A Comparison of TERO and RO Timing Sensitivity for Hardware Trojan Detection Applications

Paris Kitsos[#]
Digital IC Design and Systems Laboratory (DICES Lab)
Computer and Informatics Engineering Department
Technological Educational Institute of Western Greece
Antirrion, Greece
e-mail: pkitsos@teimes.gr

Artemios G. Voyiatzis
Industrial Systems Institute, "Athena" Research and Innovation Center in ICT and Knowledge Technologies
Platani Patras, GR-26504, Greece
e-mail: bogart@isi.gr

***Abstract*—A Ring Oscillator (RO) integrated in a design can be used for detecting insertion of malicious logic i.e., a hardware Trojan horse. Recently, the Transition Effect Ring Oscillator (TERO) was proposed as a means for implementing True Random Number Generators (TRNGs) and Physically Uncloneable Functions (PUFs). In this paper, we explore the timing sensitivity of TERO against RO, towards introducing TERO as an alternative means for detecting Trojans on FPGAs.**

Keywords— FPGA security; time analysis; ring oscillators; Transition Effect Ring Oscillator; hardware Trojan horse.

I. INTRODUCTION

Nowadays, due to globalization of the integrated circuit (IC) manufacturing and the deep supply chains involved in system design, it is very hard to guarantee that a hardware design is free of malicious logic. A hardware Trojan horse (or simply a 'Trojan') refers to the insertion of malicious logic in a design. Depending on the intentions and the resources available to the attacker, a Trojan may destroy the IC, take over its control, or leak sensitive information out of it [1]. Thus, there is an increasing need to integrate defenses in the designs, such as (hardware) Trojan detection mechanisms.

A Ring Oscillator (RO) is a closed-loop chain of an odd number of inverters. An RO oscillates at a fixed frequency depending on the exact components, the size of a circuit, the operating characteristics (e.g., the voltage) and its environment conditions (e.g., the ambient temperature). Even minimal modifications of the circuit can result in a frequency change, rendering the ROs very sensitive to process variations [2-3]. Many research works proposed already the use of an RO for detecting the small modifications caused by the integration of a Trojan in the IC design [4-6].

A Transient (or Transition) Effect Ring Oscillator (TERO) is, in principle, a more sensitive variant of a ring oscillator. The use of a TERO has been proposed for implementing True Random Number Generators (TRNGs) and Physically Uncloneable Functions (PUFs) [7-8].

In this paper, we perform a comparison on the timing sensitivity of TERO against RO, towards introducing TERO as an alternative means for detecting Trojans implanted on FPGAs.

The rest of the paper is organized as follows. In Section II, we discuss RO and TERO implementations in a FPGA. In Section III, we describe the experimental setup and in Section IV, we analyze the results of our experiments. Finally, Section V concludes the paper.

II. RO AND TERO IMPLEMENTATION IN FPGA

A ring oscillator can be realized in the FPGA technology using lookup tables (LUTs) configured as inverters. An example RO with a sequence of five inverters in a closed loop (i.e., an RO with length 5) is depicted in Fig. 1. A counter fetches the output of the RO in order to measure the oscillation frequency. The measurement accuracy is affected by the process variation and the measurement noise. All these factors can account for a variance as high as 6.6% [9].

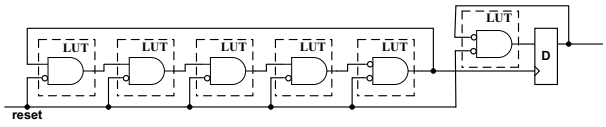


Fig. 1. Ring oscillator in FPGA

A ring oscillator can be used for detecting malicious hardware modifications through frequency changes beyond the acceptable variance. There are proposals for complex RO networks and off-chip transient power analysis [10-11]. The length of the RO can also affect the Trojan detection sensitivity; ROs with different lengths may be able to detect different kinds of Trojans [12].

A TERO can be realized in the FPGA technology as an SR flip-flop implemented with two XOR gates and two AND gates, as illustrated in Fig. 2 (a). This architecture has two control signals, one for the start and one for the reset. The *reset* signal is set constantly to '1' and the TERO resets when the *reset* signal drops to '0'. When the *ctrl* signal is activated, the TERO circuit oscillates.

