Analysis and Improvement of Virtex-4 Block RAM Built-In Self-Test and Introduction to Virtex-5 Block RAM Built-In Self-Test

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

Brooks Garrison

Certificate of Approval:

Vishwani D. Agrawal James J. Danaher Professor Electrical and Computer Engineering	Charles E. Stroud, Chair Professor Electrical and Computer Engineering
Victor P. Nelson Professor Mathematics and Statistics	George Flowers Dean Graduate School

Analysis and Improvement of Virtex-4 Block RAM Built-In Self-Test and Introduction to Virtex-5 Block RAM Built-In Self-Test

Brooks Garrison

A Thesis

Submitted to

the Graduate Faculty of

Auburn University

in Partial Fulfillment of the

Requirements for the

Degree of

Master of Science

Auburn, Alabama
May 9, 2009

Analysis and Improvement of Virtex-4 Block RAM Built-In Self-Test and Introduction to Virtex-5 Block RAM Built-In Self-Test

Brooks Garrison

_____

Signature of Author

_____

Date of Graduation

Vita

Brooks Garrison is the eldest son of Ricky and Rasa Garrison. He was born in Huntsville, Alabama on March 26, 1985. Brooks attended Auburn University from Fall 2003 to the Spring 2007. He was involved in the Engineering Honor Society, Tau Beta Pi ($TB\Pi$), during his junior and senior years. He was also involved in the Electrical Engineering Honor Society, Eta Kappa Nu (HKN), during that time, and served as Vice President during his senior year. He graduated in the spring of 2007 with a Bachelor's of Engineering with special emphasis on Computer Engineering, earning the honor of Magna Cum Laude.

Brooks immediately began working on his Master's of Science degree at Auburn University. His contributions to the Block RAM BIST for Virtex-4 and Virtex-5 FPGAs were developed under the direction of Dr. Charles E. Stroud.

Thesis Abstract

Analysis and Improvement of Virtex-4 Block RAM Built-In Self-Test and Introduction to Virtex-5 Block RAM Built-In Self-Test

Brooks Garrison

Master of Science, May 9, 2009
(B.S., Auburn University, 2007)

127 Typed Pages

Directed by Charles E. Stroud

A reliable method for testing embedded memories within Virtex-4 and Virtex-5 Field-Programmable Gate Arrays (FPGAs) is needed by the current FPGA community. A method for testing the Virtex-4 embedded Block Random Access Memories (RAMs) using Built-In Self-Test (BIST) was initially proposed by Milton in [7]. However, this method was found to have deficiencies in practical application. Several corrections and improvements are made to this proposed approach, which improve overall BIST generation and execution time.

A method for testing the Virtex-5 FPGA Block RAMs is proposed and the suggested configuration settings are described. Four Test Pattern Generators (TPGs) are proposed to implement the BIST, which will consist of 16 BIST configuration bit files and subsequent execution of their associated BIST sequences.

Acknowledgments

I would like to thank Dr. Stroud for his constant support and advice throughout my Master's studies. His guidance helped me become a better engineer by teaching me how to logically analyze a unknown behavior and turn it into a known behavior. The knowledge he imparted to me will be invaluable in my future career and for this, I am most grateful. I would also like to thank Dr. Agrawal and Dr. Nelson for serving as members on my graduate committee. Without the contributions of all three of these mentors, work like that presented in this thesis would not be possible. I would also like to acknowledge my colleagues, Brad, Joey, Jia and Mary, whose assistance was invaluable throughout my research.

I would also like to acknowledge my parents and family, whose support helped me persevere through the hard times in my research when I believed that my potential was lackluster at best. Most of all, I would like to thank my wife, Erin, whose constant belief in my abilities inspired me to strive to meet her expectations when I had no faith in myself.

Style manual or journal used Journal of Approximation Theory (together with the style known as "aums"). Bibliography follows van Leunen's *A Handbook for Scholars*.

Computer software used The document preparation package TEX (specifically LATEX) together with the departmental style-file `aums.sty`.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Field Programmable Gate Arrays

A Field Programmable Gate Array (FPGA) is a prefabricated Integrated Circuit (IC) that contains an array of programmable logic blocks (PLBs) and programmable input/output (I/O) cells with programmable interconnections as seen in Figure 1.1 [11]. The user has the freedom to program the functionality realized by each logic block and the connections between each logic block [9]. An FPGA is more flexible with respect to design errors than a traditional gate array or Mask Programmable Gate Array (MPGA). This is because MPGAs are only programmable in the factory by the manufacturer, while FPGAs can be reprogrammed at the user's discretion.

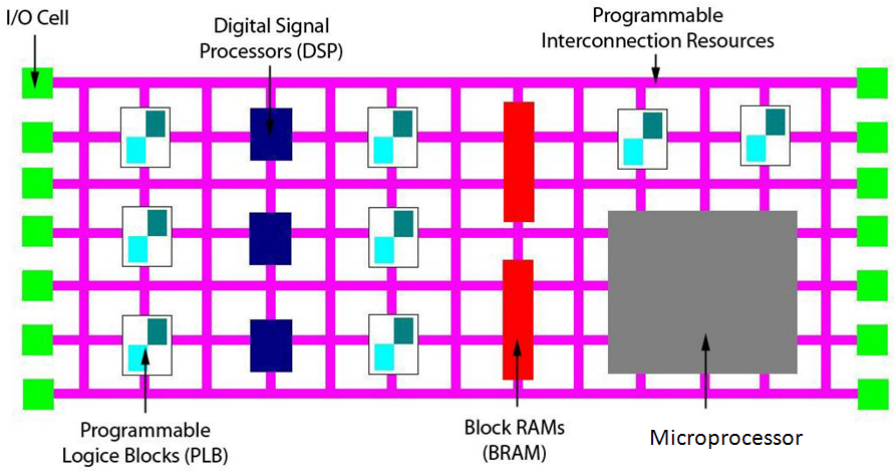


Figure 1.1: General FPGA Structure

FPGAs were developed as an alternative to Application-Specific Integrated Circuits (ASICs). ASICs are ICs developed for a specific use rather than for general use [10]. While ASICs are generally four times faster and 40 times more efficient in terms of area than FPGAs [6], they are by, nature, unforgiving if a design error is encountered. Costs are incurred through redesign, fabrication and testing of the new ASICs, which can be quite expensive in both money and time. FPGAs offer the flexibility of fixing errors in the design configuration and then simply re-downloading the design onto the device [10]. Through the use of FPGAs, overall prototyping costs and design iterations can be greatly reduced.

However, FPGAs are less dense than traditional gate arrays and MPGAs due to the fact that a lot of the FPGAs resources are spent merely to achieve the programmability. This means that FPGAs have lower performance than ASICs and MPGAs. The vast amount of programmable interconnections slow down internal signals and thus FPGAs are slower than MPGAs and ASICs [9].

1.1.1 Pros/Cons to Using an FPGA

There are both advantages and disadvantages to using FPGAs. Some of the prominent advantages of using an FPGA include the following:

- Reconfigurability
- Efficient prototyping
- Lower design costs

FPGAs provide reconfigurability to the user. If a bug is found in the design, the user can fix the design configuration and re-download it to the device. No new hardware needs to be designed or produced. This leads to lower design costs and is

optimal for prototyping. This reconfigurability can also be taken advantage of when an FPGA is incorporated into a system. For example, when a system containing the FPGA is first powered on or during times of low activity, the FPGA can be configured to test itself and/or the system to determine if there are any faults [13]. If there are none, the FPGA can then be reconfigured for the function it was intended to perform within the system.

However, the flexibility of FPGAs does not come without a cost. Some of these include the following:

- Slower device speeds
- Higher power consumption
- Volatile configuration memory
- Higher production cost

FPGAs have a multitude of programmable interconnect points within them. These points have both capacitance and resistance which slow down signals within the device causing the FPGA to run at slower speeds than ASICs or MPGAs [9]. ASICs are built for a specific purpose, while FPGAs are of a more general nature. Because FPGAs are less constrained, many of the resources (programmable interconnects, PLBs, etc) they contain are not fully utilized and are in general less power efficient than ASICs [6]. Another disadvantage to using most FPGAs is that the configuration memory must be downloaded every time the device is powered on due to the volatile nature of the configuration memory. FPGAs are very useful for prototyping and low volume designs. However, for designs that will be mass-produced, FPGAs become an expensive component to be incorporated into the design. ASICs are better suited for high volume production situations [9].

1.1.2 FPGA Implementation

Recent FPGAs are comprised of some, if not all, of the following components [13]:

- Programmable Logic Blocks (PLBs)
- Digital Signal Processors (DSPs)
- Microprocessors
- Input/Output Blocks (IOBs)
- Random Access Memories (RAMs)
- Programmable Interconnect

These components can comprise over one billion transistors [13]. Technology has advanced such that these transistors are now on the nanometer scale and the process of creating these transistors is not without error. Due the the small scale of the parts that make up the FPGA, an inexpensive and efficient way to test that the components are fault-free is needed. This thesis will be using the flexible nature of the FPGA itself in order to test that certain components, specifically the Block RAMs, were created without defects and/or have not sustained faults during system operation.

FPGA Memory

Embedded memory was incorporated onto FPGAs so that they could act as System-on-Chip (SoC) devices, which contain all the elements of a computer. This frees up PLB elements from acting as storage devices providing for the memory needs

of the circuit. The size of each Block RAM can range from 128 bits to 36 kbit and in most cases the data width versus address space can be adjusted [13]. RAMs can also be classified as single-port (SP) where reading and writing can only be implemented over a single circuit path or multi-port (MP) where memory cells can be accessed simultaneously and independently of each other [5]. The number of RAMs can vary in an FPGA. In the Virtex-4 family, the number of 18 kbit Block RAMs can range from 36 in the FX12 device to 552 in the FX140 device [15].

The introduction of Block RAMs to FPGAs also increases the overall performance of the device. Compared to an ASIC that does the same function, the FPGA with Block RAMs is approximately three times slower (compared to the four times slower without Block RAMs [6]). The power consumption of the device is reduced from 12 to 9 times more than an ASIC. Component and area utilization is also improved from 40 to approximately 20 times more than an ASIC [6].

1.2 Built-In Self-Test

Built-In Self-Test (BIST) is a way for a given circuit to test itself to determine if it is fault-free or defective. BIST consists of three main components shown in Figure 1.2 and described below. A test controller, input isolation circuitry, and some additional I/O may also be needed to run the BIST [11].

- Circuit Under Test (CUT): is the current circuit or component which is undergoing testing.
- Test Pattern Generator (TPG): the test pattern generator produces a sequence of patterns for testing the CUT.
- Output Response Analyzer (ORA): the output response analyzer compacts the output responses of the CUT into a *Pass/Fail* indication.

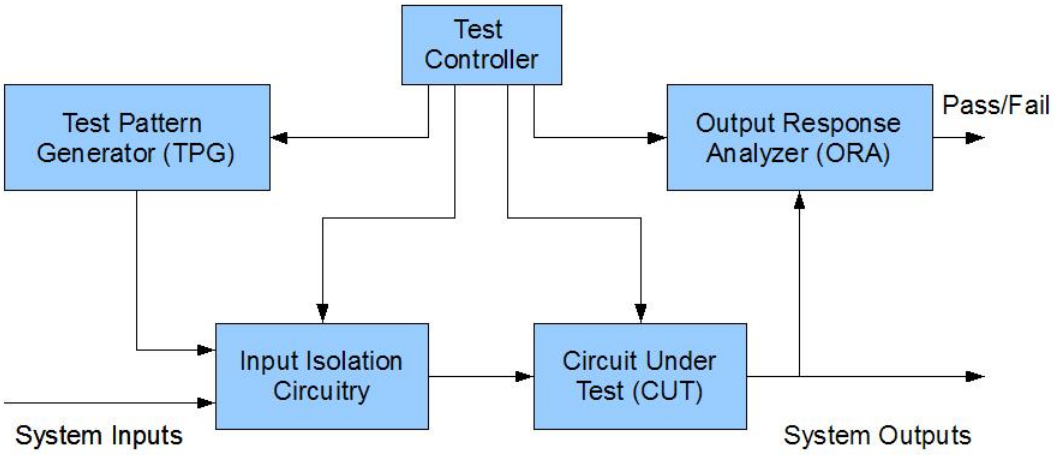


Figure 1.2: Basic BIST Structure [11]

1.2.1 Why use BIST?

There are several methods of testing a device. The first is to use an external test machine that applies a set of test vectors to the device. The other option is to design additional circuitry that makes the device easier to test or enables the device to test itself [11].

Additionally, all the components of the BIST can be clocked using the system clock. This is referred to as *at-speed* testing and is important because it can help facilitate the detection of faults that lead to excessive delay in an otherwise working CUT. This also allows testing to occur at the maximum clock frequency, which in turn minimizes testing time and thus testing cost [11].

Another advantage of using BIST is that the need for expensive external automatic test equipment (ATE) is greatly reduced. The cost of the ATE depends on the number of test vectors, desired test speed, and the number of I/O pins required by the CUT. Overall testing costs and time are reduced because the ATE requires only

a few signals to initiate and control the BIST sequence as well as retrieve the BIST results [11].

However, BIST does have some negative aspects. Additional circuitry is typically incorporated into the device to implement the BIST. This leads to a larger device area, known as *area overhead*, and thus increased cost. This larger area overhead also leads to longer signal routing paths within the device, which can affect performance [11]. Another disadvantage is that in addition to designing and verifying proper operation of the intended system, the BIST system must also be designed and verified. Two systems rather than one need to be designed and both must be valid. Although, one BIST system can be used with many application circuits.

1.2.2 BIST for FPGAs

In an ASIC, additional circuitry must be added to the original circuit design to incorporate the BIST circuitry. However in an FPGA, this is not needed because all of the components of the BIST can be constructed using the inherent components of the FPGA. In this thesis, PLBs compose the TPGs and ORAs while the Block RAMs will compose the CUTs. Since all of the components of the BIST reside within the FPGA, they all use the same clock, and at-speed testing can be taken advantage of. This eliminates any performance penalties that might have been introduced if additional testing hardware had been developed.

One of the main advantages for performing BIST on FPGAs is that BIST can easily be applied to all levels of testing, which is known as *vertical testability*. BIST also provides high diagnostic resolution by being able to specify which CUT a fault is associated with [11].

1.3 Thesis Statement

The goal of this thesis is to discuss improvements made to the Virtex-4 Block RAM BIST to address several problems found within the approach developed by Milton in [7]. A technique for implementing Block RAM BIST on Virtex-5 devices based on the techniques developed for the Virtex-4 Block RAM BIST will also be introduced and discussed.

Chapter 2 will present a more detailed overview of RAM architecture, namely the architecture and operation of Block RAMs in the Virtex-4 and Virtex-5 FPGAs. Previous Block RAM BIST methods will be described along with how they were applied to Virtex-4 RAM BIST. Chapter 3 will present the work done to correct and improve BIST for Virtex-4 Block RAMs, while Chapter 4 will present and discuss the BIST technique proposed for Virtex-5 Block RAMs. Chapter 5 will summarize the thesis and provide suggestions for future research and development.

Chapter 2

Background Information

This chapter will give an introduction to static random access memories (RAMs) and the faults associated with them. This will include a discussion on how the faults develop and implementation of known RAM test algorithms. The RAM test algorithms used for BIST of Block RAMs in Virtex-4 [7] will be discussed, along with how they test for faults. The architecture of Virtex-5 FPGA will be discussed, along with the architecture and operation of Block RAMs.

2.1 Random Access Memory

Static RAMs (SRAMs) are constructed from memory cells that have two different voltage levels, one for logic 0 and one for logic 1. This type of circuit is known as a *bi-stable* circuit. A unique feature of SRAMs is that they are a *volatile* type of memory, which means that they can only store data as long as they are supplied with power [5]. When the power is turned off, all data in the SRAM is lost.

Figure 2.1 illustrates a functional model of a multi-port SRAM. The SRAM consists of a memory cell array that is usually composed of the memory cells shown in Figure 2.2. Each cell has a shared read/write capability and each port has two bit lines that are used by the sense amplifier for the read/write operations. In addition to the memory cell array, the SRAM has row and column address decoders that select the memory cell from which to read and write. When data is being read, the read circuitry loads the data from the selected memory cell into the *Data Flow* register

and it is then sent out on the *Data-word Out* line. If data is being written, it is loaded into the *Data Flow* register from the *Data-word In* line and the write circuitry writes the data to the selected memory cell [5].

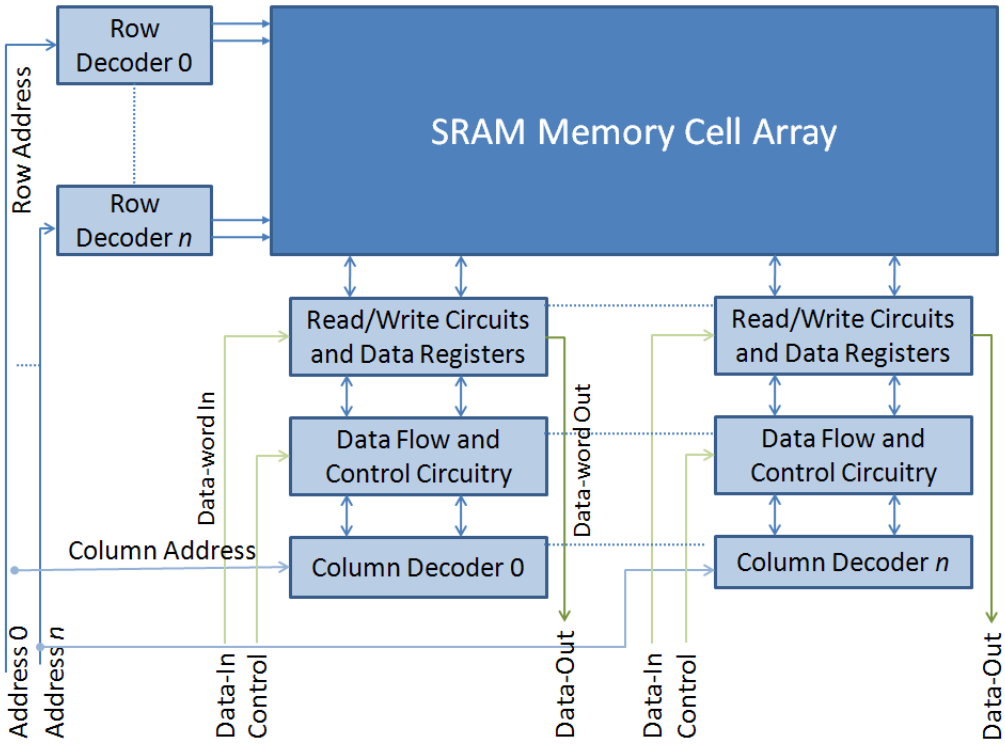


Figure 2.1: Functional Model of a Multi-Port SRAM [5]

2.1.1 Faults

Since SRAMs have many components (Figure 2.1), the need to test these components is crucial. RAMs are generally tested for faults by writing and reading various test patterns known as *march tests* to each of the memory addresses [5]. The fault detection results of these tests are described by *functional fault models*, which are abstract fault models that can be described by *fault primitives*. Fault primitives make it possible to precisely define a functional fault model. When testing SRAMs for faults,

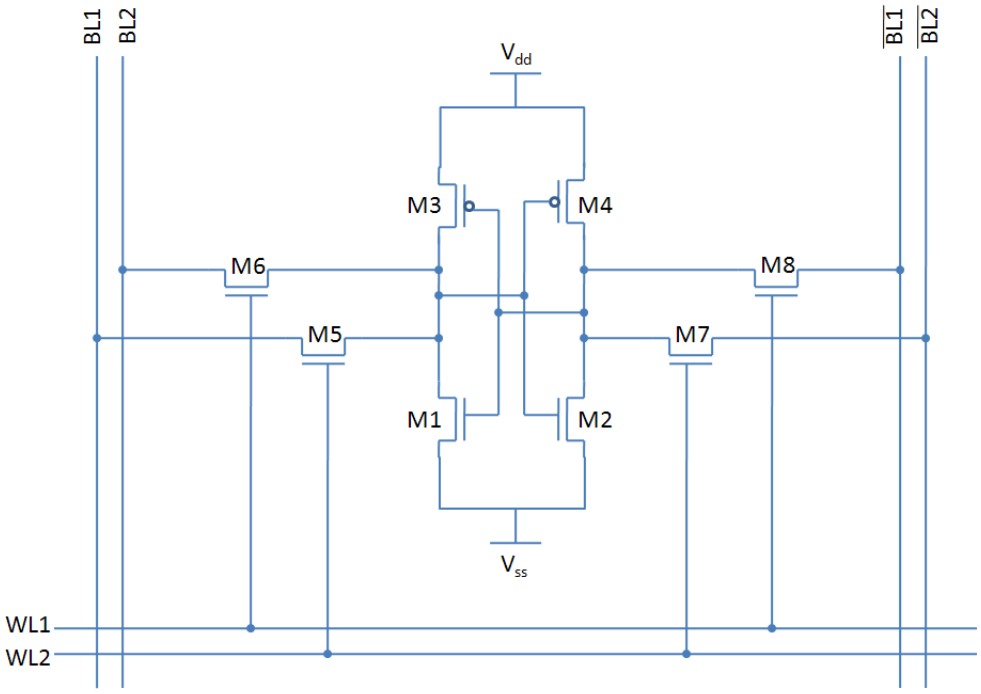


Figure 2.2: Differential Access Multi-Port Cell [5]

every possible fault need not be tested, but rather the possible *realistic* faults need to be tested [5]. There are several ways to classify memory faults. These include:

Static vs. Dynamic Faults

Static faults are characterized by the effect of a single operation to a cell. Examples of static faults include Stuck-at One/Zero. Dynamic faults are characterized by the effect of multiple operations [5].

Simple vs. Linked Faults

Simple faults are faults that have no influence on other faults and thus the behavior of one fault does not influence the behavior of another fault. This is important to note because simple faults cannot mask one another, that is hide their behavior.

Linked faults are faults that can influence other faults and therefore fault masking can occur [5].

Single-Port vs. Multi-Port Faults

Faults that require at most one port operation to occur are said to be single-port faults. These faults can occur in both single- and multi-port memories. Multi-port faults can only occur due to operations performed on two or more ports. For example, if two simultaneous read operations performed on a memory cell cause the contents of the cell to change, a multi-port fault is said to have occurred [5].

Single-Cell vs. Multi-Cell Faults

When a sequence of operations is performed on a memory cell, if a fault is sensitized in that cell, the fault is said to be a single-cell fault. However, if the operations cause a fault to be sensitized in a different memory cell, the fault is said to be a multi-cell fault, which is also known as a coupling fault [5]. Some common single-cell faults include:

- Stuck-At Faults (SAF): these faults are characterized by the logic value of a memory cell or line always being either logic 0 or logic 1 [1].
- Stuck-Open Faults (SOF): these faults are characterized by not being able to access the memory cell, usually due to an open word line [1].
- Transition Faults (TF): these faults are characterized by a memory cell not being able to transition from a logic 0 to a logic 1 or vice versa [1].
- Data Retention Faults (DRF): these faults are characterized by a memory cell not being able to store its logical value after a certain amount of time [1].

Some common coupling faults which involve two cells, $cell_1$ and $cell_2$, include:

- Inversion Coupling Faults (CFin): these faults are characterized by a write operation that causes the logic value of $cell_1$ to flip, which causes the value of $cell_2$ to flip [1].
- Idempotent Coupling Faults (CFid): these faults are characterized by a write operation that causes the logic value of $cell_1$ to flip, which causes $cell_2$ to always become either logic 0 or logic 1 [1].
- State Coupling Faults (CFst): these faults are characterized by a memory cell or line being forced to a certain value if another coupled cell or line is in a certain state [1].

2.2 FPGA Block RAM BIST

BIST for FPGA Block RAMs has been previously implemented for the Virtex-4 FPGAs. Milton [7] developed TPGs that would test each of the four modes of operation for the Virtex-4 BRAMs (regular operation, First-In-First-Out or FIFO operation, Error Correcting Code or ECC operation, and cascade operation). Milton's work was derived from the work first presented by Garimella for Virtex I and Virtex II FPGA Block RAMs [4].

Garimella tested the Block RAMs in Virtex II by testing the single-port mode and then the multi-port mode as proposed in [5]. His BIST configurations are shown in Table 2.1. Garimella used the *MarchLR* RAM test algorithm to test the single-port mode and *March s2pf/d2pf* to test the dual-port mode of the Virtex II Block RAMs [4]. He used a single TPG to generate the test patterns and apply control signals to the Block RAMs under test. Milton's approach differed from Garimella's in

that he implemented two identical TPGs that drive alternating rows of Block RAMs under test, which provided for greater fault detection capabilities. The use of two TPGs allowed for the detection of possible errors within the TPGs as well as those that could be present in the Block RAMs. Garimella was the first to implement circular comparison based ORAs (discussed in Section 2.2.1) when testing embedded memories [4] and Milton adopted this when implementing his tests for Virtex-4 [7].

However, a major disadvantage of Garimella's approach was that his BIST was implemented completely in VHDL. While this did allow for a shorter development time, there was an incurred cost of having to download each full BIST configuration to the device under test. Milton also noted that constructing the BIST circuitry entirely in VHDL reduces the control of the physical design of the BIST circuitry [7]. The synthesis tools cannot guarantee any degree of similarity between the configurations and thus does not allow for the exploitation of partial configuration files, which greatly reduce download and test time. Furthermore, they can implement the behavior of signals in a way that, while correct in implementation, does not configure the Block RAM signals as expected and as a result, the Block RAM is not completely tested [7].

Milton overcame these problems by implementing his BIST circuitry in both VHDL, for the high-level TPG model, and Xilinx Design Language (XDL), to control the placement of the BIST circuitry so as to take advantage of partial reconfiguration. By modifying the XDL for the BIST configuration, Milton was able to ensure that the placement and routing of his TPGs, ORAs and CUTs was constant and that the only changes were to the Block RAMs' configuration between each BIST configuration [7]. This consistency allowed for a drastic decrease in BIST download time when using partial reconfiguration.

