Compacting Test Vector Sets via Strategic Use of Implications

Nuno Alves, Jennifer Dworak, Iris Bahar
Division of Engineering
Brown University
Providence, RI 02912

K. Nepal
Electrical Engineering Dept.
Bucknell University
Lewisburg, PA 17837

ABSTRACT

As the complexity of integrated circuits has increased, so has the need for improving testing efficiency. Unfortunately, the types of defects are also becoming more complex, which in turn makes simple approaches for testing inadequate. Using n-detect testing can improve detect coverage; however, this approach can greatly increase the test set size. In this proof-of-concept paper we investigate the use of logic implication checkers, inserted in hardware, as an aid in compacting n-detect test sets. We show that checker hardware with minimal area overhead can reduce test set size by up to 25%. In addition, this implication checker can serve a dual purpose for online error detection.

1. INTRODUCTION

As devices continue to scale to smaller feature sizes, the efficiency of testing has become of great importance. Testing resources are severely limited, and yet the amount of logic that must be tested with those resources has increased significantly, leading to corresponding increases in test data volume. Furthermore, the complexity of defects has increased as well, leading to greater difficulty in modeling and predicting defective behavior. Finally, unacceptable process variations and environmentally sensitive defects must also be detected, leading to increasing requirements for the number of tests that must be applied to detect them and the variety of conditions under which those tests must be applied.

As a result of all these issues, it is well-recognized that attempting to detect defects with structural test sets that merely achieve high single stuck-at fault coverage is highly inadequate. Previous work has shown that multiple detections of faults and multiple observations of internal circuit sites can greatly increase the number of unmodeled defects that are detected e.g. [8, 16]. For example, in experiments documented in [16], no defects escaped testing when at least 15 detections of each fault were achieved by patterns run at rated speed. These observations have led to the introduction of n-detect testing where a minimum of n detections are guaranteed for each fault. However, guaranteeing these additional detections causes test set lengths to increase by a factor of n for compact test sets [13]—leading to a corresponding increase in test data volume and test time. Furthermore, testing for delays and parametric effects under different environmental conditions is generally required, and these all add to the overall cost of test.

To help minimize test cost, different techniques to reduce requirements for test time, test data volume, and test data bandwidth have been proposed. Some of these techniques focus on reducing the number of patterns generated. For example, static compaction techniques compact test sets after they are produced by the ATPG engine. One of the simplest of these techniques involves simply fault simulating the test patterns in reverse order and dropping any patterns that do not detect any, as yet undetected faults. Because of the way test patterns are generated, most of the inefficient patterns are created at the beginning of the automatic test pattern generation (ATPG) process, and this reverse simulation allows them to be removed. Other static compaction techniques have included analysis of independent fault sets to develop lower bounds for test set size. A set of faults is independent if no two faults can be detected by the same vector [1]. In addition to static methods, other researchers have proposed dynamic compaction techniques that alter the ATPG flow. Once a pattern is generated for a target fault, if enough "don't cares" remain in the test cube, additional faults are targeted by the ATPG engine for detection by that same pattern.

Dynamic and static techniques have also been applied specifically to n-detect testing. For example, the authors of [15] developed a greedy algorithm that involved simulating all possible assignments to five circuit inputs in parallel. The number of guaranteed and potential fault detections was incorporated into a greed metric that was used to choose one of the 32 patterns as a final assignment for those five inputs. The process could then be repeated with additional inputs. Others have used integer linear programming techniques to reduce test set sizes for n-detect sets [11] and reduced overall test set size by about 18% for $n = 3$, but the percent improvement decreased as n increases.

An obvious alternative to applying ATPG patterns through scan that can significantly reduce tester resource requirements are pseudo-random patterns (such as those used in built-in self test (BIST)). Unfortunately, previous work that uses pseudo-random patterns to increase multiple detections has shown that while easy-to-detect faults require n times the original test length for n fault detections when pseudo-random patterns are used, difficult faults may require $n log n$ times the original test length [22].

Other researchers have investigated the ability to choose the right mix of structural patterns (ATPG and/or BIST) that target multiple types of fault models for at-speed testing [7]. They achieved a pattern reduction of 5%–35% and showed a positive effect on the n-detect coverage metric with their applied optimizations.

