An Open-Source Framework for Xilinx FPGA Reliability Evaluation

Aitzan Sari, Vasileios Vlagkoulis, Mihalis Psarakis

Department of Informatics, University of Piraeus
Piraeus, Greece
aitsar@unipi.gr, v.vlagkoulis@unipi.gr, mpsarak@unipi.gr

***Abstract*—The paper presents an open-source framework that provides access to the FPGA configuration memory and circuit logic via the JTAG protocol. By implementing various configuration memory functions, such as bitstream readback and verify, configuration frame write and read, configuration frame ECC monitoring, our tool can be used to support the design and evaluation of FPGA reliability methodologies. It mainly consists of: i) an on-chip logic to provide access to the configuration memory, embedded ECC core and DUT logic, ii) a software library of low-level JTAG functions and iii) a set of high-level configuration functions (GUI) to enable the development of target applications. The main benefits of the proposed framework, except its open-source architecture, are its low cost (no extra hardware), non-intrusiveness (no DUT modifications) and expandability (it can be easily extended to support new FPGA families). We demonstrate the applicability of the proposed framework with three use cases: radiation testing logger, configuration memory scrubber and fault injection platform.**

Keywords—FPGA, reliability, JTAG protocol, fault injection, scrubbing

I. Introduction

Contemporary FPGAs are ever gaining acceptance in industry sectors, such as space avionics, where they had very low penetration in the previous decades, mainly due the benefits they provide in performance and miniaturization of the electronics systems [1]. However, the vulnerability of SRAM-based FPGAs in radiation effects, such as Single Event Effects (SEEs), must be tackled in order to enable the adoption of these devices in critical systems with strict reliability and availability requirements. Thus, new open-source frameworks that can support the FPGA reliability assessment will promote the deployment of novel FPGA-based aerospace systems.

These open-source frameworks should integrate FPGA configuration capabilities, such as configuration bitstream readback and verify, configuration frame write and read, configuration frame ECC (error correction code) monitoring, etc., in order to support the design and evaluation of soft error mitigation approaches. Using these configuration utilities, the FPGA design engineers can improve their FDIR (fault detection isolation and repair) strategy by developing rigorous fault injection campaigns and efficient fault mitigation approaches or even assist the verification and debugging process by improving the circuit observability.

Several closed-source FPGA configuration frameworks, mainly in the form of fault injection platforms, have been proposed in the literature. The most advanced fault injection tools exploit the dynamic configuration features of SRAM FPGAs, such as readback, capture and partial reconfiguration in order to monitor the configuration and user memory and induce a bit-flip, emulating a Single Event Upset (SEU), in the configuration memory or the user registers. Some approaches require extra hardware [2],[3], e.g. dedicated boards, in order to orchestrate the fault injection process, communicate with the Device Under Test (DUT) and implement the golden DUT for error detection purposes, while they interface with the target FPGA via an external access port (e.g. SelectMap). Some other approaches use the internal configuration access port (ICAP or PCAP) in order to increase the reconfiguration speed [4]-[7], while the fault injection process is controlled by on-chip logic, e.g. a hard processor core [4],[5], a soft processor core [6] or custom hardware logic [7]. All the above approaches are highly intrusive since the fault injection functionality has been integrated within the target device at a large extend. Due to this, the fault injection process runs at system speed, but it affects the DUT characteristics (placement and routing) introducing "noise" in the reliability evaluation results of the original DUT.

Some recent approaches propose low-cost, non-intrusive fault injection environments [8],[9],[10] given that they do not require extra costs for additional equipment (e.g. golden DUT board), do not modify the DUT and do not impose extra logic in the target FPGA device. In all these approaches, the fault injection manager runs on a host PC that communicates with the target board and configures the DUT via the JTAG interface. The SEU generation is performed in the fault manager (host PC) and the fault injection is based on the partial reconfiguration of the FPGA bitstream through the JTAG port. In some cases, a second communication port (e.g. UART) is also supported for DUT monitoring purposes. Despite being slower than other configuration interfaces, JTAG is the most appropriate option for FPGA configuration tools due to its universality, low overhead and small radiation-sensitive cross section [11],[12]. In [11], a radiation-hardened FPGA is used to implement the JTAG controller, while an embedded Leon3-FT processor runs the configuration memory scrubbing process. In [12], the JTAG configuration manager, implemented in a separate board, consists of a Linux-based software library with FPGA configuration functions running on an embedded ARM processor and a low-level JTAG state

This work was presented at the first Workshop on Open-Source Design Automation (OSDA), 29 March 2019, in Florence, Italy. https://osda.gitlab.io

machine implemented in an FPGA device. Notice that in the last two approaches, the JTAG-based configuration tool is used not only for fault injection, but also for memory scrubbing and debugging purposes.

