Impact of Soft Errors on Large-Scale FPGA Cloud Computing

Andrew M. Keller and Michael J. Wirthlin
NSF Center for Space, High-Performance, and Resilient Computing (SHREC)
Deptartment of Electrical and Computer Engineering
Brigham Young University
Provo, Utah
andrewmkeller@byu.edu,wirthlin@byu.edu

ABSTRACT

FPGAs are being used in large numbers within cloud computing to provide high-performance, low-power alternatives to more traditional computing structures. While FPGAs provide a number of important benefits to cloud computing environments, they are susceptible to radiation-induced soft errors, which can lead to silent data corruption or system instability. Although soft errors within a single FPGA occur infrequently, soft errors in large-scale FPGAs systems can occur at a relatively high rate. This paper investigates the failure rate of several FPGA applications running within an FPGA cloud computing node by performing fault injection experiments to determine the susceptibility of these applications to soft-errors. The results from these experiments suggest that silent data corruption will occur every few hours within a 100,000 node FPGA system and that such a system can only maintain high-levels of reliability for short periods of operation. These results suggest that soft-error detection and mitigation techniques may be needed in large-scale FPGA systems.

CCS CONCEPTS

• **Computer systems organization → Reliability**; *Availability*; • **Hardware → Fault models and test metrics**; **System-level fault tolerance**; *Board- and system-level test*; *Error detection and error correction*; *Failure prediction*; *Failure recovery, maintenance and self-repair*; *Redundancy*.

KEYWORDS

FPGA cloud computing; FPGA data centers; soft error rate, SER; single event upset, SEU; architectural vulnerability factor, AVF; fault injection; critical bit; reliability; recovery; Intel FPGA; mission time

ACM Reference Format:
Andrew M. Keller and Michael J. Wirthlin. 2019. Impact of Soft Errors on Large-Scale FPGA Cloud Computing. In *The 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '19), February 24–26, 2019, Seaside, CA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3289602.3293911

FPGA '19, February 24–26, 2019, Seaside, CA, USA

ACM ISBN 978-1-4503-6137-8/19/02…$15.00
https://doi.org/10.1145/3289602.3293911

1 INTRODUCTION

Field programmable gate arrays (FPGAs) are increasingly being used in cloud computing environments to perform application-specific computation. FPGAs offer a configurable fabric that can be used to accelerate a variety of important applications. In some applications, FPGAs provide higher performance and power efficiency than super-scalar CPUs [27] or GPUs [15]. The large number of nodes found within modern cloud computing suggests that many FPGAs will be needed in FPGA-based cloud computing systems.

With large-scale deployment of any technology, there is increased risk of a single node failing, which can jeopardized the integrity or stability of the overall system. Although the failure rate for a single node may be very small, the failure rate of large-scale systems increases linearly with the number of devices. This scaling has been seen in DRAM [22], microprocessors [14], and FPGAs [19]. An important failure mechanism for FPGAs is caused by radiation-induced soft errors. These errors do not cause permanent damage to the device, but they can modify the internal state or memory of the system [1]. Since FPGAs contain a large amount of memory to configure routing, logic, and other aspects of a design, upsets within the memory of an FPGA can corrupt more than just the data stored in sequential logic elements – such upsets can change the behavior of the circuit configured on the device. The effects of soft errors on single FPGA systems are well understood [7] and methods for addressing these effects have been developed for FPGAs operating in harsh radiation environments such as space.

This paper studies the impact that soft errors have on large-scale FPGA systems within cloud computing. Fault injection is performed on an FPGA system that is tightly coupled with a CPU host. The evaluated device is an Intel Stratix V FPGA, which has been used in cloud-scale accelerated architectures [6]. Through fault injection, the soft error rates of several data-center-like FPGA designs are estimated. Collected data is used to appropriately scale soft error rates to a realistic large-scale FPGA system. Silent data corruption (SDC) is identified as the dominant failure mechanism. SEU detection and recovery approaches are discussed. The most vulnerable design, Mandelbrot, demonstrated a 11.3% vulnerability to soft errors. Running this application, a hundred-thousand node system operating at a high altitude in Denver, Colorado, would experience an SDC every 3.75 hours. Without any methods to detect and address soft errors, such a system could only operate for two minutes with a 99% probability of no SDC.

2 FPGAS IN THE CLOUD

FPGAs are being used in cloud computing because of their flexibility, performance, and energy efficiency. Unlike CPUs and GPUs, FPGA

resources can be custom configured for a specific target application. As a result, applications running on an FPGA can make better use of resources to exploit more parallelism, yielding higher performance and better energy efficiency. In [15], the Intel Stratix 10 FPGA is shown to provide 60% higher performance at 2.3× better energy efficiency (i.e., performance per watt) than the Titan X Pascal GPU for a deep neural network application. Using FPGAs in data centers makes these benefits available to a wide variety of applications.

FPGAs can accelerate many high performance computing (HPC) applications. In science, business, and everyday living, applications exist that require HPC. In [3], several Intel Stratix V FPGAs are used to accelerate DNA sequencing – an application in Genomics [21]. In [25], financial applications are accelerated using FPGAs. In [8], Microsoft accelerated the serving of deep neural networks for real-time artificial intelligence on Intel Stratix 10 FPGAs. Many other cloud-computing applications are being accelerated using FPGAs.

Several companies are actively deploying large-scale FPGA systems. Microsoft entered the FPGAs-in-data-center scene early with their launch of Catapult in 2014. They revamped their architecture in 2016, at which point they had 50,000 Stratix V GS D5 FPGAs deployed in their test system [6]. As of August 2017, Microsoft has deployed hundreds of thousands of FPGAs in their data centers [11]. Amazon Web Services (AWS) followed suit with their announcement of FPGA cloud instances in November 2016, which became generally available in April of 2017. As of June 2018, FPGA instances are now available from four different ASW regions–US East (N. Virginia), US West (Oregon), EU (Ireland), and AWS GovCloud (US). They offer instances with up to eight Xilinx Virtex UltraScale Plus FPGAs and currently partner with sixteen companies to provide cutting edge FPGA applications and services. Other companies, such as Baidu, Nimbix, and Micron, are also involved in using FPGAs for cloud computing. It is likely that FPGAs will continue to be used in data centers for cloud computing and that the number of deployed FPGAs will continue to increase [10].

FPGAs in data centers are typically reprogrammable SRAM-based FPGAs closely coupled with a CPU host as depicted in Figure 1. SRAM-based FPGAs use static volatile memory to store device configuration, making it possible to reprogram the device an unlimited number of times. The FPGA and host are usually connected with a high bandwidth interface such as PCIe. Large memory storage with DRAM is often available for acceleration tasks and is shared between the FPGA and processor to facilitate communication between the FPGA and processor. Multiple FPGAs can be connected to the same host or be networked together as part of an advanced acceleration architecture [6]. Some FPGA resources are dedicated to external interfaces, but most are made available to the main acceleration task, often referred to as the kernel or role.

