2006-12-06

Using Duplication with Compare for On-line Error Detection in FPGA-based Designs

Daniel L. McMurtrey
Brigham Young University - Provo

USING DUPLICATION WITH COMPARE FOR ON-LINE ERROR DETECTION IN FPGA-BASED DESIGNS

by

Daniel Lee McMurtrey

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Electrical and Computer Engineering

Brigham Young University

December 2006

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Daniel Lee McMurtrey

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date	Michael J. Wirthlin, Chair
Date	Brent E. Nelson
Date	Doran K. Wilde

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of Daniel Lee McMurtrey in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date

Michael J. Wirthlin
Chair, Graduate Committee

Accepted for the Department

Michael A. Jensen
Department Chair

Accepted for the College

Alan R. Parkinson
Dean, Ira A. Fulton College of
Engineering and Technology

ABSTRACT

USING DUPLICATION WITH COMPARE FOR ON-LINE

ERROR DETECTION IN FPGA-BASED DESIGNS

Daniel Lee McMurtrey

Department of Electrical and Computer Engineering

Master of Science

Space destined FPGA-based systems must employ redundancy techniques to account for the effects of upsets caused by radiated environments. Error detection techniques can be used to alert external systems to the presence of these upsets. Readback with compare is an error detection technique commonly employed in FPGA-based designs. This work introduces duplication with compare (DWC) as an automated on-line error detection technique that can be used as an alternative to readback with compare. This work also introduces a set of metrics that is used to quantify the effectiveness and coverage of this error detection technique. A tool is presented that automatically inserts duplication with compare into a user's design. Duplication with compare is shown to correctly detect over 99.9% of errors caused by configuration upsets at a hardware cost of approximately $2X$. System designers can apply duplication with compare to designs using this tool to increase the reliability and availability of their systems while minimizing resource usage and power.

ACKNOWLEDGMENTS

I would like to thank first and foremost, my wife Laurie for her love and patience during this work. She has been supportive and understanding throughout the long hours. My daughter Megan also has been an inspiration for me to do better and be better. I couldn't have done it without them.

I would also like to thank Dr. Michael Wirthlin, my advisor, for the many long hours he has put in on my behalf. His guidance and direction has helped drive this work and benefited me as a writer. Dr. Doran Wilde and Dr. Brent Nelson, the other members of my graduate committee have also given of their time for my benefit.

A special thanks must be given to my parents, Gary and Sheri, who have pushed me to do my best and pulled me along when I needed help. Thanks to my brothers, Brian and Mike, and sisters, Erin, Kristi, Julie and Kelly who have been a great support. Family support is a wonderful thing.

I would like to thank by name Scott Morris, Paul Schumacher, Keith Morgan, Brian Pratt, Nathan Rollins, Welson Sun, and James Carroll. They have all contributed in some part to this work through suggestions and observations and to my development as an engineer.

And finally, a special thanks is deserved by Los Alamos National Laboratory. This work was supported by the Department of Energy, Los Alamos National Laboratory under contract number 3862500102-97.

Thank you.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

There is great interest in using Field Programmable Gate Arrays (FPGAs) in systems intended to operate in unreliable environments such as space. This interest in FPGAs has increased due to the advantages of programmability, flexibility, and high-performance capabilities. Space-destined systems such as the Mars Rover [2], a reconfigurable radio [3], and a DSP satellite [4] all employ FPGAs. Many other space-based systems utilizing FPGAs are in use or under development.

FPGAs are programmable. The programmable nature of FPGAs leads to decreased time-to-market compared with ASICs. A designer can create, alter, and debug a hardware design within a short period of time. Non-recurring engineering costs are also decreased since programmable FPGAs facilitate prototyping and testing of new designs.

FPGAs are highly flexible. They can be reprogrammed in the field, allowing for post-deployment bug correction and remote upgrades to the design. FPGAs can have different functions swapped in and out in the field which is advantageous since space systems are costly to send up. The same chip can perform a wide array of operations just by changing the device configuration. This can be very beneficial when the design is being sent into space for a long period of time.

In many cases, FPGAs provide higher-performance systems than processors. With embedded memories, programmable logic and user-defined flip-flops, FPGAs allow the user to employ application specific hardware that has high throughput and parallelization. Newer FPGAs contain custom multiplier blocks and even embedded microprocessors. This makes FPGAs a good option for computationally intensive operations common to signal processing or communication algorithms.

1.1 Radiation Effects

SRAM-based FPGAs, however, are sensitive to radiation-induced single event effects (SEEs) [5]. These events are caused by charged particles (protons and heavy ions) passing through and transferring charge within the device. The SEE of most concern to FPGAs, is the single-event upset (SEU) [6, 7]. A single-event upset is the change in state of a digital memory element caused by a radiated particle [8]. In an FPGA, these upsets can occur in the user-defined flip-flops, user-defined block memory, and the configuration memory of the chip. Upsets in the configuration memory are the most critical because these upsets can alter the functionality of the system and the configuration memory is the largest portion of memory on the chip (97% on the Virtex XCV1000 [8]).

Many techniques or combinations of techniques have been employed to mitigate the effects of SEUs on SRAM-based FPGAs. One option is to alter the designs to mitigate the effects of SEUs. Mitigation techniques generally involve using some form of redundancy to protect the integrity of the design. The disadvantage of redundancy is the increased resources that are needed. A second option is to employ readback with compare in conjunction with "scrubbing". Readback with compare will detect any errors in the configuration memory by reading in the bitstream and comparing with a "golden" copy. Scrubbing can then be employed to restore these upsets [9]. These options are not mutually exclusive and are often employed together. Certain combinations of scrubbing and redundancy have been shown to be effective in increasing the reliability of space-destined systems [10].

Readback with compare has disadvantages as an error detection technique. The first disadvantage is that readback with compare can only detect upsets within the configuration bitstream. The second disadvantage is that readback cannot be applied to designs that contain all primitives. Designs with LUT RAMs for example will not work properly with readback employed.

Error detection can be employed to increase the ability of systems to tolerate faults. Error detection circuitry requires less hardware than correction circuitry. Systems that can tolerate temporary interruptions of correct operation could employ

detection. These brief service interruptions would be recognized and the outputs could be discarded. Error detection is intended for systems that can tolerate these temporary deviations from normal operation as long as this deviation is detected. Such systems can be repaired and return to expected operation with a configuration memory refresh or system resets on a demand-driven basis.

Duplication with compare (DWC) is an alternative error detection technique to readback with compare. DWC is a simple hardware redundant error detection technique that can detect upsets in the configuration bitstream and in user memory. DWC has a very simple implementation and can detect most errors. Because of its simple implementation DWC is able to be automated as part of a CAD tool. DWC also provides great control over resource usage to the user and can correctly detect a high percentage of upsets. The user can modify DWC to fit his/her needs.

This thesis will demonstrate that duplication with compare can be implemented as an automated FPGA-based concurrent error-detection scheme to detect the majority of SEUs with limited additional area costs. This thesis will show that using DWC, a system can implement on-line error detection that is able to detect the majority of SEUs. This work will also show that the process of augmenting a design with DWC can be automated.

1.2 Thesis Organization

This thesis will adopt the following methodology to explore and demonstrate the effectiveness of DWC as an automated on-line error detecting tool. Chapter 2 will begin by describing the benefis of error detection in digital systems. The chapter will make comparisons of the different detection techniques, examining the advantages and disadvantages to later demonstrate the effectiveness of our chosen technique. Next, readback with compare will be presented. The technique proposed in this work is an alternative to readback. DWC will be introduced in the conclusion of this chapter.

In Chapter 3, a fault injection simulator will be introduced. The simulator is used to accurately measure the error detection capabilities of test designs. It will be used throughout this work to measure the quality of DWC.

Chapter 4 will present an approach for implementing DWC. After, the relationships between the chosen event metrics and DWC is examined. This will contribute to a better understanding of the challenges of implementing DWC in an FPGA. These challenges will be the focus of the next discussion.

After introducing DWC, the discussion will move on to the focus of this work: the automated tool to implement DWC in a design. Chapter 5 will begin with a detailed explanation of the flow of the automated tool. The results that were obtained when a set of benchmark designs were run through the tool will be presented. The results will show the effectiveness of the automated DWC tool. In Chapter 6, the work will be summarized, appropriate conclusions drawn, and some future work suggested.

This work will demonstrate that DWC can be implemented by an automated CAD tool to provide effective detection of SEUs in FPGA-based designs. This tool can be used as an alternative to readback with compare in FPGA-based designs that are destined for an unreliable environment. This work was motivated by the question: How good of error detection can be obtained in an automated tool? This work will show that good error detection can be achieved through the use of the tool created in conjunction with this work.

Chapter 2

Concurrent Error Detection

Many digital systems can operate correctly in the face of upsets provided there is sufficient detection of errors. Error detection techniques can be used to alert these systems when a deviation from correct operation is occurring. High reliability systems can respond to such alerts in a variety of ways. Systems can use detection to signal the need for correction or to signal that the data is invalid and should be discarded.

FPGAs operating in a space environment require some form of error detection. In FPGA-based designs, error detection techniques should be able to detect upsets in *all* parts of the device. This includes user memories, user-defined flip-flops, and especially the configuration memory that defines the logic, routing, and I/O operation. Upsets within the configuration memory can be difficult to detect since they can potentially alter the very behavior of a system.

The purpose of this chapter is to introduce error detection. First, this chapter will introduce the concept of error detection within FPGA circuits. Second, this chapter will discuss readback which is the most common technique for error detection with FPGAs. Finally, this chapter will introduce an error detection technique that addresses a number of the disadvantages associated with readback.

2.1 Error Detection in Digital Systems

Error correcting techniques can be applied to any system to increase the reliability. These techniques correct the effects of upsets ensuring the systems will perform as expected. It has been shown that error correction techniques can be used to improve performance in the presence of single upsets [11]. Systems that need high availability of outputs can employ these techniques to increase the fault tolerance

capacity of the system [12, 10]. This allows these systems to operate successfully in harsh environments. The major drawback of error correction techniques is the additional resources that must be utilized. Error correcting techniques always require some form of redundancy. This redundancy comes at the cost of additional area or timing. As an example, when triple modular redundancy, a well-known error correcting technique, is applied the system shows a $3 - 6X$ increase in area [13].

Error detection techniques are also widely employed in the computer world to increase the ability of systems to tolerate upsets [14, 15]. With error detection, the goal is to recognize and report errors, rather than to mitigate these errors. The goal of error detection is to detect when the system is in an erroneous or failed state. Systems with error detection may signal to any system that interacts with or depends on them that the system is in error. External systems can then take steps to correct the fault such as scrubbing or a system reset or simply ignore the invalid outputs.

Many systems employ error detection as a step in a process to increase reliability. Computer memory systems often use parity bits to detect errors. For example, the data in the instruction cache of many microprocessors contain parity bits [16]. If an error is detected the correct instruction can be fetched from main memory instead of the I-cache. Communication systems also employ error detection. The TCP/IP networking protocol stack performs error detection at multiple levels. Most notable are the TCP layer and the Ethernet layer which both perform error detecting checksums [17]. If an error is detected the packet is discarded and will eventually be resent. All communication systems also perform error detection to ensure the reliability of the data being transmitted.

The detection of an error can invoke many different reactions from external systems. Some systems use error detection as an indication to invoke corrective measures. For such systems some sort of recovery technique must be initiated when an "error detected" signal is received. This recovery will return the system to its correctly operating state. A full system reset is a good example of a potential system recovery technique. For other systems, having erroneous outputs does not pose a problem as long as the occurrence is a known event. In this case, the outputs are

usually discarded or marked as invalid. The system would then carry on its normal operation or retry.

2.2 Error Detection Strategies

Error detection is accomplished through the use of redundancy. Redundancy involves the use of additional resources to detect when an error occurs. Without some form of redundancy, error detection is not possible. This section will summarize the most common three broad categories of error detection. These include temporal redundancy, information redundancy, and hardware redundancy.

Temporal redundancy uses additional time (clock cycles) as redundancy to perform error detection. Time redundant techniques use less additional hardware than other on-line detection techniques at the expense of latency. This class of techniques conserve area while increasing the latency of the system. The general structure of these techniques can be seen in Figure 2.1. The operation is first performed normally during t_0. The first output is stored and the operation is then performed a second time during $t_0 + 1$. Only this second time the inputs may have been subjected to an encoding scheme. The encoding scheme is necessary to allow the system to detect permanent errors. Once the re-computation is completed, the output is decoded and compared to the original. If they differ, an error flag is raised signaling that the output is incorrect. The encoding and decoding of the second operation allows the system to detect both transient and permanent faults.

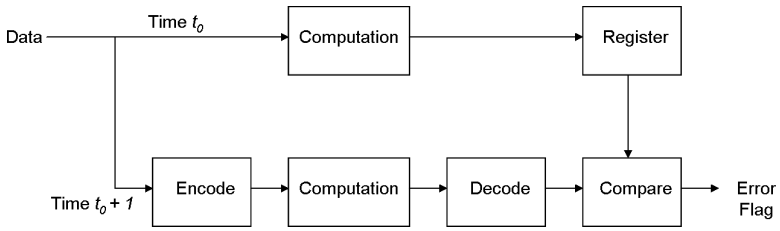


Figure 2.1: Flow chart of temporal redundancy operation [1].

