Pseudo-Exhaustive Built-In Self-Test System For Logic Circuits

Rajagopalan Srinivasan

CENG Technical Report 94-26

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, California 90089-2562
(213)740-4469

December 1994

PSEUDO-EXHAUSTIVE BUILT-IN SELF-TEST SYSTEM FOR LOGIC CIRCUITS

by

Rajagopalan Srinivasan

A Dissertation Presented to the
FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(Computer Engineering)

December 1994

UNIVERSITY OF SOUTHERN CALIFORNIA
THE GRADUATE SCHOOL
UNIVERSITY PARK
LOS ANGELES, CALIFORNIA 90007

This dissertation, written by

RAJAGOPALAN SRINIVASAN

under the direction of his Dissertation Committee, and approved by all its members, has been presented to and accepted by The Graduate School, in partial fulfillment of requirements for the degree of

DOCTOR OF PHILOSOPHY

Alice C. Parker

Dean of Graduate Studies

Date October 13, 1994

DISSERTATION COMMITTEE

Michael A. Breuer
Chairperson

Sandeep Gupta
CO-CHAIR.

Peter A. Beerel

Dedication

To my parents

Smt. Vasantha Srinivasan & Sri. V. Srinivasan

Acknowledgements

I am grateful to my advisor Prof. Melvin Breuer for his inspiration, guidance and encouragement during my dissertation work. His invaluable constructive criticisms have enhanced the quality and presentation of my research over the years. I consider it as my privilege to have worked with him. I express my deep sense of appreciation to Prof. Sandeep Gupta for his enthusiasm, guidance and support towards making my doctoral work a rewarding experience. I wish to thank Prof. Peter Baxendale for serving on the guidance and dissertation committees. I also would like to thank Prof. Alice Parker and Prof. Cauligi Raghavendra for being part of the guidance committee.

During my years at USC I benefited greatly from interacting with many colleagues and friends. In particular I would like to mention Suresh Chalasani, Pravil Gupta, Rajesh Gupta, Rajiv Gupta, Rajeshwari Krishnan, Kuen-Jong Lee, Mody Lempel, Jung-Cheun Lien, Sen-Pin Lin, Amitava Majumdar, Debaditya Mukherjee, Sridhar Narayanan, Charles Njinda, Dhabaleshwar Panda and Ishwar Parulkar. All of them have made my stay at USC memorable. I am also indebted to Donnalyn Combest, Lucille Stivers and Mary Zittercob of the Electrical Engineering Department at USC for their help and encouragement.

I would like to acknowledge the financial support provided by the Advanced Research Projects Agency (ARPA) through Contract No. J-FBI-90092 (monitored by the Federal Bureau of Investigation).

My sincere gratitude goes to my parents, my wife Sujatha and my sisters Radha and Geetha for their love, support and encouragement over the years that has significantly contributed to the successful completion of my doctoral work.

Contents

List Of Figures

List Of Tables

Abstract

Built-in self-test is a cost-effective approach for designing testable versions of logic circuits. It obviates external testing by providing on-chip hardware as test resources. Pseudo-exhaustive test strategy involves applying all possible input patterns to individual output cones of a circuit. The strategy ensures detection of all irredundant multiple stuck-at faults in the circuit and all irredundant combinational faults within individual cones. This thesis presents an integrated *pseudo-exhaustive built-in self-test system* applicable to circuits designed with random or structured logic styles. The system consists of a suite of *partitioning schemes* and *test pattern generator designs* for testing the circuits. Circuits with output cones driven by a large number of inputs need to be partitioned to reduce the overall test application time. Efficient partitioning schemes are presented to restrict the number of inputs driving the individual output cones. These schemes are based on graph-theoretical concepts and have polynomial computational complexity. Novel test pattern generators that employ knowledge of the circuit cone structures are designed for generating pseudo-exhaustive test sets. These designs are based on the theory of linear feedback shift registers and generate minimal test sets. Tight upper bounds on pseudo-exhaustive test lengths are derived based on new algebraic results. The bounds are used to characterize various classes of circuits and provide good estimates on test length. Experimental results on various benchmark circuits validate the efficiency and quality of the system. A coordinated approach to partitioning and test pattern generation provides a spectrum of testable design solutions trading off design costs such as hardware overhead and test length.

Chapter 1

Introduction

The ever-increasing complexity of VLSI circuits, in terms of gate-counts on chips, chip-counts on printed circuit boards (PCBs) and die-counts on multi-chip modules (MCMs), results in ever-increasing problems in the external testing of chips, PCBs and MCMs. Our main objective is to develop an integrated *built-in self-test system* for designing self-testable logic circuits. We consider a testing strategy, namely *pseudo-exhaustive testing*, that ensures comprehensive fault coverage leading to highly reliable circuits. In this chapter, we shall introduce the concepts of *design for testability* (DFT) and *built-in self-test* (BIST) and then present the issues addressed in this research.

1.1 Design for Testability

Logic circuits typically contain tens of thousands of gates and exhibit sequential behavior due to the presence of storage elements. The storage elements can be either latches or flip-flops. Deterministic test generation is computationally intensive especially for sequential circuits.

The ability to control an internal signal value from the primary inputs is referred to as *controllability*. Similarly, the ability to observe an internal signal value at the primary outputs by controlling necessary inputs is referred to as *observability*. DFT techniques [2] improve the *controllability* and *observability* of circuits and thus reduce the test generation effort for these circuits. Examples of DFT techniques include full scan, partial scan, boundary scan and BIST.

Full scan techniques modify all storage elements in a circuit to have shift capabilities, while partial scan techniques modify only some of these elements. The modified storage elements are configured to form one or more scan paths during the test mode and are serially accessed from the circuit I/Os. The modified storage elements can be regarded as circuit I/Os during test generation. Test patterns can be generated by an *automatic test pattern generator* (ATPG) for full scan and partial scan circuits [14]. *Automatic test equipment* (ATE) [30] is required to store and apply the test patterns to the circuit. The circuit is tested by applying stimuli from the inputs and scan paths and inspecting responses at the outputs and scan paths.

Due to the high packaging density of chips at the board level, probing of I/O pins by ATE is becoming increasingly difficult. To alleviate this problem, chips are built with *boundary scan* [26] capabilities. All I/O pins of individual chips are connected serially in a boundary scan path. The I/O pins and the internal scan paths of chips can be serially accessed through the boundary scan path. Boundary scan is used for testing the internal circuitry of the chips and interconnections between them.

Though scan techniques reduce the test generation effort for sequential circuits, the following costs are associated with them.

- **Area Overhead:** The modification to storage elements and configuring them into scan paths increase the gate count and routing overhead respectively. Each scan path requires two I/O pins for scan operations. The overhead is proportional to the number of storage elements enhanced to have scan capability.

- **Test Generation:** Deterministic tests are tailored to detect target faults and are generated by an ATPG system. The system includes test generation and fault simulation routines that are computationally intensive.

- **ATE:** Test stimuli and correct responses for the scan paths and the I/Os are stored by ATE. The amount of storage depends on the characteristics of circuits. For chips without boundary scan, the I/O pins are probed by ATE for applying the test patterns.

- **Test Time:** Test patterns are serially scanned in and out of the scan paths and a large amount of testing time is spent in scan operations. The speed

of testing is determined by the scan clock generated by ATE which is usually much slower than the system clock.

1.2 Built-in Self-Test

BIST techniques [2] overcome some of the costs associated with scan designs at the expense of more area overhead. BIST designs have on-chip dedicated hardware as test resources. Test patterns are generated internally by *test pattern generators* (TPGs) and results are compacted by *signature analyzers* (SAs). TPGs can be either counters or autonomous *linear feedback shift registers* (LFSRs), and SAs are usually *multiple input signature registers* (MISRs) based on LFSRs. During the test mode, some of the storage elements are modified to act as TPGs and/or SAs.

Parameter	Scan Designs	BIST Designs
Area overhead	Storage elements are enhanced to have shift mode.	Storage elements are enhanced to have shift, TPG and/or SA modes.
Test Generation	ATPG is used to generate deterministic patterns for target faults.	On-chip TPGs (SAs) are used to generate (pseudo) random patterns.
Test Length	Usually low.	Usually high.
Fault Coverage	Successive patterns improve fault coverage.	Successive patterns may not necessarily improve fault coverage.
Test Storage	All test patterns are stored.	Only the seeds and signatures are stored.
Test Application Time	Several clock cycles are required for applying each pattern.	Test pattern is generated and applied on every clock cycle.
Test Speed	Test clock is dictated by ATE and is usually slower than system clock.	Tested with system clock.
Test Control	ATE is responsible.	On-chip controller is used.

Table 1.1: Comparision of design parameters between scan and BIST designs.

A comparison of the characteristics of scan and BIST designs is given in Table 1.1. Both scan and BIST techniques concentrate on testing the combinational blocks of a circuit. The modified storage elements are tested by scanning special test patterns. A good strategy for testing the combinational blocks ensures the testing of all unmodified storage elements in the circuit.

The various types of testing addressed by BIST techniques can be summarized as follows.

- **Exhaustive testing** deals with applying all possible input patterns to a circuit. The test assumes no fault modeling and hence does not require simulation of faults in the circuit. Exhaustive testing ensures detection of all irredundant combinational faults in the circuit. A combinational fault does not manifest any sequential behavior of the circuit and is testable with a single input pattern. The test time increases exponentially with the number of inputs to the circuit.

- **Pseudo-exhaustive testing** considers the circuit as a collection of output cones and applies all possible input patterns to individual output cones. For an output, the subcircuit comprising of circuit inputs, logic gates and interconnections that feed the output is referred to as the *output cone*. Subcircuits common to the output cones are tested several times during the exhaustive testing of individual output cones. Pseudo-exhaustive testing ensures detection of all irredundant combinational faults within individual output cones and all irredundant multiple stuck-at faults in the circuit. The test time increases exponentially with the number of inputs feeding the output cone with the largest number of inputs.

- **Pseudo-random testing** deals with applying an incomplete set of unique patterns generated by an autonomous LFSR. The patterns have pseudo-random characteristics but are deterministic and repeat after a fixed cycle length. Fault coverage is determined either by simulation of modeled faults or by probabilistic estimators. Patterns can be biased using weighted hardware to improve the fault coverage.

- **Random testing** deals with random patterns generated by sources like SAs. The patterns have random characteristics and repeat without any fixed cycle length. The effectiveness of the patterns is measured by fault simulation. The strategy requires least additional hardware for test pattern generation.

Most BIST systems assume the sequential circuit to be made self-testable consists of combinational logic blocks separated by registers. However, designers tend to design circuits with functional hierarchy that may not be ideally suited for the systems. For the BIST system being built at USC [25], the input circuits are hierarchically reorganized and canonically partitioned into combinational blocks and registers [13, 31]. Our BIST system concentrates on testing the combinational blocks of the circuit and deals with the afore mentioned testing strategies. The system explores the design space in terms of various design constraints, viz. area overhead, test time and fault coverage. Although several systems have been developed during the recent years, open issues such as achieving optimality and providing a spectrum of solutions in the design space are addressed in our system.

This research focusses on the pseudo-exhaustive testing strategy for logic circuits. Some issues related to our BIST system that are not addressed in this research are listed below.

- various testable design methodologies (TDMs) [1] based on built-in logic block observation (BILBO) [23] architecture.
- test resource allocation for various combinational blocks and test scheduling for various test sessions.
- test controller synthesis implementing the test schedules.

The research issues related to TDMs, test resource allocation and test scheduling are addressed in [25] and the problems related to test controller synthesis are addressed in [29].

1.3 Pseudo-Exhaustive Test Strategy

Our goal is to design self-testable logic circuits ensuring comprehensive fault coverage under the combinational fault model. Exhaustive testing ensures comprehensive

fault coverage but may not be practical since the number of test patterns increases exponentially with the number of inputs to the circuit. Pseudo-exhaustive test strategy is attractive since it can test *almost* all combinational faults in the circuit with significantly less test patterns than exhaustive testing. Since each output cone is exhaustively tested, the only combinational faults that a pseudo-exhaustive test *may* miss are those faults that make an output dependent on additional inputs [2]. For example, a bridging fault between two inputs may not be detected if none of the outputs are dependent on those two inputs together. The pseudo-exhaustive test strategy obviates deterministic test pattern generation and fault simulation of stuck-at faults.

Consider a combinational circuit with n inputs and m outputs. The number of inputs feeding an output cone is referred to as the *size* of the output cone. Let the m output cones of the circuit be of sizes less than or equal to k and assume that there exists at least one output cone with size equal to k. The value k is referred to as the *maximum cone size* of the circuit. The circuit can be characterized as an (n, m, k) circuit.

Exhaustive testing of the circuit requires 2^n patterns which may be prohibitive for large values of n. Pseudo-exhaustive test strategy considers the circuit as a collection of m output cones and exhaustively tests each cone of the circuit. The test time is bounded below by 2^k since there exists at least one output driven by exactly k inputs. If k is sufficiently less than n, then the time taken for pseudo-exhaustive testing can be significantly less compared to exhaustive testing.

Pseudo-exhaustive testing gives rise to three important research issues, namely *partitioning, test pattern generation* and *bounds on test lengths.* We shall briefly describe these research issues in the following.

1.3.1 Partitioning

Pseudo-exhaustive testing may not be practical for circuits with output cones driven by an unacceptably large number of inputs. These circuits need to be partitioned such that the output cones are driven by an acceptable number of inputs. The partitioning is achieved by placing segmentation cells [16, 19] in the original circuit. The behavior of a segmentation cell is shown in Figure 1.1. The cell behaves as a

transparent signal during the normal mode of operation and manifests as a pseudo-input and a pseudo-output during the test mode. Thus the number of output cones in the circuit increases during the test mode due to segmentation cells placed in the circuit. However, the maximum cone size of the circuit decreases by a large extent. The segmentation cells amount to hardware overhead and introduce delays during normal operation. Hence it is required to place a minimum number of cells in the circuit without affecting critical paths.

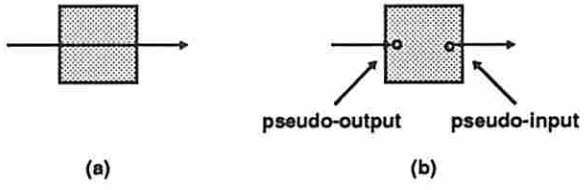


Figure 1.1: Segmentation cell behavior during (a) normal mode (b) test mode

Logic circuits can be classified into various categories such as circuits with or without fanouts, two-level or multi-level circuits and iterative logic arrays (ILAs). Partitioning strategies can take advantage of the unique structural properties of these circuits. We have developed optimal partitioning strategies of polynomial complexity for both two-level and multi-level fanout-free circuits. Circuits with fanouts make the partitioning problem hard and require strategies of exponential complexity to obtain optimal solutions. The regularity in ILAs makes it easy to determine upper bounds on the number of segmentation cells required to partition these circuits.

We have formulated the partitioning problem for fanout circuits as the classical integer linear programming (ILP) problem. For small circuits, optimal solutions can be obtained by using this formulation. However, the formulation may not be computationally viable for large circuits since the number of constraints grows non-linearly with the number of levels in the circuit. To handle large circuits, we have developed an efficient heuristic partitioning procedure of polynomial complexity. The heuristic is based on the graph-theoretical concept of articulation nodes [10].

Reduction in the maximum cone size amounts to reduction in pseudo-exhaustive test time but results in the addition of hardware overhead. Thus there exists trade-offs between test time and hardware overhead in designing pseudo-exhaustive self-testable versions of circuits.

1.3.2 Test Pattern Generation

Any pseudo-exhaustive test set for an (n, m, k) circuit must contain an exhaustive set of patterns for each of the m output cones of the circuit. The size of the test set is between 2^k and 2^n. Though any individual output cone can be exhaustively tested within a maximum number of 2^k patterns, it may not be possible to test all cones simultaneously with 2^k patterns. This is primarily due to conflicting input requirements for the output cones and hence the circuit may require more than 2^k patterns. Generation of an optimal pseudo-exhaustive test set for an (n, m, k) circuit is a hard problem [5].

An optimal pseudo-exhaustive test set for an (n, m, k) circuit can be ensured by further partitioning the circuit with segmentation cells to resolve all the conflicting requirements for the output cones. The resolution ensures that all output cones can be exhaustively tested simultaneously with 2^k patterns. An (n, m, k) circuit that requires only 2^k patterns for pseudo-exhaustive testing is referred to as *maximal test concurrent* (MTC) circuit [27]. Partitioning non-MTC circuits to achieve maximal test concurrency leads to simple TPG designs. We have developed a partitioning strategy based on the graph-theoretical concept of bridges [10]. TPGs based on either maximal length LFSRs or counters can generate optimal pseudo-exhaustive test sets for MTC circuits.

On the other hand, complicated TPGs can be designed to generate minimal pseudo-exhaustive test sets for non-MTC circuits. These circuits need not be partitioned to achieve maximal test concurrency. We have designed novel TPGs that utilize minimal hardware and generate minimal pseudo-exhaustive test sets for these circuits. Our TPGs generate minimal test sets by employing knowledge of the sets of circuit inputs driving each output cone of the circuit. Maximal length LFSRs form the basic underlying structure of our TPG designs.

1.3.3 Bounds on test lengths

We have derived tight upper bounds on the sizes of pseudo-exhaustive test sets for circuits. New algebraic results on vector spaces are derived and used in the computation of these bounds. Generic bounds that are independent of circuit output cone structures are used to characterize various classes of circuits. Circuit cone-specific bounds are also derived by utilizing the structural information about the circuit output cones. We have also developed an efficient method to permute circuit inputs in order to obtain the best improvement on the cone-specific bounds. The quality of these bounds are demonstrated by comparision with the existing bounds [3, 4]. Our bounds provide good estimates of test lengths and can be used as guiding factors in designing TPGs for circuits. The computed theoretical bounds for circuits comply well with the sizes of test sets generated by our TPGs.

1.4 Thesis Organization

The thesis is organized as follows. Chapter 2 presents the background by summarizing the previous related work on partitioning strategies, TPG designs and test length estimations. The shortcomings and open issues in the related work provide the motivation for this research. Chapter 3 deals with partitioning strategies for reducing the sizes of the output cones in circuits. Various classes of circuits are considered and their unique structural properties are utilized in developing efficient partitioning procedures. The chapter also provides details about the generalized formulation of the partitioning problem and our heuristic approach for large sized circuits. Chapter 4 presents our partitioning approach to achieve maximal test concurrency in circuits. The theory involved in our TPG designs and their characteristics are discussed in Chapter 5. The chapter provides details about the TPG design procedures and hardware overhead involved in the TPG designs. Chapter 6 deals with the derivation of both generic and cone-specific bounds on pseudo-exhaustive test lengths. The chapter provides details about new algebraic results utilized in the derivations of bounds. The integration of the pseudo-exhaustive test system and the contributions of this research are provided in Chapter 7. The chapter also contains concluding remarks and future extensions to this research work.

Chapter 2

Background

In this chapter we shall present a detailed summary of previous work on pseudo-exhaustive testing related to partitioning strategies, test pattern generator designs and bounds on test lengths. These related work form the background and provide motivation for our research.

2.1 Partitioning Strategies

These strategies partition the circuit into smaller segments so that each segment is driven by an acceptable number of inputs. The strategies can be classified into either *constrained* or *unconstrained strategies.* Constrained strategies impose constraints on the segments to be totally disjoint so that each segment can be tested independently. Unconstrained strategies do not impose any constraints on the segments and hence the segments may need to be tested together.

Consider a combinational circuit C shown in Figure 2.1(a). The circuit is fed by register $R1$ and feeds register $R2$. Assume that there exists an output cone being driven by an unacceptably large number of inputs. The constrained partitioning strategy constrains the segments to be totally disjoint as shown in Figure 2.1(b). The original circuit C is partitioned into two disjoint segments $C1$ and $C2$ such that each output cone is driven by an acceptable number of inputs. Four segmentation cells $s1$ through $s4$ are used to create the two disjoint segments. The problem of finding an optimal solution for the constrained partitioning strategy has been shown to be NP-complete [19]. The constrained strategy usually results in a large number of segmentation cells and hinders all critical paths in the circuit.

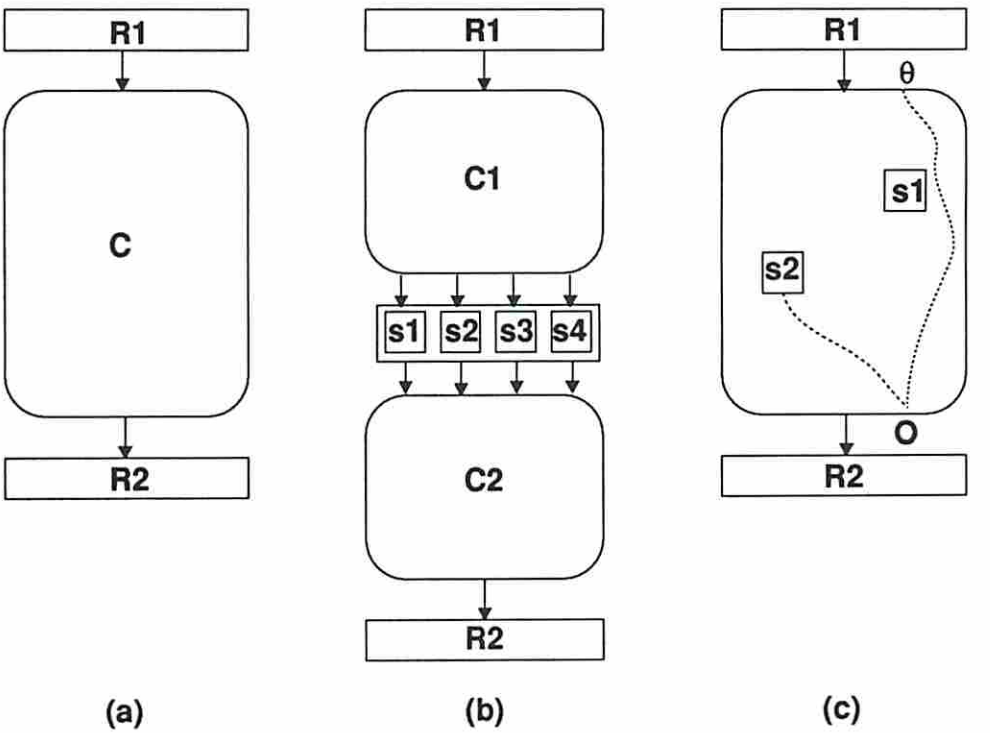


Figure 2.1: Partitioning strategies (a) Unpartitioned circuit (b) Constrained partitioning strategy (c) Unconstrained partitioning strategy

The unconstrained partitioning strategy does not constrain the segments to be disjoint, as shown in Figure 2.1(c). In this case, only two cells $s1$ and $s2$ may be required to achieve the goal. The original circuit C is not partitioned into disjoint segments. An output (say O) may be driven by both a primary input (say θ) and a pseudo-input (say due to cell $s2$) as shown in the figure. The unconstrained strategy is a generalized case of the constrained strategy and hence yields better solutions. Critical paths can sometimes be left undisturbed in the unconstrained strategy, though in some cases extra cells may be required. It is shown in [6] that determining optimal solutions to the unconstrained strategy is also NP-complete. Our research is focussed on achieving good suboptimal solutions for the unconstrained partitioning strategy.

We shall next present a brief description of previous work related to both constrained and unconstrained partitioning strategies.

2.1.1 Constrained Strategies

McCluskey and Bozorgui-Nesbat [28] partitioned a large circuit into segments with sufficiently fewer inputs so that exhaustive testing of each segment is feasible. To exhaustively test a segment, its inputs are made controllable from the primary inputs and its outputs are made observable at the primary outputs. This is achieved by two partitioning methods, namely *sensitized partitioning* and *hardware partitioning.*

In sensitized partitioning, segments are isolated by applying fixed patterns to appropriate primary inputs. Paths from other primary inputs to the segment inputs and paths from the segment outputs to the primary outputs are sensitized. A circuit may have conflicting requirements on its primary inputs for sensitizing two or more paths concurrently. TPG designs are complicated since a subset of their stages have to hold constant values for isolating a segment, while the remaining stages have to generate an exhaustive set of patterns for testing the segment. Details about the formation of segments and design of TPGs for sensitized partitioning can be found in [41] and [39] respectively.

Hardware partitioning is achieved by inserting multiplexers to gain access to the embedded inputs and outputs of the segments. Let us consider a circuit partitioned into two segments $C1$ and $C2$ as shown in Figure 2.2. Multiplexers $M1$ through

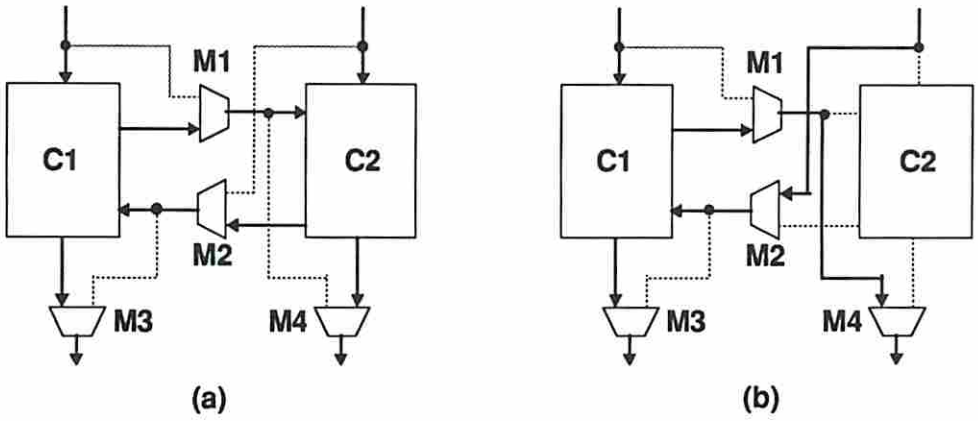


Figure 2.2: Hardware partitioning (a) normal mode (b) Testing $C1$ in test mode

$M4$ are used to gain access to the inputs and outputs of these segments. The settings of the multiplexers during normal operation are shown by the solid lines in Figure 2.2(a). Figure 2.2(b) shows the sensitized paths during testing of segment $C1$. This method deals with block-level partitioning rather than gate-level partitioning. Gate-level partitioning usually leads to reduced hardware overhead compared to block-level partitioning. We shall concentrate on hardware partitioning at the gate-level in this research.

Roberts and Lala [32] partitioned circuits based on the concept of *contribution difference*. The *contributors* to a gate are the primary inputs which feed the gate either directly or indirectly. The *contribution limit* is the maximum number of contributors allowed for any signal in the circuit. The *contribution difference* for a signal is the difference between its number of contributors and the contribution limit. Signals with least contribution difference are selected as candidates for partitioning. The circuit is partitioned such that none of its signals exceed the contribution limit. This method determines optimal solutions for fanout free circuits. However, for circuits with reconvergent fanouts, it gives poor suboptimal solutions.

Jone and Papachristou [20] developed a coordinated approach for partitioning and exhaustive test pattern generation for circuits. A circuit is partitioned into disjoint segments and TPGs are designed for individual segments that can exhaustively exercise all segment outputs concurrently. The heuristic algorithm is based on the minimum vertex cut method. The circuit is modeled as a directed acyclic graph

with the vertices and the edges representing the gates and the signals respectively. A *vertex cut* is a set of vertices whose removal leaves the graph disconnected. Each vertex cut is associated with a weight equal to the number of vertices involved with the cut. The circuit is partitioned into disjoint segments by selecting a set of vertex cuts and placing segmentation cells along these cuts. Since the weight of a vertex cut amounts to the number of segmentation cells required for the cut, the sum of weights associated with the selection of cuts is minimized.

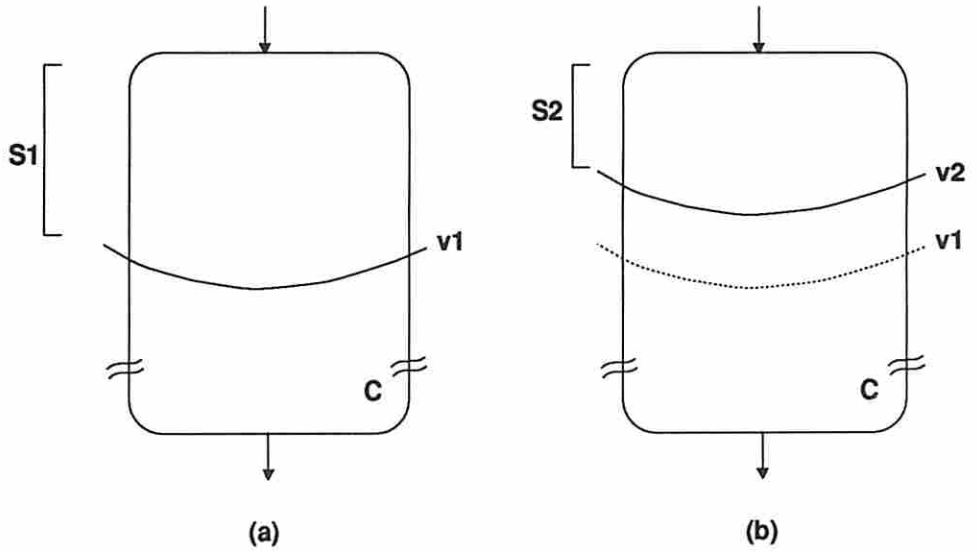


Figure 2.3: Vertex cut approach

The problem of optimally partitioning a circuit into disjoint segments is shown to be NP-complete [19]. The heuristic method for determining a suboptimal solution for a circuit C can be briefly described as follows.

1. Determine the vertex cut (say $v1$) that is farthest away from the inputs so that each gate in the segment (say $S1$) formed between the inputs and $v1$ is driven by an acceptable number of inputs.

2. Determine the minimum vertex cut (say $v2$) for $S1$ that is farthest away from the inputs. Let $S2$ be the segment formed between the inputs and $v2$. The segments $S1$ and $S2$ are shown in Figure 2.3(a) and (b) respectively.

3. Estimate the number of cells required for partitioning the segments $C - S1$ and $C - S2$ by repeatedly applying Step 1. Choose $S1$ or $S2$ (say $S1$) that results in the minimum estimated number of cells.

4. Reduce $S1$ to $S1'$ so that a TPG that generates an optimal test set can be designed for $S1'$.

5. Repeat all steps for $C - S1'$.

This method has the following drawbacks.

1. Having the restriction that segments need to be disjoint usually requires more cells.

2. Since the segments are separated by cells, all critical paths encounter delays due to these cells during the normal operation.

3. Though the minimum vertex cut results in minimal number of cells, segment reduction for designing TPGs results in non-minimal number of cells.

2.1.2 Unconstrained Strategies

Bhatt et al. [6] partitioned circuits using segmentation cells satisfying the following two objectives.

1. All outputs should depend on a restricted number of inputs.

2. All paths from inputs to outputs should encounter the same number of segmentation cells.

Segmentation cells are connected in a scan path during the test mode. The second objective ensures that all paths from inputs to outputs encounter equal register delays during the normal operation. The following two restricted classes of circuits are only considered for partitioning.

1. *Fanin/fanout restricted circuits* in which the fanout of a gate is no greater than its fanin.

2. *Leveled circuits* in which a gate at level i can only drive another gate at level $(i+1)$.

Efficient algorithms are proposed for determining optimal partitions for these restricted classes of circuits. It is shown that optimal partitioning problem satisfying the first objective is NP-complete.

Hellebrand and Wunderlich [16] partitioned circuits into exhaustively testable segments using segmentation cells. The problem is formulated as a combinatorial optimization problem. The *state* of the circuit is described by an N-bit binary vector $< v_1, v_2, \ldots, v_N >$, where N is the number of gates in the circuit. The bit v_i indicates whether a cell is placed on the output of gate i. The weight of a vector denotes the number of cells placed in the circuit. A state is said to be *admissible* if the corresponding circuit configuration has all outputs driven by acceptable of inputs. Only a few states are admissible out of the possible 2^N states for the circuit. Among the set of admissible states, the one with minimum weight gives the optimal solution. A hill climbing procedure with optional backtracking was proposed for suboptimal solutions. Exhaustive test patterns for individual segments are generated off-chip and applied through the scan path. Each cell has a bypass mode so that signals passing through them encounter only a pass transistor delay during the normal mode.

Udell [40] partitioned a circuit such that the maximal fanout-free regions are identified as initial segments. These segments are either merged together or partitioned into smaller segments depending on the number of inputs driving them. Cost functions are used for the merging and partitioning operations. Finally, a post-processing step is carried out to remove unnecessary segmentation cells from the circuit.

Kagaris et al. [21] proposed a two-pass heuristic partitioning method to reduce the maximum cone size of a circuit. In the first pass, segmentation cells are placed so that each output cone is driven by an acceptable number of inputs. In the second pass, a TPG is designed for the partitioned circuit. The second pass may result in placing more segmentation cells. A probabilistic measure is used as guidance for placing the segmentation cells. This method has a similar objective as in [20] but the segments are not required to be disjoint.

We have adopted the unconstrained strategy for partitioning a circuit driven by an unacceptable number of inputs. Critical paths will be left undisturbed by our partitioning strategy. The partitioned segments are tested simultaneously in a single test session. The practicality and efficiency of our partitioning method are demonstrated on the ISCAS combinational benchmark circuits [7].

2.2 Test Pattern Generators

Autonomous LFSRs are widely used to generate pseudo-exhaustive test sets [2]. LFSRs are characterized by their feedback connections represented as *polynomials.* For a non-zero initial state, the *period* of an LFSR is the number of states generated prior to repeating the initial state. An LFSR with n stages is said to be of *maximal length* if it has a period of $2^n - 1$ states. Maximal length LFSRs are based on primitive polynomials and most pseudo-exhaustive TPGs employ maximal length LFSRs in their designs.

Pseudo-exhaustive test pattern generators for an (n, m, k) circuit can be classified as either *universal* or *circuit-specific* TPGs. Universal TPGs generate test sets containing binary n-tuples that cover *all k-dimensional subspaces.* They are applicable for all circuits with given values of n and k irrespective of output cone dependencies. Circuit-specific TPGs generate test sets containing binary n-tuples that cover *specific k-dimensional subspaces* corresponding to the m outputs of the target circuit.

2.2.1 Universal TPGs

Test sets containing binary n-tuples that covers all k-subspaces can be generated using any of the following coding theory techniques.

Constant weight codes satisfy the property that all codewords have identical weights. It is shown in [38] that a test set T covers all k-subspaces if it contains all binary n-tuples of weight w such that $w = c \bmod (n - k + 1)$ for some constant c, where $0 \leq c \leq n - k$. The set of all binary n-tuples of weight w forms the codewords of w-out-of-n constant weight code. Constant weight counters are used to generate the codewords.

Linear codes: An (n, w) linear code contains a set of 2^w distinct codewords that are binary n-tuples satisfying the property that if c_i and c_j are codewords, then $c_i \oplus c_j$ is also a codeword. If a linear code generated with a *generator polynomial* $g(x)$ has length n and minimum distance $k+1$, then a standard LFSR with $g(x)$ as characteristic polynomial generates a test set of binary n-tuples that covers all k-subspaces [37]. Whenever $g(x)$ is not primitive, there is a need for using several seeds for the LFSR.

Condensed LFSRs [42] are also based on linear codes. The smallest integer w satisfying $k \leq \lfloor w/(n-w+1) \rfloor + \lceil w/(n-w+1) \rceil$ is determined. Let $p(x)$ be a primitive polynomial of degree w and $g(x)$ be $(1 + x + x^2 + \ldots + x^{n-w})$. A modular LFSR with characteristic polynomial $p(x)g(x)$ will generate a test set of binary n-tuples that covers all k-subspaces.

Cyclic codes: An (n, w) cyclic code contains a set of 2^w distinct codewords that are binary n-tuples satisfying the property that if c_i is a codeword, then a cyclic right shift of c_i is also a codeword. Cyclic codes form a subclass of linear codes. An (n, w) cyclic code can be generated with a *generator polynomial* $g(x)$ of degree $(n-w)$ that divides $(1+x^n)$. Let $g(x)h(x) = 1+x^n$, where $h(x)$ is the *parity-check polynomial* of the (n, w) cyclic code. Since $h(x)$ also divides $(1+x^n)$, it can be used as a generator polynomial of another cyclic code. The $(n, n-w)$ cyclic code generated by $h(x)$ is the *dual code* of the (n, w) cyclic code generated by $g(x)$. If the cyclic code generated by $g(x)$ has minimum distance of at least $(k+1)$, then the dual code generated by $h(x)$ will cover all k-subspaces. Let $p(x)$ be a primitive polynomial of degree $(n-w)$. A modular LFSR with $h(x)p(x)$ as characteristic polynomial will generate a test set of binary n-tuples that covers all k-subspaces [43].

Lempel and Cohn [24] constructed a test set containing binary n-tuples that covers all k-subspaces by concatenating several maximum-length sequences generated by different primitive polynomials.

Universal TPGs designed with given values of n and k can exhaustively test an output cone dependent on any k-out-of-n inputs. These TPGs usually require a large amount of hardware and/or generate large test sets. For a specific (n, m, k) circuit with its cone dependency information, a test set of size much smaller than that of the universal test set usually exists.

2.2.2 Circuit-specific TPGs

TPGs can be tailored efficiently, both in terms of hardware and test length, by examining the input sets driving the output cones of the circuit. A *test signal* is the unique sequence of binary values applied at a circuit input. An output cone of size k can be exhaustively tested with k linearly independent test signals. However, it may not be possible to concurrently test all the m output cones of the circuit with k signals because of conflicting input requirements among the cones. An upper bound of n test signals are required for the circuit. Two inputs are said to be *related* if there exists an output cone that depends on both of them; else they are said to be *unrelated.* The number of test signals for a circuit can be reduced by sharing some of them among unrelated circuit inputs. Procedures for determining a minimal number of signals (say w, where $k \leq w \leq n$) are described in [5] and [27].

Circuit-specific TPG structures such as LFSR/SRs [4] and LFSR/XORs [3] can be designed for generating pseudo-exhaustive test sets. An LFSR/SR structure is composed of a maximal length LFSR and a shift register (SR). An LFSR/XOR structure is composed of a maximal length LFSR and an XOR network. The TPG structures are shown in Figure 2.4.

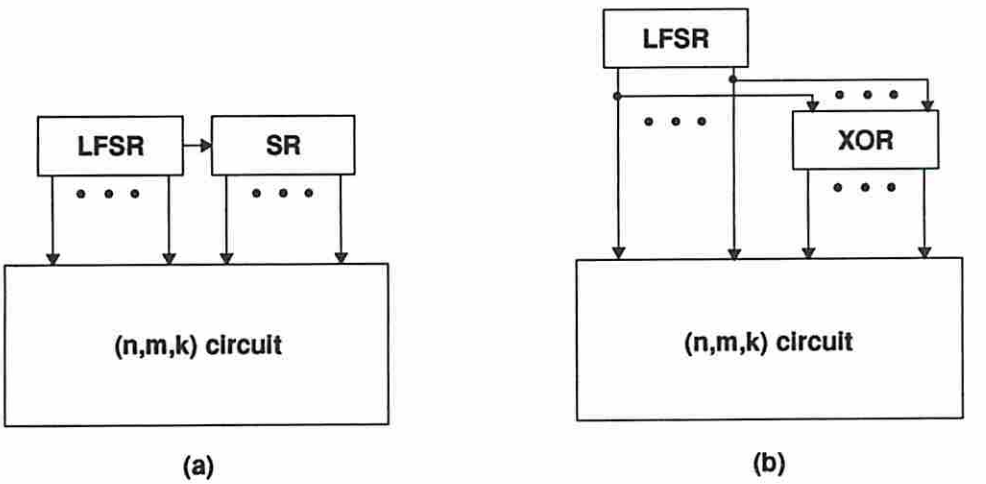


Figure 2.4: TPG structures (a) LFSR/SR (b) LFSR/XOR

LFSR/SR generates independent test signals from the maximal length LFSR portion and specific linear combinations of these signals from the SR portion. The

linear combinations depend on the feedback polynomial used for LFSR. An output cone will be exhaustively tested if and only if the cone is driven by a set of linearly independent test signals [4]. Hence the set of inputs driving an output cone must be assigned a set of linearly independent test signals to exhaustively test the output cone. LFSR/SR incurs minimal hardware overhead as its stages form a shift register. However, due to the inflexibility of the linear combinations of test signals, LFSR/SR generates a large test set.

LFSR/XOR generates independent test signals from the maximal length LFSR and desired linear combinations of these signals from an XOR network. The XOR network provides great flexibilty in generating any desired linear combinations to ensure exhaustive testing of the output cones. LFSR/XOR incurs high hardware overhead due to the XOR network, but generates a minimal length test set.

Hellebrand et al. [17] grouped all m output cones of an (n, m, k) circuit into *k-testable* groups such that all cones in a k-testable group can be exhaustively tested concurrently with 2^k patterns. The problem of determining the minimum number of k-testable groups is shown to be NP-complete [17]. An upper bound on the number of groups is given by $\lceil m/2 \rceil$. The test patterns are generated off-chip by a k-bit counter and scanned to the appropriate inputs. The method results in high test time since each pattern is applied through a scan path.

Chen [8] designed TPGs based on LFSR/XORs by adopting a three-phase procedure. During the first phase, test signals are shared among unrelated inputs. The second phase consists of assigning linear sums of two test signals to some inputs. The final phase consists of assigning linear sums of multiple test signals to the remaining inputs. An XOR network is used to realize the linear sums of test signals. The three-phase procedure attempts to reduce the hardware overhead by minimizing the number of linear sums of test signals.

Kagaris and Tragoudas [22] designed TPGs based on LFSR/SRs by allowing permutation of inputs. The inputs are indexed and the set of indices is determined for the set of inputs (say I_j) driving an output cone (say O_j). The difference (say d_j) between the maximum and minimum values among the indices for the inputs in I_j is then determined. The inputs are assigned a permutation of the indices such that the d_j values for all output cones are minimized. If d is the maximum value among various d_j values for all output cones, then an LFSR/SR based on a primitive

polynomial of degree d is sufficient to exhaustively test all the output cones of the circuit.

We have designed circuit-specific TPGs based on the principles of LFSR/SRs and LFSR/XORs. Our TPG designs utilize minimal hardware like LFSR/SRs and generate minimal test sets like LFSR/XORs.

2.3 Bounds on Test Lengths

Barzilai et al. [4] determined an upper bound on the degree of LFSR/SR for pseudo-exhaustive testing of an (n, m, k) circuit as follows. Let the n inputs be denoted by $\{\theta_1, \theta_2, \ldots, \theta_n\}$. An LFSR/SR is said to be *applicable* if it can generate an exhaustive set of patterns for all output cones in the circuit. Let w_1 be the degree of the feedback polynomial $P(x)$ used in the LFSR/SR structure. Consider an output O_j being driven by the inputs $\{\theta_{i_1}, \theta_{i_2}, \ldots, \theta_{i_k}\}$. It is shown in [4] that $P(x)$ generates an exhaustive set of patterns for O_j if and only if each polynomial $Q(x)$ of the form $\sum_{q=1}^{k} a_q x^{i_q}$ (where $a_q = 0$ *or* 1 and not all of them are zeros) is not divisible by $P(x)$. There are $(2^k - 1)$ such polynomials $Q(x)$ of degree at most n. Each one of the polynomials $Q(x)$ is divisible by no more than (n/w_1) distinct primitive polynomials of degree w_1. Therefore an upper bound on the number of inapplicable primitive polynomials of degree w_1 for O_j is given by $(n/w_1) \times (2^k - 1)$. Considering all m outputs, the total number of inapplicable polynomials of degree w_1 is bounded above by $(n \times m \times (2^k - 1)/w_1)$.

The total number of primitive polynomials of degree w_1 is given by $\Phi(2^{w_1} - 1)/w_1$, where Φ is Euler's phi function [11]. To ensure an applicable polynomial of degree w_1, the total number of inapplicable polynomials of degree w_1 must be less than the total number of primitive polynomials of degree w_1. In other words, the expression $(n \times m \times (2^k - 1)/w_1)$ must be smaller than the expression $\Phi(2^{w_1} - 1)/w_1$. This implies $n \times m \times (2^k - 1) < \Phi(2^{w_1} - 1) \approx 2^{w_1}$. Therefore there exists an LFSR/SR of degree w_1 for an (n, m, k) circuit provided w_1 satisfies the relation $w_1 > log\ (n \times m \times (2^k - 1))$.

Akers [3] determined an upper bound on the degree of an LFSR/XOR for pseudo-exhaustive testing of an (n, m, k) circuit as follows. Consider an output O_j being driven by the inputs $I = \{\theta_{i_1}, \theta_{i_2}, \ldots, \theta_{i_k}\}$. It has been shown in [3] that an LFSR/XOR structure can generate an exhaustive set of patterns for O_j if and only

if the inputs in I are assigned k linearly independent test signals. A set (say L) of 2^k unique test signals can be derived by taking all possible linear sums of these k linearly independent test signals. An input $\theta \notin I$ that drives output O'_j is assigned a test signal not contained in L. This arrangement ensures that the test signal assigned to θ is linearly independent with the test signals assigned to the subset of inputs in I that drive O'_j. The linearly independent test signals assigned to the inputs driving O'_j leads to another set of at most 2^k unique test signals. For all m outputs, the total number of unique test signals that can be possibly generated is bounded above by $(m \times 2^k)$. An LFSR/XOR of degree w_2 can generate 2^{w_2} unique test signals. The LFSR/XOR can assign linearly independent test signals to all n inputs provided the relation $2^{w_2} > m \times 2^k$ is satisfied. Therefore there exists an LFSR/XOR of degree w_2 for an (n, m, k) circuit provided w_2 satisfies the relation $w_2 > (k + log\ m)$.

Chapter 3

Partitioning for Cone Size Reduction

3.1 Introduction

Pseudo-exhaustive testing ensures exhaustive testing of all the output cones in a circuit. A circuit with an output cone driven by an unacceptable number of inputs needs to be partitioned to reduce the test time associated with pseudo-exhaustive testing. Our goal is to place segmentation cells [16, 19] in a given circuit so that each cone is driven by an acceptable number of inputs during the test mode. This is referred to as **partitioning for cone size reduction**. It should be noted that the partitioning of a given circuit may not result in disjoint subcircuits. Segmentation cells add area overhead to the original circuit and introduce delays during the normal mode of operation. Hence it is desirable to place the minimum number of cells in the circuit without affecting the critical paths. We shall adopt a partitioning strategy that does not impose that the partitioned subcircuits be disjoint.

For an (n, m, k) circuit, each output cone requires at most 2^k patterns for exhaustive testing and the circuit requires at least 2^k patterns for pseudo-exhaustive testing. The value of k may be so large as to make pseudo-exhaustive testing infeasible. The test time can be reduced by restricting the size of all the output cones to be below some desired value, say r, referred to as the *cone size limit*. The size of each output cone can be reduced by partitioning the circuit, i.e. inserting segmentation cells. In general it is desired to minimize the number, say s, of such cells. The cells are transparent during the normal mode of operation, and act as pseudo-inputs and pseudo-outputs during the test mode. With s segmentation cells, the original (n, m, k) circuit becomes an $(n + s, m + s, r)$ circuit during the test mode.

