# PyXEL: Exploring Bitstream Analysis to Assess and Enhance the Robustness of Designs on FPGAs

Corrado De Sio, Sarah Azimi, Luca Sterpone
*Dipartimento di Automatica e Informatica (DAUIN)*
*Politecnico di Torino*
Turin, Italy
{corrado.desio, sarah.azimi, luca.sterpone}@polito.it

David Merodio Codinachs, Filomena Decuzzi
*European Space Research and Technology Centre (ESTEC)*
*European Space Agency (ESA)*
Noordwijk, The Netherlands
{david.merodio.codinachs, filomena.decuzzi}@esa.int

***Abstract*— Commercial hardware-reconfigurable systems-on-chip are highly attractive for mission-critical applications in the space and automotive industries. However, their vulnerability to soft errors is a major concern, and analyzing the robustness of these systems is a complex task due to the lack of dedicated tools, information, and methodologies available. PyXEL is a tool designed to address these issues, providing the methodology for automating reliability analysis based on radiation and fault injection campaigns and facilitating the development of mitigation solutions based on customized place-and-route. Furthermore, PyXEL offers the methodology for visualizing, decoding, and analyzing the configuration data of programmable hardware devices, enabling more precise and efficient evaluation and analysis of the robustness of systems implemented on programmable hardware devices.**

***Keywords*— *Fault Injection, FPGA, Reliability, Robustness***

## I. INTRODUCTION

Field Programmable Gate Arrays (FPGAs) and, more recently, reconfigurable Systems-on-Chip (SoCs) are gaining popularity for various purposes, from prototyping to high-performance applications. The space and automotive industries are particularly drawn to these devices due to their hardware reconfigurability, which allows them to adapt to different scenarios, reduce costs, and extend system lifetime. SRAM-based reconfigurable hardware devices, such as AMD-Xilinx FPGAs and SoC, allow the final user to configure the hardware resources available in the device fabric to implement nearly any circuit. The configuration of these devices is based on configuration data, formally a bitstream, which is uploaded to the configuration memory (CRAM). Sections of the configuration memory are dedicated to programming hardware resources in the device fabric based on the content of the memory cells. For instance, Look-up-Tables can be configured after manufacturing to implement custom combinational logic functions and can be connected one with the other using a mix of hardwired and programmable interconnections, the latter usually implemented by pass-transistors, to build complex logical functions. Similarly, programmable routing can be used to connect LUTs input and outputs with Flip-Flops, Block RAMs, or pins to implement sequential logic, storage mechanism, and input-outputs, respectively. AMD-Xilinx and Intel are the leading producers of SRAM-based programmable hardware devices. Due to the higher performance and newest technology adopted by commercial-off-the-shelf devices compared to radiation-hardened versions, they are increasingly being considered and utilized in safety-critical applications such as space missions.

### A. Motivations

Sensitivity to soft errors of these devices, especially to Single Event Upsets (SEUs), is a significant concern in the adoption of FPGAs and SoCs in safety-critical applications that must be addressed early during design flow [1][2]. One of the main challenges in adopting commercial programmable hardware-based devices for safety-critical applications is their susceptibility to Single Event Upsets (SEUs). An SEU results in an undesired corruption of the configuration memory data, thus producing an undesired modification of the implemented circuit that may lead to errors and, eventually, critical failures. Both ionizing and non-ionizing radiation, ranging from high-energy protons of outer space to neutrons and alpha particles on Earth, is a source of SEUs in the configuration memory of the devices. In order to evaluate the robustness of design to SEUs, both accelerated radiation testing, which exposes the system to a high flux of high-energy particles, and fault-injection-based analysis are commonly adopted. The function of the bits in the configuration data and their relation to the hardware resources are not provided by vendors, remaining largely unknown. The lack of information from the vendors on how resources in the device fabric are encoded in configuration memory significantly limits the ability of industry and researchers to develop more accurate and performant reliability evaluation methodologies, which could benefit from the knowledge of how a circuit can be modified by SEUs occurring in the configuration memory. This problem is even more critical in recent devices, where CRAM is characterized by more than 100,000,000 bits, making unfeasible an exhaustive evaluation and exacerbated by the fact that around 90% of the bits in CRAM are actually associated with unused hardware resources, leading to long evaluation campaigns to achieve a reasonable number of events. Indeed, due to programmable architecture's intrinsic characteristics, only a subset of the hardware resources will be used by the application implemented within the device. Consequently, only a subset of configuration memory bits will be a source of errors when corrupted. The missing reliable mapping between CRAM and hardware prevents many reliability evaluation techniques from being applied to these devices, such as fine-grained fault injection, specific fault emulation (e.g., open routing, LUT corruption), fault localization in radiation testing and fault injection campaigns, and more. Additionally, automatization and integration of bitstream analysis, fault injection tasks, and custom place-and-route solutions are inhibited by the lack of APIs and integration mechanisms, currently constrained to low-performance Tcl scripts.

