Acceleration of FPGA Fault Injection Through Multi-Bit Testing.

Conference Paper · January 2010

Source: DBLP

3 authors:

Grzegorz Gilbert Cieslewski
Sandia National Laboratories
19 PUBLICATIONS **199** CITATIONS
SEE PROFILE

Alan D. George
University of Florida
250 PUBLICATIONS **2,732** CITATIONS
SEE PROFILE

Adam Jacobs
ARM
19 PUBLICATIONS **242** CITATIONS
SEE PROFILE

Acceleration of FPGA Fault Injection through Multi-Bit Testing

Grzegorz G. Cieslewski, Alan D. George, and Adam M. Jacobs
NSF Center for High-Performance Reconfigurable Computing (CHREC)
ECE Department, University of Florida, Gainesville, FL, USA

***Abstract*—SRAM-based FPGA devices are an attractive option for data processing on space-based platforms, due to high computational capabilities and a lower power envelope than traditional processing devices. These devices present unique fault-testing challenges as single-event effects can trigger changes in functionality by changing the configuration memory of the device. With each new generation, FPGA configuration memories increase in size and designs increase in complexity, making it very difficult, if not impossible, to perform exhaustive fault-injection testing to verify design reliability. We propose a novel methodology for fault injection in FPGAs using multi-bit testing that can significantly accelerate the process. Traditionally, each bit in configuration memory is tested separately; by testing multiple bits during one test, speedups of more than 10× can be achieved.**

I. Introduction

A high degree of on-board data processing is a necessity for next-generation space satellites. This need is especially true for platforms which employ new sensor technologies, capable of collecting more data than a spacecraft's ability to transmit that data to its destination. This downlink bottleneck, caused by transmitter power and bandwidth limitation, target visibility, and high latency can be mitigated by performing a higher degree of on-board data processing. As traditional CPU-based technologies struggle to provide high-performance computing in space, Field-Programmable Gate Arrays (FPGAs) provide an attractive and powerful replacement. The inherent advantages in performance, energy efficiency, size, and adaptability facilitated by reconfigurable logic can help answer demands of next-generation, space-based platforms.

Radiation-hardened FPGAs are common components in space-based platforms. However, due to the nature of the radiation-hardening process they are typically smaller, slower and more expensive than their commercial-off-the-shelf (COTS) counterparts. By contrast, commercial FPGA devices provide unprecedented levels of efficiency for space missions, but are highly susceptible to single-event effects (SEEs) caused by high-energy particles. To maintain high system reliability, traditional and innovative fault-tolerant (FT) design methods are required. An effective testing method is needed to evaluate these devices and FT design methods to expedite the space qualification process. The approach should efficiently introduce faults, test behavior, and estimate expected error rates without the need for expensive radiation testing at each step of the development cycle.

Numerous fault-injection techniques have been proposed in the past, ranging from simulation approaches to radiation testing, but no single method has provided the optimal solution to the problem. The predominant method of emulating the effects of single-event upsets (SEUs) on FPGA devices is by programming a modified bitstream into the configuration memory of the device, which accounts for the majority [1] of all susceptible bits. Bitstream modification alters the functionality of the device and allows one to observe the potential effects of an SEU. The total amount of configuration memory required to define the behavior of the reconfigurable logic in FPGAs is constantly increasing as the size of the devices increases. Additionally, to effectively gauge the effects of SEUs on a design, a comprehensive set of test vectors is required. For large designs, such sets are difficult to calculate and can be prohibitively large to use in testing. These trends make it difficult, if not impossible, to perform comprehensive fault-injection testing of the whole device. An alternative to testing the entire configuration memory of an FPGA is to use statistical sampling methods to select a subset of bits to investigate and use confidence intervals to show bounds on the susceptibility estimate of a given design. An accurate prediction of susceptibility with a tight confidence interval requires a large number of samples and can involve long testing times.

In this paper we propose and demonstrate a novel approach to FPGA fault injection through multi-bit testing which allows acceleration of the testing process while maintaining the correctness of the results. Our Simple, Portable, Fault Injector platform for FPGAs (SPFI-FPGA) [5] supports fault injection for Xilinx Virtex-4 FPGAs through partial reconfiguration (PR). This injection mode minimizes the time required to modify configuration memory and further improves injection speed. In this paper, we present a traditional single-bit and augmented multi-bit injection methodology and discuss the performance of the proposed fault-injection approach in the context of relevant case studies.

The remaining sections of this paper are organized as follows. Section 2 surveys previous work related to this topic. Section 3 provides an overview of SPFI-FPGA as well as single-bit fault-injection and testing methodologies. Section 4 details the new fault-injection approach and additional methods for improving the performance of the proposed scheme. In Section 5, we present performance results of fault-injection testing. Finally, Section 6 provides conclusions and outlines directions for future work.

