2011 14th Euromicro Conference on Digital System Design

Modular Fault Injector for Multiple Fault Dependability and Security Evaluations

Johannes Grinschgl, Armin Krieg,
Christian Steger and Reinhold Weiss
Institute for Technical Informatics
Graz University of Technology
Graz, Austria
{johannes.grinschgl, armin.krieg,
steger, weiss}@tugraz.at

Holger Bock, Josef Haid
Design Center Graz
Infineon Technologies Austria AG
Graz, Austria
{holger.bock, josef.haid}@infineon.com

***Abstract*—The increasing level of integration and decreasing size of circuit elements leads to greater probabilities of operational faults. More sensible electronic devices are also more prone to external influences by energizing radiation. Additionally not only natural causes of faults are a concern of today's chip designers. Especially smart cards are exposed to complex attacks through which an adversary tries to extract knowledge from a secured system by putting it into an undefined state. These problems make it increasingly necessary to test a new design for its fault robustness. Several previous publications propose the usage of single bit injection platforms, but the limited impact of these campaigns might not be the right choice to provide a wide fault attack coverage.**

This paper first introduces a new in-system fault injection strategy for automatic test pattern injection. Secondly, an approach is presented that provides an abstraction of the internal fault injection structures to a more generic high level view. Through this abstraction it is possible to support the task separation of design and test-engineers and to enable the emulation of physical attacks on circuit level. The controller's generalized interface provides the ability to use the developed controller on different systems using the same bus system. The high level of abstraction is combinable with the advantage of high performance autonomous emulations on high end FPGA-platforms.

***Index Terms*—fault emulation, fault injection controller, saboteurs, multi-bit faults, automatic test pattern injection**

I. Introduction

The continuing success of the semiconductor industry regarding the progress of structure downscaling has led to highly integrated but also highly sensitive devices. External radiation effects, thermal and electric degradation have become common problems for the dependability of a system [1].

Through these effects transient or even permanent faults are introduced which lead to a change of the system's behavior. These faults happen randomly in contrast to ones resulting out of so-called fault attack scenarios. In this case an attacker deliberately injects faults into a system to change the system behavior. If no precautions are taken it is possible that such an attack is successful [2].

Therefore in the last years intensive research has introduced several different tools to simulate or emulate possible fault scenarios during the design phase. Especially fault emulation

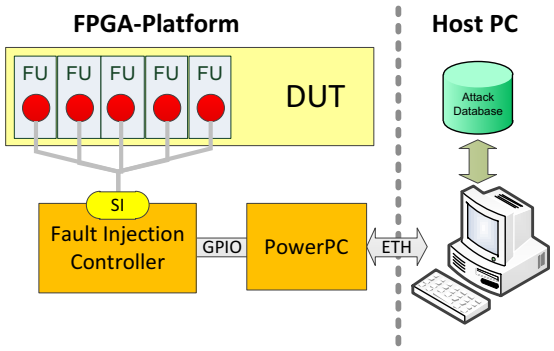

Fig. 1. Schematic view of the proposed fault injection system

proved to be a very effective way of testing a system under influence of a fault source. The usual platform to emulate such a faulty system is an FPGA because of its flexibility. An example of such an FPGA fault injection platform is shown in Figure 1. There are different ways of injecting such a fault into a circuit. One would be the use of partial reconfiguration features of the used FPGA [3]. This design approach also heavily limits the platform choice as there are only few candidates such as the Virtex families from Xilinx [4]. Another way is to instrument the given circuit either with manipulated logic elements or with integrated controllable fault elements. In the latter case it can be distinguished between saboteurs and mutants [5]. Saboteurs are small circuit elements which do not affect system behavior under normal conditions. If activated they directly inject faults into the targeted submodule by disturbing the internal signals. To disturb signals saboteurs have to placed between there source and there sink. Mutants are modified submodules that also do not affect system behavior under normal conditions but if activated behave like a faulty version of the original. To simulate or emulate fault attacks with mutants the submodule have to be replaced by a mutated submodule.

To accomplish such a test setup, it is necessary to have access to the hardware description or a standardized test interface. Such a test interface could be test chains [6]. Another important step in creating an effective fault injection platform

978-0-7695-4494-6/11 $26.00 © 2011 IEEE
DOI 10.1109/DSD.2011.76

is the selection of a proper fault model. For dependability evaluations a single event upset (SEU) fault model is often sufficient. In case of faults caused by radiation or degradation it can be safely assumed that only a single random fault will happen at a time [7]. Security evaluations on the contrary consider intentional faults. Therefore it is possible that an attacker introduces several faults at once to achieve the desired effect of secret exposure.

