5-2020

Built-In Self-Test (BIST) for Multi-Threshold NULL Convention Logic (MTNCL) Circuits

Brett Sparkman
University of Arkansas, Fayetteville

Built-In Self-Test (BIST) for Multi-Threshold
NULL Convention Logic (MTNCL) Circuits

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy in Engineering

by

Brett Sparkman
University of Arkansas
Bachelor of Science in Electrical Engineering, 2011
University of Arkansas
Master of Science in Electrical Engineering, 2017

May 2020
University of Arkansas

This dissertation is approved for recommendation to the Graduate Council.

_____
Scott Smith, Ph.D.
Dissertation Director

_____
Jia Di, Ph.D.
Committee Member

_____
Alan Mantooth, Ph.D.
Committee Member

_____
Jingxian Wu, Ph.D.
Committee Member

Abstract

This dissertation proposes a Built-In Self-Test (BIST) hardware implementation for Multi-Threshold NULL Convention Logic (MTNCL) circuits. Two different methods are proposed: an area-optimized topology that requires minimal area overhead, and a test-performance-optimized topology that utilizes parallelism and internal hardware to reduce the overall test time through additional controllability points. Furthermore, an automated software flow is proposed to insert, simulate, and analyze an input MTNCL netlist to obtain a desired fault coverage, if possible, through iterative digital and fault simulations. The proposed automated flow is capable of producing both area-optimized and test-performance-optimized BIST circuits and scripts for digital and fault simulation using commercial software that may be utilized to manually verify or adjust further, if desired.

Acknowledgements

This dissertation would not have been completed without the outstanding support from my wife, parents, and siblings. The greatest thanks of all goes to my wife, Allie, for enduring so many late nights and busy weekends. I would also like to thank Dr. Scott C. Smith and Dr. Jia Di, who have provided me with countless learning opportunities and persisted to support me throughout my various endeavors throughout graduate school. I would also like to thank my other committee members, Dr. Mantooth and Dr. Wu. Lastly, I would like to thank Wolfspeed and John Fraley, my manager and friend, for enabling me to continue pursuing higher education during my employment.

Table of Contents

1 Introduction 1

2 Background 2

2.1 NULL Convention Logic (NCL) 2

2.2 Multi-Threshold NULL Convention Logic (MTNCL) 5

2.1 Synchronous Test Methods 9

2.2 Asynchronous Test Methods 11

3 Built-In Self-Test of MTNCL Circuits 14

3.1 Area-Optimized MTNCL BIST Stage Implementation 14

3.2 Test-Performance-Optimized MTNCL BIST Implementation 16

3.3 MTNCL BIST Block Implementation 18

3.4 MTNCL BIST Top-Level Design 25

4 MTNCL BIST Automation 27

4.1 Area-Optimized Implementation Automation 33

4.2 Test-Performance-Optimized Implementation Automation 44

5 Experimental Results 53

5.1 MTNCL Design Preparation 53

5.2 General MTNCL BIST Automation Procedure 56

5.3 Area-Optimized MTNCL BIST Results 57

5.4 Test-Performance-Optimized MTNCL BIST Results 64

5.5 Comparison of Area-Optimized and Test-Performance-Optimized BIST Results 70

5.6 BIST Automation Performance 73

5.7 Transistor-Level Simulation 83

5.8 FSM and Feedback Compatibility 84

5.9 Fault Exclusion Method Based on Operation Principles 92

5.10 MTNCL Acknowledge Architecture Fault Improvement 93

5.11 Controllability and Observability Improvements 93

6 Conclusion 98

7 Future Work 99

8 References 100

List of Figures

Figure 1. NCL Pipeline Architecture [1]. 4

Figure 2. THmn Threshold Gate [1]. 4

Figure 3. NCL Gate Structure [1]. 4

Figure 4. MTCMOS Power Gating [5]. 5

Figure 5. MTNCL Gate Structure [5]. 6

Figure 6. MTNCL Pipeline Architecture [5]. 8

Figure 7. MTNCL Slept Early Completion Component. 8

Figure 8. Linear Feedback Shift Register (LFSR). 10

Figure 9. Multiple Input Shift Register (MISR). 11

Figure 10. Asynchronous DFT MTNCL Scan Chain Design [7]. © 2016 IEEE. 12

Figure 11. I_{DDQ} Test for Semi-Static NCL Gates (right) [8]. © 2016 IEEE. 13

Figure 12. Interleaved Scan Structure for NCL [9]. © 2016 IEEE. 13

Figure 13. Advanced Interleaved Scan Architecture for NCL BIST (a) RTS (b) STUMPS [9]. © 2016 IEEE. 14

Figure 14. MTNCL Area-Optimized BIST Stage. 15

Figure 15. MTNCL Test-Performance-Optimized BIST Stages. 17

Figure 16. Final MTNCL BIST Block Architecture for BIST Stage with Output Register. 19

Figure 17. Completion Tree Component for Five Input Bits. 20

Figure 18. Final MTNCL BIST Block Architecture for BIST Stage with Output Combinational Logic. 21

Figure 19. Equivalence Component for Input Pattern of 00101. 22

Figure 20. Dual-Rail Gating (DRG) Component. 23

Figure 21. MTNCL BIST Top-Level Design for Area-Optimized MTNCL BIST Implementation. 25

Figure 22. Top-Level MTNCL BIST Design for Test-Performance-Optimized MTNCL BIST Implementation for Four-Stage MTNCL Pipeline DUT. 27

Figure 23. Flowchart of High-Level Tool Procedure. 28

Figure 24. Flowchart for Parsing Netlist..... 29

Figure 25. Flowchart for Area-Optimized MTNCL BIST Implementation Automation. 35

Figure 26. Base MTNCL BIST Block Architecture for BIST Stage with Output Register. 37

Figure 27. Module MTNCL BIST Block Architecture for BIST Stage with Output Register..... 39

Figure 28. Flowchart for Test-Performance-Optimized MTNCL BIST Implementation Automation. 45

Figure 29. Base MTNCL BIST Block Architecture for BIST Stage with Output Combinational Logic. 47

Figure 30. Module MTNCL BIST Block Architecture for BIST Stage with Output Combinational Logic. 49

Figure 31. Fault Coverage vs. Pattern Count for c1355 Design. 74

Figure 32. Fault Simulation Time vs. Pattern Count for c1355 Design. 74

Figure 33. Digital Simulation Time vs. Pattern Count for c1355 Design..... 76

Figure 34. Total Run Time vs. Pattern Count for c1355 Design. 76

Figure 35. Fault Coverage Distribution for Area-Optimized c1355 Design. 78

Figure 36. Fault Coverage Distribution for Test-Performance-Optimized c1355 Design..... 78

Figure 37. Fault Simulation Time Distribution for Area-Optimized c1355 Design..... 79

Figure 38. Fault Simulation Time Distribution for Test-Performance-Optimized c1355 Design.79

Figure 39. Digital Simulation Time Distribution for Area-Optimized c1355 Design. 81

Figure 40. Digital Simulation Time Distribution for Test-Performance-Optimized c1355 Design. 81

Figure 41. Total Run Time Distribution for Area-Optimized c1355 Design. 82

Figure 42. Total Run Time Distribution for Test-Performance-Optimized c1355 Design..... 82

Figure 43. Digital Simulation of c17 Design. 84

Figure 44. Transistor-Level Simulation of c17 Design. 84

Figure 45. FSM Design Structure Block Diagram..... 85

Figure 46. Digital Simulation of scl_c17_fsm Area-Optimized BIST Stage Design. 86

Figure 47. Digital Simulation of scl_c17_fsm Test-Performance-Optimized BIST Stage 0 Design. 86

Figure 48. Digital Simulation of scl_c17_fsm Test-Performance-Optimized BIST Stage 1 Design. 86

Figure 49. Feedback Design Structure Block Diagram. 87

Figure 50. Digital Simulation of simple_fb Area-Optimized BIST Stage Design. 88

Figure 51. Fault Simulation Output Showing Invalid TetraMAX Sequential Simulation. 89

Figure 52. Digital Simulation of simple_adj_fb Area-Optimized BIST Stage Design. 90

Figure 53. Digital Simulation of simple_adj_fb Test-Performance BIST Stage 0 Design..... 91

Figure 54. Digital Simulation of simple_adj_fb Test-Performance BIST Stage 1 Design..... 91

Figure 55. Fault Simulation Output Showing Valid TetraMAX Sequential Simulation. 91

Figure 56. MTNCL Controllability Hardware. 95

Figure 57. MTNCL Observability Probing with Stalled Pipeline. 96

Figure 58. MTNCL Architecture Adjustment for Controllability and Observability Improvement. 96

List of Tables

Table 1. LFSR State Table. 10

Table 2. Truth Table for DRG. 23

Table 3. UNCLE Benchmark Synthesis Summary. 55

Table 4. MTNCL Designs Evaluated with BIST Automation. 55

Table 5. Area-Optimized MTNCL BIST Area Comparison. 58

Table 6. Area-Optimized MTNCL BIST Results for First Test Case (75% Fault Coverage, 0 Patterns, 1 Seed). 60

Table 7. Area-Optimized MTNCL BIST Maximum Results for Second Test Case (100% Fault Coverage, 1E6 Patterns, 1 Seed). 62

Table 8. Area-Optimized MTNCL BIST Relaxed Results for Second Test Case (100% Fault Coverage, 1E6 Patterns, 1 Seed). 62

Table 9. Area-Optimized MTNCL BIST Maximum Results for Third Test Case (100% Fault Coverage, 1000 Patterns, 1000 Seeds). 64

Table 10. Test-Performance-Optimized MTNCL BIST Area Comparison. 65

Table 11. Test-Performance-Optimized MTNCL BIST Results for First Test Case (75% Fault Coverage, 0 Patterns, 1 Seed). 66

Table 12. Test-Performance-Optimized MTNCL BIST Maximum Results for Second Test Case (100% Fault Coverage, 1E6 Patterns, 1 Seed). 68

Table 13. Test-Performance-Optimized MTNCL BIST Relaxed Results for Second Test Case (100% Fault Coverage, 1E6 Patterns, 1 Seed). 68

Table 14. Test-Performance-Optimized MTNCL BIST Maximum Results for Third Test Case (100% Fault Coverage, 1E3 Patterns, 1E3 Seeds). 70

Table 15. Area-Optimized and Test-Performance-Optimized MTNCL BIST Area Comparison for Second Test Case (100% Fault Coverage, 1E6 Patterns, 1 Seed). 72

Table 16. Area-Optimized and Test-Performance-Optimized MTNCL BIST Fault Coverage Comparison for Second Test Case (100% Fault Coverage, 1E6 Patterns, 1 Seed). 72

Table 17. Controllability and Observability Fault Coverage Improvements. 98

1 Introduction

While synchronous circuits have been the dominant architecture in digital systems for decades, asynchronous designs exhibit several advantages that are becoming more enticing as fabrication process technology continues to shrink. Several of the primary advantages involve the lack of a global clock, resulting in reduced power, noise, and electromagnetic interference, and robustness to PVT (process, voltage, temperature) variations [1]. However, there are several barriers to adoption of asynchronous design styles, including lack of designer familiarity with asynchronous architectures, synthesis methods to generate asynchronous circuits from register-transfer level (RTL) hardware description language (HDL) code, and testing methods to validate functionality of the resulting asynchronous circuits.

Fortunately, several asynchronous design synthesis systems have been developed [2], [3]. Although these software tools provide the ability to generate designs from RTL, little research has been done in terms of tools to support the testing of these asynchronous designs. It is imperative that testing methods are developed to allow the ease of asynchronous design creation and integration into larger systems.

The final objective of this work is to develop an automated tool to insert built-in self-test (BIST) functionality [4] into multi-threshold NULL convention logic (MTNCL) designs [5] and obtain and validate desired fault coverages. An automated tool to perform this task will reduce the effort required to design self-testing MTNCL circuits. Standard synchronous test software is leveraged when possible to minimize custom software, while also providing designers with a sense of familiarity. For a usage scenario, a designer will provide an MTNCL netlist, an area or performance preference, and the desired fault coverage. The tool will then be able to parse the netlist, determine register stages within the MTNCL design, insert the BIST structures into the

design, create and use simulation macros to both simulate the design in a digital simulation environment, and evaluate the fault coverage. The final design will be exported and include the BIST implementation, testbenches, and simulation macros so that the user can manually evaluate the design, if desired.

2 Background

2.1 NULL Convention Logic (NCL)

NCL is a quasi-delay insensitive (QDI) asynchronous design paradigm that is symbolically complete [6]. To achieve this, NCL utilizes a 1-hot encoding scheme, wherein any single one of N wires, called rails, is asserted to represent a DATA value, and all rails are de-asserted to represent a NULL value, which represents absence of DATA. Only one rail of a multi-rail signal may be asserted at any given time; if multiple rails are simultaneously asserted, the state is illegal. The most commonly used encoding scheme is dual-rail logic; in this version there are two rails, where the D^1 rail is asserted to represent a Boolean logic 1, the D^0 rail is asserted to represent a Boolean logic 0, and both rails are de-asserted to represent the NULL state. For NCL, each DATA wavefront must be followed by a NULL wavefront that resets the circuit to the NULL state (all gate outputs are logic 0) before the next DATA wavefront can be processed. The flow of DATA/NULL wavefronts is controlled by handshaking signals [1].

In an NCL pipeline, stages consist of NCL registers, NCL combinational logic (C/L), and completion detection components, as shown in Figure 1. Each NCL register has a data input port, a data output port, an acknowledge in (K_i) port, and an acknowledge out (K_o), port. For signals present on the acknowledge ports, one of two states are possible: request for DATA (RFD) or request for NULL (RFN). When RFD is present on the K_i port of an NCL register, the register allows a DATA wavefront to pass and then presents a RFN on its K_o. Likewise, when a RFN is

presented to the K_i port, the register allows a NULL wavefront to pass and then presents a RFD on its K_o. The function of the completion components is to merge the multiple signals at each register's K_o into a single signal; RFD or RFN is presented at the completion detection component's output when all inputs to the completion detection component are either RFD or RFN, respectively [1].

The NCL registers, C/L, and completion detection circuits are comprised of NCL threshold gates. Threshold gates, shown in Figure 2, use a naming convention of THmn, where *n* denotes the number of inputs and *m* denotes the threshold value. The output of a threshold gate is asserted when at least *m* of the *n* inputs are asserted. NCL gates exhibit hysteresis, such that once the output is asserted, it will remain asserted until all inputs are de-asserted [1].

To enable delay-insensitivity, NCL C/L must be input-complete and observable. Input-completeness requires that all outputs may not transition from NULL to DATA until all inputs have transitioned to DATA. Similarly, all outputs may not transition from DATA to NULL until all inputs have transitioned to NULL. Observability requires that any wires that transition during a DATA wavefront and do not affect the output may not propagate through a gate [1].

The generic block structure of an NCL threshold gate is shown in Figure 3. It consists of a pull-down Set block and a pull-up Reset block. When the inputs to complete the logical function are present, the Set block pulls down the internal node, and the output will rise. The Hold1 block provides the hysteresis function to hold the output high until all inputs are de-asserted. Likewise, when a NULL wavefront is passing through the system, the Reset block pulls the internal node high when all gate inputs are logic 0, such that the output will be de-asserted. The Hold0 block provides the hysteresis function to keep the output at logic 0 until the set function is true [1].

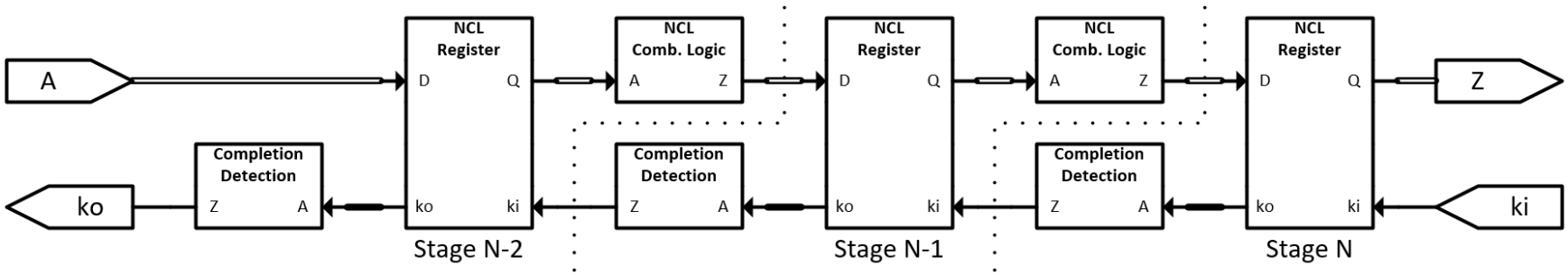

Figure 1. NCL Pipeline Architecture [1].

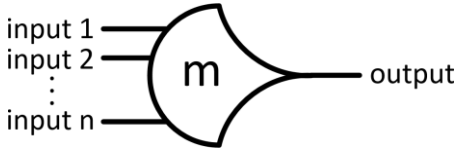

Figure 2. THmn Threshold Gate [1].

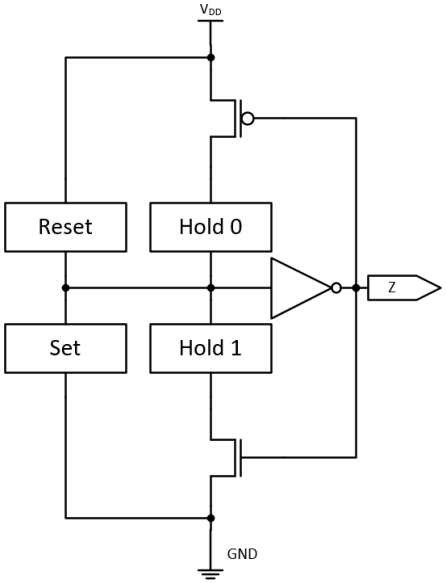

Figure 3. NCL Gate Structure [1].

2.2 Multi-Threshold NULL Convention Logic (MTNCL)

MTNCL is an adaptation of NCL that is capable of achieving substantially lower leakage power, lower dynamic power, faster performance, and reduced area. This is accomplished using multi-threshold CMOS (MTCMOS) and the concept of power gating [5].

In MTCMOS, transistors are utilized that possess different threshold voltages (V_t). High-V_t transistors are slower than standard-V_t transistors but have lower leakage current. Likewise, low-V_t transistors are faster than standard-V_t transistors but have higher leakage current. An example of MTCMOS power gating is shown in Figure 4, where a high-V_t transistor is inserted as either a header or footer switch between power or ground, respectively, and the low-V_t logic. The high-V_t transistor is controlled using an external *sleep* signal that allows the circuit to function when the *sleep* signal is de-asserted. The high-V_t transistor has much lower leakage than other transistors, enabling significant reduction of static power dissipation, since all current flows through that transistor. Performance is improved by utilizing low-V_t transistors for all other transistors, as these transistors are faster than standard-V_t or high-V_t transistors [5].

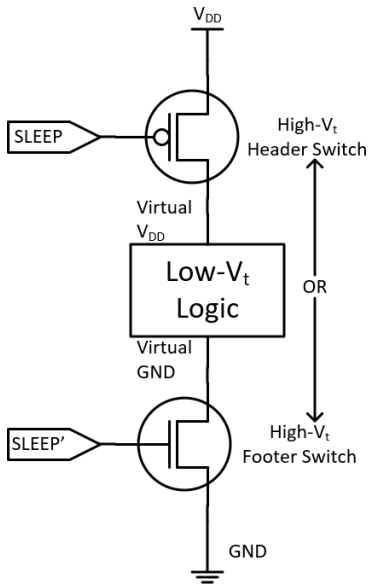

Figure 4. MTCMOS Power Gating [5].

To produce an MTNCL gate, the high-V_t (circled) and low-V_t transistors are incorporated into an NCL gate as shown in Figure 5. The NULL wavefront is forced by the *sleep* signal, so the Reset block is not necessary and is removed to save area. For the Early Completion MTNCL architecture [5], input-completeness is provided by the sleep mechanism; therefore the Hold1 block to provide hysteresis may also be removed. The Hold0 block utilizes all high-V_t transistors, since these are only turned on when the gate is slept during the NULL wavefront; therefore, performance is not reduced, but leakage current is reduced during operation. One high-V_t transistor is implemented in every path to ground in the Set block to provide low-leakage during NULL wavefronts; all other Set transistors utilize low-V_t for increased speed [5].

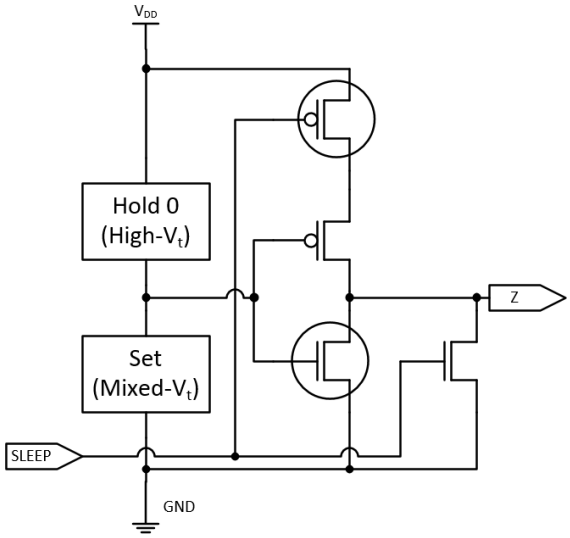

Figure 5. MTNCL Gate Structure [5].

In an MTNCL pipeline, stages consist of MTNCL registers, MTNCL combinational logic, and slept early completion detection components, as shown in Figure 6. Each MTNCL register only has a data input port, a data output port, and a Boolean *sleep* input. The acknowledge RFD and RFN signals are generated and combined in the slept early completion detection components in a fashion similar to NCL. These signals control the flow of DATA and NULL through the pipeline and also provide the sleep control mechanism. When DATA input is

present for the current pipeline and the following pipeline stage acknowledge is RFD, the current stage will be unslept to allow the DATA wavefront to pass. When the current stage is slept during RFN, all MTNCL gates in the stage are reset to 0, which is the NULL state.

The MTNCL slept early completion component is shown in Figure 7, which is comprised of MTNCL TH12 gates to detect a DATA or NULL for each register input bit, and a tree of MTNCL THnn gates to combine the multiple TH12 gate outputs into a single signal, which is then combined with the subsequent stage's early completion component's sleep output via a resettable inverted NCL TH22 gate, to generate the current stage's early completion component's sleep output.

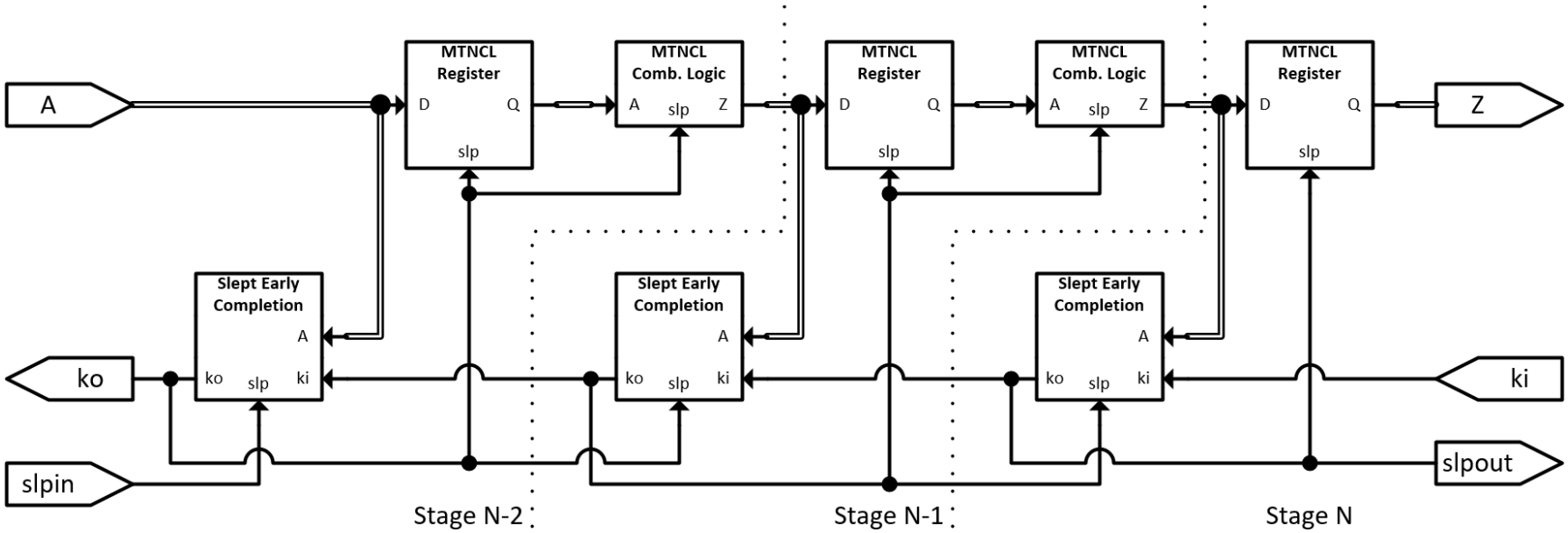

Figure 6. MTNCL Pipeline Architecture [5].

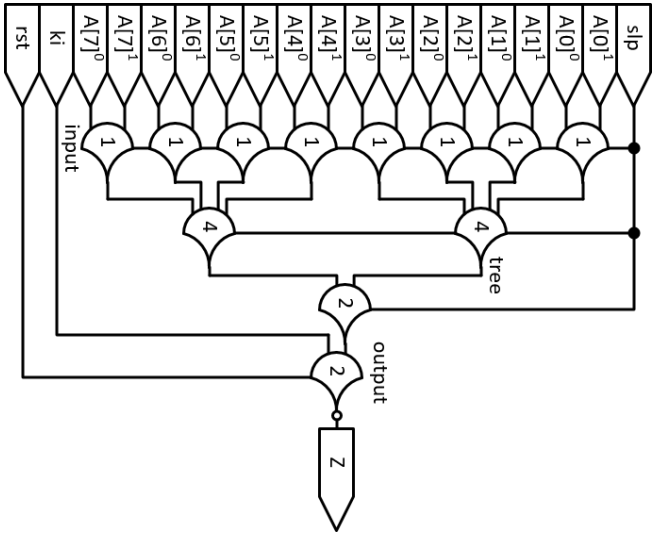

Figure 7. MTNCL Slept Early Completion Component.

2.1 Synchronous Test Methods

Integrated circuits (ICs) require testing to detect faults that can occur during the fabrication process, such as a wire being shorted to ground (i.e., stuck-at-0) or shorted to V_{DD} (i.e., stuck-at-1), to ensure correct operation. Test methods for synchronous circuit designs have been well-established in both literature and practice. Design For Testability (DFT) methods [4] are commonly employed to test circuits. This form of testing typically requires external equipment to both present the input patterns and measure the output patterns. Scan methods are frequently employed by adjusting the Device Under Test (DUT) to embed serial shift functionality into the registers of the circuit. By shifting data into a primary input, applying the appropriate number of clock periods to perform the combinational function, and then shifting data out of a primary output, it is possible to apply test patterns to numerous circuits inside the DUT and measure the responses with a low I/O count. To further improve test performance, the primary scan chain may be broken up into parallel scan chains. Scan chains are common choices for DFT due to the availability of both DFT insertion and automatic test pattern generation (ATPG) software [4].

BIST methods incorporate test validation into the DUT so that input patterns may be presented and outputs validated internally, without requiring external test hardware. To present input patterns to the DUT, linear feedback shift registers (LFSRs) are regularly used. These are essentially circular shift registers with XOR elements in various feedback paths to provide the desired input patterns. An example 3-bit LFSR circuit design is shown in Figure 8. The LFSRs require a reset to a known seed value and then produce a pseudorandom pattern that is deterministic. A depiction of the various states for the 3-bit LFSR mentioned above is shown in Table 1. LFSRs may be designed to present a maximal length 2^n-1 test patterns, where *n* is the

number of LFSR bits. The all-0 test pattern is not possible in LFSRs unless the number of bits is increased and at least one of the LFSR outputs is not utilized; however, this may lead to additional test time as the number of possible output patterns is increased. Other methods may implement a larger memory structure such as read-only memory (ROM) to provide several specific test patterns. Designs including this may utilize DFT scan methods so that overall pattern counts are low and coverage high, at the expense of additional area requirements for memory and auxiliary circuits [4].

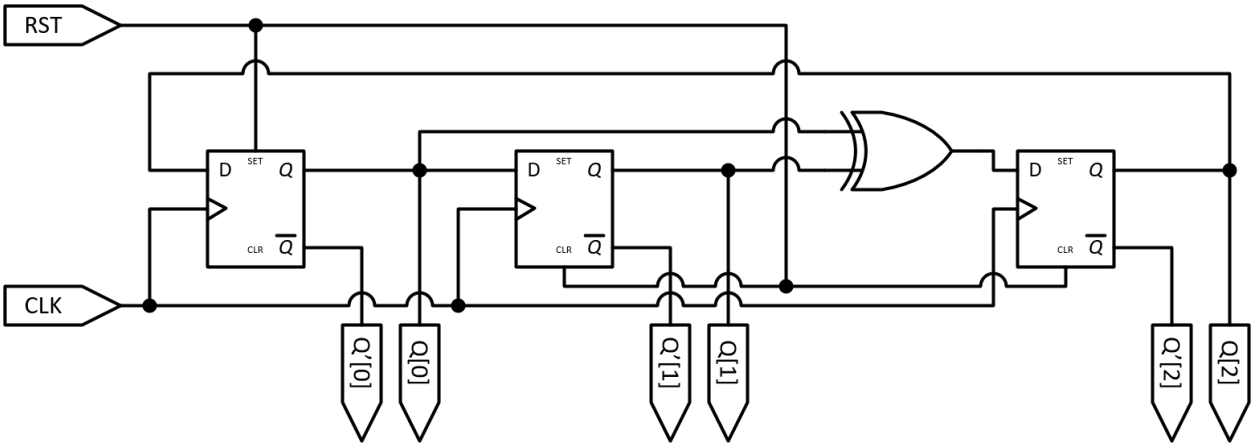

Figure 8. Linear Feedback Shift Register (LFSR).

Table 1. LFSR State Table.

