TURTLE: A Low-Cost Fault Injection Platform for SRAM-based FPGAs

Corbin Thurlow, Hayden Rowberry, and Michael Wirthlin
NSF Center for Space, High-performance, and Resilient Computing (SHREC)
Brigham Young University
Provo, Utah, USA
{corbin.thurlow, wirthlin}@byu.edu, hayden.rowberry@byu.net

***Abstract*—This paper presents the TURTLE fault injection platform for inserting faults into SRAM FPGAs. The TURTLE system is designed to gather significant fault injection data to test and validate radiation-induced single-event upset (SEU) mitigation techniques for FPGAs. The TURTLE is a low-cost fault injection platform that emulates upsets within the configuration memory (CRAM) of an FPGA through partial reconfiguration. This work successfully implemented the proposed architecture and performed several successful fault injection campaigns on multiple designs and SEU mitigation techniques. Results in this paper show large amounts of data collected from a fault injection campaign used to validate the PCMF SEU mitigation technique. Over 170 million injections were performed using the TURTLE for this campaign.**

***Index Terms*— Field Programmable Gate Array; Fault injection; SEU Mitigation**

I. Introduction

Like most semiconductor devices, FPGAs are sensitive to radiation-induced single-event upsets (SEU). In SRAM-based FPGAs where the configuration of the FPGA is determined by the state of the configuration memory or CRAM, SEUs within the CRAM can alter the operation of the circuit. Such alterations of the circuit are problematic in safety-critical or high-reliable systems such as those typically used in space systems. It is important to understand how FPGAs and FPGA designs behave in the presence of CRAM upsets before using them in such systems.

A common approach for understanding the behavior of FPGAs in radiation environments is to perform accelerated radiation testing. In a radiation test, an FPGA configured with a specific design is placed in front of a high-energy radiation beam and carefully monitored for anomalous behavior. Such testing, including neutron, proton, and heavy-ion beamlines, are available at a variety of radiation test facilities. These beam facilities provide radiation at a sufficient energy level to cause upsets within the FPGA and are used to understand device behavior that might occur in space.

Radiation testing, however, is expensive and difficult to access. Another approach for evaluating the behavior of FPGAs with CRAM upsets is through fault injection. With fault injection, upsets can be inserted into the CRAM artificially through partial reconfiguration. Although the mechanism for inserting faults into the CRAM is different from radiation testing, fault injection can provide essential information on the sensitivity of FPGA designs to CRAM upsets and provide early estimates on the effectiveness of SEU mitigation techniques [1]. In many cases, fault injection is often used to prepare for radiation testing.

This paper presents the TURTLE (Testing Ultra-Reliability Techniques using Low-cost Equipment) fault injection system that provides a low-cost approach for artificially injecting faults within an SRAM FPGA. The goal of the TURTLE system was to develop a fault injection system that can be used to evaluate SEU mitigation approaches at a relatively low-cost. This system is based on the Xilinx Artix-7 FPGA, but the concept can be applied to any FPGA family. This paper will describe the TURTLE fault injection system and provide an example of the results obtained using this system for an SEU mitigation strategy.

II. Background and Related Work

FPGA fault injection is not a new idea, and a variety of different FPGA fault injection approaches have been demonstrated [2]. These approaches all vary, but in general, all of them include the following: inject faults into the FPGA design, operate the FPGA design under the fault condition with test vectors, identify design or system failures, and report results. The approach used for each of these varies from system to system, but all of these approaches seek to understand how faults affect the behavior of FPGA designs subject to radiation-induced faults.

One approach created a fault injection platform to evaluate the effectiveness of triple-modular redundancy (TMR) [3]. This fault injection platform is composed of two separate FPGA boards: one board contains the design under test (DUT), while the other board has a golden copy of the design. The results of both boards are compared, and discrepancies in their operation indicate a system failure. The fault injection results obtained on this system were validated and correlated through radiation testing.

Other work created the FLIPPER fault injection platform [4], which operates with two FPGA devices. One FPGA is the control device, which injects faults into CRAM and sends input vectors to the second DUT FPGA. A simulation environment generates input test vectors and produces a golden

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

978-1-7281-1957-1/19/$31.00 ©2019 IEEE

output vector used for comparing system output. This platform can inject a single CRAM fault quickly to provide rapid fault injection data (50μs per fault).

The fault injection platform presented in this paper is similar to those discussed above. The TURTLE system uses one FPGA for a golden reference design and a second for a DUT. Fault injection is performed through partial reconfiguration via JTAG interface. This platform provides a low-cost solution to simplify the implementation of fault injection campaigns on FPGAs designs. This goal facilitates the generation of very large amounts of fault injection data with relatively little cost.

