Communications on Applied Electronics (CAE) – ISSN : 2394-4714
Foundation of Computer Science FCS, New York, USA
Volume 4– No.8, April 2016 – www.caeaccess.org

New Test Pattern Generators for the BIST Pseudo-Exhaustive Testing based on Coding Theory Principles

Mohamed H. El-Mahlawy
Visiting Associate Professor in
Military Technical College, Cairo, Egypt

Winston Waller
University of Kent, Canterbury,
Kent, CT2 7NT, UK

ABSTRACT

In this paper, an efficient algorithm to design convolved LFSR/SR (Linear Feedback Shift Register / Shift Register) for the pseudo-exhaustive testing (PET) is presented as far as the lengths of the test set and hardware overhead are concerning. In this algorithm, an efficient search to reduce the constraint in the size of the shift register (SR) segment and makes an efficient search to restrict on the number of feed forward stages into two stages at most and no restriction on the size of the SR segment. The residues are assigned such that minimum hardware overhead is achieved. This search generates several possible solutions for each case, from which the minimal hardware solutions may be chosen. In addition, a new test pattern generator (TPG) for the PET that bridges the gap between convolved LFSR/SR and permuted LFSR/SR is presented. It is considered to be the optimal pseudo-exhaustive test pattern generator (PETPG) as far as the lengths of the test set and hardware overhead are concerning. An efficient residue assignment for the inputs of the CUT to reduce the hardware overhead is presented. With small number of permutations in the assigned residues, the chance of obtaining efficient solutions may be increased. The presented generator in this paper is considered the general form of the PETPG. The simple LFSR/SR, the permuted LFSR/SR, and convolved LFSR/SR are considered the special case. The experimental results for all combinational benchmark circuits [1] indicate the superiority of the presented approach with respect to previous published works.

Keywords

Design for testability of VLSI design, pseudo-exhaustive testing, pseudo-exhaustive test pattern generator, LFSR/SR, permuted LFSR/SR, convolved LFSR/SR.

1. INTRODUCTION

Traditionally, the CUT is tested by the automatic test equipment (ATE), which can store the applied test patterns and the expected test response of the CUT [2-3]. In the Built-In Self-Test (BIST), extra circuitry is added around the original CUT to test itself [4]. The BIST incorporates a TPG such as autonomous linear feedback shift register (ALFSR) shown in Figure 1, and a test response compactor (TRC) such as an LFSR shown in Figure 2 and an multi-input shift-register (MISR) shown in Figure 3 [5-6]. Besides, a BIST controller is incorporated into the system (core) logic to realize self-test operations [4, 7]. Figure 1, Figure 2, and Figure 3 have c_i's as binary constants, $c_i = 1$ implies that a connection exists, while $c_i = 0$ implies that there is no connection.

The basic BIST execution *parallel BIST or test-per-clock technique* is shown in Figure 4. In the parallel BIST, the test patterns are applied from the TPG and test responses are captured in the TRC (as MISR) every clock cycle [7-10].

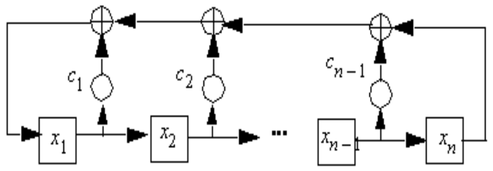


Fig. 1: ALFSR as the TPG.

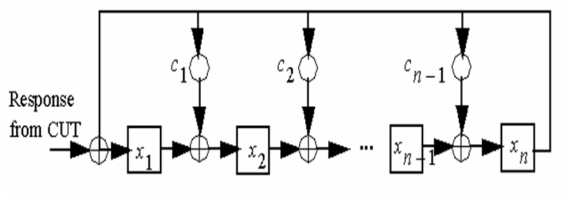


Fig. 2: LFSR as the TRC in serial BIST

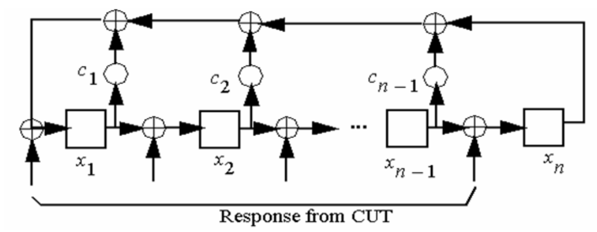


Fig. 3: MISR as the TRC in parallel BIST.

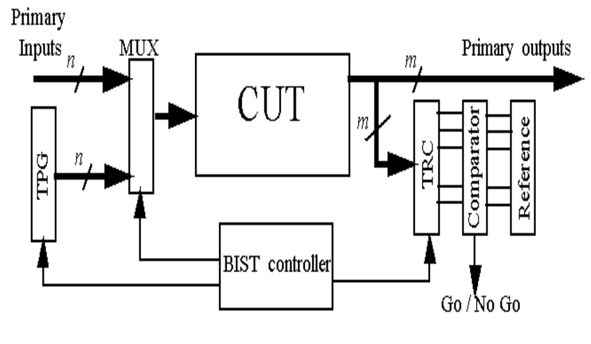


Fig. 4: Basic parallel BIST architecture.

There are four main testing types in the BIST architecture. They are exhaustive testing [4], pseudo-random testing [5-6], deterministic testing [11-14], and the pseudo-exhaustive testing (PET) [4, 15-18]. In this paper, the PET is selected to study. The PET approach retains almost all benefits of an exhaustive testing but usually requires far fewer test patterns. The time required for the PET depends on the sizes of the output cones, shown in Figure 5. For circuits with restricted output dependency, the PET provides an alternative test method. The choice of the PET depends on whether or not any combinational circuit outputs depend on all of the circuit inputs. If any circuit output depends on all of its inputs, a partitioning (or segmentation) test technique must be used to test these circuits [4, 16]. The PET reduces the testing time to a feasible workable value.

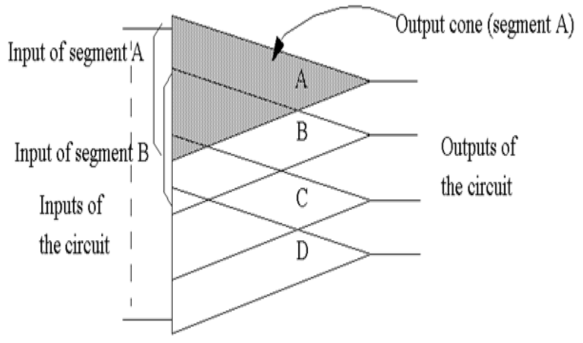


Fig. 5: Segmentation approach in the PET.

2. PREVIOUS RESEARCH

Many single-test pattern generators were proposed for the PET. Examples are *syndrome driver counters* (SDCs) [19], *constant-weight counters* (CWCs) [20], *condensed LFSRs* [21], *cyclic LFSRs* [22], *combined LFSR and XOR gates* (LFSR/XORs) [23-24], *combined LFSR and shift register* (simple LFSR/SRs) [25], *permuted LFSR/SRs* [26], and *convolved LFSR/SRs* [15, 27-28]. It has been found the PETPGs based on the universal test set method require longer test lengths than the PETPGs based on the output-specific test set method [4, 27-28]. Convolved LFSR/SR is a generator based on the output-specific approach. It is considered to be the suitable PETPG as far as the lengths of test set and hardware overhead are concerning. In [27-28], the size of shift register segment is constrained to have a desired minimum length to reduce the number of feed forward stages. This constraint weakens the potential of using convolved LFSR/SR and in a lot of cases, the number of XOR gates needed for convolved LFSR/SR is high. Sometimes, the required search time to generate the minimal test set is large, or the search procedure requires the test set length of greater than the minimal test set. Dimitrios Kagaris and Spyros Tragoudas [26] have suggested a permuted LFSR/SR which is a simple LFSR/SR that drives a permuted set of inputs. A simple LFSR/SR, which is not capable of generating the optimal test set length because of fixed assignment of residues, may become feasible through reassigning the residues. From the results in [26], additional segmentation cells are required to find an applicable primitive polynomial.

This paper introduces an efficient approach to design a generator that generates a pseudo-exhaustive test pattern set. It is required to design a generator with minimal test length and minimal hardware overhead. The algorithm to design the convolved LFSR/SR provides efficient search iteration for residue assignment to the inputs of the CUT so as to increase the number of potential solutions and thus reduce the hardware overhead. In addition, a new generator which bridges the gap between convolved LFSR/SR and permuted LFSR/SR is presented. With small number of permutations in the assigned residues, the chance of obtaining efficient results may be increased. A simple efficient heuristic approach for permutation is presented. The new generator is considered the general form of the PETPG. The experimental results indicate the efficiency of the presented approach. The results also show that the insertion of additional segmentation cells required by the method outlined in [26] is not required.

The basic concept of the PET will be started in the next section. The new algorithm to design the convolved LFSR/ SR as the PETPG is presented in section 4. Then, the new PETPG is presented in section 5. The experimental results will be in section 6 and the conclusions in section 7.

3. BASIC CONCET OF THE PST

The combinational CUT with *n* inputs and *m* outputs is modelled as a direct acyclic graph. The nodes represent gates and the interconnection signals are represented by edges. Each output cone of the circuit forms a subgraph need not be disjoint. The *dependency set*, D_i, of the output cone *i* is considered the set of the primary inputs and the pseudo-primary inputs that feed it directly or affect it through another node. The *dependency*, $|D_i|$, of the output cone *i* is the cardinality of its dependency set. Let *k* be the maximum value among the dependencies of the *m* output cones. The circuit can be characterized as an (*n, m, k*) circuit. The circuit is segmented into *m* output cones, and each cone is exhaustively tested. The test ensures detection of all detectable combinational faults with a single-test pattern within individual cones of the CUT without fault simulation. The time required for the PET depends on the sizes of the output cones.

The first *w* stages of the (*n*, *w*) simple LFSR/SR are configured as an LFSR with primitive polynomial $p(x)$ of degree *w*. The remaining *n* - *w* stages are connected as a shift register (SR). The residues $R_0, R_1, R_2 \ldots R_{w-1}$ given by $1+ x^1 + x^2 + \ldots + x^{w-1}$ represent the *w* test signals generated by the LFSR portion. These residues are fixed for LFSR portion independent of $p(x)$. The remaining *n* - *w* residues (test signals) assigned to remaining *n* - *w* circuit inputs are linear combination of the residues of the LFSR portion. The residues R_w through R_{n-1} are fixed by specific $p(x)$. For an (*n*, *w*) simple LFSR/SR based on $p(x)$, stage *i* generates the residue x^i mod $p(x)$ denoted as R_i. The minimal test set for the PET is generated when *w* equals *k* or the search procedure requires the test set length equals 2^k. The primitive polynomial $p(x)$ exhaustively exercises the dependency set $D_i = \{ d_1, d_2, \ldots, d_k\}$, if and only if the *k* residues, x^{dj} mod $p(x)$ and $1 \leq j \leq k$ are linearly independent [29].

Condition 1: For each output cone *i*, all residues R_{dj}, $d_j \in D_i$, , $1 \leq j \leq |D_i|$, must be linearly independent.

Definition 1: The residues assigned to a dependency set of an output cone that satisfies condition 1 is called *applicable residues***.**

Definition 2: A primitive polynomial $p(x)$ that satisfies condition 1 for all dependency sets of the output cones is called an *applicable polynomial*.

