Concurrent Autonomous Self-Test for Uncore Components in System-on-Chips

Yanjing Li, Stanford University
Onur Mutlu, Carnegie Mellon University
Donald S. Gardner, Intel Corporation
Subhasish Mitra, Stanford University

Abstract

Concurrent autonomous self-test, or online self-test, allows a system to test itself, concurrently during normal operation, with no system downtime visible to the end-user. Online self-test is important for overcoming major reliability challenges such as early-life failures and circuit aging in future System-on-Chips (SoCs). To ensure required levels of overall reliability of SoCs, it is essential to apply online self-test to uncore components, e.g., cache controllers, DRAM controllers, and I/O controllers, in addition to processor cores. This is because uncore components can account for a significant portion of the overall logic area of a multi-core SoC. In this paper, we present an efficient online self-test technique for uncore components in SoCs. We achieve extremely high test coverage by storing high-quality test patterns in off-chip non-volatile storage. However, a simple technique that stalls the uncore-component-under-test can result in significant system performance degradation or even visible system unresponsiveness. Our new techniques overcome these challenges and enable cost-effective online self-test of uncore components through three special hardware features: 1. resource reallocation and sharing (RRS); 2. no-performance-impact testing; and, 3. smart backups. Implementation of online self-test for uncore components of the open-source OpenSPARC T2 multi-core SoC, using a combination of these three techniques, achieves high test coverage at < 1% area impact, < 1% power impact, and < 3% system-level performance impact. These results demonstrate the effectiveness and practicality of our techniques.

1. Introduction

In many System-on-Chips (*SoC*s), uncore components account for a significant portion of the total chip area. *Uncore components* are defined as system components that are neither processor cores nor co-processors (e.g., graphical processing units), which are collectively referred to as *computation engines* [Azimi 07]. For the open-source OpenSPARC T2 SoC [SUN 09] with 8 processor cores supporting 64 hardware threads, uncore components include on-chip memories (occupying 76% of the total chip area), cache controllers, DRAM controllers, I/O controllers, and crossbar. In OpenSPARC T2, the non-memory uncore components account for 51% of the total non-memory chip area. The other 49% is occupied by processor cores (Fig. 1).

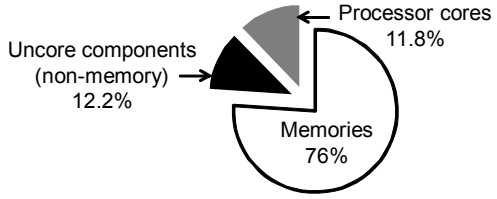


Figure 1. Area breakdown of the OpenSPARC T2 SoC.

At advanced technology nodes, hardware failure mechanisms such as transistor aging and early-life failures (also referred to as infant mortality) have been identified as major reliability challenges for future SoCs [Agostinelli 05, Borkar 05, 07, Van Horn 05]. *Concurrent autonomous self-test*, also referred to as *online self-test*, allows a system to test itself, concurrently during normal operation, without any visible downtime to the end user. Very thorough online self-test is essential to overcome the above reliability challenges for robust SoCs of the future [Borkar 07]. Processor cores have been the focus of several previous online self-test techniques (details in Sec. 5). In contrast, this paper presents efficient online self-test techniques for uncore components of SoCs. Since there has been a large amount of research on resilience techniques for on-chip memories (e.g., error correcting codes (ECC), transparent memory BIST, scrubbing, and sparing), this paper focuses on uncore logic blocks such as cache controllers, DRAM controllers, and I/O controllers.

There are two aspects of online self-test for uncore components:

1. Test coverage: High test coverage can be achieved by storing very thorough compressed test patterns in off-chip non-volatile storage (e.g., [Li 08]). In this paper, we use *CASP*, an acronym for Concurrent Autonomous chip self-test using Stored test Patterns [Li 08], because it can achieve high test coverage at low cost. Traditional pseudo-random Logic BIST may also be used but it may be expensive.

2. Orchestration of online self-test: Architectural and system-level techniques are necessary to minimize area, power, and visible system-level performance impact.

Techniques such as CASP and scan-based pseudo-random Logic BIST impose the following restrictions:

1. Use of scan chains for high test coverage prevents simultaneous execution of online self-test and normal system operation on the same piece of hardware.

2. Since online self-test is directly related to overall system reliability, it may not be optional, and may have higher priority than all other operations (e.g., operating system and user tasks).

With these restrictions, a naïve implementation of CASP for uncore components is to stall the operations of the uncore-component-under-test. However, this simple technique is inadequate, because although it is possible to utilize idle intervals for online self-test, there is no guarantee that critical operations (e.g., interrupt handling) will not be initiated during online self-test. As a result, various undesirable situations can occur. For example, while a DRAM controller is undergoing online self-test, a timer interrupt handler in the operating system requires access to the main memory to fetch instructions in the interrupt handling routine. Consider the case in which the DRAM controller is stalled due to online self-test. We emulated this situation for two scenarios of online self-test applied to the DRAM controller on an actual quad-core Intel® Core™ i7 system:

1. Infrequent online self-test that runs for 1 sec. every 10 minutes.

2. Frequent online self-test that runs for 1 sec. every 10 sec.

The infrequent online self-test results in visible system unresponsiveness of up to 2 minutes every 10 minutes. For frequent online self-test, the system completely hangs and becomes unusable.

