Embedded Soft-Core Processor-Based Built-In Self-Test of Field Programmable Gate Arrays

by

Bradley Fletcher Dutton

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
May 14, 2010

Keywords: Built-In Self-Test, Field Programmable Gate Array,
Fault Tolerance, Single-Event Upset Detection and Correction

Copyright 2010 by Bradley Fletcher Dutton

Approved by

Charles E. Stroud, Chair, Professor of Electrical and Computer Engineering
Vishwani D. Agrawal, Professor of Electrical and Computer Engineering
Victor P. Nelson, Professor of Electrical and Computer Engineering

Abstract

The exponential growth in the number of transistors on very large scale integration (VLSI) integrated circuits (ICs), coupled with increasing device interface bandwidth and new surface mount and low profile packaging technologies, have made testing of ICs increasingly difficult and costly at all levels of the testing process. Field programmable gate arrays (FPGAs) pose a particularly difficult problem for test engineers due to their programmable nature, overall size and complexity, limited number of inputs/outputs (I/O), and large number and variety of embedded cores. In addition to manufacturing defects, "soft" errors due to single event upsets (SEUs) have become a serious problem because of the increasing size of the configuration memory in FPGAs and shrinking design rules, even in fault-tolerant systems operating at ground level. Building on previous work, this thesis uses built-in self-test (BIST) as a solution to the testing problem for Xilinx Virtex-5 FPGAs. BIST configurations are presented for the configurable logic blocks (CLBs), I/O Tiles, and SEU detection/correction cores in Xilinx Virtex-5 FPGAs. In addition, this thesis presents a novel approach to BIST that uses a soft-core processor configured in the fabric of the device under test to perform reconfiguration of the resources under test, control the BIST execution, and perform fault diagnosis. This approach is particularly useful for in-system testing of FPGAs in fault-tolerant or high-reliability systems because it greatly reduces the amount and complexity of external hardware required for test. To combat the problem of "soft" errors due to SEUs that can occur in the FPGA configuration memory during normal operation, an approach for on-line detection and correction of SEUs in

the configuration memory of Xilinx Virtex-4 and Virtex-5 FPGAs is also presented. While not entirely immune to SEU effects, this approach greatly reduces the probability of an SEU induced failure in the user logic, and no single error from an SEU can cause a complete system failure.

Acknowledgments

First, I would like to thank Dr. Stroud for three great years of guidance, encouragement, employment, and education. You have taught me most of what I know about being an engineer, and what I appreciate most in hindsight is that you've always challenged me to be the best. I might not have even gone to graduate school if not for you. I also would like to thank the many students that I've had a chance to work with and learn from while in the BIST lab. Lee, Daniel, and Bobby: I learned a lot from you guys and, honestly, the lab was never the same without you (Bobby, you especially: I can't help laughing even as I write this). To the students that came later – Jia, Mary, Brooks, and Joey – thanks for being good friends through thick and thin and for making time spent at work more fun. To Joseph and Jie: for being the best engineers my age that I've ever met, and, therefore, inspiring me to always work a little harder. I would also like especially to thank my mom and dad for always being supportive in everything that I've done. Robbie: for being my best and oldest friend and future business partner (or future landlord, if engineering doesn't work out). And Bo and Samantha, thanks for dragging me out of my room and keeping me up late, regardless of projects or exams, and for teaching me some things that cannot be learned in a classroom.

Table of Contents

Abstract ii

Acknowledgments iv

List of Tables ix

List of Figures xi

List of Abbreviations xv

Chapter One. Introduction 1

1.1 Overview of Built-In Self-Test 2

1.2 Introduction to Field Programmable Gate Arrays (FPGAs) 4

1.3 Overview of Virtex-5 FPGAs 7

1.4 BIST for FPGAs 10

1.5 Single Event Upsets in FPGAs 11

1.6 Verification by Fault Injection 13

1.7 Thesis Statement 14

1.8 Thesis Format 15

1.9 References 15

Chapter Two. Built-In Self-Test of Configurable Logic Blocks in Virtex-5 FPGAs 18

2.1 Introduction And Background 18

2.2 Overview of Virtex-5 CLBs 20

2.3 BIST Approach And Architecture 22

2.4 Experimental Results 26

2.5 Summary And Conclusions 32
2.6 Acknowledgements 33
2.7 References 34
Chapter Three. Built-In Self-Test of Programmable Input/Output Tiles in Virtex-5 FPGAs 35
3.1 Introduction 35
3.2 Prior Work 37
3.3 Overview of Virtex-5 I/O Tiles 38
3.4 Overview of BIST Architecture 39
3.5 Configurations for I/O Logic Modes 43
3.6 Configurations for I/O SerDes Modes 43
3.7 Experimental Results 45
3.8 BIST for Programmable I/O buffers 48
3.9 Conclusions 49
3.10 Acknowledgements 50
3.11 References 50
Chapter Four. Built-In Self-Test of SEU Detection Cores in Virtex-4 and Virtex-5 FPGAs 52
4.1 Introduction 52
4.2 Frame ECC and ICAP Logic 54
4.3 Test Algorithm 57
4.4 BIST Approach 59
4.4.1 Test Pattern Generator 60
4.4.2 Output Response Analyzer 62
4.4.3 Additional Logic 64
4.5 Implementation Results 64
4.6 Conclusions 70

4.7 Acknowledgements 70

4.8 References 71

Chapter Five. Embedded Processor Based Fault Injection and SEU Emulation for FPGAs 73

5.1 Introduction and Background 73

5.2 Hard Core Processor Case Study 75

5.3 Soft Core Processor Case Study 79

5.3.1 Overview of Approach 80

5.3.2 Architecture and Operation 85

5.3.3 Implementation Results 88

5.4 Summary and Conclusions 92

5.5 Acknowledgements 92

5.6 References 93

Chapter Six. Soft-Core Embedded Processor-Based Built-In Self-Test of FPGAs 95

6.1 Introduction 95

6.2 Background 96

6.3 Embedded BIST Architecture 100

6.4 Software Development 104

6.5 Design Flow and Implementation Results 109

6.6 Conclusions 111

6.7 Acknowledgements 111

6.8 References 112

Chapter Seven. Soft-Core Embedded Processor-Based Built-In Self-Test of FPGAs Case Study 113

7.1 Introduction 113

7.2 Background 114

7.3 Results of Implementation in Virtex-5 118

7.4 Future Improvements 123

7.5 Other Applications 124

7.6 Conclusions 125

7.7 Acknowledgements 127

7.8 References 127

Chapter Eight. On-line Single Event Upset Detection and Correction in Field Programmable Gate Array Configuration Memories 129

8.1 Introduction 129

8.2 Background 134

8.3 Operation of SEU Detect and Correct 137

8.4 SEU Detect and Correct Architecture 140

8.5 Implementation Results 145

8.6 Experimental Results 149

8.7 Conclusions 153

8.8 Acknowledgements 154

8.9 References 155

Chapter Nine. Summary and Conclusions 157

9.1 Summary of Work 157

9.2 Future Work 160

Bibliography 162

List of Tables

Table 2.1: List of acronyms ... 20

Table 2.2: SliceL logic BIST configurations ... 25

Table 2.3: SliceM BIST configurations ... 26

Table 2.4: CLB BIST totals (17 configurations) ... 32

Table 3.1: I/O tile BIST totals (15 configurations) ... 48

Table 4.1: Frame ECC codes ... 55

Table 4.2: Hamming parity matrix example ... 57

Table 4.3: ICAP and Frame ECC BIST summary ... 70

Table 5.1: Embedded fault injection run time analysis for AT94K40 ... 78

Table 5.2: Parity bit encoding, where X = don't care ... 87

Table 5.3: Embedded fault list format ... 87

Table 5.4: Embedded fault injection core resources ... 88

Table 5.5: Fault/SEU injection core I/O descriptions ... 91

Table 6.1: BIST control registers ... 103

Table 6.2: Compressed partial reconfiguration data size ... 107

Table 7.1: Test configurations developed for various FPGAs ... 115

Table 8.1: Memory resources in two Virtex-5 FPGAs ... 130

Table 8.2: Frame ECC error codes [25][26] ... 135

Table 8.3: Hamming bit error diagnosis [25][26] ... 143

Table 8.4: SEU controller resource utilization in Virtex-4 devices 148

Table 8.5: SEU controller resource utilization in Virtex-5 devices 148

Table 8.6: SEU emulation results 152

Table 8.7: Approximate number of configuration bits for common resources [5] 153

List of Figures

Figure 1.1: Basic BIST architecture [3] 3

Figure 1.2: Typical custom ASIC, standard cell ASIC, and FPGA cost vs. volume...................... 5

Figure 1.3: Typical FPGA architecture [12] 6

Figure 1.4: Simplified basic logic element 7

Figure 1.5: Virtex-5 configurable logic block [15]... 8

Figure 1.6: Virtex-5 6-Input LUT [16] 9

Figure 1.7: Illustration of a single-event effect in a CMOS inverter 12

Figure 2.1: Simplified basic logic element 21

Figure 2.2: Virtex-5 configurable logic block [11]... 21

Figure 2.3: Circular comparison architecture 23

Figure 2.4: Equivalent ORA architecture 24

Figure 2.5: SliceL fault coverage (simulation) 29

Figure 2.6: SliceL fault coverage (fault injection).. 29

Figure 2.7: SliceM fault coverage (simulation) 30

Figure 2.8: SliceM fault coverage (fault injection)... 30

Figure 2.9: Boundary Scan interface test time.. 31

Figure 2.10: 32-bit parallel interface test time.. 31

Figure 3.1: Simplified programmable I/O cell... 37

Figure 3.2: Virtex-5 programmable I/O tile... 38

Figure 3.3: Column oriented circular comparison 40

Figure 3.4: Virtex-5 equivalent ORA architecture 41

Figure 3.5: Bitslip synchronizer circuit 45

Figure 3.6: 50 MHz Boundary Scan configuration interface test time 47

Figure 3.7: 100 MHz 32-bit parallel configuration interface test time 47

Figure 4.1: Frame ECC and ICAP primitives 56

Figure 4.2: Sequential Hamming bit calculation 60

Figure 4.3: Test pattern write sequence via ICAP interface 61

Figure 4.4: Test pattern read sequence via ICAP interface 61

Figure 4.5: ICAP and Frame ECC BIST architecture.. 65

Figure 4.6: BIST VHDL component declaration.. 66

Figure 4.7: Virtex-4 FX12 with ICAP/Frame ECC BIST 68

Figure 4.8: Virtex-5 LX20T with ICAP/Frame ECC BIST... 69

Figure 5.1: AT94K series SoC architecture.. 76

Figure 5.2: AT94K routing architecture 77

Figure 5.3: SliceL simulation stuck-at fault coverage 83

Figure 5.4: SliceL fault injection stuck-at fault coverage.. 83

Figure 5.5: Total CLB test time via Boundary Scan... 84

Figure 5.6: Frame read-modify-write flowchart 85

Figure 5.7: Block diagram of fault injection core... 88

Figure 5.8: Routed embedded fault inject core (right) with half-array of routed CLB BIST (left) in Virtex-5 LX20T 90

Figure 5.9: Fault inject core component declaration 91

Figure 6.1: Configurable logic block (CLB) BIST architecture 97

Figure 6.2: Embedded soft core processor based BIST architecture 103

Figure 6.3: Embedded processor BIST algorithms 105

Figure 6.4: Compressed BIST partial reconfiguration structure in C 107

Figure 6.5: Original reconfiguration file sizes and compressed data structure sizes for one CRC BIST and a set of 5 I/O Logic BIST partial reconfigurations 108

Figure 6.6: Embedded processor BIST design implementation 110

Figure 7.1: Simplified soft-core processor-based BIST architecture.. 117

Figure 7.2: Unrouted embedded processor-based BIST configuration for top configurable logic blocks (CLB) in Virtex-5 LX30T viewed in FPGA Editor 119

Figure 7.3: CLB BIST test time for external configuration (full compressed and partial compressed bitstreams) and embedded processor test time.. 120

Figure 7.4: Contribution to embedded processor-based CLB BIST test time by initial external configuration and by five internal partial reconfigurations 122

Figure 7.5: Comparison of CLB BIST ORA read back times with embedded processor-based approach and external Boundary Scan interface... 122

Figure 7.6: 32-bit, 100 MHz interface test time for full chip CLB west or east with one full compressed configuration and five partial reconfigurations.. 124

Figure 8.1: FIT rate (corrected for sea-level New York, NY) versus Xilinx device family, initial release year, and minimum feature size [6] where the center line represents the nominal value and the span of the line represents the upper and lower 95% confidence levels 131

Figure 8.2: Frame ECC and ICAP primitives 137

Figure 8.3: SEU controller VHDL component declaration 137

Figure 8.4: SEU controller behavioral pseudocode 139

Figure 8.5: SEU controller block diagram 142

Figure 8.6: SEU controller LOG cycle time vs. Virtex-4 device.. 146

Figure 8.7: SEU controller cycle time vs. Virtex-5 device... 147

Figure 8.8: Routed SEU controller implemented in Virtex-5 LX20T device............................. 150

List of Abbreviations

ATE	Automatic Test Equipment
BIST	Built-In Self Test
BRAM	Block RAM
BSCAN	Boundary Scan
BUT	Block under Test
CAD	Computer-aided Design
CLB	Configurable Logic Block
CMOS	Complementary Metal-oxide-semiconductor
CUT	Circuit under Test
DFT	Design for Testability
DSP	Digital Signal Processor
DUT	Device Under Test
ECC	Error Correction Code
FF	Flip-flop
FIFO	First-in First-out
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GUI	Graphical User Interface
HDL	Hardware Description Language

I/O	Input / Output
IC	Integrated Circuit
ICAP	Internal Configuration Access Port
IP	Intellectual Property
LUT	Look-up Table
LSB	Least Significant Bit
MSB	Most Significant Bit
ORA	Output Response Analyzer
PIP	Programmable Interconnect Point
PLB	Programmable Logic Block
RAM	Random Access Memory
SERDES	Serializer / Deserializer
SEU	Single Event Upset
SoC	System-on-Chip
SRAM	Static Random Access Memory
TCK	Test Clock
TDI	Test Data In
TDO	Test Data Out
TMS	Test Mode Select
TPG	Test Pattern Generator
VLSI	Very Large Scale Integration

Chapter One. Introduction

Moore's law, which predicts a doubling of integrated circuit (IC) transistor density every 18 to 24 months, has been an accurate predictor of the exponential growth in the number of transistors in ICs since it was first observed by Gordon Moore in 1965 [1]. According to the most recent International Technology Roadmap for Semiconductors (ITRS) report, minimum feature size is expected to continue to decrease by a factor of two (*e.g.* transistor density will increase by a factor of two) every two years until 2022 [2]. With very large-scale integration (VLSI) circuits already surpassing the one billion transistor mark in 2008, this report, in accordance with Moore's law, predicts that the number of transistors on a single IC of comparable physical area will exceed 128 billion by 2022.

Increasing transistor count and density and increasing device interface bandwidth, coupled with new surface mount and low profile packaging technologies, have made testing of integrated circuits increasingly difficult and costly at all levels of the testing process [3] [4]. In addition, larger device sizes and smaller feature sizes have increased both the number and type of faults that can occur [4]. Testing embedded resources in VLSI devices is especially difficult because their embedded nature makes them difficult to control and observe from the external chip I/O; furthermore, the number of external I/O is continually decreasing in proportion to the number of transistors on a single die [4]. While the number of I/O has increased by an order of magnitude for most VLSI devices, the number of transistors on a single die increased by more than 4 orders of magnitude over the same time period [4]. (This trend is commonly called Rent's Rule, for E. F. Rent of IBM, who was the first to investigate a relationship between the number

of I/O and the number of internal logic blocks in 1960 [5]). Due to the limited number of external I/O in proportion to the number of transistors on a chip, and without the inclusion of any additional test circuitry, the controllability and observability of most VLSI designs are severely limited during testing.

Another factor affecting testing of VLSI ICs is the cost of automatic test equipment (ATE). While the cost of manufacturing transistors in VLSI circuits has continued to decrease with each new technology node, the cost of testing has increased both in absolute terms and in proportion to overall manufacturing cost. In fact, the cost of testing a single transistor already exceeds its cost of production [3], and due to the ever increasing density and bandwidth of integrated circuits, testing costs will continue to rise. It is expected that by the year 2014, the cost of a leading edge VLSI test machine will exceed twenty million dollars [4]. Consequently, design for testability (DFT) methods, which incorporate additional test circuitry during the design phase to increase circuit controllability and observability during testing, are included in some form in virtually every VLSI design. Two of the most common DFT techniques are scan design and built-in self-test (BIST). Another DFT method, known as Boundary Scan or JTAG (Joint Test Action Group) [6], is usually included to facilitate board-level testing of systems with high pin-count and surface mount components [3] [7]. A recent offshoot of Boundary Scan, IEEE standard 1500-2005 [8], describes a scalable wrapper architecture and control mechanism for testing embedded cores in System-on-Chip (SOC) devices and the interconnect between cores [3]. The primary focus of this thesis will be on BIST as a solution for testing VLSI ICs.

1.1 Overview of Built-In Self-Test

BIST was introduced around 1980 as a way to test embedded cores in VLSI devices [4]. The basic idea of BIST is to incorporate extra circuitry and functionality in the device under test

such that the circuit can test itself [3] [4]. This implies that the circuit is capable of generating test patterns and compacting output responses. Therefore, BIST, in contrast to other techniques such as scan design which relies on externally applied test patterns, does not require costly ATE hardware. In addition, many BIST techniques are applicable at every level of the testing process, from wafer-level manufacturing test to board-level and in-system test. Another advantage of some BIST approaches when compared to scan-based test techniques is that patterns can be applied to the circuit under test and the output responses monitored at system speeds, which facilitates the detection of delay and coupling faults [4] [9].

A simple BIST architecture, shown in Figure 1.1, consists of a test pattern generator (TPG), output response analyzer (ORA), circuit under test (CUT), and some additional control circuitry [3] [4]. For system-level use of BIST, input isolation circuitry and a dedicated BIST controller must be included. The BIST controller can be used to initiate the BIST, initialize the CUT, activate the input isolation circuitry, and provide an indication when the test is complete. During off-line tests, the TPG generates a set of test patterns which are applied to the circuit under test (CUT) to sensitize potential fault sites, and the ORA compacts the output response of the CUT. At the conclusion of the test, the results are determined by examination of the ORA contents (generally, by comparison to the fault-free circuit "signature") [4].

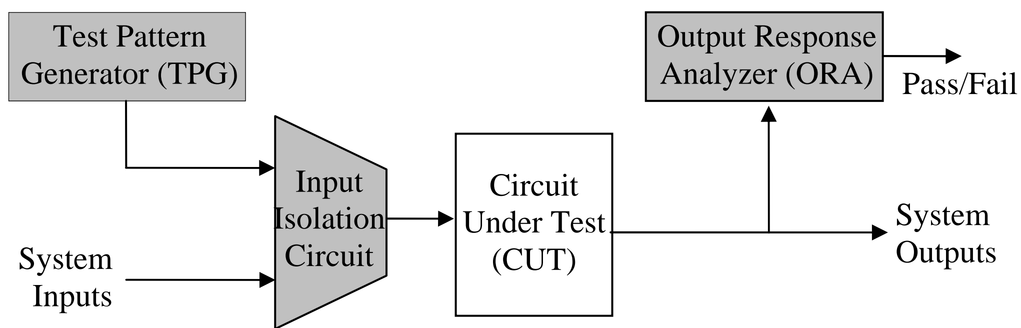


Figure 1.1: Basic BIST architecture [3]

There are some costs associated with BIST that must be taken into consideration. In ASICs, BIST requires additional circuitry and functionality that results in area and performance penalties. This additional circuitry is shown in gray in Figure 1.1. Typically, the performance penalty is minimal, amounting to no more than a multiplexer delay in the primary input data path and additional fan-out in the primary output data path of the circuit under test. The area penalty varies depending on the exact BIST architecture used (which is, in turn, usually a function of desired fault coverage and the type of circuit under test). This additional area is disadvantageous because larger chip areas result in fewer chips per wafer, and, therefore, higher cost per chip due to lower yield [4]. Also, some additional I/O pins may be required for activation of the BIST circuitry and results retrieval [4]. The inclusion of BIST also increases the design effort and risk to the project, because, on top of designing the system function, the BIST circuitry must also be designed and verified. However, most case studies have found that the benefits of BIST usually outweigh the costs (including addition design time and overhead) when included in a project [4], and many computer-aided design (CAD) tools now support automatic insertion of pre-engineered BIST circuitry during the design phase, which reduces the design effort and risk to the project.

1.2 Introduction to Field Programmable Gate Arrays (FPGAs)

Field Programmable Gate Arrays (FPGAs) are pre-fabricated semiconductor devices that can be programmed (*i.e.* configured) after manufacturing to perform complex sequential or combinational logic functions. Compared to standard-cell or custom ASIC designs, FPGAs provide lower non-recurring engineering costs and faster time-to-market [10]. The non-recurring engineering costs associated with the design and manufacture of FPGAs are initially absorbed by the manufacturer and are passed to the customer in the form of a higher price-per-part. This cost,

coupled with the cost of the additional logic required for programming of the device, makes the recurring costs of designs with FPGAs higher than those with ASICs. For these reasons, FPGAs are commonly used for rapid prototyping of designs prior to first silicon and in low-volume, highly-specialized digital systems (where the FPGA is used in lieu of an ASIC). An illustration of the total cost (*i.e.* recurring plus non-recurring costs) as a function of volume (number of parts) for a design implemented as a standard-cell ASIC, as a custom ASIC, and in an FPGA is shown in Figure 1.2 [10].

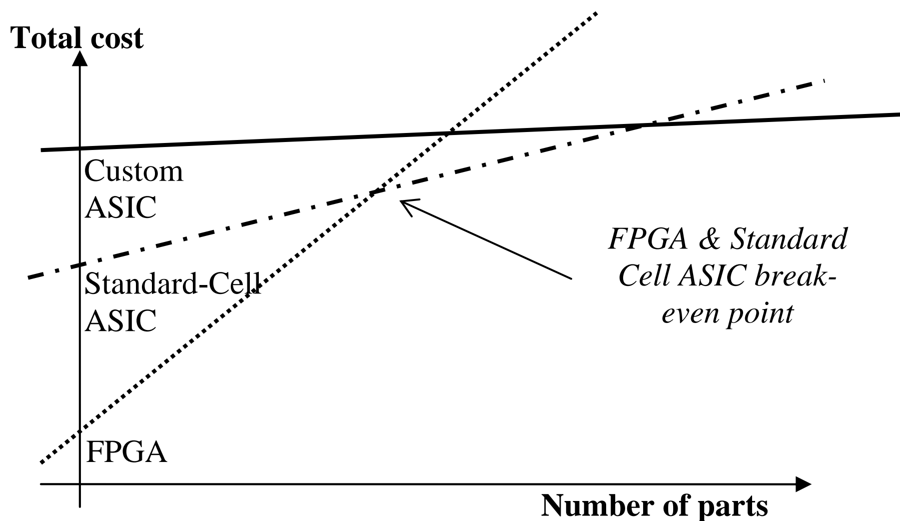


Figure 1.2: Typical custom ASIC, standard cell ASIC, and FPGA cost vs. volume

Due to the programmable nature of FPGAs, area, power and performance penalties are incurred for designs implemented in FPGAs when compared to the same design implemented as an ASIC. For several benchmark circuits implemented in both a 90 nm FPGA and 90 nm standard-cell ASIC, the FPGA implementation required between 18 and 35 times greater silicon area, and the critical path delay of the circuit increased by 3 to 4 times versus the ASIC implementation [11].

A typical FPGA is composed of an array of programmable logic blocks (PLB) (also called configurable logic blocks, or CLB) and input/output (I/O) cells connected by a

programmable interconnect network, as illustrated in Figure 1.3 [12]. Most modern FPGAs also include "hard" cores such as reduced instruction set computer (RISC) or complex instruction set computer (CISC) processors, digital signal processors (DSPs), random access memories (RAMs), and high-speed serializer/deserializer (SERDES) input/output (I/O) cells. These "hard" cores can perform certain common functions, such as multiply/accumulate or serialization/deserialization, with greater efficiency than can be achieved by implementing the same function in CLBs, which helps to reduce the performance/area penalties when compared with ASICs [11].

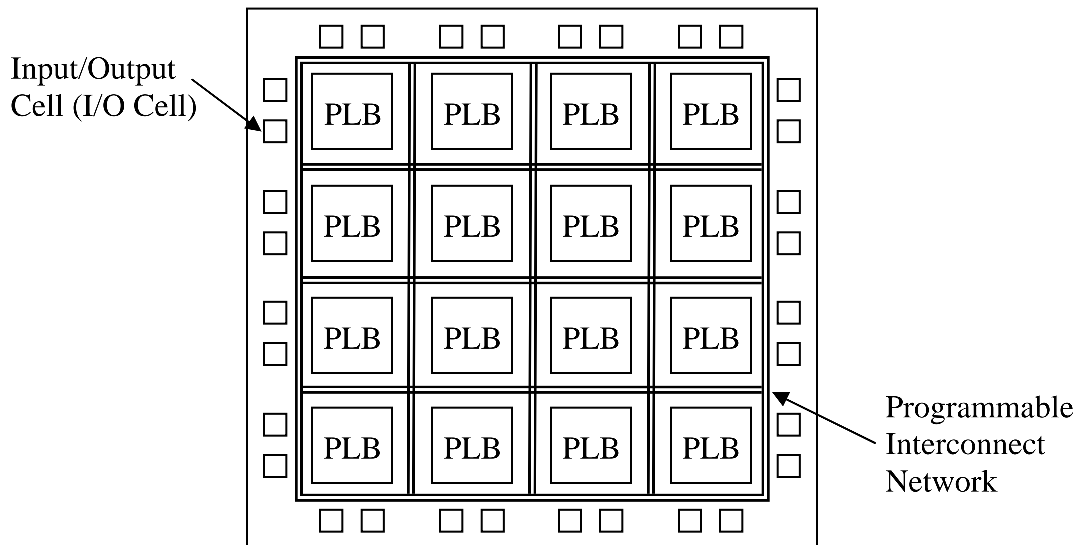


Figure 1.3: Typical FPGA architecture [12]

The front-end of the FPGA design process is identical to that for a standard-cell ASIC. However, the post synthesis design flow is much less complex for FPGA implementations. After behavioral simulation and functional verification, computer aided design (CAD) tools (usually supplied by the FPGA manufacturer, but also available through third parties) translate the digital designs in Hardware Description Language (HDL) or schematic form to a device specific netlist which maps the design into the FPGA's configurable logic and programmable routing network.

A configuration bit-file is generated from this netlist and downloaded to the configuration memory of the FPGA to implement the desired user function.

1.3 Overview of Virtex-5 FPGAs

This body of work is primarily concerned with Xilinx Virtex-5 FPGAs. Virtex-5 FPGAs are fabricated in a 1.0 V, 65 nm CMOS copper process with 12 metal layers [13]. The number of flip-flops and LUTs in a single Virtex-5 device ranges from 12,480 up to 207,360. As many as 1,200 user I/O are available in the highest pin-count package [13]. The configuration memory in all Virtex-5 devices is a large static random access memory (SRAM), ranging in size from 4.94 Mb (4,935,744 bits) to 82.7 Mb (82,687,488 bits) [14].

Each CLB in an FPGA consists of one or more basic logic elements. The Virtex-5 basic logic element, illustrated in Figure 1.4, comprises a six-input look-up table (LUT), a configurable flip-flop/latch (FF/LAT), a multiplexor to control the combinational output, and a multiplexor to control the registered output (FF/LAT input) [15].

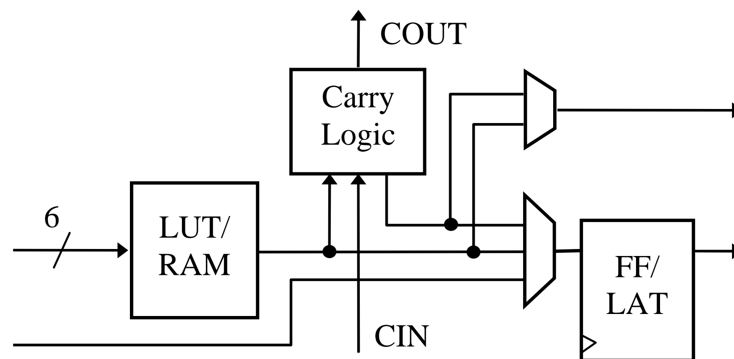


Figure 1.4: Simplified basic logic element

Additional dedicated carry logic is included to perform special logic and arithmetic functions. In some slices, the LUT can be configured as a small RAM, called a free RAM or LUT RAM, with an independent read and shared write address input. Four such logic elements

are grouped to form a slice, and two slices are grouped to form a complete configurable logic block (CLB), as illustrated in Figure 1.5. The logic blocks are replicated and tiled in columns and rows, as in Figure 1.4, and are connected via programmable switch-boxes to local and global routing resources. Larger devices include more CLBs, but the structure of the CLB is identical across all devices in the FPGA family [15].

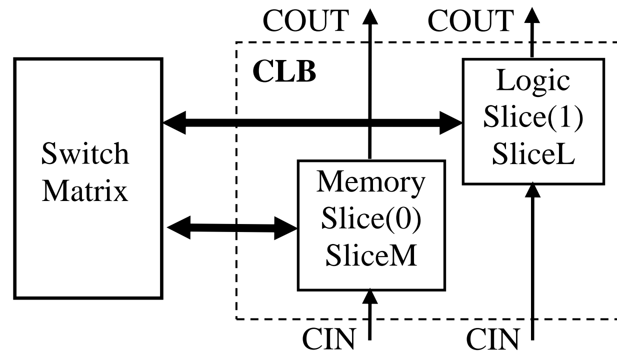


Figure 1.5: Virtex-5 configurable logic block [15]

The LUTs in Virtex-5 devices are designed with two outputs each. The primary output can utilize the full 64-bit LUT to implement any six variable Boolean function. The second output can be used to control the carry chain, or both outputs can implement two five variable Boolean functions for five shared inputs. Both outputs can be selected by the multiplexors for the registered or combinatorial CLB output paths. A block diagram of the Virtex-5 6-input LUT is shown in Figure 1.6 [16].

Select slices also support RAM and shift register modes of operation. Each LUT can be configured as a simple 64 x 1-bit or 32 x 2-bit RAM. Dynamic multiplexors in each slice allow for Shannon expansion of the four slice LUTs to form a 256 x 1-bit RAM. Additionally, the four slice LUTs can share address inputs to form a 32 x 8-bit RAM. Each LUT can also form a single 32-bit or two 16-bit shift registers. The four LUTs in the slice can be cascaded to form a 128-bit shift register or can operate in parallel form a 16 x 8-bit shift register in a slice [15] [16].

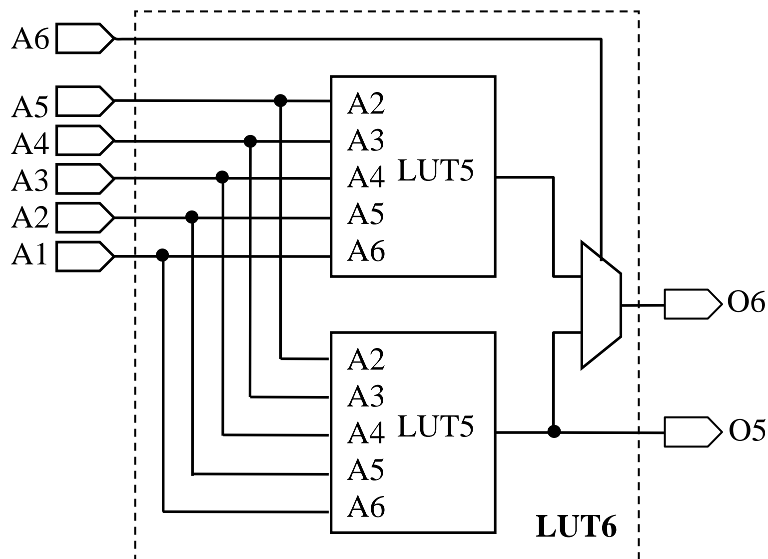


Figure 1.6: Virtex-5 6-Input LUT [16]

In addition to CLBs, every device in the Virtex-5 family includes DSP and Block RAM "hard" cores. Each DSP core can perform 25 x 18 2's complement multiplication, and includes an adder/subtractor/accumulator block. The DSP can also perform bit-wise logic operations including NOR, OR, AND, NAND, XNOR, and XOR. Up to five pipeline registers may be configured for use in the data path for increased throughput (up to 550 MHz) in high performance applications [17]. Each Block RAM core is 36 Kbit in size, with true dual-port read/write access to each memory element. Each of the read and write ports are configurable, such that the address and data bus widths can vary from 32K x 1-bit to 1K x 72-bit. In addition, the Block RAM can operate in a FIFO mode (with configurable data width and programmable almost-full and almost-empty flags) and/or in an error correction code (ECC) mode [15]. Some devices in the Virtex-5 family also include other "hard" cores such as gigabit transceivers, Ethernet MACs, PCI Express blocks, and/or Power PC processors [13].

1.4 BIST for FPGAs

Testing FPGAs is difficult when compared to testing ASICs because of their programmable nature and overall complexity [9]. Each of the programmable resources must be tested in all modes of operation to achieve high fault coverage. This implies that multiple re-configurations of the device are required during testing. Because the total test time is usually dominated by the time spent configuring the device under test, the size of FPGA configuration memories is also a factor in testing [9]. FPGAs are, in general, not well-suited for scan-based testing methods. However, the programmable nature of FPGAs allows for the creation of test circuitry in the programmable logic during testing. In addition, the regular structure of FPGAs makes pseudo-exhaustive test methods highly efficient [4] [9] [17] [18] [19].

BIST for FPGAs exploits the re-programmability of FPGAs to create BIST circuitry in the FPGA fabric during manufacturing and system-level off-line testing [4] [9] [17] [18] [19]. The only overhead is the external memory required to store the BIST configurations along with the time required to download and execute the BIST. No area overhead or performance penalties are incurred in the user function because the BIST logic is replaced by the intended system function after testing is complete. The BIST configurations are applicable to all levels of testing because they are independent of the intended system function and require no specialized external test fixture or equipment. Most research and development in BIST for FPGAs has focused on reducing the number of test configurations, reducing the size of test configuration files, and decreasing BIST execution time [4] [7] [8] [23]. Other research has focused on developing BIST techniques for the complex embedded cores included in many modern FPGAs, such as DSPs [24] and RAMs [3] [5]. This thesis presents new BIST approaches for the CLBs, I/O Tiles, and SEU detection cores in Virtex-5 FPGAs.

This thesis also presents a new approach to BIST for FPGAs that utilizes a soft-core processor configured in the fabric of the FPGA under test to execute the BIST sequence, including retrieval and analysis (fault diagnosis) of BIST results and reconfiguration of the FPGA for subsequent BIST configurations. The approach reduces the required number of configurations for BIST of any logic resource to a maximum of four, and by moving the complex BIST controller logic into the FPGA fabric, the external hardware requirements for BIST of FPGAs is greatly reduced. This approach is particularly useful in high-reliability and fault-tolerant applications, especially when fault-diagnosis is required.

1.5 Single Event Upsets in FPGAs

BIST is typically targeted at detecting manufacturing defects or "hard" faults that appear during normal operation. However, "soft" errors, known as Single Event Upsets (SEUs), are known to affect the configuration memory and other memory elements of FPGAs during normal operation. These errors are caused when charged particles, such as heavy ions or protons, travel through the FPGA, as illustrated in Figure 1.7 [27]. These particles can alter the state of any static memory element, resulting in an SEU [27] [28] [29]. While SEUs occur more frequently in high radiation environments such as space, they have also been experimentally observed in FPGAs at ground level [28] [29] [30]. Because the configuration memory of an FPGA establishes the overall system function performed by the FPGA, an SEU in the configuration memory can alter the FPGA functionality. This, coupled with the large size of the configuration memory, makes SEUs a significantly greater concern in FPGAs than in typical ASICs [31].

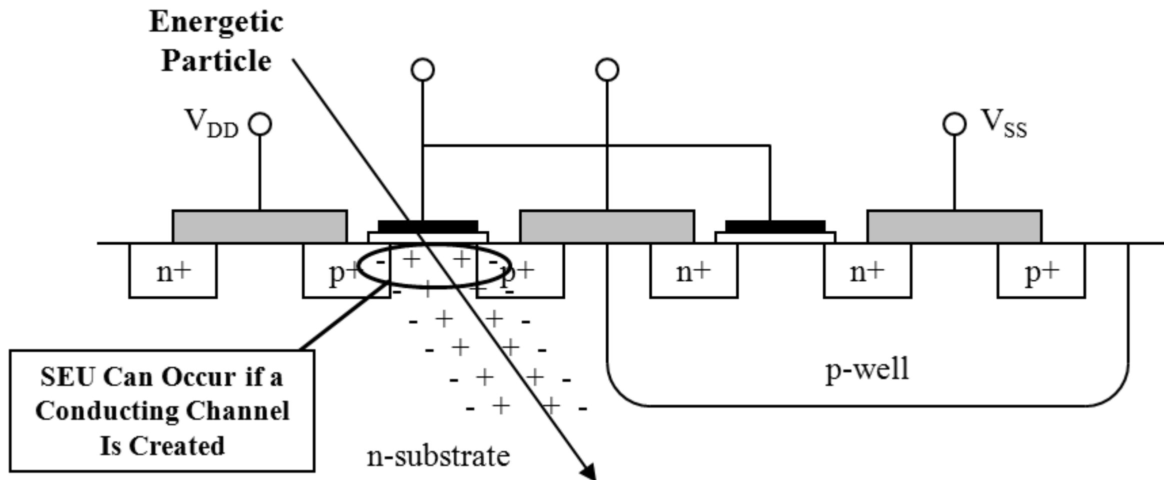


Figure 1.7: Illustration of a single-event effect in a CMOS inverter

Several methods exist to mitigate the effects of SEUs in FPGAs. The most common methods include power cycling, triple modular redundancy, redundant devices, and active configuration memory scrubbing [27]. Power cycling is essentially the simplest form of configuration memory scrubbing, because the entire configuration memory is refreshed (from a radiation hardened memory) each time that power is cycled off and on. When a power cycling mitigation scheme is employed, SEUs can persist in memory elements for a period of time equal to the power-cycling period. This approach is usually sufficient for non-critical applications in low radiation environments [27].

Triple modular redundancy creates three identical copies of the user function in the FPGA fabric and adds majority voters on the inputs to all flip-flops and on all primary outputs of the circuit [32]. This approach is very robust: any single SEU cannot cause the circuit to malfunction, and multiple SEUs must alter the same flip-flop input or primary output in two circuit copies on the same clock cycle in order for the error to propagate. However, the area penalty for any TMR approach is greater than 200% of the original circuit size, which increases system cost and power requirements. Also, circuit performance can be adversely impacted due

to the increased size of the circuit and inclusion of majority voters in critical paths [27]. Duplicating the user function in multiple FPGAs and performing voting on the outputs of the FPGAs in a radiation hardened device is the most robust from of SEU mitigation. However, designing systems with multiple FPGAs is both costly and difficult, and requires special design considerations such that the FPGAs remain synchronized after an SEU is repaired in any one of the devices [27].

Active configuration memory management (also called active configuration memory scrubbing) utilizes error correction code (ECC) stored with configuration data in the configuration memory to actively detect and repair SEUs [14]. The ECC, in conjunction with some additional user-accessible dedicated logic, can be used to detect SEUs in the configuration memory [15]. This approach incurs minimal area overhead, and SEUs persist for only a small window of time. The configuration management hardware may be hosted on an external radiation hardened FPGA, microprocessor, ASIC, or in the FPGA itself. However, in the latter case, the circuitry responsible for the repair of SEUs is also susceptible to SEUs [31]. Therefore, the area of the detection and repair circuitry should be minimized to decrease the probability of an SEU in that logic. An active configuration memory management approach for Xilinx Virtex-4 and Virtex-5 FPGAs that requires no additional external hardware is described in this thesis.

1.6 Verification by Fault Injection

During the development of BIST approaches for FPGAs, it is necessary to verify the fault coverage of the BIST configurations. It is difficult to find actual faulty devices and their usefulness is limited due to the fixed nature of the faults. Physical faults can be created by etching the packaged device and creating opens or shorts in routing resources that lie at the top level of interconnect metal for example, but once again the usefulness of these devices is limited.

A more efficient approach is to manipulate the configuration memory bits to emulate physical faults in the device [33] [34] [35] [36]. For example, a stuck-at fault in a look-up table (LUT) bit can be emulated by overwriting the particular configuration memory bit and setting it to the desired stuck-at fault value. SEUs, on the other hand, can be emulated by flipping the value of bits in the configuration memory. Shorts and opens in the interconnect network can be emulated along with almost any fault in the logic resources that can be controlled by configuration memory bits. An approach for the emulation of stuck-at faults and SEUs in the configuration memory of Virtex-4 and Virtex-5 FPGAs is presented in this thesis.

1.7 Thesis Statement

Testing FPGAs is difficult due to their high complexity, the limited observability and controllability of embedded cores, and their programmable nature. Also, the increasing density and large size of the configuration memory has made transient and on-line faults due to SEUs more common and of greater concern, even in fault-tolerant applications that operate at ground level. This work considers both "hard" faults due to manufacturing defects and device ageing as well as transient or "soft" faults induced by SEUs in Virtex-5 FPGAs. Furthermore, this work considers "hard" faults that may affect the detection and correction of SEUs by corrupting the dedicated SEU detection hardware in Virtex-5 FPGAs, and presents BIST approaches for this hardware. Other BIST methods are proposed as a solution to detect "hard" faults and manufacturing defects that can affect the configuration memory and programmable resources in Virtex-5 FPGAs, including the CLBs and I/O Tiles. A novel BIST approach for FPGAs that utilizes a soft-core processor configured in the fabric of the FPGA under test to perform complex functions such as reconfiguration of resources under test and fault diagnosis is also presented. Finally, a method for active detection and correction of temporary or "soft" errors by active

configuration memory management and without the requirement of additional external hardware is presented for Xilinx Virtex-4 and Virtex-5 FPGAs.

1.8 Thesis Format

This thesis is written in “publication format” as suggested by the Auburn University Graduate School *Electronic Thesis and Dissertation Guide*, and consists of conference and journal papers that were published (or accepted for publication) during the course of research conducted by the author while in the graduate program at Auburn University. A majority of the actual research and the writing of all published papers included in this thesis represents the efforts of the primary student author and not collaborators. Each paper is presented “as published”, with the exception of an acknowledgments section at the end of each chapter that provides the name, location, and date of publication of the original paper along with any information regarding relevant published papers that do not appear in this thesis. The papers are reformatted to comply with the guidelines set forth by the Graduate School. References are organized as follows: Each chapter in the body of the thesis contains its original list of references (numbered consecutively beginning at 1), such that the chapter may stand-alone and as it appears in the original published paper. In addition, a cumulative bibliography of all references cited in the thesis is included at the end of the thesis.

1.9 References

[1] G. Moore, “Cramming More Components onto Integrated Circuits,” *Proc. of the IEEE*, vol. 86, no. 1, pp. 82-85, 1998.

[2] Semiconductor Industry Association, *International Technology Roadmap for Semiconductors: 2007 edition*, http://public.itrs.net.

