Microelectronics Journal 41 (2010) 487–493

Contents lists available at ScienceDirect

Microelectronics Journal

journal homepage: www.elsevier.com/locate/mejo

An efficient architecture for accumulator-based test generation of SIC pairs

I. Voyiatzis [a,*], C. Efstathiou [a,b]

[a] Technological Educational Institute of Athens, Informatics, Greece
[b] University of Central Greece, Lamia, Greece

ARTICLE INFO

Article history:
Received 15 September 2008
Accepted 22 May 2009
Available online 12 June 2009

Keywords:
BIST
Two-pattern testing
Delay fault testing
Stuck-open testing
Ling adders

ABSTRACT

Research conducted over the years has shown that the application of single input change (SIC) pairs of test patterns for sequential, i.e. stuck-open and delay fault testing is extremely efficient. In this paper, a novel architecture for the generation of SIC pairs is presented. The implementation of the proposed architecture is based on Ling adders that are commonly utilized in current data paths due to their high-operating speed. Since the timing characteristics of the adder are not modified, the presented architecture provides a practical solution for the built-in testing of circuits that contain such adders.

1. Introduction

While every VLSI design project has its own unique set of goals, there is a universal need for reliability in the finished product. Built-in self test (BIST) [1] techniques constitute an attractive and practical solution. Advantages of BIST include the possibility of performing at-speed testing, very high-fault coverage, elimination of test generation effort and less reliance on expensive external testing equipment for applying and monitoring test patterns. Therefore, BIST can drive down the cost of testing.

It is widely known that a large class of physical defects cannot be modeled as stuck-at faults. For example, a transistor stuck-open fault in a CMOS circuit can convert a combinational circuit under test (CUT) into a sequential one [2], while a delay fault may cause circuit malfunction at clock speed, although it does not affect the steady-state operation [3]. Detection of such faults requires two-pattern tests.

In the literature, two types of two-pattern tests have been investigated, multiple input change (MIC) and single input change (SIC) pairs. In SIC pairs the first pattern differs from the second in exactly one bit. The utilization of SIC pairs for the detection of stuck-open and delay faults holds some very interesting properties and has been studied by a number of researchers [4–15]. Smith [4] proved that SIC tests are sufficient to detect all robustly detectable path delay faults; Wang and Gupta [5] proved that SIC pairs provide higher pseudorandom robust path delay fault coverage than MIC pairs. In other words, if a certain number of pairs is applied to the inputs of a circuit under test, the achieved fault coverage is higher if the pairs are SIC, than if the pairs are MIC. Furthermore, a number of related works [6–17] indicate that the utilization of SIC pairs for testing delay and stuck-open faults compares favorably to the utilization of MIC pairs. Experimental data indicate the effectiveness of SIC pairs in identifying path delay faults in some of the well-known ISCAS 89 benchmarks [18]. In the field of RAM testing, Gizdarski utilized SIC pairs in order to test delay faults in the address decoders of RAM memories [19].

Accumulator structures composed of an adder and a register module commonly exist in current VLSI circuits, e.g. in the data path of embedded processors, or in digital signal processing (DSP) chips [20,21]. The utilization of such structures for compression of the CUT responses [22–25], or generation of test patterns [26–29] in BIST results in low hardware overhead and low impact on the circuit normal operating speed. Accumulator-based techniques that target the detection of stuck-at faults have been presented in [26–28]; generation of MIC two-pattern tests has been achieved in [29]. In [30,31] the generation of SIC pairs based on an accumulator whose inputs are driven by a barrel shifter was proposed. Such structures are commonly found in typical DSP cores.

In the data paths of the modern processors, binary adders are typically implemented using parallel-prefix architectures based on conventional [32–34] or Ling carries [35–37], in order to increase the operating speed.

In this paper, we propose an accumulator-based SIC-pair generator where the adder of the accumulator is based on a properly modified Ling adder. One of the most important advantages of the proposed scheme is that the modifications imposed on the Ling adder do not alter its timing characteristics.

The paper is organized as follows. In Section 2, the algorithm utilized for the generation of SIC pairs is reviewed and the

* Corresponding author.
E-mail address: voyageri@otenet.gr (I. Voyiatzis).

0026-2692/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.mejo.2009.05.003

implemented architecture is presented; in Section 3, the hardware implementation is presented. In Section 4, the techniques presented in the literature for the generation of SIC pairs are compared. Finally, in Section 5 we conclude the paper.

2. Algorithm and BIST architecture

Let $V = v_{n-1}vn_{-2}\ldots v_1v_0$ be an n-bit binary vector. We denote with V^i the binary vector that differs from V in the i-th bit ($0 \le n \le n-1$). For example, if $V = 010$ then $V^0 = 011$, $V^1 = 000$ and $V^2 = 110$. Furthermore, we denote with $\bar{V}$ the vector that differs from V in all n bits. For example, for $V = 010$ and $\bar{V} = 101$.

Definition 1:. The S-sequence (or SIC-sequence) of V is the sequence of n vectors $\{V_0, V_1 \ldots V_{n-1}\}$, where

$$V_i = \begin{cases} V^0 & \text{if } i = 0 \\ V^i_{i-1} & \text{else} \end{cases}$$

For example, for $V = 000$, S-sequence $(V) = \{001, 011, 111\}$. It is easy to note that the S-sequence (V) always ends to $\bar{V}$.

