Analysis and Implementation of Built-In Self-Test for Block Random Access Memories in Virtex-5 Field Programmable Gate Arrays

by

Justin Lewis Dailey

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
August 6, 2011

Keywords: Build-In Self-Test, Field Programmable Gate Array, Block RAM, Fault Coverage

Copyright 2011 by Justin Lewis Dailey

Approved by

Charles E. Stroud, Chair, Professor of Electrical and Computer Engineering
Victor P. Nelson, Professor of Electrical and Computer Engineering
Chwan-Hwa Wu, Professor of Electrical and Computer Engineering

Abstract

In order to ensure the proper operation of the embedded Block Random Access Memories (BRAMs) in Xilinx Virtex-5 Field-Programmable Gate Arrays (FPGAs) a dependable and resource efficient test is needed so that the integrity of the memory can be guaranteed in a timely manner. The approach that is described in this thesis is based on a Built-In Self-Test (BIST) approach initially proposed by Garimella in [1] for Xilinx Virtex-1 and Virtex-2 FPGAs. It was later expanded upon by Milton in [2] for Xilinx Virtex-4 FPGAs. The work was continued by Garrison as detailed in [3] for Virtex-4 in order to improve BIST generation and execution time. Garrison also proposed a design for BRAM BIST for Virtex-5 FPGAs in [3]. Garrison's proposal for Virtex-5 FPGAs is expanded upon and implemented in this thesis.

The testing approach for these BRAMs is described along with testing configurations and details. The BIST configurations are implemented using five unique Test Pattern Generators (TPGs) running testing algorithms on a combination of 19 separate RAM configurations in order to fully test the memories. All of the BIST configurations have been generated using two C programs developed as part of this thesis which are capable of generating configurations for any Virtex-5 device. These configurations were downloaded to various Virtex-5 FPGAs and tested on these devices. The fault detection capabilities of the BIST have been verified by using fault

injection within the BIST configurations that are downloaded to the FPGA to emulate physical faults within the configuration memory bits of the BRAMs. With fault injection, it was verified that this BIST approach was able to successfully detect 100% of detectable configuration memory faults in the BRAMs present in Virtex-5 devices.

Acknowledgments

I would like to thank Dr. Charles Stroud for his guidance throughout my undergraduate and graduate studies. He has helped me develop the skills I need to become a successful engineer once I graduate through personal advice, class work, and research. I would also like to thank Dr. Victor Nelson and Dr. Chwan-Hwa Wu for their guidance and for serving on my graduate committee. I also owe a great deal of thanks to my research colleague Alex Lusco for giving valuable help and advice on countless occasions throughout both my undergraduate and graduate studies. I would like to thank my other research colleagues Neil Da Cunha and Jie Qin as well. I would also especially like to thank my friends and family for supporting me throughout my educational process and giving me the excellent opportunities I have been fortunate enough to receive.

Table of Contents

Abstract ii

Acknowledgments iv

List of Tables viii

List of Figures x

List of Abbreviations xii

Chapter 1 Introduction 1

1.1 Field Programmable Gate Arrays 1

1.1.1 FPGA Architecture 2

1.1.2 Block Random Access Memory 3

1.1.3 Benefits and Drawbacks of FPGA Usage 4

1.2 Built-In Self-Test 5

1.2.1 Pros and Cons of a BIST Approach 6

1.2.2 BIST within FPGAs 8

1.3 Thesis Statement 9

Chapter 2 Background Information 10

2.1 Fault Modeling 10

2.2 Random Access Memories 13

2.2.1 SRAM Faults 16

2.2.2 RAM Test Algorithms ... 17
2.3 Virtex-5 Architecture ... 28
2.3.1 Virtex-5 Configurable Logic Blocks ... 28
2.3.2 Virtex-5 Block RAMs ... 30
2.4 Virtex-4 Block RAM BIST ... 36
2.4.1 Dedicated Carry Chain ... 37
2.4.2 TPG Architecture ... 39
2.5 Thesis Statement ... 47
Chapter 3 Virtex-5 Block RAM BIST ... 48
3.1 Virtex-5 RAM BIST ... 48
3.2 TPG Design and Implementation ... 49
3.2.1 BRAM ... 50
3.2.2 ECC ... 51
3.2.3 FIFO ... 52
3.2.4 FIFOECC ... 54
3.2.5 CASC ... 55
3.2.6 Test Configurations Summary ... 56
3.3 ORA Design ... 57
3.3.1 ORA Comparison Routing ... 58
3.4 BIST Implementation ... 65
3.4.1 Cascade Routing ... 67
3.5 Programming Tools ... 70
3.6 Configuration File Generation ... 71

3.6.1 BIST Generation Program 72
3.6.2 Modification Program 72
3.7 Results and Analysis 73
3.7.1 Fault Detection 75
3.7.2 File Size Reduction 79
3.7.3 Timing Analysis 82
Chapter 4 Summary and Conclusions 86
4.1 Summary of Virtex-5 BRAM BIST 86
4.2 Future Work 87
Bibliography 88
Appendix 91

List of Tables

Table 2-1 – 4-Bit BDS Components 21

Table 2-2 – 4-Bit BDS Sequence 21

Table 2-3 - CLB Resources in Virtex-5 [7] 28

Table 2-4 - Virtex-5 BRAM Port Aspect Ratio (18K-bit RAM) [7] 31

Table 2-5 – Virtex-5 BRAM Port Aspect Ratio (36K-bit RAM) [7] 31

Table 2-6 - FIFO Input and Output Ports [7] 34

Table 2-7 – Virtex-5 FIFO Port Aspect Ratio [7] 35

Table 2-8 – Virtex-5 FIFO Data Depth [7] 35

Table 2-9 - BRAM BIST Configurations [7] 41

Table 2-10 – Proposed Control String Values for BRAM TPG [3] 42

Table 2-11 - Proposed Configuration Settings for BRAM TPG [3] 42

Table 2-12 – Proposed Control String Values for ECC TPG [3] 44

Table 2-13 – Proposed Configuration Settings for ECC TPG [3] 45

Table 2-14 – Proposed Configuration Settings for FIFO TPG [3] 46

Table 2-15 – Proposed Configuration Settings for FIFOECC TPG [3] 47

Table 3-1 - Final BRAM Configuration Settings 51

Table 3-2 - Final ECC Configuration Settings 52

Table 3-3 - Final Control String Values for ECC TPG 52

Table 3-4 – Final FIFO Test Phases and Control String Values.. 53

Table 3-5 – Final Configuration Settings for FIFO TPG.. 53

Table 3-6 - Final FIFOECC Test Phases 55

Table 3-7 – Final Configuration Settings for FIFOECC TPG.. 55

Table 3-8 – Final CASC Test Phases.. 56

Table 3-9 – Final Configuration Settings for CASC TPG... 56

Table 3-10 - BIST TPG Resource Usage... 56

Table 3-11 – Complete Virtex-5 BIST Procedure 57

Table 3-12 - Compared Outputs for Configuration Modes 58

Table 3-13 - ORA Input Routing Tables 61

Table 3-14 - Complete BRAM BIST... 75

Table 3-15 - BIST Configuration File Sizes for LX30.. 80

List of Figures

Figure 2-1 – Gate Level Stuck-at Fault Behavior for AND Gate 12

Figure 2-2 - Two-Dimensional SRAM Model [6] 14

Figure 2-3 - Functional model of a multi-port memory [6] 15

Figure 2-4 - Slice arrangement within Virtex-5 CLBs [7] 29

Figure 2-5 - Virtex-5 Dual-Port Flow [7] 30

Figure 2-6 - Top Level View of Virtex-5 BRAM ECC [7] 33

Figure 2-7 - Virtex-5 BRAMs in Cascade Configuration [7] 36

Figure 2-8 - BRAM BIST Architecture [2] 37

Figure 2-9 - Comparison Based ORA with Carry Chain [3] 38

Figure 2-10 - Additional Dummy ORAs [3] 38

Figure 2-11 - Iterative OR-Chain Functionality [3] 39

Figure 2-12 - Shift Register Control String for BRAM TPGs [3] 43

Figure 2-13 - Placement and Routing on TPGs in LX30 43

Figure 3-1 - BRAM TPG Area Constraints in LX30 50

Figure 3-2 - ORA Map 60

Figure 3-3 – BRAM BIST Configuration Routed on Virtex-5 LX30 66

Figure 3-4 – Virtex-5 Cascade ORA Routing 69

Figure 3-5 - BIST Configuration Process [16] 71

Figure 3-6 - V5RAMBIST Command Line Instructions 72

Figure 3-7 - V5RAMMOD Command Line Instructions 73

Figure 3-8 – BRAM Configuration Mode Fault Detections 77

Figure 3-9 – Entire BIST Sequence Fault Detections 78

Figure 3-10 - BIST Configuration File Size Reduction for LX30 81

Figure 3-11 - Maximum BIST Clock Frequencies for LX30T 84

Figure 3-12 – Maximum BIST Clock Frequency for select Virtex-5 Devices 85

List of Abbreviations

ASIC	Application Specific Integrated Circuit
BDS	Background Data Sequence
BIST	Built-In Self-Test
BRAM	Block Random Access Memory
BSCAN	Boundary Scan
CASC	Cascade Configuration Mode
CLB	Configurable Logic Block
CMOS	Complementary Metal-Oxide Semiconductor
CUT	Circuit Under Test
DRAM	Dynamic Random Access Memory
DSP	Digital Signal Processor
ECC	Error Correction Code
FIFO	First-In First-Out
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
HDL	Hardware Design Language
I/O	Input/Output

LSB	Least Significant Bit
LUT	Look Up Table
MPGA	Mask Programmable Gate Array
MSB	Most Significant Bit
NMOS	N-type Metal-Oxide Semiconductor
ORA	Output Response Analyzer
PIP	Programmable Interconnect Point
RAM	Random Access Memory
SOC	System-on-Chip
SRAM	Static Random Access Memory
TDI	Test Data In
TDO	Test Data Out
TEMAC	Tri-mode Ethernet Media Access Controller
TPG	Test Pattern Generator
VHDL	VHSIC Hardware Design Language
VHSIC	Very-High-Speed Integrated Circuit
VLSI	Very Large Scale Integration

Chapter 1 Introduction

1.1 Field Programmable Gate Arrays

A Field Programmable Gate Array (FPGA) is a prefabricated integrated circuit that can be dynamically programmed by a user in the field rather than having permanent programming from the manufacturer, like such devices as Mask Programmable Gate Arrays (MPGAs) or Application Specific Integrated Circuits (ASICs) [4]. FPGAs contain programmable logic blocks that allow a user to designate the functionality of the device with both combinational logic using logic gates such as AND or XOR gates and sequential logic using elements such as a flip-flop [4]. Many FPGAs also contain embedded components that provide users with a convenient method for implementing more complicated circuits. Commonly included embedded devices include digital signal processors (DSPs), random access memories (RAMs), and embedded microprocessors [4]. FPGAs also contain configurable interconnection resources that are user programmable. The configurable routing allows circuit elements to be placed and routed in accordance with the user designed circuit. The functional behavior for a specific design is usually created using a Hardware Design Language (HDL) such as VHDL or Verilog which is then used to generate a configuration for the device [4].

FPGAs gained popularity due to their versatility for implementing circuit designs. FPGA circuits can be designed, implemented, and tested very quickly and are also very forgiving of design error because they can be easily reprogrammed repeatedly [4]. The complexity of FPGAs has grown from only a few thousand logic gates in their infancy to tens of millions of logic gates in modern chips [4].

An ASIC implementation is generally much smaller in size and much better in performance than an FPGA implementation. The circuit design of an ASIC cannot be modified once it is manufactured and must be specially designed. An ASIC circuit is expected to have a time delay that is four to five times less than that of the same circuit implemented on an FPGA while also consuming on average 14 times less power [5]. However, FPGAs offer the ability to reprogram that an ASIC cannot. A design error in an ASIC means that an entirely new ASIC device must be created where a design error in an FPGA only means the design must be modified and re-downloaded into the FPGA. Use of an FPGA allows a designer to save time and costs throughout the design process and reduces the penalty of having a design error in a prototype [4]. The extent to which an FPGA is programmable eliminates it from being able to compete with ASICs and MPGAs in size and performance.

1.1.1 FPGA Architecture

The components generally contained within an FPGA are [4]:

- Configurable Logic Blocks (CLBs)
- Input/Output (I/O) Cells
- Programmable Interconnect Points (PIPs) and Wire Segments
- Special Cores

Each CLB is usually made up of multiple Look Up Tables (LUTs) and flip-flops. The LUTs contain the binary data necessary to implement the combinational logic truth table of the programmed design and the flip-flops are used in the implementation of sequential logic [4]. The I/O Cells allow the devices to connect peripherally, and special cores in the form of microprocessors, RAMs, and DSPs are commonly included. All of these components are interconnected internally by utilizing a system of PIPs and wire segments for signal connection.

1.1.2 Block Random Access Memory

With the inclusion of specialized cores, FPGAs have started to resemble a full System-On-Chip (SOC) [4]. These specialized cores such as RAMs, DSPs, and microprocessors make memory and arithmetic implementations less cumbersome to the user and reduce the programmable logic resources demanded [4].

The two types of RAMs are Dynamic Random Access Memory (DRAM) and Static Random Access Memory (SRAM) [6]. The Block Random Access Memories (BRAMs) within a Xilinx FPGA are classified as SRAM. These memories require more area than DRAMs, but provide the fastest possible access speed of any RAM (usually 2 nanoseconds) [6]. SRAM cells have two separate stable states used to represent logic level zero and one [6]. The cells retain their state as long as they remain connected to a power supply and do not require a periodical refresh. However, the memory is volatile meaning if the power supply is disconnected from the cell the logic state will be lost.

The RAMs contained in a Virtex-5 FPGA can be configured to operate with a data width from 1 bit to 72 bits corresponding with an address space ranging from 32K to 512 data words [7]. The number of BRAMs supplied in a given Virtex-5 device spans from 26 in the smallest

device to 516 in the largest device [7]. The memories can be configured to function in different operational modes including the ability to be connected together in order to extend the address space to 64K. The BRAMs also have the ability to function with one address port and one data port in single port mode, or they are capable of using a pair of address ports and a pair of data ports to function in dual port mode. In dual port mode each port of the BRAM may be used to write or read from the memory independently and concurrently as long as they are not attempting to write to the same address simultaneously [7]. The BRAMs may also be configured to operate in a First In, First Out (FIFO) mode. When operating in the FIFO mode the BRAM functions similar to a queue line that is storing data. In this mode the BRAM has separate read and write clocks. When a write operation is triggered a data word will be added to the end of the queue, and when a read operation is triggered the data word at the front of the queue will be retrieved. Each BRAM also contains Error Correction Code (ECC) circuitry which is capable of correcting any single-bit error in the memory or detecting any double-bit error using Hamming code [7].

1.1.3 Benefits and Drawbacks of FPGA Usage

Use of FPGAs in circuit design and implementation gives the designer many advantages but also has a few drawbacks. The advantages of FPGA use stem mostly from its flexibility [4]:

- User programmability and re-programmability
- Accelerated design implementation and prototyping process

The user programmability and re-programmability gives a designer the ability to easily create a physical prototype of a digital circuit [4]. This allows users to comprehensively test their design before spending time and money to have an ASIC created for the circuit, and eliminates

much of the risk of having an unanticipated error in the circuit show up in the ASIC that will require refabrication of the entire device.

There are also some distinct disadvantages to FPGA usage compared to using an ASIC [4]:

- Higher production cost
- Higher power consumption
- Lower performance
- Volatile configuration memory

FPGA production is efficient for a low to medium volume design and expedited time-market-systems [4]. However when mass production of a device is needed the cost of an FPGA cannot compete with the cost of an equivalent ASIC [8].

1.2 Built-In Self-Test

As the complexity of Very Large Scale Integration (VLSI) devices continues to increase, the need for an efficient and economical testing method such as Built-In Self-Test (BIST) grows as well [4]. The general idea behind BIST is to design a circuit that is capable of verifying itself as being either faulty or fault-free. A standard BIST architecture contains three major components [9]:

- Test Pattern Generator (TPG)
- Circuit Under Test (CUT)
- Output Response Analyzer (ORA)

The TPG serves as a stimulus to the CUT, providing a set of inputs that will cause the CUT to generate an expected output. The resulting data from the CUT is analyzed by the ORA and is

simplified into some sort of pass/fail status depending on whether the ORA saw the expected output or an erroneous one [9]. Other components may be needed for system level implementation of the BIST such as a testing controller and input isolation circuitry. The BIST circuitry may contain an output bit to indicate success or failure to an external device and optionally a BIST done flag to indicate a finished testing sequence. The effectiveness of a BIST test is determined by the testing time and the number of faults that are detectable compared with the total amount of faults possible in a system known as fault coverage.

1.2.1 Pros and Cons of a BIST Approach

Using a BIST approach has many advantages associated with it when compared to other testing approaches such as external testing. These advantages include, but are not limited to [9]:

- Vertical Testability
- High Diagnostic Resolution
- At-Speed Testing
- Reduced Amount of External Testing Equipment
- Reduced Test Development Time and Effort
- Reduced Manufacturing Test Time and Cost
- Reduced Time-to-Market

Vertical testability means that a BIST can be applied to a device in any stage of production to determine its validity. A BIST that is applied to a system that gives an incorrect result reveals the system as faulty. It also inherently shows that a CUT associated with the device has faulty operation. Additionally, many times the specific faulty CUT can be identified, meaning that BIST has a high diagnostic resolution [9]. BIST is also able to use a system's internal clock for

at-speed testing which enables it to detect delay faults that are only visible when operating at system speed. The need for expensive external testing equipment is also eliminated. The only external I/O pins that must be provided are power, ground, a method for initializing the BIST, a method for retrieving BIST results, and a clock [9]. The savings in time and cost on test development resulting from internal TPG and ORA circuitry outweigh additional BIST design time in most cases and, consequentially, a reduced overall time-to-market [9].

Using a BIST approach also has drawbacks [9]:

- Larger Area Overhead
- Performance Penalties
- Additional Design Time and Effort
- Additional Project Risk

The additional circuitry that must be included in the design to implement the BIST means that the overall chip area will be larger, and therefore there will be a higher cost per chip as well as an increased area for defects to occur. The incorporation of the BIST circuitry may also cause the circuitry of the CUT to be spread out, or it may introduce additional gates into the CUT's critical path [9]. These cases will result in increased signal delay due to a longer routing path and increased gate delay which can be largely significant in some systems and negligible in others [9]. Additional time must also be taken to design and implement the BIST circuitry and testing technique. When using BIST another problem arises in design verification. By adding another entire system on top of the already existing system the project risk is increased as proper function of both of these systems is essential. Despite these drawbacks case studies have shown

that the benefits of using a BIST approach are more than enough to account for the costs incurred in a majority of scenarios [9].

1.2.2 BIST within FPGAs

Using a BIST approach offers even more of an advantage due to the programmable nature of FPGAs, making the BIST option even more enticing. In contrast, the implementation of a BIST for an ASIC requires design of the circuitry as well as additional components being added to the ASIC. FPGAs require testing in multiple configurations to achieve a high fault coverage, meaning the device must be tested in all modes of operation [4]. This causes the testing time to become mainly a function of the number of configurations that must be tested and also the configuration time. In order to optimize the testing time it is critical that the number of test configurations is kept at a minimum [4].

In an FPGA testing scenario the device inherently provides an abundance of configurable hardware that can be utilized for implementing BIST circuitry. There will be no area overhead or performance penalties present in the device after testing since the configuration is erasable [4]. However, the drawbacks associated with designing a functional test circuit are still applicable. The testing circuitry consisting of TPGs and ORAs is created by programming the CLBs, I/O cells, and routing resources within the FPGA in order to detect faults in the various components of the device [4]. This approach may be used to detect faults within CLBs, routing resources, and the specialized cores which may be found on the chip [4]. With respect to FPGAs, a BIST approach proves to be an extremely practical and efficient testing approach for verifying the integrity of the device and its individual components. The BIST method gains most of its effectiveness from the inherently configurable nature of FPGAs and its ability to have no effect on the device after testing has concluded.

