FIRED – Fault Injector for Reconfigurable Embedded Devices

José Luís Nunes[1,2], Tamás Pecserke[3], João Carlos Cunha[1,4], Mário Zenha-Rela[2,4]

[1]Instituto Politécnico de Coimbra, ISEC, DEIS
Coimbra, Portugal
{jnunes, jcunha}@isec.pt

[2]University of Coimbra, Dept. of Informatics Engineering
Coimbra, Portugal
mzrela@dei.uc.pt

[3]Prolan Co.
Budakalász, Hungary
pecserke@prolan.hu

[4]Centre for Informatics and Systems of the University of Coimbra
Coimbra, Portugal

***Abstract*—Reconfigurable embedded devices built on SRAM-based Field Programmable Gate Arrays (FPGA) are being increasingly used in critical embedded applications. However, the susceptibility of such memory cells to Single Event Upsets (SEU) requires the use of fault tolerant designs, for which fault injection is still the most accepted verification technique. This paper describes FIRED, a fault injector targeted at SRAM-based FPGAs for dependability evaluation of critical systems. This tool is able to perform hardware fault injection in real-time, by inserting *bitflips* at the SRAM cells through partial dynamic reconfiguration. These faults may produce errors in the design of the VHDL or Verilog modules deployed in the FPGA. A case study of a fault injection campaign in a PID-based cruise control system is used to evaluate the capabilities of FIRED, namely its capacity of injecting faults while a physical application is being controlled.**

Keywords—dependability, fault injection, embedded systems, FPGA, SEU

I. INTRODUCTION

There is an increasing demand for in-system reconfigurable devices as part of embedded systems, specifically Field Programmable Gate Arrays (FPGA). This need is driven by the ever-strict requirements imposed by aggressive market timings, by the needs of energy-efficient computation, or by the demands for adaptive systems due to the dynamic nature of the deployment environments.

It's of no surprise that the reduced cost and the low power consumption of recent devices have had a strong impact on the use of reconfigurable systems. What started as a minor market, the use of FPGA devices in communication equipment, has now become widely spread all over different fields such as wireless sensor networks [1], automotive [2], industrial control applications [3], space [4], etc.

The possibility of having a hardware system tailored to optimize a specific function, as opposed to a generic processor-based embedded system, has a major effect on device efficiency. The verification and validation of hardware-implemented critical system is also simpler for FPGA-based systems than for processor-based software implemented approaches, due to the needs for certification of not only the application software, but also operating system, CPU, etc. Finally, FPGAs are also hitting the Application Specific Integrated Circuit (ASIC) market, since manufacturers are reducing the gap in both price and performance, which gives ASICS an advantage only when significantly larger quantities and higher speed are required.

Recently some manufacturers have extended the in-system reconfiguration capabilities of FPGA devices to support runtime reconfiguration. *Xilinx* introduced this in *Virtex* device family and named it Partial Dynamic Reconfiguration (PDR). In the Programmable Logic (PL) area of such devices the system designer can define static as well as dynamic regions. As the name states, a static region is permanent and does not change throughout execution time, whereas a dynamic region can be reconfigured at runtime, while the remaining logic of the FPGA continue to run. The use of dynamic reconfigurable systems also allows a reduction in power consumption as, in most systems, not all the modules are needed simultaneously. Therefore, the system designer is able to select a device with a smaller footprint, with lower power consumption, and switch-on and -off hardware modules as needed. Being performed at runtime, the reconfiguration delay is critical, so these systems rely on fast access memory technologies, namely SRAM, as opposed to Flash memory, for storing the implemented hardware description – the Configuration Memory (CM).

However, the drawback of SRAM memory cells lies in its susceptibility to interferences from heavy ions or electromagnetic radiation, causing Single Event Upsets (SEU). Hence the design of dependable systems requires the use of fault tolerant mechanisms, such as Scrubbing, Triple Module Redundancy (TMR), Error Detection and Correction Codes

978-1-4673-9376-8/15 $31.00 © 2015 IEEE
DOI 10.1109/PRDC.2015.43

IEEE computer society

(EDAC), etc. In critical systems, such mechanisms need to be verified to evaluate the system fault tolerant capabilities. Fault injection (FI) is the most accepted technique for experimental dependability evaluation of critical systems.

It is possible to find in the literature a few examples of FPGA-based fault injectors, but they are mainly targeted at testing the resilience of ASIC designs before the production phase [5, 6]. The approach followed in our study assumes that the system is to be deployed in an FPGA device, thus the need to evaluate its resilience to faults affecting the most sensitive underlying hardware: the SRAM memory cells.

This paper presents FIRED, a fault injection tool for dependability evaluation of SRAM FPGA-based embedded systems, whose main capabilities are i) to inject faults at SRAM cells of the configuration memory of an FPGA; ii) to inject faults in a running system without the need to halt and resume; iii) to assess the dependability properties of SRAM-based FPGA embedded systems; and iv) to inject faults in a production system without any additional artifact.

In the next section we look at some of the available FPGA fault injectors and their applications. In section three, we introduce the extensive fault model supported by FIRED, and in section four we describe its architecture and the interaction between its major components. In section five we present a case study of a fault injection campaign, targeting an FPGA running a VHDL model of a PID cruise control system. In section six we finish the description of FIRED by analyzing its fault injection properties. Finally, in section seven, we conclude the paper presenting future developments of this tool.

II. State of the Art

To assess the dependability of an embedded system and the fault tolerant mechanisms in place, a very common approach is to inject faults or errors in the system and then monitor its behavior. In SRAM FPGA-based systems, these errors usually corrupt the implemented user design stored in configuration memory or, with lower probability due to a smaller area occupation, the system state. Depending on the technology used for the injection of errors in the FPGA configuration memory, the methods found in the literature are grouped in hardware-based FI and software-based FI [7]. The high maintenance costs of hardware-based FI facilities, capable of simulating the radiation effects in FPGAs through heavy-ion and proton beaming, restrict its use to a few manufacturing companies and research institutions. A cheaper alternative is to use a laser beam to simulate the radiation effects, but some preparation of the device surface is needed before the FI campaign. To overcome these high costs, some researchers took advantage of Xilinx FPGA Partial Dynamic Reconfiguration properties to simulate the effects of SEUs in their configuration memory, assuming a *bitflip* fault model. This form of software-based FI, when compared to physical methods, has higher precision, better controllability and lower cost. The major drawbacks are the representativeness of the injected faults compared with the radiation-induced SEUs and also the higher intrusiveness.