Definition 2:. The double-SIC sequence (DS-sequence) of V is the sequence of $(2n+1)$—vectors comprised of V, the S-sequence of V, and the S-sequence of $\bar{V}$.

For example, for $V = 000$, DS-sequence $(V) = \{000, 001, 011, 111, 110, 100, 000\}$. It is easy to note that the DS-sequence (V) always ends with V.

In Fig. 1, we present the algorithm utilized to generate the SIC pairs. According to the theory proposed in [31], the algorithm generates all n-bit SIC pairs within $(n+1/2) \times 2^n$ cycles.

As an example, the patterns generated by the algorithm of Fig. 1 for $n = 3$ are presented in Table 1.

The algorithm of Fig. 1 can be implemented with a (properly modified) accumulator whose inputs are driven by a barrel shifter. The modified accumulator, shown in Fig. 2, consists of a modified n-bit adder (denoted as S-adder) and a register, and takes as

```
ARN(int n)
{ N=2^n;
for (V=0; V<N/2; V++)
  {
    apply(V);
    S-sequence(V);
    S-sequence(V̄);
  }
S-sequence(X)
{
  for (i=0; i<n; i++)
      {X=X^i; apply(X);}
  }
```

Fig. 1. SIC-pair generation algorithm.

Table 1
Patterns generated for $n = 3$.

$V = 0$	$V = 1$	$V = 2$	$V = 3$
000	001	010	011
001	000	011	010
011	010	001	000
111	110	101	100
110	111	100	101
100	101	110	111
000	001	010	011

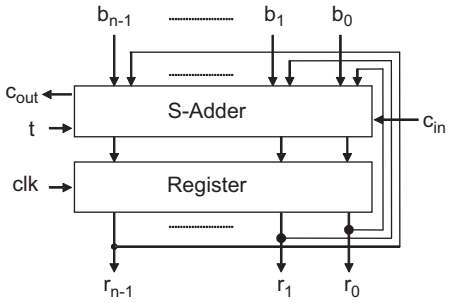

Fig. 2. Accumulator utilized for the implementation of the proposed scheme.

inputs an n-bit vector B, a carry-in c_{in} and a clock input, clk. The output of the accumulator is a vector R and a carry-out signal c_{out}. The S-adder has an additional test input, t and operates as follows: when $t = 0$, the S-adder operates as an ordinary n-bit adder; when $t = 1$, the S-adder operates as a series of n two–input XOR gates, i.e. each output of the S-adder is the eXclusive-OR (XOR) of the corresponding inputs.

The n-bit SIC pair generator implemented in this work is presented in Fig. 3. The control inputs $\{f_{k-1}, \ldots, f_1 f_0\} = F$, $(k = \lceil \log_2 n \rceil)$ of the barrel shifter are driven by a k-bit counter that counts modulo n.

The module presented in Fig. 3 operates as follows. Initially, the binary vector 00…01 is driven to the inputs of the barrel shifter and the outputs of the Accumulator $r_{n-1}, r_{n-2}, \ldots r_0$ are set to 0. The counter is initially set to 0 and increments in every clock cycle. During the first n clock cycles, the vectors {00…01, 00…010, …, 10…0} are applied to the inputs of the accumulator and the S-sequence (0) is generated. At this time, the output of the accumulator is 11…1. Then the counter starts counting from 1 to $n-1$ again, and the S-sequence (2^n-1) is generated and applied to the CUT. Then, the t signal is disabled by the control module, the accumulator accumulates the vector 00…01and the DS-sequence (1) commences.

The control unit of the proposed architecture is given in Fig. 4. Initially, the flip-flop is set to '0'. The "$F = n-1$" signal indicates that the counter has reached the value $n-1$.

The operation of the module of Fig. 3 is analyzed in Table 2 for $n = 3$. The first column of Table 2 presents the number of the cycle, the second column presents the value of the t signal, the third column presents the output of the counter that are driven to the f_i inputs of the shifter, the fourth column presents the output of the shifter and the sixth column the output of the accumulator. The fifth column presents the operation of the accumulator, either eXclusive-OR ($\oplus$) or addition (ADD), which depends on the value of the t signal. For the first six cycles, the next value of the accumulator is calculated as the bit-by-bit eXclusive-OR of the current value of the accumulator and the value of the shifter.

At clock cycle 7, the signal t is disabled, the accumulator operates normally, and the pattern 001 is generated at the outputs of the accumulator. The same holds for clock cycles 14, 21 and 28.

3. Hardware implementation

In the sequel, we shall present an implementation of the scheme based on a modified Ling adder [37]. Prior to presenting the modifications of a Ling adder to implement the S-adder, we revise some introductory concepts on parallel-prefix addition.

