BUILT-IN SELF-TEST CONFIGURATIONS FOR FIELD PROGRAMMABLE GATE ARRAY CORES IN SYSTEMS-ON-CHIP

Except where reference is made to the work of others, the work described in this thesis is my own or was done in collaboration with my advisory committee. This thesis does not include proprietary or classified information.

Jonathan McKinley Harris

Certificate of Approval:

Victor P. Nelson
Professor
Electrical and Computer
Engineering

Charles E. Stroud, Chair
Professor
Electrical and Computer
Engineering

Adit D. Singh
Professor
Electrical And Computer
Engineering

Stephen L. McFarland
Acting Dean
Graduate School

BUILT-IN SELF-TEST CONFIGURATIONS FOR FIELD PROGRAMMABLE GATE ARRAY CORES IN SYSTEMS-ON-CHIP

Jonathan McKinley Harris

A Thesis
Submitted to
the Graduate Faculty of
Auburn University
in Partial Fulfillment of the
Requirements for the
Degree of
Master of Science

Auburn, Alabama
December 17, 2004

BUILT-IN SELF-TEST CONFIGURATIONS FOR FIELD PROGRAMMABLE GATE ARRAY CORES IN SYSTEMS-ON-CHIP

Jonathan McKinley Harris

Permission is granted to Auburn University to make copies of this thesis at its discretion, upon request of individuals or institutions and at their expense. The author reserves all publication rights.

Signature of Author

Date

Copy sent to:

Name Date

VITA

Jonathan McKinley Harris, son of Ralph and Cathy Harris, was born June 25, 1979, in Winston-Salem, NC. He graduated from Forbush High School with high honors in 1997. He graduated cum laude with a Bachelor of Science degree in Electrical Engineering at the University of North Carolina at Charlotte in May 2003. After completion of his undergraduate degree, he entered the graduate program in Electrical Engineering at Auburn University in August 2003. While in pursuit of his Master of Science degree at Auburn University, he worked under the advisement of Dr. Charles Stroud as a graduate research assistant in the Electrical and Computer Engineering department. On June 12, 2004, he married Melinda L. Beaver, daughter of Bobby and Wanda Beaver.

THESIS ABSTRACT

BUILT-IN SELF-TEST CONFIGURATIONS FOR FIELD PROGRAMMABLE GATE ARRAY CORES IN SYSTEMS-ON-CHIP

Jonathan McKinley Harris
Master of Science, December 17, 2004
(B.S.E.E, University of North Carolina at Charlotte, 2003)

125 Typed Pages

Directed by Charles E. Stroud

Built-In Self-Test configurations for the logic and routing resources present in the Field Programmable Gate Array core of a System-on-Chip is presented in this Thesis. These configurations completely test the Programmable Logic Blocks and Programmable Routing Resources present in the Field Programmable Gate Array Core. A vendor-specific CAD tool, Atmel System Designer software suite, is used in conjunction with custom design automation tools to generate a complete set of logic and routing BIST configurations for any size Atmel AT94K series FPGA core as well as any size Atmel AT40K series FPGA.

ACKNOWLEDGEMENTS

The author would like to thank Dr. Charles Stroud for assistance with the development and application of the BIST configurations completed in the work for this thesis. The author would also like to thank master's committee members Dr. Victor Nelson and Dr. Adit Singh for their advice and contributions to this thesis. Many thanks are due to family members Ralph, Cathy, Gray, and Kristy for their continued support during the course of the completion of the work for this thesis. A special thanks is due to my wife, Melinda, who has given so much of herself and her time during the completion of this thesis. Above all, thanks to God for the strength and persistence to complete this thesis and the work involved therein.

IEEE (Institute of Electrical and Electronic Engineers) Format

Microsoft Word

TABLE OF CONTENTS

LIST OF TABLES AND FIGURES

LIST OF ACRONYMS

ASIC - Application Specific Integrated Circuit

AVR - Advanced Virtual RISC

BIST - Built-In Self-Test

BUT - Block Under Test

CAD - Computer Automated Design

CLB - Configurable Logic Block

CPU - Central Processing Unit

CUT - Circuit Under Test

DCT - Discrete Cosine Transform

DSP - Digital Signal Processing

FIR - Finite Impulse Response

FFT - Fast Fourier Transform

FPGA - Field Programmable Gate Array

FPSLIC - Field Programmable System Level Integrated Circuit

HDL - Hardware Description Language

IC - Integrated Circuit

IDS - Integrated Development System

I/O - Input/Output

IP - Intellectual Property

LUT - Look Up Table

MGL - Macro Generation Language

MUX - Multiplexer

NCD - NeoCAD Design Language

ORA - Output Response Analyzer

PAR - Place and Route

PCB - Printed Circuit Board

PIP - Programmable Interconnect Point

PLB - Programmable Logic Block

RISC - Reduced Instruction Set Computer

SoC - System-on-Chip

SRAM - Static Random Access Memory

STAR - Self Test ARea

TPG - Test Pattern Generator

VLSI - Very Large Scale Integration

WUT - Wire Under Test

XDL - Xilinx Design Language

CHAPTER ONE

INTRODUCTION

Meeting time to market demands and profitability requirements of digital electronics systems is increasingly important in the current economy [1]. In the past, systems have been designed at the board level, meaning a system may comprise many integrated circuits (ICs) interconnected on a printed circuit board (PCB). Currently, IC process technologies are allowing transistor sizes to pass under the 100 nanometer threshold, thus making the designs much more dense and allowing on the order of a hundred million transistors to be fabricated on a single IC [2], [3]. With the availability to comprise an entire system on a single IC, an increase in the overall device speed and reduction of device power can be achieved. This integration of a system down to the IC level is referred to as System-on-Chip (SoC). In addition to the aforementioned advantages, an SoC implementation significantly reduces the cost of the system. Therefore, SoCs have been gaining popularity in the electronic design industry due to the attractiveness of attaining a complete system level design in an IC.

Primarily, there are two types of SoC design implementations; Application Specific Integrated Circuit (ASIC) based SoCs and generic SoCs containing user-programmable logic. A typical ASIC-based SoC would incorporate standard cell and regular structure based components and, therefore, be limited to one particular application, having no programmability to make it adaptable. In contrast, a generic SoC contains user-programmable logic allowing it to be easily reprogrammed to adapt to such changes as updated industry standards. ASIC-based SoCs have primarily dominated the drive into SoC design implementations; however, generic SoCs containing Field Programmable Gate Arrays (FPGAs) as the user-programmable logic have been increasing in popularity [2]. Generic SoC designs have many advantages over the ASIC-based SoC designs. These advantages include [4]:

- The SoC does not suffer from expensive redesigns and long time to market.
- The embedded FPGA may be used for different functions at different times.
- Due to the FPGA's massive parallelism, many algorithms (such as image or signal processing and cryptography) can be implemented.
- The same SoC can be used for multiple applications.
- The FPGA can be used to implement protocols and algorithms likely to change.
- The FPGA can support remote and internet based field upgrades.

These generic SoC designs are very attractive to designers as their reconfigurability provides low design cost, shorter time to market, and possibly increased testability [2], [1]. As a result of the advantages mentioned above, the incorporation of an FPGA is increasingly becoming the standard technique in the design of an SoC [2]. This thesis will focus on the testing of FPGA core logic and routing resources within a specific generic SoC; however, the overall approach is applicable to most SoCs with user-programmable logic.

1.1 Systems on Chip (SoCs)

An example structure of an SoC includes user programmable or standard cell logic, memory (for data and program), Central Processing Units (CPUs), and possibly some analog circuitry [5]. SoCs are usually designed around Intellectual Property (IP) cores that generally fall into two categories, Hard IP cores and Soft IP cores. Hard IP cores are supplied as a predesigned physical layout of a particular circuit, whereas Soft IP cores are available in synthesizable modules described in a Hardware Description Language (HDL), such as VHDL or Verilog [5]. These Soft IP cores would be synthesized into standard cell logic or into an FPGA core and, having performed the physical layout based on the logic implementation, be incorporated into the SoC. Any combination of Hard and/or Soft IP cores may be used in the design of an SoC. An example structure comprised in an SoC is shown in Figure 1.1. This figure shows the components mentioned previously and illustrates their interconnections.

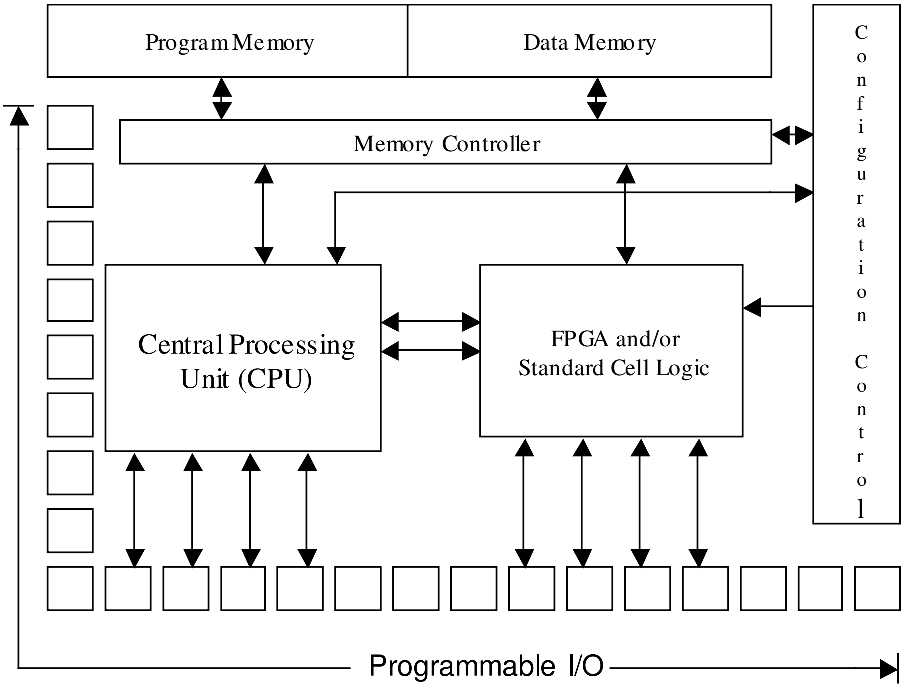


Figure 1.1 Example System on Chip (SoC) Architecture

The CPU allows complex algorithms to be implemented in software, which is almost a necessity in modern day digital systems. The CPU can either have a predefined set of operating conditions, or may be user-configurable to allow for more flexibility [5]. The user programmable logic, shown in Figure 1.1 as an FPGA, allows various hardware functions to be implemented using the programmability of the FPGA, to be discussed in the next section. Program and data memories exist to store the algorithms for execution by the CPU and to provide a place to store operands and data to be manipulated, respectively [5]. Also shown in the figure is a memory controller interface, which serves to control the interaction of the CPU and FPGA with the program and data memories. In addition to the memory controller, the configuration control can allow the FPGA to be programmed via the CPU or directly from the user.

1.2 Field Programmable Gate Arrays (FPGAs)

The FPGA portion of the SoC typically consists of four components, configurable logic blocks (CLBs) also known as programmable logic blocks (PLBs), programmable interconnect, programmable Input/Output (I/O) cells, and a configuration memory [6]. An example FPGA structure is shown in Figure 1.2. The PLBs serve as the user programmable logic and allows for the implementation of various digital logic functions. The PLBs are typically arranged in an *NxN* array and are connected via the programmable interconnect network to allow multiple logic functions to be performed. The programmable I/O cells can function as inputs or outputs, depending on the requirements of the design. The programming bits necessary to configure the CPU and/or the FPGA for the desired system function are stored in the configuration memory.

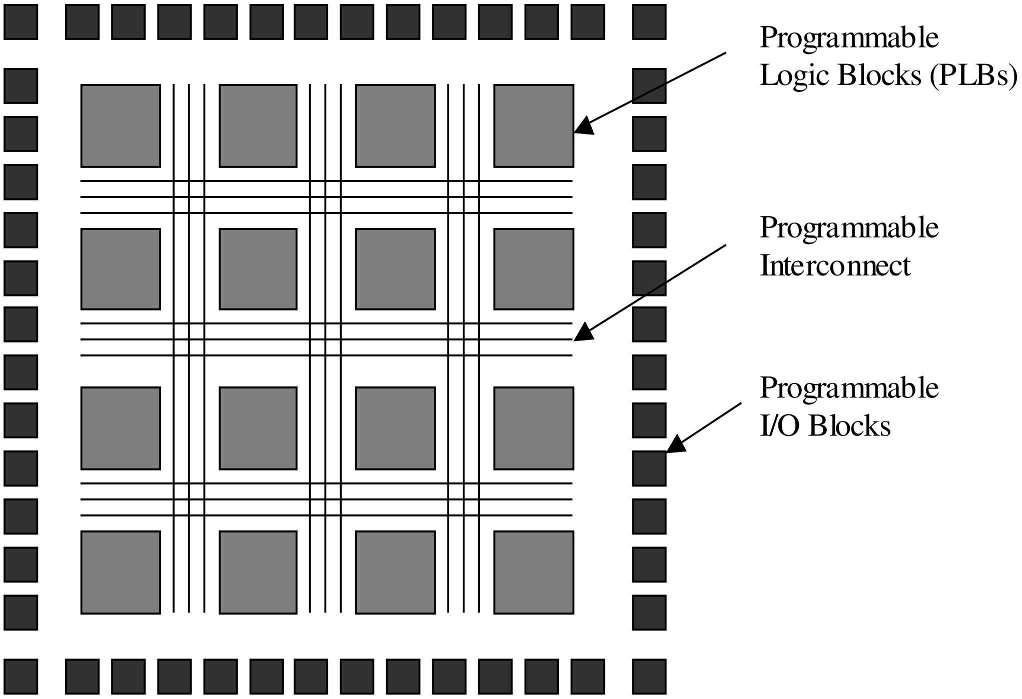


Figure 1.2 Example FPGA Block Diagram

Typically, a PLB will consist of two to eight *k*-input Look Up Tables (LUTs), one flip-flop per LUT, and various multiplexers and logic gates [7]. Each LUT can realize any combinational logic function of *k* inputs and each PLB can simultaneously execute *m* different *k*-input logic functions where *m* is the number of LUTs in the PLB. The flip-flop(s) give the user the flexibility of performing sequential logic functions as well as combinational logic functions. The multiplexers serve to set up the data paths in the PLB. These make the paths through which the appropriate input signals propagate through the PLB in order to implement the desired digital function.

The programmable interconnect consists of various wire segments controlled by programmable interconnect points (PIPs) to form connections between two or more PLBs and/or between the PLBs and the programmable I/O cells [6]. The routing resources used to make connections between adjacent PLBs are usually referred to as *local routing* and those resources used to make connections between non-adjacent PLBs are typically referred to as *global routing*. The PIPs control the connections of the various wire segments to make or break these local or global connections between PLBs [6]. Data bits stored in the configuration memory are used to control the PIPs. The PIPs act similar to switches where the data bits from the configuration memory are used to turn the PIP on and off, thus making or breaking connections between the associated wire segments.

The configuration memory stores all the configuration bits for the PLBs, I/O cells, and the programmable interconnect network and is loaded via a configuration interface. The configuration memory is typically comprised of Static Random Access Memory (SRAM) cells that contain configuration bits for each of the LUTs, multiplexers, flip-flop reset/set, and various PIPs in the FPGA [6].

Due to the volatility of the SRAM-based configuration memory, the configuration bits must be loaded each time the device is powered on and, therefore, must be stored in an external memory device or loaded from a computer [6]. However, the advantage of using an SRAM-based memory is that it allows in-system reprogramming to change the system function whenever desired.

1.3 The Testing Problem

Testing combinational and sequential logic components is an intricate process and involves many interacting aspects [8]. The primary issues in testing include the cost of the test development, the quality of the generated test and the cost and time of applying the test [8]. Other factors influencing testing are the increasing number of transistors present in an IC, which can currently number in the millions. Over the past two to three decades, the number of I/O pins on most very large scale integration (VLSI) devices has increased by an order of magnitude while the number of transistors contained in those VLSI devices has increased by about four orders of magnitude [7]. This has inherently reduced the accessibility and, therefore, the testability of the circuits within these VLSI devices [7]. In order to develop and evaluate the quality of a test for a device, a fault model is used to emulate the various types of faults that can be encountered in VLSI devices.

Typically, fault models that model gate, transistor, and other physical faults and defects are used to perform fault simulations to determine the ability of a given set of test patterns to detect faults within a circuit [7]. Testing is performed by using test patterns designed to test for the specific type of fault model and applying these patterns to the circuit under test (CUT). Once these test patterns have been applied to the CUT, the output responses are compared against responses of a fault-free circuit obtained from simulation. If all the output responses match, the CUT is assumed to be fault-free, however, in case of a mismatch, the CUT is determined to be faulty and is typically discarded.

1.4 Built-In Self-Test (BIST) Techniques

Built-In Self-Test (BIST) is a method of testing a given circuit wherein additional circuitry is added such that the circuit can test itself. BIST can be broken into two categories, on-line BIST and off-line BIST. When performing on-line BIST, the CUT is operating in its normal system mode of operation. During off-line BIST the CUT is placed in a test mode that is typically not a normal system mode of operation. The advantage of off-line BIST is that it can be applied at the manufacturing, field, depot, and operational levels [7], [8]. The primary focus of this thesis will be off-line BIST.

The particular form of off-line BIST that will be applied for the research involved in this thesis will be a form of off-line BIST where no additional circuitry outside the chip itself is required for testing, but existing flip-flops and registers within the CUT are manipulated and used for testing [7]. A typical BIST structure comprises a test controller, test pattern generator (TPG), circuit under test (CUT), and an output response analyzer (ORA) as illustrated in Figure 1.3 [9].

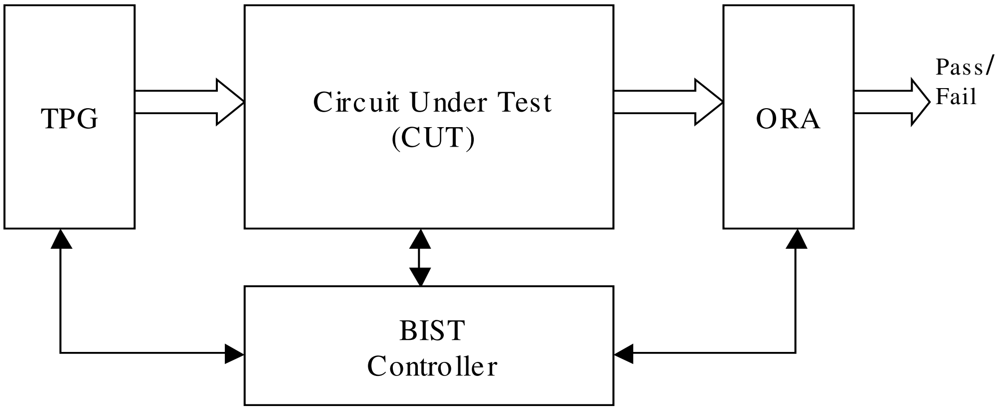


Figure 1.3 Typical BIST Structure

The TPG supplies test patterns to the CUT and the ORA analyzes the results to indicate a pass or fail condition for the CUT. Typically, the TPG generates exhaustive or pseudo-exhaustive test patterns and may be as simple as a binary *n*-bit counter or an *n*-bit linear feedback shift register (LFSR) with a primitive polynomial or as complex as an algorithmic test patter generator [7]. The ORA can be as simple as a comparator that compares known good responses with the actual responses of the CUT or can be as complex as a multiple input data compactor that may use signature analysis to indicate a fault occurrence [7]. If the TPG and ORA are separated from the CUT, they can be used to test multiple CUTs concurrently. Using this approach imposes no performance penalties on the CUT other than additional set-up time when controlling the CUT to operate in its BIST mode of operation [7].

1.5 BIST for FPGAs

Typical BIST approaches for FPGAs involve taking the FPGA off-line, testing the device, and, if found fault-free, placing the FPGA back on-line within the system. This is possible since most current FPGAs use an SRAM-based configuration memory, which is inherently in-system reprogrammable. As a result, the FPGA is reconfigured to create a BIST structure, the device is tested, and is then reprogrammed for the system function before being returned to the system operation [7]. Due to the reconfigurability and multiple modes of operation and combinations of interconnection of an FPGA, many test configurations will be required to completely test an FPGA's programmable logic and routing resources. The basic idea in BIST for the programmable logic of an FPGA is to configure rows (or columns) of PLBs as TPGs and ORAs, and other rows (or columns) of PLBs as blocks under test (BUTs), as illustrated in Figure 1.4. [7]

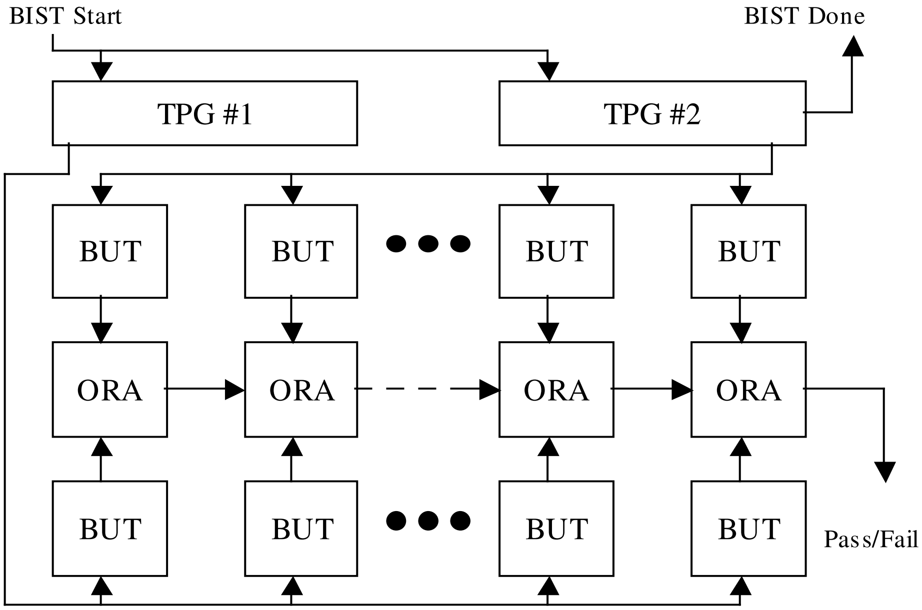
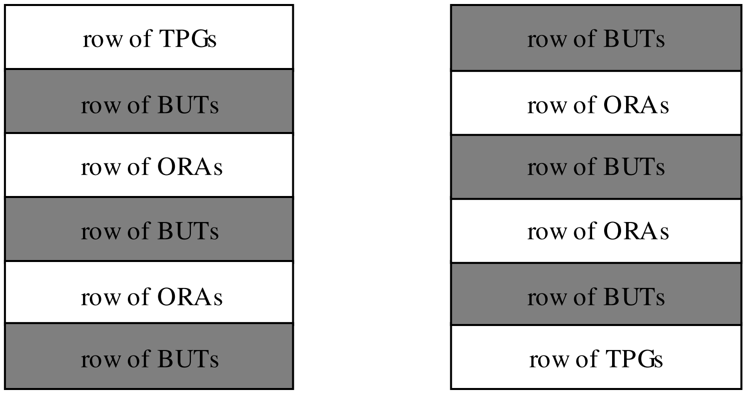


Figure 1.4 Basic Logic BIST Architecture for FPGAs [7]

The BUTs are reconfigured several times such that they are tested in all their modes of operation. Each time the BUT is reconfigured to test a different PLB mode, it is referred to as a *test phase* [7]. A collection of test phases that completely test the BUTs in all their modes of operation is referred to as a *test session* [7]. Once the BUTs have been completely tested in all their modes of operation, the test session is repeated such that the PLBs that were previously TPGs and ORAs become BUTs, and vice versa as illustrated for a 6×*N* array of PLBs in Figure 1.5 [7]. As a result, the programmable logic resources in the FPGA can be tested in a minimum of two test sessions.



Test Session #1 Test Session #2

Figure 1.5 Test Sessions for logic BIST for FPGAs

Each test phase consists of the following steps: 1) the FPGA is reconfigured with a BIST configuration, 2) the BIST sequence is executed, which involves initialization, test pattern generation, and output response compaction, and 3) the BIST results are read from the ORAs [7]. In step 1, the test controller interacts with the FPGA to reconfigure the FPGA for testing. This is done when the controller retrieves the current BIST configuration from a storage device, such as a computer, and downloads this information to the configuration memory of the FPGA. The controller also initializes the TPGs, ORAs, and BUTs and provides a BIST Start signal to initiate the test in step 2. In step 3, the Pass/Fail results from the test phase are read from the ORAs in the FPGA to indicate the condition of the device. For this type of BIST, comparator-based ORAs are used because of the nature of the BIST architecture. Since the BUTs are receiving indentical inputs from two TPGs, their outputs should match and a fault condition can be determined simply by seeing a match or mismatch between the respective BUT outputs [7].

One approach in BIST for the programmable interconnect of an FPGA is to configure a subset of the routing resources (wire segments and PIPs) to form two groups of wires under test (WUTs). The WUTs receive identical test patterns from a TPG and the values are compared at the other end of the WUTs by one or more comparison-based ORAs [7]. A set of WUTs may be composed of several wire segments connected by closed PIPs and may also include PLBs to check local routing to/from the PLBs. An example of this type of BIST architecture for programmable interconnect is shown in Figure 1.6 [7].

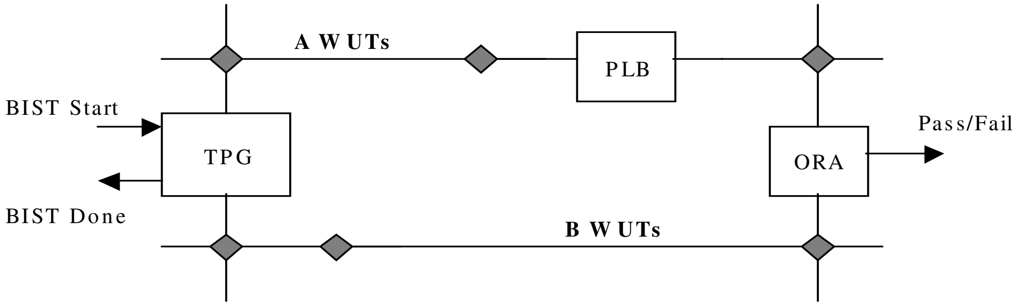


Figure 1.6 Example Interconnect BIST Architecture for FPGAs [7]

1.6 Thesis Statement

The off-line BIST approach to testing FPGAs offers the advantage of no additional circuitry, thus no area overhead in the design of the chip as well as no performance penalty in the system mode of operation [7]. Theoretically, the same BIST approach can be implemented in the FPGA core of a generic SoC. Once the BIST has been performed on the FPGA and the FPGA determined fault-free, the FPGA core can then be used to test the other cores in the SoC [4].

When applied to FPGA core logic and routing resources in an SoC, the benefits of an off-line BIST approach can be fully utilized. Since the FPGA can be reprogrammed, it can be removed from system operation, reprogrammed for BIST, tested for faults, and if no failures, reprogrammed for system operation. If faults are detected, the system function can be reconfigured to avoid the faults present in the FPGA. A disadvantage of testing an FPGA core with this approach is the requirement of multiple test configurations. Since the PLBs and routing resources must be configured in many different modes of operation, complete testing requires multiple configurations of the device [7]. Despite the disadvantage of multiple test configurations, the use of BIST for testing FPGA core logic provides the most benefits for complete testing of any test methods to date.

The research to be presented in this thesis is based upon a previously proposed SoC BIST methodology [4] and previous work completed in BIST for FPGA logic and routing resources, in [10] and [11] respectively. The work to be presented is a continuation of these two previous BIST approaches in relation to testing FPGA core logic and routing resources that are a part of a generic SoC. The BIST approach is developed for and applied to a commercially available SoC, specifically the Atmel AT94K series Field Programmable System Level Integrated Circuit (FPSLIC).

The remaining chapters of this thesis are structured in the following manner: Chapter 2 presents a detailed overview of the Atmel AT94K FPSLIC FPGA core architecture as well as a review of previous work completed in BIST for other FPGAs. Chapter 3 presents the BIST approach used in the testing of the FPGA core logic resources and the results obtained with the Atmel AT94K Series SoC. Chapter 4 presents the BIST approach used for the routing resources in the Atmel AT94K Series FPSLIC FPGA core and the obtained results. Chapters 3 and 4 also give an overview of the testing obstacles encountered with the Atmel FPSLIC and the methods used to overcome these obstacles and how they can be applied to testing other SoCs. Chapter 5 provides a summary of the work completed as well as suggestions for future work in testing FPGA core logic and routing resources in SoCs.

CHAPTER TWO

BACKGROUND INFORMATION

Presented within this chapter is an overview of the AT94K series Field Programmable System Level Integrated Circuit (FPSLIC) architecture (a generic SoC) and the FPGA architecture incorporated in the Atmel AT40K series FPGAs and the FPGA core of the FPSLIC. A review of prior work completed in the testing of FPGAs as well as work in BIST for FPGAs is presented. In addition, a comparison of the FPGA architectures in the prior BIST work to that of the Atmel FPGA is also presented. The development process of the previous applications of BIST for FPGAs will be discussed and background material for the proposed development process of BIST for application to the Atmel FPGA will be presented.

2.1 Atmel AT94K Series FPSLIC

The Atmel AT94K series FPSLIC incorporates an FPGA core utilized in the AT40K series FPGAs along with an AVR (Advanced Virtual RISC) 8-bit Reduced Instruction Set Computer (RISC) microcontroller unit, a configuration controller, a memory controller, and program and data memories [14]. The combination of the AVR and configuration controller allows for in-system reprogramming of the desired operation of the device by downloading a configuration in to the FPGA core or by utilizing the AVR to dynamically reprogram the FPGA core of the device [14]. Figure 2.1 illustrates the interconnections in the FPSLIC device between the FPGA and the AVR with the memory and configuration controller components.

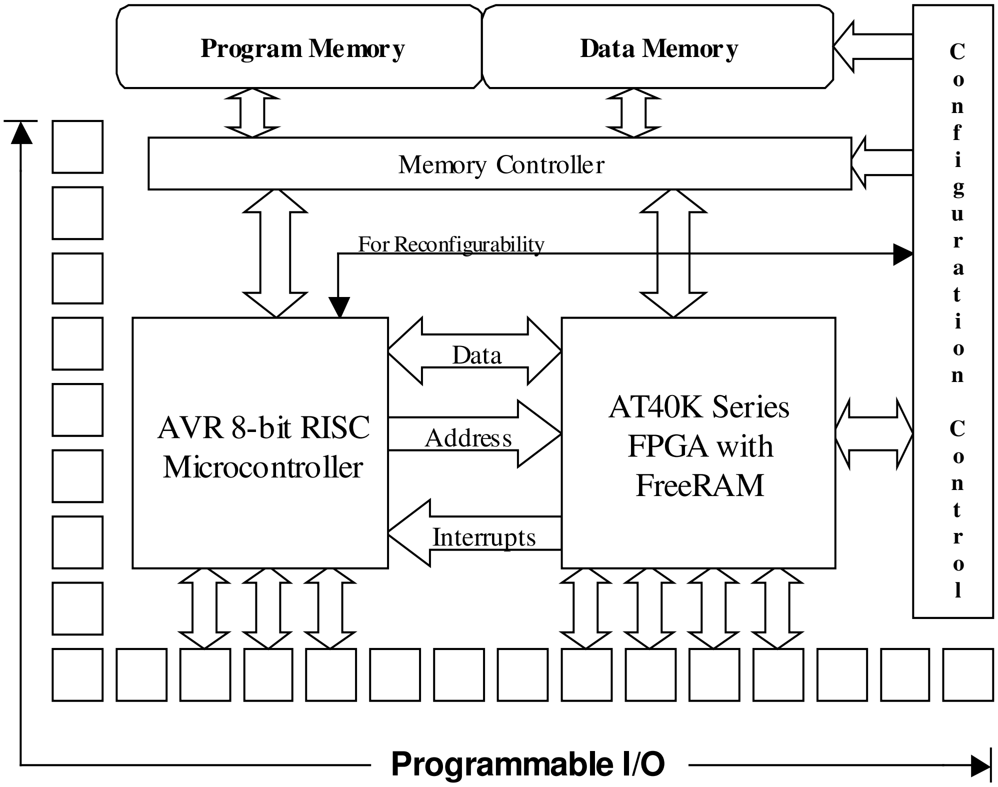


Figure 2.1 Atmel AT94K Series FPSLIC Architecture [14]

The AVR 8-bit RISC Microcontroller supports over 120 instructions, of which most execute within a single clock cycle [14]. The microcontroller architecture is optimized for C code, but may be programmed in assembly language as well [14]. For storage, up to 36 Kbytes of memory can be partitioned for up to 16 Kbytes x 16-bit program memory and up to 16 Kbytes x 8-bit data memory [14]. Between the AVR and the FPGA there are 16 address lines (8 lines for the AT94K05) decoded from 4 bits in the AVR memory map and an 8-bit bi-directional data bus to allow for accessibility from the AVR core of the device to the FPGA core [14]. In addition, 16 internal interrupt lines are supplied from the FPGA to the AVR and up to four external interrupts are available through user I/O [14]. The FPGA core of the FPSLIC can be one of four sizes of the AT40K series FPGA, ranging in usable gate count from 5,000 to 40,000 gates [14], [15]. These four sizes are summarized in Table 2.1, which details the AT40K FPGA and the associated AT94K FPGA core sizes in terms of the number of PLBs. The primary focus of the FPSLIC architecture will be on the FPGA portion of the device since this portion of the device is the focal point of the research and development work described in this thesis.