We also simulated the case of a frequent online self-test that is applied to cache controllers for 1 sec. every 10 sec. If the cache controllers are stalled for the entire duration of online self-test, significant performance overhead is introduced for the PARSEC benchmark suite: 4%-59% (Sec. 4.3). While functional testing of uncore components may reduce the severity of some of these situations, such functional testing of uncore components with **quantified** high test coverage may be difficult [Sengupta 99, Tumin 01].

The major contributions of this paper are:

- New cost-effective online self-test techniques for uncore components in multi-core SoCs.
- Detailed implementation of our techniques in the open-source OpenSPARC T2 multi-core SoC.
- System-level evaluation demonstrating the effectiveness and practicality of our techniques: high test coverage (> 99.2% stuck-at and > 92.8% transition) at minimal system-level impact (< 1% area impact, < 1% power impact, and < 3% performance impact) with a frequent online self-test that is invoked for 1 sec. every 10 sec.

Section 2 presents an overview of CASP, which forms the basis for online self-test for uncore components in SoCs. Our online self-test techniques for uncore components, together with a detailed case study on the 8-core, 64-thread OpenSPARC T2 multi-core SoC, are presented in Sec. 3. In Sec. 4, we present evaluation results: test time, test storage, test coverage, as well as system-level area, power, and performance impact. We discuss related work in Sec. 5, followed by conclusions.

2. CASP Overview

CASP stands for Concurrent Autonomous chip self-test using Stored test Patterns [Li 08]. The basic ideas of CASP are:

- Storage of thorough test patterns (both functional and structural) in **off-chip** non-volatile storage (e.g., FLASH memory) for high test coverage. These patterns include stuck-at, transition, and various delay tests. The patterns can include production tests as well as tests that may not be applied during production due to test cost reasons (e.g., special patterns to assist circuit failure prediction for aging [Baba 09]). Test compression techniques (e.g., X-Compact [Mitra 04]) can be used to reduce storage overhead. Moreover, test patterns may be updated (e.g., using patches) according to application requirements and failure characteristics after a system is deployed in the field. Note that, CASP is different from logic BIST that uses on-chip stored test patterns [Agarwal 81]. CASP utilizes **off-chip** non-volatile storage which is practical because of the wide availability of high-density and low-cost non-volatile storage such as FLASH memory [Lawton 06].
- Architectural and system-level support for CASP with a wide range of system-level power, performance, and area cost tradeoffs. For **processor cores**, we isolate the core-under-test from the rest of the system and reallocate its workload appropriately during online self-test (e.g., using virtualization support [Inoue 08]).

CASP is characterized by two parameters for each component in the system. The first is *test latency*, which refers to the duration of online self-test test (including time to fetch test patterns from off-chip storage and test application time). The second is *test period*, which specifies the time interval between two consecutive online self-test sessions, determined by target reliability mechanisms. For circuit failure prediction targeting aging, CASP may be invoked once a day. CASP for hard failure detection or circuit failure prediction targeting early-life failures may be invoked much more frequently.

One essential component of CASP is the *CASP controller* which is used to instrument online self-test in the following four phases:

Phase 1: Scheduling. The CASP controller selects one or more core or uncore components to undergo online self-test. Scheduling must satisfy test period requirements of each component, and can take into account system power and performance tradeoffs.

Phase 2: Pre-processing. The CASP controller producess appropriate control signals to prepare the selected component for online self-test (special techniques are utilized for uncore components in this phase, details in Sec. 3). The component will be temporarily unavailable for normal operation and will be isolated from the rest of the system.

Phase 3: Test application. The CASP controller fetches test patterns from off-chip storage, applies these patterns to the component-under-test (using scan chains), and analyzes test responses.

Phase 4: Post-processing. The CASP controller brings the component from test mode back to normal operation mode, if the online-self-test passes. Otherwise, appropriate self-healing or recovery actions are invoked.

The CASP controller is implemented as a finite state machine in hardware with very small area cost (< 0.01% as shown in Sec. 4). For systems with control units that collect comprehensive system information (temperature, current, and power) for power management purposes, e.g., Power Controller Unit (*PCU*) in Intel® Core™ i7 system [Gunther 08], the CASP controller may be integrated into such control units. This integration allows not only lower costs, but also exposure of online self-test requirements to the control units, enabling online self-test scheduling that optimizes power and performance tradeoffs.

3. CASP for Uncore Components in a Multi-core SoC

As illustrated in Sec. 1, a simple implementation of CASP for uncore components, which stalls the operations of the uncore-component-under-test, is inadequate. This is because visible performance impact or system unresponsiveness can be introduced. A naïve technique, which includes an additional identical "backup" component for a set of uncore components with similar functionality, can eliminate performance impact and system responsiveness, but at high area costs, e.g., > 10% (Sec. 4). Therefore, special features are necessary to optimize area, power, and performance tradeoffs.

We present three new techniques to support CASP for uncore components: 1. resource reallocation and sharing (RRS), 2. no-performance-impact testing, and, 3. smart backups. We use the OpenSPARC T2 multi-core SoC for our case study. OpenSPARC T2 is an open-source multi-core processor that closely resembles the commercial SUN Niagara 2 system [SUN 09]. It provides us access to the RTL for design modifications and analysis. Our techniques are generally applicable for uncore components supporting on-chip communications and memory accesses that are dominant in many SoCs [Lambrechts 05, Shiue 01].

