

An Optimal Partitioning Algorithm for Combinational CMOS Circuits Using Particle Swarm Optimization

Gaurav Singhal, *Student Member, IEEE*, Ganesh K Venayagamoorthy, *Senior Member, IEEE* and Scott C. Smith, *Member, IEEE*
Real-Time Power and Intelligent Systems Laboratory
Department of Electrical and Computer Engineering
University of Missouri-Rolla, MO 65409, USA
gsgf4@umr.edu, skumar@ieee.org, smithsco@umr.edu

Abstract

This paper presents a swarm intelligence based approach to optimally partition combinational CMOS circuits for pseudoexhaustive testing. The partitioning algorithm ensures reduction in the number of test vectors required to detect faults in VLSI circuits. The algorithm is based on the circuit's primary input cone (N) and fanout (F) values to decide the location and number of partitions. Particle Swarm Optimization (PSO) is used to determine the optimal values of N and F to minimize the number of test vectors as well as the number of partitions. The proposed algorithm has been applied to the ISCAS'85 benchmark circuits and the results obtained are compared with those from the I-PIFAN partitioning approach.

1. Introduction

The advancement in VLSI semiconductor technology has led to a phenomenal development in electronic systems. With the reduction in device sizes, a very large number of transistors can fit onto a single chip. However, as the chip density increases, the probability of defects occurring in a chip increases as well. The quality, reliability and cost are directly related to the intensity of product testing. As a result, testing has become a major concern. All possible irredundant faults can be found only through exhaustive testing of a circuit. However, a digital circuit with N inputs requires 2^N test vectors for exhaustive testing. As N increases, the applicability of all 2^N test vectors becomes impractical and alternative approaches are needed.

The pseudoexhaustive testing approach has been proposed by McCluskey [1]. In this method, the circuit is partitioned into a number of subcircuits, which

considerably reduces the required number of test vectors. Archambeau [2] demonstrated that when single-output partitions are used, all detectable combinational faults within each partition are detected by such a test. The PIFAN algorithm for partitioning was developed by Shaer [3]. It is based on the Primary Input cone and FANout values of each node in the circuit. This algorithm was automated and improved upon later by the authors and called I-PIFAN [4]. This method has been found to be more effective than the previously suggested approaches such as partition design methodology and simulated annealing [5-6].

A recently developed algorithm known as particle swarm optimization (PSO) that emerges and allies itself to evolutionary algorithms based on simulation of the behavior of a flock of birds or school of fish, has proven to have great potential for optimization problems [7-10]. Swarm algorithms differ from evolutionary algorithms most importantly in both metaphorical explanation and how they work. PSO is able to find optimal solutions for both single-objective and multi-objective problems [11-12]. In this paper, PSO is used to determine the optimal values of N and F for the I-PIFAN partitioning algorithm and the authors refer to this approach as PSO-PIFAN.

The paper is organized as follows. The partitioning algorithm is presented in Section 2. The PSO algorithm is described in Section 3. Section 4 presents PSO based partitioning and the fitness function formulation. Section 5 describes the results obtained with PSO-PIFAN and comparison with the I-PIFAN for the standard ISCAS'85 benchmark circuits. The conclusion and future work are given in Section 6.

2. Partitioning Approach

Circuit partitioning is carried out using the I-PIFAN algorithm [4]. This algorithm is based on reducing the primary input cones (PI) in the circuit. The initial step is to form a gate and input matrix representation of the circuit from its Verilog netlist file. The gate matrix is formed one column at a time, with each subsequent column containing all gates whose inputs come only from the wires or gates in the immediately preceding column. Each gate matrix element is a number, from 0-9, representing a particular type of gate (i.e. AND, OR, etc.), or a wire, or no gate. The input matrix represents the connection of gates in the gate matrix. Since the maximum number of inputs per gate differ greatly for the ISCAS'85 circuits, the number of rows of the input matrix are dynamically configured as the number of rows in the gate matrix multiplied by the maximum fanin of the circuit.

The partitioning is carried out by adding a primary input and a primary output to the circuit, provided the output of the node being partitioned is not already a primary output. If the node is an inverter or buffer, it is not partitionable and the inputs have to be traced back until a partitionable node is found. There are two phases in the algorithm. Phase-I involves partitioning the nodes with fanout (FO) values greater than or equal to F and primary input cone (PI) greater than one. In Phase-II, the PI value of each node is compared with N . If it is greater than N , then the node is partitioned. The primary input and fanout values at each node are recalculated every time a partition is made. The value of N should be greater than or equal to the maximum fanin of the circuit. The pseudocode and the flowchart (Figure 1) for the I-PIFAN algorithm are given below:

begin

- Start at the beginning of the node list.
- Move through the nodes until the FO value of the node is greater than or equal to F and its PI value is greater than one.
- If the node is an inverter or buffer, trace back through its inputs until a partitionable node is found.
- Make a partition at the output of the node by adding a primary input and a primary output, unless the node's output is a primary output.

