2014 IEEE Students' Conference on Electrical, Electronics and Computer Science

Detection of Hardware Trojan in SEA Using Path Delay

Prasanna Kumar
Dept. of Electronics & Communication Engineering
RMK Engineering College
Chennai, India
prasannavlsi@yahoo.com

Ramasamy Srinivasan
Dept. of Electronics & Communication Engineering
RMK Engineering College
Chennai, India
srs.ece@rmkec.ac.in

Abstract— **Detecting hardware Trojan is a difficult task in general. The context is that of a fabless design house that sells IP blocks as GDSII hard macros, and wants to check that final products have not been infected by Trojan during the foundry stage. In this paper we analyzed hardware Trojan horses insertion and detection in Scalable Encryption Algorithm (SEA) crypto. We inserted Trojan at different levels in the ASIC design flow of SEA crypto and most importantly we focused on Gate level and layout level Trojan insertions. We choose path delays in order to detect Trojan at both levels in design phase. Because the path delays detection technique is cost effective and efficient method to detect Trojan. The comparison of path delays makes small Trojan circuits significant from a delay point of view. We used typical, fast and slow 90nm libraries in order to estimate the efficiency of path delay technique in different operating conditions. The experiment's results show that the detection rate on payload Trojan is 100%.**

Keywords— **GDSII, Hardware Trojan horses (HTH), HTH detection and insertion, Scalable Encryption Algorithm (SEA), path delay, payload Trojan.**

I. INTRODUCTION

With the advent of electronic commerce and portable devices for communications, cryptology has become exceedingly important science in the present day. Indeed, remote and secure data access requires the use of appropriate security methods. Modern cryptography therefore responds to this need for security but its adapted integration in the wide variety of communication systems has opened new design challenges. Cryptographic circuits are vulnerable to various side-channel attacks that target their hardware implementations to extract secret information stored inside them. Hardware Trojan Horses (HTHs or Trojan) are malicious design modifications intended to cause the design to function incorrectly.

Physical attacks which target the implementation of cryptographic circuits (in smartcards, pay-TV and SIM cards, etc.) have been known for some years now. They are widely classified as "observation" and "perturbation" attacks. Observation or side channel attacks (SCA) consist in observing physical emanations of the system, like power (Differential Power analysis, or DPA [13]) or E=H field (Electromagnetic Analysis, or EMA [14]). Thereafter statistical tools are deployed to find dependency between the predicted and observed behavior.

Perturbation or fault attacks consist in the injection of faults during the execution of a cryptographic algorithm. From the knowledge of one or multiple couples (correct cipher text, faulted cipher text), some hypotheses on the secret key can be discarded. This generic attack strategy is referred to as DFA (Differential Fault Analysis). DFA is very effective against some cryptographic algorithms. For example in AES, the number of faulty cipher text required to break the key can be as low as two. There are several techniques known for fault injection in a system. The variations of the supply voltage, over clocking, temperature increase, or the irradiation by a laser beam will most probably lead to a wrong computation result that can be exploited to realize a DFA. This kind of attack represents a real threat for the implementation of cryptographic algorithms such as the AES.

In this work we propose to study the interest of hardware Trojan insertion and detection on Scalable Encryption Algorithm (SEA). SEA is a scalable encryption algorithm targeted for small embedded applications. It was initially designed for software implementations in controllers, smart cards or processors. But in [1] the authors presented the importance in ASIC implementation of SEA. So, we targeted mainly on two levels to insert Trojan in the ASIC design flow of SEA crypto. One is Gate-level and another one is Layout-level. Compare to Gate-level the attacker has more possibility to insert Trojan at Layout-level. Due to the advancement of reverse engineering process layout of a chip can be acquired from GDSII.

In this context we have taken path delay side channel of SEA crypto in order to detect hardware Trojan at both Gate-level and layout-level by using Fingerprint concept [6].

The rest of paper is organized as follows: Section 2 describes the SEA algorithm and its generic loop implementation. In Section 3 we discussed about VLSI design flow at abstraction level along with possible Trojan insertion levels. Based the testing procedure, the experimental setup steps are introduced in Section 4. Section 5 presents our experimental results. Finally, Conclusion and future work drawn in Section 6.

