Watermarking Techniques for Electronic Circuit Design

Edoardo Charbon[1] and Ilhami Torunoglu[2]

[1] Swiss Federal Institute of Tecnology, Lausanne, Switzerland
edoardo.charbon@epfl.ch
[2] Canesta Inc., San Jose, California U.S.A.
ilhami@canesta.com

Abstract. Enforcing copyright laws related to circuit design is usually hard, since discovery and proof of infringements are required beyond reasonable doubt. Proving that a given circuitry is trivially derived from a patented technique or method is in general a painstakingly slow task, often requiring reverse-engineering and forensic analysis of fabricated chips. These techniques are so complex that their application to large collections of commercial products is almost always prohibitive. Watermarking is one of several techniques available today to deter copyright infringement in electronic systems. The technique consists of implanting indelible stamps in the circuit's inner structure, while not disrupting its functionality or significantly degrading its performance. In this paper a series of methods is proposed for the creation of watermarks at several stages of the design cycle, from high-level design to layout. Algorithms are described for implanting robust watermarks to minimize the overhead and, ultimately, to reduce the impact on performance. Detection methods are discussed in the presence of infringement. The resilience of the methods in several tampering regimes is estimated.

1 Introduction

With the introduction of large-scale system-on-chip (SOC) design paradigms, the VLSI semiconductor industry is undergoing a revolution. As a consequence, a significant increase in operating speeds and design productivity is expected. Circuit components will be available to integrators in electronic form rather than in silicon. Such components, known as virtual components, will be delivered for a particular application according to a specific contract. Due to the new medium of storage and exchange, virtual components are prone to infringements and abuses, i.e. uses beyond the original contracts and/or by unauthorized integrators. Protecting the Intellectual Property (IP) rights of the authors of virtual component is an emerging multi-disciplinary field encompassing several aspects of today's IC design process.

Existing legal devices, such as patents, trade secrets, copyrights, and non-disclosure agreements are aimed at deterring potential infringement. However, the real Achille's heal of the system is currently the enforcement of such rights.

The goal of IP protection as a whole is to provide necessary technologies to make enforcement a more manageable task while deterring infringement at all levels of a design. A possible such scheme requires the capability of effectively detecting and subsequently tracking IP infringement cases. This task can be accomplished by a process known as *watermarking*. The method consists of embedding a unique code, or watermark, which exploits the IP's unique features. Fundamental requirements for a watermark are that it be (1) *transparent*, i.e. not interfering with the design functionality, (2) *robust*, i.e. hard to remove or forge, (3) *detectable*, i.e. easy to extract from the design. The process used for managing watermarks must not necessarily be proprietary, while the code used in the encryption process should be secret for any released IP.

There exist three basic types of virtual component, based on its characterization through (a) behavioral description (*soft IP*), (b) structural description (*firm IP*), and (c) physical description (*hard IP*). Hard IPs are generally released as a plug-and-play component for one or more technologies, while firm and soft IPs are partially implemented versions of a design, thus a number of degrees of freedom are still available to the integrator and potentially all technologies can be used for implementation. Due to the diversity of the implementation landscape of IPs, their protection is also a complex task.

Recently, watermarking has been proposed to protect digital audio-visual IPs. The literature is very extensive on the subject. Here we mention only two works [1, 2] because they allow us to introduce some of the techniques used in this paper. The main method described in [1, 2], though with small variations, essentially consist of superimposing a pseudo-random noise to the signal of the record. Such noise, though completely inaudible, can be easily detected via digital signal processing methods.

Schemes based on watermarking have been recently proposed for electronic IPs as well. In [3, 4] the watermark assumes the form of a extraneous bit stream, hidden inside large Field Programmable Gate Arrays (FPGAs). The watermark is stored in some of the unused configurable control logic blocks of the FPGA. Each unused lookup table is responsible for storing a single bit in the code. All the configurable control elements used in the watermark are then routed in the design for mimetic purposes. This is done in such a manner to avoid any impact on the original functionality. The method has later been refined in [5] where the signature is modified before being embedded so as to mimic the statistical properties of the existing design, hence reducing detectability. One issue in our opinion still remaining in this approach is the fact that to date watermarked lookup tables do not reflect any functionality, thus representing a potential weak point in the scheme.

In [6, 7] it was proposed to incorporate several watermarks, distributed over all the abstraction levels of a given design. The techniques differ depending on the abstraction level they are applicable to. At the physical design level, the watermark assumes the form of a set of topological constraints governing the relative position, orientation, and, possibly, scaling of the devices or gates of the circuit. At structural and RTL level, constraints on the structure of a selected set of nets are used to represent the watermark. Watermarks are created at multiple levels of hierarchy simultaneously. This is provably a very robust and flexible techniques since it requires the elimination of watermarks placed at every level of abstraction. Erasing one level, by resynthesizing a logic circuitry for example, may erase the watermarks created at that abstraction level and, possibly, at lower levels, while leaving higher abstraction levels (and their watermarks) intact. As expected, this is the a particularly costly and complex scheme. Another advantage of this scheme is the fact that forgery can be traced and a history of tampering actions can be derived.

Several authors have proposed to use other design constraints to implant watermarks. In [8] fixed placement and delay constraints implemented the watermark. In [9] a sequence of nodes in a multi-level logic function was permuted according to a seeded pseudo-random selection scheme. In [10, 11] schemes have been proposed to implant watermarks in regular sequential functions by modifying the original function in a structured fashion. In [8], the authors proposed two methods for embedding signatures in a design. The first method, general in nature, can be applied to several abstraction levels. It consists of adding clauses to a satisfiability problem. For convenience, the authors restricted themselves to a subset of the problem known as (3SAT), where each clause added to the original problem codes a sequence of symbols or a section of the signature to be embedded. IF a design can be represented in terms of a 3SAT formulation, then it will be possible to embed an arbitrary signature in it. The second method is applicable to firm and hard IPs only, as it consists of formulating the signature in terms of a set of delay sub-constraints which are found in delay constrained signal paths. Constraints on floorplanning block relative locations are also used to embed a signature.

In the case more than one party is involved in creation of an IP, any of the above techniques does not guarantee that the infringements can be tracked. Watermarking should be performed simultaneously at various levels of abstraction [6]. The goal is to improve the robustness of the approach and to allow quick and accurate tracking of the last licensee, who ultimately caused the infringement. In this paper we will describe a consistent strategy for incorporating watermarks in an electronic circuit at all abstraction levels, i.e. in all phases of a design flow.

At least two types of watermarking schemes exist. The first scheme, known as *active watermarking*, consists of integrating the watermark as a part of the design process, thus allowing the creation of an arbitrarily high number of uniquely watermarked designs. In the second scheme, known as *passive watermarking* or *fingerprinting*, one creates a unique and compact representation of a design at any abstraction level. This representation, known as *digital signature*, can be used to track infringement after it occurred by simply extracting the signature from an existing design and comparing it with the original one. To avoid false claims, a third party organization should maintain a database of all registered signatures for which protection is sought [12]. Both approaches are robust, since the deletion of the watermark results, with high probability, in the removal of wanted functionality.

IP protection based on watermarking consists of two phases: *synthesis* and *detection*. The synthesis phase is fully characterized by (a) a set of algorithms translating design features onto a unique watermark, and (b) P_u, the odds that an unintended watermark is detected in a design. The detection phase is fully characterized by (c) P_m, the probability of a miss and (d) $P_f = P_u$, the probability of a false alarm. The set of algorithms proposed in this paper are all classifiable based on the above criteria.