Such a multi-bit fault model results in much more complex fault relationships than SEU ones, as time and space of such faults cannot be assumed of being random. This model has to reflect the dependency of several SEUs among each other. In case of dependability analyses this means that similar types of sensitive sub-circuits will fail at the same time. In case of security evaluations this dependency is defined by the attack scheme of the adversary.

Finally, the controlling element itself has to be designed very carefully. The scope reaches from a very simple implementation completely controlled by an external source like a personal computer to very sophisticated integrated designs. Simple implementations have the advantage of being very adaptable to system changes, but they are also highly dependent on the used communication interface. Therefore they can be considered as slow. Such an implementation based on a PCI interface is shown in [6]. Sophisticated solutions allow for very high fault injection rates while being difficult to adapt to system changes [8].

In the past years implementations of fault injection emulation controllers are either simple and limited by slow interface performance or they are highly complex and therefore not portable from one design to another.

In this paper we propose a modular fault injector(MFI) that combines the advantages of a simple portable design and fast highly complex implementations. It also helps to provide a common platform for fault injection campaigns on different target architectures. Another point that is often neglected in previous work in this field is the consideration of fault attacks.

Finally the consideration of multi-bit fault patterns instead of simple SEUs can be seen as an important contribution of this work. This is especially important for security evaluations. It has also to be mentioned that this fully synthesizable controller can also be used as an online-testing implementation if desired.

The main goals of this work can be summarized as follows:

- Fully modular fault injector design
- Multi-bit fault injection to support fault attack emulation
- Online-testing support

This paper is structured as follows. Section II briefly shows the state of the art on work related to emulated fault injection using integrated controllers and different reconfiguration techniques. In Section III the design of the fault injection controller is described. Section IV shows some experimental results of this MFI by means of the LEON3 processor [9]. Finally, conclusions are drawn in Section V.

II. Related Work

The introduction of fault emulation on FPGA platforms has led to several publications especially concerned with the emulation of SEUs in space applications. It has to be distinguished between approaches using direct circuit manipulation like the solution proposed in this paper and techniques using reconfiguration features of certain FPGAs.

The former possibility can again be divided into two subcategories. One using specialized hardware units to instrument saboteurs or mutants as shown in [8].

The second variant is to use available processor cores for fault injection automation similar to the solution provided by [10]. This solution promises a high emulation performance because of fast on-chip communication channels, but it is more difficult to port to other platforms.

In the last years partial and complete runtime reconfiguration of FPGAs got a common technique to implement fault injection in a flexible way. Reconfiguration is possible on special FPGA series using an external communication interface to directly feed different fault configurations into the FPGA as shown in [11]–[13]. If this external interface is too slow it is also possible to use existing processor resources for the reconfiguration task as presented in [14], [15]. While the reconfiguration approach is tempting (and has been used by several research groups) it also limits the maximum reachable fault injection performance and the designer's platform selection. Similar to reconfiguration approaches is the work presented in [16], in which the synthesized netlist is augmented to preserve the original structure of the system.

In the field of security evaluations work has been done by the authors of [17]. The presented work is done using a proven fault injection platform presented in [18]. This investigation confirms the strong need for multiple fault injection campaigns in the design process. However in their experiments only a small fault multiplicity (six) was used to prove this point.

III. Modular Fault Injector

The following reasons summarize the main drivers behind the development of this modular fault injection controller (MFI):

- Support for fault patterns to support multi-bit fault injection
- Multi-mode saboteur support
- Scalable interfaces to support automatic saboteur placement
- Standardized communication interface
- Internal memory for automatic fault injection

To support an easy application of the fault injection system to a new design under test, a standardized communication interface is needed. The General Purpose Input/Output (GPIO) communication interface enables the developer to use the proposed controller in a wide selection of different architectures. The GPIO interface consists of pins which can be configured via software commands. These pins allow for controlling the MFI without disturbing the device-under-test.

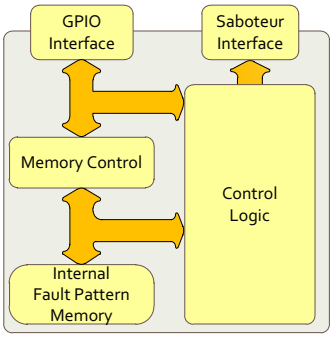

Fig. 2. Schematic view of the fault injection controller

Extensive fault injection campaigns are only possible using a large number of active saboteurs. It is not possible to route such a large number manually, therefore internal interfaces have to be scalable to enable automated injection processes.

An effective separation of the design and test-engineers tasks can only be guaranteed through a generalized view of the fault injection system. This is done using so called fault patterns, high level representations of the underlying saboteur distribution. These patterns are also necessary for efficient physical fault attack emulation.

