Fragile HFSM Watermarking Hardware IP Authentication CAD Tool

Samar M. Hussein Shukry[1], Amr T. Abdel-Hamid[1] and Mohamed Dessouky[2]
[1]Faculty of Information Engineering and Technology IET, German University in Cairo (GUC), Egypt.
[2]Electronics and Communications Engineering Department, Ain-Shams University, Egypt.

***Abstract*— Hardware Trojans are intrusive actions that tend to corrupt the functional behavior of genuine designs, leak their sensitive information, or even downgrade their performance. This paper proposes a Computer-Aided Design (CAD) tool for Fragile Hierarchical FSM Watermarking to be employed as a novel presilicon Trojan detection technique at the RTL level. The performance of the proposed watermarking tool, regarding its complexity and insertion time overhead, is evaluated. Moreover, an intentional Trojan insertion scenario is executed in order to verify the sensitivity and reliability of the presented tool against the least changes in the original RTL design.**

Keywords—Tampering Protection, Hardware Trojan Detection, HDL Analysis, Fragile Watermarking, HFSM.

I. INTRODUCTION

Nowadays, System-On-Chip (SOC) designers go for outsourcing some of the design and fabrication facilities to other companies worldwide. They adopt the growing technology of IP-Reuse to reduce IC's time-to-market and economize its manufacturing cost. Reusing virtual components might involve the resources of untrusted third parties; they can be granted an access and control over some vital design units. Thus, adversaries might have the chance to create a backdoor for many IP security attacks that jeopardize the authenticity of genuine Hardware IP designs. Those insecurities include intentional tampering insertions namely Hardware Trojans (HTs) [9]. HTs are undesired malicious alterations embedded by an intruder in the original design to either change its functionality by adding, modifying, or deleting some of its components, leak its sensitive information like its thermal, area, power, or radiation profiles, or even deny its service by temporarily/permanently interrupting/blocking the operation of its blocks. Trojans vary in their physical properties, activation mechanisms, and obviously their effects.

Watermarking is a tagging approach where the IP designer inserts a special identity (signature) in the design; a tag that enables the designer to track the block and discover any illegal use or any malicious alteration in the IP content itself. Robust watermarking is proposed to protect copyrights of IP designer and prove his authorship. On the contrary, the proposed technique in this work is based on fragile watermarking approach, extremely sensitive to any design changes and hence detects any tampering activities i.e. Trojans attacks against the IP block [1]. This work tries to maintain a base of trust for the security of Hardware IP blocks against the intrusive attacks of Trojans; inserted at the RTL-level in the design phase of the chip. It focuses on Trojans inserted in high-level system Hierarchical Finite State Machines i.e. HFSMs [4], as they represent the control circuit design and Trojans there manipulate the operation of the whole system.

A Computer-Aided Design (CAD) tool for Fragile HFSM Watermarking is proposed as a novel presilicon HDL analysis approach for Trojan detection. In [11], a Fragile Watermarking tool is applied on multiple benchmarks for one-level FSMs. The proposed tool achieves low insertion overhead and hence a slight impact on design's timing constraints. This tool is extended in this work to insert the fragile watermark in HFSMs instead of traditional FSMs. The concept of hierarchy is utilized to apply the proposed watermarking design practically on large complex systems that conventional FSMs fail to model; the insertion overhead and complexity level of HFSM watermarking are minimized compared to those of ordinary FSM watermarking. The concurrency problem in HFSMs can also be tackled, the watermark can be inserted at different sub-levels of concurrent and communicating FSM designs. HFSMs are implemented at the highest abstraction level in the hierarchy of the IC design flow, RTL-level. Hence, the inserted watermark is inherited in all down levels and the entire design is secured. Besides, the watermark is well-hidden and hard to be traced and hence resistant towards removal or unauthorized detection attacks.

The paper is organized as follows: Section II discusses the proposed framework for Fragile HFSM Watermarking. An evaluation for the proposed HFSM Watermarking tool as Trojan detection method is presented in Section III. Section IV sets comparisons between different Trojan detection techniques including the proposed approach. In Section V, an attack model is built to test the reliability of the proposed Fragile Watermarking design against minor possible changes. Section VI concludes this work.

