EMBEDDED TEST STRATEGIES FOR SYSTEM-ON-A-CHIP DESIGNS

by

Ronald Wai Kit Fung

B.A.Sc., The University of British Columbia, 1999

A thesis submitted in partial fulfillment of
the requirements for the degree of

Master of Applied Science

in

The Faculty of Graduate Studies

Department of Electrical and Computer Engineering

We accept this thesis as conforming to the required standard

The University of British Columbia

April 2003

Department of ELECTRICAL AND COMPUTER ENGINEERING

The University of British Columbia
Vancouver, Canada

Date APR. 22, 2003

ABSTRACT

System-on-a-chip (SoC) with reuse of intellectual property (IP) is gaining acceptance as the preferred style for integrated circuit (IC) designs. This paradigm shift poses great challenges to the overall design and test methodologies. To support SoC design and test, it is important to develop a corresponding set of Semiconductor Infrastructure IP (SI^2P), which includes all components surrounding an IP core to facilitate system integration, timing synchronization, and test efforts. This thesis focuses on the SI^2P needed for SoC test.

First, a relationship is established between stuck-at (DC) and transition (AC) fault detection when applying a set of test vectors to a given design. To exploit this relationship, a new test pattern generation flow is proposed to maximize the DC fault coverage level with test patterns targeted at AC faults. The resulting vector set is a combination of pseudo-random test patterns and deterministic "top-up" vectors. The fault coverage of this approach is competitive with that achievable by an automatic test pattern generation (ATPG) tool.

A hardware implementation of on-chip test pattern generation as part of a logic built-in self-test (logic BIST) solution is described. A matrix-based algorithm for constructing deterministic pattern generator circuits based on linear feedback shift registers (LFSR's) is presented. It is found that the resulting area overhead of the deterministic pattern generator is significant relative to the IP core under test and is a function of deterministic test pattern count. Additional SI^2P components required for this embedded testing approach are also described.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACRONYMS

ASIC	Application Specific Integrated Circuit
ATE	Automated Test Equipment
ATPG	Automatic Test Pattern Generation
BDR	Boundary Data Register
BIST	Built-in Self-Test
CAD	Computer Aided Design
CDR	Core Data Register
CMOS	Complementary Metal-Oxide-Silicon
CRC	Cyclic Redundancy Check
DFT	Design for Test
DSM	Deep Sub-Micron
FF	Flip-Flop
FSM	Finite State Machine
IC	Integrated Circuit
IEEE	Institute of Electrical and Electronics Engineers
I/O	Input/Output
IP	Intellectual Property
ITRS	International Technology Roadmap of Semiconductors
JTAG	Joint Test Action Group
LFSR	Linear Feedback Shift Register
MISR	Multiple-Input Signature Recognizer
MUX	Multiplexer
NIMA	Novel Indirect and Modular Architecture for TAM
PI	Primary input
PO	Primary output
PPI	Pseudo-primary input
PPO	Pseudo-primary output
PRPG	Pseudo-Random Pattern Generation
ROM	Read-Only Memory
RTL	Register Transfer Level
SI^2P	Semiconductor Infrastructure IP
SISR	Single-Input Signature Recognizer
SoC	System on a Chip
TAM	Test Access Mechanism
TAP	Test Access Port
VHDL	VHSIC (Very High Speed Integrated Circuits) Hardware Description Language
VLSI	Very Large Scale Integration
WIR	Wrapper Instruction Register

ACKNOWLEDGEMENTS

It has been an extremely pleasant experience to study for this Masters degree with Dr. Resve Saleh at University of British Columbia. Throughout the stay with the SoC Research Group, Dr. Resve Saleh has provided continuous support and valuable advice vital for the completion of this research. His efforts and involvement in this research are greatly appreciated. Expert advice has also been received from Dr. André Ivanov and Dr. Steve Wilton. Their useful comments helped to focus the research direction.

Throughout this research, a team of brilliant researchers has been remarkable in offering insightful opinions and generous assistance. This unforgettable team is sincerely honoured and treasured – Gary, Louis, Marwa, and Victor. Without their continuous motivation and support, the completion of this research would not be possible. Moreover, the entire SoC Research Group at UBC is deeply cherished for their constant sharing of wisdom and sense of humour.

Last but not least, this thesis is dedicated to my family, Eric, Nancy, Edmond, and Mancy who have been extraordinarily supportive throughout my years at UBC. Their encouragement is one of the strongest motivations for the pursuit of this Masters degree.

This research is supported by PMC-Sierra, Canadian Microelectronic Corporation, and Advanced Systems Institute.

CHAPTER 1 INTRODUCTION

1.1 Motivation

System-on-a-Chip (SoC) is gaining acceptance as the preferred style for integrated circuit (IC) designs. It serves as an effective methodology to close the gap between engineering productivity and the IC design complexity enabled by technology advancements predicted by Moore's Law [32].

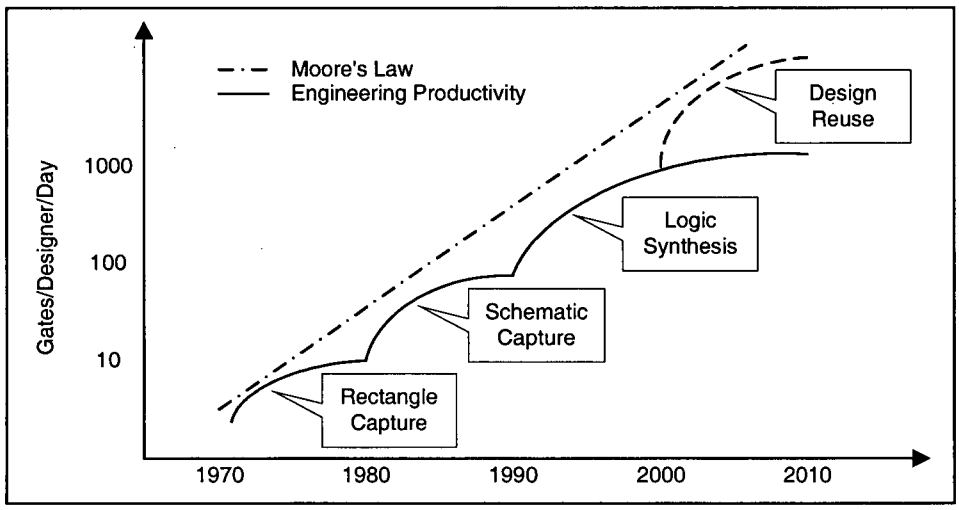


Figure 1. Productivity gap

As illustrated in Figure 1, engineering productivity in IC design always lags behind the achievable IC complexity forecasted by Moore's Law. This gap widens gradually with technology advancements. It is only with innovations in design methodologies that this gap can be reduced momentarily. At the present time, design reuse is the most promising methodology to close this gap between engineering productivity and technology advancements. Design reuse is at the heart of the introduction of intellectual property (IP) blocks and SoC design. Without such a design methodology, closing the gap

between engineering productivity and technology advancements becomes difficult and thus potential capabilities of semiconductors cannot be realized.

Aside from rigorous development of reusable IP cores, other issues such as system integration of IP, the inter-block communication and test are essential portions of a SoC design. The integration of IP cores in a SoC requires the management of the timing synchronization of cores that may operate in different clock domains. Standard bus architectures such as AMBA™ [1] and CoreConnect™ [5] have been defined to allow each of integration of processor, memory and interface blocks. Test wrappers such as IEEE P1500 [24][31] have been developed to manage IP test issues. These supporting components can be viewed collectively as Semiconductor Infrastructure IP (SI^2P) [11] to enable ease of integration. Without such interoperability, SoC designs are too complex to design and test. Consequently, SoC designs should, in fact, be defined as the integration of IP cores and SI^2P; lacking either one will limit industry adoption of the SoC design paradigm.

As an example that is pertinent to this thesis, Figure 2 shows the role that SI^2P plays in SoC testing. Each core is encased by a P1500 wrapper to provide a unified interface for test control purposes. The wrapper control signals can be generated by a user-defined test controller which is enabled by external sources. In addition, a user-defined parallel test access mechanism (TAM) can be implemented for speedy test data transportation to/from individual IP cores. All of these items comprise SI^2P that support the actual IP cores in a SoC design.

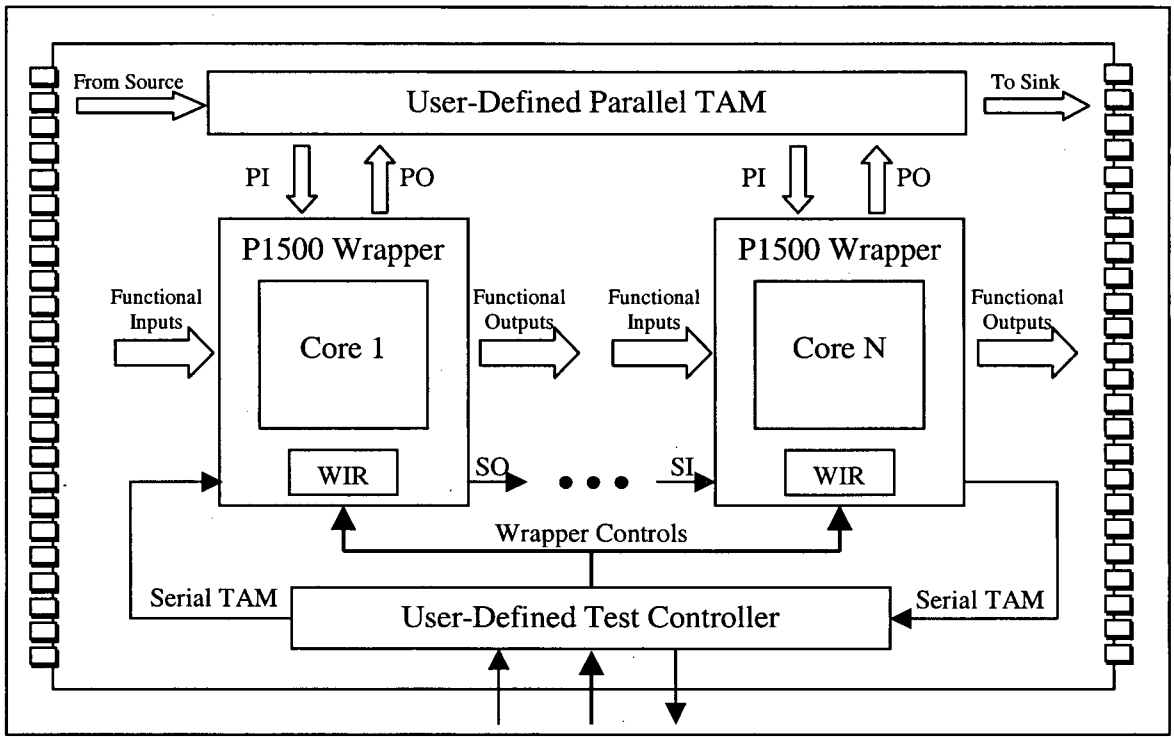


Figure 2. SI^2P for SoC test

Design-for-Test (DFT) for SoC design is a major concern for semiconductor vendors and customers [6]. This is due to increasing design complexity enabled by the reuse of intellectual property (IP) cores and the increasing cost of automated test equipments (ATE's) [4]. As SoC integrated circuits (IC's) are designed, IP cores are often buried deep in the design hierarchy that causes accessibility issues since the terminals of the IP cores may not be controllable nor observable directly by ATE's. Furthermore, third-party IP cores are often purchased and integrated by a semiconductor design house to assemble the final SoC design. With 80-90% of the IP market occupied by hard IP's provided in the layout form [25], IP cores bundled with embedded test solutions become attractive. This advocates extensive use of logic built-in self-test (logic BIST). Also, this corresponds to the future projections of the Test and Test Equipment Working Group of International Technology Roadmap of Semiconductors 2001 (ITRS 2001) which states

"Logic BIST technique must evolve to support new fault models, fault analysis, and deterministic test" [13]. Thus, the importance of logic BIST is apparent and is one of the major motivations of this research.

There are several possible options for IP block testing and each differs in how test stimuli are delivered to individual IP core. Generally, these options vary from fully off-chip to fully on-chip test pattern sources. A mix of the on-chip and off-chip options constitutes a third option which may be deemed viable in some circumstances. This research aims at minimizing off-chip intervention during testing and thus it pursues after a completely on-chip solution.

1.2 Research Goals

There are three objectives in this research:

1. To establish a relationship between stuck-at (DC) and transition (AC) fault detection. To design a logic BIST scheme for an IP core to detect as many easily detectable DC and AC faults as possible with least amount of resources in terms of hardware and design effort.
2. To develop a test generation flow for combined DC and AC fault detection using pseudo-random pattern generation (PRPG) and automatic test pattern generated (ATPG) deterministic "top-up" patterns. Such an algorithm should exploit the connection between DC and AC faults to select test patterns wisely for both stuck-at and transition fault testing. The achievable fault coverage should compete with commercial tools.

3. To develop an on-chip deterministic pattern generator which specially targets the faults not detected by pseudo-random test patterns. Such hardware should be generated automatically given a set of deterministic patterns.
4. To develop SI^2P concepts further with test harnesses flexible enough to enable communication with most test access mechanisms (TAM's). The overall approach should be validated by design, fabrication and test of an IP core with SI^2P support.

1.3 Thesis Organization

Chapter 2 of this thesis gives an overview of terminologies and basics on testing. It introduces the idea of taking advantage of the commonalities between AC and DC fault detection. Based on the AC and DC fault commonalities identified, it proposes a test pattern generation flow that combines pseudo-random and deterministic pattern generation methods. It also presents the results of the test pattern generation flow by reporting the achievable fault coverage against that of a commercial ATPG tool.

Chapter 3 describes the hardware design necessary for implementing the results of the test pattern generation flow as a logic BIST solution. It also identifies possible methods for on-chip deterministic test pattern generation and proposes an algorithm to design a linear feedback shift register (LFSR) based circuit for deterministic test pattern generation. Then it reports the area overhead incurred by the logic BIST circuitry.

Chapter 4 presents the SI^2P-related components with an introduction of P1500 standard under development by IEEE for communication with the TAM. An example of an HC11 core wrapped with P1500 and supplied with an AC/DC logic BIST scheme is described. The design is fabricated in a 0.18μm CMOS technology.

Finally, Chapter 5 concludes with a brief summary, possible future research direction, and the contributions of this project to SoC research.

CHAPTER 2 ALGORITHM FOR COMBINED AC/DC ON-CHIP TESTING

This chapter presents some principles of testing upon which this research is based. First, it explains the differences and similarities between combinational and sequential circuit testing. With this knowledge of the target circuits, two fault models, namely stuck-at (DC) and transition (AC) faults, are examined. Subsequently, the similarities between the DC and AC fault testing are identified and how these connections can be leveraged for combined DC and AC testing are explained. Further, it describes how the quality of test is measured by fault coverage evaluation.

Once the connections between the DC and AC fault testing are established, an overall test pattern generation flow is described. Then, it presents the significant results of this pattern generation flow with a summary of the achievable fault coverage readings compared to a commercial ATPG tool. These fault coverage results are obtained from the TetraMAX™ ATPG and fault simulation tool by Synopsys®.

The results are collected from ITC'99 IP cores [30] to evaluate the proposed method on standard benchmarks. Further, IP cores designed in-house at UBC for the development of a simple network processor have also been used in this evaluation. Each of these IP cores has a single scan chain and is registered at the input and output ports for the reasons outlined later in Section 4.1.

2.1 Circuit Classifications

This section examines two types of digital circuits, which serve as the test targets. These two types of circuits are combinational and sequential circuits.

2.1.1 Testing Combinational Circuits

Combinational circuits are the simplest type of digital circuits realized at the gate level. These circuits consist of elementary logic operators such as buffer, NOT, AND, NAND, OR, NOR, XOR, XNOR, and multiplexer (MUX). These logic gates are memoryless: the outputs of these elements only depend on the logical operations with the current inputs. Any logic equation can be implemented by connecting up these elementary gates accordingly.

During fabrication, a defect may be introduced in the circuit, as denoted by X in Figure 3, due to wafer defects or process variations. As a result, all parts must be tested after manufacture to ensure their proper operation. When combinational circuits are tested, test stimuli are applied to the primary inputs (PI's) and the test responses are observed at the primary outputs (PO's). If the responses at the outputs are not as expected, it implies that at least one defect exists in the fabricated circuit. Figure 3 shows a simple combinational circuit with PI's and PO's. A manufacturing defect located at the indicated connection can be detected by controlling the inputs of the AND gate and observing the signal behaviour at PO_0. Defects can be modeled as faults in a variety of ways as explained in Section 2.2.

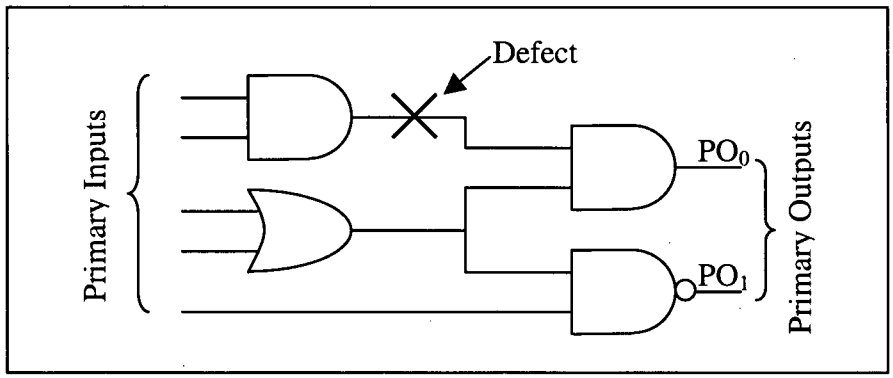


Figure 3. Combinational circuit

Combinational circuits are considered to be easy to test if all internal nodes can be directly controlled by the inputs and directly observed at the outputs by the application of appropriate test patterns.

2.1.2 Testing Sequential Circuits

Sequential circuits are identical to combinational circuits except for the inclusion of memory elements such as flip-flops (FF's). FF's are usually driven by a clock, which serves as the signal for controlling when the outputs from combinational circuits are to be stored as FF contents. The outputs of sequential circuits depend on both the current PI's and the current states of the FF's. Figure 4 shows a sequential circuit with the PI's, PO's, pseudo-primary inputs (PPI's) and pseudo-primary outputs (PPO's).

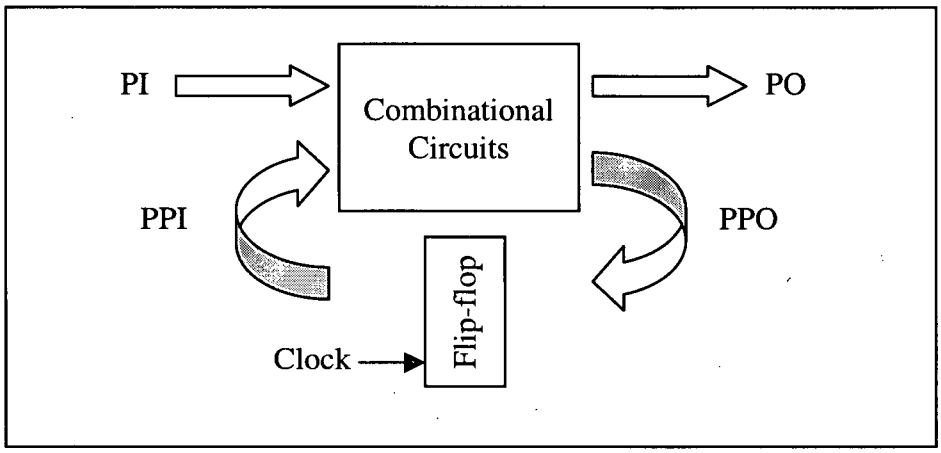


Figure 4. Sequential circuit

PPI's and PPO's are neither easily controllable nor observable externally because they may be buried deep inside the circuit. Due to this reduced controllability and observability, sequential circuits are difficult to test since the PO's are dependent upon the previous inputs encoded in the FF states. Unfortunately, this type of circuits is always encountered in logic designs since finite state machines (FSM's) are usually implemented as sequential logic systems.

In order to efficiently test these circuits, a standard technique to greatly increase controllability and observability is employed by introducing a test mode in all FF's. During the test mode, the output of a FF is functionally connected to the input of another FF in the circuit. All FF's connected in this way form a long chain to compose a shift register. Logic values can be shifted into the FF's serially to set their outputs. These values are used as inputs to the combinational circuits between the FF's. As a result, the sequential test problem is converted into a combinational test problem. This well-known technique is called scan design [33][3][23].

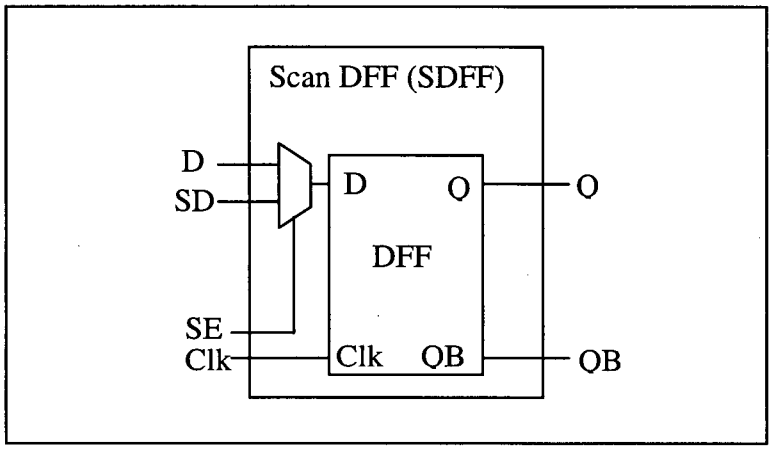


Figure 5. Scan flip-flop

Figure 5 shows the design of a sample DFF suitable for scan design. A scan FF is an ordinary DFF with a MUX inserted in front of the data input. The signal to the DFF data input is selectable with the scan enable (SE) pin. In the figure, when SE=0, the D input of the SDFF is routed as the input for the DFF. When SE=1, the SD input of the SDFF is selected as the input of the DFF. The SD input is connected to the output of the previous SDFF in the scan chain.

Long scan chains hinder manufacturing test by consuming valuable test time on ATE's. Multiple scan chains can exist in a sequential circuit to speed up shifting if one scan chain should become too long.

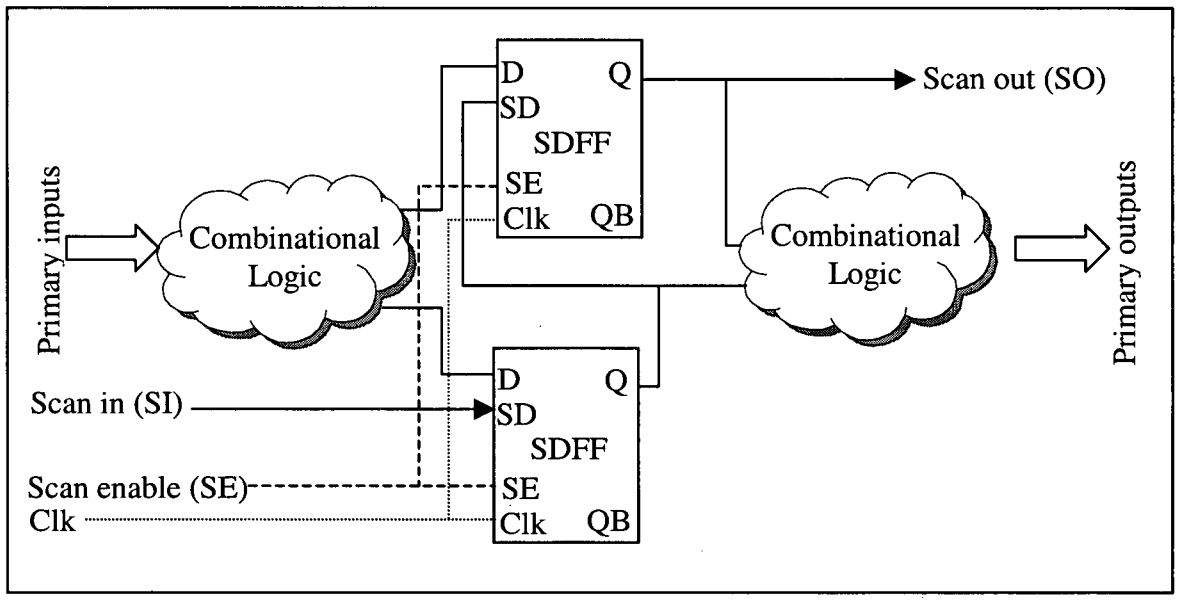


Figure 6. Sequential circuit with scan design

With the aid of SDFF design, Figure 6 shows a sequential circuit with scan insertion. When testing, the desired FF states would be shifted into the FF's in the test mode (SE=1) and the PI's would be set to appropriate values. When the system clock is applied in the normal mode (SE=0), the PPO's are captured in the FF's and PO's can be examined externally. The PPO's are examined by observing the scan out (SO) pin when shifting the scan chain as a shift register in test mode (SE=1). Note that the next desired FF states could be shifted in while the current states of the FF's are being shifted out of the scan chain.

2.2 Fault Models

Proper fault modeling is essential to high-quality IC testing. At an abstract level, it is a representation of the defects that characterize the unintended differences between the

fabricated hardware and its intended design [3][14]. Many fault models exist and this research focuses on two of them: stuck-at (DC) and transition (AC) fault models.

2.2.1 Stuck-At (DC) Fault Model

If a manufacturing defect causes an internal node to be inadvertently held to logic 1 or 0, it is referred to as a stuck-at fault. It is also referred to as a DC fault due to the static nature of the fault. This fault is modeled by assigning a constant logic value (0 or 1) to an interconnection section within a gate-level netlist. Such interconnection could be an input or output of a logic gate or a flip-flop. Each interconnection is modeled by two variants of this fault. Stuck-at-1 (s-a-1) fault is modeled by assigning a constant logic 1 to an interconnection section. Conversely, an interconnection with a constant logic 0 assignment is called a stuck-at-0 (s-a-0) fault. A combinational circuit with a s-a-0 fault, denoted by X, is shown in Figure 7.

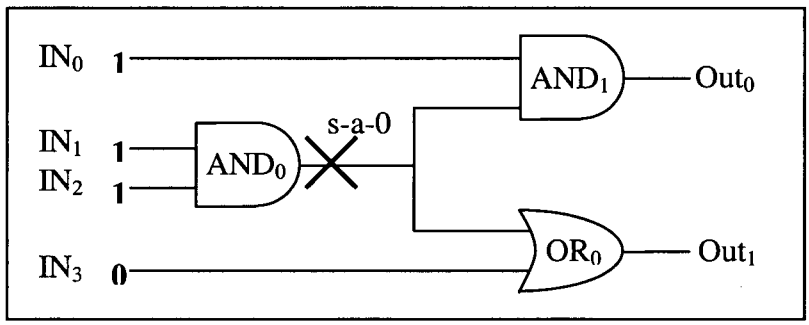


Figure 7. Stuck-at fault

The role of stuck-at faults is to model the defects which cause interconnections not to toggle irrespective of the inputs of the gates or FF's driving them. A stuck-at fault test must first activate the fault and then propagate the result to the output. For example, if fault site X has a s-a-0 fault, the output of AND_0 gate should be driven to the opposite

logic state of 1 by setting IN_1 and IN_2 to 1. In order for the response at fault site X to be observable at OUT_0 and/or OUT_1, offpaths IN_0 and IN_3 must be set to non-controlling logic values of 1 and 0 respectively to allow the propagation of output from AND_0 to OUT_0 and OUT_1.

2.2.2 Transition (AC) Fault Model

Another type of fault occurs when a signal attempts to make a logic transition but is very slow in doing so. This is referred to as a delay fault which is also called an AC fault since it is dynamic in nature. There are many different delay fault models, namely transition, gate, line, path, and segment delay fault models [19]. This research focuses on the transition fault model and it is referred to as the AC fault model in the context of this thesis. There are two types of AC transition faults for each gate: slow-to-rise (s-t-r) and slow-to-fall (s-t-f). To initiate a signal transition, two patterns (P_1, P_2) are necessary. For a s-t-f fault, P_1 is responsible for configuring the target gate for a s-a-0 fault, thus forcing the gate output to be 1. Then P_2, targeting the gate for s-a-1 fault, is applied to cause a signal transition to be observed at the PO or PPO after some specific delay. The opposite is done to detect for s-t-r faults. The purpose of this fault model is to model a defectively slow switching gate that potentially causes failure of a circuit path to meet specific timing requirements. AC faults are important to test since they validate that the design meets the timing specification. With higher and higher speeds, on-chip fault detection may be the only viable approach since the ATE's are having problems keeping pace with chip speeds.

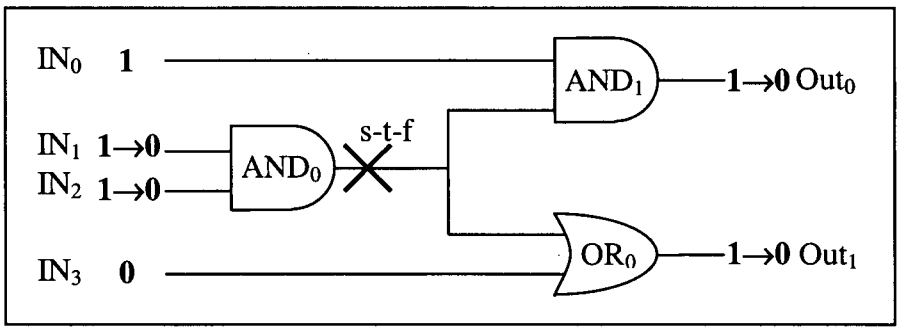


Figure 8. Transition fault

In Figure 8, the s-t-f fault at AND_0 gate can be tested by applying a s-a-0 pattern as P_1. As a result, the output of the gate is driven to 1 with IN_1 and IN_2 set to 1. Then P_2, targeting for the s-a-1 fault at AND_0 gate, is applied to initiate a signal transition by setting IN_1, IN_2 or both to 0. For the entire duration of above testing, IN_0 and IN_3 should be kept at the non-controlling states of 1 and 0 respectively in order for the transition at output of AND_0 to be observable at OUT_0 and/or OUT_1.

