

Improving the Testability and Reliability of Sequential Circuits with Invariant Logic

N. Alves*, K. Nepal†, J. Dworak*, R. I. Bahar*

*Division of Engineering, Brown University, Providence, RI 02906

†Electrical Engineering Department, Bucknell University, Lewisburg, PA 17837

ABSTRACT

In this paper, we investigate dual applications for logic implications, which can provide both online error detection capabilities and improve the testing efficiency of an integrated circuit. These logic implications are implemented in hardware and help to verify that expected invariant circuit relationships are satisfied during field operation. Thus, any implication violation will indicate the presence of an error due to some faulty circuit behavior. In addition, checking these logic implications in hardware will create additional circuit outputs, which may be useful for compacting n -detect test sets. Our results show that logic implications can provide significant error detection and test pattern count reduction with very limited hardware overhead.

1. INTRODUCTION

For the past few decades, with Gordon Moore's prediction as the motivating force, transistor dimensions have steadily downscaled. Smaller dimensions have helped the number of transistors on a single chip to increase, allowing for a higher packing density, better circuit performance, and lower cost-of-production. As transistor dimensions move into the nanometer range, a number of reliability issues arise in the form of permanent defects and transient faults. Making sure that integrated circuits containing defects do not enter the marketplace requires extensive testing of each individual chip after manufacture. The resources available for testing these circuits – including the test time, tester memory and tester bandwidth – are limited. As such, a mechanism able to reduce the burden on manufacture defect testing by minimizing the amount of test patterns needed is highly desirable. In this paper, we describe a method of test-pattern reduction which has the added benefit of detecting errors that manifest themselves during normal circuit operation.

Online or runtime error detection is a well-studied area where different techniques are used to monitor the deviation of a circuit from its normal operating behavior. An ideal online error detection scheme will detect all errors without requiring any significant modifications to the circuit and will not negatively impact performance, power, or area overhead. Unfortunately, no scheme is ideal. Common error detecting and correcting codes such as parity, Hamming,

cyclic redundancy, Berger, Bose-Lin, residue and other arithmetic codes have been successfully used in the detection of errors in both memory and execution units. Readers are referred to [1] for a detailed comparison of a number of these error codes. Error detection in processors can also be achieved by simply duplicating certain execution units and comparing their results [2]. To obtain error correction in addition to detection, triple modular redundancy (TMR) can be used [3]. However, as stated previously, an ideal online error detection scheme would have little impact on the area overhead. Thus some methods have tried to find the right balance between full error coverage and hardware overhead by only replicating certain sections of the circuitry determined to be the most vulnerable [4,5].

Other online error detection schemes include Built-In Concurrent Self Test (BICST) [6] and Reduced Observation Width Replication (ROWR) [7] where prediction hardware is added to guess the appropriate response to a set of pre-computed test vectors. Both approaches have longer detection latency and do not guarantee immediate detection of a targeted fault on its first appearance. Techniques using high-level assertions that encode some checkable property of the circuit identified during functional verification have also been proposed [8–11]. Finally in [12] and [13], the authors incorporated gate level invariant relationships identified through logic implications into checker hardware for the purpose of error detection. It was shown that logic implications can be identified without knowing the functionality of the circuit and can be used to provide excellent error coverage at a reduced hardware overhead in both combinational and sequential circuits.

In addition to *runtime* error detection, a number of algorithms focus on detecting defects during the manufacturing test process. Manufacturing test is generally expensive, and to help minimize the test cost, numerous techniques have been proposed that reduce the test set size, test data volume and test application time. For example [14, 15] try to reduce test pattern counts using static compaction algorithms. These techniques first create a list of patterns and try to compact them using vector omission/regeneration until the new set maintains the coverage of the original set. Genetic algorithms have also been used for dynamic compaction of test patterns [16]. The original patterns generated with ATPG were used as seeds in a genetic algorithm, and better patterns were evolved that detected more faults.