State	Output	State	Output	State	Output	State	Output
0	001	**2**	100	**4**	111	**6**	110
1	010	**3**	101	**5**	011	**0**	001

To measure the outputs, several methods have been developed. Transition counting is one method wherein the number of transitions on the output, or where the output changes state, is counted and compared to the correct number of transitions assuming a correctly-functioning DUT. However, this may provide a false positive diagnosis because the number of transitions for an invalid DUT could match the number of transitions for a valid DUT. Parity bits are another output response validation technique in which the parity of each consecutive output is measured,

and the final parity of the test patterns is compared to the proper parity. This parity validation could yield a false diagnosis as each pattern yields a single parity bit, so two bits flipped in the output response would yield a good result. Multiple input shift registers (MISRs) are frequently utilized to provide a form of output measurement [4]. MISRs are essentially LFSRs with additional XOR gates inserted between every stage of the shift register with the second input tied to an external input. This provides a method for the input to influence the state of the LFSR. An example MISR is shown in Figure 9.

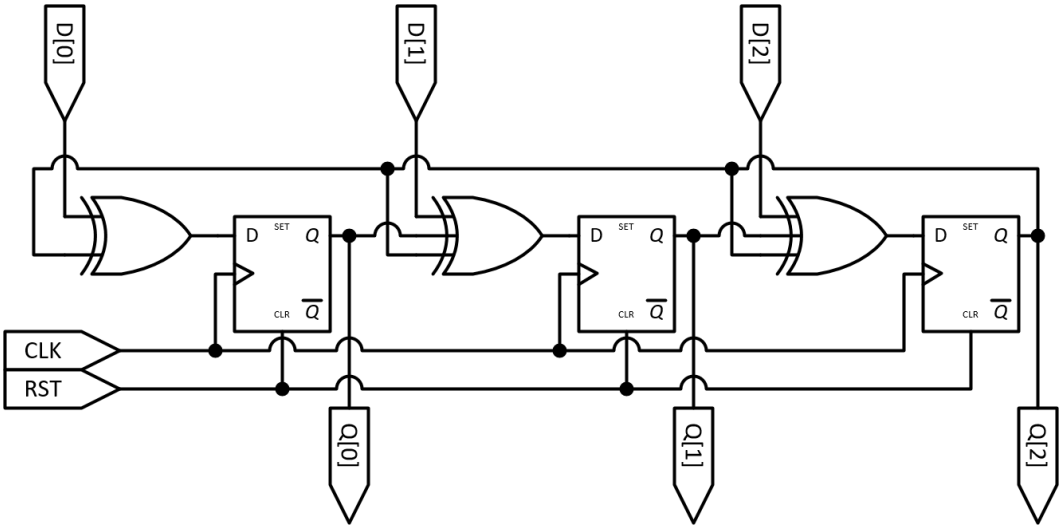

Figure 9. Multiple Input Shift Register (MISR).

2.2 Asynchronous Test Methods

There are far fewer test methods for asynchronous circuit paradigms, such as NCL or MTNCL. Hence, traditional synchronous methods are typically modified to work with a specific asynchronous paradigm. Due to the limited number of asynchronous test methods, tests requiring additional testing equipment or significant additional on-chip circuitry are frequently used.

DFT techniques involving scan chains have been examined to allow synchronous ATPG tools to be utilized with MTNCL [7]. This allows standard ATPG tools to generate low numbers of input vectors that yield high fault coverages. A block diagram of this DFT strategy is shown in Figure 10. However, this testing methodology requires testing equipment or additional integrated

hardware to provide the test patterns. Integrated hardware would likely consist of ROM that would hold the test patterns and output responses along with a controller to present the inputs, measure the outputs, and also validate the functionality of the circuit; this would require a large area overhead, especially for large pattern counts.

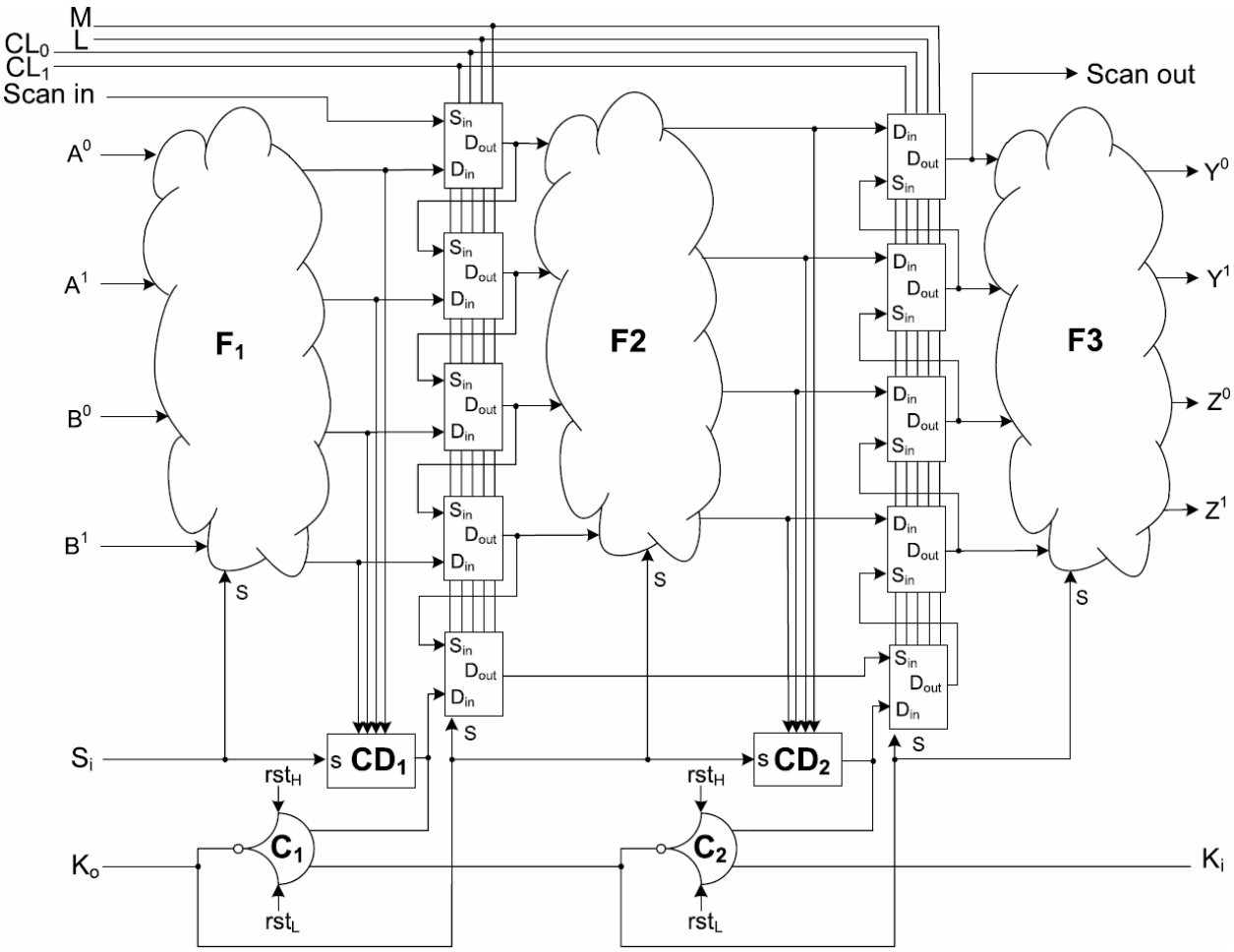

Figure 10. Asynchronous DFT MTNCL Scan Chain Design [7]. © 2016 IEEE.

Quiescent current testing, also known as I_{DDQ} testing, has been designed for NCL circuits [8]. In this test, a number of random inputs are presented to the primary inputs of the DUT, and the supply current of the DUT is measured. If stuck or bridging faults are present in the circuit, higher current draws should be observable under certain input conditions, as shown in Figure 11. This form of testing requires some form of equipment or hardware to present the stimulus to the circuit and a sensitive measuring device, since supply current draws may be rather low. Testing

is slow, as the I_{DDQ} measurements must be taken after the dynamic current draws from the charging or discharging of internal capacitance occur.

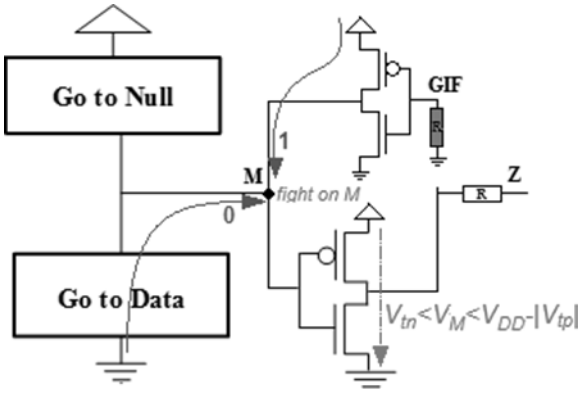

Figure 11. I_{DDQ} Test for Semi-Static NCL Gates (right) [8]. © 2016 IEEE.

BIST methods have also been investigated. An asynchronous interleaved scan architecture was implemented in [9]. This method implemented two scan paths inside of NCL circuits that would generate the required alternating DATA and NULL wavefront between consecutive register stages, as shown in Figure 12. Two versions were designed: one consisted of a single long scan chain with two test pattern generators (TPGs) and output response analyzers (ORAs), and a second that had multiple parallel scan chains with two larger TPGs and ORAs. Block diagrams for these circuits are shown in Figure 13. Detailed descriptions of the TPGs and ORAs are excluded, but [9] does mention that these exist as external structures. Additionally, a controller is necessary to facilitate the interleaved nature of DATA and NULL wavefronts.

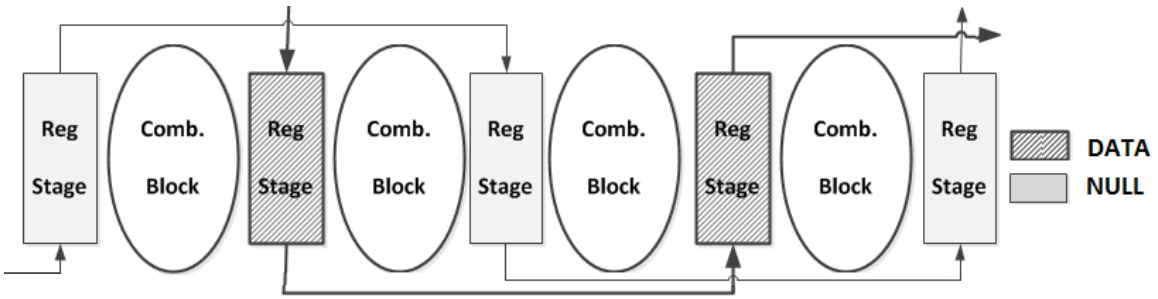

Figure 12. Interleaved Scan Structure for NCL [9]. © 2016 IEEE.

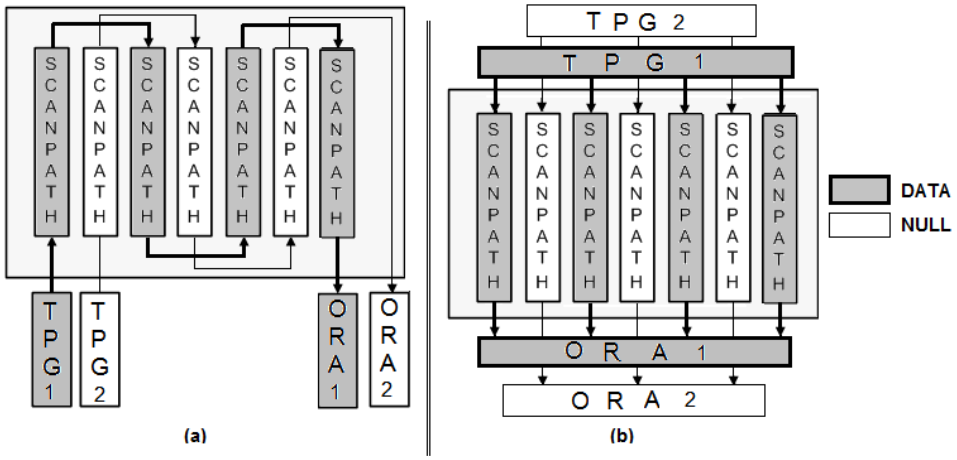

Figure 13. Advanced Interleaved Scan Architecture for NCL BIST (a) RTS (b) STUMPS [9]. © 2016 IEEE.

3 Built-In Self-Test of MTNCL Circuits

Two methods of BIST were implemented for MTNCL circuits. The first implementation requires less area but may result in a longer test duration; the second utilizes parallelism and design reuse to increase controllability and reduce test time. In this section, the two methods are presented, and a common architecture for using them is detailed.

3.1 Area-Optimized MTNCL BIST Stage Implementation

An area-optimized BIST implementation was designed that enables simple functional BIST with minimal area overhead. The area-optimized BIST stage, indicated by a dashed box in Figure 14, is essentially the complete MTNCL pipeline from the DUT, and the BIST architecture functions as a wrapper around the entire DUT. All the input and output ports remain the same for the BIST stage. Due to this, test patterns can only be applied to primary inputs, and only primary outputs can be measured. This may limit the controllability and observability of the DUT, especially as the number of pipeline stages and logic depth increases, similar to synchronous designs [4]. It is important to note that the DATA output component of this implementation is an MTNCL register; when the register's slept early completion component de-asserts the *sleep* signal, there will be a delay before the final DATA output appears on the MTNCL register.

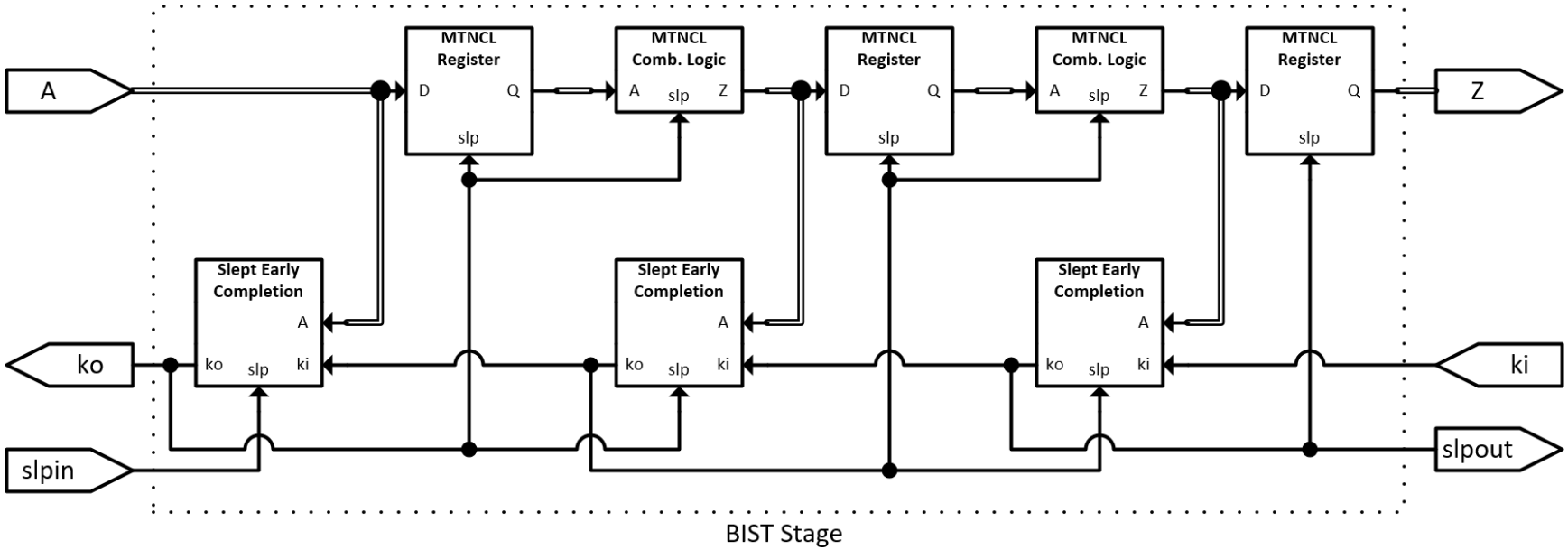

Figure 14. MTNCL Area-Optimized BIST Stage.

3.2 Test-Performance-Optimized MTNCL BIST Implementation

A test-performance-optimized BIST implementation was designed, which requires a larger area overhead compared to the area-optimized BIST implementation but yields higher test performance through parallelism of multiple BIST stages and design reuse. Contrary to the area-optimized BIST implementation, the MTNCL DUT is parsed for pipeline stages and broken up into multiple BIST stages as shown below in Figure 15. Due to the structure of MTNCL pipeline stages, the initial and final BIST stages are slightly different than the intermediate BIST stages. Intermediate BIST stages consist of the MTNCL register and combinational logic for a pipeline stage and the following pipeline stage's completion detection component. The initial stage also includes the first input completion detection component; the final stage also includes the final register. The inputs to each stage consist of a data input and output, *ki* and *slpin* inputs, and *ko* and *slpout* outputs. Once again, it is important to note that the DATA output component of the final stage is an MTNCL register; when the register's slept early completion component de-asserts the *sleep* signal, there will be a delay before the final DATA output appears on the MTNCL register. For a 2-stage MTNCL pipeline, there will be two BIST stages; intermediate BIST stages are required for MTNCL pipelines larger than 2-stages deep.

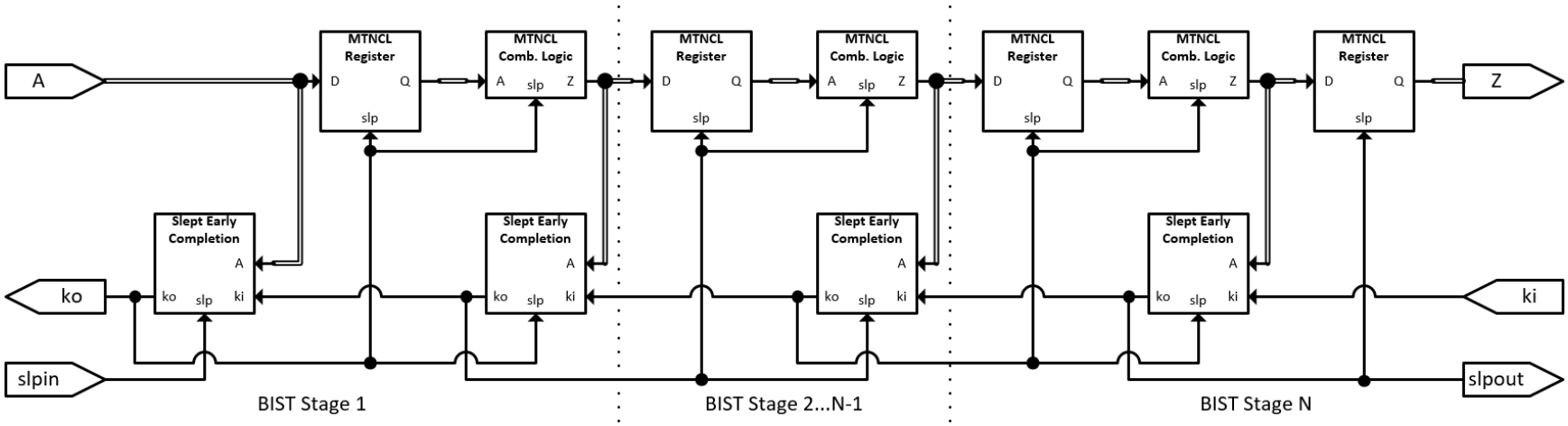

Figure 15. MTNCL Test-Performance-Optimized BIST Stages.

3.3 MTNCL BIST Block Implementation

To implement MTNCL BIST, traditional synchronous BIST methods were adjusted for compatibility with MTNCL asynchronous systems. As LFSRs are a simple and effective way to present a large number of input patterns, while requiring minimal additional logic, an LFSR was utilized to generate the BIST inputs. Since DFFs typically have both a Q and Q' output with potentially slightly different timing delays for these signals, a dual-rail gating (DRG) component was implemented to allow for proper flow of DATA and NULL wavefronts by presenting a NULL wavefront when its *D/N'* control signal is 0 and a DATA wavefront when its *D/N'* control signal is 1. The output response of the BIST stages were measured with an MISR by connecting both the D^0 and D^1 rails of the circuit to inputs of the MISR to enable checking both rails simultaneously. Multiplexers were used to control the flow of data between standard operation and BIST mode. Additionally, simple Boolean logical equivalence checkers were utilized to control the number of input patterns presented to the DUT by gating off the LFSR clock once the final input pattern was presented, and then validating that the final MISR output was the expected value, meaning that the circuit is functioning correctly.

A schematic of the final MTNCL BIST block architecture is shown in Figure 16 for the single-BIST-stage area-optimized BIST implementation and the last BIST stage of the test-performance-optimized BIST implementation. These BIST stages have an MTNCL register for the data output; they do not have a completion detection component connected to the data output, so there is no way to ensure that valid DATA is present inherently from the DUT. Thus, a completion tree component is added to detect when the BIST stage data output has become valid DATA, and then clock the MISR. This ensures that DATA is stable when the MISR is clocked. The operation of the circuit is detailed below after Table 2.

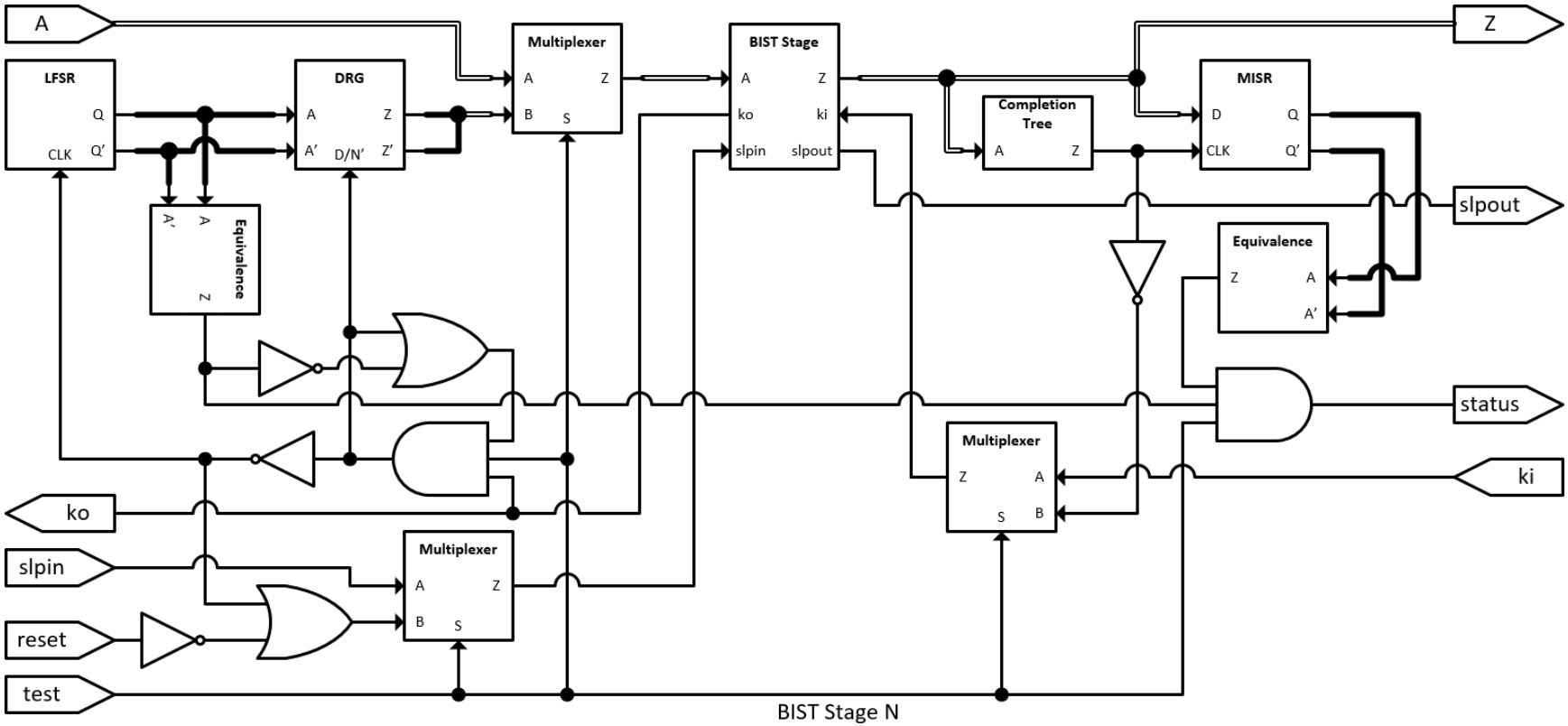

Figure 16. Final MTNCL BIST Block Architecture for BIST Stage with Output Register.

A slightly modified MTNCL BIST block architecture is shown in Figure 18. This architecture is utilized for all BIST stages in the test-performance-optimized BIST implementation except for the final stage, which requires an added completion tree component as shown in Figure 17. The test-performance-optimized implementation re-uses the completion components within the MTNCL pipeline to reduce hardware overhead. Since the internal completion component is directly connected to the BIST stage data output as shown in Figure 15, it serves as a valid function for determining when DATA arrives, and can therefore be utilized with the addition of only a single inverter. The operation of the circuit is detailed below after Table 2.

The purpose of completion tree component is to produce a Boolean logic 1 output once all inputs have transitioned from NULL to DATA and to produce a Boolean logic 0 output once all inputs have transitioned from DATA to NULL. As a valid DATA wavefront must only have one of the two rails asserted, TH12 gates are utilized to determine when each dual-rail input signal has become DATA or NULL. A threshold-gate-based AND-tree utilizing TH44, TH33, and TH22 gates is then appended to merge all the input completion logic into a single final output that is only asserted once each input is DATA, and only de-asserted once each input is NULL. A completion tree component for five input bits is shown in Figure 17.

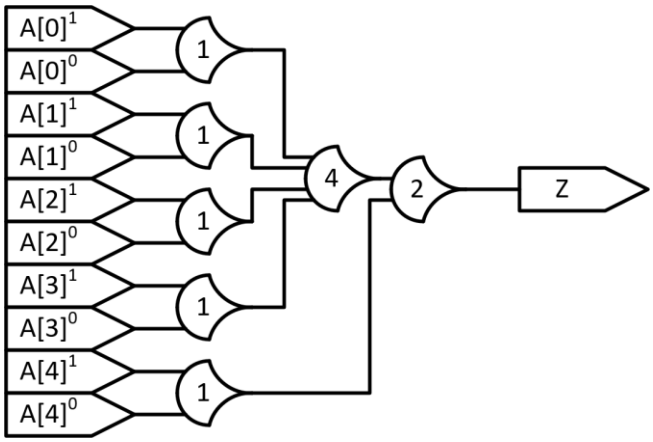

Figure 17. Completion Tree Component for Five Input Bits.

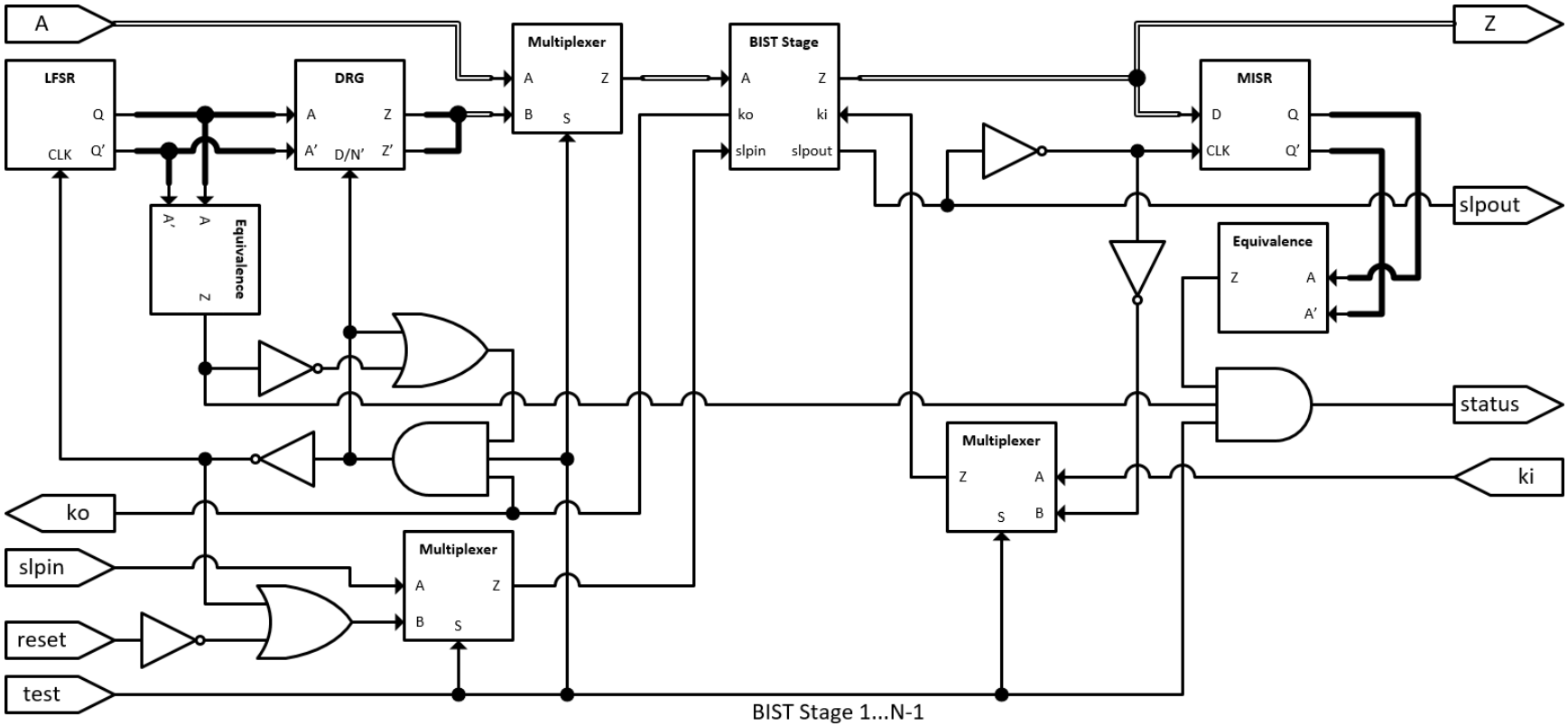

Figure 18. Final MTNCL BIST Block Architecture for BIST Stage with Output Combinational Logic.