2.3 Connection between DC and AC Fault Testing

From the descriptions in the previous sections, one can see that the fundamental concepts of DC and AC fault testing are related. As mentioned in Section 2.2.2, during AC fault testing, all gate outputs are driven to opposite states to stimulate rising and falling signal transitions. Such transitions are expected to complete within a specific time limit which is usually dictated by the system clock period of the sequential circuits under test. If any transition does not complete within the specified amount of time, it is considered as an AC fault.

When a circuit is analyzed for AC faults, a list of all possible AC fault sites is constructed. These AC fault sites are located at logic gates and each logic gate generates two faults, namely slow-to-rise (s-t-r) and slow-to-fall (s-t-f). Such a list is called the AC fault list and a complete AC fault list is usually composed of two times the number of gates [3]. This AC fault list is used by automatic test pattern generation (ATPG) and fault simulation tools. The former is responsible for crafting the necessary test patterns to detect faults specified in a fault list; the latter is responsible for determining the number of faults within a fault list that a test pattern set capable of detecting. When considering the AC fault list, each AC fault is composed of two DC faults. Each s-t-r fault is composed of a s-a-1 followed by a s-a-0 fault at the fault site as explained in Section 2.2.2. Similarly, each s-t-f fault is composed of a s-a-0 and s-a-1 sequence. AC fault testing is simply a more stringent test, in terms of timing, than its DC fault testing counterpart. In other words, an AC fault is derived from a pair of DC faults. This relationship is exploited in this research work. The claim is that patterns targeting for AC faults also provide a certain degree of DC fault coverage.

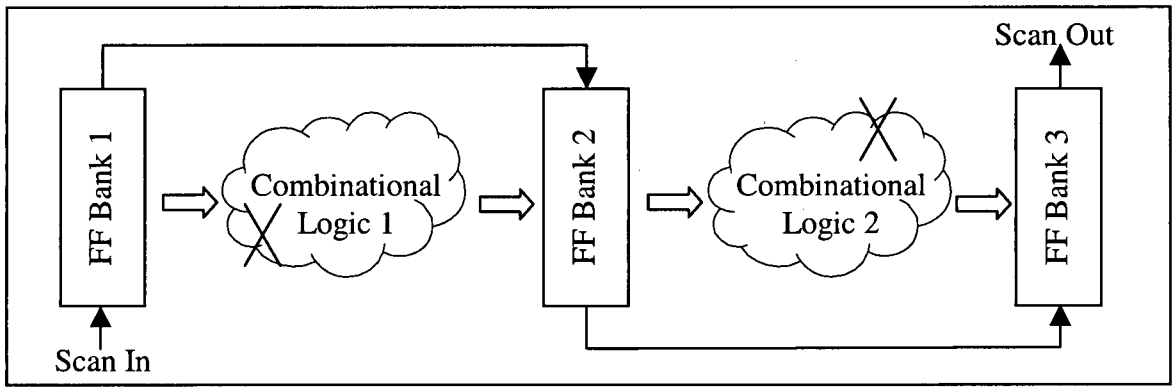


Figure 9. AC test patterns for DC fault detection

During AC fault testing of the circuit in Figure 9, a test pattern is shifted into the scan chain of an IP core as in any scan-based testing. After a desired pattern is shifted into

place, the SDFF's in the scan chain are switched to normal mode such that they are ready to capture the outputs from the combinational blocks. This is followed by two consecutive at-speed system clock cycles, which is referred to as *double-clocking* in this thesis. Then, the scan chain is filled with responses from the combinational circuit blocks, which are waiting to be shifted out through the scan chain. Correctness of these responses implies the non-existence of the AC faults that this test particular test pattern is capable of detecting.

As mentioned earlier, two consecutive at-speed system clock cycles are to be applied to the IP core. This is due to the necessity to initiate signal transitions through the combination blocks. Just before the application of the first system clock cycle in normal mode, outputs from the combinational blocks are waiting at the inputs of the FF's. When the first system clock in normal mode is applied, these outputs are stored in the FF's. These new FF contents are likely to be in the opposite states of what the FF's used to have just before the first system clock hits in normal mode. Such changes in FF contents are the sources of signal transitions necessary at the inputs of the combination blocks for AC fault detection. Then the second system clock in normal mode simply captures the output signal transitions at the combinational blocks within a functional period of the circuit.

In cases when DC faults exist in the combinational blocks as denoted by X in Figure 9, their effects are likely to emerge as erroneous responses from the combinational blocks, which are captured by the first system clock in normal mode. This directly causes inputs

to the downstream combinational blocks to be different from the expected values. As a result, the second set of responses is likely to be incorrect. These mismatches are captured by the second system clock cycle in normal mode and are shifted out for examination. In effect, AC test patterns do not immediately report DC faults as they occur. Instead, AC test patterns rely on the DC fault effects to propagate one more level downstream before they are captured for examination.

In cases when the DC faults only occur in combinational logic block 2, they prohibit certain signal transition occurrences at the combinational logic block outputs. This is true regardless of the combinational logic block 2 input changes caused by the first system clock cycle in normal mode. Effectively, AC faults in combinational logic block 2 are captured with the second system clock cycle in normal mode and are shifted out for examination.

Since two consecutive system clock cycles are supplied in normal mode, it is possible that one DC fault in the first system cycle in normal mode masks other DC faults in the second clock cycle in normal mode, or vice versa. This situation sacrifices DC fault coverage and appears to be a flaw to the approach. However, a similar situation also appears in the case of multiple stuck-at faults. It is possible that one fault cancels out the effect of another fault to produce correct outputs. These cases are usually not considered in practice due to exponential increase in complexity of test pattern generation [3]. A similar argument can be made for AC vector detection of DC faults.

This approach also promotes reduction of total test time since the AC fault test time targets not only AC faults but also DC faults. Since the first part of the logic BIST operation involves pseudo-random test patterns, these patterns normally account for most of the total pattern count. It does not require two separate passes, one for AC faults and another for DC faults.

2.4 Fault Coverage

With the DC and AC fault models established, a useful metric to quantify the test performance is required. Fault coverage is a measure of the number of tested faults versus all the possible faults in a circuit. Following the notation in [28], fault coverage is defined to be:

$$\frac{Detected\ faults + (Possibly\ detected\ faults \times Possibly\ detected\ credit)}{All\ faults} \times 100\% \qquad (1)$$

The *Detected fault* category is made up of two classes:

- Faults detected by simulation are determined by generating patterns and simulating to verify that the patterns result in the faults being detected.
- Faults detected by implication do not have to be detected by specific patterns, because these faults result from shifting scan chains. The faults in this class usually occur along the scan chain paths and include clock pins and scan-data inputs and outputs of the scan cells.

The *Possibly detected* fault category is made up of two classes:

- ATPG possibly detected class contains faults for which the difference between the good machine and the faulty machine results in a simulated output of X rather

than 1 or 0. Analysis proves that the fault cannot be definitely detected under current ATPG conditions, only possibly detected. For example, with faults on the enable line of an internal tri-state driver, the off state of the enable can only be possibly detected because the resulting Z state on the data bus quickly becomes an X state as it is captured into a scan cell or passes through other internal logic.

- Not analyzed – possibly detected class also contains faults for which the difference between the good machine and the faulty machine results in a simulated output of X rather than 1 or 0. However, the analysis to prove that the fault cannot be definitely detected using current ATPG conditions is not conclusive. Like the previous class, the simulation cannot tell the expected output of the faulty machine.

Partial credit is given for possibly detected faults in the test coverage calculation. This partial credit is by default 50 percent and is variable [28]. This credit is awarded to faults belonging to the possibly detected category because the ATPG tool cannot definitively dismiss the possibility of such faults not being detectable in the implemented design on silicon.

Fault coverage basically is a representation of the test quality of a pattern set in capturing the faults in the fault list. It also serves as a criterion for deciding whether further test pattern development is required. Often, certain minimum fault coverage must be achieved before a design is sent out for fabrication in order to guarantee quality and robustness of the resulting designs.

As described to in Section 2.3, DC fault coverage is achieved with application of AC test patterns. An experiment has been conducted to investigate such effects. Pseudo-random patterns are generated and fed to the scan chain of an IP core. Such patterns are *double-clocked* by a scheme, also referred to as the broadside method [27], where two consecutive at-speed system clocks are applied to the IP core in normal mode. Then the content of the scan chain is shifted out. Such stimuli and responses, along with a DC fault list containing all possible DC fault sites in the IP core, are used as inputs to a fault simulation tool for DC fault coverage assessment. In Figure 10, it is shown that a high DC fault coverage can be obtained in this manner. The DC fault coverage versus AC test pattern count can reach above 90% for this example. The attempt is to claim that although the DC fault coverage achievable by AC test patterns varies between IP cores, some DC fault coverage can be derived from AC fault testing.

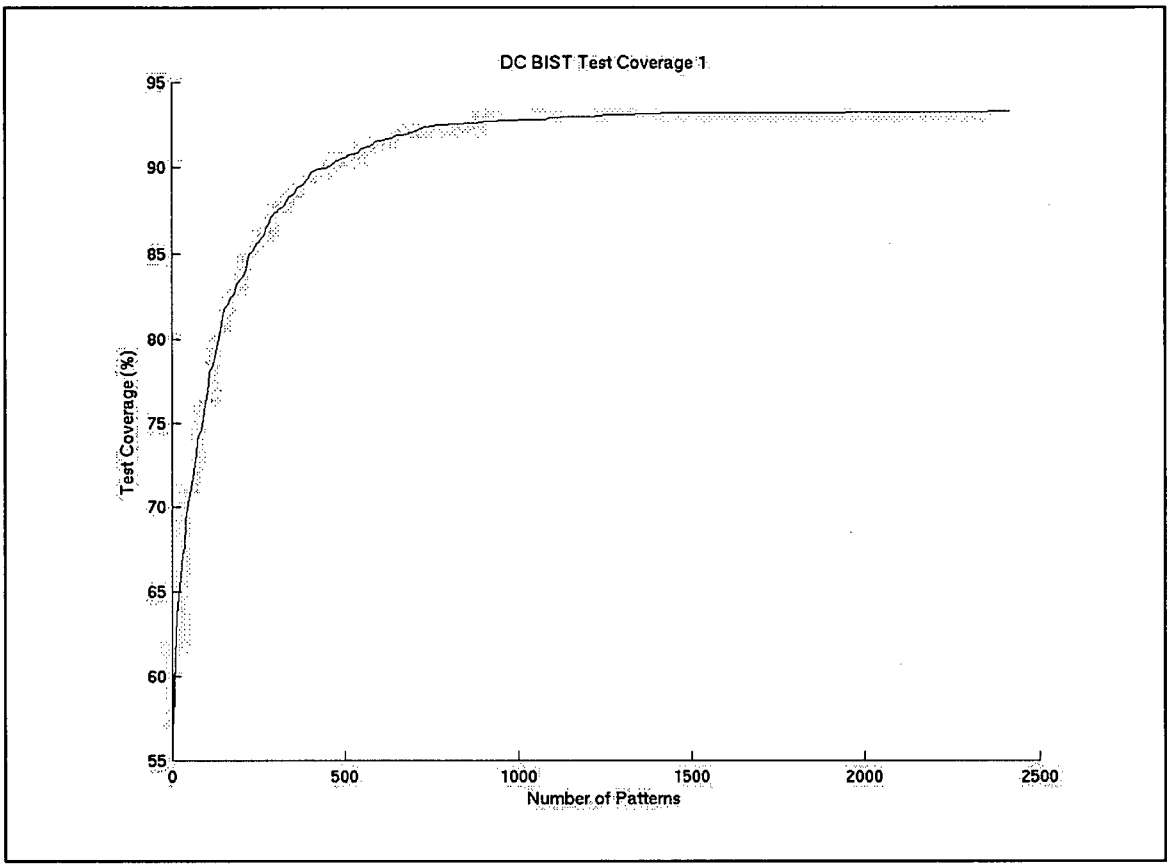


Figure 10. DC fault coverage achieved by AC test patterns

2.5 Test Pattern Generation Flow

With the relationship established between AC and DC fault tests, a test pattern generation flow suitable for a logic BIST implementation must be devised. Since the generation of an optimal test pattern set is unlikely due to the difficulty of the problem, *heuristic* methods and existing fault simulation and test generation tools are used to produce the test set. The concept is to firstly use pseudo-random test vectors for easily detectable faults. If the achieved fault coverage is insufficient, then ATPG tools are used to provide additional top-up patterns which specially target the remaining faults.

Since pseudo-random test patterns are easy to generate on-chip, they are used for easily detectable faults. Such patterns are applied using the *double-clocking* scheme explained earlier to achieve AC fault coverage and at the same time provide DC fault coverage as a side effect. If the AC fault coverage is considered insufficient after the execution of these pseudo-random test patterns, AC deterministic top-up test patterns can be generated by an ATPG tool for improved AC fault coverage. Then these AC pseudo-random and AC deterministic top-up test patterns are evaluated for DC fault coverage. Again, if the achieved DC fault coverage is deemed unsatisfactory, an ATPG tool can be employed to generate DC deterministic top-up test patterns for added DC fault coverage level.

This approach attempts to minimize the need for including these deterministic top-up patterns on-chip in the hope of simplifying the design when implemented as a logic BIST scheme. To take advantage of the relationship between AC and DC fault testing, all AC fault tests are conducted before that of DC faults. Such a strategy attempts to maximize the DC fault coverage as a side effect of AC fault tests. Consequently, all AC fault tests are detecting not only AC faults, but also DC faults simultaneously.

For this research, Verilog-XL™ by Cadence® is used for logic simulation to collect the stimuli and responses of the IP cores when pseudo-random logic BIST circuit is in operation. TetraMAX™ ATPG tool by Synopsys® is used as both the ATPG and fault simulation tool, although any ATPG and fault simulation tool can be used for this purpose. TetraMAX™ is able to generate AC test patterns based on the *double-clocking* scheme and report fault coverage numbers for the same method. Figure 11 shows that

double-clocked pseudo-random test patterns are capable of detecting many AC faults within the initial test patterns as indicated by the initial steep incline in the fault coverage. It also shows the test coverage improvement that deterministic top-up test patterns provides above that of the pseudo-random test patterns. After the test coverage achievable by the pseudo-random test patterns begins to saturate, deterministic top-up test patterns can be applied to boost up the test coverage to a desirable level.

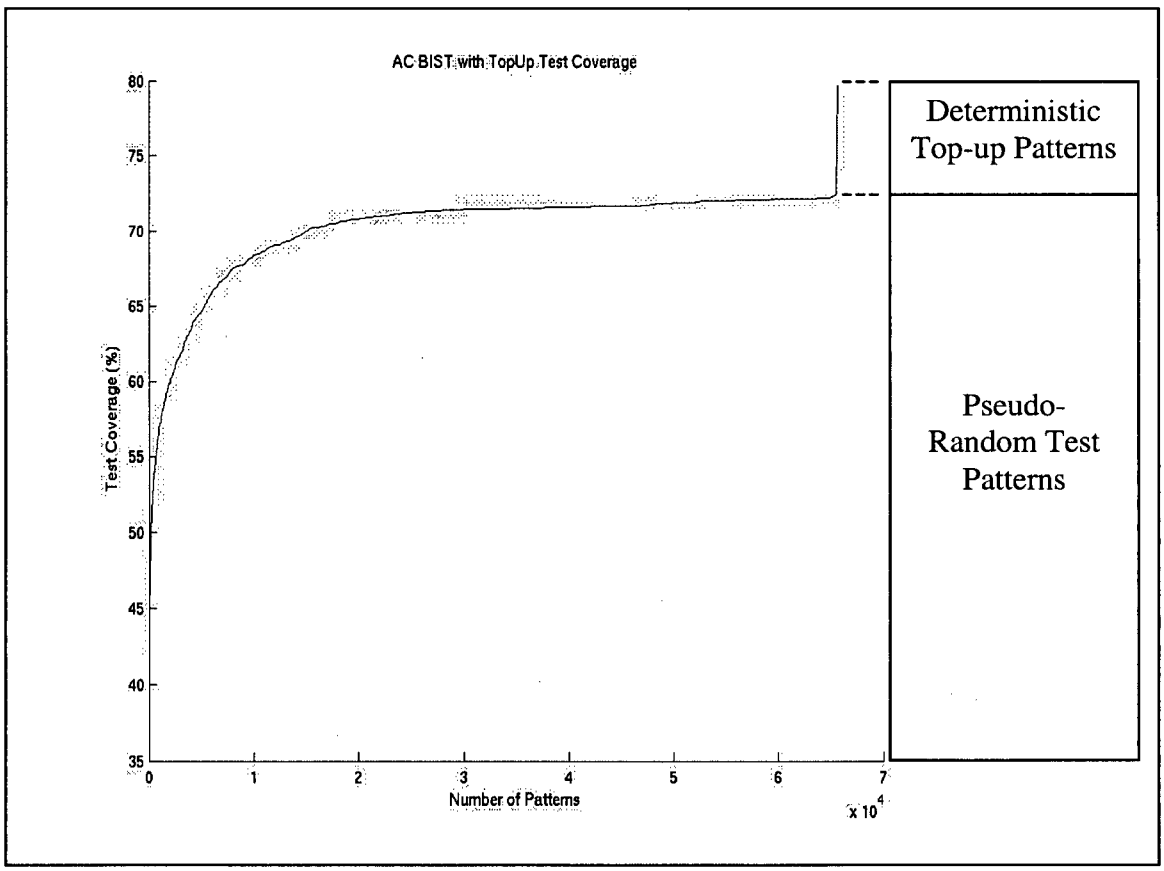


Figure 11. Pseudo-random plus deterministic pattern coverage plot

This suggests that pseudo-random test patterns are suitable for the first phase of testing to detect the easily detectable faults. After the fault coverage saturates, the pseudo-random test patterns are no longer effective at fault detection. This is where the additional top-up test patterns become important in boosting up the fault coverage to a desirable level. It is

possible to truncate the number of pseudo-random test patterns once they do not provide significant fault coverage improvement. This effectively eliminates bombardment of the IP core with more than necessary pseudo-random test patterns. Then, the remaining faults can be detected by the deterministic top-up test patterns. The point of truncation is at the IP core designer's discretion based on the target fault coverage and available test time on ATE's.

A conceptual view of the test pattern generation flow is shown in Figure 12. The particulars of this flow are explored in Figure 13.

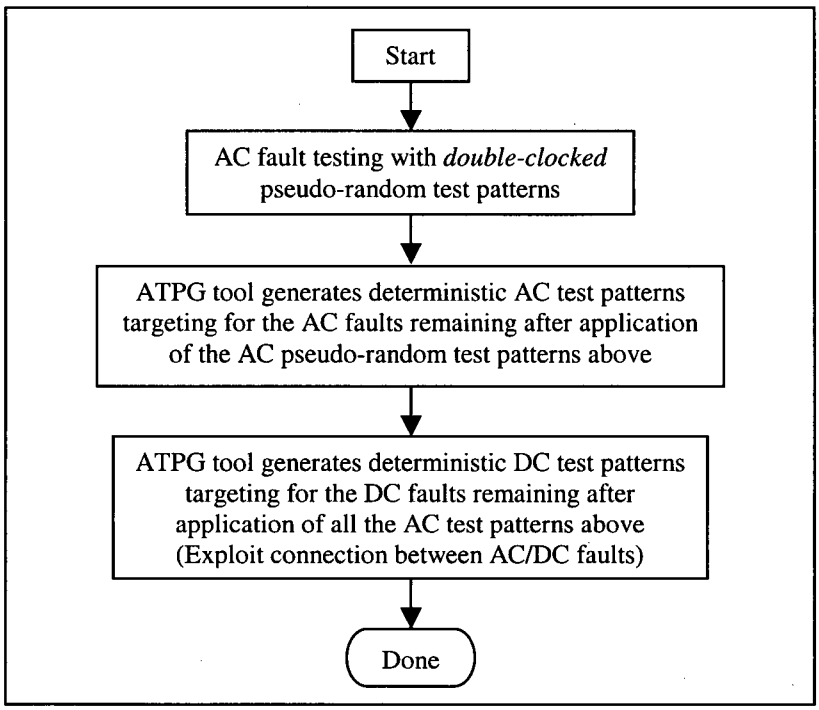


Figure 12. Conceptual view of AC/DC test pattern generation flow

Figure 13 is the detailed overall flow diagram for the test pattern generation flow. First, a logic simulation is performed on an IP core with pseudo-random test patterns according to the *double-clocking* scheme. The responses are collected and recorded for further

processing. These AC pseudo-random test patterns, along with a complete AC fault list containing all AC fault sites of the IP core, serve as inputs to a fault simulation tool to evaluate the AC fault coverage of the pseudo-random test patterns. The undetected AC faults are recorded as a partial AC fault list for later use. If the AC fault coverage is insufficient at this stage, ATPG on the IP core is performed with the partial AC fault list. This step generates deterministic top-up test patterns targeted for the remaining AC faults not detected by the pseudo-random AC patterns.

Next, the AC pseudo-random test patterns and a complete DC fault list containing all DC fault sites of the IP core, along with the AC deterministic top-up test patterns, if applicable, are fed into a fault simulation tool for DC fault coverage evaluation. The undetected DC faults are again recorded as a partial DC fault list. If the DC fault coverage is unsatisfactory, ATPG is performed on the IP core with the partial DC fault list obtained earlier. Consequently, DC deterministic top-up test patterns are generated to detect the remaining DC faults.

For an on-chip approach of this test pattern generation flow, the AC pseudo-random test patterns must be generated pseudo-randomly on-chip. If applicable, the AC and/or DC deterministic top-up test patterns must be encoded on-chip somehow to be regenerated on-chip on demand. This is the subject of Chapter 3.

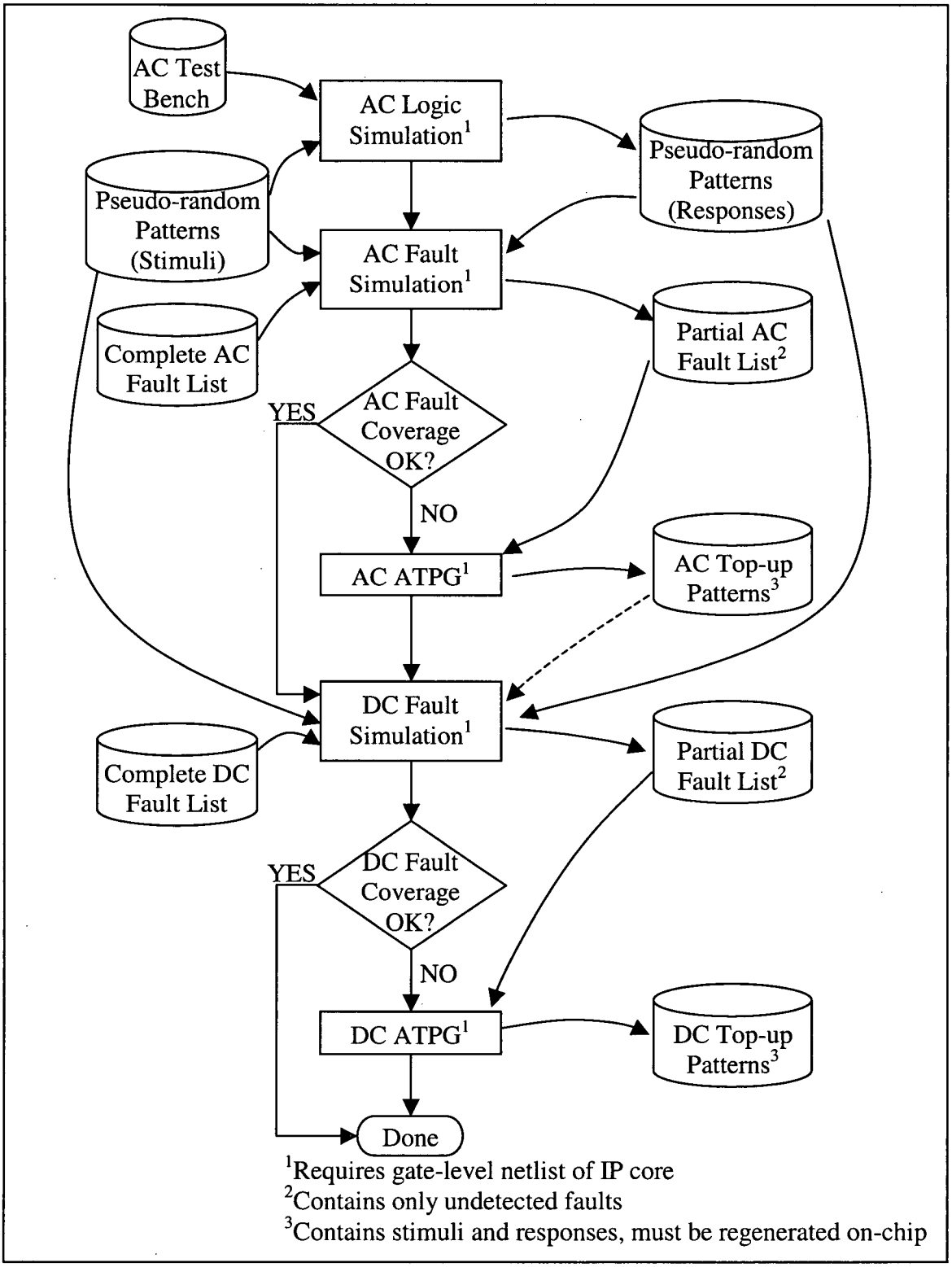


Figure 13. Detailed view of AC/DC test pattern generation flow

2.6 Fault Coverage Results

In order to evaluate the performance of the test pattern generation flow, fault coverage results of a number of IP cores are collected and analyzed. Each core is taken through the procedure outlined in Figure 13 to determine the fault coverage achievable by the test pattern generation flow. For comparison purposes, fault coverage results are collected assuming that all PI's and PO's are directly accessible by the ATE with no restriction. This situation implies the ATPG tool is allowed to generate the best suitable test patterns to attain the highest fault coverage results with least amount of test patterns. These are labeled as the "ATPG unconstrained" entries. These entries represent the best attainable fault coverage results under ideal test conditions.

However, real-time toggling of PI's and PO's on IP cores may be difficult to realize in a logic BIST scheme. Therefore, all PI's of each IP core are initially assumed to connect to the logic state 0 during test to simplify the PI's signaling. For the same reasons, all PO's of each IP core are not strobed during test. Consequently, each IP core is subjected to fault coverage analysis by the ATPG tool under the above stated conditions. This resembles the situation in which IP core is buried in the SoC design such that direct controls and examinations of each PI and PO ports are cumbersome, if not impossible. These results serve as reference points for the performance of the logic BIST test flow proposed in Section 2.5 since each IP core is tested under the similar external test conditions. These results are displayed as the "ATPG constrained" entries for each IP core.

Furthermore, all IP cores are initially tested with 65535 AC pseudo-random test patterns to provide preliminary AC fault coverage assessments while keeping the AC logic simulation time within reasonable ranges. In practice, the number of AC pseudo-random test patterns can be truncated once they are analyzed to provide limited AC fault coverage improvements. Moreover, the AC pseudo-random test pattern count, in combination with the scan chain length of an IP core, contributes to the duration that the IP core spends on an ATE. This consideration must be taken into account when determining the number AC pseudo-random test patterns to deliver to an IP core for AC fault testing. Table 1 summarizes the definition of each label entry used throughout this section.

Table 1. Label entry descriptions

Label Entry	Description
AC ATPG unconstrained	Targets AC fault coverage. PI's and PO's are controllable and observable by ATPG tool respectively. Shows the highest fault coverage achievable by ATPG tool under ideal test conditions.
AC ATPG constrained	Targets AC fault coverage. During testing, PI's are connected to constant logic state 0; PO's are not strobed. Serves as a reference point when IP core is solely tested by ATPG tool generated test patterns.
AC pseudo-random	Targets AC fault coverage. During testing, PI's are connected to constant logic state 0; PO's are not strobed. Test patterns are generated pseudo-randomly by the logic BIST circuitry.
AC pseudo-random + Det.	Targets AC fault coverage. Test conditions are similar to that of AC pseudo-random. Top-up test patterns are generated by ATPG tool according to test pattern generation flow in Section 2.5.
DC ATPG unconstrained	Targets DC fault coverage. PI's and PO's are controllable and observable by ATPG tool respectively. Shows the highest fault coverage achievable by ATPG tool under ideal test conditions.
DC ATPG constrained	Targets DC fault coverage. During testing, PI's are connected to constant logic state 0; PO's are not strobed. Serves as a reference point when IP core is solely tested by ATPG tool generated test patterns.
DC pseudo-random	Targets DC fault coverage. During testing, PI's are connected to constant logic state 0; PO's are not strobed. AC test patterns are employed according to Section 2.5.
DC pseudo-random + Det.	Targets DC fault coverage. Test conditions are similar to that of DC pseudo-random. Top-up test patterns are generated by ATPG tool according to test pattern generation flow in Section 2.5.

Fault coverage results were all obtained using the TetraMAX™ ATPG and fault simulation tool by Synopsys®. The following plots provide bar chart comparisons of the

fault coverage results obtained from the test pattern generation flow introduced in Section 2.5. Detailed numerical fault coverage results are given in Appendix A while the graphical plots of the results are included from Figure 14 to Figure 17. The DC and AC fault coverage results were obtained from TetraMAX™. It should be noted that each tool has some limitations in algorithms and implementation. Therefore, different results may be observed with different ATPG and fault simulation tools.