### B. Related Works and Open Challenges

Some vendors, such as Xilinx, attempted to address issues in evaluating the reliability of design implemented in programmable hardware by providing a feature within their vendor tool, which identifies a subset of the entire CRAM to be considered for robustness analysis called Essential Bits (EBs). The purpose of EBs is to reduce the number of bits that must be evaluated during robustness analysis. However, EBs still do not provide information for mapping faults to specific modules or hardware resources. Additionally, the coordinates

979-8-3503-3265-0/23/$31.00 ©2023 IEEE

of EBs are provided in a proprietary format, meaning it can only be used to evaluate system reliability by fault injection based on a dedicated IP Core provided by the vendor, which increases complexity since it requires modifying the design for placing the core in the programmable hardware. Finally, EBs are based on used tiles rather than basic elements, which results in overestimating the subset.

The third-party tools available to support FPGA design analysis are mainly dedicated to facilitating and automating partial reconfiguration tasks and design placement and routing [5]-[8]. Except for [7], which partially supports coarse-grained bitstream manipulation with partial reconfiguration purposes, and [8] supporting user-defined initialization states update and configuration frame identification, bitstream analysis, manipulation, and mapping are not supported at all. Except for [9], which targets two-generation old Spartan-VI FPGA successfully, tools such as [9][10]-[12] have made attempts to reverse engineering bitstream, obtaining only partial results and targeting FPGA families that are three generations or more out of date compared to the current ones. Additionally, these tools are based on the Xilinx Design Language (XDL) supported by ISE, the previous iteration of the vendor development tool, but XDL is not a viable option for current devices since it is no longer supported in the more recent Vivado Design Suite, the provided vendor tool for development on Xilinx programmable devices. The discontinued support for XDL and the lack of APIs for interfacing with the Vivado framework raise an additional challenge to interface third-party tools with the vendor's tool, and automatizing tasks such as information extraction and custom place-and-route solutions for fault mitigation purposes.

### C. *Main Contribution*

This work presents PyXEL, a Python-based tool designed to facilitate the evaluation of the robustness of programmable hardware designs, as well as the implementation of mitigation strategies by simplifying the automation of place and route. PyXEL provides the mechanism for automating and integrating fault injection tasks, bitstream analysis, and custom place-and-route solutions. Thus, PyXEL provides comprehensive support for the automation of radiation testing and fault injection experiments, mandatory for evaluating the robustness of design implemented on programmable hardware. Additionally, it includes the methodology for visualizing, decoding, and analyzing the configuration data of programmable hardware devices. With this feature, solving the mapping between CRAM and hardware, PyXEL can support comprehensive reliability analysis techniques, such as fine-grained fault injection, specific fault emulation (e.g., open routing, LUT corruption), and fault localization in radiation testing that are not currently supported by vendors or third-party tools.

Please note that PyXEL was initially designed to make it easier to evaluate the effects of faults in FPGA routing [3]. The new version of the tool presented in this work has been rebuilt from the ground up and does not share any of the code, methodology, or features.

## II. METHODOLOGY

PyXEL offers several features that help to evaluate the reliability of designs implemented on programmable hardware thoroughly. It focuses mainly on Xilinx devices, the most used programmable hardware devices. Nevertheless, due to the general characteristics of the proposed approach, the methodology can be readily adapted for use with other programmable hardware families.

