Journal of Electronic Testing (2023) 39:245–262
https://doi.org/10.1007/s10836-023-06055-w

A Tunable Concurrent BIST Design Based on Reconfigurable LFSR

Ahmad Menbari[1] · Hadi Jahanirad[1]

Received: 14 October 2022 / Accepted: 21 February 2023 / Published online: 6 March 2023
© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2023

Abstract

In concurrent online BIST, testing is conducted simultaneously during normal functional operation. A fault model enables a structural test to be undertaken for a long time while simultaneously identifying critical faults. As a result of continuous testing, intermittent and transient faults are more likely to be detected. The number of required cycles for completion of a concurrent test, known as concurrent test latency (CTL), is a critical parameter for a concurrent BIST design. Most of the existing methods have impractical CTL, while others suffer from a high hardware overhead or a presence of a substantial combinational circuit. These methods are also incompatible with situations where parameters need to be *adjusted*, like when the hardware is more critical than CTL and vice versa. This paper proposes an efficient concurrent BIST to overcome the mentioned challenges. The main components of the proposed design consist of LFSRs and a small decoding combinational module result in low hardware overhead. In addition, CTL and hardware overhead can be adjusted and tuned in an acceptable range using the proposed method. Compared to the most efficient method, the proposed method achieves a 10% reduction in hardware overhead for large-scale circuits by keeping the CTL minimum. The different experiments demonstrate the capability of tuning between CTL and hardware overhead for the proposed BIST design. In the case that CTL and hardware overhead are equally important, the proposed method significantly lowers CTL compared to previous methods, while hardware overhead is only about 4% higher than previous method for both large scale (LS) and very large scale (VLS) circuits.

Keywords Built-in self-test · Concurrent self-test · Concurrent test latency · Linear feedback shift register

1 Introduction

Offline BIST techniques can only be applied when a device under test is sufficiently idle [2, 14]. Depending on the real-time constraints, this may be difficult or impossible to achieve in some systems. Furthermore, these approaches do not address dormant faults and further fault accumulations, which reduce system reliability, as well as temporary faults that frequently occur in modern VLSI chips [3, 6, 10, 11, 15].

Concurrent BIST design, on the other hand, refers to testing a circuit in its normal mode of operation. This continuous testing increases the possibility of detecting intermittent and transient faults [9, 17, 19]. However, the number of required cycles for completion of a concurrent test, known as concurrent test latency (CTL), is a significant obstacle for concurrent BIST widespread application [8, 9].

Duplication design is the most straightforward online BIST architecture wherein the input vector is applied to both the CUT and its copy. For every applied input vector, a comparator checks if the output vectors of two circuits are equal and the error signal is generated by the disjunction of all the output bits. The hardware overhead for this method is over 100% due to the copy of CUT and the comparator [7, 9].

Almost all of the concurrent BIST designs have been proposed based on the pre-computed test set [7]. These pre-computed test vectors are selected deterministically by a test pattern generation (TPG) algorithm. These selected test vectors are identified using an input pattern detector, and an output response analyzer (ORA) decides whether the test passes or fails. Several modified methods [14, 16, 17, 19, 21] that recommend different strategies (e.g., input vector

Responsible Editor: K. K. Saluja

Hadi Jahanirad
h.jahanirad@uok.ac.ir

Ahmad Menbari
a.menbari@uok.ac.ir

[1] Department of Electronics and Communication Engineering, University of Kurdistan, Sanandaj, Iran

monitoring) to reduce hardware overhead are explained in the literature review section.

The authors of [7] proposed a method with reasonable CTL and acceptable hardware overhead. By using this method, which is called DC-based in this paper, all of the required modules to perform testing (e.g., the pattern detector and ORA) are synthesized into one logic module. This approach is based on the idea that faults are detected by test vectors with small numbers of specified bits. Putting the un-specified bits as *don't care*, the likelihood of occurring a test vector increases, and the CTL is reduced compared with previous methods.

An efficient method in terms of CTL and hardware overhead has been proposed recently [9]. The general schematic of this method is illustrated in Fig. 1. In the initial step, a *detector* is employed to identify an incoming input vector that belongs to a pre-computed test set. Then, a *mapping module* maps every detected input vector to its corresponding error-free compressed output vector, which should be compared with the compressed output vector in the final step to detect the fault occurrence. As shown in Fig. 1, $LFSR_{comp}$ is used to compress output vectors and error-free output vectors during the test and in a pre-design step, respectively.

There are two different concurrent BIST designs proposed in this method. In the hardware overhead aware (HW-aware) approach, the *detector* selects the input vectors that generate a 0 remainder when divided by an LFSR. This LFSR's architecture is designed so that the selected test vectors are most similar to a set of deterministic test patterns generated by a deterministic TPG algorithm. A mapping module consists of an LFSR to compress detected input vectors and a synthesized combinational circuit to map the compressed vectors into compressed output vectors. Even though this technique reduces hardware overhead, its CTL is impractical in most cases.

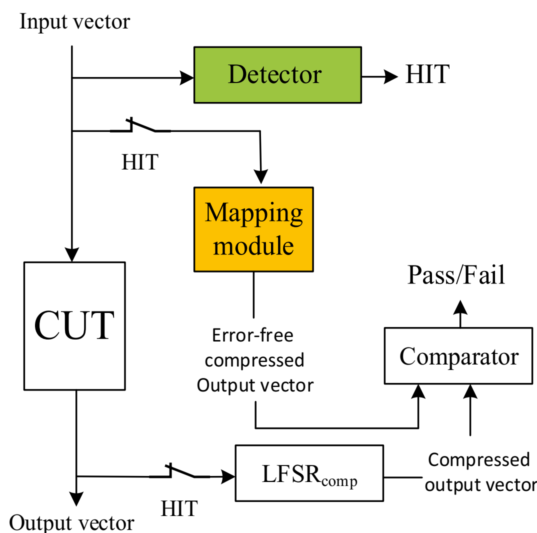


Fig. 1 General schematic of [9]

In the CTL-aware design, the detector consists of two LFSRs with different primary characteristic polynomials. All input vectors that are divisible by both of them are identified as test vectors. By employing this design, the desired fault coverage can be achieved and CTL can be significantly reduced. However, the mapping module is much larger than the HW-aware. Moreover, this approach also includes an approximation to place many test vectors into a test cube, which is unreasonable in some cases. Finally, CTL-aware and HW-aware designs cannot compromise between CTL and area overhead metrics. The CTL-aware design is just concerned with the maximum reduction of CTL, and the HW-aware design only tries to minimize the hardware overhead.

In this paper, a concurrent BIST architecture with low CTL and low hardware overhead is proposed. In this method, a different viewpoint is considered to design the *mapping module*. As a result, the abovementioned problems of CTL-aware design are tackled and its low hardware overhead version is proposed. Moreover, hardware overhead and CTL can be adjusted according to the demand using the proposed method.

The overall schematic of our proposed design has been illustrated in Fig. 2. The main modification compared to the CTL-aware design is the replacement of the Mapping Module with a bank of LFSRs. When a test vector enters the circuit, the compressed version of the CUT output (O'(x)) should match the quotient of related LFSR in the LFSR

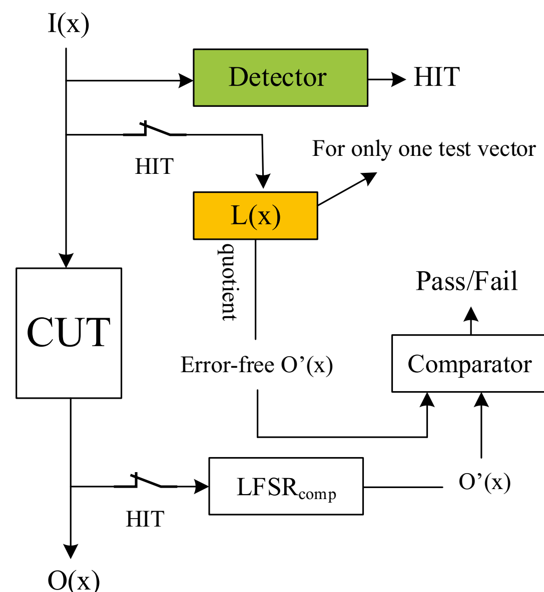


Fig. 2 General schematic of the proposed method: Mapping module for one test vector

bank. Consequently, there should be an LFSR for every test vector in the LFSR bank, which imposes a high hardware overhead on the design. To solve this problem, a configurable LFSR is used (an LFSR whose polynomial is defined arbitrarily by a polynomial ID), wherein we proposed a cost-efficient method to extract the Polynomial IDs from incoming test vectors (Fig. 3).

The main contributions of this paper are highlighted as follows:

1- A concurrent BIST design with low hardware overhead and low CTL is proposed.
2- Almost all modules required for concurrent BIST are implemented using LFSRs which improve the re-usability and reduce the design complexity.
3- The desired balance between hardware overhead and CTL could be made by regular and easy parameters tuning.