II. FRAGILE HFSM WATERMARKING FRAMEWORK

The proposed Fragile HFSM Watermarking framework is highlighted in Fig. 1. It is divided into three stages namely watermark generation, insertion, and extraction phases.

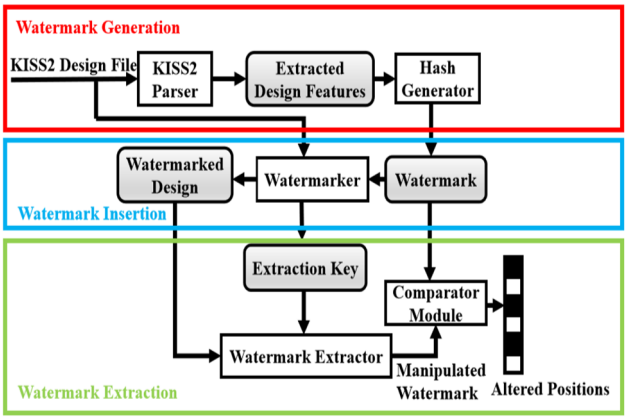

Fig. 1. Proposed HFSM Fragile Watermarking Framework.

A. Watermark Generation

Modelling of the system's RTL design in the format of SIS KISS2 files [2], is the first step in the Watermark Generation phase. KISS2 format visualizes the dynamic

reactions of state machines in response to specific input values leading to corresponding output values in an easy straightforward way. The structural format of KISS2 files doesn't change regardless the FSMs are of Moore or Mealy type and it can be transferred between different tools easily. Figure 2 shows how the state transition graph (STG) of FSMs is translated into KISS2 files. KISS2 format is effective with traditional FSMs yet it had to be extended using the approach in [10] to accommodate the HFSM watermarking design with or without concurrent modules. The tool proposed in this work can be applied on both cases. Figure 3 shows an example of an extended HFSM KISS2 formatted file without concurrent modules, while an example of an extended HFSM KISS2 formatted file with concurrent modules is presented in Fig. 4.

The scope of this paper is the detection of HTs inserted in the system HFSM design to modify its logical functionality; those Trojans are in form of minor intended alterations in the HFSM KISS2 file. The main goal is that those changes directly reflect on the sensitive fragile watermark to detect the implanted Trojan. Hence, the work doesn't set an arbitrary binary sequence for the watermark but, the original HFSM KISS2 file itself is used to generate it. The main concern is the variations that might appear in the original watermark due to malicious insertions by an intruder at the integration phase in which the RTL file is required. The proposed watermark generator consists of two main modules namely the KISS2 parser and the CRC16 (Cyclic Redundancy Check) [3] hash generator detailed as follows:

KISS2 Parser: It receives the HFSM KISS2 file and goes through its lines to sort different types of present states; the atomic states, the master hierarchical states, and their slave states. The parser builds a tree structure that pictures the available transitions at each state; each transition with its input/output function, current, and next states. At each current state (atomic/master/slave), it extracts the input/output functions of all its available transitions; it then puts them in form of one binary stream of bits.

CRC16 Hash Generator: Each of the binary streams output from the KISS2 parser is then fed into the ready-designed CRC16 hash generator; it produces a 16-bit long digest corresponding to each string. All these strings are concatenated in one long string in the same order in which they are generated to produce the whole watermark. The length of the produced watermark is proportional to the number of current states (atomic/master/slave) in the HFSM KISS2 file. CRC16 has an advantage that it generates a small number of watermark bits. A long watermark sequence is not needed; large watermark capacity only matters in robust watermarking as it should present a rigid proof of ownership in court. On the other hand, a change in only one bit of a fragile watermark is sufficient to detect intentional tampering i.e. Trojans. Moreover, the watermark length directly affects the implementation overhead of the watermark insertion cycle.

B. Watermark Insertion Algorithm

The proposed insertion algorithm is based on the FSM Coinciding Transition Watermarking technique [1]. The fragile watermark is mapped into the output patterns of different already existing transitions of all available states in the system HFSM design. It doesn't exploit any extra unused transitions with additional available input/output patterns i.e.

