IC Security Evaluation against Fault Injection Attack Based on FPGA Emulation

Song Xu, Qiang Liu, Tao Li, Hongxiang Fan
School of Electronic Information Engineering
Tianjin University
Tianjin, China
{songxu, qiangliu, taolee}@tju.edu.cn

Abstract—**Fault injection attacks (FIAs) are becoming a serious threat to the security of integrated circuits (ICs). Circuit designers need a simple and effective method to evaluate the ability of their IC designs against the FIAs at design stage. To address the need, this paper based on FPGA emulation proposes a method, which mimics FIAs on a circuit design implemented on a FPGA by instantiating a uniform architecture which can generate different fault models and span the space of faults and inserting fault injection logic into the design. After the circuit design is submitted to the emulation platform, the fault generation and injection are executed automatically, and a report about the ability of the IC design against the FIAs is generated. The method works at the circuit netlist level, leaves the source codes of the circuit design unmodified, and makes the evaluation a transparent process to the designers. The experiment with various fault models and differential fault analysis on AES-128 encryption circuit shows the effectiveness of the method.**

Keywords—FPGA emulation, fault injection attack, integrated circuit security

I. Introduction and Related Works

Combining with mathematic analysis, especially with differential fault analysis (DFA) [1], fault injection attack (FIA), a technique to induce errors into integrated circuits (ICs) such as crypto circuits used in smart cards and cell phones [2], might obtain the key data processed in the chip and threatens the security of ICs. Attack techniques like electromagnetic FIA [3], make it an urgent problem to find an effective solution to do evaluation on the ability of ICs against FIAs. Different solutions mainly including chip-based test [3], software-based simulation [4] and hardware-based emulation [5-9], have been proposed in the literatures. Chip-based attack test after fabrication takes high cost, and makes security evaluation at design stage more attractive. Software-based simulation utilizes its flexibility and observability, however raises time overhead to obtain a good coverage of faults [9]. Hardware-based emulation uses field programmable gate array (FPGA) as a platform to accelerate the process.

FPGA-based solutions can be generally achieved using reconfiguration-based [5-6] and instrumentation-based [7–9] techniques. [5] details a solution that exploits the special features of Xilinx FPGAs known as capture, readback and partial reconfiguration. It uses bit-flip model to reproduce a set of single event phenomena, thus limits the space of fault models. [6] utilizes the Xilinx ICAP to reconfigure a FPGA. The chosen frame of configuration bitstream is read from the FPGA, modified to inject a fault and written back into the FPGA over the ICAP. However, it is hard to know which gates and/or flip-flops in the circuit under test (CUT) are affected after modification. Besides, in a reconfiguration-based technique, the FPGA needs to be reconfigured for each fault test case. This results in an enormous time overhead. Instrumentation-based techniques use fault injection circuit for each fault site to produce the fault specified. [7] implements emulated fault injection components based on TRNG (true random number generator). And, only probabilistic faults are supported. [8] injects faults into the CUT by inserting mask logic and mask chains for each fault site. In [9] each D flip-flop (DFF) is modified to add two XOR gates and one 2-to-1 MUX and is directly controlled, to inject transient bit-flip faults. The mechanisms proposed in [8] and [9] both lead to considerable area overhead.

This paper proposes a generic instrumentation-based FPGA emulation framework for IC security evaluation. Instead of transient faults in [9], a uniform architecture is provided to generate various fault models. The architecture can span a wide range of possible FIAs. A formatted data structure named emulation data list is recommended to break up the relationship between the mechanism of injecting faults and the process of instantiating fault models. The fault injection mechanism proposed is designed based on scan chains and fault injection management circuit, balancing area and time overhead for fault injection emulation. The framework is based on Python rather than scripts which are proposed in [9] and result in dependency on vender-specific tools. The whole process of the method proposed is automated and transparent to users. Meanwhile, instead of being simply divided into several classes, the outputs of the CUT are analyzed in detail. Following are the details.