Let $A = a_{n-1}a_{n-2}\ldots a_1a_0$ and $B = b_{n-1}b_{n-2}\ldots b_1b_0$ be two n-bit numbers to be added and let $S = s_{n-1}s_{n-2}\ldots s_1s_0$ be their sum. A CLA adder can be considered as a three-stage circuit. The first-stage computes the carry generate $g_i = a_i\, b_i$, the carry propagate

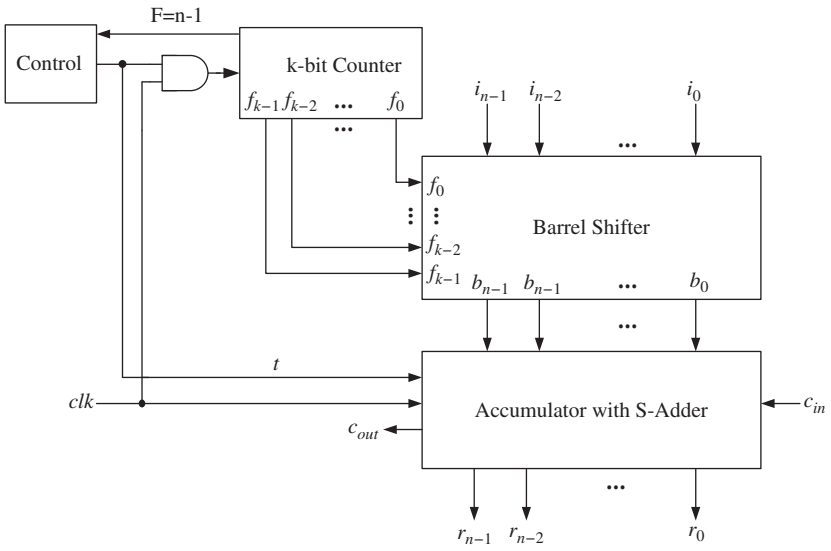

Fig. 3. n-bit SIC pair generator.

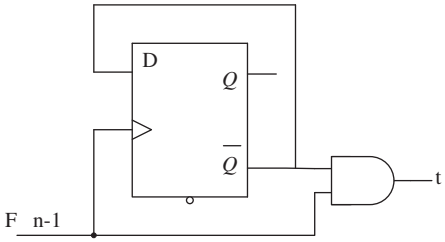

Fig. 4. The control unit.

Table 2
Operation of 3-bit SIC pair generation.

Cycle	t	F	B	Oper	R
1	1	00	001	⊕	000
2	1	01	010	⊕	001
3	1	10	100	⊕	011
4	1	00	001	⊕	111
5	1	01	010	⊕	110
6	1	10	100	⊕	100
7	0	00	001	ADD	000
8	1	00	001	⊕	001
9	1	01	010	⊕	000
10	1	10	100	⊕	010
11	1	00	001	⊕	110
12	1	01	010	⊕	111
13	1	10	100	⊕	101
14	0	00	001	ADD	001
15	1	00	001	⊕	010
16	1	01	010	⊕	011
17	1	10	100	⊕	001
18	1	00	001	⊕	101
19	1	01	010	⊕	100
20	1	10	100	⊕	110
21	0	00	001	ADD	010
22	1	00	001	⊕	011
23	1	01	010	⊕	010
24	1	10	100	⊕	000
25	1	00	001	⊕	100
26	1	01	010	⊕	101
27	1	10	100	⊕	111
28	0	00	001	ADD	011

$p_i = a_i + b_i$, and half-sum $d_i = a_i \oplus b_i$ bits for the case of inclusive-OR adders or the carry generate $g_i = a_i\ b_i$ and carry propagate $p_i = d_i = a_i \oplus b_i$ bits, for the case of exclusive-OR adders, where $0 \le i \le n-1$. The symbols $\bullet$, +, $\oplus$ denote the logic AND, OR, and eXclusive-OR operations, respectively. The second stage of the adder computes the carry bits c_i, using the carry generate and propagate bits, while the final stage computes the sum bits s_i.

A large variety of carry computation algorithms has been proposed [32–34]. Among them, parallel-prefix architectures are more suitable for VLSI implementation, since they rely on the use of simple cells and maintain regular connections. Carry computation is transformed into a prefix problem using the associate operator $\circ$, which associates pairs of propagate and generate bits according to the relation

$$(g_i, p_i) \circ (g_j, p_j) = (g_i + p_i \cdot g_j, p_i \cdot p_j)$$

In a sequence of consecutive associations of (g, p) terms the notation $(g_{k:j}, p_{k:j})$ is used to denote the group generate and the group propagate terms for the bits k, $k-1$, …, j. That is,

$$(g_{k:j}, p_{k:j}) = (g_k, p_k) \circ (g_k, p_k) \circ (g_{-1k}, p_{k-1}) \circ \ldots \circ (g_j, p_k)$$

Each carry bit c_i is equal to $g_{i:0}$. Parallel-prefix adder architectures are based on the use of tree structures for the computation of carry bits. The sum bits are implemented as $s_i = d_i \oplus c_{i-1}$.

3.1. S-adder architecture based on Ling carries

A variant of the carry look-ahead computation is based on Ling carries [35], leading to faster adder architectures. The Ling carries, defined in [35] as $H_i = c_i + c_{i-1}$, can easily be derived for the case of the inclusive-OR adders as follows. Since the carry propagate functions are defined for these adders as $p_i = a_i + b_i$ the relation $p_i = g_i \cdot p_i$ holds and we have

$$\begin{aligned} c_i &= g_i + p_i g_{i-1} + p_i p_{i-1} g_{i-2} + \cdots + p_i p_{i-1} \cdots p_1 p_0 \\ &= p_i(g_i + g_{i-1} + p_{i-1} g_{i-2} + \cdots + p_{i-1} \ldots p_1 p_0) \\ &= p_i H_i \end{aligned}$$

where $H_i = g_i + g_{i-1} + p_{i-1} g_{i-2} + \cdots + p_{i-1} \ldots p_i p_0$ are the Ling carries.