Additional work, targeted more generally, has also focused on finding ways to modify design for test hardware to allow for the

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ICCAD '09 San Jose, California USA
Copyright 2009 ACM 978-1-60558-800-1/09/11 ...$10.00.

more efficient application of ATPG or BIST vectors or a combination of the two [3, 4, 9, 14, 18, 20]. For example, [9] proposed reducing test application time by dividing the scan chain into multiple partitions and shifting in the same vector to each scan chain through a single scan-in input. The scan-out values can then be compacted and observed through a multiple input signature analyzer. In [12], an embedded processor in a system-on-chip is used to test other portions of the chip by loading compressed deterministic test data and executing a program to decompress the tests on-chip.

Test point (TP) insertion is another widely studied DFT method that aims to reduce test pattern count. Test points may be used to enhance the controllability or observability of the design. In the case of observation points, the testability of hard-to-detect faults is increased by the insertion of a branch, in a carefully selected location, which feeds its logic signal into a scan cell or a primary output. Some relatively recent work [5, 6, 21] is able to reduce the APTG test pattern count for large circuits by more than 35% on average with a single observation point every thousand gates.

In this paper, we provide a proof-of-concept demonstration of the potential benefit of using invariant relationships for test pattern count reduction. In the past, the use of logic implication checkers in hardware has been proposed for the purpose of online detection of errors at runtime [2,19]. These invariant relationships (i.e., logic implications) are determined by the functional behavior of the circuit and are always expected to hold whenever the circuit is free of errors. Both these works focus on the identification of these logic implications among circuit sites, either within a single cycle or across multiple cycles. Once identified, a small subset of these logic implications is selected to be included in the checker hardware. Violations of these expected relationships indicate that an error has occurred. Although detection of all possible errors due to stuck-at-faults is not possible through the implication checker alone, the approach does offer a mechanism for effectively trading off additional area overhead for additional fault coverage.

While the goal of [2, 19] was to use logic implication checkers as a means of online error detection and fault tolerance, this same implication checker hardware may serve another role. Specifically, it may be possible to reduce the number of test patterns required for an n-detect test pattern set with the help of an implication checker. Like test points, checker logic for these logic implications improves the observability of internal circuit sites, and thus the detectability of faults during test.

Furthermore, logic implications have some important advantages over test points. Unlike test points, the structure of the logic implication hardware may allow for the detection of errors without any *a priori* knowledge of the patterns applied and without having to determine what the internal circuit values should be for a given input combination or sequence. Thus, these same logic implications can also be used in functional testing without requiring gate-level simulation of the functional test sequence or storage of expected internal values. In addition, any difficult-to-detect defects that do manage to escape the testing process have the potential to be detected online with this checker hardware—allowing the checker hardware to serve a dual role.

In this paper, we will explore the use of logic implication checkers, inserted in hardware, as a means of compacting n-detect test sets. This paper makes the following contributions:

- We describe a methodology for identifying logic implications that may be targeted specifically for discovering defects that are hard-to-detect.
- We show that by adding checker hardware for only a few implications, the number of times hard-to-detect faults are detected can increase significantly.
- We show that even with only a 5% area overhead, the overall reduction in test patterns (i.e., the test compaction rate) is often more than 10% and may reach almost 25%.

2. METHODOLOGY

As outlined in the introduction, in this paper we explore additional uses for implication-based checker logic. In particular we are interested in studying how implications may be used as an aid to the manufacturing test portion of the VLSI design flow. Our methods focus on investigating how the implication-checking hardware may be specifically targeted for detecting "hard-to-detect" faults. Because many patterns in a test set are included solely to detect such faults, making some of these faults easier to detect (through the implication-based hardware) should significantly reduce pattern counts. Furthermore, defects that are hard-to-detect and are thus most likely to be missed during manufacturing test are likely to correspond to the least-observed sites and faults. Thus, when the test set size is strictly limited by cost requirements, increasing the detectability of such hard-to-detect faults should also improve test set quality. Our approach for checker hardware generation and test set compaction can be summarized as follows:

1. Discover all implications in a given circuit.
2. Extract a reduced set of implications that are not subsumed by other implications.
3. Generate an *n*-detect test set for a copy of the original circuit under test.
4. Using the generated *n*-detect test, create a fault dictionary for each individual implication inserted in the original circuit.
5. Apply an algorithm that scans fault detections associated with each implication and remove vectors no longer required to achieve n detections of each fault.
6. Report list of high quality implications that satisfy the user-specified hardware overhead constraints.