II. The Proposed Evaluation Method

A. Flow of the Proposed Evaluation Method

In general, faults caused by FIAs can occur in the DFFs and/or the combinational part of a circuit [9]. The faults in the combinational part are propagated and finally caught by the subsequent DFFs. Therefore, by changing the contents in one or more specific DFFs, the emulation can cover all possible faults of FIAs. The proposed security evaluation method works in the way similar to the procedure in the typical IC testing. The FPGA emulation platform is switched between the normal working mode and the scan mode, but instead of test vectors the fault vectors are scanned into the selected DFFs to change their contents to inject faults in the scan mode.

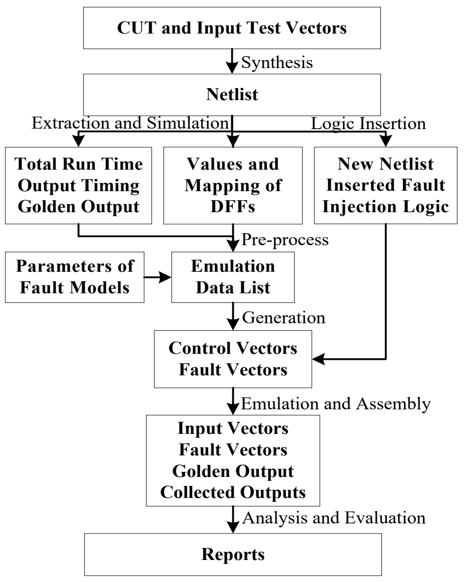

Fig. 1. The flow of the proposed evaluation method

The flow of the proposed evaluation method is shown in Fig.1. Python programs are developed to carry out all steps except synthesis and generation of bitstream. The inputs include the source codes (Verilog/VHDL) of the CUT and the input test vectors. The source codes are synthesized into a netlist firstly, and necessary information is extracted from the netlist and dumped from a simulation. Fault injection logic including fault injection manager (FIM) and scan chains is inserted into the netlist. Next, parameters of fault models are specified and translated into a list named emulation data list with the information extracted and dumped before. The emulation data list is recommended to break up the relationship between the mechanism of injecting faults and the process of instantiating fault models. Then, fault vectors and control vectors for emulation are generated automatically with the new netlist. Control vectors are used for the FIM to manage the process of emulation. When emulation starts, the input test vectors, the fault vectors and the control vectors are fed into the CUT and the outputs of the CUT are collected. Finally, evaluation program is launched to establish a report about the ability of the CUT against FIAs.

B. UNIFORM ARCHITECTURE OF FAULT MODELS AND EMULATION DATA LIST

Fault models are the key to emulate the FIAs in the real world. Various fault models were proposed based on the characteristics of the faults [10]. As shown in Table I, four characteristics of faults are parameterized in this work to provide a uniform architecture which can generate various fault models and span the space of faults. Custom fault models are also allowed. *Location* and *strength* parameterize which variable in the source codes and how many bits of it are affected by a fault. The fault *type* can be flip (f), maintain (m),

TABLE I. PARAMETERS OF THE FAULT MODEL USED IN THE PROPOSED FAULT INJECTION EMULATION PLATFORM

Location	Strength	Type	Time
top_module_name[. instance_name].var _name	number of bits	f/m/r/sa-0/sa-1	[active_clock _cycle_list]

[a]. There could be multiple levels of instantiation in the square brackets.

random (r), and stuck-at 0 or stuck-at 1. The list of *time* specifies in which clock cycles a fault is active. After specified, the parameters are parsed into emulation data list. The formatted data structure of the emulation data list is shown in Fig.2. The key word *EMU* defines a data block for an emulation. There exist four sub-blocks named with *RUNTIME*, *IN*, *OUT* and *FAULTS*. The block *RUNTIME* defines the CUT's total run time. The block *IN* specifies the CUT's input test vectors. The block *OUT* gives a list of time for the collection of the CUT's outputs. The block *FAULTS* describes the faults to be injected, specifying in which clock cycle to inject what type of fault into which DFF.

```
EMU{
    RUNTIME ( total_runtime );
    IN{
            input_vector1;
            input_vector2;
            …
    }
    OUT (capture_time);
    FAULTS{
            @clock_cycle_1
            dff1_name = fault_value1;
            dff2_name = fault_ value2;
            …
            @clock_cycle_N
            dff2_name = fault_ value3;
            dff4_name = fault_value4;
            …
    }
}
```

