J Electron Test (2011) 27:767–785
DOI 10.1007/s10836-011-5255-2

Security Against Hardware Trojan Attacks Using Key-Based Design Obfuscation

Rajat Subhra Chakraborty · Swarup Bhunia

Received: 17 January 2011 / Accepted: 15 September 2011 / Published online: 15 October 2011

Abstract Malicious modification of hardware in untrusted fabrication facilities, referred to as *hardware Trojan*, has emerged as a major security concern. Comprehensive detection of these Trojans during post-manufacturing test has been shown to be extremely difficult. Hence, it is important to develop design techniques that provide effective countermeasures against hardware Trojans by either preventing Trojan attacks or facilitating detection during test. *Obfuscation* is a technique that is conventionally employed to prevent piracy of software and hardware intellectual property (IP). In this work, we propose a novel application of key-based circuit structure and functionality obfuscation to achieve protection against hardware Trojans triggered by rare internal circuit conditions. The proposed obfuscation scheme is based on judicious modification of the state transition function, which creates two distinct functional modes: *normal* and *obfuscated*. A circuit transitions from the obfuscated to the normal mode only upon application of a specific input sequence, which defines the key. We show that it provides security against Trojan attacks in two ways: (1) it makes some inserted Trojans benign, i.e. they become effective only in the obfuscated mode; and (2) it prevents an adversary from exploiting the true rare events in a circuit to insert hard-to-detect Trojans. The proposed design methodology can thus achieve simultaneous protection from hardware Trojans and hardware IP piracy. Besides protecting ICs against Trojan attacks in foundry, we show that it can also protect against malicious modifications by untrusted computer-aided design (CAD) tools in both SoC and FPGA design flows. Simulation results for a set of benchmark circuits show that the scheme is capable of achieving high levels of security against Trojan attacks at modest area, power and delay overhead.

Keywords Design obfuscation · Hardware Trojan · IP protection · Logic testing · Trojan detection

1 Introduction

The issue of ensuring "Trust" has become prominent due to widespread outsourcing of the IC manufacturing processes to offshore locations in order to reduce cost [2, 3, 15, 30, 33]. A design can be tampered in an untrusted fabrication facility by the insertion of malicious circuitry that triggers a malfunction or leaks secret information. Such malicious circuitry, referred to as a *hardware Trojan*, can activate under rare condition and affect normal circuit operation, potentially with catastrophic consequences in critical application areas

Responsible Editor: S. T. Chakradhar

A preliminary version of this work has been published in the *International Conference on Computer Aided Design* (*ICCAD*), 2009.

R. S. Chakraborty (✉)
Department of Computer Science and Engineering,
Indian Institute of Technology,
Kharagpur, West Bengal 721302, India
e-mail: rschakraborty@cse.iitkgp.ernet.in

S. Bhunia
Department of Electrical Engineering and
Computer Science, Case Western Reserve University,
Cleveland, OH 44106, USA
e-mail: skb21@case.edu

and public infrastructure. Such malicious circuitry can also be inserted by CAD automation tools obtained from untrusted third party vendors [31]. Due to the inherently stealthy nature of hardware Trojans, inordinately large number of possible Trojans an adversary can exploit, and greatly diverse functional mode of inserted Trojans, detection of hardware Trojans by post-manufacturing test has emerged as an extremely challenging problem [5, 7, 13, 18, 26, 28, 29].

Another major security threat in modern IP-based system-on-chip (SoC) and embedded system design is *IP piracy*, whereby hardware IP vendors are deprived of their royalty through theft, reverse-engineering, cloning and illegal use of hardware IPs. An effective technique to prevent such piracy is *obfuscation* of the design [10]. *Obfuscation* is a technique that transforms a software or a hardware design description into one that is functionally equivalent to the original but is significantly more difficult to reverse engineer. Obfuscation has been proposed as a possible solution to prevent piracy of hardware IP cores, both at the register transfer level (RTL) [9, 34] and gate level [10].

In this paper, we demonstrate a novel application of design obfuscation to provide security against hardware Trojans, We derive that obfuscation can provide the dual benefits of protection against piracy and Trojan attacks. We target those Trojans whose activation depend on rare logic values at internal nodes in the circuit. Previous works [3, 13, 37] have shown that an adversary can design combinational or sequential Trojans represented by this model, which can be extremely hard to detect using conventional test process. We distinguish our proposed obfuscation-based technique from *obscurity*-based techniques. Conventional security through obscurity approaches depend on transforming the representation of functional behavior of a design, so that a function is replaced by an alternative one with the same behavior but significantly more difficult to interpret. These approaches do not provide mathematically provable robustness and has long been considered to be in-effective in practice. The weakness of the these approaches has recently been shown theoretically [8]. In particular, this paper makes the following contributions:

- It proposes a novel *design for security* technique that (a) hardens a design against Trojan insertion, and (b) increases the sensitivity of post–silicon validation to detect hardware Trojans.
- It analyzes the effectiveness of a key-based gate-level design obfuscation scheme in achieving security against hardware Trojans. It provides mathematical analysis to derive the impact of different design and obfuscation parameters on the degree of security achieved against Trojans. An advantage of the proposed technique is that the security features propagate through the IC design flow to lower levels of design abstraction (such as GDS-II). The methodology ensures that no structural signature is introduced while obfuscating the functionality of the circuit by modifying the *state transition graph* (STG), while the *normal mode* circuit functionality is kept unaltered.
- It introduces two important techniques as part of the obfuscation methodology that are highly effective in providing security against hardware Trojans: (1) addition of extra state elements to "blow up" the state space exponentially and then use large number of these states in the *obfuscated mode* of operation, and (2) use of *functionally unreachable* states of the original state machine in the *obfuscated mode*.
- It develops an automated design flow to incorporate the above modifications while incurring low design overhead. It also proposes appropriate steps for evaluating the effectiveness of the proposed approach for complex gate-level netlists.
- Malicious CAD tools and automation scripts in automated design flows are potential sources of Trojan insertion [15, 31]. This threat is relevant at all stages of the design flow. We show that the proposed obfuscation based design methodology can provide protection against such malicious CAD tools in both the SoC and FPGA design flows. We also discuss how the proposed methodology can be extended to protect against different types of Trojans such as *information leakage* Trojans [23] which try to leak secret information from inside a chip.

The rest of the paper is organized as follows. In Section 2, we provide background on hardware Trojans and obfuscation for hardware security. In Section 3, we provide mathematical analysis and describe how an obfuscation-based IP protection technique can be beneficial in countering the security threat posed by hardware Trojans. In Section 4, we describe an automated design flow that implements the proposed methodology. We present simulation results to validate the concept in Section 5. In Section 6 we discuss a technique to decrease the design overhead, applicability of the proposed design methodology to third-party IPs and system-on-chips (SoCs), and the effectiveness of this scheme in proving protection against malicious CAD tools and *information leaking* Trojans. We conclude in Section 7.

2 Background

2.1 Hardware Trojans: Models and Examples

A hardware Trojan instance can typically be associated with two sets of internal nodes: the nodes which trigger malfunction by activating the Trojan (called the *trigger nodes*), and the nodes which are affected by the Trojan (called the *payload nodes*) [37]. Figure 1a [13] shows a *combinational Trojan* where the payload node S has been modified to the node $S^{\star}$, and malfunction is triggered whenever the condition $a = 0, b = 1, c = 1$ is satisfied at the corresponding *trigger nodes*. The *sequential Trojan* shown in Fig. 1b, on the other hand, is a 3-bit counter which causes a malfunction at the payload node S on reaching a particular count, which is incremented each time the condition $a = 1, b = 0$ is satisfied.

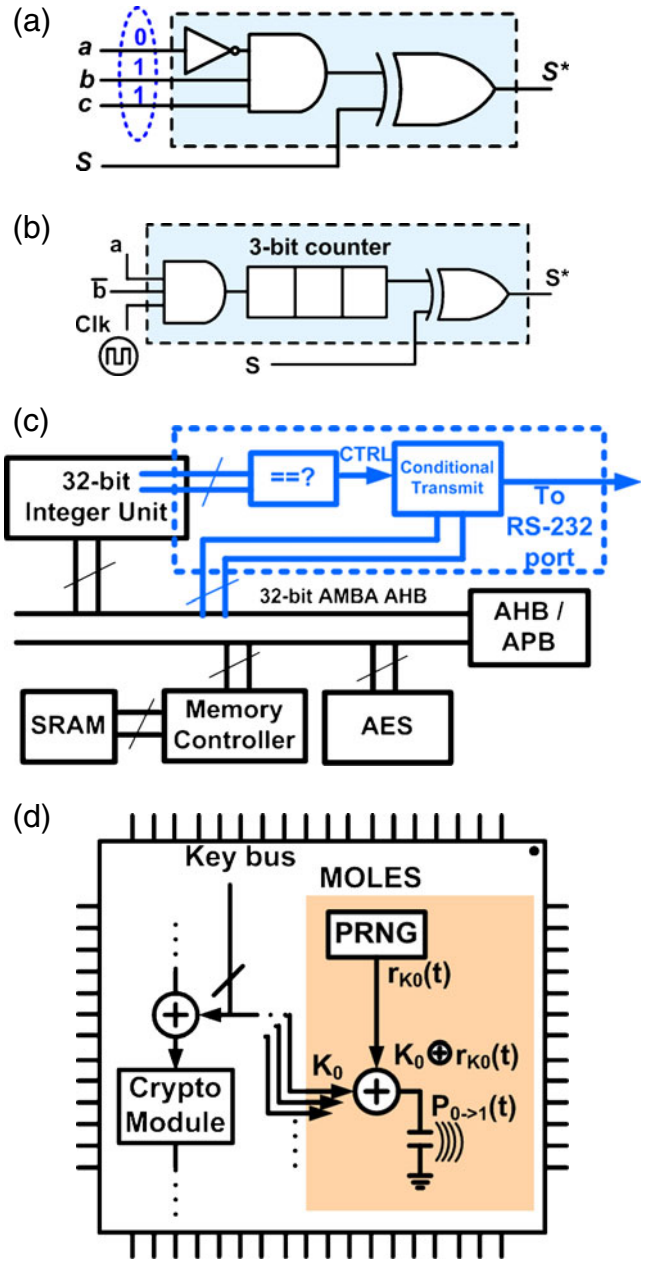

Fig. 1 Examples of combinational (**a**) and sequential (**b**) hardware Trojans that cause malfunction conditionally; examples of Trojans leaking information through logic values (**c**) and through side-channel parameter (**d**) [23]

A Trojan can also exhibit its malicious effect by leaking information through covert communication channels [19]. An example Trojan that leaks logic information conditionally is shown in Fig. 1c. The system is designed for a cryptographic application with an integer unit and a cryto-core, which performs Advanced Encryption Standard (AES) based cipher operations. The Trojan consists of a comparator and a conditional data transmitter. The comparator compares the values at a few internal nodes of the integer unit with a constant value, and the transmitter sends out the contents of the data bus (which can be the key for the AES encryption) through the RS-232 port if the comparison succeeds. Another example is shown in Fig. 1d [23], where a bank of inserted capacitors is charged depending on the result of XOR-ing the output of a pseudo-random number generator (PRNG) with the contents of a data bus connected to an AES module. Provided the adversary has a clock that is synchronized with the PRNG clock and knowledge of the PRNG structure, it is possible to extract the AES key from the *side-channel information* consisting of supply current traces of the IC.

