



Particle swarm-based optimal partitioning algorithm for combinational CMOS circuits

Ganesh K. Venayagamoorthy*, Scott C. Smith, Gaurav Singhal

Real-Time Power and Intelligent Systems Laboratory, Department of Electrical and Computer Engineering, University of Missouri-Rolla, MO 65409, USA

Received 23 December 2005; received in revised form 2 June 2006; accepted 28 June 2006

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 maximum primary input cone size (N) and minimum 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, the number of partitions, and the increase in critical path delay due to the added partitions. The proposed algorithm has been applied to the ISCAS'85 benchmark circuits and the results are compared to other partitioning approaches, showing that the PSO partitioning algorithm produces similar results, approximately one-order of magnitude faster.

© 2006 Elsevier Ltd. All rights reserved.

Keywords: Circuit partitioning; Particle swarm optimization; VLSI testing; PSO-PIFAN

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 purely combinational digital circuit with M inputs requires 2^M test vectors for exhaustive testing, while a sequential circuit may require even more test vectors. As M increases, the applicability of all 2^M test vectors becomes impractical and alternative approaches are needed.

The pseudoexhaustive testing approach has been proposed by McCluskey (McCluskey and Bozorgui-Nesbat, 1981). In this method, the circuit is partitioned into a

number of subcircuits, which considerably reduces the required number of test vectors. Archambeau (Archambeau and McCluskey, 1984) 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 et al., 2000a, b). It is based on the primary input (PI) cone and FANout (FO) values of each node in the circuit. This algorithm was automated and improved upon later by the authors and called I-PIFAN (Shaer and Dib, 2002). This method has been found to be more effective than the previously suggested approaches such as partition design methodology and simulated annealing (Al-Arian and Bolling, 1994; Shperling and McCluskey, 1987). I-PIFAN seeks to minimize the number of test vectors, the number of partitions, and the increase in critical path delay.

I-PIFAN is based on two constant inputs, maximum node fanin size, N , and minimum partitioning FO value, F . A node is partitionable if it is not an inverter or buffer and has a PI value greater than one. If a node's FO value is greater than or equal to F and the node is partitionable, a partition is inserted. The inputs of any node whose PI value

*Corresponding author. Tel.: +1 573 3416641; fax: +1 573 3414532.
 E-mail address: gkumar@ieee.org (G.K. Venayagamoorthy).

is greater than N are processed for partitioning to reduce the node's PI cone. The algorithm performs an exhaustive test of all combinations of constants, N and F , over some specified range, fully partitioning the circuit for each combination of N and F , and producing a table of possible circuit partitions, from which the best result can be selected.

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 (Kennedy and Eberhart, 2001; Kennedy and Eberhart, 1995; Venayagamoorthy and Gudise, 2004; Gudise and Venayagamoorthy, 2004). 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 (Clerc and Kennedy, 2002; Coello et al., 2004). In this paper, PSO is used to efficiently determine the optimal values of N and F for the I-PIFAN partitioning algorithm, without necessitating an exhaustive test or limiting the values to a specified range, resulting in an order of magnitude speedup; 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 using PSO-PIFAN and compares with those obtained from the I-PIFAN approach, for the standard ISCAS'85 benchmark circuits (Brglez and Fujiwara, 1985). Finally, the conclusions and future work are presented in Section 6. The nomenclature used throughout this paper is summarized in Table 1.

2. Partitioning approach

Circuit partitioning is carried out using the I-PIFAN algorithm (Shaer and Dib, 2002), which is based on reducing nodes' PI cone size and FO values. The initial step is to form a gate and input matrix representation of the circuit from its Verilog netlist file. The next two subsections describe this circuit representation and the partitioning methodology used.

2.1. Circuit representation

The circuit is represented in the form of a gate matrix, G , and an input matrix, I , where the gate matrix represents the type of gate present at each node and the input matrix represents the connection of gates. The gate matrix is formed one column at a time, with each subsequent column containing all gates whose inputs are all from the wires or gates in the immediately preceding column; therefore the number of columns in G , C_G , is equal to the maximum number of gates on any path from circuit input to output.