Table 2.1 Atmel FPGA and FPGA Core Sizes

FPGA	FPSLIC	Number of PLBs NxN
AT40K05	AT94K05	16x16
AT40K10	AT94K10	24x24
AT40K20	Not Currently Available	32x32
AT40K40	AT94K40	48x48

2.2 Atmel AT40K Series FPGA

The AT40K FPGA is designed for rapid implementation of high-performance, large gate count designs through the use of synthesis- and schematic-based tools either on a PC or Sun platform [15]. Designs can be implemented in the AT40K series FPGA through common design software such as Synplicity, ModelSim, Exemplar, and Viewlogic since the Atmel design tools are devised to integrate with these and other industry standard design software [15]. The FPGA can be used to implement arithmetic-intensive functions, which include applications for high-speed Digital Signal Processing (DSP) functions. Examples of such DSP functions include Finite Input Response (FIR) filters, Fast Fourier Transforms (FFT), convolvers, interpolators, and Discrete Cosine Transforms (DCT) [15].

The basic AT94K series FPGA core architecture is illustrated in Figure 2.2. The device consists of an array of PLBs, two planes of programmable interconnect (vertical and horizontal) and programmable I/O pads. The PLBs are arranged in an *N*×*N* array where *N* is given in Table 2.1.

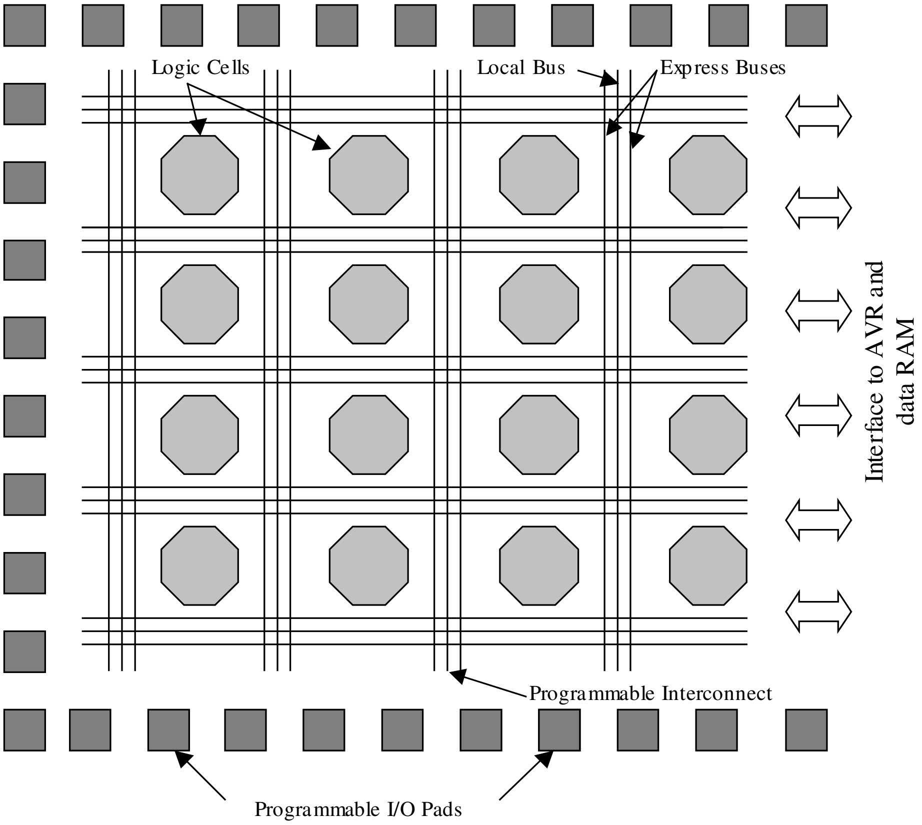


Figure 2.2 Atmel AT94K Series FPGA Core Architecture [15]

2.2.1 PLB Architecture

The logic within the PLB is shown in Figure 2.3 and, as can be seen, various combinations of functions can be implemented within the PLB. Configuration bits stored in the configuration memory of the device determine the logic function performed by the PLB. Any *n*-input logic function, where $1 \le n \le 4$, can be realized within the PLB utilizing the two 3-input LUTs simultaneously. If $n \le 3$, up to two logic functions can be obtained using one LUT for each function, since each LUT contains eight bits [15]. The logic function(s) implemented can be either sequential or combinational, by using or not using the D flip-flop, respectively [15]. A logic function with feedback may also be realized within the PLB and can be either sequential or combinational [15]. The AND gate present on the **W** input is important for logic functions implementing arrays of multipliers [15]. The multiplexers shown in gray produce a default logic value of '1' when no input is selected; otherwise, they behave similar to non-decoded multiplexers where a control bit is associated with each input to the multiplexer [7]. The multiplexer with CB (configuration bit) shown as an input has a configuration bit from the configuration memory driving one of its inputs. The remaining multiplexers shown behave as decoded multiplexers with a single configuration bit selecting one of the mulitplexer's two inputs. In addition to the various types of logic functions available, the PLB output **L** can be optionally tri-stated for bi-directional bus implementations when the PLB output needs to be in a high impedance state [15]. The **X**, **W**, **Y**, and **Z** inputs are selected from multiplexers that are a part of the local routing resources, which are discussed in more detail in the next section. In addition to **X**, **W**, **Y**, and **Z** inputs there is a clock and a set/reset input to the flip-flop and a horizontal and vertical output enable to the tri-state buffer of each PLB for a total of eight inputs. The outputs of the PLB include the **X** and **Y** outputs to adjacent PLBs and the **L** output which connects to the global routing resources for a total of three outputs.

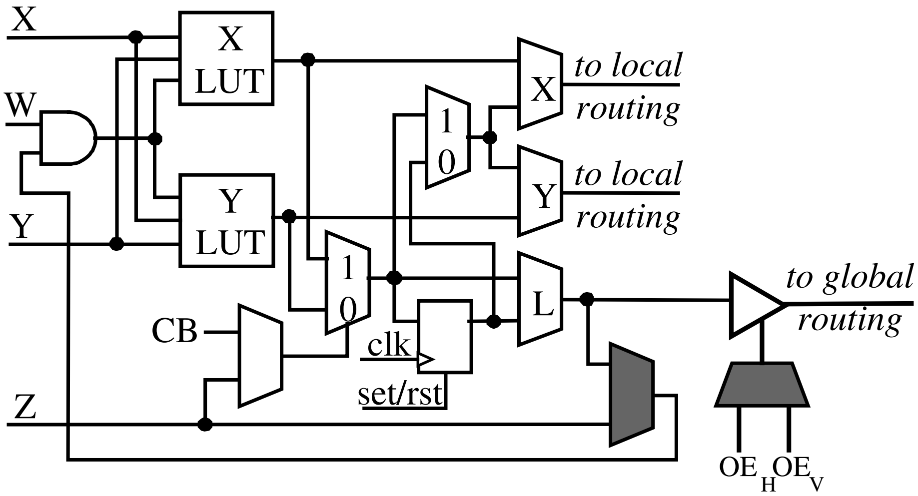
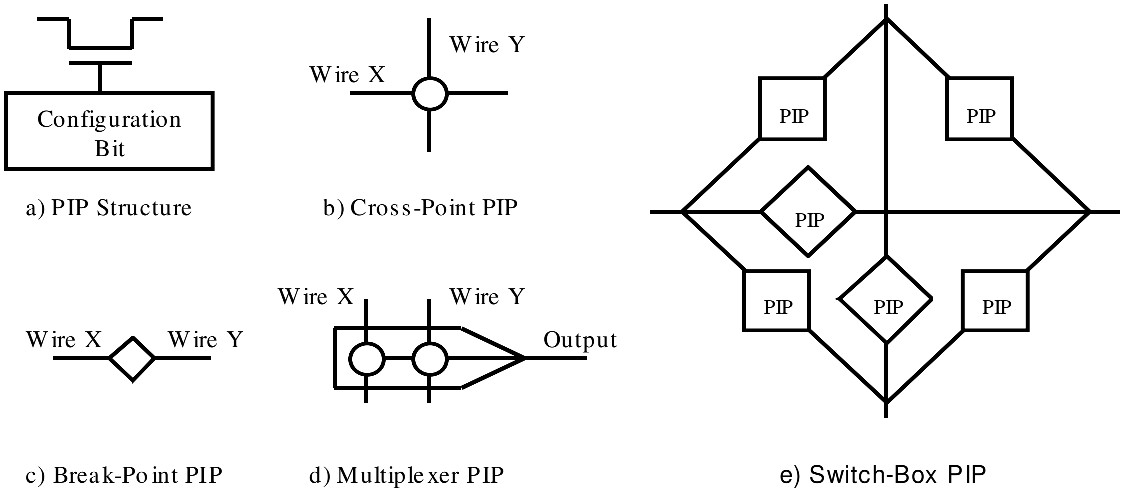


Figure 2.3 Atmel AT40K Series PLB [15]

2.2.2 Programmable Interconnect Points (PIPs) in FPGAs

In order to understand the routing architecture present in an FPGA, it is important to understand the types of PIPs and wire segments commonly found in FPGAs. The basic structure of a PIP is shown in Figure 2.4a, where a pass transistor (transmission gate) is turned on/off by the logic value of a configuration memory bit. A connection can be made or broken by setting the configuration bit to the desired logic value.



a) PIP Structure
b) Cross-Point PIP
c) Break-Point PIP
d) Multiplexer PIP
e) Switch-Box PIP

Figure 2.4 PIP Structures Typically Found in FPGAs [7]

The general types of PIPs found in FPGAs fall into four categories, cross-point PIPs, break-point PIPs, multiplexer PIPs, and switch-box PIPs [7]. The cross-point PIPs (Figure 2.4b) connect wire segments perpendicular to one another; for example, a vertical wire segment and horizontal wire segment are connected. A break-point PIP (Figure 2.4c) connects or disconnects wire segments within the same plane, such as a vertical-vertical connection or a horizontal-horizontal connection [7]. The multiplexer PIP (Figure 2.4d) can either be decoded or non-decoded and selects one of multiple input wire segments to make a connection to an output wire segment [7]. Non-decoded multiplexer PIPs are controlled by k configuration bits, where k is the number of input wire segments to the PIP [7], such that there exists one bit per wire segment and that bit controls the connection for its respective wire segment. The decoded multiplexer PIP has 2^k input wire segments and is controlled by k configuration bits [7]. For the decoded multiplexer, the binary code formed by the configuration bits controls the connection between a given input and the output wire segments as in a basic multiplexer. The last type of PIP, the switch-box PIP (Figure 2.4e), also referred to as a compound cross-point PIP, is usually made of an array of pass transistors making connections in various directions between different wire segments [7].

Typically, the types of PIPs and wire segments found in an FPGA can be separated into global and local routing resources. The multiplexer PIPs are typically found in the local routing resources while the cross-point, break-point, and switch-box PIPs are usually found in the global routing resources. The local routing resources are those associated with a given PLB and its adjacent PLBs. The PLB inputs and outputs enter and exit through the local routing resources and can make connections either to adjacent PLBs or to other PLBs or I/O cells through the global routing resources. Global routing resources allow connections to be made between PLBs that are typically at distances of more than one PLB apart in the array. These resources can also be used to route long distances from PLBs to I/O cells. The global routing resources commonly comprise longer wire segments and their associated PIPs. The wire segments present in the global routing resources span differing lengths of the FPGA array and connections can be made between these wire segments by the various types of PIPs that are present within the global routing network.

The wire segments in the programmable interconnect in the FPGA have various lengths which are typically associated with the number of PLBs that a given wire segment spans. These include short to medium length wire segments denoted as *x1* lines, *x2* lines, *x4* lines, and *x8* lines which span one, two, four, and eight PLBs, respectively. Longer wire segments may span a quarter (*xQ* lines), one half (*xH* lines), or the full length (*xL* lines) of the FPGA array.

2.2.3 Routing Architecture

The routing architecture present in the Atmel FPGA core consists of both local and global routing resources. The local routing resources are formed by wire segments and PIPs that make connections to adjacent PLBs and allow access to/from the global routing resources. The global routing resources comprise busses of wire segments that span either four PLBs (*x4* lines, which are referred to as local busses in Atmel terminology) or span eight PLBs (*x8* lines, referred to as express busses in Atmel terminology) in the array before reaching repeaters, which will be discussed shortly.

The local routing resources consist of non-decoded multiplexer PIPs, which select inputs from one of the five *x4* lines or from direct PLB connections from adjacent PLBs, as illustrated in Figure 2.5. The five inputs to each multiplexer PIP, denoted as V/H 1 through V/H 5 in Figure 2.5, come from the five vertical and five horizontal *x4* lines in the global routing resources. The **W** and **Z** inputs to PLB can come from any one of the five vertical *x4* lines or of the five horizontal *x4* lines and are selected by their respective multiplexer PIPs. In the case of the **X** and **Y** inputs, additional multiplexer PIPs select a signal either from the inputs from the *x4* lines or from one of the direct connections from an adjacent PLB.

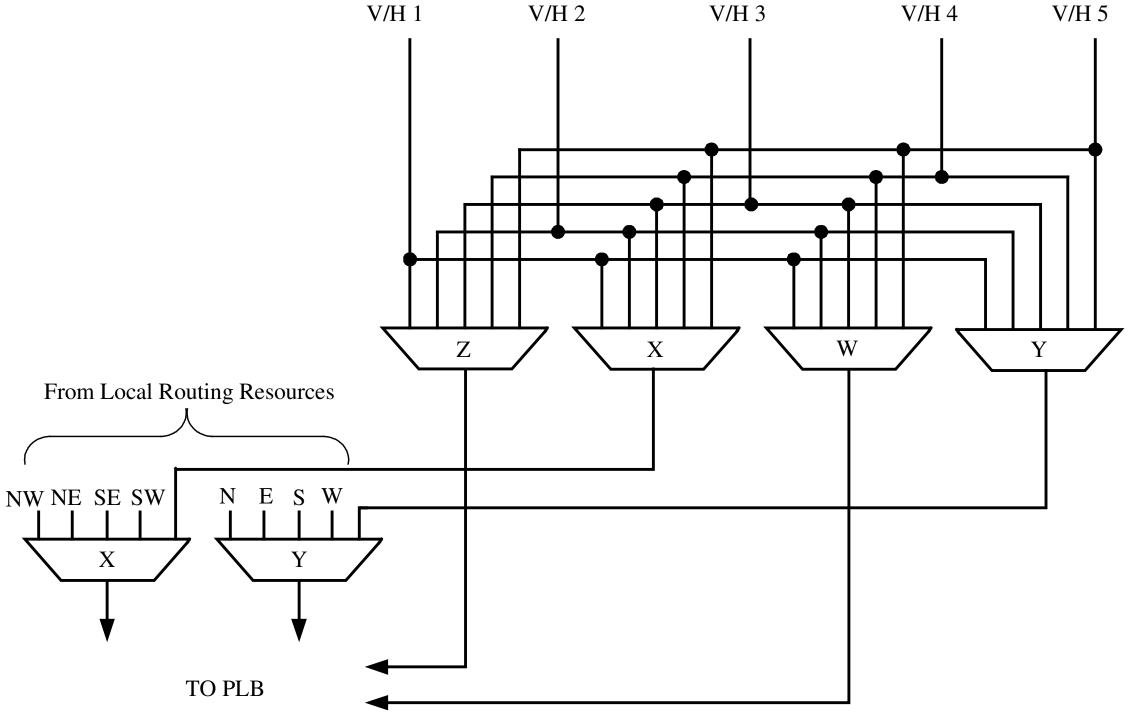


Figure 2.5 Local Interconnect Associated with the Atmel PLB [15]

The available direct PLB connections are illustrated in Figure 2.6, which shows the eight possible direct PLB connections. Only one input from any given direction can be selected at one time. The available connections denoted as **X** in Figure 2.6 include both the **X** input to the PLB and the **X** output from the PLB and make it possible to connect to a diagonally adjacent PLB [15]. It is possible to connect to adjacent PLBs located northwest, northeast, southeast, and southwest (denoted NW, NE, SE, and SW, respectively, in Figures 2.5 and 2.6) using local **X**. The connections denoted as **Y** in Figure 2.6 include both the **Y** input to the PLB and the **Y** output from the PLB and can make connections to orthogonally adjacent PLBs to the north, east, south, and west (denoted as N, E, S, and W, respectively, in Figures 2.5 and 2.6).

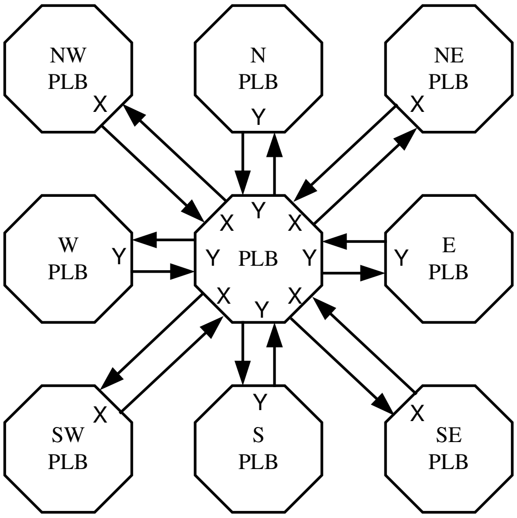
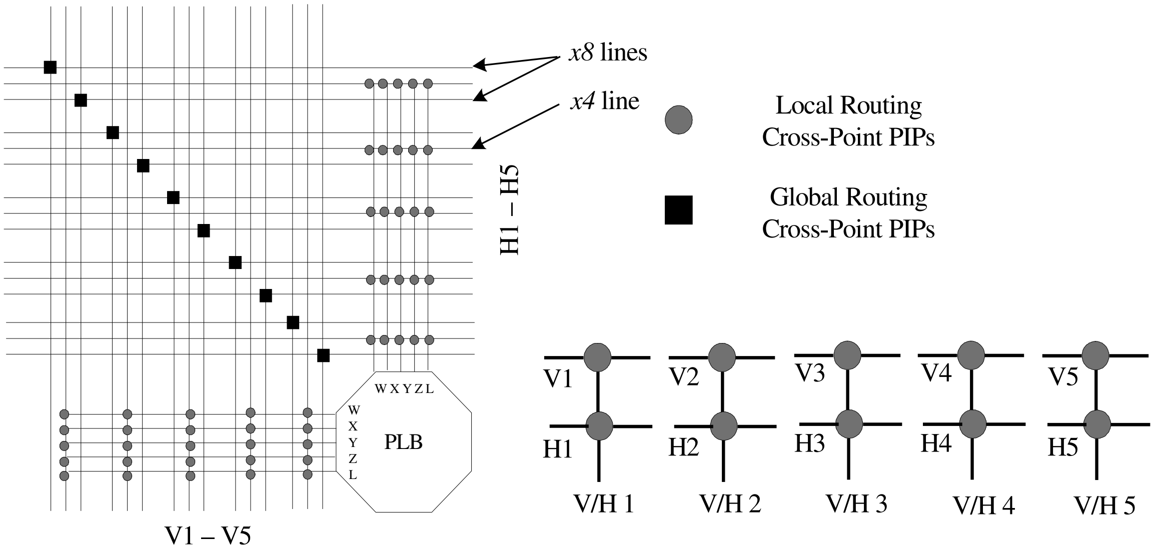


Figure 2.6 PLB-to-PLB Adjacent Connections [15]

Each PLB has connections as shown in Figure 2.6 to and from the adjacent PLBs both orthogonally and diagonally except for those PLBs that are along the edges of the PLB array. The PLBs along the edges have direct connections to and from I/O cells in place of the respective direct PLB connections.

The PLB connections to the global routing resources are shown in Figure 2.7 and are denoted as **W**, **X**, **Y**, **Z**, and **L** [15]. Here, **W**, **X**, **Y**, and **Z** serve as inputs to the PLB and **L** is a PLB output. The PLB inputs can connect to one, and only one, of the five vertical *x4* lines or of the five horizontal *x4* lines at a time [15]. The same is also true for the **L** output of the PLB. It is important to note that the **X** and **Y** inputs can enter the PLB through one and only one of the connections from either one of the *x4* lines or one of the adjacent PLB connections such that both connections cannot be made simultaneously. Therefore, if a connection is made to **X** or **Y** from a *x4* line, no connection can be made from the direct inputs connected to an adjacent PLB [15].

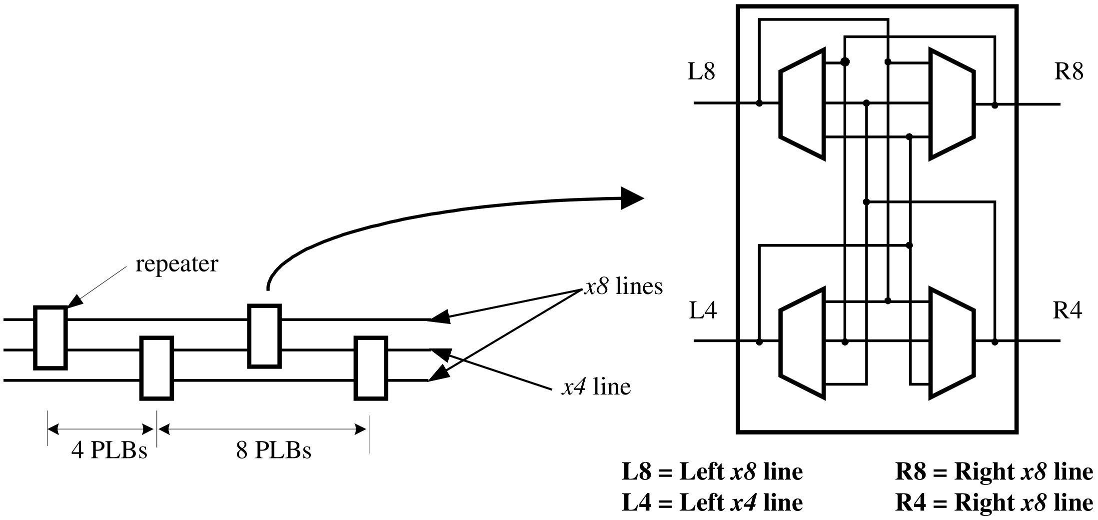


a) PLB-to-Bus Connections b) Local Routing Cross-Point PIPs

Figure 2.7 PLB-to-Bus Connections [15]

A connection is made to the *x4* lines in the five vertical or horizontal busses through the cross-point PIPs shown in Figure 2.7b, denoted as V1 - V5 and Hl - H5. When the cross-point PIP is turned on, the *x4* line is available to the PLB and multiplexer PIPs, shown in Figure 2.5, present on the **W**, **X**, **Y**, and **Z** inputs as well as the **L** output select the bus as an input or output, respectively. Cross-point PIPs in the global routing resources, illustrated in Figure 2.7a diagonally from the PLB, can be used to make connections between the horizontal *x8* and vertical *x8* lines. Connections between horizontal and vertical *x4* lines are made through the cross-point PIPs shown in Figure 2.7b.

The *x8* lines span eight PLBs before reaching a repeater and the *x4* lines span four PLBs before coming to a repeater [15]. The *x8* lines that are located furthest from the PLB shall be referred to as Abus lines while the *x8* lines closest to the PLB shall be referred to as Ebus lines. Repeaters are staggered throughout the array, meaning that every other span of eight cells the repeaters alternate which *x8* and *x4* lines can be connected as is illustrated in Figure 2.8a. These repeaters allow selected types of connections to be made between wire segments. For simplicity only one of the five busses of *x4* lines and *x8* lines is shown. The internal organization of the repeater is illustrated in Figure 2.8b and consists of four 3-input non-decoded multiplexer PIPs.



a) Staggered Repeaters b) Repeater Connections

Figure 2.8 Staggered Repeaters in Atmel Routing Architecture

Connections can be made in multiple directions through the repeater. A connection can be made from one *x4* line through the repeater to the adjacent *x4* line or from one *x4* line to either of the *x8* lines (where L4 and R4 denote left and right *x4* lines and L8 and R8 denote left and right *x8* lines as shown in Figure 2.8b). The repeaters are staggered such that the orientation of the repeater is flipped about the horizontal axis from one repeater to the next such that the *x4* lines and *x8* lines alternate entry/exit points of the repeater. As can be seen in the figure, four multiplexers within the repeater allow for the various combinations of connections to be made. The high-level functionality of the repeater is similar to that of the switch-box PIPs, however, the actual internal organization utilizes multiplexer PIPs instead of pass transistors to complete connections between the different wire segments. This provides buffering for long or heavily loaded signals and also implies directionality of the signal.

2.3 Overview of Testing Methods for FPGAs

Many previous methods for testing FPGAs rely on externally applied test vectors, hence these various testing approaches are limited to device-level testing only [16]-[18]. In addition, there also exist many restrictions in controllability and observability due to the number of I/O cells present in a particular package that can be used for testing the FPGA. This problem is further complicated in generic SoCs since a large portion of the FPGA core I/O is internally connected to other SoC components. As a result, even more controllability and observability restriction exists in the FPGA core of a generic SoC.

In BIST-based approaches, such as in [9]-[10], [19], [20], the testing configurations do not rely on externally applied test vectors, but instead use the internal circuitry of the FPGA to generate test patterns for the purpose of testing the FPGA. Multiple test configurations may be required, but the advantage is the ability to use the same test procedure from device-level testing through system-level testing, since BIST is applicable to all these levels [10]. The BIST approach results in less total test development time since BIST can be applied from device-level testing through system-level testing without the need for developing different tests for different levels of testing.

Therefore, there is less final cost since test development is applicable to the various levels of testing of electronic devices, which, in turn, is a significant portion of device cost [21].

2.4 Previous Work in BIST for FPGAs

Previous implementations of BIST for FPGAs have been applied to devices such as the Lucent Technologies Optimized Reconfigurable Cell Array (ORCA) 2C and 2CA FPGAs [9], [20] and to the Xilinx 4000 and Spartan series FPGAs [10]. Typically BIST is separated into logic and routing BIST, which test the PLBs and programmable interconnect present in an FPGA, respectively. In these previous works, there are specific architectural issues that have a large impact on the implementation of BIST in the respective device [10].

2.4.1 Logic BIST

In the previous implementations of BIST for FPGAs the basic idea for logic BIST is illustrated in Figure 2.9, where some columns (or rows) of the PLBs are configured as TPGs and ORAs and other columns (or rows) of PLBs are configured as BUTs. This architecture is flipped from the first test session to the second test session. The flipping of the architecture ensures that all PLBs become BUTs during testing as long as at least half the PLBs are BUTs during each test session.

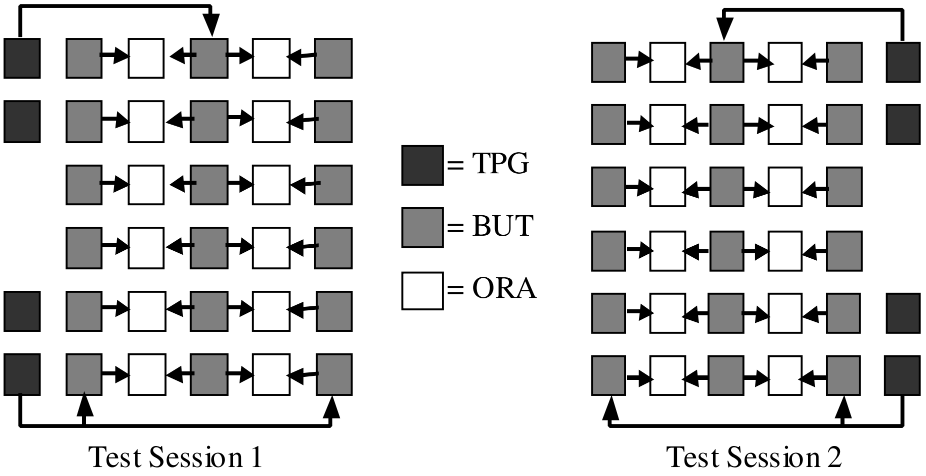


Figure 2.9 Logic BIST Architecture

Typically a binary counter is utilized as the TPG for the logic BIST test sessions since it generates exhaustive test patterns and can be used to apply all possible 2^n test patterns for an n-input logic function [7]. This type of TPG is the most economical choice since a counter mode is common in the PLBs of most FPGAs [7]. An LFSR (Linear Feedback Shift Register) could also be used; however, additional logic is required to obtain an all 0's state, which makes the TPG more difficult to implement, when compared to a binary counter [7]. Algorithmic TPGs are used to test PLBs that have RAM modes of operation such as the ORCA and Xilinx FPGAs [10]. These algorithmic TPGs are also more difficult to implement and require more logic, hence more PLBs, than a binary counter.

There are two important features to note pertaining to this logic BIST architecture. First, multiple TPGs are used in order to source identical test patterns to alternating rows or columns of identically configured BUTs [7]. Second, every BUT, except for the first and last two columns, has its corresponding outputs observed by two different ORAs and compared with different BUTs [7]. The combination of these two factors guarantees that any single faulty PLB can be guaranteed to be detected [7]. In addition, the conditions that allow multiple faulty PLBs to avoid detection are so limiting that, in practice, the chances of occurrence are very unlikely [7].

Multiple test phases are applied during each test session in order to completely test the PLBs configured as BUTs in all of their modes of operation [9], [11], [20]. The number of configurations depends on the number of modes of operation available in the PLB. Hence, the more available modes of operation that can be implemented in a PLB, the greater the number of configurations that are required to test the PLB. It is desired to minimize the number of configurations required to test an FPGA in order to minimize testing time and test development effort and time.

Since identical test patterns are applied to identically configured BUTs, the ORAs employed are comparison-based in order to compare the responses of the BUTs located in the adjacent rows or columns [7]. The basic architecture of the comparison-based ORA previously used is illustrated in Figure 2.10 where four sets of outputs from adjacent BUTs are compared [10], [19]. The feedback, OR gate, and flip-flop in the PLBs configured as ORAs latch any mismatch detected in the corresponding BUT outputs so that the faulty indication result is stored [7]. To retrieve the results, the ORAs can be connected as a shift register to shift out the results at the end of each BIST sequence or the configuration memory can be read to obtain the contents of the ORA flip-flops at the end of a BIST sequence [7].

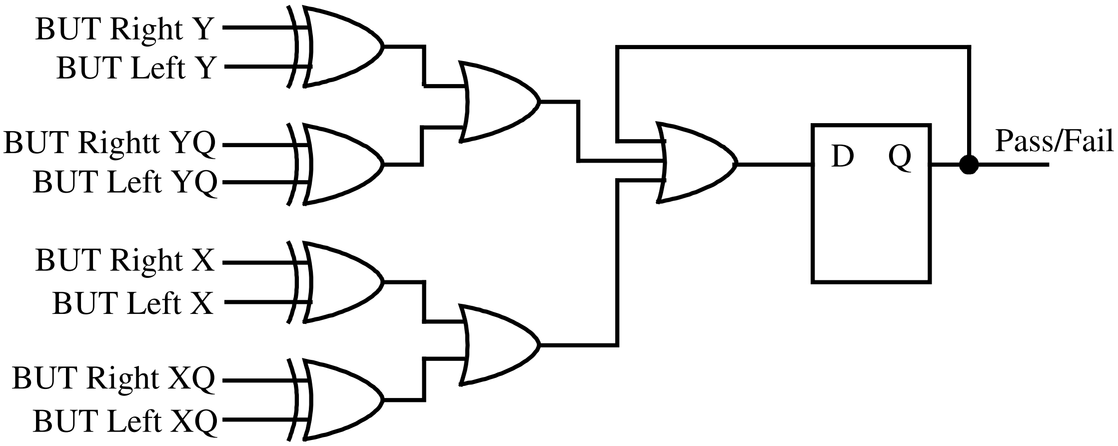


Figure 2.10 Basic ORA Structure in Logic BIST [10], [19]

2.4.2 Routing BIST

One methodology applied in the routing BIST in previous implementations consists of a counter-based TPG that sources test patterns over two sets of WUTs that have their signals compared at a destination by a comparison-based ORA, as illustrated in Figure 2.11. Groups of PLBs are configured as the TPGs in order to source the test patterns over the sets of WUTs [22]. These WUTs are configured from a subset of the routing resources of the FPGA and include selected wire segments and PIPs that are targeted for testing [22]. The ORAs receive the test patterns sourced over the WUTs by the TPGs and detect mismatches between the sets of WUTs to give a Pass/Fail indication at the end of the BIST sequence to determine if the WUTs are faulty or fault-free [22]. The ORA results can be retrieved in a similar fashion as in the logic BIST: the ORAs can be connected as a shift register and the results shifted out at the end of the BIST sequence or the results can be read directly from the configuration memory [7].

