Hindawi Publishing Corporation
VLSI Design
Volume 2008, Article ID 680157, 8 pages
doi:10.1155/2008/680157

Research Article

A Low-Cost BIST Scheme for Test Vector Embedding in Accumulator-Generated Sequences

Ioannis Voyiatzis

Department of Informatics, Technological Educational Institute of Athens, 12210 Athens, Greece

Correspondence should be addressed to Ioannis Voyiatzis, voyageri@otenet.gr

Received 27 March 2007; Accepted 5 December 2007

Recommended by Bashir M. Al-Hashimi

Test set embedding built-in self test (BIST) schemes are a class of pseudorandom BIST techniques where the test set is embedded into the sequence generated by the BIST pattern generator, and they displace common pseudorandom schemes in cases where reverse-order simulation cannot be applied. Single-seed embedding schemes embed the test set into a single sequence and demand extremely small hardware overhead since no additional control or memory to reconfigure the test pattern generator is required. The challenge in this class of schemes is to choose the best pattern generator among various candidate configurations. This, in turn, calls for a need to evaluate the location of each test pattern in the sequence as fast as possible, in order to try as many candidate configurations as possible for the test pattern generator. This problem is known as the test vector-embedding problem. In this paper we present a novel solution to the test vector-embedding problem for sequences generated by accumulators. The time overhead of the solution is of the order $\mathbf{O}(1)$. The applicability of the presented method for embedding test sets for the testing of real-world circuits is investigated through experimental results in some well-known benchmarks; comparisons with previously proposed schemes indicate that comparable test lengths are achieved, while the time required for the calculations is accelerated by more than 30 times.

1. INTRODUCTION

The problem of testing VLSI chips is becoming more and more time- and memory-consuming. For the testing of the chips fabricated today, complicated testing scenarios are applied, which incorporate both external testers and on-chip resources. The latter fall into the category of built-in self-test (BIST) techniques that provide for test pattern generation and response verification operations on chip [1].

BIST pattern generators apply the test vectors to the inputs of the circuit under test. The effectiveness of a BIST pattern generator is judged by the hardware overhead imposed on the circuit, the length of the applied test sequence, and the impact on the timing parameters of the circuit. In pseudorandom BIST schemes [2], either easily synthesizable modules (i.e., modules that can be easily implemented by altering existing registers, such as linear feedback shift registers or cellular automata [3]) or modules that already exist into VLSI chips (e.g., counters or accumulators [4]) are utilized for the generation of test patterns.

With pseudorandom BIST, both the hardware overhead and the impact on the circuit timing parameters are kept low. In order to reduce the length of the pseudorandom sequence, test vector embedding [5, 6] has been proposed. With test vector embedding, a precomputed (deterministic) test set is embedded into a sequence generated by a pseudorandom generator. In this way, the number of the applied pseudorandom patterns is decreased, without affecting the hardware or the impact on the timing parameters; furthermore, such schemes apply when reverse-order simulation [7] cannot be applied. In test vector embedding schemes, an embedding algorithm [5] is used. An embedding algorithm calculates the location (L) of a vector (V) in a sequence generated by the generator that starts from a specific value. Thus, L is the number of cycles that the generator module needs to operate until V appears at its outputs.

Embedding test vectors into sequences generated by hardware modules has been the goal of various researchers. For example, Lempel et al. [5] presented an algorithm for embedding test vectors into sequences generated by LFSRs,

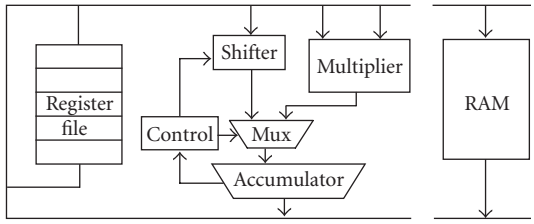

Figure 1: Typical DSP core.

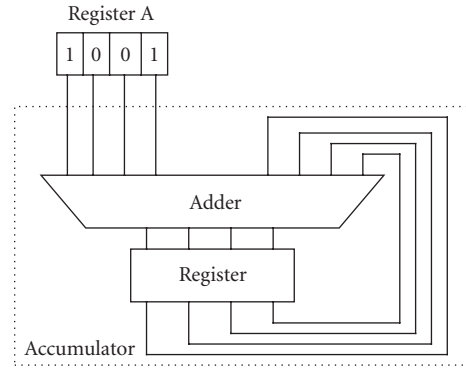

Figure 2: Accumulator fed by the pattern "1001" during BIST.

utilizing results of the theory of discrete logarithms, based on the results of [8, 9] in time proportional to $O(2^n)$, where n is the number of stages of the LFSRs. Kagaris and Tragoudas [10, 18] have presented results on embedding test vectors into counter-based sequences by using permutation and complementation operations on the counter outputs.

