On-Chip Diagnosis of Generalized Delay Failures using Compact Fault Dictionaries

Matthew Beckler, Member, IEEE, and R. D. (Shawn) Blanton, Fellow, IEEE

Abstract—One approach for achieving a robust integrated system centers on first performing test during runtime, then identifying the locations of any faults (or potential faults), and finally repairing/replacing/avoiding the affected portion(s) of the system. Conventional fault dictionary approaches can be used to locate failures but are limited to simplistic fail behaviors due to the significant computational resources required for dictionary generation and memory storage. Several contributions are described to overcome these limitations, and include: (i) enhancement of an unspecified-transition fault model (called here the transition-X fault model, or TRAX) for capturing the misbehaviors expected from scaled technologies, (ii) development of a hierarchical dictionary that only localizes to the level required, and (iii) the design of a scalable architecture for retrieving and using the hierarchical dictionary for on-chip failure diagnosis. The OpenSPARC T2 processor and other circuits are used in experiments to demonstrate the low-overhead, accurate diagnosis of early-life and wear-out failures, using TRAX dictionaries that are over five orders of magnitude smaller than full-response dictionaries.

Index Terms—Reliability, Fault diagnosis, Fault tolerance, Robustness.

I. INTRODUCTION

As SEMICONDUCTOR fabrication processes continue to scale, a variety of failure sources are becoming more pronounced and are therefore having a larger effect on correct system operation. Increasing variation in the manufacturing process [1] also means that yielding chips cost-effectively will be even more difficult. In addition, phenomena such as aging (sometimes called wear-out) and infant mortality (also called early-life failure) will pose new challenges to ensuring robustness [1], where robustness is the ability of a system to continue acceptable operation in the presence of various types of misbehaviors over its intended lifetime.

Robust system design is a problem that has been substantially addressed in the past. For example, one conventional approach for achieving system robustness is to use a conservative design technique that incorporates speed guardbands. Conservative design, however, results in significant performance loss and is increasingly expensive due to the extreme measures needed to overcome the level of variation exhibited by modern ICs. At the other extreme, one can aggressively design the IC assuming optimal fabrication conditions, and then use manufacturing test to select ICs that satisfy the specification. Such an approach would lead, however, to unacceptable cost, since yield would likely be extremely low. Traditional fault tolerance could also be employed. However, techniques such as TMR (triple modular redundancy) and duplicate-and-compare are much too expensive in terms of chip overhead and power consumption, especially for the many portable, consumer-based systems that are now pervasive.

Another class of fault tolerance includes “always-on” error-correction techniques. Although there is less area overhead, the level of error-correction needed would consume an excessive amount of power [2].

Another approach being actively investigated [3] involves periodically testing the system using tests that target the specific behaviors exhibited by known failure sources such as early-life failure (ELF) and wear-out. In various experiments involving actual test chips [4], [5], it is demonstrated that both ELF and wear-out manifest as delay increases in standard cells. Leveraging the already-existing design-for-testability (DFT) structures, thorough tests that target these delay shifts are brought into the chip and applied structurally (at configurable speeds) using the scan logic. In addition to detecting failure, adjusting the speed of the test also allows impending chip failures to be predicted before they actually occur. Testing is performed periodically to minimize the user-perceived performance loss while ensuring that the pending failure is detected before it corrupts any system data. If a failure or pending failure is detected by the test, diagnosis is performed to pinpoint the location of the affected portion of the system. To provide robustness, the system must then repair the faulty circuitry, replace it, or avoid it. One method for identifying the failure location, a fault dictionary, can be effective but has a number of drawbacks that include (i) the computation needed to generate the dictionary, (ii) the resulting size of the dictionary that can be easily multiple terabytes for modern designs, and (iii) the conventional use of the single-stuck line fault model [6] which significantly degrades diagnostic accuracy due to its mismatch with actual misbehavior.

Once a fault dictionary has been constructed, its use for on-chip diagnosis remains a challenge. The on-chip diagnosis architecture should have only a negligible effect on chip performance. For example, if hardware modules must be taken “off-line” for test and diagnosis, the duration should be minimized to reduce the performance loss. Furthermore, the hardware resources allocated for on-chip diagnosis should be limited and amortized to reduce the area and power impact on the overall chip.

In the following sections of this paper, we describe our approach for developing and using a fault dictionary with on-
/controllers (so-called system-on-chip (SoC) components such as cache and memory are enhanced with ample of system repair is detailed in [7], where non-processor system-on-chip (SoC) components such as cache and memory controllers (so-called uncore components) are enhanced with self-repair capabilities. The two repair techniques described in [7] include (i) resource reallocation and sharing, where helper components are reconfigured to share the workload of the faulty component, and (ii) sub-component logic sparing, where the design hierarchy is traversed to identify identical subcomponents that could share logic spares. Using both of these techniques, the authors of [7] achieve self-repair coverage of 97.5% with only 7.5% area and 3% power overhead for the OpenSPARC T2 processor design [8].

Before a repair technique can be applied however, the location of the faulty circuit must be identified. Moreover, the identification process must be performed in-situ, since it is unlikely it can be accomplished off-chip without affecting system performance. The process of identifying the failure location is known as diagnosis. In diagnosis, the conventional objective is to locate a fault site that corresponds to the actual failure location. Two categories of approaches exist for diagnosis. In effect-cause diagnosis, a complete model of the design, its test-vector set, and the test response from the failing circuit are analyzed by diagnostic software to identify possible fault locations. All commercial EDA vendors offer powerful software tools for diagnosing modern designs using effect-cause approaches. An effect-cause approach for on-chip diagnosis is likely infeasible since the amount of memory and run-time for modern designs can be extremely large. For example, simply loading the netlist image of a modern design into memory can take more than an hour.

The other approach to diagnosis is known as cause-effect. In this approach, all faults are simulated to create a cataloguing or dictionary of all possible test responses. Diagnosis is performed by comparing the actual circuit-failure response with all the fault-simulation responses stored in the dictionary. Fault sites having a response that matches or closely matches the actual failing-circuit response are identified as likely failure locations. Ideally, only one fault site is reported.