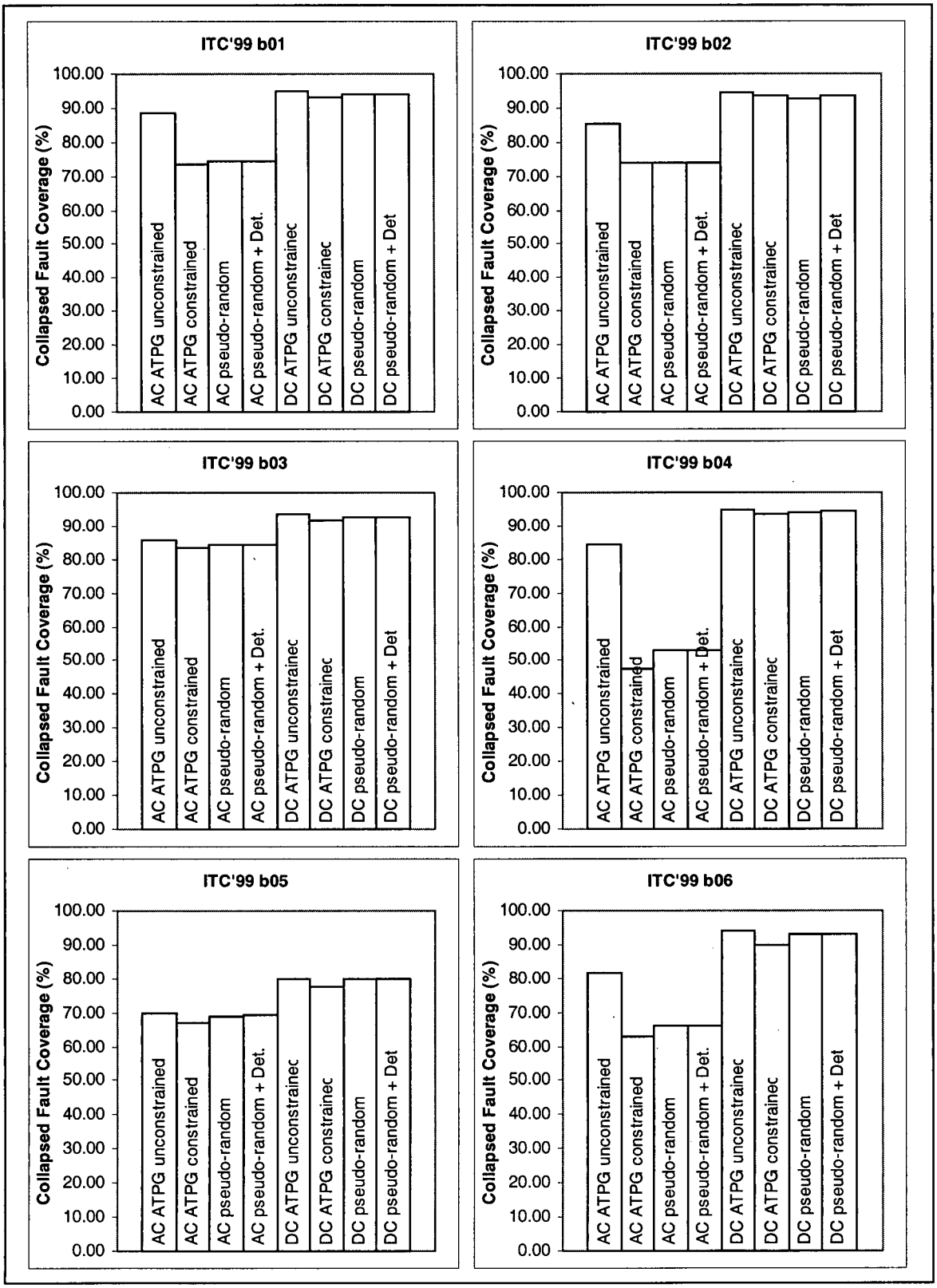


Figure 14. Fault coverage result plots 1

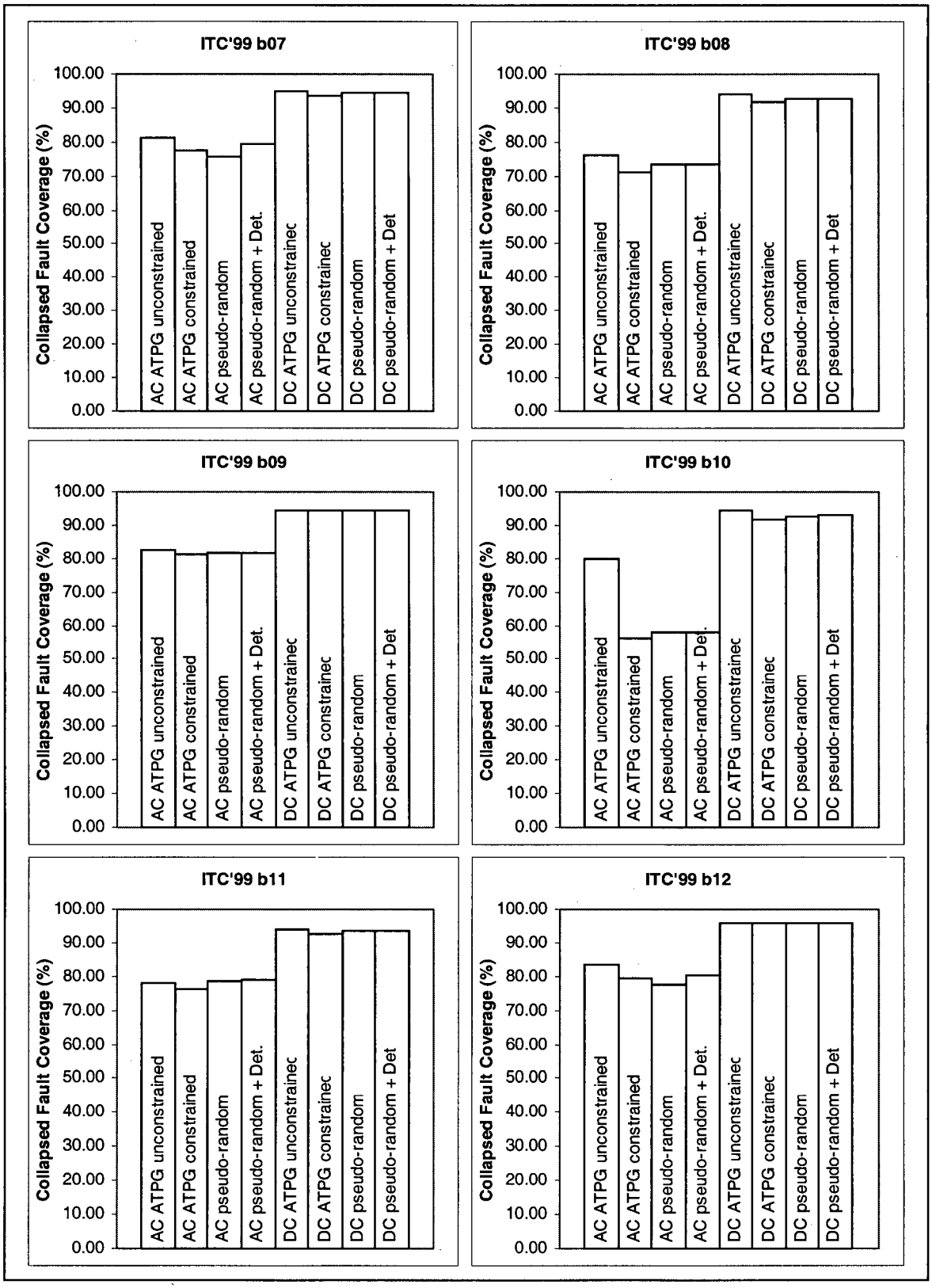


Figure 15. Fault coverage result plots 2

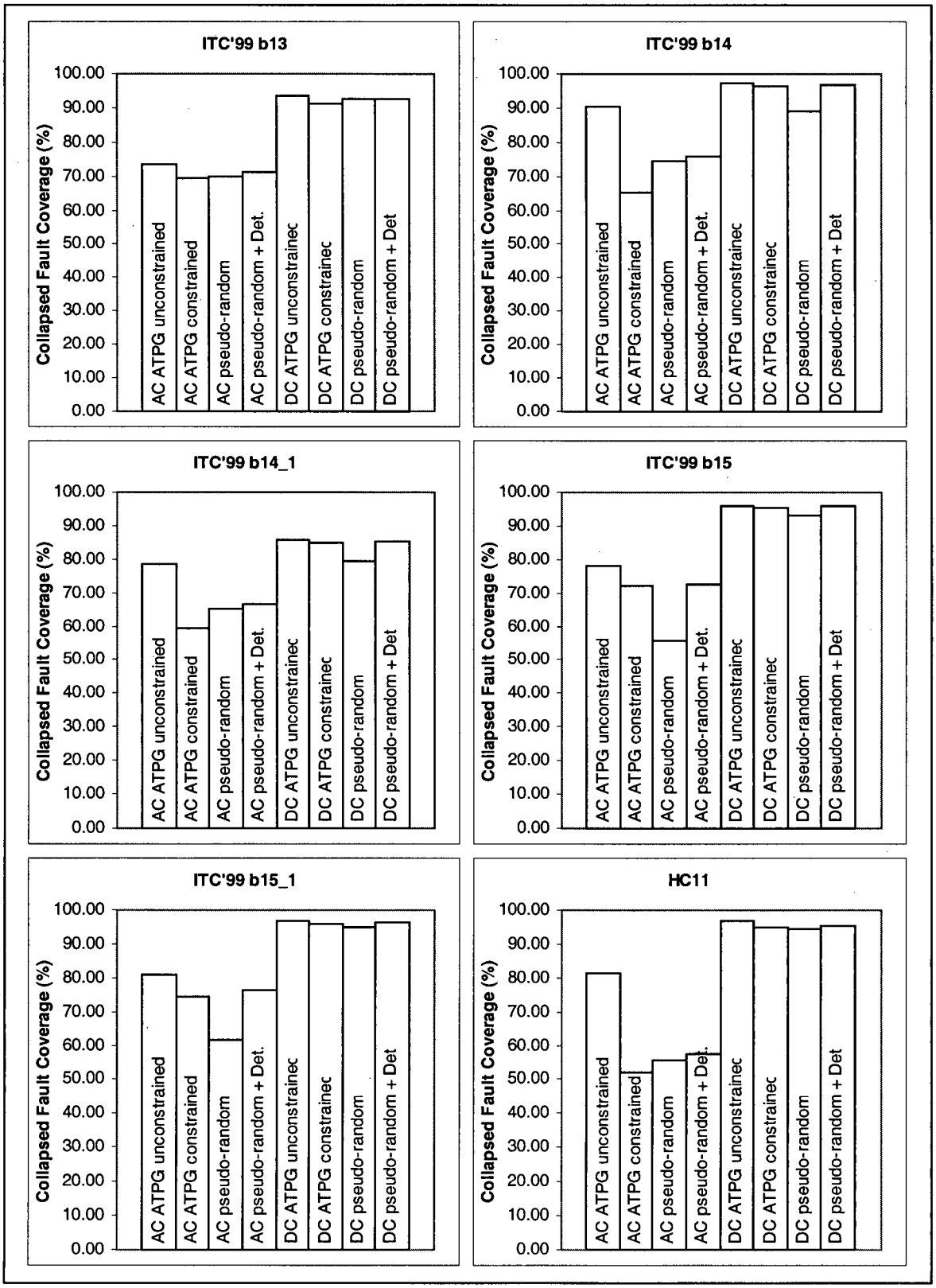


Figure 16. Fault coverage result plots 3

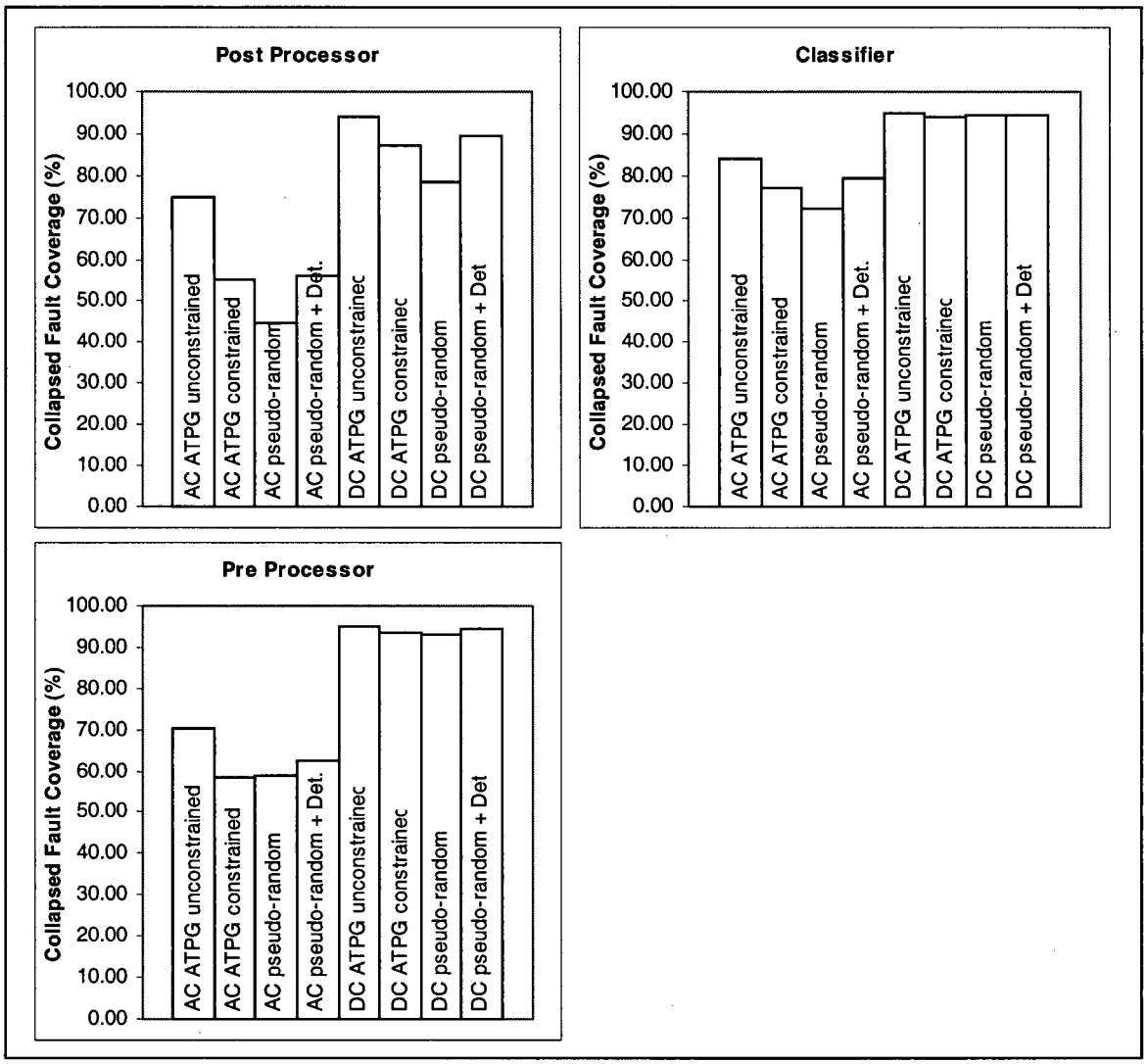


Figure 17. Fault coverage result plots 4

In general, the proposed test pattern generation flow achieves comparable performance to that of the "ATPG constrained" entries. Namely, fault coverage results of the "pseudo-random + Det." entries were equal to, or even exceed, that of the "ATPG constrained" entries. This supports the effectiveness of the proposed logic BIST test flow with reduced ATE test data volume. Such a claim is apparent with dramatic decreases in deterministic test pattern count between the "ATPG constrained" and "pseudo-random + Det." entries in each IP core. Consequently, interactions between ATE and chip are

decreased to reduce the ATE memory and speed burdens. These reduced ATE requirements would translate into the immediate benefit of lowering the costs of manufacturing tests.

2.7 AC Fault Coverage Improvement

As illustrated in the fault coverage results of Section 2.6, some IP cores such as ITC'99 b04, suffer from low AC fault coverage. This can be observed by large differences in AC fault coverage between the "AC unconstrained" and "AC constrained" results. The cause can be traced back to the constraints that all PI's are tied to logic 0 when the IP cores are being tested by the logic BIST circuitry as explained in Section 2.6. This inevitably limits the capability of the IP cores to initiate transitions to the inputs. This has negative effects on AC fault coverage. To compensate for this effect, an extra circuit can be built at each input port of the IP cores to allow input signal transition activities and is illustrated in Figure 18. Note that all IP cores are registered at the inputs and outputs according to Section 4.1.

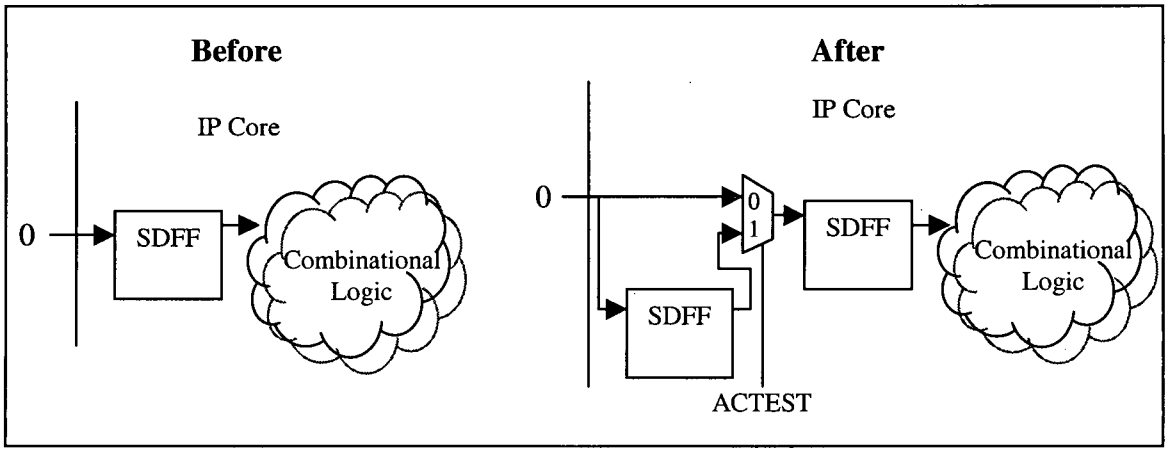


Figure 18. AC fault coverage improvement circuitry

With the addition of a SDFF and a MUX, input transitions are facilitated when the ACTEST pin is set to logic 1 throughout AC fault testing. Opposite values in the SDFF's imply transitions to the downstream combination blocks. This signal is set to logic 0 throughout DC fault testing to allow DC fault detection in the functional paths; it also should be kept at logic 0 when the IP core is in functional mode. This technique can be employed whenever AC fault coverage suffers significantly from constrained inputs. Analyses have been conducted on ITC'99 b04 IP core implemented with these extra circuits. Significant AC fault coverage improvements are observed as depicted in Figure 19 and numerical values are given in Appendix A. This addresses any perceived limitations of the approach described in this chapter.

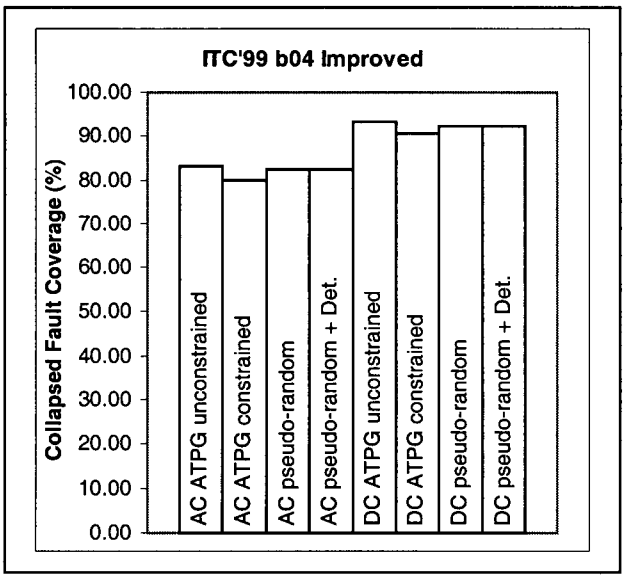


Figure 19. Improved fault coverage plot

CHAPTER 3 LOGIC BIST HARDWARE DESIGN

With the arrival of SoC design era, IP cores are often purchased from third party vendors. It is desirable to have each IP core equipped with a self-contained test methodology capable of being integrated seamlessly into any TAM. Such a requirement encourages an on-chip circuitry which automatically tests the IP and returns with a pass or fail result, or perhaps diagnostic information about any faults encountered. Typically, the on-chip solution takes the form of a built-in self-test (BIST) logic circuit. The concept of BIST is widely recognized and used in memory testing due to the structural regularity of memory. However, the utilization of BIST in logic circuits has not drawn much interest due to the inherent complexity of logic circuits, especially sequential circuits with large number of PI's and PPI's [16]. These circuits often require specially-crafted deterministic test patterns which are difficult to generate on-chip.

The goal of this research is to implement a logic BIST approach that automatically generates and applies the AC and DC test patterns from Section 2.5. The standard BIST approach is to use a pseudo-random pattern generator. However, the approach in this research also requires a set of deterministic top-up test patterns. This chapter explains the logic BIST circuit components that are responsible for pseudo-random pattern generation, response compaction, control, and deterministic pattern generation. A novel approach to deterministic pattern generator is described. The area overhead incurred by these logic BIST circuitry are presented. The area measurements are collected from the Design Compiler™ logic synthesis tool by Synopsys®.

These results are collected using standard benchmarks from ITC'99 IP cores as mentioned in Chapter 2. Further, IP cores designed in-house at UBC for the development of a simple network processor have also been used for this purpose. Each of these IP cores has a single scan chain and is registered at the input and output ports for the reasons outlined later in Section 4.1.

3.1 Pseudo-Random Pattern Generation

In order to simplify the logic BIST hardware design, a linear feedback shift register (LFSR) is implemented to produce the pseudo-random bit patterns. Figure 20 shows a canonical form of the type 1 LFSR with exclusive OR (XOR) feedback connections; exclusive NOR (XNOR) feedbacks can also be used for implementation.

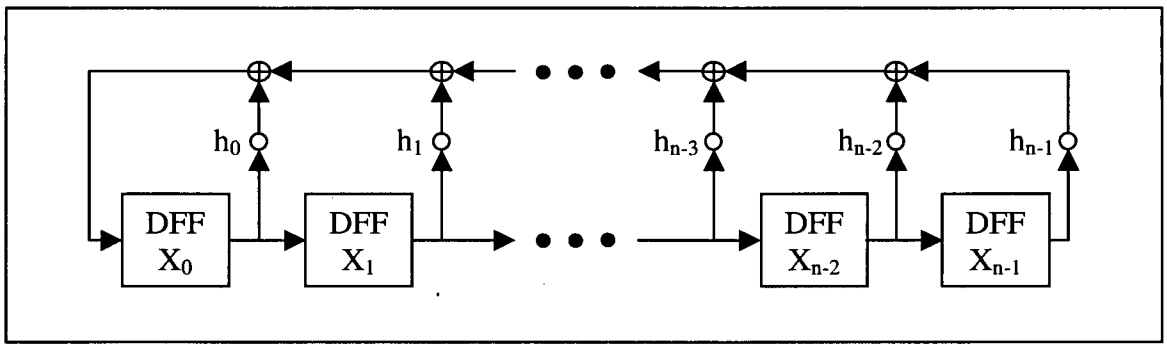


Figure 20. Canonical form of n-bit type 1 LFSR

The feedback tap locations, indicated by the h_0 to h_{n-1} coefficients, are strategically selected and represents a primitive polynomial that causes the n-bit LFSR to traverse all 2^n-1 possible states. The one particular missing state is either the all-zero or all-one state, depending on whether XOR or XNOR is used in the feedback taps, to prevent a lock-up situation. Such a sequence through 2^n-1 states is called a maximum-length sequence [8].

For mathematical analysis, an LFSR of this type can be conveniently represented and manipulated by matrix algebra with modulus-2 addition. If XOR gates were used to implement the LFSR, addition operations must comply with the XOR logic. Since there are no carry or borrow operations in XORing arithmetic, the results are 0+0=0, 0+1=1, 1+0=1, 1+1=0. Conversely, if XNOR gates were used for the LFSR implementation, addition must comply with the XNOR operation. Again, with no carries or borrows, 0+0=1, 0+1=0, 1+0=0, 1+1=1. Assuming the current LFSR state is represented by:

$$X(t) = [X_0(t), X_1(t), \cdots, X_{n-2}(t), X_{n-1}(t)] \tag{2}$$

then, the next LFSR state can be calculated as a product of the matrix multiplication:

$$X(t+1) = X(t)T_c \tag{3}$$

where the transition matrix, T_c, is an nxn binary matrix and is defined as:

$$T_c = \begin{bmatrix} h_0 & 1 & 0 & \cdots & 0 & 0 \\ h_1 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ h_{n-3} & 0 & 0 & \cdots & 1 & 0 \\ h_{n-2} & 0 & 0 & \cdots & 0 & 1 \\ h_{n-1} & 0 & 0 & \cdots & 0 & 0 \end{bmatrix} \tag{4}$$

The variables h_0 to h_{n-1} in the T_c matrix specify the feedback tap connections. When there is a connection to the feedback network, the variable corresponding to the feedback connection is assigned a binary value of 1. On the other hand, the variable is assigned a binary value of 0 if the corresponding feedback connection does not exist. Each

multiplication of the LFSR with T_c corresponds to a right shift of the LFSR. As a result, the operation of the LFSR can be fully predicted by such matrix algebra.

Bits produced at the X_{n-1} position are fed into the scan chain of the IP core as scan patterns. The implemented LFSR for pseudo-random pattern generation consists of 32 bits and employs XNOR gate with the feedback taps located at:

$$X^{31} + X^{21} + X^{1} + 1 \tag{5}$$

In other words, there are 4 feedback tap locations and they are located at h_{31}, h_{21}, h_1 and h_0. The choice of implementing a 32-bit LFSR for pseudo-random test pattern generation is based on its capability of traversing 2^{32}-1 states before it revisits any one of the previous states. This ensures the LFSR does not wrap around and produces the same pseudo-random patterns before completion of the logic BIST circuitry operation. In order for a 32-bit LFSR to traverse all 2^{32}-1 states, the aforesaid feedback tap configuration is only one of the many possible feedback tap configurations.

3.2 Response Compaction

When an IP core is being tested, the results being shifted out from the IP core scan chain must be collected for verification against the correct results. However, it is impractical to store all the response bits for later inspection nor is it acceptable to send the response bits off-chip for verification in the context of a logic BIST implementation. Both of these methods require either large amount of storage or high ATE bandwidth. Therefore, a circuit is implemented to compact the response bits into a signature which is compared, at

the end of logic BIST operation, against that obtained from logic simulation. This circuit is also known as a Single-Input Signature Recognizer (SISR) when it accepts a single response stream from the IP core under test. Similarly, it is known as a Multiple-Input Signature Recognizer (MISR) when it accepts multiple response streams from multiple scan chains.

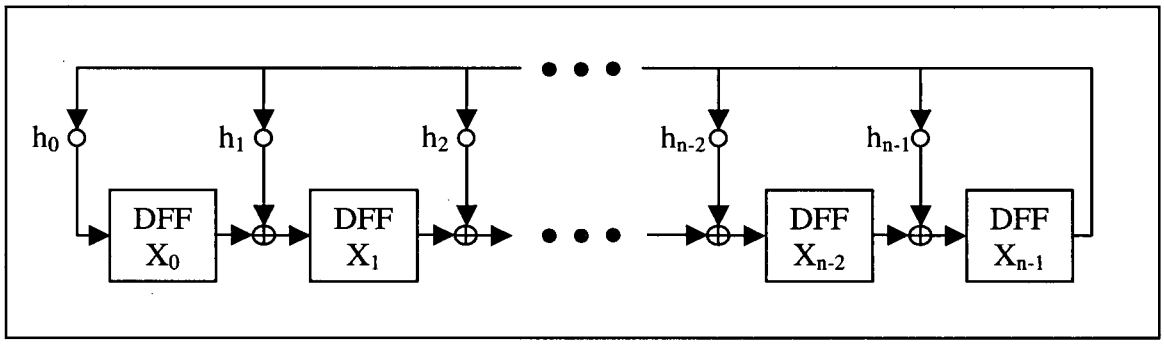


Figure 21. Canonical form of n-bit type 2 LFSR

Another type of LFSR serves as the basis of the compaction circuit. Figure 21 shows the canonical form of the circuit generally known as a type 2 LFSR. Similar to the type 1 LFSR, matrix algebra can also applied to this type of LFSR. The nxn transition matrix, T_c, for type 2 LFSR is defined as:

$$T_c = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ h_0 & h_1 & h_2 & \cdots & h_{n-2} & h_{n-1} \end{bmatrix} \tag{6}$$

A cyclic redundancy check (CRC) is implemented for the compaction task. This scheme is widely used in the field of telecommunications for error detection [8]. The implementation is a CRC-32 design based on the type 2 LFSR with feedback tap

locations selected according to cyclic code theory in telecommunications. The characteristic polynomial selected for the feedback tap location is:

$$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^{8} + X^{7} + X^{5} + X^{4} + X^{2} + X^{1} + 1 \quad (7)$$

In other words, the LFSR consists of 32 bits and has 14 feedback tap locations. These feedback taps are located at h_{26}, h_{23}, h_{22}, h_{16}, h_{12}, h_{11}, h_{10}, h_8, h_7, h_5, h_4, h_2, h_1, and h_0. The response bits from the IP core scan chain undergo an XOR operation with X_{n-1} before being fed into the X_0 DFF as shown in Figure 22.