Current VLSI chips (implementing data paths or digital signal processors) commonly contain accumulators (see Figure 1); thereby, the utilization of such modules for the generation of test patterns or verification of responses of a circuit under test has no impact on the circuit timing parameters. For example, in [12] a response compaction scheme based on accumulators is presented for the testing of RAM modules. In [13], a scheme was presented that generates weighted patterns, that is, patterns where the probability of an output is different from 0.5 (purely pseudorandom vectors) based on a properly modified accumulator module. The pseudorandom nature of accumulator-generated sequences has been studied in [4]. Dorsch and Wunderlich [6] presented a test vector embedding approach utilizing accumulators and results of the theory on reduced-order binary decision diagrams [9]. Independently, Stroele and Meyer [7] explored methods to reduce test application time for accumulator-based self-test by skipping test patterns and utilizing reverse-order simulation. Recently, Manich et al. [14, 15] further advanced the field by presenting a scheme that minimizes memory requirements for storing the seeds and addends that feed the accumulator inputs. Their scheme is based on the observation that by using as addends the test patterns extracted from an automatic test pattern generator tool, the fault coverage is increased.

The above-mentioned schemes [6, 7, 14, 15] are based on the utilization of multiple seeds, where the accumulator is loaded with different values during the test phases and different addends feed the accumulator inputs. Therefore, they share the common need to store the accumulator addends and seeds, as well as additional control to handle the BIST operations. However, the requirement for additional memory and control for BIST purposes cannot always be met. In certain low-budget applications, the BIST hardware overhead needs to be as simple as possible. In these cases, single-seed solutions, where the test pattern generator is initialized and left to operate for a predetermined number of cycles until all faults under question are detected, may be a preferable solution. The cornerstone of such schemes is their test set embedding algorithm. However, the problem of embedding test patterns into hardware-generated sequences has been typically considered to be of exponential complexity (see, e.g., [5]).

In a recent work [11], a solution to the problem of embedding a test pattern into an accumulator-generated sequence was presented, which depends on the number of the stages of the accumulator; that is, it is of the order $\mathbf{O}(n)$. However, when a test set of T vectors has to be embedded, the complexity becomes $O(n \times T)$.

Nikolos et al. [16] exploited ideas of the number theory [17] in order to speed up the calculation of the locations of the test patterns of the test set. More precisely, they found a way to speed up the calculation from $\mathbf{O}(n \times T)$ to $\mathbf{O}(n + T)$; therefore, the time required to calculate the locations of all the patterns of the test set is reduced by a factor that ranges from 16 to 29 times compared to [11].

The hardware overhead of single-seed accumulator-based BIST schemes is extremely low, since the need for storage is eliminated; for example, the module presented in Figure 1 can be easily configured in such way that the inputs of the accumulator are driven by the outputs of a register of the register file. Hence, the hardware overhead is minimal, compared even to LFSR-based schemes, where the hardware overhead is n two-way multiplexers (n is the width of the LFSR). In Figure 2, the configuration of a 4-stage accumulator that accumulates the pattern 1001 is presented. The register A (one of the registers of the register file of Figure 1) is set to the 1001 value.

Another advantage of the schemes in [11, 16] is that no additional reordering of the inputs is required, as has been proposed by other schemes (see, e.g., [10, 18]); therefore, the data path does not have to be reconfigured during the BIST operations.

In this paper, we present a novel solution to the problem of embedding test patterns into accumulator-generated sequences; more precisely, we prove that the location of a pattern V in the sequence generated by an n-stage accumulator containing one's complement adder that accumulates the pattern $B = 2^b$, where b is an integer, can be calculated by a simple formula; hence the embedding algorithm is of the order $\mathbf{O}(1)$. To the best of our knowledge, this is the only result on embedding test vectors of the order $\mathbf{O}(1)$ presented in the literature. Comparisons with previous single-seed accumulator-based schemes [11, 16] indicate that

significant reduction is achieved in the calculation time to embed the test set, while the length of the resulting sequence is comparable.