Table 1
Nomenclature

| Symbol | Description |
|-------------------|--|
| p_{id}/p_{best} | Previous best position of the particle |
| P_{gd}/g_{best} | Best position among all p_{best} of the particle |
| x_{id} | Position of the particle |
| v_{id} | Velocity of the particle |
| w | Inertia weight |
| c_1 | Cognition acceleration constant |
| c_2 | Social acceleration constant |
| C_G | Number of Columns in G |
| C_I | Number of Columns in I |
| f | Fitness function |
| F | Minimum FO value |
| f_C | Represents percentage increase in critical path in f , as compared to the original circuit |
| FO | FANout |
| f_P | Represents Par in f |
| F_R | Number of values in specified range for F |
| f_T | Represents TVs in f |
| G | Gate matrix |
| I | Input matrix |
| I-PIFAN | Improved primary input and FANout-based partitioning approach |
| I_{PSO} | Number of PSO iterations before optimal solution is found |
| N | Size of primary input cone |
| N_R | Number of values in specified range for N |
| P | Number of particles in the swarm |
| Par | Number of partitions |
| PI | Primary input |
| PIFAN | Primary input and FANout-based partitioning approach |
| PO | Primary output |
| PSO | Particle swarm optimization |
| PSO-PIFAN | PSO-based PIFAN |
| rand | Uniform random number between 0 and 1 |
| R_G | Number of rows in G |
| R_I | Number of rows in I |
| TVs | Number of test vectors |

Table 2
Gate matrix gate representation

| Gate number | Type |
|-------------|---------|
| 0 | No gate |
| 1 | AND |
| 2 | OR |
| 3 | XOR |
| 4 | INV |
| 5 | Wire |
| 6 | NAND |
| 7 | NOR |
| 8 | XNOR |
| 9 | Buffer |

The number of rows in G , R_G , is equal to the maximum cut of the circuit (i.e. the maximum of the number of circuit inputs or the number of wires at the output of any column in G). Each gate matrix element is a number, from 0–9, representing a particular type of gate, a wire, or no gate. The corresponding numbers used for representing the gates are listed in Table 2. The example circuit in Fig. 1 is

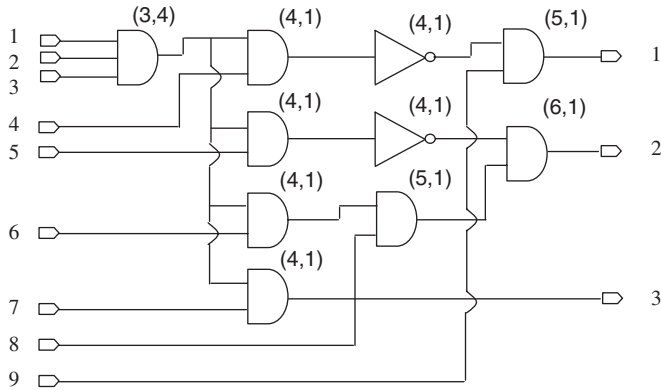


Fig. 1. Example circuit with primary input (PI) and fanout (FO) values shown next to each gate.

represented by the following gate matrix, G , where $C_G = 4$, due to the critical path of four gates, and $R_G = 9$, due to the maximum cut of the circuit, which occurs at the input in this case (i.e. there are 9 inputs). Most non-trivial circuits where pseudoexhaustive testing would actually be used would likely result in the maximum cut being located somewhere in the middle of the circuit, such that at least one column in the gate matrix would not contain any zeroes.

$$G = \begin{bmatrix} 1 & 1 & 4 & 1 \\ 5 & 1 & 4 & 1 \\ 5 & 1 & 1 & 5 \\ 5 & 1 & 5 & 0 \\ 5 & 5 & 5 & 0 \\ 5 & 5 & 0 & 0 \\ 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

The input matrix, I , represents the connection between the gates in G , such that the number of columns in I , C_I , is equal to C_G . Since the maximum number of inputs per gate differs greatly from circuit to circuit, the number of rows in I , R_I , is configured as R_G multiplied by the maximum fanin of any gate in the circuit. Each number in I corresponds to a wire or output of a specific gate from the preceding column of G (the numbers in the first column represent the circuit inputs). Therefore, the possible non-zero values for column Z are $[R_G \times (Z - 1) + 1, R_G \times Z]$. The legal values for columns 2 to C_I are further restricted to the non-zero entries in column $(Z-1)$ in G , and to the number of circuit inputs for column 1. A value of zero in I corresponds to no connection for the corresponding gate's input.