To gather soft error sensitivity data, a single node FPGA system, similar to that shown in Figure 1, will be used for the experiments in this paper. The example node consists of a single host and a single FPGA accelerator board. The host is a Dell Precision T7510 server, equipt with two Intel Xeon E5-2609 processors and 16 GB of ECC-protected RAM, operating Window 10 Professional. A Terrasic DE5-net FPGA accelerator board with the Intel Stratix V GX A7 FPGA and 4 GB of RAM is connected to the host via a PCIe 8× connection. FPGA application designs are implemented on this node using OpenCL and a Terrasic provided board support package.

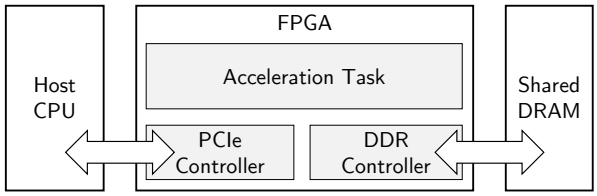

Figure 1: Typical FPGA Data Center Node

Table 1: Neutron SER for 28-nm FPGAs

Device	Stratix V GX A7	Kintex 7 325T
CRAM FIT/Mbit	63	74 ± 18%
CRAM Bits	~99,000,000	~73,000,000
CRAM FIT/Device	6,200	5,400
CRAM MTTU NYC	18.3 years	21.2 years

3 SOFT ERRORS IN FPGAS

Any observable change in a microelectronic device caused by a single energetic particle strike is known as a single event effect or SEE [1]. SEEs in terrestrial environments are primarily caused by high-energy neutrons, thermal neutrons, and alpha particles [5]. The most common SEE in FPGAs are single event upsets (SEUs) that cause flip-flops and other state elements to change their value. Although non-destructive, SEUs can have a significant effect on the operation of an FPGA design and the data it produces [28].

3.1 Soft Error Rates

The soft error rate, or the rate at which soft errors occur, is often measured in terms of failures in time (FIT) and mean time to failure (MTTF). FIT is defined as the average number of failures that occur within a billion hours of operation [1]. MTTF is the inverse of FIT scaled to a unit of time such as years; 1000 FIT roughly corresponds to a 100 year MTTF. Soft error rates for large memory components are often reported in terms of FIT per megabit, (i.e., 10^6), or FIT/Mbit. Neutron FIT rates are typically normalized to the amount of radiation present outdoors in New York City (NYC) at sea level during average solar activity. In these conditions, approximately 13 high-energy, (i.e., greater than 10 MeV), neutrons pass through a square centimeter of area every hour (13 $cm^{-2}h^{-1}$).

The raw terrestrial neutron FIT rates for two different 28-nm SRAM-based FPGAs are shown in Table 1. These devices contain large amounts of configuration memory or CRAM. For each device, the raw CRAM FIT rate per megabit is shown. CRAM FIT rate estimates for the Intel Stratix V and the Xilinx Kintex 7 come from [13] and [29] respectively; the number of CRAM bits are taken from vendor tool reports. Since both FPGAs are based on 28-nm technology, their estimated FIT rates are similar. The mean time between a single SEU occurring in a single Stratix V is 18 years.

The mean time to upset of a CRAM bit in a multiple FPGA system can be computed by dividing the mean time to upset (MTTU) of a single FPGA by the number of FPGAs in the system. The MTTU of a CRAM for different sized systems using the Intel Stratix V GX A7 FPGA is shown in Table 2. With one-hundred thousand FPGAs deployed, an upset occurs on average once every one and a half

Table 2: Stratix V GX A7 Mean Time to Upset at NYC SER

FPGAs	Years	Days	Hours	Min.	Sec.	Total (Sec.)
1	18	127	15	28	27	6E+8
10	1	304	23	8	50	6E+7
100		67	0	30	53	6E+6
1,000		6	16	51	5	6E+5
10,000			16	5	6	6E+4
100,000			1	36	30	6E+3
1,000,000				9	39	6E+2
10,000,000					57	6E+1

Table 3: Neutron flux at various locations

Location	Elevation	Relative Neutron Flux
Seattle, WA	160 ft	1.05
Moscow, Russia	490 ft	1.14
Chicago, IL	590 ft	1.19
Denver, CO	5280 ft	3.76
Los Alamos Natl. Lab.	7380 ft	5.60
Leadville, CO	10170 ft	10.79
White Mtn. Res. Sta.	12500 ft	15.07

hours. This example shows that as the number of deployed FPGAs reach cloud-scale, the rate of upsets occurring in the entire system increases linearly with the size of the system.

Location also plays an important role in the rate of upsets. Higher altitudes experience higher upset rates than New York City sea level. For example, systems deployed at White Mountain Research Center in the USA receive 15× the reference amount or a neutron flux of approximately 195 $cm^{-2}h^{-1}$ [1]. Several factors influence neutron flux including geomagnetic cutoff, solar activity, and atmospheric depth or elevation. Elevation is the most important parameter for determining terrestrial neutron flux. Table 3 shows the neutron flux measured at various locations.

An example of SER scaling based on location is found in the SEU data mentioned in [6], which reports one SEU in FPGA configuration logic every 1025 machine days, (i.e., for a single FPGA instance). The Stratix V GS D5 FPGA used in [6] has approximately 79 million CRAM bits based on the critical bits report generated by vendor tools. Using the 63 FIT/Mbit CRAM upset rate from [13] scaled to the number of CRAM bits in the device gives an entire device SEU rate of 5000 FIT at NYC sea level. This translates to 1 upset every 8333 machine days, suggesting that the location of their data center is somewhere with a neutron flux that is approximately 8× greater than NYC, (i.e., 8333 divided by 1025). As seen in Table 3, this level of neutron flux scaling is feasible based on location. Thus, location deployment can have considerable impact on the upset rate of deployed FPGAs.

3.2 Effect of SEUs on FPGA Designs

SEUs in SRAM-based FPGAs corrupt values of memory associated with device configuration and active design state. CRAM bits enable routes, set look-up table (LUT) equations, and adjust the behavior

Table 4: Breakdown of State in a Stratix V GX A7 FPGA

Type	Bits	Percentage
CRAM	91,170,156	60%
BRAM (M20K)	52,428,800	34%
LUTRAM (MLAB)	7,511,040	5%
Flip-Flop	938,880	1%

of circuit components. Some CRAM bit may be associated with device wide behavior. Active design state and user-memory values are stored in non-CRAM bits. These bits include registers, (i.e., flip-flops), distributed memories, (i.e., LUTRAMs), block memories, (i.e., BRAMs), and control registers for specialized IP, like digital signal processing blocks (DSPs) or high-speed transceivers. Values stored in non-CRAM bits can be altered during run time whereas CRAM bits do not typically change once initialized.

Corrupting CRAM bits within an FPGA can cause an operating design to operate differently than expected causing a variety of design failures [28]. Upsets in CRAM can disconnect routes, short routes together, change timing characteristics, invalidate LUT equations, and disturb general circuit operation. They can also indirectly corrupt values in user-memory by causing incorrect values to be stored or by preventing correct values from being saved in memory (e.g., stuck clock enable signal). When SEUs occur in user-memory, bits, counters, state machines, pipelines and other sequential logic elements take on incorrect values, which can be detrimental.

