Failure Estimation for Partial TMR Mitigated Designs in a Virtex-4 FPGA

by

Scott Arlo Anderson

B.S., Electrical Engineering, Utah State University, 1999
B.S. Physics/Math, Utah State University, 1999

A report submitted to the
University of Colorado Denver
in partial fulfillment
of the requirements for the degree of
Master of Science
Department of Electrical Engineering
2010
This report for the Master of Science degree by Scott Arlo Anderson has been approved by

Daniel A. Connors

Robert Grabbe

Tim C. Lei

Paul Rutt

Date ____________________
ABSTRACT

SRAM-based Field Programmable Gate Arrays (FPGAs) have proven reliable for many applications in harsh radiation environments. As the technology matures, more harsh-environment electronic design programs begin to consider the options regarding these FPGAs. Not surprisingly, the usual business constraints still apply, where the best trade-off between costs, schedule, and performance is sought. The Xilinx Virtex-4 FPGA family is especially attractive for their reprogrammability and more powerful performance. Reprogrammability improves schedule flexibility and versatility not available in the competing technologies of Application-Specific Integrated Circuits (ASICs) and one-time programmable FPGAs. However, a harsh environment creates difficulty for the SRAM configuration memory. High-energy particles hitting the FPGA will disturb the configuration memory cells which can change the functionality in random ways, possibly risking mission failure. Popular approaches to mitigating these effects include configuration scrubbing and external functional-event monitoring to deal with configuration upsets, and fault-tolerant design methods such as Triple Modular Redundancy (TMR) to deal with functional upsets. This paper considers the common challenges to achieving perfect TMR, including design constraints and tool limitations. Through fault injection experiments, a framework of rules is proposed to guide the FPGA designer to make effective trade-offs in early stages of design.

This abstract accurately represents the content of the candidate’s report. I recommend its publication.

Daniel A. Connors
Contents

Chapter

1 Introduction ................................................. 1

2 Triple Modular Redundancy in the Xilinx Virtex-4 ...... 3

  2.1 Xilinx Virtex-4 FPGA ................................... 3

  2.2 Triple Modular Redundancy .............................. 5

    2.2.1 Basic TMR .............................................. 6

    2.2.2 Xilinx TMR .............................................. 6

    2.2.3 Throughput Logic ..................................... 7

    2.2.4 Feedback Logic ....................................... 7

  2.3 Scrubbing ............................................... 8

  2.4 Analyzing TMR Effectiveness ........................... 9

3 Test Hardware ............................................... 11

  3.1 Xilinx Radiation Test Consortium ...................... 11

  3.2 SEAKR Motherboard ...................................... 11

    3.2.1 CfgMon FPGA ......................................... 12

    3.2.2 FuncMon FPGA ......................................... 13

  3.3 Virtex-4 LX200 Daughter Card .......................... 14

  3.4 Brain Box ............................................... 14
6 Analysis

6.1 Monolithic Versus Building Block .................................. 34
6.2 Throughput Versus Feedback ........................................... 35
6.3 Related Logic .............................................................. 38
6.4 Dependence on I/Os ......................................................... 39
6.5 Results for Singular I/Os .................................................... 40
6.6 Results for Convergence Points ........................................... 41
6.7 Comparative Impact ......................................................... 42

7 Summary and conclusion .................................................. 44

Bibliography ................................................................. 46
Tables

Table

6.1 Sample output from MAP tool report .................................. 38
6.2 Single points of failure for several numbers of voters ............... 43
Figures

Figure

2.1 Configurable Logic Block. ......................................................... 4
2.2 CLB Slice Internals. ................................................................. 4
2.3 Triple Modular Redundancy. ....................................................... 5
2.4 Majority Voter. ................................................................. 6
2.5 Xilinx TMR. ................................................................. 7
2.6 Throughput Logic. ................................................................. 8
2.7 Feedback Logic. ................................................................. 9
3.1 SEAKR Radiation Motherboard Block Diagram. ......................... 12
3.2 SEAKR Radiation Motherboard Hardware. ................................. 13
3.3 (a) LANL Virtex-4 LX200 DUT Daughter Card. (b) Brain Box. .... 14
3.4 The CfgMon GUI with FaultMon Sidecar. ................................. 15
3.5 FuncMon GUI. ................................................................. 16
4.1 Curve Predictions for System Error Rate Versus Raw Bit Error Rate. . 23
5.1 Register Table Design Block Diagram. ........................................ 26
5.2 Register Table Depth and Width. ............................................... 27
5.3 Mux Stage Design Top-Level Block Diagram. ............................... 28
5.4 Mux Select Decoding and Loading at Each Stage. ....................... 29
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.5</td>
<td>Counter Design Top-Level Block Diagram</td>
<td>31</td>
</tr>
<tr>
<td>5.6</td>
<td>Individual Four-Bit Counter</td>
<td>32</td>
</tr>
<tr>
<td>5.7</td>
<td>Counter Stage Groups</td>
<td>32</td>
</tr>
<tr>
<td>6.1</td>
<td>Register Table and Mux Stage SPF Versus Slices</td>
<td>35</td>
</tr>
<tr>
<td>6.2</td>
<td>Counter SPF Versus Slices</td>
<td>36</td>
</tr>
<tr>
<td>6.3</td>
<td>Counter SPF Versus Voters</td>
<td>37</td>
</tr>
<tr>
<td>6.4</td>
<td>Mux Stage SPF Versus Total I/Os</td>
<td>40</td>
</tr>
<tr>
<td>6.5</td>
<td>Mux Stage SPF Versus Singular I/Os</td>
<td>41</td>
</tr>
<tr>
<td>6.6</td>
<td>Mux Stage SPF Versus Convergence Points</td>
<td>42</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

SRAM-based Field Programmable Gate Arrays (FPGAs) have proven reliable for many applications in harsh radiation environments. As the technology matures, more electronic design programs targeted for harsh environments begin to consider the options regarding these FPGAs. Not surprisingly, the usual business constraints still apply, where the best trade-off between costs, schedule, and performance is sought. SRAM-based FPGAs are especially attractive for their reprogrammability and more powerful performance.