2 Literature Review

The first step in emerging a concurrent BIST design is the extraction of a suitable test set. In circuits with few input pins, all input vectors are included in the test set (exhaustive test). Test vectors can be randomly selected or deterministically generated using a test generation algorithm [20]. Typically, these methods use a pre-determined test set, which is selected so that a desired fault coverage can be achieved [12, 13, 18].

CBIST [14] is one of the early concurrent BIST designs. A test vector is selected from the test set as the active test vector, and the selecting part waits until it occurs in the circuit's primary inputs. As soon as the output vector is compared with the fault-free output vector, the fail/pass state is determined. Then, the next active test vector is selected from the pre-computed test set. The above process is repeated for all test vectors to complete the test. Due to the unpredictable wait time in matching the active test vector with the input vector of the circuit, CBIST has high concurrent test latency (CTL), defines a the time elapsed for the occurrence of all test vectors in the primary inputs). To decrease the high CTL of CBIST, the authors of [12, 13, 18], and [20] increase the number of active test vectors to increase the hit probability. All of the above-mentioned techniques generate a large test set based on pseudorandom test pattern generation algorithms, making them impractical for large CUTs in terms of CTL.

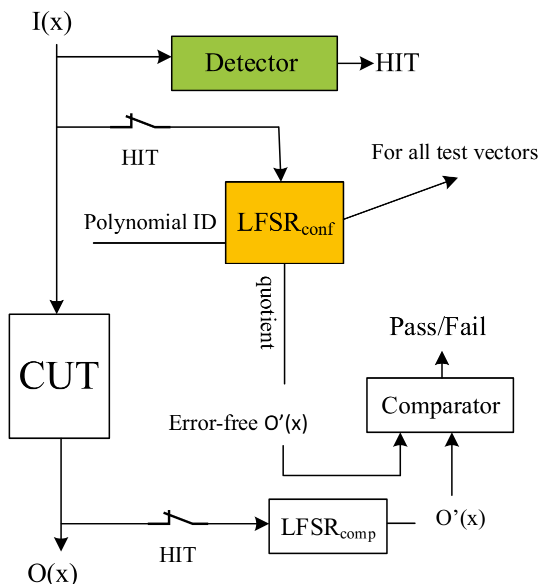


Fig. 3 General schematic of the proposed method: Mapping module for multiple test vectors

The MHSAT approach [5] utilizes L active test vectors which are generated by L LFSRs. Furthermore, a Multiple Input Shift Register (MISR)-based response verifier is activated as the related LFSR moves on to the next state. When all LFSRs sweep their states, the test would be completed. OISAT [1] uses L LFSRs for active test vector generation, while an accumulator-cased compaction (ABC) is utilized for response verification.

In w-CBIST [18], all possible input vectors are divided into windows containing t vectors. In each step, an active window is chosen, and the input vectors are matched up with test vectors in the active window. The test is complete when all windows are processed.

The test vectors of several concurrent BIST designs are generated using deterministic TPG algorithms [1, 5, 7, 17–19]. The CTL reduces in these methods due to the small number of test vectors. BICST [16] consists of a concurrent test circuit (CTC) and a programmable logic array as the pattern detector and mapping module, respectively. The CTC and the CUT have equal input and output pins. When a test vector appears on the CUT's input port, the comparator (output verifier) compares the CUT and CTC outputs to determine whether the test passes or fails.

MICSET [19] improves the BICST design by adding an offline test mechanism and modifying the pattern detector. Consider a test set that contains Tn-bit test vectors. The test matrix is generated in the first step, and a greedy algorithm is used to select t ($\log_2 T \times t \times T$) columns such that all the Tt-tuples are distinct. The remaining $(n - t)$ bits of the test vector are directed by these t columns. In the offline mode, a set of 2×1 multiplexers are used to connect the generated test vectors to the CUT inputs using a t-stage counter and $(n - t)$-bits from an OR plane.

BICST and MICSET both have a high hardware overhead, as well as impractical CTL due to the low probability of pre-computed test vectors occurring during every clock cycle [19]. Among input vector monitoring

approaches, [17] is the most efficient one, due to the presence of SRAM cells used to monitor test vectors.

NEMO [16] and later a cost-efficient NEMO [21] were proposed to remedy the area overhead problem. Figure 4 shows the schematic of NEMO, where T pre-computed test vectors ($T << 2n$) are detected by a decoding module (D). The multi-level decoder is designed based on 2-input NAND gates and generates 1 when one of the test vectors occurs in the input of the CUT. To reduce hardware overhead, the CALC module produces compacted versions of the related CUT outputs. On the output side, the CUT output vector is compressed to $m-q$ bits using a space compactor (SC). The compacted versions of output vectors generated by CALC and SC are compared to complete the test. As long as one of the T detector outputs equal 1, the OR module's output would be 1. This means a test vector appears in the incoming input vectors.

Implementation of NEMO's space compactor (SC) requires knowledge of the details of the CUT [4]. To get rid of this requirement, CE-NEMO deploys an alternative decoder in the output of CUT instead of SC. Furthermore, the following three modifications are made to the input side decoder to reduce hardware overhead:

1. Reducing the columns of the pre-computed test set is accomplished using a more efficient method.

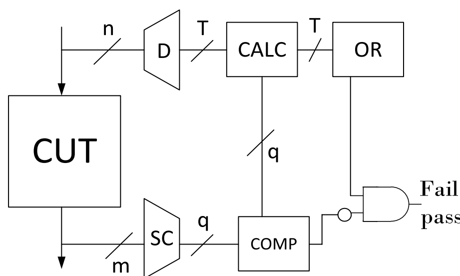


Fig. 4 Schematic of NEMO

2. Self-pairing of unpaired columns is eliminated when the number of distinct columns does not reach a power of 2, and these columns move directly to the next decoding stage.
3. A meta-heuristic optimization process based on simulated annealing (SA) is used to generate the most efficient order of distinct columns

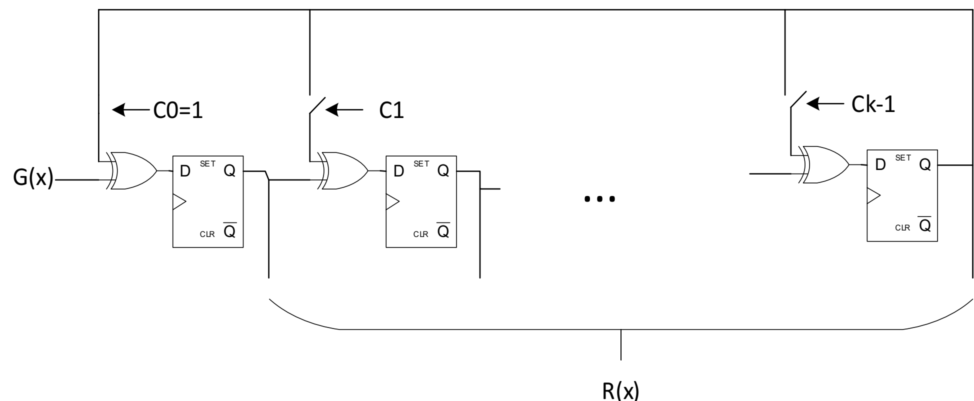
The major problems of NEMO and CE-NEMO could be summarized as follows. The hardware complexity of decoder modules in both NEMO and CE-NEMO depends on the number of pre-computed test vectors which mainly is related to the increment of the number of NAND gates by increasing the size of the test set. Moreover, the bit-width of input vectors affects both hardware overhead and the design complexity of decoders. Moreover, using the SA-based optimization approach to find the most efficient order of inputs increases the difficulty of the decoder implementation in CE-NEMO.

3 Preliminaries

3.1 Division by LFSR

A type-2 LFSR is used as the core of the proposed concurrent BIST design. Figure 5 illustrates the structure of a Type-2 LFSR based on k DFFs arranged in a right-shift register format. Inputs of DFFs are connected to XOR gates to establish feedback loops, and XOR gates are inserted at predetermined locations depending on the LFSR characteristic polynomial [2]. The characteristic polynomial is defined according to (2). According to the first term, an XOR gate will always be available in the rightmost DFF output. The second term shows a feedback loop always begins from the leftmost DFF's output and ends on the rightmost DFF's input. For any other C_i coefficient, 1 will be set if the related XOR gate exists; otherwise, it would be 0.

Fig. 5 Type-2 LFSR architecture



$$L(x) = 1 + C_{k-1}X^{k-1} + \sum_{i=1}^{k-2} C_i X^{k-i} \quad (1)$$

Assume that an N-bit binary number passes serially through an LFSR with k stages (Flip Flops). Accordingly, the final content of the LFSR's DFFs would be the remainder of dividing $G(x)$ by $L(x)$ if $G(x)$ is the polynomial associated with the N-bit binary number and $L(x)$ is the characteristic polynomial of the LFSR. The relationship between $G(x)$, the LFSR characteristic polynomial ($L(x)$), and the remainder ($R(x)$) can be expressed as (2).

$$G(x) = L(x)Q(x) + R(x) \quad (2)$$

In dividing 2^N polynomials by an LFSR ($L(x)$) of order k, the remainders are $R(x)=\{0, 1, X, 1+X, X^2, \ldots, 1+X+X^2+\ldots+X^{k-1}\}$. For instance, the remainders of $\{L(x), xL(x), (x+1), \ldots\}$ and $\{L(x)+1, xL(x)+1, (x+1)L(x)+1, \ldots\}$ divided by $L(x)$ become 0 and 1, respectively.