An efficient implementation of the sum bits using the Ling carries is derived according to the relation $s_i = \overline{H_{i-1}} d_i + H_{i-1}(d_i \oplus p_{i-1})$ given in [38] ($\bar{x}$ denotes the inverse of x).

Parallel-prefix adder architectures based on Ling carries are introduced in [36] and high-speed implementations are proposed in [37]. Two consecutive Ling carries are computed in parallel-prefix form [37] as $H_i = G_{i:0}$ and $H_{i+1} = G_{i+1:1}$ for i even, where $(G_{i:0}, P_{i-1:-1}) = (G_i, P_{i-1}) \circ (G_{i-2}, P_{i-3}) \circ \ldots \circ (G_0, P_{-1})$ and

$(G_{i+1:0},P_{i:0}) = (G_{i+1},P_i)\circ(G_{i-1},P_{i-2})\circ\ldots\circ(G_1,P_0)$, for $G_i = g_i+g_{i-1}$ and $P_i = p_i \cdot p_{i-1}$, with $g_{-1} = 0$, $p_{-1} = 1$, $G_{-1} = 0$ and $P_{-1} = 1$.

For example, the parallel-prefix expressions for the Ling carries of an 8-bit adder are as follows:

$$H_0 = (G_0, P_{-1})$$
$$H_1 = (G_1, P_0)$$
$$H_2 = (G_2, P_1) \circ (G_0, P_{-1})$$
$$H_3 = (G_3, P_2) \circ (G_1, P_0)$$
$$H_4 = (G_4, P_3) \circ (G_2, P_1) \circ (G_0, P_{-1})$$
$$H_5 = (G_5, P_4) \circ (G_3, P_2) \circ (G_1, P_0)$$
$$H_6 = (G_6, P_5) \circ (G_4, P_3) \circ (G_2, P_1) \circ (G_0, P_{-1})$$
$$H_7 = (G_7, P_6) \circ (G_5, P_4) \circ (G_3, P_2) \circ (G_1, P_0)$$

The carry-in bit c_{in} can be included in the adder architecture, either by adding a fast carry increment stage, or by treating c_{in} as an extra bit in the preprocessing stage of the adder [37]. For the case of carry increment stage the two consecutive Ling carries are computed as $H_i = G_{i:0}+P_{i-1:-1}c_{-1}$ and $H_{i+1} = G_{i+1:1}+P_{i:0}c_{-1}$ for i even. The second case is derived by setting $g_{-1} = c_{in}$.

The efficiency of the architecture in [37] relies on the use of AND–OR and OR–AND complex gates to implement the $G_i = a_ib_i+a_{i-1}b_{i-1}$ and $P_i = (a_i+b_i)(a_{i-1}+b_{i-1})$ terms, which minimizes the in-series transistors in the critical path. In addition, the computation of the sum bits according to the relation $s_i = \overline{H_{i-1}}d_i + H_{i-1}(d_i \oplus p_{i-1})$ is implemented using a multiplexer that selects either d_i or $d_i \oplus p_{i-1}$. This circuit introduces the same delay as the XOR gate that computes the s_i bits of conventional adders. The carry output bit of the adder is derived using an extra AND gate to implement the relation $c_{n-1} = p_{n-1}H_{n-1}$.

The modification of the g_i terms to convert the Ling adders to S-adders proposed in [31] leads to modification of the terms $G_i = g_i+g_{i-1} = a_ib_i+a_{i-1}b_{i-1}$ to $G_i^* = g_i^* + g_{i-1}^* = a_ib_i\bar{t} + a_{i-1}b_{i-1}$, or equivalently to $G_i^* = (a_ib_i + a_{i-1}b_{i-1})\bar{t}$. It is evident that the G_i^* terms are more complicated than their corresponding G_i terms and can alter the timing characteristics of the adder.

In the following, an alternative methodology is proposed that leads to S-adder architectures that operate at the same speed as their corresponding parallel-prefix adders based on Ling carries. According to the definition of the S-adders in [31], their carries can be expressed as $c_i^* = \bar{t}c_i = \bar{t}p_iH_i$ where H_i are the Ling carries. Using this relation and following a methodology similar to that in [35] the sum bits of the S-adder can be computed as follows:

$$\begin{aligned} s_i &= d_i \oplus c_{i-1}^* = d_i \oplus \bar{t}p_iH_{i-1} \\ &= \overline{d_i}\,\bar{t}p_{i-1}H_{i-1} + d_i\overline{(\bar{t}p_{i-1}H_{i-1})} = \overline{d_i}\bar{t}p_{i-1}H_{i-1} + d_i\overline{(\bar{t}p_{i-1})} + d_i\overline{H_{i-1}} \\ &= \overline{d_i}\bar{t}p_{i-1}H_{i-1} + d_i\overline{(\bar{t}p_{i-1})}(H_{i-1} + \overline{H_{i-1}}) + d_i\overline{H_{i-1}} \\ &= \overline{d_i}\bar{t}p_{i-1}H_{i-1} + d_i\overline{(\bar{t}p_{i-1})}H_{i-1} + d_i\overline{(\bar{t}p_{i-1})}\,\overline{H_{i-1}} + d_i\overline{H_{i-1}} \end{aligned}$$

or

$$s_i = (d_i \oplus (\bar{t}p_{i-1}))H_{i-1} + d_i\overline{H_{i-1}}$$

The additional hardware required for the implementation of the s_i terms according to the derived relation is a series n two-input AND gates which implement the $\bar{t}p_{i-1}$ functions. No extra delay is imposed by the proposed modification.

