Finite State Machine IP Watermarking: A Tutorial

Amr T. Abdel-Hamid[1], Sofiène Tahar[2], and El Mostapha Aboulhamid[3]
[1] Electrical and Computer Engineering Department
Institute of Technology, West Virginia University, Montgomery, USA
Email: amr.abdel-hamid@mail.wvu.edu
[2] Electrical and Computer Engineering Department
Concordia University, Montreal, Canada
Email: tahar@ece.concordia.ca
[3] Dep. d'informatique et de recherche operationnelle
Université de Montréal, Montreal, Canada
Email: aboulham@iro.umontreal.ca

***Abstract*— Sharing Intellectual Property (IP) blocks in today's competitive market poses significant high security risks. In this paper, we present a tutorial for a watermarking approach based on the embedding of the ownership proof as part of the IP design's finite state machine (FSM). It utilizes coinciding as well as unused transitions in the state transition graph of the design. Based on this approach, we have developed a robust watermarking framework, used for copyright protection. The developed technique increases the robustness of the watermark and allows a secure implementation, hence enabling the development of the first public-key IP watermarking scheme at the FSM level. In order to integrate these algorithms in the design cycle of industrial projects, we extend the above techniques to enable the watermarking of hierarchical finite state machines (HFSMs).**

I. Introduction

Digital media piracy problems can be categorized in the following three classes: 1) *Illegal Access*, where the pirate tries to receive a digital product from a network site without permission; 2) *Intentional Tampering*, where the pirate modifies a digital product in order to extract/insert features for malicious reasons and then proceeds to its retransmission. The authenticity of the original product is lost; and 3) *Copyright Violation*, where the pirate receives a product and resells it without getting the permission to do so from the copyright owner. The same applies to Intellectual Property (IP) blocks used in digital electronic design.

In order to solve above problems, the VSI Alliance IP protection development working group [9] identifies three main approaches to secure IPs. (1) *Deterrent* approaches, where the owner uses legal means trying to stop attempts for illegal distribution; (2) *Protection* techniques that try to prevent illegal access of the IP physically by encryption. These approaches are used at the distribution phase as well, i.e., the buyer has to have the correct key to decrypt the design and so to use it. Yet, it does not secure leakage from trusted parties, such as employees, broker; (3) *Detection* approaches, where the owner detects and traces both legal and illegal usages of the designs as in watermarking and fingerprinting. These approaches mainly try to prevent copyright violation as described above. The VSI alliance proposes the usage of the three approaches for proper protection of IP designs.

The detection approaches pose an overhead on the design cycle. IP watermarking and IP fingerprinting are the main approaches in this category, where the design is watermarked (tagged) then different tracking techniques are applied to keep track of its usage. Watermarking is considered a passive approach, because the designer can only track his/her design but not stop copying or altering effectively.

In this paper, we present a tutorial for an IP watermarking technique that can be used early in the design cycle, mainly on the behavioral or register transfer levels. The watermarking approach (proposed earlier in [1]) utilizes both existing and unused transitions in the finite state machine (FSM) part of the design to embed the signature. Using existing transitions provides a supraliminal channel [1] as it would give more strength to the system against different attacks. It also helps balancing between adding enough data to identify the owner and the design overhead (area, power, delay) this data may introduce. On the other hand, the unused transitions guarantee the uniqueness of the watermark, and ensure the addition of the desired watermarking sequence.

The rest of the paper is organized as follows: Section II describes related approaches from the open literature. Section III presents our IP watermarking insertion and extraction algorithms. Section IV evaluates the performance of the presented approach and analyzes different attacks. Section V presents a prototype implementated tool and discusses different experimental results, Finally, Section VI concludes the paper.

II. Related Work

In this tutorial we are concerned primarily with FSM watermarking techniques. In following we will limit our discussions to these techniques. For more information about other watermarking techniques, the reader can refer to a survey given in [2].

[1]A low bandwidth channel that the intruder cannot afford to modify as it uses the most significant components of the object as a means of transition [5]

FSM Watermarking Based on Unused Transitions: In [17], Torunoglu and Charbon introduced the first IP protection approach through FSM watermarking. The algorithm is mainly based on extracting the unused transitions in a state transition graph (STG) of the behavioral model. These unused transitions are inserted in the STG and associated with a new defined input/output sequence, which will act as the watermark. The main advantage of this approach is the ability to detect the presence of the watermark at all lower design levels.

