An Effort-Minimized Logic BIST Implementation Method

Xinli Gu, Sung Soo Chung, Frank Tsang
Cisco Systems, Inc.
170 W. Tasman Dr.
San Jose, CA 95134, USA

Jan A. Tofte, Hamid Rahmanian
Mentor Graphics Corp.
1001 Ridder Park Drive
San Jose, CA 95131, USA

Abstract

This paper presents LBIST (Logic Built-In Self Test) design practice at Cisco Systems. It focuses on the LBIST design tasks that could affect design schedules and efforts. These are design timing closure and signature mismatch debugging. Our timing closure technique guarantees timing closure for LBIST insertion without any iteration between synthesis and LBIST insertion. In addition, it guarantees that only one iteration between static timing analysis and LBIST insertion is required to close all timing violations. The signature mismatch debugging technique effectively identifies the causes by indicating the pattern, the scan flip-flop and its operation mode, where the mismatch happens. These techniques save design efforts and the product-to-market time. We have integrated this method into an ASIC design flow. The results of using this flow in a large tele-communication design are described.

due to the design complexity introduced. The hardware overhead is negligible compared to the design sizes and the silicon technologies used today. The extra design effort is a major obstacle of using the LBIST technique.

One of the most difficult tasks in LBIST design is timing closure. The major timing violations introduced by LBIST are due to; 1) test point insertions, 2) x-bounding logic insertions, and 3) multi-cycle paths. Due to the features of the pseudo-random pattern generator, it is hard to test random resistant logic in a design. This prevents achieving high test coverage. Test points for both control and observe purposes are inserted into functional paths to improve design testability. However, it can introduce functional path timing violations. The second type of timing violations in an LBIST design is caused by the x-bounding logic. The MISR requires that no “X” values be propagated into it. Otherwise, the signature will be corrupted. Typical “X” values could be introduced by floating nets, latches, non-scannable flip-flops, primary inputs which are not controlled on a tester or a system board, memories which are treated as black boxes and buses where contents could happen. The x-bounding logic is used to block the “X” value propagation. In many designs, timing budgets for memory access and I/O are often very tight. The x-bounding logic could affect design timing. Another type of timing violations in LBIST design is related to false paths and multi-cycle paths in functional designs. In functional mode, timing analysis is not performed on these paths. In other words, even though there are timing violations on these paths, they are ignored during the timing analysis in the functional mode. In test mode, however, all paths in the design need to be tested. Therefore, their timing should be considered. In normal scan design, this is not a problem because the test clock frequencies are very slow. In LBIST design, the test clock frequencies are at or close to the functional clock frequencies. This could cause timing violations and result in an unpredictable signature.

Another difficult and time consuming task in LBIST

1. Introduction

LBIST (Logic Built-In Self Test) is a technique that uses the built-in test logic in a design to test the design itself. In LBIST, test vectors are generated by a PRPG (Pseudo-Random Pattern Generator) in an design and the responses from the design are compressed by a MISR (Multiple Input Shift Register) [1]. The signature in the MISR is then compared with the golden signature to determine the detection of any defects in the ASIC. This technique receives the following major benefits: 1) improving test quality by performing at-speed test, 2) reducing manufacturing test costs by reducing tester usage time, tester memory requirement and the required number of tester I/O, and 3) reducing system test, diagnosis and repair costs by performing test on a system board. However, it has several disadvantages; 1) hardware overhead due to the test logic inserted, and 2) extra design efforts

Paper 36.1
1002
ITC INTERNATIONAL TEST CONFERENCE
0-7803-7169-0/01 $10.00 ©2001 IEEE
design is signature mismatch debugging. Signature mismatch happens when the expected/calculated signature differs from the simulated one. There are various reasons that can cause signature mismatches, such as library problems, simulation environment setup problems and design errors. Since simulation only compares the final signature to determine simulation pass or failure, it is hard to find out when (at which pattern and shift cycle), where (on which scan chain and cell) and how (during capture, shifting, PRPG generation or MISR compression) the signature mismatch happens.