3.2 Concurrent Test Latency (CTL)

Concurrent test latency (CTL) refers to the time elapsed for the occurrence of all test vectors in the primary inputs. There are three assumptions to compute the CTL [14]:

1- In every clock cycle, an input vector is applied to the CUT.
2- In each clock cycle, all input vectors are equally likely to occur.
3- The probability of occurrence of any input vector is independent of the probability of occurrence of the other input vectors.

Let T be the number of test vectors required to cover the acceptable number of faults. The probability of occurrence of i numbers of test vectors in a clock cycle is defined as the *hit* of i test vectors, $h(i) = i/2^n$(n is the number of CUT's inputs). The number of clock cycles required to occur i numbers of test vectors is $2^n/i$. Therefore, the CTL is given by

$$CTL = \sum\nolimits_1^T \frac{1}{h(i)} = 2^n \sum_1^T \frac{1}{i} \quad (3)$$

For circuits with fewer than 40 inputs, CTL is acceptable as demonstrated by simulation in [7]. Otherwise, the CTL would be impractical.

The only way to reduce CTL is to increase the probability of occurring a test vector in each clock cycle. DC-based and CTL-aware methods decrease CTL significantly based on this idea. In DC-based design, the occurrence probability of test vectors increases based on the fact that faults are detected by a small number of specified bits of the test vector, and the other bits would be *don't care*.

In the CTL-aware design [9], two LFSRs are used to detect test vectors, in which the input vectors with the same remainders in dividing by these LFSRs are considered as test vectors. Therefore, selected test vectors are divided into several groups containing a large number of test vectors with a small number of bits each. For instance, $L(x)$ and $L(x)+1$ are detected by LFSRs and both of them cover similar faults. Therefore, there is only one test vector required in a group to cover several specific faults. Consequently, A CTL-aware design has a CTL 2^k times smaller than Eq. 3 if the number of test vectors is 2^k. Finally, as proved in [9], the CTL can be obtained by modifying Eq. 3 as follows

$$CTL = 2^{(n+a+1)/2} \times \sum\nolimits_1^m \frac{1}{i}, 2 < a < 12 \quad (4)$$

3.3 Detector

Concurrent BIST uses a module called pattern detector to detect pre-computed test vectors from incoming input vectors. As mentioned earlier, the detector in the CTL-aware approach [9] is used for the proposed method. In this section, an overview of the detector used in [9] is presented; however, the reader can refer to [9] for more details.

In the CTL-aware design [9], two LFSRs ($L_1(x)$ and $L_2(x)$) with unequal primary polynomials are used. The input vectors that generate the same remainders in dividing by $L_1(x)$ and $L_2(x)$ are considered as a test vector. The CTL-aware design can be implemented by more than two LFSRs. Note that utilization of more than two LFSRs imposes more restrictions on the test vectors. For example, the input vectors which generate an equal remainder in the division by L1(x), L2(x), and L3(x) would belong to the test set. The probability of occurrence of such test vectors in the circuit's input would be much lower than the two LFSRs case. Consequently, the most expanded test set (with the lowest CTL) is achieved when we apply two LFSRs for selecting part.

All polynomials with the following formation (Eq. 5) are selected as test vectors in CTL-aware design. We illustrated all polynomials in Table 1 so that every row could be selected as test vectors.

$$G_i(x) = m_i(x)L_1(x)L_2(x) + R(x) \quad (5)$$

Assume that only the test vectors (t(x)) in the second row ($R(x) = 0$) are considered as test vectors. The authors of [9] proposed an algorithm to efficiently find $L_1(x)L_2(x)$ to cover the desired number of faults.

However, in the CTL-aware design, all input vectors of Table 1 are detected as test vectors to reduce CTL. This happens because, all test vectors in a same column

Table 1 Arrangement of test vectors polynomials in the division by $L_1(x)$ $L_2(x)$. t(x) is the number of test vectors selected by the algorithm proposed in HW–aware [9]

$m_1(x)=1$	$m_2(x)=x$		$m_t(x)=t(x)$	R(x)
$G(x)=L_1(x)L_2(x)$	$G(x)=x\ L_1(x)L_2(x)$	…	$G(x)=t(x)\ L_1(x)L_2(x)$	0
$G(x)=L_1(x)L_2(x)+1$	$G(x)=x\ L_1(x)L_2(x)+1$	…	$G(x)=t(x)\ L_1(x)L_2(x)+1$	1
$G(x)=L_1(x)L_2(x)+x$	$G(x)=x\ L_1(x)L_2(x)+x$	…	$G(x)=t(x)\ L_1(x)L_2(x)+x$	x
$G(x)=L_1(x)L_2(x)+x+1$	$G(x)=x\ L_1(x)L_2(x)+x+1$	…	$G(x)=t(x)\ L_1(x)L_2(x)+x+1$	x+1
•	•	•	•	•
•	•	•	•	•
•	•	•	•	•
$G(x)=L_1(x)L_2(x)+x^{k-1}+\ldots+1$	$G(x)=x\ L_1(x)L_2(x)+x^{k-1}+\ldots+1$	…	$G(x)=t(x)\ L_1(x)L_2(x)+x^{k-1}+\ldots+1$	$x^{k-1}+\ldots+1$

approximately cover the same faults. For example, $L_1(x)L_2(x)$ and $L_1(x)L_2(x)$+1 have only one different bit. However, in some cases there are lot of different bits and it may decrease the fault coverage.

3.4 Proposed Method

The schematic of the proposed concurrent BIST design is illustrated in Fig. 6. As discussed earlier, the detector includes two LFSRs and a comparator. Whenever an input vector applies to the CUT, the remainders generated by the LFSRs are compared, and if they are equal, the input vector is considered a test vector, which is then passed to the next module via two switches.

When a test vector is detected, the output vector ($O(x)$) is divided by $LFSR_{comp}(x)$ to generate the compressed output vector ($O'(x)$) as a remainder. On the input side, the selected test vector passes through the $LFSR_{conf}$ to generate error-free $O'(x)$. When the selected input vector enters the $LFSR_{conf}$, the $LFSR_{conf}$ is configured so that its quotient (output) equals

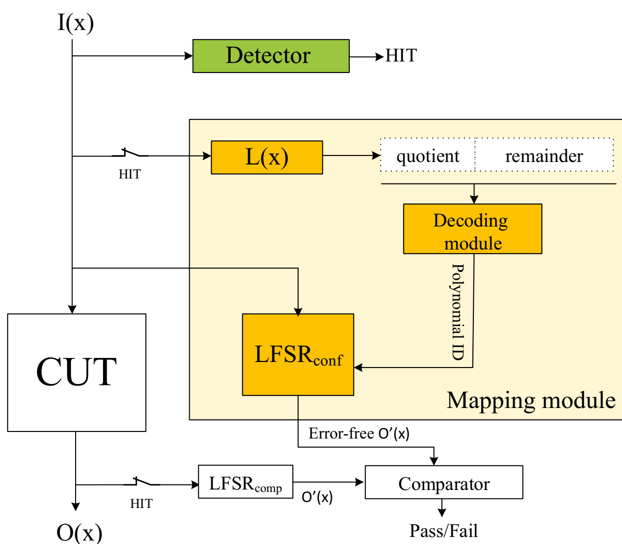


Fig. 6 Schematic of the proposed method

O'(x) when divided by the test vector. Two steps are required to generate a polynomial (polynomial ID) to configure $LFSR_{conf}$. First, the remainder and quotient of another LFSR ($L(x)=L_1(x)L_2(x)$) in dividing by test vectors are used as an identification code for test vectors. Then, it passes through a decoding module to generate the ID polynomials. Finally, the generated remainder by $LFSR_{conf}$ that is considered a fault-free compressed output vector is compared against the compressed output vector $O'(x)$ to determine whether the test passes or fails.

However, there are two challenges in this design. 1- for every test vector there must be an LFSR that generates the right $O'(x)$ for the detected test vector. It is essential to guarantee the existence of this LFSR.

2- polynomial IDs for configuring $LFSR_{conf}$ to generate the desired polynomial for different test vectors must be generated without using RAM. Therefore, each test vector must be assigned a unique code (binary number). Then this code passes through a decoding module to generate the desired polynomial ID. It is a challenge to extract a unique identification code from each test vector.

In the next Sections (4.1 and 4.2) these challenges and their solutions are discussed in details.

3.5 Challenge 1: Design a Mapping Module for a Test Vector

Notion Assume that an input vector is detected by the detector as the test vector ($T(x)$) and its corresponding compressed test vector in the output side is $O'(x)$. There exists an LFSR such as $L''(x)$, which, if $T(x)$ passes through it, produces an output (quotient) $O'(x)$.