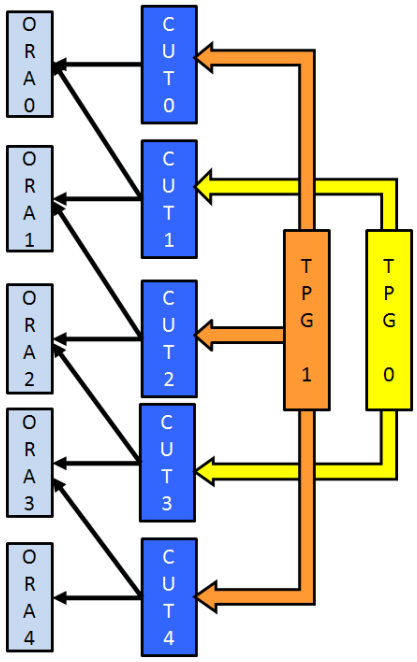
Table 2.1: Virtex II BIST Configurations [4]

BIST Configuration	Test Algorithm	Address Locations (A)	Data Width (D)	Clock Cycles
1	MarchLR w/ BDS	512	36	$58 \cdot A$
2	March LR	1k	18	$14 \cdot A$
3		2k	9	$14 \cdot A$
4		4k	4	$14 \cdot A$
5		8k	2	$14 \cdot A$
6		16k	1	$14 \cdot A$
7	March s2pf	512	36	$14 \cdot A$
8	March d2pf	512	36	$9 \cdot A$

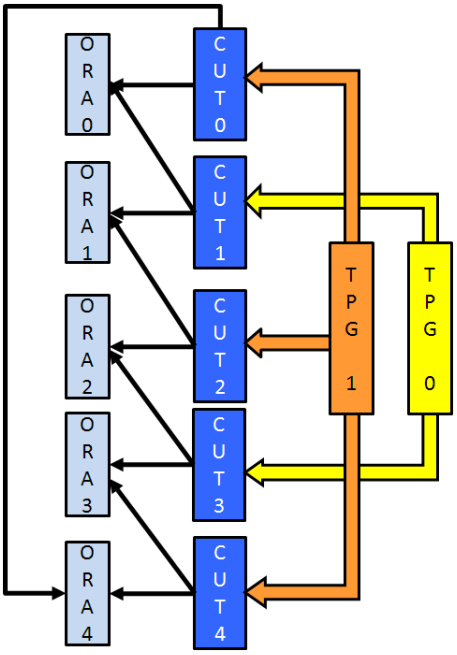
2.2.1 ORA Implementation

A basic comparison based ORA implementation can be seen in Figure 2.3(a) [12]. This ORA implementation compares the outputs of a CUT with the outputs of adjacent CUTs. If a fault is detected by the ORAs, the CUT it originated in can be determined by observing which ORAs detected the fault [12]. For example, if ORA_2 detected a fault and ORA_1 also detected a fault, then CUT_2 is where the fault occurred. However, this approach is limited by a fault detected in the edge ORA (ORA_0). If a fault is detected by ORA_0 only, then the fault is in CUT_0. However, if both ORA_0 and ORA_1 detect a fault, then CUT_1 is known to be faulty, but the functionality of CUT_0 is unknown.

A solution to this is a circular comparison based approach, which is shown in Figure 2.3(b) [13]. This approach compares the top CUT with the bottom CUT and effectively every CUT is being compared with two other CUTs. This allows for the exact determination of a fault location [12]. The circular comparison based approach was implemented by Milton in the Virtex-4 Block RAM BIST in [7] and will be adopted for the Virtex-5 Block RAM BIST.



(a) Comparison Based ORAs



(b) Circular Comparison Based ORAs

Figure 2.3: Comparison Based ORAs

2.2.2 Overview of Virtex-4 Block RAMs

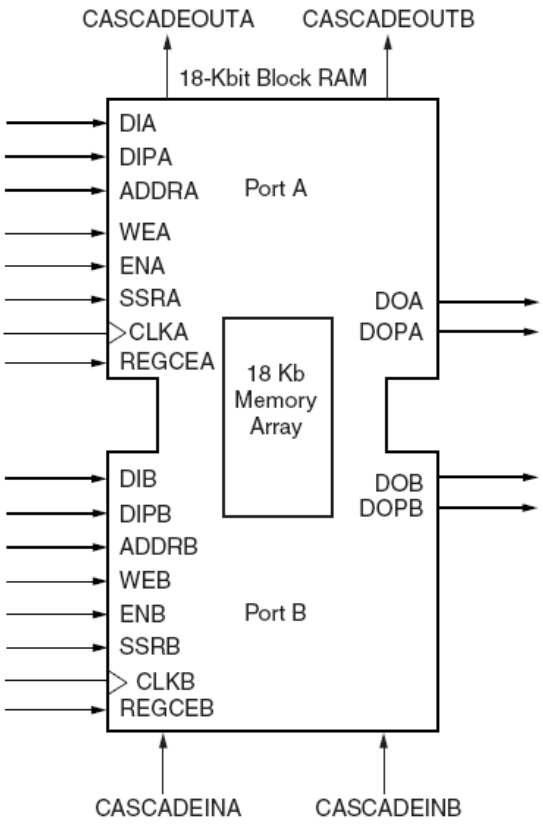


Figure 2.4: Dual-Port Data Flows [15]

Virtex-4 Block RAMs can store up to 18 kbit of data. Each Block RAM has two ports (Port A and Port B) with which it can independently synchronously read and/or write data to the Block RAM. Each port can be programmed to have different read and write widths as seen in Table 2.2. There are three writing modes of operation: WRITE_FIRST - where the input data is simultaneously written into the memory and the data output; READ_FIRST - where the data previously stored at the memory location is observable on the output while the data is being written to memory; and NO_CHANGE - where the outputs remain unchanged during a write operation [15].

The Virtex-4 Block RAMs also have the capability of using a pipeline register for the output data, which allows for higher clock rate at the expense of an additional clock cycle of latency.

Another feature of the Virtex-4 Block RAMs is that two adjacent RAMs may be cascaded together to create a 32Kx1-bit RAM. An adjacent RAM pair can also be configured in a single 512x64-bit error correcting code (ECC) mode. In both of these modes of operation, care must be taken by the user to account for PowerPC modules included within the FX devices. For instance, ECC configuration cannot be implemented in the Block RAMs immediately above or below these PowerPC modules [15]. This is because these Block RAMs do not have an associated adjacent Block RAM to construct the ECC RAM.

Table 2.2: Virtex-4 BRAM Port Aspect Ratio [15]

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 2.3: Virtex-4 FIFO Port Aspect Ratio [15]

Address Width	Address Bits	Memory Depth	Data Width	Data-In/Out Bits	Data-In/Out Parity Bits
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

Lastly, the Virtex-4 Block RAMs can be configured as First-In-First-Out (FIFO) RAMs. Virtex-4 devices contain dedicated pointer logic to implement this mode of operation. When configured as a FIFO, the BRAMs have a smaller set of configurable read/write widths shown in Table 2.3. They also have independent read and write clocks where data is read/written on the programmable active edge of the respective clock. Several flags have been incorporated to aid the user in determining the fullness of the FIFO: FULL/EMPTY and ALMOSTFULL/EMPTY. There are two operating modes: standard - where the first word written to the FIFO will not appear on the output until a read operation has been performed and First Word Fall Through (FWFT) - where the first word written appears immediately on the output [15].

There are several functions to test to ensure the Block RAMs are fault-free [7]. These include:

- Memory contents.
- Read/write data widths.
- Write mode operation.
- Pipeline register functionality.
- Control signals and active levels.
- Dedicated circuitry for a specific mode of operation (i.e. FIFO pointers and flags, cascade routing).

2.2.3 BIST for Virtex-4 Block RAMs

This section will present a more detailed discussion of the BIST configurations that were implemented for the Virtex-4 Block RAMs [7]. Each FPGA consists of

columns of configurable logic blocks (CLBs), Block RAMs (BRAMs) and digital signal processors (DSPs). Milton was concerned with only the CLBs and Block RAMs. The CLBs were used to construct both the TPGs and the ORAs, while the BRAMs were the circuits under test (CUTs) [7].

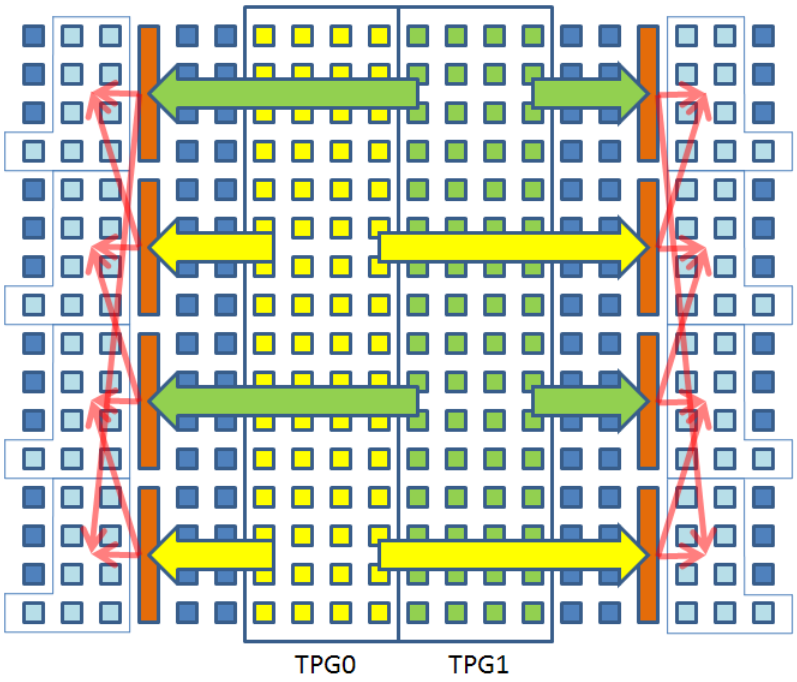


Figure 2.5: BRAM - TPG/ORA Connections [7]

Milton determined that he would use two identical TPGs. Each TPG would drive alternating rows of Block RAM, as shown in Figure 2.5. Circular comparison-based ORAs were also used as this would increase fault detection and diagnostic resolution. It can also be seen in the figure that nine ORAs (light blue squares) are used for each Block RAM. The outputs of a Block RAM are compared with the same outputs of the Block RAM directly above and below it [7].

BRAM

The first set of BIST configurations created by Milton were for the normal mode of operation for the BRAM. Most of the tests performed on the Block RAMs were done in this mode of operation as it provides the most flexibility. Later BIST configurations sought to test the circuitry for other modes of operation.

Milton implemented several known RAM tests: *MarchLR with Background Data Sequence (BDS), March s2pf/d2pf,* which were also used by Garimella to test the Virtex-II Block RAMs, and *MATS+*. The *MarchLR* algorithm was chosen because it can detect the classical stuck-at faults in addition to pattern sensitivity faults, intra-word coupling faults, and bridging faults [5].

The following notation is used to describe RAM test algorithms [5].

- $\uparrow, \downarrow$: Indicates the direction traveled through the address space ($\updownarrow$ indicates that the direction can be either up or down).
- *r, w*: Indicates a read or write operation, respectively, and is followed by the value expected to be read or the value to be written.
- Each group of operations in parentheses, called a march element, indicates the operations that are performed on a single address. For example, $\downarrow (r0, w1)$;, indicates that the address space will travel from the maximum address to the minimum address. For each address location, a *Read - 0* operation will be performed, followed by a *Write - 1* operation.

The MarchLR algorithm is given by Equation 2.1 and is order $O(14N)$, where N is the number of address locations.

$$
\begin{aligned}
MarchLR = & \\
& \{\updownarrow (w0)\,; \\
& \downarrow (r0, w1)\,; \\
& \uparrow (r1, w0, r0, w1)\,; \\
& \uparrow (r1, w0)\,; \\
& \uparrow (r0, w1, r1, w0)\,; \\
& \uparrow (r0)\}
\end{aligned}
\tag{2.1}
$$

For word-oriented memories, such as those found in the Virtex-4, background data sequences (BDS) are needed to detect the faults within each word of the memory. The number of BDS is given by Equation 2.2, where K is the number of bits in a data word [13].

$$
N_{BDS} = \lceil log_2 K \rceil + 1 \tag{2.2}
$$

For a 4-bit BDS example using the MarchLR algorithm, first expand the algorithm (2.1) to incorporate the 4-bit words. That is replace all *r0, r1, w0, w1* elements with *r0000, r1111, w0000,* and *w1111* respectively. Using Equations 2.3 and 2.4 and Tables 2.4 and 2.5, the BDS march elements can be constructed by the following steps [2].

$$BDS_1 = r_{Di},\ w_{Di+1},\ r_{Di+1} \tag{2.3}$$

$$BDS_2 = r_{Di},\ w_{Di+1},\ w_{Di+2},\ r_{Di+2} \tag{2.4}$$

Table 2.4: 4-Bit BDS Components

Normal	**Inverse**
0000	1111
0101	1010
0011	1100

Table 2.5: 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 2.5, use Equation 2.3 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 2.4 to the second row and as such, Equation 2.4 is used to

create the march element rather than Equation 2.3. The resulting equation is (r_2, w_3, w_4, r_4) and the march is $\{\uparrow (r0000, w0101, w1010, r1010)\}$. When a transition like this occurs, i is incremented by 2.

4. Using $i = 4$, Equation 2.3 is used since there is no transition of rows between $i = 4$ and $i = 5$, which results in the equation (r_4, w_5, r_5) and the march element $\{\downarrow (r1010, w0101, r0101)\}$.

5. Equation 2.4 is 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. Since there is no transition between $i = 6$ and $i = 7$, Equation 2.3 is used to get the equation (r_7, w_8, r_8) and the next march element is $\{\downarrow (r1100, w0011, r0011)\}$.

7. The last march element for the additional BDS marches consists of a read operation of the last i value, $\{\uparrow (r0011)\}$.

The MarchLR algorithm with BDS is given in Equation 2.5. Notice that the seventh and eighth lines of Equation 2.5 contain redundant march elements that are contained within the initial MarchLR marches. These can be eliminated and the optimized MarchLR algorithm with BDS is shown in Equation 2.6 [2]. The order for *MarchLR* with 4-bit BDS is $O(38N)$ and the order for the optimized *MarchLR* with 4-bit BDS is $O(34N)$.

$$
\begin{aligned}
MarchLR\ with\ BDS = \\
&\{\updownarrow\,(w0000)\,; \\
&\downarrow\,(r0000, w1111)\,; \\
&\uparrow\,(r1111, w0000, r0000, r0000, w1111)\,; \\
&\uparrow\,(r1111, w0000)\,; \\
&\uparrow\,(r0000, w1111, r1111, r1111, w0000)\,; \\
&\uparrow\,(r0000)\,; \\
&\uparrow\,(r0000, w1111, r1111)\,; \\
&\downarrow\,(r1111, w0000, r0000)\,; \\
&\uparrow\,(r0000, w0101, w1010, r1010)\,; \\
&\downarrow\,(r1010, w0101, r0101)\,; \\
&\uparrow\,(r0101, w0011, w1100, r1100)\,; \\
&\downarrow\,(r1100, w0011, r0011)\,; \\
&\uparrow\,(r0011)\}
\end{aligned}
\tag{2.5}
$$

$$
\begin{aligned}
MarchLR\ with\ BDS = \\
&\{\updownarrow (w0000)\,; \\
&\downarrow (r0000, w1111)\,; \\
&\uparrow (r1111, w0000, r0000, r0000, w1111)\,; \\
&\uparrow (r1111, w0000)\,; \\
&\uparrow (r0000, w1111, r1111, r1111, w0000)\,; \\
&\uparrow (r0000, w0101, w1010, r1010)\,; \\
&\downarrow (r1010, w0101, r0101)\,; \\
&\uparrow (r0101, w0011, w1100, r1100)\,; \\
&\downarrow (r1100, w0011, r0011)\,; \\
&\uparrow (r0011)\}
\end{aligned}
\tag{2.6}
$$

For the dual port RAM tests, some additional notation is needed [5].

- The colon operator (*:*) separates the operation to the two ports.
- n : Indicates that no operation should be applied to a particular port.
- - : Indicates that any operation is allowed, as long as it does not cause a conflicting pair (i.e. dual write operations to the same address locations with different values).
- $\overset{N-1}{n=0}$: Indicates that an operation is performed on either the row or column range specified, where *N,n* is R,r for a row range and C,c for a column range, respectively.

- $n_{r,c}$: Indicates a particular operation, where n is r for read and w for write, on a memory cell with the row, r, and the column, c.

March s2pf/d2pf: These RAM tests were chosen because they can detect all realistic single and double addressing faults for a dual port RAM [5]. The order for *March s2pf* is $O(14N)$ and the order for *March d2pf* is $O(9N)$. The algorithms for these tests are shown in Equations 2.7 and 2.8 respectively.

$$
\begin{aligned}
March\ s2pf = \\
&\{\updownarrow (w0 : n)\,; \\
&\uparrow (r0 : r0, r0 : -, w1 : r0)\,; \\
&\uparrow (r1 : r1, r1 : -, w0 : r1)\,; \\
&\downarrow (r0 : r0, r0 : -, w1 : r0)\,; \\
&\downarrow (r1 : r1, r1 : -, w0 : r1)\,; \\
&\downarrow (r0 : -)\}
\end{aligned}
\tag{2.7}
$$

$$
\begin{aligned}
March\ d2pf = & \\
& \{\updownarrow (w0 : n)\,; \\
& \\
& \uparrow_{c=0}^{C-1} (\uparrow_{r=0}^{R-1} (w1_{r,c} : r0_{r+1,c}, \\
& r1_{r,c} : w1_{r+1,c},\ w0_{r,c} : r1_{r+1,c}\ r0_{r,c+1} : w0_{r+1,c}))\,; \\
& \\
& \uparrow_{c=0}^{C-1} (\uparrow_{r=0}^{R-1} (w1_{r,c} : r0_{r,c+1}, \\
& r1_{r,c} : w1_{r,c+1}, w0_{r,c} : r1_{r,c+1},\ r0_{r,c} : w0_{r,c+1}))\}
\end{aligned}
\tag{2.8}
$$

MATS+: This RAM test was chosen because it is a simple and fast algorithm that will be used for the various address and data widths to test the programmable address decoding circuitry [13]. The order for $MATS+$ is $O(5N)$ and the algorithm can be seen in Equation 2.9.

$$
\begin{aligned}
MATS+ = & \\
& \{\updownarrow (w0)\,; \\
& \uparrow (r0, w1)\,; \\
& \downarrow (r1, w0)\}
\end{aligned}
\tag{2.9}
$$

Milton wrote a VHDL model for his TPG which would implement the tests described previously. Table 2.6 lists the BIST configurations and settings devised by Milton for the regular mode of operation. Two MarchLR configurations are implemented, but the second configuration is only executed for 512 clock cycles. The first

BIST configuration is a MarchLR with BDS. It has a read width of 36 bits and a write width of 36 bits. The choice of these widths allows for all data inputs and outputs, 32 data bits + 4 parity bits, to be tested. The second configuration performs the first march element of the *MarchLR* algorithm, *write 0*. In this configuration, the Block RAMs are configured with the Write Mode of *READ_FIRST*, which places the previous contents of the memory cell being written on the output bus when a write operation is performed. This allows for the Block RAM initialization values to be tested [7].

Note that not all of the possible address configurations are tested at this time, as the ones not present for this set of BIST configurations will appear in one of the later sets of BIST configurations. It can also be observed that the three write modes of operation (READ_FIRST, WRITE_FIRST, and NO_CHANGE) and the use of the data out pipeline register are being tested as well. The last five columns indicate the activate levels for the RAM control signals whose descriptions can be seen in Table 2.7 [7].

After all of these BIST configurations have been run, the entire memory array will have been tested for faults. Also each of the ports has been tested to ensure that simultaneous operations do not affect the memory contents and several of the configurable address/data widths have had their programmable row/column decoders tested [7].

The nine ORA CLBs can be seen in Figure 2.6. Each of the colored slices within a CLB (blue shapes) contains two bits which correspond to the RAM outputs shown on the leftmost side of the figure.

Table 2.6: BIST Configurations and Settings [7]

(a) Settings Part 1

Config. #	BIST	Read / Write Width	Address Width	Pipeline Register	Data Out CLK Invert	RAM Extension	Write Mode
1	MarchLR	36	512	No	No	None	READ_FIRST
2	MarchLR	36	512	No	No	None	READ_FIRST
3a	March (s2pf)	36	512	Yes	No	None	READ_FIRST
3b	March (d2pf)	36	512	Yes	No	None	READ_FIRST
4	MATS+	1	16k	Yes	No	None	READ_FIRST
5		8	4k	Yes	Yes	None	NO_CHANGE
6		36	512k	Yes	No	None	WRITE_FIRST

(b) Settings Part 2

Config. #	BIST	Write Enable (WE)	Output Register (REGCE)	Port Enable (EN)	Set/ Reset (SSR)	Clock (CLK)	Init Val
1	MarchLR	Active High	Active High	Active High	Active High	Active High	A
2	MarchLR	Active High	Active High	Active High	Active High	Active High	5
3a	March (s2pf)	Active High	Active High	Active High	Active High	Active High	A
3b	March (d2pf)	Active High	Active High	Active High	Active High	Active High	A
4	MATS+	Active High	Active High	Active High	Active High	Active High	5
5		Active Low	Active Low	Active Low	Active Low	Active High	A
6		Active Low	Active Low	Active Low	Active Low	Active Low	5

Table 2.7: Virtex-4 Block RAM Dual-Port Names and Descriptions [15]

Port Name	Description
DI[A, B]	Data Input Bus
DIP[A, B]	Data Input Parity Bus
ADDR[A, B]	Address Bus
WE[A, B]	Write Enable
EN[A, B]	When inactive, no data is written to the block RAM and the output bus remains in its previous state.
SSR[A, B]	Set/Reset
CLK[A, B]	Clock Input
DO[A, B]	Data Output Bus
DOP[A, B]	Data Output Parity Bus
REGCE[A, B]	Output Register Enable
CASCADEIN[A, B]	Cascade input pin for 32K x 1-bit mode
CASCADEOUT[A, B]	Cascade output pin for 32K x 1-bit mode

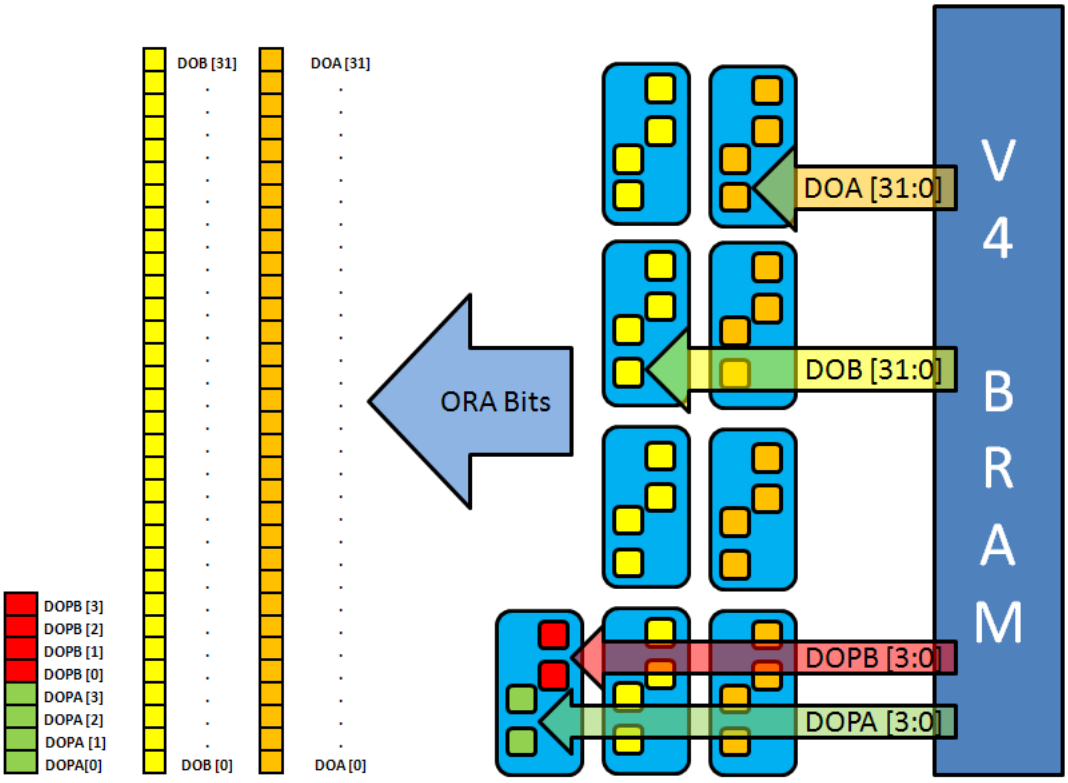


Figure 2.6: Virtex-4 Block RAM ORA Bit Assignments [7]

FIFO

Milton developed the second set of BIST configurations for when the Block RAMs were configured as FIFOs. In the Virtex-4 devices, there is dedicated logic that has been implemented so that Block RAMs can be utilized as FIFOs without the need of external CLB logic [15]. It is this additional logic that needs to be tested.

Table 2.8: FIFO Flag Assertion/Deassertion Clock Cycle Latency [15]

FIFO Output	Assertion		Deassertion	
	Standard	FWFT	Standard	FWFT
EMPTY	0	0	3	4
FULL	1	1	3	3
ALMOST EMPTY	1	1	3	3
ALMOST FULL	1	1	3	3
READ ERROR	0	0	0	0
WRITE ERROR	0	0	0	0

Milton implemented the following algorithm to test the logic associated with the FIFO mode of operation of the Block RAMs [7]. This order of this algorithm is $8N$, where N is the number of address locations.

1. Reset the FIFO
2. For 0 - (N-1),
   (a) Write a word of all zeros
   (b) Observe the *EMPTY* flag deassertion according to Table 2.8
   (c) Observe the *FULL* flag assertion according to Table 2.8
3. For 0 - (N-1),
   (a) Read a word of all zeros

(b) Perform three NO-OP operations - this allows the *FULL* flag to deassert before the write sequence

(c) Write a word of all ones

(d) Write a word of all ones - this second write asserts the *WRERR* flag for a single clock cycle

4. For 0 - (N-1),

(a) Read a word expecting all ones

Milton developed the BIST configurations seen in Table 2.9. Using each of these configurations allows for the test of each of the flags described previously and the additional special logic implemented for the Virtex-4 FIFOs. The twelve 4k x 4-bit configurations fully test the *ALMOSTFULL/EMPTY* flags by utilizing the dynamic partial reconfiguration ability of the Virtex-4 FPGA. Each configuration is clocked only enough to ensure that the flags undergo a Logic 1 to Logic 0 transition and then the next configuration is downloaded and the process is repeated. The bit assignments for the FIFO ORAs can be seen in Figure 2.7.

