Robust Techniques For Watermarking Sequential Circuit Designs

Arlindo L. Oliveira
IST-INESC / CEL
Rua Alves Redol, 9
1000 Lisboa, Portugal
aml@inesc.pt

Abstract

We present a methodology for the watermarking of synchronous sequential circuits that makes it possible to identify the authorship of designs by imposing a digital watermark on the state transition graph of the circuit. The methodology is applicable to sequential designs that are made available as firm Intellectual Property (IP), the designation commonly used to characterize designs specified as structural descriptions or circuit netlists.

The watermarking is obtained by manipulating the state transition graph of the design in such a way as to make it exhibit a chosen property that is extremely rare in non-watermarked circuits, while, at the same time, not changing the functionality of the circuit. This manipulation is performed without ever actually computing this graph in either implicit or explicit form. We present both theoretical and experimental results that show that the watermarking can be created and verified efficiently.

1 Introduction and related work

Watermarking is a technique traditionally used to securely identify the authenticity of the source of official documents, usually in paper format. The name comes from the original technique that used semi-transparent marks made on paper.

Recently, the application of similar techniques to protect and identify documents in other formats has raised considerable interest. In particular, digital watermarking has been applied to the protection of intellectual property in digital form [1]. Digital watermarking embeds digital information in a piece of intellectual property, in such a way that it is very hard to remove and, in general, also very hard to detect. The hidden information can be anything that uniquely identifies the author or proprietary of the piece of intellectual property and that is undetectable to the human perception. If necessary, the digital watermark can be used in court to prove the ownership of the piece of IP. More commonly, the presence (or potential presence) of a digital watermark will discourage unauthorized use of the intellectual property, thus avoiding the need for legal action altogether.

In the context of digital systems design, the interest in watermarking stems from the fact that, increasingly, reuse-based design methodologies offer the promise of increased productivity and reduced time to market. In this work, we are concerned with the protection of intellectual property for digital hardware designs.

Hardware made available as intellectual property may be described in a variety of forms. From behavioral descriptions in hardware description languages (HDL) to actual layouts, the problem of protecting IP from being used in inappropriate ways is relevant to both the IP producer and the computer aided design (CAD) companies that develop the integration software.

Traditionally, IP has been classified as hard, firm or soft, according to the degrees of freedom left to the user to manipulate it. Hard IP, available, for instance, in the form of a partially (or totally) routed layout, cannot be modified by the user, and should be used as is. Soft IP, on the other hand, needs to be processed and can be mapped to a variety of supports. Firm IP is used to represent designs that are made available at an intermediate level of abstraction, such as netlists or structural descriptions.

A variety of techniques has been proposed for watermarking different steps of the design process [3]. In particular, algorithms have been proposed for watermarking solutions of general purpose optimization problems, for combinational logic synthesis solutions [7], for FPGA mappings of digital circuits [8] and to identify the results of the final layout design stage [3]. Recently, a watermarking procedure for finite state machines for which the state transition graph can be extracted has also been proposed (STG) [10].

Although these results are important they do not allow a designer to achieve the objective of identifying the origin of a given complex digital design, made available in the form of firm IP. For these reasons, the work we describe here aims at protecting the design itself by encoding a digital watermark on the STG of the circuit.

2 Watermarking by state transition graph manipulation

The basic idea is to change the STG in such a way that a specific topological property is present in the sequence of states traversed by a sequence of inputs that corresponds to a given signature. If this property is chosen in the right way, it will rarely be present in designs that have been obtained independently, but will be present in any design that is a copy of the original one, even if this design is manipulated and changed in a variety of ways. For the method to be efficient and robust, the change in the STG has to be accomplished without actually storing the STG, either in explicit or implicit form.

We will assume that the sequential design in question represents a fully synchronous design, and that the specification of its functionality, from an input/output perspective is publicly available. We will also consider that the set of inputs and outputs is well identified and that each input is known by a specific name, described in a publicly available data sheet. We further assume that a specific ordering is used on the inputs and that the design is specified in the form of a structural description (e.g., a netlist).

The typical user will start from such a description, map it, if needed, to a specific technology, perform retiming and other logic level optimizations, and use it as part of more complex designs. The

DAC 99, New Orleans, Louisiana

difficulty arises when a user that has access to the description of this piece of IP wishes to use it in ways that are not compatible with the existing agreement between him and the IP provider.

In this case, the user may decide to perform a variety of changes to the IP description that may make it difficult to identify the origin of the design. Straightforward ways to change such a design include signal renaming, re-synthesis of the combinational logic, re-encoding of the states, retiming, redundancy removal or accrual, etc. If the design is not marked in some way that makes it easy to identify, it may be impossible to prove that it was actually stolen. In fact, equivalent functionality is not proof of wrongdoing, as the input/output specifications are public and a re-design from scratch is always possible.

