A Tree-Structured LFSR Synthesis Scheme for Pseudo-Exhaustive Testing of VLSI Circuits

J. C. Rau, W. B. Jone, S. C. Chang, and Y. L. Wu*

Department of Computer Science and Information Engineering
National Chung-Cheng University
Chiayi, Taiwan R.O.C

Department of Computer Science and Engineering*
The Chinese University of Hong-Kong
Shatin, N.T., Hong Kong

Abstract

This paper presents a new test architecture, called TLS (Tree-LFSR/SR), to more effectively generate pseudo-exhaustive test patterns for both combinational and sequential VLSI circuits. Instead of using a single scan chain, the proposed test architecture routes a scan tree driven by the LFSR to generate all possible input patterns for each output cone. The new test architecture is able to take advantages of both signal sharing and signal reuse. The benefits are: (1) the difficulty of test architecture synthesis can be eased by accelerating the searching process of appropriate residues, and (2) the number of XOR gates to satisfy the pseudo-exhaustive test criterion can be reduced. The TLS test scheme mainly contains three phases: backbone generation, tree growing, and XOR-tree generation. Experimental results obtained by simulating combinational and sequential benchmark circuits are very encouraging.

Key words: VLSI testing, pseudo-exhaustive testing, primitive polynomial, scan chain, tree structure

1 Introduction

Exhaustive testing is performed by applying all possible input combinations to the circuit under test (CUT), and then observing the output responses. Since all input combinations are applied, no fault model is required to estimate the fault coverage. However, test time might be intolerable if the number of circuit inputs is large. Exhaustively testing a circuit by partitioning to reduce the test time is called *pseudo-exhaustive* testing [1]. To achieve the goal of pseudo-exhaustive testing, it is sometimes necessary to partition the CUT by inserting *bypass storage cells* (*bscs*). All *bscs* are transparent in the normal mode and act as pseudo-inputs and pseudo-outputs in the test mode such that each output cone depends on no larger than k (≤ 25, mostly) inputs during the test process. For example, if there is an (n, m, r) circuit where n (m) is the number of inputs (outputs) and r is the maximal dependency of the CUT, then s *bscs* can be inserted into the circuit that becomes $(n + s, m + s, k)$ in the testing mode with $r \geq k$. Basically, *circuit partitioning* methods for pseudo-exhaustive testing can be categorized to two classes: constrained partitioning strategy (CPS) and un-constrained partitioning strategy (UPS). In the case of CPS, the CUT is partitioned into several disjoint segments [2]. However, it is not necessary that the segments are disjoint for UPS [3]-[6]. Strategies to find the minimal number of *bscs* for the purpose of partitioning the given circuit as pseudo-exhaustive testable can be found in [2]-[6].

After circuit partitioning, it is possible to pseudo-exhaustively test the entire circuit using several test sessions. Each test session applies at most 2^k patterns to exhaustively exercise some of the output cones. Hence, the entire test length is the summation of each test length used in all test sessions. Certainly, if the number of test sessions can be reduced to one, then the test length for pseudo-exhaustively testing the modified $(n + s, m + s, k)$ circuit can potentially be minimal, i.e. 2^k. A straightforward method for reducing the number of test sessions to one is to employ an LFSR/SR of degree k [7]. That is, all PIs and *bscs* of the CUT are reconfigured as an LFSR/SR chain (in test mode) of degree k for generating all possible 2^k patterns in one test session. Subject to the complex input-output relations, it is not an easy task to construct a desirable LFSR/SR chain for the CUT. Some variants of LFSR/SR such as LFSR/XOR [8]-[10], reconfigurable LFSR/SR [11], permuted LFSR/SR [11], convolved LFSR/SR [11] have been proposed to alleviate this difficulty. Though encouraging results have been reported using the above approaches, high hardware overhead or very long derivation time for primitive polynomial selection and test architecture synthesis might occur for some circuits.

In this work, we propose a test architecture, called TLS (Tree-LFSR/SR), to more effectively generate all pseudo-exhaustive test patterns in one test session. By taking advantages of both signal sharing and signal reuse, a *scan tree* is employed to implement the pseudo-exhaustive test generator for the CUT, instead of just using a single scan chain. Based on this test architecture, several tree branches which in fact are constructed using *bscs* can share the same residues to satisfy the *linear independency* constraint [7]. The results are then extended to include XOR networks, if necessary, to deal with more complicated input-output relations. An efficient method is also provided to construct the required XOR network which is part of the TLS test architecture. The proposed TLS test architecture has two major advantages: (1) the derivation time for appropriate residues (and thus the test architecture) can be greatly reduced because of the tree configuration, and (2) the number of XOR gates added to meet the pseudo-exhaustive test criterion can be reduced.