In order for any protection scheme to be effective it is necessary to define how design flows must be modified, if at all. Moreover, to allow effective infringement detection, a well-defined detection protocol must be set up. A well-known example of such protocol is the one used by law enforcement organizations to identify criminals based on natural fingerprints and, more recently, DNA samples found on the scene. In the case of IP infringements, all existing prevention and detection methods are generally used simultaneously to ensure maximal impact.

The paper is structured as follows. Section 2 describes methods to implant a watermark at the highest level of abstraction, i.e. at the Hardware Description Language (HDL) levels. Section 3 outlines the creation of a watermark at an intermediate level, i.e. at a structural level. Section 4 describes watermarking techniques at the lowest levels of abstraction, i.e. at the physical implementation of the circuit. A number of examples are presented in Section 5.

2 HDL Level Watermarking

Most HDL representations contain one or more sequential functions. In its most general form, a sequential function is a function that transforms input sequences into output sequences. Regular sequential functions are functions such that at any stage the output symbol depends only on the sequence of input symbols which have been already received. Any regular sequential function operating on finite input/output sets can be specified by means of a Finite State Machine (FSM).

A FSM is a discrete dynamical system translating sequences of input vectors into sequences of output vectors and it is generally represented by State Transition Graphs (STGs) and State Transition Tables (STTs). A STG is a graph whose nodes represent the states of the FSM and whose edges determine the input/output conditions for a state to state transition. By convention, an edge is labeled by the input/output pair causing the transition.

In real-world sequential designs, although not explicitly specified using STGs and STTs, FSMs appear in different forms. For example, case statements in VHDL and Verilog HDL are represented as FSMs using a STG or STT by HDL compilers. FSMs also appear in embedded software, especially to define the device drivers and interface protocols. In large sequential designs, usually several such small FSMs exist which can be used to watermark the entire design. By watermarking all or a selected subset of these FSMs, tampering resilience can be reached while ensuring the method's feasibility.

The essence of this technique is to find an unused input/output symbol sequence and use it as the watermark. This task can be performed by using the STG representation of the regular sequential function. By visiting every state and finding the unused input/output symbol pairs, one can determine the candidate subset of such symbol pairs at each state in the FSM.

After calculating the required input/output symbol sequence length which satisfies given uniqueness constraints, i.e. constraints on P_u, one can generate a sequence by selecting enough input/output symbol pairs. If the found input/output symbol pairs are not sufficient, then one can create extra ones by augmenting the input and/or output alphabets. The estimation of P_u and the derivation of the length of the input/output symbol sequence can be found in [13].

Finally, by connecting the states, one can generate a trace in the FSM. Some selections of input/output symbol sequences and the states may generate large FSMs.

To capture the essence of the proposed techniques, consider the example of Figure 1. The original FSM is depicted in Figure 1(a) in terms of its STG. The FSM has two input bits and one output bit. Assume one has decided that a watermark of length 2 is satisfactory and suppose the proposed watermark is represented by input/output sequence ((00,1)(11,0)). Figure 1(b) illustrates the new FSM obtained after augmentation and state selection.

Assume that the input/output pairs available are not satisfactory. Then, in this case, the number of inputs is first incremented by one (for illustrative purposes). Two extra transition relations can hence be added. The resulting FSM is depicted in Figure 1(c).

In the remainder of the paper we will restrict ourselves to deterministic FSMs, using the same notation of [14] and [15].

Definition 1 *Let a FSM be a tuple $M = (\Sigma, \triangle, Q, q_0, \delta, \lambda)$, where Σ and $\triangle$ are respectively the input and output alphabets, Q is a finite set of states, $q_0 \in Q$ is the initial state, $\delta(q, a) : Q \times \Sigma \rightarrow Q \bigcup \{\phi\}$ is the transition relation, and $\lambda(q, a) : Q \times \Sigma \rightarrow Q \bigcup \{\epsilon\}$ is the output relation.*

$q \in Q$, $a \in \Sigma$, $b \in \triangle$ refer to a state, an input and an output, respectively. ϕ denotes an unspecified next state while ϵ is an unspecified output. A FSM can be identified by the mapping of all its input and output sequences, or *IO mapping.*

Definition 2 *An IO mapping is defined to be the sequence of input/output pairs $((a_1, b_1), (a_2, b_2), \ldots, (a_k, b_k)) \in (\Sigma \times (\triangle \bigcup \{\epsilon\}))^k$ specifying the output sequence of the FSM for a given input sequence.*

Let us define Σ^* and $\triangle^*$ as the sets of all strings in Σ and in $\triangle$, respectively. Let $s = (a_1, \ldots, a_k) \in \Sigma^*$ be an arbitrary input sequence and let $d = (b_1, \ldots, b_k) \in \triangle^*$ be an output sequence. Moreover, define $\lambda(q, s)$ to be the output symbol of

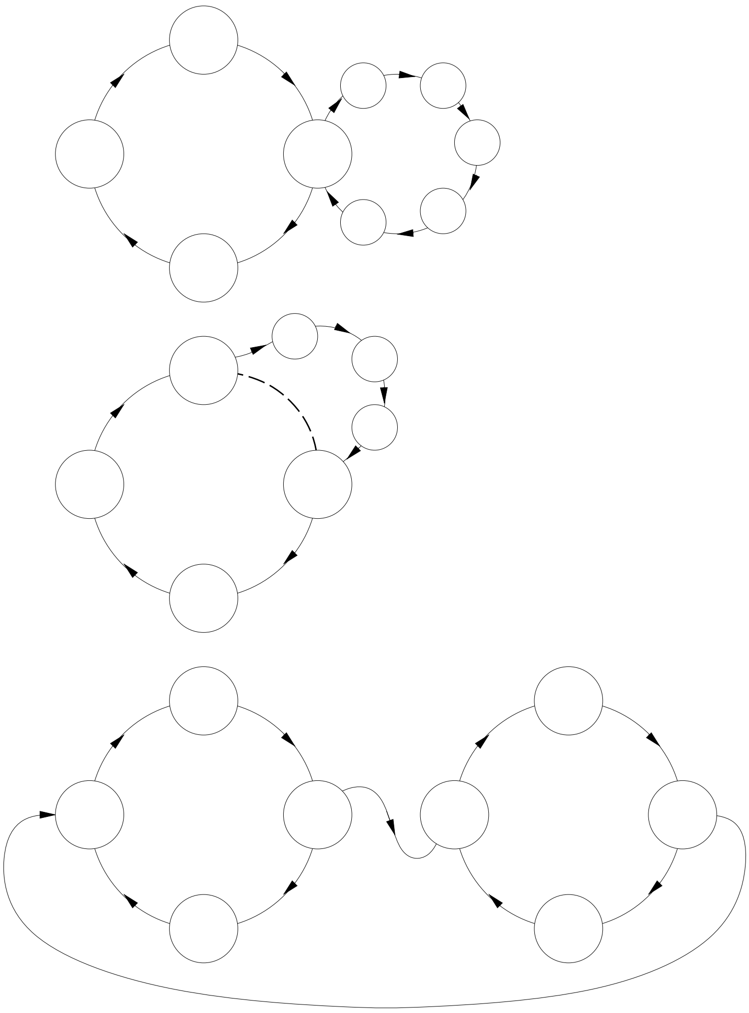


Fig. 1. An example of two possible ways of watermarking a FSM: a) original FSM; b) adding transitions; c) augmenting input and adding transitions

the FSM and $\delta(q, s)$ its state when s has been applied in state q. String s is said to be contained in M iff a state reached by applying s to state q_0 is still in M, i.e. iff $\delta(q_0, s) \in Q$.