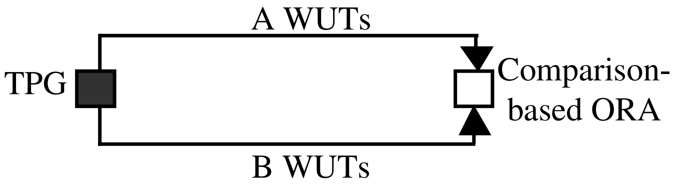


Figure 2.11 Routing BIST Architecture

Another approach to testing interconnect is demonstrated in Figure 2.12 where the counter-based TPG also generates parity over the binary count sent over a set of WUTs and, over some other routing resources, the parity bit is routed to the destination parity check-based ORA which also receives the signals from the WUTs [23]. In this routing BIST architecture, which is similar to that shown in Figure 2.11, some PLBs are configured as TPGs and some PLBs are configured as ORAs with selected wire segments and PIPs selected for testing configured as WUTs. However, in this case, the TPG generates a parity bit associated with its test pattern and the ORA is configured accordingly to check the parity bit associated with the output response of the WUTs, as illustrated in Figure 2.12. The parity bit is sent from the TPG to the ORA over known good routing resources [23].

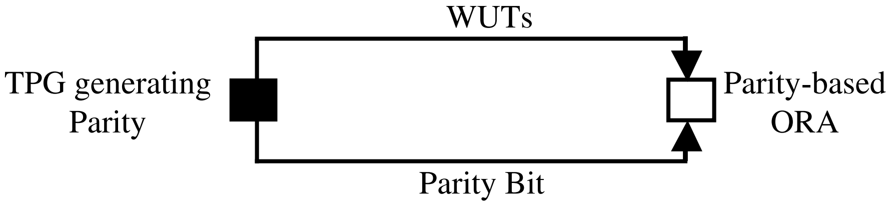


Figure 2.12 Parity check-based Routing BIST

In both the comparison- and parity check-based routing BIST approaches, Self Test AReas (STARs) have been used to implement the BIST configurations within the routing resources of the FPGA [10]. These STARs consist of selected subsets of the programmable logic and routing resources within the FPGA configured as TPGs, ORAs, and WUTs [28]. These STARs are scalable to the architecture of the device to which the approach is applied and can be as large as the entire array. The number of configurations depends on the size of the STARs and the number of routing resources to be tested within the STAR during a given test session. The size of the STAR during a given test session impacts both the speed of the test and the diagnostic resolution of the test [10]. A large STAR or a STAR with a high number of routing resources has larger resistive and capacitive loading. This is due to the finite on resistance and gate capacitance of the transmission gates in the various PIPs in the set of WUTs. These factors cause longer delays in signal propagation, which affects the speed of the test. In addition, a large STAR makes fault diagnosis more difficult since a larger number of routing resources is under test in a given test phase and determining which of the wire segments is faulty is made much more difficult.

2.4.3 BIST for the ORCA FPGAs

The logic BIST approach implemented in [9], [11], [20] for the ORCA 2C and 2CA FPGAs uses the logic BIST architecture illustrated in Figure 2.9. It is important to note the impact of the PLB and interconnect architectures on the number of configurations and testability of the FPGA. A total of 9 and 14 phases per test session were required to completely test the PLBs in the 2C and 2CA FPGAs, respectively [9], [20]. The ORCA 2CA FPGA has more modes of operation than the ORCA 2C FPGA, which include multiplier and comparator modes as well dual-port RAM modes of operation. This increased number of modes translates to more BIST configurations required to test the PLB in all its modes of operation. The PLB in the ORCA 2C and

2CA FPGAs device consists of four 16-bit LUTs, four flip-flops, a fast-carry circuit, and several multiplexers [25]. The inputs to the flip-flops can be fed in through four primary inputs to the PLB or can be driven by the four LUTs [25]. There are five primary outputs of the PLB, which can be driven either from the flip-flops or the LUTs to allow for combinational or sequential functions to be performed [25]. The LUTs can be programmed in one of three modes: logic (LUT), arithmetic (fast-adders/subtractors), and memory (RAM) [25]. The two RAM modes allow the LUTs in the PLB to function as either a 16x4 RAM or as a two 16x2 RAMs. The four flip-flops present in the PLB can be configured to act as level sensitive latches or to act as edge-triggered flip-flops [25].

The routing resources of the ORCA FPGAs required 27 and 44 BIST configurations, respectively, for the two BIST approaches described in [11] and [22]. Both approaches used the routing BIST architecture shown in Figure 2.11. The approach in [22] required more configurations due to the use smaller STARs in the application of routing BIST which were chosen to increase diagnostic resolution. The routing resources associated with each PLB in the ORCA FPGA include six horizontal and six vertical 4-bit global routing busses and four sets of direct 5-bit local routing busses to adjacent PLBs [22]. Along the horizontal and vertical direction, there are two 4-bit busses of *x1* lines , two 4-bit busses of *x4* lines, one 4-bit bus of *xH* lines, and one 4-bit bus of *xL* lines associated with each PLB.

2.4.4 BIST for the Xilinx 4000 and Spartan Series FPGAs

The logic BIST illustrated in Figure 2.9 was also applied to the Xilinx 4000 and Spartan series FPGAs [10]. For the case of these Xilinx FPGAs, a total of 12 test phases per test session were needed to completely test the PLBs. The PLB consists of two 4-input LUTs, one 3-input LUT, two flip-flops, dedicated carry logic, and various multiplexers to configure the cell interconnections and functions [26]. The two 4-input LUTs can function together as a single 32x1 RAM or as two 16x1 RAMs [26]. The two flip-flops can function as edge-triggered flip-flops or as level-sensitive latches [26]. The additional logic associated with the PLBs is the carry logic circuitry used to implement fast adders, subtractors, and counters [26].

The routing resources within the Xilinx 4000 and Spartan series FPGAs consist of a specific number of wire segments dependent upon the particular type of device, either 4000E/Spartan or a 4000XL/XLA series FPGAs. Both types of devices include an 8-bit bus of *x1* lines and a 4-bit bus of *x2* lines. The 4000E/Spartan FPGAs include ten vertical and six horizontal long lines (including *xQ*, *xH*, and *xL* lines) while the 4000XL/XLA FPGAs have 18 vertical and six horizontal long lines. The 4000XL/XLA FPGAs also include three 4-bit busses of *x4* lines and 2 direct connections to each adjacent PLB. The comparison-based routing BIST approach in Figure 2.11 was applied to all routing resources in the 4000E/Spartan and 4000XL/XLA series FPGAs [10]. A total of 128 BIST configurations were required to test all routing resources in the 4000E/Spartan series FPGAs while a total of 206 BIST configurations were required to test all of the routing resources in the 4000XL/XLA series FPGAs [10].

The parity-based routing BIST configuration was proposed in [23] to test the *x1* lines and their associated switch-box PIPs in the 4000 series FPGAs. It was theorized that these resources could be tested in three BIST configurations. However, this BIST approach was never implemented in the actual FPGA to verify that three BIST configurations were sufficient. In addition, the *x1* lines and their associated switch-box PIPs the easiest to test routing resources [10].

2.4.5 Previous Work In Logic BIST for the Atmel AT94K Series FPGA Core

The only previous work for logic BIST for the FPGA core in the AT94K series FPSLIC was proposed in [27]. This work involved on-line testing of the LUTs in the PLBs where two test configurations were required per LUT in each PLB [27]. In this approach, only the LUTs in the PLBs are tested and the remaining logic is left untested, as are the routing resources within the FPGA. In the work reported in [27], the total number of configurations to test the NxN PLB array in the FPGA is $2N^2$ since the PLBs are tested one at a time. In addition, it appears that the AVR core would perform the TPG and ORA functions.

2.5 ORCA, Xilinx, and Atmel FPGA Comparison

The composition of the PLBs and the complexity of the routing resources are considerably different in the Atmel AT40K series FPGAs than in the ORCA and Xilinx FPGAs. The PLBs in the ORCA and Xilinx FPGAs consist of more logic and hence more programmability and functionality than the Atmel FPGA, as is summarized in Table 2.2. The ORCA and Xilinx PLBs have approximately twice the number of primary inputs as the Atmel PLB and also have more outputs. In addition, the LUTs in the ORCA and Xilinx PLBs have more bits than contained in the LUTs in the Atmel PLB. There are also more configuration multiplexers and additional logic gates (including carry logic) in the ORCA and Xilinx PLBs than in the Atmel.

Table 2.2 Comparison of PLBs [14], [15], [25], [26]

PLB Component	Atmel AT40K	ORCA 2C and 2CA	Xilinx 4000 and Spartan
# Inputs	8	19	13
# Outputs	3	6	4
# LUTs	2	4	3
# Bits/LUT	8x1	16x1	(2)16x1, (1)8x1
Config. Multiplexers (incl. Carry Logic)	11	17	26
# Flip-Flops	1	4	2
# Addt'l Logic Gates (incl. Carry Logic)	1	3	6

The routing resources dispersed within the ORCA and Xilinx FPGAs are more numerous and comprise more types of wire segments and PIPs than the Atmel FPGA, as summarized in Table 2.3. The ORCA and Xilinx FPGAs contain about twice as many total types of routing resources as the Atmel FPGA. The diagonal direct lines are unique to the Atmel FPGA and are not found in either the ORCA or Xilinx FPGAs. The Atmel and ORCA routing resources are symmetrically aligned horizontally and vertically meaning that the number and interconnections of vertical and horizontal planes of wire segments is the same. However, in the Xilinx FPGA, the routing resources do not have rotational symmetry. This asymmetry contributes to the test complexity for the Xilinx since there are different numbers of horizontal and vertical bussing planes depending on the type of routing resource [10]. In addition, some of the busses in the Xilinx routing resources are shared between PLBs and create obstacles in the development of logic and routing BIST configurations [10]. There also exists dedicated carry routing in the ORCA and Xilinx FPGAs that is not present in the Atmel FPGA. Carry routing can be implemented with the direct routing (orthogonal or diagonal direct lines) in the Atmel device, but at the expense of a PLB output since there is no dedicated logic or routing for carry circuitry within the Atmel FPGA. The conclusion drawn in [10] is that the routing architecture of the FPGA is the primary component in the number of BIST configurations required to completely test the device. This was demonstrated by the total number of test configurations required for the Xilinx routing resources (128 and 206) compared to that required for the ORCA routing resources (27 and 44) [10].

Table 2.3 Comparison of Routing Resources [10], [15]

Routing Resource Type	Atmel AT40K		ORCA 2C ORCA 2CA		Xilinx 4000E and Spartan		Xilinx 4000XL and XLA	
	vertical	horizontal	vertical	horizontal	vertical	horizontal	vertical	horizontal
x1 lines	0	0	8	8	8	8	8	8
x2 lines	0	0	0	0	4	4	4	4
x4 lines	5	5	8	8	0	0	12	12
x8 lines	10	10	0	0	0	0	0	0
long lines	0	0	8	8	10	6	18	6
direct lines	1	1	5	5	0	0	2	2
diagonal direct lines	1		0		0		0	
carry lines	0		0		2		2	
Total	17	17	29	29	24	20	46	34

Another important point to consider in the comparison of the ORCA, Xilinx, and Atmel FPGAs is the percent composition of the types of PIPs present in the routing resources of the devices, as illustrated in Table 2.4. The Atmel FPGA is more similar in certain aspects to the Xilinx than the ORCA FPGAs. In terms of break-point PIPs and multiplexer (MUX) PIPs, the Atmel FPGA is more similar in composition to the Xilinx FPGA, however, as in the ORCA FPGA, there are no true switch-box PIPs in the Atmel FPGA. The closest routing resource that the Atmel FPGA has to the switch-box PIPs in the Xilinx FPGA are the repeaters and they are more similar to multiplexer PIPs than to switch-box PIPs as illustrated in Figure 2.8b.

Table 2.4 Percent Composition of PIPs in Routing Resources [10], [15]

Type of PIP	Xilinx 4000	ORCA 2C/2CA	Atmel AT40K
Break-Point	1.5%	12%	N/A
Cross-Point	11.3%	69%	34.8%
MUX	72.5%	19%	60.9%
Switch-Box	14.7%	N/A	N/A
Repeaters	N/A	N/A	4.3%

2.6 BIST Configuration Comparison

As found in [10], the routing architecture is the primary influence on the total number of test configurations and on the total test time. The comparisons of the routing architecture presented in Tables 2.2, 2.3, and 2.4, therefore, allow a good relative comparison of the total number of BIST configurations for the ORCA and Xilinx FPGAs. Table 2.5 summarizes the total number of logic and routing BIST configurations for the ORCA and Xilinx FPGAs.

Table 2.5 BIST Configurations for ORCA and Xilinx FPGAs [10]

	ORCA 2C	ORCA 2CA	Xilinx 4000E and Spartan	Xilinx 4000XL and XLA
# logic BIST configs	9	14	12	12
# routing BIST configs	27/44	27/44	128	206

As Table 2.5 demonstrates, the primary component in the total number of BIST configurations is the number of routing BIST configurations. The ORCA 2C and 2CA FPGAs required 27 and 44 routing BIST configurations, with the differing numbers due to the size of the STARS [10]. The Xilinx 4000E and Spartan FPGAs required 128 routing BIST configurations while the Xilinx 4000XL/XLA FPGAs required 206 routing BIST configurations. The Atmel device has a fine-grain PLB architecture, with a rotationally symmetrical routing architecture. In addition, the composition of routing resources in the Atmel FPGA, as illustrated in Table 2.3 and Table 2.4, is somewhere between the percentages found in the ORCA and Xilinx FPGAs, and will have an impact on the total number of configurations since the routing architecture is the primary influence. The differences between the architectures of these FPGAs affect the number of test configurations for the respective FPGA architectures to completely test the routing resources. One of these differences is that the main types of PIPs within the respective FPGAs differ in their percent composition and in their amount of inputs [10]. The cross-point and break-point PIPs make up 81% of the total PIPs in the ORCA FPGAs and the remaining 19% comprises the multiplexer PIPs [10]. In addition, there are no switch-box PIPS in the ORCA routing resources [10]. The more difficult to test PIPs are the multiplexer and switch-box PIPs. In addition, the size of the multiplexer PIPs is smaller in the ORCA FPGAs in comparison with the Xilinx FPGAs, 5 inputs compared to 35 inputs [10]. Consequently, more configurations are required to completely test these types of PIPs in comparison to the break-point or cross-point PIPs.

Another difference arises in the carry logic where, in the ORCA FPGAs, there is dedicated carry routing to the four adjacent cells but the carry-out output can also be placed on one of the five PLB outputs, allowing for better observability of the signal [10]. In the Xilinx FPGAs the carry-out can only be observed on dedicated routing that goes from a given PLB to only one PLB located directly above that PLB. A third difference is the sharing of the routing resources associated with the PLBs [10]. In the ORCA FPGAs, the routing resources are not shared between two adjacent PLBs whereas in the Xilinx FPGAs, the routing resources are shared between the two adjacent PLBs [10]. In addition to the effect of PIP types in the ORCA and Xilinx FPGAs, the routing resources in the two FPGAs contain rotating and staggered busses, which make testing the busses more difficult and increase the number of configurations required to test the FPGA interconnect [10]. Finally, the inputs and outputs to and from the ORCA FPGAs can be connected to all routing resources on any side of the PLB while in the Xilinx FPGAs, the inputs and outputs can be connected to and from a subset of the routing resources on only two sides of the PLB making testing more complex [10].

2.7 BIST Development

In the previous work in BIST for FPGAs performed on the ORCA and Xilinx FPGAs [9], [10], [11], [20], vendor-supplied computer-aided design (CAD) tools were used in conjunction with custom designed programs to develop and generate the logic and routing BIST configurations [28]. In the case of the ORCA FPGAs, custom programs were used to create textual netlist files, NeoCAD Design Language (NCL) files, which are interpreted through the vendor-supplied CAD tools provided from the manufacturer of the ORCA FPGAs in order to produce the download bitstreams needed for the logic and routing BIST sessions [28]. These textual netlists describe the configuration and placement of the PLBs (TPG, BUTs, and ORAs) and the routing resources used to route signals between the PLBs [28].

The BIST development for the Xilinx FPGAs used custom programs to generate XDL (Xilinx Design Language) files that could be interpreted via the vendor-supplied CAD tools from Xilinx in order to produce the bitstreams needed for the logic and routing BIST sessions. The vendor provided place and route (PAR) tools did not allow adequate control over the routing resources that was needed for the application of BIST.

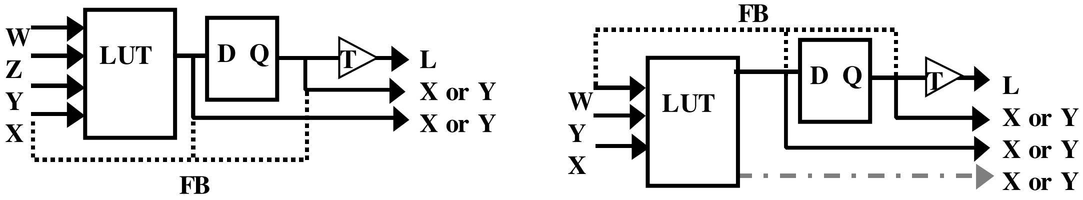
Therefore, a more controlled approach was needed and the idea of custom programs to generate XDL was chosen for its ability to control the interconnections in the routing resources of the FPGA.

2.7.1 Macro Generation Language (MGL)

In the case of the Atmel FPGA, the vendor-supplied CAD tools include a design-oriented programming language, called Macro Generation Language (MGL), that can be used to instantiate designs into the FPGA and produce download bitstreams without the need for the development of custom programs. The Figaro Integrated Development System (IDS) is the software available from Atmel for implementation of designs into AT40K FPGAs, which includes the programming language, MGL, that can be compiled and used to instantiate designs, including placement and routing, and to create bitstreams for downloading into the FPGA [29]. The MGL combines aspects of many modern programming languages with characteristics of hardware description languages such as VHDL or Verilog.

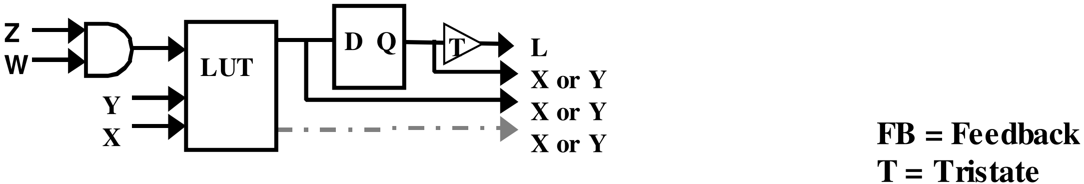
The language is a method of creating user-defined, parameterized circuits that meet desired design specifications. Designs are created in much the same manner as that used in generating a computer program in a programming language such as C or C++. Just as in many programming languages, pre-processor directives, global variables and constants may also be defined. The language is a strongly-typed language such as the case in VHDL, meaning that all objects must be assigned a specific type. However, unlike VHDL, MGL is a case-sensitive language where *function()* and *Function()* represent two different and independent functions. To generate a design using MGL, three components are necessary within the code: user-defined functions (i.e., BUTs, ORAs, TPGs), the target FPGA device (i.e., AT40K05, AT40K10, etc.), and the inputs and outputs to the circuit (i.e., clock input, reset input, pass/fail output) [29].

In order to configure a PLB in the FPGA, MGL requires the use of predefined macros ranging in complexity from AND gates to multiplexers and decoders [29]. In addition, MGL can utilize dynamic macros that allow a variety of functionality to be configured in the PLB. Three basic types of dynamic macros can be used to configure the PLB through MGL: the FGEN1 macro, the FGEN2 macro, and the MGEN macro, which are illustrated in Figure 2.13 [30].



a) FGEN1 Macro

b) FGEN2 Macro



c) MGEN2 Macro

Figure 2.13 Dynamic Macros Utilized in MGL [30]

The FGEN1 dynamic macro allows up to a 4-input logic function to be implemented in the PLB; with feedback up to a 4-input logic function can be implemented, but with only three external inputs [30]. The LUTs in this dynamic macro are used in conjunction with one another through the use of a multiplexer, shown in Figure 2.3, which selects between the individual LUT outputs resulting in a combined 16-bits for the 4-input logic function. The output can be registered or combinational by either using or not using the D flip-flop, respectively [30]. A tri-state output is also available if desired [30].

The FGEN2 macro is similar to the FGEN1 macro in that the outputs can be registered or combinational and have an optional tri-state. However, the LUTs function separately and two logic functions of only up to three inputs can be implemented in the PLB [30]. If feedback is utilized in the FGEN2 macro, then one input is sacrificed making a function using up to two external inputs possible [30].

The MGEN macro allows up to a 4-input logic function and has an upstream AND of two of the inputs to the PLB in front of the LUTs [30]. The AND gate is intended for use in multiplier-based functions such as those used in many DSP applications [14], [15], [29]. As in the FGEN1 and FGEN2 macros, the output can be registered or combinational and the optional tri-state on one output is available.

Nodal statements In MGL can be used to configure the routing resources in order to define the interconnect of a design. Signals must have a source and destination specified in the design and must be routed in order from the source to the destination. The MGL compiler ignores any nodal statements not meeting these conditions. The documentation provided by Atmel in [29], [30] gives the net names of the most commonly used routing resources, which include the PLB direct connections, *x4* lines, and *x8* lines. However, net names such as those associated with the clock and reset routing and the periphery routing at the I/O cells are not given in the documentation. Routing can be specified for these types of routing by identifying the appropriate net names associated with these routing resources through observation of the output files of the Atmel CAD tools, which is discussed further in Chapters 3 and 4.

Configuration of the design implemented in the FPGA through the use of MGL is limited to the dynamic macros. For designs not needing as much control over the FPGA configuration, the information supplied in [29], [30] is sufficient to produce working designs in the FPGA. However, for applying BIST to the FPGA, complete control of the configuration of the PLBs and routing resources in the FPGA is necessary in order to achieve maximum fault coverage.

2.8 Thesis Restatement

The focus of this thesis is to apply the BIST approaches originally developed for the ORCA and Xilinx FPGAs to the Atmel AT40K series FPGAs and the AT94K series FPGA core. The differences that exist between the various types of FPGA architectures can be expected to impact the application and development of BIST configurations as well as the total number of configurations needed. Development and generation of the BIST configurations with the vendor-supplied CAD tools, MGL and Figaro IDS, appear to offer a more integrated approach compared to the prior development efforts for ORCA and Xilinx FPGAs. Chapters 3 and 4 detail the logic BIST and routing BIST approaches applied to the Atmel FPGA, respectively. In addition, the use of MGL for development and generation of BIST configurations will be discussed along with the associated advantages and limitations. Finally, the total number of BIST configurations needed to completely test the programmable logic and routing resources in the FPGA core will be presented and analyzed.

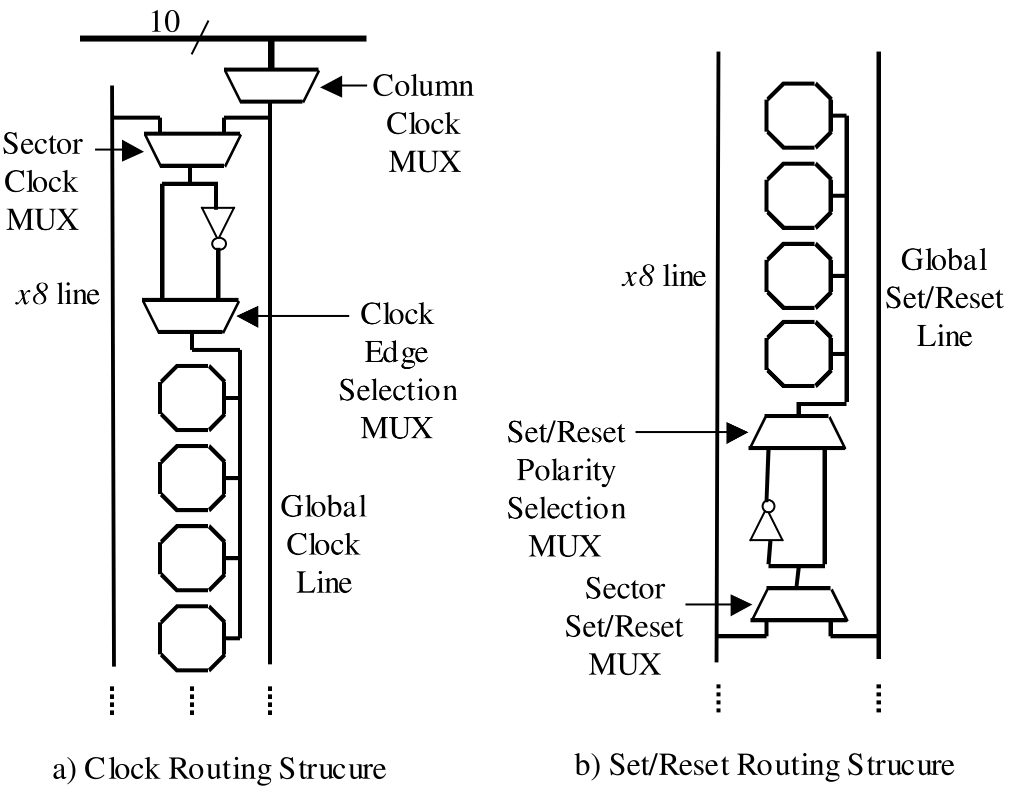
CHAPTER THREE

LOGIC BIST

In this chapter, the logic BIST methodology is presented in its application and adaptation to the FPGA core architecture in the AT94K series SoC. The architectural issues that impact the logic BIST architecture and implementation are discussed. Evaluation of the logic BIST approach through fault simulation is also presented. Automatic generation of the logic BIST configurations for any size AT40K FPGA and AT94K FPGA core using MGL will be discussed. An analysis of the advantages and limitations of MGL will be presented. Finally, the results of logic BIST for the FPGA core in the AT94K series SoC is compared to the results obtained for the ORCA [9] and Xilinx [10] FPGAs.

3.1 Architectural Implications on Logic BIST Architecture

The Atmel FPGA architecture imposes new constraints on the BIST methodology originally applied to the ORCA [9] and Xilinx [10] FPGAs, which include the type of TPG used, BUT configurations, and ORA implementations. The architecture of the PLB and its associated local routing resources has several implications on the logic BIST architecture in the FPGA core of the AT94K series SoC. The architectural implications of the FPGA core on the logic BIST approach are presented in the subsequent discussions.



a) Clock Routing Strucure b) Set/Reset Routing Strucure

Figure 3.1 Clock and Set/Reset Routing Structure in Atmel FPGA [14], [15]

One of the main factors influencing the logic BIST architecture is the column-based reset and clocking schemes employed in the Atmel FPGA core. The clock and reset inputs for the D flip-flop present in each PLB are arranged in banks of four PLBs along columns as illustrated in Figure 3.1. The clock connections to each bank of four PLBs in the FPGA array can be made either from a *x8* line, shown on the left in Figure 3.1a, or from the global clock line, shown on the right in Figure 3.1a [14], [15]. Each column of PLBs can connect to one of 10 global clock signals that connect to the FPGA I/O cells [14], [15]. The clock signal may also be routed onto a *x8* line and then connected to the PLB as well [14], [15]. Each bank of four PLBs has an associated Sector Clock Multiplexer (MUX) that selects the desired clock signal from the global clock line or from the *x8* line [14], [15]. The triggering for the D flip-flop is either rising or falling edge, and is set by selecting either the non-inverting or inverting input to the

Clock Edge Selection MUX [14], [15]. A similar scheme is employed in the routing for the asynchronous Set/Reset signal for the flip-flops. Banks of four PLBs are connected to a Sector Set/Reset Multiplexer that selects the desired set/reset signal either from the global set/reset line or the *x8* line [14], [15]. The configuration bits for the individual flip-flops determine the function (either Set or Reset). The polarity (active high or active low) of the incoming Set/Reset signal is determined by a configuration bit associated with the Set/Reset Polarity Selection MUX [14], [15].

As a result of the column-oriented clock and reset routing schemes illustrated in Figure 3.1a and 3.1b, the logic BIST architecture employed must also be column-based in order to test the clock and reset inputs to the BUTs during the logic BIST configurations. A row-based logic BIST architecture does not allow the Set/Reset or rising/falling edge triggering associated with the flip-flops in the BUTs to be tested. Due to the column-orientation of the Set/Reset of the flip-flops, an ORA in a row-based logic BIST architecture would be reset any time a BUT is reset thus causing the loss of any mismatches that could have been latched in the flip-flop within the ORA.

The architecture of the PLB also has a large impact on the type of ORA implemented in the logic BIST test sessions. In this FPGA, the PLB does not contain sufficient logic resources to implement more than a comparison of three inputs while having feedback to latch up any mismatch between the three inputs. Therefore, only one output from each adjacent BUT can be compared during a given BIST configuration as illustrated by the ORA structure in Figure 3.2.

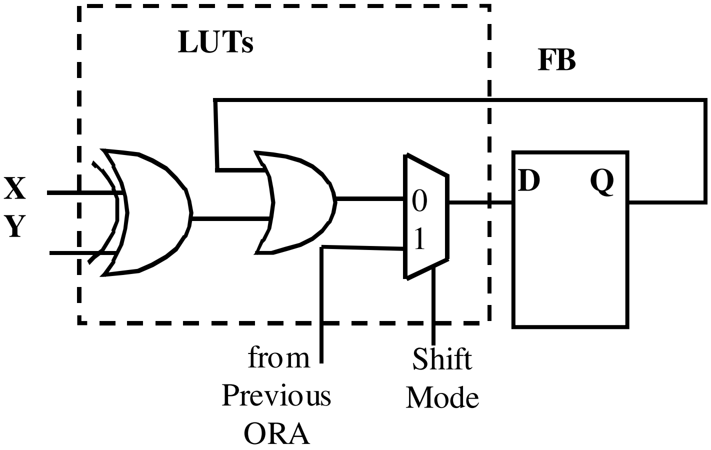
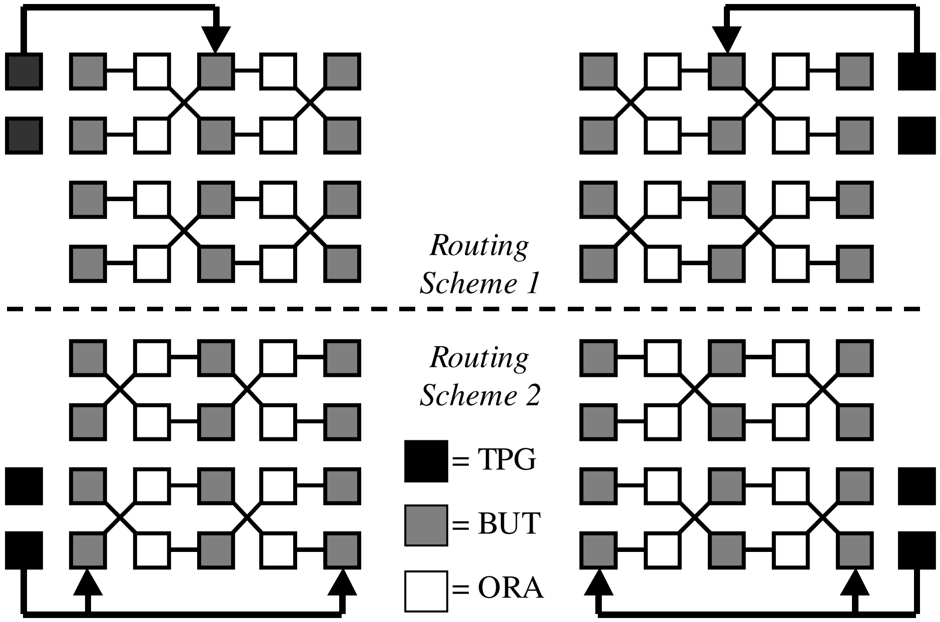


Figure 3.2 ORA Structure for Logic BIST

The PLB outputs, denoted as **X** and **Y** in Figure 2.3, have an impact on the logic BIST architecture as these outputs connect only to adjacent PLBs as shown in Figure 2.6a. The diagonal and orthogonal orientations of the **X** and **Y** connections, respectively, impose restrictions on the BUT to ORA connections in the application of logic BIST. Since each is a PLB output and must be observed, both outputs must be compared during logic BIST. Furthermore, the PLBs acting as ORAs can only observe one **X** output and one **Y** output at a given time due to the local interconnect multiplexer PIPs shown in Figure 2.5. Therefore, the most efficient choice of local routing resources for the application of BIST is the **X** and **Y** outputs to implement the BUT to ORA connections in the logic BIST architecture. In order to ensure comparison of the **X** output of one BUT with the **Y** output of another BUT, a set of routing schemes was devised, as shown in Figure 3.3, to allow outputs of the PLBs configured as BUTs to be observed by the ORAs. During subsequent test phases in a given logic BIST test session the routing schemes are alternated between routing schemes 1 and 2 such that the **X** and **Y** outputs of adjacent BUTs are observed in an alternating fashion.

