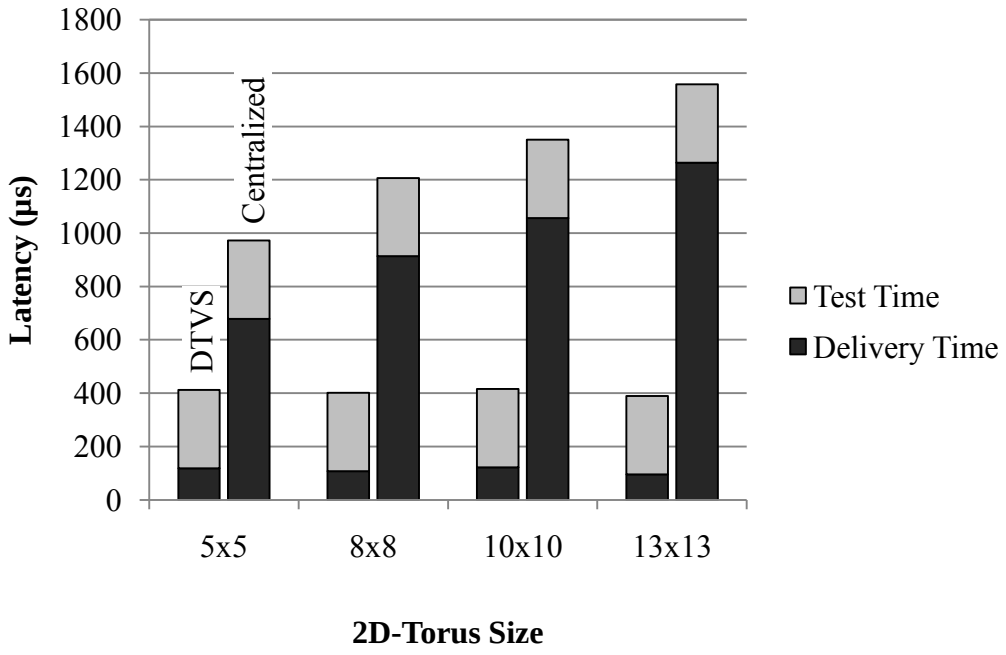

Fig. 24. Scalability of System Test Latency in 2D-Tori SoC

From Fig. 24, it is observed that test time for each core is constrained by the circuit's test application time when using DTVS. When using the centralized COLT scheme, total test time is bounded by the delivery of test vectors. Furthermore, the delivery time for each core increases linearly when using a centralized scheme, unlike DTVS, which shows no increase in delivery time as the system scales.

2. Energy Consumption of Test Delivery

Increasing test vector delivery distances affect energy consumption due to test, just as latency is affected as demonstrated in Section VIII.A. Energy consumption is a critical factor of many safety-critical applications, since a substantial portion of these applications run on mobile systems where a finite amount of energy is available. Network energy parameters used in these experiments are based on the results obtained in [58], [60]. In the following experiments, a test vector set of 256KB in size is assumed.

Fig. 25 illustrates the effect of SoC size on energy consumption during COLT. Note that energy consumption is represented on a log scale due to the exponential growth of energy consumption as SoC size increases.

For the 5x5 2D-torus system, using the proposed distributed COLT protocol results in a 83% reduction in energy consumption for an entire system test. Please note that the energy consumption for Distributed COLT within a 5x5 Torus system is 0.85mJ and therefore cannot be seen on the graph. As with latency, energy consumption improves as the SoC size increases. For the 13x13 SoC, energy consumption is reduced by 93%.

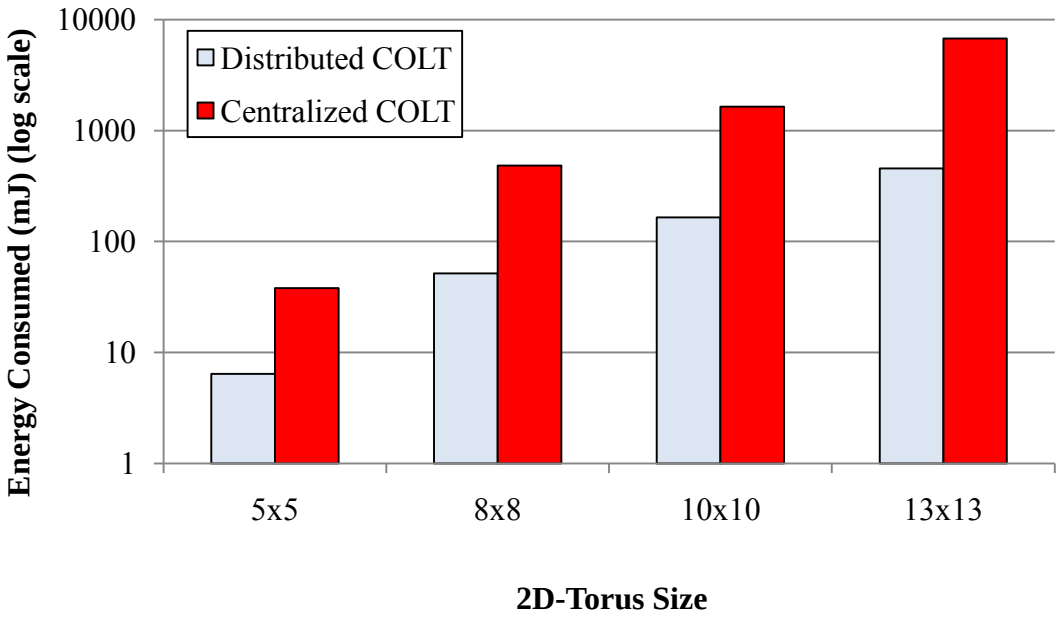

Fig. 25. Scalability of Test Delivery Energy Consumption

Reducing the energy consumption by an order of magnitude across all SoC sizes analyzes is a significant factor in determining whether using COLT with DTVS is feasible for safety-critical SoC. Many safety-critical applications have very constrained energy and power budgets, and any savings in energy translate into more technically capable systems.

3. Effect of Traffic Load on Testing

Since this research focuses on concurrent online testing, it is important to study the effects of other network traffic on test delivery. We model background application traffic by injecting flits into the network at varying levels. These flits are generated with random destinations to produce “white noise” background traffic. We vary this load from no traffic to 0.5 flits/cycle/node in increments of 0.1 flits/cycle/node.

For this experiment, each core of the SoC initiates an on-line self-test with distributed test vector storage, using the same 2D-torus configurations described in Section VII.B. Before initiating the first on-line test, a warm-up time of 1000 cycles was used to bring router activity to a steady state.

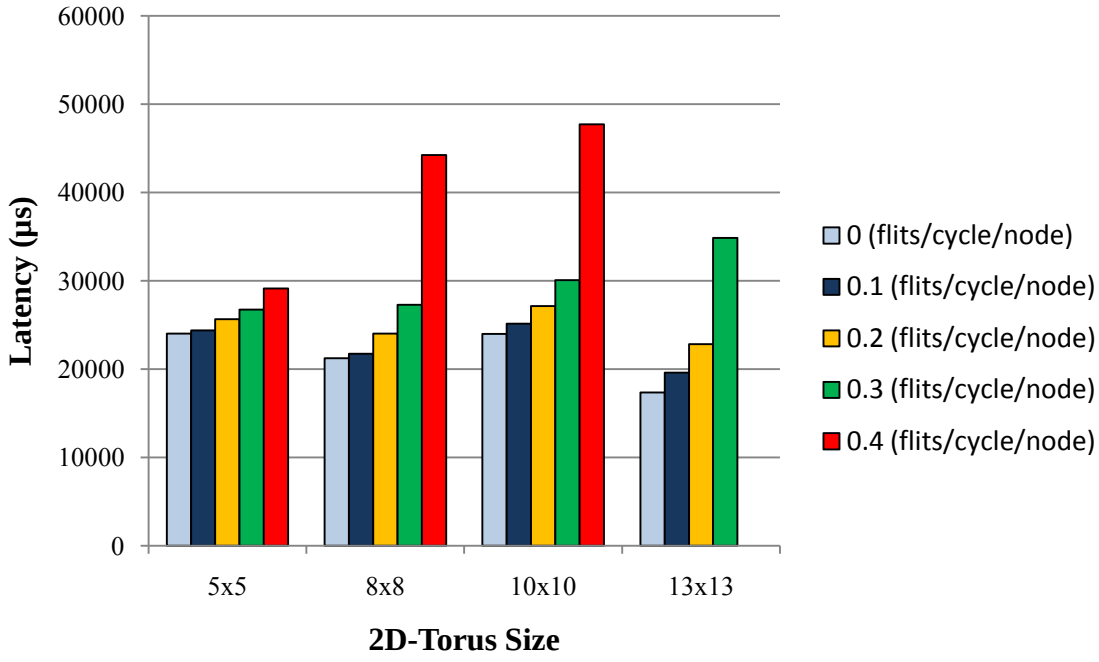

Fig. 26. Effect of Network Traffic on Test Latency

Fig. 26 illustrates the results of these experiments. We see that up to a reasonable level of background traffic (0.3 flits/cycle/node), test delivery latencies are not greatly affected. For the 25-core 5x5 2D-torus SoC, latency is increased by 21%, from 23ms to 29ms. For the 64 and 100-core SoC, latencies are increased by 28% and 25% respectively from no traffic to 0.3 flits/cycle/node. At 0.4 flits/cycle/node in these SoC, network saturation effects begin to dominate network behavior. For the 169-core 13x13 2D-torus SoC, latency doubles from 17ms to 34ms between no traffic and 0.3

