Detection of Trojan HW by using hidden information on the system

O. Keren*, I. Levin** and V. Sinelnikov*
* Bar Ilan University/School of Engineering, Ramat Gan, Israel
** Tel Aviv University/School of Education, Tel Aviv, Israel
ilia1@post.tau.ac.il, kereno@eng.biu.ac.il, sinel@zahav.net.il

Abstract – A Trojan horse is a malicious altering of hardware specification or implementation in such a way that its functionality is altered under a set of conditions defined by the attacker. The paper presents a technique for designing secure systems that can detect an active Trojan. The technique is based on utilizing specific information about the system's behavior, which is known to the designer of the system and/or is hidden in the functional specification of the system. A case study of the proposed technique conducted on an arithmetic unit of a microprocessor is provided. The study indicated a high level of Trojan detection with a small hardware overhead.

1. Introduction

Trojan horse hardware is a malicious modification of the circuitry of an integrated circuit. A Trojan hardware is created by an adversary that adds or deletes some transistors or gates to the original design and thus changes its. Since the system functions correctly when the Trojan is not activated, it is difficult to detect a Trojan by off-line testing. This motivates researchers to classify, model, and analyze Trojans and their effect on the behavior of the system, as well as to develop methods to detect a Trojan HW [3]. Various detection methods can be applied at different levels of chip design and fabrication [5]. Our approach is based on on-line testing [2]. We suggest utilizing a Concurrent Error Detection (CED) mechanism, which is normally used to cope with faults (permanent or transient) in the circuitry to detect active Trojans.

In general, there are three approaches to the design of CED schemes: the first is based on analyzing all the possible error events in a given circuit (e.g. [4]), the second is based on analyzing the possible combinations that may appear on the output lines of each logic block, and the third approach is based on analyzing the functionality of the whole system. Clearly, the first two methods are more suitable for trapping errors that are caused by faults, since a) they heavily relay on the actual implementation of the combinatorial circuit and b) they check the correctness of the current output. The third approach, which considers the functionality of the system, can detect an *incorrect sequence of outputs*, even if each output by itself is legal. The CED mechanism presented in this paper belongs to the third class of CEDs.

The main idea of this paper is that there is implicit information about the correct functionality of the system that is not necessary included in an initial specification of the system. This information is known to the designer, and in some cases can be retrieved from the functional specification of the system. However, this information disappears or become inaccessible when the system is represented in HDL and synthesized by conventional CAD tools. Next we show how such information can provide additional layer of security.

2. Case study

In general, any system has components that can be classified either as parts of the datapath (DP) or as control unites (CUs). An active Trojan can interfere or manipulate the correct operation of the system in two ways. It may directly distort the outputs of the control unit (that is, modify the microinstructions or the next state variables), or/and it may cause the control unit to generate different outputs by changing the inputs of the control unit (see Fig. 1).

We assume that the Trojan HW does not generate (for at least a period of time) a legal sequence of input/outputs vectors. We also assume that there is no mandatory requirement that the Trojan will be detected immediately at the first cycle it was activated. Therefore, it is acceptable to protect only a sub-set of the possible legal input and output words.

In this paper, we add on top of the layer that detects erroneous external inputs that enter the control unit or the datapath, another layer of security that detects an erroneous operation of the block itself. The proposed approach uses a representation of the initial CU in a form of Algorithmic State Machine (ASM) chart since such description allows to extract the hidden information in a very efficient way [1]. An ASM consists of a set of condition and operator vertexes. Each ASM operator vertex is performed during a single clock. In general, all input variables of the CU can change their values during a clock. Nevertheless, in real designs, usually it is not true - very often, a large subset of input variables cannot change their values. ASMs do not contain this information explicitly. This hidden information which is known to the developer can indicate the presence of Trojan. Namely, we suggest to associate each operator is with a set of variables (input variables or state variables) that cannot be changed as a result of this operation. Obviously, comparison of these variables before and after the operator gives valuable information about the correctness of system's operation.