Completely specified FSMs (CSFSMs) contain every element of set Σ^*, i.e. every input sequence in Σ^* results in a unique output sequence in $\triangle^*$. An incompletely specified FSM (ISFSM) is one in which there exist some transition relations with unspecified destination and/or output, i.e. there exist a set of input sequences for which no output is specified. Call $I_u \subset \Sigma^*$ such set. Conversely, there exist a set of output sequences which can be produced only by unspecified input sequences. Call $O_u \subset \triangle^*$ such set. The problem of minimizing the number of states in CSFSMs can be solved in polynomial time [16]. For ISFSMs the problem is known to be NP-complete [17]. Algorithms for reducing such machines are proposed in [14, 15, 16].

Let $M' = (\Sigma', \triangle', Q', q'_0, \delta', \lambda')$ be an ISFSM and $\mathcal{P}_{M'}$ be the set of all possible completely specified implementations of M'. Thus, for each $p \in \mathcal{P}_{M'}$, every element of I_u and O_u is eventually associated to an element of $\triangle^*$ and Σ^*, respectively. Let us select an arbitrary sequence $s_\sigma \subset I_u$ and the corresponding output sequence $d_\sigma \in \triangle^*$. Let tuple $\sigma = \{s_\sigma, d_\sigma\}$, call it *IO signature.*

Consider first an active watermarking regime. The problem of synthesizing a watermark for an ISFSM M' is equivalent to that of finding a minimum sized machine M'', whose specified IO mapping has been augmented by an IO signature σ on specification of M'. It is also required that robustness constraint specified as $\overline{P_m}$ and $\overline{P_u}$. be satisfied. The problem is formulated as following

Problem 1. Minimize size of M'', s.t.

$$P_m \leq \overline{P_m} \ , \quad P_u \leq \overline{P_u} \ , \tag{1}$$

where $\overline{P_m}$ and $\overline{P_u}$ are constraints on the watermark robustness. Note that the size is measured in terms of added states and logic.

Problem 1 can be partitioned into two tasks. The first task consists of computing the size of IO signature σ so as to satisfy the constraints on the confidence. The second task is that of finding the actual IO signature so as to minimize the overhead of M''. The IO signature must be generated with some degree of randomness to ensure that, using the same algorithm one cannot generate an identical code. The randomized algorithm is controlled by key k. The key k is provided by the user to control the generation of the IO signature and of the sequence of states activated by the it. k is used to select from n best state sequences and IO signatures. In this case, the minimality of the overhead might not be guaranteed.

In case keeping the IO signature secret were not possible, then one of the following approaches could be used. The authentication of the generated IO signature can be achieved by registering the key of a specific design in a third party database, similarly as in copyright and trademark registration.

An alternative solution is that of explicitly creating an IO signature based on the method proposed in [11]. The user specifies a string which is converted into a number by standard one-way hash function like MD5. In this manner, one can guarantee that there will be no two identical IO signatures generated by two different strings and it is computationally intractable to obtain the string from the IO signature. Using this signature, one can find a state sequence that minimizes the overhead, even though an absolute minimum can not be guaranteed.

Synthesizing watermarks in CSFSMs requires first that the machine be translated onto a ISFSM. This can be accomplished by extending the input and/or output alphabets Σ and $\triangle$. The resulting machine is then handled by solving

Problem 1. Hence, the procedure can be seen as a preprocessing step to a general watermark synthesis step.

A passive watermarking scheme consists of generating signature σ from a given ISFSM without modifying the machine itself. The process consists of first minimizing the FSM using, for example, the techniques proposed in [14], thus synthesizing a CSFSM. Then, a subset of all the sections of the non-specified IO mapping are designated as a IO signature. Randomization of the signature, controlled by key k, used to select unspecified IO sequences. Hence, the probability of accidentally synthesizing the same watermark are bounded by the degrees of freedom of the algorithm and/or by its level of randomization.

At least two approaches exist to the generation of IO signature σ. The first involves the generation of new transition relations in the FSM's STG or STT, while the second calls for the augmentation of Σ, $\triangle$ or Q. All these modifications are likely to but do not necessarily increase the size of the machine.

Let $q' \in Q'$ denote a state in an ISFSM M' and let q'_0 be its reset state. Let $I_u^{(q')}$ be the set of all the input configurations in q' for which no next state is specified, call such configurations *free*. Define U' to be the set of all the states with incompletely specified transition relations, i.e. $U' = \{q' \in Q' \;\mid\; |I_u^{(q')}| > 0\}$. The total number of free input configurations n is bounded as follows

$$n \leq n_{max} = \sum_{q' \in Q'} |I_u^{(q')}|. \tag{2}$$

Every state $q' \in U'$ must necessarily be reachable $|I_u^{(q')}|$ times, using each time one of the remaining free input configurations in $I_u^{(q')}$. Suppose that a sequence x exists of all the visited states, call s the input sequence which forces x. The resulting output sequence d, of length n, will be one of $[2^{|\triangle|}]^n$ possible implementations. Hence, the odds that an identical sequence be produced by M is

$$P_u = \frac{1}{\left[2^{|\triangle|}\right]^n - 1}. \tag{3}$$

The second term of the denominator is given by the fact that one of such sequence will result from the given input sequence in the CSFSM in $\mathcal{P}_{M'}$. By setting $P_u \leq \overline{P_u}$ and solving (3) with respect to n one obtains

$$n \geq n_{min} = \frac{1}{|\triangle|} log_2 \left| 1 + \frac{1}{\overline{P_u}} \right|. \tag{4}$$

In some cases it is not possible to satisfy both (2) and (4) to meet specification (1), i.e. $n_{min} > n_{max}$. Hence, either (1) must be relaxed and/or n_{max} must be increased.

Suppose constraints (2) and (4) are satisfied, then an output sequence $d_\sigma \in \triangle^*$ and the states which can produce it must be selected. The required output is generated by an n-long sequence of states in U'. The sequence can be seen as a path $p_\sigma = (q'_0, u'_1, \ldots, u'_{n-1})$ covering a subset of the states in U', with or without repetition. It is assumed, but it is not necessary, that $q'_0 \in U'$. If this were not the case, a different first state, say $q''_0 \in U'$, could be selected for p_σ and input sequence s_σ would need to be augmented by an input sequence s such that $\delta'(q_0, s) = q''_0$. The generation of p_σ does not contribute to the probability of coincidence P_u, but it does determine the impact state minimization will have on the final machine. The second factor impacting the effectiveness of the optimization is the selection of input sequence s_σ.

For a given output sequence d_σ, input sequence s_σ is generated in two steps: selection of p_σ and derivation of s_σ. Sequence p_σ represents a path through n of the states in U' from the original STG. Every time a state u' is touched by the path, it looses one of its $|I_u^{(u')}|$ free input configurations. We propose to use an algorithm based on the Euler path search which can be targeted to minimize the number of visited states and/or to maximize the number of remaining free configurations per state.

As an illustration, consider the ISFSM example given earlier. For each state assume there exist three out of four free input configurations. Assume that $n = 2$, then two possible paths p_σ are shown in Figure 2 (a) and (b). In the example of Figure 2 (a) the number of inputs was unchanged, while in 2 (b) it was incremented by one. Consider the example of Figure 2 (a). Path p_σ, represented in bold, is selected by maximizing the number of remaining free configurations per state. Note that the path may begin in a state other than the reset state q_0. In this case, one must additionally find the input sequence leading to p_σ's initial state.

