Effective Diagnostics through Interval Unloads in a BIST Environment

Peter Wohl
Synopsys Inc.
Williston VT 05495
wohl@synopsys.com

John A. Waicukauski
Synopsys Inc.
Tualatin OR 97062
johnwaic@synopsys.com

Sanjay Patel
Synopsys Inc.
Beaverton OR 97006
sanjayp@synopsys.com

Greg Maston
Synopsys Inc.
Denver CO 80220
gmaston@synopsys.com

ABSTRACT

Logic built-in self test (BIST) is increasingly being adopted to improve test quality and reduce test costs for rapidly growing designs. Compared to deterministic automated test pattern generation (ATPG), BIST presents inherent fault diagnostic challenges. Previous diagnostic techniques have been limited in their diagnosis resolution and/or require significant hardware overhead. This paper proposes an interval-based scan-unload method that ensures diagnosis resolution down to gate-level faults with minimal hardware overhead. Tester fail-data collection is based on a novel construct incorporated into the design-extensions of the standard test-interface language (STIL). The implementation of the proposed method is presented and analyzed.

Categories and Subject Descriptors: B.8.1 **[Performance and Reliability]:** Reliability, Testing and Fault-Tolerance.

General Terms: Algorithms, Design.

Keywords: built-in self-test (BIST), fault diagnosis.

1. INTRODUCTION

Test represents a significant portion of the total cost of digital circuits. Larger and more complex designs lead to greater increases in the cost of automated test equipment, a trend that is projected to continue and accelerate [1]. Design-for-test (DFT) methods to reduce test cost include scan ([2], [3]) and, increasingly, BIST [4], [5].

Figure 1 outlines the logic BIST architecture used in this work, based on the STUMPS architecture [6]. Values from a pseudo-random pattern generator (PRPG) are loaded into the internal scan chains of the design to be tested, and the scan-chain outputs are unloaded into a signature analyzer that performs test-response compaction. After several cycles, the state of the signature analyzer is compared to the known "signature" of the fault-free design; a mismatch indicates that at least one erroneous value was unloaded from the scan chains. The PRPG is commonly implemented as a linear-feedback shift register (LFSR) and the signature analyzer is often implemented as a modified LFSR known as a multiple-input signature register (MISR) [5], [7], [8]. To reduce test application time, most DFT schemes use a multiplicity of parallel scan chains. A combinational phase shifter (Figure 1) converts the unidimensional stream of pseudo-random values generated by the LFSR into a two-dimensional array of values to load parallel scan chains [7], [9], [10]. The values unloaded from scan chains are fed into a MISR that must have at least as many bits as there are scan chains [7]. A combinational [space-] compactor (Figure 1) is inserted between the scan chain outputs and the signature analyzer inputs, making it possible to use a smaller MISR [11].

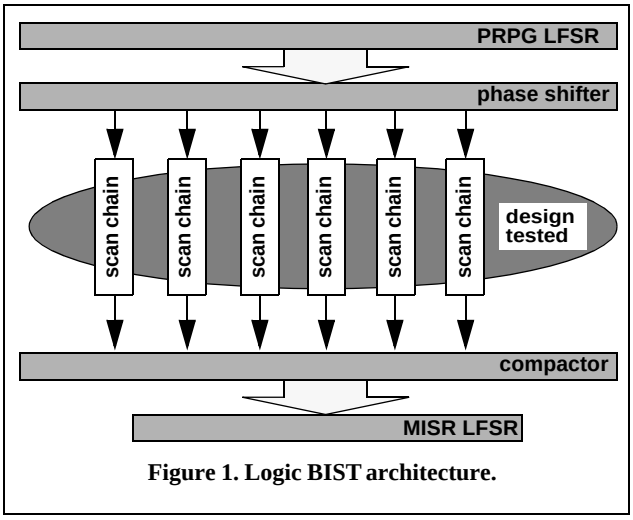

Figure 1. Logic BIST architecture.

