INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI®

University of Alberta

Design and Implementation of a Novel BIST Scheme

for Xilinx XC4000E FPGAs

by

Jian Xu

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirement for the degree of **Master of Science**.

Department of Electrical and Computer Engineering

Edmonton, Alberta

Spring 2002

National Library of Canada

Bibliothèque nationale du Canada

Acquisitions and Bibliographic Services

Acquisitions et services bibliographiques

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-69780-0

Canada

University of Alberta

Library Release Form

Name of Author: Jian Xu

Title of Thesis: Design and Implementation of a Novel BIST Scheme for Xilinx XC4000E FPGAs

Degree: Master of Science

Year this Degree Granted: 2002

Permission is hereby granted to the University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves all other publication and other rights in association with the copyright in the thesis, and except hereinbefore provided, neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatever without the author's written permission.

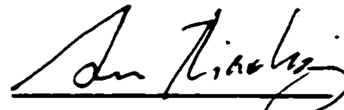
Jian Xu
405, 9323-105 Avenue
Edmonton, AB
Canada, T5H 0J7

Date: December 17, 2001

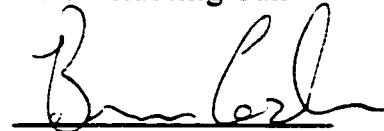
University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **Design and Implementation of a Novel BIST Scheme for Xilinx XC4000E FPGAs** submitted by **Jian Xu** in partial fulfillment of the requirements for the degree of Master of Science.



Dr. Xiaoling Sun



Dr. Bruce Cockburn

Dr. Mrinal Mandal

Dr. Martin Mueller

Date: December 17, 2001

Abstract

This thesis presents novel *built-in self-test* (BIST) schemes for testing *configurable logic blocks* (CLBs) and interconnect in Xilinx XC4000E SRAM-based *field programmable gate arrays* (FPGAs). The minimum number of test configurations for CLBs and interconnect are derived.

The proposed BIST scheme for CLBs includes the testing of FPGA *carry logic modules* (CLMs) for the first time. A systematic method is proposed for deriving the minimum number of testing configurations for CLMs, and the testing for the remaining CLB resources is integrated with the CLM test configurations.

One novel technique, functional test of D flip-flops, is proposed in the BIST scheme for the sequential part of FPGA interconnect. It can be combined with another novel technique from Dr. Sun, error-control coding for the combinational part of FPGA interconnect, to provide superior multiple fault coverage in FPGA interconnect testing.

A design flow developed by Susan Xu and others is used in this project with some modifications to implement the proposed test configurations. Simulation results are provided to demonstrate the feasibility of the proposed BIST scheme.

Acknowledgements

I would like to thank my supervisor, Dr. Xiaoling Sun, for her guidance, valuable discussions, and financial assistance from her research grant throughout this research project.

I would also like to thank Susan Xu for the development of the design flow, and Dr. Pieter Trouborst for his valuable comments.

Finally, I would like to thank my parents and my wife for providing support and understanding throughout my university education.

Table of Contents

1 Introduction **1**

1.1 Programming Technologies 1

1.2 General FPGA Architecture 2

1.2.1 Configurable Logic Blocks 3

1.2.2 Programmable Interconnect 3

1.3 Problem Statement 4

1.4 Thesis Objectives and Organization 5

2 Literature Review **8**

2.1 Current Measurement Techniques 8

2.2 Voltage Measurement Techniques 9

2.2.1 Boundary-Scan Techniques 9

2.2.2 Device Testing Techniques 10

2.2.3 Built-In Self-Test Techniques 10

3 FPGA CLB Test **13**

3.1 Background Material 13

3.2 Proposed BIST Scheme 15

3.3 Minimum CLM Test Configurations 17

3.3.1 Methodology 17

3.3.2 Testability of CLM Components 18

3.3.2.1 Preliminaries 18

3.3.2.2 Single Stuck-at Fault Testability 20

3.3.2.3 Multiple Stuck-at Fault Testability 22

3.3.2.4 Universal Fault Testability 25

3.3.3 Minimum CLM Test Configurations 26

3.3.3.1 The Minimums 26

3.3.3.2 The "Best" of the Minimums 27

3.4 Minimum CLB Test Configurations 30

3.4.1 Known Minimum and Test Challenges 31

3.4.2 Compatiability and Constraints 32

3.4.3 CLB Test Configurations 32

3.5 Evaluation and Discussion 34

3.5.1 Test vectors and test coverage 34

4 FPGA Interconnect Test **36**

4.1 Global Interconnect Testing 36

4.1.1 Background Material 36

4.1.1.1 Programmable Switches and Functional Models 36

4.1.1.2 Global and Local Routing Resources 38

4.1.1.3 Dedicated Device Features 39

4.1.2 The Proposed BIST Scheme 39

4.1.2.1 Proposed Test Strategy 40

4.1.3 Fault Models and Assumptions 41

4.1.4 BIST Architecture and Test Procedure 41

4.1.5 Test Sets 45

4.1.6 Proofs of Fault Detection 46

4.1.7 Application 50

4.2 Combined Global and Local Interconnect Testing 51

4.2.1 Background Material 51

4.2.1.1 Local Interconnect 51

4.2.1.2 Configurable Logic Blocks 53

4.2.2 The Proposed BIST Scheme 53

4.1.2.1 Proposed Test Strategy 53

4.2.3 Proposed BIST Architecture 56

4.2.4 Fault Models and Assumptions 58

4.2.5 Test Sets and Fault Detectability 59

4.2.5.1 Test Sets 59

4.2.5.2 Fault Detectability 60

4.2.6 Minimum Test Configurations 63

4.2.6.1 Interconnect Associated with a CLB 63

4.2.6.2 Derivation of the Test Configurations 64

4.2.6.3 Merging the Test for Global Interconnect 68

4.2.6.4 Combined CLB and Interconnect Testing 74

4.3 Discussion 74

4.3.1 Pros 74

4.3.2 Cons 74

5 Implementation 76

5.1 The CAD Environment 76

5.2 The Design Flow 80

5.3 The Implementation 81

5.4 Scalability and Adaptability 86

6 Conclusion and Future Work 87

6.1 Conclusion 87

6.2 Future Work 88

6.2.1 Extensions 88

6.2.2 Suggestions for Xilinx FPGA Design and CAD Tools 89

Bibliography 91

Appendices

1 Testability Equations for the CLM 97

2 Twenty-two 8-TC sets 109

3 17 Interconnect TCs 121

4 Distribution of CLB TCs into Interconnect TCs 156

5 VHDL Code for the BIST Circuitry 158

List of Figures

Figure 1.1 General FPGA Architecture 2

Figure 1.2 Configurable Logic Block 3

Figure 1.3 Function Generator 3

Figure 1.4 Typical Test Environment..5

Figure 3.1 CLB Diagram of Xilinx XC4000 Family FPGAs 14

Figure 3.2 XC4000 CLB Carry Logic Module...15

Figure 3.3 Proposed BIST Architecture 16

Figure 3.4 A Circuit Example...20

Figure 4.1 Programmable Switches..37

Figure 4.2 CLB Routing Resources..38

Figure 4.3 Conceptual Block Diagram 40

Figure 4.4 Switch Matrix Test Configurations 42

Figure 4.5(a) Orthogonal Test Architecture...43

Figure 4.5(b) Left_Diagonal Test Architecture 44

Figure 4.6 Detailed Diagram of Interconnect Associated with a CLB 52

Figure 4.7 Simplified Logic Diagram of a CLB...53

Figure 4.8 Conceptual Block Diagram of the Proposed BIST Strategy.......55

Figure 4.9 High-level BIST Architecture 57

Figure 4.10 An Example 67

Figure 4.11 Ratio between the Number of TCs and PS Test Coverage 73

Figure 5.1 The CAD Environment 79

Figure 5.2 Logic Symbols for Components in the Data Path 81

Figure 5.3 Logic Symbol of the BIST Controller 82

Figure 5.4 The State Diagram of the BIST Controller 84

Figure 5.4 The Layout of a Test Configuration 85

List of Tables

Table 3.1 CLM Weight Assignments 29

Table 3.2 A Set of Carry Logic Test Configurations 30

Table 3.3 Compatibility and Constraints between CLM TCs and Function Generators 32

Table 3.4 CLB Test Configurations 33

Table 4.1 Proposed Tests for k=3 45

Table 4.2 XC 4020E Global Routing Resources Test Configurations 51

Table 4.3 The Proposed Three Tests 60

Table 4.4 The Enumeration of Q Output Sequences of a DFF 62

Table 4.5 Interconnect Associated with a CLB 64

Table 4.6 Single and Double-Length Lines Tested in 17 Test Configurations 69

Table 4.7 16 Test Configurations That Test All Cross-Point PSs 70

Table 4.8 Number of TCs vs. Test Coverage 72

List of Theorems

Theorem 1 47
Theorem 2 48
Theorem 3 49
Theorem 4 49
Theorem 5 60
Theorem 6 61
Theorem 7 63
Theorem 8 63

List of Abbreviations

ASIC	-	Application Specific Integrated Circuit
BIST	-	Built-in Self-test
CAD	-	Computer-Aided Design
CLB	-	Configurable Logic Block
CLM	-	Carry Logic Module
CMOS	-	Complementary Metal Oxide Semiconductor
CMSFT	-	Controllability for a Multiple Stuck-at Fault Test
CSSFT	-	Controllability for a Single Stuck-at Fault Test
CUFT	-	Controllability for a Universal Fault Test
CUT	-	CLB under Test
DFF	-	D Flip Flop
DFT	-	Design for Testability
DRC	-	Design Rule Check
FPGA	-	Field Programmable Gate Arrays
GUI	-	Graphical User Interface
HDL	-	Hardware Design Language
ILA	-	Iterative Logic Array
IOB	-	Input / Output Block
LUT	-	Look-up Table
MSF	-	Multiple Stuck-at Fault
NCD	-	Native Circuit Description

NE	-	North East
NS	-	North South
NW	-	North West
ORA	-	Output Response Analyzer
PCG	-	Parity Code Generator
PS	-	Programmable Switch
RAM	-	Random Access Memory
SE	-	South East
SRAM	-	Static Random Access Memory
SSF	-	Single Stuck-at Fault
SW	-	South West
TC	-	Test Configuration
TMR	-	Triple Mode Redundancy
TMSFT	-	Testability for a Multiple Stuck-at Fault Test
TPG	-	Test Pattern Generator
TSSFT	-	Testability for a Single Stuck-at Fault Test
TUFT	-	Testability for a Universal Fault Test
UCF	-	User Constraint File
UF	-	Universal Fault
WE	-	West East
WG	-	Wire Group
WUT	-	Wire under Test

Chapter 1

Introduction

Field Programmable Gate Arrays (FPGAs) are field programmable logic devices that are widely used in digital designs. State-of-the-art FPGAs today contain millions of equivalent logic gates and can operate at clock frequencies of over 100 MHz. They have become widely accepted implementation technologies for low and medium-volume computing applications. FPGAs can be programmed in packages and in systems by end users with inexpensive programming devices. Design errors can be corrected quickly and often without changes to the physical system design. Different system functions can be re-programmed into the device for specialized data processing. System upgrades can be a matter of sending a new configuration to a user, without shipping and replacing a physical device. These features allow design engineers to design, debug, prototype, and upgrade a system many times to achieve the best results with the shortest time-to-market.

1.1 Programming Technologies

Currently there are several types of FPGAs with different programming technologies [1]. Among them SRAM-based FPGAs are of special interest due to their mainstream integrated circuit fabrication technology and their wide usage in practical applications. This thesis focuses on the testing of SRAM-based FPGAs.

In SRAM-based FPGAs, programmable connections are made through pass transistors, transmission gates, or multiplexers that are controlled by SRAM cells. As the system powers up, the SRAM programming data is downloaded into the FPGA from an external memory. The advantage of this technology is that it allows fast in-circuit

reconfiguration (the programming time is less than 100 ms) and the FPGA can be reprogrammed any number of times.

1.2 General FPGA Architecture

An FPGA consists of a two-dimensional array of *configurable logic blocks* (CLBs) which can be connected through programmable interconnect and *input/output blocks* (IOBs), as shown in Figure 1.1.

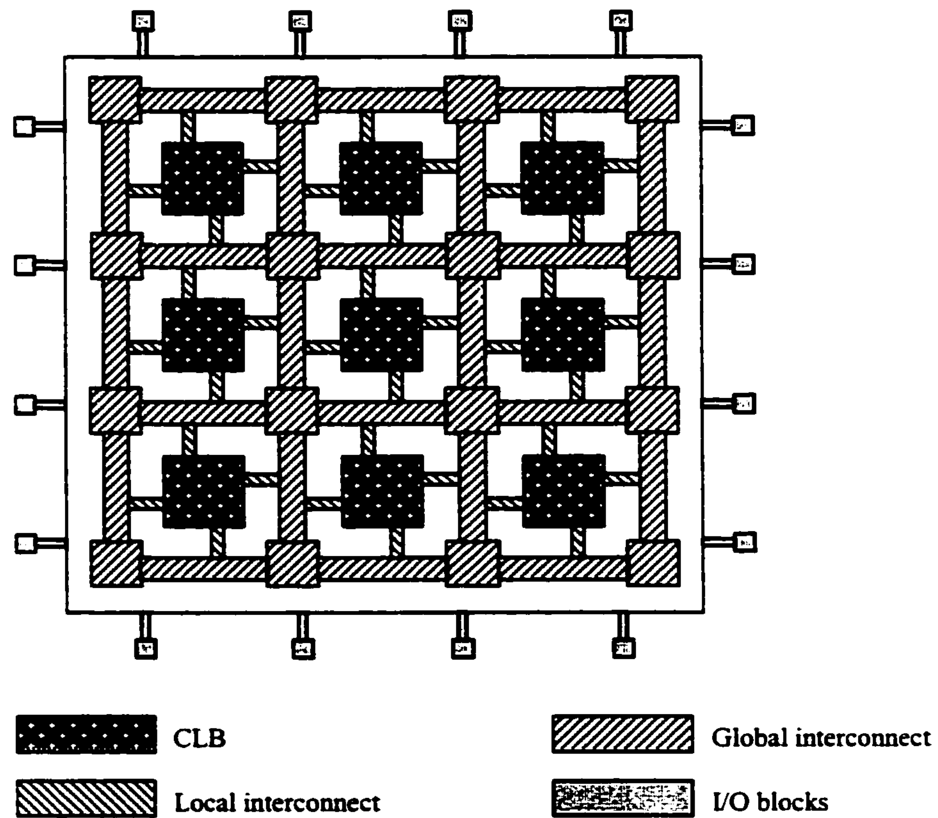


Figure 1.1: General FPGA Architecture

1.2.1 Configurable Logic Blocks (CLBs)

A CLB contains function generators, flip-flops, and multiplexers that are used to implement desired logic functions. An example giving the principal CLB elements is

shown in Figure 1.2. In Figure 1.2 each CLB contains two flip-flops, FF1 and FF2, three function generators, F, G, and H, and four multiplexers.

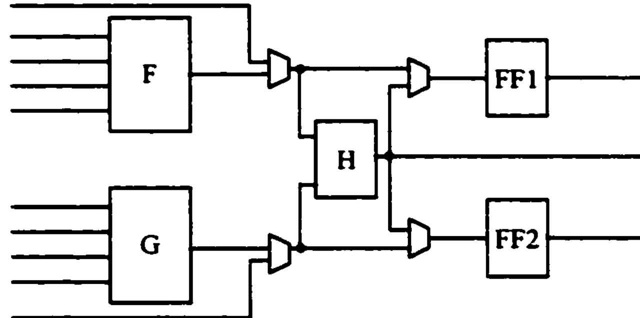


Figure 1.2: Configurable Logic Block

A function generator has two operation modes: *look-up table* (LUT) mode or RAM mode. Essentially, a function generator acts as a small ROM or RAM when operating in LUT or RAM mode respectively, whose output is selected by the input signals, as shown in Figure 1.3. To provide one Boolean function of N inputs requires 2^N configuration bits.

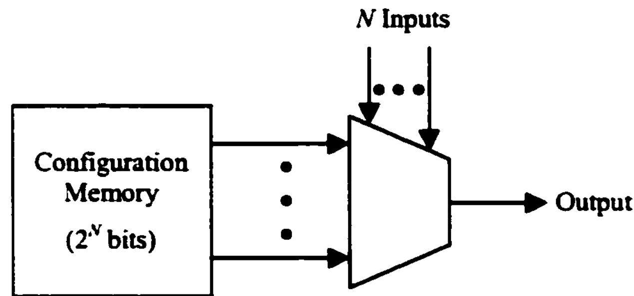


Figure 1.3: Function Generator

1.2.2 Programmable Interconnect

Programmable interconnect includes horizontal and vertical routing channels that run in between the CLBs. Each routing channel comprises wire segments and programmable

switches. An SRAM-based FPGA uses a number of basic elements to interconnect its routing resources. These include bi-directional interconnect, unidirectional interconnect, open collector interconnect and tri-state interconnect.

Bi-directional connections require pass transistors controlled by SRAM cells. Since pass transistors have relatively high electrical resistance, bi-directional buffers are required. The direction of the buffers is determined by an additional SRAM cell.

Unidirectional connections constrain wires to having one fixed driver, thus signals flowing along these wires are unidirectional. This avoids implementing bi-directional buffers but leads to less flexibility.

Open collector interconnect includes interconnect with open collector drivers. The advantage of open collector drivers is that large fan-in AND and OR gates can be constructed using wired logic.

Tri-state interconnect allows bus-like structures to be built on the FPGA using tri-state buffers. The designer has the responsibility of avoiding driver conflicts arising from more than one tri-state driver driving the signal at one time.

1.3 Problem Statement

Today it is common for *application specific integrated circuits* (ASICs) to be equipped with *built-in self-test* (BIST) features and to be provided with IEEE 1149.1 compliant standard test ports. BIST is a technique in which parts of a circuit are used to test the circuit itself. IEEE 1149.1 requires that a chip contain a set of four external signals, which are together called a JTAG port, for accessing testing-related chip features. The IEEE 1149.1 standard ensures that chips contain a common base of circuitry which will make the test development and testing of boards containing these chips significantly more effective and less costly.

In digital system products, an increasing number of FPGAs are used to implement complex system functions and co-exist with custom ASICs, as shown in Figure 1.4. In such a system, BIST function in the custom ASICs can be invoked through the JTAG port, while the FPGAs often remain untested when they are in the system. Therefore, it is desirable to develop in-system testing techniques for FPGAs, where the FPGAs

under test have already been mounted in the system to implement certain system functions and their I/O pins are connected to other system components. Moreover, the in-system testing techniques should be consistent with the BIST features in the ASICs and the standard JTAG test port.

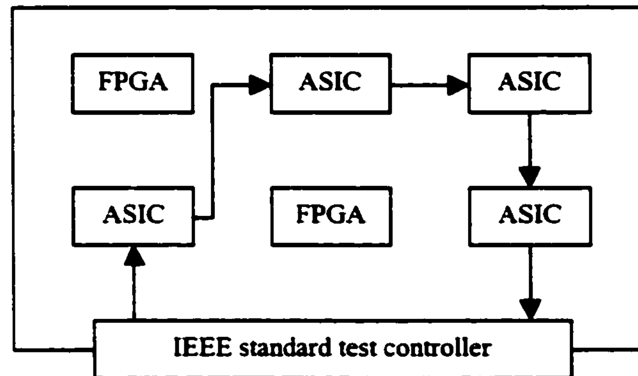


Figure 1.4: Typical Test Environment

During the last decade there has been considerable interest in developing testing techniques and strategies for FPGAs. However, in-system testing of FPGAs poses a challenging problem for test engineers because of the following reasons:

(1) Although considerable research has been done on component testing techniques for FPGAs in manufacturing testing, these techniques require full access to the I/O pins and assume the use of externally applied test sequences. Unfortunately, the full access to FPGA I/O pins is unavailable in the in-system testing environments.

(2) Because the system functions may change for FPGAs that are dynamically re-programmed, the tests provided for FPGAs are usually universal and function-independent for all instances of the same type of FPGAs. This poses a more difficult problem than the tests for the fixed-function ASICs.

(3) Testing an FPGA requires implementing various configurations in the FPGA, and changing configurations incurs re-programming costs. It is a difficult task to determine the minimum number of test configurations in order to minimize testing costs.

especially given the trend that each new generation of FPGAs contain more re-programmable components.

(4) The test coverage and the multiple fault detectability of published schemes are not satisfactory. For CLB testing there is no literature addressing the hardest-to-test components, the *carry logic modules* (CLMs), within a CLB. For interconnect testing, the current built-in self-test technique is a comparison-based technique, which does not have strong multiple fault detectability.

1.4 Thesis Objectives and Organization

The objectives of this thesis are:

- To develop in-system BIST methodologies that do not use the I/O pins of the FPGA and are independent of the system functions implemented by the FPGA.
- To modify and extend the original high-level BIST architectures in two previous project proposals [2,3] for the target FPGAs and to derive the minimum test configurations that verify the integrity of the CLBs including the CLMs, and that test the interconnect with superior multiple defect detectability.
- To construct improved fault models, develop test sequences for the modeled faults, and to study the fault detectability of the proposed schemes and look for ways of improvement.
- To implement the proposed schemes on real FPGAs to demonstrate the feasibility of the new tests.

The rest of this thesis is organized as follows.

Chapter 2 reviews the prior work on FPGA testing. General FPGA testing techniques are discussed, and some basic terminology is defined.

Chapter 3 first describes the proposed BIST strategy for CLBs by using the high-level BIST architecture in [2] with modifications. The necessary background material is presented. Various fault models under the proposed scheme are studied and the

corresponding test sequence is developed. A systematic method is introduced for finding the minimum number of test configurations for the CLMs under the proposed fault models. The search results are then given. An intuitive method for deriving the minimum number of test configurations for the remaining CLBs is given. The minimum test configurations for the entire CLB are presented.

Chapter 4 describes the novel BIST scheme for global interconnect resources of Xilinx XC4000E FPGAs using the error-control coding technique from [3]. This thesis first completes the work in [3] by constructing the fault models, developing the test sequence for the modeled faults, and analyzing the fault detectability.

The work in [3] is then extended in this thesis to include the testing of local interconnect, and to combine the testing of both local and global interconnect. A new BIST scheme is proposed for testing local interconnect and the combined global and local interconnect using a novel technique (functional test of D flip-flops) and the error-control coding technique in [3]. The fault models are given and the test sequence for the modeled faults is developed. The fault detectability is studied. The merger of the CLB and interconnect testing is also discussed.

Chapter 5 describes the implementation of the proposed test configurations, including the CAD environment, design flow, and implementation of the BIST circuitry. The test results demonstrate the feasibility of the proposed BIST scheme.

Finally, Chapter 6 draws conclusions about this project and recommends future research directions.

Appendix 1 contains testability equations for CLMs under the proposed fault models.

Appendix 2 lists twenty-two "best" 8-TC sets for CLM.

Appendix 3 provides 17 minimum TCs for FPGA local interconnect.

Appendix 4 outlines the distribution of CLB resources into the 17 TCs in Appendix 3.

Appendix 5 gives the VHDL source code for the BIST circuitry.

Chapter 2

Literature Review

All of the testing strategies proposed in the literature thus far for FPGAs use either voltage or current measurement techniques.

2.1 Current Measurement Technique

The current measurement technique detects defects in CMOS devices by measuring the quiescent power supply current. Such techniques are often called I_{DDQ} techniques [1]. They are based on the physical fact that defect-free CMOS circuits consume very little current in the quiescent state [5, 7]. The presence of defects, under the right conditions, can increase this quiescent current by an order of magnitude or more and can thus be used to detect many defects. In [5], an I_{DDQ}-based test strategy is described for detecting defects such as bridging faults in the logic resources and part of the interconnect in FPGAs, and the testing of FPGA input/output resources is discussed in [6]. The advantages of the I_{DDQ} techniques are that they can detect many defects that are undetectable by voltage measurement techniques. Test generation can be easier because there is no need to propagate the effect of the defect to a primary output. However, the disadvantages of I_{DDQ} techniques are also obvious. The measurement of the quiescent current must be very precise because a normal I_{DDQ} is very low, and an accurate measurement takes a significant amount of time. Also, it is not so easy to set an I_{DDQ} threshold for faulty devices. Because of these disadvantages, relatively little research has been done in the field of FPGA testing using I_{DDQ} techniques.

2.2 Voltage Measurement Technique

Most research that has been done in FPGA testing uses voltage measurement techniques. This testing approach distinguishes between two types of FPGA inputs: operation inputs and configuration inputs. One uses operation inputs during conventional FPGA operation to apply test sequences, and one also uses configuration inputs before conventional FPGA operation to configure the FPGA. FPGA testing using the voltage measurement technique consists of successively configuring the FPGA using the configuration inputs and then applying test sequences using the operation inputs.

Due to the inherent flexibility of FPGAs, the detectability of a fault depends on the FPGA configuration. A given fault may be redundant and therefore undetectable in a given test configuration, but the same fault may become non-redundant and detectable in another test configuration. Therefore, the test configurations should make as many modeled faults as possible appear non-redundant. Whether or not a fault can be detected also depends on the applied test sequence on the operation inputs. Therefore, under each test configuration, the objective is to generate the associated test sequence to detect all of the non-redundant faults defined by the test configuration.

Currently there are three main types of voltage measurement techniques for FPGA testing: the scan-chain based technique, the device testing technique, and the built-in self-test technique.

2.2.1 Scan Chain Based Technique

In order to ease the testing process and reduce the testing costs, one of the *design for testability* (DFT) strategies, the scan-chain based technique, has been applied in SRAM-based FPGAs. Among the scan-chain based techniques, the boundary-scan technique consists of adding scan registers to the input and output pins of FPGAs by the vendors. It requires the addition of some logic to the FPGA for control and some additional I/O ports. In general, boundary-scan provides a method for accessing all application inputs and outputs from an external test controller via a JTAG test access port. Several approaches have targeted the testing of FPGAs using boundary-scan

techniques [8, 9]. Recently, an implicit scan technique was proposed in [10]. It guarantees that any sequential circuit implemented in the FPGA can be implicitly scanned, at the cost of some area overhead and the modification of the classical FPGA architecture. The major disadvantage of scan chain based techniques for FPGA testing is the long test time, since the test patterns are scanned in and scanned out serially instead of in parallel. The test time is thus increased by at least an order of magnitude compared with parallel approaches.

2.2.2 Device Testing Technique

In the device testing techniques [11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22], test patterns are applied in parallel, which requires full access to the I/O pins and assumes the use of externally applied test vectors. In [11, 12, 13, 14, 15], the minimization of the number of CLB test configurations is considered. In [11, 12, 13], the CLBs in an FPGA are tested under the proposed AND tree and OR tree testing architecture, which provides strong multiple fault detectability for CLBs. In [14], the conventional CLB resources are modeled as a network of multiplexers when the function generators implement XOR/XNOR functions in their look-up table (LUT) modes. This last work is extended to different FPGA families in [15].

In [17], three test configurations are used for interconnect testing. The basic idea is to first configure interconnect to form "busses", then to test these busses using classical bus testing vectors applied to the I/O pins of the FPGA. Paper [18] presents heuristic test procedures for stuck-at faults, extra-device faults, and missing-device faults affecting programmable switch in the FPGA interconnect. These approaches try to solve the problem of manufacturing tests for FPGA interconnect and are only applicable for device-level testing. In [21, 22], the fault diagnosis of FPGA interconnect using the minimum number of test configurations is discussed under the single fault assumption.

2.2.3 Built-In Self-Test Technique

The first BIST approach for FPGA testing was proposed by Stroud *et al.* [23, 24, 25, 26, 27, 28, 29, 30]. In [23, 24, 25, 26], BIST for the CLBs is discussed. Essentially, a portion of the FPGA is configured into *test pattern generators* (TPGs) and *output*

response analyzers (ORAs). Then this BIST circuitry is used to test the CLBs in the remaining portion of the device. The TPGs and ORAs in [23, 24, 26] are distributed in the same area containing the CLBs under test, and many routing resources need to be used for signal connections. This may cause routing congestion when the BIST circuitry becomes complex. An iterative logic array (ILA) based approach is given in [26] to improve the routability of the above CLB BIST scheme.

In [27] and [28], BIST approaches for carry logic module testing within a CLB and the CLB testing together with the carry logic modules are discussed for the first time. The testability of the CLB including the CLM is discussed, and a methodology to derive testability Boolean equations for different fault models is provided. Routing congestion is avoided in [27, 28] by exploiting certain device features such as MUX busses and edge decoders.

The first BIST proposal for FPGA interconnect testing is a comparison-based approach [29]. The scheme configures a subset of the wire segments and programmable switches to form two groups of wires under test (WUTs). The WUTs receive identical test patterns from a group of CLBs configured as TPGs. The resulting outputs are compared by another group of CLBs configured as ORAs. This proposal promises good fault coverage for shorts and opens affecting wire segments, and stuck-on/off faults in programmable switches. However, the ORAs fail to detect multiple faults with identical faulty behavior in corresponding same positions in the two WUTs since the strategy is essentially a comparison-based method with two voting parties. In [30], an on-line FPGA testing and diagnostic method was proposed to test one column and one row of CLBs and interconnects (called the self-testing area) at a time without interrupting the normal system operation. The entire device is tested by gradually moving the self-testing area across the chip. The test configurations for the interconnect are inherited from [29] with some modifications.

The BIST approach in [31] proposes a novel technique for FPGA interconnect testing in which error control coding is applied to test FPGA interconnects for the first time. It avoids the unreliability of the comparison-based approach and can achieve strong multiple fault detectability. However, in [31] only the testing of part of the interconnect (global interconnects) is discussed.

Recently, the testing of transient and cross-talk faults in FPGA interconnects is considered for the first time [32].

Papers [33, 34] address the testing of both the CLBs and the interconnect, however, the scheme requires design modifications in the FPGA and the CLBs and interconnect are not tested simultaneously.

Chapter 3

FPGA CLB Test

This chapter focuses on the testing of the *configurable logic blocks* (CLBs) in Xilinx XC4000 series FPGAs. The background material is first prepared. The proposed *built-in self-test* (BIST) scheme is then presented. A systematic method for deriving the minimum number of test configurations for the *carry logic modules* (CLMs) is introduced, and followed by the distribution of the remaining resources of a CLB into the test configurations for CLMs. Finally, the proposed tests and the corresponding test coverage are also discussed.

3.1 Background Material

The logic block diagram of the Xilinx XC4000 CLB is shown in Figure 3.1 [31]. For ease of presentation, the CLB logic is partitioned into three parts: the conventional CLB resources, the interface logic, and the CLM.

The conventional CLB resources consist of the CLB logic commonly considered in the literature for CLB testing. It is shown to the right-hand side of the dotted line. The F and G function generators can be programmed as RAMs or *look-up tables* (LUTs), while the H block can be programmed as a LUT only.

The interface logic between the CLB and the CLM contains three multiplexers: MF1, MG1, and MG2. See the shaded multiplexers in Figure 3.1. They are used to select input signals for the function generators from among inputs G2, G3, F4, C_{in}, and the CLM output C_{out0}.

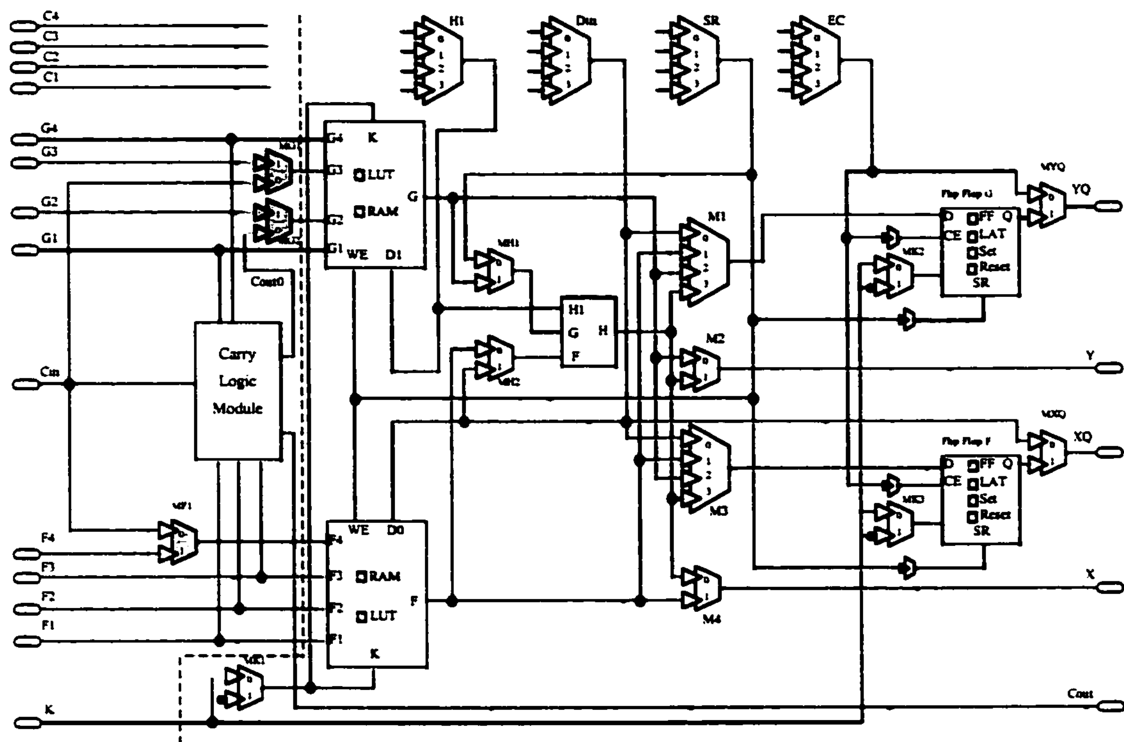


Figure 3.1: CLB Diagram of Xilinx XC4000 Family FPGAs

The logic block diagram of the CLM is shown in Figure 3.2 [31]. It has two types of inputs: *configuration inputs* (C0-C7) and *operation inputs* (C_{in}, F1-F3, G1, and G4). The former signals are part of the device configuration stream and can only be programmed through device configurations. The latter inputs supply signals to the CLM during normal operation. There are 2^8 possible binary combinations of the eight configuration inputs. Only forty-three of them are valid carry modes [31].

A CLM has two distinctive roles in a CLB when configured in one of its forty-three valid modes [31]. It supplies signals to the conventional CLB resources and the interface logic, and forms part of desired logic functions in a CLB. It can also be programmed as part of fast carry chains (iterative arrays), together with the adjacent CLM modules and the dedicated fast carry routing.