The practical issues previously mentioned have not been fully addressed so far. By minimizing the number of test points to be inserted, the number of timing violations caused by test points can be reduced [3]. However, there is no guarantee to resolve all of them. The timing-driven test point selection approach proposed in [4] can solve the timing violations caused by test points, but it cannot be applied to x-boundary timing violation since X-boundary logic has to be inserted to prevent "X" propagation. By suppressing the clock, the approach proposed in [5] can avoid capturing "X" values that will propagate into the MISR. However, adding logic to clock trees to suppress capturing will increase clock tree skew. The work in [2] provides a solution to fix multi-cycle path timing problem. It is based on a list of multi-cycle paths plus the information about the number of cycles for each multi-cycle path to disable the captures. By suppressing the capture of a transmitting flip-flop, the transmitting flip-flop holds its previous state for extra number of cycles to allow receiving flip-flops to capture data safely. When a transmitting flip-flop performs capture, the capture of the receiving flip-flops will be suppressed so that there is no potential for a hold-time violation. This approach can achieve full coverage. However, the multiplexer added in the front of the transmitting flip-flops could cause timing problem. Secondly, it is very hard to obtain the multi-cycle paths and their cycle number information in reality. In many cases, ASIC designers use a simplified approach to specify the paths that can be ignored. For example, all paths from clock domain A to clock domain B are false paths. This specification could involve thousands or even tens of thousands of paths. To identify all multi-cycle paths among these false paths and their cycle numbers, one has to analyze timing. After place and route, timing results will be changed. Hence, the test logic inserted for fixing multi-cycle path timing problem based on the timing analysis result before place and route does not reflect the timing of the final netlist and this will cause signature mismatch.

This paper presents solutions for these practical issues. A trade-off between fault coverage, design efforts and cost is made based on the motivations of using LBIST and the manufacturing environment at Cisco Systems. The timing closure is achieved by a constrained LBIST insertion technique based on timing analysis. It guarantees timing closure for LBIST insertion without any iteration between synthesis and LBIST insertion, but with limited loss in coverage. In addition, it guarantees that only one iteration between static timing analysis and LBIST insertion is required to close all timing violations due to LBIST insertion. The method for signature mismatch debugging uses several techniques such as parallel and serial scan simulations, and simulations at different hierarchical module levels, to identify the problems in affordable simulation time. It indicates the failing pattern, and scan cell and its operation mode (shift, capture, receiving from PRPG or compressing at MISR) for which the mismatch happens.

The timing closure technique has been integrated into an ASIC design flow to minimize design efforts. Figure 1 illustrates a design flow diagram where the shaded portions are LBIST related. Timing analysis is performed on synthesized netlist. These results will later be used to separate timing violations introduced by LBIST insertion from the violations that exist in the original netlist. In parallel to the timing analysis, scan and LBIST is inserted to the netlist. There is no LBIST constraints during the initial LBIST insertion. Timing analysis is performed for the new netlist which has LBIST inserted. This result together with the previous timing analysis result is then converted into an LBIST insertion constraint file to be used to guide the second LBIST insertion and some necessary netlist processing. It is guaranteed that the second LBISTed netlist has no timing violations in both functional mode and LBIST mode, provided the initial netlist is timing clean. Otherwise, the remaining timing violations are those that exist in the original netlist. This technique has been applied to a large tele-communication design that contains 1.25 million logic gates with over 1,400 I/O pins.

To outline the scope of this paper, we will first describe
our motivations and LBIST usage limitations in our environment. This will be followed by a generic LBIST architecture. The techniques for timing analysis, constrained LBIST insertion and signature mismatch debugging will then be presented. Finally, we will give an industrial design case and conclusion.

2. Motivations of Using LBIST

The main motivations behind the LBIST usage at Cisco Systems are 1) to reduce the cost of system level test, diagnosis and repair, and 2) to eliminate the expensive tester usage. Boards and systems are becoming more and more complex. This results in not only the increased cost of testing, diagnosing and repairing the boards, but also the cost for un-repairable boards due to the limited board level diagnosis capability. Board level connectivity test can be performed by IEEE149.1 boundary scan. However, we lack the capability of performing device test at the board level. If we can perform very simple chip test on the board, even with low fault coverage, we can enhance board level diagnosis capability dramatically. Industrial results show that if devices have been fully tested during semiconductor manufacturing, it is rare that the new defects introduced during system assembling or system life-cycle usage can escape from a test method with 95 percent coverage. The majority of devices on a board are ASICs and external memories. If these devices can be tested on the board, we could achieve our goal. Our strategy is to test external memories by memory BIST located on the ASICs that access the memories and test ASICs by LBIST built in the ASICs.