Built-in self test (BIST) is another popular alternative that focuses on eliminating the need to acquire high-end testers instead of test-set compaction. Pseudo-random patterns are generated and applied within the circuit using additional on-chip test circuits. Test point (TP) insertion is another widely studied method that aims to reduce pattern count. By inserting observation or control points at certain locations within the circuit, certain faults that were previously less testable because of their low observability or controlla-

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.

bility can be detected more easily. It has been shown that by applying observation points every thousand gates, the test pattern count for large circuits can be reduced on average by one-third [17, 18].

While these numerous techniques have their own merits in terms of online error detection or manufacturing test, very little effort has been made in the direction of combining these techniques so that additional hardware added for one might benefit the other. More recently [19] proposed the use of logical implications for test pattern reduction in combinational circuits instead of online error detection. The authors showed that by adding external hardware circuitry to check for violations of the invariant relationships, the test pattern count could be reduced on average by 17% for a 15-detect test-set. The greedy approach presented by the authors was memory intensive because it required that a *fault dictionary*, a boolean bi-dimensional matrix of faults and patterns, be created for every implication under consideration. The authors also limited their analysis to combinational circuits and did not explain how the approach might be used for sequential circuits. It is possible use their implication discovery technique to find implications within a clock cycle and to use those implications for test-pattern reduction or online error detection in a sequential circuit. However, invariant relationships between nodes across the latch boundaries in different clock-cycles do exist in sequential circuits [13]. These implications across latch boundaries provide us with two observation points per implication, and this is likely to be helpful for test-set compaction. Using this hypothesis, we not only expand on the work of [19] and consider sequential circuits but also show that monitoring invariant relationships across latch/flip-flop boundaries and multiple clock cycles can both significantly improve error detection rates and contribute the most to the reduction in the number of test-patterns. We also present a heuristic method for selecting the most useful of these relationships to be included in our error-checking logic optimized for runtime error detection/test-pattern reduction. We show that while certain relationships help with the detection of errors at runtime and others help with the reduction in test-pattern count, we can create a hybrid set through which both online error detection and pattern count reduction can be achieved with limited area overhead. Finally, we also manage the excessive memory requirements of [19] by eliminating the need for a multiple separate fault-dictionary and using a commercial ATPG tool for greedy implication analysis and selection.

2. BACKGROUND

Throughout our work we will be discussing the implementation of logic implications. A **logic implication** describes the invariant relationship between two nodes within a circuit. For example, in a two-input AND gate, a *logic one* at output C implies that input A is also a *logic one* (i.e. $(C=1) \implies (A=1)$ is a logic implication). These logic implications can be found across multiple levels of logic gates in a circuit. In a sequential circuit, there can be two types of implications: cross-cycle and single-cycle. A **cross-cycle** implication, is an implication that appears across multiple clock cycles. For example, the value of a circuit site at time t may imply a value at another circuit site (or even the same circuit site) at time $(t + n)$, where $n \geq 1$. A **single-cycle** implication is one that appears between two nodes within the same clock cycle.

The logic implication checking hardware proposed by the authors of [12, 13] has the potential for being useful in helping to mitigate the problem of difficult-to-detect defects. Specifically, those defects that are the most difficult to detect during test are often located deep within the circuit interior. Easy-to-detect defects tend to be closer to the primary outputs and flip-flops. At the same time, results from [12, 13] indicate that faults deep in the circuit interior are

most likely to be covered by one of the single-cycle circuit implications, while faults near the POs are much less likely to be covered. As a result, the logic implication checking hardware is most likely to be able to cover exactly those defects that were missed during manufacturing test and thus must be detected online.

In addition to runtime detection of faults, the checker hardware inserted into the design for online error detection may be useful during manufacturing test as well. Each implication we insert can essentially serve as a new test point, which will increase the observability of the logic covered by that implication. Thus, defects located in that portion of the circuit will be easier to observe and to detect during test than they would be otherwise. This paper analyzes the impact of this additional detectability on both the defect levels and on the online error rates due to missed defects.