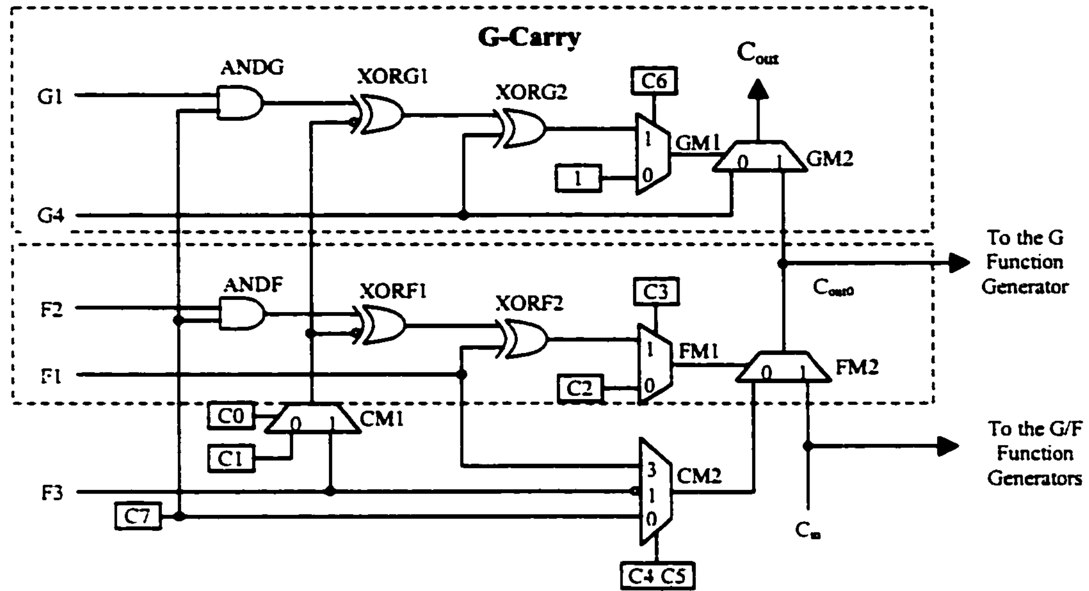


Figure 3.2: XC4000 CLB Carry Logic Module

In XC4000 series FPGAs, wired multiplexers (MUX-busses) can be formed by programming tri-state buffers and horizontal long lines. Two tri-state buffers are available for each CLB. Figure 3.3 shows a design with two MUX-busses for each row of CLBs. Dedicated carry logic routing resources link together the CLMs in CLBs to form fast carry chains.

3.2 Proposed BIST Scheme

In the proposed BIST scheme, an *m*×*m* array of CLBs in an FPGA is partitioned into two equal sections, an upper half and a lower half. When the upper half contains the *CLBs under test* (CUTs), the lower half is used to form the BIST circuitry, and vice versa. Prior to a test, each CUT is programmed into one of the CLB *test configurations* (TCs) presented in section 3.4. Figure 3.3 depicts the proposed high-level BIST architecture. The proposed BIST architecture has two distinct array structures: a parallel array of *m×(m/2)* CUTs, and *m/2* iterative arrays of the *m* CLM modules in the CUTs in every two adjacent CLB columns. For example, the two left-most CLB columns can form an iterative array.

During a test, the BIST circuitry broadcasts test vectors to the parallel array by means of a comb-like bus structure, shown in bold lines in Figure 3.3. The two outputs of the CUTs are connected to the wired MUX-busses via the tri-state buffers and the select signal S_i for column i, where $1 \leq i \leq m$. The MUX-busses carry the output responses of the CUTs back to the BIST circuitry.

In the same test, iterative arrays can be formed. For example, the CLM of the CUT at the bottom of the left-most column can take the test input from the BIST circuit, propagate signals upward to the CUT at the top of the column, then pass the signals on to the CUT at the top of the second column, and then propagate signals downward to the CUT at the bottom. The BIST circuit monitors and collects the output responses of the array. All of the *m/2* iterative arrays work in parallel. A pass/fail signal is asserted if either of the parallel array or the iterative arrays fails the test. The BIST circuitry interacts with the outside world through a built-in IEEE 1149.1 interface provided by the FPGA vendor [31].

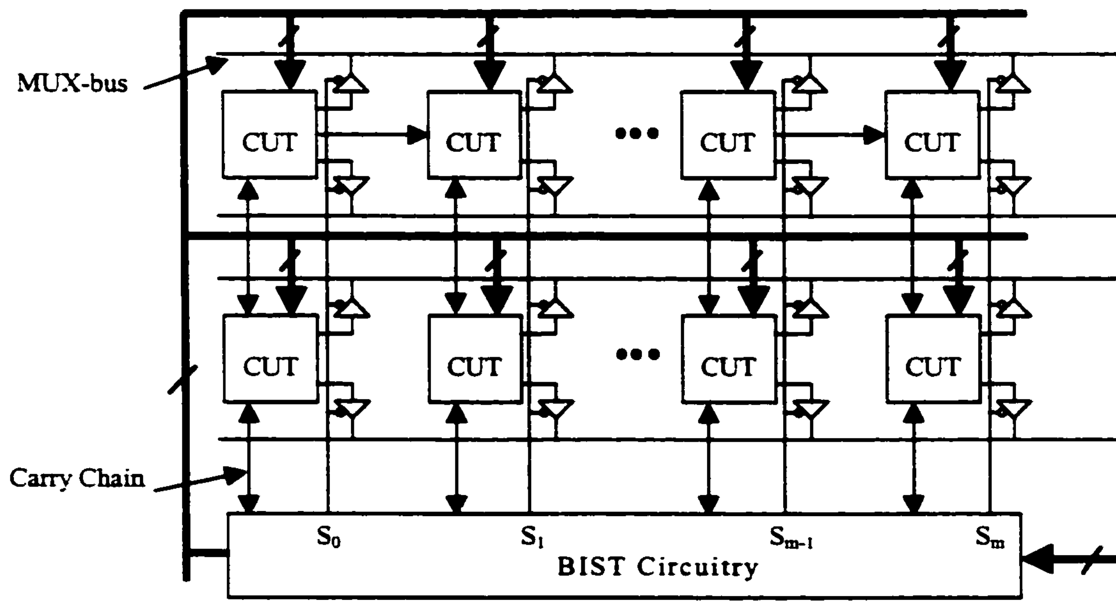


Figure 3.3: Proposed BIST Architecture

The proposed BIST scheme consists of two test sessions and each session tests half of the CLBs. It is assumed that interconnect resources of the FPGA are pre-tested and

fault-free. Such an assumption is unnecessary when the CLBs and interconnects are tested simultaneously as proposed in chapter 4. Hardware redundancy techniques, such as *triple modular redundancy* (TMR), can be used for the BIST circuitry to reduce the fault escape probability. The TMR technique is made possible by the simplicity of the BIST logic and the availability of many spare CLBs in the BIST section.

3.3 Minimum CLM Test Configurations

First, the circuit logic of the CLM and its operations are studied, and the *test configurations* (TCs) for the CLM are derived. The CLM is the largest sub-block of a CLB. Its depth of logic exceeds that of other CLB sub-blocks and it has relatively many configuration inputs. It is quite restricted in selecting CLM TCs as only 43 out of the 256 potential configurations are valid and many of the CLM configurations require using the function generators in the conventional CLB resources to implement certain logic functions. Due to these factors, it is harder to find a minimum set of TCs for the CLM than to find such a set for the rest of the CLB. After a minimum set of TCs is found for the CLM, they are extended to provide a set of TCs for the whole CLB.

3.3.1 Methodology

To test a CLM requires supplying test stimuli for both the configuration inputs and operation inputs. The configuration inputs can only be specified in the long serial bit stream that programs the FPGA. The time to download the bit-stream is in the range of tens to hundreds of milliseconds [14]. Therefore, it is not attractive to functionally verify each of the forty-three valid CLM modes or configurations.

On the other hand, once a carry mode has been configured, many different input combinations can be applied to the operation inputs in a short time. Since there are only six operation inputs in a CLM, an exhaustive test on the inputs can be done very quickly and this approach provides excellent functional coverage of the CLM in a given carry mode. The application time for an exhaustive test of a CLM is a few orders of magnitude smaller than the download time to define the configuration inputs. Therefore, to verify the integrity of the CLM, a small set of valid carry modes is

selected as the CLM TCs using the configuration inputs, and for each carry mode an exhaustive test is applied to the operational inputs. In this approach the challenge is to find a minimum set of valid carry modes so that the total test time is minimized.

Since a CLM has only forty-three valid carry modes, the integrity of the CLM can only be verified using the valid carry modes. Thus, the search for CLM TCs can be considered as selecting a minimum set of valid carry modes using the configuration inputs and verifying the integrity of the CLM when applying an exhaustive test on the operational inputs.

Among the twelve components in the CLM, there are six multiplexers, two AND and four XOR gates. A divide-and-conquer approach is used to verify the integrity of each component when applying an exhaustive test on operation inputs for a minimum set of TCs. The rationale is that the CLM should perform its intended functions if each of its components functions correctly. The modularity of an FPGA is exploited by configuring many CLMs into identical configurations and then testing them in parallel.

3.3.2 Testability of CLM Components

3.3.2.1 Preliminaries

The CLM is a combinational circuit. It has *six* primary inputs, *two* primary outputs and *twelve* embedded logic components, including AND gates, XOR gates, and multiplexers. First, the definitions in [page 343, ref. 1] are used to discuss the controllability and observability of the inputs and output of a component, then the testability of the CLM components under the *single stuck-at fault* (SSF) model, *multiple stuck-at fault* (MSF) model, and *universal fault* (UF) model are determined.

Definition 1 The *controllability* of a component input is the ability to establish a specific signal value at the component input by setting values on the primary circuit input(s) [1].

Definition 2 The *observability* of a component output is the ability to determine the signal value at the component output by controlling the primary inputs and observing the signals at the primary circuit outputs [1].

The CLM has *seven* configuration inputs and these static inputs reconfigure the circuit. The above definitions consider the ability to control and observe signals by manipulating the *operation inputs* within a certain configuration. The configuration inputs do affect the controllability and the observability: a signal may be controllable or observable in one configuration but not in others.

A very simple scheme is used to calculate controllability and observability. Given a configuration, we only need to know whether a signal can be controlled to a certain value or can be observed. Therefore the controllability of a component input or observability of a component output can assume only two values: 1 to denote the ability to control or observe, and 0 otherwise. Hence the controllability of component inputs and the observability of a component output can be represented by Boolean equations.

A *controllability equation* of a component input I_i, $C_{Ii}(0)$ *or* $C_{Ii}(1)$, defines the conditions to control input I_i to logic 0 or logic1 respectively from the primary input(s).

The *observability equation* of a component output F, O_F, specifies the conditions required to observe the output at a primary output.

Example 1 Consider the circuit in Figure 3.4. The controllability and the observability equations of the AND gate are:

$$C_{I1}(0) = 1, \qquad C_{I1}(1) = 1,$$

$$C_{I2}(0) = 1, \qquad C_{I2}(1) = 1,$$

$$O_F = \overline{I_3}.$$

Similarly, for the OR gate, they are:

$$C_F(0) = \overline{I_1} + \overline{I_2}, \qquad C_F(1) = I_1 I_2,$$

$$C_{I3}(0) = 1, \qquad C_{I3}(1) = 1,$$

$$O_G = 1.$$

Note that the controllability equations of primary inputs have constant 1 values, since they are fully controllable, and the observability of primary outputs have constant 1 values since they are fully observable. For non-primary inputs/outputs, certain primary input conditions are required to set the controllability/observability equations

to logic 1. The primary inputs that set an input controllability equation of a component to logic 1 can control the component input to the logic value defined by the controllability equation. For instance, to set input F of the OR gate to logic 1, the two primary inputs, I_1 and I_2, must be logic 1, as defined in $C_F(1) = I_1 I_2$. The primary inputs that set the output observability equation to logic 1 allow the component output to be observed at a primary output.

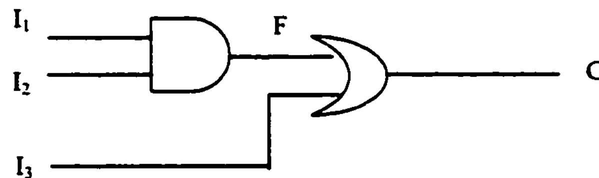


Figure 3.4: A Circuit Example

Definition 3 The *controllability* of a component is the ability to set the desired input combinations on the component inputs by controlling the primary inputs.

The observability of a component is the same as that for the component output defined in Definition 2 since a component has only one output.

A fault model determines the input combinations required for testing a component. The input is component-type dependent. Using the divide-and-conquer approach described in section 3.3.1, it is assumed that the SSF model, MSF model, and UF model are with respect to the individual CLM components.

3.3.2.2 SSF Testability

Definition 4 The *controllability for a single stuck-at fault test* (CSSFT) of a k-input AND gate is the ability to set both logic 0 and 1 on each input and set logic 1 on the remaining (k-1) inputs by controlling the primary inputs.

The CSSFT of a k-input AND gate can be expressed by a set of Boolean equations, called CSSFT equations. One CSSFT equation can be obtained by Boolean multiplication of the input controllability equations that set the component inputs to a

desired input combination. A k-input AND gate has $k+1$ CSSF equations. If the controllability or observability conditions of a CSSF equation are satisfied, the equation assumes the logic value 1, otherwise, it assumes the logic value 0.

Example 2 For the 2-input AND gate in Figure 3.4, the inputs, 01, 10 and 11, detect all the SSFs. The three respective CSSFT equations are:

$$C_{F1} = C_{I_1}(0) \times C_{I_2}(1),$$

$$C_{F2} = C_{I_1}(1) \times C_{I_2}(0),$$

$$C_{F3} = C_{I_1}(1) \times C_{I_2}(1).$$

When the primary inputs satisfy both input controllability equations of a CSSFT equation, the CSSFT equation assumes the value of logic 1. And the input combinations defined by the CSSFT can be set on the component inputs.

Definition 5 The CSSFT of a k-input XOR gate is the ability to set both logic 0 and 1 on each input and set logic 0 (or logic 1) on the remaining (k-1) inputs by controlling the primary inputs.

Definition 6 The CSSFT of a multiplexer with k data inputs and $log_2\, k$ select inputs is: (1) the ability to select each data input and apply both 0 and 1 to the selected data input, and (2) the ability to set each select input combination, and apply logic 0 (or logic 1) to the selected data input and the complementary logic value on the remaining (k-1) data inputs, by controlling the primary inputs.

Example 3 Consider a multiplexer with a select input, S, and two data inputs, I_1 and I_2. The CSSFT equations of the multiplexer are:

$$C_{MUX1} = C_S(0) \times C_{I_1}(0),$$

$$C_{MUX2} = C_S(0) \times C_{I_1}(1),$$

$$C_{MUX3} = C_S(1) \times C_{I_2}(0),$$

$$C_{MUX4} = C_S(1) \times C_{I_2}(1),$$

$$C_{MUX5} = C_S(0) \times [C_{I_1}(0) \times C_{I_2}(1) + C_{I_1}(1) \times C_{I_2}(0)],$$

$$C_{MUX6} = C_S(1) \times [C_{I_1}(0) \times C_{I_2}(1) + C_{I_1}(1) \times C_{I_2}(0)].$$

Proposition 1 If all the CSSFT equations of an AND gate, a XOR gate or a multiplexer can assume the value of logic 1, then a test for a SSF can be applied to the component from the primary inputs.

Definition 7 The *testability for a single stuck-at fault test* (TSSFT) of the CLM component is the ability to satisfy its respective CSSFT equations, and to simultaneously observe the component's output at a primary output by controlling the primary inputs.

The TSSFT of a component can be expressed by Boolean multiplication of its observability equation with each of its CSSFT equations. A TSSFT equation thus assumes the value of logic 1 if the primary inputs satisfy all its CSSFT equations and its observability equation.

Example 4 The TSSFT equations of the multiplexer in Example 3 are:

$$T_{F1} = O_F \times C_S(0) \times C_{I1}(0),$$

$$T_{F2} = O_F \times C_S(0) \times C_{I1}(1),$$

$$T_{F3} = O_F \times C_S(1) \times C_{I2}(0),$$

$$T_{F4} = O_F \times C_S(1) \times C_{I2}(1),$$

$$T_{F5} = O_F \times C_S(0) \times [C_{I1}(0) \times C_{I2}(1) + C_{I1}(1) \times C_{I2}(0)],$$

$$T_{F6} = O_F \times C_S(1) \times [C_{I1}(0) \times C_{I2}(1) + C_{I1}(1) \times C_{I2}(0)].$$

Proposition 2 If all the TSSFT equations of an AND gate, an XOR gate or a multiplexer can assume the value of logic 1, then the test for SSF can be applied to the component by controlling the primary inputs, and the output can be observed at a primary output.

3.3.2.3 MSF Testability

Definition 8 The *controllability for a multiple stuck-at fault test* (CMSFT) of a k-input AND gate is the ability to set all k walk-0 test patterns and one all-1 test pattern on the inputs of the AND gate by controlling the primary inputs [36], [page 120, ref. 1].

The CMSFT of a k-input AND gate can be expressed as the product of the k+1 controllability equations that control the inputs to the k walk-0 tests and one all-1 test.

Example 5 The CMSFT equations of the AND gate in Figure 3.4 are:

$$C_{F1} = C_{I1}(0) \times C_{I2}(1),$$
$$C_{F2} = C_{I1}(1) \times C_{I2}(0),$$
$$C_{F3} = C_{I1}(1) \times C_{I2}(1).$$

Proposition 3 If all the CMSFT equations of an k-AND gate can assume the value of logic 1, then the test for multiple stuck-at faults can be applied to the AND gate.

Definition 9 The CMSFT of a k-input XOR gate is the ability to set a walk-1 test and one all-0 test to the inputs of the XOR gate by controlling the primary inputs [1, 36].

The CMSFT of a k-input XOR gate can be expressed by the products of the k+1 controllability equations that control the inputs to the k walk-1 tests and one all-0 test.

Example 6 Replacing the 2-input AND gate in Figure 3.4 by a 2-input XOR gate, its CMSFT equations are:

$$C_{XOR1} = C_{I1}(1) \times C_{I2}(0),$$
$$C_{XOR2} = C_{I1}(0) \times C_{I2}(1),$$
$$C_{XOR3} = C_{I1}(0) \times C_{I2}(0).$$

Proposition 4 If all the CMSFT equations of the XOR gate can assume the value of logic 1, then the test for multiple stuck-at faults can be applied to the XOR gate.

Definition 10 The CMSFT of a multiplexer with k data inputs and log_2k select inputs is the ability to select each data input and to apply both 0 and 1 to it while applying the complementary logic values to the remaining data inputs by controlling the primary inputs [37].

The CMSFT of a multiplexer with k data inputs and log_2k select inputs can be expressed by the products of the controllability equations that set the log_2k select

inputs, and the equations that control the selected data input to logic 0 and logic 1 and the other inputs to the complement values.

Example 7 Consider a multiplexer with a select input, S, and two data inputs, I_1 and I_2. The CMSFT equations of the multiplexer are:

$$C_{MUX1} = C_S(0) \times C_{I1}(0) \times C_{I2}(1),$$

$$C_{MUX2} = C_S(0) \times C_{I1}(1) \times C_{I2}(0),$$

$$C_{MUX3} = C_S(1) \times C_{I1}(0) \times C_{I2}(1),$$

$$C_{MUX4} = C_S(1) \times C_{I1}(1) \times C_{I2}(0).$$

Proposition 5 If all the CMSFT equations of a multiplexer can assume the value of logic 1, then the test for multiple stuck-at faults can be applied to the multiplexer.

Definition 11 The *testability for multiple stuck-at fault test* (TMSFT) of a component in Definitions 5, 6, or 7 is the ability to set all their respective test input values by controlling the primary inputs, and the ability to observe the output of the component on a primary output.

The TMSFT of a component in Definitions 5, 6, or 7 can be represented by the products of its observability equation and its CMSFT equations.

Proposition 6 If all the TMSFT equations of a component can assume the value of logic 1, then the test for multiple stuck-at fault can be applied to the component by controlling the primary inputs and the output of the component can be observed at the primary output(s).

Example 8 The TMSFT equations of the AND gate in Figure 3.4 are:

$$T_{F1} = O_F \times C_{I1}(0) \times C_{I2}(1),$$

$$T_{F2} = O_F \times C_{I1}(1) \times C_{I2}(0),$$

$$T_{F3} = O_F \times C_{I1}(1) \times C_{I2}(1).$$

3.3.2.4 UF Testability

Definition 12 The *controllability for a universal fault test* (CUFT) of a component is the ability to set all the possible input combinations of the component by controlling the primary inputs.

The CUFT of a component can be expressed as the products of its inputs' controllability equations that control the inputs to all 2^k possible input combinations, where k is the number of inputs of the component.

Example 9 Consider the 2-input AND gate in Figure 3.4. The CUFT equations of the AND gate are:

$$C_{F1} = C_{I1}(0) \times C_{I2}(0),$$

$$C_{F2} = C_{I1}(0) \times C_{I2}(1),$$

$$C_{F3} = C_{I1}(1) \times C_{I2}(0),$$

$$C_{F4} = C_{I1}(1) \times C_{I2}(1).$$

Proposition 7 If all the CUFT equations of the component can assume the value of logic 1, then an exhaustive test can be applied to the component.

Definition 13 The *testability for a universal test* (TUFT) of a component is the ability to set all the possible input combinations to the component by controlling the primary inputs, and to observe the output of the component on the primary output for each input combination.

The TUFT of a component is the product of its observability equation and its CUFT equations.

Example 10 The TUFT equations of the AND gate in Figure 3.4 are:

$$T_{F1} = O_F \times C_{I1}(0) \times C_{I2}(0),$$

$$T_{F2} = O_F \times C_{I1}(0) \times C_{I2}(1),$$

$$T_{F3} = O_F \times C_{I1}(1) \times C_{I2}(0),$$

$$T_{F4} = O_F \times C_{I1}(1) \times C_{I2}(1).$$

Proposition 8 If all the TUFT equations of a component can assume the value of logic 1, then an exhaustive test can be applied to the component by controlling the primary inputs and the output of the component can be observed at a primary output.

3.3.3 Minimum CLM TCs

3.3.3.1 The Minima

In the previous section, the testability equations for all the component types in the CLM under the single stuck-at, multiple stuck-at, and universal fault models are defined. A set of testability equations for all the components in the CLM under each fault model [38] is derived. There are 62, 40 and 80 testability equations for the single stuck-at, multiple stuck-at and universal fault models respectively, as shown in Appendix 1. The testability equations are functions of both the configuration inputs and operation inputs of the CLM.

Example 11 Consider the C2 stuck-at 1 fault for the component FM1 in Figure 3.2 on page 15. To detect this fault we need:

(a) To set C2=0, select C2 by setting C3=0, so that FM1=C2.

(b) To observe FM1 at the primary output, C_{out}.

The testability equation, $T_{FM1}(C2_{s\text{-}a\text{-}1})$, is given in equation (1) below. Substituting equations (2)-(13) recursively into equation (1), the new $T_{FM1}(C2_{s\text{-}a\text{-}1})$ equation is a function of the operation inputs, G1, G4, F1, F3, and C_{in}, and the configuration inputs, C0-C7. This equation can assume logic 1 in one of the valid carry modes, ADD-G-F3 defined by C0-C7 = 01000111, when an exhaustive test is applied to the operation inputs. This implies that the logic 0 applied to C2 in order to activate the C2 stuck-at 1 fault can be observed at the primary output.

$$T_{FM1}(C2_{s\text{-}a\text{-}1}) = C_{C2}(0)*C_{C3}(0)*O_{FM1} = (!C2)*(!C3)*O_{FM1}; \quad (1)$$

$$O_{FM1} = ((C_{CM2}(0)*Cin + (C_{CM2}(1)*(!Cin)))*O_{FM2}; \quad (2)$$

$$C_{CM2}(0) = (!C4*(!C5)*(!C7)) + (F3*(!C4)*C5) + (!F1*C4*C5); \quad (3)$$

$$C_{CM2}(1) = (!C4*(!C5)*C7) + (!F3*(!C4)*C5) + (F1*C4*C5); \quad (4)$$

$$O_{FM2} = C_{GM1}(1); \quad (5)$$

$$C_{GM1}(1) = (!C6) + (C6 * C_{XORG2}(1)); \quad (6)$$

$$C_{XORG2}(1) = (C_{XORG1}(0) * G4) + (C_{XORG1}(1) * (!G4)); \quad (7)$$

$$C_{XORG1}(0) = (C_{ANDG}(0) * C_{CM1}(1)) + (C_{ANDG}(1) * C_{CM1}(0)); \quad (8)$$

$$C_{XORG1}(1) = (C_{ANDG}(0) * C_{CM1}(0)) + (C_{ANDG}(1) * C_{CM1}(1)); \quad (9)$$

$$C_{ANDG}(0) = (!G1) + (!C7); \quad (10)$$

$$C_{CM1}(1) = (!C0 * C1) + (F3 * C0); \quad (11)$$

$$C_{ANDG}(1) = G1 * C7; \quad (12)$$

$$C_{CM1}(0) = (!C0 * (!C1)) + (!F3 * C0); \quad (13)$$

Given a set of the testability equations, a subset of the equations can assume logic 1 in a valid carry mode (i.e. in a CLM TC). Note that n valid carry modes are required to satisfy all the equations, where $1 \leq n \leq 43$. Therefore, the problem of finding the minimum CLM TCs that verify the integrity of a CLM is equivalent to finding the minimum n that satisfy all the testability equations under a given fault model.

An algorithmic search was implemented. The search results show that the minimum values are 6, 7 and 8 for TSSFT, TMSFT, and TUFT, respectively. There exist 6,384 sets of 6 TCs, 4,756 sets of 7 TCs, and 8,928 sets of 8 TCs. All the sets under each fault model are equally good as the TCs when the CLM is considered as an independent module.

3.3.3.2 The Best of the Minimums

The TC sets with the minimum values 6, 7 and 8 CLM TCs under each fault model were studied. It has been found that at least four TCs use the F and G function generators in each set. Thus, none of the sets can be tested independently without using the F and G function generators, and therefore the CLM can not be considered as a completely independent module.

Moreover, there exist input dependencies between the inputs of the CLM and those of the CLB. Recall that the CLM shares inputs F1-F3, G1 and G4 with the CLB, and the two CLM outputs, C_{out0} and C_{out}, serve as the G2 input of the G function generator and the C_{in} input of an adjacent CLB, respectively. This implies that the conventional CLB resource may not receive exhaustive test patterns when an exhaustive test is applied to the CLB inputs, due to the input dependencies between the CLM and the

conventional CLB resource. A question that then arises is whether there exists any minimum CLM TC set in the equally good values 6, 7 and 8 TCs, that permits the simultaneous test of the CLM and the remaining part of the CLB.

There are two factors that interfere with the simultaneous test of the two parts, the use of F and/or G function generators and the input dependencies between the CLM and the CLB resource. A weight is assigned to each valid carry mode to indicate its functional dependency on the F and G function generators and the input dependencies between the CLM and the CLB resource. F_{NU} and G_{NU} are used to denote the respective function generators are *not used* in a carry mode, and F_U and G_U, otherwise. The input dependency and independence between the CLM and the CLB for each function generator are denoted by F_D and G_D, and F_I and G_I, respectively.

Table 3.1 shows all the possible combinations of the conditions and their weight assignments. A mode with a bigger weight allows a higher degree of parallel testing for the CLM and the conventional CLB. The weight assignment rules are:

(1) A weight of 0 is assigned to a carry mode that uses both F and G function generators to implement functions other than 4-input XOR/XNOR.

(2) A weight of 1 is assigned to a carry mode in which one function generator has no input dependency and either is not used or is used to implement a 4-input XOR-XNOR function.

(3) A weight of 2 is given when none of the function generators has an input dependency and each function generator either is not used or is used to implement a 4-input XOR/XNOR function.

Note that 4-input XOR/XNOR functions can be used to test the F and G function generators (see section 3.4 for details). In Table 3.1, a certain combination may have multiple weight values. This is because such a combination contains more than one valid carry mode and different carry modes may have different weight values.

Each carry mode is defined in the Xilinx Software Manuals [39]. One can determine if a carry mode requires the use of F/G function generators and has input dependency by examining the mode definition.

Table 3.1: CLM Weight Assignments

	G_UF_U	G_UF_{NU}	$G_{NU}F_U$	$G_{NU}F_{NU}$
G_DF_D	0	0	0	0
G_DF_I	0/1	1	0	1
G_IF_D	0/1	0	1	1
G_IF_I	0/1/2	1	1	2

Example 12 Carry mode ADD-F-CI defined by the following equations is F_U, G_{NU}, F_D, and G_I.

$$F = F1 \oplus F2 \oplus F4;$$
$$C_{out0} = (F1 * F2) + C_{in} * (F1 + F2);$$
$$G = \text{not used};$$
$$C_{out} = C_{out0};$$
$$F4 = C_{in};$$
$$G2 = G2;$$
$$G3 = G3;$$
$$C0 - C7 = 0\ 1\ 0\ 1\ 1\ 1\ 0\ 1.$$

Definition 14 The weight of a CLM test set is the sum of the weights of all the carry modes in the set.

The selection criteria for the "best" minimum CLM test configurations are:

(1) Having the maximum set weight.

(2) No weight-0 carry mode in the set.

In addition, the TUFT sets are also required which are the super sets of TMSFTs and the TMSFTs are the super sets of TSSFTs, so that a subset of a TUFT set can also be used as a single and multiple stuck-at fault test if needed.

Twenty-two sets of 8 TC sets with the maximum weight of 13 were found, as shown in Appendix 2. All twenty-two sets have five weight-2 and three weight-1 carry modes. Thus, they are equally good choices of CLM TCs. Table 3.2 shows one set of

configurations randomly chosen from the twenty-two. The first six, seven (note that it is not any combination of six or seven) and eight TCs are for TSSFT, TMSFT and TUFT, respectively. These CLM TCs can be combined with the CLB test configurations to achieve a complete test of a CLB, as discussed in section 3.4.

Table 3.2: A Set of Carry Logic Test Configurations

Carry Mode	Configuration Bits C0, C1, C2, C3, C4, C5, C6, C7							
	C0	C1	C2	C3	C4	C5	C6	C7
ADD-G-F3	0	1	0	0	0	1	1	1
ADDSUB-FG-CI	1	0	0	1	1	1	1	1
INC-G-1	0	0	0	0	0	0	1	0
DEC-F-CI	0	0	0	1	1	1	0	0
FORCE-1	0	0	0	0	0	0	0	1
FORCE-CI	0	0	1	0	0	0	0	0
FORCE-F3	0	0	0	0	0	1	0	0
FORCE-F1	0	0	0	0	1	1	0	0

3.4 Minimum CLB Test Configurations

In this section, the minimum TCs are derived for the whole CLB, including the CLM, conventional CLB resources and the interface logic. Using a similar approach to the derivation of the CLM TCs, the integrity of the CLBs is validated by verifying the integrity of individual components within them. Thus, the problem of testing the whole CLB is to find a minimum set of CLB TCs that sets each circuit node to the required tests from the CLB inputs and observes the nodes from the CLB outputs when exhaustive tests are applied to the CLB operation inputs. The single stuck-at fault model is assumed for all the CLB components except the function generators. The function generators are implemented as memory look-up tables, so separate treatment is required to obtain satisfactory test coverage. During a test, the CLBs are configured into one of the CLB TCs and are tested using exhaustive test patterns and the BIST architecture described in section 3.2.

3.4.1 Known Minimum and Test Challenges

The objective of this project is to derive CLB TCs that support more comprehensive test coverage of CLBs. The main challenges are:

(1) The inclusion of the CLM in the CLB and the bi-directional fast carry chains.

(2) The test of both RAM and LUT modes of the function generators in the conventional CLB resources. It includes all the three RAM modes, 32x1, 16x1 single port and 16x1 dual port.

The prior work on CLB testing focused on the conventional CLB resources [14, 15, 23, 25, 26, 40]. A minimum of five TCs was found for the LUT modes of XC4000A/D/H/L CLBs [14]. A minimum of five CLB TCs was obtained for the conventional CLB resources of XC4000E FPGAs, similar to that of [14]. A lower bound of the minimum number of CLB TCs in this research is eight: five for the LUT modes, and three for the RAM modes. The inclusion of the three RAM modes requires three CLB TCs since the LUT and RAM modes of a function generator are mutually exclusive.

3.4.2 Compatibility and Constraints

In [14], the conventional CLB resources were modeled as a network of multiplexers when the function generators implement XOR/XNOR functions in their LUT modes. In such a network, although it is easier to control and observe the inputs and outputs of a component in a configuration than that of the CLM, there are a number of restrictions. These restrictions are largely due to the use of the function generators and the routing constraints inherited from the device. For example, a 32x1 RAM requires using all three function generators, the 1-input of multiplexer MH1, and the 0-input of MH2. This greatly limits the choices when selecting CLB resources and propagating signals to a CLB output in a CLB TC.

The constraints of the CLM TCs and the function generators are studied in this section. Table 3.3 summarizes the constraints that the CLM configurations pose on the choices for configuring the CLB. A CLM TC can be associated with either a RAM or a LUT mode of a function generator. The two CLM TCs, FORCE-1 and FORCE-CI, on

rows 5-8, have two rows displaying the CLB constraints, one for each mode. On the other hand, ADD-G-F3, ADDSUB-FG-CI, INC-G-1, and DEC-F-CI on rows 1-4 require the F and G function generators to implement dedicated LUT functions. Their respective RAM modes, thus, are excluded. The check symbol "√" indicates that the tests of a CLM TC and a RAM/LUT mode of a function generator are compatible, i.e. the tests can be performed simultaneously.

Table 3.3: Compatibility and Constraints between CLM TCs and Function Generators

Column #	1	2	3	4	5	6	
CLM TC	Mode	32 RAM	16 RAM	F LUT	G LUT	H LUT	Row #
ADD-G-F3	LUT			√	$G1 \oplus G2 \oplus G4$	√	1
ADDSUB-FG-CI	LUT			√(4-XNOR)	√(4-XNOR)	√	2
INC-G-1	LUT			√	$\overline{G4}$	√	3
DEC-F-CI	LUT			$\overline{F1 \oplus F4}$	√	√	4
FORCE-1	RAM		√				5
	LUT			√	√	√	6
FORCE-CI	RAM	√	√				7
	LUT			√	√	√	8

3.4.3 CLB Test Configurations

The testability equation method introduced in section 3.3.2 for deriving CLM TCs does not take the constraint factors into consideration, therefore it is not applicable to the derivation of CLB TCs. An intuitive method is used to derive the CLB TCs in this section. It begins with the first six CLM TCs in Table 3.2, uses the constraint information in Table 3.3, and tries to integrate the test of the components in the conventional CLB resources and the interface logic into the CLM TCs. The goal is to obtain the minimum CLB TCs which support the complete single stuck-at fault test of the whole CLB when exhaustive tests are applied to the CLB.

Table 3.4 (a) and (b) depict one set of the minimum eight CLB TCs that we derived. Column 2 lists the CLM TCs used in the CLB TCs. Columns 3 to 25 correspond to the 23 components in the conventional CLB resource and interface logic. The numbers, 0-3, denote the input pins of multiplexers used in a CLM TC. The symbols "x" and "-"

indicate that a component is not used and can not be used in a CLM TC, respectively. The LUT functions and multiplexer pins in bold indicate that no other choices but the ones given are permitted in the TCs. The derivation steps of the CLB TCs are as follows.

Table 3.4 (a): CLB Test Configurations

1	2	3	4	5	6	7	8	9	10
TC #	Carry Mode	M_{F1}	M_{G1}	M_{G2}	F Function	G Function	M_{H1}	M_{H2}	H Function
1	FORCE-CI	0	1	1	RAM 32×1		1	0	$H1*F+\overline{H1}*G$
2	X	1	1	1	RAM 16x1 (single port)	RAM 16x1 (single port)	-	-	-
3	FORCE-1	1	0	1	RAM 16x1 (dual port)	RAM 16x1 (dual port)	-	-	-
4	ADD-G-F3	0	1	0	4-XNOR	G1⊕G2⊕G4	0	1	3-XNOR
5	INC-G-1	0	1	1	4-XOR	$\overline{G4}$	1	0	3-XOR
6	ADDSUB-FG-CI	0	1	0	4-XNOR	4-XNOR	1	0	3-XNOR
7	DEC-F-CI	0	1	1	$\overline{F1 \oplus F4}$	4-XOR	1	0	3-XOR
8	X	1	1	1	4-XOR	4-XNOR	1	0	3-XNOR

(1) First consider the two RAM testing modes on rows 5 and 7 of Table 3.3. They have the least numbers of check symbols, thus, they are least compatible with the CLM TCs. One additional carry mode needs to be introduced to support the three desired RAM modes, 32x1, 16x1 single port and 16x1 dual port. The three RAM TCs are the CLB TC 1, 2 and 3 in Table 3.4 (a), where TC 2 is the new TC.