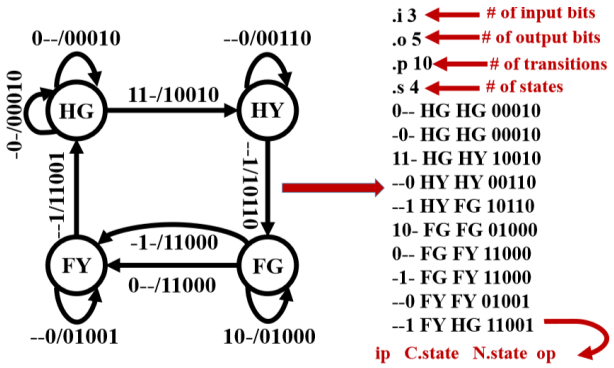

Fig. 2. The conversion of STG into KISS2 format.

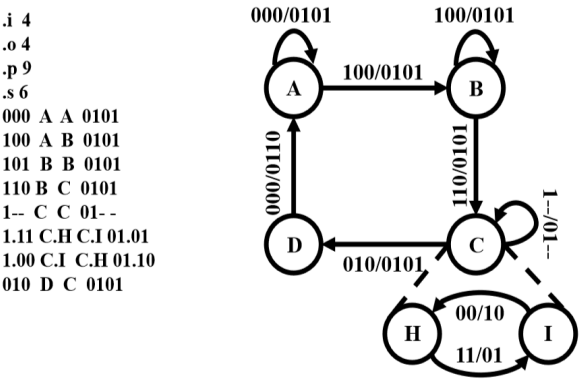

Fig. 3. Extended KISS2 format of HFSM without concurrent modules.

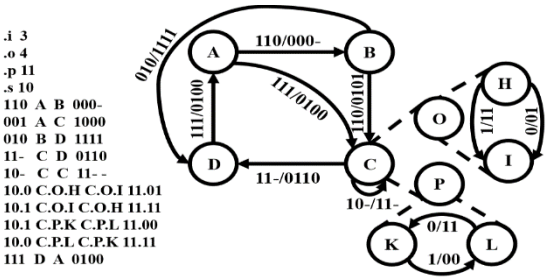

Fig. 4. Extended KISS2 format of HFSM with concurrent modules.

thus no design overhead. The coinciding approach reinforces the robustness level of the proposed watermark; the technique implants the watermark in the original behavioral model of the design. Hence, any attempt to distort the watermark or remove it will undermine the base of the whole HFSM design.

The watermarker sets an insertion window of two or three watermark bits at a time; this reduces the complexity level of the proposed watermarking design and optimizes its insertion time. At the end of this phase, a watermarking key is generated; It consists of triplets; each represents a watermarked transition. Each triplet has three elements that mark the transition in which the watermark was inserted. The first element is the input function of the watermarked transition. The second element is the watermark string that is mapped to the output function of the watermarked transition. The third element is the index position of the first mapped watermark bit within the transition's output. If the first watermark bit in the insertion window coincides with the first output bit, the third element will be 0 and if it coincides with the second one, it will be 1 and so on.

Referring to the flow chart in Figure 5, the insertion tool chooses a random current state to start the embedding loop;