3. METHODOLOGY

As described in section 1, invariant relationships in circuits have been proposed and extensively studied by others. In this paper, we expand on the state of the art by demonstrating how logic implications can be used for test-pattern reduction while providing online error detection in sequential circuits. Our approach, outlined in Figure 1, can be divided into four steps:

1. Perform good circuit simulation on a gate level circuit description. Determine all valid logic implications, followed by the removal of the weakest, via a compression algorithm.
2. Perform fault simulation to generate two distinct optimized sets of implications; one for online error detection, other for test pattern count reduction.
3. Interleave these two sets of implications, effectively creating an hybrid set of implications.
4. Compute the *Probability of Error Detection* and test pattern reduction for each implication set.

We will now proceed to describe in detail the implementation of each of these steps.

3.1 Finding, Validating and Compressing Logic Implications

Our implication discovery process begins with the insertion of a gate level circuit netlist into a good circuit simulation engine. This allows us to find the logic value of each circuit site for each set of input vectors. For our good circuit simulations, we use a set of 32k random vectors. We then find potential implications by performing pairwise compares between all circuit sites, to determine if a logic implication exists between them. For example, after logic simulation, if for every instance when circuit node N1 equals *logic value 1*, circuit node N2 is equal to *logic value 0*, then a potential implication $(N1 = 1) \implies (N2 = 0)$ is said to exist. We call it a *potential* implication because, as we performed good circuit simulation with only a subset of all possible vectors, it is entirely possible that an implication relationship may not hold for some other un-simulated input combination. Thus, the validity of all implications must be confirmed. This can be accomplished by phrasing the implication relationship as a satisfiability problem and posing it to a SAT solver.

While this approach to find and validate implications requires no modification to the gate level circuit description of a combinational circuit, in order for it to work in sequential circuits, the feedback loop must be removed. This can easily be done by applying the time-frame expansion method [20], with the transformation of the $(t + 1)$ FF output into a primary output, and the t FF input into a primary input.

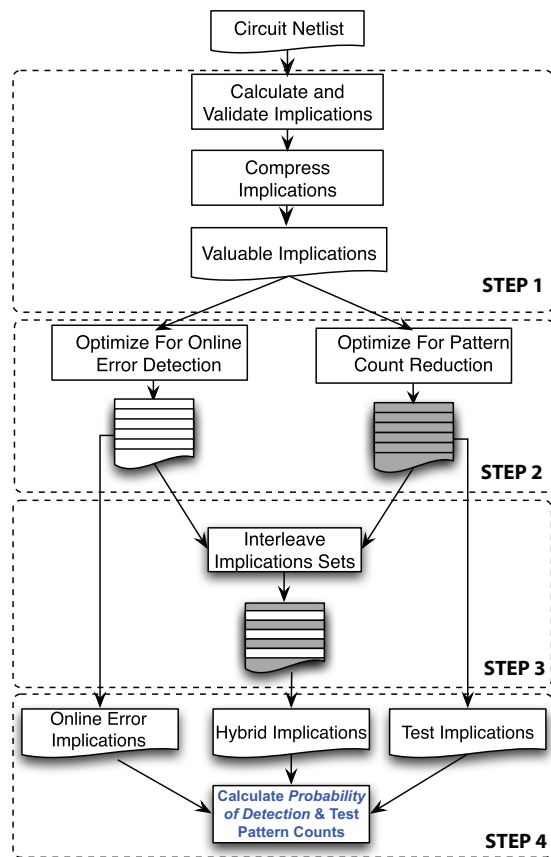


Figure 1: Our proposed flow.

Even a relatively small circuit may contain thousands of these valid invariant relationships, where not all of them are equally effective. The compression algorithm scans the list of validated implications and removes weak implications. These include implications entirely contained within other implications in terms of their fault detection capabilities and implications across a single gate. An efficient implementation of these compression algorithms is extensively discussed in [12]. Upon its successful completion, we are left with a set of the most useful invariant relationships. This implication set termed *valuable implications* will be the starting point for subsequent steps.

As outlined in step 2 of Figure 1, we send this recently discovered set of implications into two distinct optimization algorithms: one for online error detection and another for test pattern count reduction.

3.2 Optimizing Implications for Online Error Detection

The goal of the algorithm that optimizes the set of *valuable implications* for online error detection is to take the list of input implications and rank them by their ability to detect random errors. In essence, we want an implication closer to the top of the same list to be more effective at detecting errors than implications located towards the end of the same list.