2.1 Creating the watermark

The basic idea underlying the proposed method can be described in a simple way. The IP rightful owner starts by defining an arbitrary long string that clearly describes her ownership rights. For example, she may decide to mark the design with the message **"This design is the property of the Regents of the University of California"**. After encrypting this message with her private key of a known public key cryptosystem, she uses a one-way hash function, such as MD5, to obtain a compact signature of this arbitrarily long sentence. In this particular case, MD5 will produce a 128 bits message digest that is hard to invert, i.e., it is computationally infeasible to find another message that hashes to that same value.

She then breaks this sequence of 128 bits into a sequence of input combinations. For example, if the design has 16 inputs, the sequence of 128 bits defines a unique sequence of 8 input combinations.

She then proceeds to change the STG in such a way that the sequence of states reached by this sequence of inputs exhibits a specific property, that is rare in non-modified STGs. This property is purely topological and does not depend on the specific encoding used for the states, the number of registers or the details of the combinational logic.

If, later on, she wishes to prove in court that someone stole her piece of IP, she only has to show that the sequence of 128 bits obtained from her message defines a path in the STG such that the set of traversed states exhibits that specific property.

3 Watermarking state transitions graphs

3.1 Definitions

This section introduces some general definitions that will be used throughout the paper.

Definition 1 *A Mealy type finite state machine (FSM) is a tuple* $M = \left(\Sigma, \Delta, Q, q_0, \delta, \lambda\right)$ *where* $\Sigma \neq \emptyset$ *is a finite set of input symbols,* $\Delta \neq 0$ *is a finite set of output symbols,* $Q \neq \emptyset$ *is a finite set of states,* $q_0 \in Q$ *is the initial "reset" state,* $\delta(q, a) : Q \times \Sigma \rightarrow Q$ *is the transition function, and* $\lambda(q, a) : Q \times \Sigma \rightarrow \Delta$ *is the output function.*

We will assume that $Q = \{q_0, q_1, \ldots\}$ and will use will use $q, r, v \in Q$ to denote a particular state, $a \in \Sigma$ a particular input symbol and $b \in \Delta$ a particular output symbol. For finite state machines with multiple binary inputs, a^k, $1 \leq k \leq n$ will represent the value of the k-th binary input variable and q^k the value of the k-th state variable. In general, for multibit variables like a state or an input value, we will use superscripted variables to denote the value of a particular bit. For example, given an FSM and a specific encoding of its states, $\delta^j(q, a)$ represents the j-th bit of the next state obtained with input a in state q. Additionally, we will use x^i, y^i, s^i and t^i to denote the i-th input variable, output variable, state variable and next state variable, respectively.

We will use BDDs [2] as a data structure to represent implicitly the transition and output functions of the finite state machine under study, a method that is well known in the logic verification literature and first proposed in the seminal work of Coudert et al. [4].

Given a set $S = \{s^1, \ldots, s^z\}$ of present-state variables, a set $X = \{x^1, \ldots, x^n\}$ of primary inputs and a set $T = \{t^1, \ldots, t^z\}$ of next-state variables, we define the transition relation $\mathcal{T}(S, X, T)$ as follows:

Definition 2 *The transition relation of a finite state machine* $M = \left(\Sigma, \Delta, Q, q_0, \delta, \lambda\right)$ *defined over* S, X *and* T *is:*

$$\mathcal{T}(S, X, T) = \prod_{i=1}^{i=z} (t^i \equiv \delta^i(S, X)) \tag{1}$$

It is well known that, given a a transition relation, it is possible to compute the pre-image of a set of states C defined by its characteristic function using the expressions:

$$\mathrm{PImg}(C) = P(S) = \exists X \, \exists T \, C(T) \mathcal{T}(S, X, T) \tag{2}$$

3.2 Modification imposed on the structure of the STG

Consider the original STG for the design. The sequence of input combinations $a_1, \ldots, a_m$ will traverse a sequence of (not necessarily distinct) states, $q_{t_1}, \ldots, q_{t_m}$, starting at the reset state, $q_0 = q_{t_0}$. Let $r_0 = q_0$ and the sequence of STG edges traversed by this sequence be $e_1, \ldots, e_m$.

The particular STG modification we propose in this section, and that will be used to evaluate experimentally the method, is obtained by performing a number of changes in the state transition graph. For each time step i, $1 \leq i \leq m$, we will create a state r_i. Let $R = \{r_1, \ldots, r_m\}$. The modified STG is created by performing the following operations:

1. Duplicate all the states and transitions in the STG, creating a state v_i for each state $q_i \in Q$. Let $V = \{v_0, \ldots, v_{|Q-1|}\}$.