Information redundancy adds information bits to the data to allow the system to perform fault detection. This additional information is generally in the form of codewords used to represent the data. The most common example is a parity bit. The parity can be recalculated and compared with the stored original parity bit to certify the data has not been corrupted. Another common example of information redundancy can be seen in state machine encoding. The state variable is encoded into codewords that can be checked for validity [18]. These codewords are chosen as a subset of a much larger set of codewords. If a codeword appears that is not part of the valid subset of codewords the error flag is set.

Hardware redundancy techniques involve additional hardware components that are used to detect errors caused by SEUs. Additional circuitry is added and the outputs of this redundant circuit are compared with the original circuit to determine if an error has occurred. This additional hardware is usually in a similar form as the original. For example, hardware redundancy could be applied to a multiplier system in the form a second multiplier with reduced precision. The outputs could then be compared to determine if an error has occurred. Hardware redundant techniques can be applied uniformly throughout a system.

2.3 Error Detection in FPGAs

FPGAs, due to their composition and routing, present specific challenges to error detection schemes. Similar to ASICs, wires and values stored in flip-flops and memories are susceptible to the effects of upsets. Additionally, the configuration memory in FPGAs contains the bits that determine the behavior of the programmable logic and routing. This memory is also susceptible to upsets. These upsets can cause changes in the behavior of the logic, routing, clock, or the I/O of the circuit. This could result in correctly formed logic functions that do not perform the originally intended function of the system. This problem is well understood (see [19, 5]).

The challenge for error detection schemes in FPGAs is to detect when an upset in the configuration bitstream has changed the behavior of the circuit. It is difficult because these changes can be subtle, for example, changing an OR gate to

a XOR gate. The faults may also only manifest as errors sporadically with a given combination of inputs. This section will explore the difficulty that this challenge presents to detection schemes and also examine the motivation of detecting errors.

This difficulty of detecting upsets within the configuration bitstream is a known challenge in FPGAs. Traditional techniques cannot always detect these alterations. For example, traditional temporal redundancy performs an operation twice and then compares the results to determine if an error occurred. This is very effective at determining if a fault occurred in any of the wires, signals or states as the operation was being performed. However, if an upset altered the actual logic this method would be unsuccessful at detecting the error unless an encoding scheme was used. However, encoding schemes are application specific, for example, a scheme that works for addition will not work for multiplication. This makes error detection challenging.

2.4 Readback

The most common technique for SEU detection is readback with compare. An examination of readback with compare is necessary because it is an alternative to the strategy that is being proposed in this work. Readback with compare is an error detection strategy that can be employed to detect errors in the configuration bitstream. This section will examine readback with compare as an FPGA-based error detection scheme. First, the purpose of readback with compare will be presented. Then, the implementation of readback with compare will be discussed along with "scrubbing" which is readback with compare paired with error correction. Finally, this section will present the limitations of readback as an error detection technique.

Readback is a configuration bitstream management technique that allows external systems access to a device's configuration memory. Readback is designed to give external systems the ability to read and/or write to a devices configuration memory. This can be used to support partial reconfiguration of an FPGA. It is also used during readback with compare.

Readback with compare is an FPGA-specific off-line error detection technique [9]. Figure 2.2 shows how readback with compare is performed on a system. Readback with compare is a two part process. During the first part the data stored in the configuration bitstream of the system is read in by the external system. The second part, is to compare the data with a "golden" copy of the configuration bitstream stored on the external system to determine if any upsets have occurred. This comparison is often simplified by using a checksum [9]. Any differences will cause the error flag to be raised.

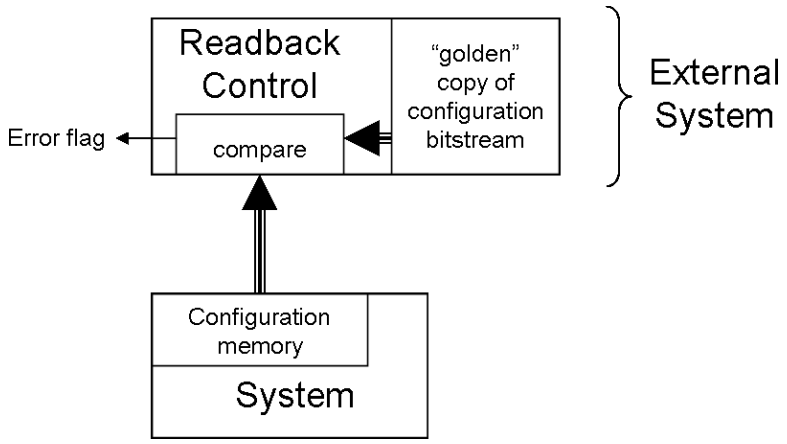


Figure 2.2: System implementing readback for error detection

Readback with compare is often paired with error correction. Once readback with compare is used to detect the error, the error can then be corrected. Only the affected data frame is restored and written to the system. This readback with compare and then correction is what is commonly referred to as "scrubbing". Once "scrubbing" has occurred, upsets in the configuration bitstream will have been restored to there original values. The system will then perform in the originally intended manner.

Although readback with compare successfully detects errors in the configuration bitstream, it does have several limitations. These limitations include increased time to detection of an error, inability to detect upsets within user defined memory

elements, incompatibility with systems that use LUT RAM cells, and increased external system overhead to perform readback with compare. Each limitation will be discussed below.

First, detection time for readback is relatively long. The time it takes for this technique to detect an error is limited by the cycle time it takes to perform readback with compare on the entire configuration memory. For example, if a system can perform full readback with compare every $2\mu s$ than the maximum time it could take to detect an error is just under $2\mu s$. The error could occur on a configuration bit just after it was checked. It would then not be detected until the entire readback with compare cycle was performed and that bit was checked again. The time to detection could be reduced by performing a full readback with compare more often, however this uses more compute power of the external system and there are physical limitations to how fast this readback with compare can be performed.

The second limitation of readback with compare is that it does not detect all errors. Readback with compare is very successful at detecting upsets that occur in the configuration bitstream. However, upsets that affect the values stored in dynamic memories (RAMs, flip-flops, or block RAMs) are not detected. Readback with compare will fail to detect upsets that cause errors in the values stored in these memory elements.

The third limitation of readback with compare is that it prevents the use of certain primitives. Primitives that involve using a LUT as a RAM memory element are incompatible with readback during run-time. Examples of such primitives include SRL16s and LUT RAMs. Any design containing these primitives cannot have error detection applied to it using the readback with compare technique.

The final limitation of readback is the need for significant external circuits. Extra memory and an extra running process are needed to implement readback with compare (see Figure2.2). The external system must have enough fault-tolerant memory to store a "golden" copy of the configuration bitstream. The external system will also have to perform the readback and compare operations. If this does not occur

on dedicated hardware it can be costly to the other processes that are sharing the hardware.

These limitations to readback with compare suggest that other forms of error detection are needed. Error detection techniques are needed that can operate on systems containing LUT RAM primitives. Also error detection techniques are needed that can detect upsets in both the configuration bitstream and in the user-defined memory are needed. Such a technique would provide a good alternative to readback with compare.

2.5 Duplication with Compare

The purpose of this work is to introduce an on-line FPGA-based error detection strategy that can be used as an alternative to readback. The hardware redundant error detection technique called duplication with compare (DWC) was chosen for this work. DWC is the most common and simple way of performing error detection with hardware redundancy [1]. It can be applied to detect all errors including transient, intermittent, and permanent.

The basic structure of DWC is shown in Figure 2.3. DWC has a very simple implementation. The full original circuit is completely duplicated to form two identical modules. Comparator circuitry is then added to the design. This circuitry detects any difference in the outputs of the two modules. If a difference is detected an error flag is set to indicate that the modules are in disagreement.

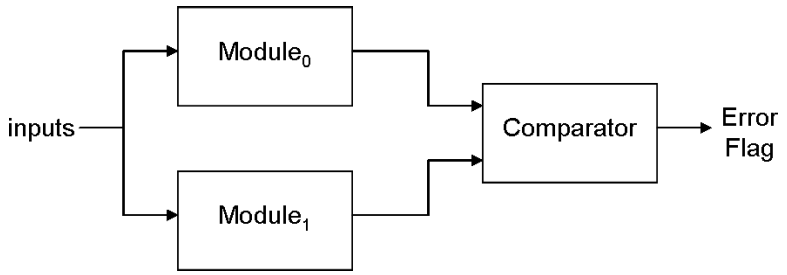


Figure 2.3: Duplication with compare

DWC was chosen for this work because it is easy to apply automatically with a CAD tool. Because DWC does not use encodings schemes that are dependent on the operation being performed, it can be uniformly applied to any operation: sequential, arithmetic, and bit-level. This makes DWC an ideal candidate for automation which was one of the main points of emphasis for this work.

This work introduces DWC as an alternative to the traditional FPGA-based error detection technique of readback. There are several advantages to using DWC for error detection when compared to readback:

- **Time to detection** - DWC can detect errors much more quickly. Errors are detected as soon as they manifest themselves at the location of the comparators. Comparators could be placed more frequently to further decrease time to detection for DWC.

- **Coverage of errors correctly detected** - Applying DWC to an entire circuit allows the system to detect *both* configuration bit upsets and upsets in user-defined memory elements.

- **Full use of cells** - DWC can be applied to designs that are composed of all types of primitives, including LUT RAM cells.

- **Reduced off-chip overhead** - DWC can be implemented on-chip without any external off-chip memories or hardware.

The primary disadvantage of DWC is the increased on-chip resources that are needed to implement the error detection. Since readback is an off-chip technique it does not require additional on-chip resources. The original design maintains its original resource count. Designs fully enhanced with DWC require at least $2X$ the resources to implement the duplication. Additional resources are also needed to implement the comparator circuitry.

In this chapter, DWC was presented as a good alternative to readback with compare. DWC can be used to overcome the challenges to error detection in FGPAs

which are upsets in the logic and routing. Many systems that need error detection could benefit from using DWC as an error detection technique.

Chapter 3

Evaluating Error Detection Coverage

In the previous chapter, DWC was proposed as an alternative to the traditional FPGA-based error detection method of readback with compare. Systems can benefit by applying error detection techniques to increase the ability of systems to operate in the presence of upsets. A method to measure the benefits of FPGA-based error detection schemes such as DWC is needed to quantitatively determine the effectiveness of an error detection scheme.

Traditional methods to measure the effectiveness of error detection schemes involve simulations using a stuck-at fault model [20]. Simulation, based on the stuck-at fault model, is very effective for determining the reliability of ASICs in the presence of SEUs [21]. However, FPGAs introduce the problem of representing upsets in the configuration bitstream that cannot be effectively modeled with this type of fault model. Upsets in the routing and programmable logic do not merely cause an internal bit to be forced to either a one or zero. These upsets can change the functionality of a circuit. For example, an XOR gate could easily become an OR gate. These upsets cannot be analyzed in a stuck-at fault model based simulation.

A fault injection simulator was created for this work to more accurately measure the coverage of error detection. Fault injection enables the full exploration of the effectiveness of an error detection scheme. A fault injection simulator, which was originally designed to measure the sensitivity of designs [22], was adapted to report our chosen error detection events with DWC designs. The purpose of this chapter is twofold: to present the fault injection simulator that is used to analyze an error-detecting design and to present a set of event categories that will be used by this simulator to quantify the effectiveness of an error detection scheme. This simulator

and these error detection event categories are used throughout the remainder of this work to validate the effectiveness of DWC for FPGA-based error detection.

3.1 Hardware Fault Injection Interface

The fault injection simulator that was created consists of the hardware FPGAs and the software. The simulator was based on an old simulator that was developed at BYU to measure the sensitivity of designs. This modified simulator will be used to test designs that have error detection capabilities.

3.1.1 Fault Injection Simulator for Sensitivity

A fault simulator [22] was designed at BYU to measure the sensitivity of SEUs on an FPGA-based design. The simulator will corrupt a configuration bit and then determine if the injected fault at that bit causes an output error by comparing the output of the corrupted design to that of a "golden" copy. The simulator is based on the SLAAC-1V FPGA computing board [23].

As seen in Figure 3.1, this board consists of three Xilinx Virtex XCV1000 chips. The X2 chip contains the "golden" version of the design. X1 is the design under test (DUT) which will be corrupted and tested. X0 is the control logic that runs the system and checks the results of each test. The system is run and controlled through a PCI interface that allows the user to track the results. The basic operation of an injection cycle is:

1. Corrupt a configuration bit in X1,
2. Run the system with a set of random inputs for a set amount of time,
3. Check outputs for errors in X0, and
4. Repair configuration bit.

This cycle is repeated for all the configuration bits that are to be tested. If an output error is detected, the configuration bit that was corrupted is considered sensitive and recorded, since it causes incorrect outputs. The software simulator is able to query

the X0 device between each cycle to determine which bit upsets corrupt the output. The bits are logged and collected to estimate overall sensitivity.