II. Background

Modern SRAM-based FPGAs are constructed from collections of configurable logic blocks (CLBs) and custom cores (multipliers, processors, BlockRAMs) which are connected by a programmable network allowing for highly intricate designs. CLBs consist of relatively small components including look-up tables (LUT), multiplexers, flip-flops (FF), and supporting structures composed of AND-OR gates making them capable of implementing complex logic functions. The on-chip programmable interconnect consists of grid of switchboxes bonded to wire segments of various lengths. Each switchbox integrates a large number of programmable switches allowing for custom routing of signals between CLBs. Information used to set the function of switchboxes, CLBs and other components is stored in the configuration memory.

While all space-based electronics are susceptible to faults caused by radiation, SRAM-based FPGAs have a unique set of additional concerns due to their reconfigurable nature. SEUs, which cause faults in FFs or BlockRAMs, are closely related to the upsets on non-reconfigurable platforms and can lead to data corruption or single-event functional interrupts (SEFIs), where a device or design can enter an unexpected state. However, upsets which occur in a device's configuration memory are in part unique to reconfigurable FPGAs, as the configuration memory controls the function of the logic and interconnect. This type of fault can lead to broken nets, formation of new connections, or other effects resulting in unpredictable behavior of the circuit.

The effects of SEUs on FPGAs can be studied by emulating how radiation affects the underlying silicon structures through modifying bits in the configuration memory of the device. It was shown in [1] that the vast majority of errors attributed to SEUs are results of changes in the configuration memory and not to the embedded FFs, as the cross-section of all the FFs is only a small fraction of the cross-section of all bits in the configuration memory. In case of a Virtex-4 SX55, the size of configuration memory (not counting BRAM) is 15.4 million bits and the total number of FFs attached to slices is approximately 49 thousand bits, making the configuration memory upsets over 300 times more likely [6].

Multiple approaches to FPGA fault injection have been studied in recent years. Most of them targeted performance of fault injection above other tradeoffs, which can lead to limited portability and reusability.

Johnson et al. [1], [2] proposed a specialized testbed, SLAAC-1V, for fault-injection experiments. It consists of two identical FPGAs (Virtex XCV1000) in a parallel configuration. Outputs are connected to a voter which constantly compares outputs of both FPGAs. The testing procedure calls for one of the FPGAs to be programmed with a corrupt bitstream, while the other remains in its original state. The designs are cycled with inputs, and outputs compared by the voter to determine if the given change in configuration memory has undesirable effects. Due to the custom parallel architecture, the system has a very high performance and is capable of rapidly testing faults without the need for a golden standard data set.

The Xilinx Research Test Consortium (XRTC) system [9] uses a base motherboard from SEAKR Engineering with a daughter-card containing an FPGA as device under test (DUT). The motherboard contains two FPGAs which are responsible for function monitoring of the DUT by providing test vectors and verifying outputs against a golden standard. Fault injection is performed through the JTAG interface using an external computer or, in later versions, is integrated with the motherboard over a SelectMap port [6]. The XRTC system offers excellent performance, but requires the use of known data sets, or a two-step testing approach where first the run establishes correct outputs and the second run determines the effects of the fault. Unfortunately, like the SLAAC-1V testbed, since the design uses specific hardware and custom boards, it cannot be used with any other FPGA devices without redesign.

Sterpone et al. [3] have proposed a different method that uses a System-on-Chip (SoC) approach. The FPGA design is divided (both physically and logically) into the unit under test (UUT) and the supporting logic consisting of embedded PowerPC, a timing unit, and an Internal Configuration Access Port (ICAP) controller. The logic that the user wishes to test is placed in the UUT and constrained to a portion of an FPGA. The support logic is responsible for fault injection, providing test vectors and collecting results. This method of fault testing performs even better than the previous two approaches as it uses the high-speed ICAP controller for partial reconfiguration and stores the test vectors directly on the FPGA. Although this approach is uniquely suitable for testing design components, its split design restricts adaptability and does not allow for testing standalone systems. Moreover, the size of the test vectors is restricted to memory available, and the UUT is constrained to resources not used by supporting logic.

The Virtex-II SEU Emulator (V2SE) briefly described in [4] uses yet another configuration approach. Similar to the XRTC testbed, it uses the SelectMap configuration port for high-speed injections in combination with COTS and custom-designed hardware. Whereas the SelectMap port allows for rapid reconfiguration, it is not as popular as the JTAG port and is not present on all platforms.

III. SPFI-FPGA Tool

As briefly introduced in [5], SPFI-FPGA is a flexible fault-injection tool devised for use with Virtex-4 FPGAs to test behavior of designs when subjected to faults in configuration memory. It is a part of a larger SPFI framework targeted at system-level testing encompassing not only FPGAs but also CPUs (PowerPC) and reconfigurable many-core-based platforms (TILE64). The primary motivation for SPFI is to provide maximum portability in order to support a wide range of systems and enable in-system testing. While performance of the tool is important, custom approaches limiting the applicability of the tool to a particular device or platform are intentionally avoided whenever possible.

