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

Jonathan Johnson, William Howes, and Michael Wirthlin
Brigham Young University
Provo, UT 84606

Daniel L McMurtrey
Sandia National Laboratory
Albuquerque, NM 87114

Michael Caffrey, Paul Graham, and Keith Morgan
Los Alamos National Laboratory
Los Alamos, NM 87545

*Abstract*—It is well known that SRAM-based FPGAs are susceptible to single-event upsets (SEUs) in radiation environments. A variety of mitigation strategies have been demonstrated to provide appropriate mitigation and correction of SEUs in these environments. While full mitigation of SEUs is appropriate for some situations, some systems may tolerate SEUs as long as these upsets are detected quickly and correctly. These systems require effective error detection techniques rather than costly error correction methods. This work leverages a well-known error detection technique for FPGAs called duplication with compare (DWC). This technique has been shown to be very effective at quickly and accurately detecting SEUs using fault injection and radiation testing.

### TABLE OF CONTENTS



## 1. INTRODUCTION

Reprogrammable field programmable gate arrays (FPGAs) are increasingly being used in space-destined systems because of their reconfigurability, density, and performance [1], [2], [3]. They provide application-specific performance while preserving flexibility. The ability to repeatedly reconfigure an FPGA can extend mission lifetime by allowing a single system to perform various tasks at different times. In addition, the use of FPGAs reduces the overall non-recurring engineering (NRE) costs involved in producing such a system.

While FPGAs offer many advantages for space-based missions, they are susceptible to radiation effects. FPGAs are especially susceptible to single-event upsets (SEUs) because of the large number of internal memory elements. SEUs can occur within user flip-flops, application block memory, and the configuration memory of an FPGA. SEUs that occur in the configuration memory of an FPGA are the most problematic as these upsets may alter the functionality of the FPGA circuit.

FPGAs that are used in radiation environments must address this issue with appropriate mitigation techniques. The most common mitigation technique is triple modular redundancy (TMR) [4]. TMR can effectively mask all circuit errors caused by single event upsets. When used in conjunction with configuration scrubbing, TMR is an effective technique for mitigating upsets within the configuration memory [5], [4].

TMR, however, is very expensive in terms of area, power, and timing. At a minimum, full TMR requires 3X as many resources as an unmitigated design. The need for voters and other mitigation logic may increase the size of the circuit even greater than 3X [6]. To address this, some have proposed the use of partial TMR or other less costly mitigation techniques [7].

In some systems, full mitigation of SEUs may not be necessary. Instead, some systems are more interested in *detecting* upsets and using other system-level mechanisms for addressing the faults. In response to a detected upset, the system may take appropriate action such as repairing the configuration bitstream, discarding data, and/or resetting the FPGA. For example, an FPGA circuit may perform a series of complex arithmetic functions on a stream of data. If a fault has been detected, the computation can be stopped and performed again once the fault has been repaired. Systems like this tol-

1-4244-1488-1/08/$25.00 ©2008 IEEE
IEEEAC Paper #1255, Version 6 Updated 12/19/2007.
LA-UR-07-8359

erate output errors as long as they are recognized quickly [8].

This paper leverages a well-known technique called *duplication with compare* (DWC) for detecting upsets within an FPGA. As the name implies, this technique operates by duplicating circuit resources and comparing the results. This technique has been shown to detect upsets within the configuration memory *and* user flip flops quickly and accurately. The ability to quickly and accurately detect these faults was demonstrated through fault injection experiments as well as radiation testing.

This paper will begin by reviewing DWC and describing how it is used within FPGA circuits. Next, this paper will describe how a circuit can be augmented to perform DWC and discuss a number of implementation options. The fault injection architecture will be described along with the results from a number of designs that exploit DWC. The results from radiation experiments will also be described and compared with the fault injection results.

## 2. DUPLICATION WITH COMPARE (DWC)

Many digital systems employ error detection techniques to improve system reliability. Error detection strategies are used to detect events that need correction or repair. Many different error detection techniques have been demonstrated. Example error detection techniques include information coding techniques such as parity, temporal redundancy (i.e. performing the same computation twice), and hardware redundancy. These techniques have been used in virtually all aspects of computing architectures.