2 Background

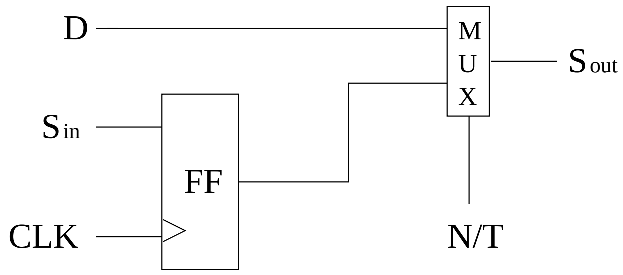

Fig. 1: A bypass storage cell (*bsc*).

Since the test time of exhaustive testing grows exponentially with the number of inputs to the CUT, it may not be practical for circuits with large number of inputs. To reduce test time, the circuit can be tested by pseudo-exhaustive testing and the basic idea is to partition the entire CUT into segments by inserting *bscs*. The architecture of a *bsc* is shown in Fig. 1 where the *bsc* is transparent in the normal mode and acts as a pseudo-input and pseudo-output in the test mode. After circuit partitioning, each output cone can be tested exhaustively. The output cones need not to be disjoint and the size of a cone is defined as the number of its inputs. For example, consider the (6, 2, 6) circuit of Fig. 2(a) [6]. Exhaustively testing the circuit requires 2^6 patterns. However, if the circuit is partitioned by two *bscs* (S_6 and S_7) as shown in Fig. 2(b), each output cone depends on at most three inputs. For example, each of both pseudo-outputs h and g depends on three inputs, while each of primary outputs i and j depends on two peudo-inputs. Thus, the CUT becomes an (8, 4, 3) circuit in the test mode. Here, pseudo-exhaustively testing the circuit merely requires 2^3 patterns to exhaustively test each output cone at the same time.

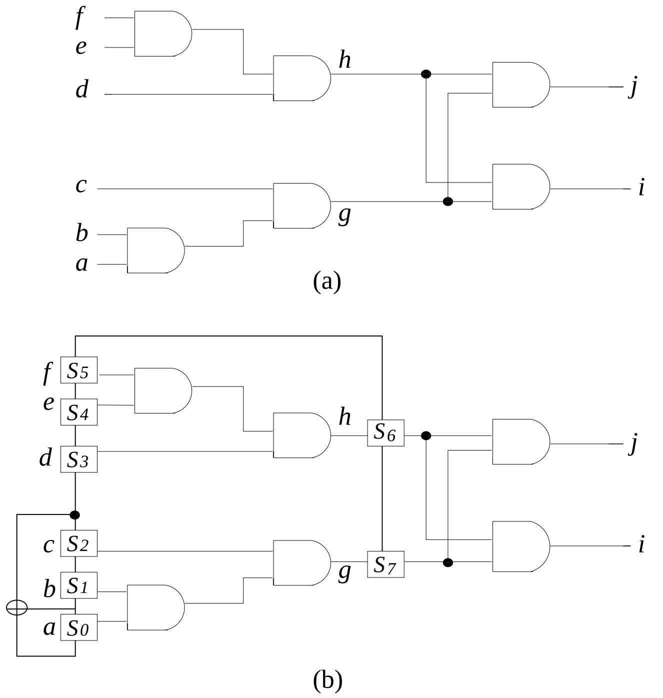

Fig. 2: (a) An example (*6, 2, 6*) circuit.
(b) Circuit partitioning and LFSR/SR test architecture.

Assume we have an $(n,\ m,\ k)$ circuit with the inputs consecutively numbered from 0 to $n-1$. For the example in Fig. 2(b), we have n=8, m=4, k=3, and a is numbered 0. Clearly, k is the minimal number of independent test signals. To construct an LFSR/SR, the first k stages must be configured as an LFSR where a primitive polynomial $P(x)$ of degree k ($P(x)$=x^3+x+1 in Fig. 2(b)) is used as the feedback polynomial. The remaining (n - k) stages are chained as an SR (Fig. 2(b)). The *residue* of stage S_i in the LFSR/SR, denoted by R_i, is defined as $[x^i\ mod\ P(x)]$. Hence, R_0, R_1,..., R_{k-1} are equal to 1, x^1,..., x^{k-1}, respectively. In the SR portion, the linear combinations of the k test signals are generated by the remaining (n - k) stages depending on the stage positions and the feedback polynomial $P(x)$. For S_j in the SR portion, the residue R_j is also defined as $[x^j\ mod\ P(x)]$ and is a unique linear combination of residues R_0 to R_{k-1}. For example, S_6 of Fig. 2(b) has residue R_6 which equals $[x^6\ mod\ P(x)]$. Since R_6 equals x^2+1, it is the linear combination of residues R_0 and R_2. Moreover, an output cone that depends on the inputs $i_1,\ i_2,\ \ldots,\ i_k$ can be exhaustively tested if and only if the residues R_{i_1}, R_{i_2},..., R_{i_k} are *linearly independent* (LI) over $GF(2)$ [7].