When test patterns fail, it is desirable to quickly locate the cause of failure to one or more circuit gates so that possible manufacturing or design errors can be corrected, thereby improving yield. Failures are located through diagnostic at the logical level; failure analysis identifies physical defects. Diagnostic relies on extensive space (scan cell) and time (pattern cycle) failure information to identify logical failures [3]. Unlike ATPG patterns, diagnosing BIST pattern failures is more difficult because only the compressed final signature is available. Analysis of the failing signature can identify the failing test vectors; but, as the number of failing vectors increases, the complexity of this method also increases and its identification abilities diminish [12]. To identify an increased number of failing vectors requires additional hardware as cyclic registers [13], [14]. Another method utilizes the quotient instead of the signature, providing improved diagnosis but requiring substantial additional hardware [15]. Various scan-cell partitioning schemes have also been presented, which require additional hardware and apply the same test multiple times with different partition elements to stepwise refine the failing candidates [16], [17]. In other methods, patterns are re-run multiple times based on a binary search technique used to identify the failing patterns, but a large number of steps may be required and signatures must be computed and analyzed for each search step [18]. In another scheme, the BIST ses-

DAC 2002, June 10-14, 2002, New Orleans, Louisiana, USA.

sion is repeated several times with the MISR programmed to collect multiple signatures for different polynomials [19]. However, the maximum number of scan cells with faulty responses that can be identified depends on how many signatures are collected. Similarly, multiple signatures can be gathered by pseudo-randomly masking out different sets of scan cells each time the BIST session is repeated [20].

This paper presents a fast, very-low hardware overhead diagnostic method that identifies not just the failing scan cells, but the failing gates(s). Section 2 presents the general concept of interval-based self test. Section 3 details the method of unloading all scan data for a selected interval. Section 4 describes the tester flow and section 5 the simulator flow, each based on a single pattern file, without requiring re-generation of BIST data.

2. INTERVAL-BASED LOGIC BIST

The self-test is divided into independent intervals of a fixed number of test patterns; e.g., 256 patterns. Each interval consists of an ordered sequence of events shown in Figure 2, in standard STIL representation [21]. We assume that the BIST test and diagnostic operations are controlled by a tester; therefore, the STIL program in Figure 2 is loaded on the tester. (If the BIST test was fully independent, an on-chip BIST controller would generate the necessary control signals.)

```
// STIL macro definition:
MacroDefs {
"compare_substitute" { /*empty*/ }
"bist_load_unload" {
  ScanStructures bist;
  Loop 100 { // chains are 100 long
     Macro "compare_substitute";
     V { "bist_clk"=P; "clk"=P; }
  }}}
// STIL BIST intervals:
Pattern "pattern" {
"interval 0":
Call "load_unload" {//load PRPG & MISR
  "lfsr_si"=1110101...; }
Macro "bist_setup"; // optional
Loop 256 { // 256 patterns
  Macro "bist_load_unload";
  V { "bist_clk"=P; "clk"=P; }
}
V { "misr_comp" = L; } // MISR compare
"interval 1": /* similar */
"interval 2": /* similar */
....
}
```

Figure 2. STIL description of BIST interval containing 256 patterns; scan chains are 100 cells long.

At the beginning of each interval, the “load_unload” procedure (not shown in Figure 2) is called to initialize, through scan, the PRPG and MISR. Initializing the PRPG and MISR at the beginning of each interval ensures that the intervals are independent of each other. Also, the PRPG is seeded with an initial value targeted to detect certain hard-to-test faults [22]. The MISR is initialized to a value so that the fault-free signature at the end of the interval is 0, simplifying signature compare [7]. Next, an optional “bist_setup” macro is called to set up scanning of the internal scan chains. In boundary-scan based BIST, the “bist_setup” macro sets the JTAG controller to the appropriate state. The 256 test patterns in the interval are run by the Loop statement; each iteration calls the “bist_load_unload” macro (shown above the interval in Figure 2) and then pulses, in a single vector (V), the “bist_clk” and the system “clk”. The “bist_clk” clocks all BIST hardware: the PRPG, MISR and BIST controller; the system “clk” is used both for shift and system capture in Figure 2. In general, the shift clock (used in the macro “bist_load_unload”) may be different from the capture clock; multiple capture clocks could also be used. The “bist_load_unload” macro (Figure 2) pulses “clk” to shift values into the scan chains defined in “ScanStructures bist” while, at the same time, pulsing “bist_clk” to advance the PRPG and compress unloaded values into the MISR. The macro “compare_substitute” called by “bist_load_unload” can be defined as an empty macro for normal BIST application and has no effect; a non-empty “compare_substitute” will be used during diagnostic (section 3).