1.3 Thesis Statement

This thesis will detail a testing approach derived from the BIST approach for FPGA embedded memory resources proposed by Garimella in [1] and later referenced and improved upon by Milton in [2] and later by Garrison in [3]. These previous FPGA BIST approaches were targeted at BRAMs contained within the Xilinx Virtex [1], Virtex-2 [1], and Virtex-4 [2][3] series of FPGAs, and an initial proposal for testing of the BRAMs within the Xilinx Virtex-5 series of FPGAs was provided by Garrison in [3]. The main focus of this thesis will be to implement, expand, and improve upon Garrison's initial proposal for BIST testing of BRAMs embedded in the Xilinx Virtex-5 devices and to detail a complete BIST approach and implementation for these memories.

This thesis will discuss the background material for the BRAM BIST in Chapter 2. Chapter 3 will detail the testing configurations for the BIST as well as the implementation of the BIST within the FPGA architecture and the method of fault injection used to measure the effectiveness of the test. Chapter 4 will present a summary and areas in which future research may be made to improve this testing approach.

Chapter 2 Background Information

This chapter will begin by discussing general fault detection techniques along with the basics of fault modeling within a circuit. Next, the circuitry and fault types that may be present within Static Random Access Memories (SRAMs) will be detailed. Then the different test algorithms used to detect the various types of faults in these memories will be described. The chapter will then describe the architecture of the Virtex-5 devices along with the embedded BRAMs and their modes of operation. Finally, the various components of the BIST architecture will be described along with previously proposed BIST configurations.

2.1 Fault Modeling

In order to test a circuit to determine its integrity a set of input stimuli is applied to the circuit and the output produced as a result of the stimuli is then compared with the expected output. A matching pair assumes the circuit as good while mismatched results will expose the CUT as faulty [9]. The input stimuli that are applied to the CUT during the test are usually a set of input vectors that are selected in order to ensure that the CUT performs as expected with no structural or functional faults [9].

In order to have an effective evaluation of the quality of a set of tests for a device and to evaluate the effectiveness of a BIST as it applies to the device, fault models must be used for

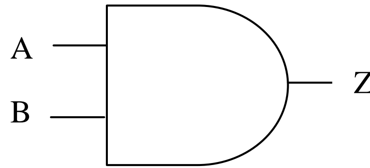
emulation of faults or defects during simulation [9]. In order for a fault model to be beneficial it must meet two requirements [9]:

- The model provides an accurate representation of the behavior of actual defects that may occur during the fabrication, manufacturing, and system operation of the device.
- The model must be computationally efficient.

These two requirements are often contradicting and make the creation of useful fault models difficult. Some of the most widely used fault models are the ones that can be emulated in a simulation environment efficiently and that provide close approximations of actual faults which may occur in a physical device are [9]:

- Gate-Level Stuck-at Faults
- Transistor-Level Faults
- Bridging Faults
- Delay Faults

The gate-level stuck fault model allows any of the inputs or outputs of a gate to be either stuck-at-0 (sa0) or stuck-at-1 (sa1). The behavior of the gate is then determined by treating the gate input or output which is being tested as either sa0 or sa1 as being disconnected and tied to either a logic zero or a logic one [9]. The results of a fault-free AND gate are compared with the results from each case of a sa0 or sa1 that may occur with that particular gate in the Figure 2-1. The cases that will be detected as faults are highlighted with grey. In the case of an AND gate each fault contains an instance where the output of the faulty gate differs from that of the fault free gate so each fault will be detected if sufficient input combinations are tested [9].



AND		A		B		Z	
AB	Z	sa0	sa1	sa0	sa1	sa0	sa1
00	0	0	0	0	0	0	1
01	0	0	1	0	0	0	1
10	0	0	0	0	1	0	1
11	1	0	1	0	1	0	1

Figure 2-1 – Gate Level Stuck-at Fault Behavior for AND Gate

Only gate-level fault models are required to simulate N-type metal-oxide-semiconductor (NMOS) circuits. However when using Complementary Metal-Oxide Semiconductor (CMOS) circuits, a transistor-level fault model is needed to obtain accurate results [9]. Bridging fault models are used to emulate shorted wire segments within a circuit [9]. Delay fault models are used to represent the case of a circuit that performs logically correct operations but does not meet the timing requirements [9].

When performing fault emulation a set of input vectors are applied to a circuit for each of a series of faults that have been artificially injected into the circuit [9]. With the fault injected into the circuit, the circuit will behave as if this fault has actually occurred. The output of this circuit will then be compared with the output of a fault free circuit. If a mismatch between the two outputs is found using the test vectors then the fault injected circuit has produced an erroneous result and the fault is considered to be detected [9]. If the complete set of test vectors is applied to the pair of circuits without a mismatch occurring then the fault is considered to be undetected [9]. The results of applying the entire set of test vectors to each of the possible faults in a circuit will determine the fault coverage of the test vectors. The fault coverage for a set of

vectors is a representation of the effectiveness of those vectors in detecting faults [9]. The calculation for determining fault coverage is given by [9]:

$$Fault\ Coverage = \frac{\#\ of\ Detected\ Faults}{\#\ of\ Total\ Faults} \qquad \text{2-1}$$

2.2 Random Access Memories

Static Random Access Memories (SRAMs) are made up of bi-stable memory cells which are capable of holding either a logic zero state or a logic one state. A memory cell holds only a single bit of information and will retain its value as long as the power remains connected to the cell without the need for a periodic refresh. However, the cells are volatile and will not retain their logic value after the power has been disconnected [6]. A general SRAM will contain input connections for controls, addresses, and data-in as well as output connections for data-out [6]. A common model for representing an SRAM is the two-dimensional model shown in Figure 2-2. This model displays the basic inputs for controls, addresses, and data-in and outputs for the data-out bits [6]. The data-in and data-out ports will be N bits wide where N will be the width of the data words in the RAM.

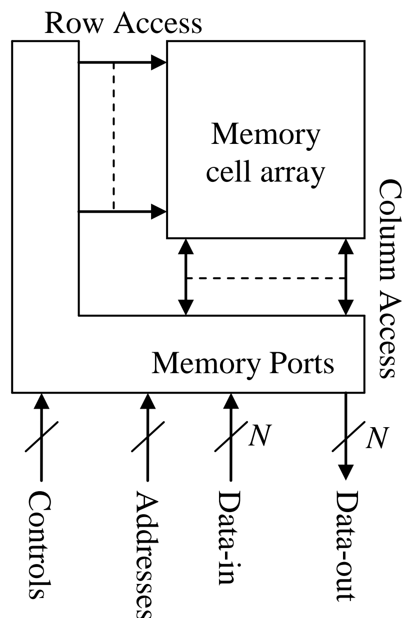


Figure 2-2 - Two-Dimensional SRAM Model [6]

The RAMs contained within the Virtex-5 FPGAs function as multi-port SRAMs. A multi-port RAM has multiple input and output ports. These ports may be read-only, write-only, or capable of both read and write operations [6]. The detailed functional operating model of an SRAM can be seen in Figure 2-3. This model illustrates how the row and column decoders will be used in order to select the memory location [6]. The control circuitry, read/write circuits, and data registers are then used to either extract or insert bits into the array [6]. In the multi-port SRAM the ports are able to read and write simultaneously in all but a few circumstances in which ports may be trying to read and write to the same address [6]. The ports share a common memory cell array that is constructed of individual memory cells. The address inputs are used by the row and column decoders in order to select a cell in the memory on which the read or write operation will be executed. When a write instruction is executed the data word on the data-in pins is written into the SRAM memory cells at the selected address. When a read instruction is

executed the data word saved at the selected address is retrieved from the memory and displayed on the data-out pins [6].

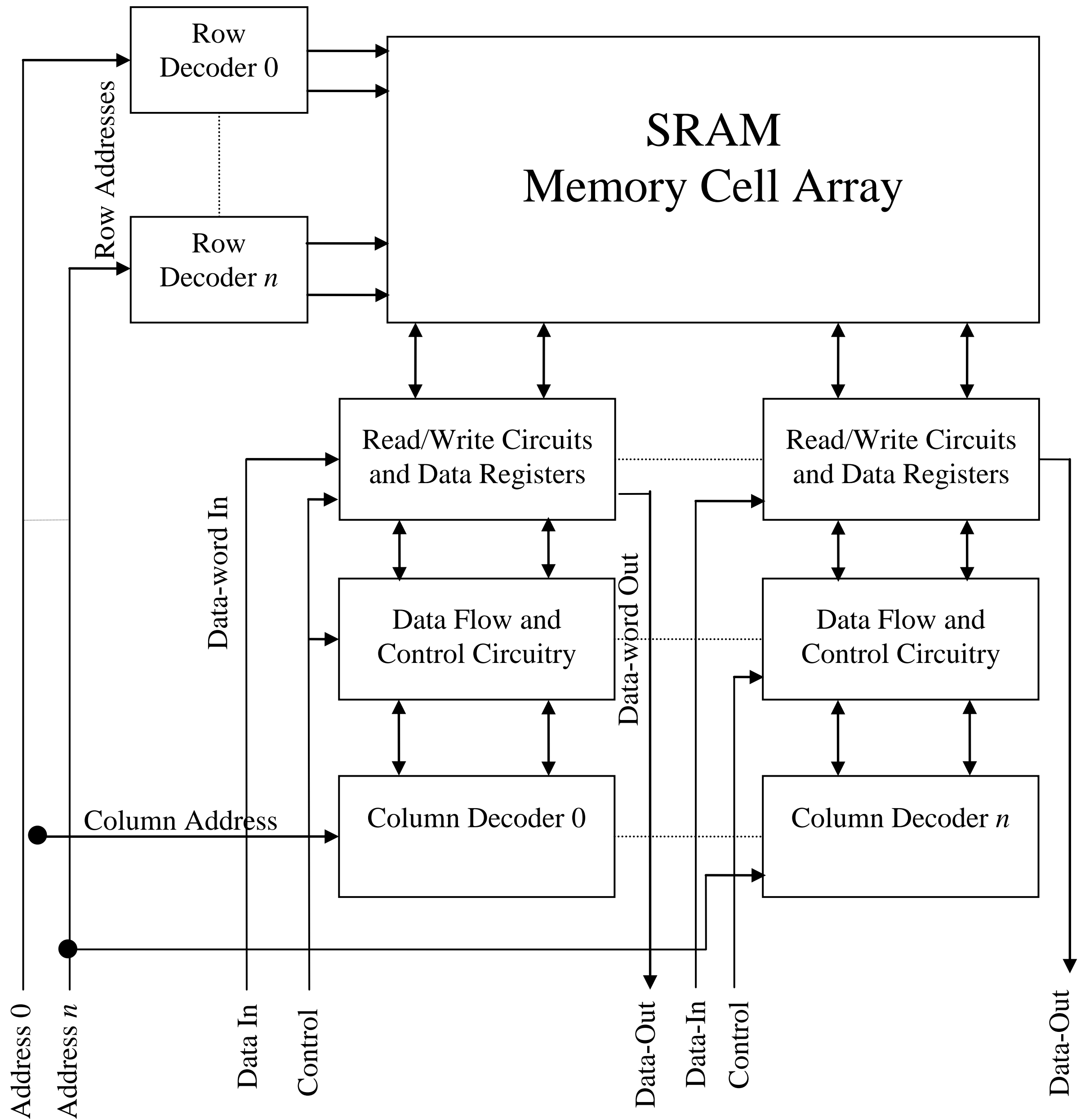


Figure 2-3 - Functional model of a multi-port memory [6]

2.2.1 SRAM Faults

In order to simplify fault testing in a memory a reduced functional memory model is used to model the operation of the memory [6]. This reduced model only consists of the address decoder, the memory cell array, and the read/write logic. These three subsystems are common to almost all mainstream memory devices [6]. In order to describe failures within a memory a set of functional fault models is defined. The functional models are described as the difference between the observed behavior and the expected behavior under a set of performed operations [6]. This means that to define any fault model two things are needed [6]:

- A list of performed memory operations
- A list of the differences in behavior observed when performing the operations

The behavior of these fault models is described by fault primitives. Each primitive is used in order to describe a fault and consists of the pattern of inputs used to sensitize the fault and the resulting faulty behavior [6]. An extremely limited subset of the most relevant primitives is selected to describe the faulty behavior of the memory rather than testing all functional specifications [6]. The fault primitives are classified according to four separate criteria, as follows.

2.2.1.1 Static vs. Dynamic Faults

Static faults are fault primitives which only require a single read or write operation in order to detect [6]. Examples of static faults are cell values being stuck-at-1 or stuck-at-0. Dynamic faults require more than a single read or write operation to expose and can be classified further by the number of operations required [6].

2.2.1.2 Simple vs. Linked Faults

Simple faults are faults that are unable to influence each other in any way. However, when faults are able to influence the behavior of other faults they are classified as linked faults [6]. This behavior means that linked faults are capable of masking each other [6]. When masking occurs, the effect of one fault will result in the faulty result of another becoming unobservable [6].

2.2.1.3 Single-port vs. Multi-port Faults

Single-port faults are fault primitives that only require usage of, at the most, one port of the RAM. Multi-port faults require the use of two or possibly more ports in order to sensitize the fault. These faults may be further classified based on the number of ports that are needed [6].

2.2.1.4 Single-cell vs. Multi-cell Faults

A fault is characterized as a single-cell fault if the cell that is used for sensitizing the fault is also the same cell in which the fault is observed [6]. Multi-cell or coupling faults involve more than a single cell to sensitize. For multi-cell faults the cell in which the operation is performed is different than the cell in which the fault is observed [6].

2.2.2 RAM Test Algorithms

In the standard dual-port mode of operation of the BRAMs in Virtex-5 devices, two known RAM tests are used to test the memories: *March s2pf/d2pf* and *MATS+* [6]. These tests are executed on the RAM with various port widths in this configuration.

The notation that will be used to describe the RAM test algorithms is as follows [6]:

- ↑,↓: Used to indicate the direction traveled through the address space (↕ indicates that the address space may be traversed in either direction).
- r, w: Used to denote between read and write operations. These characters will be directly follow by the values to be written or the values expected to be read.
- Each group of operations within parenthesis is known as a march element. All operations in these parentheses will be performed on a single address.
- Example: ↓ (r0, w1) indicates that the test will traverse the address space from the maximum address to the minimum. At each location address a *Read – 0* operation will be performed followed by a *Write – 1* operation.

Some additional notations are used for dual port RAM tests [6]:

- A colon (:) separates operations of the separate ports
- *n* : Used to indicate that no operation is to be applied on a port.
- - : Used to indicate that any operation may be used, as long as it is not in conflict (i.e. dual write operations to the same address location with different values)
- $n = 0^{N-1}$: Used to indicate that an operation is performed on either a row or column range. Where N will be R for a row range and C for a column range.

The *MATS+* algorithm was chosen to be used on the various port widths of this RAM configuration. It was selected because it is a simple algorithm which can quickly verify the address and data widths and the programmable address decoding circuitry [4]. *MATS+* is order O(5*N*) and the full algorithm can be seen in Equation 2-2 [6]. This algorithm is used to test the

programmable address and data widths, write modes, the active levels of the clock, port enable, output register clock, and the set/reset signals [3].

$$MATS+ = \{\updownarrow (w0); \uparrow (r0, w1); \downarrow (r1, w0)\} \quad 2\text{-}2$$

When testing word-oriented memories, such as the BRAMs in Virtex-5 devices, background data sequences (BDS) are needed to detect faults within the memory words. The number of BDS required for testing a memory can be seen in Equation 2-3 where K is the number of bits in the data word [4].

$$N_{BDS} = [log_2 K] + 1 \quad 2\text{-}3$$

For example in order to incorporate a 4-bit BDS into the *MarchLR* algorithm, first replace all single bit elements in Equation 2-4 with 4-bit words. The *r0, r1, w0,* and *w1* elements will be replaced with *r0000, r1111, w0000,* and *w1111* respectively. Then by using Equations 2-5 and 2-6 along with Table 2-1 and Table 2-2, the BDS can be constructed using the following process [10]:

$$MarchLR =$$

$$\{\updownarrow (w0);$$

$$\downarrow (r0, w1);$$

$$\uparrow (r1, w0, r0, w1);$$

$$\uparrow (r1, w0);$$

$$\uparrow (r0, w1, r1, w0);$$

$$\uparrow (r0)\}$$

2-4

$$BDS_1 = r_{Di,} w_{Di+1}, r_{Di+1}$$

2-5

$$BDS_2 = r_{Di,} w_{Di+1}, w_{Di+2}, r_{Di+2}$$

2-6

Table 2-1 – 4-Bit BDS Components

Normal	Inverse
0000	1111
0101	1010
0011	1100

Table 2-2 – 4-Bit BDS Sequence

i	D
0	0000
1	1111
2	0000
3	0101
4	1010
5	0101
6	0011
7	1100
8	0011

1. Starting with *i* = 0 in Table 2-2, use Equation 2-5 to get (r_0, w_1, r_1) and the resulting march element $\{\uparrow(r0000, w1111, r1111\}$.
2. Using *i* = 1, the equation results in (r_1, w_2, r_2) and the next march element is $\{\downarrow(r1111, w0000, r0000\}$.
3. Using *i* = 2, notice that from *i* = 2 to *i* = 3 there is a transition from the first row of Table 2-1 to the second row. Therefore Equation 2-6 is used to create the march element rather than Equation 2-5. The resulting equation is (r_2, w_3, w_4, r_4) and the march element will be $\{\uparrow(r0000, w0101, w1010, r1010\}$. When a transition such as this occurs *i* will be incremented by 2.
4. Using *i* = 4 Equation 2-5 is used because there is no transition of rows between *i* = 4 and *i* = 5. The resulting equation will be (r_4, w_5, r_5) and the march element will be $\{\downarrow(r1010, w0101, r0101\}$.
5. Using *i* = 5, Equation 2-6 will be used due to the transition between *i* = 5 and *i* = 6, which

results in the equation (r_5, w_6, w_7, r_7) and the march element

$\{\uparrow(r0101, w0011, w1100, r1100\}$.

6. Using *i* = 6, Equation 2-5 will be used to get (r_7, w_8, r_8) and the march element will be

   $\{\downarrow(r1100, w0011, r0011\}$.

7. The final march element will be a read operation of the final *i* value: $\{\uparrow(r0011\}$.

The resulting *MarchLR w/4-bit BDS* algorithm is O(35*N*). However, the seventh and eight march elements of this generated test algorithm repeat march elements contained within the initial *MarchLR* algorithm [10]. In order to optimize our *MarchLR w/BDS* algorithm we may eliminate these duplicated elements and we will be left with the optimized algorithm which is O(30*N*). The optimized algorithm is shown in Equation 2-7 [10].

$$MarchLR\ w/4\text{-}bit\ BDS =$$

$$\{\updownarrow(w0000);$$

$$\downarrow(r0000, w1111);$$

$$\uparrow(r1111, w0000, r0000, r0000, w1111);$$

$$\uparrow(r1111, w0000);$$

$$\uparrow(r0000, w1111, r1111, r1111, w0000);$$

$$\uparrow(r0000, w0101, w1010, r1010);$$

$$\downarrow(r1010, w0101, r0101);$$

$$\uparrow(r0101, w0011, w1100, r1100);$$

$$\downarrow(r1100, w0001, r0011);$$

$$\uparrow(r0011)\}$$

2-7

The *March Y* algorithm is used in order to test the programmable address decoding circuitry of the BRAM [4]. This algorithm will also detect destructive read faults within the BRAM [4]. The *March Y* algorithm is order O(8*N*) and can be seen in Equation 2-8 [4]. In order to test the FIFO mode of operation, as well as the programmable flags in this mode, the *March X* algorithm is used [4]. This algorithm is O(6*N*) and is shown in Equation 2-9 [4].

$$March\ Y = \{\updownarrow (w0); \uparrow (r0, w1, r1); \downarrow (r1, w0, r0); \uparrow (r0)\} \quad (2\text{-}8)$$

$$March\ X = \{\updownarrow (w0); \uparrow (r0, w1); \downarrow (r1, w0); \uparrow (r0)\} \quad (2\text{-}9)$$

In order to fully test the programmable "almost" full and "almost" empty in the First-In-First-Out (FIFO) mode of operation the RAM must be reconfigured multiple times as described in [4]. The steps in the *FIFOX* algorithm are shown below:

Step 1. *Reset the FIFO, check that Empty flag is active*

Step 2. *Repeat N times: write FIFO with all 0's, check that Empty flag goes inactive after first write cycle, Full flag goes active after last write cycle, and that Almost Empty flag*

goes inactive and Almost Full flag goes active at the appropriate points in the sequence. Perform one additional write if the FIFO has a Write Error signal to indicate an attempted write when the FIFO is full.