Fig. 2. The formatted data structure of the emulation data list

C. PROPOSED ARCHITECTURE FOR EMULATION

Fig.3 gives the proposed architecture of the platform for the fault injection emulation. It contains a FPGA system including the FIM and the CUT inserted scan chains, an off-chip RAM for data exchange and a host PC runs the Python programs. The FIM is implemented to control the process of emulation. It is mainly composed of an *input provider*, an *output dumper*, a *time manager* and a *scan controller*. The *input provider* is responsible for reading the input test vectors from the off-chip RAM and inputting the vectors to the CUT, while the *output dumper* collects the outputs of the CUT. The *time manager* controls the operation of the other modules. Several registers in it are used to set the CUT's total run time, the time to activate the *output dumper*, and the time parameter in Table I . The *scan controller* is for reading the fault vectors from the off-chip RAM and injecting them into the scan chains. The scan flip-flops (SFFs) in the CUT are divided into chains with the equivalent length, and the chains can operate concurrently or individually. Each scan chain has an individual clock, which is gated by a enable signal. The SFFs are what typically used in design-for-test. Unlike previous approaches, the technique [11] is adopted to reduce the area overhead of SFFs, in which the extra MUX is mapped with the other combinational logic on the partially used LUT.

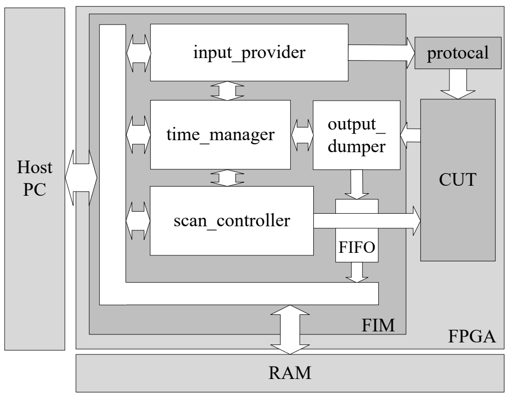

Fig. 3. Proposed architecture for fault injection emulation.

III. EXPERIMENT AND RESULTS

Faults specified by fault models shown in Table II are injected into the state *S* of AES-128 to demonstrate the effectiveness of the proposed platform on Zynq7z010. Xilinx Design Tools are used for simulation, synthesis and place&route. And DFA [1] is used for analysis and evaluation.

TABLE II. FAULT MODELS FOR EXPERIMENT

Fault Model No.	Location	Strength	Type	Time
1	AES.S	1	f	(11, 1, 0, 1)
2	AES.S	1	sa-1	(11, 1, 0, 1)
3	AES.S	1	sa-0	(11, 1, 0, 1)
4	AES.S	2	(f, f)	(11, 1, 0, 1)
5	AES.S	2	(f, sa-1)	(11, 1, 0, 1)
6	AES.S	2	(f, sa-0)	(11, 1, 0, 1)
7	AES.S	2	(sa-1, f)	(11, 1, 0, 1)
8	AES.S	2	(sa-1, sa-1)	(11, 1, 0, 1)
9	AES.S	2	(sa-1, sa-0)	(11, 1, 0, 1)
10	AES.S	2	(sa-0, f)	(11, 1, 0, 1)
11	AES.S	2	(0, 1)	(11, 1, 0, 1)
12	AES.S	2	(sa-0, sa-0)	(11, 1, 0, 1)

Table III gives the area overhead including the FIM and the eight scan chains. The synthesized AES netlist contains 530 DFFs and 1543 LUTs. The DFF overhead is 33.9% and LUT overhead is 29.1%. Note that the relative overhead will be low for large CUTs. [9] is the most related one to our work. We also implement the proposed emulation platform on the FPGA XC3S500E as used in [9]. The result shows that our platform reduces the DFFs by 35.7% and the LUTs by 47.2%, compared to [9]. We also evaluate the performance of the proposed platform. The AES circuit is clocked at 100MHz and takes about 0.12us to complete computations. The RS232 is configured to 115200bit/s and takes about 3.84ms to transfer an input test vector, configuration data and a fault vector. The time to inject a fault is about 1.02us in the scan mode. As a result, the time per fault test case is less than 3.85ms. When high speed external interface such as PCIe and DMA transfer is used, the communication overhead between the PC and the FPGA system can be significantly reduced.