3.1 Categorization of Uncore Components

Uncore components can be either digital or analog. We focus on digital components because they account for 90% of the total area in many SoCs [Negreiros 02]. For analog and mixed-signal circuits, such as PLLs, power amplifiers, SerDes, and RF circuitry, new resilience techniques may be necessary.

Digital uncore components can be categorized into two types: performance-critical and non-performance-critical. The *performance-critical* category includes all uncore components that directly affect system performance. Performance-critical uncore components in OpenSPARC T2 include:

1. Memory subsystem: cache memories, cache controllers, and DRAM controllers.
2. I/O subsystem: I/O controllers that are used to interface with PCI express and the Ethernet network.
3. System communication mechanism: crossbar.

The *non-performance-critical* category includes components with large and frequent idle intervals during which online self-test can be applied. In OpenSPARC T2, the test control unit with JTAG support and the debug unit providing post silicon debug features belong to this category. Since CASP can be applied to non-performance-critical uncore components by utilizing their idle intervals, we focus on online self-test support for performance-critical uncore components.

3.2 The Resource Reallocation and Sharing (RRS) Technique

The *resource reallocation and sharing (RRS)* technique takes advantage of the existence of multiple instances of uncore components that can accomplish "similar" functionality. As a result, during online self-test of an instance of an uncore component, workload of the uncore-component-under-test can be reallocated to another component that can accomplish similar functionality, referred to as the *helper*. The helper is able to process the workload belonging to the uncore-component-under-test in addition to its own workload (unlike redundancy techniques using spares).

For example, one L2 cache bank can allocate entries for and respond to requests (from processor cores) to the data mapped to the bank-under-test. Additional hardware may be necessary to implement this technique. For example, for L2 cache banks, additional logic is provided to distinguish the cache blocks that are originally mapped to the bank-under-test (i.e., blocks that are reallocated) from those that are actually mapped to the helper. Since the helper is shared among two sets of workloads, system performance degradation may occur during online self-test (Sec. 4).

In OpenSPARC T2, we applied the RRS technique to the cache banks (cache memories and controllers), DRAM controllers (*MCUs*), and the DMA channels in the network interface unit. With RRS, the pre-processing phase of CASP consists of the following steps (all control signals are outputs from the CASP controller):

1. Stall the uncore-component-under-test so that it does not accept new requests.
2. Drain outstanding requests in the uncore-component-under-test.
3. Transfer "*necessary states*" from the uncore-component-under-test to the helper. The necessary states are required for the helper to correctly process the requests that are reallocated from the uncore-components-under-test. Examples of necessary states include dirty blocks in L2 cache banks and refresh intervals in DRAM controllers.
4. Reallocate the workload from the uncore-component-under-test to the helper.

With RRS, the post-processing phase of CASP is similar to pre-processing, except that the roles of the uncore-component-under-test and the helper are swapped. Figure 2 shows an example of the pre-processing step of CASP with RRS for L2 cache banks.

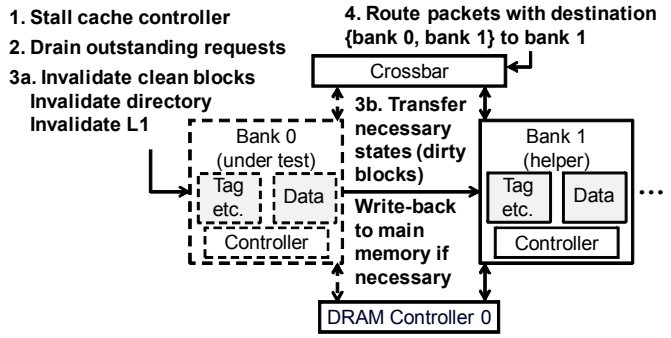


Figure 2. Pre-processing phase of CASP with RRS for L2 cache banks in OpenSPARC T2. Dashed lines and arrows indicate components and signals not used during online self-test.

3.3 No-Performance-Impact Testing

Certain uncore components in a system can undergo online self-test along with other components at no additional performance impact. This technique is called *no-performance-impact testing*. In OpenSPARC T2, CASP for the crossbar and the private L1 caches can be implemented using this technique. The crossbar consists of independent multiplexers and arbitration logic to route packets from multiple sources to a destination (e.g., a processor core or an L2 cache bank). Each independent multiplexer and arbitration logic can have independent scan chains (with 3,664 flip-flops) so that they can be tested in series (Fig. 3). When a destination (e.g., a core or an L2 bank) is undergoing online self-test, packets for the destination-under-test are reallocated (using RRS for L2 banks and virtualization techniques [Inoue 08] for cores), and the corresponding multiplexer and arbitration logic in the crossbar would be idle. Therefore, we can test the corresponding multiplexer and arbitration logic at the same time that the destination is being tested. L1 private caches can be tested at the same time that processor cores are being tested. We invalidate all entries in the L1 caches undergoing online self-test to avoid possible loss of cache coherence packets generated by the L2 cache banks.

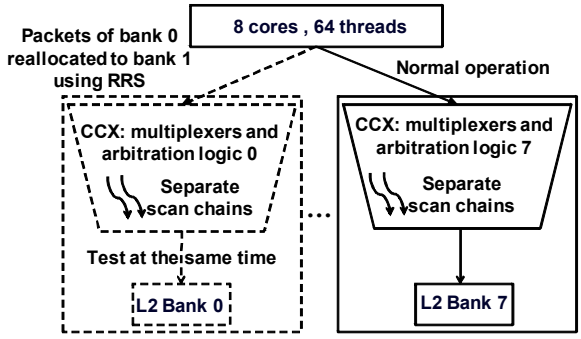


Figure 3. No-performance-impact testing for crossbar (CCX) in OpenSPARC T2. Dashed lines and arrows indicate components and signals not used during online self-test.