High speed automated operation is enabled through a flexible internal memory system. In the design phase of the proposed fault injector, future advances of the evaluation platform have to be considered. This includes support of large fault patterns and burst loading from the bus-system.

Different fault and attack scenarios call for different saboteur types. To simplify the reconfiguration process, different saboteur modes are provided by a single flexible saboteur design.

A basic block diagram of the proposed fault injection system is shown in Figure 1. The flow consists of the fault injection controller which specifies the attack scenario, the saboteurs which disturb signals and a PowerPC which is used as the interface to the PC.

A. Fault Injection Controller

The fault injection controller is the main part of the MFI. It controls the mode and the activation time of the saboteurs. As seen in Figure 2 the fault injection controller consists of two interfaces. The first one is the saboteur interface. This interface is a bus where for each saboteur an own activate signal is included. It also contains mode signals for controlling the mode of the saboteurs. The second port is the General Purpose Input/Output (GPIO) port. This interface is used to control the fault injection controller. Via the GPIO port the internal fault pattern memory is filled. This memory is used to specify when which attack pattern shall be activated.

Thanks to the GPIO interface of the MFI it is possible to automate fault injection campaigns. The easiest way to control the MFI via GPIO commands is through the FPGA-internal PowerPC. Through this solution a simple generic software interface to the generic hardware interface is provided. This has the advantage that a test-engineer does not need specific knowledge about the fault injection system itself. Especially in case of security evaluations the test-engineer will be mostly concerned with a good mapping of a real attack scenario to the saboteur matrix inside the system of interest. The GPIO interface of the MFI can be easily changed to every other interface which allows to write the internal memory of the MFI. This flexibility makes the whole flow independent to the used test environment.

TABLE I
SABOTEUR OPERATION TYPES AND THEIR DESCRIPTION

Saboteur mode	Fault type	Description
Stuck-at-Zero	Permanent	Signal value of '0' until reload
Stuck-at-One	Permanent	Signal value of '1' until reload
Indetermination	Permanent	Undefined signal state until reload
Bridging fault	Permanent	No output propagation until reload
Negation of input	Permanent	Undefined signal state until reload
Bit-flip	Transient	Output inverts input for one cycle
Artificial delay	Transient	Input to output propagation delay

Implementation: The fault injection controller is split into five main parts.

- Saboteur interface: The saboteur interface was implemented to allow for easy expandability to support a higher number of saboteurs. The size of the saboteur interface bus is equal to the number of saboteurs. The interface also includes mode signals to control the mode of the saboteurs.
- GPIO interface: The GPIO interface is the interface of the MFI to the controlled processor. Via this port all attack scenarios can be loaded into the MFI.
- Memory control: The memory control is used to place the fault patterns into the internal fault pattern memory. It also generates signals for the control logic if a special command is sent via the GPIO port.
- Internal fault pattern memory: This memory stores the fault patterns which define the attack scenario.
- Control logic: The control logic activates the saboteurs depending on the information stored in the internal fault pattern memory.

B. Saboteurs

To model a wide selection of possible faults we introduce a configurable saboteur. According to [19] and [20] saboteurs can be distinguished into several different types. Depending on their location it can be differentiated between serial and parallel saboteurs. Depending on the directionality the division can be made between uni- and bidirectional ones. Finally depending on their complexity it can be distinguished between simple and complex saboteurs. The saboteur used in this work can be classified as an unidirectional serial simple saboteur. This means it only works in one direction, it is located directly into the connection signal and affects only one port at the input and output side. The supported fault models of such an injection element are shown in Table I.

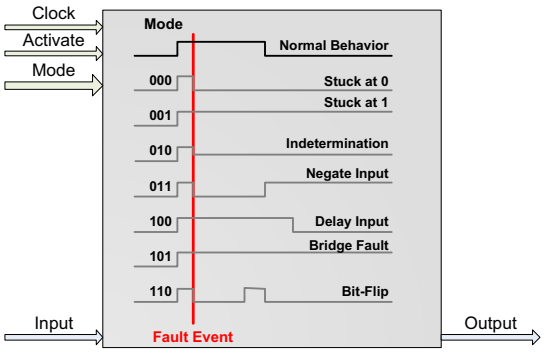

Fig. 3. Block diagram of the proposed saboteur element

The proposed saboteur can be used to inject faults into single bit lines or even complete buses. This allows the emulation of bus attacks with e.g., unfocused lasers or influences by strong external energy sources. The first four saboteur configuration modes are equivalent to direct circuit modifications. Bit-flips could also be forced by short intensive external pulses. Delay faults emulate circuit behavior in case of operating voltage changes. A schematic block diagram and fault effect visualization of the proposed single-bit saboteur is shown in Figure 3.