Towards this end, we present an open-source framework that provides several FPGA memory configuration functions to facilitate various reliability evaluation and improvement methodologies, such as fault injection, memory scrubbing, memory recording during radiation experiments, etc. The proposed framework provides access to the FPGA DUT through the JTAG interface using Xilinx Vivado TCL commands. It mainly consists of: i) an *on-chip logic* to provide access to the FPGA configuration memory, embedded ECC logic and DUT logic. Different versions of the on-chip logic are provided to enable different test setups, ii) a *JTAG configuration engine:* includes Vivado TCL scripts that implement JTAG functions for accessing the FPGA configuration memory and iii) *high-level configuration functions (GUI)* to enable the development of relevant applications.

Except its benefits as an open-source tool, the proposed framework also inherits the features of the low-cost, non-intrusive approaches [8],[9],[10] since it does not require: i) dedicated hardware (e.g. extra boards) and ii) DUT modifications (only the minimum hardware logic is integrated within the target FPGA device to enable access to the configuration memory). In addition, due to its generic and open architecture, the proposed framework is extendable (i.e. new FPGA families can be easily integrated in the list of the supported FPGA devices) and user friendly (i.e. the generic GUI enables the easy development of the target reliability applications). Furthermore, the client-server architecture of the framework enables the development of parallel and distributed approaches in order to handle the concurrent activation of several host processes or the connection to multiple FPGA boards. The last scenario allows the deployment of a parallel fault injection platform. The idea of parallel fault injection into multiple identical programmable SoCs to reduce testing time has been recently proposed in [13].

II. Open-Source Framework For Accessing FPGA Configuration Memory

A. Framework Architecture

The proposed framework provides a complete set of functions for accessing the FPGA configuration memory, and thus, to support the development of reliability-related applications that need to monitor and modify the configuration memory. Meanwhile, it aims to simplicity, providing an out-of-the-box and cross-platform tool which can be modified and extended easily to support different FPGA devices without requiring any specialized hardware tools. The framework is composed of three major components as shown in Figure 1:

i) *on-chip logic:* provides access to the FPGA configuration memory and user logic. It is a custom logic that serves as an interface of the configuration memory and the user logic to the JTAG configuration port.

ii) *JTAG configuration engine* (JTAG-CE): provides the implementation of a TCP server and low-level JTAG functions (in the form of TCL functions) for accessing the FPGA configuration memory/registers and the user logic through the Vivado tool.

iii) *High-level configuration functions (GUI)*: consists of the user application, which supports widget implementation, and the interface functionality (TCL interface handler) with the JTAG configuration engine.

The target (user) reliability application, such as a fault injection tool or a configuration memory-scrubbing manager, is built on top of the framework. The proposed framework has been designed using the Qt and PySide2 frameworks. Qt is a well-known and highly appreciated cross-platform application and User Interface (UI) development framework [14]. PySide2 is a Python binding [15] for Qt, which gives the opportunity to use the rich set of functionalities provided by the native implementation of Qt using the Python programming language. We should note here that the interpreted nature of Python as well as its flexibility and easy-to-understand, boost its popularity and is currently ranked at #3 [16] in TIOBE index. These factors guided us to choose Python programming language, which along with the TCL scripting supported by the Xilinx Vivado tool, are packed in an open-source framework.