Similarly, the purpose of the Boolean equivalence component is to assert its output once the inputs reach a specific Boolean condition. This component is generated to form a gate-count-optimized AND-tree using every input bit or its complement so that only one input condition may assert the output. This component is paired with DFFs from an LFSR or MISR that provide the input bit and its complement, so no additional hardware is required for any inversions. An example of an equivalence component with an input pattern of 00101 is shown in Figure 19. The least significant four input bits are merged using a 4-input AND gate, and the most significant input bit is merged with this using a 2-input AND gate.

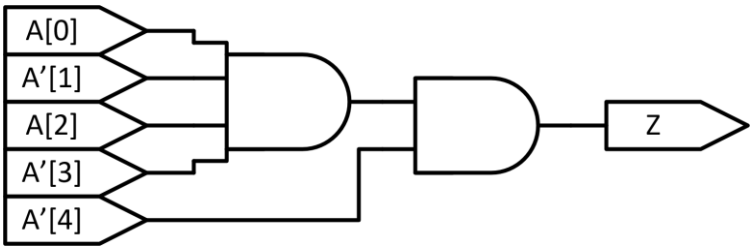

Figure 19. Equivalence Component for Input Pattern of 00101.

The DRG component is combined with the LFSR to produce a valid DATA/NULL wavefront, essentially enabling a common synchronous design to function with asynchronous circuits. The schematic of a single bit DRG component is shown in Figure 20, and the truth table is shown in Table 2. As RFD is Boolean logic 1 and RFN is Boolean logic 0, the *D/N'* control signal could be connected directly to the output of completion detection components, and the Q and Q' of a DFF enabled passing the proper DATA and NULL wavefronts from the output of the LFSR. It is important to note that the DFF should be clocked with the inverse of the *D/N'* control signal to avoid glitches and potential data corruption in the DATA wavefront. If the same polarity is used, the DATA could transition to an invalid state where both the D^0 and D^1 rails are asserted for a brief moment during the shift operation of the LFSR. Using the inverse allows the

LFSR to shift only when NULL is present, so the DUT will only receive valid static DATA wavefronts following NULL wavefronts.

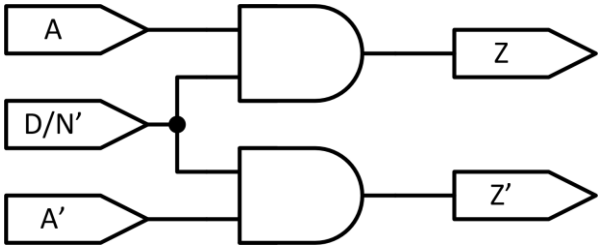

Figure 20. Dual-Rail Gating (DRG) Component.

Table 2. Truth Table for DRG.

D/N'	Output	Z	Z'
0	NULL	0	0
1	DATA	A	A'

To enter test mode, the circuit must first be reset. Then the test mode may be selected by asserting the *test* input. This configures the BIST stage input data multiplexer to provide inputs from the LFSR and DRG. The BIST stage acknowledge input signal, *ki*, is configured through a multiplexer to utilize either an inverted output from the output completion tree component in Figure 16 or a twice-inverted *slpout* signal in Figure 18, which produces the same effect. This enables the circuit to free run as long as valid DATA/NULL wavefronts are continuously provided at the BIST stage input as they are requested. Once a valid DATA wavefront arrives at the data output, the *ki* signal is transitioned to RFN to request a NULL wavefront. Likewise, once a valid NULL wavefront arrives at the data output, the *ki* signal will be adjusted to RFD to request a DATA wavefront. This simple circuit provides the full DATA/NULL flow required without the use of any actively changing control signals. Similar to how the BIST stage *ki* is controlled in an asynchronous fashion using a completion tree component or the internal completion detection component, the BIST stage sleep in signal, *slpin*, is primarily controlled by

the inverse of the BIST stage acknowledge output, *ko*. This inverted *ko* is merged with an OR of the inverted reset signal to ensure a valid state as the circuit is reset prior to test mode. This feedback system allows the BIST stage's input circuitry to be slept and produce the NULL wavefront when RFN is requested at the BIST stage *ko* output.

To enable the circuit to halt at a known test input condition, the equivalence component is configured to assert its output only once the LFSR's output matches the pattern following the desired final input pattern. The output from this equivalence component is inverted and fed into a combinational logic circuit that enables clocking the DRG and LFSR components only during test mode. Once the pattern following the final input pattern is reached, the combinational logic circuit freezes the *D/N'* control signal to the DRG component, providing the BIST stage with a static NULL data input. As the BIST stage data input never changes, the LFSR additionally is never clocked because the BIST stage acknowledge output, *ko*, is static at RFD; the equivalence component will never adjust its output. At this point, the inputs to the BIST stage will be frozen with the desired number of input patterns presented to the BIST stage.

The MISR utilized at the data output of the DUT measures that all the DATA wavefronts produced from each BIST stage are logically correct. The output of the MISR is also connected to a Boolean equivalence component to detect that the valid final checksum signature is present at the output of the MISR after all of the inputs have been presented to the BIST stage from the LFSR. To achieve this, the LFSR's equivalence component and MISR's equivalence component outputs are ANDed with the *test* signal; this allows the status output to raise if the circuit is functional during test mode and prevent variations of the status signal during standard operation of the DUT. There is a possibility that through a number of potential faults, the MISRs may evaluate to a known good checksum even if the circuit does not function properly. This is

referred to as aliasing and is beyond the scope of this dissertation, but there are methods to handle such issues [4]. The aliasing probability of the circuit can be calculated so that the probability of a faulty circuit providing the MISR output pattern indicating a properly functioning circuit is known [4]; also, MISR length, XOR taps, and both number and order of LFSR input patterns may be adjusted to reduce the aliasing probability, if needed. For each new variation of the LFSR inputs, fault analysis must be re-run to ensure that the desired fault coverage is achieved.

3.4 MTNCL BIST Top-Level Design

For the top-level BIST design for the area-optimized BIST implementation, the circuit is essentially the same as the schematic shown in Figure 16. The top-level design is shown in Figure 21. As the area-optimized BIST functions as a wrapper around the MTNCL DUT, one additional input and one additional output are required. The *test* port is added as an input to control when the circuit functions in BIST mode instead of the standard operating mode, and the *status* output is added to show if the DUT successfully arrives at the final MISR output, indicating a good self-test.

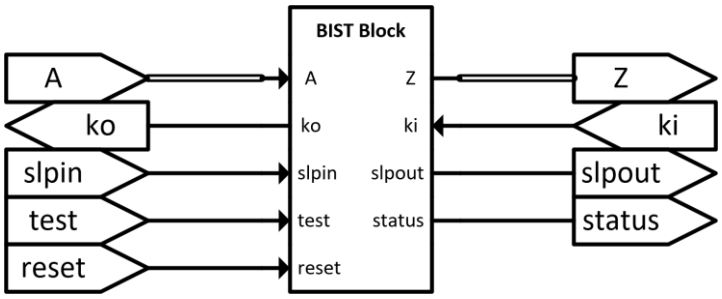

Figure 21. MTNCL BIST Top-Level Design for Area-Optimized MTNCL BIST Implementation.

In this configuration, the *ko* port may be monitored to observe that the DUT is running in the test mode. As the LFSR continues to provide inputs to the DUT, the *ko* should alternate between RFD and RFN as the circuit free-runs, indicating that the circuit is continuing to process various DATA and NULL wavefronts. Once this signal becomes static, it indicates that the test

will either be complete soon once the final input propagates to the output or that the circuit has locked up, potentially due to a fault in the circuit. In the event of a long-pipelined design, there may still be a significant period of time until the final DATA wavefront reaches the MISR, as *ko* indicates the DUT's response to inputs.

Likewise, the *slpout* port may be monitored to observe that the DUT is providing outputs to the MISR. This signal will de-assert as the final output register of the DUT is unslept to provide DATA wavefronts to the data output and MISR and assert as the final output register of the DUT is slept to provide a NULL wavefront to the data output. Note that the MISR doesn't validate NULL wavefronts; however, this is not needed, since all NULL wavefronts are the same (i.e., all rails are logic 0), and the circuit will halt operation if not able to transition back to NULL after any DATA wavefront. Once the *slpout* signal is static, the final DATA output has been clocked into the MISR, and the *status* signal indicates whether the design's built-in self-test has passed.

The top-level BIST design for the test-performance-optimized BIST implementation is shown in Figure 22. As the test-performance-optimized BIST implementation utilizes parallelism by breaking up each pipeline stage of the MTNCL design, each pipeline stage is included in its own BIST block in the top-level design. Additionally, the status of each is merged using an AND tree to show a final self-test status of all the BIST blocks. As each BIST block is capable of controlling itself using the BIST circuitry, the controllability and observability of the system is increased because each of the BIST stage data inputs and outputs are provided by an LFSR and measured using an MISR, respectively.

Similar to the area-optimized version, the *ko* of the first pipeline stage BIST block may be monitored to ensure that DATA/NULL wavefronts are being presented to the pipeline stage,

and the *slpout* of the final pipeline stage BIST block may be monitored to observe that the final BIST block MISR is receiving DATA/NULL wavefronts. The status of DATA/NULL wavefronts within internal BIST blocks may be observed by adding additional external outputs to monitor their *ko* and *slpout* signals, but this is not required.

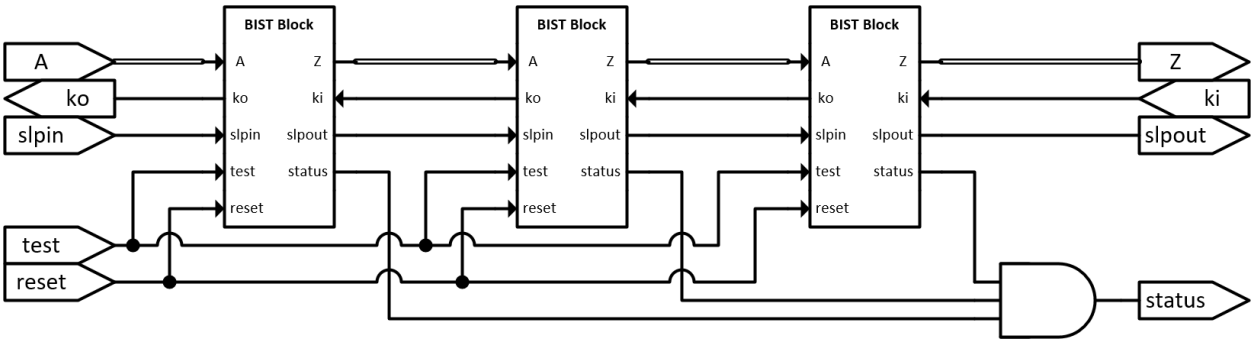

Figure 22. Top-Level MTNCL BIST Design for Test-Performance-Optimized MTNCL BIST Implementation for Four-Stage MTNCL Pipeline DUT.

4 MTNCL BIST Automation

An automated method was implemented to import an MTNCL DUT Verilog netlist, automatically insert the required MTNCL BIST logic, simulate digital functionality, evaluate fault coverage, and iterate , by first increasing the number of test patterns and then trying different LFSR initial values, until either the desired fault coverage is achieved or the maximum possible fault coverage is obtained. This automation tool was designed using Python for netlist parsing, implementing all BIST component netlists and testbenches, creating simulation macros, running both digital and fault simulations, evaluating simulation results, and iteration to improve fault coverage. Mentor Graphics ModelSim and Synopsys TetraMAX were utilized for digital simulation and fault simulation, respectively, as these are industry-standard synchronous software packages. The developed MTNCL BIST automation procedure was specifically designed to utilize these commercial tools, although modifications to function with other simulators would be possible.

A generic flowchart outlining the high-level developed procedure is shown in Figure 23 for a design in which the target fault coverage is capable of being obtained using only pattern count or LFSR seed adjustment. The details of each block are greatly expanded, and additional information is provided in following subsections. The subsections are broken up: one for area-optimized mode and one for test-performance-optimized mode.

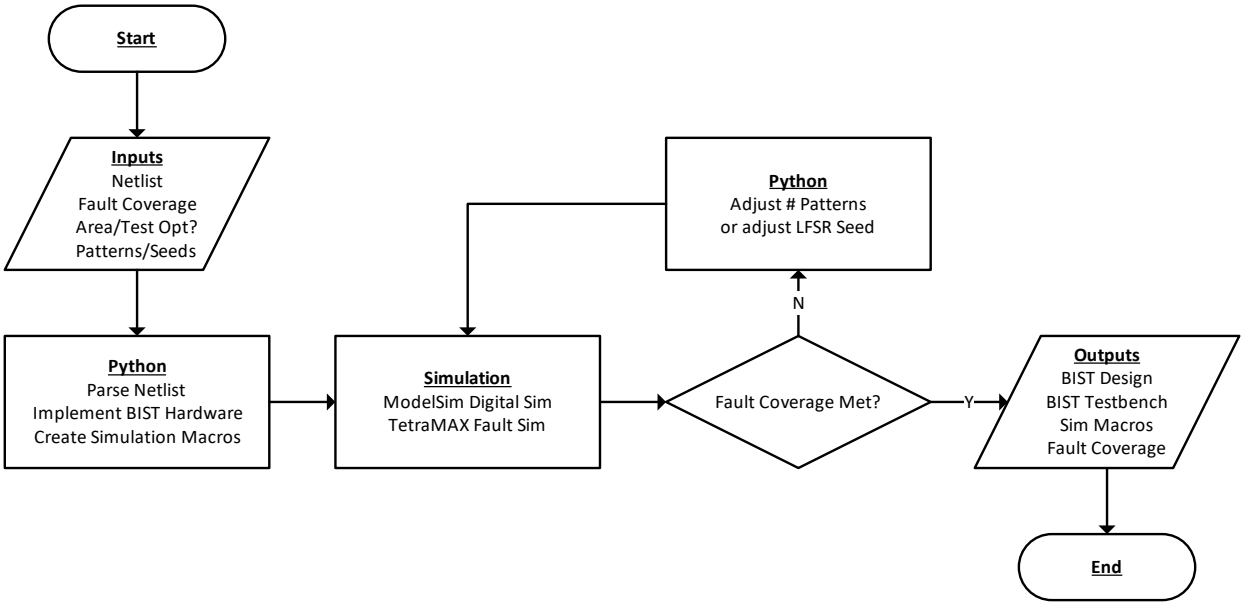

Figure 23. Flowchart of High-Level Tool Procedure.

A pipelined MTNCL design netlist is provided as input to the automation tool. This file must first be parsed to determine the input and output bit lengths, the number of pipeline stages, and what basic function block of an MTNCL pipeline each gate in the netlist belongs to: registers, completion detection components, or combinational logic. This is done through several stages of parsing the netlist. A basic flowchart of this algorithm is shown in Figure 24. The input netlists utilized for tested circuits were synthesized from RTL using the UNCLE toolset [2]. There are some slight nuances to netlists produced using this tool compared to standard MTNCL pipelines; the Python tool was designed to convert the UNCLE netlists into the standard MTNCL pipeline format.

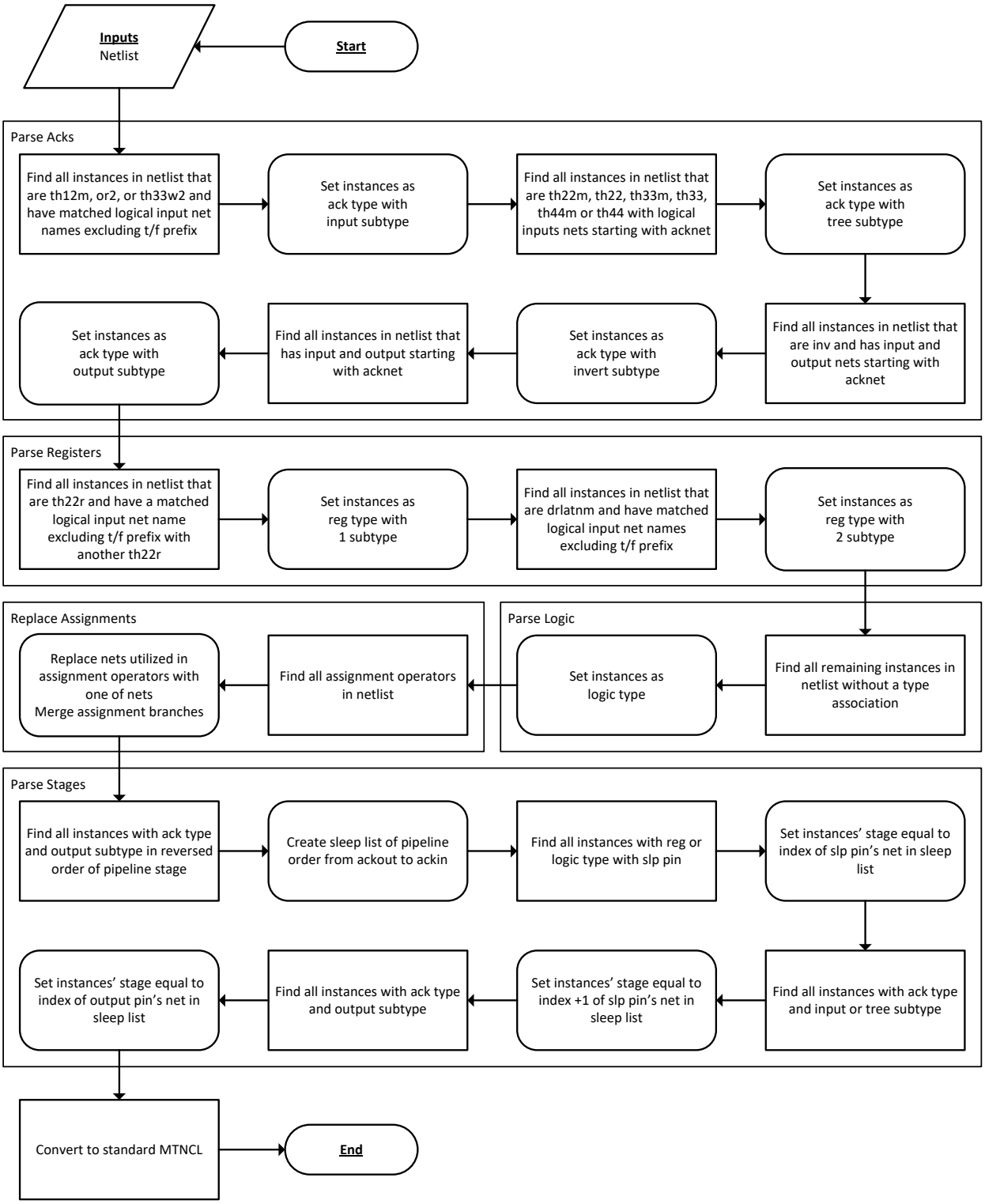

Figure 24. Flowchart for Parsing Netlist.

Once the input netlist is presented to the automation tool, it initially parses the netlist to find and classify the acknowledge completion detection components, further referred to as *acks*, as these control the flow of DATA/NULL wavefronts through the design and are critical to the proper functionality of the BIST circuitry and DUT itself. This section of parsing is referred to as parsing *acks*.

There are several subtypes of these *ack* type components. Referring back to Figure 7, the left-most components in which both rails of a dual-rail signal are connected are referred to as *input* subtypes. These consist of TH12m gates in standard MTNCL pipelines. However, UNCLE uses an NCL-based input completion detection circuit composed of OR2 gates (functionally equivalent to TH12 gates) for the first pipeline stage. UNCLE may also utilize a TH33w2, classified as an *input* subtype, to merge a single dangling dual-rail signal into the AND tree to reduce area. To ensure that none of these gates are actually utilized as combinational logic components, the inputs of the *input* subtype must have matched net names after the first character; UNCLE utilizes a "t_signal" and "f_signal" to denote D^1 rail and D^0 rail, respectively, where the "_signal" must be matched and the "signal" is the dual-rail name.

Following these dual-rail detectors is the threshold-based AND trees consisting of TH22m, TH33m, and TH44m gates, potentially with their NCL equivalents as used by UNCLE for the first pipeline stage. These gates are assigned a *tree* subtype.

The primary output stage of handshaking MTNCL completion components is a TH22ir, or a TH22r for the last stage of an UNCLE design, and is assigned an *output* subtype. In the gate name, *r* refers to the gate including a reset input to initialize the gate output to 0, and *i* means that the gate output is inverted. These merge the final output of the threshold-based AND tree, shown

in Figure 17, with the output from the following pipeline stage's acknowledge signal to enable asynchronous handshaking.

UNCLE additionally uses inverters in certain MTNCL architectures; if the input to an inverter is the output of a completion detection component, it is classified as an *inv* subtype.

Once parsing *ack* structures is complete, the registers of the MTNCL design must then be parsed for the *reg* type. UNCLE also utilizes an NCL-based output register comprised of TH22r gates for the output stage. These are grouped and checked for consistency amongst other TH22r gates to ensure that a "t_signal" and "f_signal" are present for each dual-rail signal where "signal" is a common dual-rail net name. As each of these is for a single rail of a dual-rail signal, these are given a subtype of *1* representing the total signal count of inputs to the register. The remainder of registers are comprised of drlatnm gates; these are classified as a reg type with a subtype of *2*, as each dual-rail register has both the D^1 and D^0 inputs.

As a safeguard, the sum of all *reg* types is compared against the count of all *ack* types with input subtypes. This helps ensure that both the acknowledge structure and register structures were properly parsed; a warning is placed in the log file if these quantities don't match. Once all these other classifications have been made, all remaining gates are classified as a *logic* type.

Due to difficulties managing the large number of assignments produced during synthesis, all of the assignments found in the netlist are down-selected to a single net name and replaced throughout the entire design. To further complicate this process, the order of net assignment and multi-net assignments can cause difficulties; nets may have multiple non-sequential assignments to other nets. A dictionary is created from each assignment structure, and net name priority is filtered to register inputs, as these dual-rail net names must match for both rails per the parsing implementation. The dictionary is iterated upon and restructured until only register input net

names, taking priority, or other logical net names are remaining in the design without any references to any other assignments. The assignment structures are adjusted so that they are commented out in any future exported netlists, as only the primary assignment net name that had priority is necessary. Then, all gates are iterated upon to remove any non-primary assignments in the structural port mapping and replace those net names with primary assignments.

Once the assignments are replaced, the pipeline stages are parsed to determine stage order. The primary handshaking signals coming from *ki* backwards towards *ko* passing through the *ack output* subtypes are determined. These are then reversed to present the parsing algorithm with an index-based sleep list from *ack outputs* that properly represents the *sleep* signals provided to various stages of the pipeline. Index zero would be the first pipeline stage, and index one would be the second pipeline stage. This would continue for all stages in the pipeline.

All gates are then assigned a stage using this sleep list. Any *reg* or *logic* types have a pipeline stage assigned equal to the matching index in the sleep list of their *sleep* pin because these components are slept by their pipeline stage's early completion component. The output registers utilized by UNCLE do not have a *sleep* pin as they are NCL-based; these are automatically assigned to the last pipeline stage. Any *ack* types with *input* or *tree* subtypes are assigned an index equal to the matching index plus one in the sleep list of their *sleep* pin because these components are slept from the preceding pipeline stage. As UNCLE utilizes an NCL-based early completion component for the initial pipeline stage, these are assigned a stage of 0 if they do not have a *sleep* pin. The *ack* type *output* subtypes are assigned a stage equal to the matching index in the sleep list of their output pin as these components sleep their own pipeline stage.

All UNCLE-based pipeline structures are then converted to the standard MTNCL pipeline structure shown in Figure 6. The *ack* type *input* subtypes OR2 gates are converted to

TH12m gates with the sleep pin connected to *slpin*, as these are only present in the first pipeline stage. Likewise, the TH33w2 gates are converted into TH33w2m gates. For *ack* type *tree* subtypes with NCL gates, these are adjusted to their MTNCL variation: a TH22m for a TH22, a TH33m for a TH33, or a TH44m for a TH44. Once again, these will have *sleep* inputs connected to the *slpin* signal. For the final pipeline stage, the *ack* type *output* subtype TH22r gate is replaced with a TH22ir gate to remain consistent in convention, and any inverters in the ack structures are removed. The output register TH22r gates are replaced with drlatnm gates; each pair of complimentary TH22r gates for the dual-rail signal are merged into this single component.

At this point, the function and structure of all components inside the MTNCL design are known and understood, so the various MTNCL BIST automation flows are possible. Without this parsing step, potential issues could arise due to the nuances employed by the UNCLE tool flow.

4.1 Area-Optimized Implementation Automation

A flowchart detailing the automation for the area-optimized MTNCL BIST implementation is shown in Figure 25. The inputs required are the actual MTNCL netlist, desired fault coverage, and area-optimized mode selected. Additionally, initial number of test patterns, maximum number of test patterns, initial LFSR seed, and maximum number of LFSR seeds can also be specified as optional inputs; otherwise, they default to 5 initial patterns, 2^n-2 max patterns, where *n* is the number of BIST stage inputs, a starting LFSR seed of binary 1, and 2 for maximum number of LFSR seeds, such that one additional seed other than the initial binary 1 is tried. The netlist is then parsed as previously mentioned. An iterative loop is then entered, in which digital and fault simulations are performed, and the final fault coverage is compared to the

desired fault coverage until either the desired fault coverage is achieved or the maximum number of patterns is reached. If desired fault coverage is not met, another iterative loop is entered that randomly adjusts the LFSR initial seed until either the desired fault coverage is achieved or the maximum number of seeds is reached. Once desired fault coverage is achieved, or the maximum number of patterns and initial seeds are analyzed, the final design is output, along with the additional hardware, simulation macros, and analysis of fault coverage. If the target fault coverage was not met, the final design is the last design tested, which may not be the one with the best fault coverage. In this case, the designer should check the end of the log file to determine the maximum fault coverage achieved by any of the designs analyzed, and then rerun the tool with this maximum achievable fault coverage as the desired fault coverage, in order to output the best design.

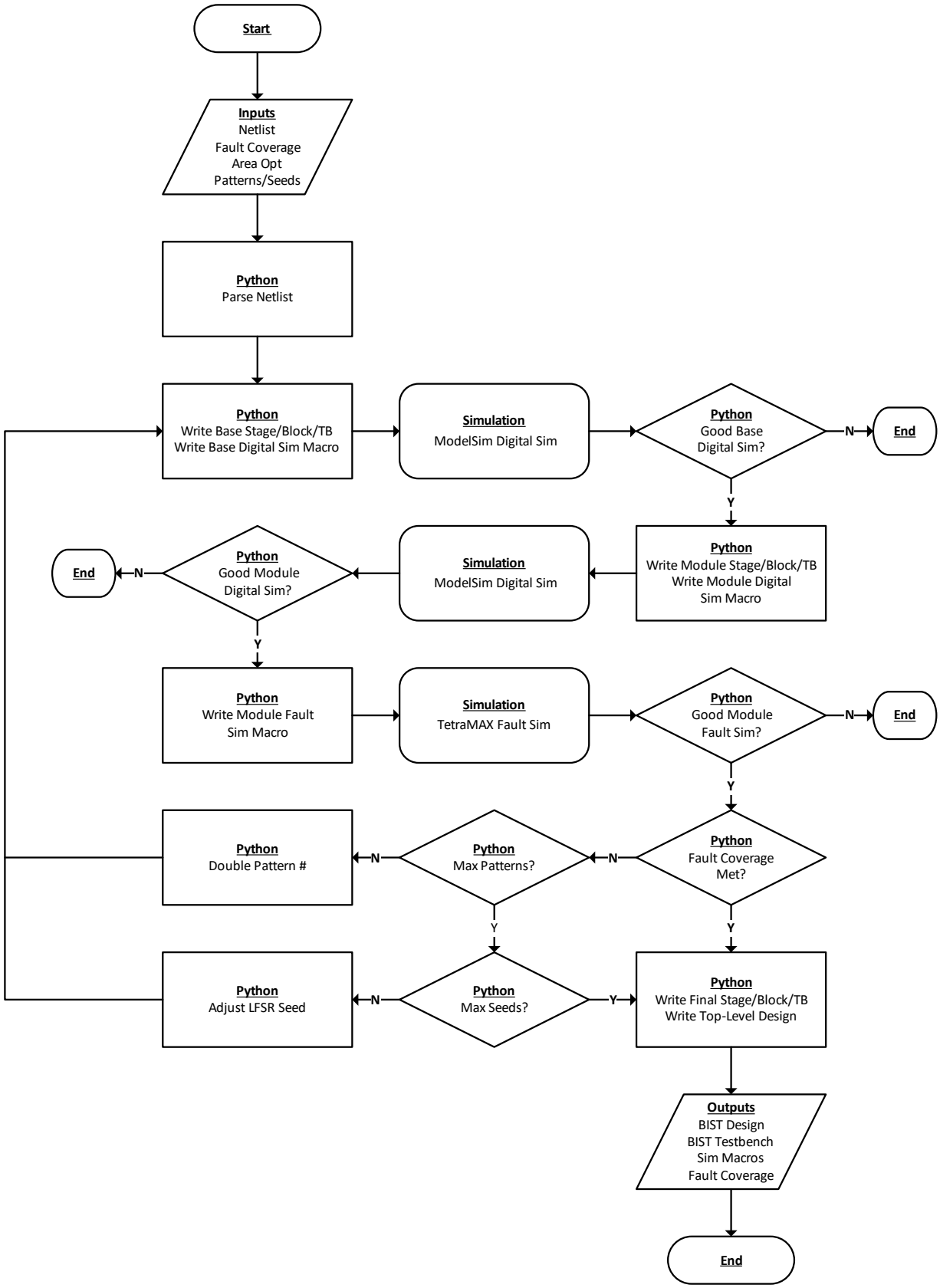

Figure 25. Flowchart for Area-Optimized MTNCL BIST Implementation Automation.

To create the various equivalence hardware, the LFSR and MISR outputs of a golden simulation must be run for the desired number of patterns. As digital simulation is an efficient manner of calculating theses values, a base BIST stage, block, and testbench is initially created.

The architecture for a base BIST block is shown in Figure 26 for the area-optimized implementation where the output of the BIST stage wrapper is always a register. Although it appears similar to the final BIST block shown in Figure 16, there are several key differences. Primarily, the equivalence circuits do not yet exist for these designs, as the final LFSR and MISR output patterns are currently unknown. Although the MISR outputs are not connected to any hardware, they are monitored during the simulation.