- Update the PI and FO values of all nodes at, and following, the partitioned node in the circuit.
- Repeat steps from b) to e) until the end of the node list.
- Go back to the beginning of node list.
- Move through the nodes until the PI value of the node is greater than N .
- Find the input of the current node with the greatest PI value. If the PI values are equal, select the first input.
- Perform steps c) through e).
- Repeat steps from h) to j) until the end of the node list.

end

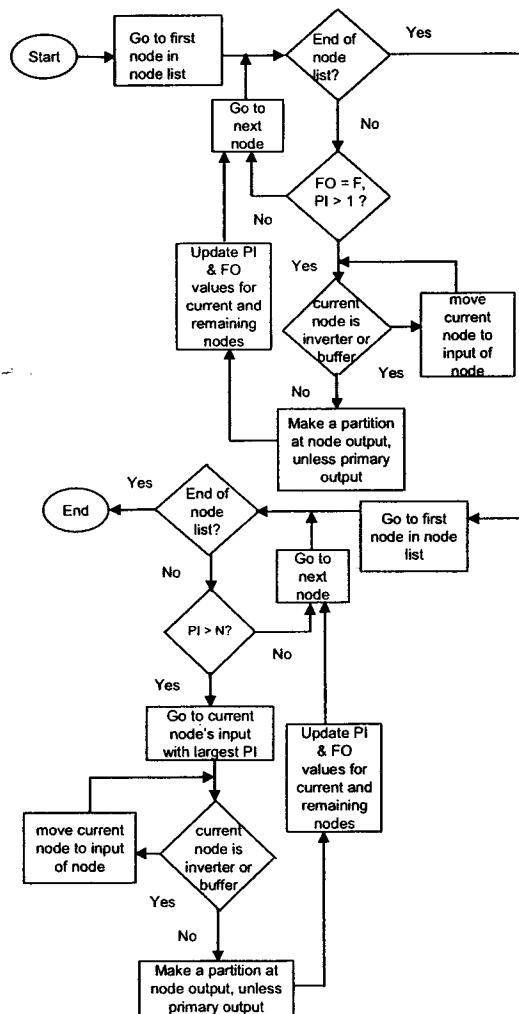


Figure 1. I-PIFAN algorithm

An example circuit is shown in Figure 2. It has 9 primary inputs and 3 primary outputs. The number of test vectors required for exhaustive testing of the circuit is therefore 512. The primary input and fanout values for the nodes are shown adjacent to each gate. Selecting N to be 3 and F to be 4, the circuit can be partitioned with the I-PIFAN algorithm described above. Circuits obtained after Phase-I and Phase-II partitioning are shown in Figures 3 and 4, respectively.

The number of test vectors required for pseudoexhaustive testing of the partitioned circuit can be calculated by summing the 2^Z test vectors needed for each primary output, where Z is equal to the PI cone of a primary output's node. Hence, the final number of test vectors required would be 36.