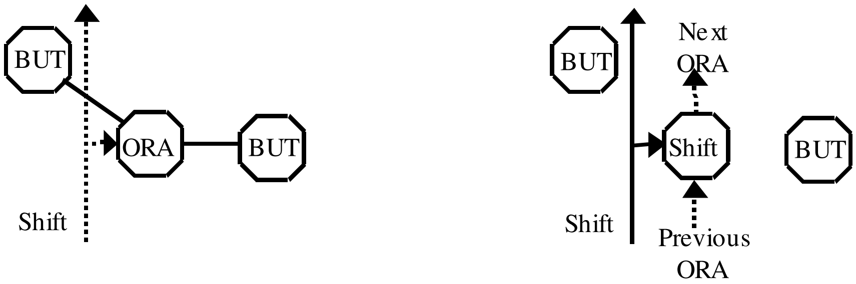


a) Test Session #1 b) Test Session #2

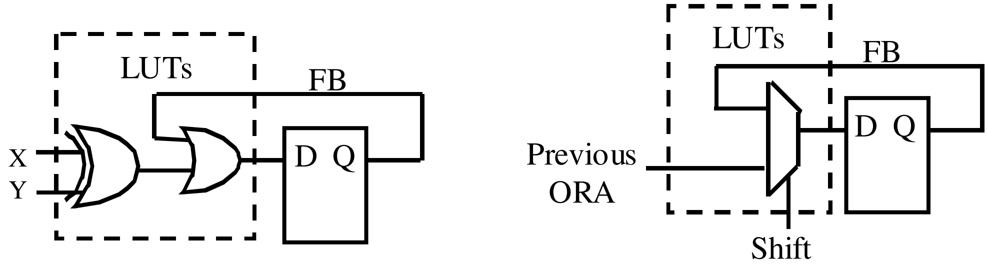
Figure 3.3 BUT to ORA Connections During Logic BIST

Another implication that the small PLB imposes on the ORA structure is that an ORA with a comparison of two inputs and feedback for the latching of mismatches does not have sufficient additional logic resources to include a shift register for retrieval of the BIST results at the end of a given test phase. Three alternatives are available to work around this problem.

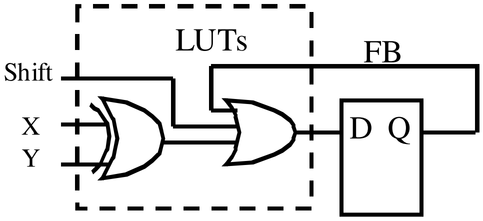
An ORA can be implemented as shown in Figure 3.4b, and after completion of a logic BIST test phase, be reconfigured as a shift register as in Figure 3.4d to shift out the BIST results. Since the FPGA is capable of dynamic partial reconfiguration, this can be done by using the synchronous RAM configuration mode to write only to the PLBs configured as ORAs and BUTs along the edges needed to route the shift data through to the next column in the shift register. This partial reconfiguration can also be performed by the AVR core in the AT94K series SoC [14], [15]. The logic BIST sequence in this case would be as follows: 1) download the bitstream for a logic BIST test session, 2) run the BIST sequence, 3) reconfigure the ORAs as a shift register, and 4) shift out the Pass/Fail results contained in the ORAs. Figure 3.4a illustrates the configuration of an individual ORA and its corresponding BUT connections. Figure 3.4c illustrates the transformation via dynamic partial reconfiguration to a shift register for retrieval of the BIST results. Figure 3.4b and 3.4d show the internal PLB structure before and after its transformation to a shift register, respectively. The shift signal is shown as an input to the PLB in Figure 3.4e in the actual implementation of the desired ORA shown in Figure 3.4b. This signal is present since it must be routed to the ORAs in the initial download bitstream for logic BIST due to constraints imposed on routing in MGL. Each route in MGL must have both a source and destination specified, therefore, the shift signal must be an input to the ORAs in order to route the signal to the ORAs for retrieval of BIST results. The presence of this signal does give an advantage. The shift signal can be used for additional testing of the PLBs and can be used to force a failure indication in the ORAs to provide a functionality check in the downloaded configuration.



a) Comparison ORA c) Reconfigured ORA



b) Comparison ORA PLB Structure d) Shift Register PLB Structure

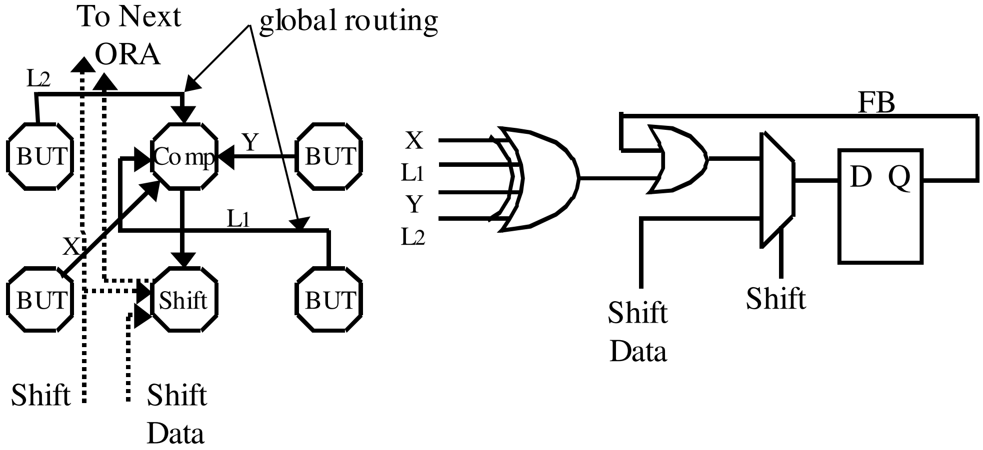


e) Comparison ORA PLB Structure w/ Shift

Figure 3.4 Comparison ORA and Reconfigured ORA for Shift Register

This approach has the best diagnostic resolution of the available approaches since an ORA compares the corresponding outputs of two BUTs, a fault can be diagnosed to a particular PLB. By knowing the shift register's order and placement, the ORA results that are shifted out can be used to diagnose the location of a faulty PLB based on the BIST results [19].

Another approach available is to implement a 2-PLB ORA where one PLB compares the corresponding outputs of four BUTs. The other PLB is used to latch any mismatch and for shifting out the results at the end of the BIST sequence as illustrated in Figure 3.5. The BIST sequence in this case would be as follows: 1) Download the bitstream for a logic BIST test session, 2) Run the BIST sequence, and 3) Shift out the Pass/Fail results contained in the ORAs. This ORA configuration yields a lower diagnostic resolution compared to the first approach since the outputs of four BUTs are being compared in a single ORA. However, partial reconfiguration is not required in order to retrieve BIST results.



a) 2-PLB ORA Connections b) 2-PLB ORA Structure

Figure 3.5 2-PLB ORA with Shift Register

The **X** and **Y** PLB outputs are not being observed on two of the BUTs in the 2-PLB ORA implementation. As can be seen in Figure 3.5, the outputs of those PLBs must be routed out of the PLB through the **L** output in order to reach the global routing resources to connect to the comparison portion of the 2-PLB ORA. This imposes a more complicated routing scheme and could lead to lower fault coverage.

The last approach available is to read the ORA flip-flop contents through configuration memory readback. The synchronous RAM mode can also be used to read the configuration memory in order to obtain pass/fail indications held within the flip-flops in the ORAs [14], [15]. However, in the case of the AT94K series SoC, the configuration memory of the FPGA core can be written by the AVR core, but cannot be read by the AVR core [14]. As a result, the single PLB ORA implementation with reconfiguration into a shift register, performed by the AVR, is the best choice for application to the FPGA core of the AT94K series SoC.

The TPG chosen for the logic BIST test sessions for the Atmel PLB was a 5-bit binary counter to drive the five inputs to the PLB. The PLB is only capable of implementing a single counter cell. The **Y** output of the PLB connects directly either vertically or horizontally to an adjacent PLB and is ideal for implementing the carry signal in a binary counter. Since the PLB is only capable of implementing a single counter cell, a TPG such as an LFSR would require more PLBs and would not be as efficient an implementation as the binary counter for the case of the PLB due to the requirement of additional logic for implementing an all 0's state in the LFSR. The smallest array size in the FPGA is 16x16, and is only capable of handling two identical TPGs no larger than eight PLBs each. In this array size, the 5-bit binary counter is easily placed within these constraints. Therefore, the logic BIST architecture as described in this section will work for any size AT40K series FPGA or AT94K series FPGA core.

3.2 Modeling the PLB for Fault Simulations

Considerable knowledge about the logic within the PLB was obtained by studying the datasheets and technical references available from Atmel in [14], [15], [29], [30]. The majority of the configuration bits associated with the PLB were determined through performing various download tests and observing the behavior of the PLB. Through knowledge of the logic and the associated configuration bits, a gate-level model of the PLB was developed in order to perform single stuck-at, gate level fault simulations.

From these fault simulations, the number of BUT configurations for the logic BIST test sessions as well as the fault coverage obtained with these BUT configurations could be evaluated.

3.2.1 MGL Generated BIST Configurations

An MGL program was developed to automatically generate BIST configurations and the logic BIST architecture for testing the PLBs in the Atmel. This MGL program was designed to generate logic BIST configuration for any size FPGA array for both the AT40K series FPGAs and the AT94K series FPGA cores. By adjusting a few parameters within the program, five logic BIST configurations can be automatically generated with four different orientations (TPG located on either the East, West, North, or South side of the array, as illustrated in Figure 3.12) for any size FPGA array. The program for the automatic generation of these logic BIST configurations consists of approximately 2500 non-commented lines of MGL source code. The flow diagram shown in Figure 3.6 illustrates the overall structure and operation of the MGL program.

The first parameter to be selected in the MGL program is the device in which the logic BIST configurations are to be generated. All possible devices and packages are given as comments within the MGL program in order to make device selection straightforward. Next, the desired TPG location is set in order to determine the orientation of the logic BIST session. As an example, setting the TPG location to '1' places the TPGs in the first column on the West side of the array. With the TPG location set to '2', the TPG column is located on the East Side of the array. For TPG locations '3' and '4', the TPGs are oriented horizontally in rows and located along the North or South sides of the array, respectively.

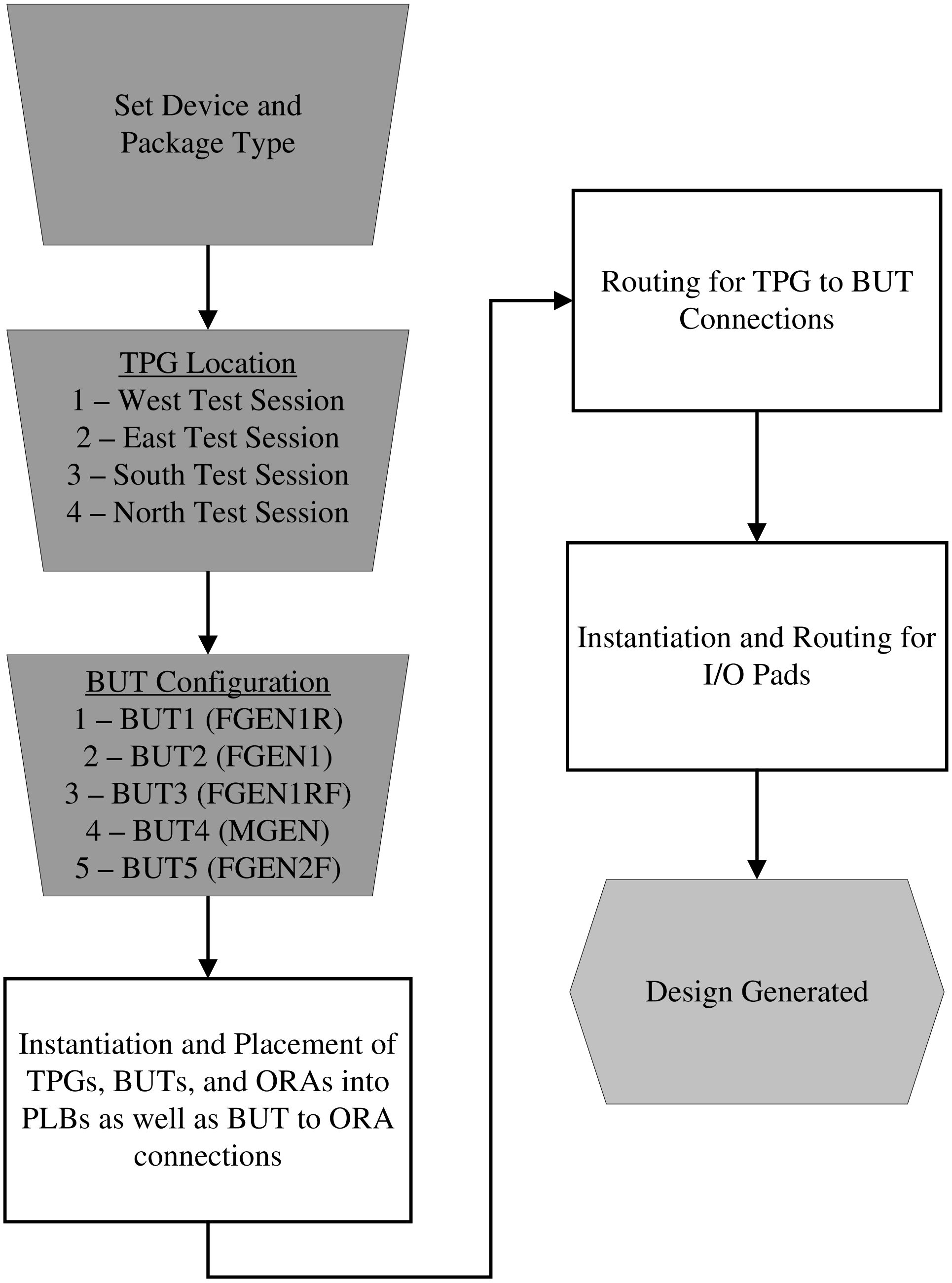


Figure 3.6 Logic BIST MGL Program Flow Diagram

Within a given session, East, West, North, or South, five BUT configurations can be generated. These five configurations were chosen from the possible dynamic macros available through MGL to give maximum fault coverage with the fewest BIST test phases. The five chosen were the FGEN1R, FGEN1, FGEN1RF, MGEN, and FGEN2F. Once these parameters are set, the MGL program can be compiled to generate the selected test session and BUT configuration for the selected device.

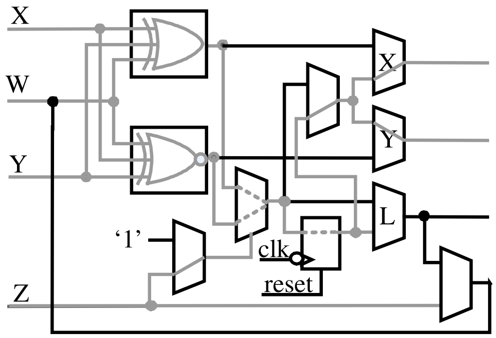
The program begins by instantiating the TPGs, BUTs, and ORAs into their respective columns or rows, depending on the orientation, as well as making the BUT to ORA connections. Next, the interconnections between the TPGs and BUTs are routed. These interconnections are followed by the instantiation of the routing for the Shift signal (Figure 3.4) for the ORAs and routing the Pass/Fail indication to an I/O pad from the last ORA that will be in the shift register after partial reconfiguration. Finally, the I/O pads are instantiated to connect the input and output signals (Clock, Shift, and Pass/Fail). The I/O pads instantiated are for the 84, 144, and 208 pin devices with package designations AJC, BQC, and DQC, respectively, which are illustrated in Table 3.1.

Table 3.1 Pin Numbers for I/O Cells for Logic BIST

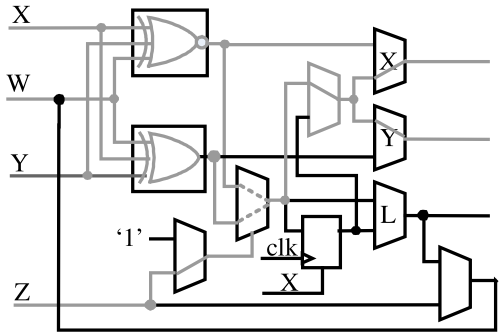
Signal	84 Pin AJC Package	144 Pin BQC Package	208 Pin DQC Package
Clock	13	2	4
Shift	81	121	174
Pass/Fail	23	19	27

In developing the MGL program, obstacles were encountered where the provided documentation was insufficient to determine how to achieve adequate control over some of the routing to the PLBs and to the I/O pads. In this case, the text-based output files produced by the Figaro IDS software were investigated for pertinent information. This information consisted of the names of routing resources within the FPGA that could be applied in the MGL program to obtain the necessary control in routing the configuration of the FPGA for application of BIST.

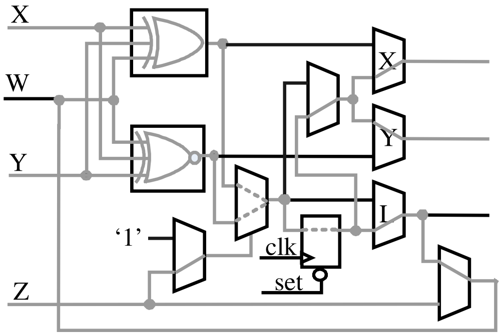
The five BUT configurations, FGEN1R, FGEN1, FGEN1RF, MGEN, and FGEN2F, are illustrated in Figure 3.7 in terms of the configuration of the logic resources. Of the three FGEN1 type macros, the first implements a registered output, 4-input exclusive-OR logic function (FGEN1R), the second implements a combinational output, 4-input exclusive-NOR logic function (FGEN1), and the third implements a 3-input exclusive-OR logic function with registered feedback (FGEN1RF). The fourth BUT configuration uses is an MGEN macro which implements a 4-input exclusive-NOR logic function with a combinational output. The final BUT configuration implements a 3-input logic function shown in Figure 3.7e and utilizes combinational feedback (FGEN2F). Where an 'X' is shown for the set/reset signal, the value is considered as a don't care value by MGL and is arbitrarily assigned a value by the CAD tools.



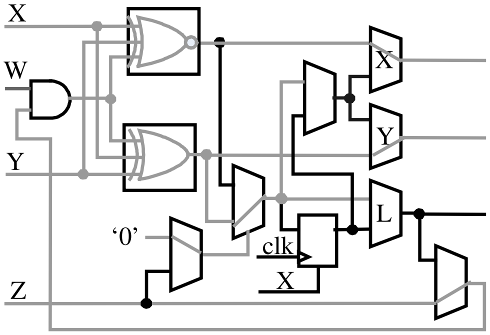
a) BUT 1 – FGEN1R



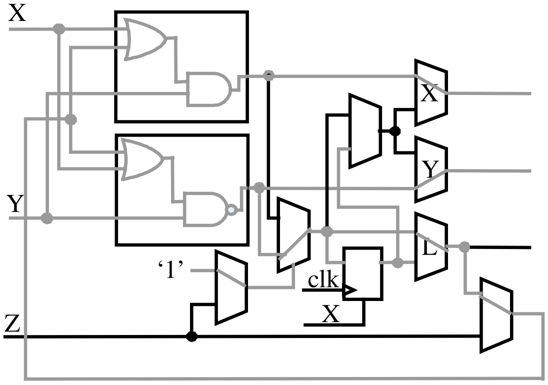
b) BUT 2 - FGEN1



c) BUT 3 – FGEN1RF



d) BUT 4 - MGEN



e) BUT 5 – FGEN2F

Figure 3.7 MGL Generated BUT Configurations

These configurations were found to be the minimum number of configurations that could be used to yield the maximum fault coverage in the PLB through the use of MGL. Depending on the BUT configuration, ORAs were configured to check for either BUT response matches or mismatches. As can be seen in Figure 3.7, the **X** and **Y** outputs of BUT configurations 4 and 5 will have different output values and require an ORA that will detect matches in the BUT output responses as an indication of a fault. BUT configurations 1 through 3 have the same output values on the **X** and **Y** outputs and require an ORA that will detect mismatches in the BUT output responses as an indication of a fault. This is accomplished by using an XOR function for configurations 1 through 3 and an XNOR function for configurations 4 and 5 in the ORA illustrated in Figure 3.4b.

Using the derived gate-level PLB model, fault simulations were performed with these five MGL generated BUT configurations. The five BUT configurations yielded a cumulative fault coverage of 97.9% for collapsed, single stuck-at gate-level faults within the PLB for those BUTs having both their **X** and **Y** outputs observed simultaneously. Table 3.2 gives the cumulative fault coverage obtained for the five BUT configurations. A total of 166 collapsed faults were simulated in the PLB with five faults left undetected after the five BIST configurations, of which three faults were potentially detected.

Table 3.2 MGL Generated BUT Configurations

BUT Configuration	Detected Faults	Undetected Faults	Potentially Detected Faults	Total Faults Simulated	Cumulative Fault Coverage
FGEN1R	99	67	0	166	59.64%
FGEN1	43	24	0	67	85.54%
FGEN1RF	9	15	1	24	91.27%
MGEN	7	8	0	15	95.48%
FGEN2F	3	5	2	8	97.89%

The edges of the PLB array pose an interesting problem in observability of the BUT outputs **X** and **Y** since, along the edge of the array, the BUTs do not have both outputs observed in the ORAs simultaneously. The fault coverage obtained in the left edge PLBs is slightly less than that obtained in the PLBs in the middle of the array. The five BUT configurations from MGL obtain 97.3% fault coverage in the left edge PLBs and 97.9% fault coverage in the right edge PLBs, which matches the number in the middle PLBs, as shown in Figure 3.8. These differences arise from the fact that the left and right edge PLBs have a different output observed in a given BUT configuration due to the alternating routing scheme shown in Figure 3.3.

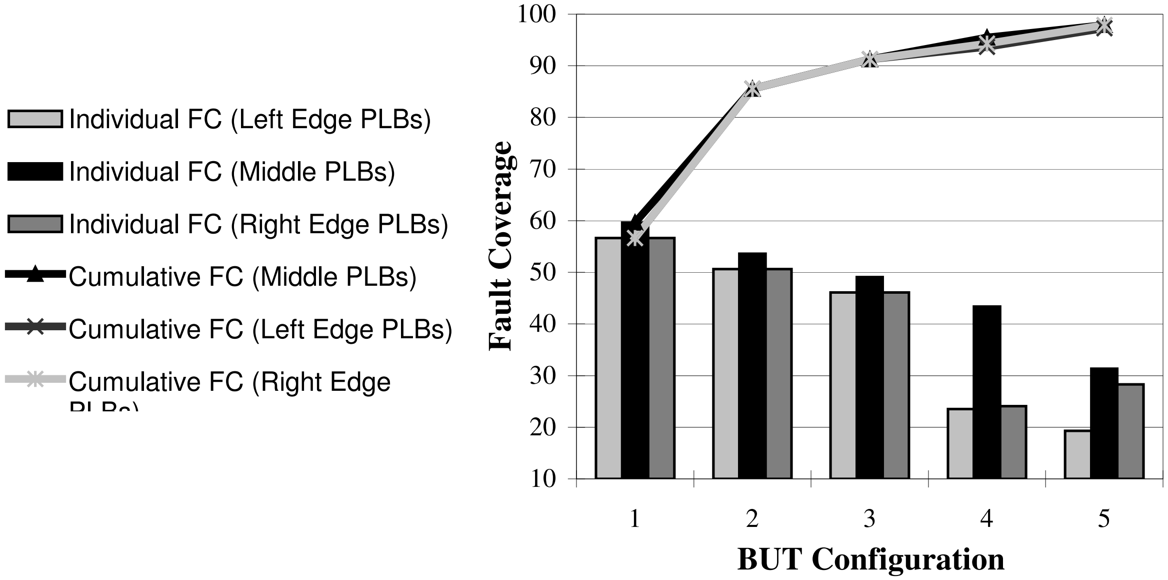
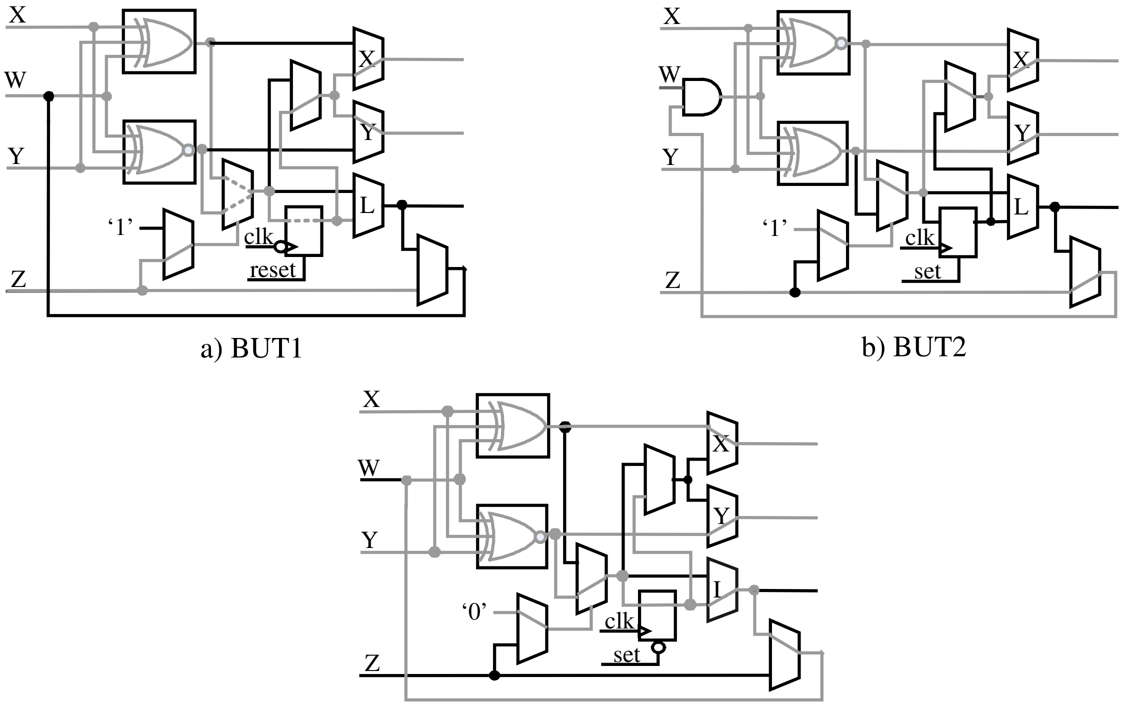


Figure 3.8 Middle and Edge Fault Coverage (Five BUT Configurations)

3.2.2 Theoretical Best Case

The MGL and Figaro software provided by Atmel does not grant the user complete control over the PLB functions and hence, the configuration bits associated with the PLB. The dynamic macros impose constraints on the controllability of the PLB that do not allow for necessary test conditions to be set up in the PLB to test for certain faults. This is in part due to the values placed on particular configuration bits, which, for the selected dynamic macro, can be either a logic '0' or logic '1', and MGL arbitrarily sets the bit to a value other than that desired for the proper test conditions. Therefore, it was necessary to look at alternatives and modifications to the MGL-based approach in order to maximize fault coverage. One solution is manipulating the download bitstream to set up the desired test conditions in the configuration bits before the BIST configuration is downloaded to the FPGA.

Through the single stuck-at, gate-level fault simulations a theoretical minimum number of BUT configurations was derived. Assuming observability of all PLB outputs, **X**, **Y**, and **L**, and complete controllability of all PLB configuration bits, a fault coverage of 100% for gate-level single stuck-at faults can be obtained in only three BUT configurations, which are shown in Figure 3.9. The first configuration (Figure 3.9a) is the same FGEN1R dynamic macro configuration generated by the MGL program while the second and third are custom configurations and do not correspond to any available dynamic macro. The second configuration (Figure 3.9b) is similar to an MGEN type dynamic macro in that it utilizes the upstream AND gate, however, the paths through the PLB logic are different than what can be arranged utilizing the MGEN dynamic macro. The third configuration (Figure 3.9c) utilizes the feedback in the PLB and utilizes different paths through the PLB logic than can be obtained with any of the dynamic macros.



a) BUT1

b) BUT2

c) BUT3

Figure 3.9 Theoretical Minimum Three BUT Configurations

These three BUT configurations represent the absolute best case for logic BIST. In actuality all outputs are not observable in the ORAs, only the **X** and **Y** outputs are observed in logic BIST, as demonstrated by the ORA structure in Figure 3.2 and the logic BIST architecture in Figure 3.3. However, through manipulation of the download bitstream, an alternative exists between this ideal case and the case with MGL.

3.2.3 Manually Generated BUT Configurations with Bitstream Manipulation

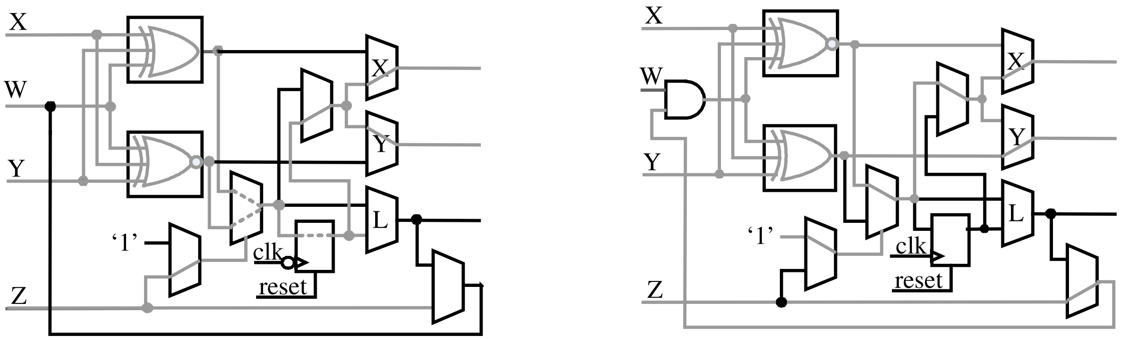
A C program was written to post-process the bitstreams produced through the compiled MGL program in order to produce the desired test conditions within the BUT configuration bits. This C program consisted of approximately 370 non-commented lines of C source code. In addition to increasing the fault coverage within the PLB, using a C program to manipulate the configuration bits within the PLB can reduce the number of

BUT configurations from five to four. This is accomplished by generating a template bitstream for the first and third BUT configurations produced by the MGL program and then manipulating the bitstream templates via the C program to produce the remaining two BUT configurations. These template bitstreams contain all the configuration and routing information needed to have the TPGs, BUTs, and ORAs and their interconnect configured in a given test session. The template bitstream for BUT configuration 1 configures the reset function and falling edge triggering of the BUTs while BUT configuration 3 configures the set function and rising edge triggering of the BUTs. BUT configurations 1, 2, and 4 are generated from the bitstream template for the first BUT configuration and BUT configuration 3 is generated from the bitstream template for the third BUT configuration. The template for the third BUT configuration is changed since BUT 3 for the manual configurations is different than for BUT 3 produced from MGL. Having the two template bitstreams accommodates testing for the set and reset functions and the rising/falling edge triggering since the C program does not manipulate the configuration bits associated with the set and reset or the rising/falling edge triggering. The only changes needed in the bitstream are performed by the C program, which changes the configuration bits for the BUTs and ORAs. The C program changes the configuration bits for those PLBs in order to change between BUT configurations, change the ORA configuration for either matching or mismatching, and change the alternating routing scheme between the BUTs and ORAs. Table 3.3 gives the cumulative fault coverage obtained for the logic associated with the BUT during the four BIST configurations.