There are only a few FPGA software-based fault injectors, resulting from academia research projects. Since these tools are targeted at specific families of FPGAs, their use on different or updated versions of FPGAs implies major changes in the internal design. All of these fault injectors use some form of *bitstream* instrumentation, either before system startup (offline) or during system execution (online).

The FLIPPER [5] fault injectors, developed by the Italian National Institute for Astrophysics, under a contract with ESA, are composed by a hardware platform (based on Xilinx XQR2V6000 device) and a software application running on a PC. Targeted at Xilinx Virtex-2 (FLIPPER 2004-08) and recently Virtex-4 (FLIPPER2 2010-12), they use partial reconfiguration to inject single and multiple *bitflips* in configuration memory. Both FLIPPERs mimic the radiation experiments and allow the identification of design sensitive bits and the evaluation of SEU sensitivity in an FPGA implementation of an ASIC prototype. The injection can be random, sequential, or user defined and is performed in accumulation (adding up the effects).

The FT-UNSHADES [6] fault injector, developed by the University of Seville, with the support of the European Space Agency (ESA), was created to study the effects of radiation-induced faults in an ASIC, assuming a *bitflip* fault model. This system is able to inject *bitflips* in Flip-Flops through the use of dynamic partial reconfiguration of a *Xilinx* FPGA. The FPGA implements both the "faulty system" and the "fault-free system" and compares their output to inform the host computer that a failure occurred. To inject *bitflip* faults, the host just has to exchange a few thousand bits (two FPGA frames) with the FPGA configuration memory.

Gokhale [8] developed a SEU Simulator for assessing dependability through fault injection. The simulator uses PDR to corrupt the FPGA frames while the system is running. It is composed of three FPGAs, two running the user circuit in parallel (golden design and design under test) and a third one responsible for real-time output comparison. The simulator can be used to classify the user design configuration bits as being sensitive or non-sensitive. It is also capable of differentiating the sensitive bits from persistent, associated with system state and control function, or non-persistent, which could be recovered by just restoring the original bit value.

The goal of FIRED is not to evaluate dependability properties of ASIC designs prior to production, as with the FLIPPERs and FT-UNSHADES, but to assess the properties of FPGA-based production systems. It also tries to improve the resource usage and the performance of the approach followed by Gokhale, which uses multiple FPGAs to isolate the interference between each system component, leading to higher reconfiguration delays due to the use of slower external interfaces.

III. Fault Model

FIRED was created to investigate the feasibility of using COTS SRAM-based FPGA devices in harsh environments. The use of generic devices instead of radiation-hardened versions and the fast access memory technology used as configuration memory, makes them more prone to upsets [9]. The effects of these upsets include the corruption of the system state, which affect both ASICS and FPGA devices, and

changes in the implemented design, which are specific to FPGAs, as they rely in SRAM cell for configuration memory. There has been a strong effort on evaluating the devised fault tolerant approaches to recover the system state, but only recently the researchers have focused on improving the resilience of reconfigurable devices, through configuration memory protection [10, 11].

A. Fault Type

SRAM-based reconfigurable devices suffer from the same sensitiveness to radiation-induced faults as the SRAM memory used in non-reconfigurable systems. This can be predominantly seen in areas were the FPGAs are experiencing an increased usage, namely space applications. The radiation effects in SRAM memory cells are manifested as SEUs and have been extensively reviewed in the literature [12]. Experimental studies, like Rosetta [13] from Xilinx, where all sorts of FPGAs are exposed to radiation, have shown that most upsets take the form of *bitflips* in memory cells. Similar results were gathered by Gunneflo through hardware-based FPGA fault injection campaigns [14]. The incidence of radiation in these devices may produce multiple *bitflip*, in which case it is common to observe a circular pattern around the hit point [15].

Therefore, to realistically mimic the radiation-induced faults in SRAM-based FPGAs, FIRED is capable of injecting both single and multiple adjacent *bitflip* errors, as presented in Fig. 1. Besides this, FIRED is also able to inject *suck-at-0* and *stuck-at-1* faults, i.e., all the bits embraced by the injection mask take the value 0 or 1.

B. Fault Location

All the errors injected by FIRED are targeted at the FPGA configuration memory, specifically the location of the Device Under Test (DUT) hardware implementation, as defined by the manufacturer design tools. Although there is no detailed information available from FPGA manufactures about the mapping between the user design and the contents of the CM in the form of a *bitstream*, they provide some information about the structure and basic elements of the device: Configuration Logic Block (CLB), Digital Signal Processor (DSP), Input/output Block (IOB), etc.

Transient faults that affect these CM cells introduce permanent changes in the DUT design that will remain until the contents of the affected cells are restored. This usually happens when the device is power cycled, at the startup phase.

In the FIRED tool, the location of a single *bitflip* fault is simply defined by an address location inside of the FPGA configuration memory. For multiple *bitflips,* an additional 8-by-8 bits mask has to be specified, as shown in Fig. 1.

C. Trigger

In order to emulate radiation bursts, resulting in transient faults with cumulative effects on the FPGA, FIRED is capable of injecting multiple faults at distinct instants throughout a single experiment. In this case the DUT is not reset after each injected fault, and the permanent effects of the upcoming faults are added to the effects of the ones already injected, which may eventually be perceived at the DUT interface as a device failure.

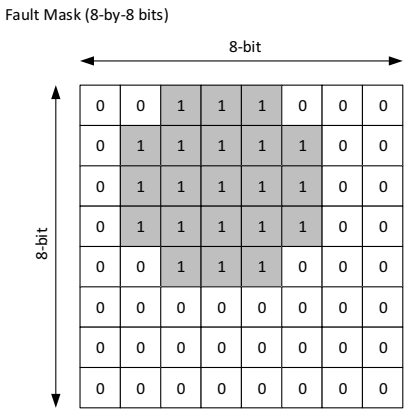

Fig. 1. Mutliple bitflip fault injection pattern

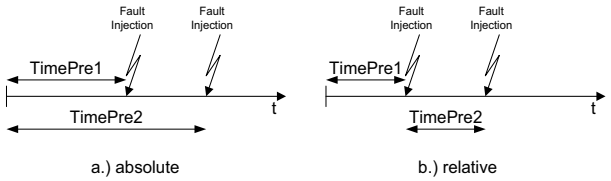

Fig. 2. Specification of a *time-triggered* fault injection: a) absolute time, b) relative time.