3 Backbone Generation

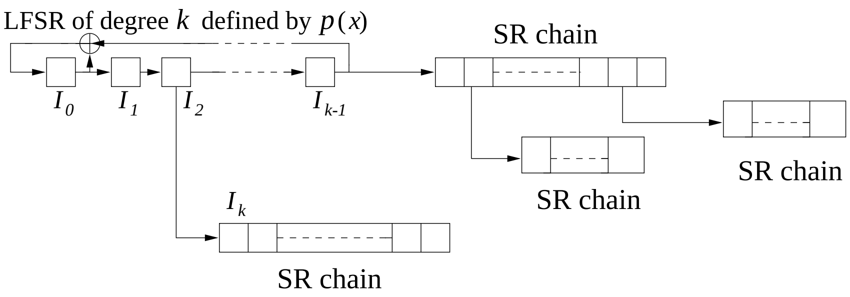

Fig. 3: The TLS test architecture.

As the term "tree-structured" indicates, the proposed test architecture is no longer a single chain consisting of an LFSR followed by an SR chain. On the contrary, it contains many SR chains linked into the LFSR and other SR chains recursively as shown in Fig. 3, and the entire test configuration looks like a tree. The rationale behind the idea of TLS is that, instead of always connecting the *bsc* (or PI) under consideration (BUC) into the tail *bsc* of the LFSR/SR chain, the BUC is allowed to be connected into any *bsc* within the scan tree as shown in Fig. 3. Here, we use the following example to explain the rationale behind the TLS test scheme. Consider the $(10, 5, 4)$ circuit in Fig. 4(a) and the corresponding coordinates of the circuit components in Fig. 4(b). To concisely demonstrate the tree configuration process, the gates and primary outputs have been removed from the circuit, and only primary inputs and *bscs* remain as shown in Fig. 5(a). For pseudo-exhaustively testing the circuit, we can construct a TLS of degree 4 with primitive polynomial $P(x) = x^4 + x^3 + 1$ as shown in Fig. 5(b). It is obvious that i_3 and i_8 share the same test signal, since both of them are the next stage of i_2; that is, i_2 is repeatedly attached. More interestingly, both i_4 and i_9 also have the same residue R_4 under this configuration. With the residues assigned, no LI violation occurs and the entire circuit of Fig. 5(a) can be pseudo-exhaustively tested.

Basically, the backbone generation process of TLS develops a scan chain (backbone) consisting of an LFSR as well as an SR chain attached to the LFSR tail. We assume that the CUT has been coordinated before the backbone generation process can proceed. As shown in Fig. 6, the scan chain travels from top to bottom of the circuit – without routing back – and connects as many *bscs* and PIs as possible. Once a backbone has been established, the isolated *bscs* can be attached to the backbone by growing a set of branches. The backbone and branches jointly will be able to pseudo-exhaustively test the entire circuit, if an appropriate primitive polynomial is employed. Given an (n, m, k) circuit resulting from circuit partitioning and a selected primitive polynomial $P(x)$, the process of backbone construction is to sequentially assign polynomial residues to PIs and *bscs* (see Fig. 6). From now on, unless otherwise stated, we use the term *input* to refer to either PI

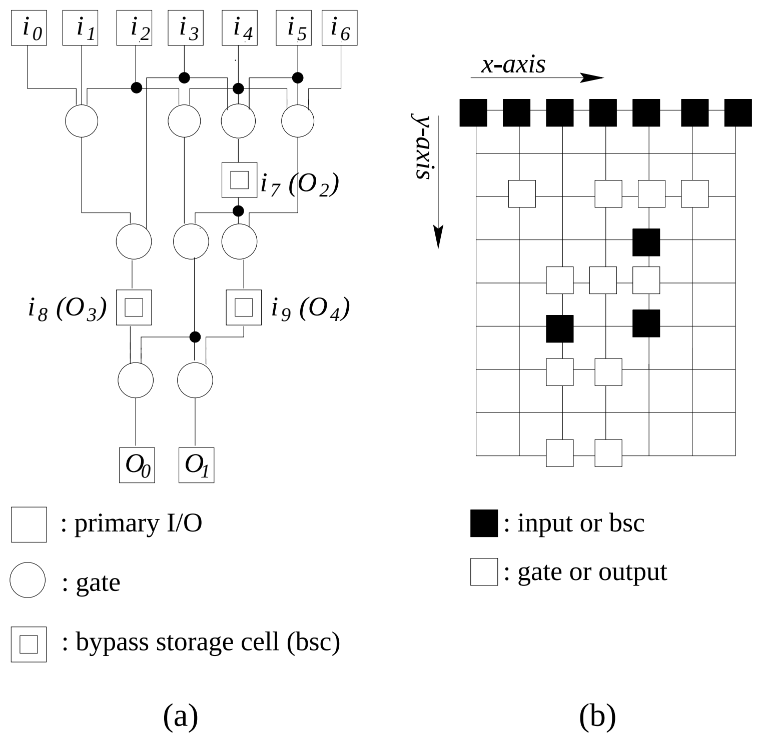

Fig. 4: (a) A ($10, 5, 4$) circuit.
(b) The coordinates of the components in (a).