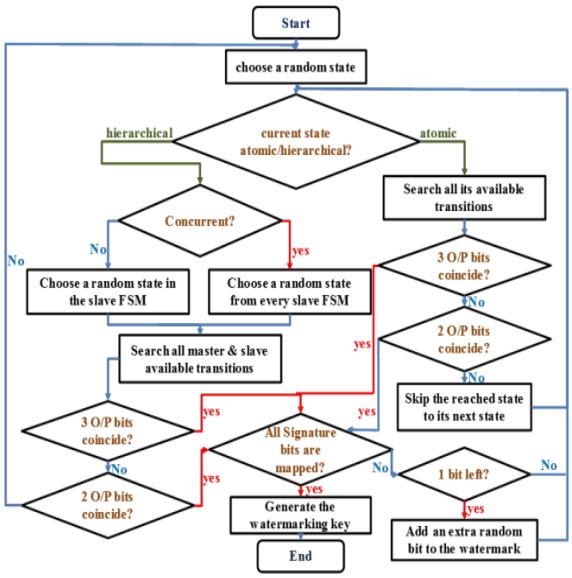

Fig. 5. A flow chart for the proposed Watermark Insertion tool

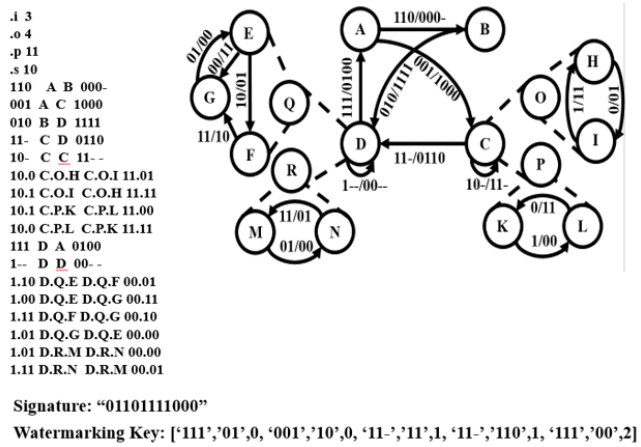

Fig. 6. Simulation results of the proposed Watermarking Insertion tool

if that state is atomic, the search engine looks for two or three watermark bits within the output functions of all available transitions at that state i.e. checks if they are equal. The transition that can map the highest number of watermark bits at a smaller index output bit, is chosen to be watermarked. If all transitions can do so, a random one is chosen. The system is transferred to the next state of the chosen transition where the tool searches for the upcoming watermark bits. If all transitions fail to map the watermark bits, another random state is selected to start over the insertion cycle. In case the state is hierarchical, similar steps are followed; the tool searches for the watermark bits within the output patterns in both master and slave FSMs. If the watermarked transition is at a master state, its self-transition is executed and the system remains at that state. At any reached state, if the watermarker fails to insert three or two watermark bits at a time starting from any output bit in the available transition/s, the insertion loop turns endless. As a solution, the tool flips the binary logic of the watermark bits. Then, the system transfers to the next state of the available transition or one random next state of one of the multiple available transitions at that state where the tool starts over the search loop trying to embed the new watermark bits. Each watermarked transition forms a part in the watermarking key until the embedding cycle is over, the whole key is generated.

The proposed watermarking insertion tool is applied on HFSM designs with and without concurrent modules, an example of an HFSM with concurrency model (which is more challenging) is chosen to insert 10 watermark bits only as shown in Figure 6. Simulation results are produced by running the code of the presented watermarker on Python 2.7.12 [8] on a PC using an Intel(R) Core™ i76500U processor with 8 GB of RAM.

Step 1: The tool randomly chose the atomic state A as the starting state at which the watermark embedding loop initiates. The search engine tries to coincide the first three watermark bits "011" within any of its output streams. Neither of the two available transitions can map the three bits at a time and hence the tool attempts to coincide the first two watermark bits "01" instead. But, both transitions also fail to coincide the two watermark bits "01". Therefore, the parser forms a list of all next states in all transitions of state A. One of these states is randomly chosen such that state A is skipped to search for "011" at the transitions of the new state.

Step 2: Atomic state B is randomly chosen; the tool starts to search for the watermark bits "011" or "01" within the output function of the only available transition of state B, but no match was to be found. Since state B has only one transition, the watermarker automatically transfers to its next state D to start searching for the first three watermark bits "011".

Step 3: Reaching state D, it is a hierarchical state that has two slave FSMs Q & R. An entry slave state is randomly chosen from each. The search engine starts to look for the three watermark bits "011" in all available transitions of both entry slave states in the slave FSMs and in those of the hierarchical state in the master FSM. Slave state G is randomly chosen from slave FSM Q and slave state M is randomly chosen from slave FSM R.

Step 4: The three watermark bits "011" cannot be mapped into any of the available transitions of either the master or the entry slave states. But instead, only two watermark bits "01" can be coincided with the output function of the transition between the hierarchical state D and the atomic state A in the master FSM. That watermarked transition forms the first triplet in the watermarking key ['111','01','0'].

