BUILT-IN SELF-TEST OF LOGIC RESOURCES IN FIELD PROGRAMMABLE GATE ARRAYS USING PARTIAL RECONFIGURATION

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.

_____

Sachin Dhingra

Certificate of Approval:

_____	_____
Victor P. Nelson Professor Electrical and Computer Engineering	Charles E. Stroud, Chair Professor Electrical and Computer Engineering
_____	_____
Vishwani D. Agrawal Professor Electrical and Computer Engineering	Stephen L. McFarland Acting Dean Graduate School

BUILT-IN SELF-TEST OF LOGIC RESOURCES IN FIELD PROGRAMMABLE GATE ARRAYS USING PARTIAL RECONFIGURATION

Sachin Dhingra

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
August 7, 2006

BUILT-IN SELF-TEST OF LOGIC RESOURCES IN FIELD PROGRAMMABLE GATE ARRAYS USING PARTIAL RECONFIGURATION

Sachin Dhingra

_____
Signature of Author

_____
Date of Graduation

VITA

Sachin Dhingra, son of Baldev Raj and Anita Dhingra, was born on May 31, 1982 in Chandigarh, India. He graduated with a Bachelor of Engineering degree in Electronics and Telecommunications Engineering in May 2003 from Maharashtra Institute of Technology, University of Pune, India. After completion of his undergraduate degree, he joined Electronic Projects Management, Canada as a Senior Test Technician in November 2003. He entered the graduate program at Auburn University in August 2004. While in pursuit of his Master of Science degree at Auburn University, he worked under the guidance of Dr. Charles E. Stroud as a graduate student research assistant in the Electrical and Computer Engineering Department.

THESIS ABSTRACT

BUILT-IN SELF-TEST OF LOGIC RESOURCES IN FIELD PROGRAMMABLE GATE ARRAYS USING PARTIAL RECONFIGURATION

Sachin Dhingra

Master of Science, August 7, 2006
(B.E, University of Pune, India, 2003)

97 Typed Pages

Directed by Charles E. Stroud

Field Programmable Gate Arrays (FPGAs) are programmable logic devices that can be used to implement virtually any digital circuit design. Built-In Self-Test (BIST) is a testing approach that enables the device to test itself without any external test equipment. The re-programmability feature of the FPGAs makes BIST a very attractive approach for testing FPGAs because it eliminates any area or performance degradation associated with BIST.

Traditional BIST for FPGAs suffers from long test times and large memory storage requirements due to the slow configuration download speeds and the large number of test configurations required to test the FPGAs. The work presented in this thesis implements testing of logic resources of Xilinx Virtex/Spartan-II and Virtex-4 FPGAs with focus on reduction of test time and memory storage requirements using

techniques like dynamic partial reconfiguration and partial configuration memory readback.

The total number of configurations required to completely test the logic resources are 28 for Virtex/Spartan-II FPGAs and 24 for Virtex-4 FPGAs. A speed-up of 5.1 times and 12.9 times in test time was achieved for Logic BIST for Virtex/Spartan-II and Virtex-4 FPGAs respectively, using dynamic partial reconfiguration and partial configuration memory readback. A reduction in configuration memory storage requirements was also achieved using partial reconfiguration; this reduction was 3.2 times and 5.3 times for Virtex/Spartan-II and Virtex-4 FPGAs respectively.

ACKNOWLEDGMENTS

I am indebted to Dr. Stroud for his support and advice throughout my research at Auburn University. I would also like to thank Dr. Nelson and Dr. Agrawal for being on my graduate committee and for their contribution to my thesis. I would like to acknowledge my research colleagues Daniel, John, Jonathan, Lee, Srinivas, and Sudheer for their help and inspirational discussions during my research. Finally, I would like to express my deepest gratitude to my parents and my brother whose love and encouragement is inspiring me to achieve my goals.

Style manual or journal used: IEEE (Institute of Electrical and Electronic Engineers) Journal style

Computer software used: Microsoft® Word® 2003, Microsoft® Excel® 2003, Microsoft® Visio® 2003

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

BIST	Built-In Self Test
BSCAN	Boundary SCAN
BUT	Block Under Test
CPLD	Complex Programmable Logic Device
CRC	Cyclic Redundancy Check
CUT	Circuit Under Test
DLL	Delay Locked Loop
DSP	Digital Signal Processor
FAR	Frame Address Register
FDR	Frame Data Register
FPGA	Field Programmable Gate Array
GSR	Global Set/Reset
IC	Integrated Circuit
ID	Identification
I/O	Input Output
IOB	Input Output Buffer
IP	Intellectual Property
LFSR	Linear Feedback Shift Register
LUT	Look-Up Table
NCD	Native Circuit Design
ORA	Output Response Analyzer
PIP	Programmable Interconnect Point
PLB	Programmable Logic Block
PROM	Programmable Read Only Memory
RAM	Random Access Memory
ROM	Read Only Memory

SRAM	Static Random Access Memory
TCK	Test Clock
TDI	Test Data Input
TDO	Test Data Output
TMS	Test Mode Select
TPG	Test Pattern Generator
VLSI	Very Large Scale Integration
XDL	Xilinx Design Language

CHAPTER ONE

Introduction

Rapid advances in semiconductor processing technologies have allowed transistor densities to double every two years; this phenomenon has led to new opportunities in Very Large Scale Integration (VLSI) design and new challenges in design verification and testing [1]. The growing complexity of design has made programmable logic devices like Field Programmable Gate Arrays (FPGAs) and Complex Programmable Logic Devices (CPLDs) some of the leading products in the semiconductor industry, as they provide an easy way to implement and verify complex digital designs.

Every innovation in Integrated Circuit (IC) design is accompanied by new challenges in testing. The test systems accordingly are becoming faster, more complex and hence more expensive. Cost and time are two of the most important factors that govern the development of any kind of test system. Built-In Self-Test (BIST) is one technique which reduces the cost and time overheads involved in external test systems. It is a technique that places a device's testing function within the device itself [6].

1.1 Field Programmable Gate Arrays (FPGAs)

FPGAs are programmable logic devices that can be configured or programmed to perform tasks specific to any digital application. The FPGAs gained popularity due to

their flexibility and short time-to-market, making them ideal for prototyping systems and low volume products. The design to be implemented in an FPGA is converted to a string of bits called the configuration file using tools provided by the FPGA manufacturer. The configuration file is used to program the memory elements inside the FPGA that control the functionality of the programmable components of the FPGA to implement the required design [2]. Traditionally, the entire configuration memory of an FPGA is rewritten with configuration data if the design needs to be modified; this is called full reconfiguration. Current FPGAs have the ability to be configured partially such that only the section of the configuration memory that changes due to the design modifications is rewritten with new configuration data. This configuration technique is known as partial reconfiguration [13] [14].

An FPGA typically consists of an array of Programmable Logic Blocks (PLBs), programmable interconnect network, Input/Output Buffers (IOBs) and embedded cores like memory blocks. The PLBs form the logic resources of an FPGA and usually consist of look-up tables, flip-flops and multiplexers. The programmable interconnect network is comprised of wire segments and programmable switches that connect or disconnect the wire segments. PLBs can be configured and connected to each other using the programmable interconnect network to implement virtually any combinational or sequential circuit. The IOBs are used to interface the circuit to the outside world [13]. A current trend in FPGAs is to embed pre-designed Intellectual Property (IP) cores into the FPGA. These IP cores include memory blocks like Random Access Memories (RAMs) and Digital Signal Processor (DSP) blocks to improve application-specific performance. Figure 1.1 shows the architecture of a typical FPGA.

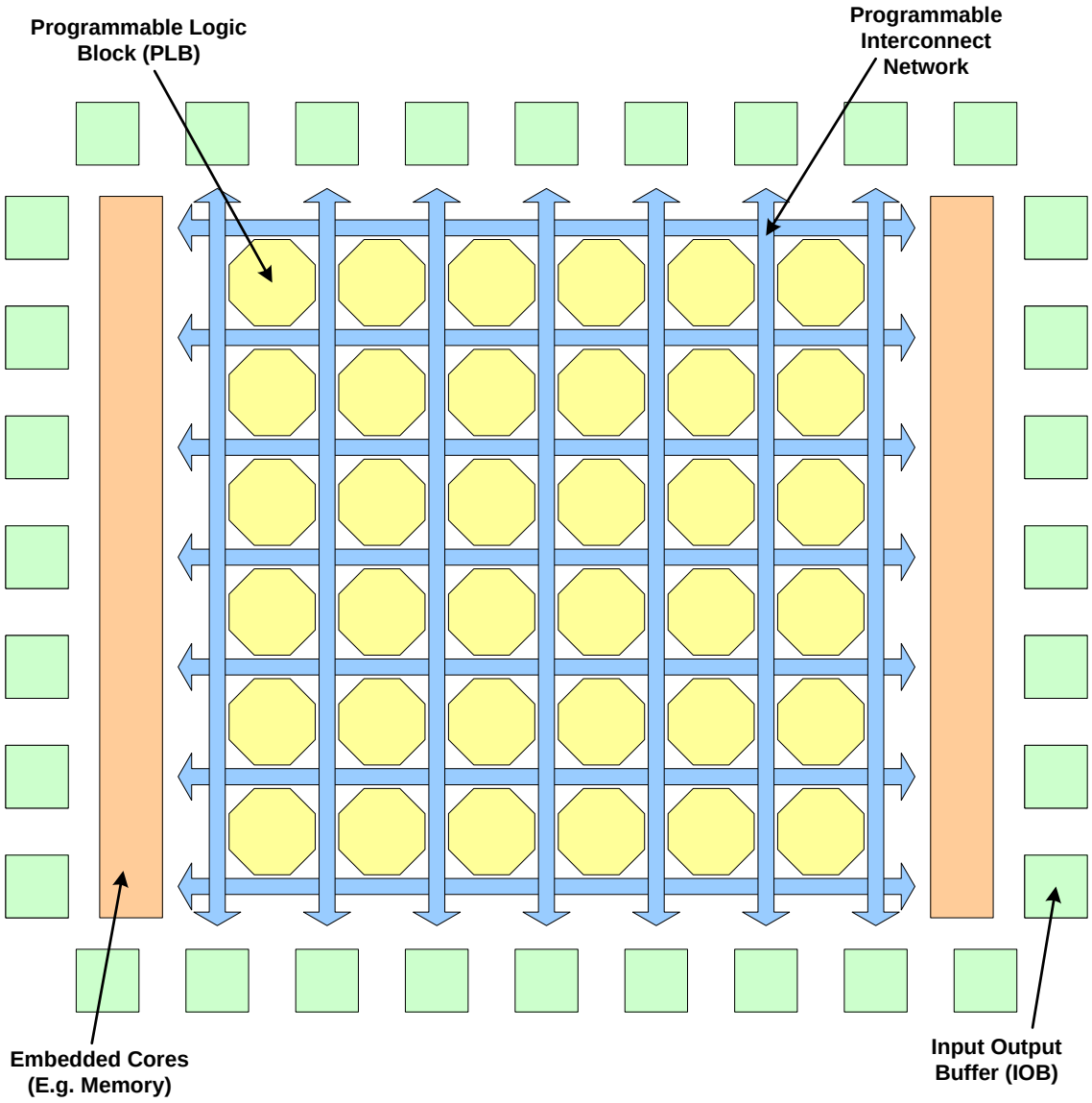

Figure 1.1 FPGA Architecture

1.2 Testing and BIST

A quality product can be delivered only if it has been tested thoroughly. Testing is done to ensure fault-free operation of a circuit. In order to test any circuit, a mechanism is needed to apply a set of input stimuli to the Circuit Under Test (CUT) and another

mechanism is required to analyze or compare the output response with the response of a known good circuit to determine whether the circuit is fault-free or faulty [5].

The input stimuli in case of external test systems are applied and the output response is analyzed or compared externally. In case of BIST, the test system is integrated within the system itself; the input stimuli are applied and the output response is analyzed internally within the system. The BIST technique involves addition of extra circuitry to an existing design. There are many variations in BIST depending on the CUT, but they all have a common purpose, which is to generate test patterns and analyze the output responses of the CUT [6]. A typical implementation of BIST consists of a Test Pattern Generator (TPG) for the CUT, input isolation circuitry for isolation of the primary inputs of the CUT during testing, an Output Response Analyzer (ORA) for verification of proper operation of the CUT, and control circuitry for execution of the test procedure as shown in Figure 1.2.

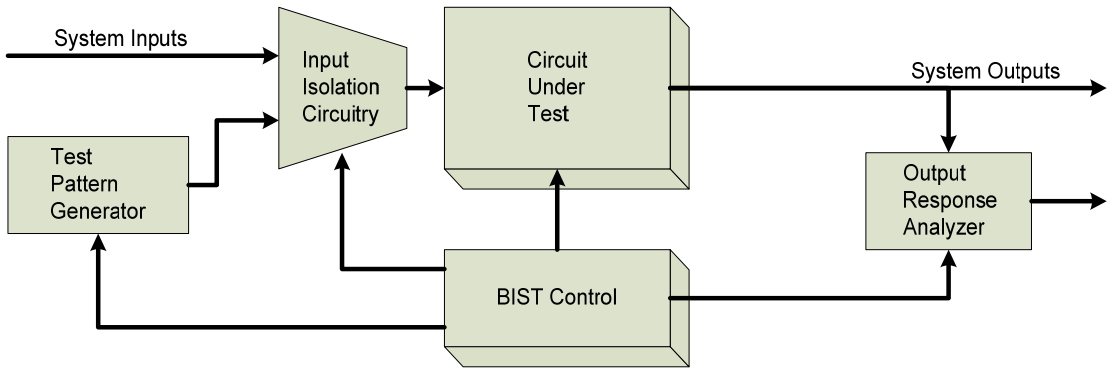

Figure 1.2 Basic BIST Architecture [6]

The external test approach is best suited for circuits that allow access to all the I/O pins for testing. Over the past two to three decades, the number of I/O pins on most very

large scale integration devices has increased by an order of magnitude while the number of transistors has increased by as much as four orders of magnitude [6]. This has resulted in reduced accessibility of the ICs; making external test systems more complicated and expensive. BIST on the other hand is much simpler and inexpensive, as external test equipment is absent. Moreover, BIST techniques can be used at any level of testing ranging from manufacturing level testing to system level testing. Major drawbacks of the BIST technique are additional design requirements, area overhead and performance penalty [6]. The drawbacks of BIST are easily compensated by the advantages it offers. BIST has been successfully implemented in many digital logic designs and finds special use in testing of FPGAs.

1.3 FPGA BIST

The growing popularity of FPGAs in the VLSI industry has fueled research on new methodologies for testing these FPGAs. The re-programmability of FPGAs makes them harder to test as compared to regular structures. This is due to the fact that the FPGA can be operated and connected in many ways internally; as a result, it must be configured multiple times in order to be tested completely. But, due to the in-system re-programmability of the FPGAs, they can be configured to test themselves [6]. The idea is to program the BIST circuitry in a part of the FPGA and treat the rest of the FPGA as the CUT. Once the CUT is completely tested, a reversal of roles takes place, as the part of the FPGA used for BIST circuitry now becomes the CUT and vice versa. This process is illustrated in Figure 1.3.

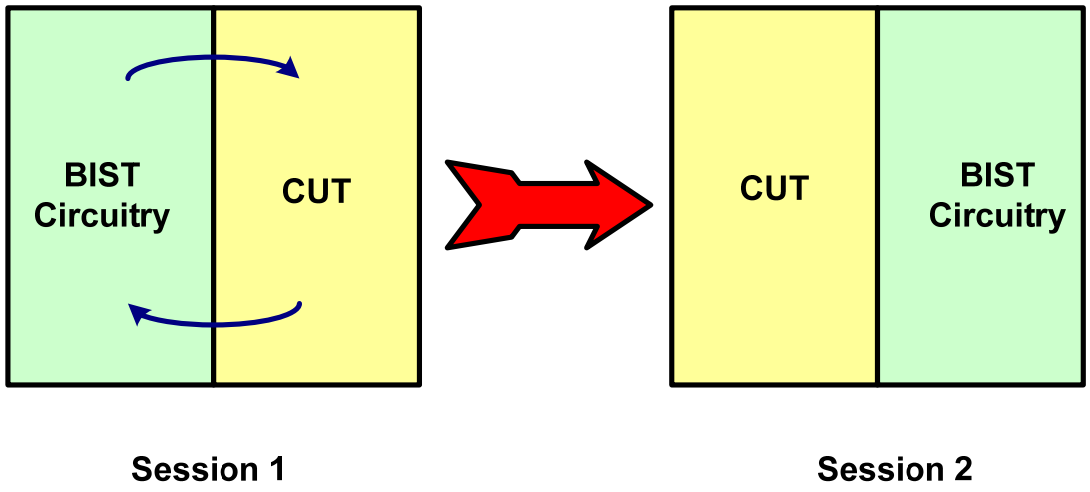

Figure 1.3 BIST in FPGAs

The BIST circuit can be designed in a number of ways to provide high resolution diagnostics for the FPGA, opening the door for fault-tolerant systems which was previously not possible with external test systems [3]. Moreover, BIST implemented in FPGAs does not suffer from any kind of area or performance overhead compared to conventional BIST techniques, as the BIST circuitry can be easily replaced by re-programming the FPGA with the system function after test [3].

Considerable work has been done in the area of BIST for FPGAs. Depending on the resources to be tested, some of the PLBs of an FPGA are configured as Test Pattern Generators and Output Response Analyzers, forming the BIST circuitry which tests the targeted resources in the FPGA [7]. An FPGA is reconfigured repetitively for testing and, as a result, a major portion of the time required to test the FPGAs is spent re-configuring them, i.e. downloading BIST configuration data into the FPGA.

The majority of an FPGA is comprised of routing and logic resources [15]. So not surprisingly, most of the research and development work done in the area of BIST for

FPGAs has been for its logic and routing resources [3] [7] [17] [20] [23] [27]. A generic approach cannot be used to completely test an FPGA since different fault models and test techniques are used to test logic and routing resources. The work presented in this thesis focuses on BIST for programmable logic resources only. Although considerable work has been done in the area of BIST for logic resources of an FPGA, the BIST technique for the testing logic resources presented in this thesis is most influenced by work described in [3] and [7]. A BIST approach for testing the PLBs of ORCA series FPGAs was presented along with a procedure for diagnosis and location of faulty PLBs in [3]. The BIST technique presented in [3] was extended to Xilinx XC4000 and Spartan series FPGAs to completely test their logic and routing resources in [7].

1.4 Xilinx FPGAs

The FPGAs used for the work presented in this thesis are Xilinx Virtex/Spartan-II and Virtex-4 FPGAs. Virtex/Spartan-II family of FPGA devices consist of primarily an array of PLBs, IOBs and memory blocks as shown in Figure 1.1 [8]. The Virtex-4 family of FPGAs combine a traditional FPGA with embedded processors, multipliers and high speed I/O interfaces in one package [9]. The architectural and operational features of these FPGAs can be exploited for implementation of BIST to speed-up the test time and also reduce the amount of memory required to store all the test configurations [16].