The discovery of logic implications and the extraction of a reduced implication set was discussed in [2, 19]. The novel contributions of this work lie in the final three steps, which apply the use of the circuit's invariant relationships to the reduction of input test patterns for an n-detect stuck-at-fault test set.

2.1 Finding Logic Implications

The process of discovering implications involving pairs of circuit sites begins with the good circuit simulation of a circuit using a set of random vectors. Potential implications are found by identifying "missing" combinations of the four possible logic assignments for each pair. For example, after logic simulation, if there never exists an instance where both circuit nodes $N1$ and $N2$ are equal to one, then a potential implication $(N1 = 1) \implies (N2 = 0)$ is said to exist. However, because we are not performing an exhaustive circuit simulation, it is possible that this relationship may not hold for some other unsimulated input combination. Thus, the validity of all implications must be confirmed through an exhaustive method. To accomplish this, we phrase the implication relationship as a satisfiability problem and pose it to a SAT solver such as ZChaff [17]. Only those implications confirmed by ZChaff are retained.

Even small circuits may contain thousands of these valid invariant relationships, but all of them are not equally effective for fault detection. For example, some implications may be *dominated* by other implications such that they only detect faults covered by other

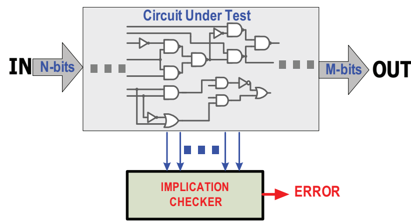

Figure 1: A circuit with implication checker hardware.

implications. These dominated implications may thus be removed from consideration—generating a *compressed set*. Even this compressed set of implications is generally much too large to enable all of them to be included in the checker hardware. In [2, 19], a subset of this compressed implication set was chosen to give the best overall coverage of all detectable circuit errors, subject to user-defined area overhead constraints.

In this work, we will begin by choosing an even smaller subset of the compressed implication set targeted specifically toward those faults that are *hardest* to detect. We will evaluate the impact that the corresponding checker hardware has on the number of patterns required to achieve an n-detect test set. Then we will explore the test set reduction that could be achieved with implications chosen for online error coverage instead.

2.2 A Motivational Example

We begin our analysis by investigating the increase in fault detectability for a small benchmark circuit. Specifically consider the combinational circuit *rd73*, from the MCNC91 benchmark suite, which we will use to motivate our approach. The original circuit has 7 inputs and 3 outputs. The small number of inputs allows us to run fault simulation on this circuit for all possible input patterns (i.e. 128 vectors) and all 700 uncollapsed stuck-at-faults. In the original circuit simulation of the 700 possible faults, 164 faults are observed at the outputs only once and 96 of the remaining faults are observed only twice. These faults are the hardest to detect by random fault simulation because they are less likely to be excited and propagated to an output. We then attach an implication hardware checker to this circuit in a setup similar to the one shown in Figure 1. The addition of the implication checker also adds one extra output to the circuit. As implications are added to the checker, some previously hard-to-detect faults may acquire an additional path (through the checker logic) through which they may be observed.

Table 1: Improvement in number of faults detected more than 5 times as checker logic is added to original *rd73* circuit netlist.

circuit	# of times faults detected				
	$N=1$	$N=2$	$N=3$	$N=4$	$N>=5$
Orig.	164	96	53	26	361
Orig.+1%imp	153	87	50	26	384
Orig.+2%imp	151	84	50	23	392
Orig.+3%imp	143	81	43	27	406
Orig.+4%imp	142	78	42	23	415
Orig.+5%imp	137	69	38	23	433

Table 1 shows the improvement in the number of times faults are detected as implication-based checker logic is added to the circuit. The first column lists the amount of area overhead allowed for the checker logic. Here, the overhead varies between 0% (original circuit with no checker logic inserted) to 5%. The subsequent columns list the number of faults detected a given number of times. For example, in the original circuit, 164 faults are detected exactly once by a test set consisting of all 128 possible patterns. However, with the addition of checker logic corresponding to an extra 5% hardware overhead, the number of fault sites detected exactly once is reduced to 137. Furthermore, although it is not apparent from this table, of the 27 faults that were detected exactly once previously, 10 of them became detectable for more than 5 patterns. The movement of faults from single-detect to a much larger multiple-detect category can obviously be beneficial when the goal of ATPG is to detect each fault multiple times.