Step 3. Repeat N times: read FIFO expecting all 0's and write FIFO with all 1's, check that Full flag toggles after each read and write cycle.

Step 4. Repeat N times: read FIFO expecting all 1's and write FIFO with all 0's, check that Full flag toggles after each read and write cycle.

Step 5. Repeat N times: read FIFO expecting all 0's, check that Full flag goes inactive after first read cycle, Empty flag goes active after last read cycle, and that Almost Empty flag goes active and Almost Full flag goes inactive at the appropriate points in the read sequence. Perform one additional read if FIFO has a Read Error signal to indicate an attempted read when the FIFO is empty.

The *March s2pf/d2pf* algorithms were chosen because they are able to detect all realistic single and double addressing faults within a dual port RAM [6]. *March s2pf* is order O(14*N*), and *March d2pf* is order O(9*N*) where *N* represents the number of addresses in the memory. The *March s2pf* and *d2pf* algorithms may be seen in Equation 2-10 and 2-11 respectively [6]. These algorithms are responsible for testing the dual-port functionality of the BRAMs [6].

$$
\begin{aligned}
&\textit{March s2pf} = \\
&\{\updownarrow (w0 : n); \\
&\uparrow (r0 : r0, r0: -, w1 : r0); \\
&\uparrow (r1 : r1, r1 : -, w0: r1); \\
&\downarrow (r0 : r0, r0 : -, w1 : r0); \\
&\downarrow (r1 : r1, r1 : -, w0 : r1); \\
&\downarrow (r0 : -)\}
\end{aligned}
\tag{2-10}
$$

$$
\begin{aligned}
&\textit{March d2pf} = \\
&\{\updownarrow (w0 : n); \\
&\uparrow c = 0^{C-1} (r = 0^{R-1} (w1_{r,c} : r0_{r+1,c,} r1_{r,c} : w1_{r+1,c}, w0_{r,c} : r1_{r+1,c}, r0_{r,c+1} : w0_{r+1,c})); \\
&\uparrow c = 0^{C-1} (r = 0^{R-1} (w1_{r,c} : r0_{r+1,c,} r1_{r,c} : w1_{r+1,c}, w0_{r,c} : r1_{r+1,c}, r0_{r,c+1} : w0_{r+1,c}))\}
\end{aligned}
\tag{2-11}
$$

The *ECC (Write)* and *ECC (Read)* algorithms both use an ECC testing algorithm described in [4] which achieves 100% coverage of an XOR parity tree circuit which the ECC circuitry is surmised to be. The *ECC Write* algorithm is responsible for testing the parity generation circuitry while the *ECC Read* algorithm is responsible for testing the error detection and correction circuitry [4].

ECC (Write) =

All 0's; walk 1-thru-0's

All 1's

Walk two 1's-thru-0's

2-12

ECC (Read) =

Output of ECC generate vectors

Init: Walk 1-thru-0's; all 1's; all hamming values w/data = 0's

Init: Walk two 1's-thru-0's

(Note: ***Init*** indicates that the test vectors are initialized in the ECC RAM during download)

2-13

2.3 Virtex-5 Architecture

This section will detail the structural architecture of the Virtex-5 devices. The programmable logic resources available to the user will be discussed along with information about the design and functionality of the BRAMs contained in the devices.

2.3.1 Virtex-5 Configurable Logic Blocks

The primary resources in the Virtex-5 for implementing sequential and combinational logic circuits are the Configurable Logic Blocks (CLBs) [7]. The resource count within these CLBs is shown in Table 2-3. Each CLB contains a pair of slices. These two slices are not interconnected and are arranged in two columns containing a dedicated carry chain as summarized in Figure 2-4. The slices are also connected to a switching matrix, granting them access to the general routing matrix [7].

Table 2-3 - CLB Resources in Virtex-5 [7]

Component	**Virtex-5 CLB**
Slices	2
Look-Up-Tables	8 (6-input)
Flip-Flips	8
Arithmetic and Carry Chains	2
Distributed RAM	256-bits
Shift Registers	256-bits

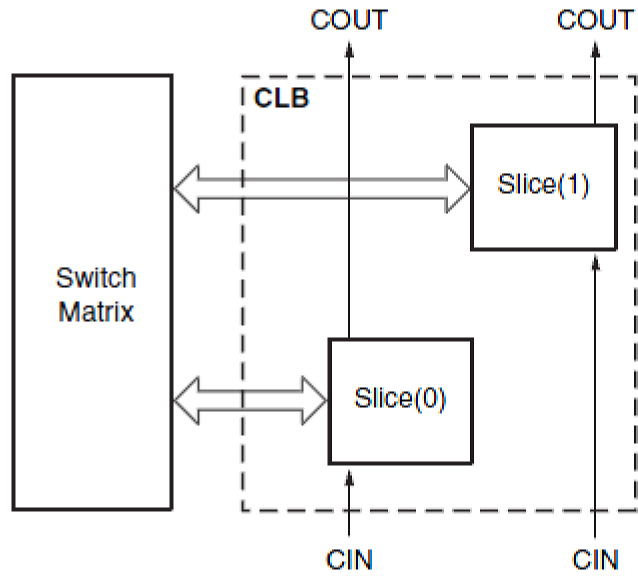


Figure 2-4 - Slice arrangement within Virtex-5 CLBs [7]

2.3.2 Virtex-5 Block RAMs

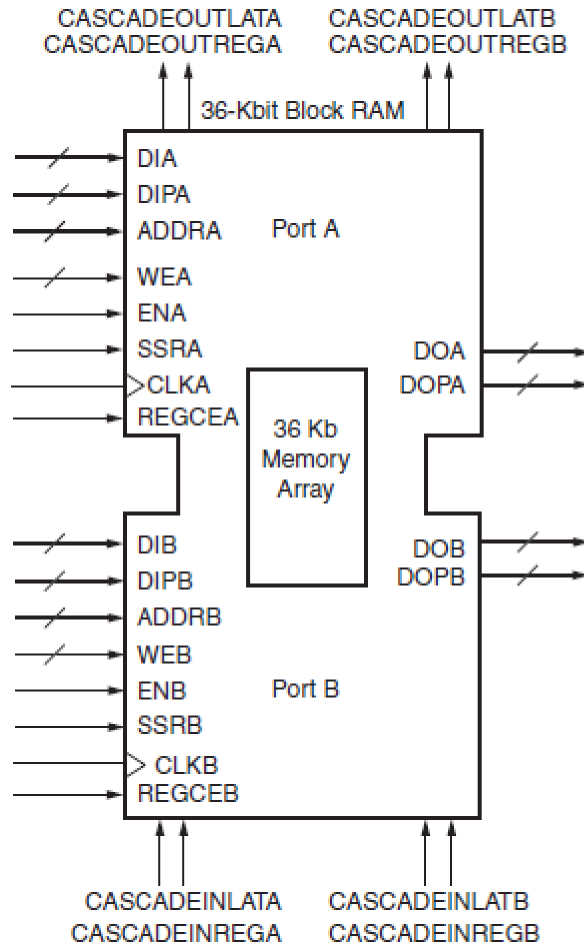


Figure 2-5 - Virtex-5 Dual-Port Flow [7]

The BRAMs contained within Virtex-5 devices are capable of operating in two main modes, single-port and dual-port [7]. The inputs and outputs available to the BRAMs can be seen in Figure 2-5 [7]. Each RAM may be used as be used as two separate 18 K-bit RAMs or as a single 36 K-bit RAM [7]. The RAMs contain two input ports, Port A and Port B. These two ports may be used independently to synchronously read data from and write data to the RAM. The RAMs may be configured to use one of three write configurations [7]. In the *WRITE_FIRST* mode, data will be immediately displayed on the output of the RAM as it is written. The

READ_FIRST mode will display the previous contents of the RAM on the output as new data is written. Finally, the *NO_CHANGE* mode will leave the outputs unchanged upon a write operation. When using this mode the data output remains the last read data and is unaffected by write operations [7]. Additionally, the RAMs may be used in either a single-port or dual-port RAM mode [7]. This option is available whether the RAMs are being used as a single 36 K-bit memory or two independent 18 K-bit memories [7]. The RAMs are also able to be configured with various port sizes and depths. The different configuration types available to be used for the independent 18 K-bit configurations are shown in Table 2-4 [7]. The configuration types available for use with the single 36 K-bit configurations are shown in Table 2-5 [7]. An option is also provided to enable a pipeline register on the output of a RAM, allowing a higher operating frequency while sacrificing an additional clock cycle of latency [7].

Table 2-4 - Virtex-5 BRAM Port Aspect Ratio (18K-bit RAM) [7]

Address Width	Address Bits	Memory Depth	Data Width	Data-In/Out Bits	Data-In/Out Parity Bits
14	13:0	16K	1	0	n/a
13	13:1	8K	2	1:0	n/a
12	13:2	4K	4	3:0	n/a
11	13:3	2K	9	7:0	0
10	13:4	1K	18	15:0	1:0
9	13:5	512	36	31:0	3:0

Table 2-5 – Virtex-5 BRAM Port Aspect Ratio (36K-bit RAM) [7]

Address Width	Address Bits	Memory Depth	Data Width	Data-In/Out Bits	Data-In/Out Parity Bits
15	14:0	32K	1	0	n/a
14	14:1	16K	2	1:0	n/a
13	14:2	8K	4	3:0	n/a
12	14:3	4K	9	7:0	0
11	14:4	2K	18	15:0	1:0
10	14:5	1K	36	31:0	3:0
9	14:6	512	72	63:0	7:0

The RAMs also may be used with ECC parity bits. In this mode of operation an 8-bit Hamming code is generated by the ECC circuitry present with each RAM which can be seen in Figure 2-6 [7]. The ECC circuitry may be used fully or in an encoder-only or decoder-only mode. The configurable *EN_ECC_WRITE* option allows the ECC bits to be provided on the parity input pins of the RAM or optionally generated by the included encoding circuitry. Similarly the *EN_ECC_READ* option may be used to bypass the decoding and correction circuitry [7]. This ECC circuitry is capable of detecting and correcting any single-bit error or detecting any double-bit error without correction in the data being read from the RAM [7].

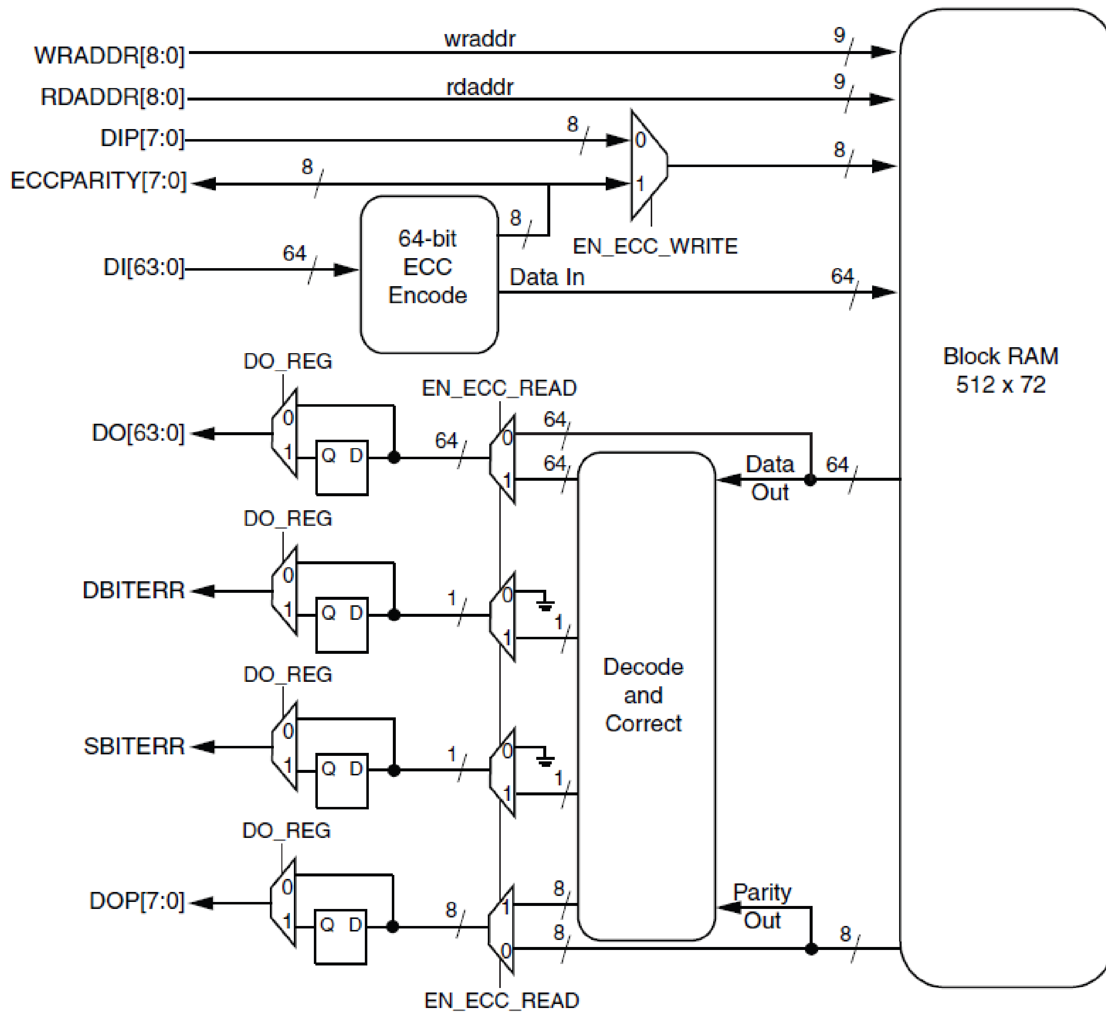


Figure 2-6 - Top Level View of Virtex-5 BRAM ECC [7]

The RAMs may also be configured in a First-In-First-Out (FIFO) mode of operation. In this mode of operation the FIFO is equipped with the inputs and outputs shown in Table 2-6 [7]. The FIFO provides separate read and write enables as well as individual clocks for each operation. The read and write addresses are displayed on outputs, and there are also flags indicating a read error or a write error. A pair of flags indicating that the FIFO is full or empty is present. Also, the FIFO features configurable almost full and almost empty flags which are controlled via a 13-bit hexadecimal value [7]. A configuration option called

FIRST_WORD_FALL_THROUGH also optionally allows the first word written into the FIFO to be immediately displayed on the output. If this option is selected the capacity of the FIFO will be increased by one [7]. The FIFO may also be used with the same ECC circuitry that is available to the standard BRAM configuration. This allows any single bit error in the FIFO data to be detected and corrected or any double bit error in the FIFO data to be detected [7]. The various port aspect ratios and memory depths that are available for use in this mode are shown in Table 2-7 [7]. Additionally, the actual capacity of the FIFO when it is used with these different port aspect ratios is shown in Table 2-8 [7].

Table 2-6 - FIFO Input and Output Ports [7]

	Port	Width	Description
Inputs	DI	32	Data input
	DIP	4	Parity-bit input
	RDEN	1	Read enable
	RDCLK	1	Read domain clock
	WREN	1	Write enable
	WRCLK	1	Write domain clock
	RST	1	Asynchronous reset
Outputs	DO	32	Data output
	DOP	4	Parity-bit output
	WRCOUNT	13	Data write pointer
	RDCOUNT	13	Data read pointer
	FULL	1	Full flag
	EMPTY	1	Empty flag
	ALMOSTFULL	1	Configurable almost full flag
	ALMOSTEMPTY	1	Configurable almost empty flag
	RDERR	1	Read error flag
	WRERR	1	Write error flag

Table 2-7 – Virtex-5 FIFO Port Aspect Ratio [7]

18K-bit Mode		36K-bit Mode	
Memory Depth	Data Width	Memory Depth	Data Width
4K	4	8K	4
2K	9	4K	9
1K	18	2K	18
512	36	1K	36
-	-	512	72

Table 2-8 – Virtex-5 FIFO Data Depth [7]

Data Width		Block RAM Memory	FIFO Capacity	
18K-bit	36K-bit		Standard	FWFT
-	4	8192	8193	8194
4	9	4096	4097	4098
9	18	2048	2049	2050
18	36	1024	1025	1026
36	72	512	513	514

Finally, the BRAMs are able to be configured in a cascade mode which allows two adjacent RAMs to be connected together and used as one larger RAM. The circuitry which allows this is shown in Figure 2-7 [7]. This option is available for any two adjacent RAMs in a column on the device [7]. The only port width available for this operating mode is 64K x 1-bit where two 32K x 1-bit RAMs are combined. The upper RAM has its *RAM_EXTENSION* configuration bit set to UPPER (0) and the lower ram has its *RAM_EXTENSION* bit set to LOWER (1) [7]. Output data is only displayed on the upper RAM. The data output of the RAM configured as the lower RAM is routed into a multiplexer by connecting the *CASCADEIN* and *CASCADEOUT* of the two RAMs as shown in Figure 2-7 [7]. This multiplexer is controlled by address bit A15 which selects the appropriate output [7].

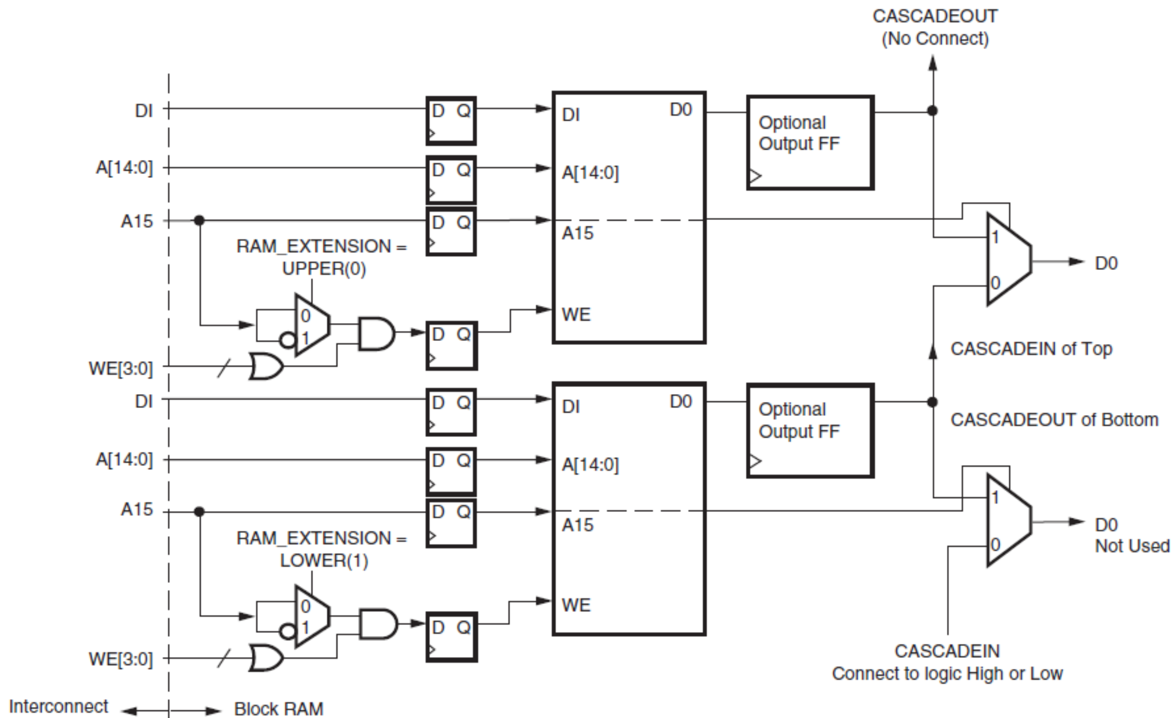


Figure 2-7 - Virtex-5 BRAMs in Cascade Configuration [7]

2.4 Virtex-4 Block RAM BIST

This section will discuss the FPGA RAM BIST procedures developed by Milton in [2] and by Garrison in [3] for Virtex-4 which are expanded upon in this work for a Virtex-5 implementation. Milton's original approach used the CLBs available within the FPGA to create the TPG and ORAs while the BRAMs served as the Circuits Under Test (CUTs). Milton also used a pair of identical TPGs which provide test vectors to alternating RAMs in the columns [2]. The ORAs are implemented using a circular comparison based approach that results in an increase in fault detection capability and diagnostic resolution [2]. These ORAs are placed in the CLB columns which neighbor the BRAMs. A layout of Milton's BIST architecture along with the TPG, CUT, and ORA connections can be seen in Figure 2-8 [2].