For the example circuit in Fig. 1 and its corresponding gate matrix, G , shown above, the input matrix, I , is represented as shown below. Since the maximum fanin of any gate in this circuit is 3, $R_I = 3 \times R_G = 27$, such that each row in G corresponds to 3 rows in I . The first three rows in column 1 of I correspond to the inputs of the AND

gate in column 1, row 1 in G , showing that this gate's inputs are circuit inputs 1–3. Rows 4–6 in column 2 in I correspond to the inputs of the second to top AND gate in column 2 of Fig. 1, showing that its inputs are the output of the 3-input AND gate from column 1 of $G(10)$ and the second wire in column 1 of $G(12)$, which corresponds to circuit input 5. Row 6 in column 2 in I is 0, meaning that the corresponding gate only has two inputs.

$$I = \begin{bmatrix} 1 & 10 & 19 & 28 \\ 2 & 11 & 0 & 32 \\ 3 & 0 & 0 & 0 \\ 4 & 10 & 20 & 29 \\ 5 & 10 & 21 & 31 \\ 6 & 10 & 22 & 0 \\ 7 & 15 & 24 & 0 \\ 8 & 16 & 0 & 0 \\ 9 & 0 & 0 & 0 \\ 10 & 0 & 0 & 0 \\ 11 & 0 & 0 & 0 \\ 12 & 0 & 0 & 0 \\ 13 & 0 & 0 & 0 \\ 14 & 0 & 0 & 0 \\ 15 & 0 & 0 & 0 \\ 16 & 0 & 0 & 0 \\ 17 & 0 & 0 & 0 \\ 18 & 0 & 0 & 0 \\ 19 & 0 & 0 & 0 \\ 20 & 0 & 0 & 0 \\ 21 & 0 & 0 & 0 \\ 22 & 0 & 0 & 0 \\ 23 & 0 & 0 & 0 \\ 24 & 0 & 0 & 0 \\ 25 & 0 & 0 & 0 \\ 26 & 0 & 0 & 0 \\ 27 & 0 & 0 & 0 \end{bmatrix}$$

2.2. Partitioning

The partitioning is carried out by adding a PI and a primary output (PO) to the circuit, provided the output (PO) of the node being partitioned is not already a PO. If the node is an inverter or buffer, it is not partitionable and the inputs need to be traced back until a partitionable node is found. There are two phases in the partitioning algorithm. Phase-I involves partitioning all nodes with a FO value greater than or equal to F and PI cone greater than one. In Phase-II, the PI value of each node is compared with N , such that the node is partitioned if

($PI > N$) and the node is not an inverter or buffer. The PI and FO values of each node are recalculated every time a partition is added. The value of N should be greater than or equal to the maximum fanin of the circuit. The flowchart for the I-PIFAN algorithm is given in Fig. 2, and the pseudocode is described below:

begin

- (a) Start at the beginning of the node list (Phase I).
- (b) Move through the node list until the FO value of the node is greater than or equal to F and its PI value is greater than one.
- (c) If the node is an inverter or buffer, trace back through its inputs until a partitionable node is found.
- (d) Make a partition at the output of the node by adding a primary input, and a primary output if the node's output is not already a primary output.
- (e) Update the PI and FO values of all nodes at, and following, the partitioned node in the circuit.
- (f) Move to the next node in the list and goto step (b), if more nodes remain.
- (g) Go back to the beginning of the node list (Phase II).
- (h) Move through the node list until the PI value of the node is greater than N .
- (i) Find the input of the current node with the greatest PI value. If the PI values are equal, select the first input.
- (j) Perform steps (c)–(e).
- (k) Move to the next node in the list and goto step (h), if more nodes remain.

end

The example circuit shown in Fig. 1 has 9 PIs and 3 POs. The number of test vectors required for exhaustive testing of the circuit is therefore $2^9 = 512$. The PI and FO values, respectively, for the nodes are shown next 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 -II partitioning are shown in Figs. 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 PO, where Z is equal to the PI cone of a PO node. Hence, the final number of test vectors required for the circuit in Fig. 4 is 36 (i.e. $2^3 + 2^3 + 2^2 + 2^3 + 2^3$).

3. PSO

PSO is an evolutionary computation technique developed by (Kennedy and Eberhart, 2001; Kennedy and Eberhart, 1995). It is motivated by the 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 discover an optimal, or sub-optimal, solution by competition as well as cooperation among themselves. The system initially has