1.5 Thesis Statement

The research work presented in this thesis, primarily focuses on ways to improve BIST implementation for programmable logic resources of FPGAs. This involves

reduction in test time, improvement in diagnostic resolution and reduction in memory storage requirements for BIST configurations. This work builds upon the previous work done in the area described in [3] [7] [17] [18] [19] [29], extending to newer FPGA device families using techniques like partial reconfiguration and partial configuration memory readback. The target devices for this research are the Xilinx Virtex/Spartan-II and Virtex-4 family of devices. Configurations for BIST for programmable logic of Virtex/Spartan-II series FPGAs are developed along with methods to improve the test time. In case of the Virtex-4 family of devices, a set of BIST configurations for PLBs is developed and using architectural and operational features, further improvement in test time and reduction in configuration memory storage requirements is achieved.

The thesis is organized as follows: Chapter 2 describes the previous work done in the area of BIST for programmable logic resources and elaborates upon the architectures of Virtex/Spartan-II and Virtex-4 FPGAs. Implementation and experimental results of BIST for programmable logic resources in the Virtex/Spartan-II family of FPGAs is described in Chapter 3. Chapter 4 presents the implementation of BIST for programmable logic resources in the Virtex-4 FPGAs along with experimental results. Chapter 5 concludes the thesis with suggestions for future work and a discussion regarding the potential use of an embedded processor to assist in BIST.

CHAPTER TWO

Background

This chapter covers the background knowledge required to understand the research work presented in the following chapters. It begins with an overview of the architectures and configuration process of the Virtex/Spartan-II and Virtex-4 FPGAs used for the work presented in this thesis. This is followed by a discussion of prior work done in FPGA testing and BIST for testing the programmable logic resources of an FPGA. The chapter concludes with a restatement of the thesis goals.

2.1 FPGA Architectures

A typical FPGA consists of an array of PLBs, programmable interconnects, IOBs and RAM cores. The PLB array is interleaved with RAM cores and IOBs are arranged on the periphery as shown in Figure 1.1. Newer FPGAs have additional embedded cores like DSP cores, embedded microprocessors, and high-speed I/O interface for better system performance [13]. A design can be programmed into the FPGA by writing data to the configuration memory of the FPGA. The configuration memory then defines the function of the various programmable components of an FPGA. The following sub-sections describe the major components of Xilinx Virtex/Spartan-II and Virtex-4 series FPGAs.

2.1.1 Configuration Memory

All programmable devices have some kind of memory elements which connect or break connections in a programmable device to establish the desired functionality. Figure 2.1 illustrates a memory element that determines the connection between two lines [2].

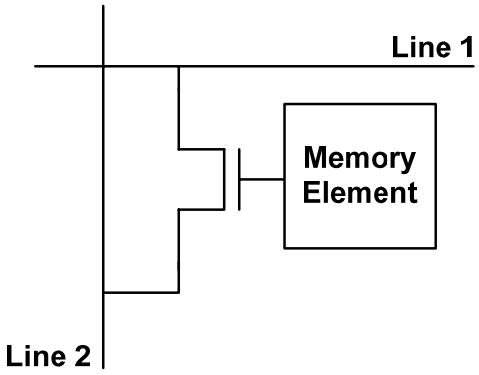

Figure 2.1 Configuration Memory Element

A memory element can be an anti-fuse, a floating-gate transistor, as in Read-Only Memory (ROM)/Flash memory, or a Static RAM (SRAM) cell. Most modern FPGAs use SRAM based memory elements which can be reprogrammed quickly in-system [2]. Xilinx Virtex/Spartan-II and Virtex-4 FPGAs are SRAM based, the drawback being volatile configuration memory. This means that the FPGA needs to be configured with the desired system function every time it is powered up, as the configuration memory elements lose their data on loss of power [11]. The configuration memory is spread across the entire device and is organized into smaller addressable segments called frames in the case of Xilinx devices. The size of the configuration memory varies depending on the size of the FPGA [11].

2.1.2 Programmable Interconnects, IOBs, Memory and DSP

All components in the FPGAs are connected using some type of routing resources; as a result the programmable interconnect network forms the biggest part of an FPGA [10]. The programmable interconnect network consists of wire segments that are connected or disconnected using Programmable Interconnect Points (PIPs), these PIPs are essentially switches controlled by configuration memory bits. A collection of these PIPs form a switch-matrix that is used in conjunction with wire segments to connect to various components of the FPGA like PLBs and RAMs. The routing resources of an FPGA are organized in a hierarchical manner that includes local, I/O, dedicated and global routing resources. Local routing resources include internal wire segments of a component for direct connections between adjacent components and switch matrices. I/O routing resources connect the internal components of the FPGA to the IOBs. Dedicated routing resources are used to implement high speed buses for better performance. Global routing consists of buffered nets used to route high-fanout signals like clock and reset [8] [9].

Over the years memory cores have become an integral part of the FPGA, as any kind of modern digital design requires storage capability. The memory cores, also known as BlockRAMs in Xilinx FPGAs, can be configured to operate in different modes depending on the data width and the size of the memory required. The BlockRAM in Xilinx FPGAs is a dual-port RAM that has two ports that can read and write to the memory simultaneously. To connect the FPGAs to the outside world, IOBs are provided which can be configured to be compatible with different IO standards, drive capabilities

and speeds [10]. Newer FPGAs have specialized DSP cores to implement high performance digital signal processing functions, these DSP cores typically consist of dedicated multipliers, adders and accumulators [10].

2.1.3 Programmable Logic Resources

The PLBs of Xilinx FPGAs are divided into smaller units of logic called *slices*. Each slice typically consists of a pair of logic cells, where a logic cell is comprised of a Look-Up Table (LUT), a storage element, some carry logic circuitry and multiplexers. Figure 2.2 illustrates a typical PLB slice of a Xilinx FPGAs. The LUT in Xilinx FPGAs can also be used to implement a shift register or a small RAM (16-bit for a 4-input LUT); these small RAMs are called distributed RAMs or Look-Up Table RAMs (LUT RAMs). Virtex/Spartan-II FPGAs have two identical slices per PLB [8] [12], whereas a Virtex-4 PLB consists of two different kind of slices, named SliceL and SliceM. The LUTs in SliceM can be used to implement LUT RAMs or shift registers, whereas LUTs in SliceL do not have this feature [10]. A Virtex-4 PLB consists of two SliceLs and two SliceMs for a total of four slices.

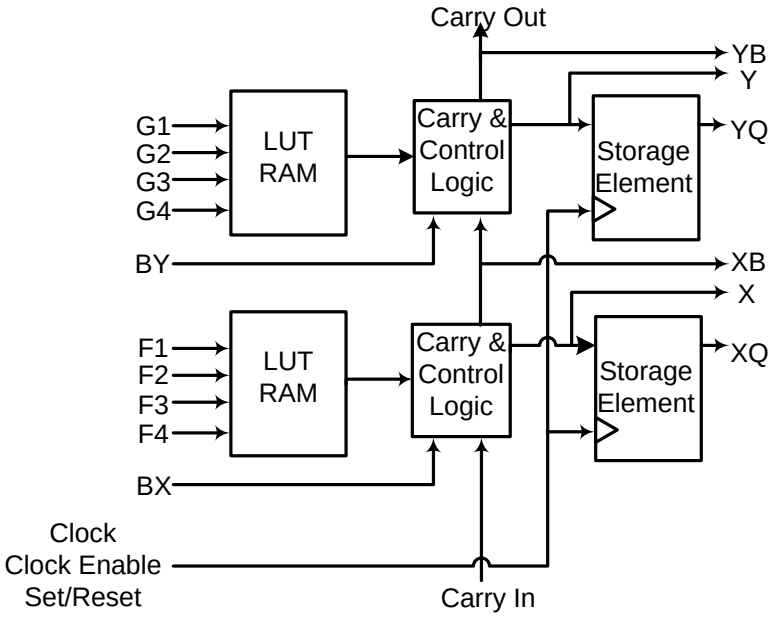

Figure 2.2 Typical PLB Slice of a Xilinx FPGA

Table 2.1 shows the various resources available in Virtex/Spartan-II and Virtex-4 families of FPGAs. Although Spartan-II and Virtex are separate families of FPGAs, Spartan-II is essentially derived from the Virtex architecture with fewer features and lower performance for lower cost. The Virtex-4 family of FPGAs is sub-divided into three sub-families:

- LX: for logic applications (higher logic resources)
- SX: for DSP applications (higher DSP resources)
- FX: for embedded applications (embedded processor, Rocket IO and Ethernet cores)

Table 2.1 Resources available in different FPGA families [8] [12] [9]

Resource	**Spartan-II**	**Virtex**	**Virtex-4 LX**	**Virtex-4 SX**	**Virtex-4 FX**
Largest PLB Array Size (Rows x Columns)	28 x 42	64 x 96	192 x 116	128 x 48	192 x 84
PLBs	1,176	6,144	22,272	6,144	16,128
Logic slices	2,352	12,288	89,088	24,576	64,152
Distributed RAM	74 Kbits	384 Kbits	1,392 Kbits	384 Kbits	987 Kbits
BlockRAMs	56 Kbits	184 Kbits	6,048 Kbits	5,760 Kbits	9,936 Kbits
I/O pins	284	512	960	640	896
DSP cores	-	-	96	512	192

2.2 Virtex/Spartan-II Architecture

The architecture of a Virtex/Spartan-II FPGA is shown in Figure 2.3. An array of PLBs and associated routing resources is at the core of the FPGA. A column of BlockRAMs is placed at the east and west edges of the PLB array. The IOBs and the Delay Locked Loops (DLL) for clocks are located at the periphery of the FPGA. The BlockRAMs and PLBs are surrounded by additional routing resources, primarily used to connect the internal resources of the FPGA to the I/O pins of the FPGA.

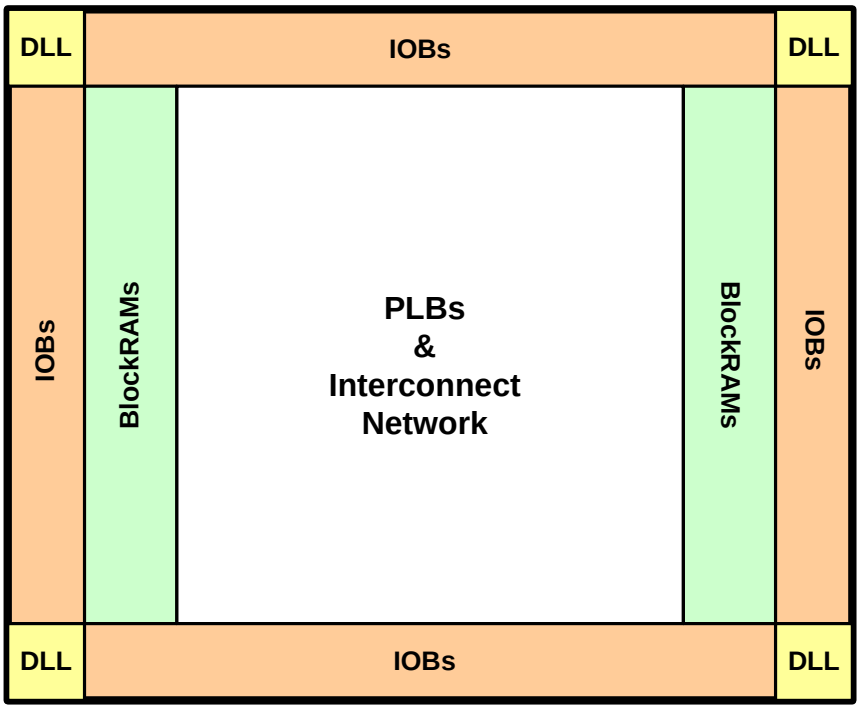

Figure 2.3 Virtex/Spartan-II Architecture

A Virtex/Spartan-II PLB consists of a pair of identical slices which are connected to a switch matrix as illustrated in Figure 2.4. The switch matrix is responsible for routing the signals in and out of the PLB. These PLBs also feature a carry chain that spans the entire column of PLBs. Each PLB slice has dedicated circuitry associated with the carry chain to implement fast arithmetic functions like an adder using look-ahead carry.

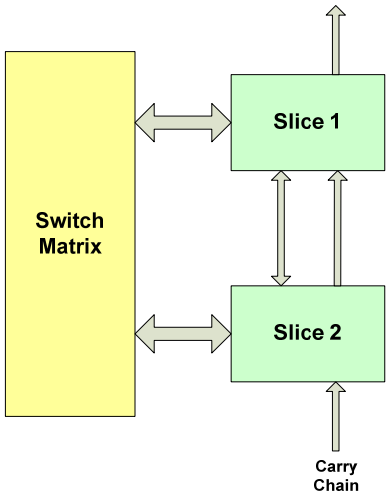

Figure 2.4 PLB of a Xilinx Virtex series FPGA

The internal architecture of a Virtex/Spartan-II PLB slice is illustrated in Figure 2.5. A single LUT of a Virtex/Spartan-II PLB can be used to implement any 4-input combinational logic function. It can also operate as a 16x1-bit RAM. Two LUTs of a PLB can be combined to form single-port 32x1-bit, 16x2-bit or a dual-port 16x1-bit RAM. The storage element can be operated either as a positive or negative edge-triggered flip-flop or as an active low or an active high level-sensitive latch. Storage elements have control signals including set/reset and clock-enable that are shared by all storage elements within a slice [8]. All four LUTs of the PLB can be combined using control logic and multiplexers provided in the PLB slices to implement any combinational logic function of up to six inputs. The PLBs feature dedicated logic like XOR gates and AND gates in order to implement fast arithmetic logic.[8].

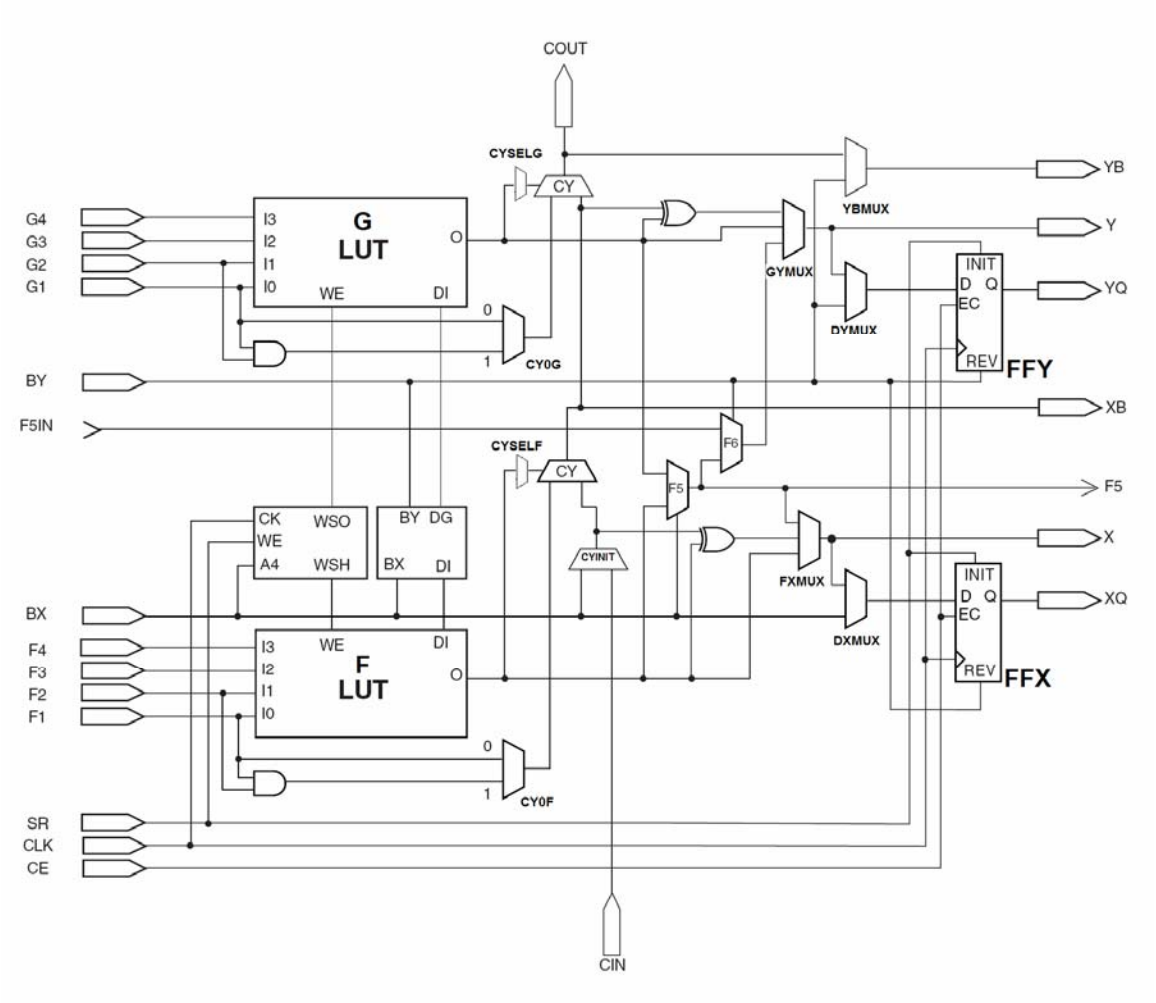

Figure 2.5 Virtex/Spartan-II PLB Slice [8]

The configuration memory of Virtex/Spartan-II FPGAs is divided into frames. The number frames per column of PLBs and associated routing is fixed at 48 frames as shown in Figure 2.6. The frame size varies from 12 words of 32 bits each for the smallest device to 39 words for the largest device in the Virtex family, depending on the number of PLB rows in the FPGA. The IOB frames are on the edges of the FPGA followed by the BlockRAM frames. The PLB frames also have some IOB configuration data at the

start and end of the frame. The Centre column consists of frame data for global clocks [32].

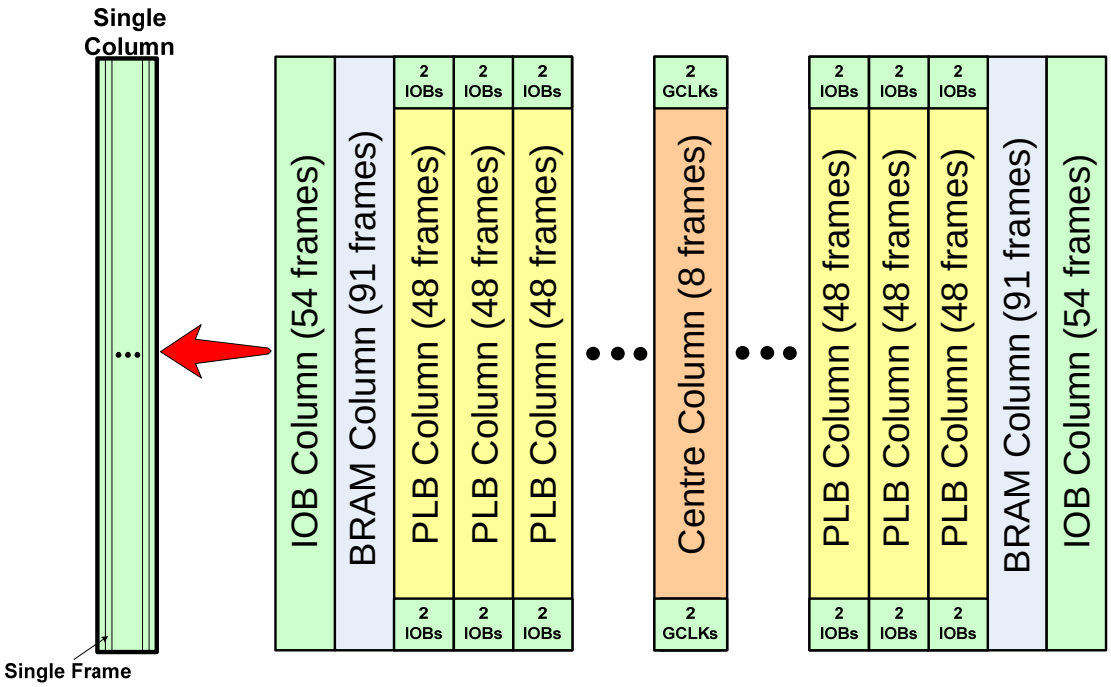

Figure 2.6 Configuration memory structure of Virtex FPGAs

2.3 Virtex-4 Architecture

The architecture of Virtex-4 FPGAs is different from Virtex/Spartan-II FPGAs, as illustrated in Figure 2.7. The PLBs and routing resources are spread across the entire FPGA. The I/O buffers are arranged in columns inside the FPGA, unlike Virtex/Spartan-II FPGAs that have IOBs only on the edges of the FPGA. Columns of BlockRAMs and DSP cores are interleaved with columns of PLBs. Virtex-4 FPGAs have up to 12 columns of BlockRAMs and 8 columns of DSP cores. The Virtex-4 FX family also features up to two embedded PowerPC cores.