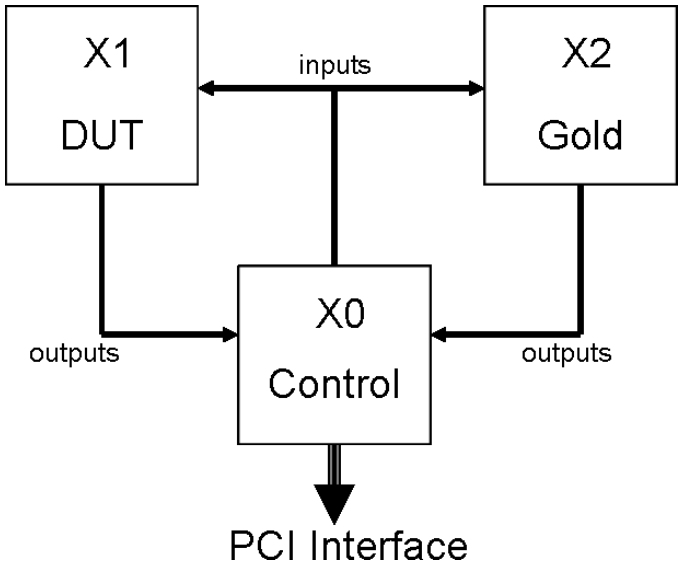


Figure 3.1: Original SLAAC-1V simulator

3.1.2 Modifications

Although the original simulator is very effective in determining the sensitivity of designs, it was not able to provide the results needed to measure the effectiveness of error detection hardware. For this work, the simulator was modified to support error detection in addition to sensitivity. As proposed in the original simulator, X2 is still the "golden" copy of the design and X1 is the "DUT". Again the steps are the same as before: a configuration bit is corrupted, the system is run for the desired time with a set of inputs, X0 reports the state of the two copies of the design, the configuration bit is restored, and the system is reset.

There are two major differences between the error detection simulator and the sensitivity simulator that must be noted. First, duplicated outputs are supported on

the DUT. Second, an internally generated error flag signal is added and routed to X0. The basic alterations to the hardware can be seen in Figure 3.2.

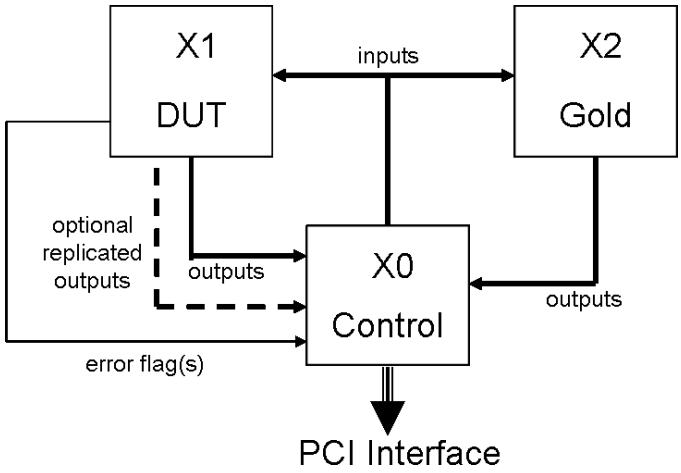


Figure 3.2: Modified simulator hardware

These modifications permit the classification of the state of the DUT into four possible outcomes. The possible outcomes are

- correct operation,
- an output error occurred (EO),
- a functional logic error was detected (EF),
- a detector circuit error was detected (ED).

The EO indicates that the system output is in error. The EF indicates that the error flag from the DUT is signaling the functional logic has been upset. The ED indicates that the error flag is signaling that an upset has occurred in the error detection circuitry. Each occurrence will be discussed in terms of the modifications to the tool that made if possible to detect the occurrence.

The first modification allows for duplicated outputs of the DUT. The hardware was modified so that the duplicated outputs from the DUT design were routed to the

X0 design. Each set of outputs was compared independently against the output of the "golden" design. If either of the outputs was incorrect an output error occurrence (EO) was recorded. In equation form it is

$$EO = (DUT\_OUT_0 \neq GOLD \; or \; DUT\_OUT_1 \neq GOLD). \tag{3.1}$$

With the modified simulator it is still possible to only use a single output from the X1 chip. Using only a single set of outputs changes the way in which EO is recorded. An EO occurs if the single output from the DUT did not match the "golden" output. Only one output needs to be compared in the X0 chip.

The second modification was to add support for the internally generated error flags. This signal is generated by error detection circuitry of the DUT and routed to the X0 chip. The signal can be read to determine if DWC has detected the occurrence of an error.

Table 3.1: Assignment of the occurrence with a 2-bit error code

Error Code	**Type of Occurrence**
"00"	none
"01"	ED
"10"	ED
"11"	EF

The error flag could be either a 1-bit or 2-bit error code signal. The 2-bit signal originates in the detection circuitry of the DUT and indicates the position where the error was detected. Using a 2-bit signal can result in either a EF if the error was detected in the functional logic of the circuit or an ED indicating the error was detected in the detection circuitry of the system. Only with a 2-bit signal can the system indicate where the error is detected. The assignment of EF and ED with a 2-bit error code is as shown in Table 3.1.

Using a 1-bit error code, if the signal is a '1' a functional error has been detected in the logic of the system (EF). An EF means the internal detection circuitry of the DUT has signaled that an error has been detected. A 1-bit signal cannot differentiate the location where the upset has occurred. If the error code is a '0' no EF occurrence has occurred. With a 1-bit signal ED is unused and tied to '0'.

The two modifications change the behavior of the simulator hardware. EO occurrences, EF occurrences, and ED occurrences can be recorded. Table 3.2 summarizes the occurrence and the reason that each would be recorded. The recording of these occurrences will be used in the next section to determine which type of error detection event has occurred.

Table 3.2: List of reasons for each possibly noted occurrence

Occurrence	**Reason**
EO	A difference in the output(s) of the DUT and GOLD was detected by X0
ED	The internal detection circuitry of the DUT has indicated an error was detected in the detection circuitry
EF	The internal detection circuitry of the DUT has indicated an error was detected in the functional circuitry

3.2 Error Detection Events

The effectiveness of an error detection technique is determined by calculating the percentage of actual errors correctly detected by the detection circuitry. This is termed coverage. Coverage will be used in this work to quantitatively determine the effectiveness of DWC as an error detection technique. The goal of this work is to demonstrate an error detection technique that maximizes coverage.

However, to understand the coverage of an error detection technique it is necessary to understand the set of error detection events that can occur. This section

will present the five events that can occur when a configuration bit has been upset. The five events are insignificant, sensitive detected, sensitive undetected, false positive, and checker errors. These five events will then be used to show the formula to determine coverage of an error detection scheme.

The first type of event that could be caused by the upset of a configuration bit is an *insignificant event* (IE). Upset configuration bits that cause insignificant events do not cause an output error and are termed insignificant bits. Insignificant events occur when the upset does not affect the system outputs or the error code. They are represented using the occurrence variables EO, EF and ED (see Table 3.2) by the logic formula,

$$insignificant\ event(IE) = \overline{EO} \cdot \overline{EF} \cdot \overline{ED}. \tag{3.2}$$

Insignificant events indicate that the configuration bit has no noticeable effect on the system. Most IE are caused by configuration bits in unused portions of the chip.

The second type of event is the *sensitive detected event*. A sensitive detected event occurs when an upset configuration bit caused the output to be incorrect and the error detection circuitry to indicate that an error has been detected in the functional logic. This configuration bit is then termed a sensitive detected bit (SD). In equation form the event is shown as,

$$sensitive\ detected\ event(SD) = EO \cdot EF. \tag{3.3}$$

Sensitive detected events indicate that the output is not valid and an error was correctly detected.

The third type of event is the *sensitive undetected event*. A sensitive undetected event is caused by a configuration bit that when upset, causes the outputs to be incorrect while the error detection system continues to indicate proper operation. This configuration bit is termed a sensitive undetected bit (SU). Sensitive undetected events can be shown by the equation,

$$sensitive\ undetected\ event(SU) = EO \cdot \overline{EF}. \tag{3.4}$$

Sensitive undetected events occur when the detection circuitry does not correctly detect an error.

The fourth type of event is the *false positive event*. A false positive event occurs when an upset causes the error detection system to indicate that an error has occurred in the functional circuit, while the output maintains the correct value. The configuration bit that causes a false positive event is termed a false positive bit (FP). False positives events are given by the equation,

$$false\ positive\ event(FP) = \overline{EO} \cdot EF. \tag{3.5}$$

False positives falsely indicate that the outputs are incorrect.

The final type of event is a *checker error event*. Checker error events occur when a system is using a 2-bit error code and the upset causes only one of the error code bits to be incorrect. These events are caused by configuration bits in the detection circuitry that are termed checker error bits (CE). This event is represented by the equation,

$$checker\ error\ event(CE) = \overline{EO} \cdot ED. \tag{3.6}$$

Checkers errors indicate the detection circuit needs to be corrected but the regular circuitry is functioning properly. The system outputs are valid and can be trusted.

Table 3.3: Summary of events and their formulas

Event	**Logic Formula**
Insignificant	$insignificant = \overline{EO} \cdot \overline{EF} \cdot \overline{ED}$
Sensitive Detected	$SD = EO \cdot EF$
Sensitive Undetected	$SU = EO \cdot \overline{EF}$
False Positive	$FP = \overline{EO} \cdot EF$
Checker Error	$CE = \overline{EO} \cdot ED$

The quantity of configuration bits that cause the events defined here can be used to quantitatively determine the coverage of an error detection scheme. These

events, summarized in Table 3.3, will be used in the proceeding section to calculate metrics that measure the effectiveness of error detecting schemes.

3.2.1 Metrics to Determine the Effectiveness of Error Detection

Two metrics will be used to demonstrate the effectiveness of an error detection technique. The first metric is the percentage of bits that cause significant events that are correctly diagnosed (CDSB). Sensitive undetected events are the only events that are not correctly diagnosed by the error detection system. Faults which cause sensitive undetected events can have very negative effects on system performance. CDSB can be determined by the following equation

$$CDSB(\%) = (1 - \frac{SU}{SU + SD + CE + FP}) * 100. \tag{3.7}$$

This metric is the fraction of bits that do not cause SUs in terms of all the bits that cause variations in the system behavior, whether the variation is in the error detection circuitry (CE and FP) or the regular logic (SD and SU). This metric is beneficial because it is device independent and would be the same if the system was implemented on different devices.

The second metric that will be used to show the effectiveness of error detection schemes is the percentage of configuration bits that are correctly diagnosed by the error detection circuitry (CDCB). This metric can be calculated with the equation

$$CDCB(\%) = (1 - \frac{SU}{total\ device\ configuration\ bits}) * 100. \tag{3.8}$$

This correctly diagnosed configuration bit count shows the percentage of errors correctly diagnosed by the error detection circuitry from the device's perspective. It includes insignificant bits which are equally likely to experience SEUs and are correctly diagnosed by the error detection circuitry. The CDCB is useful because it can be used to calculate the mean time to incorrectly diagnosed event by multiplying the CDCB by the SEU rate for the environment where the system is operating. This would be used as an estimate of how long it will take before the device fails.

Both the percentage of correctly diagnosed significant bits and the percentage of correct diagnosed configuration bits show the effectiveness of error detection techniques. These metrics will be used in conjunction with the defined events to analyze this work to determine if the detection scheme proposed accomplishes the stated objectives.

3.3 Software Simulator

The software component of the fault injection simulator controls the hardware components and classifies the results. Using driver calls, the software is able to read in if the upset of the configuration bit has caused an EO, EF, or ED to occur. With this information the software is able to categorize what type of error detection event has occurred.

Figure 3.3 shows a flow chart of the software event categorization process. The software checks first if a difference in the outputs of the DUT and the "golden" has been detected by X0 (EO). If so, the software next checks if an error flag (EF) has been reported. If so the configuration bit is a sensitive detected bit because it caused a sensitive detected event. If the EF did not occur the bit is termed a sensitive undetected. If no EO was originally reported the software next checks if a EF was reported. If the EF was reported the configuration bit is a false positive bit. If the EF was not reported the software will check if a EC was reported. If an EC was reported the configuration bit is a checker error bit. If no EC was reported the configuration bit is an insignificant bit that causes a insignificant event. It can be seen that all configuration bits fall within one of the five defined categories.

3.4 Summary

A fault injection simulator was created to detect and measure the quality and effectiveness of error detection systems. The events identified by the simulator are insignificant, sensitive detected, sensitive undetected, false positive, and checker error events. These events are used to classify the configuration bits. Each configuration bit when upset by the simulator will result in one of these five event types. The

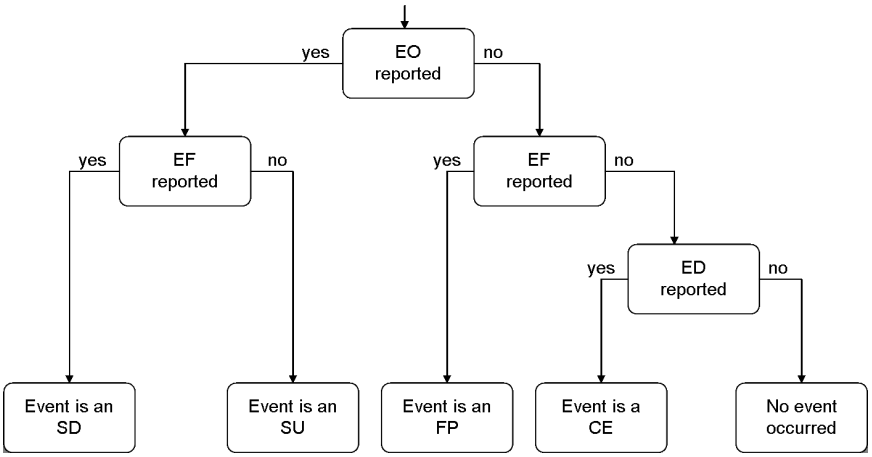


Figure 3.3: Flow chart for software event categorization

configuration bit can then be categorized and recorded as a insignificant bit, sensitive detect bit, sensitive undetected bit, false positive bit, or a checker error bit. For each design the totals of each category of configuration bit is recorded.

