



# An effective two-pattern test generator for Arithmetic BIST<sup>☆</sup>

I. Voyiatzis<sup>a,\*</sup>, C. Efstathiou<sup>a</sup>, H. Antonopoulou<sup>b</sup>, A. Milidonis<sup>a,c</sup>

<sup>a</sup> Department of Informatics, Technological Educational Institute of Athens, Ag. Spyridonos & Milou, Egaleo 12210, Greece

<sup>b</sup> Technological Educational Institute of Patras, M. Alexandrou 1, Koukouli, 263 34 Patras, Greece

<sup>c</sup> Nokia Siemens Network, Greece

## ARTICLE INFO

### Article history:

Received 29 June 2010

Received in revised form 18 October 2012

Accepted 18 October 2012

Available online 20 November 2012

## ABSTRACT

Built-In Self Test (BIST) techniques perform test pattern generation and response verification operations on-chip. In Arithmetic BIST, modules that commonly exist in datapaths (accumulators, counters, etc.) are utilized to perform the above-mentioned operations. In order to detect faults that occur into current CMOS circuits, two-pattern tests are required. Furthermore, delay testing, commonly used to assure correct temporal circuit operation at clock speed requires two-pattern tests. In this paper a novel two-pattern test generator for Arithmetic BIST is presented. Its hardware implementation compares favorably to the techniques that have been presented in the literature. Application of the proposed scheme for the two-pattern testing of ROM modules revealed that the testing of small-to-medium size ROMs is completed within reasonable time and with negligible hardware overhead.

© 2012 Elsevier Ltd. All rights reserved.

## 1. Introduction

In current integrated circuit technology, where highly complex chips have low accessibility of internal nodes, traditional testing becomes costly and ineffective. Built-In Self Test (BIST) techniques have been utilized as a powerful alternative to external testing. BIST techniques employ on-chip test generation and response verification, reducing the need for expensive external testing equipment. Furthermore, using BIST at-speed testing can be achieved which, in turn, increases the quality of the delivered integrated circuits [1–4].

Pattern generators utilized in BIST techniques are commonly discerned into *one-pattern* and *two-pattern*. One-pattern generators target the detection of combinational (i.e. stuck-at) faults. One-pattern generators can be designed using Linear Feedback Shift Registers (LFSRs) [5], Cellular Automata (CA) [6], or Accumulators [7].

However, it has been proved that many failure mechanisms that appear in CMOS circuits can alter the behavior of a combinational circuit under test into a sequential one [8]. Furthermore, increasing performance requirements emphasize the need to operate digital circuits at their highest possible speeds. This motivates testing for the correct temporal behavior, commonly known as delay testing [1,2]. The detection of these types of faults requires two-pattern tests. Therefore, BIST two-pattern generators have been proposed during the last years [9–26].

Arithmetic two-pattern generators exploit arithmetic modules (e.g. datapaths, counters, etc.) that exist in modern general-purpose microprocessors or digital signal processors in order to generate test patterns for one or more modules in the circuit. The utilization of existing modules for BIST purposes is favored by low hardware overhead, low impact on the circuit normal operating speed and the fact that the modules used for BIST purposes are exercised. Therefore, faults existing in the arithmetic modules can be detected, and the need for further testing of the modules is eliminated. Two-pattern generation based on accumulators has been proposed in [24,25].

<sup>☆</sup> Reviews processed and approved for publication by Editor-in-Chief Dr. Manu Malek.

\* Corresponding author. Tel.: +30 2105385394.

E-mail address: [voyageri@teiath.gr](mailto:voyageri@teiath.gr) (I. Voyiatzis).

In this paper an effective Arithmetic two-Pattern Test BIST generator (hereafter referred to as APT) is presented. The proposed generator can generate all two-pattern tests and is based on an accumulating structure whose inputs are driven by a binary counter or a Linear Feedback Shift Register (LFSR). The accumulating structure is composed of a controlled binary/1's-complement adder and a register. The presented scheme compares favorably to techniques that have been proposed in the open literature. Application of the proposed scheme for the two-pattern testing of ROM modules reveals that the testing of small-to-medium size ROMs is completed within acceptable time limits with negligible hardware overhead.

The paper is organized as follows. In Section 2 we present the proposed generator based on an accumulating structure whose inputs are driven by the outputs of a counter. In Section 3 an implementation of the proposed scheme based on a parallel-prefix adder with fast carry input processing, as well as the implementation of the scheme when the inputs of the accumulating structure are fed by an LFSR are given. In Section 4 the proposed two-pattern generator is compared with the techniques that have been proposed to date. In Section 5 we study the application of the proposed scheme for two-pattern testing of ROM modules; we conclude the work in Section 6.

## 2. Proposed algorithm

In the following, let  $n$  be the number of bits of the vectors to be generated. We will use  $N$  to represent  $2^n$  ( $N = 2^n$ ).

**Definition 1.** A Sequence( $V, N$ ) is a sequence of  $N$  vectors  $V_0, V_1, \dots, V_{N-1}$ , where

$$V_0 = V,$$

$$V_1 = ((V_0 - 1 + 1) \bmod (N - 1)) + 1$$

$$\vdots$$

$$V_k = ((V_{k-1} - 1 + k) \bmod (N - 1)) + 1, \quad \text{for } k = 2, 3, \dots, N - 2 \text{ and}$$

$$V_{N-1} = (V_{N-2} + N - 1) \bmod N.$$

**Example 1.** The Sequence(6,8) is presented in Table 1. In this table, the pattern 6 is the first pattern of the sequence; in the 6 cycles to follow, the patterns 1, 2, 3, 4, 5, 6 (first column) are added modulo 7, therefore the sequence 7, 2, 5, 2, 7, 6 is generated; during the last (seventh) cycle, 7 is added modulo 8 giving 5 as result.

