UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

MARCOS TOMAZZOLI LEIPNITZ

A Fault Injection Platform for FPGA-based Communication Systems

Monograph presented in partial fulfillment of the requirements for the degree of Bachelor in Computer Engineering.

Advisor: Prof. Dr. Gabriel Luca Nazar

Porto Alegre
2015

ABSTRACT

The resilience of communication systems to soft errors is a major concern in many scenarios, such as when facing stringent dependability constraints or when operating in radiation-harsh environments. Field-Programmable Gate Arrays (FPGAs) are successful platforms for the implementation of communications systems, since they provide the advantages of reconfigurability coupled with a high throughput processing of data streams and lower development costs. When used in radiation-harsh environments, however, FPGAs present a distinct set of challenges that demands specialized evaluation. The large configuration memories of SRAM-based devices are especially susceptible to radiation-induced faults, demanding the evaluation of design's resilience to such phenomena. Therefore, this work proposes a fault injection platform that targets specifically FPGA-based communication systems, in order to simulate and evaluate the effects of configuration faults on the functionality of such systems, as well as their impact on the communication quality metrics. The platform operates partially on the device and partially on a host computer, aiming at both flexibility and high performance. A Reed-Solomon decoder is used as a case study to validate the platform, showing its ability to measure metrics that are relevant for communication systems.

Keywords: Field-Programmable Gate Array. Soft errors. Fault injection. Reliability evaluation. Communication systems.

Uma Plataforma de Injeção de Falhas para Sistemas de Comunicação Baseados em FPGA

RESUMO

A resiliência de sistemas de comunicação com relação à *soft errors* é uma grande preocupação em muitos cenários, como quando se exige alta dependabilidade ou quando operam em ambientes com alta incidência de radiação. *Field-Programmable Gate Arrays* (FPGAs) são plataformas de sucesso para a implementação de sistemas de comunicação, uma vez que proporcionam as vantagens da reconfigurabilidade associada a uma alta taxa de processamento de fluxos de dados e reduzidos custos de desenvolvimento. Quando usados em ambientes com alta incidência de radiação, no entanto, FPGAs apresentam um conjunto distinto de desafios que exigem avaliação especializada. As memórias de configuração de dispositivos baseados em SRAM são especialmente suscetíveis a falhas induzidas por radiação, exigindo a avaliação da resiliência do projeto a tais fenômenos. Assim, este trabalho propõe uma plataforma de injeção de falhas que tem como alvo específico os sistemas de comunicação baseados em FPGA, a fim de simular e avaliar os efeitos de falhas de configuração sobre a funcionalidade desses sistemas, bem como o seu impacto sobre as métricas de qualidade de comunicação. A plataforma opera parcialmente no dispositivo e parcialmente em um computador hospedeiro, visando compatibilizar flexibilidade e alto desempenho. Um decodificador Reed-Solomon é utilizado como estudo de caso para validar a plataforma, mostrando a sua capacidade para medir métricas que são relevantes para sistemas de comunicação.

Palavras-chave: *Field-Programmable Gate Array*. *Soft errors*. Injeção de falhas. Avaliação da confiabilidade. Sistemas de comunicação.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

ASIC	Application Specific Integrated Circuit
AUT	Area Under Test
BER	Bit Error Rate
CLB	Configurable Logic Block
CMOS	Complementary Metal Oxide Semiconductor
COTS	Commercial Of-The-Shelf
CRC	Cyclic Redundancy Check
CUT	Circuit Under Test
DMA	Direct Memory Access
DSP	Digital Signal Processor
EDAC	Error Detection and Correction
FER	Frame Error Rate
FPGA	Field-Programmable Gate Array
HDL	Hardware Description Language
IC	Integrated Circuit
ICAP	Internal Configuration Access Port
IOB	Input/Output Block
JTAG	Joint Test Action Group
LET	Linear Energy Transfer
LFSR	Linear Feedback Shift Register
LUT	Look-Up Table
MBU	Multi-Bit Upset
NMR	N-Modular Redundancy
PRNG	Pseudo Random Number Generator
SBU	Single-Bit Upset
SEE	Single-Event Upset
SEL	Single-Event Latch-up
SER	Soft Error Rate
SET	Single-Event Transient
SEU	Single-Event Upset
SRAM	Static Random Access Memory
TMR	Triple Modular Redundancy

SUMMARY

1 INTRODUCTION

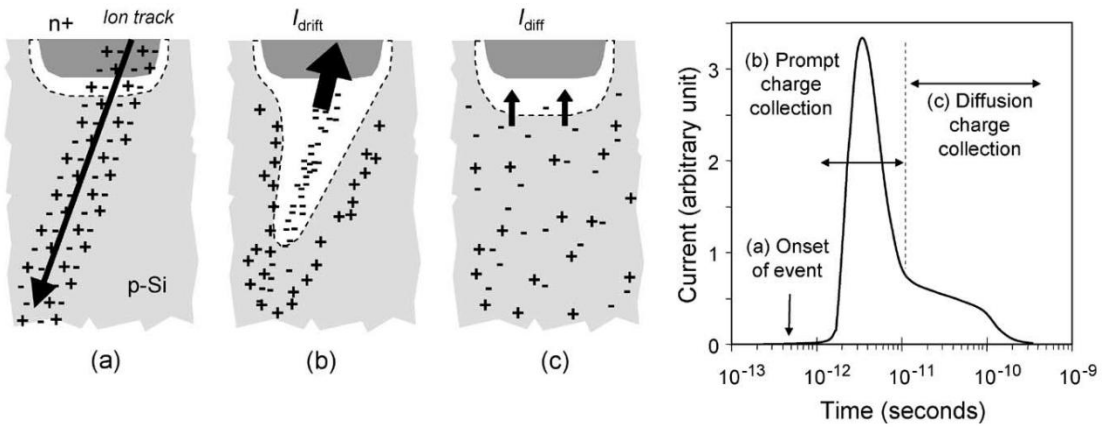
The continuous evolution of the fabrication technology of Complementary Metal Oxide Semiconductor (CMOS) devices have enabled the production of more complex and dense integrated circuits (ICs), allowing the integration of entire systems in a single chip. Moreover, the decreasing of transistor's feature sizes to nanoscale and the aggressive voltage operation reduction increased the performance and reduced the power consumption of modern computing systems (ITRS, 2010). The technology scaling, however, increased the system's sensitivity to transient faults triggered by environment disturbances such as electromagnetic interference or radiation, resulting in higher soft error rates (SERs) (Dixit & Wood, 2011). A soft error is a non-permanent error that occurs when a fault causes an incorrect change in system states. The propagation of errors through the system circuitry can lead to failures, i.e., a deviation of the system intended function. Although the advance in manufacturing technologies has greatly reduced the occurrence of soft errors caused by, for example, temperature fluctuation or noisy power supply, the incidence of ionizing particles in radiation-harsh environments remains a major concern, even for terrestrial applications (Dodd et al., 2010). Therefore, when designing highly dependable systems, designers should be aware about the selected device technology and the environment conditions.

In aerospace applications, such as communication satellites and avionics, failures occur frequently due to the increased flow of charged particles from the cosmic radiation and solar winds, such as protons, neutrons, alpha particles and heavy ions. Ground-based applications also can be affected, mainly by high-energy neutrons generated by the interaction of the cosmic radiation with the upper atmosphere. The interaction of charged particles with sensitive regions of a semiconductor device can cause enough of a charge disturbance to affect its operation transiently or permanently. When a high-energy particle strikes the silicon substrate, it liberates its energy via the production of free electron-hole pairs, resulting in an ionized track that leads to a current pulse disturbance, as illustrated in Figure 1.1. The amount of charge collected and the consequent impact on the device operation depends on the linear energy transfer (LET) of the ions, which depends on the mass and energy of the particle and the material in which it is traveling (Bauman, 2005).

Single-Event Effect (SEE) is the common term used for any measurable or observable effect on a device, component or system, due to a single charged particle strike (JEDEC, 2006). The most relevant SEEs are the Single-Event Latch-up (SEL), the Single-Event Transient (SET), and the Single-Event Upset (SEU). SEL is a potentially destructive

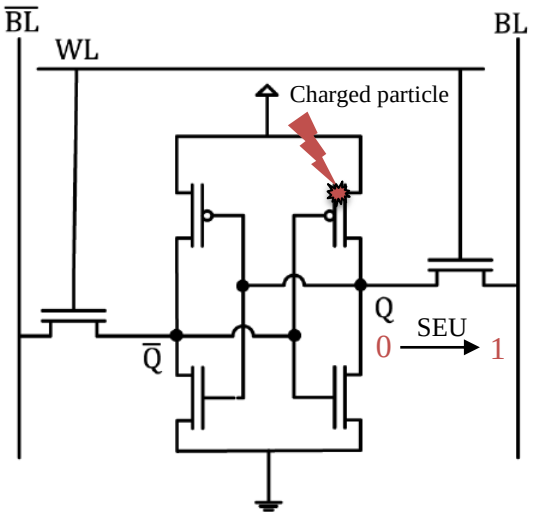
condition caused by the creation of a parasite structure in CMOS devices, resulting in a low-impedance path between power and ground. If the device is not destructed by the overcurrent generated, the malfunction can be solved by a power cycle. SET is a transient voltage pulse that can propagate through the system's circuitry. The main consequence of SETs are SEUs, which are characterized by the change of state (bit flip) of a memory cell, register, latch or flip-flop, leading to functional failures or to loss of information (Monteanu & Autran, 2008). Figure 1.2 illustrates a SEU within a typical SRAM memory cell due to a charged particle strike. Unlike SELs, SETs and SEUs do not cause any permanent damage in the device.

Figure 1.1 – Effect of an ionizing particle striking a reverse-biased p-n junction.



Source: Baumann (2005).

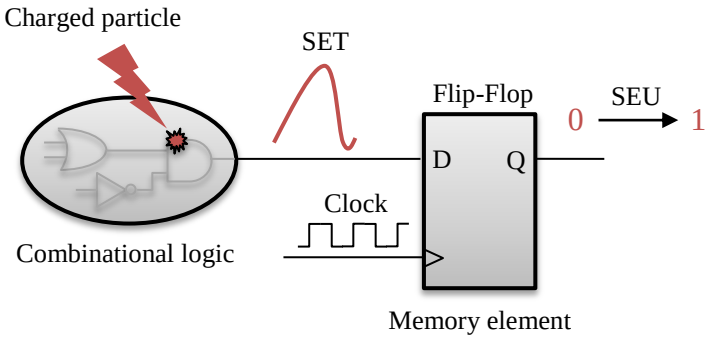
Figure 1.2 – A SEU within a typical SRAM memory cell due to a charged particle strike.



Source: the author.

In digital systems, sequential logic elements, such as flip-flops and latches, are used to hold data and control signals between combinational logic blocks that perform logic operations. When a charged particle strikes a combinational element and enough charge is collected, a SET can be generated. If the transient signal is not attenuated or logically masked through the combinational circuitry, it can be latched into a sequential element depending on the sampling clock and the set-up and hold times. The result is a SEU, as exemplified in Figure 1.3. Although the amount of charge collected in newer CMOS technologies decreased, due to smaller junctions as transistors shrink, the trend is higher SERs. The decreasing operation voltage and propagation delay, as well as the escalating operation frequency, decreased the probability of fault masking due to electrical or timing characteristics. Therefore, increased the probability of a SET to propagate through a combinational circuitry and result in a SEU.

Figure 1.3 – A SET generated in a combinational circuit can lead to a SEU.



Source: the author.

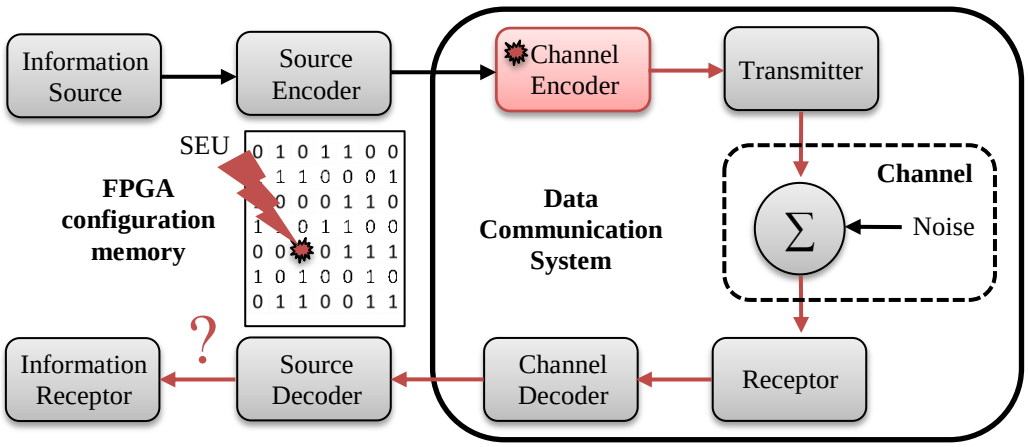
The use of radiation hardened solutions, either by technology or by design, can solve the radiation effects problem for high reliable applications or critical systems operating in radiation-harsh environments. Nevertheless, Commercial Of-The-Shelf (COTS) devices become widely used, due to much lower costs, higher performance, and the use of state-of-art technologies. This trend, however, introduced the challenge of building reliable systems with non-reliable devices, leading to the need of improving the implemented system dependability with the inclusion of fault-tolerance mechanisms, such as hardware or information redundancy. Moreover, the most efficient solution depends on the device technology and the system's design, which demands a carefully evaluation.

1.1 Motivation

Field-Programmable Gate Arrays (FPGAs) are reconfigurable devices widely used in the implementation of computing systems. These devices offer many advantages over the use of Application Specific Integrated Circuits (ASICs), such as reconfigurability coupled with a high throughput processing of data streams and lower development time and costs. Those characteristics make FPGAs a great option for data communication applications, due to the possibility of in-field reprogramming to correct faulty behaviors or to add new features, without the performance and power penalties introduced by the use of general-purpose processors (Hauck & Dehon, 2008). When used in radiation-harsh environments, however, FPGAs present a distinct set of challenges that demands specialized evaluation. The technology scaling and the growing number of configuration cells in SRAM-based FPGAs increased the configuration memory sensitivity to radiation induced SEUs. (Lesea et al., 2005; Quinn et al., 2015).

The reliability of data communication systems depends on their capacity to deal with the conditions of noise and interference from the physical channel that can corrupt the data being transmitted. In order to improve this capacity, channel coding/decoding techniques are used to provide error detection and correction, through the addition of information redundancy (Shannon, 1948). However, the channel encoder/decoder hardware is susceptible to faults, which may affect its correct operation and impact the system reliability, as illustrated in Figure 1.4. Therefore, the resilience of FPGA-based communication systems must be carefully evaluated by studying the impact of configuration faults in communication quality metrics, so as to develop effective and low cost fault-tolerance mechanisms.

Figure 1.4 – Configuration memory upset in an FPGA-based communication system.



Source: the author.

1.2 Goals

The goal of this work is to provide a fault injection platform flexible and optimized for FPGA-based communication systems, allowing emulate configuration faults on SRAM-based FPGAs. To achieve high flexibility, the platform should operates partially on an FPGA device and partially on a host computer, to easily generate various channel models and/or to measure different metrics that may be relevant for any particular communication system. To achieve high performance, the communication between the FPGA device and host computer should be made through a high speed interface. Moreover, the fault injection/removal needs to be fast as possible to not become the performance bottleneck of fault injection campaigns.

The purpose of the fault injection platform is evaluate the impact of configuration faults on the communication quality metrics of communication systems, such as the bit error rate (BER) and the frame error rate (FER), as well as validate fault-tolerance mechanisms to improve the system resilience to such faults. This evaluation demands a considerable amount of input vectors for each fault injected, since the statistical confidence of those metrics depends on the amount of data transmitted. Therefore, this process can be very time consuming, making the performance an important requirement for the proposed platform.

2 BACKGROUND

This section presents a discussion about the radiation effects on FPGAs, more specifically configuration memory faults, the fault injection methods used to verify the impact of such faults, and fault injection platforms that inspired this work.

2.1 Radiation Effects on FPGAs

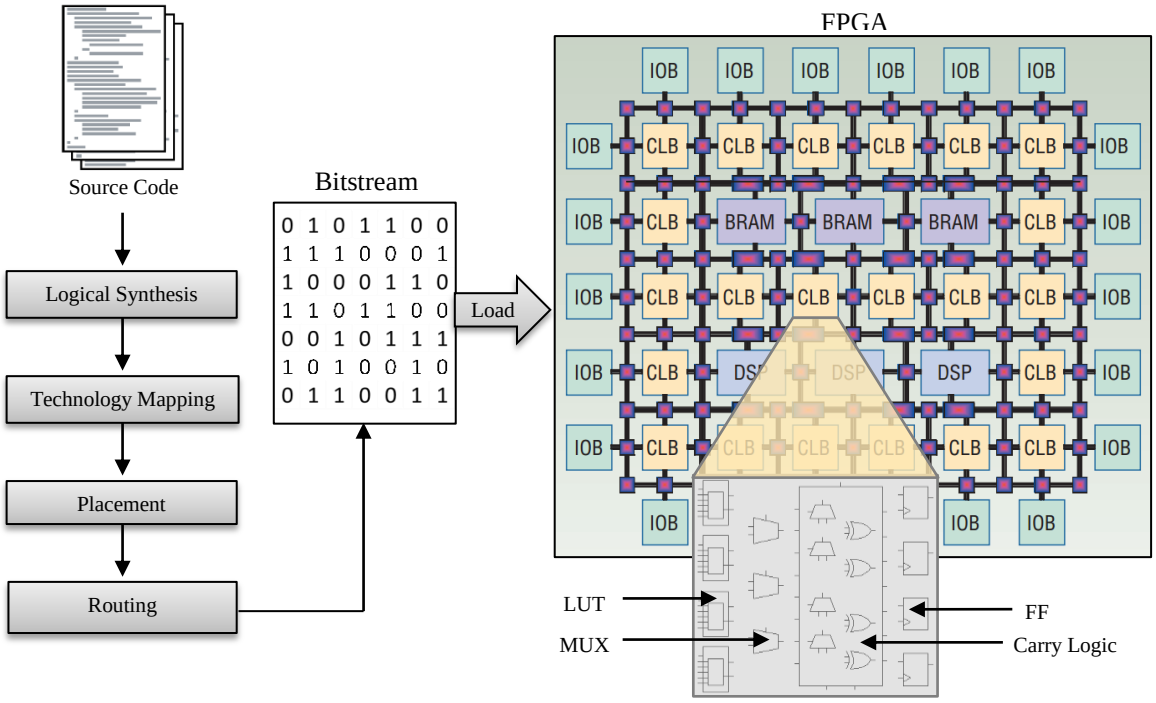
FPGAs are complex reconfigurable devices that comprise a wide variety of heterogeneous resources. The basic structure of modern FPGAs includes interconnect resources, clock-management resources, configurable logic blocks (CLBs), input/output blocks (IOBs), and embedded blocks such as digital signal processors (DSPs), general-purpose processors, high-speed IOBs, and memories. CLBs are used to perform simple combinational and sequential logic. These blocks are typically made of look-up tables (LUTs), multiplexers, flip-flops, and carry logic. Programmable interconnect resources, such as routing switches, allow interconnecting CLBs, IOBs and embedded blocks to implement complex systems (Buell et al., 2007).

The logic and routing resources in an FPGA are controlled by the bits of a configuration memory, which may be based on antifuse, flash, or SRAM technology. The design flow of FPGA-based systems involves the creation of a bitstream to load into the device, as illustrated in Figure 2.1. Typically, the process starts with the system design written in a hardware description language (HDL), such as VHDL or Verilog. Next, the design is optimized and mapped into the FPGA's available resources through logical synthesis, technology mapping, placement, and routing. Finally, the bitstream is generated and the device can be programmed. FPGA vendors such as Xilinx, Altera, Lattice Semiconductor, and Atmel provide tools to perform the design flow (Gokhale & Graham, 2005).

Like any other semiconductor device, FPGAs are susceptible to radiation effects. Mostly, these effects depend on the technology used to store the configuration data. Regarding the impact of SEEs on reliability and functionality, FPGAs based on SRAM technology are a special class of devices. The major concern for SRAM-based FPGAs is SEUs within the configuration memory. In such devices, this memory may represent more than 80 percent of the total memory bits, increasing the probability of configuration faults. Upset configuration bits may change the logic and routing of the implemented system, as illustrated in Figure 2.2, leading to functional failures in an unpredictable way. In contrast, the

primary concern for antifuse and flash-based FPGAs is SETs and SEUs within user flip-flops and block memories. Although the configuration memory cells of antifuse and flash-based FPGAs present a relative immunity to SEEs, these devices has lower logic capacity and cannot be reprogrammed an unlimited number of times, making SRAM-based FPGAs more suitable for complex systems requiring frequent reconfiguration and adaptation (Violante, 2004; Wirthlin, 2015).