[3] Y. Min and C. Stroud, “Introduction,” in *VLSI Test Principles and Architectures*, L-T Wang, C-W Wu, and X. Wen, Eds., San Francisco: Morgan Kaufmann, 2006, pp. 1-33.

[4] C. Stroud, *A Designer's Guide to Built-In Self-Test*, Boston: Springer, 2002.

[5] P. Christie, D. Stroobandt, "The Interpretation and Application of Rent's Rule," *IEEE Trans. on VLSI Systems*, vol. 8, no. 6, pp. 639-648, 2000.

[6] *IEEE Standard Test Access Port and Boundary-Scan Architecture*, IEEE Std 1149.1-2001, New York, 2001.

[7] M. Bushnell and V. Agrawal, *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*, New York: Springer, 2000.

[8] *IEEE Standard Testability Method for Embedded Core-Based Integrated Circuits*, IEEE Std. 1500-2005, New York, 2005.

[9] L-T Wang, C. Stroud, and N. Touba, *System-on-Chip Test Architectures*, San Francisco: Morgan Kaufmann, 2007.

[10] M. Smith, *Application-Specific Integrated Circuits*, Addison-Wesley, 1997.

[11] I. Kuon and J. Rose, "Measuring the Gap Between FPGAs and ASICs," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol.26, no.2, pp.203-215, 2007

[12] S. Brown and J. Rose, "FPGA and CPLD architectures: a tutorial," *IEEE Design & Test of Computers*, vol.13, no.2, pp.42-57, 1996

[13] *Virtex-5 Family Overview*, DS100 (v5.0), Xilinx Inc., 2009.

[14] *Virtex-5 FPGA Configuration User Guide*, UG191 (v3.2), Xilinx Inc., 2008.

[15] *Virtex-5 FPGA User Guide*, UG190 (v 4.2), Xilinx Inc., 2008.

[16] A. Cosoroaba and F. Rivoallon, "Achieving Higher System Performance with the Virtex-5 Family of FPGAs," Xilinx Inc., San Jose, CA, 2006.

[17] *Virtex-5 FPGA ExtremeDSP Design Considerations: User Guide*, UG193 (v3.3), Xilinx Inc., 2009.

[18] M. Abramovici and C. Stroud, "BIST-based test and diagnosis of FPGA logic blocks," *IEEE Trans. on VLSI Syst.*, vol. 9, no. 1, pp. 159-172, 2001.

[19] S. Toutounchi and A. Lai, "FPGA test and coverage," *Proc. IEEE Int. Test Conf.*, pp. 599-607, 2002.

[20] J Sunwoo and C. Stroud, "BIST of Configurable Cores in SoCs Using Embedded Processor Dynamic Reconfiguration," *Proc. Int. SoC Design Conf.*, pp. 174-177, 2005.

[21] B. Dutton and C. Stroud, "Built-In Self-Test of Configurable Logic Blocks in Virtex-5 FPGAs," *Proc. IEEE Southeastern Symp. on System Theory*, pp. 230-234, 2009.

[22] B. Dutton and C. Stroud, "Built-In Self-Test of Programmable Input/Output Tiles in Virtex-5 FPGAs," *Proc. IEEE Southeastern Symp. on System Theory*, pp. 235-239, 2009.

[23] C. Stroud, S. Konala, P. Chen, and M. Abramovici, "Built-in self-test of logic blocks in FPGAs," *Proc. IEEE VLSI Test Symp.*, pp.387-392, 1996.

[24] M. Pulukuri and C. Stroud, "Built-In Self-Test of Digital Signal Processors in Virtex-4 FPGAs," *Proc. IEEE Southeastern Symp. on System Theory*, pp. 34-38, 2009.

[25] C. Stroud, S. Garimella and J. Sunwoo, "On-Chip BIST-Based Diagnosis of Embedded Programmable Logic Cores in System-On-Chip Devices," *Proc. ISCA Int. Conf. on Computers and Their Applications*, pp. 308-313, 2005.

[26] B. Garrison, D. Milton, and C. Stroud, "Built-In Self-Test for Memory Resources in Virtex-4 FPGAs," *Proc. ISCA Int. Conf. on Computers and Their Applications*, pp. 63-68, 2009.

[27] B. Bridgford, C. Carmichael, and C. Tseng, "Single-Event Upset Mitigation Selection Guide," XAPP987 (v1.0), Xilinx Inc., 2008.

[28] E. Normand, "Single Event Upset at Ground Level," *IEEE Trans. on Nuclear Science*, vol. 43, pp. 2742-2750, 1996.

[29] A. Lesea and P. Alfke, "Xilinx FPGAs Overcome the Side Effects of Sub-90 nm Technology," WP256 (v1.0.1), Xilinx Inc., 2007.

[30] A. Lesea, "Continuing Experiments of Atmospheric Neutron Effects on Deep Submicron Integrated Circuits," WP286 (v1.0), Xilinx Inc., 2008.

[31] K. Chapman and L. Jones, "SEU Stratagies for Virtex-5 Devices," XAPP864 (v1.0.1), Xilinx Inc., 2009.

[32] *Xilinx TRMTool User Guide: TMRTool Software Version 9.2i*, UG156 (v2.2), Xilinx Inc., 2009.

[33] P. Ellervee, J. Raik, K. Tammemäe and R. Ubar, "Environment for FPGA-based Fault Emulation," *Proc. Estonian Acad. Sci. Eng.*, vol. 12, pp. 323–335, 2006.

[34] T. Slaughter, C. Stroud, J. Emmert and B. Skaggs, "Fault Injection Emulation for Field Programmable Gate Arrays," *Proc. Int. Society for Optical Eng.*, vol. 4525, pp. 1-9, 2001.

[35] E. Johnson, M. Caffrey, P. Graham, N. Rollins and M. Wirthlin, "Accelerator Validation of an FPGA SEU Simulator," *IEEE Trans. on Nuclear Sci.*, vol. 50, no. 6, pp. 2147-2157, 2003.

[36] F. Kastensmidt, L. Carro and R. Reis, *Fault-Tolerance Techniques for SRAM-based FPGAs*, The Netherlands: Springer, 2006.

Chapter Two. Built-In Self-Test of Configurable Logic Blocks in Virtex-5 FPGAs

A Built-In Self-Test (BIST) approach is presented for the configurable logic blocks (CLBs) in Xilinx Virtex-5 Field Programmable Gate Arrays (FPGAs). A total of 17 configurations were developed to completely test the full functionality of the CLBs, including distributed RAM modes of operation. These configurations cumulatively detect 100% of stuck-at faults in every CLB. There is no area overhead or performance penalty and the approach is applicable to all levels of FPGA testing (wafer, package, and in-system). A novel output response analyzer (ORA) design, which is efficiently implemented in FPGAs, provides both an overall single-bit pass/fail result and optimal diagnostic resolution when faults are detected. The implementation of the BIST approach in all Virtex-5 FPGAs and experimental results are discussed.

2.1 Introduction And Background

Built-In Self-Test (BIST) for Field Programmable Gate Arrays (FPGAs) is typically targeted at manufacturing defects and operational faults that can appear at any point in the product life-cycle. As a result, BIST for FPGAs employs a defect-oriented test strategy [1]. Ideally, a BIST approach would be applicable to all levels of testing, from manufacturing test to in-system test, and would be entirely independent of the end user function. Additionally, the BIST would achieve maximal stuck-at fault coverage and would be executed at-speed to provide high fault coverage for a variety of fault models. When possible, high diagnostic resolution of detected faults is desired for fault-tolerant applications. This chapter presents a BIST approach

for the configurable logic blocks (CLBs) in Virtex-5 FPGAs that represents the culmination of over 15 years of work in FPGA BIST to address these concerns.

The first BIST for the configurable logic in FPGAs was proposed in [2]. The approach exploits the re-programmability of FPGAs to create BIST circuitry in the FPGA fabric during off-line testing. The only overhead is the external memory required to store the BIST and system function configurations along with the time required to download and execute the BIST. No area overhead or performance penalties are incurred since the BIST logic "disappears" after the test session. Furthermore, the tests are applicable at all levels of testing since they are independent of the system function and require no external test fixture or equipment. The basic idea for the BIST is to configure some of the CLBs as Test Pattern Generators (TPGs) and Output Response Analyzers (ORAs) while configuring other CLBs as blocks under test (BUTs). The BUTs are repeatedly configured until they have been tested in every mode of operation [1]. These tests achieve maximal fault coverage by applying pseudo-exhaustive test patterns such that each sub-circuit of the BUT is exhaustively tested [2].

Several examples of BIST for the CLBs in FPGAs have been published, with each offering some improvement over the previous approach. Reference [3] introduced Boundary Scan as a means of controlling the BIST sequence. Xilinx engineers, in [4], introduced a set of iterative array logic tests with similarities to the approach presented in [2] and [3]. The general BIST approach, which is independent of the CLB array size, can also be adapted for on-line BIST techniques, as discussed in [5]. Previous examples of the implementation of this BIST approach on Xilinx 4000, Spartan, Virtex-I, Spartan-II and Atmel FPGAs are contained in [6], [7], and [8]. Partial reconfiguration was used in [9] to reduce the overall download and test times as well as system down time.

The BIST approach for Virtex-5 FPGAs builds primary on the previous work in [2], [3], [8], and [10]. However, our approach offers an improved ORA architecture and fewer total test configurations. We also improve the accuracy of the fault simulation models and add verification of the configurations on the target device via configuration memory bit fault injection. The remainder of this chapter is organized as follows. Section 2.2 gives an overview of the CLB architecture in Virtex-5 FPGAs. Section 2.3 describes the BIST approach and implementation specific to Virtex-5 FPGAs. Section 2.4 describes the experimental result and verification of the BIST. Section 2.5 summarizes and concludes the chapter.

Table 2.1: List of acronyms

Acronym	Definition	Acronym	Definition
CLB	Configurable Logic Block	BUT	Block Under Test
BIST	Built-in Self-test	LUT	Look-Up Table
ORA	Output Response Analyzer	SliceL	Logic Slice
TPG	Test Pattern Generator	SliceM	Memory Slice

2.2 Overview of Virtex-5 CLBs

The basic Virtex-5 logic element, illustrated in Figure 2.1, is composed of a 6-input look-up table (LUT), a configurable flip-flop/latch, and multiplexers to control the combinational logic output and the registered output (flip-flop/latch input). Additional dedicated fast carry logic is included to perform special logic and arithmetic functions. In some slices, the LUT can be configured as a small RAM, called a distributed RAM or LUT RAM, or as a shift register [11]. Four such basic logic elements are grouped to form a slice, and two slices are grouped to form a complete CLB, as shown in Figure 2.2 [11]. Each CLB is connected by a switch matrix to local and global programmable routing resources. Identical CLBs are tiled in columns and rows with larger devices including more columns and/or rows of CLBs. Additionally, the structure of the CLB is identical across all devices in the Virtex-5 family. The 6-input LUTs are

designed with two outputs each. The primary output, O6, can utilize the full 64-bit LUT to implement any 6-variable Boolean function. The secondary output, O5, can be used to initialize the carry chain, or both the O5 and O6 output can implement an independent 5-variable Boolean function for five shared inputs. Either LUT output can be selected by the configuration multiplexers for the registered or combinatorial CLB output paths [11].

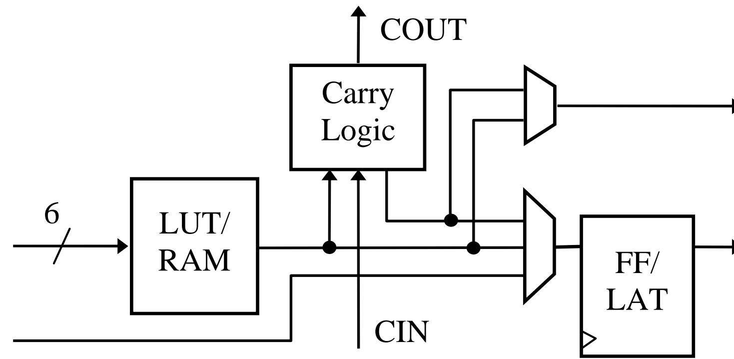


Figure 2.1: Simplified basic logic element

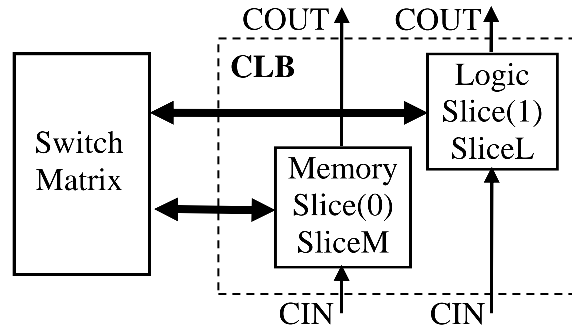


Figure 2.2: Virtex-5 configurable logic block [11]

Some slices (specifically the lower slice in every other column of CLBs and both columns to the left of a digital signal processor column) also support RAM and shift register modes of operation. The LUT RAMs in each slice have independent read address inputs and share a set of write address inputs. The independent read inputs facilitate the construction of dual-port RAMs within a slice. Each LUT can be configured as a simple 64×1-bit or 32×2-bit RAM. Dynamically controlled multiplexers in each slice allow the four LUTs to form a 256×1-

bit RAM. Additionally, the four LUTs can share five read address inputs and utilize eight independent data inputs to form a 32×8-bit RAM. Each LUT can also form a single 32-bit or two 16-bit shift registers. The four LUTs can be cascaded to form a 128-bit shift register or can operate in parallel form a 16×8-bit shift register bank [11].

2.3 BIST Approach And Architecture

The BIST approach takes advantage of the regular structure of FPGAs by using comparison-based ORAs to compare the outputs of multiple identical BUTs. This detects all faults affecting any combination of BUTs (since all fault-free BUTs must produce the same pattern) so long as all of the BUTs compared by a set of ORAs do not fail identically and at the same time [3]. Since a faulty TPG could cause a faulty BUT to escape detection, multiple identical TPGs are used to drive alternating BUTs. This eliminates the assumption that the TPGs are fault-free because, with multiple identical TPGs, a faulty TPG will cause the outputs of some of the BUTs to disagree, resulting in ORAs reporting failures.

The CLB BIST architectures can be divided into two categories based on the slice mode being tested. The first set of configurations tests every CLB in the FPGA in SliceL (logic) mode of operation. The second set of configurations tests every SliceM. Only those slices which support SliceM (memory) mode are tested during the second set of configurations.

In SliceL BIST architecture, alternating columns of CLBs are configured as ORAs and BUTs, as illustrated in Figure 2.3. The set of BIST configurations is repeated twice with the roles of the CLBs reversed such that every CLB serves both as ORA and as BUT. Two outputs of each BUT are compared by an ORA with the outputs of two adjacent identically configured BUTs in the same row, as shown in Figure 2.4. A mismatch of two identically configured BUT outputs latches a logic 0 in the ORA flip-flop. Otherwise, a logic 1 is retained in the ORA and is

interpreted as a passing result at the end of the test sequence. Traditionally, the results of the BIST are recovered via partial configuration memory readback where the contents of every ORA are retrieved from the configuration memory. However, we use a new ORA design that utilizes the dedicated carry logic in the CLB to form an iterative-OR of the ORA outputs. In each ORA, a passing result of logic 1 selects the Carry-in input, which is the Pass/Fail result of the previous ORA.

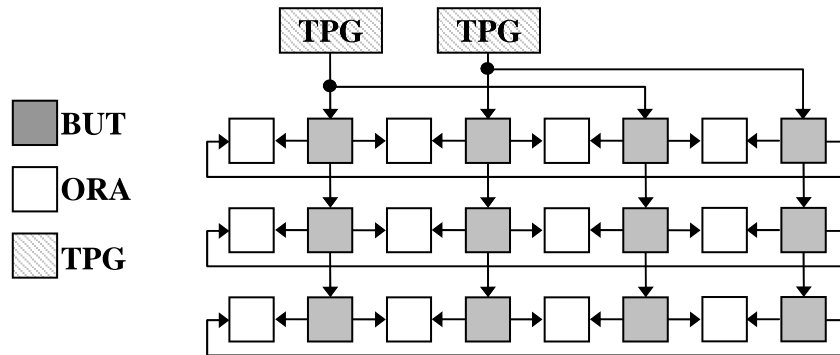


Figure 2.3: Circular comparison architecture

The Carry-in input of the first ORA in the iterative-OR chain is connected to Boundary Scan Test Data In (TDI), with the output of the last ORA connected to Test Data Out (TDO). If any ORA in the chain registers a failure, a logic 0 on the output of that ORA will select the logic 1 input of the carry chain multiplexer which translates to a logic 1 on TDO. Otherwise, TDO passes the state of TDI such that by toggling TDI and observing TDO, the integrity of the iterative-OR chain can be verified at the end of the BIST sequence. If the output of the OR chain indicates a failure (TDO is a logic 1 regardless of the state of TDI), the contents of the ORAs can be retrieved via partial configuration memory readback to determine the location(s) of the failing BUT(s). This facilitates the single-bit pass/fail indication for faster test time without sacrificing diagnostic resolution for fault-tolerant applications.

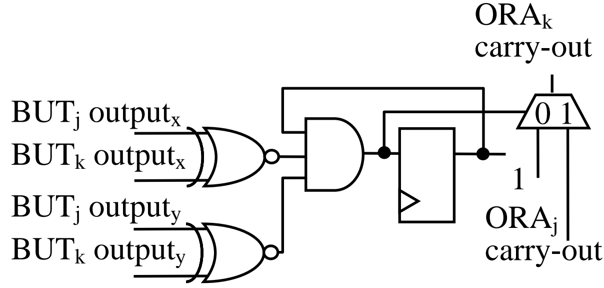


Figure 2.4: Equivalent ORA architecture

In Virtex-5 FPGAs, the carry-in of the bottom CLB and the carry-out of the top CLB in each column are not connected. To continue the carry chain, the carry-out of the top ORA in one column is connected to the D output and is routed to the AX input of the bottom ORA in an adjacent column. The AX input is selected as the carry-chain input in the bottom ORA in each column. In the ORA, each LUT is programmed with the hexadecimal value 0x90090000FFFFFFFF. By tying the A6 LUT input to logic 1, the O6 LUT output reads only the upper 32-bits of the LUT which implements the comparison ORA equation shown in Equation 2.1, while the O5 output reads only the lower 32-bits of the LUT (which controls the carry chain multiplexer for the iterative-OR chain).

$$O6 = \overline{(\mathrm{A1} \oplus \mathrm{A2})} \bullet \overline{(\mathrm{A3} \oplus \mathrm{A4})} \bullet \mathrm{A5} \qquad (2.1)$$

The architecture of the Virtex-5 CLBs requires a minimum of six configurations to test each of the 6 inputs to the flip-flop input multiplexers, (A-C)FFMUX. The first five of these configurations can also test the 5 inputs to the combinational logic output multiplexers (A-D)OUTMUX. Alternating XOR and XNOR functions in the LUTs detects every LUT stuck-at fault in two BIST configurations. Multiple identical TPGs are implemented in a column of embedded digital signal processors (DSPs) and drive alternating columns of BUTs. This reduces loading on the TPGs in large devices and eliminates the assumption that the TPG is fault-free. The DSPs are configured to accumulate a large prime number placed on the DSP inputs. This

number, 0xCA6691, was shown in [12] to produce an exhaustive sequence of 12-bit test patterns in 2^{12} clock cycles with a relatively high number of transitions in the most significant bits of the accumulator output. Virtex-5 CLBs require at least 12 TPG lines for pseudo-exhaustive testing, and, therefore, 4,096 clock cycles for the exhaustive set of test patterns to be produced by the accumulator. Six of the TPG outputs fan out to the inputs of each of the four LUTs. Adjacent LUTs are alternately programmed with XOR and XNOR functions such that adjacent LUTs will produce opposite logic values. Another six TPG lines exercise the AX, BX, CX, DX, CE, and SR slice inputs with pseudo-exhaustive test patterns. A total of 12 SliceL BIST configurations are generated, such that every CLB is a BUT for six configurations and an ORA for another six configurations. A summary of the SliceL BIST configurations is given in Table 2.2.

Table 2.2: SliceL logic BIST configurations

ConFigure#	A-D LUTs	FF/Latch	CYINIT	CLKIINV
#1	XOR/XNOR	FF INIT1	#OFF	CLK
#2	XNOR/XOR	FF INIT0	AX	CLK
#3	XOR/XNOR	FF INIT0	0	CLK
#4	XNOR/XOR	LAT INIT1	1	CLK
#5	XOR/XNOR	FF INIT0	0	CLK
#6	XNOR/XOR	FF INIT1	AX	CLK_B
ConFigure#	**A-D FFMUX**		**A-D MUX**	
#1	O6, O6, O6, O6		CY, CY, CY, CY	
#2	O5, O5, O5, O5		XOR, XOR, XOR, XOR	
#3	AX, BX, CX, DX		O5, O5, O5, O5	
#4	XOR, XOR, XOR, XOR		O6, O6, O6, O6	
#5	CY, CY, CY, CY		F7, F8, F7, CY	
#6	F7, F8, F7, DX		F7, F8, F7, CY	

Every other CLB column contains a SliceM. In addition, the CLB column to the left of a DSP column contains a SliceM and, in SX devices, the second CLB column to the right of a DSP column contains a SliceM. In columns containing SliceMs, only the bottom slice in each CLB is a SliceM. Therefore, every SliceM can be tested simultaneously since there is at least one SliceL

for every SliceM (located in the same CLB) that can serve as an ORA. The ORAs for the SliceM BIST architecture are the same as those used in the SliceL BIST architecture, including the iterative-OR chain. However, the circular comparison chain is formed along each column containing SliceMs by comparing the outputs of each BUT with the identically configured BUT in an adjacent row. A 2048×18-bit block RAM, effectively configured as a ROM, is used to store deterministic test patterns and, in conjunction with a DSP configured as an address counter, forms a TPG. Multiple identical TPGs are configured to drive alternating rows of BUTs. The SliceM BIST configurations are summarized in Table 2.3. To test the LUT RAMs in single-port modes (configurations #1 and #2), the block RAMs are initialized with the test patterns for a March Y test algorithm. A March Y RAM test requires 8*N* test patterns, where *N* is the number of address locations [10] [13]. For the remaining configurations, the block RAMs are initialized with test patterns for a dual-port RAM test algorithm [1] [6].

Table 2.3: SliceM BIST configurations

ConFigure#	**RAM mode**	**DI1MUX**	**WEMUX**	**FFMUX**
#1	SPRAM64	DX	CE	O6
#2	SPRAM32	A-DX	CE	O6
#3	DPRAM32	DX	WE	O5
#4	SRL32	MC31	WE	MC31
#5	SRL16	A-DX	WE	O6
ConFigure#	**OUTMUX**	**WA8used**	**WA7used**	**BIST CCs**
#1	O6	0	0	2,048
#2	O6	#OFF	#OFF	2,048
#3	O6	#OFF	#OFF	2,048
#4	O6	#OFF	#OFF	2,048
#5	MC31	#OFF	#OFF	2,048

2.4 Experimental Results

The BIST configurations were developed using accurate gate-level models of the Virtex-5 CLB. The SliceL and SliceM were modeled separately for fault simulation. For both SliceL

and SliceM, the BIST configurations and their associated fault coverage were first optimized using these gate-level models. The single stuck-at gate-level fault coverage for SliceL and SliceM BIST configurations obtained from fault simulations of these models are summarized in Figure 2.5 and Figure 2.7, respectively.

The BIST configurations were then verified on Virtex-5 LX30T and SX35T devices via configuration memory bit fault injection. Using the fault injection approach, configuration memory bits can be manipulated to emulate physical faults in the FPGA core including shorts and opens in programmable interconnect as well as almost any fault in logic resources controlled by a configuration memory bit. Configuration bits controlling the SliceLs and SliceMs were injected with faults and the BIST configurations were executed with the faulty configuration on the device. The BIST results of the faulty configuration are retrieved via partial configuration memory readback. The fault injection results show that the 17 BIST configurations cumulatively detect every configuration memory bit fault in every CLB. The results of the fault injection for SliceL BIST are shown in Figure 2.6. The similarity of the fault injection results and fault simulation results serve as a good indicator of the accuracy of the gate-level fault models, which include every stuck-at fault in the CLB (including configuration memory bits). Figure 2.7 and Figure 2.8 summarize the fault simulation results and the results of configuration memory bit fault injection, respectively, for the SliceM BIST configurations. It should be noted that three of the SliceM faults are detected by SliceL configurations.

There are two methods by which the results of the BIST sequence can be obtained. First, the single bit pass/fail result can be determined via the TDO output of the ORA iterative-OR chain. However, the location of failing BUTs cannot be determined using this method. Another option is to perform a partial configuration memory readback to determine the contents of each

ORA at the end of the BIST. By this method, the location of the failing BUT(s) can be easily determined with diagnostic resolution of LUT or flip-flop. To minimize test time and achieve maximum fault resolution, a combination of the two methods is used. First, the pass/fail status of the BIST is determined by observing TDO. If TDO presents a logic 1 regardless of the state of TDI, at least one ORA has observed a failure. Partial configuration memory readback can then be used to obtain the locations of the failing ORA(s) and, thereby, determine the location(s) of the faulty BUT(s).

We have developed two C programs that automatically generate the 17 BIST configurations for all Virtex-5 LX, LXT, SXT, and FXT devices. Table 2.4 summarizes the total download file size for the 17 BIST configurations, the maximum BIST clock frequency, and the total number of BIST clock cycles for full chip tests on several Virtex-5 devices. The total full chip test time for serial and parallel configuration interfaces is summarized in Figure 2.9 and Figure 2.10. The calculated test time assumes a 40 MHz BIST clock for all configurations and devices. However, on most devices, the BIST configurations can operate at higher clock frequencies.

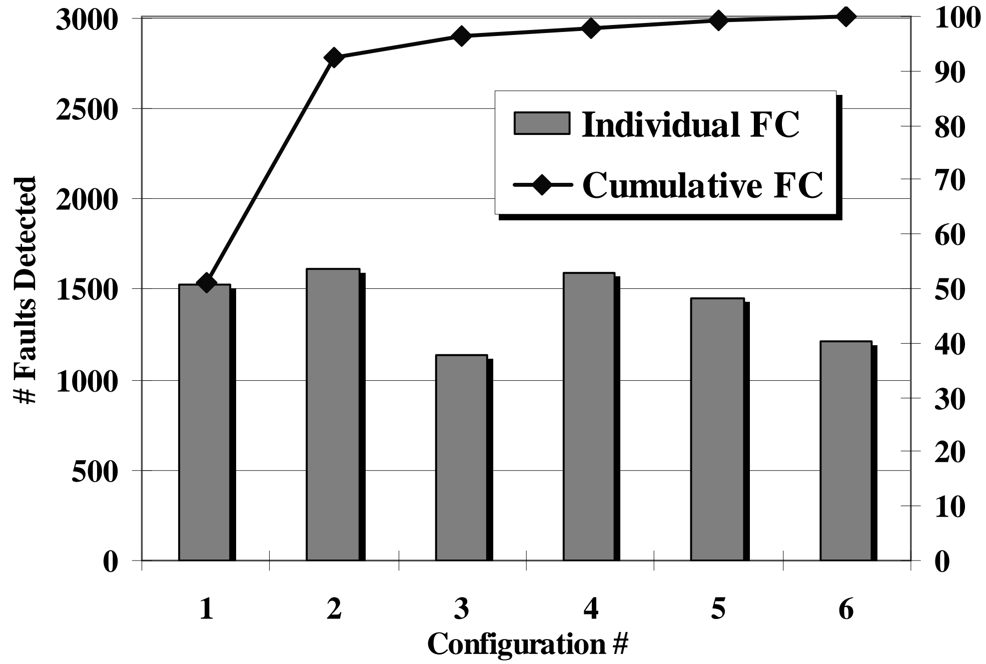


Figure 2.5: SliceL fault coverage (simulation)

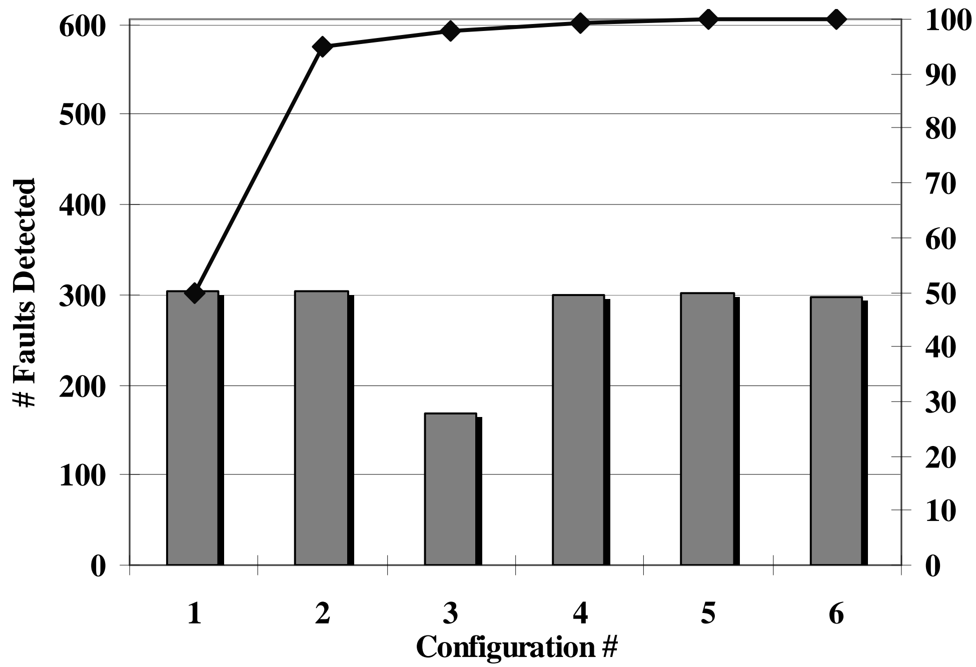


Figure 2.6: SliceL fault coverage (fault injection)

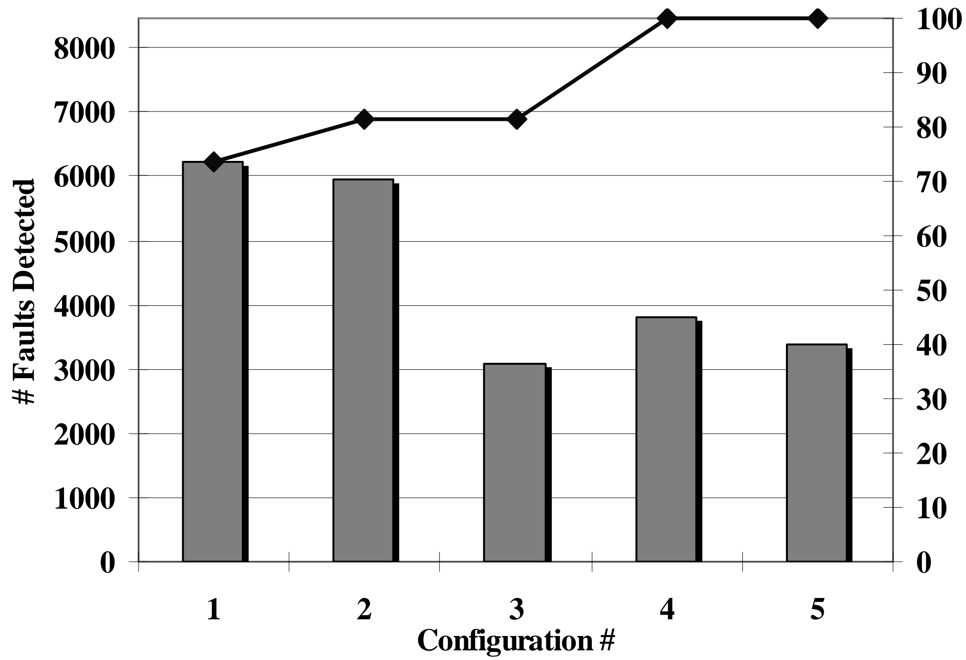


Figure 2.7: SliceM fault coverage (simulation)

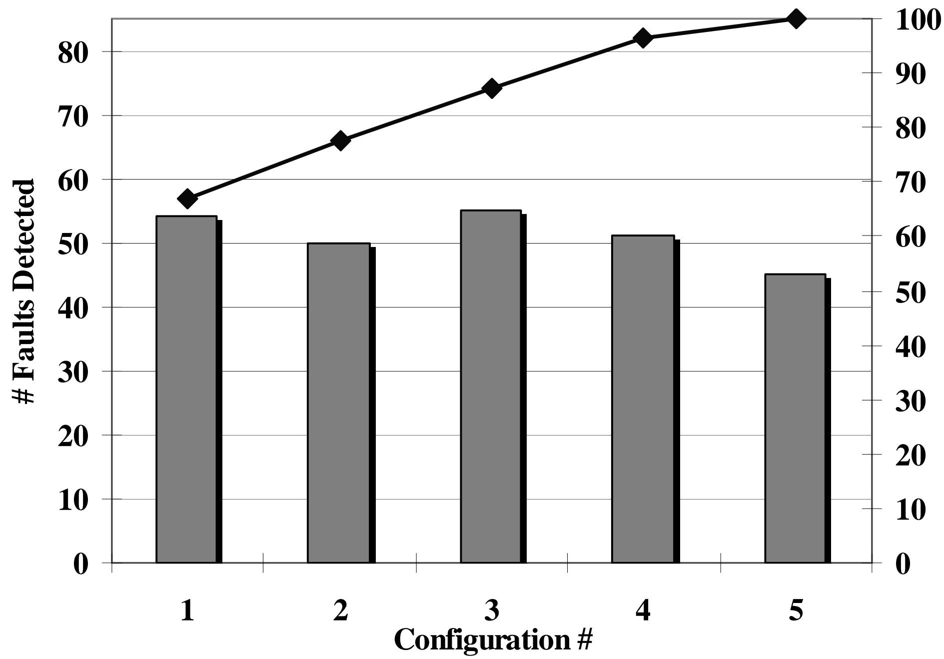


Figure 2.8: SliceM fault coverage (fault injection)

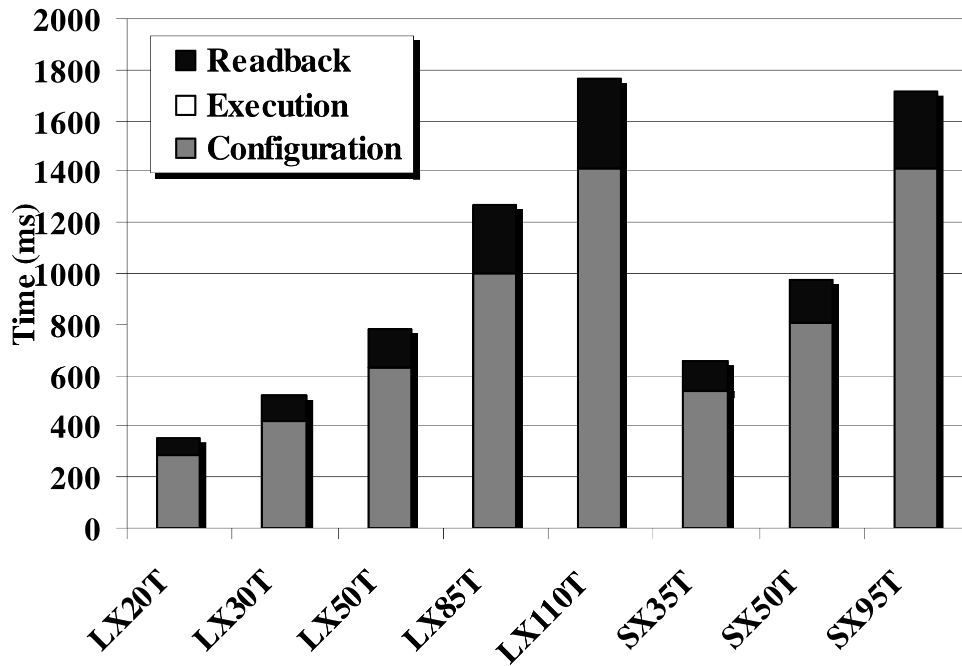


Figure 2.9: Boundary Scan interface test time

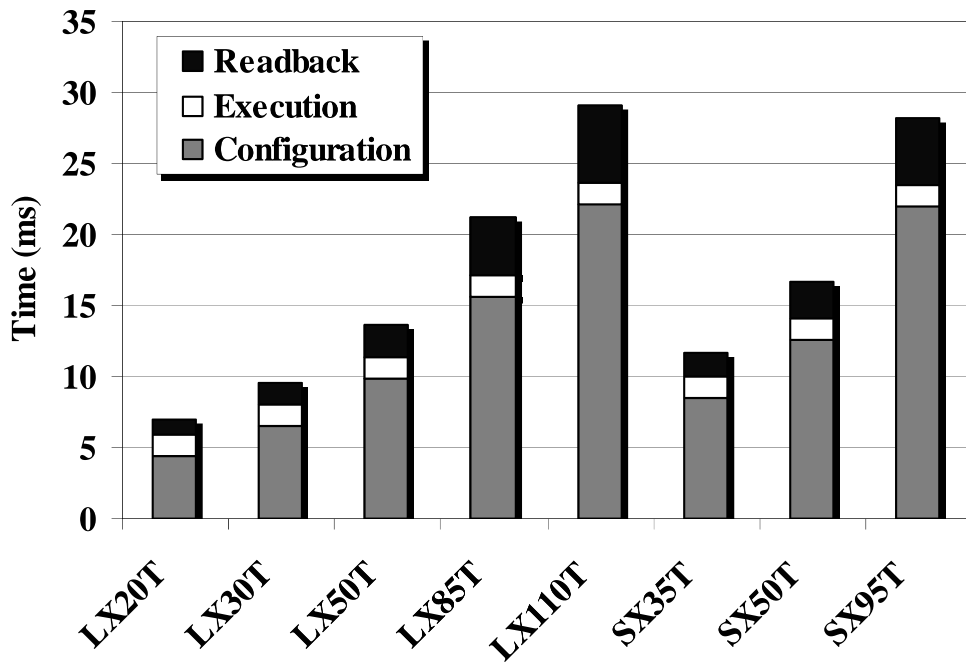


Figure 2.10: 32-bit parallel interface test time

In early FPGAs, all LUTs were able to function as small RAMs such that the first BIST configuration applied typically tested the LUTs in the RAM mode of operation. Using this approach, the first BIST configuration was able to detect most faults that could affect the LUT [2]. When combined with a simultaneous test of the flip-flop, the first BIST configuration was able to achieve around 80% fault coverage. A similar characteristic can be observed in the first SliceM BIST configuration in Figure 2.7, which achieves greater than 70% fault coverage. However, current FPGAs, such as Virtex-4 and Virtex-5, limit the number of LUTs that can function as small RAMs. Therefore, two BIST configurations are required (with alternate XOR and XNOR programming) to detect most of the faults in all LUTs. This can be observed in Figure 2.5, where the cumulative fault coverage after the first configuration reaches 51% and after two configurations exceeds 92%.

Table 2.4: CLB BIST totals (17 configurations)

Device	Total ConFigure Size (kB)	Max. BIST Clock Freq.	BIST CCs
LX20T	1,762	90.7 MHz	59,392
LX30T	2,630	74.0 MHz	59,392
LX50T	3,930	74.4 MHz	59,392
LX85T	6,265	58.2 MHz	59,392
LX110T	8,837	58.0 MHz	59,392
SX35T	3,378	59.2 MHz	59,392
SX50T	5,041	61.1 MHz	59,392
SX95T	8,818	44.7 MHz	59,392

2.5 Summary And Conclusions

A BIST approach for testing the CLBs in Virtex-5 FPGAs was presented. A total of 17 test configurations were developed to achieve 100% stuck-at fault coverage in every CLB. Twelve of these configurations pseudo-exhaustively test every SliceL and every SliceM in the SliceL mode. Another five configurations test every SliceM in their RAM and shift register

modes of operation. The BIST configurations were developed using accurate gate-level fault models of the CLB and verified using configuration memory bit fault injection. A novel ORA design provides a single bit pass/fail result for each BIST sequence and is independent of the configuration interface. Optional partial configuration memory readback provides optimal diagnostic resolution for fault-tolerant applications when the pass/fail output indicates failures. As a result, the BIST approach is applicable to all levels of FPGA testing including manufacturing testing and in-system testing for fault-tolerant applications. We modified SliceL BIST to support FXT devices by creating two circular comparison chains across rows directly above the PowerPC core because CLBs above the PowerPC have no carry-in routing. We have also applied this approach to Virtex-4 devices resulting in 20 and 5 BIST configurations for SliceL and SliceM tests, respectively, compared to 31 total configurations for Virtex-4 CLBs reported in [8]. Our Virtex-4 CLB BIST also includes the new ORA design for single bit pass/fail indication.

2.6 Acknowledgements

The contents of this chapter were published under the title "Built-In Self-Test of Configurable Logic Blocks in Virtex-5 FPGAs" in *Proceedings of the 41st IEEE Southeast Symposium on System Theory*, 2009, pp. 230-234. Prof. Charles Stroud is a co-author on the paper. The design of the ORA presented in this paper is protected by U.S. Provisional Patent #61/196,964, 2008, "Output Response Analyzer for System-Level Test of Field Programmable Gate Arrays". The student author and committee chair Prof. Charles Stroud are co-applicants on the provisional patent. A majority of the actual research and the writing of the published paper represents the efforts of the primary student author and not collaborators, and the research represents work performed while in the graduate program at Auburn University.

2.7 References

[1] L-T Wang, C. Stroud, and N. Touba, *System-on-Chip Test Architectures*, Morgan Kaufmann, 2007.

[2] C. Stroud, S. Konala, P. Chen, and M. Abramovici, “Built-in self-test of logic blocks in FPGAs,” *Proc. IEEE VLSI Test Symp.*, pp.387-392, 1996.

[3] M. Abramovici and C. Stroud, “BIST-based test and diagnosis of FPGA logic blocks,” *IEEE Trans. on VLSI Syst.*, vol. 9, no. 1, pp. 159-172, 2001.

[4] S. Toutounchi and A. Lai, “FPGA test and coverage,” *Proc. IEEE Int. Test Conf.*, pp. 599-607, 2002.

[5] M. Abramovici, C. Stroud, and J. Emmert, “Online BIST and BIST-based diagnosis of FPGA logic blocks,” *IEEE Trans. on Very Large Scale Integr. (VLSI) Syst.*, vol.12, no.12, pp. 1284-1294, 2004.

[6] C. Stroud, K. Leach, and T. Slaughter, “BIST for Xilinx 4000 and Spartan series FPGAs: a case study,” *Proc. IEEE Int. Test Conf.*, pp. 1258-1267, 2003.

[7] C. Stroud, J. Harris, S. Garimella, and J. Sunwoo, “Built-in self-test for system-on-chip: a case study,” *Proc. IEEE Int. Test Conf.*, pp. 837-846, 2004.

[8] S. Dhingra, D. Milton, and C. Stroud, “BIST for logic and memory resources in Virtex-4 FPGAs,” *Proc. IEEE North Atlantic Test Workshop*, pp. 19-27, 2006.

[9] S. Dhingra, S. Garimella, A. Newalker, and C. Stroud, “Built-in self-test of Virtex and Spartan II FPGAs using partial reconfiguration,” *Proc. IEEE North Atlantic Test Workshop*, pp. 7-14, 2005.