The procedure begins with the insertion of all *valuable implications* into the circuit under test. This circuit is then subjected to fault simulation, with the stuck-at-fault model, stimulated by 32k random input vectors. The result of this fault simulation will tell

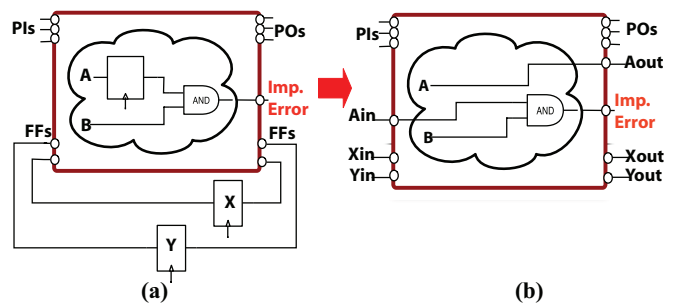


Figure 2: Circuit with cross-cycle implication $A(t-1)=1 \Rightarrow B(t)=0$ converted from (a) normal mode to (b) scan mode.

us what implications are able to detect which faults, and how many times each fault is detected. Once all faults are processed, we sort the implications in descending order according to number of fault detections with which they are associated. This procedure does not explicitly favor one fault over another; its sole purpose is to quickly process data and disregard redundant implications that cover faults better covered by other implications. These ideas can be implemented with the optimization algorithm described in [12].

3.3 Optimizing Implications for Test Pattern Count Reduction

While the implications optimized for online error detection will naturally tend to detect the most observable faults, implications best suited for test pattern count reduction need to cover the hardest-to-detect faults. Due to the different characteristics of these two set of faults, we expect little overlap between the two sets of implications. This naturally implies that a set of implications optimized for online error detection ought to be remarkably different from a set of implications optimized for test.

The idea behind optimizing implications for test is simple. As described in the section 2, implications can ease the observation requirements for certain hard-to-detect faults. If previously hard-to-detect faults become easier to detect, it implies that fewer test vectors may be required to fully test the circuit.

Cross-cycle implications in sequential circuits benefit from the fact that both the flip-flop needed for the implication and the implication output itself may be placed on the scan chain. When a scan chain is used to test sequential circuits, we are able to set/reset the state of each individual flip flop. By doing so, we can arbitrarily set the values of each flip flop at the beginning of the test, while also providing additional observation points to the circuit. With the additional observation points added by cross-cycle implications, even fewer vectors should be required to fully test the circuit.

For example, Figure 2 shows an example of a circuit, with a cross-cycle implication, in normal and scan modes. Under scan mode, we transform each flip-flop into a primary input/output. As a result, both sites involved in the implication become easily observable during test. Thus, the algorithm that optimizes the implication set for test will assume that the circuit will be tested in scan mode.

Starting with the set of *valuable implications*, we proceed to add one implication at a time to the circuit under test, and calculate the required number of 15-detect vectors using state of the art ATPG algorithms, such as FastScan. This n-detect requirement was selected based on [21], which found that no defects were missed when at least 15 detections of every fault were guaranteed during test and the patterns were run at rated speed. When the 15-detect vector count has been computed for each implication, we add the impli-

cation with the largest vector count reduction to the circuit, and repeat the process. This approach takes very little time with commercial ATPG algorithms. Our algorithm terminates when there are no more implications that can provide further test pattern reduction.

3.4 Hybrid Set of Implications

As previously stated, the goal of this paper is to find a set of implications that are equally useful for both online error detection and test. Since we know that the outcome of each of described optimization algorithms is a sorted list of implications optimized for a given function, it is natural to expect that combining both lists would grant the circuit characteristics of each. We create a new hybrid implication set with a combination of the set of implications optimized for test and the set of implications optimized for online error detection. In essence, we interleave the implications from each set, one at a time, and combine them onto a single list. Since some implications may be present in both sets, we make sure that any new implication to be added has not been previously included.