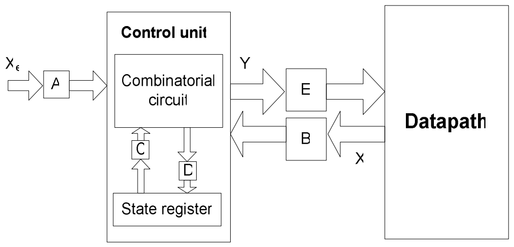

Figure 1: System with Trojan blocks. The effect of a Trojan on the external and internal inputs, the state variables and the outputs is modeled by blocks A, B, C, D, and E, respectively.

We studied the design of the Trojan detection mechanism for a CPU having 16 8-bit registers and two embedded 16-bit memories for data and instructions. The CPU supports short and long formats and five addressing modes. We concentrated on the ASM of the *add*, *subtract* and *shift* arithmetic instructions. This ASM defines and uses:

- 13 inputs (4 internal and 9 external) inputs out of 18 inputs to the whole CU that supports the complete set of instructions.
- 27 microinstructions, each consists of 45 micro-operations.
- 27 states coded as 1-hot.

We have added 10 check vertexes and two operator vertexes to the ASM to verify the correct behavior of the system at each step. These vertexes, which form the Trojan detection mechanism, check (on average) 31 out of the 40 inputs (state variables and internal inputs). This results a detection of 85% of possible unwanted changes in the states of the system, and detecting of 22% of possible (unwanted) changes in the inputs to the CU (refer to blocks B-D in Fig. 1).

The implementation cost of the Trojan detection mechanism is shown in Table 1 and measured in terms of the number of slices, LUTs and Flip-Flops in a Xilinx Spartan3a, XC3S1400An. In the table, we present the original size of the whole system (DP and CU), the size of the original CU, the CU additional logic to be included to allow CED of the microinstructions sent to the DP. Columns 5 and 6 show the additional logic that supports the check of validity of the inputs (denoted as XCh), and the state variables (denoted as state Ch). The last column shows the cost of the CED checker located right in front of the DP. The additional logic required to protect the states and the inputs of the CU, which may be distorted by a Trojan modeled by blocks B-D in Fig. 1 is 24% of the cost of the CU (when measured in slices) and is 0.1% of the cost of the whole system. The additional logic required to protect the whole system against a Trojan modeled by blocks B-E in Fig. 2, is 3% of the cost of the whole system.

TABLE 1: OVERHEAD IN CU DESIGN

	DP + CU	Orig. CU	CU + coded Y's	X Ch.	state Ch.	Y Ch.
Slices	7293	29	31	2	5	205
FF	8577	49	49	3	3	378
LUTs	5632	33	57	3	10	0

3. CONCLUSIONS

The paper suggests use an ASM based description of a system for designing a Trojan detection mechanism. The case study presented in the paper shows a high Trojan coverage with relatively small hardware overhead.

Our future research in the described direction includes development of methods and algorithms for automation of the process the hidden information and methods for synthesis of systems with Trojan detection capability.

References

[1] Baranov S. *Logic and System Design of Digital Systems,* TUT press, 2008.
[2] P. Lala *Self-checking and Fault-Tolerant Digital Design*, Morgan Kaufmann Publishers, 2000.
[3] M. Tehranipoor, F. Koushanfar, "A Survey of Hardware Trojan Taxonomy and Detection", *IEEE Design and Test of Computers*, vol. 27, no. 1, pp. 10-25, 2010 .
[4] R. Vemu, A. Jas, J. A. Abraham, S. Patil, and R. Galivanche, "Low-Cost Concurrent Error Detection Technique for Processor Control Logic", *Proc. of Design, Automation and Test in Europe (DATE*), pp. 897-902, 2008.
[5] W. Xiaoxiao, M. Tehranipoor, J. Plusquellic, "Detecting Malicious Inclusions in Secure Hardware: Challenges and Solutions," *1st International Workshop on Hardware-Oriented Security and Trust,* pp. 15-19, 2008.