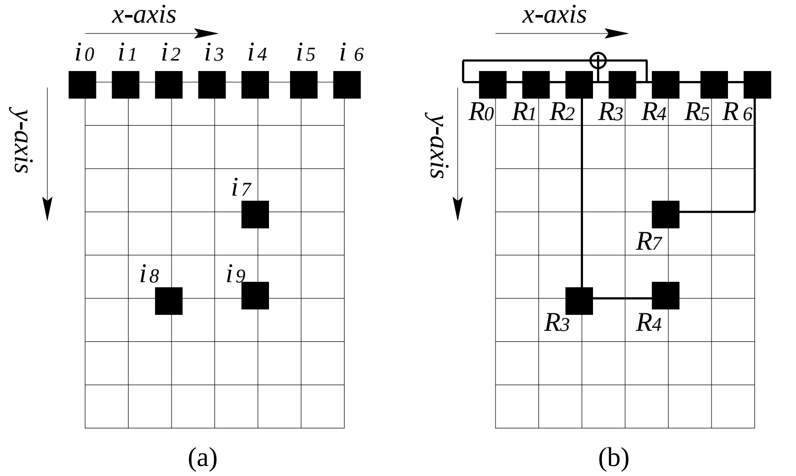

Fig. 5: (a) The simplified version of Fig. 4(b).
(b) Residue assignments and the TLS test architecture for (a).

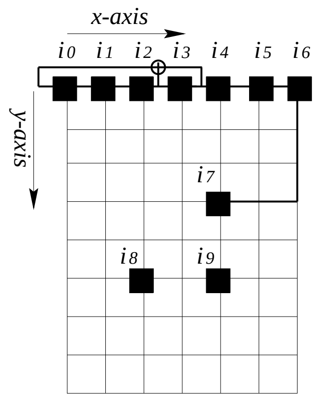

Fig. 6: The backbone of the TLS architecture for Fig. 5(a).

or *bsc*. Without loss of generalization, we also assume that the backbone has developed to input d with residue R_d. By the aid of circuit coordinates, a new residue can be assigned to an unconnected input by connecting d to the nearest unconnected input h which has $y_h \geq y_d$ (thus, the backbone will never route back).

Subject to the input-output relationships, the addition of h to the backbone might not be allowed because of LI violation. If this occurs, then h must be excluded from the backbone regardless of its shortest distance to d. The backbone generation process is resumed by removing h from the candidate list. Each time a legal input h has been found, the backbone is updated by adding h into its tail. The above process is repeated until no more inputs can be included. Some inputs may be left isolated when the backbone generation process terminates because: (1) the nodes violate the LI constraint, or (2) the nodes cannot be traversed by the non-routing-back chain. Note that, the non-routing-back chain is the chain where the Y-coordiate of any stage is less than or equal to the one of its next stage. Here, we also assume that the wire connecting two inputs, d and h, first routes along the Y-axis, then the X-axis.

4 Tree Growing

After the backbone generation process, there may be inputs remaining isolated, i.e., they cannot be assigned proper residues. Thus, it is necessary to adopt additional processes to embed these isolated inputs into the backbone. The basic idea of tree growing is to connect an isolated input g into a stage f of the backbone (Fig. 7(a)) with two objectives: (1) the addition of g as f's next stage will not cause any LI violation, and (2) the routing distance between f and g must be as small as possible. Thus, this process is similar to branch growing from the trunk of a tree. It should be noticed that, instead

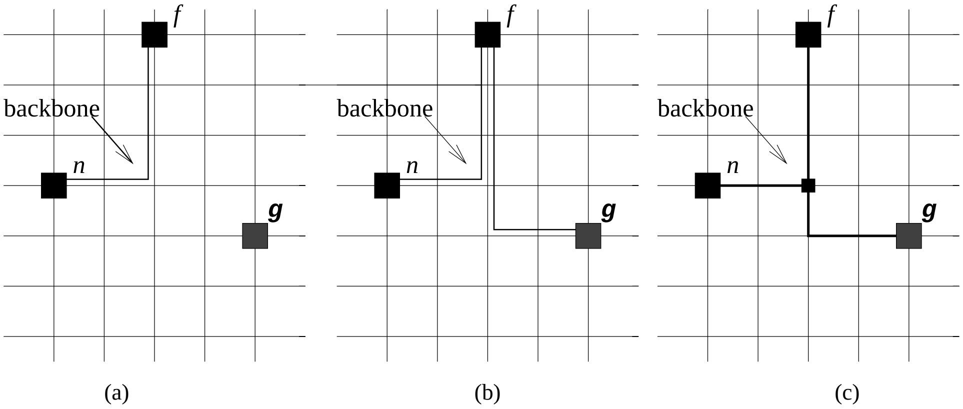

Fig. 7: (a) The backbone and an isolated input g.
(b) Direct connection between g and f.
(c) A more efficient connection between g and f.