III. Statistical Confidence of Fault Injection

One of the most important purposes of fault injection is to estimate FPGA design sensitivity to single-event upsets. The sensitivity of an FPGA design is the percentage of CRAM bits within the FPGA that will cause the design to operate incorrectly if upset. This sensitivity measure can be used to estimate failure rates and the mean-time to failure (MTTF) of a design in a variety of radiation environments.

The sensitivity of an FPGA design can be estimated with fault injection by injecting a predetermined number of CRAM faults (n) into the design and observing the number of system failures (k). After injecting a fault, the fault injection system must observe the FPGA design and determine whether the given fault caused any design failures. If the injected CRAM fault caused the design to deviate from its expected behavior, the CRAM bit is labeled as *sensitive*. If the fault did not cause any problems, it is labeled as *insensitive*. In most cases, it is not possible to test every CRAM bit, and an estimate of the sensitivity must be made. This sensitivity estimator, $\hat{r}$, of the Binomial distribution can be computed by dividing k by n as follows:

$$\hat{r} = \frac{k}{n} \tag{1}$$

A large number of faults is needed to obtain an accurate estimate of the sensitivity. The standard deviation of the sensitivity estimator is calculated with the following equation:

$$\sigma = \sqrt{\frac{k}{n^2}\left(1 - \frac{k}{n}\right)} \tag{2}$$

The standard deviation of the sensitivity estimator reduces with $1/n^2$, suggesting that a large number of faults must be injected to obtain statistical confidence in the estimate. A related metric for measuring the variation of the sensitivity estimate is the "coefficient of variation", which shows the extent of variability in relation to the mean population. This measure is obtained by dividing the standard deviation by the mean as follows:

$$\text{Coefficient of Variation} = \frac{\sigma}{\hat{r}} \tag{3}$$

As SEU mitigation techniques are developed that successfully reduce the sensitivity of FPGA designs to CRAM upsets, it is more difficult to obtain statistically accurate estimates of the design sensitivity. To obtain statistically accurate estimates, FPGA designs with lower sensitivity (i.e., those with SEU mitigation) must have significantly more faults injected than FPGA designs with higher sensitivity (i.e., those without SEU mitigation).

TABLE I: Fault Injection Data for MD5 Encryption Core

Description	Unmitigated	TMR
Faults Injected *(n)*	2,000,000	14,000,000
Failures *(k)*	129,677	486
Est. Sensitivity *($\hat{r}$)*	6.48%	.0035%
Coefficient of Variation	2.69E-3	4.54E-2

The need for large numbers of fault injection can be demonstrated with the SEU mitigation technique of Triple Modular Redundancy (TMR). TMR is a well-known SEU mitigation technique that involves triplicating circuit resources and voting on the results [5]. Errors in one circuit copy are masked by the correct results in the remaining two good copies. TMR has been applied to FPGA circuits and has been shown to significantly reduce design sensitivity. As an example of this technique applied to an MD5 encryption core, Table I demonstrates the effectiveness of TMR with the results of a fault injection campaign completed on this platform. The unmitigated design has a sensitivity of 6.48%, and with TMR, the sensitivity is reduced to .0035% (over 1000x reduction in design sensitivity).

Although the TMR design has a much lower estimated sensitivity, the relative variation of the estimate is much higher than that of the unmitigated design. Even though the TMR fault injection campaign involved 7x more faults than that of the unmitigated design, its relative variation is over 20x larger than that of the unmitigated design. More injections and observed failures are needed to reduce this coefficient and obtain tighter interval bounds in the estimated design sensitivity.

The example from Table I demonstrates the need for a fault injection platform capable of providing a large number of faults. The TURTLE platform was designed to address this need by providing a low-cost platform that is able to inject a large number of faults for FPGA designs. The primary goal was to design a fault injection platform that could collect statistically sufficient data with as little cost as possible.

IV. TURTLE Architecture

The actual TURTLE architecture was driven by the goal of building a low-cost fault injection platform that is relatively easy to use. The TURTLE platform must inject faults at a sufficient rate to collect statistically significant amounts of data for a wide variety of FPGA designs and SEU mitigation techniques. This section will provide a high-level overview of the TURTLE fault injection approach implemented to meet these goals.

The TURTLE approach follows a typical golden/DUT approach, as shown in Figure 1. In this approach, two identical circuits operate in parallel, and CRAM faults are injected into the DUT. Comparator logic checks the behavior of both circuits each clock cycle, and when the behavior of the golden