The increasing number of ASIC I/O pins requires using expensive testers. Testers get more expensive when the number of I/O connections to a tester are over 1,000. Using cheaper testers (typically with 500--800 or less I/O pin connections) to perform manufacturing test for ASICs with over 1,000 I/O pins can cut the cost dramatically, especially when the ASIC volumes are considered. LBIST can help us to achieve this since under LBIST test, primary input pins do not need to be controlled by a tester and the values on primary output pins do not need to be strobed by the tester. They are all controlled by the LBIST. Besides, reducing the number of test cycles and increasing the test clock speed can shorten the tester usage time and save test cost. In LBIST design, it is possible to use high speed test clocks, but reducing the number of test cycles could affect the fault coverage.

Our goal is to find an easy and inexpensive LBIST implementation to achieve 95 percent fault coverage. The LBIST will be used both at semiconductor manufacturing test and at board level test. By reducing the number of pseudo-random test vectors, we can cut tester usage time, but still need to guarantee 95 percent fault coverage. If the fault coverage drops below 95%, we can insert more test points to complement the coverage loss due to the test vector reduction. The area overhead used by test points can be ignored compared to our design sizes, typically from one million to ten million logic gates. One concern of using too many test points is timing impact and the cost of routing of control signals to the test points. The following sections in this paper will present a technique to quickly resolve timing problems caused by LBIST insertion. To achieve 99 percent or higher fault coverage for semiconductor manufacturing test, we need to use a few hundred additional deterministic ATPG vectors on top of the pseudo-random vectors.

Using high-speed test clock and short scan chains can also help reducing tester usage time. The limitation of the test clock speed comes from timing violation during captures and power dissipation during scan shifting. The maximum capture speed is at functional speed and maximum shifting speed is 40MHz-60MHz. In LBIST, the number of scan chains is not limited by the I/O pins. We can configure several hundred scan chains in LBIST mode to reduce the scan shifting time. Similar to test points, too many scan chains will require a lot of routing areas between PRPG/MISR and scan chains.

3. LBIST General

This section describes the LBIST controller, and the different types of test logic the LBIST tool inserts into the core design. Control and observe points are inserted to improve the testability of the core, and x-bound logic is inserted to block propagation of unknown values from x-sources.

3.1. LBIST Insertion Flow and Architecture

The LBIST architecture used in Cisco designs is based on STUMPS, which stands for Self-Test Using MISR/Parallel SRSF (shift register sequence generator). The STUMPS architecture is shown in Figure 2. The scan chains of the core design form STUMPS channels, that are fed by pseudo-random patterns from the PRPG. The captured responses from the scan cells are shifted out through the STUMPS channels and compressed in the MISR. Since the number of channels are normally much larger than the bit-length of the PRPG, a phase shifter is used to distribute data from the PRPG to the STUMPS channels. Similarly, a space compactor is used to compact the outputs from a large number of STUMPS channels to the tap inputs of the MISR. The BIST controller is synchronized to the core by incorporating lockup latches between the phase shifter and the core, and between the compactor and the core. Throughout this paper, we will frequently refer to
scan chains instead of the more uncommon term STUMPS channels.

3.2. Control Points

Pseudo-random patterns are less effective than deterministic ATPG patterns. In order to achieve acceptable fault coverage, we use MTPI (Multi-Phase Test Point Insertion) to insert control and observe points in the core design. The MTPI technique uniformly divides the patterns into multiple phases. In each phase, a different set of control and observe points are enabled. For example, a 2-phase MTPI implementation using 32k patterns divides the test into 2 phases with 16k patterns each. Similarly, using 4 phases will divide the test into 4 phases of 8k patterns each.

The logic inserted for MTPI control points is illustrated in Figure 3. There are two types of control points, AND-type and OR-type. An AND-type control point is used to force a logic 0 control, and an OR-type control point is used to force a logic 1 control. Phase decoder signals enable the control points. The phase decoder signals for the 2-phase MTPI example shown in Figure 3 are named phase_cntl1 and phase_cntl2. One-hot encoding of the phase decoder signals ensures that only control points belonging to a particular phase are enabled while the other control points are disabled. The control points active in each phase contribute to the overall fault coverage. For the example shown in Figure 3, the following control points are enabled during the different MTPI phases:

\[
\begin{align*}
\text{phase} \_\text{cntl1} = 0 \text{ and phase} \_\text{cntl2} = 0 & \Rightarrow \text{phase 0 or functional (all control points disabled)} \\
\text{phase} \_\text{cntl1} = 1 \text{ and phase} \_\text{cntl2} = 0 & \Rightarrow \text{phase 1 (C 11 and C 12 enabled)} \\
\text{phase} \_\text{cntl1} = 0 \text{ and phase} \_\text{cntl2} = 1 & \Rightarrow \text{phase 2 (C 21 and C 22 enabled)}
\end{align*}
\]