The Xilinx Virtex-4 FPGA [1] family has significant advantages to consider. Reprogrammability improves schedule flexibility and versatility not available in the competing technologies of Application-Specific Integrated Circuits (ASICs) and one-time programmable FPGAs. However, a harsh environment creates difficulty for the SRAM configuration memory. High-energy particles hitting the FPGA will disturb the configuration memory cells which can change the functionality in random ways. Serious consequences can result, including design failure, system failure, or even mission failure.

Much study has been devoted to the failure modes ([2], [3]). Single-event upset (SEU) and single-event functional interrupt (SEFI) effects are well known in the Virtex-4. Popular approaches to mitigating these effects include configuration scrubbing and external functional-event monitoring to deal with configuration upsets, and
fault-tolerant design methods such as Triple Modular Redundancy (TMR) to deal with functional upsets.

For designs considering TMR in a single Virtex-4 FPGA, first the rate of SEFI upsets must be acceptable for the program requirements. Then the application of TMR must be analyzed with the physical interfaces and performance requirements. Indeed, perfect implementation of TMR may not be feasible and a best effort approach must be adopted. However, there are many issues reported in [4], [5], [6], etc from previous research for the FPGA designer to consider. Some challenges are due to design constraints requiring untriplicated logic, such as singular I/Os and clock domain crossings. Other challenges are due to intrinsic single points of failure (SPF) that still remain after implementing TMR using the available tools.

This paper explains the relevant concepts to TMR in chapter 2. A description of the test and fault injection hardware follows in chapter 3. Chapter 4 explores the challenges to design of FPGA electronics in harsh environments. The three simple, but representative designs used for this investigation are presented in chapter 5. Curve fits of the data lead to the identification of voters as a key element, and a framework of rules is proposed in chapter 6. The rules show promise as a guide for the FPGA designer to make effective trade-offs in early stages of design.
Chapter 2

Triple Modular Redundancy in the Xilinx Virtex-4

The Xilinx Virtex-4 FPGA is a fourth generation device for the high-performance commercial market. When the Virtex-4 is used in conjunction with fault tolerance and upset mitigation techniques, the Virtex-4 performance compares well with other devices suitable for harsh environments.

2.1 Xilinx Virtex-4 FPGA

The Virtex-4 product line offers a family of FPGAs implemented in a 90-nm fabrication process. Each device combines configurable input/output blocks (I/O or IOB), embedded memory, logic blocks and other special features, including serializer/deserializers (SERDES) and high-speed math blocks called DSP48s. The product family is split into three lines, the LX, SX, and FX, each line emphasizing different features.

Within the Virtex-4 FPGA, the basic logic functions are performed in the configurable logic blocks (CLBs). Each CLB contains four slices (Figure 2.1), and each slice provides multi-purpose logic functions, including two look-up tables (LUTs), two flip-flop storage elements (registers), arithmetic logic, multiplexers (muxes), and fast-carry look-ahead chains (Figure 2.2). Mated with the CLB is a routing structure called the Switch Matrix which connects the CLB to the network of local routing and global rout-
Figure 2.1: Configurable Logic Block [7].

Figure 2.2: CLB slice internals include LUTs, registers, muxes, and arithmetic logic with fast-carry look-ahead chains [7].
the two pins in the I/O can operate differentially. The largest of the Virtex-4 LX family is the V4LX200 device, which holds 89,088 slices and 960 user I/O in 17 banks.

2.2 Triple Modular Redundancy

Triple Modular Redundancy (TMR) seeks to eliminate all SPF. The fundamental concept takes the logic in the design, makes three identical redundant copies, and majority votes the outputs of the circuit as drawn in Figure 2.3. TMR can be applied at

![Figure 2.3: Triple Modular Redundancy [8].](image)

the device level, where three devices run the same design in parallel and have a fourth device for voting, or it can be applied within a single device.

If we assume single-event effects (SEEs) upon configuration memory or user-designed functional circuits can only occur one at a time, TMR will prevent the corruption from affecting the output of the design. TMR can also protect against some multi-event effects, but it cannot protect against all. Specifically, if two upsets occur “simultaneously” in two domains of the same triplicated logic, the majority vote would choose the corrupted result instead of the correct result.
2.2.1 Basic TMR

When the circuit that performs a majority vote (see Figure 2.4) is hardened against upset, basic TMR will suffice. Basic TMR implements a single voter for the three copies at critical nodes. Critical nodes include internal state registers or outputs of the circuit. In the Virtex-4 FPGA, the majority voters are implemented in LUTs, which are susceptible to upsets in their configuration. With a Virtex-4 design using basic TMR, each voter represents an SPF.