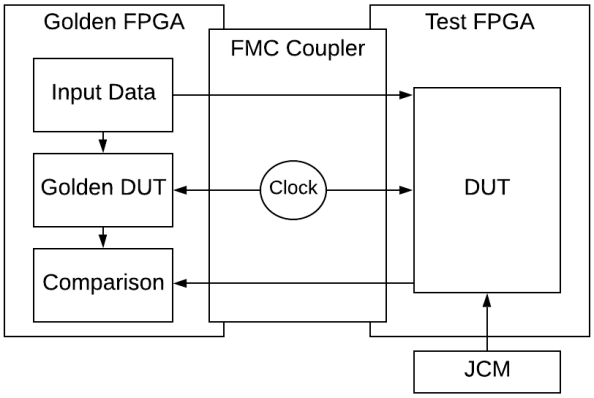

Fig. 1: The TURTLE architecture includes two FPGAs, one golden copy of the design, and the DUT, which is tested through fault injection

copy and the DUT deviate, a system error is detected and logged. An advantage of this approach is that it is relatively easy to implement – the circuitry needed for both the golden design and DUT is the same. A disadvantage of this approach is that more FPGA logic is required: one FPGA to implement the golden design and one FPGA for the DUT. Also, the need for comparator logic introduces timing delays into the system.

The fault injection system is controlled and monitored over JTAG using an external device called the JTAG Configuration Manager (JCM) [6]. This device injects the CRAM faults into the DUT, monitors the status of the designs over JTAG, and controls the overall fault injection procedure. The following subsections describe each of these components in more detail.

A. Digilent Nexys Video Board

Within the TURTLE architecture, the golden and DUT circuits are implemented in separate FPGA boards, requiring two FPGA boards for a single TURTLE system. The TURTLE uses the Digilent Nexys Video board [7], a low-cost FPGA prototyping board used primarily for video related circuits (see Figure 2). The Nexys Video board contains a single Xilinx Artix-7 series FPGA (Xilinx part number XC7A200T-1SBG484C), which contains 215,360 logic cells (32,650 slices). The FPGA is connected to a variety of I/O interfaces such as Ethernet, USB, HDMI video and is used primarily for prototyping FPGA video application circuits.

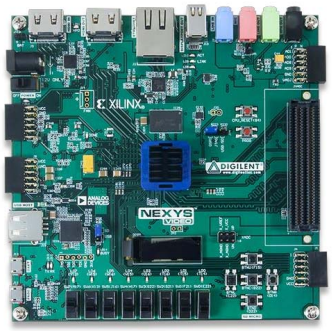

Fig. 2: Two Digilent Nexys Video Boards are paired in the TURTLE architecture [7]

An important component of the Nexys Video board used by the TURTLE system is the FPGA Mezzanine Card (FMC) connector. This board implements the 160-pin FMC low pin count (LPC) standard connector and is used in the TURTLE system for transmitting control signals and test data between the golden design and DUT. The two FPGA boards are coupled through the FMC connector by a custom FMC coupler card described in the next section.

The FPGAs on the Nexys Video board can be configured in a variety of ways including USB-JTAG, an onboard EEPROM, a microSD card, and JTAG using a 6-pin JTAG through-hole port. The external JCM system connects to the Nexys board using the 6-pin JTAG header for programming, monitoring, and testing purposes. The Nexys board also contains a variety of LEDs, push buttons, and switches that are used for the configuration, control, and monitoring of a TURTLE system.

B. FMC Coupler Card

A custom-built printed circuit board (PCB) was designed to connect two Nexys Video boards using the FMC ports (see Figure 3). This board, called the FMC coupler card, provides a mechanical connection between the FMC ports of the two Nexys Video boards of a single TURTLE system. This card also provides onboard clocks distributed to both FPGA boards so that both boards can operate synchronously during fault injection.

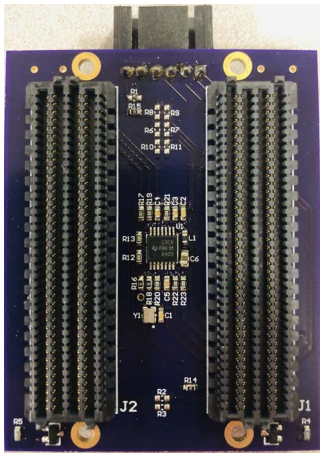

Fig. 3: The FMC coupler card couples the I/O of two Nexys Video boards

The FMC coupler card uses 32 differential I/O pairs for a total of 64 data signals between the two FPGA boards. While the LPC connector provides 160 physical pins, the LPC specification calls for only 68 of those to be data lines. The other signals are dedicated to JTAG, I2C, power, ground, and other purposes. The data signals between the boards are swapped, allowing for the same FPGA design to operate on

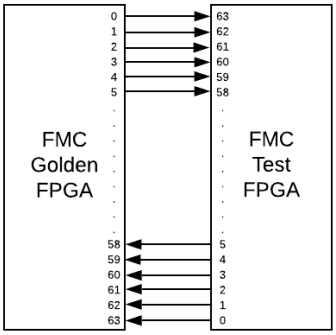

Fig. 4: FMC signal swapping through the FMC connectors on the golden and DUT FPGA boards

either board without additional I/O customization. Figure 4 shows the swapping architecture of these signals.