3.3. Observe Points

For observe points, a variety of test logic can be inserted by MTPI. The designs at Cisco use primarily two types of observe points. These observe points make use of either new scan cells or existing scan cells, and are illus-

![Diagram of Phase Controlled Control Points](image)

![Diagram of Observe Points](image)
3.4. X-bounding Logic

An important design requirement for LBIST is that "X" values cannot propagate to scan cells in the STUMPS channels. If an "X" value is captured into a scan cell, it will corrupt the MISR signature. X-bounding logic is inserted to block the propagation of x-sources. Internal x-sources can be blocked by multiplexing in a new signal from an existing scan cell. In LBIST mode, the signal from the scan cell is selected. Primary inputs are also considered x-sources, and they are always x-bounded by multiplexing in data from existing scan cells. The advantage of using the multiplexing technique (as opposed to pin constraining primary inputs) is increased controllability. We use multiplexing for all internal x-sources as well. The X-bounding using a multiplexer and an existing scan cell is shown in Figure 5.

4. Timing Analysis and Constrained LBIST Insertion

After the initial LBIST insertion, timing analysis is performed on the LBISTed netlist. If there are timing violations caused by the LBIST insertion, we need to analyze the violations and constrain the LBIST insertion. In the following, a timing analysis technique and a constrained LBIST insertion technique are described.

4.1. Timing Analysis

We need to run three timing analyses in order to analyze the LBIST related causes of the timing violations. The first timing analysis is performed on the netlist before LBIST insertion. The second one is on the netlist after LBIST insertion and still in functional mode. Except the netlist difference, the timing analysis scripts for these two are the same. The third timing analysis is run on the netlist after LBIST insertion, but under LBIST mode. This includes the mode change and adding all paths including false paths and multi-cycle paths in timing analysis. The LBIST constraint file is obtained from these timing analysis reports. Before describing obtaining the LBIST constraint file, we will first give some definitions.

Definition 1: A path $P$ is a sequence of device instances $D_1, D_2, ..., D_n$ that are connected in the sequence;

$$P = D_1, D_2, ..., D_n$$

where $D_1$ can be a flip-flop, a memory output or a primary input, $D_n$ can be a flip-flop, a memory input or a primary output, and others are combinational devices. We denote the sequence order

$$D_i \rightarrow D_j$$

if $D_i$ is prior to $D_j$ in the sequence.

Below is an example of a path, where $D_1$ is a memory and $D_5$ is a flip-flop.

$D_1 = \text{phas_outsync_reg_bist/RF32X70_U1}$
$D_2 = \text{pi_choc_0/US213}$
$D_3 = \text{pi_choc_0/US657}$
$D_4 = \text{pi_choc_0/US969}$
$D_5 = \text{pi_choc_0/cc_pbr_rd_ptr_3_reg[10]}$

Definition 2: A test point $TP$ is a device $D$ that can be inserted into a path for testability purpose. A test point can be either a control point, an observe point or an X-bound ing.

Definition 3: A test point $TP = D$ is on a path $P = D_1, D_2, ..., D_n$ if and only if

$$\exists i \in [1,n]: D = D_i$$

This is denoted by $TP \in P$.

Definition 4: Given a path $P_1 = D_{11}, D_{12}, ..., D_{1n}$, path $P_2 = D_{21}, D_{22}, ..., D_{2(n+1)}$ is path $P_1$ with a test point $TP = D$ inserted if and only if

$$D_{ji} = D_j, i \in [1,a], l \leq a \leq n,$$
$$D_{2(a+1)} = D, \text{ and}$$
$$D_{1j} = D_{2(i+1)}, j \in [a+1,n].$$

This is denoted by $P_1 \subset P_2$. Otherwise, $P_2$ is not the same path as $P_1$ with a test point inserted, and it is denoted by $P_1 \nsubset P_2$. Path $P_1$ can have several test points inserted into it $P_1 \subset P_2 \subset \ldots \subset P_n$. This is also denoted by $P_1 \subset P_n$. In the following example, $P_1 \subset P_2 \subset P_3$ is $P_4$ after $D$ is inserted.
P1:

\[\begin{align*}
D_1 &= pbus_outsync_reg_bist/RF32X70_U1 \\
D_2 &= pi_cloc_0/0x123; \\
D_3 &= pi_cloc_0/0x5657; \\
D_4 &= pi_cloc_0/0x9696; \\
D_5 &= pi_cloc_0/cc_pbr_rd_ptr_3_reg[10]
\end{align*}\]

P2:

\[\begin{align*}
D_1 &= pbus_outsync_reg_bist/RF32X70_U1 \\
D_2 &= pi_cloc_0/0x123; \\
D &= pi_cloc_0/Utest48; \\
D_3 &= pi_cloc_0/0x5657; \\
D_4 &= pi_cloc_0/0x9696; \\
D_5 &= pi_cloc_0/cc_pbr_rd_ptr_3_reg[10]
\end{align*}\]

Definition 5: \(P_{\text{func}}\) is a set of timing violation paths reported from timing analysis in functional mode for the netlist before LBIST insertion.