Proof Assume that a test vector ($T(x)$) passes through an unknown LFSR ($L''(x)$), $L''(x)$ must be specified in which its output (quotient) becomes $O'(x)$. When $T(x)$ enters $L''(x), T(x)$ is defined using the following equation.

$$T(x) = L''(x).O'(x) + R(x) \tag{6}$$

The first condition to satisfy the equation is $T(x) > O'(x)$. T(x) and $O'(x)$ are known polynomials, while $L''(x)$ and $R(x)$ polynomials are unknown. We rewrite the equation as follows

$$T(x) + R(x) = L''(x).O'(x) \quad (7)$$

To satisfy the equation, $T(x) + R(x)$ must be divisible by $O'(x)$. Because $R(x)$ is an unknown polynomial that can take any number, $T(x) + R(x)$ can be specified in which it can divide $O'(x)$.

3.6 Challenge 2: Polynomial ID Generation for $LFSR_{conf}$

It is evident from the previous subsection that an LFSR is required to generate a fault-free compressed output vector. As mentioned, the number of selected test vectors are large. In the worst case, the number of LFSRs required for mapping is therefore equal to the number of test vectors, which is obviously inefficient in terms of hardware overhead. To solve this problem, a configurable LFSR is used as illustrated in the Fig. 7. Each test vector must be assigned an identification code (polynomial ID) in order to configure $LFSR_{conf}$ to the desired polynomial. However, the presence of a large RAM again imposes a high hardware overhead on the CUT.

Alternatively, incoming input vectors (test vectors) can also be used to specify their corresponding polynomials. The selected test vectors are arranged in Table 1. Different quotients are produced by test vectors in a same row when divided by$L(x)$. For example, the first-row test vectors generate this set of quotients: $\{0, 1, X, X + 1, ..., f(x)\}$. If the test vectors in the same column are divided by$L(x)$, they produce different remainders. For example, the remainders of the second column test vectors are $\{0, 1, X, X + 1, \ldots\}$. As a result of division by$L(x)$, unique remainders and quotients are produced for each selected test vector that are used as inputs to the *decoding module* to produce the desired polynomial ID. As soon as a test vector (e.g.,$xL(x) + 1$) is detected by the detector, its quotient (x) and the remainder (1) are applied to $LFSR_{conf}$ as an identification code (binary number) that passes through the decoding module to produce its polynomial ID.

When the remainder and quotient of test vectors are determined, a combinational logic circuit that maps the quotient-remainder input to the $O'(x)$ is synthesized to generate the decoding module. The equivalent-gate size of this module is small compared with CUT due the fact that the number of input decoder input vectors are significantly lower than the number of cut input vectors. Although a circuit with a small number of input vectors is not necessarily smaller than another circuit with a larger number of input vectors, in this case we can select a small set of test vectors to synthesize a circuit of a small size.

3.7 Tradeoff Between CTL and Hardware Overhead

An important feature of the proposed design is the ability to trade off CTL and hardware overhead on demand. Hardware overhead is imposed on the circuit primarily by LFSRs (detector, $L(x)$, $LFSR_{conf}$, and $L_{comp}(x)$) and the detecting module. The number of test vectors is not related to the hardware overhead of the LFSRs; however, it affects the size of the decoding module greatly, because the decoding module input vectors are determined by dividing selected test vectors by $L(x)$, as discussed in the previous section. In fact, an input vector for the decoding module consists of a bitstream including its quotient and remainder in dividing by $L(x)$. Therefore, the number of decoding module input pins is determined by summing the quotient ($f(x)$) and remainder ($x^{k-1} + x^k + \ldots + 1$) bit widths when dividing the largest test vector of Table 1 ($f(x)L(x) + x^{k-1} + \cdots + 1$) by $L(x)$.

In Table 1, the occurrence of all test vectors from a row (or at least one test vector from a column) is required for a certain amount of fault coverage, so the quotient bit width are constant. However, the presence of so many similar test vectors in a column is only required for the purpose of achieving acceptable CTL. In the case that all polynomials of Table 1 are considered as test vectors, the best CTL is achieved. Consequently, in this case, hardware overhead will have experienced its worst values as the decoding module has its largest size (longest bitstream for the remainder). However, if only the first row of test vectors in Table 1 are allowed to pass, the most efficient number for hardware overhead is achieved (shortest

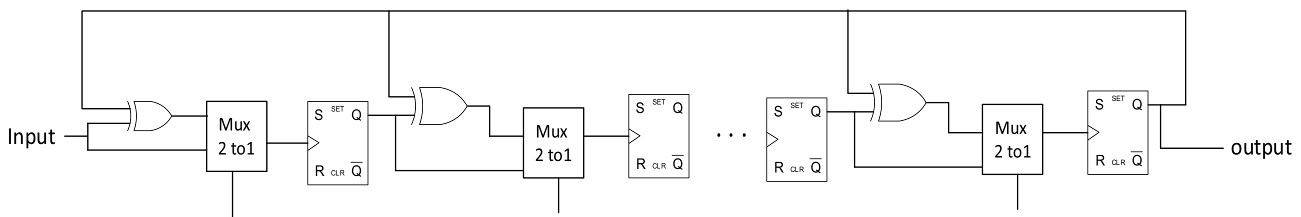


Fig. 7 Configurable LFSR used in this paper

Table 2 The overall results for the proposed method

Circuit	In	Out	Gates	Test patterns of proposed method			HW @ FC (%)			CTL
				80% FC	90% FC	95% FC	80%	90%	95%	
SS										
c432	36	7	106	42	50	57	250.32	269.45	276.98	8.3 E06
c499	41	32	202	42	53	60	150.43	165.65	171.03	4.7 E07
c880	60	36	383	54	62	67	131.54	140.90	145.67	3.43 E10
c1355	41	32	546	65	73	84	123.45	137.56	143.46	6.7 E07
c1908	33	25	880	88	98	109	65.76	73.78	80.45	4.1 E06
c2670	157	64	1193	82	90	101	144.34	160.23	165.89	5.3 E36
c3540	50	22	1669	120	134	150	51.20	55.34	64.79	1.5 E09
s298	3	6	133	–	–	–	–	–	–	–
s344	9	11	175	16	21	28	65.64	72.34	76.98	5.12 E02
s349	9	11	175	12	21	30	68.70	76.87	81.09	5.16 E02
s386	7	7	165	–	–	–	–	–	–	–
s400	3	6	183	–	–	–	–	–	–	–
s420	19	2	212	44	52	65	87.76	97.98	102.88	2.31 E04
s444	3	6	202	–	–	–	–	–	–	–
s510	19	7	217	43	51	57	93.23	101.10	103.47	2.36 E04
s641	35	24	398	45	50	59	98.11	106.56	115.90	1.67 E07
s715	35	23	512	45	53	61	82.56	90.80	93.90	2.37 E07
s1196	14	14	547	104	115	123	54.23	61.08	66.76	5.79 E03
s1238	14	14	547	106	115	122	60.23	64.78	67.77	5.83 E03
s1423	17	5	731	54	62	69	28.11	35.70	39.19	5.11 E04
s1488	8	19	659	–	–	–	–	–	–	–
s1494	8	19	653	–	–	–	–	–	–	–
Average							**97.22**	**106.87**	**112.26**	
LS										
c5315	178	123	2307	163	175	189	68.56	75.78	79.64	3.96 E28
c6288	32	32	2417	12	26	42	24.22	26.56	28.16	1.4 E06
c7552	207	207	3512	178	190	202	40.23	46.56	48.90	9.1 E32
s5378	35	35	2958	226	252	270	18.9	21.76	24.81	1.18 E07
s9234	36	36	5825	426	458	480	11.34	13.01	14.76	6.55 E04
s13207	62	62	8620	315	350	380	10.34	12.34	12.99	1.3 E11
s15850	77	77	10,306	152	394	414	13.34	14.12	15.23	3.5 E13
Average							**26.70**	**30.01**	**32.07**	
VLS										
s35932	35	320	17,793	88	97	110	9.61	10.89	12.09	12.09
s38417	28	106	23,815	973	1011	1032	13.24	15.34	16.92	16.92
s38584	38	304	20,705	713	768	790	12.56	13.98	14.50	14.50
b17	37	97	33,741	2286	2352	2398	11.23	13.01	14.55	14.55
b18	37	23	117,941	6106	6408	6676	7.14	8.34	9.14	9.14
b20	32	22	20,716	1245	1320	1397	12.34	13.98	14.71	14.71
b21	32	22	21,061	1263	1380	1443	13.12	14.34	15.09	15.09
b22	32	22	30,686	1786	1850	1954	9.43	10.13	11.14	11.14
Average							**11.08**	**12.50**	**13.51**	

bitstream); whereas, the CTL in this case has reported its most inefficient values. As a result, the desired CTL and hardware overhead would be obtained by limiting the number of passed test vectors (columns).