ECC

Milton's third set of BIST configurations is for the ECC mode of operation for the Block RAMs. When configured in this mode, two adjacent RAMs are connected to create a 512 x 72-bit RAM (Figure 2.8), which uses 8-bit Hamming code to detect double-bit errors and correct single bit errors [15]. The top-level view of the ECC mode can be seen in Figure 2.9. It should be noted, that in this mode, RAMs located in the rows immediately above and below a PowerPC module cannot be utilized [15].

Table 2.9: FIFO BIST Configurations [7]

Config. #	FIFO Mode	RST INV	RDCLK INV	RDEN INV	WRCLK INV	WREN INV	Write Mode	ALMOST FULL	ALMOST EMPTY
1	2k x 9-bit	INV	INV	INV	INV	INV	FWFT	15	2043
2	512 x 36-bit	not INV	not INV	not INV	not INV	not INV	Standard	15	496
3	1k x 18-bit	not INV	not INV	not INV	not INV	not INV	Standard	5	507
4	4k x 4-bit	not INV	not INV	not INV	not INV	not INV	Standard	5	5
5	4k x 4-bit	not INV	not INV	not INV	not INV	not INV	Standard	6	7
6	4k x 4-bit	not INV	not INV	not INV	not INV	not INV	Standard	8	8
7	4k x 4-bit	not INV	not INV	not INV	not INV	not INV	Standard	16	16
8	4k x 4-bit	not INV	not INV	not INV	not INV	not INV	Standard	32	32
9	4k x 4-bit	not INV	not INV	not INV	not INV	not INV	Standard	64	64
10	4k x 4-bit	not INV	not INV	not INV	not INV	not INV	Standard	128	128
11	4k x 4-bit	not INV	not INV	not INV	not INV	not INV	Standard	256	256
12	4k x 4-bit	not INV	not INV	not INV	not INV	not INV	Standard	512	512
13	4k x 4-bit	not INV	not INV	not INV	not INV	not INV	Standard	1024	1024
14	4k x 4-bit	not INV	not INV	not INV	not INV	not INV	Standard	2048	2048
15	4k x 4-bit	not INV	not INV	not INV	not INV	not INV	Standard	4092	4092

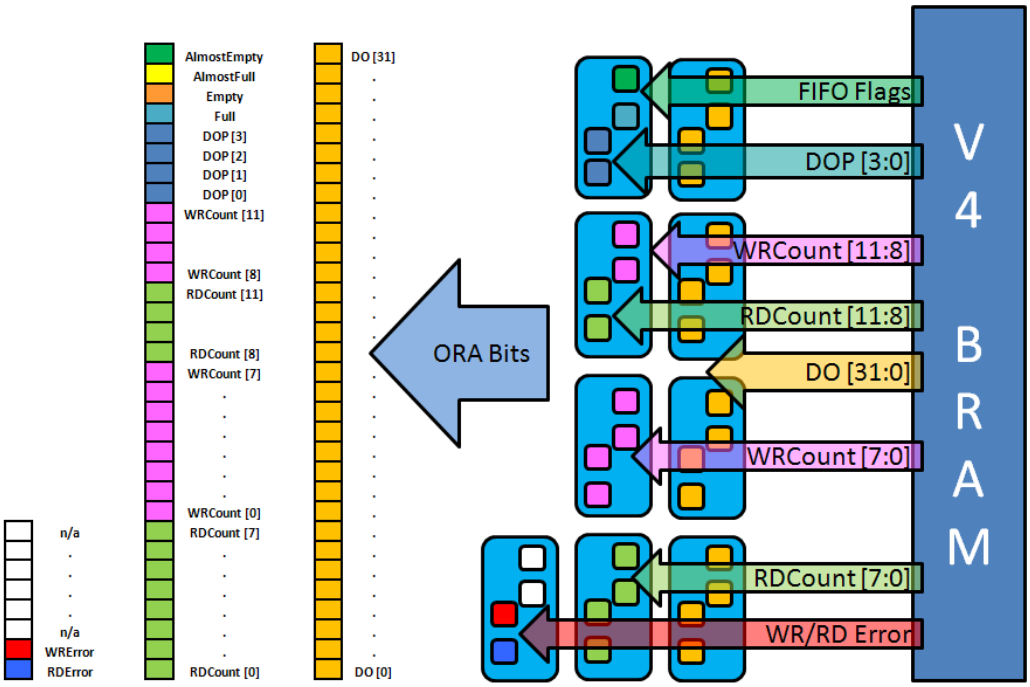


Figure 2.7: FIFO ORA Bit Assignments [7]

Since this mode requires *two* RAMs, an upper and a lower, the ORA connections are different from the previous two sets of BIST configurations (Figure 2.8). In this mode, the outputs of the lower ECC RAM are compared to the outputs of the ECC RAM in the *pair* directly above it and the same with the upper RAM outputs [15]. The TPGs are connected such that the same TPG controls an ECC RAM pair, so the two TPGs drive alternating *pairs* of ECC RAMs [7].

Milton designed his TPG to fully test the ECC encode logic. Milton surmises that the ECC encode logic consists of an XOR parity tree, but states that the parity tree structure used by Xilinx for the Virtex-4 is unknown [7]. However, a method for testing an XOR parity tree whose structure is unknown, described in [13], was implemented by Milton. 100% fault coverage can be achieved using test vectors of the following structure for an N-bit parity tree [13].

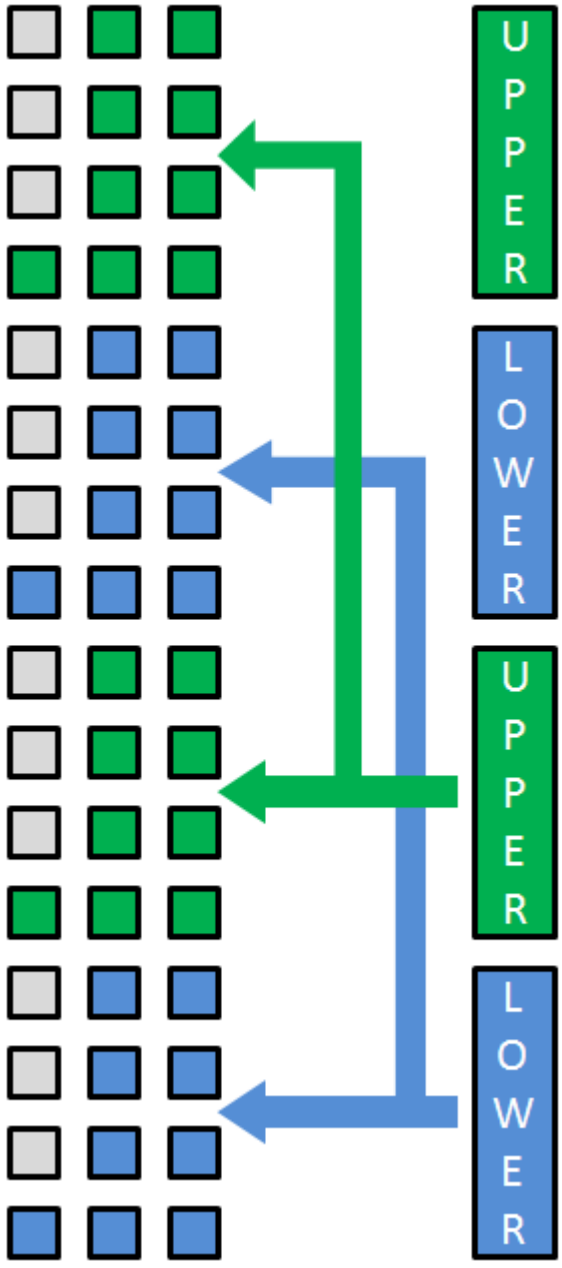


Figure 2.8: ECC/Cascade RAM ORA Comparisons [15]

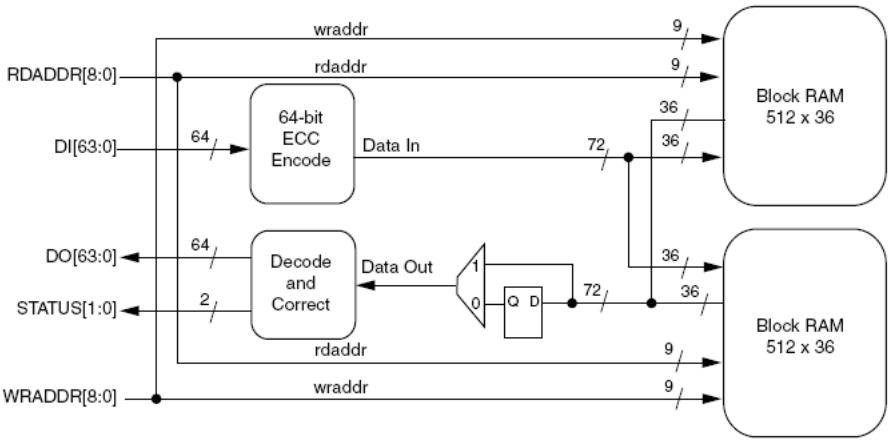


Figure 2.9: ECC RAM Top-Level View [15]

1. All zeros vector
2. All combinations of a one in a field of zeros
3. All combinations of *two* ones in a field of zeros
4. Initialize the RAM with all 2^N values, where N is the number of Hamming bits. (For Virtex-4, $N = 8$)

Vector sets 1-3 test the ECC encode and detect circuitry, while the fourth vector set tests the ECC correct circuitry [13]. Since the parity connections are unknown, Milton proposed to initialize the Hamming bits within the ECC RAMs themselves. This will result in the ECC RAMs detecting double-bit memory errors and both detecting and correcting single-bit errors when the RAM's memory locations are read [7]. Milton implemented his BIST in two phases. In the first phase, the ECC RAMs are initialized with all of the 2^8 possible Hamming values, while setting the data to all zeros. This was thought to be the only way to introduce all possible single and double-bit errors into the RAM. The RAM is then read and the single/double-bit errors are observed. In the second phase, the ECC encode circuitry is tested by

writing the test vector sets 1-3 to the memory and then reading them back. This tests the ECC encode and detect circuitry. The bit assignments for the ECC ORAs can be seen in Figure 2.10.

The eight Hamming input bits were found to be bits *DIB[17:16]* and *DIB[1:0]* on the upper Block RAM inputs and bits *DIB[31:30]* and *DIB[15:14]* on the lower Block RAM inputs. The ECC RAMs have these bits initialized in the first BIST configuration. The Block RAM ECC also has two output status bits, *STATUS[1:0]*, which correspond to bits *DOA[31]* and *DOA[0]*, respectively. These bits indicate whether or not a bit error was detected and if so, whether or not the bit error was a single- or double-bit error, as shown by Table 2.10.

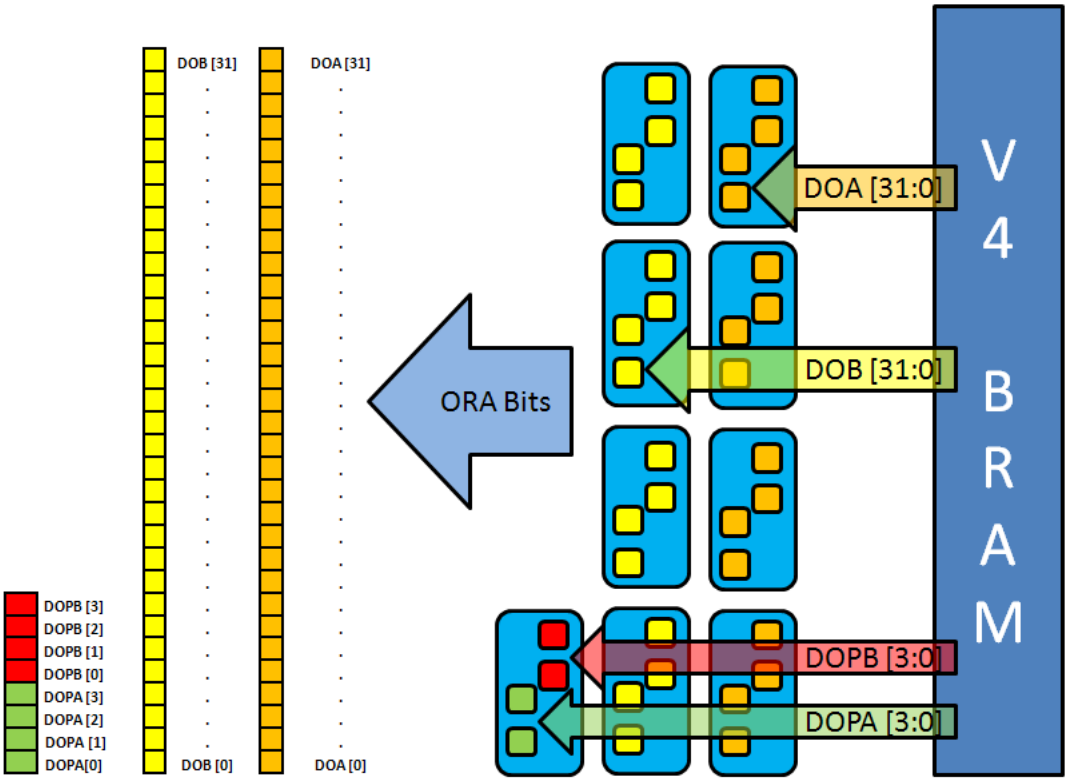


Figure 2.10: ECC ORA Bit Assignments [7]

Table 2.10: Status Bits for Virtex-4 ECC Mode of Operation

Status 1	Status 0	Description
0	0	No Bit Error
0	1	Single-bit Error - Corrected by Circuitry
1	0	Double-bit Error - Detected by Circuitry
1	1	Undefined

Cascade

Any two adjacent Block RAMs can also be configured such that they are cascaded together to form a 32k x 1-bit RAM [15]. Figure 2.11 shows the structure of the cascaded RAMs. In this configuration, one RAM is dedicated as the upper RAM and the other as the lower, similar to the ECC RAM configuration (Figure 2.8). However, this mode does not require the lower RAM to always be an even row. For this configuration, Milton places testing emphasis on the address decoder and that opposite logic values can be read/written. A MATS+ test algorithm (described in the BRAM section on page 28) is applied to only two addresses that span the two RAMs since the memory core has already been thoroughly tested by previous BIST configurations [7].

In this mode of operation, several 'failures' are expected in specific ORAs, see Figure 2.12 (pink squares). The ORAs associated with the RAMs that lie along the bottom of a device and those that lie directly above a PowerPC module will indicate these false failures. This is due to unconnected cascade routing into these RAMs [7]. Essentially, the CASCADEIN shown in Figure 2.11 is not connected for these RAMS and thus the output of the corresponding flip-flop will be different than the flip-flop output of the RAM above it, as that RAM does have a CASCADEIN. The bit assignments for the cascade ORAs can be seen in Figure 2.12.

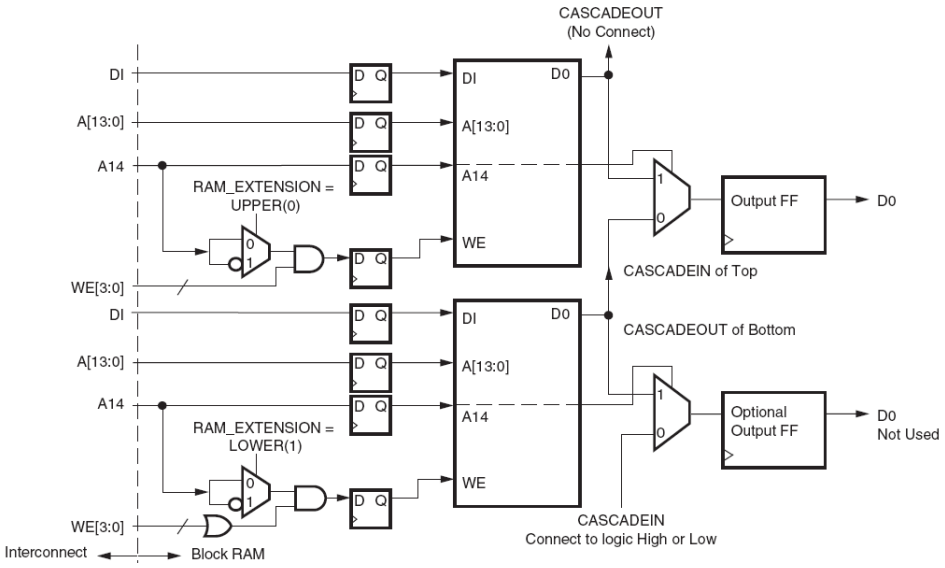


Figure 2.11: Cascadable Block RAM [15]

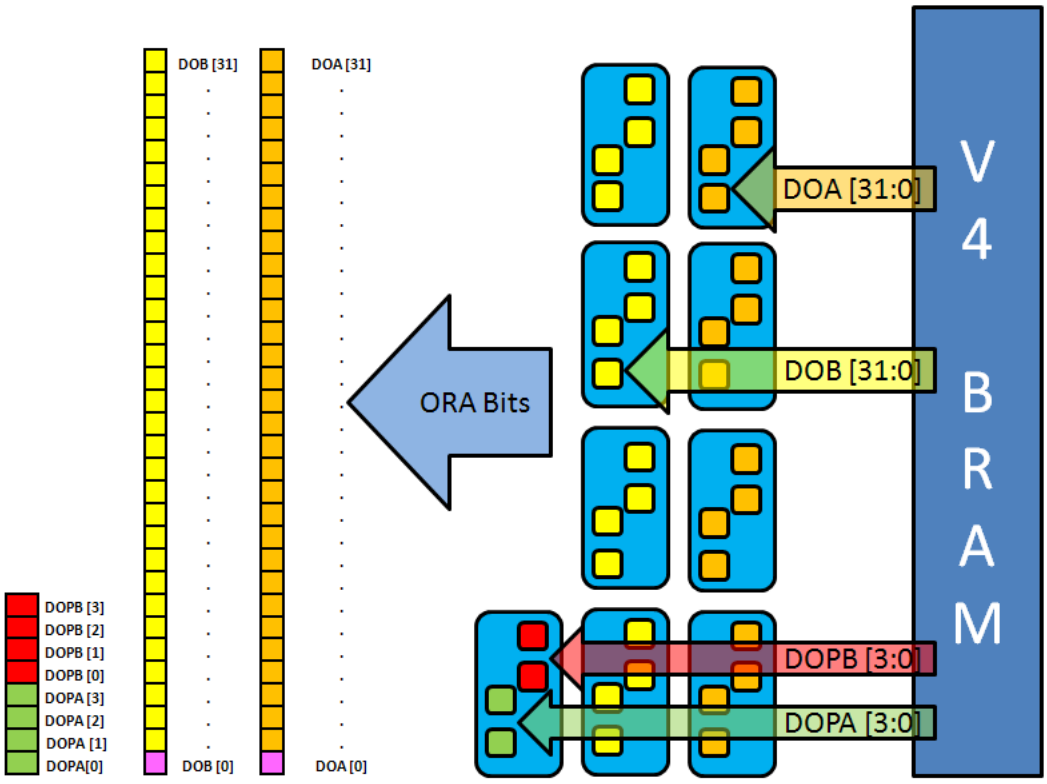


Figure 2.12: Cascade ORA Bit Assignments [7]

2.3 Virtex-5 Introduction

This section provides a general overview of the Virtex-5 FPGA, including CLBs and Block RAMs. More detail on certain aspects of the Virtex-5 Block RAMs will be discussed in the chapter that deals with that specific BIST implementation.

2.3.1 FPGA Architecture

Programmable Logic Blocks

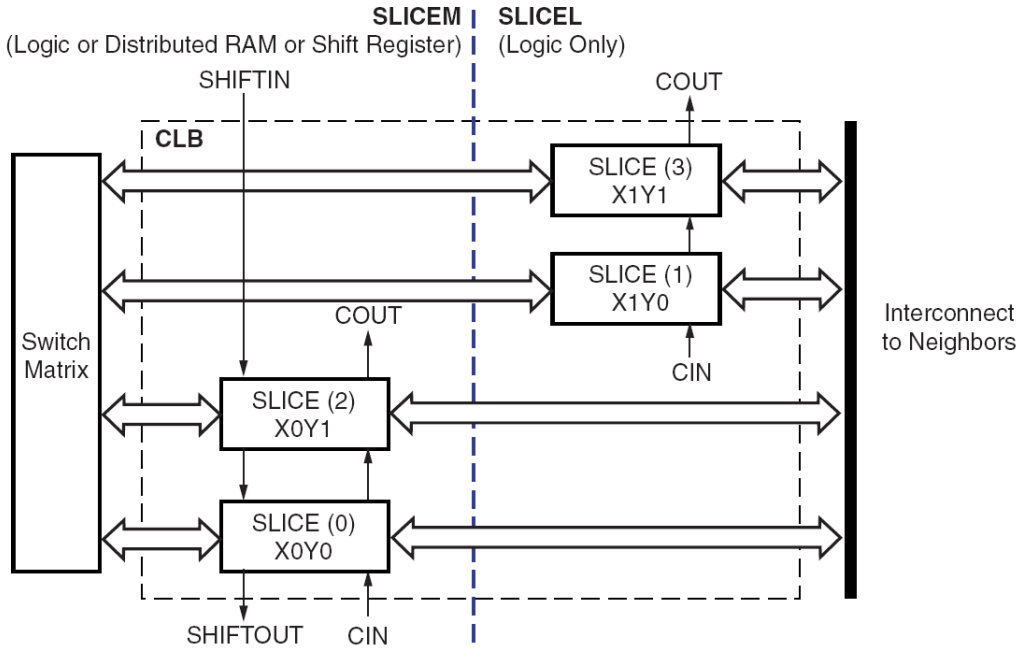
The Programmable Logic Blocks in the Virtex-5 consist of two types of slices, SliceL and SliceM. Each slice contains the following:

1. Four logic function generators (LUTs)
2. Four storage elements
3. Wide-function multiplexers
4. Carry logic

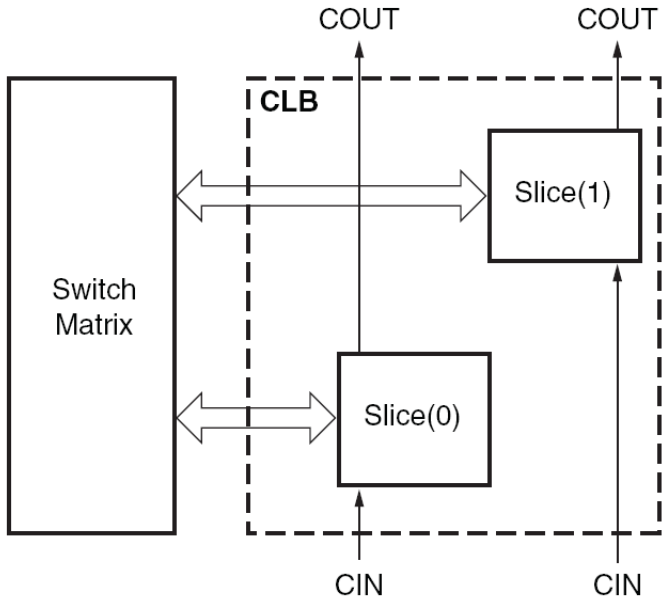
Table 2.11: CLB Contrast Virtex-4 vs. Virtex-5 [15] [16]

Component	Virtex-4	Virtex-5
Slices	4	2
LUTs	8 (4-input)	8 (6-input)
FFs	8	8
Arithmetic and Carry Chains	2	2
Distributed Ram	64-bits	256-bits
Shift Registers	64-bits	256-bits

The CLBs in the Virtex-5 are improved from the Virtex-4. A brief contrast between the Virtex-5 and Virtex-4 CLBs can be seen in Table 2.11 and in Figure 2.13. The Virtex-5 CLBs have two fewer slices than the Virtex-4. However, they have



(a) Virtex-4 CLB Arrangement [15]



(b) Virtex-5 CLB Arrangement [16]

Figure 2.13: Virtex-4 vs. Virtex-5 Slice Comparison

the same number of flip-flops. In effect, the flip-flops that were in Slices 2 and 3 were merged into Slices 0 and 1, respectively.

Block RAMs

The Virtex-5 Block RAMs are capable of storing up to 36 K-bit of data. This can be accomplished as one 36 kbit RAM or two independent 18 kbit RAMs. Table 2.12 shows the various address/data widths that the Block RAMs can be configured in the independent 18 kbit RAM mode, while Table 2.13 shows the various address/data widths that the Block RAMs can be configured as in the 36 kbit RAM mode. The Virtex-5 Block RAMs have the capability of being either single- or dual-port RAMs in either the 36 kbit or independent 18 kbit RAM modes [16].

The Virtex-5 Block RAMs also support both ECC and cascaded RAM modes. The ECC mode of operation is similar to the Virtex-4 in that the RAM uses an 8-bit Hamming code with overall parity to determine if there is a single- or double-bit error present in the output data. However, the Virtex-5 requires only one Block RAM to implement the 512 x 64-bit ECC mode rather than two adjacent Block RAMs. Also, in the Virtex-5 devices, the ECC encoder and decoder can be accessed directly, which will allow for greater control when testing these aspects of the Block RAM. In a cascaded mode, the Virtex-5 Block RAMs create a 64k x 1-bit RAM. This is twice as much as in the Virtex-4 cascaded RAM mode, but the connections between the Block RAMs are quite similar [16].

FIFO support is also available in the Virtex-5 and can be configured as seen in Table 2.14. The FIFOs have the same flags as the Virtex-4 FIFO RAMs. However, the *FULL* flag no longer has a latency associated with it. The Virtex-5 FIFO RAMs can also be configured in an ECC mode as well. This option was not available for the Virtex-4 FIFO [16].

Table 2.12: Virtex-5 BRAM Port Aspect Ratio (18 kbit RAM) [16]

Address Width	Address Bus	Memory Depth	Data Width	Data-In/Out Bus	Data-In/Out Parity Bus
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 2.13: Virtex-5 BRAM Port Aspect Ratio (36 kbit RAM) [16]

Address Width	Address Bus	Memory Depth	Data Width	Data-In/Out Bus	Data-In/Out Parity Bus
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

Table 2.14: Virtex-5 FIFO Port Aspect Ratio [16]

18 kbit Mode		36 kbit 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 2.15: Virtex-5 FIFO Data Depth [16]

Data Width		Block Ram	FIFO Capacity	
18 kbit	36 kbit	Memory	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

2.3.2 Programming Tools

Several programming tools are used to construct the Block RAM BIST configurations for the Virtex-4 and Virtex-5 devices. These include several of the Xilinx provided computer-aided design (CAD) tools, described below.