[10] C. Stroud and S. Garimella, “BIST and diagnosis of multiple embedded cores in SoCs,” *Proc. Int. Conf. on Embedded Systems and Applications*, pp. 130-136, 2005.

[11] *Virtex-5 FPGA User Guide*, UG190 (v 4.2), Xilinx Inc., San Jose, CA, May 2008.

[12] S. Gupta, J. Rajski, and J. Tyszer, “Test pattern generation based on arithmetic operations,” *Proc. IEEE Int. Conf. on Computer-Aided Design*, pp. 117-124, 1994.

[13] A. van de Goor, *Testing Semiconductor Memories Theory and Practice*, John Wiley and Sons, 1991.

Chapter Three. Built-In Self-Test of Programmable Input/Output Tiles in Virtex-5 FPGAs

A Built-In Self-Test (BIST) approach is presented for the logic resources in the programmable input/output (I/O) tiles in Virtex-5 field programmable gate arrays (FPGAs). A total of 15 BIST configurations were developed to test the I/O cell programmable logic resources in all modes of operation. The approach utilizes dedicated I/O buffer bypass routing in the I/O tile such that the BIST is package independent and applicable to all levels of testing from wafer-level to system-level. The approach offers control of BIST execution and maximal diagnostic resolution of faulty I/O tiles for device and package independent testing. Either the Boundary Scan interface or a simple system-level interface may be used for BIST execution, control, and diagnosis independent of the configuration interface. Experimental results are presented including fault detection capabilities.

3.1 Introduction

The input/output (I/O) buffers of JTAG compliant devices are typically tested using the Boundary Scan EXTEST feature [1]. However, field programmable gate arrays (FPGAs) have a significant amount of configurable logic resources associated with the I/O buffers that cannot be tested in this manner. These configurable logic resources typically include multiplexers and flip-flops/latches, as illustrated in Figure 3.1, for improving system timing specifications such as set-up and hold times as well as clock-to-output delay. Additional logic resources are included to support single data rate (SDR) and double data rate (DDR) transmission and reception as well as for serialization/de-serialization (SerDes) modes of operation. In Xilinx Virtex-5 FPGAs, for

example, there are at least 32 multiplexers and 47 flip-flops included in the configurable logic associated with each I/O cell to support various modes of operation. The Boundary Scan INTEST feature can be used to test the configurable logic resources in an I/O cell [1]. However, the INTEST feature is supported by few FPGA manufacturers. While there has been some prior work in testing I/O cells [2][3][4][5], previous work in Built-In Self-Test (BIST) for FPGAs has largely overlooked I/O cells and their associated logic resources. However, it has been observed that the programmable logic in unused or un-bonded I/O cells is sometimes used by FPGA synthesis tools for implementing system logic functions [5].

The work presented in this chapter builds primarily on the prior work in [5], in which an I/O cell BIST architecture was proposed and implemented for Atmel AT40K series FPGAs and Atmel AT94K series programmable system-on-a-chip (SoC) [6]. However, this chapter offers several improvements over that previous BIST approach. In addition, this chapter describes the actual implementation, operation, and verification of BIST configurations developed for Virtex-5 FPGAs [7] whose I/O cells are much more complex than those found in the AT40K and AT94K devices [6]. The BIST configurations presented here test the full functionality of logic resources included in the Virtex-5 I/O cells including input logic (ILOGIC), output logic (OLOGIC), as well as input and output Serializer/Deserializer (SerDes) operation. The chapter begins with an overview of the prior work in I/O cell BIST in Section 3.2, followed in Section 3.3 by an overview of Virtex-5 I/O tiles. The overall BIST approach is described in Section 3.4, and details of the specific BIST configurations are discussed for Logic and SerDes modes in Sections 3.5 and 3.6, respectively. We present experimental results from actual implementation in Virtex-5 FPGAs in Section 3.7. Section 3.8 discusses a BIST approach for the configurable I/O buffers before the summary and conclusion in Section 3.9.

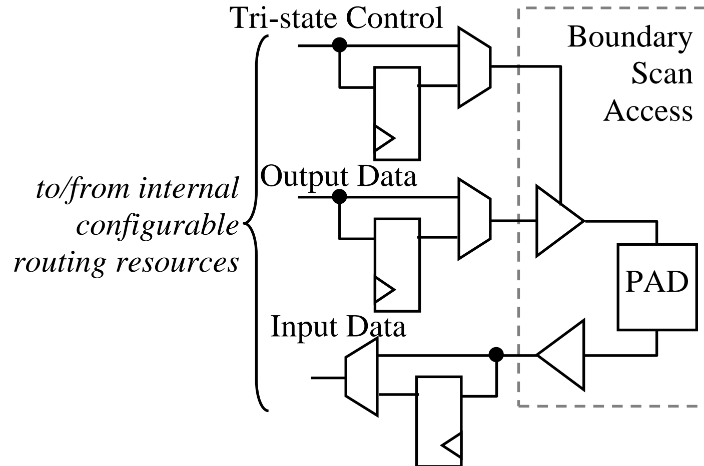


Figure 3.1: Simplified programmable I/O cell

3.2 Prior Work

There has been limited prior work in the area of testing I/O cells in, or applicable to, FPGAs [2] [3] [4] [5]. In [5], a system-level BIST architecture is presented for the I/O cells of Atmel FPGAs. The overall BIST approach was similar to that used for configurable logic resources in the FPGA core [8]. The BIST architecture in [5] consists of a single TPG implemented in configurable logic blocks (CLBs) sourcing test vectors to the I/O cells under test. A single TPG was implemented under the assumption that internal FPGA resources had already been tested and found to be fault-free. The I/O cells under test are identically configured with bidirectional I/O buffers such that the output responses are sent back into the FPGA internal resources. However, for in-system testing, this requires that all connecting devices be tri-stated during testing. The output responses of the I/O cells are monitored by CLBs configured as comparison-based output response analyzers (ORAs). While presenting a general architecture applicable to any FPGA or configurable SoC with an FPGA core and bidirectional I/O buffers, [5] implemented 27 BIST configurations applicable to the Atmel AT94K SoC and AT40K FPGA only.

3.3 Overview of Virtex-5 I/O Tiles

The I/O cells in Virtex-5 FPGAs include an output logic block (OLOGIC), input logic block (ILOGIC), I/O delay block, and a bidirectional I/O buffer, as illustrated in Figure 3.2 [7] The number of I/O cells in Virtex-5 ranges from 360 to 1,200 depending on the size of the particular FPGA.

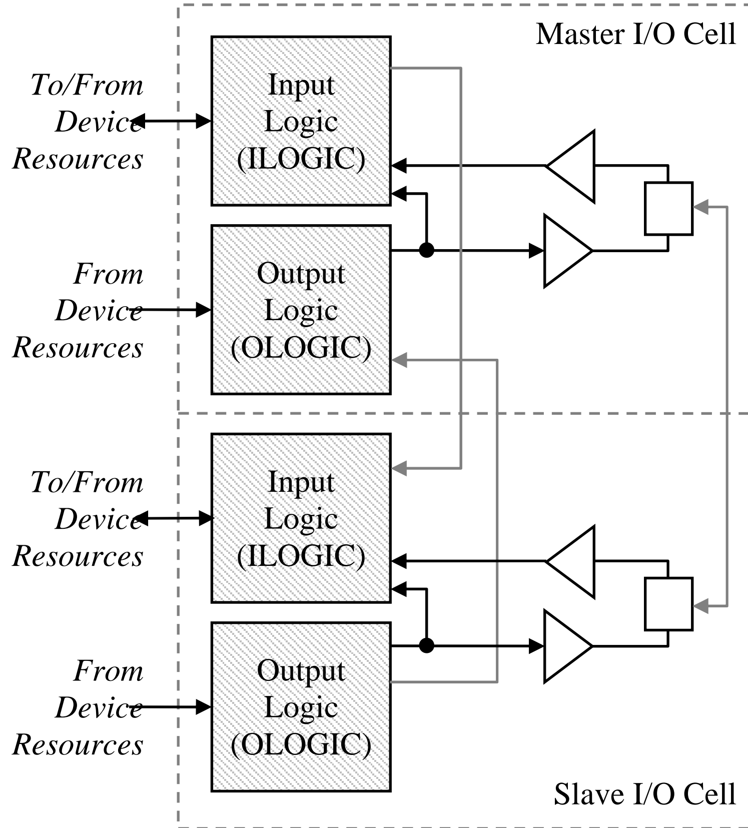


Figure 3.2: Virtex-5 programmable I/O tile

Each OLOGIC includes registers for improving system clock-to-output timing and supporting SDR and DDR transmission of data. The OLOGIC can also perform parallel-to-serial conversion of output data for widths between 2 and 6-bits when operating in SerDes mode. The ILOGIC includes registers for improving system set-up and hold times and supporting SDR and DDR reception of data. It can also perform serial-to-parallel conversion of input data for widths between 2 and 6-bits when operating in SerDes mode. The ILOGIC also incorporates a Bitslip

sub-module for synchronizing serial interfaces that include a training pattern. Invoking the Bitslip input re-orders the data on the parallel outputs of the input logic block in a barrel-shifter operation [7]. In Virtex-5 FPGAs, two I/O cells are grouped to form an I/O tile, as illustrated in Figure 3.2. Each I/O tile includes dedicated shift routing to support expanded SerDes data widths. In master/slave mode, two I/O cells in the same I/O tile are connected via the dedicated shift routing to support data widths of 7, 8 and 10-bits [7]. Each I/O cell also includes dedicated routing (also shown in Figure 3.2) directly from the OLOGIC to the ILOGIC that bypasses the I/O buffer.

3.4 Overview of BIST Architecture

Our BIST approach for I/O tiles is similar to other BIST approaches that we have developed for testing CLBs in Virtex-4 and Virtex-5 FPGAs [9]. A set of deterministic test patterns is stored in 36-kbit block random access memories (RAMs) in the FPGA fabric. The outputs of the block RAMs are connected directly to the inputs of alternating rows of I/O tiles under test. One block RAM is configured for every 5 rows of I/O tiles under test. One digital signal processor (DSP) per block RAM is configured as a counter to sequentially address the block RAM. Collectively, one 36-kbit block RAM and one DSP form the TPG for every I/O tile BIST configuration. However, the block RAM contents are modified for some configurations to target specific resources/functions under test. The advantage of configuring multiple TPGs is twofold: first, multiple TPGs reduce loading, thereby maximizing the BIST execution frequency in large devices, and, secondly, configuring multiple identical TPGs eliminates the assumption that the TPG logic resources are fault-free. Any fault affecting the behavior of a TPG will be detected by the comparison-based ORAs monitoring the I/O cells at the boundaries of any faulty and fault-free TPG.

BIST of I/O cells is well suited for circular comparison-based ORAs since many identical I/O cells are tested simultaneously. The outputs of each I/O cell under test are monitored by two ORAs and compared with the outputs of two other identically configured I/O cells in an adjacent row, as shown in Figure 3.3. To complete the circular comparison, I/O cells in the top row of the test area are compared with I/O cells under test in the bottom row of the test area.

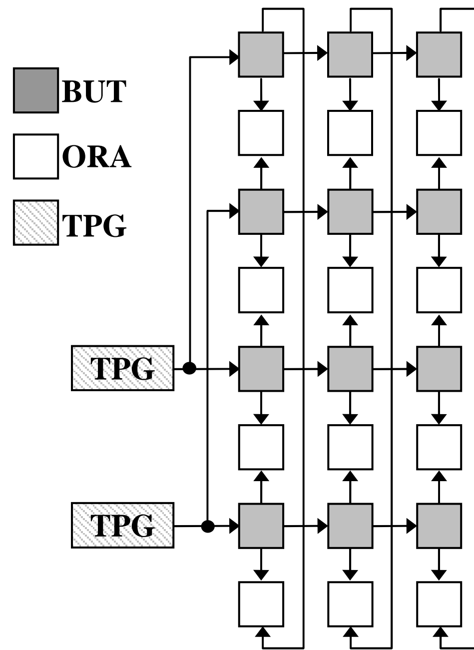


Figure 3.3: Column oriented circular comparison

The circular comparison approach does not suffer from aliasing effects as long as all of the BUTs being compared do not fail identically and at the same time. Furthermore, circular comparison improves diagnostic resolution [4]. An output response mismatch between two identically configured I/O cell outputs is latched as a logic 0 in the ORA flip-flop for the duration of the test session. Otherwise, logic 1 is retained in the ORA and is interpreted as a passing result at the conclusion of the BIST sequence. In previous implementations of the comparison-based ORA, the dedicated carry logic and routing resources in the ORA CLBs were un-used [4]. However, in all BIST configurations that we have developed for Virtex-5 FPGAs,

these resources are utilized to form an iterative-OR chain of every ORA in the test area. In each ORA, a passing result of logic 1 selects the Carry-in input to the CLB, which is the Pass/Fail result of an adjacent ORA. The carry-in input of the first MUX in the iterative-OR chain is connected to a system input, with the carry-out of the last ORA connected to a system output. If any ORA in the chain records a failure (e.g. mismatch), a logic 0 on the output of that ORA will select a logic 1 as the input to the carry MUX, as illustrated in Figure 3.4.

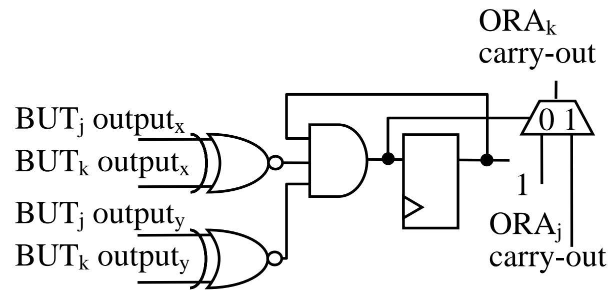


Figure 3.4: Virtex-5 equivalent ORA architecture

If no failure is observed in the ORA, the carry-in input is propagated through the CLB. If no ORAs in the iterative-OR chain observe failures, the carry-in input to the first ORA in the chain will propagate through every ORA slice to the carry-out output of the final ORA such that an overall pass/fail result is obtained without reading back the configuration memory to obtain the contents of the ORA flip-flops. By toggling the OR-chain input and observing the OR-chain output at the end of each BIST sequence, the integrity of the iterative OR-chain is verified. If the output of the iterative OR-chain indicates failures were detected, the contents of the ORAs can be retrieved via partial configuration memory readback for precise fault diagnosis.

Another important difference between our I/O tile BIST architecture and the prior work is in the configuration of the I/O tiles under test. Previous approaches have relied on bidirectional I/O buffers to provide the return path for test patterns exiting the output logic and returning to the ORAs via input logic [5] [10]. However, the reliance on bi-directionally configured I/O buffers

severely limits the applicability of this type of BIST for in-system testing. With every I/O buffer configured in the path of the logic under test, we required that all connecting devices be tri-stated during in-system testing. Connecting passive devices, such as termination resistors or light emitting diodes (LEDs), introduce another problem since these devices cannot be disconnected or tristated during in-system tests. In [9], the authors observed that, at certain frequencies, LEDs connected to I/O buffers under test caused the comparison ORAs to erroneously report failures for otherwise fault-free I/O tiles. These failures were observed at frequencies as low as 325 kHz [9], which is unacceptable for an at-speed test of the logic resources. As a result, the generality of the BIST is compromised. Fortunately, the I/O tiles in Virtex-4 and Virtex-5 FPGAs include dedicated routing from the OLOGIC to the ILOGIC that bypasses the I/O buffer [7]. Using this feedback routing instead of the I/O buffer means that no signals from the FPGA under test can reach, and therefore be influenced by, external devices. Furthermore, bypassing the I/O buffer does not sacrifice fault coverage in the I/O tile logic resources. With the I/O buffers removed from all tests for logic resources, these tests may be applied without concern for the external test environment, thus making our approach applicable to all levels of FPGA testing.

The obvious disadvantage of this approach is that it does not concurrently test the I/O buffer. However, we have developed a stand-alone BIST architecture for the I/O buffers that is applicable to device and wafer-level testing. This architecture tests the programmable analog features of the I/O buffers in every bidirectional mode of operation. Additionally, the Boundary Scan EXTEST feature may be used for in-system tests of the I/O buffers in their system mode of operation.

3.5 Configurations for I/O Logic Modes

Six test configurations are required to fully test the I/O tile logic resource in all ILOGIC/OLOGIC modes of operation. The I/O delay module is concurrently tested in these I/O Logic mode tests in two of three modes of operation. Feedback routing from the OLOGIC to the ILOGIC has two possible routes: one through the I/O delay module and one dedicated route which bypasses the I/O delay module. The route through the I/O delay module allows for testing of the output delay functionality in all supported delay modes (fixed delay, variable delay, and default). However, testing delay of input and output signals simultaneously is not possible without configuring the I/O buffers in bidirectional mode. Three of the six I/O logic BIST configurations test the DDR transmit and receive modes of operation, including, in the OLOGIC, opposite-edge, same-edge, and same-edge pipelined output modes. The fourth and fifth configurations test the flip-flop and latch functionality of the primary registers. In the sixth and final configuration, the combinatorial (un-registered) path through the I/O tile logic resources is tested. Programmable initialization values, set/reset values, and synchronous/ asynchronous reset/toggle inputs are concurrently tested. The number of clock cycles for BIST execution is 2048 for all I/O Logic BIST configurations.

3.6 Configurations for I/O SerDes Modes

A total of nine configurations are required to fully test the I/O tile logic resource in the SerDes modes of operation. Six of these configurations test the I/O SerDes logic configured for data widths of 2, 3, 4, 5, and 6-bits. Two configurations are included for the 4-bit data width to test the programmable active level on the tri-state inputs of the OLOGIC. Another three configurations test the master/slave SerDes modes for data widths of 7, 8, and 10-bits. Two of the nine configurations test the SerDes in DDR mode, with the other seven configurations testing

SDR modes of operation. SerDes operations require two clocks: a high speed clock for serial data and a divided clock for the FPGA fabric. The amount of clock division is an integer equal to the data width when testing SDR modes, and is half of the data width when testing DDR modes. We use regional clock buffers with integrated clock division, called BUFRs [7], to provide the divided clock for the ORAs and TPGs in SerDes configurations. The BUFR has programmable clock division, from 1 to 8, and BYPASS modes. There are also clear (CLR) and clock enable (CE) inputs to the BUFR. We connect the CLR and CE inputs of every BUFR to the TPGs to achieve a simultaneous test of the BUFRs and the I/O SerDes logic. Concurrent testing of the BUFRs is beneficial since they would likely be used in conjunction with SerDes. Since each BUFR clocks only one adjacent clock region, a faulty BUFR will cause failures in the ORAs along at least one boundary of an adjacent clock region. As with the I/O tiles under test, a faulty BUFR can only escape detection if every BUFR in the test area fails identically and on the same clock cycle(s).

One addition to the BIST architecture for SerDes mode testing stems from the need for synchronization of the serial bit streams before executing the BIST sequence. In SerDes mode, the positioning of deserialized data on the parallel side of the OLOGIC is initially indeterminate. Due to the nature of comparison-based ORAs, data on the parallel outputs of every I/O cell under test must be synchronized. To ensure identical alignment of deserialized test patterns, the SerDes BIST architecture adds a Bitslip synchronizer circuit, illustrated in Figure 3.5. Upon download of any SerDes mode configuration, the ORAs are held disabled and the TPGs are held in reset. A training pattern, stored in the programmable set/reset values of the block RAM output registers, is presented to the inputs of the I/O cells under test. The training pattern positions a single zero in a field of ones on the parallel side of the output logic block. The Bitslip

synchronizer circuit monitors the Q2 parallel I/O tile output and one-shots the Bitslip control line until the zero is shifted into the Q2 position. As a result of the clock division and Bitslip latency, synchronization will be obtained in no more than $4N^2-4N$ clock cycles, where N is the SerDes data width for the configuration. Each I/O cell has a dedicated Bitslip synchronizer circuit that will continue to one-shot the Bitslip control line until the training pattern is positioned with the single zero at the Q2 output, thereby identically aligning the test patterns for the comparison-based ORAs. The synchronizer is then disabled by the TPG during the BIST execution.

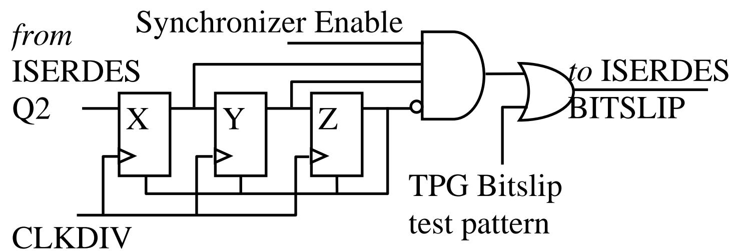


Figure 3.5: Bitslip synchronizer circuit

For SerDes configurations, the number of BIST clock cycles is equal to 1024 times the amount of clock division used during that configuration plus the worst case synchronization time for the data width being tested. It should also be noted that the number of BIST clock cycles is independent of the size of the array, and independent of the number of I/O cells under test.

3.7 Experimental Results

All of the BIST configurations are automatically generated for any size and family of Virtex-5 FPGAs by a set of ANSI C programs that we have developed. Two programs are used to generate the six configurations for the I/O logic modes of operation described in Section 3.5. Another set of two programs generates all nine of the configurations to test the I/O SerDes modes of operation described in Section 3.6. Our first program in each set generates a template BIST configuration in Xilinx Description Language (XDL) and then converts the template to

Native Circuit Description (NCD) format using Xilinx's conversion tool, *XDL.exe*. The BIST template is routed by Xilinx's place and route software, *PAR.exe*, before conversion back to XDL format. Our second program modifies the routed XDL file to produce the various BIST configurations, and converts those files back to NCD format. The final download configuration files are created using Xilinx's bitstream generation software, *BitGen.exe*.

Table 3.1 summarizes the total size of the 15 I/O BIST configuration files, the maximum BIST clock frequency, and the total number of BIST clock cycles for all Virtex-5 LXT and SXT devices. Note that the total number of BIST clock cycles is device-independent due to concurrent testing of I/O cells by the BIST architecture. The totals shown in Table 3.1 were used to calculate the best- and worst-case total test times, which are dependent on the configuration interface. The total test time for Boundary Scan and SelectMap 32-bit parallel configuration interfaces are shown in Figure 3.6 and Figure 3.7, respectively. A 50 MHz BIST clock is assumed for all configurations and all devices. Readback time is for partial configuration memory readback of the ORA contents after every configuration for diagnosis of failing BIST configurations. However, when diagnosis is not required, or there are no failures, the single bit pass/fail result can be determined via the ORA iterative-OR chain. To minimize the test time and achieve maximum fault resolution, a combination of the two methods is used. First, the pass/fail status of the BIST is determined by observing the output of the ORA iterative-OR chain. If the OR chain indicates failures, partial configuration memory readback can be used to obtain the locations of the failing ORA(s) and, thereby, determine the location(s) of the failing I/O Tile(s).

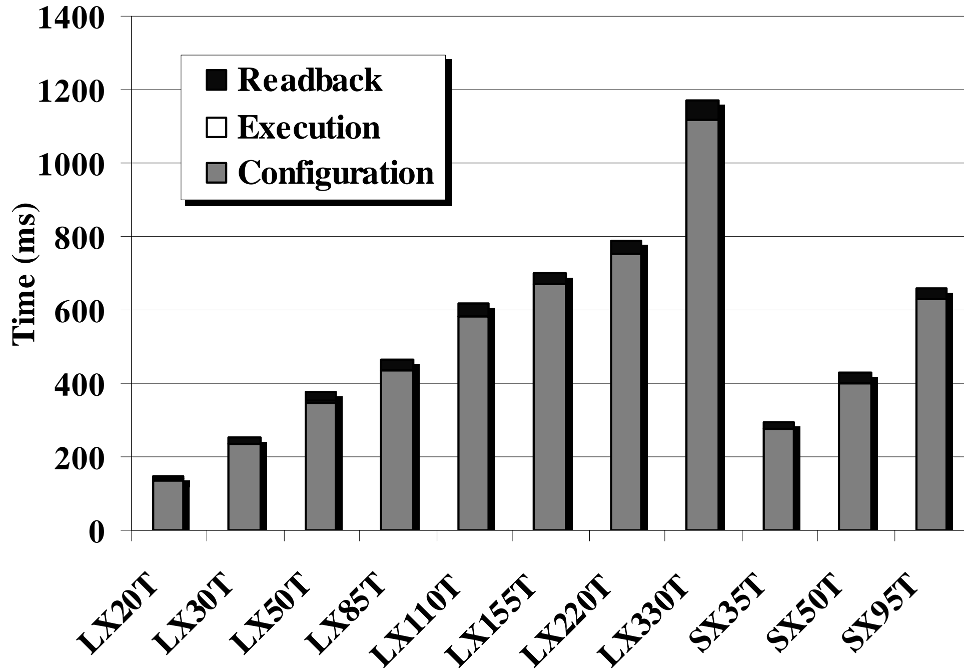


Figure 3.6: 50 MHz Boundary Scan configuration interface test time

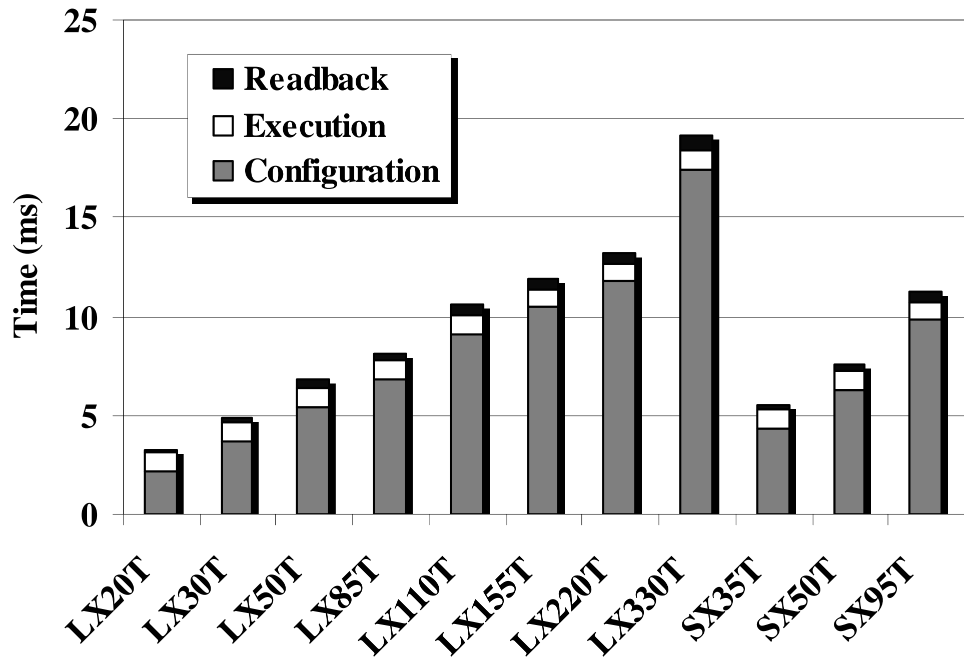


Figure 3.7: 100 MHz 32-bit parallel configuration interface test time

Table 3.1: I/O tile BIST totals (15 configurations)

Device	Total Config. Size (kB)	Max. BIST Clock Freq.	BIST CCs
LX20T	862	102.8 MHz	47112
LX30T	1482	89.38 MHz	47112
LX50T	2186	102.4 MHz	47112
LX85T	2726	73.96 MHz	47112
LX110T	3641	74.40 MHz	47112
LX155T	4181	66.10 MHz	47112
LX220T	4706	58.75 MHz	47112
LX330T	6985	56.17 MHz	47112
SX35T	1740	91.19 MHz	47112
SX50T	2511	75.17 MHz	47112
SX85T	3923	69.59 MHz	47112

3.8 BIST for Programmable I/O buffers

In addition to the BIST approach presented for I/O Logic and SerDes modes of operation, we have developed a stand-alone BIST approach for the I/O buffers in FPGAs. The approach tests the I/O buffers in all bidirectional modes of operation and associated I/O standards, requiring 77 configurations for Virtex-5 FPGAs. The approach is directly applicable to device and wafer-level testing, and is applicable to in-system testing with some customization of configurations. The bidirectional buffers configured during in-system tests can be expected to have different load characteristics in the system, depending on the way they are terminated and whether they are normally an input, output, or bidirectional port during system operation. For example, we would expect the I/O buffers that are connected to large external loads to fail if they are tested at a high frequency. For in-system testing, all of the I/O buffers can be tested at a single low frequency that is guaranteed to be sufficiently slow to allow fault-free I/O buffers to pass. However, this may result in faulty I/O buffers escaping detection in the case of delay

faults. Alternatively, the I/O buffers can be grouped together by loading characteristics to be tested independently and at different frequencies.

3.9 Conclusions

A BIST approach for testing the programmable logic resources of I/O cells in FPGAs was presented including the actual development for and implementation in Xilinx Virtex-5 FPGAs. Six BIST configurations were developed to test the input and output logic resources in ILOGIC and OLOGIC modes. Another nine configurations test the SerDes functionality of the I/O logic resources for all supported data widths. By testing the I/O buffers separately, the logic resources in the I/O tiles may be tested in-system in all modes of operation. The BIST configurations are package independent because they can test I/O tiles with both bonded and unbonded I/O buffers. This is important since FPGA synthesis tools sometimes use I/O logic and routing resources to implement the system function. All of these BIST configurations have been generated, downloaded, and verified on LX30T, LX50T, SX35T, and SX50T FPGAs. Due to similarities in architectures, features, and operational modes of the I/O cells in Xilinx Virtex-4 and Virtex-5 FPGAs, we have also applied the BIST approach described in this chapter to Virtex-4 FPGAs where a total of five I/O Logic, nine I/O SerDes, and 76 I/O buffer BIST configurations were developed, downloaded, and verified on LX60, SX35, and FX12 FPGAs. The iterative-OR ORA provides a simple interface for BIST results retrieval that is very fast relative to partial configuration memory readback and is independent of the configuration interface. However, for fault-tolerant applications, maximal diagnostic resolution of faulty I/O tiles can still be obtained via partial configuration memory readback. The BIST configurations can detect faults in the configuration memory bits associated with I/O tile logic and routing excluding the I/O buffer.

Clocking at system speeds during testing could potentially improve parametric fault coverage in the I/O delay element.

3.10 Acknowledgements

The contents of this chapter were published under the title “Built-In Self-Test of Programmable Input/Output Tiles in Virtex-5 FPGAs” in *Proceedings of the 41st IEEE Southeast Symposium on System Theory*, 2009, pp. 235-239. Prof. Charles Stroud is a co-author on the paper. Prior to publication, a preliminary version of the paper was presented at the 2008 IEEE North Atlantic Test Workshop. The proceedings of the IEEE North Atlantic Test Workshop are not published. As of this writing, a paper detailing the I/O Buffer BIST approach (describe briefly in Section 3.8) is pending publication under the title “On System-Level Use of BIST for Programmable Input/Output Buffers in FPGAs,” in *Proc. of the 2010 IEEE Southeast Regional Conference*. A majority of the actual research and the writing of the published paper presented in this chapter represents the efforts of the primary student author and not collaborators, and the research represents work performed while in the graduate program at Auburn University.

3.11 References

[1] *IEEE Standard Test Access Port and Boundary-Scan Architecture*, IEEE Std 1149.1-2001, 2001.

[2] C. Jia and L. Milor, “A BIST Solution for the Test of I/O Speed,” *Proc. IEEE Int. Test Conf.*, pp. 1023-1030, 2003.

[3] L. Zhao, D. Walker and F. Lombardi, “IDDQ Testing of Input/Output Resources of SRAM-Based FPGAs,” *Proc. Asian Test Symp.*, pp. 375-380, 1999.

[4] L-T Wang, C. Stroud, and N. Touba, *System-on-Chip Test Architectures*, Morgan Kaufmann, 2007.

[5] S. Vemula and C. Stroud, “Built-In Self-Test for Programmable I/O Buffers in FPGAs and SoCs”, *Proc. IEEE Southeastern Symp. on System Theory*, pp. 534-538, 2006.

[6] *AT94K Series Field Programmable System Level Integrated Circuit*, Data Sheet, Atmel Corp., 2001.

[7] *Virtex-5 FPGA User Guide*, UG190 (v 4.2), Xilinx Inc., San Jose, CA, May 2008.

[8] D. Milton, S. Dhingra, and C. Stroud, "Embedded Processor Based Built-In Self-Test and Diagnosis of Logic and Memory Resources in FPGAs," *Proc. Int. Conf. on Embedded Systems and Applications*, pp. 87-93, 2006.

[9] L. Lerner, S. Vemula, and C. Stroud, "System-Level BIST for Programmable I/O Buffers in FPGAs and SoCs," *Proc. IEEE North Atlantic Test Workshop*, pp. 1-9, 2006.

[10] L. Lerner, "Built-In Self-Test for Input/Output Tiles in Field Programmable Gate Arrays," M.S. thesis, Dept. of Elect. and Comput. Eng., Auburn Univ., Auburn, AL, Dec. 2007.

Chapter Four. Built-In Self-Test of SEU Detection Cores in Virtex-4 and Virtex-5 FPGAs

A Built-In Self-Test (BIST) approach is presented for the Internal Configuration Access Port (ICAP) and Frame Error Correcting Code (ECC) logic cores embedded in Xilinx Virtex-4 and Virtex-5 Field Programmable Gate Arrays (FPGAs). The Frame ECC logic facilitates the detection of Single Event Upsets (SEUs) in the FPGA configuration memory. The ICAP provides read and write access to the configuration memory from within the FPGA fabric, enabling embedded dynamic reconfiguration and fault-tolerant applications with memory scrubbing. Therefore, the fault-free operation of the ICAP and Frame ECC logic is critical for space and fault-tolerant applications that require detection and repair of SEUs. The BIST approach presented is applicable to all Virtex-4 and Virtex-5 FPGAs for both manufacturing and system-level testing of the ICAP and Frame ECC logic. The actual implementation of the BIST approach in Virtex-4 and Virtex-5 FPGAs and associated experimental results are discussed.

4.1 Introduction

The increased use of Field Programmable Gate Arrays (FPGAs) for implementing digital logic applications over the past two decades has been accompanied by increased concern about radiation effects; in particular, the effects of Single Event Upsets (SEUs). In addition to memory elements, such as flip-flops and random access memories (RAMs), the contents of the static random access memory (SRAM) used as the configuration memory to establish the overall application performed by the FPGA is also susceptible to SEUs. An SEU induced bit-flip in the SRAM configuration memory can alter the functionality of the FPGA. This makes SEUs of significantly more concern in FPGAs than in traditional application specific integrated circuits

(ASICs). Radiation experiments indicate the SEU rate in FPGAs increased by a factor of 4.74 when design rules decreased from 600nm to 350nm with a corresponding reduction in Vcc supply voltage from 5V to 3.3V [1]. Xilinx Virtex-4 FPGAs are reported to have SEU FIT (failures in 10^9 hours) rates of 246 per million bits of configuration memory, and only 151 in Virtex-5 FPGAs [2]. This reduction in SEU FIT rate from Virtex-4 to Virtex-5 indicates that Xilinx is designing FPGA configuration memories to be more robust, as suggested in [3]. However, the largest FPGAs currently have configuration memories with up to 160 million bits [4]. As a result, some recent FPGAs, like Virtex-4 and Virtex-5, have incorporated additional logic that enables the detection of SEUs in the configuration memory. This logic can be used in conjunction with user-defined circuitry in the FPGA core to correct erroneous configuration memory bits that result from SEUs [5]. Approaches for on-line SEU detection and correction for Virtex-4 FPGAs have been proposed in [5] and [6] and for Virtex-5 FPGAs in [6] and [7]. All of these approaches assume that the embedded specialized cores for SEU detection, including the Internal Configuration Access Port (ICAP) and Frame Error Correcting Code (ECC) modules, are fault-free.

This chapter presents an off-line BIST approach which completely tests the internal hardware mechanisms used for SEU detection and correction in the configuration memory of Xilinx Virtex-4 and Virtex-5 FPGAs. Since the FPGA is reconfigured for BIST only when testing is desired or required, there is no area or performance penalty incurred by the system application(s) normally executed in the FPGA. The only overhead for the BIST approach is the memory required to store one additional configuration used to configure the target device for BIST. The BIST approach is VHDL-based and is applicable to all production Virtex-4 and Virtex-5 devices. Furthermore, the BIST can be used for both manufacturing and system-level

testing of the ICAP and Frame ECC logic. The chapter begins with an overview of the ICAP and Frame ECC circuitry included in Virtex-4 and Virtex-5 FPGAs in Section 4.2. The test algorithm employed by the BIST approach to detect faults in parity-based ECC circuits is described in Section 4.3. Section 4.4 describes the method for generating and applying the test patterns to the ICAP and Frame ECC logic as well as the method used for output response analysis. Section 4.5 describes the actual implementation of the BIST approach in the fabric of Virtex-4 and Virtex-5 FPGAs along with experimental results. The chapter is summarized and concludes in Section 4.6.

4.2 Frame ECC and ICAP Logic

Like any RAM, the configuration memory of an FPGA is partitioned into words, also referred to as *frames*, which represent the smallest addressable unit of the configuration memory for write and read operations. Virtex-4 and Virtex-5 frames consist of 1,312 bits [8]-[11]. Each frame includes a 12-bit field of 11 Hamming bits and an overall parity bit for to provide the potential for single error correction (SEC) as well as double error detection (DED) in the frame data. The parity and Hamming bits are generated external to the FPGA by the configuration bitstream generation software and are subsequently downloaded with the application specific configuration data to the FPGA configuration memory. An overall cyclic redundancy check (CRC) performed on the device during the download verifies the integrity of configuration data during download. However, system memory data subject to change during the operation of the FPGA, such as contents of block RAMs and look-up tables (LUTs) used as distributed RAMs, are not covered by the overall parity and Hamming bits.

Virtex-4 and Virtex-5 FPGAs provide a specialized core, called Frame ECC, for detection and identification of single-bit errors and detection of double-bit errors in the frame

data [9][11]. The Frame ECC primitive, illustrated in Figure 4.1, has 11 syndrome outputs, an error output, and syndrome valid output. Each time that a frame is read from the configuration memory the Frame ECC module calculates the Hamming bits as well as overall parity for the frame data, and compares these bits with the Hamming bits and parity stored for that frame in the configuration memory. Based on this comparison, the Frame ECC module produces indications for no error, single-bit error, and double-bit error in addition to a syndrome indicating the location of single-bit errors. System memory element contents (for example, block RAMs, LUT RAMs, and flip-flops) are masked from the internal parity and Hamming calculation by the Frame ECC. The error codes for the Frame ECC are summarized in Table 4.1.

Table 4.1: Frame ECC codes

Error Type	**Condition (when *syndromevalid* = 1)**
No bit error	Hamming match w/ no parity error
1-bit correctable error (SEC)	Hamming mismatch w/ parity error
2-bit error detection (DED)	Hamming mismatch w/ no parity error

A Hamming mismatch with an overall parity error indicates that a single-bit correctable error has occurred. In this case, the bit-wise exclusive-OR of the stored Hamming code and the regenerated Hamming code, which is called the *syndrome*, gives the location of the single-bit error. A Hamming mismatch (non-zero syndrome) and no overall parity error indicate a non-correctable double-bit error has occurred. In the case of a double-bit error, the frame data must be repaired with data from a reliable external source. Single-bit errors in the configuration memory can be repaired with additional user logic implemented in the FPGA fabric to flip the bit in error as was done in [5], [6], and [7].

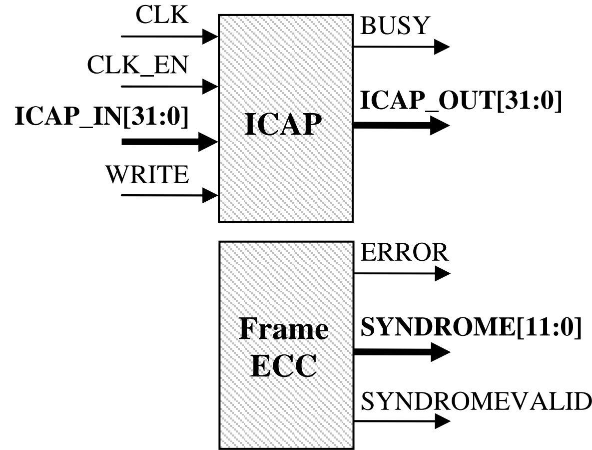


Figure 4.1: Frame ECC and ICAP primitives

The SYNDROMEVALID output is asserted for one clock cycle per frame during a frame read operation to indicate that the SYNDROME and ERROR outputs are valid for the current frame [9][11]. The most significant bit of the SYNDROME[11:0] bus is the overall parity error indication. The ERROR output is asserted when a single-bit or double-bit error is detected. To distinguish between single-bit correctable errors and double-bit non-correctable errors, the user must add logic to determine the result based on the scenarios in the last two entries in Table 4.1.

The ICAP provides access to status and control registers as well as to the configuration memory from the FPGA fabric [9][11]. The ICAP works like the external SelectMAP configuration interface except that it has separate 32-bit read and write buses, as opposed to a bidirectional 32-bit bus. The maximum operating frequency of the ICAP is 100 MHz, and it supports 8-bit, 16-bit, and 32-bit word sizes. Every device includes two ICAPs. However, both ports cannot be used simultaneously. A bit in a control register is used to select whether the upper or lower ICAP is the active port.

4.3 Test Algorithm

Hamming bits are parity calculated over a certain subset of bits in the configuration frame data. For example, the Hamming parity matrix in Table 4.2 can be extended to any number of data bits (D#) where the Hamming bits (H#) occupy the power-of-2 number locations in the counting sequence. Each Hamming bit is calculated by exclusive-ORing the data bits that have a logic 1 in the same row as that Hamming bit, yielding the logic equations shown in the lower half of the table for this example.

Table 4.2: Hamming parity matrix example

H1	**H2**	**D1**	**H3**	**D2**	**D3**	**D4**	**H4**	**D5**	**D6**	**D7**	**D8**	**D9**	**D10**	**D11**
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
$H1 = D1 \oplus D2 \oplus D4 \oplus D5 \oplus D7 \oplus D9 \oplus D11$														
$H2 = D1 \oplus D3 \oplus D4 \oplus D6 \oplus D7 \oplus D10 \oplus D11$														
$H3 = D2 \oplus D3 \oplus D4 \oplus D8 \oplus D9 \oplus D10 \oplus D11$														
$H4 = D5 \oplus D6 \oplus D7 \oplus D8 \oplus D9 \oplus D10 \oplus D11$														

As a result, the Frame ECC logic consists mainly of parity generators. A parity generator is simply an exclusive-OR tree, and can be arranged in linear tree or balanced tree forms; both arrangements are C-testable with four test patterns if *and only if* the exact parity tree construction and interconnections are known for every gate in the tree [13][14]. However, for cases where the parity tree structure is unknown, a pseudo-exhaustive test set to detect all gate level single and multiple stuck-at faults is: 1) walk a single one through a field of zeros, and 2) all combinations of two ones in a field of zeros [15]. This set of test patterns also detects all bridging faults in the Hamming generation circuit and overall parity generation circuit [16]. Therefore, the number of test vectors, N_{TV}, required in terms of the number of inputs, N, to test any parity generator (regardless of structure) is given by:

$$N_{TV} = \binom{N}{2} + N = \frac{N^2 + N}{2} \quad (4.1)$$

For the Virtex-4 and Virtex-5 Frame ECC logic, which calculates Hamming and parity over 1312-bits, the number of test patterns required by Equation 4.1 is N_{TV} = 861,328.