2. For $1 \leq i \leq m$ create state r_i by duplicating state q_{t_i} and all its outgoing edges. State q_{t_i} is left unmodified.

3. For each value of i, $1 \leq i \leq m$, state q_{t_i} has as one of its incoming edges, e_i. This edge originates at state r_{i-1}. Make this edge point to state r_i instead. Edge e_i now originates in state r_{i-1} and terminates in r_i.

4. Replace every outgoing transition from state r_m to a state $q_i \in Q$ by a transition to v_i, the duplicate of q_i created in step 1.

After this procedure terminates, the sequence of inputs a_i will start at $r_0 = q_0$ and traverse the sequence of states $r_1, \ldots, r_m$. The next input will cause a transition to a state v_i in V. Note that there is no other sequence of inputs that will traverse this specific sequence of states. As an example, consider the STG shown in figure 1. For this example, assume a three bit signature given by **010**, that, for this STG, traverses states q_0, q_1, q_2, q_3. After creating states $r_1, \ldots, r_m$ together with the states v_i, and after changing the source and destination of the involved edges in figure 1 we will obtain the STG shown

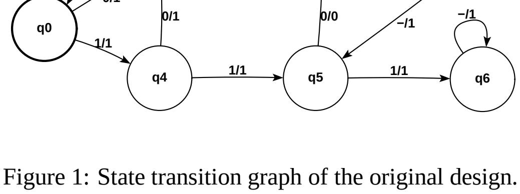

Figure 1: State transition graph of the original design.

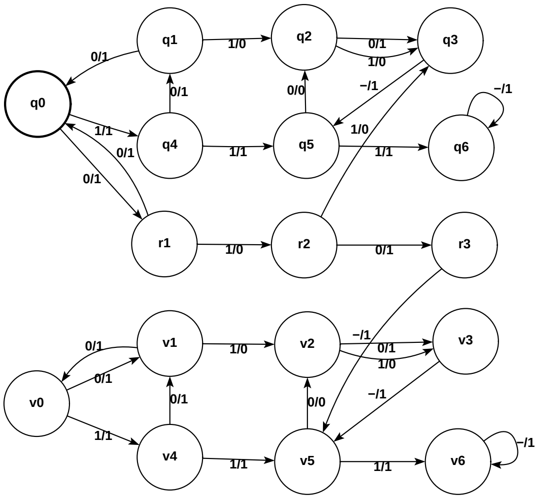

Figure 2: The modified state transition graph.

in figure 2. Note that, in this modified STG, the only way to traverse states $r_1, \ldots, r_m$ in this order, is to apply the sequence $a_1, \ldots, a_m$. We now claim that the sequence of states $r_1, \ldots, r_m$ traversed by the application of input sequence $a_1, \ldots, a_m$ exhibits a very specific property that can be used to identify this design, regardless of the state encoding used. This property is the following:

Property 1 *Each state r_i, $1 \leq i \leq m$ can only be reached from state r_{i-1}:*

$$\forall r_i \, \forall q \, \delta(q, a) = r_i \Rightarrow q = r_{i-1} \tag{3}$$

Proof: By construction of the STG, each state r_i has only one incoming edge, and this edge comes from state r_{i-1}.
□

The existence of this property for a given sequence of inputs can now be checked, as described in section 3.3. Clearly, the presence of this property for the sequence $a_1, \ldots, a_m$ is not an absolute proof that this design was marked with this specific signature. Indeed, given an arbitrary random sequence of inputs $a_1, \ldots, a_m$ and an STG, randomly picked from some arbitrary distribution, there is a finite, non-null, probability, P_β, that the sequence of states traversed by $a_1, \ldots, a_m$ exhibits property 1. We will call P_β the probability of a false positive watermark detection.

Although it is hard to derive upper bounds for the value of P_β, we present empirical evidence that, for the large majority of the designs, P_β is very low.

Additionally, we will define a second less strict property that is also exhibited by the sequence of states $r_1 \ldots, r_m$:

Property 2 *Each state r_i, $1 \leq i \leq m$ can only be reached from state r_{i-1} by applying input a_i:*

$$\forall r_i \, \delta(r_{i-1}, a) = r_i \Rightarrow a = a_i \tag{4}$$

Proof: this result is a direct consequence of the way states r_i were defined, with only one incoming transition from r_{i-1}.
□

Clearly, property 2 is usually less strict than property 1. Property 1 states that there is only one particular state in the pre-image of r_i, while property 2 states that there is only one input minterm that causes the transition from r_{i-1} to r_i, but leaves open the possibility of other edges incoming into r_i, possibly coming from other states in the STG. Although this does not happen in the modified STG, this weaker property may be useful against attacks that change unreached parts of the STG.