Definition 6: \(P_{\text{blist}}\) is a set of timing violation paths reported from timing analysis in functional mode for the netlist after LBIST insertion.

Static test signals controlling LBIST operations are included in a false path list during timing analysis for the netlist after LBIST insertion. Therefore they should not appear in \(P_{\text{blist}}\).

Definition 7: \(P_{\text{false}}\) is a set of timing violation paths reported from timing analysis in LBIST mode after LBIST insertion.

4.2. LBIST Constraint File

Heuristic 1: \(P_{\text{blist}} - P_{\text{func}}\) is a set of functional timing violation paths caused by LBIST insertion:

\[P_{\text{blist}} - P_{\text{func}} = \{ P \mid (P \in P_{\text{blist}}) \land (\forall P \in P_{\text{func}} : (P \not\subset P)) \land (P \not\subset P) \}\]

\(P_{\text{blist}} - P_{\text{func}}\) is a subset of \(P_{\text{blist}}\) which contains the paths that are not in \(P_{\text{func}}\) or the paths in \(P_{\text{func}}\) with test points inserted. Therefore they are new timing violation paths after LBIST insertion. As an example, if \(P_{\text{blist}} = \{P_2, P_3, P_a, P_b, P_c\}, P_{\text{func}} = \{P_1, P_2, P_3, P_4\} \text{ and } P_4 \subset P_b\), then \(P_{\text{blist}} - P_{\text{func}} = \{P_1, P_2, P_3, P_4\}\).

The paths in \(P_{\text{func}}\) with test points inserted are not included in \(P_{\text{blist}} - P_{\text{func}}\). Their timing violations will not be considered from LBIST insertion point of view unless functional timing violations are fixed. This could introduce an extra iteration of timing analysis. After fixing timing violations for the original netlist, LBIST insertion could cause timing violation again. An easy solution is to include them in \(P_{\text{blist}} - P_{\text{func}}\) so that LBIST insertion will not insert test points into these paths so that their timing violations are only functional related. Thus

\[P_{\text{blist}} - P_{\text{func}} = \{ P \mid (P \in P_{\text{blist}}) \land (\forall P \in P_{\text{func}} : P \not\subset P) \}\]

Using this calculation, for the same example above, we will have \(P_{\text{blist}} - P_{\text{func}} = \{P_a, P_b, P_c\}\).

To convert the timing violation paths caused by LBIST insertion to an LBIST constraint file, we need to identify which control points, observe points and x-bounds have caused timing violations, respectively. Assume \(TP_{\text{ctrl}}\), \(TP_{\text{obs}}\) and \(TP_{\text{xbd}}\) are a set of control points, observe points and x-bounds inserted in the initial LBIST insertion. The following three heuristics are used to perform this conversion.

Heuristic 2: The set of control points that were initially inserted and cause timing violations can be identified by

\[TP_{\text{ctrl_vio}} = \{ TP \mid (TP \in TP_{\text{ctrl}}) \land (\exists P \in (P_{\text{blist}} - P_{\text{func}}) : TP \in P) \}\]

\(TP_{\text{ctrl_vio}}\) is a subset of \(TP_{\text{ctrl}}\) and they appear in \(P_{\text{blist}} - P_{\text{func}}\) causing timing violations.

Heuristic 3: The set of observe points that were initially inserted and cause timing violations can be identified by

\[TP_{\text{obs_vio}} = \{ TP \mid (TP \in TP_{\text{obs}}) \land (\exists P \in (P_{\text{blist}} - P_{\text{func}}) : TP \in P) \}\]

Similar to \(TP_{\text{ctrl_vio}}\), \(TP_{\text{obs_vio}}\) is a subset of \(TP_{\text{obs}}\) and they appear in \(P_{\text{blist}} - P_{\text{func}}\) causing timing violations.

