Reducing Delay Time by Analyzing the Critical Path

Alison Buben
Computer Science Department, Indiana University of Pennsylvania
a.j.buben@iup.edu

Abstract - **Error detection for online circuits has been a topic of research for many years. Some approaches are designed for complete coverage, and others for added detection that is very cost effective. Methods that strive for the highest error detection in online circuits (probability of detection), can often have major delay time increases. This paper discusses the idea of balancing both a high probability of detection with low delay times. It will be shown that a significant reduction in delay is possible with only a slight decrease in probability of detection.**

I. Introduction

The idea of checking for errors in circuits has always been a very important topic. However, the ways errors are detected, and the total percentage of coverage can vary dramatically. My research was part of a project that intended to provide some error detection for those who don't need complete coverage or those who only have limited hardware or resource budgets. On an even more narrow scope, since this research is trying to find the best way to get the highest error detection while considering the client's hardware or resources, delay timing must be considered. In this paper a method will be described to not only get a high probability of detection (compared to a similar approach conducted), but also take into consideration delay, and to substantially improve the delay times.

II. Related Work

There are many ways to check for error detection, and these ways can range from parity to hamming code, and even duplication of the entire circuit. The research in this paper deals with finding errors in the logic of a circuit. In the past, approaches have used redundancy to deal with errors in logic, but the approach used in this paper deals with finding assertions or gate level relationships [1].

The core of this approach deals with identifying implications that tell what the output of a gate must be when another gate has a certain value. Simulations are run on circuits to find all potential implications, and then the best implications (covering the most area, finding the most faults) are selected [2].

The main topic covered in this paper is how to improve on this method by decreasing delay time. Some previous work has been done in this area by finding out how to convert an implication file into a verilog file that can be used in timing analysis. Scripts to convert an implication file were created by Kundan Nepal, and the original sorted implication files were produced by Nuno Alves' Prime Implication Algorithm (see [1], [2] or [3]).

III. Algorithm

Although there was a good algorithm to get a sorted set of implications, this set did not take into consideration the delay times. Thus, a new, “modified” method was needed. After testing several ideas, a series of selecting techniques produced a new set of implications. These implications yielded very similar probability of detection results, with a decrease in delay compared to their un-modified counterparts.

This algorithm is noteworthy because it finds a middle-ground between finding the best possible probability of detection and delay. To achieve a good balance in performance there

must be limits on what is allowed. Through analysis of what delay times we obtained with the prime implication algorithm and variations on how many implications from the critical path certain benchmarks contained, we realized there was a way to decrease delay and lose very little error coverage (or even gain some). Thus, to keep delay down and still have reasonable probability of detection, we focused on limiting the number of implications on the critical path.

Figure 1 shows a flowchart representing the various steps to get a new implication file (the modified implication set). To start the process, prime implications are needed, and these are from a circuit's "sorted.imp" file.

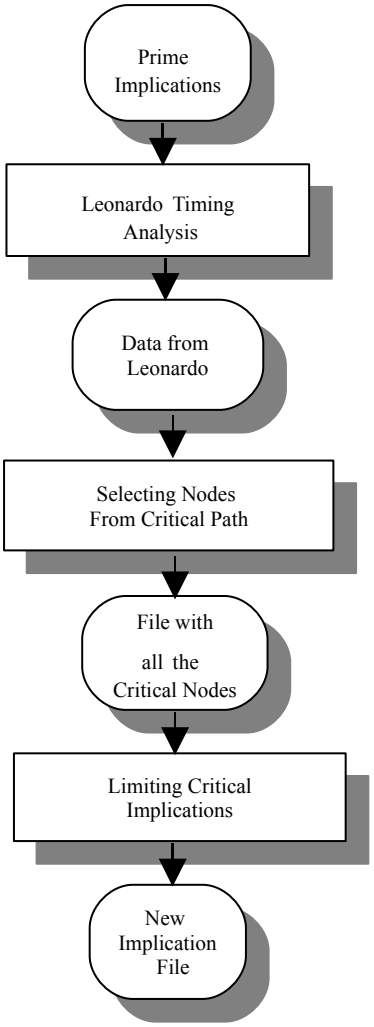

Fig. 1. Flowchart to show the steps in the modified method.

My modified method is a way to modify the sorted implication file that was obtained with the prime implication method, thus being named the "modified prime implication method." To start testing different types of methods, I made two scripts. One script, report_to_list.pl, parses a report from Leonardo Spectrum with the show nets option used, into a list of nodes on the critical path. This list of nodes on the critical path is what I call a data.txt file, and is needed for the next script. The next script, crit_path_mod.pl, uses the data.txt file along with the sorted.imp file to make a new implication file with the number of critical implications (implications that use nodes on the critical path) limited. For my method, only keeping the first ten critical implications in the new implication file worked well. Through this method, the delay is usually reduced, without causing the probability of detection to decrease much. Since critical implications are good for error detection, but can really add to the delay, only allowing ten critical implications in the new file seemed to be a good balance.