The state elements within an SRAM-based FPGA are dominated by CRAM bits making CRAM bits the primary concern for soft errors. The composition of known internal state subject to radiation-induced upsets in an Intel Stratix V GX A7 FPGA is shown in Table 4. CRAM bits make up 60% of all known internal state in this device. Although block memories make up an additional 34% of the known internal state, they can be protected from radiation induced single-event effects with error correction codes (ECC). Upsets in CRAM bits, however, immediately affect the operation of the underlying circuit. Even if such upsets are repaired at a later time, they introduce operational behaviors that may affect computations performed in the FPGA.

The number of CRAM bits listed in Table 4 exceeds that of LUTRAM and flip-flop bits by a factor of ten, making upsets in the smaller population much less likely. When a LUTRAM is used for read-only logic, its bits are counted as a CRAM bits. Although upsets in non-CRAM bits, (e.g., bits in flip-flops, M20Ks, MLABs), are less likely, they should not be disregarded; these bits play an important role and can cause disruptive behavior if upset.

3.3 Architectural Vulnerability Factor

Not all upsets in CRAM will affect the functionality of a design. Upsets in bits that are unassociated with the resources used by an active design should not disrupt the design behavior, (e.g., a bit in a LUT that is not used by an active design). Upsets in bits that are associated with resources used by a design may alter the underlying circuit, but the effects of the upset on the design may be hidden by error masking. Masking occurs when timing, logic, or design functionality prevent SEUs from causing failure. Different FPGA designs operating on the same device have different soft

Table 5: Benchmark Design Resource Utilization

Design	Total ALMs		Routing
FD3D	190,612	(81.21%)	30.50%
Mandelbrot	173,755	(74.03%)	29.40%
Channelizer	145,180	(61.85%)	23.40%
Matrix Multiply	135,405	(57.69%)	28.40%
FFT1D	129,767	(55.29%)	20.80%
FFT2D	121,015	(51.56%)	22.20%
JPEG Decoder	95,250	(40.58%)	17.70%
Compute Score	94,575	(40.29%)	22.10%
Boardtest	57,547	(24.52%)	11.90%
Video Downscaling	50,914	(21.69%)	10.80%
Vector Op	49,503	(21.09%)	9.90%
Vector Add	49,039	(20.89%)	9.70%
Sobel	48,573	(20.69%)	9.20%
Hello World	46,329	(19.74%)	8.60%

Table 6: Critical Bits

Design	Injectable	Total Critical	
FD3D	98,502,636	69,740,486	(70.8%)
Mandelbrot	98,029,036	66,122,603	(67.5%)
Channelizer	98,534,636	62,673,556	(63.6%)
Matrix Multiply	98,573,036	58,774,590	(59.6%)
FFT1D	98,514,796	57,182,508	(58.0%)
FFT2D	98,439,276	56,797,420	(57.7%)
JPEG Decoder	98,386,796	41,360,019	(42.0%)
Compute Score	98,394,476	44,450,948	(45.2%)
Boardtest	98,578,156	24,009,160	(24.4%)
Video Downscaling	98,587,116	20,934,957	(21.2%)
Vector Op	98,549,996	19,589,044	(19.9%)
Vector Add	98,587,116	19,838,814	(20.1%)
Sobel Filter	98,597,996	19,067,514	(19.3%)
Hello World	98,603,107	18,132,374	(18.4%)

error sensitivities with varying responses to soft errors. Generally speaking, designs that use more resources tend to be more sensitive to soft errors than designs that use fewer resources.

The benchmark designs used in this paper all utilize a different amount of FPGA resources and we expect the SEU sensitivity of each design to vary based on their utilization. The resource utilization of these designs are listed in Table 5. Design are listed in order of highest utilization to lowest utilization based on the number of adaptive logic modules (ALMs) that are used by the design. There are 234,720 ALMs in the Stratix V GX A7 and the device utilization ranges from 20% for the Hello World design to 81% for the FD3D design. Routing influences the overall resource utilization but does not directly contribute to total ALM utilization.

Vendors provide tools to classify bits as associated or unassociated with resources used by the active design. Intel Quartus Prime calls associated bits *critical bits* and allows the users to tag specific hierarchical modules as separate regions, meaning users could exclude portions of the design from tagging if so desired. The number of critical bits estimated by the tools for each of the benchmark designs is summarized in Table 6. Two numbers are given for each design: the injectable bits and the total critical bits. The number of injectable bits represents the number of CRAM bits that can be artificially upset through fault injection. This number varies from design to design since some LUTs are used as user-memories and cannot have faults injected into them. The critical bits percentage is also shown in parentheses. As expected, the percent of the CRAM bits that are critical varies significantly from design to design and corresponds to the resource utilization of the design.

Many critical bits that are used by a design will not cause functional failure if upset. The effect of some CRAM upsets may be masked by the timing, logic, and the current state of the design [24]. A soft error occurring too late in a clock cycle to be latched into memory, or an upset in a register whose value will be over written before it is used are examples of temporal masking. Downstream logic that prevents the errors induced by an upset from propagating is an example of logical masking. An upset in a portion of the design that is unused, like test logic, is an example of functional masking.

Because some upsets in CRAM have no effect on the overall functionality of a design, the raw SEU FIT rate of an FPGA device overestimates the soft error rate of an actual FPGA application. The *architectural vulnerability factor* or AVF [14] is a parameter that is often used to scale the raw soft error rate by an application-specific sensitivity. The AVF is defined as the probability that a CRAM fault will cause a failure in the application design. The overall failure rate is the product of the AVF and the raw SEU soft error rate.

To determine the overall failure rate of specific FPGA accelerator designs, the AVF for the FPGA circuit needs to be estimated. The AVF can be estimated through fault simulation, fault injection, or radiation testing. Fault injection emulates the occurrence of an SEU by altering values in CRAM bits during runtime [18] and is a common approach for estimating overall failure rate. AVF is estimated in fault injection by taking the ratio of the number of faults that cause design failures with the total number of faults injected into the design. Radiation testing accelerates the occurrence of SEUs by exposing the device to greatly increased levels of radiation [17].

3.4 System Failure Modes

SEUs that cause failure can trigger a variety of system failure modes. Broadly classified, failures observed during fault injection experiments of this paper fall into one of three main categories: host unresponsive, FPGA unavailable, and silent data corruption. These are likely the failure modes that would be seen in an FPGA cloud computing platform. All failure modes observed in the experiments of this paper could be resolved by repairing the fault, reprogramming the FPGA, or power cycling the system. No permanent damage was observed during the fault injection experiments of this paper.

3.4.1 Host Unresponsive. Host unresponsive failures occur when an SEU in the FPGA causes system instability in the host. Instability includes lack of network response from the host, host unresponsiveness to keyboard and mouse input, and crashes including abrupt power failure. This instability may stem from issues in the driver or hardware, but the underlying causes are unknown. In the experiments of this paper, host unresponsive failures were primarily observed as a loss of remote connection from the test operator to

the host (see Figure 2). This failure mode is relatively easy to detect and can be resolved by rebooting the system.