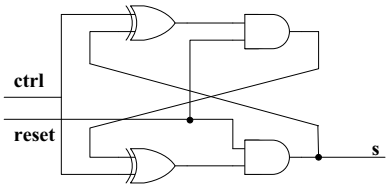
We note that a simpler TERO architecture can be achieved if the XOR and AND gates are merged into NAND gates with some inverters in the feedback loop, as depicted in Fig. 2 (b). The advantage of our proposal is that *only one control signal* is used for either resetting or oscillating the TERO circuit. The reset occurs when the control signal, *ctrl*, is set to '0' and drives the loop to the same initial conditions as before,

Collaborating Faculty with the Industrial Systems Institute, "Athena" RIC in ICT and Knowledge Technologies.

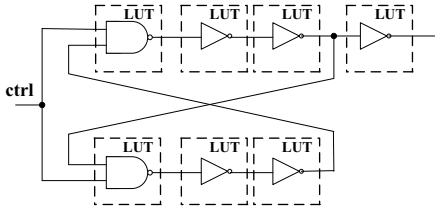
This work was supported by the GSRT Action "KRIPIS" with national and EU funds in the context of the research project "ISRTDI" and by COST Action IC1204 "TRUDEVICE".

978-1-4673-8035-5/15 $31.00 © 2015 IEEE
DOI 10.1109/DSD.2015.32

generating its output. When the control signal changes from '0' to '1', the TERO circuit starts to oscillate and oscillates for a long period. As in the case of the RO described earlier, the output of the TERO is fed to a counter in order to measure the TERO frequency.



(a) TERO using SR flip-flop



(b) TERO using NAND and NOT gates

Fig. 2. TERO architecture

III. Experimental Setup

A cryptographic primitive is an attractive target for a Trojan due to the enormous space of possible inputs (key or plaintext bits) for controlling its activation and the criticality of the information processed. In our experiments, we assume an FPGA implementation of a cryptographic algorithm that is infected with a Trojan.

We opted for experimentation with the self-synchronizing stream cipher Mosquito [14]. Mosquito uses a symmetric key of 96 bits and has an internal memory of 105 bits for self-synchronization. The architecture of Mosquito is depicted in Fig. 3. The multiplexer (MUX) is used to select between using the encryption or the decryption mode.

We designed two Trojans in order to test the capabilities of the RO and TERO designs. The first, namely Trojan1, is a sequential Trojan and acts as a time bomb. Its design is depicted in Fig. 4 (a). The Trojan trigger part comprises a clock-synchronous, free-running counter and a three-input AND gate.

The second, namely Trojan2, is a combinational one and its design is depicted in Fig. 4 (b). The Trojan trigger part comprises a tree of AND gates that monitors the values of a randomly selected subset of 8 out of the 96 key bits (namely, positions 0, 12, 13, 40, 50, 51, 70, and 91).

The Trojan payload part in both cases consists of a XOR gate that drives the signal for defining the mode of operation (encryption or decryption) for the Mosquito cipher. Once the malicious logic is triggered (once every 100 cycles for Trojan1 or when the key bits match the pattern for Trojan2), it inverses the selected mode of operation of the cipher.

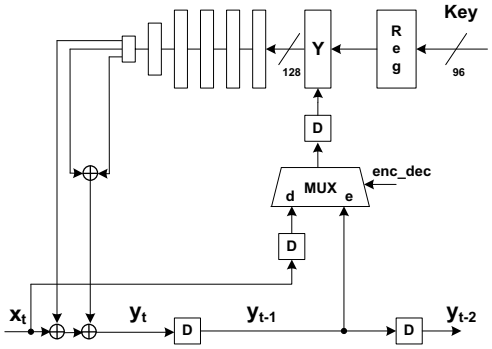
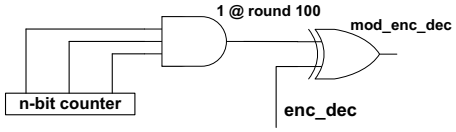
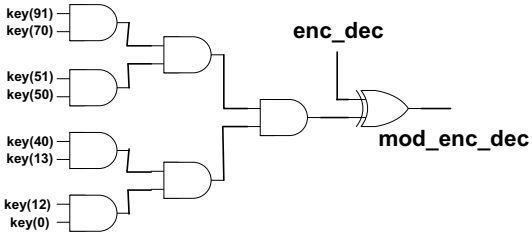


Fig. 3. Mosquito stream cipher architecture



(a)



(b)

Fig. 4. Trojan designs, a) *Trojan1*, b) *Trojan2*

We used a Spartan-6 LX75 FPGA, part of an available SAKURA-G board, for our experiments. The experimental setup consists of a) the Mosquito cipher, b) the RO and TERO components, and c) the two Trojans described in the previous paragraphs.