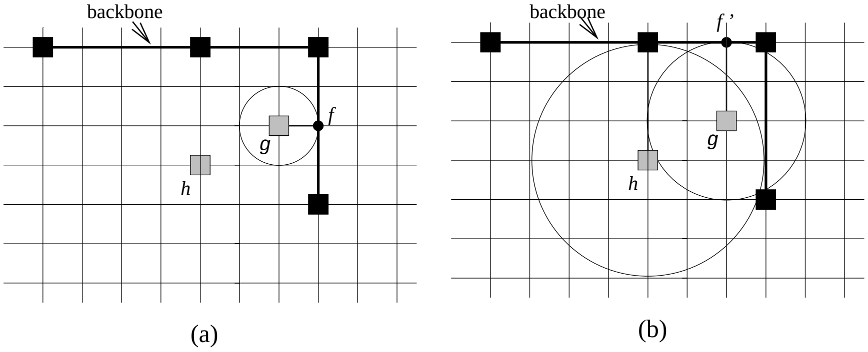

Fig. 8: (a) Hit point determination for g.
(b) The next shortest distance.

of connecting g directly to stage f (Fig. 7(b)), we may attach g to the wire driven by f (Fig. 7(c)) for the purpose of saving routing area.

Initially, the distance from each isolated input to the backbone is calculated. Since all circuit components are placed on a two-dimensional coordinate system filled with grid points, the distance from an input g to the backbone is defined as the shortest *linear* distance from g to a grid point f on the backbone as shown in Fig. 8(a). Additionally, grid point f is said to be *hit* by g. The linear distance between g and h is the summation of absolute differences along X and Y coordinates for nodes g and h. For example, the linear distance between g and h in Fig. 8(a) is 3. The reason of using linear distance is that VLSI wire routing is generally performed along X and Y axes.

After the distance for each isolated input has been derived, the consideration focuses on the one, say g, with shortest distance (than all other isolated inputs) to the backbone at grid point f as shown in Fig. 8(a). The purpose of selecting the shortest-distance (f, g) pair is to keep the tree branch as short as possible. If the component represented by grid point f is with residue R_j, then g has residue R_{j+1} if it is connected to f. The tree growing process does not have the non-routing-back limitation, therefore, g can be connected to any point of the backbone as long as the routing distance is tolerable. When g has been *virtually* connected to f, the input-output relationship must be checked to see whether adding g to the backbone causes any LI violation. If not, then the tree growing for g has been successful and the tree has one more branch with g embedded to the new backbone. Every time when an input is added to the backbone, the distance for each isolated input must be updated because addition of the input may change the distance for part of the isolated inputs.

If, unfortunately, the added input (g) cannot satisfy the LI constraint, then the (f,g) pair must be discarded from the candidate list. The next shortest distance for g must be calculated by excluding both f and its corresponding wire from further consideration as shown in Fig. 8(b), and the entire distance list must be sorted to reflect the newest situation. Again, a pair of isolated input and hit grid point with shortest distance is selected as the candidate for further consideration. The above process is repeated until all isolated inputs have been included into the tree or no inputs can be further considered. If no isolated input exists after the tree growing process, then the circuit has been well reconfigured to allow pseudo-exhaustive testing of the entire circuit by one test session. Our experimental results demonstrate that the TLS test structure can be efficiently established to pseudo-exhaustively test most benchmark circuits. However, if the tree growing process ends up with some inputs isolated or intolerably long tree branches, then XOR gates must be employed to eliminate the difficulties as discussed in the next section.

5 XOR-Tree Generation

For a given (n, m, k) circuit, probably, we might fail to generate a desirable TLS by employing a certain primitive polynomial of degree k, i.e., there exist some isolated inputs remaining disconnected after the backbone generation and tree growing processes have been completed. For example, we desire to construct the TLS configuration using primitive polynomial, $P(x) = x^4 + x^3 + 1$, for the purpose of pseudo-exhaustively testing the circuit shown in Fig. 9(a). After both backbone generation and tree growing processes are applied, the result illustrated in Fig. 9(b) tells that input i is still isolated. Once this situation takes place, the original objective of pseudo-exhaustively testing the (n, m, k) circuit in one test session using 2^k patterns cannot be achieved. Sometimes, it is quite difficult to find a primitive polynomial by which we can successfully construct the desirable TLS for a given CUT without leaving