Example 1: The dependency sets of the (8, 6, 4) CUT are $D_0 = \{0, 1, 2\}$, $D_1 = \{0, 2, 3, 6\}$, $D_2 = \{1, 4, 5, 6\}$, $D_3 = \{0, 2, 4, 5\}$, $D_4 = \{3, 4, 5\}$, $D_5 = \{0, 1, 2, 7\}$. The (8, 4) simple LFSR/SR based on $p(x) = 1 + x^3 + x^4$ is shown in Figure 6.

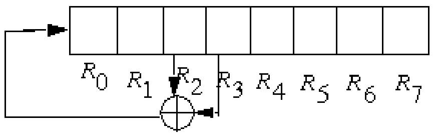


Fig. 6: The residue for (8, 4) LFSR/SR.

In Table 1, residues from R_0 *to* R_7 are calculated based on x^i mod $p(x)$ of R_i in the polynomial form and the digital form.

Table 1. Residues from R_0 *to* R_7 of $p(x) = 1 + x^3 + x^4$

Residue	Residue in the polynomial form	Residue in the binary form
R_0	1	1000
R_1	x	0100
R_2	x^2	0010
R_3	x^3	0001
R_4	$1 + x^3$	1001
R_5	$1 + x + x^3$	1101
R_6	$1 + x + x^2 + x^3$	1111
R_7	$1 + x + x^2$	1110

Residues are considered for assignment to CUT inputs in the order generated by successive stages of a simple LFSR/SR. To check, for example, if the residues assigned to D_1 are *applicable residue*s or not, the residues, R_0, R_2, R_3, and R_6, are assigned to the circuit inputs, I_0, I_2, I_3, and I_6, respectively. (The dependency set, D_1, which equals {0, 2, 3, 6}, is also referred to as a 4-subset of 8.) Matrix M_{D1} is arranged, and transformed to the upper triangle matrix, U_{D1}, whose determinant equals 1. Then, these residues are *applicable residue*s.

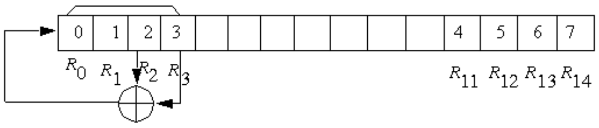
$$M_{D1} = \begin{bmatrix} R_0 & R_2 & R_3 & R_6 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \qquad U_{D1} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

These checks are repeated for all dependency sets. The residues assigned to D_2, and D_5 are linearly dependent because Matrix M_{D2} is arranged, and transformed to the upper triangle matrix, U_{D2}, whose determinant equals 0, and the determinant of M_{D5} is equals 0, also. Therefore, the polynomial $p(x) = 1 + x^3 + x^4$ is not applicable.

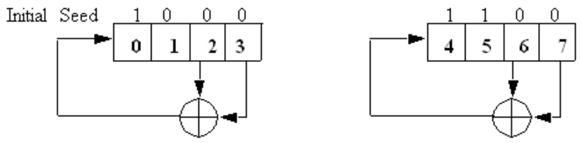
$$M_{D2} = \begin{bmatrix} R_1 & R_4 & R_5 & R_6 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \qquad U_{D1} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad M_{D5} = \begin{bmatrix} R_0 & R_1 & R_2 & R_7 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Lemma 1 [27-28]: An (n, w) multiple LFSR/SR exists for generating the PET for an (n, m, k) circuit if and only if there exists an (n, w) convolved LFSR/SR for the circuit where the length of each SR segment is at least w.

Example 2: For the (15, 4) simple LFSR/SR based on $p(x) = 1 + x^3 + x^4$. For each stage i in the (15, 4) simple LFSR/SR, the residues will be calculated based on x^i mod $p(x)$. An initial seed for the LFSR stages is 1000 is seen to be shifted from the left during the initialization phase. For the (15, 4) simple LFSR/SR, the initial seed is generated for all stages of the simple LFSR/SR and all patterns generated from all stages of the (15, 4) simple LFSR/SR in the initialization phase and the testing phase are shown in Table 2. In Table 2, stage 0 is considered the left-most bit of the pattern and stage 14 is considered the right-most bit of the pattern. In the last row of the second column of Table 2, the initial seed of all stages of (15, 4) simple LFSR/SR is shown by bold font which is the initial pattern in the testing phase. From example 1, the (8, 4) simple LFSR/SR based on $p(x)$ cannot exhaustively test all output cones of the CUT. Using multiple LFSR/SR, the *applicable residues* of all output cones in example 1 are 0-3 11-14 which exhaustively test all output cones of the CUT. Figure 7 shows a multiple LFSR/SR, composed of two (4, 4) simple LFSR/SRs. Both simple LFSR/SRs are based on the same $p(x)$. The initial seed of the first (4, 4) simple LFSR/SR is 1000 (the first row of the third column in Table 2). The initial seed of the second (4, 4) simple LFSR/SR is 1100 (the first row of the third column in Table 2). The contents of the second (4, 4) simple LFSR/SR will become 1000 after eleven clock cycles. Hence the first (4, 4) simple LFSR/SR generates R_0 through R_3 and the second (4, 4) simple LFSR/SR generates R_{11} through R_{14}. This multiple LFSR/SR generates patterns to exhaustively test all output cones. Thus, the initial seeds can be manipulated to generate the desired residues from the simple LFSR/SRs. The length of each SR segment must be at least w (according to lemma 1) to work as independent simple LFSR/SRs, run in parallel.



(a) Simple LFSR/SR.



(b) Multiple LFSR/SR.

Fig. 7: Multiple LFSR/SR as the special case of the convolved LFSR/SR.

Table 2. Initial seed determination of the Simple LFSR/SR.

Pattern number	Initialization phase	Testing phase
0	100000000000000	**1000**11110101**1100**
1	010000000000000	010001111010110
2	001000000000000	001000111101011
3	100100000000000	100100011110101
4	110010000000000	110010001111010
5	011001000000000	011001000111101
6	101100100000000	101100100011110
7	010110010000000	010110010001111
8	101011001000000	101011001000111
9	110101100100000	110101100100011
10	111010110010000	111010110010001
11	111101011001000	11110101100**1000**
12	011110101100100	011110101100100
13	001111010110010	001111010110010
14	000111101011001	000111101011001
The initial seed	**100011110101100**	100011110101100

Communications on Applied Electronics (CAE) – ISSN : 2394-4714
Foundation of Computer Science FCS, New York, USA
Volume 4– No.8, April 2016 – www.caeaccess.org

4. NEW ALGORITHM TO DESIGN THE CONVOLVED LFSR/SR

4.1 Idea of the Residue Assignment

For an (*n*, *w*) convolved LFSR/SR, all possible residues can be considered for input assignment. However, the number of possible residues increases exponentially with the degree of the primitive polynomial, *p*(*x*). In practice, only a few residues are considered for input assignment, *Q*. It is difficult to get the benefit of the convolved LFSR/SR without a good search algorithm to obtain the minimal hardware overhead solution in reasonable time. The source of hardware overhead of the convolved LFSR/SR is the number of XOR gates used to realize the individual feed forward (FF) stages. To minimize the area overhead, it is required to reduce the number of FF stages. The structure of the convolved LFSR/SR has been restricted to be one of the four forms in Figure 8. These forms restrict the number of FF stages into two stages at most.

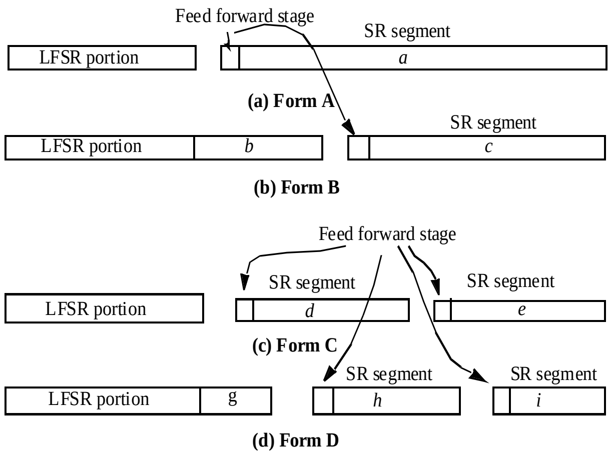


Fig. 8: Different forms of the convolved LFSR/SR.

Form A consists of two parts, the first part is the LFSR portion with size *w*, and the second part is the SR segment *a* with size *n* - *w*. If *applicable residues* are assigned to all CUT inputs in this form and $n - w \geq w$, the generator will be a multiple LFSR/SR and if *applicable residues* are assigned to all CUT inputs in this form and $n - w < w$, the number of FF stages will be one. Form B consists of two parts, the first part is a $(w + t_b, w)$ simple LFSR/SR, and the second part is the SR segment *c* with size $n - (w + t_b)$, where t_b is the size of the SR segment *b*. If *applicable residues* are assigned to all CUT inputs in this form and $n - (w + t_b) \geq w$, the generator will be a multiple LFSR/SR and if *applicable residues* are assigned to all CUT inputs in this form and $n - (w + t_b) < w$, the number of FF stages will be one.

Form C consists of three parts, the first part is the LFSR portion with size *w*, the second part is the SR segment *d* with size t_d, and the third part is the SR segment *e* with size $n - (w + t_d)$. If *applicable residues* are assigned to all CUT inputs in this form, the number of FF stages will be at most two. If the size of two or one of the SR segments *d* and *e* is greater than or equal to *w*, the generator will be a multiple LFSR/SR or at most it is required to feed a residue at the input of the FF stage. Form D consists of three parts, the first part is $(w + t_g, w)$ simple LFSR/SR, the second part is the SR segment *h* with size t_h, and the third part is the SR segment *i* with size $n - (w + t_g + t_h)$. If *applicable residues* are assigned to all CUT inputs in this form, the number of FF stages will be at most two. If the size of at least one of the SR segments *h* and *i* is greater than or equal to *w*, the generator will be a multiple LFSR/SR or at most it is required to feed a residue at the input of the FF stage.

This section presents the new algorithm to design the convolved LFSR/SR, referred to as *NEW_CONV*. There are two phases in the *NEW_CONV*. The first phase is to search for an applicable primitive polynomial using the candidate primitive polynomials. In the case of the existence of this polynomial, the convolved LFSR/SR will be a simple LFSR/SR. If no primitive polynomial is applicable, the second phase commences. This phase searches for a residue assignment to generate the convolved LFSR/SR in one of the four forms based on a primitive polynomial with minimum terms. All residues from 0 through (*w* - 1) are automatically assigned to the inputs 0 through (*w* - 1) and *n* - *w* residues of *Q* – *w* residues are assigned such that all output cones may be exhaustively tested. The procedure for selecting *n* - *w* residues of *Q* - *w* residues is required to achieve minimal hardware. The search procedure in this phase selects (*n* - *w*) residues of (*Q* - *w*) residues in a manner which restricts the number of FF stages to a maximum of two without restriction in the size of the SR segment to get the convolved LFSR/SR in any of the forms shown in Fig. 8. The search problem is converted to generate possible choices that satisfy these constraints.