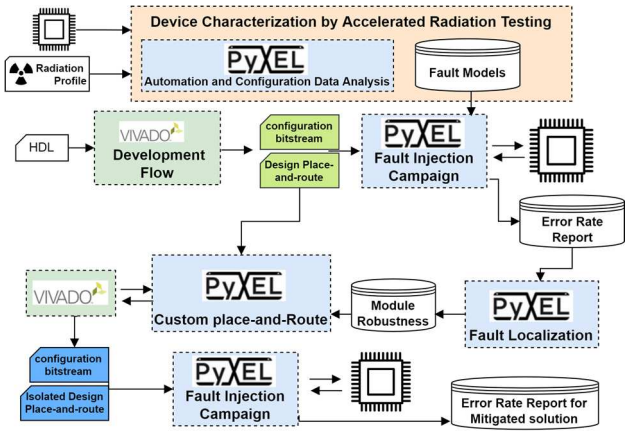


Fig. 1: Steps of the proposed case of study for the evaluation and mitigation of a benchmark circuit using PyXEL.

PyXEL can be used during radiation testing for automating experiment flow and extracting fault models occurring in the configuration memory. Additionally, PyXEL offers support for automating fault injection campaigns and mapping faults affecting the CRAM with hardware resources and modules implemented in the device. This allows for evaluating a design's robustness to soft errors and identifying modules with the highest sensitivity. Furthermore, the PyXEL integration mechanism supports the development of custom place-and-route solutions, such as module isolation for fault containment [4]. These features enable a precise and efficient evaluation and enhancement of design robustness, allowing for a comprehensive analysis, including fault model identification, robustness evaluation, and mitigation, based on the flow depicted in Figure 1.

### A. *Mechanism of Integration with the EDA tool*

Most EDA tools for FPGA development, particularly Vivado provided by Xilinx, offer a Tcl command interface for scripting. While Tcl is a standard language in the semiconductor industry, it can be slow and cumbersome, particularly when dealing with complex algorithms or object-oriented programming. Additionally, the development flow is often based on intermediate stages describing intermediate steps, such as synthesis and place-and-route, describing how a design is translated and mapped to the hardware resources in the reconfigurable device. As these file formats are proprietary, an equivalent high-level description of the intermediate stages must be extracted from Vivado to be managed in PyXEL. Integration between PyXEL and the EDA tools is based on a mechanism that wraps the Tcl interpreter internally to Vivado with a Tcl server. The server executes the request received on a dedicated socket by PyXEL directly in the Vivado-embedded interpreter, returning the results to PyXEL. This effectively wraps the Tcl interpreter, allowing PyXEL to offer optimized APIs for interacting with Vivado Tcl transparently to the user. This mechanism is essential for easing the development of complex algorithms, such as place-and-route approaches. It is also the main mechanism adopted for integrating Vivado operations within PyXEL, allowing for easy and transparent device and design queries and manipulation of place and route via in-tool APIs.

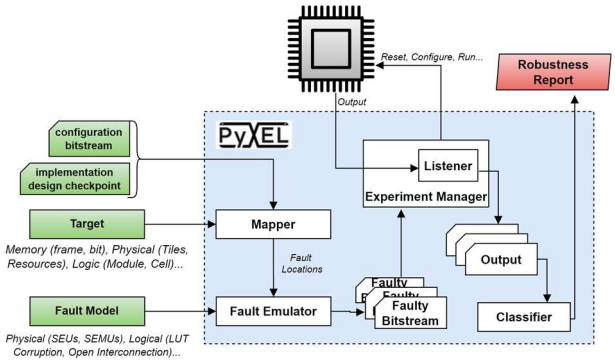


Fig. 2: Modules embedded in PyXEL for automatizing fault injection campaign for evaluating of the robustness of a design.