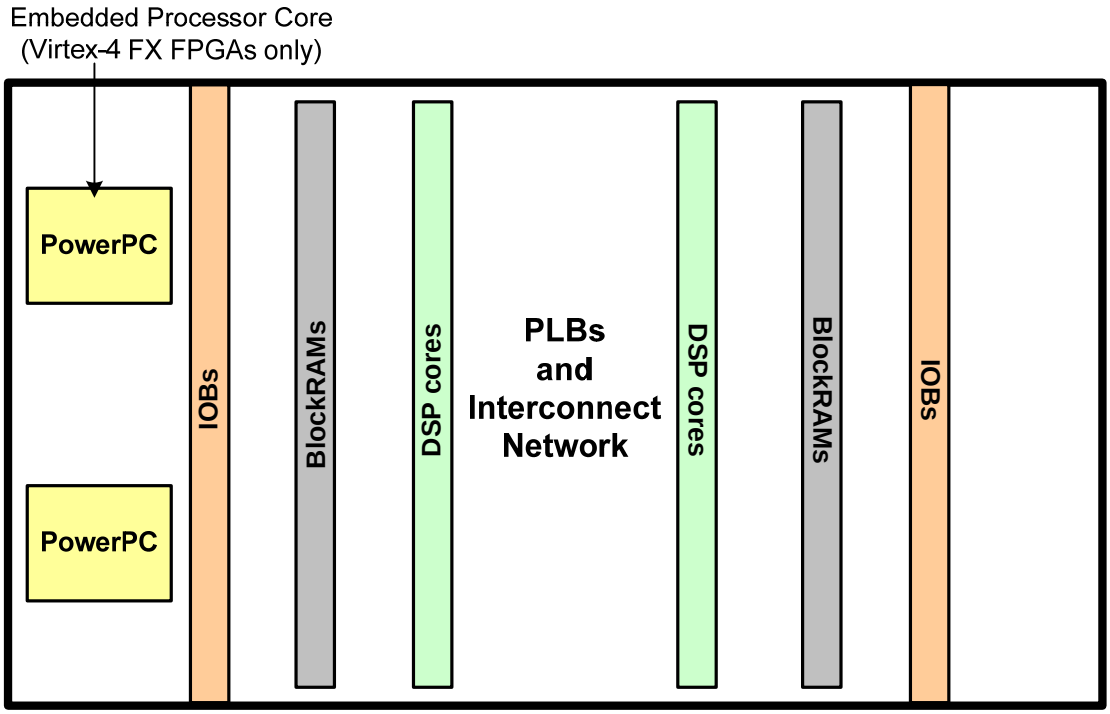

Figure 2.7 Virtex-4 Architecture

The PLB of a Virtex-4 FPGA is comprised of four slices, two SliceLs and two SliceMs as shown in Figure 2.8. All four slices are interconnected and similar slices are placed together in a column. Both pairs of slices have an independent carry chain spanning the entire column. The LUTs of SliceM also feature a Shift Register and a RAM mode of operation, consequently SliceMs feature a shift chain that can be used to combine SliceMs in single or multiple PLBs to form a long shift register [10].

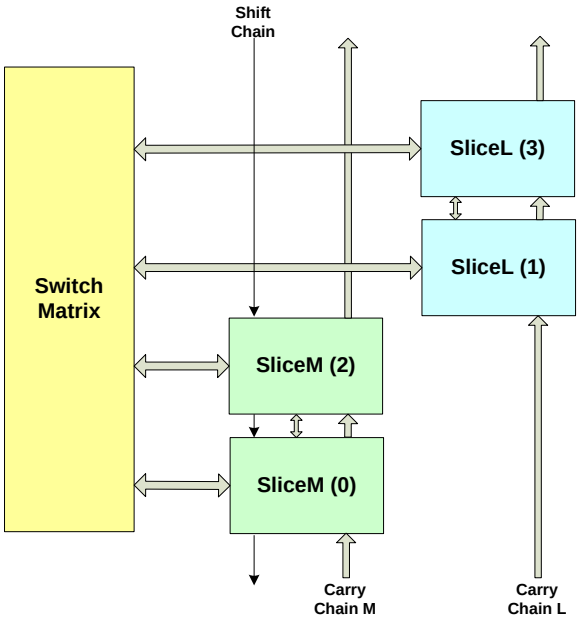

Figure 2.8 Diagram of a Xilinx Virtex-4 series FPGA PLB

The slices in Virtex-4 feature two 4-input LUTs, denoted F and G, two storage elements, carry logic, multiplexers and some arithmetic gates. The LUTs can be used as a 4-input LUT, up to a 16-bit shift register (SliceM only) or a 16-bit LUT RAM (SliceM only). The storage elements can be configured as positive or negative edge-triggered flip-flops or active high or active low level-sensitive latches with clock-enable control capability. They can be initialized to high or low value after download and set/reset synchronously or asynchronously during operation. Multiplexers present in the slices are used to cascade LUTs in multiple slices or PLBs to form up to 64x1 LUT RAM in a single PLB or a 64-bit shift register using a single PLB (multiple PLBs can be cascaded to form larger shift registers). A PLB has two carry chains that are directed vertically upwards, the carry chain logic in the slices is used to implement look-ahead carry functions. A pair of AND and XOR gates are provided in a slice as dedicated arithmetic

gates for efficient utilization of the logic resources [10]. The internal architectures of Virtex-4 SliceL and SliceM are illustrated in Figures 2.9 and 2.10, respectively.

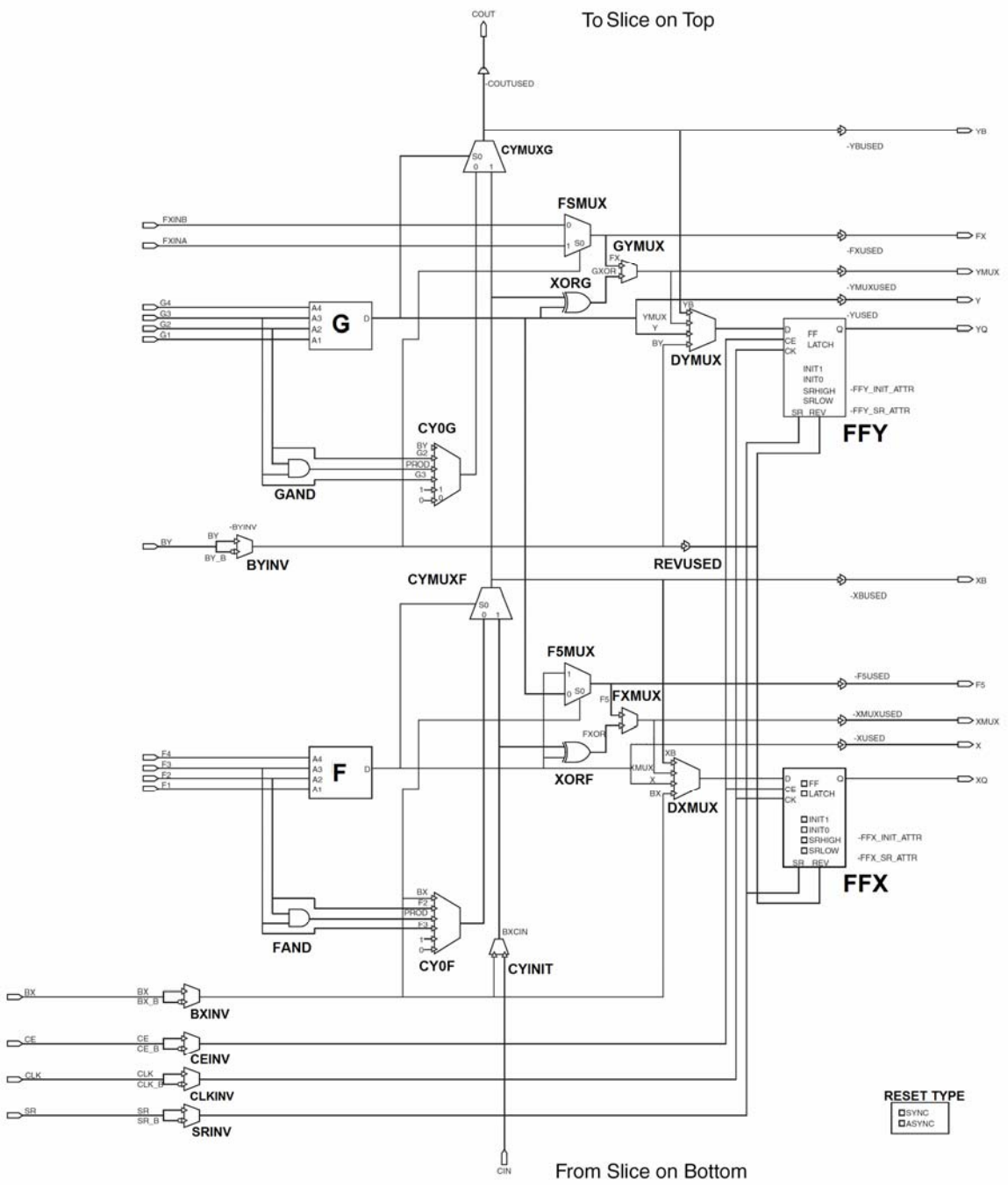

Figure 2.9 Virtex-4 SliceL [10]

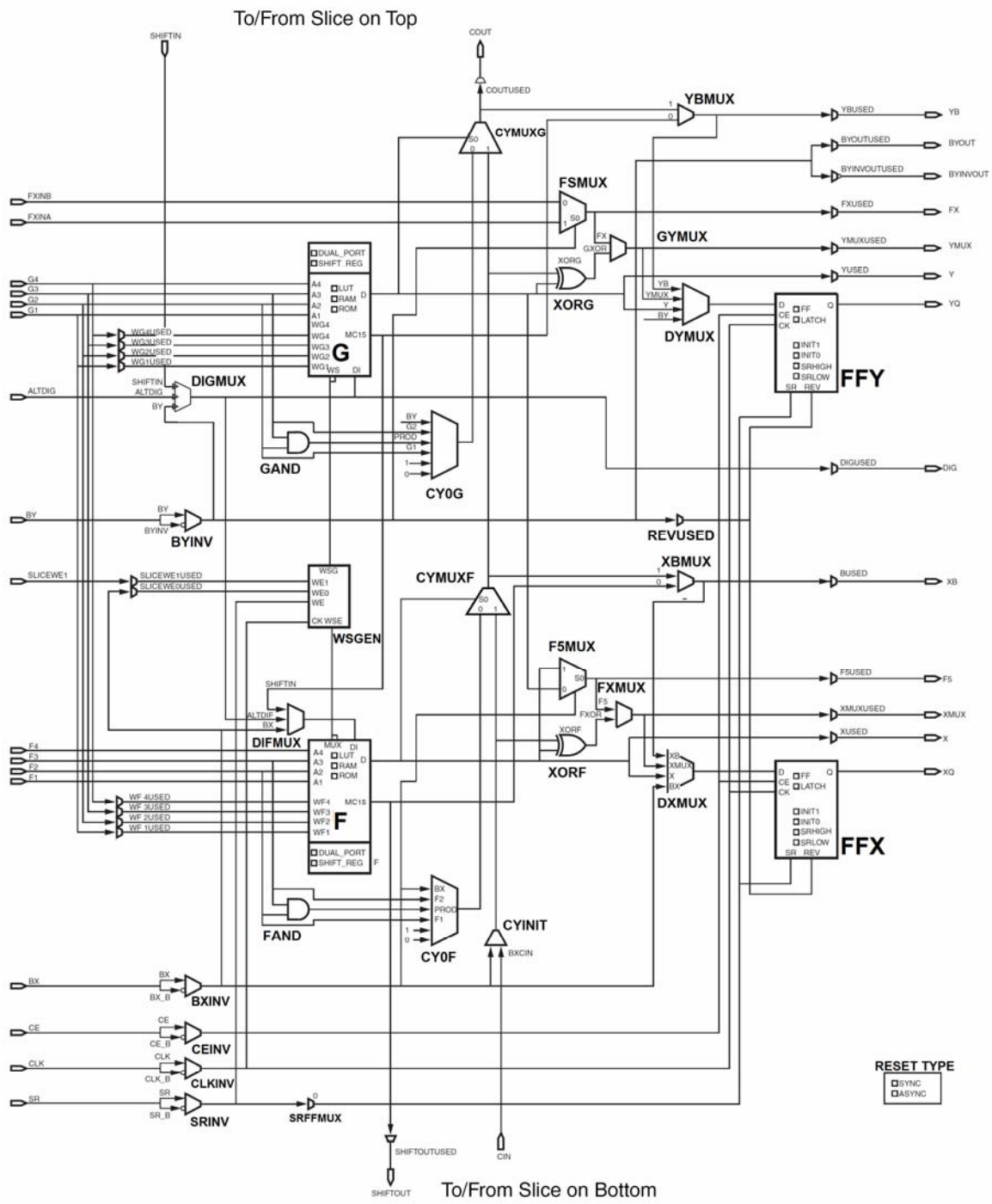

Figure 2.10 Virtex-4 SliceM [10]

The Configuration memory of a Virtex-4 FPGA is divided into frames of fixed size of 41 words of 32 bits each. These frames span a fixed number of rows of an FPGA column, unlike Virtex/Spartan-II frames that span the entire column. Frames are grouped together to form blocks based on the resources defined by them, like PLBs or BlockRAMs [11].

2.4 FPGA Configuration

A design is typically synthesized and converted to a configuration file or a bitstream that is downloaded into the FPGA to implement the required design [2]. Several interfaces are available to configure the FPGAs including Boundary Scan, dedicated serial interface and dedicated parallel interface [11]. As the FPGA devices grow bigger, the configuration file or bitstream size also grows. This leads to a longer time required to download a design.

Partial reconfiguration is a technique used to reduce the time required to reconfigure an FPGA. Once a full configuration for a design has been downloaded to the FPGA, minor changes in the design result in small changes in the bitstream. So instead of downloading the full bitstream, only parts of the bitstream that change are downloaded using partial reconfiguration [29]. In case of Xilinx FPGAs, a frame is the smallest unit of configuration memory that can be changed. One of the features of the newer FPGAs is dynamic partial reconfiguration. This feature allows the user to retain the flip-flop contents of the PLBs and IOBs during reconfiguration. Unused parts of FPGAs are reconfigured while the FPGA is operational with the system function [14].

2.4.1 Configuration Interface

There are three main configuration interfaces available in Xilinx FPGAs [11]. They are:

1. Master/Slave Serial interface
2. Master/Slave Parallel (SelectMAP) interface
3. Boundary Scan interface

The source of clock used for configuration determines whether the interface is in master mode or slave mode. If the source for generation of the configuration clock is external to the FPGA then the configuration mode is in slave mode. In master mode the configuration clock is generated internally by the FPGA. The configuration interface to be used for Xilinx FPGAs is determined by the value set on three mode pins of the FPGA [11].

One bit is downloaded to the FPGA per clock cycle when the serial interface is used. In case of the SelectMAP interface, configuration data is downloaded in parallel. The configuration data can be both downloaded to the FPGA or read back from the FPGA using the SelectMAP interface. The SelectMAP interface is capable of reading or writing 8 or 32 bits per clock cycle in parallel, greatly reducing the configuration download time as compared to the serial interface [11]. The third interface is known as Boundary Scan [5] [30]. It was originally developed to test the integrity of the connections between devices on a printed circuit board. Xilinx FPGAs make additional use of the Boundary Scan interface to download to or read back from the FPGA configuration memory [11].

The Boundary Scan interface consists of a 4-pin Test Access Port (TAP), TAP controller, instruction register and decoder, bypass register and Boundary Scan register as shown in Figure 2.11.

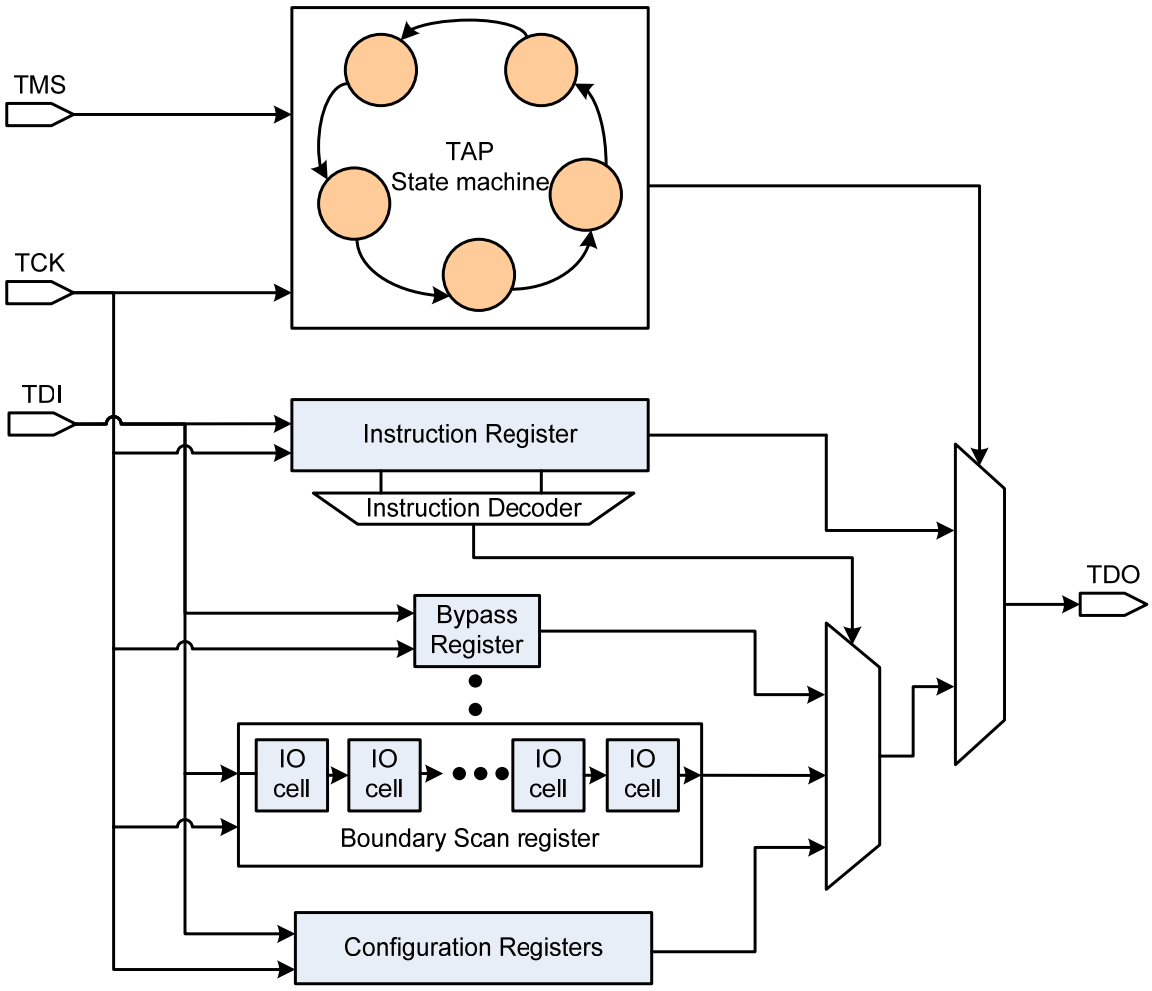

Figure 2.11 Boundary Scan Architecture

The TAP controller consists of a finite state machine that is controlled by four TAP pins, namely TCK (Test Clock), TMS (Test Mode Select), TDI (Test Data In) and TDO (Test Data Out). Depending on the state of the TAP controller, instructions or data

can be loaded into the Boundary Scan interface registers. All the IOBs are interconnected to form a Boundary Scan register in test mode and test vectors are loaded to the IOBs using TDI. Similarly TDO is used to read out the test results from the IOBs. The bypass register is a single bit register used to put the device in bypass mode to access other devices connected in the Boundary Scan chain. The instruction register and decoder are used to execute the Boundary Scan test instructions [11]. Most of the current FPGAs allow configuration download using the Boundary Scan interface. Xilinx FPGAs implement Boundary Scan instructions that allow both configuration memory download and readback by using configuration registers, like the frame data register and frame address register. The configuration memory can be written using a CFGIN command and read back using a CFGOUT command [11].