flits/cycle/node added traffic. An injection rate of .4 flits/cycle/node with uniformly random destinations in the 13x13 2D-torus completely saturated the network; therefore, results for that case are not shown.

4. Effect of Core Test Scheduling

Distributed test vector storage allows a core under test to access the complete test vector set stored across its neighbors within a certain radius. Specifically for torus-based on-chip networks, a communication pattern in the form of a Lee-metric sphere of radius *r* centered at the core under test is created. In other words, the core under test will communicate with neighboring cores at most *r* hops away. Therefore, no two cores with overlapping Lee-metric spheres of radius *r* should be tested simultaneously. This experiment shows the effect of testing cores with overlapping test vector communication compared to using the proposed Code-Division Core Test Scheduling; the time to test all cores of the SoC using a standard centralized approach is also included for the purposes of comparison.

Fig. 27 illustrates the effect of resource conflicts due to improper core test ordering. For this experiment, four scheduling algorithms are used to determine which cores of a 5x5, 8x8 and 10x10 2D-torus based SoC are tested simultaneously: Random, Code, Linear, and Centralized. In the random scheduler, cores are chosen at random to be tested such that each core is tested once until the entire SoC is tested; this algorithm is used as a baseline for comparison. Simulations using the random scheduler were run until a latency result converged. The Code scheduler is the proposed algorithm described in Section D. The Linear scheduler simply tests all n cores of a n x n SoC line-by-line.

The Centralized scheduler is the algorithm assumed by previous research and does not use distributed test vector storage.

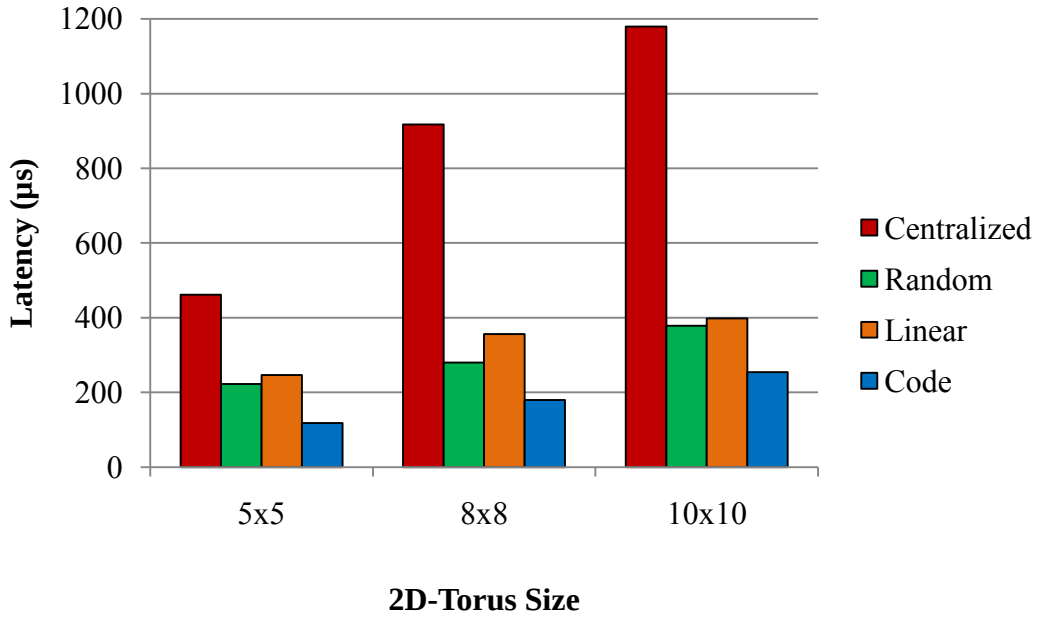

Fig. 27. Effect of Core Test Scheduling

As Fig. 27 shows, the proposed Code-Division based algorithm reduces system test latency by approximately 40% compared to both the random and linear schedulers. This demonstrates that resource conflict, in terms of network congestion and shared test vector memory usage, is a significant contributor to system test latency.

5. Distributed Test Controller Overhead

To determine the area and power costs of including the distributed COLT-based Test Controller (TC) in each CNI of the system, and HDL model of the TC was developed. This model was functionally verified using Verilog testbenches.

The HDL model was synthesized using Synopsys Design Compiler [43] and the Synopsys 90nm Generic Library [63]. The gate count of the TC was estimated to be 5.8K, while power consumption was estimated to be 6.2mW. This overhead includes the realization of the scheduling state machine, test vector and response buffers, core state buffers to save the microprocessor state, and test controller mechanisms such as the SCAN test interface.

6. Test Vector Memory Overhead

As stated in previous sections, more on-chip storage will be required to implement the distributed test vector storage scheme used with the proposed distributed COLT protocol when compared to centralized schemes. For most systems, n copies of the test vector set will need to be stored on-chip for a SoC of size n x n. However, as on-chip cache sizes increase exponentially as feature size shrinks, test vector storage will consume a relatively small percentage of on-chip memory.

For example, a 25-core SoC using the OpenRISC processor core would require approximately 770KB of test vector storage using distributed COLT. Due to these increased storage requirements, this research envisions that distributed COLT will be most applicable for safety-critical applications.

I. Conclusion

Concurrent online test of many-core SoC is only feasible if application intrusion is sufficiently reduced, allowing user applications to function correctly while detecting the formation of hard errors due to early-life failure and electronic wear-out quickly.

Previous research has shown that the most significant contributor to application intrusion is the delivery of test vectors to each core within the system. In addition, this research has demonstrated that the greatest contributor to application intrusion of COLT is the delivery of test vectors, based on the testing of two processing cores and an on-chip router.

We propose the use of t-interleaving to optimally distribute test vectors across the SoC to minimize the impact of online testing on system functionality. By minimizing and bounding the distances test vectors must travel across a chip, system designers can better estimate how testing will intrude into system applications. Additionally, test vector delivery latency and energy consumption is dramatically reduced, allowing for thorough online testing to be used in low-power and energy efficient systems.

CHAPTER V

APPLICATION-AWARE ONLINE TESTING

A. Introduction

Many safety-critical systems, including avionics and automotive systems, are rapidly integrating more functionality into single devices. These devices can host applications of many levels of criticality, termed mixed-criticality systems, in order to reduce the space, weight and power (SWaP) costs of such systems. Within avionics, this trend is called *Integrated Modular Avionics* (IMA); a single IMA system can control every aspect of the aircraft and replaces many separate single-function systems [65]. Within the next decade, these embedded systems will transition from multi-core systems-on-chip (SoC) to many-core SoC using a network-on-chip (NoC) for inter-core communication, replacing the typical on-chip bus used today [benini]. These many-core systems will contain dozens to hundreds of processing, memory and interface cores capable of running many applications simultaneously.

Beyond these functional changes, physical changes will accelerate over the next decade. As feature sizes shrink to 32nm and below, these systems become more susceptible to early life failure (ELF) and electronic wearout [1], [13]. To mitigate against this increasing vulnerability, these devices will require frequent, online testing to ensure sufficient reliability and availability [4], [5], [16].

The design requirement for online testing is already entering today's safety-critical embedded devices. For example, the Freescale MPC564xL platform allows for

online SCAN testing and diagnosis in the event that a system component experiences hard failure due to wearout in the field [66].