Using these totals the effectiveness of the error detection technique can be determined with Equation 3.8 and Equation 3.7. These equations are used to measure of the effectiveness of an error detection scheme at correctly detecting all errors caused by SEUs. For this work several designs on which DWC has been applied will be run through the error detection fault injection simulator described in this chapter. The percentages will then be calculated using the results of the simulator. Percentages that are closer to full coverage (100%) indicate systems with effective error detection. These percentages will be used to determine that DWC is a good error detection scheme that can correctly identify most errors.

Chapter 4

Implementing Duplication with Compare in an FPGA

Duplication with compare provides an error detection scheme that can detect upsets in both the configuration logic and the user defined memory. The purpose of this chapter is to introduce and describe an approach for implementing DWC in an FPGA.

This chapter will begin by summarizing the major steps involved in implementing DWC in an FPGA-based design. The next section will explore the sources of each type of detection error in a DWC-enhanced FPGA design. Finally, results will be presented and examined to determine the effectiveness of the chosen implementation of DWC.

4.1 Implementation Approach for a DWC CAD tool in an FPGA

The end goal of an error detection strategy is to generate a circuit that has been augmented with error detection circuitry. This error detection circuitry will generate a single error code output. This error code will alert external systems to the presence of errors. Four steps are used to implement DWC in FPGA-based designs:

1. Duplication of the circuit,
2. Selecting locations for comparator circuits,
3. Determine the architecture of comparator to insert,
4. Merging the local intermediate error flags of each comparator into a single error code output.

This section will explore each of these steps in greater detail.

4.1.1 Duplication of the Circuit

The first step to implement DWC in an FPGA-based design is to duplicate the original circuit. The design is duplicated to form two identical domains. In its most basic case this is simply a task of adding each component and net two times to a new DWC-enhanced design. It is important, as well, that the connectivity of the original circuit be preserved in the duplicated design. This duplication process is relatively straightforward if the entire design is to be duplicated.

Partial duplication involves duplicating a subset of the overall design. This can be done for several reasons. First, the size of the chip on which the design is to be implemented could impose size restrictions. Second, some designs have partitions that are more important than others (such as feedback). While not the focus of this work, partial duplication with compare presents two major challenges that will not be discussed here: determining which portions to duplicate and connecting duplicated portions to un-duplicated portions.

In order to achieve full coverage of errors correctly detected, a design must be fully duplicated, including input and output buffers. However, duplicating these ports presents two specific challenges to the system. First, I/O buffers are a limited resource. It may necessitate using a larger part to fully duplicate the inputs or outputs and this is not possible in all cases. Second, duplicating the inputs or outputs means that the external system must be capable of dealing with two copies of the signals. In the case of inputs the external systems must drive both sets of inputs and timing must be preserved, meaning the difference in clock skews of the sets of inputs cannot be allowed to cause errors. In the case of outputs, the external system must then handle both pairs of outputs with external error detection. This places a portion of the burden of error detection on external systems.

For this work, the input buffers will not be duplicated. The reason for this, is duplicated inputs are not currently supported in the previously described simulator. The simulator has a limited number of inputs allowed (72) and currently only drives one set of inputs. Modifications could have been made to support duplicated inputs

but were considered outside the goals of this work. In this work, support is included for both duplicated and non-duplicated outputs.

4.1.2 Selecting Locations for Comparator Insertion

The second step in applying DWC is to select net locations where the comparators will be inserted. The selection of locations is best seen with an example. Figure 4.1(a) shows a simple design. Two OR gates, A and B, are driving the inputs of XOR gate C. If the net that connects component B and C was chosen as a location where a comparator would be inserted, the resulting circuitry would look like Figure 4.1(b). The entire circuit has been duplicated into two domains and in both domains the target net is run to a comparator circuit (as well as being run to the original destination). The comparator can then detect errors that are manifested on this net.

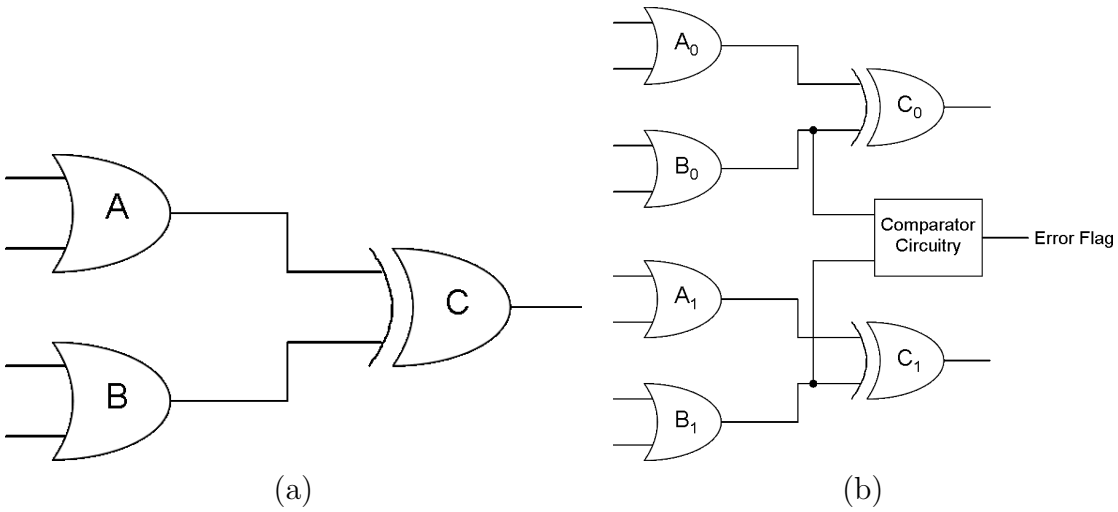


Figure 4.1: (a) Simple design (b) Duplicated simple design with a single comparator added

It is important to note that a system does not have to have comparators inserted. In this case the error detection would be performed off chip. There are several issues that must be understood to determine locations to insert comparators.

This section will explore some of the important issues such as nets that cannot be altered to include comparators, errors in feedback, and time to detection.

There are many nets within FPGA designs that cannot be altered to include comparator circuitry. Generally this is the result of the inaccessibility of the particular signal or net. For example, it would not be efficient to attempt to put a comparator on the net that leads from the 4-input LUT of a slice to the flip-flop of that same slice (see Figure 4.2). If that was done the originally one-slice component would have to be mapped to two different slices to permit access to the desired net. The first slice would contain the logic of the 4-input LUT and the second slice would contain the flip-flop. This is both inefficient in optimal use logic and could potentially alter the timing of the system with the additional routing.

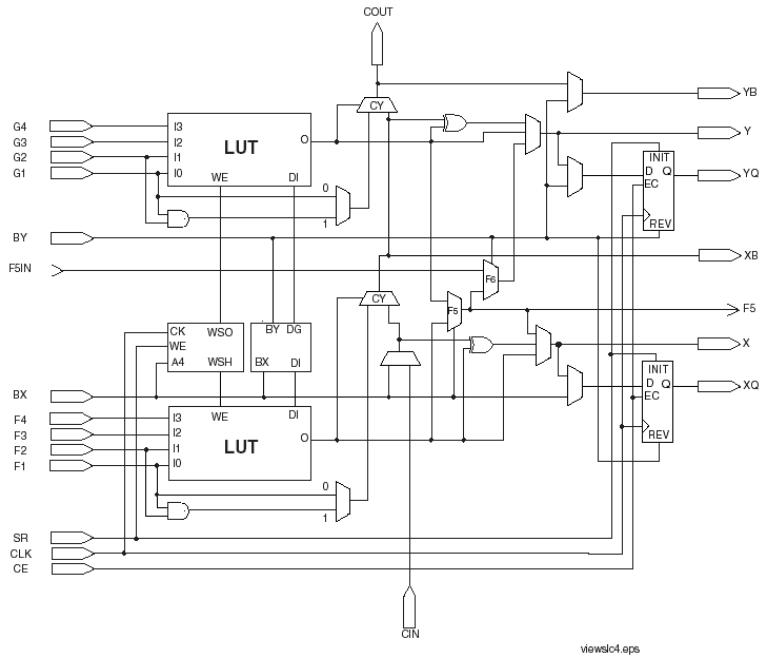


Figure 4.2: Slice schematic of the Virtex FPGA

The second issue involves errors in the feedback portions of the circuit. Errors that occur in the feedback of design, called persistent errors [24] pose two problems

to the system: persistent errors can take a long time to manifest themselves since they are located in feedback and persistent errors are not as easy to recover from as non-persistent errors. The former problem can affect the time it takes for a system to detect an error. Faults could occur in the feedback of a system forcing the system to an erroneous state. However, these errors could remain in the feedback undetected for a long period of time without manifesting at the output. The latter problem affects any external systems that are depending on the DWC-enhanced system. If external systems are performing corrective measures when an error is detected, it is beneficial for them to know whether the error is persistent. Non-persistent errors can be corrected by partial reconfiguration. However, to correct a persistent error a full system reset is needed.

The third issue is that the location of comparators can influence the time it takes for error to be detected. Long times from when an error manifests itself on the outputs to when the error flag is raised can cause problems for some systems. Ideally, the error flag would be raised before or concurrently with the manifestation of error caused by an upset. This can be ensured with the proper placement of comparators. Placing more comparators throughout the design decreases mean time to detection. Comparators can also be placed in the feedback logic where errors could sit indefinitely without manifesting themselves as previously mentioned.

For this work, a simple approach to determining where to place the comparators was adopted. Comparators are placed on all nets leading to output buffers as shown in Figure 4.3. This was done for two reasons: placing comparators on the final outputs allows the system to detect the majority of errors that occur in a system and this is a very simple format that would be easiest to implement in a CAD tool. The purpose of this work is to demonstrate that error detection can be implemented in an automated CAD tool and achieve a high percentage of errors correctly detected, not to explore the effects of comparator placement. Future work should include the study of the effects of comparator placement and more in-depth algorithms to determine the optimal placement for comparators.

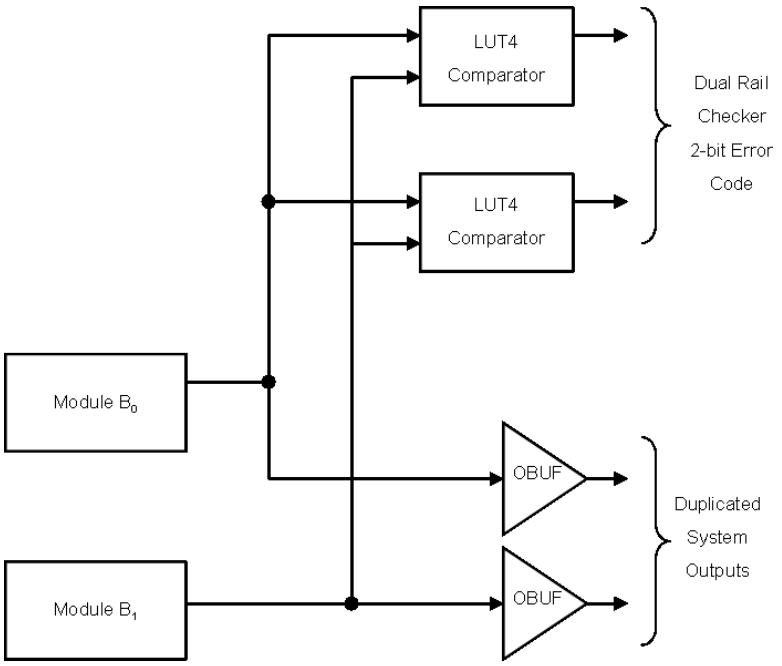


Figure 4.3: Diagram of comparators placed on the output nets of a system

4.1.3 Architecture of the Comparator

The third step to implementing DWC is to determine the architecture of the comparator circuitry to be inserted. The purpose of comparators is to check for differences in the two domains of a duplicated circuit. Comparators accomplish this by comparing two signals and outputting a zero if the signals match. If the signals do not match the output will be set high. A simple architecture for a two signal comparator can be seen in Figure 4.4. The two duplicated signals are compared with XOR gates. If they do not match an intermediate local error flag is raised. The two local error flags are combined using an OR gate. This simple comparator maps easily to a single 4-input LUT component that is the base unit for most FPGAs.

Comparators that are used to detect errors are also vulnerable to failure. These comparators represent additional logic and routing that is also susceptible to upsets. Upsets in the logic or the routing leading to the comparators can cause the detectors to signal that the system is in error when in fact it is not. A solution to this problem is to use a totally self-checking comparator [25, 26]. Totally self-checking comparators

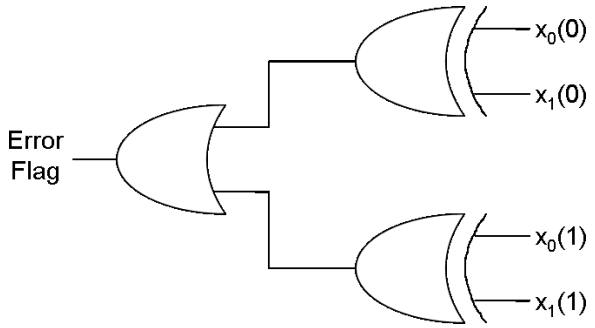


Figure 4.4: Comparator schematic

will output invalid error codes if either the inputs are invalid or a fault has occurred within the comparator circuitry. A simple self-checking checker to implement is shown in Figure 4.5. The original comparator is duplicated to form a 2-bit error code. When the code is ″00″ the circuit being checked and the comparator circuit are error free. A codeword of ″11″ indicates that the computational circuit is in error. Both ″01″ and ″10″ indicate that there is an error within the checker systems. Only ″00″ is a part of the valid subset of codewords indicating proper operation of our system.