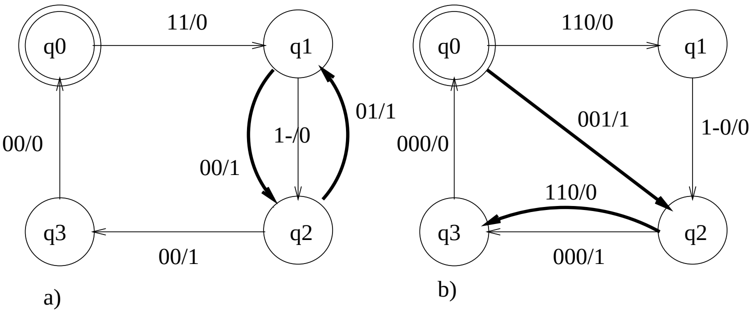


Fig. 2. Two possible paths p_σ for a given U': a) path based on minimum visited states criterion; b) path based on maximum remaining free configurations

Once p_σ for Figure 2 (a) has been selected, input sequence s_σ is derived from a path on a decision tree rooted in q_0 and whose leaves correspond to state u'_{n-1}. The solid bold line in Figure 3 represents p_σ, while the dotted line shows the path needed to reach p_σ's initial state. At each level i exactly $|I_u^{(u'_i)}| < |\Sigma|$

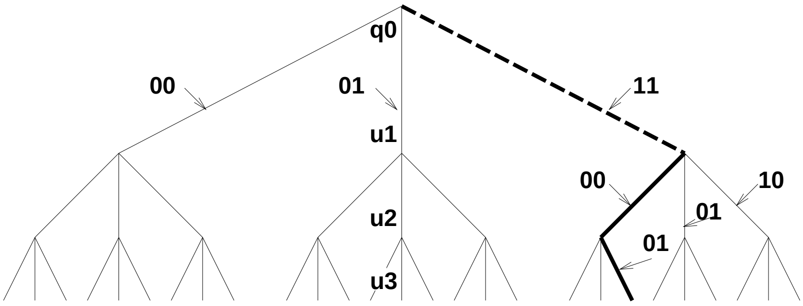


Fig. 3. Decision tree to compute s_σ

branches exist. Each branch represents the decision of using a certain free input configuration at a given state. There exists $\Pi_{i=1}^{n}|I_u^{(u'_i)}|$ possible paths connecting the root state q_0 to u'_{n-1}. One or more of these paths is associated with the smallest CSFSM $M \equiv M'$. The problem of finding such path is NP-complete since in best case the machine associated with one path must be synthesized, which in itself is an NP-complete problem. As an illustration, if the path represented in bold in Figure 3 is used for ISFSM M', the resulting IO signature is $\{s_\sigma, d_\sigma\} = \{(1,1,0,0,1,0);(0,0,1)\}$.

Several alternatives are proposed for the generation of the input sequence s_σ to minimize overhead. The first method consists of performing an exhaustive search of the decision tree. For each path a CSFSM is synthesized and the smallest machine is selected. The second method is a Monte Carlo approach, in which a set of input sequences are selected at random from all the feasible ones. The CSFSMs corresponding to such sequences are generated and the smallest one is selected. The third method is based on a branch-and-bound search. At each level of the tree an estimate is computed for the machine associated with each sub-tree underlying any decision. Such estimate is computed using a Monte Carlo approach. All the sub-trees with higher estimates are pruned, while the surviving trees are explored into the next level, i.e. the next state of p_σ. The search stops at the leaves. The complete algorithm for active watermarking in FSMs is described as follows:

1. if the FSM is CSFSM then augment Σ
2. compute the minimum size of s_σ, n_{min}, from $\overline{P_u}$
3. if $n_{min} > n_{max}$ then augment Σ or $\triangle$
4. using k, randomly generate new output sequence $d_\sigma \in \triangle^*$
5. compute path p_σ
6. compute input sequence s_σ

As a byproduct of Step 6 the FSM is synthesized. A passive watermarking scheme is applied to ISFSMs only. The method assumes that randomization can be introduced by the FSM synthesis. It consists of converting the original ISFSM onto a CSFSM using a given optimization criterion. Then, an IO signature is selected at random from all the possible ones available. The only way to synthesize a CSFSM from the original ISFSM which contains an identical IO signature is to use the same synthesis engine with an identical set of parameters and optimization criteria. Hence, P_u can be derived in this case as the inverse of all possible machines which can be generated from an ISFSM of a certain size and structure with the given engine.

Detecting a signature σ entails applying input sequence s_σ to the machine and observing the output sequence d. See Figure 4. If no tampering has occurred, then necessarily $d = d_\sigma$ and $P_m = 0$, i.e. no misses are possible. To properly analyze the effects of tampering, let us consider the following scenarios:

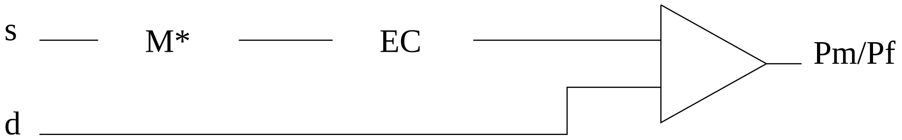


Fig. 4. Detection of signature under some tampering

1. specifications on the IO mapping of the original machine are known,
2. IO mapping of the original machine is not known but the STG of the modified machine is known,
3. no STG is known.

In case (1), infringement cannot be prevented, since the aggressor can resynthesize the FSM from specifications using techniques proposed, e.g. in [15].

In case (2) the aggressor may either: (a) modify state transition relations, i.e. changing the output or next state associated with a transition relation, or (b) apply the techniques proposed in this paper to watermark CSFSMs. In both cases, part or the totality of the watermark will be unchanged, but it may be corrupted locally. Tampering (a) may in fact result in a change in the functionality of the machine, it is therefore counterproductive. Tampering (b) will only result in literal swaps and deletions within pairs of reset states, similar to gene deletion within DNA sequences.

To combat tampering (2)b, we propose an approach based on the concept of *genome search*. Such approach was successfully used in topological watermarking [6, 7]. The method is essentially a selective pattern matching. It is assumed for simplicity that the output d_σ is a chain of sequences all rooted in a single reset state q_0. This restriction is however not necessary as multiple reset states can be used. Suppose the IO signature is

$$\begin{aligned} &p_\sigma = (q_0, q_2, q_1, q_0, q_3, q_4, q_0, q_1, q_2, q_0) \\ &\{s_\sigma, d_\sigma\} = \{\ (01, 01, 00, 10, 01, 00, 11, 10, 01); (0, 1, 1, 0, 1, 0, 1, 1, 1)\ \}\,. \end{aligned}$$

Suppose that tampering has removed or corrupted the median section of d_σ, i.e. $(0, 1, 0)$, then the sections of the IO signature which are still intact can be matched to σ using the `genome_search` algorithm described in detail in [6]. The algorithm returns an estimate of the probability that the design contains in fact watermark σ. Note that by construction, it is known when the reset state is reached. Hence, the boundary symbols or *operons* of each “gene” are known. Also note that if this or any other error correction algorithm is used, then our estimation of P_u is un upperbound on the true value, i.e. it is an optimistic estimate. It this case changes to the way P_u is estimated should be applied based on the details of the algorithm. An alternative method is that of using correction schemes such as CRC to detect and correct corrupted subsequences.

Finally, consider case (3), let us analyze the possible attempts to remove the watermark using netlist manipulations. Obviously, it is not possible to foresee all possible tampering techniques. Instead, we will analyze those that are more likely to be performed under following assumptions.

Assumption 1 *A netlist or a structural HDL description is available for tampering.*

Assumption 2 *All input and output pins are well documented and extra I/O pins (if any) used for watermarking are introduced as extra test pins and/or signal pins.*

In [11] it has been proven that generating a STG from a given netlist is an NP-Complete problem. For medium and large scale FSMs, it is unlikely that the STG can be obtained from its netlist. Therefore, if the netlist is obtained by reverse engineering, the aggressor has no other options but to perform one of the following modifications to remove or hide the watermark: (a) embed the FSM into a bigger one, (b) delete some of the circuitry related to the test inputs, (c) Add dummy I/O bits and/or shuffle the bit order using unknown mapping functions.