A. *SPFI-FPGA System Architecture*

Figure 1 shows the high-level architecture of the SPFI-FPGA tool. The architecture is divided into three major components: Campaign Generator (CG), Management Engine (ME), and Test Generator (TG).

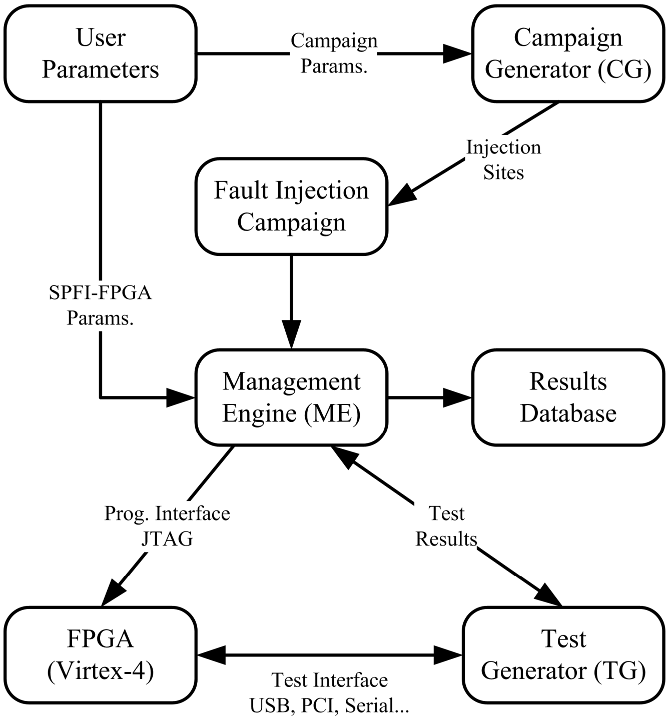

Figure 1. Block diagram of SPFI-FPGA system architecture

The CG is used to generate the campaign file, based on user-specified parameters such as injection area, resource type, and number of bits to be tested. The campaign file contains frame address and bit offsets describing the selected location of faults to be injected. To obtain those locations, the CG uses a debug bitstream file, generated by the Xilinx BitGen tool [8], which contains addresses of each data frame in the bitstream and is stripped of padding frames required in a course of full reconfiguration. This mechanism of bitstream analysis uncovers details about the geometry of the device required for fault injection and removes the need for a device database describing each supported device.

The ME is the main workhorse of the system, and is responsible for campaign management, FPGA monitoring, fault insertion and removal, and data logging. Primary inputs for the management engine are the bitstream files for the given design and a campaign file that contains a description of the experiment. FPGA monitoring and programming is facilitated by use of the JTAG port, which is used as the primary programming channel due to its high availability and accessibility on most of the platforms. Fault insertion and removal is achieved by automatically generating partial-bitstream files, which contain data frames to appropriately modify the configuration memory of the FPGA [7]. All injection results and events are logged in a text database which can be used for detailed statistical analysis.

The TG verifies the operation of the design for the FPGA and provides that information back to the ME. To maximize flexibility, this component is a user-defined, plug-in application that communicates with the FPGA using the Test Interface (TI). The complexity of this component can vary sharply due to design choices and system architecture. The simplest version would trigger a built-in self-test (BIST) function which could verify the operation of the system without any interaction with the TG. After the completion of the BIST, the TG would be notified of the results. On the other end of spectrum, the TG can generate a random set of test vectors to be transferred over a TI to the DUT. The result would consist of another set of vectors which would be verified against the correct outputs in the TG. To increase the performance of the TG, it can be partially hosted on the FPGA in a form of a wrapper that communicates with the design.

B. Fault-Injection Methodology

The fault-injection methodology developed in conjunction with SPFI-FPGA considers portability and performance as well as correctness and repeatability. The main portability consideration is the type of configuration interface. As cited previously, the JTAG port was chosen as the primary programming channel due to its availability on most of the platforms. The high level of abstraction in the SPFI-FPGA's architecture allows adding specialized programming interfaces, such as SelectMap, in a modular fashion.