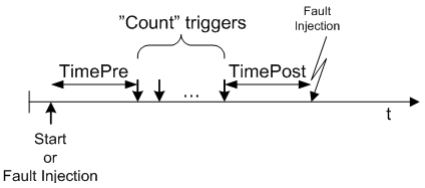

Fig. 3. Specification of an *event-triggered* fault injection.

The fault injection trigger was designed to be highly configurable, allowing the person conducting the FI experiments to precisely define the injection instants.

Triggers can be specified based on time (*time-triggered*) or in an external event (*event-triggered*). In the former case, the injection time of each fault (*TimePre*) may be specified as an absolute value, with respect to the beginning of the experiment, or relative, to the previous injected fault. In Fig. 2, a temporal diagram is depicted with the user-defined parameters for a time-triggered injection run. In the second case, an external trigger to the fault injector may be used. Besides the system settling time (*TimePre)*, also used in the time-triggered approach, the user must also specify i) the number of events (*Count triggers*) after which the fault should be injected and 2) a post injection delay (*TimePost*) for error detection, as seen in Fig. 3.

IV. Fault Injector Overview

Besides injecting faults in the SRAM cells of the target FPGA, FIRED is able to monitor the behavior of the DUT in runtime while trying to reduce its intrusiveness to a minimum, so that the gathered results are still representative of a

production system. This led to the division of the FI tool in two major components: *Injection Runtime Controller* (IRC) and *Experiment Management Environment* (EME). As it can be seen in shaded areas of Fig. 4, the IRC runs side-by-side with the DUT and has direct access to the configuration memory where the DUT implementation is stored. The EME runs in a regular PC and holds a connection to a database engine to store experiment data (not only the experiment descriptions, but also all the collected experimental results).

The parallel nature of the FPGA and the PDR properties of the device in use, allow the isolation between the IRC and the DUT, and let the IRC to precisely target the DUT configuration memory region. By running the EME in the PC, it is easier to define and faster to run the fault injection campaigns, by relieving the IRC from some intensive processing tasks of mapping the fault definition with the targeted bits. The two components communicate using a proprietary protocol, which runs over a TCP/IP stack.

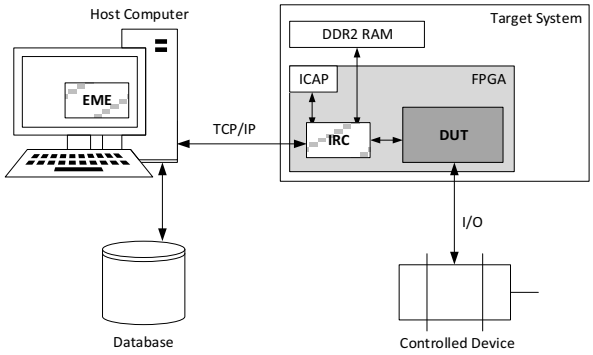

Fig. 4. FPGA fault injector architecture

A. Injection Runtime Controller (IRC)

The *Injection Runtime Controller* is responsible for the effective injection of faults in the DUT configuration memory. In order to inject faults in such memory cells, it needs access to one of the available configuration memory interfaces: SelectMAP, Joint Test Action Group (JTAG) or Internal Configuration Access Port (ICAP). The strict time requirements of critical real-time systems led us to select ICAP. Being an internal port, it is only available to hard-wired PowerPC cores or to soft-cores, such as *Microblaze*.

The current version of IRC runs in a *Microblaze* soft-core; it receives instructions from EME using a push/pull protocol and sends asynchronous events back; it also includes I/O logging facilities. To reduce the interference of the EME to IRC communication channel in the experiment outcome, the IRC includes a small internal cache memory for storing the description of the upcoming faults in the current experiment, and an external DDR2 RAM memory to locally log the I/O data and send it back only when requested for. Figure 5 presents the main interactions between the modules of FIRED.

The actual fault injection is performed using PDR though ICAP, which allows the IRC to change only a few configuration memory frames in a *Read-Modify-Write* (RMW) approach.

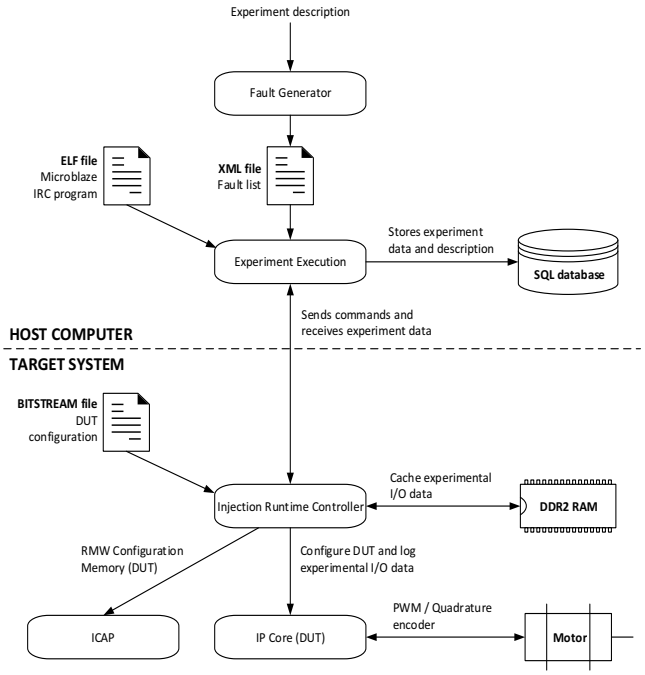

Fig. 5. FPGA fault injector module interaction

The IRC also uses the ICAP interface to restore the DUT *bitstream* after each Injection Run (IR) in an experiment, in order to start from a clean state.

B. Experiment Management Environment (EME)

The *Experiment Management Environment* is the fault injector user interface and is executed in a PC. It is responsible for the experiment definition, experiment execution and analysis of the results. Currently, all these functionalities are implemented as command line tools that are called from a script to automate the tasks.

1) Fault Generator

The Fault Generator (FG) is responsible for generating an XML file that describes the experiment according to the information provided by the user. The input data is validated against a model of the FPGA device in use, included in the FG. This tool will generate random values within limits, specified by the user. The accepted parameters are:

- Pre time range;
- Post time range;
- Injection run duration;
- Frame address range (major address, minor address, half, column, and type);
- Frame offset range;
- Fault type (*stuck-at-0*, *stuck-at-1* or *bitflip*);
- Number of affected bits per injected fault; and
- 8-by-8 bits mask.