Previous research has proposed *Concurrent Online Testing* (COLT), which allows for online tests and user applications to run on the SoC concurrently [1], [4], [5], [16]. COLT satisfies the reliability requirements met by online testing while also providing a sufficient level of system availability.

A key design aspect for COLT is *application interference*—the costs associated with online testing that include additional energy consumption, increased NoC traffic load, and core under test (CUT) downtime. The work described in previous chapters has attempted to optimize these costs; however, little work has been done in measuring the effect of COLT on application execution times.

Measuring and bounding application execution times in safety-critical systems is extremely important. These systems typically have hard realtime requirements, and any application that misses its deadline could cause catastrophic failure. Therefore, the effect of COLT on application execution times must be well understood.

Any COLT technique requires the delivery of test vectors from test sources to the CUT, and this delivery process manifests as increased NoC traffic. We have measured the effect of increased NoC traffic due to COLT on application execution times from the MiBench embedded benchmark suite [67] using the experimental setup described in Section D. Automotive, telecommunication, networking and security applications were chosen to represent safety-critical applications. Execution times of a variety of

benchmarks increased by an average of 17% while under interference, motivating the need to understand the effect of COLT on applications.

In this work, we propose and analyze application-aware online testing of many-core SoC. By respecting the deadline requirements of safety-critical applications during COLT, we allow the system designer to make a tradeoff between test speed and execution time stability.

This work makes the following contributions:

- To the best of our knowledge, this is the first analysis of the effect of online testing on application execution time, a primary aspect of application intrusion
- Two methods for minimizing interference during online testing are proposed and analyzed: *Test Vector Delivery Blocking* and *Test Vector Storage Redundancy*
- Overhead of proposed schemes determined by implementing distributed test controller

The remainder of Chapter V is organized as follows: Section B describes COLT and explains the need for distributed COLT solutions for many-core SoC. Section C introduces the proposed methods of minimizing application interference and the associated system architecture. Section D details the experimental platform, system assumptions and experiment set, and Section E presents the results of the evaluation in terms of testing and application performance and overhead. Concluding remarks are presented in Section F.

B. Application-Aware Online Testing Architecture

In previously proposed method of distributed COLT, application interference through the NoC was not considered. Following the distributed COLT protocol described in Chapter IV, a core under test would request test vectors from its neighboring tiles, and those neighboring tiles would respond immediately. In a non-critical environment, ignoring this interference may be acceptable. However, if hard deadlines must be met to avoid catastrophic failure, then the distributed COLT protocol must consider the case where one or more neighboring tiles cannot respond with test vectors immediately.

In this work, we describe two complimentary methods of allowing safety-critical applications and online tests to run concurrently within a many-core SoC while minimizing interference. These two methods are termed Test Vector Delivery Blocking and Test Vector Storage Redundancy and are described in the following subsections.

1. Test Vector Delivery Blocking

Test Vector Delivery Blocking is the simple method of requiring any tile running a safety-critical application to complete its task before delivering test vectors to a core under test. This differs from previously proposed COLT techniques that begin the delivery of test vectors immediately upon request.

In order to alert the test controller within the CNI that a safety-critical application is running, we propose adding a memory-addressable test configuration register to the CNI which provides software access. When an application enters a safety-critical

section, a "safety-critical" flag within the configuration register is set to TRUE, denying any test request from neighboring tiles to be satisfied. Once the safety-critical section of the application completes, the "safety-critical" flag is set to FALSE, and any pending test requests can then be fulfilled.

From the perspective of the core under test, test request messages are sent to its neighbors seeking test vectors, and the CUT begins to receive these test vectors from neighboring tiles that are not currently running safety-critical software. If any of its neighbors are running safety-critical software, the CUT is blocked from receiving the complete test vector set and running a full test.

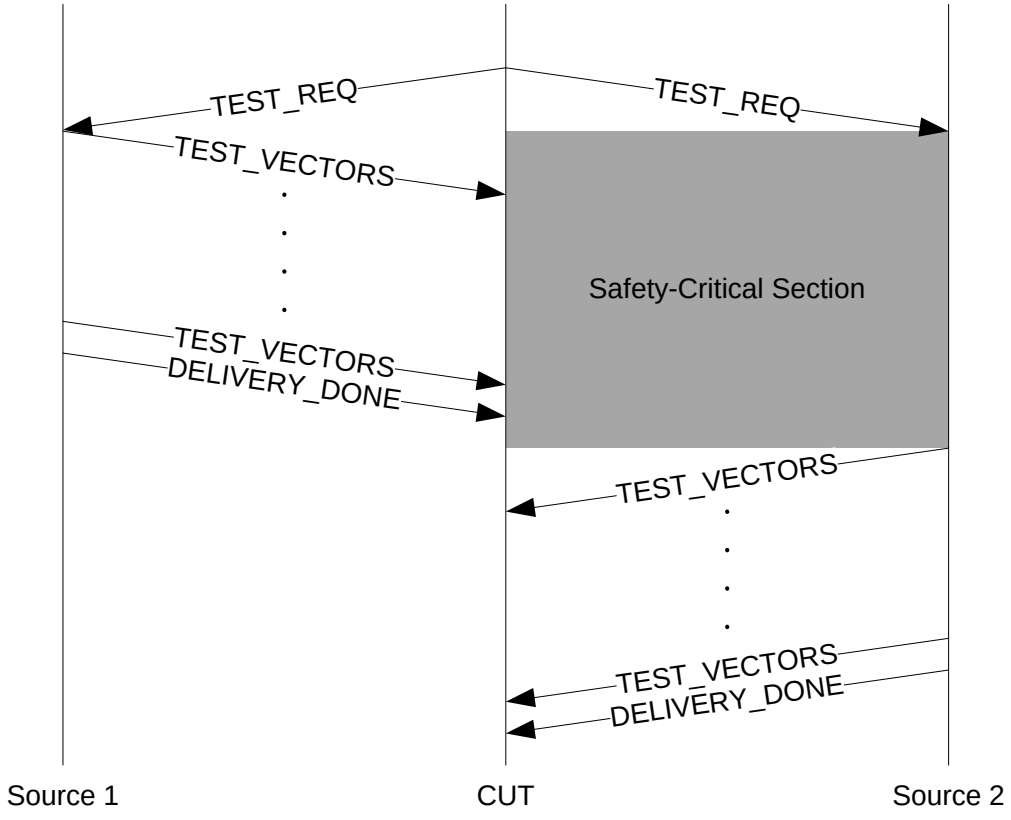

Fig. 28. Test Vector Delivery Blocking Protocol

Fig. 28 illustrates a simple example of a CUT requiring test vectors from two neighbors where one neighbor is in a safety-critical section during a test request.

2. Test Vector Storage Redundancy

Test Vector Storage Redundancy is the method of using *erasure codes* to eliminate the requirement that a core under test must access all of its neighbors' test vectors to construct a complete test vector set. An erasure code is an error-correcting code of length n, created from a k-length message (where $n > k$) such that the original message can be constructed from a subset of the n parts. The simplest example of an erasure code is the parity code of length $k+1$, where a k-length message is appended with an additional term—the sum of the terms of the original message. If any one term of the $k+1$ code is missing, the original message can be recovered.

This coding technique can be used for the distributed storage of test vectors across a SoC. Instead of directly dividing the complete test vector set into k segments and distributing it as proposed in Chapter IV, the test vector segments can be encoded into an erasure code of length n, and these n segments are distributed.

The cost of using storage redundancy include increased storage requirements equal to the code rate selected and the decoding circuitry required to transform the encoded test vectors into their original form. In the case of parity codes, this circuitry is extremely small and simple, and it can be considered negligible.