The user application typically uses the Application Programming Interfaces (APIs) exposed by the TCL interface handler to communicate with the JTAG-CE in order to read/write device's configuration memory and configuration registers. Initially, the TCL interface handler starts a Vivado instance in batch mode and a TCP client. The TCP client is used as Inter-Process Communication (IPC) between the application thread and the JTAG-CE, which runs in a separate thread as a TCP server. The use of TCP client-server scheme improves significantly the execution time overhead introduced by the Vivado instance. Instead of running a Vivado instance upon every script/command execution, the framework creates a single instance during the execution lifetime of the application, eliminating thus the overhead for setting-up the JTAG connection for every script. Notice that, in our experimental setup described in Section III, the time required for initiating a JTAG connection is about 10 seconds including Vivado start-up, open the hardware target and connect to the Xilinx hardware server.

Furthermore, the TCP client-server solution enables the development of multi-server and distributed approaches, where the application can communicate with multiple servers targeting either different FPGA devices (assuming this feature is enabled in the Vivado instance) or multiple user

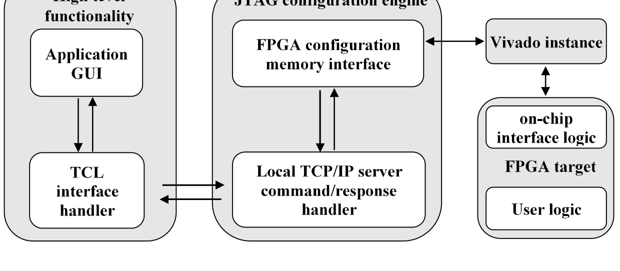


Figure 1: Framework architecture

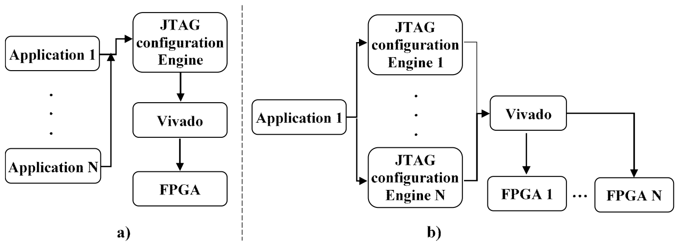

Application 1
Application N
JTAG configuration Engine
Vivado
FPGA
a)
Application 1
JTAG configuration Engine 1
JTAG configuration Engine N
Vivado
FPGA 1
FPGA N
b)

Figure 2: Using the TCP client-server approach: a) Multiple applications using a single JTAG configuration engine and b) Single application using multiple JTAG configuration engines

applications running in the same platform targeting the same FPGA device. For example, a multi-server scenario could be as follows: a fault injection tool and a memory scrubbing process are built on top of our framework and run in parallel for the same target FPGA. In this case, the scrubbing process executes continually or periodically to detect and correct any errors in the configuration memory of the device, while the fault injection tool introduces single or multiple faults in the FPGA configuration memory randomly or on-demand. Thus, the fault injection and the memory scrubbing tasks are performed asynchronously and independently allowing SEU mitigation techniques to be evaluated more effectively. The TCP client-server approach is illustrated in Figure 2.

Although multiple applications and/or JTAG configuration engines (servers) may exist, only a single Vivado instance runs in the host machine to keep the resource allocation as low as possible since main memory requirements of Vivado may be high. This is not mandatory though but can be adjusted according to the final application requirements. As already mentioned, Vivado is the key component to handle the JTAG protocol of the target FPGA.

B. *On-chip Logic*

The framework communicates with the FPGA device through the standard boundary-scan JTAG port (IEEE standard 1149.1). With the increase in complexity of FPGA devices in recent years, the JTAG has become the most well-established interface for FPGA configuration and debugging, despite its low speed (serial interface running typically up to 40 MHz). Given also that it is less sensitive to radiation effects compared to other configuration interfaces, since it has lower SEU and SEFI cross-section [11],[12],[17], it is widely used in reliability-aware applications. The write and read accesses to the configuration memory or the configuration registers are the most common JTAG operations. The JTAG also allows access to and from the internal FPGA logic for monitoring or debugging purposes. To activate a general-purpose communication port between the JTAG interface and the user design, one or more (up to 4 for the recent Xilinx FPGA device families) BSCAN primitive(s) must be instantiated while special JTAG USER instruction(s) provides access to the internal logic.