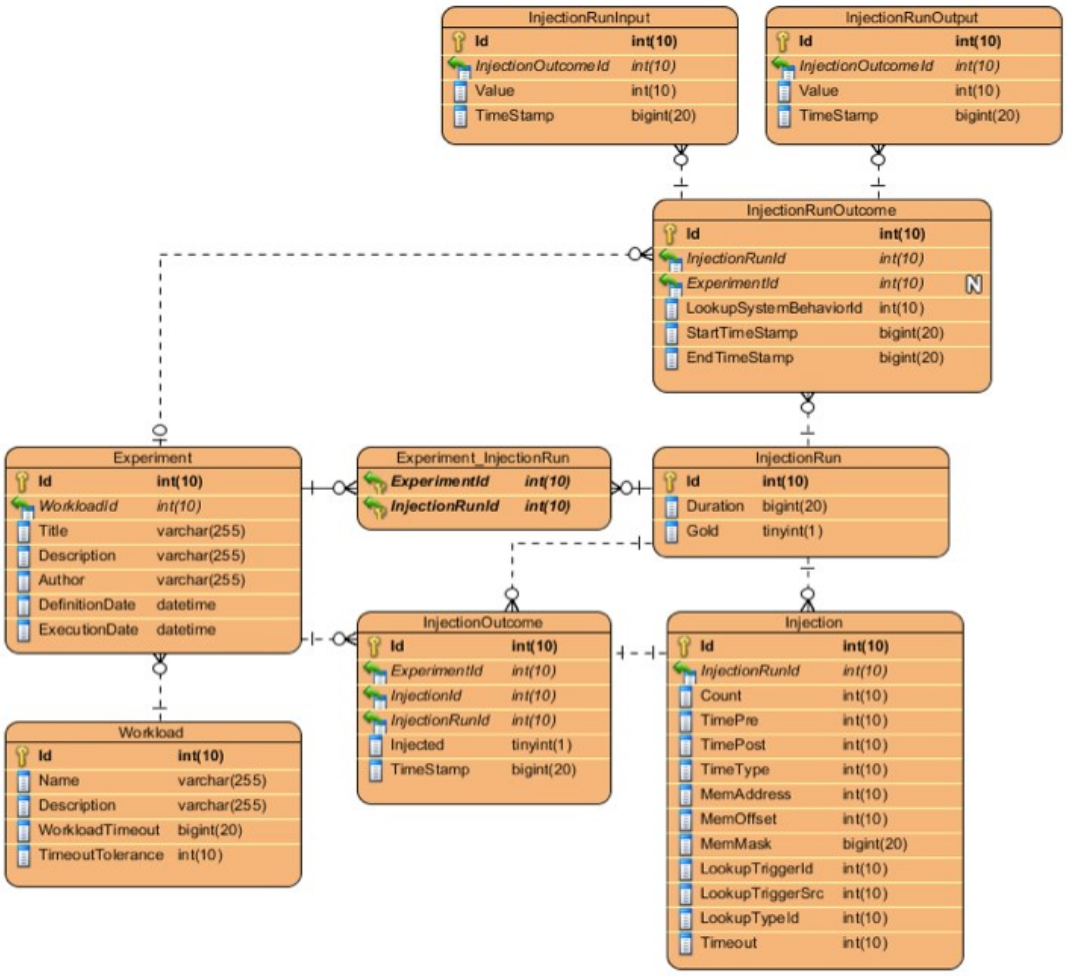

Fig. 6. FIRED external database structure

The generated XML file includes all the information needed by the IRC to actually perform the fault injection. If needed, the user can tweak this file to select specific injection runs or faults, in order to crosscheck previous results.

1) Experiment Execution

The EME is also responsible for the management of the experiments. It uses an SQL relational database to store all the experiment descriptions together with the collected experimental outcomes.

In the first stage, it reads and uploads an XML file with the experiment description data to the database. Secondly, it enters the setup phase where it connects to the IRC and issues a *reset* command, followed by a sequence of *write* commands which include the first faults to be injected in the current injection run, and then sends a *run* command to instruct the IRC to effectively start the injection run. Each time a fault is actually injected, the IRC notifies the EME, by sending an *event* command with the injection result. In this case, an entry in the IRC fault buffer is freed and the EME is allowed to send another fault for later injection. As soon as the EME detects the end of the injection run, triggered by a timer, it sends a *stop* to halt the DUT execution. After the execution of each injection run, the EME issues a sequence of *read* commands to the IRC to request the experiment I/O results. This data is then uploaded to the database for later analysis. These steps are repeated until all the injection runs of a single experiment have been performed.

In order to analyze the outcome of the fault injection campaign, the first injection run of each experiment is always a *goldrun*. This *goldrun* is identical to the other injection runs of the experiment, with the exception that it does not contain any faults to be injected. Its output will be compared with the output generated during other injection runs.

2) Result Analysis

By considering the DUT as a black box with inputs and outputs, containing some logic and internal state, the solution devised to analyze its sensitivity to the injected faults was to compare its outputs against a fault-free run. In this context, the *goldrun* reproduces the normal behavior of the DUT, by means of the collected I/O data.

Therefore, if an injected fault produces an error in the DUT and if it eventually reaches the boundary of the system, it is perceived as an erroneous output. In this situation the functionality of the DUT has been compromised and thus is considered as a system failure.

By using a tool to extract the data from the database and a *Matlab* script, each injection run is compared against the *goldrun*, and any deviation from the expected output is signaled. The final results of a fault injection campaign may contain information related to the failure rate, globally or per FPGA basic component; the error latency; the location of the faults that produced the failures; etc. From the collected data it is possible to extract system metrics (e.g. reliability, error latency, etc.) that characterize its dependability.

It is worth mentioning that this comparison with the golden run is only valid if the system is deterministic. For example, in the case of a controller interacting with a physical system, the the correct outputs of the DUT must be asserted with a reasonability analysis, e.g. by checking if they are inside some bounds.

C. Database

The diagram in Fig. 6 shows the data model of the FIRED database. The *experiment* entity identifies one fault injection campaign. Each experiment is composed by at least one *injection run*, which may have any number of *injection*s, each of which represents a single fault. For example, when performing a fault injection campaign in a PID controller, an injection run is a single run of the controller, which begins with the *start* and ends with the *stop* commands, sent by EME to IRC. Each experiment has a single *workload*, and all the experimental data collected during the experiments is held by the entities *injection run input* and *injection run output*.