The authors of [17] used the probability of coincidence as the only measure for robustness, which only covered the false-positives case. The algorithm is vulnerable to masking attacks. Masking attacks do not delete the whole watermark, yet they cover the authorship information, which means that the direct detection method proposed by the authors is not reliable enough and exhaustive search or the Genome search [17] will be the main watermark extraction method. The authors did not rely on a fixed length signature, which would raise questions about the amount of information embedded especially in large designs. Finally, finding the input sequence that satisfies the probability of coincidence and is not considered with a high overhead on the STG is an NP-hard problem [12].

FSM Watermarking by Property Implanting: In [12], Oliveira tried to manipulate implicitly the STG of the finite state machine to implant the watermark as a property in the new one. The author of [12] adds extra states and transitions in a systematic way to satisfy this property. The algorithm has a low overhead on the design flow, because it does not need to go through the FSM to find the unused transitions. In [12], it is even proposed to use a very strong way to build and implant the watermark without the need of building the FSM of the design, i.e., low building overhead.

The author in [12] used a 128 bit signature, which is large enough to identify different users. The approach depends on adding a counter that checks for the input sequence expected and reaches a certain value to indicate that the design has traversed the implanted watermark. This counter can be a real weak point when it comes to masking attacks, as deleting the counter or changing its behavior means destroying the whole watermark. Also, the counter, and the way the property is added should be secret in order to insure proper security, Kerckhoffs' secrecy law. The author in [12] did not show how the probability of false positives are calculated, yet he mentioned that he calculated and found that only 2 designs from the whole IWLS93 test bench might have higher probability of false-positives larger than zero.

Furthermore, the extra states added can be removed using reduction approaches, which he proposed to solve by slightly changing the functionality of the STG. This is hard to be done mechanically as it is pretty complicated and might affect the design functionality.

III. Watermarking FSMs Using Coinciding Transitions

As a passive technique, one of the main challenges of watermarking schemes is the authenticity of the watermark. In the scheme, we describe in this paper, this problem is solved by using a secure third party, e.g., a watermarking governing body. This governing body will be responsible for generating and distributing time-stamped authenticated signatures, as well as keeping a record for such signatures for the extraction phase.

The scheme is composed of three main parts: *Signature generation, watermark insertion (embedding)*, and *watermark detection (extraction)*. The watermark embedding phase is done by the designer, where he/she uses the authentic signature to embed the watermark using one of the two alterative embedding algorithm showed below. Finally, in the manufacturing facilities and afterwards, the designer introduces the key needed to detect the watermark, in order to prove the authenticity of the design.

A. Signature Generation

The signature generation (see Figure 1), is done by the watermarking authority (third party). This will prevent intruders from searching for ghost watermark and consider it as their watermark (known as ghost attacks). The generated signature should be time-stamped as well to prevent intruders from re-embedding a new watermark in the system.

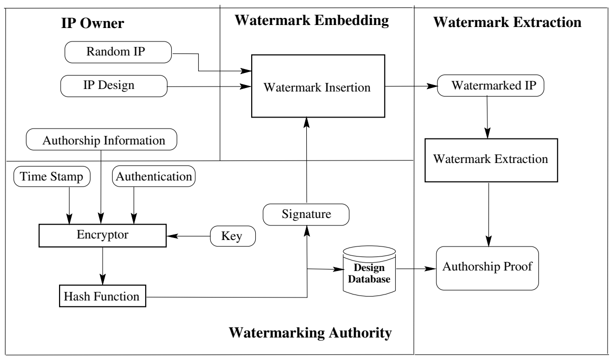
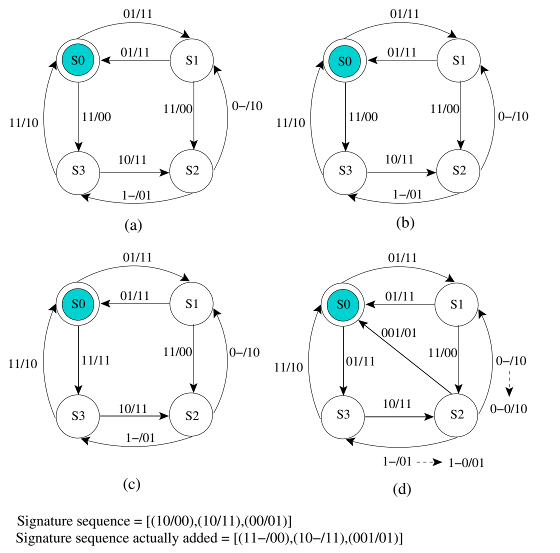