Since TetraMAX is a cyclic fault simulator, it is incapable of properly handling the asynchronous DATA and NULL wavefronts unless both the changing inputs and outputs occur in the same cycle; additionally, the DATA wavefront must be provided last so that the simulator may settle on the proper outputs. To enable this, the base BIST block requires both the input and output to transition only once during a cycle through the use of the added TH22 threshold gate that conjoins the BIST block's completion tree component's inverted output with the *ko* output from the BIST stage and feeds this back into the *ki* for the BIST stage. Essentially, this ensures that the BIST block will not request a NULL wavefront until after a valid DATA wavefront has appeared at the output, propagating from the static DATA input. To further constrain the system with a static DATA input per cycle, the DRG *D/N'* control signal is taken from this *ki* as well and inverted for the LFSR clock. If multiple DATA transitions occur in a single cycle, as in the case where inputs are provided as soon as requested, the TetraMAX functional simulator may produce invalid results compared to the golden simulation; this results in an invalid fault grading result.

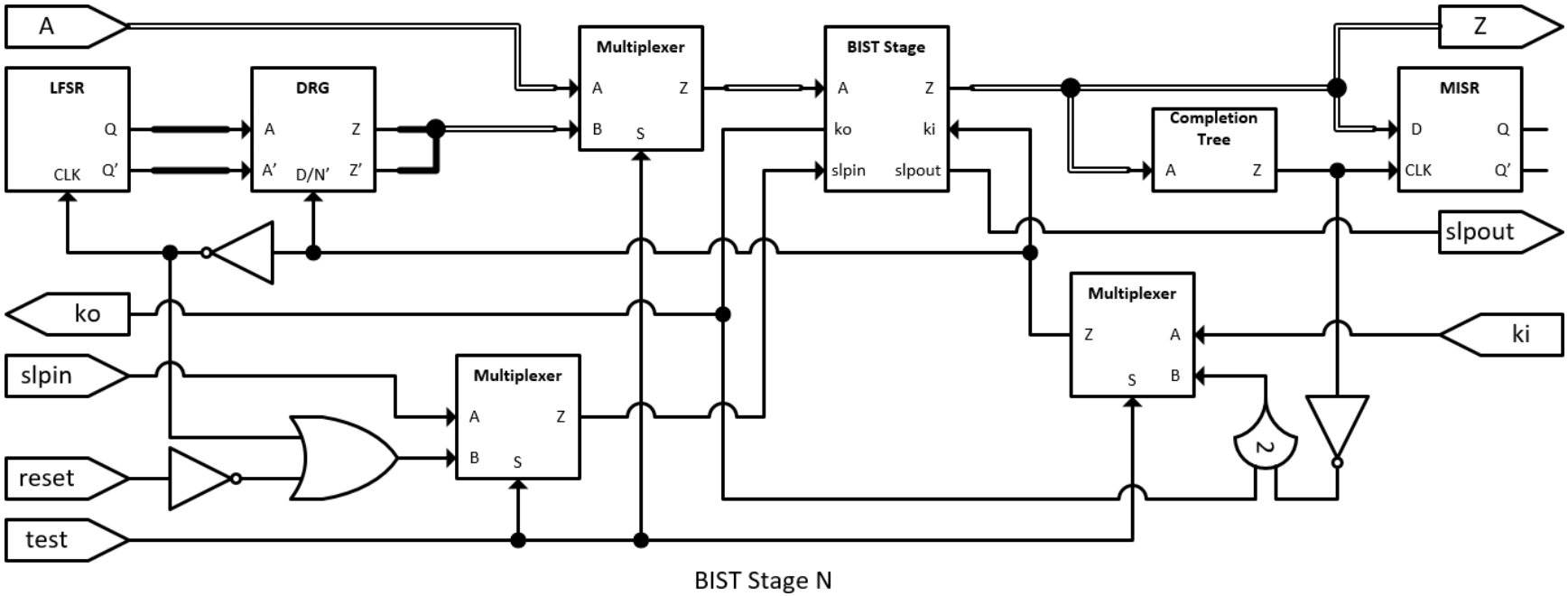

Figure 26. Base MTNCL BIST Block Architecture for BIST Stage with Output Register.

Additionally, a simulation macro for digital simulation is written, which includes measuring the outputs of the LFSR and MISR at the desired input and output pattern count along with the test time at which these outputs occur. However, this test time is not utilized by the tool. Once the final output count is reached, the simulation is stopped. This macro is run in the digital simulator, and the simulation log is checked once the simulation process closes to validate that the simulation completed successfully and to ensure that the LFSR pattern and MISR pattern are present in the log.

Once the LFSR and MISR pattern are known, the equivalence modules are created during the process of writing the module BIST stage, block, and testbench. The architecture for the module BIST block produced during the area-optimized implementation is shown in Figure 27. This architecture is a fusion of the Base and Final MTNCL BIST Block architectures. The LFSR equivalence circuit is produced to match the input pattern following the desired final input count so that the inputs will be frozen at a NULL wavefront between these two patterns, presenting the desired number of inputs to the BIST stage. The two inverters, AND, and OR gates connected to LFSR equivalence component enable this function; once the final pattern is reached, the LFSR is no longer clocked. Additionally, the *D/N'* DRG control signal remains low, providing a static NULL wavefront at the BIST stage input.

As the module BIST block architecture will be utilized for fault simulation by TetraMAX, it includes the constrained acknowledge signals merged with the TH22 gate and utilizes the BIST stage *ki* for the gating mechanism for the LFSR and DRG control signals. This ensures that only one DATA input will be presented to the BIST stage during each fault simulation cycle, as required by TetraMAX for fault simulation.

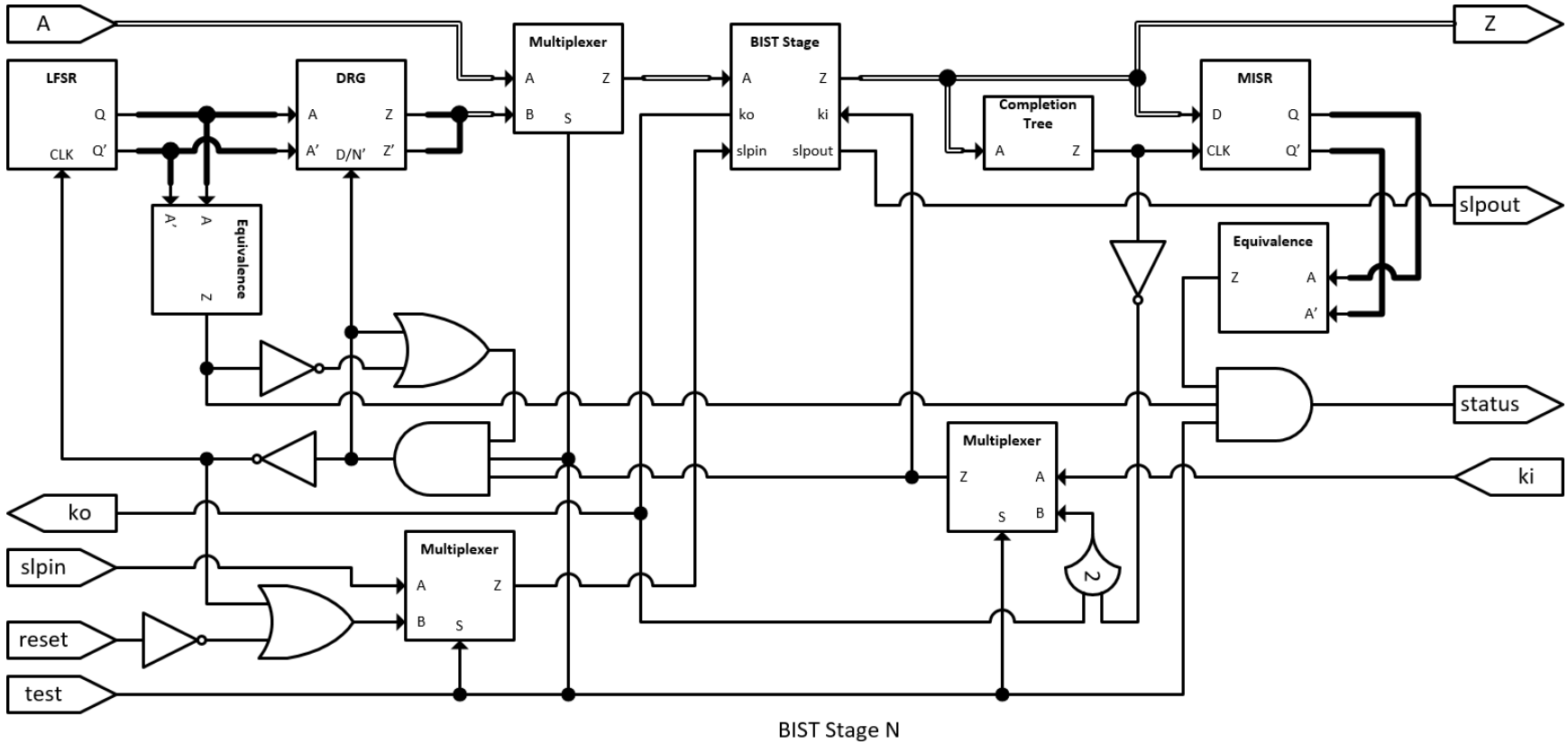

Figure 27. Module MTNCL BIST Block Architecture for BIST Stage with Output Register.

A macro, similar to the initial base BIST block simulation macro, is created for the module digital simulation. Instead of outputting the LFSR and MISR patterns at a specified input or output count, they are measured and recorded once the simulation has completed. This occurs once the final input DATA and NULL wavefronts are presented to the BIST stage, and once the circuit has reached a steady state after the final DATA and NULL wavefronts appear at the output. Additionally, all ports of the BIST stage are monitored, using a vcd dumpports command, and output to a VCD file. This creates a file structure that is checked prior to fault simulation to ensure that the digital simulation, referred to as the golden simulation, and functional fault simulation match. If the golden simulation and functional fault simulation do not match, any fault analysis may be invalid because nets may not transition as expected, and therefore any faults may not propagate correctly. Essentially, this file includes the entire set of results of the BIST stage during digital simulation; input value states, output value states, and all state transition times are recorded for all ports of the BIST stage. This macro is run in the digital simulator, and the simulation log is checked once the simulation process closes to validate that the simulation completed successfully, and to ensure that the final LFSR pattern and MISR pattern match the desired values that were previously generated during the base BIST block simulation.

Now that the MTNCL BIST block architecture including the equivalence circuitry is implemented and simulated digitally, a TetraMAX fault simulation macro is written. The models and netlists are read into the simulator. It is important to note that the version of TetraMAX simulator utilized was not capable of simulating behavioral models in Verilog. Thus, any behavioral models provided in the UNCLE toolset and utilized for MTNCL synthesis were adjusted to use a dataflow representation utilizing assignments. The fault models are built, and a

digital simulation is run inside TetraMAX and compared to the ModelSim digital simulation's VCD file. The fault grading simulation is then run, and the fault summary is recorded.

Although the fault simulation may have run, several things must be checked to ensure that it is valid. If the TetraMAX digital simulation matches with the ModelSim digital simulation, then the actual fault analysis should be valid since the circuits' behaviors match. If there is a mismatch between simulations, the fault simulation is disregarded, and the algorithm stops; additional fault simulations will incur the same issues. This should only happen if either model is not valid for use with the TetraMAX digital simulator, as mentioned above, or if there is an error in the DUT or BIST circuitry that does not follow standard MTNCL conventions.

If the fault simulation validly completes, then all fault summary information is parsed and utilized to calculate fault coverage. This includes the detected faults (DT), possibly detected faults (PT), undetectable faults (UD), ATPG untestable faults (AU), not detected faults (ND), and total faults. DT faults include faults that TetraMAX was able to completely evaluate as being detected with the present set of inputs. PT faults are evaluated when the good digital simulation values are known, but the faulty machine simulation resulted in an unknown state (X). UD faults are faults that cannot be tested and may be due to unused outputs, pins that are statically tied to 0 or 1, may have controllability or observability limitations, or may have redundant logic that would mask the actual fault. AU faults are faults that cannot be controlled or observed due to constraints utilized during fault simulation or faults regarding non-scan sequential devices. As these constraints and devices that would incur this are not utilized with this tool or MTNCL, faults of this type have not been observed. ND faults are faults that were not detected during fault simulation and may occur due to a lack of controllability or observability based on the current input pattern set. Total faults represent the total number of faults simulated.

The fault coverage is then calculated using the same method that TetraMAX uses, as shown in Eq. 1. As PT faults have a 50% chance of being detected in a binary system, they are assigned a weight of 0.5. None of the other fault types are utilized in this calculation, although they are recorded in the simulation log for preservation. Although TetraMAX directly outputs this fault coverage, additional calculation is necessary for the test-performance-optimized implementation, so the calculation is performed for the area-optimized implementation as well.

$$Fault\ Coverage = \frac{Detected\ Faults + 0.5 * Possible\ Detected\ Faults}{Total\ Faults} \qquad (1)$$

If the calculated fault coverage meets or exceeds the desired fault coverage, the tool proceeds to write the final BIST stage, block, and testbench, and the top-level design. If the desired fault coverage is not met, then the tool checks to see if the maximum number of patterns has been reached. If this number of patterns is not yet reached, the tool doubles the pattern count and repeats the entire process. By default, the maximum number of patterns is equal to 2^n-2, where *n* is the number of inputs bits to the BIST stage. For a simple feed-forward design with no feedback or state memory, this will practically produce the maximum fault coverage possible with the BIST architecture. For a design that may include a state machine or has internal feedback, additional input patterns could potentially increase fault coverage. If a maximum pattern count parameter is specified in the tool, then this number may be increased to allow repeated iterations of the LFSR patterns presented to the circuit; however, it will be limited to the maximum of the two values because the LFSR must be at least as long as the number of input bits for the BIST stage for the circuit to function. This allows the output length of the LFSR to be larger than the number of input bits; the upper bits of the LFSR are only utilized by the equivalence hardware and do not pass through the DRG or multiplexers to the BIST stage.

If the tool has reached the maximum number of patterns and the desired fault coverage still hasn't been achieved, then the tool checks to see if the maximum number of LFSR seeds has been reached. The LFSR seed is the starting LFSR value. Until this condition check is reached, 0x1 is utilized for the LFSR seed value when the pattern numbers are increased. When the LFSR seed is adjusted, a random number is generated based upon the current number of seeds utilized and the length of input bits to the stage. This random number must be an integer between one and 2^{n}-1, where *n* is the number of output bits of the LFSR. This ensures that a non-zero value is utilized to satisfy the requirements of the LFSR. Like the maximum number of LFSR patterns, the maximum number of seeds is an additional input parameter that may be adjusted; a default of two seeds is utilized to allow for one seed change, which may enable the user to determine if the fault coverage is significantly impacted by the order of patterns presented to the inputs. Both the initial pattern count and initial seed may also be specified in the software. This enables the control of both the starting and ending points of the fault simulation for greater control.

Regardless of the status of the fault coverage once the maximum number of seeds has been reached, the tool proceeds to write the final BIST stage, block, and testbench and the top-level design. These files are considered the primary outputs of the tool, as these files include the design of the area-optimized MTNCL BIST implementation. The obtained fault coverage of the MTNCL BIST design is recorded in the output log; in the event the desired fault coverage was not obtained, the MTNCL BIST design still reaches some level of fault coverage. If this is near the desired fault coverage, the user may find this number acceptable. Alternatively, all the output files may continue to be adjusted for manual tuning by the user, if desired. To aid in this task, the various simulation macros for both digital and fault simulation are maintained to allow for alterations and validation of the results.

4.2 Test-Performance-Optimized Implementation Automation

A flowchart detailing the automation for the test-performance-optimized MTNCL BIST implementation is shown in Figure 28. This algorithm is similar to the algorithm utilized for the area-optimized MTNCL BIST implementation. However, several key differences exist; the pipeline of the MTNCL DUT is parsed and separated to form multiple BIST stages; these BIST stages are iterated over for the digital and fault simulations. Additionally, each BIST stage except for the final BIST stage will utilize a variant of the MTNCL BIST block architecture to reduce component count by reusing the internal completion detection component. The calculation of fault coverage is adjusted, as shown in Eq. 2, to account for all stages in parallel.

The required inputs to the tool are the MTNCL netlist, desired fault coverage, and test-performance-optimized mode selection. Additionally, initial number of test patterns, maximum number of test patterns, initial LFSR seed, and maximum number of LFSR seeds can also be specified as optional inputs; otherwise, they default to 5 initial patterns, 2^n-2 max patterns, where n is the number of BIST stage inputs, a starting LFSR seed of binary 1, and 2 for maximum number of LFSR seeds, such that one additional seed other than the initial binary 1 is tried . The netlist is then parsed as previously detailed. The original MTNCL design is then split into multiple stages where BIST stages are formed from the original MTNCL pipeline structure as shown in Figure 15. As previously mentioned, all BIST stages between the first and last BIST stage include the current pipeline stage's register and combinational logic and the following pipeline stage's slept early completion component. The first and last BIST stages are slightly different – the first BIST stage includes these components in addition to the input slept early completion component, and the last BIST stage includes these components in addition to the output register.

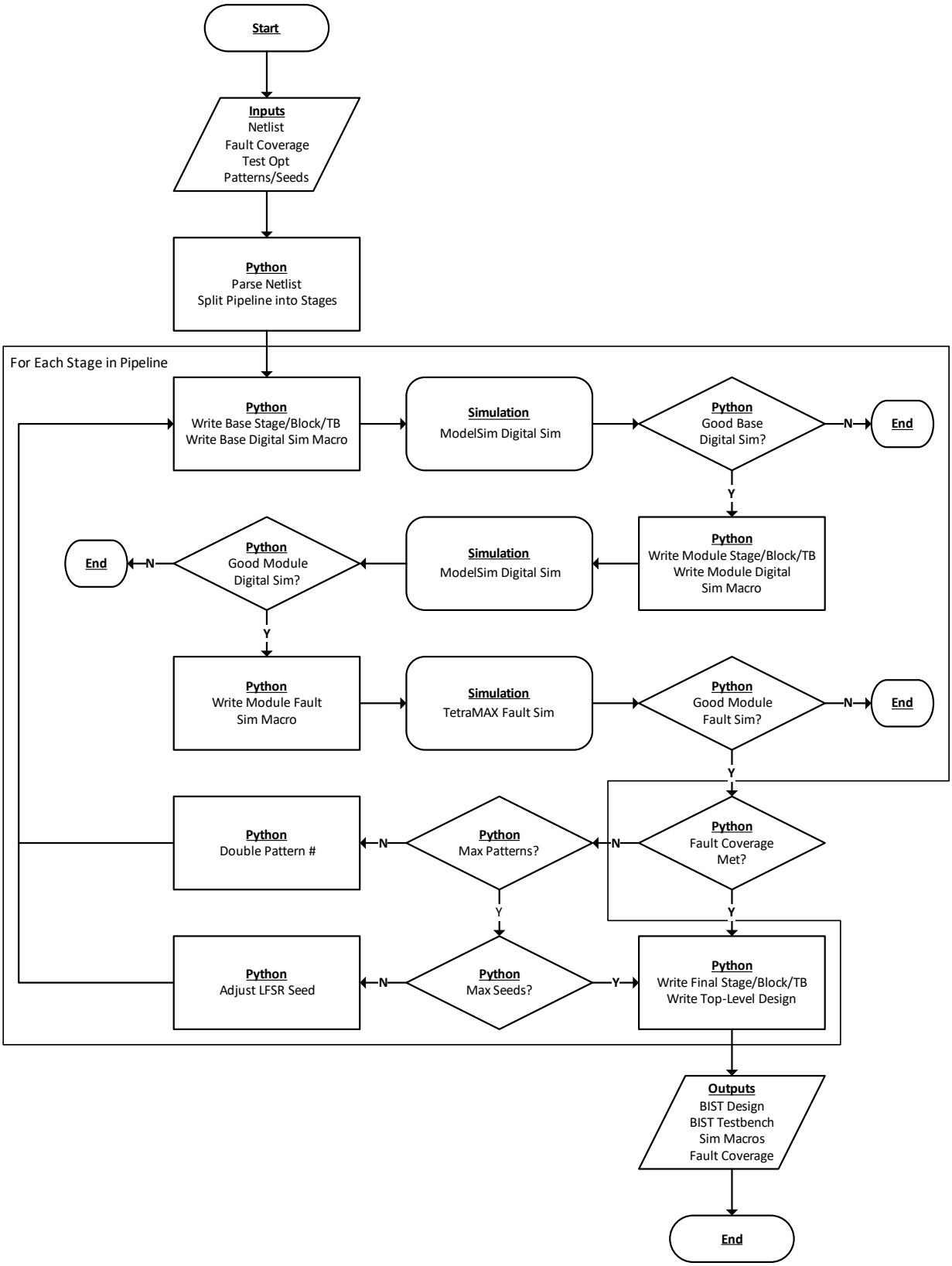

Figure 28. Flowchart for Test-Performance-Optimized MTNCL BIST Implementation Automation.

From the MTNCL pipeline structure in the MTNCL DUT, a base BIST stage, block, and testbench are created along with a macro to enable the digital simulation of this design for each BIST stage in the design. In the test-performance-optimized MTNCL BIST implementation, two varieties of BIST block architectures exist to utilize three types of BIST stages. The final BIST stage includes an output register and thus utilizes the base MTNCL BIST block architecture for BIST stages including output registers, shown in Figure 26.

However, the first and all other BIST stages output combinational logic and a slept early completion component and thus utilize the base MTNCL BIST block architecture for BIST stages including output combinational logic shown in Figure 29. The primary difference in this architecture is that the BIST stage *slpout* signal is inverted once to clock the MISR instead of adding an additional completion tree component with inputs connected to the BIST stage data output. As with the area-optimized MTNCL BIST implementation, the base MTNCL BIST block architecture for BIST stages including output combinational logic, includes a BIST stage acknowledge input from a TH22 threshold gate with inputs coming from the BIST stage *ko* and twice-inverted *slpout* to constrain the system so that only one DATA wavefront is presented to the BIST stage. Essentially, this ensures that the BIST block will not request a NULL wavefront until after a valid DATA wavefront has appeared at the output, propagating from the static DATA input. This acknowledge input is also utilized by the DRG and LFSR gating components to further constrain the system. If multiple DATA transitions occur in a single cycle, as in the case where inputs are provided once requested, the TetraMAX functional simulator may produce invalid results compared to the golden simulation; this results in an invalid fault grading result.

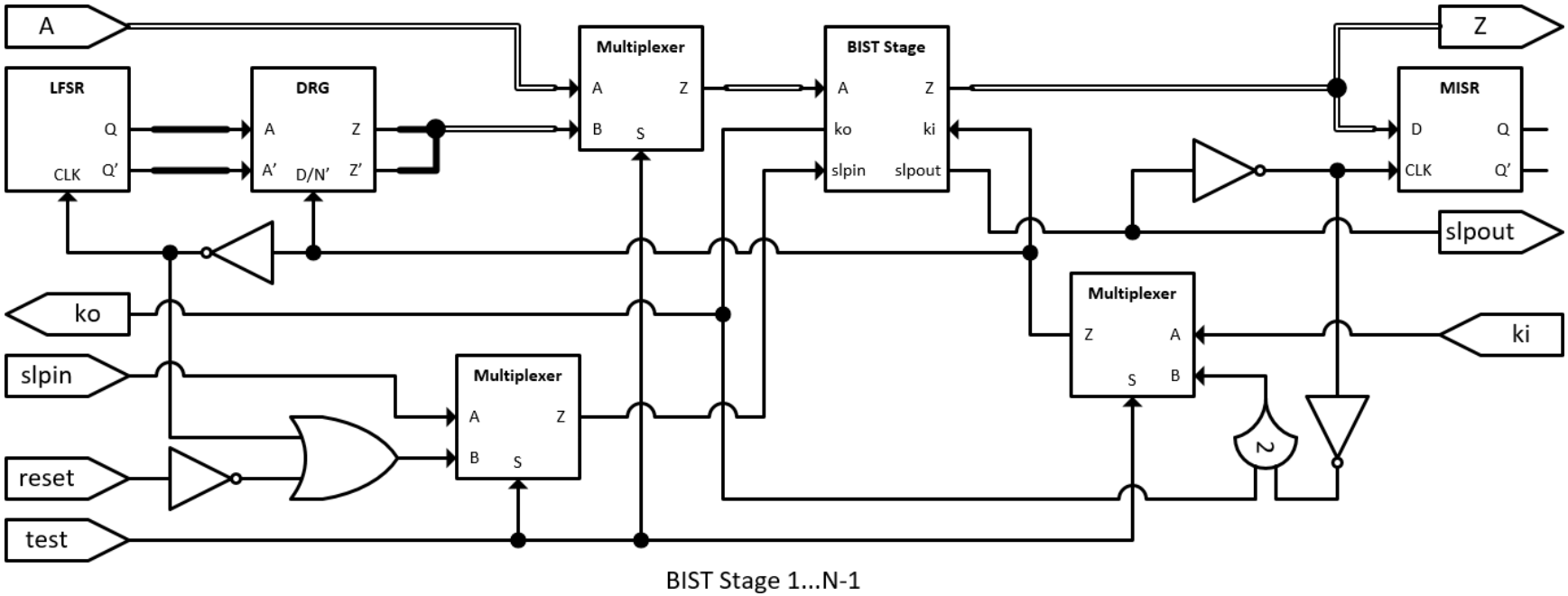

Figure 29. Base MTNCL BIST Block Architecture for BIST Stage with Output Combinational Logic.

The digital simulation macro is then run to digitally simulate the design. The digital simulation output log is parsed and validated to ensure that it ran properly; the LFSR and MISR patterns for the final input and output patterns presented to the BIST stage are included in this log.

The module BIST stage, block, and testbench are then created, including the necessary equivalence modules for the parsed LFSR and MISR patterns, along with a module digital simulation macro. Once again, two types of architectures are utilized with the test-performance-optimized implementation. For the final BIST stage, the module MTNCL BIST block architecture for BIST stage with output register, shown in Figure 27, is utilized. The module MTNCL BIST block architecture for BIST stages with output combinational logic, shown in Figure 30, is utilized for all other BIST stages in the design. Once again, the key difference between these two architectures is that the additional completion tree component is replaced with an inverter as the internal BIST stage slept early completion detection component fulfills the completion detection purpose.

The module BIST block architecture is a fusion of the base and final BIST block architectures; the added combinational logic connected to the LFSR equivalence components prevents the LFSR from being clocked and freezes the DRG output to the NULL state once the LFSR reaches the input pattern following the final desired input to the BIST stage. Additionally, the MISR equivalence component asserts its output once the valid MISR output is produced. The outputs of the two equivalence circuits are ANDed together with the test mode, so that the final status output will assert once the final input pattern has been presented, if the valid MISR output is obtained during test mode.

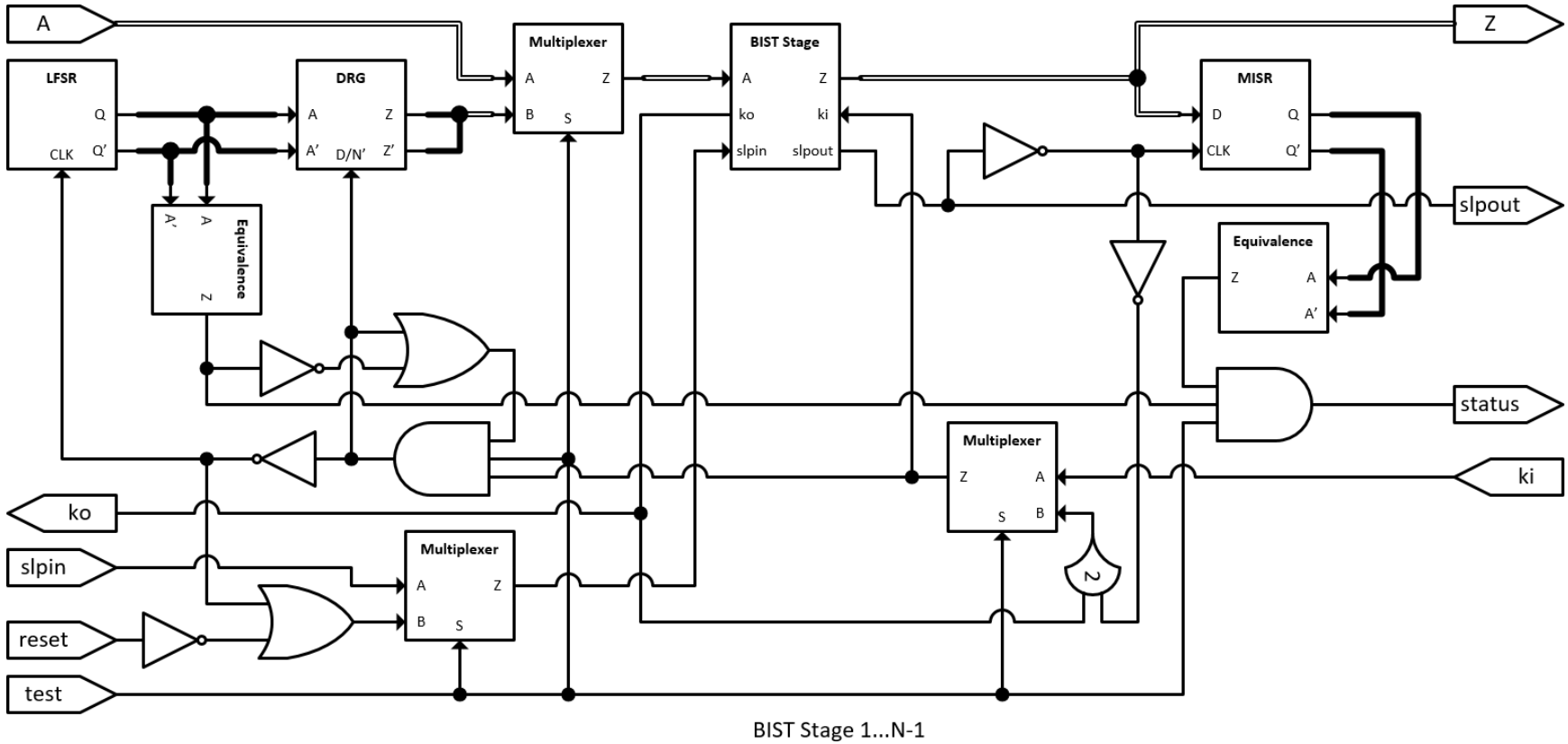

Figure 30. Module MTNCL BIST Block Architecture for BIST Stage with Output Combinational Logic.