An adversary would not be benefited by a pseudo-random Trojan insertion procedure without a thorough understanding of the original circuit behavior. This is because the effect of such a Trojan is similar to random circuit defects, and are likely to be detectable by post-manufacturing testing which would eliminate the infected IC. In such a scenario, the adversary can at most affect the yield of the IC manufacturer, but cannot really cause any substantial harm. Instead, an intelligent adversary is much more likely to first perform a comprehensive analysis of the circuit behavior, and then choose rare conditions at internal circuit nodes as Trojan triggering condition [15, 37] which are unlikely to arise during test but can occur during long hours of field operation. Moreover, the inserted Trojan is likely to be designed such that its malicious effect at the payload is difficult to observe at the output ports. An intelligently designed Trojan with rare activation condition and/or rare observability of payloads would evade post-manufacturing test with high probability. The number of possible Trojan instances in a given circuit has a combinatorial dependence on the number of

circuit nodes. For example, with an assumption of maximum four trigger nodes, a small ISCAS-85 circuit *c880* with 451 gates can have $\sim 10^9$ possible trigger conditions and $\sim 10^{11}$ single payload Trojan instances, respectively. Hence, it is not computationally feasible to enumerate all possible Trojan instances in a given circuit and generate deterministic test patterns for them.

Many existing approaches of Trojan detection rely on the measurement of side-channel parameters such as delay and power signature [3, 7, 18, 29]. A major challenge lies in the fact that these techniques can be extremely susceptible to measurement and process-variation induced noise. Moreover, they suffer from reduced detection sensitivity in detecting ultra-small Trojans consisting of few logic gates [3], and hence several techniques have been recently proposed that aim to increase the sensitivity of the measurements [1, 5, 16, 22, 26, 28]. Design techniques have also been proposed that help to detect inserted Trojans [12, 20].

As mentioned in Section 1, in our analysis and simulation results, we consider a major class of the Trojans that can be functionally represented by the examples shown in Fig. 1a and b. However, we show in Section 6 that the proposed design methodology is also effective in protecting ICs against *information leakage* Trojans which are triggered by a specific set of digital values at selected circuit nodes (similar to the types shown in Fig. 1c and d), regardless of whether the information leaked by them is logic information or side-channel information. For Trojans of other functional models (e.g. free-running state machines of arbitrarily complex behavior, not necessarily triggered by rare internal circuit conditions), side-channel techniques that offer greater detection sensitivity [1, 5, 16, 26, 28] hold greater promise.

2.2 Obfuscation for Hardware IP Protection

The approaches for obfuscation based hardware IP protection can be divided into two main classes—(a) those that affect the comprehensibility of the description of IP core (usually in a hardware description language such as Verilog or VHDL), but keep the functionality unchanged [9, 34], and (b) those that obfuscate the functionality of the IP core [10]. In [9, 34] software tools have been described that can affect the human comprehensibility of a RTL by variable renaming, removal of comments, loop unrolling, etc. However, obfuscation features do not propagate down to lower levels of design abstraction (e.g. gate-level or GDS-II), and thus cannot be used to prevent Trojan insertion in foundry.

Functional obfuscation based approaches such as those proposed in [10], on the other hand, obfuscate the way the circuit operates and normal operation is enabled only after the application of a correct *initialization key sequence*. Such approaches work essentially by changing the state transition function of a design to define two distinct modes of operation: the *obfuscated mode* and the *normal mode*. The main observation is that such functional obfuscation prevents the adversary from fully understanding the circuit operation. Consequently, the adversary cannot decide with certainty what is an actual *rare logic condition* in the circuit that would serve as the triggering condition of the inserted Trojan. A similar approach based on modifying the state transition graph of a circuit for IP protection and security has been described in [4]. Key-based hardware protection techniques have also been investigated to prevent illegal manufacturing and circulation of ICs [6, 32]. However, none of the above mentioned approaches target or provide protection against hardware Trojans.

3 Methodology

In this section, we describe the design methodology based on obfuscation to protect designs against hardware Trojans. A conceptually similar protection scheme has been earlier suggested in [36] for software, whereby by decreasing the comprehensibility of a program, malicious software modification is prevented.

The obfuscation in our scheme is achieved by two important modifications of the *state transition graph* (STG) of the circuit:

- The size of the reachable state-space is "blown up" by a large (exponential) factor using extra state elements.
- Certain states, which were *unreachable* in the original design are used and made reachable only in the *obfuscated mode* of operation.

These two modifications make it difficult for an adversary to design a functionally potent and well-hidden Trojan, as shown through the analysis presented in Sections 3.1–3.3. As would be evident from the following, this modification of the STG is different in methodology and goals from the ones proposed in [4, 6, 10, 32], in which no state-space explosion is attempted.

Figure 2a shows the proposed obfuscation scheme based on the change in the STG of the circuit. On power-up, the circuit is initialized to a state (S_0^O) in the *obfuscated mode*. On the application of an input

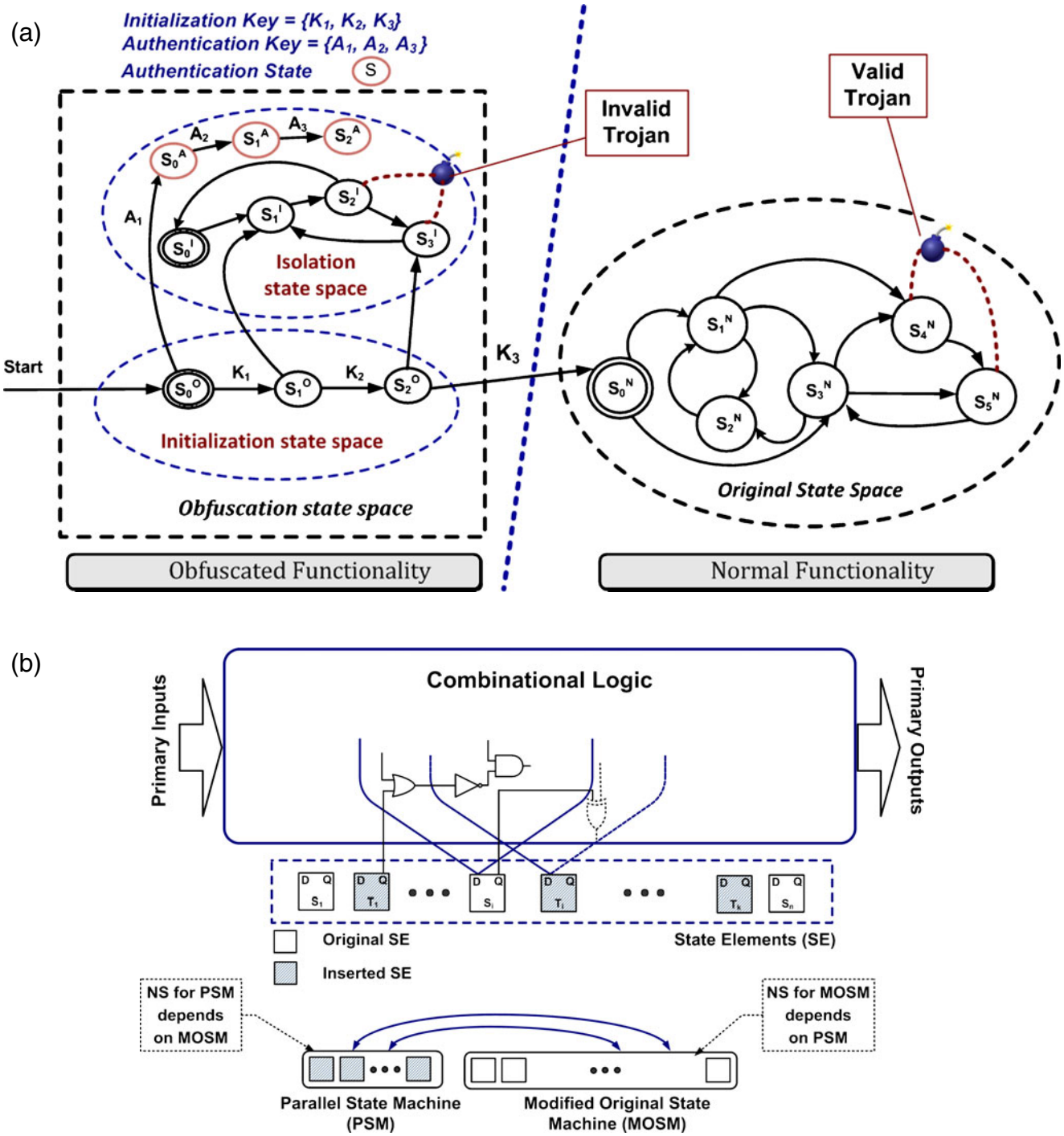

Fig. 2 The obfuscation scheme for protection against hardware Trojans: modified state transition graph (**a**) and modified circuit structure (**b**)

sequence $K_1 \rightarrow K_2 \rightarrow K_3$ in order, i.e. the *initialization key sequence*, the circuit reaches the state S_0^N, which is the *reset state* in the *original state space*, allowing *normal mode* of operation. The states S_0^O, S_1^O and S_2^O constitute the *initialization state space*. The application of even a single incorrect input vector during the initialization process takes the circuit to states in the *isolation state space*, a set of states from which it is not possible to come back to the *initialization state space* or enter the *original state space*. The *initialization state space* and the *isolation state space* together constitute the *obfuscation state space*. All state encodings in the *obfuscation state*

space are done using *unreachable* state bit combinations for selected state elements of the circuit. This ensures that the circuit cannot perform its normal functionality until the correct initialization key sequence has been applied. The initialization latency (typically <10 clock cycles) can be easily hidden from the end-user by utilizing the inherent latency of most ASICs during a "boot-up" or similar procedure on power-ON [10]. We consider the post-manufacturing testing phase to be "trusted" such that there is no possibility of the secret initializaton key to be leaked to the adversary in the fab. This is a commonly accepted convention which was first explicitly stated in [15]. To protect against the possibility of an user releasing the initialization key sequence of the design in the public domain, *user-specific initialization key sequence* or in the extreme case *instance-specific* initialization key sequence might be employed.