The proposed scheme may be well incorporated into a generic scheme for the testing of processor cores, since it can be effectively utilized to test combinational parts of the core. For example, as shown in Section 3.3, the testing of a specific benchmark (c6288), which is a 16×16 array multiplier, can be performed in realistically low time.

The paper is organized as follows. In Section 2, we present the theory underlying the proposed scheme. In Section 3, comparisons with previously proposed schemes [11, 16] are performed. Finally, in Section 4, we conclude the paper.

2. THEORITICAL BACKGROUND

In the sequel, N will represent an integer number that is, a power of two, that is, $N = 2^n$.The symbol b denotes an integer number less than n, that is, $0 \le b < n$.^-B represents the bth-power of 2, that is, $B = 2^b$.

Definition 1. An $(N-1, B)$-sequence is the sequence of vectors $(0, B, (2\times B) \bmod (N-1), (3\times B) \bmod (N-1), \ldots, ((N-1)\times B) \bmod (N-1))$.

For example, the $(7, 2)$-sequence is $(0, 2, 4, 6, 1, 3, 5, 7)$. The $(15, 4)$-sequence is $(0, 4, 8, 12, 1, 5, 9, 13, 2, 6, 10, 14, 3, 7, 11, 15)$.

It has been proved (see, e.g., [4]) that an accumulator with one's complement adder starting from a nonzero value and accumulating a constant pattern B generates all nonzero vectors if and only if B is mutually prime with $N-1$. Therefore, an $(N-1, B)$-sequence as described in Definition 1 generates all $N-1n$-bit vectors since $N-1$ and 2^b are always prime. Indeed, the only numbers dividing 2^b are $2, 2^2, 2^3, \ldots, 2^{b-1}$. On the other hand, $N-1$ is an odd number since $N = 2^n$ is even.

From Definition 1, an accumulator containing a carry-rotate adder that accumulates a constant pattern B generates an $(N-1, B)$-sequence.

From the definition of the $(N-1, B)$-sequence, it is evident that for every value of n there exist exactly $n(N-1, B)$-sequences, one for each number $B = 2^b$, for $0 \le b < n$. For example, for $n = 4$, the four $(15, B)$-sequences are presented in Table 1.

Definition 2. The location of a vector V in an $(N-1, B)$-sequence, denoted by $L(N-1, B, V)$, is the position of the vector V in the $(N-1, B)$-sequence starting from 0.

From Definition 2, $L(N-1, B, V)$ indicates when V will be generated if the n-bit carry-rotate accumulator accumulates the constant value B. Therefore, if $L(N-1, B, V) = L$, then $B \times L \bmod (N-1) = V$. For example, from Table 1, it is easy to verify that $L(15, 4, 1) = 4$, while $L(15, 8, 6) = 12$.

Following Definitions 1 and 2, the problem of embedding a vector V in a sequence generated by an n-stage accumulator fed by a constant pattern B is transformed into calculating $L(N-1, B, V)$. The presented scheme is based on Theorem 1. In the sequel, "mod" will be used to indicate the remainder of the division of two integer numbers.

```
int TEC(int N, int B, int V)
 {return((V%B)*(N div B) + (V div B))};
```

Algorithm 1: C-language routine implementing the presented scheme.

Theorem 1. *If $N = 2^n$ and $B = 2^b, b < n$, then*

$$L(N-1, B, V) = (V \bmod B) \times (N/B) + (V \operatorname{div} B). \quad (1)$$

Proof. It is enough to prove that

$$(((V \bmod B) \times 2^{n-b} + (V \operatorname{div} B)) \times B) \bmod (N-1) = V. \quad (2)$$

Indeed, since

$$\begin{aligned}(V \bmod B) \times (N/B) \times B + (V \operatorname{div} B) \times B \\ = (V \bmod B) \times N + (V \operatorname{div} B) \times B,\end{aligned} \quad (3)$$

from the definition of the mod and div operators $V = (V \operatorname{div} B) \times B + (V \bmod B)$, and therefore $(V \operatorname{div} B) \times B = V - (V \bmod B)$; thus (3) gives

$$\begin{aligned}&(V \bmod B) \times N + (V \operatorname{div} B) \times B \\ &\quad = (V \bmod B) \times N + V - (V \bmod B) \\ &\quad = (V \bmod B) \times (N-1) + V.\end{aligned} \quad (4)$$

From this, we have

$$\begin{aligned}&(((V \bmod B) \times 2^{n-b} + (V \operatorname{div} B)) \times B) \bmod (N-1) \\ &\quad = ((V \bmod B) \times (N-1) + V) \bmod (N-1) = V.\end{aligned} \quad (5)$$

□

In Algorithm 1, the C function implementing the formula (1) is presented.