In addition to the 64 data signals, the FMC coupler card provides three reset signals, two control signals, and up to three synchronous clock signals[1] for both FPGAs. The layout and routing of these signals were carefully controlled to make sure the timing of the clocking and resets are identical for both FPGAs. The coupler card also links the JTAG chain of both boards, as described in the next section.

C. JTAG

The TURTLE fault injection approach relies primarily on JTAG to inject faults, monitor system status, and control the fault injection flow. While there are other methods that provide faster configuration and fault injection speeds, many off-the-shelf boards do not support all these configuration modes [8]. JTAG was selected because of its wide use in development boards such as the Nexys Video board. The JTAG standard provides the ability to control multiple devices on a single chain.

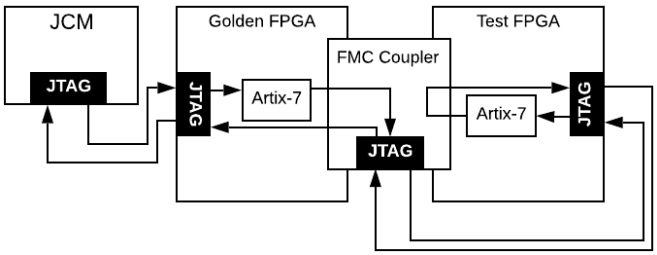

Fig. 5: The JTAG chain in the TURTLE architecture for test control

The FPGA designs used in fault injection incorporate the BSCAN primitive to provide status on the designs to the JTAG chain. The FMC coupler card unifies the JTAG chains of both boards to provide a single JTAG chain for the JCM system. The organization of the JTAG chain is shown in Figure 5. The JTAG chain, beginning with the JCM, first connects to the golden device. The chain then continues to the FMC coupler, where a custom cable is used to jump the JTAG chain to the test device, where the JTAG signals then loop back to complete the JTAG chain.

[1]Three clock signals are provided to support triple modular redundancy designs.

D. JCM

The JTAG Configuration Manager, or JCM, controls the fault injection process in the TURTLE system. The JCM is a Linux-based embedded system that operates on the Zynq 7010 programmable SOC and was designed to generate high-speed custom JTAG command sequences [6]. The programmable logic (PL) portion of the Zynq device contains a custom hardware circuit for controlling JTAG TAP controllers. The JCM has custom software for communicating with the JTAG control firmware. Combining a dedicated circuit for controlling the JTAG chain along with custom JTAG software allows the system to provide both high-speed JTAG communication as well as flexibility in creating complex JTAG command sequences.

With control of the JTAG chain shown in Figure 5, the JCM is able to control all aspects of the fault injection process. The JCM can communicate with the configuration circuit of either FPGA on the chain to perform initial FPGA configuration, inject faults into the FPGA CRAM, repair FPGA CRAM faults, and read the configuration status. The JCM is also able to send or receive data from either FPGA by communicating with the BSCAN primitives embedded within the FPGA designs. The BSCAN primitive allows FPGA designs to have access to the external JTAG communication protocol and is used in TURTLE for managing control and data transfer of the fault injection process.

The JCM also contains a network interface and communicates with the outside world through TCP/IP. In a typical configuration, an external host computer connects to the JCM using Ethernet, and the user logs into the JCM Linux system remotely to execute the fault injection code. After the fault injection activities are complete, the user transfers the logs from the JCM to the host computer for post-processing.

A complete TURTLE system that includes two Nexys Video boards, an FMC coupler card, and a JCM is shown in Figure 6. As seen in this figure, the TURTLE system is made from relatively inexpensive FPGA boards and commodity components. The following section will describe how designs are prepared for fault injection on the TURTLE system.

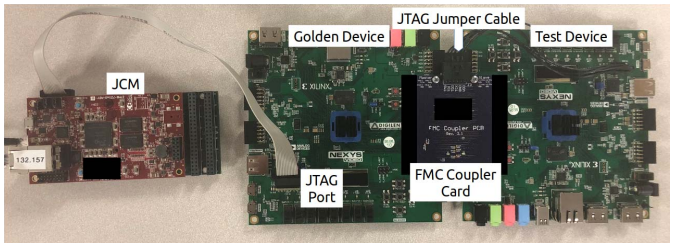

Fig. 6: A JCM controlling a fault injection test on a single TURTLE configuration

V. TURTLE Design Preparation

In the TURTLE platform, special design preparations and considerations must occur before a design is ready to be tested with fault injection. These preparations include inserting submodules, shown in Figure 7, into both the golden device and DUT. The following subsections will discuss the special design submodules and considerations.