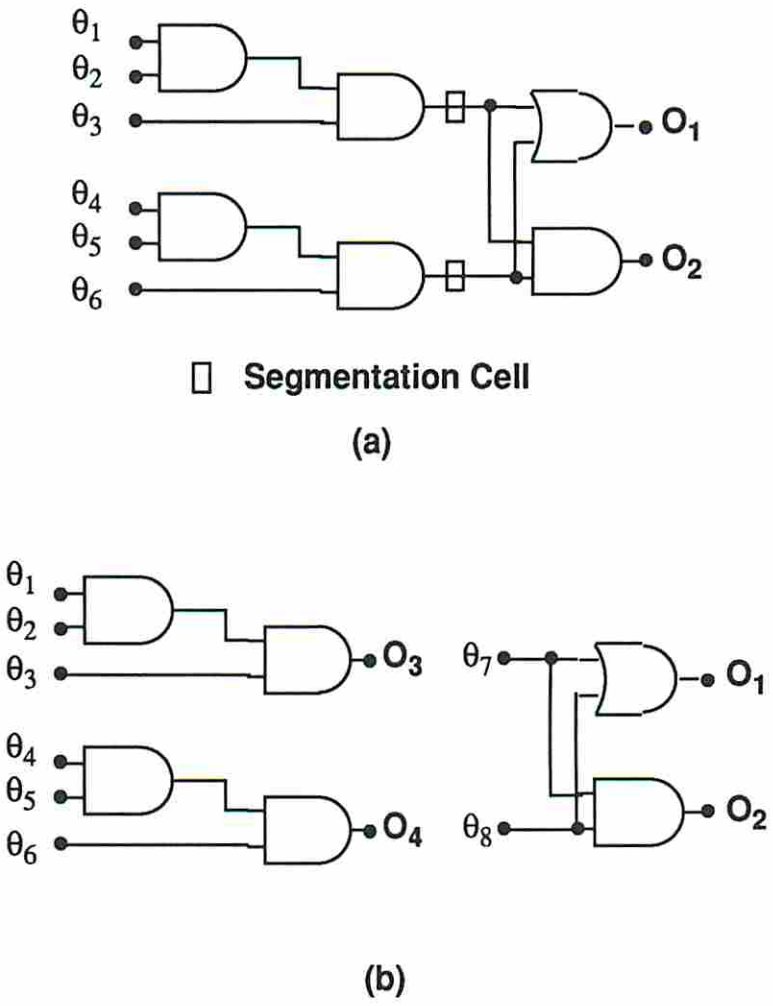


Figure 3.1: A partitioned circuit in (a) normal mode (b) test mode

Example 1 Consider the $(6,2,6)$ circuit shown in Figure 3.1(a). The six inputs are denoted as θ_1 through θ_6 and the two outputs are denoted as O_1 and O_2, respectively. The circuit requires 2^6 patterns for both exhaustive and pseudo-exhaustive testing. Let us consider partitioning the circuit for a cone size limit of three inputs. The circuit is partitioned with two segmentation cells so that each output cone depends on at most three inputs. During the test mode the circuit becomes an $(8,4,3)$ circuit as shown in Figure 3.1(b). The two pseudo-inputs are denoted as θ_7 and θ_8 and the two pseudo-outputs are denoted as O_3 and O_4, respectively. All four output cones are exhaustively tested concurrently with 2^3 patterns. □

3.1.1 Circuit Model

A combinational circuit is modeled as a directed acyclic graph. The nodes of the graph represent the inputs, gates and outputs of the circuit. The edges of the graph represent the fanout branches of the interconnection signals. The fanout stems of the signals are *not* represented explicitly in the model. The input nodes have only fanout edges and the output nodes have only fanin edges. The gate nodes have both fanin and fanout edges. An output cone of a circuit forms a subgraph and two output cones can overlap, i.e. share nodes and edges.

Figure 3.2 shows the graph model of the $(6,2,6)$ circuit in Figure 3.1(a). The input and output nodes are shown as squares and the gate nodes are shown as circles. The nodes are labelled as n_1 through n_{14} and the edges are labelled as e_1 through e_{14}. Input nodes n_1 through n_6 correspond to the circuit inputs θ_1 through θ_6, respectively. Output nodes n_{13} and n_{14} correspond to the circuit outputs O_1 and O_2, respectively.

3.2 Problem Statement

Two interesting problems arise depending on where the segmentation cells are allowed to be placed in a circuit. In the first case, cells are allowed to be placed on the signals (edges) in the circuit. Cells placed on the fanout branches of the same signal can be merged into a single cell. This is referred to as the **edge partitioning problem (EPP)**. In the second case, cells are allowed to be placed only on the gate

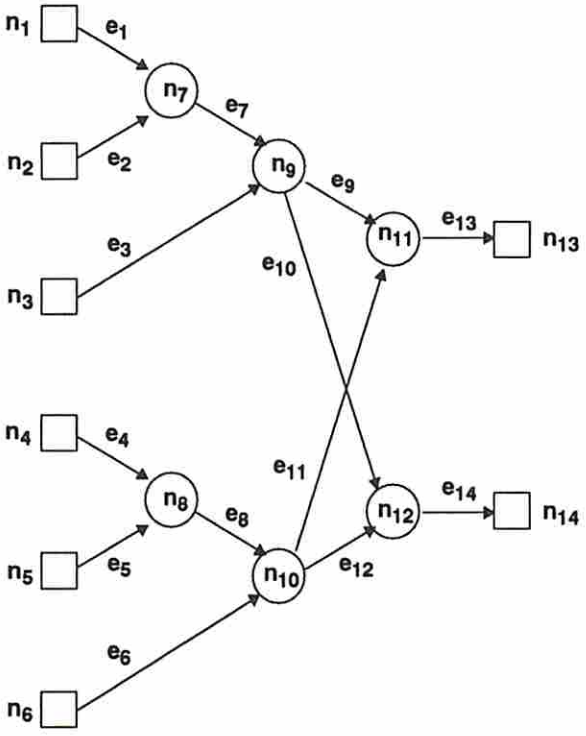


Figure 3.2: Circuit graph

outputs (nodes). This is referred to as the **node partitioning problem (NPP)**. The formal statements of the problems are given below.

3.2.1 Edge Partitioning Problem

Given an (n, m, k) circuit with f being the maximum fanin of any gate and an integer r such that $f \leq r < k$, find an assignment of the minimum number of segmentation cells to the *signals (edges)* such that each output cone depends on at most r inputs in the test mode.

3.2.2 Node Partitioning Problem

Given an (n, m, k) circuit with f being the maximum fanin of any gate and an integer r such that $f \leq r < k$, find an assignment of the minimum number of segmentation cells to the *gate outputs (nodes)* such that each output cone depends on at most r inputs in the test mode.

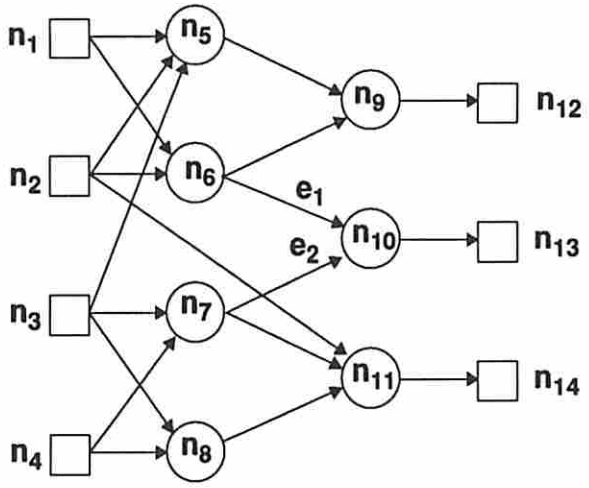


Figure 3.3: EPP and NPP

Example 2 Consider the graph model of the $(4, 3, 4)$ circuit shown in Figure 3.3. The output nodes n_{12}, n_{13} and n_{14} depends on the input node sets $\{n_1, n_2, n_3\}$, $\{n_1, n_2, n_3, n_4\}$ and $\{n_2, n_3, n_4\}$, respectively. Let $r = 3$. For the EPP, the two best solutions require a cell to be placed on the edge e_1 or e_2. For the NPP, the two best

solutions require two cells to be placed on the nodes (1) n_5 and n_6, or (2) n_7 and n_8. The two cases can be explained as follows.

Let us first consider the EPP. Since only n_{13} depends on four input nodes, a cell needs to be placed on either e_1 or e_2. Placing a cell on any of these edges does not affect the input dependencies of the other output nodes. Let us next consider the NPP. To restrict the input dependency of n_{13}, a cell needs to be placed on either n_6 or n_7. However placing a cell on n_6 increases the input dependency of n_{12}. Similarly, placing a cell on n_7 increases the input dependency of n_{14}. Therefore placing a cell on n_6 requires an additional cell to be placed on n_5. Similarly placing a cell on n_7 requires an additional cell on n_8. □

The complexity of the EPP depends on the number of signals, while that of the NPP depends on the number of gates in the circuit. Since the number of signals is more than the number of gates, the solution to the EPP involves a larger search space than the solution to the NPP. However, the EPP forms a generalized case of the NPP and hence yields better solutions as evident by Example 2. The following lemmas characterize both EPP and NPP.

Lemma 1 *For circuits without reconvergent fanouts, both EPP and NPP lead to equally good results.*

Proof : For a circuit without reconvergent fanouts, the fanout branches of any signal always feed different output cones. Consider any signal in the circuit that has fanout branches. Placing a cell on one of the fanout branches feeding some output cone is beneficial to that output cone. However placing the same cell on the fanout stem instead of the fanout branch is beneficial to all the output cones fed by that signal. Hence it is preferable to place the cell on the fanout stem rather than on the fanout branches. Thus the solution to both EPP and NPP involves the same search space. Thus both EPP and NPP lead to equally good results. □

Lemma 2 *For circuits with reconvergent fanouts, EPP leads to better results than NPP.*

Proof : Let us consider a circuit with reconvergent fanouts and an optimal solution for the NPP. This forms a solution for the EPP since the cells are placed on the

fanout stems of the signals. An optimal solution for the EPP can only have fewer or the same number of cells as an optimal solution for the NPP (see example 2). Thus EPP leads to better results than NPP. □

Hence, it is sufficient to consider the NPP for non-reconvergent fanout circuits. Any optimal solution for the NPP can be transformed to a corresponding optimal solution for the EPP and vice-versa. However, if the circuit has reconvergent fanouts, the two-way transformation is not feasible as evident by Example 2.

3.3 Problem Formulation

We have formulated both EPP and NPP as classical integer linear programming (ILP) problems. The **objective function** is to minimize the number of segmentation cells to be placed in the circuit. Let r be the desired cone size limit. The **constraints** are that all the gates in the circuit must depend on at most r inputs during the test mode. The problem formulation is illustrated with the circuit graph shown in Figure 3.2.

3.3.1 Notation

$\vee$ :: logical OR operation

$\wedge$ :: logical AND operation

$\cup$ :: set union operation

θ_i' :: pseudo-input created when a cell is placed on the signal (edge) e_i

I_i :: input dependency set for the signal (edge) e_i

d_i :: input dependency value for the signal (edge) e_i, where $d_i = |I_i|$

$$x_i = \begin{cases} 1 & \text{if a cell is placed on the signal (edge) } e_i, \\ 0 & \text{otherwise.} \end{cases}$$

$$I_i x_i = \begin{cases} I_i & \text{if } x_i \text{ is 1,} \\ \emptyset & \text{otherwise.} \end{cases}$$

$$|I_i x_i| = \begin{cases} d_i & \text{if } x_i \text{ is 1,} \\ 0 & \text{otherwise.} \end{cases}$$

3.3.2 Basic Principle

Consider the example gate and its graph model shown in Figure 3.4. The gate g is represented by the node n and the signals s_1 through s_4 are represented by the edges e_1 through e_4, respectively. The set of inputs feeding n through e_1 is either I_1 or θ_1' depending on whether a segmentation cell is placed on e_1. Thus the set of inputs feeding n through e_1 can be expressed as $(I_1\bar{x}_1 \cup \{\theta_1'\}x_1)$. The expression is valid because of the complementary nature of the boolean variables x_1 and $\bar{x}_1$. The number of inputs feeding n through e_1 is $|I_1\bar{x}_1 \cup \{\theta_1'\}x_1| = d_1\bar{x}_1 + x_1$. Similarly, the set of inputs feeding n through e_2 can be expressed as $I_2\bar{x}_2 \cup \{\theta_2'\}x_2$. The number of inputs feeding n through e_2 is $|I_2\bar{x}_2 \cup \{\theta_2'\}x_2| = d_2\bar{x}_2 + x_2$.

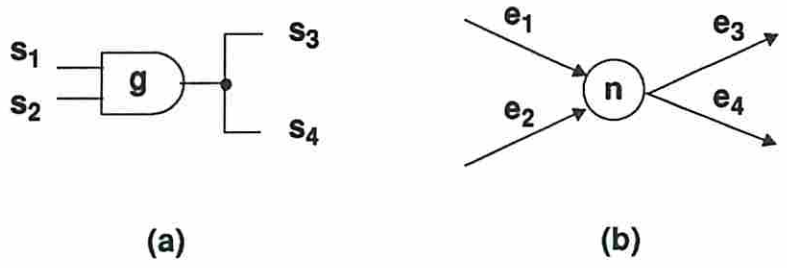


Figure 3.4: (a) An example gate and (b) its graph model

The sets of inputs I_3 and I_4 driving e_3 and e_4 are given by

$$\begin{aligned} I_3 = I_4 &= \{I_1\bar{x}_1 \cup \{\theta_1'\}x_1\} \cup \{I_2\bar{x}_2 \cup \{\theta_2'\}x_2\} \\ &= \{I_1 \cup I_2\}\bar{x}_1\bar{x}_2 \cup \{\{\theta_1'\} \cup I_2\}x_1\bar{x}_2 \cup \\ &\quad \{I_1 \cup \{\theta_2'\}\}\bar{x}_1x_2 \cup \{\theta_1', \theta_2'\}x_1x_2 \end{aligned}$$

and the input dependencies d_3 and d_4 for e_3 and e_4 are given by

$$\begin{aligned} d_3 = d_4 &= |I_1 \cup I_2|\bar{x}_1\bar{x}_2 + |\{\theta_1'\} \cup I_2|x_1\bar{x}_2 + \\ &\quad |I_1 \cup \{\theta_2'\}|\bar{x}_1x_2 + 2x_1x_2 \end{aligned}$$

In the above equation, if one of the product terms is satisfied, the rest of the terms becomes zero. In other words the terms are mutually exclusive. Hence the addition

of the terms is valid. Cells placed on different fanout branches of the same stem can be merged together and replaced with a single cell. For example, the cells placed on the signals s_3 and s_4 can be merged as a single cell and placed on the fanout stem o the gate.

3.3.3 Edge Partitioning Problem

Consider the circuit graph in Figure 3.2. Let $r = 3$. The objective is to minimize the function

$$\begin{aligned} F &= x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + \\ &\quad (x_9 \vee x_{10}) + (x_{11} \vee x_{12}) + x_{13} + x_{14} \end{aligned} \tag{3.1}$$

subject to the constraints $d_i \leq r$, $\forall i = 1, 2, \ldots, 14$. The non-linearity of the objective function is due to the following reason. For the fanouts of n_9, if cells are placed on both e_9 and e_{10}, then the cells can be merged as a single cell and hence the non-linear term $(x_9 \vee x_{10})$ appears in the objective function. The pseudo-inputs created for the fanouts, θ'_9 and θ'_{10}, are the same since the cells are merged. Cells on the fanouts of the primary inputs and the fanins of the primary outputs do not occur in any optimal solution. Hence $x_1 = x_2 \ldots = x_6 = 0$ and $x_{13} = x_{14} = 0$. Therefore the objective function in Equation 3.1 is reduced to

$$F = x_7 + x_8 + (x_9 \vee x_{10}) + (x_{11} \vee x_{12}) \tag{3.2}$$

3.3.3.1 Linearization

Each non-linear term in the objective function can be replaced by a linear term by the following transformation steps.

1. Replace each non-linear term by a new boolean variable. An OR term of the form $x_1 \vee x_2 \vee \ldots \vee x_n$ is replaced by a variable y and an AND term $x_1 \wedge x_2 \wedge \ldots \wedge x_n$ is replaced by z.

2. Add two constraints for each non-linear term. For the OR term, add the constraints $y \leq x_1 + x_2 + \ldots + x_n \leq ny$. For the AND term, add the constraints $nz \leq x_1 + x_2 + \ldots + x_n \leq z + n - 1$.

It can be easily verified that the new variables and the new constraints preserve the original objective function. Hence the non-linear objective function of Equation 3.2 can be replaced by the new linear objective function

$$F = x_7 + x_8 + y_1 + y_2 \tag{3.3}$$

along with four new constraints

$$\begin{aligned} y_1 &\leq x_9 + x_{10} \leq 2y_1 \\ y_2 &\leq x_{11} + x_{12} \leq 2y_2 \end{aligned} \tag{3.4}$$

The input dependency sets I_i and the input dependency values d_i, $\forall i = 1, 2, \ldots, 14$ are computed and listed below.

$$\begin{aligned}
I_1 &= \{\theta_1\};\ I_2 = \{\theta_2\};\ I_3 = \{\theta_3\};\ d_1 = d_2 = d_3 = 1 \\
I_4 &= \{\theta_4\};\ I_5 = \{\theta_5\};\ I_6 = \{\theta_6\};\ d_4 = d_5 = d_6 = 1 \\
I_7 &= I_1 \cup I_2 = \{\theta_1, \theta_2\};\ d_7 = 2 \\
I_8 &= I_4 \cup I_5 = \{\theta_4, \theta_5\};\ d_8 = 2 \\
I_9 &= I_{10} = \{I_7\bar{x}_7 \cup \{\theta'_7\}x_7\} \cup \{I_3\} \\
&= \{\{\theta_1, \theta_2\}\bar{x}_7 \cup \{\theta'_7\}x_7\} \cup \{\theta_3\} \\
&= \{\theta_1, \theta_2, \theta_3\}\bar{x}_7 \cup \{\theta_3, \theta'_7\}x_7 \\
d_9 &= d_{10} = 3\bar{x}_7 + 2x_7 \\
I_{11} &= I_{12} = \{I_8\bar{x}_8 \cup \{\theta'_8\}x_8\} \cup \{I_6\} \\
&= \{\{\theta_4, \theta_5\}\bar{x}_8 \cup \{\theta'_8\}x_8\} \cup \{\theta_6\} \\
&= \{\theta_4, \theta_5, \theta_6\}\bar{x}_8 \cup \{\theta_6, \theta'_8\}x_8 \\
d_{11} &= d_{12} = 3\bar{x}_8 + 2x_8 \\
I_{13} &= \{I_9\bar{x}_9 \cup \{\theta'_9\}x_9\} \cup \{I_{11}\bar{x}_{11} \cup \{\theta'_{11}\}x_{11}\} \\
&= \{I_9 \cup I_{11}\}\bar{x}_9\bar{x}_{11} \cup \{\{\theta'_9\} \cup I_{11}\}x_9\bar{x}_{11} \cup \\
&\quad \{I_9 \cup \{\theta'_{11}\}\}\bar{x}_9x_{11} \cup \{\theta'_9, \theta'_{11}\}x_9x_{11} \\
&= (\{\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6\}\bar{x}_7\bar{x}_8 \cup \{\theta_1, \theta_2, \theta_3, \theta_6, \theta'_8\}\bar{x}_7x_8 \cup \\
&\quad \{\theta_3, \theta_4, \theta_5, \theta_6, \theta'_7\}x_7\bar{x}_8 \cup \{\theta_3, \theta_6, \theta'_7, \theta'_8\}x_7x_8)\bar{x}_9\bar{x}_{11} \cup
\end{aligned}$$

$$
\begin{aligned}
& (\{\theta_4, \theta_5, \theta_6, \theta'_9\}\bar{x}_8 \cup \{\theta_6, \theta'_8, \theta'_9\}x_8)x_9\bar{x}_{11} \cup \\
& (\{\theta_1, \theta_2, \theta_3, \theta'_{11}\}\bar{x}_7 \cup \{\theta_3, \theta'_7, \theta'_{11}\}x_7)\bar{x}_9 x_{11} \cup \\
& \{\theta'_9, \theta'_{11}\}x_9 x_{11} \\
d_{13} &= (6\bar{x}_7\bar{x}_8 + 5\bar{x}_7 x_8 + 5x_7\bar{x}_8 + 4x_7 x_8)\bar{x}_9\bar{x}_{11} + \\
& (4\bar{x}_8 + 3x_8)x_9\bar{x}_{11} + (4\bar{x}_7 + 3x_7)\bar{x}_9 x_{11} + 2x_9 x_{11} \\
I_{14} &= \{I_{10}\bar{x}_{10} \cup \{\theta'_{10}\}x_{10}\} \cup \{I_{12}\bar{x}_{12} \cup \{\theta'_{12}\}x_{12}\} \\
&= \{I_{10} \cup I_{12}\}\bar{x}_{10}\bar{x}_{12} \cup \{\{\theta'_{10}\} \cup I_{12}\}x_{10}\bar{x}_{12} \cup \\
& \{I_{10} \cup \{\theta'_{12}\}\}\bar{x}_{10}x_{12} \cup \{\theta'_{10}, \theta'_{12}\}x_{10}x_{12} \\
&= (\{\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6\}\bar{x}_7\bar{x}_8 \cup \{\theta_1, \theta_2, \theta_3, \theta_6, \theta'_8\}\bar{x}_7 x_8 \cup \\
& \{\theta_3, \theta_4, \theta_5, \theta_6, \theta'_7\}x_7\bar{x}_8 \cup \{\theta_3, \theta_6, \theta'_7, \theta'_8\}x_7 x_8)\bar{x}_{10}\bar{x}_{12} \cup \\
& (\{\theta_4, \theta_5, \theta_6, \theta'_{10}\}\bar{x}_8 \cup \{\theta_6, \theta'_8, \theta'_{10}\}x_8)x_{10}\bar{x}_{12} \cup \\
& (\{\theta_1, \theta_2, \theta_3, \theta'_{12}\}\bar{x}_7 \cup \{\theta_3, \theta'_7, \theta'_{12}\}x_7)\bar{x}_{10}x_{12} \cup \\
& \{\theta'_{10}, \theta'_{12}\}x_{10}x_{12} \\
d_{14} &= (6\bar{x}_7\bar{x}_8 + 5\bar{x}_7 x_8 + 5x_7\bar{x}_8 + 4x_7 x_8)\bar{x}_{10}\bar{x}_{12} + \\
& (4\bar{x}_8 + 3x_8)x_{10}\bar{x}_{12} + (4\bar{x}_7 + 3x_7)\bar{x}_{10}x_{12} + 2x_{10}x_{12}
\end{aligned}
$$

Since $r = 3$, from the expression for d_{13} we have the constraints

$$\bar{x}_9\bar{x}_{11} = 0; \quad \bar{x}_8 x_9\bar{x}_{11} = 0; \quad \bar{x}_7\bar{x}_9 x_{11} = 0 \tag{3.5}$$

Similarly, from the expression for d_{14} we have the constraints

$$\bar{x}_{10}\bar{x}_{12} = 0; \quad \bar{x}_7\bar{x}_{10}x_{12} = 0; \quad \bar{x}_8 x_{10}\bar{x}_{12} = 0 \tag{3.6}$$

The constraints in Equations 3.5 and 3.6 are linear as evident by their dual interpretation. Consider the constraints in Equation 3.5. The dual of the first constraint $\bar{x}_9\bar{x}_{11} = 0$ is given by the linear constraint $x_9 + x_{11} \geq 1$. This constraint implies that at least one cell needs to be placed on either e_9 or e_{11}; otherwise $d_{13} > r$.

For the ILP problem with the objective function given by Equation 3.3 and the set of linear constraints given by Equations 3.4, 3.5 and 3.6, an optimal solution is given by

$$\begin{aligned} x_7 &= x_8 = 0; \quad x_9 = x_{10} = x_{11} = x_{12} = 1 \\ F &= 2 \end{aligned}$$

3.3.4 Node Partitioning Problem

For the NPP, since the cells are allowed to be placed only on the fanout stem of signals, the variables associated with the fanout branches of a signal are considered the same. Hence $x_9 = x_{10}$ and $x_{11} = x_{12}$. The objective function for the NPP is given by

$$F = x_7 + x_8 + x_9 + x_{11} \tag{3.7}$$

and the constraints are given by Equation 3.5. Thus for NPP, the objective function is linear with a reduced set of linear constraints. For the ILP problem with the objective function given by Equation 3.7 and the set of linear constraints given by Equation 3.5, an optimal solution is given by

$$\begin{aligned} x_7 &= x_8 = 0; \quad x_9 = x_{11} = 1 \\ F &= 2 \end{aligned}$$

The solution implies that two segmentation cells need to be placed at the fanout stems of n_9 and n_{10}. Lemma 1 is applicable since the circuit graph has no reconvergent fanouts. Thus we have identical optimal solutions for both EPP and NPP. The details of the formulation can also be found in [33].

3.3.5 Problem Complexity

The number of nodes and edges that need to be considered for the ILP formulation can be reduced to some extent. Later we will show that certain nodes and edges need not be considered as they are guaranteed to be absent in any optimal solution.

In spite of this reduction, the ILP formulation is not feasible for large circuits. This is due to the fact that the number of constraints grows non-linearly with the number of levels in the circuit. Thus the formulation can be used to obtain optimal solutions only for reasonably small circuits.

Since the complexity of the NPP is less than that of the EPP, henceforth we shall only concentrate on the NPP. Our approach can be easily extended for the EPP.

3.4 Test Mode Configuration

The details of a segmentation cell are shown in Figure 3.5. Consider a cell being placed at the output of gate G as shown in the figure. The cell is composed of a MISR (multiple input signature register) cell for compacting responses for the pseudo-output and a SR (shift register) cell for generating patterns for the pseudo-input. A multiplexer is used to select the output of G during the normal mode and the pseudo-input from the SR cell during the test mode. The test mode configuration of a circuit partitioned with two segmentation cells is shown in Figure 3.6.

Consider a combinational circuit C being driven by an input register $R1$ and driving an output register $R2$. Let us assume that C is partitioned with two segmentation cells $s1$ and $s2$ as shown in Figure 3.6. During the test mode, $R1$ is modified as a TPG and $R2$ is modified as a SA for the circuit. The SR cells of $s1$ and $s2$ are configured as part of the TPG as shown in the figure. Similarly, the MISR cells of $s1$ and $s2$ form the part of the SA. The output cones of the partitioned circuit can be exhaustively tested either in a single test session or in multiple test sessions.

3.5 Partitioning Special Circuits

We shall consider the following classes of combinational circuits and develop efficient partitioning strategies for restricting their maximum cone sizes to desired values.

- Circuits without any fanouts.
- Circuits without any reconvergent fanouts.

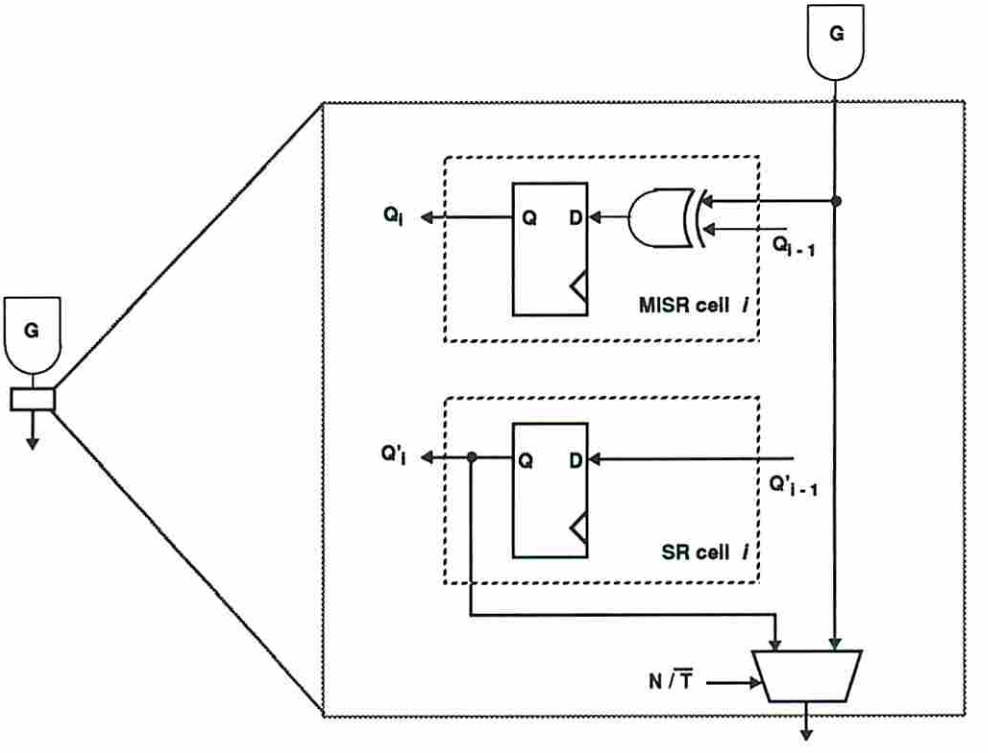


Figure 3.5: A segmentation cell

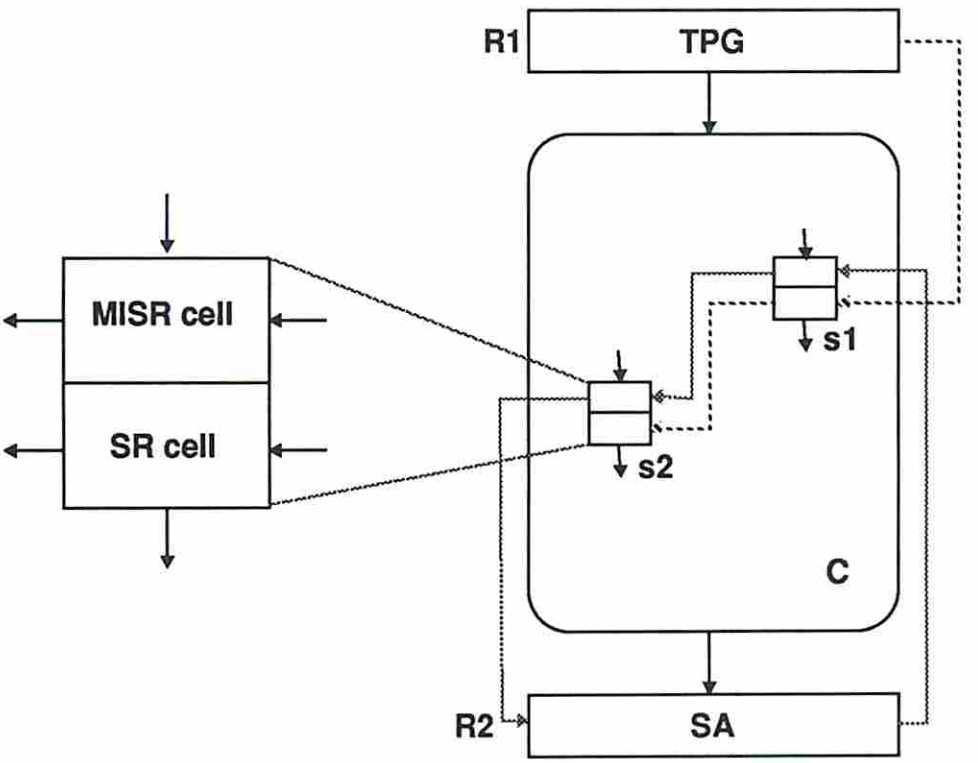


Figure 3.6: Test mode configuration of a partitioned circuit.

- Circuits with reconvergent fanouts.
- Circuits with two levels of gates.
- Circuits with multiple levels of gates.
- Circuits with iterative logic array structures.

The strategies take advantage of the unique structural properties of these circuits.

3.5.1 Definitions

We shall present a few definitions that are illustrated using the circuit graph shown in Figure 3.2. The sets of nodes $\{n_1, n_2, \ldots, n_6\}$, $\{n_7, n_8, \ldots, n_{12}\}$ and $\{n_{13}, n_{14}\}$ represent the sets of inputs, gates and outputs of the circuit, respectively.

Node n_i is said to **feed** node n_j (node n_j is said to be **fed by** node n_i) if there exists a directed path from node n_i to node n_j. The set of all nodes that feed a node is called its **fanin cone** and the set of all nodes that are fed by the node is called its **fanout cone**. The fanin and fanout cones of node n_{10} are $\{n_4, n_5, n_6, n_8\}$ and $\{n_{11}, n_{12}, n_{13}, n_{14}\}$, respectively. The set of all inputs that feed a node is called its **(input) dependency set**. The **(input) dependency** of node n_i, denoted as d_i, is given by the cardinality of its dependency set. The dependency set of node n_{10} is $\{n_4, n_5, n_6\}$ and $d_{10} = 3$.

An **articulation node** [10] is a node that is contained in all paths originating from its fanin cone and ending in its fanout cone. Node n_{10} is an articulation node since it is contained in all paths originating from its fanin cone $\{n_4, n_5, n_6, n_8\}$ and ending in its fanout cone $\{n_{11}, n_{12}, n_{13}, n_{14}\}$.

Consider two nodes n_i and n_j such that n_j is in the fanout cone of n_i. There may exist some inputs that feed n_j only via n_i. The number of such inputs is called the **articulation value** of n_i with respect to n_j and denoted as $a_{i,j}$. For example, $a_{10,11} = 3$ since the input nodes $\{n_4, n_5, n_6\}$ feed n_{11} only via n_{10}.

For a specified cone size limit, the nodes with dependencies no greater than the cone size limit are called the **good nodes**. The remaining nodes with dependencies greater than the limit are called the **bad nodes**. For the cone size limit of three, the good and the bad nodes for the circuit graph are $\{n_1, n_2, \ldots, n_{10}\}$ and

$\{n_{11}, n_{12}, n_{13}, n_{14}\}$, respectively. The set of bad nodes that are directly fed by at least one good node is called the **envelope**. For our example, the set $\{n_{11}, n_{12}\}$ forms the envelope.

Consider a good node n_i such that it is not an input node and feeds at least one bad node. The node n_i is called a **candidate node** if there exists no good node n_j such that n_j is contained in all paths from n_i to all bad nodes in the fanout cone of n_i. For example, node n_8 is not a candidate node since the good node n_{10} is contained in all paths from n_8 to bad nodes $\{n_{11}, n_{12}, n_{13}, n_{14}\}$ in the fanout cone of n_8. On the contrary, nodes n_9 and n_{10} are candidate nodes.

3.5.2 Pruning Search Space

Our goal is to partition a given circuit using the minimum number of segmentation cells in order to restrict the maximum cone size of the circuit to some desired value. We shall present a few observations that help in pruning the search involved in the solution space.

Observation 1 *A circuit graph with no reconvergence among paths contains only articulation nodes.*

Observation 2 *The articulation value of an articulation node n_i with respect to any node in its fanout cone equals the input dependency of node n_i.*

The circuit graph shown in Figure 3.2 contains only articulation nodes as there is no reconvergence among paths. Consider the articulation node n_{10} in the circuit graph. The articulation value of n_{10} with respect to any node in its fanout cone equals d_{10}.

Observation 3 *Placing a segmentation cell on node n_i decreases the input dependency of node n_j in the fanout cone of n_i by $(a_{i,j} - 1)$.*

Consider two nodes n_i and n_j such that n_j is in the fanout cone of n_i. Placing a cell on node n_i restricts the dependencies of all inputs that feed n_j only via n_i. In addition, the cell creates a pseudo-input at node n_i. Thus the input dependency of node n_j gets decreased by $(a_{i,j} - 1)$.

The following results identify a subset of nodes in the circuit that need not be considered for placing segmentation cells. Placing cells on these nodes does not lead to an optimal solution. Thus the following results help prune the search space.

Lemma 3 *The fanouts of any input node and the fanin of any output node need not be considered for placing segmentation cells.*

Lemma 4 *Let node n_i be an articulation node such that its input dependency is no greater than the cone size limit. Let all nodes in the fanin cone of n_i feed only the nodes in the fanout cone of n_i. Then the nodes in the fanin cone of n_i need not be considered for placing segmentation cells.*

Proof : Consider node n_j in the fanin cone of the articulation node n_i. Both n_i and n_j feed the same set of bad nodes. Placing a segmentation cell on n_i reduces the dependencies of all bad nodes in the fanout cone of n_i by $(d_i - 1)$. On the contrary, placing a segmentation cell on n_j reduces the dependencies of all bad nodes in the fanout cone of n_i by only at most $(d_j - 1)$. Since $d_i \geq d_j$, it is always better to place a cell on n_i than on n_j. Hence, it is not necessary to consider the nodes in the fanin cone of n_i for placing segmentation cells. □

Lemma 5 *Non-candidate good nodes need not be considered for placing segmentation cells.*

Proof : Consider a good node n_j that is not a candidate node and feeds at least one bad node. There must exist a candidate node n_i such that n_i is contained in all paths from n_j to any bad node n_k. In other words, both n_i and n_j feed the same set of bad nodes. Placing a segmentation cell on n_i decreases d_k by $(a_{i,k} - 1)$. Similarly, placing a segmentation cell on n_j decreases d_k by $(a_{j,k} - 1)$. Since $a_{i,k} \geq a_{j,k}$ for any bad node n_k, it is always better to place a cell on n_i than on n_j. Hence non-candidate good nodes need not be considered for placing the segmentation cells. □

It should be noted that the sets of good nodes, bad nodes, envelope and candidate nodes are dynamic, i.e. they change as segmentation cells are added to a given circuit. To satisfy a desired cone size limit, cells are placed sequentially, i.e. one at a time in the circuit till all nodes become good nodes. Nodes that are not considered for cell placement due to Lemma 5 *will* not affect the optimality of any solution. In other words, an optimal solution can be obtained without considering these nodes.

3.5.3 Fanout-free Circuits

A circuit without any fanouts can be trivially partitioned into disjoint subcircuits, where each subcircuit has only one output. The graph model of each subcircuit forms a directed tree structure consisting of only articulation nodes. We shall present a partitioning strategy for determining the optimal number of segmentation cells required to satisfy the desired cone size limit for such circuits.

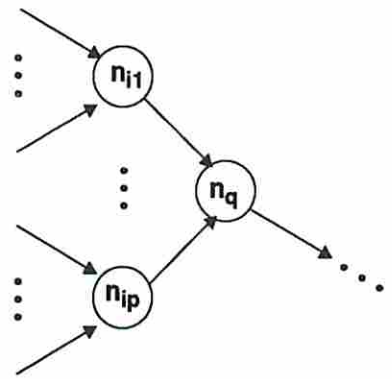


Figure 3.7: A portion of the graph model of a multi-level fanout-free circuit

Consider a portion of the graph model of a multi-level fanout-free circuit shown in Figure 3.7. Let the nodes be assigned unique indices such that a node has a lower index than all the nodes in its fanout cone. Let n_q be the bad node with the minimum index. Let the good nodes $S = \{n_{i_1}, n_{i_2}, \ldots, n_{i_p}\}$ directly feed n_q as shown in the figure. Among all the nodes in the fanin cone of n_q, it is sufficient to consider only the nodes in S as candidates for placing segmentation cells in order to reduce d_q (as per Lemma 4). This basic principle is used in the partitioning procedure *FanFreeCkt* to determine the optimal number of segmentation cells required to satisfy the cone size limit.

For a two-level fanout-free circuit, the nodes in S form the first-level nodes and n_q forms the second-level node. Hence all the nodes in S form the candidates for placing cells. Procedure *FanFreeCkt* can be used for two-level fanout-free circuits to obtain optimal solutions.

Definition 1 (Cormen 90) *(Elements of Greedy Strategy)*

- *A problem exhibits* **greedy choice** *property if a global optimal solution can be arrived at by making a locally optimal (greedy) choice.*
- *A problem exhibits* **optimal substructure** *property if an optimal solution to the problem contains within it optimal solutions to subproblems.*

The greedy choice and optimal substructure properties are the two ingredients that are exhibited by most problems that lend themselves to a greedy strategy [9]. The optimal substructure property is exploited by both greedy and dynamic programming strategies. A greedy strategy can be applied to the partitioning problem for fanout-free circuits due to the following lemma.

Lemma 6 *The partitioning problem for fanout-free circuits exhibits both greedy choice and optimal substructure properties.*

Proof : Consider the portion of a fanout-free circuit graph shown in Figure 3.7. The nodes in $S = \{n_{i_1}, n_{i_2}, \ldots, n_{i_p}\}$ directly feed the bad node n_q. The benefit b_{i_j} of placing a cell on node $n_{i_j} \in S$ is given by $(d_{i_j} - 1)$ since the cell reduces the dependencies of all nodes in the fanout cone of n_{i_j} by $(d_{i_j} - 1)$. The benefits of nodes in S are *independent* of each other. Assume that the nodes in S are sorted in the non-increasing order of their benefits. A cell is placed on n_{i_1} for reducing d_q to satisfy the cone size limit (say r). If required, a second cell is placed on n_{i_2}, a third cell is placed on n_{i_3} and so on until $d_q \leq r$.

Let $L \subseteq S$ be the set of nodes selected for placing cells to reduce d_q. We shall show that L is a local optimum solution for reducing d_q. Consider a local optimal solution $L^* \subseteq S$. Let the nodes in L^* be ordered according to their benefits and assume that node $n^*_{i_1} \in L^*$ has the maximum benefit among all nodes in L^*. We know that node $n_{i_1} \in S$ has the maximum benefit among all nodes in S and $n_{i_1} \in L$. If $n^*_{i_1} \neq n_{i_1}$, then there exists another optimal solution $\hat{L}^* = L^* - \{n^*_{i_1}\} + \{n_{i_1}\}$ since $|\hat{L}^*| = |L^*|$ and $b_{i_1} \geq b^*_{i_1}$. Repeating this argument for the remaining nodes in L, we can transform L^* to include all the nodes of L. Hence L is a local optimal solution for reducing d_q.

Let G be a set of nodes selected for placing cells representing a global optimum solution. Let L be a local optimum solution for reducing d_q. Assume G contains

a local non-optimal solution L' for restricting the dependency of node n_q. Since L' is non-optimal, $|L'| > |L|$. The set of nodes $G' = (G - L' \cup L)$ is another global solution with $|G'| < |G|$ contradicting the optimality of G. Thus any global optimum solution contains local optimal solutions. □

Procedure FanFreeCkt

Input: Circuit graph and cone size limit r.
Output: Partitioned circuit satisfying the cone size limit.

1. Assign unique index to each node such that the node has a lower index than all the nodes in its fanout cone.
2. While there exists a bad node do
   (a) Select the bad node (say n_q) with the minimum index.
   (b) Determine the good nodes $S = \{n_{i_1}, n_{i_2}, \ldots, n_{i_p}\}$ that directly feed node n_q.
   (c) Determine the benefit b_{i_j} of placing a segmentation cell on node n_{i_j}, $\forall n_{i_j} \in S$.
   $b_{i_j} \leftarrow d_{i_j} - 1$ for $n_{i_j} \in S$.
   (d) Sort the nodes in S in the non-increasing order of their benefits.
   Let $b_{i_1} \geq b_{i_2} \geq \ldots \geq b_{i_p}$.
   (e) $j \leftarrow 1$.
   (f) While $d_q > r$ do
      i. Place a segmentation cell on n_{i_j}.
      ii. Update the dependencies of all nodes in the fanout cone of n_{i_j}.
      iii. $j \leftarrow j + 1$.

Theorem 1 *The partitioning procedure for fanout-free circuits determines the optimal number of segmentation cells to satisfy the cone size limit and is of polynomial complexity.*

Proof : The outer *while* loop in the procedure selects the bad node n_q with the minimum index so that during the process of reducing d_q, the dependencies of some of the other bad nodes may also get reduced. The benefits of the nodes in S are *independent* of each other and hence the sorted order of benefits is valid during every iteration of the inner *while* loop. The node with the best possible benefit is selected during every iteration of the inner *while* loop. The procedure selects the optimal number of cells for reducing d_q to satisfy the cone size limit since the partitioning problem exhibits the *greedy choice property* (refer to Lemma 6).

Every iteration of the outer *while* loop selects an *independent* subproblem of restricting the dependency of the bad node n_q with the minimum index and determines an local optimum solution for the subproblem. The procedure determines a global optimal solution since the partitioning problem exhibits the *optimal substructure property* (refer to Lemma 6).

The complexity of assigning unique indices to nodes is linear in terms of the number of nodes in the circuit graph. The assignment is done by traversing the circuit graph in breadth first search manner. The main computation step involves sorting the nodes in S during every iteration of the outer *while* loop. Thus the complexity of the procedure is $O(N^2\ log\ N)$, where N is the number of nodes in the circuit graph. □

3.5.4 Non-Reconvergent Fanout Circuits

A circuit with non-reconvergent fanouts has multiple outputs and the circuit graph consists of only articulation nodes. Lemma 7 states the characteristics of non-reconvergent fanout circuits that makes it difficult to obtain optimal solutions for the partitioning problem. Theorem 2 states that the partitioning problem for non-reconvergent fanout circuits requires a procedure of exponential complexity to determine an optimal solution. We shall present an efficient partitioning strategy for determining a minimal number of segmentation cells required to satisfy a desired cone size limit.

Lemma 7 *The partitioning problem for non-reconvergent fanout circuits does not exhibit the optimal substructure property.*

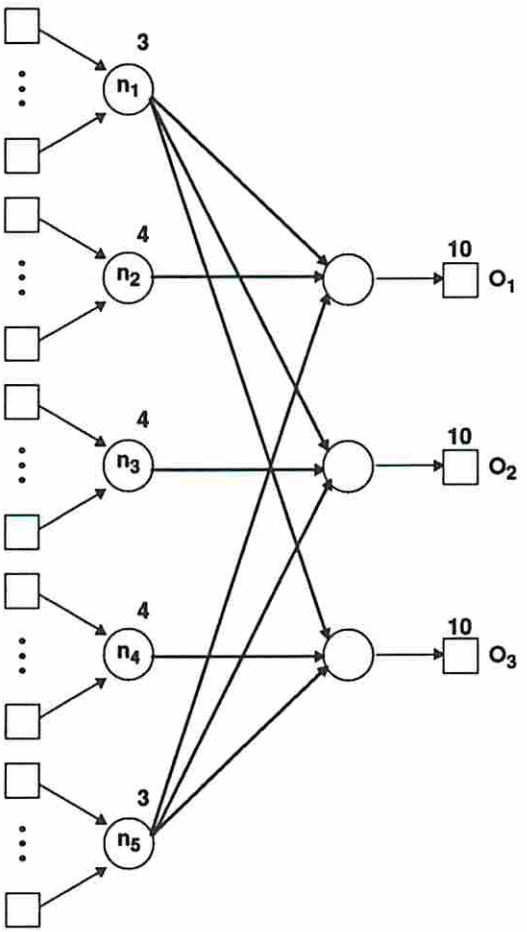


Figure 3.8: Graph model of a two-level non-reconvergent fanout circuit

Proof : *(by example:)* Consider the two-level non-reconvergent fanout circuit shown in Figure 3.8. The first-level nodes n_1 through n_5 have input dependencies of $3, 4, 4, 4$ and 3, respectively. Each of the output nodes O_1, O_2 and O_3 has a dependency of ten inputs. The dependencies of the first-level and output nodes are shown in the figure. Let us consider partitioning the circuit for a cone size limit of seven inputs. The global optimal solution is given by placing two segmentation cells on nodes n_1 and n_5. The local optimum solutions for output cones O_1, O_2 and O_3 are obtained by placing segmentation cells on nodes n_2, n_3 and n_4, respectively. Thus the global optimal solution does not contain the local optimum solutions. □

Theorem 2 (Bhatt86) *The partitioning problem for non-reconvergent fanout circuits is NP-complete.*

It has been proved in [6] that the partitioning problem for fanout circuits is NP-complete. The example circuit considered in the proof is a two-level non-reconvergent fanout circuit.

Procedure *NonReconvFanCkt* describes our partitioning strategy for placing a minimal number of segmentation cells in the circuit. The heuristic procedure is of polynomial complexity and yields good suboptimal solutions. The procedure is applicable to both two-level and multi-level non-reconvergent fanout circuits.

Procedure NonReconvFanCkt

Input: Circuit graph and cone size limit r.
Output: Partitioned circuit satisfying the cone size limit.