These two properties lead to the our main result, that represents the cornerstone of the watermarking procedure:

Theorem 1 *The STG, modified in accordance with the procedure described above, and represented by $(\Sigma, \Delta, \{Q \cup V \cup R\}, q_0, \delta', \lambda')$ is equivalent to the original STG and exhibits properties 1 and 2 for the sequence of inputs $a_1, \ldots, a_m$.*

Proof: We have already shown that the modified STG exhibits the referred properties. To prove that the STG's are equivalent, note that for every i, $v_i \equiv q_i$, since the states in V were obtained by copying the state transitions for the states q_i in Q. Therefore, state r_m is equivalent to q_{t_m}, since the outgoing edges of these states go to equivalent states. By induction, every state r_i, $1 \leq i < m$ is equivalent to q_{t_i}, which shows that the final STG is equivalent to the original.
□

3.2.1 Changing the STG using direct circuit manipulation

One important characteristic of the STG modification proposed in the previous section is that it can be obtained by a relatively simple manipulation of the circuit that realizes the original FSM.

The basic idea is to use a set of new registers to count the number of inputs that, so far, matched the desired input sequence a_i. This modification is performed using the following sequence of actions:

1. Create a number k of new registers where $k = \lceil \log_2(m+2) \rceil$. Arrange the k new registers as a counter C that can count from 0 to $m+1$.
2. Create m **and** gates G_i, $1 \leq i \leq m$ and connect their inputs so that the output of G_i is one iff the inputs have the values specified in a_i.
3. Create an **and** gate G_0 that detects the configuration for q_0 in the original state variables s^i. Connect its output as an extra input of gate G_1.
4. Connect the k bit counter such that the following is true:
   - For $1 \leq i < m$, $C_{\text{next}} \leftarrow C+1$ if $G_i = 1$ and $C = i-1$.
   - $C_{\text{next}} \leftarrow m+1$ if $C = m \vee C = m+1$
   - Otherwise, $C_{\text{next}} \leftarrow 0$

Figure 3 illustrates the modifications that need to be made in the circuit to obtain the modified STG. For illustration purposes, counter C has three synchronous control inputs: **En**, **Rst** and **Ld**, that enable its counting, reset it and enable the external load. External load has priority over count enable and reset. Analyzing these changes, it is straightforward to verify that the STG of the modified circuit has the following characteristics:

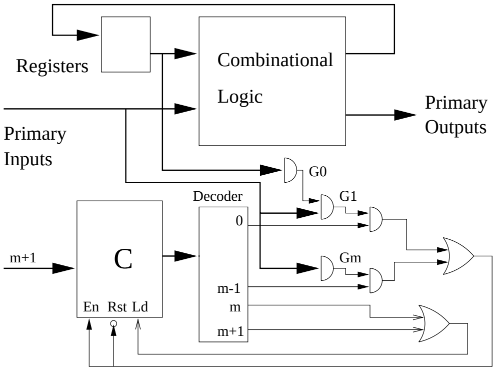

Figure 3: Changes in the circuit that create the desired modifications on the STG.

- Until gate G_1 becomes active, the circuit behaves identically to the original one, with all the registers in counter C keeping the value 0.
- When input a_1 is applied in state q_0, the output of gate G_1 becomes 1 and the counter C increments to 1.
- For each successive application of a_i, $2 \leq i \leq m$, applied in this order, gate G_i becomes active and the counter C is incremented up to the value m. If some other input is applied, the counter is reset to 0.
- Once the counter reaches the value m, its value is incremented to $m+1$ in the next clock cycle and stays at that value.

This behavior realizes the STG described in the previous section and illustrated in figure 2. For this circuit, the registers in counter C can be viewed as representing an extra set of state variables. States q_i correspond to the counter having the value 0. States r_i, $1 \leq i \leq m$ correspond to values in C varying from 1 to m. Finally, states in V correspond to C having the value $m+1$.

We remark that the presence of gate G_0 is actually not necessary, and that the STG still exhibits all the properties described above even if this gate is not there. If this gate is not present, there will exist a sequence of states with the properties exhibited by $r_1, \ldots, r_m$ starting at each state q_i in Q. There will be, in fact, $|Q|$ chains of states with these properties. By default, the watermarking procedure does not include gate G_0, unless specified by the user.