The proposed APT algorithm is presented in Fig. 1 in pseudocode. The algorithm Sequence( $V, N$ ) generates a Sequence( $V, N$ ) as given by Definition 1. APT algorithm operates in two phases, namely Phases 1 and 2; during Phase 1,  $N$  Sequences are generated; Phase 1 is completed when the zero pattern is generated. During Phase 2 all transitions to and from the all zero pattern are generated. It should be noted that during Phase 1 the zero pattern is generated at the last step and it is not generated during any previous cycle. This is proved formally in Lemma 1.

The APT( $n$ ) algorithm in a more detailed C-like notation is given in Fig. 2. During Phase 1 the consecutive vectors  $V_k$ , ( $k = 1, 2, \dots, N-2$ ) of Sequence( $N-1, N$ ) are generated. Then, a vector  $V$  (i.e.  $N-2$ ) is generated and a Sequence( $N-2, N$ ) commences. This is repeated for  $V = N-2, N-3, \dots, 1$ . At the end of the generation of all Sequence( $V, N$ ) the all-zero vector is reached as the last pattern of a Sequence( $1, N$ ). In Phase 2 of APT( $n$ ) all transitions to and from the all-zero vector are generated. This is achieved by setting  $V$  to the all-zero vector every second step and adding to it the values of  $k$  in the next step.

The vectors produced by the APT algorithm for  $n = 3$  are given in column ACC of Tables 2 (for Phase 1) and 3 (for Phase 2).

The APT( $n$ ) algorithm generates all  $n$ -bit pairs of patterns. This claim is proved in Theorem 1. The following Lemmas are used in the proof of Theorem 1; from Tables 2 and 3, it is trivial to confirm that Lemmas 1–3 hold for the case  $n = 3$ .

**Lemma 1.** The all-zero vector is not included in the vectors of any Sequence( $V, N$ ) for  $V > 1$ ; it is only generated at the end of the Sequence( $1, N$ ).

**Table 1**  
Sequence(6,8).

$K$	Sequence(6,8)	Comment
	6	Addition modulo 7
1	7	
2	2	
3	5	
4	2	
5	7	
6	6	Addition modulo 8
7	5	

```

algorithm APT
Phase1:
  V = N-1;
do
  V = Sequence(V,N);
until V=0;

Phase2:
  for i=1 to N-1
    generate pair (0,i) and pair (i,0);

Sequence(V,N)
  next = V;
  for k = 1 to N-2
    {if next + k < N then next = n+k else next = (next + k) mod (N-1);}
  next = (next + N-1) mod N;
  return (next);

```

Fig. 1. APT algorithm.

Phase	Step	
1	1	int main()
		{ int k=1, k_prev, ACC=N-1;
		printf("%d %d\n",k,ACC);
		do {
	2	do {
	3	ACC = ACC+k<N? ACC+k: ACC+k-N+1;
2	4	k = inc(k);
	5	printf("%d %d\n",k,ACC);
	6	} while (k<N-1);
	7	ACC = ACC+k<N? ACC + k: ACC+k-N;
	8	k= inc(k);
	9	printf("%d %d\n",k,ACC);
	10	} while (!(ACC==0));
	11	do{
	12	ACC=ACC+k; k=inc(k);
	13	printf("%d %d\n",k,ACC);
	14	k_prev=k;
	15	ACC=0; k=inc(k);
	16	printf("%d %d\n",k,ACC);
	17	} while (!(k_prev==N-1) && (ACC==0));
	18	}
		int inc(int k) {return(k<N-1 ? ++k : 1); }

Fig. 2. The APT(n) algorithm in C-like notation.

**Proof.** According to the definition of Sequence(V,N) the all-zero vector is not derived during the generation of vectors  $V_k$  for  $k = 1, 2, 3, \dots, N-2$ . For any Sequence(V,N),  $V_{N-2} = (V + 1 + 2 + \dots + N-2) \bmod (N-1) = [V + (N-2) \times (N-1)/2] \bmod (N-1) = V$  (since  $(N-2)$  is divisible by 2). Therefore,  $V_{N-1} = (V + N-1) \bmod N = V-1$  which is equal to zero only when  $V = 1$ .  $\square$

**Lemma 2.** In any Sequence(V,N), no identical pairs of vectors exist.

**Proof.** If two identical pairs were generated, the respective difference would be the same. However, this is not possible according to the definition of Sequence(V,N).  $\square$

**Lemma 3.** In the Sequence(V,N), derived for  $V = N-1, \dots, 1$ , no identical vector pairs  $(V_k, V_{k+1})$ , for  $k = 1, 2, 3, \dots, N-2$  are generated.

**Proof.** Since the vectors are generated in the same order, if the same vector pair was generated, it would be generated at the same position. This means that the two Sequences should have started from the same number, which contradicts to the assumption.  $\square$

Lemma 3 implies that in the Sequence(V,N), derived for  $V = 1, 2, \dots, N-1$ , a vector pair  $(V_k, V_{k+1})$ , for  $k = 1, 2, \dots, N-2$  is not generated twice. However, it should be noted that the vector pair  $(V_{N-2}, V_{N-1})$  of the Sequence(m,N) is equal to the vector pair  $(V_{N-1}, V_N)$  of Sequence(m+1,N). For example, the vector pair (7,6) is derived both during the generation of the Sequence(7,8) and the generation of the Sequence(6,8).

**Table 2**

Sequences of APT(3). Bold fonts indicate the actual sequence applied to the inputs of the CUT.