The most common way of detecting SEU induced errors within an FPGA is through readback. Readback is the process of reading the configuration memory within the device. To detect errors, this memory is compared against a golden configuration memory that is stored in an external memory. If there is a difference in the configuration memory, then an error has been detected and corrective measures such as reconfiguration can be taken. While effective at detecting errors, this method of error detection has a number of drawbacks. First, it is unable to detect errors that occur in dynamic user-defined memories (i.e. flip-flops or RAMs); only errors in the configuration bitstream are corrected. Second, there is a delay from the time an upset occurs to the time when it is detected by readback with comparison. The worst case delay is the time it takes for a full readback cycle to occur [9] (this can be over hundreds of milliseconds). Third, systems that implement readback with comparison require additional external circuitry that is generally implemented in dedicated hardware.

This work leverages a well-known error detection technique called duplication with compare (DWC) as an alternative error detection technique for FPGAs. DWC is a simple hardware redundancy to detect errors in the circuit. Specifically, DWC uses two identical copies of a circuit and compares the outputs of these circuit copies to determine if an upset has occurred. The comparator circuit detects differences in the operation of the two circuits and signals the system with an error flag.

DWC was chosen as an error detection technique for several reasons. First, it is relatively easy to apply to any circuit. Automated design tools can be created for adding this technique to a given circuit. Second, it can be used to detect a variety of errors including configuration upsets (within the sensitive cross section of the device), transient errors, and upsets within user flip-flops. Third, it can detect errors immediately and allow the system to quickly respond to circuit problems. Fourth, it requires limited external hardware support.

## 3. IMPLEMENTING DUPLICATION WITH COMPARE

Implementing DWC within a circuit is relatively straightforward. To begin, the circuit is duplicated to create two identical designs (called domains). Circuit duplication is accomplished by duplicating each primitive instance in the original design. Signal nets are also duplicated and the original connectivity of the design is copied in the second circuit domain. In the simplest form of DWC, the entire circuit is duplicated (i.e. full duplication).

While full duplication provides the greatest coverage for error detection, it is also possible to apply partial duplication to a design. Partial duplication may be necessary for two different reasons. First, partial duplication may be necessary in cases where the target FPGA design does not have sufficient resources to accommodate full duplication. For example, there may be insufficient I/O resources or block memories within the device to permit full duplication. Second, different logic regions within a design may be more important than others and partial duplication can be used to detect errors in regions of interest rather than within the entire design. For example, detection circuitry may be needed in the feedback or persistent [7] sections of a design rather than throughout the entire design.

In addition to circuit duplication, DWC requires the insertion of comparators and a method of combining comparator signals for the external system. The rest of this section will summarize these issues and the trade-offs associated with generating error detection signals.

### *Selecting Locations for Comparator Insertion*

The second step in applying DWC is to select net locations where the comparators will be inserted. Ideally, comparators are applied to each net of the circuit. However, adding comparators to all nets is very costly and not possible due to architectural constraints imposed by the FPGA. Instead, a sub-set of circuit nets must be selected for comparison.

A common way of inserting comparators is to place them at

the final outputs of the circuit. Any discrepancies between the two circuits that are seen at the outputs are detected when they occur. A disadvantage of this technique is that errors are not detected until they propagate to the circuit outputs. It may take a long time for an error within the middle of a circuit to manifest itself on the outputs. Placing more comparators throughout the design decreases mean time to detection.

One way of identifying errors more quickly is to insert the comparators within the feedback portions of the circuit. Errors that occur in the feedback of a design, called persistent errors [7], pose two problems to the system: persistent errors can take more time to manifest themselves and it is more difficult to recover from persistent errors. Placing comparators within the feedback portion of a design allows instantaneous detection of persistent faults [10]. This work will investigate two techniques for comparator insertion: insertion at the circuit outputs and insertion within feedback portions of the circuit as well as at the outputs.

*Dual Rail Comparator*

The purpose of the comparators is to check for differences in the two domains of a duplicated circuit. A simple comparator can be constructed using an exclusive or (XOR) gate. When the input signals to an XOR gate do not match, the gate generates a logic '1'. This gate can be used to generate the local error detection flag for two signals.

While this simple XOR gate can perform error detection, it is susceptible to failure. To protect against this, a totally self-checking comparator is used [11], [12]. A totally self-checking comparator uses dual rails (i.e. two outputs) to indicate an error as seen in Figure 1. Totally self-checking comparators generate invalid error codes if a fault has occurred within the comparator circuitry. The dual-rail comparator architecture is a self-checking checker that is simple to implement.