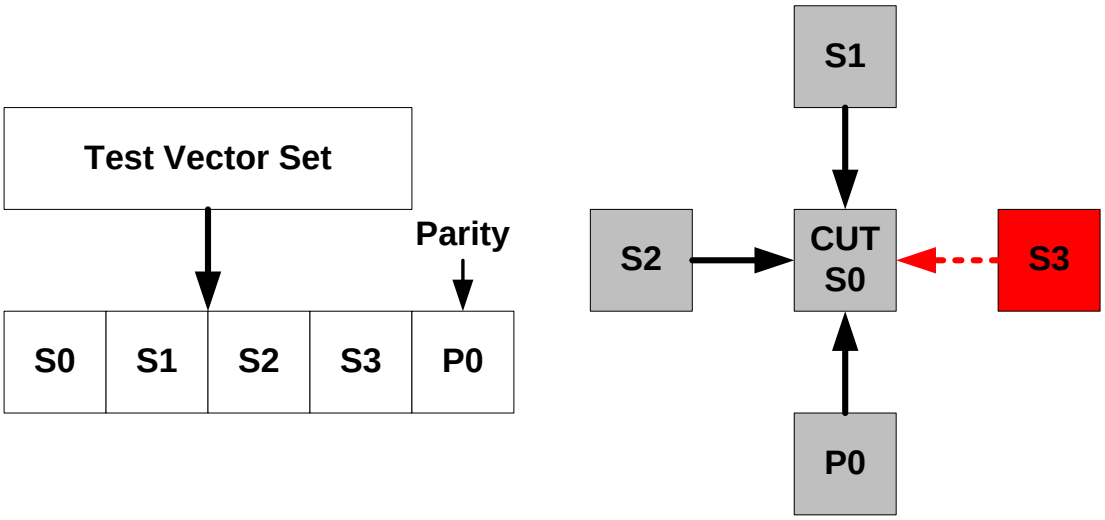

Fig. 29. Example of Test Vector Storage Redundancy

Fig. 29 illustrates an example where one neighbor, S3, is running safety-critical software; however, the CUT can access its three other neighbors and can construct a complete test vector set without delay. This is possible because the test vector set is originally divided into four segments, and a fifth, redundant segment is created through parity generation to form an erasure code. These five segments are stored within each SoC tile through 3-interleaving, as done in the example shown in Fig. 18. Through the generation of a parity code, the full test vector set can be constructed from any four of the five segments.

Test Vector Storage Redundancy provides flexibility and fault-tolerance to distributed COLT. A CUT may lose one of its test vector sources, either temporarily or permanently, and still retain the capability to access its complete test vector set.

3. Using These Methods in Combination

Depending on application-specific requirements, either of these methods can be used in isolation, or both can be used in combination. A system that has very strict storage overhead requirements is better suited to using Test Vector Delivery Blocking only. Whereas, a system that has very strict online test deadlines but can tolerate some level of application interference will prefer to use Test Vector Storage Redundancy.

Strictly using storage redundancy, application interference still may occur. Consider the situation where storage redundancy can tolerate one neighbor running safety-critical code during a test request. However, in a particular instance, more than one neighbor of a CUT is running safety-critical software. In this situation, all but one of the tiles running safety-critical software must begin sending test vectors to the CUT immediately. This can be considered a "best effort" approach to avoiding interference.

If Test Vector Delivery Blocking and Test Vector Storage Redundancy are used in combination, the situation above can be avoided if necessary. Storage redundancy allows a CUT to test itself using only a subset of its neighbors, and even if that subset is not available, delivery blocking can be enabled. Using these methods in combination can provide a good compromise between testing performance and application performance, and we evaluate each of these possibilities in Section D.

C. Experimental Setup

1. System Architecture

To measure the performance of the proposed COLT methods, we simulate a 100-core NoC-based SoC using the NoCBench on-chip network and system simulator [34]. NoCBench is a SystemC cycle-accurate simulator which models processor and memory IP cores, on-chip routers, CNI, and network links for any network topology to form a complete system. Table 6 describes the baseline simulation configuration and parameters used during our experiments.

2. Test and Application Parameters

The test vector sets used in all experiments are 256KB in size; therefore, 4096 packets of information are transmitted from test source to test sink during each COLT instance. To simulate safety-critical software, we have constructed a schedule of 10 task frames, where each frame has a duration of 200µs, and we vary the proportion of frames that are safety-critical within a schedule. Typical values are used for NoC parameters, and a 100 core SoC is simulated to understand the behavior of COLT in a many core SoC. These NoC and system parameters are described in Table 6.

Table 6. Application-Aware COLT Simulation Parameters

SoC Topology	100 cores with 2D-torus
Test Vector Set Size	256KB, 5 segments
Network Configuration	64-bit flits, 8 flits per packet, 8 VCs per link, 8 flit buffer depth, 1 GHz, wormhole routing
Processor Core	SPARC-V8, 32KB L1 I/C Caches

D. Experimental Results

To evaluate the COLT methods proposed in this work, we focus on measuring the duration of application interference of COLT under varying levels of application criticality. Each of the two methods proposed—Test Vector Delivery Blocking and Test Vector Storage Redundancy—are evaluated in isolation and in combination. Overheads of these methods, in terms of test vector storage requirements and test controller area and power consumption, are also evaluated against previously proposed COLT techniques.

1. Application Interference of COLT

As described in Section I, it is important to note the effect of application execution time due to COLT. By running actual benchmarks with COLT from the embedded MiBench suite [67] representing automotive, telecommunications, networking and security applications, we are able to observe the effect of COLT on execution times. As shown in Fig. 30, we have observed an average increase of 17% in execution times during COLT interference. An exception to this is the automotive application bitcount,

which experienced only a 6% increase in execution time. This is due to bitcount's instruction composition, which includes very little memory transactions and is therefore fairly immune to NoC traffic interference [67].

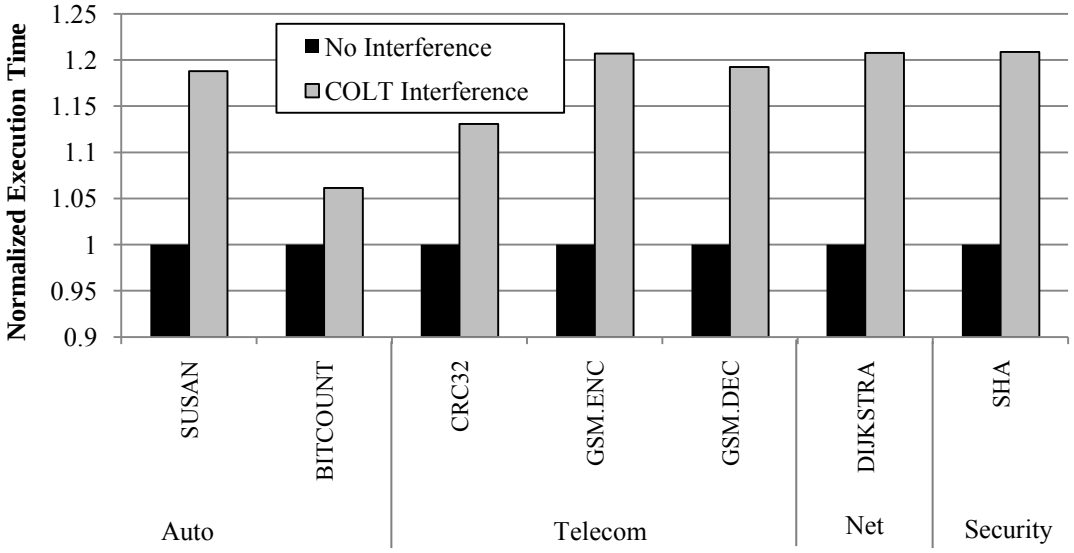

Fig. 30. Effect of NoC Traffic on Execution Time

In general, a common increase in execution time is observed across each of these benchmarks when COLT runs concurrently with applications. This increase in execution time could have profound effects for a safety-critical system which relies on hard deadlines to avoid catastrophic failure.

2. Test Vector Delivery Blocking

Fig. 31 illustrates test delivery time for Standard COLT and Test Vector Delivery Blocking COLT over varying levels of software criticality. Additionally, the amount of time that Standard COLT interferes with safety-critical software is overlaid with the

delivery time data. As expected, Standard COLT shows no change in test delivery time; however, the amount of interference increases with the amount of safety-critical software running on the SoC. If test vector delivery is blocked during safety-critical sections of software, an increase in test delivery time is observed; however, there is no interference between COLT and safety-critical software.