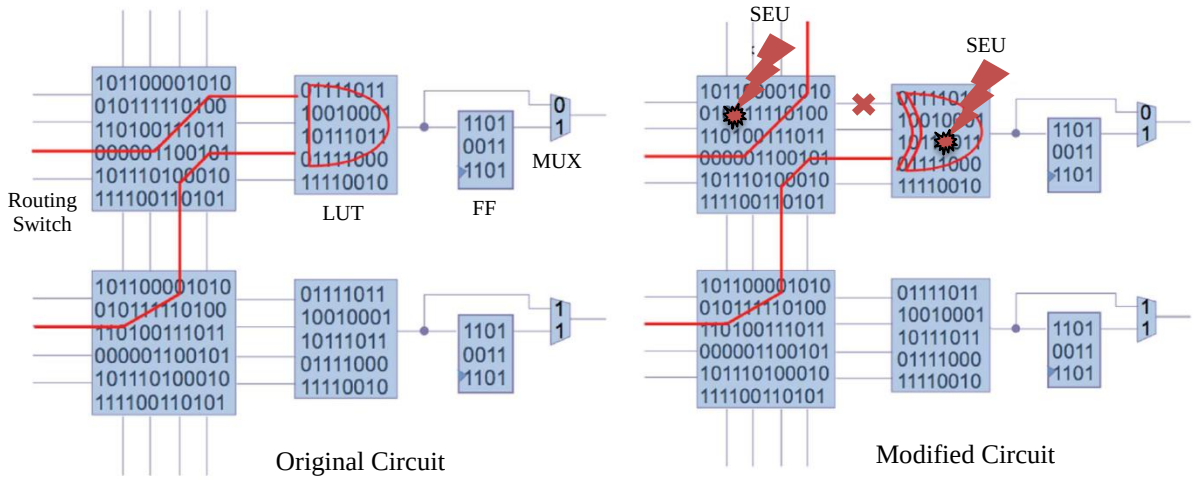
Figure 2.1 – Typical FPGA structure and design flow.



Source: adapted from Buell et al. (2007) and Hauck & Dehon (2008).

The vulnerability of SRAM-based FPGAs to configuration data SEUs demands the use of techniques to mitigate their impact on the system dependability. The correction techniques that act directly on the configuration memory are based on scrubbing and information redundancy. Scrubbing is the simplest technique, consisting in constant reconfiguration at a higher rate than the predicted SEU rate. A more effective solution is repairing the configuration memory by performing a regular read-back followed by Cyclic Redundancy Check (CRC) calculations to detect the corrupted frames. Then, partial reconfiguration can be used to update only these frames. This technique, however, requires more complex circuitry to calculate and compare the CRC values, as well as additional memory to store them. The scrubbing technique, in contrast, only requires a controller to timing the reconfiguration process (Charmichael et al., 2000).

Figure 2.2 – Upset FPGA configuration bits may change the logic and routing.



Source: adapted from Wirthlin (2015).

Regarding mitigation techniques that act on the system design, which covers not just configuration faults, but also SETs and SEUs in sequential logic and embedded blocks, the most common solutions are Triple Modular Redundancy (TMR) and Error Detection and Correction (EDAC). TMR is a hardware redundancy technique that consists in implementing the circuit three times, and their outputs compared by a voter to decide the correct one. Despite the amount of extra logic resources needed to implement this technique, it is very effective to prevent errors until more than one module is affected. This method has limitations, as demonstrated by Quinn et al. (2007), when the design is affected by Multi-Bit Upsets (MBUs), which requires a smart floorplan usage to isolates the TMR modules. The EDAC technique is based on information redundancy, i.e., additional bits are introduced in the data word through complex data encoding techniques. Depending on the encoding scheme, one or more erroneous bits can be detected and corrected, impacting the circuitry complexity and the amount of resources needed.

The best SEEs mitigation solution depends on the chosen FPGA device and the system design, but typically combines the techniques aforementioned. More details can be found in Lima & Reis (2006).

2.2 Fault Injection Methods

Fault injection can be defined as the deliberate introduction of faults within a desired system, aiming at evaluating its dependability and validating fault-tolerance mechanisms.

This method is widely used, for example, to determine the space-readiness of spacecraft applications during the development process, by analyzing the system's behavior in the presence of faults and estimating its failure rate. Then, fault-tolerance mechanisms are introduced and the process is repeated until the desired robustness in not achieved. The fault injection in the hardware of a give system can be performed with three main techniques: physical fault injection, simulation-based fault injection, and emulation-based fault injection (Clark & Pradhan, 1995; Ziade et al., 2004).

Physical fault injection is suited when a prototype of the system is already available. It is accomplished by disturbing the hardware with environment parameters, such as radiation, or corrupting the signal value of IC leads, circuit board connectors, and system back plane. Heavy ion beans are commonly used, for example, to perform accelerated radiation testing to verify the system sensitiveness to SEEs. Although signal corruption on IC pins can model many faults that occur inside the device, effects of several internal faults cannot be investigated with this method, since the access to internal nodes is limited.

Simulation-based fault injection, contrasting with physical fault injection, is suited when a prototype of the system is not yet available. It is performed by using a high-level model of the system to reproducing at software level the errors that would have been produced by faults occurring in the hardware. The errors are reproduced altering logical values before or during the simulation runtime. This method is very effective in allowing early and detailed analysis, but the simulation of complex systems in software can be very time consuming. Moreover, complex faulty behaviors in simulation models might be difficult to replicate.

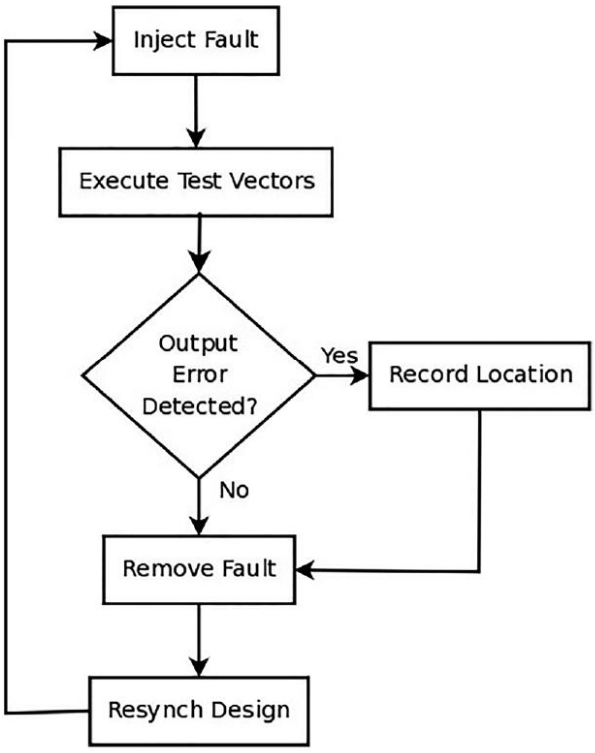
Emulation-based fault injection is an intermediate solution based on reconfigurable devices, since modern FPGAs allow rapid prototyping of complex systems. In software-based methods, the fault injection is performed through modifications on the HDL source code or synthesis byproducts (e.g., netlist or configuration files), or using circuit instrumentation. In hardware-based methods, the fault injection is performed in hardware and acts directly on the device circuitry, by changing the data content of memories or the logical state of user flip-flops, for example. Hardware-based methods are more popular because they consume much less time, which is important for fault injection campaigns aiming high fault coverage.

After choosing the fault injection technique, designers need an effective methodology to stimulate the implemented system and to verify its correctness for each fault injected. According to Arlat et al. (1990), a fault injection methodology for a target system can be fully characterized by the fault model adopted, the activation patterns used to stimulate the design

under test, the readouts values collected during the experiment, and the derived measures. Regarding emulation-based fault injection in FPGA-based systems, which is the focus of this work, the most common fault model adopted are single-bit or multi-bit SEUs within the configuration memory, due to the sensitivity of SRAM-based devices to such faults. Although emulation-based techniques do not provide a measurement of the SEU bit cross sections or a broadly sample from all the failure modes, it offers important advantages compared to actual ground radiation testing, such as faster fault injection times, more controllability, and lower costs. Moreover, appropriate configuration fault injection has been validated compared to ground radiation (Lima et al., 2001). Therefore, the emulation of SEEs is a complementary approach to radiation testing, which allows the exploration of alternatives to improve the resilience of computing systems in the early project stages.

As defined in Quinn & Wirthlin (2015), the basic features that a fault emulation system for SRAM-based FPGAs needs are the ability to access the memory where the fault will be injected, the ability to stimulate and execute the circuit under test, the ability to determine output errors, and the ability to clear errors. Figure 2.3 illustrates a basic algorithm for the fault emulation process.

Figure 2.3 – Basic fault emulation algorithm.



Source: Quinn & Wirthlin (2015).

Configuration memory faults in FPGAs are emulated by flipping the configuration bits. FPGA manufacturers offer many options to access the configuration memory. The access can be performed by a software application or tool through the serial interface provided by the Joint Test Action Group (JTAG) boundary scan port, or through an external parallel port, such as the SelectMAP available in FPGAs from Xilinx. Since the Virtex-II, Xilinx devices also provide the Internal Configuration Access Port, which allows accessing the configuration memory from within the device. The performance of the configuration method chosen impacts the feasibility of high fault model coverage, especially for complex systems with millions of configuration bits. Hence, the use of a dedicated hardware to interact with the ICAP port becomes the most common fault injection method, since it allows faster fault injection/removal times.

After each fault injected, the circuit under test must be stimulated with test vectors to verify the effects of each injected fault in its outputs. Stimulating a given system with the entire possible test vectors may be not feasible, which depends on the wide of its input. Therefore, in the most cases, a subset is chosen. For input-sensitive faults, the subset size and the sampling method affect the fault coverage and the statistic confidence of the results. Although exists different stimulus methods to exercise all areas of the circuit, the most common method uses quality pseudo-random number generators (PRNGs) to generate a representative set of input vectors, which can be made by a software application running in a host computer or by a dedicated hardware inside the FPGA. Moreover, if runtime generation decreases the fault injection system performance, the input vectors can be generated off-line and accessed from a memory.

To evaluate the effects of each injected fault, the circuit under test output is compared with the output of a fault-free version of the same system (golden system). Depending on the system complexity and the performance expected, the golden system and the outputs comparison can be made in hardware or simulated in software. Like the test vectors, the golden outputs also can be generated off-line and stored in a memory.

2.3 Related Works

Several works used the fault injection methodology to verify the robustness of fault mitigation solutions for SRAM-based FPGAs, as demonstrated in Sterpone & Violante (2005), wherein the TMR architecture robustness is analyzed. In general, the most efficient techniques combine prevention methods, such as TMR, with correction methods, such as

scrubbing. This combination is evaluated, for example, in Lima et al. (2001), wherein the scrubbing is used to eliminate the accumulation of faults that may deteriorate the efficiency of the TMR technique.

To evaluate SEU mitigation schemes, the FLIPPER fault injection platform presented in Alderighi et al. (2007) uses two FPGAs boards. The circuit under test (CUT) is implemented in one board, while another board performs the fault injection and applies the input vectors generated by a simulation software, which also generates the golden outputs. A software application in a host computer allows configuring how the tests should be performed. Regarding the performance, this platform allows injecting each fault in 50μs.

The fault injection platform proposed in Sterphone & Violante (2007) uses a single FPGA board by using the ICAP port available in devices from Xilinx, which allows faster fault injection times. This platform uses a hardwired Power PC microprocessor to perform the fault injection through the ICAP. The results are received by a host computer through a serial cable (RS-232), which ultimately impact the system performance. It offers an average fault injection time of 4.6ms for the entire execution and fault classification, although performing a bit flip on the configuration memory requires around 10μs, which is one of the main advantages of using a high speed interface for fault injection. The main drawback, however, is the use of a hardwired Power PC processor that is not available in any FPGA, especially in the newer ones, which limits its use.

The fault injection platform presented in Nazar & Carro (2012) also uses a single FPGA and the ICAP to emulate SEUs within the configuration memory, but it is designed with the most common elements that comprises a FPGA (LUTs, flip-flops, multiplexers and memory blocks.). This approach enables its implementation in any FPGA that has the ICAP. Regarding the performance, this platform offers fault injection and fault removal times less than 10μs. The performance will be defined, in general, by the CUT performance on the stimulating process, or by the platform communication with the host computer, used for receive the results obtained from the CUT for each injected fault. Since the communication is made through a serial cable (RS-232), it is likely that in most cases this will be the performance bottleneck of this platform.

The fault injection methodology presented in Di Carlo et al. (2014) uses partial reconfiguration together with the Xilinx Essential Bit technology. This technology allows identifying and extracting the essential bits of the bitstream configuration, i.e., those bits which are essential for the system functionality. This method greatly reduces the time needed for a fault injection campaign, because many bits of the configuration memory that do not

have relation with the functionality of the system (which are the majority) are ignored. In addition, this platform does not require knowledge about the addressing of the memory configuration frames, because it isolates the system to be evaluated in a previously defined reconfigurable partition through the Xilinx tools. This approach, however, requires the reconfiguration of the entire partition for each fault injected. Therefore, although this platform uses a high speed interface for reconfiguration (ICAP), the performance advantage is obtained only for exhaustive fault injection campaigns.

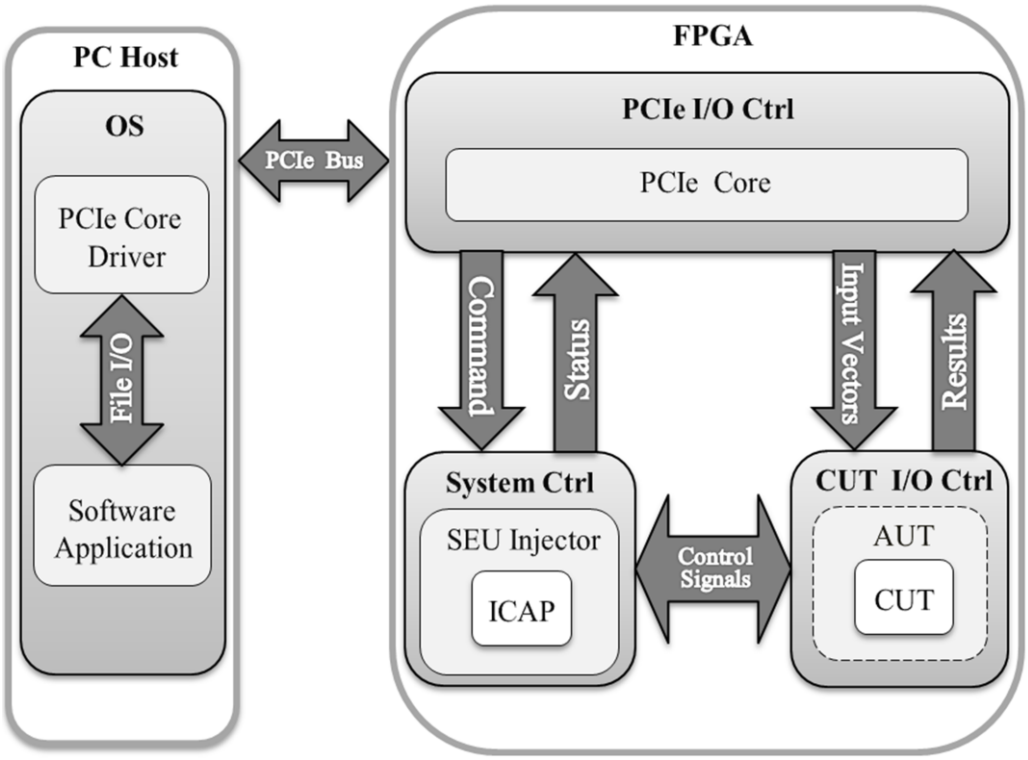
The work presented in Tarrillo et al. (2014) evaluated the neutron cross-section of N-modular redundancy (NMR) techniques for masking the effects of SEUs in FPGA-based systems. The fault injection is performed through an 8-bit PicoBlaze controller that controls the ICAP port of the Xilinx Virtex-5 FPGA. The controller takes the fault injection locations from an external memory. These locations are previously defined by neutron radiation experiments. As a special feature, it allows evaluate the effects of faults accumulation (multiple-independent upsets), with the system injecting faults until a functional error manifests.

A platform for rapid prototyping of complex wireless communication systems (MIMO-BICM) is presented in Gimmler-Dummont et al. (2012). The proposed platform is composed by a software application running on a host computer and an accelerator on a FPGA board. For this type of application, a high bandwidth between the software application and the board is required. So, the connection from the host simulation chain to the FPGA is done by an Ethernet link. The network connection on the FPGA is handled by the Xilinx MicroBlaze soft core microcontroller. This platform, however, does not target hardware fault injection.

3 PROPOSED FAULT INJECTION PLATFORM

The fault injection platform proposed in this work emulates SEUs, more specifically single-bit upsets (SBUs) within the configuration memory of SRAM-based FPGAs. Each fault injected remains active during the entire CUT execution time. Despite the restricted fault model, the platform can be easily modified to allow the emulation of MBUs and to inject/remove faults during the CUT execution. The platform structure consists of a hardware component on an FPGA device and a software application running on a host computer. The data communication between them is made through the PCIe bus. An Ethernet link was used in a previous version of the platform, but it was discontinued due to a lower performance. Figure 3.1 shows the platform topology and the communication between its main components. In the following sections, each component will be described with more details.

Figure 3.1 – Overall proposed fault injection platform.



Source: the author.

3.1 PCIe I/O Controller

The PCIe I/O controller is responsible for the communication between the hardware on the FPGA and the software application that runs on the host computer. The communication

solution adopted in this work was the Xillybus system developed by Eli Billauer (2014), which is a flexible and easy to implement PCIe solution for Xilinx or Altera FPGAs, composed of an IP core and a host driver. This system is designed to never induce excessive latency, attempting to deliver data as soon as possible, in either direction. It provides several end-to-end stream pipes for application data transport, offering a maximal data rate (full-duplex) between 200MB/s and 800MB/s, depending on the number of PCIe lanes available in the FPGA board and the host computer.

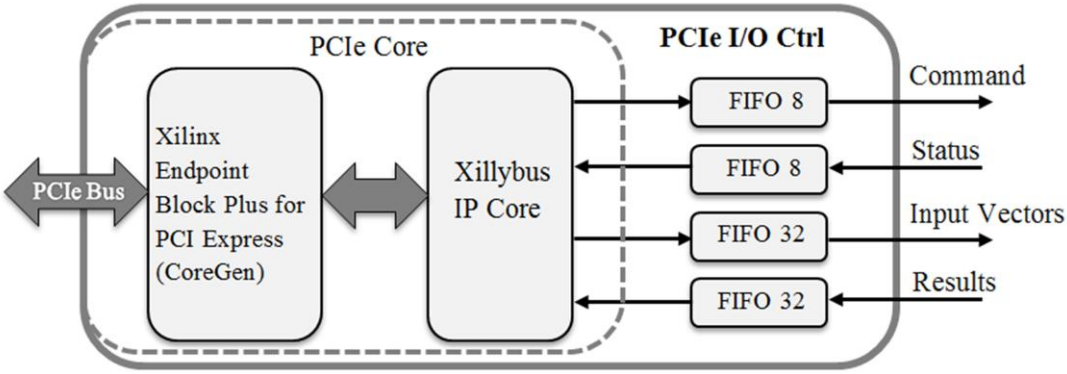
The communication between the Xillybus core and the user logic is made through standard FIFO memories, by checking the FIFOs empty and full signals in a round-robin manner. The data transfers initiates when the FIFO is ready for it, depending on the stream's direction. The interface between the Xillybus core and the PCIe bus, as well as the FIFO memories, can be generated with the Xilinx Core Generator or the Altera Mega Wizard. The host computer driver generates device files which behave like named pipes between TCP/IP streams. Through I/O operations on these files, the software application can communicate with the FPGA with one or more streams, synchronously or asynchronously. Also at driver load, Direct Memory Access (DMA) buffers are allocated in the host’s memory, and their address informed to the FPGA. A handshake mechanism between the FPGA and the host driver passes data using these buffers to make up the application preemption periods with I/O buffering, creating the illusion of a continuous data stream.

An online tool (the IP Core Factory) is available on the system website for immediate configuration and download of custom-tailored Xillybus IP cores per specification, such as the number of streams, their direction, their data width, their name, their expected bandwidth, and other parameters. The driver binary supports any Xillybus IP core configuration, i. e., the streams and their attributes are auto-detected by the driver as it's loaded into the host's operating system, and device files are created accordingly. In the FPGA to host direction, if a stream is configured as asynchronous, the Xillybus IP core will fills the host’s DMA buffers whenever possible, i.e., when the associated device file in the host is open, data is available in the FIFO and there is free space in the DMA buffers. If a stream is configured as synchronous, however, the Xillybus core will not fetch data from the FIFO unless the software application request that data from the file descriptor. The host to FPGA direction will be discussed with more details in the software application section.

For the proposed platform, the Xillybus core was configured with a synchronous 8-bit stream and an asynchronous 32-bit stream. The 8-bit stream allows the application software to send commands to the system controller and also for this controller to return the current state

of the system. A 32-bit stream allows the host computer to send the input vectors to the CUT I/O controller and also for this controller to return the results generated by the CUT. Figure 3.2 shows the PCIe I/O controller structure for a Xilinx FPGA board, using the IP Cores available in the Core Generator tool (Xilinx UG341, 2011; Xilinx UG175, 2012).