An example is given for simplicity. It is taken for the (12, *m*, 4) CUT, and (12, 4) convolved LFSR/SR with *Q* = 15. There are three possible choices for the (12, 4) convolved LFSR/SR of form A. In Table 3, to Table 6, the residue assignment for CUT inputs utilizes those residues assigned to 1 and neglects residues assigned to 0. Basically in residue assignment, R_0 through R_3 are assigned to inputs 0 through 3. In the first possible choice, R_5 through R_{12} are assigned to inputs 4 through 11 and check for *applicable residues* for all output cones. If any output cone does not satisfy condition 1, then go to the next possible choice in form A. If all output cones satisfy condition 1, then a suitable convolved LFSR/SR of form A is found. In general, the number of possible choices in the case of form A is referred to N_A according to equation (1).

$$N_A = Q - n. \qquad (1)$$

Table 3. Form A with *n* =12, *w* =*k* = 4, and *Q* = 15.

R_4	R_5	R_6	R_7	R_8	R_9	R_{10}	R_{11}	R_{12}	R_{13}	R_{14}
0	1	1	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	1	1	0
0	0	0	0	1	1	1	1	1	1	1

There are forty-two possible choices for the (12, 4) convolved LFSR/SR of form B and form C for residue assignment. Table 4 and Table 5 illustrate the possible choices of form B and form C for (12, 4) convolved LFSR/SR, respectively. For the possible choices of form B and form C, if all output cones satisfy condition 1 after residue assignment, then a suitable convolved LFSR/SR of form B and form C is found. In general, the number of possible choices in the case of form B and form C is referred to N_{BC} according to equation (2).

$$N_{BC} = \sum_{i=0}^{Q-n-1} (Q-n-i)(n-w-1) \qquad (2)$$

There are sixty-three possible choices for the (12, 4) convolved LFSR/SR of form D for residue assignment. Table 6 illustrates the possible choices of form D for (12, 4) convolved LFSR/SR. For the possible choices of form D, if all output cones satisfy condition 1 after residue assignment, then a suitable convolved LFSR/SR of form D is found. In general, the number of possible choices in the case of form D is referred to N_D according to equation (3).

$$N_D = \sum_{j=0}^{n-w-3} \sum_{i=1}^{Q-n-1} (Q-n-i)(n-w-j-2) \quad (3)$$

The search is divided into two basic searches. The first search is the partial search based on form A, B, and C. The number of the possible choices in the partial search will be the sum of N_A and N_{BC}. The second search is the full search by using all possible choices. The number of those patterns in the full search will be the sum of N_A, N_{BC}, and N_D. In the previous equations, w is the degree of the primitive polynomial $p(x)$, Q is the limit of the number of the available residues for residue assignment, and n is the number of CUT inputs. When Q increases, the number of possible choices is increased. This may increase the chance to get several solutions with minimal hardware overhead in the form of multiple LFSR/SRs.

Table 4. Form B with *n* =12, *w* =*k* = 4, and *Q* = 15.

R_4	R_5	R_6	R_7	R_8	R_9	R_{10}	R_{11}	R_{12}	R_{13}	R_{14}
1	0	1	1	1	1	1	1	1	0	0
1	0	0	1	1	1	1	1	1	1	0
1	0	0	0	1	1	1	1	1	1	1
1	1	0	1	1	1	1	1	1	0	0
1	1	0	0	1	1	1	1	1	1	0
1	1	0	0	0	1	1	1	1	1	1
1	1	1	0	1	1	1	1	1	0	0
1	1	1	0	0	1	1	1	1	1	0
1	1	1	0	0	0	1	1	1	1	1
1	1	1	1	0	1	1	1	1	0	0
1	1	1	1	0	0	1	1	1	1	0
1	1	1	1	0	0	0	1	1	1	1
1	1	1	1	1	0	1	1	1	0	0
1	1	1	1	1	0	0	1	1	1	0
1	1	1	1	1	0	0	0	1	1	1
1	1	1	1	1	1	0	1	1	0	0
1	1	1	1	1	1	0	0	1	1	0
1	1	1	1	1	1	0	0	0	1	1
1	1	1	1	1	1	1	0	1	0	0
1	1	1	1	1	1	1	0	0	1	0
1	1	1	1	1	1	1	0	0	0	1

Table 5. Form C with *n* =12, *w* =*k* = 4, and *Q* = 15.

R_4	R_5	R_6	R_7	R_8	R_9	R_{10}	R_{11}	R_{12}	R_{13}	R_{14}
0	1	0	1	1	1	1	1	1	1	0
0	1	0	0	1	1	1	1	1	1	1
0	1	1	0	1	1	1	1	1	1	0
0	1	1	0	0	1	1	1	1	1	1
0	1	1	1	0	1	1	1	1	1	0
0	1	1	1	0	0	1	1	1	1	1
0	1	1	1	1	0	1	1	1	1	0
0	1	1	1	1	0	0	1	1	1	1
0	1	1	1	1	1	0	1	1	1	0
0	1	1	1	1	1	0	0	1	1	1
0	1	1	1	1	1	1	0	1	1	0
0	1	1	1	1	1	1	0	0	1	1
0	1	1	1	1	1	1	1	0	1	0
0	1	1	1	1	1	1	1	0	0	1
0	0	1	0	1	1	1	1	1	1	1
0	0	1	1	0	1	1	1	1	1	1
0	0	1	1	1	0	1	1	1	1	1
0	0	1	1	1	1	0	1	1	1	1
0	0	1	1	1	1	1	0	1	1	1
0	0	1	1	1	1	1	1	0	1	1
0	0	1	1	1	1	1	1	1	0	1

Table 6. Form D with *n* =12, *w* =*k* = 4, and *Q* = 15.

R_4	R_5	R_6	R_7	R_8	R_9	R_{10}	R_{11}	R_{12}	R_{13}	R_{14}
1	0	1	0	1	1	1	1	1	1	0
1	0	1	0	0	1	1	1	1	1	1
1	0	1	1	0	1	1	1	1	1	0
1	0	1	1	0	0	1	1	1	1	1
1	0	1	1	1	0	1	1	1	1	0
1	0	1	1	1	0	0	1	1	1	1
1	0	1	1	1	1	0	1	1	1	0
1	0	1	1	1	1	0	0	1	1	1
1	0	1	1	1	1	1	0	1	1	0
1	0	1	1	1	1	1	0	0	1	1
1	0	1	1	1	1	1	1	0	1	0
1	0	1	1	1	1	1	1	0	0	1
1	0	0	1	0	1	1	1	1	1	1
1	0	0	1	1	0	1	1	1	1	1
1	0	0	1	1	1	0	1	1	1	1
1	0	0	1	1	1	1	0	1	1	1
1	0	0	1	1	1	1	1	0	1	1
1	0	0	1	1	1	1	1	1	0	1
1	1	0	1	0	1	1	1	1	1	0
1	1	0	1	0	0	1	1	1	1	1
1	1	0	1	1	0	1	1	1	1	0
1	1	0	1	1	0	0	1	1	1	1
1	1	0	1	1	1	0	1	1	1	0
1	1	0	1	1	1	0	0	1	1	1
1	1	0	1	1	1	1	0	1	1	0
1	1	0	1	1	1	1	0	0	1	1
1	1	0	1	1	1	1	1	0	1	0
1	1	0	1	1	1	1	1	0	0	1
1	1	0	0	1	0	1	1	1	1	1
1	1	0	0	1	1	0	1	1	1	1
1	1	0	0	1	1	1	0	1	1	1
1	1	0	0	1	1	1	1	0	1	1
1	1	0	0	1	1	1	1	1	0	1
1	1	1	0	1	0	1	1	1	1	0
1	1	1	0	1	0	0	1	1	1	1
1	1	1	0	1	1	0	1	1	1	0
1	1	1	0	1	1	0	0	1	1	1
1	1	1	0	1	1	1	0	1	1	0
1	1	1	0	1	1	1	0	0	1	1
1	1	1	0	1	1	1	1	0	1	0
1	1	1	0	1	1	1	1	0	0	1
1	1	1	0	0	1	0	1	1	1	1
1	1	1	0	0	1	1	0	1	1	1
1	1	1	0	0	1	1	1	0	1	1
1	1	1	0	0	1	1	1	1	0	1
1	1	1	1	0	1	0	1	1	1	0
1	1	1	1	0	1	0	0	1	1	1
1	1	1	1	0	1	1	0	1	1	0
1	1	1	1	0	1	1	0	0	1	1
1	1	1	1	0	1	1	1	0	1	0
1	1	1	1	0	1	1	1	0	0	1
1	1	1	1	0	0	1	0	1	1	1
1	1	1	1	0	0	1	1	0	1	1
1	1	1	1	0	0	1	1	1	0	1

R_4	R_5	R_6	R_7	R_8	R_9	R_{10}	R_{11}	R_{12}	R_{13}	R_{14}
1	1	1	1	1	0	1	0	1	1	0
1	1	1	1	1	0	1	0	0	1	1
1	1	1	1	1	0	1	1	0	1	0
1	1	1	1	1	0	1	1	0	0	1
1	1	1	1	1	0	0	1	0	1	1
1	1	1	1	1	0	0	1	1	0	1
1	1	1	1	1	1	0	1	0	1	0
1	1	1	1	1	1	0	1	0	0	1
1	1	1	1	1	1	0	0	1	0	1

The possible choices of forms B, C, and D consist of two parts. The first part is fixed in every portion in the form (shadow cells in Table 4, Table 5, and Table 6 besides the LFSR portion). The second part is variable. For every portion, the linear independence of the residue assignment of the fixed part is checked. If that check is linearly dependent, there is no need to continue the search for that portion and the procedure will move to the next portion in the form. If this check is linearly independent, the algorithm will continue with the residue assignment in that portion.

Sometimes, if R_w, assigned to input w, is not an applicable residue with respect to residues R_0 to R_{w-1}, the residue assignment using all possible choices of form B and form D does not satisfy condition 1. It is required to determine which applicable residue from $w + 1$ to $((Q- 1) - (n - w - 1))$ assigned to input w. The search from the portion corresponding to the assignment input w to an applicable residue is started. So, it is not useful to begin the search from the first possible choice, and it is required to determine the first applicable residue to assign input w before the search begins. For high values of Q, the determination of the first applicable residue assigned to input w and the check of the fixed part of the first possible choice of every portion will save considerable search time. The applicable residues will take any form of the convolved LFSR/SR in Figure 8 when this assignment satisfies condition 1. If the assigned residues do not achieve condition 1 for any output cone, the search procedure progressively assigns residues to all inputs for another possible choice.

From the explanation of the presented search, the residue assignment is restricted by the number of FF stages to be at most two with no restriction in the size of the SR segment. Residue assignment here is parallel, i.e., the search procedure assigns residues to all inputs of the CUT at the same time and checks if condition 1 is satisfied or not to obtain the convolved LFSR/SR in any of the forms shown in Figure 8. The restriction in the number of FF stages and lack of restriction in the size of the SR segment increases the chance of obtaining several solutions and from these solutions; the solution with the least hardware overhead may be selected.

Example 3: A portion from the possible choices of form B will be taken to demonstrate how the search sequence occurs.