Example 1. Suppose we want to utilize the results of Theorem 1 in order to calculate the location of $V = 01001 = 9$ in the sequence generated by a 5-stage accumulator with one's complement adder accumulating the constant value $B = 01001 = 8$. Since $n = 5$, we have $N = 2^n = 2^5 = 32$ and $N - 1 = 31, N/B = 4$. Following (1), we can see that $L(N-1, B, V) = L(31, 8, 9)$ is calculated by

$$L = ((9\%8) \times (32/8) + (9 \operatorname{div} 8) = 1 \times 4 + 1 = 5. \quad (6)$$

It is easy to see that $(5 \times 8) \bmod 31 = 9$; therefore, $L = (31, 8, 9) = 5$. Thus, $V = 9$ is expected at the 5th position in the $(31, 8)$-sequence. From Table 5 of the appendix, we can see that, indeed, this is the case.

Example 2. For $n = 8$, $b = 5$ (i.e., $B = 2^5 = 32$), and $V = 194$, the $L(N-1, B, V) = L(255, 32, 194)$ is calculated as follows: $L = ((194\%32) \times (256/32) + (194 \operatorname{div} 32) = 2 \times 8 + 6 = 22$.

Indeed, it is easy to see that $(22 \times 32)\%255 = 194$.

Table 1: $(15, B)$-sequences.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$(15, 1)$-sequence = (	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15)
$(15, 2)$-sequence = (	0	2	4	6	8	10	12	14	1	3	5	7	9	11	13	15)
$(15, 4)$-sequence = (	0	4	8	12	1	5	9	13	2	6	10	14	3	7	11	15)
$(15, 8)$-sequence = (	0	8	1	9	2	10	3	11	4	12	5	13	6	14	7	15)

Table 2: Execution times of serial and linear searches versus the proposed $O(1)$ time algorithm.

n	Serial search $O(2^n)$	Linear search $O(n)$ [11]	Proposed $O(1)$
10	0.05"	0.02"	0.02"
12	0.44"	0.03"	0.03"
14	6.36"	0.13"	0.04"
16	1.5 min	0.42"	0.05"
18	25 min	1.39"	0.06"
20	6 h	6.71"	0.20"
22	12 d	26.39"	0.57"
24	243 d	2 min	2.06"
26	11 yrs	8 min	8.67"
28	235 yrs	33 min	24.56"

3. COMPARISONS

In this section, we will evaluate the proposed scheme in three directions. At first, we will compare the proposed scheme to serial- and linear-search algorithms with respect to the time required to calculate the locations of all 2^n patterns. This first comparison is indicative of the speed of the algorithm. Then, we will perform comparisons with the scheme proposed in [11] for randomly generated test sets. The purpose of this comparison is to investigate the effect of narrowing the search space from 2^n (in [11, 16]) to n (in the proposed scheme). Finally, we will compare the proposed scheme for test sets of real benchmarks, from the ISCAS'85 suite [19], in order to investigate the applicability of the proposed scheme in real-world circuits.

3.1. Comparisons with serial- and linear-search algorithms

We implemented the serial-search algorithm that examines the $2^n - 1$ test vectors until the target vector is found (this algorithm operates in $\mathbf{O}(2^n)$ time and is, therefore, representative of the exponential time algorithms) as well as the linear-search algorithm [11], that is representative of the $\mathbf{O}(n)$ time algorithms. The C-language routine we utilized for the implementation of the serial-search algorithm is shown in Algorithm 2. For the linear search, we utilized the algorithm given in [11].

We run C-programs in order to find the locations of all nonzero $N - 1 = 2^n - 1$ vectors for a single seed (B) of the accumulator (the computer used was Pentium III 933 MHz, with 256 MB of RAM) for various values of n. The execution times of the programs are presented in Table 2. For the calculation of the time required by the serial search for values of

```
int serial-search(N, B, V)
{int V, N, B, L;
   for (L = 1; L <= N − 1; L++)
      if (L∗B% (N − 1) == V) return(L); }
```

Algorithm 2: Serial-search algorithm.