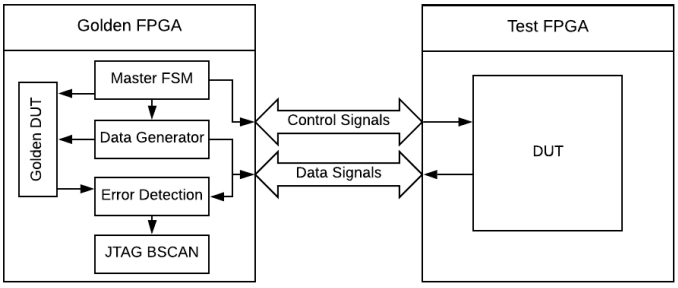

Fig. 7: Different submodules are inserted into each device for design preparation before testing

A. Design Under Test

In both the golden and DUT FPGAs, the majority of available logic is dedicated to implementing the DUT. A limitation of the TURTLE platform is the limited I/O across the FMC coupler card (only 64 data signals are available). For applications and designs that require more I/O, a reduction network needs to be implemented to reduce the I/O to the number of signals available through the FMC coupler card. In the TURTLE architecture, the DUT FPGA contains minimal additional logic for resets and control signals. The golden FPGA, however, contains other submodules that are explained in detail in the following subsections.

B. Error Detection Logic

To achieve synchronous lockstep comparison between the responses of the golden device and DUT, the golden FPGA implements an error detection submodule. For each input test vector, the error detection logic receives corresponding output responses (see Figure 7) from both devices. The responses are compared to determine if there is an error in the DUT response. An error is detected when the DUT response does not match the one given by the golden DUT.

C. Master Finite State Machine

An additional submodule implemented in the golden FPGA is a finite state machine (FSM) that aids in synchronizing the two devices. This FSM has four states. They are the initialize, reset, delay, and compare states.

Initialize The initialize state initializes reset and control signals when both devices are programmed for a fault injection test.

Reset The reset state is used to help synchronize the designs as well as issue internal resets when necessary during fault injection.

Delay The delay state is used in the golden FPGA to determine when the output response from the DUT is valid. The total time delay is design-dependent and is necessary to ensure the golden FPGA compares valid output data from both circuits.

Compare The compare state enables the golden FPGA to compare output responses from both devices and determine if a failure has occurred.

This FSM plays an important role in ensuring the DUT and golden device operate synchronously as well as aids in the fault injection approach that will be discussed in a later section.

D. User BSCAN Interface

The golden FPGA implements BSCAN primitives along with custom HDL code to make the system status and internal data signals available through JTAG. The TURTLE platform uses a system status that is 32 bits. The least significant byte in the 32-bit value is updated when failures are detected. The most significant bit in this byte is the system failure bit, which is set to 1 when a failure occurs. The lower 7 bits of this byte contains information about individual domains when implementing TMR designs. The rest of the bits in the system status are driven by a predetermined constant value. When the lower byte is all zeros, and the rest of the system status reflects the predetermined constant, the system is in a known good state with no errors. During fault injection, the JCM queries and monitors the system status through this BSCAN submodule.

VI. TURTLE Fault Injection Approach

The fault injection approach implemented in the TURTLE system uses custom software and hardware on the JCM. This section describes the fault injection algorithm implemented on the TURTLE platform, as shown in Figure 8. The flow explained in the section is a fault emulation methodology that injects faults into the CRAM of the DUT.

A. Initialization

The first step in a fault injection test on the TURTLE is initialization. An important aspect of initialization is calibrating the operating clock speed of the JTAG chain. Speed calibration is performed to determine the maximum operating clock speed for data transfer through the JTAG chain. Calibration for maximum speed must account for cable length and number of devices on the JTAG chain. The maximum data transfer speed determined during calibration affects the overall fault injection speed. After calibration of the JTAG clock speed, the golden device and DUT are programmed with bitstreams through JTAG.

After the devices are programmed, the two designs must be synchronized to ensure both circuits receive and process the same test vectors on the same clock cycle. The golden device FSM helps with the synchronization process. Internal resets are applied by the FSM to reset and synchronize the circuits in the devices. This synchronization is essential to be able to