3.5 Determining Implication Performance

To evaluate the performance of an implication set we utilize two metrics: *Probability of detection* and *15-detect vector count*.

The *Probability of detection* will state how amenable a particular set of implications is for online error detection. It is calculated by initially performing stuck-at-fault simulation, with 32k random input vectors, and then counting how often an internal fault that is observable at one of the circuit’s outputs is detected and flagged by the implication checker hardware. More formally, the *Probability of detection* is defined as,

$$\sum_{i=1}^{\text{num_faults}} \sum_{j=1}^{\text{num_vectors}} \left(\frac{(F_{i,j} \wedge E_{i,j})}{(F_{i,j} \wedge E_{i,j}) + (F_{i,j} \wedge \neg E_{i,j})} \right) \quad (3.5.1)$$

where,

$$F_{i,j} := \begin{cases} 1 & \text{Fault } i \text{ is propagated to an output when vector } j \text{ is applied} \\ 0 & \text{Else} \end{cases}$$

$$E_{i,j} := \begin{cases} 1 & \text{Checker logic violated} \Rightarrow F_{i,j} \\ 0 & \text{Else} \end{cases}$$

The *15-detect vector count* metric will simply state how amenable the circuit is for test. This is accomplished by calculating the number of ATPG vectors required to detect each fault in the circuit at least 15 different times.

4. RESULTS

We ran experiments with a number sequential circuits from the ISCAS89 benchmark suite to validate the effectiveness of our approach. For each circuit, all *valuable implications* were calculated and validated using ZChaff [22], following the procedure outlined in Section 3.1. Starting with the superset of *valuable implications*, three distinct sets of implications were generated, and each was optimized with a particular goal: online error detection, test pattern compression, or a combination of both (hybrid).

4.1 Hardware Overheads

Our experiments indicate that implementing all implications in hardware, from a given optimized set, could easily double or triple the size of the circuit. In order to make our analysis more realistic, we decided to constrain the number of selected implications with

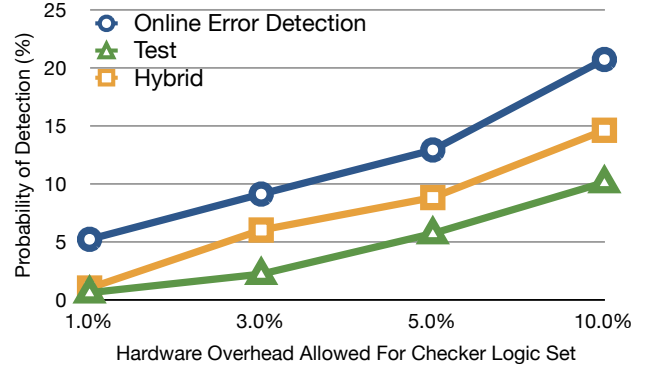


Figure 3: Average Probability of Detection for different subsets of implications.

respect to the area of the original circuit. For our experiments, we approximated hardware overhead as the total number of extra gates added, compared to the original gate count of the circuit, where each single-cycle implication check is assumed to be implemented using a single gate, with an additional latch for cross-cycle implications. This allows implications to be chosen without explicitly generating the layout. Layouts done using a TSMC 180nm library and the Mentor Graphics Toolset show that the area approximated using this approach was within 3% of the estimated area for 10% overhead even when routing and other issues were considered.

4.2 Probability of Detection

First, we determined the effectiveness of each optimized implication set for online error detection. We simulated each optimized set of implications and computed each set’s *Probability of detection* as shown in Eqn. (3.5.1).

These simulation results, reported in Table 1, provide us with interesting data. First of all, the simulations confirm our hypothesis from Section 3.3 that implications optimized for test, which target the hardest-to-detect faults, perform poorly compared to the other optimized implication sets in terms of error detection (especially in the cases with very low hardware overhead). However, despite being the worst performer, this set of implications optimized for test still had some noteworthy online error detection properties for certain circuits. For example, for a test optimized checker logic that added a 10% overhead to the original area of s1238, the *probability of detection* was 26%, a mere 2% reduction compared to the online error detection optimized set. Secondly this experiment confirms and expands previous efforts [13], by showing that when an implication set is optimized for online error detection, significant error coverage can be achieved even with a very low area overhead. This is very encouraging and provides a circuit designer an opportunity to make intelligent trade-offs between reliability and circuit area.

