# Detecting Errors using Multi-cycle Invariance Information

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*—Ensuring reliable computation at the nanoscale requires mechanisms to detect and correct errors during normal circuit operation. In this paper we propose a method for designing efficient online error detection schemes for circuits based on the identification of *invariant relationships* in hardware. More specifically, we present a technique that automatically identifies multi-cycle gate-level invariant relationships—where no knowledge of high-level behavioral constraints is required to identify the relationships—and generates the checker logic that verifies these implications. Our results show that cross-cycle implications are particularly useful in discovering difficult-to-detect errors near latch boundaries, and can have a significant impact on boosting error detection rates.**

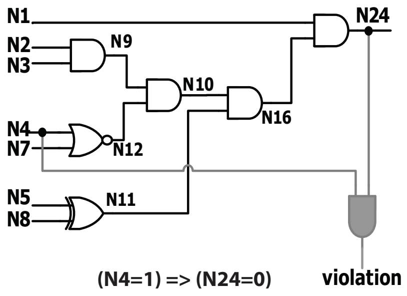

Fig. 1. A combinational circuit with added implication checker logic.

## I. Introduction

Online detection of errors is becoming increasingly critical as integrated circuits and microprocessors scale to smaller feature sizes and become more susceptible to a host of process variations and operational and environmental influences.

Strategies for detecting errors in logic vary. For instance, many previous approaches have introduced some sort of redundancy (either in time or space) to provide for detection of errors in logic. Time redundancy may involve the execution of code in multiple threads (e.g., [1]), or duplicating instructions when resources are available (e.g., [2]). Alternatively, redundancy in space may include simple duplication of the entire design, triple modular redundancy (TMR) or insertion of parity prediction logic. Finally, high-level functional assertions, such as those identified during functional verification, may also be hardcoded into the design, to signal the presence of an error (e.g., [3]).

Recent work has proposed the use of *logic implication checkers* as a means of online error detection of logic [4]. This work involves the discovery of invariant relationships among circuit sites that are expected to hold whenever the circuit is free of errors. Identifying any violation of these invariants could then be used as a means of error detection for the circuit by the addition of some simple hardware. As an example, consider the circuit shown in Figure 1. It can be verified that whenever node $N4 = 1$, node $N24 = 0$ to retain correct logical consistency. If this relationship does not hold, an error must have occurred in the intervening logic between the two sites or at the second site. The logic shown in grey can be added to the circuit to check for any violation of this implication during runtime. For instance, if node $N10$ becomes stuck-at-1 during online operation, the added checker logic would flag an error on any input vector where $N4 = 1$ and the value at node $N10$ propagates to $N24$ (i.e., is observable at $N24$). The additional hardware will independently detect errors in the logic at run time, sending a logic violation signal to the system so it can take appropriate action (e.g., re-executing the failing input sequence). A main advantage of this approach is that gate-level invariant relationships (or assertions) are automatically identified, without providing any high-level behavioral constraints. However, in the approach presented in [4] (as well as in the example shown in Figure 1) implications were identified only within a single time cycle. In practice, implications can also exist across latch boundaries, over multiple time cycles. Including these implications as well in the checker logic can help detect errors near latch boundaries, which tend to be hard to detect by single cycle implications.

In this paper, we extend the use of implications for online error detection to include implications both within a single cycle and across multiple time cycles. Violations of these implications during circuit operation are monitored by checker hardware. We demonstrate that using these implications as a means of error detection can offer very high fault coverage for the circuit and that implications that exist across latch boundaries are particularly useful for detecting a high number of potential behavioral errors. More specifically, our results show that including implications over multiple time cycles in the checker logic can significantly improve error detection rates.

## II. Related Work

Methods for detecting errors in logic online generally involve the use of some type of redundancy in time or space. Duplicating (or triplicating) the entire design in hardware, followed by a comparison of the results, is the most straightforward and obvious approach; however, it is very expensive in terms of power dissipation and area requirements. Instead, selective duplication may be used, as was done in the IBM S/390 processor, where only the instruction and execution units were duplicated [5]. Similarly, duplication may be selectively applied only for sub-circuits that are determined to be most susceptible to errors (e.g., [6]).