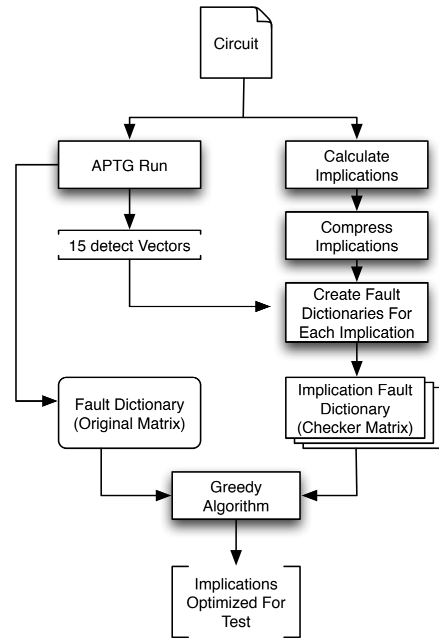

Figure 2: Flowchart of our approach.

2.3 Choosing an Implication Set

Figure 2 depicts the overall flow of our approach for choosing an optimal subset of implications suited for test pattern reduction. As shown on the rightmost path through the flowchart, we begin by generating an implication set and removing all implications dominated by others (and possibly other low quality implications as well) to form a compressed set of implications. This step was discussed in more detail in Section 2.1.

In parallel, an ATPG tool generates a 15-detect test set. In addition, we also compute a fault dictionary which reports the detectability of each fault for each pattern. This fault dictionary is a Boolean bi-dimensional matrix, which we call *Original Matrix*, with *number of detectable faults* $\times$ *number of 15-detect test patterns* elements. In our experiments, both the fault dictionary and the 15-detect set vectors were computed using Mentor Graphics FastScan (v8.2009.1).

At this point we have a test set and a set of candidate implications for the potential checker hardware. The next step consists of determining each implication's effect on the detectability of previously hard-to-detect faults. Specifically, a unique fault dictionary using the original ATPG test set is created for each candidate implication; a single dictionary captures the fault detection information due to a specific implication being added to the circuit. Thus, for each of the circuits described in this paper, we create *n* distinct fault dictionaries, where *n* is the number of candidate implications. The fault dictionary that characterizes the behavior of each implication is the

Checker Matrix. Note that whenever a new checker logic gate is added, we add new faults to the circuit. These new faults are due to the fan-out branch created by implication insertion. So every time an implication is added to the checker logic, rows corresponding to the new faults are added to the *Checker Matrix* associated with that implication. Once the *Checker Matrices* are generated, they are passed to a procedure that uses a greedy algorithm to find an optimal combination of implications that give the best reduction in the number of test patterns, for a specific hardware overhead. The procedure, called *ComputeBestMatrix*, is outlined in Figure 3.

```
procedure ComputeBestMatrix()
 1: bestMatrix ← OriginalMatrix
 2: #minPatterns ← #original vectors
 3: for i = 1 to #OverheadImplications do
 4:     for j = 1 to n do
 5:         testMatrix ← bestMatrix OR (Checker Matrix n)
 6:         #testPatterns ← CountMinPat(testMatrix)
 7:         if #minPatterns > #testPatterns then
 8:             #minPatterns ← #testPatterns
 9:             bestMatrix ← testMatrix
10:         end if
11:     end for
12: end for
```

Figure 3: Procedure *ComputeBestMatrix*

In essence, the procedure cycles through all the *Checker Matrix* files until the user specified hardware overhead of the checker logic (i.e. the implication budget) is reached. On each iteration, the algorithm searches for the implication that gives the best reduction in the test pattern count, and once found, the implication is added to the circuit's checker logic. The corresponding fault dictionary is then stored in *bestMatrix*. Following iterations are guaranteed to build upon this updated circuit as the algorithm performs an OR operation between each individual implication's *Checker Matrix* with the *bestMatrix*.