1. While there exists a bad node do
   (a) Determine the set (say S) of candidate nodes.
   (b) For each node $n_i \in S$, determine the set of output nodes S_i fed by n_i.
   (c) For each node $n_i \in S$, determine the benefit b_i of placing a cell on n_i.
   $b_i \leftarrow \sum_{O_x \in S_i} max\ \{min\ \{d_i - 1, d'_x - r\}, 0\ \}$
   where d'_x is the dependency for output node $O_x \in S_i$.

(d) Determine the node $n_i^* \in S$ that has the maximum benefit and place a cell on n_i^*.

(e) Update the dependencies of all the nodes in the fanout cone of n_i^*.

A good node n_i is a candidate node if it satisfies at least one of the following conditions: (1) n_i directly feeds a bad node or (2) n_i has multiple fanouts and feeds a bad node. The benefit of placing a segmentation cell on n_i is determined as follows. Let O_x be an output node with dependency d'_x and driven by n_i. Placing a cell on n_i reduces the dependency of O_x by $(d_i - 1)$. However, the dependency of O_x needs to be reduced only by the amount $(d'_x - r)$ to satisfy the cone size limit. With respect to O_x, the benefit of placing a cell on n_i is measured as the minimum value between $(d_i - 1)$ and $(d'_x - r)$. If d'_x is already less than r, then there is no benefit for O_x by placing the cell on n_i. Thus the benefit of placing a cell on n_i with respect to O_x is given by the expression $max\ \{min\ \{d_i - 1, d'_x - r\}, 0\ \}$. If n_i is a fanout node, then the benefit of placing a cell on n_i is measured as the cumulative sum of individual benefits for all output nodes driven by n_i.

Among the nodes in S, the node with the maximum benefit is selected for placing segmentation cell. Since every iteration of the *while* loop results in the transformation of some bad nodes to good nodes, the candidates are determined during every iteration. The main computation step involves determining the benefits of the nodes in S during every iteration of the *while* loop. The complexity of the procedure is $O(N^3)$, where N is the number of nodes in the circuit graph.

3.5.5 Reconvergent Fanout Circuits

A circuit with reconvergent fanouts can have either single or multiple outputs. All the nodes in the circuit graph need not necessarily form articulation nodes. Lemma 8 highlights the contrast among the characteristics of circuits with and without reconvergent fanouts.

Lemma 8 *Let n_k be a common node in the fanout cones of nodes n_i and n_j. Let the articulation values of n_i and n_j with respect to n_k be $a_{i,k}$ and $a_{j,k}$, respectively. If*

the circuit has (no) reconvergent fanouts, then the decrease in d_k due to placement of two segmentation cells on n_i and n_j can (not) be greater than $(a_{i,k} + a_{j,k} - 2)$.

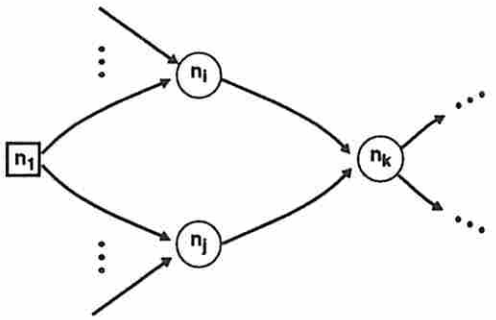


Figure 3.9: A portion of the graph model of a multi-level reconvergent fanout circuit

Proof : *Case 1: (Circuit with reconvergent fanouts)* Consider a portion of the graph model of a multi-level circuit with reconvergent fanout as shown in Figure 3.9. Assume that nodes n_i and n_j are directly driven by the input node n_1. Let n_i and n_j directly feed node n_k as shown in the figure. Node n_k is dependent on input node n_1 even after placing a segmentation cell on either n_i or n_j. In other words, the articulation values $a_{i,k}$ and $a_{j,k}$ do not account for input node n_1. Placing two cells on n_i and n_j makes n_k no longer dependent on the input node n_1 and d_k gets decreased by $(a_{i,k} + a_{j,k} - 2 + 1)$.
Case 2: (Circuit without reconvergent fanouts) Since the circuit graph contains only articulation nodes, each node has at most one directed path from each input node. Therefore placing two cells on n_i and n_j decreases d_k by $(a_{i,k} + a_{j,k} - 2)$. □

Due to this characteristics of reconvergent fanout circuits, multiple nodes need to be considered simultaneously for placement of segmentation cells in order to measure the true benefits. Hence it involves an exponential complexity procedure to determine optimal solutions.

For partitioning two-level reconvergent fanout circuits, procedure *NonReconvFanCkt* can be modified to obtain good suboptimal solutions. In the procedure, consider the expression for the benefit b_i of placing a cell on a candidate node n_i with respect to an output node O_x. Since all nodes are articulation nodes for a non-reconvergent fanout circuit, the articulation value of n_i with respect to O_x is given

by d_i. Hence the value d_i is used in the expression. For a two-level reconvergent fanout circuit, a candidate node n_i need not be an articulation node. Hence the value d_i used in the benefit expression must be replaced by the articulation value of n_i with respect to O_x. The same procedure can then be used for partitioning two-level reconvergent fanout circuits to determine a minimal number of cells required to satisfy the cone size limit.

For multi-level reconvergent fanout circuits, procedure *NonReconvFanCkt* does not yield good suboptimal solutions due to the following reason. The benefit of placing a cell on a candidate node with respect to any output node in its fanout cone tends to be zero due to reconvergence among paths. Hence we need to adopt some other heuristic measure for the candidate nodes. We shall describe an efficient heuristic procedure of polynomial complexity in the next section.

3.5.6 Iterative Logic Arrays

3.5.6.1 One-dimensional Arrays

Consider the one-dimensional iterative logic array (ILA) shown in Figure 3.10. The array is composed of m identical cells referred to as *acells*. Each cell has p top inputs and q left inputs. Similarly, each cell has p bottom outputs and q right outputs as shown in the figure. Let the acells be indexed from 1 through m, respectively. The one-dimensional ILA is formed by feeding the right outputs of the ith acell to the left inputs of the $(i+1)$st acell. The one-dimensional ILA can be characterized with three parameters m, p and q. Theorem 3 states the optimal number of segmentation cells required for partitioning the array structure for a specified cone size limit. We shall refer to the segmentation cells as *scells* and assume that placement of scells is prohibited inside any acell.

Theorem 3 *Let r be the cone size limit for the one-dimensional ILA shown in Figure 3.10. Let i satisfy the following constraint*

$$pi + q \leq r < p(i+1) + q \tag{3.8}$$

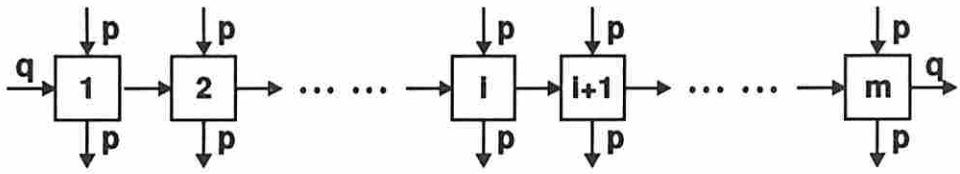


Figure 3.10: One-dimensional ILA

Let s satisfy the following relation

$$s = q(\lceil m/i \rceil - 1) \tag{3.9}$$

Then s scells are necessary and sufficient for partitioning the ILA so that each output is driven by at most r inputs.

Proof : Consider the unpartitioned ILA shown in Figure 3.10. The ith and $(i+1)$st acells are dependent on $(pi + q)$ and $(p(i + 1) + q)$ inputs, respectively. As per the constraint given by Equation 3.8, the dependency of $(i + 1)$st acell exceeds the cone size limit r. The one-dimensional ILA forms a multi-level fanout-free circuit and hence each acell forms an articulation node. Hence by Lemma 4, only the inputs to the $(i + 1)$st acell need to be considered for placing scells. It is therefore necessary to place q scells after the ith acell as shown in Figure 3.11 in order to restrict the dependency for the $(i + 1)$st acell.

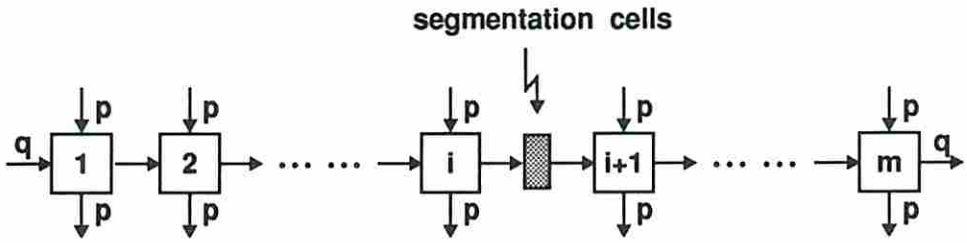


Figure 3.11: Partitioned one-dimensional ILA

Analyzing the dependencies of the acells with indices greater than i, it is evident that a second set of q scells have to be placed at the outputs of the $2i$th acell in order to restrict the dependency for the $(2i + 1)$st acell. The partitioning process is

repeated till the end of the array structure. Thus the total number of scells placed in the ILA is given by the expression provided by Equation 3.9. These scells are necessary since they are being placed at indispensible locations and are sufficient since each output is driven by at most r inputs in the partitioned ILA. □

It should be noted that the circuit graph of an one-dimensional ILA is a binary tree structure and hence forms a special case of fanout-free circuit. Procedure *FanFreeCkt* described for fanout-free circuits can be used to obtain optimal solutions for partitioning one-dimensional ILA structures. The procedure places the optimal number of scells at indispensible locations described in the proof of Theorem 3.

3.5.6.2 Two-dimensional Arrays

Consider the two-dimensional ILA shown in Figure 3.12. Let the array be composed of n identical rows with each row consisting of m identical acells. Let the acells be indexed from $(1,1)$ through (n,m) as shown in the figure. Each acell has p top inputs and q left inputs. Similarly, each cell has p bottom outputs and q right outputs as shown in the figure. The two-dimensional ILA is formed by feeding the bottom outputs of the acell (i,j) to the top inputs of the acell $(i+1,j)$ and by feeding the right outputs of the acell (i,j) to the left inputs of the acell $(i,j+1)$. The two-dimensional ILA can be characterized with four parameters m, n, p and q. Theorem 4 states an upper bound on the number of scells required for partitioning the array structure for a specified cone size limit.

Theorem 4 *Let r be the cone size limit for the two-dimensional ILA shown in Figure 3.12. Let $i_1 \geq i_2 \geq \ldots \geq i_x \geq 1$ satisfy the following row constraints*

$$\{pi_\alpha + q\alpha\} \leq r < \{p(i_\alpha + 1) + q\alpha\} \quad \forall \alpha = 1, 2, \ldots, x. \tag{3.10}$$

Let $j_1 \geq j_2 \geq \ldots \geq j_y \geq 1$ satisfy the following column constraints

$$\{qj_\beta + p\beta\} \leq r < \{q(j_\beta + 1) + p\beta\} \quad \forall \beta = 1, 2, \ldots, y. \tag{3.11}$$

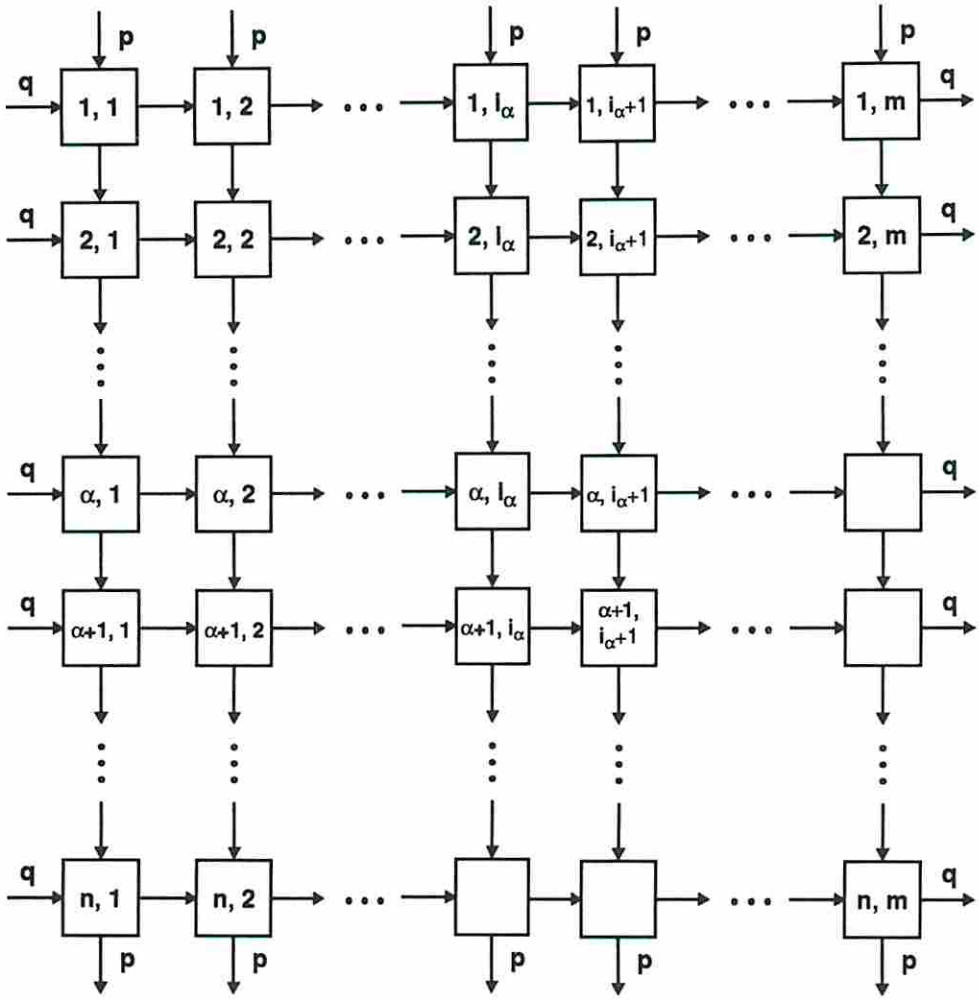


Figure 3.12: Two-dimensional ILA

Let s satisfy the following relation

$$s = min \left\{ \begin{array}{ll} \{nq(\lceil m/i_\alpha \rceil - 1) + mp(\lceil n/\alpha \rceil - 1)\} & \forall \alpha = 1, 2, \ldots, x \\ \{mp(\lceil n/j_\beta \rceil - 1) + nq(\lceil m/\beta \rceil - 1)\} & \forall \beta = 1, 2, \ldots, y \end{array} \right\} \quad (3.12)$$

Then s scells are sufficient for partitioning the ILA so that each output is driven by at most r inputs.

Proof : Consider the unpartitioned ILA shown in Figure 3.12. The acells (α, i_α) and $(\alpha, i_\alpha+1)$ are dependent on $\{pi_\alpha + q\alpha\}$ and $\{p(i_\alpha+1) + q\alpha\}$ inputs, respectively. As per the row constraint given by Equation 3.10, the dependency of the acell $(\alpha, i_\alpha+1)$ exceeds the cone size limit r. It is sufficient to place $q\alpha$ scells in the first α rows between the i_αth and $(i_\alpha + 1)$st columns as shown in Figure 3.13 in order to restrict the dependency for acell $(\alpha, i_\alpha + 1)$. The dependency for the acell $(\alpha + 1, i_\alpha)$ may also exceed the cone size limit. To restrict its dependency, it is sufficient to place pi_α scells in the first i_α columns between the αth and $(\alpha + 1)$st rows as shown in Figure 3.13.

The partitioning process is repeated by placing nq scells after every set of i_α columns and mp scells after every set of α rows as shown in the figure. Thus the total number of scells placed in the ILA is given by the expression $\{nq(\lceil m/i_\alpha \rceil - 1) + mp(\lceil n/\alpha \rceil - 1)\}$. These scells are sufficient since each output is driven by at most $\{pi_\alpha + q\alpha\} \leq r$ inputs in the partitioned ILA.

Each of the x row constraints and y column constraints gives rise to an unique expression for the total number of scells placed in the partitioned ILA. The minimum value among these $(x + y)$ expressions is given by Equation 3.12. It is sufficient to place s scells in the ILA in order to restrict the dependencies of all outputs below the cone size limit r. □

Procedure *ILA* provides a method for placing a minimal number of scells determined by Equation 3.12 on a two-dimensional ILA structure. The two-dimensional ILA has reconvergent fanouts and hence to optimally partition such an array requires a procedure of exponential complexity. Theorem 4 provides only an upper bound on the number of scells required for the two-dimensional ILA structure for a

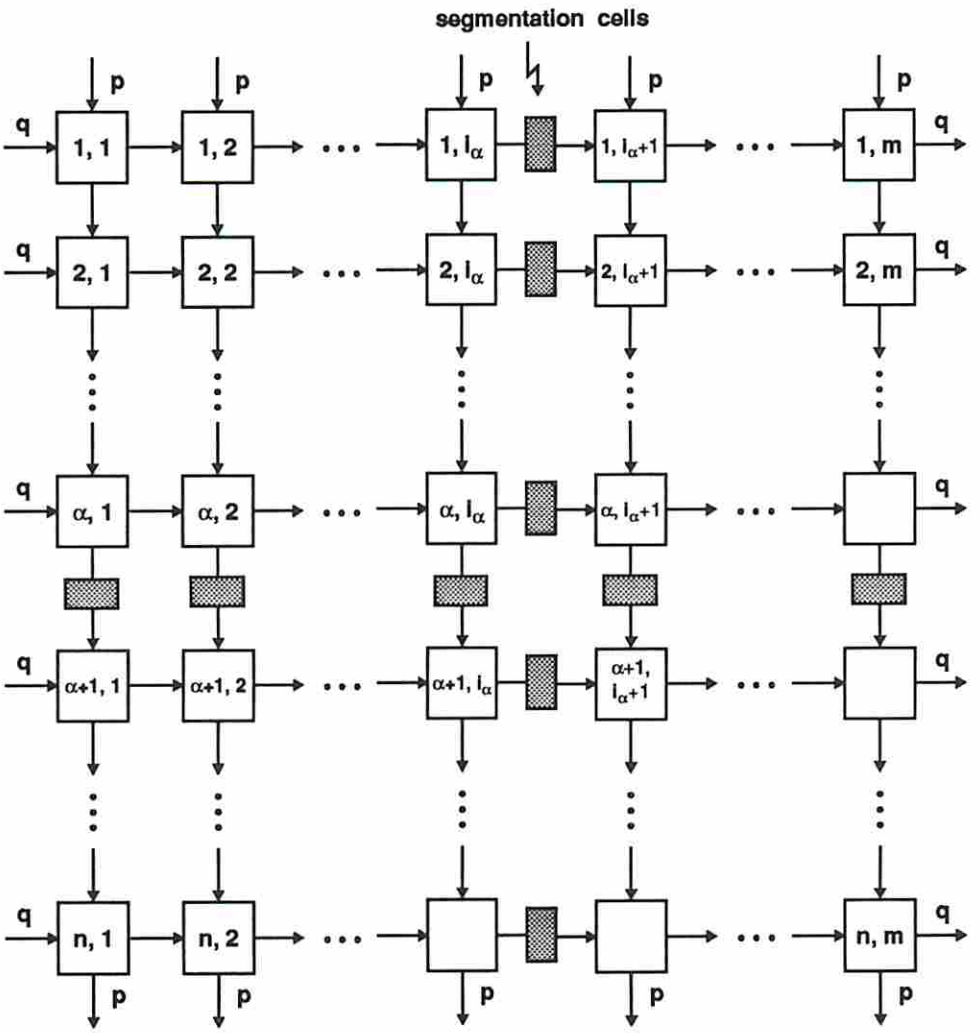


Figure 3.13: Partitioned two-dimensional ILA

specified cone size limit. The applicability of this theorem is illustrated by Example 3. For multi-dimensional ILAs, upper bounds on the number of scells required for partitioning can be derived in a similar fashion.

Procedure ILA

Input: Two dimensional ILA and desired cone size limit r.
Output: Partitioned ILA satisfying the cone size limit.

1. Determine the parameters m, n, p and q for the ILA.
2. Determine α and i_α satisfying the minimum value in Equation 3.12 of Theorem 4.
3. $c \leftarrow 1$.
4. While $(c\alpha < n)$ do
   (a) Place mp segmentation cells after $c\alpha$th row as shown in Figure 3.13.
   (b) $c \leftarrow c+1$.
5. $c \leftarrow 1$.
6. While $(ci_\alpha < m)$ do
   (a) Place nq segmentation cells after ci_αth column as shown in Figure 3.13.
   (b) $c \leftarrow c+1$.
7. Check for the necessity of all segmentation cells placed in the ILA and remove those that are not necessary.

Example 3 Consider the two-dimensional ILA shown in Figure 3.14. The indices of the acells are shown in the figure. The ILA parameters (with respect to Figure 3.12) are $p = q = 2$ and $n = m = 4$. Let us partition the ILA for $r = 8$.

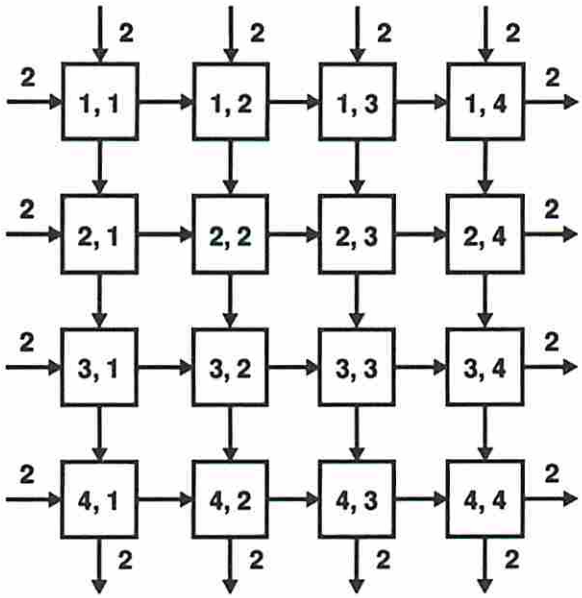


Figure 3.14: A (16, 16, 16) two-dimensional ILA circuit

The ILA has identical row and column constraints due to the symmetrical structure. The row and column constraints (refer to Equations 3.10 and 3.11) are given by

$$\begin{aligned} 2\times 3 + 2\times 1 \leq 8 < 2\times 4 + 2\times 1 \\ 2\times 2 + 2\times 2 \leq 8 < 2\times 3 + 2\times 2 \\ 2\times 1 + 2\times 3 \leq 8 < 2\times 2 + 2\times 3 \end{aligned}$$

The row and column constraints parameters (refer again to Equations 3.10 and 3.11) are $x = y = 3$, $i_1 = j_1 = 3$, $i_2 = j_2 = 2$ and $i_3 = j_3 = 1$. Substituting all the parameters into Equation 3.12 yields $s = 16$. Theorem 4 states that 16 scells are sufficient for partitioning the ILA so that each output is driven by at most eight inputs. The 16 scells are placed in the ILA by the procedure ILA as shown in Figure 3.15(a). The input dependencies of the acells are shown in the figure. Alternatively, consider a greedy procedure that places scells only before those acells that have dependencies greater than the cone size limit r. The greedy procedure results in placing 20 scells as shown in Figure 3.15(b). The input dependencies of the

acells are shown in the figure. This case illustrates that the procedure ILA results in a lower number of scells than the greedy procedure.

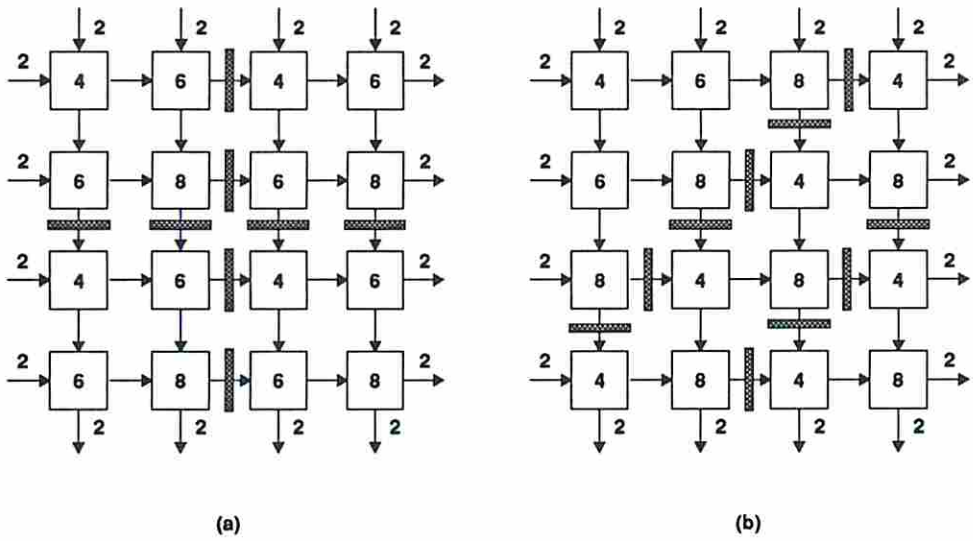


Figure 3.15: Logic array partitioned by (a) procedure ILA (b) greedy procedure for $r = 8$.

Consider again the unpartitioned ILA and let us partition for $r = 10$. The row and column constraints (again refer to Equations 3.10 and 3.11) are given by

$$\begin{aligned}
2 \times 4 + 2 \times 1 &\leq 10 < 2 \times 5 + 2 \times 1 \\
2 \times 3 + 2 \times 2 &\leq 10 < 2 \times 4 + 2 \times 2 \\
2 \times 2 + 2 \times 3 &\leq 10 < 2 \times 3 + 2 \times 3 \\
2 \times 1 + 2 \times 4 &\leq 10 < 2 \times 2 + 2 \times 4
\end{aligned}$$

The constraints parameters (refer again to Equations 3.10 and 3.11) are $x = y = 4$, $i_1 = j_1 = 4$, $i_2 = j_2 = 3$, $i_3 = j_3 = 2$ and $i_4 = j_4 = 1$. Substituting all the parameters into Equation 3.12 yields $s = 16$. Two among the 16 scells are found to be unnecessary and hence only 14 scells are placed by the procedure ILA as shown in Figure 3.16(a). Alternatively, the greedy procedure also results in placing 14 scells as shown in Figure 3.16(b). □

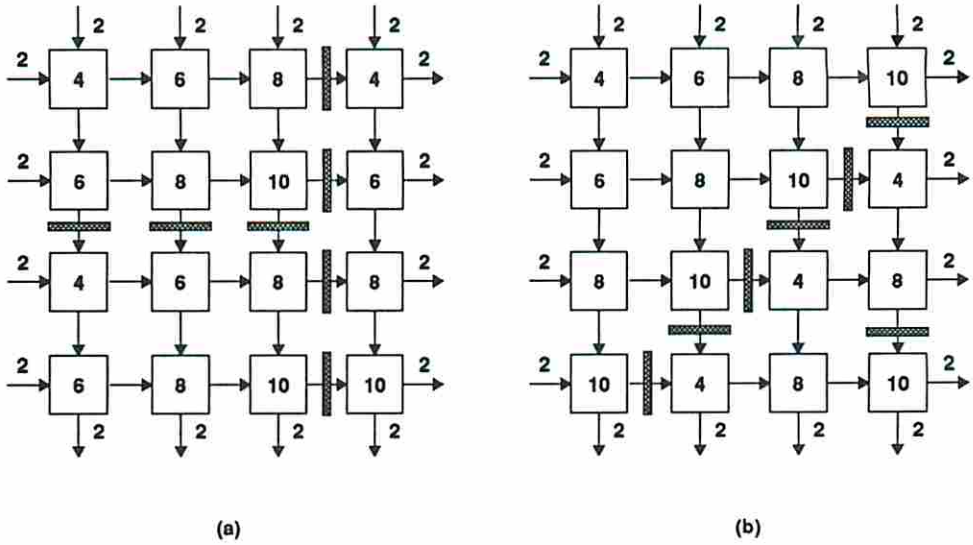


Figure 3.16: Logic array partitioned by (a) procedure ILA (b) greedy procedure for $r = 10$.

3.6 Partitioning Large Circuits

Large circuits invariably have reconvergent fanouts and hence require a procedure of exponential complexity to determine the minimum number of segmentation cells. For partitioning large circuits, we have developed an efficient heuristic procedure of polynomial complexity to obtain good suboptimal solutions. The heuristic is based on the articulation values of the nodes in the circuits. The benefit of placing a segmentation cell on a node is measured in terms of its articulation value with respect to other nodes in its fanout cone. Placing a cell on a bad node n_i requires further placement of cells in the fanin cone of n_i in order to reduce d_i. Due to this nature, the measured benefit for the cell placed on n_i may get drastically reduced. On the contrary, placing a cell on a good node does not require placement of another cell in its fanin cone. Hence we shall consider only the good nodes for placing segmentation cells. Lemma 5 states that it is sufficient to consider only the candidate nodes among the good nodes.

3.6.1 Heuristic Measure

There are three minor variations to the main theme in our heuristic approach. These variations form the alternate heuristics $H1$, $H2$ and $H3$. Let G, B, E and C represent the sets of good nodes, bad nodes, envelope nodes and candidate nodes, respectively. The benefit of placing a segmentation cell on a candidate node n_i is quantified by the heuristic measure h_i. The measure is primarily based on the articulation value of node n_i with respect to the nodes in its fanout cone and varies with the heuristics.

- **Heuristic H1:** The heuristic measure h_i for candidate node n_i is given by $h_i = \sum_{n_j \in B} a_{i,j} + \sum_{n_k \in C} a_{i,k}$. The heuristic gives equal importance to the bad nodes and other candidates. Bad nodes not in the fanout cone of n_i and candidates not fed by node n_i do not contribute to the heuristic measure.

- **Heuristic H2:** The heuristic measure h_i for candidate node n_i is given by $h_i = \sum_{n_j \in B} 2a_{i,j} + \sum_{n_k \in C} a_{i,k}$. The heuristic gives twice the importance to the bad nodes compared to other candidates. Since the candidates already have input dependency values less than r, the benefits for the bad nodes are stressed compared to the benefits for the candidates.

- *Heuristic H3:* The heuristic measure h_i for candidate node n_i is given by $h_i = \sum_{n_j \in E} a_{i,j}$. The heuristic gives importance to only the envelope nodes. Since the envelope nodes feed the rest of the bad nodes, the benefits for all bad nodes are implicitly considered.

3.6.2 Heuristic Procedure

The details of the heuristic procedure are provided below. Among the three heuristics $H1$, $H2$ and $H3$, one of them is selected prior to invoking the procedure.

Procedure CSR

Input: Graph of (n, m, k) circuit and desired cone size limit r.
Output: Partitioned circuit satisfying the cone size limit.

1. $S \leftarrow \emptyset$. /* Initialize the set of segmented nodes */
2. Determine the sets of good nodes G, bad nodes B and the envelope nodes E.
3. **candidate** (C). /* Determine the set of candidate nodes C */
4. For each node $n_i \in C$, determine its heuristic measure h_i.
   - (Heuristic H1:) $h_i = \sum_{n_j \in B} a_{i,j} + \sum_{n_k \in C} a_{i,k}$
   - (Heuristic H2:) $h_i = \sum_{n_j \in B} 2a_{i,j} + \sum_{n_k \in C} a_{i,k}$
   - (Heuristic H3:) $h_i = \sum_{n_j \in E} a_{i,j}$
5. Select the candidate node n_i^* with the highest heuristic measure.
   Place a segmentation cell on n_i^*; $S \leftarrow S \cup n_i^*$.
6. Update the dependencies of all the nodes in the fanout cone of n_i^*.
7. If there exists a bad node, then go to step 2.
8. **check** (S). /* Check the necessity of all segmented nodes */

Procedure candidate (C)

1. $C \leftarrow \emptyset$; $Q \leftarrow \emptyset$.
   /* Initialize the candidate set C ; create a FIFO queue Q. */
2. $l_i \leftarrow 0$ for all $n_i \in G$.
   /* Initialize the labels for all good nodes G */
3. $count \leftarrow 1$. /* initialize the label counter */
4. Determine the good nodes (say G') that directly feed envelope E.
5. For each node $n_i \in G'$ do
   (a) $l_i \leftarrow count$; $count \leftarrow count + 1$.
   (b) $C \leftarrow C \cup n_i$; Add n_i to Q.
6. While $Q \neq \emptyset$ do

(a) Remove the first node n_i from Q.

(b) For each fanin node n_j of n_i do

i. If $(l_j = 0)$ then $l_j \leftarrow l_i$; /* new label assigned for n_j */

ii. If $(l_j \neq l_i)$ and $(n_j \notin C)$ then /* n_j is a candidate */
$\{l_j \leftarrow count;\ count \leftarrow count + 1;\ C \leftarrow C \cup n_j\}$.

iii. If all fanout nodes of n_j are traversed, then add n_j to Q.

Procedure check (S)

1. For each node $n_i \in S$ do

(a) Remove the segmentation cell placed on n_i.

(b) Determine the maximum cone size k for the circuit.
If $(k \leq r)$ then $S \leftarrow S - \{n_i\}$; else place the cell back on n_i.

2. For each node $n_i \in S$ do

(a) Remove the segmentation cell placed on n_i.

(b) Determine the maximum cone size k for the circuit.
If $(k \leq r)$ then $S \leftarrow S - \{n_i\}$; else do

i. Determine the candidate n_i^* with the highest heuristic measure.

ii. If $(n_i^* = n_i)$, then place the cell back on n_i; else do

A. Place the cell on n_i^*.

B. If $(k > r)$ then remove the cell from n_i^* and place back on n_i.

3.6.2.1 Cone Size Reduction

Procedure CSR progressively modifies bad nodes into good nodes. The procedure terminates when the bad set becomes empty. During each iteration, the candidate (n_i^*) with the highest heuristic measure is selected for placing a segmentation cell. Due to the greedy nature of the procedure, cells placed during earlier iterations may

become unnecessary towards the end of the iteration process. As the final step, the cells are examined and unnecessary ones are removed from the circuit.

The main step is determining the heuristic measure for the candidates. The time complexity for determining the articulation values of a candidate with respect to the nodes in its fanout cone is $O(N)$, where N is the number of nodes in the circuit. Repeating for all the candidates leads to a complexity of $O(N^2)$. The steps are repeated until the maximum cone size of the circuit gets reduced to a value less than or equal to the cone size limit. Thus the complexity of the procedure is given by $O(N^3)$, where N is the number of nodes in the circuit.

3.6.2.2 Candidates

Procedure *candidate(C)* determines the candidates among the good nodes G. The good nodes G' that directly feed the nodes in the envelope form the initial set of candidate nodes. The labeling scheme described in the procedure is based on the following principle. Node n_j in the fanin cone of a candidate node n_i becomes a candidate if n_j (but not n_i) lies in the fanin cone of another candidate node n_k. The nodes in G' are assigned unique labels. Nodes are traversed in breadth first manner starting from G' towards the input nodes. Node n_j that directly feeds node n_i and yet to be assigned a label is assigned the same label as that of n_i. If node n_j is revisited from another node with a different label, then n_j forms a candidate node. The complexity of the procedure is linear in terms of the number of signals in the circuit.

3.6.2.3 Segmentation Cells

Procedure *check(S)* examines the necessity of segmentation cells placed in the circuit. Every cell placed in the circuit is checked for its necessity by temporarily removing it from the circuit. Sometimes removal of a cell from n_i may result with better placement of the cell on another node n_i^*. The complexity of the procedure is $O(N^2)$ where N is the number of nodes in the circuit.

3.6.3 Experimental Results

The procedures described above have been implemented and experiments were conducted on several circuits including the ISCAS combinational benckmark circuits [7]. The results on the benchmark circuits are compared with the results reported in [16].

Ckt	(n,m,k)	HW1	HW2	H1	H2	H3
c432	(36,7,36)	20*	21	20*	20*	20*
c499	(41,32,41)	9	9	8*	8*	8*
c880	(60,26,45)	14	14	10*	10*	11
c1355	(41,32,41)	9	9	8*	8*	8*
c1908	(33,25,33)	18	17	14*	14*	15
c2670	(233,140,122)	37	29*	29*	29*	29*
c3540	(50,22,50)	63	68	58*	66	61
c5315	(178,123,67)	42	46	37*	37*	38
c6288	(32,31,32)	65*	70	67	79	70
c7552	(207,108,194)	79	85	79	76*	83

Table 3.1: Results on Benchmark Circuits (r = 20)

Table 3.1 presents the experimental results on the benchmark circuits for the desired cone size limit (r) of 20 inputs. Column two describes the characteristics of the benchmark circuits. For example, circuit $c432$ has 36 inputs and seven outputs with a maximum dependency of 36 inputs. The subsequent columns denote the number of segmentation cells required for partitioning the circuits to satisfy the cone size limit. HW1 and HW2 are the heuristics proposed in [16]. H1, H2 and H3 are the heuristics described earlier. For example, circuit $c432$ requires 20, 21, 20, 20 and 20 segmentation cells using the heuristics HW1, HW2, H1, H2 and H3, respectively. Entries with asterisks indicate the best results obtained by the five procedures.

Heuristics HW1 and HW2 are basically hill-climbing procedures with backtracking involved in the search process. Heuristic HW1 contains an exponential complexity procedure to determine the optimal number of cells for restricting the dependencies of envelope nodes. The procedure is repeated till all the output cones have acceptable dependencies. The technique tends to neglect the global effects on

placing the individual cells. Heuristic HW2 considers the global effects but cells are allowed to be placed only on certain nodes in the fanin cones of envelope nodes.

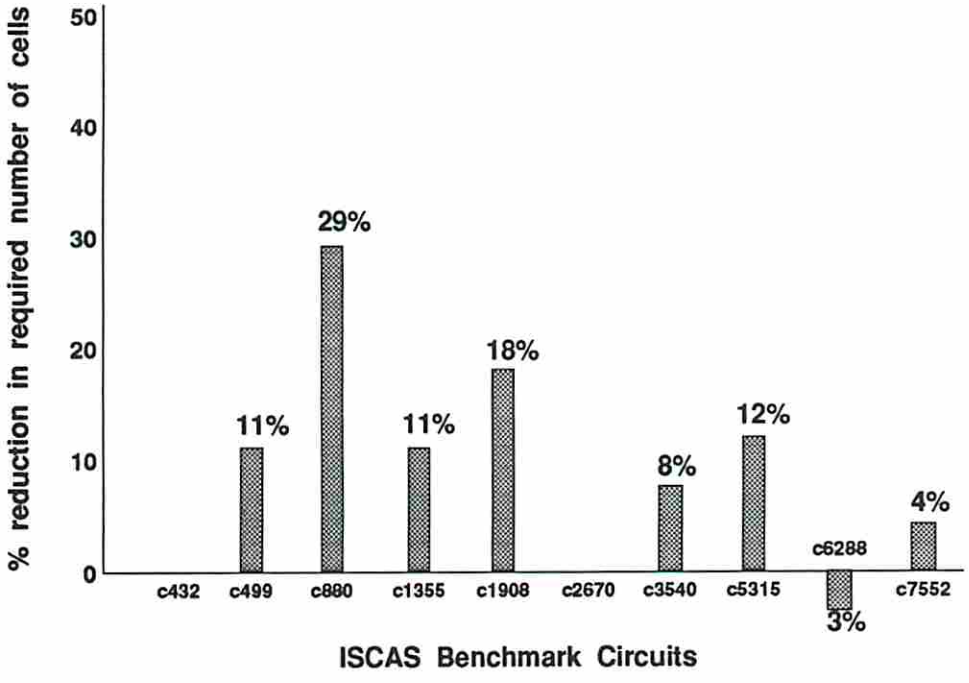


Figure 3.17: Comparision of best results (r = 20)

Our heuristics consider all the bad nodes simultaneously. We consider all the candidate nodes as opposed to the subset of candidate nodes considered in HW1 and HW2. Our heuristic measures evaluate the placement of cells on candidate nodes with respect to all the bad nodes. This evaluation provides a global approach to the problem and results in good suboptimal solutions. Figure 3.17 shows a barchart comparing the best results produced by our heuristics with the best results produced by heuristics HW1 and HW2. Table 3.2 presents the experimental results on the benchmark circuits for the desired cone size limit (r) of 16 inputs.

Table 3.3 presents the required number of segmentation cells for various circuits for different values of cone size limit. The experiments were performed on six circuits, viz. benchmark circuits $c432$, $c499$ and $c880$, a bit-sliced 16-bit adder, a 8-bit multiplier and an ALU. The value for the cone size limit (r) is varied between 40 and 12. The required number of segmentation cells (s) to achieve the desired cone sizes

Ckt	(n,m,k)	HW1	HW2	H1	H2	H3
c432	(36,7,36)	27*	27*	27*	27*	27*
c499	(41,32,41)	8*	8*	8*	8*	8*
c880	(60,26,45)	19	16	11*	11*	11*
c1355	(41,32,41)	8*	8*	8*	8*	8*
c1908	(33,25,33)	21	22	18*	18*	18*
c2670	(233,140,122)	45	33*	33*	33*	33*
c3540	(50,22,50)	87*	90	91	93	87*
c5315	(178,123,67)	52*	62	54	52*	54
c6288	(32,31,32)	93*	98	126	126	102
c7552	(207,108,194)	100*	117	108	104	102

Table 3.2: Results on Benchmark Circuits (r = 16)

Ckt	(n,m,k)	r	40	36	32	28	24	20	16	12
c432	(36,7,36)	s	-	-	9	13	16	20	27	35
		k'	-	-	32	28	24	20	16	12
c499	(41,32,41)	s	2	6	6	7	7	8	8	16
		k'	39	32	32	22	22	14	14	10
c880	(60,26,45)	s	1	2	4	9	9	11	11	19
		k'	37	36	32	24	24	20	16	12
add16	(33,17,33)	s	-	-	1	1	1	1	2	3
		k'	-	-	31	27	23	19	15	11
mult8	(18,16,18)	s	-	-	-	-	-	-	6	22
		k'	-	-	-	-	-	-	16	12
ALU	(22,16,22)	s	-	-	-	-	-	5	17	34
		k'	-	-	-	-	-	20	16	12

Table 3.3: Segmentation Cells Required for Various Cone Sizes

are listed in the table. After partitioning with the segmentation cells, the resulting maximum dependency (k') for the partitioned circuits are also listed. The difference between the desired cone size (r) and the maximum dependency (k') obtained after partitioning with (s) cells is due to the nature of the circuit. For example, circuit $c499$ requires six segmentation cells for reducing the maximum dependency to 32. However, after placing another cell, the maximum dependency of the circuit reduces to 22. This drastic reduction is due to the presence of an appropriate articulation node in the circuit.

The data in Table 3.3 is graphically depicted in Figure 3.18. The graph illustrates the variations in the required number of segmentation cells as per the variations in the desired cone size. Circuits $c432$, $mult8$ and ALU have steep curves while the circuits $c499$, $c880$ and $add16$ have curves with both steep and flat portions.

The steep behavior is due to the drastic increase in the required number of segmentation cells as the desired cone size is reduced. We believe this behavior can be attributed to the following reasons. First, the scarcity of articulation nodes and/or poor articulation values of the nodes in a circuit. For this case, segmentation cells are placed on many reconvergent fanout paths in the circuit in order to reduce the dependencies on output cones having more than r inputs. Secondly, output cones having more than r inputs may be disjoint and hence may require separate cells to reduce their dependencies.

The flat behavior illustrates that no additional segmentation cells are required for the reduction in the maximum cone size. Circuits with flat behavior have well placed articulation nodes with good articulation values. For example, $add16$ is a bit-sliced circuit, and forms a one-dimensional ILA structure. The circuit has no fanouts and contains only articulation nodes connecting the adjacent bit-slices that naturally form candidates. Hence the circuit exhibits an almost flat curve.

Note that the curves are extrapolated as straight lines between the actual data points. This extrapolation need not necessarily reflect the true values between the data points. The circuits should ideally give rise to monotonically decreasing curves (step functions) if experimented with all possible values of r. However, anomalies of the greedy heuristic procedure may result in deviations from the expected behavior.

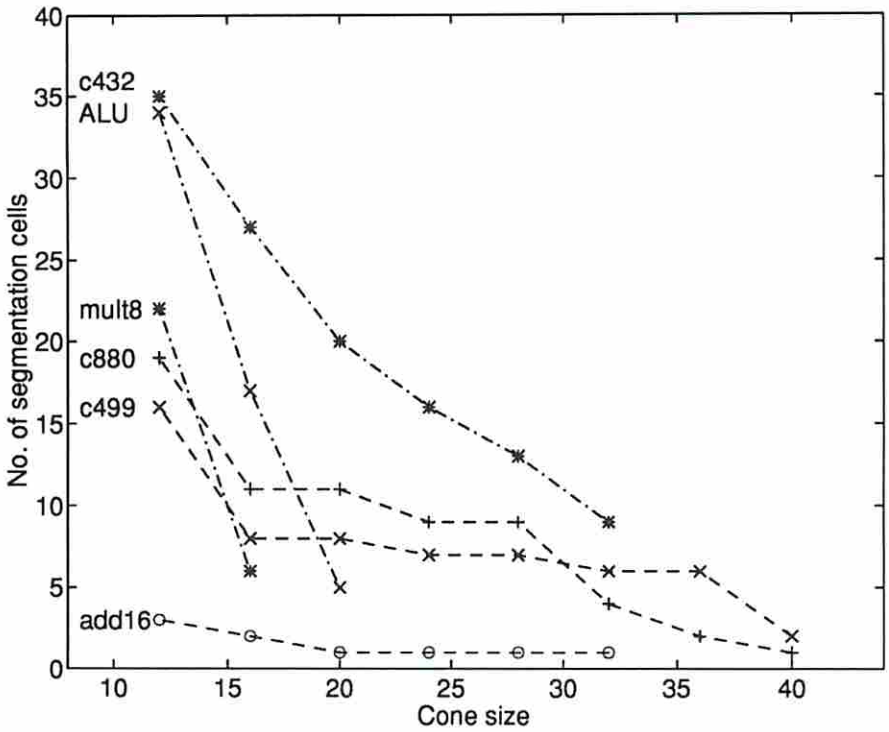


Figure 3.18: Segmentation cells required for various cone sizes

3.7 Partitioning Sequential Circuits

A sequential circuit can be hierarchically reorganized and partitioned into combinational blocks and registers [13]. A sequential circuit is said to be *balanced* [14] if all the paths between any two combinational blocks encounter equal number of register delays. For example, consider a $(24, 8, 24)$ sequential circuit shown in Figure 3.19(a). The combinational blocks and the registers are represented by squares and horizontal lines, respectively. Every pair of combinational blocks has at most one path between them except the pair $C1$ and $C2$ has two reconvergent paths between them. Both the paths encounter two register delays and hence the circuit is a balanced sequential circuit. We have extended our partitioning strategy for balanced sequential circuits.