$n \geq 20$, we simulated the time required to calculate the locations of a number of vectors 2^i, for $i < n$, and then projected these values to the total number of vectors..

From Table 2, it is evident that as the number of bits increases, the time required by the exponential as well as the linear-search algorithms may become impractical, whilst the time required by the presented method remains interestingly low.

3.2. Comparisons for randomly generated patterns

In order to validate the applicability of the presented algorithm and the quality of the applied test sequence, we performed simulations to embed sets of test patterns into accumulator-generated sequences. Our aim was to choose a "good" pair of numbers (B, S), where B is the constant value accumulated and S is the initial value of the accumulator such that the whole test set is generated in as few cycles as possible. We performed simulations utilizing random vectors generated by the random function of C for various values of the CUT inputs.

We experimented with all n candidate values of the input vector B. For every value of B, we kept $L_{\min}$ and $L_{\max}$, that is, the locations where the first and last vectors of the test set were generated. We also calculated the distance as $d = L_{\max} - L_{\min}$. Every time a value of B was found that generated the test set within fewer cycles, that is, $d < d_{\min}$, d was assigned to $d_{\min}$ and the new values of B and $d_{\min}$ were stored. For the calculation of the initial seed of the accumulator, denoted by S, it is trivial to see that if the accumulator is initialized to the starting value S calculated by $S = (B \times L_{\min}) \bmod N - 1$. and operates for $d_{\min}$ clock cycles, then the whole test set is generated. Therefore, in the sequel, S is not stated explicitly since it can be directly calculated from B and $L_{\min}$. Every time a value of B was found that generated the test set within fewer cycles, that is, $d < d_{\min}$, d was assigned to $d_{\min}$ and the new values of B and $d_{\min}$ were kept.

For each value of n (the CUT inputs), we performed experiments for four values of T (the number of test vectors in the test set), namely, 10, 20, 50, and 100. For each

Table 3: Test set embedding into randomly generated test sets (average of three experiments).

n	T	$d_{\min}$ [11]	$d_{\min}$ proposed	Difference (%)	$E[d]$
10	10	644	729	13,20%	745
	20	765	829	8,37%	878
	50	921	930	0,98%	964
	100	944	985	4,34%	994
12	10	2 973	2 923	−1,68%	2 979
	20	3 254	3 484	7,07%	3 511
	50	3 453	3 725	7,88%	3 855
	100	3 823	3 867	1,15%	3 974
14	10	11 428	9 456	−17,26%	11 916
	20	12 225	13 003	6,36%	14 043
	50	14 261	14 439	1,25%	15 420
	100	15 497	15 612	0,74%	15 897
16	10	44 673	41 792	−6,45%	47 663
	20	47 474	51 521	8,52%	56 174
	50	58 957	60 488	2,60%	61 681
	100	61 908	63 985	3,35%	63 589
18	10	184 884	162 783	−11,95%	190 650
	20	202 934	204 156	0,60%	224 695
	50	230 368	238 016	3,32%	246 724
	100	243 760	251 555	3,20%	254 358
20	10	737 360	601 989	−18,36%	762 601
	20	770 265	804 160	4,40%	898 779
	50	915 790	982 506	7,29%	986 895
	100	972 600	1 017 134	4,58%	1 017 430
22	10	3 011 089	2 317 584	−23,03%	3 050 403
	20	3 340 134	2 649 779	−20,67%	3 595 118
	50	3 787 780	3 953 723	4,38%	3 947 580
	100	3 965 825	3 988 537	0,57%	4 069 721
24	10	10 435 133	9 920 759	−4,93%	12 201 612
	20	12 462 717	14 217 794	14,08%	14 380 471
	50	14 882 312	15 513 764	4,24%	15 790 321
	100	15 475 737	16 194 611	4,65%	16 278 883
Average overhead = sum/32				**0,39**%	