To "blow up" the size of the *obfuscation state space*, a number of extra state elements are added depending on the allowable hardware overhead. The size of the *obfuscation state space* has an exponential dependence on the number of extra state elements. An inserted *parallel finite state machine* (PSM) defines the state transitions of the extra state elements. However, to hide possible structural signature formed by the inserted PSM, the circuit description of the PSM is *folded* into the *modified state machine in the original circuit* (MOSM) (as shown in Fig. 2b) to generate an integrated state machine. A logic re-synthesis step is performed, including logic optimization under design constraints in terms of delay, area or power. In effect, the circuit structures such as the input logic cones of the original state elements change significantly compared to the unobfuscated circuit, making reverse-engineering of the obfuscated design practically infeasible for a adversary. This effect is illustrated in Section 3.4 through an example benchmark circuit.

To increase the level of structural difference between the obfuscated and the original circuits, the designer can choose to insert *modification cells* as proposed in [10] at selected internal nodes. Furthermore, the level of obfuscation can be increased by using more states in the obfuscated state space. This can be achieved by: (1) adding more state elements to the design and/or (2) using more unreachable states from the original design. However, this can increase the design overhead substantially. In Section 6 we describe a technique to reduce the design overhead in such cases.

Selected states in the *isolation state space* can also serve the purpose of authenticating the ownership of the design, as described in [10]. Authentication for sequential circuits is usually performed by embedding a *digital watermark* in the STG of the design [27, 35], and our idea of hiding such information in the unused states of the circuit is similar to [40]. A *digital watermark* is a unique characteristic of the design which is usually not part of the original specification and is known only to the designer. Figure 2 shows such a scheme where the states S_0^A, S_1^A and S_2^A in the *isolation state space* and the corresponding output values of the circuit are used for the purposes of authenticating the design. The design goes through the state transition sequence $S_0^O \rightarrow S_0^A \rightarrow S_1^A \rightarrow S_2^A$ on the application of the sequence $A_1 \rightarrow A_2 \rightarrow A_3$. Because these states are unreachable in the normal mode of operation, they and the corresponding circuit output values constitute a property that was not part of the original design. As shown in [10], the probability of determining such an embedded watermark and masking it is extremely small, thus establishing it as a robust watermarking scheme.

3.1 Effect of Obfuscation on Trojan Insertion

As mentioned before, to design a functionally catastrophic but hard-to-detect Trojan, the adversary would try to select a "rare" event at selected internal "trigger nodes" to activate the Trojan. To select a sufficiently rare trigger condition for the Trojan to be inserted, the adversary would try to estimate the signal probability [25] at the circuit nodes by simulations. To do so with a certain degree of confidence, a minimum number of simulations with random starting states and random input vectors must be performed [38]. However, the adversary has no way to know whether the starting state of the simulations is in the *normal state space* or the *obfuscation state space*. If the initial state of the simulations lie in the *obfuscation state space*, there is a high probability that the simulations would remain confined in the *obfuscation state space*. This is because the random test generation algorithm of the adversary most likely would be unable to apply the correct input vector at the correct state to cause the state transition to the *normal state space*. Essentially, the STG of the obfuscated circuit has two *near-closed* (NC) set of states [14], which would make accurate estimation of the signal probabilities through a series of random simulations extremely challenging. An algorithm was proposed in [14] to detect the *NC sets* of a sequential circuit; however, the algorithm requires: (a) knowledge of the state transition matrix of the entire sequential circuit, which is not available to the adversary, and (b) a list of all the *reachable states* of the circuit, which is extremely computationally challenging to enumerate for a reasonably large sequential circuit. Hence, we can assume that the adversary would be

compelled to resort to a random simulation based method to estimate the signal probabilities at internal circuit nodes.

3.2 Effect of Obfuscation on Trojan Potency

To decrease the *potency* of the inserted Trojan, the designer of the obfuscated circuit must ensure that if the adversary starts simulating the circuit in the *obfuscation state space*, the probability of the circuit being driven to the *normal state space* is minimal. Consider a sequential circuit originally with N state elements and M *used states*, to which n state-elements are added to modify the STG to the form shown in Fig. 2a. Let the number of states in the *obfuscation state space* be $S_i = f_1 \cdot 2^n \cdot (2^N - M) = f_1 \cdot 2^n \cdot B$, where $B = (2^N - M)$, and $f_1 < 1$ represents a *utilization factor* reflecting the overhead constraint.

Let I denote the set of states in the *obfuscation state space*, U denote the set of states in the *normal state space*, and T denote the set of states actually attained during the simulations by the adversary. Let, p be the number of primary inputs (other than the clock and reset) where the initialization *key* sequence is applied, and let the length of the initialization key sequence be k. Then, it takes k correct input vectors in sequence to reach the *normal state space* from state S_0^O, $k-1$ correct input vectors from state S_1^O, and so on. Then, the probability that the simulation started in the *initialization state space* and was able to reach the *normal state space* by the application of random input vectors:

$$P\left(T \subseteq \left\{I \bigcup U\right\}\right) = \frac{k}{S_i + M} \cdot \left(\frac{1}{2^p} + \frac{1}{2^{2p}} + \cdots \frac{1}{2^{pk}}\right) \tag{1}$$

$$\approx \frac{k \cdot 2^{-p}}{(f_1 \cdot 2^n \cdot B + M)(1 - 2^{-p})} \tag{2}$$

assuming $2^{-pk} \ll 1$. Similarly, the probability that the simulations started in the *initialization state space* or the *isolation state space* and remained confined there:

$$P\left(T \subseteq \left\{I \bigcup U'\right\}\right) = \left[1 - \frac{k \cdot 2^{-p}}{(f_1 \cdot 2^n \cdot B + M)(1 - 2^{-p})}\right] \cdot \frac{f_1 \cdot 2^n \cdot B}{f_1 \cdot 2^n \cdot B + M} \tag{3}$$

where U' denotes the complement set of U. Again, the probability that the simulations started in the *normal state space*, and remained confined there is:

$$P(T \subseteq U) = \frac{M}{f_1 \cdot 2^n \cdot B + M} \tag{4}$$

To maximize the probability of keeping the simulations confined in the *obfuscation state space*, the designer should ensure:

$$P\left(T \subseteq \left\{I \bigcup U'\right\}\right) \gg P(T \subseteq U) + P\left(T \subseteq \left\{I \bigcup U\right\}\right) \tag{5}$$

Approximating $M \gg \frac{k \cdot 2^{-p}}{1 - 2^{-p}}$, and simplifying, this leads to:

$$f_1 \cdot 2^n \cdot B \gg M \tag{6}$$

This equation essentially implies the size of the *obfuscation state space* should be much larger compared to the size of the *normal state space*, a result that is intuitively expected. From the analysis above, the two main observations are:

- The size of the *obfuscation state space* has an exponential dependence of the number of extra state elements added.
- In a circuit where the size of the used state space is small compared to the size of the unused state space, higher levels of obfuscation can be achieved at lower hardware overhead.

As an example, consider the ISCAS-89 benchmark circuit *s*1423 with 74 state elements (i.e. $N = 74$), and $> 2^{72}$ unused states (i.e., $2^N - M > 2^{72}$) [39]. Then, $M < 1.42 \times 10^{22}$, and considering ten extra state elements added (i.e. $n = 10$), $f_1 > 0.0029$ for Eq. 6 to hold. Thus, expanding the state space in the modified circuit by about 3% of the available unused state space is sufficient in this case.

3.3 Effect of Obfuscation on Trojan Detectability

Consider a Trojan designed and inserted by the adversary with q trigger nodes, with *estimated* rare signal probabilities $p_1, p_2, \ldots p_q$, obtained by simulating the obfuscated circuit. Then, assuming random input vectors, the adversary expects the Trojan to be activated once (on average) by the application of

$$N = \frac{1}{\prod_{i=1}^{q} p_i} \tag{7}$$

test vectors. However, let the *actual* rare logic value probabilities of these internal nodes be $p_i + \Delta p_i$, for the

i-th trigger node. Then, the Trojan would be *actually* activated once (on average) by:

$$N' = \frac{1}{\prod_{i=1}^{q}(p_i + \Delta p_i)} = \frac{N}{\prod_{i=1}^{q}\left(1 + \frac{\Delta p_i}{p_i}\right)} \tag{8}$$

test vectors. The difference between the estimated and the actual number of test vectors before the Trojan is activated is $\Delta N = N - N'$, which leads to a percentage normalized difference:

$$\frac{\Delta N}{N}(\%) = \left(1 - \frac{1}{\prod_{i=1}^{q}\left(1 + \frac{\Delta p_i}{p_i}\right)}\right) \times 100\% \tag{9}$$

To appreciate the effect that Δp and q has on this change on the average number of vectors that can activate the Trojan, assume $\frac{\Delta p_i}{p_i} = f \quad \forall i = 1, 2 \ldots q$; then Eq. 9 can be simplified to:

$$\frac{\Delta N}{N}(\%) = \left(1 - \frac{1}{(1+f)^q}\right) \times 100\% \tag{10}$$

Figure 3 shows this fractional change plotted vs. the number of trigger nodes (q) for different values of the fractional mis-estimation of the signal probability (f). From this plot and Eqs. 9 and 10, it is evident that:

- The probability of the Trojan getting detected by logic testing increases as the number of Trojan trigger nodes (q) increases. However, it is unlikely that the adversary will have more than ten trigger nodes, because otherwise as shown by our simulations, it becomes extremely difficult to trigger the Trojans at all.
- For values $2 \leq q \leq 10$, the number of random input patterns required to activate the trojan decreases sharply with q. The improvement is more pronounced at higher values of f. This observation validates the rationale behind an obfuscation-based design approach that resists the adversary from correctly estimating the signal probabilities at the internal circuit nodes.