Coding techniques also have been proposed for error detection. Parity-check codes have been used in a number of works (e.g., [7]–[9]), while unidirectional codes such as Berger [10] or Bose-Lin codes [11] have also been proposed. Mitra and McCluskey presented a comparison of several of these coding schemes in [12]. The main disadvantage of these coding techniques is that they may require that the original circuit be altered in order to generate the codes. In addition, the size of the circuit can more than double.

Other error detection techniques have been based on the use of high-level assertions (e.g., [3], [13], [14]). These assertions are first generated during functional verification and later integrated into hardware checkers. While potentially very useful, the scope of these assertions for detecting errors may be limited, and the identification of such assertions requires an understanding of the functional intent of the circuit.

Logic implications can be detected using recursive methods [15]–[17], and other heuristics [18], [19] and have been used widely in logic synthesis for such purposes as area/delay/power optimization [20]–[22], peak current estimation [23], false noise analysis [24], and efficient ATPG [25]. Logic implications have also been used as a means of guiding localized circuit restructuring to increase the chance of logically masking errors [26], [27]. These masking techniques require modifications of the original circuit that could adversely affect delay on critical paths. In contrast, the works of [4] and [28] do not try to mask the faults, but instead, offer a means of detecting the fault using implication information without modifying the original circuit. Similar to these approaches, our goal is to make the addition of new hardware as unobtrusive as possible; however, unlike these works we do not limit ourselves to implications that exist within a single cycle and therefore we can potentially detect a larger range of faults. In addition, [28] limits itself to just checking the outputs of a circuit in the control logic of a processor against a subset of the truth-table. What we are proposing is more general. Also, unlike the works of [3], [13], [14], no understanding of the functional intent of the circuit is required in order to extract the implications and insert them in the checker logic.

## III. Methodology

A typical circuit may contain thousands of valid implications, however using all of them to create checker hardware would be too expensive. The novel contributions of this work lie in the identification of those implications, both single and multi-cycle, that are most valuable for error detection, and in the incorporation of those implications into the checking hardware. The basic flow of our approach includes the following steps:

1) Run logic simulation to identify *potential* implications.
2) Check all implications for validity.
3) Eliminate implications subsumed by others.
4) Determine the fault coverage of all valid implications.
5) Select a subset of implications that provide the desired coverage against soft or transient errors.

To find logic implications, we run circuit simulation with random vectors and record the logic state of all nodes in the circuit for each input vector applied. Once the simulation is complete, we identify potential implications by comparing node pair values in the simulation runs. For instance, for the $(a = 1) \implies (b = 1)$ implication to exist, there should never be an instance where $a = 1$ and $b = 0$. This comparison step turns out to be very fast because we can check 32 logic values (the size of an unsigned integer) for each node in parallel by doing bit vector compares. Since we are running only a small sample of all possible input vectors, we need a more formal method to verify that these implications are valid for all possible input vectors. We use ZChaff [29], a SAT solver, to check for the presence of an instance that would violate an implication. Since this implication check can be posed as a straightforward problem for the SAT solver, we can check the validity of an implication very quickly, even within a large circuit.

Once we have our list of validated implications, we run a structural analysis of the circuit to find implications in the validated list that might be contained within other implications. These implications that are subsumed by others do not enhance the fault coverage and hence can be removed from our validated list. This quick preprocessing step reduces the number of viable implications by 40%–70%.