![Figure 2.4: A Common Majority Vote Circuit [8].](image)

2.2.2 Xilinx TMR

Xilinx improved the basic scheme to protect against the SPF in the voter which is called Xilinx TMR (XTMR). In XTMR, as shown in Figure 2.5, the voters are also triplicated, which consumes more logic but improves the reliability. Outputs are further enhanced with a minority voter to prevent drive fights which may damage device and/or board components. When the inputs and outputs are also triplicated, the SPF s are eliminated.
Figure 2.5: TMR adapted to the Xilinx FPGAs by adding triplicated input pins, majority voters, and minority-voted tri-state outputs [8].

2.2.3 Throughput Logic

Logic that feeds forward only, such as a data path in which a new value is calculated every clock cycle and is not dependent on the state of the current or downstream stages is called “throughput logic.” In the case of throughput logic, the XTMR method simply creates three identical copies of the circuit and inserts voters only at the outputs as shown in Figure 2.6.

2.2.4 Feedback Logic

Logic that depends on the state of current or downstream stages is called “feedback logic.” If an upset occurs in logic that feeds back, three copies of the same design can drift due to an SEU. Without protection from drift, the eventuality of a second
Figure 2.6: Throughput logic is simply triplicated and voted at the outputs.

event will defeat the TMR method.

To prevent the three copies of functional state from drifting over time, voters are inserted at feedback points as shown in Figure 2.7. Feeding back the voted value enables a corrupted domain to resynchronize with the two correct domains.

2.3 Scrubbing

Scrubbing is the process of reloading the correct design in the configuration memory of the FPGA. The Virtex-4 supports active scrubbing, which means the FPGA can be actively functioning while the configuration is reloaded. A scrub cycle is a single iteration of reloading the complete design. The scrub period is the time it takes to complete one scrub cycle, and the scrub rate is the inverse of the scrub period.

Scrubbing cannot be done instantaneously, and the time-delay between upset and recovery is dependent on the scrub period or scrub rate. Simultaneous events are those that occur within the same scrub cycle. The longer the scrub period, the more likely it
Figure 2.7: Feedback points have voters inserted to prevent drift between triplicated domains over time [8].

is that simultaneous events may occur, which increases the likelihood that TMR may be defeated.

2.4 Analyzing TMR Effectiveness

Common techniques to analyze the effectiveness of TMR are

- accelerated beam testing,
- hardware-based in-situ fault injection,
- gate-level simulation with random upsets, and
- static circuit analysis tools.
In each approach, the measured results can be mapped to the number of SPF in the design. Better methods to finding the number of SPF through analysis or experimentation is the subject of ongoing research, including this paper.
Test Hardware

The test hardware setup used in this research was made available by SEAKR Engineering, Inc. A description of the hardware and fault injection apparatus follows.

3.1 Xilinx Radiation Test Consortium

The Xilinx Radiation Test Consortium (XRTC), originally known as the Single-Event Effects (SEE) Consortium [9], was founded in 2002 by Xilinx and the Jet Propulsion Laboratory (JPL) to combine the efforts of many space industry companies, government agencies, and educational institutes toward the common goal of using reconfigurable Xilinx SRAM-based FPGAs in space. These efforts include radiation testing and characterization of the Virtex family of FPGAs, how to mitigate upset modes, and methods to qualify FPGA applications for space environments.

To further the efforts of the consortium members, a common test platform was developed. The test platform includes the SEAKR motherboard, a custom daughter card, two “Brain Boxes,” and software running on test PCs.

3.2 SEAKR Motherboard

The SEAKR motherboard forms the backbone for the test apparatus. Figure 3.1 shows the block diagram and Figure 3.2 show the assembled hardware for the mother-
board. It has two mezzanine connectors to allow mating with a daughter card. The
daughter card holds the device-under-test (DUT) FPGA, which will get exposed to ra-
diation at a beam accelerator. The motherboard has two more Virtex-II Pro FPGAs,
the Configuration Monitor (CfgMon) and the Functional Monitor (FuncMon), which
control and observe the external signals of the DUT FPGA through the test connectors.

Two banks of 40-pin IDE connectors interface to test PCs via the Brain Boxes.

3.2.1 CfgMon FPGA

The CfgMon FPGA controls the loading of the designs in the DUT and FuncMon
FPGAs. CfgMon scrubs the DUT maintaining the integrity of the design. When SEFIs
are detected, it responds to reload the design. Configuration status is reported to one of the Brain Box through external connectors. A common interface bus allows communication between the FuncMon and CfgMon.

### 3.2.2 FuncMon FPGA

The FuncMon FPGA primarily controls the test patterns applied to and received from the DUT FPGA. The FuncMon also interfaces to the second Brain Box through external connectors. Using the common interface bus, special functions are implemented.
Figure 3.3: (a) LANL Virtex-4 LX200 DUT Daughter Card. (b) Brain Box.

to automate the fault injection process.

3.3 Virtex-4 LX200 Daughter Card

The Virtex-4 LX200 DUT daughter card was designed by Los Alamos National Labs (LANL) in cooperation with the XRTC. Figure 3.3(a) shows the daughter card which includes the DUT V4LX200 FPGA, an Ethernet port, QDR SRAM, and an RS-232 connector. The DUT connects to the FuncMon FPGA via the mezzanine connectors with 3 major buses, the single bus, the TMV bus, and the differential bus. The single bus has 145 bits, the TMV bus has 47 bits, and the differential bus has 42 bits.