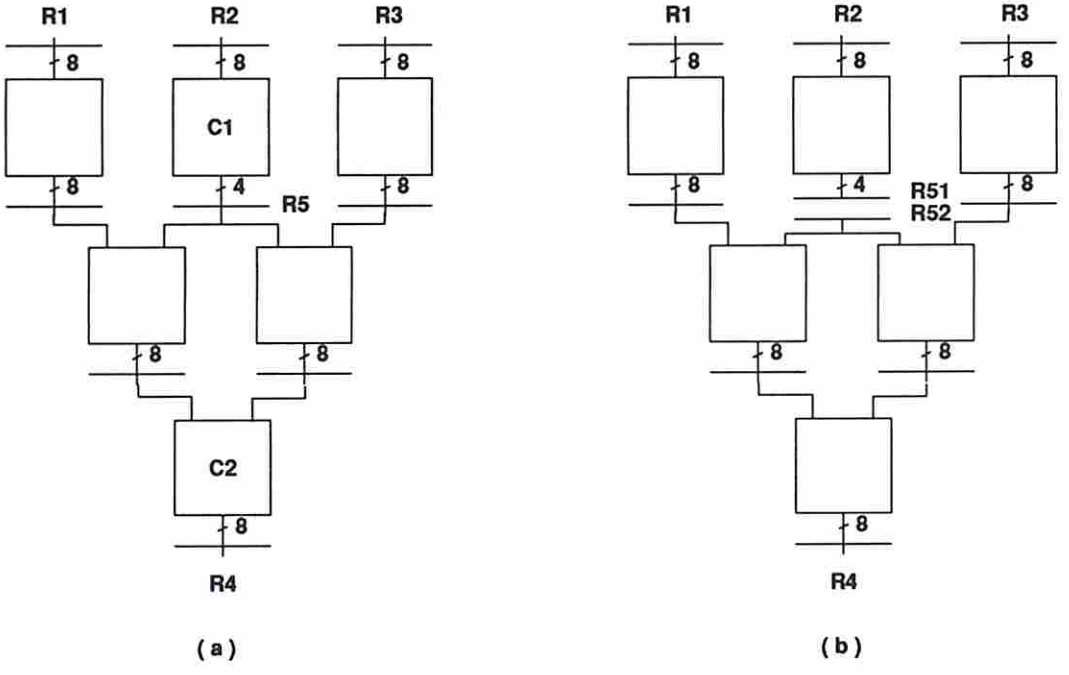


Figure 3.19: A balanced sequential $(24, 8, 24)$ circuit (a) before partitioning and (b) after partitioning

We shall assume that the individual combinational blocks have acceptable number of inputs and the segmentation cells are placed external to the blocks. In other words, only some of the existing registers are modified as segmentation registers to

satisfy the desired cone size limit. Whenever a register is selected for modification, the entire register is modified as a segmentation register.

Let us partition the circuit for a desired cone size limit of 20 inputs. Register $R5$ is selected and modified as a segmentation register as shown in Figure 3.19(b). After partitioning, the original $(24, 8, 24)$ circuit gets modified to a $(28, 12, 20)$ circuit. Registers $R51$ and $R52$ form the pseudo-output and pseudo-input registers, respectively. During the test mode, registers $R1$, $R2$, $R3$ and $R52$ are configured as TPG and registers $R4$ and $R51$ are configured as MISR. The partitioned circuit is pseudo-exhaustively tested in a single test session.

Procedure CSR can be extended for partitioning balanced sequential circuits. A balanced sequential circuit can be modeled as a directed acyclic graph with the combinational blocks and the registers being represented as nodes and edges, respectively. Each node n_i is associated with a weight w_i equal to the width of the output register of the combinational block represented by n_i. With reference to the procedure CSR, the heuristic measure h_i of a candidate node n_i is normalized by dividing w_i. The candidate node with the highest normalized heuristic measure is selected and the corresponding output register is modified as a segmentation register.

3.8 Summary

We have presented several strategies to partition logic circuits for pseudo-exhaustive testing. Various classes of circuits are considered in detail and efficient partitioning procedures are developed for these circuits. Two-level and multi-level fanout-free circuits can be partitioned with an optimal number of segmentation cells using our procedures of polynomial complexity. The characteristics of circuits with fanouts justify the need for a partitioning procedure of exponential complexity to obtain optimal solutions. For partitioning ILA structures, the necessary and sufficient number of cells are determined based on the regularity of the structures. Optimal solutions to small circuits with fanouts can be obtained by solving the ILP formulation. Good suboptimal solutions can be achieved for large circuits using our heuristic approach. The efficiency and quality of our heuristic approach are demonstrated on the ISCAS combinational benchmark circuits. The partitioning strategy is also extended for balanced sequential circuits.

Chapter 4

Partitioning for Maximal Test Concurrency

4.1 Introduction

Pseudo-exhaustive testing of an (n, m, k) circuit considers the circuit as a collection of m output cones and exhaustively tests each output cone of the circuit. The output cones can be exhaustively tested in two ways. One way is to apply k independent test signals (2^k patterns) per test session and have multiple sessions to test all the output cones. The output cones of the circuit are grouped into k-testable groups [17]. All the cones in a *k-testable* group are exhaustively tested simultaneously with k test signals. The upper bound on the number of test sessions is $\lceil m/2 \rceil$, since any two arbitrary cones can be tested simultaneously [27].

Another way is to test all the output cones of the circuit in a single test session. The number of test signals required is bounded below and above by k and n respectively. Two inputs that do not fan out to any common output can share a test signal during the test mode. Circuits requiring only k test signals are referred to as **maximal test concurrency** (MTC) circuits [27]. We shall restrict our attention to pseudo-exhaustive testing of circuits in a single test session.

Generation of an optimal pseudo-exhaustive test set for an (n, m, k) non-MTC circuit is a hard problem. An optimal test set must contain an exhaustive set of patterns for each output cone of the circuit. In order to test all the output cones in a single test session, the circuit requires w test signals, where $k < w \leq n$. However, for exhaustive testing of the output cones, all 2^w patterns are not necessary. TPGs designed for non-MTC circuits usually do not generate optimal test sets [5, 27].

On the contrary, an optimal pseudo-exhaustive test set can be easily generated for an (n, m, k) MTC circuit. A k-bit counter can exhaustively test all output cones. Therefore we attempt to modify non-MTC circuits during test mode to achieve maximal test concurrency. Since the circuit has the maximum cone size of k inputs, it may be possible to test the circuit with k independent test signals by modifying the circuit in the test mode.

Partitioning for cone size reduction described in Chapter 3 partitions an (n, m, k) circuit with s segmentation cells for a desired cone limit r. The partitioning results in an $(n + s, m + s, k')$ circuit (where $k' \leq r < k$). If the partitioning results in a MTC circuit, then the circuit can be tested with k' test signals in a single test session. Test patterns can be generated very easily without any complicated TPGs. However, the partitioning procedure does not guarantee that the modified circuit is an MTC circuit. Our goal is to further partition the circuit to achieve maximal test concurrency. This is referred to as **partitioning for maximal test concurrency** [36]. This demands more modification to the original circuit in terms of additional segmentation cells. In this chapter we investigate the feasibility of transforming non-MTC circuits to be MTC circuits. There exists a trade off between area overhead due to segmentation cells and test time due to multiple test sessions as illustrated by Example 4.

Example 4 Consider the $(3, 3, 2)$ non-MTC circuit shown in Figure 4.1(a). The inputs and outputs are denoted as $\{\theta_1, \theta_2, \theta_3\}$ and $\{O_1, O_2, O_3\}$ respectively. Exhaustive testing of all three output cones in a single test session requires three independent test signals. The three test signals consists of all 2^3 patterns as shown in Figure 4.1(a). For the circuit, an optimal test set consists of just 2^2 patterns and is given by {000, 011, 101, 110}.

Allowing multiple test sessions with 2^2 patterns per session, the circuit requires two test sessions as shown in Figure 4.1(b). In the first session, O_1 and O_3 are exhaustively tested and θ_2 and θ_3 share a test signal. In the next session, θ_1 and θ_2 share a test signal and O_2 and O_3 are tested exhaustively.

The circuit can be modified to achieve maximal test concurrency by placing a segmentation cell. The modified $(4, 4, 2)$ MTC circuit in the test mode is shown in Figure 4.1(c). Input θ_4 and output O_4 are the newly formed pseudo-input and

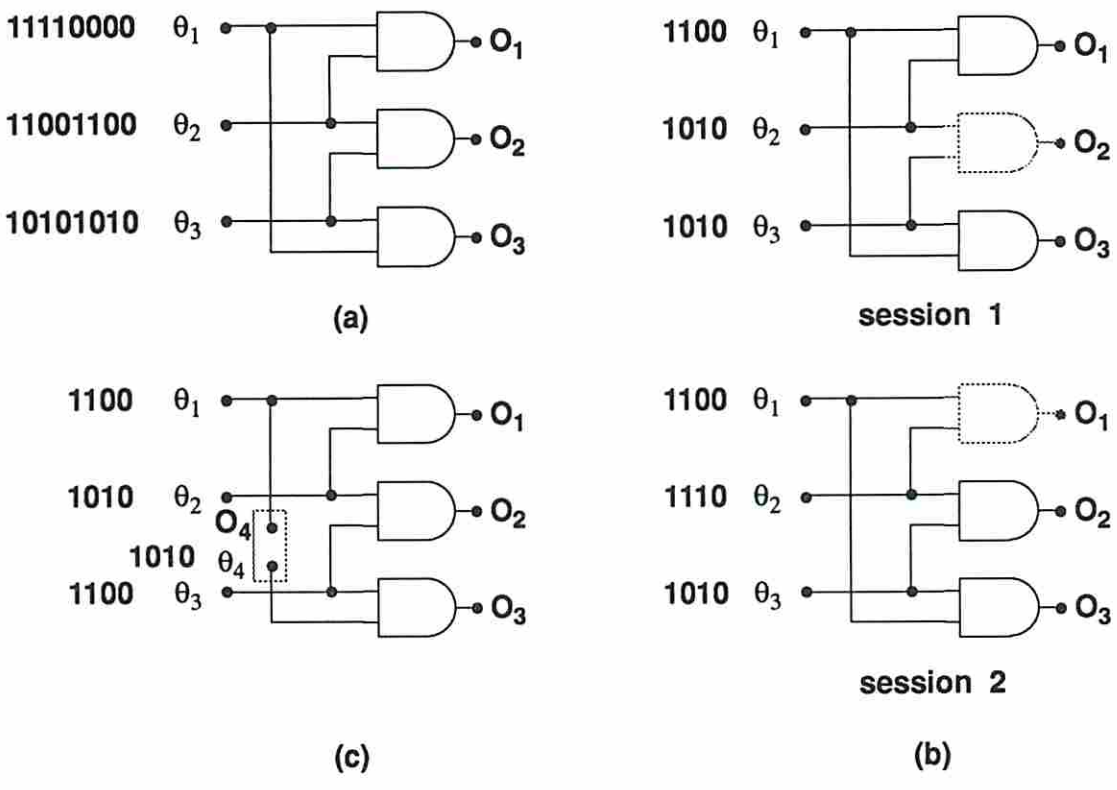


Figure 4.1: Pseudo-exhaustive testing of a non-MTC circuit (a) in a single test session (b) in multiple test sessions (c) modified to an MTC circuit.

pseudo-output respectively. Inputs θ_1 and θ_3, and θ_2 and θ_4 share test signals. The modified circuit can be tested with an optimal test set as shown in the figure. □

4.2 Definitions and Notation

We shall present a few definitions and our notation prior to describing our partitioning procedure for achieving maximal test concurrency. The circuit is modeled as a directed acyclic graph as described in Chapter 3. Each output cone of the circuit forms a subgraph and overlapping cones make the corresponding subgraphs share nodes and edges.

Let us consider an (n, m, k) circuit along with the following notation. The n inputs are denoted as θ_i, $i = 1, 2, \ldots, n$, and the m outputs are denoted as O_j, $j = 1, 2, \ldots, m$, respectively. The input dependencies of the outputs are given by a **dependency matrix (D-matrix)** denoted as $D_{m \times n}$. The matrix element $d_{i,j} = 1$ if O_i depends on θ_j; $d_{i,j} = 0$ otherwise. The **weight of input** θ_j, denoted as w_j, is given by $\sum_{i=1}^{m} d_{i,j}$. The weight w_j denotes the number of outputs dependent on θ_j. The input dependency of O_i is given by $\sum_{j=1}^{n} d_{i,j}$. Since any output depends on at most k inputs, $\sum_{j=1}^{n} d_{i,j} \leq k, \quad \forall i = 1, 2, \ldots, m$.

The relationship among the inputs are given by a **relational matrix (R-matrix)** denoted as $R_{n \times n}$. The R-matrix is a symmetric matrix consisting of binary vector elements $r_{i,j} = b_1 b_2 \ldots b_m$ where $b_x = 1$ if O_x depends on both inputs θ_i and θ_j; $b_x = 0$ otherwise $(\forall x = 1, 2, \ldots, m)$. The **weight of relational vector** $r_{i,j}$ is given by $\sum_{x=1}^{m} b_x$ and denotes the number of outputs that depend on both θ_i and θ_j. Inputs θ_i and θ_j are said to be **related** if the weight of the relational vector $r_{i,j}$ is greater than zero.

We shall present a few definitions similar to those defined in Chapter 3. The set of all edges that feed an edge is called its **fanin cone** and the set of all edges that are fed by the edge is called its **fanout cone**. The **articulation value** of an edge e_i with respect to an output O_j, denoted as $a_{i,j}$, is the number of inputs that feed O_j only via e_i. A **bridge** [10] is an edge that is contained in all paths originating from its fanin cone and ending in its fanout cone. An edge is a **bridge for an I/O pair** (θ_i, O_j) if the edge is contained in all paths originating from θ_i and ending in O_j. The definitions are illustrated by Example 5.

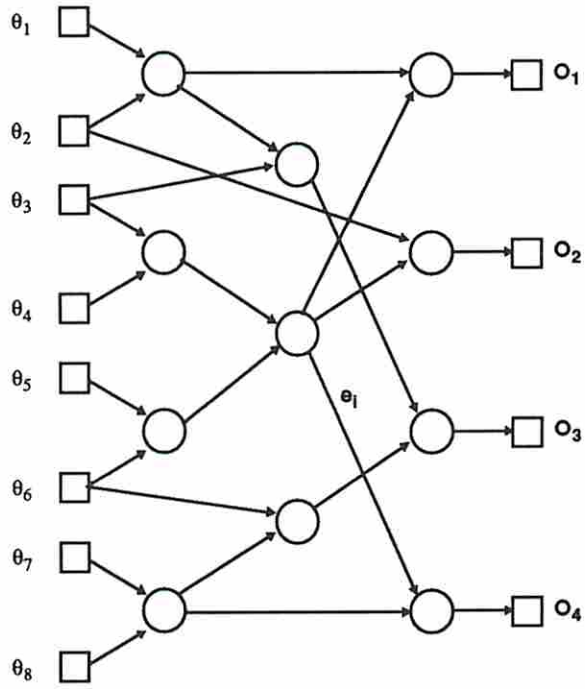


Figure 4.2: Graph model of an $(8, 4, 6)$ circuit.

	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	θ_7	θ_8
O_1	1	1	1	1	1	1	0	0
O_2	0	1	1	1	1	1	0	0
O_3	1	1	1	0	0	1	1	1
O_4	0	0	1	1	1	1	1	1

Table 4.1: Dependency matrix

	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	θ_7	θ_8
θ_1	–	1010	1010	1000	1000	1010	0010	0010
θ_2		–	1110	1100	1100	1110	0010	0010
θ_3			–	1101	1101	1111	0011	0011
θ_4				–	1101	1101	0001	0001
θ_5					–	1101	0001	0001
θ_6						–	0011	0011
θ_7							–	0011
θ_8								–

Table 4.2: Relational matrix

Example 5 Let us consider the graph model of an $(8,4,6)$ circuit shown in Fig 4.2. The D-matrix is given by Table 4.1. The weights of the inputs θ_1 through θ_8 are $2,3,4,3,3,4,2$ and 2 respectively. The R-matrix is given by Table 4.2. Since it is a symmetric matrix only the upper diagonal elements are shown in the table. The edge e_i shown in the figure forms a bridge for the subgraph formed by the output cone O_4. The articulation value $a_{i,4} = 4$ since the four inputs $\{\theta_3, \theta_4, \theta_5, \theta_6\}$ feed O_4 only via e_i. □

Consider an output cone O_j driven by the inputs $\{\theta_{i_1}, \theta_{i_2}, \ldots, \theta_{i_k}\}$. The **weight of** O_j is given by $\sum_{x=1}^{k} w_{i_x}$, where w_{i_x} is the weight of input θ_{i_x}. Among the output cones of maximum cone size k, a cone with the maximum weight is selected as the **reference cone**. The edges that *do not* belong to the reference cone are **candidates** for placing segmentation cells.

4.3 Merging

Two inputs are said to be **merged** if they share a test signal during the test mode. Unrelated inputs can be merged without affecting the exhaustive tests of any output cone in the circuit. On the contrary, merging of related inputs does not ensure exhaustive testing of the outputs fed by the related inputs.

Merging of unrelated input pairs reduces the required number of independent test signals for the circuit. The number of test signals for an (n,m,k) circuit must be reduced from n to k to achieve maximal test concurrency. Hence $(n-k)$ input pairs have to be merged during the test mode. The results on the merging concept are summarized below.

Lemma 9 *Inputs θ_i and θ_j related only by output O_x can be merged if and only if there exists a bridge for the I/O pair $(\theta_i\ (\theta_j), O_x)$ that is not being driven by $\theta_j\ (\theta_i)$.*

Proof : *(If)* Assume there exists a bridge for the I/O pair $(\theta_i\ (\theta_j), O_x)$ that is not being driven by $\theta_j\ (\theta_i)$. Placing a segmentation cell on this bridge makes O_x depend on $\theta_j\ (\theta_i)$ but not on $\theta_i\ (\theta_j)$. The new pseudo-output depend on $\theta_i\ (\theta_j)$ and not on $\theta_j\ (\theta_i)$. Thus θ_i and θ_j become unrelated and hence can be merged.

(Only if) Assume there exists no bridge for the I/O pair $(\theta_i\ (\theta_j), O_x)$ that is not being driven by $\theta_j\ (\theta_i)$. Inputs θ_i and θ_j cannot be made unrelated and hence cannot be merged. □

Theorem 5 *Inputs θ_i and θ_j related by $r_{i,j} = b_1 b_2 \ldots b_m$ can be merged if and only if for every $b_x \neq 0$ there exists a bridge for the I/O pair $(\theta_i\ (\theta_j), O_x)$ that is not being driven by $\theta_j\ (\theta_i)$.*

Proof : Repeatedly apply Lemma 9 for every output O_x dependent on both θ_i and θ_j (i.e. for every $b_x \neq 0$). □

Theorem 6 (Barzilai81) *The problem of merging the optimal number of unrelated input pairs is NP-complete.*

The following observations can be made with respect to special circuits and help prune the solution space.

Observation 4 *A fanout-free circuit is a MTC circuit.*

A circuit without any fanouts can be trivially partitioned into disjoint subcircuits, where each subcircuit has only one output. Inputs that feed different outputs are unrelated and hence can be merged. Thus a fanout-free circuit is always a MTC circuit.

Observation 5 *A non-reconvergent fanout circuit contains only bridges.*

A non-reconvergent fanout circuit has at most only one path from any input to any output and thus all signals form bridges.

Observation 6 *The fanout-free signal of any input node and the fanin signal of any output node need not be considered for placing segmentation cells.*

Placing a segmentation cell on the fanout-free signal of an input (say θ_i) makes θ_i unrelated with another input (say θ_j). However, the new pseudo-input becomes related to θ_j. Placing a cell on the fanin of any output node does not make any of the original inputs unrelated. Hence it is not necessary to consider the fanout-free signal of any input node and the fanin signal of any output node for placing segmentation cells.

4.4 Heuristic Procedure

A related input pair can be merged only if they are made unrelated. Hence a related input pair needs at least one segmentation cell to make them unrelated. Placing a segmentation cell on any bridge makes a few inputs unrelated and hence these inputs can be merged. The outline of the partitioning procedure for modifying an (n, m, k) non-MTC circuit to a MTC circuit is given below.

1. Merge all unrelated input pairs.
2. Identify the reference cone and all candidates for placing segmentation cells.
3. Place a minimal number of cells to make required number (maximum of $(n-k)$) of unrelated input pairs.
4. Merge the resulting unrelated input pairs.

The partitioned circuit being an MTC circuit can be exhaustively tested in a single test session with 2^k patterns. Our partitioning method provides a systematic way of modifying a non-MTC circuit to a MTC circuit. We shall now present the detailed procedure for modifying an (n, m, k) circuit to achieve maximal test concurrency.

Procedure MTC

Input: Graph of an (n, m, k) circuit.
Output: Partitioned MTC circuit.

1. Form D-matrix $D_{m \times n} = [d_{i,j}]$ and determine the weights of inputs.
   Form R-matrix $R_{n \times n} = [r_{i,j}]$ and determine the zero relational vectors.
2. While there exists a zero relational vector $r_{i,j}$ do
   (a) Merge inputs θ_i and θ_j and update D-matrix and R-matrix.
3. $q \leftarrow (n - k - p)$.
   (where p is the number of unrelated input pairs merged and q is the number of related inputs pairs yet to be merged).

4. Determine the reference cone and the candidates C for placing segmentation cells.

5. For each candidate $e_i \in C$, determine its heuristic measure $h_i = \sum_{O_j} a_{i,j}$ (where the summation is over all the outputs in the circuit).

6. While $(q > 0)$ do

   (a) If $(C = \emptyset)$, exit with failure.

   (b) Select the candidate e_i^* with the highest heuristic measure and place a segmentation cell on e_i^*. $C \leftarrow C - \{e_i^*\}$.

   (c) Update D-matrix and R-matrix by including the pseudo-input and the pseudo-output.

   (d) Merge all unrelated input pairs and update D-matrix and R-matrix after every merge operation.

   (e) $q \leftarrow (q - p')$.
   (where p' is the number of unrelated original input pairs merged).

   (f) If $(q = 0)$, exit with success.

The goal is to reduce the number of independent test signals from n to k. Since merging an unrelated input pair reduces the number of test signals by one, $(n - k)$ unrelated input pairs have to be merged to achieve maximal test concurrency.

Since merging the optimal number of unrelated input pairs is a hard problem [5], we shall adopt the following heuristic for merging operation. Whenever there are choices of inputs exists for merging with an input, the input with the maximum weight is selected. This selection facilitates in merging a maximal number of unrelated input pairs.

Merging an unrelated input pair θ_i and θ_j collapses their columns into one in the D-matrix. The binary entries in the original columns are *or*-ed to create entries for the collapsed column. The R-matrix is also modified accordingly. Example 6 illustrates the collapsing concept.

Among the output cones with the maximum cone size k, a cone with the maximum weight is selected as the reference cone. This cone is being driven by a set of k

inputs that fan out to most of the outputs in the circuit. Hence it will be relatively easier to make the remaining inputs unrelated compared to this set of k inputs. All the signals in the circuit that are *not* contained in the reference cone are candidates for placing segmentation cells.

The heuristic measure h_i for the candidate e_i is the cumulative sum of the articulation values of e_i with respect to all the outputs in the circuit. The heuristic measure results in high values for the bridges in the circuit. Bridges are preferred candidates because placing cells on the bridges usually result in many unrelated input pairs.

Placing a segmentation cell on a candidate creates a new pseudo-input and pseudo-output. The number of rows and columns in the D-matrix and R-matrix are increased by one. The length of all relational vectors is also incremented by one.

The computational complexity of determining the candidates is linear in terms of the number of signals in the circuit. The heuristic measure computed for all the candidates is of the order of $O(Em)$ where E is the number of signals and m is the number of outputs in the circuit. Every iteration of the *while* loop selects a candidate with the highest heuristic measure and updates the D-matrix and R-matrix. The entire circuit needs to be traversed for the update of the matrices. The complexity of the partitioning procedure is $O(E^2)$ where E is the number of signals in the circuit.

Example 6 We shall illustrate the partitioning procedure for the $(8, 4, 6)$ non-MTC circuit graph shown in Figure 4.2.

1. The D-matrix and R-matrix are given in Tables 4.1 and 4.2 respectively.

2. The R-matrix does not contain any zero relational vector and hence the number of unrelated input pairs merged is zero ($p = 0$).

3. The number of related input pairs yet to be merged is $q = n-k-p = 8-6-0 = 2$.

4. Among the output cones of maximum cone size, O_1 has the maximum weight and hence forms the reference cone. Since the circuit has no reconvergent fanouts, all the signals form bridges.

5. Candidate e_i with the highest heuristic measure is selected for placing segmentation cell. The candidate is shown in Figure 4.2.

6. The new pseudo-input is θ_9 and the pseudo-output is O_5. The modified D-matrix is given in Table 4.3.

7. Inputs θ_7, θ_8 and θ_9 can now be merged with θ_4, θ_5 and θ_6 respectively.

8. The number of original input pairs merged is $p' = 2$.
   $q = q - p' = 2 - 2 = 0$. Hence exit with success.

The test mode version of the circuit graph with the merged inputs is shown in Fig 4.3. □

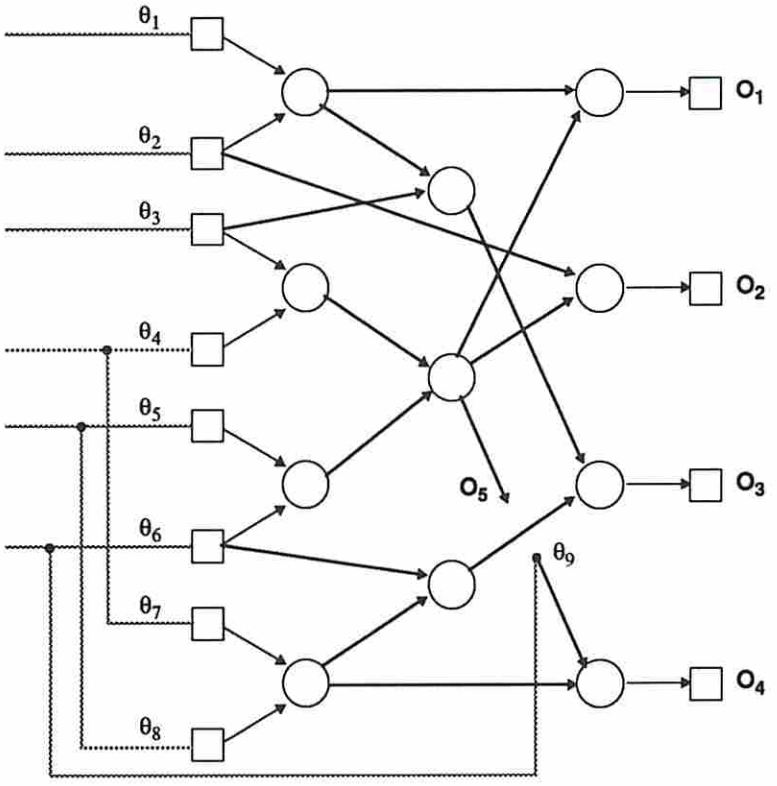


Figure 4.3: Graph model of the modified circuit.

	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	θ_7	θ_8	θ_9
O_1	1	1	1	1	1	1	0	0	0
O_2	0	1	1	1	1	1	0	0	0
O_3	1	1	1	0	0	1	1	1	0
O_4	0	0	0	0	0	0	1	1	1
O_5	0	0	1	1	1	1	0	0	0

Table 4.3: Modified dependency matrix

4.5 Summary

In this chapter we have presented a novel method for partitioning non-MTC circuits to achieve maximal test concurrency during the test mode. The partitioning procedure is based on the graph-theoretical concept of bridges in the circuit. The partitioned circuits can be pseudo-exhaustively tested with the minimum number of independent test signals in a single test session. Pseudo-exhaustive test sets can be easily generated using maximal length LFSRs or counters. Circuits can be first partitioned for cone size reduction and can be further partitioned to achieve maximal test concurrency. A combined strategy can be investigated for partitioning circuits such that the two requirements — (a) the sizes of the output cones are restricted to some user-defined value, and (b) the circuit is maximal test concurrent in test mode — are met simultaneously rather than by a two-step procedure.

Chapter 5

Test Pattern Generation

5.1 Introduction

Our goal is to design hardware efficient TPGs to generate minimal pseudo-exhaustive test sets for a given (n, m, k) circuit. Universal TPGs based on coding theory principles are not tailored for a given circuit as they do not utilize any information about the circuit output cone structures. The test sets generated by universal TPGs are often several orders of magnitudes larger than the minimum length test set for a given circuit. In this chapter we will describe novel circuit-specific TPGs that employ knowledge of the circuit output cone structures for generating minimal test sets.

We will assume that the inputs of the (n, m, k) circuit are ordered and driven by an n-stage input register. During the test mode, the stages of the input register are configured as TPG stages. Since the successive stages of the input register are adjacent to each other, it is desirable to have the successive stages of the TPG also be adjacent to each other. This configuration can significantly reduce the routing overhead due to the interconnections among the TPG stages.

LFSR/SRs [4] have successive stages adjacent to each other forming a shift register and therefore incurs minimal hardware overhead. However LFSR/SRs usually generate non-optimal pseudo-exhaustive test sets. On the contrary, LFSR/XORs [3] usually generate optimal test sets but incur high overhead due to XOR network. We shall describe novel TPGs that incur low hardware overhead like LFSR/SRs and generate small test sets like LFSR/XORs. The theory involved with LFSR/SRs and LFSR/XORs are described below as they form the basis for our TPG designs.

5.2 LFSR/SRs and LFSR/XORs

Consider an (n, m, k) circuit with its inputs being driven by an n-stage input register. Let the inputs be denoted as $\theta_1, \theta_2, \ldots, \theta_n$ and the input register stages be denoted as $s_1, s_2, \ldots, s_n$ respectively. During the test mode, the input register is configured as a circuit-specific TPG.

The (n, m, k) circuit can be exhaustively tested by configuring the n-stage input register as a maximal length LFSR with $P(x)$ as its feedback polynomial. Running the LFSR through its period (and including the all-zero pattern) generates all possible n-bit patterns. The unique sequence of 2^n binary values generated from an individual stage of the LFSR is referred to as a *test signal.* Stage s_i of the input register correspond to the stage s_i $(\forall i = 1, 2, \ldots, n)$ of the TPG. The test signal generated by stage s_i is characterized by the *residue* r_i of that stage. The residue r_i is computed as $(r_p \times x)$ *mod* $P(x)$, where r_p is the residue of the test signal feeding the input of stage s_i. The residue r_1 representing the test signal generated from stage s_1 is considered as the reference with value one (i.e. $r_1 = 1$). For the LFSR, since stage s_{i-1} directly feeds stage s_i, the residue r_i is given by $(r_{i-1} \times x)$ *mod* $P(x)$. The residues $r_1, r_2, \ldots, r_n$ are $1, x, \ldots, x^{n-1}$ represent the n independent test signals generated by the LFSR stages $s_1, s_2, \ldots, s_n$, respectively.

For the (n, m, k) circuit, an LFSR/SR can be constructed by trying all primitive polynomials of degrees $k, k+1, \ldots, w_1$ (where $w_1 \leq n$) until a TPG of degree w_1 is found that generates an exhaustive set of patterns for each of the m output cones of the circuit. The first w_1 stages of the input register are configured as a maximal length LFSR with primitive polynomial $P_1(x)$ of degree w_1. The remaining $(n - w_1)$ consecutive stages are connected as a shift register (SR). The LFSR/SR structure is shown in Figure 5.1(a). The LFSR stages generate w_1 independent test signals represented by the residues $1, x, \ldots, x^{w_1-1}$ respectively. The SR stages generate $(n - w_1)$ unique linear combinations of these w_1 independent test signals. Stage s_i of the input register correspond to stage s_i $(\forall i = 1, 2, \ldots, n)$ of the TPG. For stage s_i, the residue r_i given by x^{i-1} *mod* $P_1(x)$ is a unique linear combination of the residues r_1 through r_{w_1}. For example, if r_i equals $x + 1$, then r_i is a linear combination of the residues r_2 and r_1. Hence the test signal applied to the input θ_i is a linear combination of the test signals applied to the inputs θ_2 and θ_1. Residues r_1 through

r_n are fixed by the polynomial $P_1(x)$. We shall refer to the LFSR/SR structure described above as *single LFSR/SR*. We shall assume that *LFSR/SR* always refers to *single LFSR/SR* structure. The TPG is represented as a (w_1, n) single LFSR/SR composed of n stages and consisting of an LFSR of degree w_1. The following theorem provides the necessary and sufficient condition for exhaustive testing of the output cones of the circuit.

Theorem 7 (Barzilai83) *An output cone that depends on the inputs $\theta_{i_1}, \theta_{i_2}, \ldots, \theta_{i_k}$ will be exhaustively tested if and only if the residues $r_{i_1}, r_{i_2}, \ldots, r_{i_k}$ are linearly independent.*

An output cone dependent on the inputs $\theta_{i_1}, \theta_{i_2}, \ldots, \theta_{i_k}$ will be exhaustively tested provided it is driven by a set of k linearly independent test signals. Since the residues $r_{i_1}, r_{i_2}, \ldots, r_{i_k}$ represent the test signals applied to the inputs of the output cone, these residues must be linearly independent for the corresponding test signals to be linearly independent.

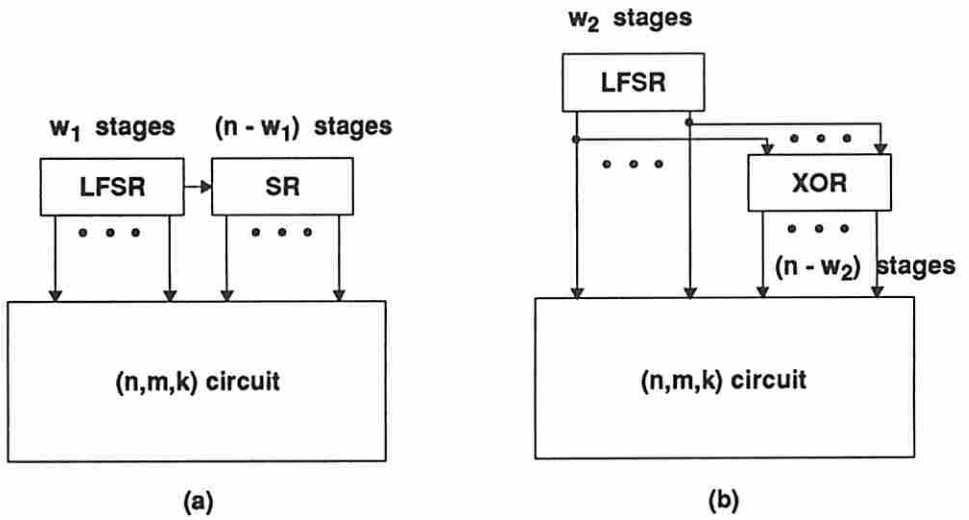


Figure 5.1: TPG Structures (a) LFSR/SR and (b) LFSR/XOR

For the (n, m, k) circuit, an LFSR/XOR [3] can be constructed as follows. Let w_2 (where $k \leq w_2 \leq n$) be the required number of independent test signals for the LFSR/XOR structure as per the design procedure in [3]. A pre-determined set of w_2

stages of the input register (not necessarily the first w_2 stages) is configured as an LFSR with a primitive polynomial $P_2(x)$ of degree w_2. The LFSR stages generate w_2 independent test signals represented by the residues $1, x, \ldots, x^{w_2-1}$ respectively. A pre-determined set of $(n - w_2)$ specific linear combinations of these w_2 independent test signals are generated using an XOR network. These $(n - w_2)$ test signals are applied to the remaining $(n - w_2)$ inputs. The residues for these $(n - w_2)$ test signals are computed as the linear combinations of the residues $1, x, \ldots, x^{w_2-1}$. The LFSR/XOR structure is shown in Figure 5.1(b). The TPG is represented as a (w_2, n) LFSR/XOR consisting of an LFSR of degree w_2 and generating n specific test signals for the circuit inputs. LFSR/XORs incur high area overhead due to the XOR network but generate minimal pseudo-exhaustive test sets by utilizing the information about cone dependencies.

Both LFSR/SR and LFSR/XOR generate independent test signals from their LFSR stages. For the LFSR/SR, the linear combinations for the remaining stages are fixed by the feedback polynomial. For the LFSR/XOR, any desired linear combination of the test signals can be generated using XOR network and assigned for the remaining inputs. The flexibility of generating desired linear combinations in LFSR/XOR does not exist in LFSR/SR. Hence for some circuits, LFSR/SRs generate larger test sets than LFSR/XORs. However, LFSR/SRs incur low area overhead due to the avoidance of XOR network.

5.2.1 Properties of LFSR/SRs

The following lemmas characterize a few important properties of single LFSR/SRs. These results are useful in reducing the complexity of the design procedure for determining single LFSR/SRs.

Lemma 10 (Barzilai83) *The residues of any w consecutive stages of a (w, n) single LFSR/SR are linearly independent.*

Corollary 1 *For a (w, n) single LFSR/SR, the residues $r_{i_1+p}, r_{i_2+p}, \ldots, r_{i_w+p}$ are linearly independent for any value of p if and only if the residues $r_{i_1}, r_{i_2}, \ldots, r_{i_w}$ are linearly independent.*

Proof : The residue r_{i_j+p} equals $x^p \times r_{i_j}$ *mod* $P(x)$, for $1 \leq j \leq w$. The test signal represented by the residue r_{i_j+p} can be considered as the delayed version of the test signal represented by the residue r_{i_j} by p clock cycles. The clock delay is common for all the test signals represented by the residues $r_{i_1}, r_{i_2}, \ldots, r_{i_w}$. Hence the residues $r_{i_1+p}, r_{i_2+p}, \ldots, r_{i_w+p}$ can be considered as the delayed versions of the residues $r_{i_1}, r_{i_2}, \ldots, r_{i_w}$. Therefore one set of residues are linearly independent if and only if the other set of residues are linearly independent. □

Lemma 11 *For a* $(w, 2^w - 1)$ *single LFSR/SR, different primitive polynomials of degree* w *generate different permutations of* $(2^w - 1)$ *distinct residues for the* 2^{w-1} *individual stages.*

Proof : Let the single LFSR/SR be based on a primitive polynomial $P_1(x)$ of degree w and let its $(2^w - 1)$ stages be denoted as $s_1, s_2, \ldots, s_{2^w-1}$ respectively. The residue r_i for stage s_i, given by $r_i = x^{i-1}$ *mod* $P_1(x)$, is a polynomial of degree less than w. Since $P_1(x)$ is primitive, x^{2^w} is the smallest power of x such that x^{2^w} *mod* $P_1(x) = x$. In other words, the residues repeat after $(2^w - 1)$ stages. Hence the stages of the LFSR/SR generate $(2^w - 1)$ distinct residues. The residue of any individual stage represents a unique linear combination of the w independent test signals. LFSR/SR based on another primitive polynomial $P_2(x)$ of degree w also generates $(2^w - 1)$ distinct residues for the stages. For stage s_w, the residue r_w for $P_2(x)$ is different from that for $P_1(x)$ since $P_2(x) \neq P_1(x)$. Hence the residues generated for the stages using $P_2(x)$ is a permutation of the residues generated using $P_1(x)$. Thus different primitive polynomials generate different permutations of the residues. □

A (w, n) single LFSR/SR designed for an (n, m, k) circuit ensures that the residue sets for all the m output cones are linearly independent. The lemmas mentioned earlier help in reducing the number of residue sets that need to be considered for linear independence. Lemma 10 guarantees the linear independence of the residue set for an output cone driven by k or less consecutive inputs. A residue set $\{r_{i_1}, r_{i_2}, \ldots, r_{i_w}\}$ (where $i_1 \leq i_2 \leq \ldots \leq i_w$) can be *normalized* to the residue set $\{r_1, r_{i_2-i_1+1}, \ldots, r_{i_w-i_1+1}\}$. The normalization of the residue sets reduces the number of unique residue sets that need to be considered for linear independence. Ensuring the linear independence of all normalized residue sets ensures the linear independence of all the residue sets for the m output cones of the circuit as per Corollary 1.

For a degree $w \geq k$, each primitive polynomial needs to be considered since each polynomial generates a unique permutation of the residues as given by Lemma 11. The residue assignment for the circuit inputs is fixed by the feedback polynomial $P(x)$. Some of the circuit output cones may not be exhaustively tested due to the *linear dependencies* arising from the fixed residue assignment.

Example 7 illustrates the application of the lemmas characterizing the properties of LFSR/SRs. The example also illustrates the construction of LFSR/SR and LFSR/XOR structures.

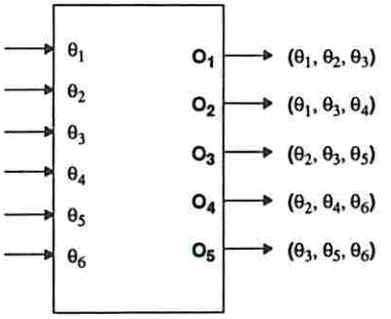


Figure 5.2: An (6,5,3) circuit

Example 7 Consider the $(6, 5, 3)$ circuit shown in Figure 5.2. The six inputs are denoted by θ_1 through θ_6 and the five outputs are denoted by O_1 through O_5 respectively. The input dependencies for the five outputs are given by $\{\theta_1, \theta_2, \theta_3\}$, $\{\theta_1, \theta_3, \theta_4\}$, $\{\theta_2, \theta_3, \theta_5\}$, $\{\theta_2, \theta_4, \theta_6\}$ and $\{\theta_3, \theta_5, \theta_6\}$ respectively. Let the inputs be driven by an input register whose stages are denoted by s_1 through s_6 respectively.

A single LFSR/SR for the circuit can be determined as follows. Output O_1 is driven by three consecutive inputs and hence it is guaranteed to be exhaustively tested (Lemma 10). The residue sets for O_2 and O_5 can be normalized to the set $\{r_1, r_3, r_4\}$. Hence the residue sets for O_2, O_3 and O_4 only need to be considered for linear independence (Corollary 1). It is not necessary to consider the residue sets for O_1 and O_5. The two primitive polynomials $P_1(x)$: $(x^3 + x + 1)$ and $P_2(x)$: $(x^3 + x^2 + 1)$ of degree three are considered for the single LFSR/SR design as they generate different permutation of residues (Lemma 11).

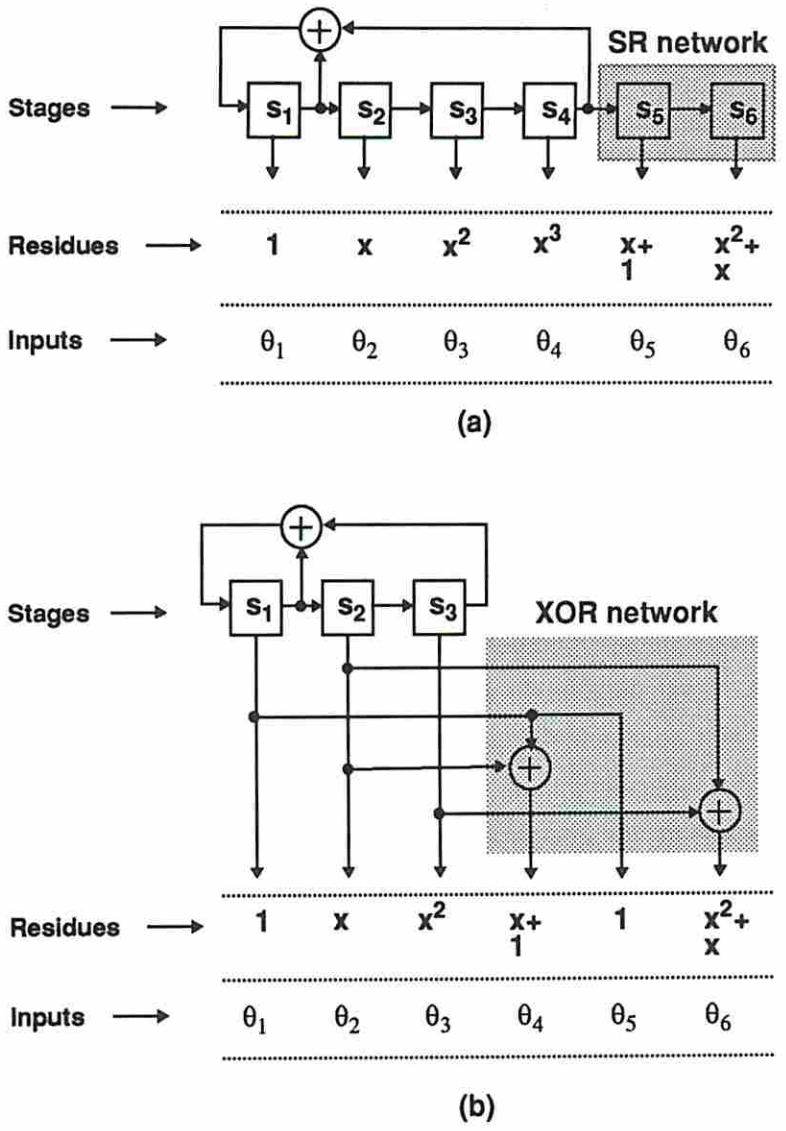


Figure 5.3: TPG Structures (a) LFSR/SR and (b) LFSR/XOR

Consider a $(3,6)$ LFSR/SR based on the primitive polynomial $P_1(x)$: (x^3+x+1). Inputs θ_1 through θ_6 are driven by six unique test signals represented by the residues $1, x, x^2, x+1, x^2+x$ and x^2+x+1 respectively. The residue set for O_3 given by $\{x, x^2, x^2+x\}$ is linearly dependent. Hence the single LFSR/SR based on $P_1(x)$ cannot test O_3 exhaustively.

Consider a $(3,6)$ LFSR/SR based on the primitive polynomial $P_2(x)$: (x^3+x^2+1). Inputs θ_1 through θ_6 are driven by six unique test signals represented by the residues $1, x, x^2, x^2+1, x^2+x+1$ and $x+1$ respectively. The residue set for O_2 given by $\{1, x^2, x^2+1\}$ is linearly dependent. Hence the single LFSR/SR based on $P_2(x)$ cannot test O_2 and O_5 exhaustively. Since $P_1(x)$ and $P_2(x)$ are the only two primitive polynomials of degree three, it is not possible to design a TPG for this circuit based on any $(3,6)$ single LFSR/SR.

Consider a $(4,6)$ LFSR/SR based on the primitive polynomial $P_3(x)$: (x^4+x+1). Inputs θ_1 through θ_6 are driven by six unique test signals represented by the residues $1, x, x^2, x^3, x+1$ and x^2+x respectively. The residue sets for O_2, O_3 and O_4 given by $\{1, x^2, x^3\}$, $\{x, x^2, x+1\}$ and $\{x, x^3, x^2+x\}$ are linearly independent. Hence the single LFSR/SR based on $P_3(x)$ exhaustively tests all the five outputs of the circuit. The single LFSR/SR structure is shown in Figure 5.3(a).

Consider a $(3,6)$ LFSR/XOR based on the primitive polynomial $P_1(x)$: (x^3+x+1). Inputs θ_1, θ_2 and θ_3 are driven by three unique test signals represented by the residues $1, x$ and x^2 respectively. Inputs θ_4, θ_5 and θ_6 are driven by three specific linear combinations given by the residues $x+1, 1$ and x^2+x respectively. The residue sets for the outputs O_1 through O_5 given by $\{1, x, x^2\}$, $\{1, x^2, x+1\}$, $\{x, x^2, 1\}$, $\{x, x+1, x^2+x\}$ and $\{x^2, 1, x^2+x\}$ are linearly independent. Hence the LFSR/XOR based on $P_1(x)$ exhaustively tests all the five outputs of the circuit. The LFSR/XOR structure is shown in Figure 5.3(b).

Note that the LFSR/SR and LFSR/XOR in Figure 5.3 are based on primitive polynomials of degree four and three respectively. Hence the LFSR/SR generates a test set consisting of sixteen patterns while the LFSR/XOR generates an optimal test set consisting of only eight patterns. However, the LFSR/XOR does not utilize all the stages of the input register and incurs hardware overhead due to XOR network. □

5.2.2 Operations on LFSR/SRs

Design operations such as *reconfiguration of feedbacks*, *permutation of stages* and *sharing of test signals* can be applied on single LFSR/SRs to obtain *reconfigurable* LFSR/SRs, *permuted* LFSR/SRs and *sharing* LFSR/SRs respectively. These operations have the potential to avoid linear dependencies in the residue sets of the output cones but result in increased hardware overhead. We shall next describe TPG structures that result from these operations on single LFSR/SRs.