R_4	R_5	R_6	R_7	R_8	R_9	R_{10}	R_{11}	R_{12}	R_{13}	R_{14}
1	1	1	1	0	1	1	1	1	0	0
1	1	1	1	0	0	1	1	1	1	0
1	1	1	1	0	0	1	1	1	1	1

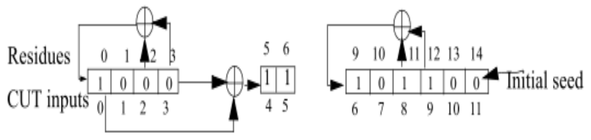
Using the first possible choice, residues 0-3, 4-7, 9-12 are assigned to the CUT inputs and check condition 1 of all output cones. If any output cone is linearly dependent, go to the next possible choice. For each possible choice in the portion, the residues 0-3, 4-7 are fixed. The linear independence of residues 0-3, 4-7 assigned to the inputs 0 through 7 is first checked for all dependency sets of the output cones. If this check is linearly dependent, there is no need to continue the search in that portion and the procedure will exit from that portion to the first possible choice in the next portion. If this check is linearly independent, the algorithm will continue with the residue assignment in this portion.

Example 4: The dependency sets of the output cones of the (12, 6, 4) CUT (D_j, $1 \le j \le 6$) are {1, 3, 5, 11}, {2, 6, 7, 12}, {2, 3, 8, 10}, {9, 11, 12}, {3, 7, 9, 11}, {2, 10, 11}, respectively.

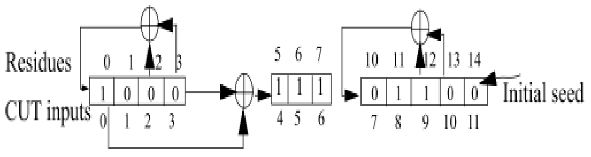
Three solutions are achieved based the algorithm *NEW_CONV* for this CUT using primitive polynomial $p(x) = 1 + x^3 + x^4$. The second column in Table 7 shows the applicable residues. In the first and second solution, it is required to feed R_4 to the input of the FF stage. The rest of the SR segments are independent simple LFSR/SRs. The number of XOR gates required to realize the convolved LFSR/SR to exhaustively test all output cones is 3 in all solutions shown in Figure 9. Using the algorithm in [27-28], the following unique solution 0-6, 8-9, 12-14 is achieved. It is required to feed R_7 and R_{11} to the inputs of the first and second feed forward stage, respectively. The number of XOR gates required to realize the convolved LFSR/SR to exhaustively test all output cones is 4 shown in Figure 9d.

Table 7. Different solutions for example 4

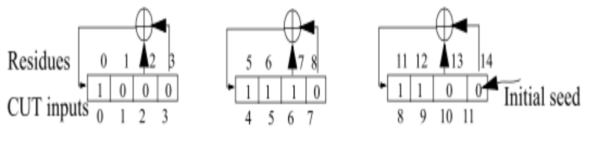
Possible Solution	Applicable residues	Feeding residue	XOR gates
1	0-3, 5-6, 9-14	R_4	3
2	0-3, 5-7, 10-14	R_4	3
3	0-3, 5-8, 11-14	-	3



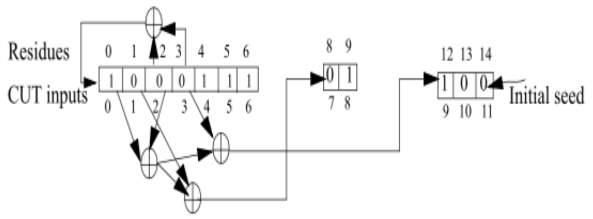
(a) First solution of algorithm *NEW_CONV*.



(b) Second solution of algorithm *NEW_CONV*.



(b) Third solution of algorithm *NEW_CONV*.



(d) Solution of algorithm in [27-28].

Fig. 9: Several convolved LFSR/SRs.

4.1 Algorithm NEW_CONV

Input: The dependency sets of output cones, w ($\geq k$), Q, partial or full search, and the number of required solutions (S).

Output: Applicable residues for CUT inputs and the required initial seed.

1. Phase 1

1.1 Retrieve all primitive polynomials or the subset of all primitive polynomials of degree w from a stored file and put them in queue L.

1.2 Select the primitive polynomial, $p(x)$ from L.

1.3 Residues from R_0 through R_{n-1} are generated, based on $p(x)$, using subroutine RESIDUE [4].

1.4 Condition 1 is checked for all dependency sets of the output cones.

1.5 If condition 1 is satisfied for all dependency sets, then the corresponding $p(x)$ is the applicable polynomial, store it as a solution and calculate the initial seed, discussed in example 2.

1.6 If the number of solutions exceed S, exit.

1.7 If there are other primitive polynomials in L, go to step 1.3.

1.8 If all candidate primitive polynomials are finished and no applicable polynomial exists, go to phase 2.

2. Phase 2

2.1 Select a primitive polynomial with minimum terms from L, and generate all residues (R_0 through R_{Q-1}) using subroutine RESIDUE in [4].

2.2 Determine the first applicable residue assigned to input w.

2.3 If R_w is not the applicable residue to input w, then all possible choices of form B and form D will be ignored.

2.4 If R_w is the applicable residue to input w and the full search is not selected, all possible choices of form D will be ignored.

2.5 Through the generation of possible choices for residue assignment, do the following steps.

2.5.1 Condition 1 of the assigned residues to the fixed part of the first possible choice of specific portion in any convolved LFSR/SR forms is checked for all dependency sets of the output cones.

2.5.2 If this check is not satisfied, this portion is completely ignored and the next one is considered. Then, go to step 2.5.1.

2.5.3 For each possible choice in the portion and the corresponding residue assignment, condition 1 is checked for all dependency sets of the output cones. If this condition is not satisfied, try the next one in the same portion. In the case of satisfying condition 1, the corresponding assigned residues are applicable residues to exhaustively test all output cones and then the initial seed is calculated. This result is stored as a solution. If the number of the solutions exceed S, exit.

2.5.4 If a new portion begins, go to step 2.5.1.

2.5.5 If the complete set of the generated possible choices is finished and L is not empty, go to step 2.1.

2.5.6 If the complete set of the generated possible choices is finished and L is empty, exit.

Example 5: The dependency sets of output cones of the (24, 6, 10) CUT from 1 to 6 (D_j, $1 \leq j \leq 6$) are

D_0 = {0, 1, 3, 4, 8, 9, 10, 13, 16, 22}

D_1 = {0, 2, 3, 5, 6, 8, 11, 14, 17, 23}

D_2 = {1, 2, 4, 5, 7, 9, 12, 15, 18, 22}

D_3 = {0, 1, 2, 6, 7, 10, 11, 12, 19, 23}

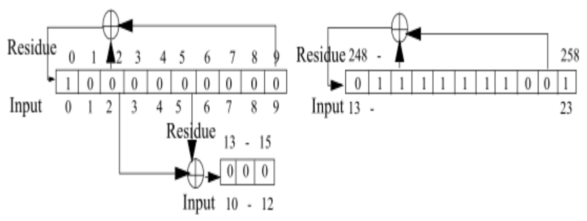
D_4 = {3, 4, 5, 6, 7, 13, 14, 15, 20, 22}

D_5 = {8, 9, 10, 11, 12, 13, 14, 15, 21, 23}

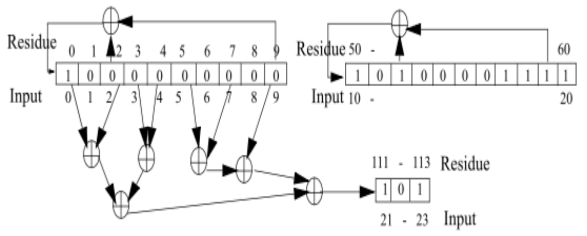
By using algorithm *NEW_CONV*, the first solution is as follows. There are two SR segments, the first one is of size 3, and the other one is of size 11 which is an independent (11, 10) simple LFSR/SR with the same primitive polynomial and different initial seed (01111111001). R_{12} feeds the input of the feed forward stage of the first SR segment to generate R_{13}, $R_{12} = x^2 + x^5 = R_2 + R_5$. This residue needs one XOR gate. The convolved LFSR/SR for this circuit with its initial seed using *NEW_CONV* is shown in Figure 10a. From Figure 10a, the convolved LFSR/SR has three XOR gates and the test set length required is 2^{10}-1 (1023) which is the optimal test set length. This generator misses the all-zero seed so, this generator needs two initial seeds.

The residue assignment according to algorithm in [27-28] will be 0-9, 50-60, 111-113. The upper bound of the number of required XOR gates is 8. The number of terms of R_{110} is 7 ($R_{110} = 1 + x^2 + x^3 + x^4 + x^5 + x^7 + x^9 = R_0 + R_2 + R_3 + R_4 + R_5 + R_7 + R_9$). The convolved LFSR/SR for this CUT with its initial seed is shown in Figure 10b.

It is clear from example that the algorithm *NEW_CONV*, presented in this paper to design convolved LFSR/SR, is the best in terms of test set lengths and the hardware overhead compared to other generators [4]. The algorithm produces several solutions which provide the same minimal hardware overhead and the same optimal test set length, some of which are shown in Table 7.



(a) Convolved LFSR/SR using *NEW_CONV*.



(b) Convolved LFSR/SR using TPG.

Fig. 10: Several convolved LFSR/SRs.

Table 7. Different solutions of example 5.

Possible Solution	Applicable residues	XOR gates
1	0-9 13-15 580-590	3
2	0-9 15-17 388-398	3
3	0-9 15-17 658-668	3
4	0-9 15-17 826-836	3
5	0-9 15-17 846-856	3
6	0-9 16-18 680-690	3
7	0-9 16-18 853-863	3
8	0-9 16-18 997-1007	3
9	0-9 16-19 127-136	3
10	0-9 16-19 380-389	3

5. DESIGN OF THE NEW GENERATOR IN THE PET

Let π_0 be an index default permutation of the n inputs of the CUT which signifies the order in which the corresponding flip-flops (stages) of the convolved LFSR/SR are linked. Consider an (n, m, k) CUT along with the notation that input I_i is assigned a unique index (label) i where $0 \leq i \leq n$. The default permutation π_0 of the inputs of the CUT is specified completely by n-tuple (0, 1, 2, 3 ... n - 1), let the set A = {I_0, $I_1, I_{2, \ldots,} I_{n-1}$}, and π_0 = {0, 1, 2, 3 ... n - 1}.

Figure 11 is considered the general scheme of the PETPG. In a convolved LFSR/SR and using the default permutation π_0, inputs 0 through i are assigned to the residues R_0 through R_i, and inputs i + 1 through n - 1 are assigned to the residues R_{i+j} through R_{i+n-2}. In the new generator, it is possible to change the default permutation to another permutation π. In Figure 11 using permutation π, every CUT inputs are assigned to the residues according to permutation π_0 except I_{w+2} assigned R_{w+1}, I_{w+1} assigned R_{w+2}, I_{i+2} assigned R_{i+j}, and I_{i+1} assigned R_{i+j+1}. So, the convolved LFSR/SR is considered a special case of the new generator. The simple LFSR/SR and the permuted LFSR/SR are considered special cases of the new generator as well.

Using a specific primitive polynomial $p(x)$, let β be the set containing all dependency sets of the output cones whose assigned residues are linearly dependent and |β| be the number of the dependency sets in β. In the case of the convolved LFSR/SRs, all dependency sets of the output cones must satisfy condition 1 so that β is an empty set. This restriction can be reduced to get a solution even if β is not an empty set. This is done by a small number of permutations of the assigned residues to the CUT inputs to make β an empty set. The permutation of the residues means the permutation of the CUT inputs assigned to those residues. The permutation of the CUT inputs may be useful in obtaining a PETPG with minimal test set length. The permutation of the inputs results in a routing overhead so the generator synthesis is done prior to the layout phase of the CUT, and that, therefore any rewiring issues are tackled in the overall layout of the CUT and the BIST circuitry as a whole.