It is interesting to note that the parity calculations could be performed sequentially (32-bits at a time), as opposed to in parallel based on the entire 1312-bit frame. This leads to a significant reduction in the amount of logic for the calculation of Hamming code bits and overall parity. By masking appropriate bits from the parity trees (forcing bits to logic 0 using a mask LUT in conjunction with AND gates) the entire set of calculations can be performed sequentially, one 32-bit word at a time, as illustrated in Figure 4.2. The sequential Hamming generator requires twelve 32-input parity trees (one for each Hamming bit and one for the overall parity bit) with the cumulative parity calculations stored in 12 flip-flops. The Hamming and overall parity bits stored in the middle word of the frame are latched for comparison with the regenerated bits to produce the syndrome and overall parity error. This sequential parity generation would require only about 372 XOR gates and 352 AND gates for the masks. Parallel calculation over the entire 1312 frame bits, on the other hand, would require approximately 8,516 XOR gates.

It is possible that the number of test vectors for the sequential Hamming and parity bit calculation circuit might be reduced from that given by Equation 4.1. However, the set of test vectors described previously will also ensure complete testing of the word counter, masking circuit, and flip-flops/latches used to perform the sequential Hamming calculation. This means the test pattern sequence is independent of the actual architecture of the Frame ECC circuit. In addition, the walking patterns in the set of test vectors will detect stuck-at and bridging faults in the ICAP.

4.4 BIST Approach

Our approach to testing the Frame ECC logic is to implement a customized embedded core in the FPGA fabric that will repetitively write and read a single frame of configuration memory via the ICAP with the set of test patterns described in Section 4.3. The target frame for the BIST is arbitrarily located in the programmable interconnect network to avoid any configuration memory bits that are masked from the Frame ECC circuitry as a result of potentially legitimate changes to LUT-RAMs and flip-flop contents [9][11]. The basic procedure is as follows: (1) Write a configuration memory frame with a test pattern via the ICAP. (2) Read the frame containing the test pattern, compacting the ICAP output response. (3) Compact the output response of the Frame ECC when the syndrome is valid. (4) Generate the next test pattern and repeat Steps 1 through 3 for all 861,328 test vectors.

Even using the 32-bit ICAP interface, this test sequence is time-intensive because each frame write and read requires a significant amount of overhead in terms of clock cycles. In our implementation of the BIST, there are 318 clock cycles of overhead for each of the 861,328 test patterns. Therefore, the actual test time is 318 times the number of test patterns (as will be discussed in Section 4.1), or 273,902,304 clock cycles. However, the amount of logic that is tested is not insignificant, and the Frame ECC logic is critical for space and fault-tolerant applications that rely on the detection and correction of SEUs during on-line operation.

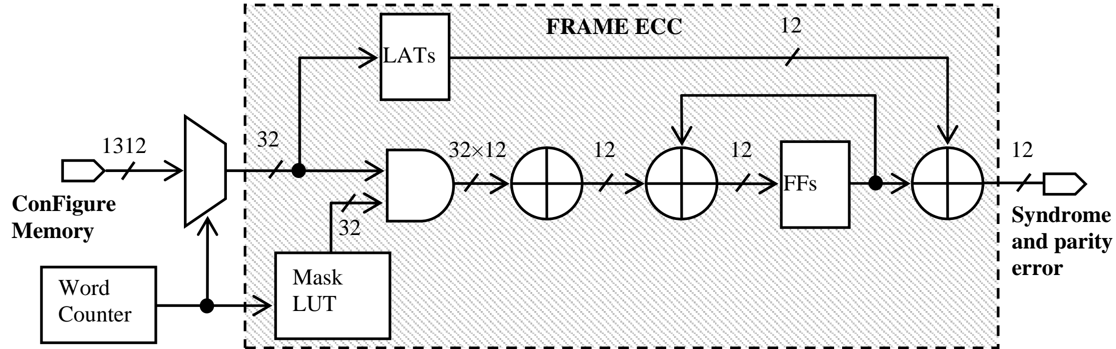


Figure 4.2: Sequential Hamming bit calculation

4.4.1 Test Pattern Generator

The test pattern generator (TPG) used to generate the parity tree test patterns is the largest component of the BIST architecture. It requires two 1,312-bit shift registers, 1,312 two-input OR gates, and a 32-bit 64-to-1 multiplexor array (the TPG is identical for both Virtex-4 and Virtex-5). In all, the TPG occupies about 1000 slices in Virtex-5 – 90% percent of all of the resources occupied by the BIST circuitry. Virtex-4 and Virtex-5 FPGAs incorporate several configuration registers to provide write/read access to the configuration memory. The Frame Address Register (FAR) stores the memory address to/from which frame data is written/read. The Frame Data Register Input (FDRI) and Frame Data Register Output (FDRO) registers facilitate input/output data to/from the configuration memory. There are other registers such as the status (STAT) register, the cyclic redundancy check (CRC) register, and the command (CMD) register which stores the next register operation to perform such as “Write FAR” or “Read FDR0”. To write/read to/from the configuration memory, a combination of these registers must be used. In Virtex-4 and Virtex-5, the frame write and read instructions for the BIST are stored in a single 512×32-bit block RAM. The complete set of write and read instructions utilize about 10% of the Block RAM. The procedure for writing/reading to/from the configuration

memory in the context of the BIST is illustrated in the pseudocode of Figure 4.3 and Figure 4.4, respectively.

```
Write_Test_Pattern (Test_Pattern, FRAME_ADDR){
     Write to Command RESET_CRC
     Write to ID Register DEVICE_ID
     Write to Command WCFG WRITE_CONFIG_MEM
     Write to Frame Address FAR FRAME_ADDR
     Write to Frame Data Input FDRI 82 words
     for(i=0; i<41; i++){
          Write word(i) of Test_Pattern
     }
     for(i=0; i<41; i++){
          Write pad word 0x00000000
     }
     Write NO-OP
     Write NO-OP
     Write to CRC 0x0000DEFC
}
```

Figure 4.3: Test pattern write sequence via ICAP interface

```
Read_Test_Pattern (FRAME_ADDR){
     Write to Command READ_CONFIG_MEM WCFG
     Write to Frame Address FAR FRAME_ADDR
     Read Frame Data Output FDRO 82 words
     for(i=0; i<41; i++){
          // Discard pad frame
     }
     for(i=0; i<41; i++){
          // Enable MISR to compact output
          // of FrameECC and ICAP
     }
     Write NO-OP
     Write NO-OP
}
```

Figure 4.4: Test pattern read sequence via ICAP interface

In both Virtex-4 and Virtex-5, the frame address selected as the write/read destination for the test patterns cannot contain LUT-RAM or flip-flop configuration bits because these bit locations are masked in the Frame ECC logic during read back (due to the fact that these bits can change after configuration if the capture command is decoded via the configuration interface or

if the capture input to the capture primitive is asserted) [9][11]. Additionally, no BIST logic or routing resources can be located in the target reconfiguration memory region. Otherwise, the test logic could overwrite and modify parts of its own architecture. To eliminate the risk of overwriting the configuration of BIST logic or routing, the target configuration memory frame is located in the routing resources in the leftmost column of I/O Tiles (however, any frame containing only routing resources and not utilized for the BIST logic could be used). In Virtex-4, the target configuration frame is arbitrarily located in the leftmost column of I/O Tiles in the 16 rows below the center line. In Virtex-5, the target configuration frame is arbitrarily located in the lower 20 rows of the leftmost column of I/O Tiles. To avoid the target frame resources, the BIST logic is physically constrained to the right half of the target device during placement and routing. Additionally, before synthesizing the BIST, the Block RAM contents may require a minor modification. The Block RAM contents are device dependent, since the correct device ID must be written to the ID register before data can be written to the configuration memory via the ICAP. This is to ensure that a configuration file formatted for one device is not written, by mistake, to the wrong device.

4.4.2 Output Response Analyzer

Since only one Frame ECC component is included in every Virtex-4 and Virtex-5 device, comparison-based output response analysis of identical blocks under test (BUT) is not possible. Furthermore, comparison with stored good circuit output responses is not practical, since the 861,328 12-bit syndromes could not be stored on the device. Instead, a 32-bit multiple input signature register (MISR) with internal feedback and primitive characteristic polynomial is employed to compact the Frame ECC output responses into a final signature. The MISR characteristic polynomial, $P(x)$, is given by:

$$P(x) = x^{32} + x^{28} + x^{27} + x + 1 \tag{4.2}$$

At the conclusion of the BIST, the signature in the MISR is compared with the known good circuit signature stored in the BIST logic, producing a single-bit pass/fail output. Additionally, the MISR is configured in a scan chain such that the signature can be retrieved via Boundary Scan for comparison with the good circuit signature. Any mismatch of the good circuit signature and the signature obtained by the BIST indicates a faulty circuit response. It should be noted that all MISRs have some probability of signature aliasing and fault escape. Signature aliasing occurs when a faulty circuit produces the same signature as the fault-free circuit. However, signature aliasing is extremely unlikely for properly designed MISRs. The classical approximation for the probability of fault aliasing is 2^{-n}, where n is the degree of the MISR's primitive polynomial [17]. Therefore, the probability of signature aliasing is approximately 1 in 4.3 billion for the 32-bit MISR described by Equation 4.2.

The ICAP is tested by adding another identical 32-bit MISR to observe the ICAP outputs during the BIST sequence. This MISR, which is enabled when the ICAP read input is asserted, will detect any stuck-at faults as well as any bridging faults in the ICAP inputs and outputs. The MISR used to detect faults in the ICAP uses a similar on-chip comparison with the known good ICAP signature to produce a pass/fail output that is logically ORed with the pass/fail output of the Frame ECC MISR and comparison circuit, as illustrated in Figure 4.5. A simultaneous test of the ICAP and Frame ECC is logical since the ICAP would almost certainly be used for any space or fault-tolerant application that actively detects and corrects SEUs. However, because each device includes two ICAPs, only one of the ICAPs may be tested per BIST configuration in our current approach. Both ICAPs can be tested by simply generating, downloading, and executing two BIST configurations that alternate between the two ICAPs. It may also be possible to modify the BIST architecture such that both ICAPs are tested during the same

configuration by using the top ICAP for the first half of the BIST sequence and switching to the bottom ICAP during the remainder of the BIST sequence, for example. This would require two additional instructions to write a logic 1 to the ICAP_SELECT bit in the control register, enabling access via the lower ICAP.

4.4.3 Additional Logic

In addition to the TPG and MISRs, the BIST architecture includes a custom soft-core embedded processor to control the BIST sequence execution. The processor is modeled in VHDL and is implemented entirely in configurable logic blocks. It controls the ICAP read/write signal and clock enable, the TPG/Block RAM multiplexor select inputs, and the TPG clock enable. The processor also includes three counters for addressing the instruction Block RAM, the TPG multiplexor, and for frame read timing. A block diagram of the ICAP and Frame ECC BIST architecture, including (from left to right) the TPG, circuits under test (CUT) and MISR output response analyzers, is shown in Figure 4.5. The input/output behavior of the architecture is discussed in Section 4.5.

4.5 Implementation Results

The entire BIST circuit is implemented in VHDL, and only one configuration download is required for the BIST application. Some minor architectural differences between Virtex-4 and Virtex-5 devices require changes to the VHDL model for the two families of devices. First, before writing to the configuration memory, a device ID check must be performed by writing the correct device ID to the IDCODE register. This prevents accidental configuration with a bitstream formatted for another device. Any attempt to write the configuration memory without a successful device ID check will cause the FPGA to attempt a fallback reconfiguration [9][11]. The device IDs are kept in a look-up table specific to Virtex-4 or Virtex-5 and are synthesized

with the design as a constant. Second, the frame address register is formatted differently for Virtex-4 and Virtex-5, requiring a modification to the stored target frame address. Finally, the input/output ordering for the ICAP in Virtex-5 is byte-swapped, compared to the Virtex-4 ICAP. Therefore, we maintain two VHDL BIST models, one for Virtex-4 and one for Virtex-5 with each model supporting all devices within that particular family.

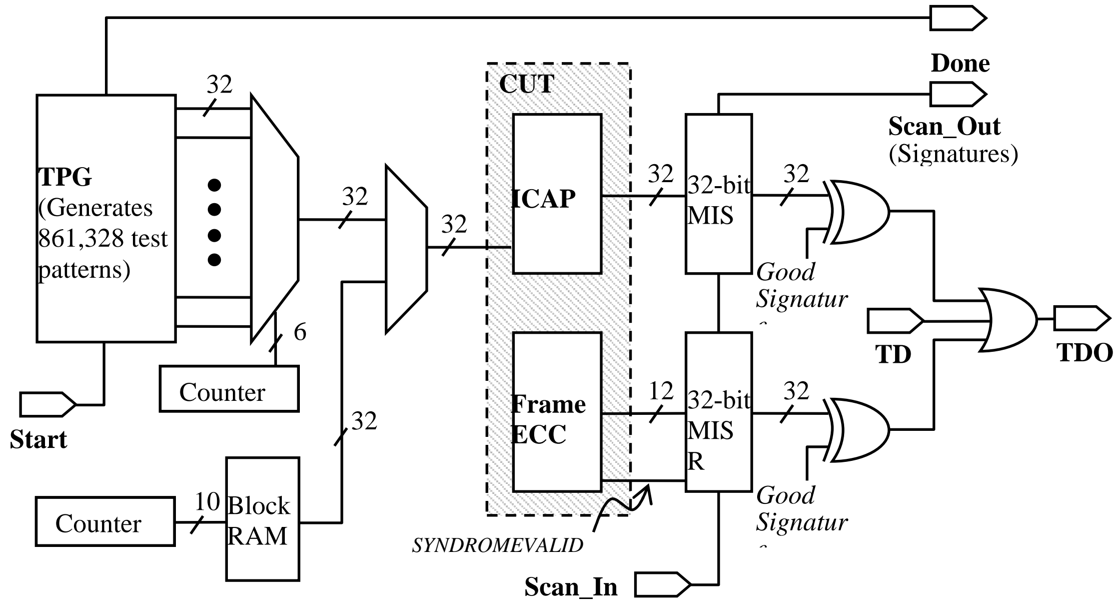


Figure 4.5: ICAP and Frame ECC BIST architecture.

There are six primary inputs and three primary outputs for the BIST architecture. The VHDL component declaration illustrating these primary inputs and outputs of the BIST configuration is given in Figure 4.6. It should be noted that the four inputs associated with the MISR scan chain, *Scan_Clock*, *Scan_Mode*, *Scan_In*, and *Scan_Out*, are included only for design verification. Therefore, only three primary inputs and two primary outputs are required for a typical application.

The *Clock* input can be a free-running system clock or can be supplied by the Boundary Scan interface via TCK (DRCK internally). The maximum BIST clock frequency when the clock is supplied externally is 100 MHz, which corresponds to the maximum ICAP clock

frequency. When the clock is supplied by Boundary Scan, the maximum BIST clock frequency is limited to 50 MHz which corresponds to the maximum TCK clock frequency. It should be noted, however, that the BIST logic in the FPGA fabric can actually operate well above the maximum configuration frequency of 100 MHz in all Virtex-4 and Virtex-5 devices based on timing analysis of the synthesized and routed design.

```
component Frame_ECC_BIST is
port(        Clock : in  std_logic;
               TDI : in std_logic;
             Start : in  std_logic;
           Scan_In : in std_logic;
         Scan_Mode : in std_logic;
        Scan_Clock : in std_logic;
               TDO : out std_logic;
              Done : out std_logic;
          Scan_Out : out std_logic);
end component Frame_ECC_BIST;
```

Figure 4.6: BIST VHDL component declaration.

The *Start* signal is an active-high, asynchronous signal which enables the execution of the BIST sequence. The *Start* signal should be asserted for a minimum of three cycles of *Clock* to begin the BIST sequence, but then may be de-asserted or may be left asserted. The BIST will start and run automatically to completion after download by tying the *Start* signal to logic 1 in the top-level VHDL model. Toggling the *Start* signal low and then high after the completion of the BIST will clear the MISRs and cause the entire BIST sequence to repeat. This feature can be used to check for reproducible BIST results during design verification. The *Scan_Mode* input places both MISRs in a scan mode. With *Scan_Mode* asserted, the *Scan_In* input is an optional input to the MISR scan chain, which can be used in conjunction with *Scan_Out* (the output of the MISR scan chain) for loading and retrieving signatures during design verification. The input *TDI* and output *TDO* provide a single-bit pass/fail result for the BIST. As illustrated in Figure

4.6, *TDI* is one input to a 3-input OR gate, with the other two inputs coming from the outputs of the MISR signature comparators. When both MISRs contain the good circuit signatures, *TDO* (the output of the OR gate) will equal *TDI*. However, if either MISR does not contain the good circuit signature, the output of the functional OR will be logic 1, regardless of the state of *TDI*. The *Done* output is asserted when the BIST sequence is complete. When the *Done* signal is asserted, the pass/fail result is valid on the *TDO* output. The BIST sequence, after download (and without tying *Start* to logic 1), is as follows: (1) Assert the Start input. (2) Wait for the Done signal to be asserted. (3) Drive TDI low, poll TDO (should be logic 0). (4) Drive TDI high, poll TDO (should be logic 1). The BIST is interpreted as passing if the TDO output presents a logic 0 in Step 3 and a logic 1 in Step 4. This ensures that the TDO output is not stuck in the fault-free state due to a fault in the FPGA. Optionally, the contents of the two 32-bit MISRs may be scanned out and verified by external comparison to the known good circuit signatures.

The total execution time for the BIST with an external 100 MHz clock is 2.739 seconds. The BIST has been downloaded, executed and verified on Virtex-4 FX12, SX35, and LX60 devices and on Virtex-5 LX30T, LX50T, SX35T, and SX50T devices using both Boundary Scan and external clock and control. Due to the differences in the configuration interfaces, Virtex-4 and Virtex-5 produce different good circuit signatures, as reflected in Table 4.3. Figures 4.7 and 4.8 show the ICAP and Frame ECC BIST implemented in the smallest Virtex-4 (FX12) and Virtex-5 (LX20T) devices, respectively. As can be seen in both figures, the BIST circuitry easily fits in programmable logic resources in the right hand half of the array. This shows that the BIST can be implemented in all other Virtex-4 and Virtex-5 devices, all of which have larger

arrays than those illustrated in Figures 4.7 and 4.8. The target configuration frame areas that should be avoided by constraining the design placement are also illustrated in the figures.

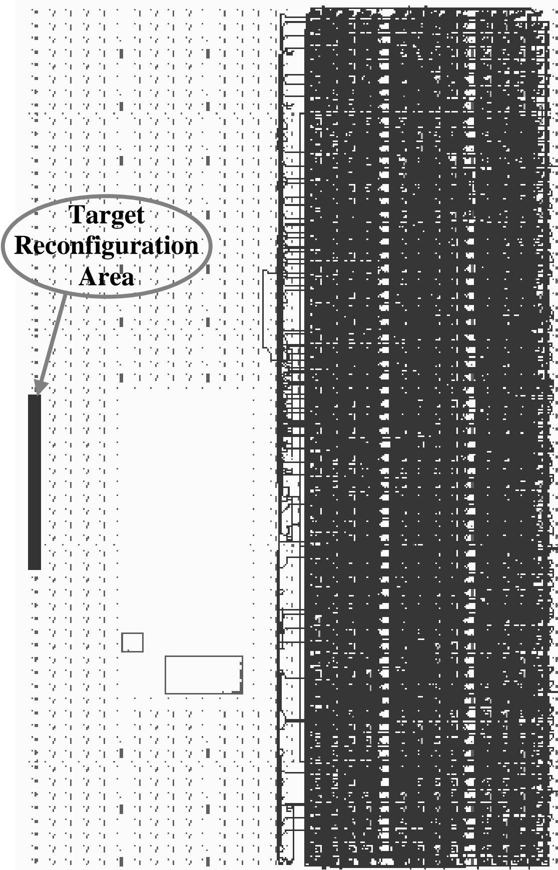


Figure 4.7: Virtex-4 FX12 with ICAP/Frame ECC BIST

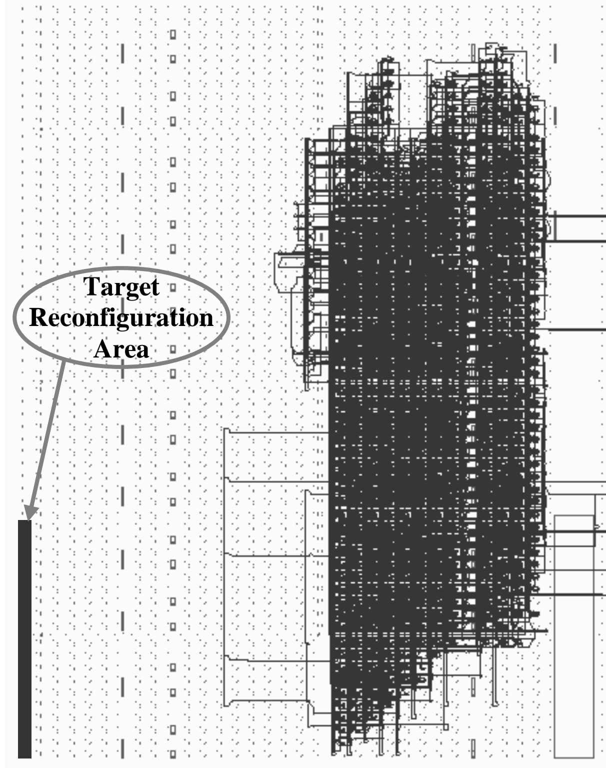


Figure 4.8: Virtex-5 LX20T with ICAP/Frame ECC BIST

Table 4.3 summarizes the actual implementation of BIST circuitry in Virtex-4 and Virtex-5 FPGAs. This includes the number of slices occupied by the BIST circuitry, the number of lines of VHDL code for the complete BIST circuit, and the total test time (excluding initial configuration time) at the maximum operating frequency of 100 MHz. The primary reason for the difference in the number of logic slices is due to the fact that Virtex-5 incorporates four 6-input LUTs and four flip-flops per slice while Virtex-4 slices incorporate only two 4-input LUTs and two flip-flops. As a result, a Virtex-5 slice has twice the logic of a Virtex-4 slice – hence, Virtex-4 requires at least twice the number of slices. The smaller LUTs in Virtex-4 account for the additional slices. The 32-bit good circuit signatures for the Frame ECC and ICAP modules are also included in Table 4.3.

Table 4.3: ICAP and Frame ECC BIST summary

	Virtex-4	**Virtex-5**
# of logic slices	2546	1010
# lines of VDHL	1125	1125
Total test time	2.739 sec.	2.739 sec.
Frame ECC signature	`0x9BC92CDB`	`0x969C47DD`
ICAP signature	`0xB3FFB18B`	`0x31D989BD`

4.6 Conclusions

This chapter has presented a BIST approach for the ICAP and Frame ECC modules in Virtex-4 and Virtex-5 FPGAs. These modules are critical components used for SEU detection and correction in the configuration memory of FPGAs for space and fault-tolerant applications. The BIST approach was developed in VHDL and is applicable to all Virtex-4 and Virtex-5 devices, and the only overhead is the memory required to store the BIST configuration and downtime for the test application. The total test time is independent of the size of the FPGA. However, when using compressed configuration bitstream files, the download time can vary with the size of the FPGA depending on the physical constraints applied during synthesis. The BIST can be periodically downloaded and executed in systems which rely on the Frame ECC and ICAP logic for on-line detection and correction of SEUs to guarantee the fault-free operation of these resources. The approach has been implemented, downloaded, and verified on a variety of Virtex-4 and Virtex-5 devices.

4.7 Acknowledgements

The contents of this chapter were published under the title "BIST of Embedded SEU Detection and Correction Cores in Virtex-4 & Virtex-5 FPGAs" in *Proceedings of the International Conference on Embedded Systems and Applications*, 2009, pp. 149-155. Prof. Charles Stroud is a co-author on the paper. A majority of the actual research and the writing of

the published paper represents the efforts of the primary student author and not collaborators, and the research represents work performed while in the graduate program at Auburn University.

4.8 References

[1] M. Ohlsson, P. Dyreklev and K. Johansson, “Neutron Single Event Upsets in SRAM-Based FPGAs,” *Proc. IEEE Nuclear and Space Radiation Effects Conf.*, pp. 177-180, 1998.

[2] A. Lesea, “Continuing Experiments of Atmospheric Neutron Effects on Deep Submicron Integrated Circuits,” WP286 (v1.0), Xilinx Inc., 2008.

[3] A. Lesea, P. Alfke, “Xilinx FPGAs Overcome the Side Effects of Sub-90 nm Technology,” WP256 (v1.0.1), Xilinx Inc., March 2007.

[4] *Virtex-6 Family Overview*, DS150 (v1.0), Xilinx Inc., 2009.

[5] L. Jones, “Single Event Upset (SEU) Detection and Correction Using Virtex-4 Devices,” Application Note XAPP714 (v 1.5), Xilinx Inc., 2007.

[6] B. Dutton and C. Stroud, “Single Event Upset Detection and Correction in Virtex-4 and Virtex-5 FPGAs,” *Proc. ISCA International Conf. on Computers and Their Applications*, pp. 57-62, 2009.

[7] K. Chapman and L. Jones, “SEU Stratagies for Virtex-5 Devices,” XAPP864 (v1.0.1), Xilinx Inc., March 2009.

[8] *Virtex-4 FPGA User Guide*, UG070 (v2.5), Xilinx Inc., 2008.

[9] *Virtex-4 FPGA Configuration User Guide*, UG071 (v1.1), Xilinx Inc., 2008.

[10] *Virtex-5 FPGA User Guide* UG190 (v4.2), Xilinx Inc., 2008.

[11] *Virtex-5 FPGA Configuration User Guide*, UG191 (v3.2), Xilinx Inc., 2008.

[12] J. Heiner, N. Collins, and M. Wirthlin, “Fault-tolerant ICAP Controller for High-Reliable Internal Scrubbing,” *IEEE Aerospace Conf.*, pp. 1-10, 2008.

[13] D. Bossen, D. Ostapko, and A. Patel, “Optimum test patterns for parity networks,” *Proc. AFIPS Fall 1970 Joint Comput. Conf.*, pp. 63-68, 1970.

[14] W-B Jone and C-J Wu, "Multiple fault detection in parity checkers," *IEEE Trans. on Computers*, vol.43, no.9, pp.1096-1099, 1994.

[15] S. Mourad and E. McCluskey, “Testability of parity checkers,” *IEEE Trans. on Industrial Electronics*, vol. 36, no. 2, pp. 254-262, 1989.

[16] L-T Wang, C. Stroud, and N. Touba, *System-on-Chip Test Architectures*, San Francisco, CA: Morgan Kaufmann, 2007.

[17] C. Stroud, *A Designer's Guide to Built-In Self-Test*, Boston, MA: Springer, 2002.

Chapter Five. Embedded Processor Based Fault Injection and SEU Emulation for FPGAs

Two embedded processor based fault injection case studies are presented which are applicable to Field Programmable Gate Arrays (FPGAs) and FPGA cores in configurable System-on-Chip (SoC) implementations. The case studies include embedded hard core and soft core processors which manipulate configuration memory bits to emulate physical and transient faults in the FPGA core including shorts and opens in programmable interconnect and many different faults in logic resources. The emulated faults are used to evaluate fault detection capabilities of Built-In Self-Test (BIST) approaches, including fault identification capabilities of diagnostic procedures, and to evaluate the effect of Single Event Upsets (SEUs), including their detection and correction. Embedded processor based approaches provide significant improvement over previous fault injection techniques and, in turn, enable a more thorough analysis of BIST, diagnosis, and SEU mitigation.

5.1 Introduction and Background

There are a number of Field Programmable Gate Array (FPGA) applications that can make use of the presence of physical faults. These applications include Built-In Self-Test (BIST) of the FPGA itself [1], some fault-tolerant design techniques [2], and Single Event Upset (SEU) detection/correction techniques for FPGA configuration memories [3]. These applications target FPGA devices as well as FPGA cores in configurable System-on-Chip (SoC) implementations. Verification, analysis, and evaluation of these applications can be performed with the ability to inject or emulate physical faults in the FPGA.

It is difficult to find actual faulty devices and their usefulness is limited due to the fixed nature of the fault [1]. Physical faults can be created by etching the packaged device and creating opens in routing resources that lie at the top level of interconnect metal for example, but once again the usefulness of these devices is limited. A more efficient approach is to manipulate the configuration memory bits to emulate physical faults in the device [4]. For example, a stuck-at fault in a look-up table (LUT) bit can be emulated by overwriting the particular configuration memory bit and setting it to the desired stuck-at fault value. SEUs on the other hand can be emulated by flipping the value of bits in the configuration memory. Shorts and opens in the interconnect network can be emulated along with almost any fault in the logic resources that can be controlled by configuration memory bits. When downloading the intended system configuration, the faults to be emulated can be injected in the configuration data just prior to the actual download process [1]. Alternatively, the intended configuration can be downloaded with subsequent partial reconfiguration used to inject and emulate the fault.

One of the first FPGA applications to use fault injection emulation was hardware acceleration techniques for fault simulation [4]. However, the download time for fault injection detracted from the hardware acceleration to the extent that the manipulation of configuration bits was abandoned and replaced by fault emulation circuitry that was modeled and downloaded with the circuit to be simulated [5][6]. The overhead of the additional fault emulation circuitry and its associated routing was significant but acceptable in the case of fault simulation [7]. The additional circuitry and routing was not acceptable in the case of BIST approaches since the goal was to maximize the resources under test in any given configuration such that there are no remaining resources available to emulate faults. As a result, fault injection via configuration memory bit manipulation has been used extensively to debug, verify, and analyze development

of BIST configurations and diagnostic procedures for FPGAs [1][8]. Similarly, analysis of the affects of SEUs [3] as well as SEU detection and correction in FPGA configuration memories [9] can use manipulation of configuration memory bits and has been shown to be effective in emulating 97% of the SEUs induced and observed in radiation chamber experiments [3].

In this chapter, we present two case studies of embedded processors used to manipulate FPGA configuration memory bits for FPGA BIST and SEU detection/correction applications. The first case study uses a hard core embedded processor that has dedicated program and data memories with write access to the configuration memory of an FPGA core in a configurable SoC. In this case study, described in Section 5.2, the device is the Atmel AT9K series Field Programmable System Level Integrated Circuit (FPSLIC). The second case study uses a soft core embedded processor in an FPGA for manipulation of configuration memory bits via an internal configuration access port (ICAP). The soft core processor is downloaded with the application to be injected with faults. In this case study, described in Section 5.3, the devices include Xilinx Virtex-4 and Virtex-5 FPGAs. Each case study includes an overview of the device architectures, description of the fault injection emulation technique, and experimental results of the actual implementation. The chapter is summarized and concludes in Section 5.4.

5.2 Hard Core Processor Case Study

The Atmel AT94K series configurable SoC consists of an FPGA core, various RAM cores, and an 8-bit Advanced Virtual RISC (AVR) microcontroller core as shown in Figure 5.1 [10]. Three types of memory resources include [10]: 1) many small 32×4-bit RAMs distributed throughout the FPGA core, 2) a 4-Kbyte to 16-Kbyte dual-port data RAM shared by AVR microcontroller and the FPGA core, and 3) a 20-Kbyte to 32-Kbyte program memory accessible only by the AVR microcontroller and used for storing machine code.

The AVR core is an 8-bit RISC architecture with 32 general purpose registers including a number of peripherals like watchdog timer, UART, etc [10]. There are two 8-bit bi-directional general purpose I/O ports. An 8-bit bi-directional data bus between the FPGA and AVR (controlled by the AVR) provides communications between the two cores. Whenever 8-bit data is written to (or read from) the data bus by the AVR, a strobe signal to the FPGA core is generated on FPGAIOWE (or FPGAIORE) along with one of 16 decoded select lines to the FPGA. There are four external interrupts to the AVR along with 16 interrupts from the FPGA.

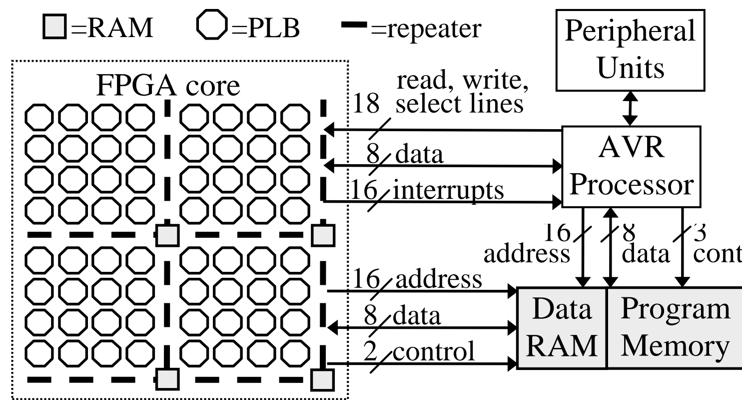


Figure 5.1: AT94K series SoC architecture

The FPGA core is constructed as a symmetrical $N \times N$ array of programmable logic blocks (PLBs), where N=48 for the AT94K40 device (the largest AT94K series SoC) [10]. Each PLB contains two 3-input LUTs, a D flip-flop, and additional multiplexers/gates. Every PLB has dedicated diagonal (X) and orthogonal (Y) local routing resources to its neighboring PLBs, as shown in Figure 5.2a [10]. As shown in Figure 5.2b, the vertical and horizontal global routing resources associated with each PLB traverse a total of four PLBs (×4 lines) and eight PLBs (×8 lines). Vertical and horizontal bus repeaters are placed at the boundaries of every 4×4 array of PLBs (shown in Figure 5.2c for the horizontal bus) to prevent signal degradation in lengthy and/or heavily loaded signal nets. The repeaters also facilitate connections between ×4 and ×8 lines as seen in Figure 5.2d.

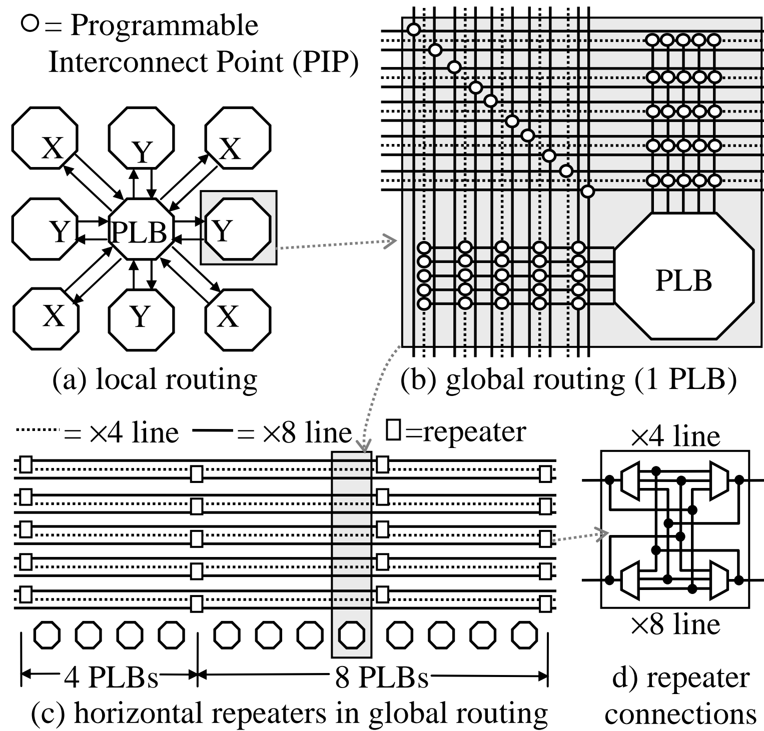


Figure 5.2: AT94K routing architecture

The AVR microcontroller core can write to (but not read from) the FPGA core configuration memory such that the FPGA can be dynamically reconfigured (either fully or partially) by the AVR core during normal system operation [10]. The FPGA configuration memory access is via a 24-bit address bus and 8-bit data bus. The address bus is partitioned into three 8-bit components referred to as FPGAX, FPGAY, and FPGAZ. FPGAX and FPGAY correspond to horizontal and vertical location of the programmable resource in the array while FPGAZ corresponds to specific logic/routing resources within the specified programmable resource. A write to the 8-bit data bus, FPGAD, results in a write cycle to a byte of the FPGA configuration memory.

Sets of BIST configurations were developed to test the various programmable resources in the FPGA core including PLBs, RAMs, and the programmable interconnect network with horizontal and vertical repeaters [11]. During the verification and analysis of the sets of BIST

configurations, every configuration bit associated with the specified resource under test was injected in turn with a stuck-at-0 fault and a stuck-at-1 fault. For each fault injected, the BIST configurations that target that resource were applied (with the injected fault present). The BIST results indicate which BIST configurations, if any, detected the emulated fault. Because of the large number of faults to be emulated (twice the number of configuration bits) for each BIST configuration, injecting the faults in the configuration download file prior to each download takes considerable time as indicated by the "download run time" in Table 5.1. Note that bank clock and set/reset lines are associated with the vertical repeaters, hence, the larger number of configuration bits when compared to the horizontal repeaters and associated routing.

Table 5.1: Embedded fault injection run time analysis for AT94K40

Resource	**BIST Configs**	**Config Bits**	**Total Faults**	**Download Run Time**	**Processor Run Time**
PLB with flip-flops	8	81	162	4 hr 29 min	4 min 34 sec
Vertical Repeaters	20	71	142	3 hr 55 min	4 min 1 sec
Horizontal Repeaters	20	65	130	3 hr 36 min	3 min 40 sec
Free RAM	3	4	8	13 min	14 sec

BIST configurations can also be generated and executed by the embedded AVR processor [11]. In this case, fault injection emulation is somewhat more difficult since the processor core has write-only access to the FPGA configuration memory. If the processor core could also read the configuration memory, it could perform a read-modify-write (RMW) operation to inject a fault at any desired configuration memory bit. With write-only access, one must also know the normal BIST configuration data for each configuration memory byte in order to inject a single fault without disturbing the other seven bits of configuration data; otherwise, we could be injecting eight faults at a time. When the embedded processor is generating the BIST

configuration, the information is contained within that resident program. As a result, the fault injection emulation can more realistically be performed from the embedded processor, although the development effort is greater without the RMW capability. Table 5.1 gives the run time when using the embedded processor core to perform fault injection emulation along with the BIST configuration generation and execution. A speed-up of almost a factor of 60 is obtained when the embedded processor core performs the fault injection emulation analysis including BIST configuration generation, BIST sequence execution, and BIST results retrieval.

5.3 Soft Core Processor Case Study

The configuration memories of Virtex-4 [12] and Virtex-5 [13] FPGAs are partitioned into frames, where each frame has a fixed length of 1,312 bits, or forty-one 32-bit words. A frame is the smallest addressable segment of the configuration memory; therefore all memory write/read operations must be performed on whole frames. In Virtex-4 devices, a frame contains the configuration data for 16 rows of configurable logic blocks (CLBs) and input/output (I/O) tiles, or four rows of block random access memories (RAMs) and digital signal processors (DSPs) tiles in the same column [12]. In Virtex-5 devices, a frame covers 20 rows of CLBs and I/O tiles or five rows of block RAMs and DSPs tiles [13]. This means that individual FPGA resources cannot be reconfigured without also providing explicit configuration data for other FPGA resources that occupy the same frame.

Virtex-4 and Virtex-5 FPGAs incorporate several configuration registers to provide write/read access to the configuration memory. The Frame Address Register (FAR) stores the memory address to/from which frame data is written/read. The Frame Data Register Input (FDRI) and Frame Data Register Output (FDRO) registers facilitate input/output data to/from the configuration memory. There are other registers such as the status (STAT) register, the cyclic

redundancy check (CRC) register, and the command (CMD) register which stores the next register operation to perform such as “Write FAR” or “Read FDR0”. To write/read to/from the configuration memory, a combination of these registers must be used. These registers are accessible from both Boundary Scan and SelectMAP configuration interfaces as well as the internal configuration access port (ICAP) located in, and accessible from, the FPGA fabric.

Emulated SEUs, or faults injected for BIST, require the reconfiguration of a single configuration memory bit after system configuration, or each BIST configuration, is downloaded. Furthermore, the contents of the frame, which configure multiple rows of resources, must be preserved during reconfiguration for emulated SEU/fault injection. Our approach takes advantage of partial reconfiguration and read back capabilities of Virtex-4 and Virtex-5 FPGAs to implement RMW for bit-level partial reconfiguration.

5.3.1 Overview of Approach

The basic approach begins with locating the frame containing the target bit for fault or SEU emulation. The frame is read in its entirety and stored. Next, the target bit is located within the frame, and overwritten with the desired stuck-at value in the case of a fault. This approach also supports emulation of SEUs by simply inverting the target bit. Finally, the modified frame is written back to the same location in the configuration memory from which it was read. Optionally, a subsequent read back of the frame can be used to verify the frame RMW results. The frame address and index of the bit targeted for fault/SEU emulation are stored in a list of faults/SEUs to be emulated. For each fault in the list, the BIST configuration is downloaded, executed with the fault on the device, and the results retrieved. If any of the output response analyzers (ORAs) record a failure, indicating a faulty block under test (BUT), the fault has been detected [9]. However, most tests of a specific FPGA resource require multiple BIST

configurations to test its programmability and achieve high fault coverage. Given N BIST configurations and M faults in the fault list, the total number of downloads, executions, and retrievals of BIST results is $N \times M$. The main reason why this many downloads are required is that there is no way to reset the ORAs once a fault is detected such that failures are latched until a new configuration is downloaded. Partial reconfiguration can be used to reduce download time, but it does not reset the ORAs between two consecutive BIST configurations. Therefore, once a fault is detected, the ORAs return failure indications for the remaining BIST configurations that may not detect the fault. Even though ORA failure indications imply a fault was detected, it is not clear which configuration detected the fault for proper evaluation.

Since the BIST approach pseudo-exhaustively tests multiple identically configured BUTs, the fault coverage in one BUT may be assumed to be the overall fault coverage for all BUTs. This assumption greatly reduces the number of faults, M, that need to be emulated to obtain accurate fault coverage. For example, consider Figure 5.3, which shows the simulated individual and cumulative single stuck-at fault coverage for our BIST configurations for Virtex-5 CLBs in SliceL mode of operation. The simulation results are based on gate-level models of the CLB. The simulation results show that six BIST configurations are required to cumulatively detect 100% of single stuck-at faults in the CLB in SliceL mode of operation. However, as discussed in [14], the SliceL configurations must be applied twice such that every CLB serves both as a BUT and an ORA.

A total of 3,006 collapsed stuck-at faults were found for the SliceL and another 8,462 faults for SliceM, all of which were cumulatively detected in fault simulation. These comprehensive fault lists include all faults affecting the CLB, including configuration memory bit stuck-at faults. Therefore, by using fault injection to emulate a subset of the complete fault

list (specifically, those faults affecting the configuration memory bits), both the quality of the BIST configurations and the accuracy of the gate-level fault simulation models can be gauged. Less than 100% fault coverage from fault injection would suggest inaccuracies in the simulation model and potentially lower fault coverage than the fault simulations suggest. Of the 3,006 faults in the SliceL, 614 represent configuration memory bit stuck-at faults. These faults were emulated using the RMW approach previously described, with results shown in Figure 5.4. Using fault injection, 100% of the configuration memory bit faults affecting the SliceL mode of operation were detected, confirming the simulation results in Figure 5.3. Furthermore, the similarity of the fault coverage trends in Figures 5.3 and 5.4 helps to verify the accuracy of simulation models.