In scenario (a), the aggressor tries to hide the watermark under a wrap to mask the original IP from input/output probing. The watermark is still intact but it may not be easily observable, if at all possible. In this case, the detection technique proposed earlier cannot be exploited. However, simulation or on-chip measurements can be used to logically insulate the original IP from the wrap.

In scenario (b), by knowing that the watermark should be related to the extra test pins, the aggressor might try ro remove the registers and circuitry related to those inputs. In this case. the attempt would damage the original behavior because the IO signature is an integral part of the FSM. Therefore, this attempt shall not be successful.

In scenario (c), the aggressor adds new dummy input and/or output bits and dummy circuitry to the FSM. In this case IP forensic can use the following exhaustive method. Let us assume that there were n input bits and m output bits in the original watermarked FSM. Moreover assume that d_n and d_m extra bits have been added. Then, one needs to apply the input sequence to each possible subset of n bits of the $n + d_n$ inputs. The output is observed to reconstruct the correct sequence. Although it is time consuming, it is guaranteed that the IO mapping can be found exactly, since the watermark is intact.

3 Structural or Netlist Watermarks

In the following two sections it is useful to introduce a mathematical formalism which helps describing the watermarking algorithms being used. Let Σ^* be the set of all strings in a finite alphabet Σ, e.g. $\Sigma = \{0, 1\}$. Assume there exists a compact representation or signature for a given design at some abstraction level. Let $s \in \mathcal{S}$ be one of all possible physical implementations of the design, let σ_s be its signature. Define *signature mapping* $\mathcal{S} \rightarrow \Sigma^* : \mathcal{M}$ as the mapping of a subset of all the layout features onto a signature $\sigma_s = \mathcal{M}(s)$. Let us define $\mathcal{S} \rightarrow \mathcal{S} : \mathcal{F}$ as a mapping which transforms implementation s onto a new implementation $s' = \mathcal{F}(s)$.

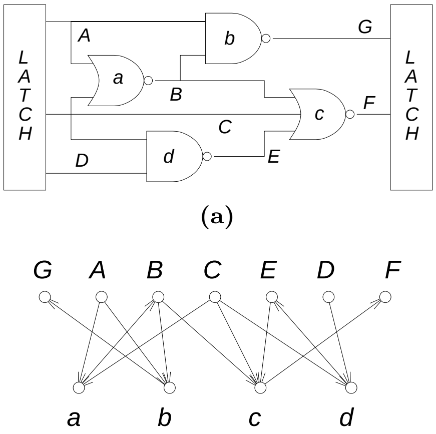
The structural level of abstraction is an intermediate representation of a circuit. Generally, after a compilation phase behavioral representations can be converted into a physical implementation by constructing the symbolic connectivity map of all the necessary electrical devices. Such map is usually represented in terms of a schematic or a netlist.

For instance, a schematic or netlist can be represented through a graph $G(N, E)$. The nodes of the graph correspond to general blocks, or single devices, as well as nets. The (directed) edges define connectivity. Let us define $\mathcal{H}$ as the set of all blocks in Ω. Let $\mathcal{N}$ be the set of all nets, with $E = \mathcal{H} \cup \mathcal{N}$. Let $\mathcal{O}_n$ be the set of edges in net $n \in \mathcal{N}$ which are connected to an output. The set of edges leading to an input is called $\mathcal{I}_n$, while the set of edges connected to a high-impedance pin or pass transistor gate is called $\mathcal{P}_n$. For simplicity, we assume that exactly one edge can be connected to an output, i.e. $|\mathcal{O}_n| = 1$, this condition is however not necessary. The pin number $|n|$ and the type and port of the gates connected by n are necessary but not sufficient properties to uniquely identify the net. A set of constraints on sets $\mathcal{O}_n$, $\mathcal{I}_n$ and $\mathcal{P}_n$ for each net, can be imposed so as to make these properties define the net uniquely, to all practical purposes.

Consider the gate-level circuit in Figure 5(a) and the corresponding connectivity graph of Figure 5(b). Let $\mathcal{N}' \subseteq \mathcal{N} = \{n \in \mathcal{N} : |n| \text{ is prime}\}$. Next, impose the following constraints on each net $n' \in \mathcal{N}'$:

$$\begin{array}{l}\mathcal{O}(n') = \{\text{gates of type } \omega_{\mathcal{O}}(|n'|)\};\\ \mathcal{I}(n') = \{\text{gates of type } \omega_{\mathcal{I}}(|n'|)\};\ |\mathcal{I}(n')| = \ell_{\mathcal{I}}(|n'|);\\ \mathcal{P}(n') = \{\text{gates of type } \omega_{\mathcal{P}}(|n'|)\};\ |\mathcal{P}(n')| = \ell_{\mathcal{P}}(|n'|),\end{array} \tag{5}$$

where $\omega_{\mathcal{O}}(), \omega_{\mathcal{I}}(), \omega_{\mathcal{P}}(), \ell_{\mathcal{I}}()$ and $\ell_{\mathcal{P}}()$ are net size-dependent parameters, generated using, for example, a parametrized pseudorandom sequence determined by key k.



(a)

(b)

Fig. 5. *(a) Gate-level circuit; (b) Connectivity graph*

It is trivial to infer that an arbitrary signature can be implanted in the structure of the netlist. Such signature can be at the hart of the watermark, which, in this case, is known as a *structural watermark*. In the case of Figure 5, Equations (5), written compactly, form the signature for net $\mathcal{C}$ as: $\sigma_C(\mathcal{C}) = (latch; a, d, c, 3; -, 0)$.

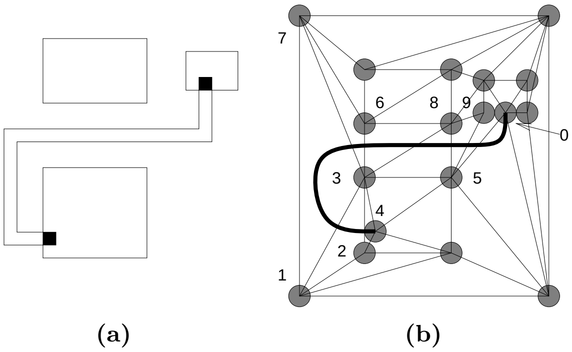
4 Watermarking Physical Implementations

Let us describe the particular mapping used in this paper to generate design signatures from given physical implementationof a circuit. The same technique can be used to implant an arbitrary signature in terms of an extraneous active or inactive circuit. Let us assume that the granularity of the original circuit is given. As a result, the set of fundamental components, such as transistors or standard cells, is determined. Call such components *atomic blocks* and Ω their set. In s, every component $\omega \in \Omega$ may have multiple instantiations.

A layout implementation defines a set of all relative positions and orientations of every component instantiation in the circuit. Interconnect can be represented in a similar fashion where components are replaced by pins, Steiner points, and bends. A composition, containing the details of all relative positions and orientations, is called *topology*. Let us now use the layout's atomic blocks, pins, Steiner points, and interconnect bends, which are in turn represented by a set of primitives called *bubbles*, as proposed in [18]. A bubble is a point associated with a given layer. Let B be the set of all bubbles in the design. Every atomic block is mapped onto m distinct bubbles according to a specific mapping $\Omega \rightarrow B : \mathcal{B}$, where m is a finite natural number. For simplicity, but without loss of generality, suppose that m is constant over Ω. Note that $|B|$ grows linearly with the number of atomic blocks and pins.