During the first pattern, scan cells have unpredictable values that must be prevented from entering the MISR; this is achieved by ANDing off each compactor input (Figure 3) during the first application of “bist_load_unload” [11]. The gating signals c0 and c1 are provided on-chip by a small counter -- part of the BIST controller. During the first unload of each interval c0=c1=0; at all other times c0=c1=1; in diagnosis mode (section 3) c0 and c1 have different values, thus a single gating signal could not be used. At the end of each interval, a logic OR gate compares the MISR content to 0; the interval test has failed if the MISR is non-0.

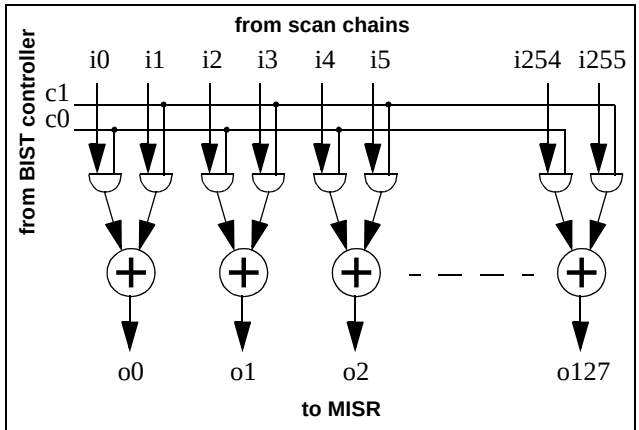

Figure 3. Compactor core showing gating of inputs i0..i255 controlled by signals c0, c1.

Figure 4 shows the timing of the shift and capture operations in non-diagnostic mode. While the shift clocks can be applied at a lower speed (to reduce power consumption), the sequence marked “AC” (from the last shift clock to the capture clock) must be applied fast enough to ensure effective launch-on-last-shift transition-fault test. The timing of the clocks is also stored in the STIL file but not shown in Figure 2. The internal shift clock (“clk”) and the BIST clock (“bist_clk”) are both pulsed every cycle. Although the bist_clk pulse at the time of the capture cycle (the gray clock pulse) is logically not needed, it simplifies the control circuitry and ensures uniformity of clock-induced noise.

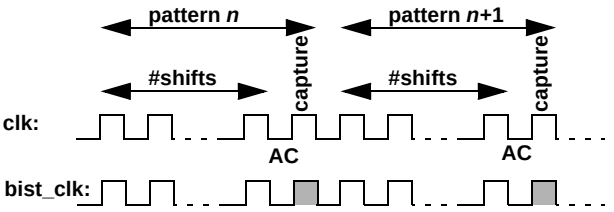

Figure 4. Clocking in non-diagnostic mode.

3. UNLOADING INTERVAL DATA

Unlike previously proposed BIST test windows ([23]), intervals are independent of each other and can be reordered or run individually in much the same way as individual deterministic ATPG patterns. Each interval is fully defined by the PRPG and MISR initial values and the clock(s) pulsed. Test windows define intermediate MISR fault-free signatures used to quickly determine the failing patterns (window) for diagnostic. Intervals achieve the same fail patterns resolution (256 patterns) but do not require the additional overhead of salvaging future windows by adjusting their signatures to fit past observed errors [23].

To make failure diagnosis of the failing gate possible, the exact position of every erroneous bit unloaded from the scan chains must be known. Diagnostic time is less critical than test time therefore, in a logic BIST environment, it is desirable and practical to unload all scan values without using compaction and signature analysis [24]. To minimize hardware overhead, the compactor (Figure 3) supports a "pass-through" mode for diagnostic and the scan values are unloaded from the MISR (Figure 5), eliminating the need for added scan-chain MUXing to bypass the compactor and MISR. The AND gates used in the compactor to gate off scan data during the first unload are re-used during diagnostic to gate off selected scan chains so that data from the remaining chains are passed through unmodified. For an *m*-bit MISR, one bit from each of the first *m* scan chains is first passed unaltered through the compactor, captured in the *m* cells of the MISR, and then scanned out of the MISR. This process is repeated to obtain values from all scan cells of all scan chains [11]. Once all scan unload data has been collected, diagnostic is performed using fault simulation similar to diagnostic of deterministic ATPG, achieving the same resolution down to the failing gates(s) [3].