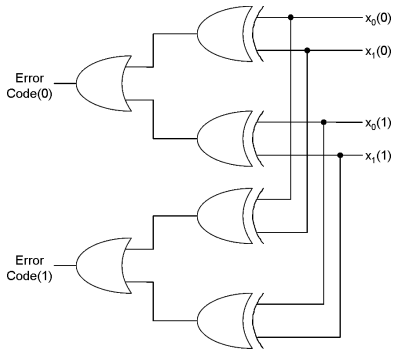


**Figure 1**. Dual rail checker

When the dual-rail comparator code is "00", the circuit being checked and the comparator circuit are error free. A codeword of "11" indicates that the input circuit is in error. Error codes "01" and "10" indicate that there is an error within the checker systems.

*Merging Local Intermediate Error Flags*

The final step in implementing DWC is to merge all of the local intermediate error flags from each comparator into a single error code output. If the design contains hundreds of local error flags (i.e. if many comparators were added) then a large reduction network is needed to reduce the error flags to a single output. A variety of merging techniques can be used including binary tree and daisy-chain reduction.

## 4. MEASURING DETECTION COVERAGE

While the DWC technique is fairly straightforward to implement, it is not clear how effective it is in detecting SEUs within an FPGA. An important goal of this work is to identify the effectiveness of this technique using fault injection. A hardware fault injection simulator was used to evaluate the effectiveness of the technique on the test designs in the presence of SEUs. This section will describe the fault injection architecture followed by a discussion of the error detection occurrences that can happen with the architecture.

*Fault Injection Architecture*

A fault injection simulator was created for this work to measure the error detection coverage of DWC. Fault injection enables the full exploration of the effectiveness of an error detection scheme for an FPGA-based design. This fault injection simulator is based on a simulator [13] that was originally designed at BYU to measure the sensitivity of bits to SEUs in FPGA-based systems. It was adapted to additionally report error detection events in designs augmented with the DWC technique. The simulator is based on the SLAAC-1V FPGA computing board [14].

As seen in Figure 2, the 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 is corrupted and tested. X0 contains 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.

Two major modifications were made to the original sensitivity simulator for this work. First, support for duplicated outputs on the DUT was added. This allows the outputs of both circuit domains to be tested in the X0 control circuit. Second, the internally generated detection error flags were added and routed to X0. These features are shown in Figure 2.

Within a simulation run, all configuration bits of the FPGA except the BRAM content bits are tested using the SelectMAP interface. This is a total of $5,810,024$ bits for the XCV1000 FPGA. The basic operation of a fault injection cycle is as follows:

1. Corrupt a configuration bit in X1,
2. Run the system with a set of random inputs for a set amount of time,

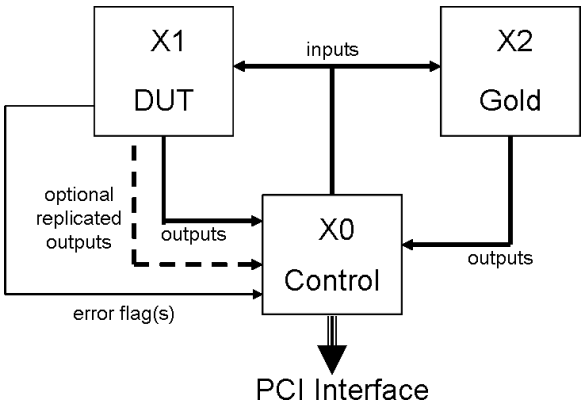


**Figure 2**. Modified simulator hardware

3. Check outputs for errors in X0,
4. Check error detection flags in X0, and
5. Repair configuration bit.

This cycle is repeated for all the configuration bits that are to be tested. The occurrence of actual output errors is compared with the status of the error detection flags. The software simulator is able to query the X0 device between each cycle to determine this information for each bit tested. The results for each bit are logged and collected to determine error detection coverage.

*Error Detection Occurrences*

This fault injection simulator generates a number of distinct error events that are used to classify the behavior caused by each upset configuration bit. Each configuration bit can be classified 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 that 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.