3.4 Brain Box

The Brain Box, shown in Figure 3.3(b), holds an FPGA and four connectors, plus various other push button controls and status displays. The CfgMon and FuncMon
each have a dedicated subset of the 40-pin IDE connectors, to which two Brain Boxes are connected. The Brain Box automates the reading of status registers in the CfgMon and FuncMon and passes the data on to the test PC.

3.5 Test PC

Two graphical user interface (GUI) programs have been written, one for the CfgMon and one for the FuncMon. A test personal computer (PC), or two test PCs, runs the GUIs to control signals to the Brain Box which are forwarded to the CfgMon FPGA or the FuncMon FPGA. The test PC also receives status back from the FPGA via the Brain Box and logs the data to files.

![Image of CfgMon GUI with FaultMon Sidecar]

Figure 3.4: The CfgMon GUI with FaultMon Sidecar.
3.6 Fault Injection

Fault injection can be performed using two GUIs [10]. Figure 3.4 shows the CfgMon/FaultMon GUI, which enables the CfgMon FPGA to step through the configuration memory, corrupting each bit and then repairing, one at a time. Between the corruption and the reparation, handshakes between the FuncMon FPGA and CfgMon FPGA coordinate the execution of the test patterns and the pass/fail status of the test. The FuncMon GUI (see Figure 3.5) displays the current status of each test. Both GUIs save results to log files.

Fault injection can identify the SPF present in a design mitigated with full or partial TMR. The SPF are defined by any error occurring in the design which prevents the correct functional result when the design is exercised by thorough test patterns. The SPF information can then be used to determine if and what design improvements are necessary.
Chapter 4

Designing for Harsh Environments

Designing electronics to work in harsh environments is more complex than common environments. The use of SRAM-based FPGAs complicates matters even further. For common environments, the usual design practices include

- behavioral simulation and verification to prove functional correctness
- spice simulation for electrical drive strength, noise, and signal integrity analysis
- synthesis and place and route to map behavioral design to available circuitry
- power and thermal analysis
- timing analysis and closure
- hardware verification

In addition to the usual practices, harsh environments require the design to be analyzed for fault tolerance due to a random upset. Extra steps are required to determine failure modes and likelihood of occurrence, followed by the application of mitigation techniques.

4.1 Common Failure Modes

CMOS transistor-based electronics have several common failure modes that occur due to high-energy particles. Some modes are destructive to the transistors themselves, and some are non-destructive failures that can be repaired. Previous papers [11] and [12] have included the following list:
• Destructive
  ◦ Single-event Latchup (SEL) failures
  ◦ Single-event Gate Rupture (SEGR) failures

• Non-Destructive
  ◦ Single-event Functional Interrupt failures
  ◦ Single-event Upset failures
  ◦ Single-event Transient (SET) failures

4.2 Advantages of SRAM-based FPGAs

SRAM-based FPGAs offer many desirable features for harsh environments in comparison to fuse-based FPGAs, general purpose processors, and ASICs. As described in [13] and [14], these features include:

- application-specific design capability which can perform operations faster and more efficiently than traditional processors
- versatility to perform any user-specified operation
- post-launch configurability means flexibility for multi-purpose roles
- adaptability to avoid hard FPGA errors over device lifetime
- greater speed and density due to commercial gains in transistor technology

On the other hand, the SRAM cells used for the FPGA configuration memory expose the application to more failure modes.

4.3 Disadvantages of SRAM-based FPGAs

In comparison to ASICs and fuse-based FPGAs, SRAM-based FPGAs are more susceptible to SEUs and SEFIs. As the SRAM-based FPGA technology improves, and circuits are run at higher speeds, the SETs also become more of an issue.
SEUs can show up in two forms. First, the configuration memory can upset possibly changing the circuit functionality, e.g. an AND gate may change into an OR gate. The routing may get disturbed, either disconnecting the correct logic, or connecting unintended logic. Second, the user-design state may also get upset. For example, a state machine register may upset causing an unexpected transition or even force an undecoded state. A phase-locked loop (PLL) in a clock circuit may lose lock disturbing the synchronous behavior of the circuit.

SEFIs are possible when the internal circuitry controlling the configuration of the FPGA gets upset. The configuration reset process may be unintentionally triggered clearing the entire design. While the impact of these upset modes is significant, many mitigation techniques have been developed to minimize the effect.

4.4 Virtex-4 Improvements

Each generation of the Xilinx Virtex product line has been extensively tested by the research community. Its fabrication process allows the device to withstand the destructive upset modes of many harsh environments [15].

The Virtex-4 carries forward and improves upon the features of the previous generations Virtex, and Virtex-II. Critical to the management of SEU and SEFI effects is the active readback and reconfiguration capabilities. Scrubbing can restore all CLB functions without disturbing the design. The SelectMAP port provides a high-speed programming interface to quickly load and reload the entire design when SEFIs are detected.
4.5 Upset Mitigation

In addition to scrubbing and SEFI monitoring, applying other common redundancy techniques can further avoid design upsets due to SEUs and SEFIs. Many papers ([8], [16], and [17]) have explored the effectiveness of the various techniques. For data protection, error detection and correction (EDAC) techniques such as Hamming encoding or Reed-Solomon encoding can repair isolated corruption. However, these techniques can fail in FPGAs if an upset causes the circuitry to corrupt multiple bits.

