Synthesis for Scan Dependence in Built-In Self-Testable Designs

LaNae J. Avra and Edward J. McCluskey

94-2 (CSL TR 94-621) May 1994	**Center for Reliable Computing** ERL 460 Computer Systems Laboratory Departments of Electrical Engineering and Computer Science Stanford University Stanford, California 94305-4055

Abstract:
This report introduces new design and synthesis techniques that reduce the area and improve the performance of embedded built-in self-test (BIST) architectures such as circular BIST and parallel BIST. Our goal is to arrange the system bistables into scan paths so that some of the BIST and scan logic is shared with the system logic. Logic sharing is possible when *scan dependence* is introduced in the design. Other BIST design techniques attempt to avoid all types of scan dependence because it can reduce the fault coverage of embedded, multiple input signature registers (MISRs). We show that introducing certain types of scan dependence in embedded MISRs can result in reduced overhead and improved fault coverage, and we describe synthesis techniques that maximize the amount of this *beneficial* scan dependence. Finally, we present fault simulation, layout area, and delay results for circular BIST versions of benchmark circuits that have been synthesized with our techniques.

Funding:
This work was supported in part by the Innovative Science and Technology Office of the Strategic Defense Initiative Organization and administered through the Office of Naval Research under Contract No. N00014-92-J-1782, and by the National Science Foundation under Grant No. MIP-9107760.

Imprimatur: Siyad Ma and Nur Touba

Synthesis for Scan Dependence in Built-In Self-Testable Designs

LaNae J. Avra and Edward J. McCluskey
CRC Technical Report No. 94-2
(CSL TR 94-621)
May 1994

CENTER FOR RELIABLE COMPUTING
Computer Systems Laboratory
Departments of Electrical Engineering and Computer Science
Stanford University
Stanford, CA, USA 94305-4055

ABSTRACT

This report introduces new design and synthesis techniques that reduce the area and improve the performance of embedded built-in self-test (BIST) architectures such as circular BIST and parallel BIST. Our goal is to arrange the system bistables into scan paths so that some of the BIST and scan logic is shared with the system logic. Logic sharing is possible when *scan dependence* is introduced in the design. Other BIST design techniques attempt to avoid all types of scan dependence because it can reduce the fault coverage of embedded, multiple input signature registers (MISRs). We show that introducing certain types of scan dependence in embedded MISRs can result in reduced overhead and improved fault coverage, and we describe synthesis techniques that maximize the amount of this *beneficial* scan dependence. Finally, we present fault simulation, layout area, and delay results for circular BIST versions of benchmark circuits that have been synthesized with our techniques.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ILLUSTRATIONS

1 INTRODUCTION

Hardware synthesis techniques automatically generate a structural hardware implementation given an abstract description of the behavior of the design. Many different hardware designs can implement a given behavioral description, a subset of which also meet specified requirements such as area, performance, and testability. Current research in hardware synthesis techniques (e.g., [Brayton 87], [McFarland 88], [De Micheli 94]) typically focuses on the use of minimum area or maximum performance as the primary criteria for selecting the best hardware implementation. We have developed and implemented new *synthesis-for-BIST* techniques whose primary objective is to satisfy requirements associated with a specific built-in self-test (BIST) architecture when generating the system logic structural implementation.

BIST techniques (design techniques that allow a circuit to test itself) such as those described in [McCluskey 85], [McCluskey 86], [Bardell 87], and [Abramovici 90] have long been recognized as a means to reduce product life cycle test and maintenance costs by embedding external tester features, such as *test pattern generation (TPG)* and *output response analysis (ORA)*, into the module that contains the *circuit under test (CUT)*. BIST can provide shorter test times than functional or deterministic testing techniques and allows the use of low-cost test equipment during all stages of the product life: system debug, production test, field maintenance, and failure diagnosis. Even though industry use of BIST techniques is becoming more common [Gelsinger 86], [Lake 86], [Nozuyama 88], [Ratiu 90], [Starke 90], [Bardell 91], [Illman 91], [Preissner 92], [Sinaki 92], [Yokomizo 92], [Bonnenberg 93], [Broseghini 93], [Gage 93], [Langford 93], [Patel 93], area overhead, performance degradation, and increased design time are often cited as reasons for the limited use of BIST. *BIST synthesis* techniques, where BIST circuitry is added or a BIST architecture is selected after the system hardware has been implemented [Abadir 85], [Zhu 88], are valuable in that they reduce the design time and the number of design mistakes by automating the BIST circuitry implementation process. Unfortunately, earlier system design decisions can reduce the effectiveness, increase the area, and reduce the performance of the implemented BIST technique.

We have developed new synthesis techniques that address two of the major issues associated with at-speed, embedded BIST architectures such as parallel BIST and circular BIST: 1) implementation costs due to increased design time, area overhead, and performance degradation, and 2) test effectiveness due to system logic design decisions. Our synthesis procedures use criteria associated with the specified BIST architecture to guide the generation of the system logic, allowing for design decisions that reduce BIST overhead and improve BIST effectiveness. In particular, the goal of our synthesis techniques is to introduce *scan dependence* in the design so that some of the BIST and scan logic can be shared with the system logic, reducing the area and improving the performance of the design. Other BIST design techniques attempt to avoid all types of scan dependence because it can reduce the fault coverage of embedded ORAs called *multiple input signature registers (MISRs)*. We show that introducing certain types of scan dependence in embedded MISRs can result in reduced overhead and improved fault coverage.

This report is organized as follows. In Sec. 2, we first give an overview of two popular embedded BIST architectures: parallel BIST and circular BIST. We then describe the implementation and operation of a register design that is often used in embedded BIST architectures. We use this register design to illustrate the concepts and design techniques presented in the remainder of the report. In Sec. 3, we define scan dependence and illustrate its advantages and disadvantages with a simple example. In Sec. 4, we describe design techniques for beneficial scan dependence and show how to introduce it in the design by arranging the system bistables into MISRs. Section 5 provides fault simulation and overhead results of several benchmark circuits generated with our synthesis techniques and optimized for the circular BIST architecture described in Sec. 2. In Sec. 6, we show that beneficial scan dependence can be maximized in data path logic when an *orthogonal scan path* configuration [Avra 92] is used. Section 7 concludes this report.

2 EMBEDDED BIST ARCHITECTURES

BIST techniques are implemented by including a *test pattern generator (TPG)* and *output response analyzer (ORA)* in the part (e.g., chip, multi-chip module, board) that contains the *circuit under test (CUT)*. During BIST operation, the TPG applies test patterns to the inputs of the CUT and the ORA captures the response of the CUT to those test patterns. Typically, the TPG and ORA are implemented such that they automatically generate patterns and analyze responses at test time so that the test pattern and test response storage requirements are minimized. *Embedded* BIST architectures use reconfigurable system bistables to implement both the normal system operation and the BIST TPG and ORA operations. Since the BIST logic is combined with the system logic, opportunities exist for the synthesis technique to generate hardware that can be shared by both the system and test operations, resulting in improved performance and reduced cost. However, in addition to system logic issues such as performance and cost, the synthesis technique must be capable of addressing issues that affect the quality of the BIST operation, issues such as self-test time, and TPG and ORA effectiveness. We discuss these issues in this section.

Most BIST architectures use pseudo-random pattern generators such as *linear feedback shift registers (LFSRs)* or weighted random pattern generators [Waicukauski 89] to perform TPG operations. The TPGs are called *pseudo-random* because, while they generate patterns whose characteristics are similar to randomly-generated patterns, their behavior is deterministic. The test is therefore repeatable, and the results of the test can be compared with expected results. In order to test for all possible combinational faults (e.g., single and multiple stuck-at faults) within an *n*-

input cone of combinational logic in the CUT, where a cone of logic is delimited by starting at a combinational logic output (either a primary output or a bistable input) and tracing backwards through the gates to each input (either a primary input or a bistable output), a TPG must be capable of applying all 2^n possible test patterns. To test for pattern-dependent faults [Hao 91], delay faults, or faults that cause combinational logic to behave sequentially, the TPG may be required to generate multiple sequences of the 2^n possible patterns. Faults that affect more than one cone of logic (e.g., a bridging fault between the inputs of two logic gates that are in different cones) can be detected if a single TPG covers multiple cones. We evaluate the TPGs generated by our synthesis techniques in terms of their ability to generate 2^n different test patterns for each n-input cone of logic. Even though it may be determined that not all of the 2^n test patterns need to be applied during BIST operation, it is difficult to determine *a priori* which subset of test patterns will be required to provide a high fault coverage test for each CUT. Therefore, the TPGs must be designed such that they could generate all 2^n different test patterns if necessary. This is known as the "philosophy of possible exhaustion" [Bardell 87]. Also, if all 2^n test patterns can be generated, one can use probabilistic models, based on the number of test patterns applied and the detectabilities of the faults in the CUT, to estimate the fault coverage provided by the TPG [McCluskey 88], thus avoiding expensive and time-consuming fault simulations.

An ideal ORA is capable of analyzing the response of each test pattern applied to the CUT. In practice, of course, this is not feasible for a built-in ORA. Instead, most BIST architectures use ORAs, such as *multiple-input signature registers (MISRs)*, that compact the test response, resulting in a loss of information. Thus, under certain error conditions, the test response of a faulty CUT may be indistinguishable from the test response of a fault-free CUT, a phenomenon known as *aliasing*. Our synthesis techniques assume that MISRs are used as the ORAs in the implemented BIST architecture and attempt to minimize the amount of aliasing in those MISRs.

In this section, we describe two popular embedded BIST architectures, parallel BIST and circular BIST, and discuss BIST-related synthesis issues for each. In Sec. 2.1, we describe parallel BIST, emphasizing often-overlooked implementation complexities that must be addressed by the synthesis technique so that it generates a design that performs an effective BIST operation. Although our synthesis techniques are applicable to all BIST architectures that use embedded MISRs, we have implemented them in a design tool that produces circular BIST architectures. We describe the assumed circular BIST architecture in Sec. 2.2. In Sec. 2.3, we describe one possible implementation of a reconfigurable register design for embedded BIST architectures. The register is configured as a normal parallel load register for system operation. During BIST operation, it is configured as either a maximum-length LFSR for test pattern generation or as a MISR for output response analysis.

2.1 PARALLEL BIST ARCHITECTURE

The parallel BIST architecture [Konemann 80] is an embedded BIST architecture in which system registers are reconfigured to perform either LFSR or MISR operations during BIST. The *built-in logic block observer (BILBO)* register [Konemann 79] is often used to implement the system registers in parallel BIST architectures. A possible implementation of the BILBO register is described in detail in Sec. 2.3. Variations of the parallel BIST architecture are discussed in [McCluskey 81], [Krasniewski 85], [Hudson 87].

Figure 2-1 illustrates how the parallel BIST architecture is used to test a block of data path logic that consists of five registers (*R1-R5*), five combinational logic units (*CLU1-CLU5*), primary input signals (*PI*), and primary output signals (*PO*). During normal operation (Fig. 2-1a), the system registers are configured to load data from the outputs of the CLUs. Prior to performing BIST operation, the system bistables must begin in a known state. This can be accomplished by configuring the registers into a scan path and shifting in known data or by configuring the registers to perform a reset operation. During BIST operation, the registers are configured as either LFSRs or MISRs. At each clock cycle, pseudo-random test patterns are applied to the inputs of the CLUs by the LFSRs at the same time that the test results (the outputs of the CLUs) are compacted in the MISRs. Figures 2-1b and 2-1c represent two different *test sessions* of the BIST operation, where each test session consists of a unique mapping of registers to LFSRs and MISRs. *CLU3*, *CLU4*, and *CLU5* are tested during the first test session (Fig. 2-1b), and *CLU1* and *CLU2* are tested during the second test session (Fig. 2-1c). After each test session, the registers are configured into a serial shift path so that the test results can be shifted out and compared with expected results. This comparison can be performed either by external test equipment or by BIST control logic that may be included as part of a test access port controller [IEEE 90]. The BIST control logic also generates signals that control the configuration of the system registers.

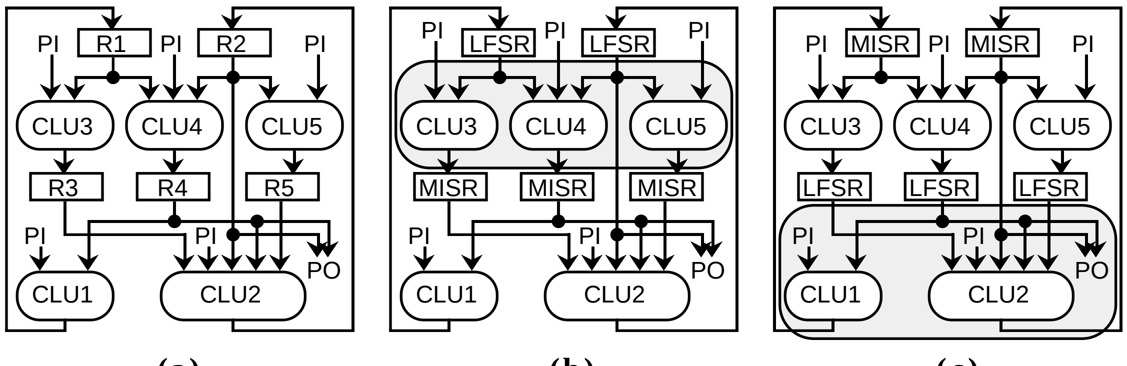


Figure 2-1 Parallel BIST architecture: (a) normal operation; (b) first test session configuration; (c) second test session configuration.