1. ISE - design suite that allows design of TPGs in VHDL, synthesis of these designs, and implementation using an internal function, PACE, to restrict certain areas for use, i.e. defining a set region for the TPGs [14].

2. FPGA Editor - graphical user interface (GUI) allows visual examination and editing of the layout of BIST configurations [14].

3. Place and Route (PAR) - performs placement and routing of a specific design [14].

4. XDL - Xilinx conversion software that converts NCD files (files generated by ISE) to XDL files (Xilinx netlist description of a design) and vice versa.

5. BitGen - conversion software that converts NCD files to BIT files, binary files that contains header information as well as configuration data, or RBT files, files that are and ASCII version of the bit files, so that they may be downloaded to a device [14].

6. TRCE - a timing analysis tool that determines the maximum clock frequency at which the BIST configurations can run [14].

7. ModelSim - simulator by Mentor Graphics, that allows verification of BIST configurations to ensure proper functionality.

2.4 Thesis Statement

This chapter has presented the work by Milton on the Virtex-4 Block RAMs. This will be the foundation for the improvements implemented in the Virtex-4 Block RAM BIST and the initial work with the Virtex-5 FPGA Block RAMs. This chapter has also briefly introduced the Virtex-5 and its CLB and Block RAM capabilities.

All of the issues discussed concerning the Virtex-4 Block RAMs will need to be addressed in addition to the new challenges that the Virtex-5 presents, such as:

- Memories are larger and have more address/data width options, so test time will need to be minimized as much as possible.

- New configuration types have been introduced, such as the FIFO ECC mode, that need BIST configurations to be designed for them.

- Different CLB architecture will have to be considered when constructing the ORAs.

One goal of this thesis is a more thorough analysis of the Virtex-4 Block RAM BIST and improvements to the BIST configurations originally developed by Milton in [7], as proposed in the next chapter. Another goal is to introduce a technique for implementing a BIST architecture and to develop BIST configurations to test the Block RAMs in Virtex-5 FPGAs.

Chapter 3

Virtex-4 Improvements

This chapter will describe and discuss improvements and corrections made to the Virtex-4 Block RAM BIST. These include simplifying the BIST configuration generation procedure, adding the ability to have a single *Pass/Fail* signal for the user, an option for system-level testing using user-defined input/output pins, improving maximum BIST clock frequency, and fixing several problems found within the original Virtex-4 Block RAM BIST developed in [7]. These improvements were implemented mainly to provide increased utility to the user, as well as to address the excessive number of programs required to generate the BIST configuration bit files. Also, several modifications were made to Milton's programs to correct errors that were found within them. These include problems with the way the FIFO BIST was being reset and overcoming the expected cascade BIST failures.

3.1 BIST Generation Simplification

The Virtex-4 Block RAM BIST generation originally consisted of three programs for each of the four BIST types (BRAM, FIFO, ECC, and Cascade), which results in a total of twelve programs [7]. These three programs include:

1. A TPG extraction program, which extracts the TPG description from the synthesized TPG physical design in XDL format.

2. A BIST template generation program, which instantiates the user-specified Block RAMs, ORAs, TPGs and the routing between these components in XDL format.

3. A modification program, which reconfigures all of the instantiated Block RAMs for each desired BIST configuration.

To address the unnecessary number of TPG extraction programs, a generic TPG extraction program, *tpgxdlext.exe*, was developed, which was based on the functionality of Milton's four TPG programs. This program no longer relied on having a hierarchical model where the TPG was connected to a dummy Block RAM to preserve signal names. However, several key signals, such as the clock and reset signals, are required to have specific names, CLK and RESET, respectively. While this improvement to the Virtex-4 Block RAM BIST configuration generation had no impact on configuration download and test time, it greatly lessened user confusion during the Block RAM BIST generation process as well as on-going support and maintenance of the BIST programs. An additional benefit of having a generalized extraction program is that it has been applied to other Virtex-4 and Virtex-5 BIST configuration programs, such as Digital Signal Processor (DSP) BIST [8].

Table 3.1: Virtex-4 TPG XDL Naming Convention

BIST Type	TPG Name
BRAM	bram_tpg.xdl
FIFO	fifo_tpg.xdl
ECC	ecc_tpg.xdl
Cascade	cas_tpg.xdl

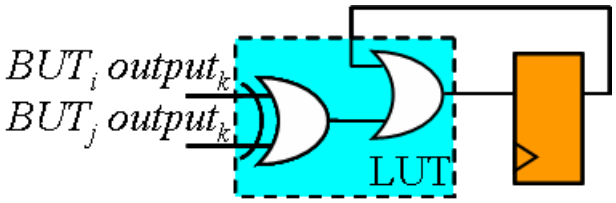
V4RamBist.exe, a generic Virtex-4 Block RAM BIST template generation program, was created to reduce the number of BIST template generation programs from

four to one. This program incorporated the functionality of Milton's four different BIST template generation programs, while also embedding the TPG extraction program, *tpgxdlext.exe*, within. This effectively hides the TPG extraction program execution from the user, reducing the number of steps required on the user's part to generate a set of BIST configurations. The only user constraint is that the TPG XDL files must be named according to the BIST type (see Table 3.1). *V4RamBist.exe* extracts, replicates, and places these TPGs, the Block RAMs and the ORAs for the desired BIST type. The program also specifies the TPG to RAM and the RAM to ORA connections and the OR-chain (discussed in Section 3.2). If the PINS interface is desired (to be discussed in Section 3.3), it adds the necessary unplaced components for the particular BIST type. Otherwise, the program incorporates the Boundary Scan interface and its routing.

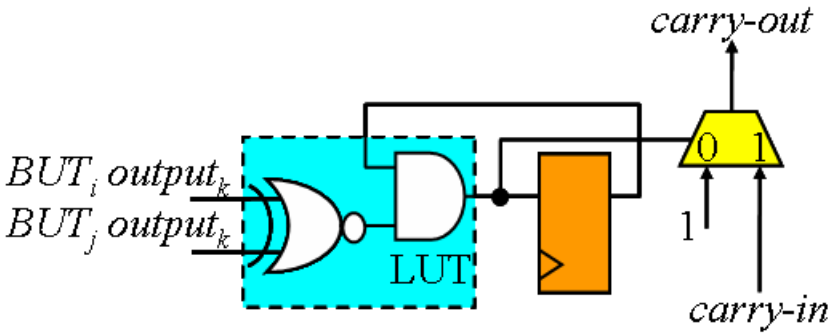
A generic Virtex-4 Block RAM BIST modification program, *V4RamMod.exe*, was also developed based on the functionality of Milton's four modification programs. This new program also resulted in reducing four programs into one. Originally, the user had to specify *Active Reconfig* when generating the configuration bit files. This was because the Block RAM contents were being initialized to the same value for the majority of the BIST configurations. When compressed or partial configuration bit files were generated using *BitGen* for the desired Block RAM BIST configuration, the Block RAM content frames would not be overwritten when the next configuration was downloaded and executed. This resulted in residual memory values acting as the initialized state of the Block RAM, which would sometimes result in false failure indications by the ORAs. The solution to this was to ensure that the initialization values were not the same between BIST configurations. This allows the user to not have to use the *Active Reconfig* option when generating the configuration bit files but increases the download time since the Block RAM content frames must be written.

3.2 ORA Modification and OR-Chain

An improvement was made to the ORAs to increase their functionality. Milton implemented the comparison shown in Figure 3.1(a). While this comparison performed adequately, the comparison shown in Figure 3.1(b) keeps the same functionality of the old ORA while taking advantage of the built-in carry chain that is within each CLB. Instead of the ORA latching a Logic 1 to indicate a fault, a Logic 0 is latched, which selects the zero input of the carry multiplexer, which is connected to a Logic 1 (Figure 3.1(b)) [3].



(a) Virtex-4 ORA (old) [7]



(b) Virtex-4 ORA (new) [3]

Figure 3.1: Virtex-4 New and Old ORAs

The BIST template generation program, *V4RamBist*, adds the connections between each CLB carry chain components to construct an iterative OR-chain. This involves using more than the nine CLBs discussed previously to implement the ORAs. In order to propagate the iterative OR-chain using the built-in carry logic, several

dummy ORAs were added to the BIST template file (Figure 3.2) and only their carry chain routing is utilized. Normally when the configuration memory is read back, these dummy ORAs would indicate a false fault detection by latching a Logic 1. However, because the normal ORAs were changed to latch a Logic 0 on a fault detection, this is no longer an issue when reading back the configuration memory to determine the number of faults.

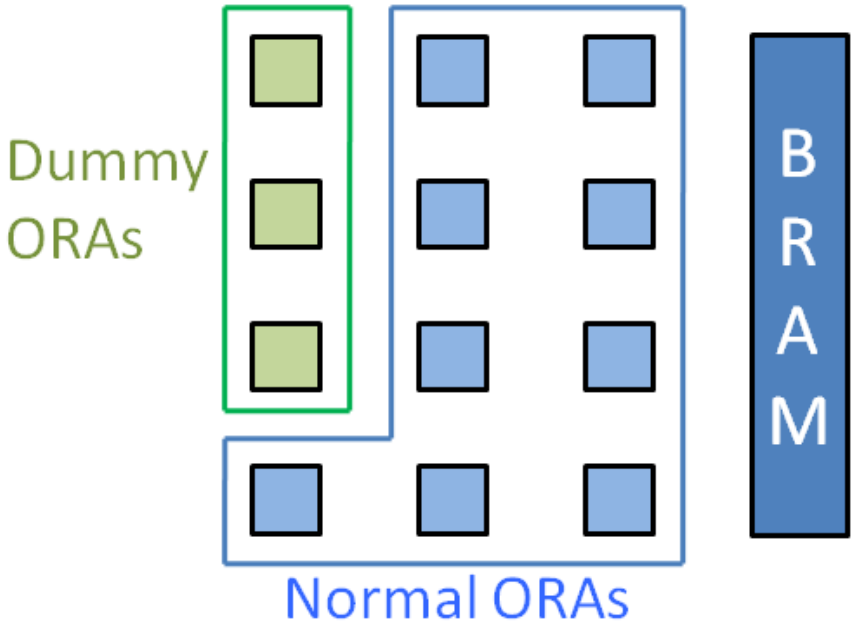


Figure 3.2: Additional Dummy ORAs

Figure 3.3 shows the functionality of these connections, where the orange boxes indicate the ORAs and their outputs. If no mismatch is detected, the ORA output is a Logic 1, which selects the carry chain output of the previous CLB. Otherwise a Logic 1 is output to the iterative OR-chain.

In effect, a single *Pass/Fail* signal has been constructed for the user to observe. At the end of a BIST sequence, the user can toggle *TDI* and observe *TDO*. If *TDO* matches the behavior of *TDI*, then no fault was detected by any of the ORAs and

the configuration memory need not be read back to obtain the contents of the ORAs. However, if *TDO* is constantly a Logic 1, then at least one ORA detected a fault. If the location of the fault(s) is desired, the user can then read back the configuration memory to determine the exact location. However, if the fault location is not an issue, merely that a fault exists, then again, the time to read back the configuration memory is eliminated.

Figure 3.3: OR-Chain Functionality

Figure 3.4 shows the connections within an FX12 device. The iterative OR-chain can be seen coming from *TDI* in the middle of the device, which also goes to the TPGs as the TPG reset signal, traveling up and down each column of ORAs and ending at the *TDO* output in the middle of the device.

3.3 PINS Option

Prior to this work, the only way to interact with the Block RAM BIST was through the Boundary Scan (BSCAN) interface. An additional interface was developed that allows a user to interact with the BIST through user-placed control pins, which will be referred to as the PINS interface. The BIST template file is generated by designating the desired options in *V4RamBist.exe* shown in Figure 3.5. Normally, the *n* or *a* option would be specified as the last input parameter. This would indicate to the program to use the BSCAN interface. Two new options *p* and *pn* were added to indicate to the program that the PINS interface is desired. In this case, the BIST template generation program does not include the BSCAN XDL module, but rather,

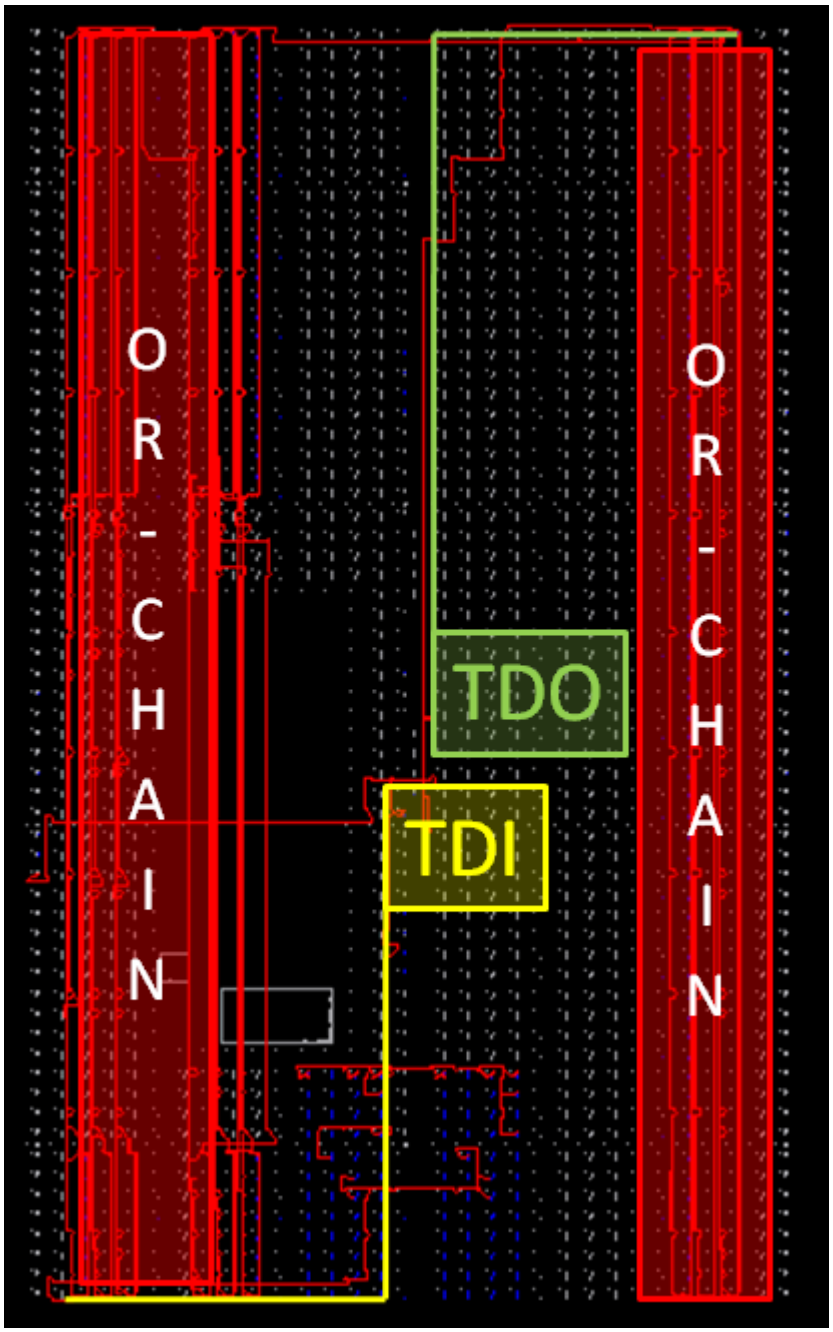


Figure 3.4: OR-Chain in an FX12 Device

produces several unplaced I/O components, which can be seen in Table 3.2, including an external clock pin for the user. The unplaced components can then be placed using FPGA Editor to desired input/output pins within the device. Special care should be taken to ensure that when the *TPG_RST* pin is placed, the corresponding *TPG_ILOGIC* component is placed with it, as shown in Figure 3.6. *TPG_RESET* keeps the TPG in a known, unchanging state so that *TDI* can be toggled and the OR-chain observed. By adding this component, a synchronization circuit is added to the asynchronous reset signal.

Table 3.2: PINS Interface - User Pins

BIST Type	BRAM	FIFO	ECC	Cascade
Provided Pins	Clock	Clock	Clock	Clock
	TDO	TDO	TDO	TDO
	TDI	TDI	TDI	TDI
	TPG_RST	TPG_RST	TPG_RST	TPG_RST
	Mode[3]	Mode[2]		
	Mode[2]	Mode[1]		
	Mode[1]	Mode[0]		
	Mode[0]			

The BRAM and FIFO BIST configurations require the user to provide control values to indicate to the TPG which BIST configuration test pattern sequence should be run. The control signal values developed by Milton in [7] can be seen in Tables 3.3 and 3.4. To reduce the number of I/O pins required by the PINS interface, the control signals *RST_LEVEL*, *WREN_LEVEL*, and *RDEN_LEVEL* were combined into a single pin (*MODE[3]*) because they are always the same value. The same procedure was performed on the FIFO signals *WE*, *REGCE*, and *EN/SSR*. The PINS interface control pins can be seen in Tables 3.5 (BRAM) and 3.6 (FIFO).

```
V4RAMbist (v2.1) - generates template file
 for Block RAM BIST configs in any Virtex-4

command line format:
V4RAMbist <xdlfile> <startrow> <startcol> <endrow> <endcol>
<dev> <part> <type> [n,a,p,pn]

 where type = bram (Block RAM mode BIST)
              fifo (FIFO mode BIST)
              ecc  (ECC RAM mode BIST)
              casc (Cascade RAM mode BIST)

dev part rows cols   dev part rows cols   dev part rows cols
lx   15   64   31    sx   25   64   55    fx   12   64   31
lx   25   96   35    sx   35   96   55    fx   20   64   47
lx   40  128   43    sx   55  128   69    fx   40   96   65
lx   60  128   61                         fx   60  128   67
lx   80  160   65                         fx  100  160   85
lx  100  192   73                         fx  140  192  103
lx  160  192   98
lx  200  192  127

 n: this option runs xdl2ncd with -nodrc option
 a: runs 'n' option followed by reentrant routing with
  PAR and converts back to XDL
 p: this option uses system-level pins instead of
  Boundary Scan interface
 pn: system-level pins PLUS runs xdl2ncd
   with -nodrc option
```

Figure 3.5: V4RamBist.exe Command Line Format

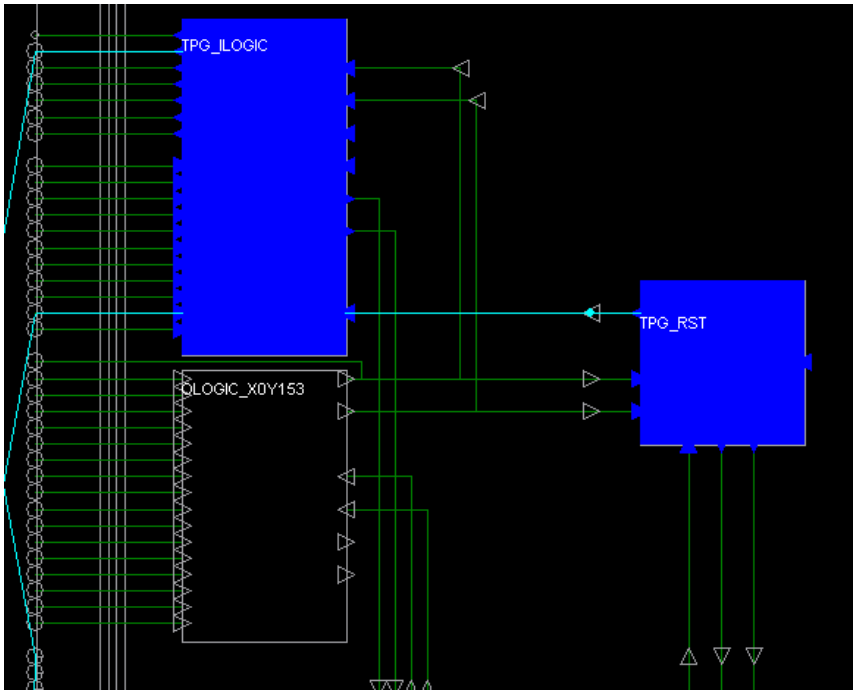


Figure 3.6: PINS - TPG_RST and ILOGIC Placement

Table 3.3: BRAM BSCAN Interface [7]

March Test	WE	REGCE	EN/SSR	MODE 2	MODE 1	MODE 0
MarchLR (Init A)	1	1	1	0	0	0
MarchLR (Init B)	1	1	1	0	0	0
March s2pf	1	1	1	0	1	1
March d2pf	1	1	1	1	0	0
MATS+ (16k)	1	1	1	0	1	0
MATS+ (8k)	0	0	0	0	0	1
MATS+ (512)	0	0	0	1	0	1

Table 3.4: FIFO BSCAN Interface [7]

FIFO MODE	RST LEVEL	WREN LEVEL	RDEN LEVEL	MODE[1]	MODE[0]
2k x 9-bit	0	0	0	0	1
512 x 36-bit	1	1	1	1	1
1k x 18-bit	1	1	1	1	0
4k x 4-bit	1	1	1	0	0

Table 3.5: BRAM PINS Interface

March Test	MODE[3]	MODE[2]	MODE[1]	MODE[0]
MarchLR (Init A)	1	0	0	0
MarchLR (Init B)	1	0	0	0
March s2pf	1	0	1	1
March d2pf	1	1	0	0
MATS+ (16k)	1	0	1	0
MATS+ (8k)	0	0	0	1
MATS+ (512)	0	1	0	1

Table 3.6: FIFO PINS Interface

FIFO Mode	MODE[2]	MODE[1]	MODE[0]
2k x 9-bit	0	0	1
512 x 36-bit	1	1	1
1k x 18-bit	1	1	0
4k x 4-bit	1	0	0

3.4 FIFO Reset Problem

A problem was found within the FIFO TPG during the PINS interface development that was not apparent initially. Milton gives the number of clock cycles that each FIFO BIST configuration requires in [7] and when executed for the specified values without being reset, the BIST performed as expected. However, during the development of the PINS interface, one of the desired options was to have the BIST run by an external free running clock. This would constantly run the BIST.

Experiments were performed to observe the behavior of the BIST when transitions occurred on the *TPG_RST* pin during BIST execution. The expected result was that the BIST would start over at the beginning of the test algorithm. However, it was observed that the BIST would become permanently stuck in a certain part of the algorithm when this behavior was simulated. The BIST configuration would then have to be re-downloaded in order for the BIST to perform its function.

The error was due to the *"element"* variable in Milton's FIFO BIST Finite State Machine (FSM) VHDL not being reset to its initial state when a *TPG_RST* was performed. Once corrected, the FIFO FSM performed correctly when reset.

Although the FIFO FSM was performing correctly, false fault detections were observed in the ORAs when the FIFO BIST was executed and transitions were simulated on the *TPG_RST* pin. The locations of the false fault detections were inconsistent, but the false fault detections always appeared in ORAs that monitor either the *ALMOSTFULL* or *FULL* flags.

Initially, the source of the problem was believed to be incorrect *ALMOSTFULL/EMPTY* offset values. Table 3.7 shows the acceptable range of values of the offsets for these flags. Tables 3.8(a) and 3.8(b) show the values Milton used in his FIFO BIST modification program. The offset values are shown both in decimal, so that the value can be verified to be in the desired range easily, and in hexadecimal to show the value specified by the original modification program.

From this table, it can clearly be seen that some of the offset values are incorrect. During examination of Milton's BIST modification program, it seems that he believed the *ALMOSTEMPTY* flag was designated normally by a hex value in the XDL. However, he seemed to believe that the *ALMOSTFULL* flag was designated by the two's complement of the offset value in the XDL. For example for the 2k x 9-bit FIFO mode, a valid *ALMOSTEMPTY* offset value would be any number in the range [5 : 2044] ([0x005 : 0x7FC]), while a valid *ALMOSTFULL* offset value could be the two's complement of any number in the range [5 : 2043] ([0xFFB : 0x805] instead of [0x005 : 0x7FB]).

Most peculiar was that Milton specified *2043*, *496*, *507* and *4* as the *ALMOSTFULL* offset for the first three and very last configurations of the FIFO BIST (See Tables 3.8(a) and 3.8(b)). If the offset does follow the two's complement of the input

offset value then the resulting offset values greatly exceed the acceptable values for that mode. However, if they do not then the majority of the 4k x 4-bit configurations test only a small range of offset values.

Table 3.7: FIFO *ALMOSTFULL/EMPTY* Flag Offset Range [15]

	ALMOSTEMPTY		ALMOSTFULL
FIFO Mode	Standard	FWFT	
2k x 9-bit	5 to 2044	6 to 2045	4 to 2043
512 x 32-bit	5 to 508	6 to 509	4 to 507
1k x 18-bit	5 to 1020	6 to 1021	4 to 1019
4k x 4-bit	5 to 4092	6 to 4093	4 to 4091

Under the assumption that the offset values of the *ALMOSTFULL* flag are the two's complement of the designated value, *V4RamMod.exe*, the BIST modification program, was modified to have offset values that would lie within the acceptable range for the desired FIFO mode. Also the offset values for the first three configurations were chosen to test the maximum range. These new values can be seen in Tables 3.9(a) and 3.9(b). However, this was not the solution to the randomly seen false fault detections during an asynchronous reset using the *TPG_RST* pin.