Implementation: The complexity of the implementation of the saboteurs depends on the features a saboteur shall support. Therefore the required number of slices for one saboteur mainly depends on their complexity. If the interface of the saboteurs is not changed only the architecture of the saboteur has to be modified to support more features. So if the saboteurs are placed the test-engineer can adapt the saboteurs by only changing one file. This makes the saboteur placement concept very flexible.

C. Fault pattern support

To evaluate a device-under-test for its fault attack sensitivity it is important to know the exact location of security relevant silicon regions. At this locations saboteurs have to be placed by a test-engineer. A fault pattern approach is used for an effective mapping of saboteurs to their corresponding fault locations. Of course, not every possible pattern combination has to be transmitted into the MFI. Every used pattern represents a very likely fault scheme determined in previous physical experiments. The basic concept of fault pattern to physical implementation mapping is shown in Figure 4.

Implementation: In this example only a random number generator is used as pattern generator to show the function of the fault MFI. The complexity of such a pattern generation does not increase the required slices of the MFI, because the patterns are not generated directly in the MFI.

D. Automatic Saboteurs

For a high amount of saboteurs an automatic saboteur placement approach is required, because manual saboteur placement is a very time consuming process. In [19], a theoretical method

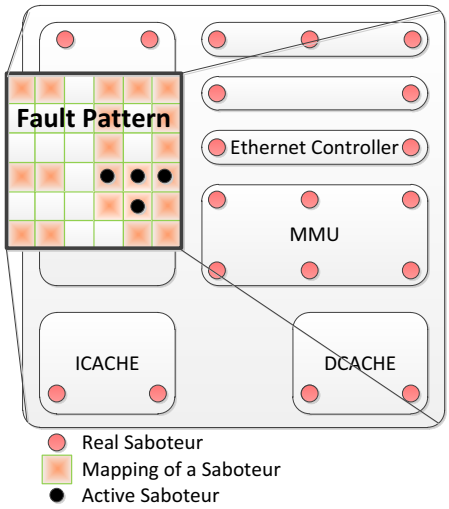

Fig. 4. Schematic view of an extracted fault pattern

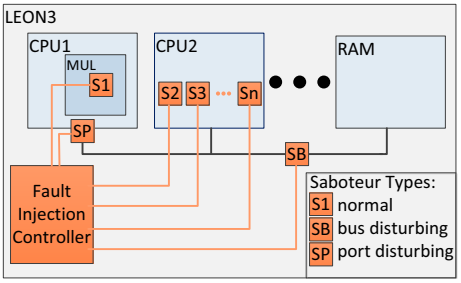

Fig. 5. Automatic routed saboteurs (adapted from [21])

is discussed how to place saboteurs automatically into a VHDL design. Our approach works very similar to that approach. In Figure 5, a schematic example of an automatic saboteur placement is shown. The saboteurs placement tool is based on the vMAGIC VHDL parser library [22]. VMagic is a Java API which allow to parse and adapt VHDL code in an automatic way. The flow can automatically add hundreds of saboteurs [21]. Then the VHDL code is automatically adapted with saboteurs and the fault injection controller. A method to modify VHDL code automatically is shown in [23].

E. Attack Scenario

As seen in Figure 6 the attack flow is based on a golden model run. The golden model runs from one reset to the next reset of the DUT. All information of the golden model is stored (e.g., the output, none volatile memory (NVM), timing, ...).

The next step is to reset the whole system. Then all attack patterns are stored into the internal fault pattern memory via the GPIO interface. Also the attack mode has to be transmitted via GPIO. The mode is used to define the attack type of the saboteurs (e.g., bit-flip). After a pre-determined time the specified fault attack will be injected and then the saboteurs are activated depending on the fault pattern. Now the faulty DUT will run until it is reset or a pre-defined timeout is reached.

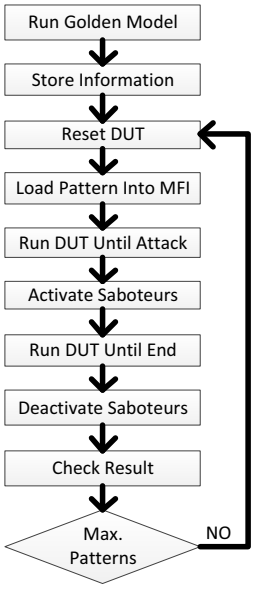

Fig. 6. Flow diagram shows a typical fault injection run

This timeout is required because the device could run into an infinite loop after the fault injection.