In order to specify the number of decoding module input pins, the number of passed test vectors (columns of Table 1) must be limited. As discussed, in Fig. 6, only the number of the remainder's bit width can be controlled, and the

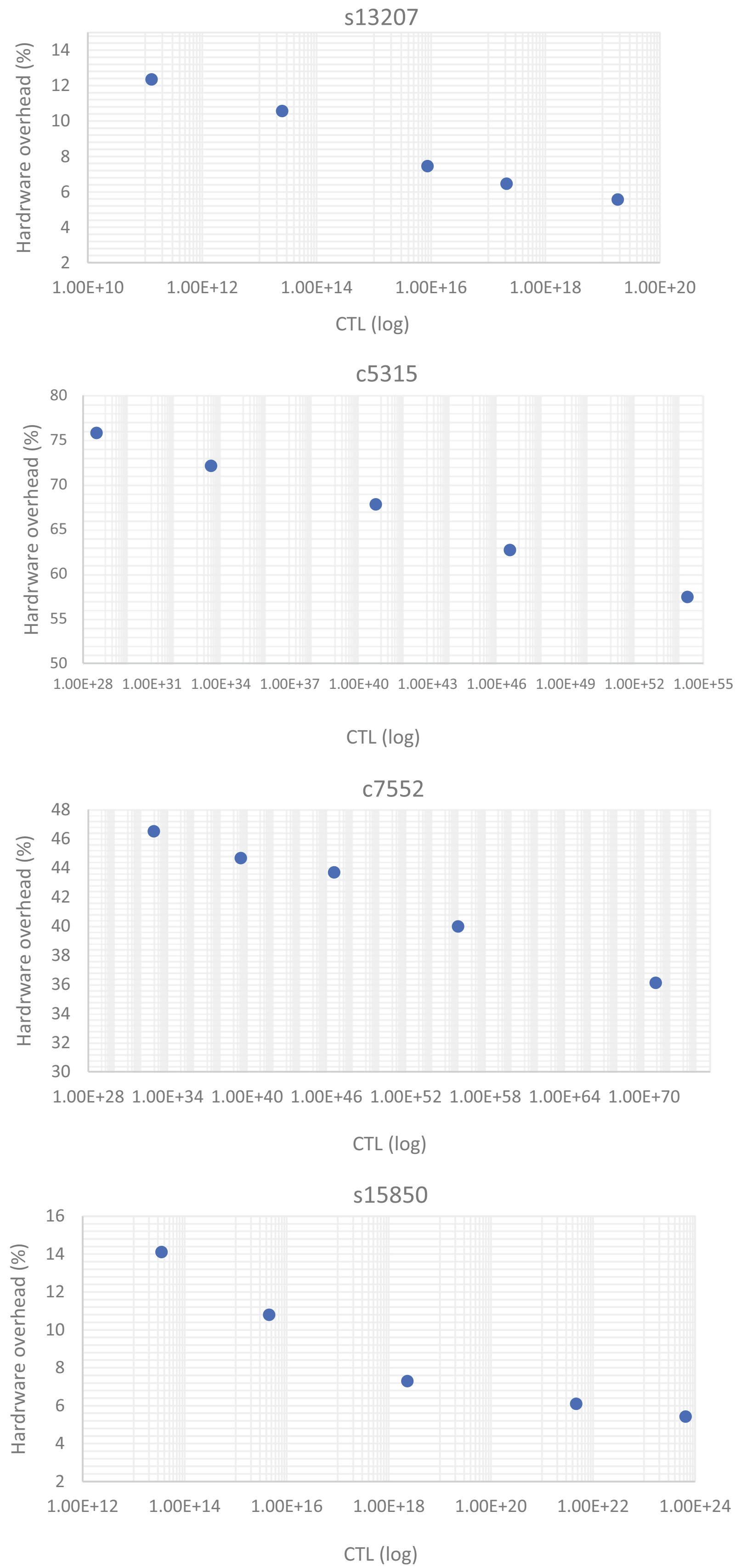


Fig. 8 Trade-off between CTL and hardware overhead for several benchmark circuits. **a**) S13207, **b**) c5315, **c**) c7553, and **d**) s15850

Table 3 CTL comparisons for different approaches

Circuit	Proposed method & CTL-aware [7]	DC-based [7]	CE-NEMO [21]	HW-aware [9]
c432	8.3 E06	3.96 E10	3.02 E11	3.08 E11
c499	4.7 E07	4.5 E11	9.94 E12	8.79 E12
c880	3.43 E10	4.93 E11	5.25 E18	4.61 E18
c1355	6.7 E07	4.59 E11	1.11 E13	1.15 E13
c1908	4.1 E06	3.62 E10	4.58 E10	4.62 E10
c2670	5.3 E36	5.72 E11	7.52 E70	7.56 E70
c3540	1.5 E09	1.58 E11	6.26 E15	6.27 E15
c5315	3.96 E28	3.94 E11	2.01 E54	2.06 E54
c6288	1.4 E06	3.11 E10	1.69 E10	1.71 E10
c7552	9.1 E32	6.18 E11	1.25 E63	1.28 E63
s298	–	9.34 E04	5.23 E 05	–
s344	5.12 E02	3.86 E07	–	5.91 E 07
s349	5.16 E02	3.86 E07	5.86 E 07	5.93 E 07
s386	–	2.79 E04	3.96 E 04	–
s400	–	8.98 E07	–	–
s420	2.31 E04	–	1.58 E11	1.61 E11
s444	–	7.73 E07	–	–
s510	2.36 E04	–	1.56 E 08	1.58 E 08
s641	1.67 E07	–	8.18 E 16	8.19 E 16
s715	2.37 E07	–	–	1.37 E 11
s1196	5.79 E03	5.98 E06	–	2.38 E 10
s1238	5.83 E03	7.72 E07	2.39 E 10	2.40 E 10
s1423	5.11 E04	5.26 E11	1.19 E 28	1.23 E 28
s1488	–	1.17 E05	8.78 E 04	–
s1494	–	1.10 E05	–	–
s5378	1.18 E07	6.38 E11	–	1.41 E11
s9234	6.55 E04	6.43 E11	–	2.76 E11
s13207	8.3 E06	6.68 E11	–	1.84 E19
s15850	3.5 E13	6.86 E11	–	6.05 E23
s35932	8.38 E06	–	–	1.43 E11
s38417	2.09 E06	–	–	1.12 E10
s38584	8.19 E03	–	–	2.80 E11
b17	9.49 E06	–	–	5.52 E11
b18	1.34 E08	–	–	5.58 E11
b20	1.18 E07	–	–	4.32 E09
b21	1.23 E07	–	–	4.35 E09
b22	1.27 E07	–	–	4.39 E09

quotient's bit width is fixed as all test vectors in a row must be applied to the CUT for achieving a certain fault coverage. Therefore, after dividing a test vector by *L*(*x*), if the number of remainder's bits is more than a desired number, the test vector would not consider as a test vector. Suppose that the desired number of the decoding module input pins is N_{dm_inp} as well as the fixed bit width of quotient is N_{qb}. If the longest possible remainder bitstream is $N_{rem\text{-}max}$ then the extra bits of remainder which should be checked would be $N_{check} = N_{rem\text{-}max}$—($N_{dm\text{-}inp}$-$N_{qb}$). Consequently, if the N_{check} rightmost bits of the generated remainder are zero then the input vector will be considered as a test vector. We utilize an N_{check}-input OR gate to check if the related bits of the remainder is zero. For instance, assume that the remainder

Table 4 Hardware overhead comparison with CTL-aware

Circuit	Proposed HW @ FC (%)		CTL-aware [9] HW @ FC (%)	
	80%	90%	80%	90%
SS				
c432	250.32	269.45	210.31	230.98
c499	150.43	165.65	139.23	147.85
c880	131.54	140.90	115.69	124.32
c1355	123.45	137.56	91.13	103.63
c1908	65.76	73.78	62.35	69.35
c2670	144.34	160.23	123.45	132.47
c3540	51.20	55.34	65.38	72.86
s298	–	–	–	–
s344	65.64	72.34	83.86	93.89
s349	68.70	76.87	84.25	95.89
s386	–	–	–	–
s400	–	–	–	–
s420	87.76	97.98	90.66	97.02
s444	–	–	–	–
s510	93.23	101.10	99.35	110.45
s641	98.11	106.56	83.65	88.72
s715	82.56	90.80	72.59	80.25
s1196	54.23	61.08	69.23	76.32
s1238	60.23	64.78	68.65	77.32
s1423	28.11	35.70	45.23	51.73
s1488	–	–	–	–
s1494	–	–	–	–
Average	**97.22**	**106.87**	**94.06**	**103.31**
LS				
c5315	68.56	75.78	71.63	75.03
c6288	24.22	26.56	28.36	33.32
c7552	40.23	46.56	43.54	45.2
s5378	18.9	21.76	30.36	32.9
s9234	11.34	13.01	15.36	19.26
s13207	10.34	12.34	21.36	24.2
s15850	13.34	14.12	27.56	29.72
Average	**26.70**	**30.01**	**33.88**	**37.09**
VLS				
s35932	9.61	10.89	18.69	21.89
s38417	13.24	15.34	21.3	25.6
s38584	12.56	13.98	22.56	26.52
b17	11.23	13.01	17.56	19.36
b18	7.14	8.34	12.56	14.23
b20	12.34	13.98	21.39	23.56
b21	13.12	14.34	20.96	24.68
b22	9.43	10.13	15.23	16.35
Average	**11.08**	**12.50**	**18.87**	**21.52**

of an input vector = $(x^5 + 1)\ L(x) + x^6$ in dividing by $L(x)$ is 1000000. If N_{check} equals 3, then 3 rightmost bits (100) are checked to decide that is test vector or not. So, the related checker would be 3-input OR gate.