V. Case Study

To evaluate the FI properties of FIRED, we tested the tool in a VHDL implementation of PID-based cruise control system. This controller is used to command the speed of a DC motor while monitoring the shaft rotation. Besides the controller itself, other modules (like quadrature decoder, PWM signal generation and clock generator) were implemented in the same FPGA.

The overall system is implemented in a Xilinx University Program development board (XUPV5), which includes a Xilinx Virtex-5 VLX110 FPGA, as an IP Core. This core interfaces with the IRC module, running bare-metal on a *Microblaze* processor, through a Processor Local Bus (PLC). The communication between these subsystems is accomplished using memory-mapped registers.

A. Device Under Test

The PID controller is implemented in a Reconfigurable Partition of the FPGA and has a fixed interface. This allows changing the DUT, without the need to adapt the other components that support the FI. The controller can be seen as a black box with inputs (clock, enable, reset and 32-bit signal input) and outputs (32-bit signal output). In the specific case of the cruise control system, depicted in Fig. 7, the controller has a 16-bit *reference* (R) and 16-bit *feedback* (Y) inputs, and one 16-bit *control signal* (U) output. The FIRED logging module monitors and registers all these signals throughout the experiments.

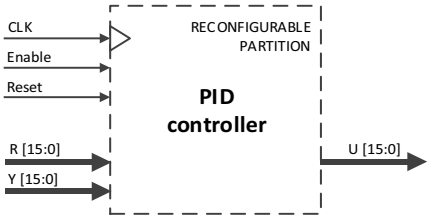

Fig. 7. PID-based controller module (DUT)

For this type of systems, the PI variant of the PID controller is known to be well suited for the task [16]. The controller in use has a proportional (K_p) and an integral (K_i) constant of 100 and 2, respectively. It also has an anti-windup feature to avoid the saturation of the integral part, by placing an upper limit to this controller component. The sampling time of the controller is set at 10ms.

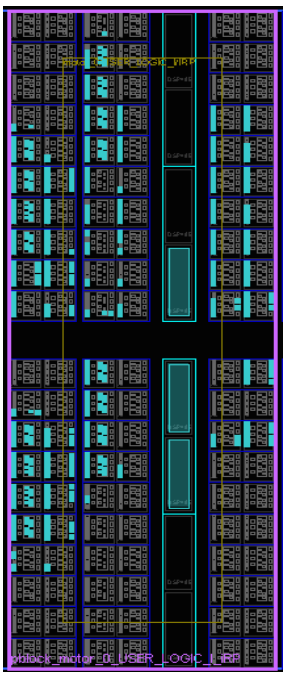

Fig. 8. FPGA slice occupancy of the PID controller.

A graphical view of the PID controller used logic resources on the RP, is depicted in Fig. 8. To avoid, as much as possible, injecting faults in unused location, the RP region was defined as the smallest area of the FPGA where the PID controller could fit in. In this case, only four major columns, spanning a single row, were needed – three CLB columns (with block coordinates from X28Y140 to X33Y159) and one DSP column (with block coordinates from X0Y56 to X0Y63). The configuration memory space occupied by the controller is 136 frames (22304 bytes)

This specific device, Xilinx Virtex-5 VLX110 [18], is divided in two *halves*, each with four *rows* numbered between 0 and 3, starting form the center. Each row is composed by a fixed number of columns, identified by a *major column number*. Each one of these includes only a single type of an FPGA basic element (CLB, DSP, IOB, etc.), and is subdivided into a variable number of *minor columns* elements (CLB and DSP blocks are composed by 36 and 28 frames, respectively). The first 25 frames map routing resources of the FPGA, while the remaining ones map specific block type elements. The frame is the smallest addressing element of an FPGA and, in this specific device, has a size of 1312 bits.

```
Area Group Information
----------------------

Area Group "pblock_motor_0_USER_LOGIC_I_iRP"
  No COMPRESSION specified for Area Group
  RANGE: DSP48_X0Y56:DSP48_X0Y63
  RANGE: SLICE_X28Y140:SLICE_X33Y159
  Slice Logic Utilization:
    Number of Slice Registers:              45 out of    480    9%
    Number of Slice LUTs:                  307 out of    480   63%
      Number used as logic:                307
  Slice Logic Distribution:
    Number of occupied Slices:              92 out of    120   76%
    Number of LUT Flip Flop pairs used:    308
      Number with an unused Flip Flop:     263 out of    308   85%
      Number with an unused LUT:             0 out of    308    0%
      Number of fully used LUT-FF pairs:    45 out of    308   14%
  Number of DSP48E:                          3 out of      8   37%
```

Fig. 9. Xilinx tools detailed information about the FPGA resources allocated to the PID controller.

The detailed information about the number of used resources by the PID controller, from the Xilinx tools, is presented in Fig. 9.

B. Simulator vs Real System

A simulator of a DC motor was also implemented in VHDL and is located inside the FPGA (but obviously outside the reconfigurable partition). This approach allows implementing everything as a System-On-a-Chip (SoC), which doesn't need any external component to run the FI experiments. This is of tremendous value in the first FI experiments, not only to get some insight of the controller behavior, but also because it is possible to speedup the simulation by reprogramming the clock module that feeds the CLK signal to the simulator. In the current setup the simulator is fed with a 100KHz clock frequency.

Some components, namely the quadrature decoder and the PWM signal generator, are not needed when interfacing the controller with the simulator. These components are only used when a physical system is connected to the *devboard*. In the this case, the interface with the real system is accomplished by using one digital output pin, for the PWM signal, and two digital input pins, for both the Channels A and B of the quadrature decoder.

C. Experiments

To better evaluate the fault injection capabilities and the possible undesired interferences of the FIRED tool in the system, we designed several experiments that encompass the most frequent fault models. The first five experiments were run against the VHDL simulator, while in the last one we connected the DUT to a real system (it is effectively controlling the shaft rotational velocity of a mobile robot wheel).

The fault list was created through the fault generator, which was instructed to randomly generate one hundred injection runs, per experiment. The total duration of each injection run is 12s and the controller was set to follow a square wave reference signal with 10s period. The faults are injected only when the controller reaches the high-speed section of the reference wave, and after a one second settling phase. In multiple-fault experiments, the successive faults, with sum-up effects, are injected in intervals between 400ms to 500ms.