In order to achieve best coverage with minimal extra hardware, we run a combination of structural and fault analysis on the circuit to determine the quality of each implication. For a given set of input vectors, we compute whether a fault that is present at an internal node will propagate to the output (i.e., is observable) and check if that fault causes a violation of the implication being tested. The *quality* of that particular implication and fault pair is then computed as:

$$implication\_quality = 100 \cdot \left(1 - \frac{fault\_undetected}{input\_patterns}\right). \quad (1)$$

The value *fault_undetected* is the number of input patterns where the fault is observable and the implication is not violated. Equation 1 essentially gives us the quality of coverage for a particular fault that is being checked with a particular implication. It differs from the fault coverage defined for manufacturing test where one merely wants to ensure that each fault was detected at least once by a given test set. Once we have estimated the implication quality for each fault and implication pair, we use this information to remove implications from our list that do not contribute significantly to reducing undetected errors at circuit outputs. Our approach identifies the “most valuable” implication for each modeled fault—where “most valuable” refers to the one that gives the

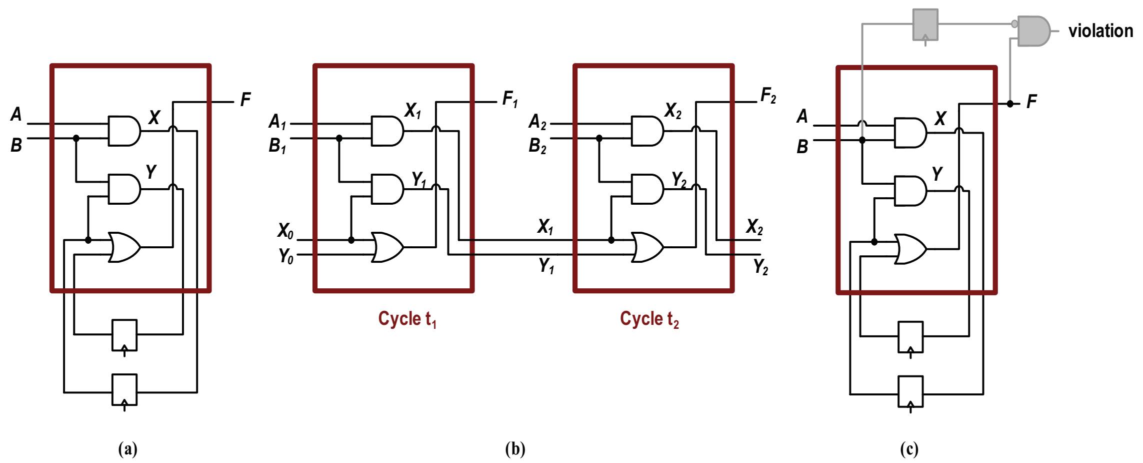

Fig. 2. (a) Simple sequential circuit. (b) Circuit expansion over two time frames. (c) Circuit with checker hardware (in gray) that raises a violation when implication $B_1 = 0 \implies F_2 = 0$ is violated.

highest coverage of errors caused by that fault. Fortunately, multiple faults will often share the same best implication allowing us to prune the list further.

Note that our approach is general and can be easily extended to include implications *across multiple time cycles* during our analysis. In this way, it may be possible to discover that 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 $n \geq 1$. Checking for these additional implication violations across clock cycles has the potential to be a particularly powerful approach for error detection.

Consider the simple circuit shown in Figure 2(a). There are no useful implications across more than a single gate that we can use for error checking in this circuit. However, we can search for implications across time cycles by creating a virtual copy of the circuit (as shown in Figure 2(b)) and executing our implication search over this expanded circuit. In this case, we find that $B = 0$ in the first clock cycle implies that $F = 0$ in the second clock cycle.

From this example, which can be generalized to more complex circuits, we can see that detecting implications that exist across multiple time cycles will increase the number of implications that we can consider and will often inherently increase the distance of those implications as well. The distance of one of these implications will include all of the logic from the original site to the appropriate latch(es) and from there to the second implication site.

More importantly, implications across time cycles will be more effective for detecting faults that are physically located only a few gates from the flip-flops and latches in the design. These faults are generally among the most difficult to detect with single-cycle implications. Thus, faults that were not adequately covered by implications in a single time cycle may find themselves better covered across cycles with the additional distance.