value, we performed three experiments. The average value of these three experiments is presented in Table 3. In Table 3, the first column presents the number of the inputs of the CUT. The second column presents T, the number of (randomly generated) vectors of the test set. The third column presents the minimum distance $d_{\min}$ given by the scheme in [11]. The fourth column presents the minimum distance for the proposed scheme. The fifth column presents the % difference in $d_{\min}$ of the proposed scheme over the one in [11]. The value of $d_{\min}$ is, in general, expected to be higher than the one in [11], since in the proposed work, we have a smaller solution space (n instead of 2^{n-1} in [11]). In the sixth column, the expected mean value of d denoted by $E[d]$ is presented; the mean value is statistically equal to $E[d] = (T-2)/(T-1) \times 2^n$. The last cell of the table (rightmost cell of the last line) indicates the average increase of $d_{\min}$ of the proposed scheme over the scheme of [11]. This cell indicates an average increase of 0.39%, that is, negligible. Therefore, we can conclude that the quality of the test sequence of the proposed scheme (measured by the length of the sequence) is comparable to that of [11, 16]. Furthermore, by comparing the values of the fourth and sixth columns, we can see that the values of $d_{\min}$ given by the proposed scheme are smaller than the expected mean value of d, $E[d]$.

Next, we investigated the relationship of $d_{\min}$ with $d_{\max}$ (the maximum value of d for each experiment). We present the value of the ratio $d_{\min}/d_{\max}$ for various values of n and T in Figure 3. From Figure 3, it is extracted that—as we expected—the smaller the value of T, the better the performance of the embedding task, since this gives lower values for the quantity $d_{\min}/d_{\max}$. Furthermore, for small values of the number of patterns (i.e., no. 10 and no. 20), we can see a trend for decrease as the number of inputs of the test set increases.

Table 4: Simulation results for the ISCAS'85 circuits.

Crc	No. of Inp	No. of Vec	Test length			Diff %
			[11]	[16]	Proposed	
c499	41	51	1 642 559 798 913	1 633 341 388 679	1 841 099 032 209	≈2 97
C432	36	45	52 907 549 189	*	60 666 416 672	≈14 66%
c880	60	53	1 103 035 270 379 361 210	1 103 035 270 379 361 210	1 048 857 218 160 706 800	≈ −4 91%
c1355	41	86	1 881 067 490 587	1 875 129 180 656	2 047 414 822 299	≈5 26
c1908	33	116	7 517 790 261	7 517 790 261	6 442 450 947	≈ −14 99
c3540	50	145	981 672 975 771 693	953 512 009 096 806	1 083 104 061 625 293	≈9 89
c6288	32	28	2 146 503 245	2 146 503 245	3 871 357 360	≈54 46
Calculation time			>10 min	>20 s	<0.7 s	

* Not available in [16].

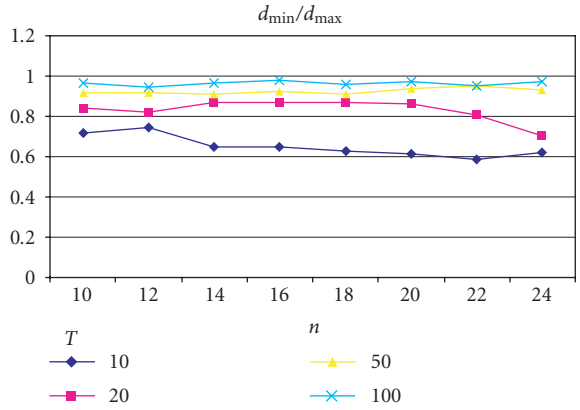

Figure 3: d_{min}/d_{max} for various values of n and T.

3.3. Comparisons for benchmark circuits

In order to illustrate the applicability of the presented scheme in real-world circuits, we applied our embedding algorithm to test sets extracted by COMPACTEST [20] for the ISCAS'85 circuits [19]; the fault coverage achieved by the utilized test patterns scales over 99% of the detectable single stuck-at faults. The BIST community generally considers the ISCAS'85 as good platforms for evaluating testing methodologies. Following the rationale of [10, 18], we have considered that the test set, once given, is not altered. This approach is mostly favorable when embedded modules such as intellectual property (IP) cores are utilized, whose inner structure is not available to the test designer; in such cases, the test designer utilizes test sets given by the designers of the modules and cannot exploit techniques such as reverse-order simulation (see, e.g., [7]).

The scheme in [11] introduced a linear-time algorithm to calculate the location of a test pattern in a sequence generated by an accumulator that accumulates a constant pattern. Nikolos et al. [16] proposed the use of the Diophantine equation in order to calculate the location of one of the test patterns of the test set. For the remaining patterns, they utilized a formula given in [17], eliminating the need to resolve the Diophantine equation. Therefore, they achieved a reduction of 16–29 times; that is, the time is reduced to 3.45% (in the best case) of the time reported in [11].