The compactor (Figure 3) and the MISR (Figure 5) can operate in several modes, controlled by the BIST controller: scan, signature analyzer, or diagnosis mode. The diagnosis operation for a design with 256 scan chains, a 2-to-1 compactor and a 128-bit MISR requires the following steps:

(1) Initialize MISR to 0 through scan: diag=1, c0=c1=0, scen=1; all values shifted into the MISR are 0.

(2) Capture in MISR outputs of group 0 scan chains (chains connected to i0, i2, ..., i254): diag=1, c0=1, c1=0, scen=0; values are captured into MISR cells through XOR gates that have all inputs 0 except the "o" inputs from the compactor.

(3) Scan out MISR, get 1 bit of every group 0 chain through so, re-initialize MISR to 0: diag=1, c0=c1=0, scen=1.

(4) Capture in MISR outputs of group 1 scan chains (chains

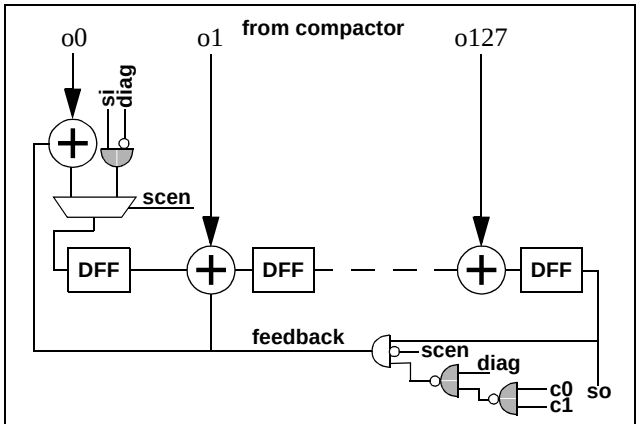

Figure 5. MISR architecture; the gates added for diagnosis support are gray.

connected to i1, i3, ..., i255): diag=1, c0=0, c1=1, scen=0; similar to step 2.

(5) Scan out MISR, get 1 bit of every group 1 chain through so, re-initialize the MISR to 0: diag=1, c0=c1=0, scen=1.

(6) If more bits in scan chains, shift scan chains and go to 2.

Note that the hardware added for diagnosis is only 3 gates (Figure 5), 2 signals (diag, one extra "c" gating signal) and a counter in the BIST controller to control c0, c1. The signal diag enables diagnosis operation in the MISR and in the BIST controller; diag is constant 1 during diagnosis and 0 during normal operation. A primary input can be used to directly control diag, or it can be generated internally by the BIST controller when its state machine enters diagnosis mode as controlled by a limited-pin interface, such as boundary scan (JTAG).

Failing intervals are re-run on the tester using diagnosis mode of pattern operation to collect diagnosis data. The pattern block of the STIL file, shown by example under the "STIL BIST intervals" comment in Figure 2 is unchanged, but a different set of procedures and macros are referenced and linked with STIL's PatternBurst construct (Figure 6) [21]. The macro "compare_substitute" is no

```
PatternBurst "normal" {
  // by default uses unnamed MacroDefs
  // which includes empty
  // "compare_substitute"
  Patlist { "pattern" // see Figure 2
}}
PatternBurst "diagnosis" {
  MacroDefs "diagnosis";// see Figure 7
  Start "interval 2"; // first failing
  Stop "interval 4; // last failing
  Patlist { "pattern" // see Figure 2
}}
```

Figure 6. STIL selection of normal vs. diagnosis mode.

longer empty, but defined as in Figure 7. Every shift in "bist_load_unload" is preceded by a call to "compare_substitute" (Figure 2). In the first phase (Figure 7), one value from each of the first 128 scan chains is passed through the compactor and captured into the MISR ("bist_clk" is pulsed). Next, the Loop scans out the