Fig. 1. IP Watermarking using a Third Entity as a Governing Body

The secure third party will use the ownership information provided by the IP designer and encrypts it using any public/private-key encryption algorithm after time-stamping it. The encrypted information is then hashed giving a short digest to decrease the watermark embedding overhead. This digest is computationally infeasible to find another message that hashes the same value. Using a constant number of bits will guarantee a certain strength for the watermark. Also, it allows the watermarking authority to specify a definite amount of bits that is long enough to differentiate and keep track of different companies.

In the above model, the owner chooses any arbitrary length message that will prove his/her ownership and encrypts it using his/her own private key of any encryption algorithm. The encrypted message is then hashed to shorten it to a certain length using a one-way hash function, MD5 [15] for instance, to generate a constant length bit sequence (128 bits) as a proof of ownership.



(a)
(b)
(c)
(d)
Signature sequence = [(10/00),(10/11),(00/01)]
Signature sequence actually added = [(11–/00),(10–/11),(001/01)]

Fig. 2. Input Comparison Watermark Insertion: Example

In following, we describe details of two watermark insertion (embedding) algorithms for plain FSMs, then an algorithm for hierarchical FSMs, and finally an algorithm for watermark extraction.

B. *Watermark Insertion Using Input Comparison Algorithm*

This algorithm associates the previously generated signature with totally randomly generated inputs, then uses these pairs as watermark transition set. Starting from an arbitrary state, these pairs will be added to the STG as follows:

1) Starting from any randomly chosen state (S_x), the random input a_w^i is compared to all inputs of the transitions associated with this state.
2) If input a_w^i is not being used in state S_x, an extra transition is added directly to the STG and the next state (S_y) will be decided randomly.
3) If input a_w^i is already being used in the selected state, the output of such pair is compared to the output of the transition, to check if it coincides with the generated signature. The transition will be then considered as part of the added signature, and the algorithm will advance to the next state that already exists in the STG.
4) In case the inputs are already being used, an extra input bit, e_w^i, is added to the system to extend the FSM. This input bit will have the same logic value for already existing transitions. For instance, a logic value '0' assigned to all existing transitions and logic value '1' will be used for the watermark transition added. The next state will be chosen randomly.
5) The algorithm will loop until the embedding of all the signature bits is done.

Figure 2 depicts a step-by-step application of the above algorithm on a simple FSM. The initial, as well as, the final signature sequences are given in the bottom of the figure. Figure 2(a) shows the original design before any signature transitions are added. Checking state S_0, input (11) is available for usage, a new transition carrying a part of our signature is then added, the next state is decided randomly, here state S_3 (Figure 2(b)). The transition pair (10/11) already coincides with an existing transition. We then advance to the next state, S_2, using this transition (Figure 2(c)). In S_2, the input (00) is already used, and the output is not coinciding with the output needed by the signature. Therefore, an extra input bit will be added for the whole design, changing (0-/10) to (0-0/10), the next state is determined randomly (Figure 2(d)).

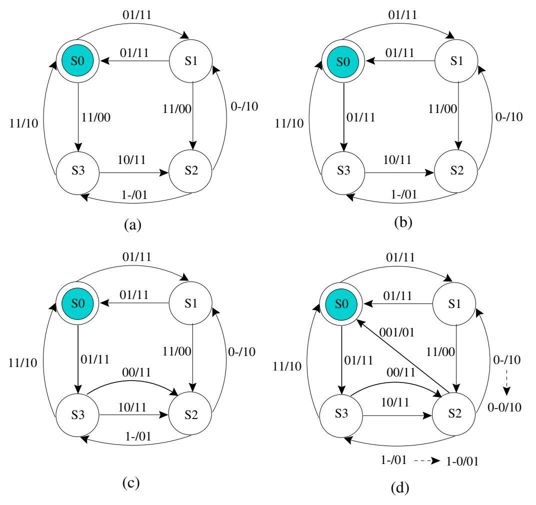
The input mapping algorithm does not search the system state of the STG to insert the watermark. This makes the algorithm faster and has a low overhead on the design flow. The algorithm is not maximizing the coinciding transitions, but it is a best-effort algorithm that randomly finds the coinciding transitions. The performance of this algorithm would decrease as the number of output bits increases, since the probability that the transitions would coincide decreases. To solve this problem, the algorithm tries several iterations. Each iteration works with different number of outputs, and tries to coincide more transitions. Also, the algorithm adds inputs even if the FSM is non-completely specified, a problem that might cause a high overhead in some systems.

