Journal of Harbin Engineering University
ISSN: 1006-7043

Vol 45 No. 9
September 2024

Verification and Testing of Flash Controller through Hierarchical DFT

[1]Tanushree Baramoji Aralikatti, [2]Dr. Srividya P, [3]Vaibhav Madali
Department of Electronics and Communication Engineering
1Student, RV College of Engineering, Bangalore
2Associate Professor, RV College of Engineering, Bangalore
3Principal Engineer (DFT), Microchip Technology Pvt. Ltd., Bangalore

Abstract
Many SOCs today are too large for a traditional flat or partition-based approach to DFT with manual steps. Breaking designs into smaller pieces makes physical implementation more manageable for designers as well as for EDA tools. Hierarchical DFT divides the design into smaller pieces, creates test structures and patterns at the core level, then retargets the core patterns to the chip level. For the purpose of ATPG, a well-proven D-algorithm is made use of. Vectors are created within the chip's DFT framework using Cadence's Modus tool at the 16nm technology node. These vectors cover various test modes, encompassing both stuck-at and transition faults. Each DFT region aims for over 99% stuck-at fault coverage and more than 85% transition fault coverage. If the target coverage (>=99% stuck-at fault coverage and >=85% transition fault coverage) is not met, fault coverage accounting method is utilized which considers untested faults from ATPG and generates test vectors making them testable resulting in coverage improvement. Before application on actual silicon in the ATE, these patterns undergo validation via simulation using Cadence's IES tool, at both block and device levels.
Keywords: Hierarchical DFT, MODUS tool, ATPG, test coverage, stuck-at and transition faults, Fault coverage accounting, TPIs

1. Introduction

Before the era of million-gate ASICs, testability was often considered a back-end issue. Large designs were “thrown over the wall” to a dedicated Production Engineering group. Frequently, this “over the wall” approach resulted in excessive rework and wasted time. Today’s design complexities and time-to-volume pressures require a proactive approach. Testability must be designed into the chip architecture right from the RTL coding phase. This proactive ASIC design strategy has come to be called design for testability (DFT). The motivation for implementation of a hierarchical DFT is to create a handshake with front-end and physical design engineers. In hierarchical DFT approach, the chip can be divided into multiple smaller pieces or cores, which can be efficiently accessed and processed. Large design issues like tool memory, large ATPG run time, and pin limitations can be resolved by Hierarchical DFT techniques. It results in a reduction in the pin count, memory and test run time due to patterns being generated at the core level. It is also possible to run cores in parallel. With all of the aforementioned benefits, this hierarchical method seems quite promising.

The main aim of this paper is to generate test patterns for the design using MODUS tool of Cadence and hence validation of the same with the help of Cadence IES simulator that simulates the given design with the generated test patterns. This experimental run is based on a 16nm technology design. The targeted test coverage is 85% for transition faults and 99% for stuck-at faults. Using hierarchical DFT approach, there exists various advantages such as test vectors can be reduced using the compression technique, test time can be reduced and hence reduction in test cost. Further, to overcome coverage loss, fault coverage accounting method is adopted wherein individual wrapped cores are tested in isolation from each other resulting in coverage improvement. Also, Test Points are inserted to improve a design’s testability and improve its test coverage by adding a certain number of control and observe points in the design.

Incremental test generation for evaluating the effects of the modifications on the testability of