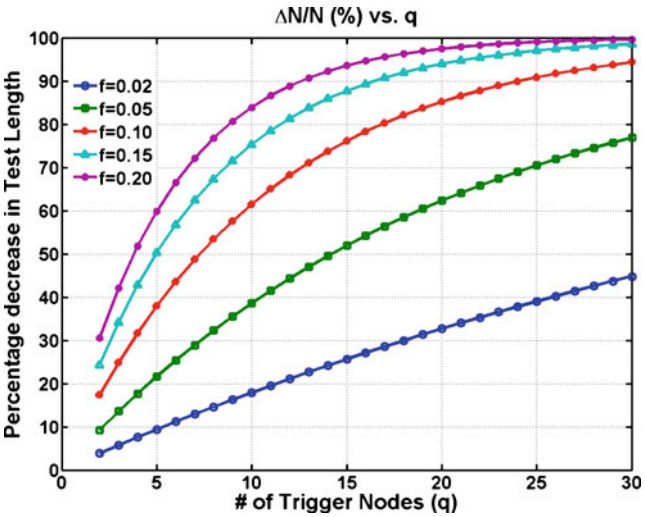

Fig. 3 Fractional change in average number of test vectors required to trigger a Trojan, for different values of average fractional mis-estimation of signal probability (f) and Trojan trigger nodes (q)

3.4 Effect of Obfuscation on Circuit Structure

The re-synthesis of the circuit after its integration with the RTL description corresponding to the *obfuscation state space* flattens the circuit into a single netlist, resulting in drastic changes to the input logic cones of the primary outputs and the state elements. This makes it infeasible to manually or automatically analyze and identify the modifications made to a practical circuit with reasonably large number of gates, even if the adversary is in possession of an unmodified version of the original circuit netlist. To appreciate this effect, consider the input logic cones (up to four levels) of a selected flip-flop in the gate level netlist of the *s*15850 ISCAS-89 benchmark, and its obfuscated version, shown in Fig. 4. Similarly significant structural difference was observed for all the benchmark circuits considered by us. If the adversary is not in possession of an unmodified *reference* gate-level design, this task is even more difficult, as the adversary would have no idea about the netlist structure of the original design. The theoretical complexity of reverse-engineering similar key-based obfuscation schemes for circuits has been analysed in [21, 24], where such systems were shown to be "provably secure" because of the high computational complexity.

3.5 Determination of Unreachable States

The construction of the *obfuscation state space* requires the determination of *unreachable states* in a given circuit. Figure 5 shows the steps of determining the set of unreachable states for a selected set of S state elements in a given circuit. First, all the possible 2^S state bit combinations are generated for the S state elements. Then, each state of these 2^S states are subjected to *full sequential justification* at the inputs of the selected S state elements, using *Synopsys Tetramax*. The justified states are discarded, while the states which fail justification are collected to form the set U of structurally unreachable states.

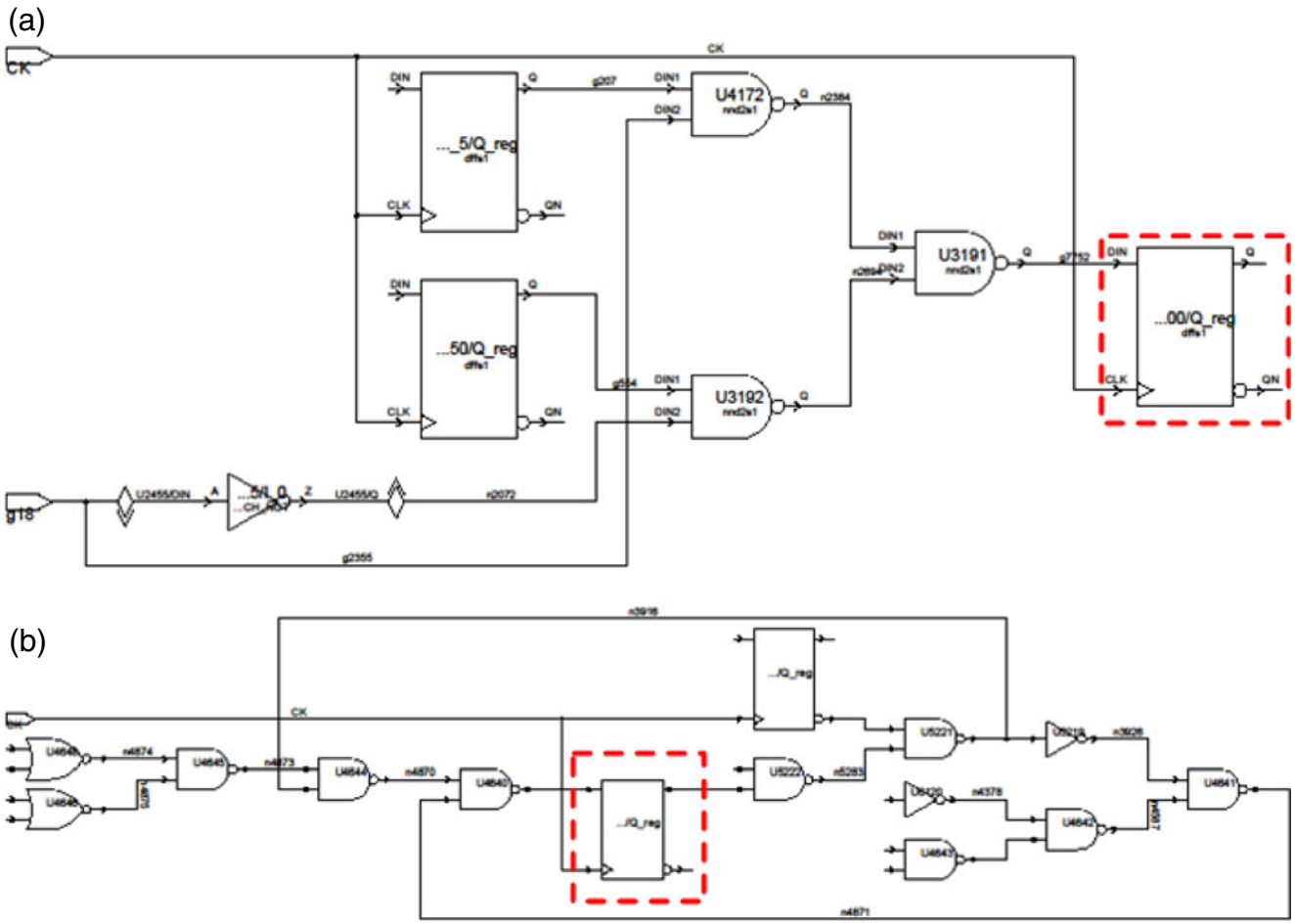

Fig. 4 Comparison of input logic cones of a selected flip-flop in *s*15850: original design (**a**) and obfuscated design (**b**)

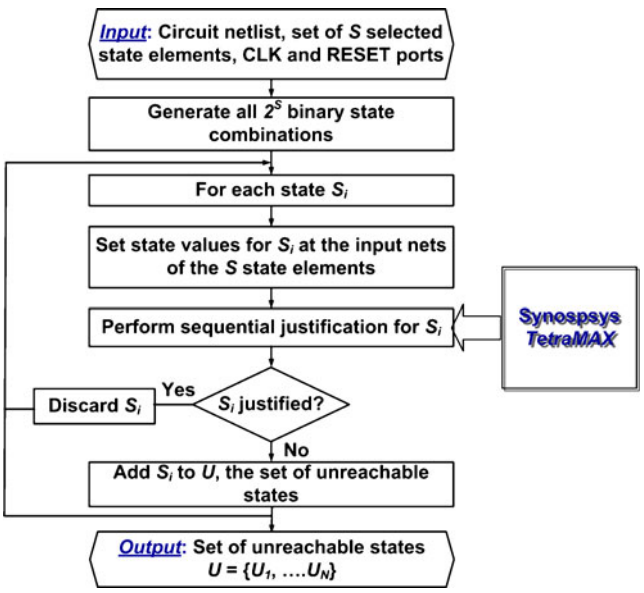

Fig. 5 Steps to find unreachable states for a given set of *S* state elements in a circuit

3.6 Test Generation for Trojan Detection

Since deterministic test pattern generation for the Trojan population is practically infeasible due to the inordinately large number of possible Trojans, we apply a statistical approach to sample and simulate a representative set of Trojan instances (10K–20K) from the total population of Trojans. First, the signal probabilities at the internal nodes of the circuit are estimated by the application of a large set of random vectors to the circuit. From the signal probability (S_p) of the internal nodes, which indicate the rareness of a logic-0 or logic-1 event in those nodes, we select a set of candidate trigger nodes with S_p less than a specified trigger threshold (θ). Next, starting from a large set of *weighted random vectors*, a smaller testset is generated to excite each of these candidate trigger nodes to its rare value at least N times, where N is a given parameter. This is done because excitation of each rare node individually to its corresponding rare value multiple times is likely to increase the probability of the Trojans triggered by them to get activated, as shown by the analysis in [13].We

have observed through extensive simulations on both combinational (ISCAS-85) and sequential (ISCAS-89) benchmark circuits, that such a statistical test generation methodology can achieve higher Trojan detection coverage than weighted random vector set, with 85% reduction in test length on average [13]. Note that for sequential circuits, we assume a *full-scan* implementation. Sequential justification is applied to eliminate *false Trojans*, i.e. Trojans which cannot be triggered during the operation of the circuit.

3.7 Determination of Effectiveness

To determine the decrease in *potency* of the Trojans by the proposed scheme, we reduce a given vector set to eliminate those vectors with state values in the *obfuscation state space*. We then re-simulate the circuit with the reduced test set to determine the Trojan coverage. The decrease in the Trojan coverage obtained from the reduced test set indicates the Trojans which are activated or effective only in the *obfuscation state space* and, hence, become benign.

To determine the increase in *detectability* of the Trojans, we compare the S_p values at the Trojan trigger nodes between two cases: (1) a large set of random vectors, and (2) a modified set of random vectors which ensure operation of the obfuscated design in only *normal mode*. The increase in Trojan detectability is estimated by the percentage of circuit nodes for which the S_p values differ by a pre-defined threshold. The difference in estimated S_p prevents an adversary from exploiting the true rare events at the internal circuit nodes in order to design a hard-to-detect Trojan. On the other hand, true non-rare nodes may appear as rare in the obfuscated design, which potentially serve as *decoy* to the adversary. The above two effects are summed up by the increase in Trojan detection coverage due to the obfuscation. The coverage increase is estimated by comparing the respective coverage values obtained for the obfuscated and the original design for the same number of test patterns.