Cycle #	$k$	ACC	
	1	<b>7</b>	Sequence(7,8)
1	2	<b>1</b>	
2	3	<b>3</b>	
3	4	<b>6</b>	
4	5	<b>3</b>	
5	6	<b>1</b>	
6	7	<b>7</b>	
7	1	<b>6</b>	Sequence(6,8)
8	2	<b>7</b>	
9	3	<b>2</b>	
10	4	<b>5</b>	
11	5	<b>2</b>	
12	6	<b>7</b>	
13	7	<b>6</b>	
14	1	<b>5</b>	Sequence(5,8)
15	2	<b>6</b>	
16	3	<b>1</b>	
17	4	<b>4</b>	
18	5	<b>1</b>	
19	6	<b>6</b>	
20	7	<b>5</b>	
21	1	<b>4</b>	Sequence(4,8)
22	2	<b>5</b>	
23	3	<b>7</b>	
24	4	<b>3</b>	
25	5	<b>7</b>	
26	6	<b>5</b>	
27	7	<b>4</b>	
28	1	<b>3</b>	Sequence(3,8)
29	2	<b>4</b>	
30	3	<b>6</b>	
31	4	<b>2</b>	
32	5	<b>6</b>	
33	6	<b>4</b>	
34	7	<b>3</b>	
35	1	<b>2</b>	Sequence(2,8)
36	2	<b>3</b>	
37	3	<b>5</b>	
38	4	<b>1</b>	
39	5	<b>5</b>	
40	6	<b>3</b>	
41	7	<b>2</b>	
42	1	<b>1</b>	Sequence(1,8)
43	2	<b>2</b>	
44	3	<b>4</b>	
45	4	<b>7</b>	
46	5	<b>4</b>	
47	6	<b>2</b>	
48	7	<b>1</b>	
49	1	<b>0</b>	

**Theorem 1.** The APT( $n$ ) algorithm generates all  $n$ -bit two-pattern pairs.

**Proof.** The APT( $n$ ) algorithm comprises two phases. During Phase 1,  $(N-1)$  Sequence( $V, N$ ) are generated, resulting into a total of  $(N-2) \times (N-1) = N \times (N-1) - 2 \times (N-1)$  distinct pairs. On the other hand, the number of pairs excluding the transitions to and from zero are  $N \times (N-1) - 2 \times (N-1)$ . The two numbers are equal. Therefore, it must be shown that the  $(N-1) \times (N-2)$  pairs generated during the application of the first  $(N-2)$  steps of each sequence are distinct. This is proved as follows.

The all-zero vector is not included in the Sequences, except at the end of the Sequence( $1, N$ ) (Lemma 1). In any Sequence, no two identical vector pairs are generated (Lemma 2). During the generation of the different Sequence( $V, N$ ) no two identical vector pairs exist (Lemma 3). Therefore, the derived  $(N-1) \times (N-2)$  vector pairs are distinct.

In Phase 2 the APT( $n$ ) algorithm generates all transitions to and from the all-zero vector. This is proved as follows. Since in every second step  $V$  is set to the all-zero value and  $k$  is added to it in the next step, all transitions from zero to the vectors  $V$  for  $V = 2k + 1$  and vice versa are generated in the first  $2n + 1$  clock cycles. At the  $(2n + 1)$ -th cycle, the value of  $k$  is set to 1. In the following  $2n$  steps, all transitions from the all-zero vector to  $V$ , for  $V = 2 \times k$  and vice versa are generated.  $\square$

**Table 3**

Patterns generated during Phase 2 of the APT(3) algorithm. Bold fonts used to emphasize the zero values generated during phase 2 of the APT(3) algorithm.

Cycle #	$k$	ACC
49	1	0
50	2	1
51	3	<b>0</b>
52	4	3
53	5	<b>0</b>
54	6	5
55	7	<b>0</b>
56	1	7
57	2	<b>0</b>
58	3	2
59	4	<b>0</b>
60	5	4
61	6	<b>0</b>
62	7	6
63	1	<b>0</b>

### 3. Hardware implementation

The block diagram of the hardware implementation of the proposed scheme is given in Fig. 3. It is based on an accumulating structure and a counter. The accumulating structure is composed of a controlled binary/1's complement adder and a register.

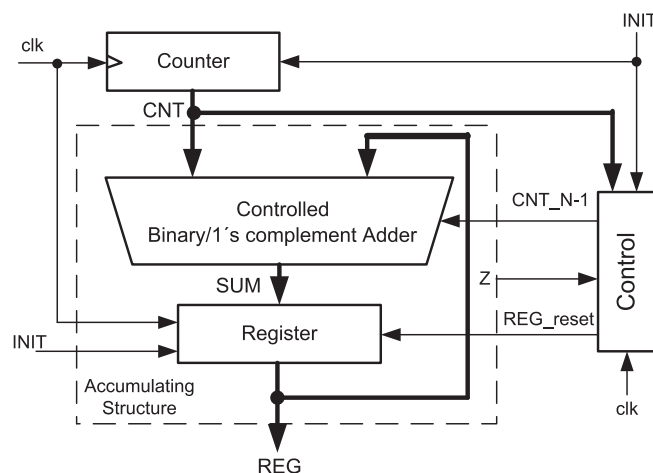
The APT( $n$ ) algorithm is implemented by the structure shown in Fig. 3 as follows. In Phase 1 of the algorithm the signal INIT is activated setting the Register to value  $N-1$  (all ones), and the counter to state 1 (CNT = 1). Then, the accumulating structure performs  $N-2$  additions modulo ( $N-1$ ) (or, equivalently, 1's complement additions) of the consecutive outputs of the counter to the initial value ( $N-1$ ) of the register. Therefore, vectors  $V_1, V_2, \dots, V_{N-2}$  of the APT( $n$ ) algorithm are generated at the REG output of the proposed generator. When the counter reaches the value  $N-1$ , the controlled adder is dictated by the activation of the signal CNT\_N-1 of the control module to perform a binary (modulo  $N$ ) addition to generate vector  $V_{N-1}$ .