As described in Section IV, each experiment contains an additional injection run, a *goldrun* (a fault-free IR), which is used for controller failure detection by output data cross comparison.

Due to the lack of information from the manufacturer about the mapping between the VHDL code and the contents of the configuration memory, the fault location was randomly selected among the RP area of the PID controller.

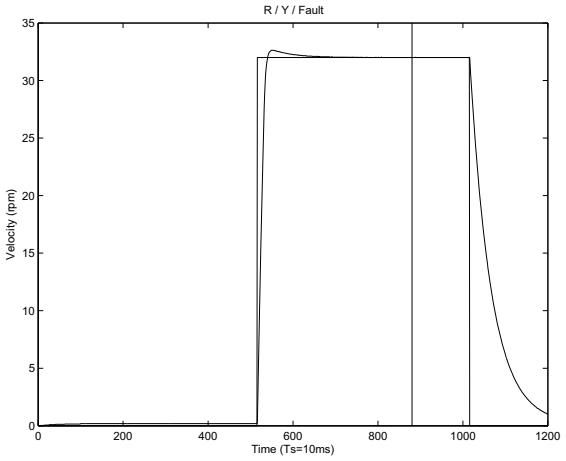

Fig. 10. *Failure-free* IR reference – R (square wave) and output – Y (curved line) signals, with 1 injected fault (vertical line).

1) Single-fault – bitflip (one bit)

For the first experiment, we inject one *bitflip* fault in each injection run, by toggling the logic value of an isolated configuration memory bit. In Fig. 10 we can see an extracted IR from this experiment, where a vertical line marks the fault injection time. In this case, no failure has been observed. However, for the 100 injection runs, 13 resulted in a failure, meaning that the output of the controller differed from the *goldrun*. Table I resumes the results from this and the following experiments.

It is worth noting that the images in Fig.10 and following represent the reference (R) and output signal (Y) from the controlled process, since it represents the behavior of the whole system as could be seen by an observer. However, our tool is currently detecting deviations from the control signal (U), which eventually affects the controlled process.

2) Single-fault – stuck-at (one bit)

We define a *stuck-at* fault that affects the configuration memory, as a fault that forces a specific bit to logic value of one (*stuck-at-1*) or zero (*stuck-at-0*).

Two identical experiments were run using the same fault list that was used in 1). In the first experiment we injected *stuck-at-0* faults, while on the second one we replaced them by *stuck-at-1* faults. Respectively 8% and 5% of the faults in the experiments resulted in failures.

3) Single-fault – bitflip (multiple bits)

In this this experiment we used a fault list identical to the one used in experiment 1), kept the same injection time and location, but replaced the single-bit mask by a 3-by-3 square mask. Thus, each injected fault toggles nine adjacent configuration memory bits. As expected, the results are more severe, as 19% of the injection runs resulted in failures.

4) Repeated-fault – bitflip (single bit, at distinct locations)

In this experiment we inject a total of 10 faults per injection run, in accumulation (sum-up effects). Each of these faults is injected at randomly selected locations and toggles a single configuration memory bit. By the end of the experiment, 1000 faults were injected and 31% of the injection runs (each with 10 faults) resulted in failures. Figure 11 shows the result of one injection run which caused a system failure.

5) Intermittent fault – bitflip (one bit, at the same location)

The aging of the device components, the stress of the materials and the manufacturing defects (e.g. fractures in BGA solder joints used in FPGAs) are known [19, 20] to produce *intermittent faults* in electronics devices.

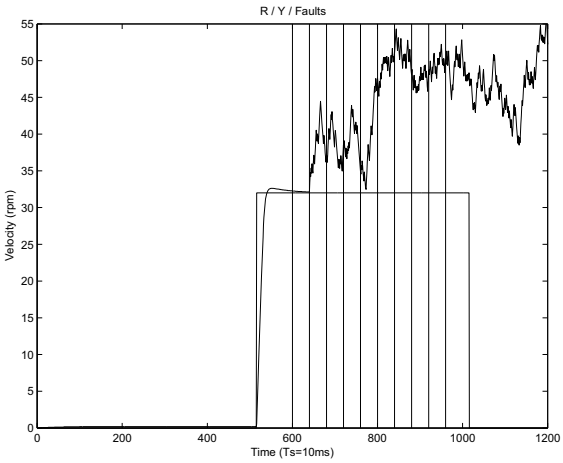

Fig. 11. *Faulty* IR reference – R (square wave) and output – Y (curved and then noisy) signals, with 10 injected faults (vertical lines).

We define an *intermittent fault* in an FPGA configuration memory cell as a fault that affects a fixed memory location, at distinct instants in time, with varying activation times. The activation time is the duration of the fault effects, every time it is triggered.

In this experiment we used a fault list identical to the one used in experiment 1), kept the same fault location, and triggered the fault 10 times. Each trigger is separated by a minimum of 400ms and the activation time is randomly selected, between 10ms and 100ms. This experiment end-up with 14% of failures.

6) Single-fault – bitflip (one bit), real system

To test FIRED with a real system, we reused the fault list of experiment 1). The reference (R) and output of the physical DC motor (Y), are depicted in Fig. 12. The observed fluctuation of Y is perfectly normal in such physical system. In this case, FIRED did not interfere with the process control. Due to the non-deterministic nature of a physical system, there was no comparison of the outputs of these tests with the *goldrun*.

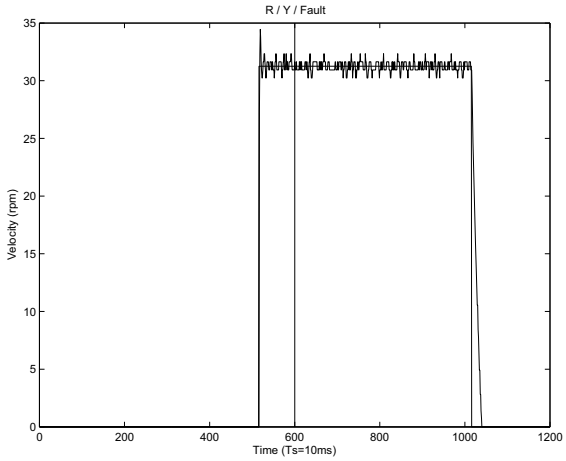

Fig. 12. *Failure-free* IR reference – R (square wave) and output – Y (noisy) signals from the real system, with 1 injected fault (vertical line).