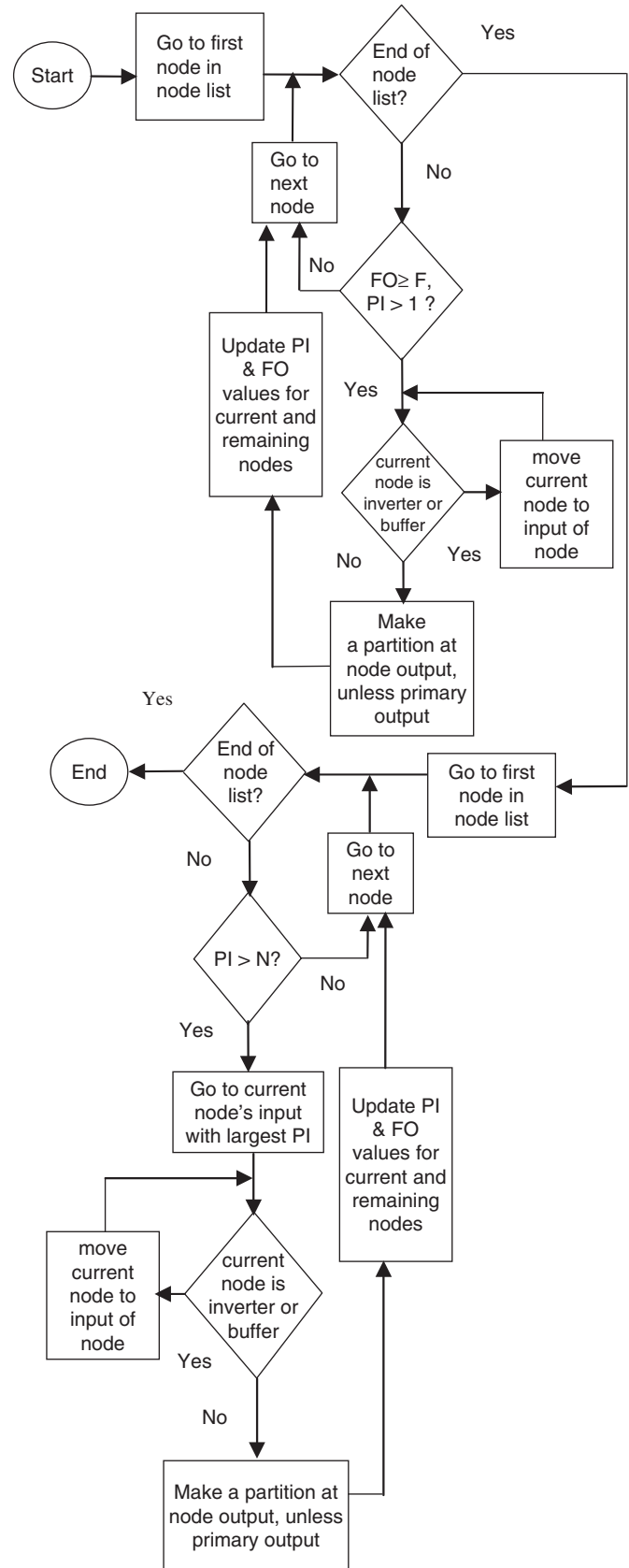


Fig. 2. I-PIFAN algorithm.

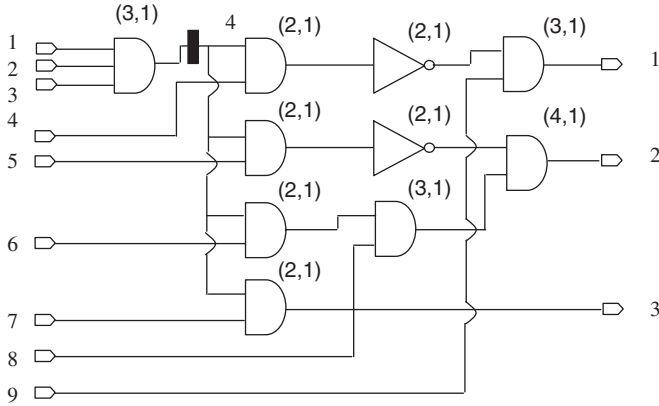


Fig. 3. Example circuit after Phase-I partitioning. A partition is added and the primary output node labeled 4 is added.

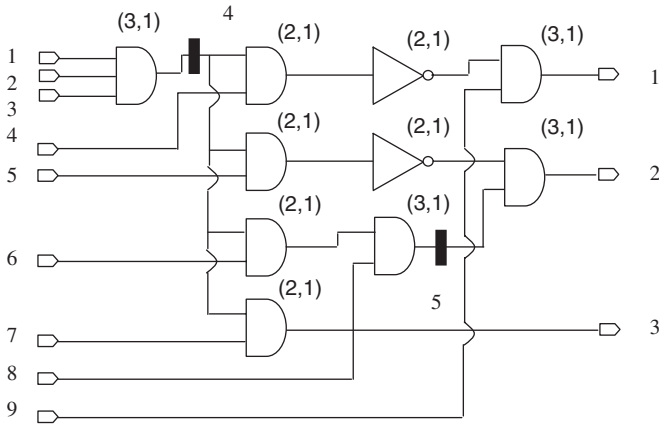


Fig. 4. Example circuit after Phase-II partitioning. A partition is added and the primary output node labeled 5 is added.