Furthermore, the results obtained from the hybrid implication set are extremely encouraging. As hypothesized in Section 3.4, by performing an alternating combination of the two implication sets, we were able to capture some of the characteristic behavior from each of the sets. This result is summarized in Figure 3. The performance of the hybrid set lies roughly between the other two sets of optimized implications for all hardware overheads.

It is worth pointing out that, based on results of Table 1, for certain circuits and hardware overheads, the *probability of detection* for the hybrid set of implications and test set of implications are the same. This is a direct consequence of our algorithm that

circuit	Online Error				Test				Hybrid			
	1%	3%	5%	10%	1%	3%	5%	10%	1%	3%	5%	10%
s298	10.5%	13.7%	16.4%	25.3%	0.3%	0.3%	0.4%	0.4%	0.3%	10.7%	14.0%	14.6%
s344	2.5%	2.5%	7.0%	15.1%	0.1%	0.2%	0.3%	1.6%	0.1%	2.6%	2.7%	7.5%
s349	4.0%	6.2%	8.1%	13.2%	0.2%	0.3%	0.4%	2.5%	0.2%	4.1%	4.3%	8.5%
s382	8.1%	13.7%	16.4%	19.6%	0.3%	0.9%	4.3%	8.2%	0.3%	8.5%	9.0%	20.1%
s526	1.8%	4.3%	7.6%	18.4%	0.1%	0.2%	1.1%	6.5%	0.1%	1.9%	4.5%	9.5%
s1196	5.4%	9.1%	13.0%	20.7%	1.7%	3.5%	11.7%	18.2%	2.5%	8.6%	11.7%	18.2%
s1238	4.1%	11.5%	17.2%	28.0%	1.1%	5.2%	13.3%	26.0%	2.0%	7.8%	13.3%	26.0%
s1488	4.9%	12.0%	17.6%	24.9%	1.0%	6.4%	14.7%	17.1%	2.7%	7.0%	14.7%	17.1%
AVG	5.2%	9.1%	12.9%	20.7%	0.6%	2.2%	5.7%	10.1%	1.0%	6.0%	8.8%	14.6%

Table 1: Probability of detection with implications from different optimized sets for various hardware overheads.

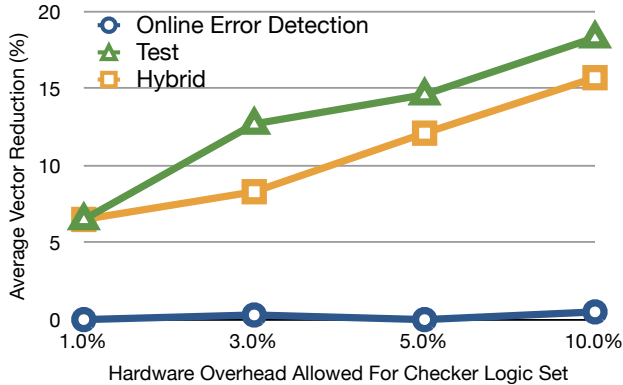


Figure 4: Average vector count reduction for different subsets of implications.

optimizes implications for test pattern reduction. As described in Section 3.3, there is a point where no unused implications are able to provide additional test pattern count reduction. When this situation occurs, our hybrid optimization algorithm simply appends the remaining online error detection implications to the hybrid set. This implies that, for certain hardware overheads, the set of hybrid and test implications are identical. The 10% hardware overhead of circuit s1488 provides an example of this.

4.3 Test pattern count

Table 2 shows the 15-detect vector counts for the different implication sets and hardware overheads, as determined by Mentor Graphic’s FastScan tool. As expected, the set of implications optimized for test perform much better than those optimized for online error detection in terms of test vector reduction. In fact, there are very few occurrences where implications optimized for online error detection provide a significant test pattern reduction. Extremely encouraging is the behavior of the hybrid implication set, which previously demonstrated promising online error detection capabilities, and now shows consistent vector count reduction as well. In fact, vector count reduction for the hybrid set closely follows the trend of the set of implications optimized for test.