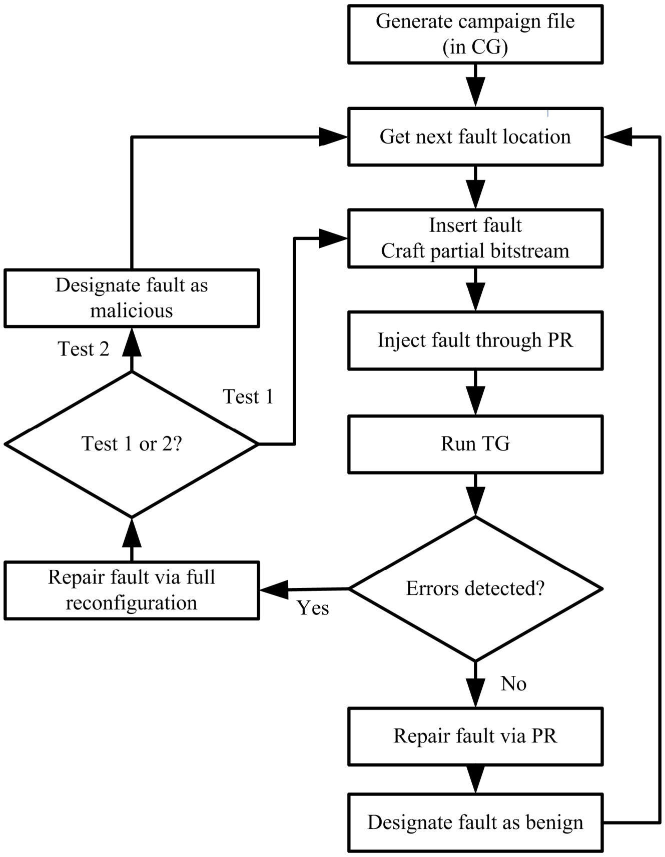
Generate campaign file (in CG)
Get next fault location
Insert fault Craft partial bitstream
Inject fault through PR
Run TG
Errors detected?
Yes
No
Repair fault via full reconfiguration
Test 1 or 2?
Test 1
Test 2
Designate fault as malicious
Repair fault via PR
Designate fault as benign

Figure 2. Single-bit fault-injection methodology

To increase the speed of fault injection, SPFI-FPGA uses a mixture of partial- and full-reconfiguration techniques as shown in Figure 2. After the fault location is read from the campaign file, the ME selects a frame with the corresponding address and inverts the specified bit. The frame is then used to create a crafted partial bitstream which is programmed onto the device. The TG is used to verify the correct functionality of the FPGA, and based on the outcome an appropriate recovery procedure is taken. In cases where an observable error does not occur, the configuration memory is repaired by programming the original frame on to the device. When the injection does cause an observable error, the full reconfiguration is performed to reset the FPGA to a nominal state. The site is retested to remove any bias introduced by false positives that can occur due to multiple partial reconfigurations as well as inconsistent behavior of other components of the system (board components, TI).

The mean time per injection is strongly dependent upon the performance of the TG as well as the speed of the programming interface. It can be modeled as

$$t_{sbfi} = t_{pr} + t_t + p_e\left(2t_{fr} + t_{pr} + t_t\right) + \left(1 - p_e\right)t_{pr} + t_c \quad (1)$$

where t_{sbfi} represents the average elapsed time for execution of one injection. t_{fr} and t_{pr} represent the time needed to perform full and partial reconfiguration. t_t denotes execution time of the TG program, p_e is the probability that the injected fault will be manifested as an observable error, and t_c is the constant software overhead per injection.

C. Testing Methodology

When considering fault injection, one must account for the general architecture of the design being tested. Such an FPGA design can be classified as a module that requires data to be provided for it, or as standalone system that interfaces with external resources. We propose classifying these systems in one of two categories, module-level testing and system-level testing.

Module-level testing as shown in Figure 3 is mostly suitable for smaller designs which occupy only a part of the chip, so that the remaining part of the chip can be used to provide fault-injection facilities. In such case, a significant part of the TG can be shifted from the attached PC onto the FPGA. The test vectors required for testing the module could be placed in the spare BlockRAMs, mitigating the delay of TI between the TG and the FPGA.

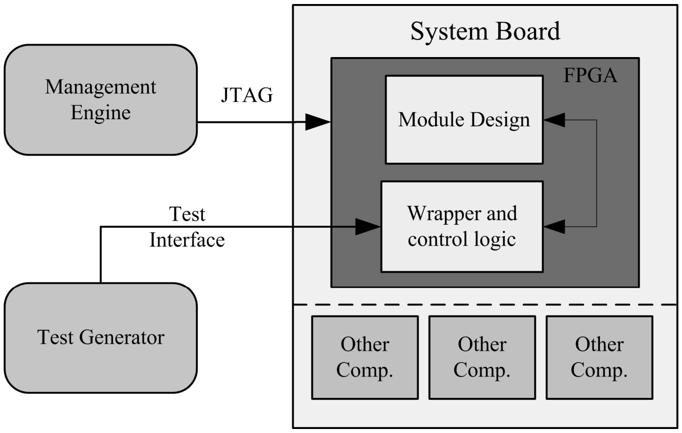

Figure 3. Module-level testing

As shown in Figure 4 system-level testing is best suited for designs that occupy the majority of the chip and are integrated with other components (SDRAM, network, ADCs). In such case, the TG program might not be directly attached to the FPGA but communicate through some other system. It might be required to provide and receive test vectors or start the system's BIST.