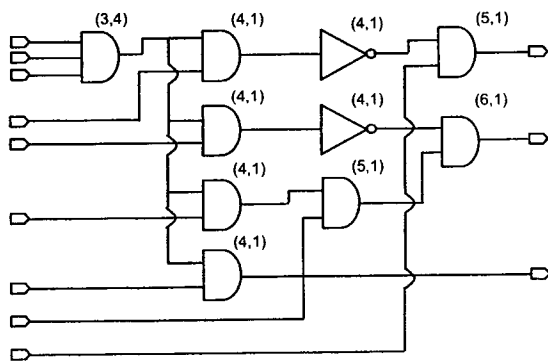


Figure 2. Example Circuit

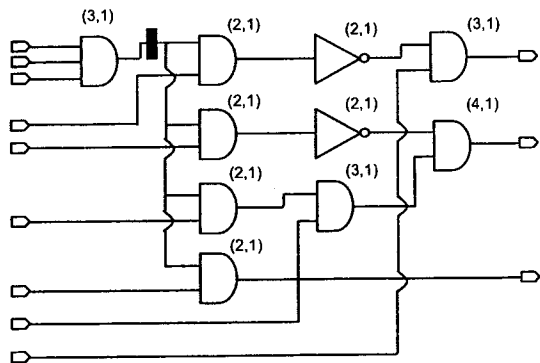


Figure 3. Circuit after Phase-1 partitioning

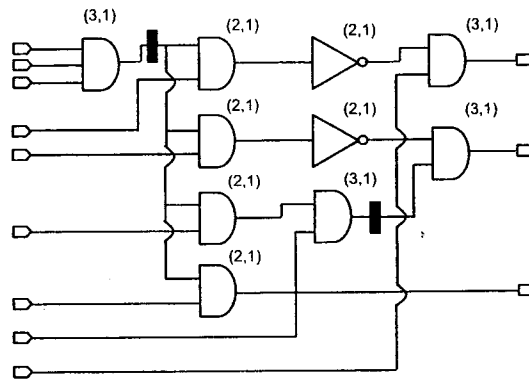


Figure 4. Circuit after Phase-2 partitioning

3. Particle Swarm Optimization

PSO is an evolutionary computation technique developed by Kennedy and Eberhart [7-8]. It is motivated by a social behavior of organisms such as bird flocking, fish schooling, and swarm theory. In PSO, the potential solutions, called "particles", fly around in a multi-dimensional search space, to find out an optimal or sub-optimal solution by competition as well as cooperation among themselves. The PSO technique starts initially with a population of random solutions. Each particle is given a random velocity and is flown through the d -dimensional problem space. The position (x_{id}) and velocity (v_{id}) of each particle i is updated based on its previous velocity, the previous *best particle's* location (p_{id} or $pbest$), and the previous *global best particle's* location (p_{gd} or $gbest$) [7]. The basic concept of PSO lies in accelerating each particle towards its $pbest$ and $gbest$ locations at each time step. The velocity and position updates are given by (1) and (2), respectively.

$$v_{id} = w \times v_{id} + c_1 \times rand_1() \times (p_{id} - x_{id}) + c_2 \times rand_2() \times (p_{gd} - x_{id}) \quad (1)$$

$$x_{id} = x_{id} + v_{id} \quad (2)$$

Where w is the inertia weight and is usually set to a value in the range of 0.5 to 1; c_1 and c_2 are cognitive and social acceleration constants respectively, and are usually set to 2; $rand_1$ and $rand_2$ are uniform random numbers between 0 and 1. The optimal choice of values for w , c_1 , and c_2 can cause the PSO

algorithm to carry out better exploration and exploitation of the search space [13].

4. PSO for Partitioning

The PSO algorithm is used to determine the optimal values of N and F , such that both the number of partitions and number of test vectors is minimized. The two objectives are represented in the form of fitness functions f_P and f_T , where f_P represents the number of partitions and f_T the number of test vectors. Generally, the number of test vectors can be reduced by increasing the number of partitions. However, this introduces hardware overhead and increases critic path delay, so it is desirable to find an optimal number of partitions (Par) and test vectors (TVs). In order to use PSO to evaluate the different possible solutions, the authors have used the fitness function given in (3).

$$f = f_P^2 \times \frac{\sqrt{f_{T,initial}}}{100} + f_T \quad (3)$$

$f_{T,initial}$ is the number of test vectors required to do the exhaustive testing of the circuit (without partitioning) and is equal to 2^{PI} (PI is the original number of primary inputs). The fitness function f in (3) is based on the assumption that the optimal value of the number of test vectors lies near the square root of $f_{T,initial}$. This assumption has been found experimentally to be reasonable by the authors for all the ISCAS'85 benchmark circuits that were tested.

The pseudocode and flowchart (Figure 5) for the PSO-PIFAN algorithm is given below:

begin

$t \leftarrow 0$

a) initialize the PSO particles in two dimensions (N and F).

b) partition the circuit based on I-PIFAN.

c) evaluate the fitness f in (3).

d) select the $pbest$ and $gbest$.

e) end if termination condition true.

f) $t \leftarrow t + 1$.

g) update the velocity and position of each particle using (1) and (2) respectively.

h) repeat b) to e).