978-1-4799-2526-1/14/$31.00 ©2014 IEEE

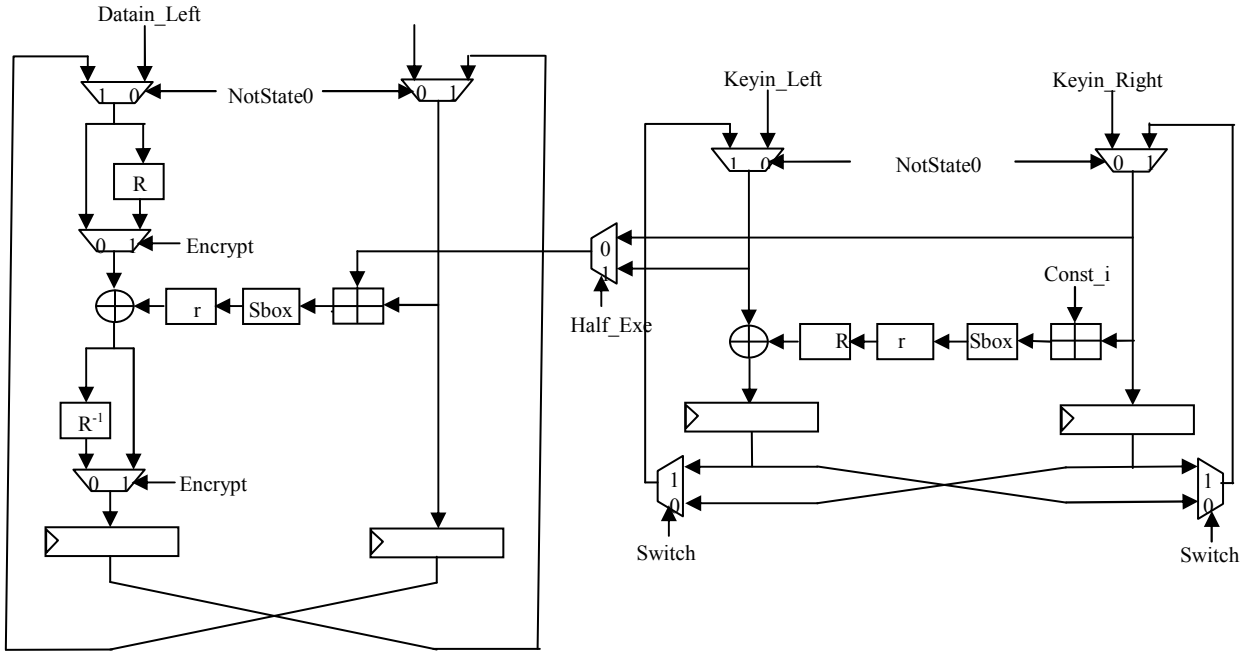

Fig.1. Generic loop architecture of SEA [1].

II. SEA ALGORITHM DESCRIPTION

SEA is a parametric block cipher for resource constrained systems (e.g. sensor networks, RFIDs) that has been introduced in [2]. It was initially designed as a low-cost encryption/authentication routine (i.e. with small code size and memory) targeted for processors with a limited instruction set (i.e. AND, OR, XOR gates, word rotation and modular addition). The algorithm takes the plaintext, key and bus sizes as parameters.

In this section we give a complete description of SEA algorithm and its loop implementation.

A. Basic Operations and Parameters

$SEA_{n,b}$ operates on various text, key and word sizes. It is based on a Feistel structure with a variable number of rounds, and is defined with respect to the following parameters:

n: plaintext size, key size.
b: processor (or word) size.
$n_b = n/2b$: number of words per Feistel branch.
n_r : number of block cipher rounds.

As only constraint, it is required that n is a multiple of $6b$. For example, using an 8-bit processor, we can derive a 96-bit block ciphers, denoted as $SEA_{96,8}$ [2].

SEA is based on a limited number of operations denoted as follows:

(1) Bitwise XOR $\oplus$
(2) Addition mod 2^b $\boxplus$
(3) 3-bit substitution box S :={0,5,6,7,4,3,1,2}
(4) Word rotation R on n_b -word vectors:
$R: yi_{+1}=x_i, \quad 0 \le I \le n_b\text{-}2$
(5) Bit rotation r on n_b-word vectors:
$r: y_{3i} = x_{3i} >>> 1,$
$y_{3i+1} = x_{3i+1},$
$y_{3i+2} = x_{3i+2} <<< 1, \quad 0 \le I \le n_b\text{-}2$

Where >>> and <<< represents the cyclic right and left shifts inside a word.