Three different alternatives of the on-chip logic have been implemented, based on the requirements of the target application:

i) *Basic setup*: This corresponds to a basic setup in order to enable write and read operations of the configuration frames (configuration memory) and the configuration registers, as shown in Figure 3. The monitoring of the user logic using the BSCAN primitive is optional. Tasks that can be performed by this setup include: full FPGA configuration, readback of the configuration memory and registers, fault injection to the configuration frame(s) and monitoring of the user logic (to detect erroneous behavior).

ii) *Frame ECC-based scrubbing setup*: This setup can be part of a configuration memory scrubber, which uses the embedded configuration frame-level ECC to detect SEUs and allows single or multiple frames to be scrubbed using dynamic partial reconfiguration through the JTAG interface (Figure 4). A FRAME ECC primitive provides access to the embedded ECC logic of the configuration frames. A FIFO retains the erroneous configuration frames (frame address, syndrome) detected by the FRAME ECC logic. The FIFO is read through the

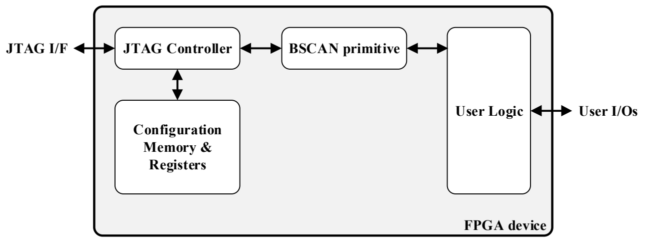


Figure 3: On-chip Logic - Basic setup

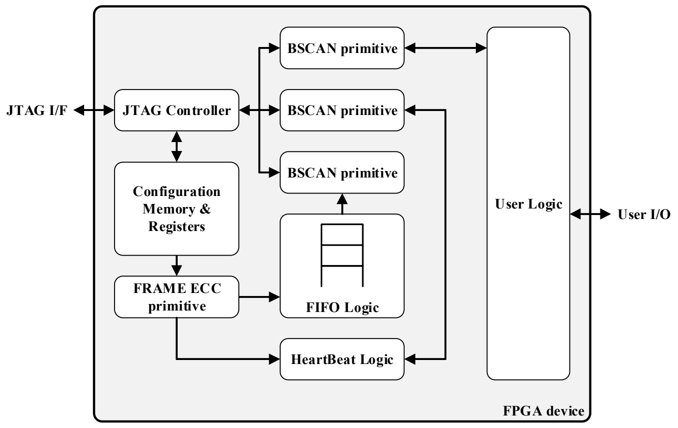


Figure 4: On-chip Logic – Frame ECC-based scrubbing setup

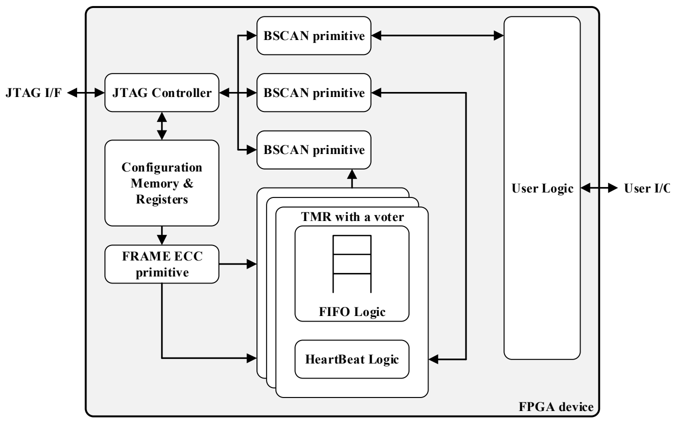


Figure 5: On-chip Logic – Hardened (TMR) version

BSCAN primitive periodically and in case of any error, repairing of the configuration memory is performed. An extra block named HeartBeat Logic, which is also accessed periodically, stores the state of the FRAME ECC when it reaches the last configuration frame during its scan operation; HeartBeat logic provides watchdog functionality in order to alarm malfunction of the frame ECC logic.

iii) *Hardened (TMR) version*: This setup version is for use in a radiation environment (e.g. radiation experiments). The FIFO and HeartBeat modules are triplicated and voted (in the fashion of triple modular redundancy – TMR), as shown in Figure 5. This scheme tolerates all the programmable resources of the on-chip logic against SEUs in order to provide a more robust and reliable setup under radiation conditions. In setups (ii) and (iii) we integrate multiple BSCAN primitives to interface different resources (i.e. User logic, FIFO logic, Heartbeat logic) instead of multiplexing all these into a single primitive. The multiple primitives simplifies the routing between the BSCAN and the corresponding components and facilitates the implementation of the Isolation Design Flow used for the hardened version of the on-chip logic.