Xilinx also provides user access to the FPGA core from the Boundary Scan interface via Boundary SCAN (BSCAN) modules in the FPGA. These user access modules can be used to create internal Boundary Scan chains to implement user-defined functions in an FPGA. The BSCAN modules have to be activated using Boundary Scan commands before they can be used to perform a user function. All BSCAN modules source the clock from the TCK pin in the Boundary Scan interface and the clock for a given BSCAN module is enabled only when it is activated. BSCAN modules also consist of output pins that indicate the status of the Boundary Scan interface [11] [32] [33].

2.4.2 Configuration Process

The configuration of a Xilinx FPGA is a multi-stage process. Before the download of configuration data, the FPGA is initialized, which involves synchronization

of the configuration interface logic of the FPGA with the configuration data to be downloaded. It may also include clearing of the configuration memory. A Cyclic Redundancy Check (CRC) is performed on the configuration data to check for errors while data is downloaded to the configuration memory. The final step is known as the startup sequence, it is a multi-step process that includes activation/de-activation of global signals like global set/reset (GSR), global write-enable for all the RAMs and flip-flops in the FPGA and global tri-state enable for all the IOBs [11].

For full configuration, all the frames in the FPGA are written with configuration data, whereas for partial reconfiguration only the frames that change are rewritten. The configuration process is similar for both methods except for the initialization. The configuration memory is not cleared during initialization of the FPGA using partial reconfiguration. During configuration download, the frame address register (FAR) is written with the address of the frame to be written and 32-bit words of configuration data are written to the specified frame in the configuration memory via the frame data register (FDR) [11].

2.4.3 Configuration Memory Readback

Xilinx FPGAs allow the user to read back contents of the complete configuration memory of the FPGA. This can be used to verify the configuration bits downloaded into the FPGA. Instead of full configuration memory readback, parts of the configuration memory can also be read back; this procedure is known as partial configuration memory readback [14]. A frame is the smallest unit of configuration memory that can be read using partial configuration memory readback. For reading a frame, the frame address is

written in the FAR and the configuration data is read out from the FDR using an external interface like SelectMAP or Boundary Scan [33]. Xilinx FPGAs are also capable of capturing the contents of the BlockRAMs and flip-flops of the FPGA during configuration memory readback [11]. In the case of Xilinx FPGAs, a CAPTURE module needs to be instantiated in the design in order to perform configuration memory readback [11] [33].

2.5 Prior Work in FPGA Testing

This section lists some of the work previously done in the area of FPGA testing. Major work related to testing of logic resources of an FPGA is presented in [3] [7] [16] [17] [18] [21] [22] [23] [24] [28]. Stroud et al. present a method to evaluate the number of configurations required to test all the logic resources of an FPGA in [21]. The most comprehensive works in testing of programmable logic resources using a BIST approach were presented in [3] [7]. Reference [7] extends the work done in [3], which was done using Lucent's ORCA series FPGAs to Xilinx 4000 series FPGAs. Reference [3] also describes an algorithm called MULTICELLO that can be used for diagnosis of faulty PLBs in FPGAs. The work done in [3] and [7] laid the basis for the work presented in this thesis.

Ideas were derived from work done in [17] [18] [19] to improve the BIST approach to test logic resources for better diagnostic resolution and faster test times. Abramovici et al. introduced a new concept of self-testing areas that are used to implement BIST in small unused areas of the FPGA, while the rest of the FPGA is operational with the system function in [17] [18]. This work inspired the use of dynamic

partial reconfiguration to achieve test time speedup. A technique to test embedded cores of SoCs that include regular structures like RAM and multiplier cores, using the FPGA core, is explained in [19]. This work introduced the concept of circular comparison that results in higher diagnostic resolution.

Reference [28] by Wang et al. presents an alternative technique to test logic resources using BIST. A non-BIST based approach to test an FPGA that uses an external Programmable Read Only Memory (PROM) to store the test configurations and test vectors is presented in [22] by Huang et al. Reference [23] introduces another technique to externally test the logic resources of Xilinx 3000, 4000 and 5000 series FPGAs.

References [7] [16] [17] [18] [20] [22] [24] [25] [26] [27] present some of the work done in the area of FPGA interconnect testing. Reference [20] expands upon the BIST technique used in [17] and [18] to test FPGA interconnects. Renovell et al. [25] and Wang et al. [27] present techniques to externally test the interconnects of an FPGA. A BIST approach to test the interconnects of an FPGA using small BIST structures known as BISTERs is presented in [26] by Harris et al. Renovell et al. describe a technique to test the Xilinx FPGAs by dividing the FPGA into separate arrays of logic and interconnects and LUT RAMs in [24].

References [16] and [19] describe comprehensive work done in testing all the resources of the FPGAs, including the embedded cores of FPGAs like RAMs and multipliers. Stroud et al. presented a case study that uses Atmel's FPGA based SoCs to present the implementation of BIST to completely test the logic, interconnect and memory resources of an FPGA [16].

2.6 General BIST Architectures

There are two primary approaches for testing an FPGA using BIST. One approach is to configure the complete FPGA with BIST circuitry to test itself and replace it with the original system function after the device has been tested; this method is known as off-line testing since the system function of the FPGA is halted to test the FPGA [6] [3]. This scheme is discussed in detail in the next section. The other option is to keep the system operational while testing unused portions of the FPGA by configuring them as Self Testing AReas (STARs). These STARs are moved around the FPGA using dynamic partial reconfiguration of the FPGA as shown in Figure 2.12. This scheme is called on-line testing because the system is on-line or operational even when the device is being tested [17] [18].

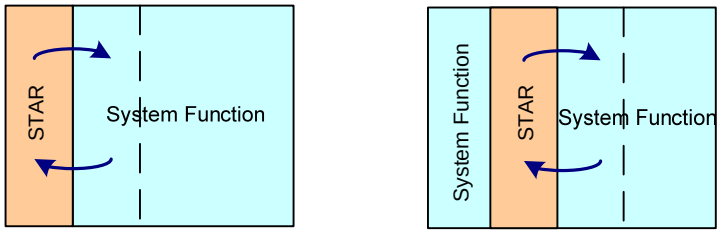

Figure 2.12 On-line BIST [6]

2.7 BIST for Logic Resources of an FPGA

This thesis deals only with off-line testing of programmable logic resources, hence the CUT in this case is the array of PLBs in the FPGA. BIST for testing logic resources from here on shall be referred to as **Logic BIST**.

2.7.1 BIST Architecture

BIST circuitry comprises Test Pattern Generators (TPGs) and Output Response Analyzers (ORAs). The TPGs generate the test patterns required to test the PLBs. The ORAs essentially compare the outputs of two identically configured PLBs under test, also called Blocks Under Test (BUTs), and record any mismatch due to a fault. Since PLBs are required to implement the BIST circuitry, all logic resources of an FPGA cannot be tested simultaneously. If half of the PLBs can be configured as BUTs, then only two test sessions are required to completely test all the PLBs. So, in the first session half of the PLBs are configured as Blocks Under Test (BUTs) and the rest are configured as TPGs and ORAs. In the second session they are swapped, i.e. the PLBs that were BUTs in first session now become TPGs and ORAs and vice versa, as illustrated in Figure 2.13 [3][7]. The two test sessions are called West and East sessions depending on the location of the TPGs, shown in Figure 2.13 (a) and (b) respectively. This scheme shows a column based arrangement of BUTs, TPGs and ORAs, but it can also be row based and the two test sessions are then called North and South sessions [3].

The BUTs are located in alternate columns of the FPGA with an ORA column sandwiched between every two columns of the BUTs such that they compare the outputs of the BUTs in the neighboring columns. The ORAs latch any mismatch between the BUT outputs being compared as a result of a fault. The fault can be associated with either of the two BUTs compared by the ORA. The ORAs are connected in a scan chain as illustrated in Figure 2.13. The BIST results can be shifted out after the BUTs have been tested using the scan chain [3]. To completely test the PLBs, they are reconfigured and tested in different modes of operation while keeping the BIST architecture untouched. A

test phase is a configuration that tests a PLB in a single mode of operation. A group of test phases that test a PLB in all of it's modes of operation form a *test session* [3]. The PLBs are also tested in their LUT RAM mode of operation which tests the logic in the PLBs associated with LUT RAMs [7] [37]. BIST results of faulty devices can then be analyzed using MULTICELLO [3] to determine the exact location of the faulty PLB.

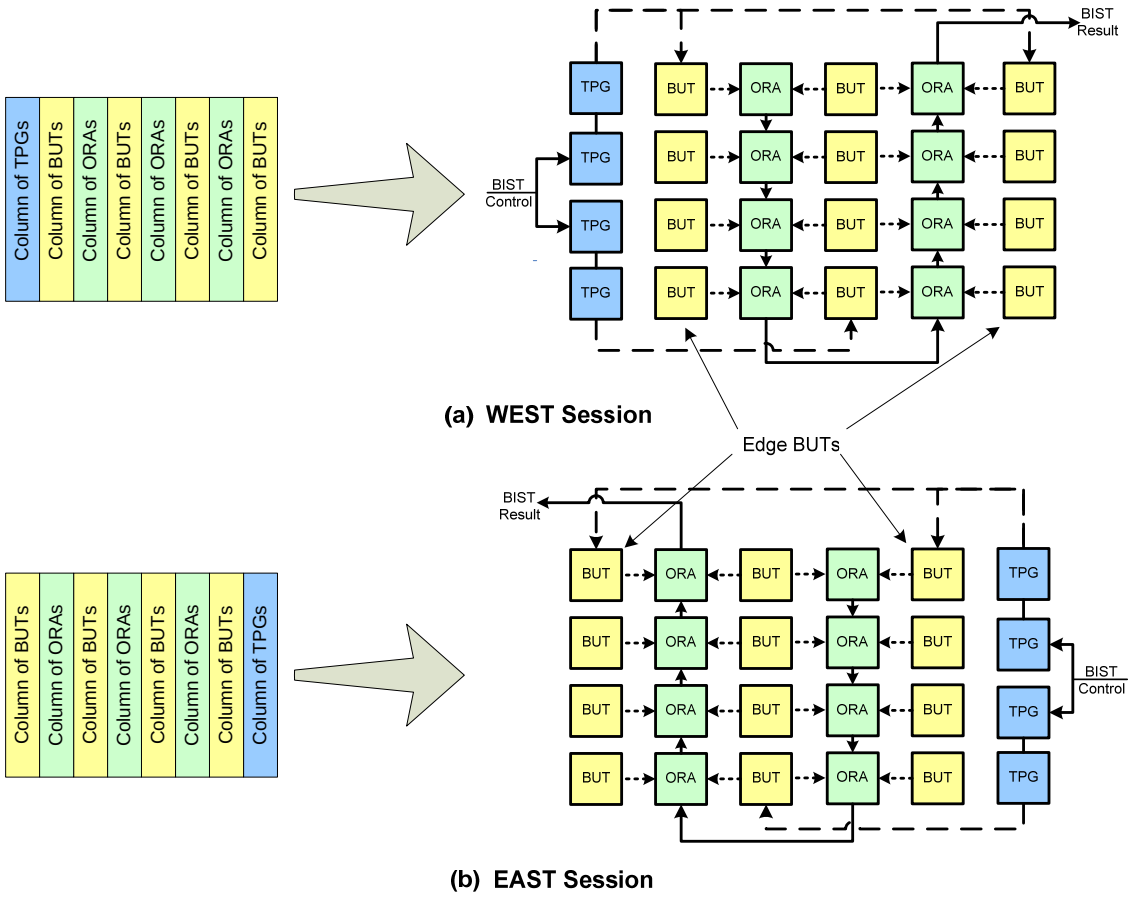

Figure 2.13 BIST Architecture to test Logic Resources [3]

It can be noticed from Figure 2.13 that BUTs in the edge columns of the BIST architecture suffer from lower diagnostic resolution, as they are compared by only one

ORA, whereas the rest of the BUTs are compared by two ORAs. Reference [19] introduces a circular comparison technique that allows the comparison of every BUT by two ORAs, thereby increasing the diagnostic resolution.

2.7.2 Test Pattern Generation and Output Response Analysis

As the number of inputs of a PLB is small, exhaustive test vectors can be used to test them. A simple counter or a linear feedback shift register (LFSR) can be implemented as a TPG using very few PLBs to generate exhaustive test vectors [5] [6]. An LFSR is more commonly used because it can generate pseudo-random patterns and utilizes fewer gates as compared to a counter [6].

Two identical TPGs drive alternate columns of BUTs in the FPGA such that every ORA compares the output response of BUTs that receive input patterns from two different TPGs. This ensures that even if one of the PLBs used as a TPG is faulty, the ORAs record a mismatch as both the TPGs generate different test patterns. Hence, using two TPGs improves fault detection because if a single faulty TPG was driving all BUTs, the ORAs would have never recorded a mismatch [3]. TPG loading is an issue in this BIST architecture, since a large number of BUTs are connected to a single TPG. The large loading on a TPG output limits the maximum operating frequency of the BIST architecture. Solutions proposed are to either use drivers for TPG signals or split the FPGA into smaller sections with independent pairs of TPGs that are tested in parallel [7]. Both the schemes limit TPG loading without increasing the number of configurations.

The comparison based approach has better fault detection capability compared to signature analysis, as the response of the BUTs is not compacted. Instead it is compared,

so unless there are equivalent faults in certain extremely rare cases the faults are guaranteed to be detected [3]. A comparison based ORA is illustrated in Figure 2.14 (a). It uses an XOR gate to detect any mismatch and the feedback from the flip-flop to the OR gate latches a '1' into the flip-flop in case of a mismatch. The multiplexer is used to form a scan chain of ORAs in order to scan out the BIST results after every test phase [3]. A good circuit is represented by a '0' and a fault is indicated by a '1' stored in the ORA flip-flop. Configuration memory readback can be used instead of using a scan chain to retrieve BIST results. Figure 2.14 (b) shows the ORA without the scan chain logic used in this case [7].

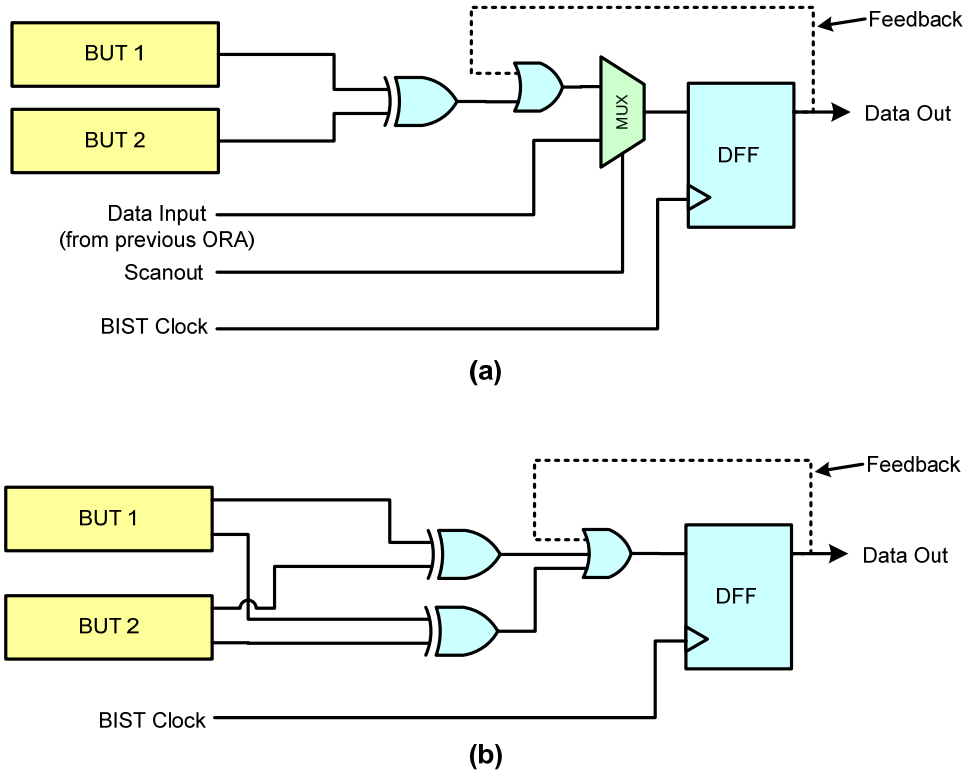

Figure 2.14 Output Response Analyzer

2.7.3 Configuration Schemes

The BIST approach described in the initial work [3] [7] uses complete reconfiguration of an FPGA to switch between test phases. Full reconfiguration is highly time consuming and in the case of Logic BIST, only the BUT configurations change from one test phase to the next for a given test session. The use of partial reconfiguration to reconfigure only the BUTs of the FPGA, to operate in a different mode of operation for a given test session, was proposed in [29]. Partial reconfiguration reduces the memory required to store the test configurations and leads to faster test times, since less configuration data is downloaded into the FPGA per test phase.

2.7.4 Results Retrieval

After execution of a test phase the BIST results have to be read out of the ORAs in the FPGA. As described in [3], the ORAs form a scan chain and the BIST results can be shifted out using the *Scanout* signal shown in Figure 2.14 (a). The data input of the ORA at the tail of the scan chain is tied to a '1', so there is a trail of ones at the end of the BIST results scanned out which serves as an indicator for the end of the scan chain and serves as a check for correct operation of the ORAs.