B. The Round and Key Round

Based on the previous definitions, encrypt round F_E, decrypt round F_D and key round F_K defined as:

$$[L_{i+1},R_{i+1}] = F_E(L_i,R_i,K_i) \leftrightarrow R_{i+1} = R(L_i) \oplus r(S(R_i \boxplus K_i))$$
$$L_{i+1} = R_i$$
$$[L_{i+1},R_{i+1}] = F_D(L_i,R_i,K_i) \leftrightarrow R_{i+2} = R^{-1}(L_i \oplus r(S(R_i \boxplus K_i)))$$
$$L_{i+1} = R_i$$
$$[KL_{i+1},KR_{i+1}] = F_K(KL_i,KR_i,C_i) \leftrightarrow$$
$$KR_{i+1} = KL_i \oplus R(r(S(KR_i \boxplus C_i)))$$
$$KL_{i+1} = KR_i$$

C. Generic Loop Architecture

Loop architecture implementation of SEA introduced in [1]. Unlike in [2] this loop architecture will support both encryption and decryption and executes one round per clock cycle. In this implementation the round function and key schedule do not share any resources. This loop architecture has benefits for FPGAs compare to [2] architecture. The structure of generic loop architecture of SEA is shown in Figure 1.

III. ABSTRACTION LEVELS-TROJAN INSERTION

The semiconductor industry has spread across borders in the time of globalization. Different design phases of an Integrated Circuit (IC) may be performed at geographically dispersed locations. This coupled with the outsourcing design and fabrication to increase profitability has become a common trend in the semiconductors industry. However, this business model comes with an ample scope of introducing malicious behavior to a part of the IC [3].

This malicious hardware is very tough to find at functional testing level. Because, the Trojan size is very less compare total chip area and it mostly never invoke for test vectors. So, we need to depend on side channels like power, delay and Electromagnetic radiation in order to detect this malicious hardware. Even though there are many Trojan detection methods are available the path delay is efficient technique for Gate-level and Layout-level Trojan insertions. We have taken the path delay to detect Trojan due to its efficiency [4].

In [8] authors classified the Trojan depend on its physical, activation and action characteristics. Trojan taxonomies that group Trojans based on their triggering and leaking mechanisms have also been developed. All of these taxonomies assume that hardware Trojans are inserted only at the fabrication phase; however, they can be inserted at other phases and have different functionalities. In [9] authors propose a new taxonomy that has a broader set of attributes. They classify Trojans according to *insertion phase, abstraction level, activation mechanism, effects* and *location.*

Here we discussed about abstraction level which gives the clear idea about possible insertion of Trojan levels in the VLSI design flow. Figure 2 shows the abstraction level of design flow along with Trojan insertion at different levels.

1) At the *system level* different hardware modules, interconnections and communication protocols used are defined. At this level, the Trojans may be triggered by the modules in the target hardware.

2) A typical *development environment* includes synthesis, simulation, verification and validation tools. The CAD tools and the scripts have been used to insert Trojans [10]. Software Trojans inserted in these CAD tools may mask the effects of the hardware Trojans.

3) At the *RT level* each functional module is described in terms of registers and signals. A trojan can be easily designed at the RT level as confirmed by the results to be discussed later.

4) At the *gate level* the design is represented as an interconnection of logic gates. This level allows an attacker to carefully control all aspects of the inserted trojan including size and location.

5) Transistors are used to build logic gates. This level gives the trojan designer control over circuit characteristics like power and timing. Individual transistors can be inserted or removed, altering the circuit functionality [11]. Transistor sizes can be modified to alter circuit parameters [11]. This is a very sophisticated attack, still in the trusted zone with difficult physical access.

6) At the *layout level*, the dimensions and locations of all circuit components are described. This is the concrete level of the design where a trojan can be inserted. Trojans may be inserted by modifying the wire sizes, distances between circuit elements and re-assigning metal layers. Physical access is easier because of the untrusted zone. However, this hack has the highest level of sophistication.