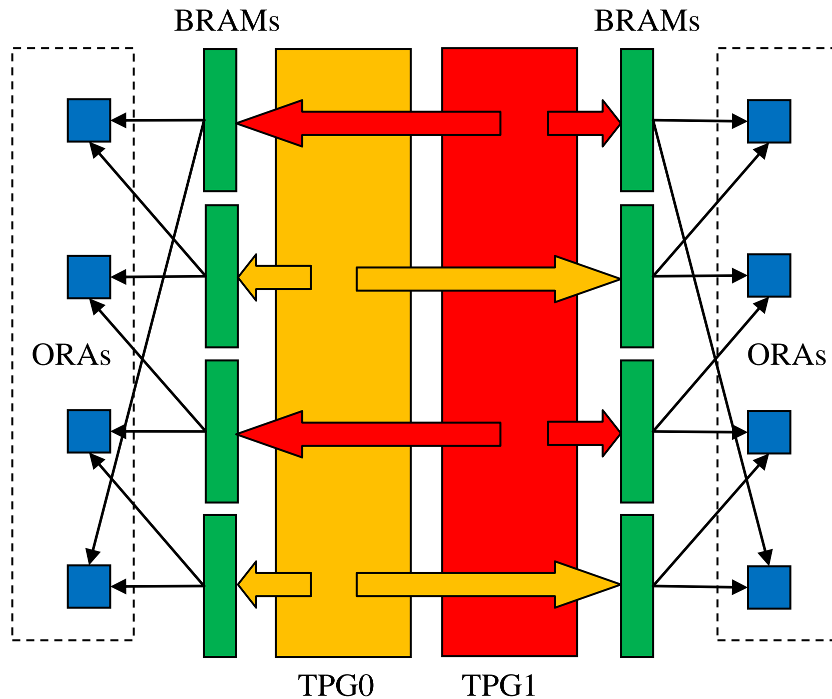


Figure 2-8 - BRAM BIST Architecture [2]

2.4.1 Dedicated Carry Chain

Milton's original ORA design was improved upon by Dutton in [11] and later used by Garrison in [3] to take advantage of the built in carry logic provided in the Configurable Logic Blocks (CLBs) of Virtex-4 and Virtex-5 devices. In order to implement this, the ORA circuitry was modified to that shown in Figure 2-9 [3]. To indicate a fault has been detected a Logic 0 is latched into the flip-flop [3]. This bit is used to select the input of a multiplexor in the carry chain which in turn provides a Logic 1 on the *carry-out* in the case of a failure. Alternatively, the input that is provided from the previous multiplexor via *carry-in* is forwarded to the *carry-out* [3].

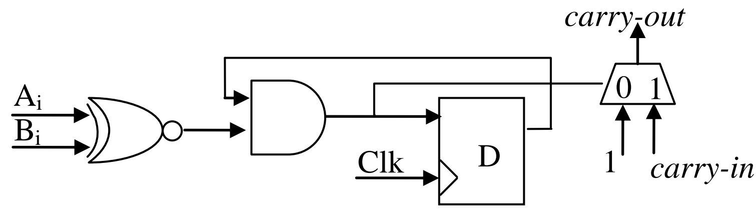


Figure 2-9 - Comparison Based ORA with Carry Chain [3]

In order to ensure the propagation of the test result through the entire built-in carry logic, several dummy ORAs must be implemented in the ORA columns[3]. This is necessary because some of the ORA columns do not span the entire height of the FPGA. In this case, the dummy ORAs are added to the configuration to complete these columns as seen in Figure 2-10. No logic is implemented in the dummy ORAs aside from the built-in carry chain [3].

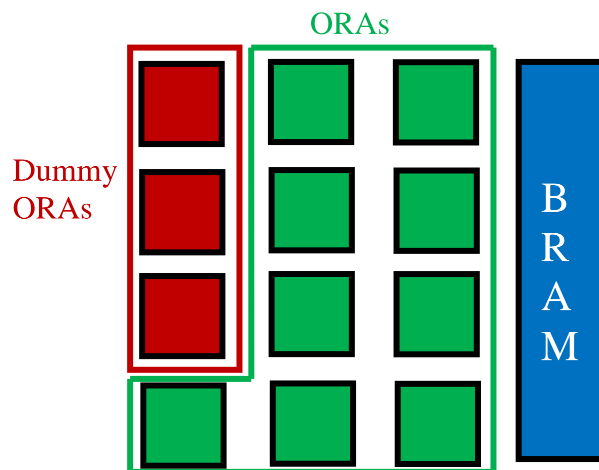


Figure 2-10 - Additional Dummy ORAs [3]

The functionality of the carry logic can be expressed as an iterative OR-chain as seen in Figure 2-11 where the boxes containing '*O*'s are the ORAs [3]. If no mismatch is detected within the ORA then the input from the previous ORA will be selected. If a mismatch is detected a Logic 1 will be output by the detecting ORA and propagated through the chain [3].

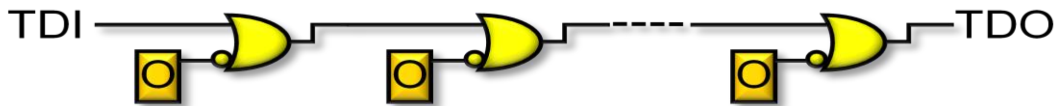


Figure 2-11 - Iterative OR-Chain Functionality [3]

The OR-chain is connected to the boundary scan interface provided on the device, with the initial input being provided by the *Test Data In* (*TDI*) pin. The final output of the chain is connected to the *Test Data Out* (*TDO*) pin of the interface [3]. The OR-chain effectively provides a single *Pass/Fail* bit to observe the test result. Once the test has concluded the user is able to toggle *TDI* and observe the behavior of *TDO* [3]. If *TDO* matches *TDI* during this process then no fault has been detected by any of the ORAs, and it is unnecessary to perform a configuration memory read back [3]. If *TDO* is observed as being constantly a Logic 1 through this process, then a configuration memory read back may be performed in order to retrieve the results from the flip-flops in the ORAs if desired [3]. If *TDO* is observed as being constantly a Logic 0 while toggling *TDI* then it must be assumed that there is a fault within the logic used to construct the OR-chain meaning *TDI* and the ORA comparison results are not being properly propagated.

2.4.2 TPG Architecture

The original designs for the Virtex-5 BRAM TPGs were proposed by Garrison in [3]. Garrison proposed that four different Xilinx BRAM primitive models be used in developing and

testing the TPGs. These models describe the operation of the BRAMs in different configuration modes and behave exactly as the physical BRAMs in simulation [7]. The TPGs are created with an aim to test the BRAMs in these modes of operation, and the operation of the TPGs is verified with these models in simulation. The models for the first four test configurations are as follows [3]:

1. BRAM (32K + 4K parity) – true dual-port BRAM that supports widths of x1, x2, x3, x4, x9, x18, and x36.
2. ECC (512 x 72-bit) – simple dual-port BRAM with 64-bit ECC.
3. FIFO (32K + 4K parity) – synchronous/asynchronous FIFO BRAM that supports widths x1, x2, x4, x9, and x18.
4. FIFOECC (512 x 72-bit) – synchronous/asynchronous FIFO with 64-bit ECC.

2.4.2.1 BRAM TPG

The TPG proposed by Garrison for testing the RAM in the BRAM configuration is responsible for testing the dual-port functionality of the BRAM and would require seven different BIST configurations [3]. The proposed test algorithm, address space, and data width used for each configuration can be seen in Table 2-9 [7]. The desired test to be run is selected by a user-supplied control string that is shifted into the TPG using the boundary scan interface as shown in Figure 2-12 [3]. The values proposed for the control strings for the various tests are shown in Table 2-10 along with the configuration settings in Table 2-11 [3]. The three *Mode* bits in the control string correspond to the BRAM *Configuration Number* and the *Level Control* bit allows us to control the active level for the TPGs [3].

Execution of the RAM test algorithms is implemented using a Finite State Machine (FSM) in a VHDL model for the BRAM TPG as well as the other BIST TPGs. The TPG model is synthesized using area constraints to restrict the placement of the resources to the smallest area possible in the lower left hand corner of the device [3]. In the BIST configurations, the TPG designs will be offset from the bottom left hand corner to achieve the desired placement in the six CLB columns directly to the right of the rightmost column of BRAMs, excluding the columns of BRAMs located in a Tri-mode Ethernet Media Access Controller (TEMAC) column in Virtex-5 devices that contain them [3]. The columns are selected for TPG placement because they are not used for any other purpose in the Virtex-5 BRAM BIST [3]. In these columns, one TPG will be placed at the bottom of the device and the other TPG will be placed exactly halfway up the device [3]. The TPGs are placed in this manner in order to minimize routing distance from each TPG to alternating BRAMs in columns spanning the entire height of the device. An example of the TPG placement and routing on the LX30 device may be seen in Figure 2-13.

Table 2-9 - BRAM BIST Configurations [7]

BRAM Config	Test Algorithm	Address Space	Data Width
1	March s2pf	1K	36
2	March d2pf	1K	36
3	MATS+	2K	18
4		4K	9
5		8K	4
6		16K	2
7		32K	1

Table 2-10 – Proposed Control String Values for BRAM TPG [3]

BRAM Config	Test Algorithm	Address Space	Level Control	Mode 2	Mode 1	Mode 0	Hex Control String
1	March s2pf	1K	0	0	0	0	0x0
2	March d2pf	1K	0	0	0	1	0x1
3	MATS+	2K	1	0	1	0	0xA
4		4K	1	0	1	1	0xB
5		8K	1	1	0	0	0xC
6		16K	1	1	0	1	0xD
7		32K	1	1	1	0	0xE

Table 2-11 - Proposed Configuration Settings for BRAM TPG [3]

(a) Settings Part 1

BRAM Config	Test Algorithm	DO (A/B) REG	READ Width (A/B)	WRITE Width (A/B)	WRITE Mode (A/B)	SAVE DATA
1	March s2pf	1	36	36	READ_FIRST	FALSE
2	March d2pf	1	36	36	READ_FIRST	FALSE
3	MATS+	0	18	18	READ_FIRST	FALSE
4		0	9	9	WRITE_FIRST	FALSE
5		0	4	4	NO_CHANGE	FALSE
6		0	2	2	WRITE_FIRST	FALSE
7		0	1	1	NO_CHANGE	FALSE

(b) Settings Part 2

BRAM Config	Test Algorithm	CLK, EN, SSR REGCLK (A/B)(U/L) INV	RAM EXT	INIT VAL	SRVAL	INIT (A/B) VAL
1	March s2pf	INV	NONE	AAAA	5555	0
2	March d2pf	not INV	NONE	5555	AAAA	FFFF
3	MATS+	not INV	NONE	AAAA	5555	0
4		not INV	NONE	5555	AAAA	FFFF
5		not INV	NONE	AAAA	5555	0
6		not INV	NONE	5555	AAAA	FFFF
7		not INV	NONE	AAAA	5555	0

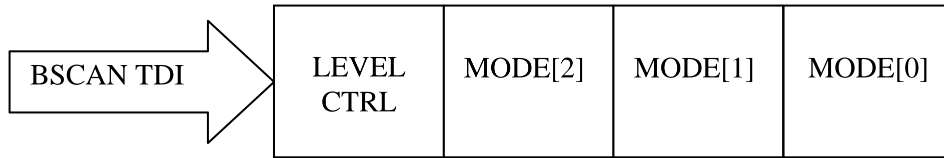


Figure 2-12 - Shift Register Control String for BRAM TPGs [3]

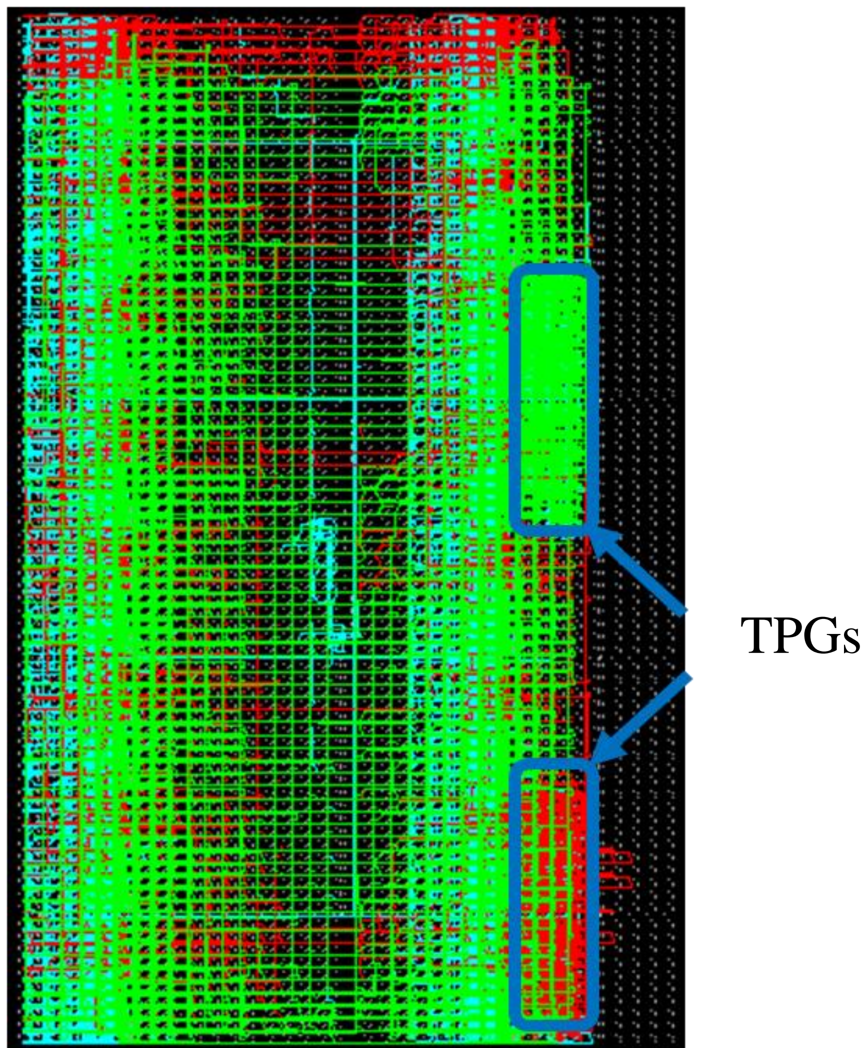


Figure 2-13 - Placement and Routing on TPGs in LX30

2.4.2.2 ECC TPG

The proposed ECC TPG is responsible for testing the memory core and the ECC read and write capabilities of the BRAM [3]. By using the largest data width available in this configuration, all memory elements within the BRAMs may be accessed [3]. This TPG is implemented using an FSM as well, and the algorithm to be run is also selected using a control string shifted in via the boundary scan interface. The control values used for selecting the algorithm are shown in Table 2-12 along with the entirety of the proposed configuration settings in Table 2-13 [3]. The RAM test algorithm *March LR w/72-bit BDS* (described in the Appendix) is used in this configuration since all memory elements are available. The background data sequence is used to ensure that all intra-word coupling faults will be detected [6]. The configurations labeled *ECC (read)* and *ECC (write)* are responsible for detecting any faults within the ECC check and correction circuitry on the BRAMs [4]. This TPG is also synthesized using area constraints to control TPG area and placement.

Table 2-12 – Proposed Control String Values for ECC TPG [3]

ECC Config	Test Algorithm	Level Control	Mode 1	Mode 0	Hex Control String
1	MarchLR w/BDS	0	0	0	0x0
2	ECC (read)	0	0	1	0x1
3	ECC (write)	1	1	0	0x6

Table 2-13 – Proposed Configuration Settings for ECC TPG [3]
(a) ECC Settings Part 1

ECC Config	Test Algorithm	DO REG	EN_ECC READ	EC_ECC WRITE	EN_ECC SCRUB	INIT VAL	SR VAL	INIT (A/B) VAL	SAVE DATA
1	MarchLR w/BDS	0	FALSE	FALSE	FALSE	AAAA	5555	0	FALSE
2	ECC (read)	1	TRUE	FALSE	FALSE	AAAA	5555	0	FALSE
3	ECC (write)	1	FALSE	TRUE	FALSE	5555	AAAA	FFFF	FALSE

(b) ECC Settings Part 2

ECC Config	Test Algorithm	RDCLK (U/L) INV	RDEN (U/L) INV	RDRCLK (U/L) INV	WRCLK (U/L) INV	WREN (U/L) INV	SSR (U/L) INV
1	MarchLR w/BDS	not INV	not INV	not INV	not INV	not INV	not INV
2	ECC (read)	not INV	not INV	not INV	not INV	not INV	not INV
3	ECC (write)	INV	INV	INV	INV	INV	INV

2.4.2.3 FIFO TPG

The TPG for testing the BRAM in the FIFO configuration mode is responsible for testing all the FIFO functionality and is designed like the previously described BRAM and ECC TPGs. This TPG will use the RAM test algorithm *FIFOX* [4]. The proposed configuration settings for this mode can be seen in Table 2-14 [3].

Table 2-14 – Proposed Configuration Settings for FIFO TPG [3]
(a) FIFO Settings Part 1

ECC Config	Test Algorithm	DO REG	DATA WIDTH	EN SYN	FWFT	RDCLK (U/L) INV	RDEN INV	RST INV
1	FIFOX	1	36	TRUE	TRUE	INV	INV	INV
2		1	18	FALSE	FALSE	not INV	not INV	not INV
3		0	9	TRUE	TRUE	not INV	not INV	not INV
4		0	4	FALSE	FALSE	not INV	not INV	not INV

(b) FIFO Settings Part 2

ECC Config	Test Algorithm	WRCLK (U/L) INV	WREN INV	ALMOST FULL OFFSET	ALMOST EMPTY OFFSET
1	FIFOX	INV	INV	5555	AAAA
2		not INV	not INV	AAAA	5555
3		not INV	not INV	5555	AAAA
4		not INV	not INV	AAAA	5555

2.4.2.4 FIFOECC TPG

The TPG for this mode is responsible for testing the ECC circuitry of the BRAM when it is configured for FIFOECC operation. This TPG will be designed similar to the previously described TPGs. The proposed test algorithm that will be used in this TPG is *FIFOX* [4]. The proposed configuration settings may be seen in Table 2-15.

Table 2-15 – Proposed Configuration Settings for FIFOECC TPG [3]
(a) FIFOECC Settings Part 1

FIFOECC Config	Test Algorithm	DO REG	EN_ECC READ	EN_ECC WRITE	EN SYN	FWFT	RST INV
1	FIFOX	1	TRUE	FALSE	FALSE	TRUE	INV
2	FIFOX	0	FALSE	TRUE	TRUE	FALSE	not INV

(b) FIFOECC Settings Part 2

FIFOECC Config	Test Algorithm	ALMOST EMPTY OFFSET	ALMOST FULL OFFSET	RDCLK (U/L) INV	RDRCLK (U/L) INV	RDEN INV	WRCLK (U/L) INV	WREN INV
1	FIFOX	5555	AAAA	INV	INV	INV	INV	INV
2	FIFOX	AAAA	5555	not INV	not INV	not INV	not INV	not INV

2.5 Thesis Statement

This chapter has presented the basics of fault modeling in SRAM memories. It has also shown and detailed the various test algorithms used when testing these memories. An overview of the architecture of Virtex-5 devices is also given along with a description of the embedded BRAMs and their modes of operation. The components of the BIST structure are also described with proposed configuration modes and settings for the BRAMs.