Table 5 Hardware overhead comparison with NEMO and CE-NEMO

Circuit	Proposed method	NEMO [16]	CE-NEMO [21]
SS			
c432	276.98	513	555
c499	171.03	656	713
c880	145.67	931	1021
c1355	143.46	957	1026
c1908	80.45	947	1007
c2670	165.89	4846	5253
c3540	64.79	1700	1795
s298	–	186	206
s344	76.98	–	–
s349	81.09	175	199
s386	–	208	219
s400	–	–	–
s420	102.88	488	521
s444	–	–	–
s510	103.47	441	477
s641	115.90	788	874
s715	93.90	–	–
s1196	66.76	–	1011
s1238	67.77	953	1760
s1423	39.19	1606	337
s1488	–	313	–
s1494	–	–	–
Average	**112.36**	**1060.87**	**981.75**
LS			
c5315	79.64	5239	4927
c6288	28.16	365	325
c7552	48.90	7888	7463
s5378	24.81	–	–
s9234	14.76	–	–
s13207	12.99	–	–
s15850	15.23	–	–
Average	**32.07**	**4497.33**	**4238.33**

4 Simulation Results and Comparisons

Various simulation experiments are performed in this section to evaluate the proposed method. Several sets of benchmark circuits (ISCAS85, ISCAS89, and ITC99) are used to determine the hardware overhead and CTL as the main parameters of a concurrent BIST design. Hardware overhead is calculated by applying the gate equivalents metric, where a circuit module (a gate or DFF) is the hardware equivalent of some two-input NAND gates [9]. For instance, an N-input OR gate is hardware equivalent of N numbers of two-input NAND gates.

In the Section 5.2, four state-of-the-art methods including HW-aware [9], CTL-aware [9], NEMO [16], CE-NEMO

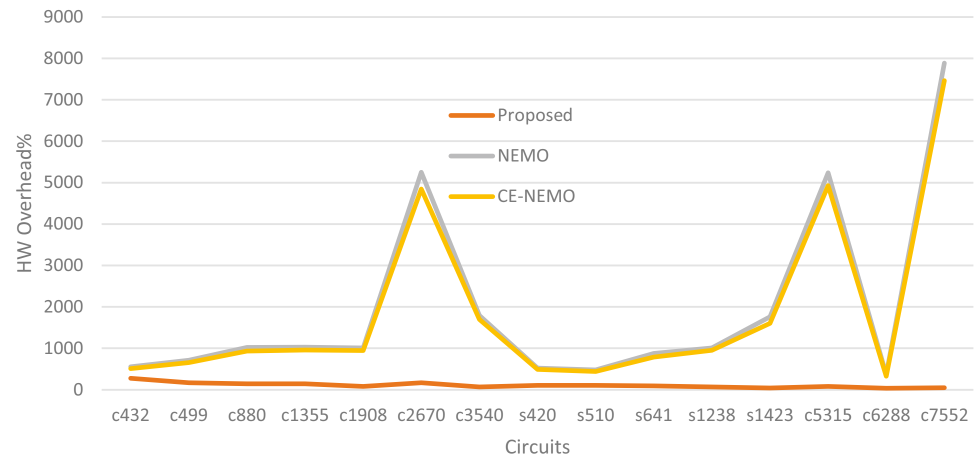


Fig. 9 Comparison of the hardware overhead of proposed design with NEMO and CE-NEMO

[21], and DC-based [15] are compared against the proposed method in terms of hardware overhead and CTL.

4.1 The Overall Results for Proposed Method

In order to evaluate the performance of the proposed BIST design, the benchmark circuits (ISCAS 85, ISCAS 89, and ITC 99) are categorized into three different types of circuits: small scale (SS), large scale (LS), and very large scale (VLS) based on the size of the circuit. The small size circuits (SS) contain less than 2000 gates, the large size circuits (LS) have less than 10,000 gates, and the very large size circuits include up to 117,000 gates.

In Table 2, hardware overhead and CTL of proposed design for different benchmark circuits are reported. The structural information for the benchmark circuits (the numbers of inputs, outputs, and internal modules) are reported in the second, third, and fourth columns of Table 2. The test vectors are derived using D-Alg algorithm according to the procedure of [9]. For test pattern generation and fault simulation, we used ATALANTA and ModelSim software tools on personal computer (PC) with 2.4 GHz core i5 CPU and, 4 GB of RAM, respectively.

ATALANTA and ModelSim software tools on personal computer (PC) with 2.4 GHz core i5 CPU and, 4 GB of RAM, respectively. The test set sizes are reported in columns 5 (for 80% fault coverage), 6 (for 90% fault coverage), and 7 (for 95% fault coverage), respectively. Obviously, the size of the test set increases with increasing the fault coverage. Hardware overhead and CTL are reported in the last columns. CTL is calculated for 95% fault coverage in all tables.

As mentioned in the previous sections, the detector proposed by the authors of [9] is used in our proposed design. Therefore, the same test vectors as CTL-aware are detected and the CTL is equal for the CTL-aware design and the proposed method. As calculated in Section 3.2, the CTL for the CTL-aware design (=proposed method) is acceptable for circuits with 80-a input pins. For the majority of circuits, in a 200 MHz clock circuit, the CTL would be less than 1 s. However, the CTL is very high for circuits with more than 80-a inputs e.g., c2670 and c5315.

Hardware overhead (HW) is reported for 80%, 90%, and 95% of fault coverage in Table 2. There are several important points with regard to hardware overhead. First, the hardware overhead increases by a slight increment of fault coverage due to the increment of the test set when the fault coverage increases. Second, for two circuits with an almost equal number of input pins, the impact of LFSRs on hardware overhead is lower for the larger circuit. Because the number of LFSRs required for testing both circuits is equal. For example, suppose that 5 LFSRs including N stages are required for CUT_1, and CUT_2 with gate equivalent sizes 1000 and 10,000, respectively. Then the LFSR-related hardware overhead for CUT_1 and CUT_2 would be 8N/1000 and 8N/10000, respectively. Therefore, the hardware overhead of LFSRs for CUT_1 is 10 times more than the LFSR-related hardware overhead for CUT_2. Third, the hardware overhead is significantly impacted by the number of primary inputs and test vectors. Because the number of LFSRs of the detector is directly related to the number of input pins. Moreover, the size of the detecting module significantly increases related to the number of test vectors as mentioned in the Section 4 in details. For example, C6288 has 32 primary inputs and 12 test vectors, which are significantly smaller than 178 primary inputs and 163 test vectors of C5315 (the number of test vectors is related to the HW-aware approach).

In Fig. 8, the graphs illustrate the hardware overhead, for 90% fault coverage, according to the changes of CTL for four benchmark circuits. Obviously, there is an indirect

Table 6 Hardware overhead comparison with DC-based and HW-aware

Circuit	Proposed (case1) HW @ FC (%)		HW-aware [9] HW @ FC (%)		DC-based [7] HW @ FC (%)	
	80%	90%	80%	90%	80%	90%
SS						
c432	250.32	269.45	178.26	199.32	70.3	89.1
c499	150.43	165.65	98.36	110.65	–	–
c880	131.54	140.90	86.56	95.56	55.5	76.6
c1355	123.45	137.56	68.59	82.65	–	–
c1908	65.76	73.78	39.45	47.35	25.7	42.2
c2670	144.34	160.23	95.68	110.65	55.7	–
c3540	51.20	55.34	36.97	42.92	18.3	26.1
s298	–	–	–	–	39.4	55.7
s344	65.64	72.34	52.32	58.2	41.6	48.1
s349	68.70	76.87	55.39	63.47	33.5	45
s386	–	–	–	–	52.3	71.5
s400	–	–	–	–	41.2	53.5
s420	87.76	97.98	69.75	79.99	–	–
s444	–	–	–	–	37	47.6
s510	93.23	101.10	73.96	81.69	–	–
s641	98.11	106.56	64.35	70.38	–	–
s715	82.56	90.80	52.69	59.56	–	–
s1196	54.23	61.08	43.93	48.69	45.2	60.7
s1238	60.23	64.78	48.56	53.96	47.1	62.7
s1423	28.11	35.70	16.45	22.68	47.3	58.5
s1488	–	–	–	–	28.8	38.6
s1494	–	–	–	–	30	38.5
Average	**97.22**	**106.87**	**67.57**	**76.73**	**41.80**	**54.29**
LS						
c5315	68.56	75.78	53.03	53.03	37.8	51.9
c6288	24.22	26.56	21.69	21.69	4.8	6.7
c7552	40.23	46.56	34.19	34.19	32.5	–
s5378	18.9	21.76	15.52	15.52	40.4	55.8
s9234	11.34	13.01	8.47	8.47	29.3	40.2
s13207	10.34	12.34	5.56	5.56	30	39.5
s15850	13.34	14.12	5.12	5.12	28.8	39.4
Average	**26.70**	**30.01**	**17.57**	**20.49**	**29.08**	**38.91**
VLS						
s35932	9.61	10.89	3.8	3.8	–	–
s38417	13.24	15.34	7	7	–	–
s38584	12.56	13.98	7.74	7.74	–	–
b17	11.23	13.01	5.9	5.9	–	–
b18	7.14	8.34	2.51	2.51	–	–
b20	12.34	13.98	7.23	7.23	–	–
b21	13.12	14.34	7.15	7.15	–	–
b22	9.43	10.13	5.55	5.55	–	–
Average	**11.08**	**12.50**	**5.35**	**5.86**		