The biggest drawback of prior fault injection approaches is the large number ($N \times M$) of downloads required to emulate a sufficient sample of configuration memory bit faults. To obtain the results shown in Figure 5.4, a total of 614×6 = 3,684 downloads, fault injections, BIST executions, and results retrievals were required. Additionally, any revision to a BIST configuration requires the complete fault list be run again to ensure that the modified configuration does not jeopardize fault detection capabilities. The total time required for fault injection can be calculated by multiplying the test time for the set of BIST configurations by the number of faults in the fault list. Figure 5.5 shows the total test time for the set of all CLB BIST configurations using compressed downloads via a 50MHz Boundary Scan interface. Consider the set of CLB BIST configurations for the mid-sized LX50T, which requires 3,147 ms using the 50 MHz Boundary Scan interface from Figure 5.5. For the complete list of 698 configuration memory bit faults (which includes SliceM mode configuration bits), the fault injection time is 698×3.147 = 2,197 seconds. The more realistic fault injection time that we experienced, using a

333 kHz PC parallel port interface to Boundary Scan, was approximately 150×2,197 = 81,666 seconds, or 91.53 hours. This lengthy application time prompted us to develop the embedded soft core processor based fault injection approach which greatly improves the test time by both increasing the achievable configuration interface frequency and by increasing the configuration interface word size using the ICAP.

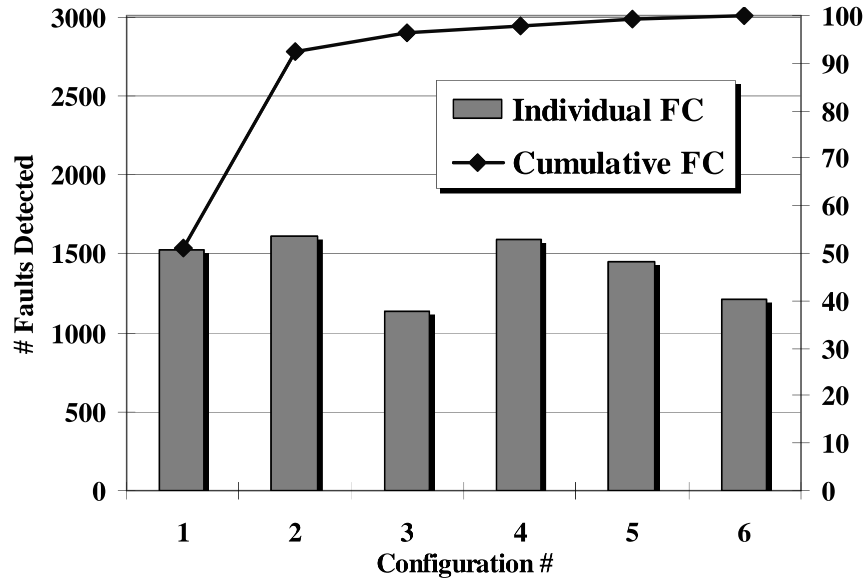


Figure 5.3: SliceL simulation stuck-at fault coverage

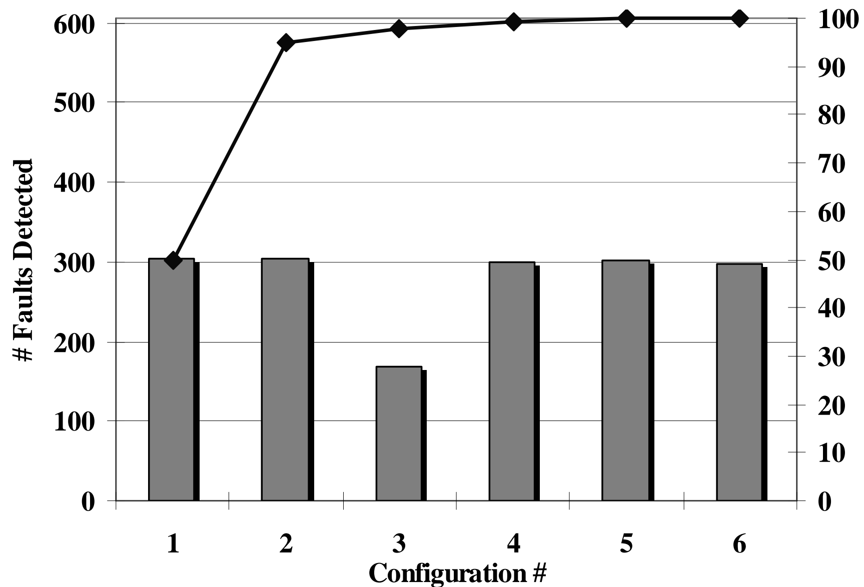


Figure 5.4: SliceL fault injection stuck-at fault coverage

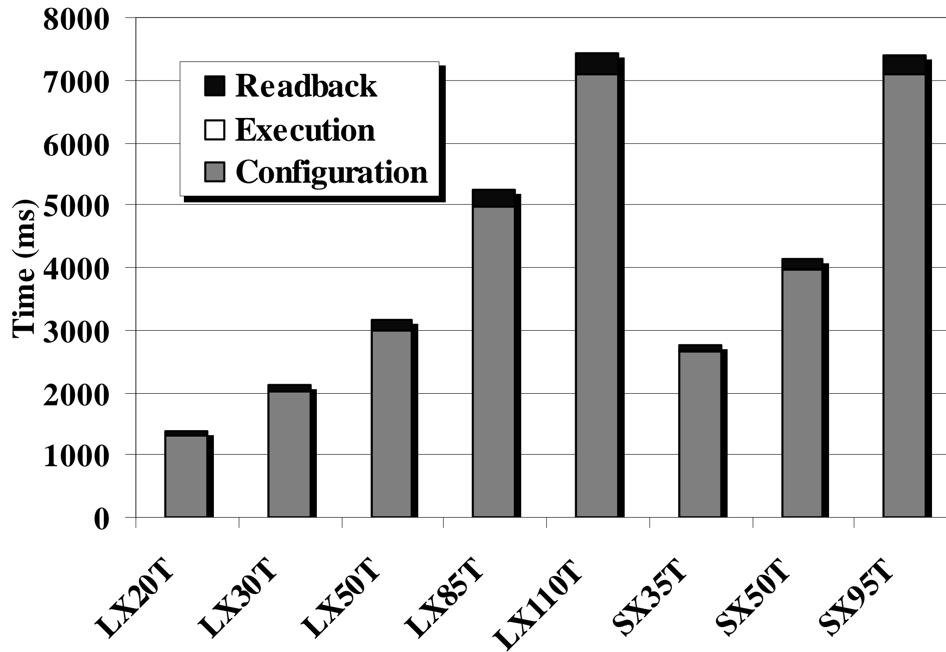


Figure 5.5: Total CLB test time via Boundary Scan

The ICAP provides access to configuration registers and the configuration memory internally from the FPGA fabric. The ICAP works like the external SelectMAP interface except that it has separate 32-bit write and read buses, as opposed to a bidirectional 32-bit bus. The maximum operating frequency of the ICAP is 100 MHz, and it supports 8-bit, 16-bit, and 32-bit word sizes [12][13]. Every device includes two ICAPs; however, both ports can not be used simultaneously. A configuration bit in the configuration interface control register selects between the upper and lower ICAPs. The basic idea of an embedded fault/SEU emulation approach is to embed all of the logic required for frame RMW operations in the FPGA with the BIST or SEU controller configuration, using the ICAP to access the configuration memory. The benefit of embedded fault/SEU emulation approach is a minimum 32 times speed up over the external Boundary Scan configuration interface operating at the same frequency. In addition, configuration frequencies of 100 MHz are achievable within the FPGA fabric.

5.3.2 Architecture and Operation

In our embedded fault/SEU emulation approach, a configuration containing both the BIST and SEU controller architecture and some additional logic is downloaded to the device. A list of fault/SEU sites (configuration memory address and bit indexes) is loaded into the embedded fault/SEU emulation logic in the FPGA either with the download or via an external interface after download. The embedded system proceeds by reading the configuration frame containing the first fault/SEU site. The frame is temporarily stored in the FPGA fabric while the target bit is located and the fault/SEU injected. Next, the frame is written back into the configuration memory and the BIST is allowed to execute as normal. When the BIST has run to completion, a single-bit pass/fail result for the configuration is stored. Normally, using the external interface, the BIST would proceed to the next configuration. However, the embedded logic can correct the previously injected fault, reset the ORAs, and then inject the next fault in the fault list, as can be seen in the flowchart in Figure 5.6. This approach has been implemented in Virtex-4 and Virtex-5 FPGAs. The implementation is discussed in the remainder of this section.

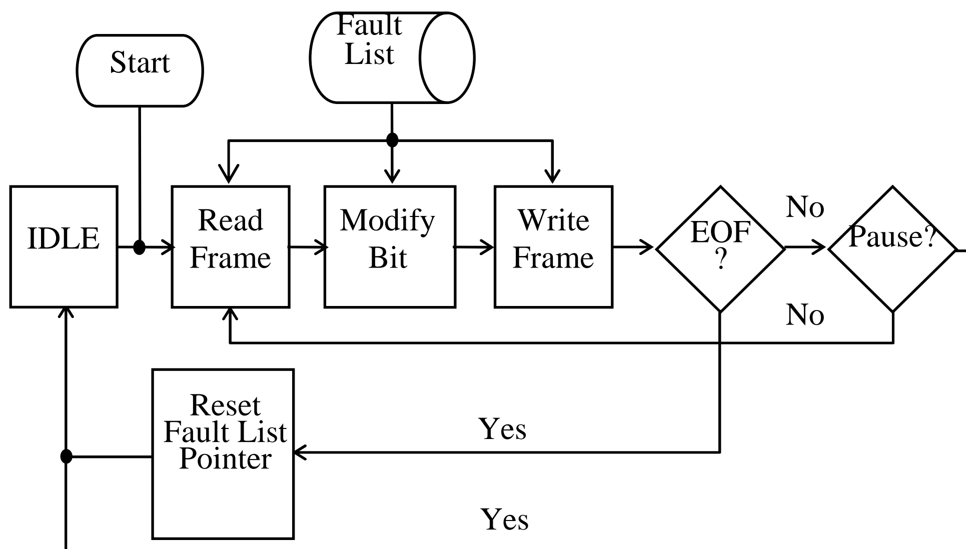


Figure 5.6: Frame read-modify-write flowchart

The embedded fault/SEU emulation core is entirely implemented in CLBs and two block RAMs in the FPGA fabric. A central component of the architecture is the dual port 18-kbit block RAM. Block RAMs have two independently configurable read and write ports (A port and B port); only the stored data is shared [12][13]. One block RAM is used to temporarily store frames during the RMW procedure. To accomplish the RMW, the B port is configured for 32-bit reads/writes and the B port input data bus is connected directly to the ICAP 32-bit data output bus. The B port data output bus is connected to the ICAP inputs via a 32-bit 2-to-1 multiplexor. A frame read is initiated at the configuration memory frame address specified by the current fault and as the frame is read it is stored in the first forty-one 32-bit words in the block RAM. Next, the A port, configured for 1-bit read/write operations, is used to locate the target bit in the location specified by the fault list entry. In the case of a stuck-at 1/stuck-at 0 fault, a 1/0 is written at the specified bit. However, for SEU emulation, the contents of the specified bit address are read, inverted, and then written back to the same address. Finally, the modified frame is written back to the same address from which it was read via the 32-bit B port output data bus.

The fault list is stored in a second dual-port 18-kbit block RAM. The block RAM is configured with independent 512×36-bit read and write ports. The write port is connected to a Boundary Scan user access register with some additional logic for controlling the address bus; namely, a 32-bit shift register and address counter. The read port output bus of the block RAM is connected to the embedded fault/SEU injection logic and state machine. This block RAM structure allows a fault list to be written into the block RAM after the device is configured, and the list is immediately accessible by the fault/SEU injection logic and state-machine. However, the block RAM contents can also be initialized with a fault list in the VHDL model, eliminating

the need to shift in the fault list via the Boundary Scan user access register. The block RAM is capable of storing up to 512 faults.

The core must be capable of facilitating any length fault list up to the maximum of 512 faults. Therefore, an end-of-file delimiter is required. Each 32-bit word in the block RAM has four parity bits which we use to store the file delimiters as well as control bits for stuck-at faults and bit-flips (SEU emulation). The ability to inject multiple faults simultaneously is also desirable. This requires the inclusion of a ‘pause’ delimiter in addition to the ‘end-of-file’ delimiter. Our solution is to use the two least significant bits of the parity word to encode the fault type (stuck-at 1, stuck-at 0, or bit-flip) and to use the two most significant parity bits to store delimiters. The encoding scheme for these bits is shown in Table 5.2, and the overall fault list format for the 32-bit data word and 4-bit parity word is shown in Table 5.3.

Table 5.2: Parity bit encoding, where X = don’t care

Parity[3:2]	**Description**	**Parity[1:0]**	**Description**
00	Continue to next fault	00	Stuck-at zero
01	Pause at fault	01	Stuck-at one
1X	End-of-file (EOF)	1X	Bit-flip (SEU)

Table 5.3: Embedded fault list format

35:34	**33:32**	**32:21**	**20:0**
Delimiters	Fault Code	Bit Index	Frame Address

The other significant component of the architecture is a 40×256-bit ROM implemented in LUTs in the FPGA fabric. This ROM is used to store all 32-bit ICAP instructions required for the frame RMW process. Another eight control bits control the ICAP write and clock enable inputs, and serve as inputs to the state machine logic. Instructions are stored in the ROM in the order in which they are written to the block RAM such that the block RAM may be sequentially

addressed to initiate new frame reads and writes. The two block RAMs, instruction ROM, and ICAP are connected by an assortment of glue logic, including the large 32-bit 2-to-1 multiplexor. A block diagram of the overall embedded fault/SEU injection core appears in Figure 5.7.

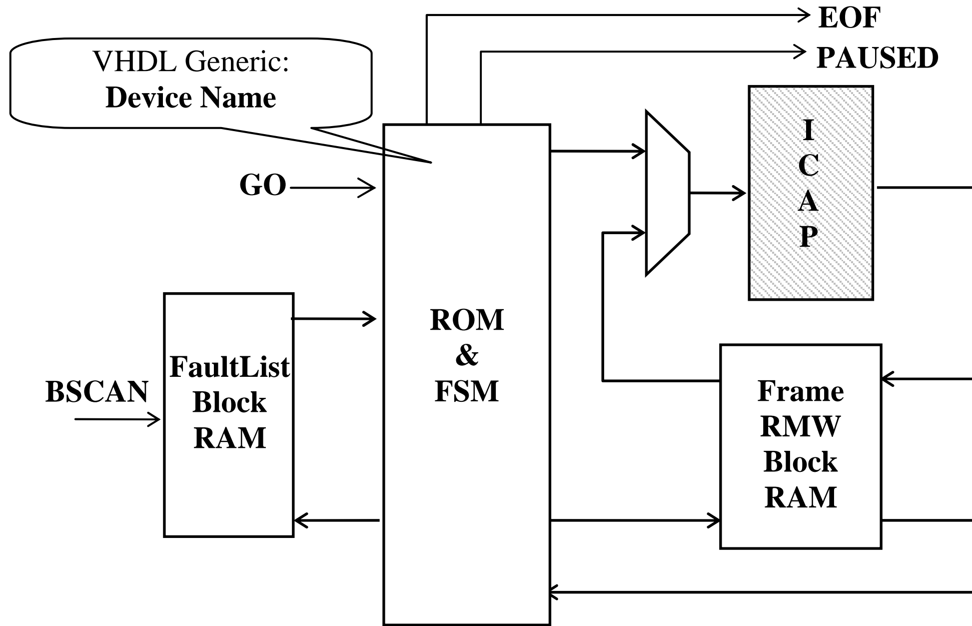


Figure 5.7: Block diagram of fault injection core

5.3.3 Implementation Results

The total number of slices used in Virtex-4 and Virtex-5 FPGAs is shown in Table 5.4. The primary reason for the difference in the number of logic slices is due to the fact that Virtex-5 incorporates four 6-input LUTs and four flip-flops per slice while Virtex-4 slices incorporate only two 4-input LUTs and two flip-flops. As a result, a Virtex-5 slice has twice the logic of a Virtex-4 slice – hence, Virtex-4 requires at least twice the number of slices. The smaller LUTs in Virtex-4 account for the additional slices.

Table 5.4: Embedded fault injection core resources

Attribute	Virtex-4	Virtex-5
# lines of VHDL	~950	~950
# block RAMs	2	2
# slices	228	67

The entire embedded fault/SEU emulation core is modeled in VHDL. For VHDL-based designs to be faulted, the fault/SEU emulation core may be instantiated in the top level of the design and synthesized with the intended system function to be faulted. Our BIST configurations are not modeled in VHDL, and in this case the fault injection core is added later in the design flow. Because our BIST configurations are modeled in Xilinx Design Language (XDL), the fault/SEU emulation core is synthesized and converted to XDL. The XDL of the embedded core and the BIST can then be combined and the design flow continued. In either case, it will be necessary to constrain the placement of the design to an area of the FPGA not targeted for fault injection. For example, if the fault injection core is embedded with a block RAM BIST configuration [15], the two fault injection core block RAMs must be constrained to an area of the device away from the BIST configuration. Furthermore, the fault list must not contain the address of fault sites located in the embedded fault/SEU emulation core's block RAMs. If any configuration memory frame addresses in the fault list happen to correspond with any of the embedded core's resources, the core could overwrite a bit controlling the functionality of its own resources, resulting in likely failure. An example of a properly constrained design is shown in Figure 5.8. In the figure, a partial array of test pattern generators ORAs and CLBs under test is placed in the left half of the device with the embedded fault injection core is constrained to the right half of the device. The embedded fault injection core is loaded with fault addresses residing only in the left half of the array.

The component declaration for the embedded fault/SEU injection core is shown in Figure 5.9. There are two primary inputs and two primary outputs for the model, as well as a generic which specifies the device. It should be noted that the Boundary Scan access to the fault list block RAM is embedded in the VHDL model, so these I/O do not appear in the top level

component declaration. While the top level component declaration is identical for Virtex-4 and Virtex-5, we maintain separate VHDL models for Virtex-4 and Virtex-5 because of some minor architectural differences between the device families. First, before writing to the configuration memory, a device ID check must be performed by writing the correct device ID to the IDCODE register. (This prevents accidental configuration with a bitstream formatted for another device.) The device IDs are kept in a LUT specific to Virtex-4 or Virtex-5 and are synthesized with the design as a constant; all Virtex-4 and Virtex-5 devices are supported. The generic device in the top level model is used to locate the correct device ID in the VHDL LUT. Second, the frame address register is formatted differently for Virtex-4 and Virtex-5, requiring small changes in the ordering of the fault list block RAM data output bus. Finally, the input/output ordering for the ICAP in Virtex-5 is byte-swapped, compared to Virtex-4 ICAP.

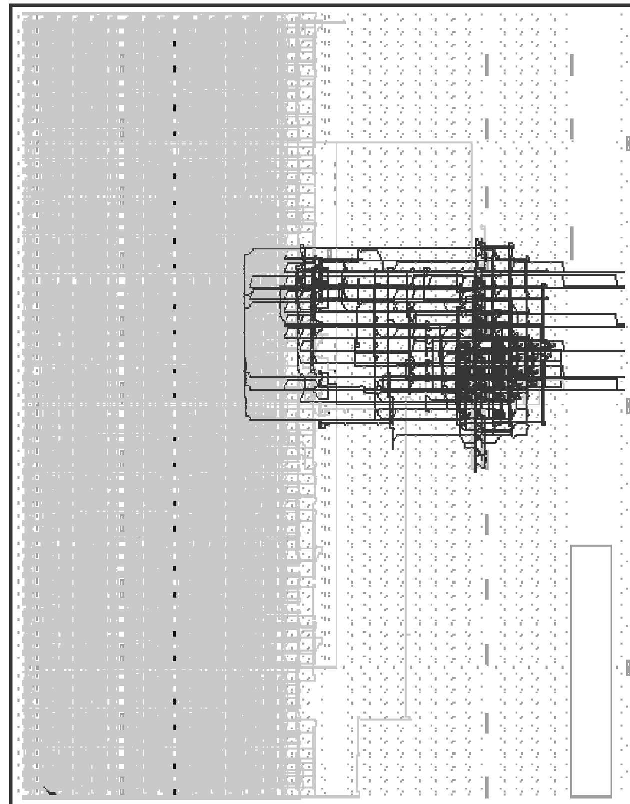


Figure 5.8: Routed embedded fault inject core (right) with half-array of routed CLB BIST (left) in Virtex-5 LX20T

Table 5.5: Fault/SEU injection core I/O descriptions

Name	Direction	Description
CLK	Input	Clock input up to 100MHz (ICAP max)
GO	Input	Digital 1-shot input asserted to start injection of 1 or more faults separated by 'pause' delimiters.
PAUSED	Output	Asserted to indicate injection of 1 or more faults separated by 'pause' delimiters is complete.
EOF	Output	End-of-file asserted when end of fault list is reached.

```
component fltinject is
generic(DEVICE : string(1 to 6):="LX110T");
port(        GO : in std_logic;
            CLK : in std_logic;
            EOF : out std_logic;
         PAUSED : out std_logic);
end component fltinject;
```

Figure 5.9: Fault inject core component declaration

The details of the primary inputs and outputs of the embedded core are summarized in Table 5.5. The normal embedded fault injection process with a free running system clock (up to 100 MHz) is as follows: (1) Download BIST configuration with embedded fault injection core. (Optionally load fault list via Boundary Scan user access register). (2) Toggle the GO input. Fault injection begins and runs to completion or until a "pause at fault" is encountered. (3) Monitor the PAUSED and EOF outputs. When PAUSED is asserted, execute the BIST configuration and record results. Repeat steps 2 and 3 until both PAUSED and EOF are asserted, then go to step 4. (4) Execute the BIST for a final time and record results. The end of fault file is reached and fault injection is complete.

The embedded fault injection core has been verified on Virtex-4 and Virtex-5 devices. The core was initially verified by synthesizing only the core, loading a fault list, and executing the fault injection. To verify the injection of faults and bit-flips, the contents of the configuration memory were read back via the Boundary Scan interface and compared line-by-line to the

original configuration download file. The core is capable of injecting stuck-at faults and SEU bit-flips anywhere in the configuration memory except block RAM contents. It is possible, however, to modify the architecture to support injection of faults in block RAM contents. Transient faults can be emulated by back-to-back SEU bit-flips such that the fault exists for a minimum of 3 μs - the minimum RMW time for a single frame. By incorporating two back-to-back bit-flips with a 'pause' delimiter, the user can control a transient fault for longer periods.

5.4 Summary and Conclusions

We have presented case studies for two embedded processor approaches for SEU and fault injection emulation in FPGA and FPGA cores in reconfigurable SoCs. In the first case, a dedicated hard core processor was used to inject emulated faults in the FPGA core configuration memory via a write-only interface. The lack of read access to the configuration memory increased the development effort and difficulty for use in the evaluation and analysis of BIST configurations for the FPGA. In the second case, a soft core processor was developed which was capable of read-modify-write access to the FPGA configuration memory. This facilitates the emulation of single and multiple stuck-at faults as well as bit-flipping for emulation of single and multiple SEUs. Hence, the embedded SEU/fault emulation processor supports a wide variety of fault types with no download penalty for more efficient and thorough evaluation of BIST and SEU mitigation. It should be noted that the fault injection is used in a fault-free device to analyze SEU detection/correction and BIST development and is not part of the manufacturing or system-level operation or test.

5.5 Acknowledgements

The contents of this chapter were published under the title "Embedded Processor Based Fault Injection and SEU Emulation for FPGAs" in *Proceedings of the International Conference*

on Embedded Systems and Applications, 2009, pp. 183-189. Prof. Charles Stroud and former Auburn University Department of Electrical and Computer Engineering students Mustafa Ali and John Sunwoo are co-authors on the paper. A majority of the actual research and the writing of the published paper represents the efforts of the primary student author and not collaborators, and the research represents work performed while in the graduate program at Auburn University.

5.6 References

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

[2] F. Kastensmidt, L. Carro and R. Reis, *Fault-Tolerance Techniques for SRAM-based FPGAs*, Springer, 2006.

[3] E. Johnson, M. Caffrey, P. Graham, N. Rollins and M. Wirthlin, "Accelerator Validation of an FPGA SEU Simulator," *IEEE Trans. on Nuclear Sci.*, vol. 50, no. 6, pp. 2147-2157, 2003.

[4] P. Ellervee, J. Raik, K. Tammemäe and R. Ubar, "Environment for FPGA-based Fault Emulation," *Proc. Estonian Acad. Sci. Eng.*, vol. 12, pp. 323–335, 2006.

[5] S. Hwang, J. Hong and C. Wu, "Sequential Circuit Fault Simulation Using Logic Emulation," *IEEE Trans. on CAD of ICs and Systems*, vol. 17, no. 8, pp. 724-736, 1998.

[6] P. Civera, L. Macchiarulo, M. Rebaudengo, M. Reorda and M. Violante, "An FPGA-Based Approach for Speeding-Up Fault Injection Campaigns on Safety-Critical Circuits," *Journal of Electronic Testing: Theory and Applications*, vol. 18, pp, 261–271, 2002.

[7] R. Sedaghat, "Routability estimation of FPGA-based fault injection," *Electronics Letters*, vol. 41, no. 14, pp. 790-792, 2005.

[8] T. Slaughter, C. Stroud, J. Emmert and B. Skaggs, "Fault Injection Emulation for Field Programmable Gate Arrays," *Proc. Int. Society for Optical Eng.*, vol. 4525, pp. 1-9, 2001.

[9] B. Dutton and C. Stroud, "Single Event Upset Detection and Correction in Virtex-4 and Virtex-5 FPGAs," *Proc. ISCA Int. Conf. on Computers and Their Applications*, pp. 57-62, 2009.

[10] *AT94K Series Field Programmable System Level Integrated Circuit*, Datasheet, Atmel Corp., 2001.

[11] J. Sunwoo and C. Stroud, “Built-In Self-Test of Configurable Cores in SoCs Using Embedded Processor Dynamic Reconfiguration,” *Proc. Int. SoC Design Conf.*, pp. 174-177, 2005.

[12] *Virtex-4 FPGA Configuration Guide*, UG071 (v1.5), Xilinx Inc., 2007.

[13] *Virtex-5 FPGA Configuration User Guide*, UG191 (v2.7), Xilinx Inc., 2008.

[14] B. Dutton and C. Stroud, “Built-In Self-Test of Configurable Logic Blocks in Virtex-5 FPGAs,” *Proc. IEEE Southeastern Symp. on System Theory*, pp. 235-249, 2009.

[15] B. Garrison, D. Milton, and C. Stroud, “Built-In Self-Test for Memory Resources in Virtex-4 FPGAs,” *Proc. ISCA Int. Conf. on Computers and Their Applications*, pp. 63-68, 2009.

Chapter Six. Soft-Core Embedded Processor-Based Built-In Self-Test of FPGAs

This chapter presents the first implementation of Built-In Self-Test (BIST) of Field Programmable Gate Arrays (FPGAs) using a soft core embedded processor for reconfiguration of the FPGA resources under test, control of BIST execution, retrieval of BIST results, and fault diagnosis. The approach was implemented in Xilinx Virtex-5 FPGAs but is applicable to any FPGA that contains an internal configuration memory access port

6.1 Introduction

Built-In Self-Test (BIST) for Field Programmable Gate Arrays (FPGAs) exploits the re-programmability of FPGAs to create BIST circuitry in the FPGA fabric during manufacturing and system-level off-line testing [1]. The only overhead is the external memory required to store the BIST configurations along with the time required to download and execute the BIST. No area overhead or performance penalties are incurred in the user function because the BIST logic is replaced by the intended system function after testing is complete. The BIST configurations are applicable to all levels of testing because they are independent of the intended system function and require no specialized external test fixture or equipment. Most research and development in BIST for FPGAs has focused on reducing the number of test configurations, reducing the size of test configuration files, and decreasing BIST execution time [2]-[8]. But the ever increasing complexity and level of integration in FPGAs has, with few exceptions, resulted in longer test times, more downloads, and more memory required for storing BIST configurations for each new generation of FPGA. However, the increasing size and complexity of FPGAs have also created opportunities for innovation in FPGA testing.

This chapter presents the first implementation of BIST for FPGAs using a soft core embedded processor synthesized into the fabric of the FPGA under test. The approach reduces the number of configuration files required for BIST by exploiting the regularity of BIST structures to significantly compress and store partial configuration data in the embedded processor's program memory. The embedded processor controls and executes the BIST sequence, including retrieval and analysis (fault diagnosis) of BIST results, and reconfiguration of the FPGA for subsequent BIST configurations. This embedded processor based BIST approach is possible for two reasons: first, the growing size and complexity of FPGAs facilitates the inclusion of complex circuitry that only occupies a small percentage of the total configurable resources, leaving adequate area for BIST logic; and, secondly, the ability to access the configuration memory from inside the FPGA fabric has made possible internal reconfiguration and read back. The approach has been successfully implemented in Xilinx Virtex-5 but is applicable to any FPGA with internal configuration memory access.

6.2 Background

A number of BIST approaches have been developed for the configurable logic and memory resources in FPGAs [1]. Due to the programmable nature of resources to be tested, all BIST approaches for FPGAs require multiple configurations in order to obtain high fault coverage. Generally, a BIST approach is organized into test *sessions* and *phases* [2]. Each test session consists of a set of test phases (test configurations) for a particular resource under test in order to test that resource in all modes of operation. For example, BIST of configurable logic blocks (CLBs) requires two test sessions. In the first test session, half of the CLBs are configured as blocks under test (BUTs), with the remaining half serving as comparison-based output response analyzers (ORAs) and test pattern generators (TPGs). In recent CLB BIST approaches,

the TPGs are implemented in non-CLB resources freeing CLBs to function as additional ORAs such that circular comparison can be implemented, as illustrated in Figure 6.1, where the outputs of each BUT in a row or column are monitored by two ORAs and compared to the outputs of two other identically configured BUTs [1]. This circular comparison in conjunction with multiple identically configured TPGs provides high diagnostic resolution with low probability of fault escape [1]. In the second test session, the positions of the BUTs and ORAs are swapped, such that every CLB is configured as a BUT in one test session and as ORA in the other test session.

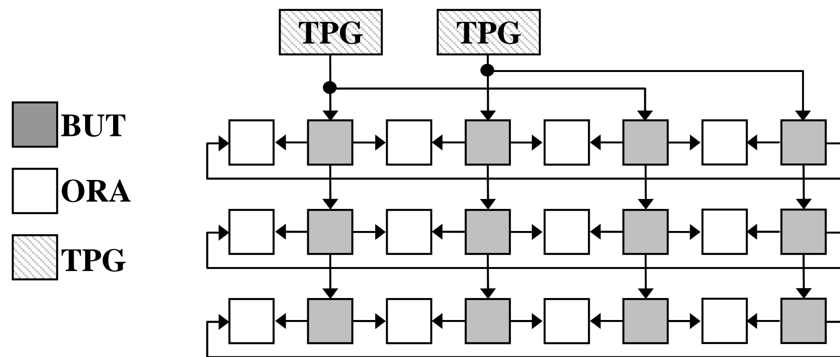


Figure 6.1: Configurable logic block (CLB) BIST architecture

BIST control, including downloading the initial BIST configuration, executing the BIST sequence, retrieval of results, fault diagnosis based on failing results, and reconfiguration of subsequent BIST phases, has traditionally been achieved via interface to an external BIST controller. However, the increased complexity of FPGAs, large number of test configurations associated with various programmable resources, and speed limitations of external download interfaces result in long manufacturing test times and limit practicality of system-level testing. Various approaches have been investigated to reduce the overall test time while achieving high quality tests. Beyond minimizing the number of test phases, partial reconfiguration reduces test time by reconfiguring only the resources under test for various modes of operation once the

overall test structure has been downloaded into the device. BIST configurations that have been recently developed for Virtex-4 and Virtex-5 FPGAs include a single-bit pass/fail output to eliminate retrieval of ORA contents for passing test phases or when fault diagnosis is not desired [5]-[8]. When failures are observed, partial configuration memory read back can be used to obtain the ORA contents to diagnose the faulty resource(s) for fault tolerant applications. Beyond these techniques, the only new development in FPGA BIST has been introduction of embedded processor based approaches.

Prior work in embedded processor based BIST includes system-on-chip (SoC) testing with hard core microprocessors [9] but did not address testing of FPGAs or FPGA cores in SoCs. The first embedded processor based BIST approach for FPGAs was developed to minimize test time, number of downloads, and complexity of the external BIST controller by relocating BIST reconfiguration, control, and diagnosis to the dedicated hard core embedded processor in the Atmel AT94K series configurable SoC [3][4]. The device consists of an FPGA core, various RAMs, and an 8-bit Advanced Virtual RISC (AVR) microcontroller [10]. Sets of BIST configurations were developed to test each of the various programmable resources in the FPGA core including CLBs, RAMs, IOBs, and programmable routing network [3][4]. The embedded processor was used to configure the FPGA for each test session, execute the BIST sequence, retrieve BIST results from the ORAs, and perform diagnosis based on failing BIST results. This embedded processor based BIST approach achieved a total test time speed-up of about 43.5 over the tradition approach of downloading each BIST configuration [4]. External memory requirements for storing BIST configurations were reduced by a factor of about 158 because only a single program needed to be downloaded into the AVR program memory, from which all BIST configurations were generated and executed.

While this embedded processor based BIST approach was practical for system-level testing, the approach was developed specifically for AT94K devices such that application to other FPGAs is limited due to reliance on the hard core processor with dedicated program memory. Some hardcore processors (such as the PowerPC in Virtex-4 and Virtex-5 FX series FPGAs) do not have a dedicated program memory and must use programmable resources in the FPGA. Soft core processors, on the other hand, can be implemented in most FPGAs such that a soft core processor based approach would be applicable to a wider range and variety of FPGAs and applications. The primary requirement is that the FPGA include an internal configuration access port (ICAP) to provide processor access to the configuration memory.

The configuration memories of Virtex-4 [11] and Virtex-5 [12] FPGAs are partitioned into frames, where each frame has a fixed length of 1,312 bits, or forty-one 32-bit words. A frame is the smallest addressable segment of the configuration memory; therefore all memory read/write operations must be performed on whole frames. This means that individual FPGA resources cannot be reconfigured without also providing explicit reconfiguration data for other FPGA resources that occupy the same frame. In Virtex-5, a frame contains the configuration data for 20 rows of CLBs and (I/O) tiles, or 5 rows of block RAMs and DSPs tiles in the same column. In Virtex-4, a frame contains configuration data for 16 rows of CLBs and I/O tiles, or 4 rows of block RAMs and DSP tiles.

Both Virtex-4 and Virtex-5 FPGAs include several configuration registers to access the configuration memory, including Frame Address Register (FAR), Frame Data Register Input (FDRI), and Frame Data Register Output (FDRO) which facilitate writing/reading data to/from a specific frame of configuration memory. There are other registers for functions such as status (STAT), cyclic redundancy check (CRC), command (CMD), etc. To access the configuration

memory, a combination of these registers must be used. These registers are normally accessible from both Boundary Scan and SelectMAP configuration interfaces but are also accessible via the ICAP located inside fabric. The ICAP works like the external SelectMAP configuration interface except that it has separate 32-bit write and read buses, as opposed to a bidirectional 32-bit bus. The maximum ICAP clock frequency is 100 MHz.

6.3 Embedded BIST Architecture

The soft core embedded processor based BIST approach for FPGAs incorporates additional logic in the FPGA fabric along with the BIST logic to perform tasks typically assigned to an external BIST controller or computer. The embedded BIST approach offers several advantages over the external BIST approach. First, the 32-bit ICAP configuration interface is used for reconfiguration, eliminating the test time penalties associated with the lower speed serial Boundary Scan interface. Secondly, the total number of external download configurations is reduced to one per test session. In addition, all control of the BIST configurations and sequences can be implemented in the embedded controller. Diagnostic procedures can also be performed by the embedded BIST controller, further reducing the complexity of the external BIST controller in fault tolerant applications and providing considerable speed-up when compared to Boundary Scan based read back and diagnosis.

The implementation of the embedded processor BIST approach in Virtex-5 FPGAs incorporates elements of both hardware and software design to achieve an architecture that is general enough for any Virtex-5 device as well as for any BIST approach for the resources in Virtex-5 FPGAs. The design is applicable to any Virtex-5 device with only minor modifications to system software and no modifications to system hardware. Furthermore, the design can easily be extended to Virtex-4 devices for similar improvements in test time. To minimize the number

of external downloads per test session, the embedded processor based BIST hardware must fit in one half of the smallest supported device. The embedded processor core must also be capable of storing configuration data for all of the subsequent test phases for each test session in memory in the FPGA fabric using Block RAMs or distributed RAMs. Finally, the core must support interfaces for connecting with the ICAP and BIST circuitry. There are a variety of designs which can be used for the embedded processor ranging from fast, full-custom register transfer level (RTL) designs, to highly configurable general purpose soft core microprocessors. While RTL level designs are useful for simple repetitive tasks, this approach is not very efficient for supporting multiple device architectures of a variety of BIST approaches. Such an approach requires a different hardware configuration for each device and for each BIST session, which requires a significant amount of hardware development time when compared with other, more general purpose software based approaches. Another option is to use a general purpose processor in the form of a "soft" intellectual property (IP) core. One of the simplest and most efficient general purpose architectures available for Xilinx FPGAs is the PicoBlaze 8-bit microcontroller [13]. The PicoBlaze occupies one block RAM and approximately 50 slices in Virtex-5 FPGAs – much less than half of an array in the smallest Virtex-5 device. The PicoBlaze is supported by a simple assembler and software simulator. However, the program memory in the PicoBlaze is limited to 1024 stored instructions and scratch-pad memory is limited to 64 Bytes. The 8-bit architecture also creates timing penalties when interfacing with the 32-bit ICAP port because each ICAP write requires a minimum of four PicoBlaze instructions of two clock cycles each. To improve timing for ICAP operations, a 32-bit architecture is best for embedded BIST applications in Virtex-4 and Virtex-5 devices. One IP core that meets the requirement for a BIST controller is the MicroBlaze soft core processor, which is a highly configurable 32-bit general

purpose RISC microprocessor for Xilinx FPGAs [14]. The MicroBlaze also includes an optional, pre-engineered, interrupt driven ICAP hardware interface. The MicroBlaze can be configured with up to 64 kB of combined program and initializable data memory in Virtex-5 FPGAs that is implemented in the FPGA fabric in Block RAMs. The processor can be modified by the addition of custom peripherals on the processor local bus (PLB). These features led to selection of MicroBlaze as the embedded processor in our implementation.

The basic architecture for the embedded processor BIST approach is illustrated in Figure 6.2 for CLB BIST where half of the FPGA array is used for processor and additional hardware resources and the other half of the array contains the CLB BIST configuration. Custom memory-mapped registers are included in the MicroBlaze VHDL model for interfacing with the BIST circuitry. The processor interfaces directly with the ICAP for reconfiguration of the BIST array and read back of BIST results. To test all CLBs in the FPGA, a second configuration is generated with locations of BIST logic and embedded processor swapped, as shown in Figure 6.2b. For some resources, such as I/O tiles or CRC modules, it is possible to test all of the resources simultaneously by placing the MicroBlaze around the BIST circuitry.

One memory mapped write-only (WO) register, shown in Table 6.1, is included for control of the BIST circuitry and sequence. The outputs of the register are connected directly to inputs to the BIST logic, but not all of the register bits are utilized in any one BIST session. One read-only (RO) register, also shown in Table 6.1, is included at the same memory-mapped address as the output register. The inputs to this register are connected directly to outputs of the BIST logic. Each register is general enough to be applicable to all BIST configurations that have been developed for Virtex-5.

Because many BIST configurations must be executed for a different minimum number of clock cycles to achieve the intended fault coverage, there is the need for a hardware timer for BIST execution. Therefore, a 16-bit down counter is included in addition to the RO and WO BIST control registers. The counter is initialized by writing to the lower 16-bits of the BIST control register. The counter automatically counts down to zero, setting the *cnt_eq* bit when zero. The *cnt_eq* bit is used to enable the BIST logic and can be polled in software to determine when the BIST is complete. The counter clock and BIST clock can share the MicroBlaze clock or can be clocked independently at a higher clock frequency by connecting the BIST clock input and 16-bit counter clock to an independent BIST clock source.

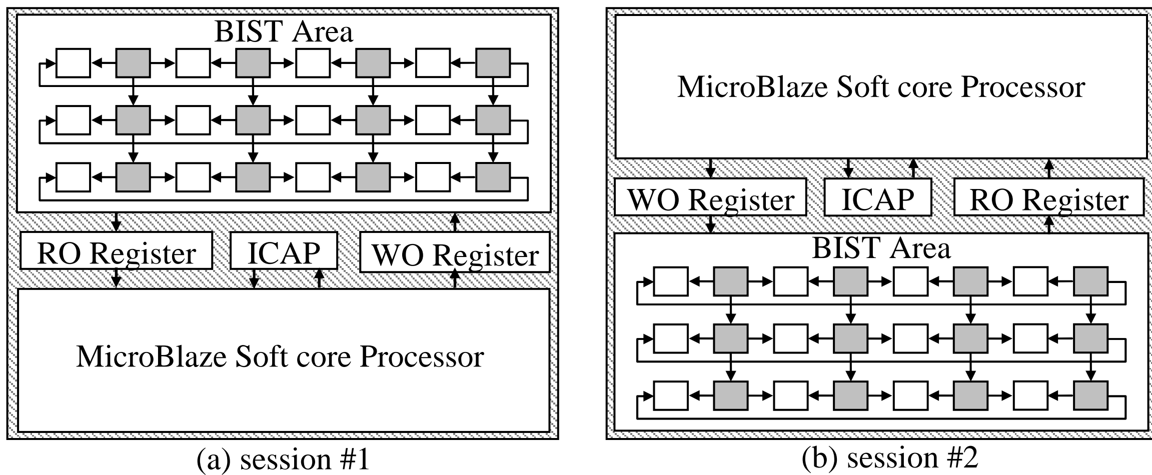


Figure 6.2: Embedded soft core processor based BIST architecture

Table 6.1: BIST control registers

Write-only register								**Read-only register**			
31:24	**23:21**	**20**	**19**	**18**	**17**	**16**	**15:0**	**31:3**	**2**	**1**	**0**
control	reserved	done	reset	start	tdi	clk_en	cnt_init	reserved (read as 0)	cnt_eq	BIST_done	tdo

6.4 Software Development

One important feature of this BIST approach stems directly from the generality of the embedded processor. Namely, that only the software changes from one BIST session to the next; the hardware remains unchanged for any and all BIST sessions. The software can be efficiently constructed in a manner that exploits the regularity of BIST configurations, and only the code for a particular BIST session need be compiled and programmed in the MicroBlaze program memory since a new download is performed at the start of each test session. Each BIST for a specific resource is composed of a set of phases, with each phase corresponding to a reconfiguration of the resources under test. Each phase comprises writes of entire set of frames of data to configuration columns that control the resources under test. Therefore, only certain portions of the partial reconfiguration files must be stored because the array-half, row, and column locations of the resources under test can be determined algorithmically based on the particular device in which BIST is implemented. The algorithm for the embedded BIST reconfiguration process is shown in Figure 6.3. The algorithm for frame address generation using multi-frame write operations, given configuration row and destination minor address for frame data previously written to FDRI, is also shown in Figure 6.3.