D. Results

The collected data of each injection run was compared against the *goldrun* to detect any deviation of the controller output from the expected behavior. The results of the fault injection experiments 1 to 5 are presented in Table I.

TABLE I. PERCENTAGE OF FAULTY INJECTION RUNS

ID	FAULT	IR[a]	FAULT[b]	BIT[c]	FAULTY[d]
1	*Single Bitflip*	100	1	1	13%
2 i)	*Stuck-at-0*	100	1	1	8%
2 ii)	*Stuck-at-1*	100	1	1	5%
3	*Multiple bits*	100	1	9	19%
4	*Repeated faults*	100	10	1	31%
5	*Intermittent*	100	1	1	14%

[a.] Injection Runs per Experiment

[b.] Faults per Injection Run

[c.] Bits per Fault

[d.] Percentage of faulty Injection Runs

In multiple-fault IR, every time a failure was detected, additional injection runs were executed to identify which of the fault(s) from that particular injection run caused the failure. This way it was possible to get a map of sensitive configuration memory cells.

1) Sensitive bits

The results obtained allowed us to classify the sensitivity of the CM cells to SEUs, i.e. the percentage of *bits* that produced system failures if affected by a fault. Due to the huge amount of CM cells, most of them related to routing and overly unused, it is not practical to test all the bits of a design. The results presented in Table II were gathered from the fourth experiment.

TABLE II. PERCENTAGE OF EFFECTIVE FAULTS PER LOCATION TYPE

ID	MAJOR COLUMNS							
	CLB (17)		*CLB (18)*		*DSP (19)*		*CLB (20)*	
	Route[a]	**Spec.**[b]	**Route**	**Spec.**	**Route**	**Spec.**	**Route**	**Spec.**
4	3,59%	3,33%	3,11%	0,00%	3,74%	2,70%	4,50%	0,00%

[a.] Interconnecting

[b.] Specific to column type

These results show that from a universe of 1000 injected faults, only 3,1% manifested as failures of the controller. This result is inline with the device manufacturer information, which states that a huge amount of resources in the FPGA are used for routing, and that a significant amount of these are unused [11], even in designs that occupy a high percentage of FPGA logic resources (~85%).

2) Accuracy of the fault injector tool

The analysis of the single-fault injection runs showed that most of the injected faults had a small latency, of around 80,4ms. The minimum latency observed was 11,9ms.

A couple of failure and failure-free injection runs where also analyzed to get some insight of FIRED repeatability and system interference. In all these situations, there was a perfect match between the *goldrun* and each injection run, for the failure-free case, and between each single-fault IR and the original 10-fault IR, for the other. This proved that the tool is able to repeat experiments with consistent results, and has no interference in the system execution.

VI. ANALYSIS OF FIRED

There are many different properties used to characterize the fault injection techniques [17], and specifically fault injectors. In this section we analyze the properties of FIRED, based on the results obtained from the previous case study.

A. Reachability

This property stands for the ability to reach all possible fault locations. The current version of the FPGA fault injector is able to inject faults in the configuration memory of the target device. This way it is possible to simulate the effects of SEUs in the memory cells of reconfigurable devices, which stand for the implemented logic. It is however unable to inject faults in the system state, as for this it needs to instrument the target device to add hookups.

B. Controllability

FIRED can inject faults in the configuration memory with a high precision, both in space and time.

By using the ICAP interface it can reach all the configuration memory cells of the target device. Due to the addressing mode used by Xilinx, the minimum readable/writable unit of the configuration memory is one frame. The used approach of *Read-Modify-Write* allows the injection of faults in each single bit location. The overhead in time of the RMW approach is approximately 50µs for a single frame.

C. Repeatability

By running the same fault injection run it is expected that the results be the same.

The experiments performed with FIRED have shown a perfect match of the target outputs (see Fig. 10 and 11). Also, when a single-fault injection run, which resulted in a failure, was repeated several times, the results were consistent.

D. Reproducibility

From the experiments conducted the results were statistically coherent throughout experiments for the specific case of the PID controller.

E. Intrusiveness

At the moment no efforts have been done to reduce the spatial overhead of the fault injector. The processor-based IRC could be substituted, with a reduction in the overall flexibility of FIRED, by a finite state-machine with a smaller footprint, in terms of used PL cells. Regarding the FPGA area used by the DUT, there isn't, however, any spatial intrusiveness.

Regarding time intrusiveness, experiments have shown that the impact of the fault injector in the execution time of the target system is negligible due to the parallel nature of the FPGAs. The only drawback of the RMW approach is the need to share the same bus with target implementations that take advantage of LUTs as distributed memory, which may impose small delays.

F. Flexibility

To isolate the target device from the IRC and allow the partial dynamic reconfiguration, the DUT has been confined to a *reconfigurable partition*. In this case, the only fixed part is the interface with the other modules, which support the fault injection, and the interface of the DUT with the real world/simulator. Due to this, the fault injector can be easily used with other target devices that have a similar interface: *clock*, *reset*, *enable*, *input* (32 bits), *output* (32 bits).

G. Efficiency

The use of the ICAP interface to access the CM allows a huge improvement in the overall FI campaign execution times, when compared to boundary scan fault injection approaches [21].

The FIRED supporting components, described in Section IV, reduce the effort needed to perform a fault injection campaign, relieving the user conducting the experiments from those tasks.

H. Observability

Without instrumenting the target device, and with the sparse information available about the mapping between the

bitstream and the actual components of the target, the present system is only capable o monitoring the target interfaces.

The logging capacity of FIRED is constrained by the size of the external memory available at the *devboard* (used to store each injection run I/O data), while the logging frequency, specified by the user, is only limited by the speed of the Microblaze processor.

VII. Conclusions

FIRED fault injector was created to test the dependability properties of SRAM-based FPGA embedded systems. The target is implemented in a reconfigurable partition of the FPGA programmable logic area, while the parallel nature of the FPGA allows the inclusion of specific components to support the fault injection campaign. Presently the system is implemented in a *Virtex-5* device and the fault injection is supported by a soft-core processor (*Xilinx Microblaze*), as the *devboard* in use does not include an embedded hard-wired processor. To port this fault injector to other architectures, it is necessary to add the new device architecture description to the fault generator and result analysis tools, and synthetize the fault injector and controller *bitstreams*.