4 Integrated Framework for Obfuscation

Algorithm 1 shows the steps of the procedure ***OBFUSCATE***, which performs the obfuscation of a given gate-level circuit. The input arguments are the gate-level synthesized Verilog netlist of the given circuit, the maximum allowable area overhead, the total number of states in the *obfuscation state space*, and the length of the input key sequence (k). Initially, the number (n) of extra state elements to be added and the number (S) of existing state elements to be used for state-encoding in the initialization state space are both set to five. These S state-elements are randomly chosen for state-encoding in the *initialization state space*, and their unreachable states are determined by the method described in Section 3.5. If the number of unreachable states found is not sufficient for the required number of states in the *obfuscation state space*, another random selection of S state elements is made and the process is continued until sufficient unreachable states are found. Once this is done, state encoding for the extra state elements and random state transitions for the 2^n states of the n extra state elements is generated. Next, an initialization key sequence of length k is selected randomly. The RTL of state transitions of the two separate set of flip-flops for the *initialization state space* is generated. As mentioned in Section 3, the RTL is constructed in a way that ensures that the chosen original state elements and the extra state elements act together as

Algorithm 1 Procedure *OBFUSCATE*

Generate the obfuscated netlist from a given circuit netlist

Inputs: Circuit netlist, maximum area overhead (*max_area_overhead*), total number of states in the *obfuscation state space*, length of *initialization key sequence* (k)

Outputs: Obfuscated circuit netlist, *initialization key sequence*

```
1:  Set no. of extra state elements to be added: n ← 8
2:  Set no. of original state elements to be used for state
    encoding: S ← 5
3:  repeat
4:    repeat
5:      Select S state elements randomly from circuit
        netlist
6:      Determine unreachable states for S state ele-
        ments using sequential justification
7:    until sufficient unreachable states found
8:    Generate state encodings for the extra state ele-
      ments
9:    Generate random state transitions for the extra
      state elements
10:   Generate random initialization key sequence of
      length k
11:   Generate RTL for obfuscation state space
12:   Integrate generated RTL with existing netlist
13:   Re-synthesize modified circuit
14:   Calculate area_overhead
15:   n ⇐ n − 1, S ⇐ S − 1
16: until area_overhead ≤ max_area_overhead
```

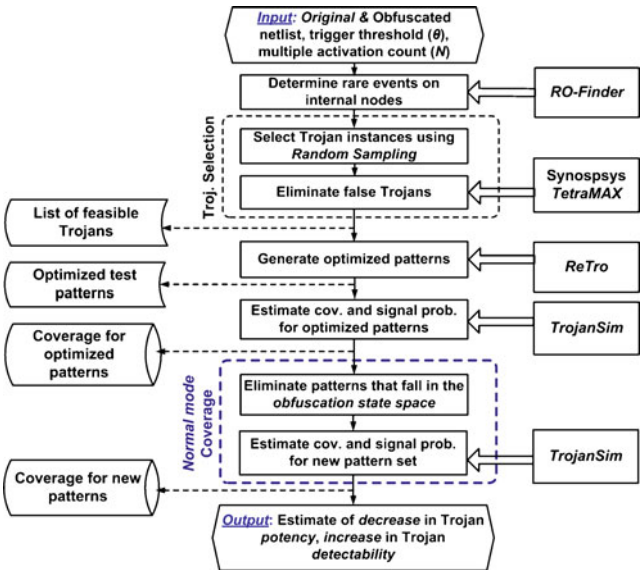

Fig. 6 Framework to estimate the effectiveness of the obfuscation scheme

parts of the same FSM during the initialization phase. The RTL is then integrated with the original gate-level netlist, with appropriate control signals to enable the operation in the two different modes. The modified circuit is then re-synthesized under input design constraints using *Synopsys Design Compiler* to generate the obfuscated version of the circuit. If the area of the re-synthesized circuit is larger than the user-specified area overhead constraint, S and n are each decreased by one and the process is repeated until the area constraint is satisfied for the obfuscated design.

We assumed the Trojan model shown in Fig. 1a. We wrote three C programs to estimate the effectiveness of the proposed obfuscation scheme for protection against Trojans. The computation of signal probabilities at the internal nodes is done by the program *RO-Finder* (**Rare O**ccurrence **Finder**). The testset for Trojan detection achieving multiple excitation of rare trigger conditions is performed by the program *ReTro* (**Re**duced pattern generator for **Tro**jans). The generation of the reduced pattern set by the elimination of the patterns with states in the *obfuscation state space* is performed by a TCL program. The decrease in the Trojan *potency* and the increase in the Trojan *detectability* are then estimated by a cycle-accurate simulation of the circuit by the simulator *TrojanSim* (**Trojan Sim**ulator). *TetraMax* is used for sequential justification of the Trojan triggering conditions. Figure 6 shows the steps to estimate the effectiveness of the obfuscation scheme [13]. The entire flow was integrated with the *Synopsys* design environment using TCL scripts. A *LEDA* 250 nm standard cell

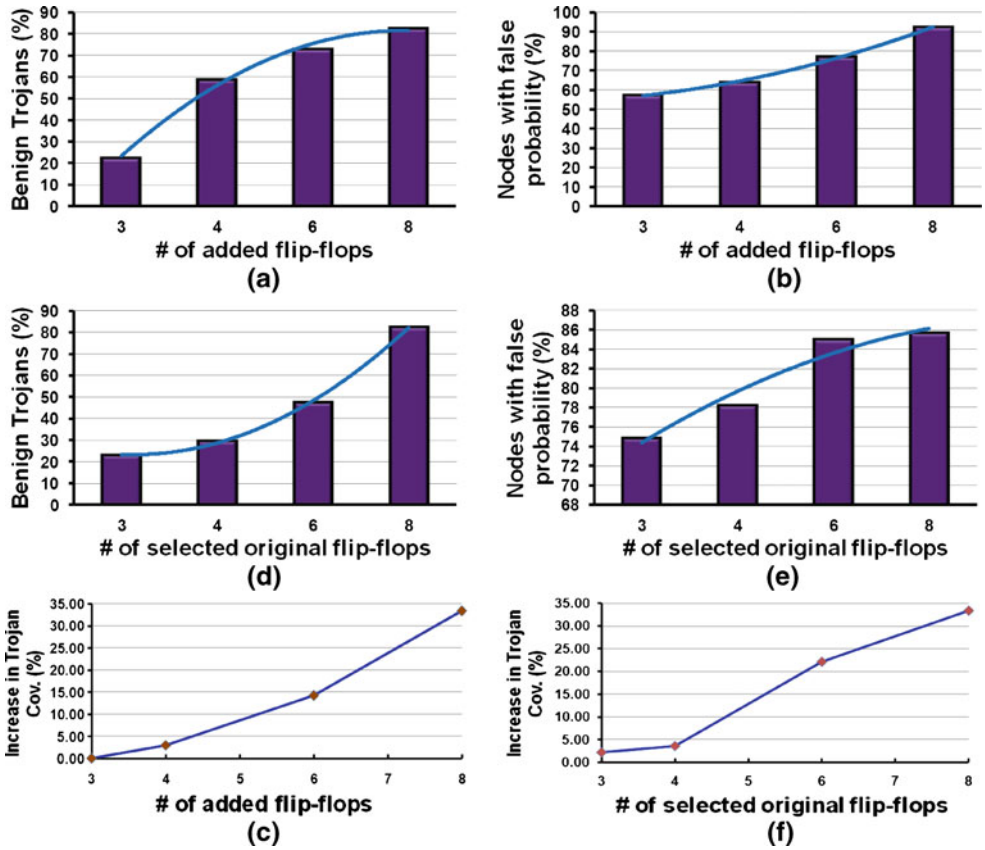

Fig. 7 Variation of protection against Trojans in *s1196* as a function of **a**–**c**: the number of added flip-flops in state encoding (S); **d**–**f**: the number of original state elements used in state encoding (n). For **a**–**c**, four original state elements were selected for state encoding, while for **d**–**f**, four extra state elements were added

library was used for logic synthesis. All simulations, test generation and logic synthesis were carried out on a Hewlett-Packard Linux workstation with a 2GHz dual-core processor and 2GB RAM.

5 Results

To verify the trends predicted in Section 3.2, we investigated the effects of adding extra state elements (n) and unreachable states determined from variable number of existing state elements (S) on the level of protection against Trojans. Figure 7 shows the variation in the percentage of Trojans rendered benign, percentage of internal nodes with false signal probability, and the percentage increase in detectability of Trojans for the *s1196* benchmark circuit. These plots clearly show the increasing level of protection against Trojans with the increasing size of the *obfuscation state space*, which matches the theoretical predictions in Section 3.2.

Tables 1 and 2 show the effects of obfuscation on increasing the security against hardware Trojans for a set of ISCAS-89 benchmark circuits with 20,000 random instances of suspected Trojans, trigger threshold (θ) of 0.2, trigger nodes (q) 2 and 4, respectively. Optimized vector set was generated using N=1000. The same value of $n + S$ applies to both sets of results. The length of the initialization key sequence was 4 ($k = 4$) for all the benchmarks. The effect of obfuscation was estimated by three metrics: (a) the fraction of the total population of structurally justifiable Trojans becoming benign; (b) the difference between the signal probabilities at internal nodes of the obfuscated and original circuit, and (c) the improvement in the *functional Trojan coverage*, i.e. the increase in the percentage of valid Trojans detected by logic testing. Note that the number of structurally justifiable Trojans (as determined by *TetraMax*) decreases with the increase in the number of trigger nodes of the Trojan, and increasing size of the benchmark circuits. From the tables it is evident that the construction of the *obfuscation state space* with even a relatively small number of state elements (i.e. a relatively small value of $n + S$) still makes a significant fraction of the Trojans benign. Moreover, it obfuscates the true signal probabilities of a large number of nodes. The obfuscation scheme is more effective for 4-trigger node Trojans. This is expected since a Trojan with larger q is more likely to select at least one trigger condition from the *obfuscation state space*.

Table 2 Effect of obfuscation on security against Trojans (100,000 random patterns, 20,000 Trojan instances, $q = 4$. $k = 4$, $\theta = 0.2$)

Benchmark circuit	Trojan instances	Obfuscation effects		
		Benign Trojans (%)	False prob. nodes (%)	Func. Troj. cov. incr. (%)
s1488	98	60.53	71.02	12.12
s5378	331	70.28	85.05	15.00
s9234	20	62.50	65.62	25.00
s13207	36	80.77	83.59	20.00
s15850	124	77.78	79.58	18.75
s38584	11	71.43	77.21	50.00

Figure 8 shows the two different effects by which Trojans are rendered benign (as discussed in Section 3.2)—i.e. some of them are triggered only in the *obfuscation state space*, while the effect of some are propagated to the primary output only in the *obfuscation state space*. In these plots, the greater effectiveness of the obfuscation approach for four-trigger node Trojans is again evident.

Figure 9 shows the improvement in Trojan detection coverage in the obfuscated design compared to the original design for the same number of random vectors. This plot illustrates the net effect of the proposed obfuscation scheme in increasing the level of protection against Trojans, with an average increase of 14.83% for $q = 2$ and 20.24% for $q = 4$. The greater effectiveness for $q = 4$ agrees with the theoretical observation in Section 3.3.