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 in dimension d is updated based on its previous velocity, the previous best particle location (p_{id} or p_{best}), and the previous global best location of a particle in the population (p_{gd} or g_{best}). The basic concept of PSO lies in accelerating each particle towards its p_{best} and g_{best} locations at each time step. The canonical velocity and position update equations are given in (1) and (2), respectively.

$$v_{id} = w \times v_{id} + c_1 \times \text{rand}_1() \times (p_{id} - x_{id}) + c_2 \times \text{rand}_2() \times (p_{gd} - x_{id}). \quad (1)$$

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

w is the inertia weight (Shi and Eberhart, 1998) and is usually set to a value in the range of 0.5–1. c_1 and c_2 are the cognitive and social acceleration constants, respectively, and are usually set to 2. 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 (Doctor

et al., 2004). rand_1 and rand_2 are uniform random numbers between 0 and 1.

4. PSO-PIFAN

The PSO algorithm in this paper is used to determine the optimal values of N and F , based on the value of a fitness function, f , which tells how well the solution achieves the specific optimization criteria. This paper analyzes two different cases. The first case considers the minimization of the number of partitions and number of test vectors. However, this does not take into consideration the increase in critical path delay due to hardware overhead introduced in the circuit from the partitioning. Therefore, the second case tries to minimize the number of partitions, the number of test vectors, and the critical path increase. The flowchart for the PSO-PIFAN algorithm is given in Fig. 5, and the pseudocode follows:

begin

$t \leftarrow 0$

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

(b) partition the circuit using I-PIFAN.

(c) evaluate the fitness function, f .

(d) select the p_{best} and g_{best} .

(e) end if termination condition true.

(f) $t \leftarrow t+1$.

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

(h) goto step (b).

end

The following two subsections describe the PSO-PIFAN algorithm for cases 1 and 2, with and without critical path consideration, respectively. The corresponding fitness functions are also discussed.

4.1. Case 1—without critical path consideration

The objective of PSO-PIFAN is to minimize both the number of partitions and number of test vectors. These 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 additional hardware overhead and may further increase critical path delay, so it is desirable to find the smallest number of partitions that minimizes the number of test vectors. In order to use PSO to evaluate the different solutions, the authors have used the fitness function in (3).

$$f_1 = f_P^2 \times \frac{\sqrt{f_{T,\text{initial}}}}{100} + f_T. \quad (3)$$

$f_{T,\text{initial}}$ is the number of test vectors required to exhaustively test the non-partitioned circuit, and is equal

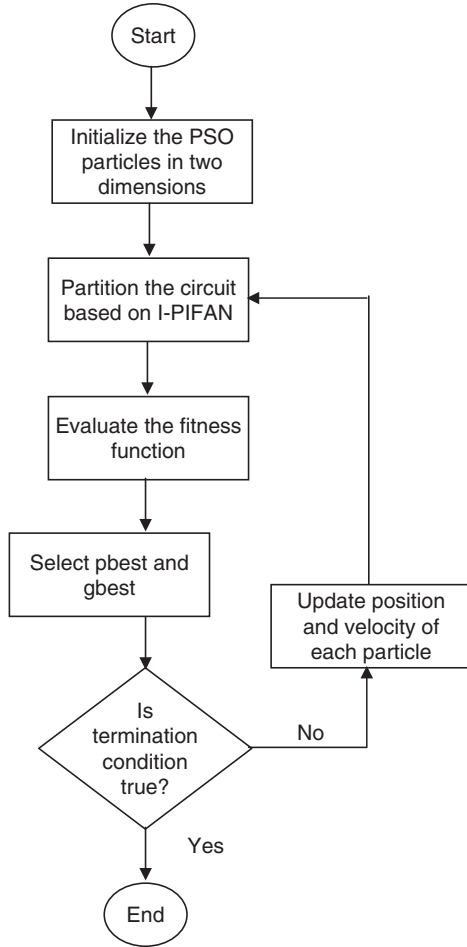


Fig. 5. PSO-PIFAN algorithm.

function, f_3 :

$$f_3 = (f_P^2 + f_C^2) \times \frac{\sqrt{f_{T,initial}}}{100} + f_T. \quad (5)$$

The results for PSO-PIFAN are discussed in Section 5 and compared with the I-PIFAN results.