C. *Watermark Insertion Using Output Mapping*

This watermark insertion algorithm coincides a part of the watermark on the FSM transitions to increase the watermark robustness. This is done by searching different outputs of each visited state in the FSM, and comparing it to a part of the generated signature in order to map this signature on the system outputs. Starting form any randomly chosen state (S_x), the watermark will be added to the FSM according to the following steps:

1) Compare the outputs of the state S_x to the generated signature to check if they coincide.
2) In case one of the outputs is equal to the watermark bits, this transition will be considered part of our watermark.
3) If the signature sequence is not equal to any of the outputs, then the inputs of S_x will be checked to determine if there is any free input that can be used to add an extra transition. The next state in this case will be chosen randomly, with preference given to states with free transitions.
4) In case all inputs are already being used, an extra input bit, e_w^i, is added to the system to extend the FSM. This input bit will have the same logic value for already existing transitions. For instance, a logic value '0' assigned to all existing transitions and logic value '1' for the watermark transition added. The next state will be chosen randomly.
5) The algorithm will loop until the embedding of all the signature bits is done.

Figure 3 illustrates an example for the above algorithm using the signature given at the bottom of the figure. Starting from state $S0$ (Figure 3(a)), we find a coinciding output (00) and move to $S3$. In $S3$ (Figure 3(b)), output 11 exists but input 00 is free. The next state in this case will be decided randomly and the algorithm advances to state $S2$. In $S2$ (Figure 3(c)), the output is not coinciding with (01) and all inputs are being used, hence an extra input bit is added to extend the whole FSM. This bit will be forced to be equal to “0” for existing transitions out of $S2$ and “1” for added ones. This extra transition will drive the FSM to state $S0$ randomly as well.



Signature sequence = [(10/00),(10/11),(00/01)]
Signature sequence actually added = [(110/00),(00-/11),(001/01)]

Fig. 3. Output Mapping Watermark Insertion: Example

It was noted that the number of coinciding transitions will decrease as the number of outputs increases. This happens as the probability to find coinciding outputs decreases as the number of output increases. To solve this problem, the algorithm tries several iterations. Each iteration works with a different number of outputs, and tries to coincide more transitions. Afterwards, the algorithm decides between the generated solutions based on the robustness, as well as lower design overhead, as discussed in the next section. It is worth to be noted that the added transitions use the whole output bits and not a part of them. This decreases the overhead further.

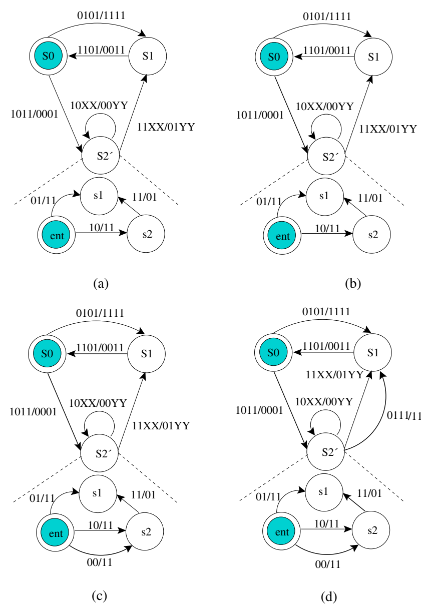
D. Watermark Insertion for Hierarchical FSMs

Practical systems have a higher number of states and transitions, This is considered a major weakness for basic FSMs, because their representation and analysis will become more difficult and the designer will be faced with the state space explosion problem. In [8], Harel introduced a Statecharts model as the first technique for hierarchal description of FSMs (HFSMs). Since then a large number of models and variations have been introduced by many people (see [18] for a description and comparison between these models). In an HFSM, a state may be further refined into another FSM to increase the system complexity. We define the HFSM in the same way as defined in Ptolemy [7]. Figure 5(a) shows an example of an HFSM machine. The inside FSM is usually called the *slave* and the outside FSM the *master* in such a composition. If the state can be refined, it is called *hierarchical state* like state *S2* in Figure 5, else it is called an *atomic state*, e.g., *S1*.