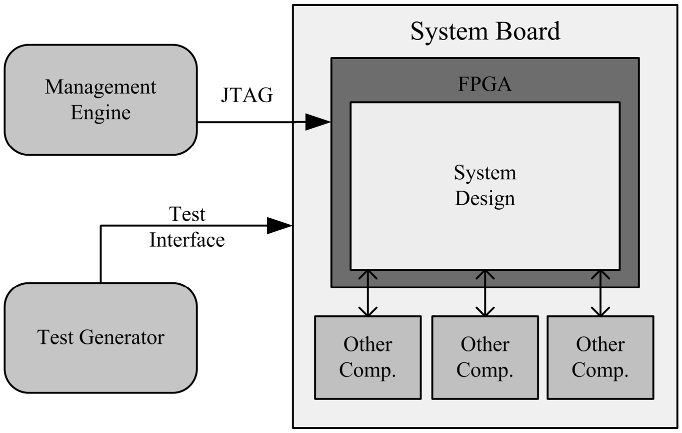

Figure 4. System-level testing

D. Fault-Injection Bottlenecks

The driving methodology in the SPFI design is to architect a tool that is portable across a variety of FPGA types and systems. As a result, there are many tradeoffs which lower the fault-injection performance. The primary bottleneck is the JTAG port. Due to its serial nature it can deliver only moderate performance, which is constrained by the JTAG clock speed supported by the system and software/hardware characteristics of the programming cable.

Another limiting factor is performance of the TG. Generating a representative set of test vectors for a complex module can be a difficult and daunting task. In addition, such a comprehensive set may be very large and require a long period of time to test. In some cases, testing time can be many orders of magnitude greater than the injection time [3]. This time can be mitigated by storing the test data on the FPGA when the module-level testing approach is taken.

The error rate also influences performance of SPFI-FPGA, since different recovery procedures are taken depending on the outcome of TG run. In the case of an observable error, the FPGA has to be fully reconfigured in addition to the retesting procedure. Full reconfiguration is very expensive, as it takes significantly longer time to execute. This problem is not as prominent when testing the susceptibility of fault-tolerant designs, as the error rate is usually very small, causing few full reconfigurations.

IV. Multi-Bit Testing

One of the major goals of any fault-injection system is a high-injection rate to allow for an accurate susceptibility characterization of a given design and device. Unfortunately, there are many tradeoffs affecting the injection rate achievable by the tools. The usual solutions involve modifying a part of the system, particularly the programming interface, in order to improve the overall performance. This approach usually involves major changes to the system components and in many cases yields specialized hardware which is tied to the particular platform. Consequently, this method decreases portability and applicability to a narrow set of platforms. An alternate way of viewing the problem is to decrease the total number of injections in order to achieve better performance while maintaining the quality and fidelity of results. Such an alternative is infeasible when testing one bit at a time but becomes practical when this condition is relaxed. To achieve this goal, we propose a new tactic for fault-injection testing, which will decrease the total number of injections by testing multiple bits or batches at a time.

The general premise behind multi-bit testing is to inject and test multiple faults at a time to decrease the total number of injections and consequently decrease the total injection time. We assume that the probability of two or more random faults masking each other's effects and yielding a correctly working circuit is incredibly small and will not affect the results of fault-injection testing. To further strengthen this assumption, we impose constraints on the location of random faults which are to be jointly tested. None of the jointly tested faults can occupy an identical CLB. This approach is related to Multiple-Bit Upset (MBU) testing [10] with the exception of location of the upsets. MBUs caused by cosmic rays are closely clustered, whereas our approach disperses faults through the testing space to minimize possibility of masking.

The proposed fault-testing methodology incorporates a combination of both single-bit and multi-bit fault-injection methods. This new approach yields identical information to the single-bit approach while significantly reducing the time required for fault testing.

A. *Multi-Bit Testing Injection Methodology*

In order to test the proposed injection methodology, we have modified the SPFI system. Similar to the original approach, the CG is used to design a campaign containing fault locations to be injected. Special care is taken to make sure that bits in a particular batch are not part of the same CLB. This step is accomplished by comparing the addresses of frames containing faults (excluding minor address bits) and making sure that none are identical within the selected set.

As presented in Figure 5, after the fault locations are known, the ME selects a batch of faults for the first test. The bits specified by the campaign file are corrupted in corresponding frames and combined into a crafted partial bitstream that is programmed onto the device. If the TG determines that the resulting configuration has identical functionality to the original, all of the tested faults in the set are deemed to be benign. When errors are reported, each fault location is tested separately by using the single-bit fault-injection approach to determine which bits caused the errors.