In addition to giving us powerful implications for the detection of otherwise hard-to-detect errors near latch boundaries, implications across time cycles also have the potential to be very effective at detecting some types of delay faults. Specifically, a delay that causes an incorrect value to be latched at the flip-flops will create a logical discrepancy when considered across multiple clock cycles. Appropriate cross-cycle implications that include this delay path will be able to detect this delay-induced error without any complicated timing or clock-gating needed for capturing the checker results. From the example shown in Figure 2, using this noted implication will also allow us to detect $X1$ slow to fall in clock cycle 1, $Y1$ slow to fall in clock cycle 1, and slow to fall faults on the branches of $B1$.

Once we have identified valuable single and multi-cycle implications, we can include them in the error checking hardware. In the simplest implementation, this translates to including a single gate in the checker hardware for each implication (for instance, the additional AND gate shown in grey in Figure 1 raises a flag if $N4 = 1$ AND $N24 = 0$ occurs due to an error in the circuit and thus violates the implication). For the multi-cycle implication $B_1 = 0 \implies F_2 = 0$, we would need to save the value of $B$ in a latch and AND its complement with $F$ in the original sequential circuit, as shown in Figure 2(c). This implication would be able to detect $X1$ stuck-at 1, $Y1$ stuck-at 1, and $F2$ stuck-at 1. Ultimately, the results of these and other individual implication signals can be OR'ed into a single error signal (possibly using a wired OR) which can then trigger an appropriate error recovery mechanism.

## IV. Results

We ran experiments with a number of sequential circuits from the ISCAS '89 benchmark suite to validate the effectiveness of our approach. Each set of implications was derived using the general algorithm described in Section III which combines the use of random vector simulation and the Zchaff SAT solver to identify and validate implications. This process is relatively fast and can quickly identify all gate level implications without the need for any high-level functional information.

Three different sets of implications were collected for each sequential circuit.

- First cycle implications (implications at time $t$)
- Second cycle implications (implications at time $t+1$)
- Cross-cycle implications

We identify first cycle implications by simulating the sequential circuit for a single cycle (with multiple vectors) where flip-flops are treated as pseudo-primary inputs and pseudo-primary outputs (as would be done for simulation of combinational test patterns of a full-scan design). We assume that all inputs (true PIs and flip-flops) are independent, and thus only implications that are valid for all possible $2^{n+f}$ input combinations will be considered valid, where $n$ is the number of primary circuit inputs (PIs), and $f$ is the number of flip-flops.

Second cycle and cross-cycle implications are both identified through the simulation of 2 clock cycles using time-frame expansion as was shown in Figure 2b. Specifically, two copies of the circuit are created, and the feedback through the flip-flops is broken and replaced with direct connections between the two circuit copies.

Second cycle implications only involve circuit sites that are both contained within the second copy of the circuit. While first cycle implications must be valid for all possible combinations of the flip-flops and primary input values, in subsequent clock cycles, the flip-flop values may be constrained due to the fact that some states are unreachable during correct operation of the circuit after the circuit is initialized. This effectively means that implications can be considered valid for a smaller set of input vectors, thereby increasing the chance that an implication will exist between two sites in a subsequent clock cycle. If even more clock cycles were analyzed, more unreachable states would likely be discovered, and more implications could be considered. Finally, cross-cycle implications are those in which a value at a circuit site in the first cycle, implies a value at a circuit site in the second cycle.

| circuit | # of | | | | # of Implications | | |
|---|---|---|---|---|---|---|---|
| | PI | PO | FF | gates | $1^{st}$ cyc. | betw. | $2^{nd}$ cyc. |
| s298 | 3 | 6 | 14 | 75 | 1687 | 3300 | 66 |
| s420 | 19 | 2 | 16 | 140 | 12898 | 20761 | 0 |
| s444 | 3 | 6 | 21 | 119 | 3054 | 8468 | 546 |
| s510 | 19 | 7 | 6 | 179 | 17845 | 25556 | 260 |
| s713 | 35 | 23 | 19 | 139 | 11485 | 10819 | 2789 |
| s953 | 16 | 23 | 29 | 311 | 13197 | 13065 | 34 |
| s1196 | 14 | 14 | 18 | 388 | 19781 | 2599 | 164 |
| s1488 | 8 | 19 | 6 | 550 | 20822 | 17336 | 750 |

TABLE I
NUMBER OF IMPLICATIONS IN EACH CLASS.