The procedure *ComputeBestMatrix* makes a call to another procedure, *CountMinPat*, which determines the minimum number of possible patterns for each *Checker Matrix*. The procedure *CountMinPat* is outlined in Figure 4. This procedure cycles sequentially through every single pattern, deleting those that are not required to maintain a minimum of 15 detections.

If the addition of implications brings an overall fault observability increase, then one may intuitively think that selecting the implications that cover the *hardest-to-detect* faults and running a state-of-the-art pattern generation algorithm will achieve the same effect as our proposed algorithms. Unfortunately, the Mentor Graphics FastScan pattern generator, while highly optimized for performance, occasionally generates a higher number of patterns upon the addition of extra checker logic. By using our proposed algorithm, we guarantee full experimental control and can therefore guarantee that all pattern count reductions are only due to the effect of the checker logic.

3. RESULTS

In this section, we evaluate the effectiveness of our approach by conducting several experiments using a number of circuits from the MCNC and ISCAS combinational benchmark suites. Specifically, we evaluate compaction rates, impact on delay, and effectiveness when combined with online error detection.

3.1 Impact of Implication Checker on N-Detect Test Patterns

In this section, we measure the effectiveness of our approach in terms of the number of *n*-detect test patterns required for circuits synthesized with and without the use of additional implication checker logic. In our experiments, we focused on the generation of a 15-detect test set to cover all detectable stuck-at faults in the circuit at least 15 times. As described in Section 2 and illustrated in the flow-chart of Figure 2, we used the FastScan tool from the Mentor Graphics suite to generate a list of uncollapsed faults for each of the circuits and to obtain the initial 15-detect test set to fully cover those faults. FastScan used standard compaction techniques when creating this test set. The number of test patterns required to cover the faults is shown in column 2 of Table 2. Using this test set as our starting point, we then ran experiments to measure how the addition of extra implications limited to a particular user-specified hardware overhead would impact the number of test-patterns required for the 15-detect test. The number of test patterns for different amounts of hardware overhead is shown in columns 3–7 of Table 2.

```
procedure CountMinPat(matrix)
 1: #reduced Patterns ← #patterns
 2: for k = 1 to #patterns do
 3:     delete pattern k
 4:     if #Detects < 15 then
 5:         restore pattern k
 6:     else
 7:         # reduced Pattern−−
 8:     end if
 9: end for
10: return #reduced Patterns
```

Figure 4: Procedure *CountMinPat*

Table 2: Number of required patterns for a 15-detect test set as a function of checker overhead.

Circuit	Original Netlist	Circuit With Added Implications				
		1%	2%	3%	4%	5%
c432	537	495	477	469	457	452
c499	793	793	793	793	793	793
c880	320	284	278	274	274	274
c1908	1699	1612	1582	1564	1564	1564
b12	375	362	349	342	336	331
clip	755	674	648	601	586	574
misex2	1220	1173	1144	1116	1086	1065
rd73	1057	938	896	862	829	800
Z5xp1	679	622	603	570	543	531
Z9sym	1115	1015	995	976	960	937

The average percent reduction in the number of test patterns is 8% across all benchmarks when checker logic corresponding to a 1% area overhead is added. With an area overhead of only 5%, an average reduction of 17% can be achieved. As can be seen in Figure 5 the reduction varies across different circuits. Circuits *clip*, *rd73* and *Z5xp1* show the most improvement while (aside from *c499*) *c1908* shows the least improvement with an overhead of 5%. A close analysis of the internal structure of the circuits gives a good indication as to why there was no reduction in test patterns for *c499* . The circuit *c499* is a single-error-correcting circuit with 41 inputs and 32 outputs [10]. Of the 202 gates comprising the circuit, 52% (104 gates) are two-input XOR gates. In a two-input XOR gate, any meaningful invariant relationship between the input and output nodes cannot be deduced by a simple pairwise comparison of the output node with one of the input nodes. If an output is at logic 0, it is possible to say that both the input nodes are at the same logic value but it is not possible to say if either input node is at a logic 0 or a logic 1 without referring to the second input node. Since all implication relationships used in this paper are simple pairwise relationships, such implications do not exist in a significant portion of