(2) The multiplexer network model and XOR/XNOR functions in [14] are used to test the LUT modes of the function generators. The LUT modes that implement these LUT functions are grouped by the shaded boxes for each function generator in Table 3.4 (a). In Step (1), the selection of the RAM modes for FORCE-1 and FORCE-CI eliminated two out of the four LUT tests for G function generator (rows 6 and 8), a new TC must be added to complete the XNOR-XOR-XNOR test for the G (the TC 8). The test for the H function can be accomplished by either TCs 4-6 or TCs 6-8. The choice of TCs 4-6 for H was arbitrary. The LUT functions that are neither in bold nor shaded in Table 3.4 (a) can implement any arbitrary function.

(3) The choice of a multiplexer input may be limited by the mode and the function that a function generator implements, see the bold pin numbers in Table 3.4. Each data input of the remaining components is used at least once within the eight CLB TCs. See Table 3.4 (b) for details.

Table 3.4 (b): CLB Test Configurations

1	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
TC #	M1	M2	M3	M4	M_{K1}	M_{K2}	M_{K3}	H1	Din	SR	EC	M_{YQ}	M_{XQ}	YQ	XQ
1	3	-	0	-	1	1	1	0	1	2	3	**1**	**1**	FF/Reset	FF/Reset
2	2	-	1	-	0	0	0	1	2	3	0	**1**	**1**	FF/Set	FF/Set
3	1	-	2	-	1	1	1	-	3	0	1	**1**	**1**	Latch/Reset	Latch/Reset
4	3	0	-	1	-	0	-	3	0	1	2	**1**	-	FF/Reset	-
5	0	-	3	-	-	0	0	2	1	3	0	**1**	**1**	Latch/Set	Latch/Set
6	-	1	-	-	-	-	-	3	-	-	1	0	-	-	-
7	-	-	-	0	-	-	-	1	2	-	-	-	0	-	-
8	0	-	3	-	-	1	1	0	3	1	2	**1**	**1**	FF/Reset	FF/Reset

3.5 Evaluation and Discussion

3.5.1 Test Vectors and Test Coverage

Function Generators in LUT Mode

The XOR-XNOR-XOR (or their inverse) functions can detect single/multiple cell stuck-at faults and transition faults in the LUT modes of the function generators, when applying exhaustive vectors on the LUT input (address lines).

Function Generators in RAM Mode

A set of standard RAM testing algorithms, such as MATS++ and SMarch tests, are used to test the RAM modes of the function generators. They guarantee the detection of stuck-at cell faults, address decoder faults, transition faults, read/write logic faults, dynamic coupling faults and parametric faults in the RAM modes of F and G function generators [41]. The internal logic of the level-sensitive RAM is slightly different from

that of the edge-triggered RAM. An additional three CLB TCs are required. Tradeoffs were made to test the edge-triggered RAM modes only since they exercise more RAM resources [31].

Remaining CLB resources

Exhaustive tests on the CLBs support the complete single stuck-at fault coverage for the CLMs (using six CLM TCs) and the remaining CLB resources.

Chapter 4

FPGA Interconnect Testing

In this chapter, the *built-in self-test* (BIST) issues of FPGA interconnect are addressed, using the popular Xilinx XC4000E FPGA as the device model.

The interconnect in an XC4000E FPGA includes horizontal and vertical routing channels that run in between the CLB arrays. Each routing channel comprises wire segments and programmable switches. *Global interconnect* (or global routing resources) contains wire segments and switch matrices located at the intersections of horizontal and vertical routing channels. *Local interconnect* brings signals into and out of CLBs through CLB I/O pins and programmable switches that connect the I/O pins to the global interconnect.

Interconnect plays an important role both for programming flexibility and performance of FPGAs. Due to the complexity of FPGA interconnect, divide-and-conquer techniques are used in this project to verify interconnect integrity. This chapter focuses first on the testing of global interconnect, and then on the testing of combined global and local interconnect.

4.1 Global Interconnect Testing

4.1.1 Background material

4.1.1.1 Programmable Switches and Functional Models

There are three types of programmable switches in XC4000E series FPGAs [35]: basic programmable switches (PSs), cross-point PSs and multiplexer PSs.

A basic PS contains a transmission gate and a SRAM cell. The SRAM cell can be programmed to open or close the transmission gate. Figure 4.1 (a) shows a basic PS.

A cross-point PS consists of six basic PSs, connecting the wire segments in west-east (WE), north-south (NS), north-west (NW), south-west (SW), north-east (NE), and south-east (SE) directions (see Figure 4.1 (b)). These transmission gates are denoted as WE, NS, NW, WS, NE, and SE, respectively. Several cross-point PSs form a switch matrix with the cross-point PSs on the main diagonal of the matrix (shown in Figure 4.1 (c)).

A multiplexer PS functions as a conventional many-to-1 MUX. It allows one of the inputs to be routed to the output for given selection signals. The selection logic can only be set through the configuration bits during the configuration stage [35]. Figure 4.1 (d) depicts the symbolic diagram of a 4-to-1 multiplexer PS and its corresponding functional model.

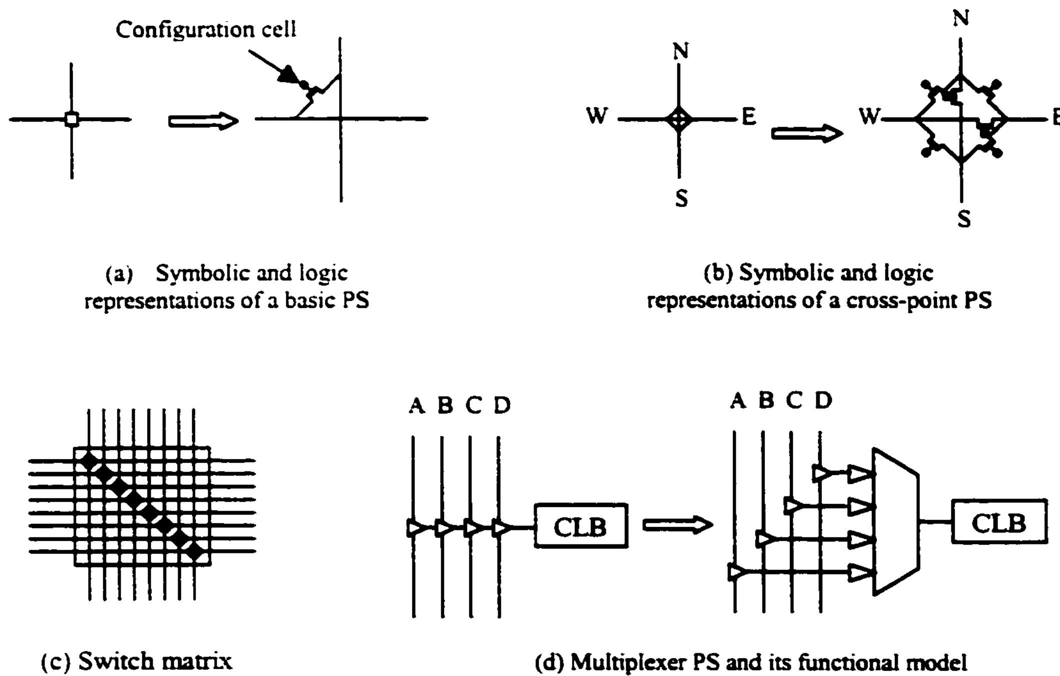


Figure 4.1: Programmable Switches

4.1.1.2 Global and Local Routing Resources

A high-level diagram of the wire segments associated with a configurable logic block (CLB) is shown in Figure 4.2. There are five types of wire segments distinguished by their relative length [35]. They are single-length lines, double-length lines, long lines, global lines, and carry lines. A two-letter acronym is used to denote the wire groups, where the first letter, H/V, indicates if a wire group is horizontal or vertical, the second letter, S/D/L/G/C, denotes single-length, double-length, long, global, and carry line(s) respectively. The number following the two letters represents the number of lines in a group. For example, HS8 denotes a group of 8 horizontal single-length lines.

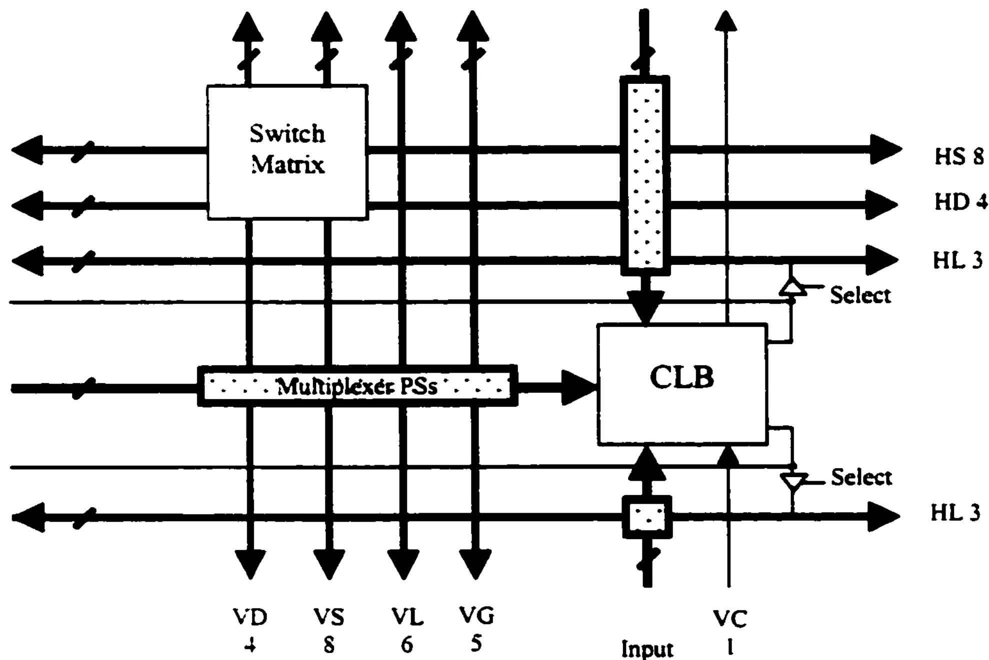


Figure 4.2: CLB Routing Resources

The single-length lines and double-length lines have the same routing architecture and vary only in the length of wire segments. Long lines run the entire length or width of the device. Global lines are designed to distribute clocks with minimum delay and skew and can also be used for carrying other high fan-out signals. Global interconnect resources encompass switch matrices, single-length lines, double-length lines, long lines and global lines. Local interconnect includes I/O pins of CLBs and programmable

switches (shown in the dotted boxes in Figure 4.2) that bring signals into and out of CLBs. Note that the dedicated fast carry lines are not tested here since they can only be tested together with the carry logic modules (CLMs) [28], as discussed in chapter 3.

4.1.1.3 Dedicated Device Features

As shown in Figure 4.2, each CLB has a pair of tri-state buffers. The assertion of the selection control "Select" allows the outputs of a CLB to be driven onto the nearest horizontal long lines implementing multiplexed busses [35]. Four wide-edge decoders are used to boost the performance of wide decoding functions. Each of them can implement wired-AND/NOR logic. They are located on the four sides of the I/O boundary. These routing resources are part of the I/O routing resources and are ignored in the test strategy for global interconnect resources.

4.1.2 The Proposed BIST Scheme

4.1.2.1 Proposed Test Strategy

The proposed BIST strategy for global interconnect is a parity-checking based scheme. It makes use of the in-system re-programmability of FPGAs to first program the device into a set of minimum test configurations, perform deterministic tests on interconnect, then program the device back into its intended system functions. The hardware resources of an FPGA are partitioned into two equal sections, an upper section and a lower section. Each contains CLBs and interconnect. When interconnect of the upper section is under test, the CLBs and interconnect in the lower section are used to implement the BIST circuitry, and vice versa.

Figure 4.3 shows the conceptual block diagram of the proposed BIST scheme. A portion of wire segments and programmable switches are configured to form *a wire group* (WG). A WG contains k wires, called *wires under test* (WUTs), and one wire conveying the parity bit, denoted as the WUTs_Parity. Later in section 4.1.6 we show that the WUTs_Parity uses a different routing channel in order to improve the fault coverage. The *test pattern generator* (TPG) supplies the WUTs and WUTs_Parity with test vectors at one end of the wires, where WUTs_Parity is the parity of each sub-test vector applied to the WUTs. *A parity checker* at the other end of the wire busses

computes the parity code of the WUTs, called PCG_Parity, and compares the PCG_parity with the WUTs_Parity. A disagreement in the two indicates an error in the WG. Figure 4.3 is an even parity implementation of the parity checker. The parity code generator (PCG) is a k-input XOR function while the Comparator is a 2-input XNOR gate. An odd parity-checking scheme can be easily implemented with minor modifications. Logic 1 at the output of the comparator, O_p, indicates error-free and logic 0 means an error is detected. The ***output response analyzer*** (ORA) is designed to record first-fails on O_p during a test and to assert a pass/fail signal (P/F) after the applications of all input vectors.

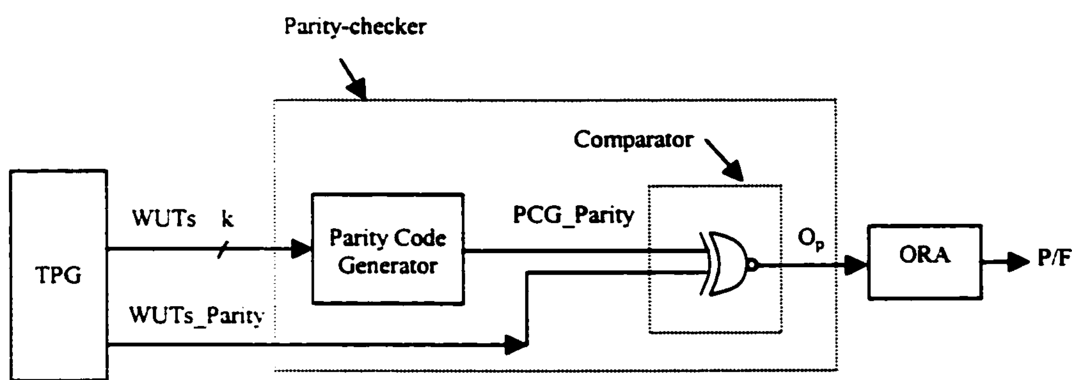


Figure 4.3: Conceptual Block Diagram

The proposed scheme may appear to be a conventional parity-checking scheme that detects only 50% of the errors in individual data vectors. However, as opposed to techniques that try to detect intermittent errors in signals that travel over noisy channels, non-intermittent faults are targeted and sensitized here with many different vectors. The key is that, in the proposed schemes, an exhaustive test set is applied: a fault missed by the parity bit of one test vector will be detected by one or more of the other vectors in the test set. A specially-designed ORA detects a parity error on any of the test vectors. In section 4.1.6, we show that the proposed scheme is capable of detecting all single and m multiple stuck-at 0/1 faults, stuck-open faults and bridging faults in wire segments, stuck-on/off faults and bridging faults in programmable switches, and combinations of the faults outlined above, where $1 \leq m \leq k$.

4.1.3 Fault Models and Assumptions

The fault models under consideration for FPGA interconnect include:

(1) Multiple segment stuck-at 0/1 faults;

(2) Multiple segment stuck-open faults;

(3) Multiple bridging faults;

(4) Multiple switch stuck on/off faults;

(5) The combinations of the above faults.

The following assumptions are made:

Assumption 1 When a logic value, 0 or 1, is applied to one end of a wire which contains stuck-open fault(s), the wire presents a logic 0 at the other end [1].

This assumption is quite realistic, even if the segment of the wire beyond the open defect would be floating at some mid rail voltage. Such a voltage will be interpreted by the parity checking logic as either a static low or a static high signal.

Assumption 2 A floating voltage caused by a bridging fault produces either a logic 0 or a logic 1, i.e. the bridging implements either a wired-AND or wired-OR function.

This assumption covers a wide range of bridging behavior of wire segments.

Assumption 3 A stuck-on fault of a switch has the same faulty behavior as a bridging fault between two wire segments connected by the switch [21].

The switch stuck-on fault is detected as long as the bridging fault between the two wire segments can be detected in *one* of the proposed test configurations. Therefore, switch stuck-on faults will be considered as wire segment bridging faults for the rest of this chapter.

Assumption 4 No detailed adjacency information is required within a WG, in addition to the information provided in the vendor's data books.

4.1.4 BIST Architecture and Test Procedure

Switch matrices can connect wire segments to form global busses. They are the most complicated programmable switches in an FPGA. Three distinct switch matrix configurations are required to test the six basic PSs in a cross point PS [17]. Figure 4.4 depicts the three test configurations, called orthogonal, left-diagonal and right-diagonal, for a switch matrix with 8 cross point PSs. In orthogonal (left-diagonal, or right-diagonal) test configurations, all the WE and NS (NW and SE, or WS and NE) PSs in the switch matrix are programmed closed and all the other PSs are programmed open.

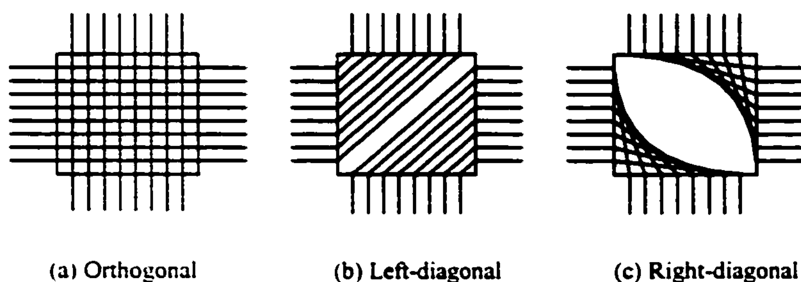


Figure 4.4: Switch Matrix Test Configurations

Figure 4.5 (a) and (b) show the proposed high-level orthogonal and left-diagonal test architectures, respectively. The right-diagonal test architecture is similar to the left-diagonal test architecture. In Figure 4.5 (a), there are two WGs, WG1 and WG2. WUTs_1 and WUTs_2 are formed with all horizontal and vertical wire segments under test. They are connected by the WE and NS PSs in all the switch matrices and by the boundary connections. WUTs_Parity_1 and WUTs_Parity_2 use boundary routing resources away from the routing channels of the WUTs so that the probability of a WUTs_Parity bit and all its WUTs bridging together is negligible.

Two parity checkers are used: one for the horizontal WG and one for the vertical WG. The outputs of the parity-checkers are first connected to the respective MUX-busses, then to the wide-edge decoder implementing a wired-AND bus. The wired-AND bus provides the input to the ORA. The select signals of the MUX-busses are provided by the TPG (not shown in the diagram). They allow direct access to the

outputs of an individual CLB (implementing a parity checker). The left and right diagonal structures are similar to the orthogonal structure. The BIST circuitry also interacts with the outside world through a built-in IEEE 1149.1 interface provided by the FPGA vendor [35].

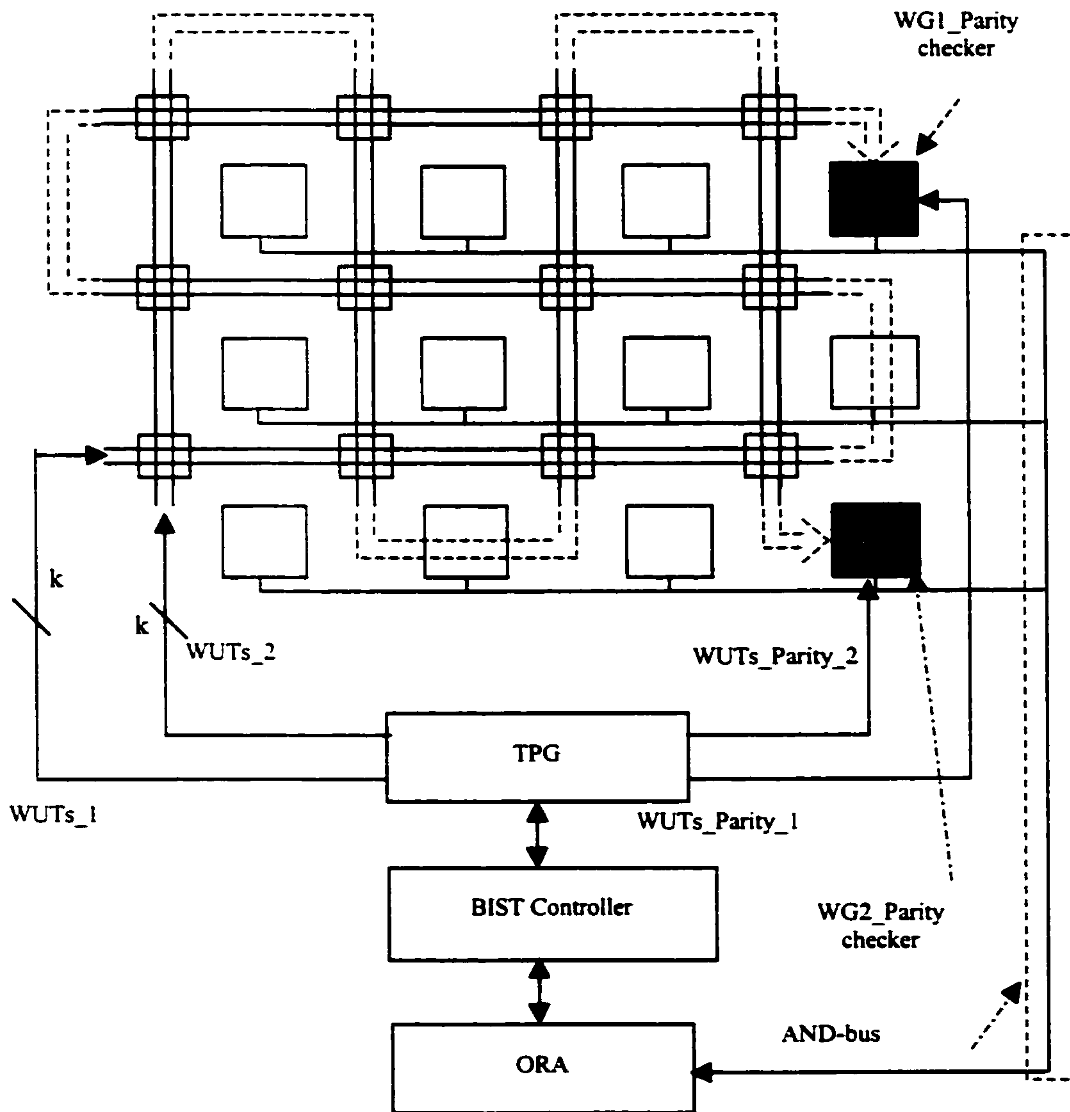


Figure 4.5 (a): Orthogonal Test Architecture

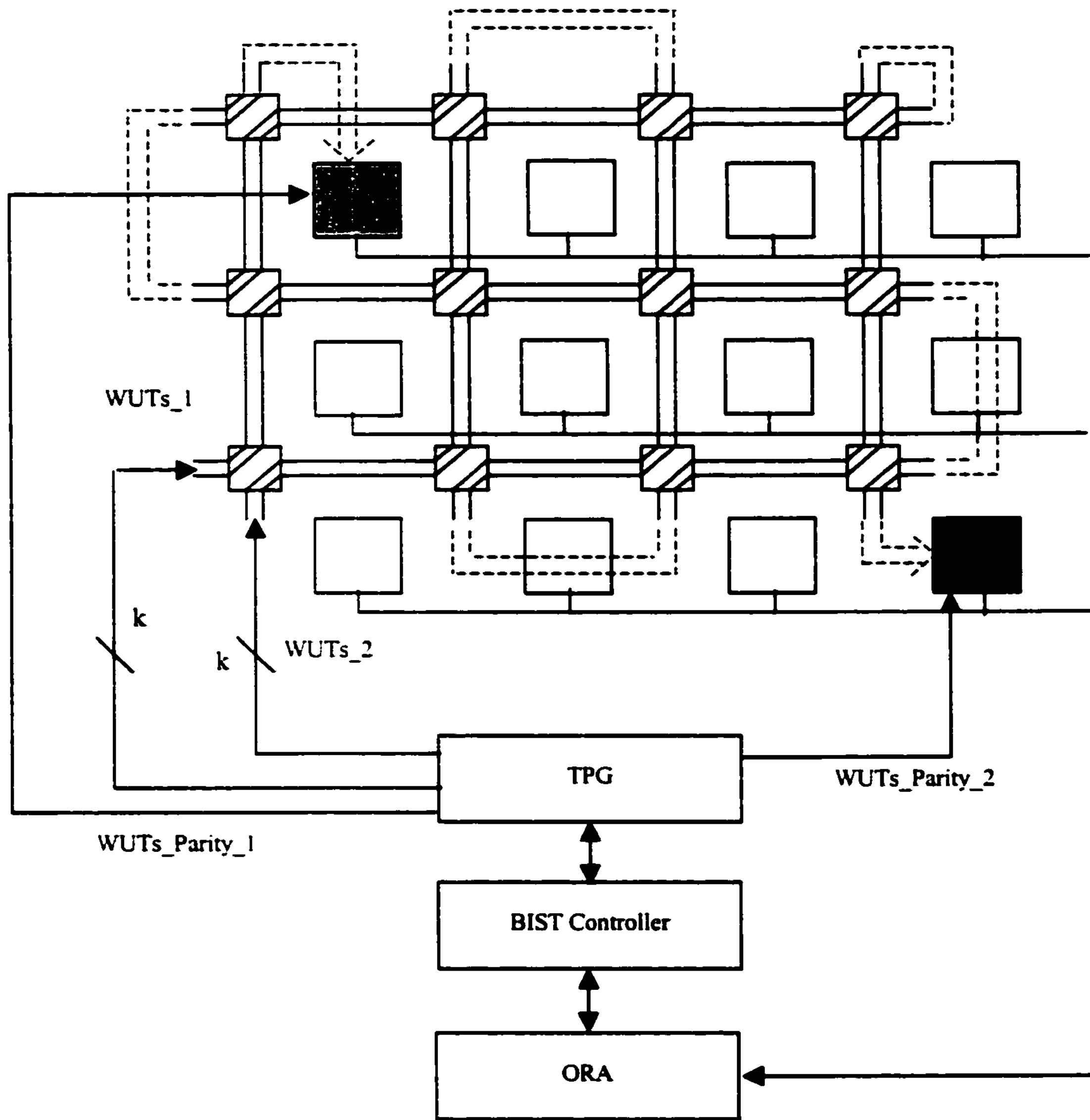


Figure 4.5 (b): **Left-diagonal Test Architecture**

4.1.5 Test Sets

To ease the presentation, we first consider two wire groups, WG1 and WG2. Later, in section 4.1.7, it will be shown that many wire groups can be tested simultaneously without causing routing congestion in an FPGA. WG1 and WG2 are formed by connecting the horizontal and/or vertical wire segments by means of programmable switching matrices into one of the three test configurations shown in Figure 4.5. A WG contains a k-wire WUTs and a WUTs_Parity wire. WG1 and WG2 have a total of $2(k+1)$ wires. Three proposed tests are shown in Table 4.1 for k=3. In all the three tests, the respective parity bits, WUTs_Parity_1 and WUTs_Parity_2, are computed in the conventional manner for the chosen parity checking (even parity) scheme. Each test contains several test vectors. A test vector is defined as $V = [e_1, e_2, ..., e_{k+1}]^T$, where $e_i \in \{0,1\}$ and $1 \leq i \leq k+1$.

Table 4.1: Proposed Tests for k=3

			Test 1 v1 v2 v3 v4 v5 v6 v7v8	Test 2 v1 v2 v3 v4 v5 v6	Test 3 v1 v2 v3 v4 v5 v6
WG1	WUTs_1	e1	0 0 0 0 1 1 1 1	1 0 0 0 0 0	0 1 1 1 1 1
		e2	0 0 1 1 0 0 1 1	0 1 0 0 0 0	1 0 1 1 1 1
		e3	0 1 0 1 0 1 0 1	0 0 1 0 0 0	1 1 0 1 1 1
	WUTs_Parity_1	e4	0 1 1 0 1 0 0 1	1 1 1 0 0 0	0 0 0 1 1 1
WG2	WUTs_2	e1	1 1 1 1 0 0 0 0	0 0 0 1 0 0	1 1 1 0 1 1
		e2	1 1 0 0 1 1 0 0	0 0 0 0 1 0	1 1 1 1 0 1
		e3	1 0 1 0 1 0 1 0	0 0 0 0 0 1	1 1 1 1 1 0
	WUTs_Parity_2	e4	1 0 0 1 0 1 1 0	0 0 0 1 1 1	1 1 1 0 0 0

Test 1 is an exhaustive test of 2^k vectors for WUTs_1 and WUTs_2, respectively. The test vectors for WUTs_2 is the complement of that of WUTs_1. Test 2 is a walk-1 test for WUTs_1 when an all-0 test is applied to WUTs_2. It is followed by an all-0 test for WUTs_1 while a walk-1 test is applied to WUTs_2. Test 3 is a walk-0 test for WUTs_1 when an all-1 test is applied to WUTs_2. It is followed by an all-1 test for WUTs_1 while a walk-0 test is applied to WUTs_2.

Each of the proposed tests is designed to detect a subset of the modeled faults. In particular, Test 1 is capable of detecting multiple segment-stuck-at 0 (stuck-open) faults, segment-stuck-at-1 faults and switch-stuck-off faults in the WGs, and multiple-wire bridging faults (switch-stuck-on faults) within a WG. Test 2 and Test 3 detect multiple-wire bridging faults (switch-stuck-on faults) between WG1 and WG2 for either wired-AND or wired-OR bridging faults. The proofs of fault detection for the proposed scheme can be found in the following subsection.

4.1.6 Proofs of Fault Detection

Let a sub-vector S of a test vector V be $S \subseteq V$ and $S \neq \varnothing$. Test vectors applied to one end of a WG are called the inputs to the wire group and the logic values appearing on the other end of the WG are the outputs.

It can be seen from Figure 4.3 that fault(s) in a WG can be detected if the outputs of PCG_Parity and WUTs_Parity are different. Thus, to prove the detectability for fault(s) under a certain test, we just need to prove that the fault(s) will cause different outputs on PCG_Parity and WUTs_Parity.

Assumption 5 Multiple faults can affect at most k wires in a WG of (k+1)-wires.

This assumption does not pose a serious limitation. Since the parity bit is routed along a different channel from the WUTs, and since the direct parity connection is much shorter than the total length of the WUTs that are formed by concatenating many wire segments, it is unlikely that a fault will affect all WUTs and the parity wire at the same time.

The following proofs make use of a pair of sub-vectors S_{W1} and S_{W2} that are identical to each other except for one or two bit positions. Since Test 1 is an exhaustive test, any arbitrary vector pair $\{S_{W1}, S_{W2}\}$ is contained in Test 1.

Definition 1 Given a set W of m faulty wires in a WG, an m-bit sub-vector S_m in which each bit corresponds to a wire in the wire set W is called an associated sub-vector of the set W, where $1 \leq m \leq k$.

Note that a m wire subset W has 2^m associated sub-vectors.

Lemma 1 For a set W of m faulty wires in a WG due to any combination of single or multiple segment stuck-at-0/1 faults, stuck-open faults, switch stuck-off faults and/or wire bridging faults, there exist two associated m-bit sub-vectors S_{m1} and S_{m2}, which are identical except for one bit, that produce the same values on the outputs of all wires in W.

Proof: Consider an associated sub-vector S_{m1} that is constructed as follows:

(1) The input bits of any wires that are affected by a wired-AND bridging fault are set to 0;
(2) The input bits of any wires that are affected by a wired-OR bridging fault are set to 1;
(3) Any other bits in S_{m1} are set to an arbitrary value.

Note that for each wire bridging fault that affects the wire set W, sub-vector S_{m1} contains at least two bits that are set to the controlling value for that wire bridging fault. The controlling value of a wired-AND (wired-OR) function is 0 (1).

The second associated sub-vector S_{m2} is derived from S_{m1} by flipping any arbitrary bit in S_{m1}. Vectors S_{m1} and S_{m2} produce the same values on the outputs of all wires in W. If the flipped bit is associated with a wire affected by a wire bridging fault, then there is still another bit in S_{m2} that has the controlling value for that wire bridging fault so that the output values are not changed. If the flipped bit corresponds to a faulty wire that is not affected by a bridging fault, then it is straightforward that the output of the wire has the same value for both vectors. **QED**.

Theorem 1 Test 1 can detect any combination of single or multiple segment stuck-at-0/1 and stuck-open faults, switch stuck-off faults and/or wire bridging faults affecting any m wires of a WG, where $1 \le m \le k$.

Proof: *Case 1: The WUTs_Parity is fault-free.*

The m faulty wires are all WUTs. According to Lemma 1 there exist two m-bit sub-vectors S_{m1} and S_{m2} associated with the m faulty wires which differ by one input bit and still produce the same values on the outputs of the faulty wires. From these sub-

vectors we can construct two vectors S_{W1} and S_{W2} for the k-wire WUTs using arbitrary values for the bits associated with the fault-free wires, such that S_{W1} and S_{W2} are identical except for the one bit that was flipped to derive S_{m2}. Therefore, all the outputs in the WG, including the output parity PCG_Parity, have the same value for S_{W1} and S_{W2}. However, the input parity WUTs_Parity is different for S_{W1} and S_{W2} because the vectors differ in one bit. Because of this parity error the fault is detected.

Case 2: The WUTs_Parity is faulty.

At least one of the wires in the WUTs is fault-free. According to Lemma 1 there exist two m-bit sub-vectors S_{m1} and S_{m2} associated with the m faulty wires which differ by one input bit and still produce the same values on the outputs of the faulty wires. From these sub-vectors we can construct a vector S_{W1} for the k-wire WUTs using arbitrary values for all but one of the bits associated with the fault-free wires. Since S_{m1} specifies a value for the input parity, one of the bits associated with the fault-free wires needs to be set to a value that generates the required input parity. Vector S_{W2} is derived from S_{W1} as follows:

(1) S_{m1} is replaced by S_{m2};

(2) One of the bits associated with the fault-free wires is flipped.

Note that parity of S_{W2} is correct. If the difference between S_{m1} and S_{m2} is in the parity bit, then the flipping of the fault-free bit is required to justify the parity. If the difference between S_{m1} and S_{m2} is in a non-parity bit, then the flipping of the fault-free bit is required to make sure the parity remains the same.

The output values of all wires, including the output of the WUTs_Parity wire, are the same for S_{W1} and S_{W2} except for the output of the fault-free wire whose bit value was flipped to derive S_{W2}. Due to the one output bit that is different for S_{W2} the output parity PCG_Parity is different for the two vectors. This will generate a parity error since the output of the WUTs_Parity wire is the same for both vectors, and the fault is detected. **QED**.

Theorem 2 Test 2 can detect any wired-AND bridging faults between m wires in WG1 and n wires in WG2, where $1 \leq m, n \leq k$.

Proof: *Case 1: WUTs_Parity_1 and WUTs_Parity_2 are fault free.*

Test 2 is a walking-1 test applied to WUTs_1 and WUTs_2 together. Consider a vector V that applies a logic 1 to one of the *m* faulty wires in WUTs_1. The WUTs_Parity_1 for this vector is 1. The outputs of WUTs_1 will all be 0 because of the wired-AND bridging fault, thus PCG_Parity_1 is 0. The disagreement between WUTs_Parity_1 and PCG_Parity_1 detects the fault.