4.3. Computational complexity and convergence

Let $O(I-PIFAN)$ be the running time of the I-PIFAN algorithm for one value of N and F . Since the I-PIFAN algorithm performs an exhaustive test of all combinations of N and F over a specified range, the running time for I-PIFAN is therefore: $O(N_R \times F_R \times O(I-PIFAN))$, where N_R and F_R are the number of values in the specified range for N and F , respectively. Since all combinations within the specified range are tested, I-PIFAN will find the optimal solution that lies within the specified range. However, solutions outside of the specified range will not be considered, so the optimal solution would not be found if it was not within the specified range.

The running time for PSO-PIFAN is based on the number of particles, P , in the swarm, and the number of PSO iterations, I_{PSO} , before the optimal solution is found. For each PSO iteration, the I-PIFAN algorithm is run on each particle using its newly updated values of N and F ; therefore, the running time for PSO-PIFAN is: $O(P \times I_{PSO} \times O(I-PIFAN))$. Hence, as long as $(P \times I_{PSO}) < (N_R \times F_R)$, PSO-PIFAN will have a faster running time. This inequality would tend to be true for most reasonable values of P , since PSO-PIFAN provides for a directed search of the solution space, whereas I-PIFAN requires an exhaustive search. This assumption is further supported by the results. Clerc and Kennedy have addressed the stability and convergence issues of PSO in Clerc and Kennedy (2002). Furthermore, PSO-PIFAN does not require a specified range of N and F values, so it will find the optimal solution wherever it exists, not only if it is contained within a given range.

5. Results

The effectiveness of the PSO-PIFAN algorithm is demonstrated using the simplified case study example presented in (Shaer and Dib, 2002). The circuit has 9 PIs and 3 POs. Hence, the number of test vectors required to exhaustively test the circuit is $2^9 = 512$. 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. For the example circuit in Fig. 1, PSO-PIFAN found the optimal values of N and F to be 3 and 4, respectively, and reduced the number of test vectors from 512 to 36.

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

to 2^{CI} , where CI is the number of circuit inputs. The fitness function f_1 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.

4.2. Case 2—with critical path consideration

A significant increase in critical path due to partitioning of the circuit can be undesirable. Here, two fitness functions f_2 and f_3 are considered. The fitness function in (3) is modified in (4) for the PSO-PIFAN algorithm to minimize the percentage increase in critical path along with the number of test vectors.

$$f_2 = f_C^2 \times \frac{\sqrt{f_{T,initial}}}{100} + f_T. \quad (4)$$

f_C represents the percentage increase in critical path as compared to the circuit without partitioning. However, the number of partitions and number of test vectors, as well as the percentage increase in critical path, can all be simultaneously optimized using the following fitness

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 circuit designer. The hardware overhead (due to added partitions), testing time (proportional to number of test vectors), and circuit speed (added partitions may increase critical path) are traded off to select the desired solution.

The PSO–PIFAN algorithm was run in this study with only 10 particles and was able to find the optimal solutions. A larger population of PSO particles could result in convergence to the optimal solution in fewer iterations although the runtime per iteration would increase accordingly.

5.1. Case 1—without critical path consideration

PSO–PIFAN is able to find several optimal values of N and F , some of them yielding fewer test vectors and others resulting in fewer partitions; however, the global optimum depends on the fitness function defined in (3). For example, PSO–PIFAN determined the optimal values of N and F to be 17 and 13, respectively, for the ISCAS'85 benchmark circuit c432, which resulted in 23 partitions (Par) and 3.12×10^5 test vectors (TVs). Another combination of N and F was 18 and 7, respectively, resulting in 22 partitions and 6.98×10^5 TVs. The I-PIFAN approach generated 18 partitions and 3.97×10^6 TVs with N and F equal to 20 and 8, respectively. Based on the fitness function chosen in (3), the fitness for the PSO–PIFAN results is found to be higher than for the I-PIFAN results. 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 (Shaer and Dib, 2002) in terms of partitions and TVs, as shown in Table 3.

The runtimes for I-PIFAN and PSO–PIFAN are compared in Table 4, which shows that PSO–PIFAN finds the optimal solution approximately one-order of magnitude faster than the I-PIFAN algorithm. Furthermore, the most optimal solution was always found in less than 10 iterations, so the runtime is even less than the total time taken for the 10 iterations (shown in Table 4).