After the fault injection run is finished the saboteurs are deactivated and the result is compared to the results from the golden model run. If the results are not equal a fault analysis process has to be launched. This procedure is continued until all attack patterns are tested.

The fault pattern generator creates attack patterns for the fault injection. In this example only a random number generator is used as pattern generator to show the function of the fault MFI. But because the patterns are not generated directly in the MFI the complexity of such a pattern generation does not increase the required slices of the MFI.

IV. Experimental Results

To prove the effectiveness of our approach we chose to implement the proposed controller on a widely used platform,the LEON3 SPARC V8 conformant processor of Gaisler Research [9].

The test setup is implemented on a Virtex5 FPGA using the Xilinx ML507 evaluation board. Simulation results are gained using the Modelsim software which is part of the ISE software package provided by Xilinx.

The fault injection setup is configured as shown in Table II.

A. LEON3 platform setup

The LEON3 is configured for a single-core configuration using default platform settings for this particular evaluation board (ML507). The MFI is configured via GPIO and can be accessed using the PowerPC.

TABLE II
Fault injection setup

Saboteur type	Fault mode	Fault target type	Fault injection targets
Single-bit	Bit-flip	Control logic	Integer pipeline Cache controller Register file Multiplier unit Divider unit

TABLE III
Simulated fault injection performance

Saboteurs	Inj. Patterns	Time [sec]	Patterns/sec.
8	256	274	0.93
16	256	282	0.91
24	256	286	0.90
32	256	299	0.86

B. Saboteur configuration

The random approach of fault injection proposed in this publication allows to find global dependencies between local fault occurrences. Of course, with rising number of saboteurs possible combinations rise until an analysis of the results is not possible in a reasonable time frame. Therefore a small number of saboteur locations has been chosen for these fault injection experiments.

C. Principle simulation results

To ensure the correct behaviour of the fault injector and its saboteur, the whole LEON3 system has been thoroughly simulated using the ModelSIM software package. It has to be mentioned that the simulation includes not only the fault injection process but also start and calculation instructions. Results of these injection campaigns are shown in Table III.

The pattern injection rate is constant for patterns smaller than 32, because of the working principle of the MFI. For larger patterns additional GPIO transfers are required. However with increasing pattern size the amount of concurrently injected faults rises accordingly.

D. VHDL Synthesis Results

To estimate the necessary FPGA area requirements a synthesis of a typical platform configuration has been performed. The results of several synthesis runs using different saboteur configurations is shown in Table IV.

It can be seen that the MFI and its saboteurs has a negligible effect on the size of the resulting synthesized design. Basing on this routed results, it can be expected that it should be possible to implement a large amount of saboteurs on this FPGA-platform.

E. Fault Injection Performance

In this subsection results of several fault injection campaigns are presented to show that higher structural flexibility does not come with disadvantages concerning emulation speed. It also

TABLE IV
VHDL SYNTHESIS RESULTS

Saboteurs	Look-up-tables	Overhead[%]	Slices	Overhead[%]
0	14897	-	10423	-
8	14950	0.36%	10513	0.86%
32	15107	1.41%	10642	2.10%
96	15194	1.99%	10716	2.81%
170	15244	2.33%	10774	3.37%
326	15478	3.9%	11088	6.34%

TABLE V
EMULATED FAULT INJECTION PERFORMANCE

Saboteurs	Inj. Patterns	Time [sec]	Patterns/sec.	Pattern Blocks/sec.
8	100M	46.76	2.17M	2.17M
32	100M	46.76	2.17M	2.17M
96	100M	114.48	873.5k	2.17M
170	100M	156,4	639.4k	2.17M
326	100M	282.16	354.4k	2.17M

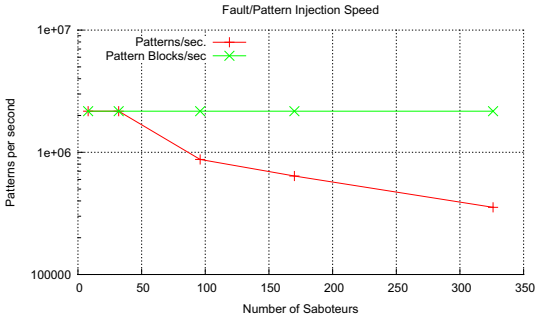

Fig. 7. Speed of the fault injection depending on the number of saboteurs

needs to be mentioned that such a multi-bit injection campaign leads to complex results in system behavior and therefore the analysis modules will most likely be the slowest link in the emulation chain. The results are summarized in Table V. It can be seen that the number of saboteurs has an influence on the number of patterns which can be injected per second. The reason is that the GPIO interface only support the transmission of 32 bits of the pattern simultaneously. If the size of the pattern is increased the time for the pattern transmission is increasing. The last column shows that the transmission of one pattern is interdependent to the number of saboteurs. It is not required to send for each attack the whole pattern. If only one pattern block has to be changed only one GPIO transmission is required. But not always only the maximum number of faults is important because not all saboteurs are activated. The more important numbers are the patterns injected per second. The speed of the pattern injections per seconds is not independent of the number of saboteurs. But the transmission of on fault pattern is constant. This is presented in Figure 7.