The output error (EO) occurrence is detected by comparing the duplicated outputs of the DUT with the output of the "golden" design. Each set of outputs is compared independently against the output of the "golden" design. If either of the outputs is incorrect, an output error occurrence (EO) is recorded. In equation form it is

$$EO = (DUT\_OUT_0 \neq GOLD \; or \; DUT\_OUT_1 \neq GOLD). \quad (1)$$

The functional logic error (EF) occurrence and the detector circuit error (ED) occurrence are detected by checking the error flags generated in the DUT. The error flag signal is routed to X0 where the checking of the signal and classification of occurrence is done.

**Table 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 is a 2-bit error code signal because of the use of a dual-rail comparator architecture. 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 an 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 1. Table 2 summarizes the occurrences and the reason that each would be recorded.

**Table 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 |

## 5. Categorizing Error Detection Events

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 five categories of events that can occur when a configuration bit has been upset. The five categories are insignificant, sensitive detected, sensitive undetected, false positive, and checker errors. The specific combination of error detection occurrences (see Table 2) that happen for each upset bit determines the error detection event category assigned to that bit. The five event categories will be used throughout the remainder of this paper to quantify the coverage of the DWC error detection technique.

The first type of event that can 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 (IB). 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 2) by the logic formula,

$$insignificant\ event(IE) = \overline{EO} \cdot \overline{EF} \cdot \overline{ED}. \quad (2)$$

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. All other events that occur are significant events and are caused by the upset of significant bits (SB).

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

$$sensitive\ detected\ event(SD) = EO \cdot EF. \quad (3)$$

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,

$$sensitive\ undetected\ event(SU) = EO \cdot \overline{EF}. \quad (4)$$

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 positive events are given by the equation,

$$false\ positive\ event(FP) = \overline{EO} \cdot EF. \quad (5)$$

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 an upset causes exactly one of the error code bits to go high when no output error has occurred. 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,

$$checker\ error\ event(CE) = \overline{EO} \cdot ED. \quad (6)$$

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

**Table 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, will be used to calculate metrics that measure the effectiveness of error detecting schemes.

*Metrics to Determine Error Detection Effectiveness*

Two metrics will be used to demonstrate the effectiveness of DWC as an error detection technique. They are the percentage of correctly diagnosed significant bits (CDSB) and the percentage of correctly diagnosed configuration bits (CDCB).

The first metric, CDSB, is the percentage of significant bits (SB) that are correctly diagnosed by the error detection circuitry. The reader will recall that only sensitive undetected (SU) events are incorrectly diagnosed. Thus the fraction of correctly diagnosed significant bits is simply the total number of significant bit events minus sensitive undetected events divided by the total number of significant bit events. In equation form CDSB is

$$\begin{aligned} CDSB(\%) &= \frac{SB - SU}{SB} * 100. \\ where, & \\ SB &= SU + SD + CE + FP \end{aligned} \quad (7)$$

In other words, this metric measures the fraction of bits that do not cause SUs in terms of all the bits that cause variations in the system behavior (i.e. significant bits), whether the variation is in the error detection circuitry or the regular logic. This metric is beneficial because it is device independent in that it does not depend on the total number of configuration bits in a device. In other words, CDSB can be used to measure the effectiveness of a detection technique on a given design independent of the size of the device.

The second metric, CDCB, is the percentage of *all* configuration bits (significant bits + insignificant bits) that are correctly

diagnosed by the error detection circuitry. Unlike CDSB this metric is device dependent because it includes insignificant bits (IB) which are equally likely to experience SEUs. In other words, since all bits are included, CDCB shows the percentage of errors correctly diagnosed by the error detection circuitry from the device's perspective. Since insignificant bits are also correctly diagnosed by the error detection circuitry, the equation to calculate CDCB is quite similar to Equation 7 for CDSB. It is simply the total number of bits (TB) minus sensitive undetected bits, divided by the total number of bits. In equation form this is

$$\begin{aligned} CDCB(\%) &= \frac{TB - SU}{TB} * 100. \\ where, & \\ TB &= total\ device\ configuration\ bits \\ &= SB + IB \end{aligned} \tag{8}$$

CDCB is useful because it can be used to calculate the mean time to an incorrectly diagnosed event, or mean time to failure (MTTF). MTTF can be calculated with the equation

$$MTTF = \frac{1}{SEU\ Rate \times (1 - CDCB)}. \tag{9}$$

Both CDSB and CDCB can be used to show the effectiveness of error detection techniques. These metrics will be used in conjunction with the defined events to analyze the effectiveness of DWC as an error detection technique.