One advantage of the parallel BIST architecture is that it supports *at-speed self-test operation*: a new test pattern is applied to the CUT at each clock cycle. At-speed operation means that the parallel BIST technique may be able to detect some delay faults. Another advantage of this technique is that, since each system register can be reconfigured as either an LFSR or a MISR, multiple CLUs can be tested in parallel, reducing the total self-test time.

We define an ideal parallel BIST architecture as one for which the system bistables can be arranged into LFSRs and MISRs such that, for each *n*-input CLU in the design, 2^n different test patterns can be applied to the CLU inputs during the same test session that the CLU outputs are captured in a MISR. Unfortunately, due to the inter-connectedness of the system bistables in typical designs, many test sessions may be required to test all of the CLUs in an ideal parallel BIST architecture. For example, register interconnection data and test session configurations for a twenty-three register, parallel BIST design is provided in [Hudson 87]. This information is shown in a different form in Table 2-1, which lists which registers, numbered 1 through 23, are configured as LFSRs and MISRs for each test session. We assume that any CLU feeding a register that is configured as a MISR during a given test session is tested during that test session. For example, during test session 10, the CLUs that feed registers 9 and 16 are tested by test patterns supplied by registers 10, 12, and 17. The design requires a minimum of twelve different test sessions, where an average of 1.75 registers are configured as MISRs and 13.9 registers are configured as LFSRs in each test session. The high number of test sessions corresponds to the low degree of self-test parallelism in the design.

Table 2-1 Test session register configurations for design example in [Hudson 87].

Test Session	MISRs	LFSRs
1	3,13*,17	1,2,4,5,6,7,8,9,10,11,12,14,15,16,19,20,21,22,23
2	19,20*	1,2,4,5,6,7,8,9,10,11,13,14,15,16,21,22,23
3	23*	1,2,4,5,6,7,8,9,10,11,12,13,14,15,16,19,20,21,22
4	6*,10*	1,2,4,5,7,8,9,11,13,14,15,16,19,20,21,22,23
5	5*,11*	1,2,4,6,7,8,9,10,13,14,15,16,20,21,22,23
6	4,12*	1,2,3,7,8,9,10,11,13,14,15,16,17,19,20,21,22,23
7	18	1,2,4,6,7,8,9,10,11,12,13,14,15,16,19,20,21,22,23
8	14*	1,2,4,7,8,9,10,11,13,15,16,20,21,22,23
9	7*,8*	1,2,9,10,13,14,15,16,20,21,22,23
10	9,16	10,12,17
11	1*	13,20,21,23
12	2*,15*	1,13,14,16,20,21,22,23

*Self-adjacent register configured as a MISR during test.

One of the barriers to implementing an economical parallel BIST architecture is self-adjacent registers. A *self-adjacent register*, marked with an asterisk in Table 2-1, is one in which at least one output of the register feeds through either a direct connection or combinational logic to at least one input of the same register (see register *R2* in Fig. 2-1a). If the self-adjacent register is configured as an LFSR in order to supply test patterns to the CLU during BIST operation, the response of the CLU cannot be observed. A self-adjacent register that is implemented with a *concurrent built-in logic block observer (CBILBO)* design as described in [Wang 86] is able to simultaneously perform both the LFSR and MISR operations because it has two sets of bistables. CBILBO register operation for a self-adjacent register is illustrated in Fig. 2-2. During normal operation, only one of the sets of CBILBO bistables is used (Fig. 2-2a). During BIST operation, the set of bistables that drives the system logic is configured into an LFSR, and the other set of bistables is configured into a MISR. Unfortunately, the prevalence of self-adjacent registers combined with the higher hardware overhead for CBILBOs (CBILBO registers are approximately 1.75 times the size of BILBO registers) greatly increases the BIST overhead for the design in Table 2-1. When CBILBOs are not used, a self-adjacent register can be configured as a MISR during BIST operation. The MISR then provides test patterns to the CLU at the same time that it captures the response of the CLU (e.g., register *R2* configured as a MISR in Fig. 2-1c). Since the state of the self-adjacent MISR depends upon its previous state and the CLU, there is no guarantee that it can generate an exhaustive set of test patterns for the CLU. Also, the CLU can adversely affect the output response capabilities of the self-adjacent MISR, as was noted by Hudson [Hudson 87] for the case when a shift operation is implemented in the CLU . We discuss the effect of the CLU on the MISR output response capabilities in detail in Sec. 3.

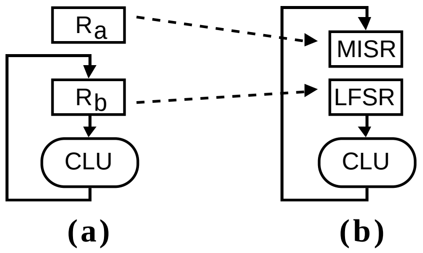


Figure 2-2 Self-adjacent register implemented with CBILBO: (a) normal operation; (b) BIST operation.

In an effort to solve this problem, high-level synthesis techniques have been proposed that generate data path logic with a minimum number of self-adjacent registers [Avra 91], [Papachristou 91], [Mujumdar 92]. These synthesis techniques are among the first to use BIST criteria to guide the high-level synthesis of the system logic. High-level synthesis techniques that only consider area and performance during synthesis tend to generate system logic with a large number of self-adjacent registers. For example, five of the eight registers in the *Tseng* example generated by the synthesis technique described in [Tseng 86] and four of the five registers in the *DiffEq* example generated by the synthesis technique described in [Paulin 89] are self-adjacent. The synthesis technique described in [Avra 91] generated system logic with only one self-adjacent register for both of these examples. For both examples, when CBILBO registers are used for the self-adjacent registers, the synthesis-for-BIST technique generated lower-cost parallel BIST implementations than the synthesis techniques that do not consider BIST.

Another implementation difficulty of the ideal parallel BIST architecture is determining the arrangement of system bistables into LFSRs during each test session so that each CLU tested during that test session could receive an exhaustive set of test patterns. The simplest arrangement is to configure each n-bit system register as a maximum-length LFSR, which generates 2^n-1 different test patterns (logic can easily be added to the LFSR so that it generates all 2^n different patterns [McCluskey 86]). This is not an acceptable solution, however, when a single CLU is fed by multiple system registers (see *CLU2* in Fig. 2-1). In this case, the total number of different test patterns, P, that can be applied to the CLU is the least common multiple of the periods of the individual LFSRs. For n maximum-length LFSRs, where LFSR i has b_i bistables and $b_1 \leq b_2 \leq \ldots \leq b_n$:

$$(2^{b_n} - 1) \quad \leq \quad P \quad \leq \quad (2^{b_1} - 1)\,(2^{b_2} - 1)\,(2^{b_3} - 1) \ldots (2^{b_{n-1}} - 1)$$
$$P \quad < \quad 2^{b_1+b_2+b_3+\ldots+b_{n-1}}$$

The maximum value for P can only be achieved when the periods of the n LFSRs are mutually prime. In data path logic, where a single CLU is often fed by n b-bit system registers, we have the worst-case scenario:

$$P \quad = \quad 2^b - 1 \quad << \quad 2^{nb}$$

A better solution for providing a nearly exhaustive set of test patterns to each CLU is to configure the bistables on the inputs of each CLU being tested during a given test session into a single, maximum-length LFSR. Figure 2-3 illustrates this solution by showing one way to configure the registers of Fig. 2-1 into LFSRs and MISRs for each of the two required test sessions. During the first test session (Fig. 2-1b and 2-3a), registers *R1* and *R2* are configured into a single maximum-length LFSR in order to provide an exhaustive set of test patterns to *CLU4*. Registers *R3*, *R4*, and *R5* are configured into a MISR to capture the output responses of *CLU3*, *CLU4*, and *CLU5*. During the second test session (Fig. 2-1c and 2-3b), *R2*, which is implemented with a CBILBO register, is configured with *R3*, *R4*, and *R5* into a single LFSR in order to test *CLU2*. The second set of *R2* bistables are configured with *R1* into a MISR, which captures the output responses of *CLU1* and *CLU2*. The signal lines in Fig. 2-3 represent the additional interconnections required for BIST operation. Each block of feedback logic (*f1*, *f2*, and *f3*) consists of a tree of one or more exclusive-OR gates that implements the polynomial of the associated LFSR or MISR. The arrangement of the registers during BIST operation ($R1 \rightarrow R2 \rightarrow R3 \rightarrow R4 \rightarrow R5$) was chosen because it minimizes the amount of BIST interconnection and feedback logic.

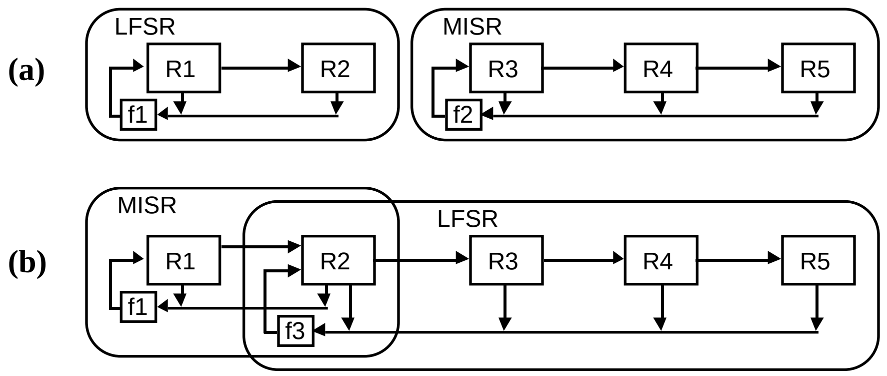


Figure 2-3 LFSR and MISR configurations for Fig. 2-1: (a) first test session; (b) second test session.

Unfortunately, the parallel BIST architecture illustrated in Fig. 2-3 can, in practice, be extremely complicated and costly to implement. In our discussion, we have ignored how to configure LFSRs to drive the input signals (primary input signals and control signals) of the data path logic during BIST operation. Considering these signals could greatly complicate the configurations of the LFSRs during each test session, particularly if a single state machine generates all of the control signals for the data path logic. Also, Fig. 2-1 is a relatively simple data path design example. More complicated data path designs, or non-BIST scan path arrangement requirements, such as minimizing test time for deterministic stuck-at tests as described in [Gupta 91] and [Narayanan 92], could greatly increase both the hardware overhead and the total self-test time for the parallel BIST architecture. Table 2-1 provides a hint of how complicated the parallel BIST architecture could be for a real design. For example, test session seven in Table 2-1 shows that the combinational logic feeding register 18 is fed by the outputs of 19 different registers. When these 19 registers are configured into a single, maximum-length LFSR, 2^b-1 different test patterns are applied to the CLU, where *b* is the total number of bistables in the 19 registers. Subsets of these 19 registers must be combined with other registers to create the LFSRs for the remaining eleven test sessions. As the number of test sessions increases, the BIST control logic and the LFSR interconnection logic for the design can become very complicated.

The synthesis techniques described in [Avra 91], [Papachristou 91], and [Mujumdar 92] address one of the implementation difficulties of parallel BIST architectures by generating system logic that has a minimum number of self-adjacent registers. However, the data in Table 2-1 shows that complicated register interconnections in the system logic can greatly complicate the LFSR configurations required for a parallel BIST architecture. This issue must be addressed by synthesis techniques before extensive use of the parallel BIST architecture is practical. To that end, we have investigated more simple TPG and ORA configuration schemes, such as those found in the circular BIST architecture. The circular BIST architecture is described in Sec. 2.2 and new synthesis techniques for this architecture are described in Secs. 4 and 6.

2.2 CIRCULAR BIST ARCHITECTURE

Circular BIST is a low overhead, embedded BIST architecture that provides at-speed self-test operation. Circular BIST has lower area overhead than parallel BIST because it has simpler BIST control logic and interconnection logic, which also simplifies and speeds the implementation process. The circular BIST architecture was first introduced as *simultaneous self-test (SST)* by Bardell and McAnney [Bardell 82]. It was later described in slightly different forms by Stroud [Stroud 88] and Krasniewski [Krasniewski 89]. Figure 2-4 illustrates the circular BIST architecture for the data path logic in Fig. 2-1a. During BIST operation, all system bistables are configured into a single MISR as shown in Fig. 2-4b. At each clock cycle during BIST operation, the outputs of all CLUs in the design are captured in a single MISR, and the outputs of the MISR provide test patterns to the CLUs. The circular BIST architecture requires little BIST control logic since there is only one test session during which a single MISR simultaneously performs the TPG and ORA operations. The problems of register self-adjacency and LFSR configuration discussed in Sec. 2.1 for the parallel BIST architecture are not applicable to circular BIST. However, since the CLUs in the circular BIST architecture are simultaneously tested by a single, self-adjacent MISR, the pattern generation and response analysis capabilities of self-adjacent MISRs must be analyzed to ensure an effective self-test operation. These issues are discussed in Sec. 3.1 and are addressed by our synthesis for scan dependence techniques described in Secs. 4 and 6.