TABLE III. AREA OVERHEAD

Resource	AES	After insertion	Overhead
FF	530	802	272 (33.9%)
LUTs	1543	2177	634 (29.1%)

TABLE IV. LAST ROUND KEY DEDUCED FROM FAULTY OUTPUTS OF AES

Fault Model No.	128-bit Key
No Faults	beef5bcb3e92e21123e951cf6f8f188e
1	beef5bcb3e92e21123e951cf6f8f188e
2	beef5bcb3e92e21123e951cf6f8f188e
3	00ef5bcb3e92e21123e951cf6f8f188e
4	beef5bcb3e92e21123e951cf6f8f188e
5	beef5bcb3e92e21123e951cf6f8f188e
6	beef5bcb3e92e21123e951cf6f8f188e
7	beef5bcb3e92e21123e951cf6f8f188e
8	beef5bcb3e92e21123e951cf6f8f188e
9	beef5bcb3e92e21123e951cf6f8f188e
10	beef5bcb3e92e21123e951cf6f8f188e
11	beef5bcb3e92e21123e951cf6f8f188e
12	00ef5bcb3e92e21123e951cf6f8f188e

The state S of AES is synthesized into 128 DFFs. Thus, the parameter *location* covers 128 DFFs of state S for fault model No.1, No.2 and No.3 in Table II, and 8128 combinations of the 128 DFFs for the others in Table II. All outputs for each fault model are analyzed considering that: (1) whether the right key can be deduced or not; (2) how many faulty outputs the fault injections lead to; (3) which byte of the AES output is more sensitive to faults. Table IV shows the last round keys deduced from the faulty outputs of AES for each fault model. The row with no faults gives the right key of the last round of AES. The results show that the complete key can be obtained, except from fault model No.3 and fault model No.12. For these two fault models, the master byte of the key is not correctly derived. Note that, the two fault models have the same fault type which is the stuck-at 0. We have changed the plaintext and repeated the same procedure. The results show that the key can be derived from all fault models. This

TABLE V. STATISTIC OF THE FAULTS INJECTED AND THE FAULTY OUTPUTS

Fault Model No.	Num of Injected Faults	Num of Faulty Outputs	Error Rate
1	128	128	1.0000
2	128	57	0.4453
3	128	71	0.5547
4	8128	8128	1.0000
5	8128	8128	1.0000
6	8128	8128	1.0000
7	8128	8128	1.0000
8	8128	5643	0.6943
9	8128	6112	0.7520
10	8128	8128	1.0000
11	8128	6097	0.7501
12	8128	6532	0.8036

demonstrates that the plaintext (i.e. the input test vectors) affects the evaluation.

Table V reports the statistics of the faults injected and the faulty outputs. The error rate is defined as the number of faulty outputs over the number of injected faults. High error rate raises the chance to obtain the key of the AES with limited fault injections. From the table we can make the following observations. Firstly, the fault models belonging to the flip fault, such as fault model No.1, fault model No.4-7 and fault model No.10, induce 100% faulty outputs. Secondly, fault model No.2 has the lowest error rate, which is single-bit stuck-at 1 fault. Thirdly, the double-bit fault tends to increase the error rate. Table VI lists the statistics about the errors of each byte of the output. For the same fault, different bytes of the output may demonstrate different error rate. Under fault model No.1 and fault model No.4, all bytes have the same error rate, while the error rate varies significantly under the other fault models. Note that, fault model No.1 and fault model No.4 are the flip fault. Comparing vertically, each byte shows different sensitivities to different fault models. This suggests that the countermeasures may be applied to the week parts first if the comprehensive protection is difficult.