No modification to the MicroBlaze hardware is required for support of other BIST approaches such as DSP and Block RAM BIST [5][6]. However, the control bits [31:24] of the WO BIST register are used during these test sessions to control the TPG mode. The outputs of these register bits are connected directly to the mode control inputs of the TPG when the MicroBlaze hardware and BIST hardware are merged. Block RAM and DSP embedded BIST architectures are otherwise arranged identically to the CLB BIST architecture shown in Figure 6.2 with the BIST circuitry occupying one-half of the device.

Reconfiguration files are generated in a manner that allows full or partial reconfiguration from an external memory without the need for an "intelligent" controller. While ideal for systems containing only non-volatile memory and an FPGA, the partial reconfiguration files are too large to be directly stored in the program or data memory of an embedded processor. For example, the total size of the 5 partial reconfiguration files for CLB BIST in half of an array of a small Virtex-5 device (LX30T) is 41,360 Bytes – exceeding the maximum 32 kB of data memory that can be allocated for MicroBlaze. Partial reconfiguration files are also device dependent since the size of the reconfiguration file is proportional to the device size. Hence, compression of partial reconfiguration files is required for the embedded processor.

Overall BIST algorithm	Addressing algorithm w/ multi-frame write
for all test phases **do**	**for all** configuration columns **do**
for all configuration *rows* in BIST *half* **do**	**if** column is block under test **then**
for all frames in reconfiguration structure **do**	**for all** minor addresses in compressed config **do**
construct configuration frame	multi-frame write to *row*, *column*, & *minor*
muti-frame write to all RUTs in *half* & *row*	**end for**
end for	**end if**
end for	**end for**
execute BIST phase	
get BIST results	
end for	
set done bit in WO control register	

Figure 6.3: Embedded processor BIST algorithms

Our compression scheme exploits four features of Virtex-5 partial reconfiguration files to compress the data for storage in the embedded processor program memory and eliminate device dependencies. First, each configuration file contains certain instructions, such as those for multi-frame writes to the configuration memory, which are repeated many times during download. Since, in the embedded processor BIST approach, the download is executed under the control of the embedded processor, ICAP instructions can be stored once and regenerated when needed. Second, multi-frame writes can only occur in one configuration row in Virtex-5 devices. For BIST configurations, which create identical configurations in BUTs and ORAs in every

configuration row, the overhead of multi-frame write instructions can be eliminated by storing frame data only once for one configuration row; the structure of the partial reconfiguration file can be regenerated by repetitively writing the frame data and frame addresses to the ICAP inside of a software loop for all configuration rows. Third, because one frame of configuration data spans 20 rows of identical resources under test, 2 to 4 words of frame data are repeated 10 to 20 times (in a repeating sequence) in each 41-word frame for BIST. Therefore, only 2 to 4 words of configuration data need to be stored for each frame in the partial reconfiguration file. The frame can be reconstructed in its entirety from the smallest repeating set of 32-bit words. Finally, the partial reconfiguration file includes the addresses of every frame to which frame data must be written for each configuration row. Again, due to the regularity of the BIST structure, only the minor addresses in the first BUT column for each configuration frame need to be stored. The remaining addresses can be regenerated algorithmically (Figure 6.3) given the locations of resources under test in the FPGA fabric. We constructed a program to automatically extract only the essential data from every partial reconfiguration files for any BIST session using the compression methods described above. The program generates a C header file with a data structure containing only essential data from the compressed partial reconfiguration file. The data structure declaration is shown in Figure 6.4, where the constant NRECONFIG is the number of test phases for the BIST session.

When the compression program was used to compress the 5 partial reconfiguration files for a CLB BIST session in a Virtex-5 LX30T, the total size of the files reduced from 41,360 Bytes to 820 Bytes. Table 6.2 shows the size of the original compressed partial reconfiguration files and the size of the essential data in compressed form for different BIST sessions for Virtex-5. The original file size given in the table is for an LX30T and the size of the file will increase in

proportion to the number of configuration rows in a given device. However, the size of the essential data in compressed format is independent of the device size. Figure 6.5 illustrates these device dependencies of reconfiguration file sizes for the smallest and largest devices in each Virtex-5 family of devices (LXT, SXT, FXT, and TXT).

```
struct framedata {
   unsigned int numword;        //# of words
   unsigned int word[MAXWORD];  //config data
    unsigned int numminor;       //# of addresses
   unsigned int minor[MAXMINOR];//minor addr
};
struct partialconfig {
   unsigned int numframe;           //# frames
   struct framedata frame[MAXFRAME];//frames
} config[NRECONFIG] = {
  //compressed frame data placed here by program
};
```

Figure 6.4: Compressed BIST partial reconfiguration structure in C

Table 6.2: Compressed partial reconfiguration data size

BIST Session	Number of BIST Sessions	Number of BIST Reconfigurations	Original File Size (Bytes)	Compressed Size (Bytes)
CLB East	2	5	41,360	820
CLB West	2	5	41,360	820
LUT-RAM	2	4	10,944	1,232
I/O Logic	1	5	11,308	1,236
I/O SerDes	1	8	94,432	2,680
CRC	1	1	4,716	184
DSP	1	9	28,836	1152
Block RAM	2	5	285,040	4920
ECC RAM	2	2	19,384	1200
FIFO	2	3	29,076	1800
FIFO ECC	2	1	9,692	600

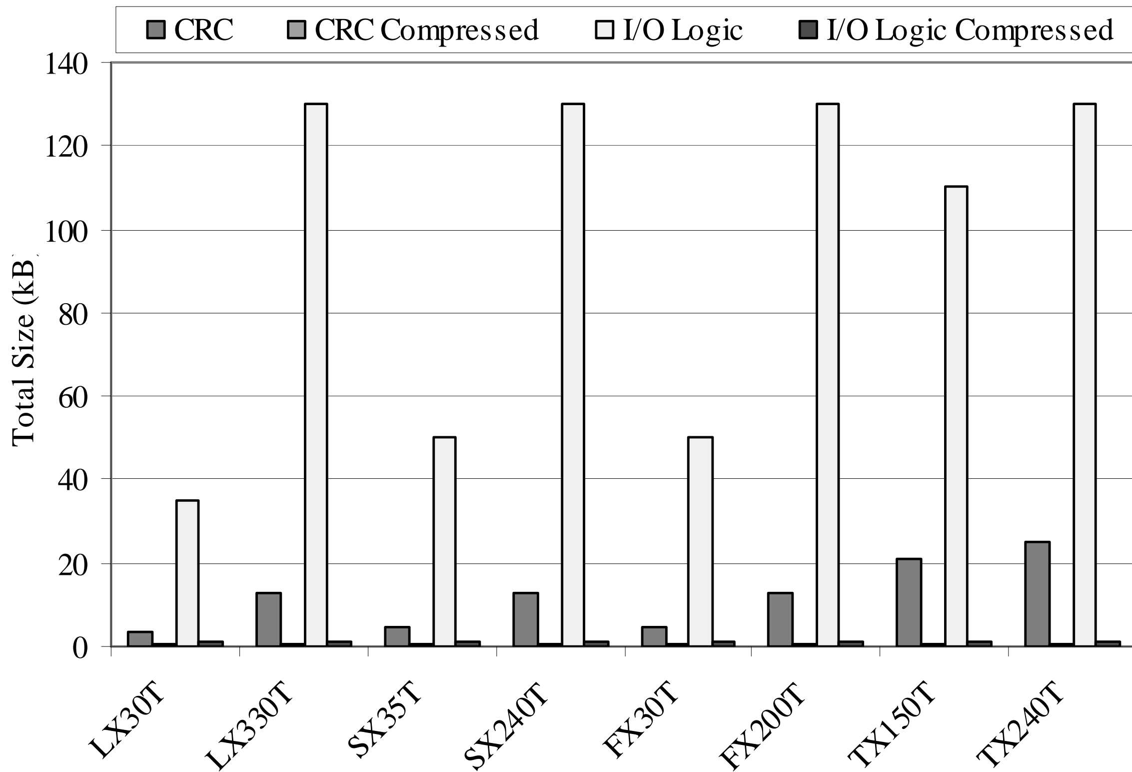
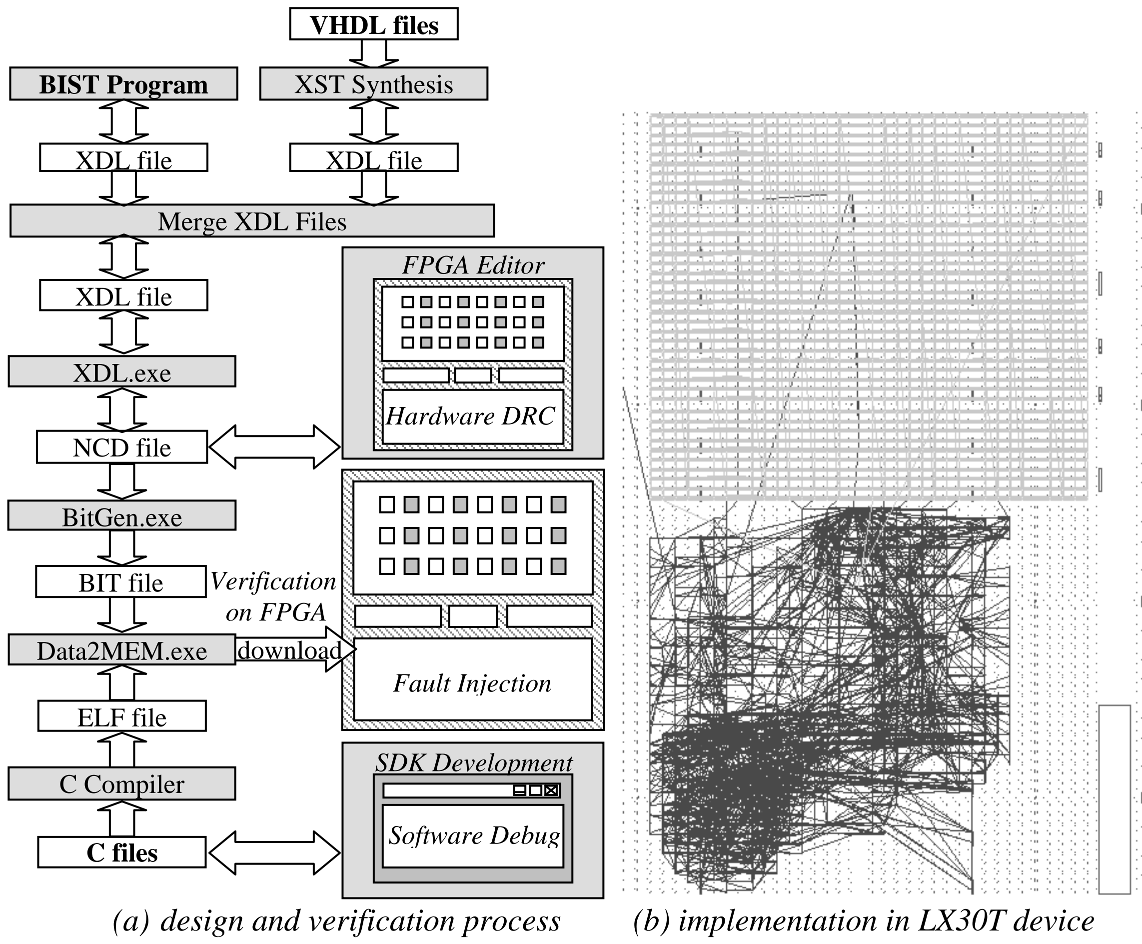


Figure 6.5: Original reconfiguration file sizes and compressed data structure sizes for one CRC BIST and a set of 5 I/O Logic BIST partial reconfigurations

Read back and diagnosis of BIST phases is performed by software in the embedded BIST processor when fault diagnosis is desired for a given application. The basic idea is to read back every frame of configuration memory containing an ORA flip-flop. The ORA flip-flop contents are then stored in an array in the processor data memory. An ORA contains a logic 0 when a failure is detected, otherwise a logic 1. Since the locations of ORAs are known for every BIST session in any device, the frame addresses of ORA flip-flops can be generated algorithmically during read back. The diagnostic algorithm [1] for circular comparison is easily implemented in the embedded processor to identify faulty resources. When combined with the 32-bit parallel access to the configuration memory, read back and diagnosis via the embedded processor provides a substantial improvement in test time when compared to serial access via Boundary Scan.

6.5 Design Flow and Implementation Results

The embedded BIST processor design flow, illustrated in Figure 6.6a, is more complex than the traditional BIST design flow due to the inclusion of the MicroBlaze processor and BIST session specific software. Generating the embedded processor based BIST configurations requires inputs from three sets of source files. First, the C source file for the specific BIST approach (e.g. CLB, DSP, block RAM, etc.) is compiled to an executable linkable file (ELF) format. The MicroBlaze hardware is modeled in VHDL and synthesized using the Xilinx ISE design flow. The placement of the MicroBlaze logic is constrained to one half of the device. The placed, unrouted design is then converted to Xilinx Design Language (XDL) format. The BIST logic is generated concurrently by the BIST generation program which produces an unrouted XDL description of the BIST circuitry. The BIST array is constrained to the other half of the FPGA. The BIST XDL description and the MicroBlaze XDL description are merged by concatenating the two XDL files and connecting primary inputs and outputs of the BIST logic to the WO and RO BIST control registers included in the MicroBlaze logic to form the complete unrouted embedded processor based BIST configuration in XDL format. Finally, the complete hardware portion of the design is converted to an NCD format and routed, from which the bitstream configuration file is generated using the Xilinx BitGen program. At this point, the compiled software in ELF format is translated into Block RAM initialization values in the bitstream download file using the Xilinx Data2Mem program.



(a) design and verification process *(b) implementation in LX30T device*

Figure 6.6: Embedded processor BIST design implementation

The embedded processor based BIST approach has been successfully implemented for BIST in Virtex-5 FPGAs. The unrouted embedded processor based BIST configuration for the CLBs implemented in the top of a Virtex-5 LX30T is shown in Figure 6.6b. Two such configurations are implemented to fully test the CLBs with the locations of the BUTs and ORAs swapped, and another two configurations are required to test the bottom half CLBs. For the purpose of embedded BIST, the MicroBlaze processor is configured with a hardware integer multiplier, five stage pipeline, and 64 kB of on-chip program and data memory (configured in Block RAMs). In Virtex-5 devices, the MicroBlaze with ICAP interface and BIST control registers occupies three DSPs, 16 block RAMs, and 400 CLBs. The percentage of utilized

resources is less than 50% in the smallest Virtex-5 device (LX20T). Timing analysis indicates that the maximum operating frequency of the MicroBlaze processor when constrained to one-half of a device is greater than 100 MHz in all Virtex-5 devices. Therefore, all ICAP operations can be performed at the maximum frequency of 100 MHz.

6.6 Conclusions

We have presented the first embedded soft core processor based FPGA BIST approach. The approach reduces the number of external configurations of the FPGA during any BIST session to a maximum of two (one for each half of the array); however, many resources can be tested in a single BIST session. The embedded processor performs reconfiguration of the resources under test at the maximum allowable clock frequency and data width. Read back of ORA contents can be performed when fault diagnosis is desired for fault-tolerant applications. The soft core processor approach was implemented in Virtex-5 FPGAs using the MicroBlaze processor. However, the overall approach is applicable to any FPGA with internal write and read access to the configuration memory.

6.7 Acknowledgements

The contents of this chapter were published under the title "Soft Core Embedded Processor Based Built-In Self-Test of FPGAs" in *Proceedings of the 24th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, 2009, pp. 29-37. Prof. Charles Stroud is a co-author on the paper. A majority of the actual research and the writing of the published paper represents the efforts of the primary student author and not collaborators, and the research represents work performed while in the graduate program at Auburn University.

6.8 References

[1] L-T Wang, C. Stroud, and N. Touba, *System-on-Chip Test Architectures*, San Francisco, CA: Morgan Kaufmann, 2007.

[2] M. Abramovici and C. Stroud, “BIST-Based Test and Diagnosis of FPGA Logic Blocks, *IEEE Trans. on VLSI Systems*, vol. 9, no. 1, pp. 159-172, 2001.

[3] C. Stroud, S. Garimella and J. Sunwoo, “On-Chip BIST-Based Diagnosis of Embedded Programmable Logic Cores in System-On-Chip Devices,” *Proc. ISCA Int. Conf. on Computers and Their Applications*, pp. 308-313, 2005.

[4] J Sunwoo and C. Stroud, “BIST of Configurable Cores in SoCs Using Embedded Processor Dynamic Reconfiguration,” *Proc. Int. SoC Design Conf.*, pp. 174-177, 2005.

[5] B. Garrison, et. al., “Built-In Self-Test for Memory Resources in Virtex-4 FPGAs,” *Proc. ISCA Int. Conf. on Computers and Their Applications*, pp. 63-68, 2009.

[6] M. Pulukuri and C. Stroud, “Built-In Self-Test of Digital Signal Processors in Virtex-4 FPGAs,” *Proc. IEEE Southeastern Symp. on System Theory*, pp. 34-38, 2009.

[7] B. Dutton and C. Stroud, “Built-In Self-Test of Configurable Logic Blocks in Virtex-5 FPGAs,” *Proc. IEEE Southeastern Symp. on System Theory*, pp. 230-234, 2009.

[8] B. Dutton and C. Stroud, “Built-In Self-Test of Programmable Input/Output Tiles in Virtex-5 FPGAs,” *Proc. IEEE Southeastern Symp. on System Theory*, pp. 235-239, 2009.

[9] R. Rajsuman, “Testing a System-On-Chip with Embedded Microprocessor,” *Proc. IEEE Int. Test Conf.*, pp. 499-508, 1999.

[10] *AT94K Series Field Programmable System Level Integrated Circuit*, DS1138, 2005.

[11] *Virtex-4 FPGA Configuration User Guide*, UG071 (v1.1), Xilinx, 2008.

[12] *Virtex-5 FPGA Configuration User Guide*, UG191 (v2.7), Xilinx, 2008.

[13] *PicoBlaze 8-bit Embedded Microcontroller User Guide*, UG129 (v1.1.2), Xilinx, 2008.

[14] *MicroBlaze Processor Reference Guide*, UG081 (v.9.0), Xilinx, 2008.

Chapter Seven. Soft-Core Embedded Processor-Based Built-In Self-Test of FPGAs Case Study

This chapter presents the results of a case study which investigates the use of an embedded soft-core processor to perform Built-In Self-Test (BIST) of the logic resources in Xilinx Virtex-5 Field Programmable Gate Arrays (FPGAs). We show that the approach reduces the complexity of an external BIST controller and the number of external reconfigurations, making it particularly appealing for in-system testing of high-reliability and fault-tolerant systems with FPGAs. However, the overall test time is not improved due to an increase in the size of the required configuration files as a consequence of the inclusion of the soft-core embedded processor logic, whose relative irregularity results in less effective compression of configuration data files.

7.1 Introduction

This chapter presents the results of the first implementation of Built-In Self-Test (BIST) for Field Programmable Gate Arrays (FPGAs) using a soft-core embedded processor synthesized into the configurable fabric of the FPGA under test. The approach, as originally proposed in [1], reduces the number of configuration files required for BIST by exploiting the regularity of BIST architectural structures to significantly compress and store partial configuration data in the embedded processor's program memory. The embedded processor controls and executes the BIST sequence, including retrieval and analysis (fault diagnosis) of BIST results, and performs partial reconfiguration of the FPGA for subsequent BIST test phases [1]. This embedded processor-based BIST approach is possible for two reasons: first, the growing size and

complexity of FPGAs facilitates the inclusion of complex circuitry that only occupies a small percentage of the total configurable resources, leaving adequate area for BIST logic and routing; and, secondly, the ability to access the configuration memory from inside the FPGA fabric has made possible internal reconfiguration and read back of FPGA logic and routing resources.

The approach was successfully implemented in Xilinx Virtex-5 [2] but is applicable to any FPGA with internal configuration memory access. The remainder of the chapter is organized as follows: Section 7.2 presents an overview of BIST for FPGAs and the previously proposed soft-core processor-based BIST technique. Section 7.3 presents the results of our implementation of soft-core embedded processor-based BIST in Virtex-5 FPGAs, including test time analysis and comparisons with other BIST approaches for FPGAs. Section 7.4 discussed ways in which the proposed approach might be improved, with Section 7.5 covering other potential applications of the approach. The chapter is summarized in Section 7.6.

7.2 Background

BIST for FPGAs exploits the re-programmability of FPGAs to create test circuitry in the FPGA fabric during off-line testing [3]. The only overhead is the external memory required to store the BIST configurations along with the time required to download and execute the numerous BIST configurations. No area overhead or performance penalties are incurred because the BIST logic is reconfigured with the intended system function after testing is complete. The BIST configurations are applicable to all levels of testing because they are independent of the end-user system function and require no specialized external test fixture or equipment. Over the past 15 years, a number of BIST approaches have been developed for the configurable logic and routing resources in FPGAs. Due to the programmable nature of FPGAs, all BIST approaches for FPGAs require multiple configurations of the resources under test in all of their modes of

operation in order to obtain high fault coverage. Some of these BIST approaches are summarized in Table 7.1, where the number of BIST configurations is given for each type of resource including configurable logic blocks (CLBs), input/output (I/O) tiles, random access memories (RAMs), digital signal processors (DSPs), and programmable routing resources.

Table 7.1: Test configurations developed for various FPGAs

FPGA	CLBs	Routing	I/O	RAMs	DSPs	References
ORCA 2C	9	27	-	0	0	[5][6]
ORCA 2CA	14	41	-	0	0	[5][6]
Delta 39K	20	419	-	11	0	[7]
4000/Spartan	12	128	-	0	0	[8]
4000XL/XLA	12	206	-	0	0	[8]
AT40K/AT94K	4	56	27	3	0	[9] - [11]
Virtex/Spartan-2	12	283	7	5	0	[11][12]
Virtex-4	10+5	84	14	15	5	[13] - [17]
Virtex-5	6+5	?	15	16	11	[17][18]

Most research and development in BIST for FPGAs has focused on reducing the number of test configurations, reducing the size of test configuration files, and decreasing BIST execution time [2]-[8]. But the ever increasing complexity and level of integration in FPGAs has, with few exceptions, resulted in longer test times, more downloads, and more memory required for storing BIST configurations for each new generation of FPGA. However, the increasing size and complexity of FPGAs has also created opportunities for innovation in FPGA testing. In [1], we proposed an embedded processor-based approach which exploits some of these features of current FPGAs in an attempt to improve test time and reduce the complexity of BIST. The soft-core embedded processor-based BIST approach for FPGAs incorporates additional logic in the FPGA fabric along with the BIST logic to perform tasks typically assigned to an external controller or computer. The new approach offers several advantages over the traditional external BIST approach. First, the 32-bit internal configuration access port (ICAP) is

used for reconfiguration of the resources under test, eliminating the test time penalties associated with the lower speed, serial Boundary Scan interface. Secondly, the total number of configurations that are downloaded via the external configuration interface is reduced to one per test session. In addition, all control of the BIST configurations and test procedures can be implemented in the embedded processor. Finally, fault diagnosis procedures can also be performed by the embedded processor, further reducing the complexity of the external BIST controller in fault-tolerant applications and providing considerable speed-up when compared to Boundary Scan based readback and diagnosis.

The basic architecture of the embedded BIST approach for CLBs is illustrated in Figure 7.1 [1]. In this particular BIST approach, one-half of the FPGA array is configured with the BIST circuitry, including multiple Test Pattern Generators (TPGs), comparison-based Output Response Analyzers (ORAs), and the Blocks Under Test (BUTs). The TPGs are constructed from CLBs or other logic resources such as DSPs, RAMs, etc. The TPGs provide identical test patterns to alternating rows or columns of identically configured BUTs whose outputs are monitored by two ORAs and compared with the outputs of two other BUTs in a circular comparison arrangement, as shown in Figure 7.1. The ORAs are constructed from CLBs such that only half of the CLBs can be BUTs in a given test session, and the positions of the BUTs and ORAs must be swapped during a subsequent test session in order to test all of the CLBs in half of the array.

The second half of the FPGA array is reserved for a MicroBlaze soft-core processor and any additional hardware resources associated with the processor [19]. Custom memory-mapped registers are included in the MicroBlaze VHDL model for interfacing with the BIST circuitry. One memory mapped write-only (WO) register is included for control of the BIST circuitry. The

outputs of the register are connected directly to all inputs to the BIST logic. One read-only (RO) register is included at the same memory-mapped address as the output register. The inputs to this register are connected directly to the outputs of the BIST logic. Each register is general enough to be applicable to all BIST configurations that we have developed for Virtex-5. Finally, the MicroBlaze interfaces directly with the FPGAs ICAP for partial reconfiguration of the BIST array and for read back of output responses. To test all of the resources in the FPGA, a second configuration is generated with the location of the BIST logic and embedded processor swapped. For BIST of some resources, such as input/output (I/O) tiles and cyclic redundancy check (CRC) circuits, it is possible to test all of the target resources simultaneously by placing the MicroBlaze around the BIST circuitry.

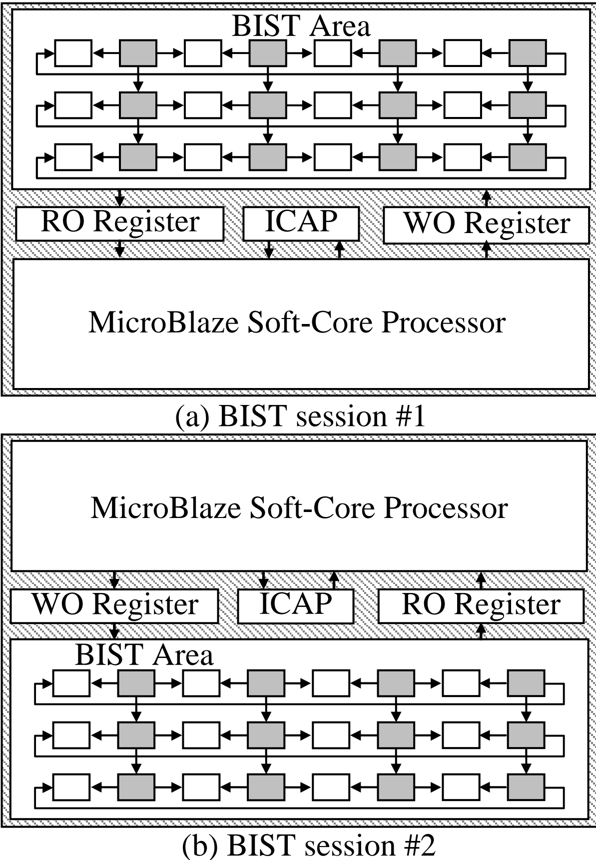


Figure 7.1: Simplified soft-core processor-based BIST architecture

7.3 Results of Implementation in Virtex-5

The embedded processor-based BIST approach was implemented for BIST of Virtex-5 FPGAs using the MicroBlaze soft processor [19]. The unrouted embedded processor-based BIST configuration for the top CLBs implemented in the Virtex-5 LX30T is shown in Figure 7.2. Note that two such configurations are implemented to fully test the top CLBs with the locations of the BUTs and ORAs swapped, and another two configurations are required to test the bottom half CLBs. For the purpose of embedded BIST, the MicroBlaze processor is configured with a hardware integer multiplier, five stage pipeline, and 64 KB of on-chip program and data memory (configured in Block RAMs). In Virtex-5 devices, the MicroBlaze with ICAP interface and BIST control registers occupies three DSPs, 16 block RAMs, and 400 CLBs. The percentage of utilized resources is less than 50% in the smallest Virtex-5 device such that the approach works for all FPGAs in the Virtex-5 family. Timing analysis indicates that the maximum operating frequency of the MicroBlaze processor when constrained to one-half of a device is greater than 100 MHz in all Virtex-5 devices. Therefore, all ICAP operations can be performed at the maximum ICAP configuration clock frequency of 100 MHz.

For accurate measurements of test time and to obtain experimental results with the MicroBlaze processor, an additional 32-bit hardware timer/counter was included in the MicroBlaze VHDL model. By starting the timer/counter at the beginning of a test phase, and stopping it at the end of the test phase, the exact number of clock cycles for reconfiguration of the resources under test, test execution, and ORA read back can be determined. To extract the value in the timer/counter at the end of each test, the MicroBlaze performs a read of the timer/counter value and reports this number via a UART interface to a connected PC, where it is displayed and logged in a terminal program.

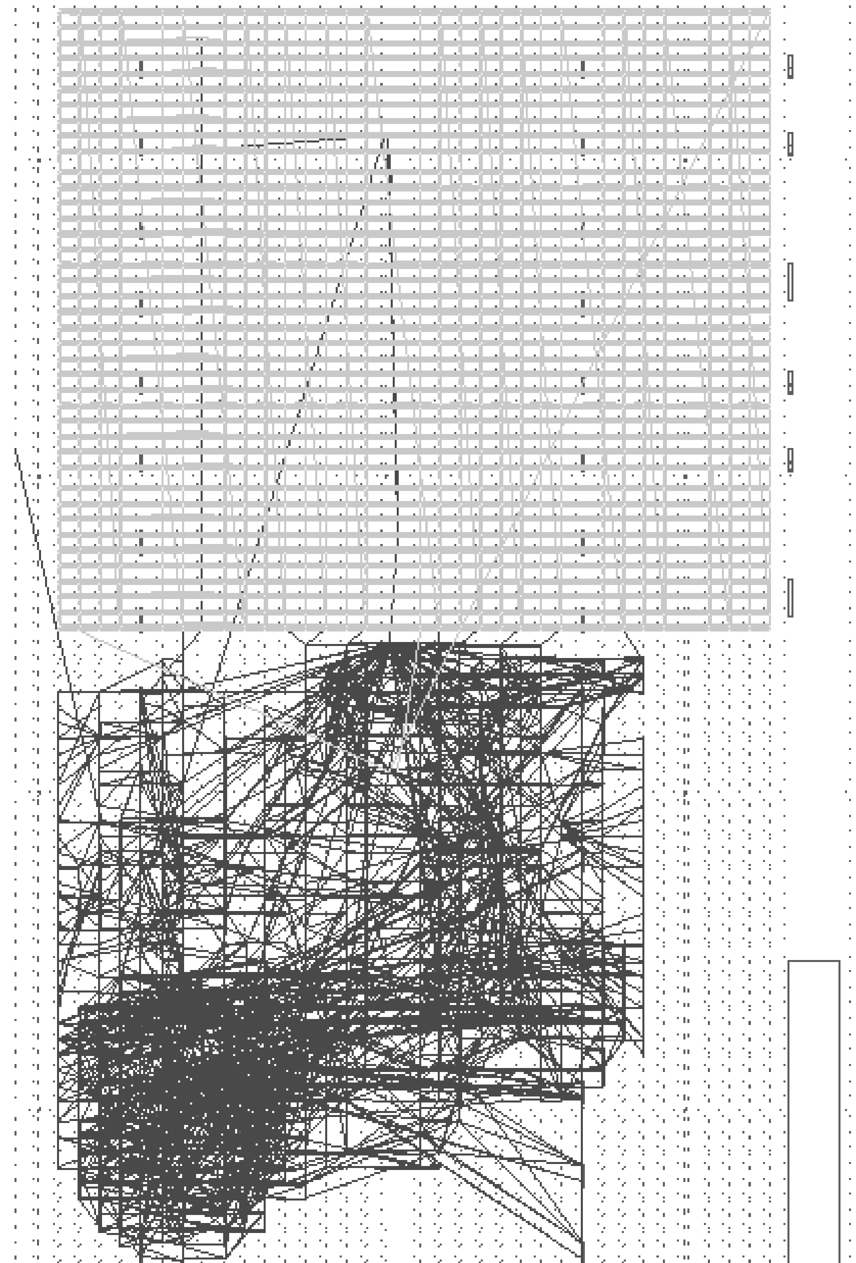


Figure 7.2: Unrouted embedded processor-based BIST configuration for top configurable logic blocks (CLB) in Virtex-5 LX30T viewed in FPGA Editor

Figure 7.3 shows the total test time for one session of CLB testing in several Virtex-5 devices for external configuration with full compressed configuration and partial compressed reconfiguration bitstreams downloaded and controlled via the 50 MHz Boundary Scan configuration interface and for the MicroBlaze embedded processor approach. These test times

take into account all of the configurations required to achieve 100% fault coverage in the CLB in SliceL mode, as reported in [7], which used traditional external reconfiguration techniques. However, these times double to achieve 100% fault coverage in every CLB, because a second set of identically sized configurations are required with the locations of the BUTs and ORAs swapped. The optimized external reconfiguration provides the fastest overall test time when compared with the other two approaches since the entire array is tested concurrently. This approach is about three times as fast as the embedded processor approach on average, but is device dependent, as can be seen in Figure 7.3. However, the embedded approach is significantly faster than external configuration with full or compressed bitstream download files.

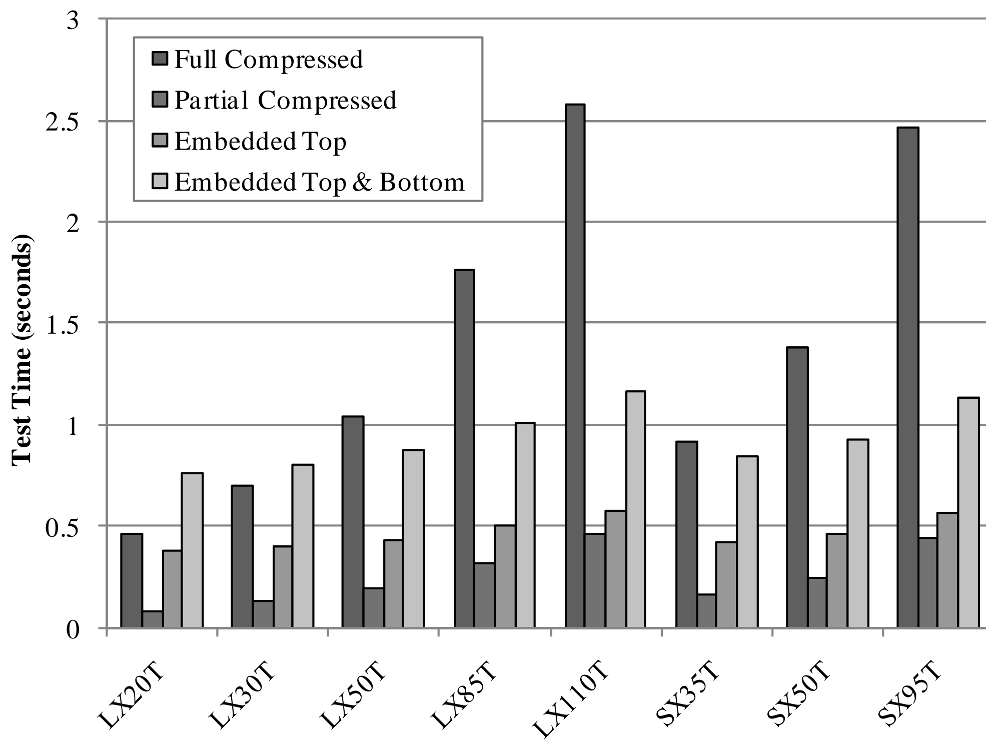


Figure 7.3: CLB BIST test time for external configuration (full compressed and partial compressed bitstreams) and embedded processor test time

By studying the configuration file sizes for the two BIST approaches, the cause for the increase in test time for the embedded processor approach becomes clear. Consider Figure 7.4,

which shows the contributions to test time for one session of CLB BIST with the embedded processor approach. The contribution from the initial compressed full configuration download (using the 50 MHz external Boundary Scan configuration interface) is shown on bottom and the contribution from the five subsequent partial reconfigurations by the embedded processor (using the 100 MHz 32-bit ICAP) is shown on top. The overall test time is dominated by the initial download time. This is due, in part, to the slower serial Boundary Scan configuration interface; however, the main contributor to the overall test time is an increase in the size of the initial configuration file (relative to the traditional BIST approach). The cause of the size increase is due to the inclusion of the MicroBlaze configuration data in the configuration file, the irregularity of which reduces the effectiveness of the configuration file compression (see Figure 7.2). We observed that the inclusion of the MicroBlaze logic increased the size of the first compressed configuration file size by 2100 kB (which is approximately constant for all devices). The additional 2100 kB of configuration data is larger than the next five partial reconfiguration files combined, and, assuming Boundary Scan configuration, increases the time for initial configuration by 336 ms. While it is possible to improve the timing for internal reconfiguration of the resources under test, there is no way to improve timing for the first compressed configuration download.

The potential for savings in test time does exists for systems which require fault diagnosis, and, therefore, read back of ORA contents at the end of each test phase. In this case, the embedded approach provides a speed-up of 5.4 times during read back of ORA results versus read back via Boundary Scan, as can be seen in Figure 7.5.

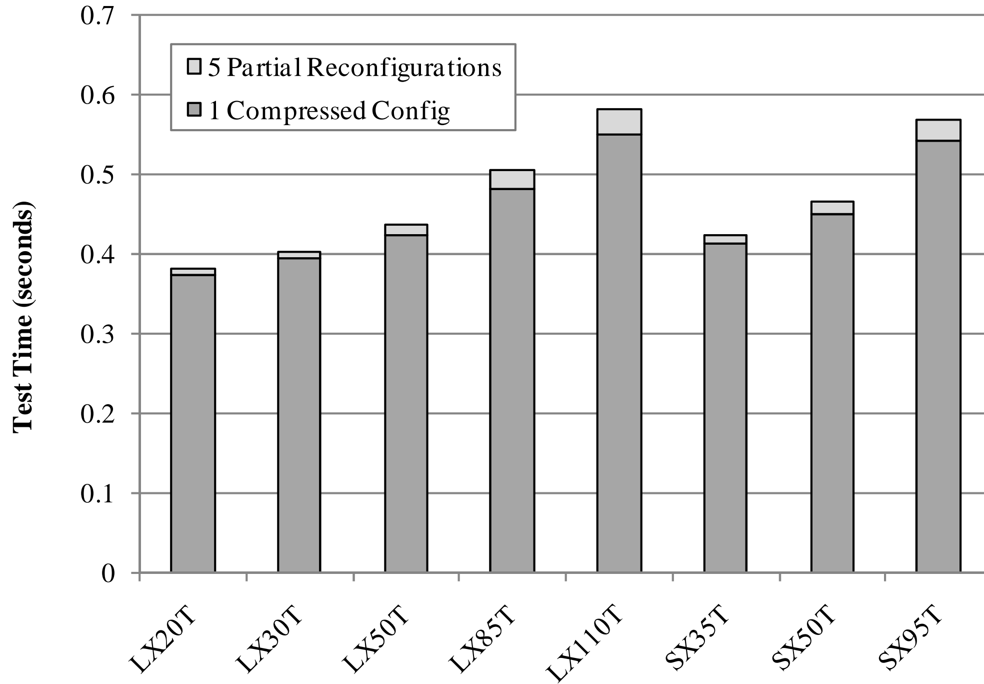


Figure 7.4: Contribution to embedded processor-based CLB BIST test time by initial external configuration and by five internal partial reconfigurations

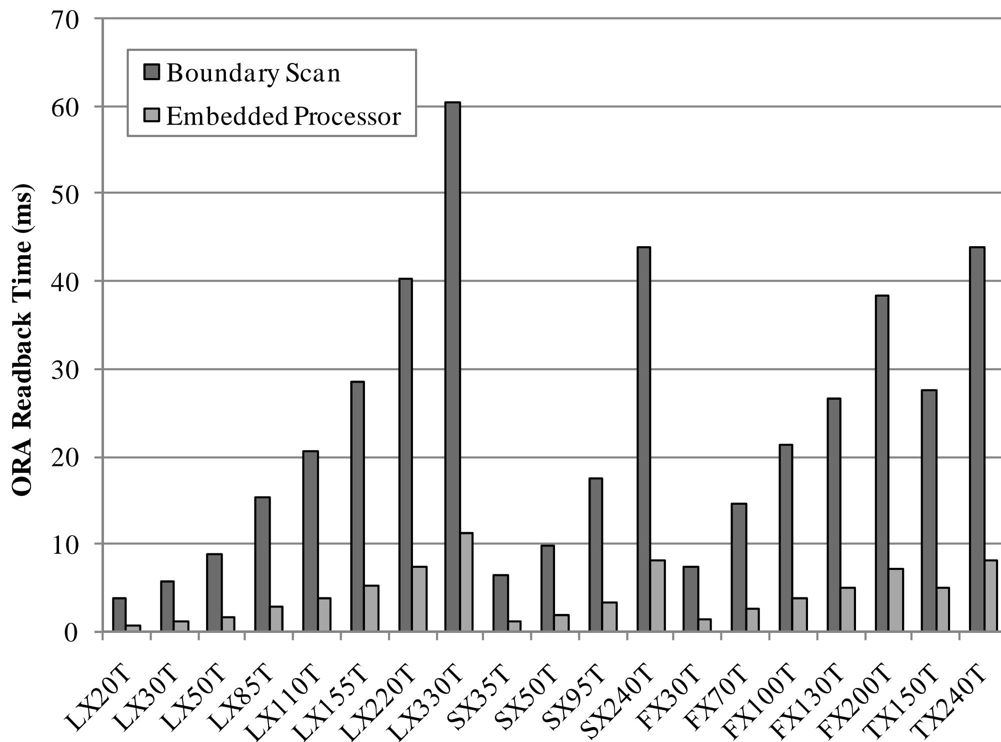


Figure 7.5: Comparison of CLB BIST ORA read back times with embedded processor-based approach and external Boundary Scan interface

7.4 Future Improvements

The overall reconfiguration times for the embedded processor-based BIST approach can be reduced by modeling a custom processor for reconfiguration and test control. When a full custom embedded fault injection approach was compared to the MicroBlaze based fault injection presented in this work, a speed-up factor of almost 12 was observed for the FSM hardware-only approach versus the general-purpose processor-based approach. However, a hardware only implementation requires a different hardware configuration for every device and BIST session, as reported in [20]. Ultimately, with custom hardware, the reconfiguration time could approach the minimum achievable test time for the 100 MHz, 32-bit SelectMAP or ICAP configuration interface. This best case timing occurs when one word is read or written on each active edge of the clock, as is the case for configuration from a dedicated memory. The best case timing for CLB BIST east or west configurations is shown in Figure 7.6 (doubling the time shown in the figure yields the total test time for all CLBs in SliceL mode). However, these times should not be directly compared to those in Figure 7.4 for the embedded processor-based approach, because Figure 7.4 assumes initial configuration from the Boundary Scan interface. Another possibility is to clock the MicroBlaze at a frequency greater than 100 MHz, using a divided clock equal to 100 MHz for the ICAP and portions of the ICAP interface logic. This will, however, require the development of a custom ICAP interface. Based on timing analysis, clock frequencies around 150 MHz are attainable in the MicroBlaze processor when constrained to one-half of the FPGA. Therefore, a speed-up of approximately 1.5 times could be achieved using a multiple clock approach.