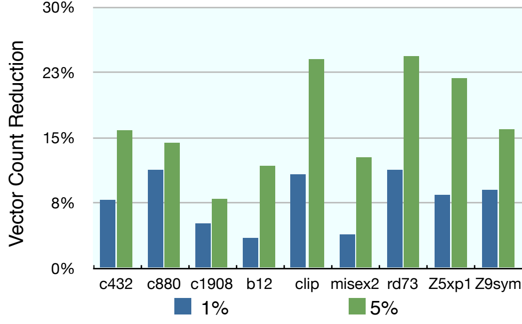

Figure 5: Vector count reduction for 1% and 5% hardware overheads.

the *c499* circuit. Unfortunately, the implications that do exist do not appear to be conducive to detecting the hardest to detect faults, and thus we see no benefit in terms of test pattern reduction for this circuit. Similarly, *c1908* is another 16-bit single-error-correcting and double-error-detecting (SEC/DED) circuit with 33 inputs and 25 outputs. The circuit contains a syndrome generator, and a large parity tree consisting of XOR structures. However, the circuit netlist we used for *c1908* was a flattened netlist with the XOR gates implemented with other logic gates. This alternate implementation appears to be at least somewhat more amenable to test pattern reduction. Finally, in some of the circuits of Table 2 (*c880* and *c1908*) we also see that after a certain hardware overhead, we stop seeing additional benefits from adding any extra implication checkers to the circuit. This suggests that there is no implication, that was not already added to the circuit, that is able to provide additional coverage of faults that were hard-to-detect with the generated pattern set.

3.2 Impact of Implication Checker on Critical Path Delay

In our approach discussed above, we did not re-synthesize the original logic of the circuit and tried to preserve the critical path. Thus, the checker logic is added to the original circuit and runs in parallel with the regular logic. However, the addition of extra fanout at nodes where implications are added can increase the delay of a critical path or can even alter which path is most critical. Given this observation, we next measured the impact of adding checker logic corresponding to a 5% area overhead on the critical path delay of the circuit. The original circuit netlist was analyzed with Mentor Graphics software, and the delay of the top four critical paths was reported. In Table 3 we show the original critical path and its associated delay computed using Mentor's static timing analysis tool. The implication checker hardware was then synthesized and added to the original circuit netlist. The critical paths of the new netlist were then analyzed. In all the circuits tested, the critical path of the original circuit was preserved although an average of 2% increase in delay was seen. The results are summarized in the Table 3.

The second column of Table 3 lists the critical path of the circuit before the addition of the checker logic. The delay results for the original circuit and the one with an additional checker logic with a 5% overhead are shown in columns 3 and 4 respectively. Note that the delay of the critical path in some circuits remains unchanged (e.g. *b12*, *clip*, *mixex2*). In the case of *b12*, the added implications increased the delay of the third most critical path (from input X00 to output Z6) from 0.45 to 0.47 but left the delay of the first and the second most critical paths intact. This increase in the delay was still not able to make the third path the most critical because the original critical path had a delay of 0.53. These results regarding the addition of 5% implication overhead are encouraging because they show that the critical path delay for the circuits remains relatively unchanged.

Table 3: Change in critical path delay when using checker logic.

Circuit	Critical Path	Orig. Delay (ns)	New Delay (ns)	% Change
c432	N102 → N421	3.21	3.25	1.2
c499	N125 → N731	2.49	2.51	0.8
c880	N51 → N878	2.15	2.22	3.3
c1908	N43 → N2899	3.29	3.44	4.5
b12	X03 → Z6	0.53	0.53	0.0
clip	X4 → Z1	0.89	0.89	0.0
misex2	B → I1	0.69	0.69	0.0
rd73	X6 → Z0	0.97	1.02	5.1
Z5xp1	X2 → Z2	1.05	1.08	2.8
Z9sym	X2 → z0	1.02	1.04	2.0

3.3 Online Error Detection and Test-Pattern Reduction

The results presented so far have been obtained using implications with a certain area overhead derived from the *original* compressed implication set to aid in the manufacturing test step of the integrated circuit design flow. The implications that we have used have been selected based on their ability to increase the observation of previously hard-to-detect faults. Unfortunately, the compaction results obtained for the circuits studied did not achieve the same reductions reported for some of the other methods listed in Section 1, including test point insertion on large circuits. However, an important advantage of the proposed method is that implications may not only contribute to test reduction, but to online error detection as well. Our previous work presented in [19] originally proposed using implications for online error detection. Implications were added in increments of 10% area overhead allowing the runtime error coverage to be improved. We now investigate whether implications that were optimized for online error detection can also be used effectively to reduce the number of patterns required for a 15-detect stuck-at fault test set during the manufacturing test process. Table 4 summarizes the results obtained for an overhead of 10% and 20%.