When all Sequence( $V, N$ ) for  $V = N-1, N-2, \dots, 1$  have been generated, at the end of the Sequence(1,  $N$ ), the output of the adder becomes zero,  $Z = 1$ , ( $Z$  is the Zero signal of the datapath), and CNT =  $N-1$ . This condition initiates Phase 2. In this phase the register is dictated by the REG\_reset signal of the control module to become zero every second clock cycle.

The operation of the proposed architecture is further exemplified for  $n = 3$  in Table 4. In Table 4, Phase 1 spans cycles 1–48, while Phase 2 spans the remaining cycles. The column CNT contains the outputs of the counter, which repeats through the sequence 1, 2,  $\dots$ , 7, 1, 2,  $\dots$ , 7. The column SUM contains the sum of the respective values of ACC and CNT. It is trivial to note that columns CNT and REG in Table 4 are identical to the columns  $k$  and ACC in Tables 2 and 3.

#### 3.1. Design of the controlled binary/1's complement adder

For the implementation of the accumulating structure, we should note that a trivial way to implement the 1's complement addition is to connect the carry output of an  $n$ -bit adder to its carry input. A problem arising from this end-around carry



**Fig. 3.** Architecture of the proposed two pattern generator.

**Table 4**Patterns generated by the generator of Fig. 3 for  $n = 3$ . Bold fonts indicate the actual sequence applied to the inputs of the CUT.

Cycle #	CNT	SUM	REG
	1	1	7
1	2	3	<b>1</b>
2	3	6	<b>3</b>
3	4	3	<b>6</b>
4	5	1	<b>3</b>
5	6	7	<b>1</b>
6	7	6	<b>7</b>
7	1	7	<b>6</b>
8	2	2	<b>7</b>
9	3	5	<b>2</b>
10	4	2	<b>5</b>
11	5	7	<b>2</b>
12	6	6	<b>7</b>
13	7	5	<b>6</b>
14	1	6	<b>5</b>
15	2	1	<b>6</b>
16	3	4	<b>1</b>
17	4	1	<b>4</b>
18	5	6	<b>1</b>
19	6	5	<b>6</b>
20	7	4	<b>5</b>
21	1	5	<b>4</b>
22	2	7	<b>5</b>
23	3	3	<b>7</b>
24	4	7	<b>3</b>
25	5	5	<b>7</b>
26	6	4	<b>5</b>
27	7	3	<b>4</b>
28	1	4	<b>3</b>
29	2	6	<b>4</b>
30	3	2	<b>6</b>
31	4	6	<b>2</b>
32	5	4	<b>6</b>
33	6	3	<b>4</b>
34	7	2	<b>3</b>
35	1	3	<b>2</b>
36	2	5	<b>3</b>
37	3	1	<b>5</b>
38	4	5	<b>1</b>
39	5	3	<b>5</b>
40	6	2	<b>3</b>
41	7	1	<b>2</b>
42	1	2	<b>1</b>
43	2	4	<b>2</b>
44	3	7	<b>4</b>
45	4	4	<b>7</b>
46	5	2	<b>4</b>
47	6	1	<b>2</b>
48	7	0	<b>1</b>
49	1	1	<b>0</b>
50	2	3	<b>1</b>
51	3	3	<b>0</b>
52	4	7	<b>3</b>
53	5	5	<b>0</b>
54	6	3	<b>5</b>
55	7	7	<b>0</b>
56	1	0	<b>7</b>
57	2	2	<b>0</b>
58	3	5	<b>2</b>
59	4	4	<b>0</b>
60	5	1	<b>4</b>
61	6	6	<b>0</b>
62	7	5	<b>6</b>
63	1	1	<b>0</b>

connection is that under certain circumstances the output of the adder may oscillate [27]. Efficient oscillation-free 1's complement adders can be derived from parallel-prefix adders using fast input carry processing [28,29]. Since we use this adder architecture for the implementation of controlled binary/1's complement adder, in the following we briefly review its design.

Let  $A = a_{n-1}a_{n-2} \dots a_0$  and  $B = b_{n-1}b_{n-2} \dots b_0$  be the two  $n$ -bit operands to be added,  $c_{-1}$  the carry input and  $S = s_{n-1}s_{n-2} \dots s_0$  be their sum. A parallel-prefix adder with fast input carry processing can be considered as a four-stage circuit. The preprocessing stage computes the carry-generate/propagate bits  $g_i = a_i \cdot b_i$ ,  $p_i = a_i \oplus b_i$ , for  $0 \leq i \leq n-1$ , ( $\oplus$  denotes the exclusive-OR operation). The second stage computes the group generate/propagate terms  $(G_i, P_i)$ , for  $0 \leq i \leq n-1$ . The  $G_i, P_i$  terms are formally defined as  $(G_i, P_i) = (g_i, p_i) \circ (g_{i-1}, p_{i-1}) \circ \dots \circ (g_1, p_1) \circ (g_0, p_0)$ , where  $\circ$  is the prefix operator defined as  $(g_m, p_m) \circ (g_k, p_k) = (g_m + p_m \cdot g_k, p_m \cdot p_k)$ . The third stage performs the computation of the carries according to the relation  $c_i = G_i + P_i \cdot c_{i-1}$ . The fourth stage computes the sum bits according to the relation  $s_i = p_i \oplus c_{i-1}$ .

The controlled binary/1's complement adder utilized by the proposed scheme is given in Fig. 4, where the square ( $\square$ ) nodes implement the functions  $g_i, p_i$ , while the diamond ( $\diamond$ ) nodes implement function  $s_i$ . The gray cycle nodes implement the functions  $c_i = G_i + P_i \cdot c_{i-1}$ . The adder performs either 1's complement addition (when  $\text{CNT\_N}-1 = 0$ ) or binary addition with no carry (when  $\text{CNT\_N}-1 = 1$ ).