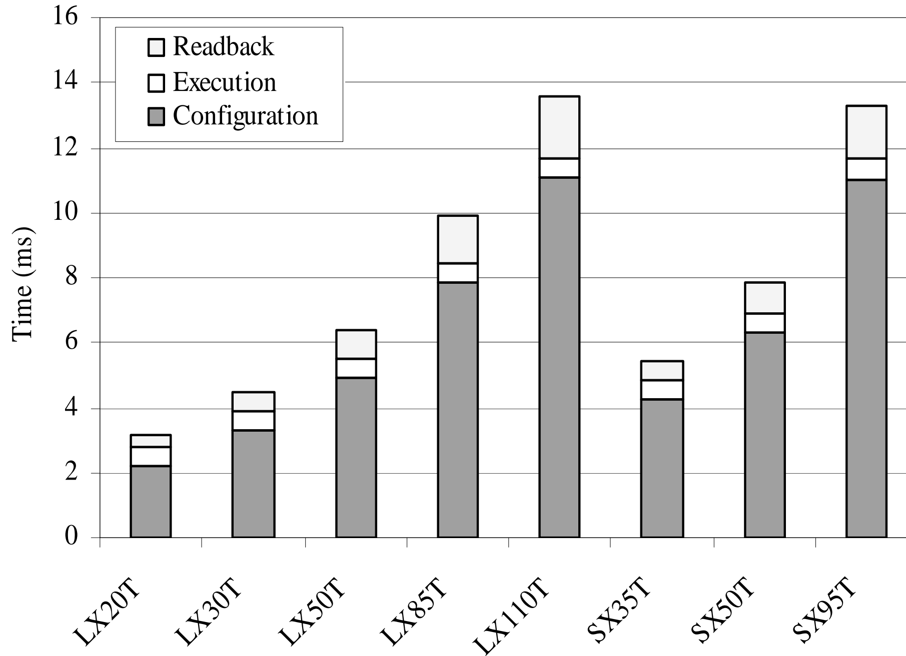


Figure 7.6: 32-bit, 100 MHz interface test time for full chip CLB west or east with one full compressed configuration and five partial reconfigurations

7.5 Other Applications

An approach similar to the embedded processor-based BIST could be applied to an external processor or microcontroller connected to the SelectMAP 32-bit configuration interface. Conceptually, the approach is similar to the approach for Atmel SoCs [3][4], except that the processor and FPGA are integrated at the board level, rather than at the chip level. The only overhead required above that for the traditional BIST approach is processor downtime for the test, additional circuit board interconnections, additional processor I/O, and a portion of the processor program memory (16,558 Bytes for one session of CLB BIST) for storing the embedded BIST software and reconfiguration data. The impact to the system could be minimized by performing tests of the FPGA as a low priority, background task, at the expense of increased test time. The approach could provide the 5.4 times speed-up of the embedded processor during reconfiguration and read back using the 32-bit, 100 MHz SelectMAP configuration interface without the penalty incurred by testing the FPGA in two sessions, one for

each half. The size of the initial download is also reduced when compared to the embedded processor-based approach due to the highly optimized structure of the BIST circuitry. Furthermore, the memory required to store the BIST configurations can be reduced at the expense of some additional program memory in the hard processor.

The embedded processor-based BIST approach for Virtex-5 FPGAs is directly applicable to Virtex-4 FPGAs [21] with some modification to the BIST specific software (including device specific subroutines such as algorithmic resource under test frame address generation) and stored configuration data. Differences between the Virtex-4 and Virtex-5 ICAP interfaces, such as byte-swapping on the Virtex-5 ICAP, are accounted for during synthesis of the MicroBlaze and associated ICAP interface circuitry based on the targeted device family. The frame address register is also arranged differently in Virtex-4 and Virtex-5, but this can be accounted for in software [22][23]. The overall test times for Virtex-4 relative to external reconfiguration closely match those results obtained in Virtex-5.

7.6 Conclusions

We have presented the results of a case study which implements the first soft-core embedded processor-based BIST approach. The approach is applicable to any FPGA with write/read access to the configuration memory from within the FPGA fabric and with sufficient configurable resources to implement both the soft-core processor and the BIST circuitry. The number of external configurations of the FPGA during any BIST session is reduced to a maximum of two (one for each half of the array) and internal reconfiguration of the resources under test are performed at the maximum allowable clock frequency and data width. Read back of ORA contents and diagnosis of faulty resources under test can be performed by the embedded soft-core processor when fault diagnosis is desired, for fault-tolerant applications for example,

providing a speed-up of 5.4 versus readback via the Boundary Scan interface. The approach can significantly decrease the overall test time in systems with a relatively slow external configuration interface, as was the case for the previous implementation of embedded processor-based BIST using a dedicated hard-core processor [4].

The soft-core processor approach was implemented in Virtex-5 FPGAs using the MicroBlaze processor for BIST reconfiguration, control of execution, fault injection, and fault diagnosis. Reconfiguration of the resources under test is achieved via the ICAP port in the FPGA fabric. When implemented in Virtex-5, the approach requires more testing time when compared with optimized external reconfiguration using compressed partial reconfiguration bitstreams. This is primarily due to the fact that the overall BIST approach has been architected for optimum configuration file compression. This includes orienting the BIST architecture with the configuration memory for maximizing the effectiveness of compressed download files with multi-frame write features, partial reconfiguration of the resources under test by maintaining constant placement and routing between test phases, and a single pass/fail indication to avoid partial configuration memory read back for BIST results. This is a testament to the advanced state of FPGA BIST techniques as well as the features and capabilities offered by FPGA manufacturers to decrease configuration times.

However, the soft-core processor approach is significantly faster than configuration with full or compressed configuration bitstreams alone. Only two downloads are required for each BIST session when the embedded processor-based approach is used, compared to six configurations for CLB east/west tests and nine for SerDes tests for example. BIST control, execution and fault diagnosis implemented in the embedded processor eliminate the need for complex external test equipment for manufacturing testing and intelligent external BIST

controllers for in-system testing and diagnosis in fault-tolerant applications. The architecture is applicable to any BIST for Virtex-4 and Virtex-5 FPGAs without modification of the embedded processor hardware; only the MicroBlaze program memory contents need to be changed.

7.7 Acknowledgements

The contents of this chapter are accepted for publication in *Proc. IEEE Southeastern Symposium on System Theory*, 2010. Prof. Charles Stroud is a co-author on the paper. A majority of the actual research and the writing of the published paper represents the efforts of the primary student author and not collaborators, and the research represents work performed while in the graduate program at Auburn University.

7.8 References

[1] B. Dutton and C. Stroud, "Soft-core Embedded Processor Based Built-In Self-Test of FPGAs," *Proc. IEEE Int. Symp. On Defect and Fault Tolerence in VLSI Sys.*, pp. 29-37, 2009.

[2] *Virtex-5 FPGA User Guide*, UG190(v4.2), Xilinx, 2008.

[3] L-T Wang, C. Stroud, and N. Touba, *System-on-Chip Test Architectures*, San Francisco, CA: Morgan Kaufmann, 2007.

[4] S. Toutounchi and A. Lai, "FPGA Test Coverage," *Proc. IEEE Int. Test Conf.*, pp. 1248-1257, 2003.

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

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

[7] C. Stroud and J. Bailey, "Bridging Fault Extraction from Physical Design Data for Manufacturing Test Development, *Proc. IEEE Int. Test Conf.*, pp. 760-769, 2000.

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

[9] C. Stroud, S. Garimella and J. Sunwoo, "On-Chip BIST-Based Diagnosis of Embedded Programmable Logic Cores in System-On-Chip Devices," *Proc. ISCA Int. Conf. on Computers and Their Applications*, pp. 308-313, 2005.

[10] J Sunwoo and C. Stroud, "Built-In Self-Test of Configurable Cores in SoCs Using Embedded Processor Dynamic Reconfiguration," *Proc. Int. SoC Design Conf.*, pp. 174-177, 2005.

[11] S. Vemula and C. Stroud, Built-In Self-Test for Programmable I/O Buffers in FPGAs and SOCs, *Proc. IEEE Southeastern Symp. on System Theory*, pp. 534-538, 2006.

[12] S. Dhingra, S. Garimella, A. Newalkar and C. Stroud, "Built-In Self-Test for Virtex and Spartan II FPGAs Using Partial Reconfiguration," Proc. IEEE North Atlantic Test Workshop, pp. 7-14, 2005.

[13] D. Milton, S. Dhingra and C. Stroud, "Embedded Processor Based Built-In Self-Test and Diagnosis of Logic and Memory Resources in FPGAs," *Proc. Int. Conf. on Embedded Systems and Applications*, pp 87-93, 2006.

[14] B. Garrison, D. Milton, and C. Stroud, "Built-In Self-Test for Memory Resources in Virtex-4 FPGAs," *Proc. ISCA Int. Conf. on Computers and Their Apps.*, pp. 63-68, 2009.

[15] M. Pulukuri and C. Stroud, "Built-In Self-Test of Digital Signal Processors in Virtex-4 FPGAs," *Proc. IEEE Southeastern Symp. on System Theory*, pp. 34-38, 2009.

[16] J. Yao, B. Dixon, C. Stroud and V. Nelson, "Built-In Self-Test of Programmable Interconnect in Virtex-4 FPGAs," *Proc. IEEE Southeastern Symp, on System Theory*, pp. 29-33, 2009.

[17] B. Dutton and C. Stroud, "Built-In Self-Test of Configurable Logic Blocks in Virtex-5 FPGAs," *Proc. IEEE Southeastern Symp. on System Theory*, pp. 230-234, 2009.

[18] B. Dutton and C. Stroud, "Built-In Self-Test of Programmable Input/Output Tiles in Virtex-5 FPGAs," *Proc. IEEE Southeastern Symp. on System Theory*, pp. 235-239, 2009.

[19] *MicroBlaze Processor Reference Guide*, UG081(v.9.0), Xilinx, 2008.

[20] B. Dutton and C. Stroud, "Embedded Processor Based Fault Injection and SEU Emulation for FPGAs," *Proc. Int. Conf. on Emb. Systems and Apps.*, pp. 183-189, 2009.

[21] *Virtex-4 FPGA User Guide*, UG070 (v2.5), Xilinx, 2008.

[22] *Virtex-4 FPGA Configuration User Guide*, UG071(v1.1), Xilinx, 2008.

[23] *Virtex-5 FPGA Configuration User Guide*, UG191(v2.7), Xilinx, 2008.

[24] R. Rajsuman, "Testing a System-On-Chip with Embedded Microprocessor," *Proc. IEEE Int. Test Conf.*, pp. 499-508, 1999.

Chapter Eight. On-line Single Event Upset Detection and Correction in Field Programmable Gate Array Configuration Memories

Larger field programmable gate array (FPGA) configuration memories and shrinking design rules have raised concerns about single event upsets (SEUs), especially for high-reliability, high-availability systems that use FPGAs. We present a design for the on-line detection and correction of SEUs in the configuration memory of Xilinx Virtex-4 and Virtex-5 FPGAs. The design corrects all single-bit errors and detects all double-bit errors in the configuration memory at maximum speed and with minimal overhead and power dissipation. A method for SEU emulation in the configuration memory of FPGAs is presented which enables the experimental verification of the approach. The results of SEU emulation in Xilinx FPGAs are discussed.

8.1 Introduction

The increased use of field programmable gate arrays (FPGAs) for implementing digital logic applications over the past two decades has been accompanied by increased concern about radiation effects, and, in particular, the effects of single event upsets (SEUs). In addition to memory elements such as flip-flops, look-up tables (LUTs), and random access memory (RAM) cores, FPGAs contain a large static random access memory (SRAM), referred to as the configuration memory, which establishes the overall system application performed by the FPGA. An SEU induced bit-flip in the SRAM configuration memory can therefore alter the functionality of the FPGA. This, coupled with the large size of the configuration memory, makes SEUs of significantly more concern in FPGAs than in traditional application specific integrated circuits

(ASICs). In Xilinx Virtex-5 FPGAs, for example, the configuration memory alone represents greater than 99% of all memory elements in a given device, as summarized in Table 8.1, where the LX30T represents one of the smallest FPGAs in the Virtex-5 family and the LX330T represents one of the largest [26][27].

Table 8.1: Memory resources in two Virtex-5 FPGAs

	Number of Memory Elements			
Memory Type	LX30T	% of Total	LX330T	% of Total
Flip-Flops	19,200	0.2%	207,360	0.25%
LUT RAM Bits	327,680	3.5%	3,502,080	4.22%
Block RAM Bits	1,327,104	14.1%	11,943,936	14.41%
Configuration Bits	9,362,432	99.8%	82,687,488	99.75%
Total	9,381,632	100.00%	82,894,848	100.00%

Finding an accurate measurement of the susceptibility of SRAM configuration memories to SEUs has been the focus of much research, including that in [15], [18], [24] and [30]. Accelerator testing conducted with Xilinx 4000 series FPGAs indicates the SEU frequency increased by a factor of 4.74 when design rules decreased from 600nm to 350nm with a corresponding reduction in power supply voltage from 5V to 3.3V [18]. On the other hand, 90nm Xilinx Virtex-4 FPGAs are reported to have SEU FIT (failures in 10^9 hours) rates of 246 per 10^6 bits of configuration memory, while 65nm Virtex-5 FPGAs have a lower SEU FIT rate of 151 per 10^6 bits (adjusted for sea-level in New York, NY) [6][15]. The FIT rate per Mb of configuration memory in Xilinx FPGAs has actually been decreasing since the Virtex-II series in the year 2000. This reduction in SEU FIT rate by a factor of about 3.5 from Virtex-II to Virtex-5, despite drastic reductions in feature size and supply voltage, indicates that Xilinx is incorporating architecture dependent SEU hardening techniques in the design of the configuration memory. This trend can be seen in Figure 8.1, where the SEU rate for each Xilinx FPGA family is plotted along with the initial release year and minimum feature size. According

to [16], since 2002 Xilinx has designed the configuration memory to be more robust in an attempt to reduce soft failure rates even as the size and density of the FPGA grows. That this attempt has been successful is supported by the fact that the FIT rates reported for Xilinx FPGAs are low when compared to typical SRAMs [18]. A more robust SRAM design is possible because the SRAM configuration memory remains static a majority of the time, in contrast to typical SRAM memories which are designed to be as small and as fast as possible [5][18].

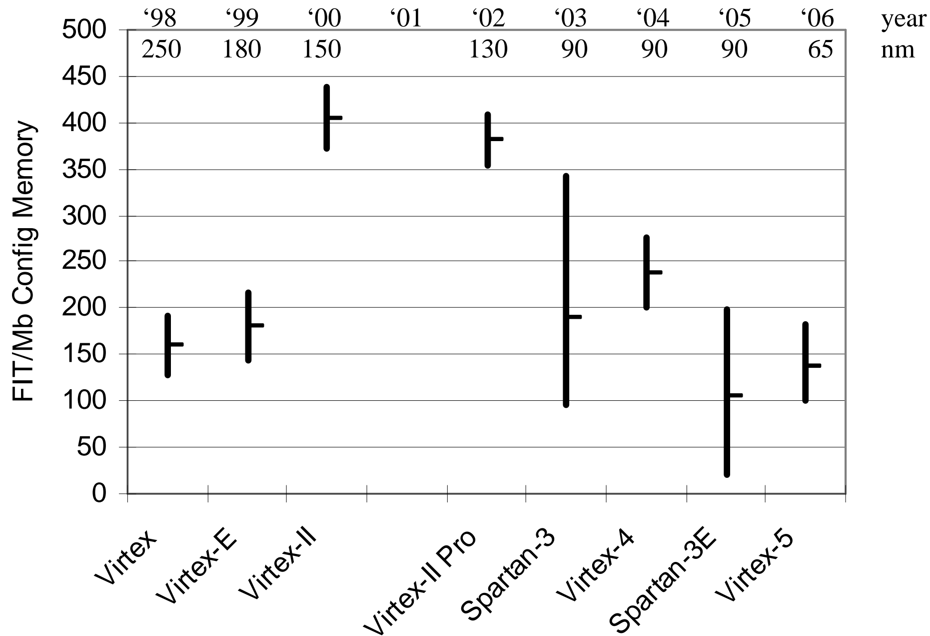


Figure 8.1: FIT rate (corrected for sea-level New York, NY) versus Xilinx device family, initial release year, and minimum feature size [6] where the center line represents the nominal value and the span of the line represents the upper and lower 95% confidence levels

However, even the relatively low FIT rates of Xilinx FPGAs can become problematic when considering the design of high reliability, high availability systems or systems which operate at high altitude or in space. The largest commercially available Xilinx FPGAs currently have configuration memories with more than 80 Mb [27] and in the next generation of devices the largest available FPGAs will include configuration memories of over 160 Mb in size [28]. For the 80 Mb Virtex-5 device, the FIT rate per device is 10,960 failures in 1 billion hours, or

mean time between failure (MTBF) of (114,155 years/10,960 FIT) = 10.4 years at sea level. At the 95% confidence level, the FIT rate is between 100-183, or MTBF between 7.8-14.3 years. However, it should be noted that an SEU in the configuration memory does not always correspond to a failure of the system. It is estimated that only between 10% [21] and 40% [24] of the configuration bits used in any given design actually affect the design functionality. Therefore, for a more accurate estimate of the MTBF the sensitivity of the design based on analysis of "care" versus "don't care" configuration bits should be taken into account. Nevertheless, for SRAM FPGAs to be adopted for critical avionics and space applications where little or no risk is acceptable, an effective SEU mitigation plan must be implemented. In addition, systems operating in high-radiation environments may require an SEU mitigation plan even if some risk is tolerable. As an example of the variance of SEU occurrence with altitude, consider that the neutron flux density increases by a factor of 383 at the typical commercial flight cruising altitude of 36,000 ft (relative to sea-level New York, NY) [9].

Techniques for hardening digital circuits against SEUs can be categorized as architecture dependent or architecture independent. An architecture dependent technique is one that requires a modification to the physical design of an integrated circuit; for example, high reliability systems can employ hardware redundancy in latches [3]. In FPGAs, however, architecture dependent SEU hardening techniques are only available if implemented by the FPGA manufacturer. Therefore, for a typical SRAM FPGA, any SEU hardening implemented by the user must be one that is architecture independent. One widely known architecture independent technique used in FPGAs is triple modular redundancy (TMR). The TMR approach triplicates all of the user logic and adds majority voters at the inputs to all flip-flops and on all primary outputs. By eliminating all single point failures, the design can be guaranteed to tolerate an SEU

in any of the three circuit copies. However, the overhead for a TMR approach can be prohibitive because it is greater than 200%. Therefore, to implement a TMR approach, the required size of the FPGA (in terms of resources) would necessarily be more than three times the size of the original, non-TMR design. TMR also consumes more power (approximately three times as much) and incurs a performance penalty. The implementation of TMR for designs in Xilinx FPGAs can be entirely automated using the Xilinx TMR Tool, which guarantees full SEU and single-event transient (SET) immunity [29]. However, without some additional form of configuration memory scrubbing, the accumulation of multiple SEUs over time can cause system failure even in designs with full TMR [23][29].

Another architecture independent method, configuration memory scrubbing, periodically refreshes the contents of the configuration memory without attempting to determine if an SEU has occurred. Power-cycling is essentially the simplest form of configuration memory scrubbing because the entire configuration memory can be refreshed each time the FPGA is power-cycled if the FPGA is set in master configuration mode [1]. A more intelligent approach is to externally read back words of configuration memory contents, comparing each word to a copy in a "golden" configuration bitstream. This approach has the advantage of being able to detect any number of SEUs in the configuration memory (when compared to error correcting codes). Any mismatch between the "golden" copy and the configuration memory contents should cause the erroneous configuration memory to be overwritten by the "golden" configuration data. However, both approaches require a radiation hardened external configuration management unit (microprocessor or ASIC) and a radiation hardened "golden" copy of the configuration data. The second approach also doubles the required amount of memory, because both the "golden"

bitstream and a mask file, which is used to mask configuration bits which are subject to change during normal system operation, must be stored in the system [1].

Some FPGAs, including Virtex-4 and Virtex-5, incorporate a Hamming error correction code (ECC) in the configuration memory. The ECC, in conjunction with some additional user-accessible dedicated logic can be used to detect SEUs in the configuration memory. With additional user-defined circuitry in the FPGA core, erroneous configuration memory bits that result from SEUs can be not only detected, but also corrected [12]. It is this method that is the focus of this chapter where we present an efficient SEU correction circuit that works in combination with existing SEU detection mechanisms in Virtex-4 and Virtex-5 FPGAs to correct SEUs in the FPGA configuration memory. This circuit can be synthesized and incorporated with any user-defined digital application in any Virtex-4 or Virtex-5 FPGA for detection and correction of SEUs during normal on-line system operation. We begin with an overview of existing SEU detection mechanisms in Section 8.2 along with an overview of previous work in on-line SEU detection and correction in Virtex-4 FPGAs. The operation and architecture of the proposed SEU detection and correction circuit are presented in Sections 8.3 and 8.4, respectively. Experimental results and analysis from the actual implementation of the SEU detection and correction circuit in Virtex-4 and Virtex-5 FPGAs are presented in Sections 8.5 and 8.6 along with a comparison to prior work. The chapter concludes with a summary in Section 8.7.

8.2 Background

Like any other RAM, the configuration memory of an FPGA is partitioned into words, also called frames, which represent the smallest addressable unit of the memory for write and read operations. Virtex-4 and Virtex-5 frames consist of 1,312 bits [25][26]. Each frame

includes a 12-bit field of eleven Hamming bits and an overall parity bit for the frame data. The eleven Hamming bits provide the potential for single error correction (SEC), and the overall parity bit enables double error detection (DED) over the frame data. The parity and Hamming bits are generated external to the FPGA by the configuration bitstream generation software and are subsequently downloaded with the application specific configuration data (an internal CRC check verifies the integrity of the downloaded data). However, system memory data subject to change during the operation of the FPGA, such as the contents of block RAMs and look-up tables (LUTs) used as distributed RAMs, are not covered by the parity and Hamming bits [4].

Virtex-4 and Virtex-5 provide a specialized core, called Frame ECC, for detection and identification of single and double-bit errors in the frame data [25][26]. For each frame read from the configuration memory, the Frame ECC module calculates the Hamming bits as well as the overall parity for the frame data, and compares these bits with the Hamming bits and parity for that frame stored in the configuration memory. Based on this comparison, the Frame ECC module produces indications for no error, single-bit error, and double-bit error conditions in addition to a syndrome indicating the location of single-bit errors. The error conditions for the Frame ECC core are summarized in Table 8.2. System memory contents—block RAMs and LUT RAMs, for example—are masked from the internal parity and Hamming calculation by the Frame ECC.

Table 8.2: Frame ECC error codes [25][26]

Error Type	**Condition (*syndromevalid* = 1)**
No bit error	Hamming match, no parity error
1-bit correctable error (SEC)	Hamming mismatch, parity error
2-bit error detection (DED)	Hamming mismatch, no parity error

The Frame ECC function is performed each time a frame is read from the external serial Boundary Scan interface or parallel SelectMAP configuration interface [25][26]. In addition to these external configuration interfaces, Virtex-4 and Virtex-5 include a 32-bit internal configuration access port (ICAP), illustrated in Figure 8.2, that provides write/read access to/from the configuration memory from within the FPGA core. As is the case with the external interfaces, the Frame ECC function is performed each time a frame is read via the ICAP. Because the Frame ECC does not provide circuitry to perform error correction, some additional logic must be implemented in the FPGA fabric that uses the ICAP and Frame ECC modules to cycle through all frames of the configuration memory to detect SEUs and to correct those SEUs. Virtex-5 FPGAs also include dedicated circuitry in the FPGA that can automatically detect SEUs using built-in cyclic redundancy check (CRC) circuitry [26]. When Readback CRC is enabled (by setting the POST_CRC configuration option to ENABLE), the contents of the configuration memory are continuously read back in the background of the user design operation to calculate and check the CRC of the configuration memory contents. An SEU anywhere in the configuration memory will cause the re-calculated CRC to disagree with the stored CRC. The mismatch is signaled by asserting the CRC Error output of the Frame ECC (only present in Virtex-5 and not shown in Figure 8.2). Optionally, the external INIT_B output pin of the FPGA may also be driven low when the error is detected [26]. The Readback CRC will begin to run automatically upon a successful configuration of the FPGA and will continue to run as long as no configuration interfaces are in use; a configuration interface is considered to be in use after the synchronize (SYNC) command is decoded and until the de-synchronize (DESYNC) command is decoded [26]. Similar background CRC read back circuitry has been incorporated in recent Altera [21] and Lattice [14] FPGAs to support SEU detection.

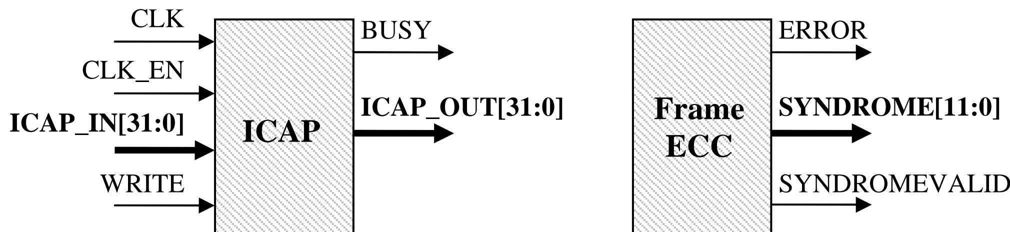


Figure 8.2: Frame ECC and ICAP primitives

An implementation of internal SEU detection and correction using the Frame ECC and ICAP logic in Virtex-4 devices was reported in [12]. The design uses an 8-bit PicoBlaze [19] soft-core processor with additional circuitry and RAM in the FPGA fabric for interfacing to the ICAP to read and write the configuration memory. The design can operate in a detection only mode, or can detect and correct single-bit errors. The design was later implemented in triple modular redundancy (TMR) [10]. While both [10] and [12] are applicable only to Virtex-4, the approach in [12] was recently extended in [5] to support Virtex-5 FPGAs.

8.3 Operation of SEU Detect and Correct

Our SEU detect and correct circuit, or *SEU controller* as it is referred to in this chapter, is designed to be integrated into any existing VHDL-based user design with minimal effort. At the top level, there are only two inputs—*clock* and *reset*—and one output— *error*. The VHDL component declaration for the SEU controller is given in Figure 8.3.

```
component seu_controller is
generic(device : string(1 to 6));
port(       rst : in std_logic;
          clock : in std_logic;
          error : out std_logic);
end component seu_controller;
```

Figure 8.3: SEU controller VHDL component declaration

The generic *device* is a text string that specifies the device in which the SEU controller will be implemented, such as “LX330T” for example. All Virtex-4 and Virtex-5 devices are

supported such that only this generic need be specified by the user to indicate the target FPGA for synthesis. The *error* output is asserted when the first multiple-bit error is detected, and should trigger a reconfiguration of the FPGA from a reliable external memory since multiple bit errors cannot be corrected by the SEU controller. The *clock* input is directly connected to the ICAP clock and the SEU controller. It is limited by the maximum ICAP clock frequency of 100 MHz, but can operate at any frequency below 100 MHz. In Virtex-5 devices, the ICAP and SEU controller clock can be supplied by an internal 50 MHz oscillator [26]. The synchronous active high *reset* input forces the SEU controller into an inactive state, releasing the configuration interface for use by other applications. Asserting the *reset* input also resets the frame address to the first frame of configuration memory and clears the *error* output. When *reset* is released, the SEU controller will resume normal operation from the first frame of the configuration memory on the next rising edge of *clock*. The *reset* input may be tied to logic 0 for free-running SEU detection and correction in user designs that do not require access to the configuration memory during normal system operation. The operation of the SEU controller consists of the following steps:

1. A 1312-bit frame of configuration memory is read through the ICAP as forty-one 32-bit words and the frame data is stored in a block RAM.

2. If an error is indicated by the outputs of the Frame ECC primitive, the type of error is determined as shown in Table 8.2. If the error indicates a double-bit error, the *error* output of the SEU controller is latched high and read back continues with the next frame of configuration memory. If a single-bit error is indicated, the location of the bit is determined from the syndrome and the erroneous bit is corrected (*i.e.* inverted) in the frame data stored in the block RAM.

3. If a single-bit error was indicated in Step 2, the repaired frame is now written back into the configuration memory at the same frame address from which it was read.

4. If a single-bit error was indicated in Step 2, read back resumes with the first frame in the configuration column containing the newly repaired frame.

5. When a configuration column has been completely read and repaired (as determined by no single-bit error indications for any frames in for that configuration column), the SEU controller advances to the next configuration row/column in the array and repeats the process starting at Step 1.

This SEU controller behavior is summarized by the pseudocode of Figure 8.4.

```
Load starting frame address
while (reset == 0) {
     Read single frame from configuration memory
     Read Frame ECC outputs
     if (single bit error is detected) {
          Translate syndrome to bit index in frame
          Read erroneous bit
          Write inverted (corrected) bit to same location
          Write frame back to configuration memory
          break
     }
     else if (double bit error is detected) {
          Assert ERROR output
     }
     Increment Frame Address
}
```

Figure 8.4: SEU controller behavioral pseudocode

In Virtex-5 devices, the SEU controller may utilize the Read Back CRC feature of the Frame ECC module for the initial detection of an SEU with a small modification to the design. By enabling the Read Back CRC (in the design constraints file) and using the complement of the

CRC Error output of the Frame ECC circuit as the *reset* input to the SEU controller, the SEU controller will remain idle (held in active high reset) with the CRC Read Back circuit operating in the background (at the frequency of the ICAP input clock [26]). When a CRC mismatch is detected, the CRC Error output of the Frame ECC circuit is asserted, de-asserting the *reset* input to the SEU controller. The SEU controller will begin normal operation, cycling through the configuration memory detecting and correcting all single-bit errors. However, after the last frame of configuration memory is reached, the SEU controller will return to the reset state and wait for a falling edge on the *reset* input before resuming operation. By entering the reset state and releasing the ICAP configuration interface via a DESYNC command, the internal CRC Read Back will resume. This approach has the disadvantage of doubling the cycle time in the worst case since both the CRC Read Back circuit and SEU controller may require a complete cycle to detect and then repair the SEU. As observed in [5], however, this approach may offer some additional immunity to SEUs in the detection phase because the CRC Read Back circuit is implemented as dedicated logic at the physical circuit level, as opposed to the SEU controller, which is implemented in configurable resources. The INIT_B signal could be used to externally verify the correction of the SEU by ensuring the INIT_B output pin of the FPGA does not remain low longer than a predetermined time period (approximately three complete scan cycles of the FPGA configuration memory). If, however, the INIT_B remains low or the error output of the Frame ECC is asserted, the error is not repaired and the configuration memory should be refreshed from a radiation hardened "golden" copy.

8.4 SEU Detect and Correct Architecture

Our SEU controller is implemented entirely in configurable logic blocks (CLBs) and one 18 Kb block RAM in the FPGA fabric. It is constructed primarily around the ICAP and Frame

ECC primitives [25][26]. The operation of the SEU controller, described in the previous section, is managed with a finite state machine (FSM) implemented in CLB logic slices. The FSM initiates reads and writes to the FPGA internal configuration memory and control registers via the 32-bit ICAP interface. A set of sixty-four 32-bit instructions are stored in a 32×64-bit read-only memory (ROM) formed in 32 LUTs (6-inputs each) in Virtex-5 and 128 LUTs (4-inputs each) in Virtex-4. The 32×64-bit LUT ROM is addressed by a counter that is enabled by combinational logic from the FSM current state. The FSM also generates the frame address for reads and writes of the configuration memory. All reads from and writes to the configuration memory are 32-bits. The logic for the frame address counter is device dependent since every device has different numbers of rows and/or columns. Furthermore, the arrangement of different types of columns (*e.g.* CLB, DSP, RAM, etc.) can vary depending on the device. The generic *device* (shown in Figure 8.3) is used to determine and synthesize the correct frame address logic for the target device.

The central component of the SEU controller architecture is the dual-port block RAM (at least two columns of 18 Kb block RAMs are included in every Virtex-4 and Virtex-5 device). A single block RAM is used to store each frame as it is read from the configuration memory. The A port of the block RAM is configured for 32-bit read/write access, and the B port is configured for 1-bit read/write access, as illustrated in Figure 8.5. The data inputs of the A port are connected directly to the outputs of the ICAP, and the A port data outputs are connected to the ICAP inputs via a 32-bit 2-to-1 multiplexer.

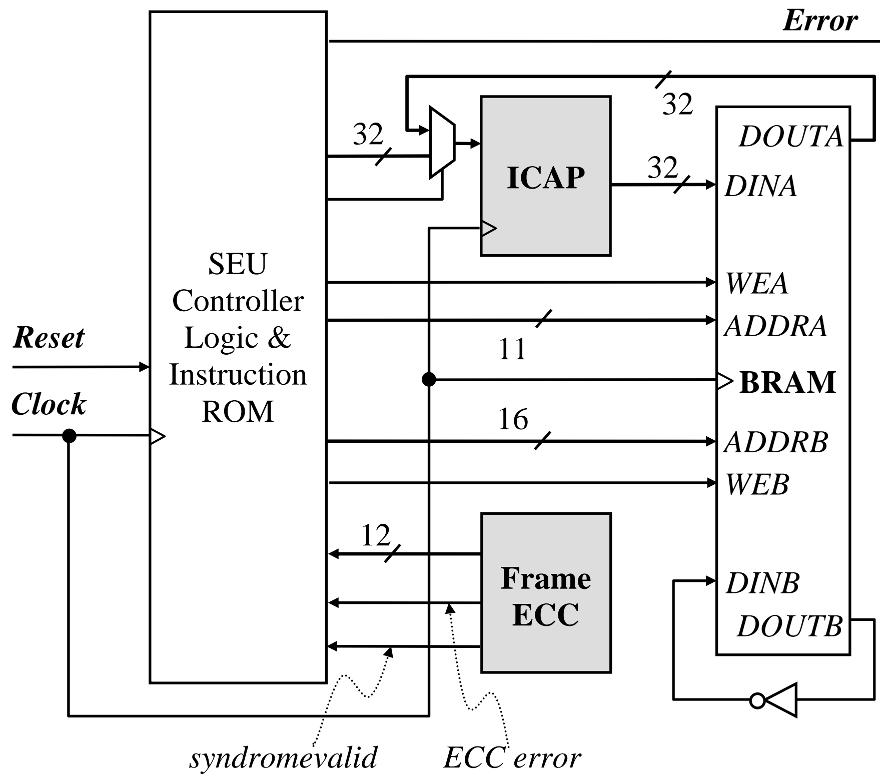


Figure 8.5: SEU controller block diagram

The A port address inputs are controlled by a counter in the FSM. Every frame that is read from the ICAP is stored in the first forty-one 32-bit words of the block RAM. Single-bit errors are corrected via the 1-bit B port interface. The B port address inputs are connected to combinational logic which provides the bit offset of the bit in error based on the Frame ECC *syndrome* outputs. The 1-bit B port data output is inverted connected to the 1-bit B port input. The B port write enable is controlled by combinational logic from the *syndromevalid* and *ECC error* Frame ECC outputs in conjunction with the FSM. The location of single-bit errors within the frame is indicated by the *syndrome*[10:0] outputs of the Frame ECC primitive, however some additional combinational computational logic is required to determine the exact bit-offset of the error within the configuration frame. An equation for determining the bit-offset of the error in the range 0-1311 is given by:

$$\text{offset} = \{S[10:5] - 6'd22 - S[10], S[5:0]\} \quad (8.1)$$

where S[10:0] are the Frame ECC *syndrome* outputs [25][26]. Otherwise, if the binary value of *syndrome*[10:0] is 0 or a power of 2, then the error is located in one of the Hamming bits, in which case the location of the bit error is determined as shown in Table 8.3. The output of the syndrome combinational logic is tied to the B port address inputs. In this manner, the erroneous bit, as indicated by *syndrome*[11:0], is inverted when the block RAM B port write enable is asserted. The repaired frame is then written back into the configuration memory via the A port 32-bit output to the ICAP.

Table 8.3: Hamming bit error diagnosis [25][26]

***syndrome*[11:0]**	**offset**	***syndrome*[11:0]**	**offset**
100000000001	640	100001000000	646
100000000010	641	100010000000	647
100000000100	642	100100000000	648
100000001000	643	101000000000	649
100000010000	644	110000000000	650
100000100000	645	100000000000	651

A rare, but potentially problematic situation can arise when an odd number of bit errors occur in a single frame of configuration memory. These errors will cause both a syndrome mismatch and overall parity mismatch, which aliases as a correctable single-bit error (refer to Table 8.2). However, in this case, the syndrome outputs do not necessarily indicate the location of any of the actual errors, and can erroneously point anywhere in the range 0 to $2^{11}-1$ (2047). Since the actual frame data only exists in the range 0 to 1311, the following two scenarios are possible.

First, the odd-multiple bit error aliases as a single-bit error with the syndrome outputs pointing in the valid range of the frame data 0 to 1311. In response to the single-bit error

indication, the SEU controller will invert the frame-bit pointed to by the syndrome, which may satisfy the Hamming code by creating a valid distance code word, and the modified frame will be written back into the configuration memory. The SEU controller will resume read back at the start of the configuration column containing the still damaged frame. When the erroneous frame, now containing an even number of multiple errors, is read, the valid code word will cause a Hamming code match and an overall parity-bit match such that a "no bit error" indication is obtained. However, by incorporating the CRC Read Back mechanism with the SEU controller, as described in Section 8.3, this multiple bit error can be detected because the CRC will continue to indicate a CRC Error with the SEU controller indicating no error.

In the second scenario, when the frame containing an odd number of errors greater than one is read, the syndrome indicates an error bit location in the range from 1312 to 2047. This range, while a valid address in the larger block RAM, lies outside of the range of valid frame data. Therefore, if events are allowed to proceed as in the first scenario, unmodified frame data would be written back into the configuration memory, effectively creating an infinite loop, since the same frame would be continually read from and written to the configuration memory without modification. Our solution is to include a greater-than comparator in the SEU controller which detects when the syndrome points outside of the range of valid frame data (0 to 1311). When this condition occurs, the SEU controller ignores the syndrome and asserts the *error* output, indicating the existence of a multiple-bit error and that the FPGA configuration bitstream data should be reloaded from a reliable external memory.

8.5 Implementation Results

The greatest benefit of our SEU controller when compared to other approaches is the relatively high speed at which errors are detected and corrected. SEUs should be corrected with a minimum amount of latency so that errors in the programming of the user logic persist for the shortest possible period of time. Figure 8.6 shows the time required for one full cycle of single-bit error correction and double-bit error detection in Virtex-4 devices for the Xilinx SEU controller described in [12] and our SEU controller, where a cycle is defined as the time to perform the operation over every configuration memory frame in the device, excluding frames containing block RAM contents. The cycle time also corresponds to the maximum amount of time that one SEU can persist in the configuration memory.

The Xilinx Virtex-4 SEU controller can operate in two modes: single and double-bit error detection only mode, and single-bit error correction and double-bit error detection mode [12]. As shown in Figure 8.6, the Xilinx "detect only" cycle time is nearly identical to our detect *and* correct mode. However, when single-bit error correction is enabled, the total cycle time for the Xilinx Virtex-4 SEU controller increases to about 20 times that of our normal detect *and* correct cycle time. On average, our SEU controller reduces the total cycle time for SEC and DED with respect to the Xilinx SEU controller by 94.7%. Figure 8.7 shows the total cycle time for our SEU controller in Virtex-5 devices. The cycle time is increased by an average of 17 μs for each SEU detected and corrected. The repair time for one frame is negligible. However, the cycle time would double if there were one SEU present in every configuration column.