Step 5: At the atomic state A, the tool searches the two available transitions to map the next three "101" or two "10" watermark bits. The first two bits in the output function of the transition between states A & C coincides with the two watermark bits "01". Hence, the second triplet in the watermarking key is ['001', '10', 0].

Step 6: State C is a hierarchical state that has two slave FSMs O & P. Slave state H is randomly chosen from slave FSM O and slave state L is randomly chosen from slave FSM P. Both transitions between the hierarchical states C&D in the master FSM and between slave states L&K in the slave FSM P can coincide the same number of watermark bits i.e. the two watermark bits "11". But they are coincided with the output pattern of the master transition starting from the second output bit while they are coincided with the output pattern of the slave transition starting from the third bit. So, the transition between hierarchical states C&D is

chosen to be watermarked and form the third triplet in the watermarking key ['11-', '11', 1].

Step 7: The reached state is still a hierarchical one "D" so the tool randomly chooses G&N to be the entry slave states. The search engine starts to search for "110" or "11". These watermark bits cannot be found in either the transitions of state D in the master FSM or those of slave states G&N in slave FSMs Q&R. The tool randomly chooses one of the states to proceed with its insertion loop. State C was chosen and slave states I&K are taken as entry states. The three watermark bits "110" were successfully mapped in the transition between hierarchical states C&D in the master FSM such that the fourth triplet in the watermarking key is ['11-', '110', 1].

Step 8: Reaching at state D, there is only one watermark bit left "0" therefore the tool adds an extra random bit ('0' in the example) to the generated watermark and the search engine starts to look for the last two watermark bits "00" to be mapped to the output of any of its available transitions in the master FSM or of those of the randomly chosen slave states E&N in the slave FSMs Q&R. The watermark bits were coincided with the output function of the transition between D&A in the master FSM such that the last triplet in the watermarking key is ['111', '00', 2]. The whole generated watermarking key is shown in Fig. 6.

Step 9: In case that the watermarked transition is in a slave FSM, the hierarchical state undergoes a self-transition such that the system is kept at the same state i.e. the system does not transfer to another state and the self-transition is watermarked to generate the watermarking key where the input/output function that triggers the watermarked transition consists of two parts; the first is the input/output function of the self-transition to stay at hierarchical state while the other is the input/output function that triggers the watermarked transition inside the slave FSM.

III. EVALUATION OF THE PROPOSED HFSM WATERMARKING TECHNIQUE

In this section, a sample of evaluation metrics is displayed; they are used to validate the reliability of Trojan detection techniques. They are classified into primary metrics and secondary ones according to their significance and applicability [12].

A. Primary metrics:

1. The detectable Trojan classes using the method

The proposed approach mainly target small-sized functional Trojans inserted in the design's state machine algorithmic model to corrupt its normal operational routine. Such Trojans are internally activated based on a logic condition that executes the whole design's state machine in response to certain input/output function.

2. Complexity/Difficulty of Trojan Detection

Referring to [7], detecting Trojans at the RTL level in the functional design phase is an easy process that achieves high accuracy, low cost, and time. Yet, the proposed HFSM Watermarking detection technique is relatively complicated; the Trojan is sensed by monitoring its altering effect on the extracted watermark, which is not an easy task. A technique that resembles transition tours may be applied in response to a specific input vector stated in the watermarking key to execute the HFSM and produce its output pattern; the index element is used to mark the positions of the watermark bits. However, this process runs only once in the HFSM simulation phase for verification and hence it is a tolerated trade-off.

3. Vulnerable points in the detection method

In the watermark generation phase, the CRC hash generator uses the input/output functions of the HFSM design to produce the signature digest. A vulnerable point occurs if there are Trojaned input/output functions that when hashed, give the same digest as the original one. In solution, a strong hash function "CRC16" is used; it gives a very small collision probability (the probability that the hashing function produces the same hash (digest) for two different messages) related to the number of hash bits n as shown in the equation in (1):

$$P_c = \frac{1}{2^n} = \frac{1}{2^{16}} = \frac{1}{65536} \quad (1)$$

4. Activation of the detection method

The proposed watermarking is dynamic i.e. in order to extract the watermark to be checked whether it is authentic or Trojaned, the HFSM design should be executed by applying specific input patterns that generate the output sequences in which the watermark bits are inserted.