Unlike effect-cause approaches, the actual task of using a dictionary does not require a complex software analysis tool. Instead, the significant task of creating the dictionary is performed only once, off-line, before the design is even fabricated, thus allowing the immense computation required to be amortized over all systems that will utilize the dictionary. Performing diagnosis using a dictionary consists of lookups, which in an on-chip environment, can be efficiently accomplished by storing the dictionary in off-chip memory.

Conventional fault dictionaries are not perfect however in that they suffer from three significant drawbacks. First, as mentioned, the generation time is significant. This is due to the fact that every fault is simulated using all test patterns (i.e., there is no fault dropping), and the complete fault response needs to be generated in the most general case. Second, since the entire response for every possible fault is stored in the dictionary, its size can be huge. Finally, because the dictionary size is a tremendous challenge, the fault model employed must have a limited universe (e.g., on par with the stuck-at model). The consequence of using a simple fault model means however that it is much more likely that the actual circuit-failure response will differ significantly from any of the modeled fault responses stored in the dictionary.

Despite these drawbacks, dictionaries are the best choice for on-chip diagnosis since these disadvantages can be mitigated. Specifically, the cost associated with dictionary generation can be amortized across the lifetimes of all the systems that will use the dictionary for on-chip diagnosis. Dictionary size can be substantially reduced using conventional techniques, and by reducing the number of faults in the dictionary through a new form of collapsing that exploits the granularity of repair (Section IV). Additionally, advanced computational resources such as GPUs can exploit the highly-parallel nature of fault simulation to drastically reduce the amount of time required for dictionary generation. Finally, the failure-fault mismatch problem is handled through the use of an enhanced delay-fault model that conservatively captures the possible misbehaviors exhibited by ELF and wear-out without increasing the size of the stuck-at fault universe (details in Section III).

It is important to consider the differences between the goals of conventional fault diagnosis and the goals of on-chip diagnosis. In conventional fault diagnosis, the focus is on process improvement and/or low-level design improvement via physical failure analysis (PFA) of the sites reported by diagnosis. PFA requires a precise result, ideally the actual x-y-z location of the failure. The on-chip diagnosis requirements are comparatively relaxed however, requiring localization only to the level of repair or avoidance, which is likely a much larger chip area than a single net or standard cell. This relaxation of localization precision significantly reduces the size of an on-chip dictionary (details in Section IV).

An additional application of such an on-chip diagnosis process is during part bring-up early in product development. With low yields limiting the number of available chips, any process to enable the correct operation of early silicon would be very valuable.
III. Fault Model for Robustness

Early life failure (ELF) caused by latent defects that are not detected by manufacturing tests, and various wear-out mechanisms, such as NBTI (negative bias temperature instability), means computer systems are not inherently robust. Over time, NBTI changes the threshold voltage of a PMOS transistor which manifests as decreased transistor drive current, ultimately leading to circuit-speed degradation [9], [10]. NBTI first became a significant issue at the 90nm technology node [4] but worsens as technology continues to scale [11]. As already alluded to, ELF can result from hidden, latent chip defects that have not fully manifested when manufacturing testing occurs. These chips can fail early in their lifetime, much earlier than the specified product lifetime. ELF-affected transistors will experience gradual delay increases over time before functional failures actually occur. The increase in delay can be sufficiently large so that their detection can be accomplished using aggressively-clocked tests [5]. NBTI degradation can also be similarly discovered before it too leads to functional failures.

An ideal fault model to represent the slow down caused by ELF and NBTI is the gate-delay fault (GDF) model [12]. Detection of a GDF requires producing a transition at the affected gate output, and sensitizing a circuit path from the gate to one or more primary outputs. A GDF will be detected only if the additional delay is larger than the slack of a path appropriately sensitized. Identifying a test to detect a GDF is thus an optimization problem which involves searching for a sensitzible path through the gate that exhibits the minimal slack. Optimization adds undesired complexity to the already-intractable test generation process, making deployment of the GDF model for use in a dictionary much more challenging.

The most commonly-deployed delay fault model is the transition fault (TF) model [12]. Whereas the GDF model makes no assumption about the increase in delay, the TF model on the other hand assumes the delay to be larger than the slack of any sensitized path through the fault site. This reduces complexity significantly, since any test that sensitizes a path from the fault site to an observable point will detect a TF fault, that is, test generation remains a satisfiability problem as opposed to an optimization problem. The TF model is more tractable but is very pessimistic since it implicitly assumes that every fault will produce errors at the output of each sensitized path.

The unspecified-transition fault model is a compromise between the gate-delay fault model and the transition fault model [13]. The unspecified-TF model assumes the fault site produces an unknown value \( X \) when activated. While the original work details a range of fault activation and error propagation requirements, these properties are not relevant here due to the scan-based testing scheme employed in this work. Additionally, an enhancement of this fault model has been developed in this work that includes activation and propagation due to glitches resulting from hazards, noise, etc. that we refer to as the TRArxn X (TRAX) fault model.

Similar to transition faults, TRAX faults exist as either a slow-to-fall (STF) or slow-to-rise (STR) fault, and are activated by either a transition of the correct polarity (1-to-0 or 0-to-1, respectively) or by a glitch hazard at the fault site. In addition to the typical zero and one logic values, the TRAX fault model utilizes the unknown \( X \) value, as well as hazard values. This results in a four-valued algebra of \{0, 1, X, H\}, which requires extending the truth tables for the eight primitive gates as shown in Figure 1. In the two-vector test scheme employed here, a hazard value is generated at a gate output when the gate inputs do not have a constant controlling input. For example, a two-input NAND gate generates a hazard output when the inputs transition 01 \( \rightarrow \) 10 or 10 \( \rightarrow \) 01.

An advantage of this fault model is that the \( X \) value captures all the possible transport-delay changes that can be exhibited by a gate affected by NBTI or ELF since the propagation of the \( X \) value is conservative. Whereas the errors from a conventional transition fault could interact to either increase or reduce the number of fault-effect observations (i.e., failing outputs), the use of the \( X \) value can only increase the number of failing outputs. In other words, the set of sensitized paths with \( X \) values will subsume the sensitized paths associated with any GDF of any delay. In addition, any error interaction (“fault masking”) that may occur due to the TF fault model assuming gross slowdown is also conservatively handled.