5.2. Case 2—with critical path consideration

The fitness functions in (4) and (5) gave slightly different results than those obtained for the case without critical

path consideration, as shown in Tables 5 and 6, respectively. Table 5 clearly indicates the reduction in critical path for the c432 and c880 circuits, although in the case of c880, the number of partitions had to be increased. The results with fitness functions f_2 and f_3 differ for the ISCAS'85 c880 circuit. The fitness function in (4) resulted in 33 partitions and an 8.33% increase in critical path, while the fitness function in (5) resulted in 20 partitions with a 20.83% increase in critical path. In the latter case, fewer partitions are traded off for a larger increase in critical path delay.

6. Conclusions and future work

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 minimizes the number of test vectors, the number of partitions, and the critical path, simultaneously. The results using the PSO–PIFAN algorithm show that it is effective in determining the optimal values of N and F , and that it is more efficient than the I-PIFAN algorithm, determining the optimal solution approximately one-order of magnitude faster. I-PIFAN tests 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, without necessitating a specified range. Hence, PSO–PIFAN performs a directed search of the solution space and uses its memory to accelerate the PSO particles towards the global best solution in a shorter time, and will always converge to the optimal solution, whereas I-PIFAN will only find the optimal solution if it exists within the specified ranges of N and F .

Table 4
Runtime comparison of I-PIFAN and PSO–PIFAN

| Circuit | I-PIFAN | PSO–PIFAN | |
|---------|--------------------|-----------------------|--------------------|
| | | Most optimal solution | Total time |
| c432 | 7.66×10^4 | 1.04×10^4 | 2.26×10^4 |
| c499 | 4.09×10^4 | 1.20×10^3 | 6.21×10^3 |
| c880 | 3.18×10^5 | 1.13×10^4 | 5.47×10^4 |
| c1355 | 7.95×10^4 | 2.34×10^3 | 2.12×10^4 |

Table 3
Comparison of I-PIFAN and PSO-PIFAN without critical path using fitness function f_1

| Circuit | # of PI | # of PO | 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 | 19 | 3.21×10^6 | 17 | 13 | 23 | 3.12×10^5 | 6.87×10^{10} |
| c499 | 41 | 32 | 14 | 12 | 8 | 1.47×10^5 | 15 | 11 | 8 | 1.47×10^5 | 2.20×10^{12} |
| c880 | 60 | 26 | 14 | 8 | 20 | 1.86×10^5 | 14 | 8 | 20 | 1.86×10^5 | 1.15×10^{18} |
| c1355 | 41 | 32 | 14 | 12 | 8 | 1.47×10^5 | 16 | 10 | 8 | 1.47×10^5 | 2.20×10^{12} |

Table 5
PSO–PIFAN comparison with critical path using fitness function f_2

| Circuit | PSO–PIFAN without critical path optimization | | | | | PSO–PIFAN with critical path optimization (fitness function f_2) | | | | |
|---------|--|-----|----------|---------|--------------------|---|-----|----------|---------|--------------------|
| | N | F | # of Par | % of CP | # of TVs | N | F | # of Par | % of CP | # of TVs |
| c432 | 17 | 13 | 23 | 47.06 | 3.12×10^5 | 18 | 11 | 21 | 41.18 | 9.49×10^5 |
| c499 | 15 | 11 | 8 | 9.09 | 1.47×10^5 | 16 | 11 | 8 | 9.09 | 1.47×10^5 |
| c880 | 14 | 8 | 20 | 20.83 | 1.86×10^5 | 16 | 11 | 33 | 8.33 | 1.38×10^6 |
| c1355 | 16 | 10 | 8 | 4.17 | 1.47×10^5 | 16 | 10 | 8 | 4.17 | 1.47×10^5 |

Table 6
PSO–PIFAN comparison with critical path using fitness function f_3

| Circuit | PSO–PIFAN without critical path optimization | | | | | PSO–PIFAN with critical path optimization (fitness function f_3) | | | | |
|---------|--|-----|----------|---------|--------------------|---|-----|----------|---------|--------------------|
| | N | F | # of Par | % of CP | # of TVs | N | F | # of Par | % of CP | # of TVs |
| c432 | 17 | 13 | 23 | 47.06 | 3.12×10^5 | 18 | 12 | 21 | 41.18 | 9.49×10^5 |
| c499 | 15 | 11 | 8 | 9.09 | 1.47×10^5 | 15 | 8 | 8 | 9.09 | 1.47×10^5 |
| c880 | 14 | 8 | 20 | 20.83 | 1.86×10^5 | 14 | 8 | 20 | 20.83 | 1.86×10^5 |
| c1355 | 16 | 10 | 8 | 4.17 | 1.47×10^5 | 16 | 9 | 8 | 4.17 | 1.47×10^5 |