Table 3 shows the design overheads (at iso-delay) and the run-time for the proposed obfuscation scheme. The proposed scheme incurs modest area and power

Table 1 Effect of obfuscation on security against Trojans (100,000 random patterns, 20,000 Trojan instances, $q = 2, k = 4, \theta = 0.2$)

Benchmark circuit	Trojan instances	Obfus. flops ($n + S$)	Obfuscation effects		
			Benign Trojans (%)	False prob. nodes (%)	Func. Troj. cov. incr. (%)
s1488	192	8	38.46	63.69	0.00
s5378	2,641	9	40.13	85.05	1.02
s9234	747	9	29.41	65.62	1.09
s13207	1,190	10	36.45	83.59	0.56
s15850	1,452	10	40.35	68.95	2.65
s38584	342	12	33.88	81.83	0.45

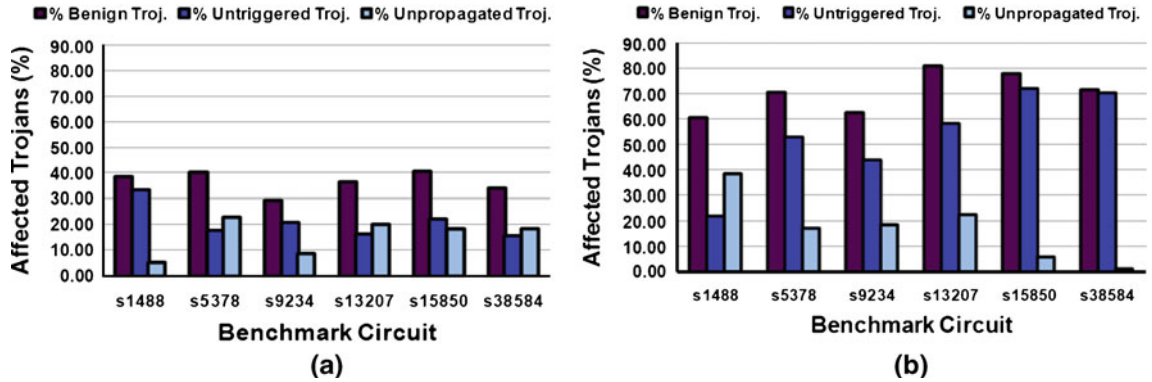

Fig. 8 Effect of obfuscation on Trojans: **a** 2-trigger node Trojans ($q = 2$), and **b** 4-trigger node Trojans ($q = 4$)

overheads, and the design overhead decreases with increasing size of the circuit. The results and trends are comparable with the STG modification based watermarking schemes proposed in [27, 40]. As mentioned earlier, the level of protection against Trojan can be increased by choosing a larger $n + S$ value at the cost of greater design overhead. The run-time presented in the table is dominated by *TetraMax*, which takes more than 90% of the total time for sequential justifications.

6 Discussion

6.1 Protection Against Malicious CAD Tools

Pre-designed third-party hardware IP blocks, which have been supplied either as synthesizable "Register Transfer Level" (RTL) descriptions (also known as "soft macros"), or as synthesized gate-level netlists (also known as "firm macros"), can be modified to implement the proposed methodology. For RTL descriptions, obfuscation of an IP module can be achieved in two ways. In the first, "direct method", the used and unused states of the circuits can be identified by direct analysis of the control and data flow graph (CDFG) of the derived from the RTL, and then additional RTL code can be automatically generated to realize the change in the STG of the circuit. An automated design flow for performing similar key-based control-flow obfuscation for the purpose of IP protection of RTL designs has been previously proposed in [11]. In the second, "indirect method", the RTL description can be synthesized to a gate-level netlist to apply the proposed technique.

The proposed technique can also be extended to multi–IP system-on-chips (SoCs), even in those cases where the communication fabric of the SoC is custom-designed. This is possible since the proposed methodology does not depend on the structure and communication protocols used by the communication fabric of the SoC. A SoC design methodology that employs key-based obfuscation of hardware IP modules has been previously proposed in [10].

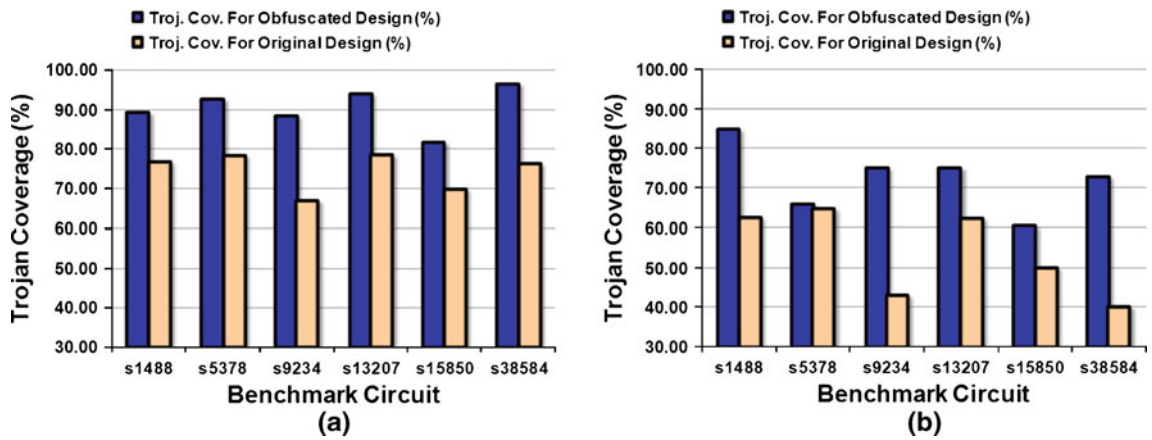

Fig. 9 Improvement of Trojan coverage in obfuscated design compared to the original design for **a** Trojans with two trigger nodes ($q = 2$) and **b** Trojans with four trigger nodes ($q = 4$)

Table 3 Design overhead (at iso-delay) and run-time[a] for the proposed design flow

Benchmark circuit	Overhead (%)		Run-time (mins.)
	Area	Power	
s1488	20.09	12.58	31
s5378	13.13	17.66	186
s9234	11.84	15.11	1,814
s13207	8.10	10.87	1,041
s15850	7.04	9.22	1,214
s38584	6.93	2.63	2,769

[a]The run time includes the sequential justification time by Synopsys *Tetramax*, which in most cases was over 90% of the total runtime

6.2 Application to Third-Party IP Modules and SoCs

Besides protecting a design in foundry, the proposed obfuscation methodology can provide effective defense against malicious modifications (manual or automated) during the IC design steps. As pointed out in Section 1, compromised CAD tools and automation scripts can also insert Trojans in a design [15, 31]. Obfuscation can prevent insertion of hard-to-detect Trojans by CAD tools due to similar reasons as applicable in a foundry. It prevents an automatic analysis tool from finding the true rare events, which can be potentially used as Trojan triggers or payloads. Moreover, since large number of states belong to the obfuscation state space, an automation tool is very likely to insert a Trojan randomly that is only effective in the obfuscation mode. Note that since we obfuscate the gate-level netlist, protection against CAD tools can be achieved during the design steps following logic synthesis (e.g. during physical synthesis and layout).

To increase the scope of protection by encompassing the logic synthesis step, we propose a small modification in the obfuscation-based design flow. Figure 10 compares a conventional IP-based SoC

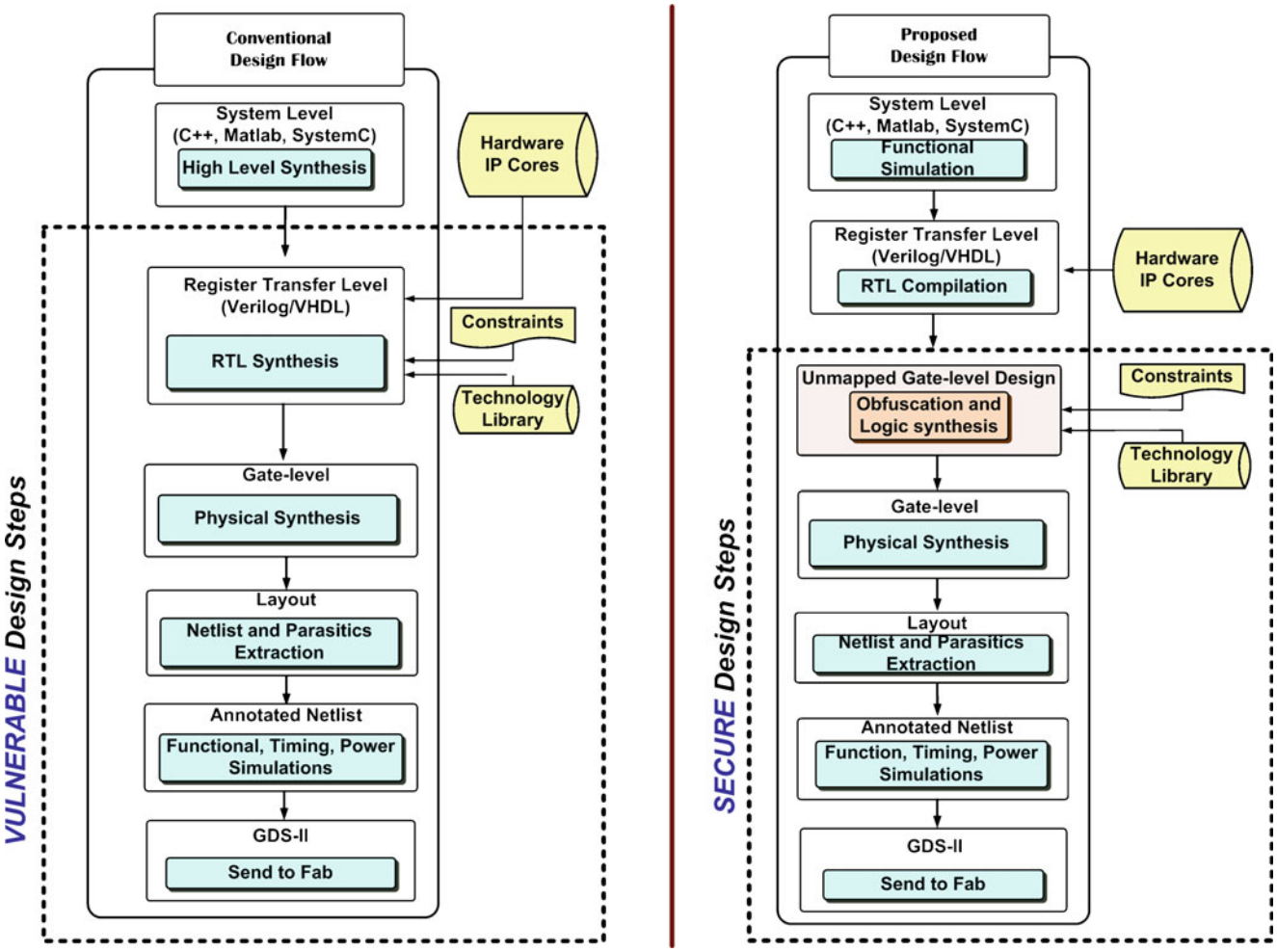

Fig. 10 Comparison of conventional and proposed SoC design flows. In the proposed design flow, protection against malicious modification by untrusted CAD tools can be achieved through obfuscation early in the design cycle

design flow with the proposed modified design flow. In the conventional design flow, the RTL is directly synthesized to a technology mapped gate-level netlist, and obfuscation is applied on this netlist. However, in the modified design flow, the RTL is first *compiled* to a technology independent (perhaps unoptimized) gate-level description, and obfuscation is applied on this netlist. Such a practice is quite common in the industry, and many commercial tools support such a compilation as a preliminary step to logic synthesis [17]. The obfuscated netlist is then optimized and technology mapped by a logic synthesis tool. Note that the logic synthesis step now operates on the obfuscated design, which protects the design from potential malicious operations during logic synthesis. Also, the RTL *compilation* (without logic optimization) is a comparatively simpler computational step for which the SoC design house can employ a trusted in-house tool. This option provides an extra level of protection.