Table 3.3 Manually Produced BUT Configurations

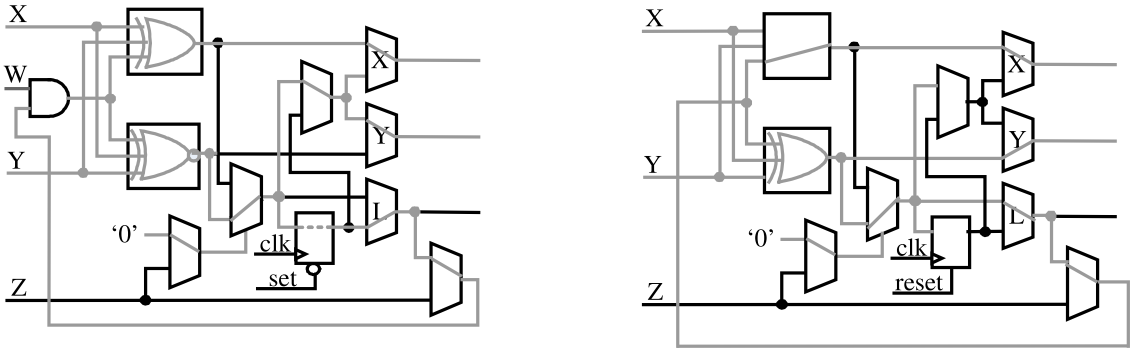
BUT Configuration	Detected Faults	Undetected Faults	Potentially Detected Faults	Total Faults Simulated	Cumulative Fault Coverage
1	99	67	0	166	59.64%
2	50	17	0	67	89.76%
3	13	4	1	37	97.89%
4	3	2	0	4	99.70%

The four BUT configurations are illustrated in Figure 3.10. The first configuration is an FGEN1R (the same as the first configuration with the five MGL generated configurations and the three theoretical minimum configurations), which is used as a template bitstream to produce the remaining three configurations. The second BUT configuration (Figure 3.10b) is a 4-input exclusive-OR logic function which utilizes the upstream AND gate and both the **X** and **Y** outputs are combinational. The third BUT configuration (Figure 3.10c) tests the registered feedback path in the PLB again utilizing the upstream AND gate by ANDing the feedback with a PLB input. The outputs in this configuration are again combinational. The final BUT configuration (Figure 3.11d) tests the combinational feedback path in the PLB and tests the outputs directly from the LUTs in the PLB. These configurations also require ORAs that detect a match or a mismatch, depending on BUT configurations. As can be seen, the **X** and **Y** outputs will be the same logic values in BUT configurations 1 and 4 while in BUT configurations 2 and 3, the **X** and **Y** outputs will have different logic values, which require ORAs configured to check for matches and mismatches, respectively.



a) BUT 1

b) BUT 2



c) BUT 3

d) BUT 4

Figure 3.10 Manually Generated Four BUT Configurations

The four BIST configurations generated from the C program yield a total fault coverage of 99.7%, which is closer to the 100% obtained with the theoretical minimum three BUT configurations. The five MGL configurations resulted in 97.9% fault coverage with faults left undetected. However, in the four BUT configurations, the one fault left undetected is actually potentially detected and is guaranteed to be detected during the routing BIST configurations.

The disadvantage in utilizing these four BUT configurations to test the PLBs in the array is a decrease in fault coverage observed in the BUTs on the left and right edges of the PLB array. Along these edges, the fault coverage observed decreases from that of the BUTs in the middle of the array shown in Table 3.2 due to the alternating **X** and **Y** BUT to ORA routing schemes and the configuration of the BUTs during the test phases.

The total fault coverage in the BUTs obtained during logic BIST along the left and right edges of the array drops to 82.23% for left edge BUTs and drops to 87.95% for right edge BUTs from the 99.7% obtained in the BUTs in the middle of the array, as is illustrated in Figure 3.11.

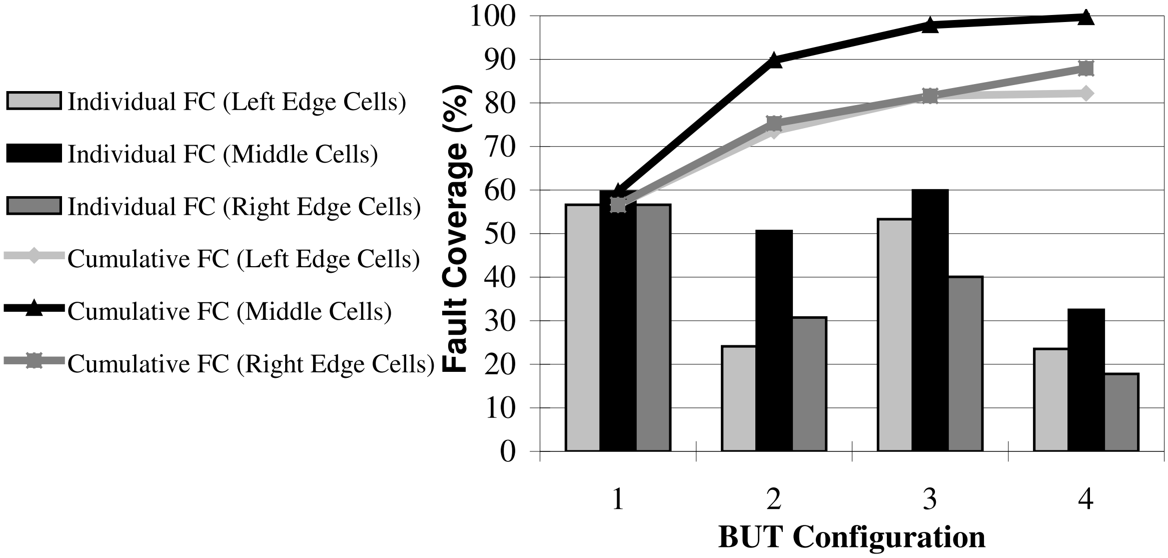


Figure 3.11 Middle and Edge PLB Fault Coverage (Four BUT Configurations)

One solution to the problem of decreased fault coverage along the edges of the array is to apply each BUT configuration twice using both BUT to ORA routing schemes. This doubles the number of test phases in order to obtain increased fault coverage only along the edges of the array. However, this problem can be almost eliminated by rotating the orientation of the logic BIST architecture from column-based to row-based to obtain North and South test sessions. This rotation is demonstrated in Figure 3.12, which shows the four directional orientations of the logic BIST test sessions, East, West, North, and South. This rotation also gives a total of 16 logic BIST configurations, four BUT configurations for each directional orientation of the logic BIST architecture.

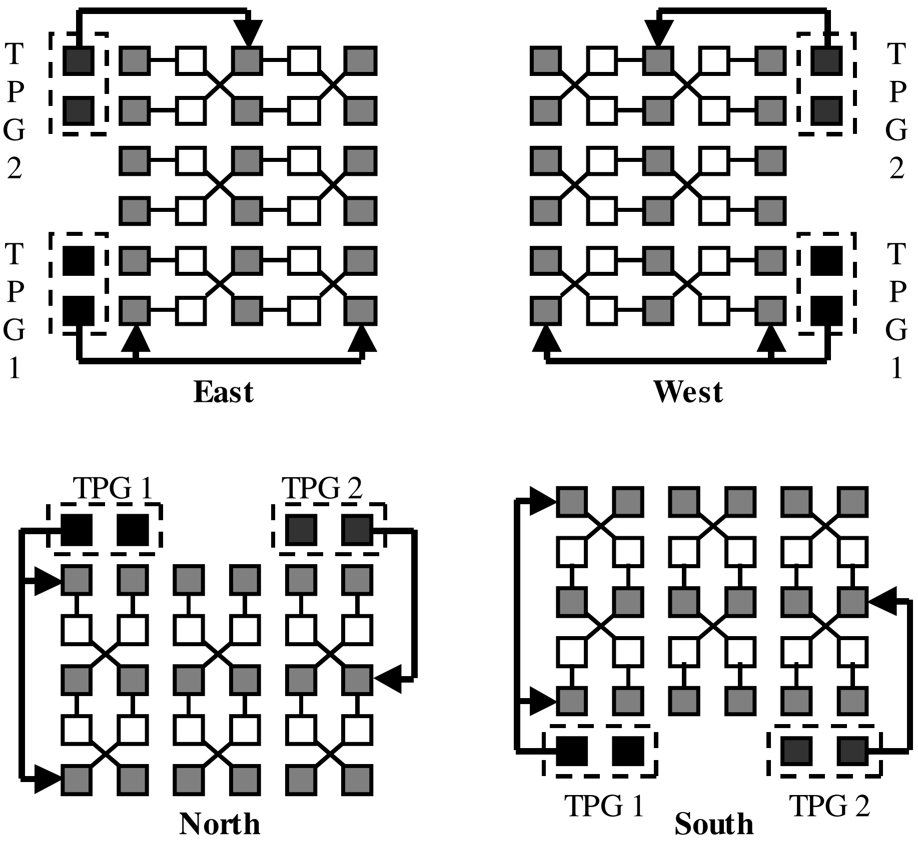


Figure 3.12 Column-Based to Row-Based Rotation for Logic BIST

Due to the column-based banks of clocks and resets, the flip-flops cannot be tested during the row-based North and South test sessions; however, these have already been tested during the column-based East and West test sessions. By rotating the logic BIST test sessions, all but eight of the PLBs in the array will have 99.7% fault coverage. The eight PLBs are located in the four corners of the array as illustrated in Figure 3.13.

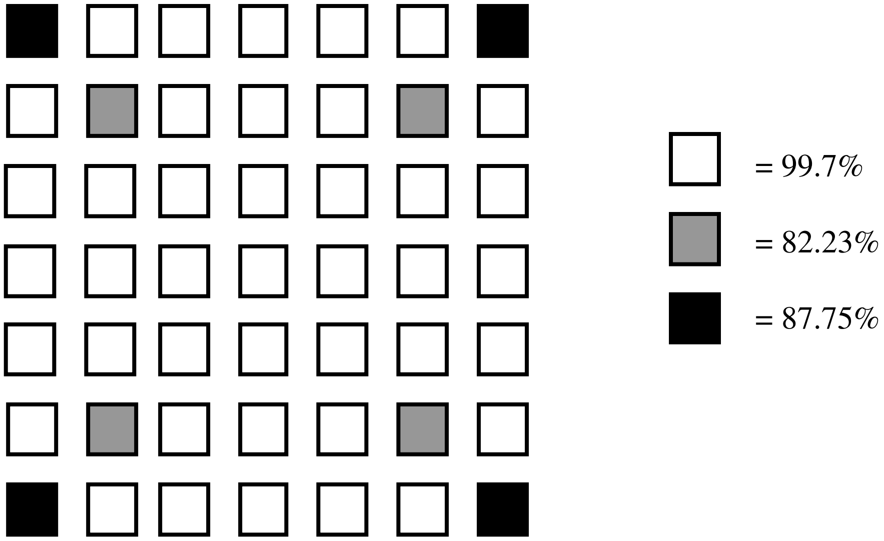


Figure 3.13 Fault Coverage of PLBs Located in Four Corners of the Array

An advantage of performing the rotation of the logic BIST test sessions includes allowing for horizontal and vertical local routing resource testing to be performed simultaneously with the logic BIST. Since the local routing configuration bits are associated with the PLB configuration bits, testing can be performed on the local interconnect simultaneously by using the C program to change connections to the local routing resources between test phases. The inputs to the PLB have multiplexers to select an input signal from one of the five horizontal or five vertical *x4* lines, as shown in Figure 2.5. By rotating through the combinations of inputs during test phases, these multiplexer PIPs can be tested simultaneously during logic BIST.

Through fault simulations, it was found that the rotation on the input connections allowed all but one input to one multiplexer in the local routing resources to be tested during the four logic BIST test sessions. However, this particular input is used for the Shift signal to the ORAs during the shifting out of the BIST results. As a result, a shift register test was added to test this input. In this additional test a logic '1' is input to the ORAs on the Shift input which causes all the ORAs to latch the logic '1' as can be seen in

Figure 3.4e. The ORAs are then reconfigured as a shift register and the ORA values are shifted out. Adding local routing resource faults to the BUTs increases the total number of collapsed faults from 166 to 298. These additional faults include faults in the MUX PIPs and cross-point PIPs at the **W**, **X**, **Y**, and **Z** inputs to the PLBs from the global routing resources.

The results of the fault simulations performed on both the PLB and local interconnect are given in Table 3.3. The configurations denoted by EW indicate fault simulations for PLBs during East/West orientations of logic BIST while the configurations denoted by NS indicate fault simulations for North/South orientations of logic BIST. The ORAShift simulation indicates the additional test for the input of the shift signal which forces ORA failures. The total fault coverage obtained was 95.81%, with very few faults undetected. In performing these fault simulations, the PLBs functioning as both BUTs and ORAs were monitored to determine the total fault coverage. These are designated respectively in Table 3.4.

Table 3.4 Total Fault Coverage for PLB and Local Interconnect

Test Phase	Detected Faults	Undetected Faults	Potentially Detected Faults	Total Faults Simulated	Cumulative Fault Coverage
BUT1EW	151	147	0	298	50.67%
BUT2EW	68	79	0	147	73.49%
BUT3EW	20	59	1	79	80.37%
BUT4EW	7	52	0	61	82.72%
ORA1EW	5	47	0	52	84.40%
ORA2EW	4	43	0	47	85.74%
BUT1NS	13	30	0	43	90.10%
BUT2NS	10	20	0	30	93.46%
BUT3NS	0	20	0	20	93.46%
BUT4NS	0	20	0	20	93.46%
ORA1NS	4	16	0	20	94.80%
ORA2NS	2	14	0	16	95.47%
ORAShift	1	13	0	14	95.81%

Undetected faults were left only on the cross-point PIPs present on the **W**, **X**, **Y**, and **Z** vertical and horizontal inputs to the PLB, which are shown in Figure 2.7 as well as some of the **X** direct PLB connections and the tri-state buffer present on the **L** output of the PLB. The undetected faults left on the cross-point PIPs as well as those in the **X** direct connections and the tri-state buffer are tested during the routing BIST sessions, as will be discussed in Chapter 4. The one potentially detected fault is associated with the multiplexer for the L output and is detected when testing the tri-state buffer during routing BIST. Due to the amount of logic and routing resources utilized in logic BIST, the cross-point PIPs are not detected since opposite logic values are needed on the vertical and horizontal cross-point PIPs in order to detect faults. The **X** direct PLB connections left untested in the PLB array include one of four direct **X** connections in a PLB in a given location in the array, which result from the routing schemes (shown in Figure 3.3). The tri-state buffer on the **L** output of the PLB was not targeted during logic BIST due to the choice of ORA, as discussed in section 3.2. The best choice for regularity and efficient implementation allowed for two observable outputs from the PLB and did not allow for three observable outputs from the PLB, which meant the exclusion of **L** as an observable output during logic BIST.

3.3 MGL's Effects on Logic BIST Development and Application

The utilization of MGL to develop logic BIST configurations for the AT94K FPGA cores had both advantages and disadvantages. The language has similarities to modern programming languages and HDLs that allow users to more easily learn the basic constructs that are used and, thus, create designs relatively quickly once the language is mastered. These similarities allow the user to easily transition from a programming language or HDL into the MGL environment. As a result, the MGL approach facilitates an integrated approach to the development of automated BIST configuration generation within the FPGA manufacturer CAD tool suite. This was not available in previous development of BIST for FPGAs in [9], [10], [19], [20]. However, the language does not give complete control over the PLBs and routing resources within the FPGA nor is complete documentation given to provide necessary details for the development and application of BIST. The development of the logic BIST configurations, therefore, requires more than MGL alone. The implementation of the basic structure of the logic BIST configurations is fairly straightforward using MGL, however, additional control over the PLBs and routing resources is required through other means, such as the C program that was utilized to manipulate the download bitstream in this research and development effort.

3.4 Comparison of Logic BIST for Atmel, ORCA, and Xilinx FPGAs

As a result of the architectural differences in the PLBs between the ORCA and Xilinx FPGAs, a different number of logic BIST configurations were required to test the two FPGAs; as summarized in Table 3.4. Similarly the number of logic BIST configurations is different for the Atmel FPGA than that required for the ORCA and Xilinx FPGAs. In order to test just the logic in the PLBs in the Atmel FPGA, a total of four configurations is required. This is less than the number of configurations required for both the ORCA and Xilinx FPGAs. The primary reason is the smaller amount of logic present within the Atmel PLB, as pointed out in Table 2.1. A result of logic BIST unique to the Atmel FPGA is the loss of fault coverage in the PLBs located on the left and right edges of the array due to the FPGA routing architecture and small PLBs. This is overcome by the additional configurations performed by rotating the orientation of the logic BIST to perform the East, West, North, and South test sessions. In performing these additional sessions, not only do the PLBs along the edges gain improved fault coverage, but diagnostic resolution is also increased [19]. In addition, a large majority of the local routing resources is also being tested simultaneously.

Table 3.5 FPGA Logic BIST Configuration Comparison

FPGA	Number of BUT Configurations
ORCA 2C	9
ORCA 2CA	14
Xilinx 4000	12
Xilinx Spartan	12
Atmel AT40K	4
Atmel AT94K	4

CHAPTER FOUR

ROUTING BIST

The adaptation of FPGA routing BIST techniques will be discussed in their application to the Atmel AT94K series FPGA core. The fault models used in testing the routing resources of the FPGA will be discussed. The routing architecture of the Atmel FPGA poses new challenges in the application of BIST, which will be discussed along with proposed solutions. The goal of the development effort described in this chapter was to completely test all global routing resources and those local routing resources not completely tested during logic BIST. The routing BIST configurations are presented along with the evaluation of the faults detected by these configurations. Finally, the number of routing BIST configurations is compared to those previously developed for ORCA and Xilinx FPGAs.

4.1 Fault Models for FPGA Routing Resources

There are three basic types of fault models that are typically considered when modeling faults in the routing resources of an FPGA [11]. These faults include PIPs (cross-point PIPs, break-point PIPs, or MUX PIPs) stuck-closed (stuck-on) and stuck-open (stuck-off), wire segments stuck-at-0 and stuck-at-1, and, wires open and shorted wires (bridging faults) [11]. BIST architectures for testing the routing resources have been developed using these fault models to test for these various types of faults that may occur. In order to test the FPGA routing resources for some of these faults the applied test must ensure that every PIP and wire segment can transmit both a '0' and a '1' [11]. This will detect any stuck-open (stuck-off) fault in any closed PIP along the wire segment as well as any open or stuck-at fault affecting the wire segment [11]. In order to detect a PIP that is stuck-closed (stuck-on) opposite logic values must be applied to the wire segments associated with the PIP with both wire segments monitored by ORAs such that a fault will result in an incorrect logic value on one of the wire segments [11]. As a result, both combinations of logic values (1,0 and 0,1) must be applied during the test to the two wire segments separated by the open PIP [11]. For the case of wire segments that may have bridging faults associated with them, the same case holds true. During the test, opposite logic values must be applied to the wire segments while the wire segments are monitored by ORAs so that a fault will result in an incorrect logic value on one of the two wire segments [11]. A MUX PIP requires one test configuration for each of its inputs where both a '0' and a '1' must be applied to each input while the opposite logic value is applied to the remaining unselected inputs [11]. By applying this test, both a stuck-open fault in the closed PIP connecting the selected input wire segment to the output wire segment as well as any stuck-closed fault in the unselected (open) PIPs is detected [11]. In the case of non-decoded MUX PIPs, only one unselected input needs to be tested for stuck-on during each stuck-off test of a selected input.

4.2 Modifications to Routing BIST Methodology

In the previous applications of FPGA routing BIST to the ORCA [9] and Xilinx [10] FPGAs, a comparison-based routing BIST approach has been used. In this particular case, only two wires and associated PIPs can be observed in the ORAs. For the case of the Atmel FPGA, it is necessary to observe more than two wires in an ORA in order to minimize the number of configurations required to test the programmable routing resources. Therefore, the parity-based approach proposed in [23] was modified for application to the global routing resources in the Atmel FPGA. The approach in [23] for the testing of FPGA interconnect involved utilization of a TPG generating parity, a set of WUTs, other assumed fault-free routing resources, and a parity-based ORA. In this case, the TPG would source the test patterns over sets of WUTs to the ORA and, over other routing resources assumed to be fault-free, a parity bit was also routed to the ORA. This approach was modified to incorporate the parity bit as part of the test patterns being sourced over the WUTs and observed in the ORAs, which is illustrated in Figure 4.1.

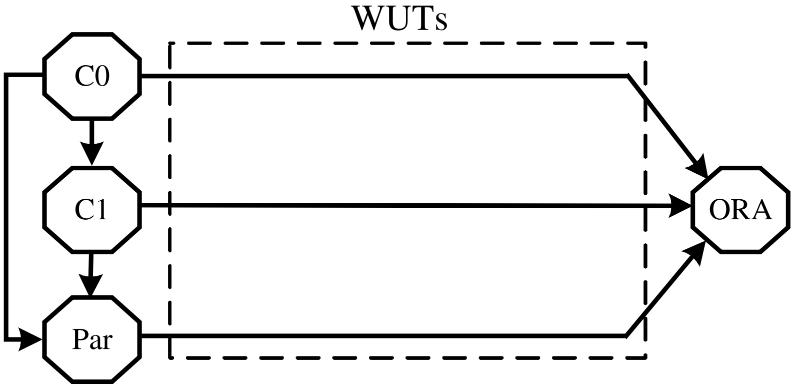


Figure 4.1 Routing BIST Architecture

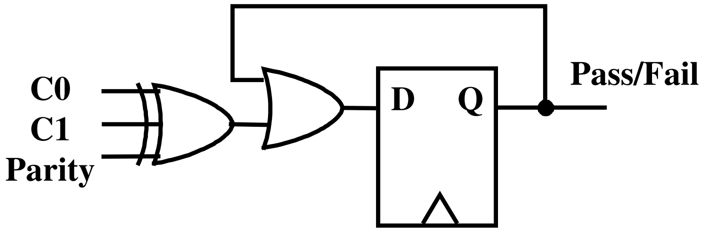
In this architecture, the TPG comprises a two-bit binary counter along with a parity bit generated over the two-bit count. The WUTs are a selected subset of routing resources, and the ORA checks for parity across the test patterns. Each of the sets of busses, both *x8* lines and the *x4* line, contain five wire segments in both vertical and horizontal directions. To apply this to the global routing resources, the parity bit is routed on the middle bus (third bus) of each set, while each of the two bits of the binary count are routed on two of the remaining four busses. These two bits are routed onto one bus on either side of the parity bit (i.e. count bit C0 is routed onto the first and fourth busses and C1 on the remaining busses). By routing the binary count and parity bits in this particular manner, any two of the wire segments in a set of busses are guaranteed to have opposite logic values (0,1 and 1,0) during the test sequence in order to detect faults present in the routing resources. In order to accommodate opposite logic values for particular tests associated with the cross-point PIPs, wire segments, and repeaters, alternating TPGs and ORAs for binary up-count with even parity and for binary down-count with odd parity are employed. This is demonstrated in Table 4.1, which shows the count sequences of both types of TPGs. Note that for the TPGs to attain opposite logic values between their test patterns, the binary up counter must be reset initially and the binary down counter must be preset initially.

Table 4.1 Test Pattern Sequences for Routing BIST

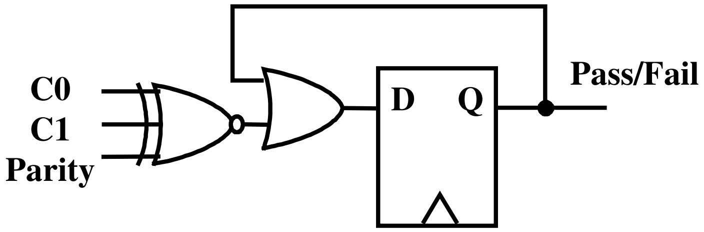
Up-count with Even Parity (C1, C0, Parity)	Down-count with Odd Parity (C1, C0, Parity)
000	111
011	100
101	010
110	001

As is illustrated in Table 4.1, between the bits of the test patterns for each TPG and parity combination, opposite logic values are present between any two of the bits in the sequence. In addition, between the respective bits of the two test patterns, there are opposite logic values. This allows the necessary conditions to be met to detect faults associated with the fault models used for the routing BIST development. This pattern can be utilized to apply both logic values to every wire segment and PIP thus testing for any stuck-off fault in any closed PIP in a set of WUTs as well as for any open or stuck-at fault affecting the wire segments in a set of WUTs. The presence of opposite logic values in the patterns can be used to test for stuck-on faults associated with PIPs in a set of WUTs in addition to testing the non-decoded MUX PIPs in the repeaters. The only exception is in the case of the **L** output configurations, which use a similar architecture and the same type TPG and ORA as those used in the logic BIST test sessions.

A TPG sourcing the two-bit binary count value and generating a parity bit across the 2-bit count was ideal for the PLB architecture. Due to the small size of the PLB, the number of inputs that can be checked for parity in a given ORA while latching an error is a maximum of three. This restriction is a result of the number of bits present in the LUTs as well as the organization of the PLB, which is shown in Figure 2.3. The LUTs in the PLB contain eight bits each and only four inputs can be routed into the PLB, which allow for up to a 4-input function to be obtained by combining the LUTs through the use of the multiplexers located just below the LUTs in Figure 2.4. In order to implement an ORA with feedback to latch errors, however, a maximum of three inputs can be checked for parity, which is illustrated in Figure 4.2. Thus, a TPG was chosen to source three test patterns, including the parity bit, over the sets of WUTs to test the programmable interconnect in the FPGA. The exclusive-OR (or exclusive-NOR) gate at the front end checks for even (or odd) parity across the test pattern and the OR gate serves to latch up an error within the flip-flop if an error in parity is detected (given by the output of the exclusive-OR or exclusive-NOR gate).



a) ORA for Up Count & Even Parity



b) ORA for Down Count & Odd Parity

Figure 4.2 ORA Structure for Routing BIST

In order to implement this ORA in the FPGA, a C program was utilized to perform bit manipulation on the download bitstream files. This was necessary due to the available dynamic macros as discussed in Section 2.6.1. A function can be implemented in the PLB with a parity-check of three inputs and feedback to latch any errors using an FGEN2 type dynamic macro. However, a Shift signal must be routed to all ORAs in order to shift out the BIST results at the end of a routing BIST phase, as was the case in logic BIST. This signal can not be routed to the ORAs unless specified as an input to the PLB in the MGL program. Thus, a dynamic macro with four inputs, the FGEN1 type dynamic macro, is necessary in order to route the Shift signal to the ORAs. Utilizing a dynamic macro with four inputs does not allow for the ORA structure to be implemented through MGL, therefore, a C program was generated to perform bit manipulation of the download bitstream files. In doing so, the fourth input to the ORAs needed for the Shift signal can be ignored during the execution of a routing BIST phase even though the signal is routed through the global routing resources and available to the PLB. Upon execution of a given routing BIST phase, the ORA must be dynamically reconfigured, utilizing the synchronous RAM mode of configuration for the FPGA or through partial dynamic reconfiguration from the AVR, to utilize the Shift signal in order to shift out the BIST results, as illustrated in Figure 3.4d.

4.3 Overview of Routing BIST Configurations

The routing BIST configurations for the programmable routing resources consist of four test sessions targeting four different types of the routing resources. These tests target the cross-point PIPs and the repeaters in the global routing resources in addition to the PLB **L** output and tri-state buffer and **X** direct connections left untested after completion of the logic BIST sessions. There is a total of 48 routing BIST configurations which include 16 configurations for cross-point type PIPs, 24 configurations for vertical and horizontal repeaters, 4 configurations for the **L** output and tri-state buffer, and 4 configurations for the **X** direct connections. All of these configurations, with the exception of the **L** output configurations, utilize the TPG and ORA discussed in section 4.2, differing only by the subset of routing targeted and the particular architecture during a given routing BIST configuration. The **L** output configurations use a similar architecture to that used in the logic BIST test sessions.

These configurations also require that additional pins be utilized for the Shift input signal and the Pass/Fail output signal. This is due to the use of busses during particular orientations of configurations. Only particular *x8* and *x4* lines connect to the I/O cells through repeaters at the edge of the array, and, therefore, if these lines are utilized as a set of WUTs, a signal cannot be routed to an I/O cell through these particular lines. This was overcome by selecting pins for the Shift and Pass/Fail signals on different sides of the array in order to avoid conflict with the sets of WUTs during the vertical and horizontal orientations of the routing BIST test sessions. In order to accomplish this, two pins each for both the Shift and Pass/Fail signals were chosen in order to be used in the different routing BIST configurations, as summarized in Table 4.2. Note that the signals denoted with a '1' have the same pin numbers as used in logic BIST.

Table 4.2 Pin Numbers for I/O Cells for Routing BIST

Signal	84 Pin AJC Package	144 Pin BQC Package	208 Pin DQC Package
Clock	13	2	4
Shift 1	81	121	174
Shift 2	17	11	17
Pass/Fail 1	23	19	27
Pass/Fail 2	41	53	77

The faults left undetected after completion of logic BIST include stuck-on and stuck-off faults in the MUX PIPs in the **X** direct connections from adjacent PLBs, stuck-on faults in the local routing cross-point PIPs, and stuck-at faults in the MUX PIP and the tri-state buffer associated with the **L** output of the PLB. Some, but not all, of the **X** direct connections are tested during logic BIST due to the applied logic values to the **X** inputs of the ORAs. The MUX PIPs that are tested are selected as inputs to the ORA during logic BIST while the untested MUX PIPs are not selected as inputs to the ORA during logic BIST. The local routing cross-point PIPs are left incompletely tested since opposite logic values are not applied to the vertical (V1-V5) and horizontal (H1-H5) during logic BIST in order to test the PIPs for stuck-on faults. The MUX PIP associated with the **L** output of the PLB is partially tested during logic BIST configurations that utilize feedback since one of its inputs is combinational and the other is sequential, which presents a clock-cycle delay between the two inputs and allows opposite logic values during a part of the test sequence being applied. It is only partially tested since the faulty circuit value must be propagated back through the LUTs as feedback. The tri-state buffer is not selected during logic BIST and is left completely untested. Each of the sets of routing BIST configurations that target the various types of routing resources is discussed in the following sections.

4.4 BIST Configurations for Cross-Point PIPs

The cross-point PIPs present in the global routing resources, illustrated in Figure 4.3, requires a total of 16 routing BIST configurations. These 16 configurations break down into two sets of eight configurations with each set targeting a particular set of *x8* lines, either the Abus or Ebus lines, on which the cross-point makes or breaks a connection. These routing BIST configurations target stuck-off and stuck-on faults in the global routing cross-point PIPs, opens and stuck-at faults in the *x8* lines, and the remaining stuck-on faults in the local routing cross-point PIPs at the inputs to the PLBs left untested during the application of logic BIST. These configurations are generated from separate set of MGL and C programs, one set of programs for the Abus line cross-point PIPs and one set for the Ebus line cross-point PIPs.

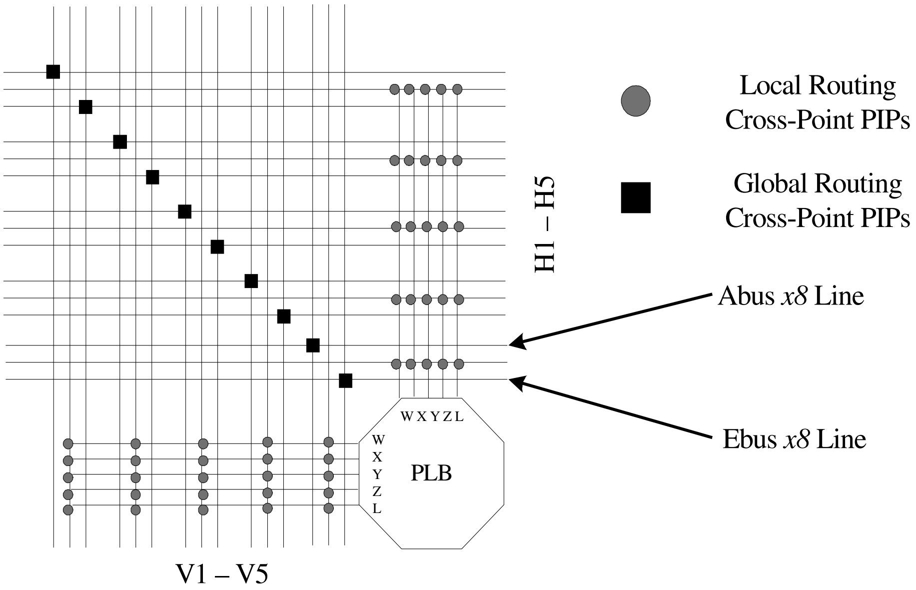


Figure 4.3 Global Routing Associated with the PLB

The basic architecture of the cross-point PIP BIST configurations is a STAR which consists of an *8x8* array of PLBs with each STAR separated by repeaters. The STARs are tiled across the FPGA array such that concurrent testing is performed on all the selected cross-point PIPs in the array, as illustrated in Figure 4.3. Each STAR has either a TPG functioning as a two-bit up counter with even parity generation and the corresponding even parity check ORA or has a TPG functioning as a two-bit down counter with odd parity generation and the corresponding odd parity check ORA. These two types of STARs are tiled across the array in a checkerboard fashion. This allows opposite logic values to be sourced into a given STAR from an adjacent STAR to test the global routing cross-point PIPs shown for stuck-off faults as well as for testing the local routing cross-point PIPs at the inputs to the PLBs for stuck-on faults, which are both illustrated in Figure 4.4. Since the configuration is made up of *8x8* STARs, there are eight phases per cross-point PIP test session. Each routing BIST phase shifts the TPG down by one row while every other phase the ORAs are shifted up by two columns such that, in the last phase, the TPG and ORAs are at the opposite end of the STAR from their positions in the first phase. In each phase the connections to the two rows of ORAs are swapped. This allows all the local routing cross-point PIPs at the ORAs to be tested for stuck-on faults since the inputs to the ORAs enter through vertical local routing cross-point PIPs and the opposite logic values from an adjacent STAR are routed onto the corresponding busses associated with the horizontal local routing cross-point PIPs above each row of ORAs.