This thesis aims to implement and expand upon the configurations proposed by Garrison in [3] for the Virtex-5. Garrison's proposed configuration settings to test the first four configuration modes of the BRAMs are shown in this chapter, but the design was not implemented in his work. In Chapter 3, this thesis will describe the implementation of BIST for the Virtex-5 BRAMs which includes Garrison's proposed configurations and settings which have been expanded upon to completely test the embedded BRAMs in these devices.

Chapter 3 Virtex-5 Block RAM BIST

This chapter will describe the design and implementation of the BIST for Virtex-5 BRAMs and the results obtained from actual generation and execution of the BIST sequence. This will include TPG development for all BIST configurations as well as the configuration settings for each of the operating modes. The design, placement, and routing of the ORAs is also shown along with an overview of the complete Virtex-5 BIST architecture. The process for generation and modification of the BIST configurations and the software tools used are also described. Finally the results and analysis will be presented including optimization, timing analysis, and fault coverage results.

3.1 Virtex-5 RAM BIST

The BIST architecture builds upon the architecture used by Milton and Garrison for Virtex-4 as described in Section 2.4. The same basic architecture is used where a pair of identical TPGs is used to drive the alternating BRAMs in the columns as shown in Figure 2-8. All BRAMs will be configured identically so any mismatch detected by an ORA is known to be a fault in a BRAM. The redundancy of the TPGs prevents fault aliasing that may occur when using a single TPG that has been synthesized containing a fault [12]. In the case of a fault being present in a TPG it will produce failures. These failures will be detected when the results of the

BRAMs being driven by the faulty TPG are compared with those from the BRAMs being driven by the fault-free TPG [12]. A circular comparison architecture which will be described later in this chapter is used for the ORA routing in order to prevent additional fault aliasing that may occur if adjacent BRAMs have identical faults.

3.2 TPG Design and Implementation

The TPGs for the five BIST configuration modes were designed as Finite State Machines (FSMs) to accommodate the multiple test phases that each TPG must run. The TPG designs were written as VHDL models and synthesized for insertion into the BIST configurations. Area constraints were used during synthesis of all the TPG models in order to minimize the resource usage of each one and restrict placement to the lower left hand corner of the device as shown in Figure 3-1. Designs are offset from this position to specify placement as described in Section 2.4.2. Prior to running the BIST procedure it is necessary to shift in the appropriate control string value for the desired phase of the test to be run. This is done via the BSCAN interface of the device. The data shifted in is consists of a level control value to specify the active level of the clocks and mode values to specify the phase of BIST the TPG will execute.

The TPG models for the BRAM, ECC, FIFO, and FIFOECC test configurations were all implemented based on the TPG models proposed by Garrison in [3] which are described in Chapter 2.

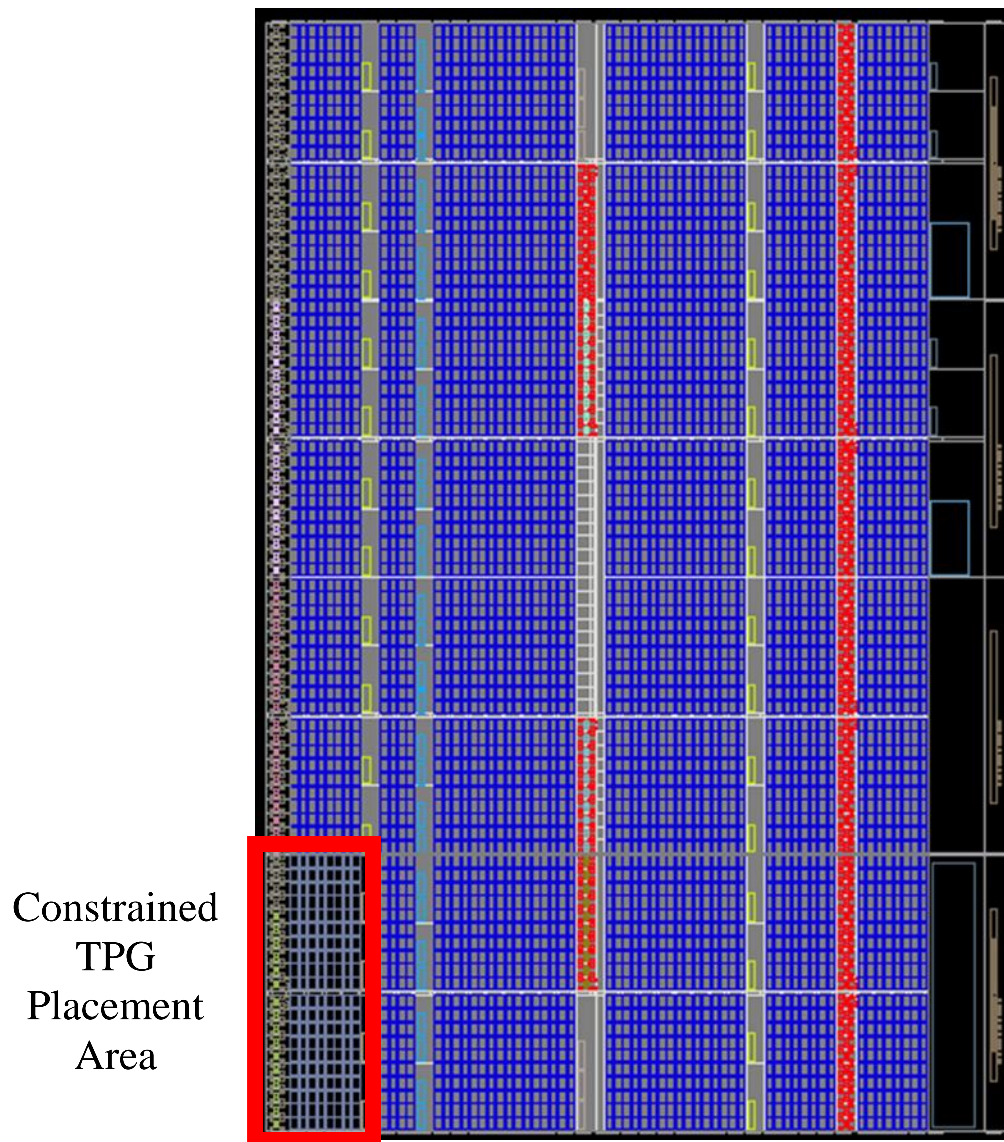


Figure 3-1 - BRAM TPG Area Constraints in LX30

3.2.1 BRAM

The final implementation of the BRAM TPG was based on Garrison's design in [3] with some minor modifications. The BIST configurations and control string values proposed by Garrison are used, but the proposed configuration settings were modified slightly by changing

the write mode of the last configuration to *READ_FIRST*. The final configuration settings for the BRAM TPG can be seen in Table 3-1. Prior to the BIST being run a control string value must be shifted in via the BSCAN interface in order to specify which phase of the test is to be run. For the BRAM TPG, the appropriate value for each test phase is shown in Table 2-10.

Table 3-1 - Final BRAM Configuration Settings
(a) Settings Part 1

BRAM Config	**Test Algorithm**	**DO (A/B) REG**	**READ Width (A/B)**	**WRITE Width (A/B)**	**WRITE Mode (A/B)**	**SAVE DATA**
1	March s2pf	1	36	36	READ_FIRST	FALSE
2	March d2pf	1	36	36	READ_FIRST	FALSE
3	MATS+	0	18	18	READ_FIRST	FALSE
4		0	9	9	WRITE_FIRST	FALSE
5		0	4	4	NO_CHANGE	FALSE
6		0	2	2	WRITE_FIRST	FALSE
7		0	1	1	READ_FIRST	FALSE

(b) Settings Part 2

BRAM Config	**Test Algorithm**	**CLK, EN, SSR REGCLK (A/B)(U/L) INV**	**RAM EXT**	**INIT VAL**	**SRVAL**	**INIT (A/B) VAL**
1	March s2pf	INV	NONE	AAAA	5555	0
2	March d2pf	not INV	NONE	5555	AAAA	FFFF
3	MATS+	not INV	NONE	AAAA	5555	0
4		not INV	NONE	5555	AAAA	FFFF
5		not INV	NONE	AAAA	5555	0
6		not INV	NONE	5555	AAAA	FFFF
7		not INV	NONE	AAAA	5555	0

3.2.2 ECC

The ECC TPG was created directly from the design proposed by Garrison. This configuration mode uses a fixed 72-bit data word length for each configuration with a fixed address space of 512. The final configuration settings used in this TPG are shown in Table 3-2.

The remaining specifications are implemented directly from those proposed by Garrison which are shown in Chapter 2. The ECC TPG also requires that a control string be shifted in via the BSCAN interface prior to beginning to test. The final control strings for this configuration are shown in Table 3-3.

Table 3-2 - Final ECC Configuration Settings
(a) ECC Settings Part 1

ECC Config	Test Algorithm	DO REG	EN_ECC READ	EC_ECC WRITE	EN_ECC SCRUB	INIT VAL	SR VAL	INIT (A/B) VAL	SAVE DATA
1	MarchLR w/BDS	0	FALSE	FALSE	FALSE	AAAA	5555	0	FALSE
2	ECC (read)	1	TRUE	FALSE	FALSE	AAAA	5555	0	FALSE
3	ECC (write)	1	FALSE	TRUE	FALSE	5555	AAAA	FFFF	FALSE

(b) ECC Settings Part 2

ECC Config	Test Algorithm	RDCLK (U/L) INV	RDEN (U/L) INV	RDRCLK (U/L) INV	WRCLK (U/L) INV	WREN (U/L) INV	SSR (U/L) INV
1	MarchLR w/BDS	not INV	not INV	not INV	not INV	not INV	not INV
2	ECC (read)	not INV	not INV	not INV	not INV	not INV	not INV
3	ECC (write)	INV	INV	INV	INV	INV	INV

Table 3-3 - Final Control String Values for ECC TPG

ECC Config	Test Algorithm	Level Control	Mode 2	Mode 1	Mode 0	Hex Control String
1	MarchLR w/BDS	0	0	0	0	0x0
2	ECC (read)	0	0	0	1	0x1
3	ECC (write)	0	0	1	0	0x2

3.2.3 FIFO

The FIFO TPG is an FSM developed from Garrison's initially proposed FIFO TPG. However an additional fifth test phase has been added. This additional phase is required in order to test the most significant bit of the configurable *almost empty* and *almost full* flags. When the BRAM is configured as a FIFO with data width 4 it is the only time the most significant bit of

the flag configuration is used. To detect faults for the MSB being stuck-at-0 and stuck-at-1 two separate test phases for this data width are necessary. The final test phases and the corresponding control string values of this TPG are shown in Table 3-4. The final test configuration settings along with the modified *almost empty* and *almost full* configuration values are shown in Table 3-5.

Table 3-4 – Final FIFO Test Phases and Control String Values

FIFO Config	Test Algorithm	Address Space	Data Width	Level Control	Mode 2	Mode 1	Mode 0	Hex Control String
1	FIFOX	1K	36	1	0	0	0	0x8
2		2K	18	0	0	0	1	0x1
3		4K	9	0	0	1	0	0x2
4		8K	4	0	0	1	1	0x3
5		8K	4	0	0	1	1	0x3

Table 3-5 – Final Configuration Settings for FIFO TPG
(a) FIFO Settings Part 1

ECC Config	Test Algorithm	DO REG	DATA WIDTH	EN SYN	FWFT	RDCLK (U/L) INV	RDEN INV	RST INV
1	FIFOX	1	36	TRUE	TRUE	INV	INV	INV
2		1	18	FALSE	FALSE	not INV	not INV	not INV
3		0	9	TRUE	TRUE	not INV	not INV	not INV
4		0	4	FALSE	FALSE	not INV	not INV	not INV
5		0	4	FALSE	FALSE	not INV	not INV	not INV

(b) FIFO Settings Part 2

ECC Config	Test Algorithm	WRCLK (U/L) INV	WREN INV	ALMOST EMPTY OFFSET	ALMOST FULL OFFSET
1	FIFOX	INV	INV	2AA	155
2		not INV	not INV	555	2AA
3		not INV	not INV	AAA	555
4		not INV	not INV	1555	AAA
5		not INV	not INV	AAA	1555

3.2.4 FIFOECC

The FIFOECC TPG has been improved from Garrison's initially proposed design. The testing algorithm used by the FIFOECC TPG is a modified version of the FIFOX algorithm designated FIFOD. This algorithm forces toggling of all of the ECC bits as it is executed by writing changing values to the FIFO. The value that is written into each address of the FIFO is a write or read count value which is repeated as many times as necessary to fill the data width being tested. This count value is incremented upon each write or read operation performed and reset at the beginning of each step. This algorithm is executed as follows:

Step 1. *Reset the FIFO.*

Step 2. *Repeat N times: write FIFO with count value repeated to match data width, check that Almost Empty flag goes inactive and Almost Full flag goes active at the appropriate points in the sequence.*

Step 3. *Repeat N times: read FIFO expecting repeated count value and write FIFO with the inversion of repeated count*

Step 4. *Repeat N times: read FIFO expecting inverted repeated count value, check that Almost Full flag goes inactive and Almost Empty flag goes active at the appropriate points in the read sequence.*

The final test phases for this test mode and the final configuration settings may be seen in Table 3-6 and Table 3-7. Control string values are not necessary for this TPG because the same algorithm is executed for both test phases and only the BRAM configuration is modified.

Table 3-6 - Final FIFOECC Test Phases

FIFOECC Config	Test Algorithm	Address Space	Data Width	Level Control
1	FIFOD (read)	512	72	0
2	FIFOD (write)	512	72	0

Table 3-7 – Final Configuration Settings for FIFOECC TPG
(a) FIFOECC Settings Part 1

ECC Config	Test Algorithm	DO REG	EN_ECC READ	EN_ECC WRITE	EN SYN	FWFT	RST INV
1	FIFOD (read)	1	TRUE	FALSE	FALSE	TRUE	not INV
2	FIFOD (write)	0	FALSE	TRUE	TRUE	FALSE	not INV

(b) FIFOECC Settings Part 2

ECC Config	Test Algorithm	ALMOST EMPTY OFFSET	ALMOST FULL OFFSET	RDCLK (U/L) INV	RDRCLK (U/L) INV	RDEN INV	WRCLK (U/L) INV	WREN INV
1	FIFOD (read)	155	AA	not INV	not INV	not INV	not INV	not INV
2	FIFOD (write)	AA	155	not INV	not INV	not INV	not INV	not INV

3.2.5 CASC

The CASC TPG executes a March Y based algorithm designed strictly to test the functionality of the cascade circuitry. The March Y algorithm simply performs the algorithm operations on one address in the *UPPER* BRAM and one address in the *LOWER* BRAM. By doing this all the cascade circuitry can be verified quickly.

The final CASC test phases can be seen in Table 3-8. No control string values are necessary for this TPG as the same test is run for both phases. The final configuration settings for this TPG can be seen in Table 3-9.

Table 3-8 – Final CASC Test Phases

CASC Config	Test Algorithm	Address Space	Data Width	Level Control
1	March Y	1K	64	0
2		1K	64	0

Table 3-9 – Final Configuration Settings for CASC TPG

CASC Config	Test Algorithm	DOA/B REG	RD WIDTH A/B	WR WIDTH A/B	RAM EXT A	RAM EXT B
1	March Y	1	1	1	UPPER	LOWER
2		1	1	1	LOWER	UPPER

3.2.6 Test Configurations Summary

Each of these TPGs is FSM based and is restricted to the smallest area possible on the FPGA devices. The resource usage for all the TPGs after synthesis can be seen in Table 3-10. It is important to note that each TPG is placed twice in each BIST configuration and that the resource usage per is independent of the device being tested. The 19 phases of the BIST sequence for the Virtex-5 devices are displayed in Table 3-11. The various configuration address spaces and data widths are shown along with the hexadecimal representation of the 4-bit control string required to run each test phase.

Table 3-10 - BIST TPG Resource Usage

TPG	Slices	Slice Registers	Slice LUTs	CLB Area (column x row)
BRAM	148	242	587	8 x 20
ECC	205	566	808	8 x 30
FIFO	34	58	135	8 x 5
FIFOECC	43	162	122	8 x 10
CASC	4	10	9	8 x 1

Table 3-11 – Complete Virtex-5 BIST Procedure

BIST Config	BRAM Mode	Test Algorithm	Address Space	Data Width	Control String
1	BRAM	March s2pf	1K	36	0x0
2		March d2pf	1K	36	0x1
3		MATS+	2K	18	0xA
4			4K	9	0xB
5			8K	4	0xC
6			16K	2	0xD
7			32K	1	0xE
8	ECC	MarchLR w/BDS	512	72	0x0
9		ECC (read)	512	72	0x1
10		ECC (write)	512	72	0x2
11	FIFO	FIFOX	1K	36	0x8
12			2K	18	0x1
13			4K	9	0x2
14			8K	4	0x3
15			8K	4	0x3
16	FIFOECC	FIFOD (read)	512	72	0x0
17		FIFOD (write)	512	72	0x0
18	CASC	March Y	1K	64	0x0
19			1K	64	0x0

3.3 ORA Design

The ORAs are designed to use a double comparison of BRAM outputs and a circular comparison routing architecture. The iterative OR-chain described in Section 2.4.1 is also implemented to accommodate results retrieval and an instantaneous *Pass/Fail* indicator. The ORAs in the BIST are placed in two columns of five CLBs immediately adjacent to the BRAMs. Each of these groups of 10 CLBs is responsible for comparing all the outputs of two distinct BRAMs as shown in Figure 2-8 where each ORA block represents one group of CLBs. Each

ORA slice is equipped with four 6-input LUTs that are used to compare the outputs of the BRAMs. The inputs to these LUTS are used for comparison of up to two pairs of BRAM outputs. This architecture provides a total of up to 160 possible comparisons per BRAM.

The number of observed outputs for the BRAM, ECC, FIFO, and CASC configurations is less than 80 (half of the total comparisons) as shown in Table 3-12. This means that each ORA performs a comparison of a single pair of BRAM outputs. The number of observed outputs for the FIFOECC is greater than 80 such that some ORAs perform a comparison of two pairs of outputs. A failure in an ORA making a double comparison is only traceable to be one of the two outputs that are routed to it.

Table 3-12 - Compared Outputs for Configuration Modes

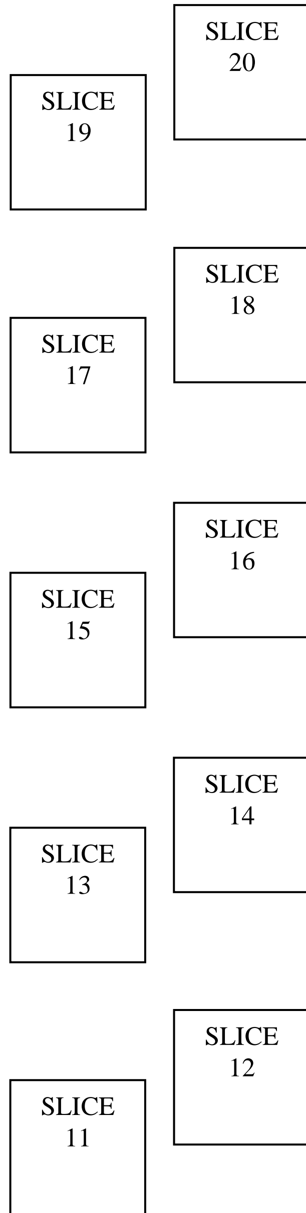
Configuration Mode	**Compared Outputs**
BRAM	72
ECC	74
FIFO	68
FIFOECC	106
CASC	4

3.3.1 ORA Comparison Routing

The outputs of each pair of BRAMs to be compared are routed to a group of two columns of five CLBs immediately to the left of one of the BRAMs. Each of these groups contains 20 slices organized as shown in Figure 3-2. Each one of these slices contains ORAs designated A through D. Table 3-13 summarizes the routing of the BRAM outputs to the ORAs within these groups. Each configuration mode of the BIST is shown in this table, and this routing is

consistent for each group of ORAs in a BIST configuration as they span the entire height of the device. This routing information may be used in order to diagnose a fault location by using configuration memory read back to locate the flip-flop which has latched a fault. Once the failing flip-flop(s) is located it can be matched to a specific ORA whose inputs are known.