Case 2: At least one of WUTs_Parity_1 and WUTs_Parity_2 is faulty.

Without loss of generality, suppose WUTs_Parity_1 is faulty. Consider a vector V that applies the single 1 to one of the $(k-m+1)$ fault-free wires in WG1. WUTs_Parity_1 is 0 due to the 0s applied to the *n* faulty wires in WG2. As the number of 1s on the outputs of WUTs_1 is odd, PCG_Parity_1 is 1. Since WUTs_Parity_1 and PCG_Parity_1 are different, the fault is detected. **QED**.

Notice that the proof of Theorem 2 uses a vector that sensitizes the faulty wires in only one of the WGs, namely WUTs_1. The same fault is also detected by another vector in Test 2 that sensitizes the faulty wires in the second WG, WUTs_2. In the same manner there are at least two vectors in Test 3 that detect the faults mentioned in Theorem 3.

Theorem 3 Test 3 can detect any wired-OR bridging faults between any *m* wires in WG1 and any *n* wires in WG2, where $1 \leq m, n \leq k$.

Proof: Similar to the proof of Theorem 2. **QED**.

Theorem 4 The parity-checking scheme with Test 1, Test 2 and Test 3 can detect the combinations of the faults in Theorems 1, 2 and 3 between WG1 and WG2.

Proof: When considering stuck-at faults and bridging faults that affect the same wires, there are six combinations of the two fault types, {stuck-at-0, wired-AND}, {stuck-at-0, wired-OR}, {stuck-at-1, wired-AND}, {stuck-at-1, wired-OR}, {stuck-at-0, stuck-at-1, wired-AND}, and {stuck-at-0, stuck-at-1, wired-OR}. The faulty outputs of wires with a fault combination above always behave as either stuck-at faults or bridging

faults since only one of the fault types can be dominating. Only bridging faults can share common wires between WG1 and WG2, and Theorems 2 and 3 have proven that bridging faults between the WGs can be detected. Therefore, only fault combinations within a WG need to be considered further. Theorem 1 shows that the combination of stuck-at and bridging faults can be detected. Thus, Tests 1, 2 and 3 can detect the combinations of the faults in Lemma 1, 2, 3 and 4 between WG1 and WG2. **QED**.

In summary, the proposed scheme has superior multiple-fault coverage compared to the classic walk-0/1 bus testing strategy [42]. This is achieved by using longer tests. However, the most notable characteristic of FPGA testing is that the configuration time required to program a device is much longer than the test application time. Therefore, several additional test vectors have little impact on the test time cost. The goal of FPGA testing is to minimize the number of test configurations rather than the number of test vectors.

4.1.7 Application

This section discusses the application of the proposed BIST scheme using the Xilinx XC4020E FPGA as the target device. Since the wire groups HS8, VS8, HD4, and VD4 in an XC4020E FPGA have dedicated switch matrices and share the same routing architecture [35], they can be tested in parallel. Table 4.2 shows the distribution of testing these wire groups and their respective programmable switches in the orthogonal, left- and right-diagonal test configurations. On the other hand, wire segments in HL6, VL6 and VG4 cross the entire chip and do not go through any switch matrices. Although in theory the testing of these long WGs can be merged with the three existing test configurations, they are not routable in practice due to the exhaustive usage of the single and double lines and the boundary routing resource in the three test configurations. This introduces the fourth test configuration for the long lines as shown in Table 4.2.

In a BIST environment, it is impossible to test interconnect resources without using CLBs, or vice versa. The proposed scheme uses CLBs to implement the BIST circuitry and parity checkers. It provides implicit testing of these functional modules. For instance, a faulty TPG generating m-bit erroneous test vectors can be detected as faulty wires, where $1 \le m \le k$. To reduce the fault escape probability hardware redundancy

techniques, such as *triple modular redundancy* (TMR), can be used to implement the BIST circuitry. This technique is feasible due to the availability of spare CLBs in the BIST section of an FPGA and the simplicity of our BIST circuitry.

Table 4.2: XC 4020E Global Routing Resources Test Configurations

Test Configuration	Component Type	Component Under Test
Orthogonal	PSs	WE, NS
	WGs	VS8, HS8, VD4, HD4
Left-diagonal	PSs	NW, SE
	WGs	VS8, HS8, VD4, HD4
Right-diagonal	PSs	WS, NE
	WGs	VS8, HS8, VD4, HD4
Long-lines	WGs	HL6, VL6, VG4

4.2 Combined Global and Local Interconnect Testing

This chapter is an extension of the work in section 4.1. A modified BIST scheme is provided that unifies the testing of both global and local interconnect. The modifications are based on the challenges that will be addressed in chapter 4.2.2 that combines the testing of both global and local interconnect.

4.2.1 Background Material

First, the following background material is provided in addition to the one given in section 4.1.

4.2.1.1 Local Interconnect

A detail diagram of the interconnect associated with a CLB of a Xilinx 4000E FPGA is shown in Figure 4.6.

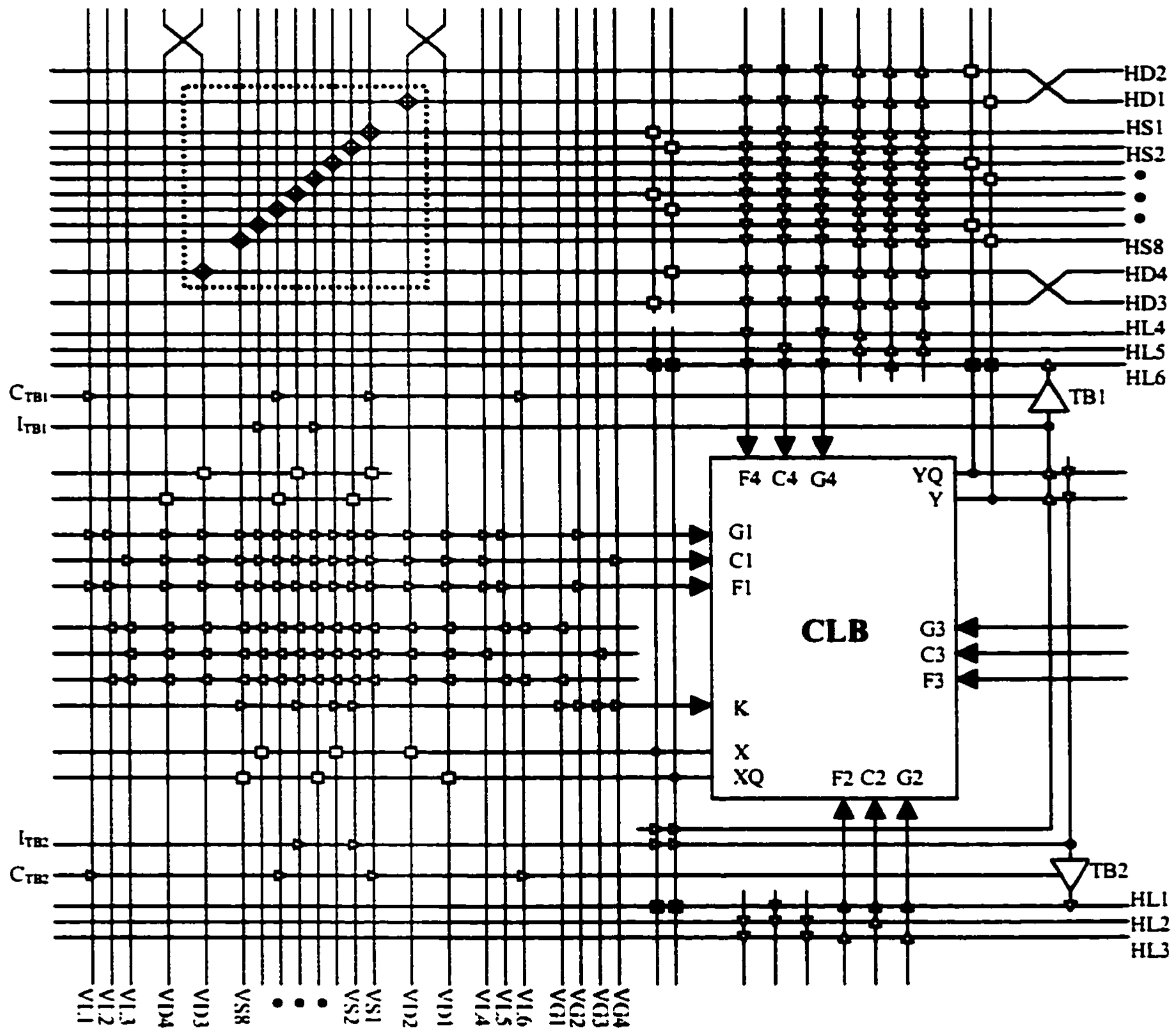


Figure 4.6: Detailed Diagram of Interconnect Associated with a CLB

Local interconnect includes pins of CLBs, and local programmable switches (basic and multiplexer PSs) that bring signals into and out of a CLB. As introduced in section 4.1, there are three types of programmable switches. Basic PSs are mainly located at the cross-points between wire segments and the output pins of a CLB, as shown in small square boxes in Figure 4.6. Multiplexer PSs are mainly located at the cross points between wire segments and the input pins of a CLB, as shown by small triangles in Figure 4.6.

4.2.1.2 Configurable Logic Block

Figure 3 shows a simplified logic diagram of a CLB. It has twelve inputs, four outputs, three function generators, G, F, and H, two D flip-flops (DFFs), and some multiplexers.

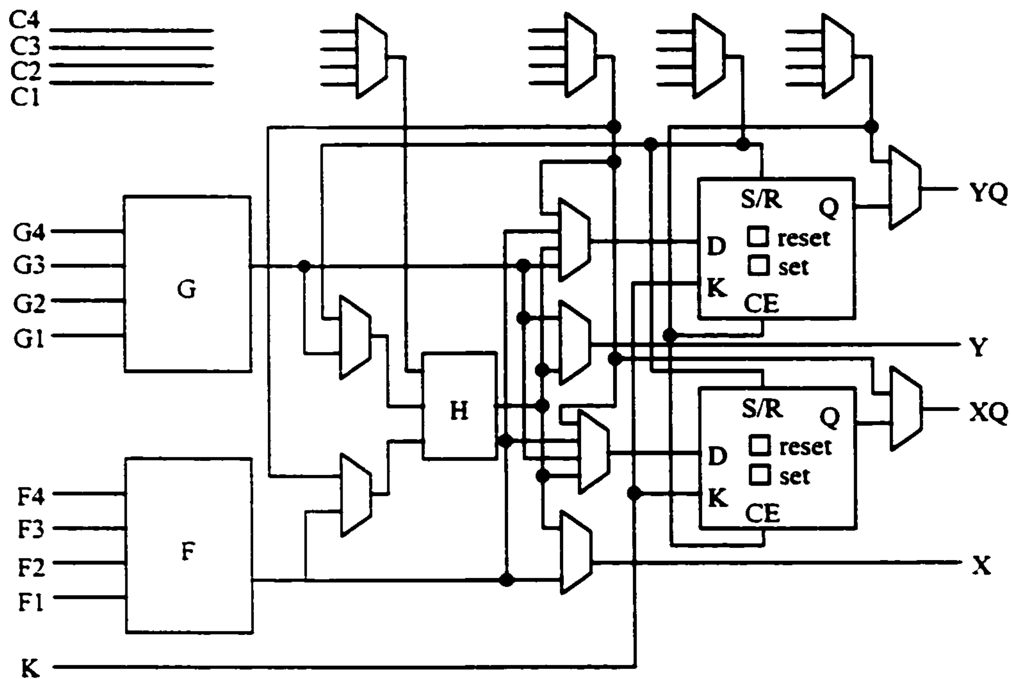


Figure 4.7: Simplified Logic Diagram of a CLB

The function generators can implement any five input logic function, and some further logic functions with up to nine variables. Each DFF has data input D, clock K, clock enable CE, and asynchronous set/reset S/R. S/R can be programmed to the set mode or reset mode in a configuration. S/R = 1 sets the DFFs in the set mode, and resets the DFF in the reset mode. The outputs X and Y are unlatched, and XQ and YQ can be either latched or unlatched by setting the respective output MUXes.

4.2.2 The Proposed BIST Scheme

4.2.2.1 Proposed Test Strategy

The proposed BIST strategy extends the work in section 4.1 on global interconnect testing to include local interconnect, so that the integrity of both global and local

interconnect resources of a device can be verified. It also makes use of in-system re-programmability of FPGAs to program a device under test into a set of minimum *test configurations* (TCs), perform deterministic tests on interconnect resources, and then program the device back into its intended system functions. The resources of an FPGA are partitioned into two equal sections: an upper half and a lower half. Each contains CLBs and interconnect. Interconnect resources under test are configured into busses. Dedicated device features, such as wired-AND busses and wide edge decoders, are used to access the test responses. When interconnect of the upper half are under test, the CLBs and the interconnect network in the lower half are used to implement the BIST circuitry, and vice versa. The BIST circuitry interacts with the external world through a built-in IEEE 1149.1 interface [35].

In section 4.1, the global interconnect are tested by configuring global wires and programmable switches into long busses, and their integrity is verified. The formation of the global busses does not require any CLB resources. On the contrary, local interconnect can only be tested indirectly by supplying inputs to CLBs and observing them at CLB outputs. This requires involving all the CLB resources in the section of the interconnect under test.

The parity-based error detecting BIST approach in section 4.1 requires using all three LUT function generators, or nine out of the twelve inputs of a CLB. This maximizes the number of global wires to be tested in parallel. The multiplexer switches connecting to a CLB input pin contains up to seventeen programmable switches. Each can be selected (or tested) in a separate TC. Therefore, the testing challenge for the combined global and local interconnect is to derive a minimum of seventeen TCs that can simultaneously verify the integrity of the local and global interconnect resources. To maximize the number of programmable switches to be tested in parallel in a TC, interconnect associated with a CLB is partitioned into two parts, the combinational part and the sequential part. The combinational part includes the three function generators, some input and output MUX PSs and an unlatched output (X or Y). The sequential part consists of some input MUX PSs and a D flip-flop (DFF) that has three control inputs, input D, clock enable CE, and set/reset S/R, and one latched output (XQ or YQ). This allows for using all the twelve inputs of a CLB

during a TC (nine and three for the combinational part and sequential part respectively), thus contributes to the derivation of the minimum number of TCs.

Figure 4.8 shows the conceptual block diagram of the proposed BIST strategy. A portion of wire segments and programmable switches are configured to form two *wire groups* (WGs), WG_c and WG_s, denoting the WGs being tested by the combinational part and sequential part respectively. A WG_c contains eight *wires under test* (WUTs_c), and one wire conveying the parity bit (WUTs_Parity_c). A WG_s contains 3 WUTs (WUTs_s) providing inputs to the D, CE and S/R inputs of the DFF.

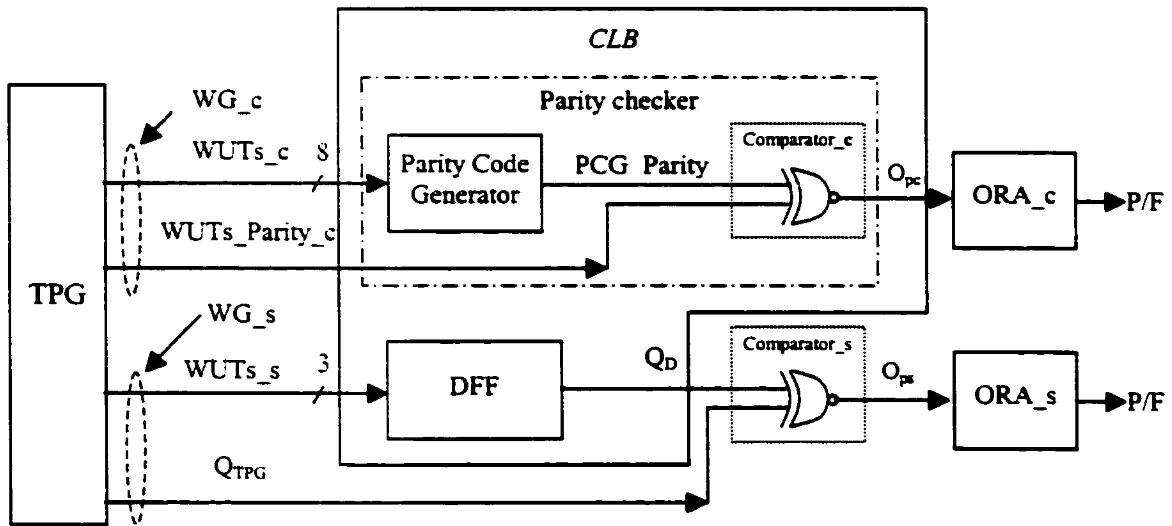


Figure 4.8: Conceptual Block Diagram of the Proposed BIST Strategy

The *test pattern generator* (TPG) supplies the WUTs_c and WUTs_Parity_c with test vectors at one end of the wires. Similar to section 4.1, WUTs_Parity_c is the parity of each sub-test vector applied to WUTs_c. A parity-checker at the other end of the wire busses computes the parity bit of WUTs, named PCG_Parity, and compares the PCG_Parity with the WUTs_Parity_c. A disagreement in the two indicates an error in the WG. Figure 4.8 is an even parity implementation of the parity checker. The PCG is an 8-input XOR function while the Comparator_c is a 2-input XNOR gate. An odd parity-checking scheme can be easily implemented with minor modifications. A logic

1 on the output of the Comparator_c, Opc, indicates error-free and a logic 0 means that an error is detected.

The TPG supplies WG_s with test vectors at one end of the three wires and the fault-free output, Q_{TPG}, for the sub-vector applied to WUTs_s. The output response analyzer for the DFF (ORA_s) compares the output of Q_D with Q_{TPG}. Similar to Comparator_c, Comparator_s is a 2-input XNOR gate. However, it is located in the BIST section of the TC due to the fact that all the CLB resources in the WUTs section are fully utilized in the proposed BIST strategy.

The fault detectability of the WG_c and WG_s groups is slightly different since distinct test methods are used. For the wires and PSs in WG_c, the proposed BIST scheme is capable of detecting all single and *m* multiple stuck-at 0/1 faults, stuck-open faults and bridging faults in wire segments, stuck-on/off faults and bridging faults in PSs, and the combination of these faults, where $1 \leq m \leq 8$, as proved in section 4.1. The wires and PSs in WG_s are tested indirectly by means of the complete functional test of a DFF that receives the test signals from WUTs_s. It will be shown in section 4.2.5 that the proposed BIST strategy for WG_s can detect all single and multiple stuck-at, stuck-open, and bridging faults in wire segments, stuck-on/off and bridging faults in programmable switches, and the combinations of these faults in a WG_s, and all the modeled bridging faults between a WG_c and a WG_s.

4.2.3 Proposed BIST Architecture

Figure 4.9 shows the proposed BIST architecture. The interconnect under test section contains a $n\times(n/2)$ array of CLBs, where n is the number of CLBs columns in a device. Each CLB implements a parity checker for WG_c and a DFF for WG_s as shown in Figure 4.8. The X and XQ outputs of a CLB symbolically represent the output of the parity checker for WG_c, O_{pc}, and the output of the DFF for WG_s, Q_D, respectively (see Figure 4.8). In a TC, they could be any combination of X, Y and XQ, YQ outputs. The O_{pc} and Q_D outputs on each row are first connected to the respective wired MUX-bus through the tri-state buffers at the outputs, then to the wide-edge decoder. The select signal S_i, where $1 \leq i \leq n$, allows the two outputs, O_{pc} and O_D, of the CLBs in

column i to be connected to their respective edge decoders implementing wired-AND and wired-NOR functions.

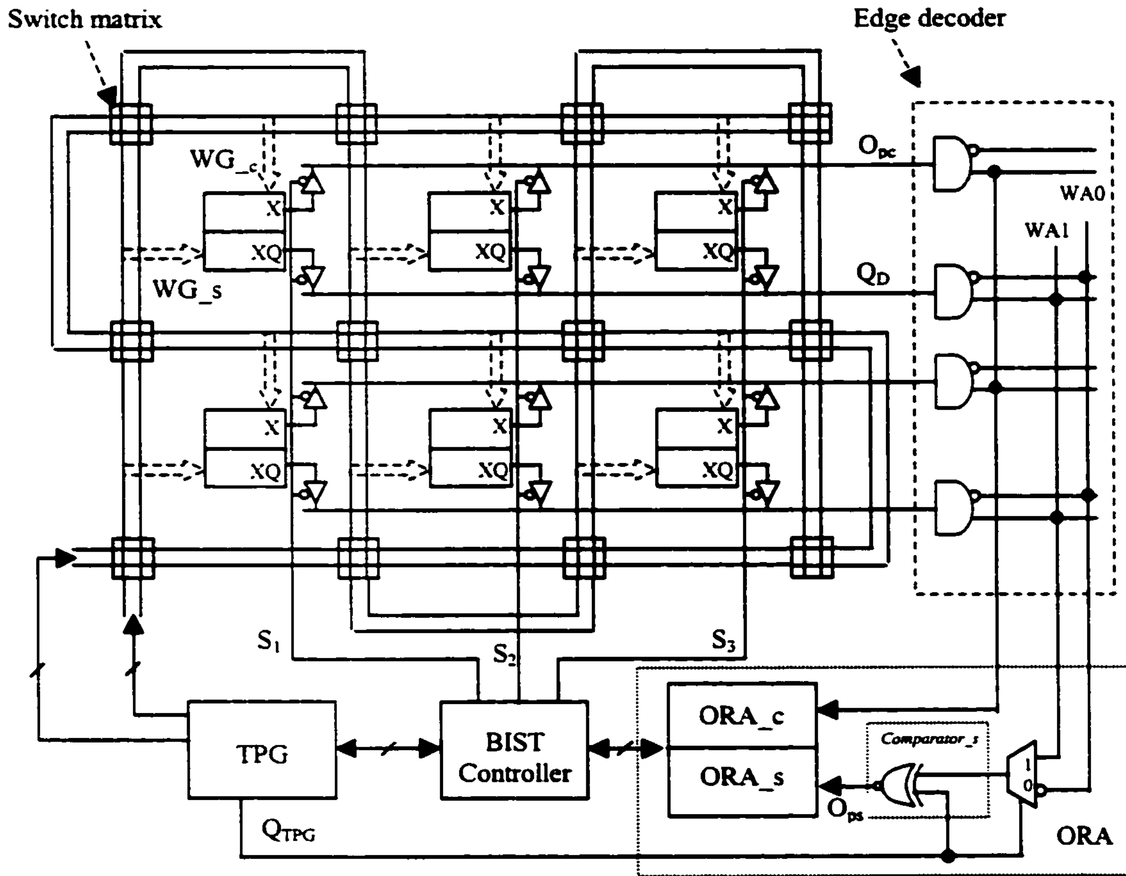


Figure 4.9: High-level BIST Architecture

The wide-edge decoder collects the WG_c outputs on column S_i by implementing the wired-AND function of all the O_{pc} signals on column S_i. Since the fault-free output of O_{pc} is logic 1, any number of faulty output(s) can be detected as long as there is one fault free output in column S_i.

The wide-edge decoder collects the WG_s outputs on column S_i by implementing wired-AND and wired-OR (wired_NOR logic combined with inverse logic) functions of all the Q_D signals column S_i. The wired-AND and wired-OR busses, WA1 and WA0, are used to provide O_{ps} signals for the fault-free '1' and '0' of Q_D signals respectively.

The wired-AND/OR tree structure used to collect Q_D signals has strong inherited multiple fault detectability. It guarantees the detection of any multiple faulty outputs from CLBs if at least one signal connecting to the busses is fault-free [44]. The fault-free WG_s signal, Q_{TPG}, selects either WA0 or WA1. The Comparator_s compares the selected WA1 or WA0 signal with Q_{TPG}. A disagreement indicates an error in WG_s and sets O_{ps} to logic '0'. The ORA_c and ORA_s pass on their respective pass/fail output signals to the BIST controller.

In a test configuration, test signals are broadcast from the TPG to all the WG_c and WG_s (represented in dashed arrows in Figure 4.9) of the CLBs in the interconnect under test section. It is achieved by forming global busses using the programmable switch matrices. This allows the programmable switches in the switch matrices and in the WG_c and WG_s to be tested in parallel. A fault in a switch matrix can be propagated to many CLBs and be tested. In section 4.1, it has been shown that a six-transistor cross-point PS in a switch matrix can be tested in three TCs, named orthogonal, left-diagonal, and right-diagonal. Figure 4.9 shows a snake-like global bus structure and the orthogonal test configuration using the NS and WE PSs for the broadcast. Similarly, the left-diagonal and right-diagonal TCs can be used to test the NW, SW, NE and SE PSs. Later, in section 4.2.6.3, it will be shown that the three TCs required to test the switch matrices can be distributed into the seventeen TCs for WG_c and WG_s, and have no impact on the overall test time.

4.2.4 Fault Models and Assumptions

The fault models under consideration are the same as the ones proposed in section 4.1.3. The following assumption is made in addition to the assumptions made in section 4.1.3.

Assumption 6 At least one CLB output connected to the MUX busses is fault-free.

Under this assumption, any multiple faulty CLB outputs connected to the MUX busses can be detected by the AND/OR tree structure. It is realistic to assume that at least one of the signals on the busses is fault-free in the BIST environment, since a

device has been tested before being placed in a system, and the probability of massive failure affecting all the MUX busses is negligible.

4.2.5 Test Sets and Fault Detectability

4.2.5.1 Test Sets

The proposed tests include three test sets, called Test 1, Test 2 and Test 3.

Test 1 is designed to test faults within WG_c and WG_s groups. The Test 1 for a WG_c, denoted as $Test1_{WG_C}$, is an exhaustive test of 2^k of exhaustive test vectors and the parity-bit, where k = 8. Test 1 for a WG_s, denoted as $Test1_{WG_S}$, is a binary counting sequence from 000 to 111 on the D, CE and S/R inputs of the DFF, and the corresponding fault-free Q output sequence. Since $Test1_{WG_C}$ consists of $2^k = 256$ test patterns, $Test1_{WG_S}$ is repeated 256/8 =32 times during the application of $Test1_{WG_C}$. The redundant 31 sets of the tests don't affect the fault detectability of the test on WG_s.

Test 2 is to test any Wired-AND bridging faults between WG_c and WG_s. $Test2_{WG_C}$ consists of eight all-zero tests on the k+1 WG_c wires (the parity of an all-zero vector is zero). Test 2_{WG_S} includes an exhaustive test on WG_s and their corresponding fault-free Q output sequence.

Test 3 is to test any Wired-OR bridging faults between WG_c and WG_s. $Test2_{WG_C}$ consists of eight all-one tests on WUT_s with the all-zero parities, and eight walk-1 tests with the all-one parities. Test 3_{WG_S} is the same as Test 2_{WG_S}, but repeats twice. The two sets in $Test2_{WG_C}$ allow all the k+1 wires in WG_c carrying an all-one sequence during application of an exhaustive test to WG_s.

Table 4.3 shows the proposed tests, where the Q_{PTG} is for the reset mode of the DFF (see the detailed discussions in section 4.2.5.2). The fault-free sequence of Q_{PTG} in the set mode can be found in Table 4.4. Note that no test pattern compaction is applied since the time required to apply the test patterns is magnitudes smaller than the time to program a device, and it is easier to generate exhaustive test patterns in the BIST environments.

Table 4.3: The Proposed Three Tests

			Test 1	Test 2	Test 3
WG_c	WUTs_c		0000000000 ••• 11	00000000	11111111 10000000
			0000000000 ••• 11	00000000	11111111 01000000
			0000000000 ••• 11	00000000	11111111 00100000
			0000000000 ••• 11	00000000	11111111 00010000
			0000000011 ••• 11	00000000	11111111 00001000
			0000111100 ••• 11	00000000	11111111 00000100
			0011001101 ••• 11	00000000	11111111 00000010
			0101010110 ••• 01	00000000	11111111 00000001
	WUTs_Parity_c		0110100100 ••• 10	00000000	00000000 11111111
WG_s	WUTs_s	D	0000111100 ••• 11	00001111	00001111 00001111
		CE	0011001100 ••• 11	00110011	00110011 00110011
		S/R	0101010101 ••• 01	01010101	01010101 01010101
	Q_{TPG}		0000001000 ••• 10	00000010	00000010 00000010

4.2.5.2 Fault Detectability

As shown in chapter 4.2.2.1, distinctive testing methods were used to test WG_c and WG_s in order to test the maximum number of wire segments in parallel to minimize the number of test configurations and to achieve minimal test time. As a result, separate treatments are required to study the fault detectability for WG_c and WG_s.

(1) Fault Detectability for WG_s

Theorem 5 Test 1_{WG_C} can detect any combination of single or multiple segment stuck-at-0/1 and stuck-open faults, programmable switch stuck-off faults and/or wire bridging faults affecting any m wires of a WG_c, where $1 \le m \le k$ [31].

(2) Fault Detectability for a WG_s

The WG_s is tested by means of the functional test of a DFF with asynchronous set/reset and edge-triggered clock. An exhaustive test is applied to the three WUTs_s connecting to the three control inputs, set/reset input S/R, data input D, and clock enable CE, of the DFF and observe its Q output directly on a MUX bus. The DFF can be configured to reset (or set) mode through a device configuration. Under the chosen DFF reset (or set) configuration, an input vector, D, CE, S/R = x x 1 can be applied to

the DFF to initialize the DFF prior to the application of the test for WUTs_s. The fail of set/reset can be observed immediately on the MUX bus. Therefore, the initialization and observability problems for sequential circuit testing are avoided here although a DFF is used to test the wire segments/programmable switches connecting to the input pins of the CLB. This greatly simplifies the testing problem for WUTs_s.

Simple enumeration is used to examine the fault detectability of the three-input circuit. There are forty-six modeled faults, including the single and multiple stuck-at, stuck-open, and bridging faults in wire segments, stuck-on/off and bridging faults in programmable switches, and the combinations of the faults in a WG_s. For both set and reset mode of a DFF:

(a) The fault-free Q values are derived with respect to each vector in an exhaustive test ($2^3 = 8$ vectors), as shown in Table 4.4.

(b) The Q values of the DFF are derived for each modeled fault under the same test sequence, as shown in Table 4.4

To detect the modeled fault(s), the Q sequence of the fault must differ from that of the fault-free Q sequence by at least one bit. The bits shown in italics in Table 4.4 are the ones that differ from the fault-free sequences.

It can be seen from Table 4.4 that the exhaustive test can detect 45 out of 46 modeled faults in set mode only, or reset mode only, but all the modeled faults can be detected in both set and reset modes. Therefore, both configuration modes are required in order to test all the modeled faults.

Theorem 6 Test 1_{WG_S} can detect any single or multiple segment stuck-at-0/1 and stuck-open faults, programmable switch stuck-off faults, wire bridging faults and the combinations of the faults above in a WG_s in the Reset and the Set modes of a DFF.

Proof: By enumerating all of the faults (see Table 4.4) and by proving it is true for all the modeled faults in the reset and the set modes of a DFF. **QED.**

Table 4.4: The Enumeration of Q Output Sequences of a DFF

		Reset mode	Set mode
Q SEQUENCE	Fault-free	00000010	11011111
	D s-a-0	000000***00***	111011***01***
	D s-a-1	00***1***00010	11***1***11111
	CE s-a-0	000000***00***	11***1***11111
	CE s-a-1	0000***1***010	***0***1011111
	S/R s-a-0	***11***00001***1***	11***0000***11
	S/R s-a-1	000000***00***	11***1***11111
	D s-a-0, CE s-a-0	000000***00***	11***1***11111
	D s-a-0, S/R s-a-0	000000***00***	***00000000***
	CE s-a-0, S/R s-a-0	000000***00***	11***1***11111
	D s-a-0, CE s-a-1	000000***00***	***0***101***0***1***01***
	D s-a-0, S/R s-a-1	000000***00***	11***1***11111
	CE s-a-0, S/R s-a-1	000000***00***	11***1***11111
	D s-a-1, CE s-a-0	000000***00***	11***1***11111
	D s-a-1, S/R s-a-0	***11111111***	11***1***11111
	CE s-a-1, S/R s-a-0	0000***1111***	***0000***1111
	D s-a-1, CE s-a-1	***1***0***1***0***1***010	11***1***11111
	D s-a-1, S/R s-a-1	000000***00***	11***1***11111
	CE s-a-1, S/R s-a-1	000000***00***	11***1***11111
	D s-a-0, CE s-a-0, S/R s-a-0	000000***00***	11***1***11111
	D s-a-0, CE s-a-0, S/R s-a-1	000000***00***	11***1***11111
	D s-a-0, CE s-a-1, S/R s-a-0	000000***00***	***00000000***
	D s-a-0, CE s-a-1, S/R s-a-1	000000***00***	11***1***11111
	D s-a-1, CE s-a-0, S/R s-a-0	000000***00***	11***1***11111
	D s-a-1, CE s-a-0, S/R s-a-1	000000***00***	11***1***11111
	D s-a-1, CE s-a-1, S/R s-a-0	***11111111***	11***1***11111
	D s-a-1, CE s-a-1, S/R s-a-1	000000***00***	11***1***11111
	D bridges with CE (wired-AND)	00000010	11***1***11111
	D bridges with CE (wired-OR)	00***1***0***1***010	11***1***11111
	D bridges with S/R (wired-AND)	000000***00***	11***000***1***0***1
	D bridges with S/R (wired-OR)	000000***00***	11011111
	CE bridges with S/R (wired-AND)	000000***00***	11***1***11111
	CE bridges with S/R (wired-OR)	000000***00***	11***1***11111
	D bridges with CE and S/R (wired-AND)	000000***00***	11***1***11111
	D bridges with CE and S/R (wired-OR)	000000***00***	11***1***11111
	S/R s-a-0, and D bridges with CE (wired-AND)	0000001***1***	11***1***11111
	S/R s-a-0, and D bridges with CE (wired-OR)	***11111111***	11***1***11111
	S/R s-a-1, and D bridges with CE (wired-AND)	000000***00***	11***1***11111
	S/R s-a-1, and D bridges with CE (wired-OR)	000000***00***	11***1***11111
	CE s-a-0, and D bridges with S/R (wired-AND)	000000***00***	11***1***11111
	CE s-a-0, and D bridges with S/R (wired-OR)	000000***00***	11***1***11111
	CE s-a-1, and D bridges with S/R (wired-AND)	000000***00***	***00000***1***0***1
	CE s-a-1, and D bridges with S/R (wired-OR)	000000***00***	***0***1011111
	D s-a-0, and CE bridges with S/R (wired-AND)	000000***00***	11***1***11111
	D s-a-0, and CE bridges with S/R (wired-OR)	000000***00***	11***1***11111
	D s-a-1, and CE bridges with S/R (wired-AND)	000000***00***	11***1***11111
	D s-a-1, and CE bridges with S/R (wired-OR)	000000***00***	11***1***11111

Note that the clock input is not included in the DFF. A fault on the clock input can be easily detected. Similarly, a failure to set/rest a DFF can be observed and detected on the MUX busses.

(3) Fault Detectability between WG_c and WG_s

Theorem 7 Test 2 can detect any wired-AND bridge faults between a WG_c and a WG_s in the reset and the set modes of a DFF.

Proof: Consider a wire, *i*. in WG_c, and a wire, *j*, in WG_s. The all-zero bits of $Test_2_{WG_c}$ on wire *i* can be considered as a stuck-at 0 fault on wire *j* because of the wired-AND bridging between the two wires. And the stuck-at fault can be detected by $Test_2_{WG_s}$, which is identical to $Test_1_{WG_s}$, by Theorem 2. **QED.**

Theorem 8 Test 3 can detect any wired-OR bridge faults between a WG_c and a WG_s in the reset and the set modes of a DFF.