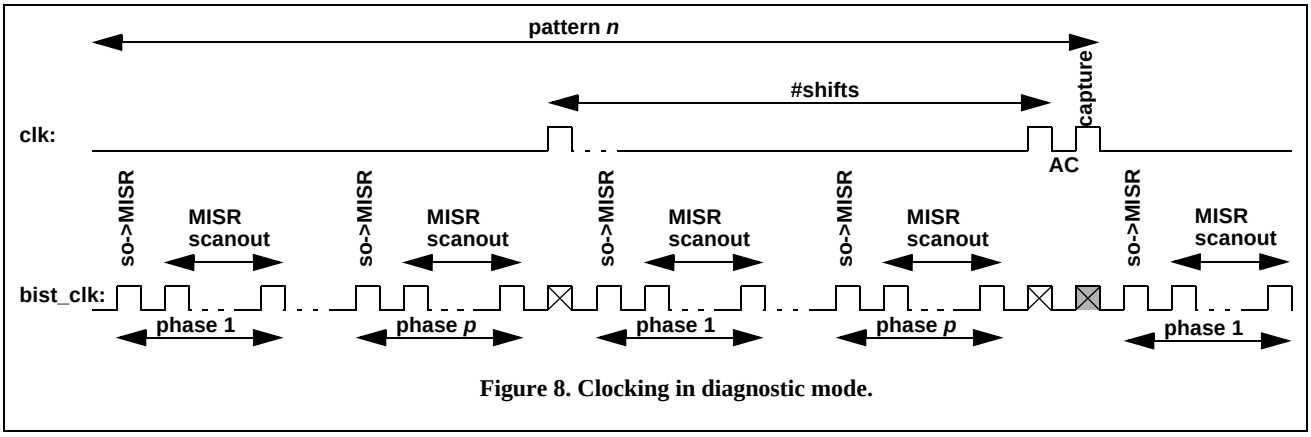

Figure 8. Clocking in diagnostic mode.

```
MacroDefs "diagnosis" {
"compare_substitute" {
  C { "clk"=0; } // Condition clk OFF
  // 1st phase: first 128 scan chains
  V { "bist_clk"=P; } // so->MISR
  Loop 128 { // MISR is 128 bits
     V { "bist_clk"=P; "lfsr_so"=S; }
  }
  // 2nd phase: next 128 scan chains
  V { "bist_clk"=P; } // so->MISR
  Loop 128 {
     V { "bist_clk"=P; "lfsr_so"=S; }
  }
  // more phases if needed
}}
```

Figure 7. STIL definition of diagnostic macro for a MISR of *m* = 128 bits.

MISR. The "S" character on the scanout pin "lfsr_so" references a Waveform that contains a "CompareSubstitute" event for the output strobe operation, but the value (compare High or compare Low) is not known; instead, the tester substitutes the High or Low value measured from the device-under-test, thus accumulating diagnostic data each time an "S" is encountered [25]. This process is repeated in the second phase (and subsequent phases if needed; e.g., 4 phases for 512 scan chains) to obtain values from all scan cells of all scan chains.

In this diagnostic operation, the MISR is re-used as a shift register to shift out the content of all scan cells (Figure 5). Figure 8 shows clocking in diagnostic mode, focusing on the same events as shown in Figure 4, clocking in normal mode. In diagnostic mode (Figure 8), each shift clock of every pattern is preceded by a complete unload of scan values, which requires *p* MISR unload phases. Unlike Figure 4, the BIST clock ("bist_clk") is pulsed in Figure 8 while the internal shift clock ("clk") is off. Each MISR unload phase starts by copying scanout values to the MISR (so->MISR), followed by a complete MISR scanout during which the PRPG and the internal scan chains are not pulsed. To preserve AC-testing capability, there is no MISR unload between the last shift and the capture; therefore, the data collected in diagnostic mode does not include the last-shift data, which is part of the data compressed in non-diagnostic mode. As with non-diagnostic timing (Figure 4), the PRPG is advanced by pulsing "bist_clk" every time "clk" is pulsed; for the MISR the "bist_clk" pulses (marked with X in Figure 8) during "clk" pulses are unnecessary because diagnostic data collection starts after the "clk" pulses and completes before next "clk" pulse.

4. TESTER FLOW

In a typical deterministic ATPG test flow, ATPG generates a large deterministic pattern file which is loaded and run on the tester; failing patterns can be selectively re-run in diagnostic mode. It is important that the very same pattern file is re-used for the diagnostic run so that the ATPG tool is not required to produce another large file. The diagnostic run typically requires additional tester settings to only run selected patterns and to log fail data. The fail data file produced by the tester is then returned (possibly after format translation) to the ATPG tool that performs diagnosis by fault simulation [3].