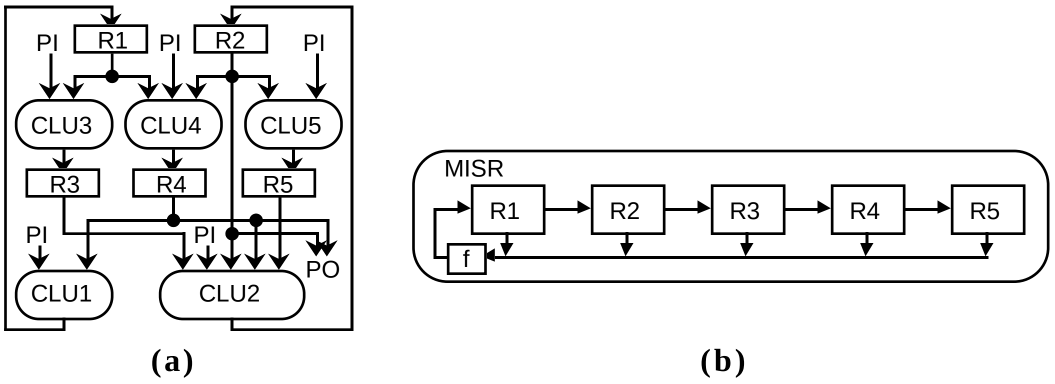


Figure 2-4 Circular BIST architecture: (a) normal operation; (b) BIST register configuration.

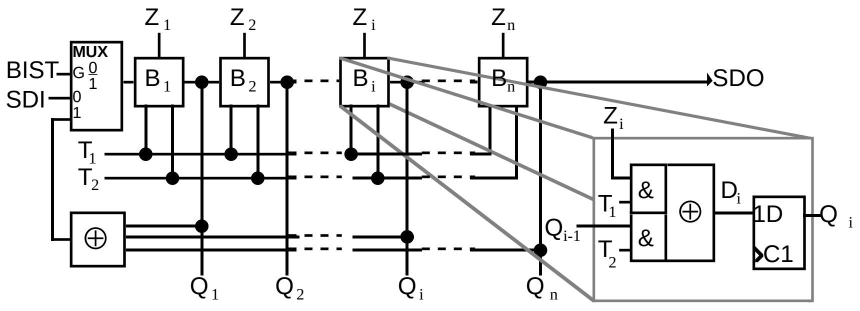
Since slightly different versions of the circular BIST architecture have been described in [Bardell 82], [Stroud 88], and [Krasniewski 89], we state here our assumptions concerning the circular BIST architecture implemented in our synthesized designs. First, we assume that all bistables in the design are included in the MISR during BIST operation. Other architectures ([Krasniewski 89], [Stroud 88]) allow some bistables to be configured in the normal mode during BIST operation in order to reduce the BIST overhead, but there is evidence that this can reduce the observability of the logic feeding those bistables [Kim 88]. Second, we assume that every system bistable is an edge-triggered flip-flop that can be configured to perform normal, shift, and MISR operations. A reset mode of operation is optional since the bistables can be controlled through the shift mode of operation. Using flip-flops simplifies the discussion of the synthesis techniques. Similar synthesis techniques can be applied if the double-latch, level-sensitive scan design method is used to implement the system bistables as described in [Bardell 82], but we do not discuss these techniques in this paper. Finally, we assume that the MISR feedback logic is simply a direct connection from the output of the last bistable in the MISR to the input of the first bistable. A reconfigurable register design that supports these operations is described in Sec. 2.3.

2.3 RECONFIGURABLE REGISTER DESIGN

The distinguishing characteristic of embedded BIST architectures is that the system bistables can be configured to perform both normal operation (parallel load) as well as test operations such as serial shift, TPG, and ORA. The test operations that must be supported by each bistable depend upon the embedded BIST architecture used. The Circular Self-Test Path architecture [Krasniewski 89], for example, does not require that the system bistables perform a serial shift operation. The built-in logic block observer (BILBO) register [Konemann 79] is often used in embedded BIST architectures and implements normal, synchronous reset, serial shift, TPG, and ORA modes of operation.

Figure 2-5 illustrates a reconfigurable register design that can be used for both the parallel BIST and circular BIST architectures. The register implementation is not necessarily optimal. Its purpose is to more easily illustrate the BIST operations. In Secs. 3 and 4, we use Fig. 2-5 to illustrate scan dependence and our synthesis techniques. Figure 2-5a shows the register configuration, where each bistable in the register is implemented as shown in Fig. 2-5b. The bistable could also be implemented with a dual-port latch, such as the design specified for the SST technique [Bardell 82]. The Q_{i-1} input to the first bistable (bistable B_1) in the register is determined by BIST mode select signal *BIST*, and is either the scan data input (*SDI*) signal for shift operation or the output of the MISR feedback logic for MISR operation or LFSR operation. For circular BIST architectures, since a separate TPG operation is not required, signal *BIST* can be replaced by signal T_1, resulting in four possible modes of operation: reset, shift, normal, and MISR. Test mode select signals T_1 and T_2 determine the D input to each bistable in the register, as specified in Fig. 2-5c, where Z_i is the system logic input to bistable B_i.

For circular BIST architectures and for parallel BIST architectures with self-adjacent registers that are not implemented with CBILBOs (i.e., for self-adjacent MISRs), the MISR operation must simultaneously provide both sufficient test patterns and accurate output response analysis. It has been proven that, when the Z inputs are independent of the state of an n-bit MISR, the MISR generates test patterns that are similar in behavior to random patterns [Krasniewski 89], [Kim 88]. Specifically, as P goes to infinity, where P is the number of test patterns applied during BIST operation, the probability of the MISR being in any one of the 2^n possible states is 2^{-n}. The Z inputs to a self-adjacent MISR, however, are a function of the state of the MISR. Simulation results presented in [Kim 88] and [Stroud 88] indicate that the MISR test pattern characteristics are not significantly different from random patterns when the Z inputs depend upon the state of the MISR. It is not clear whether a MISR implemented with a primitive polynomial [Kim 88] produces any better patterns than a MISR implemented with a single Q_n feedback connection [Krasniewski 89].

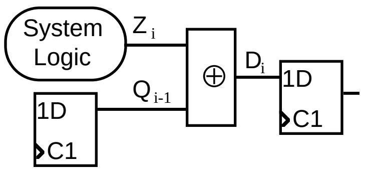


T_1T_2	D_i	Mode
00	0	Reset
01	Q_{i-1}	Shift or LFSR
10	Z_i	Normal
11	$Z_i \oplus Q_{i-1}$	MISR

(c)

Figure 2-5 Reconfigurable register: (a) bistable interconnections; (b) reconfigurable bistable; (c) modes of operation.

Figure 2-6a shows a portion of the reconfigurable register during MISR operation to illustrate its error capture capabilities. During MISR operation, the input to each bistable, D_i, is the exclusive-OR of the output of the previous bistable in the MISR, Q_{i-1}, and the system logic input to the bistable, Z_i. At each clock cycle, an error in either Z_i or Q_{i-1} is observable at D_i (i.e., captured in bistable B_i) regardless of the state of the error-free signal. Figure 2-6b is a truth table for signals D_i ($D_i = Q_{i-1} \oplus Z_i$) and DE_i ($DE_i = QE_{i-1} \oplus Z_i$), where QE_{i-1} is the faulty version of signal Q_{i-1}. Figure 2-6b shows that, regardless of the state of signal Z_i, D_i differs from DE_i, so when Z_i is fault-free, an error in Q_{i-1} is always captured in bistable B_i. Because the exclusive-OR operation is symmetric, the same is true when Q_{i-1} is fault-free and Z_i is faulty. However, errors in both signals Z_i and Q_{i-1} in a single clock cycle are not observable at D_i. Thus, once captured in a bistable of the MISR, an error is transferred from one bistable (Q_{i-1}) to the next (Q_i) at each clock cycle as long as it is not masked by a simultaneous error in the corresponding system logic input signal Z_i. In addition, the error may be transferred to other bistables in the MISR through the MISR feedback logic or, if the MISR is self-adjacent, through the system logic.



(a)

Q_{i-1}	QE_{i-1}	Z_i	D_i	DE_i
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	0	1

(b)

Figure 2-6 MISR error capture capabilities: (a) MISR configuration; (b) error response.

3 SCAN DEPENDENCE

Performance degradation, area overhead, and test transparency (the fraction of defects not detected by the test) are three issues that must be addressed when synthesizing a built-in self-testable design. Sharing of system logic and test logic can reduce the area and improve the performance of embedded BIST architectures without increasing the test transparency. In this section, we define scan dependence and discuss how to use scan dependence to increase the sharing of system and test logic.

A system bistable B_i is that can be reconfigured to perform scan operation is *scan dependent* if and only if its system logic input, Z_i, is functionally dependent on the bistable that immediately precedes it in the scan path. For example, assume that the system logic input to bistable B_5 is the function $Z_5 = Q_1 Q_3' + Q_2 Q_4$. If any of the bistables B_1 - B_4 immediately precede B_5 in the scan path, bistable B_5 is scan dependent on that preceding bistable. If, however, bistable B_6 immediately precedes B_5 in the scan path, B_5 is not scan dependent since Z_5 is not a function of Q_6. Bhatia [Bhatia 93] showed that scan dependence can be used to reduce the area overhead of certain types of scan architectures. We showed in [Avra 93] that some types of scan dependence can be used to reduce the overhead and improve the effectiveness of embedded BIST architectures.

Figure 3-1 illustrates a problem that can occur in embedded MISRs if the type of scan dependence is not identified when arranging bistables in the scan path. In this example, the system logic input to bistable B_2 is the function $Z_2 = Q_1 + f$, where f is an output of combinational logic (CLU), the inputs to which are primary inputs (PIs) and bistable outputs excluding the output of bistable B_1 (DFFs). When bistable B_2 is implemented as shown in Fig. 2-5b, the function of its D input during MISR mode ($T_1 = 1$, $T_2 = 1$) is $D_2 = Q_1 \oplus (Q_1 + f) = Q_1' f$, as shown in Fig. 3-1b. The problem with this function is that it greatly reduces the number of errors that the MISR can capture during MISR operation. An error previously captured by the MISR and located at bistable B_1 will only remain in the MISR (via transfer to bistable B_2) when $f = 1$. Similarly, an error in f is only captured in the MISR when $Q_1 = 0$.

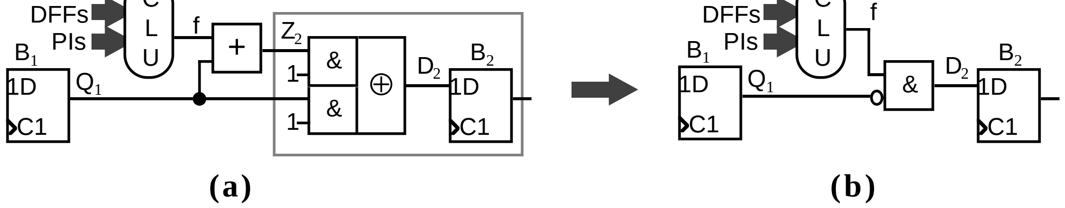


Figure 3-1 Embedded MISR with scan dependence: (a) implementation; (b) MISR operation.

In order to determine whether or not scan dependence could be a significant problem in embedded BIST architectures, we implemented the circular BIST architecture, as described in Sec. 2.2, on three of the ACM/SIGDA LGSynth91 benchmark circuits [ACM 91]. For each circuit, the order of the bistables in the MISR was selected to maximize the number of scan dependent bistables in order to represent a worst-case scenario. We then fault-simulated BIST operation, using the bistable outputs as observation points. For each circuit, BIST operation was first executed until all of the single stuck-at faults in the circuit had been captured at least once in the MISR (i.e., *detected* by the BIST operation). The number of BIST operation clock cycles, *P*, required to detect 100% of the single stuck-at faults varies from circuit to circuit, depending upon the testability of the combinational logic, and from simulation to simulation, depending upon the initial state of the bistables. To determine a value for *P*, we ran five fault simulations on the combinational logic portion of each circuit, applying test patterns randomly-generated by the fault simulator to the inputs of the circuit. We chose *P* to be equal to the average number, over the five simulations, of randomly-generated test patterns required to detect all single stuck-at faults in the combinational logic. We then fault-simulated the BIST operation of the entire circuit (combinational logic plus bistables) for *P+90* clock cycles, observing the states of all of the bistables during the last 90 clock cycles. The fault simulation results of these 90 clock cycles are shown in Fig. 3-2. At each clock cycle, the percentage of faults that alias is the percentage of faults for which the states of the bistables in the faulty and fault-free circuits are identical. Our motivation for performing the experiment in this manner is that, in practice, BIST operation would probably be terminated, and the test response shifted out, at some point during that window of 90 clock cycles. Figure 3-2 shows that, even though the TPG function of the BIST architecture is sufficient to detect all single stuck-at faults, there is a significant chance that a faulty circuit will not be identified due to aliasing in the ORA function. For example, if BIST operation is halted and the state of the circuit is observed at clock cycle *P+40*, there is a probability of approximately 40% for *mult32b*, 25% for *mult16b*, and 10% for *s641* that the faulty circuit will be indistinguishable from the fault-free circuit. We show in Sec. 6.3.2 that these probabilities are approximately zero for the same circuits if there is no scan dependence in the design.