The proposed architecture of an 8-bit S-adder based on Ling carries is shown in Fig. 5. The parallel-prefix computation of the Ling carries is derived according to the Lander–Fischer algorithm [39]. It is trivial to see that, $c_{out} = p_7H_7$.

The logic-level implementations of the basic cells used in the proposed architecture are shown in Fig. 6.

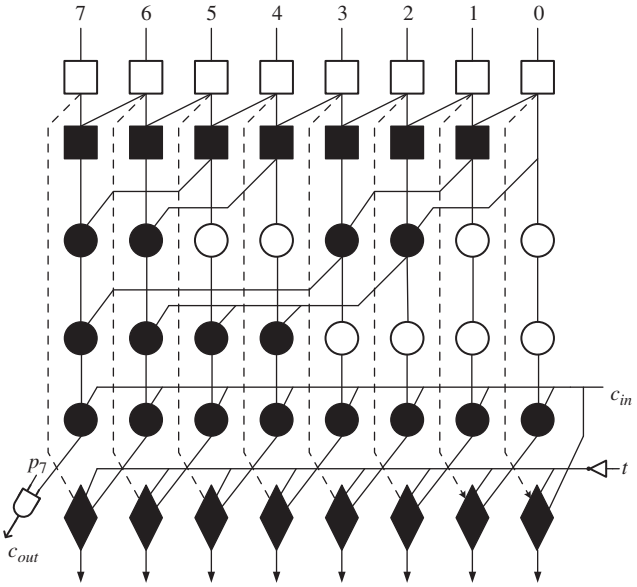

Fig. 5. Modified 8-bit Ling adder.

3.2. S-adder architectures based on conventional carries

The proposed methodology is also applicable to the case of S-adder architectures based on conventional CLA or parallel-prefix carry computation. As noted earlier, the architecture of the S-adder in [31] is based on the modification of the generate functions from $g_i = a_ib_i$ to $g_i^* = a_ib_i\bar{t}$, where t is the test input. The proposed S-adder architecture is derived as follows. According to the definition of the S-adder [31] the carries can be expressed as the logical AND of the conventional carries and the inverted test signal, that is, $c_i^* = \bar{t}c_i$. Instead of modifying the g_i terms the sum bits are computed, following a methodology similar to that for the S-adders based on Ling carries, as $s_i = \overline{c_{i-1}}d_i + c_{i-1}(d_i \oplus \bar{t})$. According to this relation each s_i term can be implemented using a 2-to-1 multiplexer controlled by c_{i-1}. Since the multiplexer is of almost equal delay to an XOR gate and the signals d_i and $d_i \oplus \bar{t}$ are computed in fewer logic levels than c_{i-1}, the timing characteristics of the adder are not changed by the proposed modification.

3.3. Calculation of the hardware overhead

The S-adder architecture proposed here, instead of modifying the G_i terms, introduces the test line t in the last stage of the adder, which computes the s_i bits. The additional hardware required is a series of n 2-input AND gates, where n is the number of bits of the accumulator–shifter. Since the G_i terms are not modified and the introduced AND gates are not in the critical path, no extra delay is imposed by the proposed architecture and the derived S-adders operate at the same speed as their corresponding ordinary adders.

The proposed S-adder architecture based on conventional carries does not modify the g_i terms. Since no extra delay is imposed by the proposed computation of the sum bits, these S-adders operate at the same speed as the original Ling adders.

4. Comparisons

In this section, the proposed scheme will be compared with the techniques proposed in [5–8,16,30,31], in terms of hardware overhead and time required to generate the entire sequence of SIC pairs.

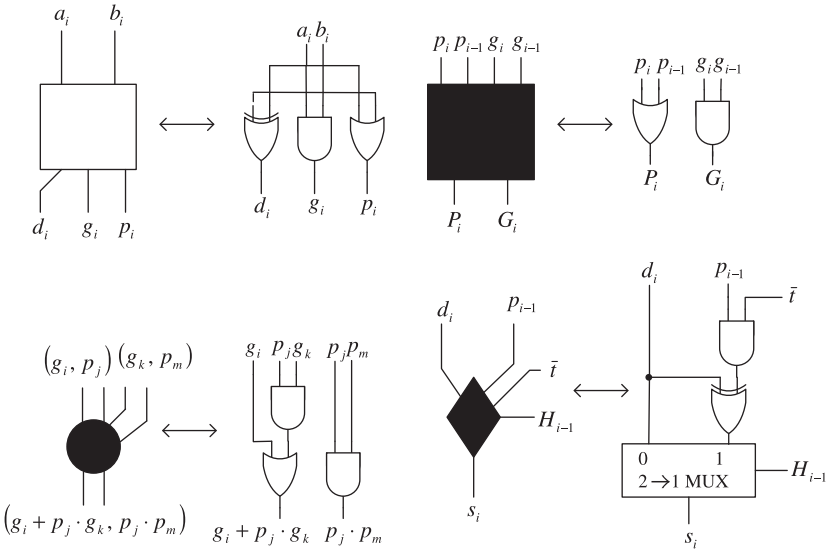

Fig. 6. Cells implemented in the proposed scheme.

For the comparisons, we shall assume that an n-bit register (consisting of n flip-flops) exists at the inputs of the n-input CUT for the case of [5–8,16], and we shall calculate the overhead imposed on these flip-flops. For the case of the schemes proposed in [30,31] and in this work, we shall assume the existence of the accumulator whose inputs are driven by a barrel shifter.