### 3.2. Design of the control module

The design of the control module is presented in Fig. 5. Initially the two flip-flops are set to zero. When the output of the adder is zero ( $Z = 1$ ) and  $\text{CNT\_N}-1$  is enabled, the control module proceeds in a state where the  $\text{REG\_reset}$  signal is driven by the divided-by-two clock signal  $\text{clk}/2$ . The Zero signal ( $Z$ ) is enabled when the output of the adder is equal to the all-zero vector and is available in the datapath of all processors as it is the signal that drives the zero flag.

### 3.3. LFSR-driven implementation

The proposed BIST scheme can be implemented by driving the inputs of the accumulating structure with the outputs of a maximal cycle Linear Feedback Shift Register (LFSR) instead of a counter. LFSRs are commonly utilized circuits and can be easily implemented by modifying existing registers.

An  $n$ -stage LFSR consists of  $n$  memory elements (flip-flops) called stages, connected via using a small number of XOR gates. An LFSR that can generate all  $2^n - 1$   $n$ -bit non-zero patterns (if it is initialized to a non-zero value) is called a maximal cycle LFSR. A maximal cycle 3-stage LFSR is presented in Fig. 6. An extensive discussion on the design of LFSR structures can be found in [1].

**Definition 2.** We define as an  $L\text{Sequence}(V, N)$ , the sequence of vectors starting from  $V$  and performing  $N-2$  consecutive additions of  $k$  modulo  $N-1$ , for discrete values of  $k$  for  $k = 1, 2, 3, \dots, N-2$  (irrespective of the order of the values of  $k$ ) and a final addition of  $N-1$  modulo  $N$ .

It is trivial to show that Theorem 1 holds in case the inputs of the accumulator are driven by the outputs of an LFSR instead of a counter; indeed an  $L\text{sequence}$  and a  $\text{Sequence}$  have the same output effect, since it is easy to generate the LFSR output sequence in such way that the  $N-1$  value is generated as the last value of the sequence. For example, for the sequence generated by the LFSR of Fig. 6  $\{011, 101, 010, 001, 100, 110, 111\} = \{3, 5, 2, 1, 4, 6, 7\}$  the  $L\text{sequence}(k, N)$  for the various values of  $k$  are presented in Table 5.

## 4. Comparisons

In this section we compare the proposed scheme with the ones that have been proposed in the literature and can generate all two-pattern tests. Such techniques have been proposed by Starke [9], Vuksic and Fuchs [10] and Chen and Gupta [11].

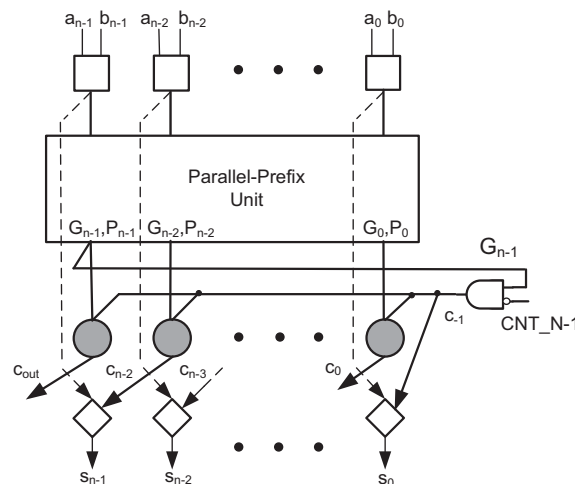


Fig. 4. Proposed controlled parallel-prefix adder.





$$HO_{[11]-LFSR}(n) = n \times DFF + n \times MUX_{21}$$

The CA-version of the technique requires  $n$  flip-flops (for the formation of the  $2n$ -stage CA) and a number of XOR gates for the formation of the CA rules. In order to calculate the number of XOR gates, we assume that half of the stages implement rule 90, while the others implement rule 150. This assumption is justified since these two rules are the most commonly used in Cellular Automata applications [6]. Rule 90 requires one 2-input XOR gate, while Rule 150 requires two 2-input XOR gates. Therefore, the hardware overhead of the technique is:

$$HO_{[11]-CA}(n) = n \times DFF + 1.5 \times n \times XOR + n \times MUX_{21}$$

In the hardware overhead estimations that follow, the existence of an  $n$ -stage accumulator and an  $n$ -stage counter or LFSR is assumed. The technique presented in [24], utilizes a carry-free accumulator whose inputs are driven by either a counter or an LFSR. In order to implement this technique a control module (composed of six 2-input gates, 2 flip-flops and two  $n$ -input logic gates) is required. Therefore, the hardware overhead is given by the following formula:

$$HO_{[24]}(n) = 2 \times GT(n) + 2 \times DFF + 6 \times GT(2) + n \times MUX_{21}$$

The implementation of the technique proposed in [25] for exhaustive two-pattern generation requires a detect module (two  $n$ -input gates) and a control module (composed of 3 flip-flops and 8 two-input gates). Therefore, the hardware overhead is given by:

$$HO_{[25]}(n) = 2 \times GT(n) + 3 \times DFF + 18 \times GT(2) + n \times MUX_{21}$$

In order to implement the proposed generator a control module (Fig. 5), composed of one logic gate with  $n$  inputs, two 2-inputs logic gates and two flip-flops are required. Therefore, the hardware overhead is given by:

$$HO_{APT}(n) = 2 \times DFF + GT(n) + 2 \times GT(2) + n \times MUX_{21}$$

In Table 6, comparison data are given. For each one of the techniques, the module that generates the two-pattern test is presented in the second column. The existing module(s) utilized by the technique are presented in the third column. In the fourth column the hardware overhead (in gate equivalents) is presented.

For our calculations the following have been taken into consideration. An  $n$ -input gate, denoted here as  $GT(n)$ , is equivalent to  $n-1$  gates; a 2-input XOR gate or a 2-to-1 MUX is equivalent to 4 gates, while an edge triggered D flip-flop with reset or preset input is equivalent to 8 gates.