This proposed obfuscation methodology also provides protection against malicious CAD tools in Field Programable Gate Array (FPGA) based design flows. As noted in [15], the main threat of Trojan insertion in such a flow comes from the CAD tools which convert the RTL description of a design to the FPGA device specific configuration *bitstream*. Typically, the fabric itself can be assumed to be Trojan-free [15]. Similar to the SoC design flow, we propose a small modification to the FPGA design flow that maximizes the scope of protection against FPGA CAD tools. Figure 11 shows the proposed design flow. The RTL corresponding to the circuit can be "compiled" to a unoptimized, technology-independent gate-level netlist. This netlist can then be obfuscated, and the obfuscated design can then be optimized and mapped by either third-party CAD tools or vendor-specific tools to a netlist in an intermediate format. This netlist is then converted to a vendor-specific bitstream format by the FPGA mapping tool to map the circuit to the FPGA. Note that as before, the security against malicious CAD tools propagate to lower levels of design abstraction.

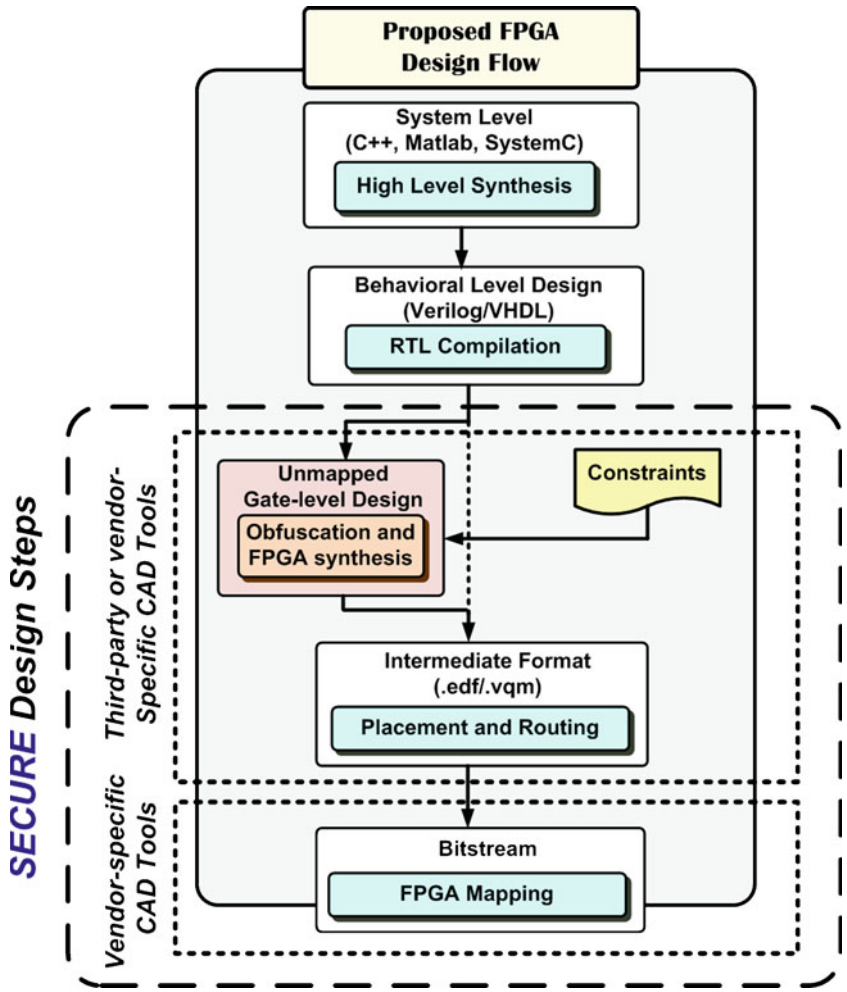

Fig. 11 Proposed FPGA design flow for protection against CAD tools

6.3 Improving Level of Protection and Design Overhead

Equation 6 suggests that for large designs with a significantly large *original state space*, to attain satisfactory levels of design obfuscation, it is necessary to have the *obfuscation state space* much larger than the *original state space*. This can be achieved by either: (a) addition of a large number of extra state elements, or (b) using a large number of unreachable states in the *obfuscation state space*. However, finding large number of unreachable states through sequential justification in a complex design is extremely computationally expensive. To keep the problem computationally tractable and reduce the design overhead, we propose a systematic approach to modify the state transition function as shown in Fig. 12. The n extra state elements are grouped into p different groups to form parallel FSMs PSM_1 through PSM_p, and RTL code for each of them is generated separately. Similarly, the S existing state elements (corresponding to the unreachable states) used for state encoding in the *obfuscation state space* are grouped in q different groups PSM'_1 through PSM'_q. RTL code for each of the parallel FSMs PSM'_1 through PSM'_q is generated separately based on the unreachable states. Such a scheme of having multiple parallel FSMs to design the *obfuscation state space* achieves similar design obfuscation effects, without incurring high computational complexity and design overhead.

6.4 Protection Against Information Leakage Trojans

Although in our simulations we considered the Trojans according to the model shown in Fig. 1a, as pointed out in Section 2, the proposed methodology can also help to protect against Trojan attacks that aim at leaking secret information about internal state of the circuit, either in the form of a data-stream (similar to Fig. 1c) or as side-channel signature (similar to Fig. 1d). Such a Trojan is shown in Fig. 13, where it transmits out a secret cryptographic key through a covert communication channel by "sniffing" the values on the communication bus. *Bus scrambling* or bus re-ordering is a simple technique to resist against this kind of an attack, so that the data transmitted out by the Trojan is also scrambled. To overcome this defense mechanism, the adversary has to figure out the actual order of bits in the scrambled bus to correctly interpret the collected data. Figuring

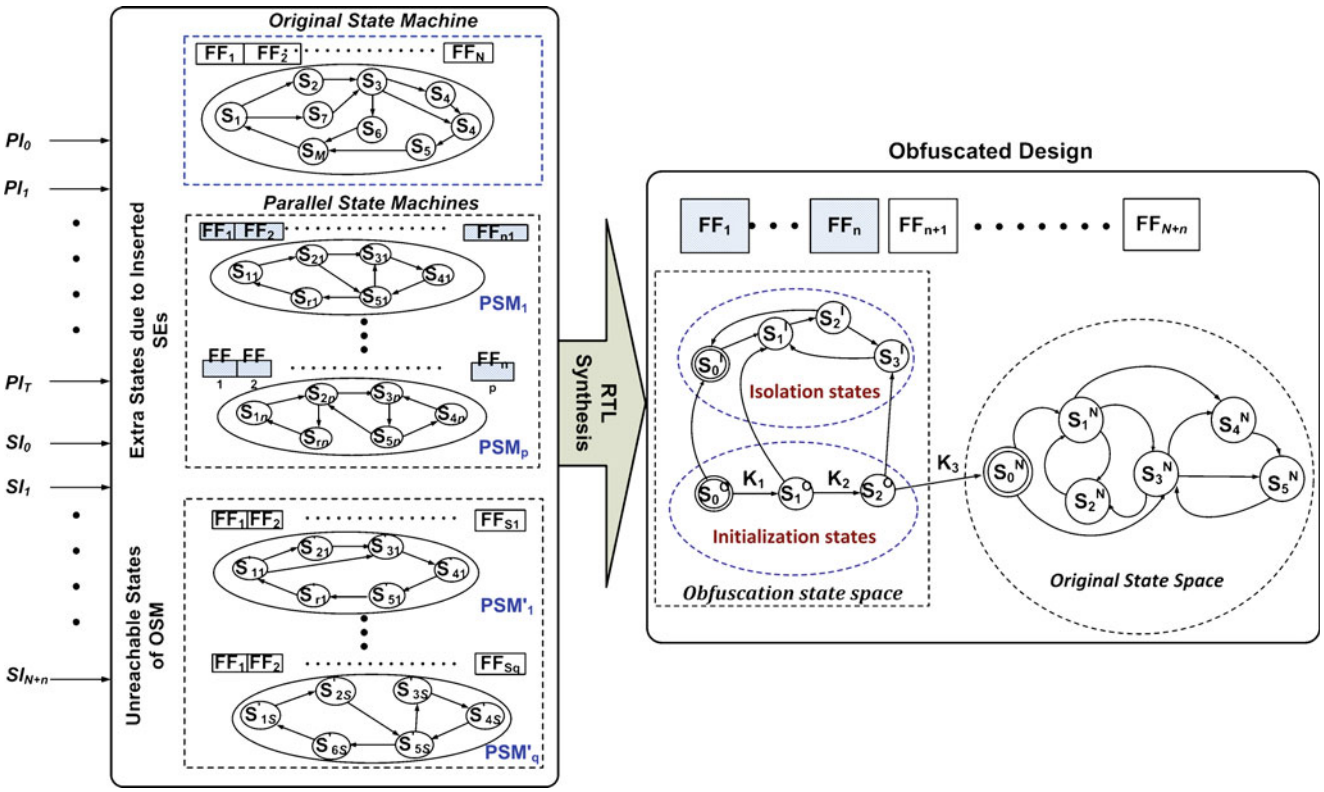

Fig. 12 Obfuscation for large designs can be efficiently realized using multiple parallel state machines which are constructed with new states due to additional state elements as well as unreachable states of original state machine

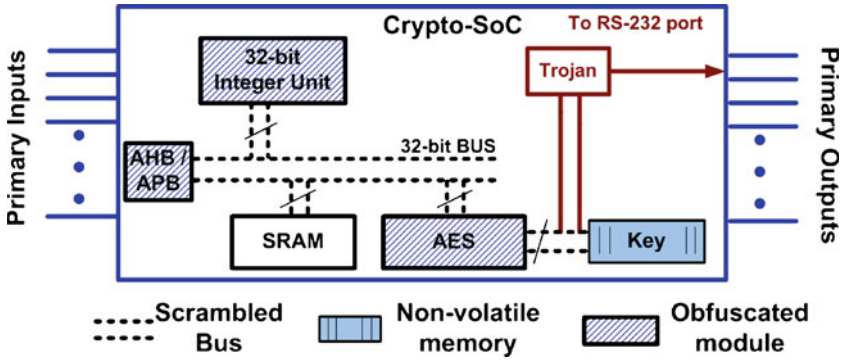

Fig. 13 Functional block diagram of a crypto-SoC showing possible Trojan attack to leak secret key stored inside the chip. Obfuscation coupled with bus scrambling can effectively prevent such attack

out the actual order of the bits in a n-bit bus by simulations will require a search among $n!$ possibilities, e.g. $\sim 2 \times 10^{35}$ possibilities for a 32-bit data bus. However, since the attacker has access to the design, he/she is likely to perform structural analysis of the design to determine the order of bits in the bus. If the functional blocks are not obfuscated, one can employ equivalence checking between a functional block in the design and a corresponding reference design to identify the order of bits in both input/output bus for a module. For example, an attacker can perform formal verification between the integer unit in Fig. 13 and a functionally equivalent reference design to find the port association.