C. *JTAG Configuration Engine (JTAG-CE)*

This section describes the low-level APIs provided by the JTAG configuration engine, which runs in a separate thread as TCP server listening in specific TCP port. The JTAG-CE accepts predefined commands from the target application and executes the associated low-level API while sending back a response to the application asynchronously. This means the user application should: i) send a dedicated command using the TCP client, ii) wait asynchronously for the response from the JTAG-CE component and iii) process the response as needed. Table I describes the low-level APIs implemented in the JTAC-CE as well as the associated TCP commands.

Let's use an example to illustrate the use of the provided APIs. Assume the user application wants to read 10 configuration frames starting from address 0x100 and save the content in a text file (D:\readback-data.txt). In this case the application, through the TCP client, sends the following command:

FRAMES_READ%D:\readback-data.txt%0x100%10

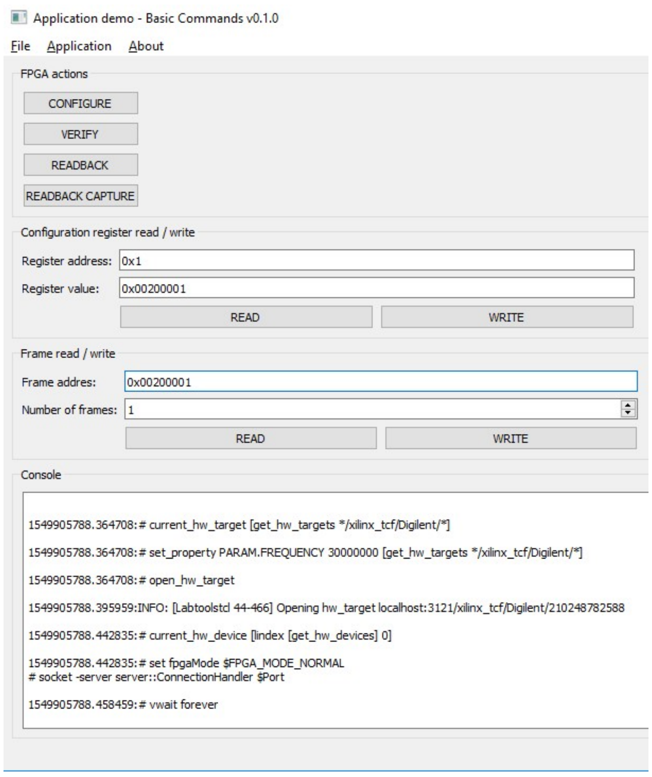
Note that the command and its parameters are separated by the *"%"* character. Upon reception of the command, the TCP server executes the *FrameRead* low-level function and sends back the response commands to the application with content **"FRAMES_READ%D:\readback-data.txt"** to notify the application that the command execution has finished. All the provided APIs respond with the issued command including parameters when necessary (the Configure API for example responds just with the called command: CONFIGURATION as opposed with the FrameRead which adds the readback file path). Although Table I mentions few APIs, new functions can be easily added inside the JTAG-CE to support different needs and/or the current available functions can be modified and extended as required.