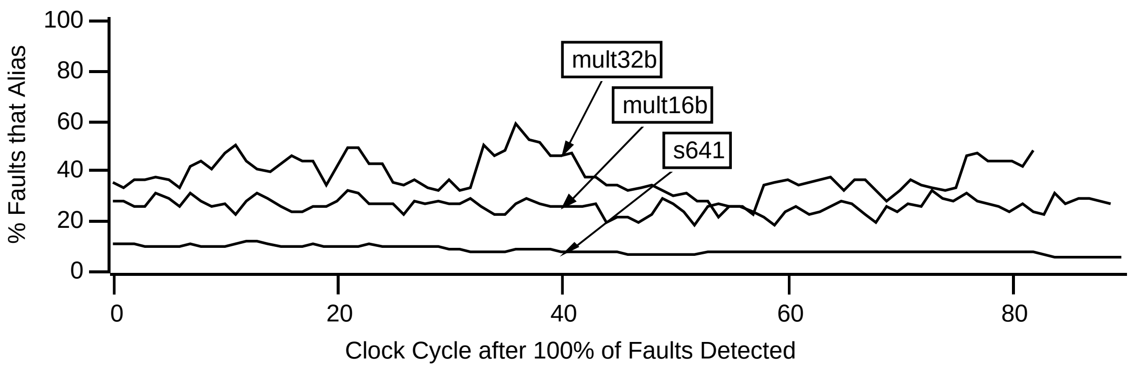


Figure 3-2 Fault simulation results of BIST operation of circular BIST circuits with maximum scan dependence.

One way to address the scan dependence problem is to arrange the bistables in the MISR such that a minimum number of bistables are scan dependent [Stroud 88], [Pilarski 92]. For example, in order to eliminate scan dependence for bistable B_2 ($Z_2 = Q_1 + f$) in Fig. 3-1, neither B_1 nor any of the bistables that are inputs to function f can immediately precede B_2 in the MISR. Unfortunately, it may not always be possible to create an embedded MISR that has no scan dependent bistables, particularly in “control-dominated” designs where bistable inputs are often a function of a significant number of the bistable outputs in the design. Also, eliminating all scan dependence eliminates opportunities to share system and test logic.

We propose to solve the problem of MISR aliasing due to scan dependence by using a different function for MISR mode for the scan dependent bistables. The MISR function used depends upon the type of scan dependence. For example, when the system logic for bistable B_2 has the general form $Z_2 = Q_1 + f$, where B_1 immediately precedes B_2 in the MISR, we use $Q_1 \oplus f$ as the MISR function. The Boolean function for the input to B_2 is then:

$$\begin{aligned} D_2 &= T_1' T_2' Q_1 + T_1' T_2 Q_1 + T_1 T_2' (Q_1 + f) + T_1 T_2 (Q_1 \oplus f) \\ &= T_1' Q_1 + T_1 T_2' (Q_1 + Q_1' f) + T_1 T_2 (Q_1 \oplus f) \\ &= Q_1 (T_1' + T_2' + f') + Q_1' (T_1 f) \end{aligned}$$

One possible logic implementation for this function is compared with a scan dependence avoidance implementation in Fig. 3-3. Figure 3-3a shows our implementation, where the MISR function is $D_2 = Q_1 \oplus f$. The scan dependence avoidance implementation is shown in Fig. 3-3b, where the MISR function is $D_2 = Q_x \oplus Z_2 = Q_x \oplus (Q_1 \oplus f)$, and f is not a function of Q_x. Both implementations were mapped to LSI Logic 1.0 micron standard cell gates [LSI 91]. Area (in LSI Logic cell units) and delay (f to D_2 in nanoseconds) for the shaded areas of Fig. 3-3 are provided in Table 3-1. Note that the circuit in Fig. 3-3a performs the shift operation ($D_2 = Q_1$) when $T_1 = T_2 = 0$, whereas the circuit in Fig. 3-3b performs a synchronous reset operation ($D_2 = 0$). Note also that bistable B_2 in Fig. 3-3a has greater observability of system logic errors during MISR mode than bistable B_2 in Fig. 3-3b. If bistable B_2 is not scan dependent, an error observable at f is only captured in the MISR when Q_1 is 0 (i.e., when the error is observable at Z_2), whereas if scan dependence is implemented as shown in Fig. 3-3a, all errors observable at f are captured in the MISR. Greater observability of system logic errors can reduce the test transparency of the BIST operation.

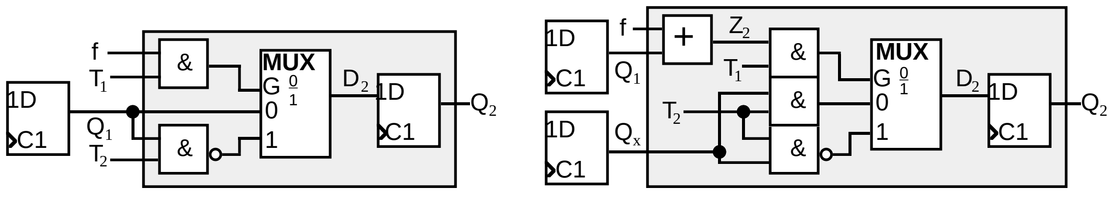


Figure 3-3 Scan dependence solutions for $Z_2 = Q_1 + f$: (a) our technique, MISR operation: $D_2 = Q_1 \oplus f$; (b) avoiding scan dependence, MISR operation: $D_2 = Q_x \oplus (Q_1 \oplus f)$.

Table 3-1 Area and delay for shaded logic in Fig. 3-3 based on data in [LSI 91].

Circuit	Area (CUs)	f$\rightarrow D_2$ Delay (ns)
Fig. 3-3a	33	1.88
Fig. 3-3b	41	2.21

The disadvantage of scan dependence, as shown in Fig. 3-2, is that it can result in significant error loss in embedded MISRs if the function for MISR mode is not specifically selected for the type of scan dependence. The advantage of scan dependence, as shown in Fig. 3-3, is that, when the MISR function is carefully selected, the delay and area of the test logic for the scan dependent bistable can sometimes be reduced. When arranging the bistables in the scan path, the synthesis technique must therefore analyze not only the bistable interconnection information, but also the system logic functions in order to determine whether or not scan dependence will be beneficial to BIST. The identification, analysis, and synthesis of this system logic is discussed in Sec. 4.

4 SYNTHESIS FOR SCAN DEPENDENCE

Scan dependence that may be beneficial for embedded MISRs must be identified early in the design cycle since it can affect how the system logic is implemented, as was illustrated in Fig. 3-3. Rather than analyzing a structural description of the system logic, the synthesis-for-BIST technique must analyze the system logic Boolean function for each system bistable to determine whether or not scan dependence is beneficial, and if so, to select the appropriate test mode functions. In Sec. 4.1, we describe how to identify beneficial scan dependence, and we specify the test mode functions for each scan dependence classification. Section 4.2 formalizes the problem of arranging bistables in the embedded MISR in order to maximize beneficial scan dependence. In Sec. 4.3, we discuss possible solutions when non-beneficial scan dependence is unavoidable.

4.1 SCAN DEPENDENCE CLASSIFICATION

We use Shannon decomposition to transform each system logic equation, Z_j, into a form that can be easily analyzed for scan dependence, and if beneficial for embedded BIST, synthesized into an efficient structural implementation. *Shannon decomposition* of a Boolean equation $Z_j(Q_1,Q_2,...)$ with respect to variable Q_1 results in:

$$Z_j(Q_1, Q_2, ...) = Q_1\, Z_j(1, Q_2, ...) + Q_1'\, Z_j(0, Q_2, ...)$$

where $Z_j(1, Q_2, ...)$ is the Q_1-residue and $Z_j(0, Q_2, ...)$ is the Q_1'-residue of Z_j. For each Q_i that is essential for system logic equation Z_j, $i \neq j$, we perform Shannon decomposition on Z_j with respect to Q_i. Then, based on the values of the residues, we classify Q_i as one of four cases according to Table 4-1. Q_i is *essential* for Z_j if and only if all sum-of-products expressions for Z_j include either Q_i or Q_i' or both. If Q_i cannot be classified as case 1, case 2, or case 3, it is classified as case 4.

The classification is then used to determine whether or not to have bistable B_j be scan dependent on bistable B_i. In Table 4-1, "constant" is either logic 0 or logic 1, and Q_i^* is either Q_i or Q_i' .

Table 4-1 Scan dependence classification of decomposed Boolean equations.

Class	Q_i-residue	Q_i'-residue	Equation Form	MISR Function
Case 1	constant	constant	$Z_j = Q_i^*$	$D_j = Q_i^*$
Case 2	constant not constant	not constant constant	$Z_j = (Q_i^* + f)^*$	$D_j = Q_i^* \oplus f$
Case 3	$(Q_i'$ -residue)'	$(Q_i$-residue)'	$Z_j = Q_i^* \oplus f$	$D_j = Q_i^* \oplus f$
Case 4	not constant	not constant	$Z_j = Q_i f + Q_i' g$	AVOID

Case 1 is the most obvious case of beneficial scan dependence and includes all system logic equations of the form $Z_j = Q_i^*$. Case-1 equations for all but the first bistable of the MISR are implemented as shown in Fig. 4-1a. Figure 4-1b shows the implementation for bistable B_1, where the equation for shift operation is $D_j = SDI$. Table 4-2 lists the bistable input functions for the normal, shift, and MISR operations of case-1 equations. The Q_i^* column in Table 4-2 specifies whether Q_i or Q_i' is used in Fig. 4-1. In each case, MISR operation is identical to the normal operation of the system logic, and there is no performance overhead or area overhead for BIST. The first bistable in the MISR requires an addition multiplexer for shift operation. Note that if the reconfigurable bistable of Fig. 2-5b were used for this case, the function for MISR operation would be $D_j = Q_i \oplus Q_i = 0$, and any errors captured in the MISR would be eliminated when transferred to bistable B_j. Also, the output of bistable B_j would never supply a logic 1 as a test pattern during MISR operation.

Table 4-2 Case 1 bistable operations: $Z_j = Q_i^*$.

Q_i^*	Normal	Shift	MISR
Q_i	$D_j = Q_i$	$D_j = Q_i$	$D_j = Q_i$
Q_i'	$D_j = Q_i'$	$D_j = Q_i'$	$D_j = Q_i'$

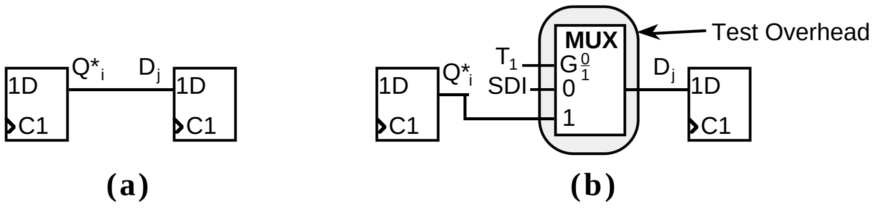


Figure 4-1 Case 1 bistable implementation: (a) all bistables except B_1; (b) bistable B_1.

Table 4-3 lists the four different case-2 system logic equations, which have the general form $Z_j = (Q_i^* + f)^*$, where f is a function of primary inputs and outputs of bistables excluding bistable B_i. Figure 4-2 shows one possible circuit implementation for case-2 equations, where L is either an AND gate or an OR gate, Q_i^* is either Q_i or Q_i' , T_1^* is either T_1 or T_1', and T_2^* is either T_2 or

T_2', as specified in Table 4-3. Note that the equation for the shift operation, the MISR operation, or both may be complemented, depending upon the normal operation equation. Complementation does not change the effectiveness of the shift and MISR operations, but, depending upon the areas and delays of the available logic gates, may have a significant effect on the test logic overhead. However, it may also complicate test pattern generation software that uses the scan path to shift test patterns into the design. We found that using the operations listed in Table 4-3 (complemented shift function for the second and fourth equations, complemented MISR function for the second and third equations) results in minimum performance and area overhead when the equations are mapped to the technology library in [LSI 91]. In general, the test logic overhead for case-2 equations (illustrated in Fig. 4-2) is lower than the overhead shown in Fig. 3-3b, so we consider case-2 scan dependence equations to be beneficial. Table 4-4 and Fig. 4-2b show the operations and implementation when the first bistable in the MISR is a case-2 function of the last bistable in the MISR.

Table 4-3 Case 2 bistable operations: $Z_j = (Q_i^* + f)^*$.

Q_i^*	Normal	Shift	MISR	L	T_1^*	T_2^*
Q_i	$D_j = Q_i + f$	$D_j = Q_i$	$D_j = Q_i \oplus f$	&	T_1	T_2
Q_i'	$D_j = Q_i' + f$	$D_j = Q_i'$	$D_j = Q_i' \oplus f$	&	T_1	T_2
Q_i'	$D_j = Q_i f$	$D_j = Q_i$	$D_j = Q_i' \oplus f$	+	T_1'	T_2'
Q_i	$D_j = Q_i' f$	$D_j = Q_i'$	$D_j = Q_i \oplus f$	+	T_1'	T_2'

Table 4-4 Case 2 bistable operations for the first bistable in the MISR.