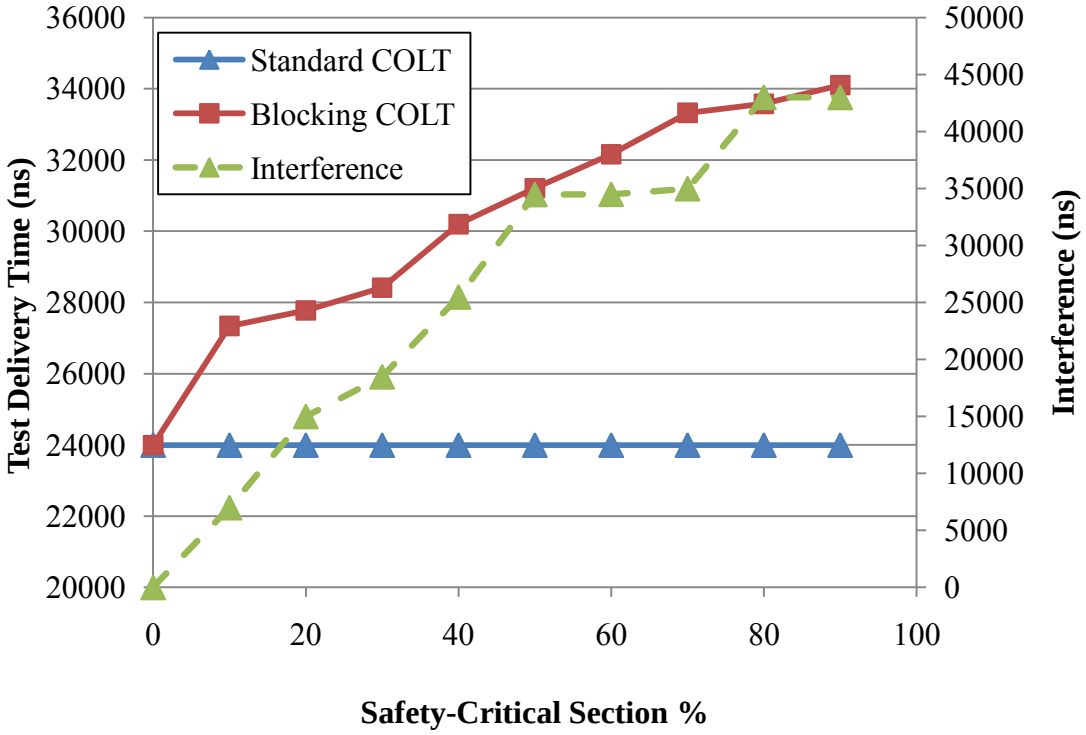

Fig. 31. Comparison of Standard COLT and Test Vector Delivery Blocking

As seen in Fig. 31, test vector delivery time increases from approximately 24µs to 34.1µs, an increase of 42%, as the SoC's proportion of safety-critical software increases from 0% to 90%. When the proportion of safety-critical software is 40%, the increase in test delivery time is 26%.

3. Test Vector Storage Redundancy

By using redundancy in test vector storage and delivery, the CUT only requires a subset of its neighbors to access the complete test vector set. In these experiments, the test vector set was divided into 4 segments, and one parity segment was generated as described in the example in Section B. This redundancy introduces a 20% overhead in test vector storage.

As Fig. 32 shows, using storage redundancy with COLT allows for a significant reduction in application interference under a variety of safety-critical loads. The greatest reduction in interference occurs between safety-critical section proportions of 10% and 70%; the average reduction of interference in these systems is 47%. Once the proportion of safety-critical software exceeds 70%, the benefits of storage redundancy are reduced due to the fact that most cores of the SoC are executing safety-critical software most of the time—interference is largely unavoidable.

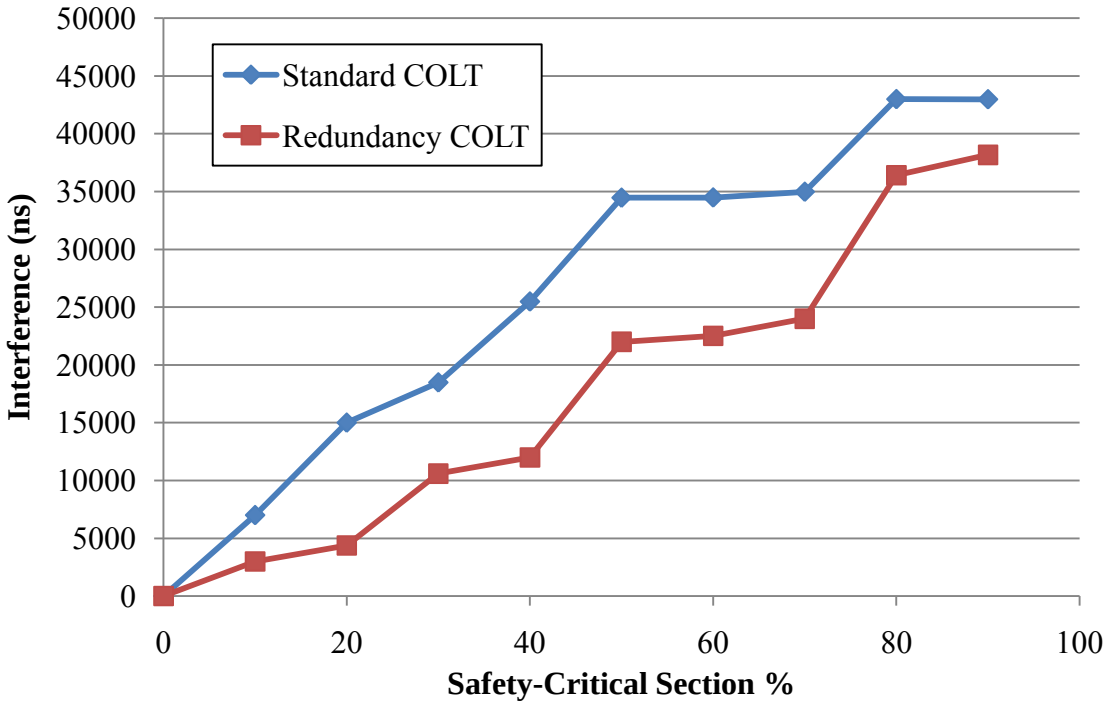

Fig. 32. Interference Reduction of Storage Redundancy

Apart from application interference, it is also important to observe the effect of using storage redundancy with COLT on test vector delivery times. As Fig. 33 shows, test vector delivery times decrease as the proportion of safety-critical software increases. On average, the test delivery time is increased by 15%. This may seem counter-intuitive; however, an explanation is found when looking at the nature of storing and delivering redundant test vectors.

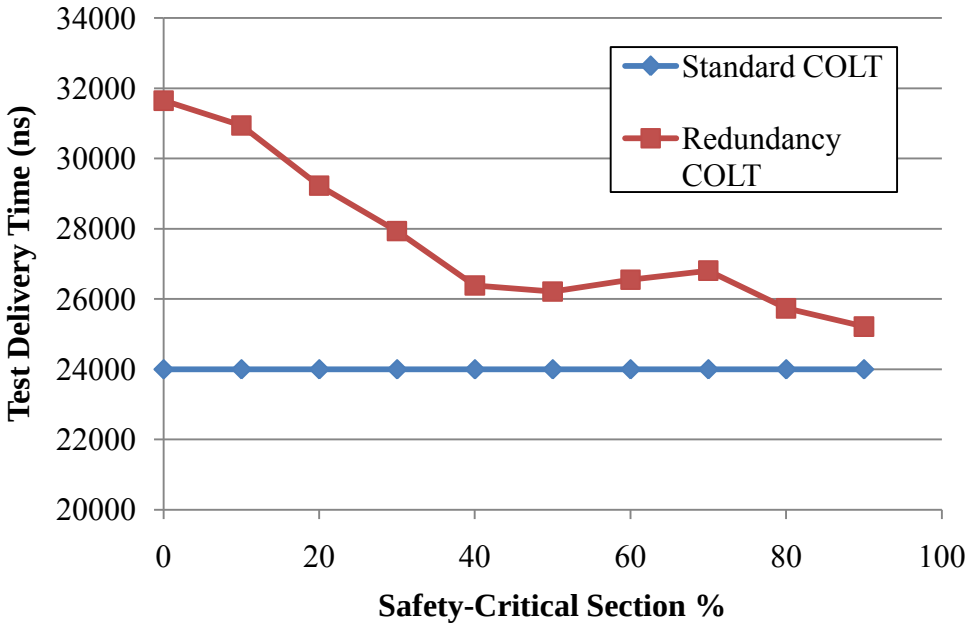

Fig. 33. Delivery Times of Standard COLT and Storage Redundancy COLT

When the proportion of safety-critical software is low (30% or less), a significant portion of the test vector traffic is redundant information—the parity segment as described in Section B. This redundancy increases the effective test vector delivery time of COLT. This redundant traffic may be used to verify the correctness of the test vector traffic in the event of transmission or storage errors, but that benefit is outside the scope of this dissertation.