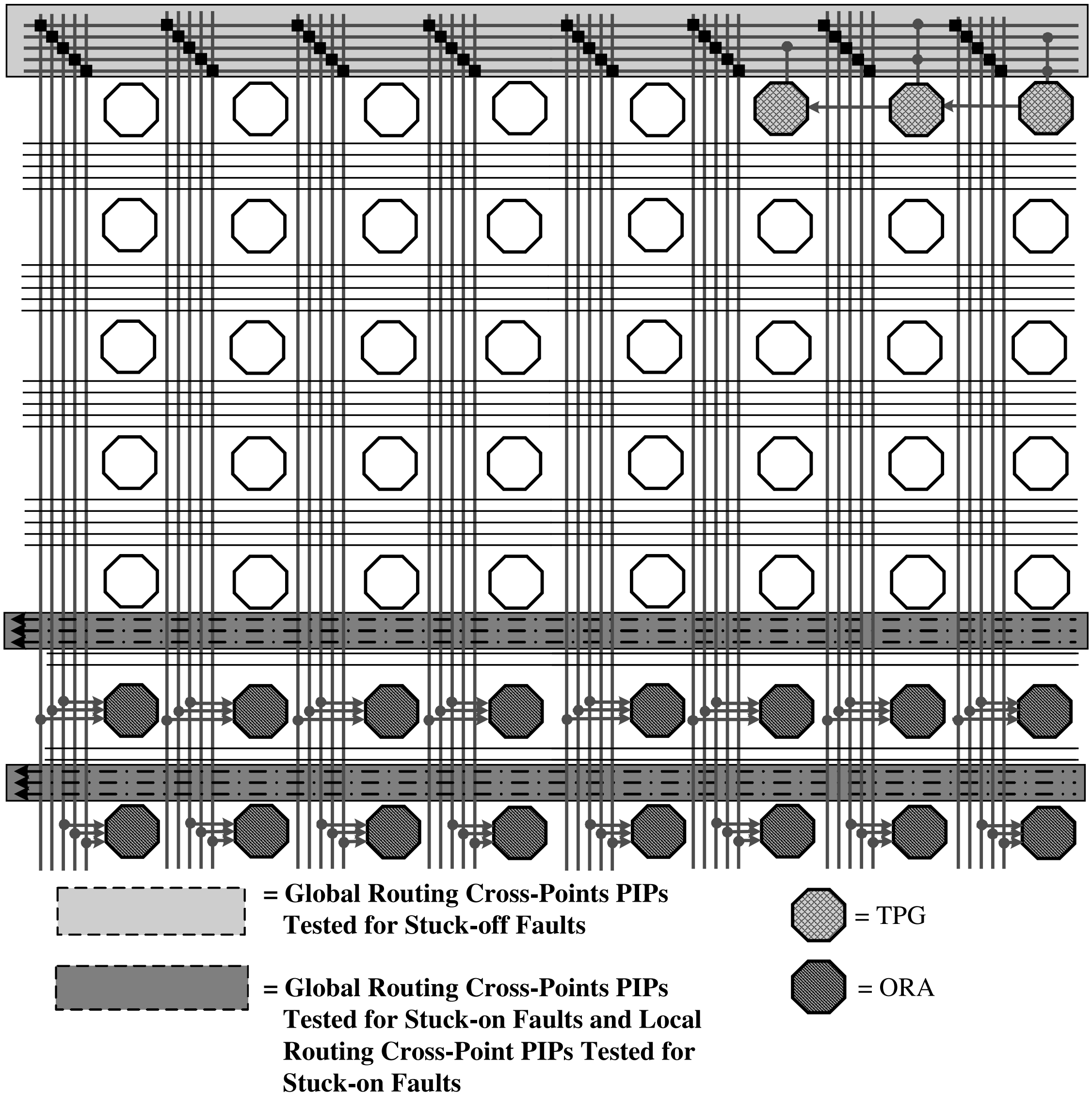


Figure 4.4 Cross-Point PIP Routing BIST Architecture

Since the cross-point PIPs make or break connections between vertical and horizontal wire segments, there is no difference in the testing of cross-point PIPs when orienting the BIST architecture horizontally (as in Figure 4.4) or vertically (can be derived by rotating Figure 4.4 counter-clockwise 90^o). However, by orienting the architecture both vertically and horizontally during different BIST configurations, all the local routing cross-point PIP stuck-on faults at the inputs to the PLBs can be tested. Therefore, the set of configurations for the cross-point PIPs on the Abus lines was oriented vertically while the Ebus line cross-point PIP set of BIST configurations was oriented horizontally. These configurations facilitate testing of the cross-point PIPs denoted in Figure 4.4 for stuck-off faults since both a logic '0' and a logic '1' are passed through the PIPs. The *x4* lines are tested for shorts between the five wire segments in the set of *x4* lines since opposite logic values are guaranteed between every pair of wire segments during the test pattern. The vertical and horizontal *x4* lines are both tested since the test is oriented both vertically and horizontally and the x4 lines have to be used to enter into the ORAs from the global routing. The same case holds true for the Abus lines and Ebus lines. The five wire segments in the set of Abus or Ebus lines are guaranteed to have both opposite logic values between any pair of wire segments. For the Abus lines, the repeaters are staggered evenly across the array, meaning that the boundaries of the repeaters selecting from the Abus *x8* lines and the *x4* lines matches up with the boundaries of the FPGA array, as is illustrated in Figure 4.5. This is true because all the array sizes are multiples of eight: 16, 24, 32, and 48.

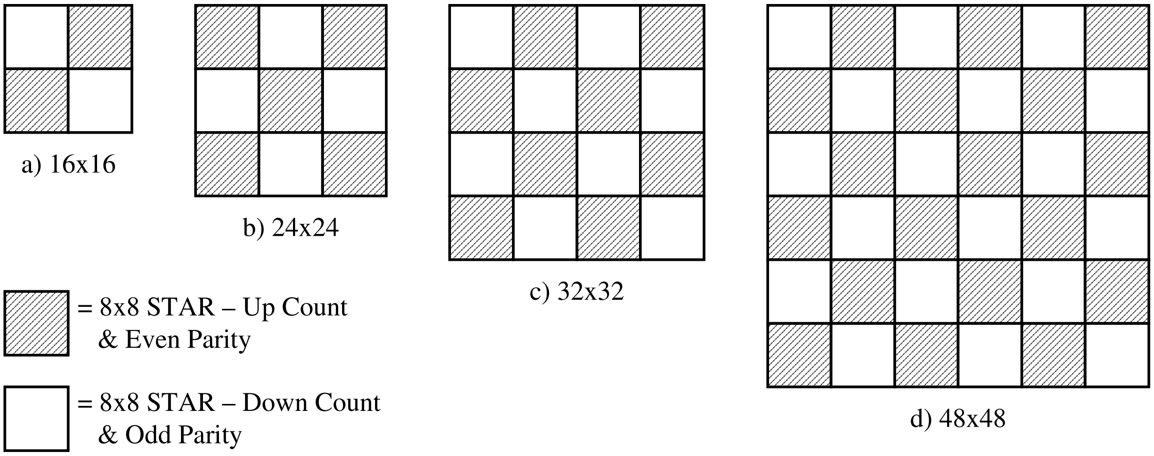

a) 16x16
b) 24x24
c) 32x32
d) 48x48

Figure 4.5 Abus Cross-Point STAR Tiles for all FPGA Array Sizes

This is not the case for the repeaters selecting from the Ebus *x8* lines and the *x4* lines since these Ebus *x8* lines start with an offset of four PLBs from either edge of the array. Due to this staggering of repeaters, the cross-point PIP test session for the Ebus lines have not only 8x8 STARs but also have 4x4, 4x8, and 8x4 STARs, which are shown in Figure 4.6. In addition to having different STAR sizes, the checkerboard tiling of the STARs for the Ebus cross-point configurations is different in the 24x24 array due to the offset of the Ebus repeaters from the edges and the arrangement of STARs that form the checkerboard pattern, which is illustrated in Figure 4.6b. For the Ebus line cross-point PIP configurations, the MGL program is much more complex than that for the Abus line cross-point PIP configurations.

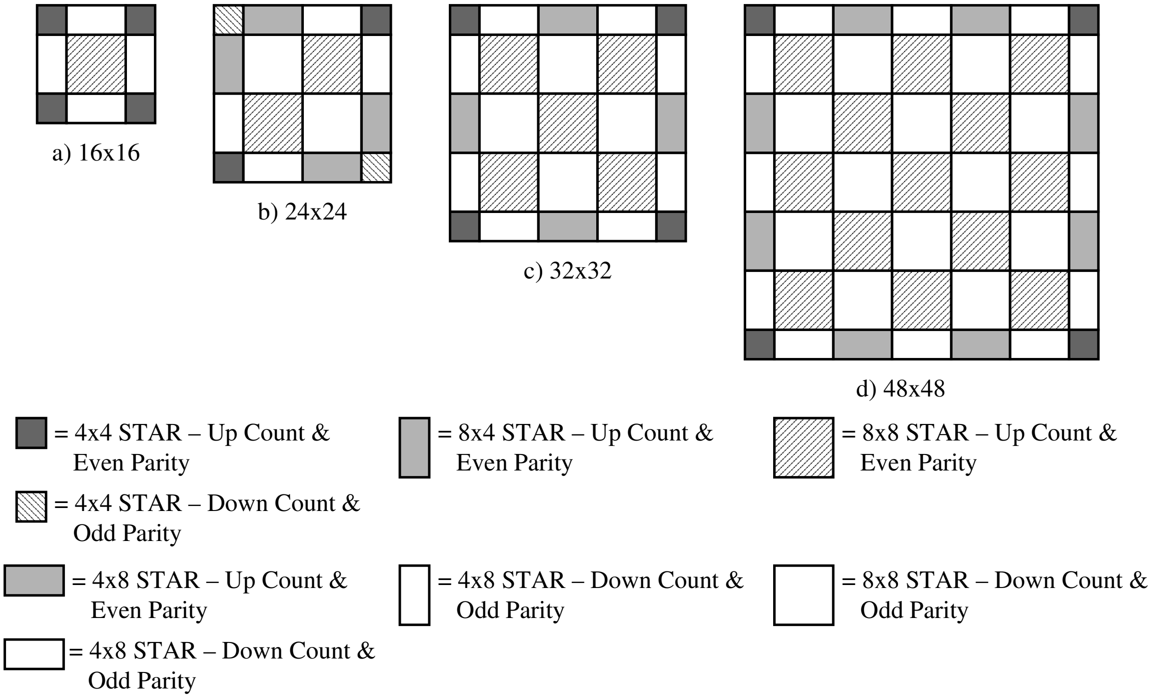
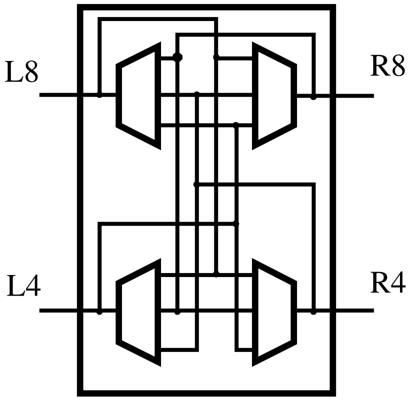


Figure 4.6 Ebus Cross-Point STAR Tiles for all FPGA Array Sizes

4.5 BIST Configurations for Repeaters

The repeaters that are disbursed within the global routing resources are shown in Figure 4.7a. These repeaters required multiple test phases to completely test for all possible combinations of stuck-on and stuck-off faults that may exist in the MUX PIPs within the repeater. In addition, the opens and stuck-at faults are further tested for the Abus *x8* lines as well as for the Ebus *x8* lines. Since the repeaters are dispersed symmetrically both horizontally and vertically, tests were derived that test both types of repeaters. As in the cross-point PIP test sessions, the staggering of the repeaters throughout the array has several implications on the application of BIST.

Three sets of BIST configurations were developed to test the MUX PIPs in the repeaters for both stuck-on and stuck-off type faults. These three sets of configurations were developed for both the Abus line and Ebus line repeaters. The faults targeted during each set of configurations are given in Figure 4.7. Figure 4.7a shows the possible connections that can be made through the MUX PIPs in a repeater and Figure 4.7b lists the faults that are targeted during a given set of configurations. Since the repeater structure allows connections to be made in both directions (i.e. from L8 to R8 or from R8 to L8), for a given set of configurations the repeaters must be tested in both directions.



a) Repeater Connections

	R8 Input		L8 Input		R4 Input		L4 Input	
MUX PIP	S-On	S-Off	S-On	S-Off	S-On	S-Off	S-On	S-Off
R8			2	3	2	1	1	2
L8	2	3			1	2	2	1
R4	2	1	1	2			2	3
L4	1	2	2	1	2	3		

b) Stuck-on and Stuck-off Faults in MUX PIPs Targeted in a Given Set of Configurations

Figure 4.7 Repeater Connections and Targeted Fault Types

For each type of repeater, there are three sets of configurations each of which has orientation horizontally and vertically and tests both directions of the connections through the repeaters. Therefore, there are four test phases per set of configurations: two horizontal configurations, which test both directions through horizontal repeaters, and two vertical configurations, which test both directions through vertical repeaters. This makes a total of 12 BIST configurations for each set of repeaters associated with the Abus *x8* line and the Ebus *x*8 line.

The first set of *x8* line repeater configurations targets only the MUX PIPs that make connections between wire segments R8-L4, R8-R4, R4-L8, and L4-R8 for stuck-on faults and the MUX PIPs that make connections between wire segments R8-R4, L8-L4, R4-R8, and L4-L8 for stuck-off faults, as given in Figure 4.7b. This is done by creating STARs that alternate between TPGs with up-count with even parity and with down-count with odd parity, as illustrated in Figure 4.6. Connections are formed that loop around through the repeaters such that connections made on each side of the repeater have opposite logic values during the test sequence, thus testing for the desired set of faults in the MUX PIPs. For example, the MUX PIP for the L8 output receives an input signal from the L4 input while the opposite logic value is applied to the MUX PIP's unselected input coming from R8, facilitating a test of the L8 MUX PIP according to the fault models being used. The orientation shown is for horizontal STARs; the vertical orientation can be derived by rotating the figure counter-clockwise by 90^o since the routing architecture is rotationally symmetric. The repeaters form loop-around connections between the *x4* line and *x8* line or vice-versa to make connections between the TPGs and ORAs. Swapping the locations of the TPGs and ORAs facilitates a test of the opposite direction through the repeaters.

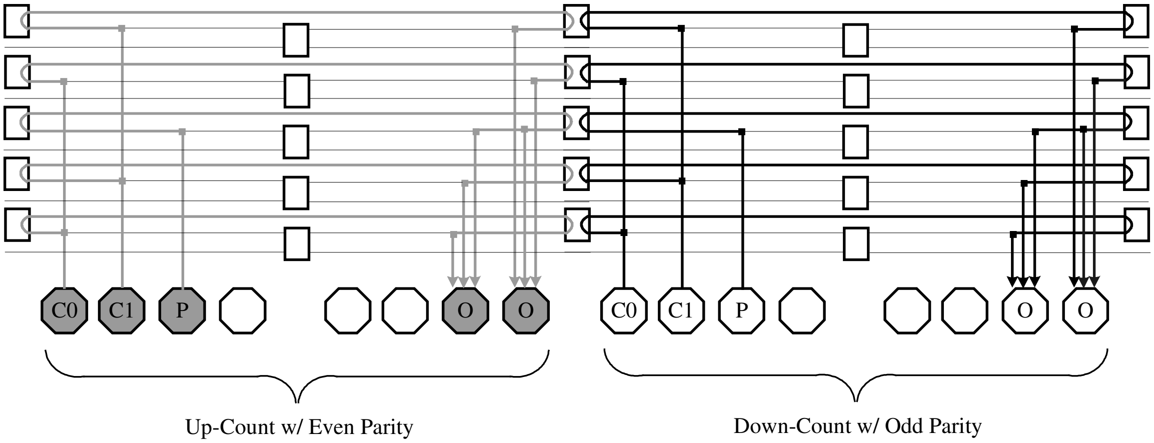


Figure 4.8 Repeater Set 1 Configuration Architecture

In order to test both directions through the repeaters, the architecture shown in Figure 4.6 is flipped such that the respective TPGs and ORAs swap positions and the signals from the TPGs are driven in the opposite direction. The STARs consist of *1x8* arrays of PLBs and are tiled across the array to form the BIST architecture. Figure 4.8 gives the horizontal orientation; the vertical orientation can be derived by rotating the figures counter-clockwise by 90^o (which gives *8x1* STARs).

The second set of *x8* line repeater BIST configurations targets connections between wire segments R8-L8, R8-R4, L8-R8, L8-L4, R4-R8, R4-L4, L4-L8, and L4-R4 for stuck-on faults and targets connections between wire segments R8-L4, L8-R4, R4-L8, and L4-R8 for stuck-off faults. The STARs in this case overlap, as can be seen in Figure 4.9 where all five x8 lines are being driven by the TPGs and observed by the ORAs. A STAR in the second set of configurations must be *1x16* to accommodate the diagonal connections through the repeaters needed to target the desired fault types. In this set of configurations opposite logic values are applied to the inputs on either side of the repeater, such that diagonal connections made in the MUX PIPs through the repeaters are tested. As an example, the MUX PIP for the R8 output is selecting the signal coming from the L4 input while having the opposite logic value applied to its MUX PIP for the L8 input. This set of configurations also tests the wire Abus lines and x4 lines as well as the Ebus lines and x4 lines for shorts between the wire segments. This is true because the respective wire segments are guaranteed to have opposite logic values between them during the test sequence. Not only are these wire segments tested for shorts between them, but they are also tested for stuck-at 0, stuck-at 1, and opens, since each wire segment is driven by a logic '0' and a logic '1' and observed in an ORA during the execution of the configuration. The only wire segments not completely tested are the Abus lines that are at the edges of the FPGA array (along the 8 PLBs from either edge). These wire segments do not have opposite logic values with respect to the *x4* lines during the test sequence. Therfore, they are tested only for stuck-at 0, stuck-at 1, and opens, with no testing done for bridging faults between these lines and the *x4* lines. Rotating the configuration counter-clockwise by 90^{o} gives the vertical orientation (which yields *16x1* STARs). The alternating TPGs, which have up-count with even parity or down-count with odd parity, are used in this architecture, as was the case in the architecture of the Set 1 configuration.

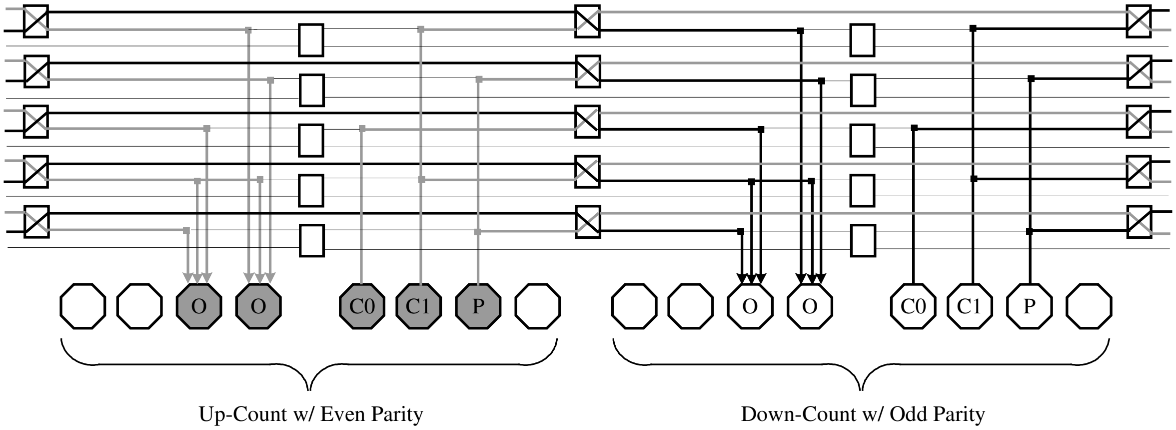


Figure 4.9 Repeater Set 2 Configuration Architecture

The third set of configurations for the *x8* line repeaters test the stuck-off faults on connections between wire segments R8-L8, L8-R8, R4-L4, and L4-R4, which connect straight through the repeaters between *x4* lines or between *x8* lines. The BIST architecture is shown in Figure 4.10 for this set of configurations, STARs of size *4xarray* (*arrayx4* for vertical sessions) are tiled in the array to produce a test of the straight-through connections (i.e. in the repeaters). In this configuration there is no need for alternating TPGs and ORAs since one TPG is driving an entire row (or column) of ORAs and since the repeaters are being targeted for stuck-off faults only. In order to test both directions of connections in the repeaters, the architecture is flipped such that the TPG moves to the opposite end of the array between a given horizontal or vertical orientation of the BIST architecture. As the case with the set 1 configurations, these do not test for any faults in the wire segments.

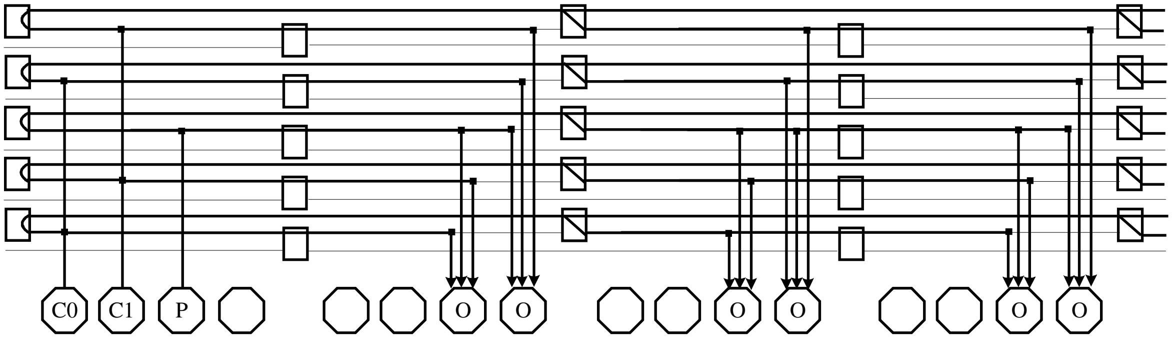
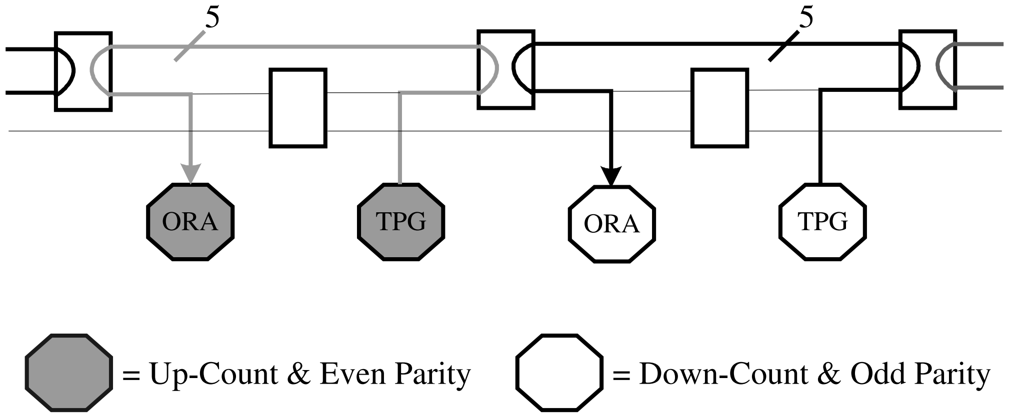


Figure 4.10 Repeater Set 3 Configuration Architecture

The three sets of configurations match well with Abus repeaters in the global routing resources; however, the case is much different with the Ebus repeaters. Since the Abus repeaters evenly divide busses in the routing resources into sections that span eight PLBs as well as line up with the boundaries of the array, the three sets of configurations tile uniformly into any size FPGA array. The case for the Ebus repeaters is much different. These repeaters do not divide the busses in the routing resources into even sections spanning eight PLBs and do no match up with the boundaries of the array. Instead, the boundary of the Ebus repeaters lies four PLBs from the edges of the array, such that, inside the boundaries, the busses span eight PLBs before reaching an Ebus repeater, and, outside the boundaries, busses span only four PLBs before reaching an Ebus repeater.

The Set 1 configurations apply uniformly to the Abus repeaters, which is illustrated by Figure 4.11. The repeaters in the middle of the array are completely tested for the targeted faults. The exceptions lie at the edges of the array where test patterns are not supplied to the loop-back connections on the outer side of the repeaters. This means that only stuck-off faults are tested in connections between wire segments R8-R4 and R4-R8 or in connections between wire segments L8-L4 and L4-L8, depending on the side of the array that the repeater is located. However, these repeaters make connections to I/O cells and can be tested during BIST configurations developed for the I/O cells.

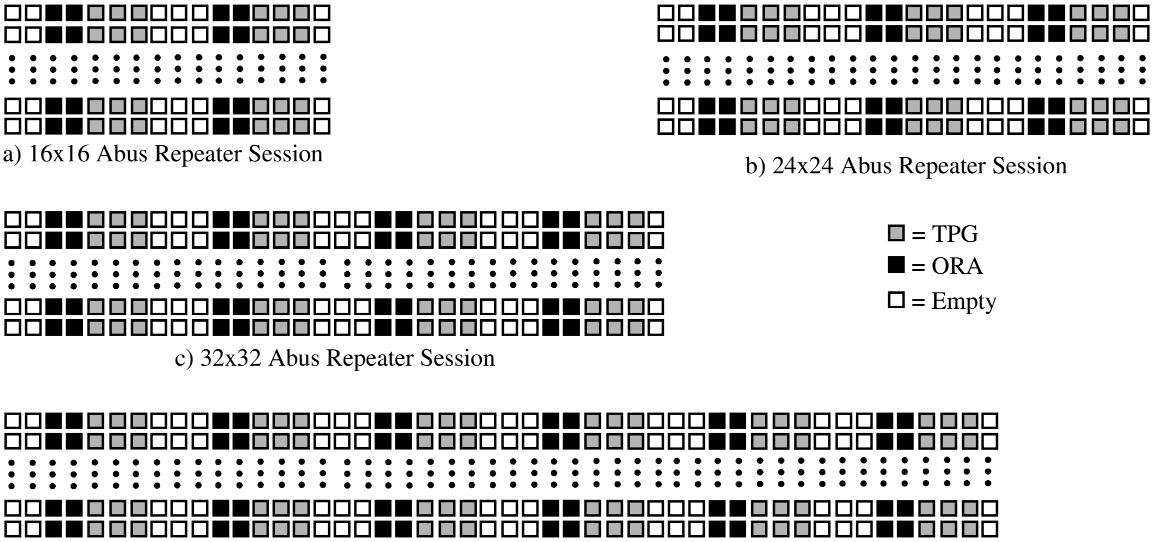


*TPGs = 3 PLBs & ORAs = 2 PLBs

Figure 4.11 Abus Repeater Set 1 Configuration Architecture

The implementation of the Abus repeater Set 1 configuration into the array sizes is given in Figures 4.12 and 4.13. Figure 4.12 shows the BIST architecture in its implementation to test one direction of connections through the repeater. Figure 4.13

shows the BIST architecture having been flipped to make connections through the repeaters in the opposite direction as that of Figure 4.12. As illustrated in these figures, the implementation of the configurations is very uniform in its application to any of the array sizes in both its normal and flipped connections for the repeaters. Although the horizontal direction is shown, the vertical implementation has the same architecture and can be derived by rotating Figures 4.12 and 4.13 counter-clockwise by 90°. This is possible due to the rotational symmetry of the programmable routing resources in the array.



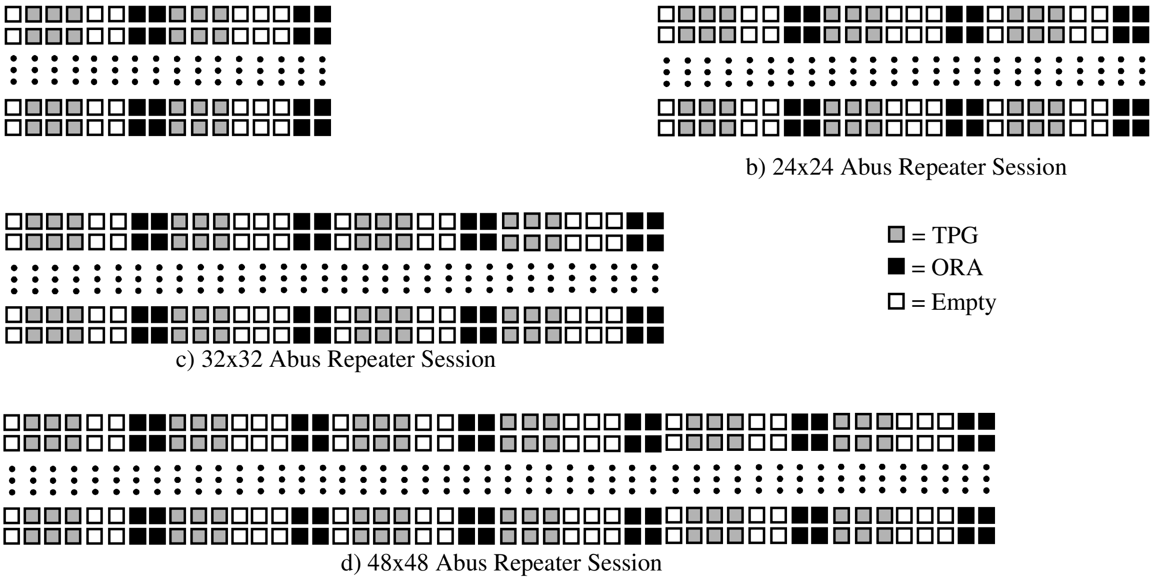
a) 16x16 Abus Repeater Session

b) 24x24 Abus Repeater Session

c) 32x32 Abus Repeater Session

d) 48x48 Abus Repeater Session

Figure 4.12 Abus Set 1 Configurations for All Array Sizes



b) 24x24 Abus Repeater Session

c) 32x32 Abus Repeater Session

d) 48x48 Abus Repeater Session

Figure 4.13 Flipped Abus Set 1 Configurations for All Array Sizes

The repeaters staggered along the Ebus lines and *x4* lines do not match up with the boundaries of the FPGA array. Instead, the boundaries for these repeaters fall four PLBs from either edge of the array. This makes the generation of routing BIST configurations for these repeaters more difficult than for the Abus line repeaters discussed previously.

As can be seen in Figure 4.14, which illustrates the architecture of the Set 1 Ebus line repeater configurations, the misalignment of the boundaries Ebus repeaters with the boundaries of the FPGA array has an effect on the implementation of the BIST configurations. Since the Ebus line repeater boundary is four PLBs from the edge, a scheme was devised to compensate for the mismatch. STARs of size *1x16* (*16x1* for vertical sessions) are constructed in the middle of the array to allow for the targeted faults to be tested in the Ebus line repeaters. At the four PLBs along the edges of the array (east and west sides for horizontal configurations and north and south for vertical configurations), conflicts arise in making the desired connections. A scheme was devised to allow the desired connections through the MUX PIPs in the repeaters to be made.

However, this scheme imposes constraints on the diagnostic resolution at the edges, which limits the diagnosis of a detected fault to one of two repeaters since, between any TPG and ORA, there are two repeaters through which the test patterns must travel. However, this scheme is the most efficient method to implement a test of the loop-back connections in the Ebus line repeaters. In addition, this scheme does not suffer from the problem of incomplete testing at the repeater boundaries as seen in the Set 1 configurations for the Abus repeaters.