The module digital simulation macro created includes measurements to ensure that at the end of the simulation, once all inputs and outputs are static, the LFSR and MISR patterns are equivalent to the base digital simulation, indicating that the equivalence component generation was correct and circuit functionality is valid. Additionally, all ports of the BIST stage are monitored using a vcd dumpports command and output to a VCD file. This creates a file structure that is checked prior to fault simulation to ensure that the digital simulation, referred to as the golden simulation, and functional fault simulation match.

The module digital simulation is run, and the simulation log output is parsed to ensure that the simulation successfully completed. If the LFSR and MISR patterns match the previous digital simulation, then a TetraMAX fault simulation macro is written; this macro reads in the modules and netlists into the simulator. The fault models are built, and a digital simulation is run inside TetraMAX and compared to the ModelSim digital simulation's VCD file. The fault grading simulation is then run, and the fault summary is recorded. It was noted that for large designs, fault simulation may take less time for the test-performance-optimized MTNCL BIST implementation as less memory is necessary because the design size in each simulation is reduced. However, more fault simulations may be necessary because fault simulations must be run for each BIST stage. Once again, the fault simulation output log is parsed to ensure that the fault simulation successfully completed and yielded valid fault information for the BIST stage. This includes the detected faults (DT), possibly detected faults (PT), undetectable faults (UD), ATPG untestable faults (AU), not detected faults (ND), and total faults for each BIST stage.

The above process is repeated for each BIST stage in the design. The fault coverage is then calculated using the same method that TetraMAX uses; the formula is shown below. As PT faults have a 50% chance of being detected in a binary system, they are assigned a weight of 0.5.

However, for the test-performance-optimized MTNCL BIST implementation, the entire design is considered as a single entity; all BIST stages are included together as one complete design. All detected faults are summed together from each BIST stage and added to the half-weighted summation of possibly detected from each BIST stage, and then this value is divided by the summation of the total faults of each BIST stage. This is not output from TetraMAX, although it does output the individual fault coverage of each BIST stage separately.

$$Fault\ Coverage = \frac{\sum Detected\ Faults + 0.5 * \sum Possible\ Detected\ Faults}{\sum Total\ Faults} \qquad (2)$$

If the calculated fault coverage meets or exceeds the desired fault coverage, the tool proceeds to write the final BIST stage, block, and testbench for each BIST stage and the top-level design. If the desired fault coverage is not met, then the tool checks to see if the maximum number of patterns has been reached. This occurs for each of the BIST stages. If this number of patterns is not yet reached, the tool doubles the specific BIST stage pattern count and repeats the entire process. By default, the maximum number of patterns for each BIST stage is equal to 2^n-2, where *n* is the number of input bits to that specific BIST stage. If maximum pattern count parameter is specified in the tool, then this number may be increased to allow repeated iterations of the LFSR patterns presented to the circuit; however, it will be limited to the maximum of the two values because the LFSR must be at least as long as the number of inputs bits for the BIST stage for the circuit to function. This allows the output length of the LFSR to be larger than the number of inputs bits; the upper bits of the LFSR are only utilized by the equivalence hardware and do not pass through the DRG or multiplexers to the BIST stage.

If the tool has reached the maximum number of patterns for a BIST stage, and the desired fault coverage still hasn't been achieved, then the tool checks to see if the maximum number of LFSR seeds has been reached. The LFSR seed is the starting LFSR value. Until this condition

check is reached, 0x1 is utilized for the LFSR seed value while the pattern numbers are increased. When the LFSR seed is adjusted, a random number is generated based upon the current number of seeds utilized and the length of input bits to the stage. This random number must be an integer between one and 2^n-1, where *n* is the number of output bits of the LFSR for the BIST stage. This ensures that a non-zero value is utilized to satisfy the requirements of the LFSR. Like the maximum number of LFSR patterns, the maximum number of seeds is an additional input parameter that may be adjusted; a default of two seeds is utilized to allow for one seed change, which may enable the user to determine if the fault coverage is significantly impacted by the order of patterns presented to the inputs. Both the initial pattern count and initial seed may also be specified in the software. This enables the adjustment of both the starting and ending points of the fault simulation for greater control.

For the test-performance-optimized MTNCL BIST implementation, each BIST stage will continue to simulate in an iterative loop until all BIST stages have reached their maximum pattern and seed count, or if the calculated fault coverage met or exceeded the desired fault coverage. This allows some stages to improve the fault coverage above the desired fault coverage if other BIST stages have a harder time achieving the desired fault coverage due to limited controllability or observability.

Regardless of the status of the fault coverage once the maximum number of seeds has been reached, the tool proceeds to write the final BIST stage, block, and testbench and the top-level design. These files are considered the primary outputs of the tool, as these files include the design of the test-performance-optimized MTNCL BIST implementation. The obtained fault coverage of the MTNCL BIST design is recorded in the output log; in the event the desired fault coverage was not obtained, the MTNCL BIST design still reaches some level of fault coverage.

If this is near the desired fault coverage, the user may find this number acceptable. Alternatively, all the output files may continue to be adjusted for manual tuning by the user, if desired. To aid in this task, the various simulation macros for both digital and fault simulation are maintained to allow for alterations and validation of the results.

5 Experimental Results

5.1 MTNCL Design Preparation

A number of circuit netlists were used to evaluate the developed automation flow. UNCLE [2] was utilized to synthesize the MTNCL circuits from synchronous RTL.

To select the designs for evaluation, the ISCAS '85 combinational logic circuits [10] were selected. These circuits are available as structural Verilog netlists. UNCLE requires a linear pipeline design with a minimum of two C/L stages so that an MTNCL pipeline, such as one shown in Figure 6, may be created. To implement this linear pipeline functionality into these purely combinational circuits, a *clk* input port was added to enable a synchronous design, an input register was added before the combinational logic, and two registers connected in series were added to the output of the combinational logic. The default flow of UNCLE synthesis was adjusted to further balance these pipelines; Synopsys Design Vision was utilized in the UNCLE toolflow, and the synthesis template scripts used by Design Vision within UNCLE were adjusted. The input and output register stages are still required, so the set_dont_retime command was used on all registers that have net names matching the top-level design input or output ports, excluding the clock signal. The register stage immediately following the combinational logic is allowed to be adjusted in the design; the optimize_registers command with a minimum_period_only option was utilized to appropriately balance the pipeline. This would yield a 2-stage pipeline design with input and output registers, and one internal register. Aside

from this optimization, all standard synthesis requirements of UNCLE were left intact to avoid breaking the UNCLE tool during fast MTNCL synthesis.

During UNCLE synthesis, the majority of the ISCAS ‘85 benchmark pipelined combinational logic designs were able to synthesize into functional MTNCL designs. However, some warnings and errors were observed during UNCLE synthesis; these are shown in Table 3. The c17 design had a complete synthesis without any warnings. After synthesis, UNCLE runs a simulation to validate that the design possesses basic functionality and looks for issues within the simulation. Six of the ten synthesis attempts resulted in warnings for input transition values or output capacitive loads during simulation; these warnings are related to a lack of buffering. In MTNCL, the sleep nets of the MTNCL pipeline stages may have large capacitive loads as the number of gates in that pipeline stage increase. For UNCLE fast MTNCL synthesis, the net buffering option must be disabled. However, these designs still properly simulate in a digital simulator such as ModelSim. Additional buffering could be added to the design if slow performance is observed during transistor-level simulation. Three designs failed UNCLE synthesis, as shown in Table 3, since UNCLE detected a non-linear pipeline inside the design. These designs were excluded from the evaluation.

Additionally, a few other designs were evaluated. An 8-bit adder and a 32-bit multiplier were selected as generic circuits. These two designs were implemented as 2-stage pipelines using behavioral RTL. UNCLE was utilized to synthesize the designs into MTNCL. Pipeline balancing was not implemented for these designs. Table 4 provides area measures for all designs, where gates are the number of MTNCL gates in the design, and area is calculated using the UNCLE library’s gate information. Although c6288 is a 16-bit multiplier, the 32-bit multiplier has a slightly smaller area due to optimization enabled by RTL versus the structural c6288 netlist (i.e.,

the 32-bit multiplier uses optimized 4-gate full adders [1], whereas c6288 full adders are decomposed into basic logic functions, such as AND and NOR, each requiring at least 2 MTNCL gates).

Table 3. UNCLE Benchmark Synthesis Summary.

ISCAS '85 Design	UNCLE Status	Notes
c17	Pass	Success
c432	Warning	Input transition value
c499	Warning	Input transition value
c880	Warning	Input transition value, output capacitive load
c1355	Warning	Input transition value
c1908	Warning	Input transition value
c2670	Error	Non-linear pipeline detected
c3540	Error	Non-linear pipeline detected
c5315	Error	Non-linear pipeline detected
c6288	Warning	Input transition value, output capacitive load

Table 4. MTNCL Designs Evaluated with BIST Automation.

Design	Gates	Area
c17	38	153
c432	581	2403
c499	676	2200
c880	1030	4502
c1355	1276	5632
c1908	1018	4471
c6288	5320	23731
adder8	177	621
mult32x32	7588	21523

5.2 General MTNCL BIST Automation Procedure

For both the area-optimized and test-performance-optimized implementations, three test cases of the MTNCL BIST automation were performed for each design. The initial test case had a target fault coverage of 75%, no maximum pattern count (the default), and no additional seeds besides the initial one. The number of input patterns was initialized to 5. This would enable the circuit to utilize the maximum number of patterns for each BIST stage, 2^n-2 patterns, where *n* is the number of input bits to that specific BIST stage, provided the target fault coverage was not reached first. This run was primarily performed to ensure that the fault simulation would complete in a short amount of time.

For the second test case, a target fault coverage of 100% was utilized. The number of patterns was initialized to 5, while the maximum number of input patterns was set to 1E6 patterns. For any BIST stage with fewer than 20 input bits, this would utilize all possible input patterns, with some patterns repeated. For BIST stages with 20 input bits or more, only 1E6 patterns would be tested. Similar to the previous test case, the number of seeds was limited to one and would thus use only the initial seed pattern. This test case was utilized to obtain a maximal fault coverage obtainable by each design within 1E6 patterns and using only the initial seed. This was used to evaluate performance as the number of input patterns increases. The test time increases significantly as the design size and pattern count increases; a maximum pattern count was set to essentially limit the test time required for the large designs, specifically c6288 and mult32x32.

The third test case had a target fault coverage of 100%, a pattern count set to exactly 1000 patterns (i.e., 1000 starting pattern count and 1000 maximum pattern count), and a maximum seed count set to 1000 seeds. The seed was initialized to the default initial seed. The

1000 static pattern count was selected after reviewing results from the other two test cases; most designs converged to a moderately high fault coverage within 1000 patterns. Although this does not run an exhaustive simulation for all possible LFSR seeds, this does give insight into how fault coverage and test time may vary across different seeds; an untested seed could potentially produce better fault coverage.

All designs were able to run until successful completion, either obtaining the desired fault coverage or iterating through all patterns and seeds. For the 75% fault coverage simulations, the desired fault coverage was obtained by all designs. During the 100% fault coverage simulations with pattern count variations, none of the designs were able to obtain target fault coverage, as expected; TetraMAX identified ND faults in all designs. Likewise, none of the designs were able to achieve 100% fault coverage by varying the initial seed using 1000 patterns. The large designs, consisting of thousands of gates, took a significantly longer time to run than the small designs.

5.3 Area-Optimized MTNCL BIST Results

All of the designs were initially run through the area-optimized MTNCL BIST automation tool using the first test case of 75% fault coverage. Since this test case had a maximum pattern count set to 0, only LFSR lengths up to the BIST stage input length would be generated. This would provide the minimum sized BIST hardware required for this automation task; increased pattern counts could yield slightly larger designs as the LFSR length and equivalence circuits would increase in size. The output BIST designs were imported into Synopsys Design Vision, ungrouped, and analyzed for area. Libraries utilized were generated from the information provided in the UNCLE toolset. The default non-BIST designs were also

run through the same procedure so that the BIST insertion area impact could be determined; the comparison is shown in Table 5. These and all further results are arranged by design area.

For the small designs consisting of less than 200 gates, a large area impact was observed of 100% or greater. This occurs because the ratio of BIST stage input bits to logic is rather high, and a significant area is required to generate the additional BIST hardware compared to the combinational logic. For the medium-sized designs of 200-1300 gates, an area impact of approximately 30-40% was generally observed. The c499 design had a significantly larger area overhead due to an increased ratio of BIST stage inputs to actual combinational hardware. For large designs greater than 1300 gates, area overheads ranging from 14% down to 6.4% were realized. Overall, the area overhead is dependent on the ratio of BIST stage input length to combinational logic. For a coarsely pipelined design with few BIST stage inputs, the area overhead will generally be low. Likewise, designs with a small combinational logic to BIST stage inputs ratio will incur large area penalties.

Table 5. Area-Optimized MTNCL BIST Area Comparison.

Design			**Area-Optimized BIST**		
Name	**Gates**	**Area**	**Gates**	**Area**	**% Area Overhead**
c17	38	153	90	355	132.0%
adder8	177	621	336	1245	100.5%
c499	676	2200	1100	3891	76.9%
c432	581	2403	829	3317	38.0%
c1908	1018	4471	1358	5822	30.2%
c880	1030	4502	1520	6370	41.5%
c1355	1276	5632	1700	7323	30.0%
mult32x32	7588	21523	8328	24513	13.9%
c6288	5320	23731	5698	25260	6.4%

The fault coverage results from the area-optimized first test case are shown below in Table 6. In the table, the fault coverage column lists the fault coverages obtained from fault simulation as a percentage. The patterns column indicates the number of input patterns simulated to reach the specific fault coverage. The test time represents the final test time for the circuits to complete the BIST function in the digital simulation nanosecond timescale. The run time is the summation of the base and module digital simulation time as well as the fault simulation time for the current set of fault simulations, in seconds. This includes time for the tools to read in netlists, run the simulations, and produce the output logs, but it excludes any delay from the Python's parsing, file I/O, and process calls to open simulation software. During the short simulations of the 75% test case, file I/O and process calls are actually a large portion of the overall tool time; however, each run was generally in the range of tens to hundreds of seconds, so the additional time is somewhat negligible in the sense that waiting a few extra seconds does not have a large impact; all designs are also similarly affected.

All the area-optimized BIST circuits were able to obtain a fault coverage of at least 75% with input pattern counts of 160 or fewer. In some cases, the obtained fault coverage was significantly higher than the target fault coverage. Most of the run times for each specific pattern count were less than 15 seconds, except for the large designs. For the mult32x32 design, 80 input patterns produced a fault coverage of only 40.88%. This is one case where a different starting seed may have drastically increased the fault coverage; 65 input patterns are required before the output is non-zero due to the initial LFSR seed. However, this simulation did take significantly longer than any of the other simulations, likely due to the large area count and lengthy fault simulations.

Table 6. Area-Optimized MTNCL BIST Results for First Test Case (75% Fault Coverage, 0 Patterns, 1 Seed).

Conditions	Design	Area-Optimized			
	Name	**Fault Coverage [%]**	**Patterns**	**Test Time [ns]**	**Run Time [s]**
75% Fault 0 Patterns 1 Seed	c17	86.168	10	409	12
	adder8	80.338	40	1932	10
	c499	84.120	80	5129	7
	c432	75.287	160	13129	6
	c1908	77.298	80	6249	13
	c880	77.452	160	12009	11
	c1355	77.716	80	6009	8
	mult32x32	91.775	160	20640	501
	c6288	89.390	80	13449	64

The results from the second test case with a target of 100% fault coverage and a maximum pattern count limit of 1E6 patterns are shown in Table 7. None of the designs were able to obtain the target fault coverage of 100% within 1E6 patterns. The maximum fault coverage obtained through any of the 1E6 patterns is shown for each design; in many cases, this fault coverage may be obtained for numerous consecutive iterations as more patterns are presented, so the minimum pattern count wherein this fault coverage was achieved is shown along with the test time and run time for that specific run.

All the designs were able to obtain a fault coverage of at least 86%, which may be an acceptable fault coverage depending upon user requirements. For all but one design, the maximum fault coverage took less than 10 minutes to simulate at that specific scenario. The run time does not represent the summation of all the simulations, which may still be substantial. The largest design in terms of gate count (but not circuit area), mult32x32, took approximately 230

hours to run the digital and fault simulations for 655360 patterns. The only remaining pattern count for this design, 1E6 patterns, took over 346 hours to run while achieving the exact same fault coverage. Thus, large designs using the area-optimized mode could become a burden in terms of both CPU resources and available time for high pattern counts. Tradeoffs of computing run time, circuit area, and fault coverage between area-optimized and test-performance-optimized mode will be discussed in a later section; the test-performance-optimized mode was able to achieve a higher fault coverage while having a significantly reduced test time. Test Time, listed as the time required to perform a complete BIST of the DUT, approximated by the digital simulator, may also be significant to the user. The mult32x32 design required approximately 88 ms to test using the digital models. Depending upon the cost associated with test equipment, the tradeoff of a lower fault coverage may be enticing to reduce testing time and associated costs.

Since none of the designs achieved the desired fault coverage and the maximum fault coverage generally took a moderate test time to obtain, the lowest fault coverage within 2.5% of the maximum fault coverage was identified. The results from this relaxation are shown in Table 8 and demonstrate that although the maximum fault coverage may take a long time to obtain, it may be possible to obtain an acceptable fault coverage within a much shorter timespan by relaxing the requirements. In some cases, the fault coverage penalty was small.

As the fault coverage began to increase for designs with a very large pattern count, substantial time savings may be observed for both the test and run time. For the mult32x32 design, the run time difference to obtain 91.78% fault coverage instead of the 93.40% maximum value was approximately 230 hours; the slightly lower fault coverage only took 8.4 minutes at that specific pattern count. Likewise, the BIST digital simulation would also complete in a significantly faster time, approximately 21 μs instead of 88 ms.

Table 7. Area-Optimized MTNCL BIST Maximum Results for Second Test Case (100% Fault Coverage, 1E6 Patterns, 1 Seed).

Conditions	Design	Area-Optimized			
	Name	**Fault Coverage [%]**	**Patterns**	**Test Time [ns]**	**Run Time [s]**
100% Fault 1E6 Patterns 1 Seed Maximum	c17	86.929	40	1609	4
	adder8	87.956	163840	7999728	191
	c499	86.783	5120	327689	55
	c432	87.492	2560	209929	19
	c1908	86.552	20480	1597449	286
	c880	86.097	40960	3072009	563
	c1355	86.920	5120	384009	118
	mult32x32	93.396	655360	87679426	828165
	c6288	90.424	160	26889	92

Table 8. Area-Optimized MTNCL BIST Relaxed Results for Second Test Case (100% Fault Coverage, 1E6 Patterns, 1 Seed).

Conditions	Design	Area-Optimized			
	Name	**Fault Coverage [%]**	**Patterns**	**Test Time [ns]**	**Run Time [s]**
100% Fault 1E6 Patterns 1 Seed Relaxed	c17	86.168	10	409	4
	adder8	87.017	80	3889	4
	c499	86.563	160	10249	8
	c432	87.424	640	52489	10
	c1908	84.823	640	49929	20
	c880	84.432	640	48009	17
	c1355	84.646	320	24009	15
	mult32x32	91.775	160	20640	506
	c6288	89.390	80	13449	72

The results from the third test case with a target of 100% fault coverage and a maximum seed count of 1000 seeds are shown in Table 9. For these simulations, all designs used a starting and maximum pattern count of 1000, so exactly 1000 sequential input patterns were presented for all the various seeds. This may have resulted in slightly larger BIST structures for small-input-count BIST stages. Like the second test case, none of the designs were able to obtain the target fault coverage of 100% within 1000 seeds. The maximum fault coverage obtained through any of the 1000 seeds is shown for each design.

For most of the designs, the test and run times were very similar across all seeds as the same number of input patterns was presented. For 1000 patterns, the c499 design actually had a maximum fault coverage using the initial seed. Additionally, this design had a slightly lower fault coverage than the maximum fault coverage from the second test case. There were only 1000 patterns simulated here; the second test case required 5120 patterns to obtain the maximum fault coverage. All other designs were able to achieve a higher fault coverage using different seeds than the initial seed used in the second test case. For all other designs except the c17 and c6288 designs, the 1000 patterns also had a significantly reduced pattern count compared to the maximum fault coverage, and thus test time and run time, compared to the second test case while achieving this higher fault coverage. These effects are design dependent, as is the rate of increasing fault coverage as pattern counts increase across various seeds. Trade-offs here may also be evaluated through iterative simulations and analysis of test logs.

Table 9. Area-Optimized MTNCL BIST Maximum Results for Third Test Case (100% Fault Coverage, 1000 Patterns, 1000 Seeds).

Conditions	Design	Area-Optimized			
	Name	**Fault Coverage [%]**	**Seed Count**	**Test Time [ns]**	**Run Time [s]**
100% Fault 1000 Patterns 1000 Seeds Maximum	c17	90.355	120	40009	8
	adder8	88.814	843	48876	9
	c499	86.742	1	64009	19
	c432	88.624	836	82009	14
	c1908	86.679	824	78009	23
	c880	86.226	154	75009	23
	c1355	87.614	940	75009	31
	mult32x32	94.504	721	133790	1183
	c6288	93.250	720	168009	310

5.4 Test-Performance-Optimized MTNCL BIST Results

All the designs were then run through the test-performance-optimized MTNCL BIST automation tool using the first test case of 75% fault coverage. As this provides the minimum sized BIST hardware required for automation, the area of the circuits was analyzed and is shown in Table 10. As additional BIST hardware is required for each pipeline stage in the design, the area impact is more substantial compared to the area-optimized implementation; this is also impacted by the number of inputs to each BIST stage. The area overheads for the test-performance-optimized implementation were a minimum of 27.1% for the c6288 design, the largest-area design, and a maximum of 254.9% for c17, the smallest design. Although this implementation requires a more significant area overhead, especially for smaller circuits, there may be other benefits in terms of circuit performance.

Table 10. Test-Performance-Optimized MTNCL BIST Area Comparison.

Design			**Area-Optimized BIST**		
Name	**Gates**	**Area**	**Gates**	**Area**	**% Area Overhead**
c17	38	153	135	543	254.9%
adder8	177	621	501	1901	206.1%
c499	676	2200	1605	6873	212.4%
c432	581	2403	1406	5584	132.4%
c1908	1018	4471	1906	7985	78.6%
c880	1030	4502	2208	9079	101.7%
c1355	1276	5632	2197	9278	64.7%
mult32x32	7588	21523	9624	29575	37.4%
c6288	5320	23731	6953	30167	27.1%

The fault coverage results from the test-performance-optimized first test case are shown below in Table 11. The fault coverage column lists the fault coverages calculated for all BIST stages and the fault simulation results. The patterns column indicates the maximum number of input patterns simulated to reach the specific fault coverage. As multiple stages were simulated in parallel, this is the maximum of patterns counts for each BIST stage; some stages may be limited to fewer patterns due to BIST stage input lengths. The test time represents maximum final test time among BIST stages for the circuits to complete the BIST function in the digital simulation nanosecond timescale. The run time is the summation of the base and module digital test times as well as the fault test time for the current set of fault simulations, in seconds.

Table 11. Test-Performance-Optimized MTNCL BIST Results for First Test Case (75% Fault Coverage, 0 Patterns, 1 Seed).

Conditions	**Design**	**Area-Optimized**			
	Name	**Fault Coverage [%]**	**Patterns**	**Test Time [ns]**	**Run Time [s]**
75% Fault 0 Patterns 1 Seed	c17	76.519	20	589	8
	adder8	78.636	40	1332	9
	c499	77.771	320	13449	11
	c432	78.883	160	8009	9
	c1908	77.477	320	15689	13
	c880	76.646	160	7369	14
	c1355	78.607	160	7689	13
	mult32x32	92.175	160	12705	588
	c6288	78.619	80	7369	58

All the test-performance-optimized BIST circuits were able to obtain the target fault coverage of 75% with input patterns of 320 patterns or fewer. The obtained fault coverage was less than 79% for most designs; the mult32x32 obtained a high fault coverage of 92.18%. Most of the run times for each specific pattern count were less than 15 seconds except for the large designs.

The results from the second test case with a target of 100% fault coverage and a maximum pattern count limit of 1E6 patterns are shown in Table 12. None of the designs were able to obtain the target fault coverage of 100% within 1E6 patterns. The maximum fault coverage obtained through any of the 1E6 patterns is shown for each design; in many cases, this fault coverage may be obtained for numerous consecutive iterations as more patterns are presented, so the minimum pattern count wherein this fault coverage was achieved is shown along with the test time and run time for that specific run

All the designs were able to obtain a fault coverage of at least 76%, and most fault coverages were greater than 81%. The largest design in terms of gate count (but not circuit area), mult32x32, took approximately 342.6 hours to run the digital and fault simulation for 655360 patterns.

As none of the designs achieved the desired fault coverage and the maximum fault coverage generally took a moderate test time to obtain, the same relaxation was applied to the test-performance-optimized circuits as with the area-optimized circuits; the lowest fault coverage within 2.5% of the maximum fault coverage was identified and is presented in Table 13. Although the maximum fault coverage may take a long time to obtain, it may be possible to obtain a comparable fault coverage within a much shorter timespan by relaxing the requirements.

As the fault coverage began to increase for designs with a very large pattern count, substantial time savings may be observed for both the test and run time. For the mult32x32 design, the run time difference to obtain 94.67% fault coverage instead of the 95.330% maximum value was approximately 342.3 hours; the slightly lower fault coverage only took 17.5 minutes at that specific pattern count. Likewise, the BIST digital simulation would also complete in a significantly faster time, approximately 26 µs instead of 55 ms. Similarly, the c1908 design was able to obtain a fault coverage within a 1% difference of the maximum value using only 2.6% of the run time and 1.6% of the test time.

Table 12. Test-Performance-Optimized MTNCL BIST Maximum Results for Second Test Case (100% Fault Coverage, 1E6 Patterns, 1 Seed).

Conditions	**Design**	**Area-Optimized**			
	Name	**Fault Coverage [%]**	**Patterns**	**Test Time [ns]**	**Run Time [s]**
100% Fault 1E6 Patterns 1 Seed Maximum	c17	76.752	160	4649	8
	adder8	79.539	163840	5542128	224
	c499	84.664	2560	107529	30
	c432	83.237	2560	128009	21
	c1908	81.612	81920	4014089	1090
	c880	82.855	5120	235529	99
	c1355	83.848	5120	245769	99
	mult32x32	95.330	655360	54583533	1233222
	c6288	86.509	5120	471049	1544

Table 13. Test-Performance-Optimized MTNCL BIST Relaxed Results for Second Test Case (100% Fault Coverage, 1E6 Patterns, 1 Seed).

Conditions	**Design**	**Area-Optimized**			
	Name	**Fault Coverage [%]**	**Patterns**	**Test Time [ns]**	**Run Time [s]**
100% Fault 1E6 Patterns 1 Seed Relaxed	c17	76.519	20	589	9
	adder8	78.636	40	1332	8
	c499	83.416	640	26889	15
	c432	83.142	320	16009	13
	c1908	80.879	1280	62729	28
	c880	80.584	320	14729	18
	c1355	81.807	640	30729	22
	mult32x32	94.673	320	26019	1051
	c6288	84.123	640	58889	223

The results from the third test case with a target of 100% fault coverage and a maximum seed count of 1000 seeds are shown in Table 14. For these simulations, all designs used a starting and maximum pattern count of 1000, so exactly 1000 sequential input patterns were presented for all the various seeds. This may have resulted in slightly larger BIST structures for small-input-count BIST stages. Like the second test case, none of the designs were able to obtain the target fault coverage of 100% within 1000 seeds. The maximum fault coverage obtained through any of the 1000 seeds is shown for each design.

For most of the designs, the test and run times were very similar across all seeds as the same number of input patterns was presented. All the test-performance-optimized designs were able to achieve a higher fault coverage with 1000 patterns than the second test case's maximum value using different seeds than the initial seed. For all other designs except the c17 design, the 1000 patterns also had a significantly reduced pattern count compared to the maximum fault coverage, and thus lower test time and run time, compared to the second test case while achieving this higher fault coverage. These effects are design dependent, as is the rate of increasing fault coverage as pattern counts increase across various seeds. Trade-offs here may also be evaluated through iterative simulations and analysis of test logs.

Table 14. Test-Performance-Optimized MTNCL BIST Maximum Results for Third Test Case (100% Fault Coverage, 1E3 Patterns, 1E3 Seeds).

Conditions	Design	Area-Optimized			
	Name	**Fault Coverage [%]**	**Seed Count**	**Test Time [ns]**	**Run Time [s]**
100% Fault 1000 Patterns 1000 Seeds Maximum	c17	80.023	277	29009	15
	adder8	81.670	339	33881	16
	c499	84.814	49	42009	24
	c432	83.637	768	50009	20
	c1908	82.223	520	49009	28
	c880	84.478	630	46009	32
	c1355	84.812	807	48009	31
	mult32x32	95.380	453	83285	2065
	c6288	88.784	688	92009	268

5.5 Comparison of Area-Optimized and Test-Performance-Optimized BIST Results

To better illustrate some of the trade-offs between the area-optimized and test-performance-optimized designs, Table 15 and Table 16 compare the two implementations' area and fault coverage information, respectively. Regarding area, all test-performance-optimized designs required significant additional area as additional BIST structures are required for each pipeline stage in the design instead of only the input BIST stage. For designs with large BIST stage input lengths, this can become substantial. The test-performance-optimized implementation required up to an additional 135.5% for the c499 design; it was as low as 20.7% more area for the c6288 design.

In all cases except for the mult32x32 circuit, the area-optimized implementation was able to achieve a higher fault coverage than the test-performance-optimized implementation. Most incurred a 3-5% reduction in fault coverage. Although controllability increases when using

multiple BIST stages, total number of faults also increases due to additional nodes being evaluated for fault coverage, such as the outputs from the added BIST multiplexers in the data path. For the mult32x32 circuit, increased controllability using an internal BIST stage more than offset the additional UD and ND faults, such that fault coverage was higher for the test-performance-optimized version vs. the area-optimized design. Which one yields higher fault coverage is circuit dependent.