The input alphabet for the slave FSM is defined to be a subset of the input alphabet of its master FSM. Similarly, the output signals from the slave FSM are a subset of the output signals from its master. Also, the slave FSM reacts relative to the reaction of its master FSM. Girault *et al.* [7] define one reaction of the hierarchical FSM as follows: if the current state is not refined, the hierarchical FSM behaves just like a basic FSM. If the current state is refined, then first the corresponding slave FSM reacts and then the master FSM reacts. Thus, two transitions are triggered, so two actions are taken. These two actions must be somehow merged into one. Using this definition, we extend our watermarking technique for such designs as following (Figure 4):

1) Starting from any randomly chosen state S_x, check if this state is atomic or hierarchical.
2) In case of an atomic state, compare the outputs of the state S_x to the generated signature to check if they coincide.
3) In case one of the outputs is equal to the watermark bits, this transition will be considered part of our watermark. The next state will be decided according to the transition used.
4) If the signature sequence is not equal to any of the outputs, then the inputs of S_x will be checked to determine if there is any free input that can be used to add an extra transition. The next state in this case will be chosen randomly, with preference given to states with free transitions.
5) If state S_y is a hierarchical state, the entry point of the slave FSM is considered the newly reached state.
6) Compare the outputs of the state S_y to the generated signature to check if they coincide.
7) In case one of the outputs is equal to the watermark bits, this transition will be considered part of our watermark. The next state will be decided according to the transition used. In this case, the newly reached state is either an exit point or, just another state in the slave FSM.
8) If the reached state is not an exit point of the slave HFSM, the master state will be forced to stay in the same state by using the self loop transition. Only a part of the output, the slave part, will be considered in the watermark.
9) If the reached state is an exit point of the slave HFSM, the master state will be considered as an atomic state.
10) The algorithm will loop until the embedding of all the signature bits is done.

Figure 5 illustrates an example for the above algorithm using the signature given at the bottom of the figure. Starting from an atomic state $S0$ (Figure 5(a)), we find a coinciding output (0010) and move to $S2$. $S2$ is a hierarchical state, so we use its entry point (ent) as our new state. In Figure 5(b), output 11 does not exist but input 00 is free. The next state in this case will be decided randomly and the algorithm advances to state $s2$. State $s2$ is not an exit point for the slave FSM, so $S3$ will be forced to use the inputs 10 as the self loop input, and state $s2$ will still be considered our reached state. Only the first part of the output is considered a part of our watermark in



Signature sequence to be added = [0001111101]
Signature sequence actually added = [(1011/0001),(1000/0011),(1101/1101)]

Fig. 5. Watermarking HFSM: Example

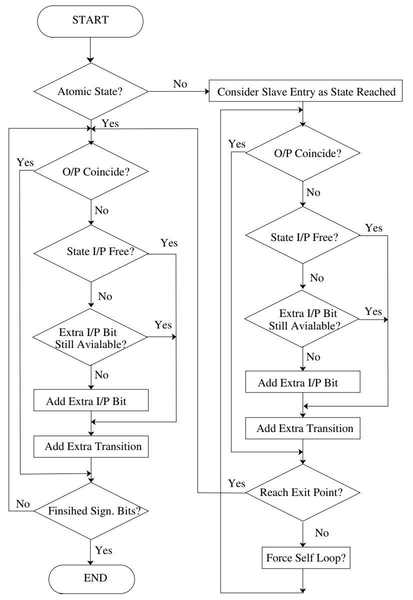


Fig. 4. Watermarking HFSM Algorithm

this case. In $s2$ (Figure 5(c)), the output coincides with (01), hence the next reached state is $s1$, which is an exit state. The output (11) is found not to coincide with any of the outputs of state $S3$, but the input (10) can be used to embed such output, and it will drive us randomly to state $S1$ as shown in Figure 5(d). This will generate a fully watermarked design using the input sequence shown below.

E. Watermark Extraction

The third phase of the watermarking process is tracking the watermarked design. We use a direct detection, by supplying the previously generated input and checking the generated output signature. Yet, direct detection is not immune to masking attacks as discussed in details in the next section. Deleting one added transition will prevent the watermark from being detected using the direct detection. Traces of the watermark still exist and it is enough to be considered as an evidence in front of a court. In such case, rebuilding the whole FSM is the ultimate solution to extract all the traces left from the watermark, but this is an expensive and complicated task.