5.2.2.1 Reconfigurable LFSR/SRs

A reconfigurable LFSR/SR [39] has the capability to reconfigure its feedback taps to realize different primitive polynomials for different test sessions. For a single LFSR/SR, a *single* feedback polynomial is selected such that the residue sets for all cones are linearly independent. For a reconfigurable LFSR/SR, a *minimal set* of feedback polynomials is selected such that the residue set for each cone is linearly independent for at least one of the polynomials. The feedback taps are reconfigured using multiplexers to realize a different primitive polynomial during each test session. A subset of output cones is exhaustively tested during each session and each cone will be exhaustively tested in at least one of the sessions. Reconfiguration hardware overhead can be minimized by judiciously selecting polynomials that have common feedback taps.

Example 8 For the $(6, 5, 3)$ circuit shown in Figure 5.2, a single LFSR/SR based on $P_1(x): x^3 + x + 1$ exhaustively tests the outputs O_1, O_2, O_4 and O_5. Similarly, a single LFSR/SR based on $P_2(x): x^3 + x^2 + 1$ exhaustively tests the outputs O_1, O_3 and O_4. A reconfigurable LFSR/SR based on the polynomials $P_1(x)$ and $P_2(x)$ is shown in Figure 5.4. Polynomials $P_1(x)$ and $P_2(x)$ are realized during the first and the second test session respectively. This arrangement ensures that all the outputs are exhaustively tested at least once. The TPG generates a test set with $2 \times 2^3 = 16$ patterns. □

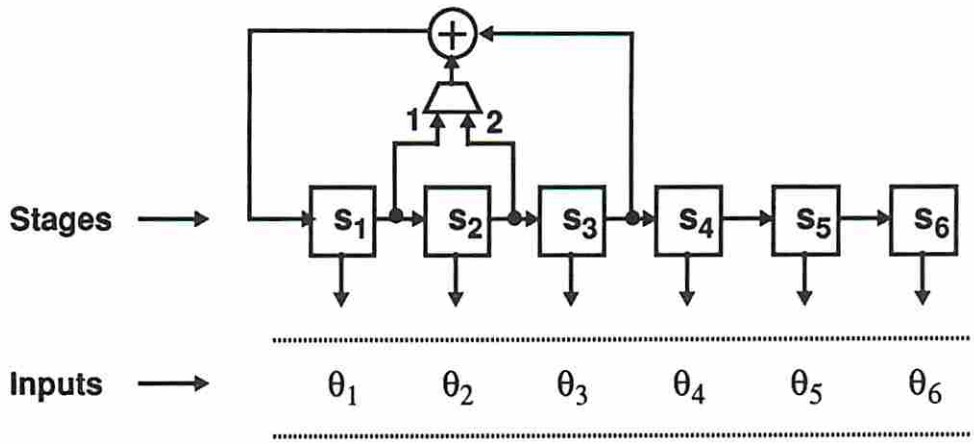


Figure 5.4: Reconfigurable LFSR/SR

5.2.2.2 Permuted LFSR/SRs

A permuted LFSR/SR [22] is essentially a single LFSR/SR whose stages form a permution of stages of the input register. In other words, stage s_i of the TPG need not necessarily drive the input θ_i of the circuit. The inputs are permuted such that the residue sets for all cones are linearly independent. A single LFSR/SR that could not be used because of the fixed assignment of residues may lead to an acceptable solution after reassigning the residues. The permutation of the inputs can result in hardware overhead due to routing among the TPG stages.

Example 9 A permuted LFSR/SR based on the primitive polynomial $P_1(x)$: $(x^3 + x + 1)$ can be determined for the $(6, 5, 3)$ circuit in Figure 5.2 as follows. Residues are assigned to the inputs such that the residue sets for all outputs are linearly independent. The residue assignment for the inputs and the ordering of the stages for the $(3, 6)$ permuted LFSR/SR is shown in Figure 5.5. Stages s_1 through s_4 drive inputs θ_1 through θ_4 respectively. Stages s_5 and s_6 drive inputs θ_6 and θ_5 respectively. The outputs are exhaustively tested with $2^3 = 8$ patterns. □

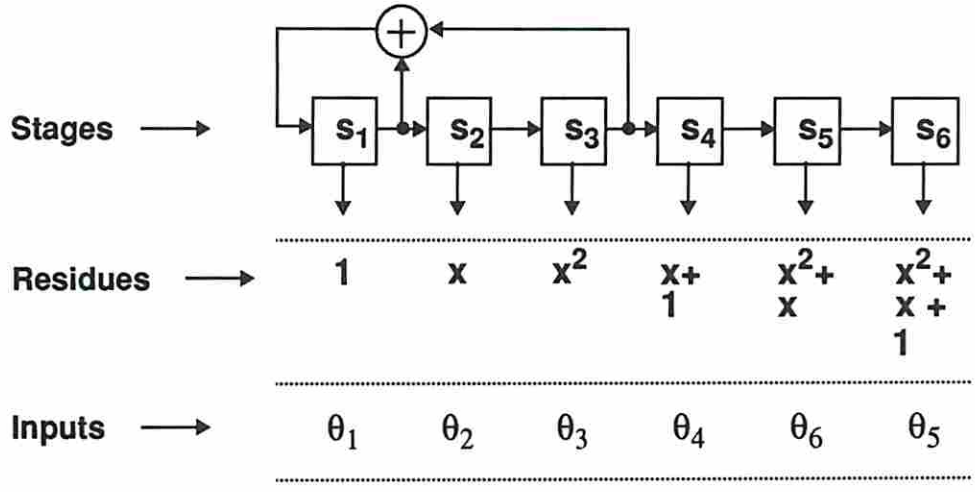


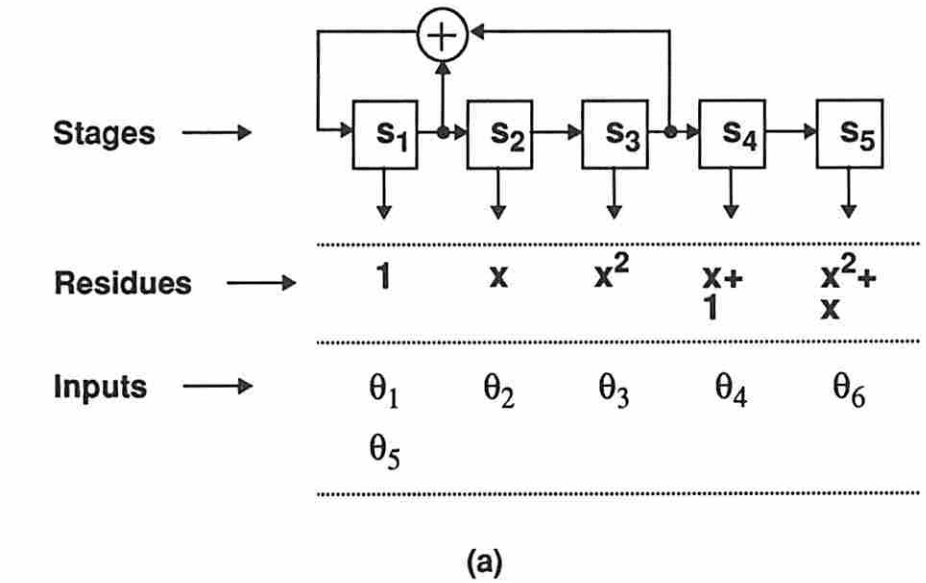
Figure 5.5: Permuted LFSR/SRs

5.2.2.3 Sharing LFSR/SRs

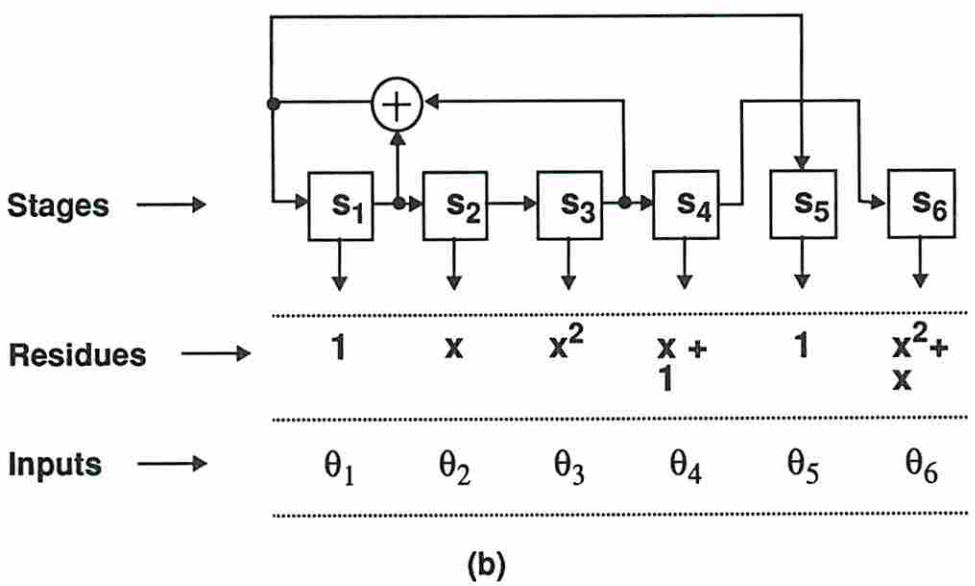
A sharing LFSR/SR [43] allows sharing of test signals among different inputs. In other words, a residue can be assigned to more than one input. Unrelated inputs are assigned the same residues and identical test signals are applied to them.

Example 10 For the $(6, 5, 3)$ circuit shown in Figure 5.2, a possible residue assignment for the inputs (allowing sharing of residues) is shown in the Figure 5.6(a). Inputs θ_1 and θ_5 are unrelated and hence share the residue r_1. The single LFSR/SR shown in Figure 5.6(a) has only five stages and is modified to a $(3, 6)$ sharing LFSR/SR shown in Figure 5.6(b). Stages s_1 and s_5 generate identical test signals. The $(3, 6)$ sharing LFSR/SR can exhaustively test all outputs with $2^3 = 8$ patterns. □

The operations discussed above enhance the capabilities of single LFSR/SRs only by a limited extent. In the next section we shall describe a new TPG design, called *convolved LFSR/SR*, which is a powerful extension to single LFSR/SR. For simplicity of presentation, we shall discuss convolved LFSR/SRs without reconfiguration of feedbacks, permutation of stages and sharing of test signals. Nevertheless convolved LFSR/SRs can also be constructed allowing these design operations.



(a)



(b)

Figure 5.6: Sharing LFSR/SRs

5.3 Convolved LFSR/SRs

A convolved LFSR/SR is derived from a single LFSR/SR design. Circuit inputs are sequentially assigned residues generated by the successive stages of a single LFSR/SR. During the assignment process, it may not be possible to assign a residue to an input due to linear dependencies for some output. Stages whose residues give rise to the problem of linear dependencies are skipped as shown in Figure 5.7(a). This single LFSR/SR ensures linear independence for all outputs but has more stages than the input register. The extra stages required by the single LFSR/SR can be avoided by using XOR gates as shown in Figure 5.7(b). The resulting structure is referred to as *convolved LFSR/SR.*

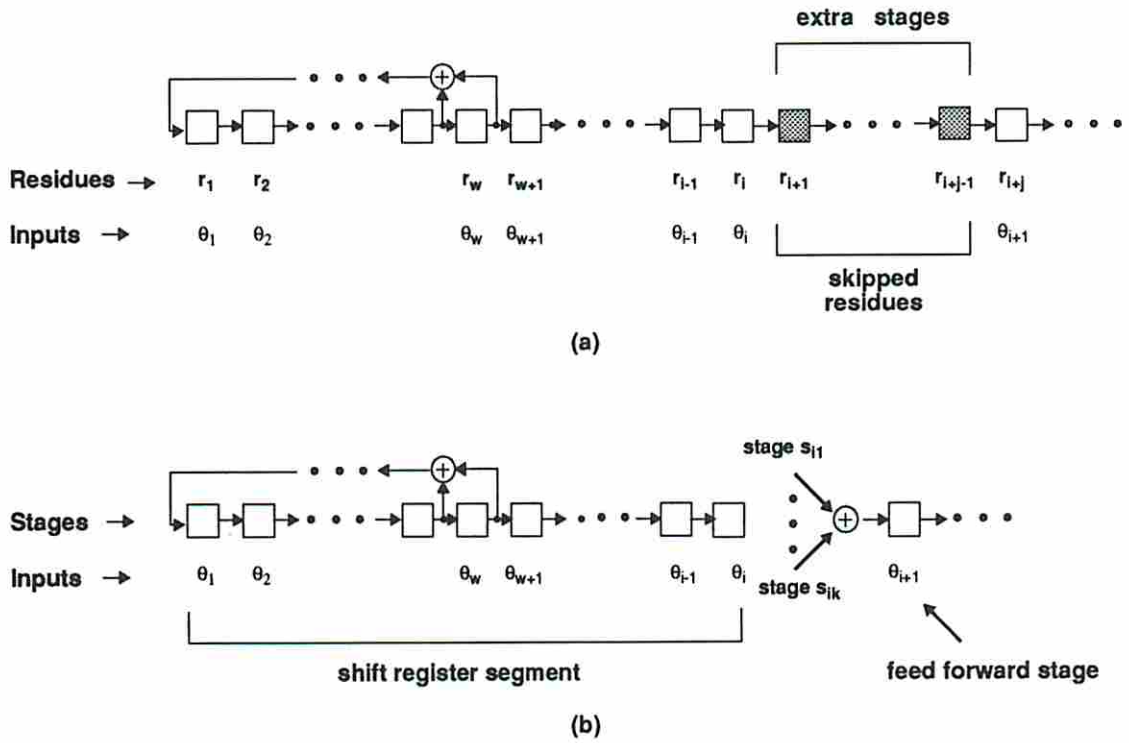
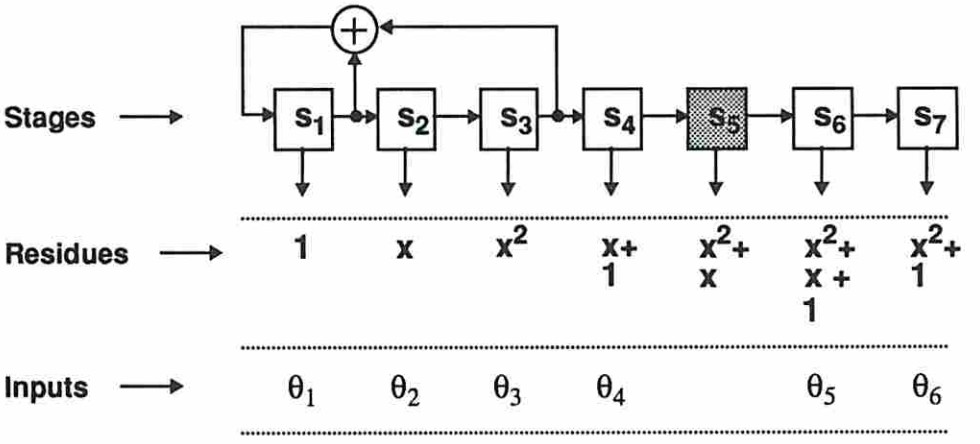


Figure 5.7: Convolved LFSR/SR: (a) Residue assignment for inputs; (b) TPG stages.

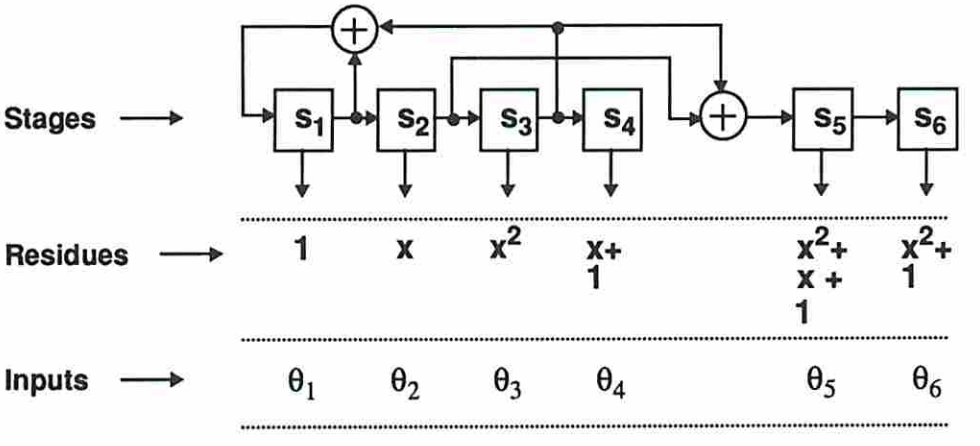
A (w, n) convolved LFSR/SR for an (n, m, k) circuit can be designed as follows. Residues generated by a single LFSR/SR of degree w are considered for assignment to circuit inputs. The inputs are assigned residues one at a time avoiding linear

dependencies with already assigned residues. Let the inputs θ_1 through θ_i be assigned the residues r_1 through r_i respectively. Stages s_1 through s_i of the convolved LFSR/SR are identical to the single LFSR/SR. Assume that input θ_{i+1} cannot be assigned any of the residues $r_{i+1}, r_{i+2}, \ldots, r_{i+j-1}$ because all of them are linearly dependent on already assigned residues for some output. Residue r_{i+j} is then selected for assignment to the input θ_{i+1} as shown in Figure 5.7(a). The single LFSR/SR requires $(j-1)$ extra stages (shown as shaded stages in the figure) whose residues are not assigned to inputs. These extra stages are avoided in the convolved LFSR/SR design. Stage s_{i+1} of the convolved LFSR/SR is made to generate the residue r_{i+j} by feeding the residue r_{i+j-1} at its input. Let the residue r_{i+j-1} be a linear combination of the residues $r_{i_1}, r_{i_2}, \ldots r_{i_k}$. Stages $s_{i_1}, s_{i_2}, \ldots, s_{i_k}$ generate the residues $r_{i_1}, r_{i_2}, \ldots, r_{i_k}$ respectively. These residues are combined using XOR gates and fed at the input of stage s_{i+1} as shown in Figure 5.7(b). The stage s_{i+1} is referred to as *feed forward stage* and the maximal set of contiguous stages (stages s_1 through s_i) is referred to as *shift register segment.* The assignment process is continued until all the inputs are assigned residues such that the residue sets for all output cones are linearly independent. The stages between the feed forward stages form shift register segments. The area overhead for the convolved LFSR/SR design is given by the number and size of XOR gates used to realize the individual feed forward stages.

Example 11 Let us design a $(3, 6)$ convolved LFSR/SR for the example circuit in Figure 5.2. Consider the residues from a single LFSR/SR based on $P_1(x)$: (x^3+x+1). Residues r_1 through r_4 are assigned to inputs θ_1 through θ_4 without any linear dependence problem. Residue r_5 cannot be assigned to input θ_5 since the residue set $\{r_2, r_3, r_5\} = \{x, x^2, x^2+x\}$ for output O_3 is linearly dependent. Skipping residue r_5, inputs θ_5 and θ_6 are assigned residues r_6 and r_7 respectively as shown in Figure 5.8(a). This assignment ensures linear independence of the residue sets for all five outputs. The extra stage required by the single LFSR/SR can be avoided using a two-input XOR gate for the convolved LFSR/SR. Stage s_5 of the convolved LFSR/SR can generate the residue r_6 by feeding r_5 at its input. Residue r_5 is obtained by combining the residues r_2 and r_3. Hence the linear combination of the outputs of stages s_2 and s_3 is fed as input to stage s_5 as shown in Figure 5.8(b). The convolved LFSR/SR can exhaustively test all outputs with $2^3 = 8$ patterns. □



(a)



(b)

Figure 5.8: Convolved LFSR/SR: (a) Residue assignment for inputs; (b) TPG stages.

Theorem 8 *A (w,n) convolved LFSR/SR exists for generating pseudo-exhaustive tests for an (n,m,k) circuit if and only if there exists a (w,n) LFSR/XOR for the circuit.*

Proof : *(If)* Consider any (w,n) LFSR/XOR designed for the (n,m,k) circuit. The LFSR/XOR can be transformed to an equivalent (w,n) convolved LFSR/SR as follows. The residues for the test signals generated by the LFSR/XOR stages are determined from the XOR network. These residues can be generated from the corresponding stages of the convolved LFSR/SR by making some stages as feed forward stages if necessary. In constructing the convolved LFSR/SR, design operations such as sharing of residues and permutation of inputs may be required.
(Only if) It is sufficient to show that any convolved LFSR/SR can be transformed to an equivalent LFSR/XOR. The residues of the convolved LFSR/SR stages are determined from the TPG structure. The XOR network for the LFSR/XOR can be designed such that the corresponding stages of the LFSR/XOR generate the assigned residues. □

Convolved LFSR/SRs have great potential to generate minimal test sets. They *bridge the gap* between LFSR/SRs and LFSR/XORs. A trivial convolved LFSR/SR is one having no XOR gates and is simply an LFSR/SR. On the other extreme, any stage of a convolved LFSR/SR can be made as a feed forward stage to generate any desired residue using XOR gates similar to LFSR/XOR. Typically convolved LFSR/SRs achieve low test lengths like LFSR/XORs and utilize low area overhead like LFSR/SRs. The linear independence for the outputs are assured by adding XOR gates to stages whenever necessary. However, most of the stages form shift register segments thereby avoiding high area overhead.

5.4 Multiple LFSR/SRs

A multiple LFSR/SR forms a special case of convolved LFSR/SR. It is composed of two or more independent single LFSR/SRs that are run in parallel. The single LFSR/SRs have identical feedback polynomials but may have different shift register lengths and initial seeds.

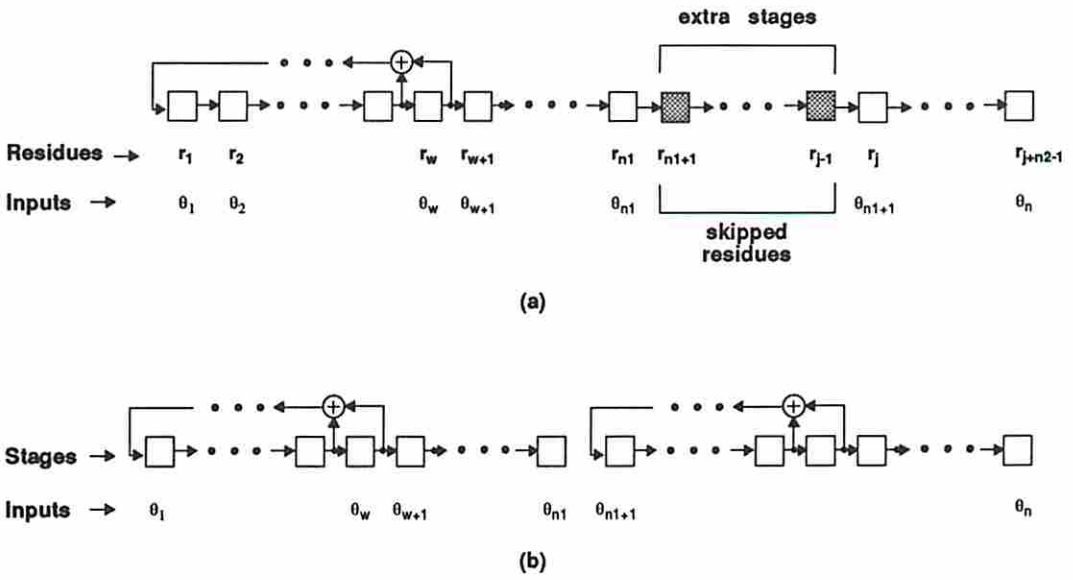


Figure 5.9: Multiple LFSR/SR: (a) Residue assignment for inputs; (b) TPG stages.

A multiple LFSR/SR is derived from a single LFSR/SR. A multiple LFSR/SR for an (n, m, k) circuit can be determined as follows. Consider the residue assignment for the inputs as shown in Figure 5.9(a) ensuring linear independence for all output cones. Let both n_1 and n_2 be greater than or equal to w and the sum of n_1 and n_2 be n. Residues r_1 through r_{n_1} are assigned to inputs θ_1 through θ_{n_1} respectively. Residues r_{n_1+1} through r_{j-1} are skipped and not assigned to any input. Residues r_j through r_{j+n_2-1} are assigned to inputs θ_{n_1+1} through θ_n respectively. The single LFSR/SR requires $(j - n_1 - 1)$ extra stages (shown as shaded stages in the figure) whose residues are not assigned to inputs. These extra stages are avoided in the multiple LFSR/SR design. Stages s_1 through s_{n_1} of the input register are modified to an (w, n_1) single LFSR/SR. Stages s_{n_1+1} through s_n of the input register are modified to an (w, n_2) single LFSR/SR. A (w, n) multiple LFSR/SR composed of (w, n_1) and (w, n_2) single LFSR/SRs is shown in Figure 5.9(b). Both single LFSR/SRs have identical LFSRs of degree w.

A residue assigned to an input can be expressed as a linear combination of the residues $r_1, r_2, \ldots, r_w$. The residues $r_1, r_2, \ldots, r_w$ are generated by the LFSR stages of the first single LFSR/SR. Let an initial seed S be applied to the LFSR portion

of the first single LFSR/SR. From this seed, the initial contents for the rest of the stages of the multiple LFSR/SR can be determined. For example, if an input θ_i is assigned a residue which is a linear combination of the residues $r_{i_1}, r_{i_2}, \ldots, r_{i_k}$ (where $i_1, i_2, \ldots, i_k < w$), then the initial content of stage s_i is a linear combination of the initial contents of the stages $s_{i_1}, s_{i_2}, \ldots, s_{i_k}$. This initialization ensures that stage s_i of the multiple LFSR/SR generates the residue assigned to input θ_i.

Example 12 A multiple LFSR/SR for the $(6, 5, 3)$ circuit shown in Figure 5.2 can be designed as follows. The residue assignment for the inputs shown in Figure 5.10(a) satisfy the linear independence of residue sets for all outputs. We can construct a multiple LFSR/SR composed of two $(3, 3)$ single LFSR/SRs as shown in Figure 5.10(b). Both single LFSR/SRs are based on the same primitive polynomial $x^3 + x + 1$. The stages of the first LFSR/SR generates residues r_1, r_2 and r_3. Let an initial seed $S_1 = 100$ be applied to the first LFSR/SR. The initial seed S_2 for the second LFSR/SR is determined such that its stages generate the residues r_5, r_6 and r_7. Input θ_4 is assigned the residue r_5 which is the linear combination of the residues r_2 and r_3. Hence the initial content of stage s_4 is zero which is a linear combination of the initial contents of the stages s_2 and s_3. Thus the initial seed of the second LFSR/SR is computed as $S2 = 011$. The stages of the multiple LFSR/SR generate the assigned residues shown in Figure 5.10(a). This multiple LFSR/SR generates $2^3 = 8$ patterns to exhaustively test all the five outputs. □

The following lemma characterizes the relation between multiple LFSR/SRs and convolved LFSR/SRs.

Theorem 9 *A (w, n) multiple LFSR/SR exists for generating pseudo-exhaustive tests for an (n, m, k) circuit if and only if there exists a (w, n) convolved LFSR/SR for the circuit where the length of each shift register segment is at least w.*

Proof : *(If)* For the (w, n) convolved LFSR/SR, since the shift register segments are of length at least w, each one of them can be modified as an independent single LFSR/SR with the same feedback polynomial as that of the convolved LFSR/SR. The initial seeds for the single LFSR/SRs are the same as the initial seeds of the shift register segments of the convolved LFSR/SR.

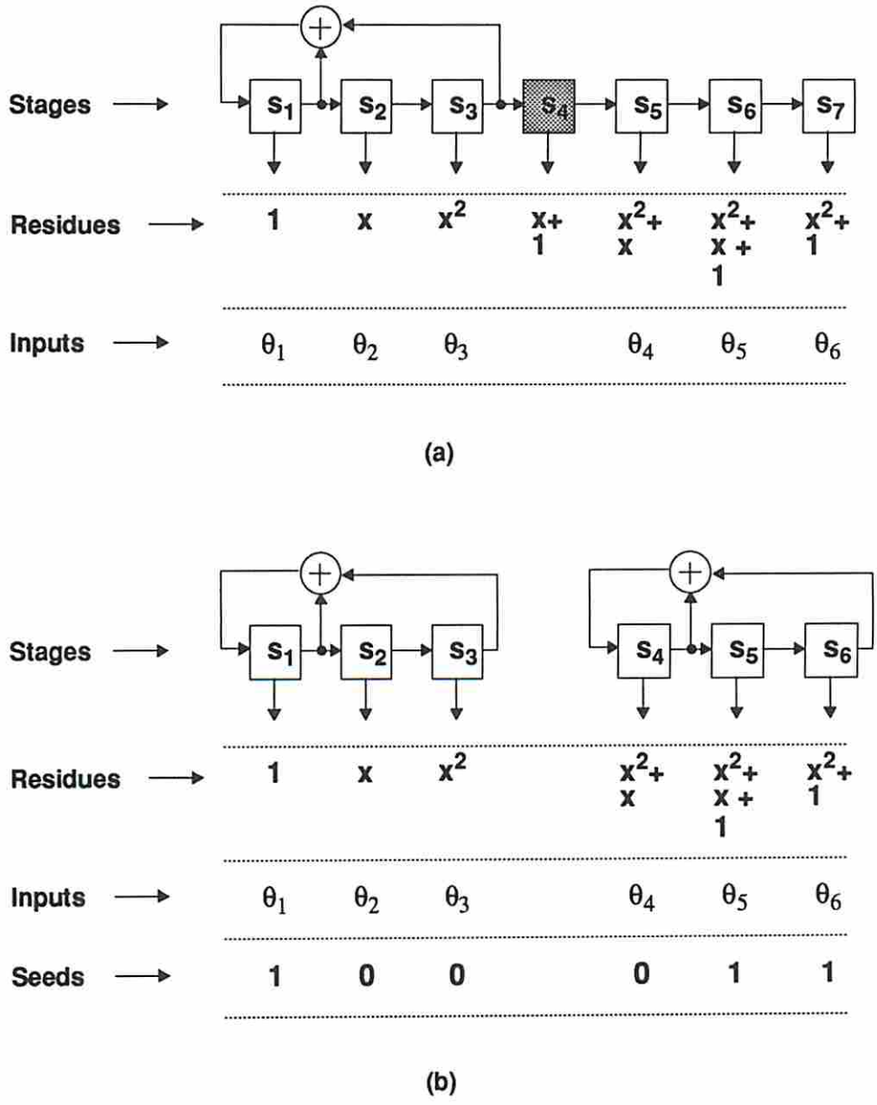


Figure 5.10: Multiple LFSR/SR: (a) Residue assignment for inputs; (b) TPG stages.

For an (n, m, k) circuit, a (w, n) convolved LFSR/SR can be designed as follows. A primitive polynomial $P(x)$ of degree w is selected as the feedback polynomial. The polynomial can generate $(2^w - 1)$ unique residues from r_1 through r_{2^w-1} and all of them can be considered for assignment to the inputs. The number of possible residues is exponential with the degree of the polynomial. Practically only a few residues, say r_1 through r_N, are considered for input assignment. This practical consideration limits the convolved LFSR/SR designs. The shift register segments are constrained to have a desired minimum length, say l. This constraint attempts to reduce the number of feed forward stages and hence the area overhead due to XOR network. A (w, n) multiple LFSR/SR can be obtained by restricting $l \geq w$ and a (w, n) single LFSR/SR can be obtained by restricting $l = n$.

Procedure TPG

Input: Input dependencies for all the outputs of (n, m, k) circuit;
A primitive polynomial $P(x)$ of degree $w \geq k$;
(l) :: minimum length requirement for the shift register segments;
(N) :: maximum number of residues considered for assignment.

Output: Residue assignment for the inputs;
Initial seeds for the shift register segments;
XOR network for the feedforward stages.

1. **residues** ();
   /* determines residues for the inputs such that the sets of residues for all outputs are linearly independent. */

2. **seeds** ();
   /* determines the seeds for the TPG stages such that the inputs are driven by the test signals that represent the corresponding assigned residues. */

3. **network** ();
   /* determines the XOR network to implement the linear combinations of the test signals for the feedforward stages. */

Procedure residues ()

1. Assign residues r_1 through r_w to inputs θ_1 through θ_w.

2. $i \leftarrow (w+1)$; $j \leftarrow (w+1)$.
   /* residue r_j is assigned to input θ_i */

3. While not all inputs are assigned residues do

   (a) If $(i = w)$ and $(j = w+1)$ exit with failure.

   (b) If $(N - j) < (n - i)$ /* not enough residues left for remaining inputs */
   then $\{i \leftarrow i'$; $j \leftarrow (j'+1)\}$
   (where $i' = i - 1$ and $r_{j'}$ is the residue assigned to input $\theta_{i'}$).

(c) Assign residue r_j to input θ_i.

(d) If the assigned residues are linearly independent for all outputs, then $i \leftarrow (i+1)$.

(e) $j \leftarrow (j+1)$.

(f) If the last shift register segment length $< l$ /* requirements not met */ then $\{i \leftarrow i''; \; j \leftarrow (j''+1)\}$
(where i'' is the first input in the last shift register segment and $r_{j''}$ is the residue assigned to input $\theta_{i''}$)

4. Print the residue assignment for the inputs.

Procedure seeds ()

1. For every input θ_i do

   (a) Determine the residue $r_{i'}$ assigned to input θ_i.

   (b) If $r_{i'}$ contains the term r_1 $(= 1)$ initialize stage s_i to 1, else initialize stage s_i to 0.

2. Print the initial values of all stages.

Procedure network ()

1. For every feed forward stage s_i do

   (a) Determine the residue $r_{i'}$ that should appear at the input of stage s_i.

   (b) $R \leftarrow \emptyset$. /* R contains the collection of residues to realize $r_{i'}$ */

   (c) While $(r_{i'} \neq 0)$ do

      i. Determine the residue $r_{j'}$ from a stage j $(< i)$ which differs from the residue $r_{i'}$ in minimum number of terms.

      ii. $R \leftarrow R \cup r_{j'}$: $\quad r_{i'} \leftarrow r_{i'} \oplus r_{j'}$.

   (d) Combine the residues in R using two-input XOR gates.

5.5.1 Determination of Residues

The iterative search procedure progressively assigns residues to the inputs. The linear independence among the residues for all outputs is checked after every assignment. Backtracking occurs whenever there are not enough residues left for the unassigned inputs or the shift register segments lengths become less than l. The procedure reports after all inputs are assigned residues. Alternate primitive polynomials must be tried if a convolved LFSR/SR cannot be determined with $P(x)$. Convolved LFSR/SR will generate the minimum test set only if the degree of the selected polynomial equals k. Note that if the TPG designed by this procedure requires test length greater than 2^k, then operations such as reconfiguration, permutation and sharing may be used to obtain TPG with lower test lengths.

5.5.2 Determination of Seeds

The first w stages generate independent signals and these stages are initialized with the seed $100\ldots0$. All other stages generate linear combinations of these signals as given by their assigned residues. For stage s_i, if the assigned residue is a linear combination of the residues $r_{i_1}, r_{i_2}, \ldots, r_{i_k}$ (where $i_1, i_2, \ldots, i_k < w$), then the initial content of stage s_i is a linear combination of the initial contents of the stages $s_{i_1}, s_{i_2}, \ldots, s_{i_k}$. Among the stages $s_1, s_2, \ldots, s_w$, only stage s_1 has the initial content of non-zero value. Hence stage s_i is initialized to non-zero value only if its assigned residue contains the term $r_1 = 1$. The initial seed determined by this method ensures that the TPG stages generate their respective assigned residues.

5.5.3 Determination of XOR network

The feedback and the feedforward stages need XOR gates to realize their assigned residues. The XOR network required by the feedback stage is specified by its feedback taps. The XOR network for a feed forward stage s_i is determined as follows. From the residue assigned to the input θ_i, the residue (say $r_{i'}$) of the test signal that needs to be fed at the input of stage s_i is determined. This residue $r_{i'}$ is obtained as the linear sum of a minimal number of residues from the earlier stages. The

procedure *network* gives an upper bound on the number of two-input XOR gates to realize the feed forward stages.

Multiple LFSR/SRs can be realized if the shift register segments are of length at least equal to the degree of the feedback polynomial $P(x)$. In that case, each shift register segment is transformed into an independent single LFSR/SR with $P(x)$ as the feedback polynomial.

5.6 Experimental Results

We have designed various pseudo-exhaustive TPGs for the ISCAS combinational benchmark circuits [7]. The circuits were partitioned using our partitioning procedure [34] such that the output cones are restricted to have twenty or less inputs. The test time for the TPGs are tabulated in Table 5.1. The first three columns in the table describe the characteristics of the benchmark circuits before and after the application of the partitioning procedure. Three circuit specific TPGs — convolved LFSR/SRs, multiple LFSR/SRs and single LFSR/SRs — were designed for the partitioned circuits. The last three columns in the table denote the pseudo-exhaustive test time for these TPG designs.

Ckt	(n,m,k)		Test Length		
	Before Partitioning	After Partitioning	Convolved LFSR/SR	Multiple LFSR/SR	Simple LFSR/SR
c432	(36,7,36)	(56,27,20)	2^{20}	2^{20}	2^{20}
c499	(41,32,41)	(49,40,14)	2^{14}	2^{14}	2^{15}
c880	(60,26,45)	(70,36,17)	2^{17}	2^{17}	2^{17}
c1355	(41,32,41)	(49,40,14)	2^{14}	2^{14}	2^{15}
c1908	(33,25,33)	(47,39,20)	2^{20}	2^{20}	2^{21}
c2670	(233,140,120)	(262,169,20)	2^{20}	2^{20}	2^{20}
c3540	(50,22,50)	(108,80,20)	2^{20}	2^{20}	2^{22}
c5315	(178,123,67)	(215,160,20)	2^{20}	2^{20}	2^{22}
c6288	(32,31,32)	(99,98,20)	2^{20}	2^{20}	—
c7552	(207,108,194)	(286,187,20)	2^{20}	2^{20}	2^{22}

Table 5.1: Test Length for partitioned benchmark circuits

The TPGs effectively utilize the information about the circuit output cone dependencies. For each circuit, only two primitive polynomials were considered while designing convolved LFSR/SRs and multiple LFSR/SRs and up to 100 primitive polynomials were considered while designing single LFSR/SRs. Both convolved LFSR/SRs and multiple LFSR/SRs generate minimum test sets for all the partitioned circuits. For example, the partitioned $c3540$ circuit has a maximum dependency of 20 inputs and hence any pseudo-exhaustive test set must contain at least 2^{20} patterns. Both convolved LFSR/SR and multiple LFSR/SR generate minimum pseudo-exhaustive test lengths. On the other hand, a single LFSR/SR generates a test set with 2^{22} patterns.

LFSR/XORs can be designed by transforming the convolved LFSR/SR designs (refer to Theorem 8). LFSR/XORs will also generate minimum test sets for all the partitioned circuits. However, too many XOR gates may be needed for realizing the LFSR/XOR structures.

Single LFSR/SRs were determined by considering only 100 primitive polynomials of each degree because of practical limitations. Due to this restriction, there may exist single LFSR/SRs for the partitioned circuits with smaller degrees than the tabulated designs. For the partitioned $c6288$ circuit, no single LFSR/SR design was found in spite of trying 100 primitive polynomials of degrees from 20 to 40. The circuit contains a few outputs driven by 19 consecutive inputs and one non-consecutive input. Single LFSR/SRs failed to generate exhaustive test sets for these cones. On the other hand, both convolved LFSR/SRs and multiple LFSR/SRs generated test sets due to their flexibility in assigning residues. Another set of convolved LFSR/SR designs for the partitioned benchmark circuits can be found in [35].

Table 5.2 presents the details about the residue assignment for the convolved LFSR/SR designs for the partitioned benchmark circuits. The first two columns provide the characteristics of the partitioned circuits. The exponent terms for the feedback polynomial, the residue assignment for the stages and the length of shift register segments are given by the third, fourth and fifth columns respectively. For example, for the partitioned $c432$ circuit, convolved LFSR/SR design is based on the feedback polynomial $P(x): \ x^{20} + x^6 + x^4 + x + 1$. The residue assignment for all the 56 input register stages are given by the fourth column. Stages s_1 through

Ckt	(n,m,k)	Polynomial	Residue Assignment	SR Segments
c432	(56,27,20)	20 6 4 1 0	1-36 49-68	36 20
c499	(49,40,14)	14 5 3 1 0	1-16 53-68 151-167	16 16 17
c880	(70,36,17)	17 3 0	1-24 118-146 194-210	24 29 17
c1355	(49,40,14)	14 5 3 1 0	1-16 53-68 151-167	16 16 17
c1908	(47,39,20)	20 6 4 1 0	1-20 56-82	20 27
c2670	(262,169,20)	20 3 0	1-100 103-244 260-279	100 142 20
c3540	(108,80,20)	20 3 0	1-20 129-152 157-176	20 24 20
			210-232 379-399	23 21
c5315	(215,160,20)	20 3 0	1-110 121-142 156-177	110 22 22
			208-227 577-597 683-702	20 21 20
c6288	(99,98,20)	20 6 4 1 0	1-20 209-229 272-292	20 21 21
			350-386	37
c7552	(286,187,20)	20 3 0	1-49 51-114 133-190 196-229	49 64 58 34
			246-267 290-310 456-493	22 21 38

Table 5.2: Convolved LFSR/SR designs for partitioned benchmark circuits

s_{36} are assigned residues r_1 through r_{36} respectively. Stage s_{37} forms a feed forward stage and stages s_{37} through s_{56} are assigned residues r_{49} through r_{68}. The residue r_i is given by x^{i-1} mod $P(x)$. The lengths of the two shift register segments are 36 and 20 respectively. All convolved LFSR/SR designs have very few shift register segments and each segment is of length at least equal to the degree of the LFSR. The residue assignment enables to realize multiple LFSR/SRs for the circuits.

Table 5.3 presents the details about the single LFSR/SR designs for the partitioned benchmark circuits. The feedback polynomials and the residue assignments are given by the third and fourth columns respectively. The hardware overhead involved in these designs is given by the number of 2-input XOR gates required to realize the LFSR structure. For example, for the partitioned $c432$ circuit, seven XOR gates are needed to realize the LFSR of degree 20.

The comparison of test lengths and hardware overhead among the TPG designs are given in Table 5.4. Columns 3, 5 and 7 presents the pseudo-exhaustive test lengths for the TPG designs. The hardware overhead in terms of 2-input XOR gates required for realizing the TPG designs are given in columns 4, 6 and 8 respectively. For example, the number of XOR gates utilized in TPG designs for the partitioned

Ckt	(n,m,k)	Polynomial	Residue Assignment
c432	(56,27,20)	20 11 7 5 4 3 2 1 0	1-56
c499	(49,40,14)	15 9 8 6 5 3 0	1-49
c880	(70,36,17)	17 10 8 6 5 3 2 1 0	1-70
c1355	(49,40,14)	15 9 8 6 5 3 0	1-49
c1908	(47,39,20)	21 8 6 3 2 1 0	1-47
c2670	(262,169,20)	20 9 5 4 0	1-262
c3540	(108,80,20)	22 11 10 7 4 2 0	1-108
c5315	(215,160,20)	22 9 7 5 3 2 0	1-215
c6288	(99,98,20)	—	—
c7552	(286,187,20)	22 9 8 6 5 3 0	1-286

Table 5.3: Single LFSR/SR designs for partitioned benchmark circuits

		Convolved		Multiple		Simple	
Ckt	(n,m,k)	TL	XOR	TL	XOR	TL	XOR
c432	(56,27,20)	2^{20}	6	2^{20}	6	2^{20}	7
c499	(49,40,14)	2^{14}	10	2^{14}	9	2^{15}	5
c880	(70,36,17)	2^{17}	7	2^{17}	3	2^{17}	7
c1355	(49,40,14)	2^{14}	10	2^{14}	9	2^{15}	5
c1908	(47, 39,20)	2^{20}	10	2^{20}	6	2^{21}	5
c2670	(262,169,20)	2^{20}	3	2^{20}	3	2^{20}	3
c3540	(108,80,20)	2^{20}	11	2^{20}	5	2^{22}	5
c5315	(215,160,20)	2^{20}	7	2^{20}	6	2^{22}	5
c6288	(99,98,20)	2^{20}	18	2^{20}	12	–	–
c7552	(286,187,20)	2^{20}	11	2^{20}	7	2^{22}	5

Table 5.4: Comparison among circuit specific TPG designs

$c432$ circuit is computed as follows. The convolved LFSR/SR residue assignment for $c432$ comprises of two shift register segments with stage 36 forming a feed forward stage. This stage can be realized as the linear sum of the test signals from the output of stages 7, 9, 15 and 19 using three XOR gates. The LFSR requires three XOR gates and hence the convolved LFSR/SR requires six XOR gates. For this circuit, a multiple LFSR/SR can be realized by transforming the two shift register segments into two independent single LFSR/SRs. Since each LFSR requires three XOR gates, the multiple LFSR/SR design also requires six XOR gates. On the other hand, a single LFSR/SR design requires seven XOR gates for this circuit.

Both convolved LFSR/SRs and multiple LFSR/SRs generate minimum test sets for all the partitioned benchmark circuits. However, multiple LFSR/SR designs usually incur less hardware overhead than convolved LFSR/SRs. It should be noted that only an upper bound on the number of XOR gates required for convolved LFSR/SRs is given in the table. This is due to the fact that the procedure *network* uses a suboptimal heuristic to determine the XOR network for the feed forward stages. Single LFSR/SRs generate minimum test sets only for three circuits $c432$, $c880$ and $c2670$. For these circuits, multiple LFSR/SRs utilize fewer XOR gates than single LFSR/SRs. For the remaining circuits, single LFSR/SRs generate up to four times the size of the minimum test sets. For these circuits, multiple LFSR/SRs require less than twice the number of XOR gates utilized by single LFSR/SRs.

Figure 5.11 highlights the characteristics of various TPG designs based on the empirical observations. Among the four TPG designs, single LFSR/SRs generate the largest test sets but incur the least hardware overhead. On the other extreme, LFSR/XORs generate smallest test sets but utilize maximum hardware. Convolved LFSR/SRs generate small test sets like LFSR/XORs but utilize less hardware. The arrows indicate the possible transformations from one TPG design to another. Any convolved LFSR/design can be transformed to an LFSR/XOR design and vice versa. Similarly, any multiple LFSR/SR design can be transformed to a convolved LFSR/SR design. However, a convolved LFSR/SR design can be transformed to a multiple LFSR/SR design provided the shift register segments are of lengths greater than or equal to the degree of the LFSR (refer to Theorem 9). Similarly, a multiple LFSR/SR design can be transformed to a single LFSR/SR design provided the length of the shift register segment equals the number of inputs to the circuit.