Figure 4 shows the average test pattern count reduction for all the simulated benchmarks. With just 10% hardware overhead, we show an average reduction in test pattern count by 18.3% with using implications optimized only for test. However the true added value of our approach is in the use of the hybrid set of implications. Using this set, with a relatively small impact on test pattern count, we are

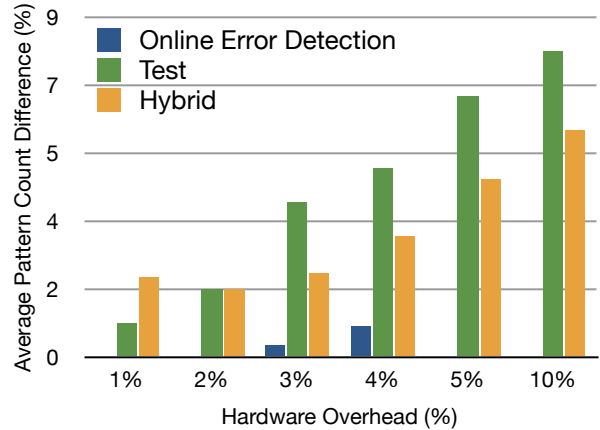


Figure 5: Average improvement in pattern count by using cross-cycle implications over single-cycle implications.

able to simultaneously improve the circuits’ ability to detect online errors dramatically. Allowing for a hardware overhead of 10%, implications from the hybrid set yield a test pattern count reduction of 15.7%, while providing an average probability of detection of 15.2%.

4.4 Testability of implications

The other major benefit of our approach compared to [19], is the testability of sequential logic containing cross-cycle implications, as described in Section 3.3. Since the scan chain mode provides additional observation points to the circuit for each implemented flip-flop, we tried to quantify the impact of cross-cycle implications on vector count reduction. For each of the three sets of optimized implications, we created two distinct implications subsets; one with just single-cycle implications and the other with cross-cycle implications. We then calculated the test pattern count for each of the circuits, with these two subsets. Figure 5 shows the average vector count difference between all circuits with only cross-cycle implications, against all circuits with only single-cycle implications. With the exception of the implications optimized for online error detection, which provide very little test pattern count reduction to begin with, there is a distinct benefit from having just cross cycle implications, even though its implementation is slightly more expensive and thus fewer implications are included. On average there is a test pattern count reduction, by solely using cross-cycle implications, of 8.1% for the optimized test set of implications, and 6% for the

circuit	none	Online Error				Test				Hybrid			
		1%	3%	5%	10%	1%	3%	5%	10%	1%	3%	5%	10%
s298	362	362	361	362	359	336	313	305	306	336	331	323	314
s344	219	215	215	212	215	206	208	202	179	206	214	208	187
s349	213	217	212	218	216	206	195	195	178	206	208	199	193
s382	391	396	395	389	397	357	351	345	336	357	356	355	337
s526	763	760	761	768	768	690	634	584	517	690	694	635	571
s1196	1817	1809	1812	1824	1808	1742	1484	1481	1480	1738	1623	1483	1480
s1238	1941	1950	1941	1953	1881	1778	1588	1587	1575	1775	1654	1587	1580
s1488	1555	1556	1547	1546	1538	1485	1379	1359	1361	1488	1404	1365	1367

Table 2: 15-detect vector counts with implications from different optimized implication sets for various hardware overheads. The column none shows the test pattern counts for the circuit with no implications.

hybrid set for a total hardware overhead of 10%.

5. CONCLUSIONS

Previous works have optimally selected logic implications for online error detection [13], and for test [19] separately. In this work we have demonstrated a method that combines both goals. In the process, we made significant efficiency improvements in implication selection algorithms, and extended the analysis of the implication framework for test into sequential circuits with cross-cycle implications. We also discussed the implementation of a circuit with cross-cycle implications in scan mode and showed that significant improvements can be made over just a single-cycle case.