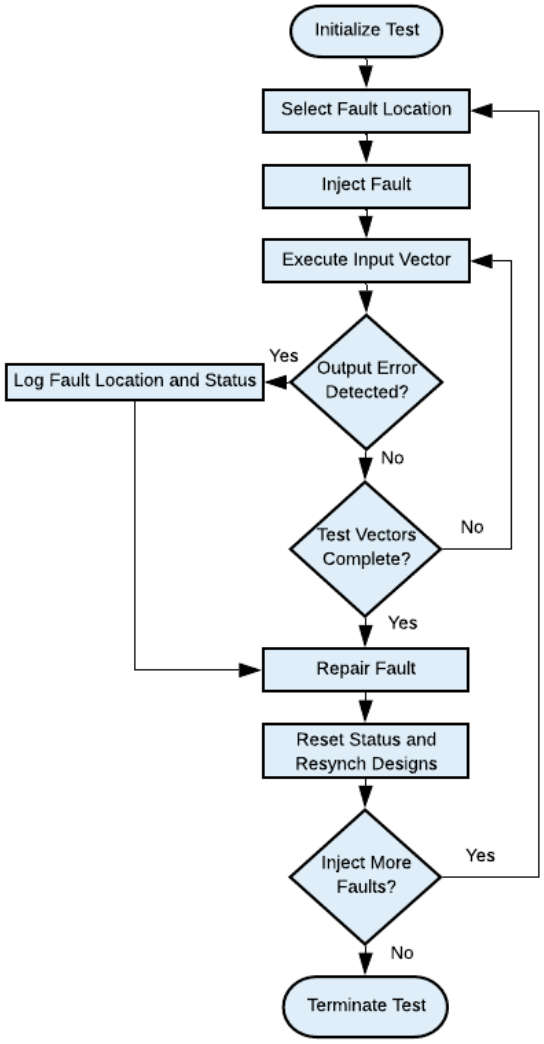

Fig. 8: Fault injection flow in the TURTLE platform

run both designs in lockstep and compare output results from the golden device and DUT cycle-by-cycle. Once the system status shows the designs are synchronously operating, the fault injection test can proceed.

B. Fault Location Selection and Injection

After initialization, software on the JCM selects a CRAM bit where the fault will be injected (see Figure 8). At this stage of fault injection, the test enters a continuous loop. Figure 8 shows this loop that starts with fault location selection and terminates when there are no more faults to be injected. There are a variety of ways that fault location selection can be performed on the TURTLE. These selection methods are random, targeted, sequential, and multi-cell.

Random Random fault injection consists of selecting a random frame, word, and bit in the CRAM as the fault to inject. This form of bit selection is similar to what might occur in a radiation test facility and is often used to predict what will happen during radiation testing.

Targeted Targeted fault injection selects specific CRAM bits or device tiles to inject as specified by a user-generated file. This form of bit selection is often used to "replay" faults that occurred at a radiation test facility.

Sequential Sequential fault injection injects faults sequentially in the bitstream from the first location to the last location. This form of bit selection is used to exhaustively test the full bitstream.

Multi-Cell Multi-cell fault injection injects multiple faults into the bitstream according to predetermined multi-cell fault injection patterns. This bit selection is used to more accurately emulate what occurs during radiation where multiple CRAM bits are upset by a single ionizing particle.

After selecting the fault location within the CRAM, the JCM injects the fault through partial reconfiguration. To inject the fault, the JCM performs the following steps:

1) Read the target frame(s) from the DUT,
2) Invert the value at the target frame, word, and bit within the CRAM data,
3) Write the corrupted CRAM data using partial reconfiguration to the DUT, and
4) Read the target frame, word, and bit to verify the fault injection.

C. Design Execution

After the fault is injected in the DUT, both designs operate on input test vectors, as shown in Figure 8. While faults are being injected, the two circuits execute continuously and synchronously on the same input test vectors provided by the golden device. After each fault injection, a predetermined delay occurs before the JCM can check the system status. This delay is calculated using the clock frequency, and the number of input vectors applied to the DUT. The delay must be sufficient enough such that the DUT has time to process all input vectors with the fault present in the system. This delay allows time for errors caused by CRAMS to propagate to the outputs of the design.

D. Error Checking and Recovery

After every input test vector, a corresponding output response from both devices is sent to the golden device. These two responses are compared to determine if there is an error in the DUT response, and the system status is updated accordingly. The JCM continuously monitors the system status to log when an error is detected. This section will discuss the specific steps performed for when an error is detected, and then when no error is detected. First, if an error is detected, the following steps take place:

1) The golden device sets the failure bit in the system status to 1,
2) The status is written to a BSCAN register,
3) Cycle-by-cycle data of output responses from both devices are logged to BSCAN registers,
4) The JCM reads the system status from the BSCAN registers on the golden device through JTAG,

5) The injected bit is classified as sensitive and logged with the system status,
6) The CRAM bit is repaired,
7) The system status is reset, and
8) If more faults are to be injected, testing proceeds, otherwise the test terminates.

The response data logged in step 3 is for post-processing of data in both fault injection and radiation testing. During the fault injection campaign, the system status must be reset after an error occurs. In the TURTLE, the method used to reset the status varies depending on the severity of the error. First, the system status is reset in the BSCAN register through JTAG. If the error persists, the JCM issues a reset signal to the golden device, which then applies resets to both circuits. Finally, if the error continues, the DUT is reprogrammed.

Second, if no error is detected, the designs completely process all input test vectors. After processing all test vectors, the CRAM bit is repaired. If more faults are to be injected, testing proceeds, otherwise the test is terminated.

Upon test termination, all logged data, including each injection and recorded failure with their corresponding system status, are written to output files. Precise error detection and data logging ensure each injection and DUT response is recorded for further analysis.