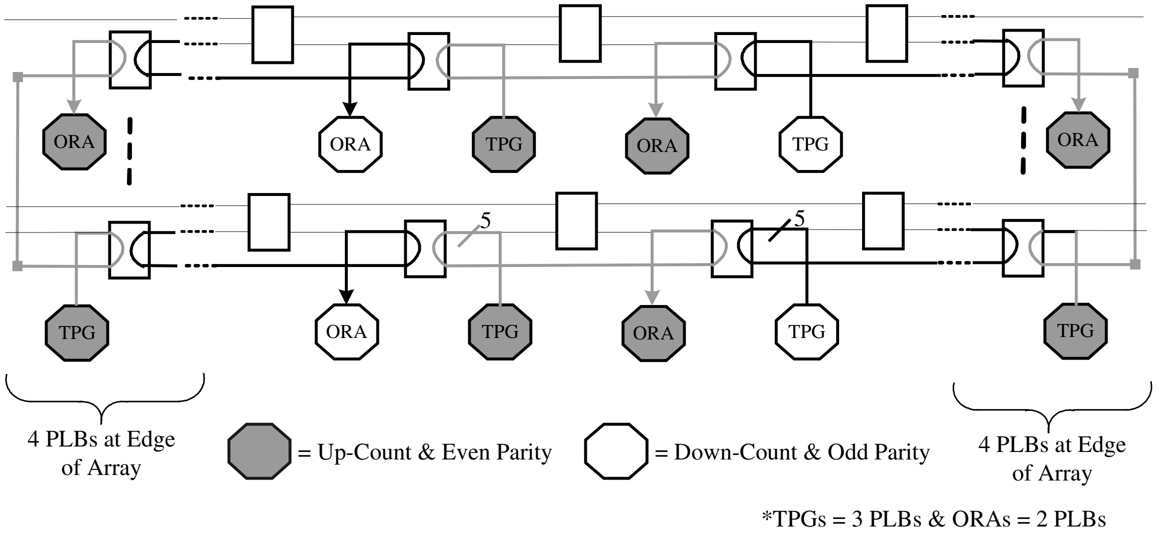
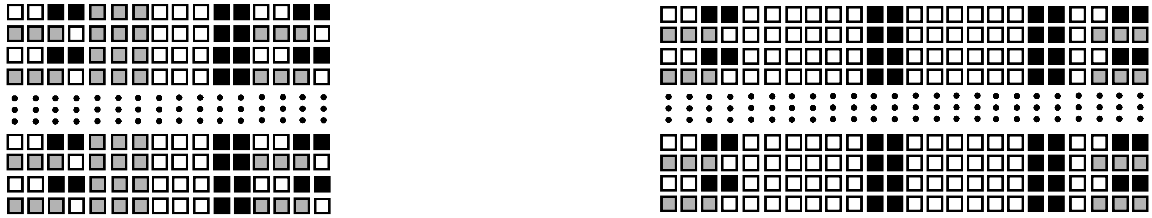


Figure 4.14 Ebus Repeater Set 1 Configuration Architecture

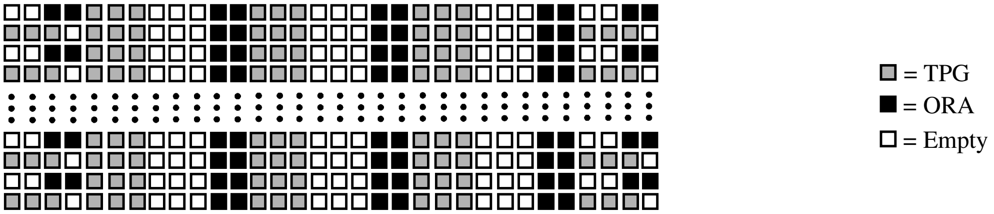
The Set 1 Ebus configurations are shown in Figures 4.15 and 4.16 as they tile into the array sizes for both the normal and flipped architecture, respectively, for testing both directions of connections through the MUX PIPs in the repeaters. These configurations are much less regular and less structured than those configurations for the Abus line repeaters. This is due to the mismatch of boundaries between the Ebus line repeaters and the FPGA boundaries and the compensation made in the routing BIST architecture to accommodate this difference.

The Ebus line repeaters do maintain rotational symmetry, however, such that the configurations shown can be rotated counterclockwise by 90^{o} to derive the vertically oriented Set 1 Ebus line repeater configurations.

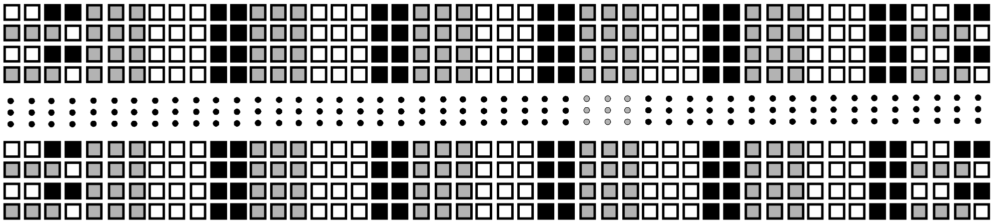


a) 16x16 Ebus Repeater Session

b) 24x24 Ebus Repeater Session

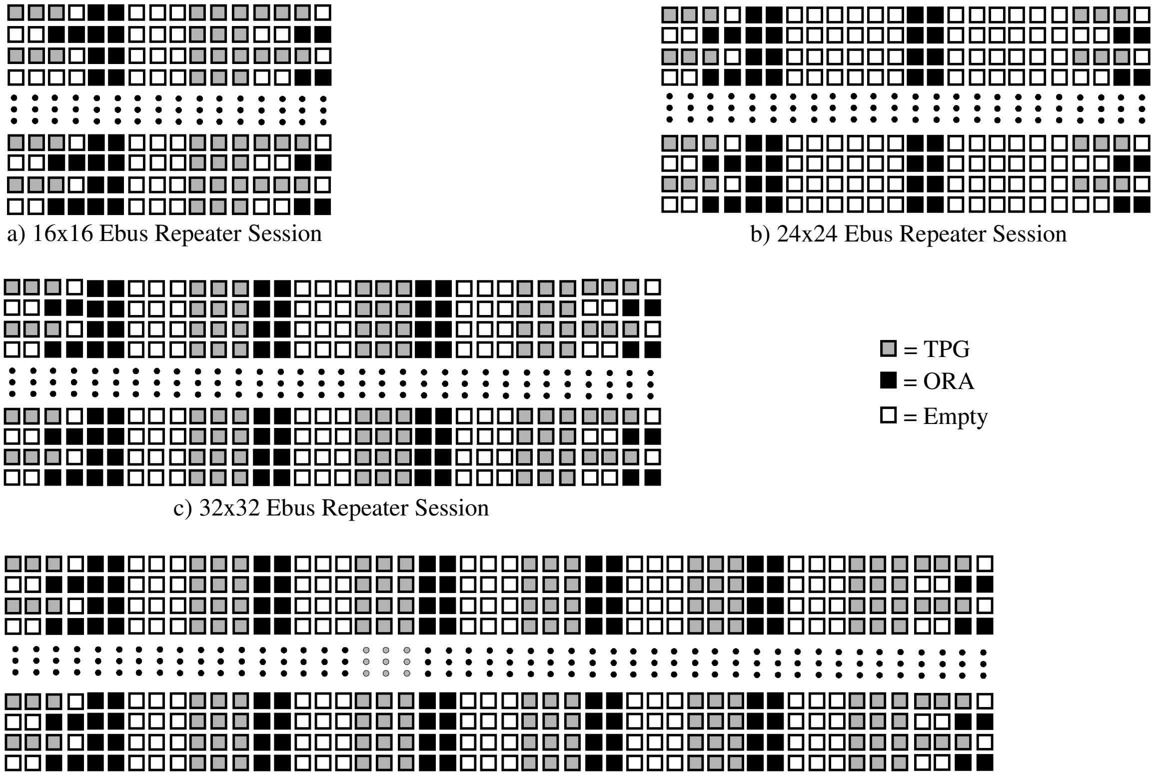


c) 32x32 Ebus Repeater Session



d) 48x48 Ebus Repeater Session

Figure 4.15 Ebus Set 1 Configurations for Each Array Size



a) 16x16 Ebus Repeater Session

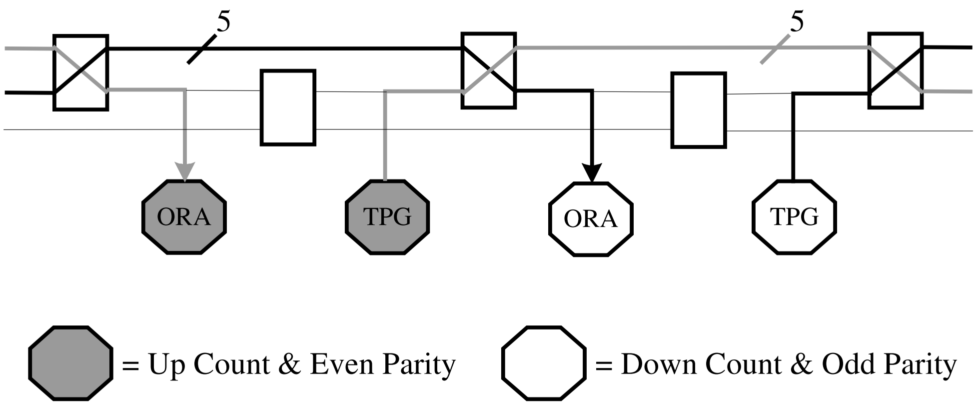
b) 24x24 Ebus Repeater Session

c) 32x32 Ebus Repeater Session

d) 48x48 Ebus Repeater Session

Figure 4.16 Flipped Ebus Set 1 Configurations for Each Array Size

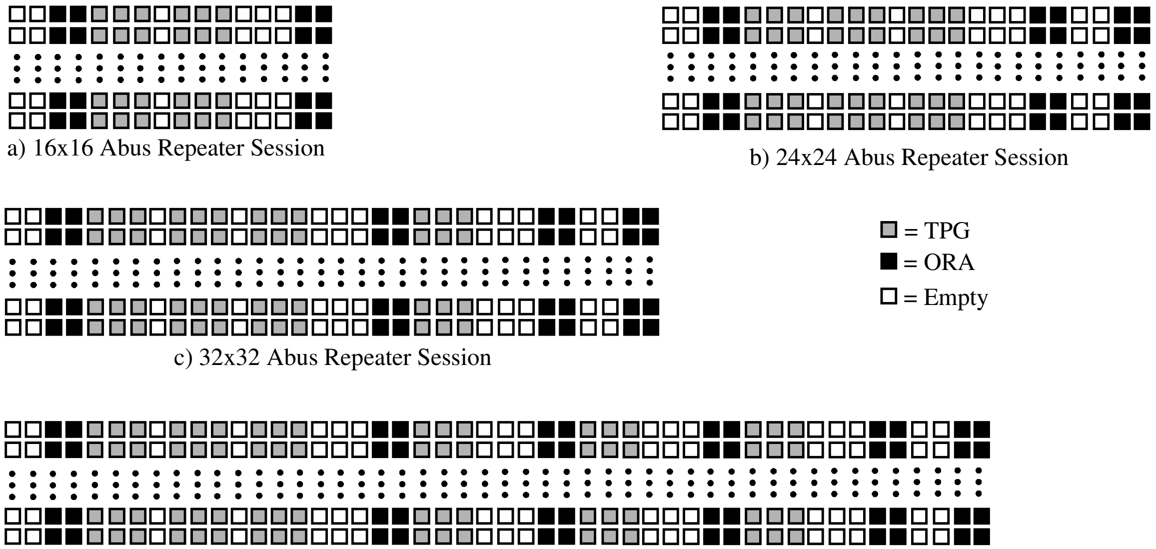
The architecture for the Set 2 configurations for the Abus repeaters is given in Figure 4.17. Although the Abus repeater boundaries line up with the array boundaries, the scheme for testing the diagonal-type connections made through the MUX PIPs in the repeaters does not match up well with the repeater boundaries, as is illustrated in the architecture of the Set 2 configuration given in Figure 4.17. These configurations face similar problems to those faced in the Set 1 configurations for the Abus repeaters. The repeaters at the edge of the array are incompletely tested. Actually, the repeaters at the edges are tested again for stuck-off faults in connections R8-R4 and R4-R8 or in connections L8-L4 and L4-L8, and the targeted faults for the Set 2 configurations are not tested in these repeaters. However, these can be tested through BIST configurations developed for the I/O cells dispersed around the array.



*TPGs = 3 PLBs & ORAs = 2 PLBs

Figure 4.17 Abus Repeater Set 2 Configuration Architecture

In this case, *1x16* STARs (*16x1* for vertical sessions) are shown tiled across the array in Figures 4.18 and 4.19, which show the normal and flipped placement of the TPGs and ORAs, respectively. Notice that in the 24x24 array, the STARs must be overlapped since the *1x16* STAR size does not fit exactly into the array. The tiles are not as uniform in the Set 2 configurations as in the Set 1 configurations. This is due to the diagonal connections through the repeaters, which result in the configuration not lining up exactly with the repeater boundaries in the array.



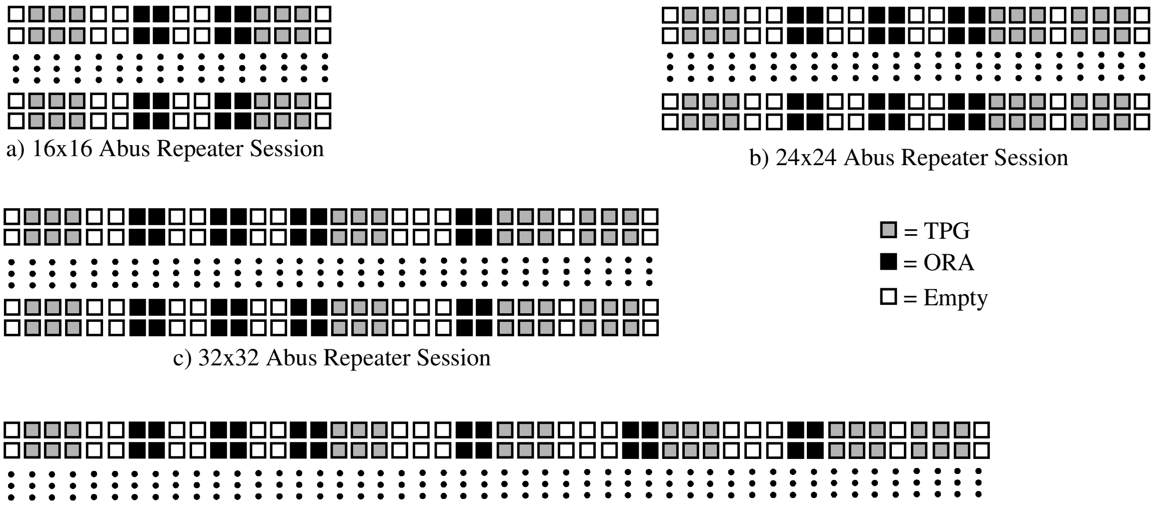
a) 16x16 Abus Repeater Session

b) 24x24 Abus Repeater Session

c) 32x32 Abus Repeater Session

d) 48x48 Abus Repeater Session

Figure 4.18 Abus Set 2 Configurations for Each Array Size


a) 16x16 Abus Repeater Session

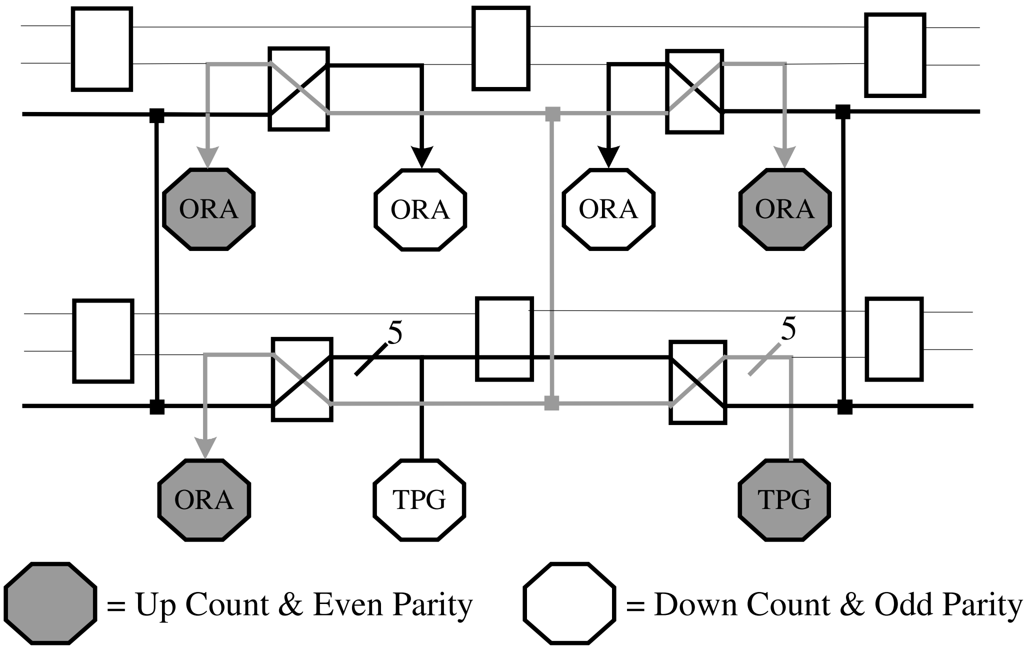
b) 24x24 Abus Repeater Session

c) 32x32 Abus Repeater Session

d) 48x48 Abus Repeater Session

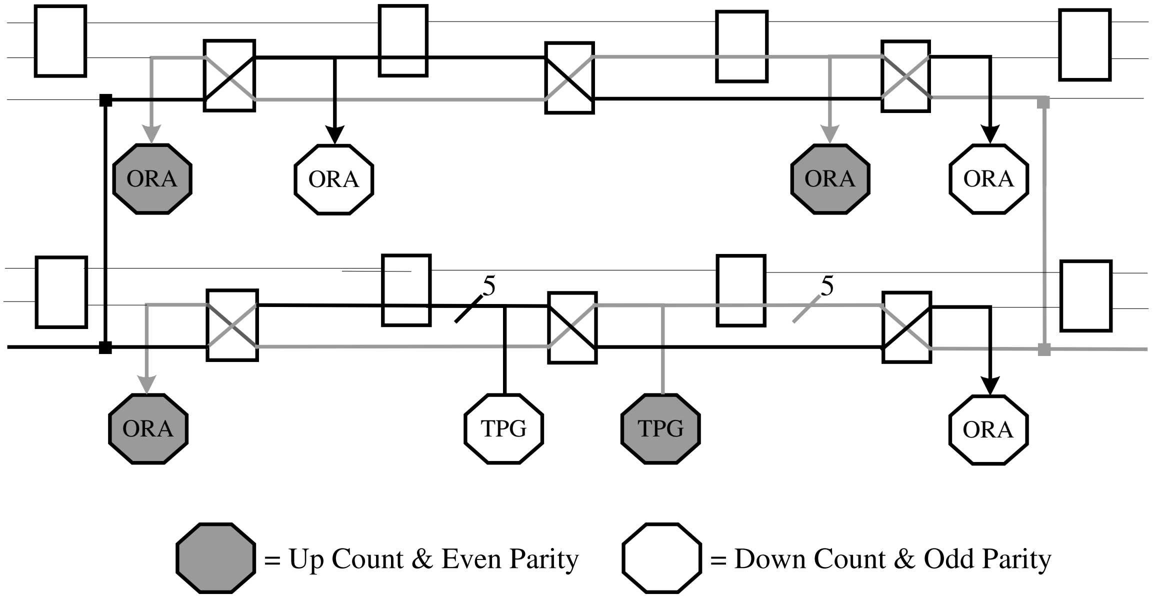
Figure 4.19 Flipped Abus Set 2 Configurations for Each Array Size

The basic architecture of the Set 2 Ebus line repeater configurations is illustrated in Figures 4.18 and 4.19, which give the structure for the 16x16, 32x32, and 48x48 arrays and for the 24x24 array, respectively. The Set 2 configurations have STARs that are of size *2x16* for horizontal configurations and *16x2* for vertical configurations. In the Set 2 configurations, however, the problems that arose at the boundaries of the Ebus line repeater are overcome by the schemes implemented for the different array sizes, as illustrated in Figures 4.20 and 4.21. Since the array sizes 16x16, 32x32, and 48x48 are multiples of 16, the architecture illustrated in Figure 4.20 can be overlapped and tiled to fit into these array sizes. Since the 24x24 array is not a multiple of 16, the architecture needed to be adapted to fit into the array. Therefore, the scheme for the 16x16, 32x32, and 48x48 array sizes was adapted for the 24x24 array. The basic architecture for the configuration is similar between the two schemes, with both having STARs that are two rows (or two columns) wide. These configurations are generated from an MGL program that accounts for both architectures of the Set 2 Ebus configurations.



*TPGs = 3 PLBs & ORAs = 2 PLBs

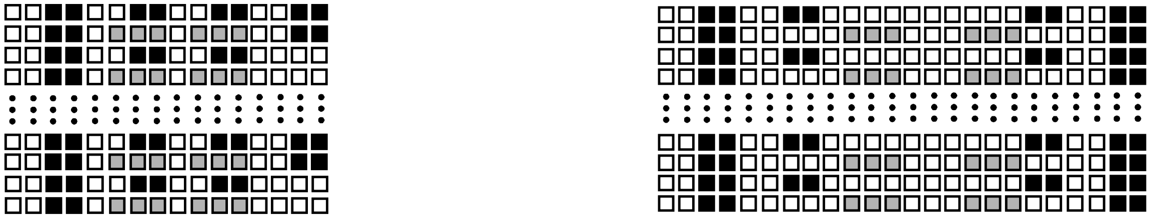
Figure 4.20 Ebus Line Repeater Set 2 Configuration (16x16, 32x32, & 48x48)



*TPGs = 3 PLBs & ORAs = 2 PLBs

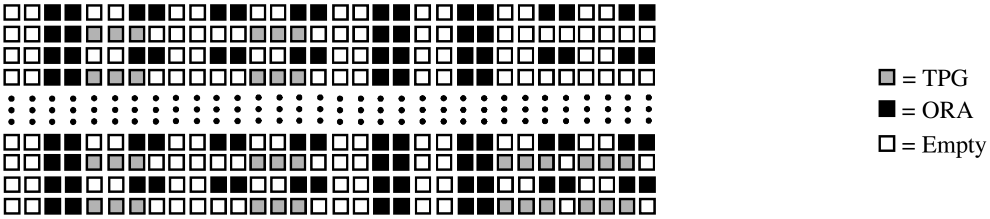
Figure 4.21 Ebus Line Repeater Set 2 Configuration (24x24)

The Set 2 configurations are shown in Figures 4.22 and 4.23 tiled into the different array sizes for both the unflipped and flipped versions of the configuration, respectively, which test both directions through the Ebus line repeaters. As with the Set 1 Ebus configurations, the tiling of the STARs in the Set 2 Ebus line repeater configurations is not as regular and structured as the Set 2 Abus line repeater configurations. These configurations impose constraints on the diagnostic resolution, as seen in the edges of the Ebus line repeater Set 1 configurations. Between any TPG and ORA in this scheme, there are two repeaters that the test patterns must travel through, thus, in the presence of a fault, diagnosis can be made to one of two repeaters.

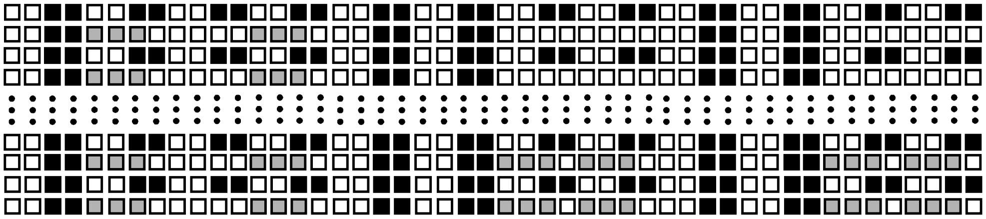


a) 16x16 Ebus Repeater Session

b) 24x24 Ebus Repeater Session

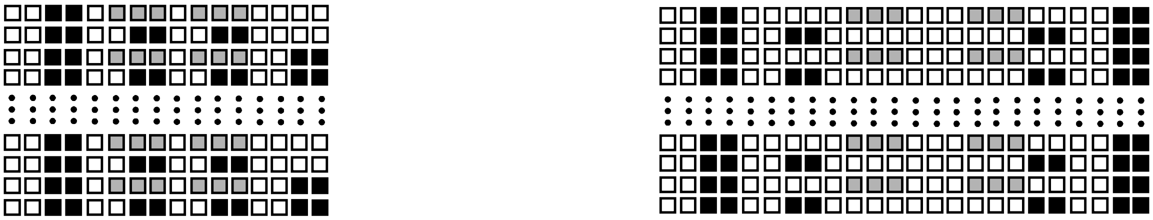


c) 32x32 Ebus Repeater Session



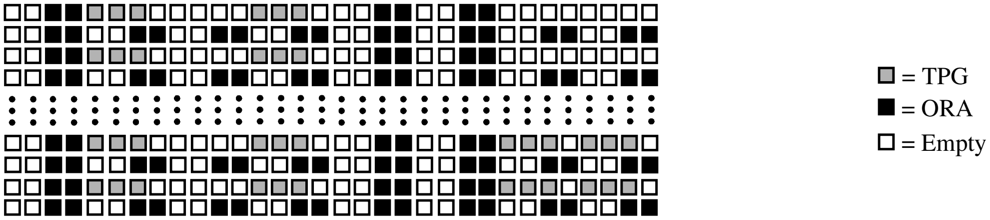
d) 48x48 Ebus Repeater Session

Figure 4.22 Ebus Set 2 Configuration for Each Array Size

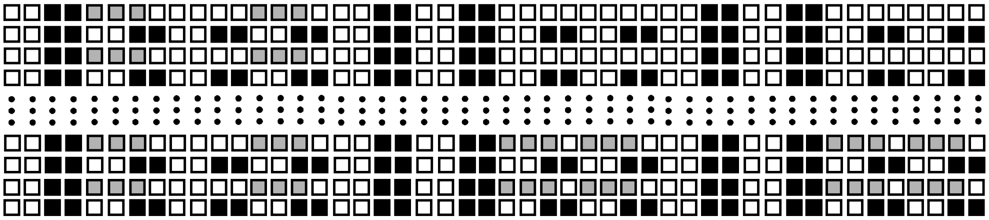


a) 16x16 Ebus Repeater Session

b) 24x24 Ebus Repeater Session



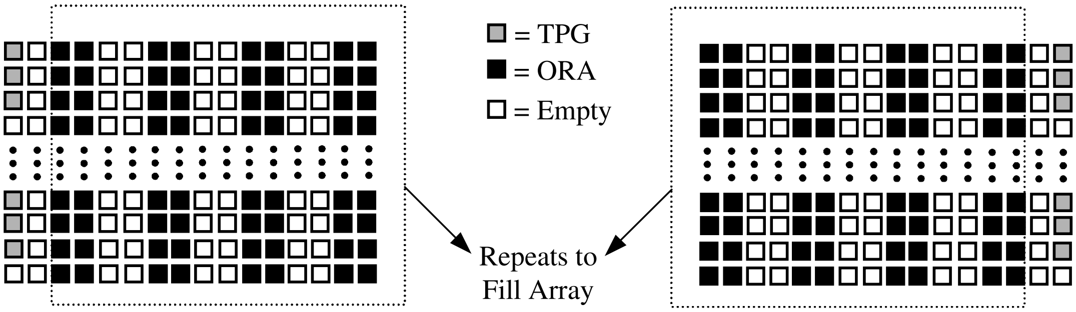
c) 32x32 Ebus Repeater Session



d) 48x48 Ebus Repeater Session

Figure 4.23 Flipped Ebus Set 2 Configuration for Each Array Size

The Abus and Ebus line repeater Set 3 configurations target faults in the straight through connections in the MUX PIPs in the repeaters, which are stuck-off faults on connections R8-L8 and R4-L4 or on connections L8-R8 and L4-R4, depending on the direction of the configuration. The basic architecture of the Set 3 configurations is given in Figure 4.10, which illustrates that the STARs in these configurations are *4xarray* (*arrayx4* for vertical sessions). This architecture can be applied to both the Abus and Ebus lines in a similar manner differing only by which *x8* line is tested during a given configuration. These configurations do not require alternating TPGs with up or down-count with even or odd parity, since stuck-off faults in the straight through connections of the repeaters do not require any type of opposite logic values on unselected inputs for detection. The tiling of the *4xarray* (or *arrayx4*) STARs for the Abus and Ebus line repeater Set 3 configurations is illustrated in Figures 4.24a and 4.24b for the unflipped and flipped architectures, respectively, which test both directions through the repeaters for the targeted stuck-off faults. This Set of configurations forms a regular structure and is easily tiled to fill the various array sizes.



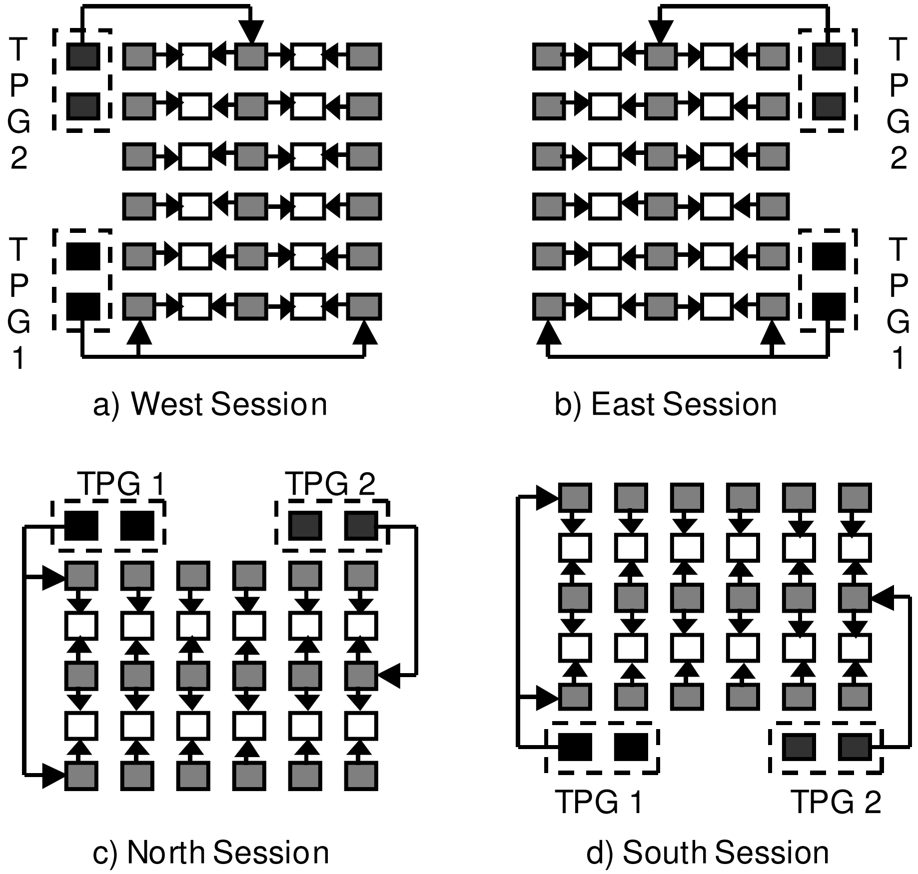
a) Normal Set 3 Configuration

b) Flipped Set 3 Configuration

Figure 4.24 Abus and Ebus Set 3 Configuration for Each Array Size

4.6 BIST Configurations for the Tri-Stated L Output and the X Direct Connections

The BIST configurations targeting the tri-stated PLB **L** output utilize the same architecture employed for the logic BIST configurations with the exception being in the BUT to ORA connections and the size of the TPG utilized to supply test patterns. The tri-state buffer and its associated output enable lines present on the **L** output as well as the connections to the vertical and horizontal global routing resources via local routing cross-point PIPs are targeted for testing. The local routing cross-point PIPs targeted for testing are closely coordinated with the **X** direct connection configurations such that all local routing cross-point PIPs making connections to the global routing resources are tested. This means that the proper selection of the local routing cross-point PIPs must be coordinated with the **X** direct configurations so that each local routing cross-point PIP is observed during one of the two sets of routing BIST configurations. For the **L** output configurations, there are North, South, East, and West test sessions. These test sessions are required to ensure that most PLBs have their **L** output observed and that both the horizontal and vertical output enable inputs to the tri-state buffer are tested during the configurations. The architecture of the **L** output configurations is demonstrated in Figure 4.25, which shows all four directional orientations of the configuration.