Q_i^*	Normal	Shift	MISR	L	f^*	T_2^*
Q_i	$D_j = Q_i + f$	$D_j = SDI$	$D_j = Q_i \oplus f$	&	f	T_2
Q_i'	$D_j = Q_i' + f$	$D_j = SDI$	$D_j = Q_i' \oplus f$	&	f	T_2
Q_i	$D_j = Q_i f$	$D_j = SDI$	$D_j = Q_i' \oplus f$	+	f'	T_2'
Q_i'	$D_j = Q_i' f$	$D_j = SDI$	$D_j = Q_i \oplus f$	+	f'	T_2'

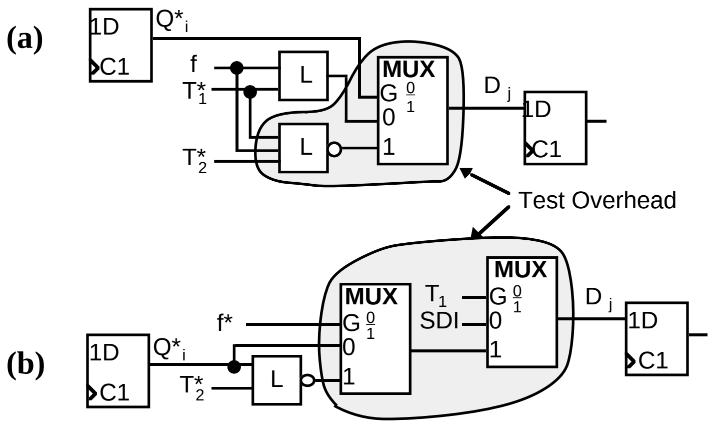


Figure 4-2 Case 2 bistable implementation: (a) all bistables except B_1; (b) bistable B_1.

Case-3 equations also represent beneficial scan dependence and include all system logic equations of the form $Z_j = Q_i^* \oplus f$, where f is a function of primary inputs and outputs of bistable excluding bistable B_i. The bistable input functions for the normal, shift, and MISR operations of case-3 equations are listed in Table 4-5, and the implementation of case-3 equations is shown in Fig. 4-3. As with case-1 equations, MISR operation is identical to the normal operation of the system logic. However, some overhead logic is required for the shift operation: a 2-to-1 multiplexer when the case-3 equation is for the first bistable in the MISR, and a 2-input AND gate when the case-3 equation is for any other MISR bistable. An uncomplemented shift function could be implemented for the second equation in Table 4-5 if Q_i is input to the XOR gate and $f\,'$ is input to the AND gate in Fig. 4-3a.

Table 4-5 Case 3 bistable operations: $Z_j = Q_i^* \oplus f$.

Q_i^*	Normal	Shift	MISR
Q_i	$D_j = Q_i \oplus f$	$D_j = Q_i$	$D_j = Q_i \oplus f$
Q_i'	$D_j = Q_i' \oplus f$	$D_j = Q_i'$	$D_j = Q_i' \oplus f$

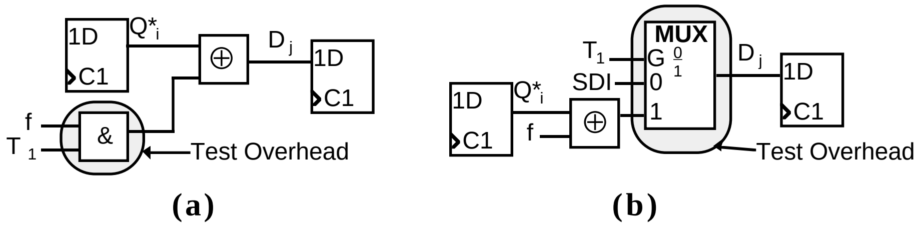


Figure 4-3 Case 3 bistable implementation: (a) all bistables except B_1; (b) bistable B_1.

To date, we have not determined a way for system logic equations classified as case-4 equations to benefit, in terms of reduced area and performance overhead, from scan dependence. Therefore, if the Shannon decomposition of Z_j with respect to Q_i is a case-4 equation, bistable B_i should not immediately precede bistable B_j in the scan path, otherwise the effectiveness of the

MISR will be compromised. To show that this is true, let p and q be the Q_i- and Q_i' -residues, respectively, of Z_j ($Z_j = Q_i\, p + Q_i'\, q$), and let B_i immediately precede B_j in the scan path. If B_j is implemented as shown in Fig. 2-5b, then during MISR operation, $D_j = Q_i \oplus (Q_i\, p + Q_i'\, q) = Q_i\, p' + Q_i'\, q$. Any error in p or q that is observable at Z_j (an error in p is observable when Q_i is 1; an error in q is observable when Q_i is 0) is also observable at D_j (i.e., captured in the MISR) during MISR operation. However, an error in Q_i is only observable at D_j (i.e., transferred to the next bistable in the MISR) when $p = q$, because when $p=q=0$, $D_j = Q_i$, and when $p=q=1$, $D_j = Q_i'$, whereas when $p=q'$, $D_j = 0$. If the bistables in the scan path are arranged such that B_i precedes bistable B_k where Q_i is not essential for Z_k, then during MISR operation, $D_k = Q_i \oplus Z_k$, and all errors in either Q_i or Z_k, but not both, are observable at D_k. Therefore, case-4 scan dependence reduces the observability of errors in the previous bistable in the scan path, but does not reduce the observability of system logic errors.

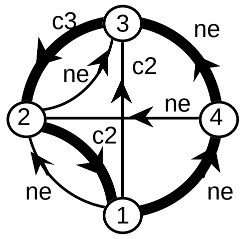
4.2 BISTABLE ARRANGEMENT

To determine the best arrangement of bistables in an embedded MISR for beneficial scan dependence, our synthesis procedure first creates a directed graph called a *dependence graph*, where the vertices of the graph represent the system bistables. We add a weighted edge from vertex i to vertex j if Z_j is a case-1, case-2, or case-3 function of Q_i (see Table 4-1). The weight of the edge is a cost function associated with the test logic overhead as illustrated in Figs. 4-1, 4-2, and 4-3. The cost function could represent the performance overhead, the area overhead, or a combination of the two. We also add a weighted edge from vertex i to vertex j if Q_i is not essential for Z_j. In this case, bistable B_j is not scan dependent on B_i, and the weight of the edge is a cost function associated with the test logic overhead of Fig. 2-5b. No edge is added from vertex i to vertex j if Z_j is a case-4 function of Q_i since this represents a scan dependence situation that we wish to avoid. The best order of bistables is determined by finding the lowest-cost Hamiltonian cycle in the graph, where the cost of the cycle is the sum of the weights of the edges in the cycle. A Hamiltonian cycle contains all vertices in the graph, and all vertices in a Hamiltonian cycle are distinct. Frieze [Frieze 88] describes a polynomial-time algorithm for finding Hamiltonian cycles in directed graphs.

Figure 4-4 is an example that illustrates how the dependence graph is used to determine the best order of bistables. Figure 4-4a gives the system logic functions for four system bistables. Figure 4-4b is the dependence graph, where each vertex represents a system bistable. A cost is assigned to each edge based on whether Q_i is a case-1, case-2 ("c2"), or case-3 ("c3") function of Z_j or whether it is non-essential ("ne") for Z_j. Since there is only one Hamiltonian cycle in this dependence graph, the only possible order of bistables in the MISR is $1 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 1$. Note that, for this example, it is not possible to completely avoid scan dependence as is required by the circular BIST techniques described in [Stroud 88] and [Pilarski 92]. To determine whether scan dependence avoidance is possible for a given design, the dependence graph must have a Hamiltonian cycle made up of only non-essential edges. No such cycle exists in the graph of Fig. 4-4b.

$$Z_1 = Q_2 + Q_3 Q_4$$
$$Z_2 = Q_2 \oplus Q_3$$
$$Z_3 = Q_1 Q_3'$$
$$Z_4 = Q_2 Q_3 Q_4 + Q_2' Q_3' Q_4'$$

(a)



(b)

Figure 4-4 Bistable arrangement example: (a) system logic functions; (b) dependence graph.

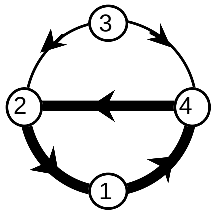
After the bistable order has been determined for the MISR, the dependence graph can be used to determine the first bistable in the MISR (i.e., which bistable receives *SDI* during shift operation) and which bistable output signals can be inputs to the MISR polynomial feedback logic. A Hamiltonian cycle in the dependence graph corresponds to a MISR with just the n^{th} bistable fed back to the input of the first bistable in the MISR. This is the polynomial feedback logic implemented in [Krasniewski 89] and in the circular BIST designs generated by our synthesis technique. As shown in Fig. 2-5a, the MISR function for the first bistable in the MISR is $D_1 = Z_1 \oplus FB$, where FB is the output of the feedback logic exclusive-OR tree. The inputs to the exclusive-OR tree are a subset of the outputs of the MISR bistables, and the subset depends upon the MISR polynomial implemented. For example, the MISR function for the first bistable of a 5-bit MISR with feedback polynomial $x^5+x^3+x^1+x^0$ is $D_1 = Z_1 \oplus Q_2 \oplus Q_4 \oplus Q_5$. A discussion of polynomial implementations is given in [McCluskey 86] and [Bardell 87]. An exclusive-OR tree has the characteristic that if an odd number of its inputs are in error, the error is observable at the output. Thus, if an odd number of inputs to FB are in error and there is no error in Z_1, the error is observable at D_1. Similarly, if Z_1 is in error, it is observable at D_1 if an even number of the FB inputs are in error. However, if any input, Q_i, to FB is essential for Z_1, then $Z_1 = Q_i p + Q_i' q$, where p and q are the Q_i- and Q_i'-residues, respectively, of Z_1 and the equation for D_1 becomes:

$$\begin{aligned} D_1 &= Z_1 \oplus FB \\ &= (Q_i p + Q_i' q) \oplus (Q_i \oplus FBx) \qquad \text{where } FBx = Q_i \oplus FB \\ &= (Q_i p' + Q_i' q) \oplus FBx \end{aligned}$$

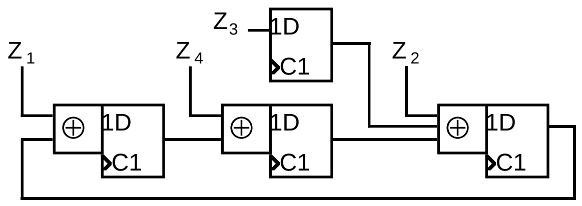
Since Q_i is not essential for FBx, an error in Q_i is only observable at D_1 when $p = q$ and there is no error in FBx. Thus, a dependence of Z_1 of any of the inputs to FB can reduce the observability of that input during MISR operation. We can avoid this situation by analyzing the dependence of Z_1 on all inputs to FB and using the methods described in Sec. 4.1 to determine the MISR function for D_1 based on the type of dependence. For example, if $Z_1 = Q_n + Q_3$, and $FB = Q_n \oplus Q_3$, the MISR function for bistable B_1 could be implemented as $D_1 = Q_n \oplus Q_3$.

4.3 NON-HAMILTONIAN DEPENDENCE GRAPHS

Unfortunately, some designs may have a dependence graph that does not contain a Hamiltonian cycle. There are various solutions to this problem depending upon the characteristics of the dependence graph. For example, no Hamiltonian cycle exists in a dependence graph that has a vertex with no incoming edges. One possible solution to this problem is illustrated in Fig. 4-5. Figure 4-5a is a dependence graph that does not have a Hamiltonian cycle because vertex *3* has no incoming edges. This means that Z_3 is a case-4 function of all other bistables in the MISR. Our solution is to remove bistable B_3 from the MISR since case-4 scan dependence occurs if any of the other bistables immediately precede B_3 in the scan path. We remove vertex *3* and its adjacent edges from the dependence graph and search for a Hamiltonian cycle in the remaining graph. Figure 4-5b shows a possible configuration during MISR mode. Errors in Z_3 are captured in the MISR during BIST operation by feeding the output of B_3 through an exclusive-OR gate into the MISR at bistable B_2. For this example, B_3 can feed either B_2 or B_4 as shown by the edges from vertex *3* to vertices *2* and *4* in Fig. 4-5a. Note that if errors occur simultaneously at bistables B_3 and B_4, they will cancel. Note also that, since B_3 is not included in the MISR, test pattern quality at the output of B_3 may be compromised during BIST operation, and logic must be added to include B_3 in the scan chain during scan operation.



(a)



(b)

Figure 4-5 Bistable 3 is a case-4 function of all other bistables: (a) dependence graph; (b) MISR design.

A more general solution for a non-Hamiltonian dependence graph is to add vertices to the graph, one at a time, until a Hamiltonian cycle is created. In the worst case, which corresponds to an original dependence graph with *n* vertices and no edges (each bistable is a case-4 function of every other bistable), *n* vertices must be added to create a Hamiltonian cycle. Adding a vertex to the dependence graph corresponds to adding a bistable to the design that is only used during test operations. Since the added bistable does not have a system logic function, it can perform a shift operation during both MISR mode and normal mode, so a case-1 edge is added from each vertex in the graph to the new vertex. Also, the output of the new bistable is not essential for any of the system bistable functions, so a “non-essential” edge is added from the new vertex to each “system” vertex in the graph. In the worst case, which corresponds to an original dependence graph with *n* vertices and no edges (each bistable is a case-4 function of every other bistable), *n* vertices must be added to create a Hamiltonian cycle, and the cycle will have alternating "system" and "test-only" vertices. Figure 4-6 illustrates this solution for the dependence graph of Fig. 4-5a. Figure 4-6a shows the dependence graph with the added vertex, *x*. For clarity, only the edges from the original dependence graph and the edges adjacent to vertex *x* that are used in the Hamiltonian cycle are shown in Fig. 4-6a.