Heuristic 4: The set of x-bounds that were initially inserted and cause timing violations can be identified by

\[TP_{\text{xbd_vio}} = \{ TP \mid (TP \in TP_{\text{xbd}}) \land (\exists P \in (P_{\text{blist}} - P_{\text{func}}) : TP \in P) \}\]

Similar to \(TP_{\text{ctrl_vio}}\) and \(TP_{\text{obs_vio}}\), \(TP_{\text{xbd_vio}}\) is a subset of \(TP_{\text{xbd}}\) and they appear in \(P_{\text{blist}} - P_{\text{func}}\) causing timing violations.

\(TP_{\text{ctrl_vio}}, TP_{\text{obs_vio}}\) and \(TP_{\text{xbd_vio}}\) will be included in the LBIST insertion constraint file. The constrained LBIST insertion will use these informations to make sure no timing violation caused by the constrained LBIST insertion.

For timing violations in false paths or multi-cycle
paths, they are all considered as multi-cycle path timing violations and can be identified by the following heuristic.

**Heuristic 5:** The set of timing violation multi-cycle paths in LBIST mode can be identified by

\[ P_{\text{false}} \cdot P_{\text{lbist}} = \{ P | (P \in P_{\text{false}}) \land (\forall P \in P_{\text{lbist}}: P \neq P) \} \]

\( P_{\text{false}} \cdot P_{\text{lbist}} \) will also be included in the LBIST insertion constraint file.

**4.3. Constrained LBIST Insertion**

To avoid timing violations caused by control and observe point insertions, we remove those points that are identified by \( TP_{\text{ctrl_vio}} \) and \( TP_{\text{obs_vio}} \) from the test point insertion list. Thus, the new control and observe points to be inserted in the constrained LBIST insertion (the second insertion) are \( \{ TP_{\text{ctrl}} \cdot TP_{\text{ctrl_vio}} \} \) and \( \{ TP_{\text{obs}} \cdot TP_{\text{obs_vio}} \} \), respectively.

For the inserted x-bounding boundaries that cause timing violations, we first remove them from the x-bounding list. The new x-bounding list becomes \( \{ TP_{\text{xbd}} \cdot TP_{\text{xbd_vio}} \} \). Timing violation problem is thus resolved. However, “X” values will propagate into scan flip-flops and later into the MISR. To block the “X” value propagations, we need to identify all flip-flops where “X” value could be propagated into. This is performed by searching all paths starting from the “X” source net. If the final device on the path is a flip-flop, we will modify its scan enable connection so that the capture of this flip-flop is suppressed in LBIST mode, as shown in Figure 6. If the final device on the path is a memory input or a primary output, we just ignore it since the “X” value will not be propagated to the MISR through the memory or the primary output. One thing needs to be noticed is that not all paths from the “X” source to the receiving flip-flops are reported in \( \{ TP_{\text{xbd}} \cdot TP_{\text{xbd_vio}} \} \). We need to search the netlist and find them out. One drawback of this method is that the faults on the paths from “X” source to receiving flip-flops cannot be detected. They will rely on ATPG to cover them.

For timing violations in multi-cycle paths \( P_{\text{false}} \cdot P_{\text{lbist}} \), we use a similar method as in fixing x-bounding timing violations. Find the receiving flip-flops for all multi-cycle paths in \( P_{\text{false}} \cdot P_{\text{lbist}} \) and suppress their capturing during LBIST mode. The capture suppressing can avoid capturing unpredictable values due to the fact that the value propagation through these paths takes more than one clock cycle.

With the constrained LBIST insertion, we guarantee no timing violation in both functional mode and LBIST mode.

**5. Signature Mismatch Debugging**

Signature mismatch happens when the calculated signature differs from the simulated one. The cause of the signature mismatch could be timing problem, library problem, simulation environment setup problem, LBIST design problem, etc. To isolate the cause of the signature mismatch, debugging is carried out at different design hierarchies: the core module level, LBISTed core module level, and top module level as shown in Figure 2. The core module contains design core logic, all scan chains and LBIST test points. The LBISTed core module contains the core module, PRPG, MISR, and other LBIST control logic. The top module contains the LBISTed core module and the JTAG BSR module. The signature mismatch debugging starts at the core module by verifying that all scan shifting and capture operate correctly. Once the core module is verified, the verification is moved one level up in the design hierarchy to the LBISTed core module where the PRPG, MISR and other LBIST control logic are verified. Finally, the top module containing the LBISTed core module plus the JTAG control is verified. If a simulation mismatch occurs during the simulation of a particular module in the hierarchy, the problem is located in that module since the lower level module(s) have been previously verified.