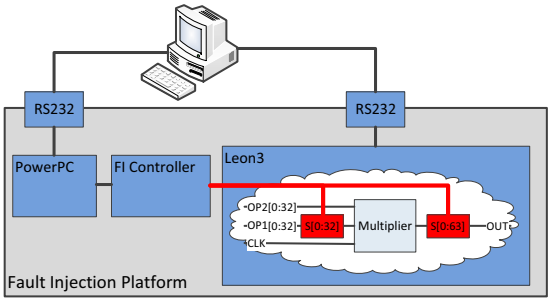

Fig. 8. Overview of the test environment.

F. Case Study: Attack on RSA Authentication

The MFI is used to show a case study of a fault attack on RSA authentication [24]. The RSA authentication process signs a message with a private key. To generate such an signature the RSA algorithm uses multiplications. In [24] it has been concluded that fault attacks on the multiplication unit can be used to attack the RSA algorithm. For the described fault attack the authors assume that only one bit of the multiplier output switches. In this case study we show that our approach can emulate exactly such fault attacks.

For this test we chose the LEON3 processor in single core configuration. For the attack target the output signals and one of the operand registers of the multiplier have been chosen.

An overview of the test environment is given in Figure 8. Listing 1 shows pseudo code describing the fault injection flow. The saboteur placement methodology is described in [21]. The FPGA-internal PowerPC is running a Linux kernel to allow for programming the fault injection flow in an comfortably way.The fault injection control software is running on the PowerPC. The host PC evaluates the results of the fault injection. During this attack the saboteurs are configured in bit-flip mode.

```
run_golden_model();
store_all_information();
for(i=0;i<n_patterns;i++){
        reset_dut();
        load_pattern(pattern[i]);
        run_DUT(pattern_time[i]);
        activate_saboteurs();
        run_DUT(max_time);
        deactivate_saboteurs();
        check_result();
}
```

Listing 1. Pseudo code of the fault injection flow

In Table VI and Figure 9 the results are shown. As seen the MFI can emulate a fault attack on the multiplier interface. Because the MFI supports a large amount of saboteurs it is possible to place the saboteurs not only at the output. With this it is possible to test also the effect of a fault injection to the input of the multiplier. As assumed, attacks to the output of the multiplier can be used to emulate attacks as shown in [24]. Attacks to the input of the multiplier mostly disturb more than one output bit. But in [24] the authors assume for there attack that only one bit of the multiplication has to be disturbed.

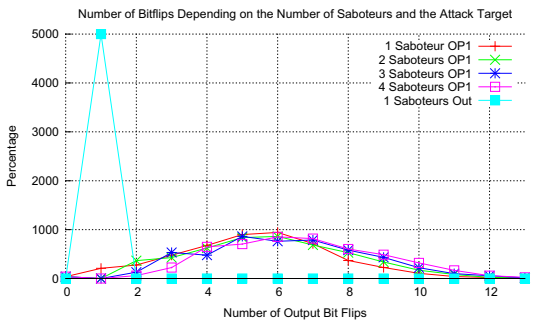

Fig. 9. Influence of saboteurs on the multiplication unit. Bit-flips of the output depending on the number of active saboteurs at the input/output.

TABLE VI
NUMBER OF FAULTS DETECTED DEPENDING ON THE ATTACK PATTERN

Attack Target	OP1	OP1	OP1	OP1	Out
Number of Saboteurs	1	2	3	4	1
Nr. of Bit-flips					
0	37	37	37	37	0
1	208	0	0	0	5000
2	275	365	129	59	0
3	477	428	532	224	0
4	676	624	478	646	0
5	899	836	862	704	0
6	942	862	762	852	0
7	714	686	787	817	0
8	369	529	580	604	0
9	226	337	432	487	0
10	105	170	221	319	0
11	40	82	99	167	0
12	20	22	56	58	0
13	9	13	13	24	0

G. Comparison to existent fault injector solutions

During the last years many high performance fault injection emulation platforms have been published. While fault injection speed is only one parameter to quantify the performance of a fault injector solution, it is important to measure how many faults can be injected in a reasonable time. This is especially important for complex system-on-chip designs with a high amount of interesting injection points. In [6] fault injection campaigns using different benchmark configurations have been compared to typical simulation solutions. To cover all considered design approaches also reconfiguration techniques are presented in [12]. This publication shows two implementations, one using autonomous emulation (AE) and one using partial reconfiguration (PR). The test object in this case is the freely available CORDIC16 core.