E. Performance

The rate of fault injection on the TURTLE plays an important role in collecting significant amounts of fault injection data. A single injection consists of reading, writing, and rereading the target frame. The last read operation is to verify the fault injection. In addition to reading and writing the target frame, the speed of fault injection is affected by the length of the implemented propagation delay in the fault injection algorithm. These factors affect the overall speed of fault injection on the TURTLE. Taking into consideration these factors, the JCM, with a JTAG clock speed of 50 Mhz, injects faults at an average rate of 95 injections per second. The Artix-7 device bitstream contains 77,845,216 total bits. Given this bitstream length and the average rate of fault injection on the TURTLE, it would take approximately 238 hours to exhaustively test all bits in the bitstream using a single TURTLE.

VII. PCMF Fault Injection Campaign

The TURTLE has been used in many successful fault injection campaigns to estimate the sensitivity of a variety of FPGA designs and to test a variety of SEU mitigation techniques. Tests were performed on B13 FSMs, MD5 Hash cores, AES cores, and SHA3 cryptographic hash functions. For each baseline design, different SEU mitigation techniques were applied to create different design variations. A fault injection campaign was created to test these different designs and SEU mitigation techniques.

The TURTLE was used to perform an important fault injection campaign to test a specific SEU mitigation technique. This technique is called Placement Common Mode Failure (PCMF) [10]. Common mode failures refer to potential single CRAM bits that can cause TMR to fail within a mitigated FPGA design. This mitigation technique addresses low-level common mode failures that can occur in TMR designs due to the physical placement of the FPGA design. A placed design is analyzed to determine potential placed resources within the device that could experience common mode failures. After this analysis, the PCMF technique modifies the placement of an FPGA design to reduce the risk of common mode failures. The rest of this section will summarize how the TURTLE platform was used to validate the PCMF SEU mitigation technique.

TABLE II: Fault Injection Results for B13 [9]

Metric/ Technique	Number of Injections	Number of Failures	Bit Sensitivity	Coefficient of Variation	Improvement
Unmitigated	$45,542$	617	1.35×10^{-2}	4.01×10^{-2}	$1\times$
Common-IO (1-Voter)	$2,000,000$	$12,027$	6.01×10^{-3}	9.10×10^{-3}	$2.3\times$
Common-IO (3-Voter)	$2,000,000$	$2,791$	1.40×10^{-3}	1.89×10^{-2}	$9.7\times$
Split-IO	$2,000,000$	39	1.95×10^{-5}	1.60×10^{-1}	$695\times$
Split-clock	$373,836$	307	8.21×10^{-4}	5.71×10^{-2}	$16.5\times$
Split-clock-PCMF	$336,939$	200	5.94×10^{-4}	7.06×10^{-2}	$22.8\times$
ES	$3,255,262$	70	2.15×10^{-5}	1.20×10^{-1}	$630\times$
ES-PCMF	$2,000,000$	31	1.55×10^{-5}	1.80×10^{-1}	$874\times$
Trip-IO (1-Voter)	$2,000,000$	$14,872$	7.44×10^{-3}	8.17×10^{-3}	$1.8\times$
Trip-IO (3-Voter)	$2,000,000$	26	1.30×10^{-5}	1.96×10^{-1}	$1,042\times$
PCMF	$2,000,000$	6	3.00×10^{-6}	4.08×10^{-1}	$4,516\times$
Striping	$2,000,000$	0^{*}	5.00×10^{-7}	1.00	$27,096\times$

* one error is assumed for metric calculations.

A fault injection campaign was developed and implemented to validate the PCMF technique and collect enough data to obtain statistical confidence that the technique reduced design sensitivity. This campaign tested a variety of different mitigation techniques on the 4 different designs mentioned at the beginning of this section. However, results in Table II are only for the B13 benchmark circuit [11]. The resulting estimated design sensitivities were compared to determine the improvement provided by the PCMF technique. 11 different mitigation techniques were applied to the B13 design. These B13 design variations were created using the design preparation strategy described in Section V.

The unmitigated technique is the baseline design. The common-IO techniques apply TMR to all logic expect the I/O of the device coming from the FMC coupler card. There are two variations of this technique, both triplicated and single voter networks. Like common-IO, triplicated-IO (trip-IO) applies TMR to all logic in the design, including I/O from the FMC coupler card [9]. There are three split-IO/clock techniques, including one that uses PCMF. These techniques split the I/O or clock signals within the design to attempt to reduce design sensitivity. The early split (ES) techniques force I/O and clock signals to split early in the device close to the physical pin. TMR is not applied to the split signals but is applied to other design logic. Finally, PCMF and striping are techniques that, in addition to applying TMR, constrain the placement of the design to reduce design sensitivity.