Wang and Gupta [5] utilized a $(2n+1)$-bit shift register, an n-bit LFSR and n 2-input XOR gates in order to generate the n-bit SIC pairs within $n \times 2^{n+1}$ clock cycles. Thereby, $2n+1$ scan flip-flops and n 2-input XOR gates are implemented.

In PEAT [6], an n-bit NFSR, an n-bit shift register and an n-bit shift registers with flip capability are utilized to generate the SIC pairs within $(n+1) \times 2^n$ clock cycles. To implement the technique the NFSR and n scan flip-flops with flip capability are implemented. Furthermore, the n flip-flops of the existing register are substituted by scan flip-flops.

Girard et al. [7] proposed an optimization of [5] that saves n stages of the shift register substituting them with n 2-input AND gates. The time required to generate the SIC pairs is the same as in [6].

In [8] a different approach was presented that utilizes a $(n-1)$-bit counter, an n-bit shift register and a series of n 2-input XOR gates in order to generate the SIC pairs within $(n+1/2) \times 2^n$ clock cycles. The hardware overhead of the technique is n flip-flops and n 2-input XOR gates.

In [16], Das et al. presented an optimal solution to the problem of generating SIC pairs, in the sense that the pairs are generated within time equal to the theoretical minimum, i.e. $n \times 2^n+1$. However, the hardware overhead of [16] is rather high, thus the value of the scheme lies mainly on its high theoretical significance. The hardware overhead of the scheme is, according to [16], $3n+2$ flip-flops, n XOR gates (2-input), $(2n-1)$OR gates (2-input), $(n+1)$ AND gates (2-input) and 1 NOT gate.

In [30] an accumulator-based generator was presented, that requires the transformation of the adder to an S-adder; it also requires the implementation of a counter whose inputs drive the shift control inputs of the shifter; the time required to generate the SIC pairs is $(n+2) \times 2^n$. Two implementations were presented, based on a ripple carry adder (RCA) and a CLA adder, respectively. The ripple carry implementation forces no overhead on the adder, while the implementation based on a CLA adder affects the adder timing characteristics, since the modification is performed on the critical path of the module, hence it will not be taken into account in the comparisons.

The scheme proposed in [31] utilizes an accumulator-based architecture and generates the SIC pairs within $(2n+1) \times 2^{n-1} = (n+1/2) \times 2^n$ cycles. Two implementations were also proposed there, based on a ripple carry adder and on a CLA adder. With a reasoning similar to the one followed for [30], the CLA adder-based implementation will not be taken into account in the comparisons, since it affects the adder timing characteristics.

For the comparisons, the following are taken into account [40]. A 2-input NAND/NOR gate requires 4 MOS transistors; a 2-input AND requires 6 transistors and a 2-input XOR gate can be implemented using 6 transistors. The memory elements used are considered to have set/reset capability. Thereby, the flip-flop requires 26 transistors, the scan flip-flop requires 34 transistors and the scan flip-flop with flip capability [5] requires 46 transistors. All techniques, except from the schemes proposed in [30,31] and in this work, require that the n flip-flops at the inputs of the CUT be transformed into scan flip-flops.

In Table 3 we present, for each one of the SIC-pair generation techniques (first column) the time required to generate the SIC pairs (in clock cycles, second column) the formulas used for the calculation of the hardware overhead (third column) and the hardware overhead (in transistors, fourth column).

In order to perform a quantitative comparison of the techniques, we define the following metrics. Let SG can be any one of the techniques proposed in [5–8,16,30,31] and the proposed. Let $ho_n(\mathrm{SG})$ denote the hardware overhead of SG and $t_n(\mathrm{SG})$ denote the time required by SG to generate all n-bit SIC pairs. Since the hardware overhead is of the order $\boldsymbol{O}(n)$, we define the *effective hardware overhead*, $e\_ho_n(\mathrm{SG})$ as a metric of the hardware overhead as follows:

$$e\_ho_n(SG) = \frac{ho_n(SG)}{n}$$

Similarly, since the *time* in clock cycles is of the order $\boldsymbol{O}(n \times 2^n)$, we define the *effective time*, $e\_t_n(\mathrm{SG})$

$$e\_t_n(SG) = \frac{t_n(SG)}{n \times 2^n}$$

Table 3
SIC pair generation techniques: Comparison technique.

Technique	Time cycles $\times 2^n$	Hardware overhead	
		Modules	Transistors
Peat [6]	n+1	$n \times$ (DFF+NOR)+$n \times$ $DFF_{scanwithflip}$+$n \times$ (DFF_{scan}−DFF)	84 × n
Wang [5]	2n	(2n+1) × DFF+n × XOR+n × (DFF_{scan}−DFF)	66 × n+26
Girard [7]	2n	(n+1) × DFF+n × AND+n × XOR+n × (DFF_{scan}−DFF)	42 × n
[8]	n+1/2	n × DFF+n × XOR+n × (DFF_{scan}−DFF)	40 × n
[16]	n	(2n+2) × DFF+n × XOR +(2n−1) × OR_2+(n+1) × AND_2+NOT	76 × n+52
[30] (Ripple Carry Adder)	2n+2	n × AND+2 × DFF+(⌈$\log_2 n$⌉+2) × DFF+AND+OR	12 × n+65
[31] (Ripple Carry Adder)	n+1/2	n × AND+(⌈$\log_2 n$⌉+2) × DFF+control	6 × n+36 × ⌈$\log_2 n$⌉+86
Proposed scheme (Ling Adders)	n+1/2	1 × AND+n × (MUX_{2-to-1})+(⌈$\log_2 n$⌉+2) × DFF+control	4 × n+36 × ⌈$\log_2 n$⌉+30