5.1 The Search to Obtain Minimum |Β|

The search to get the minimum |β|, referred to as $|\beta|_{\min}$, is carried out either by searching in the candidate primitive polynomials or by using the proposed residue assignment, discussed in section 4 for a specific primitive polynomial. Choosing a small value of $|\beta|_{\min}$ will reduce the required number of permutations that reduces the routing overhead. First, two useful definitions need to be introduced.

Definition 3: An *applicable primitive polynomial for permutation* is the polynomial for which at least (m - $|\beta|_{\min}$) dependency sets of the output cones satisfy condition 1 before permutation.

Definition 4: The *applicable residues for permutation* are produced by the residue assignment for the CUT inputs before permutation such that at least (m - $|\beta|_{\min}$) dependency sets of the output cones satisfy condition 1.

The algorithm to design the new generator consists of two phases. The first phase is dedicated to search from the candidate primitive polynomials to find the applicable primitive polynomial for permutation and, with a small number of permutations; the set β may be an empty set. The generator resulting from this phase is the permuted LFSR/SR (the simple LFSR/SR with permutation π). In the second phase, the applicable residues for permutation can be found using the proposed residue assignment, discussed in section 4 and, with a small number of permutations; the set β may be an empty set. The generator resulting from this phase is the new generator (convolved LFSR/SR with permutation). A combination of changing the assigned residues with a small number of permutations may increase the chance of obtaining a solution for the case where the convolved LFSR/SR cannot get a solution with the minimal test set length and the minimal hardware overhead (i.e, reduce the required number of XOR gates).

For simplicity, the algorithm divides the length of the generator into two parts. The first part is the LFSR portion with size w and the second part is either a simple LFSR/SR with size n - w (n-$w \geq w$) or shift register with size n - w (n-w < w). The residues of the second part of the generator can be changed according to the residue assignment. Refer to the convolved LFSR/SR in form A shown Figure 8. This division is practically adequate and the experimental results in section 6 will illustrate that situation.

5.2 Idea of the Permutation

In this section, the idea of the permutation of one dependency set, whose assigned residues are linearly dependent, is explained. The permutation approach of all dependency sets, whose assigned residues are linearly dependent, will be explained in section 5.3.

It is known from condition 1, that all residues assigned to the dependency set must be linearly independent. If the residues assigned to the dependency set are linearly dependent, one or more residues are linear combinations of the each other. The idea of the permutation will be demonstrated by an example.

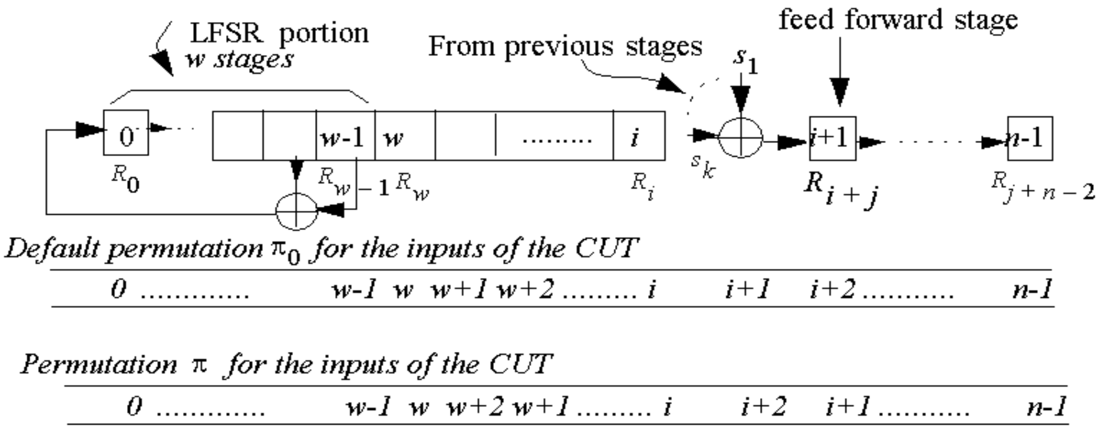

LFSR portion
w stages
From previous stages
feed forward stage
s_1
0
w-1
w
....
i
i+1
n-1
R_0
R_{w-1}
R_w
R_i
s_k
R_{i+j}
R_{j+n-2}
Default permutation π_0 for the inputs of the CUT
0 .... w-1 w w+1 w+2 .... i i+1 i+2 .... n-1
Permutation π for the inputs of the CUT
0 .... w-1 w w+2 w+1 .... i i+2 i+1 .... n-1

Fig. 11: New generator for the PET.

Example 6: The (24, 6, 10) CUT with its dependency sets according to example 5 is considered.

This example consists of two phases. The first phase indicates the solution using the applicable primitive polynomial for permutation. The second phase indicates the solution using the applicable residues for permutation for a specific primitive polynomial.

1. The first phase: The primitive polynomial $p(x)$ with minimum terms, and $|\beta|_{\min}$ is determined. This polynomial $p(x) = 1 + x^2 + x^7 + x^8 + x^{10}$, and $|\beta|_{\min}$ equals 1 so the $p(x)$ is an applicable primitive polynomial for permutation. The dependency set whose assigned residues are linearly dependent, b, is {0 1 2 6 7 10 11 12 19 23}, let π_0 = {0, 1, 2,...., 23}. First, construct set b' = {0, 1, 2} ($b' \subset b$) and check if its assigned residues are linearly independent or not. If the residues are linearly independent, add to b' the next element which is 6 in b ($b' = b' \cup \{6\}$). The set b' is now {0, 1, 2, 6} ($b' \subset b$). If its assigned residues are linearly independent, add to b' the next element. This process is continued until the assigned residue for last added element to b' causes linear dependence with other assigned residues of elements in b'. The residue assigned to last added element is the first residue that causes the linear dependence of b. In this example, this residue is the residue assigned to I_{23} in b (input 23) which is R_{23}. Now, it is required to permute this residue with another residue, assigned to input p such that $p \in \pi_0$ and $p \notin b$. The first choice is the residue assigned to I_{22}. The residues assigned to set b will be linearly independent but this permutation will generate linear dependence for three other dependency sets of the output cones. This permutation is ignored. The residue assigned to I_{21} is then considered but again the residues assigned to set b are linearly dependent. The residue assigned to I_{20} is then considered, followed by I_{18} and so on until the residue assigned to I_{13} is permuted by the residue assigned to I_{23} and all residues assigned to all dependency sets of the output cones are linearly independent. The new input assignment of the permuted LFSR/ SR, π, is {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 23, 14, 15, 16, 17, 18, 19, 20, 21, 22, 13}. In this case, three XOR gates are required, and the required number of permutations is 1. The permuted LFSR/SR for that CUT with its initial seed and new permutation of the CUT inputs, π, is shown in Figure 12.

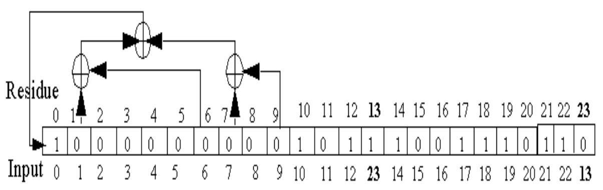


Fig. 12 Permuted LFSR/SR of phase 1.

2. The second phase: Set $|\beta|_{\min}$ to 1 and limit the number of terms of the primitive polynomial to 3 so as to use one XOR gate in the LFSR stages. The number of primitive polynomials of degree 10 with one XOR gates is 2 [4]. Unfortunately, $|\beta|$ of these primitive polynomials are greater than one (greater than $|\beta|_{\min}$). So, it is required to divide the generator into two parts, the first part with length 10 which is LFSR portion and the second part with length 14 which is a (14, 10) simple LFSR/SR (convolved LFSR/SR with form A). The residues of the second part are changed to find $|\beta|_{\min}$ which equals 1. The solution in this case is the primitive polynomial $p(x) = 1+x^7+x^{10}$, and the residues of the generator are: 0-9, 40-53. The residue assignment, 0-9, 40-53, is the applicable residues for permutation. By permuting R_{49} with R_{53}, all dependency sets of the output cones satisfy condition 1 (this permutation is carried out as in the first part). R_{49}, assigned to I_{19}, is now assigned to I_{23} and R_{53}, assigned to I_{23}, is now assigned to I_{19}. The number of required XOR gates for this generator is 2 and the required number of permutations is 1. The generator with its initial seed and permutation of the CUT inputs, π, is shown in Figure 13. The convolved LFSR/SR designed using algorithm *NEW_CONV* requires three XOR gates to test all output cones exhaustively in example 5. The new generator needs just two XOR gates.

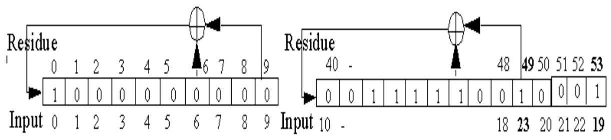


Fig. 13 Novel generator of phase 2.

5.3 Permutation Approach and the Algorithm NEW_GEN

Each phase in the new generator design consists of other two phases. The first one searches for either the applicable primitive polynomial for permutation or the applicable residues for permutation. The second one is the permutation phase. In this section, the permutation approach for all dependency sets whose assigned residues are linearly dependent is presented.

All dependency sets whose assigned residues are linearly dependent are collected in β. (The number of the dependency sets in β is $|\beta|_{min}$.) The first dependency set in β is chosen, referred to as set *b*, and the first assigned residue causing linear dependence in *b* is determined. Let input *s* be the input whose assigned residue is the first residue in *b* causing linear dependence. Let set *b'* ($b' \subset b$) be the set that covers the elements of *b* from the first element to element *s*. Now, it is required to permute the residue assigned to input *s* with the residue assigned to another input, *p*, such that $p \in \pi_0$ and $p \notin b$. The residues assigned to set *b'*, after permuting the residue assigned to input *p*, are checked for linear independence. In the case that the residues assigned to set *b'* are linearly independent and the residues assigned to set *b* after the previous permutation are linearly dependent, store *p* in buffer *pp* as the first permuted element in set *b*. The search to find other permuted elements in *b* to make the residues assigned to *b* linearly independent is repeated.

For example, if $b = \{0, 2, 3, 6, 7\}$, and $\pi_0 = \{0, 1, 2,...., 9\}$. Let the residues assigned to set *b* be linearly dependent, and the first residue in *b* causing linear dependence be the residue assigned to I_6. It is required to permute the residue assigned to I_6 (which is input *s*) with another residue assigned to input *p* such that $p \in \pi_0$ and $p \notin b$. If the residue assigned to I_6 is permuted with the residue assigned to I_8 (which is input *p*) and the residues assigned to set *b'*, which are {0, 2, 3, 8}, are linearly independent and the residues assigned to set *b*, which is {0, 2, 3, 8, 7}, are still linearly dependent after permutation, then store *p*, which is I_8, in buffer *pp* as the first permuted element in set *b*. The first residue, causing linear dependence in *b* after the first permutation, is the residue assigned to I_7 and it is required to permute it with another residue assigned to input *p* such that $p \in \pi_0$ and $p \notin b$ until the residues assigned to set *b* are linearly independent. (Generally, input *p* is considered in the discussion as the permuted element.)

In the case that the residues assigned to *b* are linearly independent, there are three options:

(1) The number of the dependency sets is less than $|\beta|_{min}$. A new β is constructed.