Figure 3.2 – Communication interface with the host computer.

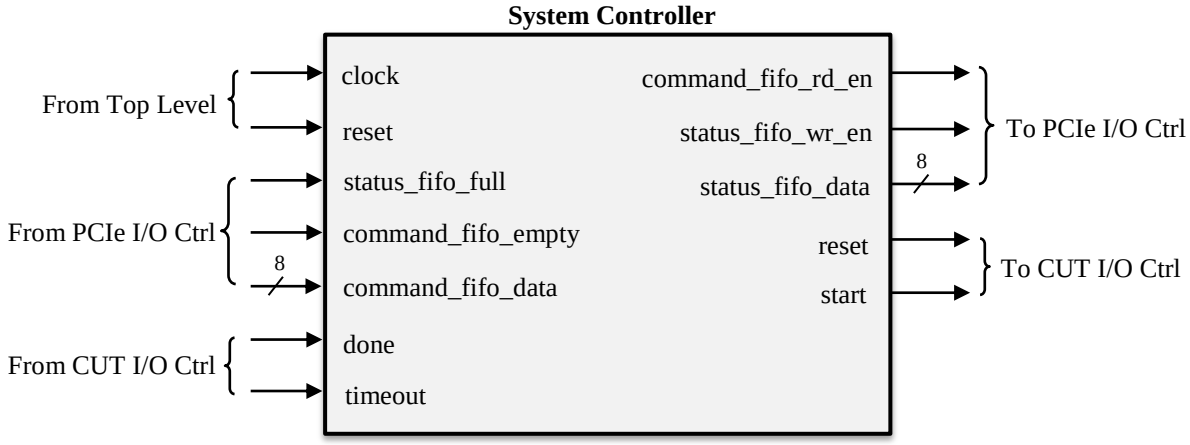


Source: the author.

3.2 System Controller

The System controller is responsible for starting a fault injection campaign by receiving a *start* command from the software application, as well as for requesting the fault injection/removal to the fault injector (SEU Injector) and controlling the CUT I/O controller through the signals *start*, *done*, *timeout* and *reset*. It also sends the fault injection campaign status to the software application. Figure 3.3 shows the system controller interface.

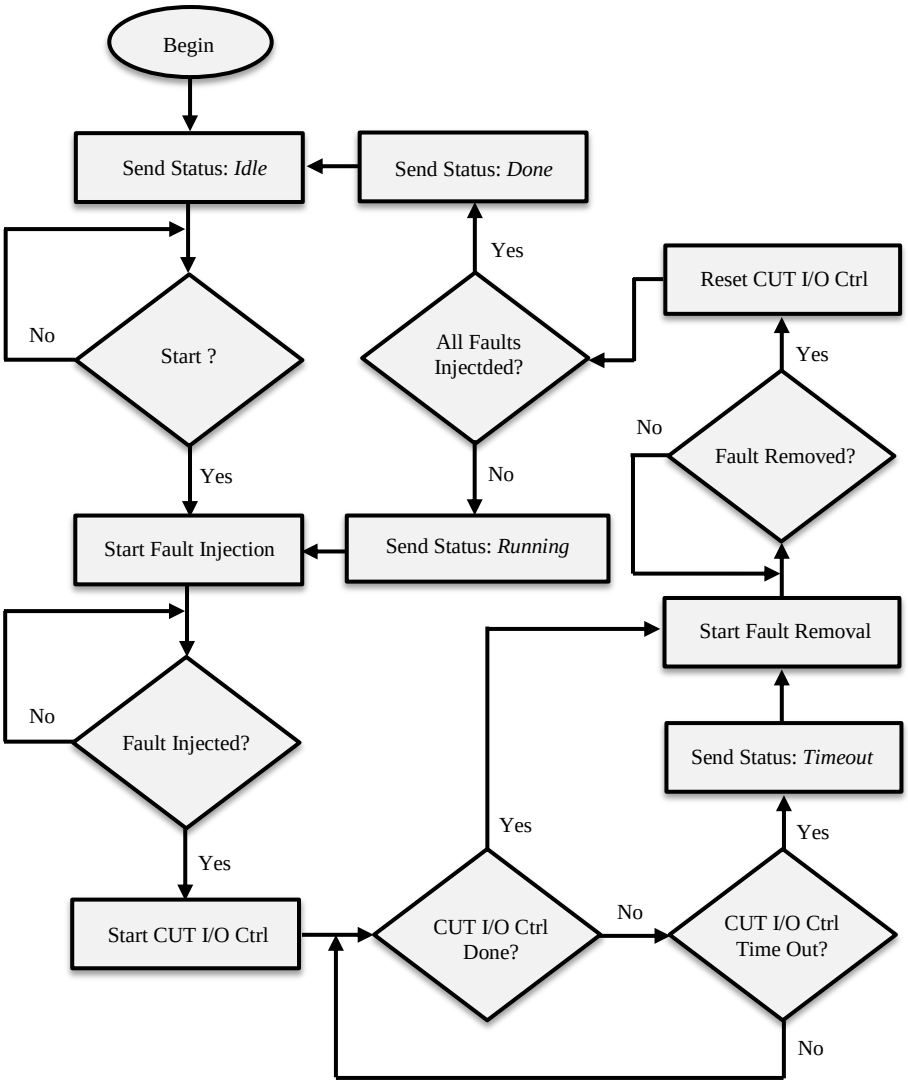
Figure 3.3 – System Controller interface.



Source: the author.

The system controller algorithm to perform a fault injection campaign is illustrated in Figure 3.4. After receiving the *start* command from the software application, the system controller requests a fault injection and remains in wait state until the SEU injector returns the fault injection confirmation. Next, it starts the CUT I/O controller and remains in wait state until the CUT I/O controller report that the execution already finished or a time out condition occurs. Then, it requests the fault removal to restore the correct state of the configuration memory, followed by a reset in the CUT I/O controller. This cycle is repeated until the total amount of faults to be injected is not reached, which is signaled by the SEU injector.

Figure 3.4 – System Controller operation flowchart.



Source: the author.

During the fault injection campaign process, the system controller sends some important information to the software application. The system status can be *idle* (the system is

ready to start a fault injection campaign), *done* (the fault injection campaign already finished), *running* (the fault injection campaign continues) or *timeout* (the CUT data flow stopped). The time out condition is especially important, since the software application needs to be aware that faults affecting clock, reset or control signals may crash the CUT and stop its data flow with the CUT I/O controller. The treatment for this situation will be discussed with more details in the next sections.

The SEU injector is adapted from Nazar & Carro (2012). The Xilinx's Virtex-5 FPGA used in this work has the ICAP, which allows manipulating the configuration memory bits. In this device, the configuration memory is divided into frames whose bits can be accessed through a known addressing scheme (Xilinx UG191, 2012). The FPGA structure is divided into rows, numbered from 0 (up to 9) in the top and bottom halves, starting from the center. Each row is divided into the same number of columns, where a column corresponds to a basic block in the array (CLB, DSP, block RAM, IOB, etc.). Each column is configured by a certain number of frames, which depends on the block type. A frame can be thought as a vertical stack of 1312 bits (41 words of 32 bits) that spans the whole height of a row. Figure 3.5 illustrates the structure of a row in the Virtex-5 FPGA and its configuration frames.

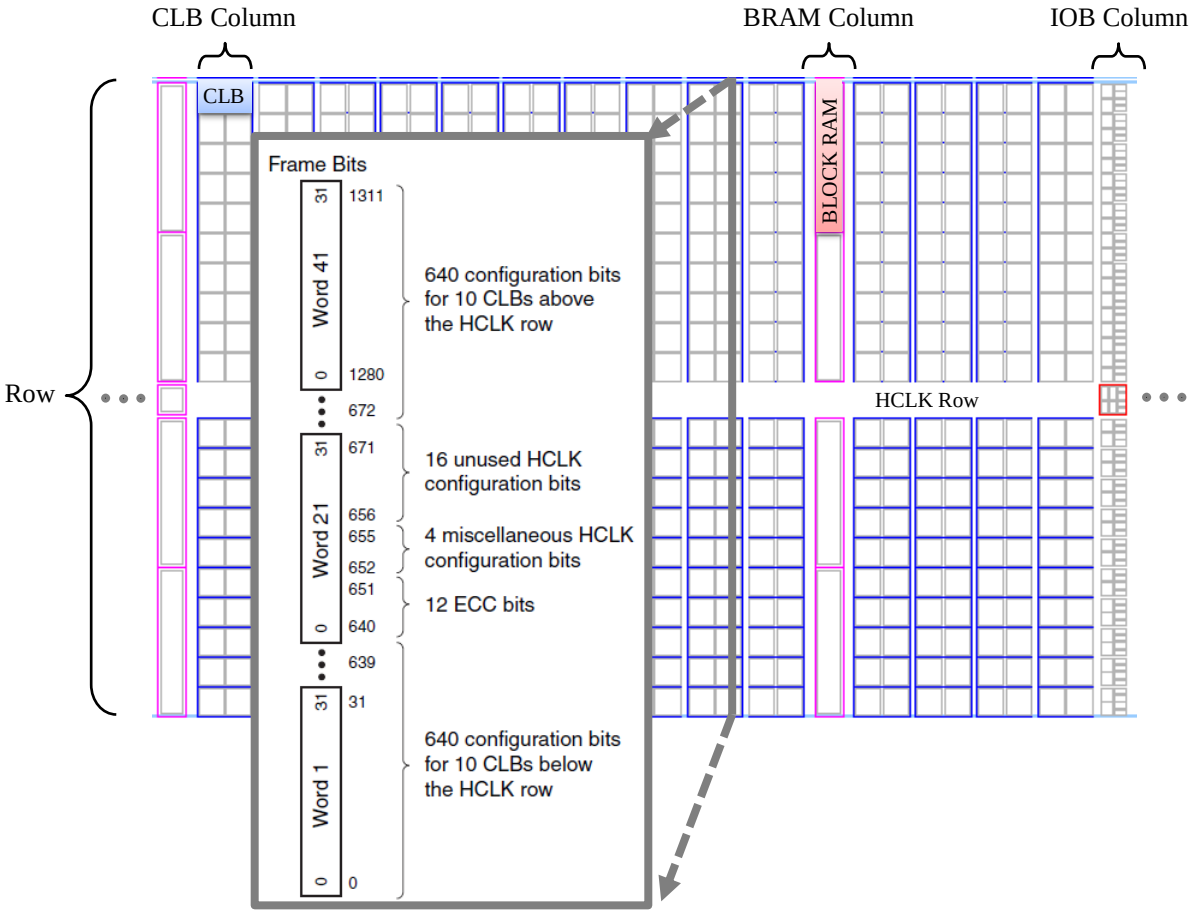
Each configuration frame has a unique 32-bit address divided into five segments, as illustrated in Figure 3.6. The block type segment divides the configuration frames depending on their function and how they are accessed. In this work, all the configuration frames accessed are for interconnect and block configuration (CLB, DSP, IOB, block RAM parameters, etc.), which means the block type segment is constant. The top/bottom segment defines if the row address is related to the top half or to the bottom half of the FPGA. The major address segment defines which column of the selected row will be accessed. Finally, the minor address segment defines which frame will be accessed in the selected column. Table 3.1 shows the number of interconnect and block configuration frames per column type.

Table 3.1 – Virtex-5 interconnect and block configuration frames per column type.

Block	Number of Frames
CLB	36
DSP	28
Block RAM	30
IOB	54
Clock Column	4

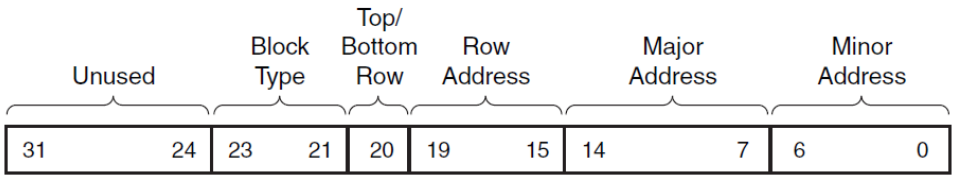
Source: Virtex-5 FPGA Configuration User Guide (2012).

Figure 3.5 – Virtex-5 FPGA row structure and its configuration frames.



Source: adapted from Virtex-5 FPGA Configuration User Guide (2012).

Figure 3.6 – Segmentation of the configuration frame address.

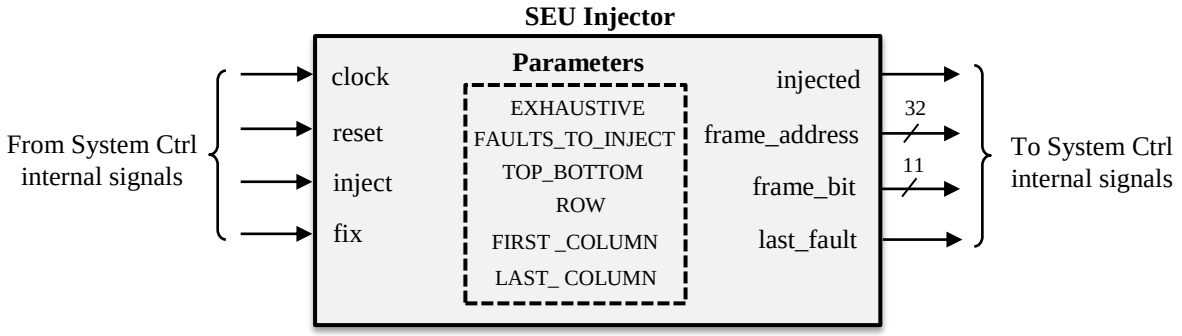


Source: Virtex-5 FPGA Configuration User Guide (2012).

The configuration frames address scheme allows a controlled access to each configuration bit related to any region of the device. With this possibility, an Area Under Test (AUT) is defined, where the CUT should be positioned with placement constraints for the implementation tool. Through partial reconfiguration with the ICAP, the SEU injector can flip only the bits responsible for the configuration of the AUT, which prevents the fault injection in other components of the system that are not related with the CUT. Through the SEU

Injector parameters, the user can define the FPGA area where the faults will be injected and which configuration bits will be flipped (random or exhaustive). Depending on the method chosen, the frame address and the frame bit will be generated through a Linear Feedback Shift Register (LFSR), operating as a PRNG, or through a counter, always respecting the AUT limits and the frame address formation. Figure 3.7 shows the SEU injector interface.

Figure 3.7 – SEU Injector interface.



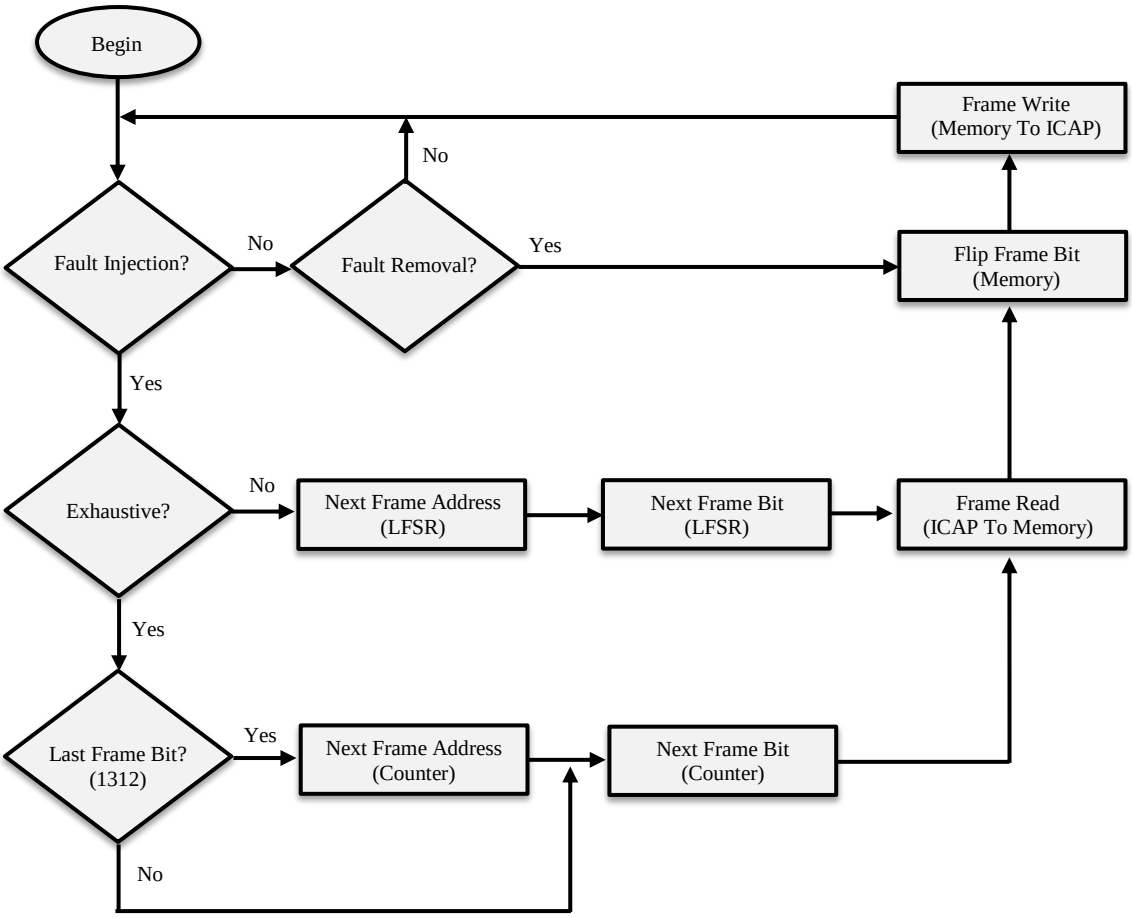
Source: the author.

When a fault injection is requested, the SEU injector generates the frame address and the bit to flip according the user parameters. Next, a read operation is performed with the ICAP using the frame address, and the incoming words are saved in a memory (frame memory). Then, the frame bit is flipped, followed by a write operation with the ICAP. Finally, the *injected* flag is raised. If the bit flipped is the last one (based on the parameters), the *last_fault* flag is raised too. When the system controller requests the fault removal, the same frame bit in the frame memory is flipped again to restore the original value, followed by a write operation with the ICAP using the frame address generated in the last fault injection request. Then, the *injected* flag is lowered. Figure 3.8 shows the basic SEU injector algorithm.

It is important to note that each frame has 16 unused configuration bits which are skipped when the frame bit is chosen, leaving 1296 bits to flip per frame. Moreover, as described in Nazar & Carro (2012), flipping some very specific bit positions may lead to multiple bit errors in other frames. In the Virtex-5 device, this situation happens with some LUTs that can work as shift registers. If the configuration bit that determines this behavior is flipped, multiple bit errors will occur on the configuration bits associated with that LUT, since those bits will be shifted, as shown in Figure 3.9. Therefore, when a critical bit is identified, not only the frame of the flipped bit will be read, but also the following five frames (worst

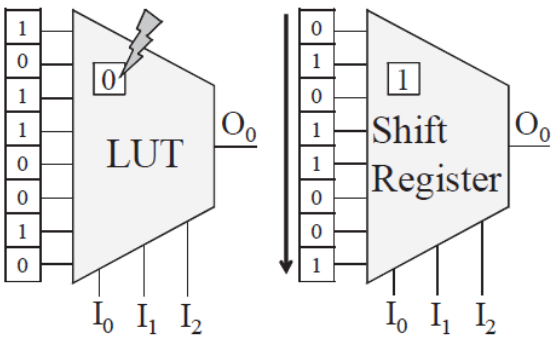
case for the Virtex-5). When the fault removal is requested, all those frames will be written back to ensure that the next iterations of the SEU injector operation will not be corrupted.

Figure 3.8 – SEU Injector operation flowchart.



Source: the author.

Figure 3.9 – A single bit flip may transform a LUT into a shift register.



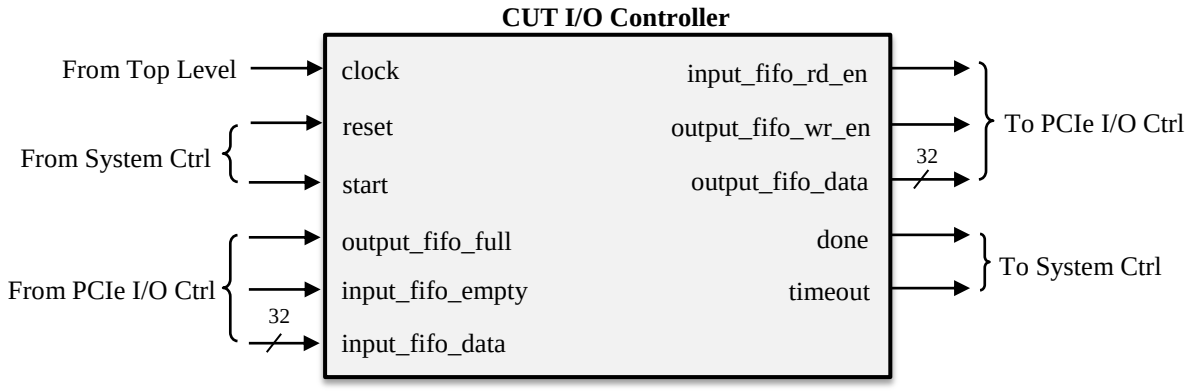
Source: Nazar & Carro (2012).