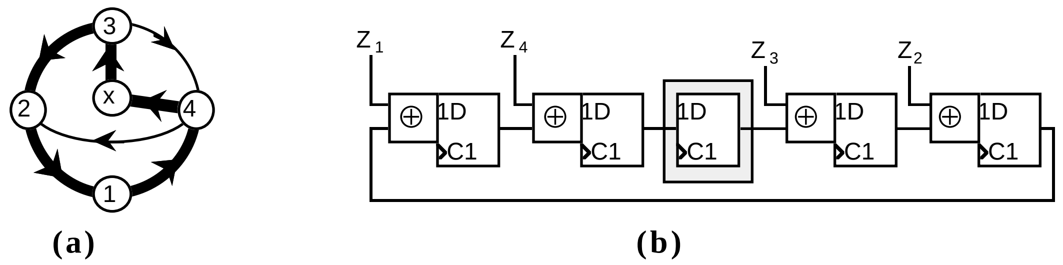


Figure 4-6 General solution for non-Hamiltonian dependence graphs: (a) dependence graph; (b) MISR design.

For maximum benefit, higher-level synthesis operations, such as state assignment and register binding, should be biased toward generating more system bistables with beneficial dependence equations. Eschermann [Eschermann 91] describes state assignment procedures for case-1 and case-3 equations, but does not consider using case-2 equations or avoiding the detrimental consequences of case-4 equations.

5 RESULTS

We have implemented both our synthesis technique (called *BSD* for *beneficial scan dependence*) and the synthesis technique of avoiding all types of scan dependence (called *AVD*) using procedures from the *SIS* logic synthesis tool [Sentovich 92]. The synthesis techniques use the categories listed in Table 4-2 to first identify the different dependence cases for each bistable input and create a dependence graph. Then, for the *BSD* technique, the bistables are arranged into a circular BIST scan path that has the maximum number of case-1, case-2, and case-3 functions, and no case-4 functions. For the *AVD* technique, the bistables are arranged into a circular BIST scan path that has no scan dependencies. For both techniques, "test-only" vertices are added one at a time to the dependence graph, if necessary, until a Hamiltonian cycle is created. We applied the synthesis techniques to the ACM/SIGDA LGSynth91 benchmark circuits [ACM 91] and provide both implementation results and fault simulation results in this section.

Table 5-1 lists the number of system and test-only bistables in the generated circular BIST versions of the benchmark circuits, and the number of beneficial scan dependence equations in the *BSD* versions of the benchmark circuits. Twelve of the 23 benchmark circuits have no instances of beneficial scan dependence in the circular BIST path selected by the *BSD* technique, so there is no difference between the *AVD* and *BSD* versions of these circuits. Additional implementation details of the remaining circuits, highlighted in Table 5-1, are provided in Tables 5-2, 5-3, and 5-4.

5.1 AREA AND DELAY RESULTS

Tables 5-2 and 5-3 compare area and delay values for the circular BIST benchmark circuits. For both tables, the *SIS* logic synthesis tool [Sentovich 92] was used to apply multi-level logic optimizations, and the *CERES* technology mapping tool [Mailhot 93] was used to map the logic to LSI logic 1.0 micron standard cell gates [LSI 91]. The area values are given in terms of LSI logic cell units, and the delay values are in nanoseconds. "% Overhead" is the percentage increase in area or delay of the circular BIST version of the circuit over the system logic version with no test logic added. "% Diff" is the percentage decrease in area or delay of the *BSD* version of the circuit over the *AVD* version. Area optimization techniques were used for both logic optimization and technology mapping for the circuits in Table 5-2, and delay optimization techniques were used for the circuits in Table 5-3.

Table 5-1 Bistable characteristics for circular BIST benchmark circuits.

	AVD Bistables		BSD Bistables		# BSD
Circuit	System	Test-Only	System	Test-Only	Eqns
s1196	18	0	18	0	0
s1488	6	6	6	6	0
s1494	6	6	6	6	0
s208.1	8	1	8	1	0
s298	***14***	***0***	***14***	***0***	***1***
s344	15	3	15	3	0
s349	15	3	15	3	0
s382	***21***	***0***	***21***	***0***	***3***
s386	***6***	***6***	***6***	***3***	***3***
s400	***21***	***0***	***21***	***0***	***3***
s420.1	16	1	16	1	0
s444	***21***	***0***	***21***	***0***	***3***
s510	6	6	6	6	0
s526	***21***	***0***	***21***	***0***	***1***
s641	***17***	***0***	***17***	***0***	***6***
s713	***17***	***0***	***17***	***0***	***8***
s820	5	5	5	5	0
s832	5	5	5	5	0
s838.1	32	1	32	1	0
sbc	***27***	***0***	***27***	***0***	***6***
mm4a	12	0	12	0	0
mult16b	***30***	***0***	***30***	***0***	***14***
mult32b	***62***	***0***	***62***	***0***	***30***

Table 5-2 Area and overhead for area-optimized circular BIST benchmark circuits.

	AVD		BSD		
Circuit	Area	% Overhead	Area	% Overhead	% Diff
s298	779	38.4	778	38.2	0.1
s382	1131	40.3	1110	37.7	1.9
s386	649	51.3	624	50.7	3.9
s400	1124	39.3	1111	37.7	1.2
s444	1137	44.3	1134	43.9	0.3
s526	1207	36.5	1206	36.4	0.1
s641	1534	42.6	1635	52.0	-6.6
s713	1539	43.4	1595	49.5	-3.6
sbc	3053	23.7	3045	23.4	0.3
mult16b	1516	46.7	1303	17.3	14.1
mult32b	3096	49.2	2648	16.5	14.5

Table 5-3 Delay and overhead for delay-optimized circular BIST benchmark circuits.

	AVD		BSD		
Circuit	Delay	% Overhead	Delay	% Overhead	% Diff
s298	6.8	19.0	6.8	20.2	0.0
s382	7.3	33.6	7.4	36.3	-2.1
s386	6.8	20.0	7.9	39.1	-15.9
s400	6.8	26.7	6.8	26.7	0.0
s444	7.5	25.8	7.1	19.5	5.0
s526	6.7	11.4	6.8	12.9	-1.3
s641	11.7	31.9	11.5	29.4	1.9
s713	11.0	24.8	11.0	24.7	0.1
sbc	11.8	10.1	11.8	10.0	0.1
mult16b	9.1	-0.4	8.6	-6.6	6.1
mult32b	14.2	-13.1	13.7	-16.1	3.5

Table 5-4 shows the gate area overhead, net area overhead, and total area overhead for layouts of the circular BIST benchmark circuits. The layouts were generated using *SIS* for area-optimized logic optimization, *CERES* for area-optimized technology mapping, and *TimberWolf 6.0* for standard cell place and route. The circuits were mapped to version 2.2 of the scalable CMOS standard cell library that is included with the *TimberWolf* tool. We used the best of five different layouts generated by *TimberWolf* for each circuit in Table 5-4.

Table 5-4 Layout area overhead for area-optimized circular BIST benchmark circuits.

	AVD % Overhead			BSD % Overhead			% Difference		
Circuit	Gate	Net	Total	Gate	Net	Total	Gate	Net	Total
s298	46.7	61.5	55.1	45.1	53.5	51.8	1.1	4.9	2.1
s382	44.8	62.4	53.7	44.3	62.3	53.7	0.4	0.1	0.0
s386	39.8	32.9	36.3	39.0	39.1	39.9	0.6	-4.7	-2.6
s400	45.1	51.5	49.7	43.8	55.7	52.9	0.9	-2.8	-2.1
s444	51.2	62.0	52.5	51.4	54.9	53.0	-0.1	4.3	-0.3
s526	39.5	53.5	49.3	38.3	53.8	49.3	0.9	-0.2	0.0
s641	45.1	57.0	54.3	55.3	71.9	68.2	-7.0	-9.5	-9.0
s713	44.4	56.5	53.9	53.9	68.5	67.3	-6.6	-7.6	-8.7
sbc	21.3	22.3	26.4	21.1	22.8	26.2	0.2	-0.4	0.2
mult16b	56.8	68.1	57.3	17.0	18.7	16.9	25.4	29.4	25.7
mult32b	57.3	84.5	74.5	16.1	15.2	14.6	26.2	37.6	34.3

For most of the benchmark circuits we investigated, the sizes of the *AVD* and *BSD* versions are not significantly different. Two of the *BSD* circuits (*s641* and *s713*) are larger than the *AVD* circuits. We determined that this is due to high fanout of some of the system logic equations that are re-structured for beneficial scan dependence in the circular BIST path according to Fig. 4-2. An enhancement to the synthesis technique should consider the fanout of the system logic when determining whether or not a particular system logic equation is beneficial for scan dependence. According to Tables 5-2, 5-3, and 5-4, the *BSD* versions of two of the circuits (*mult16b*, *mult32b*) have significantly less area and delay than the *AVD* versions of these circuits. The reason for this low overhead is that both *mult16b* and *mult32b* perform a serial multiplication operation which has the property that many of the system logic equations are case-3 equations. Our synthesis technique is able to arrange the bistables into a circular BIST path such that most of the bistables are case-3 scan dependent on the previous bistable in the path, and these case-3 equations have very little overhead for BIST (see Fig. 4-3). We focus on this property of arithmetic operations in the Sec. 6 discussion of scan dependence in data path logic.

5.2 FAULT SIMULATION RESULTS

We performed extensive fault simulations on several of the benchmark circuits in order to evaluate how scan dependence affects the quality of BIST operation. For each benchmark circuit, we created four different circuit models for gate-level, single stuck-at fault simulation: *FNL*, *IGN*, *AVD*, and *BSD*. The *FNL* version is the original benchmark circuit, and it executes normal operation during the fault simulation. The front-end test logic shown in Fig. 2-5b was used for every bistable in the *IGN* version (*IGN* stands for *ignore* scan dependence), and the bistables were arranged into a circular BIST path such that the number of scan dependent bistables was maximized. The *AVD* and *BSD* circuits were generated as described previously in this section. The fault simulation results are given in Table 5-5. The fault coverage is the percentage of collapsed, single stuck-at faults that are detected at least once during BIST operation, and is a measure of the quality of the TPG operation. We define *error loss* as the percentage of clock cycles for which the faulty and fault-free machines are identical (i.e., the percentage of aliased clock cycles) given that the fault has been detected. Error loss is a measure of the quality of the ORA operation. Table 5-5 shows that the *BSD* versions of the benchmark circuits have the highest fault coverage, and for most of the circuits, the lowest error loss. The slightly elevated error loss for the *BSD* version of circuit *s386* could be attributed to the fact that its circular BIST path has three fewer bistables than the *AVD* version. Even if a circuit has a high fault coverage percentage, a non-zero error loss percentage means that there is a chance that a defective circuit will be labeled fault-free even if the fault is detected at some time during the BIST operation. The defective circuit can only be identified if the BIST operation terminates on a non-aliased state.

Table 5-5 Fault simulation results for circular BIST benchmark circuits.

	% Fault Coverage				% Error Loss			
Circuit	FNL	IGN	AVD	BSD	FNL	IGN	AVD	BSD
s298	33.3	100.0	100.0	100.0	49.1	0.6	0.2	0.0
s386	37.6	30.2	77.8	80.2	59.2	0.2	0.0	2.8
s444	35.5	100.0	100.0	100.0	50.5	0.0	0.0	0.0
s641	88.9	100.0	100.0	100.0	73.4	9.5	0.0	0.0
mult16b	100.0	98.6	100.0	100.0	37.9	26.0	0.0	0.0
mult32b	99.0	95.7	100.0	100.0	1.9	39.7	0.0	0.0

6 SCAN DEPENDENCE IN DATA PATH LOGIC

The synthesis technique described in Sec. 4 is a general technique for maximizing beneficial scan dependence given that the system logic is described in terms of Boolean equations and bistables. One way to maximize beneficial scan dependence in data path logic is to first apply high-level synthesis operations to a data flow description of the design, then apply the technique described in Sec. 4 to the generated data path logic. Unfortunately, due to the types of function blocks typically found in data path logic (e.g., multipliers, adders), generating the dependence graph could be very computationally expensive. More importantly, the high-level synthesis technique may generate logic with very few beneficial dependencies between the bistables. There are several advantages when beneficial scan dependence is considered during high-level synthesis of data path logic, where the input to synthesis is a data flow description, rather than Boolean equations. First, we can simplify the generation of the dependence graph since the Boolean equations for the system logic can be inferred for multiple bistables given a single operation in a data flow equation. For example, given the data flow equation *R3* = *R1* ADD *R2*, where *R1*, *R2*, and *R3* are *n*-bit registers, we can infer that the system logic equation for each bistable, B_i, of register *R3* has the form $R3_i = R1_i \oplus R2_i \oplus C_i$, where C_i is the carry function for B_i. We need not perform Shannon decomposition on the system logic equations of the addition operation in order to determine the dependence information for the bistables in *R3*. Second, in data path logic, all bistables in a register are typically controlled by the same group of control signals. When test operation control signals can be used in the same manner, i.e., one block of test control logic for each register rather than for each bistable in the design, test overhead can be reduced. Finally, since the synthesis technique determines how variables in the data flow description are assigned to bistables, it may be able to generate data path logic with more instances of case-1, case-2, and case-3 dependencies between the bistables, resulting in a lower-cost, self-testable design.