Our efforts have shown that even with very low hardware overhead dedicated to the implication logic we are able to provide the circuit with significant error detection and 15-detect vector count reduction. With 10% of the hardware area dedicated to implication logic, the optimized set of dual-purpose implications obtained error detection rates of up to 26% and test pattern count reduction up to 25.2%. We have also showed that, despite being slightly more expensive to implement, implications across multiple cycles can play an important role in the reduction of test pattern counts.

6. REFERENCES

- [1] S. Mitra and E. McCluskey, "Which concurrent error detection scheme to choose?" in *ITC*, Oct. 2000, pp. 985–994.
- [2] C. F. Webb and J. S. Liptay, "A high frequency custom CMOS S/390 microprocessor," *IBM Journal of Research and Development*, vol. 41, pp. 463–473, 1997.
- [3] J. von Neumann, "Probabilistic logics and synthesis of reliable organisms from unreliable components," in *Automata Studies*. Princeton University Press, 1956, pp. 43–98.
- [4] K. Mohanram and N. Touba, "Cost-effective approach for reducing soft error failure rate in logic circuits," in *ITC*, Oct. 2003, pp. 893–901.
- [5] R. Sedmak and H. Liebergot, "Selective triple modular redundancy (STMR) based single-event upset (SEU) tolerant synthesis for FPGAs," *IEEE Trans. on Nuclear Science*, vol. 51, pp. 2957–2969, 2005.
- [6] R. Sharma and K. Saluja, "An implementation and analysis of a concurrent built-in self-test technique," in *FTCS*, June 1988, pp. 164–169.
- [7] P. Drineas and Y. Makris, "Concurrent fault detection in random combinational logic," in *ISQED*, Mar. 2003, pp. 425–430.
- [8] Y. Abarbanel, I. Beer, L. Glushovsky, S. Keidar, and Y. Wolfsthal, "FoCs: automatic generation of simulation checkers from formal specifications," in *CAV*, 2000, pp. 538–542.
- [9] M. Boule, J.-S. Chenard, and Z. Zilic, "Assertion checkers in verification, silicon debug and in-field diagnosis," in *ISQED*, 2007, pp. 613–620.
- [10] I. Pomeranz and S. M. Reddy, "Reducing fault latency in concurrent on-line testing by using checking functions over internal lines," in *DFT*, 2004.
- [11] R. Vemu, A. Jas, J. Abraham, S. Patil, and R. Galivanche, "A low-cost concurrent error detection technique for processor control logic," in *DATE*, March 2008, pp. 897–902.
- [12] K. Nepal, N. Alves, J. Dworak, and R. I. Bahar, "Using implications for online error detection," in *ITC*, October 2008.
- [13] N. Alves, K. Nepal, J. Dworak, and R. I. Bahar, "Detecting errors using multi-cycle invariance information," in *DATE*, April 2009.
- [14] J. Waicukauski, P. A. Shupe, D. Giramma, and A. Matin, "ATPG for ultra-large structured designs," in *ITC*, 1990, pp. 44–51.
- [15] I. Pomeranz and S. M. Reddy, "Vector restoration based static compaction of test sequences for synchronous sequential circuits," in *ICCD*, 1997, p. 360.
- [16] E. M. Rudnick and J. H. Patel, "Putting the squeeze on test sequences," in *ITC*, 1997, p. 723.
- [17] 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.
- [18] 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*, vol. 0, pp. 385–393, 2008.
- [19] N. Alves, K. Nepal, J. Dworak, and R. I. Bahar, "Compacting test vector sets via strategic use of implications," in *ICCAD*, November 2009.
- [20] M. Bushbell and V. Agrawal, *Essentials Of Electronic Testing For Digital Memory And Mixed Signal VLSI Circuits*. Kluwer Academic Publishers, 2000.
- [21] S. C. Ma, P. Franco, and E. J. McClusky, "An experimental chip to evaluate test techniques: experiment results," in *ITC*, 1995, pp. 663–672.
- [22] Y. Mahajan, Z. Fu, and S. Malik, "Zchaff2004: An efficient sat solver," *Lecture Notes in Computer Science*, vol. 3542, pp. 360–375, 2005.