ORA
Column 2



ORA
Column 1

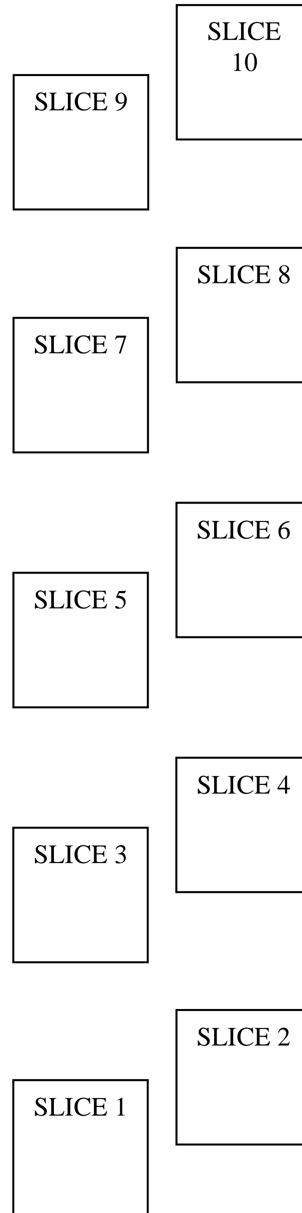


Figure 3-2 - ORA Map

Table 3-13 - ORA Input Routing Tables
(a) BRAM ORA Routing

Output	**Slice**	**ORA**	**Output**	**Slice**	**ORA**	**Output**	**Slice**	**ORA**
DOA0	1	A	DOA24	7	A	DOB12	14	A
DOA1		B	DOA25		B	DOB13		B
DOA2		C	DOA26		C	DOB14		C
DOA3		D	DOA27		D	DOB15		D
DOA4	2	A	DOA28	8	A	DOB16	15	A
DOA5		B	DOA29		B	DOB17		B
DOA6		C	DOA30		C	DOB18		C
DOA7		D	DOA31		D	DOB19		D
DOA8	3	A	DOPA0	9	A	DOB20	16	A
DOA9		B	DOPA1		B	DOB21		B
DOA10		C	DOPA2		C	DOB22		C
DOA11		D	DOPA3		D	DOB23		D
DOA12	4	A	DOB0	11	A	DOB24	17	A
DOA13		B	DOB1		B	DOB25		B
DOA14		C	DOB2		C	DOB26		C
DOA15		D	DOB3		D	DOB27		D
DOA16	5	A	DOB4	12	A	DOB28	18	A
DOA17		B	DOB5		B	DOB29		B
DOA18		C	DOB6		C	DOB30		C
DOA19		D	DOB7		D	DOB31		D
DOA20	6	A	DOB8	13	A	DOPB0	19	A
DOA21		B	DOB9		B	DOPB1		B
DOA22		C	DOB10		C	DOPB2		C
DOA23		D	DOB11		D	DOPB3		D

(b) ECC ORA Routing

Output	Slice	ORA	Output	Slice	ORA	Output	Slice	ORA
DO0	1	A	DO24	7	A	DO48	14	A
DO1		B	DO25		B	DO49		B
DO2		C	DO26		C	DO50		C
DO3		D	DO27		D	DO51		D
DO4	2	A	DO28	8	A	DO52	15	A
DO5		B	DO29		B	DO53		B
DO6		C	DO30		C	DO54		C
DO7		D	DO31		D	DO55		D
DO8	3	A	DO32	9	A	DO56	16	A
DO9		B	DO33		B	DO57		B
DO10		C	DO34		C	DO58		C
DO11		D	DO35		D	DO59		D
DO12	4	A	SBITERR	10	A	DO60	17	A
DO13		B	DO36	11	A	DO61		B
DO14		C	DO37		B	DO62		C
DO15		D	DO38		C	DO63		D
DO16	5	A	DO39		D	DOP0	18	A
DO17		B	DO40	12	A	DOP1		B
DO18		C	DO41		B	DOP2		C
DO19		D	DO42		C	DOP3		D
DO20	6	A	DO43		D	DOP4	19	A
DO21		B	DO44	13	A	DOP5		B
DO22		C	DO45		B	DOP6		C
DO23		D	DO46		C	DOP7		D
			DO47		D	DBITERR	20	A

(c) FIFO ORA Routing

Output	Slice	ORA	Output	Slice	ORA	Output	Slice	ORA
DO0	1	A	DO24	7	A	WRCOUNT10	14	A
DO1		B	DO25		B	WRCOUNT11		B
DO2		C	DO26		C	WRCOUNT12		C
DO3		D	DO27		D	RDCOUNT0		D
DO4	2	A	DO28	8	A	RDCOUNT1	15	A
DO5		B	DO29		B	RDCOUNT2		B
DO6		C	DO30		C	RDCOUNT3		C
DO7		D	DO31		D	RDCOUNT4		D
DO8	3	A	DOP0	9	A	RDCOUNT5	16	A
DO9		B	DOP1		B	RDCOUNT6		B
DO10		C	DOP2	11	A	RDCOUNT7		C
DO11		D	DOP3		B	RDCOUNT8		D
DO12	4	A	WRCOUNT0		C	RDCOUNT9	17	A
DO13		B	WRCOUNT1		D	RDCOUNT10		B
DO14		C	WRCOUNT2	12	A	RDCOUNT11		C
DO15		D	WRCOUNT3		B	RDCOUNT12		D
DO16	5	A	WRCOUNT4		C	FULL	18	A
DO17		B	WRCOUNT5		D	EMPTY		B
DO18		C	WRCOUNT6	13	A	ALMOSTFULL		C
DO19		D	WRCOUNT7		B	ALMOSTEMPTY		D
DO20	6	A	WRCOUNT8		C	RDERR	19	A
DO21		B	WRCOUNT9		D	WRERR		B
DO22		C						
DO23		D						

(d) FIFOECC ORA Routing

Output	Slice	ORA	Output	Slice	ORA	Output	Slice	ORA
DO0	1	A	DO32	5	A	DOP6	12	A
DO1		A	DO33		A	DOP7		B
DO2		B	DO34		B	SBITERR		C
DO3		B	DO35		B	DBITERR		D
DO4		C	DO36		C	FULL	13	A
DO5		C	DO37		C	EMPTY		B
DO6		D	DO38		D	ALMOSTFULL		C
DO7		D	DO39		D	ALMOSTEMPTY		D
DO8	2	A	DO40	6	A	RDERR	14	A
DO9		A	DO41		A	WRERR		B
DO10		B	DO42		B	RDCOUNT0		C
DO11		B	DO43		B	RDCOUNT1		D
DO12		C	DO44		C	RDCOUNT2	15	A
DO13		C	DO45		C	RDCOUNT3		B
DO14		D	DO46		D	RDCOUNT4		C
DO15		D	DO47		D	RDCOUNT5		D
DO16	3	A	DO48	7	A	RDCOUNT6	16	A
DO17		A	DO49		A	RDCOUNT7		B
DO18		B	DO50		B	RDCOUNT8		C
DO19		B	DO51		B	RDCOUNT9		D
DO20		C	DO52		C	RDCOUNT10	17	A
DO21		C	DO53		D	RDCOUNT11		B
DO22		D	DO54	8	A	RDCOUNT12		C
DO23		D	DO55		B	WRCOUNT0		D
DO24	4	A	DO56		C	WRCOUNT1	18	A
DO25		A	DO57		D	WRCOUNT2		B
DO26		B	DO58	9	A	WRCOUNT3		C
DO27		B	DO59		B	WRCOUNT4		D
DO28		C	DO60		C	WRCOUNT5	19	A
DO29		C	DO61		D	WRCOUNT6		B
DO30		D	DO62	10	A	WRCOUNT7		C
DO31		D	DO63		B	WRCOUNT8		D
			DOP0		C	WRCOUNT9	20	A
			DOP1		D	WRCOUNT10		B
			DOP2	11	A	WRCOUNT11		C
			DOP3		B	WRCOUNT12		D
			DOP4		C			
			DOP5		D			

(e) CASC ORA Routing

Output	Slice	ORA
DOA0	1	A
DOPA0		B
DOB0	11	A
DOPB0		B

3.4 BIST Implementation

The fully routed BIST configuration on a physical device is shown in Figure 3-3. This design was created for the LX30, one of the smaller devices in the Virtex-5 family, and is for the BRAM configuration mode. In this device there are two columns of BRAMs running vertically on the device. The ORAs are placed directly to the left of the BRAMs in the immediately adjacent CLB columns, and the BRAM outputs are routed directly to the appropriate ORAs. The two TPGs are visible on the right side of the device. The bottom TPG is placed on the lowest row of CLBs available and in the six columns of CLBs to the right of the rightmost BRAM column. The second TPG is placed in these same six columns above the first TPG, beginning exactly half way up the device. The TPG outputs are then each routed to alternating BRAMs in the columns. The routing from the boundary scan interface is located directly in the center of the device.

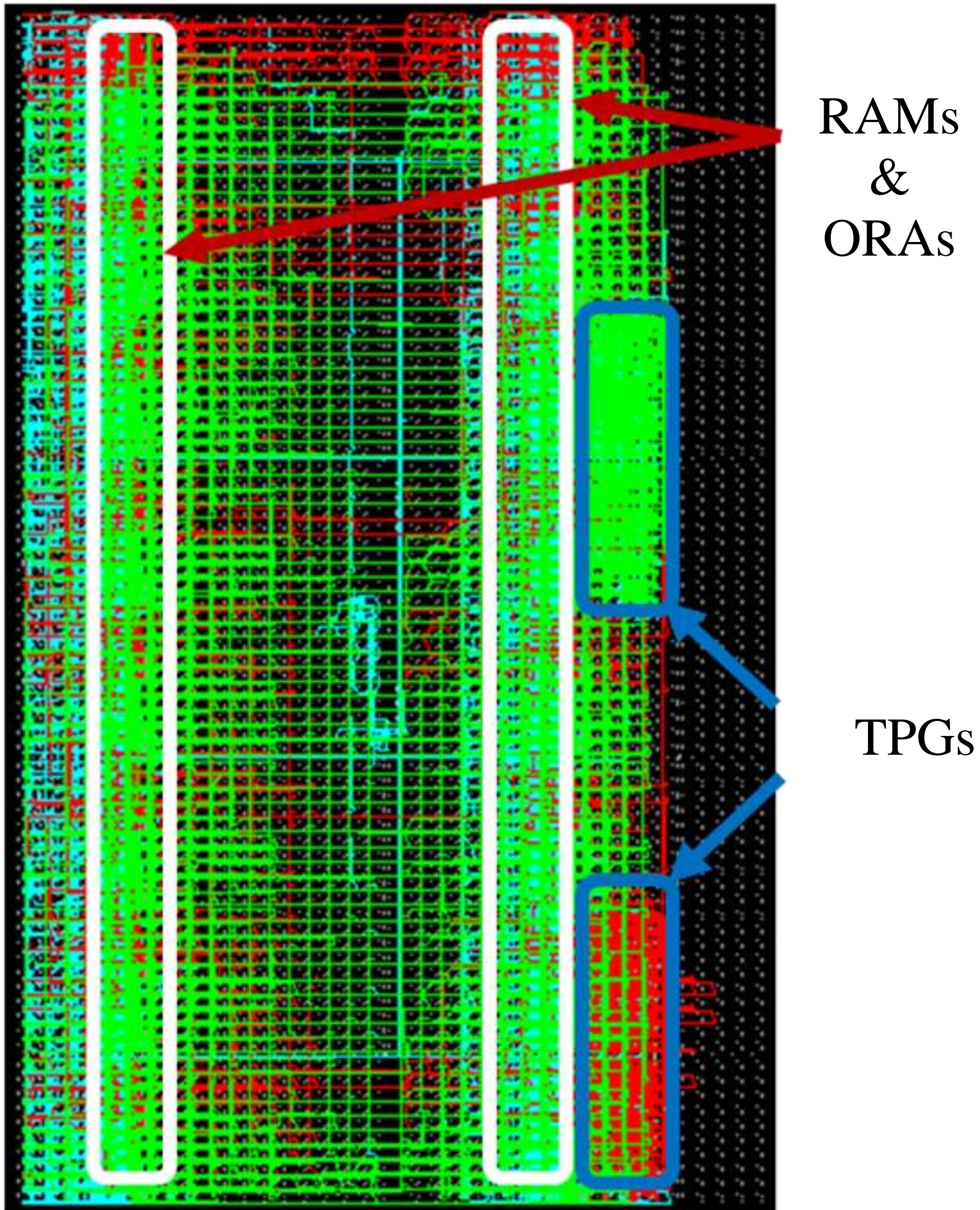


Figure 3-3 – BRAM BIST Configuration Routed on Virtex-5 LX30

3.4.1 Cascade Routing

The cascade configuration mode presents a unique situation for ORA routing. When the BRAMs are functioning in this mode of operation two of the memories are cascaded together in order to form one larger memory, and the output of this memory is only displayed on the output port of the BRAM configured in the *UPPER* mode. The output of the BRAM configured in the *LOWER* mode is routed to the output of the *UPPER* memory as shown in Figure 2-7. This means that the outputs of every other BRAM in a column will be identical, rather than all BRAM outputs being identical. Therefore, the outputs of each BRAM are routed to properly reflect this change, and every other BRAM will be compared.

Using this approach presents another problem during the second cascade testing phase. In the first testing phase BRAMs are configured as *LOWER* and *UPPER* alternating starting at the bottom of the column. In the second testing phase these configurations will be reversed such that the bottom BRAM will be configured as an *UPPER*, and the configurations will alternate from there up the column. When this occurs the BRAMs located without another BRAM directly beneath them are configured as *UPPER* and are used to output data, but will not output any data that is expected from the *LOWER* memory because there is no *CASCADEIN* routing available for these components. This will produce failures, even with fault free circuitry, if the cascade routing approach described above is used.

A solution used by Milton and Garrison for Virtex-4 devices in a similar cascade mode of operation is described in [2] and [3]. The solution used by them accounts for these expected failures by using clock enable controls in the ORAs to avoid clocking the result from an expected

failure into the flip-flips of the ORAs. This approach required tedious modifications to the TPG in order to enable the ORA flip-flops during some clock cycles and disable them during others when the failures were expected. It also requires that the ORA design be modified to include these clock enables. A simpler solution is implemented for the Virtex-5 devices which eliminates any expected failures from the design.

The solution requires modification to the initial routing from the BRAMs to the ORAs. Instead of routing all BRAM outputs to ORAs, the routing from any BRAM that does not have an available *CASCADEIN* input is omitted completely as shown in Figure 3-4. Additionally, the output of the BRAM that would normally be compared to these outputs to complete the circular comparison is routed to the next ORA in the column to maintain the circular comparison. This situation occurs for any BRAMs located at the bottom of columns, directly above a PowerPC module, or some BRAMs in the special TEMAC columns which are present in some Virtex-5 devices [13]. This omission of routing will not result in any reduction in fault coverage because there is no need to observe the outputs of these BRAMs in the cascade mode. When they are configured in the *LOWER* mode the output is routed to the *UPPER* BRAM and displayed on its outputs. These specific BRAMs do not need to be observed when configured in the *UPPER* mode because should never be used with this configuration in practice because there is no available *CASCADEIN* routing.

The ORAs located at the bottom of these columns that do not have BRAM outputs routed to them still retain their OR-chain routing. In this case, these ORAs are made into dummy ORAs that simply propagate the carry chain result. This solution eliminates the need for special modifications to the TPG or ORAs for the cascade BIST configuration by eliminating the

expected failures all together. All fault detection and diagnosis ability is retained and the circular comparison ORA architecture is maintained.

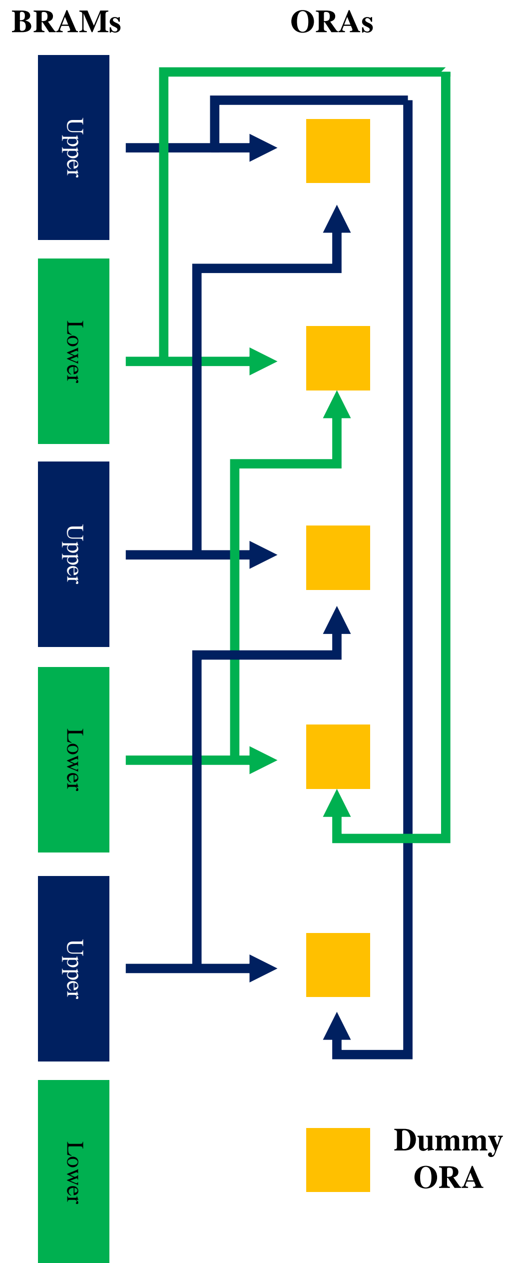


Figure 3-4 – Virtex-5 Cascade ORA Routing

3.5 Programming Tools

A series of programming tools are used to aid in the development, implementation, and simulation of the BRAM BIST configurations. The way in which each of these tools is used in the BIST development will be described in subsequent sections. A brief statement on the capabilities of these tools is given below:

- ISE – a Xilinx design suite for creating, synthesizing, and implementing VHDL models for use in Xilinx FPGAs. Allows area constraints to be created to specify placement of a design [14].
- FPGA Editor – a Xilinx tool that provides of graphical user interface (GUI) for visual examination and editing of designs on the FPGA [14].
- Place and Route (PAR) – a Xilinx tool that performs placement and routing of FPGA designs [14].
- XDL – a Xilinx tool which converts between Xilinx file formats: NCD (FPGA Editor files) and XDL (Xilinx netlist description files) [14].
- BitGen – a Xilinx tool which generates BIT or RBT files from NCD files. These BIT and RBT files contain the configuration information which is downloaded into the FPGA [14].
- TRCE – A Xilinx tool for timing analysis of a design. Specifically, it determines the maximum clock frequency at which a design may be run [14].
- ModelSim Xilinx Edition – A simulator made by Mentor Graphics which is able to simulate VHDL models using Xilinx primitives [15].