In Table 4, we present comparison data for the three schemes. In the first three columns, the circuit name, the number of its inputs, and the number of vectors extracted by COMPACTEST are presented. The fault coverage of these patterns (for single stuck-at faults) is over 99%. In the fourth column, the test length reported by [11, 16] is shown (the test lengths of [11, 16] are similar, having a difference from 0% to 2.8%, i.e., negligible) and in the fifth column, the test length of the proposed scheme is illustrated. In the sixth column, the % difference of the test length is presented (the ≈ symbol is used since the test lengths of [11, 16] are not exactly equal); in the last row of the table, the calculation times of the three schemes are presented. It is noted that the complexity of the scheme presented in [11] is $O(n \times T)$, where T is the number of patterns in the test set, the complexity of [16] is $O(n + T)$, and the complexity of the proposed scheme is $O(1 \times T)$.

The results reveal that for almost all benchmarks, the proposed scheme results in test sequence lengths that are comparable to those reported in [11, 16]; in one case (c1908), it even outperforms previous schemes. This result is somewhat "strange" since the space of solution of [11], which includes a much larger set of candidate values for the addend, should give much better results with respect to the value of d_{min}. However, this paradox may be deciphered as follows. In the experiments conducted in [11], since the times were prohibitively large, simulation was stopped after a predetermined time limit, for example, 20 minutes. Therefore, although better solutions might exist, they were not found. The scheme proposed in this work, due to the extremely low time required for the calculations, exhausts the solution space very quickly.

As for the calculation time, the proposed scheme always requires less than 0.7 seconds; that is, it is reduced by ≈850 times compared to the time of [11] (i.e., less than 0. 12%) and ≈30 times compared to the scheme in [16].

It should be further noted that the proposed scheme could be utilized in combination with the schemes proposed in [11, 16] as follows. At first, a very quick (in orders of milliseconds) test can be performed using the scheme proposed here in order to investigate if an acceptable test length can be achieved. If the test length given by the proposed scheme is acceptable, then the procedure stops; if a shorter test sequence is required, then the schemes in [11, 16] can be utilized and left to run for longer test time in order to try to

Table 5

(a) $(31, B)$-sequences: steps 0–15.

Steps	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$(31, 1)$-sequence	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$(31, 2)$-sequence	0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
$(31, 4)$-sequence	0	4	8	12	16	20	24	28	1	5	9	13	17	21	25	29
$(31, 8)$-sequence	0	8	16	24	1	9	17	25	2	10	18	26	3	11	19	27
$(31, 16)$-sequence	0	16	1	17	2	18	3	19	4	20	5	21	6	22	7	23

(b) $(31, B)$-sequences: steps 16–31.

Steps	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$(31, 1)$-sequence	0	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$(31, 2)$-sequence	0	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31
$(31, 4)$-sequence	0	6	10	14	18	22	26	30	3	7	11	15	19	23	27	31
$(31, 8)$-sequence	0	12	20	28	5	13	21	29	6	14	22	30	7	15	23	31
$(31, 8)$-sequence	0	24	9	25	10	26	11	27	12	28	13	29	14	30	15	31

find such a shorter sequence. Furthermore, if the achieved test length is not acceptable, then we may resort to alternative solutions like reseeding and multiple addends; such solutions are the subject of ongoing research.

4. CONCLUSIONS

A novel solution has been presented to the problem of embedding test vectors into sequences generated by accumulators containing one's complement adders. The presented solution calculates the location of a pattern into the sequence generated by a carry-rotate accumulator accumulating a constant pattern B of the form $B = 2^b$. The time complexity of the presented algorithm is of the order $\mathbf{O}(1)$. To the best of our knowledge, this is the first solution to the problem of embedding patterns into hardware-generated sequences of the order $\mathbf{O}(1)$ presented in the literature. Comparisons with previous schemes based on exponentiall $\mathbf{O}(2^n)$ and linear $\mathbf{O}(n)$ time algorithms reveal that the proposed scheme results in comparable test lengths, in noticeably lower time. For the examined ISCAS85 benchmark circuits, the reduction in time to embed the test set is 30 times lower than [16] and 850 times lower than [11].