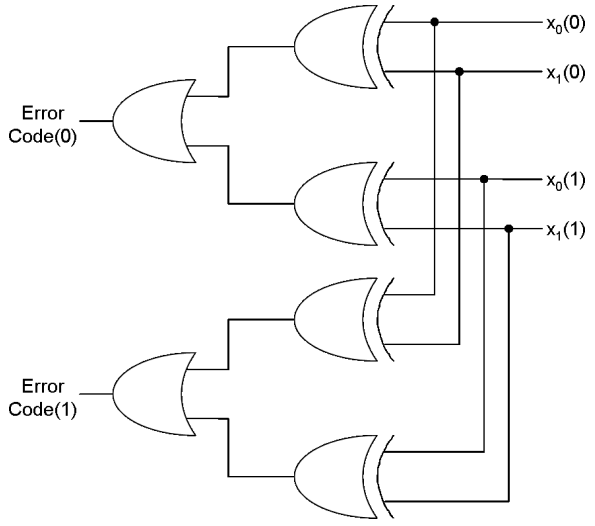


Figure 4.5: Dual rail checker

For this work, both the dual rail comparator architecture in Figure 4.5 and single bit comparator in Figure 4.4 were used and studied. The decision of which type of comparator to use is system dependent and can be controlled by the user. The impact of this decision on detectability will be examined later in this thesis.

4.1.4 Merging Local Intermediate Error Flags

The final step in implementing DWC is to merge all the local intermediate error flags from each comparator into a single error code output. Designs could consist of hundreds of local error flags if many comparators were added. They could also consist of no error flags if the error detection was performed off-chip in which case this step could be skipped. This section will present two different reduction networks that would accomplish this merging: a binary tree network and a daisy chaining network. If a dual rail comparator was used two separate networks would be used to merge the signals from each rail of the compare circuitry. These signals would be kept separate.

The binary tree reduction network shown in Figure 4.6(a) reduces the error flag signals in a tree fashion. The first level is 4-input LUTs that are used to compare two sets of signals (four inputs since the each signal is duplicated). All the other levels of the tree would contain 4-input LUTs that are the equivalent of 4-input OR gate. The number of levels of gates can be determined by the equation

$$number\ of\ levels = \lceil log_4(2 * n) \rceil, \tag{4.1}$$

where the variable n is equal to the number of outputs in the original system that are to be compared.

The daisy-chaining reduction network can be seen in Figure 4.6(b). This method creates a chain of LUTs that are comparing one signal (two inputs when duplicated) and merging that result with the results from previous compares. The error flag is passed along the entire chain and each compare merges into along the way. The number of levels would be equal to one less than the number of inputs.

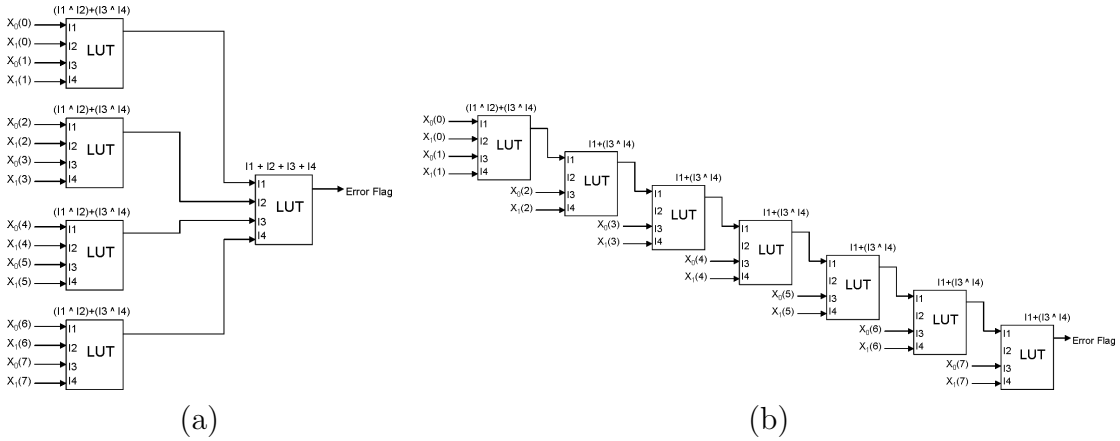


Figure 4.6: (a) Binary tree network for merging signals (b) Daisy chaining network for merging signals

Either method of merging the signals would introduce additional hardware and routing. These additional resources could alter the timing of the circuit. To prevent this, it may be necessary to pipeline the merging circuitry by adding registers. This would mitigate the potential reduction in clock frequency.

For this work only a simple implementation is used. Comparators are only being added to nets at the outputs of the system and are limited to a maximum of 70 comparators. Since the number of comparators is limited, merging of error flags is not a major concern. This work adopts the binary tree method shown in Figure 4.6(a) of merging intermediate error flags. This work will allow the place and route tools to map the merging elements in the configuration it deems best. Since few comparators are being used no pipelining is needed.

4.2 Classification of Detection Errors in FPGA-based DWC

The implementation approach of DWC for FPGAs chosen for use in this work will influence the types of errors detected. The purpose of this section is to classify the sources of each type of detection error. These types of detection errors defined for this work, discussed in Section 3.2, are sensitive undetected (SU), sensitive detected (SD), checker errors (CE), and false positives (FP). Each type of error will be explored in the context of actual circuit structure. This will be explained using the sample design

of Figure 4.7. Classifying these errors as done here will permit a better understanding of the effectiveness of the implementation approach of DWC chosen for this work.

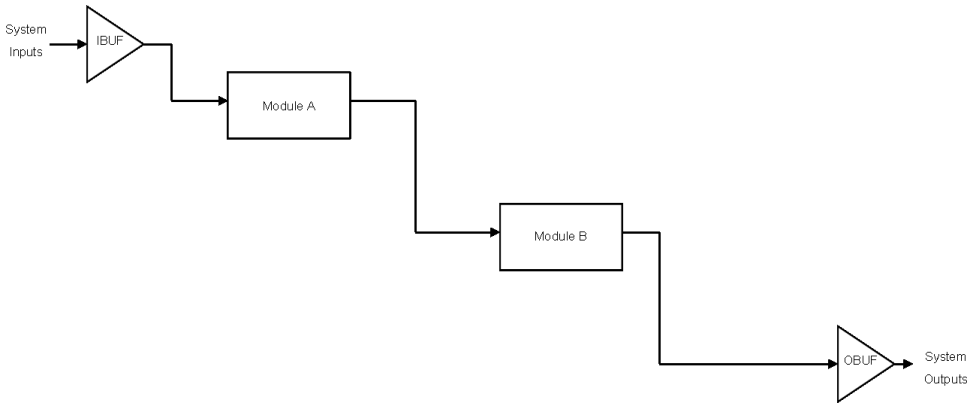


Figure 4.7: Original un-enhanced sample design

Figure 4.7, shows a very simple sample design with two circuit modules. The input drives the first module, termed module A. This module in turn drives module B. Module B drives the output of the system. Though small and simple, this design is representative of larger, more complex designs that could be used.

Figure 4.8 shows the same design once it has been modified to implement DWC. Both module A and module B have now been duplicated. The two DWC domains are represented by a subscript 0 and 1. The inputs of the circuit have been duplicated while the outputs were not duplicated. This is done to show the impact of un-duplicated inputs on a detection scheme which is what is done in this work.

Once the duplication was performed, the next step was to determine where to insert comparators. For this sample design, comparators were inserted between modules A and B and also between module B and the output buffers. Dual rail comparators were used to provide fault secure error detection (the domain of the comparator circuitry is shown using subscripts 0 and 1). Finally, the intermediate error flags from both locations of compare elements were merged to form a final 2-bit error code.

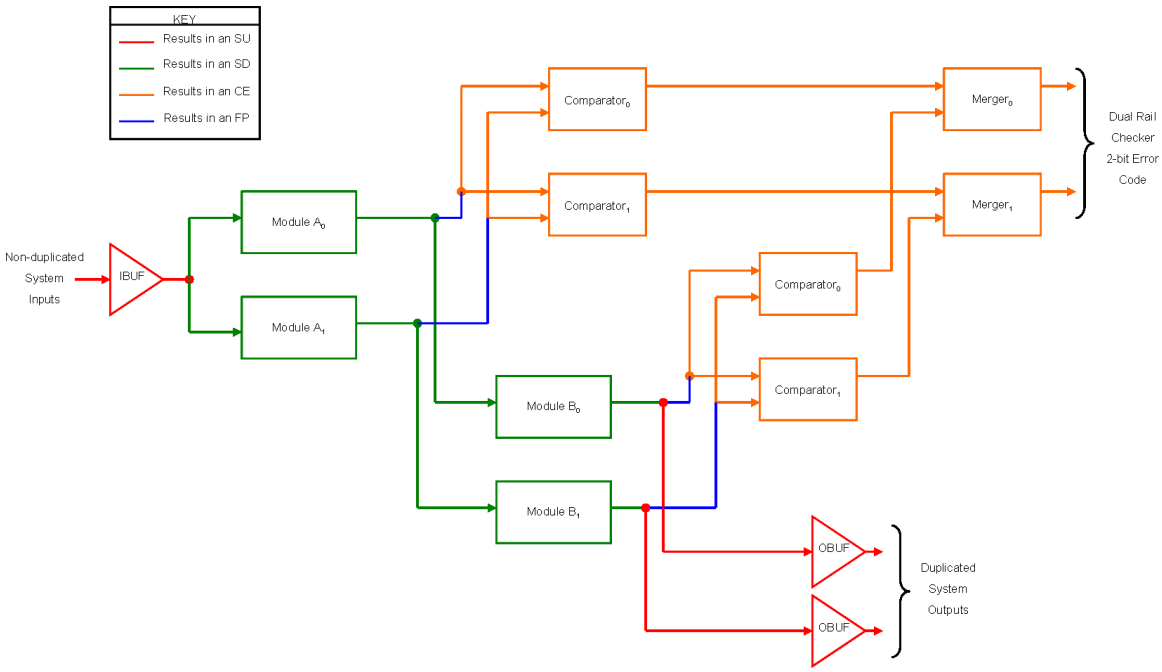


Figure 4.8: Classification of error types by routing location

Figure 4.8 also shows the locations in the circuit structure that cause each type of detection error. The next sections will discuss each type of error. Each section will begin with a brief introduction of the error and its relation to this work's implementation of DWC. Then the source locations in the circuit structure will be explored using the sample design. Locations will be in the routing, computational circuit (Modules A and B), I/O, and compare circuitry (comparators and mergers). Finally, the severity of each error will be discussed along with techniques to reduce the error if that is deemed necessary.

4.2.1 Sensitive Detected Bits in DWC

The first category of detection errors is sensitive detected. Sensitive detected errors are caused by upsets in the configuration memory that affect the routing or logic of only one of the duplicated domains. The circuit structures that cause these errors are shown in the figure with a green line. The comparator circuitry detects

that error and signals that an error has been detected with the appropriate error code output.

The figure shows that once the system input signal is split and routed to each domain, configuration bits controlling that routing to the modules result in sensitive detected bit errors. Any SEU that occurs in any of the modules would also result in a SD bit error. The same principle applies along the entire circuit. Routing connecting to the duplicated modules and the modules themselves are the source of SD bits themselves. The routing bits of the outputs are a cause of SD errors up to the point where the signal splits and is routed to the final comparators and the OBUFs. Once the split has occurred any errors on the signal do not cause sensitive detected errors. Sensitive detected bits are any bit that would affect one domain but not the others. These bits cause output errors that are detected.

The number of SD bits is primarily determined by the size, placement, and routing of the system. The goal of DWC in this work is to detect the majority of errors which results in higher numbers of sensitive detected bits. Therefore, reducing the number of sensitive detected bits is not needed because these type of bits are correctly diagnosed by the detection circuitry. Any effects caused by these bits would be correctly reported to the external systems.

4.2.2 Sensitive Undetected Bits in DWC

The second category of detection errors is sensitive undetected. Sensitive undetected errors occur when an error caused the outputs of the system to be incorrect but the error detection circuit failed to detect this error. These error types are caused by a routing net that does not get compared by the comparator circuit. In Figure 4.8, system resources that cause sensitive undetected errors are represented by a red line.

From the figure, it can be seen that the sources of SUs are the inputs to and the outputs from the circuit. Any errors that occur on the system input nets before they split and are routed to the different domains can cause SU errors. This is because both domains would get an identical erroneous input and the outputs would both be wrong in the same manner. No undetected errors are possible in the internal portion

of the circuit since it is not possible cause both domains to be incorrect in the same manner. The final system outputs of the system can also cause SU errors. The SU causing event can only occur in upsets located in the circuitry after the output has split, being routed to both the compare circuitry and the OBUF.

One final case of SU errors is not shown here. This case is an upset in globally shared resources such as the reset or clock lines. If these signals are not replicated any SEU that would affect them would cause the system to fail in an undetected manner.

The goal of DWC and error detection in general is to reduce or even eliminate sensitive undetected errors. SU errors are detrimental to the performance of the system. Duplicating both the inputs and the clock and reset lines would eliminate the SUs that occurred on that routing. Optimizing the routing at the OBUFs would help to reduce the SUs caused at the output. Without duplicating everything on the design (clocks, resets, etc.) the best that can be done is to get a very low maximum ceiling of SUs.

4.2.3 False Positive Bits in DWC

The third category of detection error type is false positives. False positive occur when an error affects both domains of the comparator system, causing both to output a positive error signal. False positive errors occur only on the routing to the compare circuitry. In the figure, routing that causes FPs is represented with a blue line.