The second column of Table 4 lists the number of patterns in the

Table 4: Number of required patterns for a 15-detect test set as a function of checker overhead using implications optimized for online error detection. The probability of undetected faults (Prob(undetected)) were obtained from [19].

Circuit	Circuit With Added Implications				
	Number of Test Patterns			Probability Undetected [19]	
	Original	10%	20%	10%	20%
c432	537	492	462	7.2	4.5
c499	793	793	793	16.9	13.7
c880	320	297	297	22.6	19.1
c1908	1699	1581	1581	14.3	13.1
b12	375	363	362	12.3	9.7
clip	755	736	712	11	10.2
misex2	1220	1216	1216	12.8	9.5
rd73	1057	1007	947	7.2	6.3
Z5xp1	679	661	624	14.5	11.5
Z9sym	1115	1016	1010	5.6	5.2

15-detect test set generated for the original circuit netlist. These are the same as those reported in Table 2. The number of patterns required using implications optimized for online error detection are presented in columns 3 and 4 for an area overhead of 10% and 20% respectively. These implications were directly obtained from [19]. Notice that the reduction in patterns obtained for the 20% case is not as good as the 5% overhead case from Table 2 since these implications were optimized for runtime error detection and not for pattern reduction. The most encouraging results come from 10% hardware overhead, where our greedy algorithm reported around 8% pattern count reduction for *c432*, *Z9sym* and *c1908*. Also encouraging is that some of the lowest performing circuits, *clip* and *Z5xp1* still managed to get around 2.5% pattern count reduction. Obviously circuit *c499* did not see any reduction in the number of patterns, since there was no optimal combination of pattern reducing implications, as discussed in Section 3.1. The work presented in [19] used these implications for runtime error detection and had used the probability of undetected online faults as a metric to measure the effectiveness of their checker hardware. Columns 5 and 6 present the probability that an error at runtime will go undetected by the same set of implications. Despite increased hardware overhead, the reductions in test patterns are very encouraging considering that the implications used in this particular experiment were not selected based on their propensity to reduce the number of vectors. This indicates that we may be able to optimize our implication set based on these two parameters (error detection at runtime and pattern count reduction) jointly.

4. CONCLUSIONS

In this paper we have investigated the use of logic implications for test set reduction. We have presented a method for finding and choosing a set of implications to be incorporated in checker logic based upon its ability to make the hardest-to-detect faults within a circuit more observable. While the exact amount of reduction varies, for a 15-detect test set, our analysis has shown that with only a 1% area overhead we can obtain test set size reduction of approximately 8% on average. Furthermore, with a 5% area overhead, more than a 25% reduction is possible. We also investigated the impact of the addition of implication logic on circuit delay. While an average estimated increase in the delay of the longest path of 2% was observed, in several cases, no change in the length of the longest path occurred.

Although this work focused on choosing implications for test set compaction, previous work has chosen implications for online error detection. The faults ideally targeted in each instance are different. In the case of manufacturing test, hard-to-detect faults should be covered, while in the case of online error detection, errors that are likely to propagate to an output are generally targeted. As a result of these differences in focus, we also investigated the ability of implications chosen for online error detection to reduce test set size. Although the reduction was not as great as could be achieved with implications targeted toward manufacturing test, encouraging reduction in test lengths did occur for many circuits. This suggests that implication hardware in general may perform a dual role of both reducing test set lengths and subsequently detecting defects that escape test or transient faults that appear in the field.

Indeed, we expect that by better balancing both manufacturing test and online error detection when selecting implications to include in the checker logic, overall better results can be obtained. Also, in addition to area overhead, the estimated increase in circuit delay could be used as an additional constraint for the creation of the checker logic. In fact, a multi-focused analysis is possible—constraining the type and number of implications chosen to reduce power, delay, and area while targeting both manufacturing test and online error detection with different degrees of importance.

We also believe that sequential circuits may benefit significantly from the approach presented in this paper. In particular, cross-cycle implications that were used in [2] to improve error detection rates, can also be used to reduce test set lengths by providing multiple new observations points in the circuit. Future work will focus on evaluating this approach more thoroughly for sequential circuits.