Most FPGAs have the ability to capture the contents of flip-flops in the PLBs during configuration memory readback. This feature can be used to retrieve the BIST results captured in the flip-flops of PLBs configured as ORAs by reading back the configuration memory. Although configuration memory readback increases the testing time per test phase it reduces the total number of test configurations and improves the diagnostic resolution [7]. Due to the limited resources of some FPGA PLBs, not all outputs of a

BUT can be observed in a single configuration, therefore there are multiple configurations for every test phase. If configuration memory readback is used then the scan chain to retrieve BIST results can eliminated from the ORA, making extra logic resources available in the PLBs to be used as ORAs. The extra logic resources in the ORAs make it possible in some FPGAs to compare all the BUT outputs in a single configuration for a test phase. Figure 2.14 (b) illustrates the ORA without the scan chain logic that is capable of comparing more BUT outputs and also has fewer control signals than the ORA with the scan chain logic.

2.8 Restatement of Thesis Goals

A significant amount of work has been done in the area of Logic BIST for FPGAs. As the technology advances and the feature sizes shrink, FPGAs grow larger in size and feature many more capabilities, compared to their predecessors. The testing time increases as the size of the FPGA grows, so new methods and architectural features have to be used in order to keep the test times and the memory storage requirements to a minimum.

The work presented in this thesis builds upon the work previously done in [3] [7] [17] [19] [29] and introduces some new techniques to implement Logic BIST for newer FPGA devices. As the majority of the time required for testing FPGAs using BIST is spent on their reconfiguration, emphasis is put on techniques like partial reconfiguration and partial configuration memory readback to reduce the test time and configuration memory storage requirements for Logic BIST. This chapter introduced the basic concepts and overview of the previous work done, required to understand the work presented in

the following chapters. Chapter 3 presents the implementation of Logic BIST on Virtex/Spartan II series FPGAs, along with the use of partial reconfiguration and partial configuration memory readback to achieve speedup in test time and reduction in memory storage requirements. Chapter 4 presents the implementation of Logic BIST for Virtex-4 FPGAs. The Logic BIST architecture was modified for Virtex-4 to achieve better diagnostic resolution. In both cases, the PLB slices were modeled and, based on the resultant fault simulations, Logic BIST configurations were developed to test the PLBs. Chapter 5 concludes with a summary and suggestions for future improvements along with a discussion regarding the use of an embedded processor for BIST.

CHAPTER THREE

Logic BIST for Virtex/Spartan II

3.1 Introduction

This chapter discusses the implementation of Logic BIST for Xilinx Virtex and Spartan-II FPGAs. The details regarding the Logic BIST architecture and test configurations are described along with the fault coverage of the logic resources. The methods to achieve speed-up in test time and reduction in memory storage requirements are also discussed. These methods include techniques like partial reconfiguration and partial configuration memory readback, which reduce the configuration download time and BIST results retrieval time, respectively. The experimental results of all the methods employed are presented with a summary and analysis of the results to conclude the chapter. The work presented in this chapter is primarily the work presented in [31] with some additional details.

3.2 Virtex/Spartan-II PLB Architecture

An overview of the Virtex/Spartan-II FPGA architecture was presented in Chapter 2. In this chapter, additional details regarding the internals of the PLB are provided. Figure 2.8 shows the block diagram of a PLB slice of a Virtex/Spartan-II FPGA [8]. It consists of two 4-input LUTs, F and G, that can also function as 16-bit LUT RAMs or

16-bit shift registers. For the RAM or shift-register modes of operation, additional circuitry is provided to generate the write enable signals. Two AND and two XOR gates are provided in each PLB slice to efficiently implement arithmetic functions. Multiplexers CY and input CIN are used to implement the carry chain logic. A slice has two storage elements, FFX and FFY, which can be used either as flip-flops or as latches to implement sequential circuits. Multiplexers DXMUX and DYMUX are provided to choose the data input for the storage elements FFX and FFY, respectively. Multiplexers F5 and F6 are used to combine LUTs to implement combinational logic functions with five or six inputs using a single PLB [8].

3.3 BIST Architecture

Logic BIST for Virtex/Spartan-II builds upon previous work done on Lucent's ORCA and Xilinx 4000 series FPGAs, as described in [3] and [7], respectively. The BIST approach is very similar to those described in [3] and [7], as illustrated in Figure 2.13. It is modified with focus on partial reconfiguration and partial configuration memory readback.

The BIST architecture can be either row-oriented or column-oriented, but column-oriented BIST architecture emerges as the more efficient BIST implementation for Virtex/Spartan-II FPGAs for three major reasons. Firstly, the carry chain implemented between the PLBs is implemented vertically upwards within each column, so in order to test logic resources associated with the carry chain the BIST architecture has to be column-oriented. Secondly, dedicated local routing is available for making direct connections between horizontally adjacent PLBs [8]. Therefore it is easier to make

BUT to ORA connections across rows in a column-oriented BIST architecture without any routing issues. Lastly, the structure of the configuration memory is also column-oriented. As mentioned earlier, configuration memory is comprised of frames and it takes multiple frames to configure a column of PLBs and their associated routing in an FPGA, as illustrated in the Figure 2.6. A column-oriented BIST architecture aids in reducing the number of frames to be written using partial reconfiguration and read using partial configuration memory readback for retrieval of BIST results.

Figure 3.1 illustrates the architecture of Logic BIST for Virtex/Spartan-II FPGAs. Two identical TPGs are restricted to one column of the FPGA and alternate columns are configured as ORAs and BUTs. The TPG is a 12-bit LFSR that generates pseudo-exhaustive test vectors, providing identical vectors to both the slices of each PLB configured as a BUT. Each TPG provides identical input patterns to alternate BUT columns, which improves fault detection in case of a faulty TPG [3]. The ORAs compare the outputs from the two neighboring BUTs that get identical test patterns from two different TPGs. BIST results after testing are either scanned out or captured in the configuration memory. Figure 3.1 (a) shows the ORAs connected in a scan chain that allows scanning out of BIST results. Figure 3.1 (b) illustrates the architecture in which the BIST results are captured in the configuration memory and retrieved using configuration memory readback.

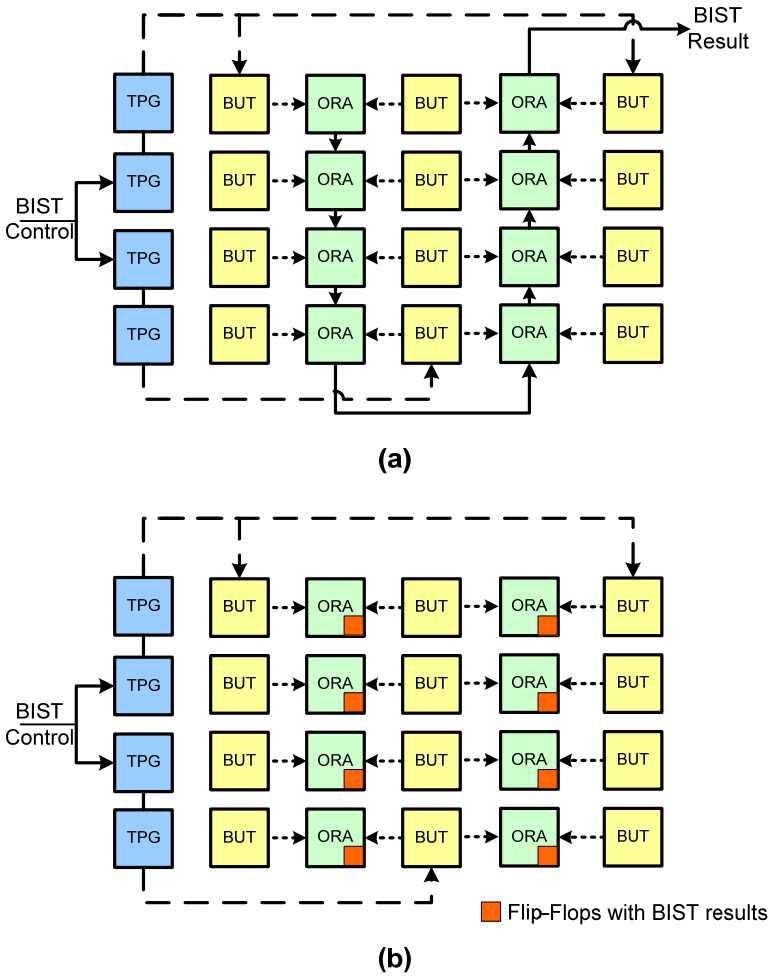

Figure 3.1 Logic BIST Architecture for Virtex/Spartan-II FPGAs

Two *test sessions* are required to test all the logic resources. BUTs in each test session are configured in different modes of operation in order to be tested completely; these configurations are called *test phases*. In a PLB only 12 out of the 16 outputs can be observed, as four outputs related to carry and multiplexer logic cannot be routed out of the PLB. The limited logic resources of a PLB allow a maximum of five BUT outputs to be observed by an ORA in a single configuration. Therefore a set of test phases has to be repeated three times, each time looking at a different set of four BUT outputs for a total

of twelve BUT outputs as shown in Figure 3.2 (a). This set of test phases is called a *slice test set*. To retrieve the BIST results, if readback is used instead of implementing a scan chain of ORAs, the ORA can be modified to compare six BUT outputs in a single configuration as shown in Figure 3.2 (b), where the logic resources of an ORA previously used for scan chain implementation are now used for comparing more BUT outputs. As a result the number of slice test sets can be reduced from three to two, where one slice is tested in each slice test set.

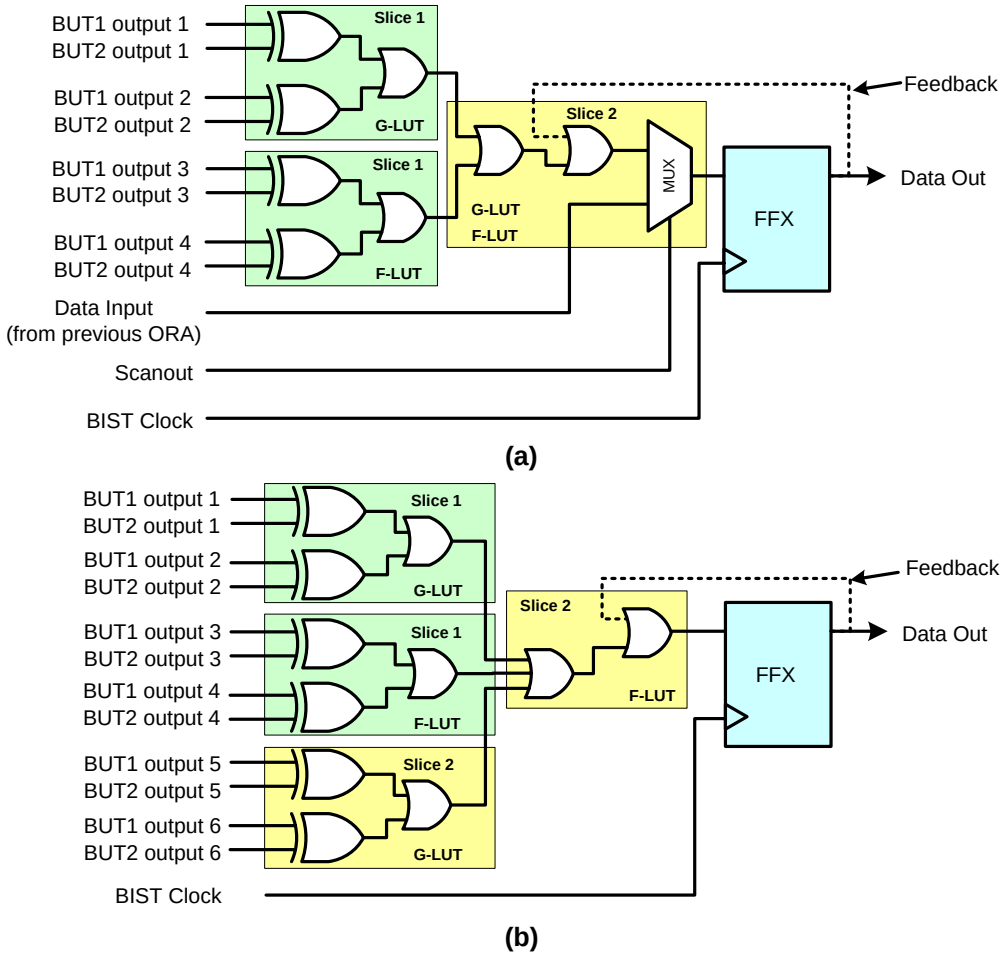

Figure 3.2 Output Response Analyzers

The Boundary Scan interface was used for the implementation of Logic BIST. The frame address register is written with the address of the frame to be written or read and 32-bit words of configuration data are written to or read from the frame data register, depending on the operation being performed. Xilinx provides two user access registers in Virtex/Spartan-II FPGAs that can be used by invoking a Boundary Scan module (BSCAN_VIRTEX). For Logic BIST, user access register 1 was used to source the BIST clock for BUTs, TPGs and ORAs from the Boundary Scan interface and user access register 2 was used to generate a reset signal for all the TPGs and ORAs.

There are two test sessions: East and West, each testing half of the PLBs. To completely test a PLB, except for the case when it is configured as LUT RAM, a total of seven different test configurations of a PLB are required. Therefore the total number of Logic BIST configurations depends on the method used for BIST results retrieval.

- Scan chain method: 2 sessions x 7 phases x 3 slice test sets = **42** configurations
- Readback method: 2 sessions x 7 phases x 2 slice test sets = **28** configurations

3.4 Partial Reconfiguration

Using partial reconfiguration, only BUT configurations are changed in a given test session. Most of the 48 frames of a PLB column are associated with routing resources rather than BUT configurations. So, in order to reconfigure the BUTs, a small number of frames per PLB column in only the columns of BUTs have to be rewritten with new configuration data. After the first test configuration is downloaded for a test session, the rest of the configurations can be partial reconfigurations.

The sequence in which the test configurations are applied is crucial for keeping the partial reconfigurations small, as discussed in [29]. Since multiple slice test sets are required for each test phase, three scenarios were investigated regarding the sequence of configurations to be applied:

Scenario 1. For a given test session, the configuration of both the slices is kept fixed but the BUT outputs compared by the ORAs are changed. Therefore each test phase consists of two or three slice test sets, depending on the BIST results retrieval technique used.

Scenario 2. For a given test session, the BUT outputs compared by the ORAs are kept fixed and the configurations of both PLB slices are changed. Therefore each slice test set consists of seven test phases.

Scenario 3. For a given test session, the BUT outputs compared by the ORAs are kept fixed and the configuration of only the slice whose outputs are being compared is changed, while maintaining the first configuration in the other slice. Therefore each slice test set consists of seven test phases and each test session has two or three slice test sets depending on the BIST results retrieval method used.

Partial reconfiguration is not effective in reducing the configuration file size when routing changes from one configuration to the next, as frames related to interconnects comprise the majority of the total number of frames in the FPGA. Consequently, the third scenario turns out to be most effective [29]. The sequence in which the test phases are applied can also be optimized to reduce the difference between consecutive test configurations, thereby reducing the partial reconfiguration file size.

3.5 Partial Configuration Memory Readback

Partial configuration memory readback can be used instead of using full configuration memory readback or scan chain to retrieve BIST results. Full configuration memory readback reduces the number of slice test sets from three to two but it takes the amount of time comparable to full configuration. On the other hand, scan chain implementation only requires a few clock cycles (equal to the number of ORAs) to retrieve BIST results, making it faster by a few orders of magnitude. This gap is greatly reduced by using partial configuration memory readback. The ORAs are designed such that the BIST results are stored in a single flip-flop of a PLB. This allows the BIST results to be captured in only one frame per ORA column. So, a total of $(M/2)-1$ frames are read back to retrieve BIST results, where M is the total number of PLB columns of the FPGA.

The configuration bit generation tool provided by Xilinx is used to obtain a logic allocation file. This file provides the information regarding the location of the configuration memory bits that contain the data captured from the PLB flip-flops. The location of each ORA flip-flop is defined in terms of the frame address and an offset within the frame.

3.6 Logic BIST Configurations for Virtex/Spartan-II

The following subsections present the details regarding the implementation of Logic BIST for Virtex/Spartan-II FPGAs.

3.6.1 Fault Model and Fault Coverage

The PLB of a Virtex/Spartan-II FPGA consists of 2 identical slices, so a single slice was modeled instead of modeling the entire PLB for fault simulations. The gate level stuck-at fault model was considered for fault coverage. The logic in the slice related to the RAM mode of operation of the LUTs was not considered, as faults in that logic would get detected by a LUT RAM test presented in [37]. The storage elements of the PLB were not tested in the asynchronous mode of operation, the reason for which is discussed in the Section 3.6.2. A total of seven configurations are required to completely test the PLB, not including the LUT RAMs and related logic. Cumulative fault coverage was evaluated by simulating the complete fault list for the first test configuration and then the list of undetected faults is used as the fault list for simulation of successive test configurations. Individual fault coverage of each test configuration was evaluated by using the complete fault list for simulation of all the test configurations. Both cumulative and individual fault coverage are shown in Figure 3.3.

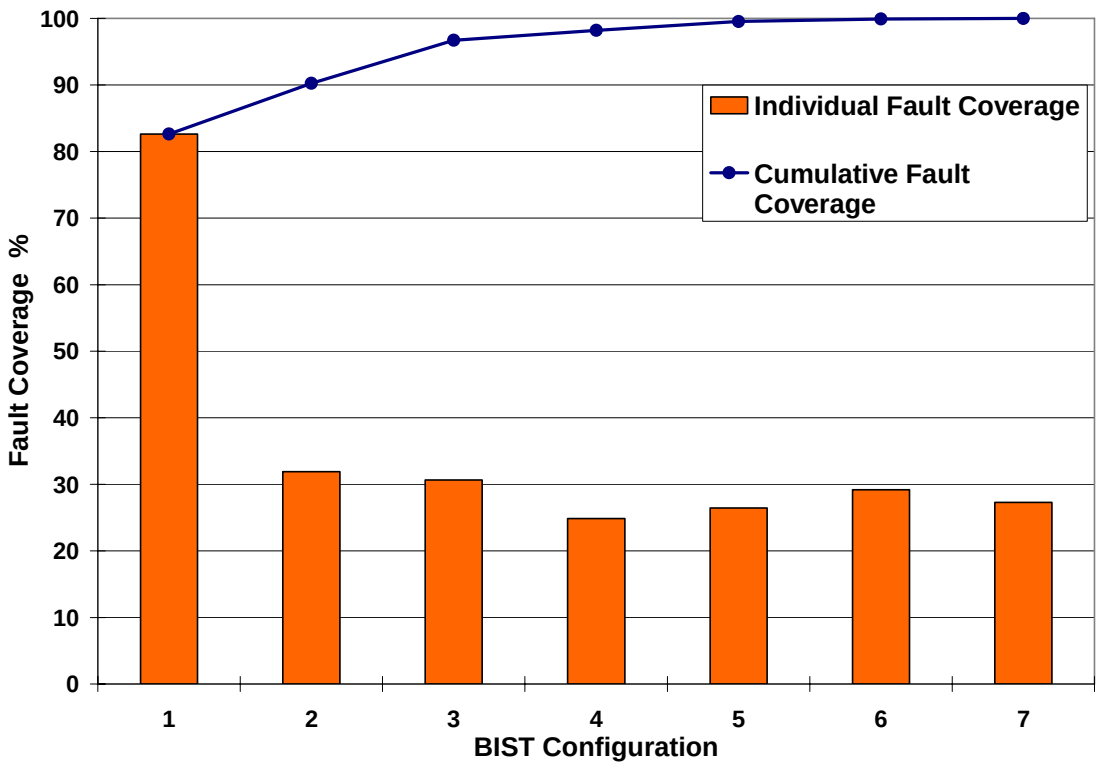

Figure 3.3 Fault Coverage of a Virtex FPGA PLB slice

3.6.2 Configuration Details

The details of the seven configurations of a Virtex Slice are summarized in Table 3.1. Some BUTs were diagnosed as faulty during Logic BIST when asynchronous reset was used; the cause for this was attributed to timing skew in the TPG output signals controlling the reset signal to the flip-flops, which introduced an uncertainty regarding the value stored in the storage elements of the BUTs. This issue remained unresolved during development for Virtex/Spartan-II FPGAs, but it was later resolved for Virtex-4, which is discussed in Section 4.4.2.