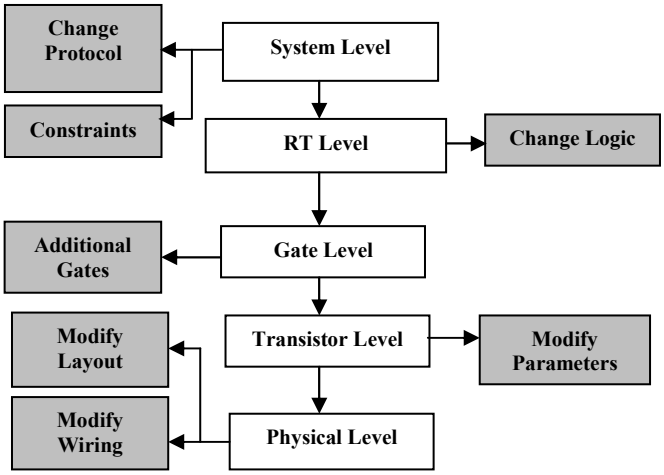

Fig.2. Abstractions in a VLSI design flow. The colored boxes on either side show the example Trojans in that level.

HTH detection is an extremely challenging problem; traditional structural and functional tests do not seem to be effective in targeting and detecting HTHs. Since HTH can be introduced during different design phases, the nature of HTH differs from one design phase to the others. Therefore it is difficult to find a unique detection technique for all HTH. For instance Automatic Test Pattern Generation (ATPG) methods which are used in manufacturing test for detecting defects generally operate on the netlist of the HTH-free circuit. Existing ATPG algorithms cannot target HTH activation/detection directly [11] because HTH are designed such that they are silent most of their lifetime and have very small size relative to their host design, with featuring limited contribution into design characteristics. Such HTHs are most likely connected to nets with low controllability and/or observability[11],[12].

IV. EXPERIMENTAL SETUP

A. Faraday 90nm Technology Library

The FSD0A _A library from Faraday used in this work is a 90nm standard cell library for UMC's 90 nm logic SP-RVT (Low-K) process, where SP1 stands for "Standard Performance" and RVT for "Regular Voltage Threshold". The Low-K term refers to the small dielectric constant (*k*) of the material that has been used to replace the silicon dioxide in the manufacturing process. This substitution is aimed to reduce parasitic capacitance, enable faster switching and get lower heat dissipation.

We synthesized our design at different operating conditions in order to estimate the efficiency of path delay technique to detect Trojan. Table 1 shows the different operating conditions in Faraday 90nm library along with supply voltages and Temperatures Minimum (Fast-Fast), Typical (TT) and Maximum (Slow-Slow), libraries.

TABLE I. DIFFERENT OPERATING CONDITIONS

Symbol	Minimum	Typical	Maximum	units
VCC	0.9	1.0	1.1	V
	-40	25	125	°C

B. Target Circuit

A synthesized SEA IP core is our target circuit. The SEA design is an area optimized sequential design. It takes one round per one cycle. The top module contains three modules key round along with encryption and decryption rounds. We use generic loop architecture of SEA in order to increase benefits. Both plaintext/cipher text are 144-bit wide.

The generic loop architecture of SEA has equivalent area of 9237 NAND gates under 90nm, 1 V technology library from Faraday Technology Corporation. The maximum clock frequency 253MHz. SEA algorithm needs to perform many rounds compare to AES. Example, for 144-bit Plaintext SEA needs 135 rounds. But AES needs standard 10 rounds for 128, 196 and 256 block sizes.

C. Trojan Circuits

In this work we mainly concentrate on combinational type Trojan. The combinational comparator is used to compare four internals and alters the two signal values if Trojan triggers. In the payload phase, two extra OR gates will be inserted in the path. The Bypass Trojan also compares the four internal signals and passes a signal of plaintext to the output without rounding. Here the payload is a 2:1 MUX which select signal is trigger by the comparator. The Trojan used to disclose secret information inside the chip to hackers, the plaintext, for example.

In our experiment, a total of three Trojan are used. The delay fingerprint, different from the power fingerprint, will be affected not only by the type of Trojan but also from its position in the nominal circuit [5]. First one is a comparator Trojan located at the output stage of encryption round. This Trojan with an equivalent area of 6 NAND gates occupies only 0.01% of the total circuit area. The Bypass Trojan occupies only 0.11% of the total chip area.

D. Fingerprint Generation

We use Synopsys Design Compiler Tool to synthesize our SEA circuit without Trojan circuits to get the gate level netlist. For the sake of simplicity we generate netlist for 18-bit SEA. After that, we modify the netlist by inserting Trojan circuits. This is the step a hacker will probably do when he has access to the manufacturing process but not to the RTL code, which is not provided to the manufacturing factory. The netlists with and without Trojan circuits are then loaded into Synopsys Prime Time with technology libraries to generate Standard Delay Format (SDF) which gives detailed delay information of each gate in the design.