(2) If $|\beta| \geq |\beta|_{min}$ and there are possibilities for other inputs to be permuted, then ignore the last permutation in *b* and try the next possible input.

(3) If $|\beta| \geq |\beta|_{min}$ and there is no possibility for other inputs to be permuted, then ignore all previous permutations applied to *b* and retrieve the stored input from buffer *pp* (first permuted element in set *b*) and select the possible input next to the stored input, let this input be *p*, such that $p \in \pi_0$ and $p \notin b$.

I will state the steps of algorithm *NEW_GEN* to design the new generator for the PET.

Algorithm *NEW_GEN*

Input: The dependency sets of output cones, *w* ($\geq k$), $|\beta|_{min}$, *Q* (number of generated residues), and *S* (number of required solutions).

Output: Applicable residues for the CUT inputs, the initial seed, π, and the required number of permutations (*N_PER*).

1. Phase 1

1.1 Retrieve all primitive polynomials or the subset of all primitive polynomials of degree *w* from a stored file and put them in queue *L*, set buffer *pp* to -1.

1.2 If *L* is not empty, then select the primitive polynomial, *p(x)*, from *L*. If *L* is empty, then go to phase 2.

1.3 Residues from R_0 through R_{n-1} are generated, based on *p(x)*.

1.4 Condition 1 is checked for all dependency sets of the output cones.

1.5 If condition 1 is satisfied for at least (m - $|\beta|_{min}$) dependency sets of the output cones, then the corresponding *p(x)* is an applicable primitive polynomial for permutation and set β is constructed. If *p(x)* is not an applicable *p(x)* for permutation, go to step 1.2.

1.6 Take the first set in β, referred to as *b*, and set buffer *pp* to -1.

1.7 Determine the first residue (assigned to input *s*) in *b* causing linear dependence. Let set *b'* be a subset of *b* containing all elements in *b* from the first element up to input *s*. The residue assigned to input *s* is permuted with the residue assigned to input *p* such that $p \in \pi_0$ and $p \notin b$.

1.8 If the residues assigned to *b'* are linearly dependent, there are two options:

1.8.1 If buffer *pp* equals -1, try next possible input to be permuted. If there is no possibility for other inputs to be permuted, then go to step 1.2.

1.8.2 If buffer *pp* does not equal -1, try next possible input to be permuted. If there is no possibility for other inputs to be permuted, then ignore all previous permutations applied to *b* and retrieve the stored input from buffer *pp* (first permuted element in set *b*) and select the possible input next to the stored input, let this input be *p*, such that $p \in \pi_0$ and $p \notin b$, construct *b'*, set the buffer *pp* to -1 and go to step 1.8. If there is no possible input next to the stored input as a permuted element, then go to step 1.2.

1.9 If the residues assigned to *b'* are linearly independent and the residues assigned to *b* are linearly dependent after the permutation, then store *p* in the buffer *pp* when *p* is the first permuted element (input) in *b* then go to step 1.7.

1.10 If the residues assigned to *b* are linearly independent and the number of linearly dependent sets of the dependency sets is less than $|\beta|_{\min}$ and do not equal zero, then construct the new β, update $|\beta|_{\min}$ with the new β, and go to step 1.6.

1.11 If the residues assigned to *b* are linearly independent, and the number of linearly dependent sets of the dependency sets is greater than or equal to $|\beta|_{\min}$ and there are possibilities for other inputs to be permuted, then ignore the last permutation in *b*, try next possible input, construct *b'*, and go to step 1.8.

1.12 If the residues assigned to *b* are linearly independent, and the number of linearly dependent sets of the dependency sets of the output cones is greater than or equal to $|\beta|_{\min}$ and there is no possibility for other inputs to be permuted, then ignore all previous permutations applied to *b* and retrieve the stored input from buffer *pp* (first permuted element in set *b*) and select the possible input next to the stored input, let this input be *p*, such that $p \in \pi_0$ and $p \notin b$, construct *b'*, set buffer *pp* to -1 and go to step 1.8. If there is no possible input next to the stored input as a permuted element (or buffer *pp* equals -1), then go to step 1.2.

1.13. If β is an empty set, print the applicable residues for the inputs, π, and store it as a solution and determine the initial seed of the generator.

1.14 If the number of solutions is less than *S* and there are other primitive polynomials in *L*, select the next primitive polynomial in *L* and go to step 1.3. If the number of solutions is equal to *S*, exit.

1.15 If *L* is empty and no applicable primitive polynomial for permutation exists, go to phase 2.

2. Phase 2

2.1 Select a primitive polynomial with minimal terms from *L*, and generate all required residues (R_0 through R_{Q-1}).

2.2 Using the residue assignment discussed in section 4, find the applicable residues for permutation.

2.3 Construct set β.

2.4 Repeat steps 1.6 through 1.13. (In step 1.8 and step 1.12 of phase 1, there is possibility to branch to step 1.2 but this step in phase 2 will branch to 2.1 when *L* is not empty)

2.5 If the number of solutions equals *S*, exit.

2.6 If the complete set of the generated possible choices discussed in section 4 is finished and *L* is not empty, go to step 2.1.

2.7 If the complete set of the generated possible choices discussed in section 4 is finished and *L* is empty, exit.

6. EXPERIMENTAL RESULTS

Using algorithm *NEW_CONV*, algorithm *NEW_GEN*, new PETPG are designed for all combinational benchmark circuits (labelled as *ckt* in the first column of Table 8 and Table 9) in [1], after they had been segmented using the new algorithm presented in [30]. Table 8 presents the comparison between the design of the convolved LFSR/SR based on the algorithm *NEW_CONV* in this paper and the algorithm in [27-28] for the segmented benchmark circuits with a cone size reduction, *l*, of 16, 20, 24, and 28 inputs. In addition, Table 9 presents the design of the new generators for the segmented benchmark circuits with a cone size reduction, *l*, of 16, 20, 24, and 28 inputs. (After segmenting the circuit with the segmentation cells, the resulting *k* for the segmented circuits may be less than *l*. The difference between *l* and *k*, obtained after segmenting, is due to the nature of the circuit [16, 30].) The first two columns of Table 8 and Table 9 provide the characteristics of the segmented circuits. The exponent terms for the primitive polynomial, *p(x)*, is given in the third column of Table 8 and Table 9. The applicable residues for the stages of the PETPG, the total number of required XOR gates to realize the PETPG, and the run-time in seconds on a SUN Sparc II workstation are shown in the fourth and fifth column of Table 8 and Table 9, respectively. The required number of permutations (*N_PER*) is given in the sixth column Table 9. For example in Table 9, for the segmented c432 circuit in the first row, the primitive polynomial, *p(x)*, is $1 + x^3 + x^4 + x^5 + x^{16}$. Stages 0 through 62 have residues R_0 through R_{62}, respectively. The total number of XOR gates required to realize the new PETPG is 3 in less than one second with number of permutations four. Referring to Table 8, the corresponding design of the convolved LFSR/SR requires 6 XOR gates in 2 seconds for the algorithm *NEW_CONV* and 14 XOR gates in less than one second for the algorithm in [27-28].

The number of permutations in the last column of Table 9, referred to *N_PER*, is calculated as follows. Let π_0 be {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}, and π be {1, 0, 2, 3, 4, 9, 6, 7, 8, 5}. Then, α is {1, 1, 0, 0, 0, 1, 0, 0, 0, 1}. The elements in the set α are either 0 or 1. If element *i* of π_0 equals element *i* of π, then element *i* of α = 0, if the elements are not equal then element *i* of α = 1. The number of the permutations in π equals the number of 1's in α divided by 2 which equals 2 in this case.

From Table 8 and Table 9, note the following:

1. The test set lengths designed for all combinational benchmark circuits are optimal (minimal) test set lengths (the degree of *p(x)*, *w*, equals *k*) without requiring the insertion of segmentation cells. From the results in [26], additional segmentation cells are required to find an applicable primitive polynomial.

2. The hardware overhead (XOR gates) for *NEW_CONV* is always less than that required by algorithm in [27-28] for the same primitive polynomial and Q. In every case, *NEW_CONV* produces several possible solutions which increase the chance of obtaining a result with minimal hardware overhead.

3. The algorithm *NEW_CONV* obtains a solution in cases where, for the same *p*(*x*) and *Q*, the algorithm in [27-28] cannot obtain any solution of the optimal test set length. Two cases (in c499, and c1355) do not provide any solution after two days run-time whereas algorithm *NEW_CONV* does in reasonable time for the optimal test set length.

4. The same applicable residues with the same hardware overhead was obtained by the algorithm *NEW_CONV* and the algorithm in [27-28] for c1908 and using $p(x) = 1 + x^7 + x^{20}$, and $Q = 500$. But the run-time using *NEW_CONV* was 613 seconds while the algorithm in [27-28] requires 5832 seconds. The hardware overhead of the algorithm *NEW_CONV* and the algorithm in [27-28] for c1908 is the same, but the run-time

using *NEW_CONV* is faster than the run-time using the algorithm in [27-28].

5. Comparing the number of XOR gates and the required run-time between the convolved LFSR/SR using *NEW_CONV* in Table 8 and the new generator in Table 9 indicates the efficiency of the new generator where the required hardware overhead and the run-time are always the least. In every case, there are several possible solutions which increase the chance of obtaining a result with minimal hardware overhead.

6. The number of required permutations of the CUT inputs is small compared to the length of the generator, *n*.

7. The new generator has found solutions in cases where the convolved LFSR/SRs based on the algorithm in [27-28] could not find a solution in reasonable time. In c5315 and c7552, the algorithm in [27-28] does not provide any solution when k = 16 for any primitive polynomial of degree 16 (the algorithm in [27-28] found a solution when w = 17) but the new generator succeeds with low hardware overhead. All results in Table 9 prove the efficiency of the new generator where a solution in all cases is achieved. Table 9 demonstrates the potential of the new generator approach, generating optimal (minimal) test set lengths with low hardware overhead for the segmented combinational benchmark circuits.

7. CONCLUSION

An efficient new algorithm to design convolved LFSR/SRs so as to generate the applicable residues to exhaustively test all output cones with low hardware overhead has been introduced and demonstrated. With no restriction in the size of the SR segment and with restriction in the number of FF stages, a significant reduction in the number of XOR gates is achieved. The success of the approach is indicated in the experimental results where the generation process of possible choices enables us to produce several possible solutions from which a minimal hardware solution is chosen.

In addition, a new PETPG, that bridges the gap between the convolved LFSR/SR and the permuted LFSR/SR, has been introduced and demonstrated. An efficient heuristic search approach has been presented. It permutes the CUT inputs after the applicable residues for permutation become available. The changing residues assigned to the CUT inputs followed by permutation gives the new generator a special character in that the convolved LFSR/SR and the permuted LFSR/SR are special cases of the new generator. The approach is successful because: (1) The test set lengths designed for all combinational benchmark circuits are optimal (minimal) test set lengths without requiring additional segmentation cells. (2) The required hardware overhead and the run-time are always less than that produced by the convolved LFSR/SR using *NEW_CONV* and the algorithm in [27-28]. (3) The number of required permutations of the CUT inputs is small compared to the length of the generator, *n*. (4) The new generator has found solutions for those cases where the convolved LFSR/SRs using the algorithm in [27-28] could not find a solution.