For case1: minimum CTL, maximum hardware overhead

relation between CTL and hardware overhead for all circuits. There are two critical points (A and B) on the graphs. The point A shows the best recorded CTL (worst hardware overhead) for circuits where every test vectors in Table 1 are detected as test vectors. While, in the point B, the circuits have experienced the most efficient values for hardware

Table 7 Hardware overhead comparison with DC-based and HW-aware

Circuit	Proposed (case2) HW @ FC (%)		HW-aware [9] HW @ FC (%)		DC-based [7] HW @ FC (%)	
	80%	90%	80%	90%	80%	90%
SS						
c432	201.45	219.34	178.26	199.32	70.3	89.1
c499	109.34	123.12	98.36	110.65	–	–
c880	98.7	107.23	86.56	95.56	55.5	76.6
c1355	81.24	95.23	68.59	82.65	–	–
c1908	48.11	55.44	39.45	47.35	25.7	42.2
c2670	112.56	126.66	95.68	110.65	55.7	–
c3540	43.98	47.57	36.97	42.92	18.3	26.1
s298	–	–	–	–	39.4	55.7
s344	55.89	61.22	52.32	58.2	41.6	48.1
s349	63.33	70.09	55.39	63.47	33.5	45
s386	–	–	–	–	52.3	71.5
s400	–	–	–	–	41.2	53.5
s420	73.76	84.23	69.75	79.99	–	–
s444	–	–	–	–	37	47.6
s510	82.23	90.02	73.96	81.69	–	–
s641	77.8	83.25	64.35	70.38	–	–
s715	66.43	74.38	52.69	59.56	–	–
s1196	49.69	54.11	43.93	48.69	45.2	60.7
s1238	53.23	57.33	48.56	53.96	47.1	62.7
s1423	23.12	28.34	16.45	22.68	47.3	58.5
s1488	–	–	–	–	28.8	38.6
s1494	–	–	–	–	30	38.5
Average	**77.36**	**86.09**	**67.57**	**76.73**	**41.80**	**54.29**
LS						
c5315	49.32	56.89	53.03	53.03	37.8	51.9
c6288	21.34	22.34	21.69	21.69	4.8	6.7
c7552	31.87	35.78	34.19	34.19	32.5	–
s5378	13.45	16.78	15.52	15.52	40.4	55.8
s9234	8.13	9.89	8.47	8.47	29.3	40.2
s13207	5.4	6.07	5.56	5.56	30	39.5
s15850	5.09	5.84	5.12	5.12	28.8	39.4
Average	**18.94**	**22.01**	**17.57**	**20.49**	**29.08**	**38.91**
VLS						
s35932	3.67	4.34	3.8	3.8	–	–
s38417	7.55	7.65	7	7	–	–
s38584	7.88	8.33	7.74	7.74	–	–
b17	5.91	6.89	5.9	5.9	–	–
b18	2.45	3.12	2.51	2.51	–	–
b20	7.65	7.99	7.23	7.23	–	–
b21	7.54	7.8	7.15	7.15	–	–
b22	5.88	6.32	5.55	5.55	–	–
Average	**6.06**	**6.55**	**5.35**	**5.86**		

For case2: maximum CTL, minimum hardware overhead

Table 8 Hardware overhead comparison with DC-based and HW-aware

Circuit	Proposed (case3) HW @ FC (%)		HW-aware [9] HW @ FC (%)		DC-based [7] HW @ FC (%)	
	80%	90%	80%	90%	80%	90%
SS						
c432	225.56	240.98	178.26	199.32	70.3	89.1
c499	124.88	137.12	98.36	110.65	–	–
c880	110.98	120.87	86.56	95.56	55.5	76.6
c1355	99.08	113.7	68.59	82.65	–	–
c1908	56.12	62.22	39.45	47.35	25.7	42.2
c2670	126.43	139.01	95.68	110.65	55.7	–
c3540	46.77	50.62	36.97	42.92	18.3	26.1
s298	–	–	–	–	39.4	55.7
s344	59.8	64.11	52.32	58.2	41.6	48.1
s349	65.23	72.79	55.39	63.47	33.5	45
s386	–	–	–	–	52.3	71.5
s400	–	–	–	–	41.2	53.5
s420	78.99	90.66	69.75	79.99	–	–
s444	–	–	–	–	37	47.6
s510	87.22	95.3	73.96	81.69	–	–
s641	86.54	92.19	64.35	70.38	–	–
s715	75.77	81.76	52.69	59.56	–	–
s1196	51.9	57.32	43.93	48.69	45.2	60.7
s1238	57.88	61.34	48.56	53.96	47.1	62.7
s1423	25.14	31.09	16.45	22.68	47.3	58.5
s1488	–	–	–	–	28.8	38.6
s1494	–	–	–	–	30	38.5
Average	**86.14**	**94.44**	**67.57**	**76.73**	**41.80**	**54.29**
LS						
c5315	55.12	62.56	53.03	53.03	37.8	51.9
c6288	22.67	23.9	21.69	21.69	4.8	6.7
c7552	36.23	39.08	34.19	34.19	32.5	–
s5378	15.2	18.1	15.52	15.52	40.4	55.8
s9234	9.35	10.87	8.47	8.47	29.3	40.2
s13207	7.5	8.39	5.56	5.56	30	39.5
s15850	8.34	9.6	5.12	5.12	28.8	39.4
Average	**22.05**	**23.35**	**17.57**	**20.49**	**29.08**	**38.91**
VLS						
s35932	5.13	6.22	3.8	3.8	–	–
s38417	10.32	10.67	7	7	–	–
s38584	9.98	10.32	7.74	7.74	–	–
b17	8.12	9.34	5.9	5.9	–	–
b18	4.44	5.34	2.51	2.51	–	–
b20	9.23	9.78	7.23	7.23	–	–
b21	10.1	10.68	7.15	7.15	–	–
b22	7.32	7.45	5.55	5.55	–	–
Average	**8.08**	**8.6**	**5.35**	**5.86**		

For case3: average CTL, average hardware overhead

Table 9 CTL comparison with DC-based and HW-aware

Circuit	Proposed method (Case 3)	DC-based [7]	HW-aware [9]
c432	1.4 E09	3.96 E10	3.08 E11
c499	4.9 E09	4.5 E11	8.79 E12
c880	6.9 E15	4.93 E11	4.61 E18
c1355	7.2 E10	4.59 E11	1.15 E13
c1908	4.9 E08	3.62 E10	4.62 E10
c2670	6.71 E55	5.72 E11	7.56 E70
c3540	2.34 E13	1.58 E11	6.27 E15
c5315	4,66 E40	3.94 E11	2.06 E54
c6288	2.30 E08	3.11 E10	1.71 E10
c7552	2.79 E49	6.18 E11	1.28 E63
s298	–	9.34 E04	–
s344	5.34 E04	3.86 E07	5.91 E 07
s349	6.78 E04	3.86 E07	5.93 E 07
s386	–	2.79 E04	–
s400	6.31 E07	8.98 E07	–
s420	–	–	1.61 E11
s444	6.16 E06	7.73 E07	–
s510	5.47 E12	–	1.58 E 08
s641	3.17 E09	–	8.19 E 16
s715	7.19 E06	–	1.37 E 11
s1196	6.89 E07	5.98 E06	2.38 E 10
s1238	5.11 E8	7.72 E07	2.40 E 10
s1423	–	5.26 E11	1.23 E 28
s1488	–	1.17 E05	–
s1494	1.18 E08	1.10 E05	–
s5378	6.95 E08	6.38 E11	1.41 E11
s9234	5.31 E14	6.43 E11	2.76 E11
s13207	2.67 E17	6.68 E11	1.84 E19
s15850	5.34 E08	6.86 E11	6.05 E23
s35932	1.67 E08	–	1.43 E11
s38417	6.69 E07	–	1.12 E10
s38584	6.39 E08	–	2.80 E11
b17	6.14 E09	–	5.52 E11
b18	–	–	5.58 E11
b20	4.88 E08	–	4.32 E09
b21	5.23 E08	–	4.35 E09
b22	5.87 E08	–	4.39 E09

For case3: average CTL, average hardware overhead

overhead. According to these graphs, the CTL and the hardware overhead can be arbitrarily selected among the numbers of graphs.