We used the implementation file extracted by the PleanAhead tool [15] and especially the VHDL code of the Post-Place-and-Route simulation model. This VHDL code contains true timing delay information (by using a Standard Delay File) for the design based on the FPGA device used.

We simulated this model and derived the oscillation frequency for the RO and the TERO considering die delay variations for a fixed temperature of 85 degrees Celsius and a voltage of 1.14 Volts. This deemed necessary because the Spartan-6 family does not support temperature and voltage prorating in the simulation level. We also experimented with the effect of different RO and TERO lengths, as suggested in [12].

The Trojans occupied in general a small percentage of the available FPGA area and in comparison with the area needed for the Mosquito implementation. All the designs were captured in VHDL and were synthesized on the same FPGA board.

Once additional logic is inserted in the design (a longer RO or a Trojan), the placement-and-route tools devise a completely different place-and-route [16]. However, we need efficient, fair, and, most importantly, *accurate and comparable* measurements. To achieve these, we must build designs that share the same place-and-route.

One possible solution for the aforementioned problem is to use *Hard Macros*. However, this is very difficult for large and complex systems. We consider as a better option to utilize *BEL* and *LOC* placement constraints. One point of concern is that this is inefficient due to the large time costs. In our experiments, we performed the following steps through the Xilinx ISE 14.7 (Webpack) tool and achieved the same place-and-route for Mosquito enhanced with either a TERO or an RO in a short time:

1) We use the smaller length for the TERO (i.e., 5) in our source VHDL code.

2) We define the I/O constraints of the design (e.g., Mosquito with integrated TERO). So, a User Constraints file (ucf) was generated with the information of I/Os.

3) We synthesized the VHDL code.

4) We ran the design implementation, which converts the logical design into a physical file format.

5) We ran the PlanAhead tool to automatically place-and-route the design.

6) We then locked the generated place-and-route of the logic using the "fix instances" option. This process updates the ucf file with BEL and LOC constraints.

7) We generated the VHDL code of the Post-Place-and-Route Simulation model.

8) We simulated the new VHDL code (for example with ModelSim tool) in order to measure the TERO frequency.

9) We used the updated ucf file with the next TERO length (i.e., 7) in our source VHDL code, for synthesis and implementation. We note that one should not run the PlanAhead process to fix again the instances (i.e., skip steps 5 and 6).

10) We generated the VHDL code of the Post-Place-and-Route Simulation model for this design (i.e., with a TERO length of 7).

11) We simulated the generated VHDL code (of Step 10) in order to measure the TERO frequency.

12) We repeated the above Steps (9, 10 and 11) for the remaining lengths (i.e., 9 and 11).

The overall output of this procedure is a set of three FPGA designs: a) a *Trojan-free* design that includes only the Mosquito with an RO or a TERO of varying lengths; b) a *Trojan1-injected* design that includes Mosquito, an RO or a TERO, and Trojan1; and c) a *Trojan2-injected* design that includes Mosquito, an RO or a TERO, and Trojan2. As a cautionary note, we remind that for TERO to function properly, the place-and-route must ensure the same length of the interconnections between its NAND gates.

IV. Results

The layouts of the three different implementations for the Mosquito with the RO protection are depicted in Fig. 5. Similarly, the layouts of the three different implementations for the Mosquito with TERO protection are depicted in Fig. 6. It is evident that the same layout (place-route and I/Os) was created in all cases. This means that the hardware resources of the identical parts of the circuits (i.e., the Mosquito and its RO/TERO) were placed and were routed on the same locations on the FPGA.