It has to be mentioned that our fault injector solution profits from a higher clock frequency and state-of-the-art FPGA hardware. Another difference that has to be considered is that because of the pattern approach the fault injection rate is not constant. Therefore the total number of activated faults has been calculated, based on the number of activated faults per turn. For the evaluation the worst case is assumed, that only one fault is injected per pattern. If the simultaneous injected faults are increased the number of faults per seconds will be increased (one turn consists of all possible bit combinations of the selected fault pattern width). The results of these investigations are compared to our result in Table VII. Because of the different test systems the values have to be normalized. The last column shows the fault injection speed normalized to a clock cycle of 25 ns.

TABLE VII
FAULT INJECTION PERFORMANCE COMPARED TO PREVIOUS RESULTS

Approach	Clock cycle	Inj. Faults	Time	Inj. Speed	Nor. Inj. Speed
	[ns]		[s]	1/[s]	1/[s]
[6]	50	100k	83	1205	603
[12] AE	16.8	129.75M	698	185888	276619
[12] PR	18.97	10k	1014	9.86	13
[14] DPR	10	-	-	12987	32468
[14] PRR	10	-	-	414.94	1037
This work	25	100M	46.76	2.17M	2.17M

The comparison shows that the in-system solution does not suffer from additional performance penalties. Strong spacial containment of possible fault locations leads to very high injection results if combined with large fault injection patterns.

It can also be concluded that autonomous emulation provides injection speed advantages compared to partial reconfiguration techniques. The reconfiguration approach suffers not only from limiting communication interfaces but also from long delays caused by the reconfiguration process. This also does not include the time needed to generate reconfigurable sub-blocks. These problems are targeted by the work presented in [14]. In this publication partial reconfiguration is compared to direct reconfiguration using an FPGA internal port to its configuration memory. While reconfiguration speed is greatly improved, it is still slower than autonomous emulation solutions like the one proposed in this paper.

V. CONCLUSION

This paper presents a highly modularized controller solution for portable fault injection systems. It has been shown that these fault injection campaigns can be executed very efficiently through a generalized interface using a high level abstraction of physical fault sources. The design is scalable to allow both, fully automated campaigns with a larger fault pattern memory or more user controlled campaigns using a small silicon footprint. The performance is comparable to existing platforms using circuit manipulation and significantly faster than ones using partial and complete FPGA reconfiguration.

This is the first step towards a complete fault injection platform for dependability and security evaluations. Its flexible design will allow the test-engineer to shift the focus from complex testing architectures to more complex fault models. This should finally allow to not only model simple SEUs but also complex fault attack scenarios. Through the additional abstraction level a better separation of the test and design

engineer's tasks is provided. The gained knowledge of these experiments will be used to get a better understanding of inner-chip fault mechanics. Such full scale investigations need fully automated saboteur injection techniques which are currently under development.

The consideration of such attacks will be necessary to design efficient and secure smart card systems. This is also valid for highly integrated systems or systems under high environmental stress.

ACKNOWLEDGMENT

The authors would like to thank the Austrian Federal Ministry for Transport, Innovation, and Technology, which funded the Power-Modes project under the FIT-IT contract FFG 825749. We would also like to thank our project partners Infineon Technologies Austria AG and Austria Card GmbH.

REFERENCES

[1] M. Nicolaidis, "Time redundancy based soft-error tolerance to rescue nanometer technologies," in *Proc. IEEE VLSI Test Symposium (1999)*, 1999, pp. 86 –94.

[2] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan, "The sorcerer's apprentice guide to fault attacks," *Proceedings of the IEEE*, vol. 94, no. 2, pp. 370–382, feb 2006.

[3] D. D. Andres, J. C. Ruiz, D. Gil, and P. Gil, "Fades: a fault emulation tool for fast dependability assessment," in *Proc. IEEE Int. Conf. Field Programmable Technology (FPT 2006)*, 2006, pp. 221–228.

[4] Xilinx, "Virtex-5 fpga family," Online, 07 2010. [Online]. Available: http://www.xilinx.com/products/virtex5/index.htm

[5] J. Boue, P. Petillon, and Y. Crouzet, "Mefisto-l: a vhdl-based fault injection tool for the experimental assessment of fault tolerance," in *Twenty-Eighth Annual International Symposium on Fault-Tolerant Computing, 1998. Digest of Papers.*, 23-25 1998, pp. 168 –173.