The next suspicion explored was that the false fault detections were caused by an initialization problem. The *Virtex-4 User Guide* states,

> "Reset is an **asynchronous** signal to reset all read and write address counters, and must be asserted to initialize flags after power up. Reset does not clear the memory, nor does it clear the output register. When reset is asserted High, EMPTY and ALMOST_EMPTY will be set to 1, FULL and ALMOST_FULL will be reset to 0. The reset signal must be High for at least three read clock and write clock cycles to ensure all

Table 3.8: *ALMOSTFULL/EMPTY* Flag Offset Range described in [7]

(a) *ALMOSTEMPTY* Offset Values

Config #	FIFO Mode	Normal		2's Compliment	
		Decimal	Hex	Decimal	Hex
1	2k x 9-bit	15	00F	4081	FF1
2	512 x 32-bit	15	00F	4081	FF1
3	1k x18-bit	5	005	4091	FFB
4	4k x 4-bit	5	005	4091	FFB
5	4k x 4-bit	6	006	4089	FF9
6	4k x 4-bit	8	008	4088	FF8
7	4k x 4-bit	16	010	4080	FF0
8	4k x 4-bit	32	020	4064	FE0
9	4k x 4-bit	64	040	4032	FC0
10	4k x 4-bit	128	080	3968	F80
11	4k x 4-bit	256	100	3840	F00
12	4k x 4-bit	512	200	3584	E00
13	4k x 4-bit	1024	400	3072	C00
14	4k x 4-bit	2048	800	2048	800
15	4k x 4-bit	4092	FFC	4	004

(b) *ALMOSTFULL* Offset Values

Config #	FIFO Mode	Normal		2's Compliment	
		Decimal	Hex	Decimal	Hex
1	2k x 9-bit	2043	7FB	2053	805
2	512 x 32-bit	496	1F0	3599	E0F
3	1k x18-bit	507	1FB	3589	E05
4	4k x 4-bit	4091	FFB	5	005
5	4k x 4-bit	4089	FF9	6	006
6	4k x 4-bit	4088	FF8	8	008
7	4k x 4-bit	4080	FF0	16	010
8	4k x 4-bit	4064	FE0	32	020
9	4k x 4-bit	4032	FC0	64	040
10	4k x 4-bit	3968	F80	128	080
11	4k x 4-bit	3840	F00	256	100
12	4k x 4-bit	3584	E00	512	200
13	4k x 4-bit	3072	C00	1024	400
14	4k x 4-bit	2048	800	2048	800
15	4k x 4-bit	4	004	4092	FFC

Table 3.9: *V4RamMod.exe ALMOSTFULL/EMPTY* Flag Offset Range

(a) *ALMOSTEMPTY* Offset Values

Config #	FIFO Mode	Normal		2's Compliment	
		Decimal	Hex	Decimal	Hex
1	2k x 9-bit	6	006	4089	FF9
2	512 x 32-bit	5	005	4091	FFB
3	1k x18-bit	5	005	4091	FFB
4	4k x 4-bit	5	005	4091	FFB
5	4k x 4-bit	6	006	4089	FF9
6	4k x 4-bit	8	008	4088	FF8
7	4k x 4-bit	16	010	4080	FF0
8	4k x 4-bit	32	020	4064	FE0
9	4k x 4-bit	64	040	4032	FC0
10	4k x 4-bit	128	080	3968	F80
11	4k x 4-bit	256	100	3840	F00
12	4k x 4-bit	512	200	3584	E00
13	4k x 4-bit	1024	400	3072	C00
14	4k x 4-bit	2048	800	2048	800
15	4k x 4-bit	4092	FFC	4	004
4 - alt	4k x 4-bit	1365	555	2731	AAB
5 - alt	4k x 4-bit	2730	AAA	1366	556

(b) *ALMOSTFULL* Offset Values

Config #	FIFO Mode	Normal		2's Compliment	
		Decimal	Hex	Decimal	Hex
1	2k x 9-bit	2053	805	2053	805
2	512 x 32-bit	3589	E05	507	1FB
3	1k x18-bit	3077	C05	1019	3FB
4	4k x 4-bit	4091	FFB	5	005
5	4k x 4-bit	4089	FF9	6	006
6	4k x 4-bit	4088	FF8	8	008
7	4k x 4-bit	4080	FF0	16	010
8	4k x 4-bit	4064	FE0	32	020
9	4k x 4-bit	4032	FC0	64	040
10	4k x 4-bit	3968	F80	128	080
11	4k x 4-bit	3840	F00	256	100
12	4k x 4-bit	3584	E00	512	200
13	4k x 4-bit	3072	C00	1024	400
14	4k x 4-bit	2048	800	2048	800
15	4k x 4-bit	5	005	4091	FFB
4 - alt	4k x 4-bit	2730	AAA	1366	556
5 - alt	4k x 4-bit	1365	555	2731	AAB

internal states are reset to the correct values. During RESET, RDEN and WREN must be held Low" [15].

In the FIFO TPG, the reset signal was synchronous to set up signals for the FIFO FSM. An additional statement in the VHDL description of the FIFO TPG was added that sent the reset signal asynchronously to the FIFOs, while the rest of the reset of the TPG operation was performed on the next rising clock edge. This solved the initialization problem and the false fault detections no longer appeared.

3.5 Additional FIFO Configurations

A major drawback of the FIFO BIST developed by Milton [7] is the sheer number of configurations that it requires to thoroughly test the *ALMOSTFULL/EMPTY* flags. Two alternative BIST configurations were developed to reduce the number of FIFO BIST configurations. Fault coverage is potentially reduced however, because fewer possible *ALMOSTFULL/EMPTY* flag bit combinations are tested. Fault simulation indicated the twelve 4k x 4 configurations developed by Milton provided higher fault coverage only if one knows the construction of the *ALMOSTFULL/EMPTY* logic and Milton's patterns were appropriately converted. Since this information is unknown, the additional test time is not worthwhile. These two new BIST configurations are shown in Table 3.10.

Table 3.10: Additional FIFO BIST Configurations

FIFO	Active Level			Write	ALMOST	ALMOST
Mode	RST	RDEN	WREN	Mode	FULL	EMPTY
4k x 4	1	1	1	Standard	1366 (AAA)	2731 (555)
4k x 4	1	1	1	Standard	2731 (555)	1366 (AAA)

The new set of BIST configurations constructed for FIFO BIST are shown in Table 3.11. Fault injection was performed for both the original and new sets of FIFO BIST configurations. A known list of FIFO RAM faults was used to modify the RAM configuration bits downloaded to the FPGA and each of the BIST configurations was executed. The faults that were detected by the FIFO BIST configurations were noted for each of the two sets. Both sets were found to detect the same faults, despite the potentially reduced fault coverage due to not testing as many combinations of the *ALMOSTFULL/EMPTY* flag configuration bits.

The total number of BIST clock cycles was 65,561 for all fifteen configurations in [7], while the total number of clock cycles is 64,672 for the configurations in Table 3.11. Despite the fact that this is only a difference of less than a thousand clock cycles, configuration download and memory readback time need to be taken into account since these are for more significant. Additionally, Milton assumes that the user has complete control of the BIST clock. However, this may not be the case when using the PINS option if, for example, the BIST is clocked using an external oscillator. The two new configurations were designed such that meticulous clock control is not needed.

If more thorough testing of the *ALMOSTFULL/EMPTY* flags is desired, then it is preferable to use the set of fifteen configurations. If merely the functionality of the flags is desired, then it is preferable to use the new set of five configurations to reduce overall test time.

Table 3.11: Shortened FIFO BIST Configurations

FIFO Mode	Active Level RST	RDEN	WREN	Write Mode	ALMOST FULL	ALMOST EMPTY
2k x 9	0	0	0	FWFT	15	2043
512 x 36	1	1	1	Standard	15	496
1k x 18	1	1	1	Standard	5	507
4k x 4	1	1	1	Standard	1366 (AAA)	2731 (555)
4k x 4	1	1	1	Standard	2731 (555)	1366 (AAA)

3.6 Cascade ORA Clock Enables

When in the Cascade mode, there are expected failures in the ORAs along the bottom row of the device and in the row above a PowerPC in the FX devices (re-illustrated in Figure 3.7). This is due to the lack of cascade routing, namely a *CascadeIn* (re-illustrated in Figure 3.8) input on these RAMs. These expected failures interfere with the iterative OR-chain mentioned earlier by latching up the error in the ORAs resulting in a constant *Fail* signal from the OR-chain output. A solution was developed to handle the ORA mismatches that would otherwise occur. This required adding a clock enable to the ORAs that monitor these particular RAMs. The TPG then controls the clock enable such that the ORAs do not latch the expected failure, but will latch mismatches due to faults.

Milton's approach required only 20 clock cycles using a FSM to execute the BIST. When performing asynchronous resets using the PINS option for the cascade BIST, the behavior of the BIST was erratic and the expected failures would become latched into the ORAs, which was undesirable. The VHDL description of the cascade mode TPG was modified to be a 5-bit counter that would output the desired address, read/write operation, data, Port A/B enable, and additional ORA clock enables, *ORACE* and *ORBCE* – the clock enables (CE) added to enable or disable the ORA

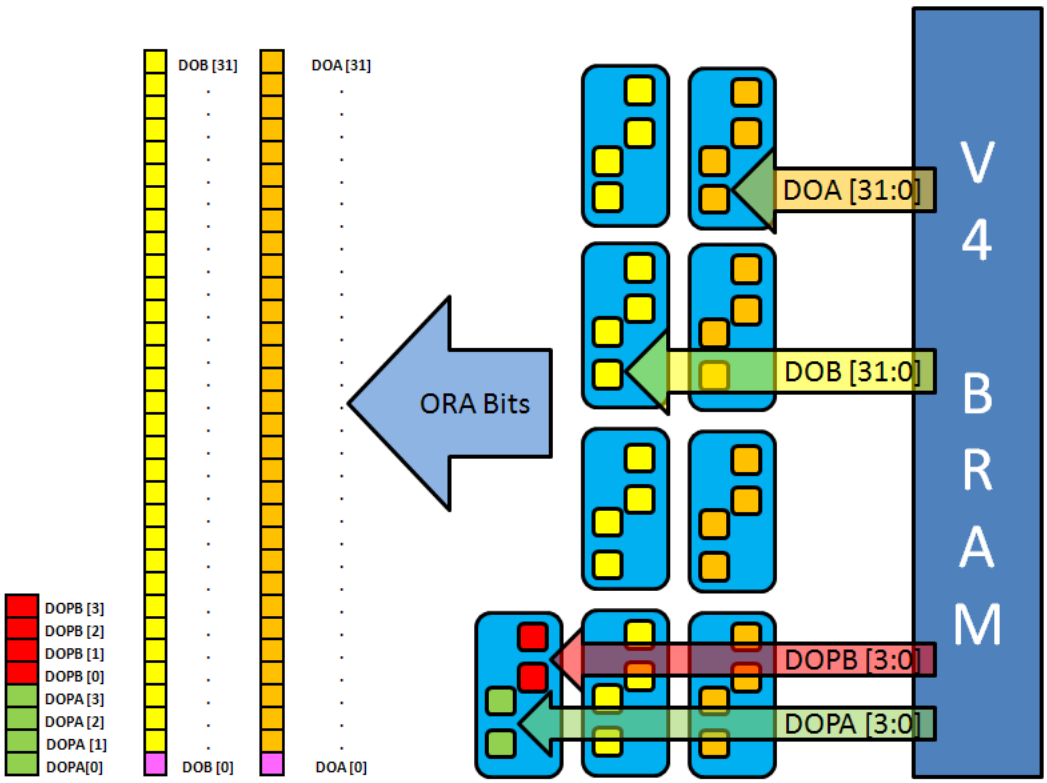


Figure 3.7: Cascade ORA Bit Assignments [7]

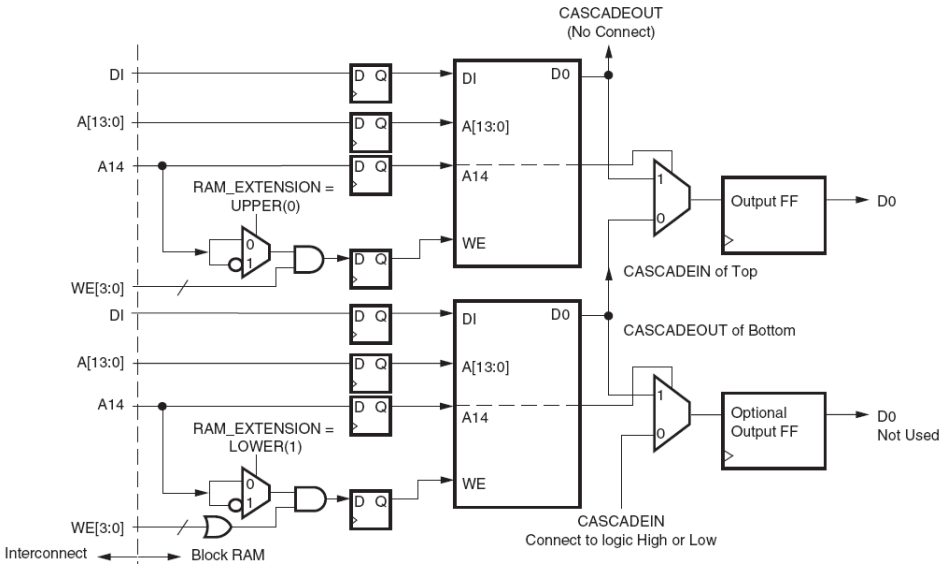


Figure 3.8: Cascadable Block RAM [15]

for a desired number of clock cycles. This was a feasible option because the cascade BIST sequence is only 32 clock cycles.

The *V4RamBist.exe* program added routing from the Cascade TPG to the ORAs with expected failures. The BIST was run normally, that is the *ORACE* and *ORBCE* had no impact on the BIST, on a Virtex-4 FX12 device. The ORAs were enabled for the entire execution. The clock cycles where ORA mismatches occurred were noted and can be seen in Tables 3.12 - 3.20 as those rows colored in red. The specific clock cycle can be seen from the *CNTR* columns, while the inputs to the ORAs, which expect failures, can be seen in the *RAM* and *ORA* columns respectively.

It can be clearly seen that for the counter values *7 (00111), 14 (01110)*, and *15 (01111)* mismatches occur on Port A between the RAMs along the bottom of the device when the first cascade configuration (*UPPER*) is executed. When configured in the second cascade configuration (*LOWER*) the counter value *3 (00011)* also has mismatches on Port A in addition to the counter values described for the *UPPER* configuration. Counter values *21 (10101), 22 (10110), 23 (10111), 30 (11110)*, and *31 (11111)* have mismatches for Port B for both *UPPER* and *LOWER* configurations.

The TPG VHDL description was modified to disable the specific ORAs for either Port A or B when there are mismatches. These changes were able to ensure that the expected ORA mismatches did not get latched, even during the reset experiment using the PINS option.

3.7 Timing Improvements and Analysis

Several timing improvements were also made to the Virtex-4 Block RAM BIST configurations. Figure 3.9 shows the maximum BIST clock frequencies at which the original Virtex-4 BIST configurations, described in Table 3.21, can be executed

Table 3.12: Cascade Mismatches - Port A - *UPPER* Configuration (1)

CNTR	RAM R0C5	RAM R56C5	ORA R0C4	RAM R0C10	RAM R56C10	ORA R0C9
1	0	0	1	0	0	1
2	0	0	1	0	0	1
3	0	0	1	0	0	1
4	0	0	1	0	0	1
5	0	0	1	0	0	1
6	0	0	1	0	0	1
7	**0**	**0**	**1**	**0**	**0**	**1**
8	1	1	1	1	1	1
9	1	1	1	1	1	1
10	1	1	1	1	1	1
11	1	1	1	1	1	1
12	1	1	1	1	1	1
13	1	1	1	1	1	1
14	**0**	**0**	**1**	**0**	**0**	**1**
15	**0**	**0**	**1**	**0**	**0**	**1**
16	0	0	1	0	0	1

Table 3.13: Cascade Mismatches - Port A - *UPPER* Configuration (2)

CNTR	RAM R0C25	RAM R56C25	ORA R0C26	RAM R4C5	RAM R36C5	ORA R36C4
1	0	0	1	0	0	1
2	0	0	1	0	0	1
3	0	0	1	0	0	1
4	0	0	1	0	0	1
5	0	0	1	0	0	1
6	0	0	1	0	0	1
7	**0**	**0**	**1**	**1**	**0**	**1**
8	1	1	1	0	0	1
9	1	1	1	0	0	1
10	1	1	1	1	1	1
11	1	1	1	1	1	1
12	1	1	1	1	1	1
13	1	1	1	0	0	1
14	**0**	**0**	**1**	**1**	**0**	**1**
15	**0**	**0**	**1**	**1**	**0**	**1**
16	0	0	1	0	0	1

Table 3.14: Cascade Mismatches - Port A - *UPPER* Configuration (3)

CNTR	RAM R4C10	RAM R36C10	ORA R36C9
1	0	0	1
2	0	0	1
3	0	0	1
4	0	0	1
5	0	0	1
6	0	0	1
7	**1**	**0**	**1**
8	0	0	1
9	0	0	1
10	1	1	1
11	1	1	1
12	1	1	1
13	0	0	1
14	**1**	**0**	**1**
15	**1**	**0**	**1**
16	0	0	1

Table 3.15: Cascade Mismatches - Port A - *LOWER* Configuration (1)

CNTR	RAM R0C5	RAM R56C5	ORA R0C4	RAM R0C10	RAM R56C10	ORA R0C9
1	1	1	1	1	1	1
2	1	1	1	1	1	1
3	**0**	**1**	**1**	**0**	**1**	**1**
4	1	1	1	1	1	1
5	0	0	1	0	0	1
6	0	0	1	0	0	1
7	**0**	**1**	**1**	**0**	**1**	**1**
8	0	0	1	0	0	1
9	0	0	1	0	0	1
10	1	1	1	1	1	1
11	1	1	1	1	1	1
12	1	1	1	1	1	1
13	0	0	1	0	0	1
14	**0**	**1**	**1**	**0**	**1**	**1**
15	**0**	**1**	**1**	**0**	**1**	**1**
16	0	0	1	0	0	1

Table 3.16: Cascade Mismatches - Port A - *LOWER* Configuration (2)

CNTR	RAM R0C25	RAM R56C25	ORA R0C26	RAM R4C5	RAM R36C5	ORA R36C4
1	1	1	1	1	1	1
2	1	1	1	1	1	1
3	**0**	**1**	**1**	**1**	**0**	**1**
4	1	1	1	0	0	1
5	0	0	1	0	0	1
6	0	0	1	0	0	1
7	**0**	**1**	**1**	**0**	**0**	**1**
8	0	0	1	1	1	1
9	0	0	1	1	1	1
10	1	1	1	1	1	1
11	1	1	1	1	1	1
12	1	1	1	1	1	1
13	0	0	1	1	1	1
14	**0**	**1**	**1**	**0**	**0**	**1**
15	**0**	**1**	**1**	**0**	**0**	**1**
16	0	0	1	0	0	1

Table 3.17: Cascade Mismatches - Port A - *LOWER* Configuration (3)

CNTR	RAM R4C10	RAM R36C10	ORA R36C9
1	1	1	1
2	1	1	1
3	**1**	**0**	**1**
4	0	0	1
5	0	0	1
6	0	0	1
7	**0**	**0**	**1**
8	1	1	1
9	1	1	1
10	1	1	1
11	1	1	1
12	1	1	1
13	1	1	1
14	**0**	**0**	**1**
15	**0**	**0**	**1**
16	0	0	1

Table 3.18: Cascade Mismatches - Port B - *UPPER/LOWER* Configurations (1)

CNTR	RAM R0C5	RAM R56C5	ORA R0C3	RAM R0C10	RAM R56C10	ORA R0C8
17	0	0	1	0	0	1
18	0	0	1	0	0	1
19	**0**	**0**	**1**	**0**	**0**	**1**
20	1	1	1	1	1	1
21	0	1	1	0	1	1
22	0	1	1	0	1	1
23	**0**	**1**	**1**	**0**	**1**	**1**
24	0	0	1	0	0	1
25	0	0	1	0	0	1
26	0	0	1	0	0	1
27	0	0	1	0	0	1
28	0	0	1	0	0	1
29	0	0	1	0	0	1
30	**0**	**1**	**1**	**0**	**1**	**1**
31	**0**	**1**	**1**	**0**	**1**	**1**
32	0	1	1	0	1	1

Table 3.19: Cascade Mismatches - Port B - *UPPER/LOWER* Configurations (2)

CNTR	RAM R0C25	RAM R56C25	ORA R0C27	RAM R4C5	RAM R36C5	ORA R36C3
17	0	0	1	0	0	1
18	0	0	1	0	0	1
19	**0**	**0**	**1**	**0**	**0**	**1**
20	1	1	1	0	0	1
21	0	1	1	1	0	1
22	0	1	1	1	0	1
23	**0**	**1**	**1**	**0**	**0**	**1**
24	0	0	1	1	1	1
25	0	0	1	1	1	1
26	0	0	1	0	0	1
27	0	0	1	0	0	1
28	0	0	1	0	0	1
29	0	0	1	1	1	1
30	**0**	**1**	**1**	**0**	**0**	**1**
31	**0**	**1**	**1**	**0**	**0**	**1**
32	0	1	1	1	0	1

Table 3.20: Cascade Mismatches - Port B - *UPPER/LOWER* Configurations (3)

CNTR	RAM R4C10	RAM R36C10	ORA R36C8
17	0	0	1
18	0	0	1
19	**0**	**0**	**1**
20	0	0	1
21	1	0	1
22	1	0	1
23	**0**	**0**	**1**
24	1	1	1
25	1	1	1
26	0	0	1
27	0	0	1
28	0	0	1
29	1	1	1
30	**0**	**0**	**1**
31	**0**	**0**	**1**
32	1	0	1

on a Virtex-4 LX60 device. It can be seen that the worst-case clock frequencies are configurations six, seven, nine, and eleven for the four BIST types, which correspond to the frequencies 47.9 MHz, 101.4 MHz, 82.4 MHz, and 67.4 MHz, respectively, for a Virtex-4 LX60 device. Configurations six and eleven are special cases where the Block RAMs use opposite edge clocking from that of the TPGs and ORAs. This was previously shown in Table 2.6 for Block RAM BIST and while not explicitly stated in Milton's thesis, examination of his work shows this is the case for the FIFO BIST (2K) configuration. As a result, the normal BIST clock frequency is halved. The timing analysis for every Virtex-4 device worst-case BIST configuration can be seen in Figure 3.10. The number above each set of bars indicates the total number of Block RAMs within that device and all Block RAMs are under test in these BIST configurations.

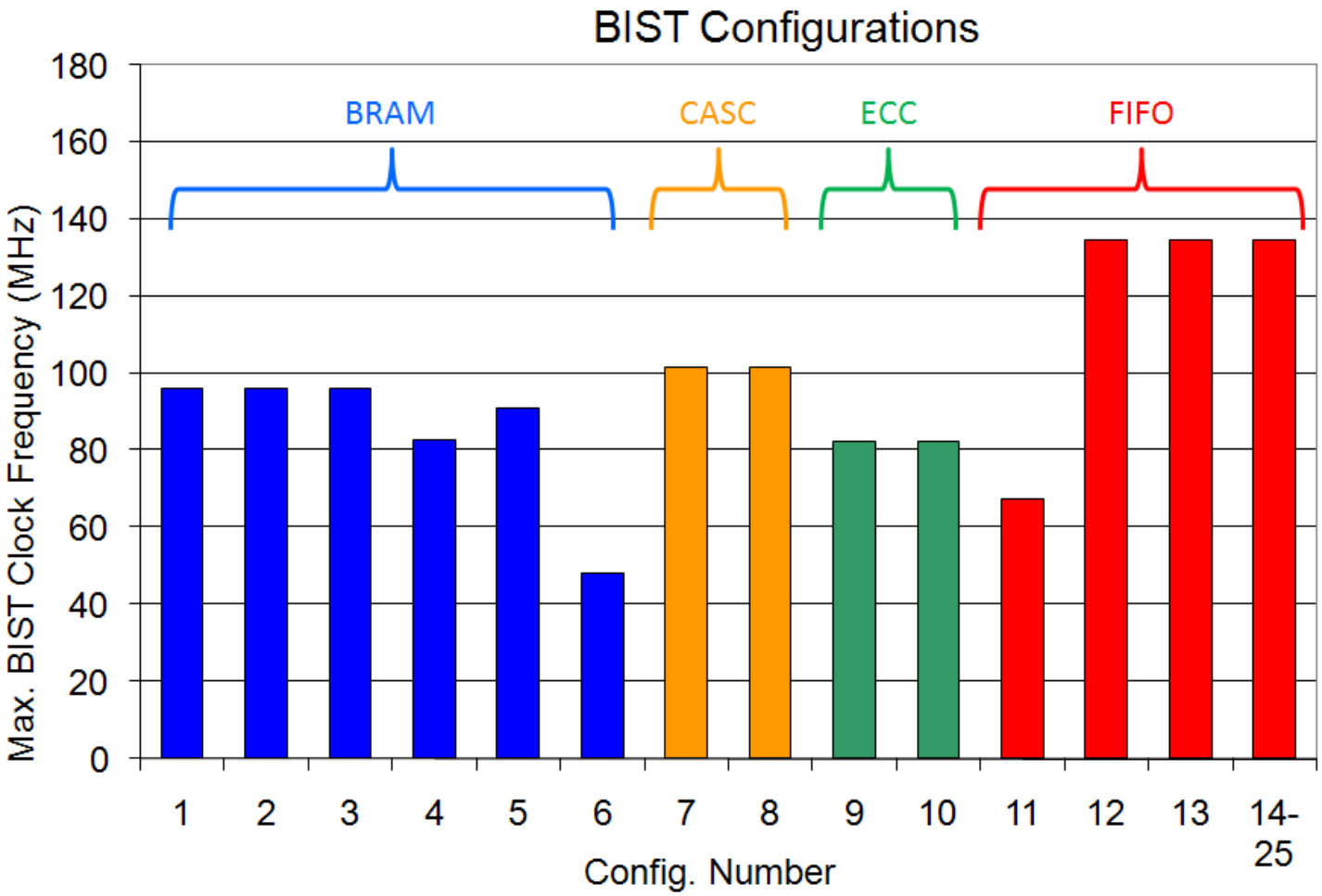


Figure 3.9: Max. BIST Clock Frequency for Virtex-4 LX60

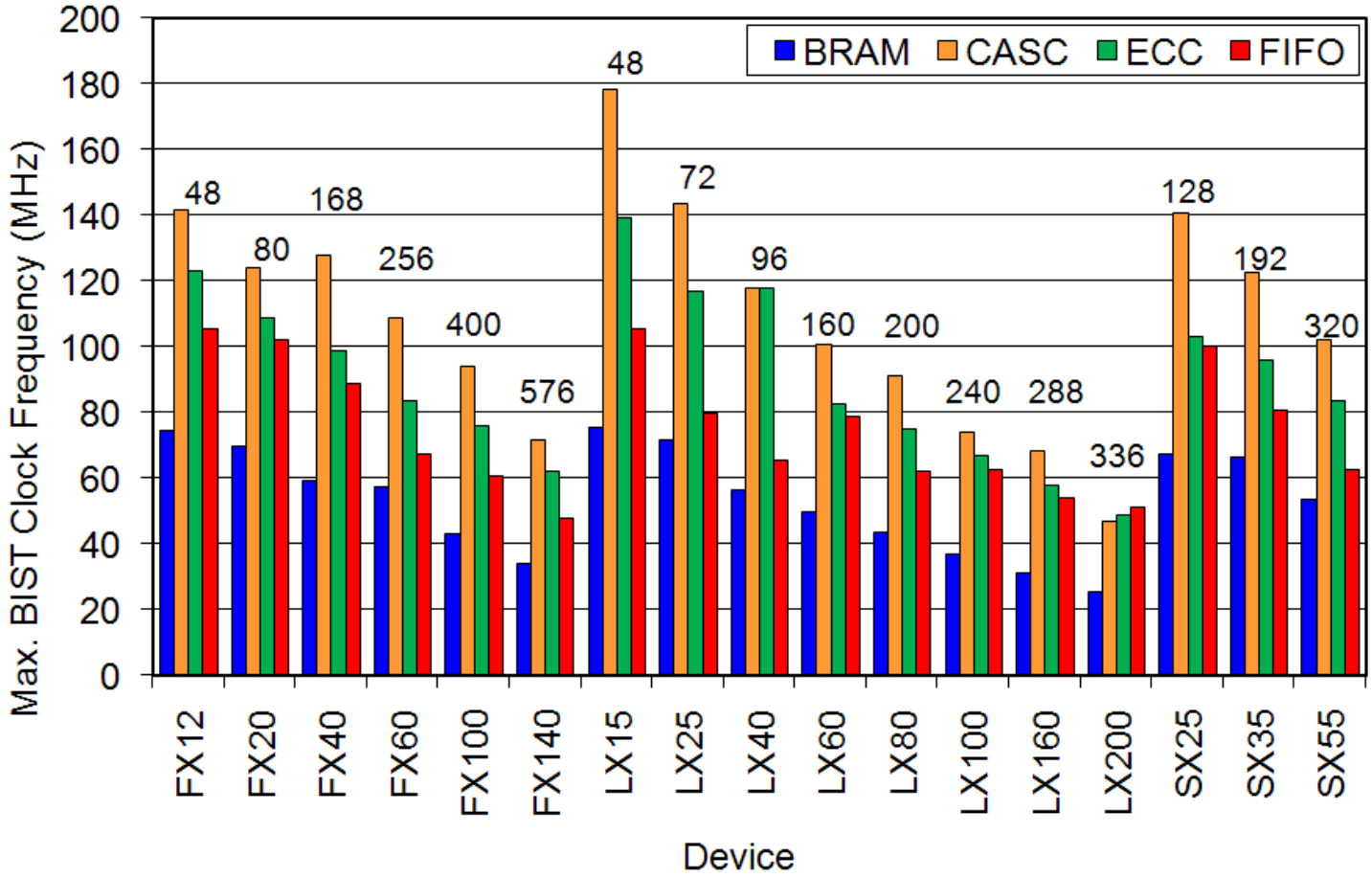


Figure 3.10: Timing Analysis for Worst-Case BIST Configurations - Virtex-4 Devices