the design is provided in [1]. This brief argues that without completing test generation, a quick (incomplete) method for detecting undetected faults is effective in determining that the testability will degrade due to a local design alteration. A technique that executes consecutive local design adjustments is used in two tests. A modification is only approved if it eliminates existing alterations and doesn't cause any new, undetected faults. In [2], various design for testability (DFT) methodologies for total-ionizing-dose (TID) testing of flash-based field-programmable gate arrays (FPGAs) are thoroughly compared. The methodologies for enhanced scan DFT, clocked scan, and muxed D scan are all compared. Circuits from the ISCAS'89 benchmarks are used for the comparison. Comparison points include how FPGA resources are used, how hard it is to set up a design, how much latency DFT logic adds, and how reliable tested paths are in each method. Using Microsemi ProASIC3 FPGAs and the Cobalt 60 facility, the authors have experimentally validated the outcomes of each technique. When employing worst-case test vectors (WCTVs) in total-dose testing of FPGA devices, the experimental results demonstrate a near totaldose failure level for all methodologies. Although functional test sequences often produce a lower gate-level fault coverage than scan-based tests, they can detect defects that are missed by scan-based testing. Functional test sequences created using design-for-testability (DFT) techniques stray arbitrarily from functional operation conditions throughout the design. The concept of invisibleDFT is presented in [3] as a DFT strategy for functional test sequences, in which specific logic blocks are the only ones affected by activating the DFT logic. In this paper, an invisible-scan method is developed. The process outlined in this work considers a single logic block and involves inserting scan shift cycles into a functional test sequence while preserving the logic block's principal input and output sequences. Other logic blocks cannot see the DFT logic activated because of this. For this purpose, only a restricted set of primary output vectors can be rectified using the approach. The presentation of experimental results demonstrates how invisible scan might lead to an increase in fault coverage. The implementation of the Hierarchical DFT approach—which aims to increase test coverage at the block level—is presented in [4]. Depending on the functionality required, the SOC is split up into many layouts and DFT areas. For ATPG purposes, the tried-and-true Dalgorithm method is taken into consideration. The Modus tool from Cadence is used to generate the vectors at the DFT level inside the chip using 16nm technology. Patterns are intended for several test modes, including stuck-at and transition fault coverage, for the DFT region of the SOC. The goal is to achieve Stuck-at and transition fault coverage of more than 99% and 85%, respectively, in every single DFT area.

2. Objectives

This paper explores a hierarchical Design for Testability (DFT) approach aimed at enhancing the testing process for flash controllers. By decomposing the testing strategy into multiple levels—ranging from individual functional blocks to the complete system—the hierarchical DFT approach enables more comprehensive and efficient validation. Key benefits include improved test coverage, streamlined fault isolation, and reduced testing time and costs. The approach incorporates techniques such as scan chains, built-in self-test (BIST), and structured test points, tailored to address the unique requirements of flash memory management. Results demonstrate that hierarchical DFT significantly enhances the reliability and performance of flash controllers, making it a valuable strategy for modern semiconductor testing and verification.

The objectives of the project are to generate test patterns for the design using MODUS tool of Cadence, to validate the generated test patterns using IES from Cadence, to reduce test time and test cost and to improve test coverage.

3. Methods

The generation of test patterns for the given design happens as per the MODUS process flowchart as shown in figure 1. There are five major steps in this flowchart as explained below:

• Build Model: Generates the logic model that the MODUS uses by reading in Verilog netlists, technology libraries, memory, and analog models.

• Build Test Mode(s): Constructs particular test structure configurations (scan chain, test pins,

compression structures) needed for pattern generation or design diagnostics.

• Verify Test Structures: Confirms that the design's scan chains, and other test structures adhere to the rules meant to guarantee testability. Severe warnings point to structures that could lead to the generation of incorrect vectors. Additional cautions point to possible lesser test coverage achievable structures. Before starting ATPG, resolve all serious warnings. It assists in scanning a design database for possible issues such as contentions, timing, and scan chain issues.

• Build Fault Model: It creates the fault model for ATPG. It is used for ATPG/diagnostics simulation. It helps in building individual fault model for static and dynamic faults for the design and prints statistics for various categories of faults it has created.

• Test Pattern Generation: Develops test patterns to validate and test faults.

A. Pattern Generation – ATPG

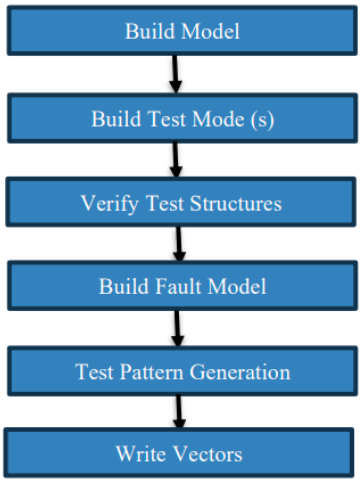


Figure 1: MODUS process flowchart

The Cadence Modus tool is mostly utilized for ATPG in 16nm technology. The act of choosing which stimuli to apply to a design in order to illustrate its proper structure is known as test pattern creation. The application of these test vectors is used to demonstrate that there are no manufacturing-induced faults in the design. To produce test patterns, a script file supporting the Modus tool must be written. In order to ensure that there are no serious design violations, the flow in figure 1 illustrates the different models that must be created while generating patterns. These models include the netlist and the 16nm technology libraries, as well as the definition of test configurations, test structures, and the reporting of analyzed fault models and their corresponding tested status.