Fault simulation of TRAX faults is very similar to the TF model. Instead of assuming a grossly-slowed transition, an \( X \) value is injected when a TRAX fault is activated. These injected \( X \) values are propagated through the simulated circuit, producing the fault-free values of 0, 1, or the potentially faulty value \( X \) at the observation points.

It is important to further discuss the interpretation of observed \( X \) values at the circuit outputs. Observed \( X \) values resulting from TRAX fault simulation identify outputs that could, but not necessarily, be affected by a slowdown at the fault site. In other words, outputs that have \( X \) values may or may not exhibit an error, depending on the level of slowdown. We assume for every test that detects a given TRAX fault that at least one observed \( X \) value will actually be erroneous, thus guaranteeing detection. Since no knowledge about the timing is assumed, it cannot be known a priori however which output...
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2018.2803621, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

4 IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, VOL. , NO. , MONTH YEAR

Table I

<table>
<thead>
<tr>
<th>Fault site</th>
<th>Fault type</th>
<th>Test: abde</th>
<th>Output: fg</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>STF</td>
<td>v1: 10100</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>v2: 11100</td>
<td>11</td>
</tr>
<tr>
<td>F2</td>
<td>STR</td>
<td>v1: 11010</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>v2: 11110</td>
<td>10</td>
</tr>
</tbody>
</table>

will take on an incorrect value when a TRAX fault is detected.

The ISCAS85 benchmark circuit c17 [14] shown in Figure 2 is used to contrast the TF and TRAX models. TF and TRAX faults are simulated at sites F1 and F2, respectively, using two test-vector pairs. Table I shows how fault effects propagate from each fault site to the circuit outputs. The first pair of vectors activates a slow-to-fall fault at F1; both the TF and TRAX produce a faulty value on the lower output g. For this case, both faults are detected and produce the same response. The second pair of vectors activates a slow-to-rise fault at F2. Here, more complex behavior arises, demonstrating the difference between TF and TRAX. For TF, the reconvergent fanout logically masks the slowed transitions reaching the upper output f. Masking is not possible for TRAX, resulting in both outputs having X values, which subsumes the response of the TF. The X values at the two outputs means it is possible that the actual response of a slowdown could subsume, match, or mismatch the TF response.

To further explore and validate the TRAX subsumption of TF fault responses, we perform a series of circuit simulation experiments on a few benchmark circuits. Fault simulation is used to determine the number of tests that detect each simulated fault, as well as determine the count of failing circuit outputs. This fault simulation is performed for both TRAX and TF faults, for four different benchmark circuits. The resulting counts of fault detecting tests and failing output bits are normalized for each circuit to permit cross-circuit comparison. Figures 3 and 4 show a scatter plot comparison between TRAX and TF fault responses for each simulated fault. Each point in these two figures is a single simulated fault, where the position is determined by the normalized count of failing outputs (Figure 3) or detecting tests (Figure 4) for TRAX and TF simulations of that fault. A line is drawn to indicate where TRAX and TF have equal counts. We observe that for both detections and failing output count, for every fault, the TRAX behavior subsumes the TF behavior, as made evident by no faults falling below the indicated line.

IV. FAULT DICTIONARY CONSTRUCTION

It is convenient to view a fault dictionary as a table of values with one row for each fault and one column for each test, where table entries represent the fault simulation response (or some subset thereof) of the corresponding fault/test pair. A fault dictionary that contains the full response for each fault/test pair is a full-response dictionary. A variety of compaction schemes ([15]–[22]) have been developed for reducing the size of a fault dictionary, each exhibiting their own trade-off between diagnostic resolution and dictionary size/complexity. Existing techniques are effective but do not enable a sufficient level of compaction when applied to modern designs [17], [23]. One straightforward compaction technique is to only include faults relating to the targeted defect model (i.e., NBTI and ELF). To this end, the TRAX fault dictionary explored here only includes faults located at standard-cell outputs since NBTI and ELF affect the delay of standard cells. Elimination of the remaining faults (i.e., faults located at primary inputs or fanout branches) reduces dictionary size through the removal of rows.
Given that system resources for test and diagnosis must be limited, it is not feasible to store the entire simulation response of every fault. Instead, a single bit per test is stored to indicate pass/fail status of the corresponding simulation response, resulting in a pass/fail dictionary. A pass/fail dictionary is a well-known technique for significantly reducing the size of a fault dictionary at the expense of diagnostic resolution [22]. Resolution degradation however is minimal since the level of precision required is limited to the repair level. Specifically, for all benchmark circuits surveyed, the resolution loss averaged less than 0.023%, where resolution is equated to the number of distinguished fault pairs divided by the total number of fault pairs.

For the purposes of on-chip test and diagnosis, the tested core or uncore [3][24] is assumed to be partitioned into a set of interconnected modules, each of which can be independently repaired, replaced, or avoided if found to be faulty. Figure 5 illustrates an example system-on-chip that contains three cores/uncores which themselves are composed of multiple modules that include TRAX faults shown as circles within each module. The modules within each core/uncore are assumed to be at the repair level. In other words, these modules are at the finest level of granularity for which fault repair, replacement, or avoidance can be cost-effectively deployed. Put another way, it would be ideal if the actual transistor responsible for slowdown could be both identified and repaired/replaced/avoided. Achieving robustness however at this very fine level of granularity is likely too costly. So instead, the repair-level is defined to be the set of sub-circuits (i.e., modules) within a core or uncore that can be effectively repaired/replaced/avoided. We make no assumptions however how small or large these modules should be since there are challenging tradeoffs involving performance, power, and robustness. Regardless of how these modules are defined, the approach for on-chip diagnosis described here remains applicable, albeit with an associated level of efficiency and effectiveness that is dependent on core/uncore partitioning into modules. Finally, it is also important to point out that the core/uncore is the smallest testable entity. That is, we assume it is not cost-effective to test each module within a core/uncore separately from all other modules. If this were not true, then diagnosis would be unnecessary since test failure alone would identify the culprit module.