**5.1. Core Module Verification**

The main task of the core module verification is to verify correct data capture and scan chain shifting. Similar to pattern verification for deterministic ATPG, the core level testbench generated by the LBIST tool applies patterns to the core and compares calculated vs. simulated responses from the core. The LBIST simulation environment makes use of mathematical models that mimic the PRPG and

![Figure 6. SE Suppression for Blocking "X" Propagation](image-url)
MISR operations in the real hardware. The pseudo-random patterns and their responses are saved in an external pattern file that is referenced by the core level testbench. Two types of testbenches are saved-serial and parallel. The parallel testbench saves simulation time by simultaneously forcing values on all scan cells in parallel, and then measuring the responses after the capture cycle from all scan cells in parallel. The parallel testbench verifies that there is no capture mismatch between the calculated and simulated values for all patterns. The serial testbench verifies correct scan chain operation by simulating a few serial patterns. The serial patterns are loaded serially through the scan chains, and the responses are unloaded serially after capture.

If a simulation mismatch occurs in the parallel testbench, the failing pattern, scan chain, and scan cell are reported. In the serial testbench, the failing pattern, shift cycle, and scan chain are reported. Once this information is available, debugging is straightforward.

5.2. LBISTed Core Module Verification

Since the core module has been fully verified, we can use testbench to mimic its operation instead using core netlist in the simulation. In other word, the simulation testbench only needs the LBIST control modules, PRPG, MISR plus a dummy core which contains only the port of the core in simulation. Since this is a core-less simulation, the simulation speed should be very fast.

The possible design location causing the MISR signature mismatch can be within the PRPG module or its connection to the input of the core, or it can be within the MISR module or its connection to the output of the core. In this simulation, the testbench will monitor the scan inputs at the dummy core to verify PRPG and its connection to the core. At the same time, the captured values in the scan cells available from the core module testbench are used to mimic the scan outputs of the dummy core. The MISR module then compresses the scan output values. By comparing the calculated vs. simulated state of the MISR, we can verify the MISR module and its connection from the core.

5.3. Top Module Simulation

The top module verification only checks if the final MISR signature can be shifted out through the JTAG TDO pin correctly. The verification complexity is not related to LBIST design.

6. A Design Case

This technique has been applied to a tele-communication design. The design contains 17.25 million logic gates, 14 clock domains and over 1,400 I/O pins. In the core, there are about 64,000 flip-flops that are all scanned. All 150 embedded memories have bypass logic implemented for scan mode and LBIST mode. Scan chains are re-configurable; 10 scan chains in scan mode and 200 scan chains in LBIST mode. The run_lbist is controlled by IEEE1149.1.

Table 1 shows the comparison between the initial LBIST insertion and the constrained LBIST insertion. In the initial LBIST insertion, 993 control points, 712 x-boundings and 1,000 observe points are selected. With 8,192 pseudo-random vectors, we achieved 96.43% test coverage. Static timing analysis shows that 26 control points and 1 x-bounding are on timing violation paths. It is unnecessary that all these 26 control points contribute to timing violations. At the time we did LBIST insertion and timing analysis, the functional timing analysis was not clean yet. Due to the fact that DFT processes are often in parallel with functional debugging and functional timing violation fixing, we chose to remove all control points that are on timing violation paths as discussed in Heuristic_1. In the constrained LBIST insertion, we removed 26 control points and fixed 1 x-bounding that caused timing violations. The coverage dropped to 95.43%. The major reason for 1.00% coverage loss is due to the large fanout from the x-source where the added x-boundning causes timing violation.

<table>
<thead>
<tr>
<th></th>
<th>#ctrl points</th>
<th>#x-bound</th>
<th>#obs points</th>
<th>#vectors</th>
<th>cov. (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial LBIST</td>
<td>993</td>
<td>712</td>
<td>1,000</td>
<td>8,192</td>
<td>96.43</td>
</tr>
<tr>
<td>constrained LBIST</td>
<td>967</td>
<td>711</td>
<td>1,000</td>
<td>8,192</td>
<td>95.43</td>
</tr>
</tbody>
</table>

Table 2 shows the comparison between LBIST and ATPG. LBIST uses 8,192 pseudo-random vectors to achieve 95.43% coverage with 1,000 observe points, 967 control points and 711 x-boundning logic inserted. The deterministic ATPG achieved 97.79% coverage with 5,924 vectors. It can achieve higher coverage than 97.79%, but the tester we used has memory limitation to store test vectors.