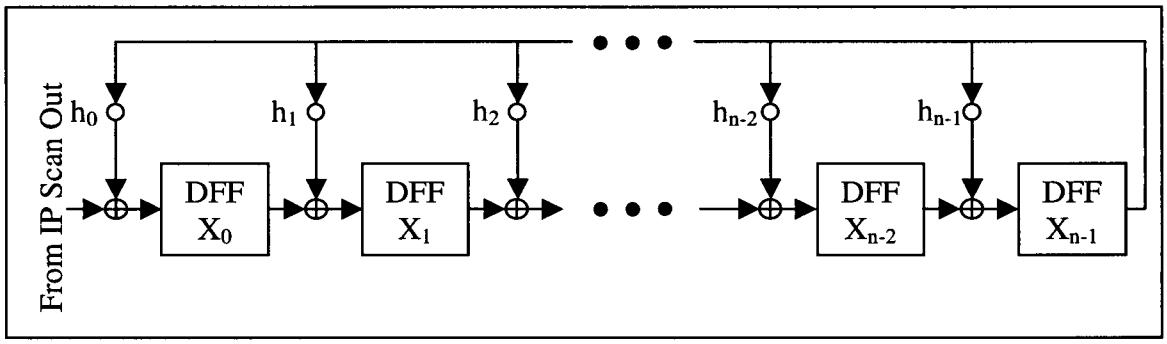


Figure 22. Response compaction circuit design

This CRC circuit simply performs polynomial modulus-2 division. The characteristic polynomial, implemented as the feedback tap locations, acts as the divisor while the bits received from the IP scan chain compose the dividend. In each clock cycle, this CRC circuit divides the characteristic polynomial into the IP response bits and stores the remainder in the LFSR state. This remainder serves as the signature of the test and is compared against the signature derived from simulation to determine the IP core integrity. This circuit is able to calculate the signature in a real-time fashion; it is capable of performing the division using the serial bit stream from the IP scan chain. In effect, as

the last bit from the IP core enters the CRC circuit, the signature already resides in the current LFSR state and thus results of the IP core test is available immediately after the last response pattern is scanned out from IP core.

3.3 Controller

The logic BIST controller is responsible for coordinating the operation between the IP, pattern generation block and response compaction block. Figure 23 provides a conceptual relationship of these four blocks.

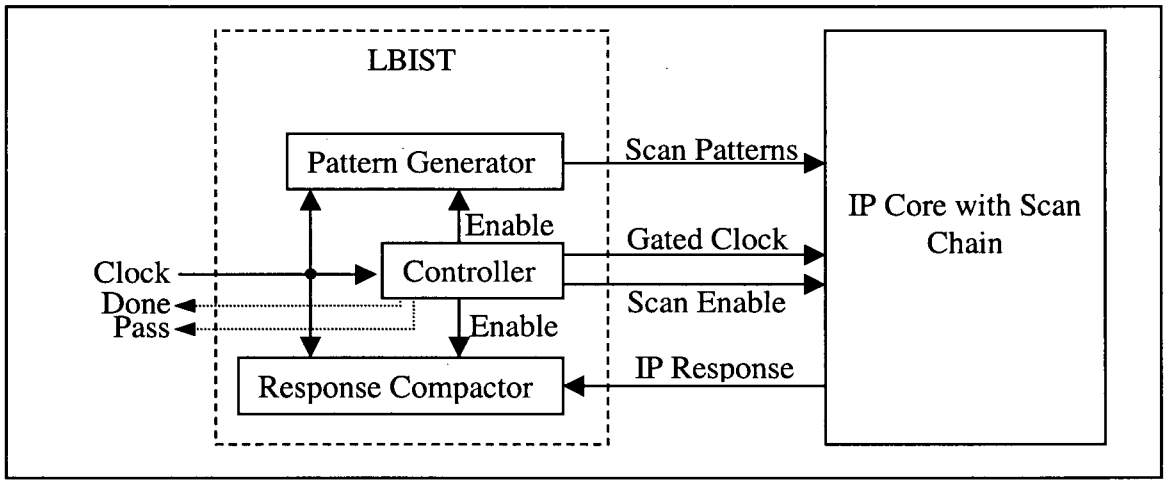


Figure 23. Overview of logic BIST design

The controller is an FSM and is responsible for gating the system clock to the IP during test. It is also responsible for keeping count of number of bits and number of patterns being shifted into the IP. This is necessary for proper sequencing of the scan enable signal to switch the IP between normal and test modes as required by any scan design. Further, it controls the enabling of the pattern generation and response compaction

blocks. The controller has a minor difference in operation when it is targeting for DC versus AC faults.

3.3.1 Controller for DC Faults

When the controller is used for DC fault detection, it only supplies one system clock cycle to the IP core during normal mode, after shifting the test patterns into the scan chain. This conforms with the clocking for DC fault detection with scan designs. Figure 24 shows the timing diagram of the controller when detecting DC faults.

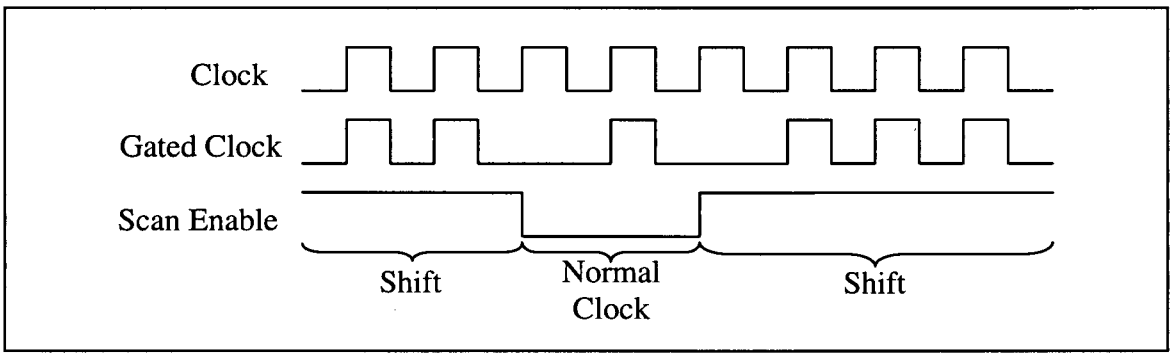


Figure 24. DC fault timing diagram

Since there is only a single system clock cycle applied to the IP core during normal mode, the outputs at one level of FF's are propagated through only one block of combinational logic as illustrated in Figure 25. Then the responses are captured and ready for shifting out of the scan chain in order to be analyzed.

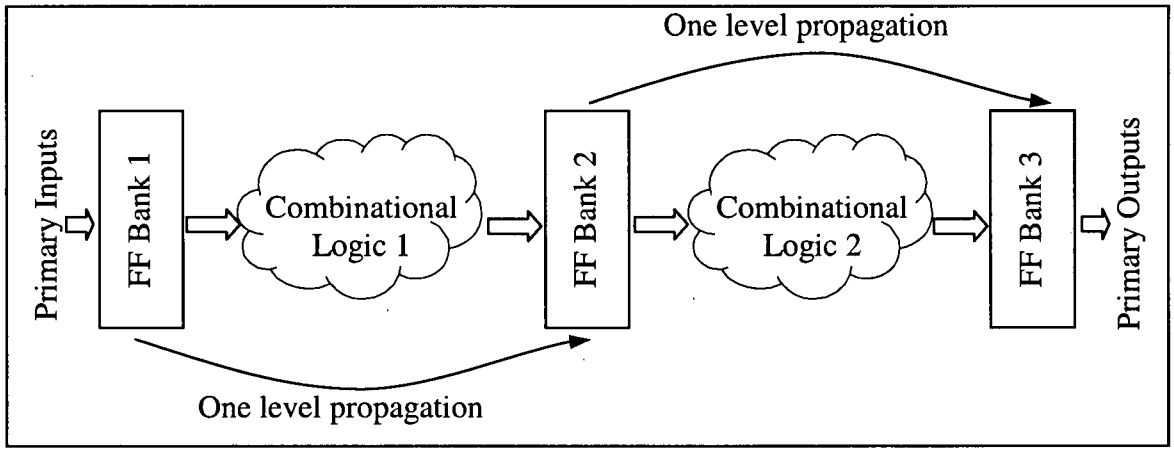


Figure 25. DC pattern propagation

3.3.2 Controller for AC Faults

When the controller is used for AC fault detection, it applies two consecutive at-speed system clock cycles to the IP core during normal mode surrounded by test modes [27][20][3]. They are called *double-clocked* AC test patterns in this research. Figure 26 shows a timing diagram of the controller when detecting AC faults.

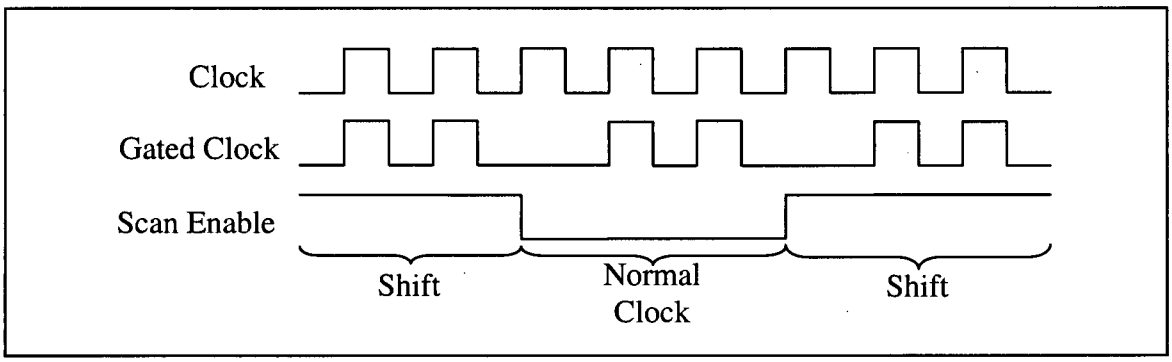


Figure 26. AC fault timing diagram

Since there are two consecutive at-speed system clock cycles applied to the IP core during normal mode, the outputs at one level of FF's are propagated two levels downstream through two blocks of combinational logic as shown in Figure 27. This is

often called the broadside method [27], although it is referred to as the *double-clocking* method in this thesis.

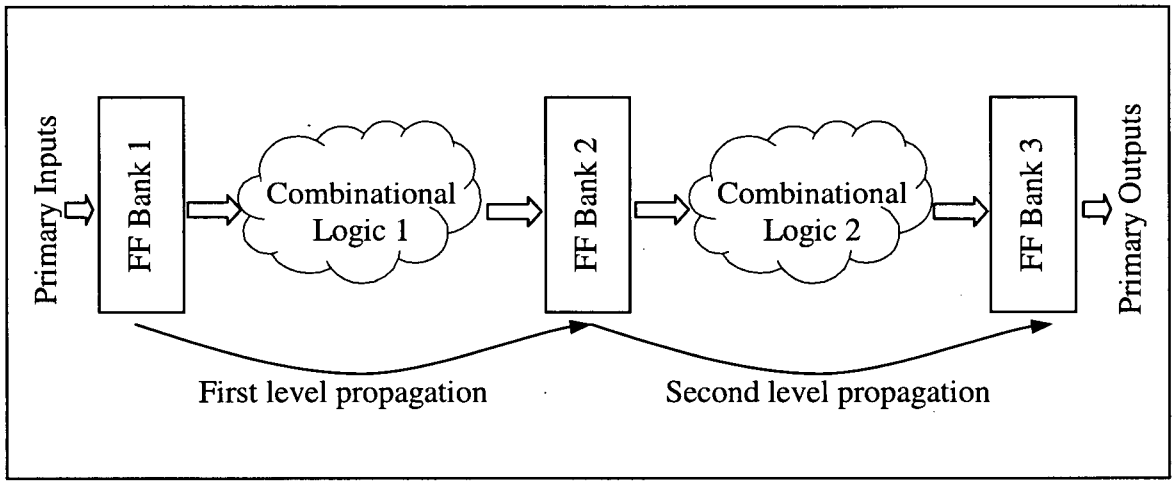


Figure 27. AC pattern propagation

As mentioned in Chapter 2, an AC fault requires two patterns to initiate a desired signal transition. Before the first system clock hits in normal mode, FF bank 2 has certain bit pattern created by the scan chain shifting and serves as the first vector for combinational block 2. After the first system clock in normal mode, FF bank 2 captures outputs from combinational block 1 and serves as the second pattern for combinational block 2. When the second system clock cycle in normal mode hits, combinational block 2 outputs are captured in FF bank 3 and ready for shifting out for examination. *This scheme relies on the circuit's combinational logic, namely combinational block 1, to produce the second pattern as required for AC fault testing.* Therefore, the logic BIST circuit generates pseudo-random vectors that will be *double-clocked* to serve as the AC test.

3.4 Deterministic Pattern Generation

As an integral part of the test pattern generation flow proposed in Section 2.5, deterministic pattern generators are required to produce the deterministic top-up test pattern to boost the fault coverage up from that achieved by the pseudo-random test pattern set. This section investigates the generation of deterministic test patterns on-chip. This has always been a difficult issue and often hinders the widespread acceptance of logic BIST. In this chapter, possible approaches for on-chip test vector generation are investigated. Furthermore, a method based on using LFSR's and matrix algebra is proposed to overcome the shortcomings which exist in the other approaches.

3.4.1 On-Chip Deterministic Test Pattern Generation Approaches

There are several methods that can be used when generating deterministic test patterns on-chip. All of these methods have a common goal of reproducing a pre-defined set of test patterns. They mainly vary in how test pattern bits are stored and represented. Memory storage, compaction, and compression are discussed.

3.4.1.1 Memory

This is probably the most intuitive and simple method to generate deterministic test patterns on-chip. The pattern bits are simply stored in read-only memory (ROM) which is designed and implemented at design time. During operation, a controller with address decoding capability simply fetches the test pattern bits by performing memory reads and feeds the bits to the IP core under test. Test responses are collected from the IP core and

compacted by a response compaction block. Figure 28 shows logic BIST with use of memory.

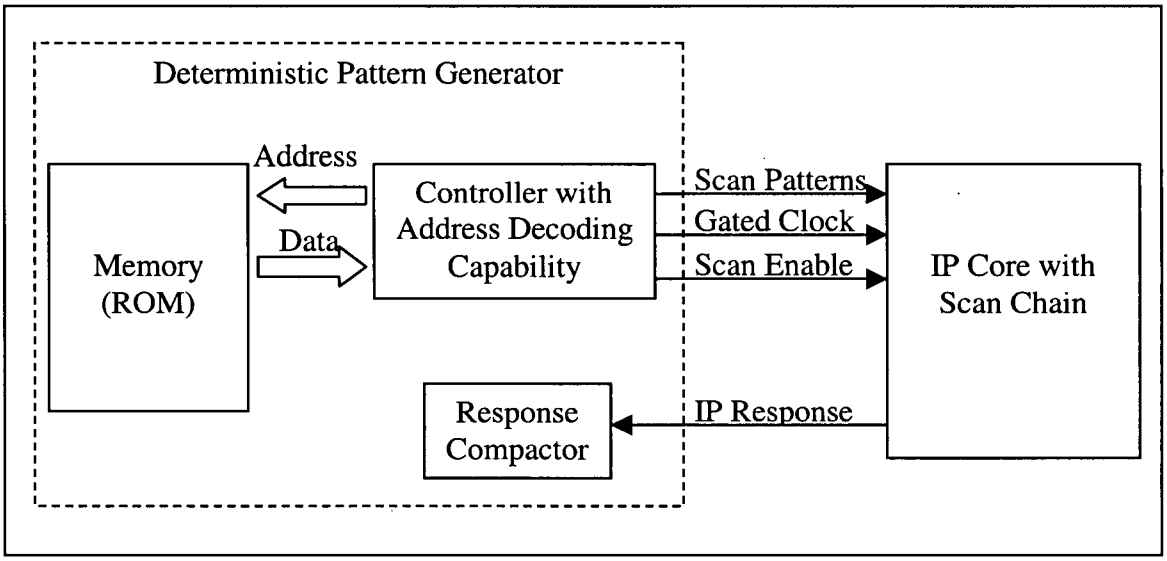


Figure 28. Memory-based deterministic pattern generator

The advantage of this approach is the simplicity in implementation. Designers are required to decide on the test patterns and design a ROM block to store these test pattern bits. Then the entire design can be sent for fabrication.

With this method, it minimizes the coupling between the ATE and the IP testing since only minimal interaction is required; thus reduction on ATE requirements for testing high performance chips can be achieved. However, there are several disadvantages. One obvious drawback is the correctness of ROM contents after fabrication. It is quite possible that defective ROM invalidates the entire deterministic test vector generation process and causes misleading results. A failure reported by logic BIST may be caused by incorrect test patterns due to defective ROM contents. As a result, a memory BIST and bit error correction for the ROM may be necessary to ensure the ROM integrity.

Therefore, memory testing, diagnosis and repair would be required in a ROM implementation.

3.4.1.2 Compaction

This method involves generating non-exact test patterns on-chip which achieve fault coverage comparable to that of the exact test patterns. It attempts to simplify the hardware involved by sacrificing the correct generation of some bits in test vectors. These non-exact bits are the don't care bits, usually represented by X, which exist in most of the test patterns generated by ATPG tools.

One implementation is the bit-flipping algorithm [35][18][17]. It relies on an LFSR to produce a stream of bit patterns. This bit stream is then compared against the test pattern bits generated by the ATPG tool to determine bits with opposite values. These bits, identified by the pattern and bit counts, are the ones which must be forced to the opposite logic values (flipped). The don't care bits generated by the ATPG tool are ignored and do not require flipping to the opposite values. In order to flip the LFSR generated bits appropriately, a combinational logic block is required to generate a signal based on the pattern and bit counter. This signal represents whether current bit requires to be flipped. Figure 29 depicts the overall structure of this method.

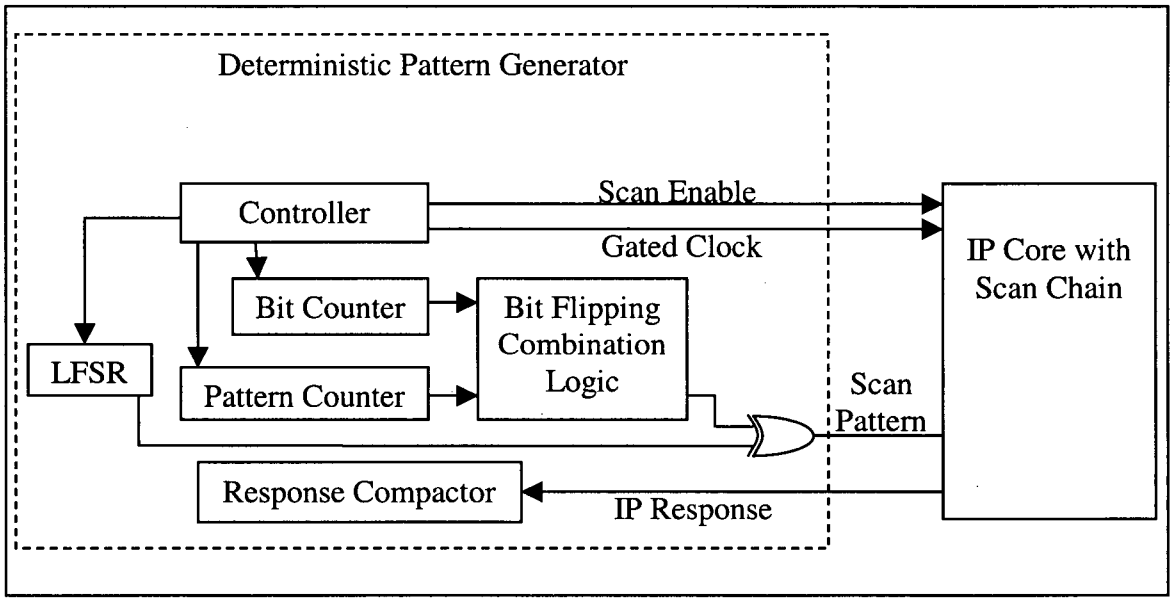


Figure 29. Bit-flipping deterministic pattern generator

The advantage of this method is the simplicity of implementation. It merely requires a comparison between the LFSR generated bit sequence and that of the test patterns. Then the bit-flipping combinational logic can be automatically generated based on the pattern and bit counts. A probabilistic analysis claims that the output of LFSR which feeds an IP core scan chain has to be modified only at a few bit positions in order to transform the pseudo-random patterns into a complete test set [35][18].

Nevertheless, the robustness of this algorithm may be compromised when half of the LFSR generated bits mismatch the ATPG tool generated deterministic test pattern bits. In this case, the XOR gate in Figure 29 can simply be replaced by an XNOR gate to reduce logic. Furthermore, size of the bit-flipping combinational logic expands as the number of bits flips increases. As a result, this method may not be as robust as claimed in [35][18].

3.4.1.3 Compression

This method attempts to minimize the volume of test data from the ATE to the chip. By sending compressed test data to each IP core and providing hardware to decompress the test data, the ATE bandwidth requirement is expected to be reduced [34]. Such a method involves implementation of a decompressor to decode the test data. Figure 30 shows an overview of the design.

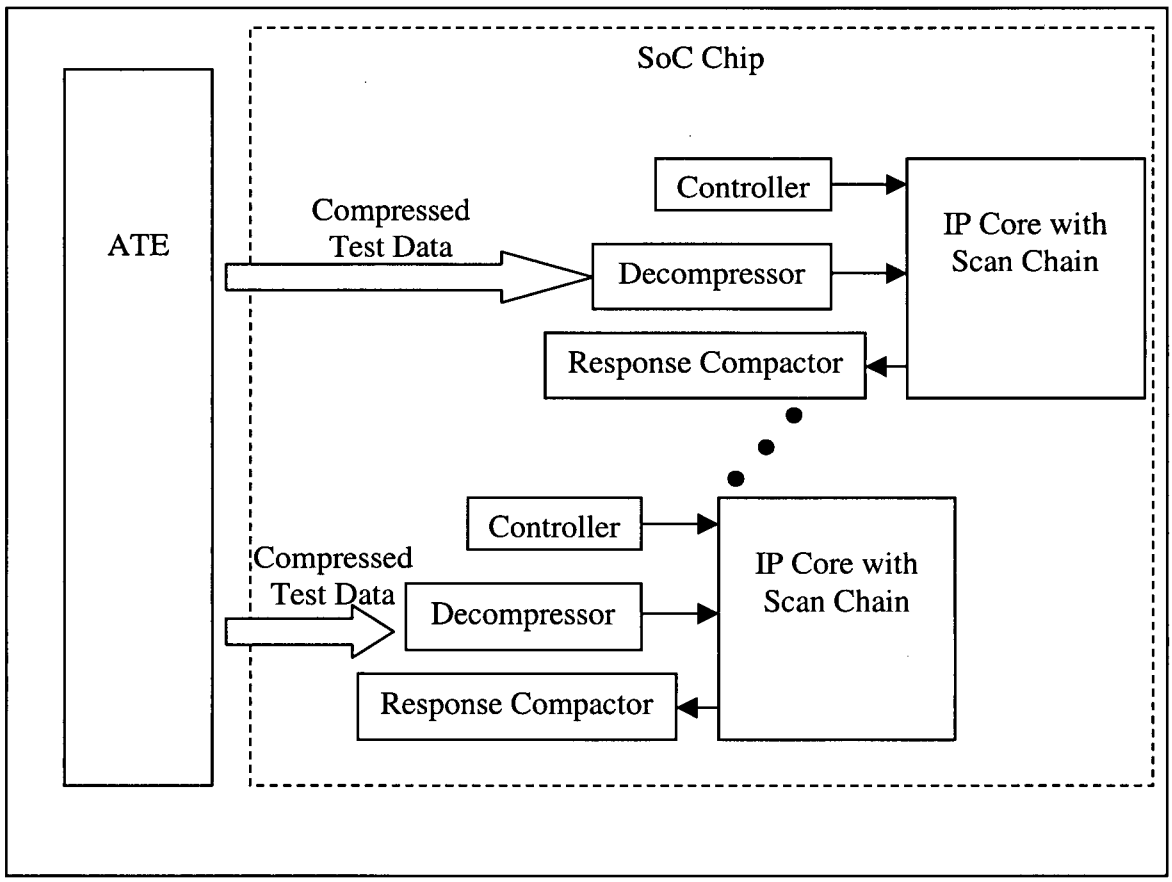


Figure 30. Compression-based deterministic pattern

The coupling between the ATE and the core still exists to some extent although the test data volume is reduced. It is possible to store the exact, compressed test data in an on-chip ROM to minimize the involvement of ATE. By introducing the use of a ROM, the drawbacks of using memory mentioned in Section 3.4.1.1 are encountered. Furthermore,

the degree of compression possible is always limited for a variety of reasons. If only minimal compression can be achieved, the main purpose and advantages of this method are defeated.

3.4.2 LFSR-Based Deterministic Pattern Generation Principles

This section describes an LFSR-based deterministic pattern generation method which utilizes matrix algebra to determine the feedback tap locations of a LFSR. This algorithm is based on the work in references [29][7]. With properly specified feedback tap locations, the desirable patterns can be regenerated in consecutive clock cycles. The algorithm initially assumes an n-bit type 2 LFSR illustrated in Figure 21.

Recall that the next state of type 2 LFSR can be calculated by matrix algebra according to Equation 3: $X(t+1) = X(t)T_c$. Consider a transformation of X(t) to X'(t) with an arbitrary, invertible nxn matrix A according to the relationship:

$$X'(t) = X(t)A \tag{8}$$

then:

$$X'(t+1) = X(t+1)A \tag{9}$$

$$X'(t+1) = X(t)T_c A \qquad \text{according to (3)}$$

$$X'(t+1) = X'(t)A^{-1}T_c A \qquad \text{according to (8)}$$

$$X'(t+1) = X'(t)T_s \tag{10}$$

By observation, Equation 3 and Equation 10 have similar forms. This implies the next state of the LFSR, X'(t+1), can be controlled if the LFSR feedback connections are implemented according to that specified by T_s. Further, note that nxn matrix A is arbitrary and can be chosen such that its rows correspond to the desired deterministic patterns.

The LFSR feedback connections can be read off from the T_s matrix. The input of jth FF is calculated by exclusive-or (XOR) operation of the ith FF outputs if the corresponding t_{ij} entries in T_s matrix equal to 1. Input of the jth FF is connected to ground if all entries in the jth column of T_s matrix equal to 0. Figure 31 illustrates the feedback tap connections of a 4-bit LFSR implied by a 4x4 T_s matrix.

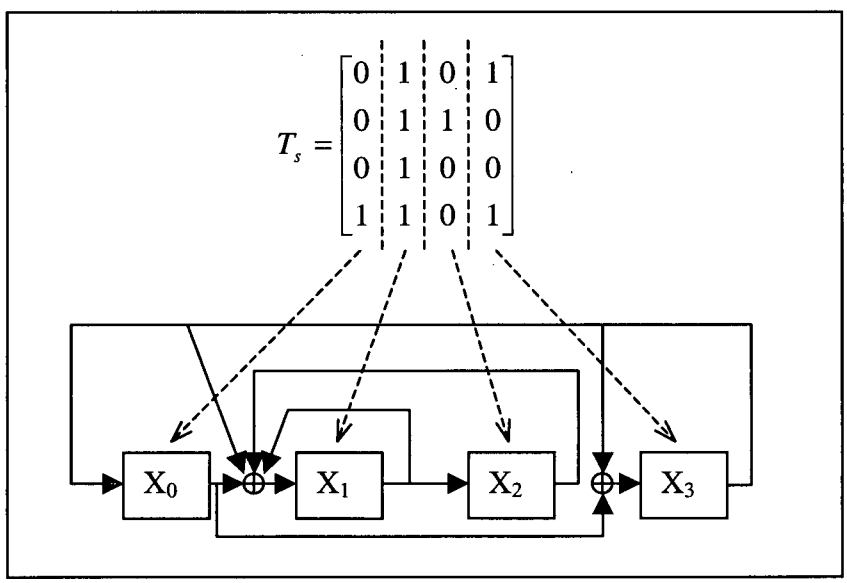


Figure 31. Deterministic LFSR implementation

If Equation 10 were modified such that X'(t) = A, then Equation 10 becomes:

$$X'(t+1) = X'(t)T_s = AA^{-1}T_cA = IT_cA = T_cA \tag{11}$$

where I is an nxn identity matrix with all entries set to 0 except along the diagonal. Considering the definition of the nxn T_c matrix and principles of matrix algebra, the quantity X'(t+1) merely represents a transformed version of A with all rows shifted up by one row. This is precisely the desired behavior which the LFSR in one clock cycle later should produce: the next desirable deterministic pattern corresponding to the next row in the nxn matrix A.

One of the issues of this approach is that the implementation of such a pattern generator may require a large number of FF's. Analysis was performed on this algorithm to observe its behaviour when the number of bits and number of test patterns increase. Figure 32 is a 2-D plot of a slice of the worst-case cost function, in terms of number of FF's, against the number of patterns or the number of bits in a pattern. The 3-D plot of simultaneous variations in the number of patterns and the number of bits in a pattern is given in Figure 33. It is observed that the number of FF's required for the LFSR implementation is linear with the number of test patterns when the number of bits per pattern is fixed. A similar observation is seen with variation in the number of bits per pattern while the number of patterns is kept constant. This suggests that the cost, measured by the number of required FF's, for implementing an LFSR-based deterministic pattern generator is directly proportional to the variations in the number of patterns and the number of bits per pattern. As a result, the number of deterministic patterns to be generated should be kept small in an effort to minimize the amount of hardware involved in the deterministic pattern generator.