Proof: Consider a wire. *i*. in WG_c, and a wire, *j*, in WG_s. The all-one bits of $Test_3_{WG_c}$ on wire *i* when the exhaustive test is applied to WG_s can be considered as a stuck-at 1 fault on wire *j* because of the wired-OR bridging between the two wires. The stuck-at fault can be detected by $Test_3_{WG_s}$, which is identical to $Test_1_{WG_s}$, by Theorem 2. **QED.**

4.2.6 Minimum Test Configurations

4.2.6.1 Interconnect Associated with a CLB

As shown in Figure 4.6 on page 52, the local interconnect associated with a CLB includes twelve input pins, four output pins, two control inputs and two data inputs of the tri-state buffers, and the basic (MUX-based) PSs on the output (input) pin segments. Table 4.5 lists the element types and the signals associated with them.

Table 4.5: Interconnect Associated with a CLB

<table>
<tr><td rowspan="10">Local interconnect associated with a CLB</td><td rowspan="2">Pin segments</td><td colspan="2">Input</td><td>F1-F4, G1-G4, C1-C4</td></tr>
<tr><td colspan="2">Output</td><td>X, XQ, Y, YQ</td></tr>
<tr><td>Other wires</td><td colspan="2">Inputs of tri-state buffers</td><td>C_{TB1}, C_{TB2}, I_{TB1}, I_{TB2}</td></tr>
<tr><td rowspan="7">Programmable switches</td><td rowspan="5">PSs associated with input / output pins</td><td>17-to-1 MUX PS</td><td>G1, G3, F1, F3</td></tr>
<tr><td>16-to-1 MUX PS</td><td>G2, G4, F2, F4, C2, C4</td></tr>
<tr><td>15-to-1 MUX PS</td><td>C1, C3</td></tr>
<tr><td>8-to-1 MUX PS</td><td>K</td></tr>
<tr><td>8 basic PSs</td><td>X, XQ, Y, YQ</td></tr>
<tr><td rowspan="2">PSs associated with control inputs of two 3-state buffers</td><td>4-to-1 MUX PS</td><td>C_{TB1}, C_{TB2}</td></tr>
<tr><td>6-to-1 MUX PS</td><td>I_{TB1}, I_{TB2}</td></tr>
</table>

4.2.6.2 Derivation of the Test Configurations

4.2.6.2.1 Discussion of the minimum number of test configurations

It has been shown in [43] that all of the global interconnect can be tested in three configurations (theoretically) and in four configurations (in practice) due to limited boundary connection resources.

For local interconnect, we need w_{max} test configurations, where w_{max} is the maximum value of the w-to-1 multiplexer PS in local interconnect [45]. w_{max} = 17 in XC4000E series FPGAs and therefore the minimum number of TCs is 17. One set of 17 TCs was derived that can test all the global and local routing resources, as shown in Appendix 3. Therefore, 17 is the minimum number of test configurations required.

4.2.6.2.2 Selection of the interconnect under test

As discussed in chapter 4.2.2.1, the resources of an FPGA are partitioned into two equal sections: one section contains interconnect under test, and the other is used to implement the BIST circuitry. In the section that contains interconnect under test, local interconnect associated with each CLB is tested in parallel, as shown in Figure

4.9 on page 57, and the minimum number of test configurations developed for local interconnect associated with one CLB is the minimum number of test configurations for the whole local interconnect under test. Therefore, the selection of interconnect under test targets the local interconnect associated with one CLB.

The local interconnect in Xilinx XC4000E FPGAs can be modeled as a matrix with M rows and N columns, S[M][N], where M (equals to 21) is the total number of inputs and outputs associated with a CLB (including 13 input pins, 2 control inputs and 2 data inputs of tri-state buffers, and 4 output pins). Parameter N (equals to 40) is the total number of different wire segments associated with a CLB (including 8 horizontal single lines, 8 vertical single lines, 4 horizontal double lines, 4 vertical double lines, 6 horizontal long lines, 6 vertical long lines, and 4 vertical global lines). For each S[m][n], where $0 \le m < M$ and $0 \le n < N$, logic value 1 is assigned if there exists a MUX-based PS, and logic value 0 is assigned if there is no switch at S[m][n].

The selection of interconnect under test in each of the minimum test configurations is based on the following constraints. These constraints are proposed due to (1) the requirement of minimum number of TCs; (2) the limitation of the proposed scheme; and (3) the device features in the FPGA interconnect.

Constraint 1: In each test configuration, each input pin of a CLB must be connected to a wire segment through the MUX PSs to maximize the amount of local interconnect under test, until all the PSs on the input pins have been selected (tested).

This constraint is proposed in order to minimize the number of test configurations. It translates to the selection of each input row in S[M][N] until all the PSs in that row have been selected.

Constraint 2: The signals on all the inputs must be independent of each other (except for inputs C_{TB1} and C_{TB2} which share the same column select signal). That is, a test signal can not be applied to more than one input pin of a CLB in a test configuration.

This constraint is due to the limitation of the proposed BIST scheme, which may fail to detect some faults when input test signals are not independent. Constraint 2

indicates that the selection of the same column for two or more input rows in S[M][N] is prohibited.

Note that in left-diagonal and right-diagonal test architectures, a horizontal single line HS_i is connected to a vertical single line VS_i, and a horizontal double line HD_j is connected to a vertical double line VD_j, where $1 \leq i \leq 8$, and $1 \leq j \leq 4$. It is unknown which type of architecture (orthogonal, left-diagonal, or right-diagonal) is used before merging the test of the cross-point PSs in switch matrices. Therefore, to ease the later merging process, the selection of the two rows that connect to HS_i and VS_i, or connect to HD_i and VD_i is also prohibited.

Constraint 3: The selection of two or more PSs in an input row is prohibited in a test configuration.

This is due to the feature of the MUX-based programmable switches described in chapter 4.1.1.1.

Constraint 4: A basic PS on an output pin is tested by connecting the signal on the output pin segment to an input pin through the basic PS on the output pin segment and the multiplexer PS on the input pin segment.

Under constraint 4, the basic PSs located at the same position of different CLB output pin segments are tested in parallel, or in one TC. For example, in Figure 4.6 on page 52, every basic PS that connects output pin X and vertical single line VS7 can be tested by forming the path that connects X to VS7, and then to C1. Otherwise, the number of TCs for these basic PSs will be linear in the array size of CLBs. These PSs are shown in Figure 4.6 on page 52 as little square boxes without fill-in color. Note that the proper MUX configuration within a CLB can guarantee the signal independence within a WG_c or a WG_s introduced by the connection and the fault detectability for a WG_c or a WG_s is not affected. Figure 4.10 shows such an example. Figure 4.10 (a) shows how an original wire group is formed without testing the basic PS on any output pin, and Figure 4.10 (b) shows how the same wire group is formed without signal dependence among its inputs when including the test of a basic PS on the output pin X.

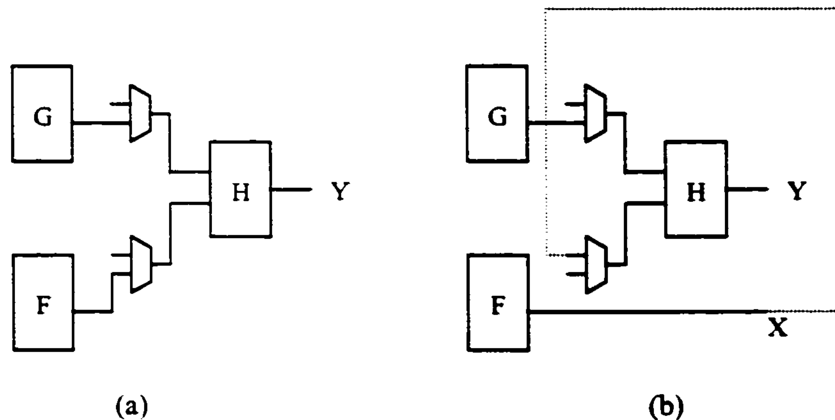


Figure 4.10: An Example

In Xilinx XC4000E FPGAs, eight of twelve CLB input pins have less than 17 MUX PSs that can be tested in less than 17 TCs. After these pins were tested, they can be used to test the basic PSs on the output pins. When forming a feedback path to test a basic switch, the number of test signals required from the TPG will be reduced and more than one basic PSc can be tested in a TC. The formation of both combinational and sequential wire groups is not necessary for all the 17 TCs. When all the PSs on an input pin are tested, there is no need to repeat the test in the later TCs.

One set of 17 TCs was derived manually based on the above constraints, as shown in Appendix 3. In the derived 17 TCs, trade-offs were made between the number of TCs and the test coverage, because some PSs are hard to test under the current design architecture of XC4000E FPGAs. These PSs are shown in Figure 4.6 as gray-shaded little square boxes or triangles.

It can be seen from the proposed BIST scheme (see Figure 4.9) that the parallel test of interconnect requires the use of MUX busses, that is, horizontal long lines HL1 and HL6 (see Figure 4.6) in each horizontal channel of an FPGA. Also, the proposed scheme in Figure 4.9 is C-testable, that is, the number of TCs is independent of the array size of FPGA interconnect.

However, when the testing of PSs located on HL1 and HL6 (except the PSs that connect the output of tri-state buffers to HL1/HL6) is included, it requires the use of

HL1 and HL6 which prohibits the forming of MUX busses, and thus prohibits the parallel test of interconnect associated with each CLB. If these PSs are tested, at most one of the basic PS on HL1/HL6 can be programmed on (connected to HL1/HL6) in a TC because the direct merging (that is, not through tri-state buffers) of multiple outputs is prohibited by the tool. Therefore the minimum number of TCs will be $17 + x$, where x is the maximum number of basic PSs on a horizontal long line, HL1 or HL6. Also, the FPGA interconnect becomes linearly testable, that is, the number of TCs is linear in the array size of an FPGA. Take the XC4020E device, for example. Here x equals to 112 (4 PSs associated with each CLB times 28 CLBs in a row), and the minimum number of TCs will be $17 + 112 = 129$. It can be seen that the number of TCs (or the testing cost) increases by more than 750% while these PSs occupy less than 4.5% of the PSs associated with a CLB.

4.2.6.3 Merging the Test for Global Interconnect

The complete test of local interconnect also tests all the wire segments in global interconnect that are used to propagate test signals, and all the cross-point PSs are tested if the WG structures used to propagate the test signals are properly chosen.

Note that in Xilinx 4000E series FPGAs, there are 10 cross-point PSs in each switch matrix, with each cross-point PS having six basic PSs: WE, NS, NW, SE, and WS, NE. Eight of the 10 cross-point PSs are used to connect single length lines while the remaining 2 are used to connect double length lines. When global and local interconnect are tested simultaneously, not all of the WE and NS (NW and SE, or WS and NE) PSs in a switch matrix are tested under the orthogonal (left-diagonal, or right-diagonal) structure in a test configuration since only part of the single and double length lines are used to propagate test signals to the input pins of a CLB. Therefore, several other orthogonal (left-diagonal, or right-diagonal) test configurations are required to test all the WE and NS (NW and SE, or WS and NE) PSs in a switch matrix.

Table 4.6 shows the single and double length lines that are tested (selected) in each of the 17 test configurations. The selection of the cross-point PSs in a TC is based on the following two rules:

Rule 1: At most eight cross-point PSs are selected in each TC.

Note that in the proposed BIST scheme, the cross-point PSs are used to form global wires to broadcast the test signals, and these wires need to go through the boundaries of an FPGA (see Figure 4.9). Rule 1 is proposed due to limitations of the boundary resources, which allow at most eight wires to be routed through them.

Table 4.6: Single and Double-Length Lines Tested in 17 Test Configurations

TC #	HS8								HD4				VS8								VD4			
	1	2	3	4	5	6	7	8	1	2	3	4	1	2	3	4	5	6	7	8	1	2	3	4
1						√	√	√					√				√					√		
2					√		√		√		√	√	√					√		√				
3	√			√				√		√					√				√		√		√	
4	√				√						√				√	√				√				
5		√						√		√		√				√	√				√		√	
6			√	√			√					√						√					√	
7					√			√	√	√							√	√	√					√
8						√			√		√		√						√	√		√		
9	√					√			√	√				√			√			√		√		
10		√						√			√	√				√					√	√		
11	√		√		√									√				√				√		
12		√				√		√		√			√		√								√	√
13	√		√		√	√						√		√					√					
14			√	√							√		√						√				√	√
15	√	√		√			√								√		√			√				
16														√		√		√			√		√	
17																√								

Rule 2: Each column in Table 4.6 (or each wire segment in single and double wire group) must be selected at least three times (or in three TCs).

Rule 2 is proposed because we need at least three test configurations to test a cross-point PS. See section 4.1.4 for details.

In Table 4.6, HS8 (VS8) means 8 horizontal (vertical) single length lines, and HD4 (VD4) means 4 horizontal (vertical) double length lines. The number i ($1 \le i \le 8/4$ for single/double length lines) in the second row of Table 4.6 represents the ith line from top to bottom (right to left) of the corresponding horizontal (vertical) wire group shown in Figure 4.6, and the symbol "√" indicates that the corresponding line is tested.

Table 4.7 shows the required 16 test configurations that cover the test of all the cross-point PSs. In Table 4.7, WE, NS (NW, SE, or WS, NE) indicate the corresponding wire segments under test (marked with "√") using WE, NS (NW, SE, or WS, NE) switches to form the wire groups. These PSs are tested when the wire groups are tested. It can be seen from Table 4.7 that all the cross-point PSs are tested since each WE, NS (NW, SE, or WS, NE) switch has at least one "√" symbol.

Table 4.7: 16 Test Configurations That Test All Cross-Point PSs

(a) TCs test all WE and NS PSs

TC #	HS8								HD4				VS8								VD4			
	1	2	3	4	5	6	7	8	1	2	3	4	1	2	3	4	5	6	7	8	1	2	3	4
	WE and NS PSs																							
2					√		√		√		√	√	√					√		√				
3	√			√				√		√					√				√		√		√	
9	√					√			√	√				√			√			√		√		
10		√						√			√	√				√					√	√		
14			√	√							√		√						√				√	√

(b) TCs test all NW and SE PSs

TC #	HS8								HD4				VS8								VD4			
	1	2	3	4	5	6	7	8	1	2	3	4	1	2	3	4	5	6	7	8	1	2	3	4
	NW and SE PSs																							
5		√						√		√		√				√	√				√		√	
6			√	√			√					√						√					√	
8						√			√		√		√						√	√		√		
11	√		√		√									√				√				√		
12		√				√		√		√			√		√								√	√

(c) TCs test all **WS and NE PSs**

TC #	HS8								HD4				VS8								VD4			
	1	2	3	4	5	6	7	8	1	2	3	4	1	2	3	4	5	6	7	8	1	2	3	4
	WS and NE PSs																							
1						√	√	√					√				√					√		
4	√				√						√				√	√				√				
7					√			√	√	√							√	√	√					√
13	√		√		√	√						√		√					√					
15	√	√		√			√								√		√			√				
16														√		√		√			√		√	

Since the number of test configurations to test all interconnect in one half of XC4000E devices is 17, a total of 34 test configurations is required for a complete test of all the global and local interconnect, which is also the minimum for a BIST approach.

Table 4.8 and Figure 4.11 show the relationship between the number of TCs from our set and the test coverage of interconnect when using the XC4020E as the device model. The total numbers of PSs and wire segments in the interconnect resources in the XC4020E are 244,608 and 47,824 respectively. It can be seen that all the wire segments in one half of the FPGA are tested in the first 7 TCs. The first 17 TCs test about 95.51% PSs and 129 TCs are needed to test all the PSs in one half of the FPGA.

The faults that can be tested in the PSs and wire segments are: (1) any combination of single or multiple segment stuck-at-0/1 and stuck-open faults, programmable switch stuck-off faults and/or wire bridging faults that affect any eight wires of a WG_c and any three wires of a WG_s; (2) any modeled bridging faults between a WG_c and a WG_s.

Table 4.8: Number of TCs vs. Test Coverage

Number of TCs	Test Coverage	
	PSs (%)	Wire Segments (%)
1	7.37	**62.90**
2	15.38	**73.77**
3	23.72	**88.52**
4	31.73	**95.08**
5	38.14	**95.08**
6	43.59	**96.72**
7	48.72	**100**
8	55.13	**100**
9	60.58	**100**
10	66.35	**100**
11	71.47	**100**
12	75.96	**100**
13	80.45	**100**
14	85.26	**100**
15	90.06	**100**
16	92.63	**100**
17	95.51	**100**
18	95.56	**100**
19	95.60	**100**
20	95.65	**100**
21	95.67	**100**
33	96.15	**100**
45	96.63	**100**
57	97.12	**100**
69	97.60	**100**
81	98.08	**100**
93	98.56	**100**
105	99.04	**100**
117	99.52	**100**
129	100	**100**

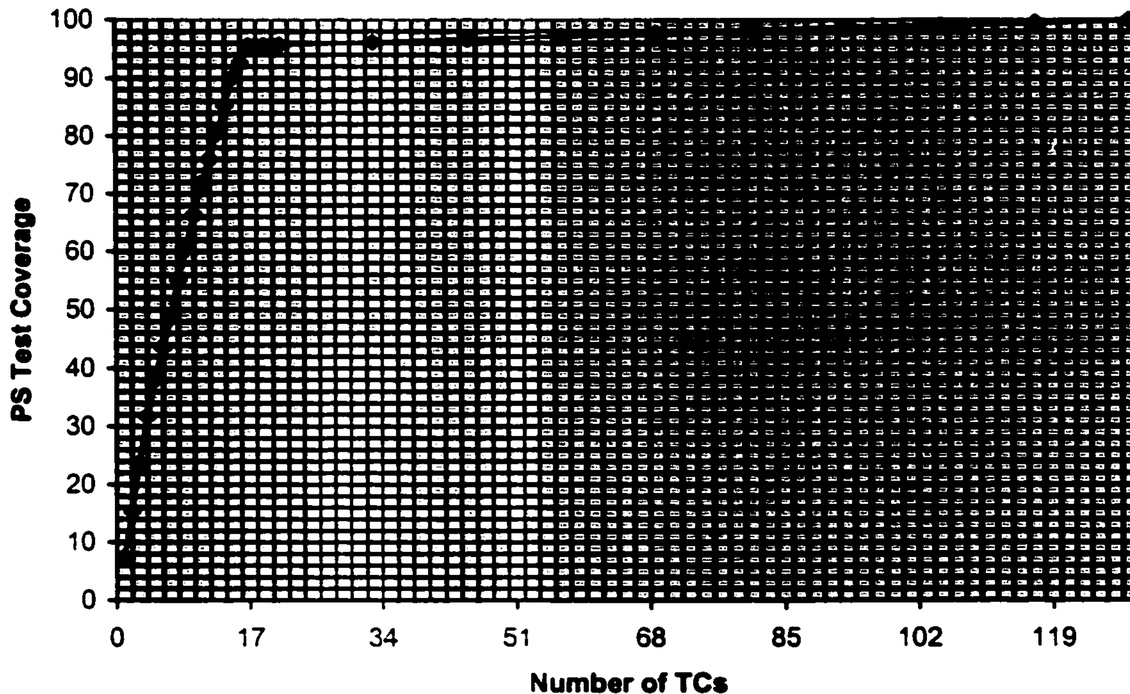


Figure 4.11: Relationship between the Number of TCs and PS Test Coverage

4.2.6.4 Combined CLB and Interconnect Testing

Minor changes are needed when combining the tests for both the CLBs and the interconnect. It can be seen that during the interconnect testing, parity-checkers and D flip-flops are formed in the section with the interconnect under test. Since parity-checkers are essentially XOR/XNOR functions that can be used to test the LUT mode function generators, and the D flip-flops are fully functionally tested during the interconnect testing. The resources that remain untested within a CLB are the CLM, the RAM mode of function generators and the MUXes. They can be easily distributed over the 17 interconnect TCs. This is achieved by keeping the connections outside a CLB for the 17 TCs unchanged, and changing the internal configurations and MUX connections of a CLB to test all the CLB resources. Such a distribution can easily be derived manually and is shown in Appendix 4. It can be seen from Appendix 4 that 12 of the 17 TCs are sufficient to test all CLB resources. TCs 1 to 3 test the LUT mode of

all function generators, TCs 4 to 8 test all the RAM modes, TCs 2, 4, and 9 to 12 test the CLM, and the testing of all the MUXes and D flip-flops is distributed among TC 1 to 12. The test coverage is the same as the one discussed in chapter 3.5.1.

4.3 Discussion

4.3.1 Pros

Compared with the BIST approaches for FPGA interconnect in [29, 30], the proposed BIST approach has superior multiple fault detectability. For example, if there are 12 wires under test, in the proposed approach two wire groups are formed, a WG_c with 9 wires and a WG_s with 3 wires. Only two combinations of multiple faults can not be detected under the proposed fault models (all the 9 wires in the WG_c are stuck-at-0, and all the 9 wires in the WG_c are bridged together). Note that any bridging faults between WG_c and WG_s can be detected. In the approach proposed in [29, 30], however, two wire groups WG1 and WG2 are formed with each WG containing 6 wires, the ith wire in WG1 is compared with the ith wire in WG2, where $1 \leq i \leq 6$. In this case, there are 189 combinations of multiple faults that can not be detected. They are: an even number of stuck-at-0 faults on the same position of wires in WG1 and WG2 ($C_6^1 + C_6^2 + C_6^3 + C_6^4 + C_6^5 + C_6^6 = 63$ combinations), an even number of stuck-at-1 faults on the same position of wires in WG1 and WG2 ($C_6^1 + C_6^2 + C_6^3 + C_6^4 + C_6^5 + C_6^6 = 63$ combinations), and bridging faults that bridge the ith wire in WG1 and ith wire in WG2 ($C_6^1 + C_6^2 + C_6^3 + C_6^4 + C_6^5 + C_6^6 = 63$ combinations).

Also, the proposed approach guarantees routability by exploiting dedicated device features such as MUX busses and edge decoders, while in [29, 30], routing congestion may occur when the BIST circuitry becomes complex.

4.3.2 Cons

The cost for the superior multiple fault detectability is that the applied test pattern is longer than that in [29, 30]. However, in FPGA testing, the major factor that affects the testing time is the time to configure an FPGA instead of the time to apply the test

patterns, since the former is usually two or three orders of magnitude greater than the latter. Therefore, the major concern to minimize the testing time is to minimize the number of test configurations, as will be justified in section 5.3 of chapter 5.

Chapter 5

Implementation

This chapter discusses the implementation of the BIST circuitry for testing both the CLBs and the interconnect, using a XC4020E FPGA as the device model. The CAD environment, the design flow, the implementation of the BIST circuitry, and the performance analysis are presented.

The CAD design flow has been developed by Susan Xu *et al.* in [46], it is used in this project with some modifications. In [46], a test configuration of the proposed BIST scheme for global interconnects is also implemented. However, the BIST circuitry and the BIST architecture need to be redesigned and implemented when both CLBs and interconnects are under test because the BIST circuitry and the BIST architecture are much more complex and the formation of bus structures is totally different.

5.1 The CAD Environment

The commercial CAD tools used to implement the proposed test configurations include Xilinx Foundation and third-party simulators such as ModelSim (Mentor Graphics). Together with the patch tool developed in-house, the CAD environment allows for the complete design and implementation of the proposed test configurations. Figure 5.1 shows the CAD environment.

FPGA EXPRESS: Accepts VHDL/VeriLog or schematic design, synthesizes and simulates the design, and translates it into a netlist file, using .XNF or .EDIF format.

NGDBUILD: Reads a netlist file in .XNF or .EDIF, and creates an Native Generic Database (NGD) file describing the logic design. The NGD file resulting from an NGDBUILD run contains a logical description of the design that is reduced to Xilinx NGD primitives and a description in terms of the original hierarchy.

MAP: Takes an NGD file as input and maps a logical design to a Xilinx FPGA. It first performs a logical Design Rule Check (DRC) on the design in the NGD file, then maps the logic to the components such as CLBs, I/O blocks, and other components in the target Xilinx FPGA. The output design is a Native Circuit Description (NCD) file, which is a physical representation of the design mapped to the components in the FPGA.

PAR: Accepts an NCD file as input, places and routes the design, and outputs an NCD file.

A user constraint file, .UCF, can be added before running the three tools, NGDBUILD, MAP and PAR. It specifies the desired locations of the CLBs and I/O blocks used to implement the design. It is used to ensure that the BIST circuitry is located in the half of the FPGA that is not under test.

FPGA EDITOR: A graphical application for displaying and configuring FPGAs. It allows for visual inspection and manual intervention of the layout of a design that has been synthesized and mapped on to the logic components of an FPGA. In FPGA EDITOR, one can locate individual logic components and signal nets. A set of GUI features provided by the vendor allows manually editing a layout, such as re-positioning certain logic modules and re-routing a critical path of a design. This step can be taken after placement and routing in the conventional design flow. The file editable in the FPGA editor is in .NCD format and FPGA EDITOR updates the .NCD file generated by PAR before completing the remaining design steps.

BITGEN: Convert a .NCD file into a bit-stream file, a binary file with a .BIT extension, for Xilinx device configuration.

TIMING ANALYZER: Provides static timing analysis of a post-mapping design, including the maximum operation frequency and the maximum net delay information, and outputs a timing report file *_TRCE.XML, or a plain text timing report file .TWR.

NGDANNO: Generates a generic timing simulation model and outputs an annotated logical design that has a .NGA (Native Generic Annotated) extension.

NGD2VHDL: Translates the design into a VHDL file containing a netlist description of the design in term of Xilinx simulation primitives, which can be used to perform a post-mapping simulation by a VHDL simulator.

Patch Program: The in-house C program that was developed to perform the automatic routing to form the desired net architecture. It generates a .SCR file that updates the .NCD file by using the playback feature of the FPGA editor.

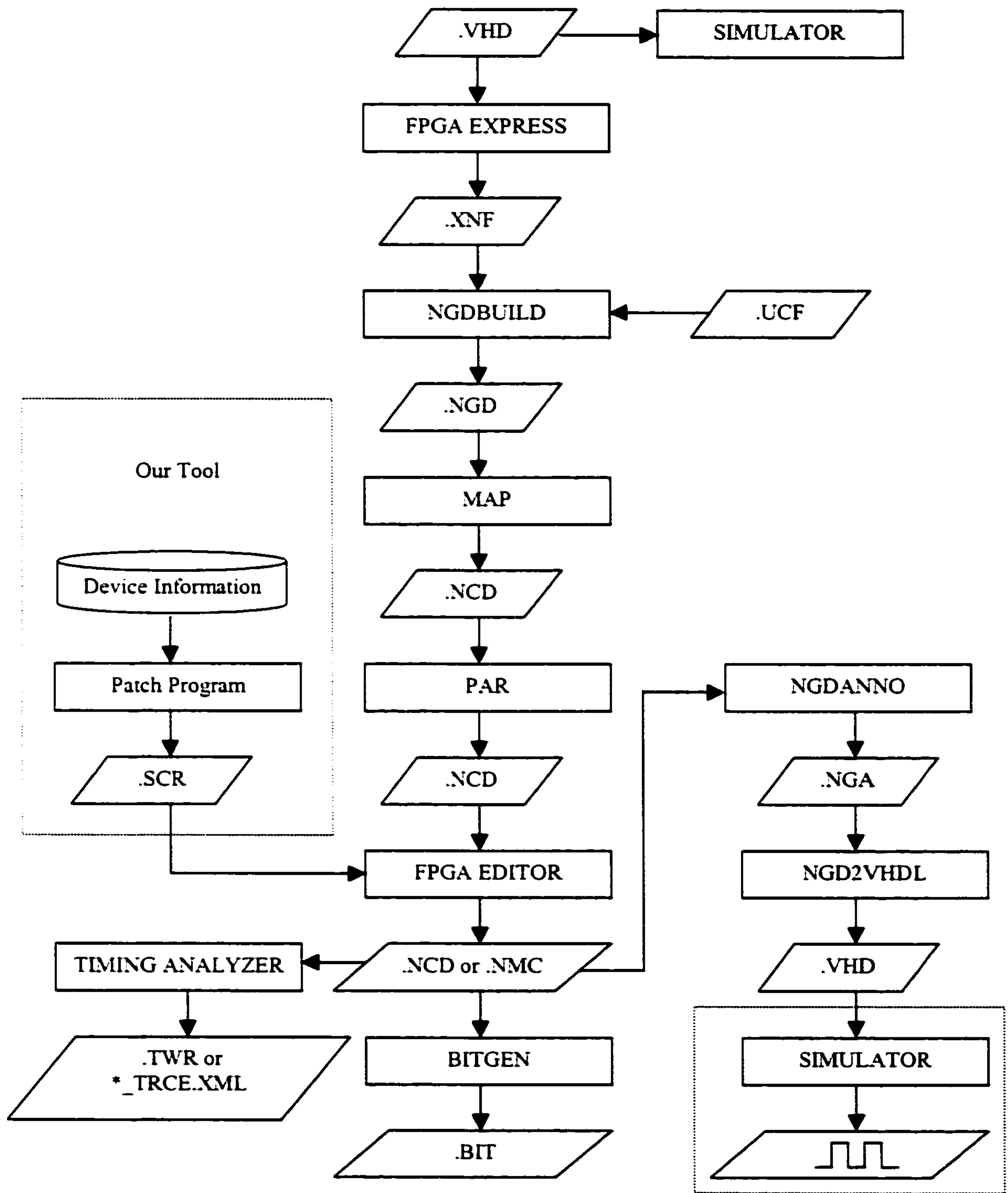


Figure 5.1: The CAD Environment

5.2 The Design Flow

The design flow is somewhat different from the conventional one that is used for the design and implementation using an FPGA. The BIST circuitry, on the one hand, and the interconnect structures to be tested, on the other hand, are generated separately. The former is built and then combined with the latter through the FPGA editor. The implementation of a test configuration is carried out in the following three stages.

(1) Design the BIST circuitry using either HDL or schematic design entry. Synthesize the BIST circuitry using FPGA-Express and obtain the .XNF file. Compose the .UCF to specify the desired locations of CLBs (within one half of the FPGA) for implementing the BIST. Run NGDBUILD, MAP and PAR to generate the .NCD file.

(2) Run the patch program to generate the .SCR description of the desired bus structure and the configuration of CLBs in the other half of the FPGA where there exist CLBs and interconnects under test. Load the .SCR file into the .NCD file of the BIST circuitry by using the playback feature of the FPGA EDITOR.

One modification is made in this step compared with the design flow in our previous work in [46]. The .SCR file is loaded directly into the .NCD design instead of saving both .SCR and .NCD files as two macros and then combining the two macros together. The modification is based on the fact that the Xilinx CAD tools prohibit any operation with a macro, even the adjustment of wire connections that do not change the functionality. The modification results in more flexibility during the design and better performance, especially when a macro contains the critical path(s).

(3) Use the patch program to integrate the BIST circuitry in one half of the FPGA and the CLBs and interconnects under test in the other half automatically to form the test configuration. The results update the file .NCD. Use BITGEN to generate .BIT. Run TIMING ANALYZER to perform the static timing analysis. Run NGDANNO and NGD2VHDL to get the .VHD design. Use third party simulators, such as ModelSim, to perform the post-mapping simulation.

5.3 The Implementation

This section presents the design and implementation of the BIST circuitry for the proposed test configurations, using the Xilinx XC4020E FPGA as the target device.

The BIST circuitry consists of a data path and a control path. The data path includes two general TPGs, one ORA and one shift register. The control path is the BIST controller. The VHDL design for the BIST circuitry is shown in Appendix 4.

The general TPG generates four types of test patterns, exhaustive (counts up from all 0s to all 1s), walk-1, all-1, and all-0 with variable bit width. Figure 5.2 (a) shows the logic symbol of a general TPG with four inputs and three outputs. Input signal "Pattern select" selects one of the four types of test patterns (exhaustive, walk-1, all-1, and all-0), and input signal "Next test vector" asks the general TGA to generate the next test vector under a certain test pattern. For example, if "Pattern select" selects exhaustive test, and the current test vector is "010", then the TPG will generate the test vector "011" when the input signal "Next test vector" is active. The output signals "outputs" output the generated test vector, and the signal "Parity" is the odd-parity of the generated test vector. The signal "Done" is active when all the test vectors have been generated for the selected test.

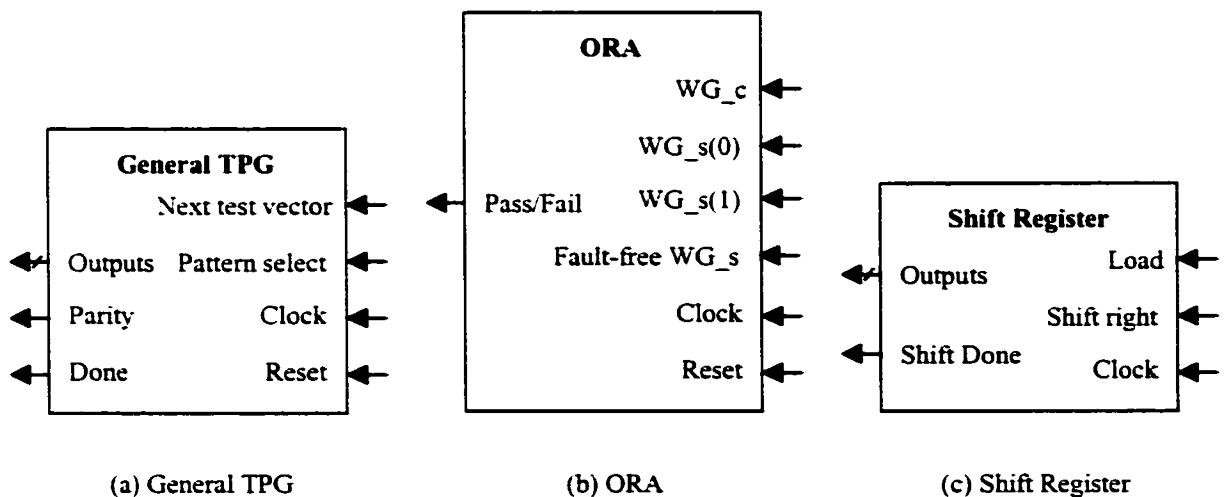


Figure 5.2: Logic Symbols for Components in the Data Path

The ORA collects the test responses from the resources under test, carries out the output response analysis, and reports to the BIST controller the pass/fail result of the test. It has six inputs and one output, as shown in Figure 5.2 (b). Signals WG_c, WG_s(0), WG_s(1) and Fault-free WG_s correspond to signals O_{pc}, WA_0, WA_1, Q_{TPG} in Figure 4.9 respectively. See Figure 4.9 in chapter 4 for details concerning the signals.

The shift register provides the patterns for the column select of the CLB outputs, as shown in Figure 5.2 (c).