This section contains a discussion of how to apply the synthesis technique described in Sec. 4 to data path logic. We assume that an *orthogonal scan path*, where the shift direction in the orthogonal scan path is orthogonal to the shift direction in traditional scan paths (shifting bits within registers), is implemented in the data path logic. We then identify data path functions that are well-suited for the synthesis technique described in Sec. 4, and show that the occurrence of these functions can be increased if an orthogonal scan path is used in the data path logic. Finally, we describe a register binding algorithm that, assuming an orthogonal scan path architecture, attempts to maximize the number of beneficial scan dependence equations in the generated data path logic.

6.1 ORTHOGONAL SCAN PATH ARCHITECTURE

Figure 6-1a is an example of logic that is commonly found in data path designs. The outputs of two registers, *R1* and *R2*, are the inputs to a combinational logic unit, *Adder*, that performs the addition operation, the output of which is stored in register *R3*. Each of the registers consists of *n* bistables. Traditional scan path architectures are arranged as shown in Fig. 6-1b, where each bistable of register *R1* feeds the next bistable of register *R1* during scan operation. When BIST is implemented in the data path logic, this scan path arrangement implies a correspondence between system registers and BIST registers, such as LFSRs and MISRs. For example, registers *R1* and *R2* could be implemented as LFSRs and *R3* could be implemented as a MISR during BIST operation if the parallel BIST architecture is used.

Figure 6-1c shows the bistable arrangement for an orthogonal scan path. Each bistable of *R1* feeds the corresponding bistable of register *R3* during scan operation. The outputs of *R3* may, in turn, be inputs to another register in the data path logic during scan operation. Note that, for this example, the flow of data during normal operation (from *R1* to *R3*) is parallel to the flow of data during scan operation. This may allow for the sharing of system and test logic, which is the motivation for the orthogonal scan path arrangement.

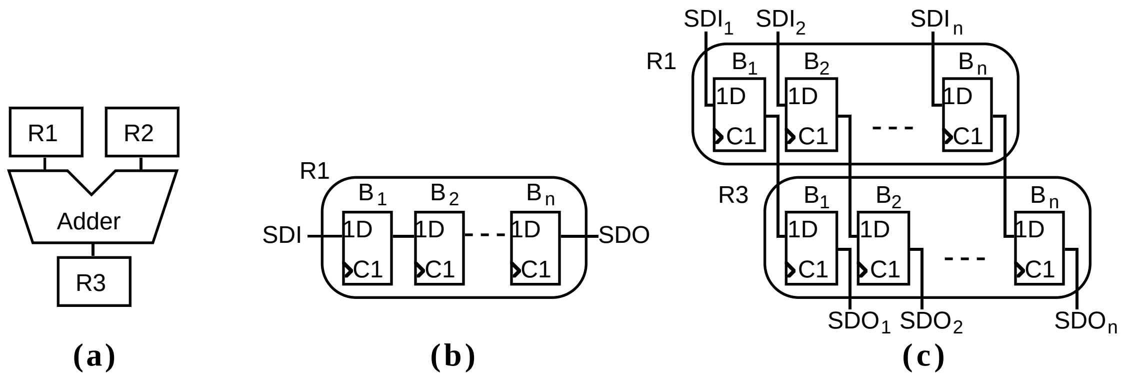


Figure 6-1 Orthogonal scan path example: (a) normal operation; (b) typical scan path arrangement for *R1*; (c) orthogonal scan path arrangement for *R1*.

6.2 SCAN DEPENDENCE FUNCTIONS IN DATA PATH LOGIC

The system logic equations for the bistables in register *R3* in Fig. 6-1a have the general form $R3_i = R1_i \oplus R2_i \oplus C_i$, where C_i is the carry function for bistable B_i of *R3*. Since C_i is a function of the less-significant bistables of *R1* and *R2*, and is not a function of $R1_i$ or $R2_i$, each bistable in *R3* is a case-3 function of the corresponding bistable in *R1* and the corresponding bistable in *R2*. The implementation for case-3 equations shown in Fig. 4-3 can only be used if the bistables in either *R1* or *R2* immediately precede the corresponding bistables in *R3* in the scan path. This can be accomplished when an orthogonal scan path is used in the data path logic.

Table 6-1 lists seven types of system logic equations for bistables in data path logic that we have determined can benefit from scan dependence. The equation type and an associated area overhead cost estimate based on LSI Logic cell units [LSI 91] are listed, where *n* represents the width of the data path (i.e., the number of bistables in each register). Lower-case letters *a* and *b* represent system logic control signals that determine the input to each bistable of register *R1*. Upper-case letters *F* and *G* represent outputs of combinational logic, the inputs to which are primary inputs and outputs of bistables excluding the corresponding bistables associated with register *R2*. For example, for a 2-bit data path (*n* = 2), the equation $R1 = a'\,R2 + a\,F$ represents two system logic equations:

$$Z1_0 = a'\,Q2_0 + a\,F_0$$
$$Z1_1 = a'\,Q2_1 + a\,F_1$$

where F_0 is not a function of $Q2_0$, and F_1 is not a function of $Q2_1$.

Data flow equation types 1, 7, and 3 correspond to the case-1, case-2, and case-3 equations, respectively, described in Sec. 4.1, and the bistables of these registers can be implemented as shown in Figs. 4-1, 4-2, and 4-3. Equation types 2, 4, 5, and 6 are combinations of case-1 and case-3 equations and have control logic can be shared between the test control signals and system control signals *a* and *b*. If the system logic equation for *R1* cannot be classified as one of the first seven types of equations shown in Table 6-1, it is classified as type 8, and the more general synthesis technique described in Sec. 4 can be applied to determine whether or not beneficial scan dependence exists for any of the bistables in the register. Register *R3* in Fig. 6-1a can be represented by the type-3 equation in Table 6-1, $R1 = R2 \oplus F$.

Table 6-1 System equations for data path registers that benefit from scan dependence.

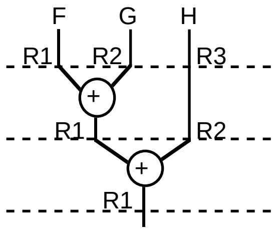
Type	Cost	Function
1	0	R1 = R2
2	4	R1 = a' R2 + a (R2 ⊕ F)
3	4n	R1 = R2 ⊕ F
4	12+2n	R1 = a' b' R2 + a' b (R2 ⊕ F) + a G
5	4+8n	R1 = a' (R2 ⊕ F) + a G
6	8+8n	R1 = a' R2 + a F
7	10n	R1 = R2 + F
8	18n	R1 = F

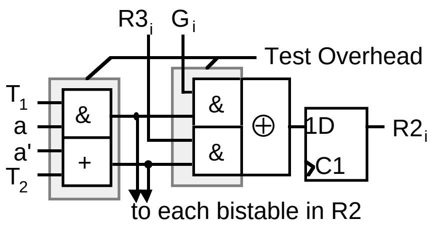
We have identified two types of operations that are common in data path designs and can be mapped to equation types 1-6 in Table 6-1: addition operations and live variable motion. Addition operations are common in DSP-based data path designs. [Waser 82] describes several types of adder implementations, including conditional sum, Ling, and carry-look-ahead adders. The output equations for each of these adder implementations can be expressed as: $R1_i = R2_i \oplus R3_i \oplus C_i$, where C_i is a function of the less-significant bistables of *R2* and *R3*, and is not a function of $R2_i$ or $R3_i$. The logic implementation for C_i varies with adder type. If any of the variables in the data flow description that are assigned to register *R1* are the result of an addition operation, it may be possible to implement *R1* by one of equation types 2-5 in Table 6-1.

Another type of function that can occur in data path designs is the transfer of data from one register to the next, known as *live variable motion*. Variables whose lifetimes are longer than one clock cycle can either be held in a single register, or transferred from one register to another. When the latter technique is used, equations of the form $R1 = a\ R2 + a'\ F$ occur in the data path logic, where a live variable is transferred from *R2* to *R1* whenever control signal $a = 1$. Depending on the logic that generates *F*, this equation can be mapped to equation types 1, 2, 4, 5, 6, or 7 in Table 6-1. Figure 6-2a is a data flow graph illustrating live variable motion. Variable *H* is loaded into register *R3* during the first clock cycle, but is not needed for computation until the second clock cycle, and so is loaded into *R2* during the second clock cycle. Variable *G* is loaded into *R2* during the first clock cycle. The system logic equations for the bistables of *R2* are type-6 equations, which can be implemented as shown in Fig. 6-2b. The control signal *a* is high during the first clock cycle and low during the second. The test logic overhead for this case consists of two AND gates per bistable in *R2* plus an AND gate and an OR gate to generate control signals for all bistables in the register. The exclusive-OR gate is used to perform the MISR function during BIST operation and a multiplexer function during normal operation.

6.3 RESULTS

Our high-level synthesis procedure for beneficial scan dependence assumes that the data path logic has an orthogonal scan path architecture, and biases the register binding operation to assign variables in the data flow equations to registers such that equation types 1-7 listed in Table 6-1 are used most often. This synthesis procedure was applied to two high-level synthesis benchmark circuits, *Tseng* [Tseng 86] and *DiffEq* [Paulin 89]. The results are presented in Table 6-2. The size of the data path (in LSI Logic cell units [LSI 91]) and the overhead for test logic as a percentage of total size are given for two situations: 1) bistables are arranged into orthogonal scan paths after register binding (labeled "Test Synthesis"), and 2) equation types 1-7 in Table 6-1 are favored during register binding (labeled "Synthesis-for-Test"). We assumed a 16-bit data path, and estimated interconnect area as being one cell unit per interconnection. For both the test synthesis and synthesis-for-test procedures, beneficial scan dependence was maximized by using an orthogonal scan path arrangement in a circular BIST architecture. For both examples, the total size and the test overhead are smaller when scan dependence is considered during register binding.





(a) **(b)**

Figure 6-2 Live variable motion: (a) data flow graph; (b) *R2* bistables.

Table 6-2 Data path design examples with orthogonal scan path and circular BIST.

	Test Synthesis		Synthesis-for-Test	
Design	Size (cell units)	% Overhead	Size (cell units)	% Overhead
Tseng	14,146	13.5	11,574	6.4
DiffEq	12,342	11.7	11,518	7.3

7 CONCLUSIONS

We have introduced new synthesis techniques for generating low-cost, built-in self-testable designs that are free of the types of system bistable dependencies that can reduce the effectiveness of the embedded MISRs that are used to perform BIST operations. We showed that some types of system bistable dependencies can reduce the effectiveness of BIST, whereas other types of dependencies can allow sharing of BIST and scan logic with system logic, thereby reducing BIST overhead. Care must be taken, however, when system logic and BIST logic is shared, so that the BIST operation effectiveness is not compromised. Our logic synthesis technique (Sec. 4) and our high-level synthesis technique (Sec. 6) both attempt to maximize sharing of system logic and test logic to reduce the cost and improve the performance without increasing the test transparency of BIST architectures, such as circular BIST and parallel BIST, that make use of embedded MISRs.

We have implemented the synthesis technique described in Sec. 4 in a system-for-test design tool and used it to generate circular BIST implementations of the sequential logic synthesis benchmark circuits. For most of the benchmark circuits we investigated, the sizes of the circuits generated by our technique did not differ significantly from the sizes of circuits with no scan dependence. However, better results for our technique should be possible when higher-level synthesis operations, such as state assignment and register binding, are biased toward generating more system bistables with beneficial dependence equations. Results presented in Sec. 5 showed that this is true for the register binding operation when an orthogonal scan path configuration is used. The orthogonal scan path allows for greater sharing of BIST logic and commonly-used data path logic such as multiplexers and adders.

While our synthesis techniques are applicable to any BIST architecture that uses embedded MISRs, our implementation of the technique generates circular BIST designs because circular BIST has far fewer complex implementation issues than the parallel BIST architecture. Unfortunately, theoretical results for TPG fault coverage and ORA aliasing are not valid for circular BIST because of the dependence of the MISR inputs on the state of the MISR. Extensive fault simulations of the circular BIST implementation must therefore be executed to guarantee high-quality BIST operation. Fault simulations of the designs generated by our logic synthesis technique show that the test transparency is comparable to circular BIST architectures that allow no scan dependence. These simulations also showed that for two common design practices, not including any BIST circuitry, or if circular BIST is implemented, ignoring scan dependence, fault coverage can be high, but high error loss reduces the overall quality of the BIST operation. This suggests that further investigation into techniques for improving the ORA characteristics of self-adjacent MISRs could yield lower-cost, self-testable designs.

ACKNOWLEDGMENTS

The authors wish to thank Piero Franco, Siyad Ma, Nur Touba, and Xingning Xu for their useful comments and suggestions. This work was supported in part by the Innovative Science and Technology Office of the Strategic Defense Initiative Organization and administered through the Office of Naval Research under Contract No. N00014-85-K-0600, and in part by the National Science Foundation under grant Nos. MIP-8709128 and MIP-9107760.