3.4.2 FPGA Unavailable. Any SEU that makes it so that the host can not connect to or initialize the FPGA is considered an FPGA unavailable failure. This failure mode manifested itself in many different forms during the fault injection experiments performed during this work. These behaviors include the host not being able to find any FPGAs on the system, not being able to open the target FPGA, PCIe link errors, reading the incorrect kernel or hardware ID, and not being able to initialize the clock, etc. Any behavior that prevents the host-application from initializing the kernel without reprogramming the FPGA is considered an FPGA unavailable failure. Like host unavailable, this failure mode is easy to detect. Recovering from this behavior mainly involves reprogramming the FPGA, but recovering may sometimes require a power cycle.

3.4.3 Silent data corruption (SDC). The most challenging behavior is when an upset causes the FPGA to return incorrect data. In most cases, this data corruption is not detectable by the application and passes as correct data. This event is called silent data corruption or SDC and is the most common failure mode. The severity of SDCs vary from application to application. For some applications, SDCs are not a problem because the returned data is transient and can afford being incorrect, (e.g., web-search results); in other applications, SDC caries severe consequences, (e.g., SQL database commit). SDC can originate from soft errors in several places throughout a design. Errors in the interfaces between the FPGA and host or DRAM could instill SDC during read and write operations. This was observed in the Boardtest application when 4 GB of randomly generated data was transfered to and from DRAM through the FPGA. SDC could also occur from upsets deep inside the logic of a target application. SDC failure modes were caught in the fault injection experiments by comparing the FPGA results against golden result test vectors stored on the host. In practice, SDCs are not detectable by the system.

4 FAULT INJECTION EXPERIMENTS

Fault injection is a common way of emulating soft errors to observe soft error system response [18]. In the experiments of this paper, a random fault injection campaign is used. Faults to inject are selected at random for two purposes. First, this mode of injecting faults better models the behavior of SEUs in deployed environments or at an accelerated radiation test. Second, collecting random samples models population sampling and makes it possible to statistically estimate the overall SEU sensitivity of a design under test. Using random faults to estimate overall sensitivity is known as statistical fault injection (SFI) [20]. SFI provides a good estimate of the overall sensitivity of a target device given sufficient samples. The more samples that are collected, the tighter confidence intervals will be.

Confidence intervals in these experiments are calculated using the Clopper-Pearson binomial proportions confidence interval [9]. Data collected in these fault injection experiments are presented as percentages of sensitivity, or AVF, bounded by a proper confidence interval. The data claims no absolutes but rather a 95% confidence bound within a certain interval, meaning that the true sensitivity lies within the interval with a 95% confidence. To achieve reasonably tight confidence intervals, many thousands of faults were injected into each design. Confidence intervals are such that the overall confidence interval, (i.e., upper bound minus the lower bound), of each experiment falls with 10% of the estimated value.

The goals of the fault injection experiments summarized in this paper are to record system response behavior to configuration upsets, quantify the sensitivity of a design to upset-induced failure, and discuss the benefits and drawbacks of various SEU response approaches. This information is then used to better understand the impact of SEUs on a much broader scale for large-scale FPGA deployments. As such, several benchmarks running on an example node are tested using a fault injection framework. The framework is designed to iteratively inject a fault, execute a test application and record system response. This flow is applied to all benchmark designs until sufficient data is collect. The following sections detail the benchmark designs, and fault injection framework used to collect the data needed to accomplish the goals of this paper.

4.1 Benchmark Designs

To gain a better understanding of soft-error induced failure modes and how frequent such failure modes occurs, it is necessary to sample a diverse set of designs. As such, 15 different applications are included in the experiments of this paper ranging from a simple "Hello World" design to complex signal processing and computation. As expected, AVF failure rates vary among the designs, but common trends are found among the results.

All of the benchmark designs used in this work originated from example designs posted on the Intel FPGA SDK for OpenCL – Developer Zone [12] and from the Terrasic OpenCL board support package for the DE5-net Stratix V GX A7 FPGA developer board [26]. Some designs were excluded from this experiment because they did not fit on the target device. All of the included designs and their respective resource utilizations are listed in Table 5.

All designs listed in Table 5 were compiled by the Intel Quartus Prime 18.0 Standard edition OpenCL compiler using the Terrasic DE5-net BSP. They each use a wide range of resource. The smallest design, Hello World, has the simplest kernel and is dominated by the overhead of OpenCL supporting hardware. OpenCL hardware includes a PCIe controller, a DDR3 controller, and other interface components used by the framework. In addition to the OpenCL hardware, a fault injection and SEU detection IP core were added to each design, with very little overhead, to support fault injection.

4.2 Fault Injection Setup

To collect enough data for this experiment, it was necessary to develop an automated test flow and accompanying equipment setup. The behavior of the host, FPGA, and application under an imposed configuration upset is the object of interest in these experiments. The test setup and accompanying flow are geared towards capturing strange behavior and overcoming adverse effects to restore the environment to a working state for subsequent sampling.

Figure 2 displays the test setup and components used by the experiments in this work. The host is the Dell Precision T7610 described in Section 2. The Stratix V GX A7 FPGA is part of the Terrasic DE5-Net accelerator board. The test operator is an Intel NUC. It orchestrates the entire flow of the experiments. The test

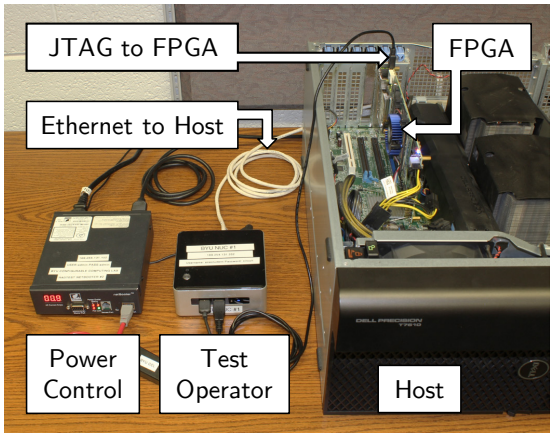

Figure 2: Experiment Setup for Fault Injection Testing

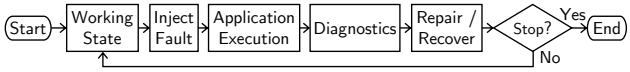

Figure 3: Fault Injection Flow

operator interacts with the FPGA via JTAG to inject faults and configure the device. Interaction with the host by the test operator is conducted via Ethernet. This allows the test operator to execute host applications and observe behavior. When injected faults cause experiment instability, the remote power control allows the test operator to power cycle the host and FPGA concurrently.

4.3 Fault Injection Flow

Several fault injection flows have been developed to capture system response behavior to upsets and estimate sensitivity [18]. Experiments in this paper follow the flow shown in Figure 3. To begin, the system must be brought into a working state. At this point, a fault is introduced and the host application is executed to stress the system while the fault is present. After a period of delay, which allows any errors to propagate, a series of diagnostics are run to observe the health of the system. These diagnostics check the output of the application correctness and check the status of the host and FPGA. If abnormal behavior occurs, the event is recorded and a series of recovery procedures are conducted. If there is no abnormal behavior, the injected fault is repaired. With the system back in a working state, the test is repeated until sufficient data is collected for the target application.