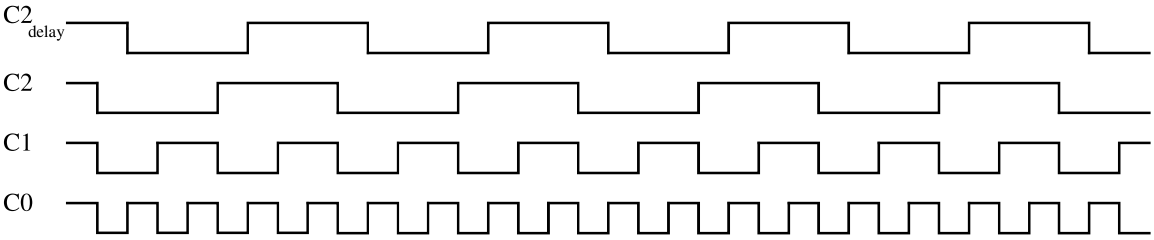


a) West Session b) East Session

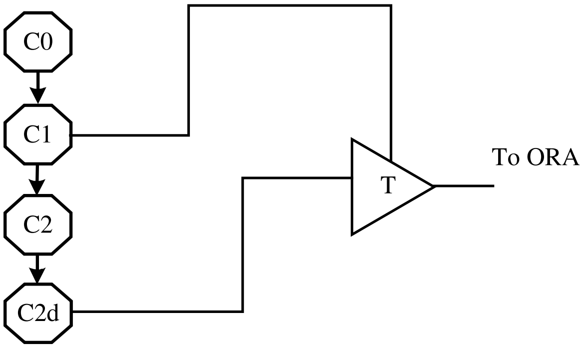
c) North Session d) South Session

Figure 4.25 L Output Configuration Architecture

The TPG employed to supply test patterns is similar to that used in the logic BIST test sessions. However, in the **L** output configurations, a 3-bit binary counter is used instead of the 5-bit binary counter used in logic BIST. In addition, the third bit of the binary count is delayed by one clock cycle by feeding the bit through a fourth PLB's flip-flop. The second bit of the 3-bit binary counter connects to the output enable of the tri-state buffer and the most significant bit of the binary counter is delayed by the one clock cycle and is the input to the tri-state buffer as illustrated in Figure 4.26. The connection scheme and delayed counter bit are employed to avoid any clock data races between the count values due to the behavior of the tri-state buffer on the **L** output. The tri-state buffer actually has behavior that mimics that of a dynamic latch, meaning that the data at the input to the tri-state buffer is actually latched for a brief period of time after the buffer is tri-stated. This means that the counter must be preset such that the tri-state buffer is disabled at the beginning of the test in order to avoid unknown initial data from being latched. This architecture allows the PLBs with their **L** output under test to be tested in a minimum of two sessions, since, during a given test session, half the PLBs have this output tested and observed in an ORA. However, since the tri-state buffer has both a horizontal and vertical output enable signal, the rotation as shown in Figure 4.25 is used to test for the operability of both signals. As illustrated by the timing diagram in Figure 4.26, the connections scheme allows the tri-state buffer to be enabled and disabled during a test session allowing its ability to pass data to be tested as well the functionality of the output enable to be tested.



a) Timing Diagram for Binary Count Sequence

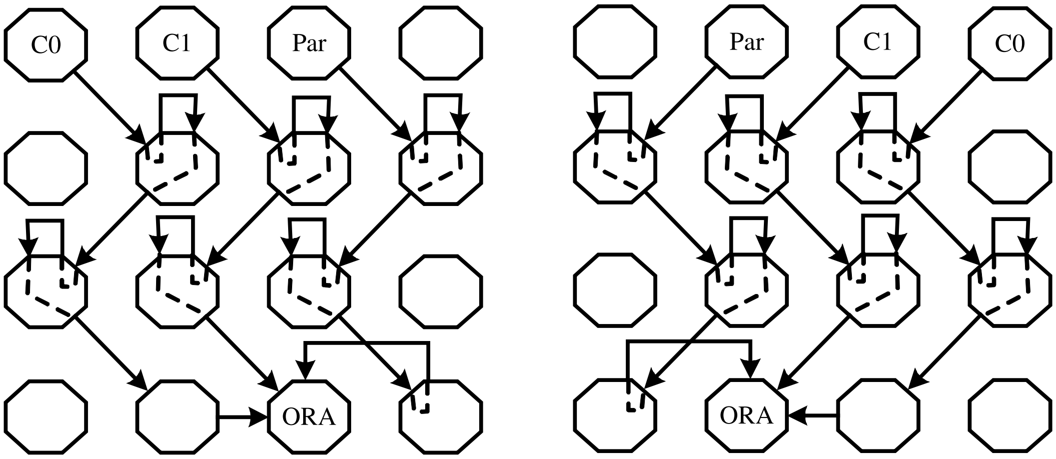


b) TPG Connections to **L** Output Tri-State Buffer

Figure 4.26 L Output Configuration Structure and Timing Diagram

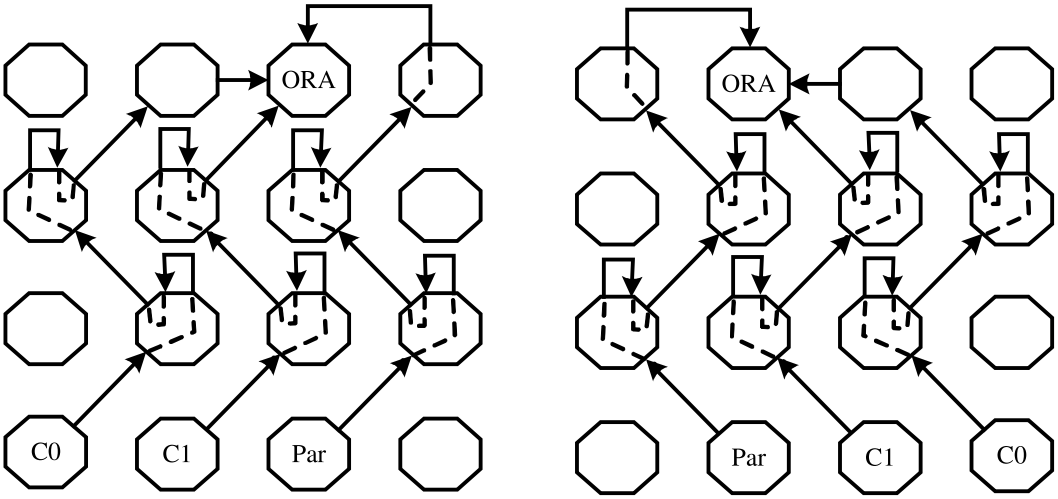
The BIST configurations developed for the **X** direct connections target the PLB input **X** MUX PIPs left incompletely tested after completion of logic BIST test sessions. Due to the routing schemes employed for logic BIST, which are shown in Figure 3.3, particular PLBs in the array have **X** direct connections that are not observed during logic BIST test sessions or any other routing BIST test session, even after the rotation to form North, South, East, and West logic BIST sessions. In addition to targeting faults left undetected in the **X** direct connections, these BIST configurations also target faults in the PLB **L** output and its connections to the global routing resources via local routing cross-point PIPs. These tests are closely corresponded to the **L** output tri-state test configurations such that the local routing cross-point PIPs making connections to the global routing resources are completely tested between the two sets of routing BIST configurations. To test for these faults, the input signal entering the PLB at the **X** input is sent to the **L** output and then fed back into the PLB through connections made in the local routing cross-point PIPs to route the signal back into the PLB on the **Y** input from the local routing cross-point PIP connections in the global routing.

The configurations targeting the **X** direct connections consist of four phases, which are illustrated in Figure 4.27 and show only a 4x4 array of PLBs for simplicity. The four phases consist of STARs that are actually of size *arrayx3*, where the first two phases have TPGs placed along the north side of the array and source test patterns down through the PLBs to ORAs along the south side of the array. The directions of the PLB **X** connections are flipped about the vertical axis from the first phase to the second phase. The first and second test phases are flipped about the horizontal axis such that the TPGs and ORAs swap positions and the test reverses directions through the PLBs in order to form the third and fourth test phases, respectively. The STARs shown in Figure 4.27 are butted against one another such that between two STARs there are no empty PLBs. These configurations provide test conditions for all of the **X** direct connections through the middle of the FPGA array.



a) Phase 1

b) Phase 2



c) Phase 3

d) Phase 4

Figure 4.27 X Direct Connection Configuration Test Phases

The direct connections that are at the edges of the array are left untested since they only connect to I/O cells. However, these can be tested in BIST configurations developed specifically for the I/0 cells. The connections that are left untested at the edge of the array also include **Y** direct connections in addition to the **X** direct connections due to their connections only to the I/O cells. Some of the local routing cross-point PIPs associated with the **L** output are also left untested. These, like the unobserved **X** and **Y** direction connections, are also at the edges of the array. However, for those PLBs at the edges of the array, not all local routing cross-point PIPs are untested, only some are unobserved. Figure 4.28 provides an illustration of the particular connections left untested after the completion of all the logic and routing BIST configurations. The number denoted in the PLBs gives the number of untested **L** output cross-point PIPs with no number signifying all PIPs are tested. The black and gray lines, respectively, give the **X** and **Y** direct connections unobserved after completion of logic and routing BIST.

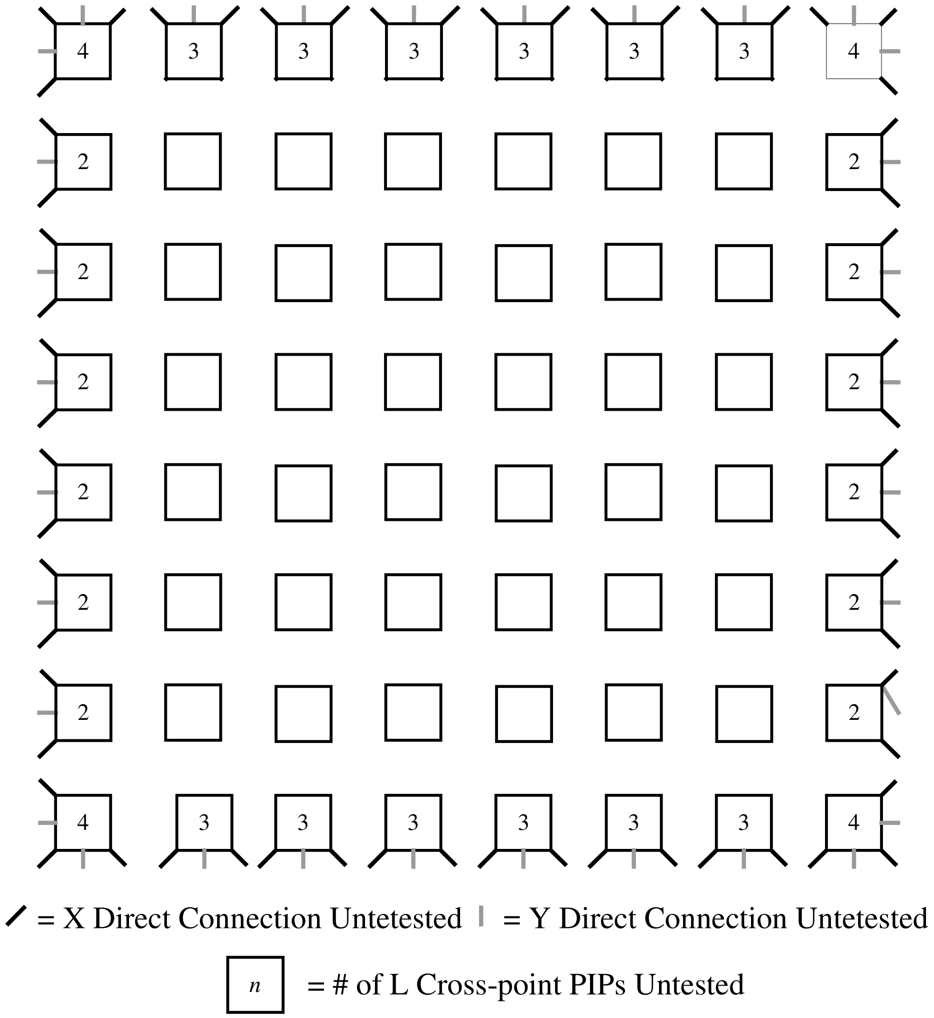


Figure 4.28 Untested Direct Connections and L Output Cross-point PIPs

4.8 Summary of Routing BIST Configurations

The routing BIST configurations for the Atmel FPGA test most of the programmable routing resources dispersed within the array. The resources left untested appear at the periphery of the array near the I/O cells of the device. All of the global routing and local routing cross-point PIPs, the Ebus repeaters, and the L output and associated tri-state buffer along with the **X** direct connections and the Abus repeaters in the middle of the array are completely tested for the types of fault models assumed. The **X** direct and **Y** direct connections associated with the PLBs along the edges of the array that connect only to I/O cells are not tested for any faults since these are not selected during logic or routing BIST. Some of the **L** output cross-point PIPs at the edges of the array, given in Figure 4.28, are left untested. The Abus repeaters that lie on the edge of the array are also incompletely tested since these repeaters do not have proper test conditions set up during testing. These repeaters have test patterns sourced only to one side of inputs being either the L8 and L4 inputs or the R8 and R4 inputs, depending on the exact side of the array. In addition, the Abus line wire segments along the 8 PLBs nearest the edges of the array are not tested for shorts between the wire segments. In the case of routing BIST, due to the abstract behavior of routing faults, no fault simulations were performed to determine fault coverage. However, based on the fault models used, the targeted faults can be said to be detected in all cases except where noted since the proper test conditions are set up for the fault models used. An exact figure for the fault coverage obtained with the routing BIST configurations can be obtained, however, an estimate can be made of the approximate total percentage of routing resources that are tested. Based on the composition of routing resources and the number of routing resources tested during the routing BIST test sessions, it is estimated that between 90% and 95% of the programmable routing resources are tested for the assumed fault models. However, all of the incompletely tested resources can be tested in BIST configurations developed for the purpose of testing the I/O cell network and associated routing.

All of these routing BIST configurations were generated through the use of both MGL and C programs. Table 4.3 provides a summary of the BIST configurations along with the non-commented lines of code associated with the respective configurations. Separated codes were generated for each of the cross-point PIP configurations, for the **L** output and tri-state configurations, and for the **X** direct connection configurations. For the case of the Abus and Ebus repeater configurations, separate MGL programs were used to generate each set of configurations, but only one C program was used to perform bit manipulation for each of the three sets of Abus and Ebus repeater configurations. The complexity caused by the staggering of the repeaters can be seen in the difference of the number of lines of code required for the Abus and Ebus configurations. The Ebus programs consistently have more lines of code, which is due to the increased complexity of the configuration architectures employed to test the targeted routing resources associated with the Ebus *x8* lines.

Table 4.3 Summary of Routing BIST MGL and C Programs

Routing BIST Configuration	MGL Non-commented Line Count	C Code Non-commented Line Count
Abus Cross-Point PIPs	750	200
Ebus Cross-Point PIPs	1600	650
Abus Repeaters Set 1	1000	650
Abus Repeaters Set 2	2000	
Abus Repeaters Set 3	900	
Ebus Repeaters Set 1	1900	1000
Ebus Repeaters Set 2	2500	
Ebus Repeaters Set 3	900	
L Ouptut and tri-state	1200	450
X Direct Connections	750	350

4.9 MGL's Effect on Routing BIST Development and Application

The utilization of MGL resulted in a significant impact on the development and application of routing BIST. To be able to implement the routing BIST configurations such that the BIST results can be retrieved at the end of a BIST sequence, the ORAs must have similar conditions to those of the logic BIST configurations. A shift signal must be routed to all the ORAs so that, at the end of a BIST sequence after the ORAs are reconfigured into a shift register, the results can be shifted out. This is due to the requirement in MGL for a source and destination to be specified for any route required. Thus, in order to have a shift signal present after reconfiguration for a shift register, it must be routed in the MGL program. Due to the architecture of the routing BIST configurations, the three bits in the test pattern must be routed through a set of WUTs to each ORA. This condition makes for even more congestion of needed functionality in the PLBs than is the case for logic BIST. A total of four signals must be routed to the ORAs in order to perform routing BIST, the three bits of the test pattern along with the Shift signal. Thus, an ORA function can not be implemented through MGL for two reasons. The first reason lies in the fact that the a dynamic macro implementing a 4-input function with feedback does not exist and, furthermore, cannot be implemented in the PLB. The second, as mentioned previously, is that a signal must have a source and a destination in MGL such that a Shift signal can only be routed if a source node (i.e. input from an I/O cell) and a destination (i.e. an input to a PLB, namely an ORA) are both specified. Thus a dynamic macro implementing a 4-input registered function (FGEN1 dynamic macro) is needed to set up conditions initially for a routing BIST configuration. This requires that a template bitstream be created containing the necessary routing information for the routing BIST configurations. Thus, bit manipulation via a C program is required before a routing BIST configuration can be downloaded into the FPGA. For each set of routing BIST configurations, a different C code was generated to perform the required bit manipulation.

The effort in generating BIST configurations through the use of MGL is similar to previous efforts in the generation of BIST configurations for the ORCA [9] and Xilinx [10] FPGAs. In the previous work, C programs were used to generate textual netlists of the BIST configurations that were interpreted by the vendor-supplied CAD tools for the respective FPGAs. In the generation of BIST configurations using MGL, C programs are again needed in order to produce the BIST configurations for the device. The actual process is somewhat different, with the previous work creating netlists via C programs while the work described in this thesis uses C programs to perform bit manipulation on the download bitstream files. However, the concept is the same between the different cases; the vendor supplied CAD tools themselves do not provide adequate control over the resources within the FPGA.

4.10 Comparison of the Routing BIST for Atmel, ORCA, and Xilinx FPGAs

The routing architectures of the three devices have several differences which include the direct connections to adjacent PLBs within the local routing resources, the percentage composition of the different types of PIPs in the local and global routing resources, and the number and type of wire segments in the global routing resources, as was summarized in chapter 2. As found in [10], the routing architecture of a given FPGA has the most impact on the total number of BIST configurations required for complete testing of that FPGA. The asymmetric routing architecture in the Xilinx devices is more complex and suffers more from factors that affect the number of routing BIST configurations required, such as shared routing resources between PLBs, rotating and staggered busses, and large numbers of MUX PIPs with a high number of inputs, all of which cause an increase in the number of configurations [10]. The Atmel and ORCA devices employ a more symmetric and regular routing architecture that is much less complex than that found in the Xilinx FPGAs. This is demonstrated in the total number of routing BIST configurations required for the respective FPGAs. Table 4.4 provides a summary of the routing BIST configurations for the ORCA, Xilinx, and Atmel FPGAs. The Xilinx FPGAs required 128 configurations for the 4000E and Spartan series FPGAs and 206 configurations for the 4000XL and 4000XLA series FPGAs to completely test the routing resources [10]. The ORCA 2C and 2CA FPGAs required 27 and 44 configurations, depending on the size of the STARs, to completely test the routing resources [10]. The Atmel FPGA requires 48 configurations to completely test the routing resources, which is more similar to that of the ORCA FPGAs.

Table 4.4 FPGA Routing BIST Comparison

FPGA	Number of BUT Configurations
ORCA 2C	27,44
ORCA 2CA	27,44
Xilinx 4000	128
Xilinx Spartan	206
Atmel AT40K	48
Atmel AT94K	48

In contrast, however, parts of the local routing resources are being tested during logic BIST for the Atmel FPGA, which is not the case with the ORCA FPGA. In addition, the routing architecture of the Atmel device does not have as much access to the PLB inputs and outputs as found in the ORCA FPGA. A large factor in the testability of the routing resources of an FPGA lies in their access to the PLBs. Even though the routing architecture is much more complex in the ORCA FPGA than in the Atmel FPGA, there are fewer required routing BIST configurations. This is due in large part to the poor access of the PLBs in the array to all routing resources. The PLB only has access to the routing resources on two of its sides and, furthermore, can access only *x4* lines directly while having to go through a repeater to reach the either of the *x8* lines. The rotational symmetry of the routing architecture does help to provide a fewer number of configurations as compared to the asymmetric routing architecture in the Xilinx FPGAs, but is not enough to overcome the limited access of the PLBs to the routing resources.

CHAPTER FIVE

SUMMARY

The application of BIST to the programmable logic and routing resources present in an FPGA core in a commercially available generic SoC has been presented. Automatic generation of BIST configurations using vendor supplied CAD tools along with custom developed C programs has been discussed. These automatically generated BIST configurations can be applied to any AT40K series FPGA or AT94K series FPGA core.

The amount of available logic in the PLB has been found to impact the application of BIST. The ORAs for both logic and routing BIST are significantly impacted due to the small amount of logic present in the PLB. This problem has been overcome in this research work by generating logic and routing BIST architectures that minimize the number of configurations while allowing sufficient observation of the BUTs or WUTs in the ORAs. In addition, the device's ability to be partially reconfigured was used to help overcome the issues of the small PLBs. The ORAs were reconfigured into a shift register after completion of a BIST sequence in order to maximize the use of the available logic during the BIST sequence without having the shift register use required logic for the comparison or parity check of the BUTs or WUTs.

The limitations imposed by the available direct connections in the local routing resources associated with the PLBs were overcome by the alternating BUT-to-ORA routing schemes shown in Figure 3.3. The type of ORA chosen for the implementation of logic BIST was greatly impacted by these connections and by the size of the PLB. The routing schemes shown in Figure 3.3 overcome most of the problems in the observability of the **X** and **Y** PLB outputs which connect only to adjacent PLBs. However, the remaining issues in the observability of the **X** and **Y** PLB outputs at the edges of the PLB array are overcome by the rotation of the logic BIST architecture to form not only the East and West test session, but also the North and South test sessions. In any given array size, only eight PLBs are left incompletely tested after the rotation of the logic BIST architecture.

Existing methods in routing BIST techniques were adapted for this architecture to overcome issues caused also by the size of the PLBs. The method described in [23] was adapted to work with the fine-grain architecture of the Atmel FPGA. Since the size of the PLB allows for implementation of only one bit of a binary counter, the adapted method, utilizing a two-bit binary counter with odd or even parity generation, was the best choice to overcome the associated implications. This new methodology for routing BIST provides the best solution for FPGAs with fine-grain architecture similar to the AT40K FPGAs and AT94K FPGA cores.

As found in [10], the staggering and inaccessibility of routing resources has been found to have a large impact on the application of routing BIST. In this application, BIST architectures were developed to overcome the obstacles imposed by the inaccessibility to the *x8* lines, and the staggering of repeaters in the array. In this case, BIST architectures were developed that are as similar as possible, independent of the particular *x8* lines on which a given repeater is staggered. The rotational symmetry of the routing resources allowed the configurations to be applied to both horizontal and vertical repeaters dispersed in the array.

In this particular case, the vendor-supplied CAD tools also imposed restrictions on the test development and application. It was found that newer versions of the software used to compile the MGL code to instantiate designs into the FPGA place and route tool and create bitstreams created problems in particular instances with the routing needed to apply the BIST configurations. This was not encountered in the application of logic BIST, but was seen in the application of routing BIST. Particular routes are required to test certain parts of the programmable routing resources and, in some cases, these routes could be achieved in older versions of the software but not in newer versions. Therefore, this imposes a requirement that new and old versions of the software must be maintained in order to generate all the BIST configurations. Since as few mediums as possible are preferred to generation test configurations, this inhibits the test development and application process since multiple versions of software must be used to generate BIST configurations for a single family of FPGAs.

Partial reconfiguration of the FPGA core has important positive implications on the application of both logic and routing BIST. Previously, for the case of Xilinx FPGAs, the configuration memory of the FPGA has been read to retrieve the BIST results at the end of a BIST configuration. In the case of the Atmel AT94K FPGA cores, the FPGA configuration memory can be written or re-written but not read from the AVR core [14]. However, the configuration memory can be read in the AT40K series FPGAs via the synchronous RAM mode in order to retrieve the values in the ORA flip-flops [15]. For the case of the AT40K series FPGAs, the ORA flip-flop contents can be read through the configuration memory in order to determine pass or fail results. However, this can not be done for the FPGA core in the AT94K SoC. Partial reconfiguration must be used in order to retrieve the results at the end of a BIST configuration. In this case, a BIST configuration is downloaded and executed, the ORAs are then reconfigured as a shift register, and the BIST results shifted out. For the case of the ORCA FPGAs, there is sufficient logic for the implementation of both the ORA function and shift register within a single PLB. In the case of the Xilinx FPGAs, the configuration memory can be read in order to read the values of the ORA flip-flops to determine pass or fail results. Partial reconfiguration allows the BIST architecture to be applied to FPGA cores where the configuration memory can not be read and where the PLBs are not capable of implementing both an ORA function and shift function.

5.1 Comparison of Work

The work completed for the Atmel AT40K series FPGAs and AT94K series FPGA cores provides new insights on the application of BIST for the programmable logic and routing resources in an FPGA. The work completed previously for ORCA and Xilinx FPGAs along with the work completed in this thesis provide a basis of comparison in the application of BIST to coarse-grain architecture FPGAs to the application of BIST to fine-grain architecture FPGAs. It is shown in this thesis that the fine-grain architecture facilitates fewer BIST configurations to achieve complete testing of the logic resources present in the FPGA in comparison to a course-grain architecture. In addition, the fine-grain architecture is shown to have a significant impact on the application of routing BIST. The parity-based routing BIST approach described in [23] was modified and is shown to work well with the fine-grain architecture in the Atmel FPGAs and FPGA cores. Partial reconfiguration is also exploited in this case in the application of BIST for the logic and routing resources. With the ability to perform partial reconfiguration, fine-grain architecture FPGA cores can decrease the development effort in the application of BIST to offset the impact of limited PLB logic resources by maximizing the number of observable BUT outputs or WUTs during a given test session. This is true because the PLBs can be more efficiently used to perform ORA functions only during the BIST sequences instead of having to utilize multiple PLBs to implement ORA and shift register functions. Once the BIST sequences have been applied, partial reconfiguration can be done to create a shift register in the ORAs in order to shift out the BIST results.

5.2 Future Work

The work completed for this thesis has generated many interesting ideas for the application of BIST for FPGAs. The fine-grain architecture FPGA has many implications on the implementation of BIST. Methods have been presented in this thesis to overcome limitations on the application of BIST to fine-grain architecture FPGAs not previously encountered with coarse-grain architecture FPGAs such as the ORCA and Xilinx FPGAs. Suggestions for future work in this area would be to improve upon the adaptations made to the logic and routing BIST methodologies. In particular, future work might include evaluating the advantages of developing more specific BIST configurations for the AT94K series FPGA cores. The BIST configurations in this thesis can be applied to both the AT40K series FPGAs and the AT94K series FPGA cores. Since the AVR core in the AT94K series devices can be used to write or rewrite the configuration memory of the FPGA core, it would be advantageous to research new methodologies to exploit these capabilities and evaluate the impact this has on the number of BIST configurations and the respective diagnostic resolution of such configurations.

5.3 Conclusion

This research set out to develop BIST configurations for the logic and routing resources present in the Atmel AT40K series FPGAs and AT94K series FPGA cores. Central to this research was the need to further develop existing methodologies used in BIST for FPGAs, namely in the application of BIST to the ORCA and Xilinx FPGAs. In doing so, a methodology was developed for testing fine-grain architecture FPGAs. These BIST configurations were developed using both vendor-specific CAD tools along with custom developed C programs to automatically scale to any size device in the AT40K series FPGAs and AT94K series FPGA cores. However, the methodology utilized in the application of BIST for FPGAs to these devices can be applied to any similar fine-grain architecture FPGA.

REFERENCES

[1] G. Aldrich, “Yes, You Can Get a Testable SoC Design to Market on Time,” *Electronic Design Magazine*, ED Online ID # 1820, November 2002.

[2] D. Bursky, “Digital ICs: Programmable Logic,” *Electronic Design Magazine*, ED Online ID # 3294, January 2003.

[3] D. Maliniak, “Tool Reveals SoC Hot Spots,” *Electronic Design Magazine*, ED Online ID # 3146, April 2003.

[4] M. Abramovici, C. Stroud, and M. Emmert, “Using Embedded FPGAs for SoC Yield Improvement,” *Proc.ACM/ IEEE Design Automation Conf.*, pp. 713-724, 2002.

[5] W. Wolf, “Modern VLSI Design: System-on-Chip Design,” Prentice Hall, New Jersey, 2002.

[6] P. Chan and S. Mourad, “Digital Design Using Field Programmable Gate Arrays,” Prentice Hall, New Jersey, 1994.

[7] C. Stroud, “A Designer’s Guide to Built-In Self-Test,” Kluwer Academic Publishers, Boston, 2002.

[8] M. Abramovici, M. Breuer, and A. Friedman, “Digital Systems Testing and Testable Design,” Computer Science Press, New York, 1990.

[9] E. Lee, “Built-In Self-Test and Diagnosis of Field Programmable Gate Arrays,” M.S.E.E. Thesis, University of Kentucky, 1997.

[10] C. Stroud, K. Leach, and T. Slaughter, “BIST for Xilinx 4000 and Spartan Series FPGAs: A Case Study,” *Proc. IEEE International Test Conf.*, pp. 1258-1267, 2003.

[11] C. Stroud, J. Nall, M. Lashinsky, and M. Abramovici, “BIST-Based Diagnosis of FPGA Interconnect,” *Proc. IEEE International Test Conf.*, pp. 618-627, 2002.

[12] J. Nall, “On-Line and Off-Line Built-In Self-Test Based Diagnosis of Interconnect Faults in Field Programmable Gate Arrays,” M.S.E.E. Thesis, University of North Carolina at Charlotte, 2002.

[13] M. Lashinsky, “On-Line and Off-Line Built-In Self-Test of Field Programmable Gate Array Interconnect Resources,” M.S.E.E. Thesis, University of North Carolina at Charlotte, 2001.

[14] __, AT94K Series Field Programmable System Level Integrated Circuit, Data Sheet, Atmel Corporation, 2003.

[15] __, AT40K Series Field Programmable Gate Array, Data Sheet, Atmel Corporation, 2003.

[16] W. K. Huang, F. J. Meyer, and F. Lombardi, “An Approach for Detecting Multiple Faulty FPGA Logic Blocks,” *IEEE Trans. on Computers*, vol. 49, pp. 48-54, January 2000.

[17] W. K. Huang, F. J. Meyer, X. Chen, and F. Lombardi, “Testing Configurable LUT-based FPGAs,” *IEEE Trans. on VLSI Systems*, vol. 6, pp. 276-283, June 1998.

[18] T. Inoue, S. Miyazaki, and H. Fujiwara, “Universal Fault Diagnosis for Lookup Table FPGAs,” *IEEE Design and Test of Computers*, vol. 15, pp. 39-44, Januray 1998.

[19] M. Abramovici and C. Stroud, “BIST-Based Test and Diagnosis of FPGA Logic Blocks,” *Proc. IEEE Trans. on VLSI Systems*, vol. 9, pp. 159-172, 2001.

[20] C. Stroud, E. Lee, and M. Abramovici, “BIST-Based Diagnosis of FPGA Logic Blocks,” *Proc. IEEE International Test Conf.*, pp. 539-547, 1997.

[21] M. Bushnell and V. Agrawal, “Essentials of Electronic Testing: For Digital, Memory, and Mixed-Signal VLSI Circuits,” Kluwer Academic Publishers, Boston, 2000.

[22] C. Stroud, S. Wijesuriya, and C. Hamilton, “Built-In Self-Test of FPGA Interconnect,” *Proc. IEEE International Test Conf.*, pp. 404-411, 1998.

[23] X. Sun, J. Xu, B. Chan, and P. Trouborst, "Novel Technique for BIST of FPGA Interconnects," *Proc. IEEE International Test Conf.*, pp. 795-803, 2000.

[24] M. Abramovici, C. Stroud, S. Wijesuriya, C. Hamilton, and V. Verma, "Using Roving STARs for On-Line Testing and Diagnosis of FPGAs in Fault-Tolerant Applications," *Proc. IEEE International Test Conf.*, pp. 973-982, 1999.

[25] __, ORCA Series 2 Field Programmable Gate Arrays, Data Sheet, Lattice Semiconductor Corporations, 2003.

[26] __, Xilinx XC4000E and XC4000X Series Field Programmable Gate Arrays, Data Sheet, Xilinx Inc., 1999.

[27] S. Pontarelli, G.C. Cardarilli, A. Malvoni, M. Ottavi, M. Re, and A. Salsano, "System-on-Chip Oriented Fault-Tolerant Sequential Systems Implementation Methodology", *Proc. IEEE International Symp. on Defect and Fault Tolerance in VLSI Systems*, pp. 455-460, 2001.

[28] C. Stroud, J. Nall, A. Taylor, M. Ford, and L. Charnley, "A System for Automated Generation of Built-In Self-Test Configurations For Field Programmable Gate Arrays", *Proc. International Conf. on Systems Engineering*, pp. 437-443, 2002.

[29] __, Integrated Development System Technical Reference and Release Notes Version 6.0, Atmel Corp., 1998.

[30] __, Integrated Development System AT40K Macro Library Version 6.0, Atmel Corp., 1998.