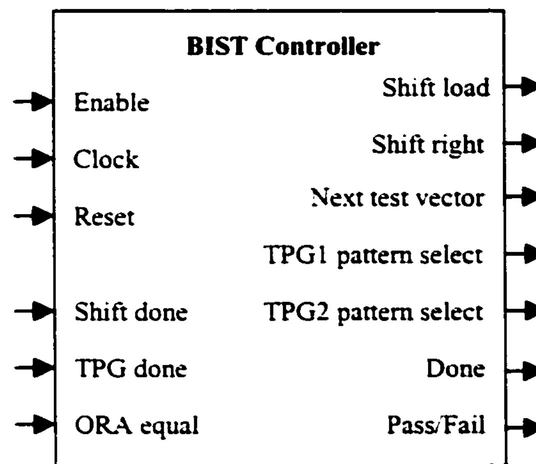


Figure 5.3: Logic Symbol of the BIST Controller

The BIST controller, the core component in the BIST circuitry, enables the test to start, controls the process of the test, and generates the pass/fail signal. The logic symbol of a BIST controller is shown in Figure 5.3. The signals "Shift load" and "Shift right" control the shift register to load the initial pattern and to shift one bit right, respectively. Signals "TPG1 pattern select", "TPG2 pattern select", and "Next test vector" are used to control the selection of different types of test patterns, and to go on to the next test vector for the selected test pattern for the two general TPGs. Signals "Shift done", "TPG done", and "ORA equal" monitor the process of the shift register, TPGs, and the ORA, respectively. The remaining five signals ("Enable",

"Clock", "Reset", "Done", "Pass/Fail") are used to communicate with the IEEE 1149.1 standard test port.

The BIST controller is designed as a finite state machine. Its state diagram is shown in Figure 5.4. It can be seen from Figure 5.4 that the operation of the BIST controller can be divided into four parts, as shown in the four dashed boxes in Figure 5.4. Part 1 and part 2 correspond to test 1 and test 2 in chapter 4.2.5, respectively. Part 3 and part 4 correspond to test 3 in chapter 4.2.5. That is, test 3 is divided into two parts: (1) Part 3 is the all-1 test for WG_c with the corresponding parities and exhaustive test for WG_s with the fault-free Q outputs. (2) Part 4 is the walk-1 test with the corresponding parities for WG_c and exhaustive test with the fault-free Q outputs for WG_s. See Table 4.3 in chapter 4 for details.

It can be seen from Figure 5.4 that the state diagrams in the dashed boxes are similar to each other. This is because the flow of the state diagram is the same for different types of test patterns. During each applied test pattern, the state diagram begins with the first test vector for the test, and collects the outputs in the first column. Recall that once a test vector is applied, all the CLBs in the resources under test section will generate their outputs. See the high level BIST architecture in Figure 4.9 for details. If the test vector for the first column is passed, the outputs from the next column will be collected and examined. Finally, the state goes to the "Next test vector" if there is no fail for all columns, and the output collection is repeated column by column again, until all test vectors in the selected test pattern have been applied.

Once the test fails, the state goes to "Test failed" and stops the application of the remaining test vectors. Otherwise, after all the four tests have been applied and there is no fail, the BIST controller enters state "All tests passed".

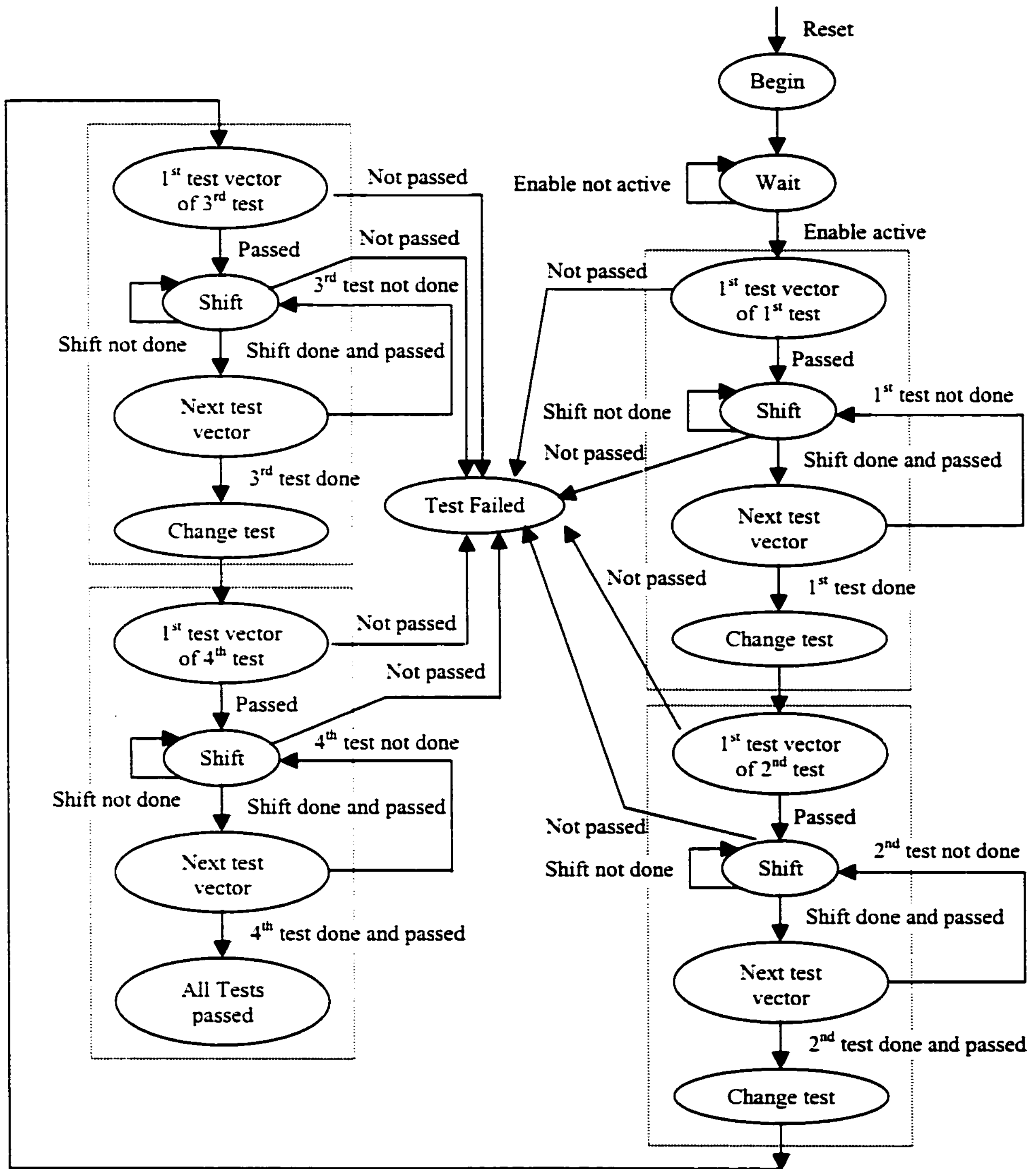
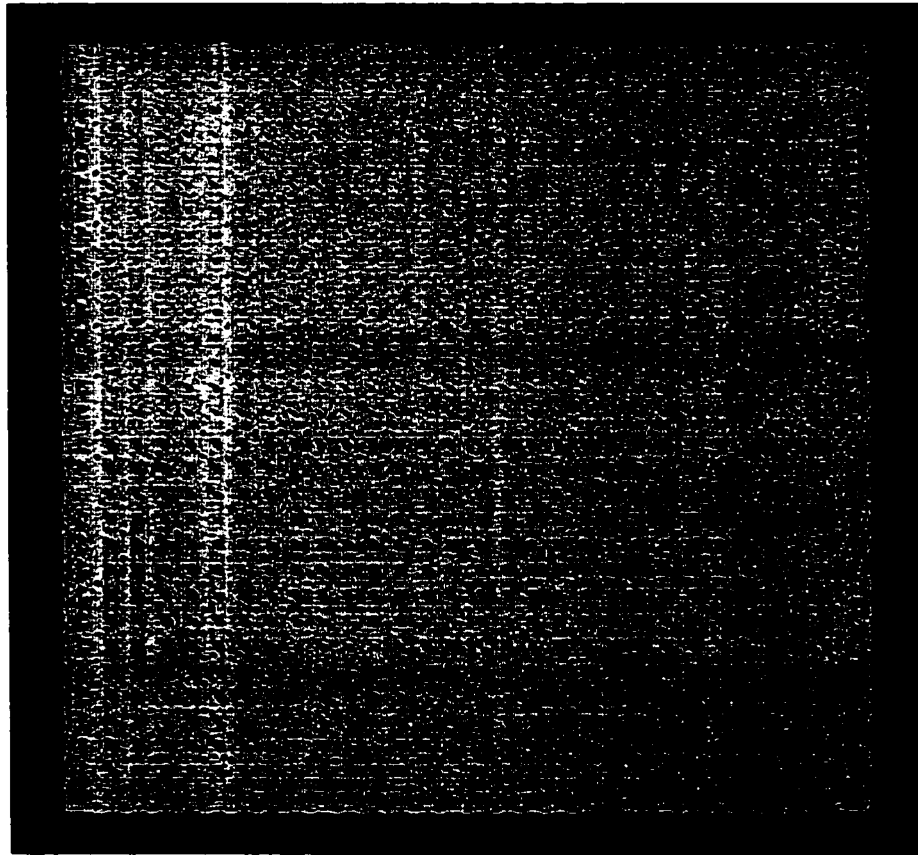
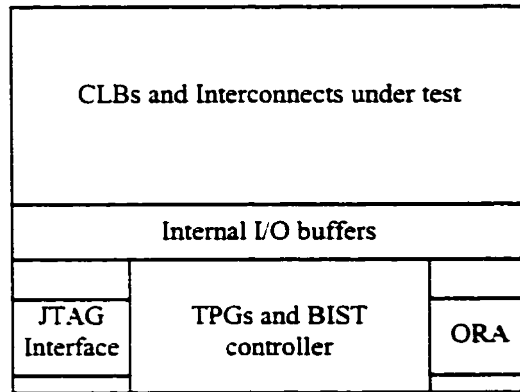


Figure 5.4: The State Diagram of the BIST Controller

Figure 5.5 (a) shows the physical layout of one of the proposed test configurations, and Figure 5.5 (b) illustrates the distribution of the key components in the configuration. It can be seen from Figure 5.5 that the whole FPGA chip can be divided into five areas. The upper half contains resources under test. The lower half contains the TPGs, the ORA, the BIST controller, the interface for the IEEE standard test port (JTAG port), and the internal I/O buffers. The insertion of the internal I/O buffers separates the BIST circuitry and the resources under test so that the design of the BIST circuitry and the build of the bus structures for the resources under test can be carried out independently. All the communications between the two become communications with these internal I/O buffers.



(a) Physical Layout



(b) Illustration of the Component Distribution

Figure 5.5: The Layout of a Test Configuration

It can be seen from the Xilinx data book [35] that the download time of a configuration for the XC4020E is between 263ms~659ms. The post-map operation frequency of the BIST system is 2.375MHz, which translates to a total test application time of 3.3ms. Therefore, the test application time is less than 1.3% of the time required for configuring the FPGA. This justifies the strategy of minimizing the number of test configurations as the most effective way of reducing the test time (or testing costs) instead of the compacting of the test patterns.

5.4 Scalability and Adaptability

The proposed BIST scheme is applicable and scalable for Xilinx XC4000 FPGAs with over 200 CLBs. Our BIST scheme is adaptable to all XC4000 FPGAs. For CLB testing, there is no adaptability problem since all the CLBs in XC4000 FPGAs are the same. For interconnect testing, XC4000X FPGAs have additional quad lines. However, these quad lines share the same routing architecture as the single and double lines in XC4000E FPGAs, therefore, they can be configured into similar bus structures and be tested in the same way as for single and double lines.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis the author modified the high-level *built-in self-test* (BIST) architecture in [2, 3] for testing *configurable logic blocks* (CLBs) and interconnects, and determined the minimum test configurations for the CLBs and interconnect in SRAM-based Xilinx XC4000E *field programmable gate arrays* (FPGAs). This study includes both theory and implementation. The main contributions of this thesis are:

(1) This thesis modified the BIST architecture to test the CLBs including the *carry logic modules* (CLMs) in [2] by introducing the comb-shaped bus structure to replace the originally proposed snake-shaped bus structures. This greatly increases the maximum possible system operational frequency. The author introduced a systematic method for testing the CLMs. It includes the construction of fault models, the development of the test sequence for the modeled faults, the derivation of the testability equations, and the implementation of an exhaustive search program for the valid CLM operation modes [35] to identify all the solutions of the proposed testability equations. A set of selection criteria was established to select the "best" CLM *test configurations* (TCs) that permits the parallel testing of the CLM and the remaining CLB resources.

An intuitive method was used to derive minimum TCs for the remaining CLBs. A set of minimum TCs was derived manually that merges the testing of the remaining CLB resources with the "best" CLM TCs.

(2) This thesis completes the work on FPGA global interconnect testing using the error-control coding technique from [3]. The author constructed the fault models, derived a test sequence for the modeled faults, and conducted a detailed analysis of the fault detectability of global interconnects under the proposed fault models. This analysis led to the formal proofs of Theorems 1-4, a joint work with the other authors in [31].

This thesis further extends the work in [31] to include the testing of local interconnect. A novel technique was proposed to test the sequential wire groups through the functional test of D flip-flops (DFFs) in the CLBs in order to maximize the usage of CLB inputs, which in turn minimizes the number of TCs. A detailed analysis of the faulty behaviors of the sequential wire groups tested by the DFFs was conducted and the proofs of the fault detectability were given in Theorems 5-8. This novel technique combines the error-control coding technique in [31] for testing the combinational wire groups to provide superior multiple fault detectability on the modeled interconnect faults, including stuck-at, stuck-open, bridging faults on wire segments and stuck-on/off faults of programmable switches, and the combination of the faults outlined above.

The trade-off between the test coverage for interconnect and the test cost was discussed. A minimum of 17 TCs, which verified the integrity of 95.51% of the local and global interconnects, was derived under the proposed constraints and rules. The remaining 4.49% interconnects can be tested in $4*n$ TCs due to architectural limitations of the device family, where n is the number of CLB columns in an FPGA. We showed that the 8 CLB TCs can be easily merged into the 17 interconnect test TCs to support the simultaneous test of both CLBs and interconnects.

(3) One of the 17 TCs for both CLB and interconnect testing was designed and implemented on a XC4020E FPGA. The BIST circuitry, including the *test pattern generators* (TPGs), the *output response analyzers* (ORAs), and the BIST controller, was designed and synthesized using the standard Xilinx FPGA tools. The CAD environment and design flow in [43] is modified to create the desired bus structure. The post-mapping simulation results demonstrate the feasibility of the proposed BIST scheme.

6.2 Future Work

Future work will address both the extensions to this project and the suggestions for future modifications of Xilinx tools.

6.2.1 Extensions

(1) Introduce design for testability (DFT) techniques into the BIST circuitry.

In this thesis the BIST circuitry was implemented in one half of the FPGA to test the resources in the other half of the FPGA. To guarantee the fault-free operation of the BIST circuitry itself, certain fault-tolerant techniques, such as triple mode redundancy, could be applied to the BIST design. However, it is only applicable for medium and large FPGAs, which contain more than enough CLBs in one half for the implementation of BIST circuits. For small FPGAs, DFT techniques combined with boundary scan can be added to the BIST design so that the BIST could be verified online verified to guarantee fault-free operation.

(2) Build or obtain a layout information database to ease test automation.

Currently, the test configurations generated for one FPGA device family can not be implemented directly on another FPGA device family because of the different layout information contained in different device families. For the same device family, the BIST circuitry designed for one TC can be used for another TC, but it needs lots of work to form the test structures for different TCs due to the bad component reference rules in current Xilinx CAD tools, as further discussed in section 6.2.2. In the future if there are no modifications to the component reference rules, nor to the CAD tool from Xilinx that we used to obtain the layout information for all the components in the FPGA, the following database could be built for each device family as an alternate solution for the automatic generation of different TCs for a device family, or for different device families. Such a database would include: (a) the X and Y coordinates of the origin for each type of component, (b) the row and column number of each type of component, (c) the offset of each type of component from its origin, and (d) boundary information and tile information. Then a scripting program could use the

device information from this database to generate the test structures for FPGAs with different layouts.

6.2.2 Suggestions for Xilinx FPGA Design and CAD Tools

(1) Component Reference Rules

The test automation problem discussed in 6.2.1 (2) is inherently caused by the component reference rules in the Xilinx CAD tools. In Xilinx CAD tools, some FPGA components are referred by their relative position. For instance, the output pin X of CLB in row 2 column 3 of an FPGA is referred as "CLB_R2C3.X". This reference rule makes it possible for the program to refer to a component without knowing its physical layout information, i.e., its X and Y coordinates. The test configurations generated for the component, therefore, can be re-used from family to family without any modification. However, some other components in the FPGA (such as the single, double length lines, long lines, global lines and boundary connection lines) are referred to using absolute values of the X and Y coordinates. For instance, the first vertical single line associated with CLB in row 1 and column 1 is referred to as "/X:496 /Y:10189". Such X and Y coordinates are device dependent. Furthermore, for the lines in the boundaries and between the tiles, the offset of a certain type of line and its origin is irregular, so that it is impossible to know its X and Y coordinates unless the line is manually selected in the CAD tool. This makes it impossible for a user-developed program to generate the test structure automatically for different device families, or different TCs (different test structures) for the same device family. Our suggestion is that the component reference rule be modified such that all the components are referred to by their relative positions without knowing the X and Y coordinates. For instance, the above vertical single line could be referred to as "CLB_R1C1.VS1". The design automation problem could be solved if the proposed component reference rule were to be applied.

(2) Flexibility in Macros

In Xilinx tools, one can implement a design and later insert the design as a macro into another design. However, once a macro is formed, only communications through its inputs and outputs are permitted and nothing can be changed within the macro, not

even routing changes that do not affect the functionality of the design. These limitations restrict both the implementation flexibility and the circuit performance. For instance, the tool will not be able to optimize the routing if the macro contains a critical path that can be replaced by a non-critical path. Our suggestion is to allow more freedom in the macro operation. At least routing changes should be permitted within a macro that do not affect the functionality.

Bibliography

[1] Virtual Computer Corporation. Overview of the FPGA (Online document at http://www.vcc.com/fpga.html).

[2] X. Sun. Testing configurable logic blocks of Xilinx FPGAs. Project Proposal. Department of Electrical &Computer Engineering, University of Alberta/Nortel Networks. April 30. 1999.

[3] X. Sun. Testing the interconnect of Xilinx FPGAs. Project Proposal. Department of Electrical &Computer Engineering, University of Alberta/Nortel Networks, May 21. 1999.

[4] M. Abramovici. M. Breuer and A. Friedman. Digital systems testing and testable design. Computer Science Press. 1990.

[5] L. Zhao. D. Walker and F. Lombardi. IDDQ testing of bridging faults in logic resources of reconfigurable field programmable gate arrays. IEEE Transactions on Computers. 1136-1152. 1998.

[6] L. Zhao. D. Walker. F. Lombardi. IDDQ testing of input/output resources of SRAM-based FPGAs. IEEE Asian Test Symposium, 1999.

[7] R. Gulati and C. Hawkins. IDDQ testing of VLSI circuits. Kluwer Academic Publishers. 1997.

[8] C. Su. S. Jeng and Y. Chen. Boundary-scan BIST methodology for reconfigurable systems. In Proceeding of the IEEE International Test Conference. 774-783. 1998.

[9] G. Gibson. L. Gray and C. Stroud. Boundary scan access of BIST for filed programmable gate arrays. In IEEE International Application Specific Integrated Circuits Conference. 57-61, 1997.

[10] M. Renovell, P. Faure, J. Portal, J. Figueras, and Y. Zorian. IS-FPGA: A new symmetric FPGA architecture with implicit SCAN. In IEEE International Test Conference, 924-931, 2001.

[11] X. Sun, X.T. Chen, W.K. Huang and F. Lombardi. A row-based FPGA for single and multiple stuck-at fault detection. In IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems, 225-233, 1995.

[12] W.K. Huang, F.J. Meyer and F. Lombardi. Multiple fault detection in logic resources of FPGAs. In International Symposium on Defect and Fault Tolerance in VLSI Systems, 183-191, 1997.

[13] W.K. Huang, F.J. Meyer and F. Lombardi. An approach for detecting multiple faulty FPGA logic blocks. In IEEE Transactions on Computers, Vol. 49, No.1, 48-54, 2000.

[14] M. Renovell, J.M Portal, J. Figueras, and Y. Zorian. An approach to minimize the test configuration for the logic cells of the Xilinx XC4000 FPGAs family. In Journal of Electronic Testing: Theory and Applications, 289-299, 2000.

[15] M. Renovell, J. Figueras and Y. Zorian. Test of RAM-based FPGA: methodology and application to the interconnect. In IEEE VLSI Test Symposium, 230-237, 1997.

[17] M. Renovell, J. Figueras and Y. Zorian. Testing the inteconnect of RAM-based FPGAs. In IEEE Design and Test of Computers, 45-50, 1998.

[18] H. Michinishi, T. Yokohira and T. Okamoto. A test methodology for interconnect structures of LUT-based FPGAs. In proceeding of Asian Test Symposium, 68-74, 1996.

[19] W. Cheng, J. Lewandowshi and E. Wu. Diagnosis for wiring interconnects. In IEEE International Test Conference, 565-571, 1990.

[20] W.K. Huang, X. Cheng and F. Lombardi. On the diagnosis of programmable interconnect systems: theory and application. In IEEE VLIS Test Symposium, 204-209, 1996.

[21] Y. Yu, J. Xu, W.K. Huang, and F. Lombardi. A diagnosis method for interconnects in SRAM-based FPGAs. In IEEE Asian Test Symposium, 278-282, 1998.

[22] Y. Yu, J. Xu, W.K. Huang, and F. Lombardi. Minimizing the number of programming steps for diagnosis of interconnect faults in FPGAs. In IEEE Asian Test Symposium, 1999.

[23] C. Stroud, S. Konala, Pin Chen, and M. Abramovici. Built-in self-test of logic blocks in FPGAs. In Proceeding of IEEE VLSI test symposium, 387-392, 1996.

[24] C. Stroud, E. Lee, S. Konala, and M. Abramovici. Selecting built-in self-test configurations for field programmable gate arrays. In Proceeding of IEEE Automatic test Conference, 29-35, 1996.

[25] C. Stroud, E. Lee, and M. Abramovici. BIST-based diagnostics of FPGA logic blocks. In Proceeding of IEEE International Test Conference, 539-547, 1997.

[26] C. Stroud, E. Lee, S. Konala, and M. Abramovici. Using ILA testing for BIST in FPGAs. In Proceeding of IEEE International Test Conference, 68-75, 1996.

[27] X. Sun, J. Xu, and P. Troubst. Testing the carry logic modules of SRAM-based FPGAs. In IEEE Workshop on Memory Technology, Design and Testing, 91-98, 2001.

[28] X. Sun, J. Xu and P. Trouborst. Testing Xilinx XC4000 configurable logic blocks with carry logic modules. To appear in the Proceeding of IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2001.

[29] C. Stroud. S. Wijesuriya, C. Hamilton, and M. Abramovici. Built-in self-test of FPGA interconnect. In Proceeding of IEEE International Test Conference, 404-410, 1998.

[30] M. Abramovici. C. Stroud. C. Hamilton, S. Wijesuriya, and V. Verma. Using roving stars for on-line testing and diagnosis of FPGAs in fault-tolerant applications. In Proceeding of IEEE International Test Conference, 973-982. 1999.

[31] X. Sun. J. Xu, and P. Troubst. Novel technique for built-in self-test of SRAM-based FPGAs. In Proceeding of IEEE International Test Conference, 795-803, 2000.

[32] C. Metra. A. Pagano, and B. Ricco. On-line testing of transient and crosstalk faults affecting interconnects of FPGA-implemented systems. In IEEE International Test Conference, 939-947, 2001.

[33] A. Doumar. T. Ohmameuda. and H. Ito. Design of an automatic testing for FPGAs. In IEEE Asian Test Symposium, 1999.

[34] A. Doumar. H. Ito. Testing the logic cells and interconnect resources for FPGAs. In IEEE Asian Test Symposium. 1999.

[35] The programmable logic data book. Xilinx Inc., 2000.

[36] Frank F. Tsui. LSI/VLSI testability design. McGraw-Hill Book Company, 1986.

[37] M. Renovell. J.M. Portal, J. Figueras. and Y. Zorian. SRAM-Based FPGA's: testing the LUT/RAM module. In Proceeding of IEEE International Test Conference, 1102-1111, 1998.

[38] X. Sun, J. Xu, and P. Trouborst. Testing xilinx XC4000 configurable logic blocks with the carry logic modules. Technical Report, Department of Electrical and Computer Engineering, University of Alberta, January 2001.

[39] Xilinx Online Software Manual. Chapter 12: Attributes, constraints, and carry logic. *Library Guides, Alliance Series 2.1i Software Documentation*, Xilinx Software Manuals.

[40] C. Metra, *et al.* Novel techniques for testing FPGAs. In Design, Automation and Test in Europe Conference, 89-94, 1998.

[41] A.J. van de Goor. Testing semiconductor memories: Theory and Practice, John Wiley & Sons Ltd., 1991.

[42] T. Liu, F. Lombardi and J. Salinas. Diagnosis of interconnects and FPICs using a structured walking-1 approach. In Proceeding of IEEE VLSI Test Symposium, 256-261, 1995.

[43] X. Sun, S. Xu, J. Xu, and P. Trouborst. Design and implementation of a parity-based BIST scheme for FPGA global interconnects. IEEE Canadian Conference on Electrical and Computer Engineering, 2000.

[44] W.K. Huang, F.J. Meyer, and F. Lombardi. Array-based testing of FPGAs: architecture and complexity. In Proceeding of IEEE Conference on Innovative Systems in Silicon, 249-258, 1997.

[45] M. Renovell, J.M. Portal, J. Figueras, and Y. Zorian. Testing the local interconnect resources of SRAM-based FPGAs. In Journal of Electronic Testing: Theory and Application, 513-520, 2000.

Appendix 1

Testability Equations for the CLM

Part 1: Equations for substitution

```
C0_ANDF  = (!F2) + (!C7);

C1_ANDF  = F2 * C7;

C0_CM1 = (!C0*(!C1)) + (!F3*C0);

C1_CM1 = (!C0*C1) + (F3*C0);

C0_XORF1 = (C0_ANDF*C1_CM1) + (C1_ANDF*C0_CM1);

C1_XORF1 = (C0_ANDF*C0_CM1) + (C1_ANDF*C1_CM1);

C0_XORF2 = ( C0_XORF1*(!F1) ) + ( C1_XORF1*F1 );

C1_XORF2 = ( C0_XORF1*F1 ) + ( C1_XORF1*(!F1) );

C0_FM1 = (C3*C0_XORF2)+(!C3*(!C2));

C1_FM1 = (!C3*C2) + (C3*C1_XORF2);

C0_CM2 = (!C4*(!C5)*(!C7)) + (F3*(!C4)*C5) + (!F1*C4*C5);

C1_CM2 = (!C4*(!C5)*C7)   + (!F3*(!C4)*C5) + (F1*C4*C5);

C0_FM2 = (C0_FM1*C0_CM2) + (C1_FM1*(!Cin));

C1_FM2 = (C0_FM1*C1_CM2) + (C1_FM1*Cin);

C0_ANDG  = (!G1) + (!C7);

C1_ANDG  = G1 * C7;

C0_XORG1 = (C0_ANDG*C1_CM1) + (C1_ANDG*C0_CM1);

C1_XORG1 = (C0_ANDG*C0_CM1) + (C1_ANDG*C1_CM1);

C0_XORG2 = ( C0_XORG1*(!G4) ) + ( C1_XORG1*G4 );

C1_XORG2 = ( C0_XORG1*G4 ) + ( C1_XORG1*(!G4) );

C0_GM1 = C6*C0_XORG2;

C1_GM1 = (!C6) + (C6*C1_XORG2);

O_FM2 = C1_GM1;
```

```
O_FM1 = ( (!Cin*C1_CM2) + (Cin*C0_CM2) ) * O_FM2;

O_CM2 = C0_FM1 * O_FM2;

O_GM1 = (!G4 * C1_FM2) + (G4 * C0_FM2);

O_XORG2 = C6 * O_GM1;

O_XORG1 = O_XORG2;

O_CM1_2 = O_XORG1;

O_ANDG = O_XORG1;

O_XORF2 = C3 * O_FM1;

O_XORF1 = O_XORF2;

O_CM1_1 = O_XORF1;

O_ANDF = O_XORF1;
```

Part 2: Testability Equations for Universal Fault Model of a CLM

ANDG_00= (!G1)*(!C7)*O_ANDG;

ANDG_01= (!G1)*(C7)*O_ANDG;

ANDG_10= (G1)*(!C7)*O_ANDG;

ANDG_11= (G1)*(C7)*O_ANDG;

ANDF_00= (!F2)*(!C7)*O_ANDF;

ANDF_01= (!F2)*(C7)*O_ANDF;

ANDF_10= (F2)*(!C7)*O_ANDF;

ANDF_11= (F2)*(C7)*O_ANDF;

XORG1_00= C0_ANDG * C0_CM1 * O_XORG1;

XORG1_01= C0_ANDG * C1_CM1 * O_XORG1;

XORG1_10= C1_ANDG * C0_CM1 * O_XORG1;

XORG1_11= C1_ANDG * C1_CM1 * O_XORG1;

XORF1_00= C0_ANDF * C0_CM1 * O_XORF1;

XORF1_01= C0_ANDF * C1_CM1 * O_XORF1;

XORF1_10= C1_ANDF * C0_CM1 * O_XORF1;

XORF1_11= C1_ANDF * C1_CM1 * O_XORF1;

XORG2_00= C0_XORG1 * (!G4) * O_XORG2;

XORG2_01= C0_XORG1 * (G4) * O_XORG2;

XORG2_10= C1_XORG1 * (!G4) * O_XORG2;

XORG2_11= C1_XORG1 * (G4) * O_XORG2;

XORF2_00= C0_XORF1 * (!F1) * O_XORF2;

XORF2_01= C0_XORF1 * (F1) * O_XORF2;

XORF2_10= C1_XORF1 * (!F1) * O_XORF2;

```
XORF2_11= C1_XORF1 * (F1) * O_XORF2;
GM1_00= (!C6) * C0_XORG2 * O_GM1;
GM1_01= (!C6) * C1_XORG2 * O_GM1;
GM1_10= (C6) * C0_XORG2 * O_GM1;
GM1_11= (C6) * C1_XORG2 * O_GM1;
FM1_000= (!C3) * C0_XORF2 * (!C2) * O_FM1;
FM1_001= (!C3) * C0_XORF2 * (C2) * O_FM1;
FM1_010= (!C3) * C1_XORF2 * (!C2) * O_FM1;
FM1_011= (!C3) * C1_XORF2 * (C2) * O_FM1;
FM1_100= (C3) * C0_XORF2 * (!C2) * O_FM1;
FM1_110= (C3) * C1_XORF2 * (!C2) * O_FM1;
GM2_000= C0_GM1 * (!G4) * C0_FM2;
GM2_001= C0_GM1 * (!G4) * C1_FM2;
GM2_010= C0_GM1 * (G4) * C0_FM2;
GM2_011= C0_GM1 * (G4) * C1_FM2;
GM2_100= C1_GM1 * (!G4) * C0_FM2;
GM2_101= C1_GM1 * (!G4) * C1_FM2;
GM2_110= C1_GM1 * (G4) * C0_FM2;
GM2_111= C1_GM1 * (G4) * C1_FM2;
FM2_000= C0_FM1 * C0_CM2 * (!Cin) * O_FM2;
FM2_001= C0_FM1 * C0_CM2 * (Cin) * O_FM2;
FM2_010= C0_FM1 * C1_CM2 * (!Cin) * O_FM2;
FM2_011= C0_FM1 * C1_CM2 * (Cin) * O_FM2;
FM2_100= C1_FM1 * C0_CM2 * (!Cin) * O_FM2;
FM2_101= C1_FM1 * C0_CM2 * (Cin) * O_FM2;
```

```
FM2_110= C1_FM1 * C1_CM2 * (!Cin) * O_FM2;

FM2_111= C1_FM1 * C1_CM2 * (Cin) * O_FM2;

CM1_000= (!C0) * (!C1) * (!F3) * (O_CM1_1 +O_CM1_2);

CM1_001= (!C0) * (!C1) * (F3) * (O_CM1_1 +O_CM1_2);

CM1_010= (!C0) * (C1) * (!F3) * (O_CM1_1 +O_CM1_2);

CM1_011= (!C0) * (C1) * (F3) * (O_CM1_1 +O_CM1_2);

CM1_100= (C0) * (!C1) * (!F3) * (O_CM1_1 +O_CM1_2);

CM1_101= (C0) * (!C1) * (F3) * (O_CM1_1 +O_CM1_2);

CM2_00000= (!C4) * (!C5) * (!F1) * (!F3) * (!C7) * O_CM2;

CM2_00001= (!C4) * (!C5) * (!F1) * (!F3) * (C7) * O_CM2;

CM2_00010= (!C4) * (!C5) * (!F1) * (F3) * (!C7) * O_CM2;

CM2_00011= (!C4) * (!C5) * (!F1) * (F3) * (C7) * O_CM2;

CM2_00100= (!C4) * (!C5) * (F1) * (!F3) * (!C7) * O_CM2;

CM2_00101= (!C4) * (!C5) * (F1) * (!F3) * (C7) * O_CM2;

CM2_00110= (!C4) * (!C5) * (F1) * (F3) * (!C7) * O_CM2;

CM2_00111= (!C4) * (!C5) * (F1) * (F3) * (C7) * O_CM2;

CM2_01000= (!C4) * (C5) * (!F1) * (!F3) * (!C7) * O_CM2;

CM2_01001= (!C4) * (C5) * (!F1) * (!F3) * (C7) * O_CM2;

CM2_01010= (!C4) * (C5) * (!F1) * (F3) * (!C7) * O_CM2;

CM2_01011= (!C4) * (C5) * (!F1) * (F3) * (C7) * O_CM2;

CM2_01100= (!C4) * (C5) * (F1) * (!F3) * (!C7) * O_CM2;

CM2_01101= (!C4) * (C5) * (F1) * (!F3) * (C7) * O_CM2;

CM2_01110= (!C4) * (C5) * (F1) * (F3) * (!C7) * O_CM2;

CM2_01111= (!C4) * (C5) * (F1) * (F3) * (C7) * O_CM2;

CM2_11000= (C4) * (C5) * (!F1) * (!F3) * (!C7) * O_CM2;
```

```
CM2_11001= (C4) * (C5) * (!F1) * (!F3) * (C7) * O_CM2;
CM2_11010= (C4) * (C5) * (!F1) * (F3) * (!C7) * O_CM2;
CM2_11011= (C4) * (C5) * (!F1) * (F3) * (C7) * O_CM2;
CM2_11100= (C4) * (C5) * (F1) * (!F3) * (!C7) * O_CM2;
CM2_11101= (C4) * (C5) * (F1) * (!F3) * (C7) * O_CM2;
CM2_11110= (C4) * (C5) * (F1) * (F3) * (!C7) * O_CM2;
CM2_11111= (C4) * (C5) * (F1) * (F3) * (C7) * O_CM2;
```

Part 3: Testability Equations for Multiple Fault Model of a CLM

ANDG_01= (!G1)*(C7)*O_ANDG;

ANDG_10= (G1)*(!C7)*O_ANDG;

ANDG_11= (G1)*(C7)*O_ANDG;

ANDF_01= (!F2)*(C7)*O_ANDF;

ANDF_10= (F2)*(!C7)*O_ANDF;

ANDF_11= (F2)*(C7)*O_ANDF;

XORG1_00= C0_ANDG * C0_CM1 * O_XORG1;

XORG1_01= C0_ANDG * C1_CM1 * O_XORG1;

XORG1_10= C1_ANDG * C0_CM1 * O_XORG1;

XORF1_00= C0_ANDF * C0_CM1 * O_XORF1;

XORF1_01= C0_ANDF * C1_CM1 * O_XORF1;

XORF1_10= C1_ANDF * C0_CM1 * O_XORF1;