In Table I we report the number of implications discovered from each class ($1^{st}$-cycle, between cycles, or $2^{nd}$-cycle only). Note that by expanding the search beyond a single cycle, many more implications can be discovered. We also investigated the effect that implication class has on the distance between implication sites. As noted in Section III, by considering implications across multiple cycles, we expect to increase the distance of the potential implications we are considering. In

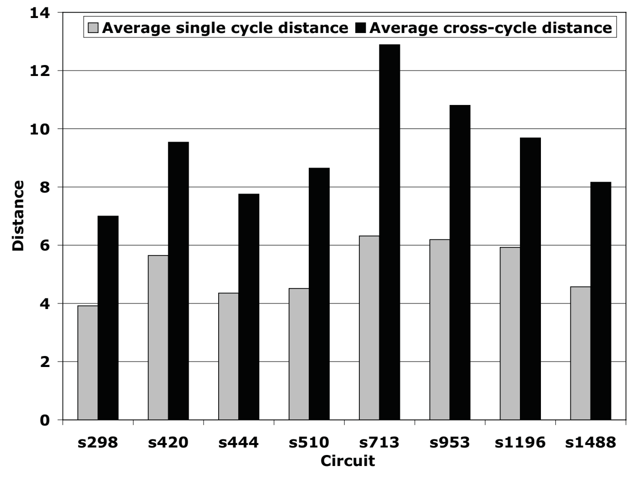

Fig. 3. Average implication distance for single and between-cycle implications.

Figure 3 we show the average implication distance for the single and cross-cycle implications. Notice that, as expected, the cross-cycle implications tend to have a larger logical distance between the implication sites than implications contained within a single cycle. Implications with larger distance have the potential to cover more errors and hence are likely to be more valuable when used for error detection.

Due to hardware overhead limitations it may not be reasonable to include all discovered implications in the checker logic. Nevertheless, it is instructive to estimate what the upper-bound error coverage would be if all these implications were included in the checker. To do this, we next ran stuck-at-fault analysis on the circuits while simultaneously analyzing the ability of these implications to detect each of these faults. Of course, stuck-at faults are unlikely to be truly representative of the errors actually occurring in one of these circuits because such faults would be almost certainly detected by the ATPG test set. Indeed, the errors which must be detected online should generally be much harder to excite and/or may only be present on random clock cycles. However, if our implications can successfully detect stuck-at faults, they are likely to be able to detect many of these transient errors as well.

Figure 4 shows this average error coverage, for the different subsets of implications outlined in Table I. In this figure, the error coverage is calculated for each fault as the fraction of all patterns for which that fault will cause an error at an output for which at least one implication will be violated—allowing the propagated error to be detected and flagged by checker logic. Averages over all faults are shown for each implication set and for the combined set of all possible implications. The coverage considering all implications ranges from approximately 62% for s420 to 90% for s510. For several of the benchmarks, almost all of the error detection could be achieved solely through cross-cycle implications. As stated earlier, this is likely due to both their increased distances and the fact that they are physically capable of covering faults near the flip-