Triple Modular Redundancy (TMR) is a favored technique due to its ability to prevent system errors under normal assumptions. Typically upsets occur randomly and independently. If the rate of upset is slow enough that the recovery techniques can prevent the accumulation of multiple upsets, the majority voting maintains the correct result. In other words, isolated or single failure events are assumed, and TMR can theoretically protect against all SPF.

4.6 Challenges to Achieving Perfect TMR in a Single Device

Many high performance designs have elements that cannot perfectly implement TMR in a single device. For example, pure combinational paths may also be required to be glitch-free. Triplicating and voting without the synchronizing effect of the clock would likely introduce glitches. Another example is the reception of an asynchronous serial data stream. Sampling the data and synchronizing to the local clock domain cannot guarantee each domain would synchronize in lock-step. Similarly, when transferring data across clock domains, the strobe signals are singular to ensure data coherency. TMR applied to the strobos cannot guarantee lock-step functionality across the three domains.
Furthermore, physical limitations can prevent the application of TMR. Many high speed I/O standards have such tight physical constraints that triplicating the pins in a customary wire-OR fashion corrupts the integrity of the signal. Resources may not be sufficient in a single device for the three copies of the basic design. Another hazard for TMR occurs with CAD tools. Optimization techniques from synthesis and place and route tools can also have the unintended consequence of removing a redundant copy. Often, the SPF is difficult to detect in the optimized circuit by inspection.

The Xilinx Virtex-4 architecture has other intrinsic features that can defeat TMR through configuration upsets. As reported in [18], the Switch Matrix (or “Switch Box”) has multiple-stages, each stage providing a group of signals to connect. Single bits in the configuration can switch multiple signals through the switch. Without TMR awareness, the routing tool may combine two copies of the same signal in the same group, allowing a single bit to corrupt the routes of both signals at the same time. This type of SPF is referred to as “routing isolation” failure.

Another obscure failure mode in the Virtex-4 occurs in the IOB. Through previous fault injection experiments, a few configuration bits have been discovered that cause the pair of signals in an IOB to corrupt at the same time. For differential I/O standards, a workaround to this problem has not been found. For I/O standards that allow triplicating pins, care must be used to separate the copies of the same signal on different IOB tiles.

4.7 Partial TMR

Using a best effort approach to TMR, many designs will apply TMR where possible and forego where impractical, a technique dubbed “partial TMR.” The parts
of the design without TMR introduce many SPF. Determining and predicting how many is an area of active research [19]. Other efforts [20] have applied partial TMR on a prioritized basis which is especially useful when the fully TMR design will not fit in the FPGA.

4.8 High Costs

All engineering projects are subject to the typical business constraints, namely performance, cost, and schedule. While the Virtex-4 provides many advantages for performance and schedule, the fault analysis requirements can add significant costs. Terrestrial radiation testing using ion beam accelerators are very costly. Fault injection can help, but if a fault injection hardware platform is not already available, the development can be a prohibitive cost. New formal and static analysis tools show promise, but have not reached maturity and can come with restrictive licensing.

4.9 System Specification

In many systems, the performance is known to have some random errors. Provided the random errors remain below an established threshold, the effectiveness of the system can be maintained. The system error rate specifies what threshold is tolerable, and each component in the system is budgeted a portion.

Some applications have found that the errors induced by radiation will fit within the budgeted error rate [21]. A method to estimate the system error rate starts with finding a raw upset rate that overwhelms the TMR protection. See Figure 4.1. At this raw upset rate, the system error rate is saturated. Following the analytical approach suggested by Edmonds [22] and demonstrated by Allen [6], as the raw upset rate de-
creases, the system error rate decreases two times faster for a TMR mitigated system than for an unmitigated system. The change in slope can be understood intuitively because in a TMR system, a system error can only occur if two upsets occur within a scrub cycle, where an unprotected system can have a system error for every upset.

Each SPF decreases the slope of the Edmonds curve. The allowable SPF for the design occurs is determined by the intersection of the Edmonds curve with the specified system error rate.

4.10 Single Point of Failure Estimation

The focus of this research is to provide a method to estimate SPF for occasions where fault injection and fault analysis tools are not available. Using example designs
and the patterns observed, a framework of rules is proposed to aid the FPGA designer to estimate the number of SPF. Using these rules, tradeoffs can be evaluated and the allowable SPF can be budgeted.
Chapter 5

Designs

The strategy was to identify patterns in the number of SPF as basic circuit structures from common designs were varied. Three different designs were implemented each highlighting a different logic component: registers, muxes, or adders.

For each design type, the DUT FPGA was created within the limitations of the fault injection test hardware, utilizing parameters to allow the scaling of the designs, and interfaces were designed to detect SPF externally. A second complementary test design was also built for the FuncMon FPGA to externally apply test patterns to the DUT and monitor the outputs for the correct result.

The test patterns were designed to use a minimal set yet still provide complete fault coverage to detect all upsets. The DUT and FuncMon designs were proved in simulation for functional correctness first without TMR. Then the designs were synthesized and run through place and route. Timing was analyzed to confirm the tests would be functional in hardware, and then the designs were verified in hardware.