3.3 CUT I/O Controller

The CUT I/O controller is responsible for starting the CUT execution after the system controller request. It consumes the input vectors received from the software and makes them available for the CUT, as well as sends the CUT output to the software application. The logic needed to accomplish this task depends of the communication system that is being evaluated, as it converts the raw stream from the PCIe I/O controller (32 bits) to the specific width expected by the CUT input, as well as converts the specific width of the CUT output to the width expected by the PCIe I/O controller. The user of the platform can also determine the information that should be transmitted to the software, through the inclusion of additional hardware in this module. Although the control logic depends on the CUT, the following behavior must be taken as a model for proper system operation.

After the CUT execution with all the received stimuli vectors and after sending all the results, the CUT I/O controller needs to inform the system controller that the CUT execution ended normally, by raising the flag *done*. If this task does not end after a certain time (defined by the user), however, is very likely that a clock, reset, or control signal of the CUT was affected by the configuration fault, interrupting the stimuli process. In that case, the CUT I/O controller and the software application need a way to handle the problem related with the amount of data that the application expects to send and receive, since it performs blocking I/O operations. The CUT I/O controller handles this problem consuming all the stimuli vectors received and completing with dummy data the amount of results that the CUT output should have generated. After this process, the flag *timeout* is raised to inform the system controller that a time out occurred. Figure 3.10 shows the CUT I/O controller interface.

Figure 3.10 – CUT I/O Controller interface.



Source: the author.

3.4 Software Application

The software application communicates with the hardware in the FPGA through the PCIe interface in order to perform some basic functions:

- Control the hardware component through sending commands (*start*), and receiving the system state (*idle*, *running*, *done*, and *timeout*).
- Generate test vectors and send them for the CUT.
- Receive the results from the CUT and calculate any relevant metric for data communication systems that allows evaluate the impact of the injected faults on the system reliability.

The software application defines a thread for each data stream (device files), to enhance the throughput of the host computer. Since the communication is based upon I/O system calls with a certain overhead, several data are grouped to be sent or received with a single system call. Figure 3.11 shows the proposed strategy.

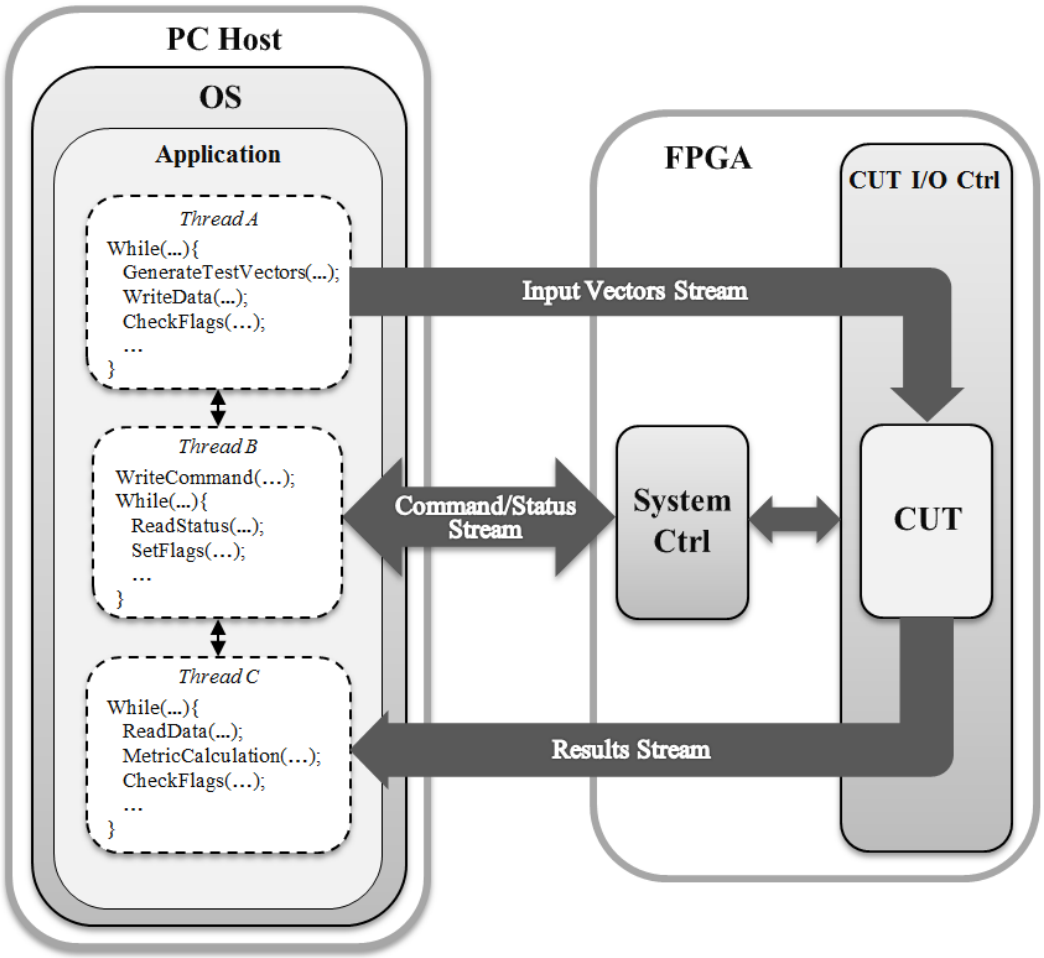
To establish a communication with a continuous data flow, asynchronous streams are used to send the test vectors and to receive the results. A call to the function writing test vectors to the stream will return immediately if the data can be stored entirely in the DMA buffers. Then, the data can be transmitted to the FPGA at the rate requested by the CUT I/O controller, with no involvement of the software application. A call to the function reading the results from the stream will return immediately if the DMA buffers have the amount of data requested.

Synchronous streams are used to send commands and receive the system's status. A call to the function writing a command to the stream will not return until the data has arrived to the command FIFO in the PCIe I/O controller. A call to the function reading the system status from the stream will not return until the PCIe I/O controller fetch data from the status FIFO in response to the request. The thread responsible for receiving the system's status after each fault injected communicates with the other threads through flags.

After writing the amount of test vectors defined for each fault injected and reading the amount of CUT outputs expected, each thread checks the system status flags (*running* and *done*) to proceed to the next iteration or to end the execution. Moreover, the thread that receives the results needs to verify if the data received in the last iteration is related to a normal CUT operation or to a time out condition, by checking another flag (*timeout).* If a time out occurred, the results received are dummy data generated by the CUT I/O controller.

Therefore, any metric calculated during the previous iteration should be discarded and a time out failure registered.

Figure 3.11 – Software application operation.



Source: the author.

The threads are created using the OpenMP API, which supports multi-platform shared-memory parallel programming in C/C++. The three threads are defined using the *sections* directive, which is a non-iterative work-sharing construct that allows the execution of code sections by a team of threads. The communication (flags) between the threads (sections) is performed through shared variables. The synchronization is made using the *flush* directive, which allows a write of a variable on one thread to be guaranteed visible and valid on another thread. This operation is necessary because the OpenMP threads can maintain a temporary view of shared memory which is not consistent with that of other threads. The *flush* directive identifies a synchronization point at which the implementation must provide a consistent view of memory. Thread-visible variables are written back to memory at this point.

4 EXPERIMENTAL RESULTS

In this section, we present the results obtained with the proposed fault injection platform. The hardware module was described in VHDL and synthesized for the Xilinx's Virtex-5 FPGA, with the evaluation platform XUPV5-LX110T (Xilinx UG347, 2011). The software application was developed in C++ (305 lines of code), and the communication with the FPGA is made through the PCIe 1.0 x1 interface provided by the development platform. In the following subsections, we describe the resources occupied by the hardware module, the platform performance and its validation with some experiments done on a Reed-Solomon decoder, which is a family of channel codes suitable for space applications (CCSDS, 2011). The platform and the Reed-Solomon decoder are available in https://goo.gl/iNHdHG.

4.1 Resource Occupation and Performance

Table 4.1 shows the amount of resources used by the hardware components of the platform, as well as the proportional use of the available resources in the device. Note that the CUT I/O controller is specific for each different CUT to be used by the platform. The results for this component include only the resources needed to interact with the CUT used as a case study in this work, more specifically the first implementation, which will be described in the following section. Besides the CUT I/O controller, the PCIe I/O controller is the component that consumes the most part of the resources required, due to the FIFO memories used for the I/O interface and the IP Cores that compose this component. The low device occupation by the system controller and the PCIe I/O controller is an important result, since it saves resources for the CUT.

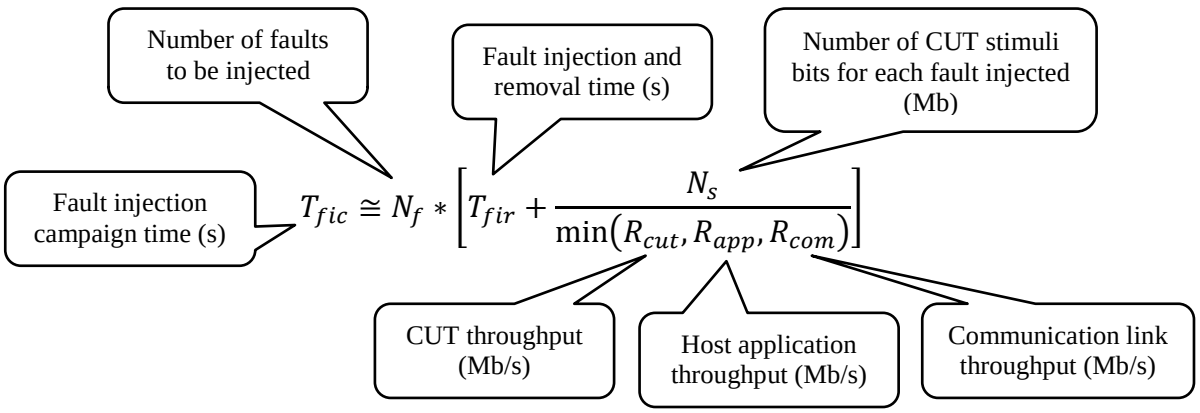
Table 4.1 – Platform resource usage and device occupation (Virtex-5 XC5VLX110T).

Platform Component	**Required amount**			**Device occupation**		
	LUT	***FF***	***BRAM***	***LUT***	***FF***	***BRAM***
PCIe I/O Ctrl	2509	3386	10	3.63%	4.9%	6.75%
System Ctrl	514	164	1	0.74%	0.24%	0.68%
CUT I/O Ctrl	417	266	1	0.60%	0.39%	0.68%
Total	3440	3816	12	4.97%	5.53%	8.11%

Source: the author.

In this work, the system controller, that includes the SEU injector and the ICAP interface, runs at 100MHz. This operating speed allows a strict fault injection and correction time under 10μs, as described in Nazar & Carro (2012). That means, for example, that a strict fault injection and correction campaign that covers the entire FPGA device used in this work, which has around 25 million configuration bits, can be done in approximately 250 seconds. Therefore, the time that the platform needs to flip a configuration bit and correcting it is not a concern. The main concern regarding the performance is about the data throughput between the host computer and the FPGA. A complex communication system may have millions of configuration bits, making an exhaustive fault injection campaign impracticable if the data throughput is not high enough. The Xillybus system for the PCIe 1.0 x1 interface operates with a bus clock of 62.5MHz and a data processing path of 32 bits (PCIe's natural word length), which allows a theoretic data transfer rate of 250MB/s in both directions. In our experiments, the data throughput achieved without any data processing overhead, was around 200MB/s. In this scenario, the communication link will not be the performance bottleneck until the data processing capabilities of the software application and the CUT exceed 200MB/s. The CUT throughput depends on its operating speed and design, while the software application throughput depends on the host computer capability, as well as how the stimuli vectors are generated and which metric will be calculated using the CUT outputs received. Figure 4.1 shows how the fault injection campaign time can be estimated.

Figure 4.1 – Fault injection campaign time estimation.



Source: the author.

4.2 Platform Validation: Reed-Solomon Decoder

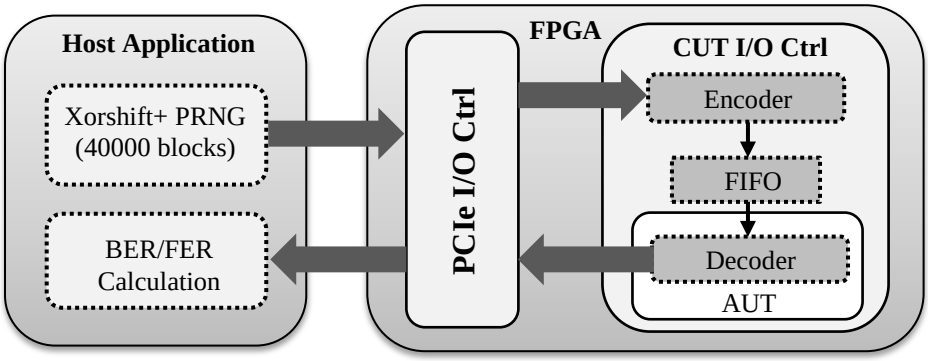
In order to validate the platform proposed in this work, we have performed some experiments with a Reed-Solomon decoder. Reed-Solomon codes are cyclic linear block codes that are computed over m-bit words. An RS(N, K) code uses blocks of N words, each with m bits, K of which are data words. The remaining N-K words are redundant, and allow correcting up to (N-K)/2 wrong words. A detailed description of Reed-Solomon codes and examples can be found at Wicker & Bhargava (1994). As is typical for most channel codes, the Reed-Solomon decoder is substantially more complex than the encoder. Therefore, this work focuses on studying the decoder's reliability.

For the experiments, we used two different implementations of a same Reed-Solomon decoder, more precisely an RS(255, 239) code with 8-bit words, in order to verify the design influence on the system reliability. For both implementations, the placement of the decoder in the AUT was made separating its main modules, which allows evaluate how each module impacts the system reliability when their configuration bits are flipped. Initially, the encoder was intended to be simulated in software, but due to a poor performance that becomes the bottleneck of the systems performance (throughput around 5MB/s), it was moved to the FPGA, more specifically inside the CUT I/O controller, but kept outside of the AUT.

The software application was programmed to generate 40000 blocks (approximately 10 MB) for each fault injected through a Xorshift+ PRNG (Vigna, 2014), and sends them to the encoder. Next, the encoder sends the codified words to the decoder, which sends its output to the software application through the PCIe I/O controller. Then, the software verifies if the words received are the same words that were transmitted and calculates the BER and FER of the system for the defined amount of input vectors. These metrics are commonly used to verify the communication quality of communication systems. The synchronization between the encoder output and the decoder input was made through a FIFO memory. Figure 4.2 illustrates the basic structure for the experiments.

In this work, an exhaustive fault injection campaign was performed, i.e., all the configuration bits of the AUT were flipped. The evaluation of the effects of configuration bit flips was made for two cases: without the insertion of errors between the encoder and the decoder (i.e., a perfect channel), and with the insertion of errors inside of the code capacity (i.e., a noisy channel). The number of incorrect words considered for the second case was defined by a PRNG.

Figure 4.2 – Reed-Solomon decoder experiment for each fault injected.

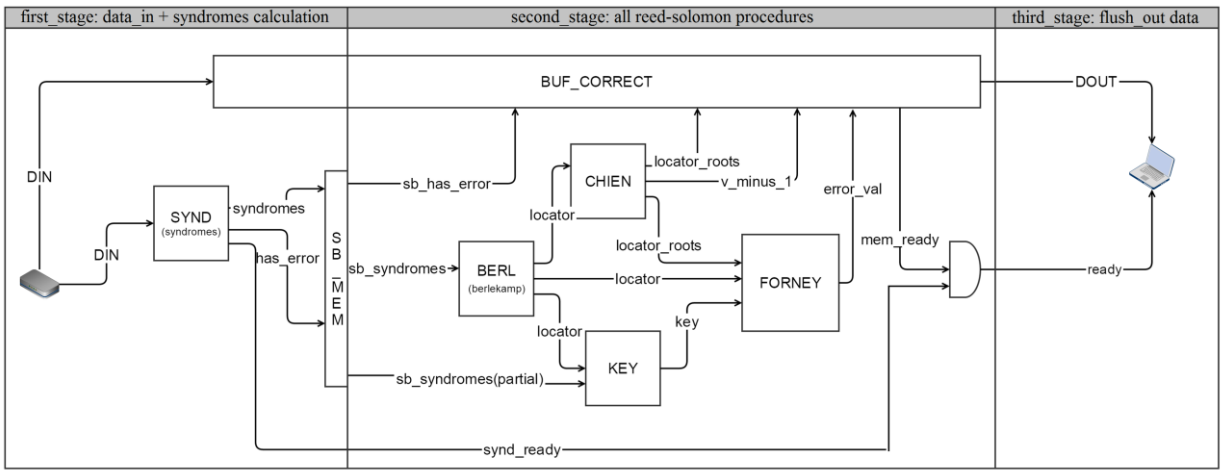


Source: the author.

4.2.1 First Implementation

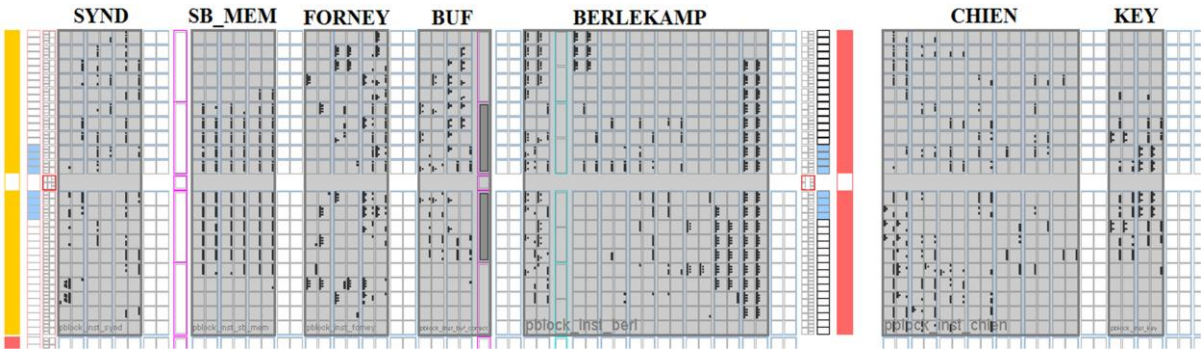
The first Reed-Solomon decoder implementation evaluated is based on the traditional decoder architecture described in Clark (2002), divided into three pipeline stages. It consists in a Reed-Solomon encoder/decoder pair that is configurable in terms of N, K and m, within the limits specified for these codes ($N \leq 2^m - 1$). Assuming an RS(N, K) code, we first receive the data, K symbols, followed by the N-K redundant words, and start the syndromes calculation. After that, all the remaining decoder stages are processed, including error correction. Then, the corrected symbols are flushed out. One symbol is received and/or flushed out per clock cycle. Figure 4.3 shows the three pipeline stages and their main modules, while Figure 4.4 shows their placement in the AUT.

Figure 4.3 – Reed-Solomon decoder structure (first implementation).



Source: the author.

Figure 4.4 – Placement of the Reed-Solomon decoder modules (first implementation).



Source: the author.

Regarding the amount of basic resources needed, the decoder uses 760 registers (flip-flops), 2535 LUTs, and one block RAM. Moreover, this decoder implementation allows an operating clock speed of up to 125MHz, which means we have a data throughput of up to 125MB/s. The PCIe bus allows a data throughput of 200MB/s, as mentioned in the previous section. For this implementation, the AUT consist of 1897600 configuration bits. With these numbers, each fault injection campaign took approximately two days, with the CUT performance being the performance bottleneck. Table 4.2 and Table 4.3 show the BER results for a perfect channel and for a noisy channel, respectively.

With a perfect channel condition, as expected, the majority of the configuration bits do not affect the system operation. Regarding the critical bits, a small part introduces a BER around 0.5, meaning that the output has become random noise, decoupled from the input. Another part causes a time out condition by affecting the clock, the reset, and control signals. The majority of the critical bits introduce a BER that varies with the component affected and the data blocks received. A change on a LUT configuration or the loss of a signal connection might explain this behavior. The *Buf_Correct* was the most critical component in the perfect channel scenario. The *Key* was the only component not affected, as it only operates when faults are identified in the received vector.

With a noisy channel, the number of critical bits increases. Moreover, almost all of those new critical bits introduce a variable BER. As can be seen, the *Berlekamp* was the most affected component, becoming the most critical. An interesting result is related with the *Key* module. In this implementation, this module is the only one to not have a signal to control the sequence of operations through the system modules. Therefore, it was expected that a failure in this module would not cause a time out.

Table 4.2 – Reed-Solomon decoder evaluation (first implementation/perfect channel).