4.2 Comparisons

The proposed design is compared against the state-of-the-art methods by different experiments. In Table 3, CTL is reported for the proposed method, CTL-aware, DC-based, HW-aware, and CE-NEMO in the case that CTL is minimum (maximum hardware overhead). Therefore, the proposed method and the CTL-aware approach have the same CTL as a result of detecting the same test vectors. For the majority of circuits, the CTL is significantly reduced in comparison with DC-based design, while for benchmarks with many input pins e.g., c2670 and c5315, the proposed method reports unpractical times. Other methods focus on reducing hardware overhead and do not focus on CTL; therefore, the CTL is impractical in most case for these methods.

Hardware overhead is compared against CTL-aware, NEMO, and CE-NEMO when the proposed method experiences the best values for CTL. In Table 4, the results show that the hardware overhead is reduced about 5% and 10% for LS and VLS circuits in average for the proposed method in comparison with CTL-aware method. In fact, CTL-aware design consists of a huge combinational circuit that imposes a large hardware overhead on the CUT. Whereas, in the proposed design, almost all modules are LFSRs except the decoding module which maps small number of input vectors to small number of output vectors.

In comparison with NEMO and CE-NEMO, the proposed method significantly reduces the hardware overhead on average as illustrated in Table 5. To represent the superiority of the proposed design comparing with NEMO and CE-NEMO, we illustrate the related hardware overheads in Fig. 9. Moreover, according to Fig. 9 we could deduce that there is a small improvement in CE-NEMO compared to NEMO and the irregular behavior in hardware overhead is apparent for NEMO and CE-NEMO.

The proposed method is compared with DC-based and HW-aware in three different cases:

1- When CTL is minimum and hardware overhead is maximum.
2- When CTL is maximum and hardware overhead is minimum.
3- When both CTL and hardware overhead have their average values.

In all cases, as shown in Tables 6, 7, and 8, DC-based has reported most efficient values in term of hardware overhead hardware overhead for SS benchmarks due to the fact that HW-aware and proposed methods are not appropriate for circuits with a few numbers of logic gates. Following interpretations could be derived using Tables 6, 7, and 8.

In the first case, HW-aware has the most efficient hardware overhead values for LS and VLS circuits as shown in Table 6. DC-based and proposed methods are almost equally efficient in term of hardware overhead in this case. However, the proposed method reduces CTL significantly in comparison with DC-based design as shown in Table 3; while, HW-aware CTL is impractical for the majority of benchmarks.

In the second case, the proposed method CTL is equal with HW-aware CTL reported in Table 3, and both of them are impractical for the majority of circuits. In Table 7, results show that hardware overhead is almost equal for HW-aware and the proposed method especially for LS and VLS benchmarks. In this case, the proposed method reduced CTL about 10% for LS circuits in comparison with DC-based design.

In the third case, the number of selected test vectors for proposed method are limited in which CTL and hardware overhead are acceptable; however, neither one has experienced its most efficient value. In this case, the proposed method lowers CTL significantly as shown in Table 9. In terms of hardware overhead, the propose method is only about 4% more than HW-aware design for both LS and VLS circuits as demonstrated in Table 8. Hardware overhead reduced by about 10% for the proposed method for LS circuits in comparison with DC-based design in this case.

5 Conclusion

in this paper a low-cost concurrent BIST in term of hardware overhead and CTL has been proposed. A small combinational module and LFSRs form the major components of this design, resulting in very low hardware overhead. Furthermore, the proposed method allows for CTL and hardware overhead to be adjusted and tuned to an acceptable level. Compared to the most efficient method, the proposed method achieves a 10% reduction in hardware overhead for large-scale circuits by keeping the CTL minimum. As a result of several experiments, the proposed BIST design has proven to be capable of tuning both the CTL and hardware overhead. In the case that CTL and hardware overhead are equally important, the proposed method significantly has been lowered CTL compared to previous methods, while hardware overhead was only about 4% higher than previous method for both LS and VLS circuits.

Data Availability The datasets generated and/or analyzed during the present study are available from the corresponding author on reasonable request.

Declarations

Conflict of Interest The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

1. Abirami S, Paulin NS, S. Venkateshwaran SP (2015) A concurrentBIST architecture for online input vector monitoring. International Conference on Science, Technology and Management (ICSTM), New Delhi, India, pp 1411–1417
2. Abramovici M, Breuer M, Friedman A (1990) Digital Systems Testing and Testable Design. Computer Science Press
3. Askarzadeh M, Haghparast M, Jabbehdari S (2021) Power consumption reduction in built-in self-test circuits. J Ambient Intell Humaniz Comput 14:1109–1122
4. Biswas S, Das SR, Petriu EM (2006) Space compactor design in VLSI circuits based on graph theoretic concepts. IEEE Trans Instrum Meas 55(4):1106–1118
5. Divyapreethi B, Karthik T (2015) Input vector monitoring concurrent BIST architecture using modified SRAM cells. ARPN J Eng Appl Sci 10(9):4042–4046
6. Jurj SL, Rotar R, Opritoiu F, Vladutiu M (2020) Online Built-In Self-Test Architecture for Automated Testing of a Solar Tracking Equipment. In: Proc. IEEE International Conference on Environment and Electrical Engineering and IEEE Industrial and Commercial Power Systems Europe (EEEIC/I&CPS Europe), pp 1-7
7. Kochte MA, Zoellin CG, Wunderlich H-J (2010) Efficient concurrent self-test with partially specified patterns. J Electron Test 26(5):581–594
8. Menbari A, Jahanirad H (2020) A Concurrent BIST Architecture for Combinational Logic Circuits. In: Proc. 10th International Conference on Computer and Knowledge Engineering (ICCKE), pp 262–267
9. Menbari A, Jahanirad H (2022) A low-cost BIST design supporting offline and online tests. J Electron Test 38(1):107–123
10. Murugan SV, Sathiyabhama B (2021) Bit-swapping linear feedback shift register (LFSR) for power reduction using pre-charged XOR with multiplexer technique in built in self-test. J Ambient Intell Humaniz Comput 12(6):6367–6373
11. Pavlidis A, Louërat MM, Faehn E, Kumar A, Stratigopoulos H-G (2021) SymBIST: Symmetry-based analog and mixed-signal built-in self-test for functional safety. IEEE Transactions on Circuits and Systems I: Regular Papers 68(6):2580–2593
12. Saluja KK, Sharma R, Kime CR (1987) Concurrent comparative testing using BIST resources. In: Proc. International Conference on Computer Aided Design, pp 336–339
13. Saluja KK, Sharma R, Kime CR (1987) Concurrent comparative built-in testing of digital circuits. University of Wisconsin, Engineering Experiment Station, Madison, Wisconsin, USA
14. Saluja KK, Sharma R, Kime CR (1988) A concurrent testing technique for digital circuits. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 7(12):1250–1260
15. Shivakumar V, Senthilpari C, Yusoff Z (2021) A low-power and area-efficient design of a weighted pseudorandom test-pattern generator for a test-per-scan built-in self-test architecture. IEEE Access 9:29366–29379
16. Voyiatzis I (2012) Input Vector Monitoring On line Concurrent BIST based on multilevel decoding logic. In: Proc. Design, Automation & Test in Europe Conference & Exhibition (DATE), pp 1251–1256
17. Voyiatzis I, Efstathiou C (2013) Input vector monitoring concurrent BIST architecture using SRAM cells. IEEE Trans Very Large-Scale Integr (VLSI) Syst 22(7):1625–1629
18. Voyiatzis I, Halatsis C (2005) A low-cost concurrent BIST scheme for increased dependability. IEEE Transactions on Dependable and Secure Computing 2(2):150–156
19. Voyiatzis I, Paschalis A, Gizopoulos D, Halatsis C, Makri FS, Hatzimihail M (2008) An input vector monitoring concurrent BIST architecture based on a precomputed test set. IEEE Transactions on Computers 57(8):1012–1022
20. Voyiatzis I, Paschalis A, Gizopoulos D, Kranitis N, Halatsis C (2005) A concurrent built-in self test architecture based on a self-testing RAM. IEEE Trans Reliab 54(1):69–78
21. Wu TB, Liu HZ, Liu PX, Guo DS, Sun HM (2013) A cost-efficient input vector monitoring concurrent online BIST

scheme based on multi-level decoding logic. J Electron Test 29(4):585–600

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.

Ahmad Menbari received his BS degree in electrical engineering from the Department of Electrical Engineering, University of Kurdistan, Sanandaj, Iran, in 2019. His main research interests are fault-tolerant systems, digital circuit testing, and reliability analysis of logic circuits.

Hadi Jahanirad received his BS degree in electrical engineering from the Department of Electrical Engineering, Khaje Nasir Toosi University, Tehran, Iran, in 2006, and his MS and a Ph.D. degree from Iran University of Science and Technology, Tehran, Iran in 2008 and 2012, respectively. Since 2013, he has been with the Department of Electrical Engineering, University of Kurdistan, Sanandaj, Iran, where he is now an associate professor. His main research interests are digital system design, VLSI design, reliability analysis of logic circuits, digital circuit testing, approximate computing, and evolutionary computing.