IMPROVED TEST GENERATION FOR HIGH-ACTIVITY CIRCUITS

KEWAL SALUJA
KYUCHULL KIM
University of Wisconsin

The authors propose an implementation of a test-pattern generator based on the Podem (path-oriented decision-making) algorithm. The implementation allows the packing of more than one signal value into a word during implication. The authors implement both the event-driven and compiled-code versions of the imply function with and without packing signal values. Results show that conventional Podem with event-driven implication performs better for low-activity circuits; whereas Podem with compiled code and packed signal values performs better for high-activity circuits.

Podem is one of several basic automatic test-pattern generation algorithms for combinational logic circuits.[1-3] Podem, short for path-oriented decision-making, uses a depth-first search from the fault location to assign primary input values. The result of these assignments at internal nodes is then determined by logic simulation (implication). Podem must compute primary input combinations that both excite the fault and propagate it to primary outputs.

We can improve this algorithm for high-activity circuits by packing more than one signal value into a word. Such packing introduces parallelism into the implication part of test generation. Although other researchers have used this method to achieve parallelism in fault simulation, none have reported its use in the implementation of test-generation algorithms.

We can use this technique of packing more than one signal value into a word in the implication part of Podem to examine both the normal assignment and the alternative assignment to an input variable in parallel. With parallelism, we can search the input space faster.

In experiments to assess the benefits of such a scheme, we implemented both compiled-code and event-driven versions of the *imply* function in Podem with and without the parallelism offered by value packing. At first glance, the results seemed inconclusive for the benchmark circuits. For some circuits, the packed implementation outperformed the other implementation but for other circuits the opposite was true. This contradiction led us to give a characterization of the circuits using a metric, called the *input-cone-based measure*, or ICBM. By using this ICBM metric, we could determine beforehand which method will perform better than the other.

BACKGROUND

In Podem, we perform implication for only one input combination. In parallel Podem, or P-Podem, typically two or four logic values are packed into a one byte or word. Hence, we can perform implication for different input combinations simultaneously. After each implication operation in P-Podem, which we call *parallel implication*, we have available the results for different input combinations after only one implication step.

By looking at these results, we can determine the input combinations that cannot be tests for the fault of interest. Because we do not have to perform the implication operation on that input combination again, the

0740-7475/90/0008-26$1.00©1990 IEEE

time between backtracks is much lower with parallel implication. The advantage of this approach is that it can detect the failure of excitation or propagation of the fault in fewer steps than Podem.

To illustrate the power of parallel implication, we consider a scheme in which two logic values are packed in a byte. Each of these logic values can be one of five choices, just as it can in conventional Podem. Further, we assume that the processing of different gate types in the circuit, for which the tests are derived, is identical in Podem and in parallel implication. For those interested in more about P-Podem application than just the implication step, we describe the limitations and advantages of packing more than two values (such as four values) in an earlier work.[4]

In a P-Podem application based on two values, we pack a primary input value from a backtrace operation into the lower field of a byte. We pack the complemented PI value, which is the alternative assignment, into the higher field of the byte. We then perform forward implication for the given PI assignments. Thus, P-Podem, implication is performed for two input combinations in parallel.

In conventional Podem, just after we do the implication for the faulty gate to excite the fault, we check if the signal value at the fault site is the same as the fault value. Suppose these values are indeed equal. After the implication for the given input assignment and if the last node of the decision tree is not flagged, we can make an alternative assignment for the last PI and perform implication for it. If we had done the implication for the alternative assignment simultaneously with the assignment that failed to generate a test, we could have avoided doing the implication for the alternative assignment. This simultaneous evaluation process is called a *localized breadth-first search* because at each node of the decision tree, we check two input combinations together to see if they can be tests for the given fault.

Implication is one of the most expensive and time-consuming steps in Podem and other test generators.

IMPLICATION

Although our primary goal in test generation is to reduce backtracking, one of the most expensive and time-consuming steps in Podem and other test generators is implication. Both Podem and P-Podem invoke implication after every backtrace, which can be either *event-driven* implication or *compiled-code* implication.

In event-driven implication, which is also called the selective trace method, we process only the gates whose input(s) change when a new PI is assigned. Typically, a test-generation program that uses this method is independent of the logic circuit. That is, once the program is generated, we can run it for any circuit. We do not need to separately compile the program if the circuit changes.

In compiled-code implication, we process all the gates in a circuit regardless of whether their input changes when a new PI is assigned. Compiled code implication is similar to the methods used in SSIM[6] and HSS.[7] In these programs, the description of the logic circuit is stored implicitly in implication. Consequently, we must compile the implication part of the test-generation program for each circuit separately, although each circuit needs to be compiled only once.