4. Combination of Blocking and Redundancy

As described in Section B, both proposed schemes can be implemented together to allow both blocking and redundancy during COLT. In isolation, we have observed that blocking removes interference but has increasing test delivery times as the amount of safety-critical software increases. Alternately, storage redundancy has a decrease in

test delivery time as the amount of safety-critical software increases, and it also provides a reduced level of interference when compared to Standard COLT.

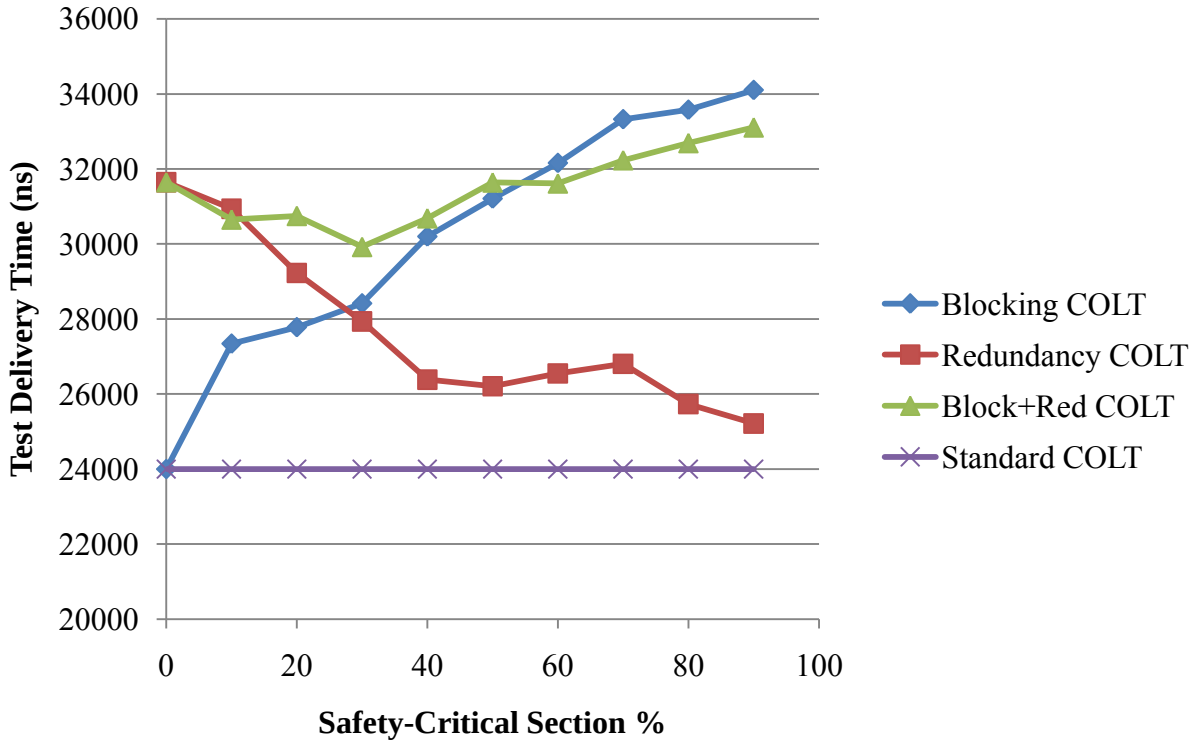

Fig. 34. Test Delivery Times of All COLT Schemes

Fig. 34 compares the test delivery times of all schemes: Standard COLT, Test Vector Delivery Blocking, Test Vector Storage Redundancy, and the combination of the two. We observe that the test vector delivery times of the combination (Block+Red COLT) remains relatively stable under all safety-critical proportions.

The combination suffers from the relatively high delivery times of the storage redundancy scheme during low amounts of safety-critical software. However, when the proportion of safety-critical sections exceeds 50%, the combination outperforms Test

Vector Delivery Blocking in isolation. It is important to note that the combination eliminates all application interference, just as blocking in isolation does.

5. Test Controller Overhead

To determine the area and power costs of adding the Test Vector Delivery Blocking and Test Vector Storage Redundancy schemes to the distributed COLT-based Test Controller (TC) in each CNI of the system, an HDL model of the TC was developed. This model was functionally verified using Verilog testbenches.

The HDL model was synthesized using Synopsys Design Compiler [43] and Oklahoma State University's 45nm FreePDK library [68]. The gate count of the TC was estimated to be 6.1K, while power consumption was estimated to be 6.52mW. This represents an approximately 5% increase in area and power costs over the standard distributed COLT-based TC proposed in Chapter IV.

E. Conclusion

We present and analyze two methods for mitigating application interference during COLT. Test Vector Delivery Blocking allows certain test sources to withhold test vectors when they are running safety-critical software. Test Vector Storage Redundancy uses erasure coding to provide a CUT with access to the complete test vector set with only a subset of its neighbors.

Based on our observations, the strengths and weaknesses of these approaches have been found. Test Vector Delivery Blocking appears to be the method of choice when the amount of safety-critical software running on the SoC is low. If there is a large

proportion of safety-critical software running on the SoC, then Test Vector Storage Redundancy or a combination of both schemes are the most appropriate choice.

CHAPTER VI

CONCLUSIONS

This dissertation presents a variety of techniques which assist COLT in becoming a more effective and feasible solution to achieve reliability and availability in safety-critical SoC. This research has focused on shortening the time between failures and test initiation, and it has also focused on optimizing the storage/performance tradeoff of delivering tests in many core SoC.

An on-demand testing mechanism, the ATTU, has been proposed and analyzed, and it is shown that this mechanism is effective and relatively inexpensive. Experiments show that the ATTU can detect approximately 80% of errors originating in processing cores that eventually manifest as NoC traffic anomalies. Based on system needs, a system designer can vary the amount of memory used to monitor and record the history of NoC traffic, allowing for course or fine-grain anomalous patterns. Additionally, the ATTU can be configured to trigger tests based on the number of anomalies detected.

COLT depends on stored tests to achieve a sufficiently high level of coverage; however, there has been no serious analysis of how the delivery and storage of these tests scales with the number of cores per SoC. There has also been no effort in identifying techniques that can scale the storage and delivery costs of tests as SoC enter the many core era. This research has applied a coding theory technique to distributed the storage of test vectors across the SoC, allowing for COLT to become scalable with many core SoC. By distributed the test vectors in this manner, any core within the SoC can

access the complete test vector set within a bounded distance. A DTVS test protocol is proposed and analyzed, and the hardware overhead associated with this protocol is determined. To evaluate the proposed test storage scheme, real IP cores are used in the generation, storage and application of test vectors. Through experiments, it has been shown that test delivery latency can be reduced by up to 90% for many core SoC, and the reduction in test latency improves as the number of cores per SoC grows. Additionally, a core test scheduling based on code division has been analyzed, and experiments show that test delivery latency can be reduced by 50% over other schedules possibly employed by DTVS.

To date, there has been no analysis on the effect of COLT on the execution times of applications. Through the use of a system simulator employing a NoC as the communication infrastructure, this is the first work to measure the effect of COLT on NoC traffic and the effect of this increased traffic on software execution times. Experiments show that application execution time can increase by 17% on average across a set of benchmarks while COLT is in operation. This increase in execution time has the potential to create serious consequences for safety-critical systems that depend on hard deadlines being met. Based on these findings, an application-aware COLT protocol is proposed and analyzed. Two methods of modifying COLT are proposed: test vector delivery blocking and test vector storage redundancy. These methods can either be used in isolation or in combination. Experiments show that application interference can be eliminated completely, or it can be reduced significantly based on the needs of the system designer. The increase in test delivery time by avoiding application

interference can increase beyond 40%; however, this is an acceptable tradeoff if application interference cannot be tolerated.

A. Future Work

There are several additional avenues for improvement of COLT that follow from the work described in this dissertation. To date, there has been no research effort in analyzing the security aspects of COLT. It is important to understand the vulnerabilities introduced by COLT, and adoption of COLT will depend on the ability of system designers to ensure that malicious attacks to the COLT mechanisms do not jeopardize the system.