[6] P. Civera, L. Macchiarulo, M. Rebaudengo, M. S. Reorda, and M. Violante, "An fpga-based approach for speeding-up fault injection campaigns on safety-critical circuits," *Journal of Electronic Testing*, vol. 18, no. 3, pp. 261–271, 2002, 10.1023/A:1015079004512. [Online]. Available: http://dx.doi.org/10.1023/A:1015079004512

[7] A. Benso and B. Prinetto, *Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation*. Kluwer Academic Publishers, 2003.

[8] P. Ellervee, J. Raik, K. Tammemae, and R.-J. Ubar, "Fpga-based fault emulation of synchronous sequential circuits," *IET Computers & Digital Techniques*, vol. 1, no. 2, pp. 70–76, 2007.

[9] Gaisler, "Leon3 processor," Online, 07 2010. [Online]. Available: http://www.gaisler.com/cms/index.php?option=com_content&task=view&id=13&Itemid=53

[10] A. Pellegrini, K. Constantinides, D. Zhang, S. Sudhakar, V. Bertacco, and T. Austin, "Crashtest: A fast high-fidelity fpga-based resiliency analysis framework," in *Proc. IEEE Int. Conf. Computer Design (ICCD 2008)*, oct 2008, pp. 363–370.

[11] L. Antoni, R. Leveugle, and M. Feher, "Using run-time reconfiguration for fault injection in hardware prototypes," in *Proc. IEEE Int. Symp. Defect and Fault Tolerance in VLSI Systems (DFT 2002)*, 2002, pp. 245–253.

[12] C. Lopez-Ongil, L. Entrena, M. Garcia-Valderas, M. Portela, M. A. Aguirre, J. Tombs, V. Baena, and F. Munoz, "A unified environment for fault injection at any design level based on emulation," *IEEE Transactions on Nuclear Science*, vol. 54, no. 4, pp. 946–950, 2007.

[13] M. Jeitler, M. Delvai, and S. Reichor, "Fuse - a hardware accelerated hdl fault injection tool," in *Proc. SPL Programmable Logic 5th Southern Conf*, 2009, pp. 89–94.

[14] L. Kafka, "Analysis of applicability of partial runtime reconfiguration in fault emulator in xilinx fpgas," in *DDECS '08: Proceedings of the 2008 11th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 1–4.

[15] B. F. Dutton, M. Ali, C. E. Stroud, and J. Sunwoo, "Embedded processor based fault injection and seu emulation for fpgas," in *Proc. International Conf. on Embedded Systems and Applications 2009*, 2009, pp. 183–189.

[16] L. Kafka, M. Danek, and O. Novak, "A novel emulation technique that preserves circuit structure and timing," in *International Symposium on System-on-Chip, 2007*, 20-21 2007, pp. 1 –4.

[17] R. Leveugle, "Early analysis of fault-based attack effects in secure circuits," *IEEE Transactions on Computers*, vol. 56, no. 10, pp. 1431–1434, 2007.

[18] R. Leveugle and K. Hadjiat, "Multi-level fault injections in vhdl descriptions: Alternative approaches and experiments," *Journal of Electronic Testing*, vol. 19, no. 5, pp. 559–575, 2003.

[19] J. Baraza, J. Gracia, D. Gil, and P. Gil, "Improvement of fault injection techniques based on vhdl code modification," in *Tenth IEEE International High-Level Design Validation and Test Workshop 2005*, 30 2005, pp. 19 – 26.

[20] J.-C. Baraza, J. Gracia, S. Blanc, D. Gil, and P.-J. Gil, "Enhancement of fault injection techniques based on the modification of vhdl code," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 6, pp. 693 –706, jun. 2008.

[21] J. Grinschgl, A. Krieg, C. Steger, R. Weiss, H. Bock, and J. Haid, "Automatic saboteur placement for emulation-based multi-bit fault injection," In Press.

[22] C. Pohl, R. Fuest, and M. Porrmann, "vmagic – automatic code generation for vhdl," *newsletter edacentrum*, vol. 2, pp. 7–10, Jul. 2010.

[23] C. Bachmann, A. Genser, C. Steger, R. Weiss, and J. Haid, "Automated power characterization for run-time power emulation of soc designs," in *Proc. 13th Euromicro Conf. Digital System Design: Architectures, Methods and Tools (DSD)*, 2010, pp. 587–594.

[24] A. Pellegrini, V. Bertacco, and T. Austin, "Fault-based attack of rsa authentication," in *Proc. Design, Automation & Test in Europe Conf. & Exhibition (DATE)*, 2010, pp. 855–860.