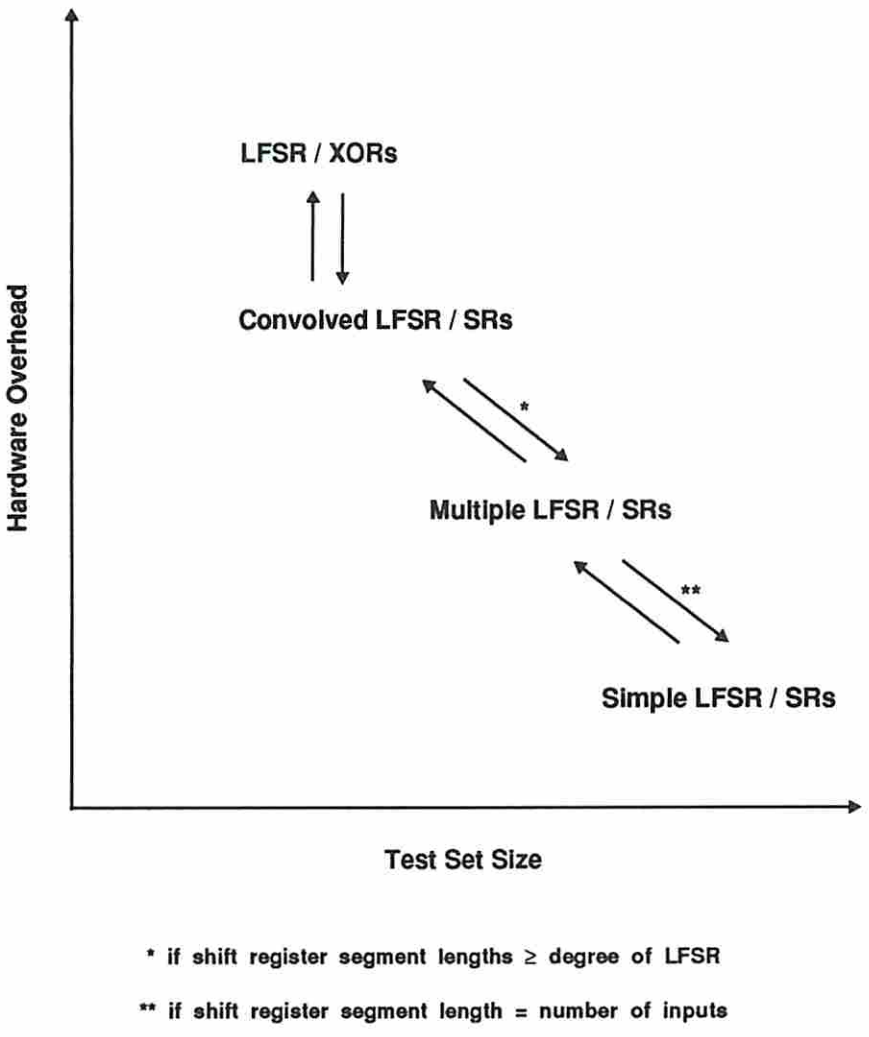


Figure 5.11: Characteristics of various TPGs

5.7 Summary

In this chapter we have described hardware efficient TPG designs to generate minimal pseudo-exhaustive test sets. These TPGs are tailored for a given circuit and utilize the information about the circuit output cone dependencies. Convolved LFSR/SRs have great potential to generate minimum test sets as demonstrated by the experiments on the combinational benchmark circuits. These structures can be used to derive other test pattern generators such as LFSR/XORs and multiple LFSR/SRs. The flexibility of manipulating residues with XOR network that was absent in single LFSR/SRs but present in convolved LFSR/SRs makes convolved LFSR/SRs powerful test pattern generators.

Chapter 6

Bounds on Test Lengths

6.1 Introduction

Circuit-specific TPGs such as LFSR/XORs [3] and LFSR/SRs [4] can be designed to generate pseudo-exhaustive test sets for a given circuit. In this chapter we shall derive tight upper bounds on the sizes of test sets generated by these TPG structures. We shall first derive a few important algebraic results that are used in the derivation of the bounds on pseudo-exhaustive test lengths. We shall then determine a few generic bounds on test lengths that are independent of the structural information about the circuit output cones. Circuit-specific bounds are later derived by utilizing the structural information about the circuit output cones.

6.2 Algebraic Results

We shall present a few algebraic definitions that are used in the bound computations. Our definitions of *Abelian group* and *vector space* are simplified due to the fact that the group is defined over modulo-2 addition (denoted as $+$) and the space is defined over the Galois field GF(2).

Definition 2 *(Abelian Group and Vector Space)*

- *A non-empty set S is an* **Abelian group** *if S is closed under modulo-2 addition.*
- *A non-empty set S is a* **(vector) space** *over GF(2) if S is an Abelian group under modulo-2 addition.*

- *The* **(linear) span** *of a non-empty set B, denoted as $L(B)$, is the set of all linear combinations (modulo-2 addition) of elements in B.*

- *Any subset B of a vector space S is a* **basis** *of S if B consists of linearly independent elements and $L(B) = S$.*

- *The* **dimension** *of a vector space S spanned by basis B equals $|B|$.*

The modulo-2 addition operation satisfies the group properties such as commutativity, associativity and existence of additive inverses. Hence it suffices to check only the closure property to validate an Abelian group. GF(2) forms a field with respect to modulo-2 addition and modulo-2 multiplication operations and satisfies all the axioms defined for a vector space. Hence it suffices to check only the closure property to validate a vector space.

A k-dimensional space S spanned by a basis B consists of 2^k elements. The space S forms an Abelian group under modulo-2 addition operation. We shall next define operations between subspaces.

Definition 3 *(Operations between Vector Subspaces)*

- *A non-empty subset S_1 of a vector space S over GF(2) is a* **subspace** *if S_1 is an Abelian group under modulo-2 addition.*

- *The* **direct sum** *of two subspaces S_1 and S_2, denoted as $S_1 \oplus S_2$, equals the set $\{s_1 + s_2 \mid s_1 \in S_1,\ s_2 \in S_2\}$.*

- *The* **set union** *of two subspaces S_1 and S_2, denoted as $S_1 \cup S_2$, equals the set $\{s \mid s \in S_1$ or $s \in S_2\}$.*

- *The* **set intersection** *of two subspaces S_1 and S_2, denoted as $S_1 \cap S_2$, equals the set $\{s \mid s \in S_1$ and $s \in S_2\}$.*

Conventional algebraic theory deals with direct sum operation between vector subspaces. In contrast, our bound computations are based on set union and intersection operations between subspaces.

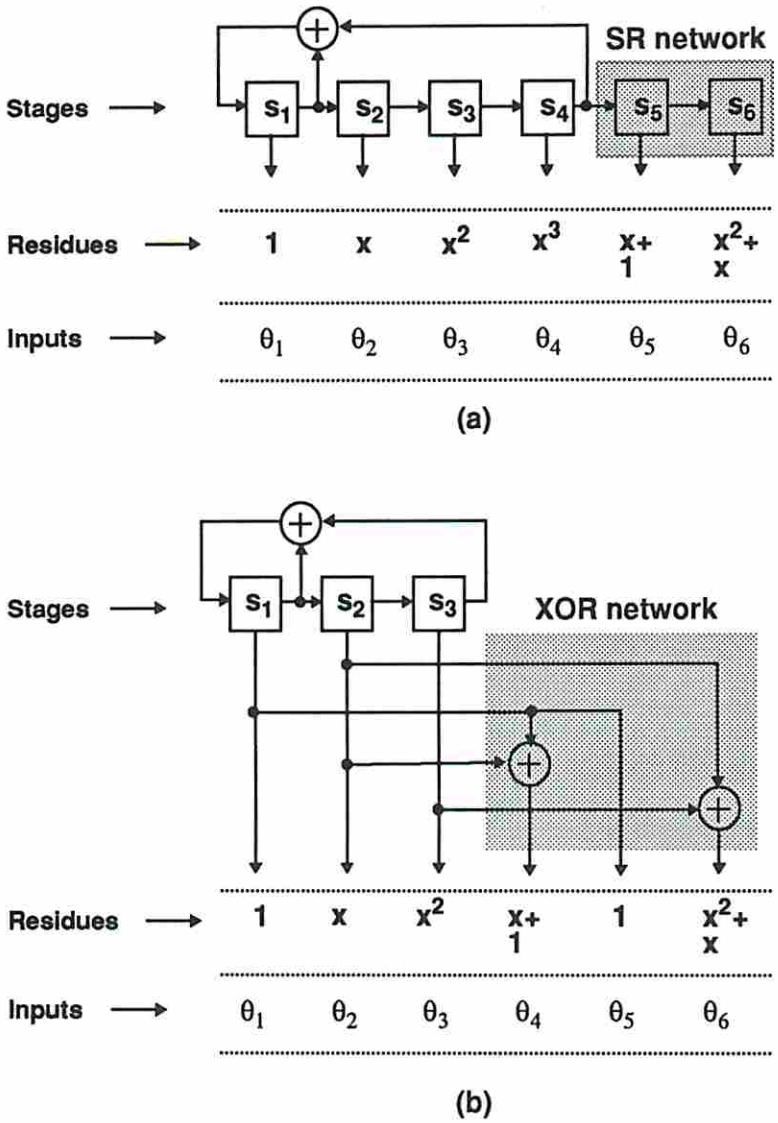


Figure 5.3: TPG Structures (a) LFSR/SR and (b) LFSR/XOR

Consider a $(3,6)$ LFSR/SR based on the primitive polynomial $P_1(x)$: (x^3+x+1). Inputs θ_1 through θ_6 are driven by six unique test signals represented by the residues $1, x, x^2, x+1, x^2+x$ and x^2+x+1 respectively. The residue set for O_3 given by $\{x, x^2, x^2+x\}$ is linearly dependent. Hence the single LFSR/SR based on $P_1(x)$ cannot test O_3 exhaustively.

Consider a $(3,6)$ LFSR/SR based on the primitive polynomial $P_2(x)$: $(x^3 + x^2 + 1)$. Inputs θ_1 through θ_6 are driven by six unique test signals represented by the residues $1, x, x^2, x^2+1, x^2+x+1$ and $x+1$ respectively. The residue set for O_2 given by $\{1, x^2, x^2+1\}$ is linearly dependent. Hence the single LFSR/SR based on $P_2(x)$ cannot test O_2 and O_5 exhaustively. Since $P_1(x)$ and $P_2(x)$ are the only two primitive polynomials of degree three, it is not possible to design a TPG for this circuit based on any $(3,6)$ single LFSR/SR.

Consider a $(4,6)$ LFSR/SR based on the primitive polynomial $P_3(x)$: (x^4+x+1). Inputs θ_1 through θ_6 are driven by six unique test signals represented by the residues $1, x, x^2, x^3, x+1$ and x^2+x respectively. The residue sets for O_2, O_3 and O_4 given by $\{1, x^2, x^3\}$, $\{x, x^2, x+1\}$ and $\{x, x^3, x^2+x\}$ are linearly independent. Hence the single LFSR/SR based on $P_3(x)$ exhaustively tests all the five outputs of the circuit. The single LFSR/SR structure is shown in Figure 5.3(a).

Consider a $(3,6)$ LFSR/XOR based on the primitive polynomial $P_1(x)$: $(x^3 + x + 1)$. Inputs θ_1, θ_2 and θ_3 are driven by three unique test signals represented by the residues $1, x$ and x^2 respectively. Inputs θ_4, θ_5 and θ_6 are driven by three specific linear combinations given by the residues $x+1, 1$ and x^2+x respectively. The residue sets for the outputs O_1 through O_5 given by $\{1, x, x^2\}$, $\{1, x^2, x+1\}$, $\{x, x^2, 1\}$, $\{x, x+1, x^2+x\}$ and $\{x^2, 1, x^2+x\}$ are linearly independent. Hence the LFSR/XOR based on $P_1(x)$ exhaustively tests all the five outputs of the circuit. The LFSR/XOR structure is shown in Figure 5.3(b).

Note that the LFSR/SR and LFSR/XOR in Figure 5.3 are based on primitive polynomials of degree four and three respectively. Hence the LFSR/SR generates a test set consisting of sixteen patterns while the LFSR/XOR generates an optimal test set consisting of only eight patterns. However, the LFSR/XOR does not utilize all the stages of the input register and incurs hardware overhead due to XOR network. □

5.2.2 Operations on LFSR/SRs

Design operations such as *reconfiguration of feedbacks*, *permutation of stages* and *sharing of test signals* can be applied on single LFSR/SRs to obtain *reconfigurable* LFSR/SRs, *permuted* LFSR/SRs and *sharing* LFSR/SRs respectively. These operations have the potential to avoid linear dependencies in the residue sets of the output cones but result in increased hardware overhead. We shall next describe TPG structures that result from these operations on single LFSR/SRs.

5.2.2.1 Reconfigurable LFSR/SRs

A reconfigurable LFSR/SR [39] has the capability to reconfigure its feedback taps to realize different primitive polynomials for different test sessions. For a single LFSR/SR, a *single* feedback polynomial is selected such that the residue sets for all cones are linearly independent. For a reconfigurable LFSR/SR, a *minimal set* of feedback polynomials is selected such that the residue set for each cone is linearly independent for at least one of the polynomials. The feedback taps are reconfigured using multiplexers to realize a different primitive polynomial during each test session. A subset of output cones is exhaustively tested during each session and each cone will be exhaustively tested in at least one of the sessions. Reconfiguration hardware overhead can be minimized by judiciously selecting polynomials that have common feedback taps.

Example 8 For the $(6,5,3)$ circuit shown in Figure 5.2, a single LFSR/SR based on $P_1(x): x^3+x+1$ exhaustively tests the outputs O_1, O_2, O_4 and O_5. Similarly, a single LFSR/SR based on $P_2(x): x^3+x^2+1$ exhaustively tests the outputs O_1, O_3 and O_4. A reconfigurable LFSR/SR based on the polynomials $P_1(x)$ and $P_2(x)$ is shown in Figure 5.4. Polynomials $P_1(x)$ and $P_2(x)$ are realized during the first and the second test session respectively. This arrangement ensures that all the outputs are exhaustively tested at least once. The TPG generates a test set with $2 \times 2^3 = 16$ patterns. □

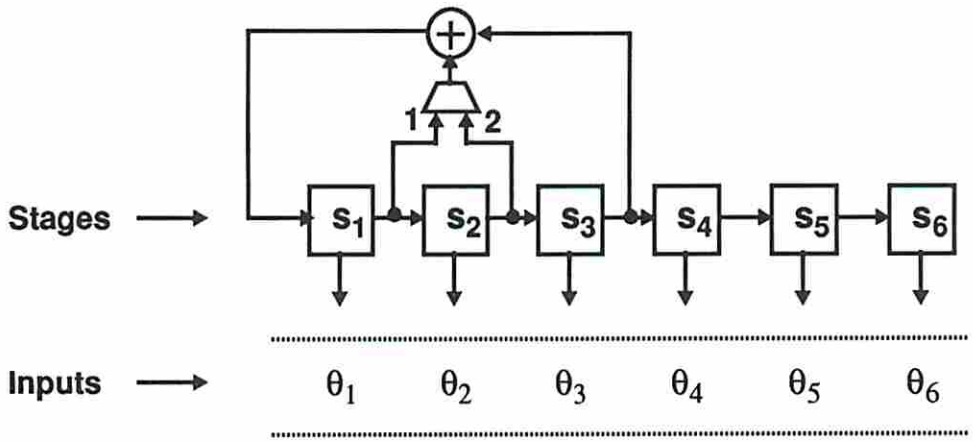


Figure 5.4: Reconfigurable LFSR/SR

5.2.2.2 Permuted LFSR/SRs

A permuted LFSR/SR [22] is essentially a single LFSR/SR whose stages form a permution of stages of the input register. In other words, stage s_i of the TPG need not necessarily drive the input θ_i of the circuit. The inputs are permuted such that the residue sets for all cones are linearly independent. A single LFSR/SR that could not be used because of the fixed assignment of residues may lead to an acceptable solution after reassigning the residues. The permutation of the inputs can result in hardware overhead due to routing among the TPG stages.

Example 9 A permuted LFSR/SR based on the primitive polynomial $P_1(x)$: $(x^3 + x + 1)$ can be determined for the $(6, 5, 3)$ circuit in Figure 5.2 as follows. Residues are assigned to the inputs such that the residue sets for all outputs are linearly independent. The residue assignment for the inputs and the ordering of the stages for the $(3, 6)$ permuted LFSR/SR is shown in Figure 5.5. Stages s_1 through s_4 drive inputs θ_1 through θ_4 respectively. Stages s_5 and s_6 drive inputs θ_6 and θ_5 respectively. The outputs are exhaustively tested with $2^3 = 8$ patterns. □

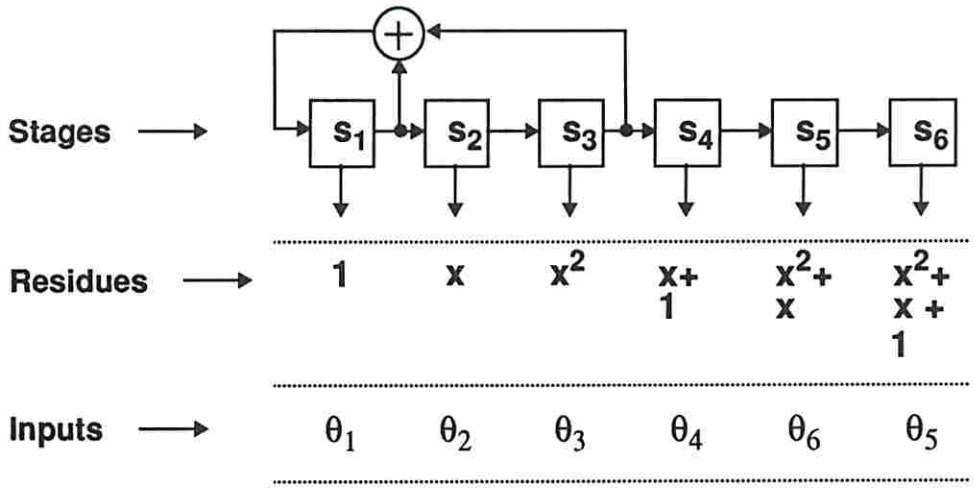


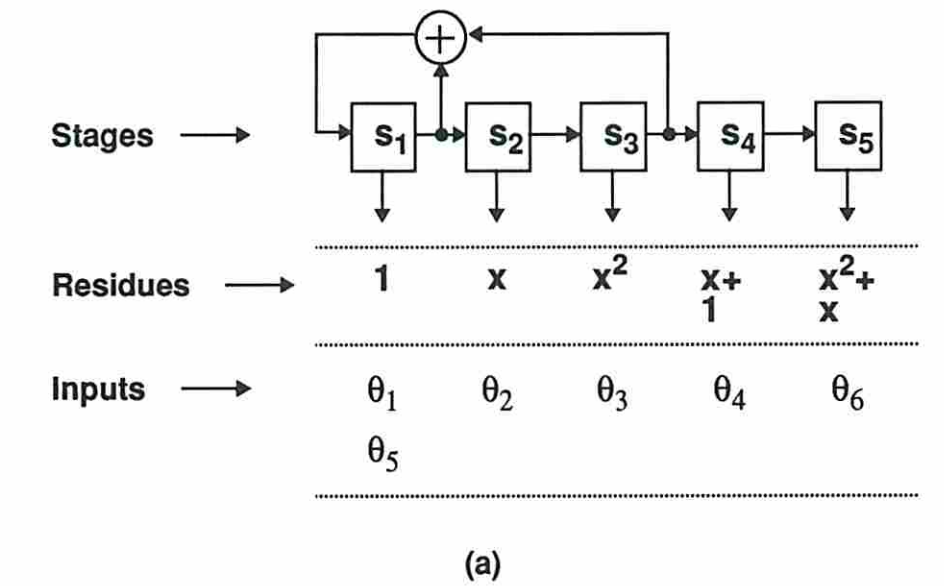
Figure 5.5: Permuted LFSR/SRs

5.2.2.3 Sharing LFSR/SRs

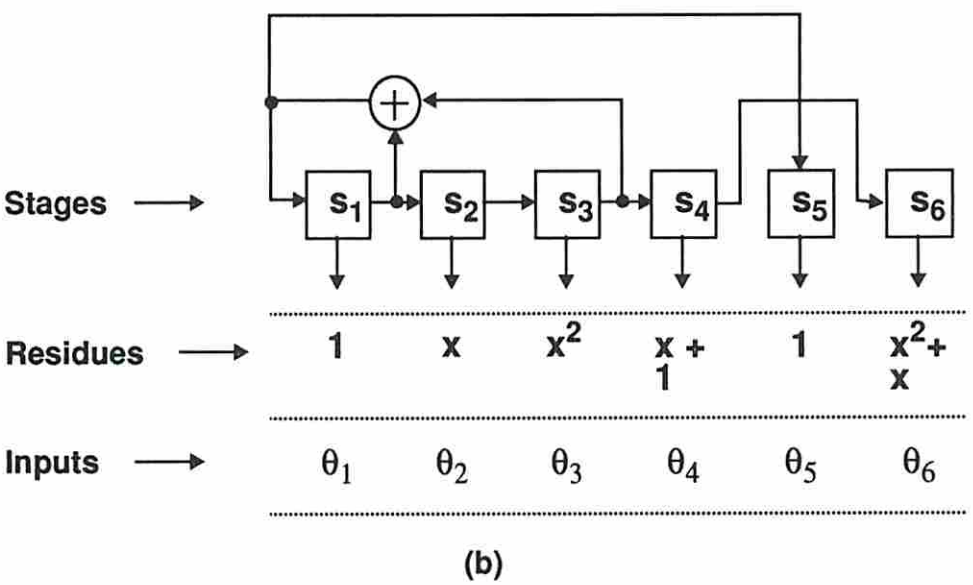
A sharing LFSR/SR [43] allows sharing of test signals among different inputs. In other words, a residue can be assigned to more than one input. Unrelated inputs are assigned the same residues and identical test signals are applied to them.

Example 10 For the $(6, 5, 3)$ circuit shown in Figure 5.2, a possible residue assignment for the inputs (allowing sharing of residues) is shown in the Figure 5.6(a). Inputs θ_1 and θ_5 are unrelated and hence share the residue r_1. The single LFSR/SR shown in Figure 5.6(a) has only five stages and is modified to a $(3, 6)$ sharing LFSR/SR shown in Figure 5.6(b). Stages s_1 and s_5 generate identical test signals. The $(3, 6)$ sharing LFSR/SR can exhaustively test all outputs with $2^3 = 8$ patterns. □

The operations discussed above enhance the capabilities of single LFSR/SRs only by a limited extent. In the next section we shall describe a new TPG design, called *convolved LFSR/SR*, which is a powerful extension to single LFSR/SR. For simplicity of presentation, we shall discuss convolved LFSR/SRs without reconfiguration of feedbacks, permutation of stages and sharing of test signals. Nevertheless convolved LFSR/SRs can also be constructed allowing these design operations.



(a)



(b)

Figure 5.6: Sharing LFSR/SRs

5.3 Convolved LFSR/SRs

A convolved LFSR/SR is derived from a single LFSR/SR design. Circuit inputs are sequentially assigned residues generated by the successive stages of a single LFSR/SR. During the assignment process, it may not be possible to assign a residue to an input due to linear dependencies for some output. Stages whose residues give rise to the problem of linear dependencies are skipped as shown in Figure 5.7(a). This single LFSR/SR ensures linear independence for all outputs but has more stages than the input register. The extra stages required by the single LFSR/SR can be avoided by using XOR gates as shown in Figure 5.7(b). The resulting structure is referred to as *convolved LFSR/SR.*

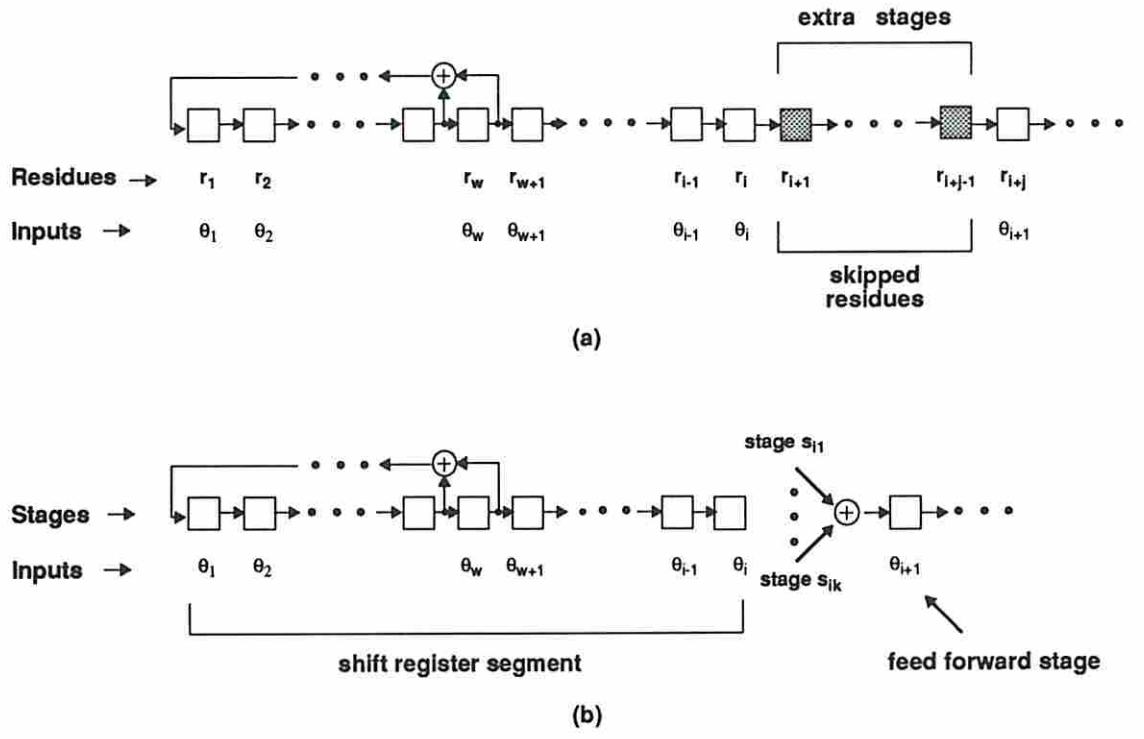
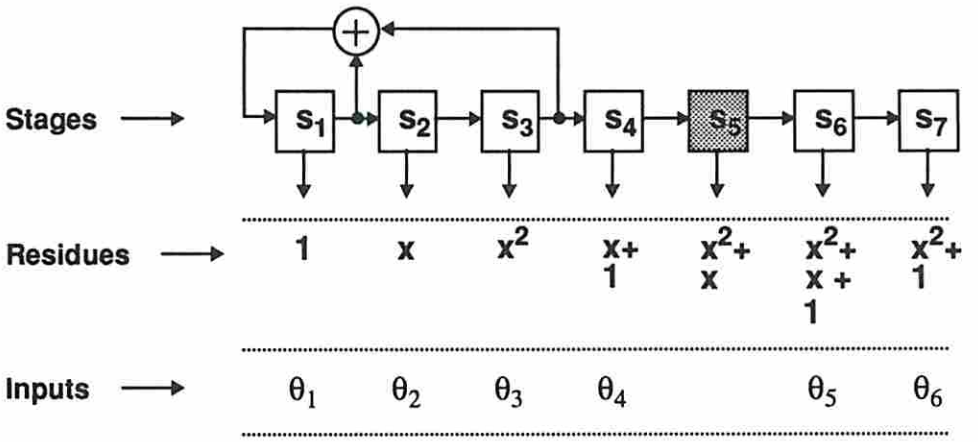


Figure 5.7: Convolved LFSR/SR: (a) Residue assignment for inputs; (b) TPG stages.

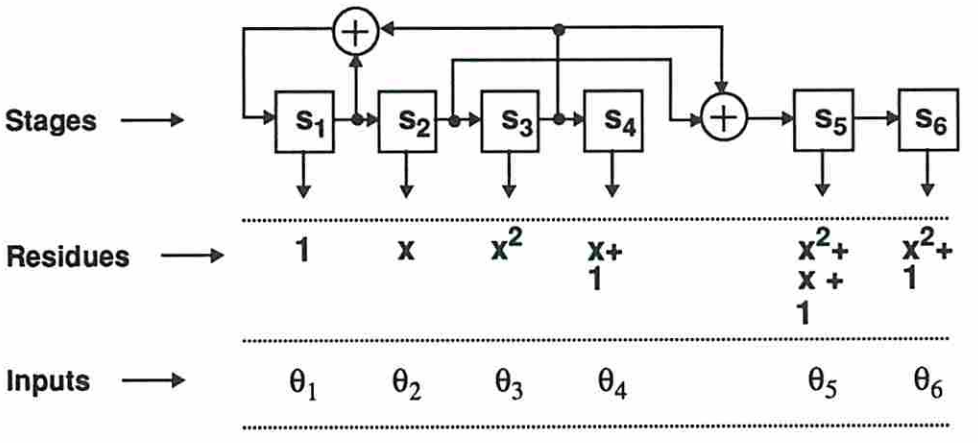
A (w, n) convolved LFSR/SR for an (n, m, k) circuit can be designed as follows. Residues generated by a single LFSR/SR of degree w are considered for assignment to circuit inputs. The inputs are assigned residues one at a time avoiding linear

dependencies with already assigned residues. Let the inputs θ_1 through θ_i be assigned the residues r_1 through r_i respectively. Stages s_1 through s_i of the convolved LFSR/SR are identical to the single LFSR/SR. Assume that input θ_{i+1} cannot be assigned any of the residues $r_{i+1}, r_{i+2}, \ldots, r_{i+j-1}$ because all of them are linearly dependent on already assigned residues for some output. Residue r_{i+j} is then selected for assignment to the input θ_{i+1} as shown in Figure 5.7(a). The single LFSR/SR requires $(j-1)$ extra stages (shown as shaded stages in the figure) whose residues are not assigned to inputs. These extra stages are avoided in the convolved LFSR/SR design. Stage s_{i+1} of the convolved LFSR/SR is made to generate the residue r_{i+j} by feeding the residue r_{i+j-1} at its input. Let the residue r_{i+j-1} be a linear combination of the residues $r_{i_1}, r_{i_2}, \ldots r_{i_k}$. Stages $s_{i_1}, s_{i_2}, \ldots, s_{i_k}$ generate the residues $r_{i_1}, r_{i_2}, \ldots, r_{i_k}$ respectively. These residues are combined using XOR gates and fed at the input of stage s_{i+1} as shown in Figure 5.7(b). The stage s_{i+1} is referred to as *feed forward stage* and the maximal set of contiguous stages (stages s_1 through s_i) is referred to as *shift register segment.* The assignment process is continued until all the inputs are assigned residues such that the residue sets for all output cones are linearly independent. The stages between the feed forward stages form shift register segments. The area overhead for the convolved LFSR/SR design is given by the number and size of XOR gates used to realize the individual feed forward stages.

Example 11 Let us design a $(3,6)$ convolved LFSR/SR for the example circuit in Figure 5.2. Consider the residues from a single LFSR/SR based on $P_1(x)$: (x^3+x+1). Residues r_1 through r_4 are assigned to inputs θ_1 through θ_4 without any linear dependence problem. Residue r_5 cannot be assigned to input θ_5 since the residue set $\{r_2, r_3, r_5\} = \{x, x^2, x^2+x\}$ for output O_3 is linearly dependent. Skipping residue r_5, inputs θ_5 and θ_6 are assigned residues r_6 and r_7 respectively as shown in Figure 5.8(a). This assignment ensures linear independence of the residue sets for all five outputs. The extra stage required by the single LFSR/SR can be avoided using a two-input XOR gate for the convolved LFSR/SR. Stage s_5 of the convolved LFSR/SR can generate the residue r_6 by feeding r_5 at its input. Residue r_5 is obtained by combining the residues r_2 and r_3. Hence the linear combination of the outputs of stages s_2 and s_3 is fed as input to stage s_5 as shown in Figure 5.8(b). The convolved LFSR/SR can exhaustively test all outputs with $2^3 = 8$ patterns. □



(a)



(b)

Figure 5.8: Convolved LFSR/SR: (a) Residue assignment for inputs; (b) TPG stages.

Theorem 8 *A (w,n) convolved LFSR/SR exists for generating pseudo-exhaustive tests for an (n,m,k) circuit if and only if there exists a (w,n) LFSR/XOR for the circuit.*

Proof : *(If)* Consider any (w,n) LFSR/XOR designed for the (n,m,k) circuit. The LFSR/XOR can be transformed to an equivalent (w,n) convolved LFSR/SR as follows. The residues for the test signals generated by the LFSR/XOR stages are determined from the XOR network. These residues can be generated from the corresponding stages of the convolved LFSR/SR by making some stages as feed forward stages if necessary. In constructing the convolved LFSR/SR, design operations such as sharing of residues and permutation of inputs may be required.
(Only if) It is sufficient to show that any convolved LFSR/SR can be transformed to an equivalent LFSR/XOR. The residues of the convolved LFSR/SR stages are determined from the TPG structure. The XOR network for the LFSR/XOR can be designed such that the corresponding stages of the LFSR/XOR generate the assigned residues. □

Convolved LFSR/SRs have great potential to generate minimal test sets. They *bridge the gap* between LFSR/SRs and LFSR/XORs. A trivial convolved LFSR/SR is one having no XOR gates and is simply an LFSR/SR. On the other extreme, any stage of a convolved LFSR/SR can be made as a feed forward stage to generate any desired residue using XOR gates similar to LFSR/XOR. Typically convolved LFSR/SRs achieve low test lengths like LFSR/XORs and utilize low area overhead like LFSR/SRs. The linear independence for the outputs are assured by adding XOR gates to stages whenever necessary. However, most of the stages form shift register segments thereby avoiding high area overhead.

5.4 Multiple LFSR/SRs

A multiple LFSR/SR forms a special case of convolved LFSR/SR. It is composed of two or more independent single LFSR/SRs that are run in parallel. The single LFSR/SRs have identical feedback polynomials but may have different shift register lengths and initial seeds.

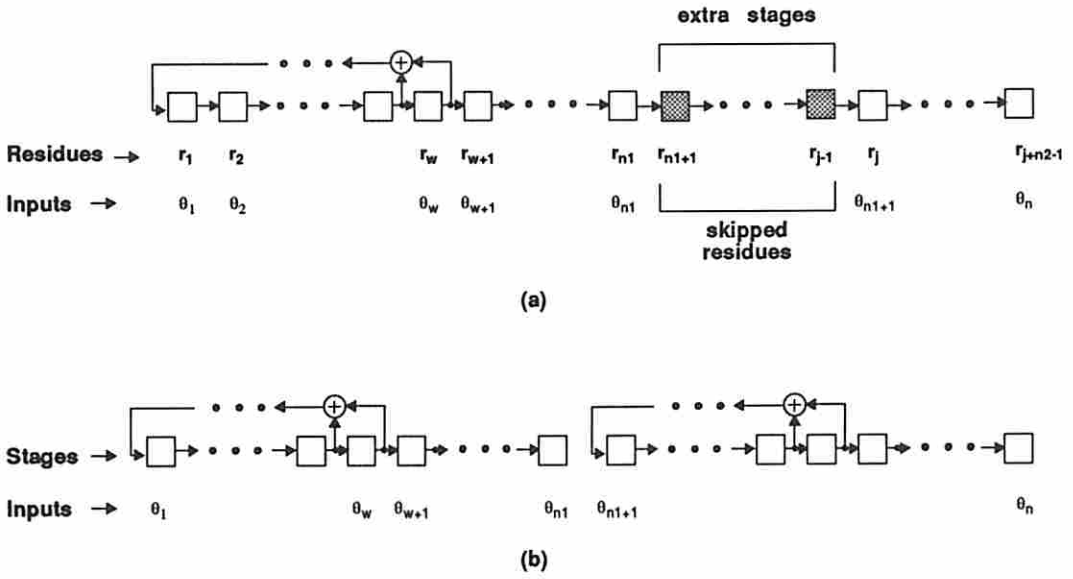


Figure 5.9: Multiple LFSR/SR: (a) Residue assignment for inputs; (b) TPG stages.

A multiple LFSR/SR is derived from a single LFSR/SR. A multiple LFSR/SR for an (n, m, k) circuit can be determined as follows. Consider the residue assignment for the inputs as shown in Figure 5.9(a) ensuring linear independence for all output cones. Let both n_1 and n_2 be greater than or equal to w and the sum of n_1 and n_2 be n. Residues r_1 through r_{n_1} are assigned to inputs θ_1 through θ_{n_1} respectively. Residues r_{n_1+1} through r_{j-1} are skipped and not assigned to any input. Residues r_j through r_{j+n_2-1} are assigned to inputs θ_{n_1+1} through θ_n respectively. The single LFSR/SR requires $(j - n_1 - 1)$ extra stages (shown as shaded stages in the figure) whose residues are not assigned to inputs. These extra stages are avoided in the multiple LFSR/SR design. Stages s_1 through s_{n_1} of the input register are modified to an (w, n_1) single LFSR/SR. Stages s_{n_1+1} through s_n of the input register are modified to an (w, n_2) single LFSR/SR. A (w, n) multiple LFSR/SR composed of (w, n_1) and (w, n_2) single LFSR/SRs is shown in Figure 5.9(b). Both single LFSR/SRs have identical LFSRs of degree w.

A residue assigned to an input can be expressed as a linear combination of the residues $r_1, r_2, \ldots, r_w$. The residues $r_1, r_2, \ldots, r_w$ are generated by the LFSR stages of the first single LFSR/SR. Let an initial seed S be applied to the LFSR portion

of the first single LFSR/SR. From this seed, the initial contents for the rest of the stages of the multiple LFSR/SR can be determined. For example, if an input θ_i is assigned a residue which is a linear combination of the residues $r_{i_1}, r_{i_2}, \ldots, r_{i_k}$ (where $i_1, i_2, \ldots, i_k < w$), then the initial content of stage s_i is a linear combination of the initial contents of the stages $s_{i_1}, s_{i_2}, \ldots, s_{i_k}$. This initialization ensures that stage s_i of the multiple LFSR/SR generates the residue assigned to input θ_i.

Example 12 A multiple LFSR/SR for the $(6, 5, 3)$ circuit shown in Figure 5.2 can be designed as follows. The residue assignment for the inputs shown in Figure 5.10(a) satisfy the linear independence of residue sets for all outputs. We can construct a multiple LFSR/SR composed of two $(3, 3)$ single LFSR/SRs as shown in Figure 5.10(b). Both single LFSR/SRs are based on the same primitive polynomial $x^3 + x + 1$. The stages of the first LFSR/SR generates residues r_1, r_2 and r_3. Let an initial seed $S_1 = 100$ be applied to the first LFSR/SR. The initial seed S_2 for the second LFSR/SR is determined such that its stages generate the residues r_5, r_6 and r_7. Input θ_4 is assigned the residue r_5 which is the linear combination of the residues r_2 and r_3. Hence the initial content of stage s_4 is zero which is a linear combination of the initial contents of the stages s_2 and s_3. Thus the initial seed of the second LFSR/SR is computed as $S2 = 011$. The stages of the multiple LFSR/SR generate the assigned residues shown in Figure 5.10(a). This multiple LFSR/SR generates $2^3 = 8$ patterns to exhaustively test all the five outputs. □

The following lemma characterizes the relation between multiple LFSR/SRs and convolved LFSR/SRs.

Theorem 9 *A (w, n) multiple LFSR/SR exists for generating pseudo-exhaustive tests for an (n, m, k) circuit if and only if there exists a (w, n) convolved LFSR/SR for the circuit where the length of each shift register segment is at least w.*

Proof : *(If)* For the (w, n) convolved LFSR/SR, since the shift register segments are of length at least w, each one of them can be modified as an independent single LFSR/SR with the same feedback polynomial as that of the convolved LFSR/SR. The initial seeds for the single LFSR/SRs are the same as the initial seeds of the shift register segments of the convolved LFSR/SR.

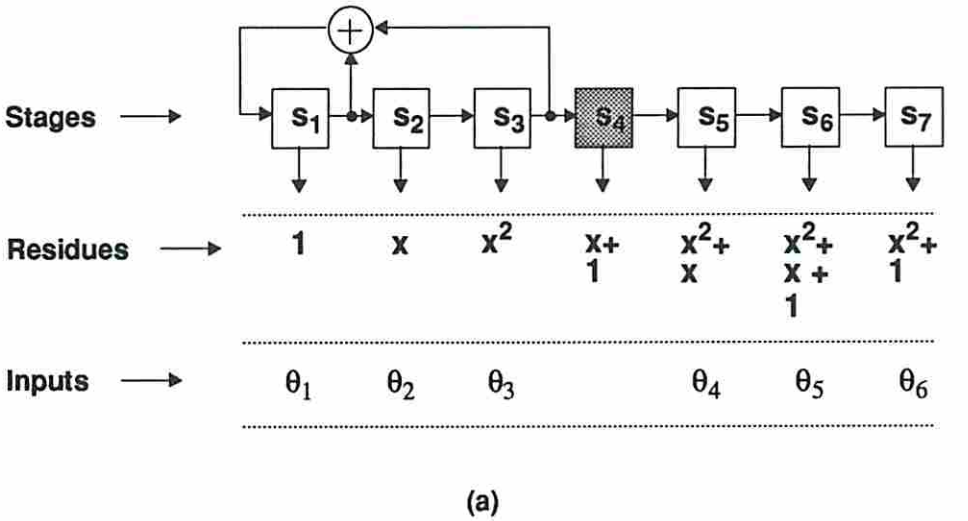


Figure 5.10: Multiple LFSR/SR: (a) Residue assignment for inputs; (b) TPG stages.

(Only if) It is sufficient to show that any multiple LFSR/SR can be transformed to an equivalent convolved LFSR/SR. The independent single LFSR/SRs of the (w, n) multiple LFSR/SR can be modified to form shift register segments of a (w, n) convolved LFSR/SR. The initial seed of the multiple LFSR/SR determines the residues generated by the individual stages. For the convolved LFSR/SR, the feed forward stages are made to generate their assigned residues by using XOR networks. The remaining stages of the convolved LFSR/SR automatically generates their respective assigned residues. The resulting convolved LFSR/SR has each shift register segment having at least w stages. □

Multiple LFSR/SRs utilize XOR network for realizing the feedback polynomials of the individual single LFSR/SRs. On the contrary, convolved LFSR/SRs utilize XOR network for realizing the feed forward stages. The hardware overhead due to XOR network is the deciding factor for selecting either a multiple LFSR/SR or a convolved LFSR/SR. It is usually hard to construct a multiple LFSR/SR that generates an identical size test set as that of a convolved LFSR/SR due to the restriction on the length of the shift register segments.

5.5 Design Procedure

For an (n, m, k) circuit, a (w, n) convolved LFSR/SR can be designed as follows. A primitive polynomial $P(x)$ of degree w is selected as the feedback polynomial. The polynomial can generate $(2^w - 1)$ unique residues from r_1 through r_{2^w-1} and all of them can be considered for assignment to the inputs. The number of possible residues is exponential with the degree of the polynomial. Practically only a few residues, say r_1 through r_N, are considered for input assignment. This practical consideration limits the convolved LFSR/SR designs. The shift register segments are constrained to have a desired minimum length, say l. This constraint attempts to reduce the number of feed forward stages and hence the area overhead due to XOR network. A (w, n) multiple LFSR/SR can be obtained by restricting $l \geq w$ and a (w, n) single LFSR/SR can be obtained by restricting $l = n$.

Example 13 Consider the set $S = \{0, 1, x, 1+x, x^2, 1+x^2, x+x^2, 1+x+x^2\}$. The set S is closed under modulo-2 addition and hence forms an Abelian group and a vector space. The set $B = \{1, x, x^2\}$ consists of linearly independent elements and $L(B) = S$. Thus B forms a basis of S and the dimension of S equals $|B| = 3$.

Consider two distinct subspaces $S_1 = \{0, 1, x, 1+x\}$ and $S_2 = \{0, 1, x^2, 1+x^2\}$ contained in S. The direct sum operation between S_1 and S_2, $S_1 \oplus S_2 = \{0, 1, x, 1+x, x^2, 1+x^2, x+x^2, 1+x+x^2\} = S$. The set union operation between S_1 and S_2 is $S_1 \cup S_2 = \{0, 1, x, 1+x, x^2, 1+x^2\} \neq S$, and set intersection operation between S_1 and S_2 is $S_1 \cap S_2 = \{0, 1\}$. □

We have derived a few algebraic results regarding set union and intersection operations between subspaces and these results differ from the classical results. The following results characterize a few properties of subspaces contained in a vector space.

Theorem 10 *Consider a k-dimensional space S and any two distinct subspaces S_1 and S_2 of dimensions k_1 and k_2 contained in S. The set $S_1 \cap S_2$ is a subspace contained in S and consists of at least $\lceil 2^{k_1+k_2-k} \rceil$ elements.*

Proof : Let $S_3 = S_1 \cap S_2$. Consider any two elements a and b such that $a, b \in S_3$. Since $S_3 \subset S_1$ and $S_3 \subset S_2$, we have $a, b \in S_1$ and $a, b \in S_2$. Since S_1 and S_2 are subspaces, we have that $(a+b) \in S_1$ and $(a+b) \in S_2$ implies $(a+b) \in S_3$. Thus S_3 forms an Abelian group and is a subspace contained in S. Let x be the dimension of subspace S_3.

Let S_1, S_2 and S_3 be spanned by the bases B_1, B_2 and B_3 respectively. Since $S_3 \subset S_1$ and $S_3 \subset S_2$, we can choose B_1 and B_2 such that $B_3 \subset B_1$ and $B_3 \subset B_2$. Since $|B_1| = k_1$, $|B_2| = k_2$ and $|B_3| = x$, we have

$$|B_1 \cup B_2| \;=\; |B_1| + |B_2| - |B_1 \cap B_2| \;=\; |B_1| + |B_2| - |B_3| \;=\; k_1 + k_2 - x$$

Let S_4 be the $(k_1 + k_2 - x)$-dimensional subspace spanned by the basis $B_1 \cup B_2$. Since $S_4 \subseteq S$ we have

$$k_1 + k_2 - x \;\leq\; k \;\Rightarrow\; x \geq k_1 + k_2 - k$$

Hence $S_1 \cap S_2$ is a subspace of dimension at least $(k_1 + k_2 - k)$. Although the term $(k_1 + k_2 - k)$ could be negative, $S_1 \cap S_2$ always contains the additive identity element (zero). Hence $S_1 \cap S_2$ is a subspace contained in S and has at least $\lceil 2^{k_1+k_2-k} \rceil$ elements. □

Corollary 2 *Consider a k-dimensional space S and any two distinct $(k-1)$- dimensional subspaces S_1 and S_2 contained in S. The set $S_1 \cap S_2$ is a $(k-2)$-dimensional subspace contained in S.*

Theorem 11 *Consider a k-dimensional space S and any three distinct $(k-1)$-dimensional subspaces S_1, S_2 and S_3 contained in S. Let $S_4 = S_1 \cap S_2$. The subspace S_3 satisfies the relation $S_1 \cup S_2 \cup S_3 = S$ if and only if $S_1 \cap S_2 \cap S_3 = S_4$.*

Proof : Since S_1 and S_2 are distinct $(k-1)$-dimensional subspaces contained in S, S_4 is a $(k-2)$-dimensional subspace as per Corollary 2. Consider an element a such that $a \in S_1$ and $a \notin S_4$. Let $T_1 = \{a + s \mid \forall s \in S_4\}$. Then we have $T_1 \subset S_1$ and $|T_1| = |S_4| = 2^{k-2}$. The set $S_4 \cap T_1 = \emptyset$ since $a \notin S_4$. The set $S_4 \cup T_1$ contains 2^{k-1} elements and hence $S_4 \cup T_1 = S_1$. Consider another element b such that $b \in S_2$ and $b \notin S_4$. Let $T_2 = \{b + s \mid \forall s \in S_4\}$. Then we have $T_2 \subset S_2$ and $|T_2| = |S_4| = 2^{k-2}$. The set $S_4 \cap T_2 = \emptyset$ since $b \notin S_4$. The set $S_4 \cup T_2$ contains 2^{k-1} elements and hence $S_4 \cup T_2 = S_2$. The Venn diagram of these sets are shown in Figure 6.1. Thus we have

$$
\begin{aligned}
S_1 &= S_4 \cup \{a + s \mid \forall s \in S_4\} = S_4 \cup T_1 \\
S_2 &= S_4 \cup \{b + s \mid \forall s \in S_4\} = S_4 \cup T_2 \\
S_1 \cup S_2 &= S_4 \cup \{a + s \mid \forall s \in S_4\} \cup \{b + s \mid \forall s \in S_4\} = S_4 \cup T_1 \cup T_2
\end{aligned}
$$

Let $T_3 = \{a + b + s \mid \forall s \in S_4\}$. Since $a, b \notin S_4$, we know that $a \notin S_2$, $b \notin S_1$ and $a + b \notin S_1 \cup S_2$. Therefore $T_3 \cap S_1 = T_3 \cap S_2 = \emptyset$. We know that $T_3 \subset S$ and $|T_3| = |S_4| = 2^{k-2}$. The sets S_4, T_1, T_2 and T_3 are disjoint to each other and the set $S_4 \cup T_1 \cup T_2 \cup T_3$ contains 2^k elements and hence $S_4 \cup T_1 \cup T_2 \cup T_3 = S$. The elements of S are partitioned into four equal sized subsets S_4, T_1, T_2 and T_3 (the subsets are called cosets in algebraic terminology [18]) as shown in Figure 6.1.