Paths can be represented by a continuous curve of finite length which begins and ends in a bubble. Such curve is known as *rough routing* [19]. The design rules of a given technology can be seen as minimum spacing constraints between the perimeters of bubbles and paths. Alternatively, after proper scaling of the design rules, one can consider bubbles as points, and paths as curves of zero-thickness. For simplicity we have adopted this convention. Let *topological routing* be an equivalence class of rough routings connecting its pins. Two rough routings of a wire are equivalent when one can be obtained from the other by continuous deformation with no violations of any of the scaled design rules. Assume that every pair of bubbles is connected by an edge, then if a topological routing crosses such an edge, it is said to *intersect topologically* the edge.

If every region in the layout is partitioned in simply connected regions, each containing no bubbles, then such regions are called *simple regions*. Figure 6(a) shows an interconnect and some obstacles, while Figure 6(b) depicts the corresponding partition into simple regions. The rough routing connecting bubble 4 to 0 can be represented in terms of the sequence of all topological intersections. In this case such a sequence is: $\sigma = (\overline{23},\overline{13},\overline{37},\overline{36},\overline{38},\overline{58},\overline{59},\overline{50})$. Note that symbol $\overline{X_i\ X_j}$ represents the topological intersection of the rough routing with the edge spanned by bubbles X_i and X_j. Define E_ℓ as the set of all simple regions in a given layer ℓ and a *planar subset* $T_\ell \subset E_\ell$ as one in which distinct edges do not intersect or they intersect at only one of the vertices. In addition, if T_ℓ has a convex boundary or *convex hull*, it is said to be maximally planar. Under these conditions, T_ℓ is called *triangulation* [20]. Let us now assume that an arbitrary triangulation T_ℓ is in place for each layer. Let B_ℓ be the set of all the bubbles



(a) **(b)**

Fig. 6. *Bubbles and rough routings*

associated with T_ℓ. For convenience, although not needed, let us set four bubbles at the extremities of the union of all the layers, so as to encompass every layer.

Sequence σ is a non-unique representation of all the rough routings associated with the class of this topological routing. Hence, to make such representation resilient to minor modifications, it is necessary to convert it onto a canonical form. This is done simply removing adjacent identical edges, which form so-called *loops*. The unique canonical form of an arbitrary topological routing τ is called *topological signature* σ_τ. The complexity of loop removal is higher when it involves a large number of rough routings. The process in this case must be recursively performed.

Triangulations are not unique. However if the method used to obtain a certain triangulation is an invariant, then the signature is also invariant for a certain design. Figure 7(a) for example shows a simple layout based on standard cells

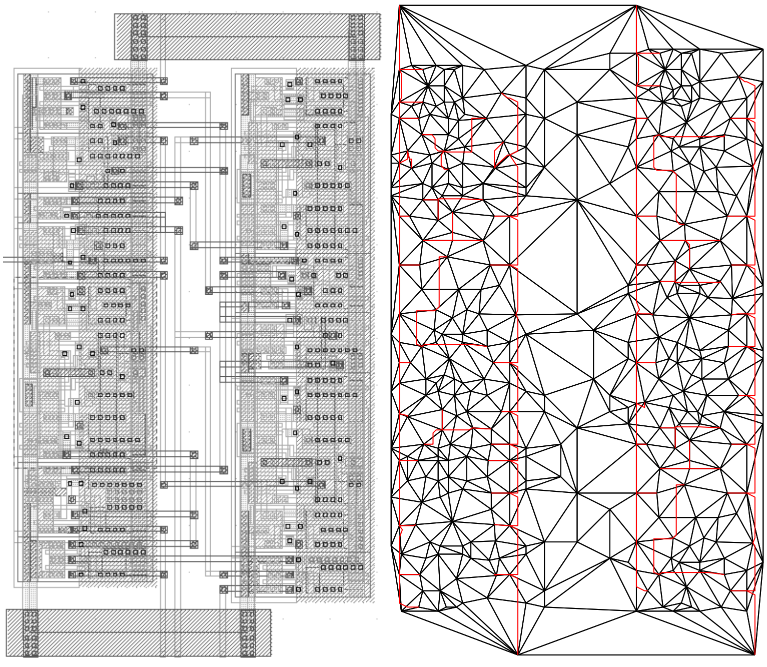


Fig. 7. *(a) Layout; (b) Associated triangulation*

organized in two rows with the corresponding interconnect. Figure 7(b) shows a possible triangulation of the associated topology. The computational scheme and circuit topology determine the final result [20].

Recall that the uniqueness of a signature is defined by probability P_u, its robustness by P_m. Topological intersections are unique for a given design and triangulation, while a triangulation is determined by the utilized algorithm and by set B. For each layer the number of possible triangulations grows factorially as $(|B_\ell| - 1)!/3!$, hence it is reasonable to choose a layer ℓ^* which maximizes $|B_\ell|$ over all layers. By a conservative estimate, N_T, the total number of possible triangulations over all layers, is then $N_T \geq (|B_{\ell^*}| - 1)!/3!$. Suppose now that all N_{ℓ^*} topological routings in ℓ^* consist of N_i i-terminal nets, $i = 2, \ldots, N_{max}$. Then, all N_{ℓ^*} topological routings can be represented in terms of N' two-terminal sub-routings, with $N' = \sum_{i=2}^{N_{max}} N_i \ (i-1)$. As a consequence, the number of possible topological signatures can be computed as $N_\sigma \geq N_T \ \binom{N'}{2}$, hence the estimate of P_u becomes $\overline{P_u} \leq \frac{1}{N_\sigma}$. For example, suppose that for a given design $|B_{\ell^*}| = 20$, $N_{\ell^*} = 10$, $N_2 = 3, N_3 = 5, N_4 = 2$. Then, $N_\sigma \geq (20-1)!171/3! = 3.5 \times 10^{18}$, hence $\overline{P_u} \leq 2.9 \times 10^{-19}$.

In the absence of tampering $P_m = 0$, i.e. the signature extracted from a topology matches 100% with the one which is registered in the signature bank. If tampering has occurred, it needs to be modeled in order to properly estimate its effects on P_m. Let us consider the following tampering attempts: (1) routing modification, (2) atomic block modification, and (3) atomic block move and/or addition/deletion. Attempt (1) does not change triangulation, however it may cause changes in the signature. Such changes are of three basic types: symbol addition, deletion and swap. More than one symbol may be involved in the change at any time, however, when this occurs, the change can be modeled in terms of a composition of simple symbol modifications. Attempts (2) and (3) may change the triangulation. However, their effects can be modeled in terms of simple symbol operations.

Define P_r as the probability that a symbol change occurs. Then, the probability that a signature of size t mutates is $P_t = \sum_{j=1}^{t} \binom{|B|}{j} \ [P_r]^j \times [(1-P_r)]^{|B|-j}$. Hence, for example, if $t = 1$ and $P_r = 10^{-5}$, then $P_m \leq 9 \times 10^{-6}$.

Signature detection consists of the following phases

1. bubble extraction
2. transformation inference
3. bubble matching
4. triangulation
5. signature computation

The initial layout is flattened and all its layers are extracted and deconstructed into polygons or basic standard cells. Using standard slicing techniques [21], the layout is partitioned in rectilinear areas encompassing exactly one atomic block. The complexity of this operation is $O(|\Omega| \ log|\Omega|)$ where $|\Omega|$ is the number of objects in the layout. Using mapping $\mathcal{B}$, the design is entirely converted into a bubble-based representation in $O(|\Omega|)$ time (phase 1).

In order to detect the presence of blocks with known signatures embedded in the design, one has to infer the most likely orientation of every candidate block. This operation is performed by matching complex interconnect patterns present in both the host and the embedded design. Consider the designs of Figure 8.