D. *High-level Configuration Functions (GUI)*

The High-level configuration functions include the user application and the TCL interface handler, which adds an abstraction layer between the user application and the TCP client commands. As mentioned earlier, the TCL interface includes a TCP client to communicate with the JTAG-CE by sending predefined commands and getting any response sent from the JTAG-CE. As in JTAG-CE, the TCL interface component can be easily modified and expanded to support new functions. This is a three-step procedure:

TABLE I. LOW-LEVEL APIS FOR THE CONFIGURATION MEMORY ACCESS

API	Server Command	Parameters	Description
Configure	CONFIGURATION	bit_filepath: bitstream file path mask_filepath: mask file path	Configures the FPGA device given a bitstream file and a maskfile (the maskfile is used when verification is issued; see ReadbackVerify API)
Readback	READBACK	rdb_filepath: readback file path	Reads-back the FPGA device and saves its content in the target file
ReadbackCapture	READBACK_CAPTURE	rdb_filepath: readback file path	Reads-back the FPGA device in capture mode and saves its content in the target file
ReadbackVerify	READBACK_VERIFY	-	Verifies the FPGA against the configured bitstream file while using also the maskfile (see Configure API)
RegisterWrite	REGISTER_WRITE	register_address: target register address register_value: value to be written	Writes a configuration register
RegisterRead	REGISTER_READ	register_address: target register address	Reads a configuration register
FrameWrite	FRAMES_WRITE	frame_address: start address frame_filepath: file holding the frames data size: number of frames to write append_dummy_frame: True to append a dummy frame use_hex_format: True if frame data is in HEX reset_fifo: True to reset the internal FIFO (Figures 4 and 5)	Writes a given number of frames to the configuration memory
FrameRead	FRAMES_READ	readback_filepath: target file path (where data be saved) frame_address: start address size: number of frames to be read overwrite: True to overwrite the file	Reads a given number of configuration frames from the FPGA. The read data is saved in the target file.



Register address: 0x1
Register value: 0x00200001
Frame addres: 0x00200001
Number of frames: 1

```
1549905788.364708:# current_hw_target [get_hw_targets */xilinx_tcf/Digilent/*]

1549905788.364708:# set_property PARAM.FREQUENCY 30000000 [get_hw_targets */xilinx_tcf/Digilent/*]

1549905788.364708:# open_hw_target

1549905788.395959:INFO: [Labtoolstcl 44-466] Opening hw_target localhost:3121/xilinx_tcf/Digilent/210248782588

1549905788.442835:# current_hw_device [lindex [get_hw_devices] 0]

1549905788.442835:# set fpgaMode $FPGA_MODE_NORMAL
# socket -server server::ConnectionHandler $Port

1549905788.458459:# vwait forever
```

Figure 6: Example application illustrating the proposed framework

i) implement the low-level API inside JTAG-CE, ii) map the implemented function with a new server command and iii) implement the new high-level function inside the TCL interface (sending the mapped command to the TCP server).

Figure 6 shows an application example with a simple GUI for executing the mentioned APIs. The application can be used for configuration memory read/write, configuration register read/write as well as FPGA configuration and readback.

III. Use Cases

The proposed open-source framework has been demonstrated for different target applications. It was actually implemented for the needs of a radiation testing experiment performed in CERN for the evaluation of the susceptibility of Xilinx Zynq-7000 devices in Single-Event Effects (SEEs) when irradiated with high-energy heavy ions. Two different test setups were established:

i) *Test #1*: to calculate the SEU cross section of the Xilinx Zynq-7000 APSoC device by measuring the upsets in all the memory resources (Configuration memory, Block RAM and user Flip-Flops) of the FPGA device.

ii) *Test #2*: to evaluate the effectiveness of a mixed scrubbing scheme (internal scrubbing for the detection of upsets and external scrubbing for the identification and correction of single and multiple bit upsets)

A. *Radiation Testing Logger*

In the case of Test #1, our framework is used to record the bit upsets of the FPGA memories during irradiation. No SEU hardening or mitigation approach is applied in this test. The configuration memory is readback when the beam goes off and re-written in the case of upsets. Both readback and readback-capture commands are executed. Readback-capture is used to record the state of the internal CLB registers. Readback-verify is used to check for uncorrected upsets.