B. *Campaign Sequence Optimization*

Through the course of our investigation of fault-injection techniques, we have observed much higher incidence of errors caused by '1' to '0' transition faults than '0' to '1' transition faults. The majority of the configuration bits in the FPGA is responsible for routing of the signals and could account for this observation. In the case of Xilinx FPGAs, configuration of an empty switchbox consists of only '0' bits. If a signal is running through a switchbox, then some configuration bits are set to '1'. This situation translates to the '0' to '1' transition creating possible short circuits, while the '1' to '0' transitions create possible open circuits. On average, a bitstream consists of more '0' bits than '1' bits because very few designs can take full advantage of the interconnect fabric.

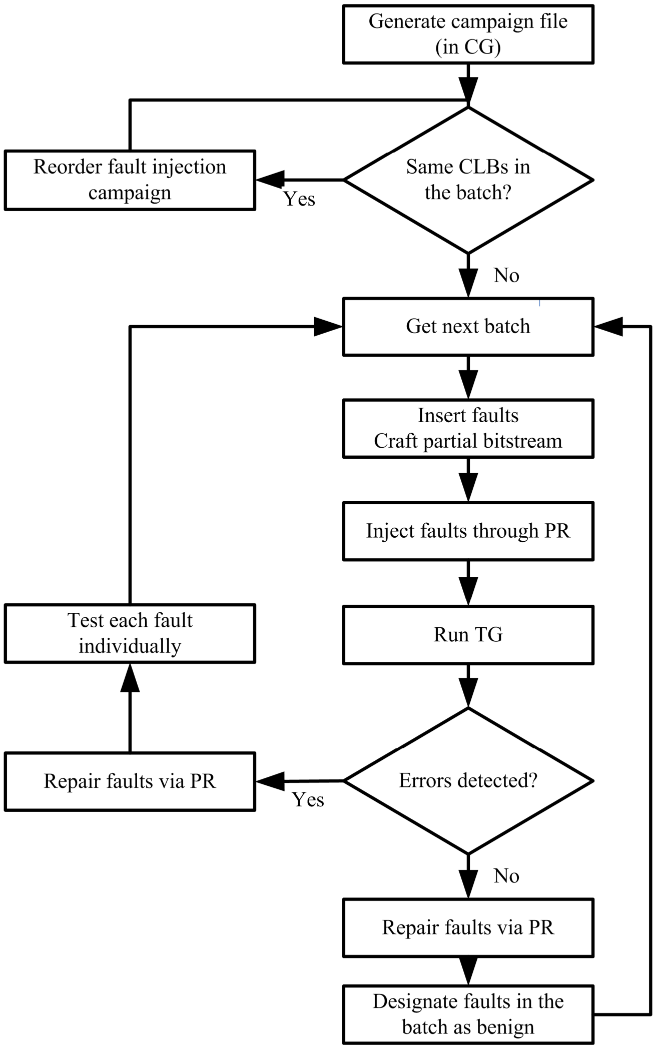

Figure 5. Multi-bit fault-injection methodology

Such fault incidence suggests a possible optimization of the campaign structure which would take an advantage of what order the faults are tested. By grouping the '1' to '0' transitions at the beginning of the campaign, and '0' to '1' transitions at the end, it is possible to skew the distribution of faults from uniform throughout the campaign to the short, high-concentration area, where the '1' to '0' transitions are located the large, low-concentration area. Campaign reordering will allow for selecting larger batch sizes and result in higher speedups. The campaign-reordering process takes into account the location of the faults to prevent having multiple faults in the same CLB. In cases where such an arrangement is not possible, members of the other set are used.

C. Batch Size

The optimal batch size is difficult to calculate and requires a detailed timing model of the fault-injection system and prior knowledge of the design's FT characteristics. For fault-mitigated designs, the situation is even more complicated, as a detailed knowledge of design partitioning and susceptibilities of each subcomponent is needed.

An additional issue with selecting an optimal batch size is the occurrence of false-positive batch tests. These cases occur when a batch test fails but no constituent individual bit causes a fault by itself. By decreasing the batch size, we can minimize the occurrence of MBU-like effects which are responsible for the false positives. This problem is especially prominent in replicated designs where a combination of faults is required in order for an observable error to appear.

Additional complications are also introduced when using reordered campaigns where susceptible bits are not distributed uniformly throughout the campaign. It is foreseeable that use of different batch sizes for each part of the campaign could yield results nearing optimal, but prior knowledge of susceptibilities would be required for both types of transitions.

V. Results

To showcase this new injection technique, we have modified our SPFI-FPGA tool to support multi-bit injection as well as campaign sorting. The experimental testbed consists of a Linux-based computer connected to an ML-401 development board with a Xilinx Virtex-4 LX25 FPGA. The JTAG programming interface consists of a Xilinx Platform Cable USB II, and the TI uses a FTDI 232R USB-to-serial converter cable to interface with FPGA logic. The programming of the FPGA takes approximately 3.6 seconds for a complete bitstream and 110-180 milliseconds for partial bitstreams depending on the number of faults being injected.