The same flow is maintained with BIST patterns by using the described diagnostic data collection method. BIST ATPG generates a very small BIST pattern file (Figure 2) that contains only the interval data (PRPG and MISR initial values and clocks pulsed in each interval). The BIST pattern file is loaded and run on the tester; failing intervals can be selectively re-run in diagnostic mode. As with deterministic ATPG, the very same pattern file is re-used for the diagnostic run so that additional ATPG runs or other files are not required. The diagnostic run requires additional tester settings to only run selected intervals (using STIL's "start" and "stop" operations in PatternBurst diagnosis) and to select the diagnostic procedures and macros from the STIL file (Figure 6). The diagnostic data file collected by the tester is then returned (possibly after format translation) to the ATPG tool that performs diagnostic by fault simulation. The diagnostic data file is much smaller than a deterministic patterns file because it contains only unload (no load) data of only selected intervals, and unload values are encoded with only 1 bit (2 states -- high and low); unlike deterministic patterns unload, 2 bits are not needed for BIST unloads because X values never propagate to the MISR.

For example, the diagnostic data file for a 10,000 scan cell design requires only 320 KB. For increased diagnosis resolution of multiple faults, multiple diagnostic data files can be collected for multiple intervals and returned to the ATPG tool for diagnostic. The diagnostic flow is enabled by usage of the CompareSubstitute construct in STIL (Figure 7) ([25]) and is not achievable in languages such as WGL ([26]) that lack such construct.

To support deterministic ATPG diagnostic, some testers implement the ability to unwind the hardware pipeline, starting with the first miscompare, to find the cycle corresponding to each failure. Failures occurring within a limited window from the first failure can be logged in one pass; to log additional fail data, the first miscompares are masked out and the test is repeated. If a separate BIST diagnostic pattern file is generated (in a WGL-based flow), the same fail-data logging mechanism can be used for BIST diagnostic. Supporting CompareSubstitute (for a STIL-based flow) requires the tester to efficiently log a larger volume of data, but does not require selectively masking out previously logged bits. Some testers implement this, or a similar mechanism for known-good-die response logging.

5. SIMULATOR FLOW

Before testing a device, it is common that at least a subset of the patterns are first verified though simulation (commonly Verilog [27] or VHDL [28]) for logical and timing accuracy. Verification is particularly important for BIST because the BIST structures (PRPG, phase-shifter, compactor, MISR, BIST controller) are simulated only functionally by the ATPG tool (for increased performance), whereas Verilog or VHDL simulation verifies the structural (gate-level) design. For verification, patterns are written out by the ATPG tool in a testbench format that provides stimuli to a simulator and compares the simulator outputs with the expected ATPG outputs. Both deterministic ATPG and BIST patterns can be verified through this scheme. To verify the diagnostic capability of deterministic ATPG, a fault can be injected in the simulated design; and, in turn, the simulator mirroring a tester, could produce a fail file that the ATPG tool then diagnoses.

Verification of the BIST diagnosis flow is even more important because the ATPG tool simulates all BIST structures functionally, including the mechanisms of unloading scan-chain values through the MISR. For this verification, the testbench is written so that a simulator compile-time definition controls whether the simulation is done in normal or diagnostic mode (using the corresponding STIL procedures and macros translated into Verilog or VHDL); this allows the same BIST pattern file to be used for either normal or diagnostic verification. In diagnostic mode, the simulator mirrors a tester and collects the diagnostic data file by scanning out the MISR. The file is then diagnosed by the ATPG tool in a special fail-only mode in which only the failing scan cell (not the internal gate) is identified to aid in debugging simulator differences. Diagnosing the internal gate is unnecessary because a simulator mismatch in simulating internal gates would have shown up as a MISR miscompare when simulating non-diagnostic mode BIST patterns; diagnostic-mode simulator mismatches can be caused only by functional and structural representation differences of the BIST structures involved in the diagnostic data collection. Evidently, if the simulator run matches the ATPG expected data, then the fail-only diagnostic reports no miscompares.

6. RESULTS

The simulator flow presented in section 5 was used to test the diagnosis method presented on designs ranging from 200K to 1M gates. A typical session, obtained for a 1M gate design, is detailed below.

(1) Generated BIST patterns in 32-pattern intervals.

(2) Saved patterns as Verilog testbench.

(3) Ran Verilog simulation and confirmed that MISR signatures match the expected values.

(4) Edited the Verilog netlist to inject a stuck-at fault.

(5) Re-ran Verilog simulation and confirmed that MISR signatures now miscompare (intervals 0 and 14 miscompared).

(6) Re-ran interval 0 Verilog simulation in diagnosis mode to generate the interval data (section 3).

(7) Read the file produced by the simulator into the ATPG tool and performed fault diagnosis.