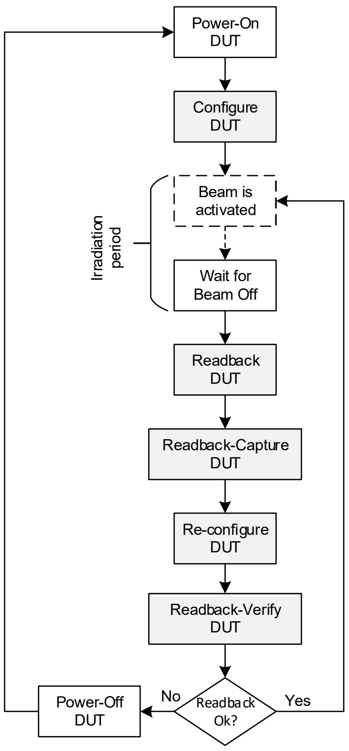


Figure 7: Test #1 flow diagram

The flow diagram of Test #1 is shown in Figure 7. The shaded functions of the flow diagram are performed using the high-level functions of the open-source framework. The execution times of the corresponding functions (FPGA memory configuration, readback, readback capture and readback verify) are presented in Table II.

B. *Configuration Memory Scrubber*

In the case of Test #2, our proposed framework is used to check the embedded ECC logic and read the corresponding FIFO (see the Frame ECC-based scrubbing setup in Section II.B), check the heartbeat logic, read a set of (erroneous) configuration frames and write a set of (corrected) configuration frames. The target application runs the error correction algorithm (the description of the algorithm is out of scope in this paper).

Table II. Execution Time of API Commands

API command	Execution time (sec)
configure	1.45
readback	4.10
readbackCapture	4.18
readbackVerify	14.11
writeFrames (1 frame)	0.060
readFrames (32 frames)	1.36
readRegister (ECC FIFO read, check HeartBeat)	0.020
writeRegister	0.016

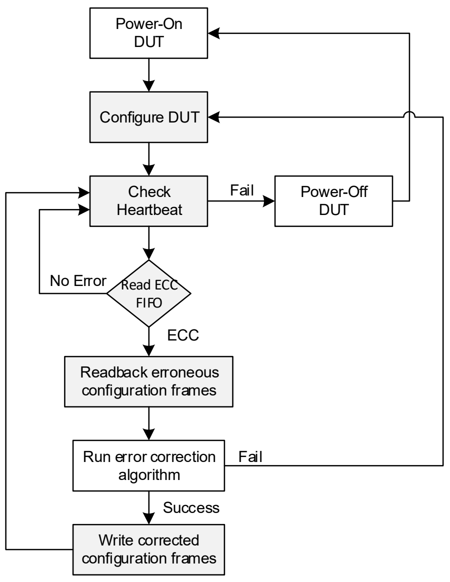

Power-On DUT
Configure DUT
Check Heartbeat
Fail
Power-Off DUT
No Error
Read ECC FIFO
ECC
Readback erroneous configuration frames
Run error correction algorithm
Fail
Success
Write corrected configuration frames

Figure 8: Test #2 flow diagram

The flow diagram of Test #2 is shown in Figure 8. The shaded functions of the flow diagram are performed using the high-level functions of the open-source framework. The execution times of the corresponding functions (FPGA memory configuration, check heartbeat, read ECC FIFO, readback a set of configuration frames and write a set of configuration frames) are presented in Table II.

C. Fault Injection Platform

Both the above setups were evaluated before the radiation experiments using fault injection tests. The proposed framework was also used to implement a fault injection tool. In any case, the fault injection runs concurrently with the target application, i.e. the Logger or the Scrubber, in a different client. For the implementation of the fault injection tool, the high-level functions used are the read and write configuration frame.

IV. Conclusion and Future Work

We presented an open-source framework that provides access to the FPGA configuration memory and circuit logic via the JTAG protocol. It provides a low-cost, non-intrusive tool to perform various configuration memory functions, such as bitstream readback and verify, configuration frame write and read, configuration frame ECC monitoring. Due to its open-source nature, the proposed framework can support the development of FPGA reliability-aware methodologies and consequently promote the deployment of FPGA-based systems in critical applications.

Note that the current software version requires the use of the Vivado tool in order to open the target DUT in JTAG mode through the USB interface and run basic boundary scan commands (e.g. "scan_ir_hw_jtag" and "scan_dr_hw_jtag" commands that access the boundary scan Instruction Register and Data Register, respectively). More complex Vivado TCL commands (e.g. "write_bitstream"), that are composed of simple JTAG commands, are also called by our tool. We aim at developing a software driver to handle the communication with the JTAG interface and perform the low-level JTAG commands in a future version of the framework in order to eliminate the reliance on closed software (e.g. Vivado).