In LBIST mode, test clocks are running at 62.5MHz. In scan mode, test clocks are running at 25 MHz due to the tester limit on scan chain channels. To minimize scan shifting time in LBIST mode, we implemented a 64-bit PRPG and a 32-bit MISR to connect to 200 scan chains. Due to these reasons, LBIST test uses much less tester time compared to normal scan test, even with larger number of random vectors than ATPG vectors.
Table 2. LBIST and ATPG comparison

<table>
<thead>
<tr>
<th></th>
<th>#gates (k)</th>
<th>#scan chains</th>
<th>longest scan chain</th>
<th>#vectors</th>
<th>cov. (%)</th>
<th>#test cycle (k)</th>
<th>test time (msec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LBIST</td>
<td>1,263</td>
<td>200</td>
<td>321</td>
<td>8,192</td>
<td>95.43</td>
<td>2,638</td>
<td>42</td>
</tr>
<tr>
<td>ATPG</td>
<td>1,250</td>
<td>10</td>
<td>6,852</td>
<td>5,924</td>
<td>97.79</td>
<td>40,597</td>
<td>1,624</td>
</tr>
</tbody>
</table>

Removing control and observe points or changing x-bounding to fix timing violations cause fault coverage loss. Reducing pseudo-random vectors to save tester cost also eliminates achieving high fault coverage. Table 3, table 4 and table 5 illustrate the impacts on fault coverages with the changes of the number of pseudo-random vectors, the number of control points and the number of observe points, respectively. In this design, we can see that the fault coverage is not very sensitive to the increase of the number of control points and the number of observe points after we have selected 967 control points and 1,000 observe points (see table 4 and table 5). However, it is still sensitive to the number of random test vectors (see table 3). Considering the cost of the tester usage time, we can implement two run_l bist commands to control semiconductor manufacturing test and board level chip test separately. For manufacturing test, we can use 8,192 test vectors. On board level test, we can use 16,384 or more test vectors.

Table 3. Fault coverage vs. # vectors

<table>
<thead>
<tr>
<th>#ctrl points</th>
<th>#obs points</th>
<th>#vectors</th>
<th>cov. (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>967</td>
<td>1,000</td>
<td>4,096</td>
<td>94.18</td>
</tr>
<tr>
<td>967</td>
<td>1,000</td>
<td>8,192</td>
<td>95.43</td>
</tr>
<tr>
<td>967</td>
<td>1,000</td>
<td>16,384</td>
<td>96.15</td>
</tr>
</tbody>
</table>

Table 4. Fault coverage vs. # ctrl points

<table>
<thead>
<tr>
<th>#ctrl points</th>
<th>#obs points</th>
<th>#vectors</th>
<th>cov. (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>399</td>
<td>1,000</td>
<td>8,192</td>
<td>92.08</td>
</tr>
<tr>
<td>967</td>
<td>1,000</td>
<td>8,192</td>
<td>95.43</td>
</tr>
<tr>
<td>1,997</td>
<td>1,000</td>
<td>8,192</td>
<td>95.58</td>
</tr>
</tbody>
</table>

Table 5. Fault coverage vs. # obs points

<table>
<thead>
<tr>
<th>#ctrl points</th>
<th>#obs points</th>
<th>#vectors</th>
<th>cov. (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>967</td>
<td>300</td>
<td>8,192</td>
<td>94.98</td>
</tr>
<tr>
<td>967</td>
<td>1,000</td>
<td>8,192</td>
<td>95.43</td>
</tr>
<tr>
<td>967</td>
<td>2,000</td>
<td>8,192</td>
<td>95.53</td>
</tr>
</tbody>
</table>

7. Conclusion

In this paper, we have presented a logic BIST practice in our environment. Our focus is on using LBIST for board level chip tests, eliminating the usage of expensive testers and reducing the tester usage time. We set the goal to achieve 95% or above fault coverage with LBIST and the top of 95% to 99% or above with ATPG. The approach presented here minimizes the design efforts and possible schedule impact due to the introduction of LBIST. This approach is based on timing analyses to constrain the LBIST insertion. A signature mismatch debugging technique is also discussed to effectively identify design bugs. This approach has been used in a large industrial design, and the results met our setting goals.

References


THE INFORMATION HEREIN IS PROVIDED ON AN "AS IS" BASIS, WITHOUT ANY WARRANTIES OR REPRESENTATIONS, EXPRESS, IMPLIED OR STATUTORY, INCLUDING WITHOUT LIMITATION, WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Paper 36.1 1010