Reed-Solomon Decoder Component	**Number of Configuration Bits**	**Perfect Channel**				
		BER = 0 (no effect)	**BER > 0 or Time Out (configuration failures)**			
		% of total	***% of total***	***% of failures***		
				BER ≈ 0.5 (noise)	***Time Out***	***Other***
Synd	139968	99,1	0,9	2.07	33.23	64.7
Sb_Mem	139968	99.97	0.03	5.26	21.05	73.69
Forney	139968	99.44	0.56	3.58	43.28	53.14
Buf_Correct	132192	96.59	3.41	19.92	9.92	70.16
Berlekamp	409536	99.73	0.27	21.19	43.12	35.69
Chien	326592	99.61	0.39	2.43	51.57	46
Key	93312	100	0	0	0	0
Area Under Test	1897600	99.52	0.48	13.48	26.87	59.65

Source: the author.

Table 4.3 – Reed-Solomon decoder evaluation (first implementation/noisy channel).

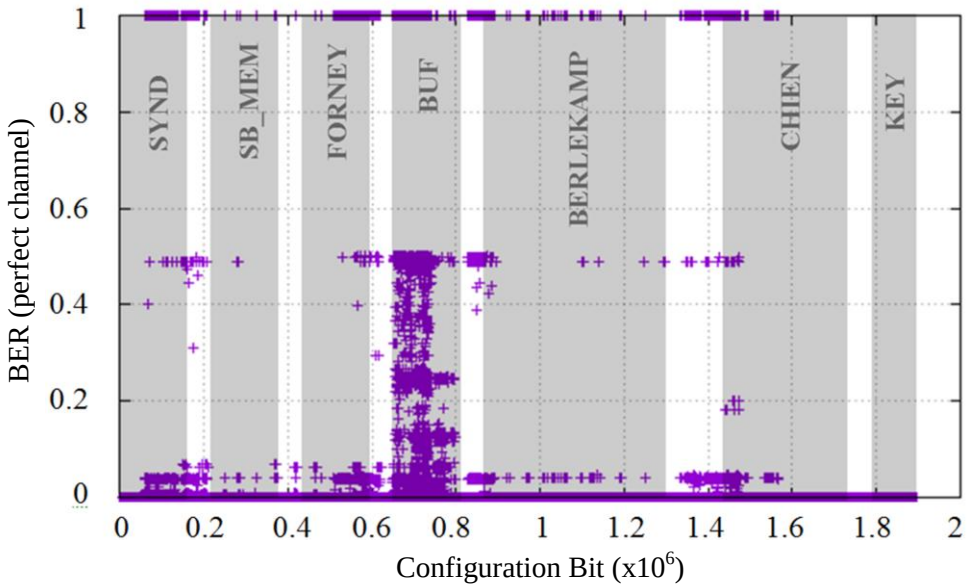
Reed-Solomon Decoder Component	**Number of Configuration Bits**	**Noisy Channel (errors inside of the code capacity)**				
		BER = 0 (no effect)	**BER > 0 or Time Out (configuration failures)**			
		% of total	***% of total***	***% of failures***		
				BER ≈ 0.5 (noise)	***Time Out***	***Other***
Synd	139968	88.11	11.89	0.17	1.39	98.44
Sb_Mem	139968	96.39	3.61	0	0.08	99.92
Forney	139968	88.63	11.37	0.26	2.19	97.55
Buf_Correct	132192	92.43	7.57	10.98	8.04	80.98
Berlekamp	409536	85.61	14.39	0.06	0.14	99.8
Chien	326592	95.89	4.11	0.13	5.44	94.43
Key	93312	91.02	8.98	0	0	100
Area Under Test	1897600	93.12	6.88	1.1	2.11	96.79

Source: the author.

The detailed results presented in Table 4.2 and Table 4.3 does not show the BER variation over the configuration bits flipped. For that purpose, the Figure 4.5 shows the BER of the decoder associated with each bit flip when no channel error is introduced, while Figure 4.6 shows the FER for the same situation. Despite the large amount of configuration bits makes it difficult to perform an accurate analysis of the graphical distribution of the BER values for critical bits, it is evident that most of them are confined in specific regions. Except for the *Buf_Correct* module, which has a big BER variation in the region were $0 \leq BER \leq 0.5$,

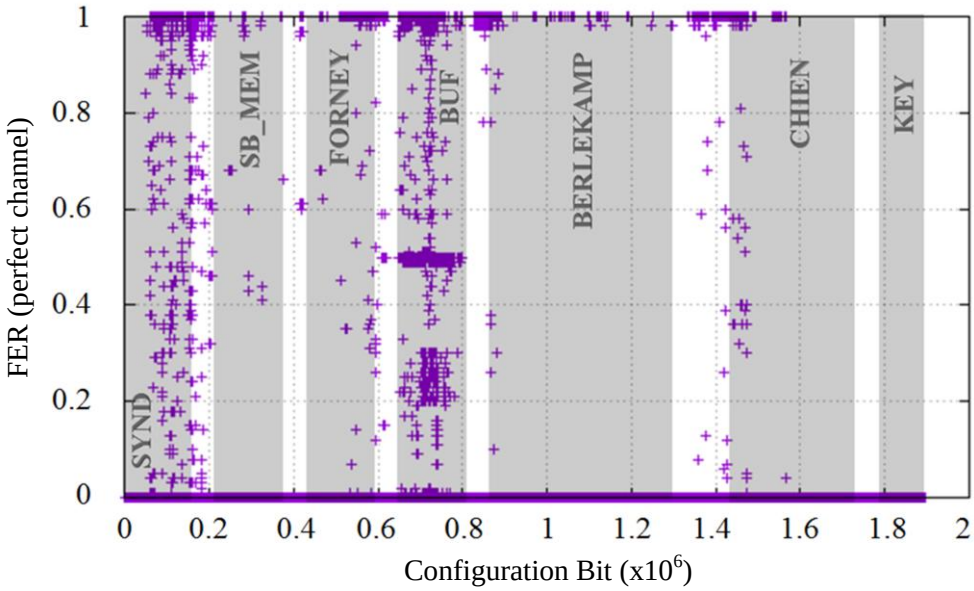
the majority of the configuration bits introduces BER values in the regions were $0 \leq BER \leq 0.1$, $BER \approx 0.5$, and $BER \approx 1$ (which denotes CUT time out). Regarding FER, most faults either do not affect output or make every frame incorrect. Some faults, however, introduce intermediate FER figures, which means these faults are most likely input-sensitive.

Figure 4.5 – BER due to injected faults (first implementation/perfect channel).



Source: the author.

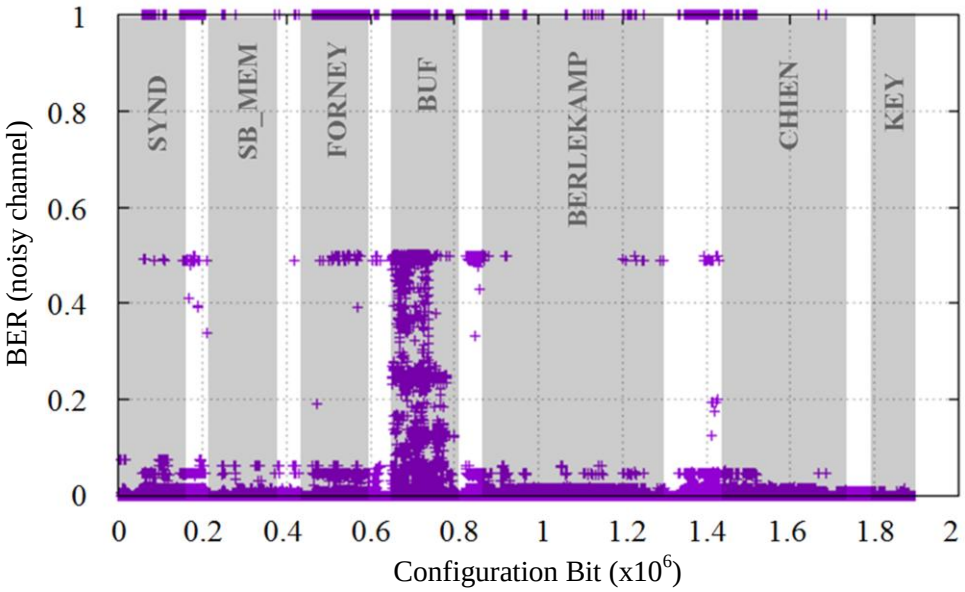
Figure 4.6 – FER due to injected faults (first implementation/perfect channel).



Source: the author.

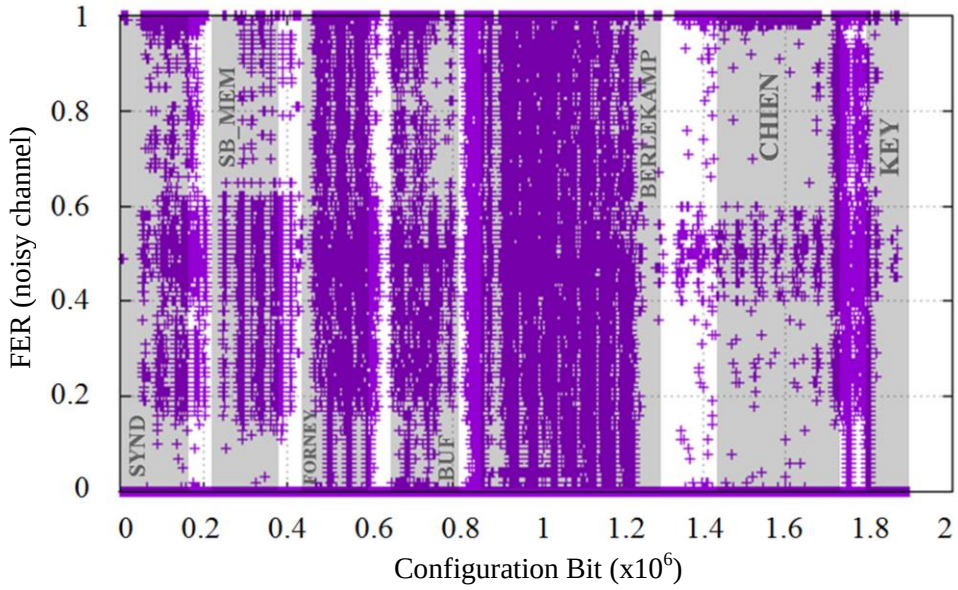
The BER variation with a noisy channel is similar to the perfect channel, as can be seen in Figure 4.7. The most important difference is a bigger concentration of BER values in the region were $0 \leq BER \leq 0.1$ and the appearance of sensitive bits in the *Key* module. The FER for the noisy channel can be seen in the Figure 4.8. The amount of faults that introduce intermediate FER values is considerable bigger compared to the perfect channel, especially when the *Berlekamp* module is affected, which indicates that the new critical bits introduced by the noisy channel are mostly input-sensitive.

Figure 4.7 – BER due to injected faults (first implementation/noisy channel).



Source: the author.

Figure 4.8 – FER due to injected faults (first implementation/noisy channel).

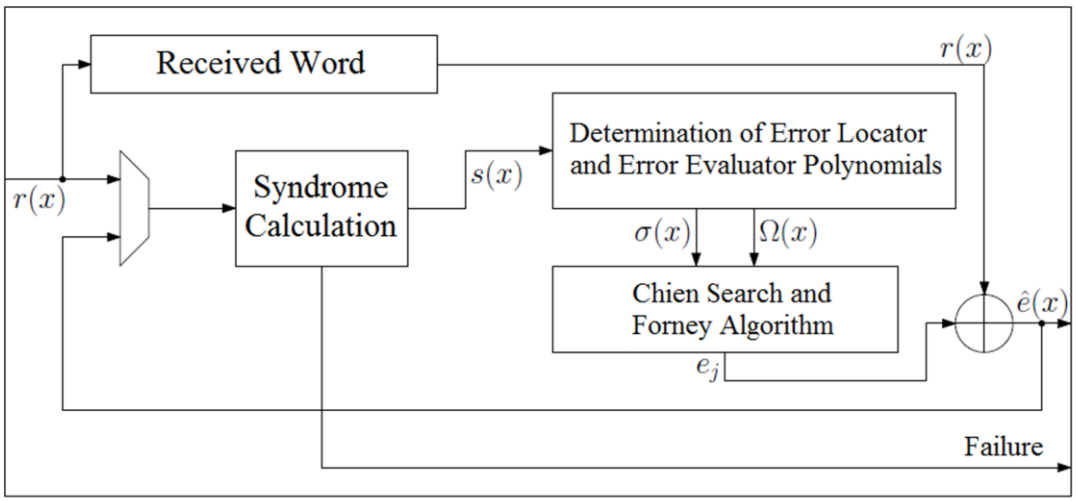


Source: the author.

4.2.2 Second Implementation

The second Reed-Solomon decoder implementation evaluated was taken from the work presented by Grimm (2014), which uses state-of-art techniques to implement the algorithms needed through optimized structures. Regarding the amount of basic resources needed, the decoder uses 2891 registers (flip-flops) and 4894 LUTs. Moreover, this decoder implementation allows an operating clock speed of up to 254 MHz, with one symbol being received or flushed out per clock cycle. Despite being twice as fast as the previous Reed-Solomon decoder evaluated, it does not use a pipelined architecture, which reduces the data throughput. Using an RS(255,239) decoder with 8-bit words, the throughput achieved was very similar to the previous implementation (around 125 MB/s). For this implementation, the AUT consist of 1897600 configuration bits. Therefore, each fault injection campaign took approximately the same time as the first implementation (around two days). Figure 4.9 shows the internal architecture of the decoder and its main modules.

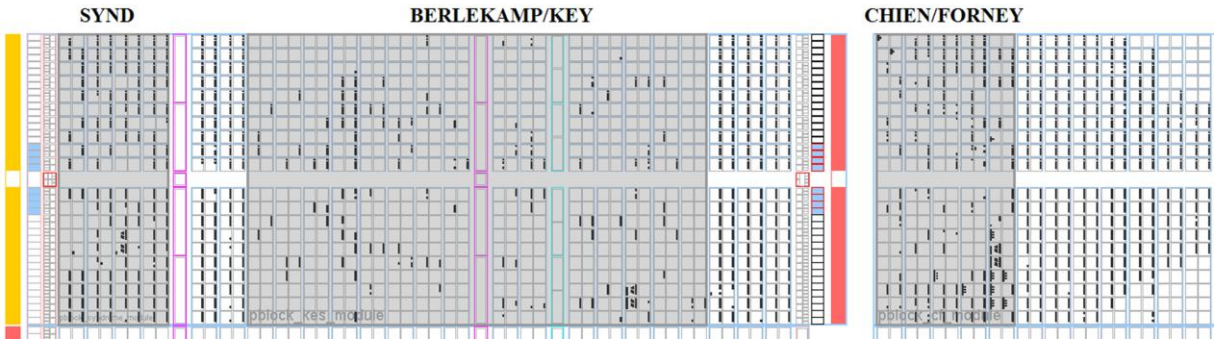
Figure 4.9 – Reed-Solomon decoder structure (second implementation).



Source: adapted from Grimm (2014).

The determination of the error locator and the error evaluator polynomials (Berlekamp's algorithm and Key equation) are performed together in one module, while the Chien search and the Forney algorithm are performed together in another one. The result is only three main modules, whose placement in the AUT was made as shown in Figure 4.10. Table 4.4 and Table 4.5 show the BER results for a perfect channel and for a noisy channel, respectively.

Figure 4.10 – Placement of the Reed-Solomon decoder modules (second implementation).



Source: the author.

Table 4.4 – Reed-Solomon decoder evaluation (second implementation/perfect channel).

Reed-Solomon Decoder Component	**Number of Configuration Bits**	**Perfect Channel**				
		BER = 0 (no effect)	**BER > 0 or Time Out (configuration failures)**			
		% of total	***% of total***	***% of failures***		
				BER ≈ 0.5 (noise)	***Time Out***	***Other***
Synd	186624	97.07	2.93	2.49	3.90	93.61
Berlekamp/Key	775008	99.49	0.51	3.93	12.34	83.73
Chien/Forney	233280	96.83	3.17	3.08	0.40	96.52
Area Under Test	1897600	97.52	2.48	1.51	1.92	96.57

Source: the author.

Table 4.5 – Reed-Solomon decoder evaluation (second implementation/noisy channel).

Reed-Solomon Decoder Component	**Number of Configuration Bits**	**Noisy Channel (errors inside of the code capacity)**				
		BER = 0 (no effect)	**BER > 0 or Time Out (configuration failures)**			
		% of total	***% of total***	***% of failures***		
				BER ≈ 0.5 (noise)	***Time Out***	***Other***
Synd	186624	90.91	9.09	0.77	1.27	97.96
Berlekamp/Key	775008	76.12	23.88	0.08	0.13	99.79
Chien/Forney	233280	89.57	10.43	0.97	1.36	97.67
Area Under Test	1897600	86.06	13.94	0.30	0.38	99.32

Source: the author.

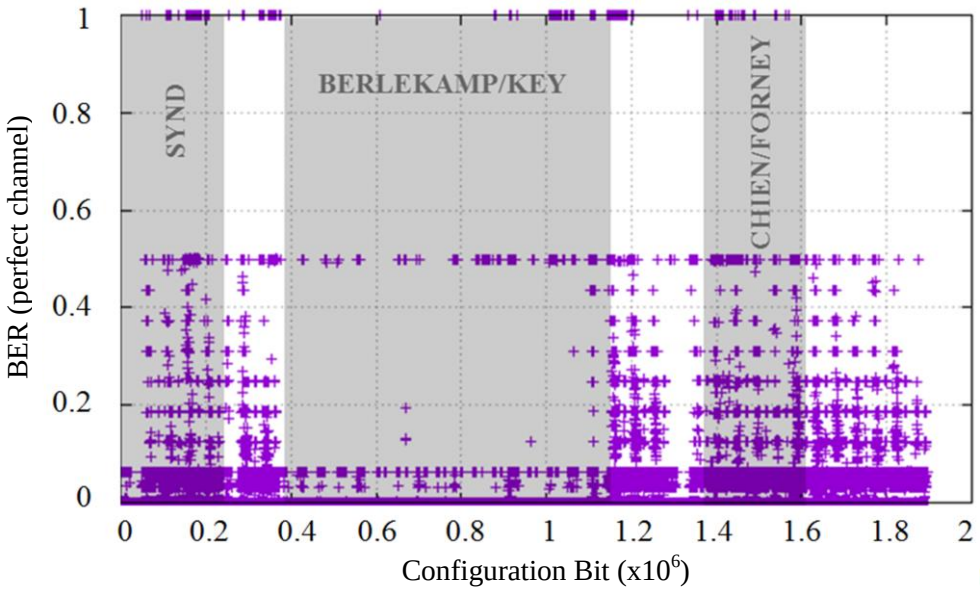
With a perfect channel condition, as expected, the majority of the configuration bits do not affect the system operation. Regarding the critical bits, the majority introduce a BER that varies with the component affected and the data blocks received. The predominance of intermediate BER values is considerable higher when compared with the previous

implementation, i.e., it is less sensible to faults that cause a noisy output or a time out condition. This result may be related to the fact that this implementation does not use a pipelined architecture.

With a noisy channel, the number of critical bits increases in a very similar way compared with the previous implementation, i.e., almost all of those new critical bits introduce a variable BER. As can be seen, the *Berlekamp/Key* was the most affected component, with nearly 25% of critical bits. Therefore, the experiments suggest that the Berlekamp's algorithm, independent of the decoder design, is the most critical operation in Reed-Solomon decoders.

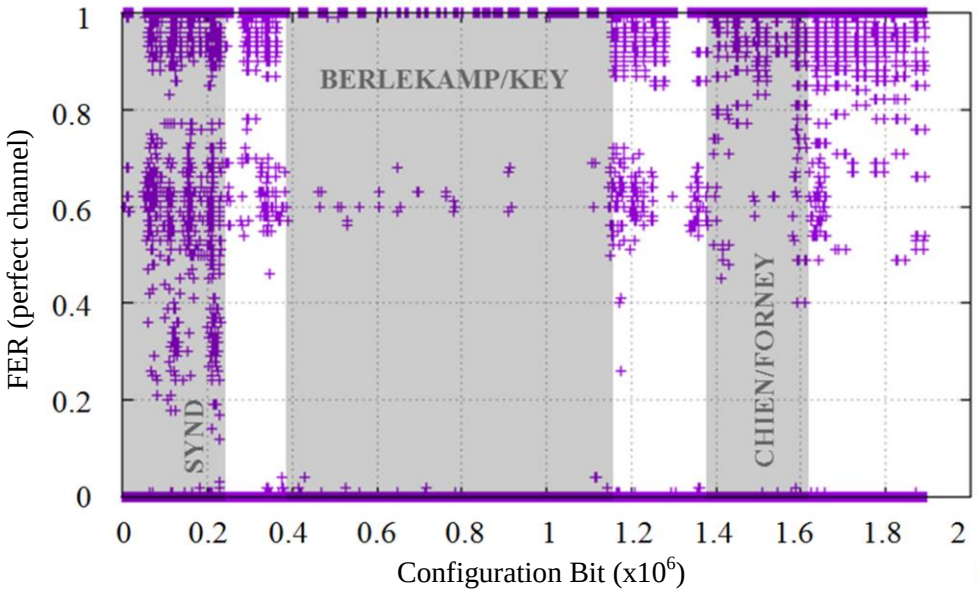
For a more precise evaluation of the BER variation and the identification of input-sensitive faults in the perfect channel scenario, Figure 4.11 and Figure 4.12 show the graphical results for BER and FER, respectively. As can be seen, except for the *Berlekamp/Key* module, the majority of the configuration bits introduce BER values in the regions were $0 \leq BER \leq 0.5$, with a bigger concentration in the region were $0 \leq BER \leq 0.1$. Regarding FER, most faults either do not affect output or make every frame incorrect. Except for the *Syndrome* module, most of intermediate FER values are concentered in the region were $0.5 \leq FER < 1$, which means these faults lead to a system failure for the majority of the input frames. Moreover, the *Berlekamp/Key* module has less input-sensitive critical bits than the other modules.