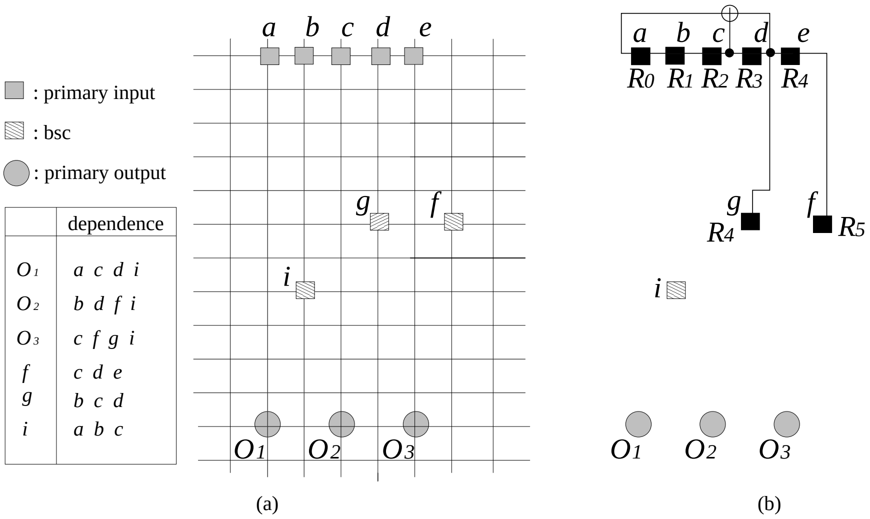

Fig. 9: (a) An (*8, 6, 4*) circuit.
(b) Node isolation after the backbone and tree-growing process.

any inputs isolated. For this reason, we shall not attempt to overcome the problem of rescuing isolated inputs by brutally identifying another primitive polynomial for the CUT, if better alternatives can be used.

As discussed above, an isolated input N remains disconnected if it cannot be the next stage of any register on the resultant Tree-LFSR/SR. However, some of the stages *jointly* might be able to resolve the LI problem for node N if they are properly summed using XOR gates. The following lemmas and theorems will prove this fact, and a method to effectively synthesize the desirable residue using a limited number of existent register stages will be given.

Lemma 1: Given a simple LFSR/SR chain which is constructed from a primitive polynomial $P(x)$ of degree k, the first $2^k - 1$ stages of the chain generate all possible residues R_j, $0 \leq j \leq 2^k - 2$; that is, the remainders of x^j's divided by $P(x)$ are all different [11].

Many interesting properties can be found, if the residues are spanned into a k-dimensional linear space. By definition, a vector set S_i is linear independent provided that all vectors in S_i are linearly independent.

Lemma 2: Given two linearly independent vector sets, S_1 and S_2, in a k-dimensional vector space with $r = max(rank(S_1),\ rank(S_2))$, there exists another set containing at most $k - r$ vectors that can be added into S_1 and S_2 without causing S_1 and S_2 to be linearly dependent [12].

Theorem 3: If the largest residue implemented by the Tree-LFSR/SR configuration is R_i and input node N is isolated, then there exists a residue R_j, $j > i$, that can be assigned to N without causing N to be linearly dependent [12].

Theorem 4: Given a residue R_j, $j > k$, there exists a set of residues $\{R_{j_1}, R_{j_2}, ..., R_{j_n}\}$ such that

$$R_j = R_{j_1} + R_{j_2} + ... + R_{j_n}$$

where $0 \leq j_i < k$ for $1 \leq i \leq n$, and "+" stands for modulo-2 addition [12].

In fact, the desirable residue R_j can be synthesized using a subset of the residues taken from the LFSR portion based on Theorem 4. The number of required XORs is one less than the number of non-zero terms consisting of R_j as shown in Theorem 4. Unfortunately, implementing R_j by taking only the combination of LFSR stages into consideration might not be economical, as the SR stages are unnecessarily wasted. Hardware overhead can be further saved if the SR stages are also involved in the R_j substitution process. Based on this concept, an effective residue replacement technique will be developed. Theorem 6 below establishes the cornerstone for further discussions.

Lemma 5: Under $GF(2)$, if $x^j + Q(x) \equiv 0 \bmod P(x)$, then $x^j \equiv Q(x) \bmod P(x)$ [12].

Theorem 6: Under $GF(2)$, consider a degree-k primitive polynomial, $P(x) = x^k + \Sigma_{i=k-1}^{0} a_i x^i$, where $a_i \in \{0, 1\}$. If we have $j > k$ and $d = j - k$, then

$$R_j = \Sigma_{i=k-1}^{0} \; a_i R_{i+d} \; [12].$$

Theorem 6 above provides a guideline to develop a systematic strategy in implementing the desirable residues, instead of arbitrarily searching a combination of residues from the existent residue domain. Consider again the circuit shown in Fig. 9(b) with input i remaining disconnected, when primitive polynomial $P(x) = x^4 + x^3 + 1$ is used. Starting from R_7 to obtain a target R_j, $j > 6$, we immediately find that residue R_7 can be assigned to the isolated input i without causing the LD problem. By Theorem 6, the target residue R_7 can be replaced by $R_6 + R_3$ under $P(x) = x^4 + x^3 + 1$, and we can use both R_6 and R_3 to synthesize R_7. Unfortunately, R_6 is not existent and must be further substituted by R_5 and R_2. Finally, residue R_7 is synthesized by $R_2 + R_3 + R_5$ and assigned to the isolated input i as illustrated in Fig. 10(a).