After proof of concept without TMR had been established, the DUT design was triplicated using the Xilinx TMR Tool. TMR Tool was run after synthesis, but before place and route to avoid any synthesis optimizations that undo the TMR. Timing was again confirmed and the test was verified in the hardware. At this point, fault injection was run on the design and the results were recorded.
The DUT design parameters were varied to scale the size of the design, and the entire process was repeated to obtain comparable fault injection results. From the results, the data was analyzed for patterns to estimate SPF based on design features and characteristics.

5.1 Register Table Design

The first design shown in Figure 5.1 was a large register table implemented in the DUT FPGA. Reads and writes to the register table were controlled by a read port and a write port. The read port consisted of an input read address and an output read data bus. The write port consisted of an input clock, an input write address, an input write data bus, and an input write enable.

To read from the register table, a read address was driven. The read address selected which register table value was output on the read data bus. No clock was used for the read bus.
To write to the register table, the write address, data and enable were driven synchronous to the clock. The write address selected which entry in the table was to be loaded. The write data bus provided the data to load. The write enable signal qualified when the data and address were ready. The clock synchronized the transaction between the DUT and FuncMon.

![Register Table Diagram]

Figure 5.2: Register Table Depth and Width.

As shown by Figure 5.2, the width of the registers and data buses was set to a constant 32, but the depth of the table was varied by the parameter N.

5.1.1 Register Table Design Test Patterns

To detect the SPF in the register table design, three test patterns were applied and read back from the table. The first was a simple incrementing pattern. The width 32 was larger than the depth N, so a unique value was loaded into each register in the table. The incrementing pattern detected upsets in the addresses, the data buses, the
clock, and the enable. Two more patterns, a checker board and its inverse, were also applied to check that each bit in each register could be set to both 1 and 0.

Figure 5.3: Mux Stage Design Top-Level Block Diagram.

5.2 Mux Stage Design

Figure 5.3 shows the second design. It was built using a matrix of muxes arranged in rows and columns or stages. Within a stage, each mux could select any of the inputs to pass to its output. For timing purposes, the output of each mux was registered. The input pins connected to the inputs of the first stage. The outputs of the first stage
connected to the inputs of the second stage. The outputs of the second stage connected to the inputs of the next stage, and so on until the last stage outputs connected to the output pins.

![Stage Selects Decode and Load](image)

Figure 5.4: Mux Select Decoding and Loading at Each Stage.

The number of muxes in the stage matched the number of inputs, which was controlled by the parameter N. The parameter P determined the number of stages. The selects for each stage of muxes were registered as shown in Figure 5.4. An address bus of width L (where $2^L \geq N$), a select value of width L, and a select enable provided the ability to load the select for each mux. Reset cleared the select registers.

An input data enable qualified when data was sent in a serial stream from the FuncMon to the DUT. The data enable was pipelined at each stage and passed as an
output to indicate when the return data was valid.

5.2.1 Mux Stage Design Test Patterns

To prove there were no upsets in the design, the test had to show each mux could pass each input signal to the output, and the value passed from each input to the output had to transition both to 1 and 0. The test patterns applied a unique sequence of 1’s and 0’s on each input pin of the DUT, called a channel.

For the first sequence, the first mux of the first stage passed the first channel, the second mux passed the second channel, and so on. The remaining stages were configured the same. For the second sequence, an offset was applied so that in each stage the first mux passed the second channel, the second mux passed the third channel, and so on. For each successive sequence, the offset was increased by one until every channel had been passed by every mux.

5.3 Counter Design

The third design built stages of counters arranged in columns (see Figure 5.5 and Figure 5.6). An input address, data bus, and load signal allowed a parallel load of an initial value for each counter individually. A separate input select chose which counter stage to output on the return data bus, and an input enable caused the selected counter to increment. Reset cleared the output select registers and all of the counters.

The width of the counter was set to the constant 32, like the register table for direct comparison. The number of counter stages was determined by the parameter L. Figure 5.7 shows how the 32-bit counters were subdivided into groups of four bits to allow each bit to flip values twice in 16 cycles. Larger groups would make the fault
injection testing prohibitively long.

5.3.1 Counter Design Test Patterns

First, each counter column was initialized to a unique starting number using the input address and data buses. The unique initial value allowed the detection of address and select decoding faults. Then, selecting each counter one at a time, the counter was enabled and driven on the output data bus. The counters were incremented until they rolled over and returned to the starting value, which forced each bit in the counter to change from a 0 to a 1 and back, or vice versa.
Figure 5.6: Individual Four-Bit Counter.

Figure 5.7: Counter Stage Groups.
5.4 Further Modifications

After determining a baseline number of SPF for one of the Mux Stage designs (N=16, P=16), two more sets of experiments were conducted, one to identify the number of SPF due to singular I/O and another to identify the number of SPF due to convergence points such as required by clock domain crossing.

5.4.1 Singular I/O

For the singular I/O experiments, an input or output was selected to purposefully create a singular point in the design. Each configuration bit affecting the I/O became a potential SPF. The number of singular I/O in the Mux Stage design was varied between 1 and 16, sampling several data points in between.

5.4.2 Convergence Points

The experiments for the convergence points were performed by using TMR Tool to converge, or vote, the value in front of the output register in an individual mux. The voter and following register were made singular, which means each configuration bit affecting the logic became potential SPF. The number of convergence points was varied from 1 to 256, sampling several data points in between.
Chapter 6

Analysis

Using fault injection, the number of SPFs for various sizes in each design group were measured. The data analysis revealed patterns in the relationship between the number of SPF versus the size of the designs. The patterns analyzed include monolithic scaling versus building block scaling, and throughput logic versus feedback logic.