The blue line in the figure shows that false positives can occur on a small portion of the net leading to the comparators. They can only occur on routing that only affects both domains of the error detection circuitry. In Figure 4.8, the source of these errors is the nets leading to the compare elements. The section of this net before it has split to be routed to both domains is to upsets that induce false positive events.

False positives are not detrimental to the system however they are misleading and may result in unnecessary repair. FPs indicate the output is in error when in fact it is not. Reducing false positives is a secondary goal of error detection schemes.

There are two controlling factors when determining the quantity of these types of errors: routing to the comparators and the number of comparators. Routing is usually fairly difficult to control with an automated tool. The number of comparators is controllable, using less comparators would result in less false positives.

4.2.4 Checker Error Bits in DWC

The final type of detection errors are checker errors. Checker errors occur from SEUs that occur in the comparator circuitry. In the figure checker errors are represented by orange lines. The figure shows that checker errors occur once the output of the modules has split into the two branches that are tied to the inputs of the comparator domains. Also any errors in the comparator logic itself also produces CEs. This is because errors here would change the output of one domain of the compare circuit to be incorrect resulting in error codes of ″01″ or ″10″. It is important to note that errors that occurred in the circuitry that merged the multiple error flags together (see Figure 4.6(a) and Figure 4.6(b)) would also result in CEs.

Checker errors, like false positives, are determined by the routing and number of comparators. A system with a large number of comparators would result in extensive routing and extra logic to merge the error flag signals. All this excess logic and routing would be susceptible to producing CEs if SEUs occur. However, checker errors do not pose a problem for error detection schemes and do not need to be reduced.

4.2.5 Summary

The goal of DWC is to maximize the percentage of correctly detected errors. This can be done by minimizing the number of sensitive undetected errors. Sensitive undetected is the only error type that is not reported by the error detection scheme. In this discussion of error types, it was shown that the more duplication that is applied to the circuit the less SU errors that would occur. A fully duplicated circuit, including inputs, clocks, outputs, and resets would result in no sensitive undetected errors. DWC could provide full coverage of correctly detected errors and in the implementation chosen for this work provides nearly full coverage.

4.3 Results from Manually Implemented DWC

To verify that DWC could be successfully implemented and achieve reasonable results, several designs were manually modified with DWC. A single bit comparator architecture was used in these designs. The fault injection simulator was used to show the effectiveness of DWC in terms of error detection. This section will introduce the designs that were modified and how the modifications were enacted. Then, the results obtained by running these designs through the fault injection simulator will presented. The designs will be discussed in terms of size, SUs, SDs, and FPs. Finally, a summary of the overall results will be presented.

Three simple designs were used to generate these results: a 5-bit counter, a 30-bit counter, and a design with 30 counters. These designs were chosen for their simplicity and regularity which made it easier to implement DWC. Table 4.1 summarizes the results that were obtained when these simple DWC-enhanced designs were run through the fault injection simulator. The numbers shown in this table are size (in terms of slice count), SUs, SDs, FPs, and significant bits which is the sum total of the SUs, SDs, and FPs. Since a dual rail comparator was not used no CEs occurred.

Table 4.1: Results for the hand-DWC designs

Design	**Slices**	**SUs**	**SDs**	**FPs**	**Significant Bits**
5-bit counter	12	525	632	201	1,358
30-bit counter	59	965	1,302	402	2,669
multi-counters	322	874	6,866	415	8,155

Sensitive undetected errors are the errors that negatively affect the performance of the system. A good error detection scheme will limit or eliminate the number of sensitive undetected errors. The results show relatively high numbers of SUs. A closer examination reveals an encouraging trend; the number of SUs does not increase

significantly with increased size. Both the 30-bit counter and the multi-counters design have similar numbers of SUs. This, despite the fact that the multi-counters design is significantly larger and has more significant bits. This suggests that SUs have a fixed ceiling that is determined by the number of inputs and outputs of the design. If the desire was to eliminate these SUs, the inputs and outputs could be duplicated and the final comparison performed off-chip.

Sensitive detected do not negatively affect the performance of the system. High numbers of sensitive detected, especially when compared with the total significant bits, indicate the error detection scheme is achieving high percentage of errors correctly detected relative to design size. The numbers present in the table show that SDs are highly correlated to the size of the design. In fact, with larger designs, such as the multi-counters design, where the fixed ceiling of SUs is a smaller part, SDs represent a clear majority of significant bits. This indicates high percentage of errors detected.

False positives are errors that occur in the checker circuitry. The table shows that the smaller designs large amounts of false positives when compared with the total number of significant bits. However, it is clear that FPs behave similar to SUs and are constant. This can be seen by comparing the 30-bit counter design with the multi-counters design. This supports the earlier discussion that FPs are determined by the number of checkers (for a further discussion of the cost of a comparator see Appendix A). It is important to note that FPs could be reduced by using a dual rail checker.

These results demonstrate that DWC can be used as an effective error detection scheme. The most important metric is the sensitive undetected errors. These errors demonstrate a lack of coverage for the error detection scheme. It can be seen from a comparison of the three designs and the previous discussion that this is a relatively fixed value. As the design increases in complexity the SUs represent a smaller portion of the total number of significant bits. SDs dominate the significant bits which indicates good coverage can be achieved with this implementation of DWC.

4.4 Summary

Duplication with compare provides an error detection technique that can detect the majority of errors. The process of applying DWC can be divided into four steps: duplicating the circuit, determining locations to insert comparators, choosing a comparator architecture, and merging the intermediate error flags. For this work, the full circuit will be duplicated, except for the inputs. Comparators will be inserted on the nets leading to the output buffers. This works will support both single bit comparators and dual rail comparators. The topology of the merging of the error flags will be left to the place and route tool. These decisions were made to provide an implementation of DWC that is easy to automate and provided high percentages of errors detected.

With this implementation determined, it was possible to classify the error types that would result from a given circuity. The errors were divided into sensitive undetected, sensitive detected, checker errors, and false positives. The circuit elements that caused each source were examined to determine the potential error detection coverage of the chosen implementation of DWC.

Several small designs that had been hand altered with DWC were tested. The results verified that this implementation of DWC could obtain very good percentages of errors detected. The results suggest that both FPs and SUs had a ceiling that was determined by the number of comparators and the number of I/O respectively. DWC was able to detect a large portion of the errors and as the design size increased the percentage of errors detected increased.

Chapter 5

Automated DWC Tool and Results

Applying DWC to a design by hand is a very time-consuming process. To facilitate the use of this technique, an automated CAD tool that can apply DWC to a design is needed. This chapter demonstrates that DWC can be implemented as part of an automated CAD tool. With this tool, larger designs could be enhanced with DWC. The purpose of this chapter is to give a brief overview of the automated tool that was developed and to examine the results, in terms of the metrics percentage of correctly diagnosed configuration bits (CDCB) and the percentage of correctly diagnosed significant bits (CDSB), that were obtained using this tool on a suite of benchmark designs.

This chapter will begin with a discussion of the automated tool that was developed. Next, the discussion will focus on the results that were obtained by implementing DWC on larger designs with the CAD tool. Then, the results that were obtained when running the DWC-enhanced designs through the fault simulator will be presented. The results of several different runs using the different user-options will be presented. These results will be analyzed and explained in terms of the effects of the user options on the behavior of the DWC and then in terms of the effectiveness of the error detection scheme. The results will be used to show that the implementation of DWC fulfills the goals of this work: it is automatable and it correctly detects most SEUs.

5.1 Automated DWC Tool

A Java-based automated tool was designed to apply the DWC technique to FPGA designs. The automated tool known as BYU LANL DWC or BLDwc for short,

takes as an input a design's netlist (in EDIF format) and manipulates the design to add DWC. The output of the tool is a modified netlist with DWC. This section will give an overview of the most important steps the tool performs and then some of the options controllable by the user. This tool provides verification that DWC is automatable.

The process of DWC can be broken into seven major steps. This tool was based on previous tools, including a TMR tool created at BYU [27]. The basic flow of the tool can be seen in Figure 5.1. The steps are as follows:

1. **Parse EDIF File** The first step is to parse the EDIF netlist into separate components. Objects are created representing each of the different components in the netlist (i/e port, net, cell, cell instance, etc.). These objects are linked to one another to preserve the structure and connectivity of the original design.

2. **Flatten EDIF File** The second step is to flatten the design into primitive components (i/e LUTs, Flip-flops, etc). The flattened structure is needed to allow the tool to be able to fully analyze the structure of the design. Hierarchy abstracts away many of the details and is difficult for the automated tool to handle. This steps creates a flattened version of the original design that is devoid of hierarchy while still maintaining the original connectivity.

3. **Duplicate circuit** The third step in the BLDwc tool is to duplicate the design. During this step a new EDIF cell representing the top level of the DWC enhanced design is created. All the primitive cells and nets from the flattened original cell are added twice to the new cell. The connectivity is retained as it was in the original design. Duplication of the outputs is a user-defined parameter that will be discussed later.

4. **Select comparator locations** The next step is to select locations where the comparators are to be inserted. For this work, all nets that lead to output buffers are selected as locations for comparators. Since these nets are duplicated they are passed as pairs to the next step.

5. **Determine comparator architecture** The next step is to determine the architecture of the comparators to be inserted. There are two possible comparator architectures, a single bit comparator or a dual rail comparator. This is a user defined parameter that will be discussed in more detail later in this chapter. The DRC results in two XORs being added to the design while the single bit comparator results in the addition of only one XOR to the design. The inputs of the XOR gate(s) are tied to the duplicated nets. The outputs of all the added comparators added are passed to the next step.

6. **Merge local error flags** The next step is to merge all the local intermediate error flags until only a single error code is left. This final error code could be a 1-bit line if the user specified single bit comparators or a 2-bit line if the user specified dual rail comparators. The nets are all merged by adding OR gates to the design. Two intermediate flags are tied to the inputs and the output will be merged repeatedly until only a final error code is left dangling. This final error code is then tied to the appropriate output port.

7. **Generate DWC EDIF** The final step is to use the newly created EDIF cell to generate the resulting .edf file. This is done by converting each object (net, port, cell, cell instance, etc.) back to EDIF code. When combined together properly it forms a functioning .edf file.

   The resulting .edf file has the same functionality as the original design only it has been augmented with error detection circuitry. The design now has additional output ports for the error flags and the duplicated outputs. The user must be aware of this and tie these pins to the appropriate ports.

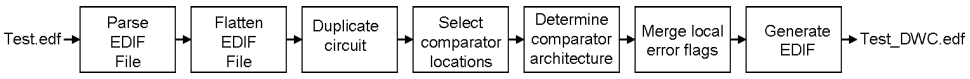


Figure 5.1: Flow chart of the automated BLDwc tool

5.2 User Specified Parameters

User parameters permit the designer fine-grained control over the operation of the DWC tool. The parameters affect the area usage of DWC, the information that is reported to external systems, and the system's interaction with external systems. The two major parameters the user will select are the architecture of the comparator and the duplication of the output ports. The former allows the user to choose between a single bit comparator with single bit error codes and a dual rail comparator with 2-bit error codes. The latter option allows the user to determine if the outputs should be duplicated or not.

The user defined options allow the user some control over the implementation of DWC on the design. These options can be used to control the additional resources needed, the amount of detail the detection scheme provides, and the interaction with the external systems. In future work, more options could be added to allow the user control over the amount of the circuit that is duplicated (partial DWC) and over the placement of comparators.

5.3 Automated Tool Results

The automated BLDwc tool has shown that DWC can be implemented in an automated CAD tool. The tool provides many designs that could be augmented with DWC to determine the effectiveness of DWC on FPGA-based designs. The purpose of this section is to present the results that were achieved when DWC was applied to a suite of benchmark designs. These results will show the effectiveness of DWC as an error detection technique.

This section will begin by presenting the suite of benchmark designs that were used. Next, the results obtained by running each maximally enhanced (duplicated outputs and DRC) design through the simulator will be explored. Maximally-enhanced designs were used to provide the most detailed results. The discussion will focus on the quantity of each event type that resulted. Finally, a detailed comparison of the effects of the two user-controlled options will be undertaken. The two comparator architectures and the duplication of the output ports will be evaluated.

5.3.1 Benchmark Designs

A suite of benchmarks was created to demonstrate the effectiveness of the error detection methodology presented in this work. These designs were chosen because they represent a wide array of functionality, implementation, and sizes.

The *counters200* design is a useful test case because it has significant amounts of feedback and state. The design is a synthetic design composed of 200 loadable 8-bit counters chained together. The output of the design is a 16-bit parity check of the counters. Each counter generates a parity bit. The 200 parity bits are XOR-ed together to form a 16-bit value that is tied to the system outputs.

The *synthetic* design contains moderate amounts of state and significant portions of feed-forward logic. The design consists of several LFSRs whose outputs are combined together by an adder tree. This output is multiplied with the 16-bit input value. The top 16 bits of this output are added to the bottom 16 bits to form a sum that is tied to the outputs of the system.

The *quadrature phase-shift keying (QPSK) demodulator* implements a real-world communication algorithm. QPSK is a digital modulation that encodes data using the phase of the carrier signal. QPSK is often used in the communication world. This design is a good representation of a real world design with significant computations and large amounts of feedback.

A *triple DES encrypter* was chosen to represent real world computationally intensive system. Triple DES is a block cypher formed by applying the standard DES encryption scheme three times to a data packet. This design was chosen for use in this work for two reasons. The first reason is that it represents real world computationally intensive feed-forward designs. The second reason is the size of the design. In its original form 46% of the chip was used. Once DWC was applied, 99% of the chip resources were used up. This design is difficult to place and route since the whole chip is used. This design show how well DWC can work in the face of resource constraints.