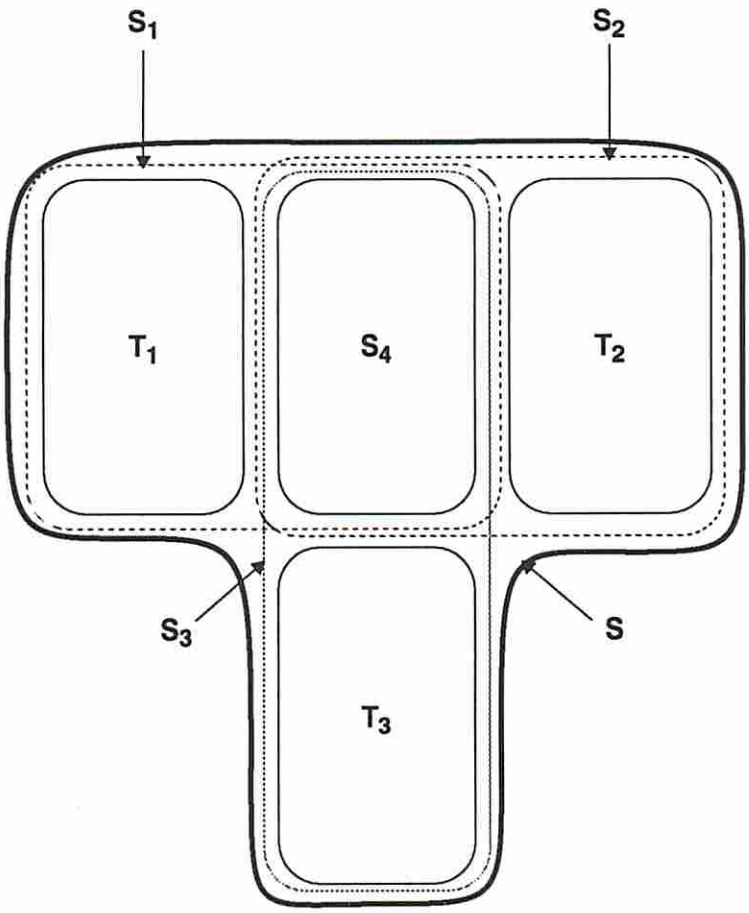


Figure 6.1: A vector space and its subspaces and cosets

The set T_i $(i = 1, 2, 3)$ does not form a subspace and $S_4 \subset L(T_i)$. If a subspace (say S_x) contains S_4 and an element from T_i, then $T_i \subset S_x$. The subsets S_4, T_1, T_2 and T_3 are unique to any given two subspaces S_1 and S_2.
(If):: Assume that $S_1 \cap S_2 \cap S_3 = S_4$. The set $S_3 \cap T_1 = \emptyset$ since if $S_3 \cap T_1 \neq \emptyset$, then $T_1 \subset S_3$ and $S_3 = S_1$. Similarly the set $S_3 \cap T_2 = \emptyset$ since if $S_3 \cap T_2 \neq \emptyset$, then $T_2 \subset S_3$ and $S_3 = S_2$. Hence $S_3 \cap T_3 \neq \emptyset$. Since $S_4 \subset S_3$ and $T_3 \cap S_3 \neq \emptyset$, we have $T_3 \subset S_3$ and $S_3 = S_4 \cup T_3$. Therefore $S_1 \cup S_2 \cup S_3 = S_4 \cup T_1 \cup T_2 \cup T_3 = S$.
(Only If):: Assume that $S_1 \cup S_2 \cup S_3 = S$. This implies $T_3 \subset S_3$. Since S_3 is a subspace, $S_4 \subset L(T_3) \subseteq S_3$. Therefore $S_1 \cap S_2 \cap S_3 = S_4$. □

Theorem 12 *A k-dimensional space is composed of at least $(2^i + 1)$ distinct subspaces of dimensions less than or equal to $(k - i)$, where $1 \leq i \leq (k - 1)$.*

Proof : We shall prove the theorem in two parts. First we will show that a k-dimensional space is composed of at least $(2^i + 1)$ distinct $(k - i)$-dimensional subspaces. Then we will generalize the dimensions of the $(2^i + 1)$ distinct subspaces to less than or equal to $(k - i)$.
Part I: Consider any two distinct $(k-i)$-dimensional subspaces S_1 and S_2 contained in a k-dimensional space S. From Theorem 10, the two subspaces S_1 and S_2 must have a common subspace of dimension at least $(k - 2i)$. Hence we have

$$\begin{aligned} |S_1| &= |S_2| = 2^{k-i} \\ |S_1 \cap S_2| &\geq 2^{k-2i} \\ |S_1 \cup S_2| &= |S_1| + |S_2| - |S_1 \cap S_2| \leq 2^{k-i} + 2^{k-i} - 2^{k-2i}. \end{aligned}$$

Let $S_1, S_2, \ldots, S_x$ be x distinct $(k-i)$-dimensional subspaces such that $\cup_{j=1}^{x} S_j = S$. Each of these subspaces can have at most $(2^{k-i} - 2^{k-2i})$ elements unique to them. Hence we have

$$|S| = \left| \bigcup_{j=1}^{x} S_j \right| = 2^k \leq 2^{k-i} + (x-1)(2^{k-i} - 2^{k-2i})$$

Multiplying throughout by 2^{2i-k} we get

$$2^{2i} \leq 2^i + (x-1)(2^i - 1)$$

$$\Rightarrow \quad x \geq \frac{2^{2i} - 2^i}{2^i - 1} + 1$$
$$\Rightarrow \quad x \geq 2^i + 1.$$

Thus a k-dimensional space is composed of at least (2^i+1) distinct $(k-i)$-dimensional subspaces.

Part II: Now we shall generalize the dimensions of the $(2^i + 1)$ distinct subspaces. Let $S_0^*, S_1^*, \ldots, S_{2^i}^*$ be $(2^i + 1)$ distinct subspaces contained in a k-dimensional space S with dimensions $k_0, k_1, \ldots, k_{2^i}$ respectively. Let $k_j \leq (k - i) \quad \forall\, j = 0, 1, \ldots, 2^i$. Let $S_0, S_1, \ldots, S_{2^i}$ be $(2^i + 1)$ distinct $(k - i)$-dimensional subspaces contained in S such that $S_j^* \subseteq S_j \quad \forall\, j = 0, 1, \ldots, 2^i$. From Part I we know that

$$\bigcup_{j=0}^{2^i} S_j \subseteq S.$$
$$\Rightarrow \bigcup_{j=0}^{2^i} S_j^* \subseteq \bigcup_{j=0}^{2^i} S_j \subseteq S.$$

Thus a k-dimensional space is composed of at least $(2^i + 1)$ distinct subspaces of dimensions less than or equal to $(k - i)$. □

Theorem 10 gives a condition on the minimum overlap between any two subspaces contained in a k-dimensional space. Theorem 11 states that the elements of a k-dimensional space S are not entirely covered by the elements of any two distinct $(k-1)$-dimensional subspaces contained in S. A unique third subspace of dimension $(k-1)$ is required to cover all the elements of S. Theorem 12 specifies the minimum number of distinct subspaces of smaller dimensions contained in a k-dimensional space. In fact Theorem 11 provides an outline for constructing these minimum number of distinct subspaces.

6.3 Cone Independent Bounds

For an (n, m, k) circuit, the computation of an upper bound on the pseudo-exhaustive test length involves determining the smallest number of independent test signals (say $k^* \geq k$) that are sufficient for pseudo-exhaustive testing of the circuit. We shall derive a few important cone independent results on the bounds on test lengths. In

other words, these results do not utilize the information about the specific output cone dependencies of the circuit.

A set of 2^{k^*} distinct test signals can be obtained as linear combinations of k^* independent test signals. The distinct test signals are considered as distinct *residues*. The k^* independent test signals can be considered as a basis of a k^*-dimensional space and the residues can be considered as elements of this space. The k^* independent test signals can be generated using a k^* degree LFSR and linear combinations of these test signals can be obtained by an XOR network.

Definition 4 *A residue r is said to be a* **proper residue** *with respect to a set of residues R if r is linearly independent with respect to the residues in R. Residue r is said to be a* **prohibited residue** *with respect to R if r is a linear combination of a subset of residues in R.*

Theorem 13 (Barzilai83) *An output cone will be exhaustively tested if and only if the inputs driving the output cone are assigned proper residues.*

For an (n, m, k) circuit we need to assign proper residues to the circuit inputs from a k^*-dimensional space (where $k^* \geq k$) such that the residues assigned to the inputs driving any output cone are linearly independent. The bound computation involves ensuring the availability of proper residues (elements) to all circuit inputs from the k^*-dimensional space.

Let us consider an (n, m, k) circuit along with the following notation. The n inputs are denoted as θ_i, $i = 1, 2, \ldots, n$, and the m outputs are denoted as O_j, $j = 1, 2, \ldots, m$, respectively. The inputs are partitioned into m sets $I_1, I_2, \ldots, I_m$ such that I_i denotes the set of inputs that drive exactly i outputs in the circuit. During the residue assignment process, inputs in I_i are considered prior to inputs in I_{i-1}.

Definition 5 *Output O_i is said to* **dominate** *output O_j if each input that drives O_j also drives O_i.*

Lemma 12 *It is sufficient to consider dominating circuit outputs for determining pseudo-exhaustive test lengths.*

Proof : Let an output O_i dominate another output O_j in a circuit. Proper residue assignment to the set of inputs driving O_i ensures exhaustive testing for both output cones O_i and O_j. Hence there is no need to consider residue assignments separately for O_j. □

Definition 6 *A circuit is said to be* **reduced** *if none of its outputs is dominated by any other output.*

Lemma 12 states that dominated outputs are guaranteed to be exhaustively tested provided the dominating outputs are guaranteed to be exhaustively tested. Any given circuit can be reduced by ignoring all its dominated outputs. Henceforth we shall consider only reduced circuits.

Lemma 13 *For an (n, m, k) circuit let k^* ($\geq k$) independent test signals be sufficient to assign proper residues for all inputs in I_i for all $i > 2^{k^*-k+1}$. Then these test signals are also sufficient to assign proper residues for all inputs in I_j for all $j \leq 2^{k^*-k+1}$.*

Proof : Let S be the k^*-dimensional space generated by k^* independent test signals. Assume that all inputs in I_i for all $i > 2^{k^*-k+1}$ have been assigned proper residues from S. Let input $\theta \in I_{2^{k^*-k+1}}$ drive output O_j. Assume that k_j inputs driving O_j have been already assigned proper residues and the residue assignment for θ is under consideration (Note that $k_j \leq (k-1)$). The residues assigned to k_j inputs span a k_j-dimensional subspace and none of the elements from this subspace can be assigned as a proper residue for θ. In other words, all the elements in this subspace are prohibited residues for θ. Since θ drives exactly 2^{k^*-k+1} outputs, there are at most 2^{k^*-k+1} distinct subspaces of dimensions less than or equal to $(k-1)$ whose elements are prohibited residues for θ.

Theorem 12 states that S is composed of at least $(2^{k^*-k+1}+1)$ distinct subspaces of dimensions less than or equal to $(k-1)$. Thus the total number of prohibited residues for θ is less than 2^{k^*}. Hence θ can be assigned a proper residue from S. Since θ is arbitrary, all inputs in $I_{2^{k^*-k+1}}$ can be assigned proper residues from S.

Similarly, it can be shown that the total number of prohibited residues is less than 2^{k^*} for any input in I_j for all $j < 2^{k^*-k+1}$. Hence all inputs in I_j for all $j \leq 2^{k^*-k+1}$ can be assigned proper residues by k^* test signals. □

Corollary 3 *For an (n, m, k) circuit, let k independent test signals be sufficient to assign proper residues for all inputs in I_i for all $i > 2$. Then these test signals are also sufficient to assign proper residues for all inputs in I_2 and I_1.*

Definition 7 (McCluskey 84) *An (n, m, k) circuit is said to be a* **maximal test concurrent** *(MTC) circuit, if it can be pseudo-exhaustively tested with k independent test signals.*

Theorem 14 (McCluskey 84) *Any (n, m, k) circuit with $m < 3$ is a MTC circuit.*

Any (n, m, k) circuit needs at least k test signals due to its maximum cone size. If $m < 3$, then the circuit inputs can only be partitioned into I_2 and I_1. Thus Corollary 3 directly leads to Theorem 14 inferred from [27].

Lemma 14 *For an (n, m, k) circuit with $m < 6$, let k independent test signals be sufficient to assign proper residues for all inputs in I_5 and I_4. Then these test signals are also sufficient to assign proper residues for all inputs in I_3.*

Proof : Let S be the k-dimensional space spanned by the k independent test signals. Assume that all inputs in I_5 and I_4 have been assigned proper residues from S. We shall show that all inputs in I_3 can also be assigned proper residues from S for any $(n, 5, k)$ circuit. The lemma follows for any (n, m, k) circuit with $m < 6$.

Let the five outputs of the circuit be denoted as O_1, O_2, O_3, O_4 and O_5 respectively. Let us sequentially assign proper residues to inputs in I_3 and assume that input $\theta \in I_3$ is under consideration for residue assignment. Let θ drive outputs O_1, O_2 and O_3. Each of these three outputs can have at most $(k-1)$ inputs that are already assigned proper residues. Let k_1, k_2 and k_3 be the number of inputs driving O_1, O_2 and O_3, respectively, that have already been assigned proper residues. Without loss of generality, assume $k_1 \leq k_2 \leq k_3 \leq (k-1)$. Let S_i $(i = 1, 2, 3)$ be the subspace spanned by the residues assigned to k_i inputs driving O_i. The subspaces S_1, S_2 and S_3 are of dimensions k_1, k_2 and k_3 respectively. The elements in $S_1 \cup S_2 \cup S_3$ are prohibited residues for θ. As per Theorem 10, we have

$$|S_1 \cap S_3| \geq 2^{k_1+k_3-k}$$
$$|S_2 \cap S_3| \geq 2^{k_2+k_3-k}$$

Hence the total number of prohibited residues for θ is given by

$$|S_1 \cup S_2 \cup S_3| \leq 2^{k_1} + 2^{k_2} + 2^{k_3} - 2^{k_1+k_3-k} - 2^{k_2+k_3-k} \leq 2^k \quad (6.1)$$

The equality in Equation 6.1 is satisfied only for $k_1 = k_2 = k_3 = (k-1)$. That means any input θ in I_3 can be assigned a proper residue from S, provided the values of k_1, k_2 and k_3 are not simultaneously equal to $(k-1)$. Since the circuit has only five outputs, there can be at most only one input in I_3 with $k_1 = k_2 = k_3 = (k-1)$. Let θ^* be the unique input in I_3 satisfying the condition $k_1 = k_2 = k_3 = (k-1)$. Therefore all the inputs in I_3 except θ^* can be assigned proper residues from S. Input θ^* appears in O_1, O_2 and O_3 as shown below.

$$\begin{aligned}
O_1 &:: \cdots\cdots \theta_1 \cdots\cdots \theta^* \\
O_2 &:: \cdots\cdots \theta_2 \cdots\cdots \theta^* \\
O_3 &:: \cdots\cdots \theta_3 \cdots\cdots \theta^* \\
O_4 &:: \cdots \theta_1\ \theta_2\ \theta_3 \cdots \\
O_5 &:: \cdots \theta_1\ \theta_2\ \theta_3 \cdots
\end{aligned}$$

Let $T = S_1 \cup S_2 \cup S_3$. Input θ^* can be assigned a proper residue as long as $T \subset S$. Let $T = S$ under a residue assignment for inputs in $I_3 - \{\theta^*\}$ so that θ^* cannot be assigned a proper residue from S. We shall show that there exists another residue assignment for inputs in $I_3 - \{\theta^*\}$ such that $T \subset S$ and θ^* can also be assigned a proper residue from S.

Let R_1, R_2 and R_3 be the sets of $(k-1)$ residues assigned to the remaining $(k-1)$ inputs driving O_1, O_2 and O_3 respectively. The $(k-1)$-dimensional subspaces S_1, S_2 and S_3 are spanned by the sets R_1, R_2 and R_3 respectively. Let $S_4 = S_1 \cap S_2 \cap S_3$. Since $T = S$ by our assumption, S_4 is a $(k-2)$-dimensional subspace as per Theorem 11. Since $T = S$, there exists residues r_1, r_2 and r_3 unique to R_1, R_2 and R_3, respectively, such that $r_1 \notin S_2 \cup S_3$, $r_2 \notin S_1 \cup S_3$ and $r_3 \notin S_1 \cup S_2$. Following similar arguments given in the proof of Theorem 11, we can show that

$$\begin{aligned}
S_1 &= S_4 \cup \{r_1 + s \mid \forall s \in S_4\} \\
S_2 &= S_4 \cup \{r_2 + s \mid \forall s \in S_4\}
\end{aligned}$$

$$S_3 \;=\; S_4 \;\cup\; \{r_3 + s \mid \forall s \in S_4\}$$

Let $T_1 = \{r_1 + s \mid \forall s \in S_4\}$, $T_2 = \{r_2 + s \mid \forall s \in S_4\}$ and $T_3 = \{r_3 + s \mid \forall s \in S_4\}$. Since $T = S$, Theorem 11 implies that the set $\{r_1 + r_2 + s \mid \forall s \in S_4\}$ must be equal to T_3. In other words, r_3 must be equal to $(r_1 + r_2 + s^*)$ where $s^* \in S_4$.

Let inputs θ_1, θ_2 and θ_3 drive outputs O_1, O_2 and O_3 (as shown above) and be assigned the residues r_1, r_2 and r_3 respectively. Since the residues r_1, r_2 and r_3 are unique to R_1, R_2 and R_3, the inputs θ_1, θ_2 and θ_3 are also unique to O_1, O_2 and O_3 respectively. Inputs θ_1, θ_2 and θ_3 cannot belong to I_4 or I_5 and hence must belong to I_3. This implies the last two outputs O_4 and O_5 must be driven by all three inputs θ_1, θ_2 and θ_3 as shown above.

We shall show that the residue $r'_3 = (r_1 + s^*)$ instead of $r_3 = (r_1 + r_2 + s^*)$ is still a proper residue for input θ_3. Input θ_3 drives O_3, O_4 and O_5. Let us consider O_3 and show that r'_3 can also be assigned as a proper residue for θ_3 instead of r_3. Since $r_1 \notin S_3$, we infer that $r'_3 \notin S_3$. Since $L(R_3 - \{r_3\}) \subset S_3$, we know that $r'_3 \notin L(R_3 - \{r_3\})$. Hence r'_3 is linearly independent with the residues in $(R_3 - \{r_3\})$ and r'_3 instead of r_3 can be assigned as a proper residue for θ_3 as far as O_3 is concerned. Next let us consider O_4. Let R_4 be the set of linearly independent residues assigned to the inputs driving O_4. Since the inputs θ_1, θ_2 and θ_3 appear together in O_4, $\{r_1, r_2, r_3\} \subset R_4$. Since $r_3 \notin L(R_4 - \{r_3\})$, $r_2 \in L(R_4 - \{r_3\})$ and $r'_3 = (r_3 + r_2)$, we infer that $r'_3 \notin L(R_4 - \{r_3\})$. Therefore r'_3 instead of r_3 can be assigned as a proper residue for θ_3 as far as O_4 is concerned. Similarly, it can be shown that r'_3 instead of r_3 can be assigned as a proper residue for θ_3 as far as O_5 is concerned. Thus we reassign r'_3 instead of r_3 as a proper residue for θ_3.

Let $R'_3 = R_3 - \{r_3\} + \{r'_3\}$ and $S'_3 = L(R'_3)$. By the reassignment process R'_3 instead of R_3 becomes the set of $(k-1)$ residues assigned to the remaining $(k-1)$ inputs driving O_3. Since $r_2 \notin L(R_3) = S_3$, we know that $r_2 \notin L(R_3 - \{r_3\})$. Since $r_2 \notin L(R_3 - \{r_3\})$, $r_3 \notin L(R_3 - \{r_3\})$ and $r'_3 = r_2 + r_3$, we infer that $r_2 \notin L(R'_3) = S'_3$ and $r_3 \notin L(R'_3) = S'_3$. Since $r_3 \notin S_1 \cup S_2$, we infer $r_3 \notin S_1 \cup S_2 \cup S'_3$ and therefore θ^* can be assigned r_3 as a proper residue. Thus all inputs in I_3 can be assigned proper residues from S. □

Theorem 15 *Any (n, m, k) circuit with $m < 6$ is a MTC circuit.*

Proof : Consider any (n, m, k) circuit with $m < 6$. Since the maximum cone size of the circuit is k, it requires at least k independent test signals for pseudo-exhaustive testing. Let S be the k-dimensional space spanned by the basis $B = \{1, x, x^2, \ldots, x^{k-1}\}$ (representing k independent test signals). We only need to show that all inputs in I_5 and I_4 can be assigned proper residues from S. Inputs in I_3 are guaranteed of proper residues from S as per Lemma 14. Inputs in I_2 and I_1 are guaranteed of proper residues from S as per Corollary 3.

Case $m = 4$*:* Let $|I_4| = k_4$. Since each output is driven by all inputs in I_4 and the maximum cone size for the circuit is k, $k_4 \leq k$. Hence all inputs in I_4 can be assigned proper residues by selecting k_4 elements $\{1, x, x^2, \ldots, x^{k_4-1}\}$ of B. Hence the circuit is a MTC circuit.

Case $m = 5$*:* Let $|I_5| = k_5$ and $|I_4| = k_4$. Since each output is driven by all inputs in I_5 and the maximum cone size for the circuit is k, $k_5 \leq k$. Inputs in I_5 can be assigned proper residues by selecting k_5 elements $\{1, x, x^2, \ldots, x^{k_5-1}\}$ of B. We shall consider inputs in I_4 and assign proper residues from the subspace spanned by the remaining $(k - k_5)$ elements $\{x^{k_5}, x^{k_5+1}, \ldots, x^{k-1}\}$ of B.

Let the five outputs of the circuit be denoted as O_1, O_2, O_3, O_4 and O_5 respectively. Partition the inputs in I_4 into five subsets $I_{4,1}, I_{4,2}, \ldots, I_{4,5}$ such that $I_{4,i} =$ {inputs that do not drive O_i } $(i = 1, 2, \ldots, 5)$. Let $|I_{4,i}| = k_{4,i}$ $(i = 1, 2, \ldots, 5)$. Without loss of generality, assume that $I_{4,5}$ is the smallest subset among the five subsets. Select one input (say θ_i) from each $I_{4,i}$ and form the input set $I = \{\theta_1, \theta_2, \theta_3, \theta_4, \theta_5\}$. Note that only four inputs from I appear together in any output as shown below.

$$
\begin{aligned}
O_1 &:: \cdots\cdots \theta_5\ \theta_4\ \theta_3\ \theta_2 \cdots\cdots \\
O_2 &:: \cdots\cdots \theta_5\ \theta_4\ \theta_3\ \theta_1 \cdots\cdots \\
O_3 &:: \cdots\cdots \theta_5\ \theta_4\ \theta_2\ \theta_1 \cdots\cdots \\
O_4 &:: \cdots\cdots \theta_5\ \theta_3\ \theta_2\ \theta_1 \cdots\cdots \\
O_5 &:: \cdots\cdots \theta_4\ \theta_3\ \theta_2\ \theta_1 \cdots\cdots
\end{aligned}
$$

The inputs in I completely occupy four columns in the cone dependencies. Consider a four dimensional subspace spanned by the four elements $\{x^{k_5}, x^{k_5+1}, x^{k_5+2}, x^{k_5+3}\}$

of B. We shall assign the five residues $\{x^{k_5}, x^{k_5+1}, x^{k_5+2}, x^{k_5+3}, x^{k_5} + x^{k_5+1} + x^{k_5+2} + x^{k_5+3}\}$ to the five inputs in I. Since only any four inputs from I appear together in any output, this assignment ensures proper residues to all inputs in I. This process is repeated until all inputs are selected from $I_{4,5}$. Thus $5k_{4,5}$ inputs in I_4 are assigned proper residues from the subspace spanned by $4k_{4,5}$ elements of B.

The remaining $(k_4 - 5k_{4,5})$ inputs in I_4 need to be assigned proper residues from the subspace spanned by the remaining $(k - k_5 - 4k_{4,5})$ elements in B. Since none of the remaining inputs in I_4 belong to $I_{4,5}$, all of them drive O_5. Also all k_5 inputs in I_5 and $4k_{4,5}$ inputs in I_4 drive O_5. Hence the total number of inputs driving O_5 must be greater than or equal to $(k_5 + 4k_{4,5} + k_4 - 5k_{4,5}) = (k_5 + k_4 - k_{4,5})$. Since the maximum cone size for the circuit is k, we have $k \geq k_5 + k_4 - k_{4,5}$ which implies $k - k_5 - 4k_{4,5} \geq k_4 - 5k_{4,5}$. Hence we have the number of remaining elements in B is greater than or equal to the number of remaining inputs in I_4 and we can assign each of the remaining elements in B to each of the remaining inputs in I_4. Thus all inputs in I_5 and I_4 can be assigned proper residues from S. Hence the circuit is a MTC circuit. □

Theorem 15 states that any five output circuit is a MTC circuit. The result is independent of the number of inputs and the maximum cone sizes of the circuits. Our result is a significant improvement over the well known result that any two output circuit is a MTC circuit (Theorem 14).

Example 14 illustrates a six output non-MTC circuit. In the example, note that even though all inputs drive exactly three outputs, the circuit is not a MTC circuit since it contains six outputs. The example illustrates the tightness of the results stated in Lemma 14 and Theorem 15.

Example 14 Consider an $(4, 6, 2)$ circuit driven by inputs $\{\theta_1, \theta_2, \theta_3, \theta_4\}$. The circuit has the following input dependencies for the six output cones: $\{(\theta_1, \theta_2), (\theta_1, \theta_3), (\theta_2, \theta_3), (\theta_1, \theta_4), (\theta_2, \theta_4), (\theta_3, \theta_4)\}$. The circuit is not a MTC circuit and needs three independent test signals (say $1, x, x^2$). Inputs θ_1 through θ_4 can be assigned residues $1, x, 1 + x$ and x^2 respectively. □

Theorem 16 *For any $(n, m, 2)$ circuit, let $k^* \geq 2$ be the smallest number satisfying at least one the following inequalities*

$$\begin{aligned} n &< 2^{k^*} \\ m &< (2^{k^*-1})(2^{k^*} - 1) \end{aligned}$$

Then k^* independent test signals are sufficient for pseudo-exhaustive testing of the circuit.

Proof : Since the maximum cone size is two for the circuit, every output is driven by at most two inputs. A pair of inputs that drive an output must be assigned two distinct residues for exhaustive testing of that output.
Case $n < 2^{k^*}$*:* A set of k^* independent test signals can generate $(2^{k^*} - 1)$ distinct non-zero residues. Since the number of inputs is at most $(2^{k^*} - 1)$, each input can be assigned a unique non-zero residue. This assignment ensures that any pair of inputs that drive an output are assigned two distinct residues. Thus k^* test signals are sufficient for exhausting testing of all outputs.
Case $m < (2^{k^*-1})(2^{k^*} - 1)$*:* Let us partition the set of circuit inputs into p subsets $I_1, I_2, \ldots, I_p$ such that the following two conditions are satisfied.

1. No two inputs from the same subset fan out to any common output.

2. For every pair of subsets (say I_i and I_j), there exists an input in each subset (say $\theta_i \in I_i$ and $\theta_j \in I_j$) such that these inputs (θ_i and θ_j) fan out to some common output.

Note that a pair of subsets that do not satisfy the second condition can be further combined into a single set. The first condition allows us to assign the same residue to all inputs in a subset. The second condition mandates that each pair of subsets be assigned two distinct non-zero residues. Therefore assigning a unique residue to each subset ensures exhaustive testing of all outputs.

In order to satisfy the second condition, there must exist at least one output for every pair of subsets. Hence to have a valid partition of p subsets, the circuit must have at least $p(p-1)/2$ outputs. For an $(n, m, 2)$ circuit to have a valid partition of 2^{k^*} subsets, the circuit must have at least $(2^{k^*})(2^{k^*} - 1)/2 = (2^{k^*-1})(2^{k^*} - 1)$

outputs. Since $m < (2^{k^*-1})(2^{k^*}-1)$, the circuit inputs can only be partitioned into at most $(2^{k^*}-1)$ subsets. A set of k^* independent test signals can generate $(2^{k^*}-1)$ unique non-zero residues that can be assigned to the subsets. Thus k^* test signals are sufficient for pseudo-exhaustive testing of the circuit. □

Corollary 4 *Any $(n, m, 2)$ circuit with either $n < 4$ or $m < 6$ is a MTC circuit.*

Example 15 Consider again the $(4, 6, 2)$ circuit given in Example 14. Theorem 16 states that three independent test signals are sufficient for pseudo-exhaustive testing of the circuit. In fact, three independent test signals are necessary for the circuit as evident by the residue assignment given in Example 14. □

Test Signals	Inputs	Outputs
2	3	5
3	7	27
4	15	119
5	31	495
...	...	...
k^*	$2^{k^*}-1$	$2^{k^*-1}(2^{k^*}-1)-1$

Table 6.1: Bounds on test lengths for $(n, m, 2)$ circuit

Table 6.1 shows the number of test signals sufficient for pseudo-exhaustive testing of any $(n, m, 2)$ circuit. Column 1 indicates the number of test signals. Columns 2 and 3 indicate upper bounds on the number of inputs and outputs for an $(n, m, 2)$ circuit that can be pseudo-exhaustively tested with the number of test signals given in the first column. An $(n, m, 2)$ circuit with at most seven inputs or with at most 27 outputs can be pseudo-exhaustively tested with three test signals. For example, an $(8, 16, 2)$ circuit requires four (three) test signals as far as the inputs (outputs) are concerned. Hence three test signals are sufficient for pseudo-exhaustive testing of any $(8, 16, 2)$ circuit.

Theorem 17 *For any (n, m, k) circuit, let $k^*(\geq k)$ be the smallest number satisfying the following inequality*

$$m \leq 2^{k^*-k+1} \tag{6.2}$$

Then k^* independent test signals are sufficient for pseudo-exhaustive testing of the circuit.

Proof : Since the circuit has only at most 2^{k^*-k+1} outputs, any input can drive only at most 2^{k^*-k+1} outputs. From Lemma 13, we know that all inputs that drive at most 2^{k^*-k+1} outputs can be assigned proper residues by k^* independent test signals. □

Theorem 18 *For any (n, m, k) circuit, our bound on the number of independent test signals for pseudo-exhaustive testing given by Theorem 17 is tighter than the bound derived in [3].*

Proof : It has been shown in [3] that k^* independent test signals are sufficient if k^* satisfies the inequality

$$m \leq 2^{k^*-k}. \tag{6.3}$$

It is evident that our bound is tighter than the bound derived in [3] as we can accommodate twice the number of outputs for the same number of test signals. □

Conjecture 1 *For any (n, m, k) circuit, let k^* ($\geq k$) be the smallest number satisfying the following inequality*

$$m \leq 2^{k^*-k+2} + 1 \tag{6.4}$$

Then k^* independent test signals are sufficient for pseudo-exhaustive testing of the circuit.

Conjecture 1 is true for MTC circuits since any (n, m, k) circuit with $m \leq 5$ is a MTC circuit as per Theorem 15.

Table 6.2 shows three upper bounds on the number of outputs for an (n, m, k) circuit that can be pseudo-exhaustively tested with the number of test signals given in the first column. For example, any (n, m, k) circuit with at most four outputs can be pseudo-exhaustively tested with $(k + 2)$ test signals according to the bound derived in [3]. Theorem 17 states that $(k + 1)$ test signals are sufficient for pseudo-exhaustive testing of any (n, m, k) circuit with $m \leq 4$. Conjecture 1 states that k test

Number of Test Signals	Number of outputs		
	Akers	Theorem 17	Conjecture 1
k	1	2	5
$k+1$	2	4	9
$k+2$	4	8	17
$k+3$	8	16	33
...	...	...	...
k^*	2^{k^*-k}	2^{k^*-k+1}	$2^{k^*-k+2}+1$

Table 6.2: Bounds on test lengths for (n, m, k) circuit

signals are sufficient for the same circuit. For a given number of test signals (say k^*), we guarantee exhaustive testing of twice the number of output cones (Theorem 17) and possibly four times the number of output cones (Conjecture 1) compared to the number of output cones guaranteed by the bound in [3].

6.4 Cone Dependent Bounds

Given an (n, m, k) circuit, we can utilize the information about cone dependencies to achieve a tighter bound on the number of independent test signals required for pseudo-exhaustive testing of the circuit. We shall derive tight upper bounds for both LFSR/XOR and LFSR/SR structures and show that our bounds are better than those derived in [3] and [4].

Let us consider the (n, m, k) circuit along with the following notation. The n inputs are denoted as $\theta_i, i = 1, 2, \ldots, n$, and the m outputs are denoted as $O_j, j = 1, 2, \ldots, m$, respectively. Input θ_i is assigned a unique index π_i, where $1 \leq \pi_i \leq n$. A permutation of inputs is specified completely by the n-tuple $(\pi_1, \pi_2, \ldots, \pi_n)$. The default permutation is given by $\pi_i = i$, $i = 1, 2, \ldots, n$. We shall assume the default permutation of inputs unless stated otherwise.

The input dependencies for an output is represented by an ordered set of inputs. The inputs are arranged in the ordered set in increasing order of their indices. Consider output O_j being driven by k inputs $\theta_{i_1}, \theta_{i_2}, \ldots, \theta_{i_k}$. Let $1 \leq i_1 < i_2 < \ldots i_k \leq n$. Under the default permutation of inputs, the input dependencies for O_j is represented by the ordered set $\{\theta_{i_1}, \theta_{i_2}, \ldots, \theta_{i_k}\}$. Let $p_{i,j}$ denote the position of θ_i in the ordered

dependency set for O_j. If θ_i drives O_j, then $p_{i,j}$ takes appropriate value between 1 and k, otherwise $p_{i,j} = 0$. Let $p_i^* = max\ \{p_{i,1}, p_{i,2}, \ldots, p_{i,m}\}$ denote the maximum position in which θ_i occurs among the input dependencies for all m outputs. Let $f_{i,j}$ be a Boolean variable such that $f_{i,j} = 1$ if $p_{i,j} > 0$ and $f_{i,j} = 0$ if $p_{i,j} = 0$. Let $f_i^* = \sum_{j=1}^{m} f_{i,j}$ denote the number of occurrences (frequency) of θ_i among all m outputs. The notation is illustrated in the following example.

Example 16 Consider a $(6, 6, 3)$ circuit along with inputs denoted by θ_1 through θ_6 and outputs denoted by O_1 through O_6 respectively. Let us assume the default permutation where θ_i is assigned index $\pi_i = i$. Let the input dependencies for the six outputs be the following ordered sets: $\{\theta_1, \theta_2, \theta_3\}$, $\{\theta_1, \theta_3, \theta_4\}$, $\{\theta_2, \theta_3, \theta_5\}$, $\{\theta_2, \theta_4, \theta_5\}$, $\{\theta_1, \theta_5, \theta_6\}$ and $\{\theta_4, \theta_5, \theta_6\}$ respectively. For θ_2 we have $p_{2,j}$ values ($j = 1, 2, \ldots, 6$) of 2, 0, 1, 1, 0 and 0, respectively, and $f_{2,j}$ values ($j = 1, 2, \ldots, 6$) of 1, 0, 1, 1, 0 and 0, respectively. Hence $p_2^* = 2$ and the frequency f_2^* equals three. □

6.4.1 LFSR/XORs

We shall derive tight upper bounds for the pseudo-exhaustive test sets generated by LFSR/XOR structures for a given (n, m, k) circuit. Since these bounds are derived based on the ordering of the circuit inputs, we shall determine the best permutation of inputs in order to achieve the best improvement of these bounds.

Theorem 19 *For an (n, m, k) circuit, let $p_{i,j}$, p_i^* and $f_{i,j}$ be the circuit parameters (defined earlier) characterizing the cone dependencies. Let k^* be the smallest number satisfying the following inequality for all inputs θ_i, $1 \leq i \leq n$.*

$$\lceil 2^{2p_i^* - 2 - k^*} \rceil + \sum_{j=1}^{m} f_{i,j} \{ 2^{p_{i,j}-1} - \lceil 2^{p_i^* + p_{i,j} - 2 - k^*} \rceil \} < 2^{k^*} \tag{6.5}$$

Then k^* independent test signals are sufficient for pseudo-exhaustive testing of the circuit.

Proof : An (n, m, k) circuit can be pseudo-exhaustively tested by k^* independent test signals if all inputs can be assigned proper residues from the k^*-dimensional space (say S). Inputs θ_1 through θ_n are considered in succession for residue assignment. Let us assume that inputs θ_1 through θ_{i-1} have been successfully assigned

proper residues and input θ_i is under consideration. We shall explore the feasibility of assigning a proper residue for θ_i from S.

Consider an output O_{j^*} in which θ_i appears at position p_i^* among the input dependencies. For this output, θ_i appears along with $(p_i^* - 1)$ inputs that have been already assigned proper residues. These $(p_i^* - 1)$ residues span a $(p_i^* - 1)$-dimensional subspace (say S_{j^*}) and all the elements in this subspace are prohibited residues for θ_i. Consider another output O_j with $p_{i,j} > 0$ and hence $f_{i,j} = 1$. For O_j, θ_i appears along with $(p_{i,j} - 1)$ inputs that have been already assigned proper residues. These residues span a $(p_{i,j} - 1)$-dimensional space (say S_j) and all the elements in this space are prohibited residues for θ_i. From Theorem 10, we know that subspaces S_{j^*} and S_j have at least $\lceil 2^{p_i^* + p_{i,j} - 2 - k^*} \rceil$ common elements. Hence the number of prohibited residues for θ_i due to O_{j^*} and O_j is given by

$$\begin{aligned} |S_{j^*} \cup S_j| &= |S_{j^*}| + |S_j| - |S_{j^*} \cap S_j| \\ &\leq 2^{p_i^* - 1} + 2^{p_{i,j} - 1} - \lceil 2^{p_i^* + p_{i,j} - 2 - k^*} \rceil \end{aligned}$$

Considering all outputs driven by θ_i, the total number of prohibited residues for θ_i is given by

$$\begin{aligned} |\bigcup_{j=1; p_{i,j} > 0}^{m} S_j| &\leq |S_{j^*}| + \sum_{j=1; j \neq j^*}^{m} f_{i,j} \{ |S_j| - |S_{j^*} \cap S_j| \} \\ &\leq 2^{p_i^* - 1} + \sum_{j=1; j \neq j^*}^{m} f_{i,j} \{ 2^{p_{i,j} - 1} - \lceil 2^{p_i^* + p_{i,j} - 2 - k^*} \rceil \} \\ &= \lceil 2^{2p_i^* - 2 - k^*} \rceil + \sum_{j=1}^{m} f_{i,j} \{ 2^{p_{i,j} - 1} - \lceil 2^{p_i^* + p_{i,j} - 2 - k^*} \rceil \} \end{aligned}$$

Thus the LHS expression of Equation 6.5 gives an upper bound on the total number of prohibited residues for θ_i. As long as this expression is less than 2^{k^*}, a proper residue from S is guaranteed for θ_i. Hence the satisfiability of Equation 6.5 for all inputs guarantees the existence of proper residues for all inputs in the space generated by k^* independent test signals. □

Theorem 20 *The cone dependent bound on the number of independent test signals given by Theorem 19 is tighter than the cone independent bound given by Theorem 17.*

Proof : It is enough to show that the cone independent bound can be derived by assuming the worst case in the derivation of cone dependent bound. For an input θ_i with $p_{i,j} = k$ for all m outputs, we have $p_i^* = k$ and Equation 6.5 simplifies to

$$\begin{aligned} \lceil 2^{2k-2-k^*} \rceil + m \times (2^{k-1} - \lceil 2^{2k-2-k^*} \rceil) &< 2^{k^*} \\ \Rightarrow \quad (m-1)(2^{k-1} - \lceil 2^{2k-2-k^*} \rceil) &< 2^{k^*} - 2^{k-1} \\ \Rightarrow \quad m &\leq 2^{k^*-k+1} \end{aligned}$$

Thus the cone dependent bound is tighter than the cone independent bound. □

Theorem 21 *For an (n, m, k) circuit, let I_i denote the set of inputs that drive exactly i outputs in the circuit. Let k^* be the smallest number satisfying the following inequality*

$$\sum_{i=2^{k^*-k+1}+1}^{m} |I_i| \leq (k^* + 1) \tag{6.6}$$

Then k^* independent test signals are sufficient for pseudo-exhaustive testing of the circuit.

Proof : The number of inputs that drive more than 2^{k^*-k+1} outputs is given by the LHS expression of Equation 6.6. Assume the worst case, where the number of such inputs equals $(k^* + 1)$. Assign the $(k^* + 1)$ residues $\{1, x, x^2, \ldots, x^{k^*-1}, 1 + x + \cdots + x^{k^*-1}\}$ to those $(k^* + 1)$ inputs. Any output will be driven by only at most k of these $(k^* + 1)$ inputs and hence this assignment ensures that all those $(k^* + 1)$ inputs are assigned proper residues. From Lemma 13, we know that inputs in $I_j \ \forall j \leq 2^{k^*-k+1}$ are guaranteed of proper residues in a k^*-dimensional space. □

6.4.1.1 Improvement on Bounds by Input Permutation

Given an (n, m, k) circuit, the bound on the number of independent test signals given by Theorem 19 can be improved by allowing permutation of inputs. We shall describe a permutation algorithm that assigns unique indices to circuit inputs resulting in low (high) $p_{i,j}$ values for inputs driving many (few) outputs. The algorithm modifies the circuit parameters (that characterizes the cone dependencies) and allows Equation 6.5 to be satisfied for a smaller value of k^*.

Procedure XORBound

Input: Output cone dependencies of (n, m, k) circuit.
Output: Upper bound on the number of independent test signals k^* ($\geq k$).

1. Determine all dominating outputs and consider only the reduced circuit.
2. $k^* \leftarrow k$.
   /* k^* is the number of independent test signals */
3. Determine f_i^* for input θ_i $\forall i = 1, 2, \ldots, n$.
   /* determine the frequencies of inputs */
4. $\pi_i \leftarrow 0$ $\forall i = 1, 2, \ldots, n$; $n^* \leftarrow n$.
   /* initialize the indices of inputs and n^* is the current highest index */
5. For each unassigned θ_i do
   (a) If $f_i^* \leq 2^{k^*-k+1}$ then { $\pi_i \leftarrow n^*$; $n^* \leftarrow n^* - 1$ }
6. While n^* is decremented do
   (a) For each unassigned θ_i do
      i. $\pi_i \leftarrow n^*$.
      ii. Check the satisfiability of Equation 6.5 for θ_i.
      iii. If the equation is satisfied then $n^* \leftarrow n^* - 1$; else $\pi_i \leftarrow 0$.
7. If $n^* > 0$ then
   (a) If $k^* = n$, go to Step 8.
   (b) $k^* \leftarrow k^* + 1$; Go to Step 4.
8. Output the number of test signals (k^*).

The algorithm *XORBound* determines a minimal number of independent test signals that are sufficient for pseudo-exhaustive testing of a given circuit. Lemma 12 enables us to consider only dominating outputs for determining the bound on test

length. Lemma 13 states that a set of k^* test signals guarantees proper residues for each input that drives at most 2^{k^*-k+1} outputs and hence all these inputs are assigned highest possible indices. From the remaining set of unassigned inputs, an input (say θ_i) is assigned the current highest index (n^*) provided it satisfies Equation 6.5. The $p_{i,j}$ values for θ_i are determined based on the fact that the remaining unassigned inputs can have indices only less than n^*. The unassigned inputs are repeatedly considered for assignment until there is no decrease in the value of n^*. Any further existence of unassigned inputs mandates an increment to the number of test signals and an iteration of the entire algorithm.

The complexity of the algorithm can be computed as follows. Every iteration of the *while* loop results in assigning proper indices to one or more inputs. The number of iterations of the *while* loop is bounded above by $n(n+1)/2$ since every iteration can result in assigning a proper index to only one input. The satisfiability check for input θ_i involves determining $p_{i,j}$ values for all m outputs. Thus the complexity of the *while* loop is given by $O(mn^2)$. The number of iterations of the entire algorithm is bounded above by $(n-k)$. Thus the complexity of the algorithm is given by $O(mn^3)$, where n and m are the number of inputs and outputs to the circuit respectively.

Note that in general, considering all permutations of inputs and Theorem 19 for determining the tightest possible bound has exponential complexity. The following theorem states that our permutation algorithm of polynomial complexity is sufficient to find the tightest possible bound using Theorem 19.

Theorem 22 *The XORBound algorithm (of polynomial complexity) determines the tightest possible bound on the number of test signals that can be achieved using Theorem 19.*

Proof : Let I denote the set of circuit inputs. Let k^* be the number of test signals considered during some iteration of the *XORBound* algorithm. Assume that a subset of inputs (say I_1 with $|I_1| = n_1$) are not assigned indices after the completion of the *while* loop. This implies that all the $(n - n_1)$ inputs in $I - I_1$ have been successfully assigned indices greater than n_1. Let Π_1 denote the partial permutation of circuit inputs in which n_1 inputs are not assigned indices and the remaining $(n - n_1)$ inputs are assigned indices greater than n_1. The *while* loop must have terminated after determining that none of the inputs in I_1 can be assigned the index

n_1. We claim that I_1 is the minimum set under any partial permutation of inputs and shall prove the claim as follows.

Let Π_2 denote another partial permutation of circuit inputs that results in the *minimum* subset of inputs (say I_2 with $|I_2| = n_2$) such that (1) none of the n_2 inputs in I_2 can be assigned the index n_2 and satisfy Equation 6.5 and (2) all of the $(n - n_2)$ inputs in $I - I_2$ are assigned proper indices greater than n_2 and satisfy Equation 6.5.

We shall prove that $I_1 = I_2$ by contradiction. Let $I_1' = I_1 - (I_1 \cap I_2)$ and consider an input $\theta_1 \in I_1'$. Input θ_1 does not satisfy Equation 6.5 with index n_1 under Π_1 but satisfies the equation with an index greater than n_2 under Π_2. Hence for some output (say O_j), the $p_{1,j}$ value for θ_1 under Π_1 must be greater than the $p_{1,j}$ value under Π_2. This is possible only if there exists another input (say θ_2) that appears before θ_1 among the dependencies for O_j under Π_1 and appears after θ_1 among the dependencies for O_j under Π_2. This implies (1) $\theta_2 \in I_1$ under Π_1; (2) $\theta_2 \notin I_2$ under Π_2 and (3) θ_2 being assigned an index greater than that of θ_1 under Π_2. Repeating the argument for $\theta_2 \in I_1'$ leads to a third input $\theta_3 \in I_1'$ and θ_3 being assigned an index greater than that of θ_2 under Π_2. The argument can thus be repeated for all inputs in I_1'. The argument fails for the last input in I_1' since there are no more inputs left in I_1'. This is a contradiction. Hence there exists no $\theta_1 \in I_1'$ and $I_1 \subseteq I_2$. Since I_2 is the minimum set by definition, $I_1 = I_2$.