The kernel developed for the following experiments performs matrix multiplication (MM) on two 9×9 matrices of 16-bit integer or fixed-point values. Matrix multiplication is a common kernel in signal and image processing applications. Although FPGA area constraints limit the size of a single matrix multiplication, larger matrix sizes can be processed by dividing them into blocks. The MM algorithm is parallelized over n processing units, which allows for the calculation of the dot product in a single clock cycle therefore reducing computational time from $O(n^3)$ to $O(n^2)$ clock cycles. The design makes use of embedded BlockRAM and DSP resources available on the Virtex-4.

In order to assess the performance of the augmented version of SPFI-FPGA, the execution time required to test 10,000 faults is compared, both with and without campaign reordering. The batch size is varied in the range of 2 to 40 bits, in order to determine area of the best performance. The TG uses a pre-computed set of 50 randomly generated vectors to verify correct operation of the core. Based on testing, approximately 1.6% of bits are susceptible to SEUs for the core without fault-tolerance features.

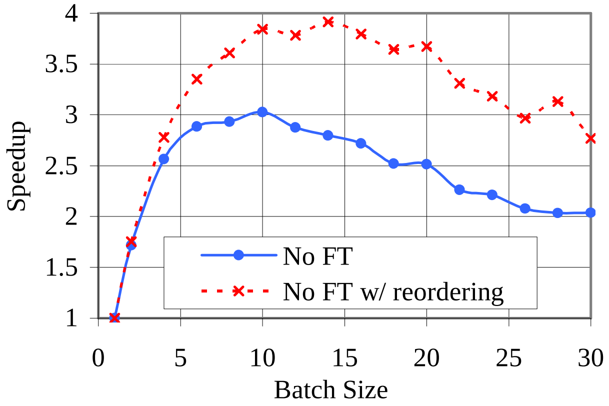

Figure 6. Fault-injection speedup for MM without FT features

Figure 6 illustrates the execution speedup attained by SPFI-FPGA versus the batch size while testing the MM core without FT features. The optimal batch size for the unsorted campaign is 10 and achieves speedup of 3.02× versus the single-bit injection method. For the sorted campaign, the optimal batch size is 14 with a speedup of 3.91×. Runtime of the experiments is summarized in Table I.

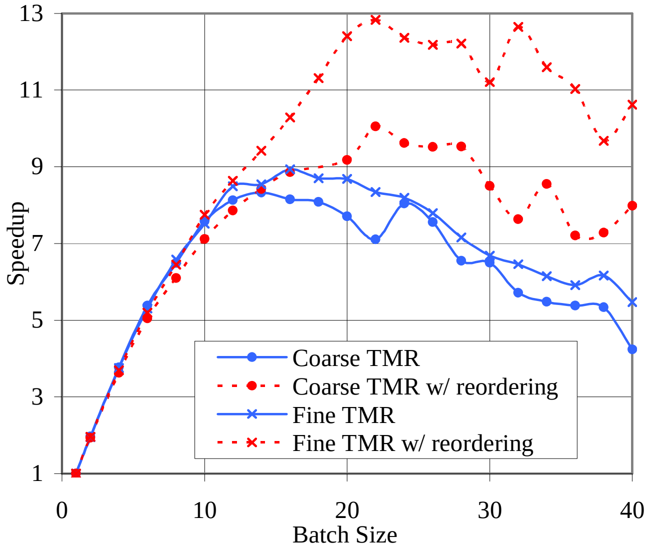

Figure 7. Fault-injection speedup for MM using coarse and fine grain TMR

Designs that exploit FT features show a more dramatic performance improvement. Figure 7 shows speedup of SPFI-FPGA for the MM core with two different TMR approaches. The design labeled Coarse TMR [11] consists of manually instantiated, triplicated MM cores. The voter, located in the wrapper part of the design, selects correct output in a bitwise fashion. The design labeled Fine TMR uses the EDIF replication tool developed at BYU [12], which replicates low-level components of the core and inserts internal voters into the design. Fault injection on Coarse TMR achieves speedups of

8.33× and 10.03×, for batch sizes of 14 and 22, with regular and sorted campaigns, respectively. Injection on Fine TMR achieves speedups of 8.93× and 12.83×, with batch sizes of 16 and 22, for regular and sorted campaigns, respectively. The testing performance of Fine TMR is better than Coarse TMR due to lesser occurrence of false-positive batch tests that cause significant performance degradation. Similarly, reordered campaigns show improved performance with higher batch sizes, because of significant decrease in the incidence of false-positive batch tests in the '0' to '1' phase of the campaign.