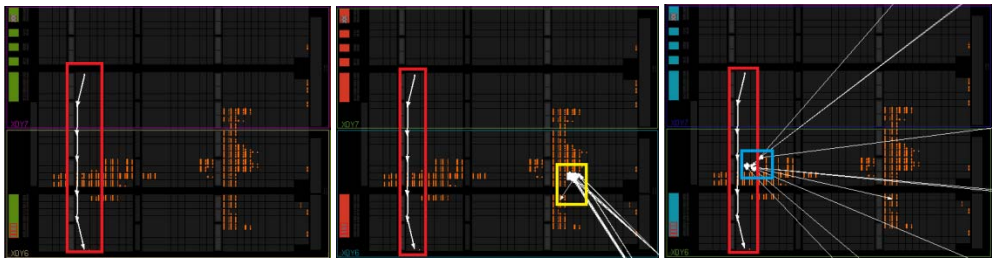


Fig. 5. Implementation layout of (a) the Mosquito cipher and the RO in the big red rectangle on the left, (b) Trojan1 in the small yellow rectangle on the right, and (c) Trojan2 in the small blue rectangle partially overlapping in the area of the RO in the big red rectangle

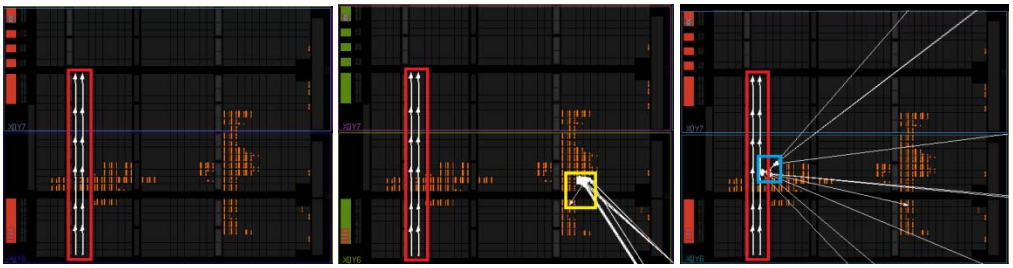


Fig. 6. Implementation layout of (a) the Mosquito cipher and the TERO in the big red rectangle on the left, (b) Trojan1 in the small yellow rectangle on the right, and (c) Trojan2 in the small blue rectangle partially overlapping in the area of the TERO in the big red rectangle

The simulation results in regards to RO and TERO frequency counts are depicted in Fig. 7.

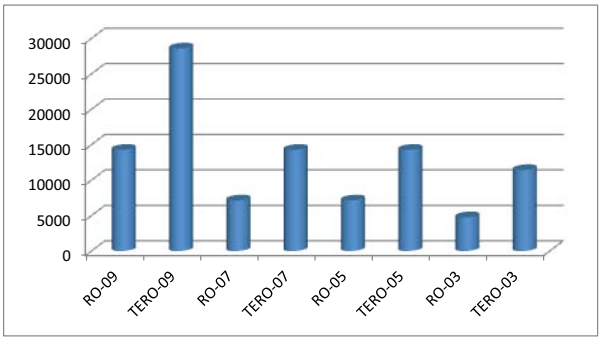


Fig. 7. Comparison between RO and TERO frequency counts

For the measurements, a simulation process of 77.12 μsec was used with 16 different Mosquito keys. The operation frequency was 20 nsec. The RO or the TERO are enabled at the 20th μsec, so as the circuit can gain a constant frequency.

In real operation conditions, the placement of an RO or a TERO on an FPGA also affects its oscillation frequency. However, the VHDL generated by the Post-Place and Route Simulation model on the ISE tool contains only the timing delay information of the design based on the FPGA device we have used, without other parameters for conditions such as voltage, temperature, and placement.

For the smallest length i.e., three, the TERO oscillates in a much higher frequency that is more than double the frequency of the respective RO. As the length increases (i.e., 5 and 7), TERO oscillates approximately on double the frequency of the RO, while for a length equal to 9, it oscillates a bit less than the double frequency. The experimental measurements on an FPGA device reported in the previous paragraphs confirm the theoretical assumption that a TERO will oscillate at least on the double frequency compared to an RO [7].

V. Conclusions

In this paper, a timing analysis on a Spartan-6 FPGA device for a TERO and an RO are given. First, we share detailed information on implementing a TERO on a Spartan-6 FPGA; an effort only occasionally described in the literature until now.

Our experimental setup consists of the Xilinx ISE WebPack and the ModelSim CAD tools. We used Post-Place and Route Simulation models that contain true timing delay information of the design and we described an efficient method to achieve the same placement-and-routing of identical designs with varying RO/TERO chain lengths for fair comparison.