### B. *Automating Evaluation of Robustness*

PyXEL can be used for evaluating the robustness of design implemented on programmable hardware devices. An overview of the modules embedded in PyXEL to facilitate the automatization of radiation testing and fault injection campaigns on actual devices is depicted in Figure 2. For performing operations, such as device configuration, system reset, software deployment and execution for system-on-chips, and configuration memory readback, the modules exploit the previously introduced mechanism for EDA integration, which is used transparently by the user. Furthermore, PyXEL includes a *listener* module for receiving data from the device implemented as a parallel thread of the *Experiment Manager* to capture outputs on serial connections or sockets to be classified later that run. PyXEL can be instrumented to emulate different fault models at the memory level as well as at the logic elements level. This feature is managed by a *Fault Emulator* module able to emulate fault models in the configuration data, aware of the implemented design characteristics and the configuration memory architecture of the device, accordingly to the user instrumentation. PyXEL also provides a *Mapper* to relate bits in the configuration data with the associated hardware resources, so supporting fault localization during radiation testing and fault location selection during fault injection campaigns.

### C. *Bitstream Analysis and Decoding*

Configuration bitstreams are binary files used to program programmable hardware consisting of hundreds of millions of bits containing metadata, configuration data, and programming instructions. The structure of these devices consists of multiple tiles, each containing basic hardware resources that can be configured to implement a specific function. For example, these resources include elements such as programmable interconnects (PIPs) connecting routing paths or Look-Up Tables (LUTs) that can implement custom logic functions. Each tile is associated with specific bits in the

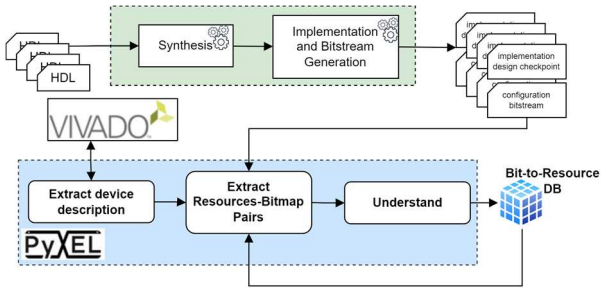


Fig. 3: Automatized Flow for associating configuration data with hardware resources.

configuration memory, configuring the elements within the tile. The configuration memory structure is made of frames, the smallest addressable group of bits in the configuration memory. Each frame typically contains around 3,000 bits, and the number of frames varies significantly depending on the device size and architecture. While vendors partially disclose the bitstream file structure, the actual function of the bits in the configuration data and their relation to hardware resources remain largely unknown.

PyXEL provides a suite of features to facilitate bitstream analysis, comprehension, and decoding of configuration bitstreams, as well as a database of specific devices. It can parse the bitstream to identify metadata, headers, and configuration commands. Based on this information, PyXEL can create an in-memory object of the configuration data that can be manipulated for fault emulation and visualization. PyXEL includes a visualization mechanism that converts the configuration data into a two-dimensional bitmap based on configuration frames, allowing for visual inspection of the bitstream and correlation between sections of the bitstream and parts of the programmable hardware fabric.

Additionally, PyXEL provides a database of devices and their associated bitstream information, which can be used to perform module-aware and resource-aware fault injection and localization. To provide this feature, PyXEL includes a comprehensive methodology and suite of software modules for automatically decoding configuration bitstreams. This consists of a TIle-Mapper (TIM) and a Tile Encoder-Decoder (TED). TIM uses an algorithm to map tiles to the corresponding section of the configuration memory. The required device architecture information is extracted through the previously introduced EDA integration mechanism. TED associates bits in the memory section identified by TIM with the basic hardware resources. To instrument TED, PyXEL generates problem-solution pairs consisting of the state of the hardware resources and associated bits, retrieved using TIM from a set of input designs. Tiles of the same type and their related bitmaps are correlated to link bits to the state of the resources, thus incrementally reducing the number of bits associated with a specific configuration of a particular resource within a tile, as depicted in Figure 3. This approach does not require particular designs to decode the resources in a tile, providing quick convergence due to many tiles of the same type using different resources usually available within an implemented design. If needed, further refinement of the results can be achieved with designs tailored to specific resources. This methodology has been applied to interconnection and logic tiles, allowing for the mapping and decoding of the vast majority of PIPs and logic within configurable logic and interconnection tiles. For instance, about the UltraScale+ device family, PyXEL was able to associate bits to used PIPs with excellent accuracy without any user intervention, reaching an accuracy greater than 98% in only a few minutes per design.