end

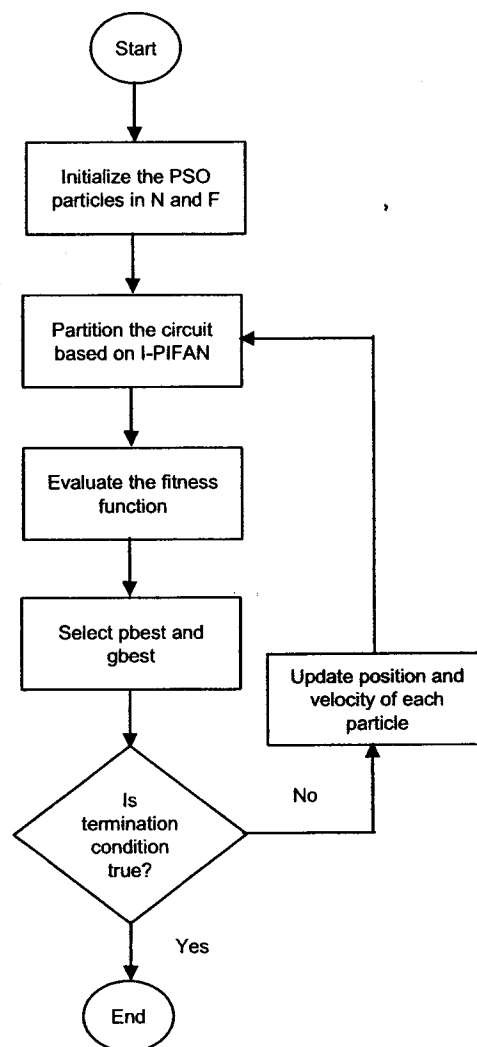


Figure 5. PSO-PIFAN algorithm

Table 1. Comparison of No. of partitions and Test Vectors for ISCAS'85 benchmark circuits using I-PIFAN and PSO-PIFAN

ISCAS'85 Circuit	# of PI	# of FO	I-PIFAN				PSO-PIFAN				# of TVs without partitions
			N	F	# of Par	# of TVs	N	F	# of Par	# of TVs	
c432	36	7	20	8	18	3.97e+06	17	13	23	312124	6.87e+10
c499	41	32	14	12	8	147456	15	11	8	147456	2.20e+12
c880	60	26	14	8	20	186204	14	8	20	186204	1.15e+18
c1355	41	32	14	12	8	147456	16	10	8	147456	2.20e+12

5. Results

The effectiveness of the algorithm is demonstrated using the simplified case study example presented in [4]. The circuit has nine primary inputs and three primary outputs. Hence, the number of test vectors required to exhaustively test the circuit is 512. The partitioning based on the I-PIFAN approach resulted in 4 partitions and 46 test vectors. The PSO-PIFAN algorithm resulted in the same number of partitions and test vectors.

It is worth pointing out that different combinations of N and F can yield the same number of partitions and test vectors for a given circuit. Also, the same number of partitions can yield a different number of test vectors depending on the locations at which the partitions are made. However, there is no single global optimum solution possible for the problem. It varies according to the application and the requirements of the user. The hardware overhead due to the partitions, testing time, and speed have to be taken into consideration to select the most optimal values. PSO is able to find several optimal values of N and F , some of them yielding fewer test vectors and others resulting in fewer partitions. But the global optimum for PSO depends on the fitness function defined in (3). For example, for the ISCAS'85 benchmark circuit c432, PSO-PIFAN determined the optimal values of N and F to be 17 and 13, respectively, which resulted in 23 partitions and $3.12e+05$ test vectors. Another combination of N and F was 18 and 7, respectively, resulting in 22 partitions and $6.98e+05$ test vectors. The I-PIFAN approach generated 18 partitions and $3.97e+06$ test vectors with N and F equal to 20 and 8, respectively. Based on the reduced number of test vectors and

corresponding partitions, PSO-PIFAN yields better results than I-PIFAN. Similarly, there are several other sub-optimal solutions found by PSO-PIFAN. The results for the c499, c880, and c1355 ISCAS'85 benchmark circuits matched exactly with the I-PIFAN results [4] in terms of partitions and test vectors, and are shown in Table 1.

The PSO-PIFAN algorithm was run with only 10 particles and was able to find the above solutions in less than 10 iterations. A larger number of PSO particles and additional iterations could result in better solutions. This remains to be tested.

6. Conclusion

In this paper, particle swarm optimization (PSO) is applied to the circuit partitioning problem. The I-PIFAN algorithm is used as the partitioning approach and the parameters N and F are determined using PSO. The PSO-PIFAN algorithm optimizes the number of test vectors and the number of partitions simultaneously. The results with the PSO-PIFAN algorithm show that it is effective in determining the optimal values of N and F . Results also indicate that PSO-PIFAN is more efficient than the I-PIFAN algorithm. I-PIFAN needs to test all combinations of N and F exhaustively within a specified range. On the other hand, PSO-PIFAN searches all combinations of N and F simultaneously, and without necessitating a specified range. Hence, PSO-PIFAN searches the solution space and uses its memory to accelerate the PSO particles towards the global best solution faster.