## 6. Fault Injection Results

*Test Designs*

A suite of benchmarks was created to demonstrate the effectiveness of the error detection methodology presented in this paper. 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 a 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 shows how well DWC can work in the face of resource constraints.

A *DSP Kernel* was chosen to represent a real world digital signal processing circuit and to test the effectiveness of partial DWC. This particular design is interesting because the target chip for our tests (XCV1000) does not have enough block ram resources to allow full duplication of the design. The block rams were left unduplicated while applying DWC to the rest of the design in order to evaluate partial DWC coverage.

A modified version of the *synthetic* design was also used. It is similar to the first version but includes more logic to use up the resources of the chip more fully. This was done to facilitate radiation test validation of the fault injection results.

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

All of these designs were tested with the dual-rail comparator architecture. The first four designs were tested with comparators placed only at the outputs, and the clock and reset lines were left unduplicated. The DSP Kernel and the modified synthetic design were tested with comparators placed not only at the outputs but also at feedback locations spread throughout the designs in order to facilitate a quick error detection time. Also, the clock and reset lines in these last two designs were duplicated to provide better error detection coverage.

*Results*

Table 4 shows the results that were obtained from the suite of benchmark designs. The percentages are given for each error detection event in terms of a percentage of the total significant bits (SB). This section will briefly consider the causes of events in each category. A more detailed explanation of their causes can be found in the thesis *Using Duplication*

**Table 4**. 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%) |
| DSP Kernel[a] | 12,286 | 1,631,780 | 34,996 (2.15%) | 1,122,363 (68.78%) | 415,934 (25.49%) | 58,487 (3.58%) |
| Modified Synthetic | 12,286 | 1,098,322 | 2669 (0.24%) | 1,077,504 (98.11%) | 5,095 (0.46%) | 13,054 (1.19%) |

[a]The excessive false positives and checker errors in the DSP Kernel are artifacts of partial duplication

*with Compare for On-line Error Detection in FPGA-based Designs* [9].

*Sensitive Detected Errors*—The 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 comparator circuitry detects these errors and signals that an error has been detected with the appropriate error code output.

The number of SD bits is primarily determined by the size, placement, and routing of the system. These type of bits are correctly diagnosed by the detection circuitry. Any effects caused by these bits are correctly reported to the external system.

*Sensitive Undetected Errors*—Sensitive undetected errors occur when an error causes the outputs of the system to be incorrect but the error detection circuit fails to detect this error. These errors are undesirable as it indicates that the error detection approach is not working. This type of error is caused by an SEU in a portion of the design that does not get compared by the comparator circuitry (i.e. upsets in unduplicated circuitry).

There are several reasons why some circuitry will not be duplicated. First, there may not be enough I/O resources to duplicate the inputs or outputs. Upsets within unduplicated inputs will cause failures in both copies of the duplicated circuit as the inputs drive both circuits. Upsets within unduplicated outputs will also cause a problem because the internal comparators are placed *before* the single outputs. Second, it may not be possible to duplicate global resources such as clocks and resets. Upsets in these global resources will negatively affect both copies of the circuit. Third, there may not be enough resources to fully duplicate all logic. In this case, upsets within unduplicated logic are not detectable.

The presence of sensitive undetected events in Table 4 is due to the lack of full circuit duplication. First, the inputs for all designs are unduplicated. Further, the clock and resets are not duplicated for all designs except the DSP kernel and the modified synthetic design. The relatively high presence of sensitive undetected events in the DSP kernel is due to the fact that the BRAMs in the design are not duplicated.

*False Positives*—False positives occur when an error affects both domains of the comparator circuitry, causing both to output a positive error signal. These events occur when upsets occur in the routing to the comparator circuitry [9]. False positives can also occur when output errors are masked internally before leaving the device.

False positives indicate that the output is in error when in fact it is not. False positives are not detrimental to the system. They are, however, misleading and may result in unnecessary repair. Reducing false positives is a secondary goal of error detection schemes. Routing to the comparators and the number of comparators are the key determinants of false positives.

*Checker Errors*—Checker errors occur when SEUs affect the comparator circuitry or the circuitry that merges the multiple local error flags into a final error code output. Checker errors, like false positives, are determined by the routing and number of comparators. Checker errors do not pose a problem for error detection schemes and do not need to be reduced.