3.4 Smart Backups

In the smart backup technique, for a set of uncore components with similar functionality, we include an additional component called the *smart backup*, which processes all incoming requests in the place of the original(s) during online self-test. This is different from duplication used in traditional concurrent error detection for the following reasons: 1. One backup can be shared by multiple uncore components that have similar functionality. 2. The power penalty is significantly smaller than duplication, because a backup is not required to operate concurrently with the original(s) and can be turned off whenever the original(s) is not undergoing online self-test. 3. The smart backup does not necessarily have to be identical to the original(s), unlike *identical backup*. It can be less complex than the original(s) at the price of performance degradation that occurs only during online self-test, as long as such degradation is acceptable at the system level.

The main principle behind the smart backup technique is to maintain a fine balance between area cost and performance impact. On the one hand, we do not want to include all the complexity of the original(s) in the backup because the area costs can be too high for identical backup (e.g., > 10%, shown in Sec. 4). On the other hand, if the backup is too simple, it may not be able to efficiently process critical operations (e.g., interrupt handling), which can result in significant performance impact or visible system unresponsiveness (demonstrated in Sec. 1). We need to be smart about the implementation of the backup, which only provides minimally necessary functionality to process critical operations (i.e., operations that cannot sustain long latencies), thereby keeping both area and performance costs within acceptable limits.

In OpenSPARC T2, we demonstrate the smart backup technique for the I/O subsystem, which consists of four modules: non-cachable unit (*NCU*), system interface unit (*SIU*), PCI Express interface unit (*PIU*), and network interface unit (*NIU*). Table 1 describes the operations, performed by each of these modules, which can be divided into critical and non-critical operations. All critical operations that cannot sustain long delays are summarized into the following three categories:

1. Interrupt processing, whose latency is usually within tens of microseconds;
2. Programmed I/O (*PIO*), which is crucial for interfacing devices such as keyboard and mouse;
3. 10Gigabit Ethernet packet processing, which, if stalled, may cause up to thousands of packets to be dropped.

All other I/O operations, including DMA requests to hard disks and accesses to configuration registers, can tolerate long latencies, and thus can be stalled for the entire duration of online self-test. Note that, stalling these I/O operations does not stall processor cores, because the operating system would switch out the tasks waiting for I/O so that other tasks can be executed [Li 09]. Our smart backups for the I/O subsystem in OpenSPARC T2 only support the three types of critical I/O operations described above.

Table 1. Critical and non-critical operations performed by the I/O subsystem in OpenSPARC T2. Only the critical operations are supported by smart backups. PIO refers to Programmed I/O.

	Operations of original components	
	Critical operations (☑ supported by smart backups)	Non-critical operations (☒ not supported by smart backups)
NCU	Process interrupts; transfer PIO packets between PIU and CCX.	Configuration register access; Interface with boot ROM.
SIU	Transfer DMA requests between NIU and L2 banks.	Transfer DMA requests between PIU and L2 banks.
PIU	Transfer PIO and interrupt packets between I/O devices and NCU.	Transfer DMA packets between I/O devices and SIU.
NIU*	Interface with physical Ethernet ports; Layer 2 receive packet decode, checksum, and classification.	Layers 3 and 4 receive packet classification (for smart backup, it is performed in software instead).

*DMA channels in the NIU are not included in this table. They use the RRS technique as discussed in Sec. 3.2.

To implement the smart backup technique, we apply the same inputs into the original(s) and the backup, and select one set of outputs from the original(s) or the backup, with the select signals controlled by the CASP controller. Pre-processing consists of the following steps (all control signals are produced by the CASP controller):

1. Stall the uncore-component-under-test so that it does not accept new requests.
2. Drain outstanding requests in the uncore-component-under-test.
3. Turn on and reset the backup.
4. Transfer necessary states from the uncore-component-under-test to the backup. Similar to RRS, the necessary states are required for the backup to operate correctly.
5. Select outputs from the backup.

Post-processing is similar to pre-processing, except that the roles of the uncore-component-under-test and the backup are swapped. The implementation of pre-processing of CASP with the smart backup technique is shown in Fig. 4, using the NCU as an example.

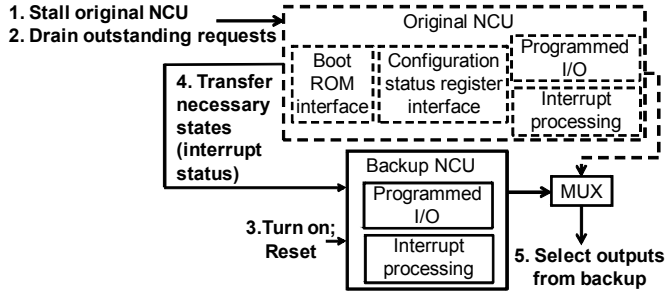


Figure 4. Pre-processing phase of CASP with the smart backup technique for NCU. Dashed lines and arrows indicate components and signals not used during online self-test.