In the future, we will use the available SAKURA-G board to study the applicability of TERO for Trojan detection and its efficiency compared to conventional RO-based approaches.

References

[1] S. Mal-Sarkar, A. Krishna, A. Ghosh and S. Bhunia, "Hardware Trojan attacks in FPGA devices: Threat analysis and effective countermeasures", in Proc. of the 24th edition of the Great Lakes Symposium on VLSI (GLSVLSI 2014), Pittsburgh, Pennsylvania, USA, 2014, pp. 287-292.

[2] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in Proc. of the Design Automation Conference, 2007. DAC '07. 44th ACM/IEEE, San Diego, USA, 2007, pp. 9–14.

[3] J. Guajardo, S. Kumar, G.-J. Schrijen, and P. Tuyls, "FPGA intrinsic PUFs and their use for IP protection," in Cryptographic Hardware and Embedded Systems (CHES 2007), Vienna, Austria, 2007, pp. 63–80.

[4] J. Rilling, D. Graziano, J. Hitchcock, T. Meyer, X. Wang; P. Jones, J. Zambreno, "Circumventing a ring oscillator approach to FPGA-based hardware Trojan detection", Computer Design (ICCD), 2011 IEEE 29th International Conference on , pp.289-292, 9-12 Oct. 2011.

[5] A. Ferraiuolo, X. Zhang, and M. Tehranipoor, "Experimental analysis of a ring oscillator network for hardware Trojan detection in a 90nm ASIC", Computer-Aided Design (ICCAD), 2012 IEEE/ACM International Conference on, pp. 37-42, 5-8 Nov. 2012.

[6] J. Rajendran, V. Jyothi, O. Sinanoglu, R. Karri, "Design and analysis of ring oscillator based Design-for-Trust technique", VLSI Test Symposium (VTS), 2011 IEEE 29th, pp. 105-110, 1-5 May 2011.

[7] M. Varchola and M. Drutarovsky, "New high entropy element for FPGA based true random number generator", in Cryptographic Hardware and Embedded Systems (CHES 2010), Santa Barbara, USA, 2010, pp. 351-365.

[8] L. Bossuet, X, Thuy Ngo, Z. Cherif, and V. Fischer, "A PUF based on a transient effect ring oscillator and insensitive to locking phenomenon", IEEE Trans. on Emerging Topics in Computing, Vol. 2, No. 1, pp. 30-36, March 2014.

[9] A.Maiti, J. Casarona, L. Mchale, and P. Schaumont, "A large scale characterization of RO-PUF", IEEE International Symposium on Hardware Oriented Security and Trust (HOST 2010), pp. 94-99.

[10] X. Zhang, A. Ferraiulo and M. Tehranipoor, "Detection of Trojans using a combined ring oscillator network and off-chip transient power analysis", ACM Journal on Emerging Technologies in Computing Systems (JETC), Volume 9, Issue 3, Article No. 25, September 2013.

[11] J. Rilling, D. Graziano, J. Hitchcock, T. Meyer, X. Wang, P. Jones, and J. Zambreno, "Circumventing a ring oscillator approach to FPGA-based hardware Trojan detection", in Proc. of the International Conference on Computer Design (ICCD), Amherst, MA, 9-12 Oct. 2011.

[12] P. Kitsos and A.G. Voyiatzis, "FPGA Trojan detection using length-optimized ring oscillators", in Proc. 17th Euromicro Conference on Digital Systems (DSD'14), Verona, Italy, 27-29 August, 2014.

[13] R. Karri, J. Rajendran, K. Rosenfeld, and M. Tehranipoor, "Trustworthy hardware: Identifying and classifying hardware Trojans", IEEE Computer, vol. 43, no. 10, October 2010.

[14] J. Daemen and P. Kitsos, "The self-synchronizing stream cipher Mosquito", First Phase of ECRYPT Stream Cipher Project Report 2005/018, 2005, Scandinavian Congress Center, Aarhus, Denmark, 26-27 May 2005.

[15] PlanAhead Design and Analysis Tool, available at: http://www.xilinx.com/tools/planahead.htm

[16] W. Xinmu, S. Narasimhan, A. Krishna, T. Mal-Sarkar, and S. Bhunia, "Sequential hardware Trojan: Side-channel aware design and placement", Computer Design (ICCD), 2011 IEEE 29th International Conference on, pp. 297-300, 9-12 Oct. 2011.