As shown in Fig. 10(a), R_7 has been implemented by an XOR circuit, and register stage i immediately following the output of the XOR network will implement R_8. Thus, the above residue replacement process must be performed by synthesizing an XOR network to realize R_{j-1}, which is the residue immediately preceding the target one. Then, adding a register stage to the output of the XOR network will give the target residue R_j. The example of Fig. 9(a) can be repeated by synthesizing R_6 using existent residues. By applying Theorem 6, we can have R_6 synthesized by taking the module-2 addition of both R_5 and R_2 which are already existent. If

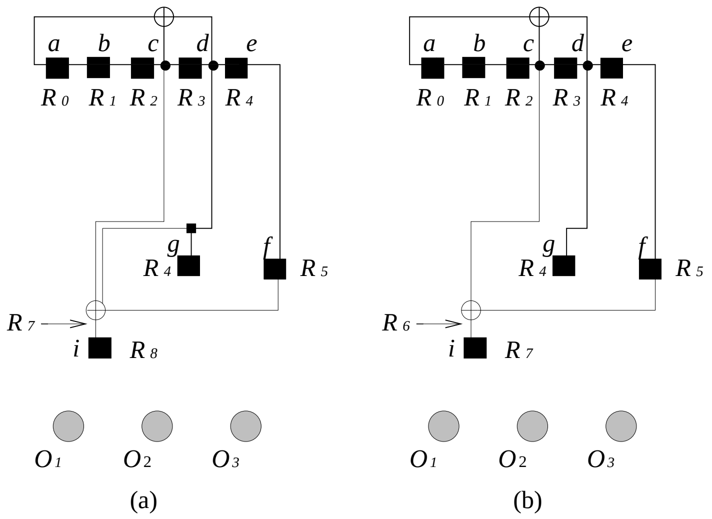

Fig. 10: (a) Residue replacement without considering bsc.
(b) A correct residue replacement with bsc considered.

the output of the XOR network is connected to register stage i, the target residue R_7 is accurately generated as shown in Fig. 10(b). The reason for not directly taking the XOR output is to avoid any circuit instability caused by the XOR tree delay. The above process, called XOR-tree generation, is repeated until all isolated inputs are connected to the TLS architecture. To save the routing area, the isolated node closest to the TLS has the highest priority to be selected for residue substitution. Previous discussions indicate that the residue replacement process terminates once the target residue has been substituted by existent ones. However, existent residues can be further substituted by lower-indexed residues which are closer to the isolated input, under the guidance of component coordinates. Of course, determining a set of minimal number of existent residues, which are closest to the isolated input, is not an easy task since many factors must be considered. After all isolated inputs have been connected, the TLS with XOR networks added will be able to pseudo-exhaustively test the CUT.

6 Simulation Results

To evaluate the performance of the proposed TLS test synthesis scheme, simulation software has been developed using C as the high-level language on a SUN Sparc workstation. Ten ISCAS85 combinational benchmark circuits are first used to test the program. Before applying the TLS method, we use a quite efficient strategy, similar to [6], to partition the benchmark circuits such that each of them has a maximum dependency smaller than or equal to 20. The circuit partitioning results are given in Table 1, and *bscs* are added to satisfy the maximum dependency requirement. For example, originally, C432 has 36 inputs, 7 outputs, and its maximum dependency is 36. However,

Table 1: Circuit partitioning (combinational circuits).

Ckt	(n, m, k)	
	Before Partitioning	After Partitioning
c432	(36, 7, 36)	(56, 27, 20)
c499	(41, 32, 41)	(49, 40, 14)
c880	(60, 26, 45)	(75, 41, 20)
c1355	(41, 32, 41)	(49, 40, 14)
c1908	(33, 25, 33)	(47, 39, 19)
c2670	(233, 140, 122)	(262, 169, 20)
c3540	(50, 22, 50)	(118, 90, 20)
c5315	(178, 123, 67)	(225, 170, 20)
c6288	(32, 32, 32)	(87, 87, 20)
c7552	(207, 108, 194)	(296, 197, 20)

after circuit partitioning, the number of inputs (including the psudo-inputs induced by *bscs*) has increased to 56, and the number of outputs (including the pseudo-outputs) has increased to 27. Thus, we know that 20 *bscs* are inserted while the maximum dependency has been successfully reduced from 36 to 20. It should be noted that the number of *bscs* inserted by our circuit partitioning method is only slightly different from that of [6] (the best results to date) in most cases.

Then, we apply the proposed TLS method to the partitioned benchmark circuits with the results shown in Table 2 where columns 1 and 2 give the circuit names and their basic characteristics. The primitive polynomial employed for each circuit is shown in column 3. Here, we only depict the exponents of non-zero items. Column 4 provides the total number of inputs (including *bscs*) that are involved in the TLS test architecture, while columns 5 and 6 give the numbers of inputs that have been solved by the backbone generation and tree growing processes, respectively. The number of nodes remaining isolated is presented in column 7, and the number of XOR gates required to further embed these isolated nodes into the tree architecture is shown in column 8. The CPU time consumed for executing TLS using SPARC 10 can be found in column 9 while column 10 gives the number of XOR gates required by [11] (the best results to date).

7 Conclusions

We have presented a novel test generation architecture, called TLS, to pseudo-exhaustively test combinational and sequential networks in one test

Table 2: Simulation results for the TLS synthesis scheme (combinational circuits).