Table 3.1 Configuration Details

Slice Component		Configuration						
		1	**2**	**3**	**4**	**5**	**6**	**7**
LUT F/G	**LUT**	0000	xnor /xor	xor /xnor	xnor /xor	xor /xnor	xnor /xor	xor /xnor
	MODE	shift register	lut	lut	lut	lut	lut	lut
FF X/Y	**MODE**	ff	ff	latch	latch	latch	ff	ff
	X INIT	0	1	0	1	1	0	1
	Y INIT	0	1	0	1	1	0	1
	RESET	sync	sync	async	async	async	sync	async
CY0G / CY0F		prod	g1/f1	prod	0	1	1	1
CYSELG / CYSELF		g/f	g/f	1	g/f	g/f	1	1
GYMUX / FXMUX		g/f	f6/f5	gxor /fxor	f6/f5	f6/f5	gxor /fxor	gxor /fxor
DYMUX / DXMUX		I1	I0	I1	I1	I1	I0/I1	I0
YBMUX		I1	I1	I1	I0	I1	I0	I0
BY / BX		byinv /bxinv	by /bx	byinv /bxinv	0	1	by /bx	by /bx
SR		srinv	sr	sr	sr	1	0	1
CE		ceinv	1	ce	0	1	1	1
CLK		clk	clkinv	clk	clkinv	clk	clkinv	clk
CYINIT		bx	bx	cin	cin	cin	cin	cin
Breakpoints								
Cout		on	on	on	on	off	on	on
Y		on	on	on	on	off	on	on
XB		on	on	on	on	off	on	on
F5		on	on	on	on	off	on	on
X		on	on	on	on	off	on	on
REV USED		on	on	on	on	off	off	on
SR		on	on	on	on	off	on	on

3.7 Logic BIST Configuration Generation Process

Two programs were developed to generate all the test configurations, referred to as the *template generation program* and the *template modification program*. The design is described in Xilinx Design Language (XDL), a netlist format used by Xilinx. The template generation program generates a template file depending on the session and the slice test set, where the BUTs are configured with Configuration 1 as summarized in Table 3.1. The template file generated does not contain routing information as it simplifies configuration file generation process. The template is converted from XDL format to a Native Circuit Description (NCD) format that can be used by Xilinx CAD tools for routing the design. The template is routed using Xilinx routing tools and converted back to XDL format. The template modification program uses the routed template configuration file and modifies only the BUT configurations while keeping the routing fixed to generate all the other BIST configuration files. This approach results in generation of small partial reconfiguration files as the routing structure remains fixed for all test phases of a slice test set. The routed configuration files are used to generate the configuration bitstreams that are downloaded to the FPGA.

The Xilinx routing tools try to swap input pins of the LUTs and modify the LUT values to improve routability of the design. This is undesirable for the template modification program as it assumes the routing of the template file to be without modification of the LUT contents. Xilinx routing tools are prevented from swapping the LUT inputs by configuring the LUTs as shift registers in the first configuration. It can also be done by setting a ‘no pin swap’ option in FPGA Editor (a design editing tool by Xilinx) for routing a design.

3.8 Methods for Application of BIST

In this section, the methods used to speed up test time and reduce the configuration storage requirements for Logic BIST are described. The following are the various configuration download methods used:

FC - Full Configuration; partial reconfiguration is not used and all the test configurations downloaded to the FPGA are full configurations.

PR^2 - Partial Reconfiguration using Scenario 2 defined in Section 3.4; the first configuration in a test session is a full configuration, followed by six partial reconfigurations.

PR^3 - Partial Reconfiguration using Scenario 3 defined in Section 3.4.

OPR - Optimized Partial Reconfiguration using Scenario 3 defined in Section 3.4; the sequence in which the test configurations are applied was optimized to reduce the number of different configuration frames between two consecutive test configurations.

The following are the ORA results retrieval techniques used for Logic BIST:

FCRB - Full Configuration memory ReadBack after each test configuration

SR - Scan chain Readback after each test configuration

SRE - Scan chain Readback at the end of a test session

PCRB - Partial Configuration memory ReadBack after each test configuration

PCRE - Partial Configuration memory Readback at the End of a test session

Scan chain readback (SR, SRE) involves the use of ORAs connected as a scan chain, resulting in three slice test sets, whereas configuration memory readback (FCRB, PCRB, PCRE) requires only two slice test sets. Dynamic partial reconfiguration is used

for the methods SRE and PCRE; BIST results are retained until the end of a test session and retrieved only after all the test phases have been applied. Readback at the end reduces the diagnostic resolution of Logic BIST from a faulty PLB and its mode of operation down to a faulty PLB. Table 3.2 summarizes all the methods used for Logic BIST.

Table 3.2 Methods used for Logic BIST

Method	Configuration	BIST Results Retrieval	Total Slice test sets	Total number of configurations
1	FC	FCRB	2	28
2	FC	SR	3	42
3	PR^2	SR	3	42
4	PR^3	SR	3	42
5	OPR	SR	3	42
6	OPR	SRE	3	42
7	OPR	PCRB	2	28
8	OPR	PCRE	2	28

3.9 Results

Experimental results regarding the test time and memory storage requirements for implementation of Logic BIST are presented in this section. These results were obtained by applying the Methods 1 through 8 described in Table 3.2 on a Spartan-II XC2S200 FPGA which has a PLB array of size 28x42. Figure 3.4 shows the speed-up in test time and reduction in configuration memory storage requirements achieved.

It can be observed from the results that memory storage requirements are increased by using a scan chain for ORAs but a speed-up is achieved compared to full configuration memory readback. Partial configuration memory readback, although 40 times slower than scan chain for retrieval of BIST results, compensates for its lack of speed by eliminating a slice test set thereby reducing the total number of test

configurations from 42 to 28. The partial reconfiguration file sizes are also reduced by changing the configuration of only the slice under test (Scenario 3) and by ordering the test phases optimally. Retrieving the BIST results at the end of a test session rather than a test phase provides further speed-up at the cost of reduced diagnostic resolution. The actual test time using Boundary Scan (including all overhead related to the Boundary Scan operation) was reduced from 113 seconds (Method 1) to 22 seconds (Method 8), a speed-up of over 5 times. The configuration memory storage requirements were reduced by a factor of 3.25 for a Spartan-II XC2S200 FPGA.

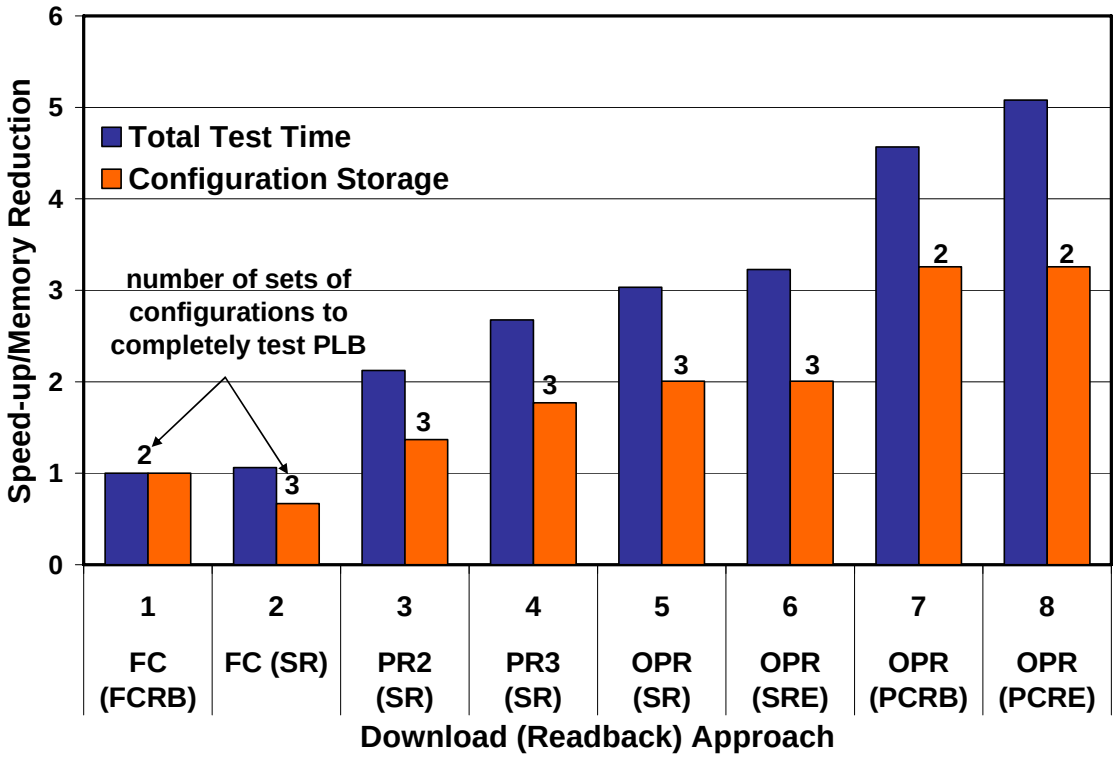

Figure 3.4 Test time speed-up and reduction in memory storage requirements

The effect of size of the device on speed-up is also evaluated using four different devices in the Virtex/Spartan-II FPGA family. Method 5 is chosen for comparison as the total test time in this case is a direct function of configuration file size since the time required to apply test vectors and retrieve BIST results is negligible. Table 3.3 illustrates the different speed-up values achieved, depending on the size of the device.

Table 3.3 Speed-up vs. Device Size

FPGA	Array size	Speed-up
XC2S15 (smallest)	8 × 12	3.61
XC2S50/XCV50	16 × 24	3.18
XC2S200/XVC200	28 × 42	3.02
XCV1000 (largest)	64 × 96	2.86

It is observed that the test time speed-up for Logic BIST drops by a small factor for larger devices. The ratio of PLB configuration data to the total configuration data increases as the size of the device increases. For Logic BIST, only the configuration of PLBs changes. Therefore, relatively larger partial reconfiguration files are generated for larger devices. This explains the reduction in test time speed-up for larger devices.

3.10 Summary

The architectural and operational features of Virtex/Spartan-II FPGAs were exploited to successfully achieve test time speed-up and reduction in memory storage requirements for Logic BIST configurations. Useful knowledge was gained from the implementation of Logic BIST, partial reconfiguration and partial configuration memory readback. Although the study was done using the Virtex/Spartan-II family of devices,

these approaches are also valid for other FPGA devices. The knowledge gained was applied to the Virtex-4 family of FPGAs discussed in the next chapter.

CHAPTER FOUR

Logic BIST for Virtex-4

4.1 Introduction

This chapter presents the implementation of Logic BIST on the Virtex-4 family of FPGAs. The architecture for Logic BIST is described along with the details of the test configurations and their timing analysis. Experimental results are presented for the methods used to achieve test time speed-up and reduction of configuration memory storage requirements, followed by analysis of the results and a summary. The work presented in this chapter is primarily the work presented in [34] with some additional details.

4.2 Virtex-4 Architecture

An overview of Virtex-4 architecture was presented in Chapter 2. In this chapter the details of FPGA resources relevant to Logic BIST are discussed. The PLB of a Virtex-4 FPGA consists of two SliceMs and two SliceLs. SliceL is illustrated in Figure 2.9. A SliceL has two LUTs, F and G, and storage elements, FFX and FFY, which can be configured as flip-flops or latches. Multiplexers CYINIT, CYMUXF and CYMUXG are used to implement the carry chain logic that spans the entire column of PLBs. DYMUX and DXMUX are used to select the input to the storage element. F5MUX and FSMUX

combine the LUTs of a PLB to implement combinational logic functions with greater than four inputs. A pair of AND and XOR gates are provided for arithmetic functions. CLK, CE and SR inputs provide common control inputs clock, clock enable and set/reset, respectively, for the storage elements FFX and FFY. The REV control places a logic value opposite to that determined by set/reset control signal in the storage element [10]. SliceMs feature extra circuitry like the write signal generator (WSGEN) and multiplexers (DIGMUX and DIFMUX) for shift register and RAM modes of operation of the LUTs. SliceM is illustrated in Figure 2.10.

The DSP cores in Virtex-4 FPGAs are arranged in columns as shown in Figure 2.7. There are two DSP cores for every four rows of PLBs in a DSP column. A DSP core consists of an 18x18-bit multiplier and a 48-bit adder/subtractor/accumulator, which can be configured to operate in different modes of operation as described in [35].

4.3 BIST Architecture

The BIST architecture is similar to the one used for Virtex/Spartan-II FPGAs. It is modified to exploit the architectural features of Virtex-4 to achieve higher diagnostic resolution. A column-based architecture is used for Virtex-4 for reasons similar to Logic BIST for Virtex/Spartan-II. Figure 4.1 illustrates the Logic BIST architecture for Virtex-4 FPGAs.

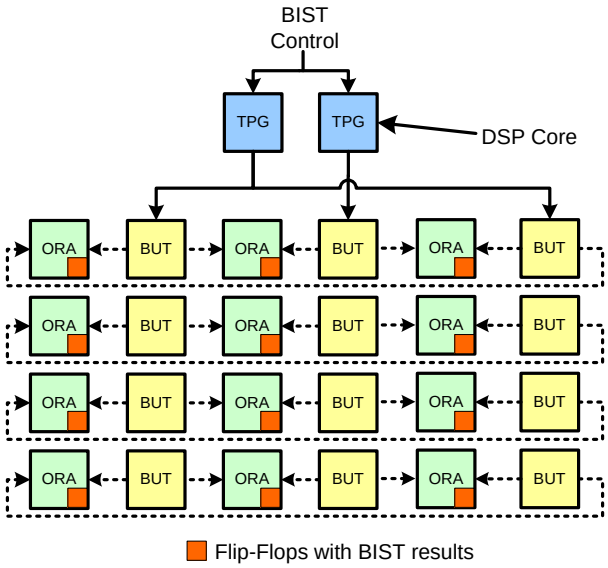

Figure 4.1 Logic BIST Architecture for Virtex-4 FPGAs

The PLBs in the FPGA are divided into alternate columns of BUTs and ORAs, where each BUT is compared by two ORAs. The outputs of the BUTs on the edge of the FPGA are compared by the ORAs on the other edge of the FPGA. This leads to a circular-comparison based BIST architecture as shown in Figure 4.3. This technique was originally developed for testing BlockRAMs of Virtex and Virtex-II FPGAs in [19]. It was possible to implement circular-comparison for Logic BIST because of the abundance of routing resources in Virtex-4 FPGAs.

All the primary outputs of a Virtex-4 PLB can be routed through the storage elements, this feature allows testing of all four slices of a PLB simultaneously by monitoring only eight outputs per PLB (one output per storage element). A PLB slice is divided into two halves, where each half can be used to implement an ORA that compares only one BUT output as shown in Figure 4.2. So, a total of eight independent ORAs are implemented in a PLB that compare the eight BUT outputs. This leads to better

diagnostic resolution, as each mismatch recorded in an ORA flip-flop now points to the exact half of a faulty PLB slice. This approach may increase the number of configurations required to test a PLB slice, but since few PLB outputs are observed, all slices are tested simultaneously and minimal routing changes are required to test the entire PLB, reducing partial reconfiguration file sizes. This may not be the case with other approaches that try to monitor all PLB outputs because they require multiple slice test sets to test all the slices and may require more configurations to test the entire PLB.

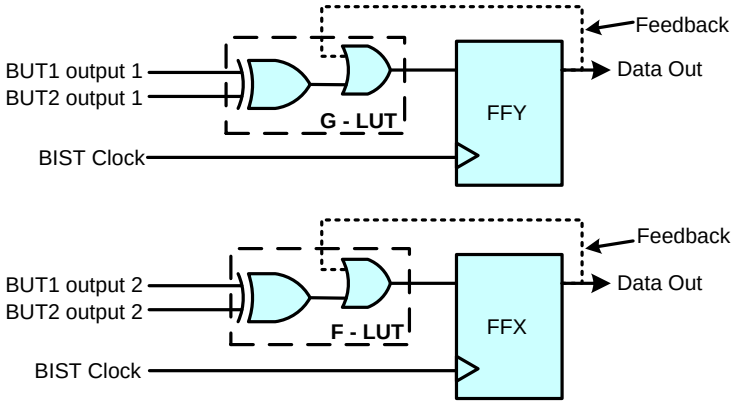

Figure 4.2 ORAs in a Single PLB slice

The configuration memory of Virtex-4 FPGAs is also organized in frames oriented vertically, but unlike Virtex/Spartan-II, the frame size is fixed. A single frame is associated with a fixed number of PLBs in a column instead of the entire column of PLBs. The data stored in the flip-flops of PLBs can be captured in the configuration memory by instantiating a CAPTURE_VIRTEX4 module in the design. The CAPTURE_VIRTEX4 module defines which clock edge is used to capture flip-flop data and whether it is captured once or multiple times. Configuration memory readback is used to retrieve the frames of configuration memory that contain the BIST results in the

ORA flip-flops. For speed-up in test time, partial configuration memory readback can be used instead of full configuration memory readback. A single frame of Virtex-4 captures the values contained in all the flip-flops of PLBs associated with that frame. This reduces the total number of frames to be read to retrieve BIST results, thereby improving the test time. The total number of frames *(F)* needed to be read is given by:

$$F = (R \div 16) \times (C \div 2) = R \times C \div 32$$

where, R is the number of rows and C is the number of columns of the PLB array under test. In the case of the XC4VLX25-10 FPGA which has 96 rows and 28 columns, only 84 frames need to be read back to obtain BIST results, as compared to 6022 frames for full configuration memory readback.

Traditionally, two TPGs are implemented using a column of PLBs [3]. The availability of DSP cores in newer FPGAs allow the use of DSPs to implement TPGs instead of the PLBs. This approach frees up a column of PLBs that are now used to implement an extra column of ORAs for circular comparison as shown in Figure 4.1. Since at least two DSPs are available for every four rows of PLBs, two TPGs are implemented (one TPG in each DSP) for every four rows of BUTs. This solves the issue of TPG loading and improves fault detection, as a faulty TPG only affects the testing of four rows of BUTs rather than the entire FPGA. An exhaustive set of test patterns is generated by initializing the accumulator of the DSP to zero and repeatedly adding a prime number ‘0x691’ to its contents [36]. One drawback of this approach is that the test patterns generated are not pseudo-random in nature, unlike the LFSR-based TPGs used for Virtex/Spartan-II. The 12 TPG outputs are connected to the 12 inputs of each of the four slices of a BUT, providing identical test vectors to all four slices of a PLB.