In order to perform a unified quantitative comparison of the techniques, we define the effectiveness metric in a way similar to [24,25] as follows. Let  $G$  can be any one of the techniques proposed in [9–11,24,25] and the proposed here. Let  $HO_n(G)$  denote the hardware overhead of  $G$  and  $t_n(G)$  denote the time required by  $G$  to generate the  $n$ -bit pairs. Since the hardware overhead is of the order  $O(n)$ , we define the *effective hardware overhead*,  $e_{HO_n}(G)$  as a metric of the hardware overhead as follows.

$$e_{HO_n}(G) = \frac{HO_n(G)}{n}$$

Similarly, since the *time* in clock cycles is of the order  $O(2^{2n})$  we define the *effective time*,  $e_{t_n}(G)$  as follows:

$$e_{t_n}(G) = \frac{\log(t_n(G))}{2n}$$

We integrate the above two metrics into the *effectiveness*  $E_n(G)$ :

$$E_n(G) = \frac{1}{e_{HO_n}(G) \times e_{t_n}(G)} = \frac{1}{\frac{HO_n(G)}{n} \times \frac{\log(t_n(G))}{2n}} = \frac{2n^2}{HO_n(G) \times \log(t_n(G))}$$

**Table 6**

Comparison of two-pattern generation techniques.

	TPG based on	Existing modules	h/w (gates)	Time (cycles)
[9]	NFSR( $2n$ )	Reg( $n$ )	$14 \times n$	$2^n \times 2^n$
[10]	MISR( $n$ ) + Cnt( $n$ )	Reg( $n$ ) MISR( $n$ )	$16 \times n$ $12 \times n$	$2^n \times (2^n - 1)$
[11]	LFSR( $2n$ )	Reg( $n$ )	$12 \times n$	$2^n \times 2^n$
	CA( $2n$ )	Reg( $n$ )	$18 \times n$	
[24]	Acc( $n$ ) + LFSR( $n$ )	Acc( $n$ ) + Reg( $n$ )	$6 \times n + 22$	$2^n \times (2^n - 1)$
	Acc( $n$ ) + Cnt( $n$ )	Acc( $n$ ) + Cnt( $n$ )		
[25]	Acc( $n$ ) + Cnt( $n$ )	Acc( $n$ ) + Cnt( $n$ )	$6 \times n + 32$	$2^n \times (2^n - 1)$
APT	Acc( $n$ ) + Cnt( $n$ )	Acc( $n$ ) + Cnt( $n$ )	$5 \times n + 18$	$2^n \times 2^n$
		LFSR	$5 \times n + 18$	$2^n \times 2^n$

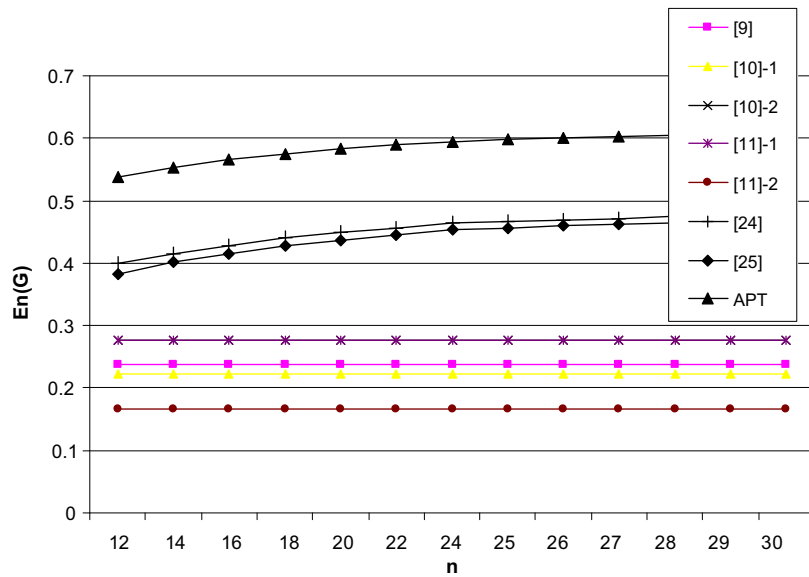


Fig. 7. Comparison of two-pattern generation schemes ( $E_n(G)$ ) for various values of the CUT inputs.

Since it is desirable that both  $HO_n(G)$  and  $t_n(G)$  be as low as possible, the higher the value of  $E_n(G)$ , the more effective is  $G$ . In Fig. 7  $E_n(G)$  is presented for each one of the techniques for various values of the inputs of the CUT. From Fig. 7 it is derived that APT is the most effective of the techniques that have been presented in the literature for the generation of vector pairs, with respect to the hardware overhead and the time required to complete the test. It should be noted that, with respect to the second scheme (the one proposed in [24]) the increase of the effectiveness ranges from 21% to 26% for the values of  $n$  under consideration, and increases with the value of  $n$ .

In order to further validate the proposed scheme, we have implemented and performed simulations of the schemes following. The schemes were implemented for generators with 4, 8, and 16 stages. The implementation and simulation results for the modules exceeding the existing modules are presented in Table 7. In Table 7 we have group d the results in three in the first column we present the number of stages for the experiment; in the second and third columns we present the power (in watts  $\times 10^{-4}$ ) and area (in  $\mu\text{m}^2$ ) of the scheme proposed in [24], while in the fourth and fifth columns the respective data for the proposed scheme are presented. In the sixth column we present the reduction in the Power  $\times$  Area metric achieved by the proposed scheme.

## 5. Case study: Two-pattern testing of ROM modules

Read-Only Memories (ROMs) are commonly embedded into current VLSI chips and constitute critical parts in complex circuits; therefore high-quality testing is required. Typical chips contain a large number of embedded small-to-medium sized memories and very few large blocks. Practical testing schemes for ROMs use exhaustive testing [30]. This kind of

Table 7  
Implementation results of the schemes.