I started out with the sorted implication file (called prime implications), ran it through Matlab and Leonardo Spectrum to get the timing delay, and saved the report I got from Leonardo. Then I took this report and parsed it so I had a list of the nodes on the critical path. From this list I could limit the number of implications on the critical path and make a new implication file. Then I made 10, 20 and 30 percent overheads from my newest implication file and ran the probability of detection with Fastscan and the timing delay with Leonardo Spectrum.

With these modified implications, delay time should decrease since every implication on the critical path would not be used. Though limiting the implications on the critical path can decrease the amount of error detection, we hoped that the new modified method would only have minimal differences from the original prime implications. We theorized that

decreasing the delay will prove to be more critical than the amount of error detection lost.

IV. Results

Through timing analysis in Leonardo Spectrum, and probability of detection in Fastscan, the results of the modified method were able to be compared against the original set of implications. Each benchmark circuit was divided into percentage overheads, with the overhead relating to the total number of gates in the circuit. In Figure 2. the ten circuits are displayed, along with a comparison between the original (non-modified) and modified methods. Only overheads 10 % – 30% are displayed.

Circuit	10% Original	10% Modified	20% Original	20% Modified	30% Original	30% Modified
c432	3.39	3.36	3.43	3.39	3.53	3.41
c499	2.57	2.59	2.61	2.64	2.73	2.68
c880	2.28	2.20	2.43	2.24	2.59	2.26
c1355	2.50	2.48	2.60	2.58	2.69	2.74
c1908	3.44	3.33	3.58	3.41	3.74	3.56
c2670	3.83	3.80	3.98	3.81	4.05	3.82
c3540	4.55	4.53	4.82	4.72	4.98	4.89
c5315	3.81	3.81	3.89	3.89	3.90	3.86
c6288	13.23	13.11	14.01	13.87	14.96	14.50
c7552	3.35	3.35	3.59	3.53	3.84	3.63

Fig. 2. Comparison of the original and the modified implications in regards to delay.

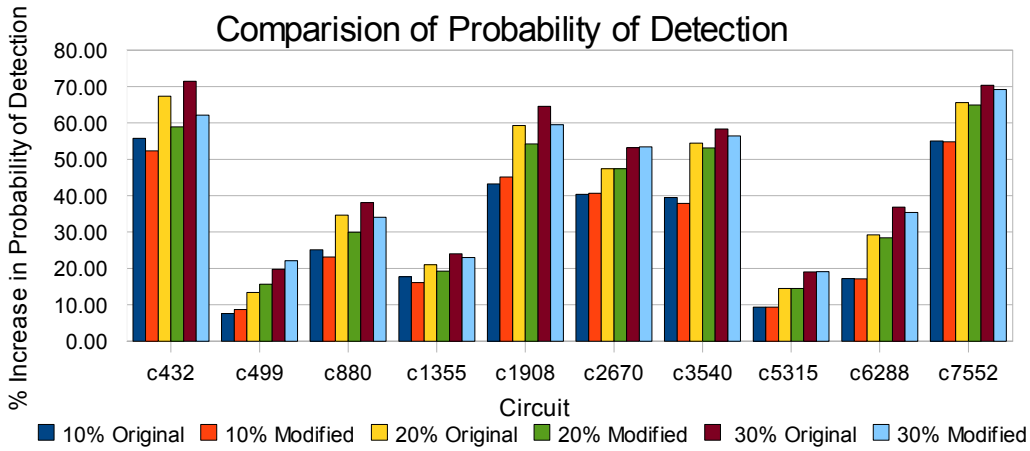

Fig. 3. Comparison of the Prime Implication algorithm and the modified Prime Implication algorithm for Probability of Detection.

Figure 2 shows the comparison of original and modified implications in regards to delay. The numbers are presumably in nanoseconds. The difference between the two methods might appear miniscule at times, however, this can actually make a major difference. Initially, the delay of our sorted set of implications was calculated by Leonardo Spectrum, and then it was compared to the delay of our new modified set to see the difference (also by Leonardo Spectrum). Notice that the results are as expected: the delay decreases as the number of implications on the critical path are limited. Although there are a few places where the modified implication set has more delay, overall it has a

greater decrease in delay than the original prime implication set.

One can see the results of comparing the prime implication algorithm and the modified prime implication algorithm for probability of detection in Figure 3. Usually the probability of detection decreases when implications on the critical path are limited. Looking at the results, the probability of detection does not decrease for all the benchmarks; for some it will stay relatively the same or even increase. When all the data are compared, there is no difference between the original and modified implications greater than 10%, with 1.6% being the average difference. This shows that even though the modified method has a delay decrease, it is still very similar to the results the original method achieved.

V. Conclusions

In this paper a new approach was presented that takes into consideration the need for a balance among high probability of detection and low delay time. The research has shown that limiting the number of critical implications in each set of overheads can produce better delay times with hardly any reduction in probability of detection. The need to further refine work is imperative. New research topics are always needed, but they should first fully consider all available possibilities.

REFERENCES

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

[2] N. Alves, K. Nepal, J. Dworak, and R. I. Bahar, "Detecting errors using multi-cycle invariance information," in Design Automation and Test in Europe (DATE), April 2009.

[3] N. Alves, K. Nepal, A. Buben, J. Dworak, and R. I. Bahar, "A Cost Effective Approach For Online Error Detection Using Invariant Relationships," Work in Progress.