The evaluation of this fault injector using a simulation of a physical system gave us valuable information about the properties of the tool and its capabilities. By demonstrating low intrusiveness in the system execution time, it is possible to use it online, with physical systems, thus avoiding the need to halt the system to inject each fault. The use of dynamic partial reconfiguration for fault injection makes it feasible to use the production system without any additional artifact to support the fault injection. It has limited effectiveness in exercising fault tolerant mechanisms due to the lack of information about the mapping between the *bitstream* and the VHDL representation of the DUT. The reachability is limited to the configuration memory cells, whereas we experienced a high controllability, repeatability and reproducibility. The modularity of the FI and the isolation of the controller in a RP also improve its efficiency and flexibility.

In the future there are plans to use the new *Zynq devboard*, to take advantage of the hard-wired ARM processor and thus reduce, not only the spatial overhead, but also the duration of the fault injection campaign. In the first case, by removing all the logic that support the fault injection, with the exception of the DUT and the logging module, and in the second case by taking advantage of a faster dual-core A7 processor which is able to dynamically reprogram the PL area through the Parallel Configuration Access Port (PCAP). We also plan on using other controllers as test cases for better assessment of FIRED.

Acknowledgment

This work has been partially supported by CECRIS (http://www.cecris-project.eu, FP7-PEOPLE-2012-IAPP).

References

[1] R. Garcia, A. Gordon-Ross, and A.D. George, Exploiting Partially Reconfigurable FPGAs for Situation-Based Reconfiguration in Wireless Sensor Networks. 17th IEEE Symposium on Field Programmable Custom Computing Machines, 2009.

[2] J. Yu, and B.M. Wilamowski, Recent advances in in-vehicle embedded systems. 37th Annual Conference on IEEE Industrial Electronics Society, pp. 4623-4625, 2011.

[3] E. Monmasson, L. Idkhajine, M. N. Cirstea, I. Bahri, A. Tisan, and M. W. Naouar, FPGAs in Industrial Control Applications. IEEE Transactions on Industrial Informatics, vol. 7, no. 2, pp. 224-243, May 2011.

[4] J.J. Wang, R.B. Katz, J.S. Sun, B.E. Cronquist, J.L. McCollum, T.M. Speers, and W.C. Plants, SRAM based re-programmable FPGA for space applications. IEEE Transactions on Nuclear Science, vol. 46, no. 6, pp. 1728-1735, 2002.

[5] M. Alderighi, F. Casini, M. Citterio, S. D'Angelo, M. Mancini, S. Pastore, G. R. Sechi and G. Sorrenti, Using FLIPPER to predict irradiation results for Virtex 2 devices. Radiation and Its Effects on Components and Systems (RADECS), pp. 300-305, 2008.

[6] H. Guzman-Miranda, J. N. Tombs, M. A. Aguirre, FT-UNSHADES-uP: A platform for the analysis and optimal hardening of embedded systems in radiation environments. IEEE International Symposium on Industrial Electronics, pp. 2276-2281, 2008.

[7] R. Barbosa, J. Karlsson, H. Madeira, and M. Vieira, Fault Injection. Resilience Assessment and Evaluation, vol. 1, pp. 263-282, 2012.

[8] M. Gokhale, P. Graham, M. Wirthlin, D. E. Johnson, and N. Rollins, Dynamic reconfiguration for management of radiation-induced faults in FPGAs. International Journal of Embedded Systems, pp. 1-10, 2004.

[9] M. Bellato, P. Bernardi, D. Bortolato, A. Candelori, M. Ceschia, A. Paccagnella, M. Rebaudengo, M. Sonza Reorda, M. Violante, and P. Zambolin, Evaluating the effects of SEUs affecting the configuration memory of an SRAM-based FPGA. Conference on Design, Automation and Test in Europe, pp. 584-589, 2004.

[10] E. Fuller, M. Caffrey, A. Salazar, C. Carmichael, J. Fabula, Radiation characterization, and SEU mitigation, of the virtex FPGA for space-based reconfigurable computing. IEEE Nuclear and Space Radiation Effects Conference, 2000

[11] K. Chapman, "SEU strategies for Virtex-5 devices". Xilinx XAPP864 (v2.0), April 2010.

[12] H. Asadi, M. B. Tahoori, B. Mullins, D. Kaeli, and K. Granlund, Soft Error Susceptibility Analysis of SRAM-Based FPGAs in High-Performance Information Systems. In IEEE Transactions on Nuclear Science, 2007, vol. 54, no. 6, pp. 2714-2726

[13] A. Lesea, S. Drimer, J. Fabula, C. Carmichael, and P. Alfke, The Rosetta Experiment: atmospheric Soft Error Rate testing in different technology FPGAs. IEEE Trans. Device Mater. Rel., pp. 317-328, 2005.

[14] U. Gunneflo, J. Karlsson, and R. Johansson, Using Heavy-Ion Radiation to Validate Fault Handling Mechanisms. IEEE Micro, vol. 14, no. 1, pp. 8-23, 1994

[15] H. Quinn, P. Graham, J. Krone, M. Caffrey, S. Rezgui, C. Carmichael, Radiation-Induced Multi-Bit Upsets in Xilinx SRAM-Based FPGAs. IEEE Transactions on Nuclear Science, vol. 52, no. 6, pp. 2455-2461, 2005.

[16] S. Singh, A. K. Pandey, and Dipraj, Design of PI Controller to Minimize the Speed Error of DC Servo Motor. International Journal of Scientific & Technology Research, vol. 1, no. 10, 2012.

[17] J. Vinter, J. Aidemark, D. Skarin, R. Barbosa, P. Folkesson, and J. Karlsson, An Overview of GOOFI – A Generic Object-Oriented Fault Injection Framework. Technical Report No. 05-07, Chalmers University of Technology, 2005.

[18] Xilinx, Inc., "Virtex-5 configuration user guide". Xilinx User Guide 191 v3.11, 2012.

[19] C. Constantinescu, Impact of deep submicron technology on dependability of VLSI circuits. International Conference on Dependable Systems and Networks (DSN), Bethesda, USA, pp. 205-209, 2002.

[20] C. Constantinescu, Impact of intermittent faults on nanocomputing devices. IEEE/IFIP DSN (Suplemental Volume), Edinburgh, UK, pp. 238-241, 2007.

[21] T.J. Chakraborty, and C. Chiang, A Novel Fault Injection Method for System Verification Based on FPGA Boundary Scan Architecture. IEEE International Test Conference, Baltimore, USA, pp. 923-929, 2002.