The output of the diagnosis run is shown in Figure 9. First, the interval is simulated and the simulated values are compared to the actual values to identify the failing scan cells. Then, fault simulation is performed on these patterns to determine faults that best explain the failing scan cells. The summary shows the number of defects found (1) and the total CPU time (10 seconds on a 1 GHz Pentium®). Next, the fault candidates are listed: there are two faults marked with “DS” (detected by simulation), followed by their equivalent faults marked with “--”. The fault injected at step (4) above was a stuck-at 0 (sa0) on output pin S of XOR 179345: the full pin-pathname of this fault is listed on the last line in Figure 9, preceded by two equivalent faults. The first 2 lines in Figure 9 list another defect candidate: the equivalent faults sa0 on output CO of AND 179152 and input B of buffer 179221 could cause the same fault signature. Diagnosis successfully identified the fault simulated. If this were real silicon, failure analysis would follow to analyze the fault candidates physical locations and identify the defect. In many cases, a single fault candidate is identified making failure analysis very simple; the two fault candidates in this case would still be easy to analyze because of their proximity.

7. CONCLUSIONS

The BIST diagnostic method presented allows identification of the failing gates -- not just the failing scan cells -- offering the same diagnosis resolution as deterministic ATPG test. Only minimal hardware is added for diagnostic: a few gates and internal signals controlled by the BIST controller. Determining if BIST will run in normal or diagnosis mode can be done using a “diag” primary input or, in a reduced-pin implementation, by a “start diagnosis” instruction to the BIST controller. Any number of fails can be handled and binary searches or other time-consuming methods are not required. Instead, the BIST test is composed of small, independent intervals. Only failing intervals need to be re-run in diagnostic mode to collect all scan data of the interval. Both the tester and the simulator flows use the same BIST pattern file for normal and diagnostic mode. The device-under-test timing in diagnostic mode is the same as in normal mode, preserving AC-test capabilities. The streamlined tester flow relies on a tester’s ability to process the new CompareSubstitute event (added to the standard STIL language). Traditional (WGL-based) tester flows are supported with a separate diagnostic pattern file.

ACKNOWLEDGMENT

We are very grateful to Bill Lloyd and Martin Bell for careful reviews and insightful comments.

```
Failures for interval 0: #patterns=32, clocks= bist_clk clk
-------------------------------------------------
interval  pat#  chain     pos#  gate_id  simv/expv
--------  ----  --------  ----  -------  ---------
0         9     c359      34    389863     1 / 0
0         9     c359      35    389864     0 / 1
0         9     c359      36    389865     0 / 1
Failure summary: #failing_pats=1, #failures=3, #passes=564569
-------------------------------------------------
Diagnosis summary for failure file pat.diag
#failing_patterns=1, #defects=1, #unexplained_fails=0, CPU=10.20
Warning: Internal pattern set is now deleted. (M133)
-------------------------------------------------------------------------------
Fault candidates for defect 1: #failing_patterns_explained=1
-------------------------------------------------------------------------------
val  code  pin_pathname  (module_name)
---  ----  ---------------------------------------------
sa0   DS   core_i/compare_nf/b03_eq_port_fifo_read_60/add_136/U7/B
sa0   --   core_i/compare_nf/b03_eq_port_fifo_read_60/add_136/U6/CO
sa0   DS   core_i/compare_nf/b03_eq_port_fifo_read_60/U26/B2
sa0   --   core_i/compare_nf/b03_eq_port_fifo_read_60/U26/B1
sa0   --   core_i/compare_nf/b03_eq_port_fifo_read_60/add_136/U7/S
```

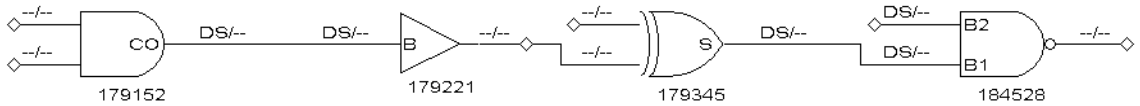

Figure 9. Diagnosis output: text (top) and schematic view of the fault candidates.

REFERENCES

[1] Semiconductor Industry Association (SIA), *International Technology Roadmap for Semiconductors (ITRS)*, 1999.

[2] M. Abramovici, M.A. Breuer, A.D. Friedman, *Digital Systems Testing and Testable Design*, IEEE Press, 1990.