Future work involves applying Pareto optimality concepts for solving the partitioning problem (Baumgartner et al., 2004; Hu Xiaohui and Eberhart, 2002), which will guarantee optimal solutions. In addition, comparing the PSO algorithm with other algorithms used in solving combinatorial optimization problems such as the ant colony optimization is worth exploring (Bonabeau et al., 1999). Furthermore, sequential circuits can be reduced to combinational circuits (Shaer et al., 2000a,b) 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

- Al-Arian, S., Bolling, R., 1994. Improving the testability of VLSI circuits through partitioning. *IEEE Symp. Circuits Syst.* 4, 199–202.
- Archambeau, E.C., McCluskey, E.J., 1984. Fault coverage of pseudo-exhaustive testing. *Digest, International Conference on Fault Tolerant Computation*, Orlando, FL, pp. 141–145.
- Baumgartner, U., Magele, Ch., Renhart, W., 2004. Pareto optimality and particle swarm optimization. *IEEE Trans. Magn.* 40 (2), 1172–1175.
- Bonabeau, E., Dorigo, M., Theraulaz, G., 1999. *Swarm Intelligence*. Oxford University Press, Oxford.
- Brglez, F., Fujiwara, H., 1985. A Neutral Netlist of 10 Combinational Benchmark circuits. *Proceedings of the IEEE International Symposium on Circuits and Systems*. IEEE Press, Piscataway, NJ (pp. 695–698; see also the ISCAS-85 benchmark directory at <http://www.cbl.ncsu.edu/benchmarks>).
- Clerc, M., Kennedy, J., 2002. The particle swarm—explosion, stability, and convergence in a multidimensional complex space. *IEEE Trans. Evol. Comput.* 6 (1), 58–73.
- Coello, C.A.C., Pulido, G.T., Lechuga, M.S., 2004. Handling multiple objectives with particle swarm optimization. *IEEE Trans. on Evol. Comput.* 8 (3), 256–279.
- Doctor, S., Venayagamoorthy, G.K., Gudise, V.G., 2004. Optimal PSO for collective robotic search applications. *IEEE Congress on Evolutionary Computation*, June 20–23, pp. 1390–1395.
- Gudise, V.G., Venayagamoorthy, G.K., 2004. FPGA placement and routing using particle swarm optimization. *IEEE Computer Society Annual Symposium on VLSI*, February 19–20, pp. 307–308.
- Hu Xiaohui, A., Eberhart, R., 2002. Multiobjective optimization using dynamic neighborhood particle swarm optimization. In: *Proceedings of the 2002 Congress on Evolutionary Computation*, vol. 2, May 12–17, pp. 1677–1681.
- Kennedy, J., Eberhart, R., 1995. Particle swarm optimization. In: *Proceedings IEEE International Conference on Neural Networks*, vol. IV, Perth, Australia, pp. 1942–1948. ISBN 1558605959.
- Kennedy, J., Eberhart, R., 2001. *Swarm Intelligence*. Academic Press, San Diego, USA. ISBN 0195131592.
- McCluskey, E.J., Bozorgui-Nesbat, S., 1981. Design for autonomous test. *IEEE Trans. Circuits Syst. CAS-28* (11), 1070–1079.
- Shaer, B., Dib, K., 2002. An Efficient Partitioning Algorithm of Combinational CMOS Circuits. In: *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, pp. 142–146.
- Shaer, B., Landis, D., Al-Arian, S., 2000a. Partitioning algorithm to enhance pseudoexhaustive testing of digital VLSI circuits. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* 8 (6), 750–754.
- Shaer, B., Al-Arian, S., Landis, D., 2000b. Partitioning sequential circuits for pseudoexhaustive testing. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* 8 (5), 534–541.
- Shi, Y., Eberhart, R.C., 1998. A modified particle swarm optimizer. In: *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 69–73.
- Shperling, I., McCluskey, E.J., 1987. Circuit segmentation for pseudo-exhaustive testing via simulated annealing. In: *Proceedings of the International Test Conference*, pp. 58–65.
- Venayagamoorthy, G.K., Gudise, V.G., 2004. 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.