4. Evaluation

4.1 Test Coverage, Test Storage, and Test Time

We obtained test patterns for the uncore components in OpenSPARC T2 using the ATPG tool *TetraMax* from Synopsys. Test coverage, test storage, and test time results are shown in Table 2. We use the reference clock (166 MHz) in OpenSPARC T2 as the test clock for shifting test patterns. The test clock is multiplexed with the system or I/O clock, controlled by the CASP controller. Advanced timing tests (e.g., Cadence True-Time) require more advanced multiple-speed structural test support (e.g., [Iyengar 06]). We also assume 10X test compression, which reduces test storage and test transfer time by 10X.

Table 2. Test results for uncore components in OpenSPARC T2. (L2: an L2 cache controller; MCU: a DRAM controller; CCX: a set of independent multiplexer and arbitration logic in crossbar.)

a. Test coverage and pattern count.

	Stuck-at		Transition	
	Test Coverage	Pattern count	Test Coverage	Pattern Count
L2	99.9%	211	92.8%	613
MCU	99.2%	1,833	94.1%	3,334
CCX	99.9%	91	97.0%	204
NCU	99.8%	144	97.2%	443
SIU	99.6%	681	97.8%	1,250
PIU	99.9%	1,798	95.0%	2,985
NIU	99.5%	819	94.1%	2,220
Overall	> 99.2%	5,577	> 92.8%	11,049

b. Test storage and test time (10X test compression assumed).

	Storage	Transfer time	Test application time	Overall Test Time
L2	0.32MB	6 ms	23 ms	29 ms
MCU	2.5MB	47 ms	140 ms	187 ms
CCX	0.03MB	0.5 ms	6 ms	7 ms
NCU	0.4MB	7 ms	16 ms	23 ms
SIU	1.1MB	20 ms	52 ms	72 ms
PIU	4.9MB	92 ms	104 ms	196 ms
NIU	7.1MB	135 ms	148 ms	283 ms
Overall	16.35MB	-	-	-

The same test patterns are used for all identical instances of L2 cache controllers, DRAM controllers (MCUs), and individual multiplexer and arbitration logic in the crossbar, resulting in 16.35MB storage overhead (Table 2). (However, this is not a limitation of the presented techniques and separate test pattern sets may be generated even for instances with identical functionality). If we store patterns separately for each instance, the total storage overhead becomes 27MB. If Cadence True-Time test patterns are included, we estimate a 5X pattern count increase (projected from test patterns generated for processor cores in OpenSPARC T1 [Li 08]), resulting in a 5X increase in storage and test time. However, even with such an increase, off-chip storage of 162MB for uncore components is reasonable for the FLASH memory capacity available today.

4.2 Area and Power Impact

We used *Design Compiler* and *Power Compiler* (using a DesignWare 90nm library) from Synopsys to quantify area and power costs of our techniques. RRS is used to implement CASP for the L2 cache banks, the DRAM controllers (MCUs), and the DMA channels in NIU. The smart backup technique is applied to the I/O subsystem. As a comparison point, identical backups for these uncore components are also implemented. To compute the power impact, we assume that the backups are turned on for 10% of the time (corresponding to online self-test with test latency = 1 sec. and test period = 10 sec.). Table 3 summarizes the results. For the no-performance-impact testing technique, minimal hardware modification is required to respond to the control signals from the CASP controller, and each uncore-component-under-test must be properly isolated from the rest of the system.

Table 3. Area and power impact of CASP for uncore components in OpenSPARC T2: naïve identical backup technique vs. our techniques. I/O refers to the I/O subsystem.

a. Identical backups.

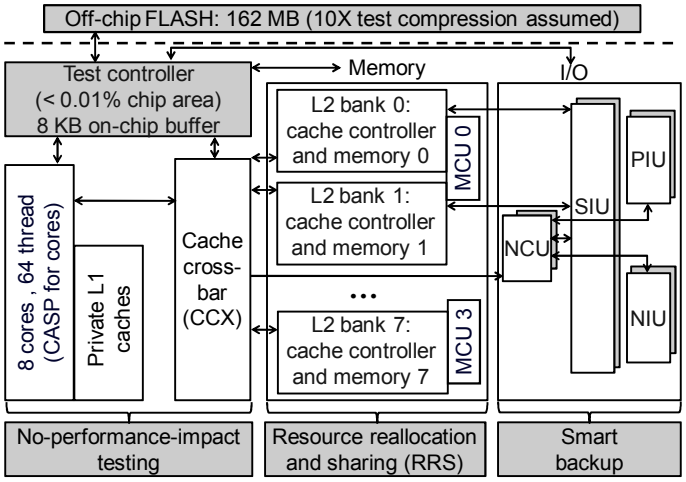
	I/O	MCU	L2 bank	Overall
Area Impact	3.25%	0.23%	8.89%	12.37%
Power Impact	0.42%	0.05%	0.64%	1.11%

b. Smart backups for I/O, RRS for MCU and L2 bank.

	I/O	MCU	L2 bank	Overall
Area Impact	0.38%	0.13%	0.41%	**0.92%**
Power Impact	0.11%	0.19%	0.43%	**0.73%**

The naïve identical backup technique has chip area impact of 12.37%. With our techniques (RRS, no-performance-impact testing, and smart backups), area impact is significantly reduced to < 1%. For RRS and smart backups, performance degradation can occur during online self-test (results in Sec. 4.3). The smart/identical backup technique can sometimes be more advantageous than RRS because: 1. It may impose less power impact (e.g., for MCUs). While the backup only needs to be turned on during online self-test, additional logic supporting RRS can be intrusive to the original implementation. 2. The area impact of backup and RRS may be comparable (e.g., for MCUs), because while one backup can be shared by all identical components, design modifications to RRS are applied to all components. 3. The hardware modification for backups can be less intrusive than RRS.