To insert Trojan at layout level after getting netlist we had taken that netlist into Cadence SoC Encounter for Place & Route. Then we insert Trojan in layout without interrupting the design by using the gaps between standard cells. After that we take that post layout netlist into Design Compiler Tool to generate SDF which gives the delay information of both gates and interconnections.

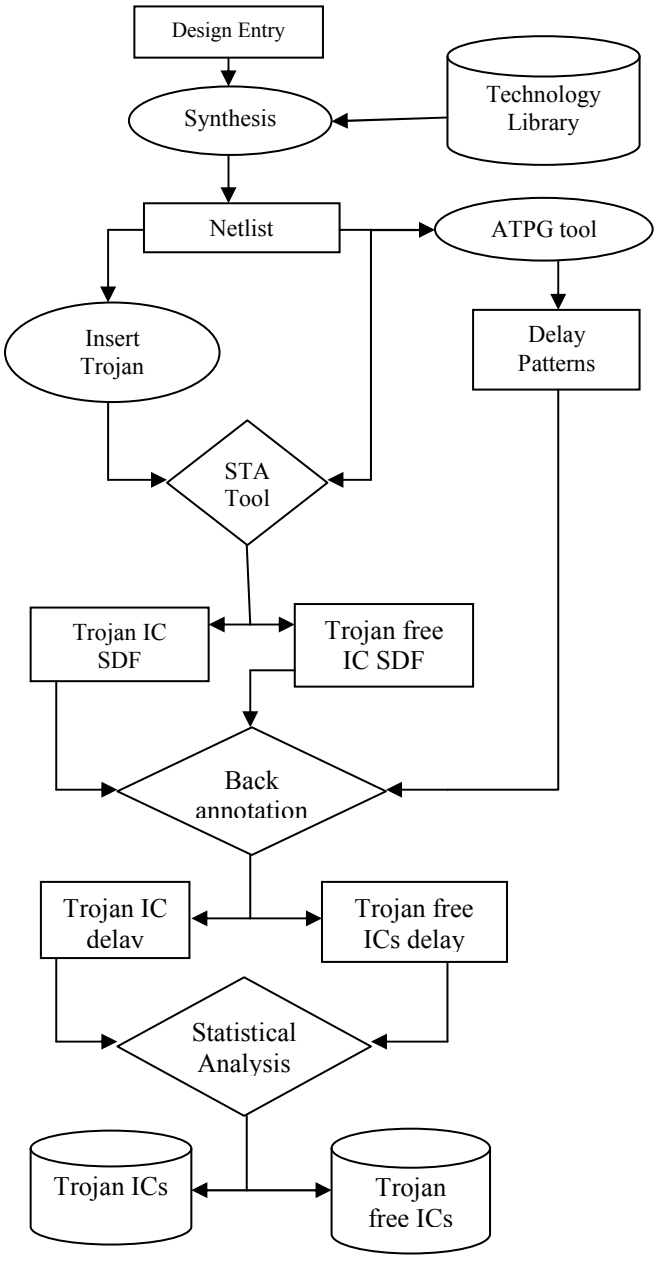

Fig.3. Flow chart of experimental setup.

E. Delay Test patterns

Test patterns are the key part in collecting path delay information for both nominal circuits and circuits with Trojan in our experiments. The goal of generating test patterns is to cover as many parts of the whole chip as possible, such that the delay fingerprints will represent the full chip's characteristics and any change in the chip can be detected easily. We used Synopsys TetraMAX ATPG Tool to analyze the netlist and then generate the delay test patterns. Since the fingerprint reflects the nominal chips, only the genuine netlist is loaded into TetraMAX and a total of 218 test patterns are generated to be the basis of our testing of path delays and chip fingerprints. The coverage of the chip corner cases are 100%, i.e., these 312 test patterns can detect every corner case of the SEA design. As we mentioned before, the path delays are used to generate the fingerprint of nominal chips, so every path in the chip should be included to be part of the chip fingerprint. The SDF files generated by Prime Time are back-annotated into test benches. All 312 test patterns are simulated to generate the delay information.

Since the delay information we collected for each chip contains a high number of dimension (312x143), it is necessary to reduce the dimensionality before we deal with the data. For this purpose we need to use some statistical methods like Principle Component Analysis (PCA) or Quick hull algorithm [7]. Experimental flow chart is shown in Figure 3.