XORG2_00= C0_XORG1 * (!G4) * O_XORG2;

XORG2_01= C0_XORG1 * (G4) * O_XORG2;

XORG2_10= C1_XORG1 * (!G4) * O_XORG2;

XORF2_00= C0_XORF1 * (!F1) * O_XORF2;

XORF2_01= C0_XORF1 * (F1) * O_XORF2;

XORF2_10= C1_XORF1 * (!F1) * O_XORF2;

GM1_00= (!C6) * C0_XORG2 * O_GM1;

GM1_10= (C6) * C0_XORG2 * O_GM1;

FM1_001= (!C3) * C0_XORF2 * (C2) * O_FM1;

FM1_010= (!C3) * C1_XORF2 * (!C2) * O_FM1;

FM1_110= (C3) * C1_XORF2 * (!C2) * O_FM1;

```
GM2_001= C0_GM1 * (!G4) * C1_FM2;

GM2_010= C0_GM1 * (G4) * C0_FM2;

GM2_101= C1_GM1 * (!G4) * C1_FM2;

GM2_110= C1_GM1 * (G4) * C0_FM2;

FM2_001= C0_FM1 * C0_CM2 * (Cin) * O_FM2;

FM2_010= C0_FM1 * C1_CM2 * (!Cin) * O_FM2;

FM2_101= C1_FM1 * C0_CM2 * (Cin) * O_FM2;

FM2_110= C1_FM1 * C1_CM2 * (!Cin) * O_FM2;

CM1_001= (!C0) * (!C1) * (F3) * (O_CM1_1 +O_CM1_2);

CM1_010= (!C0) * (C1) * (!F3) * (O_CM1_1 +O_CM1_2);

CM1_101= (C0) * (!C1) * (F3) * (O_CM1_1 +O_CM1_2);

CM2_00001= (!C4) * (!C5) * (!F1) * (!F3) * (C7) * O_CM2;

CM2_00110= (!C4) * (!C5) * (F1) * (F3) * (!C7) * O_CM2;

CM2_01010= (!C4) * (C5) * (!F1) * (F3) * (!C7) * O_CM2;

CM2_01101= (!C4) * (C5) * (F1) * (!F3) * (C7) * O_CM2;

CM2_11011= (C4) * (C5) * (!F1) * (F3) * (C7) * O_CM2;

CM2_11100= (C4) * (C5) * (F1) * (!F3) * (!C7) * O_CM2;
```

Part 4: Testability Equations for Single Fault Model of a CLM

ANDG_G1_0= (!G1)*(C7)*O_ANDG;

ANDG_G1_1= (G1)*(C7)*O_ANDG;

ANDG_C7_0= (G1)*(!C7)*O_ANDG;

ANDG_C7_1= (G1)*(C7)*O_ANDG;

ANDF_F2_0= (!F2)*(C7)*O_ANDF;

ANDF_F2_1= (F2)*(C7)*O_ANDF;

ANDF_C7_0= (F2)*(!C7)*O_ANDF;

ANDF_C7_1= (F2)*(C7)*O_ANDF;

XORG1_ANDG_0= C0_ANDG * O_ANDG;

XORG1_ANDG_1= C1_ANDG * O_ANDG;

XORG1_CM1_0= C0_CM1 * O_CM1_2;

XORG1_CM1_1= C1_CM1 * O_CM1_2;

XORF1_ANDF_0= C0_ANDF * O_ANDF;

XORF1_ANDF_1= C1_ANDF * O_ANDF;

XORF1_CM1_0= C0_CM1 * O_CM1_1;

XORF1_CM1_1= C1_CM1 * O_CM1_1;

XORG2_XORG1_0= C0_XORG1 * O_XORG1;

XORG2_XORG1_1= C1_XORG1 * O_XORG1;

XORG2_G4_0= (!G4) * O_XORG2;

XORG2_G4_1= G4 * O_XORG2;

XORF2_XORF1_0= C0_XORF1 * O_XORF1;

XORF2_XORF1_1= C1_XORF1 * O_XORF1;

XORF2_F1_0= (!F1) * O_XORF2;

```
XORF2_F1_1= F1 * O_XORF2;

GM1_C6_0= (!C6) * C0_XORG2 * O_GM1;

GM1_C6_1= (C6) * C0_XORG2 * O_GM1;

GM1_XORG2_0= (C6) * C0_XORG2 * O_XORG2;

GM1_XORG2_1= (C6) * C1_XORG2 * O_XORG2;

FM1_C3_0= (!C3) * ( (C2 * C0_XORF2) + ((!C2) * C1_XORF2) ) * O_FM1;

FM1_C3_1= (C3) * ( (C2 * C0_XORF2) + ((!C2) * C1_XORF2) ) * O_FM1;

FM1_XORF2_0= (C3) * C0_XORF2 * O_XORF2;

FM1_XORF2_1= (C3) * C1_XORF2 * O_XORF2;

FM1_C2_0= (!C3) * (!C2) * O_FM1;

FM1_C2_1= (!C3) * (C2) * O_FM1;

GM2_GM1_0= C0_GM1 * ( (G4*C0_FM2) + ((!G4)*C1_FM2) );

GM2_GM1_1= C1_GM1 * ( (G4*C0_FM2) + ((!G4)*C1_FM2) );

GM2_G4_0= C0_GM1 * (!G4);

GM2_G4_1= C0_GM1 * (G4);

GM2_FM2_0= C1_GM1 * C0_FM2;

GM2_FM2_1= C1_GM1 * C1_FM2;

FM2_FM1_0= C0_FM1 * ( (Cin*C0_CM2) + ((!Cin)*C1_CM2) ) * O_FM2;

FM2_FM1_1= C1_FM1 * ( (Cin*C0_CM2) + ((!Cin)*C1_CM2) ) * O_FM2;

FM2_CM2_0= C0_FM1 * C0_CM2 * O_FM2;

FM2_CM2_1= C0_FM1 * C1_CM2 * O_FM2;

FM2_Cin_0= C1_FM1 * (!Cin) * O_FM2;

FM2_Cin_1= C1_FM1 * (Cin) * O_FM2;

CM1_C0_0= (!C0) * ( ((!C1)*F3) +(C1*(!F3)) ) * (O_CM1_1 +O_CM1_2);

CM1_C0_1= (C0) * ( ((!C1)*F3) +(C1*(!F3)) ) * (O_CM1_1 +O_CM1_2);
```

```
CM1_C1_0= (!C0) * (!C1) * (O_CM1_1 +O_CM1_2);

CM1_C1_1= (!C0) * (C1) * (O_CM1_1 +O_CM1_2);

CM1_F3_0= (C0) * (!F3) * (O_CM1_1 +O_CM1_2);

CM1_F3_1= (C0) * (F3) * (O_CM1_1 +O_CM1_2);

CM2_C4C5_00= (!C4) * (!C5) * ( ((!F1)*(!F3)*(C7)) + ((F1)*(F3)*(!C7)) ) * O_CM2;

CM2_C4C5_01= (!C4) * (C5) * ( ((F1)*(!F3)*(C7)) + ((!F1)*(F3)*(!C7)) ) * O_CM2;

CM2_C4C5_11= (C4) * (C5) * ( ((!F1)*(F3)*(C7)) + ((F1)*(!F3)*(!C7)) ) * O_CM2;

CM2_F1_0= (C4) * (C5) * (!F1) * O_CM2;

CM2_F1_1= (C4) * (C5) * (F1) * O_CM2;

CM2_F3_0= (!C4) * (C5) * (!F3) * O_CM2;

CM2_F3_1= (!C4) * (C5) * (F3) * O_CM2;

CM2_C7_0= (!C4) * (!C5) * (!C7) * O_CM2;

CM2_C7_1= (!C4) * (!C5) * (C7) * O_CM2;

O_1 = (!C6) * O_GM1;
```

Appendix 2

The "Best" 22 CLM TCs

TC SET 1

ADD-G-F3
ADDSUB-FG-CI
INC-G-1
DEC-F-CI
FORCE-1
FORCE-CI
FORCE-F3
FORCE-F1

TC SET 2

ADD-G-F3
ADDSUB-FG-CI
INC-G-1
DEC-F-CI
FORCE-1
EXAMINE-CI
FORCE-F3
FORCE-F1

TC SET 3

ADD-G-F3
ADDSUB-FG-CI
INC-G-1
INCDEC-F-CI
FORCE-1
FORCE-CI
FORCE-F3
FORCE-F1

TC SET 4

ADD-G-F3
ADDSUB-FG-CI
INC-G-1
INCDEC-F-CI
FORCE-1
FORCE-CI
FORCE-F3
FORCE-F1

TC SET 5

ADD-G-F3
ADDSUB-FG-CI
DEC-G-0
DEC-F-CI
FORCE-1
FORCE-CI
FORCE-F3
FORCE-F1

TC SET 6

ADD-G-F3
ADDSUB-FG-CI
DEC-G-0
DEC-F-CI
FORCE-1
EXAMINE-CI
FORCE-F3
FORCE-F1

TC SET 7

ADD-G-F3
ADDSUB-FG-CI
DEC-G-0
INCDEC-F-CI
FORCE-1
FORCE-CI
FORCE-F3
FORCE-F1

TC SET 8

ADD-G-F3
ADDSUB-FG-CI
DEC-G-0
INCDEC-F-CI
FORCE-1
EXAMINE-CI
FORCE-F3
FORCE-F1

TC SET 9

ADD-G-F3
ADDSUB-FG-CI
INCDEC-G-0
DEC-F-CI
FORCE-1
FORCE-CI
FORCE-F3
FORCE-F1

TC SET 10

ADD-G-F3
ADDSUB-FG-CI
INCDEC-G-0
DEC-F-CI
FORCE-1
EXAMINE-CI
FORCE-F3
FORCE-F1

TC SET 11

ADD-G-F3
ADDSUB-FG-CI
INCDEC-G-0
INCDEC-F-CI
FORCE-1
FORCE-CI
FORCE-F3
FORCE-F1

TC SET 12

ADD-G-F3
ADDSUB-FG-CI
INCDEC-G-0
INCDEC-F-CI
FORCE-1
EXAMINE-CI
FORCE-F3
FORCE-F1

TC SET 13

ADD-G-F3
ADDSUB-FG-CI
INC-G-CI
DEC-F-CI
FORCE-0
FORCE-1
FORCE-F3
FORCE-F1

TC SET 14

ADD-G-F3
ADDSUB-FG-CI
DEC-F-CI
DEC-G-CI
FORCE-0
FORCE-1
FORCE-F3
FORCE-F1

TC SET 15

ADD-G-F3
ADDSUB-FG-CI
DEC-F-CI
INCDEC-G-CI
FORCE-0
FORCE-1
FORCE-F3
FORCE-F1

TC SET 16

ADD-G-F3
ADDSUB-FG-CI
DEC-G-CI
INCDEC-F-CI
FORCE-0
FORCE-1
FORCE-F3
FORCE-F1

TC SET 17

SUB-G-F3
ADDSUB-FG-CI
INC-G-CI
DEC-F-CI
FORCE-0
FORCE-1
FORCE-F3
FORCE-F1

TC SET 18

SUB-G-F3
ADDSUB-FG-CI
INC-G-CI
INCDEC-F-CI
FORCE-0
FORCE-1
FORCE-F3
FORCE-F1

TC SET 19

ADDSUB-FG-CI
ADDSUB-G-F3
INC-G-CI
DEC-F-CI
FORCE-0
FORCE-1
FORCE-F3
FORCE-F1

TC SET 20

ADDSUB-FG-CI
INC-G-CI
DEC-F-CI
FORCE-0
FORCE-1
FORCE-F3
ADD-G-F3
FORCE-F1

TC SET 21

ADDSUB-FG-CI
INC-G-CI
DEC-F-CI
FORCE-0
FORCE-1
FORCE-F3
SUB-G-F3
FORCE-F1

TC SET 22

ADDSUB-FG-CI
INC-G-CI
DEC-F-CI
FORCE-0
FORCE-1
FORCE-F3
ADDSUB-G-F3
FORCE-F1

Appendix 3

The 17 TCs for Interconnect Testing

In the following tables:

(a) A blank cell indicates that there are no programmable switches at the cross-point between the corresponding row and column.

(b) A cell with symbol "x" indicates there is a programmable switch at the cross-point between the corresponding row and column, but the programmable switch is not selected in the test configuration.

(c) A cell with logic 1 indicates there is a programmable switch at the cross-point between the corresponding row and column, and the programmable switch is selected in the test configuration.

TC1

(a) PS Selection between CLB pins and Horizontal Wire Segments

	HS8 (from top to down)								HD4 (from top to down)				HL6 (from top to down)					
	1	2	3	4	5	6	7	8	1	2	3	4	1	2	3	4	5	6
G1																		
G2	×	1	×	×	×	×	×	×	×	×	×	×			×	×	×	
G3																		
G4	×	×	×	1	×	×	×	×	×	×	×	×		×	×	×		
F1																		
F2	×	×	×	×	×	1	×	×	×	×	×	×			×	×	×	
F3																		
F4	×	×	×	×	×	×	×	1	×	×	×	×		×	×	×		
C1																		
C2	×	×	×	×	×	×	×	×	×	×	×	×		1			×	
C3																		
C4	×	×	×	×	×	×	1	×	×	×	×	×		×			×	
K																		
C_{TB1}																		
I_{TB1}																		
C_{TB2}																		
I_{TB2}																		
X	×				×							×						
XQ		×				×					×							
Y				×				×		×								
YQ			×				×		×									

(b) PS Selection between CLB pins and Vertical Wire Segments

	VS8 (from right to left)								VD4 (from right to left)				VL6 (from right to left)						VG4 (from right to left)			
	1	2	3	4	5	6	7	8	1	2	3	4	1	2	3	4	5	6	1	2	3	4
G1	1	×	×	×	×	×	×	×	×	×	×	×	×	×		×	×			×		
G2																						
G3	×	×	1	×	×	×	×	×	×	×	×	×		×	×		×	×	×			
G4																						
F1	×	×	×	×	1	×	×	×	×	×	×	×	×	×		×	×			×		
F2																						
F3	×	×	×	×	×	×	×	×	×	×	×	×		1	×		×	×	×			
F4																						
C1	×	×	×	×	×	×	×	×	×	×	×	×			1	×						×
C2																						
C3	×	×	×	×	×	×	×	×	×	1	×	×			×	×					×	
C4																						
K		×	×		×			×											1	×	×	×
C_{TB1}	×					×							×					×				
I_{TB1}				×			×															
C_{TB2}	×					×							×					×				
I_{TB2}		×			×																	
X			×				×			×												
XQ				×				×	×													
Y		×				×						×										
YQ	×				×						×											

TC2

(a) PS Selection between CLB pins and Horizontal Wire Segments

	HS8 (from top to down)								HD4 (from top to down)				HL6 (from top to down)					
	1	2	3	4	5	6	7	8	1	2	3	4	1	2	3	4	5	6
G1																		
G2	×	×	1	×	×	×	×	×	×	×	×	×			×	×	×	
G3																		
G4	×	×	×	×	1	×	×	×	×	×	×	×		×	×	×		
F1																		
F2	×	×	×	×	×	×	1	×	×	×	×	×			×	×	×	
F3																		
F4	×	×	×	×	×	×	×	×	1	×	×	×		×	×	×		
C1																		
C2	×	×	×	×	×	×	×	×	×	×	×	1		×			×	
C3																		
C4	×	×	×	×	×	×	×	×	×	×	1	×		×			×	
K																		
C_{TB1}																		
I_{TB1}																		
C_{TB2}																		
I_{TB2}																		
X	×				×							×						
XQ		×				×					×							
Y				×				×		×								
YQ			×				×		×									

(b) PS Selection between CLB pins and Vertical Wire Segments

	VS8 (from right to left)								VD4 (from right to left)				VL6 (from right to left)						VG4 (from right to left)			
	1	2	3	4	5	6	7	8	1	2	3	4	1	2	3	4	5	6	1	2	3	4
G1	×	1	×	×	×	×	×	×	×	×	×	×	×	×		×	×			×		
G2																						
G3	×	×	×	×	×	×	×	×	×	×	×	×		1	×		×	×	×			
G4																						
F1	×	×	×	×	×	1	×	×	×	×	×	×	×	×		×	×			×		
F2																						
F3	×	×	×	×	×	×	×	1	×	×	×	×		×	×		×	×	×			
F4																						
C1	×	×	×	×	×	×	×	×	1	×	×	×			×	×						×
C2																						
C3	1	×	×	×	×	×	×	×	×	×	×	×			×	×					×	
C4																						
K		×	×		×			×											×	1	×	×
C_{TB1}	×					×							×					1				
I_{TB1}				1			×															
C_{TB2}	×					×							×					1				
I_{TB2}		×			×																	
X			×				×			×												
XQ				1				×	×													
Y		×				×						×										
YQ	×				×						×											

TC3

(a) PS Selection between CLB pins and Horizontal Wire Segments

	HS8 (from top to down)								HD4 (from top to down)				HL6 (from top to down)					
	1	2	3	4	5	6	7	8	1	2	3	4	1	2	3	4	5	6
G1																		
G2	×	×	×	1	×	×	×	×	×	×	×	×			×	×	×	
G3																		
G4	×	×	×	×	×	×	×	×	1	×	×	×		×	×	×		
F1																		
F2	×	×	×	×	×	×	×	1	×	×	×	×			×	×	×	
F3																		
F4	×	×	×	×	×	×	×	×	×	1	×	×		×	×	×		
C1																		
C2	1	×	×	×	×	×	×	×	×	×	×	×		×			×	
C3																		
C4	×	×	×	×	×	×	×	×	×	×	×	×		×			1	
K																		
C_{TB1}																		
I_{TB1}																		
C_{TB2}																		
I_{TB2}																		
X	×				×							×						
XQ		×				×					×							
Y				×				×		×								
YQ			×				×		×									

(b) PS Selection between CLB pins and Vertical Wire Segments

	VS8 (from right to left)								VD4 (from right to left)				VL6 (from right to left)						VG4 (from right to left)			
	1	2	3	4	5	6	7	8	1	2	3	4	1	2	3	4	5	6	1	2	3	4
G1	×	×	1	×	×	×	×	×	×	×	×	×	×	×		×	×			×		
G2																						
G3	×	×	×	×	1	×	×	×	×	×	×	×		×	×		×	×	×			
G4																						
F1	×	×	×	×	×	×	1	×	×	×	×	×	×	×		×	×			×		
F2																						
F3	×	×	×	×	×	×	×	×	1	×	×	×		×	×		×	×	×			
F4																						
C1	×	×	×	×	×	×	×	×	×	×	1	×			×	×						×
C2																						
C3	×	×	×	×	×	×	×	×	×	×	×	×			1	×					×	
C4																						
K		×	×		×			×											×	×	1	×
C_{TB1}	1					×							×					×				
I_{TB1}				×			×															
C_{TB2}	1					×							×					×				
I_{TB2}		×			1																	
X			×				×			×												
XQ				×				×	×													
Y		×				×						×										
YQ	×				1						×											

TC4

(a) PS Selection between CLB pins and Horizontal Wire Segments

	HS8 (from top to down)								HD4 (from top to down)				HL6 (from top to down)					
	1	2	3	4	5	6	7	8	1	2	3	4	1	2	3	4	5	6
G1																		
G2	×	×	×	×	1	×	×	×	×	×	×	×			×	×	×	
G3																		
G4	×	×	×	×	×	×	1	×	×	×	×	×		×	×	×		
F1																		
F2	×	×	×	×	×	×	×	×	×	×	×	×			1	×	×	
F3																		
F4	×	×	×	×	×	×	×	×	×	×	1	×		×	×	×		
C1																		
C2	×	1	×	×	×	×	×	×	×	×	×	×		×			×	
C3																		
C4	1	×	×	×	×	×	×	×	×	×	×	×		×			×	
K																		
C_{TB1}																		
I_{TB1}																		
C_{TB2}																		
I_{TB2}																		
X	×				×							×						
XQ		×				×					×							
Y				×				×		×								
YQ			×				×		×									

(b) PS Selection between CLB pins and Vertical Wire Segments

	VS8 (from right to left)								VD4 (from right to left)				VL6 (from right to left)						VG4 (from right to left)			
	1	2	3	4	5	6	7	8	1	2	3	4	1	2	3	4	5	6	1	2	3	4
G1	×	×	×	1	×	×	×	×	×	×	×	×	×	×		×	×			×		
G2																						
G3	×	×	×	×	×	×	×	×	×	×	×	×		×	1		×	×	×			
G4																						
F1	×	×	×	×	×	×	×	1	×	×	×	×	×	×		×	×			×		
F2																						
F3	×	×	×	×	×	×	×	×	×	×	×	×		×	×		×	1	×			
F4																						
C1	×	×	×	×	×	×	×	×	×	×	×	×			×	×						1
C2																						
C3	×	×	×	×	×	×	×	×	×	×	×	×			×	1					×	
C4																						
K		×	1		×			×											×	×	×	×
C_{TB1}	×					1							×					×				
I_{TB1}				×			×															
C_{TB2}	×					1							×					×				
I_{TB2}		1			×																	
X			×				×			×												
XQ				×				×	×													
Y		1				×						×										
YQ	×				×						×											

TC5

(a) PS Selection between CLB pins and Horizontal Wire Segments

	HS8 (from top to down)								HD4 (from top to down)				HL6 (from top to down)					
	1	2	3	4	5	6	7	8	1	2	3	4	1	2	3	4	5	6
G1																		
G2	×	×	×	×	×	1	×	×	×	×	×	×			×	×	×	
G3																		
G4	×	×	×	×	×	×	×	1	×	×	×	×		×	×	×		
F1																		
F2	×	×	×	×	×	×	×	×	×	1	×	×			×	×	×	
F3																		
F4	×	×	×	×	×	×	×	×	×	×	×	1		×	×	×		
C1																		
C2	×	×	×	×	×	×	1	×	×	×	×	×		×			×	
C3																		
C4	×	1	×	×	×	×	×	×	×	×	×	×		×			×	
K																		
C_{TB1}																		
I_{TB1}																		
C_{TB2}																		
I_{TB2}																		
X	×				×							×						
XQ		×				×					×							
Y				×				×		×								
YQ			×				×		×									

(b) PS Selection between CLB pins and Vertical Wire Segments

	VS8 (from right to left)								VD4 (from right to left)				VL6 (from right to left)						VG4 (from right to left)			
	1	2	3	4	5	6	7	8	1	2	3	4	1	2	3	4	5	6	1	2	3	4
G1	×	×	×	×	1	×	×	×	×	×	×	×	×	×		×	×			×		
G2																						
G3	×	×	×	1	×	×	×	×	×	×	×	×		×	×		×	×	×			
G4																						
F1	×	×	×	×	×	×	×	×	1	×	×	×	×	×		×	×			×		
F2																						
F3	×	×	×	×	×	×	×	×	×	×	1	×		×	×		×	×	×			
F4																						
C1	×	×	1	×	×	×	×	×	×	×	×	×			×	×						×
C2																						
C3	×	×	×	×	×	×	×	×	×	×	×	×			×	×					1	
C4																						
K		×	×		×			×											×	×	×	1
C_{TB1}	×					×							×					1				
I_{TB1}				×			×															
C_{TB2}	×					×							×					1				
I_{TB2}		×			×																	
X			1				×			×												
XQ				×				×	×													
Y		×				×						×										
YQ	×				×						×											

TC6

(a) PS Selection between CLB pins and Horizontal Wire Segments

	HS8 (from top to down)								HD4 (from top to down)				HL6 (from top to down)					
	1	2	3	4	5	6	7	8	1	2	3	4	1	2	3	4	5	6
G1																		
G2	×	×	×	×	×	×	1	×	×	×	×	×			×	×	×	
G3																		
G4	×	×	×	×	×	×	×	×	×	×	×	×		×	1	×		
F1																		
F2	×	×	×	×	×	×	×	×	×	×	×	1			×	×	×	
F3																		
F4	×	×	×	×	×	×	×	×	×	×	×	×		1	×	×		
C1																		
C2	×	×	×	1	×	×	×	×	×	×	×	×		×			×	
C3																		
C4	×	×	1	×	×	×	×	×	×	×	×	×		×			×	
K																		
C_{TB1}																		
I_{TB1}																		
C_{TB2}																		
I_{TB2}																		
X	×				×							×						
XQ		×				×					×							
Y				×				×		×								
YQ			×				×		×									

(b) PS Selection between CLB pins and Vertical Wire Segments

	VS8 (from right to left)								VD4 (from right to left)				VL6 (from right to left)						VG4 (from right to left)			
	1	2	3	4	5	6	7	8	1	2	3	4	1	2	3	4	5	6	1	2	3	4
G1	×	×	×	×	×	1	×	×	×	×	×	×	×	×		×	×			×		
G2																						
G3	×	×	×	×	×	×	×	×	×	×	×	×		×	×		1	×	×			
G4																						
F1	×	×	×	×	×	×	×	×	×	×	1	×	×	×		×	×			×		
F2																						
F3	×	×	×	×	×	×	×	×	×	×	×	×		×	1		×	×	×			
F4																						
C1	×	×	×	×	×	×	×	×	×	1	×	×			×	×						×
C2																						
C3	×	1	×	×	×	×	×	×	×	×	×	×			×	×					×	
C4																						
K		×	×		×			×											×	×	×	1
C_{TB1}	×					×							1					×				
I_{TB1}				×			×															
C_{TB2}	×					×							1					×				
I_{TB2}		×			×																	
X			×				×			1												
XQ				×				×	×													
Y		×				×						×										
YQ	×				×						×											

TC7

(a) PS Selection between CLB pins and Horizontal Wire Segments

	HS8 (from top to down)								HD4 (from top to down)				HL6 (from top to down)					
	1	2	3	4	5	6	7	8	1	2	3	4	1	2	3	4	5	6
G1																		
G2	×	×	×	×	×	×	×	1	×	×	×	×			×	×	×	
G3																		
G4	×	×	×	×	×	×	×	×	×	1	×	×		×	×	×		
F1																		
F2	×	×	×	×	1	×	×	×	×	×	×	×			×	×	×	
F3																		
F4	×	×	×	×	×	×	×	×	×	×	×	×		×	×	1		
C1																		
C2	×	×	×	×	×	×	×	×	1	×	×	×		×			×	
C3																		
C4	×	×	×	1	×	×	×	×	×	×	×	×		×			×	
K																		
C_{TB1}																		
I_{TB1}																		
C_{TB2}																		
I_{TB2}																		
X	1				×							×						
XQ		×				×					×							
Y				×				×		×								
YQ			×				×		×									

(b) PS Selection between CLB pins and Vertical Wire Segments

	VS8 (from right to left)								VD4 (from right to left)				VL6 (from right to left)						VG4 (from right to left)			
	1	2	3	4	5	6	7	8	1	2	3	4	1	2	3	4	5	6	1	2	3	4
G1	×	×	×	×	×	×	1	×	×	×	×	×	×	×		×	×			×		
G2																						
G3	×	×	×	×	×	1	×	×	×	×	×	×		×	×		×	×	×			
G4																						
F1	×	×	×	×	×	×	×	×	×	×	×	1	×	×		×	×			×		
F2																						
F3	×	×	×	×	×	×	×	×	×	×	×	×		×	×		1	×	×			
F4																						
C1	1	×	×	×	×	×	×	×	×	×	×	×			×	×						×
C2																						
C3	×	×	1	×	×	×	×	×	×	×	×	×			×	×					×	
C4																						
K		×	×		×			×											×	×	×	1
C_{TB1}	×					×							×					1				
I_{TB1}				×			×															
C_{TB2}	×					×							×					1				
I_{TB2}		×			×																	
X			×				×			×												
XQ				×				×	×													
Y		×				×						×										
YQ	×				×						×											

TC8

(a) PS Selection between CLB pins and Horizontal Wire Segments

	HS8 (from top to down)								HD4 (from top to down)				HL6 (from top to down)					
	1	2	3	4	5	6	7	8	1	2	3	4	1	2	3	4	5	6
G1																		
G2	×	×	×	×	×	×	×	×	1	×	×	×			×	×	×	
G3																		
G4	×	×	×	×	×	×	×	×	×	×	1	×		×	×	×		
F1																		
F2	×	×	×	×	×	×	×	×	×	×	×	×			×	×	1	
F3																		
F4	×	×	×	×	×	×	×	×	×	×	×	×		×	1	×		
C1																		
C2	×	×	1	×	×	×	×	×	×	×	×	×		×			×	
C3																		
C4	×	×	×	×	×	1	×	×	×	×	×	×		×			×	
K																		
C_{TB1}																		
I_{TB1}																		
C_{TB2}																		
I_{TB2}																		
X	×				1							×						
XQ		×				×					×							
Y				×				×		×								
YQ			×				×		×									

(b) PS Selection between CLB pins and Vertical Wire Segments

	VS8 (from right to left)								VD4 (from right to left)				VL6 (from right to left)						VG4 (from right to left)			
	1	2	3	4	5	6	7	8	1	2	3	4	1	2	3	4	5	6	1	2	3	4
G1	×	×	×	×	×	×	×	1	×	×	×	×	×	×		×	×			×		
G2																						
G3	1	×	×	×	×	×	×	×	×	×	×	×		×	×		×	×	×			
G4																						
F1	×	×	×	×	×	×	×	×	×	1	×	×	×	×		×	×			×		
F2																						
F3	×	×	×	×	×	×	×	×	×	×	×	×		×	×		×	×	1			
F4																						
C1	×	×	×	×	1	×	×	×	×	×	×	×			×	×						×
C2																						
C3	×	×	×	×	×	×	1	×	×	×	×	×			×	×					×	
C4																						
K		1	×		×			×											×	×	×	×
C_{TB1}	×					×							×					1				
I_{TB1}				×			×															
C_{TB2}	×					×							×					1				
I_{TB2}		×			×																	
X			×				×			×												
XQ				×				×	×													
Y		×				×						×										
YQ	×				×						×											

TC9

(a) PS Selection between CLB pins and Horizontal Wire Segments

	HS8 (from top to down)								HD4 (from top to down)				HL6 (from top to down)					
	1	2	3	4	5	6	7	8	1	2	3	4	1	2	3	4	5	6
G1																		
G2	×	×	×	×	×	×	×	×	×	1	×	×			×	×	×	
G3																		
G4	×	×	×	×	×	×	×	×	×	×	×	×		1	×	×		
F1																		
F2	×	×	×	×	×	×	×	×	1	×	×	×			×	×	×	
F3																		
F4	1	×	×	×	×	×	×	×	×	×	×	×		×	×	×		
C1																		
C2	×	×	×	×	×	1	×	×	×	×	×	×		×			×	
C3																		
C4	×	×	×	×	×	×	×	×	×	×	×	1		×			×	
K																		
C_{TB1}																		
I_{TB1}																		
C_{TB2}																		
I_{TB2}																		
X	×				×							1						
XQ		×				×					×							
Y				×				×		×								
YQ			×				×		×									

(b) PS Selection between CLB pins and Vertical Wire Segments

	VS8 (from right to left)								VD4 (from right to left)				VL6 (from right to left)						VG4 (from right to left)			
	1	2	3	4	5	6	7	8	1	2	3	4	1	2	3	4	5	6	1	2	3	4
G1	×	×	×	×	×	×	×	×	×	×	×	×	×	×		×	1			×		
G2																						
G3	×	×	×	×	×	×	×	1	×	×	×	×		×	×		×	×	×			
G4																						
F1	×	×	×	×	×	×	×	×	×	×	×	×	×	×		1	×			×		
F2																						
F3	×	×	×	×	×	×	×	×	×	1	×	×		×	×		×	×	×			
F4																						
C1	×	1	×	×	×	×	×	×	×	×	×	×			×	×						×
C2																						
C3	×	×	×	×	1	×	×	×	×	×	×	×			×	×					×	
C4																						
K		×	×		×			×											×	×	×	1
C_{TB1}	×					×							1					×				
I_{TB1}				×			×															
C_{TB2}	×					×							1					×				
I_{TB2}		×			×																	
X			×				×			×												
XQ				×				×	×													
Y		×				×						×										
YQ	×				×						×											

TC10

(a) PS Selection between CLB pins and Horizontal Wire Segments

	HS8 (from top to down)								HD4 (from top to down)				HL6 (from top to down)					
	1	2	3	4	5	6	7	8	1	2	3	4	1	2	3	4	5	6
G1																		
G2	×	×	×	×	×	×	×	×	×	×	1	×			×	×	×	
G3																		
G4	×	×	×	×	×	×	×	×	×	×	×	1		×	×	×		
F1																		
F2	×	×	×	×	×	×	×	×	×	×	×	×			×	1	×	
F3																		
F4	×	1	×	×	×	×	×	×	×	×	×	×		×	×	×		
C1																		
C2	×	×	×	×	×	×	×	×	×	×	×	×		×			1	
C3																		
C4	×	×	×	×	×	×	×	1	×	×	×	×		×			×	
K																		
C_{TB1}																		
I_{TB1}																		
C_{TB2}																		
I_{TB2}																		
X	×				×							×						
XQ		×				×					×							
Y				×				×		×								
YQ			×				×		×									

(b) PS Selection between CLB pins and Vertical Wire Segments

	VS8 (from right to left)								VD4 (from right to left)				VL6 (from right to left)						VG4 (from right to left)			
	1	2	3	4	5	6	7	8	1	2	3	4	1	2	3	4	5	6	1	2	3	4
G1	×	×	×	×	×	×	×	×	×	1	×	×	×	×		×	×			×		
G2																						
G3	×	×	×	×	×	×	×	×	1	×	×	×		×	×		×	×	×			
G4																						
F1	×	×	×	×	×	×	×	×	×	×	×	×	×	×		×	1			×		
F2																						
F3	×	×	×	×	×	×	×	×	×	×	×	1		×	×		×	×	×			
F4																						
C1	×	×	×	1	×	×	×	×	×	×	×	×			×	×						×
C2																						
C3	×	×	×	×	×	1	×	×	×	×	×	×			×	×					×	
C4																						
K		×	×		×			1											×	×	×	×
C_{TB1}	×					×							×					1				
I_{TB1}				×			1															
C_{TB2}	×					×							×					1				
I_{TB2}		×			×																	
X			×				1			×												
XQ				×				×	×													
Y		×				1						×										
YQ	×				×						×											

TC11

(a) PS Selection between CLB pins and Horizontal Wire Segments

	HS8 (from top to down)								HD4 (from top to down)				HL6 (from top to down)					
	1	2	3	4	5	6	7	8	1	2	3	4	1	2	3	4	5	6
G1																		
G2	×	×	×	×	×	×	×	×	×	×	×	×			1	×	×	
G3																		
G4	×	×	×	×	×	×	×	×	×	×	×	×		×	×	1		
F1																		
F2	1	×	×	×	×	×	×	×	×	×	×	×			×	×	×	
F3																		
F4	×	×	1	×	×	×	×	×	×	×	×	×		×	×	×		
C1																		
C2	×	×	×	×	1	×	×	×	×	×	×	×		×			×	
C3																		
C4	×	×	×	×	×	×	×	×	×	×	×	×		1			×	
K																		
C_{TB1}																		
I_{TB1}																		
C_{TB2}																		
I_{TB2}																		
X	×				×							×						
XQ		×				×					×							
Y				×				×		×								
YQ			×				×		×									