For most of the designs, the test-performance-optimized implementation also reached the maximum fault coverage in a reduced test time. This would yield faster test times for the final circuit. However, it is important to note that the test-performance-optimized fault coverage may not be acceptable when compared to the area-optimized fault coverage. The mult32x32 design was the largest test time difference, wherein the test-performance-optimized implementation reduced the test time by 60 ms while achieving a higher fault coverage. All other tests completed within a test time difference of less than 3 ms. The difference in time may also be significant in comparison; the c6288 design took approximate 16 times longer to achieve its maximum fault coverage for the test-performance-optimized vs. area-optimized implementation. Although a few milliseconds may not seem like a large difference, this may significantly impact overall test time and cost as significant quantities of a design are placed on a wafer, which all need to be tested.

Additionally, the run time required for the simulations varies as well. Many design simulations reaching maximum fault coverage completed within a half hour for the specific run in either direction. For designs that required many runs to obtain this fault coverage, the different in test time could be substantial, especially as the maximum fault coverage was obtained repeatedly for several runs.

Table 15. Area-Optimized and Test-Performance-Optimized MTNCL BIST Area Comparison for Second Test Case (100% Fault Coverage, 1E6 Patterns, 1 Seed).

Design		**Area-Optimized**		**Test-Performance-Optimized**	
Name	**Area**	**Area**	**% Area Overhead**	**Area**	**% Area Overhead**
c17	153	355	132.0%	543	254.9%
adder8	621	1245	100.5%	1901	206.1%
c499	2200	3891	76.9%	6873	212.4%
c432	2403	3317	38.0%	5584	132.4%
c1908	4471	5822	30.2%	7985	78.6%
c880	4502	6370	41.5%	9079	101.7%
c1355	5632	7323	30.0%	9278	64.7%
mult32x32	21523	24513	13.9%	29575	37.4%
c6288	23731	25260	6.4%	30167	27.1%

Table 16. Area-Optimized and Test-Performance-Optimized MTNCL BIST Fault Coverage Comparison for Second Test Case (100% Fault Coverage, 1E6 Patterns, 1 Seed).

Design	**Area-Optimized**			**Test-Performance-Optimized**		
Name	**Fault Coverage [%]**	**Test Time [ns]**	**Run Time [s]**	**Fault Coverage [%]**	**Test Time [ns]**	**Run Time [s]**
c17	86.929	1609	4	76.752	4649	8
adder8	87.956	7999728	191	79.539	5542128	224
c499	86.783	327689	55	84.664	107529	30
c432	87.492	209929	19	83.237	128009	21
c1908	86.552	1597449	286	81.612	4014089	1090
c880	86.097	3072009	563	82.855	235529	99
c1355	86.920	384009	118	83.848	245769	99
mult32x32	93.396	87679426	828165	95.330	54583533	1233222
c6288	90.424	26889	92	86.509	471049	1544

Overall, the merits of the area overhead, fault coverage, test time, and actual time may yield different tradeoffs based on the specific circuit design. Due to this, it is recommended that implementing both the area-optimized and test-performance-optimized implementations for a design may yield the highest potential benefit by enabling the evaluation of trade-offs.

5.6 BIST Automation Performance

To further investigate the performance of the BIST automation, all designs were extensively analyzed in terms of their simulation performance across the second and third test cases. The information for the c1355 design is detailed below, as it was the largest of the medium-sized designs.

The fault coverage vs. pattern count is shown in Figure 31. Both the area-optimized and test-performance-optimized implementations achieved an initial fault coverage around 38%; the area-optimized method was slightly higher. As pattern counts increase, the fault coverage increases at a similar rate and flattens out once the maximum is obtained for both designs.

The fault simulation time vs. pattern count is shown in Figure 32. The fault simulation time measured the time from the first line until the final line of the script and does not include tool start-up time. The initial instability is due to only having a 1-second resolution. As expected, the fault simulation time is reduced for the test-performance-optimized implementation. As the pattern count increases, the time increases logarithmically at approximately the same rate for both designs. To obtain the full sweep of simulations for the second test case, a total time of 34590 seconds was required for the test-performance-optimized version; 45843 seconds were required for the area-optimized version. An increase of over 3 hours is required over the entire simulation sweep.

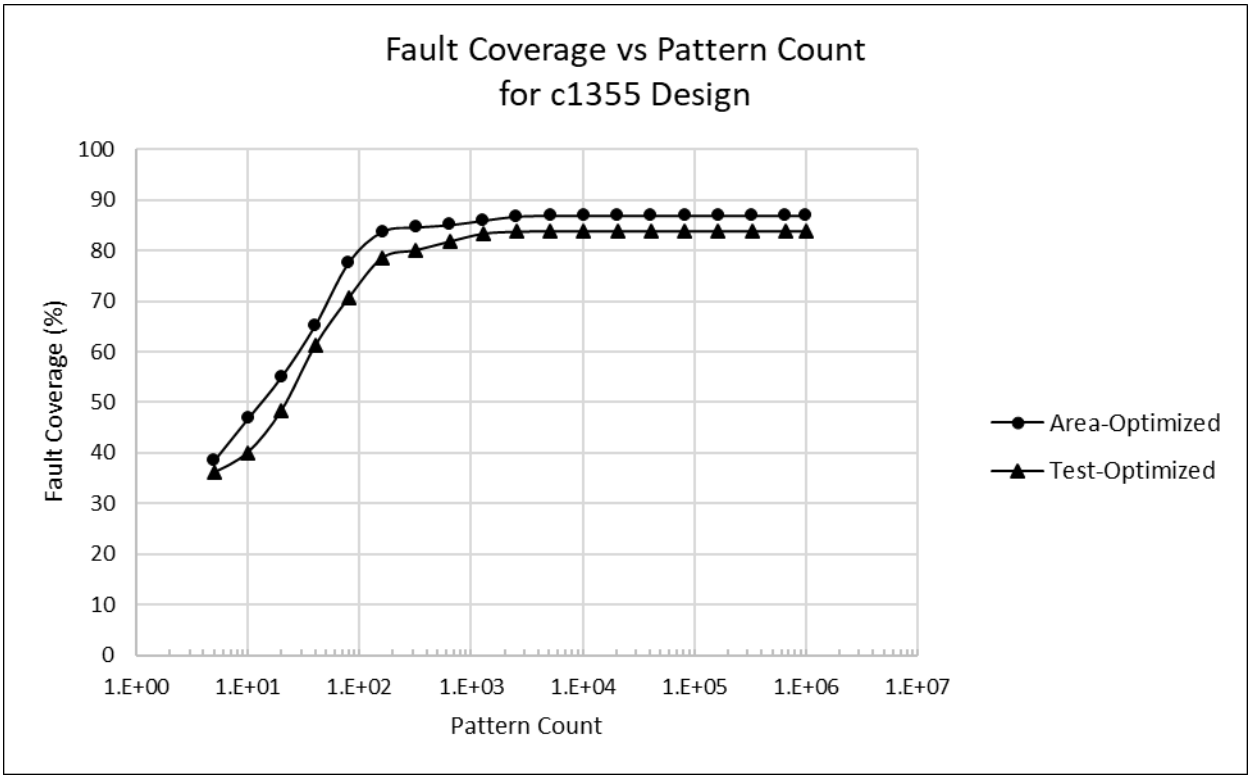

Figure 31. Fault Coverage vs. Pattern Count for c1355 Design.

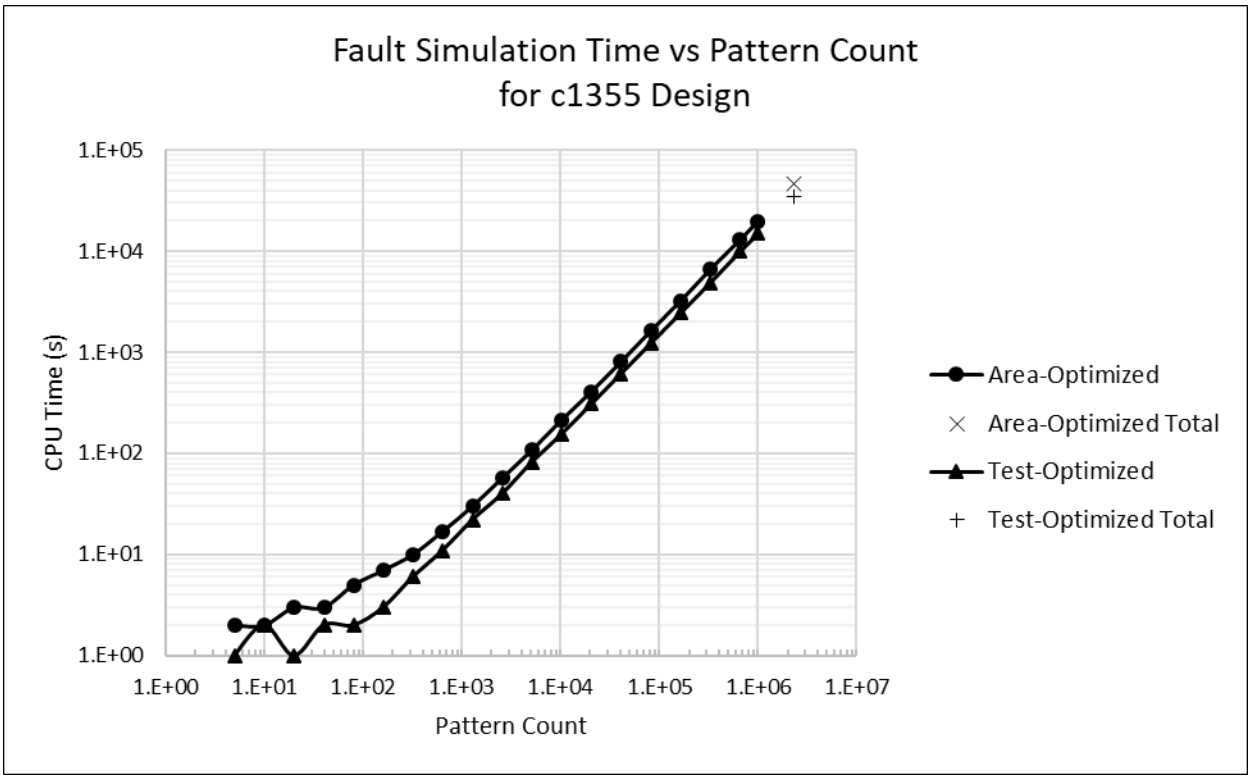

Figure 32. Fault Simulation Time vs. Pattern Count for c1355 Design.

Although the fault simulation time was reduced for the test-performance-optimized version, digital simulations required longer to complete, as shown in Figure 33. The area-optimized digital simulations required 1833 seconds, whereas the test-performance-optimized digital simulations required 3149 seconds. Some of the initial instability may be due to server loading. Although this is a substantial difference between the two designs, the fault simulation total time is an order of magnitude larger. Perhaps this difference in digital simulation totals could be reduced by only reading the gate models once at the beginning of the simulation loop instead of at the start of every simulation. This could affect both implementations; it would affect the test-performance-optimized simulations more because the test-performance-optimized implementation required two reads for all designs due to the two BIST stages in each design.

A plot indicating total run time vs. pattern count is shown in Figure 34. This time was calculated from the automation logs by comparing the time difference between the final fault coverage output of each run; due to this, this time indicates the complete run time per setpoint including any parsing, file I/O, software startup, and digital/fault simulation time, and is a good overall indicator of performance. The five-pattern count value is excluded due to no previous time stamp. For lower pattern counts with this design, the digital simulations took a longer time to complete than the fault simulations. However, the fault simulations began to take more time starting just before 1000 patterns, and then increased faster. Due to this, the test-performance-optimized implementation required additional overall run time to complete, as pattern counts increased. The percentages of total run time for the fault simulation, digital simulation, and other processing accounted for 96.04%, 3.84%, and 0.12%, respectively, for the area-optimized method; for the test-performance-optimized method, this was 91.37%, 8.32%, and 0.31%, respectively. The fault simulation was the dominant time factor in both implementations.

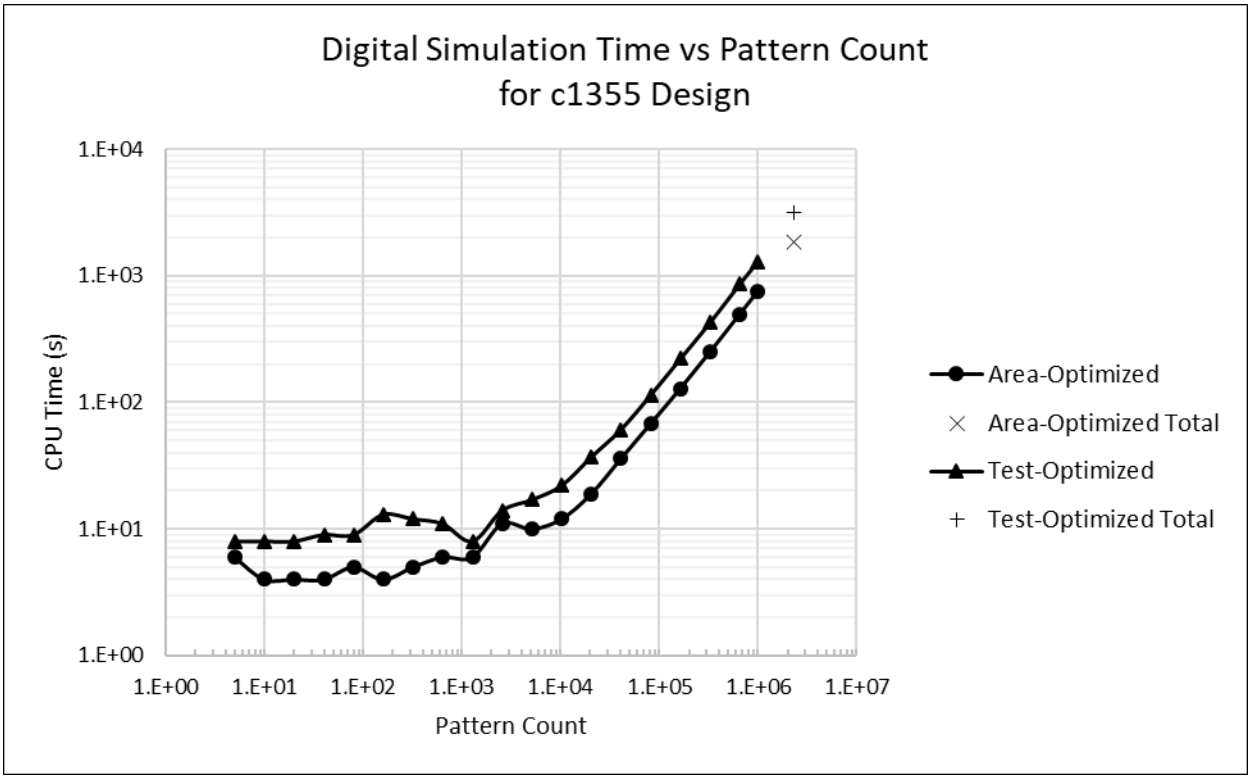

Figure 33. Digital Simulation Time vs. Pattern Count for c1355 Design.

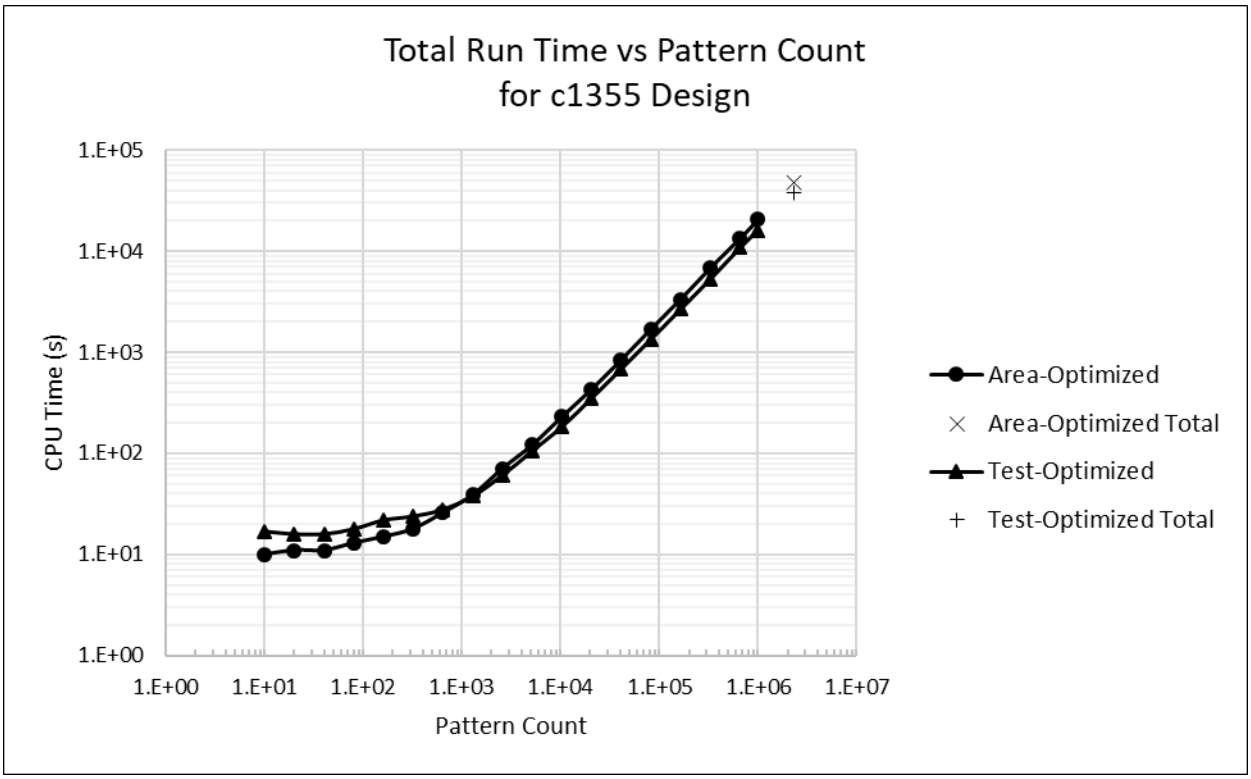

Figure 34. Total Run Time vs. Pattern Count for c1355 Design.

During analysis of the third test case, histograms showing the distributions are utilized to observe differences using a static 1000 pattern count as the starting LFSR seed is adjusted for 1000 different seeds. The fault coverage distributions for the c1355 design are shown in Figure 35 and Figure 36 for the two implementations. The average fault coverage was 85.6% and 83.6% for the area-optimized and test-performance-optimized implementation, respectively. As seen in the second case testing, the area-optimized implementation was able to achieve a higher fault coverage reaching a maximum of 87.6%. The test-performance-optimized method only reached a maximum of 84.8%. Additionally, the minimum fault coverage of the area-optimized version is within 1% of the maximum value for the test-performance-optimized version. Although the distribution of the test-performance-optimized version appears wider, the bins are smaller; the max is within 0.8% of the average as opposed to 2% for the area-optimized version. This may be due to the increased controllability.

The fault simulation time distributions for the c1355 design are shown in Figure 37 and Figure 38 for the two implementations. Two of the area-optimized runs took 30-31 seconds; the plot was adjusted to improve bins. The average fault simulation time was 23.97 seconds and 15.81 seconds for the area-optimized and test-performance-optimized implementation, respectively. The minimum fault simulation time for the area-optimized versions is over 20% longer than the maximum of the test-performance-optimized version. This occurs due to the increased fault coverage simulation requirements for the area-optimized design. Although the overall area is smaller, the fault simulator must simulate the entire design at once; the faults that are harder to determine due to longer paths taking a longer time to simulate. In the test-performance-optimized version, the design is broken up into two simulations and thus has a reduced fault simulation time because controllability is improved, and fault paths are shorter.

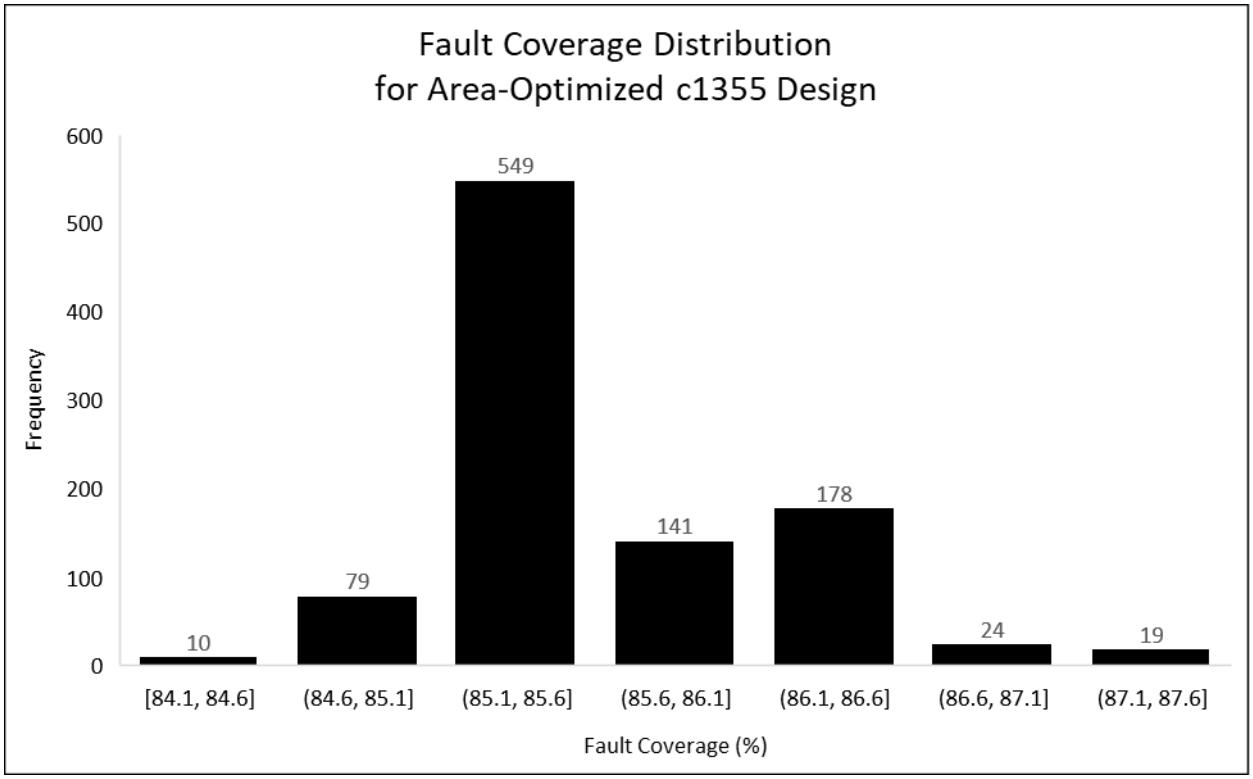

Figure 35. Fault Coverage Distribution for Area-Optimized c1355 Design.

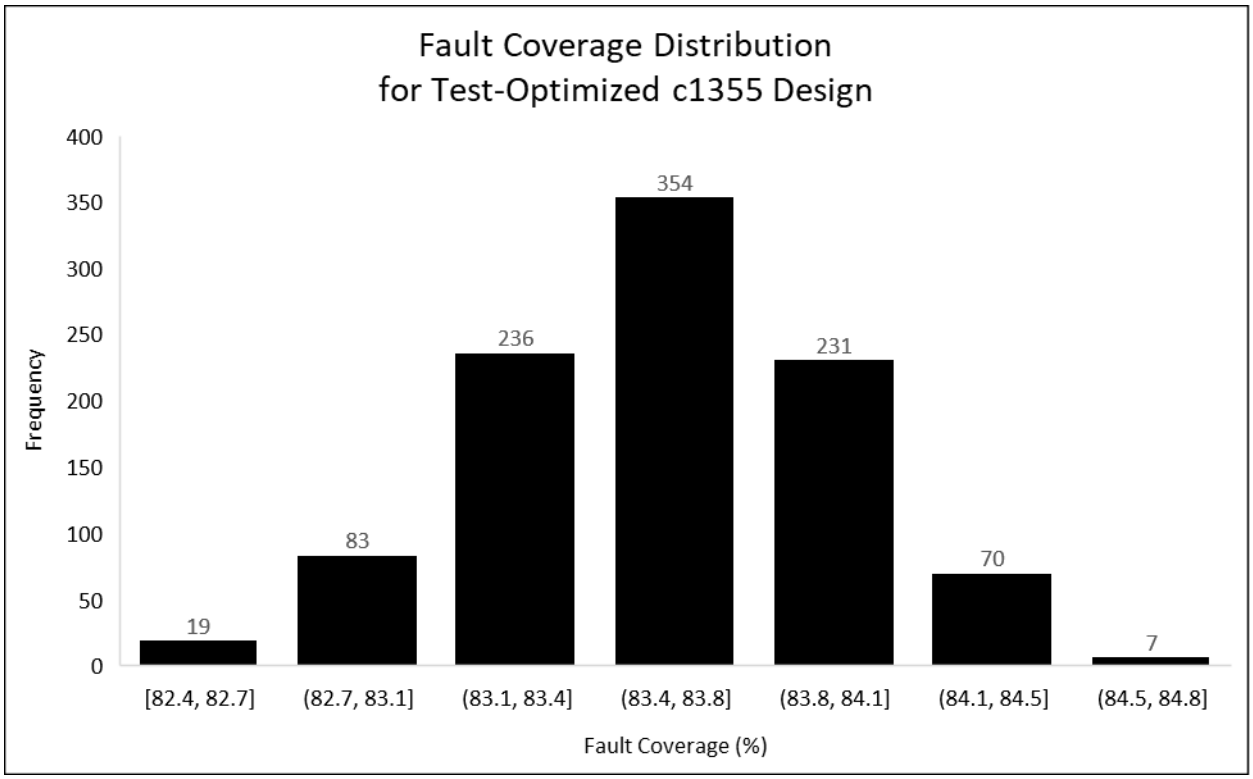

Figure 36. Fault Coverage Distribution for Test-Performance-Optimized c1355 Design.

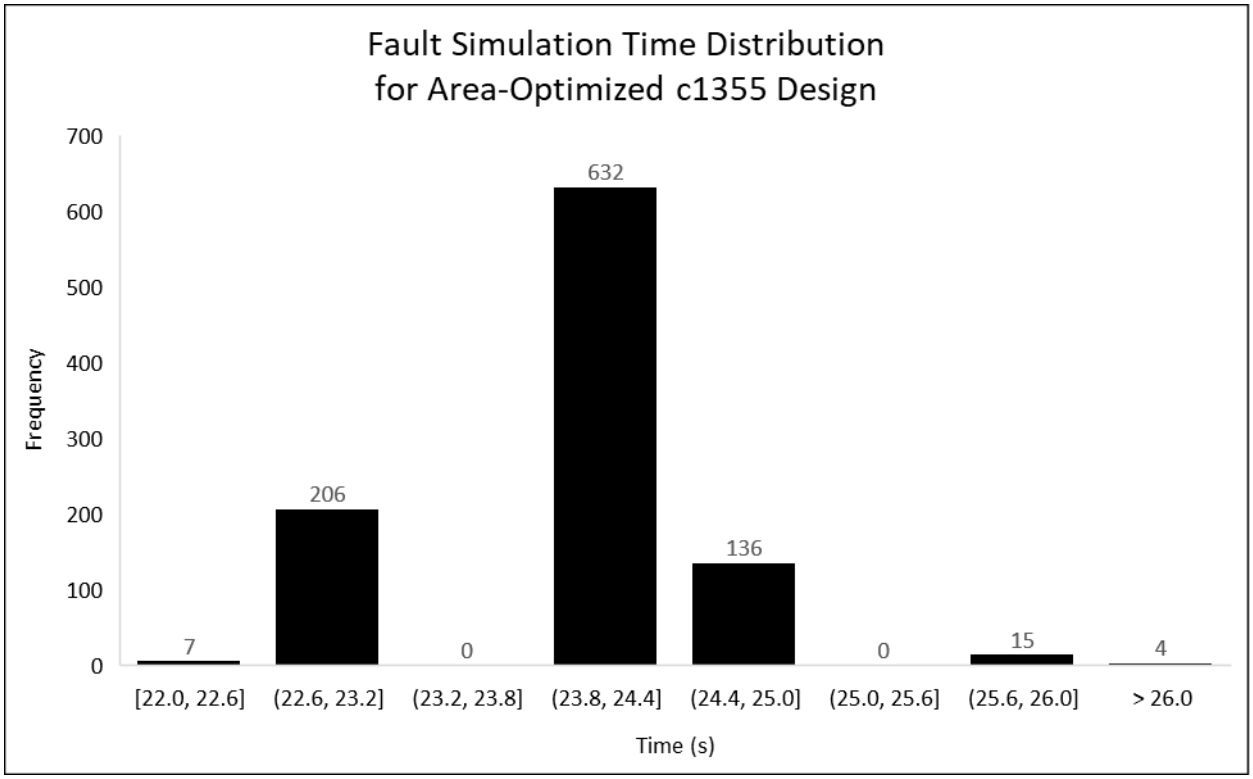

Figure 37. Fault Simulation Time Distribution for Area-Optimized c1355 Design.

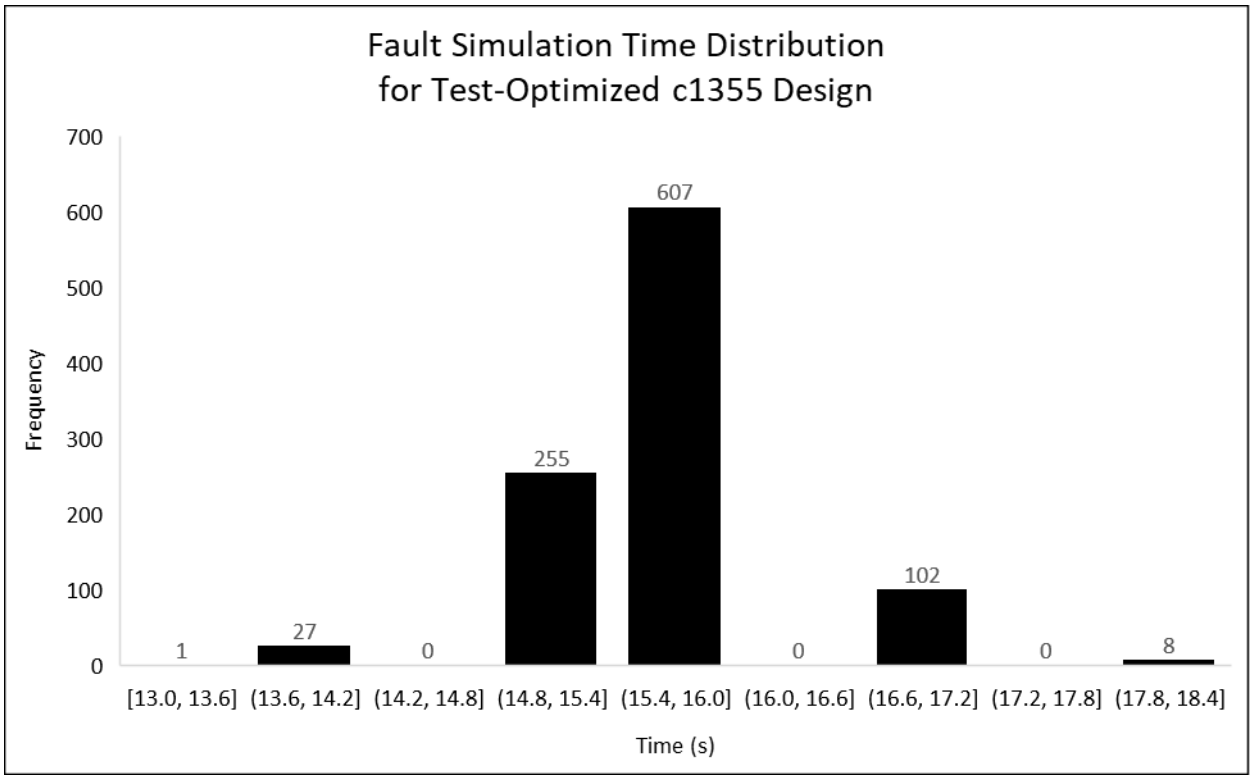

Figure 38. Fault Simulation Time Distribution for Test-Performance-Optimized c1355 Design.