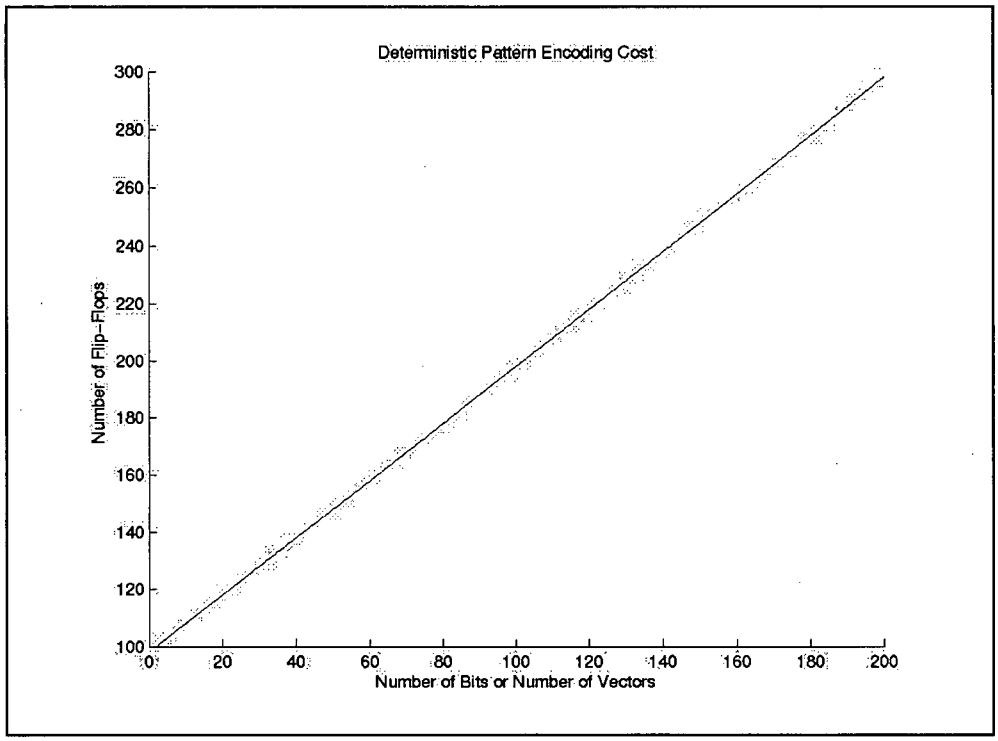


Figure 32. 2-D Cost function of LFSR-based deterministic pattern generation

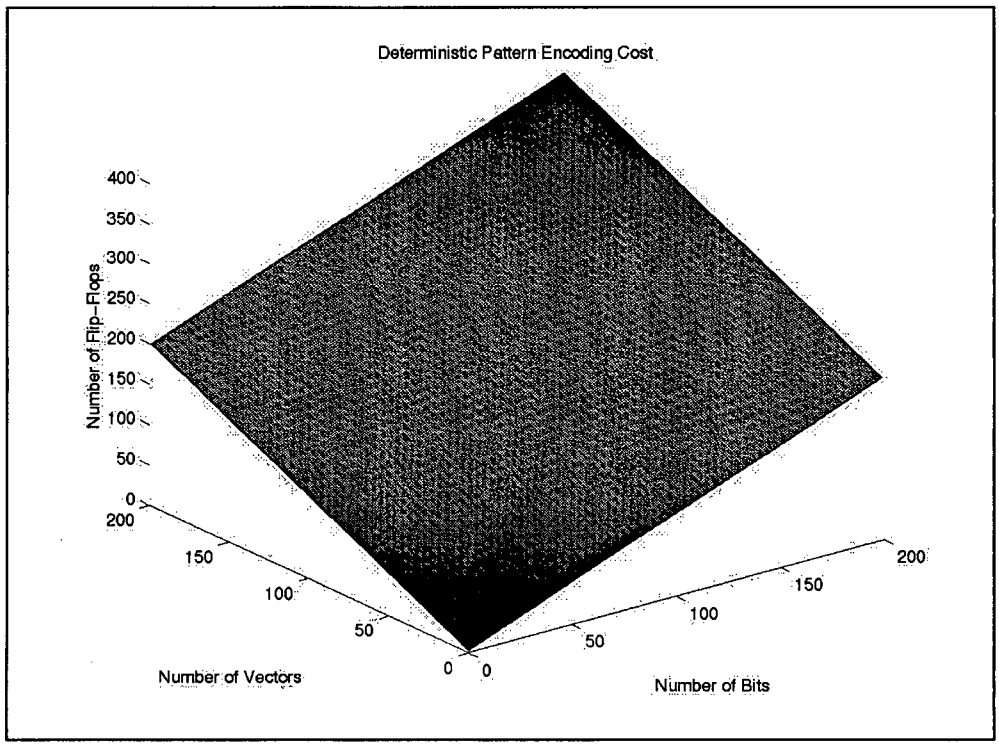


Figure 33. 3-D Cost function of LFSR-based deterministic pattern generation

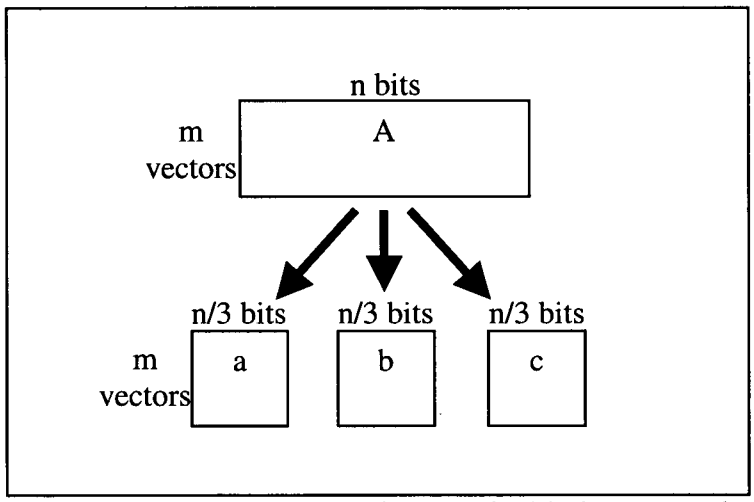
3.4.3 LFSR-Based Deterministic Pattern Generation Pitfalls

With the algorithm introduced in Section 3.4.2, the LFSR feedback tap locations capable of generating deterministic patterns can be calculated conveniently. However, further consideration of this algorithm reveals a serious flaw. The deterministic test patterns are fed into the IP core scan chain with hundreds of scan elements which corresponds to the length of each test pattern. This length, n, is represented by the width of the A matrix. According to the algorithm, the LFSR length is at least as long as the width, n, of the A matrix. This implies the LFSR length is as long as, if not longer than, the IP scan chain length. Such a property is undesirable because it potentially leads to large area overhead.

Further, the algorithm requires the A matrix to be a square matrix which implies the number of rows, m, equals the number of columns, n. Since rows in matrix A represent deterministic test patterns, the above requirement implies the patterns must be appended with dummy bits when pattern count, m, mismatches the scan chain length, n. Because these deterministic patterns are assembled by ATPG tools, designers have limited control over the pattern count given a target fault coverage. As a result, the above requirement produces an inefficient implementation of the LFSR-based deterministic pattern generator. The length of such LFSR thus becomes at least max(m, n) if the algorithm were to be strictly followed.

There is a possible argument that the aforementioned issues can be alleviated, to some extent, by implementation of multiple scan chains in the IP core. Such inclusion of multiple scan chains affects vertical splitting of matrix A into multiple matrices. Figure

34 shows an example of splitting the A matrix up into three matrices a, b, and c



corresponding to IP core implemented with three scan chains of length n/3 each.

Figure 34. Vertical test pattern split

There are situations where the number of scan cells is approximately three times as many as the number of patterns. In terms of the A matrix, $n \approx 3m$. As a result of having multiple scan chains, the three resultant matrices have dimensions of n/3, which approximates m. This satisfies the requirement of having square matrices for deterministic patterns.

Unfortunately, this solution has the cost of having to implement one LFSR for each scan chain. Further, the IP synthesis flow is disrupted as a consequence. Designers must estimate the number of patterns, m, ahead of synthesis in order to decide on the number of scan chains to be inserted for the design. Unfortunately, such estimation is not always easy as it is often difficult to guess without gate-level synthesis of the IP core. As a result, this approach can potentially cause unnecessary iterations on the design flow and

may have negative consequences on the design schedule. Moreover, it is usually not a good approach to over-constrain the design based on the test strategy used; otherwise, the reusability of the IP cores can be compromised.

In cases where the number of patterns, m, exceeds the scan chain length, n, it is possible to partition the number of patterns, m, such that each partition resembles the length of the scan chain. Figure 35 shows an example of dividing the matrix A into three sections of a, b, and c. Similar to earlier example, the resultant matrices a, b, and c are divided such that square matrices are produced with $n \approx m/3$.

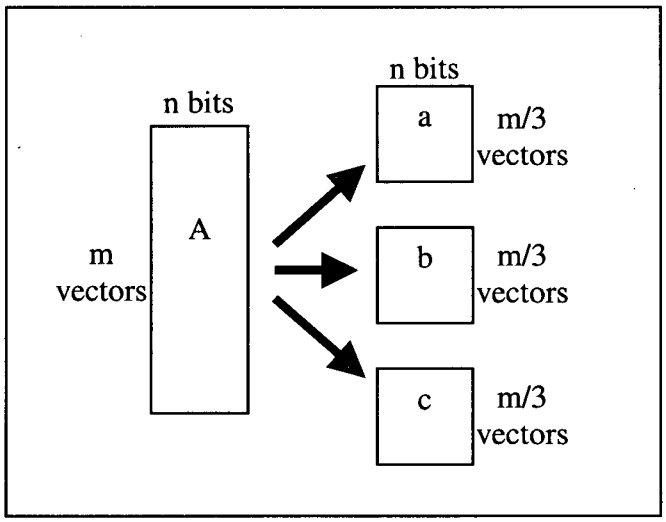


Figure 35. Horizontal test pattern split

This approach treats the three matrices a, b, and c as separate matrices and each requires an independent LFSR implementation. Each LFSR is responsible for generating the deterministic patterns specified in each matrix. This approach does not require unnecessary iterations to the design flow since the deterministic patterns are divided in accordance to the scan chain length n. Therefore, a priori estimation of the deterministic pattern count is unnecessary. As a result, the scan chain design of IP core is not

constrained by the test strategy. However, if the bits per pattern n should become too large due to lengthy scan chain, this scheme suffers from implementation of long LFSR length. Again, this is due to the fact that LFSR length is at least as long as the larger of the pattern matrix length and width dimensions.

In order to address the above mentioned issues, a deterministic pattern partitioning scheme is introduced in Section 3.4.4. This approach combines the above two methods discussed in this section to eliminate their pitfalls.

3.4.4 Improved LFSR-Based Deterministic Pattern Generation

In order to facilitate robust application of the algorithm introduced in Section 3.4.2 for calculating the feedback taps, a scheme is proposed to partition the deterministic patterns represented in matrix A. Such a scheme eliminates the issues mentioned in Section 3.4.3; it also guarantees robustness regardless of the scan chain length n and pattern count m of the deterministic pattern set. Moreover, the length of the resultant LFSR only ranges from $\lceil \sqrt{n} \rceil$ to $2\lceil \sqrt{n} \rceil$ inclusive. The schemes in Section 3.4.3 require the LFSR length to range between n to $2n - 2\frac{n}{m}$ in the first scheme; the second scheme requires the LFSR length to range between m to $2m - 2\frac{m}{n}$.

In the proposed scheme, each row in matrix A representing a single pattern is considered as a matrix. Therefore, m matrices are formed from a deterministic pattern set of m

patterns. Each of the m patterns is divided into segments of length $\lceil \sqrt{n} \rceil$. Figure 36 depicts the partitioning of a deterministic pattern set in a generalized notation.

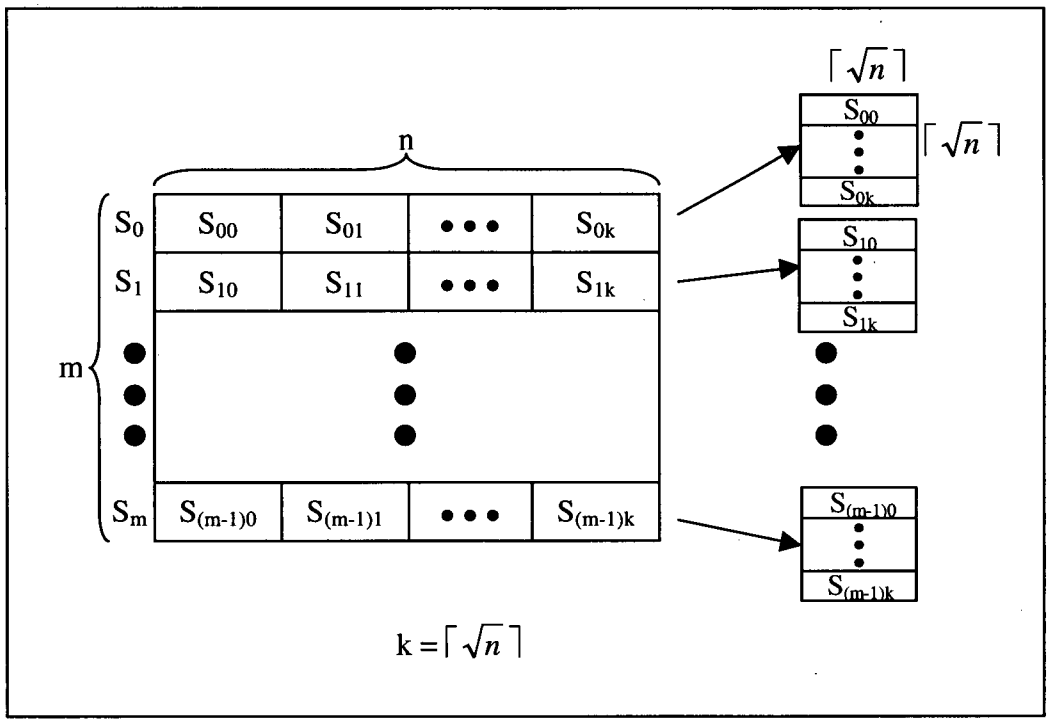


Figure 36. Deterministic pattern set segmentation

Figure 37 shows a numerical example of partitioning a 4-pattern deterministic set for an IP core with a scan chain length of 9. Four 3x3 matrices are formed as a result.

$$
\begin{array}{c}
S_0 \\ S_1 \\ S_2 \\ S_3
\end{array}
\left[\begin{array}{ccc|ccc|ccc}
1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\
1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\
1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0
\end{array}\right]
\qquad
\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} = S_0
\quad
\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix} = S_1
\quad
\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} = S_2
\quad
\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix} = S_3
$$

Figure 37. Deterministic pattern set segmentation example

Each of the matrices is treated independently with the LFSR feedback taps calculated by the algorithm outlined in Section 3.4.2. Figure 38 shows how the feedback taps for the S_0 matrix is calculated.

$$
S_0 = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}
$$

Matrix Inversion :

$$
\left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{array}\right]
\begin{array}{l} Row0 \\ Row1 \\ Row2 \end{array}
$$

$$
\left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{array}\right]
\begin{array}{l} Row0 \\ Row1 \oplus Row0 \\ Row2 \oplus Row0 \end{array}
$$

$$
\left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{array}\right]
\begin{array}{l} Row0 \\ Row1 \oplus Row2 \\ Row2 \end{array}
$$

$$
S_0^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}
$$

$$
Ts = S_0^{-1} T_c S_0
$$

$$
Ts = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}
\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}
\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}
$$

$$
Ts = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}
\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}
= \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}
$$

Figure 38. Transition matrix calculation

The LFSR feedback taps would be implemented as depicted in Figure 31, according to the T_s matrix in Figure 38. Each subsequent matrix is calculated in a similar fashion, yielding one feedback tap configuration each. Consequently, there is a total of 3 feedback tap settings. Setting S_0 corresponds to feedback tap configuration for pattern 0 and similar for the latter. When setting S_0 is selected, segments for pattern 0 are produced in the LFSR with each clock cycle. Therefore, the contents of the LFSR must be serially shifted into the IP core scan chain before the LFSR is clocked again for the next segment. Figure 39 shows a block diagram of the resulting LFSR for a set of deterministic patterns.

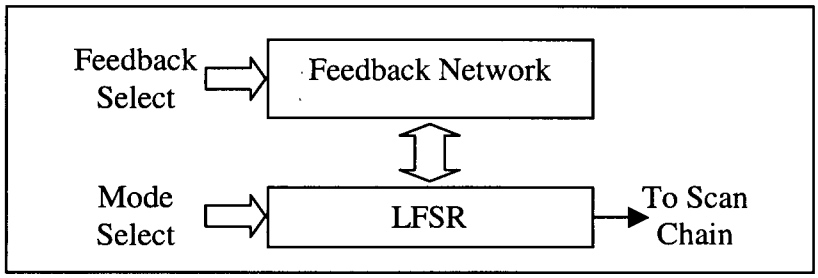


Figure 39. Block diagram of LFSR

In order to provide smooth transition from one pattern to the next, each matrix is appended with the first segment of the next pattern. The reason for having such a structure is to ensure the LFSR content is initialized to first segment of next pattern after all segments of current pattern are shifted into the IP scan chain. When the next feedback tap setting is selected, the first segment corresponding to next pattern already resides as the content of the LFSR. As a result, the segment length is determined according to the condition exemplified by the pseudo code:

$$
\begin{aligned}
&if \quad \left(n > \left(\lceil\sqrt{n}\rceil \times \left(\lceil\sqrt{n}\rceil - 1\right)\right)\right) \\
&\quad segmentLength = \lceil\sqrt{n}\rceil + 1 \\
&else \\
&\quad segmentLength = \lceil\sqrt{n}\rceil
\end{aligned}
$$

This condition ensures that the resultant matrices are as square as possible with the capability to hold one extra segment after accommodating all the bits in a single pattern. Figure 40 shows the overall structure of the deterministic pattern generator. The pattern count serves as the selection control for feedback tap settings. There is a mode selection on the shift register to select whether to serially shift contents to the IP core scan chain or take the outputs of the feedback block to produce the next segment.

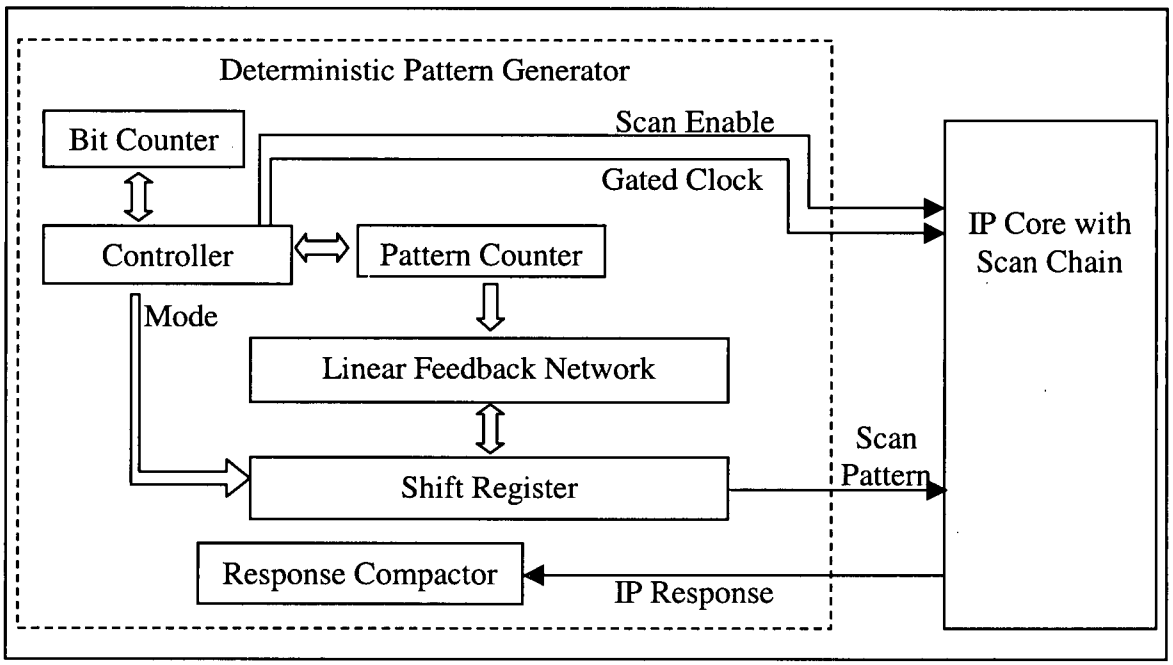


Figure 40. Proposed LFSR-based deterministic pattern generator

The reason that LFSR ranges up to $2\lceil\sqrt{n}\rceil$ is due to the matrix inversion process. Invertible matrices must be square, and linearly independent for all rows. Linear

dependency implies a row can be formed by linear combination of any other rows in the matrix [2]. Unfortunately, it is impractical to force the segments to be linearly independent from one another since they represent deterministic test pattern bits specified by the ATPG tool. When linear dependency exists, a new column is appended to the right side of the matrix to eliminate the linear dependency. Figure 41 serves as an example for such a process.

$$A = \begin{array}{c} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix} \end{array} \begin{array}{l} \\ R_1 \\ R_2 \end{array} \longrightarrow A = \begin{array}{c} \left[\begin{array}{ccc:c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ \hline 0 & 0 & 1 & 0 \end{array}\right] \end{array} \begin{array}{l} R_0 \\ R_1 \\ R_2 \\ R_3 \end{array}$$

Linearly dependent Linearly independent

Figure 41. Matrix linear dependency elimination example

As seen in the figure, R_2 of matrix A is a linear combination of R_0 and R_1 (modulus-2 addition). In order to break the linear dependency, a new column is added with a 1 in R_2 to make it linearly independent from R_0 and R_1. Since the matrix must be square in order for inversion, a new row R_3 is added with a 1 in the third column to again ensure linear independency with other rows. Such a matrix is now invertible with the addition of one extra column which translates into expansion of LFSR length by one. In the worst case, all rows in the matrix are linear combinations of the first two rows. Consequently, the LFSR length ranges up to $(\lceil \sqrt{n} \rceil+1) + (\lceil \sqrt{n} \rceil+1) - 2 = 2\lceil \sqrt{n} \rceil$.

3.5 Pattern Generator Hardware and Area Overhead

In order to facilitate the calculation for T_s matrices according to the algorithm described in Section 3.4.4, a computer program was implemented. This program takes the deterministic pattern set as input and then segments the pattern set as described. It produces the RTL (Verilog and/or VHDL) code that specifies the LFSR with all the calculated feedback settings. Since the controller design only requires minor parametric modifications according to the deterministic pattern set specifications, it is simply attached to the LFSR. Finally, these blocks are synthesized together to produce gate-level netlist. Therefore, the LFSR calculation and coding process are completely automated once the deterministic pattern set is available from the ATPG tool. Furthermore, such deterministic pattern generator can be easily integrated with the logic BIST circuitry to execute the test flow outlined in Section 2.5.

The advantages of this scheme are its simplicity in implementation and capability of being automated. Also, the required number of FF's only ranges from $\lceil \sqrt{n} \rceil$ to $2\lceil \sqrt{n} \rceil$. However, this scheme also has some disadvantages. One major concern is the amount of hardware involved in the implementation of the feedback tap configurations; one configuration is required by each pattern. This is partially justifiable. As the number of patterns m increases, the total number of bits to be reproduced (mxn) increases accordingly. It is unreasonable to demand the hardware implementing such generator to remain fixed while the number of bits to be reproduced increases. Thus, the hardware growth induced by the increase in the total number of bits in the deterministic pattern set is deemed necessary and reasonable.

The area overhead contributed by the pseudo-random and deterministic pattern generation circuitry were determined with report_area command in the Design Compiler™ synthesis tool by Synopsys®. Detailed area measurements are given in numerical format in Appendix B.

The graphical plots of the area measurements are displayed from Figure 42 to Figure 45. The plots provide area comparisons between the IP cores and the corresponding pattern generators. The area of the pattern generators are shown as bar charts to illustrate their respective contributions to the total test circuitry area. Also, the number of test patterns encoded by each pattern generator is shown in the legend of each plot.