• Write Vectors: Lays out the vectors for simulation or to satisfy IC manufacturers' criteria for the production interface. It logs errors to compile a list of each defect model and its state at the time of testing. The following formats for writing vectors are supported by Modus:

▪ IEEE established the Standard Test Interface Language (STIL) format.

▪ WGL stands for Waveform Generation Language, a Fluence Technology, Inc. format.

▪ Test Description Language (TDL) is a Texas Instruments Inc. proprietary format.

▪ NC-Simulation uses the Verilog format from Cadence Design Systems, Inc.

From the list of faults, the test generator chooses

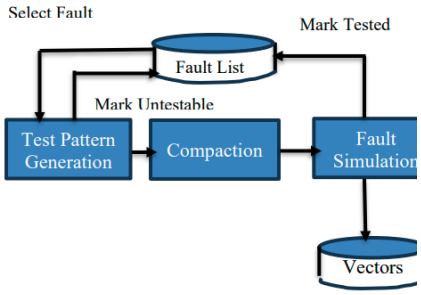


Figure 2: ATPG Process Overview

one untested fault and attempts to create a test for it alone. As seen in Figure 2, the test generator either develops a test for the defect or marks it as untestable based on the input limitations of the testmode and optional inputs. Another fault is chosen while keeping the pattern for a testable defect. A single test has all patterns that have the same clocks and don't have any conflicting stimulus on PIs, PPIs, or flops/latches.

Upon determining the ideal quantity of tests to run concurrently, each test is completed to ensure that all PIs and scannable flops/latches are assigned a value. The command line's parameter settings determine how the fill is carried out. The fault simulator receives the finished set of tests after that.

The tests are simulated by the fault simulator. A test is eliminated if it exhibits any issues (cause X,

cause disagreement, etc.). The fault list indicates checked for the defects found by the remaining tests. If the values of the good and defective machines are opposite, then a fault is said to have been identified. If the fault machine value is X (don't care), the problem is deemed to be "possibly detected."

Test generation is the first step in the process, which is repeated after the fault simulation is finished. This cycle keeps going until either all defects for which patterns may be generated have been found, or until a certain stopping point (such the maximum amount of time, patterns, or coverage) has been met.

Fault Coverage accounting and TPIs: Once the ATPG run is completed, those faults which remained untested are targeted using Fault Coverage accounting method and inserting test points (Observe points and Control points) at appropriate places within the design to further reach the coverage target (99% stuck-at fault coverage and 85% transition fault coverage). Fault Coverage accounting method helps to determine those pins which were untested. TPIs also enhances the testability of the design by making those pins/nodes which could not be tested during ATPG process as PIs or POs as appropriate.

4. Results

The expert system technique has numerous drawbacks, most of which are related to the implementation support tools. The hierarchical DFT method is by no means efficient or optimal in and of itself. Additional challenges might be observed by closely observing the updates and optimization of the fundamentals, but prior research and the hierarchical approach to design for testability (DFT) verification appear fresh and encouraging, and they may offer VLSI engineers and designers a workable solution.

Test Coverage

The ATPG are written out using the design netlist of the specified ISP block in a communication chip utilizing the previously stated DFT technique. Because these vectors are formed compactly, there are fewer vectors overall. The run-time has been shortened by the compression logic employed during pattern creation.

Two top-up transition test modes are tried for the ISP block. As seen in Figure 3, the first test mode appends its coverage % to the second test modes. The vector count, however, won't during the ATPG procedure. Figure 3 above illustrates that the 80.34% Transition coverage was attained.