Because of the repair level, on-chip diagnosis is significantly eased as compared to conventional approaches since precision can be relaxed to the module-level of localization. Existing dictionary compaction techniques have not considered this relaxation in diagnosis precision and are therefore not directly applicable to the on-chip diagnosis task formulated here. In other words, diagnosing to the repair level means that circuit-level resolution (e.g., signal line, standard cell, or layout) can be replaced with module-level resolution to achieve a smaller dictionary size. This means that only certain inter-module fault pairs have to be distinguished within the fault dictionary (i.e., have dictionary rows where at least one entry is different).

For intra-module faults that are test-set equivalent (identical simulation responses for all tests), all but one can be eliminated since distinguishing them using advanced diagnosis techniques (e.g., [25]) is now unnecessary. Other intra-module faults can also be eliminated based on subsumption. Subsumption is similar to dominance but is subtly different due to the use of the X value and the opposite way it is exploited to reduce the fault universe. A fault \( c \) is said to subsume another fault \( d \), if \( c \) and \( d \) are not test-set equivalent and the erroneous test response of \( c \) subsumes the erroneous response of \( d \). In other words, fault \( c \) subsumes fault \( d \) if \( c \) produces an X value for all the same tests and outputs as fault \( d \), and their fault-simulation responses are not identical. A subsumed fault (e.g., fault \( d \)) can be eliminated since, if it exists, it will produce a response that can also be produced by fault \( c \). Again, since faults \( c \) and \( d \) are in the same module, there is no need to distinguish them. Note that the subsuming fault (e.g., fault \( c \)) cannot be eliminated since it could produce a response that is not captured by another fault in the module, thus making it possible that it would not be distinguished from some other fault outside the module if it were removed.

Using subsumption to eliminate faults is not the same as dominance-based fault collapsing. First, in dominance collapsing, the dominating fault (e.g., fault \( c \)) is the one collapsed. Second, and more importantly, dominance-collapsed faults cannot be altogether eliminated in an application such as ATPG since they must still be explicitly considered if a test cannot be found for any of its dominated faults. The dictionary that results from applying equivalence and subsumption fault elimination to a pass/fail dictionary is referred to as a collapsed pass/fail dictionary.

Measuring dictionary size reduction due to fault elimination requires access to designs that exhibit module-level hierarchy. The hierarchical versions of the ISCAS85 benchmark circuits [14] produced by the University of Michigan [26] serve this purpose well since many of the benchmarks have been reverse engineered to at least two levels of hierarchy. Additionally, three other sources of benchmarks are utilized. First, seven circuits are taken from the ITC99 benchmark

---

1It should be noted that fault subsumption is only valid for a fault model such as TRAX since conventional fault models have only a single simulation response. TRAX faults, on the other hand, have a set of possible responses given that the observed X values all do not have to correspond to errors.
Fig. 6. Comparison of fault-dictionary size for various circuits and compaction techniques.

TABLE II

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Faults</th>
<th>Test pairs</th>
<th>Fault coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>b12</td>
<td>1,880</td>
<td>80</td>
<td>100.00%</td>
</tr>
<tr>
<td>b14_opt</td>
<td>10,694</td>
<td>345</td>
<td>99.97%</td>
</tr>
<tr>
<td>b14</td>
<td>19,534</td>
<td>657</td>
<td>99.67%</td>
</tr>
<tr>
<td>b15</td>
<td>16,734</td>
<td>407</td>
<td>97.39%</td>
</tr>
<tr>
<td>b17_opt</td>
<td>45,514</td>
<td>600</td>
<td>98.38%</td>
</tr>
<tr>
<td>b18_opt</td>
<td>139,826</td>
<td>704</td>
<td>99.71%</td>
</tr>
<tr>
<td>b22</td>
<td>59,568</td>
<td>554</td>
<td>99.69%</td>
</tr>
<tr>
<td>L2B</td>
<td>24,444</td>
<td>90</td>
<td>99.99%</td>
</tr>
<tr>
<td>NCU</td>
<td>67,124</td>
<td>211</td>
<td>99.95%</td>
</tr>
<tr>
<td>epflDiv</td>
<td>203,206</td>
<td>612</td>
<td>99.25%</td>
</tr>
<tr>
<td>epflMult</td>
<td>101,008</td>
<td>77</td>
<td>99.96%</td>
</tr>
<tr>
<td>epflSqrt</td>
<td>82,084</td>
<td>161</td>
<td>91.91%</td>
</tr>
<tr>
<td>epflSquare</td>
<td>70,984</td>
<td>79</td>
<td>100.00%</td>
</tr>
<tr>
<td>epflMemCtrl</td>
<td>165,076</td>
<td>1,276</td>
<td>99.84%</td>
</tr>
</tbody>
</table>

Each set normalized to the size of each full-response dictionary. The average reduction achieved for TRAX is 667x. For TRAX-without-hazards the maximum reduction is 27,153x for NCU, with an average reduction of 161x.

V. On-Chip Test and Diagnosis

The on-chip diagnosis work presented here assumes the availability of a suitable on-chip test framework, specifically CASP (Concurrent Autonomous chip self-test using Stored test Patterns) [3], [7], [30]. This section presents an explanation of the workings of the CASP on-chip test architecture. The overall approach employed by CASP is to periodically test core/uncore components of a SoC during runtime, without creating any user-visible system downtime. The key ideas of CASP include:
The core of the CASP process is the CASP controller, a finite state machine (FSM) implemented in additional hardware added to the IC design. The CASP controller iterates through the following high-level steps described next and also illustrated in Figure 7.

1) Scheduling - A core/uncore is selected for the next test cycle. This selection can be as simple as a round-robin approach where each core/uncore is tested in turn, or selection can be guided by usage (workload) or based on canary circuits indicating which cores/uncores are likely to need testing.