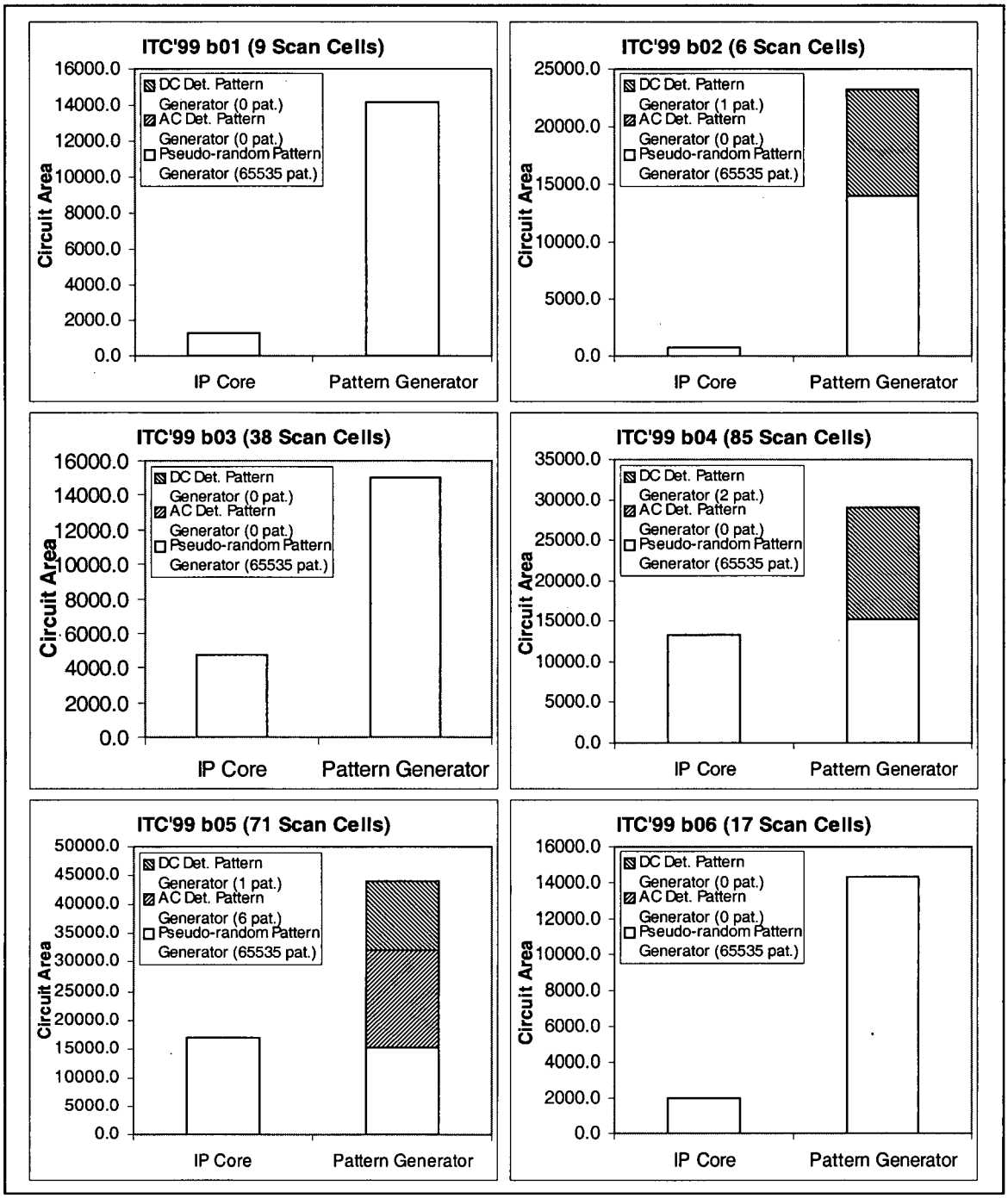


Figure 42. Test circuitry area measurement plots 1

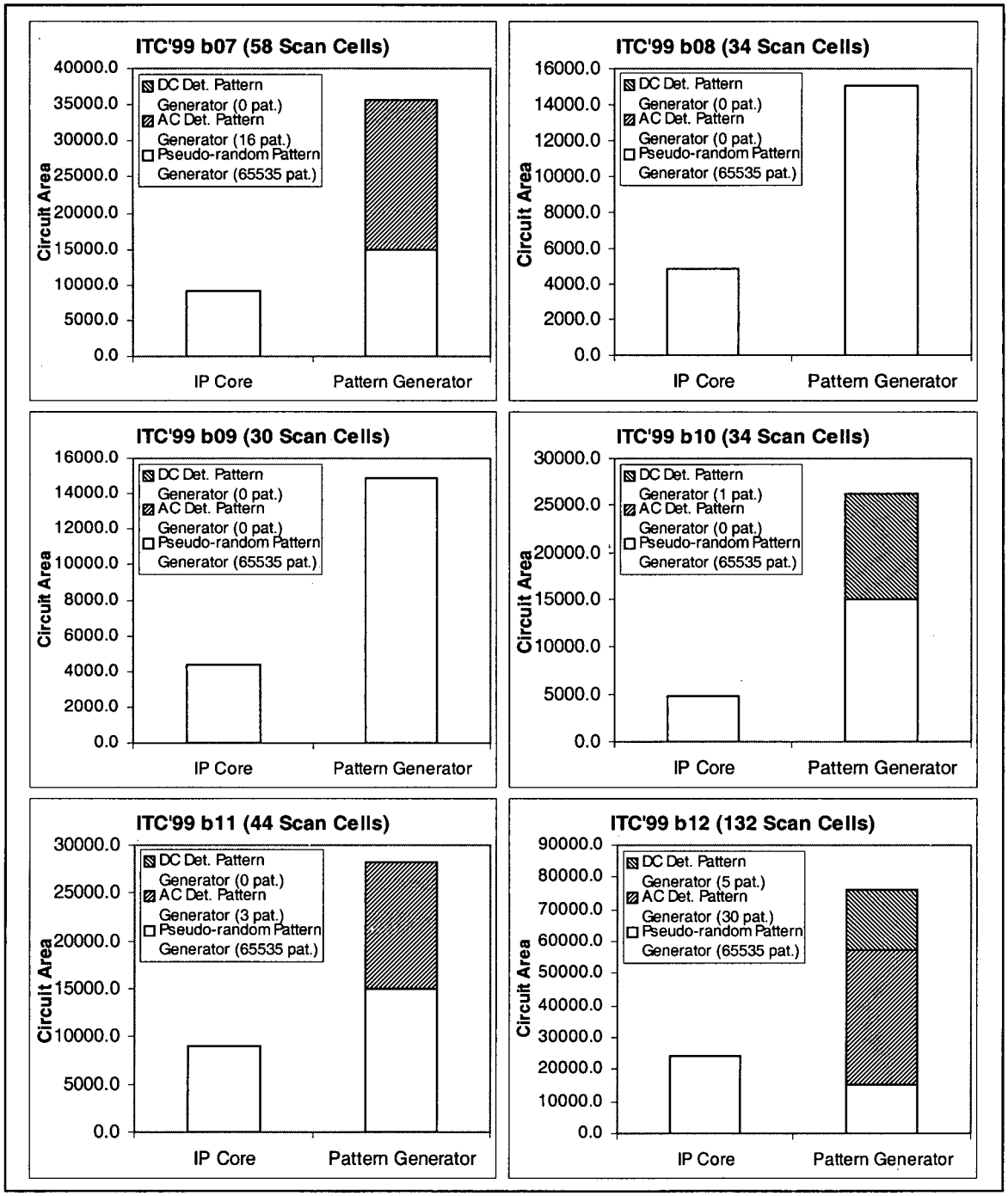


Figure 43. Test circuitry area measurement plots 2

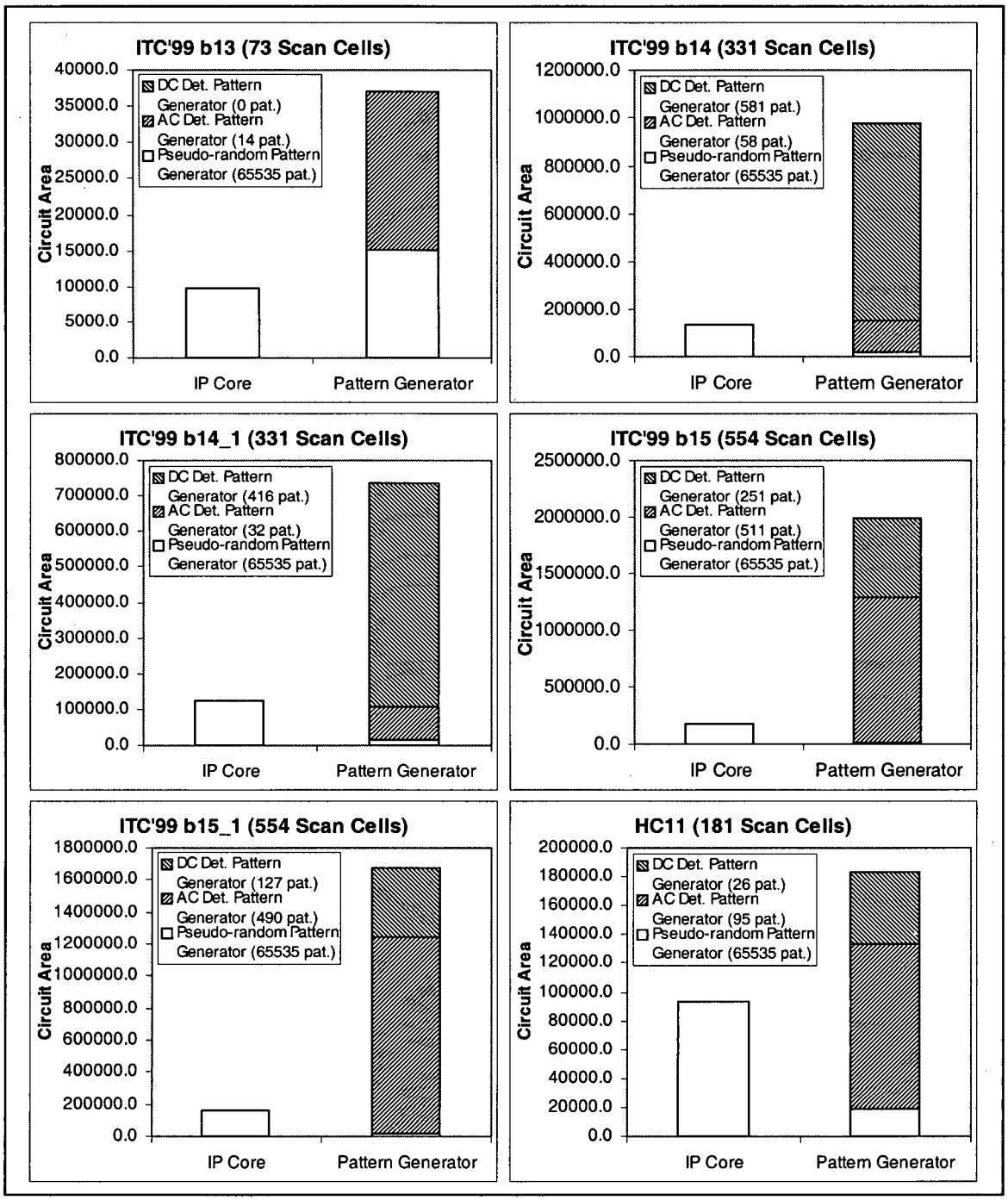


Figure 44. Test circuitry area measurement plots 3

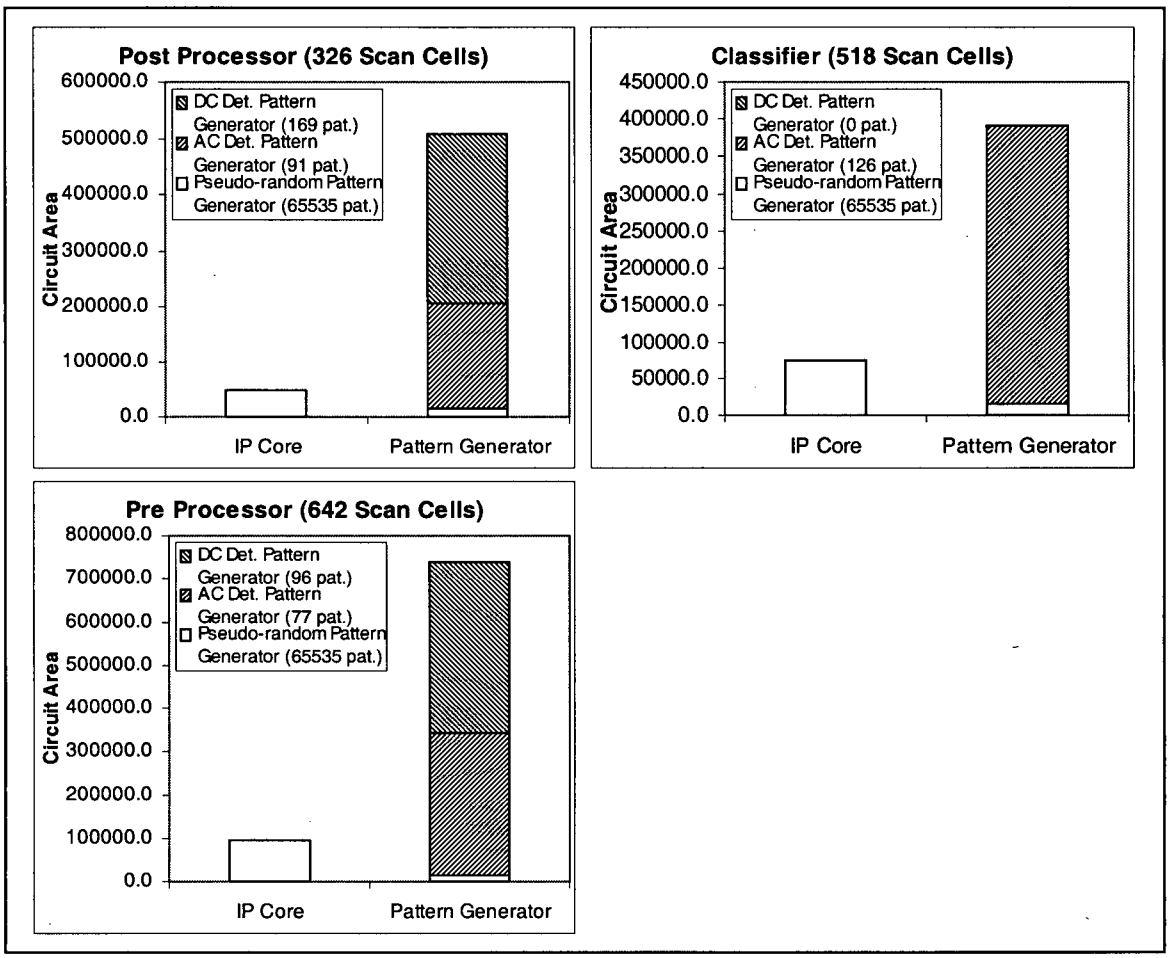


Figure 45. Test circuitry area measurement plots 4

As seen in the above plots, the area overhead required by the test circuitry are high. For the deterministic pattern generators, the area overhead measurements vary from 53.3% (HC11) to 1180.3% (ITC'99 b02). From the plots, the area overhead measurements appear to be more significant with smaller IP cores with short scan chain lengths such as ITC'99 b02. This observation is sensible because each deterministic pattern generator includes certain basic components, such as controller and response compactor, which occupy certain silicon area irrespective to size of the IP core under test. As a result, the deterministic pattern generation circuitry must occupy certain amount of area which may deem large relative to the IP core under test. Further, the number of deterministic test

patterns has a direct effect on the deterministic pattern generator area. This is apparent as more deterministic test patterns translate into more deterministic test bits. In order to generate more deterministic test bits, it is reasonable to employ more hardware and demand more silicon estate. This is analogous to using more RAM cells to store more data bits.

The AC pseudo-random pattern generation circuitry area overhead measurements suggest the AC pseudo-random pattern generation circuitry area overhead vary relative to the size of the IP core under test. They range from 9.1% (ITC'99 b15) to 1786.0% (ITC'99 b02). In general, AC pseudo-random pattern generators incur higher area overhead on smaller IP cores with shorter scan chain length. This observation is justified because each AC pseudo-random pattern generator requires certain minimum amount of silicon area for basic components such as controller, pattern generator, and response compactor. This is supported by the fact that AC pseudo-random pattern generators occupy relatively constant amount of area irrespective to the IP core under test.

From the above analysis, the test pattern generation flow implemented as logic BIST solution is more suitable for large IP cores in order to alleviate the impact of test circuitry area overhead to acceptable levels. Also, the number of deterministic top-up test patterns plays major role in the area overhead contributions. Therefore, it is desirable to minimize the number of deterministic top-up test patterns. This can be achieved by increasing the number of pseudo-random test patterns in the logic BIST circuitry in the hope of detecting more faults before involvement of the ATPG tool. With less remaining faults

in the fault list as input to the ATPG tool, the number of deterministic top-up patterns generated should theoretically be reduced.

Also, the deterministic pattern generator area can be reduced by taking advantage of the don't care test bits in the ATPG generated top-up test patterns. Thus, during manipulation of the deterministic pattern generator, greater degree of freedom can be exploited to produce a possibly simpler design with less complex LFSR feedback network connections. Such a reduction in complexity equates to reduction in the overall area of the resulting deterministic pattern generator.

Ultimately, as attempts are made to reduce the role of the off-chip ATE, the amount of silicon area devoted to on-chip test infrastructure will increase. With ITRS predictions of memory dominating the chip area [12], it is possible to claim some of this area for test purposes. In parallel, techniques to further reduce the area of on-chip deterministic pattern generators can be investigated.

CHAPTER 4 SOC TEST STRATEGY

One of the most challenging issues in SoC design is to establish the top-level test strategy for the fabricated design. In the SoC design paradigm, an IC is developed by integrating individual IP cores onto a single chip. Since SoC design promotes reuse of IP cores, they are often deployed in various design applications. As a result, IP cores must be designed to support various test access mechanisms (TAM) due to different test strategies. In order to enhance IP core development and test efforts, standards that encourage a common platform for testing are essential. Such a platform is then leveraged to devise a TAM which facilitates top-level testing.

Figure 46 shows an overview of a possible SoC test strategy with the concept of SI^2P introduced in Section 1.1. Depicted this way, the IP cores are a small part of the SoC integration process. The key point is that there is a substantial amount of work needed to design the general infrastructure for a chip, and in particular the test SI^2P.

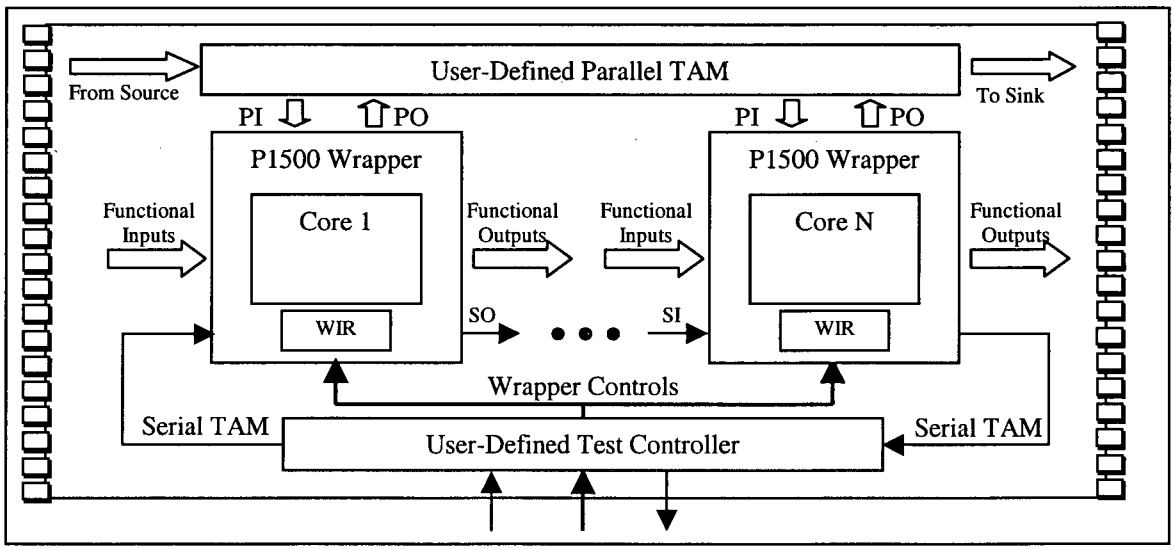


Figure 46. SoC test strategy with SI^2P

This chapter focuses on the design-for-test (DFT) infrastructure needed for SoC design. It first explains the importance of input/output (I/O) registers on individual IP cores. Then it examines the operation of the P1500 wrapper which acts as the test harness around all IP cores. As an illustration, an HC11 core is designed and fabricated with a DC logic BIST circuitry, AC logic BIST circuitry, and P1500 wrapper. Lastly, the chapter describes the supportive logic, which enables the seamless interface between the P1500 wrappers and a particular TAM implementation known as NIMA [21].

4.1 Input/Output Registers

Placing registers at the inputs and outputs (i.e., I/O buffering) for IP cores is a strongly recommended practice for the SoC design style [15]. As the size and speed of design increase, signals are given shorter time window to propagate from one point to another on the silicon. Due to aggressive design goals and complexities, signals must propagate through increasingly more stages of system logic within a given time frame. Therefore, the increase in risks of designs not capable of fulfilling timing constraints becomes inevitable. I/O buffering offers synchronous inputs and outputs to/from an IP core by placing FF's at all signals around the core except some special signals such as the system clock and reset. Each input signal is connected to the system logic through a FF. Similarly, each output signal leaves the system logic by connecting to a FF which in turn drives the signal from the IP core. Figure 47 illustrates the concept of I/O buffering.

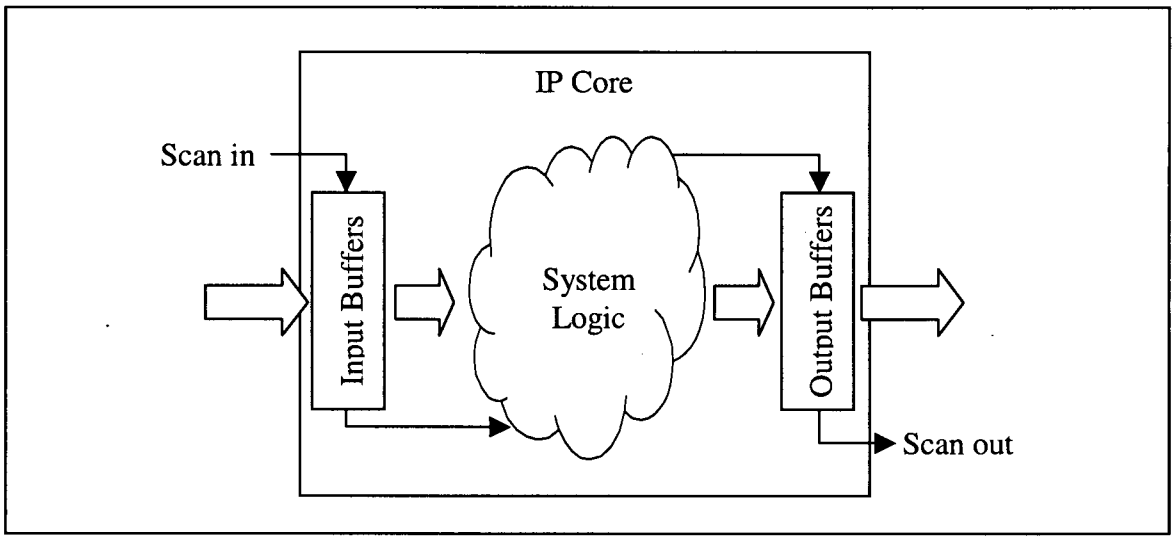


Figure 47. I/O buffering

The addition of I/O buffers enables localized timing on each IP core such that timing budgets can be analyzed and estimated on a core-to-core basis. As a result, specific setup and hold times may be established for all inputs and outputs. This is essential for IP core development because they are often designed as standalone modules which may interact with any other modules not known at design time. Furthermore, it is important for each core to achieve timing closure independently. Since they are often reused in many designs, their ability to meet and guarantee their own specified timing requirements are imperative to overall functionality of the final integrated SoC. These I/O registers act as a boundary for an IP core such that any timing error within the IP core can be intercepted and isolated from the rest of the system. Consequently, timing delays occurring in one IP core does not affect the timing of another. This makes the SoC design style manageable and predictable because timing related issues can be identified at early stages with a bottom-up design style.

From the point of view of DFT and scan deign, I/O buffering required for timing can be exploited for testing on IP cores. Since the PI's and PO's of individual cores are buried deep in the SoC architecture, they often cause accessibility problems with respect to external ATE's. With I/O registration, the I/O buffers are included as part of the scan chain. As a result, the PI's and PO's are indirectly controlled through the scan chain. Since scan design is the most widely accepted and efficient method thus far to test sequential circuits, at least one scan chain must exist in an IP core. Thus, the efforts involved in including the I/O buffers in a scan chain are minimal. However, the gains in controllability on the IP core inputs and observability on the IP core outputs are enormous when these inputs and outputs may not even be accessible at all if I/O registers were not deployed.

4.2 P1500 Core Wrapper

To serve as a standard platform for testing IP cores in the SoC environment, the P1500 standard is being drafted by IEEE to specifically target embedded core testing [31]. Its main role is to define a uniform interface that allows all P1500-compliant IP cores to precisely exchange test-related signals with any test access mechanism (TAM) designed according to the P1500 standard. As a result, IP cores can be simply viewed as plug-and-play modules which can be replaced by one another with minimal compatibility concerns on test harnesses.

The P1500 standard is modeled after the IEEE JTAG 1149.1 standard for board-level testing, also known as boundary scan [31][26][23]. P1500 can be considered as a scaled

version of the JTAG standard; IP cores are viewed as individual IC chips while the entire SoC is viewed as the system board. Despite the similarity in concept between the two standards, there are notable distinctions between them. These variations provide more flexibility to P1500 compliant IP cores without sacrificing the robustness offered in the JTAG standard. One of the most prominent differences between the two standards is the absence of the test access port (TAP) controller from the P1500 standard. This controller is a FSM and is an integral part of the JTAG standard to enforce strict signal sequencing when communicating with the test harnesses. With the absence of such a controller, the P1500 standard relies on the TAM designer to provide the proper signal sequences for test information correspondence. Another notable difference is the addition of parallel test port. The JTAG standard stipulates only one pair of test data input (TDI) and test data output (TDO), which limits test data throughput. On the other hand, the P1500 standard allows possibility for increased throughput with a variable number of test data inputs and outputs. This allows the opportunity for realizing parallelism in test data movement when speedy transportation of test data is demanded.

The P1500 standard is realized by a wrapper design around an IP core. The wrapper contains six mandatory control inputs, one serial data input and one serial data output. Optionally, the wrapper can include variable numbers of inputs and outputs for parallel test data movement. Figure 48 presents an overview of the P1500 wrapper [24].

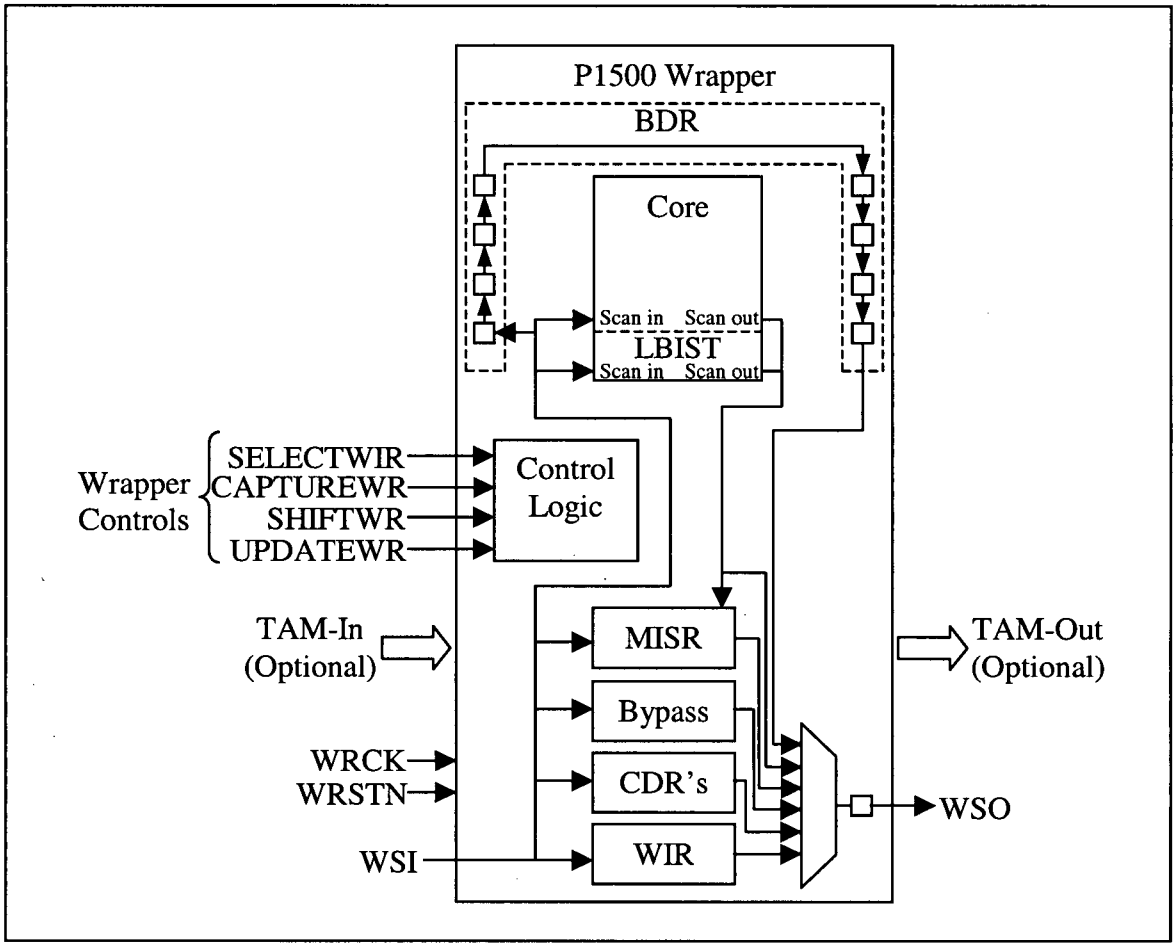


Figure 48. P1500 wrapper

The WRCK and WRSTN pins are the wrapper clock and reset respectively to provide the corresponding signals to wrapper logic. The wrapper serial input (WSI) is responsible for transporting instructions and serial data into the wrapper. The SELECTWIR input determines whether the signal on the WSI input is serially shifted to the wrapper instruction register (WIR) or other destinations inside the wrapper. Based on the content in WIR and the four control inputs, various internal control signals are generated to control other components within the wrapper. The ring of registers around the wrapper comprises the boundary data register (BDR) which isolates the core from its functional connections in the SoC during test. Before and during the exercise of full-scan test or

operation of logic BIST circuitry on the core, the BDR is set to a known state in order to provide a predictable test environment. During full-scan core test, test patterns are serially shifted into the WSI pin while the scan responses are serially shifted out of the WSO pin. The scanned-out bits of the full-scan test can also be compacted at the multiple-input signature recognizer (MISR) for a signature which is captured by a core data register (CDR) to be serially shifted out from the WSO output. WSO acts as the sole output from which core and wrapper responses are exported. In an attempt to model the JTAG standard, the WSO pin changes state at the falling edge of WRCK while all other wrapper logic are triggered on the rising edge. As a result, a negative-edge triggered FF is implemented to buffer the WSO output. The CDR's are designer definable and are instantiated as many times as deemed necessary. One major function of the CDR is to provide and capture the BIST or MISR signature after its operation. The bypass register is responsible for transporting the signal from WSI input to WSO output verbatim.

The architecture of each BDR cell is not strictly specified in the P1500 standard and can be varied upon different requirements. The implementation used in this research consists of two FF's and two MUX's as illustrated in Figure 49.

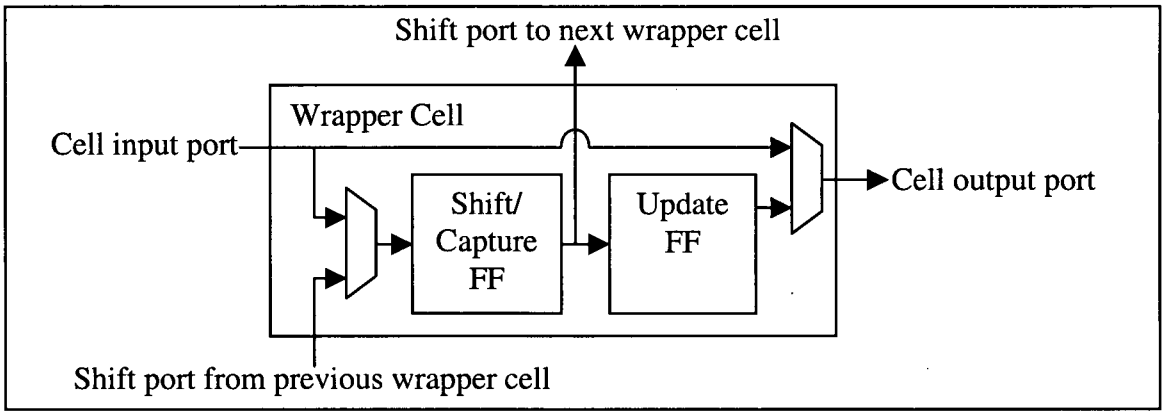


Figure 49. Wrapper cell structure

The structure is similar to that specified in the JTAG standard [22]. The update FF keeps the data connected to the functional output port stable while shifting occurs between the shift/capture FF's in adjacent cells. The content of shift/capture FF is sent to the update FF only after shifting in the BDR completes. This ensures signals conceived by the IP core inputs are unaltered while the desired contents are being shifted into the BDR. This is a precaution against adverse effects on the core while BDR shifting. During BDR shift operations, the MUX in front of the shift/capture FF selects the shift out port from the previous BDR cell to form a shift register chain. During BDR capture operation, the MUX is configured to capture the cell functional input such that the value is stored and then exported by later shift operation. Depending on WIR content and the control inputs, the MUX at the cell functional output selects either functional input or update FF. BDR becomes non-invasive when each cell is bypassed by connection of the functional input to functional output through this MUX.

4.3 Fabrication of Core Level SI^2P Solution

In order to demonstrate the use of the aforementioned concepts, an IP core is designed and fabricated. The implemented IP core is a microcontroller modeled after the functionalities of an HC11 microcontroller by Motorola®. It is equipped with DC and AC pseudo-random pattern generation circuitry to provide a feasibility test for incorporating the *double-clocking* scheme of Section 3.3.2 in silicon. The IP core and the two logic BIST circuitry are wrapped by a P1500 wrapper as described in Section 4.2. This design was taken through the physical design flow for 0.18μm CMOS and was sent for fabrication.

Figure 50 is an overview of the fabricated design. The HC11 core is implemented with two logic BIST circuitry blocks. One logic BIST circuitry is responsible for generating pseudo-random test patterns targeted for DC faults while the other is targeted for AC faults. The HC11 core and logic BIST circuitry blocks are wrapped according to the P1500 standard. The HC11 core area is approximately equivalent to 7542 gates (The area of a 2-input NAND gate in this library is 11.53μm^2 and is used as a reference for gate count measurements). The DC logic BIST circuitry block occupies an area of about 1518 gates while its AC counterpart occupies an area of about 1526 gates. Each equates to approximately 20% of the HC11 core area. The P1500 wrapper accounts for additional 4350 gates and is equivalent to 58% of the HC11 core area. The area taken up by the SI^2P components are large relative to the size of the HC11 core. However, given the size of the HC11 core is small by itself, the area overhead incurred by the SI^2P components are magnified as a result. Figure 51 is a die photo of the fabricated design

and testing of this is still in progress. A logic simulation of the test sequence is available on the gate-level netlist of the design. Stimuli and responses of the design are collected and converted to external test patterns to be applied from the ATE.