TABLE I. SUMMARY OF THE RESULTS

Experiment Name	Single-Bit Injection Runtime	Optimal Batch Size	Multi-Bit Injection Runtime	Max. Speedup
No FT	4.20 h	10	1.39 h	3
No FT w/ reord.	4.15 h	14	1.06 h	4
Coarse TMR	3.79 h	14	0.45 h	8
Coarse TMR w/ reord.	3.79 h	22	0.38 h	10
Fine TMR	3.80 h	16	0.43 h	9
Fine TMR w/ reord.	3.80 h	22	0.30 h	13

VI. CONCLUSIONS AND FUTURE WORK

Use of FPGAs on space-based platforms is highly effective for increasing computational power of the system per unit energy, but appropriate measures must be taken to determine reliability of the design. By using our novel multi-bit injection methodology in conjunction with reordering of fault-injection campaigns, high speedups are possible when testing a design. On our testbed, we were able to achieve speedups up to 4× for unmitigated designs and up to 13× for designs employing a form of TMR. Such high testing speedups can dramatically decrease the time required to test complex designs in preparation for space deployment.

Future work in this direction may explore further optimizations to the proposed methodology. In particular, adaptively controlling the batch size during run-time would increase performance and usability of the fault injector. Another campaign optimization strategy would involve separating bits belonging to the switchboxes from bits belonging to other components in the FPGA fabric. Campaigns optimized with this information in mind would yield even better results than the statistical approach based on transitions proposed in the paper. Unfortunately, such an approach would require in-depth knowledge of bitstream structure, which is vendor-proprietary and thus not publicly available.

ACKNOWLEDGMENTS

This work was supported in part by the I/UCRC Program of the National Science Foundation under Grant No. EEC0642422. The authors gratefully acknowledge vendor equipment and/or tools provided by Xilinx.

REFERENCES

[1] E. Johnson, M. J. Wirthlin, and M. Caffrey, "Single-event upset simulation on an FPGA," *Proc. Int. Conf. Engineering of Reconfigurable Systems and Algorithms (ERSA)*, Las Vegas, NV, 24-27 Jun. 2002.

[2] E. Johnson, M. Caffrey; P. Graham, N. Rollins, M. Wirthlin, "Accelerator validation of an FPGA SEU simulator", *IEEE Trans. on Nuclear Science*, vol.50, no.6, pp. 2147-2157, Dec. 2003.

[3] L. Sterpone, M. Violante, "A New Partial Reconfiguration-Based Fault-Injection System to Evaluate SEU Effects in SRAM-Based FPGAs," *IEEE Trans. on Nuclear Science*, vol.54, no.4, pp.965-970, Aug. 2007.

[4] M. French. P. Graham, M. Wirthlin, L. Wang, and G. Larchev, "Radiation Mitigation and Power Optimization Design Tools for Reconfigurable Hardware in Orbit", *Proc. Earth-Sun System Technology Conference*, Hyattsville, MD, 28-30 Jun. 2005.

[5] G. Cieslewski, A. D. George, "SPFFI – Simple Portable FPGA Fault Injector," *Military and Aerospace Programmable Logic Devices (MAPLD) Conference*, Greenbelt, MD, Aug. 31-Sep. 3. 2009.

[6] G. Allen, G. Swift, and C. Carmichael, "Virtex-4VQ static SEU characterization summary," Xilinx Radiation Test Consortium, Tech. Rep. 1, 2008.

[7] "Virtex-4 Configuration Guide," Tech. Doc. UG071 (v1.5), Xilinx Corporation, San Jose, CA, pp. 1–116, 2007.

[8] "Command Line Tools User Guide," Tech. Doc. UG628 (v 11.4), Xilinx Corporation, San Jose, CA, pp. 1-380, 2009.

[9] D. Petrick, W. Powell, J. Howard Jr., K. LaBel, "Virtex-II Pro PowerPC SEE Characterization Test Methods and Results", *Military and Aerospace Programmable Logic Devices (MAPLD) Conference*, Washington D.C., 7-9 Sep. 2005.

[10] H. Quinn, P. Graham, J. Krone, M. Caffrey, and S. Rezgui, "Radiation induced multi-bit upsets in SRAM-based FPGAs," *IEEE Trans. on Nuclear Science*, vol. 52, no. 6, pp. 2455–2461, Dec. 2005.

[11] C. Carmichael, "Triple module redundancy design techniques for Virtex FPGAs," Tech. Doc. XAPP197, Xilinx Corporation, Nov. 2001.

[12] B. Pratt, M. Caffrey, J. Carroll, P. Graham, K. Morgan, and M. Wirthlin, "Fine-grain SEU mitigation for FPGAs using partial TMR," *IEEE Trans. on Nuclear Science*, vol. 55, pp. 2274–2280, Aug. 2008.