2) Isolation - The selected core/uncore is isolated from the rest of the system. For a CPU core this typically involves stalling the execution pipeline, waiting until in-flight instructions complete, invalidating the local private cache(s), and saving critical states to shadow registers. This can be significantly simplified on systems utilizing virtualization [24], where the virtual machine monitor can pause or migrate the CPU selected for testing without any hardware modifications or shadow registers. For an uncore component, isolation typically involves migrating the workload of the tested uncore to a so-called “helper” uncore that provides some or all of the functionality of the tested uncore, enabling continued operation during test [31]. For example, since CASP does not cover memory arrays such as caches (existing resilience techniques for on-chip memories include row/column sparing, built-in self test, and error-correcting codes), when a cache controller is under test, the corresponding cache memory is still available for use. A neighboring cache controller can respond to requests for data stored in the cache-under-test, provided it is made aware of the data stored there. While this sharing may require additional hardware (such as enabling a cache controller to distinguish between data cached locally or in a neighboring cache), and can result in a performance degradation due to the same workloads sharing fewer active uncores, this is a better result than pausing the tasks of the uncore under test.

3) Testing - The high-quality test set is loaded from off-chip (flash, DRAM, or hard drive) into on-chip buffers. The CASP controller sets the proper signals and states in the core/uncore under test to enable the JTAG interface that is used to load the test data into the scan chains.

4) Reintegration / Recovery - After the test phase is complete, if the core/uncore passes all tests it must be reintegrated back into the system. Otherwise, appropriate recovery actions are taken for the faulty core/uncore. For a non-faulty core/uncore, the isolating actions of phase two are reversed, restoring any saved state, restarting execution, and invalidating any potentially-state data from data caches. For a CPU core, the execution state is restored from shadow registers, or the virtual machine OS is migrated back to the tested core (as in [24]). For an uncore, this may involve migrating state from the “helper” uncore that temporarily handled requests to the uncore under test.

A few notes must be made as to how this architecture fits into the broader SoC. The CASP testing process (step 3 above) compares the circuit response against the expected response, for each applied test, to determine if the core/uncore is operating correctly. If any failures are detected, the on-chip diagnosis process is performed using the recorded circuit responses and the diagnosis architecture presented in Section VI. With regard to connections between the CASP test controller and the individual cores/uncores that must be tested, there are a few details to consider. First, the test controller must have access to override the control logic of each core/uncore in order to isolate it and apply the test. Additionally, to perform circuit failure prediction [4], where test pairs are applied at speeds greater than the normal system clock rate, in an effort to detect gradually-slowing gates, the test controller must have the ability to supply an accelerated clock to each testable core/uncore. This is a no easy feat, as significant IC routing resources are dedicated to the normal clock distribution network, and it can be a challenge to provide a secondary high-speed cross-chip clock distribution network.

The objective of on-chip diagnosis is to determine which module of the core/uncore is most likely the reason for any observed test failure. For on-chip diagnosis, the previously-described TRAX fault dictionary is used, which stores a single bit for certain fault/test pairs that indicates if the fault is detected by the corresponding test. Using the TRAX fault dictionary in conjunction with the pass/fail test response from the CASP test execution, a list of potentially-responsible modules is generated. Specifically, starting with a list of all faults within the tested core/uncore dictionary, each failing test response is used one at a time to eliminate any faults that are not detected by the corresponding test. When all test responses have been processed, what remains is a list of TRAX faults consistent with the observed behavior. Each fault in this list belongs to one of the modules of the tested core/uncore. Moreover, counting the number of potentially-responsible faults in each module gives an indication of the likelihood of finding the actual fault in each of the implicated modules, and thus is reported as the final outcome of on-chip diagnosis.

On-chip diagnosis can be performed in two different situations, the first being failure prediction, where an accelerated
clock is used to detect pending failures. In this situation the diagnosis can be run in the background with no performance overhead, with just some extra power consumption. The other situation is after a real failure is detected, implying that the core/uncore is unable to be used because it has failed. For this case, diagnosis time is important because there is performance overhead associated with diagnosing the failure and deploying fault repair or avoidance before its execution can resume.

VI. TEST AND DIAGNOSIS ARCHITECTURE

In this section, we describe the hardware architecture developed for performing on-chip test and diagnosis. Specifically, in Section VI-A, the off-chip storage used for test and diagnosis data is discussed, while Section VI-B describes the on-chip hardware for actually performing the test and diagnosis using the off-chip data. More emphasis is placed on the diagnosis since the CASP [3] methodology for testing is assumed.

A. Off-Chip Hardware

To enable on-chip test and diagnosis requires the addition of specialized hardware, both off and on the chip itself. The off-chip hardware includes memory for storing the test and dictionary data. The use of off-chip data storage is prudent because the high-quality test patterns required for ELF and wear-out failure detection cannot be obtained using conventional BIST techniques, for example. Additionally, the use of off-chip storage allows the test and dictionary data to be updated (patched), if better-performing data is later identified.

Each unique core (e.g., CPU, FPU, or GPU) and uncore (DRAM controller, cache controller, etc.) has its own separate fault dictionary. The use of multiple fault dictionaries, one per tested core/uncore, is justified since it is the smallest testable/repairable object in the chip. Using separate dictionaries reduces the overall amount of dictionary data since faults from the tested core/uncore do not have to be distinguished from all the faults of all the remaining untested cores/uncores. The actual data stored in the off-chip memory consists of test vectors, their expected responses, and response-mask vectors that indicate which response bits should be ignored. Additional data necessary for on-chip diagnosis includes the fault dictionary data for each core/uncore. One example structure of the off-chip data storage is illustrated in Figure 8.

B. On-Chip Hardware

In addition to the off-chip storage, there is dedicated on-chip hardware for actually performing diagnosis that we call the Diagnosis Architecture (DA); its structure and operation are given in Figure 9 and Algorithm 1, respectively.

Test execution produces a pass/fail bit for each test, which is stored in an on-chip circular shift register called the PF (Figure 9a), where a one indicates the test has failed (Tester Fail) and a zero indicates a test has passed (Tester Pass). The dictionary diagnosis data for the $F$ faults of the tested core/uncore is associated with each test and includes a pass/fail list of $F$ binary values, one per fault, where a one indicates that the corresponding fault is detected by the test (Simulation Fail), and a zero indicates a non-detection (Simulation Pass). The pass/fail test response in the PF is compared with the pass/fail simulation response of each fault (line 7 of Algorithm 1). A fault is eliminated if there is any one-sided failure disagreement between the test response and the fault response stored in the dictionary. Specifically, if there is a Tester-Fail Simulation-Pass (TFSP) found for any test, then the corresponding fault is eliminated from consideration.
Elimination of a fault using TFSP is justifiable since the TRAX fault model is very conservative in nature. In other words, it is very unlikely for a slowed standard cell to fail a given test without the corresponding fault also being detected given both the generalized activation and fault-effect propagation properties exhibited by the TRAX fault model.