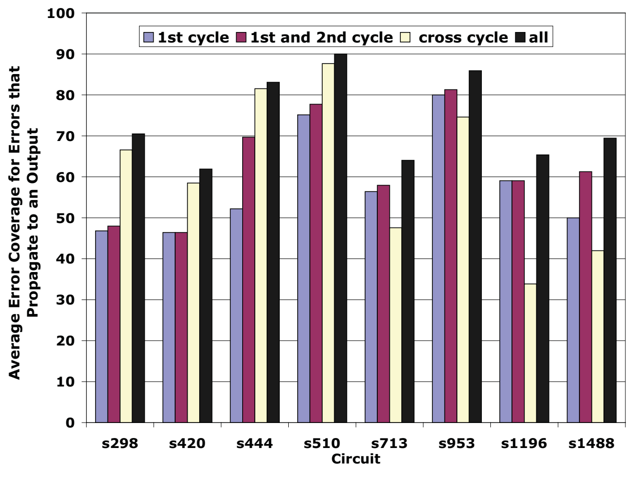

Fig. 4. Contribution of different implication classes to error detection.

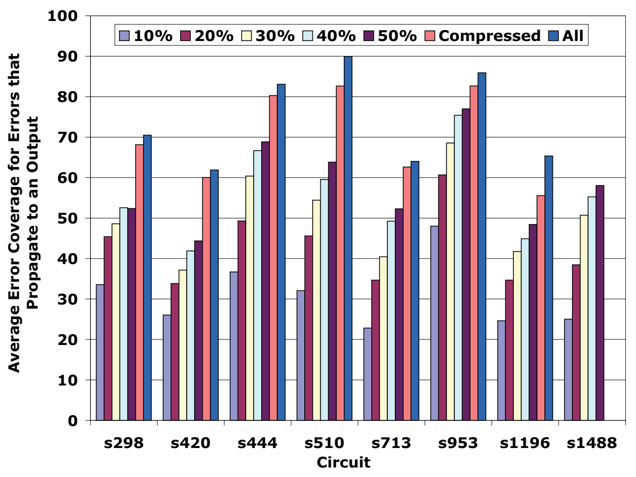

Fig. 5. Average error coverage achieved for different area overhead thresholds.

flops that are highly unlikely to be covered by single-cycle implications.

Including all of the implications in our checker logic would be prohibitively expensive. However, from the discussion in Section III, we do not expect all these implications to provide the same quality (as defined in Eqn. 1). Thus the low quality implications can be removed from our set without significantly hurting the overall fault detection rate. We initially compress our implication set by determining which implication provides the best error coverage for each fault. Each implication identified in this manner is added to the implication list. While this could imply one implication per fault, this is an unlikely case because a single implication may provide the best coverage for multiple faults. Thus, this "implication sharing" allows us to compress our implication set even further.

| circuit | $1^{st}$ cycle | cross-cycle | $2^{nd}$ cycle only |
|---|---|---|---|
| s298 | 39 | 109 | 3 |
| s420 | 35 | 160 | 0 |
| s444 | 33 | 182 | 19 |
| s510 | 94 | 167 | 6 |
| s713 | 195 | 156 | 8 |
| s953 | 233 | 150 | 6 |
| s1196 | 449 | 44 | 2 |
| s1488 | 409 | 263 | 51 |

TABLE II
NUMBER OF COMPRESSED IMPLICATIONS.

Table II summarizes the results of this pruning operation. Note that the third data column only includes those implications that were "newly" discovered in the second cycle and are not also present in a first cycle analysis. In general, we can get over an order of magnitude reduction in the number of implications in the checker logic by using this compression process. Also note that for all circuits, the cross-cycle implications remain an important component of the final implication list.