6.1 Monolithic Versus Building Block

One of the key features in the design of the register table was how it grew in size as a single unit as the parameter L increased. Each register was 32 bits wide. As each register was added to the table, the fanout of the read address, write address, data, and enable increased. The fanin to the output mux for the read data bus also increased.

The Mux Stage design used muxes as building blocks to create rows in a stage, and then used columns of stages to scale the size of the design. As the number of rows in a stage increased, the size of the muxes increased and the fanout of the input signals increased, but as the number of stages increased, the fanout did not change. Each stage had the same number of inputs and outputs regardless of how many stages were added.

The Counter designs used columns of 32-bit counters, and each counter was individually loaded and enabled. As the number of columns increased, the fanout of the input data, address, and load increased, and the fanin to the output mux for the output
data bus increased.

The Counter designs and Register Table designs were very similar in the “monolithic” feature. The main difference in the two designs became apparent when comparing throughput versus feedback logic styles.

6.2 Throughput Versus Feedback

The Register Table designs received data from the write port, and passed data forward to the read port. No feedback of the current value was necessary, and the design was entirely throughput logic. The Mux Stage designs also received data from the previous stage (or input data bus) and forwarded data to the next stage (or output data bus). The data did not feedback, and the design only contained throughput logic.

![Register Table and Mux Table Designs: Single Points of Failure vs Slices](image)

Figure 6.1: The number of SPF in Register Table designs and Mux Stage designs versus slice resource usage. Little to no dependence on size is observed.
Counters inherently feed the current state back to increment to the next state. The Counter designs exhibited feedback logic, in contrast to the Register Table and Mux Stage designs. An easy way to confirm the characterization of logic style was to count the number of majority voters in the design. The Register Table designs had no voters, and the Mux Stage designs also had no voters, regardless of the size of the design. However, the Counter design had a voter for every bit in each counter. Figure 6.1 plots the data for the number of SPF versus slices for the Register Table and Mux Stage designs in two separate curves. Figure 6.2 plots the data for the number of SPF versus slices for the Counter designs.

By comparing the three curves, it became apparent that identifying monolithic versus building block types was not helpful. The data for one monolithic design, the
Figure 6.3: The number of SPF in Counter designs versus voters resource usage. A quadratic increase in SPF as the design size increases is observed.

Register Table, showed no dependence on the number of SPF versus the size of the design, while the data for the Counter designs, also monolithic, showed a strong dependence of the number of SPF versus the size of the design. Note that the data for the Mux Stage designs, which were building block, also showed very little dependence on the number of SPF versus size.

In contrast, the presence of voters, which distinguished feedback logic versus throughput logic, was a key feature in the relationship between the number of SPF and the size of the design. Both of the throughput designs showed little to no dependence on the size of the design, but the feedback design showed a very strong dependence. As shown in Figure 6.3, the number of SPF increased quadratically as the design size increased. Significantly, a similar quadratic relationship occurred between the SPF and
the number of voters, $v$, with curve fit Equation 6.1.

$$SPF_v = 4.0 \times 10^{-6} v^2 + 0.0297 v + 28 \quad (6.1)$$

Indeed, the presence of voters triggered the “routing isolation” failures to occur. Note that SPF were apparent even under low slice utilization (<7% of available slices). Also note, even at very high slice utilization in the Mux Stage design (>80%), without voters, almost no routing isolation failures occurred.

6.3 Related Logic

While not conclusive, there are two hints to give insight to why the voters triggered the high number of SPF. The first is a note found in the report generated by the MAP. An example report is shown in Table 6.1. The second hint is found in the Development System Reference Guide [23] in the chapter for the MAP tool under the description for the -c (Pack CLBs) option. In the notes of the report it explains

<table>
<thead>
<tr>
<th>Logic Distribution:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of occupied Slices:</td>
<td>11,374 out of 33,088 34%</td>
</tr>
<tr>
<td>Number of Slices containing only related logic:</td>
<td>11,374 out of 11,374 100%</td>
</tr>
<tr>
<td>Number of Slices containing unrelated logic:</td>
<td>0 out of 11,374 0%</td>
</tr>
</tbody>
</table>

*See NOTES below for an explanation of the effects of unrelated logic

that related logic is “logic that shares connectivity - e.g. two LUTs are related if they share common inputs.” If logic is unrelated, the MAP tool automatically keeps the logic in separate slices. The -c option description further clarifies that only related logic is packed in the same CLB by default.
It appears the presence of the voter caused the MAP tool to relate the logic from the three TMR domains, while without the voter the logic from the three TMR domains were not related. When logic was not related, they were packed in separate CLBs, which maintained routing isolation.

The paper [24] recommends Xilinx incorporate “reliability aware placement and routing” algorithms in their tools. It may not be too difficult to attach properties to voter circuits to help maintain the three TMR domains “unrelatedness,” which may naturally apply routing isolation within the Xilinx Place and Route tool.

In the meantime, the relationship in Equation 6.1 can guide the FPGA designer in applying the trade-offs available for a given system failure rate.