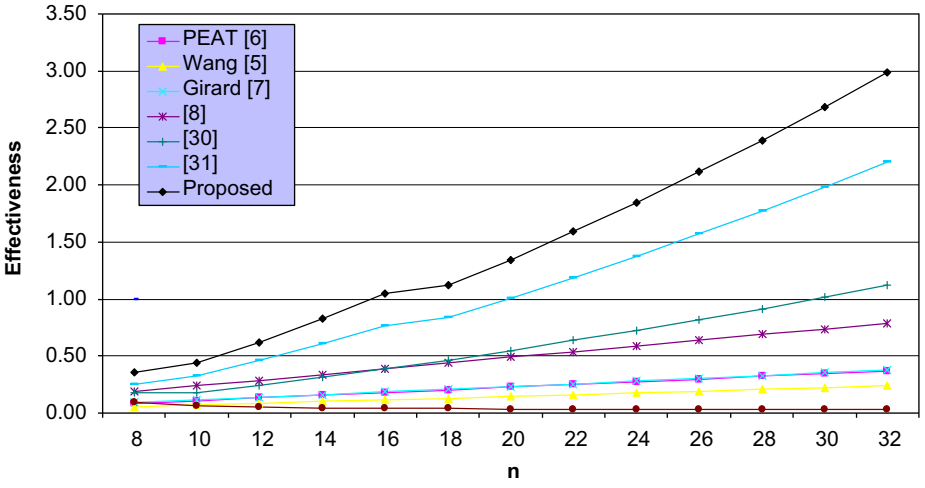

Fig. 7. SIC-pair generators: comparison.

We integrate the above two metrics into the *effectiveness* E_n(SG)

$$E_n(\mathrm{SG}) = \frac{1}{e\_ho_n(\mathrm{SG}) \times e\_t_n(\mathrm{SG})} = \frac{1}{(ho_n(\mathrm{SG})/n) \times (t_n(\mathrm{SG})/n \times 2^n)} = \frac{n^2 \times 2^n}{(ho_n(\mathrm{SG})) \times (t_n(\mathrm{SG}))}$$

Since it is desirable that both ho_n(SG) and t_n(SG) be as low as possible, the higher the value of E_n(SG), the more effective is SG. In Fig. 7, E_n(SG) is presented for each one of the techniques for various values of the inputs of the CUT. From Fig. 7 it is derived that the proposed implementation is the most effective of the techniques that have been presented in the literature for the generation of SIC pairs with respect to the hardware overhead and the time required to complete the test.

5. Conclusions

A novel implementation of a SIC two-pattern generator based on Ling adders has been proposed. Ling adders are typically used in current systems, since they have been proved to be faster than competitive architectures. The proposed scheme generates the SIC pairs within (n+1/2) × 2^n cycles without altering the timing characteristics of the adder. Comparisons with the techniques that have been proposed in the literature for the generation of Sic pairs revealed that the proposed implementation is more effective in terms of hardware overhead and time required to generate the Sic pairs when accumulators and barrel shifters are available.

References

[1] M. Abramovici, M. Breuer, A. Freidman, Digital Systems Testing and Testable Design, Computer Science Press, 1990.
[2] R.L. Wadsack, Fault Modeling and logic simulation of CMOS and MOS integrated circuits, Bell System Technical Journal 57 (5) (1987) 1449–1474 (May–June).
[3] M.H. Woods, MOS VLSI reliability and yield trends, Proceedings of the IEEE 74 (12) (1986) 1715–1729 (December).
[4] G.L. Smith, Model for delay faults based upon paths, Proceedings of the IEEE International Test Conference (1985) 309–314.
[5] W. Wang, S. Gupta, Weighted random robust path delay testing of synthesized multilevel circuits, Proceedings of the 12th IEEE VLSI Test Symposium (1994) 291–297.
[6] G. Craig, C. Kime, Pseudo-exhaustive adjacency testing: a bist approach for stuck-open faults, Proceedings of the IEEE International Test Conference (1985) 126–137.
[7] P. Girard, C. Landrault, V. Moreda, S. Pravossoudovitch, An Optimized BIST Test Pattern Generator for Delay Testing, In: Proceedings of the 15th IEEE VLSI Test Symposium. 1997, pp. 94–100.
[8] I. Voyiatzis, A. Paschalis, D. Nikolos, C. Halatsis, An efficient built-in self test method for robust path delay fault testing, Journal of Electronic Testing: Theory and Applications (1996) 219–222 (June).
[9] P. Girard, C. Landrault, S. Pravossoudovitch, A. Virazel, Comparison between random and pseudorandom generation for BIST of delay, stuck-at and bridging faults, Proceedings of the 6th IEEE On-Line Testing Workshop (2000) 121–126.
[10] R. David, P. Girard, C. Landrault, S. Pravossoudovitch, A. Virazel, On using efficient test sequences for BIST, Proceedings of the 20th VLSI Test Symposium (2002) 145–150.
[11] A. Virazel, R. David, P. Girard, C. Landrault, S. Pravossoudovitch, Delay fault testing: choosing between random SIC and random MIC sequences, Proceedings of the IEEE European Test Workshop (2000) 9–14.
[12] H. Rahaman, D. Das, B. Bhattacharya, Transition count based BIST for detecting multiple stuck-open faults in CMOS circuits, Proceedings of the 2nd IEEE Asia Pacific Conference on ASICs (2000) 307–310.
[13] S. Crepaux-Motte, M. Jacomino, R. David, An algebraic method for delay testing, Proceedings of the 14th VLSI Test Symposium (1996) 308–315.