Table 3.21: Max. BIST Clock Frequency for each Configuration in LX60 Device

Config. #	BIST	Max. Freq.
1	BRAM - MLRA w/ BDS	95.997
2	BRAM - MLRB w/ BDS	95.997
3	BRAM - DUALP	95.997
4	BRAM - MATS+ 16k	82.556
5	BRAM - MATS+ 8k	91.017
6	BRAM - MATS+ 512	47.998
7	CASC - UPPER	101.43
8	CASC - LOWER	101.43
9	ECC - WREN	82.379
10	ECC - RDEN	82.379
11	FIFO - 2k x 9	67.367
12	FIFO - 512 x 36	134.735
13	FIFO - 1k x 18	134.735
14-25	FIFO - 4k x 4 (all)	134.735

This halved clock frequency greatly limits the speed at which the BIST can be executed. However, a solution was implemented within the Virtex-4 modification program that eliminates this halving effect. In addition to the RAM Port A/B clocks for configuration six and the Write/Read clocks for configuration eleven being inverted, the clocks of all the other components of the BIST (ORAs and TPGs) are inverted, effectively making them all the same edge, while still testing the opposite edge clocking of the Block RAM. The new worst-case clock frequencies for the Block RAM BIST now become governed by configuration four and nine (Figure 3.11). The timing analysis for all Virtex-4 device new worst-case configuration can be seen in Figure 3.12. This change resulted in a dramatic increase in the maximum BIST clock frequency for all devices. For example, the LX200 device previous had a maximum BIST clock frequency of approximately 25 MHz. However with this change implemented, the maximum BIST clock frequency increased to approximately 42 MHz.

A penalty is incurred from this change. The size of the partial configuration bit file with the inverted clocks increases. This can be seen in Tables 3.23 and 3.24. However, by making the configuration with the inverted clocks either the first or last configuration in a set of BIST configurations, the impact of this increased file size is reduced. All of the clocks in the device have to be reconfigured only a single time, rather than twice if the configuration is in the middle of the set of BIST configurations. The bolded rows show the files that have an increase in size due to this new clocking scheme.

When using all twenty-five configuration bit files, the file size increase is found to be 0.000518% using the compressed configuration bit files and 1.029% when using partial compressed configuration bit files. When using the set of fifteen configuration bit files, the file size increase was 0.00849% using the compressed configuation bit files and 1.940% when using partial compressed configuration bit files. This file size increase has very little impact on the total download time of all BIST configuration files.

3.8 Fault Coverage

In [7], the only results discussed by Milton are the differences in test time and file size when downloading full, compressed, and partial configuration bit files to a device. However, the actual fault coverage of the Virtex-4 Block RAM BIST was never really discussed. After implementing the corrections and improvements discussed in this chapter, a quantitative analysis of the achievable fault coverage was performed.

A total of 456 configuration memory bit faults were emulated within a device, specifically a Virtex-4 FX12 FPGA, by overwriting a single bit in the configuration memory to a desired stuck-at value after each BIST configuration was downloaded

Table 3.22: Virtex-4 Block RAM BIST Configurations with Clock Modifications for LX60 Device

Config. #	BIST	Max. Freq.
1	BRAM - MLRA w/ BDS	95.997
2	BRAM - MLRB w/ BDS	95.997
3	BRAM - DUALP	95.997
4	BRAM - MATS+ 16k	82.556
5	BRAM - MATS+ 8k	91.017
6	BRAM - MATS+ 512	96.488
7	CASC - UPPER	101.43
8	CASC - LOWER	101.43
9	ECC - WREN	82.379
10	ECC - RDEN	82.379
11	FIFO - 2k x 9	133.851
12	FIFO - 512 x 36	134.735
13	FIFO - 1k x 18	134.735
14-25	FIFO - 4k x 4 (all)	134.735

Table 3.23: File Size without Clock Inversion Change for LX60 Device

Config. #	BIST	Compressed (# bits)	Partial (# bits)
1	MLRA	9,566,976	9,566,976
2	MLRB	9,566,976	446,688
3	DUALP	9,566,976	446,688
4	MATS+ 16K	9,566,976	446,688
5	MATS+ 8K	9,566,976	545,248
6	**MATS+ 512**	**9,566,976**	**545,248**
7	CAS UPPER	7,962,624	7,962,624
8	CAS LOWER	7,962,624	446,688
9	ECC WREN	9,461,856	9,461,856
10	ECC RDEN	9,448,800	545,248
11	FIFO 2K	8,684,832	8,684,832
12	**FIFO 512**	**8,684,832**	**122,688**
13	FIFO 1K	8,684,832	24,032
14 -25	FIFO 4K x 1	8,684,832	24,032
	Total	222,510,240	39,533,888
14 - alt	FIFO 4K x 1	8,684,832	24,032
15 - alt	FIFO 4K x 1	8,684,832	24,032
	Total	135,661,920	39,293,568

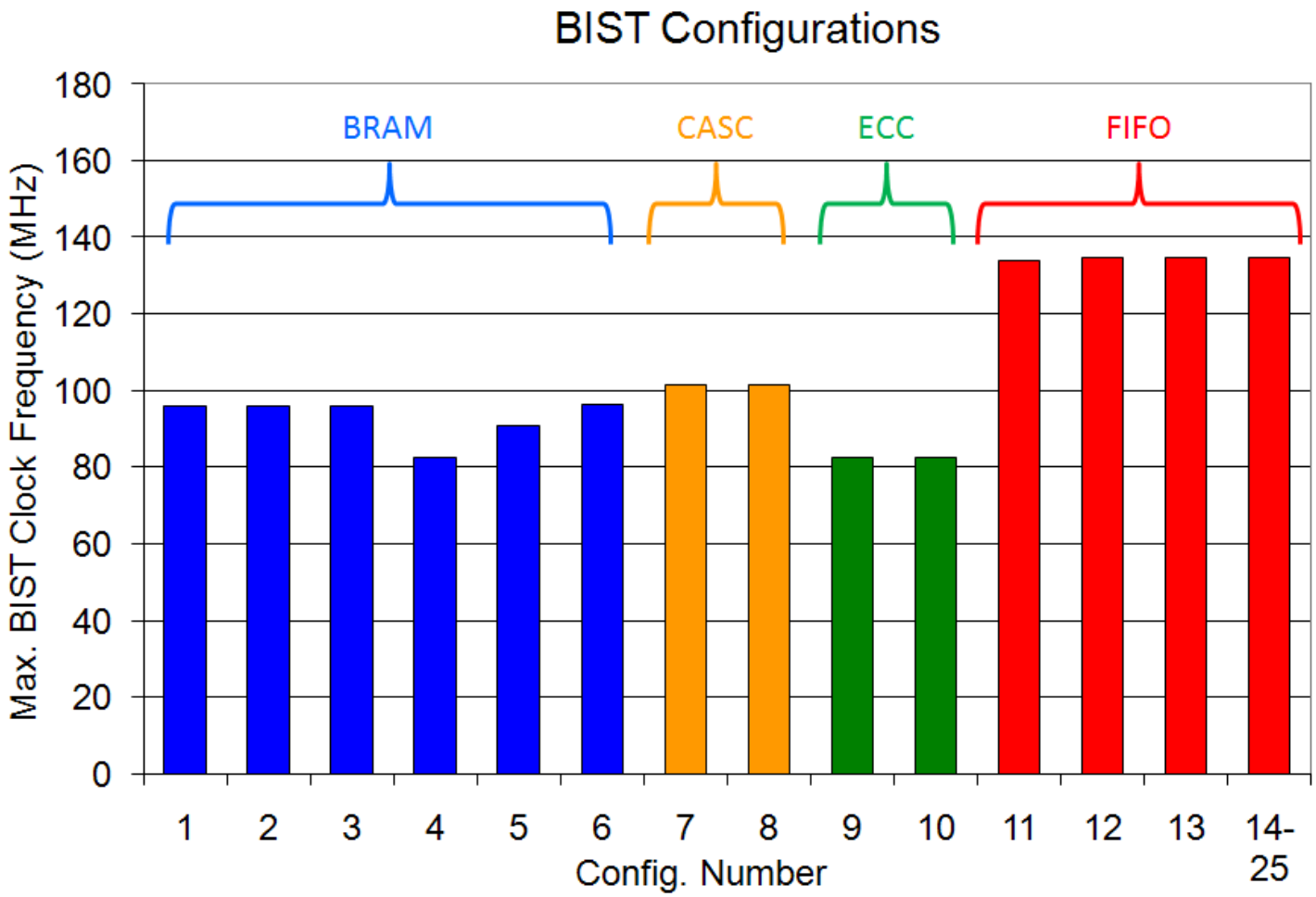


Figure 3.11: Max. BIST Clock Frequency for Virtex-4 LX60 with Clock Modifications

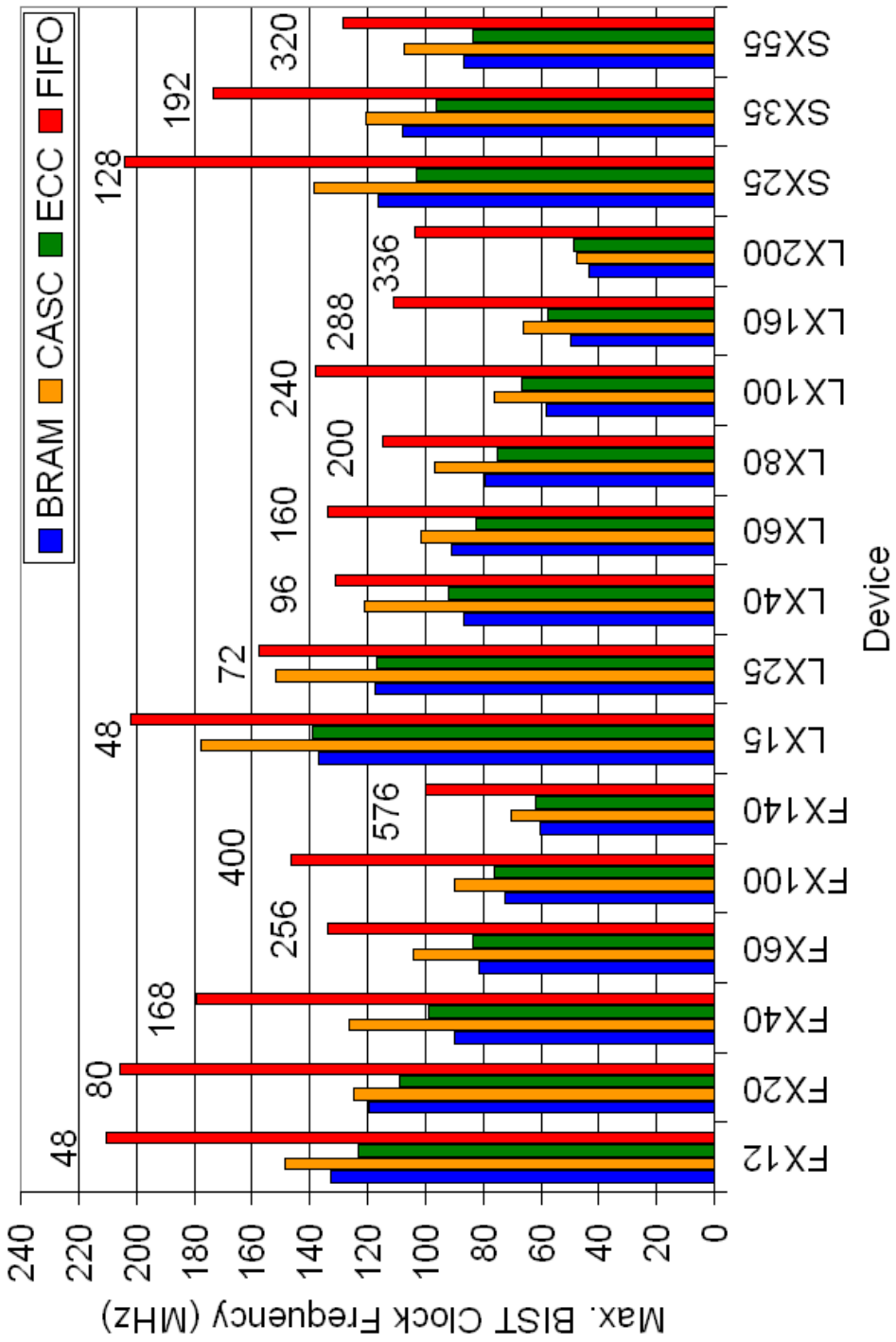


Figure 3.12: Timing Analysis for New Worst-Case BIST Configurations - Virtex-4 Devices

Table 3.24: File Size with Clock Inversion Change for LX60 Device

Config. #	BIST	Compressed (# bits)	Partial (# bits)
1	MLRA	9,566,976	9,566,976
2	MLRB	9,566,976	446,688
3	DUALP	9,566,976	446,688
4	MATS+ 16K	9,566,976	446,688
5	MATS+ 8K	9,566,976	545,248
6	**MATS+ 512**	**9,566,976**	**949,792**
7	CAS UPPER	7,962,624	7,962,624
8	CAS LOWER	7,962,624	446,688
9	ECC WREN	9,461,856	9,461,856
10	ECC RDEN	9,448,800	545,248
11	FIFO 2K	8,683,680	8,683,680
12	**FIFO 512**	**8,684,832**	**481,600**
13	FIFO 1K	8,684,832	24,032
14-25	FIFO 4K x 1	8,684,832	24,032
	Total	222,509,088	40,296,192
14 - alt	FIFO 4K x 1	8,684,832	24,032
15 - alt	FIFO 4K x 1	8,684,832	24,032
	Total	135,660,768	40,055,872

to the device but before the BIST configurations were executed. The complete set of BIST configurations was executed and the configurations that detected the emulated fault were noted. The fault coverage accrued by each BIST type can be seen in Figure 3.13, as well as the cumulative fault coverage for executing the complete set of BIST configurations, which results in a cumulative fault coverage of 98.67%. The fault coverage attained by each individual configuration for each of the BIST types can be seen in Figures 3.14, 3.15, and 3.16.

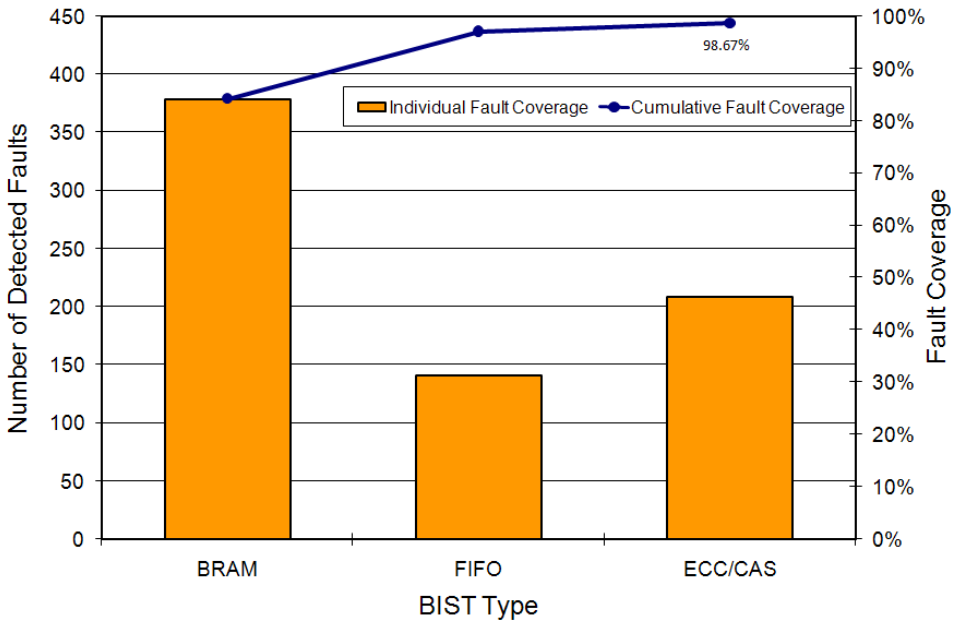


Figure 3.13: Overall Fault Coverage for Virtex-4 Devices

3.9 Summary

Overall, the process for generating the Virtex-4 Block RAM BIST configurations was improved. Twelve specific BIST generation programs were condensed into two visible and one hidden program that maintained the same functionality. The ORAs

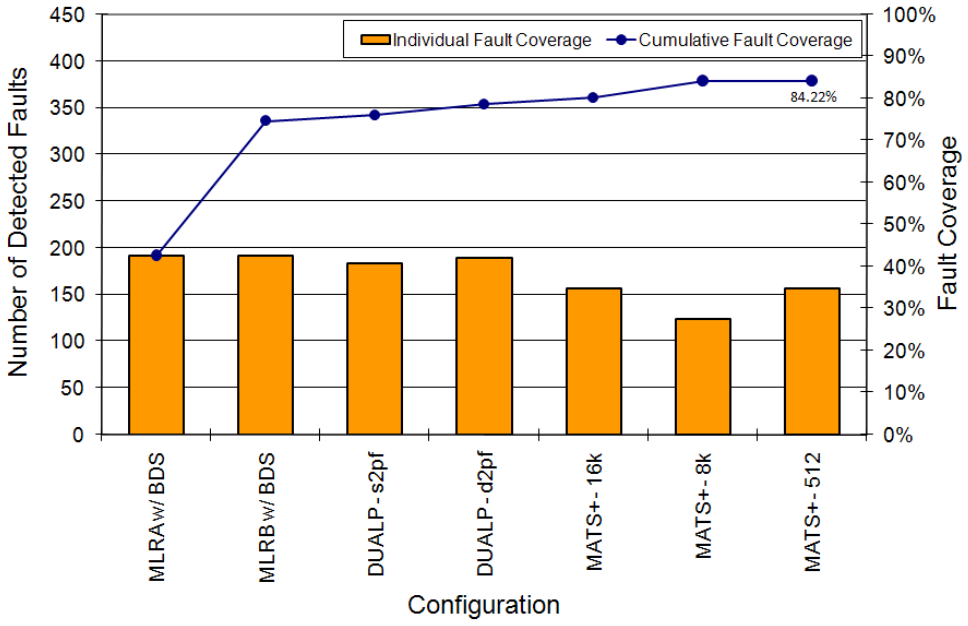


Figure 3.14: BRAM Fault Coverage for Virtex-4 Devices

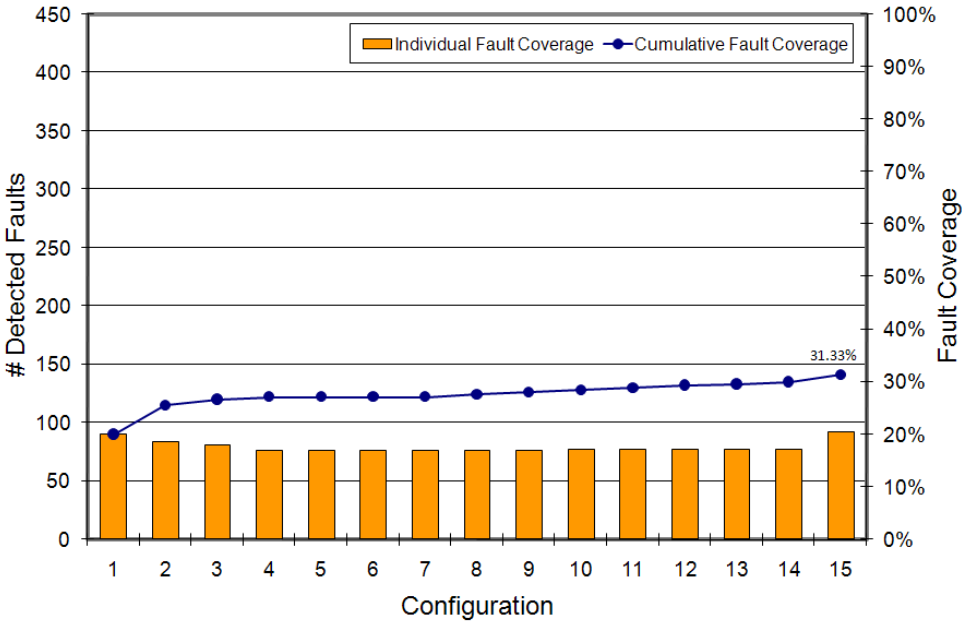


Figure 3.15: FIFO Fault Coverage for Virtex-4 Devices

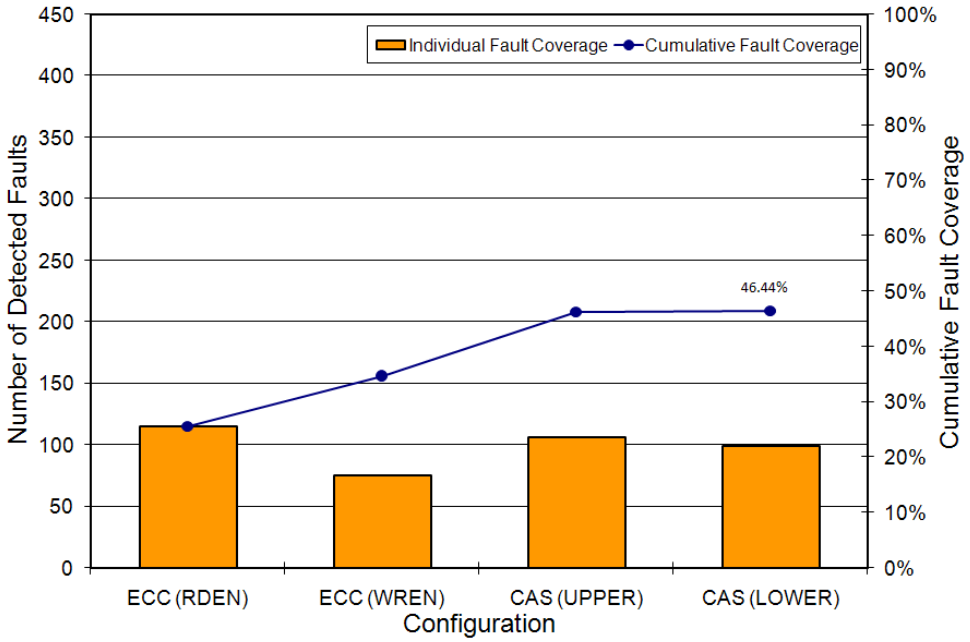


Figure 3.16: ECC and Cascade Fault Coverage for Virtex-4 Devices

were modified to exploit the built-in carry chain functionality of the CLBs to provide the user a single *Pass/Fail* signal using an iterative OR-chain. An additional interface for interacting with the BIST, the PINS interface, was developed to give the user another option if BSCAN is not available or desired.

Also, two new configurations were developed for FIFO Block RAM BIST to reduce overall test time by testing all of the ALMOSTFULL/EMPTY flag offset configuration bits simultaneously. Resetting the FIFO BIST no longer results in the BIST becoming stuck or false fault detections being latched up in the ORAs. The TPG for the Cascade Block RAM mode was modified to incorporate clock enable signals to enable/disable the Port A/B ORAs along the bottom of the device and in the row above a PowerPC, so as not to have false fault detections that could interfere with the *Pass/Fail* signal generated by the OR-chain. Changes were made to the modification program to increase the maximum BIST clock frequency so that

it could be executed faster, reducing overall BIST test time. And finally the fault coverage, both cumulative and individual, was found for the Virtex-4 Block RAM BIST configurations.

Chapter 4

Virtex-5 Block RAM BIST

This chapter will discuss the initial development of the Virtex-5 Block RAM BIST programs and configurations. The approaches applied to the Virtex-4 Block RAM BIST were adapted and applied to develop the Virtex-5 Block RAM BIST. Several TPGs were initially developed for the various Virtex-5 Block RAM types and new comparison-based ORAs were implemented to monitor the outputs of the Block RAMs. The new TPGs will be described first, followed by discussion of the new ORAs and ending with initial results for the Block RAM BIST configurations.