AN EXPERIMENTAL STUDY

Although event-driven implication allows us to process fewer gates, compiled-code implication can actually be faster, depending on the

It seems that no one method is decidedly superior, although for six of the 10 circuits, P-Podem-C performed better than Podem-E.

activity in a circuit. To analyze the performance of conventional Podem and P-Podem in test generation, we developed the following four programs derived from both versions of the algorithm:

1. Podem-E, Podem with event-driven implication
2. Podem-C, Podem with compiled-code implication
3. P-Podem-E, P-Podem with event-driven implication
4. P-Podem-C, P-Podem with compiled-code implication

We used these programs to derive tests for the ISCAS benchmark circuits, which were identified at the 1985 International Symposium on Circuits and Systems.[8] We ran all the programs for all faults that were supplied with the circuits. We used a Astronautics Corp.'s ZS-1,[9] a high-speed computer designed to achieve one-third the performance of a Cray X-MP1. It has a performance of 44 MIPS and 22 MFLOPS. The programs are in C. We did not use any special features of the ZS-1 architecture.

We set a limit of 10,000 on the number of backtracks for Podem. If and when the program exceeded that limit, we aborted test generation for that fault. We set the limit for P-Podem so that the input space explored by P-Podem would be the same as that for Podem. In this way, all methods would achieve the same fault coverage.

Table 1 gives the normalized CPU time with respect to the CPU time of Podem-E for the 10 ISCAS benchmark circuits. Compilation time was relatively insignificant for circuits with 500 or more gates, so we do not include it in the CPU time in Table 1.

RESULTS

From the data in Table 1, it seems that no one method is decidedly superior, although for six of the 10 circuits, P-Podem-C performed better than the conventional Podem-E. Clearly, we need to find a method that can tell us which test generator—Podem-E or P-Podem-C—is likely to perform better for a given circuit.

Since circuits with fewer than 500 gates are of relatively little interest, we confine the rest of our discussion on results to the seven circuits with more than 500 gates. From Table 1, we observe that the test-generation time for P-Podem-C is between 49% to 91% that of Podem-E for four of

Table 1. *Normalized CPU time of test generators.*

Circuit	No. of Gates	Podem-E	P-Podem-E	Podem-C	P-Podem-C
c432	160	1	1.15	0.61	0.65
c499	202	1	1.12	0.42	0.55
c880	383	1	1.67	1.17	1.50
c1355	546	1	1.13	0.97	0.91
c1908	880	1	1.19	0.82	0.74
c2670	1,269	1	1.07	1.41	1.22
c3540	1,669	1	1.04	0.94	0.75
c5315	2,307	1	1.21	1.80	1.72
c6288	2,416	1	1.03	0.54	0.49
c7552	3,513	1	1.22	4.05	3.80

the seven circuits (with more than 500 gates). P-Podem-C performed worse than Podem-E for the remaining three circuits. Also, P-Podem-C performed better than Podem-C for all seven of the larger circuits.

The poor performance of P-Podem-C over Podem-E for the three (of seven) circuits is probably because these circuits have low activity. For these low-activity circuits, we need to process only a few gates in event-driven implication. Because of this difference in low and high activity, we used the following ICBM metric, which we briefly described earlier, to divide the circuits into two classes:

$$\text{ICBM} = \frac{1}{\text{No. of PIs}} \sum_i \frac{\text{No. of nodes in the forward cone of PI}_i}{\text{No. of nodes in the circuit}}$$

In this equation, a node is a gate or a PI in the circuit. The forward cone of an input is the subcircuit of the circuit whose nodes can be reached from the input. Thus, the number of nodes in the forward cone of PI_i is the maximum number of gates that will be processed during implication when PI_i is assigned a new value.

Thus, we can view ICBM as a metric that gives an upper bound of activity in the circuit during implication. A circuit with a higher ICBM is likely to be more active than a circuit with a lower ICBM. Table 2 gives the ICBM values for the seven ISCAS circuits, as well as the relative performance of all four test generators. We have normalized the entries in the table with respect to the CPU time for Podem-E.

The table shows that P-Podem-C outperforms all other versions of Podem for large circuits when the ICBM value is higher than 0.1. Further, the improvement in P-Podem-C performance is larger when the ICBM value is higher. We can reason that when the ICBM value is high, a considerable portion of a circuit is likely to change logic values. Therefore, we have to carry out implication for that portion before termination. In these cases, event-driven implication may involve processing many gates. For this reason, compiled-code implication is likely to outperform event-driven implication when the ICBM value is high. We can also see this in the comparative performance of Podem-E and Podem-C.

The reason for the poor performance of P-Podem-E over Podem-E is that in event-driven implication, we process gates along signal paths. Because P-Podem-E must process these signal paths for two input combinations, it is processing more gates than Podem-E, which processes paths for only one input combination. This extra processing offsets the advantage P-Podem-E has over Podem-E of searching the enumeration tree with fewer backtracks.

The input-cone-based measure, or ICBM, allows us to determine beforehand which version of Podem will perform best.