However, if all the modules in the given SoC are obfuscated using the proposed approach, it would be practically infeasible for a formal verification tool to establish structural equivalence [10]. The other choice left to the attacker is to simulate the circuit by applying input vectors. For simplicity, assume all modules in the SoC are initialized simultaneously and by the same initialization key sequence. Then, to reach the *normal mode* of operation, the adversary needs to first apply the correct *unknown* initialization vectors in correct order to enable normal operating mode of the IC. Only then the adversary would be able to establish the actual bus order through simulations, the complexity of which has already been shown to be extremely high. The probability of succeeding in reaching the normal mode by the application of random vectors to the primary input of a SoC with M primary inputs and an initialization key sequence length of N is $\frac{1}{2^{M \cdot N}}$. Assuming the SoC shown in Fig. 13 has 32 inputs, and assuming the length of the *initialization key sequence* to be 4, the probability of the adversary taking the obfuscated SoC to the *normal mode* is $\sim 10^{-39}$. The width of the data bus for the key is typically 128 or 256, which would increase the complexity exponentially. A similar argument can be presented for Trojans of the type shown in Fig. 1d.

7 Conclusion

In this paper, we have presented a novel application of design obfuscation to achieve effective protection against hardware Trojans. We have shown that obfuscation can provide comprehensive protection against conditionally triggered Trojan attacks including defense against untrusted CAD tools, for both SoC and FPGA. We have also shown that obfuscation can provide protection against Trojans that tries to leak secret information from an IC. The level of obfuscation and hence the protection can be increased by state-space blow-up and use of unreachable states. The required design modifications can be easily automated and integrated with conventional design flow. Simulation results for a set of benchmark circuits show that a well-formulated obfuscation scheme can provide simultaneous protection against hardware Trojan and IP piracy at low design overhead. Future work will involve application of the proposed approach to higher level of design abstraction such as RTL IPs.

References

1. Aarestad J, Acharyya D, Rad R, Plusquellic J (2010) Detecting Trojans through leakage current analysis using multiple supply pad. IEEE Trans Inf Forensics Secur 5(4):893–904
2. Adee S (2008) The hunt for the kill switch. IEEE Spectrum 45(5):34–39
3. Agrawal D, Baktir S, Karakoyunlu D, Rohatgi P, Sunar B (2007) Trojan detection using IC fingerprinting. In: IEEE symposium on security and privacy
4. Alkabani Y, Koushanfar F (2007) Active hardware metering for intellectual property protection and security. In: USENIX security symposium
5. Alkabani Y, Koushanfar F (2009) Consistency-based characterization for hardware Trojan detection. In: International conference on CAD
6. Alkabani Y, Koushanfar F, Potkonjak M (2007) Remote activation of ICs for piracy prevention and digital right management. In: International conference on CAD

7. Banga M, Hsiao MS (2008) A region based approach for the identification of hardware Trojans. In: International workshop on hardware-oriented security and trust
8. Barak B, Goldreich O, Impagliazzo R, Rudich S, Sahai A, Vadhan SP, Yang K (2001) On the (im)possibility of obfuscating programs. In: Cryptology conference on advances in cryptology
9. Brzozowski M, YarmolikVN (2007) Obfuscation as intellectual rights protection in VHDL language. In: CISIM
10. Chakraborty RS, Bhunia S (2009) HARPOON: an obfuscation-based SoC design methodology for hardware protection. IEEE Trans CAD 28(10):1493–1502
11. Chakraborty RS, Bhunia S (2011) RTL hardware IP protection using key-based control and data flow obfuscation. In: International conference on VLSI design
12. Chakraborty RS, Paul S, Bhunia S (2008) On-demand transparency for improving hardware Trojan detectability. In: International workshop on hardware-oriented security and trust
13. Chakraborty RS, Wolff F, Paul S, Papachristou C, Bhunia S (2009) MERO: a statistical approach for hardware Trojan detection. Lect Notes Comput Sci 5747:396–410
14. Chou T, Roy K (1996) Accurate power estimation of CMOS sequential circuits. IEEE Trans VLSI 4(3):369–380
15. DARPA BAA06-40. TRUST for integrated circuits. [Online]. Available: http://www.darpa.mil/BAA/BAA06-40mod1/html. Accessed Oct 2011
16. Du D, Narasimhan S, Chakraborty RS, Bhunia S (2010) Self–referencing: a scalable side-channel approach for hardware Trojan detection. In: Workshop on cryptographic hardware and embedded systems
17. Interra Systems, Concorde-Fast Synthesis. [Online]. Available: http://www.interrasystems.com/eda/eda_concorde.php. Accessed Oct 2011
18. Jin Y, Makris Y (2008) Hardware Trojan detection using path delay fingerprint. In: International workshop on hardware-oriented security and trust
19. Jin Y, Kupp N, Makris Y (2009) Experiences in hardware Trojan design and implementation. In: International workshop on hardware-oriented security and trust
20. Kim L-W, Villasenor JD, Koc CK (2009) A Trojan-resistant system-on-chip bus architecture. In: MILCOM
21. Koushanfar F (2011) Provably secure active IC metering techniques for piracy avoidance and digital rights management. IEEE Trans Inf Forensics Secur. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5966342. Accessed Oct 2011
22. Koushanfar F, Mirhoseini A, Alkabani Y (2010) A unified submodular framework for multimodal IC Trojan detection. In: International conference on information hiding
23. Lin L, Burleson W, Parr C (2009) MOLES: malicious off-chip leakage enabled by side-channels. In: International conference on CAD
24. Lynn B, Prabhakaran M, Sahai A (2004) Positive results and techniques for obfuscation. In: International conference on the theory and applications of cryptographic techniques
25. Najm FN (1993) Transition density: a new measure of activity in digital circuits. IEEE Trans CAD 14(2):310–323
26. Narasimhan S, Du D, Chakraborty RS, Paul S, Wolff F, Papachristou C, Roy K, Bhunia S (2010) Multiple-parameter side-channel analysis: a non-invasive hardware Trojan detection approach. In: International symposium on hardware-oriented security and trust
27. Oliveira AL (1999) Watermarking-based copyright protection of sequential functions. In: Design automation conference
28. Potkonjak M, Nahapetian A, Nelson M, Massey T (2009) Hardware Trojan horse detection using gate-level characterization. In: Design automation conference
29. Rad RM, Wang X, Tehranipoor M, Plusquellic J (2008) Power supply signal calibration techniques for improving detection resolution to hardware Trojans. In: International conference on CAD
30. Ravi S, Raghunathan A, Chakradhar S (2004) Tamper resistance mechanisms for secure embedded systems. In: VLSI design
31. Roy JA, Kaushanfar F, Markov IL (2008) Circuit CAD tools as a security threat. In: International workshop on hardware-oriented security and trust
32. Roy JA, Koushanfar F, Markov IL (2008) EPIC: ending piracy of integrated circuits. In: Design, automation and test in Europe
33. Tehranipoor M, Koushanfar F (2010) A survey of hardware Trojan taxonomy and detection. IEEE Des Test Comput 27(1):10–25
34. Thicket™ family of source code obfuscators. [Online]. Available: http://www.semdesigns.com. Accessed Oct 2011
35. Torunoglu I, Charbon E (2000) Watermarking-based copyright protection of sequential functions. IEEE J Solid-State Circuits 35(3):434–440
36. Wang C, Davidson J, Hill J, Knight J (2001) Protection of software-based survivability mechanisms. In: International conference on dependable systems and networks
37. Wolff F, Papachristou C, Bhunia S, Chakraborty RS (2008) Towards Trojan-free trusted ICs: problem analysis and detection scheme. In: Design, automation and test in Europe
38. Xakellis MG, Najm FN (1994) Statistical estimation of the switching activity in digital circuits. In: Design automation conference
39. Yotsuyanagi H, Kinoshita K (1998) Undetectable fault removal of sequential circuits based on unreachable states. In: VLSI test symposium
40. Yuan L, Qu G (2004) Information hiding in finite state machine. Lect Notes Comput Sci 3200:340–354

Rajat Subhra Chakraborty received the B.E. (Hons.) degree in electronics and telecommunication engineering from Jadavpur University, Kolkata, India in 2005, and the Ph.D degree in computer engineering from Case Western Reserve University in Cleveland, OH, in 2010. Currently he is an assistant professor in the Computer Science and Engineering Department at Indian Institute of Technology, Kharagpur (IIT Kharagpur) in India. He has worked as a CAD software engineer in National Semiconductor at Bangalore, India, and has held internship positions at National Semiconductor and AMD. His research interests are hardware security including hardware IP protection, low-power VLSI, digital watermarking and nano-circuit design methodologies.

Swarup Bhunia received the B.E. (Hons.) from Jadavpur University, Kolkata, India, in 1995, the M.Tech. degree from the Indian Institute of Technology (IIT), Kharagpur, India, in 1997, and the Ph.D. degree from Purdue University, West Lafayette, IN, in 2005. Currently, he is an associate professor of electrical engineering and computer science at Case Western Reserve University, Cleveland, OH. He has worked in the semiconductor industry on RTL synthesis, verification, and low power design for about three years. His research interest includes low-power and robust VLSI design, adaptive nanocomputing, and bio-implantable devices for neural engineering. Dr. Bhunia

received the 2005 SRC Technical Excellence award as a team member, the Best Paper Award in International Conference on Computer Design (ICCD 2004), the Best Paper award in Latin American Test Workshop (LATW 2003), and the Best Paper Nomination in Asia and South Pacific Design Automation Conference (ASP-DAC 2006). He has served in the technical program committee of Design Automation and Test conference in Europe (DATE 2006–2007), International Symposium on Low Power Electronics and Design (ISLPED 2007), Test Technology Educational Program (TTEP 2006-2007), and on the Program Committee of International Online Test Symposium (IOLTS 2005).