V. Experimental Results

A. Trojan 1: Comparator

This type of Trojan compares the four internal signals and inverts the two output values when Trojan is trigged. We inserted this Trojan at the output stage of encryption module. In our experiment Trojan compares the 4^{th}, 5^{th}, 6^{th}, 7^{th} bits of encryption data and tries to change the values of 5^{th} and 6^{th} bits.

Tables show the finger prints of Trojan and Trojan free circuit paths. For compatibility here we show only the affected paths. Increase in the path delay will shows the Trojan affected data paths.

Table 2 shows the path delays at FF operating conditions. V= 0.9 Volts, Temperature = -40 °C

TABLE II. Path Delay With & Without Trojan (**Fast-Fast**)

Data path	Enc_data [4]	Enc_data [5]	Enc_data [6]	Enc_data [7]
W/O Trojan	5063ps	5063ps	5063 ps	5063ps
W/T Trojan	5069ps	5143ps	5096 ps	5068ps
Increased delay	*6 ps*	*80 ps*	*33 ps*	*5 ps*

Table 3 shows the path delays at TT operating conditions. V= 1 Volt, Temperature = 25 °C

TABLE III. Path Delay With & Without Trojan (**Typical-Typical**)

Data path	Enc_data [4]	Enc_data [5]	Enc_data [6]	Enc_data [7]
W/O Trojan	5099ps	5099ps	5099 ps	5099ps
W/T Trojan	5108ps	5226ps	5150 ps	5105ps
Increased delay	*9 ps*	*127 ps*	*51 ps*	*6 ps*

Table 4 shows the path delays at FF operating conditions. V= 1.1 Volts, Temperature = 125 °C

TABLE IV. Path Delay With & Without Trojan (**Slow-Slow**)

Data path	Enc_data [4]	Enc_data [5]	Enc_data [6]	Enc_data [7]
W/O Trojan	6173ps	6174ps	6173 ps	6174ps
W/T Trojan	6186ps	6392ps	6260 ps	6185ps
Increased delay	*13 ps*	*218 ps*	*87 ps*	*11 ps*

B. Trojan 2: Bypass Trojan (MUX)

Unlike Trojan 1 it bypass the one of the plaintext signal to the encryption data output. This Trojan contains a comparator which compares the four internal signals. Here the payload is MUX which have the select line as output of comparator.

In our experiment we located payload at the output of encryption data [0] bit signal. Table 2 shows the path delays with and without Trojan. Here the delay decreases. Because when Trojan is active the MUX allows the plaintext 0^{th} –bit directly to the output.

TABLE V. Path Delay With & Without Trojan (**Fast-Fast**)

Data path	Enc_data[0]	Delay difference
W/O Trojan	5063 ps	
Trojan Active	1117 ps	*-3946 ps*
Not Active	5104 ps	*41 ps*

TABLE VI. Path Delay With & Without Trojan (**Typical-Typical**)

Data path	Enc_data[0]	Delay difference
W/O Trojan	5099 ps	
Trojan Active	1174 ps	*-3925 ps*
Not Active	5163 ps	*64 ps*

TABLE VII. Path Delay With & Without Trojan (**Slow-Slow**)

Data path	Enc_data[0]	Delay difference
W/O Trojan	6174 ps	
Trojan Active	1327 ps	*-4847 ps*
Not Active	6288 ps	*114 ps*

C. *Trojan 3: Trigger-Payload Trojan:*

This Trojan is simple one which explains the concept of Trigger and payload type Trojan. Here the trigger is a NOR gate and payload is a XOR gate as shown in Figure 4. Here the trigger is connected parallel with a AND gate and these two gates output given as inputs to the XOR gate. This Trojan simply alters the output signal of AND gate.

The concept is the NOR gate will trigger when the AND input is '00'. This '00' pattern never exist in test vector of a AND gate.