Although the modified circuit shown in figure 3 has a functionality represented by a modified STG, clearly the value of the extra state variables does not affect the value of the outputs. Therefore, although this change modifies the STG in such a way as to create the desired watermark, it is trivial to remove it by simply removing any logic and registers not connected to the primary outputs[1].

The next step will make the extra state variables influence the value of the primary outputs, thereby making their removal a complicated and time consuming process. The idea is to change the state encodings in such a way that the new state variables will change value even when inputs not belonging to the sequence representing the signature are applied. Note that the circuit in figure 3 is a fully synchronous circuit. By adding two transcoders (one before and the other one after the registers) the newly introduced state variables will not only influence the value of the primary outputs, but will also change value for inputs other than $a_1, a_2 ... a_m$. In our experiments, the transcoders shown are obtained by performing a series of linear transformations. An elementary linear transformation transforms the set of variables $X = \{x^1, \ldots, x^i, \ldots, x^j, \ldots, x^n\}$ in the set of variables $X' = \{x^1, \ldots, x^i, \ldots, x^i \oplus x^j, \ldots, x^n\}$. Any function $F : X \rightarrow \{0,1\}^k$ can be transformed into a function F' of the transformed variables $F' : X' \rightarrow \{0,1\}^k$. An arbitrary linear transformation can be obtained by a series of elementary linear transformations, each one of them implementable by the use of two **exor** gates, one gate in the transcoder used before the registers and one gate in the transcoder used after the registers. Many other transcoder functions may be used in actual applications of the method.

It may seem that the extra circuits added to obtain the modified finite state machine will have such a complexity that the method will impose a large overhead if applied to any circuit of medium complexity. Note, however, that the final circuit can be optimized using any logic synthesis tools available. This optimization may also include retiming operations, which means that the extra delay added will not be as significant as it may seem from a cursory analysis of the procedure. The experimental results in section 5 show that, in most cases of interest, the area and delay overhead are very small and well within the range of normal variations expectable from the performance of state of the art synthesis tools.

3.3 Watermark verification

Given a specific design and a signature, the verification of the presence of the corresponding watermark in that design can be made be checking for the presence of the desired properties for the set of states traversed by inputs $a_1, \ldots, a_m$. Although the theoretical worst case complexity of this procedure is provably high, it can usually be performed with reasonable computational resources using one of the approaches described in the following sections. We start by describing how property 1 can be verified.

3.3.1 Implicit computation of the pre-image

Given the transition relation for the circuit and the state codes for the set R of states reached by $a_1, \ldots, a_m$, it is straightforward to check for the presence of property 1.

This can be done by computing the pre-image of each state r_i. Given a transition relation $\mathcal{T}(S, X, T)$, the pre-image of a set R of states, defined in terms of the variables T can be computed using expression (2). By performing m pre-image computations, one for each state r_i, the check for property 1 is successful if, for $1 \leq i \leq m$, $\mathrm{PImg}(\{r_i\}) = \{r_{i-1}\}$.

In practice, and for complex designs, it may be difficult or impossible to compute $\mathcal{T}$. In this case, it is possible to perform the above computation without actually computing $\mathcal{T}$. To avoid the need to actually compute the transition relation, we note that to compute the pre-image of a state r_i, represented by the values of the state variables $r_i^1, r_i^2, \ldots, r_i^z$ it is sufficient to compute the following expression:

$$\mathrm{PImg}(\{r_i\}) = \exists X \prod_{j=1}^{j=z} \delta^j(S, X) \equiv r_i^j \tag{5}$$

In general, the computation of this expression requires much less computational resources than the computation of the full transition relation, since, in expression (5), r_i^j represents a constant value.

3.3.2 Using ATPG to compute the Pre-Image

Although the techniques shown in the previous section are easy to state and understand, their applicability is restricted to small and medium size designs. Given the nature of the property one is interested in verifying, the use of ATPG techniques provides the most efficient method to check for its existence. Knowing the set of states R traversed by $a_1, \ldots, a_m$, and knowing the specific primary input values that exercised each transition e_i, it is possible to use a standard ATPG tool to answer the following question: Is it true that for each value of i, $1 \leq i \leq m, \delta(q, a) = r_i \Rightarrow q = r_{i-1}$?

This question can be answered by forcing specific values for the output signals that correspond to the next state variables that represent r_i and using the ATPG tool to find **all** the assignments to the state variables S that justify the observed output values. This is easily done by building the circuit in figure 4 and verifying that:

- For each state r_i, a test for the fault *node X stuck at* 0 implies that $s^j = r^j_{i-1}$.
- No test exists for the fault *node y stuck at* 0.