The ISP block has three distinct DC testmodes. As was previously said, the ultimate DC coverage of 99.01% was reached after coverage was added at

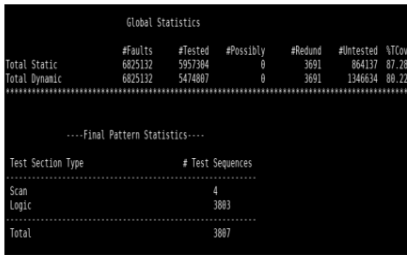


Figure 3: AC testmode coverage for ISP block

each testmode. The coverage details are

Figure 4: DC testmode coverage for ISP block

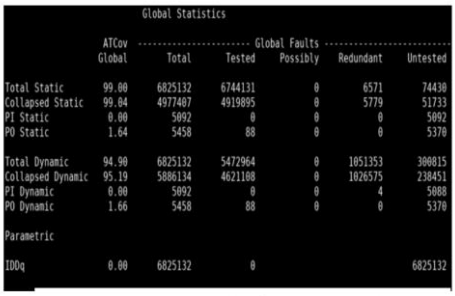


Figure 5: Coverage details after ac_accounting

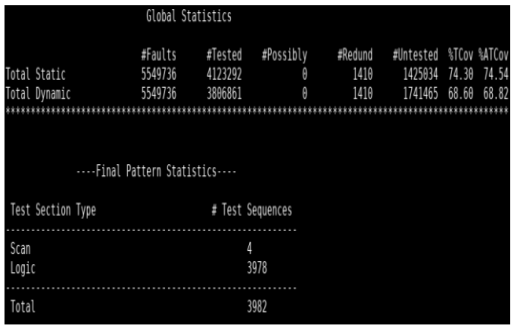
presented in Figure 4 format.

As can be seen in Figure 5, the AC coverage of 94.90% was achieved by taking ac_accounting into account.

For the RAM dominated block, two top-up transition testmodes are attempted. The first test mode adds its coverage % to the second test modes, as seen in Figure 6. During the ATPG process, however, the vector count won't

increase. As seen in Figure 6 below, the 68.82% Transition coverage was reached.

Figure 7 illustrates DC testmode coverage for RAM dominated block with a coverage of 95.26



Global Statistics

	#Faults	#Tested	#Possibly	#Redund	#Untested	%TCov	%ATCov
Total Static	5549736	4123292	0	1410	1425834	74.30	74.54
Total Dynamic	5549736	3806861	0	1410	1741465	68.60	68.82

****

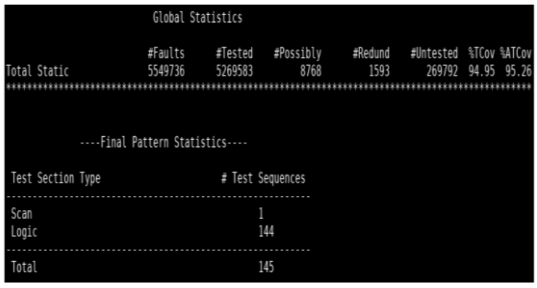
----Final Pattern Statistics----

Test Section Type	# Test Sequences
Scan	4
Logic	3978
Total	3982

Figure 6: AC testmode coverage for RAM dominated block

%.

Figure 8 & 9 demonstrates coverage after ac_accounting and dc_extest respectively achieving a total coverage of 76.85% and 96.25% respectively.



Global Statistics

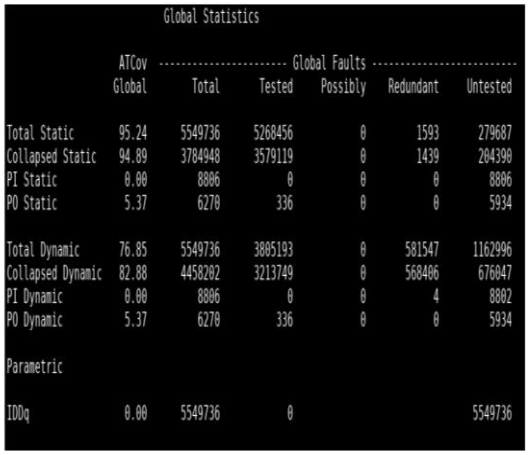
	#Faults	#Tested	#Possibly	#Redund	#Untested	%TCov	%ATCov
Total Static	5549736	5269583	8768	1593	269792	94.95	95.26

****

----Final Pattern Statistics----

Test Section Type	# Test Sequences
Scan	1
Logic	144
Total	145

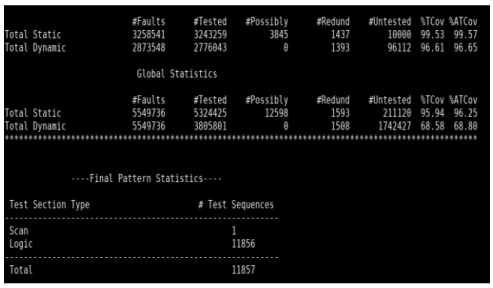
Figure 7: DC testmode coverage for RAM dominated block