[14] M. Gharaybeh, M. Bushnell, V. Agrawal, Parallel concurrent path delay fault simulation using single-input change patterns, Proceedings of the 9th IEEE International Conference on VLSI Design (1996) 426–431.
[15] M. Gharaybeh, M. Bushnell, V. Agrawal, A parallel-vector concurrent fault simulator and generation of single-input-change tests for path-delay faults, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 17 (9) (1998) 673–676 (September).
[16] D. Das, I. Chaudhuri, B. Bhattacharya, Design of an optimal test pattern generator for built-in self testing of path delay faults, Proceedings of the 12th International Conference on VLSI Design (1998) 205–210.
[17] S. Lu, M. Lu, Testing iterative logic arrays for delay faults with a constant number of patterns, Proceedings of the 4th International Symposium on Electronic Materials and Packaging (2002) 492–498.
[18] F. Brglez, D. Bryan, K. Kozminski, Combinational profiles of sequential benchmark circuits, Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS) (1989) 1229–1234.
[19] E. Gizdarski, Detection of delay faults in memory address decoders, Journal of Electronic Testing: Theory and Applications 16 (4) (2000) 381–387 (August).
[20] E. Blokken, H. De Keulenaer, F. Catthoor, H.J. De Man, A flexible module library for custom DSP applications in a multiprocessor environment, IEEE Journal of Solid-State Circuits 25 (3) (1990) 720–729 (June).
[21] R.J. Higgins, Digital Signal Processing in VLSI, Prentice Hall, Englewood Cliffs, NJ, 1990.
[22] J. Rajski, J. Tyszer, Accumulator-based compaction of test responses, IEEE Transactions on Computers 42 (6) (1993) 643–650 (June).
[23] J. Rajski, J. Tyszer, Test responses compaction in accumulators with rotate carry adders, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 12 (4) (1993) 531–539 (April).
[24] A.P. Stroele, Test response compaction using arithmetic functions, Proc. of the 14th VLSI Test Symposium (1996) 380–386.
[25] J. Raskси, J. Tyszer, Arithmetic Built-In Self Test for Embedded Systems, Prentice Hall PTR, Upper Saddle River, New Jersey, 1998.
[26] S. Gupta, J. Rajski, J. Tyszer, Arithmetic additive generators of pseudo-exhaustive test patterns, IEEE Transactions on Computers 45 (8) (1996) 939–949 (August).
[27] A.P. Stroele, BIST pattern generators using addition and subtraction operations, Journal of Electronic Testing:Theory and Applications 11 (1) (1997) 69–80 (August).
[28] K. Radecka, J. Rajski, J. Tyszer, Arithmetic built-in self test for DSP cores, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 16 (11) (1997) 1358–1369 (November).
[29] I. Voyiatzis, A. Paschalis, D. Nikolos, C. Halatsis, Accumulator-based BIST approach for stuck-open and delay testing, Proceedings of the European Design and Test Conference (1995) 431–435.
[30] I. Voyiatzis, N. Kranitis, D. Gizopoulos, A. Paschalis, C. Halatsis, An accumulator-based built-in self-test generator for robustly detectable sequential fault testing, IEE Proceedings, Computers and Digital Techniques 151 (6) (2004) 466–472 (November).
[31] I Voyiatzis, D. Gizopoulos, A Paschalis, Accumulator-based test generation for robust sequential fault testing in DSP cores in near-optimal time, IEEE Transactions on VLSI Systems 13 (9) (2005) 1079–1086 (September).
[32] I. Koren, Computer Arithmetic Algorithms, A.K. Peters Ltd., 2002.
[33] B. Parhami, Computer Arithmetic—Algorithms and Hardware Designs, Oxford University. Press, 2000.
[34] M. Ergecovac, T. Lang, Digital Arithmetic, Morgan-Kauffman, 2003.
[35] H. Ling, High-speed binary adder, IBM Journal of Research and Development 25 (3) (1981) 156–166 (May).
[36] C. Efstathiou, H.T. Vergos, D. Nikolos, Ling adders in standard CMOS technologies, Proceedings of the IEEE International Conference. Electronics, Circuits, and Systems (ICECS) (2002) 485–488 (September).
[37] G. Dimitrakopoulos, D. Nikolos, High-speed parallel-prefix VLSI ling adders, IEEE Transactions on Computers 54 (2) (2005) 225–231 (February).
[38] S. Vassiliadis, Recursive equations for hardwired binary adders, International Journal of Electronics 67 (2) (1989) 201–213 (August).
[39] R.E. Ladner, M.J. Fisher, Parallel prefix computation, Journal of ACM 27 (4) (1980) 831–838 (October).
[40] N. Weste, K. Eshraghian, Principles of CMOS VLSI Design: A Systems Perspective, Addison Wesley Company, 1985.