Figure 4.11 – BER due to injected faults (second implementation/perfect channel).



Source: the author.

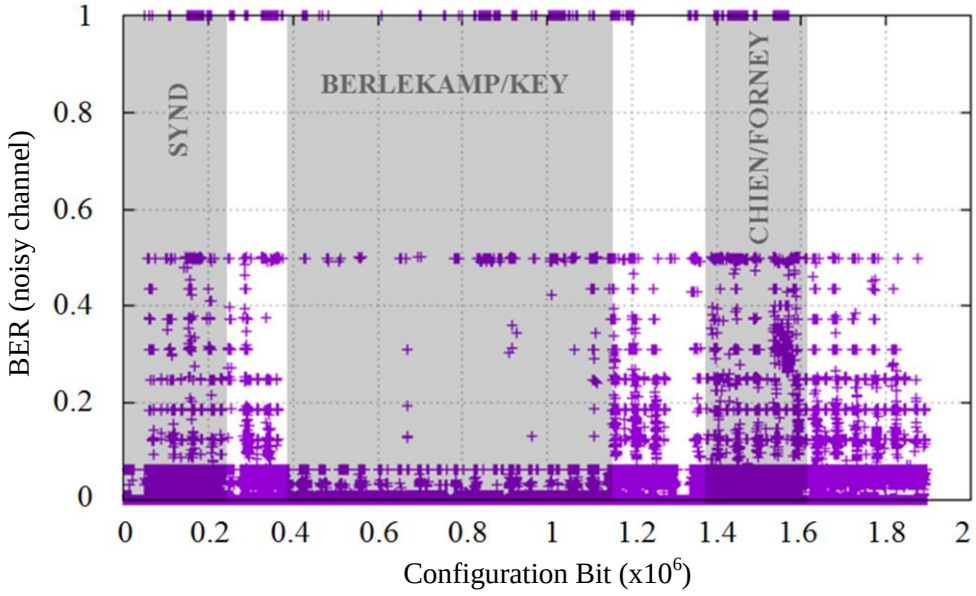
Figure 4.12 – FER due to injected faults (second implementation/perfect channel).



Source: the author.

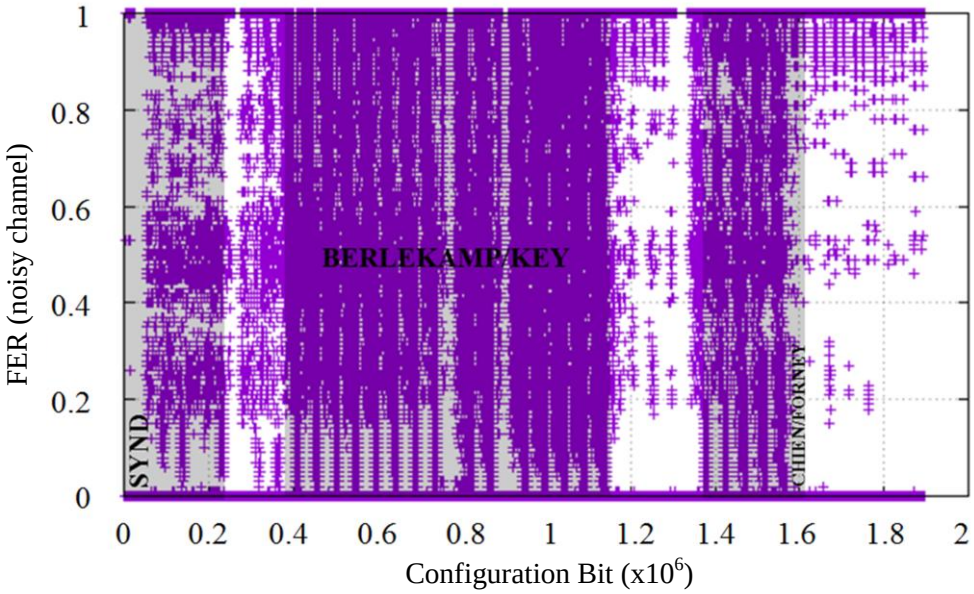
The BER variation with a noisy channel is similar to the perfect channel, as can be seen in Figure 4.13. The most important difference is a bigger concentration of BER values in the region were $0 \leq BER \leq 0.1$. The FER for the noisy channel can be seen in the Figure 4.14. Like the previous implementation, the amount of faults that introduce intermediate FER values with a noisy channel is considerable bigger compared to the perfect channel, especially when the *Berlekamp/Key* module is affected, which indicates that the new critical bits are mostly input-sensitive.

Figure 4.13 – BER due to injected faults (second implementation/noisy channel).



Source: the author.

Figure 4.14 – FER due to injected faults (second implementation/noisy channel).



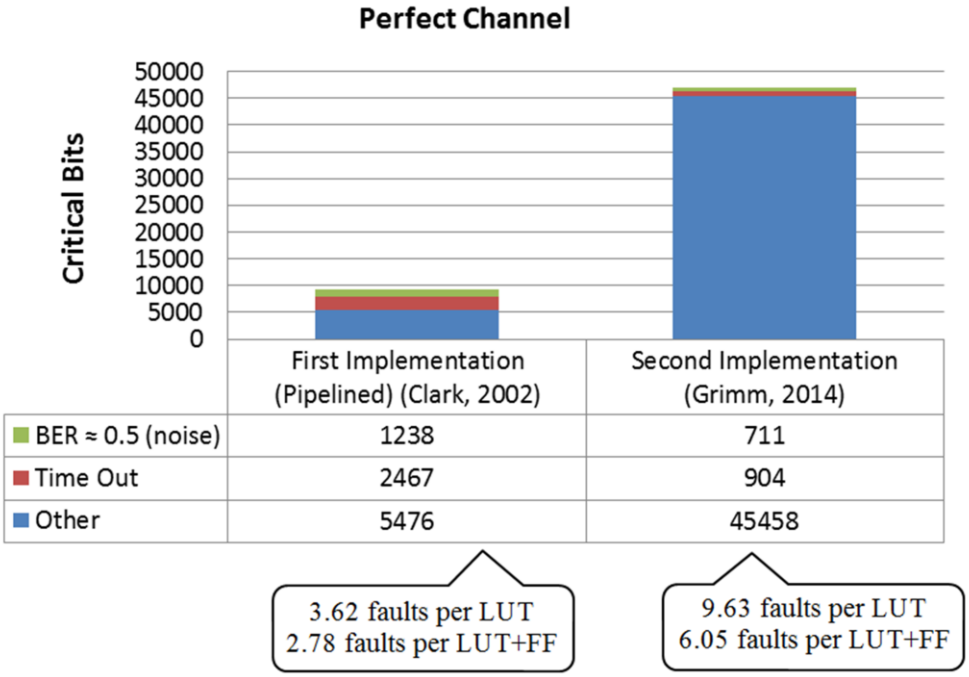
Source: the author.

4.2.3 Critical Bits and Failure Modes Comparison

When evaluating the effects of configuration faults on different implementations of a given system, its important compare no just the amount of critical bits, but also the failure modes related. For that purpose, Figure 4.15 and Figure 4.16 show a comparison between the implemented decoders regarding the critical bits and the failure modes (BER) for a perfect channel and for a noisy channel, respectively.

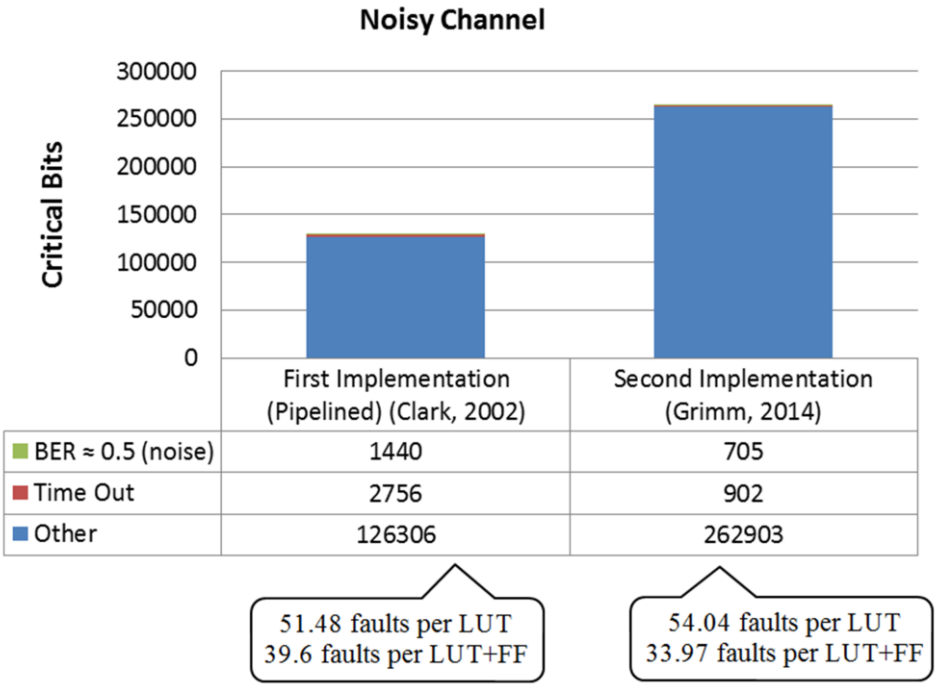
The comparison endorses the observations made in the previous sections. As can be seen, the second implementation is proportionally more sensitive to configuration faults that cause a variable BER when the decoder do not need to correct the incoming data words. However, the proportion of faults that cause time out or noisy output failures is bigger in the first implementation. When the decoder needs correct words corrupted by a noisy channel, i.e., more areas of the circuit are stimulated, the increasing of critical bits is proportional with the amount of resources used by those implementations (the second implementation uses approximately twice as much resources). Moreover, in both implementations, the vast majority of the new critical bits introduce a variable BER.

Figure 4.15 – Critical bits and failure modes comparison (perfect channel).



Source: the author.

Figure 4.16 – Critical bits and failure modes comparison (noisy channel).

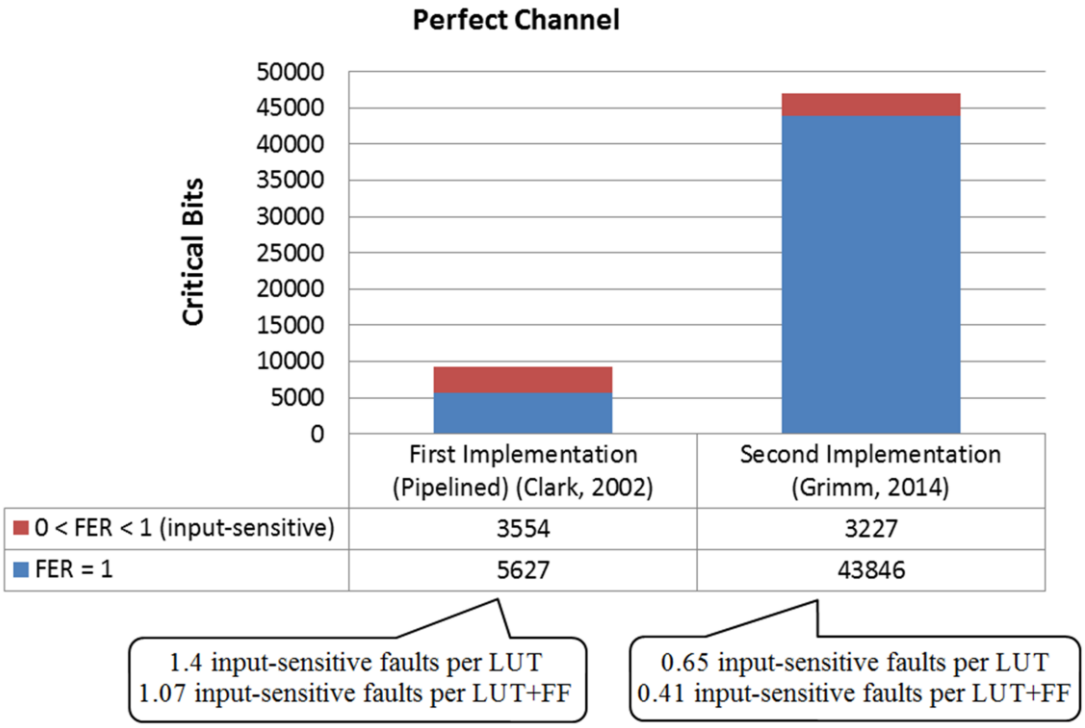


Source: the author.

Regarding the comparison of input-sensitive faults through the FER, as can be seen in Figure 4.17 and Figure 4.18, both implementations present a big increase of those faults with a noisy channel. Moreover, input-sensitive faults become predominant in that scenario, with a

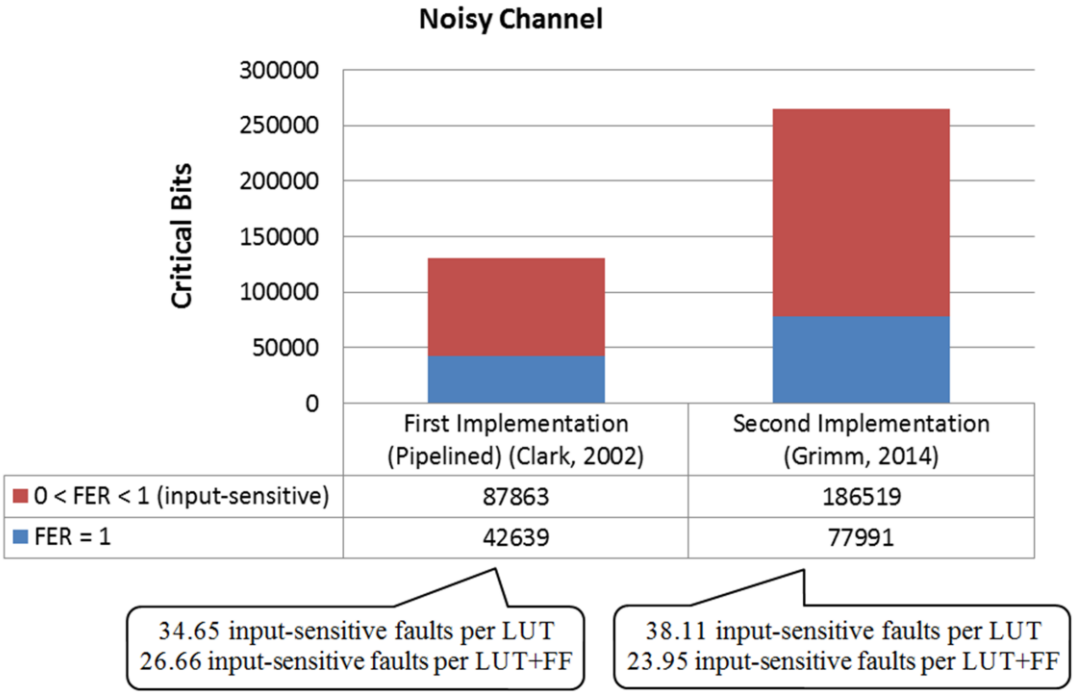
more notable change in the second implementation. In the perfect channel scenario, however, the proportion of input-sensitive faults is bigger in the first implementation.

Figure 4.17 – Input-sensitive faults comparison (perfect channel).



Source: the author.

Figure 4.18 – Input-sensitive faults comparison (noisy channel).



Source: the author.

5 CONCLUSIONS

In this work we have presented a fault injection platform optimized for use with FPGA-based communication systems. The main goal was to provide a tool that allows evaluating the impact of the most critical FPGA faults (configuration memory faults) on communication metrics, such as FER and BER. The tool allows the partitioning of tasks between the FPGA device and a host computer, in order to maximize performance and simplify modifying and extending the platform for different CUTs. The results obtained with a Reed-Solomon decoder showed its ability to measure important metrics for this class of applications.

The two different implementations of an RS(255, 239) code showed a considerable increasing of critical bits when the decoder needs correct words corrupted by a noisy channel. Moreover, the Berlekamp's algorithm was the most critical operation for that scenario in both implementations. Regarding the failure modes, the pipelined decoder based on the traditional architecture presented by Clark (2002) presents more faults that cause a noisy output (decoupled from the input) or a time out condition, while the Grimm's (2014) state-of-art implementation presents a bigger concentration of input-sensitive faults for a noisy channel scenario. Part of those differences may be related to the fact that the previous implementation does not use a pipelined architecture.

Main future works include extending the platform to support other FPGA families, as well as using its fault injection results to guide the development of fault mitigation techniques for communication systems.

REFERENCES

ARDEN, W.; BRILLOUËT, M.; COGEZ, P.; GRAEF, M.; HUIZING, B.; MAHNKOPF, R. **More-than-Moore.** International Roadmap Technology for Semiconductors (ITRS), 2010 [Online]. Available: http://www.itrs.net/ITRS%201999-2014%20Mtgs,%20Presentations%20&%20Links/2010ITRS/IRC-ITRS-MtM-v2%203.pdf.

DIXIT, Anand; WOOD, Alan. **The impact of new technology on soft error rates.** In Reliability Physics Symposium (IRPS), IEEE International, Monterey, CA, April 2011, pp. 5B.4.1 - 5B.4.7.

DODD, P. E.; SHANEYFELT, M. R.; SCHWANK, J. R.; FELIX, J. A. **Current and future challenges in radiation effects on CMOS electronics.** Nuclear Science, IEEE Transactions on, Vol. 57, No. 4, pp. 1747–1763, Aug 2010.

BAUMANN, Robert C. **Radiation-induced soft errors in advanced semiconductor technologies.** Device and Materials Reliability, IEEE Transactions on, vol. 5, no. 3, pp. 305–316, Sept. 2005.

JEDEC, Solid State Technology Association. **Measurement and reporting of alpha particle and terrestrial cosmic ray-induced soft errors in semiconductor devices**. JEDEC Standard JESD89A, Oct. 2006 [Online]. Available: http://www.jedec.org/sites/default/files/docs/jesd89a.pdf.

MUNTEANU, D.; AUTRAN, J. L. **Modeling and simulation of single-event effects in digital devices and ICs.** Nuclear Science, IEEE Transactions on, vol. 55, no. 4, pp. 1854–1878, Aug 2008.

HAUCK, Scott; DEHONN, André. **Reconfigurable computing: theory and practice off FPGA-based computation**. 1 ed., Morgan Kaufmann, US, 2008.

LESEA, A.; DRIMER, S.; FABULA, J.; CARMICHAEL, C.; ALFKE, P. **The Rossetta Experiment: Atmospheric soft error rate testing in differing technology FPGAs.** IEEE Trans. Device Mater. Rel., vol. 5, no. 3, pp. 317–328, Dec. 2005.

QUINN, Heather; ROUSSEL-DUPRE, Diane; CAFFREY, Mike; GRAHAM, Paul; WIRTHLIN, Michael; MORGAN, Keith; SALAZAR, Aanthony; NELSON, Tony; HOWES, Will; JOHNSON, Eric; JOHNSON, Jon; PRATT, Braian; ROLLINS, Nathan; KRONE, Jim. **The Cibola Flight Experiment**. Reconfigurable Technology and Systems, ACM Transactions on, vol. 8, no. 1, article 3, Feb. 2015.

SHANNON, C. E. **A mathematical theory of communication.** Bell System Technical Journal, The, vol. 27, no. 3, pp. 379-423, July 1948. DOI: 10.1002/j.1538-7305.1948.tb01338.x.

BUELL, Duncan; EL-GHAZAWI, Tarek; GAJ, Kris; KINDRATENKO, Volodymyr. **High-performance reconfigurable computing**. Computer, IEEE Computer Society, vol. 40, no. 3, pp. 23–27, March 2007 [Online]. Available: http://www.computer.org/csdl/mags/co/2007/03/r3023.pdf.

GOCKHALE, Maya B.; GRAHAM, Paul S. **Reconfigurable computing: accelerating computation with field-programmable gate arrays**. 1 ed., Springer, Netherlands, 2005.

VIOLANTE, M.; STERPONE, L.; CESCHIA, M.; BORTOLATO, D.; BERNARDI, P.; REORDA, M; S.; PACCAGNELLA, A. **Simulation-based analysis of SEU effects in SRAM-based FPGAs**. IEEE Trans. Nucl. Sci., vol. 51, no. 6, pp. 3354–3359, Dec. 2004.

WIRTHLIN, Michael. **High-reliability FPGA-based systems: space, high-energy physics, and beyond**. IEEE Proceedings, vol. 103, no. 3, pp. 379–389, March 2015.