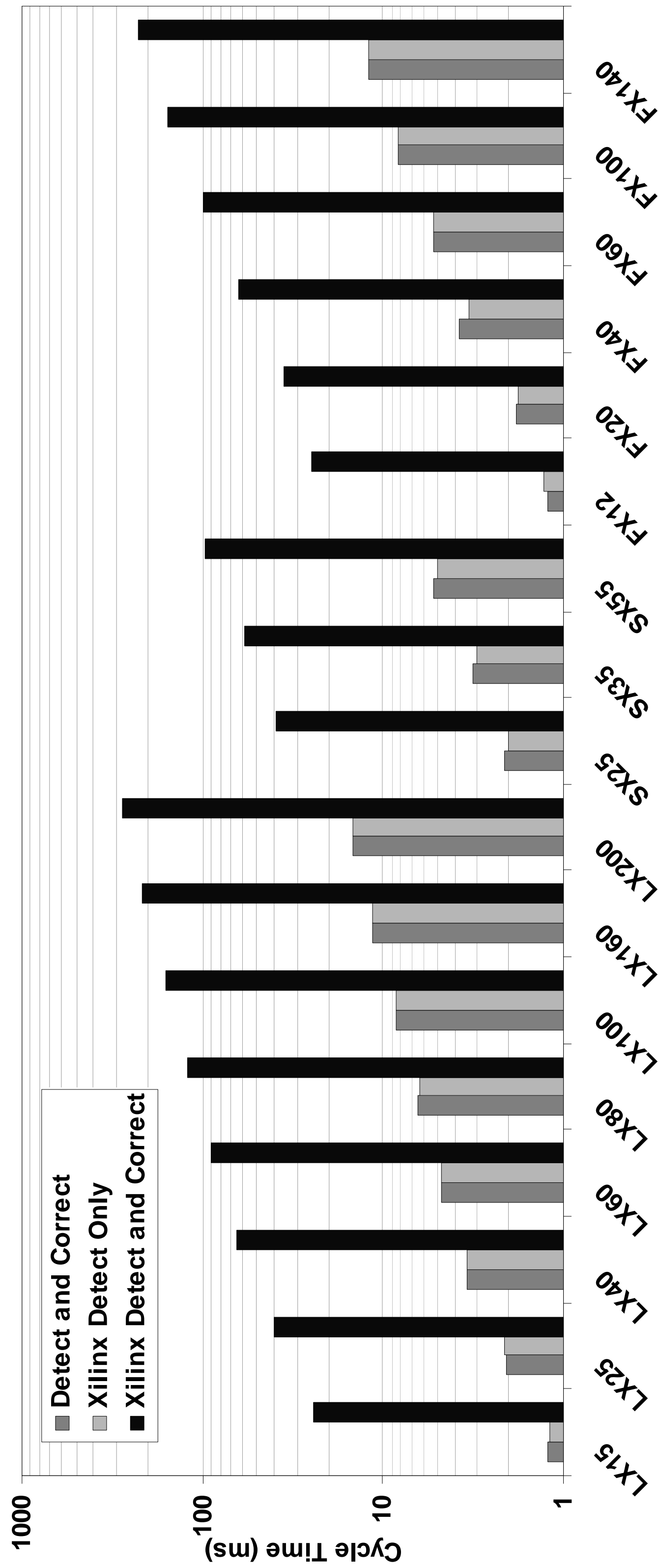


Figure 8.6: SEU controller LOG cycle time vs. Virtex-4 device

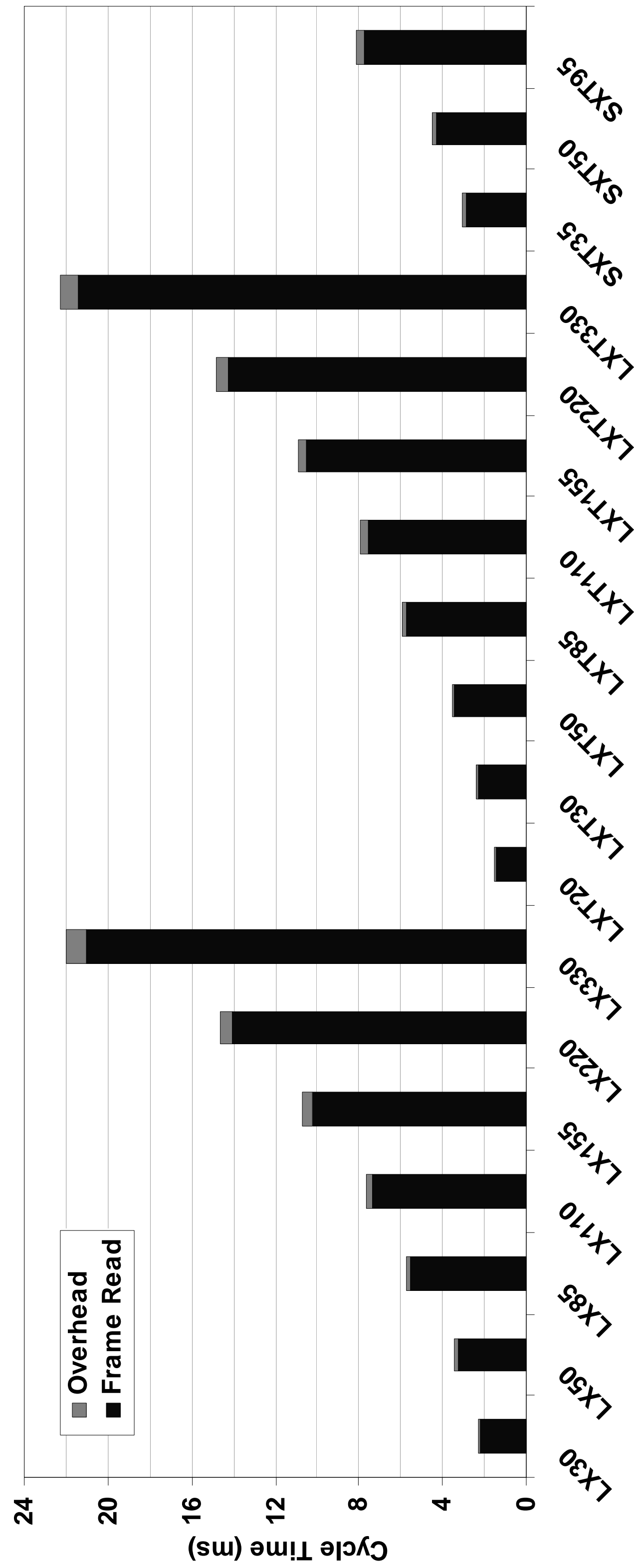


Figure 8.7: SEU controller cycle time vs. Virtex-5 device

To increase the reliability of the Xilinx SEU controller, the authors of [10] used the Xilinx TMR Tool [29] to implement the Xilinx Virtex-4 SEU controller with full TMR. However, as the results in [10] show, this approach may be impractical for some applications because of its high area overhead. A comparison of the device utilization for the Xilinx SEU controller [12], the Xilinx SEU controller with TMR [10], and our approach implemented in Virtex-4 is summarized in Table 8.4. While the Xilinx approach uses 23 fewer slices, we use one less block RAM and complete each cycle of the configuration memory an average of 20 times faster. The Xilinx Virtex-4 SEU controller with TMR utilizes 1,308 logic slices and 6 block RAMs [10] – a 770% increase in area versus the non-TMR SEU controller.

Table 8.4: SEU controller resource utilization in Virtex-4 devices

Resource	**Xilinx [12]**	**Xilinx TMR [10]**	**SEU Controller**
# Slices	149	1308	182
# Block RAMs (18 Kb)	2	6	1
Avg Cycle (ms)	105.5	105.5	5.603
# Lines VHDL	3656	--	1051

A comparison of our SEU controller with the recently proposed Xilinx Virtex-5 SEU controller [5] is given in Table 8.5. Our approach uses one less block RAM and 30 fewer slices. The cycle time for the Xilinx Virtex-5 SEU controller approach was not reported. However, due to the similarity of the Virtex-4 [12] and Virtex-5 [5] SEU controller architectures, our Virtetx-5 SEU controller is likely to have a speed-up factor similar to that observed in Virtex-4.

Table 8.5: SEU controller resource utilization in Virtex-5 devices

Resource	**Xilinx [5]**	**SEU Controller**
# Slices	95	65
# Block RAMs (18 Kb)	2	1
Average Cycle Time (ms)	--	9.338
# Lines VHDL	2625	945

Our SEU controller could also be implemented using the Xilinx TMR Tool to mitigate the risk of failure due to an SEU, as was done in [10] for the Xilinx Virtex-4 SEU controller TMR design. This approach would essentially allow two error-free SEU controllers to correct an SEU affecting the third SEU controller. However, the configurable routing resources surrounding the ICAP and Frame ECC cores could still be vulnerable to SEUs since these modules and their interfaces cannot be replicated.

8.6 Experimental Results

Our SEU controller has been synthesized for all Virtex-4 and Virtex-5 FPGAs. Furthermore, the SEU controller has been downloaded and verified on Virtex-4 FX12, SX35, and LX60 devices as well as Virtex-5 LX30T, LX50T, SX35T, SX50T, FX30T and FX70T devices. The number of utilized CLB logic slices has been observed to vary by ±3 slices in both Virtex-4 and Virtex-5 devices depending on the device and the area optimization used with the place and route software. During synthesis, the SEU controller logic and block RAM may be constrained to any area of the FPGA or may be left unconstrained for automatic placement with the user's system function. The routed SEU controller in a Virtex-5 LX30T device is shown in Figure 8.8 where its location was constrained to the area shown. The dynamic power dissipation of the SEU controller was measured on both Virtex-4 and Virtex-5 FPGAs and found to be less than 5 mW at 100 MHz. Power requirements for the previous approaches in [5], [10] and [12] were not reported.

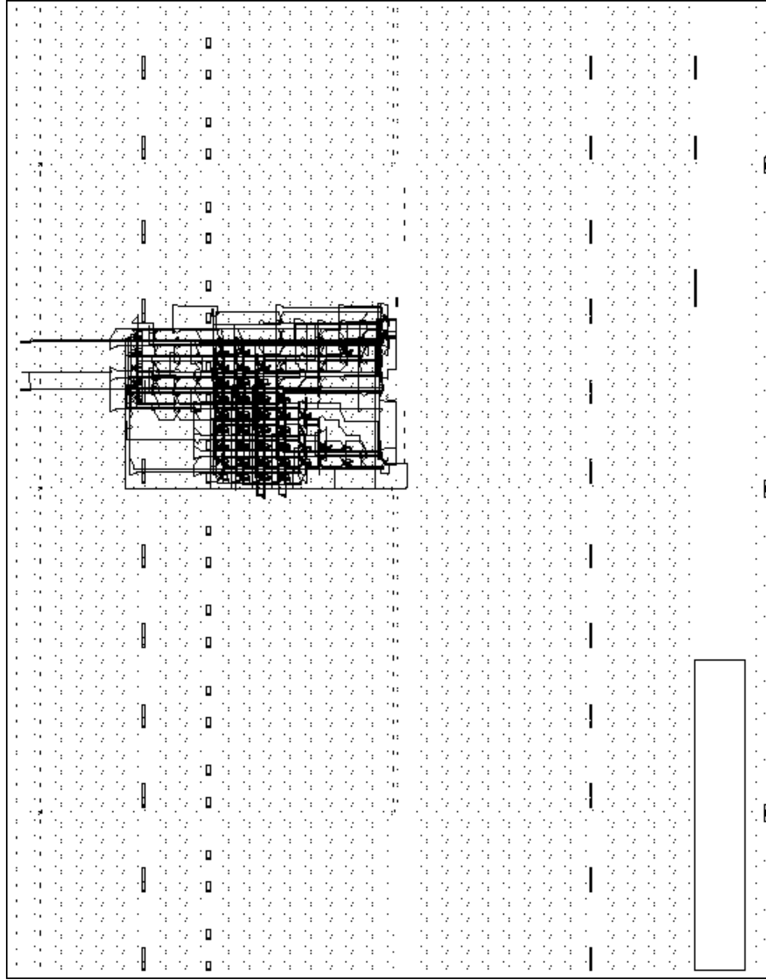


Figure 8.8: Routed SEU controller implemented in Virtex-5 LX20T device

For design verification and analysis, we developed an approach to emulate SEUs in the configuration memories of Virtex-4 and Virtex-5 FPGAs using a configuration memory read-modify-write process [8] similar to the approach described in [11]. The read-modify-write process is executed by an external computer connected to the FPGA via the Boundary Scan configuration interface. A list of configuration bit addresses is generated by software we developed to select random locations for SEU injection. Our SEU list generation software also allows for control of the locations of the SEUs to either a specific region or the entire configuration memory. Additionally, a rectangular area of the FPGA can be masked such that SEUs are randomly located outside of the mask area. Our approach is capable of injecting any

number of errors in the configuration memory, simultaneously or individually, as determined by the length of the SEU target list [8]. This SEU emulation approach was shown in [11] to reproduce 97% of actual SEU and SET induced faults in radiation chamber experiments. Furthermore, because the entire configuration memory is accessible, greater than 99% of all possible SEUs in the configuration memory of a given Virtex-5 FPGA can be emulated with this approach (refer to Table 8.1).

The analysis process begins by configuring the target device with the error-free SEU controller configuration. The SEU controller is held in reset while the SEUs are injected into the configuration memory via the Boundary Scan interface. For each SEU in the list, the corresponding frame of configuration memory is read back from the target device to the external computer. The SEU emulation bit in the frame is inverted, and the frame is written back to the same location in the configuration memory. After injection of the SEU(s), the SEU controller is released from reset and executed for one or more complete cycles. The number of single-bit and multiple-bit errors reported by the SEU controller are recorded by internal counters included for analysis and verification only, and these count values are read via the Boundary Scan interface at the end of the error detection/correction cycles. The success of the SEU controller is determined by comparing the values in the counters to the number of SEUs contained in the original list. Emulated configuration memory SEUs are classified in two categories. The first category includes all SEUs that are detected and corrected normally, as verified by a comparison of the retrieved count values and the original SEU list. The second category encompasses any SEU that affects the operation of the SEU controller such that either the SEU cannot be detected and corrected or the values contained in the counters are incorrect or cannot be retrieved for verification. Note that a slight penalty is incurred for the inclusion of the counters, which are

susceptible to SEUs, and could produce a failing pattern despite the correction of the emulated SEU. A total of 8,000 randomly generated SEUs were individually injected in the configuration memory of a Virtex-5 LX50T and the result of each trail was recorded. Our trials showed that, of the 8,000 random SEUs, all but 178 were detected and corrected in the first full execution cycle, yielding a probability of detection and correction of 97.78%. Considering the SEU locations to be randomly distributed, independent samples, the lower bound for the probability of correction of SEUs at the 99% confidence level is 97.30% [22]. Therefore, the likely probability of detection and correction of any number of simultaneous SEUs greater than one is given by:

$$\Pr(\text{correction}) = [1 - \Pr(\text{failure})]^N$$

where N is the number of simultaneously occurring SEUs. The results of SEU emulation for 1000 SEUs in four Virtex-5 devices are shown in Table 8.6. In our trials, 100% of SEUs that lie outside of the area of the configuration memory that controls the functionality of the SEU controller are corrected. The experimental success rates for [10] and [12] were not reported.

Table 8.6: SEU emulation results

Device	**Slice Count**	**Pop. Size (Mb)**	**Corrected/Injected**	**Pr(correction) 99% Confidence**
LX30T	59	7.29	950/1000	93.22%
SX35T	60	9.26	955/1000	93.46%
LX50T	59	10.9	980/1000	96.86%
SX50T	60	13.9	967/1000	94.96%
LX50T	59	10.9	7822/8000	97.35%

In general, the percentage of correctable SEUs is positively correlated to the size of the configuration memory of the given device because the number of configuration bits affecting the SEU controller functionality are fixed in relation to the total size of the configuration memory. According to the data provided in [5], the adjusted FIT rate, considering only the vulnerable bits which implement the SEU controller functionality, may be approximately calculated based on

the number of resources in use by the SEU controller and the number of configuration bits affecting the programming of each type of resource (shown in Table 8.7). For the Xilinx Virtex-5 SEU controller, the approximate number of sensitive configuration bits was reported to be 113,365 bits, or 0.108 Mb, yielding a nominal FIT rate of 16.33, or MTBF of approximately 6,992 years [5]. For our SEU controller, which utilizes less logic resources in Virtex-5, there are approximately $[(65 \times 1{,}181) + (1 \times 585)] = 77{,}350$ bits, or 0.0738 Mb, that are sensitive to SEUs. Therefore, the adjusted FIT rate for our SEU controller is 11.14, or MTBF of approximately 10,247 years. As was observed in the SEU emulation results, the adjusted FIT rate for designs protected by the SEU controller is independent of the device size because the size of the SEU controller is approximately device independent.

Table 8.7: Approximate number of configuration bits for common resources [5]

Resource	Approximate number of configuration bits
Logic Slice	1,181
Block RAM (36 Kb)	1,170
Block RAM (18 Kb)	585
I/O Tile	2,657
DSP48E Slice	4,592

8.7 Conclusions

The increased use of FPGAs for implementing digital systems, in conjunction with their larger configuration memories and shrinking design rules, has raised concerns about the effects of SEUs, particularly for high-altitude and space applications as well as for high-reliability, high-availability applications. As a result, some FPGA manufacturers are reducing the FIT rate through their design of the configuration memory and by incorporating modules that support SEU detection, such as the Frame ECC and ICAP in recent Xilinx FPGAs [25][26] and CRC background check circuitry in recent Altera [21] and Lattice [14] FPGAs. We have presented an

SEU controller applicable to all Xilinx Virtex-4 and Virtex-5 FPGAs that is capable of correcting single-bit errors and detecting double-bit errors in the FPGA configuration memory, which represents greater than 99% of all memory elements susceptible to SEUs. Note that block RAMs account for the second largest percentage (approximately 14%) of memory elements susceptible to SEUs. However, recent Xilinx [25][26] and Altera [21] FPGAs include RAMs cores with user optional ECC modes of operation. The SEU controller VHDL is easily integrated with any existing user design with minimal resource overhead and power dissipation. Our approach detects and corrects errors in the configuration memory 20 times faster than other reported approaches in [10] and [12]. In addition, our design is less susceptible to SEU induced failure because it uses less logic resources, which results in a failure rate improvement of about 46.6% for Virtex-5 FPGAs. Finally, TMR techniques can be used to prevent SEUs that occur within the configuration bits that establish the SEU controller logic from causing the SEU controller to fail in high-reliability, high-availability applications.

8.8 Acknowledgements

The contents of this chapter are published under the title "On-line Single Event Upset Detection and Correction in FPGAs Configuration Memories" in *The ISCA International Journal on Computers and Their Applications*, Vol. 17, No. 2. Prof. Charles Stroud is a co-author on the journal article. The journal article is an extended version of the work previously published in *Proceedings of the ISCA International Conference on Computers and Their Applications*, 2009, pp. 57-62, under the title "Single Event Upset Detection and Correction in Virtex-4 and Virtex-5 FPGAs". A majority of the actual research and the writing of the published paper represents the efforts of the primary student author and not collaborators, and the research represents work performed while in the graduate program at Auburn University.

8.9 References

[1] B. Bridgford, C. Carmichael, and C. Tseng, "Single-Event Upset Mitigation Selection Guide," XAPP987 (v1.0), Xilinx Inc., March 2008.

[2] M. Caffrey, P. Graham, E. Johnson, M. Wirthlin, C. Carmichael, "Single-Event Upsets in SRAM FPGAs," Military and Aerospace Programmable Logic Devices Conf., Sept. 2002.

[3] T. Calin, M. Nicolaidis, and R. Velazco, "Upset Hardened Memory Design for Submicron CMOS Technology," *IEEE Trans. on Nuclear Science*, vol. 43, no. 6, pp. 2874-2878, Dec. 1996.

[4] C. Carmichael and C. Wei Tseng, "Correcting SEUs in Virtex-4 Platform FPGA Configuration Memory," XAPP988, (v1.0), Xilinx Inc., March 2008.

[5] K. Chapman and L. Jones, "SEU Strategies for Virtex-5 Devices," XAPP864 (v1.0.1), Xilinx Inc., March 2009.

[6] *Device Reliability Report: Fourth Quarter 2008*, UG116 (v5.3) , Xilinx Inc., Feb. 2009.

[7] B. Dutton and C. Stroud, "Single Event Upset Detection and Correction in Virtex-4 and Virtex-5 FPGAs," *Proc. ISCA Int. Conf. on Computers and Their Applications*, pp. 57-62, April 2009.

[8] B. Dutton, M. Ali, J. Sunwoo and C. Stroud, "Embedded Processor Based Fault Injection and SEU Emulation for FPGAs," *Proc. Int. Conf. on Embedded Systems and Applications*, pp. 183-189, July 2009.

[9] "Flux Calculation," <http://www.seutest.com/cgi-bin/FluxCalculator.cgi>, April 2009.

[10] J. Heiner, N. Collins, and M. Wirthlin, "Fault Tolerant ICAP Controller for High-Reliable Internal Scrubbing," *Proc. IEEE Aerospace Conf.*, pp. 1-10, March 2008.

[11] E. Johnson, M. Caffrey, P. Graham, N. Rollins, and M. Wirthlin, "Accelerator Validation of an FPGA SEU Simulator," *IEEE Trans. on Nuclear Science*, vol. 50, no. 6, pp. 2147-2157, Dec. 2003.

[12] L. Jones, "Single Event Upset (SEU) Detection and Correction Using Virtex-4 Devices," XAPP714 (v 1.5), Xilinx Inc., Jan. 2007.

[13] F. Kastensmidt, L. Carro, and R. Reis, *Fault-Tolerance Techniques for SRAM-based FPGAs*, Frontiers in Electronic Testing, Vol. 32, Dordrecht, The Netherlands, Springer, 2006.

[14] "LatticeECP3 Soft Error Detection (SED) Usage Guide," TN1184 (v1.0), Lattice Semiconductor Inc., Feb. 2009.

[15] A. Lesea, “Continuing Experiments of Atmospheric Neutron Effects on Deep Submicron Integrated Circuits,” WP286 (v1.0), Xilinx Inc., March 2008.

[16] A. Lesea and P. Alfke, “Xilinx FPGAs Overcome the Side Effects of Sub-90 nm Technology,” WP256 (v1.0.1), Xilinx Inc., March 2007.

[17] A. Lesea, S. Drimer, J. Fabula, C. Carmichael, and P. Alfke, “The Rosetta Experiment: Atmospheric Soft Error Rate Testing in Differing Technology FPGAs,” *IEEE Trans. on Device and Materials Reliability*, Vol. 5, No. 3, pp. 317-328, Sept. 2005.

[18] M. Ohlsson, P. Dyreklev, and K. Johansson, “Neutron Single Event Upsets in SRAM-based FPGAs,” *Proc. IEEE Radiation Effects Data Workshop*, pp. 177-180, July 1998.

[19] *PicoBlaze 8-bit Embedded Microcontroller User Guide*, UG129 (v1.1.2), Xilinx Inc., June 2008.

[20] H. Quinn, P. Graham, K. Morgan, M. Caffrey and J. Krone, "A Test Methodology for Determining Space-Readiness of Xilinx SRAM-based FPGA Designs," *Proc. IEEE Automatic Test Conf. (AUTOTESTCON),* pp. 252-258, Sept. 2008.

[21] “Robust SEU Mitigation with Stratix III FPGAs,” WP-01012-1.0, Altera Inc., Jan. 2007.

[22] J. Sauro and J.R. Lewis, “Estimating Completion Rates From Small Samples Using Binomial Confidence Intervals,” *Proc. Human Factors and Ergonomics Society*, pp. 2100-2104, 2005, available at <www.measuringusability.com/wald>.

[23] L. Sterpone and M. Violante, “A Design Flow for Protecting FPGA-based Systems Against Single Event Upsets," *Proc. IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Systems*, pp. 436-444, Oct. 2005.

[24] P. Sundararajan, S. McMillan, B. Blodget, C. Carmichael, and C. Patterson, “Estimation of Single Event Upset Probability Impact of FPGA Designs,” Military and Aerospace Programmable Logic Devices Conf., Sept. 2003.

[25] *Virtex-4 FPGA Configuration User Guide*, UG071 (v1.10), Xilinx Inc., April 2008.

[26] *Virtex-5 FPGA Configuration User Guide*, UG191 (v3.6), Xilinx Inc., Feb. 2009.

[27] *Virtex-5 Family Overview*, DS100 (v5.0), Xilinx Inc., Feb. 2009.

[28] *Virtex-6 Family Overview*, DS150 (v1.0), Xilinx Inc., Feb. 2009.

[29] *Xilinx TRMTool User Guide: TMRTool Software Version 9.2i*, UG156 (v2.2), Xilinx Inc., 2009.

[30] C. Yui, G. Swift, and C. Carmichael, “Single Event Upset Susceptibility Testing of the Xilinx Virtex-II FPGA,” Military and Aerospace Programmable Logic Devices Conf., Sept. 2002.

Chapter Nine. Summary and Conclusions

This chapter concludes and summarizes the thesis. First, a summary of the work presented in this thesis is provided, followed by suggestions for future research and any improvements to the work.

9.1 Summary of Work

A BIST approach was presented for the CLBs in Virtex-5 FPGAs. A total of 17 configurations were used to obtain 100% stuck-at fault coverage in every CLB in any Virtex-5 device. Gate level fault simulation and configuration memory fault emulation were used for the development and verification of test configurations and for calculating fault coverage. A new ORA design was introduced which provides a single-bit pass/fail result for all of the resources under test. This ORA design has since been used in every BIST configuration that has been developed for Virtex-4 and Virtex-5 FPGAs. The overall test time is minimized by using partial reconfiguration of the resources under test and the single-bit pass/fail indication at the conclusion of each test session. However, for fault diagnosis, the contents of every ORA may be retrieved via partial configuration memory readback, and the locations of faults determined algorithmically based on the locations of the failing ORAs.

This thesis also presented a BIST approach for the I/O Tiles in Virtex-5 FPGAs. This approach shares many features of the approach for CLBs, including pseudo-exhaustive testing of the embedded resources and comparison-based output response analysis (using the improved ORA design with single-bit pass/fail). One interesting difference with the I/O BIST approach is the ability to apply a limited number of deterministic test patterns using block RAMs in the

FPGA fabric to store the test pattern set. For Virtex-4, 512 test patterns could be stored in a block RAM, and in Virtex-5 the number increased to 1024. However, due to the lack of any gate-level description of the I/O Tiles in Xilinx devices, it is difficult to evaluate the effectiveness of the test patterns. One of the most significant contributions of this work is the use of dedicated feedback routing in the I/O Tile to bypass the I/O buffer (and pad) during tests of the digital logic resources in the I/O Tile. This effectively separates the digital logic portion of the I/O tiles from the external "analog" environment, making the approach applicable to board-level and in-system testing. Consequently, independent tests for the I/O buffers were developed. These BIST configurations are also package independent because they can test I/O tiles with both bonded and unbonded I/O buffers, which is important because synthesis tools will sometimes use the logic resources in an I/O Tile with an un-bonded I/O buffer to implement a portion of the system function.

Next, a BIST approach was presented for the embedded cores in Xilinx Virtex-4 and Virtex-5 FPGAs that are used for the detection and correction of SEUs in the configuration memory of these devices. This work is related to the SEU controller that is presented later in the thesis in that the SEU controller uses these cores for detection and correction of SEUs; therefore, the fault-free operation of the cores is essential. One interesting difference between this BIST approach and the approaches presented for CLBs and I/O Tiles is that this approach was developed entirely in VHDL (as opposed to an XDL netlist). A VHDL-based approach is possible because there is only one circuit to test, and, therefore, no redundant TPG or ORA logic and no placement restrictions for the CUT.

Fault injection is a well known method for emulating faults or SEUs in the configuration memory of FPGAs. However, this thesis improves upon the existing approach by performing

fault-injection using a soft-processor configured in the fabric of the FPGA. This approach can be used during the development of BIST for FPGA resources or for verification of SEU mitigation schemes (but not as part of the manufacturing or system-level test). For example, the fault-injection core could "inject" a list of random SEUs while monitoring the behavior of the system function. Based upon the occurrence of errors in the system function, the actual FIT rate of the user function in any environment could be estimated, and several different SEU mitigation schemes could be quickly evaluated.

The next two chapters of the thesis present a new approach for BIST of FPGAs. This approach uses a soft-core processor configured in the fabric of the FPGA under test to perform reconfiguration of the BUTs, control the BIST sequence, and even perform fault diagnosis. However, the irregularity of the embedded processor makes configuration files too large to compete with the highly optimized BIST configurations. This thesis shows that the overall test time is significantly less when performing partial reconfiguration of the full FPGA array from an external BIST controller. However, the approach may still be useful for in-system testing, especially in fault tolerant applications, because it significantly reduces the complexity of the external BIST control hardware. For example, the embedded processor can perform all of the reconfigurations of the BUTs and determine the results of the BIST, reporting a single-bit pass/fail result to the system for all of the resources under test.

Finally, an approach for the on-line detection and correction of SEUs in the configuration memory of Virtex-4 and Virtex-5 FPGAs is presented. This chapter shows that no external hardware is required for the approach, because readback of configuration data and error detection and correction are all performed by additional logic included in the FPGA fabric. While greatly reducing the probability of an SEU, experimental results are provided to show that the approach

is not entirely immune to an SEU induced error. However, no single SEU can permanently corrupt the user function, and SEUs can only persist in the user function for a period of time equal to the cycle period of the SEU controller (*i.e.* the amount of time for the SEU controller to read every frame of configuration data in a given device). The thesis also shows that the cycle time and probability of an SEU induced failure are functions of the device size, with larger devices having a longer cycle time and lower probability of failure. In addition, a quantitative method for estimating the FIT rate in devices protected by the SEU controller is provided based on an approach in the previous work.

9.2 Future Work

The BIST approaches presented for the CLBs and I/O Tiles in Virtex-5 FPGAs can be adapted to Virtex-6 devices with few architectural modifications. The TPGs and ORAs can be implemented in a similar manner in Virtex-6 devices (which include DSPs and Block RAMs), but the detailed test configurations will need to be modified for the new device architectures.

The embedded BIST approach can also be updated to support Virtex-6 devices, but larger configuration file sizes for these devices may make the approach impractical. However, in systems with an intelligent BIST controller (embedded processor, PC, etc…) the configuration file compression methods presented in this thesis are applicable and potentially very useful for saving memory, especially for in-system testing.

The SEU controller is becoming more important due to the increasing size of the configuration memory and shrinking design rules. The configuration memory size in Virtex-6 devices is on average double that of Virtex-5 devices; and because the SEU controller cycle time is a function of the size of the configuration memory, the average cycle time can be expected to double. Testing the Frame ECC logic is also more important in Virtex-6 devices. Due to the

doubling of the configuration frame size, there is more logic in the Frame ECC that must be tested.

Bibliography

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

[2] M. Abramovici, C. Stroud, and J. Emmert, "Online BIST and BIST-based diagnosis of FPGA logic blocks," *IEEE Trans. on Very Large Scale Integr. (VLSI) Syst.*, vol.12, no.12, pp. 1284-1294, 2004.

[3] *AT94K Series Field Programmable System Level Integrated Circuit*, DS1138, Atmel Corp., 2001.

[4] J. Bailey et. al., "Bridging Fault Extraction from Physical Design Data for Manufacturing Test Development," *Proc. IEEE Int. Test Conf.*, pp. 760-769, 2000.

[5] D. Bossen, D. Ostapko, and A. Patel, "Optimum test patterns for parity networks," *Proc. AFIPS Fall 1970 Joint Comput. Conf.*, pp. 63-68, 1970.

[6] B. Bridgford, C. Carmichael, and C. Tseng, "Single-Event Upset Mitigation Selection Guide," XAPP987 (v1.0), Xilinx Inc., 2008.

[7] S. Brown and J. Rose, "FPGA and CPLD architectures: a tutorial," *IEEE Design & Test of Computers*, vol.13, no.2, pp.42-57, 1996.

[8] M. Bushnell and V. Agrawal, *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*, New York: Springer, 2000.

[9] M. Caffrey, P. Graham, E. Johnson, M. Wirthlin, C. Carmichael, "Single-Event Upsets in SRAM FPGAs," Military and Aerospace Programmable Logic Devices Conf., 2002.

[10] T. Calin, M. Nicolaidis, and R. Velazco, "Upset Hardened Memory Design for Submicron CMOS Technology," *IEEE Trans. on Nuclear Science*, vol. 43, no. 6, pp. 2874-2878, 1996.

[11] C. Carmichael and C. Wei Tseng, "Correcting SEUs in Virtex-4 Platform FPGA Configuration Memory," XAPP988, (v1.0), Xilinx Inc., 2008.

[12] K. Chapman and L. Jones, "SEU Stratagies for Virtex-5 Devices," XAPP864 (v1.0.1), Xilinx Inc., 2009.

[13] P. Christie, D. Stroobandt, “The Interpretation and Application of Rent’s Rule,” *IEEE Trans. on VLSI Systems*, vol. 8, no. 6, pp. 639-648, 2000.

[14] P. Civera, L. Macchiarulo, M. Rebaudengo, M. Reorda and M. Violante, “An FPGA-Based Approach for Speeding-Up Fault Injection Campaigns on Safety-Critical Circuits,” *Journal of Electronic Testing: Theory and Applications*, vol. 18, pp, 261–271, 2002.

[15] A. Cosoroaba and F. Rivoallon, “Achieving Higher System Performance with the Virtex-5 Family of FPGAs,” Xilinx Inc., 2006.

[16] *Device Reliability Report: Fourth Quarter 2008*, UG116 (v5.3) , Xilinx Inc., 2009.

[17] S. Dhingra, D. Milton, and C. Stroud, “BIST for logic and memory resources in Virtex-4 FPGAs,” *Proc. IEEE North Atlantic Test Workshop*, pp. 19-27, 2006.

[18] S. Dhingra, S. Garimella, A. Newalker, and C. Stroud, “Built-in self-test of Virtex and Spartan II FPGAs using partial reconfiguration,” *Proc. IEEE North Atlantic Test Workshop*, pp. 7-14, 2005.

[19] B. Dutton, M. Ali, J. Sunwoo and C. Stroud, “Embedded Processor Based Fault Injection and SEU Emulation for FPGAs,” *Proc. Int. Conf. on Embedded Systems and Applications*, pp. 183-189, 2009.

[20] B. Dutton and C. Stroud, “Built-In Self-Test of Configurable Logic Blocks in Virtex-5 FPGAs,” *Proc. IEEE Southeastern Symp. on System Theory*, pp. 230-234, 2009.

[21] B. Dutton and C. Stroud, “Built-In Self-Test of Programmable Input/Output Tiles in Virtex-5 FPGAs,” *Proc. IEEE Southeastern Symp. on System Theory*, pp. 235-239, 2009.

[22] B. Dutton and C. Stroud, “Single Event Upset Detection and Correction in Virtex-4 and Virtex-5 FPGAs,” *Proc. ISCA Int. Conf. on Computers and Their Applications*, pp. 57-62, 2009.

[23] B. Dutton and C. Stroud, “Soft-core Embedded Processor Based Built-In Self-Test of FPGAs,” *Proc. IEEE Int. Symp. On Defect and Fault Tolerence in VLSI Systems*, pp. 29-37, 2009.

[24] P. Ellervee, J. Raik, K. Tammemäe and R. Ubar, “Environment for FPGA-based Fault Emulation,” *Proc. Estonian Acad. Sci. Eng.*, vol. 12, pp. 323–335, 2006.

[25] “Flux Calculation,” <http://www.seutest.com/cgi-bin/FluxCalculator.cgi>, April 2009.

[26] B. Garrison, D. Milton, and C. Stroud, “Built-In Self-Test for Memory Resources in Virtex-4 FPGAs,” *Proc. ISCA Int. Conf. on Computers and Their Applications*, pp. 63-68, 2009.

[27] S. Gupta, J. Rajski, and J. Tyszer, “Test pattern generation based on arithmetic operations,” *Proc. IEEE Int. Conf. on Computer-Aided Design*, pp. 117-124, 1994.

[28] J. Heiner, N. Collins, and M. Wirthlin, “Fault-tolerant ICAP Controller for High-Reliable Internal Scrubbing,” *IEEE Aerospace Conf.*, pp. 1-10, 2008.

[29] S. Hwang, J. Hong and C. Wu, “Sequential Circuit Fault Simulation Using Logic Emulation,” *IEEE Trans. on CAD of ICs and Systems*, vol. 17, no. 8, pp. 724-736, 1998.

[30] *IEEE Standard Test Access Port and Boundary-Scan Architecture*, IEEE Std 1149.1-2001, New York, 2001.

[31] *IEEE Standard Testability Method for Embedded Core-Based Integrated Circuits*, IEEE Std. 1500-2005, New York, 2005.

[32] C. Jia and L. Milor, “A BIST Solution for the Test of I/O Speed,” *Proc. IEEE Int. Test Conf.*, pp. 1023-1030, 2003.

[33] E. Johnson, M. Caffrey, P. Graham, N. Rollins, and M. Wirthlin, “Accelerator Validation of an FPGA SEU Simulator,” *IEEE Trans. on Nuclear Science*, vol. 50, no. 6, pp. 2147-2157, Dec. 2003.

[34] W-B Jone and C-J Wu, "Multiple fault detection in parity checkers," *IEEE Trans. on Computers*, vol.43, no.9, pp.1096-1099, 1994.

[35] L. Jones, “Single Event Upset (SEU) Detection and Correction Using Virtex-4 Devices,” Application Note XAPP714 (v 1.5), Xilinx Inc., 2007.

[36] F. Kastensmidt, L. Carro, and R. Reis, *Fault-Tolerance Techniques for SRAM-based FPGAs*, Frontiers in Electronic Testing, Vol. 32, Dordrecht, The Netherlands: Springer, 2006.

[37] I. Kuon and J. Rose, “Measuring the Gap Between FPGAs and ASICs,” *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol.26, no.2, pp.203-215, 2007

[38] K. Leach et. al., “BIST for Xilinx 4000 and Spartan Series FPGAs: A Case Study,” *Proc. IEEE Int. Test Conf.*, pp. 1258-1267, 2003.

[39] “LatticeECP3 Soft Error Detection (SED) Usage Guide,” TN1184 (v1.0), Lattice Semiconductor Inc., 2009.

[40] A. Lesea, “Continuing Experiments of Atmospheric Neutron Effects on Deep Submicron Integrated Circuits,” WP286 (v1.0), Xilinx Inc., 2008.

[41] A. Lesea and P. Alfke, “Xilinx FPGAs Overcome the Side Effects of Sub-90 nm Technology,” WP256 (v1.0.1), Xilinx Inc., 2007.

[42] A. Lesea, S. Drimer, J. Fabula, C. Carmichael, and P. Alfke, “The Rosetta Experiment: Atmospheric Soft Error Rate Testing in Differing Technology FPGAs,” *IEEE Trans. on Device and Materials Reliability*, Vol. 5, No. 3, pp. 317-328, 2005.

[43] L. Lerner, "Built-In Self-Test for Input/Output Tiles in Field Programmable Gate Arrays," M.S. thesis, Dept. of Elect. and Comput. Eng., Auburn Univ., Auburn, AL, Dec. 2007.

[44] L. Lerner, S. Vemula, and C. Stroud, "System-Level BIST for Programmable I/O Buffers in FPGAs and SoCs," *Proc. IEEE North Atlantic Test Workshop*, pp. 1-9, 2006.

[45] *MicroBlaze Processor Reference Guide*, UG081(v.9.0), Xilinx Inc., 2008.

[46] D. Milton, S. Dhingra, and C. Stroud, "Embedded Processor Based Built-In Self-Test and Diagnosis of Logic and Memory Resources in FPGAs," *Proc. Int. Conf. on Embedded Systems and Applications*, pp. 87-93, 2006.

[47] G. Moore, "Cramming More Components onto Integrated Circuits," *Proc. of the IEEE*, vol. 86, no. 1, pp. 82-85, 1998.

[48] S. Mourad and E. McCluskey, "Testability of parity checkers," *IEEE Trans. on Industrial Electronics*, vol. 36, no. 2, pp. 254-262, 1989.

[49] E. Normand, "Single Event Upset at Ground Level," *IEEE Transs on Nuclear Science*, vol. 43, pp. 2742-2750, 1996.

[50] M. Ohlsson, P. Dyreklev and K. Johansson, "Neutron Single Event Upsets in SRAM-Based FPGAs," *Proc. IEEE Nuclear and Space Radiation Effects Conf.*, pp. 177-180, 1998.

[51] *PicoBlaze 8-bit Embedded Microcontroller User Guide*, UG129 (v1.1.2), Xilinx Inc., 2008.

[52] M. Pulukuri and C. Stroud, "Built-In Self-Test of Digital Signal Processors in Virtex-4 FPGAs," *Proc. IEEE Southeastern Symp. on System Theory*, pp. 34-38, 2009.

[53] H. Quinn, P. Graham, K. Morgan, M. Caffrey and J. Krone, "A Test Methodology for Determining Space-Readiness of Xilinx SRAM-based FPGA Designs," *Proc. IEEE Automatic Test Conf. (AUTOTESTCON),* pp. 252-258, 2008.

[54] R. Rajsuman, "Testing a System-On-Chip with Embedded Microprocessor," *Proc. IEEE Int. Test Conf.*, pp. 499-508, 1999.

[55] "Robust SEU Mitigation with Stratix III FPGAs," WP-01012-1.0, Altera Inc., Jan. 2007.

[56] J. Sauro and J.R. Lewis, "Estimating Completion Rates From Small Samples Using Binomial Confidence Intervals," *Proc. Human Factors and Ergonomics Society*, pp. 2100-2104, 2005, available at <www.measuringusability.com/wald>.

[57] R. Sedaghat, "Routability estimation of FPGA-based fault injection," *Electronics Letters*, vol. 41, no. 14, pp. 790-792, 2005.

[58] Semiconductor Industry Association, *International Technology Roadmap for Semiconductors: 2007 edition*, http://public.itrs.net.

[59] T. Slaughter, C. Stroud, J. Emmert and B. Skaggs, "Fault Injection Emulation for Field Programmable Gate Arrays," *Proc. Int. Society for Optical Eng.*, vol. 4525, pp. 1-9, 2001.

[60] M. Smith, *Application-Specific Integrated Circuits*, Addison-Wesley, 1997.

[61] L. Sterpone and M. Violante, "A Design Flow for Protecting FPGA-based Systems Against Single Event Upsets," *Proc. IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Systems*, pp. 436-444, 2005.

[62] C. Stroud, *A Designer's Guide to Built-In Self-Test*, Boston: Springer, 2002.

[63] C. Stroud, S. Konala, P. Chen, and M. Abramovici, "Built-in self-test of logic blocks in FPGAs," *Proc. IEEE VLSI Test Symp.*, pp.387-392, 1996.

[64] C. Stroud and S. Garimella, "BIST and diagnosis of multiple embedded cores in SoCs," *Proc. Int. Conf. on Embedded Systems and Applications*, pp. 130-136, 2005.

[65] C. Stroud, S. Garimella and J. Sunwoo, "On-Chip BIST-Based Diagnosis of Embedded Programmable Logic Cores in System-On-Chip Devices," *Proc. ISCA Int. Conf. on Computers and Their Applications*, pp. 308-313, 2005.

[66] C. Stroud, J. Harris, S. Garimella, and J. Sunwoo, "Built-in self-test for system-on-chip: a case study," *Proc. IEEE Int. Test Conf.*, pp. 837-846, 2004.

[67] C. Stroud, K. Leach, and T. Slaughter, "BIST for Xilinx 4000 and Spartan series FPGAs: a case study," *Proc. IEEE Int. Test Conf.*, pp. 1258-1267, 2003.

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

[69] P. Sundararajan, S. McMillan, B. Blodget, C. Carmichael, and C. Patterson, "Estimation of Single Event Upset Probability Impact of FPGA Designs," Military and Aerospace Programmable Logic Devices Conf., 2003.

[70] J. Sunwoo and C. Stroud, "Built-In Self-Test of Configurable Cores in SoCs Using Embedded Processor Dynamic Reconfiguration," *Proc. Int. SoC Design Conf.*, pp. 174-177, 2005.

[71] S. Toutounchi and A. Lai, "FPGA test and coverage," *Proc. IEEE Int. Test Conf.*, pp. 599-607, 2002.

[72] A. van de Goor, *Testing Semiconductor Memories Theory and Practice*, Hoboken: John Wiley and Sons, 1991.

[73] S. Vemula and C. Stroud, “Built-In Self-Test for Programmable I/O Buffers in FPGAs and SoCs”, *Proc. IEEE Southeastern Symp. on System Theory*, pp. 534-538, 2006.

[74] *Virtex-4 FPGA Configuration User Guide*, UG071 (v1.1), Xilinx Inc., 2008.

[75] *Virtex-4 FPGA User Guide*, UG070 (v2.5), Xilinx Inc., 2008.

[76] *Virtex-5 Family Overview*, DS100 (v5.0), Xilinx Inc., 2009.

[77] *Virtex-5 FPGA Configuration User Guide*, UG191 (v3.2), Xilinx Inc., 2008.

[78] *Virtex-5 FPGA ExtremeDSP Design Considerations: User Guide*, UG193 (v3.3), Xilinx Inc., 2009.

[79] *Virtex-5 FPGA User Guide*, UG190(v4.2), Xilinx Inc., 2008.

[80] *Virtex-6 Family Overview*, DS150 (v1.0), Xilinx Inc., 2009.

[81] L-T Wang, C. Stroud, and N. Touba, *System-on-Chip Test Architectures*, San Francisco: Morgan Kaufmann, 2007.

[82] L-T Wang, C-W Wu, and X. Wen, *VLSI Test Principles and Architectures*, San Francisco: Morgan Kaufmann, 2006.

[83] *Xilinx TRMTool User Guide: TMRTool Software Version 9.2i*, UG156 (v2.2), Xilinx Inc., 2009.

[84] *XPS HWICAP Product Specification*, DS586(v1.00.a),. Xilinx Inc., 2007.

[85] J. Yao et. al., “Built-In Self-Test of Programmable Interconnect in Virtex-4 FPGAs,” *Proc. IEEE Southeastern Symp, on System Theory*, pp. 29-33, 2009.

[86] C. Yui, G. Swift, and C. Carmichael, “Single Event Upset Susceptibility Testing of the Xilinx Virtex-II FPGA,” Military and Aerospace Programmable Logic Devices Conf., 2002.

[87] L. Zhao, D. Walker and F. Lombardi, “IDDQ Testing of Input/Output Resources of SRAM-Based FPGAs,” *Proc. Asian Test Symp.*, pp. 375-380, 1999.