To understand the way the algorithm works, note that an input pattern for the fault *node X stuck at 0* in figure 4 will generate the values of the previous state variables, and the input combination required to exercise the transition. If this pattern corresponds to a state other than r_{i-1}, then property 1 is not valid. Otherwise, the property may be present. On the other hand, if a test exists for the fault *node Y stuck at* 0, it means that states other than r_{i-1} are in the pre-image of r_i.

[1] For example, the `sweep` command in SIS would remove any extraneous logic and registers, not required to compute the values of the primary outputs.

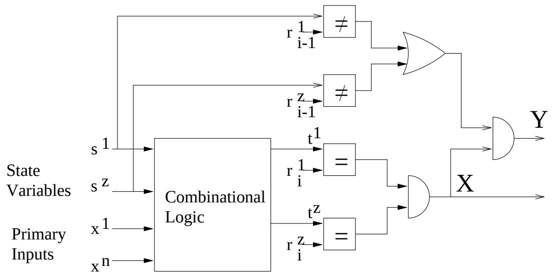

Figure 4: Circuit required to verify the presence of property 1 using ATPG techniques.

3.3.3 Verifying property 2

The results described in the previous sections describe how a circuit can be tested for the presence of property 1, given a specific signature.

It will be useful, in some cases, to be able to check also for the presence of property 2. This property, although slightly more likely to appear by chance in an unmarked STG, is more resilient to removal by sophisticated attacks, and its presence is also sufficiently conclusive to prove the presence of the watermark.

Clearly, the approach described in section 3.3.1 can be easily adapted to check for property 2. In fact, it is sufficient to verify if the expression

$$\exists S \prod_{j=1}^{j=z} \delta^j(S,X) \equiv r_i^j \quad \prod_{j=1}^{j=z} s^j \equiv r_{i-1}^j \qquad (6)$$

is true only for $X = a_i$. If that is the case, then it exists one and exactly one input combination that leads from state r_{i-1} to state r_i.

In an analog way, it is easy to adapt the ATPG based method described in section 3.3.2, by using the circuit in figure 5 and verifying that, for each state r_i, a test exists for *node X stuck at* 0 and no test exists for the condition *node Y stuck at* 0.

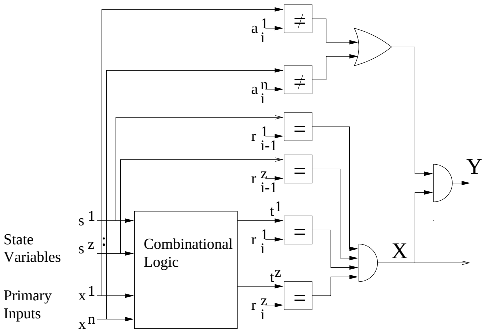

Figure 5: Circuit required to verify the presence of property 2 using ATPG techniques.

4 Possible attacks on the method

There are several different ways in which an attacker may try to circumvent the methodology presented in this work and falsely claim a given piece of IP as his own. For absolute lack of space, we will only address very briefly some of the possibilities, with many more details being given in an extended version of this work.

Basically, an attacker may try to remove the watermark by performing extensive circuit manipulation and redundancy removal. A few possibilities are: re-synthesis of the combinational logic, retiming of the circuit, arbitrary re-encoding of the states and combinational redundancy removal. All these approaches will fail, since, by not taking into consideration the sequential behavior of the circuit, they will fail to remove the properties imposed by the watermark.

More powerful approaches include STG extraction followed by state reduction [9], and sequential redundancy removal based on implicit state enumeration [6]. These approaches have the potential to remove the watermark, but are, in general, too expensive to apply to any designs of medium or large complexity. Their range of applicability is briefly addressed in the following section.

5 Implementation and results

To test the applicability of the method described in this work, we implemented the watermarking method described in this work and integrated it with the SIS [5] framework.

More specifically, we created three commands in SIS, that allowed us to perform a number of experiments. These commands are:

- **create_watermark** This command takes the current sequential network and creates a watermark, given a signature.
- **verify_watermark** This command takes the current sequential network and checks for the presence of the watermark of a specific signature.
- **verify_random_watermark** This command checks for the presence of randomly generated watermarks in a given design.

5.1 Experimental results

We used the extended set of ISCAS89 sequential benchmarks to evaluate the impact of the watermarking process. The size of the circuits in this benchmark varies widely, ranging from circuits that are too small to be effectively watermarked by the method proposed here to circuits that can be representative of real designs in terms of complexity and size.

For this set of benchmarks, we inserted the watermark that corresponds to a specific signature (actually, the one described in section 2.1), and evaluated the time and resources required to create, verify and attempt to remove the watermark. We also evaluated the impact of the watermarking creation process, both on circuit size and on circuit delay. The circuits were optimized before and after watermarking with the algebraic script from SIS, and mapped to the MSU library. The machine used to run the experiments was a 350 MHz Pentium II with 128 MBytes of memory.