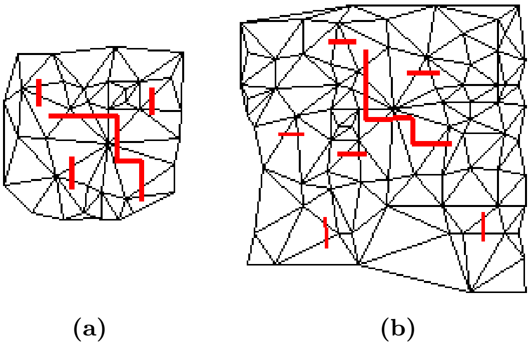


Fig. 8. *Transformation inference: (a) embedded, (b) host design*

Suppose the interconnects shown in shaded lines are to be used to determine the orientation of the embedded circuit within the host. Let us first catalog all the interconnects present in both layouts in order of size (equal to the number of interconnect segments) in $O(n \log n)$ time. Then, for each pair of interconnects of identical size, a transformation $(\triangle x, \triangle y, \theta, s_x, s_y)$ is derived which maximizes the number of points that can be transformed from the embedded to the host design. Note that s_x, s_y represent a possible scaling operation. Deriving $(\triangle x, \triangle y, \theta, s_x, s_y)$ requires the solution of a system of eight linear equations for each pair of candidate interconnects in the worst case. Then, the most frequently occurring transformation is selected. The solution time of each system of equations is constant, the worst case time complexity is therefore quadratic in the number of interconnects of identical size. (phase 2).

Next, the bubble representation of the host needs to be matched with that of the transformed embedded design. This procedure is accomplished by superimposing the designs and by assigning every bubble in the host to exactly one in the embedded design which minimizes the Euclidean distance. The search is initially performed within a zero range, which is augmented multiple times by a unit length until a neighbor is found. Figure 9 shows the range search process (phase 3).

Finally, using optimal algorithms, a Delauney triangulation is computed in $O(|B|\ log|B|)$ time for both designs [20, p. 241] (phase 4). The line segment intersection algorithm is used for the computation of the edges being intersected by each topological routing. The complexity of this operation is again $O(|B|\ log|B|)$

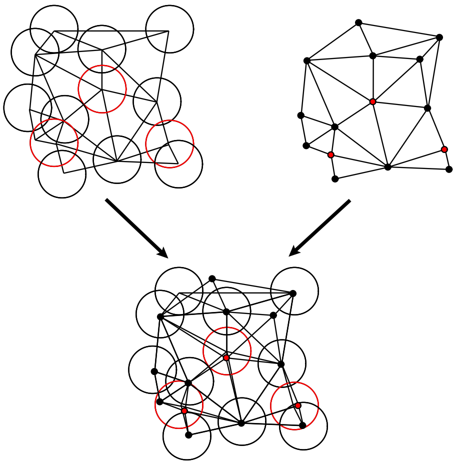


Fig. 9. *Principle of range search*

[20, p. 285]. The signature is derived from this information in a straightforward way (phase 5). In summary, the complexity of entire signature detection process is $O(n\ log\ n)$, where n is the number of atomic blocks, pins and Steiner points in the topology.

5 Examples

5.1 HDL Level

In our experiments we have used FSMs from the IWLS93 benchmark set. The tools were implemented in C/C++ and run under UNIX and Linux operating systems. Watermarking was performed on ISFSMs as well as CSFSMs. Constraint $\overline{P_u}$ was selected so as to require, in some cases, expansion of Σ and/or $\triangle$. The increase in the number of states $|Q|$ and input/output bits $|\Sigma|$ is expressed by the area estimates. The estimates are based on technology mapping obtained with SIS[22] using the MSU script. Table 1 lists all relevant experimental data and specifications on the robustness of the watermark. For the FSM minimization stage in the algorithm the tools STAMINA and NOVA[14] were used. The area results are based on the actual circuit implementation after technology mapping obtained via SISand relate to the number of gates.

As expected, larger FSMs require less overhead for comparable robustness. Note, as shown in benchmark `ex1`, that overhead can be traded for smaller values of $\overline{P_u}$. These overhead results are comparable to the ones obtained in [11]. The overhead of benchmark `s27` was extremely high due to the increase of the output alphabet. Such expansion was however necessary to boost the watermark's confidence.

circuit	# states	# I/O	# I/O chg.	n_{min}	orig. FSM		Monte Carlo		$\overline{P_u}$	Overhead
					area	CPU	area	CPU		
s27	6	4/1	1/3	9	632	0s	1.53k	0.1s	1.4×10^{-11}	143%
bbara	10	4/2	1/1	10	1.16k	0.1s	2.01k	0.1s	9.3×10^{-10}	73%
dk14	7	3/5	1/0	7	1.48k	0.1s	1.84k	0.1s	2.9×10^{-11}	24%
styr	30	9/10	1/0	4	8.6k	0.1s	10.69k	0.1s	9.1×10^{-13}	22%
bbsse	16	7/7	1/0	10	2.28k	0s	2.62k	0.1s	2.9×10^{-11}	6.3%
cse	16	7/7	1/0	5	3.84k	0.1s	4.08k	0.1s	2.9×10^{-11}	6%
sse	16	7/7	0/0	3	2.29k	0s	2.43k	0.1s	4.7×10^{-7}	5.9%
ex1	20	9/19	0/0	4	5.37k	0.1s	5.55k	0.1s	1.3×10^{-23}	3.2%
ex1	20	9/19	0/0	2	5.37k	0.1s	5.40k	0.1s	3.6×10^{-12}	0.6%
viterbi	68	15/59	1/0	2	13.49k	1.5s	13.61k	15s	3.0×10^{-36}	0.8%
dec	56	16/23	1/0	2	14.75k	0.5s	14.78k	5s	1.4×10^{-14}	0.2%
scf	121	27/56	0/0	2	20.97k	3.4s	21.02k	34s	1.9×10^{-34}	0.2%

Table 1. *IWLS 93 FSM benchmarks. The number of States and the number of I/O pins refer to the original FSM, while I/O chg. refers to the modified FSM. Overhead is the extra area of the modified FSM*

Exhaustive search could be performed only in `sse` due to the extreme computational complexity of the method. The CPU time in this case was 1.0 second for an area of 2.33k gates. For the other circuits an estimate of a lowerbound of the time required by the search can be computed. Such time estimates are derived multiplying the time required by one minimization with the minimum number of free configurations, i.e. $2^{|I_u^{(min)}|\, n_{min}}$, where $|I_u^{(min)}| = \underset{q' \in U'}{min} |I_u^{(q')}|$.

In the Monte Carlo approach a maximum of ten input sequences s_σ was explored. Alternatively one could select such upperbound based on some estimate or measurement of the standard deviation of the minimized machine's size.

5.2 Structural and Physical Implementation Levels

A complete pass in a typical design flow was simulated in order to verify the suitability of the approach. The tools utilized in the flow were implemented in C/C++ running under UNIX/LINUX operating systems. All CPU times are referred to a Sun UltraSparc 2 with 256MB of memory. The experiments were based on a set of MCNC 86 and ISCAS 85/89 benchmarks. Each circuit was synthesized and mapped to a SCMOS technology using Sis[23]. Place&route was performed by TimberWolfSC-4.1[24].

To simulate the registration phase, a signature was generated for each benchmark. Then, small modifications were introduced in every benchmark to check whether the signature was resilient to "official" Engineering Change Orders (ECOs) and scaling. Later, a variable number of random non signature-invariant mappings $\mathcal{F}$ were performed on the benchmark's layout so as to maximize the potential damage to the circuit. $\mathcal{F}$ introduced changes on atomic blocks, pins, Steiner points, and nets, uniformly distributed over the entire circuit. Three types of modifications were implemented: (1) translation/rotation, (2) swap, and (3) stretch, aimed at simulating illegal tampering. The signatures associated to the modified designs were compared with the original ones. Finally, the benchmarks were entirely redesigned and the signatures were again compared to the original ones, thus estimating the event that a design could be mistakenly detected even when a "legal" redesign had taken place.