Figure 9b shows the part of the DA responsible for finding any cases of TFSP. Because the number of faults $F$ for a tested core/uncore can be large, the DA only compares $k < F$ faults at a time. Specifically, for a given test $t_i \in T$, its pass/fail response bit stored in the PF is simultaneously compared with the pass/fail simulation bits from $k$ faults (line 7 of Algorithm 1). The result of each comparison is stored in a circular shift register as shown in Figure 9a. The register and the associated hardware for identifying any cases of TFSP among $k$ faults is called the fault accumulator (FA). Locations within the FA that store a logic zero indicate which of the $k$ faults are potentially responsible for the core/uncore failure.

The remaining portion of the DA (Figure 9c) implements the faulty module identification circuitry (FMIC). The FMIC associates the potentially-responsible faults (locations in the FA with a zero stored) with their corresponding modules within the tested core/uncore. Additionally, the FMIC counts the number of potentially-responsible faults associated with each module, allowing their relative ranking based on the number of faults that each contains. The FMIC receives the fault data from the FA register serially through its shift function. A counter called the fault-index (FI) is used to track which of the $F$ faults of the tested core/uncore is being examined. This implies that fault data within the dictionary are arranged by modules so that all the faults for each module are adjacent. Based on this fault ordering, a comparator and a bank of counters are used to count the number of potentially-responsible faults that belong to each module. These per-module fault counts are returned to the operating system for further processing and decision making.

Before diagnosis begins, all $(M + 1)$ counters in Figure 9c are reset (line 2 of Algorithm 1). As each bit is shifted in from the FA, the FI counter is incremented. As already mentioned, faults for a given module are grouped together so that there is a distinct range of FI count values for each module of a core/uncore. The boundary values between adjacent modules in the fault list are stored in a group of registers called the module index registers (MIRs) which are appropriately initialized for the core/uncore under test (line 3 of Algorithm 1) before diagnosis begins. The MIR values are loaded into a multi-bit shift register with $(M - 1)$ entries. A parallel single-bit shift register is initialized to a one-hot code corresponding to the first module; these bits are used to generate the enable signals to each of the $M$ counters. When the FI counter comparator equals one, indicating that the next fault boundary has been reached and this is the last fault of the current module, the two parallel shift registers are configured to perform a shift in the next clock cycle. This directs the next boundary MIR into the comparator, and moves the single one-bit of the counter-enable shift register to the next module position. When the incoming FA data indicates that fault $i$ is compatible with the core/uncore test response, the corresponding module counter (module_count[$j$]) is incremented (lines 11-17 of Algorithm 1).

A few more comments should be made about the DA operation given in Algorithm 1. Lines 2 and 3 perform initialization, and line 4 executes the loop that uses the fault accumulator to process all $F$ core/uncore faults in groups of $k$ faults at a time. Lines 5-9 compare the test response with the fault responses of the $k$ faults, and lines 10-20 describe the update of the module counters by the FMIC. For a tested core/uncore with $F$ faults and $T$ tests, \( [F/k] \cdot T_{\text{max}} + F \) cycles (ignoring the few cycles needed for initialization) are needed to perform diagnosis, where $T_{\text{max}}$ is the maximum number of tests needed for any core/uncore in the SoC. Note that $T_{\text{max}}$ and not $T$ appears in the expression \( [F/k] \cdot T_{\text{max}} + F \) since the PF register is sized for the largest core/uncore. This means for any $T < T_{\text{max}}$, the PF register must be shifted $T_{\text{max}} - T$ times after each group of $k$ faults are processed by the FA (lines 21-23 of Algorithm 1).
Algorithm 1 – On-Chip Diagnosis

1: perform test and store response in PF
2: reset FL counter and M module counters in FMIC
3: initialize \((M - 1)\) module index registers (MIRs)
4: for \(c = 0\) to \([F/k]\) do
5: reset fault accumulator \((FA[0:k-1] = 0)\)
6: for \(i = 0\) to \((T-1)\) do
7: \(FA[0:k-1] = (\sim\text{fault}[ck:(c+1)k-1][i] \& PF[0])\)
8: \(PF = PF >> 1\)
9: end for
10: for \(i = 0\) to \((k-1)\) do
11: for \(j = 0\) to \((M - 1)\) do
12: if \(FA[0] == 0\) then
13: if lower_bound\([j]\) \(\leq\) FI \(<\) upper_bound\([j]\) then
14: module_count\([j]\)++
15: end if
16: end if
17: end for
18: \(FA = FA >> 1\)
19: FI++
20: end for
21: for \(i = 0\) to \((T_{max} - T)\) do
22: \(PF = PF >> 1\)
23: end for
24: end for

in order to analyze the next group of \(k\) faults starting with the first test. It should also be noted that the for loop describing the module fault counter operation of the FMIC (lines 11-17 of Algorithm 1) is actually executed in parallel as shown in Figure 9c; the for loop in Algorithm 1 is only used to ensure clarity. For one of the largest circuits analyzed (NCU, with \(T = 6315, F = 29732, M = 10\)), the total number of cycles required for \(k = \{1000; 5000; 10000\}\) is \(219450; 67890; 48945\) cycles, respectively.