Table 1 summarizes the results obtained. This table lists some statistics for each circuit (registers, literals and delay) together with the CPU times for watermark creation and for watermark verification using the two techniques described in sections 3.3.1 and 3.3.2. It also shows the area and delay overhead imposed by the watermarking procedure. We remark that, although the area and delay overhead are important for the smaller circuits, they are almost negligible for the larger circuits. In fact, for the last 10 circuits, the average area increase is only 3.2% and the average delay increase is 0.2%. For the last 5 circuits, the area and delay actually decrease, on the average, respectively 0.4% and 0.3%. These results mean that for circuits of reasonable size, the watermarking procedure has a negligible impact on both the performance and the used silicon area.

5.2 False positive detections and successful attacks

By checking for the existence of random watermarks it is possible to estimate the value of P_β, the probability of a false positive detection. From the total of 44 circuits, only one circuit (s3271) exhibited an empirical value of P_β higher than 0 (0.04) [2].

[2] Note that we performed a relatively small number of experiments (compared with the number of total watermarks) and P_β may not be exactly 0 for the other circuits.

Table 1: Statistics for circuits in the ISCAS89 benchmark, CPU times for watermark creation and verification and observed area and delay overheads.

	Statistics			CPU			Overhead	
Circuit	Reg	Lits	Del.	Cre.	ATPG	BDD	% Lits	% Del
s27 †	3	19	6.8	0.0	1.9	0.8	2747.4	273.5
s820 †	5	516	20.6	0.0	0.9	0.8	57.8	16.5
s832 †	5	547	18.2	0.0	0.8	0.5	47.0	34.1
s1488 †	6	1154	28.8	0.1	2.9	1.7	29.5	7.6
s1494 †	6	1182	27.0	0.1	3.0	1.7	30.9	16.3
s386 †	6	254	12.4	0.0	1.4	0.6	153.5	85.5
s510 †	6	432	16.8	0.0	0.6	0.8	59.3	25.0
s208 †	8	125	12.8	0.0	0.7	0.4	252.8	54.7
s27n3 †	9	175	32.0	0.0	2.8	1.3	296.6	15.0
s298 †	14	186	15.8	0.0	4.6	2.1	312.4	53.2
s344	15	231	19.6	0.0	1.1	1.0	155.0	24.5
s349	15	243	18.4	0.0	1.1	1.1	146.1	32.6
s420	16	260	24.2	0.0	0.6	0.4	88.1	5.8
s641 †	17	259	34.6	0.0	0.3	0.3	70.3	-2.3
s713 †	17	266	35.2	0.0	0.3	0.3	68.4	-4.0
s1196 †	18	974	26.8	0.0	1.4	0.7	27.7	4.5
s1238	18	1041	29.8	0.1	1.5	0.7	26.8	8.7
s991 †	19	531	63.6	0.0	0.3	3.2	50.7	6.6
s382	21	260	15.6	0.0	5.5	3.3	213.8	76.9
s400	21	269	15.6	0.0	5.6	3.4	201.9	76.9
s444	21	273	17.6	0.0	5.0	1.6	197.4	47.7
s526	21	323	16.0	0.0	5.0	2.1	168.7	67.5
s526n	21	325	16.0	0.0	5.0	2.1	167.4	67.5
s499 †	22	340	15.6	0.1	32.8	7.9	261.8	153.8
s967 †	29	697	19.8	0.0	1.0	0.5	35.6	2.0
s635	32	336	53.4	0.1	12.4	4.0	220.8	8.2
s838	32	531	46.6	0.0	0.4	0.4	31.8	6.4
s938	32	492	46.4	0.0	0.4	0.4	42.1	7.3
s1269	37	769	44.4	0.0	1.0	223.8	28.3	0.5
s1512	57	857	33.2	0.1	1.7	6.7	25.8	-13.9
prolog	65	1218	26.6	0.1	0.8	3.7	12.6	4.5
s3330	65	1222	28.2	0.1	0.9	3.9	16.4	4.3
s1423	74	1108	74.2	0.1	1.5	26.3	23.7	0.0
s4863	81	2680	83.8	0.1	1.4	26.6	6.2	0.0
s3271	116	2108	31.2	0.1	1.9	1.7	8.6	7.7
s9234	135	1816	26.6	0.1	1.4	3439.3	6.2	-1.5
s5378	162	2068	24.0	0.1	1.8	15.7	5.4	-7.5
s3384	183	2097	82.2	0.1	3.6	2589.5	7.3	-1.5
s6669	231	4141	96.0	0.2	2.0	failed	6.8	6.2
s13207	453	3769	41.4	0.4	8.9	36.1	5.1	-3.4
s15850	540	6130	70.2	0.6	27.0	470.6	2.1	-2.3
s38584	1294	17910	307.0	2.1	239.2	2468.8	0.7	4.8
s38417	1463	19258	113.8	3.4	137.2	failed	-5.2	-3.0
s35932	1728	19086	349.2	1.9	172.9	failed	-5.0	2.3