5. REFERENCES

[1] S. Akers, C. Joseph, and B. Krishnamurthy. On the role of independent fault sets in the generation of minimal test sets. In *ITC*, pages 1100–1107, 1987.

[2] N. Alves, K. Nepal, J. Dworak, and R. I. Bahar. Detecting errors using multi-cycle invariance information. In *DATE*, April 2009.

[3] C. Barnhart, V. Brunkhorst, F. Distler, O. Farnsworth, A. Ferko, B. Keller, D. Scott, and B. Koenemann. Expanding opmisr beyond 10x scan test efficiency. *IEEE Design and Test of Computers*, pages 65–73, 2002.

[4] D. Das and N. A. Touba. Reducing test data volume using external/lbist hybrid test patterns. In *ITC*, pages 115–122, 2000.

[5] M. Geuzebroek, J. van der Linden, and A. van de Goor. Test point insertion for compact test sets. In *ITC*, 2000.

[6] M. J. Geuzebroek, J. T. van der Linden, and A. J. van de Goor. Test point insertion that facilitates atpg in reducing test time and data volume. *ITC*, 2002.

[7] S. Goel and R. Parekhji. Choosing the right mix of at-speed structural test patterns: Comparisons in pattern volume reduction and fault detection efficiency. In *14th Asian Test Symposium*, Dec. 2005.

[8] M. R. Grimaila, S. Lee, J. Dworak, K. M. Butler, B. Stewart, H. Balachandran, B. Houchins, V. Mathur, J. Park, L.-C. Wang, and M. R. Mercer. REDO—random excitation and deterministic observation. In *VTS*, pages 268–274, 1999.

[9] I. Hamzaoglu and J. Patel. Reducing test application time for full scan embedded cores. In *29th Int Symposium on Fault-Tolerant Computing*, pages 260–267, 1999.

[10] M. Hansen, H. Yalcin, and J. P. Hayes. Unveiling the ISCAS-85 benchmarks: A case study in reverse engineering. *IEEE Design & Test*, 16:72–80, 1999.

[11] Y. Huang. On n-detect pattern set optimization. In *ISQED*, 2006.

[12] A. Jas and N. Touba. Using an embedded processor for efficient deterministic testing of systems-on-a-chip. In *ICCD*, page 418, 1999.

[13] K. R. Kantipudi. On the size and generation of minimal n-detection tests. In *International Conference on VLSI Design*, 2006.

[14] B. Koenemann, C. Barnhart, B. Keller, T. Snethen, O. Farnsworth, and D. Wheater. A smartbist variant with guaranteed encoding. In *10th Asian Test Symposium*, pages 325–330, 2001.

[15] S. Lee, B. Cobb, J. Dworak, M. R. Grimaila, and M. R. Mercer. A new atpg algorithm to limit test set size and achieve multiple detections of all faults. In *DATE*, 94-101, 2002.

[16] S. C. Ma, P. Franco, and E. J. McClusky. An experimental chip to evaluate test techniques: experiment results. In *ITC*, 1995.

[17] Y. Mahajan, Z. Fu, and S. Malik. Zchaff2004: An efficient sat solver. *Lecture Notes in Computer Science*, 3542:360–375, 2005.

[18] F. Muradali, V. Agarwal, and B. Nadeau-Dostie. A new procedure for weighted random built-in self-test. In *ITC*, pages 660–669, Sep 1990.

[19] K. Nepal, N. Alves, J. Dworak, and R. I. Bahar. Using implications for online error detection. In *ITC*, October 2008.

[20] J. Rajski, M. Kassab, N. Mukherjee, N. Tamarapalli, J. Tyszer, and J. Qian. Embedded deterministic test for low-cost manufacturing. *IEEE Des. Test*, 20(5):58–66, 2003.

[21] S. Remersaro, J. Rajski, T. Rinderknecht, S. M. Reddy, and I. Pomeranz. Atpg heuristics dependant observation point insertion for enhanced compaction and data volume reduction. *DFT*, 0:385–393, 2008.

[22] C.-W. Tseng, S. Mitra, S. Davidson, and E. J. McCluskey. An evaluation of pseudo random testing for detecting real defects. In *19th VLSI Test Symposium*, pages 404–409, 2001.