The other ISCAS'85 benchmark circuits remain to be tested with the PSO-PIFAN algorithm. The current PSO-PIFAN algorithm did not consider minimizing the critical path delay introduced due to partitioning. The fitness function can be modified to

incorporate this. The run times for the different methods need to be compared as part of future work. The future work involves applying the Pareto front optimality concept for solving the partitioning problem [14-15]. Furthermore, sequential circuits can be reduced to combinational circuits and the PSO-PIFAN technique can be applied to them as well.

Acknowledgment

The authors gratefully acknowledge the support from the National Science Foundation under CAREER grant ECS # 0348221.

References

- [1] E.J. McCluskey and S. Bozorgui-Nesbat, "Design for Autonomous Test", *IEEE Trans. Comput.*, vol. C-30, pp. 866-875, Nov. 1981.
- [2] E.C. Archambeau and E. J. McCluskey, "Fault Coverage of Pseudo-Exhaustive Testing", *Digest, Int. Conf. Fault Tolerant Comput.*, pp. 141-145, Orlando, FL, 1984.
- [3] B. Shaer, D. Landis and S. Al-Arian, "Partitioning Algorithm to Enhance Pseudoexhaustive Testing of Digital VLSI Circuits", *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, Vol. 8, Issue 6, pp. 750-754, Dec. 2000.
- [4] B. Shaer and Khaled Dib, "An Efficient Partitioning Algorithm of Combinational CMOS Circuits", *Proc. of IEEE Comp. Society Annual Symp. on VLSI*, pp. 142-146, 2002.
- [5] S. Al-Arian and R. Bolling, "Improving the Testability of VLSI Circuits through Partitioning", *IEEE Symp. on Circuits and Systems*, Vol. 4, pp. 199-202, 1994.
- [6] I. Shperling and E. J. McCluskey, "Circuit Segmentation for Pseudo-Exhaustive Testing via Simulated Annealing", *Proc. Int. Test Conf.*, pp. 58-65, 1987.
- [7] J. Kennedy and R. Eberhart, *Swarm Intelligence*, San Diego, USA, Academic Press, 2001.
- [8] J. Kennedy and R. Eberhart, "Particle Swarm Optimization", in *Proc. IEEE Int. Conference on Neural Networks*, vol. IV, pp. 1942-1948, Perth, Australia, 1995.
- [9] G.K. Venayagamoorthy and V.G. Gudise, "Swarm Intelligence for Digital Circuits Implementation on Field Programmable Gate Arrays Platforms", *2004 NASA/DoD Conference on Evolvable Hardware*, June 24 - 26, pp. 83 - 86, 2004.
- [10] V. G. Gudise and G.K. Venayagamoorthy, "FPGA Placement and Routing Using Particle Swarm Optimization", *IEEE Comp. Society Annual Symp. on VLSI*, February 19 - 20, pp. 307 -308, 2004.
- [11] M. Clerc and J. Kennedy, "The particle swarm - explosion, stability, and convergence in a multidimensional complex space", *IEEE Trans. on Evolutionary Computation*, Vol. 6, Issue 1, pp. 58-73, Feb. 2002.
- [12] C.A.C. Coello, G.T. Pulido and M.S. Lechuga, "Handling multiple objectives with particle swarm optimization", *IEEE Trans. on Evolutionary Computation*, Vol. 8, Issue 3, pp. 256-279, June 2004.
- [13] S. Doctor, G.K. Venayagamoorthy and V.G. Gudise, "Optimal PSO for Collective Robotic Search Applications", *IEEE Congr. on Evolutionary Computation*, pp. 1390 - 1395, June 20 - 23, 2004.
- [14] U. Baumgartner, Ch. Magele and W. Renhart, "Pareto Optimality and Particle Swarm Optimization", *IEEE Trans. on Magnetics*, Vol. 40, Issue 2, pp. 1172-1175, March 2004.
- [15] Hu Xiaohui and R. Eberhart, "Multiobjective Optimization Using Dynamic Neighborhood Particle Swarm Optimization", *Proc. of the 2002 Congr. on Evolutionary Computation*, Vol. 2, pp. 1677-1681, May 12-17, 2002.