Finally, the practical results indicate that a generator with minimal test set length, minimal hardware overhead, minimum routing overhead, and good performance can be obtained.

Table 8. Results of the new convolved LFSR/SR for segmented benchmark circuits

Ckt	*(n,m',k)*	*p(x)*	*Applicable Residues*	*XOR / time*	*Applicable Residues*	*XOR /time*
c432	(63, 19, 16)	16 5 4 3 0	0-27 64-98	6* / 2 sec.	0-40 48-67 69-70	14 / < 1 sec.
		16 6 4 1 0	0-34 64-91	6* / 4 sec.	0-40 44-61 63-66	14 / < 1 sec.
		16 15 12 10 0	0-33 88-116	6* / 4 sec.	0-36 40-55 61-70	12 / < 1 sec.
		16 15 13 4 0	0-29 52-84	6* / 3 sec.	0-41 50-68 70-71	10 / < 1 sec.
	(55, 26, 20)	20 19 4 3 0	0-19 21-31 116-139	9* / 4 sec.	0-20 23-55 58	17 / < 1 sec.
		20 3 0	0-33 216-236	2* / 4 sec.	0-37 243-258	7 / < 1 sec.
		20 9 5 1 0	0-19 64-98	6* / < 1 sec.	0-36 110-127	11 / 1 sec.
		20 17 9 7 0	0-19 27-61	6* / < 1 sec.	0-46 59-66	12 / < 1 sec.
	(51, 22, 24)	24 4 3 1 0	0-25 214-238	6* / 1 sec.	0-33 135-151	12 / < 1 sec.
	(47, 18, 28)	28 3 0	0-27 177-195	4* / < 1 sec.	0-29 380-396	10 / 1 sec.
c499	(49, 40, 14)	14 9 8 3 0	0-13 22-40 46-61	9* / 16 sec	0-23 55-69 81-90	13 / 2 sec
c1355		14 13 11 4 0	0-13 18-36 169-184	9* / 21 sec	0-21 69-83 139-150	12 / 25 sec
		14 11 7 1 0	0-13 15-32 91-107	9* / 12 sec	0-22 26-41 87-96	12 / 1 sec
		14 13 3 2 0	0-13 18-32 57-76	9* / 10 sec	0-16 23-39 72-86	9* / 24 sec
	(48, 39, 22)	22 11 2 1 0	0-21 185-192 264-281	17* / 646 sec.	no results after two days run	-
		22 15 12 9 0	0-21 82230-82255	6*/ 1359 sec.	no results after two days run	-

Ckt	*(n,m',k)*	*p(x)*	*Applicable Residues*	*XOR / time*	*Applicable Residues*	*XOR /time*
c880	(71, 28, 16)	16 5 4 3 0	0-15 49-82 135-155	9* / 287 sec.	0-15 17-39 47-64 124-137	17 / 1827 sec.
		16 6 4 1 0	0-15 46-76 428-451	9* / 219 sec.	0-32 51-76 393-404	16 / 2 sec.
		16 15 12 10 0	0-45 216-240	6* / 14 sec.	0-46 70-92 95	11 / 1 sec.
		16 15 13 4 0	0-15 23-54 179-201	9* / 95 sec.	0-38 50-69 209-220	10 / 956 sec.
	(70, 28, 17)	17 16 3 2 0	0-16 37-70 268-286	9* / 254 sec.	0-34 51-76 100-108	11 / < 1 sec.
		17 3 0	0-16 110-143 214-232	3* / 569 sec.	0-23 117-151 397-407	10 / 3 sec.
		17 5 0	0-16 86-121 214-230	3* / 395 sec.	0-26 90-123 186-194	9 / 1 sec.
		17 6 0	0-16 91-120 180-202	3* / 442 sec.	0-25 89-122 197-206	10 / 1 sec.
	(69, 28, 24)	24 4 3 1 0	0-23 29-48 59-83	9* / 36 sec.	0-23 29-68 130-134	17 / 1 sec.
	(68, 24, 28)	28 3 0	0-27 343-382	2* / < 1 sec.	0-27 308-338 410-418	14 / 28 sec.
c1908	(50, 27, 16)	16 5 4 3 0	0-15 220-235 446-463	9* / 594 sec.	0-18 326-341 364-378	11 / 1326 sec.
		16 11 3 2 0	0-15 170-187 231-246	9* / 214 sec.	0-17 149-164 219-234	9* / 831 sec.
		16 11 6 5 0	0-15 96-111 202-219	9* / 123 sec.	0-15 96-111 202-219	9* / 1839 sec.
	(44, 27, 20)	20 17 0	0-19 316-330 449-457	15* / 613 sec.	0-19 316-330 449-457	15* / 5832 sec.
		20 6 5 3 0	0-19 345-355 437-449	17* / 906 sec.	0-19 263-273 291-301 315-316	20 / 16072 sec.
		20 9 5 1 0	0-19 48-57 224-237	15* / 100 sec.	0-19 29-39 312-321 441-443	16 / 4583 sec.
		20 10 5 1 0	0-19 167-177 364-376	14* / 439 sec.	0-19 100-110 472-482 492-493	19 / 37994 sec.
	(42, 28, 21)	21 5 2 1 0	0-20 183-192 466-476	14* / 520 sec.	0-21 168-177 280-289	15 / 21436 sec.
	(40, 11, 28)	28 3 0	0-27 58 332-342	7* / 3 sec.	0-27 337-347 362	9 / 3 sec.
c2670	(265, 42, 16)	16 5 4 3 0	0-15 18-217 390-438	9* / 177 sec.	0-220 223-243 257-274 279-283	17 / 1 sec.
		16 6 4 1 0	0-215 344-392	6* / 56 sec.	0-237 272-289 306-314	13 / 2 sec.
		16 15 12 10 0	0-15 17-210 311-365	9* / 151 sec.	0-238 340-357 359-366	14 / 8 sec.
	(262, 40, 20)	20 9 5 1 0	0-80 88-268	6* / 46 sec.	0-209 211-260 264-265	16 / 1 sec.
		20 15 5 4 0	0-203 207-264	6* / 171 sec.	0-242 266-284	16 / 2 sec.
		20 15 7 4 0	0-87 115-288	6* / 57 sec.	0-238 261-282 285	13 / 2 sec.
		20 15 8 7 0	0-205 207-262	6* / 164 sec.	0-242 249-267	10 / 1 sec.
		20 15 9 2 0	0-206 232-286	6* / 167 sec.	0-246 253-267	13 / < 1 sec.
	(256, 33, 24)	24 4 3 1 0	0-213 404-445	6* / 240 sec.	0-223 240-266 270-274	21 / 2 sec.
	(254, 31, 27)	27 8 7 1 0	0-26 29-210 240-284	9* / 830 sec.	0-204 214-242 329-348	16 / 12 sec.
c3540	(128, 61, 16)	16 11 9 7 0	0-15 126-217 463-482	9* / 944 sec.	0-24 26-41 43-66 76-112 310-332 412-414	19 / 811 sec.
		16 12 6 1 0	0-15 152-176 243-329	9* / 812 sec.	0-23 28-77 81-103 106-132 192-195	19 / 5 sec.
	(107, 50, 20)	20 19 4 3 0	0-19 3908-3994	6* / 9 sec.	0-22 48-71 86-117 191-212 221-226	18 / 8 sec.
	(94, 40, 24)	24 4 3 1 0	0-23 83-106 135-180	9* / 76 sec.	0-24 82-106 109-133 138-156	18 / 1 sec.
	(82, 31, 28)	28 3 0	0-27 577-601 2556-2584	8* / 19396 sec.	0-27 739-766 1319-1344	12 / 100 sec.
c5315	(215, 76, 20)	20 6 5 3 0	0-19 156-282 424-491	9* / 9399 sec.	0-146 148-169 212-235 281-300 357-358	22 / 63 sec.
		20 9 5 1 0	0-19 103-268 300-328	9* / 7238 sec.	0-171 199-218 248-268 370 371	19 / 2376 sec.
		20 19 4 3 0	0-146 12488-12555	6* / 10612 sec.	0-147 150-176 234-256 279-295	18 / 17 sec.

Ckt	*(n,m',k)*	*p(x)*	*Applicable Residues*	*XOR / time*	*Applicable Residues*	*XOR /time*
	(204, 79, 24)	24 8 5 2 0	0-23 26-141 182-245	9* / 433 sec.	0-160 163-189 324-339	19 / 245 sec.
	(194, 68, 28)	28 6 4 1 0	0-142 448-498	6* / 108 sec.	0-142 175-207 401-418	15 / 823 sec.
c6288	(118, 36,16)	16 10 9 6 0	0-15 210-241 437-506	9* / 270 sec.	0-22 416-433 447-466 531-549 560-578 580-598	18 / 61 sec.
	(87, 25, 20)	20 6 5 3 0	0-19 669-705 917-946	9* / 1111 sec.	0-19 233-253 296-330 342-352	16 / 7 sec.
	(65, 25, 24)	24 4 3 1 0	0-23 96-112 1619-1642	10* / 198 sec.	0-23 96-118 752-769	19 / 20 sec.
	(47, 8, 28)	28 3 0	0-27 1078-1096	6 / 2 sec.	0-27 428-446	9 / 2 sec.
c7552	(274, 45, 20)	20 19 4 3 0	0-79 12609-12802	6* / 2383 sec.	0-172 180-209 213-260 295-315 318-319	21 / 4 sec.
		20 6 4 1 0	0-79 987-1180	6* / 61 sec.	0-167 169-189 224-281 322-341 377-383	21 / 9 sec.
	(261, 34, 24)	24 7 2 1 0	0-23 163-399	6* / 1 sec.	0-177 184-214 221-260 264-275	21 / 4 sec.
	(255, 38, 28)	28 9 5 1 0	0-27 127-353	6* / < 1 sec.	0-202 211-240 259-280	15 / 5 sec.