REFERENCES

[Abadir 85] Abadir, M. S., and M. A. Breuer, “A Knowledge-Based System for Designing Testable VLSI Chips,” *IEEE Des. and Test of Comput.*, pp. 56-68, August 1985.

[Abramovici 90] Abramovici, M., M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design*, Computer Science Press, New York, NY, USA 1990.

[ACM 91] ACM/SIGDA 1991 Logic Synthesis Benchmark Circuits, available via anonymous ftp from mcnc.mcnc.org.

[Avra 91] Avra, L., “Allocation and Assignment in High-Level Synthesis for Self-Testable Data Paths,” *Int. Test Conf.*, Nashville, TN, USA, pp. 463-472, October 26-30, 1991.

[Avra 92] Avra, L., “Orthogonal Built-In Self-Test,” *COMPCON Spring 1992 Dig. of Papers*, San Francisco, CA, USA, pp. 452-457, February 24-28, 1992.

[Avra 93] Avra, L. J., and E. J. McCluskey, “Synthesizing for Scan Dependence in Built-In Self-Testable Designs,” *Int. Test Conf.*, Baltimore, MD, USA, pp. 734-743, October 17-21, 1993.

[Bardell 82] Bardell, P. H., and W. H. McAnney, “Self-Testing of Multichip Logic Modules,” *Int. Test Conf.*, pp. 200-204, November 1982.

[Bardell 87] Bardell, P. H., W. H. McAnney, and J. Savir, *Built-In Test for VLSI: Pseudorandom Techniques*, John Wiley & Sons, Inc., 1987.

[Bardell 91] Bardell, P. H., and M. J. Lapointe, “Production Experience with Built-In Self-Test in the IBM ES/9000 System,” *Int. Test Conf. Proc.*, Nashville, TN, USA, pp. 28-36, October 26-30, 1991.

[Bhatia 93] Bhatia, S. and N. K. Jha, “Synthesis of Sequential Circuits for Robust Path Delay Fault Testability,” *6th Int. Conf. on VLSI Des.*, pp. 275-280, January 1993.

[Bonnenberg 93] Bonnenberg, H, A. Curiger, N. Felber, H. Kaeslin, R. Zimmermann, and W. Fichtner, “VINCI: Secure Test of a VLSI High-Speed Encryption System,” *Int. Test Conf.*, Baltimore, MD, USA, pp. 782-790, October 17-21, 1993.

[Brayton 87] Brayton, R. K., R. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang, “MIS: A Multiple-Level Logic Optimization System,” *IEEE Trans. on Comput.-Aided Des.*, Vol. CAD-6, No. 6, pp. 1062-1081, November 1987.

[Broseghini 93] Broseghini, J., and D. H. Lenhert, “An ALU-Based Programmable MISR/Psedudorandom Generator for a MC68HC11 Family Self-Test,” *Int. Test Conf.*, Baltimore, MD, USA, pp. 349-358, October 17-21, 1993.

[De Micheli 94] De Micheli, Giovanni, “Synthesis and Optimization of Digital Circuits,” McGraw-Hill, Inc., Hightstown, NJ, USA, 1994.

[Eschermann 91] Eschermann, B., and H.-J. Wunderlich, “A Unified Approach for the Synthesis of Self-Testable Finite State Machines,” *28th ACM/IEEE Des. Autom. Conf.*, San Francisco, CA, USA, pp. 372-377, June 17-21, 1991.

[Frieze 88] Frieze, A. M., “An Algorithm for Finding Hamiltonian Cycles in Random Directed Graphs,” *J. of Algorithms*, pp. 181-204, June 1988.

[Gage 93] Gage, R., “Structured CBIST in ASICs,” *Int. Test Conf.*, Baltimore, MD, USA, pp. 332-338, October 17-21, 1993.

[Gelsinger 86] Gelsinger, P. P., “Built In Self Test of the 80386,” *Int. Conf. Comput. Des.*, pp. 169-173, 1986.

[Gupta 91] Gupta, R., and M. A. Breuer, “Ordering Storage Elements in a Single Scan Chain,” *IEEE Int. Conf. Comput.-Aided Des.*, Santa Clara, CA, USA, pp. 408-411, November 11-14, 1991.

[Hao 91] Hao, H., and E. J. McCluskey, “‘Resistive Shorts’ Within CMOS Gates,” *Int. Test Conf.*, Nashville, TN, USA, pp. 292-301, October 26-30, 1991.

[Hudson 87] Hudson, C. L. Jr., and G. D. Peterson, “Parallel Self-Test with Pseudo-Random Test Patterns,” *Dig. Int. Test Conf.*, Washington, DC, USA, pp. 954-963, September 1-3, 1987.

[IEEE 90] IEEE Standard 1149.1-1990, “IEEE Standard Test Access Port and Boundary Scan Architecture,” Institute of Electrical and Electronics Engineers, Inc., New York, NY, USA, 1990.

[Illman 91] Illman, R., T. Bird, G. Catlow, S. Clarke, L. Theobald, and G. Willetts, “Built-In Self-Test of the VLSI Content Addressable Filestore,” *Int. Test Conf. Proc.*, Nashville, TN, USA, pp. 37-46, October 26-30, 1991.

[Kim 88] Kim, K, D. S. Ha, and J. G. Tront, “On Using Signature Registers as Pseudorandom Pattern Generators in Built-in Self-Testing,” *IEEE Trans. on Comput.-Aided Des.*, Vol. 7, No. 8, pp. 919-928, August 1988.

[Konemann 79] Konemann, B., J. Mucha, and G. Zwiehoff, “Built-In Logic Block Observation Techniques,” *1979 IEEE Test Conf.*, Cherry Hill, NJ, USA, pp. 37-41, 1979.

[Konemann 80] Konemann, B., J. Mucha, and G. Zwiehoff, “Built-In Test for Complex Digital Integrated Circuits,” *IEEE J. of Solid-State Circuits*, Vol. SC-15, No. 3, pp. 315-319, June 1980.

[Krasniewski 85] Krasniewski, A., and A. Albicki, “Automatic Design of Exhaustively Self-Testing Chips with BILBO Modules,” *Int. Test Conf.*, Philadelphia, PA, USA, pp. 362-370, November 19-21, 1985.

[Krasniewski 89] Krasniewski, A., and S. Pilarski, “Circular Self-Test Path: A Low-Cost BIST Technique for VLSI Circuits,” *IEEE Trans. on Comput.-Aided Des.*, Vol. 8, No. 1, pp. 46-55, January 1989.

[Lake 86] Lake, R., “A Fast 20K Gate Array with On-Chip Test System,” VLSI Systems Design, pp. 46-55, June 1986.

[Langford 93] Langford, T., “Utilizing Boundary Scan to Implement BIST,” *Int. Test Conf.*, Baltimore, MD, USA, pp. 167-173, October 17-21, 1993.

[LSI 91] *LSI Logic 1.0-Micron Cell-Based Products Databook*, LCB007 Cell-Based ASICs, February 1991.

[Mailhot 93] Mailhot, F., and G. De Micheli, “Algorithms for Technology Mapping Based on Binary Decision Diagrams and on Boolean Operations,”*IEEE Trans. on Comput.-Aided Des.*, Vol. 12, No. 5, pp. 599-620, May 1993.

[McCluskey 81] McCluskey, E. J., and S. Bozorgui-Nesbat, “Design for Autonomous Test,” *IEEE Trans. on Comput.*, Vol. C-30, No. 11, pp. 866-874, November 1981.

[McCluskey 85] McCluskey, E. J., “Built-In Self-Test Structures,” *IEEE Des. and Test*, pp. 29-36, April 1985.

[McCluskey 86] McCluskey, E. J., *Logic Design Principles with Emphasis on Testable Semicustom Circuits,* Prentice-Hall, Englewood Cliffs, NJ, USA, 1986.

[McCluskey 88] McCluskey, E. J., S. Makar, S. Mourad, and K. D. Wagner, “Probability Models for Pseudo-Random Test Sequences,” *IEEE Trans. on Comput.-Aided Des.*, Vol. 7, No. 1, pp. 68-74, January 1988.

[McFarland 88] McFarland, M. C., A. C. Parker, and R. Camposano, “Tutorial on High-Level Synthesis,” *25th ACM/IEEE Des. Autom. Conf.*, Anaheim, CA, USA, pp. 330-336, June 12-15, 1988.

[Mujumdar 92] Mujumdar, A., K. Saluja, and R. Jain, “Incorporating Testability Considerations in High-Level Synthesis,” *Int. Symp. Fault-Tolerant Comput.*, Boston, MA, USA, pp. 272-279, July 8-10, 1992.

[Narayanan 92] Narayanan, S., R. Gupta, and M. Breuer, “Configuring Multiple Scan Chains for Minimum Test Time,” *Int. Conf. on Comput.-Aided Des.*, Santa Clara, CA, USA, pp. 4-8, November 8-12, 1992.

[Nozuyama 88] Nozuyama, Y., A. Nishimura, and J. Iwamura, “Design for Testability of a 32-Bit Microprocessor, the TX1,” *Int. Test Conf. Proc.*, Washington, DC, USA, pp. 172-182, September 12-14, 1988.

[Papachristou 91] Papachristou, C. A., S. Chiu, and H. Harmanani, “A Data Path Synthesis Method for Self-Testable Designs,” *28th Des. Autom. Conf.*, San Francisco, CA, USA, pp. 378-384, June 17-21, 1991.

[Patel 93] Patel, R, and K. Yarlagadda, “Testability Features of the SuperSPARC Microprocessor,” *Int. Test Conf.*, Baltimore, MD, USA, pp. 773-781, October 17-21, 1993.

[Paulin 89] Paulin, P. G., and J. P. Knight, “Force-Directed Scheduling for the Behavioral Synthesis of ASICs,” *IEEE Trans. on Comput.-Aided Des.*, Vol. 8, No. 6, pp. 661-679, June 1989.

[Pilarski 92] Pilarski, S., A. Krasniewski, and T. Kameda, "Estimating Testing Effectiveness of the Circular Self-Test Path Technique," *IEEE Trans. on Comput.-Aided Des.*, Vol. 11, No. 10, pp. 1301-1316, October 1992.

[Preissner 92] Preissner, J., G.-H. Huamann-Bollo, G. Mahlich, J. Schuck, H. Sahm, P. Weingart, D. Weinsziehr, J. Yeandel, R. Evans, "An Open Modular Test Concept for the DSP KISS-16V2," Int. Test Conf., Baltimore, MD, USA, pp. 678-683, September 20-24, 1992.

[Ratiu 90] Ratiu, I. M., and H. B. Bakoglu, "Pseudorandom Built-in Self-Test Methodology and Implementation for the IBM RISC System/6000 Processor," *IBM J. Res. Develop.*, Vol. 34, No. 1, pp. 78-84, January 1990.

[Sentovich 92] Sentovich, E.M., J. K. Singh, C. Moon, H. Savoj, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Sequential Circuit Design Using Synthesis and Optimization," *Int. Conf. on Comput. Des.*, Los Alamitos, CA, USA, pp. 328-333, 1992.

[Sinaki 92] Sinaki, G., "C-17A Mission Computer Built-in Test and Fault Management Strategies," *IEEE National Aerospace and Electronics Conf.*, Dayton, OH, USA, pp. 822-828, May 18-22, 1992.

[Starke 90] Starke, C. W., "Design for Testability and Diagnosis in a VLSI CMOS System/370 Processor," *IBM J. Res. Develop.*, Vol. 34, No. 2/3, pp. 355-362, March/May 1990.

[Stroud 88] Stroud, C. E., "Automated BIST for Sequential Logic Synthesis," *IEEE Des. and Test of Comput.*, pp. 22-32, December 1988.

[Tseng 86] Tseng, C.-J., and D. P. Siewiorek, "Automated Synthesis of Data Paths in Digital Systems," *IEEE Trans. on Comput.-Aided Des.*, Vol. CAD-5, No. 3, pp. 379-395, July 1986.

[Waicukauski 89] Waicukauski, J. A., E. Lindbloom, E. B. Eichelberger, and O. P. Forlenza, "A Method for Generating Weighted Random Test Patterns," *IBM J. Res. Develop.*, Vol. 33, No. 2, pp. 149-161, March 1989.

[Wang 86] Wang, L.-T., and E. J. McCluskey, "Concurrent Built-In Logic Block Observer (CBILBO)," *Int. Symp. on Circuits and Systems*, San Jose, CA, USA, pp. 1054-1057, May 5-7, 1986.

[Yokomizo 92] Yokomizo, K., and K. Naito, "A 333 MHz, 72 Kb BiCMOS Pipelined Buffer Memory with Built-in Self Test," *Symp. on VLSI Circuits*, Seattle, WA, USA, pp. 32-33, June 4-6, 1992.

[Waser 82] Waser, S., and M. Flynn, *Introduction to Arithmetic for Digital Systems Designers*, Holt, Rinehart, and Winston, 1982.

[Zhu 88] Zhu, X.-A., and M. A. Breuer, "A Knowledge-Based System for Selecting Test Methodologies," *IEEE Des. and Test of Comput.*, pp. 41-59, October 1988.