## III. Case of Study and Validation

PyXEL has been utilized for supporting numerous fault injection campaigns and radiation tests [13][14]. This section presents a comprehensive case study of a hardened-by-redundancy benchmark circuit, implemented on an XCZU3EG, consisting of three independent replicas of a CORDIC computational core implemented in the programmable hardware and controlled by software voter on the processor system, as depicted in Figure 4, is presented. The

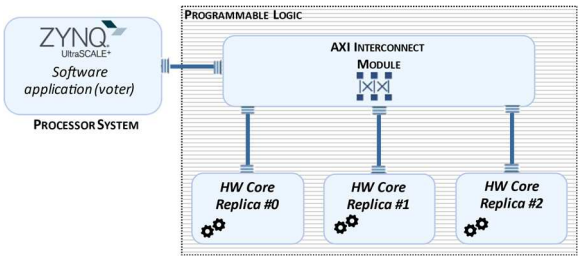


Fig. 4: Benchmark implemented on the ZU3EG

comprehensive analysis flow, illustrated in Figure 1, based on PyXEL, includes fault model extraction during radiation testing, an assessment of system and module robustness through fault injection campaigns, and design hardening via a custom isolation-based placement solution. First, PyXEL was employed to characterize the configuration memory of an UltraScale+ device during a proton test [13]. The device under test was located in the irradiation room and connected to a host computer running the PyXEL tool via a serial link. Through PyXEL, the configuration memory content was automatically evaluated through periodic readback operations, and fault models were extracted and used in a fault injection campaign to evaluate system robustness. Subsequently, PyXEL was utilized to assess the robustness of the benchmark design by conducting a fault injection campaign based on the emulation of fault models resulting from the radiation test experiment in the device's configuration memory. Each emulated fault, including single and multiple-bit upsets, was evaluated by extensively executing computations on the hardware cores. PyXEL was employed to manage all the steps of the experiment, such as fault location generation, fault emulation, device configuration, and results collection and categorization. Outcomes have been categorized into *Data Detectable Data Corruption* (DDC), *Data Unavailability* (DU), and *Silent Data Corruption* (SDC). The outcome is classified as DDC when a fault is a source of errors in the computations of a single core, while it is classified as DU if it prevents the system from completing the execution. Finally, SDC is the most critical effect, resulting in a corruption of the outputs that affected more than one module, leading to undetectable data corruption. After 10,000 fault injection experiments targeting used tiles, the DDC Rate was 5.47%, the DU rate was 1.39%, and the SDC rate was 1.12%.PyXEL fault mapping capability has been used to identify modules associated with faulty behavior. As a result, we found that some SDC occurrence was associated with multiple replicas failing together due to a single fault. This is a known phenomenon where resources, such as interconnections, shared between different TMR domains can lead to TMR failures. A technique for mitigating this phenomenon is based on the isolated placement of modules to prevent multi-domain failure [4]. PyXEL has been used for controlling the placement of the modules using the feature exposed in II.A. The isolated version presented an overhead of resources of about 2%. A new fault injection campaign reported a significant drop in errors rate, resulting in a DDC of 2.82%, a DU of 0.27%, and no SDC, as reported in Table I.

TABLE I. ROBUSTNESS EVALUATION RESULTS

| Circuit | Error Rates | | | |
|---|---|---|---|---|
| | DDC [%] | UD [%] | SDC [%] | Total |
| TMR | 5.47% | 1.39% | 1.12% | 7.98% |
| ISOLATED TMR | 2.82% | 0.27% | 0.00% | 3.09% |

## IV. CONCLUSIONS

We presented PyXEL, a tool for automating reliability analysis, bitstream decoding, and custom place-and-route. PyXEL provides methodologies and means for performing a comprehensive automatized evaluation of the robustness of designs implemented in programmable hardware.