[3] E.B. Eichelberger, E. Lindbloom, J.A. Waicukauski, T.W. Williams, *Structured Logic Testing*, Prentice-Hall, 1991.

[4] H.J. Nadig, "Testing a Microprocessor Product Using a Signature Analysis", *International Test Conference* 1978, pp.159-169.

[5] V.D. Agrawal, C.R. Kime, K.K. Saluja, "A Tutorial on Built-In Self-Test, Part 1: Principles", *IEEE Design & Test* 1993, Vol. 10, No.1, pp. 73-82.

[6] P.H. Bardell, W.H. McAnney, "Self-Testing of Multichip Logic Modules", *International Test Conference* 1982, pp.200-204.

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

[8] V.D. Agrawal, C.R. Kime, K.K. Saluja, "A Tutorial on Built-In Self-Test, Part 2: Applications", *IEEE Design & Test* 1993, Vol. 10, No.2, pp. 69-77.

[9] J. Rajski, N. Tamarapalli, J. Tyszer, "Automated Synthesis of Large Phase Shifters for Built-In Self-Test", *International Test Conference* 1998, pp. 1047-1056.

[10] J. Rajski, N. Tamarapalli, J. Tyszer, "Automated Synthesis of Phase Shifters for Built-In Self-Test Applications", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 2000, Vol. 19 No. 10, pp. 1175-1188.

[11] P. Wohl, J.A. Waicukauski, T.W. Williams, "Design of Compactors for Signature-Analyzers in Built-In Self Test", *International Test Conference* 2001, pp.54-63.

[12] W.H. McAnney, J. Savir, "There is Information in Faulty Signatures", *International Test Conference* 1987, pp. 630-636.

[13] J. Savir, W.H. McAnney, "Identification of Failing Tests with Cycling Registers", *International Test Conference* 1988, pp. 322-328.

[14] J. Ghosh-Dastidar, D. Das, N.A. Touba, "Fault Diagnosis in Scan-Based BIST Using Both Time and Space Information", *International Test Conference* 1999, pp. 95-102.

[15] R.C. Aitken, V.K. Agarwal, "A Diagnosis Method Using Pseudo-Random Vectors without Intermediate Signatures", *International Conference on Computer-Aided Design* 1989, pp. 574-580.

[16] I. Bayraktaroglu, A. Orailoglu, "Deterministic Partitioning Techniques for Fault Diagnosis in Scan-Based BIST", *International Test Conference* 2000, pp. 273-282.

[17] J. Ghosh-Dastidar, N.A. Touba, "A Rapid and Scalable Diagnosis Scheme for BIST Environments with a Large Number of Scan Chains", *VLSI Test Symposium* 2000, pp. 79-85.

[18] P. Song, F. Motika, D. Knebel, R. Rizzolo, M. Kusko, J. Lee, M. McManus, "Diagnostic Techniques for the IBM S/390 600 MHz G5 Microprocessor", *International Test Conference* 1999, pp. 1073-1082.

[19] Y. Wu, S. Adham, "BIST Fault Diagnosis in Scan-Based VLSI Environments", *International Test Conference* 1996, pp. 48-57.

[20] J. Rajski, J. Tyszer, "Fault Diagnosis in Scan-Based BIST", *International Test Conference* 1997, pp. 894-902.

[21] IEEE Std 1450-1999, *IEEE Standard Interface Test Language (STIL) for Digital Test Vectors* 1999.

[22] B. Könemann, "LFSR-Coded Test Patterns for Scan Designs", *European Test Conference*, Munich, 1991.

[23] J. Savir, "Salvaging Test Windows in BIST Diagnostics", *VLSI Test Symposium* 1997, pp. 416-425.

[24] J.A. Waicukauski, V.P. Gupta, S.T. Patel, "Diagnosis of BIST Failures by PPSFP Simulation", *International Test Conference* 1987, pp. 480-484.

[25] IEEE P1450.1, *Extensions to STIL for Semiconductor Design Environments*, http://grouper.ieee.org/groups/1450/dot1/index.html

[26] Summit (Fluence) Inc., *Waveform Generation Language (WGL) TDS Release 9.1*.

[27] IEEE Standards Department, "Verilog Hardware Description Language", IEEE-1364, 1994.

[28] IEEE Standards Department, "IEEE Standard VHDL Language Reference Manual", IEEE-1076-1987, IEEE, NY, 1988.