CASP support for uncore components in OpenSPARC T2 is summarized in Fig. 5.



Overall Area Impact: <1%; Overall Power Impact: < 1%

Figure 5. OpenSPARC T2 with CASP support for uncore components.

In our implementation, the area of the CASP controller is < 0.01% of the total chip area. An on-chip buffer is used to store one set of test patterns and the corresponding masks and expected responses, which results in a capacity of 8KB. This number is obtained by assuming that all components (core and uncore) are tested one at a time in a round-robin fashion (except for no-performance-cost testing, where multiple

components are tested at the same time). The required on-chip buffer capacity is the greatest when no-performance-impact testing is used to test the crossbar, the private L1 cache, and the processor core at the same time. The overall area and power costs are both < 1%.

4.3 Performance Impact of RRS and Smart Backup Techniques

RRS for L2 cache banks: To evaluate the performance impact of RRS for the L2 cache banks (controllers and memories), we used the *GEMS simulator* [Martin 05], which was modified to model RRS. We simulated the PARSEC benchmark suite [Bienia 08] for a pessimistic scenario with frequent online self-test that runs for 1 sec. every 10 sec. Figure 6 shows that the execution time overhead of RRS is minimal: < 1.4% across the entire benchmark suite. For comparison purposes, we also simulated a simple technique, *stall-and-test*, which stalls the cache controller, and all blocks mapped to the bank-under-test are fetched directly from the main memory during online self-test. The simple stall-and-test technique introduces significant overheads ranging from 4%-59%, which demonstrates that this simple stall-and-test technique may not be desirable for uncore components.

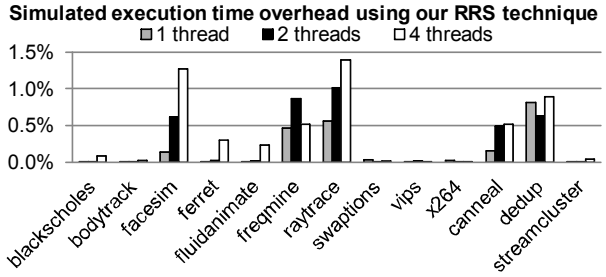


Figure 6. RRS for L2 cache banks: execution time overhead of the PARSEC benchmark suite (< 1.4%). Execution time overhead with a simple stall-and-test technique: 4%- 59%.

RRS for MCUs: We evaluated the performance impact of RRS for the DRAM controllers (MCUs) using the same simulation platform with the same test parameters as described above. We obtained the total number of main memory accesses (with a worst-case delay of 2,000 cycles [SUN 09]) that result in resource contentions in the MCUs during online self-test. For each such access, we assume that the worst case delay (2,000 cycles) is seen by the processor core waiting for the data being fetched, which results in performance impact of < 0.01% for each core across the entire benchmark. When taking into account the interactions of multi-threaded programs, this impact may increase by up to 6X due to inter-thread synchronization overhead (for an application with up to 4 threads) [Li 09], but would still result in < 0.1% performance impact at the system level.

RRS for DMA channels in NIU: There are 16 independent DMA channels in both the receiving and transmitting directions in NIU. DMA channels are tested in series, and the packets that are mapped to the DMA-channel-under-test are reallocated to another DMA channel. In the worst case, the bandwidth can be degraded by up to 6.25% [SUN 07] during online self-test. However, if all 16 DMA channels are not utilized, there may not be any performance impact.

Smart backups for PIU and SIU: To evaluate the performance impact of smart backups for PIU and SIU, we used the Bonnie file system benchmark [Bonnie 09], which generates continuous write requests to the hard disk. To emulate the lack of DMA support in the smart backups for PIU and SIU, a delay of 250ms (approximately the sum of test time of PIU and SIU) is inserted in the benchmark program every 10 sec. (we assume that the PIO support available in the smart backups is not used to process the write requests to the hard disk). The measured performance overhead on an actual Intel® Core™ i7 system is 2.7%. Such performance overhead is small because smart backups are used only during online self-test. If the smart backups were used in place of the original components during normal operation (all the time), all I/O device accesses would need to be processed using PIO instead of DMA. In that case, the throughput may degrade by 50% and CPU utilization may saturate [Ramakrishnan 93] (which may not be desirable).

Smart backup for NIU (excluding DMA channels): The smart backup for NIU does not perform classification of packets in layers 3 and 4. Instead, the classification is performed in the software network stack, which increases processing latency by 200ns (~3%) for a single TCP/IP packet as measured on an actual Intel® Core™ i7 system.

Smart backup for NCU: The smart backup for NCU supports all the critical operations (interrupt and PIO processing). Configuration status register accesses and interfacing with boot ROM can tolerate long latencies [SUN 09].