The digital simulation time distributions for the c1355 design are shown in Figure 39 and Figure 40. Unlike the fault simulations, the area-optimized version is significantly shorter than the test-performance-optimized version for all cases. The average digital simulation time was 5.3 seconds and 9.9 seconds for the area-optimized and test-performance-optimized implementation, respectively; this was an 87% increase for the test-performance-optimized version. As previously mentioned, this may be due to re-reading the gate models during the digital simulation of each BIST stage.

The total run time distributions for the c1355 design are shown in Figure 41 and Figure 42. The average digital simulation time was 32.6 seconds and 32.2 seconds for the area-optimized and test-performance-optimized implementation, respectively. This is a small difference and equates to a time savings of approximately 6.7 minutes for the test-performance-optimized design using 1000 patterns over 1000 seeds. The percentages of total run time for the fault simulation, digital simulation, and other processing accounted for 73.59%, 16.20%, and 10.21%, respectively, for the area-optimized method; for the test-performance-optimized method, this was 49.14%, 30.61%, and 20.26%, respectively. Although the fault simulation dominates both designs in terms of percentages, it is a significantly higher portion of the area-optimized version. Time savings are balanced for this design due to the lower digital simulation and processing time. The other processing times are small portions but still can impact the overall simulation time. Opening of the digital and fault simulators is likely a significant portion of this extra time.

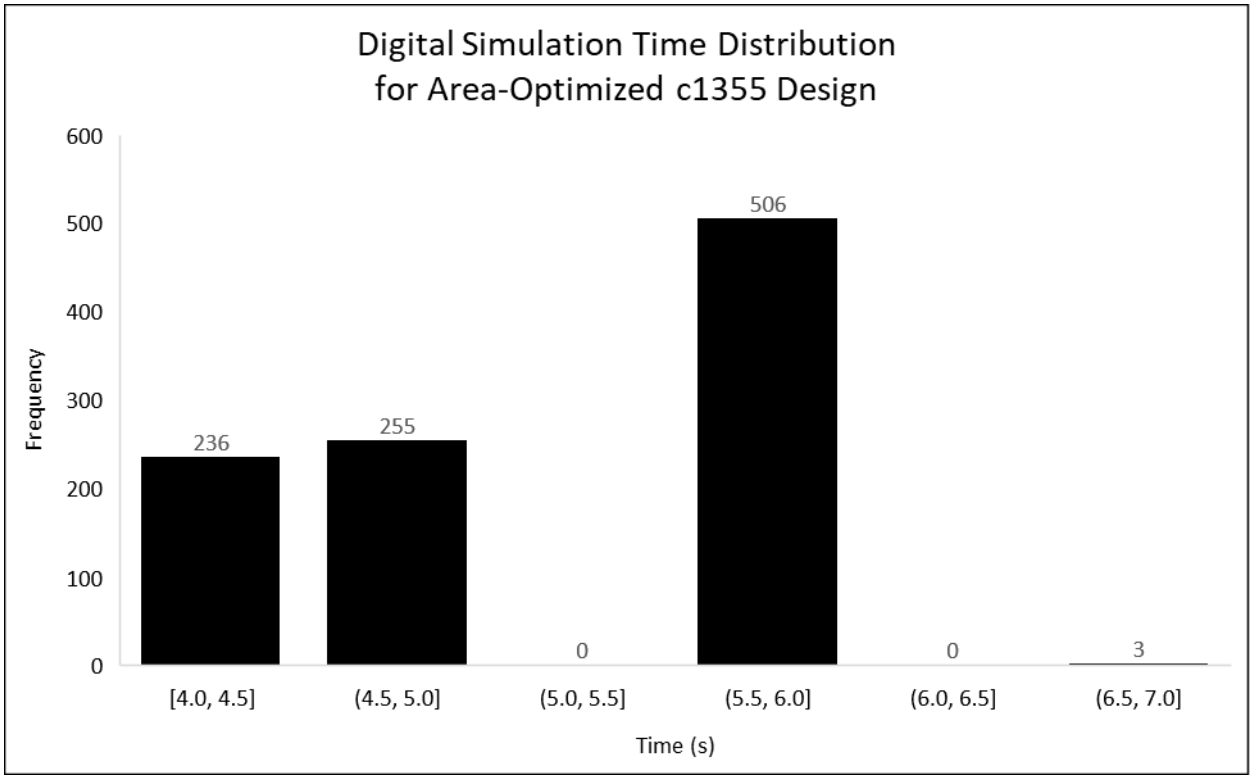

Figure 39. Digital Simulation Time Distribution for Area-Optimized c1355 Design.

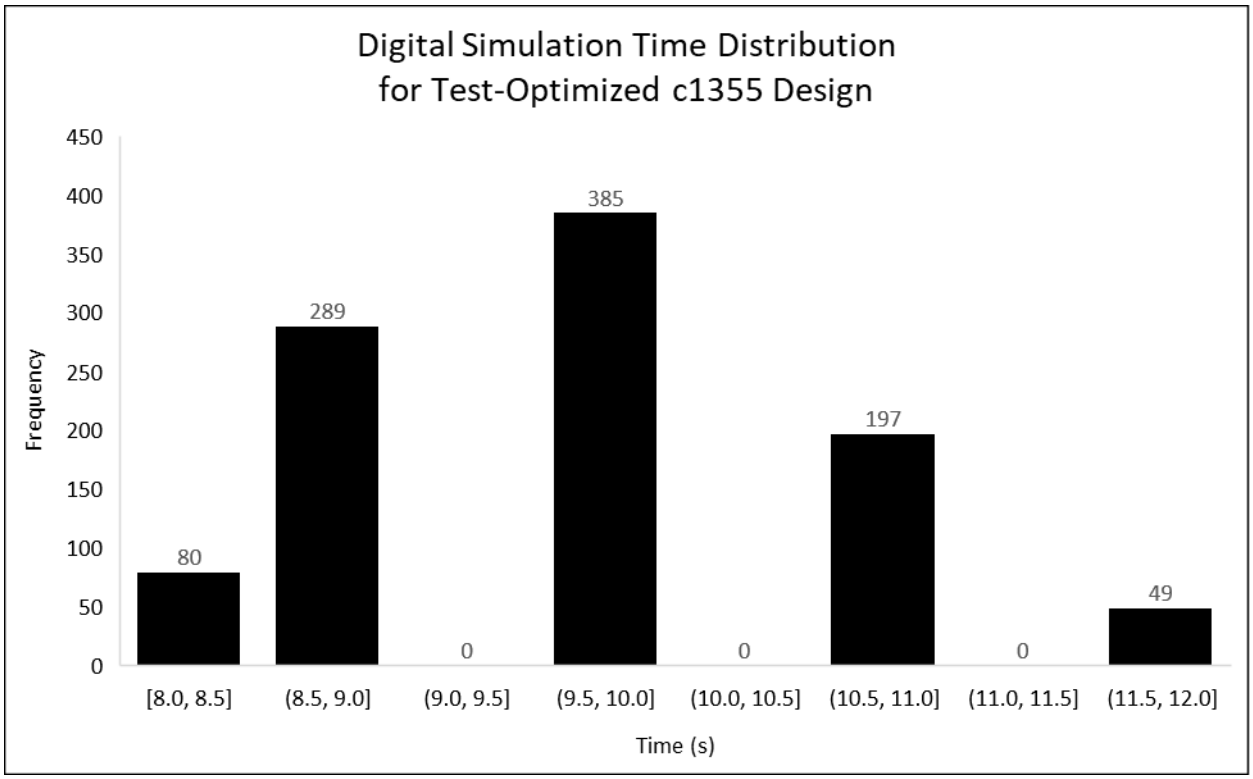

Figure 40. Digital Simulation Time Distribution for Test-Performance-Optimized c1355 Design.

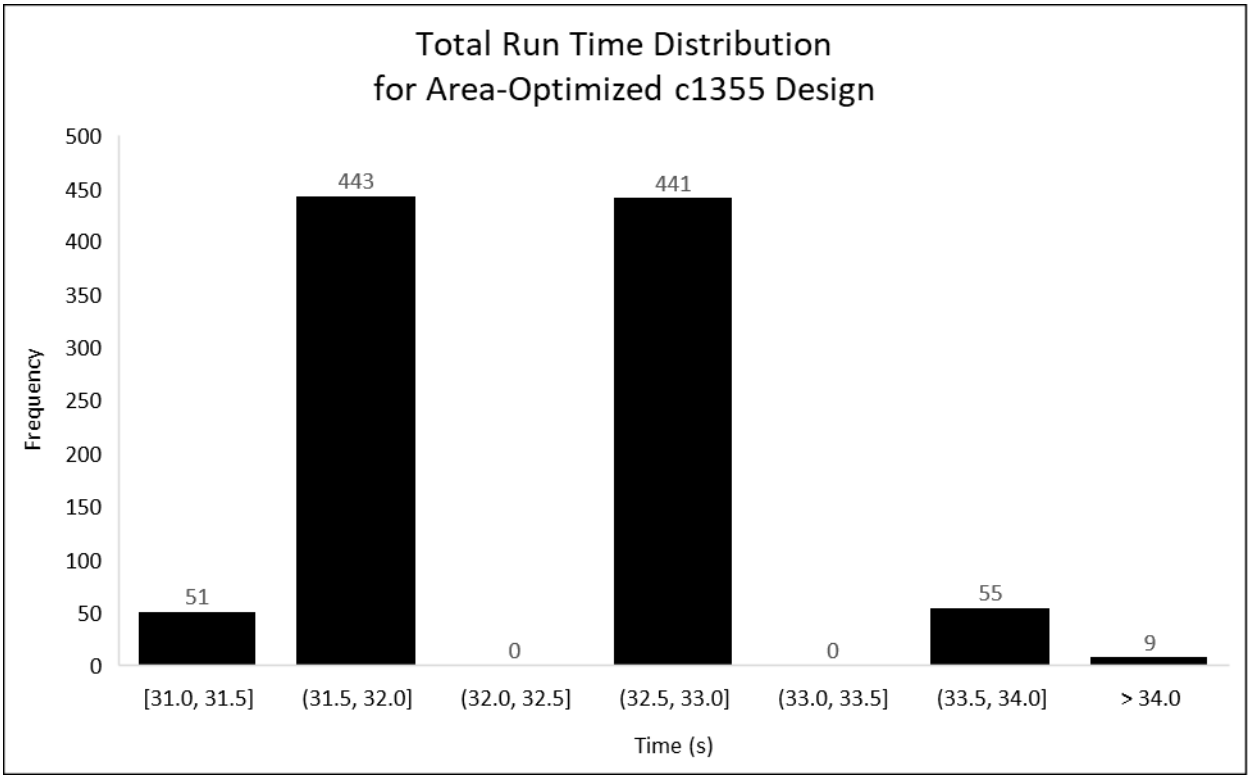

Figure 41. Total Run Time Distribution for Area-Optimized c1355 Design.

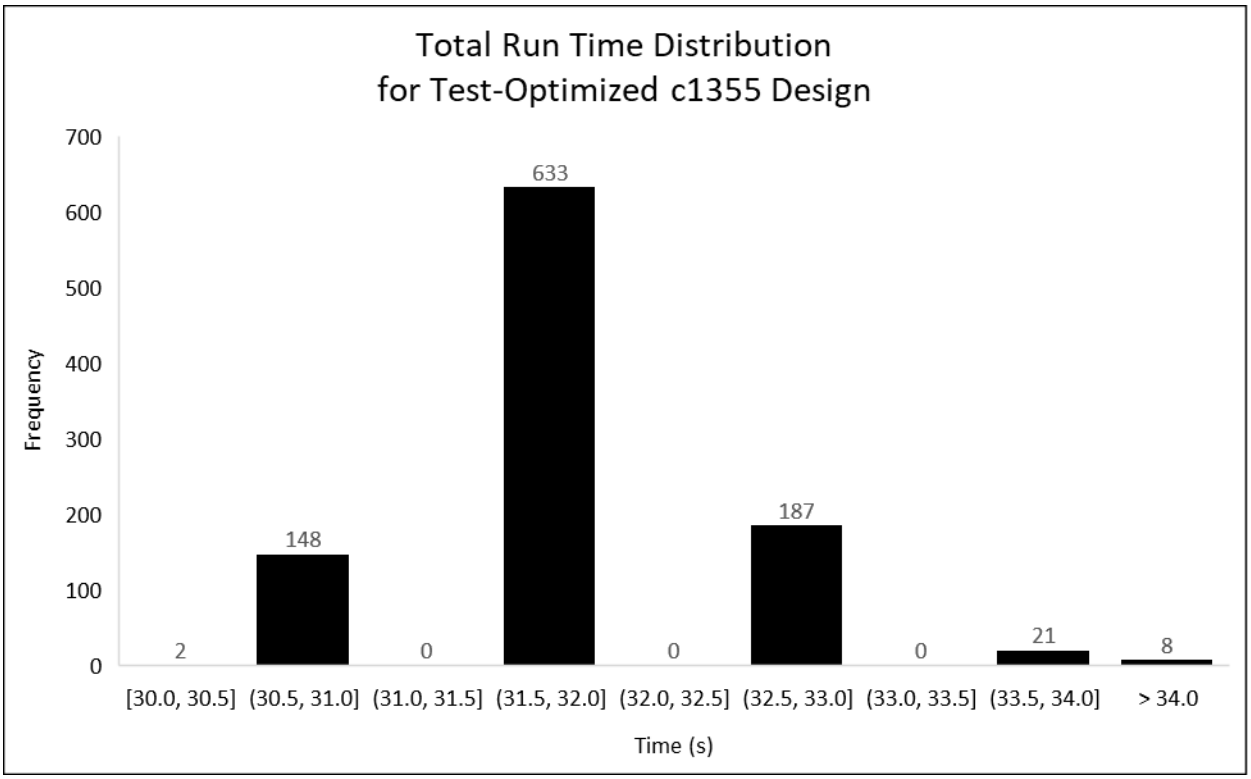

Figure 42. Total Run Time Distribution for Test-Performance-Optimized c1355 Design.

5.7 Transistor-Level Simulation

To confirm that this BIST testing methodology performs as desired using actual transistors and not high-level digital models, transistor-level transient simulations using a 130 nm CMOS process in Cadence Virtuoso were performed on the smallest design, c17. The maximum pattern count was set to 5 so that the waveforms remain legible in the time domain. The digital simulation performed in ModelSim is shown in Figure 43, and the transistor-level simulation using Virtuoso ADE XL with Spectre is shown in Figure 44 for comparison.

Inputs presented to the DUT from the BIST hardware are 0x03E, 0x05D, 0x09B, 0x117, and 0x20F, respectively, in the time domain. The outputs of both designs are 0x3, 0xC, 0x3, 0x3, and 0x9. Once the final input is presented, no further inputs are presented to the DUT as the LFSR is no longer clocked from the other BIST hardware. Additionally, the *lfsr_equiv* and *misr_equiv* signals both rise at the end of the simulation, showing that the desired number of inputs has been provided and that the MISR has reached the correct final value. The *status* signal also rises at the end of the simulation, indicating a good self-test. The *dut_ack_out* signal has a slow transition compared to the rest of the signals; some buffering may be necessary for an actual hardware implementation. This would require some adjustment of the parsing mechanisms to accommodate and thus is not implemented here.

The only main difference is in the actual overall test time; the digital simulation has a slower response than the transistor-level simulation. The models used for digital simulation utilize unit delays; all gates provide one timescale delay for the digital simulation. As the design is asynchronous, this simple timing difference does not affect the overall functionality of the design. The digital models could be updated to provide accurate timings provided by the

transistor-level design gates, but this would require additional control mechanisms during BIST automation to ensure that the proper library models were utilized.

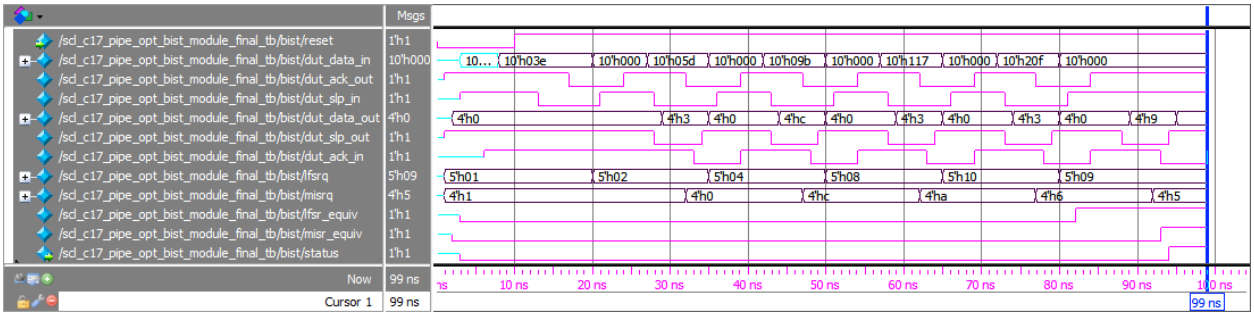

Figure 43. Digital Simulation of c17 Design.

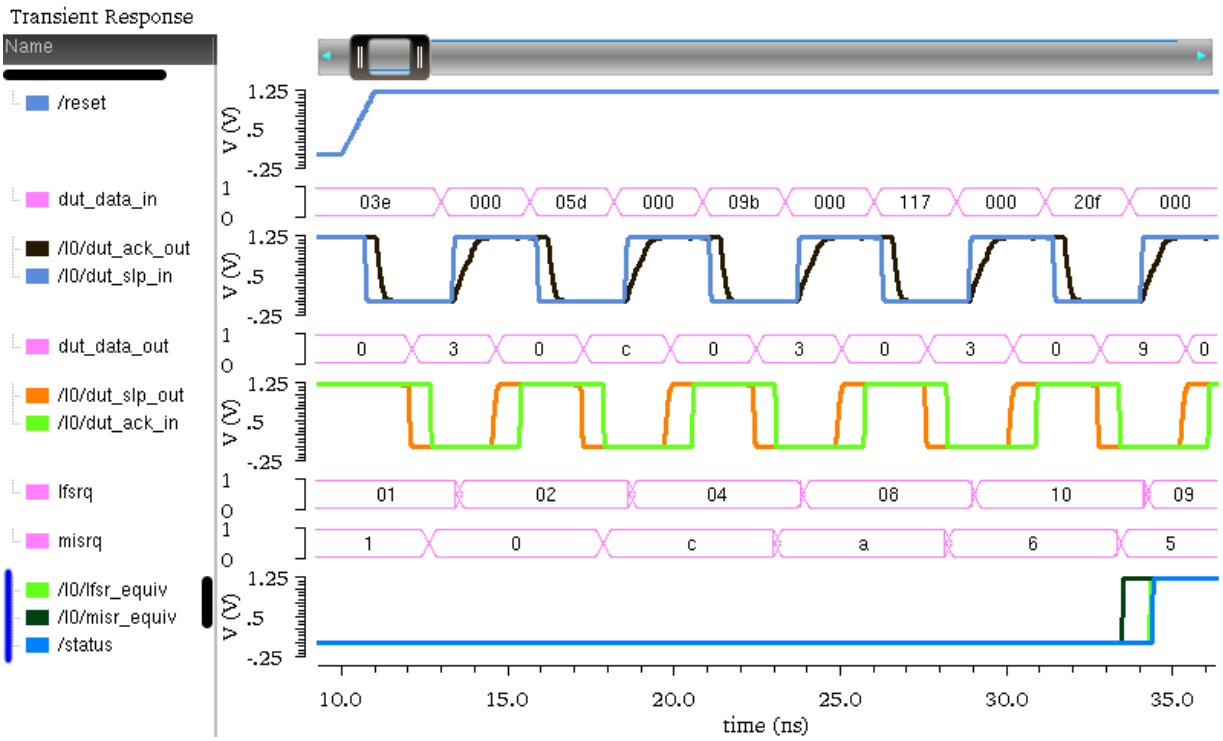

Figure 44. Transistor-Level Simulation of c17 Design.

5.8 FSM and Feedback Compatibility

As Finite State Machines (FSMs) are common in digital design for control circuitry, compatibility with FSMs is required. As UNCLE is incapable of synthesizing designs that include feedback, the c17 design was manually adjusted to include a small pipeline that toggles its output between DATA1 and DATA0 every DATA/NULL cycle and provides this signal as an

input to the final register. The register and acknowledge structures were adjusted to accommodate this. A high-level block diagram to represent this structure is shown in Figure 45.

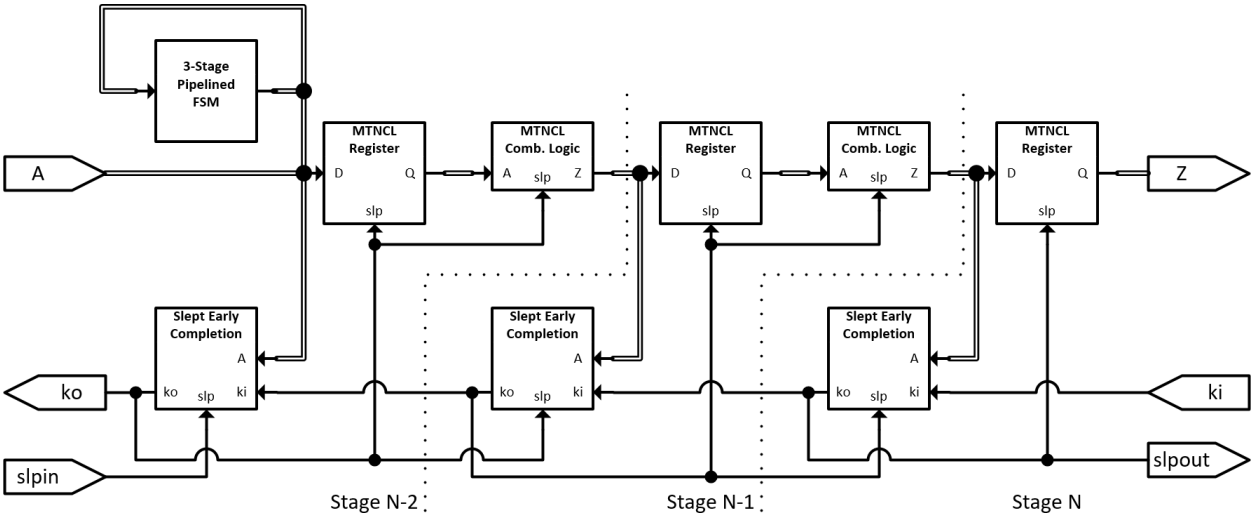

Figure 45. FSM Design Structure Block Diagram.

Some manual adjustments were performed to separate FSMs during parsing, but were not included in the testing previously performed; the delay impact would be minimal as none of the benchmark designs incorporated FSMs, and thus the vast majority of the FSM handling would be skipped.

The BIST automation was run for both the area-optimized and test-performance-optimized methods; the design was capable of digital simulation for each of these. The digital simulation of the scl_c17_fsm design is shown in Figure 46 for the area-optimized method; and Figure 47 and Figure 48 present the digital simulation for the test-performance-optimized BIST stage 0 and BIST stage 1, respectively. The maximum pattern count was set to 5 so that the waveforms remain legible in the time domain. In all the simulations, the status signal rises near the end of the simulation to represent the LFSR reaching its final input pattern and the MISR having a good signature at the end of the simulation. This shows that the actual hardware methodology is capable of supporting BIST where FSM structures are utilized.

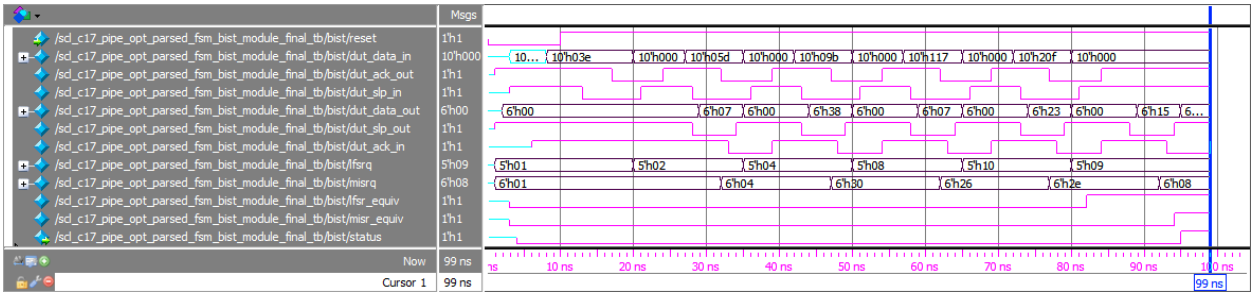

Figure 46. Digital Simulation of scl_c17_fsm Area-Optimized BIST Stage Design.

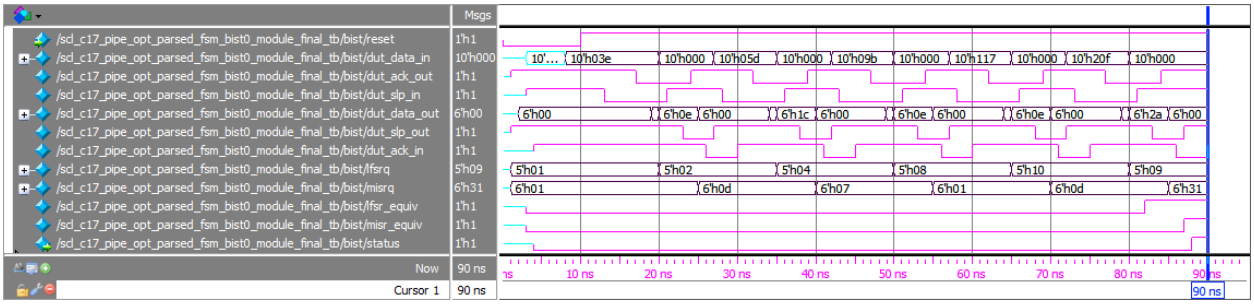

Figure 47. Digital Simulation of scl_c17_fsm Test-Performance-Optimized BIST Stage 0 Design.

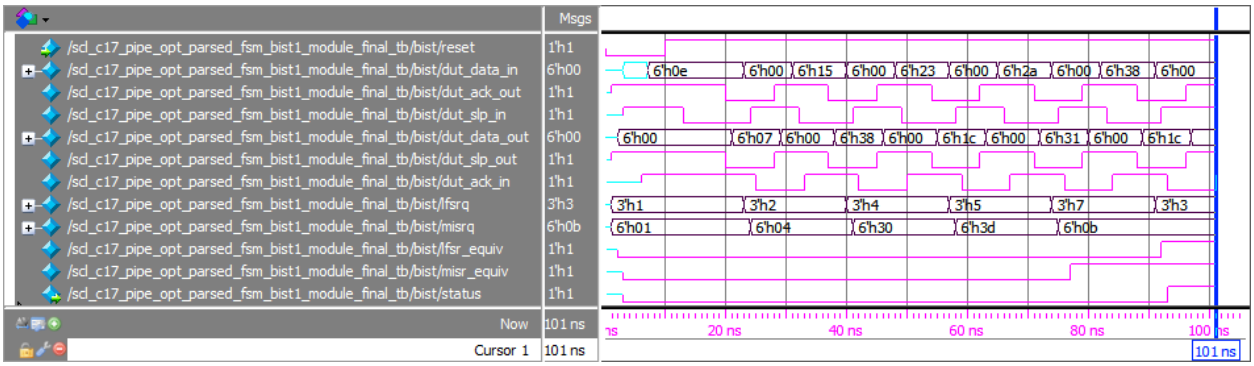

Figure 48. Digital Simulation of scl_c17_fsm Test-Performance-Optimized BIST Stage 1 Design.

Like FSMs, designs incorporating data feedback in the primary data path are common in digital design and must be capable of utilizing BIST functionality. A design with a feedback path from the final output register to the first input register, similar to a multiply and accumulate (MAC) circuit, was implemented to validate this functionality. A high-level block diagram to represent this structure is shown in Figure 49. The first input register was initialized using TH22s and TH22r gates to provide a valid DATA0 wavefront at the start of the simulation, and to provide the required three asynchronous latches in the feedback loop. One input from each gate

was connected directly to one of the final output register's data rails. The remaining input was connected to the following stage's *ko* signal to ensure valid DATA is held until it reaches the next pipeline stage.

Some manual adjustments to parsing were required. This included setting the initialization TH22s and TH22r gates and related completion logic to be utilized by the first stage. As the input register was connected to the output of these initialization gates, the register input nets also had to be removed from the BIST stage input list; these nets are connected internally and do not require an input provided by the LFSR. They are identified as BIST stage inputs during netlist parsing because they are inputs to the pipeline register. The input completion component also required adjustment to determine the correct stage due to register adjustments.