4.1 TPG Development

Initially, only one Block RAM XDL primitive (RAMBFIFO36) was to be used for generation of all the BIST configurations. This Block RAM primitive was found to be a superset of all of the other Virtex-5 Block RAM primitives. A single Block RAM primitive and TPG design would reduce the complexity of BIST generation. Since this primitive is a superset of all the Virtex-5 Block RAM primitives, all modes of operation could be tested while reducing download and testing time. However, this was found to be an overzealous starting point for this project as there was no documentation to utilize. Instead, four documented Block RAM primitives were chosen to implement the Virtex-5 BIST in order to establish a known behavior for each of these four types, which could later be combined to design a single TPG for

the RAMBFIFO36 Block RAM primitive. The implemented Block RAM primitives are as follows:

1. RAMB36 (32k + 4k parity) - true dual-port Block RAM that supports widths of x1, x2, x4, x9, x18, and x36.

2. RAMB36SDP (512 x 72-bit) - simple dual-port Block RAM with 64-bit ECC.

3. FIFO36 (32k + 4k parity) - synchronous/asynchronous FIFO Block RAM that supports widths of x1, x2, x4, x9, and x18.

4. FIFO36_72 (512 x 72-bit) - synchronous/asynchronous FIFO with 64-bit ECC.

Four TPG designs are proposed for the Virtex-5 Block RAM BIST that would perform specific test algorithms depending on the corresponding Block RAM configuration. The BIST configurations and their corresponding address and data widths can be seen in Table 4.1 and will be discussed in more depth in the following subsections.

4.1.1 RAMB36 TPG

The first TPG developed was for the Virtex-5 Block RAM BIST in the RAMB36 mode of operation. Comprehensive testing of the memory core is performed by the RAMB36SDP TPG, which will be described in the next subsection. The RAMB36 TPG is responsible for testing the dual-port functionality of the Block RAM using the RAM test algorithms, *S2PF* and *D2PF* (See Equations 2.7 and 2.8, respectively). Also, the MATS+ RAM test algorithm (See Equation 2.9) is used to test the programmable address and data widths, write modes, the active levels of the clock, port enable, output register clock, and set/reset signals. The settings for each BIST configuration can be seen in Table 4.3. The desired test is selected by a user-supplied control string which is shifted into the TPG through the BSCAN interface, as shown

Table 4.1: Virtex-5 BIST Configurations

TPG Type	BIST Config. #	Algorithm	Address Depth	Data Width
RAMB36	1	s2pf	1k	36
	2	d2pf	1k	36
	3	MATS+	2k	18
	4	MATS+	4k	9
	5	MATS+	8k	4
	6	MATS+	16k	2
	7	MATS+	32k	1
RAMB36SDP	1	MarchLRw/BDS	512	72
	2	ECC (RDEN)	512	72
	3	ECC (WREN)	512	72
FIFO36_72	1	FIFOX (RDEN)	512	72
	2	FIFOX (WREN)	512	72
FIFO36	1	FIFOX	1k	36
	2	FIFOX	2k	18
	3	FIFOX	4k	9
	4	FIFOX	8k	4

in Figure 4.1 and the control string values for each BIST configuration can be seen in Table 4.2. The three *Mode* bits correspond to the *Configuration Number* in Table 4.3, while the *Level Control* bit corresponds to the *ENA* level.

The RAM test algorithms described in Chapter 2 are implemented by a Finite State Machine (FSM) in a VHDL model for the RAMB36 TPG. The model was constrained the to first thirty-two rows and the first six columns in the bottom of an LX50T device, as shown in Figure 4.2, and is a total of 118 slices. All of the TPGs developed are held to this constraint, which allows for easier processing later by the Virtex-5 BIST template generation program, *V5BramBIST.exe*.

Table 4.2: Control String Values for RAMB36 TPG

Config. #	BIST Algorithm	Level Control	Mode 2	Mode 1	Mode 0	Hex String
1	s2pf	0	0	0	0	0x0
2	d2pf	0	0	0	1	0x1
3	MATS+ (2k)	1	0	1	0	0xA
4	MATS+ (4k)	1	0	1	1	0xB
5	MATS+ (8k)	1	1	0	0	0xC
6	MATS+ (16k)	1	1	0	1	0xD
7	MATS+ (32k)	1	1	1	0	0xE

Table 4.3: Configuration Settings for RAMB36 TPG

(a) Settings Part 1

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

(b) Settings Part 2

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

Figure 4.1: Shift Register Control String - RAMB36

4.1.2 RAMB36SDP TPG

The TPG proposed for the RAMB36SDP primitive is responsible for testing the memory core, as well as the ECC read and write capabilities. By using the widest data width for this Block RAM type, all memory elements can be reached by the implemented BIST configurations. A control string is shifted in through the BSCAN interface to the TPG to select the desired BIST configuration. The control string values for each BIST configuration can be seen in Table 4.4.

Table 4.4: Control String Values for RAMB36SDP TPG

Config. #	Algorithm	LEVEL CTRL	MODE 1	MODE 0	Hex String
1	MarchLR with BDS	0	0	0	0x0
2	ECC	0	0	1	0x1
3	ECC	1	1	0	0x6

The RAM test algorithm, *MarchLR with BDS*, is incorporated into this TPG and because the data width is set to its widest width of 72-bits, a Background Data Sequence is developed to ensure that all intra-word Coupling Faults can be detected. The implemented March test and corresponding BDS can be seen in Appendix A. The procedure for testing ECC RAMs is also incorporated into the TPG VHDL model to execute the BIST configurations. The proposed settings for each BIST configuration can be seen in Table 4.5.

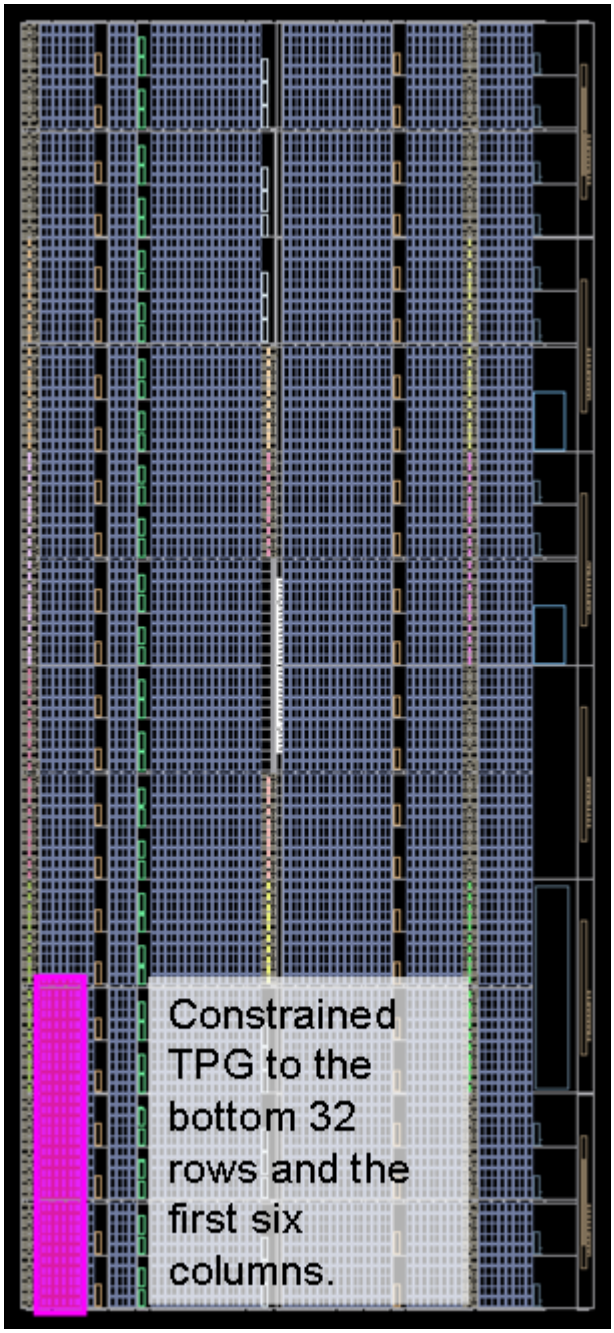


Figure 4.2: RAMB36 TPG Area Constraint for an LX50T Device

Table 4.5: Configuration Settings for RAMB36SDP TPG

(a) Settings Part 1

Config. #	Algorithm	DO REG	EN_ECC READ	EN_ECC WRITE	EN_ECC SCRUB
1	MarchLR with BDS	0	FALSE	FALSE	FALSE
2	ECC (RDEN)	1	TRUE	FALSE	FALSE
3	ECC (WREN)	1	FALSE	TRUE	FALSE

(b) Settings Part 2

Config. #	Algorithm	INIT VAL	SR VAL	INIT (A/B) VAL	SAVE DATA
1	MarchLR with BDS	AAAA	5555	0	FALSE
2	ECC	AAAA	5555	0	FALSE
3	ECC	5555	AAAA	FFFF	FALSE

(c) Settings Part 3

Config. #	Algorithm	RDCLK (U/L) INV	RDEN (U/L) INV	RDRCLK (U/L) INV
1	MarchLR with BDS	not INV	not INV	not INV
2	ECC	not INV	not INV	not INV
3	ECC	INV	INV	INV

(d) Settings Part 4

Config. #	Algorithm	WRCLK (U/L) INV	WREN (U/L) INV	SSR (U/L) INV
1	MarchLR with BDS	not INV	not INV	not INV
2	ECC	not INV	not INV	not INV
3	ECC	INV	INV	INV

Figure 4.3: Shift Register Control String - RAMB36SDP

4.1.3 FIFO36 TPG

The TPG proposed for the FIFO36 primitive is responsible for testing the FIFO functionality. The RAM test algorithm, *FIFOX*, described in Chapter 2, is incorporated into a VHDL model to execute the BIST configurations for this TPG. The proposed settings for each BIST configuration can be seen in Table 4.6.

4.1.4 FIFO36_72 TPG

The proposed FIFO36_72 TPG is responsible for testing the ECC capabilities of the Block RAM when it is configured as a FIFO. The RAM test algorithm, *FIFOX*, described in Chapter 2, is incorporated into a VHDL model to execute the BIST configurations for this TPG. The proposed settings for each BIST configuration can be seen in Table 4.7.

4.2 ORA Placement

The ORAs for the Virtex-5 Block RAM BIST differ from those of the Virtex-4 Block RAM BIST described previously in Section 2.2.3. In the Virtex-5 devices, there are five rows of CLBs adjacent to each Block RAM, rather than the four rows of CLBs in the Virtex-4 devices. Instead of using nine CLBs spread across three columns, as in the Virtex-4 Block RAM BIST, two columns of five CLBs are used for the ORAs.

Table 4.6: Configuration Settings for FIFO36 TPG

(a) Settings Part 1

Config. #	Algorithm	DO REG	DATA WIDTH	EN SYN	FWFT
1	FIFOX	1	36	TRUE	TRUE
2	FIFOX	1	18	FALSE	FALSE
3	FIFOX	0	9	TRUE	TRUE
4	FIFOX	0	4	FALSE	FALSE

(b) Settings Part 2

Config. #	Algorithm	RDCLK (U/L) INV	RDRCLK (U/L) INV	RDEN INV	RST INV
1	FIFOX	INV	INV	INV	INV
2	FIFOX	not INV	not INV	not INV	not INV
3	FIFOX	not INV	not INV	not INV	not INV
4	FIFOX	not INV	not INV	not INV	not INV

(c) Settings Part 3

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

Table 4.7: Configuration Settings for FIFO36_72 TPG

(a) Settings Part 1

Config. #	Algorithm	DO REG	EN_ECC READ	EN_ECC WRITE	EN SYN
1	FIFOX	1	TRUE	FALSE	FALSE
2	FIFOX	0	FALSE	TRUE	TRUE

(b) Settings Part 2

Config. #	Algorithm	FWFT	RST INV RSTINV	ALMOST EMPTY OFFSET	ALMOST FULL OFFSET
1	FIFOX	TRUE	INV	5555	AAAA
2	FIFOX	FALSE	not INV	AAAA	5555

(c) Settings Part 3

Config. #	Algorithm	RDCLK (U/L) INV	RDRCLK (U/L) INV	RDEN INV RDENINV	WRCLK (U/L) INV	WREN INV WRENINV
1	FIFOX	INV	INV	INV	INV	INV
2	FIFOX	not INV	not INV	not INV	not INV	not INV

Each ORA slice contains four 6-input Look-Up Tables (LUTs), which can be seen in Figure 4.4. The A1-A4 inputs to the LUTs are used to compare the designated outputs of the Block RAMs (See Figure 4.5(a)). This results in eight RAM output comparisons in each slice of the CLB, for a total of 160 possible comparisons and since the maximum number of possible outputs for any of the RAM primitives is 118 (RAMBFIFO36) this is more than enough logic to implement the comparisons.

The four LUTs are used differently, depending on the type of the BIST. For the RAMB36 and FIFO36 BIST types, all of the RAM outputs will be implemented as a single ORA comparison. This is accomplished by connecting the same RAM outputs to the A1 and A3 inputs and A2 and A4 inputs as shown in Figure 4.5(b). This effectively results in the comparison ORA acting as if it were the circuit in Figure 4.5(c).

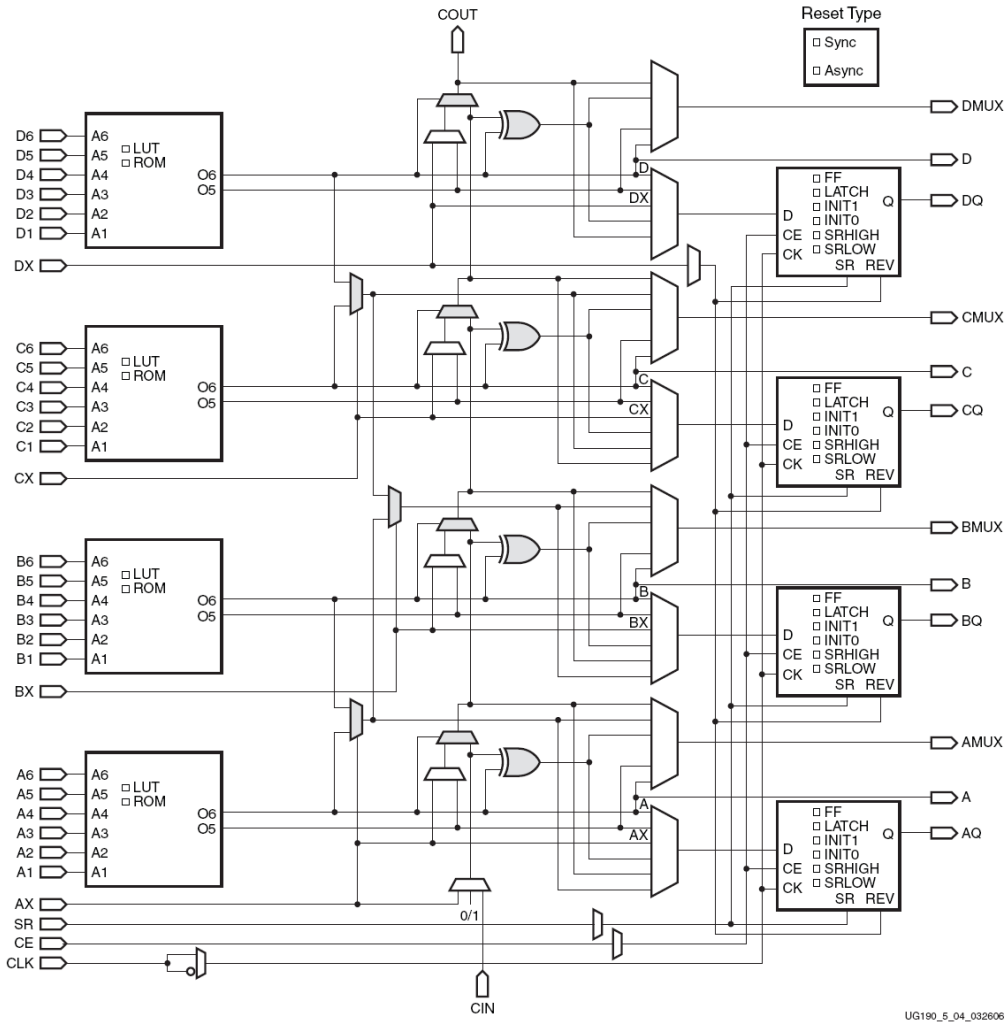


Figure 4.4: Internal Slice Components [16]

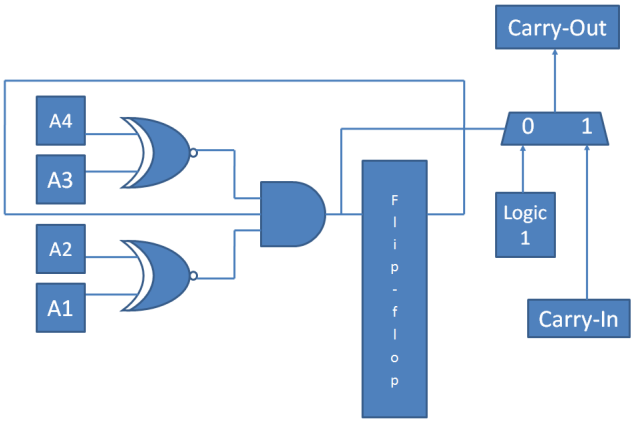
Since the number of outputs for these Block RAM types is less than 80, which is half the total number of possible comparisons if two comparisons were implemented per LUT, this single comparison can be implemented. This comparison technique improves the diagnostic resolution of the BIST by allowing any RAM output failure to be precisely determined.

However, for the RAMB36SDP and FIFO36_72 BIST types, single RAM output comparison is not possible. RAMB36SDP has 82 RAM outputs, while FIFO36_72 has 114 RAM outputs. Therefore, it is necessary to implement the dual comparison in several of the LUTs. Care is taken to implement the outputs that are only associated with that particular RAM primitive as single comparisons, and implement the outputs that had been tested thoroughly in another BIST configuration as dual comparisons.

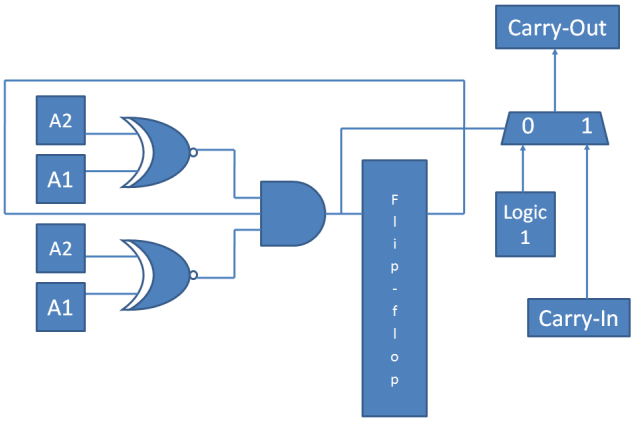
An iterative OR-chain, which was adopted from the Virtex-4 Block RAM BIST (Section 3.2), was implemented for the Virtex-5 BIST [3]. The fast-carry logic within each CLB was exploited to provide a single *Pass/Fail* signal.

4.3 BIST File Generation

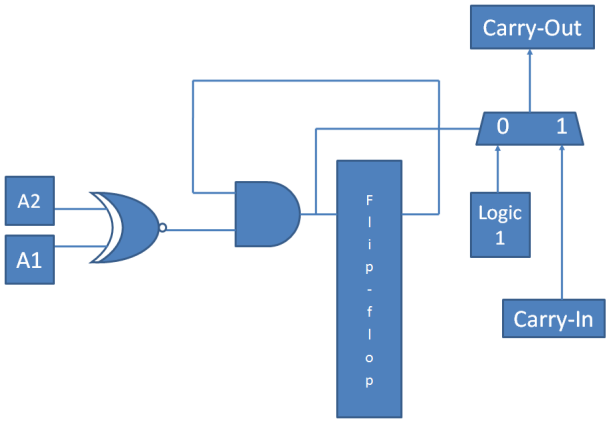
Two programs were developed for the Virtex-5 Block RAM BIST and are similar in operation to those developed for the Virtex-4 Block RAM BIST. The BIST template generation program, *V5BramBIST.exe*, is responsible for algorithmically placing and interconnecting the Block RAMs and ORAs within the user-specified FPGA row/column ranges, as well as placing and interconnecting the two TPGs for the designated BIST type. The modification program, *V5BramMod.exe*, is responsible for correctly configuring the RAMs for the desired BIST configuration.



(a) LUT Double Comparison



(b) LUT Single Comparison



(c) Effective Single Operation

Figure 4.5: Virtex-5 ORA LUT Comparison

4.3.1 BIST Template Generation Program

A BIST template generation program, *V5BramBist.exe*, was developed for the Virtex-5 Block RAM BIST. This program processes user-specified parameters for the desired BIST template to generate. This template can be for any one of the four BIST types described previously. Depending on the BIST type selected, the Block RAM primitive that matches that type is selected and placed within the desired device for all Block RAMs that lie within the specified FPGA row/column range. The command line format provided to the user is given in Figure 4.6.

The ORAs are placed in the two columns of CLBs directly to the left of a Block RAM column and the outputs of the placed Block RAMs are then routed to their specific comparison LUT within the ORA CLBs. Depending on the type, this may either be the single or double comparison described in the previous section, and the iterative OR-chain is routed through the fast-carry logic.

Once the Block RAMs and ORAs are placed and routed, a TPG is extracted from an XDL description of the TPG and stored in several files, which are used to replicate the TPG. The two TPGs, TPG0 and TPG1, are placed in the six CLB columns to the right of the rightmost column of Block RAMs, excluding the columns of Block RAMs that are located in a Tri-mode Ethernet Media Access Controller (TEMAC) column in the LXT, SXT, FXT, and TXT Virtex-5 devices. These columns were selected because they remain unused for the Virtex-5 Block RAM BIST in every Virtex-5 device. One TPG, TPG0, is placed in the lower half of the device, while the other TPG, TPG1, is placed in the upper half of the device. An example of this can be seen in Figure 4.7. The routing is placed between the TPG outputs and their respective Block RAMs, which alternate rows so that faults that may occur within the TPG can be detected.

```
V5RAMbist (v1.4) - 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,p,a]
 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

where the type is defined as:
  RAMB36        = 1
  RAMB36SDP     = 2
  FIFO36        = 3
  FIFO36_72     = 4

npa options:
  n = runs XDL2NCD with -nodrc
  p = runs 'n' option followed by reentrant routing with PAR
  a = runs 'n' & 'p' options and coverts back to XLD

Note:
  All parameters can be upper or lower case (but not mixed).
  Also, the extension on <xdlfile> need not be specified.
  The .xdl extension will be appended by the program.
```

Figure 4.6: *V5BramBist.exe* Command Line Format

Finally the BSCAN interface is added to the template XDL file, which allows the user to shift in a BIST control string (if applicable) and control the BIST clock and reset signals. It also enables the user to toggle *TDI* to observe the *Pass/Fail* signal propagated by the iterative OR-chain.

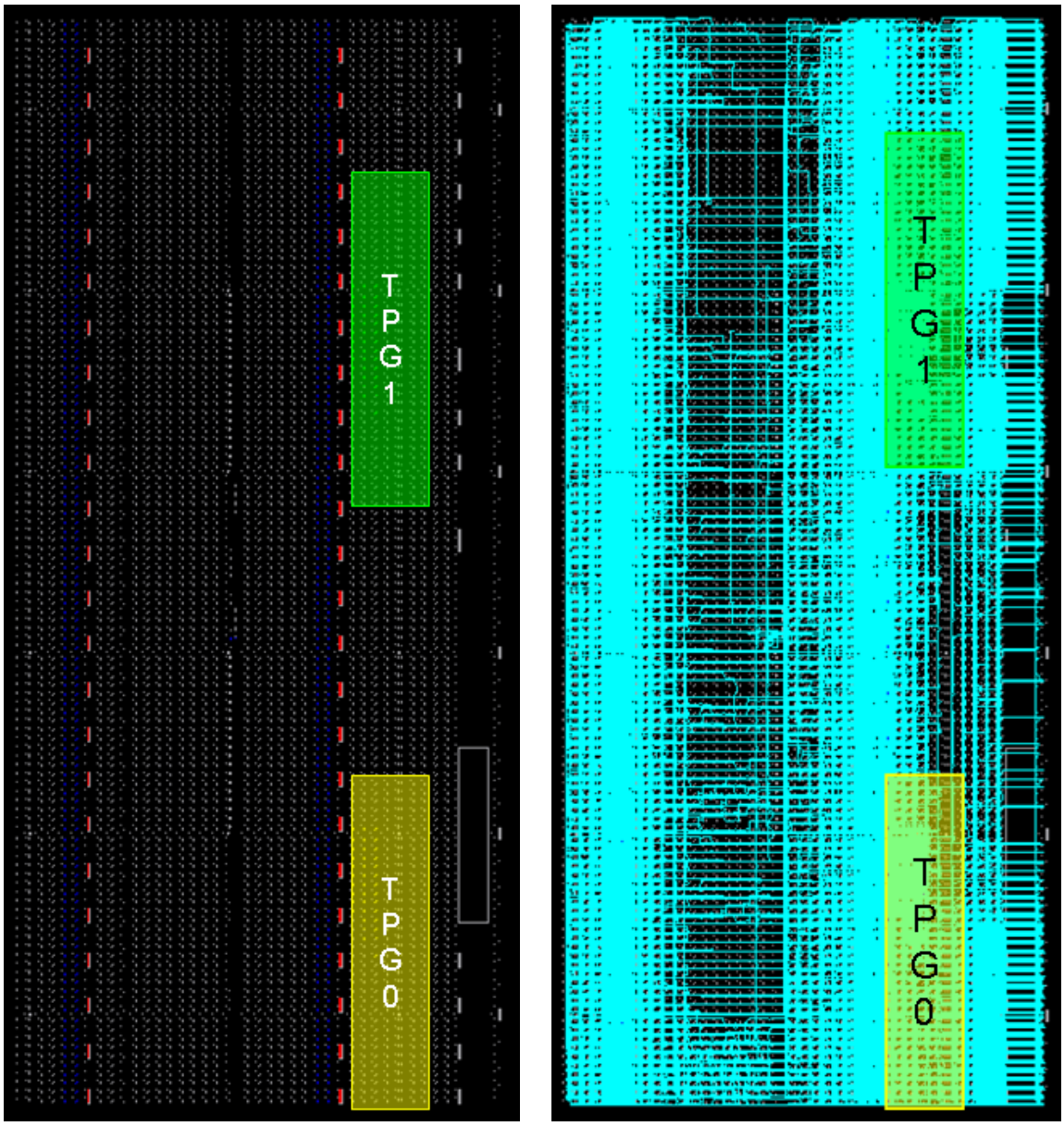
4.3.2 Modification Program

A modification program, *V5BramMod.exe*, was developed for the Virtex-5 Block RAM BIST. This program is responsible for establishing the correct Block RAM configuration parameters for the specified BIST type and the desired BIST configuration. The program also inverts the TPG and ORA clocks for the BIST configurations where opposite edge clocking is to be tested to reduce the impact of inverting the clocks, as was discussed in the previous chapter. The command line format provided to the user is shown in Figure 4.8.