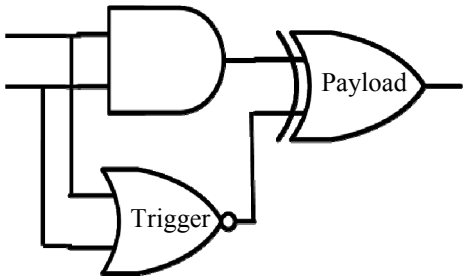

Fig.4. Trigger payload Trojan example

VI. CONCLUSION AND FUTURE WORK

In this work we presented the possibility of insertion Trojan in SEA crypto at Gate-Level and Layout-level. Also we estimated the Trojan detection efficiency of path delay technique in architectures like SEA. By these results we proved that path delay technique is efficient to detection Trojan at different operating conditions (Typical, fast and slow). But, the increased delays are very small to detect at real time. This is the limitation of path delay technique.

For our future work, first we would like to improve the Trojan detection efficiency by talking the number of transitions at critical nodes along with the path delay. Second, we need to verify the possibility of Trojan insertion on SEA crypto at Register-Transfer Level (RTL).

REFERENCES

[1] F. Mace, F.-X. Standaert, and J.-J. Quisquaterr, "ASIC Implementations of the Block Cipher SEA for Constrained Applications". In Proc. of RFIDSEC'07, pp. 103-114, Malanga, Spain, 2007.

[2] F.-X. Standaert, G. Piret, N. Gershenfeld, J.-J. Quisquater, "SEA: A Scalable Encryption Algorithm for Small Embedded Applications". In Proc. of CARDIS 2006, LNCS , pp 222-236, Tarragona, Spain, April 2006.

[3] S. Bhasin, J-L. Danger, S. Guilley, Xuan Thuy Ngo and Laurent Sauvage, "Hardware Trojan Horses in Cryptographic IP Cores". In Proc. Of Fault Diagnosis and Tolerance in Cryptography (FDTC), 2013, pp. 15-29.

[4] Yier Jin and Yiorgos Makris, "Hardware Trojan Detection Using Path Delay Fingerprint". In proc. Of Hardware-Oriented Security and Trust (HOST), 2008,June, pp. 51 – 57.

[5] Yier Jin and Yiorgos Makris, "Hardware Trojan in Wireless Cryptographic ICs". In Proc. Of Design & Test of Computers, IEEE (Volume:27), Jan.-Feb. 2010, pp.26 – 35.

[6] Dakshi Agrawal, Selcuk Baktir, Deniz Karakoyunlu, Pankaj Rohatgi, and Berk Sunar, "Trojan detection using ic fingerprinting," in Security and Privacy, 2007. SP '07. IEEE Symposium on, 2007, pp. 296–310.

[7] C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa, "The quickhull algorithm for convex hulls," *ACM Trans. Math. Softw.,* vol. 22, no. 4, pp. 469–483, 1996.

[8] M. Tehranipoor and F. Koushanfar, ''A survey of hardware Trojan taxonomy and detection,'' *IEEE Design Test Comput.,* pp. 10–25, Jan.–Feb. 2010.

[9] Rajendran et.al, "Trustworthy Hardware: Identifying and Classifying Hardware Trojans", In proc. of IEEE Computer Society,2010, vol.43, pp.39-46.

[10] J. A. Roy, F. Koushanfar, and I. L. Markov, "Extended abstract: Circuit cad tools as a security threat", In Proc. IEEE Workshop on Hardware Oriented Security and Trust, pages 65–66, June 2008.

[11] Xiaoxiao Wang, M. Tehranipoor, and J. Plusquellic, "Detecting malicious inclusions in secure hardware: Challenges and solutions", In Hardware Oriented Security and Trust, 2008. HOST 2008. pp. 15–19, June 2008.

[12] Mainak Banga and Michael S. Hsiao, "A Novel Sustained Vector Technique for the Detection of Hardware Trojans". In Proceedings of the 2009 22nd International Conference on VLSI Design, VLSID '09, pages 327–332, Washington, DC, USA, 2009. IEEE Computer Society.

[13] Paul C. Kocher, Joshua Jaffe and Benjamin Jun, "Differential Power Analysis". In Proceedings of CRYPTO'99, volume 1666 of LNCS, pages 388–397. Springer-Verlag, 1999.

[14] Jean-Jacques, Quisquater and David Samyde. "ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smard Cards". In I. Attali and T. P. Jensen, editors, Smart Card Programming and Security (Esmart 2001), volume 2140 of LNCS, pages 200–210. Springer-Verlag, September 2001. Nice, France.

[15] S. Adee, ''The Hunt for the Kill Switch,'' IEEE Spectrum, vol. 45, no. 5, 2008, pp. 34-39.

[16] TrustHub: http://trust-hub.org/