The chip area overhead of the DA hardware has also been calculated as a closed-form expression. This expression is a function of the largest number of faults \((F_{max})\), test vectors \((T_{max})\), and modules \((M_{max})\) in any core/uncore, as well as the number of modules concurrently processed \((k)\) by the FA. The variable \(x\) is defined as the number of bits required to store a fault identifier \((x = [\log_2 F_{max}])\). The number of gates used in the DA hardware is \((16 \cdot T_{max} + 4) + (23 \cdot k) + (40 \cdot x \cdot M_{max} + 2 \cdot x + 20 \cdot M_{max} + 3)\), where the grouped expressions belong to the PF, FA, and FMIC modules, respectively. For NCU, the total number of gates required for \(k = \{1000; 5000; 10000\}\) is \(\{130277; 222277; 337277\}\) gates, respectively. Since the DA hardware is used for all diagnosis activities, and is not dedicated to any specific core or uncore, the overhead is relatively small. Based on an estimated total of 500M transistors for the OpenSPARC T2 processor (containing the NCU and L2B uncores, among others), the gates added for the DA hardware result in an overhead of less than \(\{0.16\%; 0.27\%; 0.40\%\}\) for \(k = \{1000; 5000; 10000\}\), respectively. Given this relatively small overhead, and the fact that the DA hardware is unable to test or diagnose its own failures, it is reasonable to use conservative transistor sizing or triple-module redundancy to ensure robustness of the DA hardware in the absence of runtime test and diagnosis.

VII. EXPERIMENTS

To validate the diagnosis architecture and evaluate the quality of the TRAX dictionary, gate-level simulation is performed to mimic gate slowdown due to ELF. Specifically, additional delay is added to a randomly-selected gate and then simulated using the ATPG test patterns. The injected gate delay is gradually increased until incorrect responses are observed. Slowly increasing the gate delay in this way mimics the gradual change due to ELF, approximating the situation where system test is periodically performed to ensure detection before the slowdown becomes excessive. Each test set response is tabulated as a list of passing and failing tests, which is used by the DA hardware (as described in the previous section) to perform on-chip diagnosis, resulting in a list of modules and their associated counts of potentially-responsible faults.

To evaluate diagnosis quality for each injected gate delay fault, we adapt the conventional metrics of resolution and accuracy for a module-based design. Resolution is defined as the number of modules with non-zero fault-count values. Ideal resolution results when only one module has a non-zero fault-count value. The diagnosis result is deemed accurate if the module with the injected gate delay has a non-zero fault-count. Ideal accuracy occurs when the module with the injected delay has the largest count. Collective resolution and accuracy values are reported in Table III for each circuit.

In addition to the diagnosis characteristics previously reported [32], an alternative method of accuracy characterization (module fault count normalization) is also included. Instead of selecting the module with the largest count of candidate faults, the final row of Table III first normalizes the count of candidate faults in each module, by dividing each count by the total number of faults in the module. This normalization lends additional diagnostic weight to smaller modules with a higher proportion of candidate faults than larger modules with a lower proportion of candidate faults. For example, a large module may have hundreds of candidate faults remaining after diagnosis, yet those faults may amount to only a small percentage of the module fault sites. Further, a small module may have a high percentage of candidate faults remaining, but that may amount to only a handful of faults. When ranking the module likelihood, the candidate fault count normalization uses the proportion of candidate faults instead of the absolute count, in an effort to avoid penalizing smaller modules. In practice, candidate fault count normalization does not result in significant improvement to the proportion of ideal accurate diagnoses, and can result in reduced ideal accuracy for some circuits.

Unlike the results of previous work [32], using a fully TRAX-compatible fault simulator eliminates any instances of gate-injected delay fault diagnoses where there are no modules being reported, i.e., “empty diagnosis”. Additionally, all of the diagnoses are accurate, where the faulty module is correctly reported in the set of potentially-responsible modules.
Unfortunately, none of the diagnoses have ideal resolution (that is, only one module with non-zero fault count). This is a side effect of the very-conservative TRAX fault model correctly subsuming all possible faulty behavior, and when every diagnosis result has candidate faults in at least two modules, this results in no diagnoses with ideal resolution. It is worth noting however that ideal accurate diagnoses (those diagnoses where the culprit module has the highest count) also result in an effective ideal resolution if the module count values are used for final determination of the faulty module. Future work in this area will focus on improving diagnostic resolution while maintaining the existing high accuracy.

Prompted by the observation of relatively high mean resolution values nearly equal to the number of modules, further fault simulation is performed where the TRAX fault module is slightly modified to disallow fault activation due to hazard values. The results of this are presented in Table IV, and show that a slight increase in empty and inaccurate diagnoses can be traded for an increase in the ideal accurate and ideal resolution diagnoses and a reduction of the mean resolution. The normalization of candidate fault counts to module size does not provide an improvement in ideal accuracy here with the TRAX-without-hazards model.

Additional experiments are performed to evaluate the diagnosis of TF or TRAX injected faults using fault dictionaries created using the TF or TRAX model. Instead of gradually-increasing the injected gate delay as with TRAX, injected TF gate delays are set to the system clock period to simulate a gross delay through the selected gate. Diagnosis results for c7552, L2B, and NCU show that the TF dictionary has an increased number of empty diagnoses when compared to the TRAX-without-hazards model. This result further supports the conclusion that a fault dictionary created using TRAX fault simulation requires less storage and offers improved diagnostic performance compared to a TF fault dictionary, for both TRAX and TF faults.

As previously mentioned, the non-ISCAS85 benchmarks lack any pre-existing module-level hierarchical and are partitioned into ten modules using graph clustering software. A brief analysis is performed to determine the sensitivity of dictionary characteristics to the number of module partitions. This analysis consists of repeatedly performing the following steps:

1) Use graph clustering to partition the circuit into \( m \) modules.
2) Construct and compact the TRAX fault dictionary.
3) Evaluate diagnostic ability of resulting fault dictionary.