These designs will be used throughout this section to generate results. This suite represents a wide range of designs ranging from high-feedback designs to feed-forward only designs and from synthetic to real world designs.

5.3.2 Results of the Exploration of DWC

The designs of the benchmark suite were run through the simulator with the options set for duplicated outputs and dual rail comparators. This section will present the results obtained and analyze them in terms of the four significant events: SUs, SDs, FPs, and CEs. The discussion of each event will focus on the severity of each event, the location of the configuration bits that cause the event, and the ways to minimize each event.

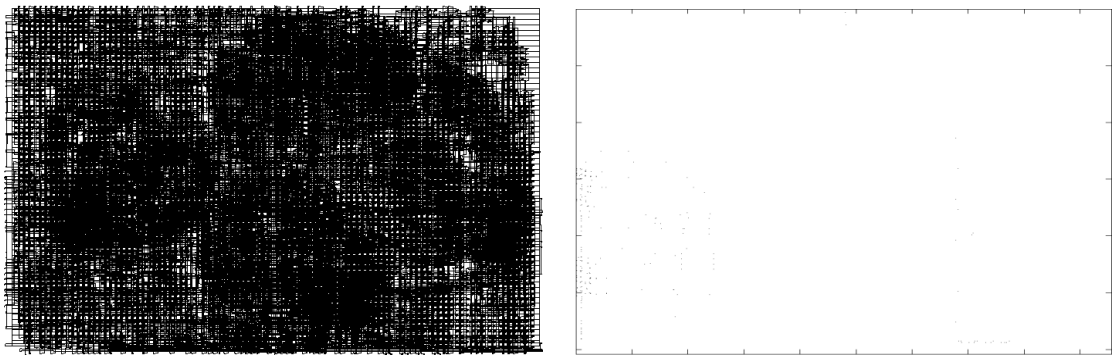
Table 5.1: Results from simulator

Design	Slices	Significant Bits	SUs (%)	SDs (%)	FPs (%)	CEs (%)
Counters200	2,171	273,196	696 (0.25%)	271,542 (99.39%)	188 (0.07%)	770 (0.28%)
Synthetic	9,343	587,200	1,603 (0.27%)	584,191 (99.49%)	113 (0.02%)	1,293 (0.22%)
QPSK Demodulator	2,248	182,754	1,236 (0.68%)	180,467 (98.75%)	208 (0.12%)	763 (0.42%)
Triple DES	12,286	1,368,386	3,087 (0.23%)	1,363,041 (99.61%)	432 (0.03%)	1,826 (0.13%)

Table 5.1 shows the results that were obtained from the suite of benchmark designs. The percentages are given for each event in terms of a percentage of the total significant bits. For example, SUs represent 0.25% of the total significant bits of the counters200 design. This section will explore each of the event types and the bits that cause them.

Sensitive undetected bits are bits that when upset affect the system but are undetected by the DWC error detection scheme. Sensitive undetected errors are the

worst type of errors as they result in system failures and reduce the percentage of errors that are correctly diagnosed. The table shows that sensitive undetected bits represent a very small amount of the significant bits. Specifically, less than half a percentage point of all the significant bits. A more detailed exploration into this produces Figure 5.2(a) and Figure 5.2(b). Figure 5.2(a) shows the layout of the synthetic design on the actual chip. As you can see the design takes up a large part of the chip. Figure 5.2(b) shows the location of the configuration bits that cause sensitive undetected errors in the synthetic design. Upon careful examination, it can be seen that the SU bits represented by small dots are sprinkled along the left edge and also along the bottom edge on the right side. The left edge is the location of the input buffers which are susceptible since they are not duplicated. The bottom right edge is the location of the output buffers which are also susceptible since they are also not duplicated. As stated earlier and shown in these figures, if the entire circuit was duplicated including all the inputs and the outputs and the compare elements were moved off-chip, the sensitive undetected bits would be eliminated.

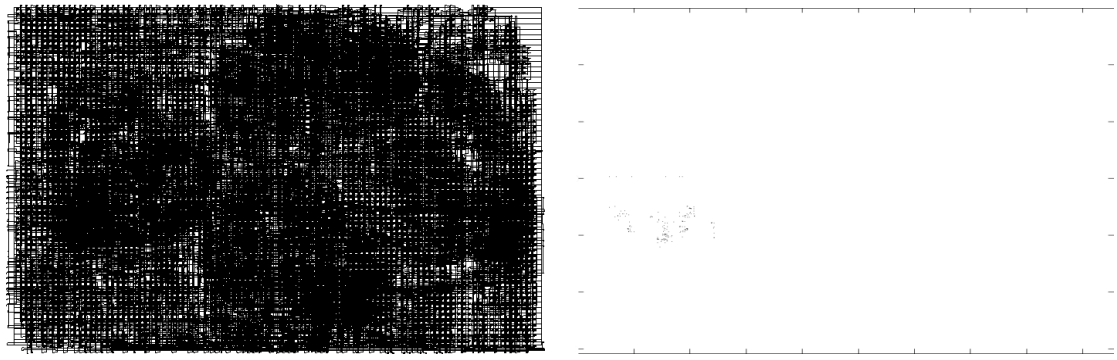


(a) (b)

Figure 5.2: (a) Design layout and routing (b) Location of the configuration bits that cause sensitive undetected events

False positives errors also negatively affect a system by insinuating that the output cannot be trusted, when in actuality the output is uncorrupted. From the results table, it can be seen that bits that cause false positives represent a negligible

portion of the overall significant bits (< 0.1%). Examination of the location of the false positives is difficult because the number of bits is very small. However, the locations shows that false positives are caused by the routing of signals leading to the comparators. The routing of duplicated nets that contain comparators is susceptible to false positives before it has split to go to the two comparator domains (see Section 4.2). It is not possible to eliminate the false positives without reducing the number of comparators[1]. However, false positives do not cause system failures, they merely result in external systems not trusting data that is unaltered.



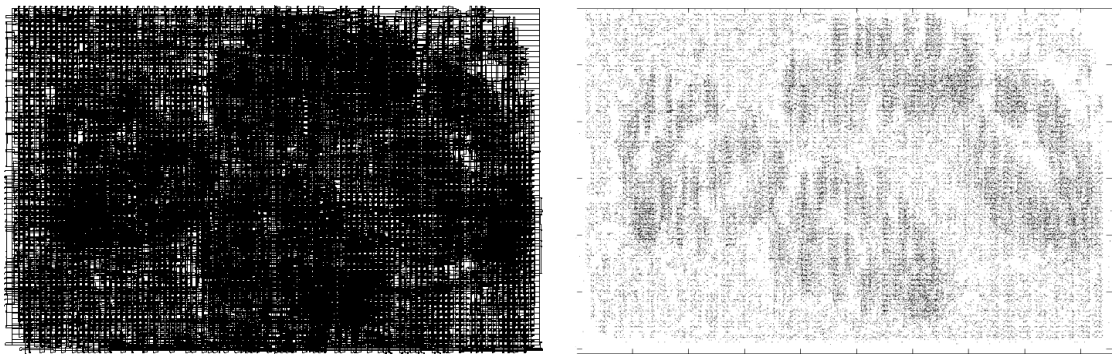
(a) (b)

Figure 5.3: (a) Design layout and routing (b) Location of the configuration bits that cause checker error events

Bits that cause checker error events do not unnecessarily retard the operation of the system. Checker errors indicate an fault has occurred in the detection circuitry. CEs represent a small portion of the total significant bits (< 0.5%). Figure 5.3(b) shows the location of the configuration bits that cause CEs. These bits are local to specific positions which correspond to the slices containing comparator circuitry and the routing leading to it.

Sensitive detected bits also do not reduce the efficiency of a system. SD bits are those that cause errors in the functional logic and are correctly diagnosed. Sensitive detected bits represent greater than 99.3% of the significant bits in each of the tested

[1]A more in-depth treatment of the risk analysis of each comparator can be seen in Appendix A



(a) (b)

Figure 5.4: (a) Design layout and routing (b) Location of the configuration bits that cause sensitive detected events

designs. These bit errors occur from upsets located anywhere that there is utilized logic. Figure 5.4(b) shows the locations of the bits causing SD events. The figure shows that the configuration bits do in fact appear anywhere logic is utilized. If there is heavier utilization of the logic then there will be a heavier concentration of sensitive detected bits. Both sensitive detected events and checker errors events are reported and handled correctly by the system.

Table 5.2: Comparison of the original design with the DWC design

Design	Implementation	Slices	% Increase	Failure Bits
Counters200	BLDwc	2,171	102%	696
	original	1,076	0%	120,167
Synthetic	BLDwc	9,343	210%	1,603
	original	3,012	0%	256,668
QPSK Demodulator	BLDwc	2,248	114%	1,316
	original	1,049	0%	87,543
Triple DES	BLDwc	12,286	128%	3,087
	original	5,387	0%	667,067

The major drawback of DWC is size. As can be seen from Table 5.2, the designs are larger. The size results for the synthetic design are misleading. If viewed

in terms of number of LUTs and flip-flops the design is roughly $2X$ larger. When mapped the size expands to roughly $3X$ in terms of slices. Overall, as expected, the designs are $2-3X$ larger than the original designs. This extra area is due to the duplicated circuit and also to the extra hardware and routing needed to implement the compare circuitry.

Table 5.3: Effectiveness calculations of the testbench designs

Design	CDCB	CDSB
Counters200	99.988%	99.745%
Synthetic	99.972%	99.727%
QPSK	99.979%	99.324%
Triple	99.947%	99.774%
Average	99.972%	99.642%

From these results, it is shown that, based on the assumption that only sensitive undetected errors cause system failures, the chosen error detection scheme, DWC, has an estimated 99.95% to 99.99% probability of correctly diagnosing a configuration bit and a 99.32% to 99.77% probability of correctly diagnosing a significant bit. These numbers are based on the results obtained in this work and shown here. CDCB and CDSB were calculated using the definitions in Equation 3.8 and Equation 3.7. For example, the CDCB of the synthetic design is determined as follows

$$CDCB(\%) = (1 - \frac{1,603}{5,810,024}) * 100 = 99.972\%, \tag{5.1}$$

where the number $5,810,024$ is the number of configuration bits in the Virtex1000 device. The CDSB of the synthetic design would be determined in the equation

$$CDSB(\%) = (1 - \frac{1,603}{1,603 + 584,191 + 113 + 1,293}) * 100 = 99.727\%. \tag{5.2}$$

This equations generates the CDCB and CDSB for the synthetic design using the numbers shown in Table 5.1. Table 5.3 shows the results for each of the testbench

designs and the average. The average percentage of correctly diagnosed configuration bits is 99.972% and the average percentage of correctly diagnosed significant bits is 99.642%.

The effectiveness of the DWC-enhanced designs is in stark contrast to the unprotected design in which every configuration bit utilized by the design is sensitive to SEUs and if hit would result in system failure. When a comparison of the number of bits that would cause a failure is made the results are even more dramatic. Table 5.2 shows that the unprotected system has $160-175X$ more bits that would cause failures if upset. This is a very significant difference. With a minimal size increase of $2-3X$, the automated tool can augment a design so that it can detect almost all failure-inducing errors. If all the inputs and outputs were duplicated, all failure causing errors would be detected and full coverage of errors correctly diagnosed would be achieved. The DWC implementation used for this work has been shown to be automatable and to correctly detect approximately 99.972% of SEUs that affect a chip.

5.3.3 Results of the Exploration of the Comparator Architecture

The user has the option of employing single bit comparators and dual-rail comparators (DRC) in the design. Dual rail comparators provide a 2-bit final error code that contains more information than can be encoded in a single bit final error code. The choice of comparator will affect both the size and the manner in which errors in the detection circuitry are reported. Table 5.4 shows the results that were obtained when each of the designs in the benchmark suite were run twice, once using single bit comparators and once using dual rail comparators. These results will be compared in terms of size and error detection events.

The table shows that the resource increase due to the usage of dual rail comparators is minimal. This is in part due to the chosen implementation of DWC. Using the BLDwc tool, comparators are only inserted on outputs which results in only a limited number of comparators being added. However, it is interesting to observe that on the triple DES design no change in size is noted. This is because the design

is being packed, just to fit on the chip. Comparing the sizes shows that size is not a huge drawback of the dual rail checker.

Table 5.4: Analysis of dual rail and single bit checkers

Design	Type of Checker	Slices	CEs	FPs
Counters200	Single	2,156	N/A	533
	DRC	2,171	770	188
Synthetic	Single	9,343	N/A	597
	DRC	9,334	1,293	113
QPSK Demodulator	Single	2,246	N/A	572
	DRC	2,248	763	208
Triple DES	Single	12,286	N/A	1,033
	DRC	12,286	1,826	432

The DRC results in better reporting of events. All errors in the detection circuitry become FPs using the single bit comparators. Using a 2-bit error code allows for more information to be passed to the external systems. The detector is able to differentiate between errors in the detection circuitry (CEs) and errors that occur in the logic. The table shows that this results in significantly less FPs for the DRC systems. However, because twice as many compare elements are needed to form the DRC system, the DRC system results in more total events in the detection circuitry (FPs and CEs). For example, the triple DES system results in 1,033 detection events if single bit comparators are used. However, if DRCs are used the triple DES benchmark results in 2,258 total detection events. That is over a $2X$ increase which is expected because the DRC is twice the compare units of the single bit system.