(b) PS Selection between CLB pins and Vertical Wire Segments

	VS8 (from right to left)								VD4 (from right to left)				VL6 (from right to left)						VG4 (from right to left)			
	1	2	3	4	5	6	7	8	1	2	3	4	1	2	3	4	5	6	1	2	3	4
G1	×	×	×	×	×	×	×	×	×	×	×	×	×	1		×	×			×		
G2																						
G3	×	×	×	×	×	×	×	×	×	1	×	×		×	×		×	×	×			
G4																						
F1	×	×	×	×	×	×	×	×	×	×	×	×	×	×		×	×			1		
F2																						
F3	×	1	×	×	×	×	×	×	×	×	×	×		×	×		×	×	×			
F4																						
C1	×	×	×	×	×	1	×	×	×	×	×	×			×	×						×
C2																						
C3	×	×	×	×	×	×	×	×	×	×	×	1			×	×					×	
C4																						
K		×	×		×			×											×	×	×	1
C_{TB1}	×					×							×					1				
I_{TB1}				×			×															
C_{TB2}	×					×							×					1				
I_{TB2}		×			×																	
X			×				×			×												
XQ				×				×	×													
Y		×				×						1										
YQ	×				×						×											

TC12

(a) PS Selection between CLB pins and Horizontal Wire Segments

	HS8 (from top to down)								HD4 (from top to down)				HL6 (from top to down)					
	1	2	3	4	5	6	7	8	1	2	3	4	1	2	3	4	5	6
G1																		
G2	×	×	×	×	×	×	×	×	×	×	×	×			×	1	×	
G3																		
G4	×	×	×	×	×	1	×	×	×	×	×	×		×	×	×		
F1																		
F2	×	1	×	×	×	×	×	×	×	×	×	×			×	×	×	
F3																		
F4	×	×	×	×	1	×	×	×	×	×	×	×		×	×	×		
C1																		
C2	×	×	×	×	×	×	×	1	×	×	×	×		×			×	
C3																		
C4	×	×	×	×	×	×	×	×	×	1	×	×		×			×	
K																		
C_{TB1}																		
I_{TB1}																		
C_{TB2}																		
I_{TB2}																		
X	×				×							×						
XQ		×				×					×							
Y				1				×		×								
YQ			×				×		×									

(b) PS Selection between CLB pins and Vertical Wire Segments

	VS8 (from right to left)								VD4 (from right to left)				VL6 (from right to left)						VG4 (from right to left)			
	1	2	3	4	5	6	7	8	1	2	3	4	1	2	3	4	5	6	1	2	3	4
G1	×	×	×	×	×	×	×	×	×	×	×	1	×	×		×	×			×		
G2																						
G3	×	×	×	×	×	×	×	×	×	×	1	×		×	×		×	×	×			
G4																						
F1	1	×	×	×	×	×	×	×	×	×	×	×	×	×		×	×			×		
F2																						
F3	×	×	1	×	×	×	×	×	×	×	×	×		×	×		×	×	×			
F4																						
C1	×	×	×	×	×	×	1	×	×	×	×	×			×	×						×
C2																						
C3	×	×	×	1	×	×	×	×	×	×	×	×			×	×					×	
C4																						
K		×	×		×			×											×	×	×	1
C_{TB1}	×					×							×					1				
I_{TB1}				×			×															
C_{TB2}	×					×							×					1				
I_{TB2}		×			×																	
X			×				×			×												
XQ				×				×	×													
Y		×				×						×										
YQ	×				×						×											

TC13

(a) PS Selection between CLB pins and Horizontal Wire Segments

	HS8 (from top to down)								HD4 (from top to down)				HL6 (from top to down)					
	1	2	3	4	5	6	7	8	1	2	3	4	1	2	3	4	5	6
G1																		
G2	×	×	×	×	×	×	×	×	×	×	×	1			×	×	×	
G3																		
G4	1	×	×	×	×	×	×	×	×	×	×	×		×	×	×		
F1																		
F2	×	×	1	×	×	×	×	×	×	×	×	×			×	×	×	
F3																		
F4	×	×	×	×	×	1	×	×	×	×	×	×		×	×	×		
C1																		
C2	×	×	×	×	×	×	×	×	×	×	×	×		1			×	
C3																		
C4	×	×	×	×	1	×	×	×	×	×	×	×		×			×	
K																		
C_{TB1}																		
I_{TB1}																		
C_{TB2}																		
I_{TB2}																		
X	×				×							×						
XQ		×				×					×							
Y				×				1		×								
YQ			×				×		×									

(b) PS Selection between CLB pins and Vertical Wire Segments

	VS8 (from right to left)								VD4 (from right to left)				VL6 (from right to left)						VG4 (from right to left)			
	1	2	3	4	5	6	7	8	1	2	3	4	1	2	3	4	5	6	1	2	3	4
G1	×	×	×	×	×	×	×	×	×	×	×	×	1	×		×	×			×		
G2																						
G3	×	×	×	×	×	×	×	×	×	×	×	×		×	×		×	1	×			
G4																						
F1	×	1	×	×	×	×	×	×	×	×	×	×	×	×		×	×			×		
F2																						
F3	×	×	×	×	×	×	1	×	×	×	×	×		×	×		×	×	×			
F4																						
C1	×	×	×	×	×	×	×	×	×	×	×	×			×	1						×
C2																						
C3	×	×	×	×	×	×	×	1	×	×	×	×			×	×					×	
C4																						
K		×	×		×			×											×	×	×	1
C_{TB1}	×					×							1					×				
I_{TB1}				×			×															
C_{TB2}	×					×							1					×				
I_{TB2}		×			×																	
X			×				×			×												
XQ				×				×	×													
Y		×				×						×										
YQ	×				×						×											

TC14

(a) PS Selection between CLB pins and Horizontal Wire Segments

	HS8 (from top to down)								HD4 (from top to down)				HL6 (from top to down)					
	1	2	3	4	5	6	7	8	1	2	3	4	1	2	3	4	5	6
G1																		
G2	×	×	×	×	×	×	×	×	×	×	×	×			×	×	1	
G3																		
G4	×	×	1	×	×	×	×	×	×	×	×	×		×	×	×		
F1																		
F2	×	×	×	×	×	×	×	×	×	×	1	×			×	×	×	
F3																		
F4	×	×	×	1	×	×	×	×	×	×	×	×		×	×	×		
C1																		
C2	×	×	×	×	×	×	×	×	×	1	×	×		×			×	
C3																		
C4	×	×	×	×	×	×	×	×	1	×	×	×		×			×	
K																		
C_{TB1}																		
I_{TB1}																		
C_{TB2}																		
I_{TB2}																		
X	×				×							×						
XQ		×				×					×							
Y				×				×		1								
YQ			×				×		×									

(b) PS Selection between CLB pins and Vertical Wire Segments

	VS8 (from right to left)								VD4 (from right to left)				VL6 (from right to left)						VG4 (from right to left)			
	1	2	3	4	5	6	7	8	1	2	3	4	1	2	3	4	5	6	1	2	3	4
G1	×	×	×	×	×	×	×	×	×	×	×	×	×	×		1	×			×		
G2																						
G3	×	×	×	×	×	×	1	×	×	×	×	×		×	×		×	×	×			
G4																						
F1	×	×	×	×	×	×	×	×	×	×	×	×	1	×		×	×			×		
F2																						
F3	1	×	×	×	×	×	×	×	×	×	×	×		×	×		×	×	×			
F4																						
C1	×	×	×	×	×	×	×	×	×	×	×	1			×	×						×
C2																						
C3	×	×	×	×	×	×	×	×	×	×	1	×			×	×					×	
C4																						
K		×	×		1			×											×	×	×	×
C_{TB1}	×					×							×					1				
I_{TB1}				×			×															
C_{TB2}	×					×							×					1				
I_{TB2}		×			×																	
X			×				×			×												
XQ				×				×	×													
Y		×				×						×										
YQ	×				×						×											

TC15

(a) PS Selection between CLB pins and Horizontal Wire Segments

	HS8 (from top to down)								HD4 (from top to down)				HL6 (from top to down)					
	1	2	3	4	5	6	7	8	1	2	3	4	1	2	3	4	5	6
G1																		
G2	1	×	×	×	×	×	×	×	×	×	×	×			×	×	×	
G3																		
G4	×	1	×	×	×	×	×	×	×	×	×	×		×	×	×		
F1																		
F2	×	×	×	1	×	×	×	×	×	×	×	×			×	×	×	
F3																		
F4	×	×	×	×	×	×	1	×	×	×	×	×		×	×	×		
C1																		
C2	×	×	×	×	×	1	×	×	×	×	×	×		×			×	
C3																		
C4	×	×	×	×	×	×	×	×	×	×	×	×		×			×	
K																		
C_{TB1}																		
I_{TB1}																		
C_{TB2}																		
I_{TB2}																		
X	×				×							×						
XQ		×				1					×							
Y				×				×		×								
YQ			×				×		×									

(b) PS Selection between CLB pins and Vertical Wire Segments

	VS8 (from right to left)								VD4 (from right to left)				VL6 (from right to left)						VG4 (from right to left)			
	1	2	3	4	5	6	7	8	1	2	3	4	1	2	3	4	5	6	1	2	3	4
G1	×	×	×	×	×	×	×	×	×	×	1	×	×	×		×	×			×		
G2																						
G3	×	×	×	×	×	×	×	×	×	×	×	×		×	×		×	×	1			
G4																						
F1	×	×	1	×	×	×	×	×	×	×	×	×	×	×		×	×			×		
F2																						
F3	×	×	×	×	1	×	×	×	×	×	×	×		×	×		×	×	×			
F4																						
C1	×	×	×	×	×	×	×	1	×	×	×	×			×	×						×
C2																						
C3	×	×	×	×	×	×	×	×	1	×	×	×			×	×					×	
C4																						
K		×	×		×			×											×	×	×	×
C_{TB1}	×					×							×					1				
I_{TB1}				×			×															
C_{TB2}	×					×							×					1				
I_{TB2}		×			×																	
X			×				×			×												
XQ				×				×	1													
Y		×				×						×										
YQ	×				×						×											

TC16

(a) PS Selection between CLB pins and Horizontal Wire Segments

	HS8 (from top to down)								HD4 (from top to down)				HL6 (from top to down)					
	1	2	3	4	5	6	7	8	1	2	3	4	1	2	3	4	5	6
G1																		
G2	×	×	×	×	×	×	×	×	×	×	×	×			×	×	×	
G3																		
G4	×	×	1	×	×	×	×	×	×	×	×	×		×	×	×		
F1																		
F2	×	×	×	×	×	×	×	×	×	×	×	×			×	×	×	
F3																		
F4	×	×	×	×	×	×	×	×	1	×	×	×		×	×	×		
C1																		
C2	×	×	×	×	×	×	×	×	×	×	1	×		×			×	
C3																		
C4	×	×	×	×	×	×	×	×	×	×	×	×		×			×	
K																		
C_{TB1}																		
I_{TB1}																		
C_{TB2}																		
I_{TB2}																		
X	×				×							×						
XQ		×				×					1							
Y				×				×		×								
YQ			1				×		1									

(b) PS Selection between CLB pins and Vertical Wire Segments

	VS8 (from right to left)								VD4 (from right to left)				VL6 (from right to left)						VG4 (from right to left)			
	1	2	3	4	5	6	7	8	1	2	3	4	1	2	3	4	5	6	1	2	3	4
G1	×	×	×	×	×	×	×	×	1	×	×	×	×	×		×	×			×		
G2																						
G3	×	1	×	×	×	×	×	×	×	×	×	×		×	×		×	×	×			
G4																						
F1	×	×	×	1	×	×	×	×	×	×	×	×	×	×		×	×			×		
F2																						
F3	×	×	×	×	×	1	×	×	×	×	×	×		×	×		×	×	×			
F4																						
C1	×	×	×	×	×	×	1	×	×	×	×	×			×	×						×
C2																						
C3	×	×	×	×	×	×	×	×	×	×	1	×			×	×					×	
C4																						
K		×	×		×			×											×	×	×	×
C_{TB1}	×					×							×					×				
I_{TB1}				×			×															
C_{TB2}	×					×							×					×				
I_{TB2}		×			×																	
X			×				×			×												
XQ				×				×	×													
Y		×				×						×										
YQ	×				×						×											

TC17

(a) PS Selection between CLB pins and Horizontal Wire Segments

	HS8 (from top to down)								HD4 (from top to down)				HL6 (from top to down)					
	1	2	3	4	5	6	7	8	1	2	3	4	1	2	3	4	5	6
G1																		
G2	1	×	×	×	×	×	×	×	×	×	×	×			×	×	×	
G3																		
G4	×	×	×	×	×	×	1	×	×	×	×	×		×	×	×		
F1																		
F2	×	1	×	×	×	×	×	×	×	×	×	×			×	×	×	
F3																		
F4	×	×	×	×	×	×	×	×	×	×	1	×		×	×	×		
C1																		
C2	×	×	×	×	×	×	×	×	×	×	×	×		×			×	
C3																		
C4	×	×	×	×	×	×	×	×	×	×	×	×		×			×	
K																		
C_{TB1}																		
I_{TB1}																		
C_{TB2}																		
I_{TB2}																		
X	×				×							×						
XQ		1				×					×							
Y				×				×		×								
YQ			×				1		×									

(b) PS Selection between CLB pins and Vertical Wire Segments

	VS8 (from right to left)								VD4 (from right to left)				VL6 (from right to left)						VG4 (from right to left)			
	1	2	3	4	5	6	7	8	1	2	3	4	1	2	3	4	5	6	1	2	3	4
G1	×	×	×	×	×	×	×	×	×	×	×	×	×	×		×	×			1		
G2																						
G3	×	×	×	×	×	×	×	×	×	×	×	1		×	×		×	×	×			
G4																						
F1	×	×	×	×	×	×	×	×	×	×	×	×	×	1		×	×			×		
F2																						
F3	×	×	×	1	×	×	×	×	×	×	×	×		×	×		×	×	×			
F4																						
C1	×	×	×	×	×	×	×	1	×	×	×	×			×	×						×
C2																						
C3	×	×	×	×	×	1	×	×	×	×	×	×			×	×					×	
C4																						
K		×	×		×			×											×	×	×	×
C_{TB1}	×					×							×					×				
I_{TB1}				×			×															
C_{TB2}	×					×							×					×				
I_{TB2}		×			×																	
X			×				×			×												
XQ				×				1	×													
Y		×				×						×										
YQ	1				×						1											

Appendix 4

Distribution of CLB Resources into Interconnect TCs

TC #	Carry	M_{F1}	M_{G1}	M_{G2}	F	G	M_{H1}	M_{H2}	H
1	-	**1**	**1**	**1**	**4-XOR**	**4-XOR**	**1**	**0**	**3-XOR**
2	**ADDSUB-FG-CI**	**0**	1	**0**	**4-NXOR**	**4-NXOR**	1	0	**3-NXOR**
3	-	1	1	1	**4-XOR**	**4-XOR**	1	0	**3-XOR**
4	**FORCE-CI**	0	1	1	**32x1 edge single**		1	0	HF + $\overline{HG}$
5	-	1	1	1	**16x2 edge single**		-	-	-
6	-	1	1	1	**32x1 level single**		1	0	HF + $\overline{HG}$
7	-	1	1	1	**16x2 level single**		1	0	HF + $\overline{HG}$
8	-	1	1	1	**16x1 edge dual**		1	0	HF + $\overline{HG}$
9	**ADD-G-F3**	0	1	0	4-XOR	3-XOR	1	0	3-NXOR
10	**FORCE-1**	1	**0**	1	4-XOR	4-XOR	**0**	**1**	3-NXOR
11	**DEC-F-CI**	0	1	1	F1 XNOR F4	4-XOR			3-XOR
12	**INC-G-1**	0	1	1	4-XOR	$\overline{G4}$			3-XOR

TC #	M1	M2	M3	M4	M_{K1}	M_{K2}	M_{K3}	H1	Din	SR	EC	M_{YQ}	M_{XQ}	FF(YQ)	FF(XQ)
1	-	-	**0**	**0**	-	-	**0**	**0**	**1**	**2**	**3**	-	**1**	-	**FF/Set**
2	-	-	0	0	-	-	**1**	**1**	**2**	**3**	**0**	-	1	-	**FF/Reset**
3	**0**	**1**	-	-	-	**0**	-	**2**	**3**	**0**	**1**	**1**	-	**FF/Reset**	-
4	0	1	-	-	**1**	**1**	-	**3**	**0**	**1**	**2**	1	-	**FF/Set**	-
5	0	-	**2**	**1**	**0**	0	0	0	1	2	3	1	1	FF/Reset	FF/Reset
6	**3**	-	-	1	1	0	-	2	3	0	1	1	**0**	FF/Set	-
7	-	-	0	0	0	-	1	0	1	2	3	**0**	1	-	FF/Set
8	**2**	-	**3**	1	1	0	0	2	3	0	1	1	1	FF/Reset	FF/Reset
9	0	1	-	0	-	1	-	2	3	0	1	1	1	FF/Set	-
10	0	**0**	-	0	-	1	-	3	0	1	2	1	-	FF/Reset	-
11	0	1	**1**	-	-	0	0	0	1	2	3	1	1	FF/Set	FF/Set
12	**1**	1	0	-	-	1	1	0	1	2	3	1	1	FF/Reset	FF/Reset

Appendix 5

VHDL Code for the BIST Circuitry

```
------------------------------------------------------------
-- A N bit Test Pattern Generator with Odd Parity
-- Author         : Jian Xu
-- Student ID     : 0392076
-- Date           : September11, 2001
-- File Name      : gentpg.vhd
-- Architecture   : Behavioral
-- Description    : The width of the TPG is determined by generic N
------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

-- BLACK BOX description of I/O's
entity gentpg is

   generic (n: positive := 8);

   port ( clock     : in std_logic;
          reset     : in std_logic;
          advance   : in std_logic;
          mode      : in std_logic_vector(2 downto 0);
          output    : out std_logic_vector((n-1) downto 0);
          odd_parity        : out std_logic;
          done              : out std_logic
          );

end gentpg;

architecture behavioral of gentpg is

          signal temp_out: std_logic_vector((n-1) downto 0);
          signal temp_out_minus_1bit: std_logic_vector((n-2) downto 0);
          signal monitor_all_0_1 : std_logic_vector((n-1) downto 0);
          signal mode_previous: std_logic_vector( 2 downto 0 );

BEGIN

main_proc: PROCESS (clock, mode)

BEGIN

IF ( clock'EVENT AND clock = '1' ) THEN
-- rising edge of clock

          IF ( reset = '1' OR mode_previous /= mode ) THEN
          -- sy. reset active high

                    case mode is

                              when "010" => --Walk 1
                                        temp_out_minus_1bit <= ( others => '0');
                                        temp_out <= '1' & temp_out_minus_1bit;
```

```
                    when "100" => --Count UP
                            temp_out <= (others => '0');

                    when "110" => --ALL 0
                            temp_out <= (others => '0');
                            monitor_all_0_1 <= (others => '0');
                            --use a count-up counter to monitor the end of all 0 test

                    when "111" => --ALL 1
                            temp_out <= (others => '1');
                            monitor_all_0_1 <= (others => '0');
                            --use a count-up counter to monitor the end of all 1 test

                    when others =>
                            temp_out <= (others => '0');

            end case;

            mode_previous <= mode;

        ELSIF ( advance = '1' ) THEN
        -- if advance signal high, progress pattern

            case mode is

                    when "010" => --Walk 1
                            temp_out <= '0' & temp_out((n-1) downto 1);
                            -- add a zero into right side of vector

                    when "100" => --Count UP
                            temp_out <= temp_out + 1;

                    when "110" => --ALL 0
                            temp_out <= temp_out;
                            monitor_all_0_1 <= monitor_all_0_1 + 1;

                    when "111" => --ALL 1
                            temp_out <= temp_out;
                            monitor_all_0_1 <= monitor_all_0_1 + 1;

                    when others => --refresh
                            temp_out <= temp_out;
            end case;

        END IF;

    END IF;

    END PROCESS;
```

```
-- detect when sequence is DONE

det_done: PROCESS(temp_out)

BEGIN
	case mode is

		when "010" => --Walk 1
			done <= temp_out(0);

		when "100" => --Count UP
			done <= '1';
			FOR i IN temp_out'range loop
			-- if any element is logic 0, counter is not done
				IF temp_out(i) = '0' THEN
					done <= '0';
				END IF;
			END LOOP;

		when "110" => --ALL 0
			done <= '1';
			FOR i IN monitor_all_0_1'range loop
			-- if any element is logic 0, counter is not done
				IF monitor_all_0_1(i) = '0' THEN
					done <= '0';
				END IF;
			END LOOP;

		when "111" => --ALL 1
			done <= '1';
			FOR i IN monitor_all_0_1'range loop
			-- if any element is logic 0, counter is not done
				IF monitor_all_0_1(i) = '0' THEN
					done <= '0';
				END IF;
			END LOOP;

		when others => --refresh
			done <= '1';
	end case;

END PROCESS;

-- determine the parity for the temp_out sequence
det_parity: PROCESS(temp_out)

variable temp_parity : std_logic;

BEGIN
	temp_parity := '0';
	FOR i IN temp_out'range loop
		temp_parity := temp_parity XOR temp_out(i);
	END LOOP;
	odd_parity <= temp_parity;
```

```
END PROCESS;

output <= temp_out; --connect output to port

end behavioral;
```

```
--------------------------------------------------------
-- A N bit Register
-- Author        : Jian Xu
-- Student ID    : 0392076
-- Date          : September12, 2001
-- File Name     : registerN.vhd
-- Architecture  : RTL
-- Description   : The width of the shift register is determined by generic N
--------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;

entity registerN is

   generic(N : positive := 28);

   port ( clock      : in std_logic;
          Load       : in std_logic;
          ShiftR     : in std_logic;
          shift_done : out std_logic;
          Output     : out std_logic_vector(N-1 downto 0));

end registerN;

-- structural implementation of the N-bit adder
architecture behavioural of registerN is

signal temp_out . std_logic_vector(N-1 downto 0);

BEGIN

PROCESS(clock, Load)

BEGIN
        IF (Load = '1' ) THEN                                   -- asy. loading
                temp_out <= (others => '1');
                temp_out(n-1) <= '0';       -- start sequence

        ELSIF ( clock'EVENT AND clock = '1' ) THEN      -- rising edge of clock
                IF (ShiftR = '1') THEN
                        temp_out <= '1' & temp_out((n-1) downto 1);        -- add a 1 as the left
                END IF;
        END IF;

END PROCESS;

shift_done <= not temp_out(0);
Output <= temp_out;

END behavioural;
```

```
--------------------------------------------------------
-- Control Path MODULE
-- Author          : Jian Xu
-- Student ID      : 0392076
-- Date            : September 11, 2001
-- File Name       : controller.vhd
-- Architecture    : Behavioral
-- Description     : Determines the control signals to pass to  the datapath componet of the BIST controller
--------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

ENTITY controller is

port (
        -- Black Box Signals
        Enable, Clk, Reset          : in std_logic;
        Done, Pass_Fail   : out std_logic;

        -- Internal Signals
        Shift_Done, TPG_Done, ORA_Equal                 : in std_logic;
        Shift_Load, Shift_Right, Advance_Sequence       : out std_logic;
        TPG1_Mode_Select                                : out std_logic_vector( 2 downto 0 );
        TPG2_Mode_Select                                : out std_logic_vector( 2 downto 0 ));

end controller;

ARCHITECTURE behavioral OF controller IS

-- Declare a new type for the states
 TYPE STATE_TYPE IS (           Waiting,
                                Reset_State,
                                First_Test,
                                Shift1,
                                Increment1,
                                Change_To_Test2,
                                Second_Test,
                                Shift2,
                                Increment2,
                                Change_To_Test3,
                                Third_Test,
                                Shift3,
                                Increment3,
                                Change_To_Test4,
                                Fourth_Test,
                                Shift4,
                                Increment4,
                                PASSED,
                                FAILED
                        );

   SIGNAL state: STATE_TYPE;
```

```
BEGIN

-- Create a process that handles the state transitions

PROCESS (Clk, Enable, Reset)

BEGIN

IF ( Reset = '0' AND Enable = '1' ) THEN                -- asy. reset, active low
        state <= Reset_State;

ELSIF ( Clk'EVENT AND Clk = '1' ) THEN

        IF ( Enable = '1' ) THEN

                CASE state IS

                        WHEN Waiting =>
                                state <= Waiting;

                        WHEN Reset_State =>
                                state <= First_Test;

                        WHEN First_Test =>
                                IF ( ORA_Equal = '0' ) THEN        -- state FAILED if column fails
                                        state <= FAILED;
                                ELSE
                                        state <= Shift1;
                                END IF;

                        WHEN Shift1 =>
                                IF ( ORA_Equal = '0' ) THEN        -- state FAILED if column fails
                                        state <= FAILED;
                                ELSIF ( Shift_Done = '0' ) THEN
                                        state <= Shift1;
                                ELSIF ( Shift_Done = '1' ) THEN
                                        state <= Increment1;
                                END IF;

                        WHEN Increment1 =>
                                IF ( TPG_Done = '0' ) THEN
                                        state <= Shift1;
                                ELSIF ( TPG_Done = '1' ) THEN   -- test done, next test
                                        state <= Change_To_Test2;
                                END IF;


                        WHEN Change_To_Test2 =>
                                state <= Second_Test;

                        WHEN Second_Test =>
                                IF ( ORA_Equal = '0' ) THEN        -- state FAILED if column fails
                                        state <= FAILED;
                                ELSE
```

```
                state <= Shift2;
        END IF;

WHEN Shift2 =>
        IF ( ORA_Equal = '0' ) THEN         -- state FAILED if column fails
                state <= FAILED;

        ELSIF ( Shift_Done = '0' ) THEN
                state <= Shift2;

        ELSIF ( Shift_Done = '1' ) THEN
                state <= Increment2;
        END IF;

WHEN Increment2 =>
        IF ( TPG_Done = '0' ) THEN
                state <= Shift2;
        ELSIF ( TPG_Done = '1' ) THEN   -- test done, next test
        state <= Change_To_Test3;
        END IF;


WHEN Change_To_Test3 =>
        state <= Third_Test;

WHEN Third_Test =>
        IF ( ORA_Equal = '0' ) THEN         -- state FAILED if column fails
                state <= FAILED;
        ELSE
                state <= Shift3;
        END IF;

WHEN Shift3 =>
        IF ( ORA_Equal = '0' ) THEN         -- state FAILED if column fails
                state <= FAILED;
        ELSIF ( Shift_Done = '0' ) THEN
                state <= Shift3;
        ELSIF ( Shift_Done = '1' ) THEN
                state <= Increment3;
        END IF;

WHEN Increment3 =>
        IF ( TPG_Done = '0' ) THEN
                state <= Shift3;
        ELSIF ( TPG_Done = '1' ) THEN   -- test done, next test
                state <= Change_To_Test4;
        END IF;

WHEN Change_To_Test4 =>
        state <= Fourth_Test;

WHEN Fourth_Test =>
        IF ( ORA_Equal = '0' ) THEN         -- state FAILED if column fails
                state <= FAILED;
```

```
                    ELSE
                            state <= Shift4;
                    END IF;

            WHEN Shift4 =>
                    IF ( ORA_Equal = '0' ) THEN          -- state FAILED if column fails
                            state <= FAILED;
                    ELSIF ( Shift_Done = '0' ) THEN
                            state <= Shift4;
                    ELSIF ( Shift_Done = '1' ) THEN
                            state <= Increment4;
                    END IF;

            WHEN Increment4 =>
                    IF ( TPG_Done = '0' ) THEN
                            state <= Shift4;
                    ELSIF ( TPG_Done = '1' ) THEN    -- all the tests are passed
                            state <= PASSED;
                    END IF;

            WHEN FAILED =>
                    state <= FAILED;          -- hold results until reset is pushed

            WHEN PASSED =>
                    state <= PASSED;          -- hold results until reset is pushed

        END CASE;

    ELSE --that is, IF ( Enable = '0')
        state <= Reset_State;
    END IF;

END IF;

END PROCESS;


-- Define the outputs based on what state the machine is in.

WITH state SELECT

-- use modes for TPG1

    TPG1_Mode_Select     <=     "100"        WHEN  Reset_State,
                                "100"        WHEN  First_Test,
                                "100"        WHEN  Increment1,
                                "100"        WHEN  Shift1,
                                "010"        WHEN  Change_To_Test2,
                                "010"        WHEN  Second_Test,
                                "010"        WHEN  Increment2,
                                "010"        WHEN  Shift2,
                                "111"        WHEN  Change_To_Test3,
                                "111"        WHEN  Third_Test,
                                "111"        WHEN  Increment3,
```

```
                                "111"        WHEN  Shift3,
                                "110"        WHEN  Change_To_Test4,
                                "110"        WHEN  Fourth_Test,
                                "110"        WHEN  Increment4,
                                "110"        WHEN  Shift4,

                                "000"        WHEN  others;

WITH state SELECT

-- use modes for TPG2

    TPG2_Mode_Select      <=    "100"        WHEN  Reset_State,
                                "100"        WHEN  First_Test,
                                "100"        WHEN  Increment1,
                                "100"        WHEN  Shift1,

                                "100"        WHEN  Change_To_Test2,
                                "100"        WHEN  Second_Test,
                                "100"        WHEN  Increment2,
                                "100"        WHEN  Shift2,

                                "100"        WHEN  Change_To_Test3,
                                "100"        WHEN  Third_Test,
                                "100"        WHEN  Increment3,
                                "100"        WHEN  Shift3,

                                "100"        WHEN  Change_To_Test4,
                                "100"        WHEN  Fourth_Test,
                                "100"        WHEN  Increment4,
                                "100"        WHEN  Shift4,

                                "000"        WHEN  others;


WITH state SELECT

-- reset shifter when reset or next vector

    Shift_Load      <=    '1'        WHEN  Reset_State,
                          '1'        WHEN  Increment1,
                          '1'        WHEN  Increment2,
                          '1'        WHEN  Increment3,
                          '0'        WHEN  others;


WITH state SELECT
    Shift_Right     <=    '1'        WHEN  Shift1,  -- walk-0 or walk-1
                          '1'        WHEN  Shift2,
                          '1'        WHEN  Shift3,
                          '0'        WHEN  others;
```

```
WITH state SELECT
        Advance_Sequence<=      '1'             WHEN   Increment1,
                                '1'             WHEN   Increment2,
                                '1'             WHEN   Increment3,
                                '0'             WHEN  others;


WITH state SELECT
        DONE            <=      '1'             WHEN   PASSED,  -- DONE when PASSED
                                '1'             WHEN  FAILED,           -- DONE when FAILED
                                '0'    WHEN    others;      -- NOT DONE any other time


WITH state SELECT
        PASS_FAIL       <=      '1'             WHEN   PASSED,  -- P/F = 1 when PASSED
                                '0'             WHEN  others;   -- P/F = 0 when FAILED or others

END behavioral;
```

```
--------------------------------------------------------
-- Package for Components for the BIST TC #1
-- Author         : Jian Xu
-- Student ID     : 0392076
-- Date           : September 11, 2001
-- File Name      : package_comp.vhd
-- Architecture   : N/A
-- Description    : Package definitions for control path, counter, shifter
--------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

package package_comp is

component controller is

        port (
                -- Black Box Signals
                Enable, Clk, Reset          : in std_logic;
                Done, Pass_Fail             : out std_logic;

                -- Internal Signals
                Shift_Done, TPG_Done, ORA_Equal          : in std_logic;
                Shift_Load, Shift_Right, Advance_Sequence : out std_logic;
                TPG1_Mode_Select                          : out std_logic_vector( 2 downto 0 );
                TPG2_Mode_Select                          : out std_logic_vector( 2 downto 0 ));

end component controller;

component registerN is

        generic(N : positive := 28);

        port (  clock      : in std_logic;
                Load       : in std_logic;
                ShiftR     : in std_logic;
                shift_done : out std_logic;
                Output     : out std_logic_vector(N-1 downto 0));

        end component registerN;

component gentpg

        generic (n: positive := 8);

        port (  clock      : in std_logic;
                reset      : in std_logic;
                advance          : in std_logic;
                mode             : in std_logic_vector(2 downto 0);
                output           : out std_logic_vector((n-1) downto 0);
                odd_parity       : out std_logic;
```

```
            done                : out std_logic);
        end component;

end package package_comp;
```

```
-------------------------------------------------------
-- The BIST for TC1
-- Author        : Jian Xu
-- Student ID    : 0392076
-- Date          : September 14, 2001
-- File Name     : wutbist.vhd
-- Architecture  : structural
-- Description   : Connects the control path, counter, shifter, and the MUX
-------------------------------------------------------package_comp

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.package_comp.all;

entity wutbist is

        generic (BIT_SIZE1: positive      := 8;
                 BIT_SIZE2: positive      := 3;
                 COLUMNS: positive        := 28);

    port (Enable          : in std_logic;

        Clk               : in std_logic;
        Reset             : in std_logic;
        Done              : out std_logic;
        Pass_Fail         : out std_logic;
        ORA_Equal         : in std_logic;
        Parity1           : out std_logic;
        Column_Select     : out std_logic_vector(COLUMNS-1 downto 0);
        Output            : out std_logic_vector(BIT_SIZE1+BIT_SIZE2-1 downto 0));

end entity wutbist;

architecture structural of wutbist is

        signal gnd                      : std_logic;
        signal sig_Shift_Done           : std_logic;
        signal sig_TPG_Done             : std_logic;
        signal sig_Shift_Load           : std_logic;
        signal sig_Shift_Right          : std_logic;
        signal sig_Advance              : std_logic;
        signal sig_TPG1_Mode_Select     : std_logic_vector( 2 downto 0 );
        signal sig_TPG2_Mode_Select     : std_logic_vector( 2 downto 0);
        signal sig_TPG1_Output          : std_logic_vector(BIT_SIZE1-1 downto 0);
        signal sig_TPG2_Output          : std_logic_vector(BIT_SIZE2-1 downto 0);

BEGIN

        gnd <= '0';

        BIST_CONTROLLER : component controller
```

```
port map (
            --EXTERNAL SIGNALS
            Enable          =>          Enable,
            Clk             =>          Clk,
            Reset           =>          Reset,
            Done            =>          Done,
            Pass_Fail       =>          Pass_Fail,

            --INTERNAL SIGNALS
            Shift_Done      =>      sig_Shift_Done,
            TPG_Done        =>      sig_TPG_Done,
            ORA_Equal       =>      ORA_Equal,
            Shift_Load      =>      sig_Shift_Load,
            Shift_Right     =>      sig_Shift_Right,
            Advance_Sequence        =>      sig_Advance,
            TPG1_Mode_Select        =>      sig_TPG1_Mode_Select,
            TPG2_Mode_Select        =>      sig_TPG2_Mode_Select
      );

BIST_TPG1 : component gentpg

      generic map ( n => BIT_SIZE1 )

      port map (
                  clock           =>      Clk,
                  reset           =>      gnd,
                  advance         =>      sig_Advance,
                  mode            =>      sig_TPG1_Mode_Select,
                  output          =>      sig_TPG1_Output,
                  odd_parity      =>      Parity1,
                  done            =>      sig_TPG_Done
            );

BIST_TPG2 : component gentpg

      generic map ( n => BIT_SIZE2 )

      port map (
                  clock           =>      Clk,
                  reset           =>      gnd,
                  advance         =>      sig_Advance,
                  mode            =>      sig_TPG2_Mode_Select,
                  output          =>      sig_TPG2_Output
            );


BIST_SHIFTER : component registerN

      generic map ( N => COLUMNS )

      port map (  clock           =>      Clk,
                  Load            =>      sig_Shift_Load,
                  ShiftR          =>      sig_Shift_Right,
                  shift_done      =>      sig_Shift_Done,
```

```
                                Output          =>      Column_Select );
Output(BIT_SIZE1+BIT_SIZE2-1 downto BIT_SIZE2) <= sig_TPG1_Output;
Output(BIT_SIZE2-1 downto 0) <= sig_TPG2_Output;
end architecture structural;
```