3.6 Configuration File Generation

The entire generation procedure using the two BIST generation programs and the tools mentioned in Section 3.5 is diagramed in Figure 3-5 [16]. The initial generation of all the Virtex-5 BRAM BIST configurations is done using two separate programs which are both written in the *C* programming language. These two programs are responsible for the creation of the *XDL* files containing exact placement and routing information for the entire BIST configuration [16]. The synthesized VHDL models of the TPGs are converted into *XDL* format and inserted into the generated *XDL* file. The *XDL* file is then converted into an *NCD* file which is able to be graphically displayed within *FPGA Editor*. *FPGA Editor* is used in order to automatically route the unrouted nets which have been designated in the design [16]. After the design has been completely routed it will be converted into a configuration *BIT* file capable of being downloaded directly into the FPGA device [16].

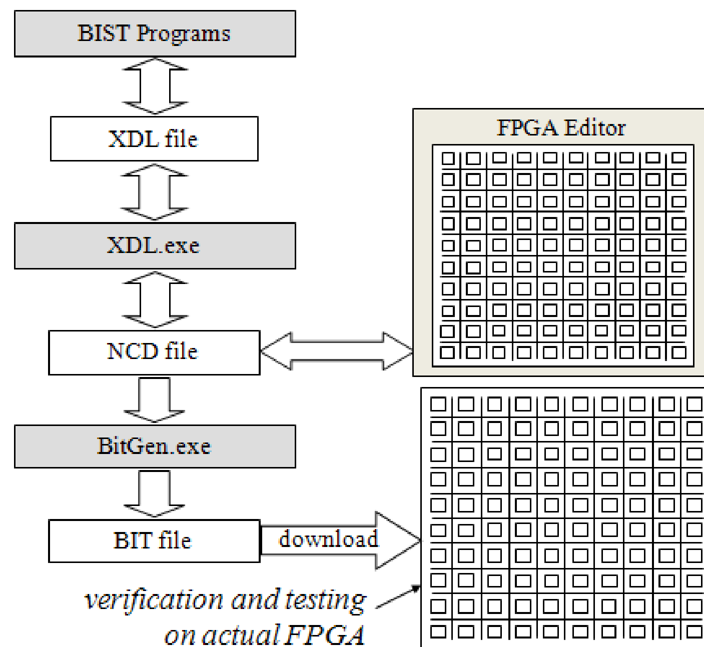


Figure 3-5 - BIST Configuration Process [16]

3.6.1 BIST Generation Program

The program responsible for the generation of the *XDL* file containing the BIST design is called *V5RAMBIST.exe*. This program is run by the user and several parameters are provided in order to specify the target device and type for the BIST as well as several other details. The exact command line formatting may be seen in Figure 3-6.

```
V5RAMbist (v1.6) - generates template file for block RAM BIST config in any Virtex 5
command line format:
V5RAMbist <xdlfile> <startrow> <startcol> <endrow> <endcol> <dev> <part> <type> [n,a,p]
 where type = bram (RAMB36 mode BIST)
             fifo (FIFO36 mode BIST)
             ecc  (RAMB36SDP mode BIST)
             fifecc  (FIFO36_72 mode BIST)
             casc (Cascade RAM mode BIST)
   dev  part rows cols   dev part rows cols   dev part rows cols
   lxt   20   60   33
   lx/t  30   80   38    sxt   35   80   50    fxt   30   80   50
   lx/t  50  120   38    sxt   50  120   50    fxt   70  160   50
   lx/t  85  120   64    sxt   95  160   68    fxt  100  160   73
   lx/t 110  160   64    sxt  240  240  104    fxt  130  200   70
   lx/t 155  160   87                          fxt  200  240   87
   lx/t 220  160  121    txt  150  200   70
   lx/t 330  240  121    txt  240  240   91
 n: this option runs xdl2ncd with -nodrc option
 a: runs 'n' option followed by FPGA Editor routing with no pinswap and converts back to XDL
 p: this option uses system-level pins instead of Boudary Scan interface
    PLUS runs xdl2ncd with -nodrc option
 note: all parameters can be upper or lower case (but not mixed)
```

Figure 3-6 - V5RAMBIST Command Line Instructions

3.6.2 Modification Program

The second *C* program called *V5RAMMOD*.exe is responsible for the modification of the configuration settings in the *XDL* files. In order to run this program the user specifies the generic

input *XDL* file which has been generated using the generation program and the name of the desired output file. The BRAM configuration mode must also be specified along with the phase of the test and several other parameters seen in Figure 3-7.

```
V5RAMmod (ver 1.2) - modifies routed XDLs for Block RAM to subseuqent BIST configs
command line format:
V5RAMmod <xdl_in> <xdl_out> <phase> <type> [ncd,bit]
where the type is defined as:
  Type:    bram(RAMB36) ecc(RAMB36SDP) fifo(FIFO36) fifecc(FIFO36_72) casc(RAMB36)
--------------------------------------------------------------------------------
  Phase 1: S2PF          MarchLR    FIFOx 1K    FIFOx_ECC_RD     CASC_RD
  Phase 2: D2PF          ECC_RD     FIFOx 2K    FIFOx_ECC_WR     CASC_WR
  Phase 3: MATS+ 2K      ECC_WR     FIFOx 4K
  Phase 4: MATS+ 4K                 FIFOx 9K
  Phase 5: MATS+ 8K                 FIFOx 9K-SWAP
  Phase 6: MATS+ 16K
  Phase 7: MATS+ 32K
--------------------------------------------------------------------------------
Generation Options:
- ncd option runs XDL -XDL2NCD
- bit option runs XDL -XDL2NCD and BITGEN -D -B -G COMPRESS
- if no option is selected, only the XDL file will be generated
```

Figure 3-7 - V5RAMMOD Command Line Instructions

3.7 Results and Analysis

In this section the results of the BRAM BIST will be presented. This will include the fault detection capabilities of the BIST, size optimizations for the configurations, and analysis of the timing capabilities of the configurations. The complete BIST procedure consists of 19 separate configurations. All configurations were generated for all Virtex-5 devices using the BIST programs, and the configurations for LX30T, LX50T, SX35T, SX50T, FX30T, and FX70T

FGPAs were downloaded to and verified on actual devices. All 19 test phases are displayed in Table 3-14. The number of clock cycles required to run each phase of the test is also shown in the table in terms of the total number of clock cycles for the BIST. These running times are negligible when compared to the time taken to download the configurations to the devices, which becomes the dominant factor in total test time. This places a high emphasis on reducing configuration file size to improve test time.

Table 3-14 - Complete BRAM BIST

BIST Config	BRAM Mode	Test Algorithm	Address Space	Data Width	Clock Cycles
1 (C)	BRAM	March s2pf	1K	36	20,000
2 (P)		March d2pf	1K	36	15,000
3 (P)		MATS+	2K	18	25,000
4 (P)			4K	9	45,000
5 (P)			8K	4	85,000
6 (P)			16K	2	165,000
7 (P)			32K	1	330,000
8 (C)	ECC	MarchLR w/BDS	512	72	23,000
9 (P)		ECC (read)	512	72	7,000
10 (P)		ECC (write)	512	72	7,000
11 (C)	FIFO	FIFOX	1K	36	8,500
12 (P)			2K	18	34,000
13 (P)			4K	9	66,000
14 (P)			8K	4	131,500
15 (P)			8K	4	131,500
16 (C)	FIFOECC	FIFOD (read)	512	72	10,000
17 (P)		FIFOD (write)	512	72	10,000
18 (C)	CASC	March Y	1K	64	36
19 (P)			1K	64	36
Total BIST Clock Cycles = 1,113,572					
(C) = Compressed Configuration (P) = Partial Configuration					

3.7.1 Fault Detection

The most important factor when evaluating the effectiveness of a test procedure is the fault coverage. In order to evaluate the fault coverage of the BRAM BIST physical fault injection was applied to the bits in the configuration memory of the BRAMs. There are a total of 488 possible configuration memory faults associated with each of the BRAMs. This number results from each BRAM having 244 total configuration bits which may each either be stuck-at-0 or stuck-at-1. Each of these faults was emulated by overwriting the desired configuration bit

with the stuck-at value of the desired fault before performing the entire BIST sequence. This process is repeated for each of the 488 configuration memory bit faults. The *Pass/Fail* result of each test phase was recorded after the injection of each fault.

The individual and cumulative fault coverage for the seven BRAM BIST configurations is shown in Figure 3-8. This graph displays the individual number of fault detections from each of BRAM test phases. The line displayed above the bars is a representation of the cumulative fault coverage of the phases. Each of the phases detects between 100 and 200 of the configuration memory bit faults. The sequence results in a fault coverage of 84% from running only the BRAM BIST configurations.

The overall fault coverage of configuration memory bits obtained from running the entire BIST sequence is shown in Figure 3-9. This graph also shows both the faults detected by each phase of the test and the cumulative detections. The entire test was able to detect 481 of the configuration memory faults resulting in a fault coverage of 98.57%. The other seven undetected faults are non-functional faults, which gives the BIST a 100% fault coverage of detectable faults in the BRAM configuration memory.

Fault injection with the configuration memory bits was used to verify the fault detection capabilities of the BIST since it is not possible to emulate actual SRAM faults that may occur within the BRAMs of a Virtex-5 device such as those described in Section 2.2.1. The injected configuration memory faults produce faulty outputs on the BRAMs that mimic those that would be produced by a BRAM containing SRAM faults. Thus, the fault coverage of the configuration memory bit faults gives an accurate representation of the fault coverage of the BIST for SRAM faults [17].

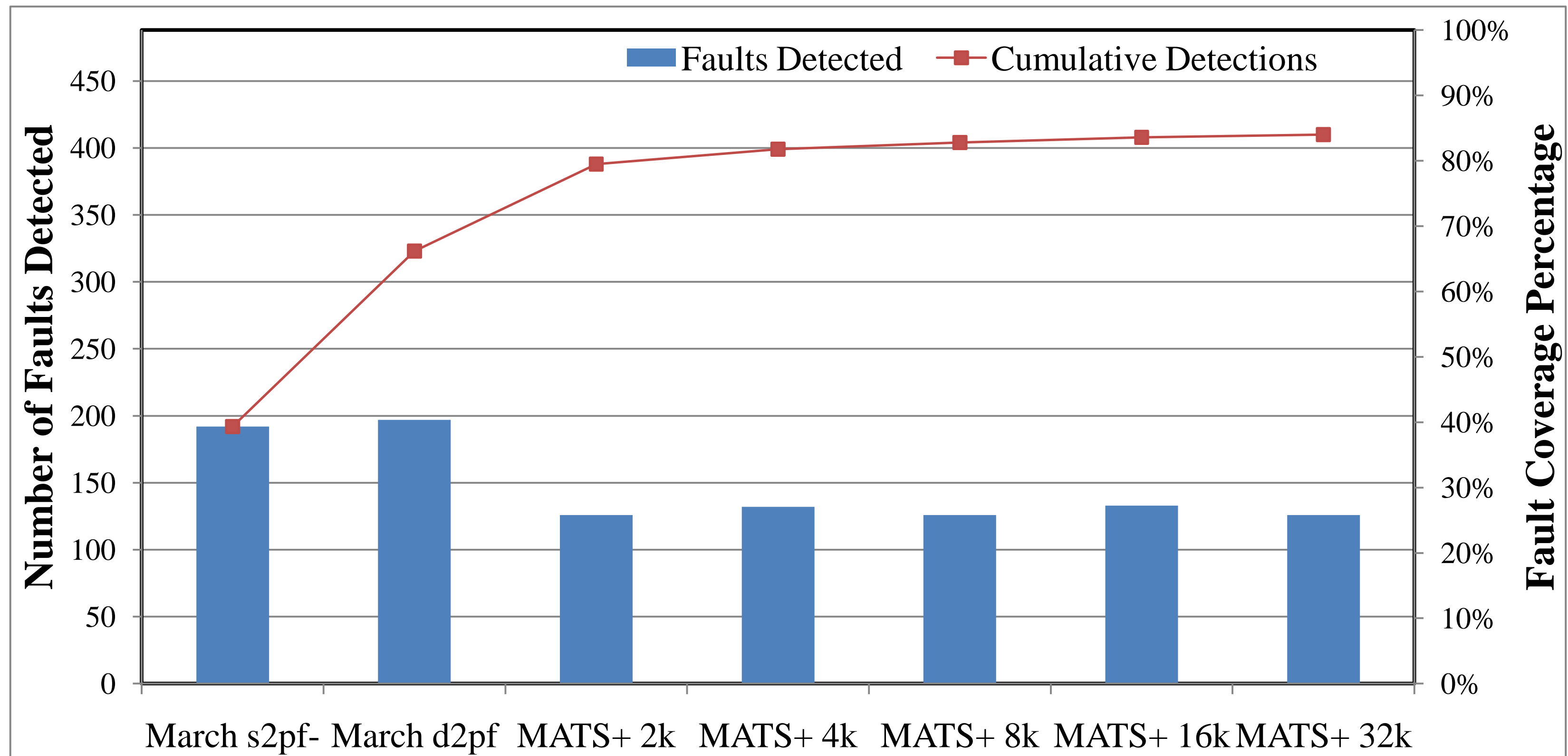


Figure 3-8 – BRAM Configuration Mode Fault Detections

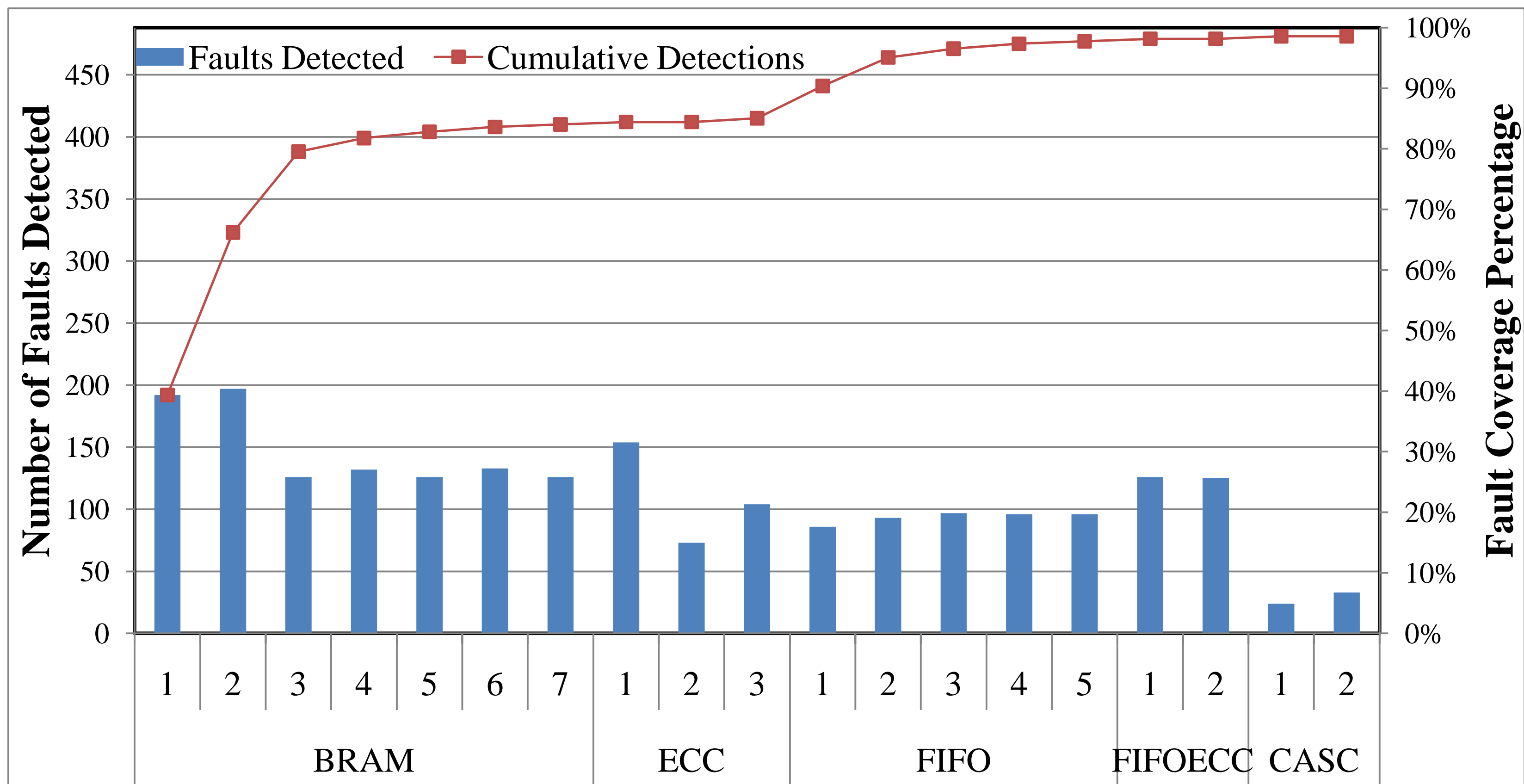


Figure 3-9 – Entire BIST Sequence Fault Detections

3.7.2 File Size Reduction

Once the fully routed *NCD* files for each BIST configuration have been generated, the Xilinx *BitGen.exe* tool mentioned in Section 3.5 is then used to create the configuration *BIT* file that will be downloaded directly to the FPGA. This tool is capable of generating three different types of configuration files: full, compressed, and partial [14]. The full configurations have no compression and contain values for every configuration memory bit within the device. Compressed configuration files take advantage of a feature in the Virtex-5 FPGAs called multiple frame writing. This feature allows identical frames of data to be stored as a single frame in the configuration file and written to multiple addresses in the configuration memory [14]. This allows for a significant reduction in configuration file size for designs containing many identical components, such as the BRAM BIST. The Virtex-5 device also supports partial reconfiguration, which can be utilized to provide the greatest reduction in configuration file size. These partial reconfiguration files are created by comparing two *NCD* file designs and the partial reconfiguration file will be created that details only the differences between the two designs [14]. Knowing this, the BIST configurations were designed in an extremely regular manner in order to minimize the differences between sequential configurations and configuration file size. Only compressed configurations and partial reconfigurations are used for the BIST in order to fully minimize download size. A compressed configuration is used for the first test phase of each of the five configuration modes, and partial reconfiguration is used for the remaining phases. The final file sizes of the BIST generated for the LX30 are shown in Table 3-15. The file size reduction achieved from the use of the compression methods mentioned is shown in Figure 3-10.