Table II shows the improvement of each technique compared to the unmitigated baseline B13 design. After the unmitigated design, there is a gradual increase in improvement as each technique attempted to apply implements an improved mitigation scheme. The best technique, striping, showed a 27,096$\times$ improvement over the unmitigated. The duration of the fault injection campaign was 58 days. This period included preliminary fault injection as well as verification of data through replaying the data. A total of 20,011,579 injections were performed on the TURTLE for this campaign on the B13 design. Table II also shows that the coefficient of variation for each technique. It should be noted that through the TURTLE platform, enough faults were observed to obtain relatively low variations in the estimated sensitivity values.

The TURTLE platform provided the ability to inject a large number of faults at a relatively low cost. This large amount of data helped reduce the coefficient of variation for the estimated sensitivity. The reduction in the coefficient of variation shows the TURTLE was successful in collecting statistically significant data to help validate the PCMF mitigation technique.

Table II shows results only for the B13 design, but a total of 170,346,105 injections were performed across the B13, MD5, SHA3, and AES cores with multiple mitigation techniques. This campaign was accomplished using 15 different TURTLE platforms. These 15 TURTLEs operated in parallel during the fault injection campaign to inject as many faults as possible. These TURTLE platforms were successful in performing the fault injection campaign and collecting data that was used to help validate the PCMF mitigation technique.

VIII. Conclusion and future work

The TURTLE fault injection platform presented in this paper was built using relatively low-cost hardware. The fault injection system implemented in this work was successful in testing and implementing a variety of FPGA designs and SEU mitigation techniques.

A successful fault injection campaign of over 170 million injections was conducted using the TURTLE to help validate the PCMF SEU mitigation technique. Using multiple TURTLEs in parallel, the campaign was able to provide a statistically significant amount of injections. The TURTLE provides a platform where SEU mitigation can be injected with large amounts of faults to observe failures. The amount of data collected on the TURTLE can be used to calculate the coefficient of variation, and help establish greater statistical confidence in SEU mitigation techniques.

Future areas of research will include improvements to fault injection speed. This includes improved methods for executing parallel fault injection tests on the TURTLE platform. Future work will also expand the TURTLE concept and architecture to other FPGA devices and designs. Specifically, fault injection tests for different designs and SEU mitigation techniques on the UltraScale FPGA will be pursued and researched.

References

[1] H. M. Quinn, D. A. Black, W. H. Robinson, and S. P. Buchner, "Fault simulation and emulation tools to augment radiation-hardness assurance testing," *IEEE Transactions on Nuclear Science*, vol. 60, no. 3, pp. 2119–2142, June 2013.

[2] M.-C. Hsueh, T. K. Tsai, and R. K. Iyer, "Fault injection techniques and tools," *Computer*, vol. 30, no. 4, pp. 75–82, April 1997.

[3] F. Lima, C. Carmichael, J. Fabula, R. Padovani, and R. Reis, "A fault injection analysis of Virtex FPGA TMR design methodology," in *RADECS 2001. 2001 6th European Conference on Radiation and Its Effects on Components and Systems (Cat. No.01TH8605)*, Sep. 2001, pp. 275–282.

[4] M. Alderighi, F. Casini, S. D'Angelo, S. Pastore, G. R. Sechi, and R. Weigand, "Evaluation of single event upset mitigation schemes for SRAM based FPGAs using the FLIPPER fault injection platform," in *22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT 2007)*, Sep. 2007, pp. 105–113.

[5] F. L. Kastensmidt, L. Sterpone, L. Carro, and M. S. Reorda, "On the optimal design of triple modular redundancy logic for SRAM-based FPGAs," in *Design, Automation and Test in Europe*, March 2005, pp. 1290–1295 Vol. 2.

[6] A. Gruwell, P. Zabriskie, and M. J. Wirthlin, "High-speed FPGA configuration and testing through JTAG," *2016 IEEE AUTOTESTCON*, pp. 1–8, 2016.

[7] "Nexys Video FPGA board reference manual," vol. 82, no. 6, Mar 2017. [Online]. Available: https://reference.digilentinc.com/_media/reference/programmable-logic/nexys-video/nexysvideo_rm.pdf

[8] "7 series FPGAs configuration (ug470)," Aug 2018. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug470_7Series_Config.pdf

[9] M. Cannon, "Improving the single event effect response of triple modular redundancy on SRAM FPGAs through placement and routing," Ph.D. dissertation, Brigham Young University, August 2019.

[10] M. J. Cannon, A. M. Keller, H. C. Rowberry, C. A. Thurlow, A. Prez-Celis, and M. J. Wirthlin, "Strategies for removing common mode failures from TMR designs deployed on SRAM FPGAs," *IEEE Transactions on Nuclear Science*, vol. 66, no. 1, pp. 207–215, Jan 2019.

[11] F. Corno, M. S. Reorda, and G. Squillero, "RT-level ITC'99 benchmarks and first ATPG results," *IEEE Design Test of Computers*, vol. 17, no. 3, pp. 44–53, July 2000.