Ckt	(n, m, k)	Polynomial	Inputs	Inputs in backbone	Inputs in branches	Isolated Inputs	XOR	CPU (secs)	XOR [16]
c432	(56, 27, 20)	20 19 4 3 0	56	44	12	0	0	0.56	9
c499	(49, 40, 14)	14 9 8 3 0	49	48	1	0	0	0.54	11
c880	(75, 41, 20)	20 19 4 3 0	75	69	6	0	0	0.64	13
c1355	(49, 40, 14)	14 9 8 3 0	49	47	2	0	0	2.77	11
c1908	(47, 39, 19)	19 6 5 1 0	47	41	4	2	3	2.41	10
c2670	(262, 169, 20)	20 19 4 3 0	262	247	15	0	0	13.74	7
c3540	(118, 90, 20)	20 19 4 3 0	118	72	45	1	6	34.82	27
c5315	(225, 170, 20)	20 19 4 3 0	225	186	39	0	0	75.66	36
c6288	(87, 87, 20)	20 19 4 3 0	87	59	25	3	15	259.37	25
c7552	(296, 197, 20)	20 19 4 3 0	296	216	80	0	0	33.59	31

session. Basically, the proposed test architecture takes advantages of both signal sharing and signal reuse properties to minimize XOR gate overhead and derivation time of the test architecture. As discussed before, the process of constructing the desirable TLS mainly consists of three phases – backbone generation, tree growing, and XOR-tree generation. After the first two phases, there may be isolated inputs that cannot be added into the TLS configuration due to LI violation. If this occurs, we may employ XOR gates to overcome the encountered LD problem. We have also developed a method to generate signatures for the proposed test architecture [12]. Experimental results obtained by computer simulation on benchmark circuits demonstrate that the TLS test architecture is indeed very powerful. Note that experimental results for sequential benchmark circuits can be found in [12].

Acknowledgement

This work was partially supported by Hong Kong Earmarked Research Grant CUHK4163/97E, CUHK4412/99E, and CUHK Direct Grant 2050199.

References

1. E. J. McCluskey, "Verification Testing - A Pseudoexhaustive Test Technique", *IEEE Trans. on Computers,* vol. **C-33**, no. 6, pp. 541-546, June 1984.

2. W. B. Jone and C. A. Papachristou, "A Coordinate Circuit Partitioning and Test Generation Method for Pseudo-Exhaustive Testing of VLSI Circuits", *IEEE Trans. on CAD,* vol. **14**, no. 3, pp. 374-384, Mar. 1995.

3. D. Kagaris, F. Makedon and S. Tragoudas, "A Method for Pseudo-Exhaustive Test Pattern Generation", *IEEE Trans. on CAD,* vol. **13**, no. 9, pp. 1170-1178, Sep. 1994.

4. M. W. Roberts and P. K. Lala, "An Algorithm for the Partitioning of Logic Circuits", *IEE Proc.,* vol. **131**, pt. E, no. 4, Jul. 1984.

5. S. N. Bhatt, F. R. K. Chung and A. L. Rosenburg, "Partitioning Circuits for Improved Testability", *Proc. of 4th MIT Conf. on Advanced Research in VLSI,* pp. 91-106, Apr. 1986.

6. R. Srinivasan, S. K. Gupta and M. A. Breuer, "An efficient Partitioning Strtegy for Pseudo-Exhaustive Testing", *Proc. of Design Autamation Conf.,* pp.242-248, Jun. 1993.

7. Z. Barzilai, D. Coppersmith and A. L. Rosenberg, "Exhaustive Generation of Bit Patterns in Discontiguous Positions with Applications to VLSI Self-Testing", *IEEE Trans. on Computers,* vol. **C-32**, no. 2, pp. 190-194, Feb. 1983.

8. S. B. Akers, "On the Use of Linear Sums in Exhaustive Testing", *Proc. of 15th Int'l. Symp. on Fault-Tolerant Computing,* pp. 148-153, Jun. 1985.

9. E. Wu and P. W. Rutkowski, "PEST - A Tool for Implementing Pseudo-Exhaustive Self Test", *Proc. of 1st European Design Automation Conf.,* pp. 639-643, Mar. 1990.

10. C.-I. H. Chen and J. T. Yuen, "Automated Synthesis of Pseudo-Exhaustive Test Generator in VLSI BIST Design", *IEEE Trans. on VLSI Systems,* vol. **2**, no. 3, pp. 273-291, Sep. 1994.

11. R. Srinivasan, S. K. Gupta and M. A. Breuer, "Novel Test Pattern Generators for Pseudo-Exhaustive Testing", *Proc. of International Test Conf.,* pp. 1041-1050, 1993.

12. J. C. Rau, W. B. Jone, S. C. Chang, and Y. L. Wu, *Pseudo-Exhaustive Testing of VLSI Circuits Using Scan-Tree Test Architecture,* Technical Report 00-01, Dept. of Computer Science and Information Engineering, National Chung-Cheng University, Mar. 2000.