QUINN, Heather; MORGAN, Keith; GRAHAM, Paul; KRONE, Jim; CAFFREY, Michael; LUNDGREEN, Kevin. **Domain crossing errors: limitations on single device triple-modular redundancy circuits in Xilinx FPGAs**. Nuclear Science, IEEE Transactions on, vol. 54, no. 6, pp. 2037–2043, Dec. 2007.

KASTENSMIDT, Fernanda Lima; REIS, Ricardo. **Fault-Tolerance Techniques for SRAM-Based FPGAs**. Frontiers in Electronic Testing, 1 ed., vol. 32, Springer, US, 2006.

CLARK, J. A.; PRADHAN, D. K. **Fault injection: a method for validating computer-system dependability**. Computer, IEEE Computer Society, vol. 28, no. 6, pp. 47–56, June 1995.

ZIADE, Haissam; AYOUBI, Rafic; VELAZCO, Raoul. **A Survey on Fault Injection Techniques**. International Arab Journal of Information Technology, vol. 1, no. 6, pp. 171–186, July 2004.

ARLAT, Jean; AGUERA, Martine; AMAT, Louis; CROUZET, Yves; FABRE, Jean-Charles; LAPRIE, Jean-Claude; MARTINS, Eliane; POWELL, David. **Fault injection for dependability validation: a methodology and some applications**. Software Engineering, IEEE Transactions on, vol. 16, no. 2, pp. 166–182, Feb. 1990.

QUINN, Heather; WIRTHLIN, Michael. **Validation techniques for fault emulation of SRAM-based FPGAs**. Nuclear Science, IEEE Transactions on, vol. 62, no. 4, pp. 1487–1500, Aug. 2015.

STERPONE, L.; VIOLANTE, M. **Analysis of the robustness of the TMR architecture in SRAM-based FPGAs**. Nuclear Science, IEEE Transactions on, vol. 52, no. 5, pp. 1545–1549, Oct. 2005.

LIMA, F.; CARMICHAEL, C.; FABULA, J.; PADOVANI, R.; REIS, R. **A fault injection analysis of virtex FPGA TMR methodology**. In Proc. IEEE Eur. Conf. Radiation and Its Effect on Component and Systems, 2001, pp. 275–282.

ALDERIGHI, M.; CASINI, F.; D'ANGELO, S.; MANCINI, M.; PASTORE, S.; SECHI, G. R. **Evaluation of Single-Event upset mitigation schemes for SRAM based FPGAs using the FLIPPER fault injection platform**. In Proc. 2007 Int. Symp. Defect and Fault Tolerance in VLSI Systems, Rome, Italy, Sept. 2007, pp. 105-113.

STERPONE, L.; VIOLANTE, M. **A new partial reconfiguration-based fault injection system to evaluate SEU effects in SRAM-based FPGAs**. Nuclear Science, IEEE Transactions on, vol. 54, pp. 965–970, Aug 2007.

NAZAR, G.; CARRO, L. **Fast single-FPGA fault injection platform**. In Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2012 IEEE International Symposium on, pp. 152–157, Oct 2012.

DI CARLO, Stefano; PRINETTO, Paolo; ROLFO, Daniele; TROTTA, Pascal. **A fault injection methodology and infrastructure for fast Single-Event upsets emulation on xilinx SRAM-based FPGAs**. In Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2014 IEEE International Symposium on, pp. 159–164, Oct 2014.

TARRILLO, Jimmy; KASTENSMIDT, Fernanda Lima; RECH, Paolo; FROST, Christopher; VALDERRAMA, Carlos. **Neutron cross-section of n-modular redundancy technique in SRAM-based FPGAs.** Nuclear Science, IEEE Transactions on, vol. 61, no. 4, pp. 1558–1566, Aug. 2014.

GIMMLER-DUMONT, C.; SCHLAFER, P.; WEHN, N. **FPGA-based rapid prototyping platform for MIMO-BICM design space exploration.** In Reconfigurable Computing and FPGAs (ReConFig), 2012 IEEE International Conference on, pp. 1–7, Dec. 2012.

CCSDS. **TM synchronization and channel coding**. Recommendation For Space Data System Standard, Aug. 2011 [Online]. Available: http://public.ccsds.org/publications/archive/131x0b2ec1.pdf

CARMICHAEL, C.; CAFFREY, M.; SALAZAR, A. **Correcting single-event upsets through virtex partial configuration**. Xilinx Application Notes XAPP216 (v1.0), 2000.

BILLAUER, Eli. **An FPGA IP core for easy DMA over PCIe with windows and linux**. 2014 [Online]. Available: http://xillybus.com.

XILINX, Inc. **LogiCORE IP Endpoint Block Plus v1.15 for PCI Express**. UG341, June 2011 [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/pcie_blk_plus/v1_15/pcie_blk_plus_ug341.pdf.

XILINX, Inc. **LogiCORE IP FIFO generator v8.4**. UG175, Jan. 2012. [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/fifo_generator/v8_4/fifo_generator_ug175.pdf.

XILINX, Inc. **Virtex-5 FPGA configuration user guide**. UG191, Oct. 2012 [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug191.pdf.

XILINX, Inc. **ML505/ML506/ML507 evaluation platform user guide**. UG347, May 2011 [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug191.pdf.

XILINX, Inc. **Xilinx university program XUPV5-LX110T development system**. [Online]. Available: http://www.xilinx.com/univ/xupv5-lx110t.htm.

VIGNA, Sebastiano. **Further scramblings of Marsaglia's xorshift generators**. 2014. [Online]. Available: http://vigna.di.unimi.it/ftp/papers/xorshift.pdf.

WICKER, S. B.; BHARGAVA, V. K. **Reed–Solomon Codes and Their Applications**. Piscataway, NJ: IEEE Press, 1994.

CLARKE, C. K. P. **Reed-Solomon Error Correction**. BBC R&D White Paper WHP 031, July 2002 [Online]. Available: http://downloads.bbc.co.uk/rd/pubs/whp/whp-pdf-files/WHP031.pdf.

GRIMM, Tomás. **Desenvolvimento em linguagem de descrição de hardware de codificador e decodificador Reed-Solomon.** Florianópolis, SC, 2014. 142 p**.** Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Engenharia Elétrica.

APPENDIX A – GRADUATION PROJECT I

Uma Plataforma para Avaliação da Confiabilidade de Sistemas de Comunicação em FPGAs

Marcos T. Leipnitz[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul
Av. Bento Gonçalves, 9500 – Porto Alegre, RS – Brazil

mtleipnitz@inf.ufrgs.br

Abstract*. The reliability of communication systems in environments with radiation incidence is a major concern, especially when using FPGAs (Field-Programmable Gate Arrays) as a design platform. This paper presents a fault injection platform for communication systems based on FPGAs, in order to simulate and evaluate the effects of radiation-induced SEUs (Single-Event Upsets) on the functionality of these systems. It is intended that this platform will be useful for design and validation of effective strategies to mitigate faults. A schedule of implementation and validation of the proposed platform, as well as some preliminary results are presented at the end of this document.*

Resumo*. A confiabilidade dos sistemas de comunicação em ambientes com incidência de radiação é uma grande preocupação, especialmente quando utilizamos FPGAs (Field-Programmable Gate Arrays) como plataforma de implementação. Este trabalho apresenta uma plataforma de injeção de falhas em sistemas de comunicação baseados em FPGAs, com o objetivo de simular e avaliar os efeitos de SEUs (Single-Event Upsets) induzidos por radiação na funcionalidade desses sistemas. Pretende-se que esta plataforma seja útil no processo de elaboração e validação de estratégias eficientes de mitigação de falhas. Um cronograma de implementação e validação da plataforma proposta, bem como resultados preliminares, são apresentados no final deste documento.*

1. Introdução

Alta confiabilidade é um requisito fundamental no projeto de sistemas de comunicação de dados, presentes em uma ampla gama de aplicações. O esforço necessário para atender a esse requisito, com o uso de técnicas de tolerância a falhas, depende de diversos fatores, como a plataforma de implementação ou o ambiente onde o sistema será utilizado.

Os FPGAs (*Field-Programmable Gate Arrays*) são dispositivos reconfiguráveis largamente utilizados na implementação de sistemas de comunicação de dados, pois oferecem muitas vantagens em relação ao uso de circuitos integrados dedicados, tais como reconfigurabilidade, alto desempenho e baixo custo de desenvolvimento. Em aplicações aeroespaciais, como satélites de comunicação, falhas ocorrem com frequência devido ao maior fluxo de partículas ionizantes, provenientes, principalmente, dos raios cósmicos, dos ventos solares e do cinturão de Van Allen.

Em ambientes com incidência de radiação ionizante, determinados eventos podem induzir falhas em dispositivos semicondutores, especialmente naqueles que trabalham com pequenas variações de corrente e com baixos níveis de tensão, como os dispositivos CMOS (*Complementary Metal Oxide Semiconductor*) presentes em sistemas VLSI (*Very Large Scale*

Integration). *Single-Event Effects* (SEEs) são eventos em que uma única partícula ionizante incidente (normalmente nêutrons, prótons ou íons pesados) deposita energia suficiente para causar um efeito no dispositivo. Esse efeito pode ser destrutivo, ou seja, pode danificar fisicamente o dispositivo; ou pode ser não destrutivo, como a propagação de um transiente que pode alterar o estado lógico do sistema ou inverter um bit em uma célula de memória, por exemplo. Um SEE que gera uma mudanc¸a de estado em um dispositivo, sem danificá-lo, também é conhecido como Single-Event Upset (SEU) [Ken Label 1996].

Apesar dos fabricantes de FPGAs, como a Xilinx e a Altera, oferecerem linhas com maior tolerância à incidência de radiação, através de processos de fabricação diferenciados que conferem até mesmo imunidade a SEEs potencialmente destrutivos, como o *Single-Event Latch-up* (SEL), a ocorrência de SEUs nesses dispositivos é frequentemente objeto de estudo [A. Lesea et al. 2005]. Em especial, a memória de configuração de dispositivos baseados em SRAM (*Static Random Access Memory*) tem grande sensibilidade a esse tipo de evento [E. Fuller et al. 1999], potencializada pelo uso de memórias cada vez maiores e mais densas, em função do avanço da tecnologia de fabricação de semicondutores.

A memória de configuração em FPGAs tem a função de definir a funcionalidade do sistema, configurando os elementos que compõem o dispositivo, como flip-flops, multiplexadores, LUTs (*Lookup Tables*), blocos de memória, entre outros. Além disso, essa memória configura a interconexão entre esses elementos. Logo, a ocorrência de SEUs nesses dispositivos pode, além de corromper os dados processados ou alterar o estado lógico do sistema, inverter um ou mais bits da memória de configuração. Essas alterações podem modificar a funcionalidade do sistema, gerando resultados incorretos, ou não ter nenhum efeito observável [M. Violante et al. 2004]. Portanto, é essencial caracterizar o impacto desse tipo de falha em aplicações que exigem alta confiabilidade, de forma a prover mecanismos de tolerância a falhas eficazes e com o menor custo possível.

O impacto de SEUs na confiabilidade de sistemas de computação em FPGAs baseados em SRAM depende dos elementos utilizados na implementação, bem como do posicionamento e do roteamento dos mesmos. As LUTs, que são utilizadas na implementação de lógica combinacional, são os elementos mais sensíveis a SEUs, enquanto que os elementos de interconexão tem maior probabilidade de serem atingidos, pois ocupam a maior parte dos bits de configuração. O impacto de SEUs nestes elementos é alto, pois frequentemente induzem a curtos-circuitos que corrompem os dados do sistema [M. Ceschia et al. 2003]. Além disso, na maioria das aplicações, apenas uma pequena parte da memória de configuração é utilizada [A. Lesea et al. 2005]. Consequentemente, é muito importante identificar os bits críticos dessa memória, ou seja, aqueles bits que, se invertidos, geram uma falha funcional no sistema, pois ajudam no dimensionamento adequado de técnicas de mitigação de falhas [Sheng Wang et al. 2015].

Este trabalho propõe uma plataforma de injeção de falhas na memória de configuração de FPGAs baseados em SRAM, com recursos específicos para avaliar o impacto de SEUs na funcionalidade de sistemas de comunicação de dados. O objetivo é oferecer uma plataforma que permita avaliar e validar mecanismos de tolerância a falhas eficientes e de baixo custo, visando aumentar a confiabilidade desses sistemas em ambientes com incidência de radiação ionizante.

O trabalho está estruturado da seguinte forma: a seção 2 apresenta uma breve análise de trabalhos existentes na área de injeção de falhas em FPGAs; a seção 3 apresenta a plataforma proposta; a seção 4 apresenta alguns resultados preliminares, como os recursos utilizados e a área ocupada pelos componentes de hardware no FPGA; a seção 5 apresenta um

cronograma de atividades para a conclusão do trabalho (implementação e validação) e a seção 6, por fim, apresenta as considerações finais.

2. Trabalhos Relacionados

A injeção de falhas é um método amplamente utilizado para avaliar a dependabilidade de sistemas de computação, bem como para validar mecanismos de tolerância a falhas [Jean Arlat et al. 1990]. Com este método, diversas técnicas de tolerância a falhas foram desenvolvidas e avaliadas com o objetivo de proteger sistemas críticos da ocorrência de SEUs em FPGAs baseados em SRAM [F. L. Kastensmidt et al. 2006]. As técnicas mais eficientes, em geral, combinam métodos de prevenção, como TMR (*Triple Modular Redundancy*), com métodos de correção, como scrubbing, que consiste em reconfigurar periodicamente o dispositivo com uma versão livre de falhas da memória de configuração [Uros Legat et al. 2012]. Este conjunto é avaliado, por exemplo, em [F. L. Kastensmidt et al. 2001], em que o scrubbing é utilizado para eliminar o acúmulo de falhas que pode deteriorar a eficiência da técnica TMR.

TMR é uma técnica de tolerância a falhas muito utilizada para mitigar os efeitos de SEUs, em que usa-se três cópias do mesmo sistema e um votador que escolhe a saída majoritária, com o objetivo de evitar a propagação de erros nas saídas do sistema. Esta técnica, no entanto, tem uma série de limitações, especialmente quando ocorrem *Multi-Bit Upsets* (MBUs), ou seja, quando um SEU modifica mais de um bit na memória de configuração [Heather Quinn et al. 2007]. O votador, por exemplo, é um ponto único de falha. Além disso, essa técnica é especialmente suscetível a falhas que afetam os elementos de roteamento do FPGA, podendo gerar múltiplos erros nos módulos [L. Sterpone and M. Violante 2005]. Muitos trabalhos foram desenvolvidos visando contornar limitações desse tipo e aumentar a eficiência de técnicas de tolerância a falhas para FPGAs baseados em SRAM, exemplificando como as plataformas de injeção de falhas podem ser utilizadas para este fim.

Com a importância de determinar a confiabilidade dos sistemas de computação nos primeiros estágios do projeto, de forma a permitir a exploração de alternativas para proteger esses sistemas em ambientes com incidência de radiação ionizante, muitas plataformas de injeção de falhas por simulação foram desenvolvidas na literatura. Nessas plataformas, os bits da memória de configuração são invertidos por reconfiguração total ou parcial, interna ou externamente, de forma a simular a ocorrência de SEUs. Após a inversão, o sistema é executado com vetores de teste e os seus resultados são comparados com os resultados de uma versão livre de falhas do mesmo sistema (Golden Results). Muitos trabalhos já demostraram a validade deste método [F. L. Kastensmidt et al. 2001][M. Wirthlin et al. 2003], que apresenta muitas vantagens em relação ao uso de infraestrutura especializada para a simulação de SEUs (acelerador de partículas, por exemplo), como baixo custo e alta controlabilidade. A figura 1 mostra um fluxograma básico do método de injeção de falhas [Heather Quinn et al. 2008].

Em [Gabriel L. Nazar and Luigi Carro 2012] é apresentada uma plataforma baseada em um único FPGA, projetada com os elementos mais comuns nesses dispositivos (LUTs, flip-flops, blocos de memória, entre outros), o que possibilita a sua implementação em qualquer FPGA que possua uma interface interna de configuração. O uso dessa interface é o que permite o uso do mesmo FPGA tanto para o injetor de falhas como para o *Circuit Under Test* (CUT), pois as falhas são injetadas em uma região específica do dispositivo, *Area Under Test* (AUT), através de reconfiguração parcial, evitando que o próprio injetor seja atingido pelas falhas injetadas, o que inutilizaria a plataforma. No que diz respeito ao desempenho, essa plataforma permite tempos de injeção e remoção de falhas da ordem de 10 µs ou menos,

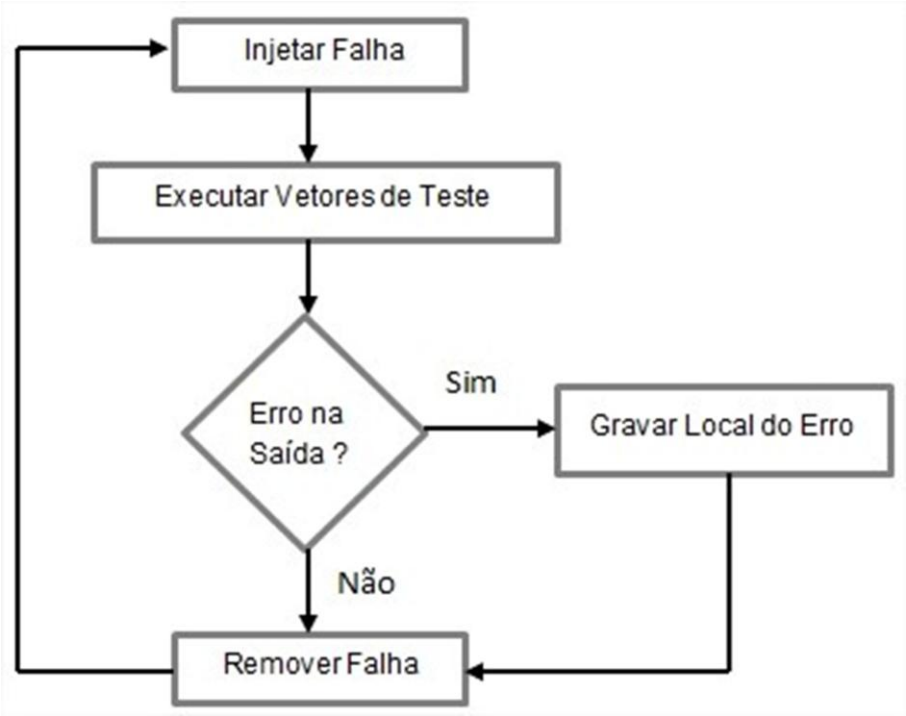


Figura 1. Funcionamento básico de um injetor de falhas.

sendo esta uma das suas principais características. O desempenho será ditado, em geral, pelo desempenho do CUT no processo de estímulo, ou pela capacidade de comunicação da plataforma com o computador, utilizada para enviar os resultados obtidos com cada falha injetada. Dado que a comunicação é via cabo serial (RS-232), é provável que na maioria dos casos este seja o gargalo de desempenho do sistema proposto.

Em [Stefano Di Carlo et al. 2014] é apresentada uma metodologia de injeção de falhas que utiliza reconfiguração parcial em conjunto com a tecnologia *Essential Bits*, da Xilinx. Esta tecnologia permite identificar e extrair os bits essenciais do *bitstream* de configuração, ou seja, aqueles bits que são essenciais para a funcionalidade do sistema. Esse método diminui muito o tempo do processo de injeção de falhas, pois todos os bits da memória de configuração que não tem relação com a funcionalidade do sistema (a grande maioria) são desconsiderados. Além disso, esta plataforma não exige nenhum conhecimento a respeito do endereçamento dos frames da memória de configuração, pois isola o sistema a ser testado em uma partição reconfigurável definida previamente com o software da Xilinx.

A plataforma de injeção de falhas proposta neste trabalho consiste em uma versão modificada da plataforma apresentada em [Gabriel L. Nazar and Luigi Carro 2012]. Uma das principais modificações é a inclusão de uma interface PCIe para estabelecer a comunicação entre o FPGA e o PC. Aproveitando a capacidade dessa interface, um software fica responsável por inicializar o sistema de controle no FPGA, enviar os vetores de teste, receber os resultados e avaliá-los, aumentando a flexibilidade da plataforma, pois as funções oferecidas pelo software são parametrizáveis e podem ser facilmente estendidas. Além disso, o impacto da comunicação do FPGA com o PC no tempo para realizar uma campanha de injeção de falhas é consideravelmente menor, se compararmos com a versão original (RS-232). Essa característica é especialmente importante quando avaliamos a confiabilidade de sistemas de comunicação de dados, pois esse tipo de avaliação exige uma vazão de dados relativamente grande.

3. Plataforma de Injeção de Falhas

A plataforma de injeção de falhas proposta é constituída por dois grandes componentes: um de hardware e outro de software. O hardware está sendo implementado em um FPGA da linha Virtex-5, da Xilinx, com a plataforma de desenvolvimento XUPV5-LX110T [Xilinx XUPV5LX110T] disponível no Laboratório de Sistemas Embarcados (LSE). O software está sendo desenvolvido para no sistema operacional Windowns 7, na linguagem de programação C++. A comunicação entre o hardware no FPGA e o software no computador é feita pelo barramento PCIe 1.0 x1 oferecido pela plataforma de desenvolvimento, ou seja, permite uma taxa de transferência, em teoria, de até 250 MB/s. A figura 2 mostra, de forma simplificada, os componentes que constituem a plataforma. Nas subseções seguintes, cada um desses componentes será apresentado com maiores detalhes.