Table 2 reports circuit data, such as device, IO pin, and net count. The signature matching rates are given for several modification densities, simulating an ECO applied to the circuit. The signature was constructed with a minimum net size n_n of 2, 3, 4 or 10 terminals, while no net size upperbound N_n was used. As expected, small ECOs generally resulted in perfect matching, while re-designs resulted in very low matching rates. Moreover, small circuits were less robust to tampering than large ones, due to the lower number of degrees of freedom available to their design.

circ.	n_n/	dev./	ECO density		re-	CPU
	N_n	IO/nets	5 %	10 %	des.	[s]
s27	2/∞	69/5/96	99.05	96.68	8.24	76.9
s27	3/∞		100	100	7.80	53.0
s27	10/∞		100	100	4.28	43.0
s444	2/∞	709/9/932	100	93.0	10^{-6}	1598
s444	10/∞		100	93.5	10^{-6}	1087
s832	4/∞	1686/37/2127	100	-	10^{-6}	1950
s832	10/∞		100	-	10^{-6}	1620
s1196	10/∞	2105/28/2682	100	96.0	10^{-6}	2383

Table 2. *Signature matching with ECOs and re-design*

For the detection phase a large benchmark was selected as the host design. Small benchmarks were embedded, at random locations, in the host. The detection algorithm was run on this example to extract the original signature of the host as well as that of the embedded designs. In various experiments the embedded circuits made up 1% to 10% of the entire host. Finally, tampered circuits were embedded in the host to verify the robustness of the approach in the presence of multiple levels of tampering. Table 3 summarizes the results of the detection experiment. Despite the presence of embedded circuits, the host still maintained high signature matching (rows 3-6 in Table 3). The recognition algorithm performed well in identifying both untampered embedded circuits and heavily tampered ones.

embedded	host	n_n/	ECO density		CPU
circ.	circ.	N_n	0 %	10 %	[s]
s27	s1196	$1/\infty$	73.8	73.8	218
s27	s1196	$2/\infty$	72.7	72.7	218
s27	s1196	$5/\infty$	72.7	72.7	218
s1196	-	$1/\infty$	100.0	99.2	1241
s1196	-	$2/\infty$	100.0	99.0	1241
s1196	-	$5/\infty$	100.0	98.6	1241

Table 3. *Signature matching with embedded circuits*

6 Conclusions

Protecting copyrights of intellectual property providers and integrators has become a serious problem. It arises from a recent industry shift in which electronic circuits are readily available in a form of virtual blocks at any abstraction levels, thus allowing for abuses and theft. Several methods have been described to generate watermarks at various levels of hierarchy during the electronic design flow. Modifications to the flow have been described to integrated watermarks at all abstraction levels. Ways of effectively detecting the presence of a watermark have been suggested so as to minimize the disruption of product cycles while allowing forensic analysis of large numbers of suspected circuits. If supported by an enforcement infrastructure at the place of fabrication, the described methods are effective in detecting and tracing intellectual property infringement *before* fabrication, thus minimizing potential litigation.

References

1. M. D. Swanson, B. Zhu, and A. H. Tewfik. Transparent robust image watermarking. In *Proc. IEEE International Conference on Image Processing*, volume 3, pages 211–214, September 1996.
2. L. Boney, A. H. Tewfik, and K. N. Hamdy. Digital watermarks for audio signals. In *Proc. IEEE International Conference on Multimedia Computing and Systems*, pages 473–480, June 1996.
3. J. Lach, W. H. Mangione-Smith, and M. Potkonjak. Fpga fingerprinting techniques for protecting intellectual property. In *Proc. IEEE Custom Integrated Circuit Conference*, pages 299–302, May 1998.
4. J. Lach, W. H. Mangione-Smith, and M. Potkonjak. Robust fpga intellectual property protection through multiple small watermarks. In *Proc. IEEE/ACM Design Automation Conference*, pages 831–836, June 1999.
5. J. Lach, W. H. Mangione-Smith, and M. Potkonjak. Signature hiding techniques for fpga intellectual property protection. In *Proc. IEEE International Conference on Computer Aided Design*, pages 194–198, November 1998.
6. E. Charbon. Hierarchical watermarking in IC design. In *Proc. IEEE Custom Integrated Circuit Conference*, pages 295–298, May 1998.

7. E. Charbon and I. Torunoglu. Watermarking layout topologies. In *Proc. IEEE Asia South-Pacific Design Automation Conference*, pages 213–216, January 1999.
8. A. Kahng, J. Lach, W. H. Mangione-Smith, S. Mantik, I. L. Markov, M. Potkonjak, P. Tucker, H. Wang, and G. Wolfe. Watermarking techniques for intellectual property protection. In *Proc. IEEE/ACM Design Automation Conference*, pages 776–781, June 1998.
9. D. Kirovski, Y. Y. Hwang, M. Potkonjak, and J. Cong. Intellectual property protection by watermarking combinational logic synthesis solutions. In *Proc. IEEE International Conference on Computer Aided Design*, pages 194–198, November 1998.
10. I. Torunoglu and E. Charbon. Watermarking-based copyright protection of sequential functions. In *Proc. IEEE Custom Integrated Circuit Conference*, pages 35–38, May 1999.
11. A. L. Oliveira. Robust techniques for watermarking sequential circuit designs. In *Proc. IEEE/ACM Design Automation Conference*, pages 837–842, June 1999.
12. E. Charbon and I. Torunoglu. Copyright protection of designs based on multi source IPs. In *Proc. IEEE International Conference on Computer Aided Design*, pages 591–595, November 1999.
13. I. Torunoglu and E. Charbon. Watermarking-based copyright protection of sequential functions. *IEEE Journal of Solid State Circuits*, SC-35(3):434–440, 2000.
14. T.Villa, T.Kam, R.Brayton, and A.Sangiovanni-Vincentelli. *Synthesis of Finite State Machines: Logic Optimization.* Kluwer Academic Publ., Boston, MA, 1997.
15. J. M. Pena and A. L. Oliveira. A new algorithm for the reduction of incompletely specified finite state machines. In *Proc. IEEE International Conference on Computer Aided Design*, pages 482–489, November 1998.
16. G. De Micheli. *Synthesis and Optimization of Digital Circuits.* McGraw-Hill, 1994.
17. C. F. Pfleeger. State reduction in completely specified finite state machines. *IEEE Trans. on Computers*, C-22:1099–1102, 1973.
18. T. Whitney. *Hierarchical Composition of VLSI Circuits.* PhD thesis, California Institute of Technology, 1985.
19. J. Valainis, S. Kaptanoglu, E. Liu, and R. Suaya. Two-dimensional ic layout compaction based on topological design rule checking. *IEEE Trans. on Computer Aided Design*, CAD-9(3):260–275, March 1990.
20. F. P. Preparata and M. I. Shamos. *Computational Geometry. An Introduction.* Springer, second edition, 1988.
21. R. H. J. M. Otten. Automatic floorplan design. In *Proc. IEEE/ACM Design Automation Conference*, pages 261–267, June 1982.
22. E. M. Sentovich, K. J. Singh, C. Moon, H. Savoj, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Sequential circuit design using synthesis and optimization. In *Proc. IEEE International Conference on Computer Design*, pages 328–333, October 1992.
23. E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Sis: A system for sequential circuit synthesis. Memorandum UCB/ERL M92/41, UCB, Univ. of California, Berkeley, CA 94720, May 1992.
24. C. Sechen and A. L. Sangiovanni-Vincentelli. Timberwolf3.2: A new standard cell placement and global routing package. In *Proc. IEEE/ACM Design Automation Conference*, pages 432–439, 1986.