Global Statistics

	ATCov Global	Global Faults Total	Tested	Possibly	Redundant	Untested
Total Static	95.24	5549736	5268456	0	1593	279687
Collapsed Static	94.89	3784948	3579119	0	1439	204390
PI Static	0.00	8806	0	0	0	8806
PO Static	5.37	6270	336	0	0	5934
Total Dynamic	76.85	5549736	3885193	0	581547	1162996
Collapsed Dynamic	82.88	4458202	3213749	0	568406	676047
PI Dynamic	0.00	8806	0	0	4	8802
PO Dynamic	5.37	6270	336	0	0	5934
Parametric						
IDDq	0.00	5549736	0			5549736

Figure 8: Coverage details after ac_accounting



	#Faults	#Tested	#Possibly	#Redund	#Untested	%TCov	%ATCov
Total Static	3258541	3243259	3845	1437	10000	99.53	99.57
Total Dynamic	2873548	2776043	0	1393	96112	96.61	96.65

Global Statistics

	#Faults	#Tested	#Possibly	#Redund	#Untested	%TCov	%ATCov
Total Static	5549736	5324425	12598	1593	211128	95.94	96.25
Total Dynamic	5549736	3805801	0	1508	1742427	68.58	68.80

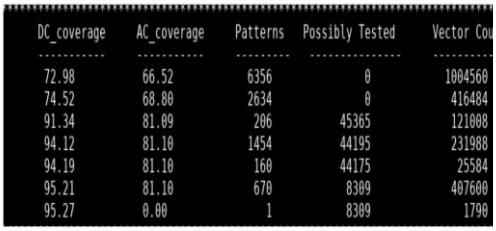
****

----Final Pattern Statistics----

Test Section Type	# Test Sequences
Scan	1
Logic	11856
Total	11857

Figure 9: Coverage details after dc_extest

Figure 10 and Figure 11 illustrates total number of test points inserted in a design and the improvement in test coverage after TPI insertion and one extra transition top-up mode.



DC_coverage	AC_coverage	Patterns	Possibly Tested	Vector Cou
72.98	66.52	6356	0	1004560
74.52	68.80	2634	0	416484
91.34	81.09	206	45365	121008
94.12	81.10	1454	44195	231988
94.19	81.10	160	44175	25584
95.21	81.10	670	8309	407600
95.27	0.00	1	8309	1790

Figure 13: Pattern Validation for RAM dominated block

Table 1: Test Coverage comparison of two blocks

Parameters	ISP block	RAM dominated
DC Coverage	99.01%	96.25
AC Coverage	80.31%	81.09
# of test vectors	29837	13351
Pattern Validation	✓	✓

Figure 12 and 13 demonstrates pattern validation for one of the testmodes for ISP and RAM dominated blocks respectively using IES simulator of Cadence.

5. Discussion

The ISP block has the various test modes which includes stuck- at (DC) and transition (AC) modes. The ATPG coverage for DC and AC modes are 99.01% and 80.34% respectively. Similarly, for the RAM dominated block, the ATPG coverage achieved are 95.26% (DC) and 68.82% (AC). By considering fault coverage accounting to improve AC coverage of ISP block, the coverage was improved to 94.90%. In the similar manner, although extest and ac_accounting was performed to improve DC and AC coverages respectively, the coverages were really not up to the mark (76.85% for transition coverage and 95.27% for stuck-at fault coverage). Other methods such as Test point insertion and also extra transition top-up mode was utilized to further improve the coverage. The AC and DC coverage achieved after Test point insertion and one extra transition top-up mode introduction are respectively 81.09% and 96.25%. Design is simulated using IES simulator of Cadence to validate the generated patterns.

Further studies involve mapping these block pins to device level pins for ease of test.