## REFERENCES


[1] H. Quinn, "Radiation effects in reconfigurable FPGAs*", Semicond. Sci. Technol.*, vol. 32, no. 4, Apr. 2017.

[2] C. De Sio, S. Azimi and L. Sterpone, "FireNN: Neural Networks Reliability Evaluation on Hybrid Platforms," in *IEEE Transactions on Emerging Topics in Computing*, vol. 10, no. 2, pp. 549-563, 1 April-June 2022, doi: 10.1109/TETC.2022.3152668.

[3] L. Bozzoli, C. De Sio, L. Sterpone and C. Bernardeschi, "PyXEL: An Integrated Environment for the Analysis of Fault Effects in SRAM-Based FPGA Routing," *2018 International Symposium on Rapid System Prototyping (RSP)*, Turin, Italy, 2018, pp. 70-75, doi: 10.1109/RSP.2018.8632000.

[4] A. Portaluri, C. De Sio, S. Azimi and L. Sterpone, "A New Domains-based Isolation Design Flow for Reconfigurable SoCs," *2021 IEEE 27th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, Torino, Italy, 2021, pp. 1-7, doi: 10.1109/IOLTS52814.2021.9486687.

[5] S. A. Guccione, D. Levi, P. Sundararajan, "JBits: A Java-based interface for reconfigurable computing", *Second Annual Military and Aerospace Applications of Programmable Devices and Technologies Conference (MAPLD)*, Laurel, MD, September 1999.

[6] T. Haroldsen, B. Nelson, and B. Hutchings, "RapidSmith 2: A Framework for BEL-level CAD Exploration on Xilinx FPGAs", *2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '15)*. Association for Computing Machinery, New York, NY, USA, 66–69. https://doi.org/10.1145/2684746.2689085.

[7] K. Dang Pham, E. Horta and D. Koch, "BITMAN: A tool and API for FPGA bitstream manipulations," *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2017, Lausanne, Switzerland, 2017, pp. 894-897, doi: 10.23919/DATE.2017.7927114.

[8] C. Lavin and A. Kaviani, "RapidWright: Enabling Custom Crafted Implementations for FPGAs," *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Boulder, CO, USA, 2018, pp. 133-140, doi: 10.1109/FCCM.2018.00030.

[9] T. Zhang, J. Wang, S. Guo and Z. Chen, "A Comprehensive FPGA Reverse Engineering Tool-Chain: From Bitstream to RTL Code," in *IEEE Access*, vol. 7, pp. 38379-38389, 2019, doi: 10.1109/ACCESS.2019.2901949

[10] F. Benz, A. Seffrin and S. A. Huss, "Bil: A tool-chain for bitstream reverse-engineering," *22nd International Conference on Field Programmable Logic and Applications (FPL)*, Oslo, Norway, 2012, pp. 735-738, doi: 10.1109/FPL.2012.6339165.

[11] Z. Ding, Qiang Wu, Yizhong Zhang, Linjie Zhu, Deriving an NCD file from an FPGA bitstream: Methodology, architecture and evaluation, *Microprocessors and Microsystems*, Volume 37, Issue 3, 2013, pp 299-312, ISSN 0141-9331, https://doi.org/10.1016/j.micpro.2012.12.003.

[12] Y. Junghwan, et al.. 2018. "A Bitstream Reverse Engineering Tool for FPGA Hardware Trojan Detection", *2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18)*. Association for Computing Machinery, New York, NY, USA, 2318–2320. https://doi.org/10.1145/3243734.3278487

[13] S. Azimi, et al. , A comparative radiation analysis of reconfigurable memory technologies: FinFET versus bulk CMOS, *Microelectronics Reliability*, Volume 138, 2022, 114733, ISSN 0026-2714, https://doi.org/10.1016/j.microrel.2022.11473

[14] C. De Sio, S. Azimi, L. Sterpone, On the analysis of radiation-induced failures in the AXI interconnect module, Microelectronics Reliability, Volume 114, 2020, 113733, ISSN 0026-2714, https://doi.org/10.1016/j.microrel.2020.113733.