6.4 Dependence on I/Os

The Mux Stage designs varied the number of inputs and outputs as the parameters scaled. While the “routing isolation” failures did not appear in any experiments, it did appear the number of SPF increased slightly as the number of I/Os increased, shown in Figure 6.4. The data samples were limited due to the fix placement of the test hardware, but between 81 total I/Os to 135 I/Os a rise of 1 SPF per 6 I/Os was observed. The Heldt counter design highlighted in [6] used the same test hardware and found no I/O related failures. Comparing the Heldt counter design with the counter design in this work showed one major difference, the use of an input bus. The Heldt counter used only reset to initialize the counters and a few enables to run the counters. The rest of the available I/Os drove outputs. The counter in this effort used a 32-bit input bus to load the counters, which suggests the use of pins as inputs may be more sensitive to the I/O related upsets than the use of pins as outputs.
A request for analysis from the Xilinx support staff was able to determine the signals affected by the SPF identified by the fault injection. The signals were indeed only inputs to the design, which confirmed the observation comparing the two designs. Future work may shed more light on the nature of these SPF.

### 6.5 Results for Singular I/Os

The results for the experiments incorporating singular I/Os were plotted in Figure 6.5. The pattern suggests a varying number of SPF per I/O, and a diminishing marginal effect as more I/Os were singularized at the same time. Previous work [5] reported for the earlier generation Virtex FPGA that 324 configuration bits controlled the function of each IOB. The number of bits controlling an individual IOB in the Virtex-4 is not available publicly at this time. The results presented here indicate on average
only 30–40 bits can functionally upset the IOB in an observable manner. A weak linear fit of the data is presented in Equation 6.2.

\[
SPF_i = 32.087i + 28
\]  

(6.2)

where \(i\) is the number of singular I/Os. Again, the accuracy of these results is somewhat limited due to the hardware available for testing. There may be more to learn as each IOB is exercised in its various modes of operation. Still, as a means of estimating failures, the rule established here may be good enough to allow the FPGA designer to make rational trade-offs until more data is obtained.

### 6.6 Results for Convergence Points

Figure 6.6 shows the linear relationship between SPF and convergence points.
Each voter and register pair appear to independently increase the number of SPF as indicated in Equation 6.3. The results suggest each convergence point, \( c \), contributed 64 SPF to the design.

\[
SPF_c = 63.456 c + 28
\]  

(6.3)

### 6.7 Comparative Impact

It is not hard to see that the voter-induced “routing isolation” failures have the potential to expose the greatest number of SPF for a common design. Table 6.2 summarizes some estimates for SPF versus voters for several orders of magnitude, normalized to remove the common triplicated I/O (y-intercept) contribution of 28. A design with 100 voters would be expected to contribute a mere 3 SPF. At 1000 voters, there are still only about 34 SPF, which would be about the same as 250 triplicated I/Os, about
Table 6.2: A summary of SPF versus voters showing rules of thumb over several orders of magnitude

<table>
<thead>
<tr>
<th>Voters</th>
<th>SPF&lt;v&gt;</th>
<th>Single I/Os</th>
<th>SPF&lt;i&gt;</th>
<th>Convergence Points</th>
<th>SPF&lt;c&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1000</td>
<td>33.7</td>
<td>1</td>
<td>32.1</td>
<td>1</td>
<td>63.5</td>
</tr>
<tr>
<td>10000</td>
<td>697</td>
<td>22</td>
<td>706</td>
<td>11</td>
<td>698</td>
</tr>
<tr>
<td>20000</td>
<td>2194</td>
<td>69</td>
<td>2214</td>
<td>35</td>
<td>2220</td>
</tr>
<tr>
<td>30000</td>
<td>4519</td>
<td>140</td>
<td>4520</td>
<td>71</td>
<td>4533</td>
</tr>
</tbody>
</table>

the same as one singular I/O and half as much as one convergence point. At an order of magnitude higher, 10000 voters contribute as much as 22 singular I/Os and 11 convergence points.

Since state machines, counters, and other common processing algorithms incorporate feedback logic, it is not unrealistic to expect the number of voters to reach in the ten thousands. It is also realistic to expect some wide differential buses that require 50-100 singular I/Os. Convergence points, such as clock domain crossings, are less likely to reach high numbers, especially if using proper techniques to transfer buses.
Chapter 7

Summary and conclusion

The three designs described above and the results obtained by fault injection have successfully facilitated the development of a framework of rules to estimate the number of SPF in common designs. The results showed that the presence of voters was a key feature to the exposure of intrinsic “routing isolation” failures that have previously been reported.

Common physical design requirements such as high-speed interfaces may result in the use of many singular I/Os. Singular I/Os seem to have demonstrated a diminishing contribution as the number increased, but no limiting factor or saturation effect has been established. As this research was limited to available test hardware, future work may want to focus more attention on all modes of the IOBs.

Convergence points can also contribute many SPF in large numbers. However, many designs can avoid the frequent use of them. It should be noted if the convergence points interface to unmitigated logic, the exposure of the convergence logic will not be significant by comparison.

While the focus of this research in regards to the “routing isolation” issues has been to evaluate the straight-forward application of the Xilinx MAP and Place and Route tools, special efforts may be possible to reduce the “routing isolation” exposure through floorplanning or more advanced place and route algorithms.
It has been shown that a predominantly throughput logic design would perform well using the normal process, whereas a feedback intensive design would need extra attention. By applying the rules presented, an FPGA designer can evaluate the relative strengths and weaknesses with regard to reliability at an earlier point in the development process. The valuable insight has the opportunity to help project planning, reduce cost and improve schedule estimates.
Bibliography