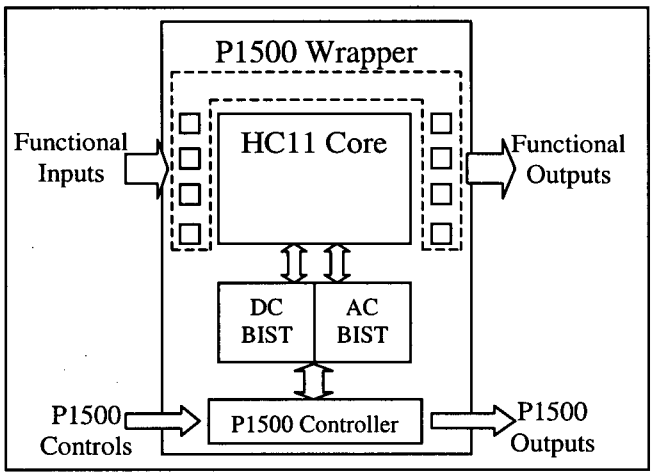


Figure 50. Overview of the fabricated HC11 with core level SI^2P solution

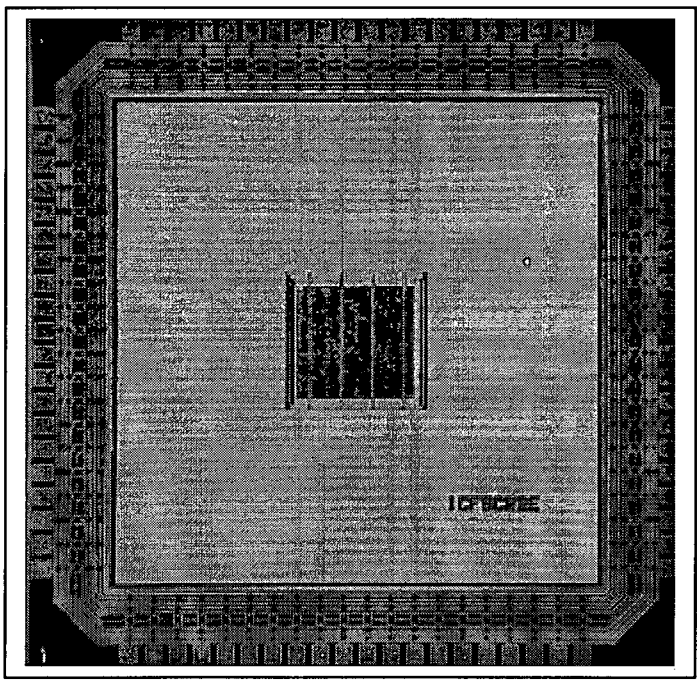


Figure 51. Die photo of the fabricated HC11 with core level SI^2P solution

4.4 Application of P1500 Core Wrapper

The next stage of the SI^2P design is the chip level integration of the P1500-wrapped IP cores with a packet-switched test network known as NIMA [21][10]. Additional SI^2P components are required as adaptors for interfacing with NIMA test network.

The NIMA test network is deployed in a network processor designed in-house at UBC. It is fabricated and included as part of another research project at UBC [9]. A paper is also published describing the deployment and performance of this NIMA test network [10]. The in-depth implementation details are given in Appendix C.

CHAPTER 5 CONCLUSIONS

Intellectual property (IP) cores, which are the building blocks for System-on-a-Chip (SoC) design, are often purchased from third-party vendors and integrated together by system architects to construct the final SoC design. IP cores released with certain self-test mechanisms are highly recommended to relieve the test development burden from the system architects who may not have enough internal knowledge of the IP cores to effectively devise suitable test methodologies. For this purpose, logic built-in self-test (logic BIST) is an attractive solution. The logic BIST concept is realized by inclusion of an extra circuitry which tests an IP core and reports the result.

The first part of this research investigates the relationship between the stuck-at (DC) and transition (AC) faults. These commonalities are then justified and analyzed to support the proposal of a heuristic test pattern generation flow which exploits this relationship to reduce the test efforts. It utilizes the AC test pattern to detect not only the AC faults, but also the DC faults. Therefore, the number of test patterns which strictly target DC faults can be reduced. Also in the proposed test flow, pseudo-random test patterns, sequenced according to the *double-clocking* method, are applied to the IP core under test first to detect the easy-to-catch AC and DC faults. The remaining AC faults are then detected with the AC deterministic test patterns generated by an ATPG tool. After the AC pseudo-random and AC deterministic test patterns are applied to test for the DC faults, an ATPG tool is again employed to assemble additional DC deterministic patterns which target the remaining DC faults. This scheme effectively cuts down the number of both AC and DC deterministic test patterns necessary to test an IP core. Analysis shows that

the proposed test flow achieves comparable, in many instances better, fault coverage results to that which simply relies on the ATPG tool to generate deterministic test patterns. The DC and AC fault coverage results were obtained from TetraMAX™. It should be noted that each tool has some limitations in algorithms and implementation. Therefore, different results may be observed with different ATPG and fault simulation tools.

The second part of this research focuses on deployment of the test pattern generation flow as a logic BIST scheme. It involves on-chip generation of pseudo-random test patterns and ATPG tool specified AC and DC deterministic test patterns. For deterministic test pattern generation, a scheme based on a type 2 linear feedback shift register (LFSR) is proposed. The LFSR feedback tap locations are systematically identifiable with matrix manipulations. The deterministic patterns are partitioned into segments which are reorganized into matrices. These matrices are then manipulated to identify the feedback tap locations which are capable of generating the specified segments when the LFSR is sequenced correctly. Due to the regularities of these operations, a computer program is written to facilitate the matrix manipulation efforts. The RTL code, Verilog or VHDL, for the LFSR can be generated according to a given set of deterministic test patterns. Area overhead measurements of the deterministic pattern generators are analyzed to conclude that the area overhead results are more significant for smaller IP cores. This is due to the fact that at least certain minimum amount of area must be occupied by some basic components necessary for the deterministic pattern generation. Furthermore, it is observed that the number of deterministic test patterns to be generated greatly affect the

area overhead measurements. More deterministic test patterns translate into greater area overhead. Therefore, it is advisable to apply the logic BIST test flow to large IP cores to absorb the minimum area requirements of the test circuitry. Moreover, the deterministic test pattern count should be kept at a minimum to aid in area requirement reduction.

The third part of the research involves the realization of Semiconductor Infrastructure IP (SI^2P) in a SoC environment. This includes the implementation of test harnesses around IP cores to interface with a packet switched test access mechanism (TAM) known as NIMA. As a result, IP cores are encapsulated by wrappers which are implemented according to the P1500 standard. The implementation serves as a feasibility test for the emerging P1500 standard. In order to control each P1500 wrapper with ease, a state machine which resembles the TAP controller from the IEEE1149.1 JTAG standard is designed to properly sequence the P1500 wrapper control signals. For the purposes of communication with the packet switched NIMA network, the NIMA interface and NIMA serializer are implemented as adapters to bridge the NIMA network and the IP cores. All of these components are implemented as reusable modules that can be replicated in different SoC designs. They are imperative to the concept of SoC design since they offer modularity and reusability. Without them, SoC testing becomes a difficult and unmanageable task which possibly impedes the progress of the SoC design paradigm.

5.1 Future Work

The future work involves testing of a fabricated design of the HC11 IP core with AC and DC logic BIST circuitry. Test vectors and waveforms are readily available for the task.

The LFSR responsible for generating on-chip deterministic test patterns can be optimized by exploiting the don't care bits in the deterministic test pattern set generated by the ATPG tool. In the hope of providing greater degree of freedom during matrix manipulations, the LFSR feedback network can be simplified to occupy less valuable silicon area.

The area overhead of the total test circuitry can be reduced by sharing common components, such as counters and response compactors, amongst pseudo-random and deterministic pattern generators. Further, controllers of the AC and DC deterministic pattern generators can potentially be merged to minimize hardware resources. Finally, the test circuitry and harnesses can be upgraded with minimal modifications to support multiple scan chains in an IP core when one scan chain is deemed inadequate.

5.2 Contributions

This section summarizes the contributions of this research:

1. Identified relationship between DC and AC faults to propose a heuristic test pattern generation flow which potentially reduces the total number of test patterns required to test an IP core. The test flow is applicable for AC and DC fault detection and takes advantage of pseudo-random test pattern generation to reduce the deterministic test pattern count.
2. Conducted fault coverage analyses on the test pattern generation flow with the TetraMAX™ ATPG tool from Synopsys® to demonstrate that high fault coverage can be obtained.

3. Devised an LFSR-based scheme for on-chip deterministic test pattern generation.
4. Facilitated the design of hardware for deterministic test pattern generation by writing a computer program to automatically generate the RTL code for the LFSR given a deterministic test pattern set.
5. Implemented test harnesses for interface between IP cores and a packet-switched test access mechanism (TAM) known as NIMA to strengthen the concept of SI^2P.

REFERENCES

[1] "AMBA Specification Rev 2.0." ARM Ltd., 1999 <http://www.arm.com>.

[2] R.F.V. Anderson, *Introduction to Linear Algebra*, Holt, Rinehart and Winston of Canada, New York, 1986.

[3] M.L. Bushwell and V.D. Agrawal, *Essential of Electronic Testing for Digital, Memory & Mixed-Signal VLSI Circuits*, Kluwer Academic Publishers, Boston, 2000.

[4] R. Chandramouli and S.Pateras, "Testing Systems on a Chip", IEEE Spectrum, 1996, pp. 42-47.

[5] "CoreConnect Bus Architecture." International Business Machines Corporation, 1999 <http://www.chips.ibm.com/products/coreconnect>.

[6] A.L. Crouch, *Design-for-Test for Digital ICs and Embedded Core Systems*, Prentice Hall, New Jersey, 1999.

[7] C. Dufaza and G. Cambon, "LFSR based Deterministic and Pseudo-random Test Pattern Generator Structures", 2nd European Test Conference, 1991, pp. 27-34.

[8] S. Haykin, *Communication Systems Third Edition*, John Wiley & Sons, Inc., New York, 1994.

[9] L. Hong, "System-on-a-Chip (SoC) Design and Test – A Case Study", M.A.Sc. Thesis, University of British Columbia, August 2002.

[10] L. Hong, M. Nahvi, R. Fung, A. Ivanov, and R. Saleh, "Novel Test Methodologies for SoC/IP Design", IEEE International Workshop on System-on-Chip for Real-Time Applications, 2002, pp20-30.

[11] A. Ivanov, "UBC System-On-Chip Annual Review", University of British Columbia, 2002.

[12] The International Technology Roadmap for Semiconductors: Design Chapter, International Sematech, Texas, 2001, <http://public.itrs.net>.

[13] The International Technology Roadmap for Semiconductors: Test and Test Equipment Chapter, International Sematech, Texas, 2001, <http://public.itrs.net>.

[14] S.M. Kang and Y. Leblebici, *CMOS Digital Integrated Circuits: Analysis and Design, Second Edition*, McGraw-Hill, Boston, 1999.

[15] M. Keating and P. Bricaud, *Reuse Methodology Manual, Second Edition*, Kluwer Academic Publishers, Boston, 1999.

[16] G. Keifer, H. Vranken, E.J. Marinissen, and H-J Wunderlich, "Application of Deterministic Logic BIST on Industrial Circuits", International Test Conference, 2000, pp. 105-114.

[17] G. Kiefer and H-J Wunderlich, "Deterministic BIST with Multiple Scan Chains", International Test Conference, 1998, pp. 1057-1064.

[18] G. Kiefer, and H-J Wunderlich, "Using BIST Control for Pattern Generation", International Test Conference, 1997, pp. 347-355.

[19] A. Krstić and T. Cheng, *Delay Fault Testing for VLSI Circuits*, Kluwer Academic Publishers, Boston, 1998.

[20] N. Mukherjee, T.J. Chakraborty, and S. Bhawmik, "A BIST Scheme for the Detection of Path-Delay Faults", *International Test Conference*, 1998, pp. 422-431.

[21] M. Nahvi and A. Ivanov, "A Packet-Switching Communication-Based Test Access Mechanism for System Chips", IEEE European Test Workshop, 2001.

[22] K.P. Parker, *The Boundary-Scan Handbook: Analog and Digital, Second Edition,* Kluwer Academic Publishers, Boston, 1998.

[23] J.M. Rabaey, *Digital Integrated Circuits: A design Perspective*, Prentice Hall, New Jersey, 1996.

[24] M. Richhetti, "Overview of Proposed IEEE P1500 Scalable Architecture for Testing Embedded Cores", Slides of the presentation at Design Automation Conference, June 20, 2001, pp. 1-26.

[25] R. Saleh. Notes, ms. University of British Columbia, August 2002.

[26] M.J.S. Smith, *Application-Specific Integrated Circuits*, Addison-Wesley, Boston, 1997.

[27] N. Tendolkar, R. Woltenberg, R. Raina, X. Lin, B.Swanson, and G. Aldrich, "Scan-Based At-Speed Testing for the Fastest Chips", Mentor Graphics Corporation Technical publications, June 2001 <http://www.mentor.com/dft>.

[28] "TetraMAX ATPG User Guide: Chapter 8", Synopsys Inc., 2001, pp. 2-10.

[29] B. Vasudevan, D.E. Ross, M. Gala, and K.L. Watson, "LFSR Based Deterministic Hardware for At-Speed BIST", *VLSI Test Symposium*, 1993, pp. 201-207.

[30] Website, <http://www.cad.polito.it/tools/itc99.html>.

[31] Website, <http://grouper.ieee.org/groups/1500>.

[32] Website, <http://www.intel.com/research/silicon/mooreslaw.htm>.

[33] M.J.Y. Williams and J.B. Angell, "Enhancing Testability of Large-Scale Integrated Circuits via Test Points and Additional Logic", IEEE Transaction on Computers, Vol. C-22, no. 1, January 1973, pp. 46-60.

[34] F. Wolff and C. Papachristou, "Multiscan Based Test Compression and Decompression Using LZ77", International Test Conference, 2002, pp. 331-339.

[35] H-J Wunderlich and G. Kiefer, "Bit-Flipping BIST", Digest of Technical Papers in International Conference on Computer-Aided Design, 1996, pp. 337-343.

APPENDIX A NUMERICAL FAULT COVERAGE RESULTS

The appendix presents the numerical fault coverage results corresponding to the fault coverage plots of the IP cores in Section 2.6.

Table 2. Numerical fault coverage results

	Pattern Count	Fault Coverage (%)	
		Collapsed	Uncollapsed
ITC'99 b01 (9 scan cells)			
AC ATPG unconstrained	16	88.36	90.14
AC ATPG constrained	14	73.71	72.79
AC pseudo-random	65535	74.57	73.47
AC pseudo-random + Det.	65535 + 0	74.57	73.47
DC ATPG unconstrained	13	95.07	95.92
DC ATPG constrained	12	93.10	93.88
DC pseudo-random	65535	94.09	94.56
DC pseudo-random + Det.	65535 + 0	94.09	94.56
ITC'99 b02 (6 scan cells)			
AC ATPG unconstrained	13	85.21	86.78
AC ATPG constrained	10	73.94	74.71
AC pseudo-random	65535	73.94	74.71
AC pseudo-random + Det.	65535 + 0	73.94	74.71
DC ATPG unconstrained	11	94.44	94.83
DC ATPG constrained	10	93.65	93.68
DC pseudo-random	65535	92.86	93.10
DC pseudo-random + Det.	65535 + 1	93.65	93.68
ITC'99 b03 (38 scan cells)			
AC ATPG unconstrained	25	86.04	87.25
AC ATPG constrained	25	83.48	83.96
AC pseudo-random	65535	84.62	84.97
AC pseudo-random + Det.	65535 + 0	84.62	84.97
DC ATPG unconstrained	22	93.67	94.32
DC ATPG constrained	21	91.87	92.30
DC pseudo-random	65535	92.77	93.06
DC pseudo-random + Det.	65535 + 0	92.77	93.06
ITC'99 b04 (85 scan cells)			
AC ATPG unconstrained	57	84.37	85.91
AC ATPG constrained	6	47.57	45.87
AC pseudo-random	65535	52.79	51.64
AC pseudo-random + Det.	65535 + 0	52.79	51.64
DC ATPG unconstrained	54	94.80	95.57
DC ATPG constrained	61	93.62	94.31
DC pseudo-random	65535	94.29	94.73
DC pseudo-random + Det.	65535 + 2	94.39	94.88
ITC'99 b05 (71 scan cells)			
AC ATPG unconstrained	94	69.78	70.65
AC ATPG constrained	94	66.95	68.22
AC pseudo-random	65535	69.04	70.04

AC pseudo-random + Det.	65535 + 6	69.34	70.28
DC ATPG unconstrained	46	80.02	81.76
DC ATPG constrained	44	77.58	80.21
DC pseudo-random	65541	79.99	81.72
DC pseudo-random + Det.	65541 + 1	80.02	81.74
ITC'99 b06 (17 scan cells)			
AC ATPG unconstrained	14	81.52	83.17
AC ATPG constrained	7	63.04	64.18
AC pseudo-random	65535	66.03	66.83
AC pseudo-random + Det.	65535 + 0	66.03	66.83
DC ATPG unconstrained	15	93.87	94.71
DC ATPG constrained	15	90.18	91.35
DC pseudo-random	65535	93.25	93.75
DC pseudo-random + Det.	65535 + 0	93.25	93.75
ITC'99 b07 (58 scan cells)			
AC ATPG unconstrained	54	81.13	82.61
AC ATPG constrained	51	77.54	79.15
AC pseudo-random	65535	75.93	76.87
AC pseudo-random + Det.	65535 + 16	79.33	80.71
DC ATPG unconstrained	41	94.80	95.82
DC ATPG constrained	40	93.70	94.93
DC pseudo-random	65551	94.73	95.71
DC pseudo-random + Det.	65551 + 0	94.73	95.71
ITC'99 b08 (34 scan cells)			
AC ATPG unconstrained	41	76.35	77.10
AC ATPG constrained	40	71.31	70.65
AC pseudo-random	65535	73.30	72.39
AC pseudo-random + Det.	65535 + 0	73.30	72.39
DC ATPG unconstrained	37	93.97	94.68
DC ATPG constrained	38	91.78	92.13
DC pseudo-random	65535	92.74	92.94
DC pseudo-random + Det.	65535 + 0	92.74	92.94
ITC'99 b09 (30 scan cells)			
AC ATPG unconstrained	25	82.60	84.78
AC ATPG constrained	25	81.08	82.90
AC pseudo-random	65535	81.63	83.49
AC pseudo-random + Det.	65535 + 0	81.63	83.49
DC ATPG unconstrained	25	94.64	95.78
DC ATPG constrained	23	94.48	95.55
DC pseudo-random	65535	94.48	95.55
DC pseudo-random + Det.	65535 + 0	94.48	95.55
ITC'99 b10 (34 scan cells)			
AC ATPG unconstrained	41	79.72	81.32
AC ATPG constrained	22	55.99	54.84
AC pseudo-random	65535	57.83	56.42
AC pseudo-random + Det.	65535 + 0	57.83	56.42
DC ATPG unconstrained	40	94.63	95.75
DC ATPG constrained	39	91.81	92.59
DC pseudo-random	65535	92.62	93.18
DC pseudo-random + Det.	65535 + 1	93.15	93.58
ITC'99 b11 (44 scan cells)			
AC ATPG unconstrained	65	77.89	78.79

AC ATPG constrained	65	76.25	76.78
AC pseudo-random	65535	78.63	78.88
AC pseudo-random + Det.	65535 + 3	79.02	79.25
DC ATPG unconstrained	58	94.04	93.04
DC ATPG constrained	58	92.84	91.92
DC pseudo-random	65538	93.54	92.38
DC pseudo-random + Det.	65538 + 0	93.54	92.38
ITC'99 b12 (132 scan cells)			
AC ATPG unconstrained	260	83.71	84.71
AC ATPG constrained	254	79.63	80.02
AC pseudo-random	65535	77.61	77.63
AC pseudo-random + Det.	65535 + 30	80.53	80.75
DC ATPG unconstrained	112	96.09	96.90
DC ATPG constrained	108	95.70	96.53
DC pseudo-random	65565	95.83	96.60
DC pseudo-random + Det.	65565 + 5	95.96	96.71
ITC'99 b13 (73 scan cells)			
AC ATPG unconstrained	33	73.34	73.98
AC ATPG constrained	33	69.43	69.03
AC pseudo-random	65535	69.89	69.03
AC pseudo-random + Det.	65535 + 14	71.35	70.68
DC ATPG unconstrained	28	93.50	93.75
DC ATPG constrained	24	91.46	91.59
DC pseudo-random	65549	92.77	92.61
DC pseudo-random + Det.	65549 + 0	92.77	92.61
ITC'99 b14 (331 scan cells)			
AC ATPG unconstrained	1068	90.55	91.99
AC ATPG constrained	273	65.18	65.43
AC pseudo-random	65535	74.49	74.43
AC pseudo-random + Det.	65535 + 58	75.99	76.01
DC ATPG unconstrained	1000	97.12	98.21
DC ATPG constrained	1014	96.53	97.79
DC pseudo-random	65593	88.99	87.80
DC pseudo-random + Det.	65593 + 581	96.98	98.05
ITC'99 b14_1 (331 scan cells)			
AC ATPG unconstrained	847	78.50	79.47
AC ATPG constrained	194	59.17	58.95
AC pseudo-random	65535	65.19	64.85
AC pseudo-random + Det.	65535 + 32	66.48	66.18
DC ATPG unconstrained	724	85.74	84.85
DC ATPG constrained	708	85.12	84.41
DC pseudo-random	65567	79.54	76.71
DC pseudo-random + Det.	65567 + 416	85.60	84.69
ITC'99 b15 (554 scan cells)			
AC ATPG unconstrained	907	77.88	77.91
AC ATPG constrained	858	72.18	72.68
AC pseudo-random	65535	55.88	56.16
AC pseudo-random + Det.	65535 + 511	72.83	72.94
DC ATPG unconstrained	723	95.94	95.89
DC ATPG constrained	725	95.37	95.46
DC pseudo-random	66046	93.30	92.80
DC pseudo-random + Det.	66046 + 251	95.87	95.81

ITC'99 b15_1 (554 scan cells)			
AC ATPG unconstrained	909	80.81	82.31
AC ATPG constrained	832	74.38	76.43
AC pseudo-random	65535	61.50	63.99
AC pseudo-random + Det.	65535 + 490	76.15	77.96
DC ATPG unconstrained	648	96.60	97.16
DC ATPG constrained	653	96.02	96.69
DC pseudo-random	66025	94.86	95.58
DC pseudo-random + Det.	66025 + 127	96.53	97.06
HC11 (181 scan cells)			
AC ATPG unconstrained	679	81.36	82.14
AC ATPG constrained	346	51.99	53.99
AC pseudo-random	65535	55.81	57.41
AC pseudo-random + Det.	65535 + 95	57.53	59.23
DC ATPG unconstrained	436	96.61	97.37
DC ATPG constrained	431	94.79	95.67
DC pseudo-random	65630	94.74	95.45
DC pseudo-random + Det.	65630 + 26	95.27	96.01
Post Processor (326 scan cells)			
AC ATPG unconstrained	251	74.70	75.54
AC ATPG constrained	115	55.25	52.41
AC pseudo-random	65535	44.63	39.84
AC pseudo-random + Det.	65535 + 91	56.31	52.76
DC ATPG unconstrained	209	94.24	95.74
DC ATPG constrained	213	87.01	88.36
DC pseudo-random	65626	78.33	73.07
DC pseudo-random + Det.	65626 + 169	89.42	90.05
Classifier (518 scan cells)			
AC ATPG unconstrained	317	84.23	85.09
AC ATPG constrained	242	77.35	78.05
AC pseudo-random	65535	72.10	73.01
AC pseudo-random + Det.	65535 + 126	79.59	80.08
DC ATPG unconstrained	141	95.08	95.54
DC ATPG constrained	136	93.90	94.24
DC pseudo-random	65661	94.67	94.87
DC pseudo-random + Det.	65661 + 0	94.67	94.87
Pre Processor (642 scan cells)			
AC ATPG unconstrained	289	70.19	69.09
AC ATPG constrained	128	58.50	57.24
AC pseudo-random	65535	59.00	56.45
AC pseudo-random + Det.	65535 + 77	62.33	60.60
DC ATPG unconstrained	319	94.76	95.35
DC ATPG constrained	322	93.57	94.30
DC pseudo-random	65612	93.33	92.97
DC pseudo-random + Det.	65612 + 96	94.68	95.23

Table 3. Improved numerical fault coverage results

	Pattern Count	Fault Coverage (%)	
		Collapsed	Uncollapsed
ITC'99 b04 (96 scan cells)			
AC ATPG unconstrained	56	82.99	84.57
AC ATPG constrained	52	79.85	81.38
AC pseudo-random	65535	82.22	83.56
AC pseudo-random + Det.	65535 + 0	82.22	83.56
DC ATPG unconstrained	57	93.28	94.34
DC ATPG constrained	62	90.62	91.92
DC pseudo-random	65535	92.22	93.53
DC pseudo-random + Det.	65535 + 1	92.36	93.64

APPENDIX B NUMERICAL AREA MEASUREMENT RESULTS

The appendix presents the area measurements in detailed numerical format. These results correspond to the area measurement plots of the IP cores and pattern generators given in Section 3.5.

Table 4. Deterministic pattern generation circuitry area measurements

IP Core	IP Core Scan Chain Length	IP Core Cell Area	AC Deterministic Pattern Generator			DC Deterministic Pattern Generator		
			Pattern Count	Cell Area	%	Pattern Count	Cell Area	%
ITC'99 b01	9	1288.8	0	0.0	0.0	0	0.0	0.0
ITC'99 b02	6	784.7	0	0.0	0.0	1	9261.4	1180.3
ITC'99 b03	38	4769.0	0	0.0	0.0	0	0.0	0.0
ITC'99 b04	85	13375.8	0	0.0	0.0	2	13949.1	104.3
ITC'99 b05	71	16880.4	6	16900.7	100.1	1	11993.5	71.1
ITC'99 b06	17	2004.3	0	0.0	0.0	0	0.0	0.0
ITC'99 b07	58	9070.4	16	20677.6	228.0	0	0.0	0.0
ITC'99 b08	34	4874.6	0	0.0	0.0	0	0.0	0.0
ITC'99 b09	30	4411.2	0	0.0	0.0	0	0.0	0.0
ITC'99 b10	34	4809.6	0	0.0	0.0	1	11188.5	232.6
ITC'99 b11	44	9041.9	3	13140.0	145.3	0	0.0	0.0
ITC'99 b12	132	24222.9	30	42119.7	173.9	5	18726.2	77.3
ITC'99 b13	73	9729.0	14	21742.8	223.5	0	0.0	0.0
ITC'99 b14	331	130314.9	58	132197.3	101.4	581	828859.4	636.0
ITC'99 b14_1	331	125180.0	32	90411.0	72.2	416	630437.8	503.6
ITC'99 b15	554	171954.5	511	1274621.8	741.3	251	691185.7	402.0
ITC'99 b15_1	554	162717.2	490	1231612.3	756.9	127	425490.6	261.5
HC11	181	93232.2	95	114682.5	123.0	26	49665.4	53.3
Post Processor	326	47559.4	91	189709.5	398.9	169	302875.8	636.8
Classifier	518	73721.7	126	374821.0	508.4	0	0.0	0.0
Pre Processor	642	94513.1	77	327297.9	346.3	96	394291.1	417.2

Table 5. Pseudo-random pattern generator and total test circuitry area measurements

IP Core	IP Core Cell Area	AC Pseudo-Random Pattern Generator		Total Test Circuitry	
		Cell Area	%	Cell Area	%
ITC'99 b01	1288.8	14128.0	1096.2	14128.0	1096.2
ITC'99 b02	784.7	14014.1	1786.0	23275.6	2966.3
ITC'99 b03	4769.0	15050.9	315.6	15050.9	315.6
ITC'99 b04	13375.8	15168.8	113.4	29117.9	217.7
ITC'99 b05	16880.4	15254.2	90.4	44148.4	261.5
ITC'99 b06	2004.3	14282.5	712.6	14282.5	712.6
ITC'99 b07	9070.4	15022.4	165.6	35700.1	393.6

ITC'99 b08	4874.6	15042.7	308.6	15042.7	308.6
ITC'99 b09	4411.2	14880.1	337.3	14880.1	337.3
ITC'99 b10	4809.6	15042.7	312.8	26231.3	545.4
ITC'99 b11	9041.9	15059.0	166.5	28199.0	311.9
ITC'99 b12	24222.9	15400.5	63.6	76246.3	314.8
ITC'99 b13	9729.0	15176.9	156.0	36919.7	379.5
ITC'99 b14	130314.9	15620.1	12.0	976676.8	749.5
ITC'99 b14_1	125180.0	15620.1	12.5	736468.8	588.3
ITC'99 b15	171954.5	15721.7	9.1	1981529.1	1152.4
ITC'99 b15_1	162717.2	15721.7	9.7	1672824.6	1028.1
HC11	93232.2	18827.7	20.2	183175.6	196.5
Post Processor	47559.4	15571.3	32.7	508156.5	1068.5
Classifier	73721.7	15725.8	21.3	390546.7	529.8
Pre Processor	94513.1	15644.5	16.6	737233.4	780.0

APPENDIX C IMPLEMENTATION OF SI^{2}P COMPONENTS

This appendix contains in-depth implementation details on how the P1500 wrapper would be integrated at the chip level when using the NIMA packet-switching test network as the TAM design [21][10]. NIMA is a packet-switched test network which is constructed in a tree configuration. All IP cores in the SoC are located at the leaves of the tree. The internal nodes represent routers which are responsible for forwarding test packets towards the leaves. In order for seamless communication, supportive logic blocks are inserted between each P1500 wrapper and NIMA test network. A P1500 wrapper connects to the test network, in sequence, through a wrapper controller, a NIMA interface, and a NIMA serializer. This appendix focuses on the design of these three components.

C.1 P1500 Wrapper Controller

To control the P1500 wrapper with ease, a controller which resembles that of the IEEE 1149.1 JTAG standard is adopted [22]. The wrapper controller is a Moore FSM with sixteen states that sequences the four P1500 control inputs, namely SELECTWIR, CAPTUREWR, SHIFTWR and UPDATEWR. Figure 52 shows interface of wrapper controller to a P1500 wrapper.

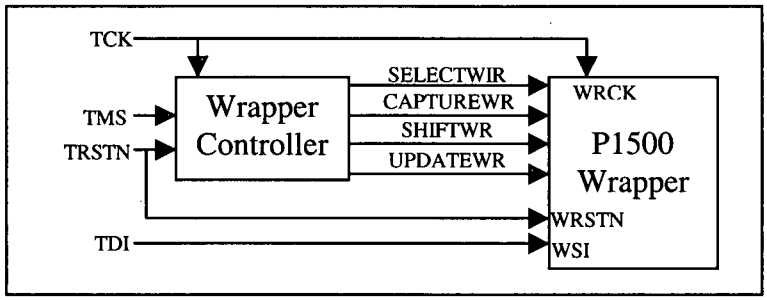


Figure 52. P1500 wrapper controller

The wrapper controller operates in accordance with the JTAG TAP controller. Its state changes are controller by a single test mode select (TMS) input triggered on the rising edge of the test clock (TCK). The state diagram of a P1500 wrapper controller is given in Figure 53. It is also the basis of the NIMA interface since it is designed to translate the packets from the NIMA test network to generate control signals understandable to the wrapper controller.