As electronic wearout becomes a greater concern for safety-critical SoC, designing tests to be applied by COLT specifically for wearout will become a necessity. Currently, tests are generated based on fault models which capture the behavior of wearout failure but do not capture the emergence of those failures. As a specific example, the small-scale delay fault is a good candidate fault model for detecting electronic wearout. However, the delay fault model assumes that these faults can manifest anywhere within the system with equal probability—an assumption that is not accurate for electronic wearout. Wearout is more likely to occur in highly active, hot areas of the chip, and tests can be generated to specifically target these regions.

Another major area of future work includes a full-scale test of COLT within a real safety-critical SoC. The work described in this dissertation has measured the performance of COLT using only a few real IP cores as an experimental basis; however, real safety-critical SoC will contain many different IP cores. This increased system

complexity should reveal new challenges in applying COLT in safety-critical SoC and drive new areas of research.

A rarely discussed but extremely important consideration of COLT is the effect of testing on a core's temperature profile. It is well understood that testing leads to increased core temperatures due to increased switching activity, and this elevated temperature may remain for dozens of milliseconds after testing completes due to the relatively high thermal time constant of electronic devices. This "afterglow" of core temperature can have adverse performance effects for applications running immediately after COLT. To this point, all COLT research assumes that applications may begin running on a tested core immediately after COLT has been completed. Studying this thermal effect and creating techniques to mitigate this limitation will be extremely important to advancing the feasibility of COLT in future devices.

REFERENCES

[1] S. Borkar, “Thousand Core Chips—A Technology Perspective,” *Proc. IEEE/ACM Design Automation Conf. (DAC ’07)*, pp. 746-749, 2007.

[2] A.W. Strong, E. Wu, R.P. Vollertsen, J. Sune, G. LaRosa, and T. Sullivan, "Reliability Wearout Mechanisms in Advanced CMOS Technologies," *IEEE Press Series on Microelectronic Systems*, S.K. Tewksbury and J.E. Brewer, eds., Wiley-IEEE Press, Aug. 2009.

[3] J. Kao, S. Narendra, and A. Chandrakasan, "Subthreshold Leakage Modeling and Reduction Techniques," *Proc. IEEE/ACM Conf. Computer-Aided Design (ICCAD ’02)*, pp. 141-148, 2002.

[4] P. Bhojwani and R. Mahapatra, “An Infrastructure IP for On-Line Testing of Network-on-Chip Based SoCs,” *Proc. IEEE Int’l Symp. Quality Electronic Devices (ISQED ’07),* pp. 867-872, 2007.

[5] P. Bhojwani and R. Mahapatra, “A Robust Protocol for Concurrent On-Line Test (COLT) of NoC-based Systems-on-a-Chip,” *Proc. IEEE/ACM Design Automation Conf. (DAC ’07)*, pp. 670-675, 2007.

[6] R. Ronen, A. Mendelson, K. Lai, S.L. Lu, F. Pollack, and J.P. Shen, "Coming Challenges in Microarchitecture and Architecture," *Proc. of the IEEE*, vol. 89, no. 3, pp. 325-340, Mar. 2001.

[7] O. Mutlu, J. Stark, C. Wilkerson, and Y.N. Patt, "Runahead Execution: An Alternative to Very Large Instruction Windows for Out-of-Order Processors,"

Proc. 9th Int'l Symp. High-Performance Computer Architecture (HPCA '03), pp. 129-140, 2003.

[8] J.L. Hennessy and D.A. Patterson, *Computer Architecture: A Quantitative Approach*, 3rd Ed., Morgan Kaufmann, 2003.

[9] R. Hetherington, "The UltraSPARC T1 Processor - Power Efficient Throughput Computing," white paper, Sun Microsystems, Dec. 2005.

[10] L. Benini and G. De Micheli, "Networks on Chip: A New SoC Paradigm," *IEEE Computer*, vol. 35, no. 1, Jan. 2002.

[11] W.J. Dally and B. Towles, "Route Packets, Not Wires: On-Chip Interconnection Networks," *Proc. IEEE/ACM Design Automation Conf. (DAC '01)*, pp. 684-289, 2001.

[12] A. Avizienis, J.C. Laprie, B. Randell, and C. Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing," *IEEE Trans. Dependable and Secure Computing*, vol. 1, no. 1, pp. 11-33, Jan.-Mar. 2004.

[13] J. Bernstein, M. Gurfinkel, X. Li, J. Walters, Y. Shapira, and M. Talmor, "Electronic Circuit Reliability Modeling," *Microelectronics Reliability*, vol. 46, no. 12, pp. 1957-1979, Dec. 2006.

[14] M. White and Y. Chen, *Scaled CMOS Technology Users Guide*, tech. report JPL 08-14, Jet Propulsion Lab., California Institute of Technology, 2008.

[15] M.L. Bushnell and V.D. Agrawal, *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*, Springer, 2000.

[16] Y. Li, S. Makar, and S. Mitra, “CASP: Concurrent Autonomous Chip Self-Test Using Stored Test Patterns,” *Proc. Design, Automation, and Test in Europe (DATE ’08)*, pp. 885-890, 2008.

[17] Y. Li, O. Mutlu, and S. Mitra, “Operating System Scheduling for Efficient Online Self-Test in Robust Systems,” *Proc. IEEE/ACM Int’l Conf. Computer-Aided Design (ICCAD ’09)*, pp. 201-208, 2009.

[18] D. Lorenz, G. Georgakos, and U. Schlichtmann, "Aging Analysis of Circuit Timing Considering NBTI and HCI," *Proc. 15th IEEE On-Line Testing Symp. (IOLTS ’09)*, pp. 3-8, 2009.

[19] M. Agarwal, V. Balakrishnan, A. Bhuyan, K. Kim, B. Paul, W. Wang, B. Yang, Y. Cao, and S. Mitra, "Optimized Circuit Failure Prediction for Aging: Practicality and Promise," *Proc. IEEE Int’l Test Conf. (ITC ’08)*, pp. 1-10, 2008.

[20] Y. Zorian, “Guest Editor’s Introduction: What is Infrastructure IP?,” *IEEE Design & Test of Computers*, vol. 19, no. 3, pp. 3-5, May-June 2002.

[21] P. Bhojwani and R. Mahapatra, “Core Network Interface Architecture and Latency Constrained On-Chip Communication,” *Proc. IEEE Int’l Symp. Quality Electronic Devices (ISQED ’06)*, pp. 363-368, 2006.

[22] S. Shyam, “Ultra Low-Cost Defect Protection for Microprocessor Pipelines,” *Proc. ACM ASPLOS*, pp. 73-82, 2006.

[23] Sun Microsystems, "OpenSPARC," http://www.opensparc.net, June 2010.

[24] N. Kranitis, A. Paschalis, D. Gizopoulos, and G. Xenoulis, “Software-Based Self-Testing of Embedded Processors,” *IEEE Trans. Computers*, vol. 54, no. 4, pp. 461-475, Apr. 2005.

[25] V. Singh, M. Inoue, K.K. Saluja, and H. Fujiwara, “Instruction-Based Self-Testing of Delay Faults in Pipelined Processors,” *IEEE Trans. Very Large Scale Integration*, vol. 14, no. 11, pp. 1203-1215, Nov. 2006.

[26] G. Xenoulis, D. Gizopoulos, M. Psarakis, and A. Paschalis, "Instruction-Based Online Periodic Self-Testing of Microprocessors with Floating-Point Units," *IEEE Trans. Dependable and Secure Computing*, vol.6, no.2, pp.124-134, Apr.-June 2009.

[27] D. Gizopoulos, "Online Periodic Self-Test Scheduling for Real-Time Processor-Based Systems Dependability Enhancement," *IEEE Trans. Dependable and Secure Computing*, vol.6, no.2, pp.152-158, Apr.-June 2009.

[28] Tilera Corporation, "Tilera," http://www.tilera.com, June 2010.

[29] M.B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodra, B. Greenwald, H. Hoffman, P. Johnson, J.W. Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, and V. Strumpen, "The Raw Microprocessor: A Computational Fabric for Software Circuits and General-Purpose Programs," *IEEE Micro*, pp. 25-35, Mar. 2002.

[30] J. Srinivasan, S.V. Adve, P. Bose, and J.A. Rivers, "Exploiting Structural Duplication for Lifetime Reliability Enhancement," *Proc. 32nd Int’l Symp. Computer Architecture (ISCA ’05)*, pp. 520-531, 2005.