B. Secondary Metrics

1. Effect of detection method on performance, design time and power consumption

The proposed watermarking tool uses the Coinciding transition technique [1] to insert the watermark in the already existing transitions of the HFSM design without any additional states, transitions, or unused input/output functions. Hence, the functionality, power consumption, and area constraints of the watermarked design are similar to those of the original one i.e. the watermarking strategy imposes no design overhead. In addition, it embeds two or three watermark bits at a time in order to reduce the design complexity and shorten the insertion time. Thus, the technique's implementation overhead in both embedding and extraction processes, is enhanced

2. The need for support (design-for-security) circuitry

This paper proposes a watermarking CAD tool that is integrated at the RTL level in the system design phase. Hence, it doesn't need any extra hardware circuitry.

3. Detection methods robustness against process variations

Process variations have no impact on the proposed Fragile Watermarking tool for Trojan detection.

IV. HT DETECTION TECHNIQUES COMPARISON

Among the Hardware Trojan presilicon detection techniques [6], the proposed Fragile HFSM Watermarking design belongs to the HDL analysis. It detects the Trojans inserted in the behavioral HDL code that describes the state machine of the original hardware design. Trojans can be in form of extra added transitions or flipped input/output functions.

To highlight the cons and pros of the proposed approach versus other Trojan detection techniques, a comparison is performed between them based on some evaluating

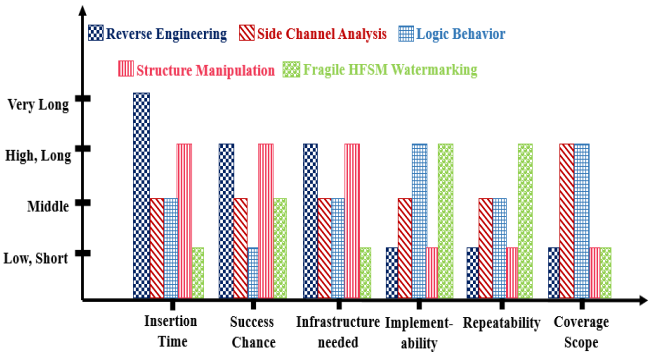

Fig. 7. Performance specifications of different Hardware Trojan detection techniques

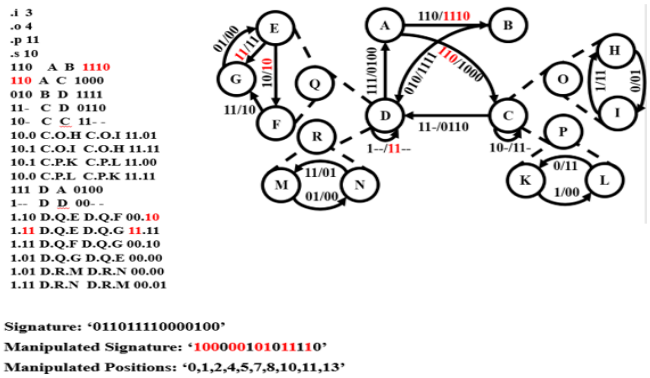

Fig. 8. Simulation results of the proposed attack scenario

performance aspects proposed by Sharifi et al. [5] as shown in Figure 7.

Insertion Time: The proposed watermarking tool imposes an insignificant insertion overhead. The insertion time is directly related to the complexity of the insertion loop, that is proportional to the total number of states and transitions used in the HFSM design of the system. The example shown in Fig. 6, the proposed tool managed to embed the 208-bit long watermark in 2.15 seconds.

Success Chance: The proposed watermarking design has a fair success chance to detect minor alterations in the HFSM design. However, it might be challenged by an attack in which the intruder bypasses the original state machine and uses a parallel control path to manipulate the design without being detected.

Infrastructure Needed: The tool does not need any extra infrastructure to implement the watermarking design, because it only depends on an extra software tool to insert the watermark bits in the behavioral RTL code of the design.