An important part of the described flow is the ability to diagnose incorrect system behavior. Detecting host unresponsiveness and unavailability of the FPGA is straight forward: if the test operator can not communication with the host, the host is unresponsive; if the host can not initialize the kernel, the FPGA is unavailable. Detecting SDC is less strait forward. In these experiments, SDC is detected by comparing the output of the kernel against co-computation by the CPU on the host, or by comparing the output against a golden copy stored on the host.

When a failure is detected, recovery is necessary. After recover actions are taken, (see Section 6), the system is assumed to be in a working state. This is not a perfect assumption. The injected fault can cause non-CRAM bits to become corrupt, (e.g. bits in an M20K). This kind of corruption can persist between runs of the target application. Several runs of the host application may complete successfully before corruption of non-CRAM bits causes a failure. It is assumed that subsequent runs will catch failures caused by non-CRAM bit corruption and that eventually the FPGA will be reprogrammed to remove this corruption. Once the system has been recovered, the test proceeds.

Each injected fault took a minute to test on average. This includes time spent reprogramming the FPGA, injecting faults, waiting for host application and diagnostic tests to complete, and recovering from failure. The data presented in the next section represents approximately 100 days of continuous testing distributed among all of the included benchmark designs. Faults were injected and externally scrubbed using the Quartus fault injection debugger (FID) and associated IP cores. These operations took an average of 4 to 6 seconds to complete. Programming via JTAG takes 12 seconds with an additional overhead of 20 seconds to start the FID software. When reprogramming via JTAG, a system reboot is necessary so that the host can recognize the FPGA, which takes three minutes to complete. The same overhead is needed to perform a complete power cycle.

4.4 Results

The results from the fault injection experiments are shown in Table 7. This table lists all of the tested benchmark designs, the ALM utilization, the number of faults injected for each design, and the percent of injections that caused SDC, FPGA unavailable, and host unresponsive failure. Designs are listed in descending order by the overall AVF of the design for *any* failure. The percentage within parentheses represents the percent of injections that caused failure normalized to the ALM utilization. This metric is included to give a sense of the relative sensitivity of each design to soft errors. Some designs are much larger than others, but by using a normalized AVF, it can be seen that smaller designs, like video downscaling, are more sensitive to soft errors per utilized ALM than larger designs, like Mandelbrot. The ratio between the percentage of critical bits and the AVF for any failure is also included.

Even some of the most highly utilized designs have a small AVF based on the results from Table 7. For example, Mandelbrot utilizes 74% of the available ALMs, yet only 11.6% of randomly upset bits will cause a failure in the system. ALM utilization is closely related to the percentage of critical bits. Comparing the percentage of critical bits to the AVF for any failure, most designs have an AVF that is 6× smaller their percentage of critical bits. The AVF for any failure, of all designs, is at least 5× smaller than their respective ALM utilizations, suggesting that the AVF for any failure of any FPGA design is likely much less than the ALM utilization or the percentage of bits that are critical in the design.

Due to all of the masking effects present in a design, (see Section 3.3), and variations in routing utilization, it is possible for a design with lower utilization to have a higher overall AVF than another design with higher device utilization. For example, FD3D uses 7% more of the device than Mandelbrot yet it is 2.8% less sensitive (i.e., its AVF is 2.8% less than that of Mandelbrot). Another example

Table 7: Fault Injection Results, AVF for specific behaviors (normalized to ALM utilization)

Design	ALM Utilization	Critical Bits	Faults Injected	Silent Data Corruption	FPGA Unavailable	Host Unresponsive	Any Failure AVF	Critical Bits to AVF Ratio
Mandelbrot	74%	67%	9,301	11.3% (15.3%)	0.3% (0.4%)	0.02% (0.03%)	11.6% (15.7%)	5.8×
Matrix Multiply	58%	60%	17,094	9.0% (15.5%)	0.5% (0.9%)	0.03% (0.05%)	9.6% (16.6%)	6.2×
FFT2D	52%	58%	5,223	8.5% (16.4%)	0.5% (0.9%)	0.00% (0.00%)	9.0% (17.4%)	6.4×
FFT1D	55%	58%	7,389	8.2% (14.8%)	0.4% (0.7%)	0.18% (0.32%)	8.8% (15.9%)	6.6×
FD3D	81%	71%	8,094	8.2% (10.1%)	0.5% (0.6%)	0.00% (0.00%)	8.8% (10.8%)	8.0×
JPEG Decoder	41%	42%	7,948	3.4% (8.5%)	3.3% (8.2%)	0.14% (0.34%)	7.0% (17.2%)	6.0×
Compute Score	40%	45%	7,310	5.3% (13.2%)	1.2% (3.0%)	0.00% (0.00%)	6.6% (16.3%)	6.8×
Channelizer	62%	64%	10,709	5.1% (8.2%)	0.4% (0.6%)	0.03% (0.05%)	5.5% (8.9%)	11.6×
Boardtest	25%	24%	12,247	3.4% (13.7%)	0.6% (2.6%)	0.03% (0.13%)	4.1% (16.6%)	5.9×
Video Downscaling	22%	21%	11,641	3.0% (13.7%)	0.6% (2.9%)	0.04% (0.20%)	3.7% (17.1%)	5.7×
Vector Op	21%	20%	6,790	2.6% (12.2%)	0.4% (1.7%)	0.04% (0.21%)	3.0% (14.3%)	6.6×
Vector Add	21%	20%	11,623	2.0% (9.6%)	0.4% (1.8%)	0.04% (0.21%)	2.5% (11.7%)	8.0×
Sobel Filter	21%	19%	6,638	1.3% (6.2%)	0.5% (2.5%)	0.03% (0.15%)	1.9% (9.0%)	10.2×
Hello World	20%	18%	15,442	0.1% (0.4%)	0.2% (0.9%)	0.01% (0.03%)	0.3% (1.5%)	61.3×

of this is Channelizer and JPEG decoder. Generally speaking, the fewer resources a design uses, the smaller its overall AVF will be.

When the AVF for any failure within a design is scaled by the ALM utilization of a design, the normalized AVF among all designs is quite similar. Excluding Hello World, the normalized AVF for any failure among all of the designs resides between 9% and 17% with a mean of 14%. This suggests a strong relationship between device utilization and actual AVF. There is greater relative variation in the normalized AVF for FPGA unavailable and host unresponsive failure modes. This suggests that these failure modes are more closely tied to shared overhead among the designs such as bits associated with device interfaces and device wide status registers.

Silent data corruption is the dominating failure behavior. It accompanied up to 11.3% of all randomly injected faults depending on application resource utilization and other factors. For most of the designs, this failure modes represents the great majority of all failure occurrences. The other two failure modes, FPGA unavailable and host unresponsive, are fairly small in comparison. With the exception of the JPEG Decoder and Compute Score applications, the AVF for the FPGA unavailable failure behavior was fairly consistent across applications. It accompanied about 0.6%, (i.e., one in 168), of all random faults. The host unresponsive behavior did not occur very frequently. It accompanied about 0.04%, (i.e., one in 2,500), of all random faults. This failure mode has a similar AVF among all designs suggesting that it is likely related to SEUs in shared overhead rather than SEUs in the logic of specific applications.