*Effectiveness Metrics for Benchmark Designs*

The CDSB (see Equation 7) and CDCB (see Equation 8) effectiveness metrics were calculated from the fault injection results to measure the effectiveness of DWC as applied to the designs in the benchmark suite. They are shown in Table 5.

From these results, it is shown that when full design duplication is used, DWC has an estimated 99.95% to 99.99%

**Table 5**. Effectiveness calculations of the benchmark designs

| Design | CDCB | CDSB |
|---|---|---|
| Counters200 | 99.99% | 99.75% |
| Synthetic | 99.97% | 99.73% |
| QPSK | 99.98% | 99.32% |
| Triple | 99.95% | 99.77% |
| DSP Kernel* | 99.40% | 97.86% |
| Modified Synthetic | 99.95% | 99.76% |
| Average | 99.87% | 99.37% |

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.

In some cases, it is not possible to fully duplicate the design (i.e. due to insufficient resources). The DSP kernel uses a partial duplication strategy due to a limitation in the number of Block RAMs within the device. As expected, the test results from the DSP kernel show higher rates for sensitive undetected (SU) events. Because part of the design does not have detection, sensitive upsets will occur that are not detected by the detection circuitry.

The DSP kernel also showed a higher number of FP and CE events. The increase in these events is also due to partial duplication but the reason is not as clear. In a partially duplicated system, there are often places in the circuit when two copies of the logic (i.e. duplicated logic) feed into a single copy of the logic (i.e. unduplicated). As seen in Figure 3, the output from only one of these two copies can be fed forward, so one of the two copies is essentially a "dead branch" that does not affect the behavior of the single, unduplicated logic. The dead branches do, however, affect the outputs of the error detection circuitry because error checkers are placed at the end of the dead branch. Upsets in the dead branches are detected but do not result in erroneous circuit output (i.e. they result in FP and CE events).

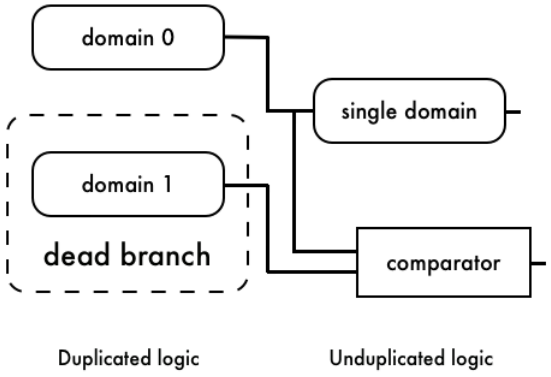


**Figure 3**. Dead Branch Example

When using partial duplication in a deployed system, the increased presence of false positives and checker errors should be considered. It should be expected that not all of the events reported by the detection circuitry as errors will actually cause output errors. Further, lower effectiveness of error detection should be expected. Although there are limitations associated with partial duplication, both full and partial DWC have been shown to provide very high fault coverage for those areas of the circuit that are duplicated and compared.

## 7. Radiation Experiments

Radiation effects tests were performed at the Crocker Nuclear Laboratory proton accelerator facility using two of the designs from the benchmark set: the DSP Kernel and the modified synthetic design. The purpose of the tests was to validate the results obtained through the fault injection simulator. This section will summarize the test architecture, explain the classification of error detection events in the accelerator test environment, present the results of the experiments, and show that they correlate with the previously presented fault injection results.

*Radiation Test Architecture*

Like the fault injection simulator, the radiation test architecture is based on the SLAAC-1V FPGA computing board. The hardware setup is identical to the fault injection simulator except that a configuration controller is added to constantly scrub and report SEUs that occur in the configuration bitstream of the DUT via the SelectMAP interface. The hardware setup is shown is Figure 4.