Logic BIST for Virtex-4 also uses Boundary Scan to access the configuration memory of the FPGA. The details of configuration download and readback procedures are described in [11]. Boundary Scan is also used to control the operation of BIST by means of the user access registers in the BSCAN modules. The Boundary Scan interface in Virtex-4 FPGAs features four BSCAN modules, two of which are used to implement Logic BIST. BSCAN module 1, when selected, enables BIST clock which is sourced from the TCK pin of the TAP. BSCAN module 2, when selected, disables the BIST clock and generates a reset signal that resets all the ORAs and TPGs.

Two test sessions (East and West) are required to test all the PLBs. In a given test session, only the BUTs are reconfigured multiple times to be tested completely. After the first configuration of a test session is downloaded, partial reconfiguration can be used to download the rest of the test phases to reduce the configuration download time. To keep the partial reconfiguration files small, the routing changes are kept to a minimum for a given test session. The routing between TPGs and BUTs is kept fixed and the routing between BUTs to ORAs is changed only once in a given test session. Virtex-4 allows multiple frames with identical data to be written simultaneously, where the frame data is loaded only once and the address in the FAR is changed. This feature helps reduce the partial reconfiguration file size since the Logic BIST architecture is a regular structure and all BUTs in most configurations are configured identically. The test time can be further reduced at the price of reduced diagnostic resolution by using dynamic partial reconfiguration as explained in Chapter 3. Using this approach the contents of the ORA flip-flops are not cleared when the BUTs are reconfigured and the BIST results are retrieved only at the end of a test session.

4.4 Logic BIST Configurations for Virtex-4

A total of 12 BUT configurations are required to completely test the logic resources of a PLB, excluding the circuitry associated with the LUT RAM mode of operation of SliceMs. For the first ten configurations only outputs XQ and YQ, associated with the storage elements FFX and FFY, respectively, are observed by the ORAs, completely testing SliceLs. Two extra configurations are required to test the logic circuitry associated with the shift register mode of SliceMs. In this case outputs X and Y of all four slices are observed by the ORAs. These 12 configurations also test the carry chain logic and the routing associated with it, along with the dedicated inter-slice routing. Therefore the total number of configurations required to test all the PLBs in the FPGA = 2 (test sessions) x 12 (test phases) = **24**.

4.4.1 Fault Model and Fault Coverage

The gate-level stuck-at fault model is used for fault coverage analysis. Some of the slice inputs cannot be accessed by resources external to the PLB, as they are only connected to the outputs of other slices in the PLB. Therefore, the complete PLB was modeled with dedicated inter-slice routing instead of individual slice models, leading to a more accurate fault coverage analysis. The cumulative and individual fault coverage of the 12 Logic BIST configurations is shown in Figure 4.3.

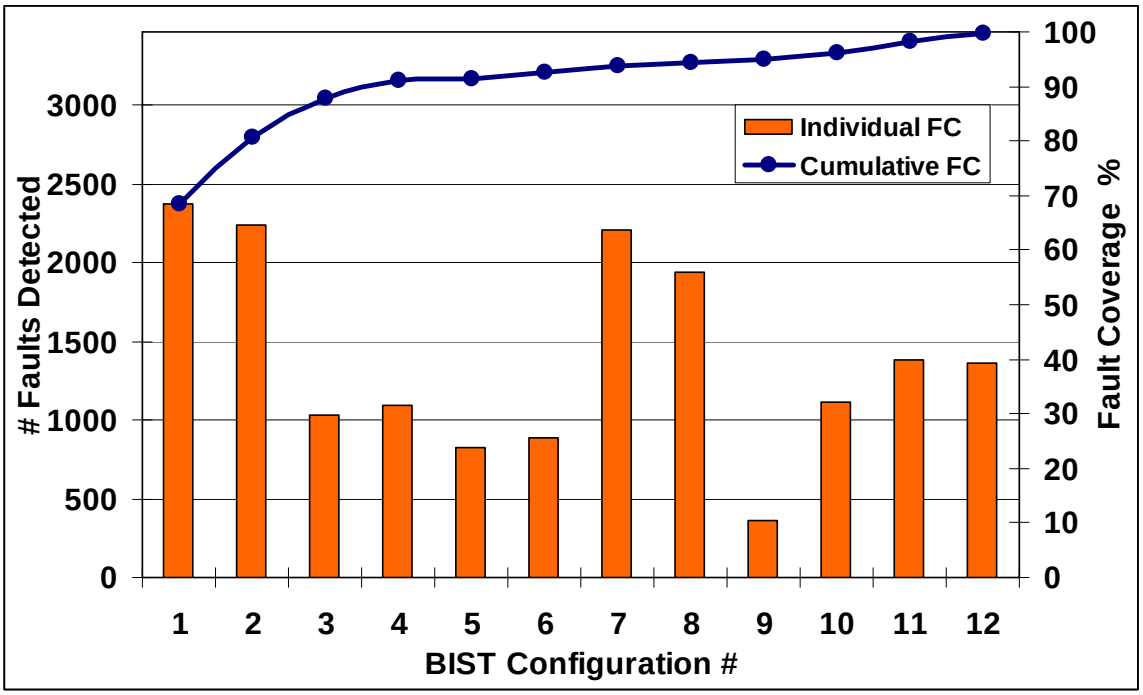

Figure 4.3 Fault Coverage of a Virtex-4 PLB

These 12 BIST configurations do not detect the logic resources of the PLB associated with the LUT RAM mode of operation like the WSGEN component, as they can be tested using the test for LUT RAMs [34]. The route-throughs in the PLB and breakpoints associated with PLB outputs not monitored by the ORAs are also not tested.

4.4.2 Configuration Details

The details of the 12 test configurations are summarized in Table 4.1.

Table 4.1 Configuration Details

Config	Slice Type	LUT F/G		FF X/Y				CY0G / CY0F	GYMUX / FXMUX	DYMUX / DXMUX	BY	BX /SR /CE	CLK	YBMUX / XBMUX	CYINT	REV USED
		LUT	MODE	MODE	INIT	SR	RESET									
1	SliceM	xor /xnor	shift register	ff	0	0	sync	prod /prod	fx/f5	yb/xb	non-inv	non-inv	clk	I1	bx	no
	SliceL	xor /xnor	lut	ff	0	0	sync	prod /prod	fx/f5	yb/xb	non-inv	non-inv	clk	I1	bx	no
2	SliceM	xnor /xor	shift register	ff	1	1	async	g2/f2	gxor /fxor	ymux /xmux	inv	inv	clkinv	I1	bx	no
	SliceL	xnor /xor	lut	ff	1	1	async	g2/f2	gxor /fxor	ymux /xmux	inv	inv	clkinv	I1	bx	no
3*	SliceM	xor /xnor	lut	latch	1	1	async	g3/f3	fx/f5	ymux /xmux	inv /non-inv	non-inv	clk	I1	bx	no
	SliceL	xor /xnor	lut	latch	1	1	async	g3/f3	fx/f5	ymux /xmux	inv /non-inv	non-inv	clk	I1	bx	no
4	SliceM	xnor /xor	lut	latch	0	0	async	0/0	fx/f5	yb/xb	non-inv	non-inv	clk	I1	bx	no
	SliceL	xnor /xor	lut	latch	0	0	async	0/0	fx/f5	yb/xb	non-inv	non-inv	clk	I1	bx	no
5	SliceM	aaaa /5555	lut	ff	0	1	sync	1/1	fx/f5	yb/xb	inv	inv	clkinv	I1	bx	yes
	SliceL	aaaa /5555	lut	ff	0	1	sync	1/1	fx/f5	yb/xb	inv	inv	clkinv	I1	bx	yes

Config	Slice Type	LUT F/G		FF X/Y				CY0G / CY0F	GYMUX / FXMUX	DYMUX / DXMUX	BY	BX /SR /CE	CLK	YBMUX / XBMUX	CYINT	REV USED
		LUT	MODE	MODE	INIT	SR	RESET									
6	SliceM	5555 /aaaa	lut	ff	1	1	async	g2/f2	gxor /fxor	yb/xb	non-inv	non-inv	clkinv	I1	bx	no
	SliceL	5555 /aaaa	lut	ff	1	1	async	g2/f2	gxor /fxor	yb/xb	non-inv	non-inv	clkinv	I1	bx	no
7*	SliceM	xor /xnor	shift register	latch	0	0	async	g3/f3	gxor /fxor	yb/xb	inv	inv	clk	I1	bx/cin	no
	SliceL	xor /xnor	shift register	latch	0	0	async	g3/f3	gxor /fxor	yb/xb	inv	inv	clk	I1	bx/cin	no
8	SliceM	xnor /xor	shift register	ff	1	0	sync	by/bx	fx/f5	y/x	non-inv	non-inv	clk	I1	bx	yes
	SliceL	xnor /xor	shift register	ff	1	0	sync	by/bx	fx/f5	y/x	non-inv	non-inv	clk	I1	bx	yes
9	SliceM	xor /xnor	lut	ff	1	1	sync	prod /prod	fx/f5	by/bx	inv	inv	clk	I1	cin	no
	SliceL	xor /xnor	lut	ff	1	1	sync	prod /prod	fx/f5	by/bx	inv	inv	clk	I1	cin	no
10*	SliceM	xnor /xor	lut	ff	0	0	sync	by/bx	gxor /fxor	yb/xb	non-inv	non-inv	clk	I1	cin/bx	no
	SliceL	xnor /xor	lut	ff	0	0	sync	by/bx	gxor /fxor	yb/xb	non-inv	non-inv	clk	I1	cin/bx	no
11	SliceM	ffff	shift register	ff	0	0	sync	by/bx	fx/f5	yb/xb	non-inv	non-inv	clk	I0	bx	no
	SliceL	ffff	shift register	ff	0	0	sync	by/bx	fx/f5	yb/xb	non-inv	non-inv	clk	I0	bx	no

Config	Slice Type	LUT F/G		FF X/Y				CY0G / CY0F	GYMUX / FXMUX	DYMUX / DXMUX	BY	BX /SR /CE	CLK	YBMUX / XBMUX	CYINT	REV USED
		LUT	MODE	MODE	INIT	SR	RESET									
12	SliceM	0000	shift register	ff	0	0	sync	by/bx	fx/f5	yb/xb	non-inv	non-inv	clk	I0	bx	no
	SliceL	0000	shift register	ff	0	0	sync	by/bx	fx/f5	yb/xb	non-inv	non-inv	clk	I0	bx	no

* These configurations were modified for fault detection. In these configurations all BUTs in the PLB array are not configured identically but all BUTs in a row of PLBs are configured identically.

Faults related to the FSMUX of Slice 3 of a Virtex-4 PLB are not detected if all the BUTs are configured identically. In order to detect those faults, Configuration 3 was modified such that the Slice 2 of any two adjacent BUTs in a column are configured with opposite value of BYINV multiplexer. Configurations 7 and 10 were also modified similarly for a timing issue related to the carry chain, explained in the next section. In this case the input of CYINIT multiplexer of Slices 0 and 1 is BX in alternate rows of BUTs and CIN for the remaining BUTs. This is reversed in Configuration 10 to completely testing the carry chain logic.

Due to the timing skew of TPG signals, some BUTs were diagnosed as faulty during BIST when asynchronous reset was used, similar to Logic BIST for Virtex/Spartan-II FPGAs described in Section 3.6.2. To resolve this issue for Virtex-4, two corrective measures were taken. Firstly, appropriate clock edge or active level was chosen for the storage element, of the BUTs. The clock edge or active level was chosen such that the storage element depending on its mode of operation (latch or flip-flop), was immune to timing skew. The storage element during BIST assumed a value defined by either the REV or *set/reset* input of the storage element, whichever changed last due to timing skew. So, the second corrective measure was to turn off the *revused* breakpoint internal to the PLB slice to disconnect the REV input from the storage element. This removed the contention between REV and *set/reset* inputs. The same techniques can be applied to Logic BIST for Virtex/Spartan-II FPGAs.

Storage elements of the Virtex-4 PLB are not cleared if dynamic partial reconfiguration is used or if CAPTURE_VIRTEX4 module is instantiated. This prohibits the initialization of the storage elements of BUTs after a test configuration download. In

order to test the PLB storage elements for initialization, two full configuration downloads are required to initialize the PLB storage elements to both high and low states. This can be achieved by using full configuration downloads for both Configuration 1 and Configuration 2, as they initialize the storage elements to high and low states, respectively.

4.5 Timing Analysis

Timing analysis was performed for all the test configurations implemented on the XC4VLX25-10 FPGA. Depending on the data collected, two configurations with the slowest and the fastest clock frequencies were chosen. These two configurations were analyzed for timing on Virtex-4 FPGAs of different sizes with a speed grade of 10 (10 being the slowest and 12 being the fastest speed grade). Figure 4.4 shows the fastest and the slowest clock frequencies at which the BIST configurations can operate for Virtex-4 FPGAs with different sizes.

It is noticed that the maximum clock frequency is a function of the number columns of PLBs, instead of the product of the number of rows and columns of PLBs, as was in the case of previous Logic BIST implementations [3]. This was achieved because the timing issues due to TPG loading were resolved for Virtex-4 by using a pair of TPGs for every four rows of BUTs. The maximum BIST clock frequency for the XC4VLX25-10 FPGA, for the first ten configurations, ranges from 70 to 150 MHz.

A major timing issue was discovered in Configuration 10 which tests the carry chain logic. The critical path for this configuration included the carry chain from the lowest PLB to the uppermost PLB in a column of BUTs. The excessive delay introduced

by the carry chain made the maximum clock frequency for Configuration 10 a function of the product of the number of rows and columns of PLBs. To avoid this situation, the carry chain was broken up as described in Section 4.4.2 which led to an increase in the maximum BIST clock frequency for the XC4VLX25-10 device from 40MHz to 140 MHz.

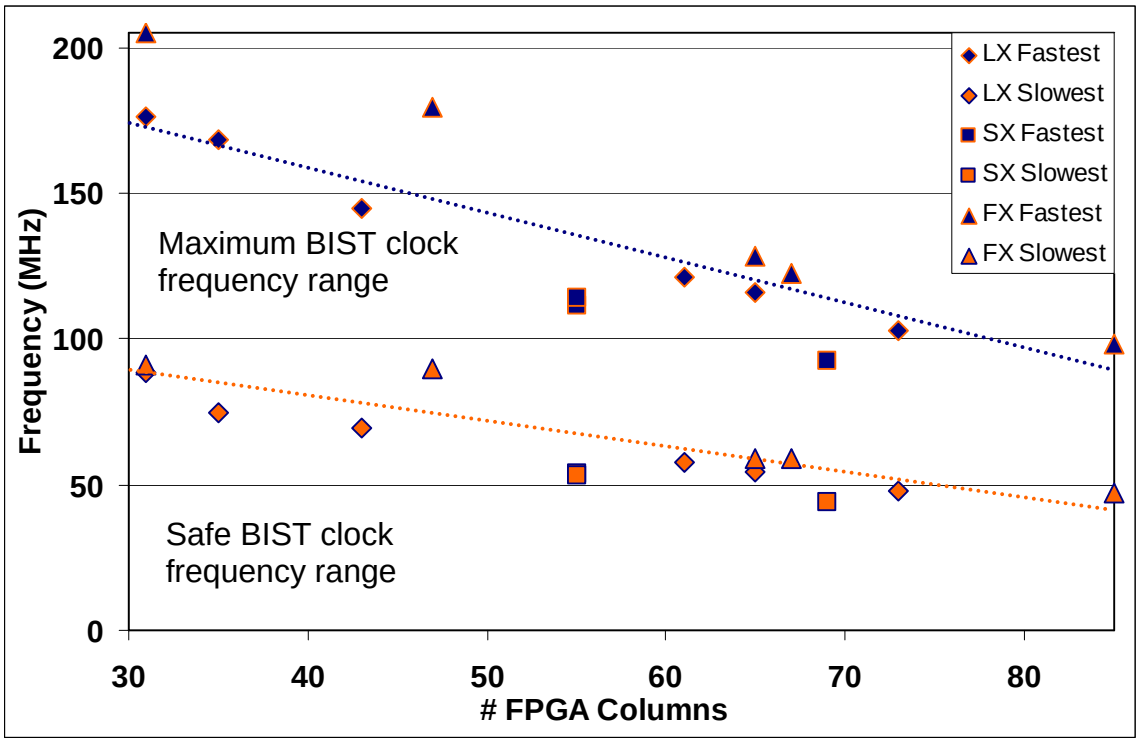

Figure 4.4 Maximum BIST clock frequency vs. Device size

4.6 Logic BIST Configuration Generation Process

Two parameterized C programs were developed for generation of the Logic BIST configurations of any PLB array size for all Virtex-4 FPGAs. The *template generation program* generates the template configuration in XDL format with a dummy BUT

configuration. The template configuration file is converted to an NCD format and routed using Xilinx tools. The routed template configuration file is then converted back to XDL format. The *template modification program* modifies the BUT configurations of the routed template configuration to generate all of the first ten test configurations. To generate the last two test configurations that test the SliceMs in shift register mode, both BUT configurations and BUT to ORA routing (X and Y outputs are monitored instead of XQ and YQ) of the routed template configuration are changed. Using this approach the routing of BIST architecture remains fixed for all configurations except for Configuration 11, resulting in generation of smaller partial reconfiguration files.

The Logic BIST structure for Virtex-4 can be defined for the entire FPGA or a portion of the PLB array. The number of columns in the PLB array to be tested has to be an even number greater than or equal to four, in order to implement circular comparison as shown in Figure 4.1. The presence of PowerPC core in Virtex-4 FX FPGAs complicates the implementation of Logic BIST, as shown in Figure 4.5. In the case of the West session the BUTs on the edge of the PowerPC core are compared by only one ORA instead of two, thereby losing some diagnostic resolution similar to the BUTs on the edge of Virtex/Spartan-II FPGA. But, for the East session the BUTs near the edges of the PowerPC core are compared by three ORAs instead of two without losing any diagnostic resolution.