The two additional options allow the user to convert the generated .XDL file into an .NCD file. This .NCD file can be edited in FPGA Editor to insert probes, if the user so desires, or can be manually processed by the user using *Bitgen.exe*, to create the configuration bit files. If the *bit* option is specified, the processed .XDL file will be converted to an .NCD file and then processed using *Bitgen.exe* to create a compressed configuration bit file without any additional effort by the user.

4.4 BIST Results

The subsequent subsections will describe the current results of executing the proposed BIST configurations. These results may change as the Virtex-5 BIST configurations are refined.



(a) TPG Placement (b) TPG Placement in a Routed BIST

Figure 4.7: TPG Placement for a Virtex-5 LX50T

```
V5RAMmod (ver 1.3) - modifies routed XDLs for Block RAM to
                     subsequent BIST configs

Command line format:
V5RAMmod <xdl_in> <xdl_out> <type> <phase> [ncd,bit]

where the type is defined as:
  RAMB36        = 1
  RAMB36SDP     = 2
  FIFO36        = 3
  FIFO36_72     = 4

  Type:       RAMB36       RAMB36SDP    FIFO36      FIFO36_72
-------------------------------------------------------------------
  Phase 1:    S2PF         MarchLR      FIFOx 1K    FIFOx_ECC_RD
  Phase 2:    D2PF         ECC_RD       FIFOx 2K    FIFOx_ECC_WR
  Phase 3:    MATS+ 2K     ECC_WR       FIFOx 4K
  Phase 4:    MATS+ 4K                  FIFOx 9K
  Phase 5:    MATS+ 8K
  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

Note:
  On both <xdl_in> and <xdl_out> the .xdl file extension should
  be specified.
```

Figure 4.8: *V5BramMod.exe* Command Line Format

4.4.1 File Size Comparison

The compressed configuration bit file and compressed partial configuration bit file sizes for the RAMB36 BIST configurations can be seen in Table 4.8. Using compressed partial reconfiguration files yields additional benefits when implementing the Virtex-5 Block RAM BIST, as it did in the Virtex-4 Block RAM BIST. In the case of the RAMB36 configurations the total number of downloaded bits is reduced approximately 4.6 times, which reduces the total BIST execution time.

Inverting all of the TPG and ORA clocks also has an impact on the Virtex-5 partial configuration bit file sizes, as it did in the Virtex-4 partial configuration bit files. In the case of RAMB36, the TPG and ORA clocks are inverted in the first configuration, while the remaining configurations use non-inverted clock signals. This results in the compressed partial configuration bit file for this configuration to be noticeably larger than its counterparts.

Table 4.8: BIST Configuration File Sizes for LX50T Device - RAMB36

Config. #	BIST	Compressed (# bits)	Partials (# bits)
1	S2PF	7,348,384	7,348,384
2	D2PF	7,342,816	689,600
3	MATS 2k	7,342,816	625,536
4	MATS 4k	7,342,816	625,536
5	MATS 8k	7,342,816	625,536
6	MATS 16k	7,342,816	625,536
7	MATS 32k	7,342,816	625,536
	Total	51,405,280	11,165,664

4.4.2 Timing Analysis

The maximum BIST clock frequency for an LX50T device is shown in Figure 4.9. All of the configurations execute faster than the desired BIST clock frequency

of 50 MHz. The MATS+ (2K) configuration has the lowest maximum BIST clock frequency of approximately 65 MHz. This configuration is generated for every Virtex-5 device except LX20T, which is not supported because the number of RAMs in the TEMAC column is insufficient to implement the circular comparison with the ORAs. The worst-case timing analysis and the total number of Block RAMs for each device can be seen in Figure 4.10 and the total number of Block RAMs can be seen above each column. The three largest devices (LX330, LXT330, and SXT240) are the only devices where the worst-case configuration fails to achieve the desired BIST clock frequency. However, for these devices, the BIST can be performed separately on each half of the device, which increases the BIST clock frequency at the cost of an additional set of BIST configurations.

4.4.3 Fault Coverage

Configuration memory bit faults were emulated using the same process as implemented in the Virtex-4 fault simulation. These bits are modified before the BIST is executed to emulate stuck-at fault behavior. The fault coverage accrued by each configuration, as well as the cumulative fault coverage for executing the BIST for the RAMB36 configurations can be seen in Figure 4.11 and the total fault coverage of the RAMB36 BIST configurations is 82.17%, which is an acceptable value for the fault coverage. The RAM test algorithms implemented in the RAMB36 BIST configurations should detect mismatches between the Block RAM initialization values. However, not all of the expected initialization faults were detected, which resulted in a lower fault coverage than could actually be achieved. This is an important area for future investigation.

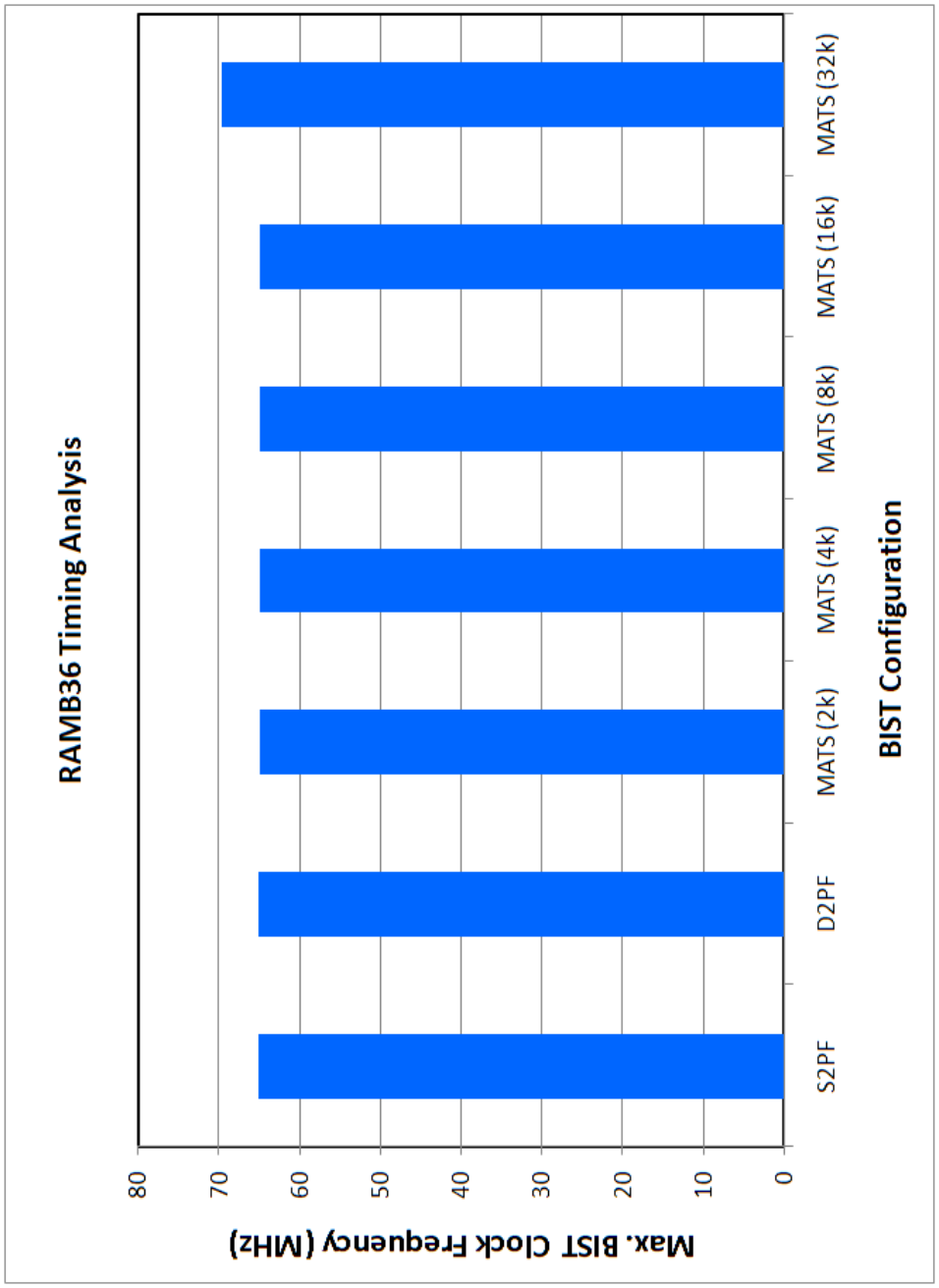


Figure 4.9: Timing Analysis for Virtex-5 LX50T Device

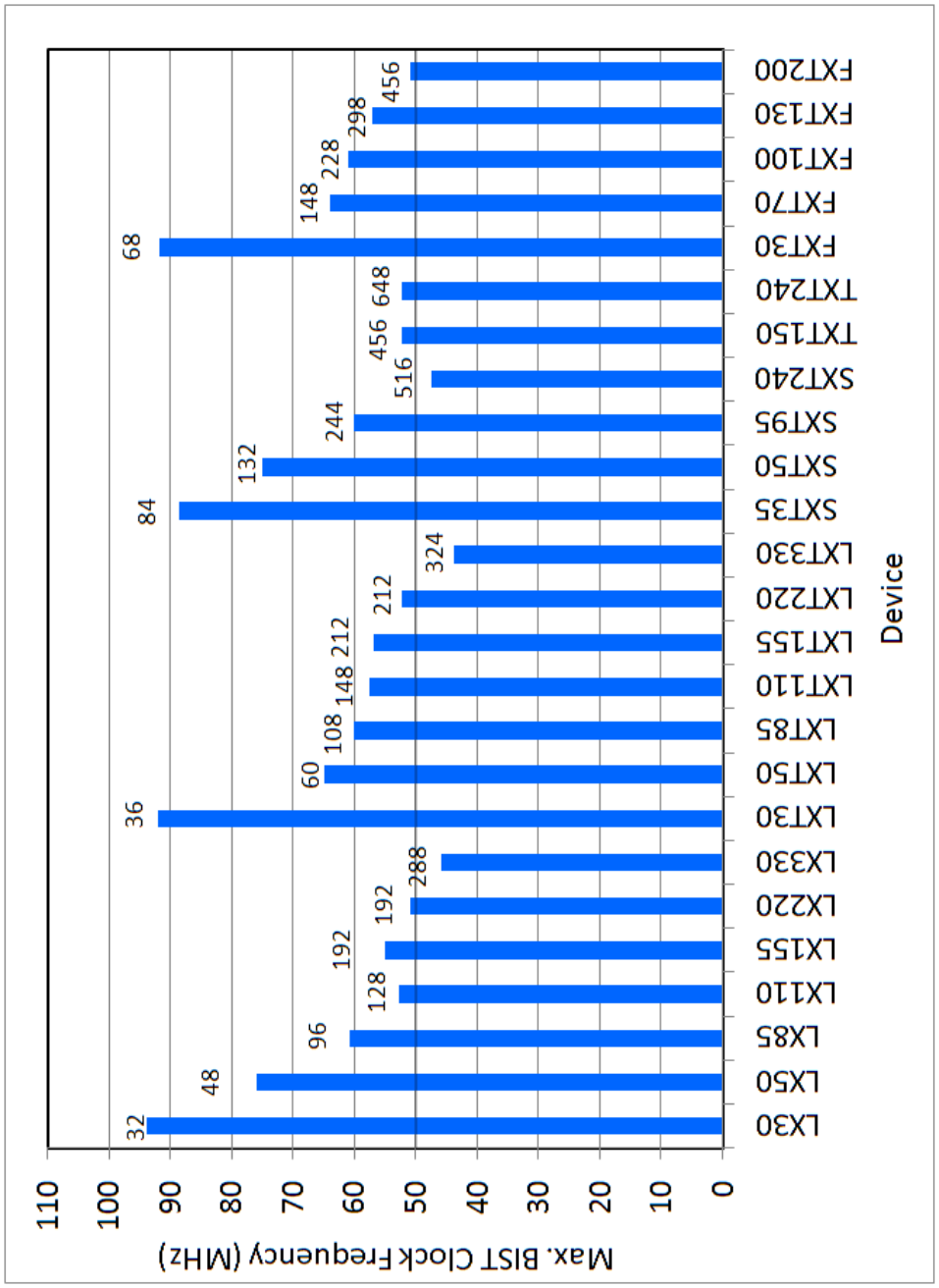


Figure 4.10: Timing Analysis for Worst-Case BIST Configurations - Virtex-5 Devices

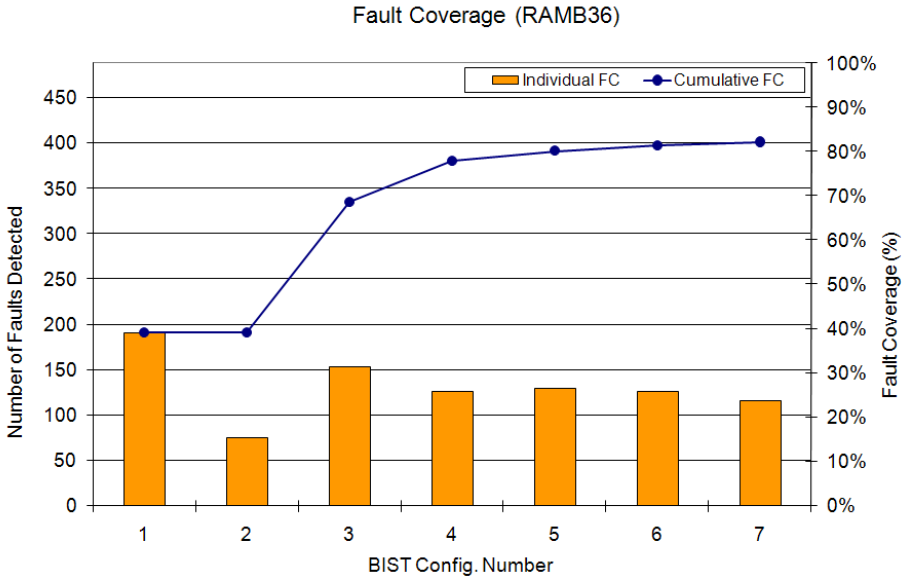


Figure 4.11: RAMB36 Fault Coverage for Virtex-5 Devices

Chapter 5

Summary and Conclusions

This thesis discussed FPGAs, their advantages and their disadvantages, as well as discussing BIST and why it is needed. An overview of RAMs was given and the particular associated faults were presented. The Block RAM BIST developed for the Virtex-4 devices developed by Milton [7] and each component of the BIST were discussed. A preliminary Virtex-5 Block RAM BIST was presented, highlighting the differences between the Virtex-4 and Virtex-5 BIST approaches.

5.1 Virtex-4 Block RAM BIST Improvements

A number of improvements were implemented for the Virtex-4 Block RAM BIST. A major improvement that was implemented but did not directly affect the BIST execution time was the simplification of the twelve Virtex-4 Block RAM BIST generation programs into two programs. This greatly simplified the configuration file generation process from the user's point of view.

Another major improvement was the implementation of the new ORA that used the built-in carry chain to provide the user with a single *Pass/Fail* signal. This signal allowed the user to choose whether or not to perform a configuration memory readback. This results in a much faster BIST execution time if no configuration memory readbacks are performed and the iterative OR-chain is used to determine if a device was fault-free or not. In the case that the configuration memory is read back,

the BIST execution time can still be improved by only performing the readback when the iterative OR-chain indicates a fault was detected.

The development of the two new FIFO BIST configurations also helped to reduce the overall Block RAM BIST execution time. If extensive testing of the *ALMOST-FULL/ALMOSTEMPTY* flags is not desired, the total number of configurations for the FIFO BIST is reduced from fifteen to five configurations. The fix to the *Reset* state also ensured that the BIST will execute properly whenever it is reset during execution.

The addition of clock enables for the bottom-most ORAs and the ORAs directly above the PowerPC (if applicable) allowed for the use of the iterative OR-chain in conjunction with the Block RAM BIST when the RAMs are cascaded. Rather than allow the ORAs to latch up false fault detections, the ORAs are disabled for the clock cycles where known mismatches will occur.

Several improvements were made to the BIST configurations, which tested the opposite-edge clocking of the Block RAMs. In addition to inverting the RAM clocks, setting the clock edges of the ORA and TPG clocks to the opposite-edge clock setting allowed the BIST configuration to run at approximately twice the frequency of what it was previously. This also changed which BIST configurations governed the maximum speed at which the BIST could be executed.

Fault simulation was performed on all of the Virtex-4 BIST configurations to determine the cumulative fault coverage of the Virtex-4 Block RAM BIST, which was found to be 98.67%. This was done by manually overwriting a desired configuration memory bit before executing each BIST configuration and then observing if the fault was detected by the configuration or not.

5.2 Virtex-5 Block RAM BIST

This thesis also presents an initial approach to the Virtex-5 Block RAM BIST. Four Block RAM primitives, *RAMB36, RAMB36SDP, FIFO36* and *FIFO36_72*, are used to implement all of the BIST configurations. TPGs were proposed for each of the four primitives. The TPG for the RAMB36 mode of operation is the most complete and was discussed in greater detail than the other three TPGs.

A new ORA structure was introduced in the Virtex-5 Block RAM BIST to take advantage of the new slice composition. When the number of RAM outputs is less than 80, a single comparison can be implemented in each of the four LUTs contained within a Virtex-5 slice. When the number of RAM outputs is greater than 80, single comparisons are implemented on as many of the RAM outputs as possible, while implementing a double comparison in several LUTs for the RAM outputs which are tested in other BIST configurations.

Two new programs, *V5BramBist.exe* and *V5BramMod.exe*, were developed and are used in the generation of the Virtex-5 Block RAM BIST configuration bit files. The first generates an XDL template description for the desired BIST type, and the second modifies the BIST template to create a specific configuration within that BIST type.

Timing analysis was performed on the configurations generated for the RAMB36 BIST type to determine the maximum BIST clock frequency. The slowest configuration was found to be the MATS+ (2k) configuration with BIST clock frequency of approximately 65 MHz. This configuration was generated for every Virtex-5 device, except the LX20 device, and every device except the three largest devices (LX330, LXT330, and SXT240) were found to operate at a frequency that exceeds the desired BIST clock frequency of 50 MHz.

Fault emulation was also performed for the RAMB36 TPG type. A fault coverage of 82.17% was achieved using just these configurations. However, detection of the Block RAM initialization faults was inconclusive. Under certain conditions no initialization faults would be detected, while under a different set of conditions all initialization faults would be detected.

5.3 Future Work

Currently, only the modifications for the RAMB36 Block RAM type are believed to be functional. It is recommended that for future research, the functionality of the other three Block RAM types be verified, rather than assuming them to be correct. This can include generating the BIST configurations for the partially complete TPG (RAMB36SDP), up to designing the complete VHDL models for the FIFO36 and FIFO36_72 TPGs based on the proposed settings in Chapter 4. Also, the issue of only part of the Block RAM initialization mismatches being detected should be explored.

Timing analysis should be performed on all of the newly created BIST configurations to determine the worst-case timing configuration for each of the four TPG types. The newly determined worst-case timing configurations should be generated for every device to determine which devices, such as the LXT300, should perform the BIST on partial arrays rather than the whole of the device.

Fault emulation should also be performed on all of the newly developed BIST configurations to determine the overall fault coverage provided by executing the complete set of BIST configurations on a Virtex-5 device. This fault coverage is desired to be as high as possible.

The ultimate goal, however, is to understand the functionality of the four designed BIST types well enough to be able to incorporate them into a single TPG for

the undocumented RAMBFIFO36 Block RAM type introduced at the beginning of Chapter 4. Drastically decreased BIST test time can result from the culmination of this goal as only one full, compressed configuration bit file need be downloaded to the device, while the subsequent configurations can be compressed partial configuration bit files.

Bibliography

[1] A. J. Van de Goor, *Using March Tests to Test SRAMs*, IEEE Design and Test of Computers, **10** (1993), no. 1, 8–14.

[2] Ad J. Van de Goor and I.B.S Tlili, *March Tests for Word-Oriented Memories*, Design, Automation and Test in Europe, Design, Automation and Test in Europe, 1998, pp. 501–508.

[3] Bradley F. Dutton and Charles E. Stroud, *Built-In Self-Test of Configurable Logic Blocks in Virtex-5 FPGAs*, IEEE Southeastern Symposium on System Theory, IEEE Southeastern Symposium on System Theory, 2009, pp. 230–234.

[4] Srinivas Garimella, *Built-In Self Test for Regular Structure Embedded Cores in System-on-Chip*, Master's thesis, Auburn University, 2005.

[5] Said Hamdioui, *Testing Static Random Access Memories*, Kluwer Academic Publishers, 2004.

[6] Ian Kuon and Johnathon Rose, *Measuring the Gap between FPGAs and ASICs*, Computer-Aided Design of Integrated Circuits and Systems, **26** (2007), no. 2, 203–215.

[7] Daniel Milton, *Built-In Self-Test of Configurable Memory Resources in Field-Programmable Gate-Arrays*, Master's thesis, Auburn University, 2007.

[8] Mary Pulukuri and Charles E. Stroud, *Built-In Self-Test of Digital Signal Processors in Virtex-4 FPGAs*, IEEE Southeastern Symposium on System Theory, IEEE Southeastern Symposium on System Theory, 2009, pp. 34–38.

[9] Charles H. Roth and Lizy Kurian John, *Digital Systems Design using VHDL*, Thomson Learning, 2008.

[10] Michael John Sebastian Smith, *Application-Specific Integrated Circuits*, Addison Wesley, 1997.

[11] Charles E. Stroud, *A Designer's Guide to Built-In Self-Test*, Kluwer Academic Publishers, 2002.

[12] Charles E. Stroud and Srinivas Garimella, *Built-In Self-Test and Diagnosis of Multiple Embedded Cores in SoCs*, Proceedings, International Conference on Embedded Systems and Applications, International Conference on Embedded Systems and Applications, 2005, pp. 130–136.

[13] Laung-Terg Wang, Charles E. Stroud, and Nur A. Touba, *System-on-Chip Test Architectures*, Morgan Kaufmann Publishers, 2008.

[14] Xilinx, *Development System Reference Guide*, 2008.

[15] _____, *Virtex-4 FPGA User Guide*, 2008.

[16] _____, *Virtex-5 FPGA User Guide*, 2008.

Appendix A

MarchLR with 72-bit BDS

The following 72-bit MarchLR algorithm and accompanying BDS sequence were developed directly from the method described in [2]. The BDS sequence can also be optimized to eliminate the redundant march elements (Element 7 and 8) found within the BDS and the MarchLR sequence. This reduces the test time from *O(23 N)* to *O(21 N)*, where N is the number of address locations.

	March Element	Address Direction	RAM Operation	Data Hex Value
MarchLR	1	up/down	write	0000000000000000
	2	down	read	0000000000000000
			write	FFFFFFFFFFFFFFFF
	3	up	read	FFFFFFFFFFFFFFFF
			write	0000000000000000
			read	0000000000000000
			write	FFFFFFFFFFFFFFFF
	4	up	read	FFFFFFFFFFFFFFFF
			write	0000000000000000
	5	up	read	0000000000000000
			write	FFFFFFFFFFFFFFFF
			read	FFFFFFFFFFFFFFFF
			write	0000000000000000
	6	up	read	0000000000000000
BDS	7	up	read	0000000000000000
			write	FFFFFFFFFFFFFFFF
			read	0000000000000000
	8	down	read	FFFFFFFFFFFFFFFF
			write	0000000000000000
			read	0000000000000000
	9	up	read	0000000000000000
			write	555555555555555
			write	AAAAAAAAAAAAAAAA
			read	AAAAAAAAAAAAAAAA
	10	down	read	AAAAAAAAAAAAAAAA
			write	555555555555555
			read	555555555555555
	11	up	read	555555555555555
			write	333333333333333
			write	CCCCCCCCCCCCCCCC
			read	CCCCCCCCCCCCCCCC
	12	down	read	CCCCCCCCCCCCCCCC
			write	333333333333333
			read	333333333333333
	13	up	read	333333333333333
			write	0F0F0F0F0F0F0F0F
			write	F0F0F0F0F0F0F0F0
			read	F0F0F0F0F0F0F0F0

	March Element	Address Direction	RAM Operation	Data Hex Value
BDS	14	down	read	F0F0F0F0F0F0F0F0
			write	0F0F0F0F0F0F0F0F
			read	0F0F0F0F0F0F0F0F
	15	up	read	0F0F0F0F0F0F0F0F
			write	FF00FF00FF00FF00
			write	00FF00FF00FF00FF
			read	00FF00FF00FF00FF
	16	down	read	00FF00FF00FF00FF
			write	FF00FF00FF00FF00
			read	FF00FF00FF00FF00
	17	up	read	FF00FF00FF00FF00
			write	FF0000FFFF0000FF
			write	00FFFF0000FFFF00
			read	00FFFF0000FFFF00
	18	down	read	00FFFF0000FFFF00
			write	FF0000FFFF0000FF
			read	FF0000FFFF0000FF
	19	up	read	FF0000FFFF0000FF
			write	FF00000000FFFFFF
			write	00FFFFFFFF00000000
			read	00FFFFFFFF00000000
	20	down	read	00FFFFFFFF00000000
			write	FF00000000FFFFFF
			read	FF00000000FFFFFF
	21	up	read	FF00000000FFFFFF
			write	00FFFFFFFFFFFFFFFF
			write	FF0000000000000000
			read	FF0000000000000000
	22	down	read	FF0000000000000000
			write	00FFFFFFFFFFFFFFFF
			read	00FFFFFFFFFFFFFFFF
	23	up	read	00FFFFFFFFFFFFFFFF