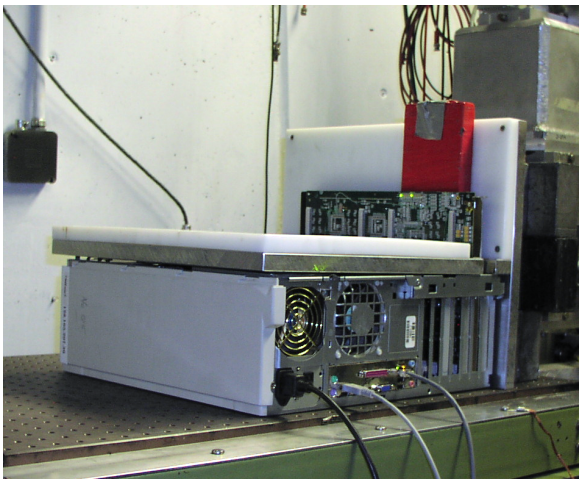

**Figure 4**. Radiation Test Setup

The software for the radiation test setup is different from the fault injection simulator software. Instead of injecting faults in the configuration bitstream and observing their effects, the radiation test software waits for error detection events to happen. As the DUT FPGA is irradiated, SEUs that occur in the configuration bitstream cause error detection events that the software detects and logs. Concurrently, the configura-

tion controller continuously logs and repairs the SEUs in the DUT.

The radiation source was a $63.5$ MeV proton beam. Over the course of the test, the DUT received a total fluence of $3.14 \times 10^{11}$ p/cm$^2$ and a total dose of $42.2$ krad.

*Event Classification*

Error detection events are classified in the radiation test software much the same way as they are in the fault injection software. The radiation test software knows that an error detection event has occurred when the occurrence of at least one of the three error detection occurrences that are listed in Table 2 (EO, EF, ED) is reported to the X0 FPGA through the PCI interface. The software then waits a sufficient amount of time for all error detection occurrences associated with the current SEU to happen. Then, it queries the X0 FPGA through the PCI interface to find out which occurrences actually happened. Finally, event categorization takes place according to the flow chart in Figure 5.

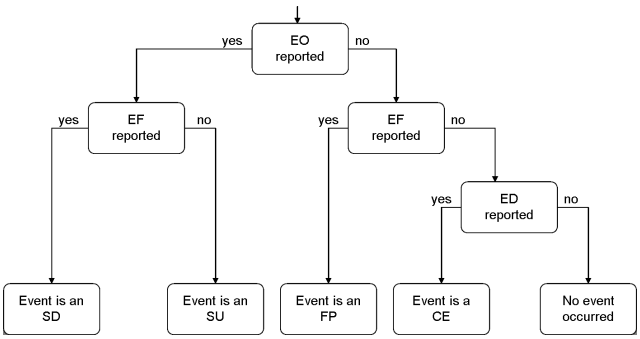


**Figure 5**. Flow chart for software error detection event categorization

*Results*

The results of the radiation test for the DSP Kernel and the modified synthetic design are shown in Table 7. The percentages are given for each error detection event in terms of a percentage of the total significant bits tested. The CDSB effectiveness metric calculations for these results are shown in Table 6 and compared to the previously presented fault injection CDSB calculations. This comparison shows that the radiation test results closely match the fault injection simulation results.

**Table 6**. Radiation Test Effectiveness Metric Calculations

| Design | Radiation Test CDSB | Fault Injection CDSB |
|---|---|---|
| DSP Kernel* | $98.02\%$ | $97.86\%$ |
| Modified Synthetic | $99.85\%$ | $99.76\%$ |

## 8. CONCLUSION

In this paper, the DWC error detection method is shown to be an effective way of detecting errors in FPGA-based systems. The simplicity of the DWC technique allows it to be applied with an automated CAD tool. Radiation test results show that the DWC technique can detect approximately $99.85\%$ of all circuit errors. This very good coverage is achieved at the cost of an approximate $2\times$ increase in design size. These factors make the DWC technique a viable error detection method for systems that can tolerate known temporary deviations from normal operation.

There is potential for much future work building on this study. One major area of concern is the reporting of persistent errors that occur in design feedback paths. Persistent errors cannot be corrected in the same manner as non-persistent errors. Generally, a full system reset is required to recover from a persistent error. Because of this, future modifications of the automated tool for applying DWC could provide the option of separating the error detecting lines into non-persistent and persistent error signals.

Another interesting application of DWC would be to try a hybrid TMR/DWC approach to protecting a design from SEUs. The persistent cross-section of the design could be augmented with TMR and the rest of the design could be protected with DWC to provide elimination of persistent errors and detection of non-persistent errors.