$n$	[9]	[10]-MISR	[11]-LFSR	[11]-CA	[24]	[25]	APT
<i>Power (watts <math>\times 10^{-4}</math>)</i>							
4	5228	2948	2868	2868	1841	2893	1204
8	5708	9121	5680	1014	1841	2942	1126
16	1136	16,680	5684	1621	1846	2950	1132
32	2272	33,050	5684	3266	1846	2879	1126
<i>Area (<math>\mu\text{m}^2</math>)</i>							
4	768	792	727	792	504	695	417
8	1496	1650	1427	1630	565	764	443
16	2963	3276	2825	3256	687	894	512
32	5911	6553	5647	6533	951	1166	638
<i>Power <math>\times</math> area</i>							
4	4,017,195	2,337,174	2,087,044	2,273,750	928,048	2,011,214	502,188
8	8,539,739	15,055,123	8,105,360	1,653,124	1,040,349	2,248,571	498,931
16	3,366,877	54,658,692	16,060,710	5,278,787	1,268,387	2,638,480	579,810
32	13,430,474	216,599,785	32,098,116	21,338,084	1,756,100	3,358,929	718,726

**Table 8**

Percentage hardware overhead and test time vs. ROM size.

#Words (K)	Word width				Test time (s)
	8 (%)	16 (%)	32 (%)	64 (%)	
8	0.48	0.24	0.12	0.06	0.33
16	0.26	0.13	0.06	0.03	1.34
20	0.22	0.11	0.05	0.03	2.09
32	0.14	0.07	0.03	0.02	5.37
40	0.11	0.05	0.03	0.01	8.39
64	0.07	0.04	0.02	0.01	21.48

testing is enough to cover all logically testable (i.e. stuck-at) faults [30]. Consequently, when delay and/or stuck-open faults are considered, exhaustive two-pattern testing needs to be deployed to ensure absence of such faults.

In order to evaluate the applicability of APT, we shall calculate the overhead of the BIST hardware over the overhead of the ROM. For the calculations we have considered that a ROM bit is equivalent to 1/4 gates, as has been also considered in [31,32].

Results are presented in Table 8. In the first column of Table 8 we present the ROM size (number of words); in the second through the fifth column we present the hardware overhead of the proposed scheme over the hardware overhead of the ROM. In the last column of Table 8 we present the test time (in seconds) required to complete the two-pattern test for each memory size. For the calculations, a 200 MHz clock has been considered.

From Table 8 it is derived that the hardware overhead of the proposed scheme is practically negligible (<0.5%). Furthermore, the two-pattern test completes within less than 22 s even for medium-size ROMs. Therefore, taking into account that typical chips contain a large number of embedded small-to-medium sized memories and very few large blocks, we can conclude that the proposed scheme may constitute a useful component in the arsenal of BIST schemes for testing ROMs for faults that cannot be detected with tests that target the stuck-at fault model.

## 6. Conclusions

A novel algorithm for the generation of two-pattern tests is presented. The proposed algorithm can be implemented in hardware utilizing modules existing in data path circuitry, and compares favorably to techniques already presented in the open literature with respect to the hardware and the test time from 21% to 26%. Furthermore, implementations and experiments carried out revealed that control module of the proposed scheme presents a significant decrease (over 45%) over the other scheme with respect to the Area  $\times$  Power metric. A case study on the two-pattern testing of ROM modules revealed that for small-to-medium size ROMs the two-pattern testing can complete within reasonable time with negligible (less than 0.5%) hardware overhead.

## Acknowledgments

This research has been co-financed by the European Union (European Social Fund – ESF) and Greek national funds through the Operational Program “Education and Lifelong Learning” of the National Strategic Reference Framework (NSRF) – Research Funding Program: Archimedes III: Investing in knowledge society through the European Social Fund.

## References

- [1] Bardell PH, McAnney WH, Savir J. Built-in test for VLSI: pseudorandom techniques. John Wiley and Sons; 1987.
- [2] Wang L-T, Wu C-W, Wen X. VLSI test principles and architectures: design for testability (systems on silicon). Morgan Kaufmann; 2006.
- [3] Li A, Hong B. On-line control flow error detection using relationship signatures among basic blocks. *Comput Electr Eng Jan*. 2010;36(1):132–41.
- [4] Zhan W, Liang H, Jiang C, Huang Z, El-Maleh A. A scheme of test data compression based on coding of even bits marking and selective output inversion. *Comput Electr Eng* 2010;36(5):969–77.
- [5] Koneman B, Mucha J, Zwiehoff G. Built-in logic block observation technique. In: *Proc of the IEEE international test conference (ITC'79)*; 1979. p. 37–41.
- [6] Hortensius PD, McLeod R, Pries W, Miller DM, Card H. Cellular automata-based pseudorandom generators for built-in self-test. *IEEE Trans CAD Integr Circ Syst* 1989;8(8):842–59.
- [7] Stroele A. A self test approach using accumulators as test pattern generators. In: *Proc of int symposium on circuits and systems*; 1995. p. 2120–23.
- [8] Wadsack R. Fault modeling and logic simulation of CMOS and nMOS integrated circuits. *Bell Syst Tech J* 1978;57(5):1449–74.
- [9] Starke C. Built-in test for CMOS circuits. In: *Proc of IEEE Int test conference*; 1984. p. 309–14.
- [10] Vuksic A, Fuchs K. A new BIST approach for delay fault testing. In: *Proc of European design and test conference*; 1994. p. 284–88.
- [11] Chen Ch, Gupta S. BIST test pattern generators for two-pattern testing-theory and design algorithms. *IEEE Trans Comput* 1996;45(3):257–69.
- [12] Gharaybeh M, Bushnell M, Agrawal V. Parallel concurrent path delay fault simulation using single-input change patterns. In: *Proc of 9th IEEE international conference on VLSI design*; 1996. p. 426–31.
- [13] Gharaybeh M, Bushnell M, Agrawal V. A parallel-vector concurrent fault simulator and generation of single-input-change tests for path-delay faults. *IEEE Trans CAD Integr Circ Syst* 1998;17(9):873–6.
- [14] Li X, Cheung PYS. Exploiting BIST approach for two-pattern testing. In: *Proc of the 7th Asian test symposium*; 1998. p. 424–29.
- [15] Das D, Chaudhuri I, Bhattacharya B. Design of an optimal test pattern generator for built-in self testing of path delay faults. In: *Proc of 11th conference on VLSI design*; 1998. p. 205–10.