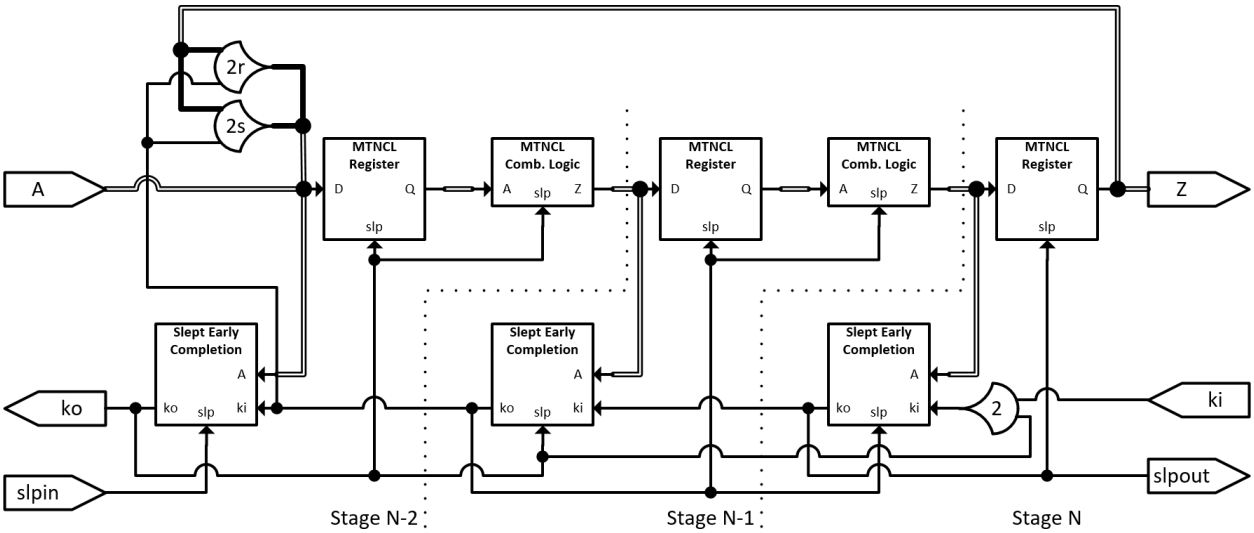

Figure 49. Feedback Design Structure Block Diagram.

The BIST automation was then run for the area-optimized method. As the feedback spans multiple pipeline stages, the test-performance-optimized version is not inherently capable of supporting this type of design. It could be adapted to handle these types of circuits by representing all pipeline stages containing the feedback loop as a single BIST stage. In the tested

design, the feedback loop is the entire pipeline, and this would result in the same circuity as the area-optimized version.

The digital simulation of the simple_fb design is shown in Figure 50 for the area-optimized method. The maximum pattern count was set to 5 so that the waveforms remain legible in the time domain. The *status* signal rises near the end of the simulation, indicating the LFSR reached its final input pattern and the MISR contained a good signature at the end of the simulation. This shows that the area-optimized hardware methodology is capable of supporting BIST where feedback structures in the data path are utilized.

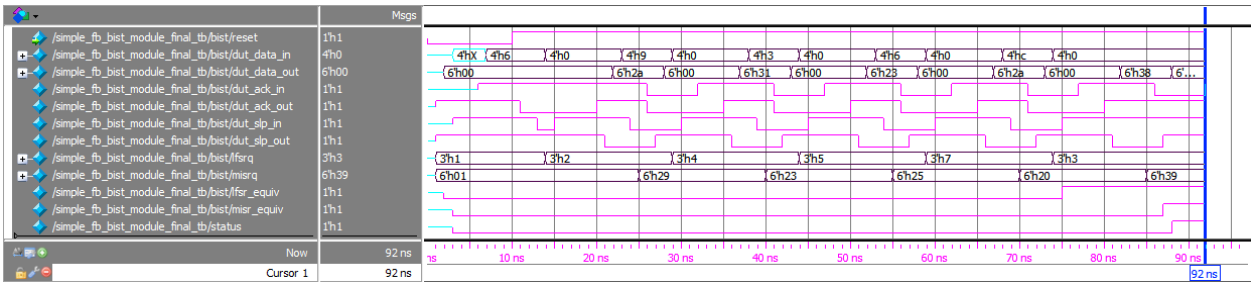

Figure 50. Digital Simulation of simple_fb Area-Optimized BIST Stage Design.

Although the hardware implementations support the digital simulation of these designs, the fault simulation is incapable of directly supporting fault simulation when these FSM and data path feedback structures are incorporated. Since TetraMAX is a cyclic fault simulator, internal datapath feedback (e.g., FSMs, MAC) for MTNCL asynchronous circuits causes significant issues due to multiple internal transitions occurring during a single cycle. Additionally, TetraMAX has issues properly simulating the initialization of specific gates, such as the TH22s, when hysteresis after a reset is required for proper logical evaluation. It was observed that following a reset, the output of a TH22s gate transitioned to low when one of the inputs remained high; the TH22s output should remain high until both inputs are de-asserted. This resulted in no DATA wavefront propagating through the design after circuit reset. Due to these issues, the

TetraMAX sequential simulation does not match the ModelSim digital simulation, and therefore the fault simulation results in an invalid calculation. When this occurs, output occurs on the terminal and in the fault simulation log as shown in Figure 51. It details which patterns the errors occurred on, what the expected value from the digital simulation is, and what the fault sequential simulation obtained. The number of failed patterns is displayed after this command. The automated tool halts and shows a warning if any failed patterns are found in the fault simulation log.

```
run_simulation -sequential
 Begin sequential simulation of 5 external patterns.
    0  data_out[5]  (exp=1, got=0)
    0  data_out[3]  (exp=1, got=0)
    0  data_out[1]  (exp=1, got=0)
    1  data_out[5]  (exp=1, got=0)
    1  data_out[4]  (exp=1, got=0)
    1  data_out[0]  (exp=1, got=0)
    1  slp_out  (exp=0, got=1)
    2  data_out[5]  (exp=1, got=0)
    2  data_out[1]  (exp=1, got=0)
    2  data_out[0]  (exp=1, got=0)
    2  slp_out  (exp=0, got=1)
    3  data_out[5]  (exp=1, got=0)
    3  data_out[3]  (exp=1, got=0)
    3  data_out[1]  (exp=1, got=0)
    3  slp_out  (exp=0, got=1)
    4  data_out[5]  (exp=1, got=0)
    4  data_out[4]  (exp=1, got=0)
    4  data_out[3]  (exp=1, got=0)
    4  slp_out  (exp=0, got=1)
 Simulation completed: #patterns=5/21, #fail_pats=5(0), #failing_meas=19(0), #rejected_pats=0, CPU time=0.00
```

Figure 51. Fault Simulation Output Showing Invalid TetraMAX Sequential Simulation.

Although exact simulation including these types of feedback is not possible, several alternative solutions exist to address these limitations and provide MTNCL BIST functionality. For designs with FSMs, the FSM circuitry may be excluded from the design, and these FSM nets that are utilized by main pipeline logic can be added as BIST stage inputs. Similarly, feedback loops in the data path may be disconnected and set as BIST stage inputs. In both cases, any excluded circuitry must be added back into the design at the top level, and multiplexers must be inserted to include the original desired functionality while adding in BIST functionality.

To validate this solution, the simple_adj_fb design was tested. This design is an exact copy of the simple_fb design but renamed to simple_adj_fb, both for the module and file name. It was separated so both examples would be preserved in the tool. The input completion component required adjustment to determine the correct stage, and the initialization gates were manually removed using the automation tool during parsing. This resulted in the feedback register being controlled by the LFSR, although adding external initialization gates is also possible. As this broke the feedback loop, no additional modification was necessary.

The BIST automation was then run for both the area-optimized and test-performance-optimized methods. The digital simulation of the simple_adj_fb design is shown in Figure 52 for the area-optimized method; and Figure 53 and Figure 54 present the digital simulation for the test-performance-optimized BIST stage 0 and BIST stage 1, respectively. The maximum pattern count was set to 5 so that the waveforms remain legible in the time domain. In all the simulations, the *status* signal rises near the end of the simulation to represent the LFSR reaching its final input pattern and the MISR having a good signature at the end of the simulation. Additionally, the fault sequential simulation properly completes, as shown in Figure 55. No patterns were recognized as discrepancies between the two simulations, so the fault analysis is valid. As this was merely a functional test, only 5 inputs patterns were presented to the design; this resulted in a fault coverage of 72.23%.

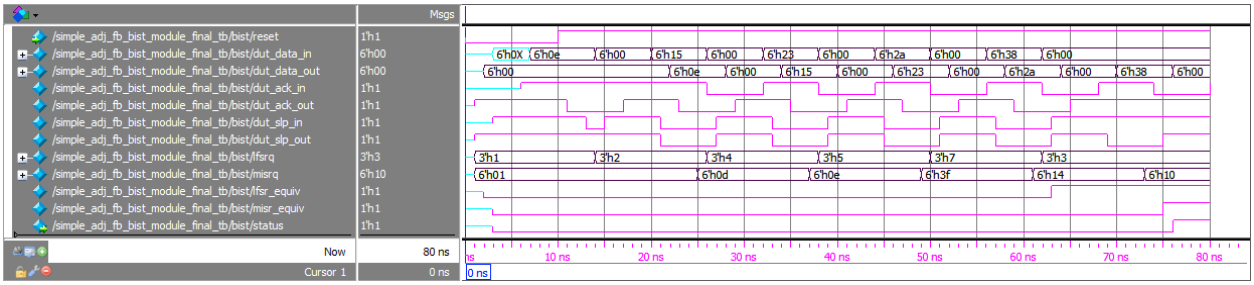

Figure 52. Digital Simulation of simple_adj_fb Area-Optimized BIST Stage Design.

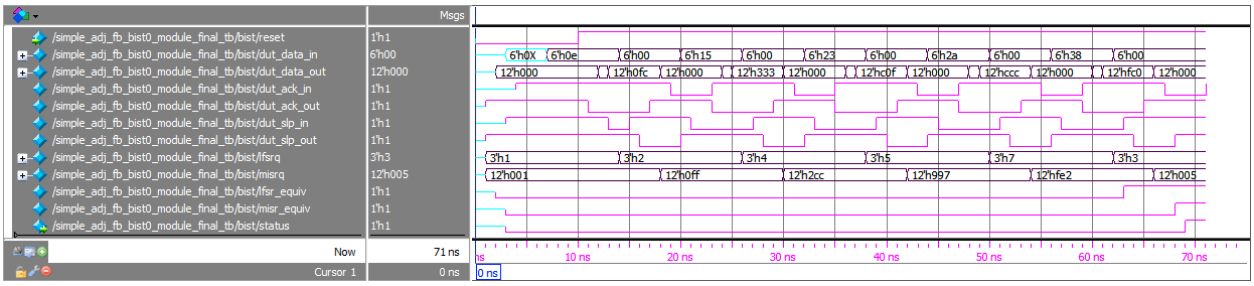

Figure 53. Digital Simulation of simple_adj_fb Test-Performance BIST Stage 0 Design.

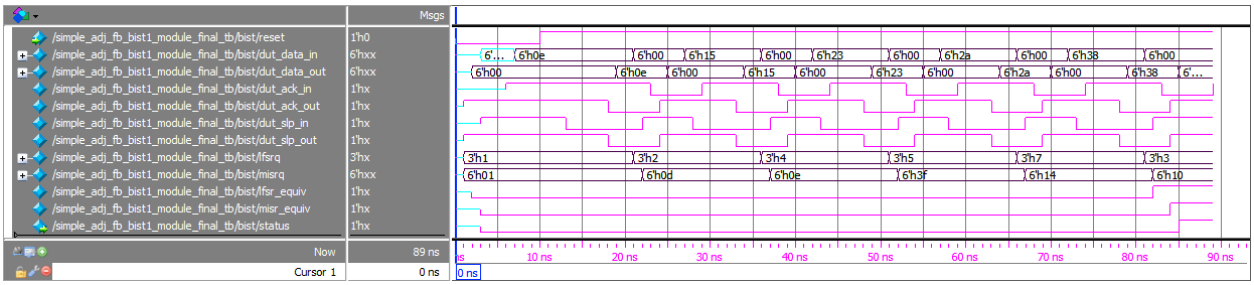

Figure 54. Digital Simulation of simple_adj_fb Test-Performance BIST Stage 1 Design.

```
run_simulation -sequential
 Begin sequential simulation of 5 external patterns.
 Simulation completed: #patterns=5/21, #fail_pats=0(0), #failing_meas=0(0), #rejected_pats=0, CPU time=0.00
```

Figure 55. Fault Simulation Output Showing Valid TetraMAX Sequential Simulation.

It is important to note that both the area-optimized and test-performance-optimized implementations are possible for this specific design, with manual adjustments, because the feedback loop feeds back into the first pipeline stage in the design. If the design includes a feedback loop that directly feeds into any pipeline stage other than the first stage, only the test-performance-optimized is suitable to handle this.

These modifications were completed using the automation tool; FSM structures and feedback loops could be automatically detected in the tool and excluded for BIST purposes. Although excluded in the BIST designs, the structures could be incorporated at the top-level design to preserve the desired functionality. As UNCLE is incapable of exporting designs including feedback, automation of these tasks was not included in the developed BIST tool. Alternatively, asynchronous circuit specific fault simulation software could be implemented to

evaluate the fault coverage. However, this is beyond the scope of this dissertation and conflicts with the goal of using industry-standard software.

5.9 Fault Exclusion Method Based on Operation Principles

As MTNCL circuits operate in an asynchronous fashion with local handshaking, specific combinations of stuck-at fault type and asynchronous gate function can be applied to the TetraMAX verbose fault list to better depict actual fault coverage. For example, if one of the BIST stage sleep nets is stuck-at-1, then that entire stage will always be slept, such that it will never transition to DATA. Likewise, if one of the BIST stage sleep nets is stuck-at-0, that would cause the previous stage's sleep net to be stuck-at-1 (i.e., the previous stage's slept early completion component final TH22 NCL gate, shown in Figure 7, would be stuck-at-0 due its *Ki* input, which is the stuck-at-0 sleep net; and this TH22 NCL gate output is inverted to generate the previous stage sleep net, which would therefore be stuck-at-1), which would cause the circuit to deadlock as mentioned above. Furthermore, any slept early completion component gate output (except for the final inverter, which is already considered in the previous case) that is stuck-at-0 will cause the sleep net generated by that component to be stuck-at-1, which would cause the circuit to deadlock as mentioned above. Since these scenarios would cause the circuit to immediately deadlock, any undetected faults on any of these nets flagged by TetraMAX can be ignored, since they would be immediately detected, in either test mode or normal operation, due to circuit deadlock. A summary of these rules is provided below. Applying these exclusion rules to the c17 circuit increases fault coverage from the original 86.93% to 90.74%.

1. Stuck-at faults on sleep nets can be excluded.
2. Stuck-at-0 faults on slept early completion component gate outputs can be excluded.

5.10 MTNCL Acknowledge Architecture Fault Improvement

Through review of verbose fault lists produced by TetraMAX for several designs, it was determined that many undetectable faults were located in the slept early completion logic. When slept early completion components are designed using MTNCL threshold gates (with sleep input), stuck-at-1 faults can be masked by the sleep mechanism, and therefore cannot be excluded. However, if NCL gates (with hysteresis) are used to implement the early completion logic instead of MTNCL gates (i.e., replace the MTNCL gates in Figure 7 with NCL gates), then any stuck-at-1 fault in this logic will result in a stuck-at-0 fault on its corresponding generated sleep net, and can therefore be excluded as discussed in Section 5.9. A summary of these rules is provided below. Applying these additional exclusion rules to the c17 circuit increases fault coverage from the previous 90.74% to 98.10%. The tradeoff for using this method to increase fault coverage is a decrease in performance and an increase in area, energy/operation, and leakage power, as NCL gates are larger, with increased leakage power and energy per transition, compared to their MTNCL equivalent, and this requires a NULL input to flow through the non-slept early completion logic instead of all gates being simultaneously slept to 0 [5].

3. Stuck-at-1 faults on early completion component gate outputs can be excluded, when designed using NCL gates.
4. Stuck-at-1 faults on any input to an early completion component can be excluded, when designed using NCL gates.

5.11 Controllability and Observability Improvements

Since reported fault coverages are slightly lower than current industry-standard requirements, the ability to add controllability and observability points in asynchronous dual-rail logic was investigated. To improve controllability, the designer must have the capability to inject

a specific signal at a desired net in the circuit to control hard-to-reach faults. In a synchronous design, this is primarily done on a net-by-net basis by inserting C/L or multiplexers to enable internal control of nets during the testing sequence [4]. For MTNCL circuits based upon the proposed BIST topology, this cannot be done for signals in the early completion logic, as injecting signals into this handshaking control logic could adversely affect circuit operation. However, undetectable faults in the early completion logic can be dealt with in other ways, as detailed in the previous two sections.

Controllability points can be added to dual-rail nets to inject a DATA value in order to improve fault coverage, using the hardware shown in Figure 56; a NULL value should not be injected, as this may cause the pipeline to deadlock. Furthermore, each dual-rail signal transitions to NULL after every DATA value, so there would be no need to inject a NULL value. When *Ctrl Sel* is asserted, the *Ctrl* D^0 and *Ctrl* D^1 inputs replace D^0 and D^1 generated by the preceding C/L, respectively, allowing for injection of a DATA0 (D^0 asserted, D^1 de-asserted), a DATA1 (D^0 de-asserted, D^1 asserted), or even an INVALID (both D^0 and D^1 asserted) value, as desired. An INVALID value should only be injected if not part of a feedback loop; otherwise, this could result in perpetual INVALID values in the feedback loop until the circuit is reset. C/L could also be utilized instead of two multiplexers; however, multiplexers offer a higher level of control, with increased area as the tradeoff.

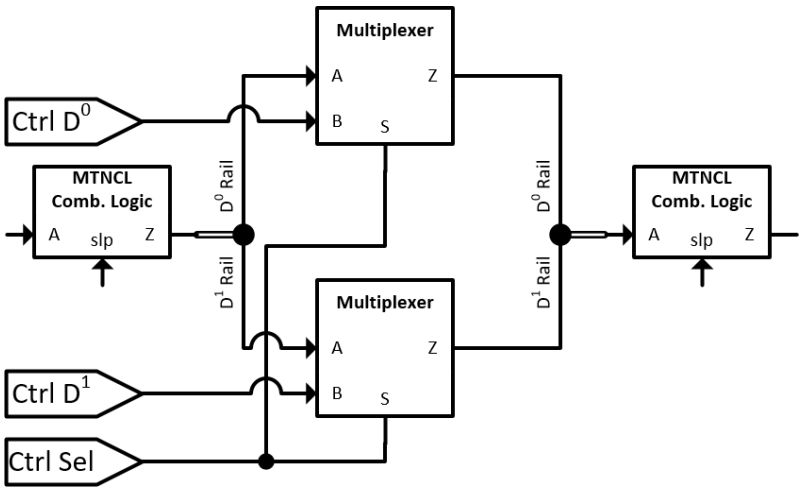

Figure 56. MTNCL Controllability Hardware.

It may also be desirable to increase observability separately, or in addition to controllability techniques. As DATA/NULL wavefronts propagate through a pipeline in the final output design but the MISR only reads from the final pipeline stage, it may not be possible to read the value of internal pipeline nets without modification. To view any signal, the pipeline may be stalled so that a DATA wavefront exists on all pipeline stages. This is exactly the same methodology used to enable the asynchronous fault simulation; instead of utilizing the final designs shown in Figure 16 and Figure 18, the module BIST blocks shown in Figure 27 and Figure 30 could be utilized instead. The additional TH22 gate that ties the BIST stage *ko* and *slpout* or external completion tree component together forces pipeline stalls.

Any net, whether dual-rail, a single rail of a dual-rail signal, or an acknowledge net, inside the design may be probed for improved observability when the pipeline is stalled, as shown in Figure 57. Although the area overhead required to stall the pipeline is small, each additional bit probed requires one additional MISR bit. These observability nets may bypass the external completion tree component if the design has a BIST stage with output register. This is shown in Figure 58; the BIST stage has added *controllability* and *observability* ports.

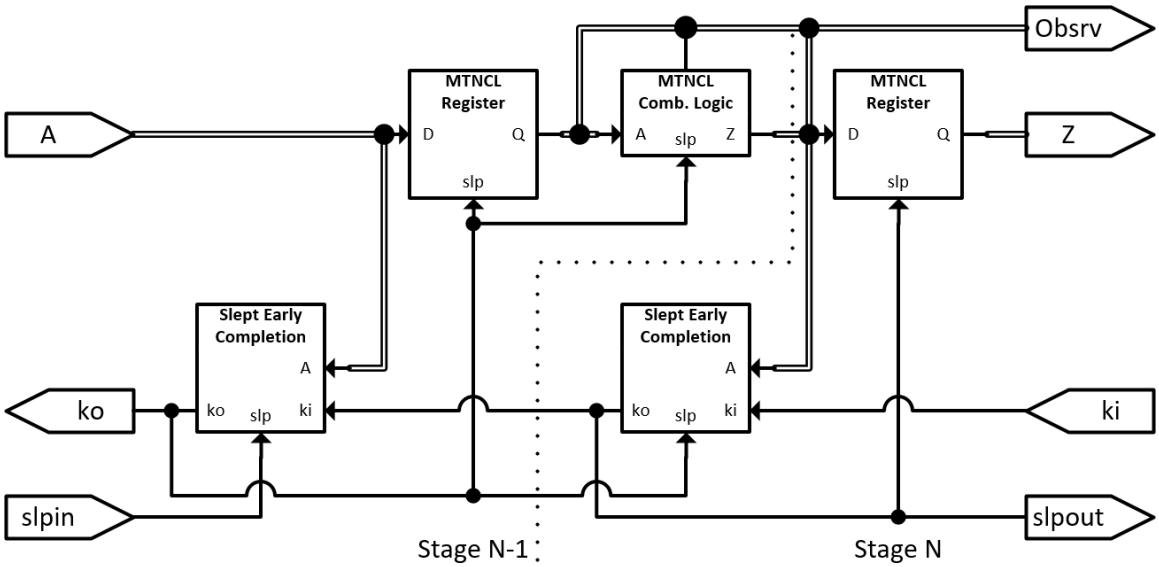

Figure 57. MTNCL Observability Probing with Stalled Pipeline.

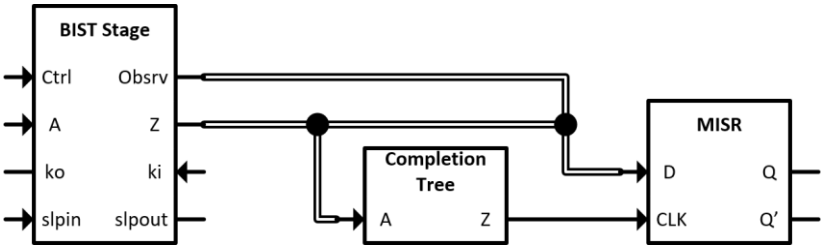

Figure 58. MTNCL Architecture Adjustment for Controllability and Observability Improvement.

The controllability and observability improvements were applied to the c17 design to increase fault coverage further. For observability, three single rail nets from the data pipeline and the output of a subsequent gate were manually added to the *obsrv* output and connected to additional MISR inputs, but not to the completion tree. For controllability, three dual-rail faulty nets had multiplexers inserted in-line along with the associated *ctrl* signals. The original 40 LFSR patterns were presented, as with the original design. Instead of inserting additional hardware for extra control, behavioral control was implemented inside the design testbench by adjusting the *ctrl* signals based on the current input pattern count. For each of the three controllability nets, a DATA0, DATA1, and INVALID wavefront was presented sequentially after the original 40 patterns (e.g., the first controllability net was forced to DATA0 on pattern

41 and DATA1 on pattern 42, and the second controllability net was forced to DATA0 on pattern 44). Therefore, 49 patterns were needed for the first attempt. After seeing some fault coverage improvement based upon the insertion of these, the number of pattern counts was increased to 64 so that the controllability nets would have eight additional wavefronts each wherein a fault may be detected instead of only three, as with the first test using 49 input patterns. The results are shown in Table 17.

Using NCL early completion logic with fault exclusion rules and the controllability and observability improvements, a maximum fault coverage of 98.54% was obtained. This fault coverage is approaching acceptable industry levels. Applying only the controllability and observability methods to the base c17 without any additional fault exclusion methods improved the fault coverage from 86.93% to 91.17%. Similarly, applying the MTNCL early completion fault exclusions results in an increased fault coverage of 94.79%, up from 90.74%. The NCL early completion fault exclusion increase was much less significant, rising only to 98.54% from 98.10%. However, there were only a few faults that could be improved upon, and manual determination of valid controllability states is not a trivial matter. It is worth noting that the addition of controllability hardware does result in an increased number of total faults, area, and test time, as applied. However, this may be required to achieve specific fault coverage targets.

Table 17. Controllability and Observability Fault Coverage Improvements.

	Fault Exclusion Method	Total Faults	Faults	Fault Coverage
c17 original 40 patterns	Original	394	51.5	86.93%
	MTNCL EC	394	36.5	90.74%
	NCL EC	368	7	98.10%
c17 obsrv/ctrl 49 patterns	Original	470	44	90.64%
	MTNCL EC	470	31.5	93.30%
	NCL EC	444	8.5	98.09%
c17 obsrv/ctrl 64 patterns	Original	470	41.5	91.17%
	MTNCL EC	470	24.5	94.79%
	NCL EC	444	6.5	98.54%

6 Conclusion

In this dissertation, a method of BIST for MTNCL circuits was designed, automated, and validated. Two hardware implementations were detailed in Section 3; the area-optimized version required a reduced area while still maintaining testability, whereas the test-performance-optimized version increases controllability, potentially reducing testing time and increasing fault coverage for the actual hardware, at the expense of additional area. Note that the test-performance-optimized version sometimes increased test time and decreased fault coverage due to additional nodes being evaluated, such as the outputs from the added BIST multiplexers; hence, which method is better is circuit dependent. A tool was developed that parses MTNCL circuits, inserts the appropriate BIST hardware, and evaluates possible fault coverages with minimal user input, as explained in Section 4. The results from this tool were analyzed in Section 5, using several circuits as input, to demonstrate that the automation tool is functional and enables the evaluation of tradeoffs for the two BIST implementations. Although not automated, this method is capable of analyzing circuits that include internal feedback through minor manual adjustments that enable it to break any feedback loops, provide LFSR-controlled inputs to these nets, and include the original functionality at the top-level design. Additionally,

several operation-principle-based fault exclusion methods were determined, and controllability and observability improvements were manually implemented to yield higher fault coverages.

7 Future Work

Although this dissertation presented two methods of successful BIST implementation for MTNCL circuits, there are still many topics that can be expanded upon from this work. As many iterative simulations must be performed to obtain the results, potentially across the entire selection of LFSR seeds, a method to calculate the optimal seed pattern for the LFSR could yield reduced run time and actual test time in hardware, by isolating hard-to-detect faults earlier in the BIST automation flow. Additionally, if a specific pattern of inputs can obtain the desired fault coverage with a non-maximal-length LFSR, these could be utilized. Currently, the pattern count only grows in increasingly large steps (i.e., doubles each iteration); hence, once the desired fault coverage is obtained, the pattern count could then be decremented to reduce pattern overhead while maintaining the target fault coverage. This would result in an overall longer run time, as additional simulation iterations would be required. Potentially, other methods of pattern generation may function with the MTNCL design, provided the DRG component is utilized as an asynchronous interface with the pattern generator.

Aliasing verification should be investigated for the MISR outputs. Although the MISR output could have the correct final output, multiple offsetting faults inside the design could potentially produce this valid final signature. Although intensive, it should be possible to calculate this so that the MISR taps could be adjusted to minimize the chance of a faulty circuit producing a valid final signature. A comprehensive study of the trade-offs discussed could be beneficial for test designers to better understand the benefits of the area-optimized vs. test-performance-optimized implementations for various design types. Additionally, if UNCLE was

updated to synthesize sequential MTNCL circuits, the proposed test-performance-optimized method to insert BIST stages so that no BIST stage contains a feedback loop, could be automated, such that the automated MTNCL BIST method would be applicable to both feedforward circuits as well as those containing feedback loops. Currently, some manual manipulation is required, as discussed in Section 5.8.

8 References

[1] S. C. Smith and J. Di, "Designing Asynchronous Circuits using NULL Convention Logic (NCL)," *Synthesis Lectures on Digital Circuits and Systems,* vol. 4, no. 1, p. 96, July 2009.

[2] R. B. Reese, S. C. Smith and M. A. Thornton, "Uncle - An RTL Approach to Asynchronous Design," *IEEE International Symposium on Asynchronous Circuits and Systems*, pp. 65-72, May 2012.

[3] P. Palangpour, "CAD Tools for Synthesis of Sleep Convention Logic," Theses and Dissertations, 2013. [Online]. Available: https://scholarworks.uark.edu/etd/755.

[4] H. T. Nagle, S. C. Roy, C. F. Hawkins, M. G. McNamer and R. R. Fritzemeier, "Design for Testability and Built-In Self Test: A Review," *IEEE Transactions on Industrial Electronics,* vol. 36, no. 2, pp. 129-140, 1989.

[5] L. Zhou, R. Parameswaran, F. A. Parsan, S. C. Smith and J. Di, "Multi-Threshold NULL Convention Logic (MTNCL): An Ultra-Low Power Asynchronous Circuit Design Methodology," *Journal of Low Power Electronics and Applications,* vol. 5, no. 2, pp. 81-100, May 2015.

[6] K. M. Fant and S. A. Brandt, "NULL Convention Logic: A Complete And Consistent Logic For Asynchronous Digital Circuit Synthesis," in *International Conference on Application Specific Systems, Architectures and Processors*, Chicago, 1996.

[7] F. A. Parsan, S. C. Smith and W. K. Al-Assadi, "Design for Testability of Sleep Convention Logic," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 24, no. 2, February 2016.

[8] N. Nemati, P. Beckett, M. C. Reed and K. Fant, "Clock-less DFT-less Test Strategy for Null Convention Logic," *IEEE Transactions on Emerging Topics in Computing,* vol. PP, no. 99, 2016.

[9] N. Nemati, M. C. Reed, K. Fant and P. Beckett, "Asynchronous Interleaved Scan Architecture for On-line Built-in Self-test of Null Convention Logic," *2016 IEEE International Symposium on Circuits and Systems (ISCAS),* pp. 746 - 749, 2016.

[10] P. Fišer, "Collection of Digital Design Benchmarks," Czech Technical University in Prague, [Online]. Available: https://ddd.fit.cvut.cz/prj/Benchmarks/index.php?page=download.