The DWC technique could also be combined with TMR in a different manner to provide full error mitigation as well as error detection. Currently, TMR must be used in combination with periodic scrubbing in order to prevent upsets from accumulating in the configuration bitstream. TMR by itself masks output errors but does not report their occurrence. Augmenting a design with both TMR and DWC would provide both error correction and detection. Such a system could use interrupt-driven scrubbing rather than periodic scrubbing.

## ACKNOWLEDGMENTS

This work was supported by the Los Alamos National Laboratory under the Soprano and CFE projects.

## REFERENCES

[1] D. Ratter, "FPGAs on Mars," Xilinx, Tech. Rep., August 2004, xCell Journal #50.

[2] 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.

[3] 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 Sig-*

**Table 7**. Results from Radiation Test

| Design | Bits Tested | Significant Bits | SUs (%) | SDs (%) | FPs (%) | CEs (%) |
|---|---|---|---|---|---|---|
| DSP Kernel* | 29,824 | 7,360 | 146 (1.98%) | 5,235 (71.13%) | 1,712 (23.26%) | 267 (3.63%) |
| Modified Synthetic | 33,079 | 4,733 | 7 (0.15%) | 4,632 (97.87%) | 20 (0.42%) | 74 (1.56%) |


*nal Processing (DSP 2002)*, vol. 2, 2002, pp. 711–717.

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

[5] 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] N. Rollins, M. J. Wirthlin, and P. S. Graham, "Evaluation of power costs in triplicated fpga designs," in *Proceedings of the Military and Aerospace Programmable Logic Devices International Conference (MAPLD)*, Washington, D.C., September 2004.

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

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

[9] D. L. McMurtrey, "Using Duplication with Compare for On-Line Error Detection in FPGA-Based Designs," Master's thesis, Brigham Young University, December 2006.

[10] R. Katz, R. Barto, P. McKerracher, B. Carkhuff, and B. Koga, "Seu hardening of field programmable gate arrays (fpgas) for space applications and device characterization," *IEEE Transactions on Nuclear Science*, vol. 41, no. 6, pp. 2179–2186, 1994.

[11] 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.

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

[13] 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.

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


## BIOGRAPHY

**Jonathan Johnson** is a senior undergraduate Computer Engineering student at Brigham Young University. His interests include FPGA reliability and embedded systems programming.

**William Howes** is a senior undergraduate Computer Engineering student at Brigham Young University. His interests include FPGA reliability studies and embedded systems development.

**Michael Wirthlin** received his B.S. and Ph.D. degrees from Brigham Young University (BYU) in 1992 and 1997 respectively. After completing his Ph.D., he worked as a Staff Research Engineer in the Systems Architecture Laboratory at National Semiconductor Corporation in Santa Clara, CA. He is currently an Associate Professor in the Department of Electrical and Computer Engineering at BYU. His research interests include Configurable Computing Systems, FPGA reliability, fault tolerant computing, high-level synthesis, and computer-aided design for application-specific computing.

**Daniel L McMurtrey** received his B.S. and M.S. degrees from Brigham Young University in 2004 and 2006, respectively. He is currently a hardware/software engineer at Sandia National Laboratories.

**Michael Caffrey** is a member of technical staff at Los Alamos National Laboratory. His current work involves reconfigurable computing in space. He has interests in reliability and high performance digital signal processing. Mr. Caffrey obtained an M.S. degree in electrical engineering from the University of New Mexico in 1995.

**Paul Graham** is a project leader in the International, Space and Response Division of Los Alamos National Laboratory. His research interests include reconfigurable computing with FPGAs, electronic design automation tools for FPGAs, radiation effects on electronics, fault-tolerant computing, and embedded computing. Paul received his Ph.D. from Brigham Young University in electrical engineering in 2001. He has authored many technical papers for conferences, journals, and book chapters in the areas of reconfigurable computing, reliability, and radiation effects on FPGAs. He is also a co-author of the recent book Reconfigurable Computing: Accelerating Computation with Field Programmable Gate Arrays published by Springer.

**Keith Morgan** is currently a technical staff member at Los Alamos National Laboratory in the Space Data Systems group within the International, Space, and Response division. Morgan received a Masters degree in electrical engineering from Brigham Young University (BYU) in 2006. He also received a Bachelors degree in computer engineering from BYU in 2005. His interests are in reliability and reconfigurable architectures. In 2005 Morgan married Natalie Jane Christensen from Lindon, UT. Morgan was born and raised in Medford, OR.