5. Related Work

Existing online self-test techniques include various types of logic Built-In Self-Test (BIST) such as pseudo-random BIST and native-mode testing [Bardell 87, Parvathala 02, Shen 98], input-vector monitoring concurrent BIST [Saluja 88], roving emulation [Breuer 86], periodic functional testing [Corno 96, Krstic 02, Baldini 05], access-control-extension (ACE) [Constantinides 07], and Concurrent Autonomous chip self-test using Stored test Patterns (CASP) targeting processor cores [Inoue 08, Li 08]. Several of these existing online self-test techniques may be applied to uncore components. However, without appropriate architectural and system-level support, applying some of these online self-test techniques would require the uncore-components-under-test to be stalled, which incurs significant performance degradation (Sec. 4). Furthermore, most existing online self-test techniques exhibit one or more of the following limitations: low test coverage, high area costs, and high design complexity. Logic BIST assisted by techniques such as test points, mapping logic, and reseeding ([Al-Yamani 03, Nakao 99, Touba 95]) to enhance test coverage can be expensive. Functional BIST techniques generally have low test coverage. For roving emulation, test coverage depends on the workload executed during the checking window and cannot be guaranteed *a priori*. Input-vector monitoring concurrent BIST (C-BIST) does not require stalling, thereby avoiding the associated performance impact. However, the test coverage of C-BIST also depends on applications (similar to roving emulation). Moreover, there are several open questions about specific implementation details of input monitoring such as which signals to monitor, which sequences to monitor (any test pattern appearing in input patterns "out of sequence" may be ignored by input monitoring), and the associated routing costs. CASP overcomes these limitations by utilizing stored patterns in off-chip non-volatile memory. In particular, CASP for uncore components, described in this paper, overcomes limitations of stall-and-test while achieving high test coverage. Table 4 qualitatively compares and contrasts various online self-test techniques.

Error correcting codes (ECC), scrubbing, and sparing for memories, and redundant channels for interconnects and buses, are already used in commercial systems [Shah 07]. Various memory built-in self-test are also available [Nicolaidis 96]. Techniques presented in this paper are compatible with such existing resilience techniques for memories.

6. Conclusions

Uncore components account for a significant portion of the overall non-memory area and power in SoCs. Hence, error resilience techniques for future SoCs must focus on uncore components in addition to cores. Online self-test and diagnostics constitute an essential component of error resilience solutions for future SoCs. Hence, efficient techniques for online self-test and diagnostics of uncore components are essential. The new techniques presented in this paper enable effective use of CASP with high test coverage for uncore components in SoCs with very little system-level power, performance, and area impact. This is made possible by utilizing several characteristics of SoC designs that enable efficient resource reallocation and sharing, no-performance-impact testing, and smart-backups. Future research directions include: 1. extensions of our techniques for a wider variety of SOCs, 2. development of adaptive test scheduling for both core and uncore components to optimize system power and performance tradeoffs, 3. integration of software techniques to assist CASP for uncore components.

Acknowledgement

This research is supported in part by FCRP GSRC and C2S2, and NSF. We thank Samy Makar of Apple Inc., and Ted Hong, Michael Lentine, Sung-Boem Park, and Yifan YangGong of Stanford University.

Table 4. Qualitative comparison of existing online self-test techniques.

	LBIST	Functional testing	Roving emulation	C-BIST	ACE	CASP
Applicability	Core and uncore	Core and uncore	Core and uncore	Core and uncore	Cores only	Core [Li 08] and uncore (this paper)
Test Coverage	High with high costs	Moderate to low for cores; not easily quantified for uncore components	Depends on application	Depends on application	High	High
Area Cost	High for high coverage	Low	Low	Low	Moderate	Low (<1%)
Design complexity	High for high coverage	Low	Moderate	Moderate	Moderate	Moderate

References

[Agarwal 81] Agarwal, V.K., and E. Cerny, "Store and Generate Built-In Testing Approach," *Proc. Intl. Symp. Fault Tolerant Comp.,* pp. 35-40, 1981.

[Agostinelli 05] Agostinelli, M., *et al.*, "Random Charge Effects for PMOS NBTI in Ultra-Small Gate Area Devices," *Proc. Intl. Reliability Physics Symp.*, pp. 529–532, 2005.

[Al-Yamani 03] Al-Yamani, A., S. Mitra, and E.J. McCluskey, "BIST Reseeding with Very Few Seeds," *Proc. VLSI Test Symp.*, pp. 132-138, 2003.

[Azimi 07] Azimi, M., *et al.,* "Package Technology to Address the Memory Bandwidth Challenge for Tera-Scale Computing," *Intel Technology Journal,* pp. 197-206, 2007.

[Baba 09] Baba, H., and S. Mitra, "Testing for Transistor Aging," *Proc. VLSI Test Symp.,* pp. 215-220, 2009.

[Baldini 05] Baldini, A., A. Benso, and P. Prinetto, "A Dependable Autonomic Computing Environment for Self-Testing of Complex Heterogeneous Systems," *Electronic Notes on Theoretical Computer Science*, pp. 45-57, 2005.

[Bardell 87] Bardell, P.H., W.H. McAnney, and J. Savir, "Built-In Test for VLSI: Pseudorandom Techniques," *Wiley*, 1987.

[Bienia 08] Bienia, C., *et al.*, "The PARSEC Benchmark Suite: Characterization and Architectural Implications," *Proc. Intl. Conf. on Parallel Arch. and Compilation Technologies,* pp. 72-81, 2008.

[Bonnie 09] "Bonnie," http://www.textuality.com/bonnie/.

[Borkar 05] Borkar, S., "Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation," *IEEE Micro*, pp. 10-16, 2005.

[Borkar 07] Borkar, S., "Thousand Core Chips – A Technology Perspective," *Proc. Design Automation Conf.*, pp. 746-749, 2007.

[Breuer 86] Breuer, M., and A. Ismaeel, "Roving Emulation as a Fault Detection Mechanism," *IEEE Trans. Comp.,* pp. 933-939, 1986.