Table 3-15 - BIST Configuration File Sizes for LX30

BIST Config	File Size K-bytes	BIST Config	File Size K-bytes	BIST Config	File Size K-bytes
1 (C)	583	7 (P)	49	13 (P)	4
2 (P)	55	8 (C)	592	14 (P)	4
3 (P)	49	9 (P)	3	15 (P)	4
4 (P)	49	10 (P)	50	16 (C)	564
5 (P)	49	11 (C)	532	17 (P)	4
6 (P)	49	12 (P)	4	18 (C)	387
Total File Size = 3,034 K-bytes				19 (P)	3

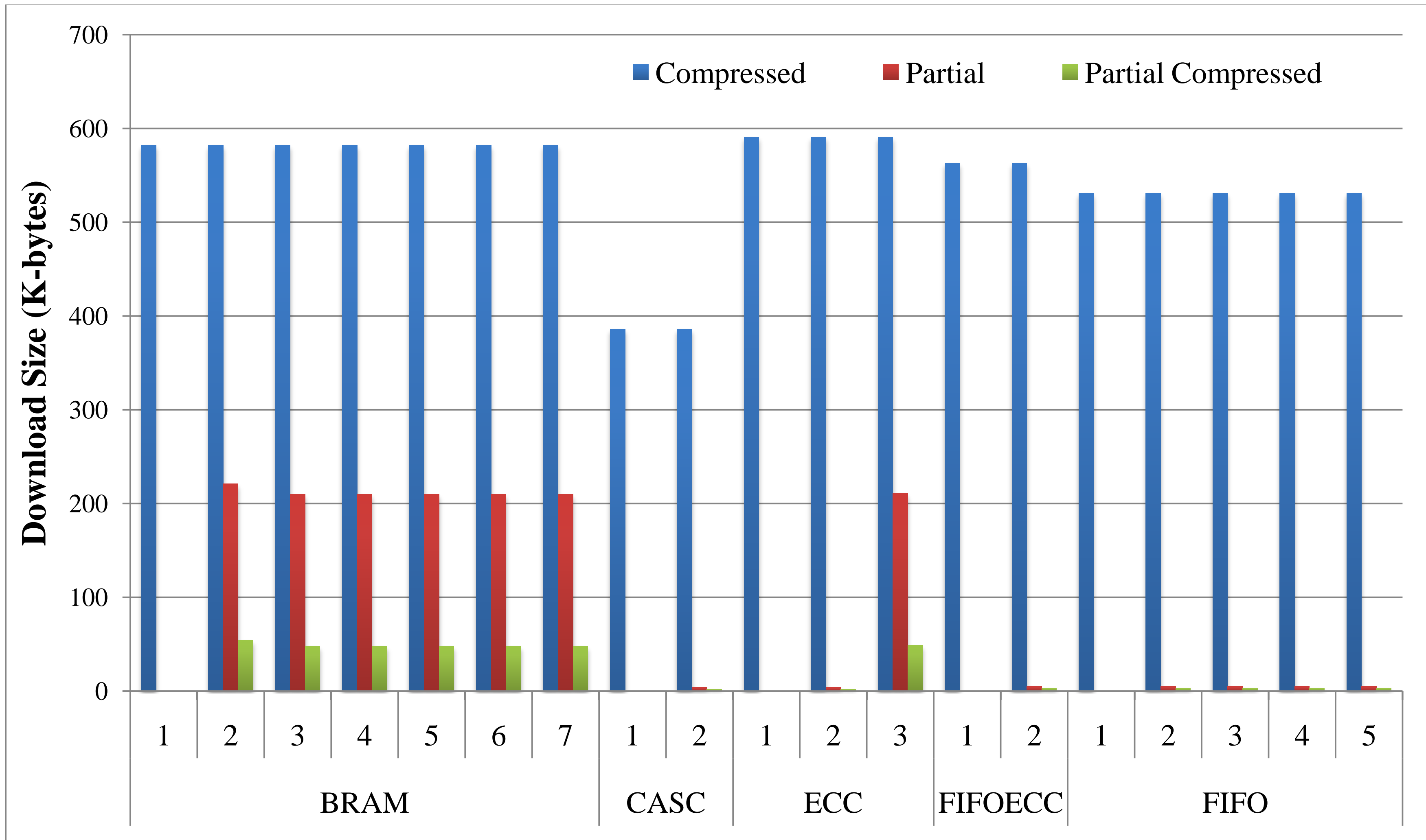


Figure 3-10 - BIST Configuration File Size Reduction for LX30

3.7.3 Timing Analysis

By using the Xilinx timing analysis tool *trce.exe* mentioned in Section 3.5 the maximum BIST clock frequency for each of the BIST configurations on all Virtex-5 devices has been determined. The results of this analysis on the LX30T device are shown in Figure 3-11. The FIFOECC and the CASC configurations are able to be run at the fastest clock frequency in this device. It can also be seen that for the different configurations the clock frequency remains in a consistent range except for the third ECC configuration and the first FIFO configuration. This results from the inversion of the BRAM clocks for testing in these two phases. When the clocks are inverted it presents a case where opposite edge clocking occurs which effectively halves the maximum BIST clock frequency.

This problem is overcome by inverting the TPG and ORA clocks in the CLBs during these two BIST configurations. These configurations with the inverted TPG and ORA clocks are positioned at either the beginning or end of a configuration mode sequence so that the inversion is only performed once. This is done to minimize download and test time.

The final maximum clock frequencies obtained from the analysis of select devices in the Virtex-5 family are shown in Figure 3-12. These frequencies reflect the speeds after the change which accounts for opposite edge clocking was applied. For each of the five configuration modes the lowest maximum BIST clock frequency is displayed in the figure. It can also be seen that the larger devices have a much slower maximum clock frequency due to longer routing requirements in these devices. By comparing the data in Figure 3-11 to the final data for the LX30T device in Figure 3-12 it can be seen that after inverting the clocks to account for opposite edge clocking the maximum speed of the ECC and FIFO configurations for the LX30T device

increases from just over 40 MHz to over 80 MHz, putting these configuration modes in a range similar to the others.

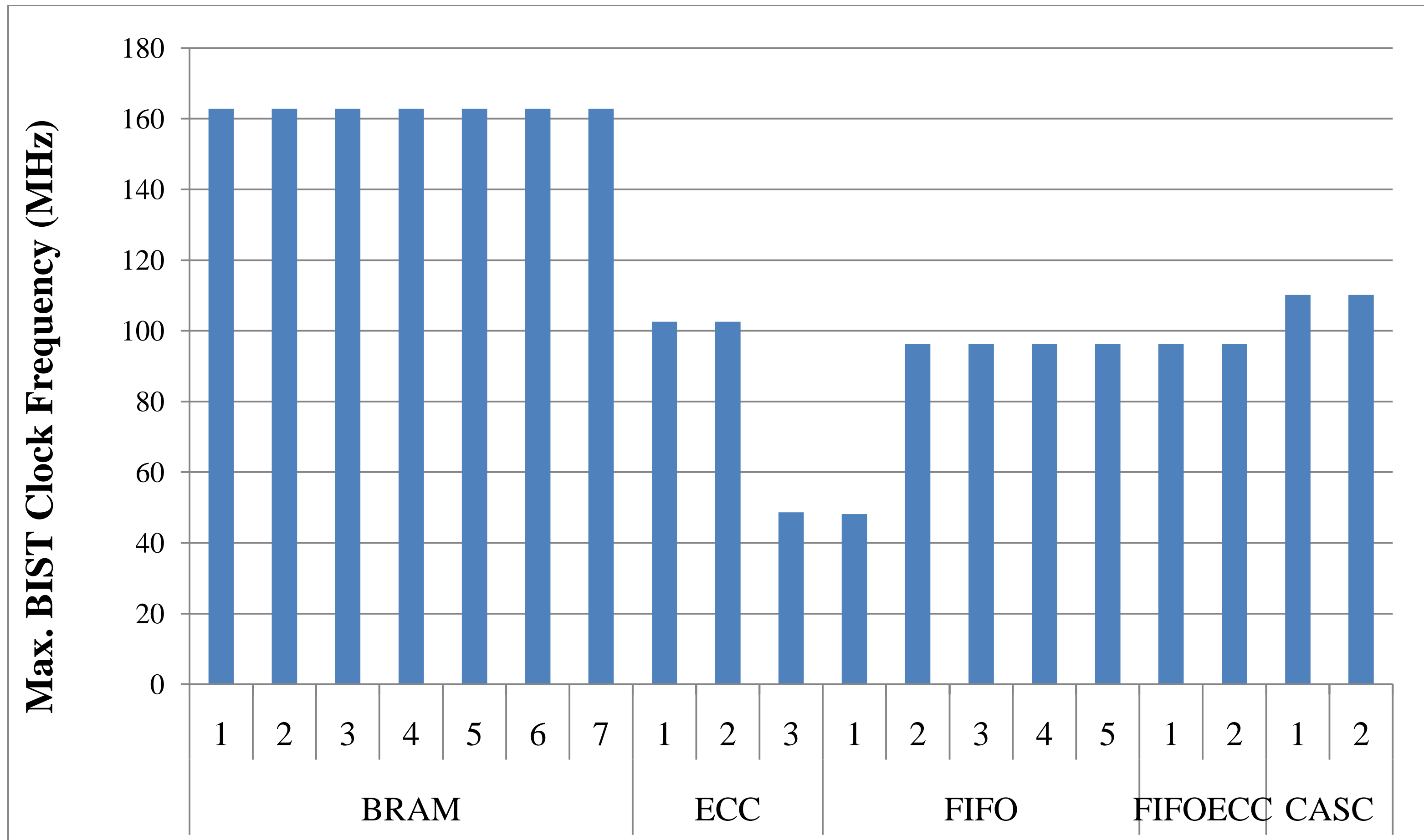


Figure 3-11 - Maximum BIST Clock Frequencies for LX30T

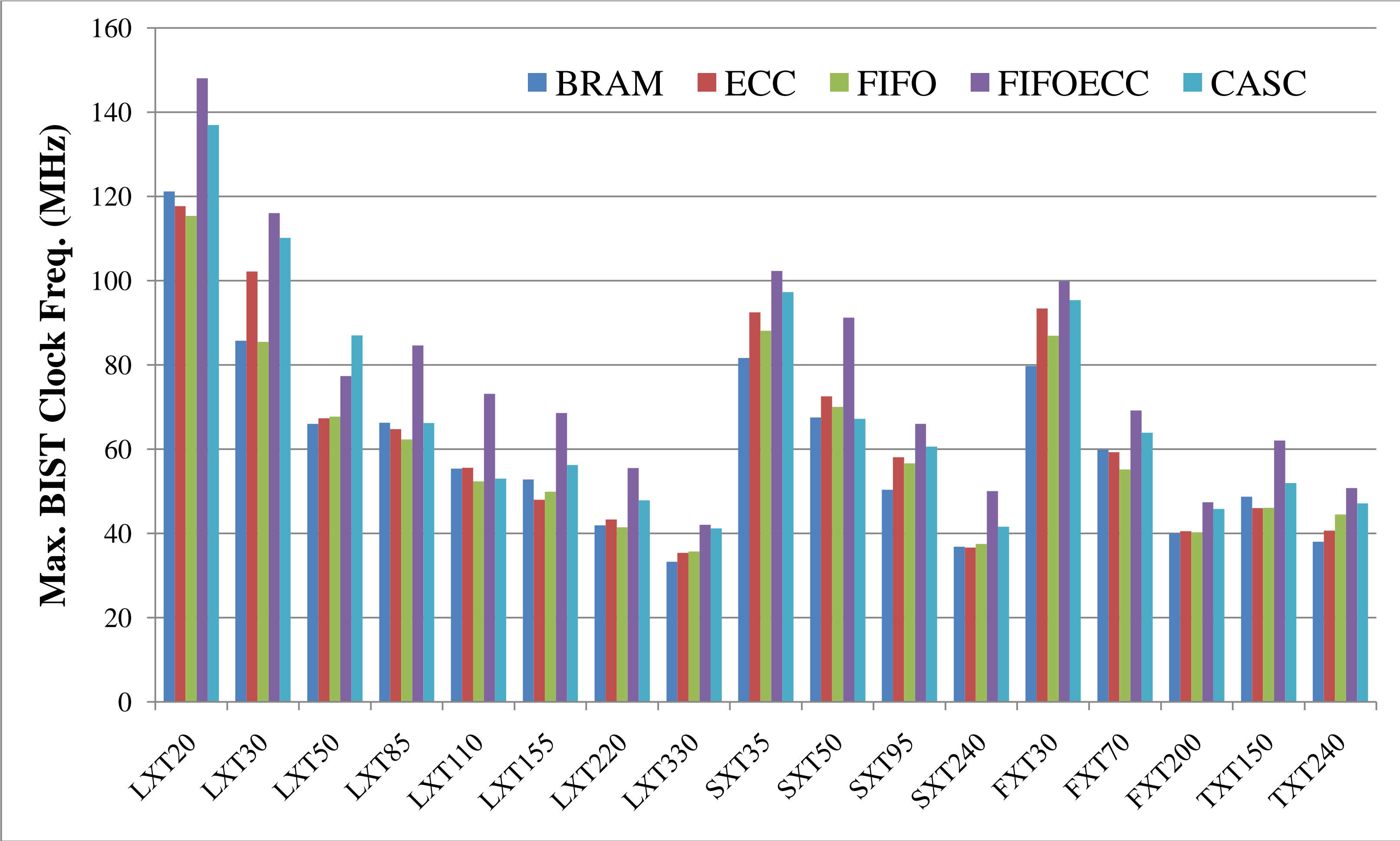


Figure 3-12 – Maximum BIST Clock Frequency for select Virtex-5 Devices

Chapter 4 Summary and Conclusions

4.1 Summary of Virtex-5 BRAM BIST

This thesis presents the development and verification of a BIST for the BRAMs contained in Virtex-5 FPGAs. The work done in this thesis is largely based on the BRAM BIST designs for Virtex-4 presented by Milton in [2] and Garrison in [3]. This design of the Virtex-5 BRAM BIST builds directly on the test design proposed by Garrison in [3].

In order to sufficiently test the embedded BRAMs, tests are run on the memories in five separate configuration modes. The BRAM mode of operation requires seven total test configurations. The ECC mode requires three test configurations. The FIFO mode demands five separate test configurations. Finally, the FIFOECC and CASC modes require two configurations each, for a total of 19 test configurations. These BIST configurations each contain a pair of identical TPGs designed to perform the required tests on the RAM along with ORAs to observe the results of the tests. The configurations also contain a boundary scan interface for communication with the BIST circuitry and retrieval of the test results.

By using the compressed configuration and partial reconfiguration features of the Virtex-5 FPGAs, the BIST configurations have been optimized in terms of download size. This in turn reduces the total testing time by a substantial amount as a majority of testing time is attributed to

configuration downloads. Timing analysis has also been performed on the configurations to determine what the maximum BIST clock frequency is for each device.

Each BIST configuration was generated for and tested on the LX30T, LX50T, SX35T, SX50T, FX30T, and FX70T Virtex-5 devices. In order to verify the fault detection capability of the BIST, faults were injected into the BRAM configuration memory of the devices and the BIST was executed. The results of these fault injections show that the BRAM BIST detects 481 of the 488 possible BRAM configuration memory faults which gives a fault coverage of 98.57%. The BIST configurations can be downloaded and executed in-system during off-line operation and are applicable for high reliability/availability systems as well as fault-tolerant applications

4.2 Future Work

For future work in this area, this BRAM BIST design could be applied to the Spartan 6 and other families of FPGAs. Additionally, some improvements that have been made with this BIST approach may be applied to the previous approaches for the Virtex-4 device. The modification to the BRAM output routing in the cascade mode of operation could be applied to these previous test approaches. Using this improvement would simplify the TPG used for this configuration mode. It would also allow the removal of the clock enables that had been added to the Virtex-4 ORA flip-flops to prevent the expected faults from being recorded.

Bibliography

[1] S. Garimella, "Built-In Self Test for Regular Structure Embedded Cores in System-on-Chip," Auburn University, MS Thesis 2005.

[2] D. Milton, "Built-In Self Test of Configurable Memory Resources in Field Programmable Gate Arrays," Auburn University, MS Thesis 2007.

[3] B. Garrison, "Analysis and Improvement of Virtex-4 Block RAM Built-In Self-Test and Introduction to Virtex-5 Block RAM Built-In Self-Test," Auburn University, MS Thesis 2009.

[4] L. Wang, C. Stroud, and N. Touba, *System-On-Chip Test Architectures*.: Morgan Kaufmann Publishers, 2008.

[5] I. Kuon, "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, February 2007.

[6] S. Hamdioui, *Testing Static Random Access Memories*.: Kluwer Academic Publishers, 2004.

[7] "Virtex-5 FPGA User Guide," Xilinx Inc., 2010.

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

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

[10] A. van de Goor and I. Tlili, "March Tests for Word-Oriented Memories," *Trans. Design, Automation and Test in Europe*, pp. 501-508, 1998.

[11] 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.

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

[13] "Virtex-5 Family Overview," Xilinx Inc., 2009.

[14] "Xilinx Development System Reference Guide," Xilinx Inc., 2008.

[15] Xilinx Inc. ModelSim Xilinx Edition-III Details. [Online]. http://www.xilinx.com/ise/verification/mxe_details.html

[16] B. Garrison, D. Milton, and C. Stroud, "Built-In Self-Test for Memory Resources in Virtex-4 Field Programmable Gate Arrays," *Proc. ISCA International Conf. on Computers and Their Applications*, pp. 63-68, 2009.

[17] B. Dutton, A. Mustafa, C. Stroud, and J. Sunwoo, "Embedded Processor Based Fault Injection and SEU Emulation for FPGAs," *International Conf. on Embedded Systems and Applications*, pp. 183-189, 2009.

[18] "Virtex-5 FPGA Configuration User Guide," Xilinx Inc., 2010.

[19] 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.

[20] C. Stroud and S. Garimella, "A System for Automated Built-In Self-Test of Embedded Memory Cores in System-on-Chip," *Proc. IEEE Southeastern Symp. on System Theory*, pp.

50-54, 2005.

[21] C. Stroud and S. Garimella, "Built-In Self-Test and Diagnosis of Multiple Embedded Cores in SoCs," *Proc. International Conf. on Embedded Systems and Applications*, pp. 130-136, 2005.

[22] M. Pulukuri, "Built-In Self Test for Digital Signal Processor Cores in Virtex-4 and Virtex-5 Field Programmable Gate Arrays," Auburn University, MS Thesis, 2010.

[23] A. Sarvi and J. Fan, "Automated BIST-based diagnostic solution for SOPC," *Proc. International Conf. on Design and Test of Integrated Systems in Nanoscale Technology*, pp. 263-267, 2006.

[24] A. van de Goor, I. Tlili, and S. Hamdioui, "March LR: A Test for Realisted Linked Faults," *Proc. IEEE VLSI Test Symp.*, pp. 272-280, 1996.

Appendix

The following is the *MarchLR* testing algorithm with a 72-bit BDS sequence which is used to test Virtex-5 BRAMs. This algorithm was developed using the BDS method described in [10]. This sequence is created after optimizing the algorithm by removing duplicate elements as described in [10]. This optimization will result in a reduction in test time from O(70N) to O(64N), where N represents the number of address locations.

	March Element	Address Direction	RAM Operation	Data Hex Value
MarchLR	1	up/down	write	000000000000000000
	2	down	read write	000000000000000000 FFFFFFFFFFFFFFFFFF
	3	up	read write read write	FFFFFFFFFFFFFFFFFF 000000000000000000 000000000000000000 FFFFFFFFFFFFFFFFFF
	4	up	read write	FFFFFFFFFFFFFFFFFF 000000000000000000
	5	up	read write read write	000000000000000000 FFFFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFFFF 000000000000000000
	6	up	read	000000000000000000
BDS	7	up	read write write read	000000000000000000 555555555555555555 AAAAAAAAAAAAAAAAAA AAAAAAAAAAAAAAAAAA
	8	down	read write read	AAAAAAAAAAAAAAAAAA 555555555555555555 555555555555555555
	9	up	read write write read	555555555555555555 333333333333333333 CCCCCCCCCCCCCCCCCC CCCCCCCCCCCCCCCCCC
	10	down	read write read	CCCCCCCCCCCCCCCCCC 333333333333333333 333333333333333333
	11	up	read write write read	333333333333333333 0F0F0F0F0F0F0F0F0F F0F0F0F0F0F0F0F0F0 F0F0F0F0F0F0F0F0F0
	12	down	read write read	F0F0F0F0F0F0F0F0F0 0F0F0F0F0F0F0F0F0F 0F0F0F0F0F0F0F0F0F
	13	up	read write write read	0F0F0F0F0F0F0F0F0F FF00FF00FF00FF00FF 00FF00FF00FF00FF00 00FF00FF00FF00FF00
	14	down	read write read	00FF00FF00FF00FF00 FF00FF00FF00FF00FF FF00FF00FF00FF00FF

	March Element	Address Direction	RAM Operation	Data Hex Value
BDS	15	up	read	FF00FF00FF00FF00FF
			write	FF0000FFFF0000FFFF
			write	00FFFF0000FFFF0000
			read	00FFFF0000FFFF0000
	16	down	read	00FFFF0000FFFF0000
			write	FF0000FFFF0000FFFF
			read	FF0000FFFF0000FFFF
	17	up	read	FF0000FFFF0000FFFF
			write	FF00000000FFFFFFFF
			write	00FFFFFFFF00000000
			read	00FFFFFFFF00000000
	18	down	read	00FFFFFFFF00000000
			write	FF00000000FFFFFFFF
			read	FF00000000FFFFFFFF
	19	up	read	FF00000000FFFFFFFF
			write	00FFFFFFFFFFFFFFFF
			write	FF0000000000000000
			read	FF0000000000000000
	20	down	read	FF0000000000000000
			write	00FFFFFFFFFFFFFFFF
			read	00FFFFFFFFFFFFFFFF
	21	up	read	00FFFFFFFFFFFFFFFF