5 FAILURE RATE FOR LARGE-SCALE SYSTEMS

In very large-scale FPGA systems, a design with a small AVF can still fail frequently. For the purpose of studying the impact of soft errors on large-scale FPGA cloud-computing, a 100,000 node system deployed in Denver, Colorado, (3.8× NYC neutron flux) made up of Stratix V GX A7 FPGAs is considered. An AVF of 1% in this situation equates to 23.6 million FIT or a MTTF of one-and-a-half days.

Unlike the other failure modes, SDC is not detectable and will occur with up to 11.3% of all random upsets in the data set collected, (i.e., Mandelbrot). An AVF of 11.3% in the considered 100,000 node system equates to 266 million FIT or an MTTF of 3.75 hours. Table 8 lists the FIT and corresponding MTTF for each design as if deployed on the considered 100,000 node system.

Table 8: SDC MTTF on a 100,000 node system in Denver, CO

Design	FIT	MTTF
Mandelbrot	266,000,000	3.8 Hours
Matrix Multiply	212,000,000	4.8 Hours
FFT2D	200,000,000	5.0 Hours
FFT1D	193,000,000	5.2 Hours
FD3D	193,000,000	5.2 Hours
JPEG Decoder	80,000,000	12.5 Hours
Compute Score	125,000,000	8.0 Hours
Channelizer	120,000,000	8.3 Hours
Boardtest	80,000,000	12.5 Hours
Video Downscaling	71,000,000	14.2 Hours
Vector Op	61,000,000	16.3 Hours
Vector Add	47,000,000	21.2 Hours
Sobel Filter	31,000,000	1.3 Days
Hello World	2,000,000	2.5 Weeks

Other failure modes would occur less frequently and would be easier to detect and address. In the considered system, FPGA unavailable events would range from 5 M to 78 M FIT with a 14 M FIT average among designs, (i.e., 3 day MTTF); host unresponsive events would have a FIT up to 4 M averaging 1 M, (i.e., 1.5 month MTTF). Because these failure modes occur much less frequently and are more easily detected and addressed, they pose a much smaller threat to system integrity and stability than SDC.

5.1 Reliable Computing and Mission Time

Although the metric "Mean-Time to Failure" (MTTF) provides a useful measure for understanding the overall rate at which failures in the system will occur, it does not adequately represent the fact that many failures in the system will occur in sooner than the

Table 9: Mission Time for Different Reliabilty Constraints

r	0.5	0.9	0.99	0.999	0.9999	0.99999
$MT(r)$ (sec)	9,446	1,436	137	13.6	1.36	.14

MTTF estimate. In fact, 63% of the failures expected in the system will occur within a time that is less than the MTTF (with some of these failures occurring in a much shorter time than the MTTF). The MTTF metric can give a false sense of security in suggesting that the system will operate without failure for the amount of time specified by the MTTF estimate.

A better measure for evaluating the rate of SDC failure for a computing system is to estimate the time in which the system can operate above a pre-specified level of reliability, r. This measure is called the "mission time" (MT) and is a function of a minimum reliability constraint, r. For example, the mission time of a system that must operate with a reliability of r=0.99 (i.e., MT(0.99)) indicates the amount of time that the system can operate with a probability of success of 99% or higher. The mission time can be computed from the continuous time reliability function, R(t). The reliability function for failure due to soft errors can be modeled by an exponential function used for modeling constant failure rate systems [23]:

$$\mathrm{R}(t) = e^{-\lambda t}, \tag{1}$$

where λ is the constant failure rate (failures/time). The mission time can be determined by assigning $R(T)$ the reliability constraint, r, and solving for t,

$$\mathrm{MT}(r) = \frac{-ln(r)}{\lambda}. \tag{2}$$

The constant failure rate can be determined by taking the reciprocal of the mean-time to SEU upset (MTTU) estimate and scaling it by the AVF for SDC events, or $\lambda = \mathrm{AVF_{SDC}}/\mathrm{MTTU}$.

For the 100,000 node system in Denver running the Mandelbrot benchmark, the MTTU = 5790/3.76 = 1540 sec and λ = .113/1540 = 7.3×10^{-5} failures/second. With a reliability constraint of r = .99, the mission time for this system is only 137 seconds (about 6 minutes). This result suggests that the system can only operate for about two minutes with a 99% confidence that no SDC events have corrupted the data. This time is far less than the estimated mean-time to SDC failure of 13,628 seconds. The mission time for other reliability constraints on this platform is summarized in Table 9.

Examining mission time in relation to reliable computing shows that a high confidence in the computational correctness comes in short bursts of operation. Even designs with a low SDC AVF have short-lived mission-times for high-reliability. For example, on a 100,000 node Stratix V GX A7 FPGA system at NYC flux, a design with a 1% AVF for SDCs only maintains a reliability higher than five-nines (i.e., 0.99999) for six seconds. Such short periods of time at high reliability may not be long enough for application tasks to complete and still meet reliability requirements.

6 SEU DETECTION AND RECOVERY

Because SRAM-based FPGAs are sensitive to single-event effects, FPGA vendors provided methods for detecting SEUs within the CRAM and and repairing the CRAM state. The technique for detecting and repairing CRAM state is called Configuration Scrubbing.

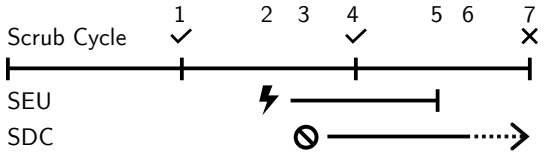

Figure 4: Scrub cycles, SEU occurrence, and SDC

Configuration scrubbing is the continuous process of reading the configuration memory, determining if there are errors, and correcting these errors if found. Configuration scrubbing is asynchronous to the operation of the active design and is performed silently in the background. The data in the configuration memory can be repaired by the use of error correction coding (ECC) that is included in the configuration memory frames. Single-error correction, double-error detection (SECDED) codes are used to allow all single-bit errors within a frame to be corrected and all double-errors to be detected. In addition to SECDEC encoding, a checksum is computed for the entire configuration memory to detect complex errors that are not detected or corrected with SECDED encoding [2].

Although configuration scrubbing is able to detect and correct SEU-induced CRAM errors, this process is not instantaneous and takes a considerable amount of time. For the Stratix V device used in paper, the time it takes to perform a whole device CRAM scrub cycle varies from 47 milliseconds to 24.20 seconds depending on the internal clock frequency and the clock divisor. No matter how long the scrub cycle takes, SEUs in CRAM will be present for millions of clock cycles before they are detected and possibly corrected. For example, consider a device with a 100 ms scrub period and a global clock operating on a 100 MHz clock. On average, an upset in CRAM will be present for 50 ms or five million clock cycles in this device before it is detected and corrected by scrubbing. This temporary CRAM upset may cause undesirable behavior in the design that may or may not go away with the scrubbing.