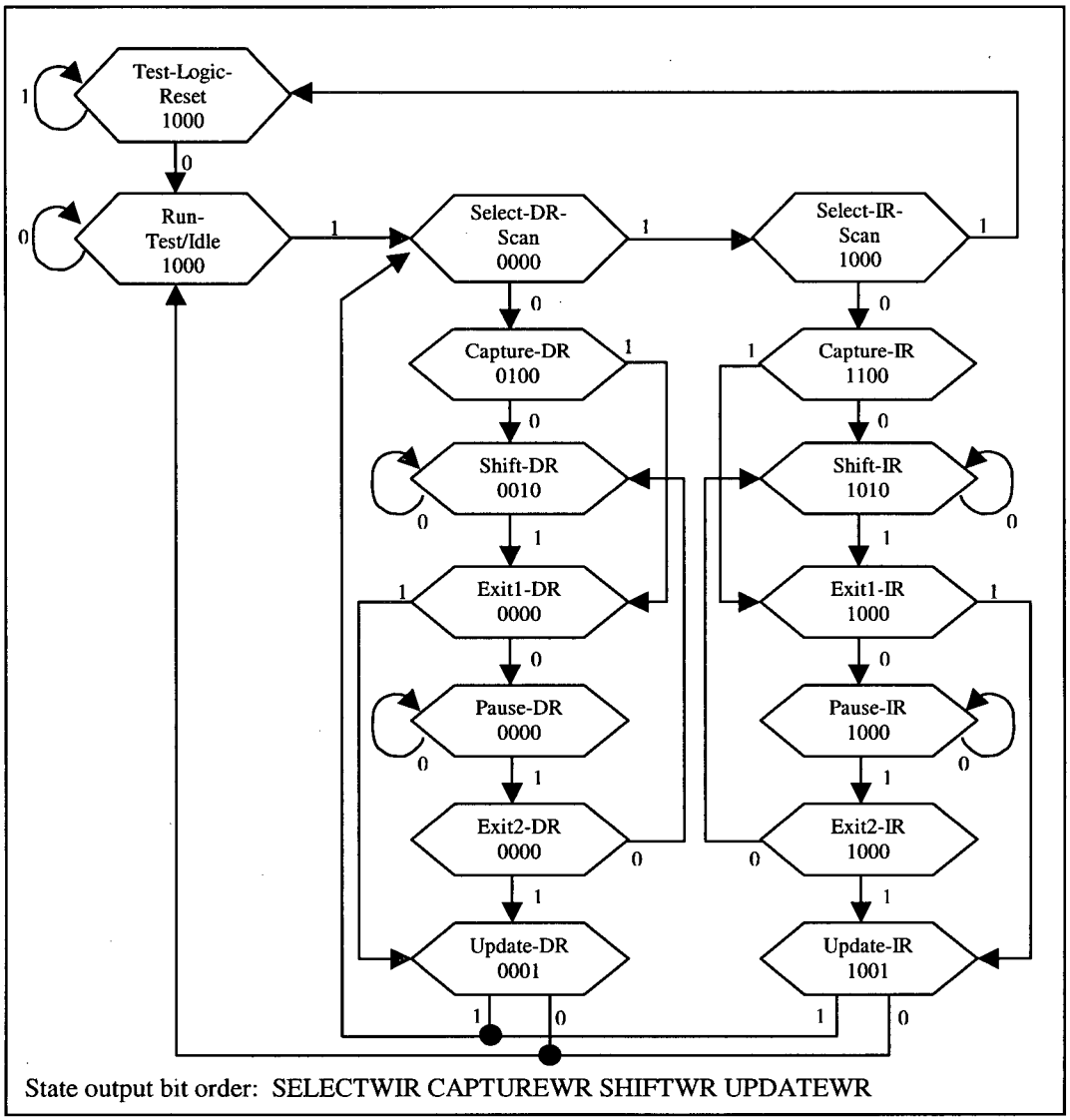
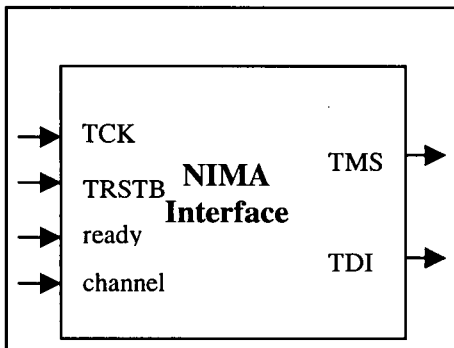


Figure 53. P1500 wrapper controller state diagram

C.2 NIMA Interface

The NIMA interface is implemented as a Mealy FSM that controls the P1500 wrapper controller by accepting and interpreting serial bit streams from the NIMA test network. Since NIMA is a packet-switched test network, incoming serial bit streams represent packets encoded with flag and payload. The responsibility of NIMA interface is to

identify the flag in each packet and sequence the downstream P1500 wrapper controller accordingly. Figure 54 shows a block view of the NIMA interface.



Pin	Type	Description
TCK	Input	Clock input to the block
TRSTB	Input	Active low synchronous reset
ready	Input	Flag indicating the validity of the channel pin
channel	Input	Serial bit data from the NIMA network interface
TMS	Output	Test mode select signal to P1500 wrapper controller
TDI	Output	Test data input to P1500 wrapper controller

Figure 54. NIMA interface block diagram

TCK and TRSTB inputs are the system clock and reset to NIMA interface respectively. Since this block and the P1500 wrapper controller operate in accordance, they share these two system inputs. The ready and channel inputs are the NIMA interface signals and are responsible for accepting packets of serial data from the NIMA test network. The NIMA test network provides the ready signal when a valid packet is available. It notifies the NIMA interface to accept the first bit of a packet from the channel pin. The ready signal is asserted shortly after a rising edge of the clock, which is at the same time as the bit at the channel input becomes valid. The ready signal is deasserted shortly after the rising clock edge of the last valid data bit at the channel input. Figure 55 shows a timing diagram of the interface.

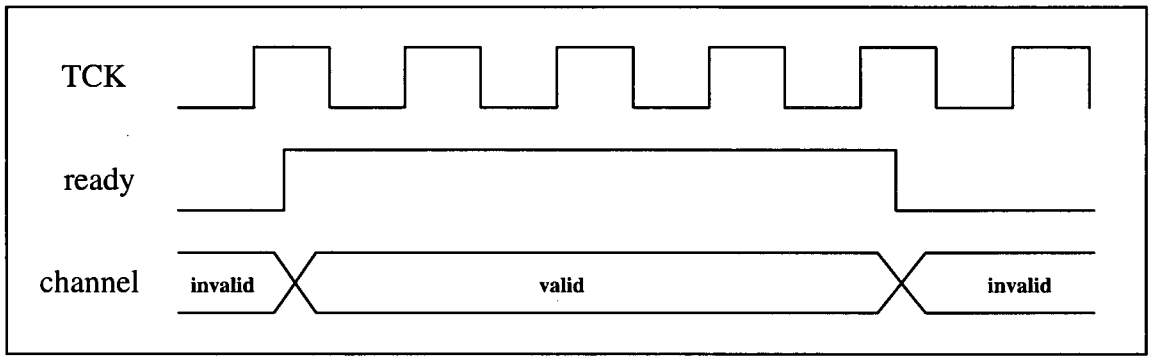


Figure 55. NIMA test network interface timing diagram

TMS and TDI inputs are responsible for controlling the P1500 wrapper controller and traversing it through the proper states. TMS pin is responsible for state changes in the P1500 wrapper controller state diagram in Figure 53. TDI is responsible for serially shifting out bits at the rising edge of the clock during the Shift-IR or Shift-DR states. TDI starts shifting at the first rising edge following the P1500 wrapper controller enters the shift state; and the last shift occurs as the P1500 wrapper controller leaves the shift and enters the exit1 state. Figure 56 shows the corresponding timing diagram.

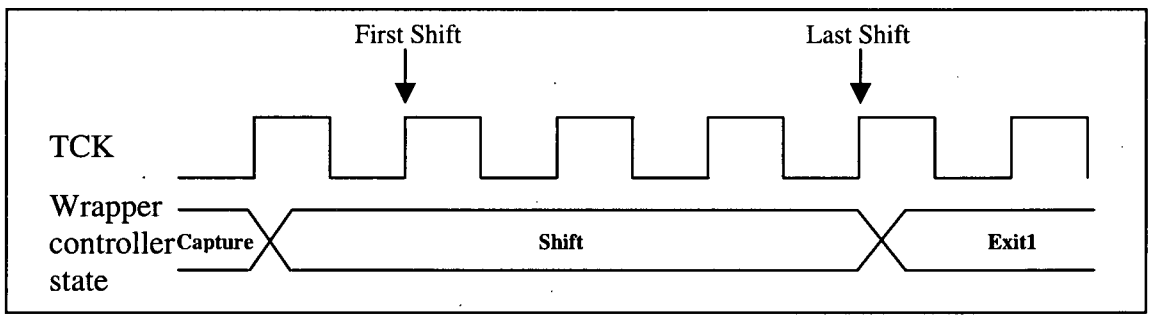


Figure 56. P1500 wrapper controller interface timing diagram

As described earlier, NIMA interface is implemented as a Mealy FSM and designed to manipulate the P1500 wrapper controller. They share a very similar state transition diagram with absence of several states, given the characteristics of a packet-switched

network such as NIMA. Figure 57 exhibits the state transition diagram of the NIMA interface.

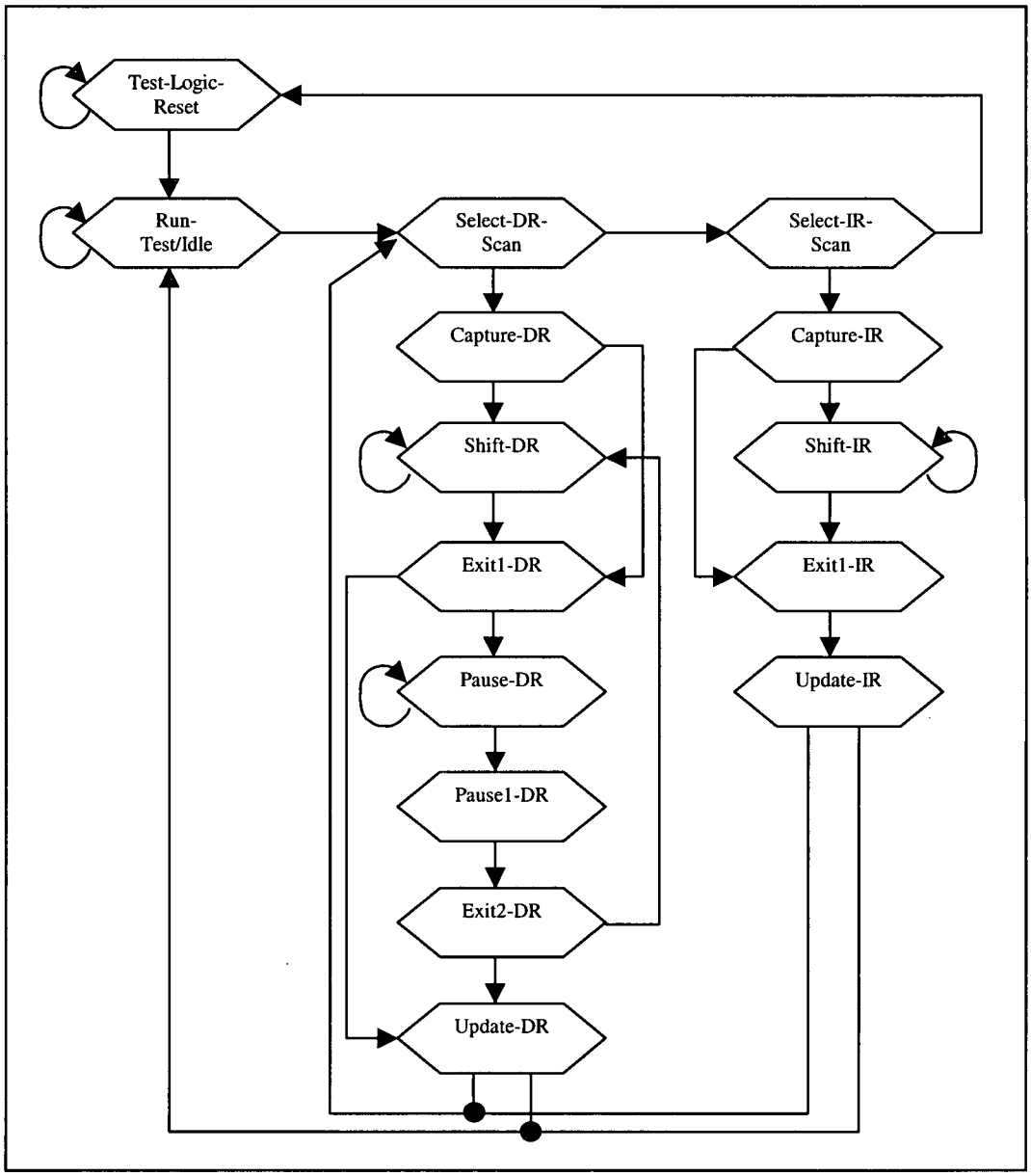


Figure 57. NIMA interface state diagram

Each packet arriving the NIMA interface is divided into two parts, flag and data. The flag specifies the type of information contained in the data portion of the packet; it also specifies the path to take in the P1500 wrapper controller state machine. There are two

main sets of flags. One set is mainly associated with instructing the machine to travel through the instruction branch of the P1500 wrapper controller; the other set is mainly associated with traveling solely along the data branch of the P1500 wrapper controller. All instruction flags have 3 bits with a value of 0 in the most significant bit (MSB) in the form of 0XX; on the other hand, all data flags only have 2 bits with a value of 1 in the MSB in the form of 1X. Table 6 lists the flags with corresponding definitions.

Table 6. NIMA interface packet flags

Flag	**Value**	**Description**
RESET [000]	000	Resets the NIMA interface by bringing the state machine to the Test-Logic-Reset state with a maximum of 5 clock cycles regardless of its currently state.
INSTR_ONLY [001 | instr]	001	Specifies the data portion of the packet contains instruction to be sent to the P1500 WIR. There should be just enough number of data bits to fill the P1500 WIR.
INSTR_COMP_DATA [010 | instr | data]	010	Specifies the data portion of the packet contains instruction to be sent to the P1500 WIR, as well as at least one bit to be sent to the P1500 wrapper data register. There should be enough number of bits to fill both the P1500 WIR and at least one bit to completely fill the P1500 data register.
INSTR_INCOMP_DATA [011 | instr | data]	011	Similar to INSTR_COMP_DATA flag. The data portion contains incomplete data that can be continued with subsequent packets having DATA_CONTINUE or DATA_END flags.
DATA_CONTINUE [10 | data]	10	Specifies the data portion only contains incomplete data to be sent to the data register of the P1500 wrapper. The incomplete data can be continued with subsequent packets having DATA_CONTINUE or DATA_END flags.
DATA_END [11 | data]	11	Specifies the data portion only contains data to be sent to the data register of the P1500 wrapper. Data packets with this flag are the

		last packets that complete the data transmission to the data register.

When the INSTR_ONLY flag is received, the state machine traverses along the instruction branch, shifts in the instruction bits in the Shift-IR state, then returns and stays idle in the Run-Test/Idle state. At the Run-Test/Idle state, it waits for the next packet to come in from the NIMA test network.

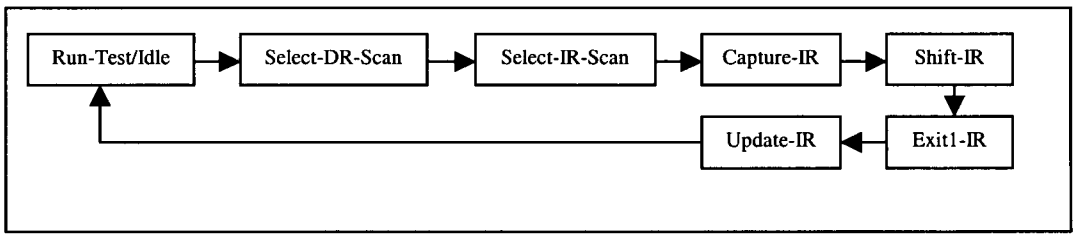


Figure 58. States traversed by INSTR_ONLY flag

When the INSTR_COMP_DATA flag is received, the state machine behaves similarly to the INSTR_ONLY flag except that it does not return to the Run-Test/Idle state immediately after the instruction bits have been shifted in. Instead, after shifting of the instruction bits, it moves to the Select-DR-Scan state and progresses to the Shift-DR state to shift in the data bits before moving back to the Run-Test/Idle state.

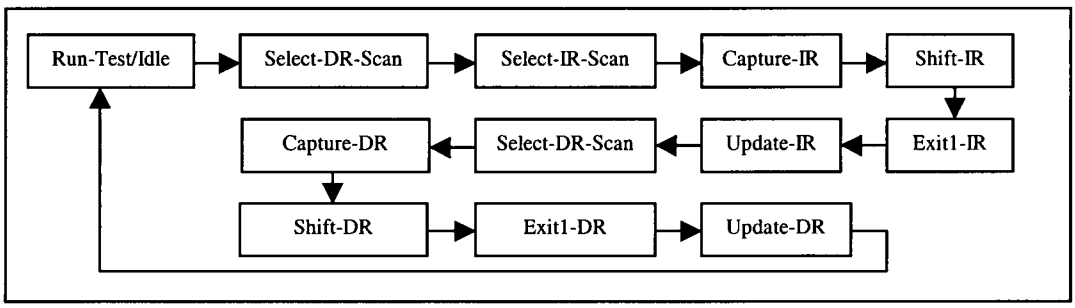


Figure 59. States traversed by INSTR_COMP_DATA flag

When the INSTR_INCOMP_DATA flag is received, the state machine behaves similarly to receiving the INSTR_COMP_DATA flag except that it does not return to the Run-Test/Idle state after the data have been shifted in. Instead, after shifting of the data bits, it moves to the Pause-DR state where it waits for the next data packet to come in.

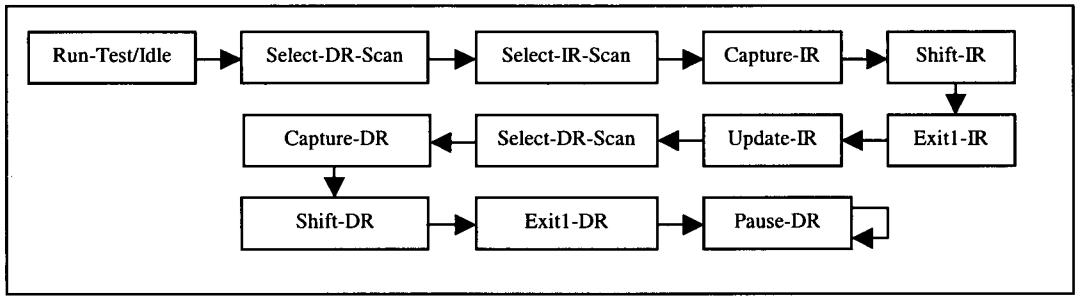


Figure 60. States traversed by INSTR_INCOMP_DATA flag

The first bit of the DATA_CONTINUE flag can start arriving at the NIMA interface when the state machine is in either Run-Test/Idle or Pause-DR state. This flag specifies that the data portion of the packet contains data bits that are to be shifted in during the Shift-DR state. After the data bits have been shifted in, the state machine moves to the Pause-DR state and waits for the next data packet to arrive.

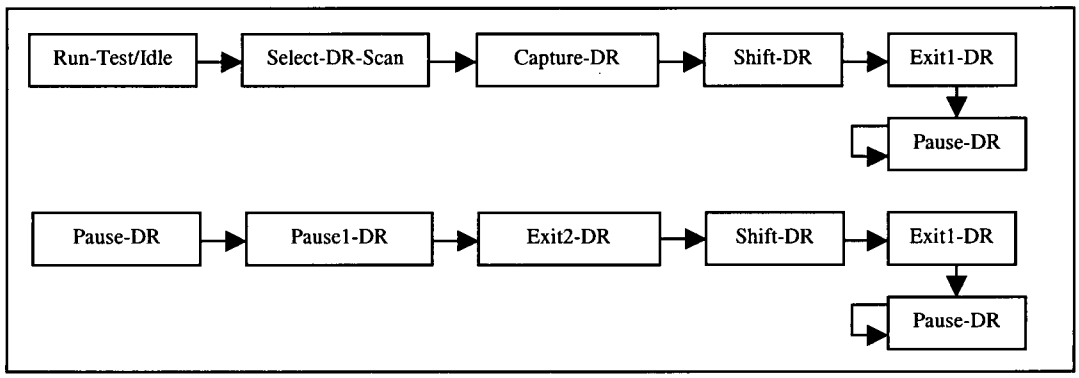


Figure 61. States traversed by DATA_CONTINUE flag

The DATA_END flag is basically identical to the DATA_CONTINUE flag except that after the data bits have been shifted in, the state machine moves to the Run-Test/Idle

instead of moving to the Pause-DR state. It is at the Run-Test/Idle state that the NIMA interface waits for the next packet to arrive.

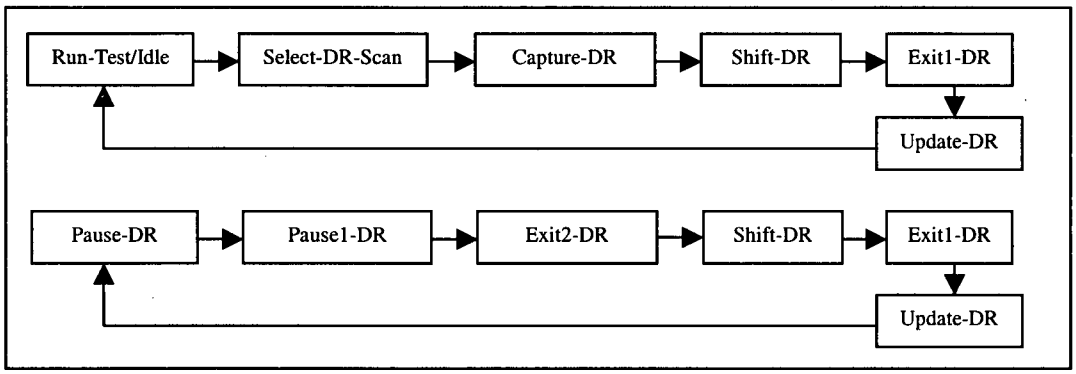


Figure 62. States traversed by DATA_END flag

There are circumstances where a soft reset is triggered. This is implemented to ensure the NIMA interface, as well as the P1500 wrapper controller, gracefully handles packets that do not comply with the protocol. When a soft reset is triggered, the state machine traverses through the state diagram and returns to the reset state. There are several ways to engage the soft reset mode. One way to engage a soft reset to the NIMA interface is to issue a packet with a RESET flag. When the NIMA interface sees this flag, it engages a soft reset and proceeds as described earlier. A soft reset is also engaged if the INSTR_ONLY packets do not contain enough data bits to fill the P1500 WIR. For example, if the P1500 WIR is four bits wide, the data portion of the packet must be at least 4 bits long; otherwise, a soft reset is engaged.

A soft reset is initiated with inappropriately formatted INSTR_COMP_DATA or INSTR_INCOMP_DATA packets. There must be not only enough bits to fill the P1500 WIR, but also at least one bit to fill the P1500 wrapper data register. For example, given a four-bit-wide P1500 WIR, the data portion of the packet must be at least 5 bits long.

Packets received at inappropriate states can also trigger a soft reset. The start of data packets, DATA_CONTINUE and DATA_END, can only be received when the NIMA interface is in either Run-Test/Idle or Pause-DR state. If instruction packets are received when the NIMA interface is in the Pause-DR state, a soft reset is triggered because during that state, the state machine expects bits to be shifted into the TAP controller data register only.

Since the bits from the NIMA test network arrive continuously and serially, these incoming bits must be buffered in order to properly pass the incoming bits to the output TDI pin. To accomplish this task, a multi-buffer selection system is implemented. Figure 63 shows a conceptual view of the buffer pool.

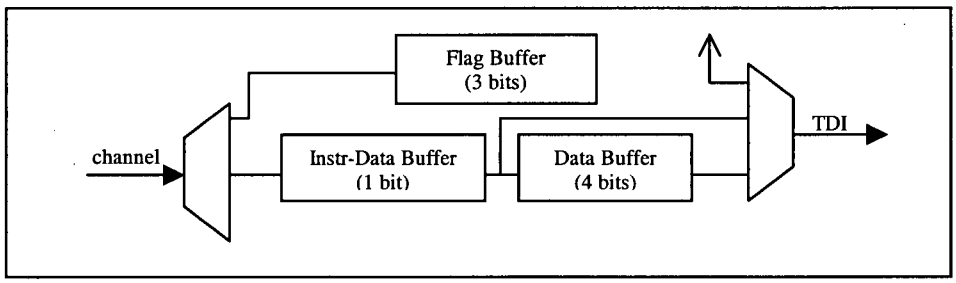


Figure 63. Buffering in NIMA interface

The flag buffer is responsible for storing the flag portion of the packets. Once the appropriate number of bits, 2 or 3 depending on the flag type, are shifted into the flag buffer, the flag buffer is disabled to retain the flag values for later reference.

The instruction-data buffer provides a one-cycle delay for bits to be shifted out during the Shift-IR and Shift-DR states. This is necessary due to the fact that the last shift operation

occurs as the state machine changes from Shift to Exit state. If the channel were connected directly to TDI, the P1500 wrapper controller would shift in one extra bit as it enters the Exit state.

The data buffer is simply a concatenation of the one-bit instruction-data buffer and 4 more shift registers. These 5 shift registers provide sufficient delay for INSTR_COMP_DATA and INSTR_INCOMP_DATA as the state machine works its way from the Shift-IR state to the Shift-DR state. This is necessary because these two types of packets have the P1500 WIR instruction bits immediately followed by the P1500 wrapper data bits in the data part of the packets.

During the design of the NIMA interface, efforts have been made to maximize the robustness of the block. However, to ensure the proper state transitions, it is required to have a gap of at least 8 clock cycles between payloads of the packets. Each payload contains a flag portion and a data portion understandable to the NIMA interface. Each header holds the routing information on which the corresponding payload relies to navigate the NIMA network. This should be easily complied because the number of bits required in the header of each packet usually exceeds the stated requirement. Thus the bit stream appears as depicted in Figure 64.

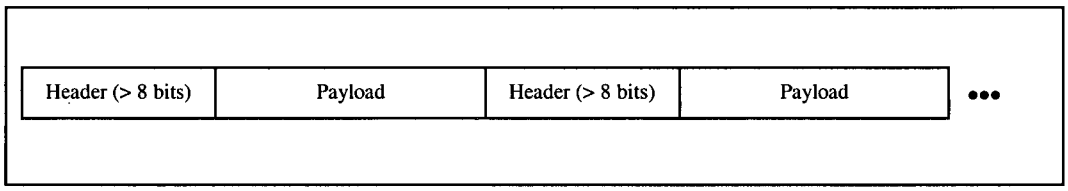


Figure 64. NIMA packet spacing requirement

C.3 NIMA Serializer

Since NIMA test network is a packet-switched network, test packets must be able to arrive the designated cores at a higher rate in order to spare the network for other cores. If this were not achieved, the NIMA test network would become a circuit-switched network rather than a packet switching network.

There are several ways to accomplish the goal of having the NIMA network operating faster than the cores. The most obvious method is to have the NIMA network running at a higher clock speed than that of the cores. However different clock domains in the test network would be introduced as a result. This is undesirable because interfacing across different clock domains is quite error prone and requires extra design precautions. The second method is to widen the NIMA test network bus and introduce buffering in each NIMA router such that they all behave like the ones used for the Internet connections.

The third method is to again widen the NIMA test network bus and introduce First-In-First-Out (FIFO) control at each core whose incoming traffic rate is larger than the outgoing traffic rate. In order to achieve this method, a NIMA serializer is required to temporarily store the incoming test packets and feed the cores with those packets at a slower rate by having the incoming bus width (M) strictly larger than the outgoing bus width (N). In other words, $M > N$. When $M = N$, then there is no need for the existence of the NIMA serializer.

The NIMA serializer is an essential component for enabling the packet-switching capability of the NIMA test network. It is simply a FIFO controller with the input data bus width larger than that of the output data bus. In order to implement the NIMA serializer, dual port memory, either implemented as banks of flip-flops or RAM, is required. At the read side of the dual port memory, a shift register is employed to control the number of parallel bits available to the serializer output. Figure 65 depicts the NIMA serializer structural overview.

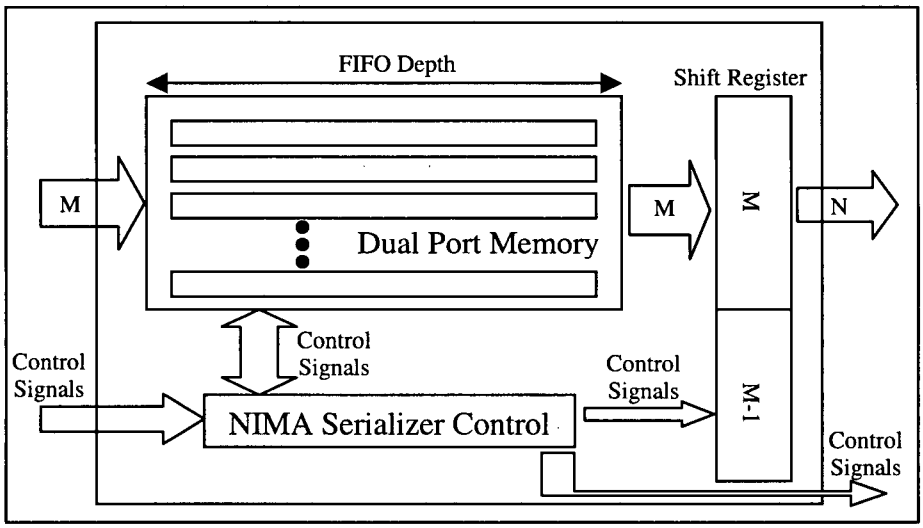


Figure 65. NIMA serializer structural overview