TABLE VI. STATISTIC OF THE FAULTY OUTPUTS BY BYTE

Fault Model No.	**Statistics by byte of the 128-bit Key**															
	16	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**
1	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
2	7	3	2	2	2	2	5	4	2	3	2	4	3	4	6	6
3	1	5	6	6	6	6	3	4	6	5	6	4	5	4	2	2
4	988	988	988	988	988	988	988	988	988	988	988	988	988	988	988	988
5	953	610	353	857	634	334	949	772	438	952	718	608	973	806	842	752
6	743	748	765	953	870	770	923	728	812	934	898	608	979	824	550	280
7	903	766	903	399	622	922	667	724	818	424	538	888	403	690	894	984
8	868	378	253	253	253	253	625	502	253	378	253	502	378	502	747	747
9	656	518	675	359	494	697	601	453	635	360	447	500	391	520	447	274
10	393	868	971	783	866	966	453	768	924	682	838	888	637	672	706	976
11	353	483	329	645	510	307	400	547	369	641	557	500	610	480	557	730
12	127	625	747	747	747	747	378	502	747	625	747	502	625	502	253	253

Based on the above analysis, we can make the following evaluation about the security of the AES circuit design. Firstly, the AES design is not secure against the emulated fault attacks. The key can be easily cracked. Secondly, the fault models with the flip fault and multiple-bits (high strength) show high threats to the design. Therefore, the AES circuit design should be modified with possible countermeasures, especially against the FIA techniques such as electromagnetic pulse injection.

IV. CONCLUSION

This paper proposed an FPGA-based emulation method for security evaluation of IC design against FIAs. The whole flow is automated and is easy to be used. The parameterized fault model allows mimicking a wide range of faults caused by FIAs. The scan chain-based fault injection mechanism and the fault injection management approach reduce area overhead and enables fast fault injection emulation. The emulation platform has been used to evaluate the security of an AES-128 circuit design, validating the proposed evaluation approach.

ACKNOWLEDGEMENT

This work was supported in part by the National Natural Science Foundation of China under No.61574099.

REFERENCES

[1] E. S. Abuelyaman, B. Devadoss, “Differential fault analysis,” Proceedings of The 2005 International Conference on Internet Computing, January 2005.

[2] A. Barenghi, L. Breveglieri, I. Koren and D. Naccache, "Fault injection attacks on cryptographic devices: theory, practice, and countermeasures," Proceedings of the IEEE, vol. 100, pp. 3056-3076, November 2012.

[3] C. Sun, H. Li, Y. Yang, J. Chen, “Research on fault-electromagnetic attack on block cipher,” Journal of Convergence Information Technology, pp.409-417, 2011.

[4] S. D. Carlo, P. Prinetto, D. Rolfo, P. Trotta, “A fault injection methodology and infrastructure for fast single event upsets emulation on Xilinx SRAM-based FPGAs,” IEEE International Symposium on Defect & Fault Tolerance in VLSI & Nanotechnology Systems, pp.159-164, 2014.

[5] E. Sanchez, L. Sterpone, A. Ullah, “Effective emulation of permanent faults in ASICs through dynamically reconfigurable FPGAs,” International Conference on Field Programmable Logic & Applications, pp.2336-2342, 2014.

[6] G. L. Nazar, L. Carro, “Fast single-FPGA fault injection platform,” IEEE International Symposium on Defect & Fault Tolerance in VLSI & Nanotechnology Systems, pp.152-157, 2012.

[7] O. Boncalo, A. Amaricai, C. Spagnol, E. Popovici, “Cost effective FPGA probabilistic fault emulation,” Norchip, pp.1-4, 2014.

[8] P. Civera, L. Macchiarulo, M. Rebaudengo, M. S. Reorda, M. Violante, “An FPGA-Based approach for speeding-up fault injection campaigns on safety-critical circuits,” Journal of Electronic Testing, pp.261-271, 2002.

[9] A. Janning, J. Heyszl, F. Stumpf, G. Sigl, “A cost-effective FPGA-based fault simulation environment,” Fault Diagnosis and Tolerance in Cryptography, pp.21-32, 2011.

[10] A. Papadimitriou, D. Hely, V. Beroulle, P. Maistri, R. Leveugle, “A multiple fault injection methodology based on cone partitioning towards RTL modeling of laser attacks,” Design, Automation and Test in Europe Conference and Exhibition (DATE), pp.1-4, March 2014.

[11] Tao Li, Qiang Liu, “Cost effective partial scan for hardware emulation,” IEEE International Symposium on Field-Programmable Custom Computing Machines, 2015.