Figure 4 diagrams scrubbing along side an SEU event and occurrence of an SDC. At position 1 in the diagram, a scrub cycle completes. No SEUs are detected or corrected during the scrub cycle as denoted by a check mark. At position 2, an SEU occurs in the CRAM. This upset remains in the CRAM until it is detected and corrected by the scrubbing engine. At position 3, an SDC appears on the outputs of the design. This occurs after the upset has been present in the device for several clock cycles. Errors introduced by the upset take time to propagate through the design. At position 4, a subsequent scrub cycle completes. This scrub cycle reports that no SEUs were detected or corrected even though an SEU occurred since the last completion of a scrub cycle. If an SEU occurs in a CRAM frame after the frame is evaluated as part of the current scrub cycle, then the SEU may remain present and undetected until the next scrub cycle. At position 5, the scrubbing engine detects the SEU and corrects it. At position 6, the SDC induced by the upset either flushes out of the design or remains present as denoted by the dotted continuation of the SDC. At position 7, the final scrub cycle shown completes and reports that an SEU was detected and corrected as denoted by an X.

Enabling internal CRAM scrubbing does not eliminate erroneous behavior caused by SEUs in CRAM. Scrubbing only limits the periods of time during which SEU induced errors can propagate. If an upset corrupts state that persists for long periods of time, (e.g., counters, state-machines, status registers, infrequently written block memories), then this corruption will remain even after the upset is removed. In some cases, the corrupted data will naturally flush out of the system during normal use or through a design reset. Other situations will require the device to be reprogrammed or power cycled to remove the corruption [16].

In the fault injection experiments of this paper, every time a failure is detected, a series of successively more drastic recovery actions are conducted in succession to bring the system back into a working state. This procedure makes it possible to determine the least invasive recovery option necessary for each occurrence of a failure. The first action taken is scrubbing. Scrubbing is performed by repairing the injected fault or removing it from CRAM. Success of the recovery action is determined by a subsequent execution of the host application and diagnostic results. If no issue is encountered, (i.e., the application completes without any failure), the system is considered to be a working state and fault injection continues. If failure behavior remains after scrubbing, a reprogramming of the FPGA is attempted to resolve the issue. After this recovery action, the host application is run again. If the application finishes cleanly, fault injection continues with the next random fault; otherwise, the host is gracefully shutdown and a power cycle is performed on the host and FPGA concurrently.

Table 10 lists each benchmark design, the number of SDC events observed, and the distribution of successful recovery actions taken. SDC occurred in these benchmarks because of an upset present in the CRAM of the device, (i.e. injected fault). Repairing the upset does not undo the SDC that has already occurred, but it can restore the system to a working state so that subsequent computations are be processed correctly. Table 10 shows that configuration scrubbing can restore the proper functionality of a design a high percentage of the time when an SEU occurs. For many of the designs, configuration scrubbing is able to restore functionality 98% of the time. That being said, some situations still required the FPGA to be reprogrammed or the host computer and FPGA to be power cycled before proper functionality is restored. This is likely do the corruption of user memory, (e.g., non-CRAM bits, block memory, flip-flops), or other persistent state [16]. This data shows that scrubbing is effective in restoring design functionality and that, although the rate is small, some situations still require reprogramming the FPGA or power cycling the node.

Table 10: SDC System Restoration

Design	Events	Scrub	Reprogram	Power Cycle
Mandelbrot	1,050	99.5%	0.5%	0.0%
Matrix Multiply	1,530	99.2%	0.8%	0.0%
FFT2D	442	99.1%	0.9%	0.0%
FFT1D	606	99.2%	0.8%	0.0%
FD3D	667	98.7%	1.3%	0.0%
JPEG Decoder	273	98.2%	1.1%	0.7%
Compute Score	390	97.2%	2.8%	0.0%
Channelizer	541	98.0%	2.0%	0.0%
Boardtest	412	97.1%	2.9%	0.0%
Video Downscaling	345	94.2%	5.8%,	0.0%
Vector Op	174	95.4%	4.6%	0.0%
Vector Add	233	98.3%	1.3%	0.4%
Sobel Filter	85	90.6%	8.2%	1.2%
Hello World	12	66.7%	8.3%	25.0%

6.1 Response Options

Knowing that SEUs can cause disruptive behavior, an appropriate SEU detection and recovery mechanism should be implement to address this concern. Depending on the application, some failure behavior may be tolerable and users may only want to respond to the most severe failure behavior caused by SEUs. There are a number of different recovery approaches a user could take:

(1) Disable scrubbing, respond when a failure is detected,
(2) Enable scrubbing, respond when a failure is detected,
(3) Respond every time an SEU is detected, or
(4) Respond only to upsets in the most critical regions.

A user could take a minimal approach by disabling scrubbing and only responding to detectable failure modes: FPGA unavailable and host unresponsive. In the system referenced in Section 5, these events are infrequent. FPGA unavailable would occur once every 3 days on average and host unreachable would occur once every 1.5 month on average. Using this approach, users should at least consider periodically checking for SEUs and recording their occurrence for use in failure analysis when unexpected behavior occurs. This mode misses out on the protection from continual SDCs that scrubbing provides. Going without this protection may not be a great concern, especially if the FPGA is frequently reprogrammed.

Enabling scrubbing and only responding to detectable failures is a good option for preventing continual SDC. In most cases, scrubbing is a very effective means for restoring the system to a working state, but scrubbing alone will not completely prevent SDCs from ever happening. User concerned about SDCs should consider responding to SEUs as they are detected. Responding only to upsets in critical regions reduces how frequently action must be taken but adds complexity recovery implementation. In the system referenced in Section 5, CRAM SEU events have a 25 minute MTTF, and SEU in critical CRAM bit events have a 38 minute MTTF. Depending on the system and operating design, responding only to upsets in critical bits may or may not be worth the additional effort required. In the referenced system, SDC events have a 3.8 hour MTTF. This shows that only 1 in 9 SEU responses would actually prevent an SDC if the user responded to every SEU and only 1 in 6 responses would be necessary if the user responded only to upsets in critical bits within the reference system. The other responses are effectively overhead for the selected recovery approach.

Response to an SEU can range from a simple reset to discarding previously generated data and power cycling the system. In order to respond to an SEU, the user must be given access to the appropriate SEU detection signals. With these signals, the user can then decide how to respond when an SEU occurs. To tolerate a large number of SEU-induced failures at a software level, at a minimum, hardware must be able to detect and report any SEUs in a timely enough manner so that the software level can isolate resulting

errors and take appropriate recovery actions [4]. Assuming computation between nodes is independent, overhead of response would be minimal, as only effected nodes would need to be recovered. Additional software or design level fault-tolerance techniques could also be used to detect and correct errors. As more large-scale FPGA accelerator platforms come online, the number of nodes and risk of SEU-induced SDC necessitate the implementation of an appropriate SEU response mechanism.

7 CONCLUSION

Industry is making use of hundreds of thousands of FPGAs to accelerate cloud computing applications. The FPGAs being used are susceptible to radiation-induced soft errors. Individual nodes have a relatively low soft error rate, (e.g., 1 upset every 18 years on average for a Stratix V GX A7 FPGA in NYC neutron flux); but when numerous nodes are deployed, the soft error rate increases dramatically. This paper looks at the impact of SEUs on large-scale FPGA cloud-computing systems. A hypothetical, but realistic hundred-thousand node Stratix V GS A7 FPGA system deployed in Denver is considered where upsets occur on average every half-hour.