Table 2. *Performance in normalized CPU time for the seven circuits with more than 500 gates.*

Circuit	ICBM Value	Podem-E	P-Podem-E	Podem-C	P-Podem-C
c1355	0.45	1	1.13	0.97	0.91
c1908	0.30	1	1.19	0.82	0.74
c3540	0.28	1	1.04	0.94	0.75
c6288	0.45	1	1.03	0.54	0.49
c2670	0.04	1	1.07	1.41	1.22
c5315	0.04	1	1.21	1.80	1.72
c7552	0.05	1	1.22	4.05	3.80

Overall, the event-driven method performs better than the compiled-code method for low-activity circuits, but parallelism is more effective when used with the compiled-code method.

We also confirmed our experiments to determine circuit activity statistically using logic simulation. We measured the number of nodes that had to be processed becaue of one input change. We assigned a binary value to an input, and performed true logic simulation, which is done in a way similar to an event-driven method. We collected average node counts by repeating this process until we had assigned a random binary value to all PIs. We found that circuits with a low ICBM value indeed showed low activity and those with a high ICBM value showed a substantially higher activity. We present more details of our study in an earlier work.[4]

We can also measure the degree of circuit activity dynamically using random-pattern fault simulation. Many test-generation systems start with random vectors, which makes the cost of this approach minimal. We can use this information in place of the ICBM value as a guide to determine which type of Podem is for test generation for a circuit.

In another experiment, we extended our set of circuits by making serial and parallel connections of the ISCAS circuits.[4] We used Podem-E and P-Podem-C exclusively to generate tests for these larger circuits. We found that for circuits with an ICBM value greater than 0.1, P-Podem-C performed substantially better than Podem-E.

Overall, our results show that the event-driven method performs better than the compiled-code method for low-activity circuits, while parallelism is more effective when used with the compiled-code method. We thus recommend the the following steps to determine best test generator for a given circuit.

1. For circuits with fewer than 500 gates, use Podem-E.
2. For circuits with more than 500 gates, determine the ICBM value for a circuit, which you can do in linear time. If the ICBM value is higher than a threshold value (0.1 in our examples), use P-Podem-C; otherwise, use Podem-E.

We have proposed a method of improving the performance of a test generator by (1) efficiently using unused bits in a byte or word and (2) using compiled-code implication. We used these methods to improve the performance of Podem. We determined through experimental study that these methods result in reduced CPU time for test generation for a class of combinational circuits. To characterize these circuits, we have defined a metric that allows test engineers to choose the most efficient test-generation program for that circuit. We found that P-Podem-C, P-Podem with compiled-code implication, which is based on multiple signal values per word, substantially outperforms conventional Podem for circuits with high activity. D&T

REFERENCES

1. J. Roth, W. Bouricius, and P. Schneider, "Programmed Algorithms to Compute Tests to Detect and Distinguish Between Failures in Logic Circuits," *IEEE Trans. Electronic Computers*, Vol. EC-16, Oct. 1967, pp. 567-579.
2. P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," *IEEE Trans. Computers*, Vol. C-30, Mar. 1981, pp. 215-222.
3. H. Fujiwara and T. Shimono, "On the Acceleration of Test Generation Algorithms," *IEEE Trans. Computers*, Vol. C-32, Dec. 1983, pp. 1137-1144.

4. K. Kim and K. Saluja, *Speeding up the Podem Test Generation Process*, tech. rpt. ECE-89-22, Dept. of Electrical and Computing Eng., Univ. of Wisconsin, Madison, 1989.

5. R. Kessler, *Adapting Compiled Code Simulators to Aid in Test Generation*, project rpt. ECE/CS 756, Dept. of Computer Science, Univ. of Wisconsin, Madison, 1988.

6. L. Wang et al., "SSIM: A Software Levelized Code Compiler," *Proc. Design Automation Conf.*, 1987, pp. 2-8.

7. Z. Barzilai et al., "HSS—High-Speed Simulator," *IEEE Trans. Computer-Aided Design*, Vol. CAD-6, July 1987, pp. 601-617.

8. F. Brglez and H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translation in Fortran," *Proc. Circuits and Systems Symp.*, 1985, pp. 705-712.

9. J. Smith, "Dynamic Instruction Scheduling and the Astronautics ZS-1," *Computer*, Vol. 20, July 1989, pp. 21-35.

Kewal K. Saluja is an associate professor in the Department of Electrical and Computer Engineering at the University of Wisconsin, Madison, where he teaches logic design, computer architecture, microprocessor-based systems, VLSI design, and testing. His research interests include design for testability, fault-tolerant computing, VLSI design, and computer architectures. He holds a BE in electrical engineering from the University of Roorkee and an MS and a PhD in electrical engineering from the University of Iowa.

Kyuchull Kim is working towards a PhD in electrical and computer engineering at the University of Wisconsin, Madison. His research interests include automatic test-pattern generation, fault diagnosis, and VLSI design and test. He holds a BS and an MS in physics from the Seoul National University and an MS in electrical and computer engineering from the University of Wisconsin.

Direct questions or comments on this article to K. Saluja, Dept. of Electrical and Computer Engineering, University of Wisconsin, 1415 Johnson Dr., Madison, WI 53706.