3.1. Controlador de E/S do PCIe

O controlador de E/S do PCIe é responsável pela comunicação do hardware da plataforma com o software que executa no PC, através da interface PCIe disponibilizada pela plataforma XUPV5LX110T. Como o desenvolvimento de uma solução PCIe para comunicação entre o FPGA e o PC é uma tarefa relativamente complexa e está fora do escopo deste trabalho (pelo menos neste momento), foram analisadas algumas opções já desenvolvidas em outros trabalhos.

O projeto Riffa (*Reusable Integration Framework for FPGA Accelerators*) [Matthew Jacobsen et al. 2012] apresenta uma solução de comunicação PCIe, mas a primeira versão, que é a única que oferece suporte ao Virtex-5, tem um problema de limitação com a taxa de transferência, que fica em torno de apenas 24 MB/s. Já a biblioteca de comunicação EPEE [Jian Gong et al. 2014] oferece uma solução que pode ser explorada futuramente, pois oferece suporte apenas para Linux, assim como o Riffa.

A solução de comunicação adotada neste trabalho foi o sistema Xillybus [xillybus 2014]. Esta solução é composta por um IP Core disponível para diversos modelos de FPGAs (incluindo o Virtex-5) e um driver de dispositivo, tanto para Linux como para Windows. Segundo o desenvolvedor, o sistema oferece uma taxa de transferência de até 200 MB/s (*full-duplex*) para o padrão PCIe 1.0 x1, ou seja, cerca de 80% da taxa máxima. A comunicação pode ser feita de forma síncrona ou assíncrona, com um ou mais *streams*, através de operações de E/S em *device files*, que fazem a interface com o driver do dispositivo. A alta vazão de dados é obtida pelo uso extensivo de buffers DMA (*Direct Memory Access*). O uso desta solução é livre de licença, desde que utilizado exclusivamente para trabalhos acadêmicos ou de pesquisa.

Através da ferramenta disponibilizada pelo desenvolvedor, o Xillybus IP Core foi configurado com dois *streams*: um de 8 bits, síncrono, com taxa de transferência de até 1 MB/s (*full-duplex*), e outro de 32 bits, assíncrono, com taxa de transferência de até 195 MB/s (*full-duplex*). O *stream* de 8 bits permite que o software envie comandos para o controlador do sistema e também para que este controlador envie o seu estado atual para o software. Já o *stream* de 32 bits permite que o software envie os vetores de teste para o controlador do CUT e também para que este controlador envie os resultados gerados pelo CUT para o software avaliar.

A interface do Xillybus IP Core com o barramento PCIe é feita com um IP Core fornecido pela Xilinx através do *Core Generator*, o *Endpoint Block Plus for PCI Express* [Xilinx EBP PCIe]. A interface com o controlador do sistema e o com o controlador de E/S do CUT é feita com o uso de memórias FIFO, que também podem ser geradas com o *Core Generator*. A figura 3 mostra os detalhes do controlador de E/S do PCIe.

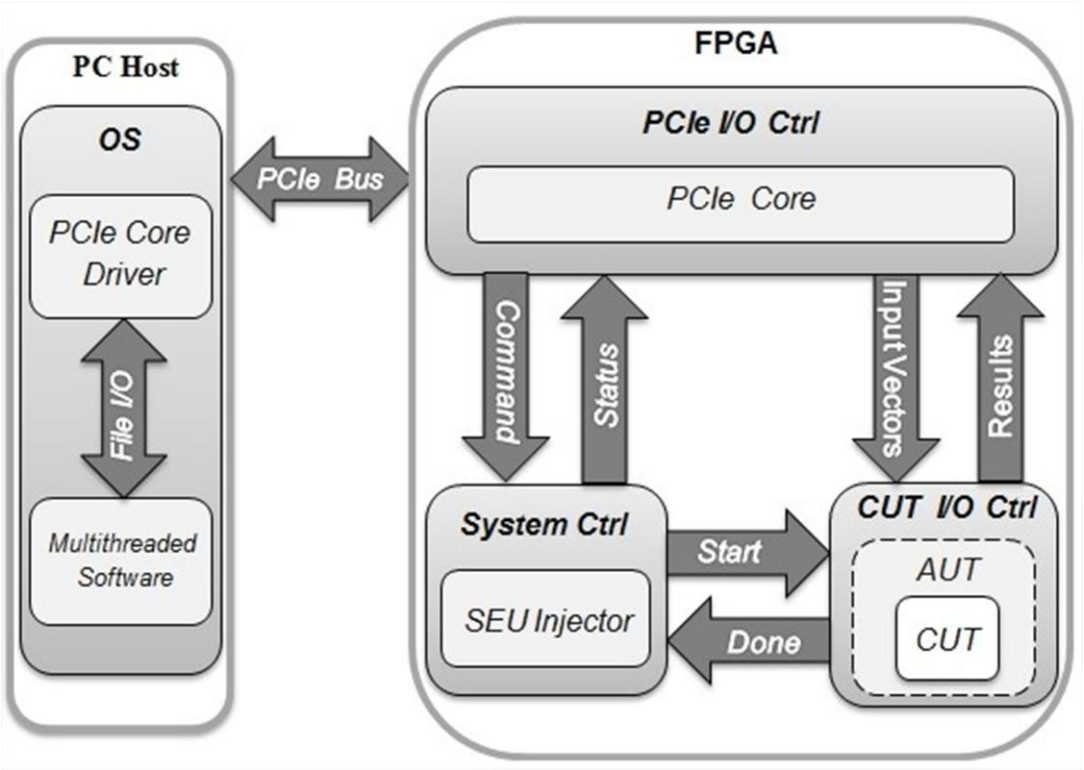


Figura 2. Componentes básicos da plataforma de injeção de falhas proposta.

3.2. Controlador do Sistema

O controlador do sistema é responsável por solicitar a injeção e a remoção de falhas junto ao injetor de falhas (SEU Injector), bem como controlar a execução do CUT, através dos sinais *start* e *done*. Enquanto o CUT estiver executando, este controlador fica em estado de espera, até que o controlador de E/S do CUT informe que a execução já terminou.

Nesse momento, o controlador solicita a remoção da falha injetada para reestabelecer o estado correto da memória de configuração. Em seguida, um novo pedido de injeção de falha é realizado, de forma que este ciclo é executado até que o total de falhas desejado seja injetado.

Paralelamente ao ciclo de injeção de falhas, o controlador do sistema monitora o recebimento de comandos do software, que pode cancelar a campanha de injeção de falhas ou reiniciá-la com novos parâmetros, por exemplo. Além disso, o controlador pode periodicamente informar o software sobre o estado atual da campanha, como o percentual de falhas que já foram injetadas ou o término da execução.

O SEU injector é o único componente que não sofreu nenhuma alteração em relação ao apresentado em [Gabriel L. Nazar and Luigi Carro 2012]. O Virtex-5 utilizado nos experimentos possui a *Internal Configuration Access Port* (ICAP), que permite manipular os bits da memória de configuração. Nesse dispositivo, essa memória é dividida em frames cujos bits podem ser acessados através de um esquema de endereçamento conhecido [Xilinx ug191 2012], o que permite o acesso controlado a cada bit de configuração relacionado a qualquer região do dispositivo. A partir dessa possibilidade, é definida uma *Area Under Test* (AUT) onde o CUT deve ser posicionado, através da indicação de restrições de posicionamento para a ferramenta de implementação. O injetor de falhas, então, é construído de forma a inverter apenas os bits responsáveis pela configuração da AUT, através de reconfiguração parcial pela ICAP, o que impede que o próprio injetor ou os demais componentes de controle da

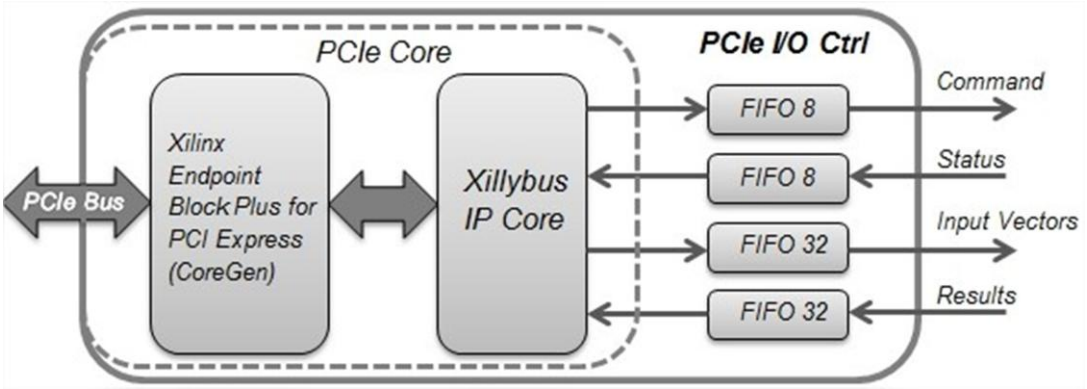


Figura 3. Detalhes do controlador de E/S do PCIe.

plataforma sejam atingidos. Adicionalmente, o injetor é capaz de lidar com erros gerados pela inversão de um único bit, corrigindo-os.

3.3. Controlador de E/S do CUT

O Controlador de E/S do CUT tem a função de receber os vetores de teste e disponibilizá-los ao CUT de forma adequada, bem como enviar os resultados da saída do CUT para o software. Como os vetores de teste são recebidos com tamanho fixo de 32 bits, o usuário da plataforma fica responsável por desenvolver a lógica que realiza esta tarefa, pois é altamente dependente do CUT. Está em estudo a implementação de uma lógica que permita ao usuário apenas informar número de bits da entrada e da saída do CUT, através de constantes, de forma a simplificar o uso da plataforma.

Diferentemente da plataforma desenvolvida em [Gabriel L. Nazar and Luigi Carro 2012], a versão *gold* dos resultados do CUT, que são utilizados para comparação e verificação de erros, é simulada no software, de forma que os resultados enviados pela plataforma não são os resultados dessa comparação, mas tão somente as saídas do CUT. Todo o trabalho de comparação e identificação de erros é feita em software, o que diminui a quantidade necessária de recursos de hardware e flexibiliza a plataforma.

3.4. Software

O componente de software tem algumas funções básicas: controlar o componente hardware, através do envio de comandos e do recebimento periódico do estado do sistema; gerar uma determinada quantidade de vetores de teste para o CUT, definida pelo usuário; receber os resultados da execução do CUT e calcular alguma métrica relevante para sistemas de comunicação de dados que permita avaliar o impacto das falhas injetadas no sistema, como a taxa de erro de bit para cada falha injetada, por exemplo. O número de falhas a serem injetadas, no caso da injeção randômica, pode ser determinado via software.

Para estabelecer uma comunicação com um fluxo contínuo de dados e alta vazão, a interface PCIe utiliza *streams* assíncronos para enviar os vetores de teste e receber os resultados. Adicionalmente, define-se uma thread para cada *stream* de dados, de forma que a vazão depende, na maior parte do tempo, da capacidade de processamento do computador e do sistema implementado no FPGA. A figura 4 mostra a estratégia proposta.

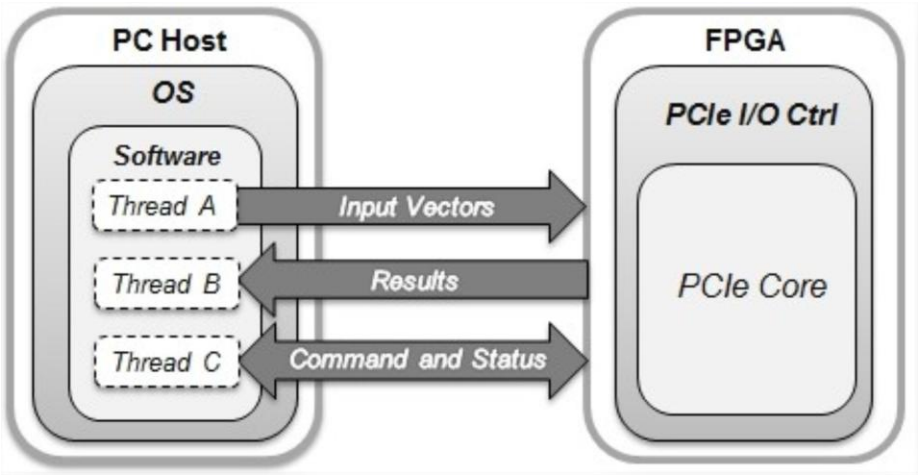


Figura 4. Software *multithreaded* e comunicação assíncrona permitem fluxo contínuo de dados e alta vazão.

4. Resultados Preliminares

Como a implementação do módulo de hardware da plataforma está em estágios avançados, pode-se apresentar alguns resultados preliminares com relação ao uso de recursos do FPGA (LUTs, Flip-Flops e BRAMs). O controlador de E/S do CUT não foi incluído, já que no momento ele possui apenas roteamento de entrada e saída (a lógica de controle e o CUT deste módulo dependem do usuário). A tabela 1 mostra os resultados.

Os resultados mostram que, como era esperado neste momento, o módulo de controle PCIe é disparado o mais custoso em termos de recursos, em especial pelo uso de várias memórias FIFO.

5. Cronograma de Atividades

As tarefas a serem realizadas no Trabalho de Graduação 2 estão enumeradas a seguir. A tabela 2 apresenta o cronograma de atividades.

1. Conclusão da implementação do hardware.
2. Conclusão da implementação do software, acrescentando funções úteis relacionadas à área de comunicação de dados e verificando a eficiência das mesmas.
3. Verificação da eficiência no uso do barramento PCIe, fazendo as devidas alterações, caso necessário.
4. Validação da plataforma utilizando um decodificador Reed-Solomon como estudo de caso.
5. Validação da plataforma com um segundo estudo se caso, a ser definido.
6. Redação da monografia do Trabalho de Graduação 2.
7. Entrega e apresentação do trabalho.

Tabela 1. Custo preliminar da plataforma em recursos do FPGA (XC5VLX110T).

	Quantidade			Ocupação		
	LUT	*FF*	*BRAM*	*LUT*	*FF*	*BRAM*
PCIe I/O Ctrl	2509	3386	10	3%	4%	6%
System Ctrl	514	164	1	0,74%	0,24%	0,68%
Total	3023	3550	11	3,74%	4,24%	6,68%

Tabela 2. Cronograma de atividades para a segunda etapa do trabalho.

	2015						
Tarefa	Jun	Jul	Ago	Set	Out	Nov	Dez
1	X	X					
2	X	X					
3		X					
4			X	X			
5				X	X		
6			X	X	X	X	
7							X

6. Considerações Finais

Este artigo apresentou uma nova plataforma de injeção de falhas em FPGAs para a avaliação do impacto de SEUs, com recursos que facilitam o uso em sistemas de comunicação de dados. Alguns desafios que ainda permanecem, como a conclusão da implementação e a validação do sistema, serão tratados na segunda etapa do trabalho. É importante frisar que a plataforma proposta neste artigo está em um estágio inicial de teste das suas funcionalidades, como a capacidade de comunicação do FPGA com o PC através do barramento PCIe. Portanto, pequenas modificações poderão ocorrer, como a inclusão de novas funcionalidades ou a reformulação de outras. Mesmo que uma grande quantidade de trabalhos desse tipo sejam encontrados na literatura, espera-se demonstrar, na sequência do trabalho, que o conjunto de características proposto pode ser um diferencial na hora de desenvolver e validar técnicas de mitigação de falhas induzidas por SEUs em sistemas de comunicação de dados implementados em FPGAs.

Referências

A. Lesea, S. Drimer, J.J. Fabula, C. Carmichael and P. Alfke (2005). *The rosseta experiment: atmospheric soft error rate testing in differing technology FPGAs*. In IEEE Trans., Device and Materials Reliability, pages 317–328, no. 3, Sept. 2005.

E. Fuller, M. Caffrey, P. Blain, C. Carmichael, N. Khalsa, and A. Salazar (1999). *Radiation test results of the Virtex FPGA and ZBT SRAM for space based reconfigurable computing*. In MAPLD, Proceeding of the Military and Aerospace Programmable Logic Devices International Conference, Laurel, MD, September 1999.

M. Violante, L. Sterpone, M. Ceschia, D. Bortolato, P. Bernardi, M. Sonza Reorda, and A.Paccagnella (2004). *Simulation-Based Analysis of SEU Effects in SRAM-Based FPGAs*. IEEE Trans., Nuclear Science, Vol. 51, no. 6, December 2004.

M. Ceschia, M. Violante, M. Sonza Reorda, A. Paccagnella, P. Bernardi, M. Rebaudengo, D. Bortolato, M. Bellato, P. Zambolin, and A. Candelori (2003). *Identification and Classification of Single-Event Upsets in the Configuration Memory of SRAM-Based FPGAs*. IEEE Trans., Nuclear Science, Vol. 50, no. 6, December 2003.

Sheng Wang, Adrian Evans, Shi-Jie Wen, Rick Wong and GengSheng Chen (2015). *New insights into the impact of SEUs in FPGAs CRAMs*. IEICE, Electronics Express, pages 1–12, Vol. 12, no. 6.

F. L. Kastensmidt, L. Carro, and R. Reis (2006). *Fault-Tolerance Techniques for SRAM-Based FPGAs*. Springer 2006, 1st ed., Dordrecht, The Netherlands.

Uros Legat, Anton Biasizzo and Frank Novak (2012). *SEU Recovery Mechanism for SRAM-Based FPGAs*. IEEE Trans., Nuclear Science, Vol. 59, no. 5, October 2012.

F. Lima, C. Carmichael, J. Fabula, R. Padovani, R. Reis (2001). *A Fault Injection Analysis of Virtex FPGA TMR Design Methodology*. IEEE, European Conference on Radiation and its Effect on Component and Systems, pages 275–282, 2001.

Heather Quinn, Keith Morgan, Paul Graham, Jim Krone, Michael Caffrey, and Kevin Lundgreen (2007). *Domain Crossing Errors: Limitations on Single Device Triple-Modular Redundancy Circuits in Xilinx FPGAs*. IEEE Trans., Nuclear Science, Vol. 54, no. 6, December 2007.

Jean Arlat, Martine Aguera, Louis Amat, Yves Crouzet, Jean-Charles Fabre, Jean-Claude Laprie, Eliane Martins and David Powell (1990). *Fault Injection for Dependability Validation: A Methodology and Some Applications*. IEEE Trans., Software Engineering, Vol. 6, no. 2, February 1990.

L. Sterpone and M. Violante (2005). *Analysis of the Robustness of the TMR Architecture in SRAM-Based FPGAs*. IEEE Trans., Nuclear Science, Vol. 52, no. 5, October 2005.

M. Wirthlin, E. Johnson, N. Rollins, M. Caffrey, and P. Graham (2003). *The Reliability of FPGA Circuit Designs in the Presence of Radiation Induced Configuration Upsets*. IEEE Proc., 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, pages 133–142, 2003.

Heather Quinn, Paul Graham, Keith Morgan, Michael Caffrey, and Jim Krone (2008). *A Test Methodology for Determining Space-Readiness of Xilinx SRAM-based FPGA Designs*. IEEE, AUTOTESTCON 2008, pages 8-11, Salt Lake City, UT, September 2008.

Gabriel L. Nazar and Luigi Carro (2012). *Fast Single-FPGA Fault Injection Platform*. IEEE, International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2012.

Stefano Di Carlo, Paolo Prinetto, Daniele Rolfo, Pascal Trotta (2014). *A Fault Injection Methodology and Infrastructure for Fast Single-Event Upsets Emulation on Xilinx SRAM-based FPGAs*. IEEE, International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2014.

Xilinx (2015). *Virtex-5 FPGA Configuration User Guide* (online). Disponível em http://www.xilinx.com/support/documentation/ug191.pdf.

Matthew Jacobsen, Yoav Freund, Ryan Kastner (2012). *RIFFA: A Reusable Integration Framework for FPGA Accelerators*. IEEE, International Symposium on Field-Programmable Custom Computing Machines, 20th ed., 2012.

Jian Gong, Tao Wang, Jiahua Chen, Haoyang Wu, Fan Ye, Songwu Lu, Jason Cong (2014). *An Efficient and Flexible Host-FPGA PCIe Communication Library*. ACM/SIGDA Proc., international symposium on Field-programmable gate arrays, pages 255–255, New York, NY, USA, 2014.

Kenneth A. LaBel (1996). *Single-Event Effect Criticality Analysis* (online). Disponível em http://radhome.gsfc.nasa.gov/radhome/papers/seecai.htm.

Eli Billauer (2014). *An FPGA IP core for easy DMA over PCIe with Windows and Linux* (online). Disponível em http://xillybus.com.

Xilinx (2015). *Xilinx University Program XUPV5-LX110T Development System* (online). Disponível em http://www.xilinx.com/univ/xupv5-lx110t.htm.

Xilinx (2015). *Xilinx IP Endpoint Block Plus for PCI Express* (online). Disponível em http://www.xilinx.com/support/documentation.