This value of P_β is simply a property of the circuit. Some circuits will exhibit a high value of P_β, and, for these, the method, if applied, can be attacked. For all the circuits that were watermarked, we also attempted to remove the watermark using one of the following methods: a) sequential redundancy removal based on implicit state enumeration [6] and b) state transition graph extraction and reduction [9]. In table 1 circuits for which is was possible to remove the watermark are marked with a †. These experiments were performed in a Sun Ultra1 machine with 384Meg of memory, with the timeout set at 24 hours of CPU time.

The results in this table show that attacks that actually remove the watermark are limited, for the examples and computational resources used, to circuits with less than 32 registers. Any realistic design that needs to be protected using these techniques is likely to be considerably more complex than this limit, thereby making the watermarking technique proposed very resilient to these and related attacks.

6 Conclusions and future work

This work presented a novel methodology for the watermarking of sequential digital designs that is, for reasonably sized designs, robust to attacks and easy to verify. This technique is the first one proposed in the literature that actually protects a design specified as a netlist, and should therefore be very interesting to designers interested in protecting their intellectual property rights. This is specially true given the strong motivation for reuse of blocks that is the key for increased effectiveness in the design of complex systems.

There are many interesting directions for future research in this area. Characterization, both from a theoretical and practical standpoint, of the class of designs for which the method is secure is an interesting topic for future research.

It is also clear that this work describes only one specific technique, applicable in a very particular setting. More general techniques, that may include functional changes in the design should be studied and analyzed, and may eventually lead to watermarking methods that may be even more robust and hard to attack.

Finally, we believe the more interesting direction for future research consists on investigating the possibility of generalizing this or similar approaches to a larger class of design specifications, including, potentially, soft IP available in the form of behavioral HDL. This technique could also be extended to the watermarking of software, an application with enormous potential impact given the amount of investment made in the development of reusable software modules.

Acknowledgments

The authors would like to thank António Leal, José Monteiro and Tiziano Villa who were kind enough to read early versions of this work and contributed with many useful suggestions.

For absolute lack of space, several relevant references and additional results were left out. They are included in an extended version of this work, available upon request.

References

[1] H. Berghel and L. O'Gorman. Protecting ownership rights through digital watermarking. *IEEE Computer*, 29(7):101–103, 1996.

[2] R. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, August 1986.

[3] E. Charbon. Hierarchical watermarking in IC design. In *Proc. Custom Integrated Circuit Conference*, pages 295–298, Santa Clara, CA, May 1998.

[4] O. Coudert, C. Berthet, and J. C. Madre. Verification of synchronous sequential machines based on symbolic execution. In J. Sifakis, editor, *Proceedings of the Workshop on Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, pages 365–373. Springer-Verlag, June 1989.

[5] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton and A. Sangiovanni-Vincentelli. SIS: A system for sequential circuit synthesis. Technical report, U.C. Berkeley, May 1992.

[6] H. Cho, G.D. Hachtel, and F. Somenzi. Redundancy identification/removal and test generation for sequential circuits using implicit state enumeration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(7):935–945, July 1993.

[7] D. Kirovski, Y. Hwang, M. Potkonjak, and J. Cong. Intellectual property protection by watermarking combinational logic synthesis solutions. In *Proc. of the ACM/IEEE International Conference on Computer Aided Design*, pages 194–198. IEEE Computer Society Press, 1998.

[8] J. Lach, W. H. Mangione-Smith, and M. Potkonjak. Signature hiding techniques for FPGA intellectual property protection. In *Proc. of the ACM/IEEE International Conference on Computer Aided Design*, pages 186–189. IEEE Computer Society Press, 1998.

[9] J.-K. Rho, G. Hachtel, F. Somenzi, and R. Jacoby. Exact and heuristic algorithms for the minimization of incompletely specified state machines. *IEEE Transactions on Computer-Aided Design*, 13(2):167–177, February 1994.

[10] I. Torunoglu and E. Charbon. Watermarking-based copyright protection of sequential functions. In *Proc. Custom Integrated Circuit Conference*, Sa Diego, CA, May 1999.