Implementability: The watermarking tool is implemented once without adding extra design circuits to the original system and hence it has a low implementation overhead.

Repeatability: The watermarking algorithm is easily repeated without exhaustive implementation routine, cost, or time.

Coverage Scope: The approach covers the whole design state machines efficiently, yet it is not designed to work on the data path.

V. ATTACK MODEL

In order to check the robustness of the proposed approach against possible Trojans, an attack scenario is simulated based on a testing framework that consists of two main modules namely the test and the comparator modules.

The test module: It acts in the role of the malicious intruder. It inserts some intended Trojans in the original HFSM KISS2 file; they are in form of changed input or output bits at random transitions of atomic or slave states. A Trojaned file is generated. Both the original and Trojaned files are fed into the CRC16 hash generator to produce the authentic and faulty watermarks shown in Fig. 8.

The comparator module: It receives both watermarks and checks if they are equal bit by bit. It outputs the exact positions of the altered watermark bits if exist as shown in Fig. 8.

It is noted that the least variations in random input/output functions at any state whether in master or slave FSM, are directly reflected on the proposed Fragile watermark. That makes it a reliable tool to detect minor Trojans effectively and hence maintain the authenticity of the jeopardized RTL design.

VI. CONCLUSION

A Fragile coinciding HFSM watermarking tool was proposed to detect the malicious Hardware Trojans inserted in an IP block design to threaten its authenticity. The fragile watermark is extremely sensitive to any minor alteration in the original design and hence is reliable to detect Trojans prior to chip fabrication. In addition, it has a low complexity level, robust, and imposes an insignificant insertion overhead. The proposed tool allows us to detect different HTs in HFSMs described in KISS2 format, hence complex control systems can be modeled and watermarked using this technique.

REFERENCES

[1] A.T. Abdel-Hamid, S. Tahar, and E.M. Aboulhamid, "A Public-Key Watermarking Technique for IP Designs", *Conference on Design, Automation and Test in Europe (DATE)*, Germany, 2005, pp. 330 - 335.

[2] A.T.Abdel-Hamid, M.Zaki, and S.Tahar, "A tool converting finite state machine to VHDL", Canadian Conference on Electrical and Computer Engineering, IEEE, 2004, pp. 1907-1910.

[3] Crcmod.Predefined – CRC Calculation Using Predefined Algorithms — Crcmod V1.7 Documentation". Crcmod.sourceforge.net. N.p., 2017. Web. 2 Mar. 2017.

[4] D.Harel, "Statecharts: A visual formalism for complex systems", Science of computer programming, 1987, pp. 231-274.

[5] E.Sharifi, K.Mohammadiasl, M.Havasi, and A.Yazdani, "Performance analysis of Hardware Trojan detection methods", International Journal of Open Information Technologies, 2015.

[6] K Xiao, D Forte, Y Jin, R Karri, and S Bhunia, "Hardware Trojans: Lessons learned after one decade of research", ACM Transactions on Design Automation of Electronic Systems (TODAES), 2016, pp. 1-23.

[7] N.Jacob, D.Merli, J.Heyszl, and G.Sigl, "Hardware Trojans: current challenges and approaches", IET Computers & Digital Techniques, 2014, pp. 264-273.

[8] "Python Release Python 2.7.12". Python.org. N.p., 2017. Web 25 June 2016.

[9] R.Karri, J.Rajendran, K.Rosenfeld, and M.Tehranipoor, "Trustworthy hardware: Identifying and classifying hardware trojans", 2010, pp. 39-46.

[10] S.M.Ahmed, "Hierarchical finite state machine watermarking", 2012.

[11] S.M.H.Shukry., A.T.Abdel-Hamid, and M. Dessouky, "Affirming Hardware Design Authenticity Using Fragile IP Watermarking", International Conference on Computer and Applications (ICCA), IEEE, 2018, pp. 1-347.

[12] X.Wang, M.Tehranipoor, and J.Plusquellic, "Detecting malicious inclusions in secure hardware: Challenges and solutions", International Workshop on Hardware-Oriented Security and Trust, IEEE, 2008, pp. 15-19.