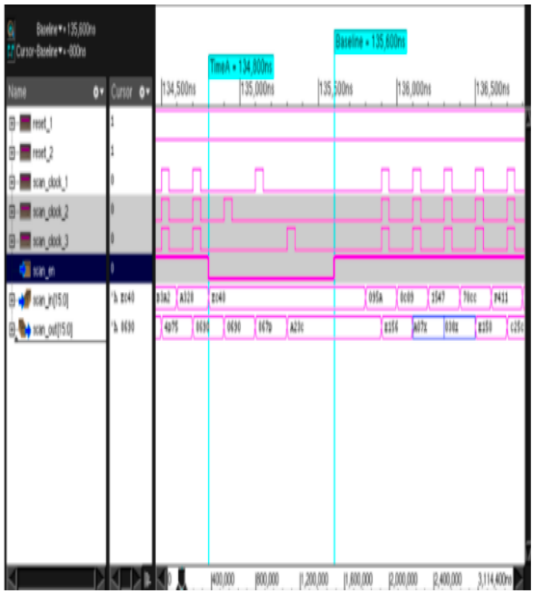


Figure 12: Pattern Validation for ISP block

References

[1] I. Pomeranz, "Testability Evaluation for Local Design Modifications," in IEEE Transactions on Very Large-Scale Integration (VLSI) Systems, vol. 32, no. 1, pp. 195-199, Jan. 2024

[2] M. A. Ibrahim, M. M. Abdel-Aziz, M. S. Abdelwahab, A. A. Mohamed, N. S. Soliman and A. A. Abou-Auf, "A Comprehensive Comparison Between Design for Testability Techniques for Total Dose Testing of Flash-Based FPGAs," in IEEE Transactions on Nuclear Science, vol. 68, no. 8, pp. 2232-2238, Aug. 2021

[3] I. Pomeranz, "Invisible-Scan: A Design-for-Testability Approach for Functional Test Sequences," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 38, no. 12, pp. 2357-2365, Dec. 2019

[4] E. Renold Sam Vethamuthu, S. Sivanantham and R. Sakthivel, "Implementation of Hierarchical DFT Approach for Better Testability," 2018 International Conference on Emerging Trends and Innovations in Engineering and Technological Research (ICETIETR), Ernakulam, India, 2018

[5] J. Aerts and E. J. Marinissen, "Scan chain design for test time reduction in core-based ICs," Proceedings International Test Conference 2021 (IEEE Cat. No.98CH36270), Washington, DC, USA, 2021

[6] K. M. Butler, J. Saxena, A. Jain, T. Fryars, J. Lewis and G. Hetherington, "Minimizing power consumption in scan testing: pattern generation and DFT techniques," 2019 International Conference on Test, Charlotte, NC, USA, 2019

[7] H. H. Lo, W. F. Lee, M. B. I. Reaz, N. Hisham and A. Y. M. Shakaff, "Design methodology to achieve good testability of VLSI chips: An industrial perspective," 2008

[8] Ravikiran S E, P. Narashimaraja, "DFT Flow for Automotive SOC", IJAST, vol. 29, no. 06, pp. 6838 - 6843, Jun. 2020.

[9] P., P.., Sai, Vishnu, Soudri., Ramakanth, Kumar, P., Sahana, Bailuguttu.(2022). Enhancement of observability using Kubernetes operator. Indonesian Journal of Electrical Engineering and Computer Science, 25(1), 496-496. Available from: 10.11591/ijeecs.v25.i1.pp496-503

[10] Kumar YGP, Kariyappa BS, Kurian MZ. (2022) Design, Implementation and Performance Analysis of Test Pattern Generator for Built-In Self-Test using an m-GDI Technology. Indian Journal of Science and Technology. 15(5):221226. https://doi.org/10.17485/IJST/v15i5.1846

[11] M. Renovell, S. Rayon, Y. Bertrand and G. Cambon," Testability design for PLA-implemented finite state machine," [1989] Proceedings of the 1st European Test Conference, Paris, France, 1989, pp. 246-251, doi: 10.1109/ETC.1989.36250

[12] Kowen Lai, C. A. Papachristou and M. Baklashov," BIST testability enhancement using high level test synthesis for behavioral and structural designs," Proceedings Sixth Asian Test Symposium (ATS'97), Akita, Japan, 1997, pp. 338-343, doi: 10.1109/ATS.1997.643980

[13] G. Bezzi, C. Bolchini, I. Bolzoni, S. Cantu, F. Fummi and D. Sciuto," Design for testability issues in the implementation of sequential array architectures," Proceedings of 1994 International Conference on Wafer Scale Integration (ICWSI), San Francisco, CA, USA, 1994, pp. 169-178, doi: 10.1109/ICWSI.1994.291254