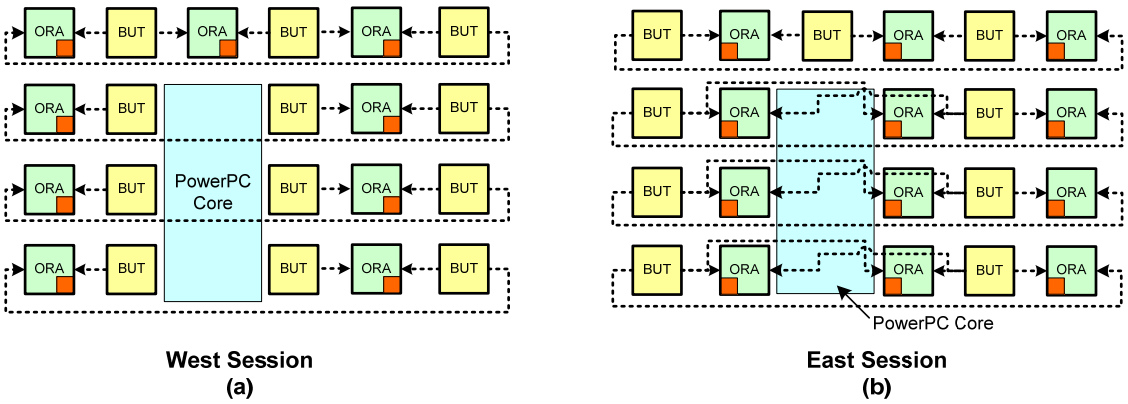

Figure 4.5 BIST Architecture in Virtex-4 FX FPGAs

4.7 Methods for Application of BIST

Various methods were used to evaluate the speed-up in test time and reduction in the memory required to store the BIST configurations. The following methods were used for configuration:

FC - Full Configuration

PR - Partial Reconfiguration using Scenario 3 defined in Section 3.4

Full configuration of all the BIST configurations was used in the first method; partial reconfiguration was used for all other methods. For the first ten test configurations ORAs monitor the XQ and YQ outputs of all four slices of a BUT, so the first configuration is a full configuration download followed by nine partial reconfiguration downloads, since there are no routing changes. For Configurations 11 and 12, ORAs monitor X and Y outputs. This leads to a change in the BUT to ORA routing, so Configuration 11 is again a full configuration download followed by a partial reconfiguration download for Configuration 12. Optimal ordering of the test configurations was investigated to minimize the difference between successive test

configurations for Virtex-4. But the reduction in partial reconfiguration files generated was negligible due to the organization of the configuration memory and multiple frame write feature of Virtex-4 FPGAs.

The following readback techniques were used for retrieval of BIST results:

FCRB - Full Configuration memory ReadBack after each test configuration

PCRB - Partial Configuration memory ReadBack after each test configuration

PCRE - Partial Configuration memory Readback at the End of a test session

FCRB was used for the first method, PCRB and PCRE were used to achieve speed-up in BIST results retrieval. It can be noticed that scan chain of ORAs was not implemented to retrieve the BIST results. From Logic BIST for Virtex/Spartan-II, it was observed that even though the scan chain is much faster for BIST results retrieval, it increases the total number of test configurations, thereby increasing the total test time. Table 4.2 summarizes all the methods used for Logic BIST in Virtex-4.

Table 4.2 Methods used for Logic BIST

Method	Configuration	BIST Results Retrieval	Total Slice test sets	Total number of configurations
1	FC	FCRB	2	24
2	PR	FCRB	2	24
3	PR	PCRB	2	24
4	PR	PCRE	2	24

4.8 Results

Logic BIST was implemented on a Virtex-4 XC4VLX25-10 device and the results of application of the four methods described earlier for test time speed-up and reduction in configuration memory storage requirements are shown in Figure 4.6. Method

1 was used primarily as a benchmark of the test time and memory storage requirements for comparison with speed-up techniques like partial reconfiguration and partial configuration memory readback applied in the other methods. Method 2 shows the improvements by using partial reconfiguration over full configuration. Method 3 shows the improvement after using partial configuration memory readback over full configuration memory readback to retrieve BIST results. The improvement due to dynamic partial reconfiguration is shown in Method 4.

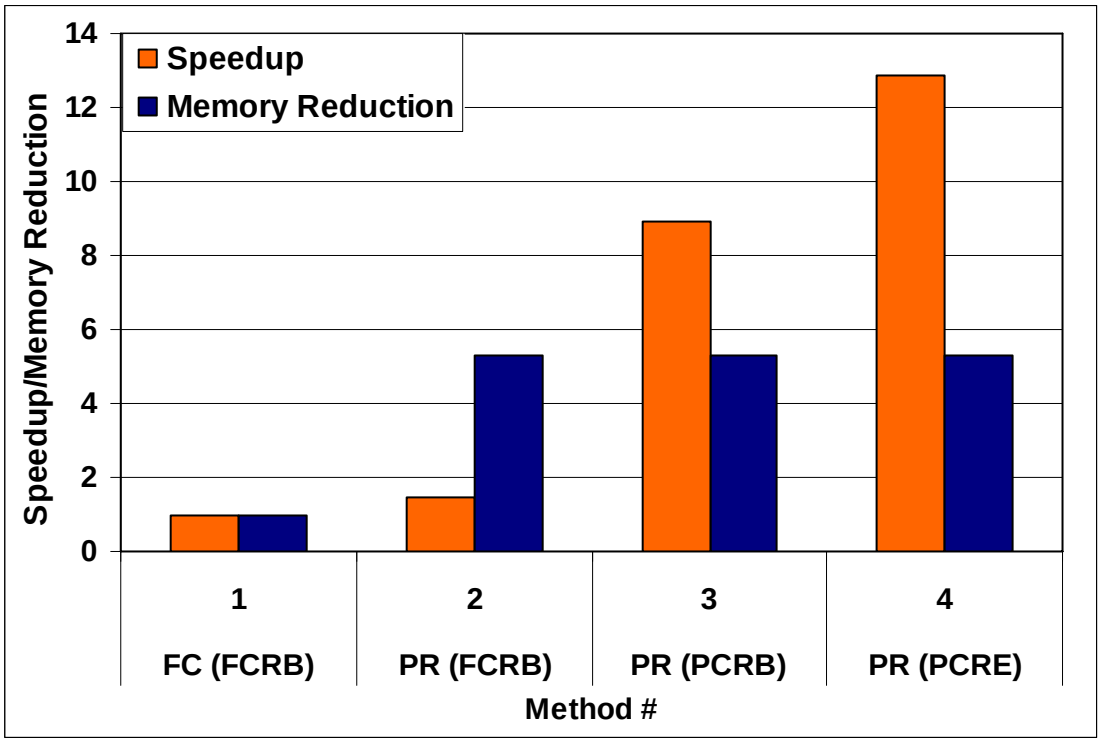

Figure 4.6 Test time speed-up and reduction in memory storage requirements

An overall speed-up of 12.9 is observed using PCRE and a net reduction of memory storage requirements by a factor of 5.3 is achieved using partial reconfiguration.

Methods 1, 3 and 4 for Virtex-4 can be compared to Methods 1, 7 and 8 used for Virtex/Spartan-II. A comparison of the results is given in Table 4.3

Table 4.3 Comparison of test time speed-up and reduction in memory storage requirements of Virtex/Spartan-II and Virtex-4 FPGAs

Method	**Test time speed-up**		**Memory reduction**	
	Virtex/Spartan-II	Virtex-4	Virtex/Spartan-II	Virtex-4
1	1	1	1	1
3	4.6	8.9	3.2	5.3
4	5.1	12.9	3.2	5.3

It is clear from Table 4.3 that the test time speed-up and reduction in memory storage requirements are better in Virtex-4 than in Virtex/Spartan-II FPGAs. The ability to write multiple frames with identical configuration data greatly reduces the partial reconfiguration time. The configuration memory is better organized, as configuration data for similar components is grouped together; the result is that fewer frames need to be written when BUTs are reconfigured. Test time speed-up using partial configuration memory readback is also enhanced, as fewer frames are read to retrieve the BIST results captured in the ORA flip-flops. Fewer frames are read because the contents of all the flip-flops in a column of a fixed number of PLBs are captured in a single frame instead of being spread across multiple frames.

4.9 Summary

Logic BIST for Virtex-4 showed considerable improvement in test time speed-up and reduction in memory storage requirements over Virtex/Spartan-II. Suitable

modifications were made to the BIST architecture to exploit architectural and operational features of the Virtex-4 FPGA in order to achieve better diagnostic resolution, faster test times and reduced memory storage requirements. Although the Logic BIST approach discussed pertains to Xilinx FPGAs, this approach can be applied to other FPGAs that support similar features, like partial reconfiguration and partial configuration memory readback.

CHAPTER FIVE

Summary and Conclusions

This thesis presented the testing of programmable logic resources in Xilinx FPGAs using BIST. Logic BIST configurations were developed for Virtex/Spartan-II and Virtex-4 FPGAs. Emphasis was put on techniques to improve the use of BIST for FPGAs. These include speed-up in test times due to the slow FPGA configuration process and reduction in memory storage requirements because of the large number of test configurations. Different techniques depending on the architectural and operational features of the FPGA were applied and their effects were studied for the speed-up in test time and reduction in memory storage requirements. The following sections in this chapter present a brief summary of the main contributions of the work presented in the thesis. Areas for future work are also proposed, along with a short discussion regarding the implementation of Logic BIST using embedded processors.

5.1 Thesis Summary and Main Contributions

The Logic BIST architecture was essentially derived from previous work done for Lucent ORCA and Xilinx 4000 series FPGAs. The BIST architecture was modified for Virtex/Spartan-II and Virtex-4 FPGAs to test their logic resources. The main contribution

of the work presented in this thesis was investigation of different techniques to reduce the test time and memory storage requirements for implementing Logic BIST.

A PLB slice of Virtex/Spartan-II FPGA was modeled for fault simulations, based on which the test configurations were developed. Most of the test time is devoted to configuration download rather than actual testing of the FPGA, so a reduction in the size of configuration files is a logical way to reduce the test time. This was achieved by using partial reconfiguration in a column-oriented Logic BIST architecture for Virtex/Spartan-II FPGAs. The test configurations were applied in a specific order such that the size of partial reconfiguration files was as small as possible. The scan chain method previously used for retrieval of BIST results, although faster, increased the total number of configurations required to test a PLB. Configuration memory readback was used instead of using a scan chain. This reduced the total number of configurations but continued to have a long test time as retrieval of BIST results was time consuming with full configuration memory readback. Partial configuration memory readback was used to overcome this issue and achieve the desired speed-up in BIST results retrieval time. Dynamic partial reconfiguration was used to further reduce the BIST results retrieval time by reading ORA contents at the end of a test session rather than a test phase. Two 'C' programs were developed to automate the generation of the Logic BIST configurations.

For Virtex-4 FPGAs, three major changes were made to the Logic BIST architecture. Firstly, the DSP cores were used instead of PLBs to implement TPGs. A pair of TPGs was used for every four rows of BUTs, which greatly reduced the TPG loading and hence improved the maximum BIST clock frequency. Secondly, circular

comparison was implemented to improve the diagnostic resolution of BUTs on the edge of the BIST architecture. Thirdly, the diagnostic resolution was further improved using eight ORAs per PLB rather than one. The techniques used for test time speed-up and reduction of memory storage requirements for Virtex/Spartan-II FPGAs were also applied to the Virtex-4 FPGA. The results obtained were better for Virtex-4 due to the enhanced architectural and operational features. The entire Virtex-4 PLB was modeled for more accurate fault coverage analysis, based on which the test configurations were developed. Two parameterized 'C' programs were developed to generate BIST configurations for any size PLB array in any of the Virtex-4 FPGAs.

5.2 Application to Embedded Processors

The Logic BIST approach was also applied to Virtex-II Pro FPGAs using embedded processors. Traditionally, an external source like a personal computer is used to download BIST configurations and run BIST, which is slow. The embedded processor can be used instead to internally run BIST and reconfigure BUTs [38]. This significantly speeds up the test time, as the number of external downloads are reduced and BIST runs at a much higher clock frequency [39]. For Xilinx FPGAs, the basic approach is to divide the FPGA into two halves; one consists of the Logic BIST structure and the other consists of the embedded processor. The embedded processor and the Logic BIST structure are swapped, after one half of the FPGA has been completely tested, to test the entire FPGA. The goal is to automate Logic BIST as much as possible using the embedded processor and achieve test time speed-up. The work done on Virtex-II Pro indicated that the

embedded processor and Logic BIST structure can be successfully integrated and the processor can read and write to the configuration memory.

5.3 Areas for Future Research and Development

The next step to achieve speed-up is to use an embedded processor to assist Logic BIST, as described in the previous section. Once a configuration is downloaded with a BIST structure in one half and an embedded processor in the other, the processor should be able to perform all the other functions for a test session, which include running BIST, retrieval and analysis of BIST results, and reconfiguration of BUTs in different modes of operation.

The current procedure for developing Logic BIST configurations is slow and tedious. Work can be done to automate the process of fault simulation and generation of BIST configurations by possibly using a generic BIST architecture described in a Hardware Description Language that can be synthesized for a new FPGA architecture with minor modifications. The work presented in this thesis was developed for Xilinx FPGAs only. Similar work can be done for testing the logic resources of different FPGA architectures from different FPGA manufacturers. The effects of the techniques used for test time speed-up and better diagnostic resolution can also be explored for other FPGAs and other programmable resources in FPGAs.

REFERENCES

[1] B. Arnaldo, "Systems on Chip: Evolutionary and Revolutionary Trends", Proc. of Intn'l Conf. on Computer Architecture, pp. 121-128, 2002.

[2] M.J.S. Smith, *Application Specific Integrated Circuits,* Addison-Wesley, 1997.

[3] M. Abramovici, C. Stroud, "BIST-Based Test and Diagnosis of FPGA Logic Blocks", IEEE Trans. on VLSI Systems, Vol. 9, No. 1, pp. 159-172, 2001.

[4] K. Mori, H. Yamada, S. Takizawa, "System on Chip Age", Proc. of Intn'l Symp. on VLSI Technology, Systems, and Applications, pp. K15-K20, 1993.

[5] M.L. Bushnell, V.D. Agrawal, *Essentials of Electronics Testing for Digital, Memory & Mixed Signal VLSI Circuits*, Kluwer Academic Publishers, Boston, MA, 2000.

[6] C. Stroud, *A Designer's Guide to Built-In Self-Test*, Kluwer Academic Publishers, Boston MA, 2002.

[7] C. Stroud, K. Leach, T. Slaughter, "BIST for Xilinx 4000 and Spartan Series FPGAs: A Case Study", Proc. of Intn'l Test Conf., pp. 1258-1267, 2003.

[8] __, "Virtex Field Programmable Gate Arrays", Product Specification DS003-1, Xilinx Inc., 2001.

[9] __, "Virtex-4 Family Overview", Product Specification DS-112, Xilinx Inc., 2005.

[10] __, "Virtex-4 User Guide", UG070, Xilinx Inc., 2005.

[11] __, “Virtex-4 Configuration Guide”, UG071, Xilinx Inc., 2005.

[12] __, “Spartan-II 2.5V FPGA Family: Introduction and Ordering Information”, DS-001, Xilinx Inc., 2004.

[13] J.M. Rabaey, A. Chandrakasan, B. Nikolić, *Digital Integrated Circuits: A Design Perspective, 2nd Edition*, Pearson Education, 2003.

[14] __, “Two Flows for Partial Reconfiguration: Module Based or Difference Based”, Application Note XAPP290, Xilinx Inc., 2003.

[15] J. Rose, A.E. Gamal, A. Sangivanni-Vincentelli, “Architecture of Field-Programmable Gate Arrays”, Proc. of IEEE, Invited Paper, pp. 1013-1029, 1993.

[16] C. Stroud, J. Sunwoo, S. Garimella, J. Harris, “Built-In Self-Test for System-on-Chip: A Case Study”, Proc. of Intn’l Test Conf., pp.837-846, 2004.

[17] M. Abramovici, C. Stroud, S. Wijesuriya, C. Hamilton, V. Verma, “Using Roving STARs for On-Line Testing and Diagnosis of FPGAs in Fault-Tolerant Applications”, Proc. of Intn’l. Test Conf., pp. 973-982, 1999.

[18] M. Abramovici, C. Stroud, B. Skaggs, J. Emmert, “Improving On-Line BIST-Based Diagnosis for Roving STARs”, Proc. of Intn’l On-Line Test Workshop, pp. 1-39, 2000.

[19] C. Stroud, S. Garimella, “Built-in Self-test and Diagnosis of Multiple Embedded Cores in SoCs”, Proc. of Intn'l Conf. on Embedded Systems and Applications, pp. 130-136, 2005.

[20] C. Stroud, S. Wijesuriya, C. Hamilton, M. Abramovici, “Built-In Self-Test of FPGA Interconnect”, Proc. of Intn’l Test Conf., pp. 404-411, 1998.

[21] C. Stroud, P. Chen, S. Konala, M. Abramovici, "Evaluation of FPGA Resources for Built-In Self-Test of Programmable Logic Blocks", Proc. of ACM Intn'l. Symp. on FPGAs, pp. 107-113, 1996.

[22] W. K. Huang, F. Lombardi, "An Approach to Testing Programmable/Configurable Field Programmable Gate Arrays", Proc. of VLSI Test Symp., pp. 450-455, 1996.

[23] W. K. Huang, F. J. Meyer, X. Chen, F. Lombardi, "Testing Configurable LUT-based FPGAs", IEEE Trans. on VLSI Systems, pp. 276-283, 1998.

[24] M. Renovell, Y. Zorian, "Different Experiments in Test Generation for Xilinx FPGAs", Proc. of Intn'l Test Conf., pp. 854-862, 2000.

[25] M. Renovell, J. Portal, J. Figueras, Y. Zorian, "Testing the Interconnects of RAM-based FPGAs", IEEE Design and Test of Computers, vol. 15, pp. 45-50, 1998.

[26] I. G. Harris, R. Tessier, "Interconnect Testing in Cluster-based FPGA Architectures", Proc. of Design Automation Conf., pp. 49-54, 2000.

[27] S. Wang, C. Huang, "Testing and Diagnosis of Interconnect Structures in FPGAs", Proc. of Asian Test Symp. , pp. 283-287, 1998.

[28] S. Wang, T. Tsai, "Test and Diagnosis of Faulty Logic Blocks in FPGAs", Proc. of Intn'l Conf. on Computer-Aided Design, pp. 722-727, 1997.

[29] A. Newalkar, "Alternative Techniques for Built-In Self-Test of Field Programmable Gate Arrays", Master's Thesis, Auburn University, 2005.

[30] IEEE Standards Board, 345 E. 47th St. New York 10017, IEEE Standard Test Access Port and Boundary-Scan Architecture, 1994. IEEE/ANSI Standard 1149.1-1994.

[31] S. Dhingra, S. Garimella, A. Newalkar, C. Stroud, “Built-In Self-Test of Virtex and Spartan II using Partial Reconfiguration”, Proc. of North Atlantic Test Workshop, pp. 7-14, 2005.

[32] __, “Virtex Series Configuration Architecture User Guide”, Application Note XAPP151, Xilinx Inc., 2003.

[33] __, “Virtex FPGA Series Configuration and Readback”, Application Note XAPP138, Xilinx Inc., 2003.

[34] S. Dhingra, D. Milton, C. Stroud, “BIST for Logic and Memory Resources in Virtex-4 FPGAs”, Proc. of North Atlantic Test Workshop, pp. 19-27, 2006.

[35] __, “XtremeDSP for Virtex-4 FPGAs User Guide”, UG073, Xilinx Inc., 2005

[36] S. Gupta, J. Rajski, J. Tyszer, “Test Pattern Generation Based on Arithmetic Operations”, Proc. of Intn’l Conf. on Computer-Aided Design, pp. 117-124, 1994

[37] S. Garimella, “Built-In Self-Test for Regular for Regular Structure Embedded Cores in System-on-Chip”, Master’s Thesis, Auburn University, 2005

[38] J. Sunwoo, C. Stroud, “BIST of Configurable Cores in SoCs Using Embedded Processor Dynamic Reconfiguration”, Proc. of Intn'l SoC Design Conf., pp. 174-177, 2005

[39] C. Stroud, S. Garimella, J. Sunwoo, “On-Chip BIST-Based Diagnosis of Embedded Programmable Logic Cores in SoCs”, Proc. of ISCA Intn'l Conf. on Computers and their Applications, pp. 308-313, 2005