[31] L. Bolzani, M. Rebaudengo, M.S. Reorda, F. Vargas, and M. Violante, "Hybrid Soft Error Detection by Means of Infrastructure IP Cores," *Proc. IEEE Int'l On-Line Testing Symp. (IOLTS '04)*, pp. 79-84, 2004.

[32] E. Cota, L. Carro, F. Wagner, and M. Lubaszewski, "Power-Aware NoC Reuse of the Testing of Core-Based Systems," *Proc. IEEE Int'l Test Conf. (ITC '03)*, pp. 612-621, 2003.

[33] J.L. Henning, "SPEC CPU2006 Benchmark Descriptions," *ACM SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 1-17, Sept. 2006.

[34] S.K. Mandal, N. Gupta, A. Mandal, J. Malave, J.D. Lee, and R. Mahapatra, "NoCBench: A Benchmarking Platform for Network on Chip," *Proc. Workshop Unique Chips and Systems (UCAS '09)*, pp. 1-6, 2009.

[35] T. Dumitras and R. Marculescu, "On-Chip Stochastic Communication," *Proc. Design, Automation and Test in Europe (DATE '03)*, pp. 790-795, 2003.

[36] C. Grecu, A. Ivanov, R. Saleh, E.S. Sogomonyan, and P. Pande, "On-Line Fault Detection and Location for NoC Interconnects," *Proc. IEEE Int'l On-Line Testing Symp. (IOLTS '06)*, pp. 145-150, 2006.

[37] C. Ciordas, T. Basten, A. Radulescu, K. Goossens, and J.V. Meerbergen, "An Event-Based Monitoring Service for Networks on Chip," *ACM Trans. Design Automation of Electronic Systems*, vol. 10, no. 4, pp. 702-723, Oct. 2005.

[38] A. Webb, *Statistical Pattern Recognition*, E. J. Arnold, 1999.

[39] J. Hu and R. Marculescu, "Energy- and Performance-Aware Mapping for Regular NoC Architectures," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 4, pp. 551-562, Apr. 2005.

[40] OCP-IP Organization, "OCP International Partnership," http://www.ocpip.org, June 2010.

[41] OSC Initiative, "SystemC," http://www.systemc.org, June 2010.

[42] W. Qin, "SimIt-ARM," http://simit-arm.sourceforge.net, June 2010.

[43] Synopsys, "Design Compiler 2010," http://www.synopsys.com/Tools/Implementation/RTLSynthesis/Pages/DesignCompiler2010-ds.aspx, June 2010.

[44] J.B. Sulistyo, J. Perry, and D.S. Ha, *Developing Standard Cells for TSMC 0.25um Technology Under MOSIS DEEP Rules*, tech. report TR VISC-2003-01, Dept. Electrical and Computer Engineering, Virginia Tech Univ., 2003.

[45] J.B. Sulistyo and D.S. Ha, "A New Characterization Method for Delay and Power Dissipation of Standard Library Cells," *VLSI Design*, vol. 15, no. 3, pp. 667-678, 2002.

[46] Intel Corp., "Intel Tera-scale Computing Research Program," http://www.intel.com/research/platform/terascale, June 2010.

[47] Intel Corp., "Intel Tera-scale Single-Chip Cloud Computer," http://techresearch.intel.com/articles/Tera-Scale/1826.htm, June 2010.

[48] "Embedded Multicore: An Introduction," white paper, Freescale Semiconductor, July 2009.

[49] Freescale Semiconductor, "Freescale Semiconductor QorIQ P4080 Communications Processor," http://www.freescale.com/webapp/sps/site/prod_summary.jsp?fastpreview=1&code=P4080, June 2010.

[50] S. Borkar, "Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degredation," *IEEE Micro*, pp. 10-16, Nov.-Dec. 2005.

[51] K. Constantinides, O. Mutlu, T. Austin, and V. Bertacco, "Software-Based Online Detection of Hardware Defects: Mechanisms, Architectural Support, and Evaluation," *Proc. IEEE/ACM Int'l Symp. Microarchitecture (MICRO '07)*, pp. 97-108, 2007.

[52] Y. Li, and S. Mitra, "VAST: Virtualization-Assisted Concurrent Autonomous Self-Test," *Proc. IEEE Int'l Test Conf. (ITC '08)*, pp. 1-10, 2008.

[53] D. Lampret, *OpenRISC 1200 IP Core Specification*, tech. report, OpenCores, 2001.

[54] B. Broeg, B. Bose, Y. Kwon, and Y. Ashir, "Lee Distance and Topological Properties of k-ary n-cubes," *IEEE Trans. Computers*, vol. 44, no. 8, pp. 1021-1030, Aug. 1995.

[55] B. Broeg, B. Bose, and V. Lo, "Lee Distance, Gray Codes, and the Torus," *Telecommunication Systems*, vol. 10, nos. 1-2, pp. 21-32, Jan. 1998.

[56] A. Jiang, M. Cook, and J. Bruck, "Optimal Interleaving on Tori," *SIAM Journal of Discrete Math*, vol. 20, no. 4, pp. 841-879, Dec. 2006.

[57] C. Martinez, E. Vallejo, R. Beivide, C. Izu, and M. Moreto, "Dense Gaussian Networks: Suitable Topologies for On-Chip Multiprocessors," *Int'l Journal of Parallel Programming*, vol. 34, no. 3, pp. 193-211, June 2006.

[58] Y. Jin, E. Kim, and K. Yum, "Peak Power Control for a QoS Capable On-Chip Network," *Proc. Int'l Conf. Parallel Processing (ICPP '05)*, pp. 585-592, 2005.

[59] L. Shang, L.S. Peh, and N.K. Jha, "PowerHerd: A Distributed Scheme for Dynamically Satisfying Peak-Power Constraints in Interconnection Networks," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 1, pp. 92-110, Jan. 2006.

[60] A. Banerjee, R. Mullins, and S. Moore, "A Power and Energy Exploration of Network-on-Chip Architectures," *Proc. IEEE Symp. Networks-on-Chip (NOCS '07)*, pp. 163-172, 2007.

[61] N. Chang, K. Kim, and H.G. Lee, "Cycle-Accurate Energy Consumption Measurement and Analysis: Case Study of ARM7TDMI," *Proc. Int'l Symp. Low Power Electronics and Design (ISLPED '00)*, pp. 195-190, 2000.

[62] Stanford Concurrent VLSI Architecture Group, "Open Source Network-on-Chip Router RTL, http://nocs.stanford.edu/router.html, June 2010.

[63] Synopsys, "Synopsys 90nm Generic Library," http://www.synopsys.com/Community/UniversityProgram/Pages/Library.aspx, June 2010.

[64] Synopsys, "Synopsys Textramax ATPG," http://www.synopsys.com/Tools/Implementation/RTLSynthesis/Pages/TetraMAXATPG.aspx, June 2010.

[65] C. Spitzer, ed., *Avionics: Development and Implementation*, CRC Press, 2007.

[66] Freescale Semiconductor, "MPC564xL: 32-Bit Power Architecture Microcontrollers for Chassis and Safety Applications," http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=MPC564xL, June 2010.

[67] M.R. Guthaus, J.S. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R.B. Brown, "MiBench: A Free, Commercially Representative Embedded Benchmark Suite," *Proc. IEEE 4th Workshop Workload Characterization (IISWC '01)*, pp. 3-14, 2001.

[68] J. Stine, I. Castellanos, M. Wood, J. Henson, F. Love, W. Davis, P. Franzon, M. Bucher, S. Basavarajaiah, J. Oh, and R. Jenkal, "FreePDK: An Open-Source Variation-Aware Design Kit," *Proc. Int'l Conf. Microelectronic Systems Education*, pp. 173-174, 2007.

VITA

Jason Daniel Lee was born in Thibodaux, Louisiana, USA. He attended the Louisiana School for Math, Science, and the Arts (LSMSA), a state supported residential high school for gifted and talented students. Upon graduation in 2000, he was selected by the faculty as the top student in math and computer science for his class. He then received his Bachelor of Science degree in computer engineering from Texas A&M University in 2004, Magna Cum Laude. He was invited to return to the Department of Computer Science and Engineering for graduate work and completed the Doctor of Philosophy in computer engineering in 2010. His research interests include fault-tolerant systems, network-on-chip (NoC) and many-core architecture, electronic design automation and testing.

Jason Lee may be reached at Department of Computer Science and Engineering, Texas A&M University, TAMU 3112, College Station, TX 77843. His email is jdlee@tamu.edu.