Thus the *XORBound* algorithm determines the minimum set of inputs that cannot be assigned indices and iterates with an increment to the number of test signals. Thus the algorithm determines the tightest possible bound on the number of test signals that can be achieved using Theorem 19. □

Example 17 Consider the $(6, 6, 3)$ circuit described in Example 16. Akers' bound using Equation 6.3 requires six signals. Our bound using Equation 6.5 without allowing a permutation of inputs requires four signals. Applying the *XORBound* algorithm reduces our bound to three test signals. The circuit can be tested with three independent test signals. Residues $\{1, x, x^2, 1 + x, 1 + x^2, x\}$ are assigned to inputs 1 through 6 respectively. □

Table 6.3 presents the upper bounds on test lengths for LFSR/XORs for the partitioned versions of ISCAS combinational benchmark circuits. The benchmark circuits are partitioned using our partitioning procedure [34] such that the output

Ckt	(n,m,k)	Dominating Outputs	Bound on Test Length		
			Akers [3]	with default permutation	with best permutation
c432	(56,27,20)	20	2^{25}	2^{20}	2^{20}
c499	(49,40,14)	40	2^{20}	2^{16}	2^{14}
c880	(70,36,17)	29	2^{22}	2^{18}	2^{17}
c1355	(49,40,14)	40	2^{20}	2^{16}	2^{14}
c1908	(47,39,20)	26	2^{25}	2^{20}	2^{20}
c2670	(262,169,20)	117	2^{27}	2^{20}	2^{20}
c3540	(108,80,20)	57	2^{26}	2^{21}	2^{20}
c5315	(215,160,20)	91	2^{27}	2^{21}	2^{20}
c6288	(99,98,20)	39	2^{26}	2^{22}	2^{21}
c7552	(286,187,20)	69	2^{27}	2^{20}	2^{20}

Table 6.3: Upper bounds on pseudo-exhaustive test lengths for LFSR/XORs

cones are driven by 20 or less inputs. Columns 2 and 3 present the (n, m, k) characteristics and the number of dominating outputs of these circuits. The last three columns present the bounds on test lengths by considering the reduced circuits. Akers' bounds are determined by using Equation 6.3. Our bounds with default permutation of inputs are determined using Equation 6.5. The *XORBound* algorithm achieves tighter bounds on pseudo-exhaustive test lengths by determining one of the best permutation of inputs. The improvement of the bounds by allowing permutation of inputs is evident from the table. Our bounds determined by allowing permutation of inputs are optimal for all circuits except *c*6288, and a few orders of magnitude smaller than Akers' bound. It should be noted that we exploited the information about the input dependencies of the output cones in the circuit.

6.4.2 LFSR/SRs

An (n, m, k) circuit can be pseudo-exhaustively tested by an LFSR/SR if there exists a primitive feedback polynomial of degree $k^*(\geq k)$ such that the residues assigned to the inputs driving each output are linearly independent (Theorem 13). We shall assume the default permutation of inputs where input θ_i is fed by the ith stage of

LFSR/SR. Input θ_i is assigned the residue $x^i \ mod \ P(x)$, where $P(x)$ is the primitive feedback polynomial of the LFSR/SR.

Definition 8 *The primitive feedback polynomial of an LFSR/SR considered for a given circuit is said to be* **inapplicable** *if the polynomial results in a set of linearly dependent residues for the set of inputs driving some output of the circuit.*

Theorem 23 (Golomb 1982) *The total number of primitive polynomials of degree k^* is given by $\Phi(2^{k^*}-1)/k^*$, where Φ is Euler's phi function.*

Theorem 24 *For an (n, m, k) circuit, let $p_{i,j}$ and $f_{i,j}$ be the circuit parameters (defined earlier) characterizing the cone dependencies. Let k^* be the smallest number satisfying the following inequality*

$$\sum_{j=1}^{m} \sum_{i=k^*}^{n} i \times f_{i,j}(2^{p_{i,j}-1}-1) < \Phi(2^{k^*}-1) \approx 2^{k^*}-1 \tag{6.7}$$

Then a LFSR/SR based on a degree k^* primitive polynomial is sufficient for pseudo-exhaustive testing of the circuit.

Proof : (The following is an extension of the arguments presented in [4]). Let us consider output O_j and determine an upper bound on the number of inapplicable primitive polynomials of degree k^* for this output. Let the input dependencies of O_j contain input θ_i in $p_{i,j}$th position. Let inputs $\theta_{i_1}, \theta_{i_2}, \ldots, \theta_{i_{p_{i,j}-1}}$ appear in positions 1 through $(p_{i,j}-1)$ respectively for this output. An applicable primitive polynomial $P(x)$ of degree k^* should ensure that the residues $\{x^{i_1} \ mod \ P(x), \ x^{i_2} \ mod \ P(x), \ldots, \ x^{i_{p_{i,j}-1}} \ mod \ P(x), \ x^i \ mod \ P(x)\}$ are linearly independent. In other words, each polynomial $Q(x)$ of the form $x^i + \sum_{q=1}^{p_{i,j}-1} a_q x^{i_q}$ (where $a_q = 0$ *or* 1 and not all of them are zeros) must not be divisible by $P(x)$. There are $(2^{p_{i,j}-1}-1)$ such polynomials $Q(x)$ of degree i. Each one of the polynomials $Q(x)$ is divisible by no more than i/k^* distinct primitive polynomials of degree k^*. Therefore an upper bound on the number of inapplicable primitive polynomials of degree k^* that may assign linearly dependent residues to some inputs in the set $\{\theta_{i_1}, \theta_{i_2}, \ldots, \theta_{i_{p_{i,j}-1}}, \theta_i\}$ driving O_j is given by the expression $E_{i,j} = (i/k^*)(2^{p_{i,j}-1}-1)$. Summing up $E_{i,j}$ for all values of $i \geq k^*$ yields an upper bound on the number of inapplicable primitive polynomials

of degree k^* for O_j. There is no need to consider any $Q(x)$ polynomial of degree less than k^* since the primitive polynomial $P(x)$ is of degree k^*. The Boolean variable $f_{i,j}$ ensures that only those inputs that drive O_j are considered.

Again summing up for all values of j yields an upper bound on the total number of inapplicable primitive polynomials of degree k^* for all circuit outputs. This double summation is given by the LHS expression of Equation 6.7.

Theorem 23 states that the total number of primitive polynomials of degree k^* is given by $\Phi(2^{k^*}-1)/k^*$. To ensure that the total number of inapplicable primitive polynomials of degree k^* is less than the total number of primitive polynomials of degree k^*, we must have

$$\begin{aligned} \sum_{j=1}^{m}\sum_{i=k^*}^{n} i/k^* \times f_{i,j}(2^{p_{i,j}-1}-1) &< \Phi(2^{k^*}-1)/k^* \\ \Rightarrow \sum_{j=1}^{m}\sum_{i=k^*}^{n} i \times f_{i,j}(2^{p_{i,j}-1}-1) &< \Phi(2^{k^*}-1) \approx 2^{k^*}-1 \end{aligned}$$

Thus the satisfiability of Equation 6.7 guarantees a primitive polynomial of degree k^* applicable to all outputs. □

Theorem 25 *For any (n, m, k) circuit, our bound on the degree of LFSR/SR given by Theorem 24 is tighter than the bound derived in [4].*

Proof : It has been shown in [4] that a LFSR/SR of degree $\hat{k}^*$ is sufficient for pseudo-exhaustive testing of an (n, m, k) circuit if $\hat{k}^*$ satisfies the equation

$$n \times m \times (2^k - 1) < \Phi(2^{\hat{k}^*}-1) \approx 2^{\hat{k}^*}-1 \tag{6.8}$$

We shall show that the value of k^* in Equation 6.7 is bounded above by the value of $\hat{k}^*$ in Equation 6.8.

Let E_j denote the expression $\sum_{i=k^*}^{n} i \times f_{i,j} \times (2^{p_{i,j}-1}-1)$. The LHS expression of Equation 6.7 can be expressed as $\sum_{j=1}^{m} E_j$. Let us consider the input dependencies for output O_j given by the ordered set $\{\theta_{i_1}, \theta_{i_2}, \ldots, \theta_{i_k}\}$. For this output we compute E_j as

$$E_j = \sum_{i=k^*}^{n} i \times f_{i,j} \times (2^{p_{i,j}-1}-1)$$

$$
\begin{aligned}
&\leq \sum_{q=1}^{k} i_q \times (2^{q-1} - 1) \\
&\leq n \times \sum_{q=1}^{k} (2^{q-1} - 1) \quad (\text{since } i_q \leq n) \\
&< n \times (2^k - 1)
\end{aligned}
$$

Summing up E_j for all values of j, we get

$$
\sum_{j=1}^{m} E_j < \sum_{j=1}^{m} n \times (2^k - 1) = m \times n \times (2^k - 1)
$$

Thus the LHS expression of Equation 6.7 is smaller than the LHS expression in Equation 6.8. Hence k^* value in Equation 6.7 is bounded above by $\hat{k}^*$ value in Equation 6.8. □

6.4.2.1 Improvement on Bounds by Input Permutation

Given an (n, m, k) circuit, the bound on the degree of the applicable primitive polynomial for an LFSR/SR given by Theorem 24 can be improved by permuting the inputs. We shall attempt to minimize the total number of inapplicable primitive polynomials given by the LHS expression in Equation 6.7. Thus improvement on the bound can be obtained for the degree of the applicable primitive polynomial for LFSR/SR. This is similar to the improvement on the bound achieved for LFSR/XORs.

Procedure SRBound

Input: Output cone dependencies of (n, m, k) circuit.
Output: Upper bound on the degree k^* ($\geq k$) of applicable primitive polynomial.

1. Determine all dominating outputs and consider only the reduced circuit.

2. $k^* \leftarrow k$.
   /* k^* is the degree of the primitive polynomial */

3. Assign indices to inputs according to the input permutation determined by the algorithm *XORBound*.

4. While Equation 6.7 is not satisfied do

   (a) If $k^* = n$, go to Step 5.

   (b) $k^* \leftarrow k^* + 1$.

5. Output the degree of the applicable primitive polynomial (k^*).

For a given circuit, the *SRBound* algorithm usually determines an applicable primitive polynomial of smaller degree than the default permutation. Only dominating outputs are considered as per Lemma 12. The input permutation determined by the *XORBound* algorithm is used to minimize the LHS expression of Equation 6.7. The satisfiability check involves computing $p_{i,j}$ values for all inputs driving each output. Since the input permutation determined by the *XORBound* algorithm is used again in the *SRBound* algorithm, the complexity of the *SRBound* algorithm is same as that of the complexity of the *XORBound* algorithm. However, the *SRBound* algorithm for LFSR/SRs does not guarantee the tightest possible bound unlike the *XORBound* algorithm for LFSR/XORs.

Example 18 Consider the $(6, 6, 3)$ circuit described in Example 16. For LFSR/SRs, Barzilai's bound determined by Equation 6.8 requires eight test signals. The bound computed using Equation 6.7 without allowing permutation of inputs requires a primitive polynomial of degree five. The *SRBound* algorithm still requires a degree five primitive polynomial. However, the circuit can be tested with an LFSR/SR using the primitive polynomial $x^4 + x + 1$. □

Table 6.4 presents the upper bounds on test lengths for LFSR/SRs for the partitioned versions of ISCAS combinational benchmark circuits. The last three columns present the bounds on test lengths by considering only the reduced circuits. Barzilai's bounds are determined using Equation 6.8 and our bounds with default permutation of inputs are determined using Equation 6.7. The *SRBound* algorithm results in tighter bounds by using the same permutation of inputs that were originally determined for LFSR/XORs. The improvement of the bounds by allowing permutation

Ckt	(n,m,k)	Dominating Outputs	Bound on Test Length		
			Barzilai [4]	with default permutation	with good permutation
c432	(56,27,20)	20	2^{31}	2^{28}	2^{26}
c499	(49,40,14)	40	2^{25}	2^{23}	2^{22}
c880	(70,36,17)	29	2^{28}	2^{26}	2^{25}
c1355	(49,40,14)	40	2^{25}	2^{23}	2^{22}
c1908	(47,39,20)	26	2^{31}	2^{27}	2^{26}
c2670	(262,169,20)	117	2^{35}	2^{30}	2^{29}
c3540	(108,80,20)	57	2^{33}	2^{31}	2^{30}
c5315	(215,160,20)	91	2^{35}	2^{32}	2^{31}
c6288	(99,98,20)	39	2^{32}	2^{30}	2^{30}
c7552	(286,187,20)	69	2^{35}	2^{32}	2^{30}

Table 6.4: Upper bounds on pseudo-exhaustive test lengths for LFSR/SRs

of inputs is evident from the table. It should be noted that our LFSR/SR bounds are of a few orders of magnitude smaller than Barzilai's bounds.

6.5 Summary

In this chapter we have first derived a few important algebraic results on the set union and intersection operations between vector subspaces. We have determined (a) the minimum overlap between distinct subspaces and (b) the minimum number of distinct subspaces contained in a vector space. These algebraic results are used in the derivation of the bounds on pseudo-exhaustive test lengths.

We have determined a few generic bounds on test lengths that are independent of the structural information about the circuit output cones. We have shown that any circuit with less than six outputs is a maximal test concurrent circuit. We have derived expressions for the number of independent test signals that are sufficient for pseudo-exhaustive testing of any given $(n, m, 2)$ circuit. The expressions are based on either the number of inputs or outputs to the circuit. Similar expressions are derived for an (n, m, k) circuit based on the number of outputs and the maximum cone size to the circuit.

We have also derived a few circuit-specific bounds utilizing the structural information about the circuit output cones. We have derived tight upper bounds on the test sets generated by LFSR/XORs and LFSR/SRs and shown that our bounds are better than those derived in [3] and [4]. We have developed algorithms of polynomial complexity to permute circuit inputs to obtain good improvements on these bounds. Our bounds provide good estimates of pseudo-exhaustive test lengths and can be used as guiding factors in designing circuit-specific TPGs. The computed theoretical bounds for the partitioned benchmark circuits comply well with the pseudo-exhaustive test lengths generated by circuit-specific TPGs as reported in [35].

Chapter 7

Pseudo-Exhaustive Test System

7.1 Introduction

The results of the research work presented in this thesis are being implemented in a prototype software system — named the **PET (pseudo-exhaustive test)** system. In this chapter we shall present the details of the PET system that forms an integral part of the BIST system developed at USC [25]. We shall also list the significant contributions and possible extensions to our research work.

PET is a test application system built upon Cbase [15], an object oriented framework for VLSI design and test applications. The PET environment is given in Figure 7.1. Circuits that need to be made testable are first hierarchically reorganized and partitioned into clouds and registers by the CRETE system [13]. The test hierarchical view of the circuits outputted by the CRETE system are compatible with both our scan and BIST systems, namely SIESTA [12] and BITS [25].

7.2 CRETE System

Circuit designers tend to perceive circuits as a collection of functional logic blocks. A functional logic block is usually composed of many levels of hierarchy and invariably contains storage elements such as latches or flip-flops. Test engineers tend to perceive circuits as a collection of combinational logic blocks separated by the storage elements. Thus the perception of circuits differs drastically between the circuit designers and the test engineers. Test engineers develop testable versions of the

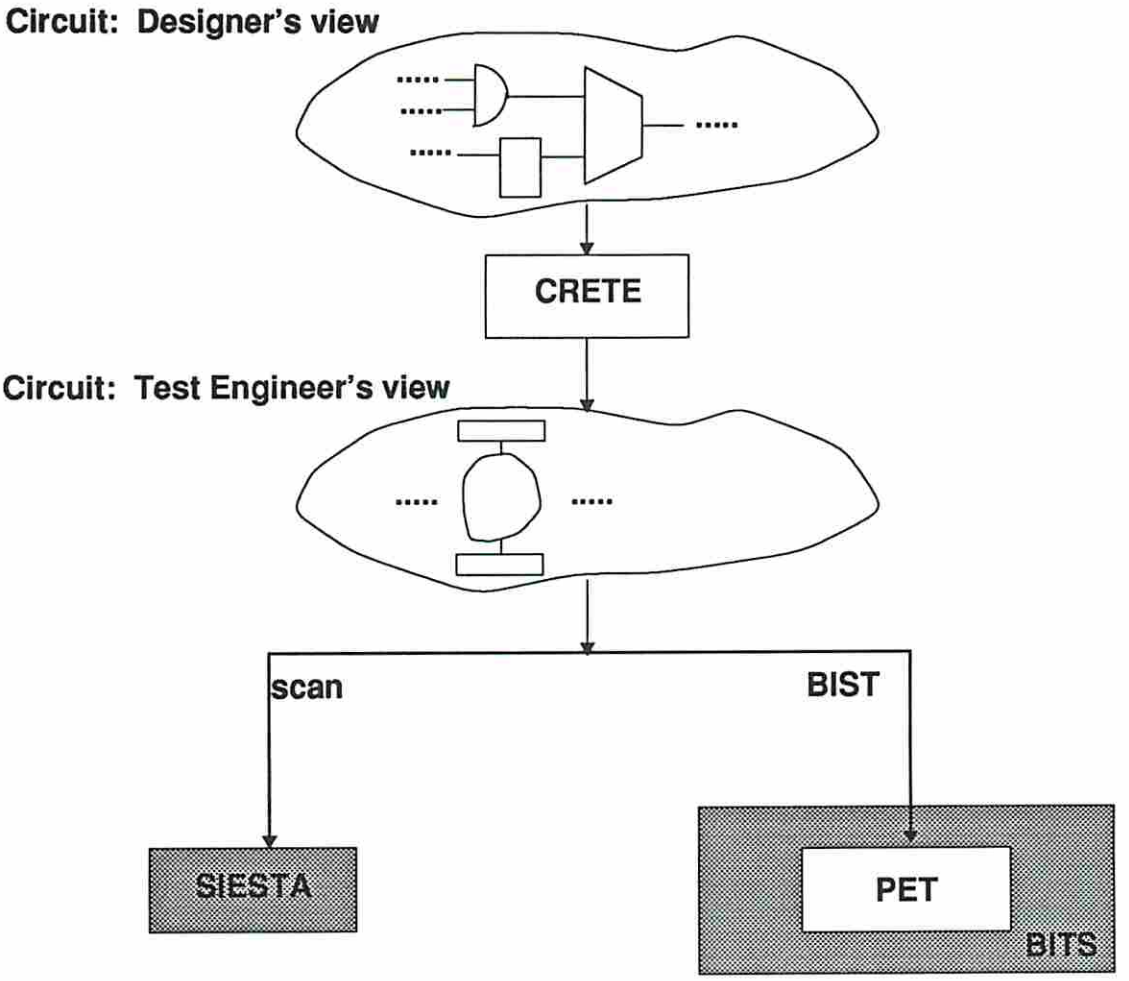


Figure 7.1: PET enviroment

circuits designed by circuit designers. In general, the hierarchical circuit description provided by the circuit designers may not be suited for the test engineers to apply various testable design methodologies (TDMs) [1]. Circuits may need to be hierarchically reorganized prior to the application of suitable TDMs.

We have developed a methodology, named *CRETE* [13], for hierarchical reorganization of circuits. **CRETE** is an acronym for **C**louding, hierarchical **R**eorganization, **E**quivalence determination, **T**est methodology embedding and **E**diting. CRETE *bridges the gap* between circuit designers and test engineers and can be interpreted as a meta-methodology that enables application of and experimentation with various TDMs. The required hierarchical reorganization of circuits is carried out by CRETE and various TDMs can then be applied to the hierarchically reorganized circuits.

Scan based approaches usually assume a gate and flip-flop model of the circuit, while BIST techniques assume a clustering based on blocks of logic separated by registers. Many TDMs result in very large combinational or sequential circuits. It is desirable to have a technique for partitioning circuits to reduce (1) the test generation effort and (2) the test application time. CRETE attempts to achieve these goals by partitioning and reorganizing the hierarchical descriptions of circuits. The combinational logic gates are clustered to form disjoint partitions referred to as *clouds* and the storage elements are clustered to form *registers*. The clouds and the registers form *canonical partitions* of circuits. For scan designs, deterministic tests can be generated for the cloud and the registers can be modified to have scan capabilities. For BIST designs, the cloud can be tested by modifying the registers as LFSRs.

7.3 PET Flowchart

The flow chart of the PET system is given in Figure 7.2. The input to the system consists of the following.

1. Gate level description of a combinational circuit.

2. Constraint on hardware overhead in terms of percentage with respect to the total number of gates in the circuit (say $\leq P\%$).

3. Constraint on test time in terms of the maximum number of patterns allowed (say $\leq 2^{k^*}$).

The circuit is first processed to determine its maximum size. The necessary condition to satisfy the constraint on test time is that each of the output cones must be driven by no greater than k^* inputs. If necessary, the circuit is partitioned to reduce its maximum cone size to a value less than or equal to k^*. Partitioning involves placement of segmentation cells on appropriate gate outputs in the circuit. During the iterative partitioning process, the constraint on hardware overhead is checked after the placement of each cell in the circuit. If the constraint is not satisfied, the user decides whether to modify the constraints or to exit the system.

Test pattern generators are then designed for the partitioned circuit. Different LFSRs of the same degree are considered as part of the TPG designs. Though the maximum cone size of the partitioned circuit is no greater than k^*, sometimes it may not be possible to generate a pseudo-exhaustive test set of size 2^{k^*}. TPG designs incur hardware overhead and hence the constraints on both hardware overhead and test time are checked during the design process. Again the user intervenes to make decision upon the violation of any of the two constraints.
The output of the system consists of the following.

1. Partitioned version of the original circuit.

2. Circuit-specific TPG design(s) for the partitioned circuit.

3. Hardware overhead and test time involved with the test strategy.

The PET system is written in C and C++ and is being integrated as a subsystem within the BITS system. The BITS system invokes PET subsystem for determining the pseudo-exhaustive testable versions of the subcircuits that require full comprehensive fault coverage.

7.4 Research Contributions

The research work presented in this thesis can be classified into three major categories, namely *partitioning*, *test pattern generation* and *bounds on test lengths*. We shall describe our significant contributions under each of these categories.

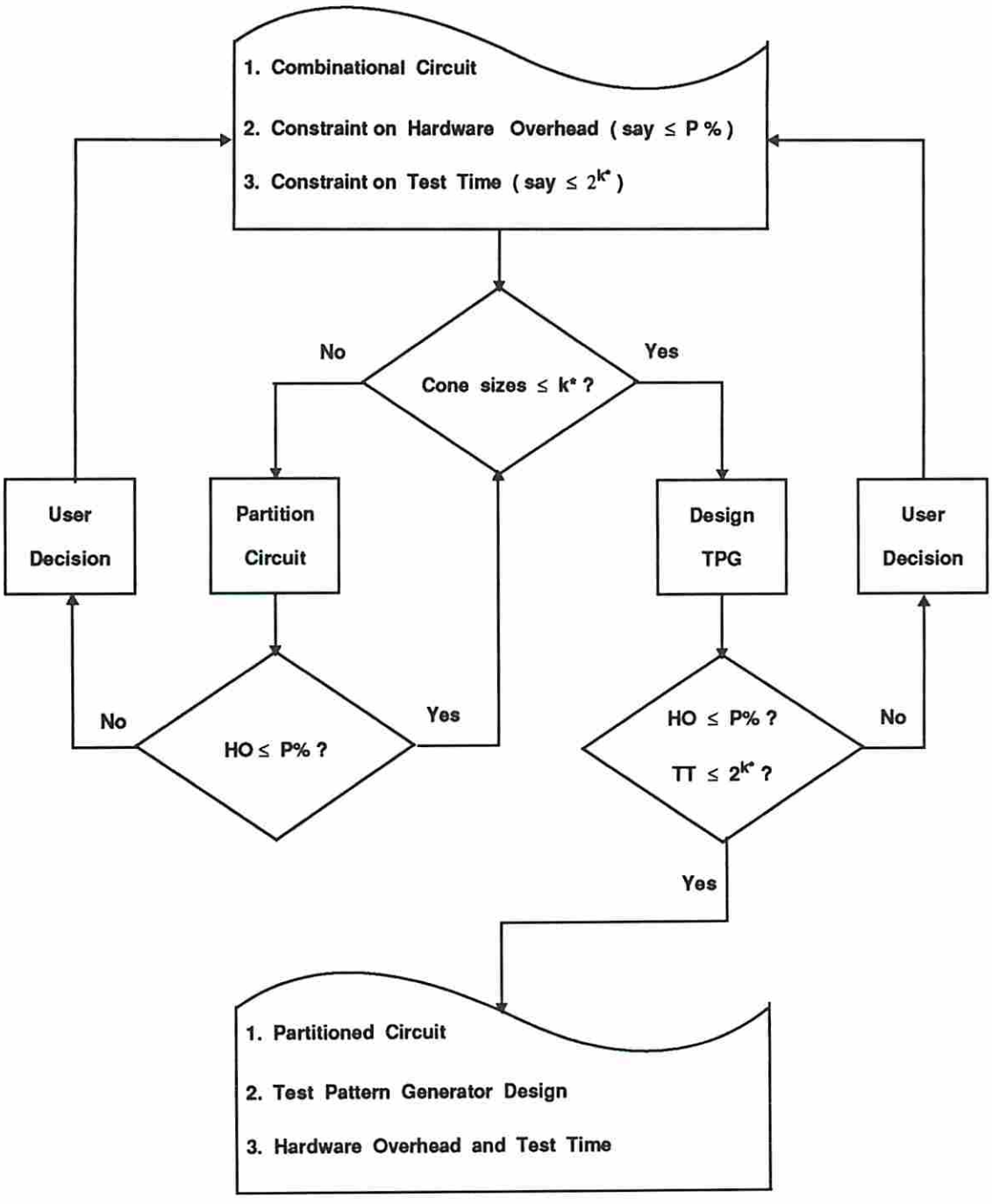


Figure 7.2: PET Flowchart

7.4.1 Partitioning

Pseudo-exhaustive test strategy is *attractive* since, compared to exhaustive testing, it offers most of the benefits with significantly less number of test patterns. The maximum cone size of the given circuit dictates the lower bound for the pseudo-exhaustive test time. Pseudo-exhaustive testing may not be practical for circuits with output cones driven by a large number of inputs. Chapter 3 presents an efficient partitioning strategies for reducing the maximum cone sizes of various classes of logic circuits. Reduction in the maximum cone size amounts to a reduction in the pseudo-exhaustive test time but results in the addition of hardware overhead. Thus there exists a trade-off between test time and hardware overhead in designing a pseudo-exhaustive self-testable versions of logic circuits.

Logic circuits can be classified into various classes such as (1) circuits without any fanouts, (2) circuits with or without non-reconvergent fanouts, (3) two-level or multi-level circuits, and (4) iterative logic array (ILA) structures. We have developed efficient partitioning procedures for all these various classes of circuits. Two-level and multi-level fanout-free circuits can be partitioned with the optimal number of segmentation cells using our procedures which have polynomial complexity. We have determined a few important characteristics of circuits with reconvergent fanouts that justify the need for partitioning procedures of exponential complexity. We have derived upper bounds on the number of segmentation cells required for partitioning ILA structures. The expressions are derived based on the regularity of these structures.

We have formulated the partitioning problem as the classical integer linear programming (ILP) problem. Optimal solutions for small circuits can be obtained by solving the ILP formulation. However, the formulation may not be computationally viable for large circuits since the number of constraints grows non-linearly with the number of levels in the circuit. To handle large circuits with reconvergent fanouts, we have developed an efficient heuristic procedure of polynomial complexity based on the graph-theoretical concept of articulation nodes [10]. The quality of our heuristic approach is demonstrated on the ISCAS combinational benchmark circuits. We have extended the partitioning strategy for balanced sequential circuits [12].

Chapter 4 presents our partitioning method for logic circuits to achieve maximal test concurrency during the test mode. The heuristic procedure is based on the graph-theoretical concept of bridges [10]. Circuits partitioned for maximal test concurrency can be pseudo-exhaustively tested with the minimum test set in a single test session. Pseudo-exhaustive test sets can be easily generated using maximal length LFSRs or counters. Circuits can be first partitioned for cone size reduction and can be further partitioned to achieve maximal test concurrency.

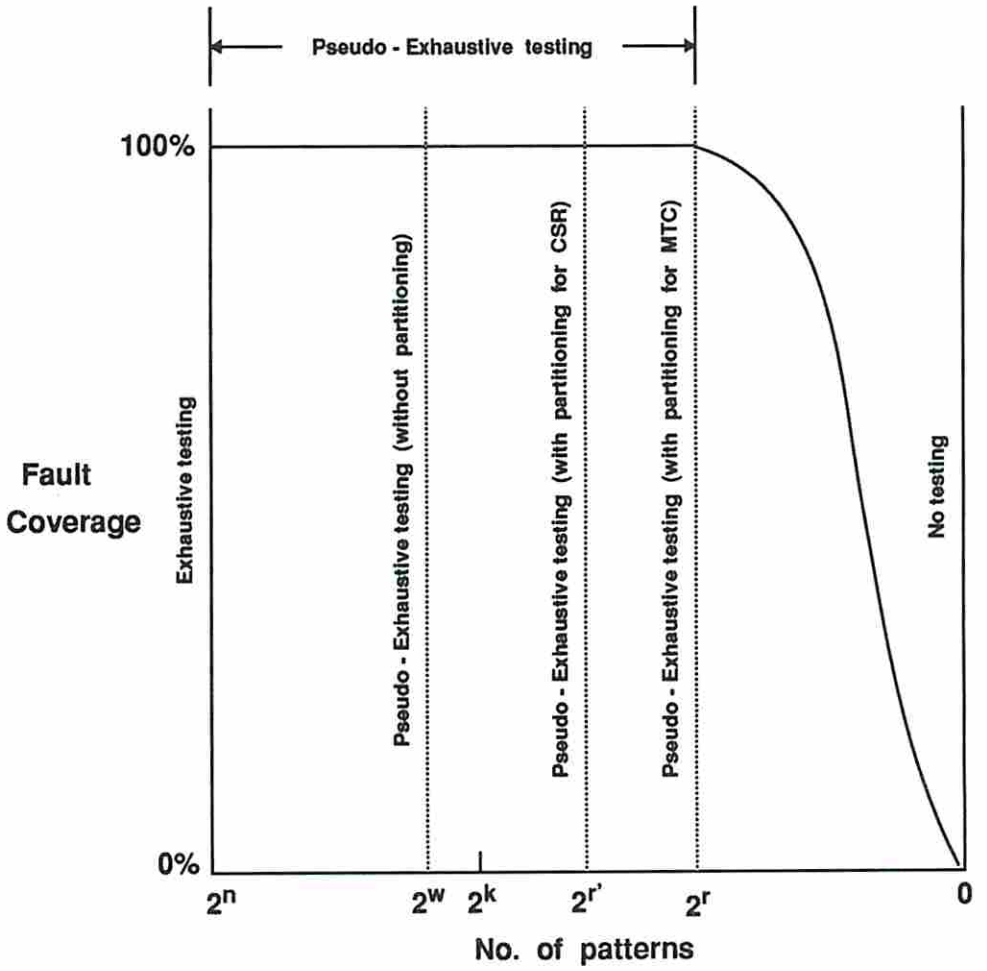


Figure 7.3: Pseudo-exhaustive testing spectrum of an (n, m, k) circuit

Figure 7.3 shows the stuck-at fault coverage curve with respect to the number of distinct test patterns applied to an (n, m, k) circuit. Exhaustive testing applies the complete set of all possible 2^n patterns to the circuit. On the contrary, the circuit

can be pseudo-exhaustively tested (without any partitioning) by applying a test set of 2^w patterns, where $k \leq w \leq n$. The maximum cone size (k) of the circuit can be reduced to a desired value (say r) by partitioning the circuit with a minimal number of segmentation cells. The circuit partitioned for cone size reduction (CSR) can be pseudo-exhaustive tested with $2^{r'}$ patterns, where $r \leq r' < k$. The circuit can be partitioned further for achieving maximal test concurrency (MTC) during the test mode. Thus partitioning for both CSR and MTC results in a pseudo-exhaustive test set containing 2^r patterns, where r is the desired cone size limit. Thus partitioning significantly reduces the size of a pseudo-exhaustive test set ensuring full coverage of stuck-at faults. However, a price is paid in terms of hardware overhead for the reduction in test time.

7.4.2 Test Pattern Generation

A pseudo-exhaustive test set for an (n, m, k) circuit contains an exhaustive set of patterns for each of the m output cones of the circuit. The size of the test set is bounded below by 2^k and bounded above by 2^n. Circuit-specific TPGs [3, 4] utilize the cone dependency information of the given circuit and usually generate test sets of sizes much smaller than that of the universal TPGs [37, 42, 43].

Chapter 5 describes our novel circuit-specific TPG designs that utilize minimal hardware and generate minimal pseudo-exhaustive test sets. Maximal length LFSRs form the basic underlying structure of our TPG designs. Among our novel TPGs, convolved LFSR/SRs have great potential to generate minimum test sets. Convolved LFSR/SRs *bridge the gap* between LFSR/XORs [3] and (simple) LFSR/SRs [4]. Typically, convolved LFSR/SRs achieve low test lengths like LFSR/XORs and utilize low hardware like LFSR/SRs. The design procedure for convolved LFSR/SRs can also be used to determine other TPGs such as multiple LFSR/SRs and simple LFSR/SRs. The XOR network to form test signals makes convolved LFSR/SRs into very efficient TPGs. The efficiency of convolved LFSR/SRs in terms of both hardware and test time are demonstrated by the experiments on the combinational benchmark circuits.

We have derived the necessary and sufficient conditions for possible transformations among the following TPGs — LFSR/XORs, convolved LFSR/SRs, multiple

LFSR/SRs and simple LFSR/SRs. Based on empirical observations, we have constructed a lattice among these TPG designs in terms of hardware overhead and test time.

7.4.3 Bounds on Test Lengths

We have derived tight upper bounds on the sizes of pseudo-exhaustive test sets. Chapter 6 has derivations of a few important algebraic results on the set union and intersection operations between vector spaces. These algebraic results form the backbone for the derivation of the bounds on pseudo-exhaustive test lengths.

We have determined a few generic bounds on test lengths that are independent of the structural information about the circuit output cones. Generic bounds are used to characterize various classes of circuits. We have shown that any circuit with less than six outputs is a MTC circuit. This is a significant improvement over the existing result that any circuit with less than three outputs is a MTC circuit [27]. We have derived expressions for generic upper bounds on the pseudo-exhaustive test sets for $(n, m, 2)$ circuits. These expressions are based on either the number of inputs or the number of outputs to the circuits. Similar expressions are derived for (n, m, k) circuits based on their number of outputs and their maximum cone size.

We have also derived a few circuit-specific bounds utilizing the structural information about the circuit output cones. Tight upper bounds on the pseudo-exhaustive test sets have been derived for LFSR/XORs and LFSR/SRs. We have shown, both theoretically and experimentally, that our bounds are significantly better than those derived in [3] and [4]. We have developed algorithms of polynomial complexity to permute the circuit inputs in order to obtain good improvements on these bounds. Our bounds provide good estimates of pseudo-exhaustive test lengths and can be used as guiding factors in designing circuit-specific TPGs. The computed theoretical bounds comply well with the sizes of test sets generated by our TPG designs for the combinational benchmark circuits.

7.5 Future Extensions

Future work on the various aspects of pseudo-exhaustive testing can be carried out in various prospective directions. We shall describe a few extensions to our research work under each of the following categories: *partitioning*, *test pattern generation* and *bounds on test lengths.*

7.5.1 Partitioning

We have considered pseudo-exhaustive testing of partitioned circuits in a single test session. During the test session, each segmentation cell is configured as both a TPG cell and a SA cell. Allowing multiple test sessions may provide the flexibility of configuring a segmentation cell as either TPG cell or SA cell during a test session. Thus the hardware involved in the design of a segmentation cell can be reduced to some extent since the segmentation cell need not be configured necessarily as both TPG cell and SA cell during the same session. Also, the number of segmentation cells required for partitioning may be reduced by possible sharing of cells required at different locations during different test sessions. However, a price has to be paid in terms of an increase in the pseudo-exhaustive test time.

We have highlighted a few important characteristics of circuits with reconvergent fanouts providing justification for partitioning procedures of exponential complexity. We have proposed heuristic partitioning procedures of polynomial complexity to obtain good suboptimal solutions for circuits with and without non-reconvergent fanouts. It will be interesting to characterize and classify these fanout circuits based on the guarantee of obtaining optimal solutions using our heuristic procedures.

We have considered partitioning for cone size reduction and for achieving maximal test concurrency as a two-step process. A combined strategy can be investigated for partitioning circuits such that the two requirements — (1) the sizes of the output cones are restricted to some user-defined value, and (2) the circuit is maximal test concurrent in test mode — are met simultaneously rather than by a two-step process. The combined strategy may lead to a reduced number of segmentation cells compared to the two-step process.

We have extended the partitioning strategy for balanced sequential circuits [12]. The strategy can be extended in a similar fashion for unbalanced acyclic sequential circuits. Unbalanced acyclic circuits contain stuck-at faults that can be detected only by multiple sequences of test patterns. Hence it involves a detailed study of the TPG designs for generating pseudo-exhaustive test sets for unbalanced acyclic circuits prior to partitioning these circuits for reducing the test time.

Our strategies can be classified under hardware partitioning approaches for reducing the pseudo-exhaustive test time. Sensitized partitioning [41] is another approach where the functionality of the logic gates in the circuit are exploited for reducing the pseudo-exhaustive test time. In sensitized partitioning, some of the circuit inputs are held at constant values while the remaining inputs are applied an exhaustive set of patterns. Thus a portion of the given circuit is exhaustively tested by sensitizing the remaining portion of the circuit. A combined strategy involving both hardware and sensitized partitioning approaches can be investigated in order to reduce both the hardware overhead and the test time.

7.5.2 Test Pattern Generation

Our design procedure for convolved LFSR/SRs described in Chapter 5 considers only the minimum length requirement for the shift register segments. This consideration helps to some extent to reduce the number of feed-forward stages in the convolved LFSR/SR designs and thus reduces the hardware. The hardware overhead can be minimized by considering the following optimization criteria: (1) minimize the number of feed-forward stages, and (2) minimize the number of two-input XOR gates used for realizing the feed-forward stages. Both the criteria lead to optimization problems of exponential complexity and efficient procedures can be investigated for obtaining good suboptimal solutions.

TPG design for balanced sequential circuits is complicated since each pattern may need to be applied at the circuit inputs for a few successive clock cycles. In other words, TPG may have to hold each pattern for a few successive clock cycles prior to generating another new pattern. Alternatively, TPG designs can be investigated without holding each pattern for two or more clock cycles. The most

important criterion in the TPG design is to ensure the minimality of the pseudo-exhaustive test set in order to reduce the overall test application time. Generating a pseudo-exhaustive test set for unbalanced acyclic sequential circuits is even more complicated since the test set must contain multiple sequences of patterns in order to test the stuck-at faults that are not *single pattern testable* [12].

7.5.3 Bounds on Test Lengths

We have conjectured an important result in Chapter 6 that needs to be investigated and proven for any arbitrary circuit. The conjecture states a generic (circuit cone independent) upper bound on the number of circuit outputs that can be exhaustively tested with a given number of independent test signals (allowing linear combinations of these signals). It should be noted that the result is independent of the number of inputs to the circuit. The conjecture, if proven, will be a significant improvement over the proven upper bound given by Theorem 17.

The circuit cone dependent bound for LFSR/XORs given by Theorem 19 can be improved in the following manner. In the proof of the theorem, the number of prohibited residues is overestimated due to multiple countings resulting in a loose upper bound. The upper bound can be improved by determining a minimal set of prohibited residues. The circuit inputs can be permuted to further improve the upper bound. Similarly, the cone dependent bound for LFSR/SRs given by Theorem 24 needs to be improved by determining a minimal set of inapplicable primitive polynomials as described in the proof of the theorem. An input permutation algorithm to achieve the best improvement on the LFSR/SR bound, similar to the permutation algorithm described for LFSR/XORs, needs to be investigated.

Reference List

[1] M. S. Abadir and M. A. Breuer. A Knowledge-Based System for Designing Testable VLSI Chips. *IEEE Design & Test*, 2:56–68, August 1985.

[2] M. Abramovici, M. A. Breuer, and A. D. Friedman. *Digital Systems Testing and Testable Design.* IEEE Computer Science Press, 1992.

[3] S. B Akers. On the Use of Linear Sums in Exhaustive Testing. In *Proc. 15th Int'l. Symp. on Fault-Tolerant Computing*, pages 148–153, June 1985.

[4] Z. Barzilai, D. Coppersmith, and A. Rosenberg. Exhaustive Bit Pattern Generation in Discontiguous Positions with Applications to VLSI Testing. *IEEE Trans. on Computers*, C-32(2):190–194, February 1983.

[5] Z. Barzilai, J. Savir, G. Markowsky, and M. G. Smith. The Weighted Syndrome Sums Approach to VLSI Testing. *IEEE Trans. on Computers*, C-30(12):996–1000, December 1981.

[6] S. N. Bhatt, F. R. K. Chung, and A. L. Rosenburg. Partitioning Circuits for Improved Testability. In *Proc. 4th MIT Conf. on Advanced Research in VLSI*, pages 91–106, April 1986.

[7] F. Brglez and H. Fujiwara. A Neutral Netlist of Ten Combinational Benchmark Circuits and a Target Translator in FORTRAN. In *Proc. Int'l. Symp. on Circuits and Systems*, pages 663–698, June 1985.

[8] C. H. Chen. BISTSYN - A Built-In Self-Test Synthesizer. In *Proc. Int'l Conf. on Computer Aided Design*, pages 240–243, 1991.

[9] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* The MIT Press and McGraw-Hill Book Company, 1990.

[10] N. Deo. *Graph Theory with Applications to Engineering and Computer Science.* Prentice Hall, Inc., 1974.

[11] S. W. Golomb. *Shift Register Sequences.* Aegean Park Press, 1982.

[12] R. Gupta. Advanced Serial Scan Design for Testability. Technical Report CENG 91-10, University of Southern California, 1991.

[13] R. Gupta, R. Srinivasan, and M. A. Breuer. Reorganizing Circuits to Aid Testability. *IEEE Design & Test*, 8(3):49–57, September 1991.

[14] Rajesh Gupta, Rajiv Gupta, and M. A. Breuer. The BALLAST Methodology for Structured Partial Scan Design. *IEEE Trans. on Computers*, C-39(4):538–543, April 1990.

[15] Rajiv Gupta, W. H. Cheng, Rajesh Gupta, I. Hardonag, and M. A. Breuer. An Object-Oriented VLSI CAD Framework: A Case-Study in Rapid Prototyping. *IEEE Computer*, 22(5):28–37, May 1989.

[16] S. Hellebrand and H-J. Wunderlich. Tools and Devices Supporting the Pseudo-Exhaustive Test. In *Proc. 1st European Design Automation Conf.*, pages 13–17, March 1990.

[17] S. Hellebrand, H-J. Wunderlich, and O. F. Haberl. Generating Pseudo-Exhaustive Vectors for External Testing. In *Proc. Int'l Test Conf.*, pages 670–679, September 1990.

[18] I. N. Herstein. *Topics in Algebra.* Xerox College Publishing, 1975.

[19] W. B. Jone. Methodology of Partitioning and Exhaustive Test Pattern Generation for Built-in Self-Testing of VLSI Circuits. Technical Report CES-88-xx, Case Western Reserve University, January 1988.

[20] W. B. Jone and C. A. Papachristou. A Coordinated Approach to Partitioning and Test Pattern Generation for Pseudoexhaustive Testing. In *Proc. Design Automation Conf.*, pages 525–530, June 1989.

[21] D. Kagaris, F. Makedon, and S. Tragoudas. On Minimizing Hardware Overhead for Pseudo-Exhaustive Circuit Testability. In *Proc. Int'l Conf. on Computer Design*, pages 358–364, 1992.

[22] D. Kagaris and S. Tragoudas. Cost-Effective LFSR Synthesis for Optimal Pseudo-Exhaustive BIST Test Sets. *IEEE Trans. on VLSI Systems*, 1(4):526–536, December 1993.

[23] B. Konemann, J. Mucha, and G. Zwiehoff. Built-In Logic Block Observation Technique. In *Proc. Int'l Test Conf.*, pages 37–41, October 1979.

[24] A. Lempel and M. Cohn. Design of Universal Test Sequences for VLSI. *IEEE Trans. on Information Theory*, IT-31(1):10–17, January 1985.

[25] S. P. Lin. A Design System to Support Built In Self Test of VLSI Circuits Using BILBO Oriented Test Methodologies. Technical Report CENG 94-11, University of Southern California, 1994.

[26] C. M. Maunder and R. E. Tulloss. *The Standard Test Access Port and Boundary-Scan Architecture.* Addison-Wesley Publishing Company, 1991.

[27] E. J. McCluskey. Verification Testing — A Pseudoexhaustive Test Technique. *IEEE Trans. on Computers*, C-33(6):541–546, June 1984.

[28] E. J. McCluskey and S. Bozorgui-Nesbat. Design for Autonomous Test. *IEEE Trans. on Computers*, C-30(11):866–875, November 1981.

[29] D. Mukherjee. An Integrated Test Controller Synthesis System. Technical Report CENG 94-21, University of Southern California, 1994.

[30] K. P. Parker. *Integrating Design and Test: Using CAE Tools for ATE Programming.* IEEE Computer Society Press, 1987.

[31] I. Parulkar, M. A. Breuer, and C. A. Njinda. Extraction of a High-Level Structural Representation from Circuit Descriptions with Applications to DFT/BIST. In *Proc. Design Automation Conf.*, pages 345–350, June 1994.

[32] M. W. Roberts and P. K. Lala. An Algorithm for the Partitioning of Logic Circuits. *IEE Proc.*, 131(4):113–118, July 1984.

[33] R. Srinivasan, S. K. Gupta, and M. A. Breuer. An Efficient Partitioning Strategy for Pseudo-Exhaustive Testing. Technical Report CENG 93-08, University of Southern California, 1993.

[34] R. Srinivasan, S. K. Gupta, and M. A. Breuer. An Efficient Partitioning Strategy for Pseudo-Exhaustive Testing. In *Proc. Design Automation Conf.*, pages 242–248, June 1993.

[35] R. Srinivasan, S. K. Gupta, and M. A. Breuer. Novel Test Patttern Generators for Pseudo-Exhaustive Testing. In *Proc. Int'l Test Conf.*, pages 1041–1050, October 1993.

[36] R. Srinivasan, C. A. Njinda, and M. A. Breuer. A Partitioning Method for Achieving Maximal Test Concurrency in Pseudo-Exhaustive Testing. In *Proc. of IEEE VLSI Test Symp.*, pages 34–39, April 1991.

[37] D. T. Tang and C. L. Chen. Logic Test Pattern Generation using Linear Codes. *IEEE Trans. on Computers*, C-33(9):845–850, September 1984.

[38] D. T. Tang and L. S. Woo. Exhaustive Test Pattern Generation with Constant Weight Vectors. *IEEE Trans. on Computers*, C-32(12):1145–1150, December 1983.

[39] J. G. Udell. Reconfigurable Hardware for Pseudo-Exhaustive Test. In *Proc. Int'l Test Conf.*, pages 522–530, September 1988.

[40] J. G. Udell. Efficient Segmentation for Pseudo-Exhaustive BIST. In *Proc. Custom Integrated Circuits Conf.*, pages 13.6.1–13.6.5, May 1992.

[41] J. G. Udell and E. J. McCluskey. Efficient Circuit Segmentation for Pseudo-Exhaustive Test. In *Proc. Int'l Conf. on Computer Aided Design*, pages 148–151, November 1987.

[42] L. T. Wang and E. J. McCluskey. Condensed Linear Feedback Shift Register (LFSR) Testing — A Pseudoexhaustive Test Technique. *IEEE Trans. on Computers*, C-35(4):367–370, April 1986.

[43] L. T. Wang and E. J. McCluskey. Circuits for Pseudoexhaustive Test Pattern Generation. *IEEE Trans. on Computer-Aided Design*, 7(10):1068–1080, October 1988.