These steps are performed for \( m = 5 \) to 30, for both the L2B circuit (Figure 10) and the NCU circuit (Figure 11). As the number of module partitions increases from 5 to 30, there is a slight downward trend in the percentage of ideal accurate diagnosis results, and an upward trend in the size of the fault dictionary. While it may appear counter-intuitive for the dictionary size to increase as the number of modules increase (which implies fewer faults per module) there is a logical explanation. Dictionary compaction due to equivalence and subsumption is only applied to intra-module fault pairs, so more compaction is possible with larger modules and therefore

TABLE III

<table>
<thead>
<tr>
<th></th>
<th>c7552</th>
<th>L2B</th>
<th>NCU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of diagnoses</td>
<td>5530</td>
<td>2444</td>
<td>450</td>
</tr>
<tr>
<td>Number of sub-modules</td>
<td>12</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Empty diagnoses</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>Mean resolution</td>
<td>9.67</td>
<td>9.38</td>
<td>9.19</td>
</tr>
<tr>
<td>Ideal resolution</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>Accurate diagnoses</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Ideal accurate diagnoses</td>
<td>18.15%</td>
<td>23.01%</td>
<td>29.45%</td>
</tr>
<tr>
<td>Ideal accurate diagnoses (norm)</td>
<td>63.92%</td>
<td>25.63%</td>
<td>25.47%</td>
</tr>
</tbody>
</table>

TABLE IV

<table>
<thead>
<tr>
<th></th>
<th>c7552</th>
<th>L2B</th>
<th>NCU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of diagnoses</td>
<td>5530</td>
<td>2444</td>
<td>450</td>
</tr>
<tr>
<td>Number of sub-modules</td>
<td>12</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Empty diagnoses</td>
<td>13.93%</td>
<td>19.10%</td>
<td>16.17%</td>
</tr>
<tr>
<td>Mean resolution</td>
<td>6.64</td>
<td>5.17</td>
<td>6.48</td>
</tr>
<tr>
<td>Ideal resolution</td>
<td>9.94%</td>
<td>19.96%</td>
<td>15.66%</td>
</tr>
<tr>
<td>Accurate diagnoses</td>
<td>79.96%</td>
<td>78.75%</td>
<td>81.27%</td>
</tr>
<tr>
<td>Ideal accurate diagnoses</td>
<td>33.20%</td>
<td>38.89%</td>
<td>30.01%</td>
</tr>
<tr>
<td>Ideal accurate diagnoses (norm)</td>
<td>14.96%</td>
<td>38.20%</td>
<td>22.73%</td>
</tr>
</tbody>
</table>

Fig. 10. Experiments performed for L2B show that as the number of module partitions increase, there is a slight downward trend in ideal accurate diagnoses and a slight upward trend in dictionary size.

Fig. 11. Experiments performed for NCU show that as the number of module partitions increase, there is a slight downward trend in ideal accurate diagnoses and a slight upward trend in dictionary size.
more intra-module fault pairs.

While this may lead to the conclusion that a smaller number of modules is more desirable by these two metrics, this conclusion does not include two important considerations. When the number of modules is small, any detected defect will require the repair, replacement, or avoidance of a significant fraction of the core/uncore itself. Furthermore, in the context of replacing modules, this reduces the ability of the core/uncore to recover from any future defects, as more of the available spare modules have been diagnosed as faulty and are unavailable for future use. Taken to extremes, a core/uncore partitioned into a single module would have a zero-byte dictionary with perfect accuracy (every defect is located within the single “module” of the core/uncore), but the available recovery options are quite undesirable. It is not very practical to wait to repair an entire core/uncore, replace the entire core/uncore, or try to avoid the core/uncore in its entirety. At the other extreme, a core/uncore partitioned into dozens of modules would have a much larger fault dictionary with an associated reduced percentage of ideal accurate diagnosis results but would have more desirable recovery options available. That is, repairing, replacing, or avoiding a small portion of a defective core/uncore is more appealing than doing the same for the entire core/uncore. Additionally, when considering the replacement of defective modules, smaller and more numerous modules can handle more failures before running out of spare modules. Consider the situation where a single spare is reserved for each module of a core/uncore. If the core/uncore is partitioned into many modules, each can fail independently without reducing the overall functionality of the core/uncore. If there are only a few modules, however, it is more likely that subsequent defects will lie within an already replaced module.

VIII. CONCLUSIONS

The contributions of this paper are centered upon on-chip diagnosis and are three-fold. Firstly, we introduced a new enhancement to the uncertain-transition fault model (which we call TRAnsition-X or TRAX for short) that conservatively and efficiently captures all possible activation and propagation paths for a degraded standard cell. Secondly, several fault-dictionary compaction techniques have been described and investigated. While the most significant size reduction stems from using only pass/fail test responses (as opposed to full test responses), recognizing that diagnosis precision only needs to match the repair level also significantly contributes to dictionary-size reduction. For example, the pass/fail dictionary is further compacted by another 53% on average when employing repair-level resolution. Thirdly, the on-chip diagnosis hardware architecture provides an area- and performance-efficient methodology for the localization of detected failures to the optimal level necessary for repair or avoidance. For the OpenSPARC T2 processor, the area overhead is less than 0.16%. With regards to performance efficiency, the on-chip hardware architecture only affects execution during the test-application time. The on-chip diagnosis itself uses the stored test results and in the best-case scenario, runs in the background without affecting system execution.

ACKNOWLEDGMENT

The authors acknowledge Professor Subhashish Mitra and Dr. Yanjing Li from Stanford University for their many fruitful discussions of on-chip test, NBTI, ELF, and many other topics. Thanks to Harsh Shrivastava for his help gathering simulation data comparing TRAX and TF fault subsumption in Section III. Additional thanks to fellow PhD student Ben Niewenhouis for the suggestion of simplifying the diagnostic architecture by using a single comparator instead of one per module.

REFERENCES


Matthew Beckler (S’08-M’13) received the B.S. degree in computer engineering from the University of Minnesota, Minneapolis, MN, in 2008, and the M.S. and Ph.D. degrees in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, PA, in 2011 and 2017. His current research interests include the area of on-chip test and diagnosis, fault models, and computation acceleration with graphics processing units.

Matthew is a recipient of the Lamme/Westinghouse ECE Graduate Fellowship and the Bertucci Graduate Fellowship.

R.D. (Shawn) Blanton (S’93-M’95-SM’03-F’09) received the B.S. degree in engineering from Calvin College, Grand Rapids, MI, in 1987, the M.S. degree in electrical engineering from the University of Arizona, Tucson, in 1989, and the Ph.D. degree in computer science and engineering from the University of Michigan, Ann Arbor, in 1995. He is currently a Professor of electrical and computer engineering at Carnegie Mellon University, Pittsburgh, PA, where he also serves as leader of the Advanced Chip Testing Laboratory (ACTL). His current research interests include test and diagnosis of integrated heterogeneous systems.