Table 9: Results of the new generator for segmented benchmark circuits

Ckt	*(n,m',k)*	*p(x)*	*Applicable Residues*	*XOR / time*	*N_PER*
c432	(63, 19, 16)	16 5 4 3 0	0-62	3 / < 1 sec.	4
		16 6 4 1 0	0-62	3 / < 1 sec.	4
		16 15 12 10 0	0-62	3 / < 1 sec.	2
		16 15 13 4 0	0-62	3 / < 1 sec.	4
	(55, 26, 20)	20 19 4 3 0	0-19 27-61	6 / < 1 sec.	4
		20 3 0	0 - 19 151 - 185	2 / 1 sec.	6
		20 9 5 1 0	0 - 19 21 - 55	6 / < 1 sec.	2
		20 17 9 7 0	0-54	3 / < 1 sec.	1
	(51, 22, 24)	24 4 3 1 0	0 - 23 208 - 234	6 / 1 sec.	2
	(47, 18, 28)	28 3 0	0 - 27 53 - 71	2 / 12 sec.	6.5
c499 c1355	(49, 40, 14)	14 9 8 3 0	0 - 13 43 - 77	6 / 1 sec.	2
		14 13 11 4 0	0 - 13 156 - 190	6 / 3 sec.	2
		14 11 7 1 0	0 - 13 24 - 58	6 / 2 sec.	2
		14 13 3 2 0	0 - 13 20 - 54	6 / 1 sec.	3
	(48, 39, 22)	22 11 2 1 0	0 - 48	3 / 1 sec.	3.5
		22 15 12 9 0	0 - 21 12689 - 12714	6 / 9193 sec.	1
c880	(71, 28, 16)	16 5 4 3 0	0 - 15 62 - 116	6 / 3 sec.	4
		16 6 4 1 0	0 - 15 46 - 100	6 / < 1 sec.	5
		16 15 12 10 0	0 - 15 27 - 81	6 / < 1 sec.	3
		16 15 13 4 0	0 - 70	3 / < 1 sec.	5.5
	(70, 28, 17)	17 16 3 2 0	0 - 16 42 - 94	6 / < 1 sec.	5.5
		17 3 0	0 - 16 97 - 149	2 / 9 sec.	6
		17 5 0	0 - 16 54 - 106	2 / 7 sec.	7
		17 6 0	0 - 16 71 - 123	2 / 4 sec.	7.5
	(69, 28, 24)	24 4 3 1 0	0 - 23 25 - 69	6 / < 1 sec.	5.5
	(68, 24, 28)	28 3 0	0 - 27 146 - 185	2 / 9 sec.	3
c1908	(50, 27, 16)	16 5 4 3 0	0 - 15 508 - 541	6 / 108 sec.	4
		16 11 3 2 0	0 - 15 570 - 603	6 / 295 sec.	6.5
		16 11 6 5 0	0 - 15 6637 - 6670	6 / 2635 sec.	5.5
	(44, 27, 20)	20 17 0	0 - 19 2507 - 2530	2 / 1923 sec.	2.5
		20 6 5 3 0	0 - 19 714 - 737	6 / 400 sec.	6

Ckt	*(n,m',k)*	*p(x)*	*Applicable Residues*	*XOR / time*	*N_PER*
		20 9 5 1 0	0 - 19 1668 - 1691	6 / 1159 sec	3
		20 10 5 1 0	0 - 19 107 - 130	6 / 65 sec.	3.5
	(42, 28, 21)	21 5 2 1 0	0 - 20 566 - 586	6 / 121 sec.	1
	(40, 11, 28)	28 3 0	0 - 27 471 - 482	3 / 34 sec.	9.5
c2670	(265, 42, 16)	16 5 4 3 0	0-264	3 / < 1 sec.	3
		16 6 4 1 0	0-264	3 / < 1 sec.	9
		16 15 12 10 0	0 - 15 17 - 265	6 / 341 sec.	5
	(262, 40, 20)	20 9 5 1 0	0 - 261	3 / < 1 sec.	2
		20 15 5 4 0	0 - 261	3 / < 1 sec.	1
		20 15 7 4 0	0 - 261	3 / < 1 sec.	4
		20 15 8 7 0	0 - 261	3 / 1 sec.	3
		20 15 9 2 0	0 - 261	3 / 1 sec.	3
	(256, 33, 24)	24 4 3 1 0	0 - 255	3 / < 1 sec.	4.5
	(254, 31, 27)	27 8 7 1 0	0 - 26 29 - 255	6 / 1159 sec.	7
c3540	(128, 61, 16)	16 11 9 7 0	0-127	3 / 2 sec.	6
		16 12 6 1 0	0-127	3 / 1 sec.	6.5
	(107, 50, 20)	20 19 4 3 0	0-106	3 / 20 sec.	9.5
	(94, 40, 24)	24 4 3 1 0	0-23 30-99	6 / 56 sec.	11
	(82, 31, 28)	28 3 0	0-27 310-363	2 / 1273 sec	14.5
c5315	(215, 76, 20)	20 6 5 3 0	0 - 15 47 - 258	6 / 12087 sec.	13.5
		20 9 5 1 0	0 - 19 27 - 221	6 / 1634 sec.	11
		20 19 4 3 0	0 - 19 22 - 216	6 / 576 sec.	7.5
	(204, 79, 24)	24 8 5 2 0	0 - 23 25 - 204	6 / 177 sec.	5.5
	(194, 68, 28)	28 6 4 1 0	0 - 193	3 / 37 sec.	14
c6288	(118, 36,16)	16 10 9 6 0	0 - 117	3 / 3 sec.	11.5
	(87, 25, 20)	20 6 5 3 0	0 - 19 57 - 123	6 / 44 sec.	7.5
	(65, 25, 24)	24 4 3 1 0	0 - 23 94 - 134	6 / 5 sec.	4.5
	(47, 8, 28)	28 3 0	0 - 27 526 - 544	4 / 3 sec.	3
c7552	(274, 45, 20)	20 19 4 3 0	0 - 304	3 / 8 sec.	21.5
		20 6 4 1 0	0 - 273	3 / 4 sec.	6
	(261, 34, 24)	24 7 2 1 0	0 - 260	3 / 8 sec.	4
	(255, 38, 28)	28 9 5 1 0	0 - 254	3 / 7 sec.	4

8. REFERENCES

[1] F. Brglez and H. Fujiwara. 1985. A neutral netlist on ten combinational benchmark circuits and a target translator in FORTRAN. In Proceedings of the International Symposium on circuits and systems (June, 1985).

[2] Mohamed H. El-Mahlawy. 1995. Automatic Measurement of Digital Circuits. M.Sc. Thesis. Military Technical College, Egypt.

[3] M. El Said Gohniemy, S. Fadel Bahgat, Mohamed H. El-Mahlawy, and E. E. M. Zouelfoukkar. 1996. A Novel Microcomputer Based Digital Automatic Testing Equipment using Signature Analysis. In Proceedings of the IEEE conference on Industrial Applications in Power Systems Computer Science and Telecommunications (13-16 May 1996), 140-144.

[4] M. H. El-Mahlawy. 2000. Pseudo-Exhaustive Built-In Self-Test for Boundary Scan. Ph.D. Thesis. Kent University, U.K.

[5] Paul H. Bardell, Willian H. McAnney, Jacob Savir. 1987. Built-In test for VLSI: pseudorandom techniques. John Wiley and Sons.

[6] S. W. Golomb. 1982. Shift Register Sequences. Laguna Hills CA: Aegean Park Press.

[7] Mohamed H. El-Mahlawy. 2015. Signature Multi-Mode Hardware-Based Self-Test Architecture for Digital Integrated Circuits. In Proceedings of the IEEE International Conference on Electronics, Circuits, & Systems (6-9 Dec. 2015), 437-441.

[8] Sherif I. Morsy, Mohamed H. El-Mahlawy, Gouda I. Mohamed. 2013. Hybrid based Self-Test Solution for Embedded System on Chip. International Journal of Computer Applications, Vol. 84, No. 12, (Dec. 2013), 7-14.

[9] Mohamed H. El-Mahlawy and A. Seddik. 2007. Design and Implementation of New Automatic Testing System for Digital Circuits Based on the Signature Analysis. In Proceedings of the 12th International Conference on Aerospace Sciences & Aviation Technology (ASAT-12) (May 2007), CRS-9-1 - CRS-9-12, Egypt.

[10] Mohamed H. El-Mahlawy, A. Abd El-Wahab, and A.S. Ragab. 2008. FPGA Implementation of The Portable Automatic Testing System for Digital Circuits. In Proceedings of the 6th International Conference of the Electrical Engineering (ICEENG-6) (May 2008), EE126-1 - EE126-24, Egypt.

[11] Parag K. Lala. 1997. Digital circuit testing and testability. Academic Press.

[12] Angela Krstic, and Kwang-Ting (Tim) Cheng. 1998. Delay Fault Testing for VLSI Circuits. Kluwer Academic Publishers.

[13] Mukund Sivaraman, and Andrzej J. Strojwas. 1998. A unified Approach for Timing Verification and delay Fault Testing. Kluwer Academic Publishers.

[14] Alexander Miczo. 2003. Digital Logic Testing and Simulation. John Wiley & Sons.

[15] Mohamed H. El-Mahlawy, and Winston Waller. 2000. An efficient algorithm to design convolved LFSR/SR. In Proceedings of the 17th National Radio Science Conference (22-24 Feb. 2000), C23 (1-10), Egypt.

[16] Mohamed H. El-Mahlawy, and Winston Waller. 2000. An efficient algorithm to partition the combinational circuits for pseudoexhaustive testing. In Proceedings of the 17th National Radio Science Conference, (22-24 Feb. 2000), C24 (1-11), Egypt.

[17] E. J. McCluskey and S. Bozorgui-Nesbat. 1981. Design for autonomous test. IEEE transaction on computers Vol. C-30, No. 11, (Nov. 1981), 866-875.

[18] E. J. McCluskey. 1984. Verification testing-A pseudoexhaustive test technique. IEEE transaction on computers Vol. C-33, No. 6 (June 1984), 541-546.

[19] Zeev Barzilai, Jacob Savir, George Markowsky, and Merlin G. Smith. 1981. The weighted syndrome sums approach to VLSI testing. IEEE Transactions on Computers, Vol. C-30, No. 12 (Dec. 1981), 996-1000.

[20] D. T. Tang and L. S. Woo. 1983. Exhaustive test pattern generation with constant weight vectors. IEEE transaction on computers, Vol. C-32, No. 12 (Dec. 1983), 1145-1150.

[21] L. -T. Wang and E. J. McCluskey. 1986. Condensed linear feedfack shift register (LFSR) testing - A pseudo-exhaustive test technique. IEEE transaction on computers Vol. C-35, No. 4 (April 1986), 367-370.

[22] L. -T. Wang and E. J. McCluskey. 1988. Circuits for pseudoexhaustive test pattern generation. IEEE transaction on computer-aided design, Vol. 7, No. 10, (Oct. 1988) 1068- 1080.

[23] S. B. Akers. 1985. On the use of linear sums in exhaustive testing. Digest of papers, 15th Annual International on Fault Tolerant Computing Symposium (1985), 148-153.

[24] N. Vasanthavada, P. N. Marinos. 1985. An operationally efficient scheme for exhaustive test-pattern generation using linear codes. In Proceedings of the International Test Conference (Nov. 1985), 476-482.

[25] Zeev Barzilai, Don Coppersmith, and Arnold L. Rosenberg. 1983. Exhaustive generation of bit patterns with applications to VLSI self-testing. IEEE Transactions on Computers, Vol. C-32, NO. 2 (Feb. 1983), 190-194.

[26] Dimitrios Kagaris and Spyros Tragoudas. 1993. Cost-effective LFSR synthesis for optimal pseudoexhaustive BIST test sets. IEEE transactions on very large scale integration systems, Vol. 1, NO. 4 (Dec. 1993), 526-536.

[27] Srinivasan, R., S. K. Gupta, and M. A. Breuer. 1993. Novel test pattern generators for pseudoexhaustive testing. In Proceedings of the International Test Conference (1993), 1041-1050.

[28] Srinivasan, R., S. K. Gupta, and M. A. Breuer. 2000. Novel test pattern generators for pseudoexhaustive testing. IEEE transaction on computers Vol. 49, No. 11 (Nov. 2000), 1228-1239.

[29] Z. Barzilai, D. Coppersmith, and A. Rosenberg. 1983. Exhaustive Bit Pattern Generation in Discontiguous Positions with Applications to VLSI Testing. IEEE Transaction on Computers, Vol. 32, No. 2 (Feb. 1983), 190-194.

[30] Mohamed H. El-Mahlawy, and Winston Waller. 2004. A New Segmentation Approach for Pseudoexhaustive Testing of Combinational Circuits. In Proceeding of the 4th International Conference of the Electrical Engineering (ICEENG-4) (Nov. 2004), 251-265, Egypt.