The dual rail comparator architecture uses a 2-bit error flag to provide information indicating the location of the upset. It can effectively reduce false positive events by correctly categorizing many of them as upsets in the detection circuitry (CEs). DRC is applied at the cost of additional area. This discussion has shown that the area increase when using the DRC comparator architecture does not seem

significant. However, this additional area is susceptible to upsets resulting in an increase total number of events. In conclusion, DRC is better at correctly categorizing events but suffers from an increased number of configuration bits that cause events. Users should be aware of these tradeoffs when making decisions concerning the type of comparator architecture to use during DWC.

5.3.4 Results of the Exploration of Duplicated Outputs

The user also has the option of duplicating the outputs. As previously discussed, this option affects both the number of outputs needed and the need for off-chip comparisons. For this study, designs were run through the tool twice, once with duplicated outputs and once without duplicated outputs. The resulting designs were then run through the fault injection simulator. This section will explore the effect duplicated outputs has on the accuracy of the report from the fault injection simulator. This will justify the decision made in this work to always duplicate the outputs when testing designs.

If the outputs are not duplicated the simulator only has access to one set of outputs from the design under test (DUT). This prohibits the simulator from comparing the outputs of both domains of the DWC, resulting in skewed results. The simulator cannot accurately differentiate between false positives events and sensitive detected events if it cannot determine whether an error was found on the output it cannot access. Take for example the case where an upset causes domain 1 to be in error and the detection circuit correctly reports it by setting the appropriate error code. If the system has duplicated outputs, the simulator correctly reports that this is an SD event. If the design does not have duplicated outputs the simulator cannot tell that domain 1 is in error because it only has access to the outputs of domain 0. It therefore incorrectly classifies this event as an FP.

Table 5.5 verifies this conclusion. The designs without duplicated outputs result in a $2X$ increase the number of FP events. Un-duplicated output designs also results in half the number of SD events. Events that should have been classified as

SDs are being classified as FPs because the simulator does not have access to both outputs to correctly ascertain the event that actually occurred.

Table 5.5: Analysis of tradeoffs of duplicated outputs

Design	Duplicated Outputs	SDs	SUs	FPs
Counters200	Yes	266,306	655	533
	No	120,218	493	113,800
Synthetic	Yes	574,784	1,640	597
	No	285,141	1,311	285,770
QPSK Demodulator	Yes	183,357	1,236	572
	No	90,083	781	88,797
Triple DES	Yes	1,373,542	2,990	1,033
	No	718,139	2,495	712,436

In conclusion, when using the simulator duplicated outputs should always be used to allow for correct classification of detection events. If duplicated outputs are not used, the simulator has difficulty differentiating between many SD and FP events. For systems destined for real world applications the decision should be made based on the hardware available and the availability of off-chip comparators.

Chapter 6

Conclusion

Duplication with compare has been shown to provide an alternative to readback with compare as an FPGA-based error detection scheme. Readback with compare has limitations that it cannot detect upsets in the user defined memory. DWC can detect errors in both the configuration bitstream and the user defined memory. DWC also has lower off-chip cost than readback with compare.

This work has shown that DWC can be implemented in an automated CAD tool. DWC was implemented as part of the BLDwc CAD tool. This tool can read in an EDIF netlist, apply DWC, and then output a new netlist with error detection circuitry. This tool offers great benefits to those who are interested in applying error detection to a design. The tool is simple to use and can quickly alter the inputted design.

This work has also shown that for designs that are able to tolerate known temporary variations from normal operation, error detection provides a good solution to the problem of SEU-induced failures. Applying duplication with compare using the automated BLDwc tool to the circuit will increase the reliability of the circuit. For our test cases, duplication with compare was able to detect approximately 99.97% of the errors caused by SEUs. This nearly full coverage of errors detected is the result of an approximate $2X$ increase in the size of the design.

The findings of this work offer benefits to any space-destined system that has resource constraints and can tolerate temporary interruptions of service. With the tool DWC can be applied quickly, and generates significant benefits. This could be applied to many communication and signal processing applications as shown by the example QPSK demodulator circuit. With streaming input data, temporary failures

often do not adversely affect the results as long as the interruptions are known and limited.

There is much future work that could be done to build on these studies. One of the major areas of concern is the reporting of persistent errors. Persistent errors cannot be corrected in the same manner as non-persistent errors. Because of this, future modifications of the BLDwc tool should allow the user the option of separating the error detecting lines into a non-persistent error flag and a persistent error flag. Even so, the high cost of the system reset needed to correct faults that cause persistent errors, indicates that other solutions should be explored. One possible solution is a hybrid TMR/DWC approach. Modifying the tool to protect the persistent cross-section with TMR, while protecting the remainder of the design with DWC would result in the elimination of persistent errors and detection of all non-persistent errors.

It would also be interesting to explore a combination TMR/DWC technique in which TMR mitigated the errors but also reported that a fault had occurred. This would be useful in both a research setting and also in applications. TMR must currently be paired with periodic scrubbing to restore the configuration bitstream. Detecting the faults would allow the periodic scrubbing to be replaced by interrupt-driven scrubbing. A more extensive risk analysis should be done on placement of comparators to determine the trade-offs of additional comparator elements in terms of both numbers and location. Future research should focus on techniques to improve the automated tool placing greater control in the hands of the user. This would allow design specific reliability which would make the best use of area and redundancy.

Bibliography

[1] D. K. Pradhan, *"Fault-Tolerant Computer System Design"*. Prentice Hall, 1998. xix, 7, 12

[2] "FPGAs on Mars," August 2004. [Online]. Available: http://www.xilinx.com/publications/xcellonline/xcell_50/xc_pdf/xc_mars50.pdf 1

[3] M. Caffrey, "A space-based reconfigurable radio," in *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, T. P. Plaks and P. M. Athanas, Eds. CSREA Press, June 2002, pp. 49–53. 1

[4] A. S. Dawood, S. J. Visser, and J. A. Williams, "Reconfigurable FPGAs for Real Time Image Processing in Space," in *14th International Conference on Digital Signal Processing (DSP 2002)*, vol. 2, 2002, pp. 711–717. 1

[5] M. J. Wirthlin, D. E. Johnson, N. H. Rollins, M. P. Caffrey, and P. S. Graham, "The reliability of FPGA circuit designs in the presence of radiation induced configuration upsets," in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM '03)*, K. L. Pocek and J. M. Arnold, Eds. IEEE Computer Society Press, April 2003, pp. 133–142. 2, 8

[6] E. Fuller, M. Caffrey, P. Blain, C. Carmichael, N. Khalsa, and A. Salazar, "Radiation test results of the Virtex FPGA and ZBT SRAM for space based reconfigurable computing," in *MAPLD Proceedings*, September 1999. 2

[7] R. Katz, K. LaBel, J. Wang, B. Cronquist, R. Koga, S. Penzin, and G. Swift, "Radiation effects on current field programmable technologies," *IEEE Transactions on Nuclear Science*, vol. 44, no. 6, pp. 1945–1956, December 1997. 2

[8] M. Wirthlin, E. Johnson, N. Rollins, M. Caffrey, and P. Graham, "The reliability of FPGA circuit designs in the presence of radiation induced configuration upsets," in *Proceedings of the 2003 IEEE Symposium on Field-Programmable Custom Computing Machines*, K. Pocek and J. Arnold, Eds., IEEE Computer Society. Napa, CA: IEEE Computer Society Press, April 2003, p. TBA. 2

[9] C. Carmichael, M. Caffrey, and A. Salazar, "Correcting single-event upsets through Virtex partial configuration," Xilinx Corporation, Tech. Rep., June 1, 2000, xAPP216 (v1.0). 2, 10

[10] F. Lima, C. Carmichael, J. Fabula, R. Padovani, and R. Reis, "A fault injection analysis of Virtex FPGA TMR design methodology," in *Proceedings of the 6th*

European Conference on Radiation and its Effects on Components and Systems (RADECS 2001), 2001. 2, 6

[11] C. Carmichael, "Triple module redundancy design techniques for Virtex FPGAs," Xilinx Corporation, Tech. Rep., November 1, 2001, xAPP197 (v1.0). 5

[12] M. J. Wirthlin, D. McMurtrey, B. Pratt, and K. Morgan, "Fault Tolerant Design Techniques for FPGAs - a Case Study," 2006. 6

[13] N. Rollins, M. Wirthlin, M. Caffrey, and P. Graham, "Evaluating TMR techniques in the presence of single event upsets," in *Proceedings fo the 6th Annual International Conference on Military and Aerospace Programmable Logic Devices (MAPLD)*. Washington, D.C.: NASA Office of Logic Design, AIAA, September 2003, p. P63. 6

[14] S. Mitra and E. J. McCluskey, "Which Concurrent Error Detection Scheme to Choose?" *IEEE Proceedings International Test Conference 2000*, p. 985, 2000. 6

[15] L. Spainhower and T. A. Gregg, "IBM/S390 Parallel Enterprise Server G5 Fault Tolerance: A Historical Perspective," *IBM Journal of Research and Development*, vol. 43, no. 5/6, 1999. 6

[16] R. E. Bryant and D. R. O'Hallaron, *"Computer Systems"*. Prentice Hall, 2003. 6

[17] L. L. Peterson and B. S. Davie, *"Computer Networks"*. Morgan Kaufmann Publishers, 2003. 6

[18] D. B. Hamming, "Error Detecting and Error Correcting Codes," *The Bell System Technical Journal*, vol. 26, no. 2, pp. 147–160, 1950. 8

[19] R. Katz, J. J. Wang, R. Koga, K. A. LaBel, J. McCollum, R. Brown, R. A. Reed, B. Cronquist, S. Crain, T. Scott, W. Paolini, and B. Sin, "Current radiation issues for programmable elements and devices," *IEEE Transactions on Nuclear Science*, vol. 45, no. 6, pp. 2600–2610, December 1998. 8

[20] Wikipedia, "Stuck-at fault — wikipedia, the free encyclopedia," 2006, [Online; accessed 11-October-2006]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Stuck-at_fault&oldid=79131164 15

[21] K. M. Butler and M. R. Mercer, "The influences of fault type and topology on fault model performance and the implications to test and testable design," in *DAC '90: Proceedings of the 27th ACM/IEEE conference on Design automation*. New York, NY, USA: ACM Press, 1990, pp. 673–678. 15

[22] E. Johnson, M. J. Wirthlin, and M. Caffrey, "Single-event upset simulation on an FPGA," in *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, T. P. Plaks and P. M. Athanas, Eds. CSREA Press, June 2002, pp. 68–73. 15, 16

[23] U. East, "SLAAC-1V user VHDL guide," Tech. Rep., October 2004. 16

[24] K. S. Morgan, "SEU-Induced Persistent Error Propagation in FPGAs," Master's thesis, Brigham Young University, August 2006. 30

[25] S. Kundu and S. M. Reddy, "Embedded totally self-checking checkers: A practical design," *IEEE Design and Test of Computers*, vol. 07, no. 4, pp. 5–12, Jul/Aug 1990. 32

[26] D. P. Siewiorek and R. S. Swarz, *"Reliable Computer Systems"*. A K Peters, 1998. 32

[27] B. C. C. Laboratory, "BYU FPGA Reliability Studies," 2006, [Online; accessed 28-October-2006]. [Online]. Available: http://reliability.ee.byu.edu/edif/ 46

Appendix A

Risk Analysis of the Number of Comparators

The events SD, SU, FP, and CE are used to determine the effectiveness of our chosen detection scheme. These metrics can also used to analyze the trade-offs in several of the parameters that impact the detection scheme. For example, from discussions in this work, it can be seen that using more comparators placed more frequently would result in a better time to detection. However, it should be noted that increased number of detectors would results in larger numbers of CEs and FPs. The purpose of this section is to determine the relative cost of each comparator in terms of additional FPs and CEs.

Table A.1 shows the number of comparators added to several designs and the number of FPs and CEs that occurred. This risk analysis shows that it is difficult to estimate the cost of a comparator in terms of CEs and FPs. The difficulty is a result of the mapping of the design to the actual FPGA. Some designs are easy to map and therefore require less routing. Other designs are more complicated and require the use of more programmable routing bits. This quantity of routing bits needed to add the comparators is a large factor in the number of CEs and FPs as shown from our earlier discussion of the causes of CEs and FPs. This can be seen by examining the difference between the 16-bit counter and the multiple counters design. Despite having the same number of comparators the number of CEs and FPs are very different. In fact, the multiple counters design has more FPs than the 30-bit counter which has 30 comparators added. Because the multiple counter design is much larger (322 slices as opposed to 30 slices) and more complex, the routing and packing of

the logic is not as compact. The routing is longer and more complex. This increase in routing causes the additional CEs and FPs. It is difficult to obtain a precise estimate of the increase in CEs and FPs per added comparator. A rough estimate based on the numbers collected in the simple designs shown in the table would result in the following conclusions: each additional comparator adds 16.07 CEs and 6.29 FPs. These numbers are very rough estimates as they are heavily influenced by the additional routing needed which is difficult to estimate. It would be very beneficial to conduct a more in-depth risk analysis to more accurately determine the cost of additional comparators. In future work, it would also be beneficial to explore the cost of additional comparators that are not located on the output lines of the circuit. For this work, all comparators have been placed on the outputs lines just before the OBUFs.

Table A.1: Risk analysis of additional comparators

Design	Slices	Comparators	CEs	FPs
3-bit counter	6	6	137	47
5-bit counter	12	10	180	117
16-bit counter	30	32	549	131
multiple counters	322	32	457	190
30-bit counter	59	60	486	114