Not all SEUs adversely affect the functionality of an FPGA cloud computing application, but some compromise system integrity and stability. Fault injection testing of 15 designs estimates their overall AVF to be between 0.3% and 11.6% depending on resource utilization and error masking. Estimated AVF was found to be at least 5× smaller than the percentage of utilized ALMs or percentage of bits tagged as critical. Most observed failures were SDC events. Within a 100,000 node system deployed in Denver, the design with the highest AVF, (i.e., Mandelbrot), would have a 3.8 hour MTTF for SDC, a 3 day MTTF for FPGA unavailable, and a 1.5 month MTTF for host unresponsive. FPGA unavailable and host unresponsive behaviors are easy to detect; responding to these events carries negligible overhead.

SDC is difficult to detect and can have broad impact on the system. Applications that require high-reliability must have short mission-times or implement appropriate SEU response techniques. In the considered 100,000 node environment mission-time for a reliability specification of 0.99 is less than two minutes. Scrubbing can be used to address SDC by responding to SEUs as they occur. One less the AVF is the percentage of SEU responses that are unnecessary. System wide response to each SEU can significantly lower availability and will not necessarily address all SDC events. One in 10 upsets occurs in non-CRAM bits based on the ratios between the populations and approximately one in every hundred SEU would occur in non-CRAM design state of the Mandelbrot design. To avoid SDC from upsets in non-CRAM bits, additional mitigation-techniques need to be applied to the target design.

ACKNOWLEDGMENTS

This work was supported by the Utah Space Grant Consortium and by the I/UCRC Program of the National Science Foundation under Grant No. 1738550.

REFERENCES

[1] 2006. Measurement and reporting of alpha particle and terrestrial cosmic ray-induced soft errors in semiconductor devices. Retrieved December 12, 2018 from https://www.jedec.org/sites/default/files/docs/JESD89A.pdf

[2] P. Adell et al. 2008. Assessing and mitigating radiation effects in Xilinx SRAM FPGAs. In *2008 European Conference on Radiation and Its Effects on Components and Systems*. 418–424.

[3] J. Arram et al. 2015. RAMETHY: Reconfigurable acceleration of bisulfite sequence alignment. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, New York, NY, USA, 250–259.

[4] L. Barroso et al. 2018. The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Third edition. *Synthesis Lectures on Computer Architecture* 13, 3 (2018), 1–189. https://doi.org/10.2200/S00874ED3V01Y201809CAC046

[5] R. Baumann. 2001. Soft errors in advanced semiconductor devices – Part I: The three radiation sources. *IEEE Transactions on Device and Materials Reliability* 1, 1 (2001), 17–22.

[6] A. Caulfield et al. 2016. A cloud-scale acceleration architecture. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 1–13. https://doi.org/10.1109/MICRO.2016.7783710

[7] M. Ceschia et al. 2003. Identification and classification of single-event upsets in the configuration memory of SRAM-based FPGAs. *IEEE Trans. Nucl. Sci.* 50, 6 (2003), 2088–2094.

[8] E. Chung et al. 2018. Serving DNNs in real time at datacenter scale with Project Brainwave. *IEEE Micro* 38, 2 (2018), 8–20.

[9] C. Clopper and E. Pearson. 1934. The use of confidence or fiducial limits illustrated in the case of the binomial. *Biometrika* 26, 4 (1934), 404–413.

[10] Deloitte. 2017. Hitting the accelerator: the next generation of machine-learning chips. Retrieved December 12, 2018 from https://www2.deloitte.com/content/dam/Deloitte/global/Images/infographics/technologymediatelecommunications/gx-deloitte-tmt-2018-nextgen-machine-learning-report.pdf

[11] B. Frank. 2017. Microsoft unveils Brainwave, a system for running super-fast AI. Retrieved December 12, 2018 from https://venturebeat.com/2017/08/22/microsoft-unveils-brainwave-a-system-for-running-super-fast-ai/

[12] Intel. 2018. Intel FPGA SDK for OpenCL – Developer Zone. Retrieved December 12, 2018 from https://www.intel.com/content/www/us/en/programmable/products/design-software/embedded-software-developers/opencl/developer-zone.html

[13] A. Keller et al. 2018. Dynamic SEU Sensitivity of Designs on Two 28-nm SRAM-Based FPGA Architectures. *IEEE Trans. Nucl. Sci.* 65, 1 (2018), 280–287.

[14] S. Mukherjee et al. 2003. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.* 29–40.

[15] E. Nurvitadhi et al. 2017. Can FPGAs beat GPUs in accelerating next-generation deep neural networks?. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, New York, NY, USA, 5–14.

[16] B. Pratt et al. 2006. Improving FPGA Design Robustness with Partial TMR. In *2006 IEEE International Reliability Physics Symposium Proceedings*. IEEE, 226–232.

[17] H. Quinn. 2014. Challenges in Testing Complex Systems. *IEEE Trans. Nucl. Sci.* 61, 2 (2014), 766–786. https://doi.org/10.1109/TNS.2014.2302432

[18] H. Quinn et al. 2013. Fault Simulation and Emulation Tools to Augment Radiation-Hardness Assurance Testing. *IEEE Trans. Nucl. Sci.* 60, 3 (2013), 2119–2142.

[19] H. Quinn and P. Graham. 2005. Terrestrial-based radiation upsets: a cautionary tale. In *13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*. 193–202.

[20] P. Ramachandran et al. 2008. Statistical Fault Injection. In *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*. IEEE, 122–127.

[21] E. Schadt et al. 2010. Computational solutions to large-scale data management and analysis. *Nature Reviews Genetics* 11, 9 (2010), 647–657.

[22] B. Schroeder. 2011. DRAM errors in the wild: A large-scale field study. *Commun. ACM* 54, 2 (2011), 100–107.

[23] D. Siewiorek and R. Swarz. 1998. *Reliable computer systems* (third ed.). A. K. Peters, Natick, MA.

[24] A. Silburt et al. 2008. Specification and Verification of Soft Error Performance in Reliable Internet Core Routers. *IEEE Trans. Nucl. Sci.* 55, 4 (2008), 2389–2398.

[25] I. Stamoulias et al. 2017. Hardware accelerators for financial applications in HDL and High Level Synthesis. In *2017 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*. 278–285.

[26] Terasic. 2018. Stratix V - DE5-Net FPGA Development Kit. Retrieved December 12, 2018 from https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=526

[27] D. Thomas et al. 2009. A comparison of CPUs, GPUs, FPGAs, and massively parallel processor arrays for random number generation. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. ACM, New York, NY, USA, 63–72.

[28] M. Wirthlin. 2015. High-reliability FPGA-Based systems: Space, high-energy physics, and beyond. *Proc. IEEE* 103, 3 (2015), 379–389.

[29] Xilinx Inc. 2018. *Device Reliability Report*. Xilinx Inc. Retrieved December 12, 2018 from https://www.xilinx.com/support/documentation/user_guides/ug116.pdf