Unfortunately, even this compressed set of implications may be too large to implement in checker hardware. However, another advantage of our method lies in the fact that area overhead may be easily traded off for additional fault coverage. Because cross-cycle implications require an additional flip-flop to hold circuit site values from one clock-cycle to the next, they are more expensive than single cycle implications. Analysis with the Mentor Graphics layout tool, ICStation, shows that in general the insertion of a cross-cycle implication is approximately twice as expensive in terms of area as a single site implication when both standard cell area and routing are considered. Given these estimates, we created implication sets where the area overhead in terms of gate count for the checker was limited to 10, 20, 30, 40, or 50 percent, where a single-cycle implication counted as 1 gate and cross-cycle implication as 2. In a physical layout of the circuit the exact area overhead may vary somewhat due to routing and other issues. However, on average, the overhead is often relatively close to our targeted goal (e.g., experiments with ISCAS combinational circuits showed an average 12.6% area overhead post-layout when 10% overhead was targeted [4]). A greedy algorithm was used to choose the included implications for each hardware overhead limit. We then found the average error coverage for each set of implications in the same way as it was calculated earlier in Figure 4. The results are shown in Figure 5. Of particular note is the fact that the full compressed implication set has almost the same error coverage as the set of all possible implications. The reduction in coverage varies from less than 2% to slightly over 7%. It is also important to realize that, even when the checker is restricted to low overheads, the final error rate for the circuit will still generally be very small. The error coverages shown in Figure 5 presuppose that an error is always present at a circuit site and has propagated to an output. In actuality, errors will almost never be present on every clock cycle. Even stuck-at faults will be either "not excited" or "not observed" for many input combinations and thus will cause no error at the output for those combinations.

Finally, Figure 6 shows the percentage cross-cycle implica-

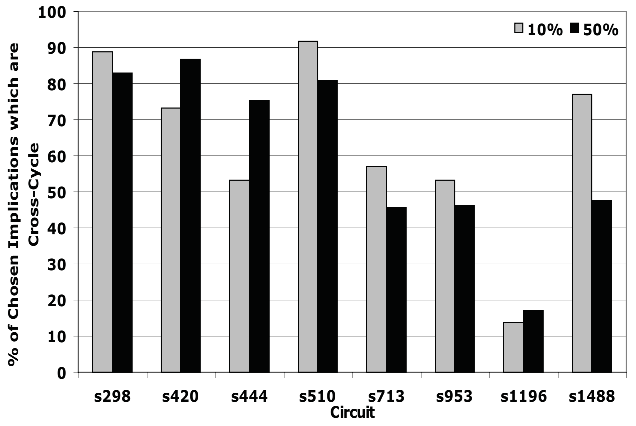

Fig. 6. Percentage of cross-cycle implications for two area overhead thresholds.

tions making up both the 10% and 50% overhead experiments. Even with the additional cost of cross-cycle implications, their superior error detection capability ensures that they play a significant role in maximizing error coverage.

## V. Conclusions

We have presented an online error detection technique based on the use of logic implication information discovered using multi-cycle logic analysis. Overall, our results point to the promising use of single- and cross-cycle implications as a means of detecting faults in circuits. While implications alone cannot be used to achieve complete fault detection, up to 90% error coverage is possible.

For future work we would like to explore implications targeted intelligently towards "important" faults that cause catastrophic failures. These implications could provide excellent coverage while trading off area overhead. Because all observable faults do not have a similar impact on output usability, implication checkers could be targeted for only covering these catastrophic faults.

We also plan to analyze more completely the relationship between the logical distance between node sites in an implication and the fault detection rate for the implication. In addition, we plan to evaluate the effect these multi-cycle implications have on detection of delay faults across various benchmarks. Finally, we note that all the implications we use are simple 2-node implications. It is possible that higher fault coverage could be obtained if more complex implications were included in the checker logic; future work will explore ways of efficiently identifying these more complex implications as well.

## References

[1] J. Ray, J. C. Hoe, and B. Falsafi, "Dual use of superscalar datapath for transient-fault detection and recovery," in *Intl. Symp. on Microarchitecture*, 2001, pp. 214–244.

[2] M. A. Gomaa and T. N. Vijaykumar, "Opportunistic transient-fault detection," in *ISCA*, June 2005.

[3] 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.

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

[5] 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.

[6] K. Mohanram and N. Touba, "Cost-effective approach for reducing soft error failure rate in logic circuits," in *ITC*, Oct. 2003, pp. 893–901.

[7] S. Almukhaizim, P. Drineas, and Y. Makris, "Cost-driven selection of parity trees," in *VTS*, 25-29 April 2004, pp. 319–324.