APPENDIX

SEE TABLE 5 (31,B)-SEQUENCES

REFERENCES

[1] M. Abramovici, M. Breuer, and A. Freidman, *Digital Systems Testing and Testable Design*, Computer Science Press, Rockville, Md, USA, 1990.

[2] P. Bardell, W. McAnney, and J. Savir, *Built-In Test for VLSI: Pseudorandom Techniques*, John Wiley & Sons, New York, NY, USA, 1987.

[3] C. Dufaza and G. Gambon, "LFSR-based deterministic and pseudorandom test pattern generator structures," in *Proceedings of the European Test Conference*, pp. 27–34, Nashville, Tenn, USA, October 1991.

[4] A. P. Stroele, "BIST patter generators using addition and subtraction operations," *Journal of Electronic Testing*, vol. 11, no. 1, pp. 69–80, 1997.

[5] M. Lempel, S. Gupta, and A. Breuer, "Test embedding with discrete logarithms," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 5, pp. 554–566, 1995.

[6] R. Dorsch and H.-J. Wunderlich, "Accumulator based deterministic BIST," in *Proceedings of IEEE International Test Conference (TC '98)*, pp. 412–421, Washington, DC, USA, October 1998.

[7] A. P. Stroele and F. Mayer, "Methods to reduce test application time for accumulator-based self-test," in *Proceedings of the 15th IEEE VLSI Test Symposium*, pp. 48–53, Monterey, Calif, USA, April 1997.

[8] S. Pohlig and M. Hellman, "An improved algorithm for computing logarithms over GF(p) and its cryptographic significance," *IEEE Transactions and Information Theory*, vol. 24, no. 1, pp. 106–110, 1978.

[9] S. Akers, "Binary decision diagrams," *IEEE Transactions on Computers*, vol. 27, no. 6, pp. 509–516, 1978.

[10] D. Kagaris and S. Tragoudas, "On the design of optimal counter-based schemes for test set embedding," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 2, pp. 219–230, 1999.

[11] I. Voyiatzis, "Test vector embedding into accumulator-generated sequences: a linear-time solution," *IEEE Transactions on Computers*, vol. 54, no. 4, pp. 476–484, 2005.

[12] I. Voyiatzis, "Accumulator-based compression in symmetric transparent RAM BIST," in *Proceedings of International Conference on Design and Test of Integrated Systems in Nanoscale Technology (DTIS '06)*, pp. 273–278, Tunis, Tunisia, September 2006.

[13] I. Voyiatzis, D. Gizopoulos, and A. Paschalis, "Accumulator-based weighted pattern generation," in *Proceedings of the 11th IEEE International On-Line Testing Symposium (IOLTS '05)*, pp. 215–220, St. Rafael, France, July 2005.

[14] S. Manich, L. Garcia, L. Balado, et al., "On the selection of efficient arithmetic additive test pattern generators," in *Proceedings of the 8th IEEE European Test Workshop (ETW '03)*, pp.

9–14, Maastricht, The Netherlands, May 2003.

[15] S. Manich, L. Garcia-Deiros, and J. Figueras, "Minimizing test time in arithmetic test-pattern generators with constrained memory resources," *IEEE Transactions in Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 11, pp. 2046–2058, 2007.

[16] D. Nikolos, D. Kagaris, and S. Gidaros, "Diophantine-equation based arithmetic test set embedding," in *Proceedings of the 12th IEEE International On-Line Testing Symposium (IOLTS '06)*, pp. 193–194, Como, Italy, July 2006.

[17] J. Hunter, *Number Theory*, Oliver and Boyd, London, UK, 1964.

[18] D. Kagaris, S. Tragoudas, and A. Majumdar, "On the use of counters for reproducing deterministic test sets," *IEEE Transactions on Computers*, vol. 45, no. 12, pp. 1405–1419, 1996.

[19] F. Brglez and H. Fujiwara, "A neutral netlist of 10 combinational benchmarks circuits and a target translator in FORTRAN," in *Proceedings of International Symposium on Circuits and Systems*, Kyoto, Japan, June 1985.

[20] I. Pomeranz, L. N. Reddy, and S. M. Reddy, "COMPACTEST: a method to generate compact test sets for combinational circuits," in *Proceedings of IEEE International Test Conference*, pp. 194–203, Nashville, Tenn, USA, October 1991.