[Constantinides 07] Constantinides, K., *et al.*, "Software-Based On-line Detection of Hardware Defects: Mechanisms, Architectural Support, and Evaluation," *Proc. Intl. Symp. on Microarchitecture,* pp. 97-108, 2007.

[Corno 96] Corno, M., *et al*, "On-line Testing of an Off-the-shelf Microprocessor Board for Safety-critical Applications," *Proc. European Dependable Comp. Conf.,* pp. 190-202, 1996.

[Gunther 08] Gunther, S., and R. Singhal, "Next Generation Intel Microarchitecture (Nehalem) Family: Architectural Insights and Power Mangement," *Intel Developer Forum,* 2008.

[Inoue 08] Inoue, H., Y. Li, and S. Mitra, "VAST: Virtualization-Assisted Concurrent Autonomous Self-Test," *Proc. Intl. Test Conf.*, pp. 1-10, 2008.

[Iyengar 06] Iyengar, V., *et al.*, "At-Speed Structural Test for High-Performance ASICs," *Proc. Intl. Test Conf.,* pp. 1-10, 2006.

[Krstic 02] Krstic, A., *et al.*, "Embedded Software-Based Self-Test for Programmable Core-Based Designs," *IEEE Design & Test of Computers*, pp. 18-27, 2002.

[Lambrechts 05] Lambrechts, A., *et al.,* "Power breakdown Analysis for a Heterogeneous Noc Platform Running a Video Applcation," *Proc. Intl. Conf. on Application-Specific Systems, Architectures, and Processors,* pp. 179-184, 2005.

[Lawton 06] Lawton, G., "Improved Flash Memory Grows in Popularity," *Computer,* pp. 16-18, 2006.

[Li 08] Li, Y., S. Makar and S. Mitra, "CASP: Concurrent Autonomous Chip Self-Test using Stored Test Patterns," *Proc. Design Automation and Test in Europe*, 2008.

[Li 09] Li, Y., O. Mutlu, and S. Mitra, "Operating System Scheduling for Efficient Online Self-Test in Robust Systems", *Proc. Intl. Conf. on CAD,* pp. 201-208, 2009.

[Martin 05] Martin, M.M.K., *et al.*, "Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset," *ACM SIGARCH Computer Architecture News*, 2005.

[Mitra 04] Mitra, S., and K.S. Kim, "X-Compact: an efficient response compaction technique," *IEEE Trans. CAD,* pp. 421-432, 2004.

[Nakao 99] Nakao, M., *et al.*, "Low Overhead Test Point Insertion for Scan-Based BIST," *Proc. Intl. Test Conf.*, pp. 348-357, 1999.

[Negreiros 02] Negreiros, M., L. Carro, and A. Susin, "A Statistical Sampler for Increasing Analog Circuits Observability," *Proc. Symp. on Integrated Circuits and Systems Design,* pp. 141-145, 2002.

[Nicolaidis 96] Nicolaidis, M., "Theory of Transparent BIST for RAMs," *IEEE Trans. Computers,* pp. 1141-1156, 1996.

[Parvathala 02] Parvathala, P., K. Maneparambil, and W. Lindsay, "FRITS: A Microprocessor Functional BIST Method," *Proc. Intl. Test Conf.*, pp. 590-598, 2002.

[Ramakrishnan 93] Ramakrishnan, K.K., "Performance Considerations in Designing Network Interfaces," *IEEE Journal on Selected Areas in Communications,* pp. 203-219, 1993.

[Saluja 88] Saluja, K.K., R. Sharma, and C.R. kime, "A Concurrent Testing Technique for Digital Circuits," *IEEE Trans. CAD,* pp. 1250-1260, 1988.

[Sengupta 99] Sengupta, S., *et al.*, "Defect-Based Test: A Key Enabler for Successful Migration to Structural Test," *Intel Technology Journal,* 1999.

[Shah 07] Shah, M., *et al.,* "UltraSPARC T2: A highly threaded, power-efficient SPARC SOC," *Proc. Asian Solid-State Circuits Conf.,* 2007.

[Shen 98] Shen, J., and J.A. Abraham, "Native Mode Functional Test Generation for Processors with Applications to Self Test and Design Validation," *Proc. Intl. Test Conf.,* pp. 990-999, 1998.

[Shiue 01] Shiue, W.T., and C. Chakrabarti, "Memory Design and Exploration for Low Power, Embedded Systems," *Journal of VLSI Signal Processing,* pp. 167-178, 2001.

[SUN 07] "Tuning on-chip 10GbE (NIU) on T5120/5220," http:// blogs.sun.com/puresee/entry/10_gigabit_ethernet_ on_ultrasparc.

[SUN 09] "OpenSPARC T2 Processor," http://www.opensparc.net/ opensparc-t2/index.html.

[Touba 95] Touba, N., and E.J. McCluskey, "Synthesis of Mapping Logic for Generating Transformed Pseudo-Random Patterns for BIST," *Proc. Intl. Test Conf.*, pp. 674-682, 1995.

[Tumin 01] Tumin, K., *et al.*, "Scan vs. Functional Testing – a Comparative Effectiveness Study on Motorola's MMC2107™," *Proc. Intl. Test Conf.,* pp. 443-450, 2001.

[Van Horn 05] Van Horn, J., "Towards Achieving Relentless Reliability Gains in a Server Marketplace of Teraflops, Laptops, Kilowatts, & Cost, Cost, Cost," *Proc. Intl. Test Conf.*, pp. 671-678, 2005.