- [16] Girard P, Landrault C, Pravossoudovitch S, Virazel A. Comparison between random and pseudo random generation for BIST of delay, stuck-at and bridging faults. In: Proc of IEEE on-line testing workshop; 2000. p. 121–26.
- [17] Virazel A, David R, Girard P, Landrault C, Pravossoudovitch S. Delay fault testing: choosing between random SIC and random MIC sequences. In: Proc of IEEE European test workshop; 2000. p. 9–14.
- [18] Rahaman H, Das D, Bhattacharya B. Transition count based BIST for detecting multiple stuck-open faults in CMOS circuits. In: Proc of the 2nd IEEE Asia pacific conference on ASICs; 2000. p. 307–10.
- [19] Lu S, Lu M. Testing iterative logic arrays for delay faults with a constant number of patterns. In: Proc of 4th int symposium on electronic materials and packaging; 2002. p. 492–98.
- [20] Gizdarski E. Detection of delay faults in memory address decoders. J Electron Test: Theor Appl 2002;16(4):381–7.
- [21] David R, Girard R, Landrault C, Pravossoudovitch S, Virazel A. On using efficient test sequences for BIST. In: Proc of VLSI test symposium; 2002. p. 145–50.
- [22] Puczek M, Yarmolik VN. Two-pattern test generation with low power consumption based on LFSR. Inform Process Secur Syst 2005;159–66.
- [23] Namba K, Ito H. Deterministic delay fault BIST using adjacency test pattern generation. IEICE Trans Inform Syst 2005;E88-D(9):2135–42.
- [24] Voyiatzis I, Paschalis A, Nikolos D, Halatsis C. Accumulator-based BIST approach for two-pattern testing. J Electron Test: Theor Appl 1999;15(3):267–78.
- [25] Voyiatzis I. Accumulator-based pseudo-exhaustive two-pattern generation. J Syst Archit 2007;53(11):846–60.
- [26] Voyiatzis I, Gizopoulos D, Paschalis A. Recursive pseudo-exhaustive two-pattern generation. IEEE Trans VLSI Syst 2010;18(1):142–52.
- [27] Rajsiki J, Tyszer J. Test responses compaction in accumulators with rotate carry adders. IEEE Trans CAD Integr Circ Syst 1993;12(4):531–9.
- [28] Zimmermann R. Efficient VLSI implementation of modulo ( $2^n \pm 1$ ) addition and multiplication. In: Proc 14th IEEE symposium on computer arithmetic; 1999. p. 158–67.
- [29] Zimmermann R. Binary adder architectures for cell based VLSI and their synthesis. PhD thesis. Swiss Federal Institute of Technology (ETH) Zurich, HartungGorre Verlag; 1998.
- [30] Zorian Y, Ivanov A. An effective BIST scheme for ROM's. IEEE Trans Comput 1992;41(5):646–53.
- [31] Kalligeros E, Kavousianos X, Bakalis D, Nikolos D. An efficient seeds selection method for LFSR-based test-per-clock BIST. In: Proc of international symposium on quality electronic design; 2002. p. 261–66.
- [32] Huang LR, Jou JY, Kuo SY. Gauss-elimination-based generation of multiple seed-polynomial pairs for LFSR. IEEE Trans CAD 1997;16(9):1015–24.

**Ioannis Voyiatzis** received his B.Sc. degree in Informatics, M.Sc. degree in Electronics Automation, and Ph.D. degree in Computer Science, all from the Department of Informatics and Communications, University of Athens, Greece. He is currently with the Department of Informatics, Technological Educational Institute of Athens, Greece. His research interests include Built-In Self Test, on-line testing and fault tolerance of circuits. He is author of technical papers published in refereed international scientific journals and conference proceedings and 10 books in the IT field. He is a member of IEEE and has been listed in the Marquis 'Who's who in Science and Engineering' and 'Who's Who in the World'.

**Costas Efstathiou** received the B.S. in Physics and the M.S. degree in Electronics both from the University of Athens and the Ph.D. degree in Computer Science from the University of Thessaloniki, Greece. Since 1994 he holds a Professor position in the Department of Informatics of the Technological Institute of Athens. His research interests include computer arithmetic, fault-tolerant computer systems and computer networks.

**Hera Antonopoulou** received the BSc. in Mathematics and the PhD degree from the Department of Computer Engineers and Informatics from the University of Patras. She has taught courses in the University of Patras, the Technological Educational Institute of Patras, and the Hellenic Open University of Greece and worked as a Researcher in the Academic Computer Technology Institute. She has more than 30 Scientific Publications in International Journals and Conferences with reviewers and has participated in over 15 European research projects.

**Athanasios Milidonis** was born in 1976 in Athens, Greece. He received a Diploma in Electrical & Computer Engineering in 2000, MSc in Hardware and Software for Integrated Systems in 2002 and PhD in 2007 from University of Patras in Greece. Since 2008 he has been working as software engineer for Nokia Siemens Networks, as a Lab instructor in National Technical University of Athens and as a Visiting Lecturer in the University of Peloponnese. His research interests are: compiler techniques for low power consumption and high performance, VLSI design, Computer Architecture and Telecommunication design.