[8] K. Mohanram, E. Sogomonyan, M. Gossel, and N. Touba, "Synthesis of low-cost parity-based partially self-checking circuits," in *On-Line Testing Symposium*, 2003.

[9] N. Touba and E. McCluskey, "Logic synthesis of multilevel circuits with concurrent error detection," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 16, no. 7, pp. 783–789, July 1997.

[10] J. M. Berger, "A note on an error detection code for asymmetric channels," *Information and Control*, vol. 4, pp. 68–73, 1961.

[11] D. Das and N. A. Touba, "Synthesis of circuits with low-cost concurrent error detection based on Bose-Lin codes," in *VTS*, 1998, pp. 309–315.

[12] S. Mitra and E. McCluskey, "hich concurrent error detection scheme to choose?" in *ITC*, Oct. 2000, pp. 985–994.

[13] R. Drechsler, "Synthesizing checkers for on-line verification of system-on-chip designs," in *ISCAS*, May 25-28 2003, pp. IV–748 – IV–751.

[14] M. Boule, J.-S. Chenard, and Z. Zilic, "Assertion checkers in verification, silicon debug and in-field diagnosis," in *ISQED*, 2007, pp. 613–620.

[15] L. Entrena, E. Olías, J. Uceda, and J. Espejo, "Timing optimization by an improved redundancy addition and removal technique," in *EURO-DAC/EURO-VHDL*, 1996, pp. 342–347.

[16] W. Kunz and D. K. Pradhan, "Recursive learning: a new implication technique for efficient solutions to CAD problems—test, verification, and optimization," *IEEE Trans. on CAD*, vol. 13, no. 9, pp. 1143–1158, Sep. 1994.

[17] J. Zhao, E. M. Rudnick, and J. H. Patel, "Static logic implication with application to redundancy identification," in *IEEE VLSI Test Symposium*, 1997, pp. 288–293.

[18] K. Gulrajani and M. S. Hsiao, "Multi-node static logic implications for redundancy identification," in *DATE*, 2000, pp. 729–735.

[19] M. A. Iyer and M. Abramovici, "Fire: a fault-independent combinational redundancy identification algorithm," *IEEE Trans. on VLSI Systems*, vol. 4, no. 2, pp. 295–301, 1996.

[20] R. Bahar, M. Burns, G. Hachtel, E. Macii, H. Shin, and F. Somenzi, "Symbolic computation of logic implications for technology-dependent low-power synthesis," in *ISLPED*, Aug. 1996, pp. 163–168.

[21] W. Kunz and P. Menon, "Multi-level logic optimization by implication analysis," in *ICCAD*, Nov. 1994, pp. 6–10.

[22] B. Rohfleisch, A. Kölbl, and B. Wurth, "Reducing power dissipation after technology mapping by structural transformations," in *DAC*, 1996, pp. 789–794.

[23] S. Bobba and I. Hajj, "Estimation of maximum current envelope for power bus analysis and design," in *ISPD*, Apr. 1998, pp. 141–146.

[24] A. Glebov, S. Gavrilov, D. Blaauw, and V. Zolotov, "False-noise analysis using logic implications," *ACM Trans. on Design Automation of Electronic Systems*, vol. 7, no. 3, pp. 474–498, 2002.

[25] P. Tafertshofer, A. Ganz, and M. Henftling, "A SAT-based implication engine for efficient ATPG, equivalence checking, and optimization of netlists," in *ICCAD*, Nov. 1997, pp. 648–655.

[26] S. Almukhaizim and Y. Makris, "Soft error mitigation through selective addition of functionally redundant wires," *IEEE Trans. on Reliability*, vol. 57, no. 1, pp. 23–31, March 2008.

[27] S. Krishnaswamy, S. M. Plaza, I. L. Markov, and J. P. Hayes, "Enhancing design robustness with reliability-aware resynthesis and logic simulation," in *ICCAD*, Nov. 2007, pp. 149–154.

[28] 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.

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