The source code (TCL scripts, GUI, Python code for Test #1 and VHDL code for on-chip logic) is available from the github project FREtZ (FPGA Reliability Evaluation through JTAG) [18]. The project is licensed under the GNU GPLv3 (commercial use is permitted).

Acknowledgment

This research has been co-financed by the European Union and Greek national funds through the Operational Program Competitiveness, Entrepreneurship and Innovation, under the call RESEARCH–CREATE–INNOVATE (project code:T1EDK-04298).

References

[1] Furano G., Menicucci A. (2018) Roadmap for On-Board Processing and Data Handling Systems in Space. In: Ottavi M., Gizopoulos D., Pontarelli S. (eds) Dependable Multicore Architectures at Nanoscale. Springer, Cham.

[2] Aguirre, Miguel Angel, et al. "FT-UNSHADES: A new system for SEU injection, analysis and diagnostics over post synthesis netlist." Proc. NASA Military and Aerospace Programmable Logic Devices (2005).

[3] Mogollon, J. M., et al. "FTUNSHADES2: A novel platform for early evaluation of robustness against SEE." Radiation and Its Effects on Components and Systems (RADECS), 2011 12th European Conference on. IEEE, 2011.

[4] L. Sterpone, M. Violante, "A new partial reconfiguration-based fault injection system to evaluate SEU effects in SRAM-based FPGAs," IEEE Trans. Nucl. Sci, vol. 54, no. 4, 2007, pp. 965–970.

[5] Stoddard, Aaron, et al. "High-speed PCAP configuration scrubbing on Zynq-7000 All Programmable SoCs." Field Programmable Logic and Applications (FPL), 2016 26th International Conference on. IEEE, 2016.

[6] Sari, Aitzan, and Mihalis Psarakis. "A fault injection platform for the analysis of soft error effects in FPGA soft processors." Design and Diagnostics of Electronic Circuits & Systems (DDECS), 2016 IEEE 19th International Symposium on. IEEE, 2016.

[7] Nazar, Gabriel L., and Luigi Carro. "Fast single-FPGA fault injection platform." 2012 IEEE international symposium on defect and fault tolerance in VLSI and nanotechnology systems (DFT). IEEE, 2012.

[8] Battezzati, Niccolo, Luca Sterpone, and Massimo Violante. "A new low-cost non intrusive platform for injecting soft errors in SRAM-based FPGAs." Industrial Electronics, 2008. ISIE 2008. IEEE International Symposium on. IEEE, 2008.

[9] M. Straka, J. Kastil, Z. Kotasek, "SEU simulation framework for Xilinx FPGA: first step towards testing fault tolerant systems," in Proc.IEEE DSD, 2011, pp. 223-230.

[10] Ghazaani, Elyas Abolhassani, Zana Ghaderi, and Seyed Ghassem Miremadi. "A non-intrusive portable fault injection framework to assess reliability of FPGA-based designs." Field-Programmable Technology (FPT), 2013 International Conference on. IEEE, 2013.

[11] Michel, Holger, et al. "Read back scrubbing for SRAM FPGAs in a data processing unit for space instruments." Adaptive Hardware and Systems (AHS), 2015 NASA/ESA Conference on. IEEE, 2015.

[12] Gruwell, Ammon, Peter Zabriskie, and Michael Wirthlin. "High-speed programmable FPGA Configuration through JTAG." Field Programmable Logic and Applications (FPL), 2016 26th International Conference on. IEEE, 2016.

[13] https://stfleming.github.io/files/fpt_2018_parallel_fault_injection.pdf

[14] https://www.qt.io

[15] https://pypi.org/project/PySide2/

[16] https://www.tiobe.com/tiobe-index/

[17] Carmichael, Carl, and Chen Wei Tseng. "Correcting single-event upsets in Virtex-4 FPGA configuration memory." Xilinx Application Note (XAPP1088) (2009).

[18] https://github.com/unipieslab/FREtZ