In our approach, coinciding transitions cannot be deleted. This gives the watermarked design extra robustness as the intruder will have to decide between used and unused transitions. To solve masking attacks, we utilize an extraction algorithm making use of coinciding transitions as marks, or semaphores, to detect if the watermark traces do exist. The extraction algorithm is given as follows:

1) During the embedding of the watermark, we save the different sequence paths that lead to the coinciding transitions.
2) When the direct detection fails, we use the previously saved information to pin-point and check the availability of coinciding transitions.
3) In case of finding a coinciding transition, we consider this state as a pivot and search for all the extra added transitions that might exist around this state.
4) We extract all coinciding transitions and check for the non-deleted extra transitions in the system.

This algorithm will force the attacker to delete all extra added transitions in the system. A very hard process with a very low probability, defined as a measure of robustness in the next section. The algorithm still needs to be optimized in order to decrease the search time. Other algorithms that can help in extracting the watermark as well could be investigated, such as the Genome search proposed in [17].

IV. Evaluating Watermark Performance and Comparison

In [13], Petitcolas identified a set of measures for watermark evaluation. Although these measures were developed mainly for multimedia applications, we find some of them to be applicable to IP watermarking. We defined the requirements of any IP watermarking approach as follows:

1) Does not rely on the secrecy of the algorithm: According to one of the oldest security rules, defined by Kerckhoffs [10] in 1883, any encryption or security technique should not rely on the secrecy of the algorithm, but to the mathematical complexity of such algorithm, "*The system must not require secrecy and can be stolen by the enemy without causing trouble*".
2) Prevents intruder from re-embedding another watermark: One of the main problems facing watermarking schemes is the ability of intruders to embed another watermark in the design, especially if these are third parties and have the source code of the design.

3) Easy detection and tracking tools: Watermark insertion is only half of the process. Tracking and detection is the second important aspect in any watermarking technique. A watermarking technique should be easily detectable at all lower levels of the design, even after design manufacturing.
4) Does not affect the design functionality: Testing and verification of hardware systems is an extremely complicated task. A watermarking technique should not affect the design functionality under any circumstances, i.e., not introducing new design behavior, nor deleting behavior that already exists.
5) Embeds enough data to identify the owner of the system: The watermarking scheme should add enough data to identify the owner of the design. This data should be concrete enough to be considered as an ownership proof (ownership evidence) in front of a court.
6) Does not have high implementation overhead: Watermarking a design is a complementary process to increase its competitiveness but affecting the design performance or having a high time overhead in the insertion process would be considered a real drawback.
7) Robustness: Any proposed watermarking technique should be strong enough to face most of the attacking techniques without being totaly destroyed.
8) Asymmetric: Since Diffie and Hellman [6] presented their public encryption scheme, public techniques have proven their strength especially in non-secure environments. Sharing IP designs poses the same threats as other secret data in the public domain.

We used these aspects to evaluate and compare different watermarking algorithms. These measures are described in details in the next subsections. We will state the mathematical theorem for the soundness of the approach as a measure of perceptibility. Asymmetry or public-key operating mode will be addressed with the level of reliability, because its measure is directly related to robustness.

A. Impact on Design Functionality

A watermark insertion should not affect the behavior of the system nor interfere with the original system operation. Soundness of the watermark can be shown through the following theorem that states the relationship between the watermarked design automaton M_w and the original system M.

Theorem*: The watermarked design M_w behaves exactly as the original design M for any set of arbitrary inputs, under the condition that all extra added bits e_w, used for watermarking, are set to the same secure logical level defined at the watermarking insertion stage.*

Considering $\tilde{a}$ is any arbitrary input sequence composed of *m* elements, such that $\tilde{a} = (a^0, a^1, ...a^{m-1})$. $\tilde{a}_{wm}$ is the same input sequence for the watermarked design, where $\tilde{a}_{wm} = \langle \tilde{a}, e_{wm} \rangle$, then:

$$\forall\ \tilde{a}\ e_{wm}.\ \ e_{wm} = C \rightarrow\ \ (\lambda(q^0, \tilde{a})\ =\ \lambda_{wm}(q^0_{wm}, \tilde{a}_{wm}))$$

where C is a secure constant logical value of all extra added bits pre-defined at the time of inserting the watermark, q^0_{wm} is the initial state of the watermarked design. The theorem was proved mathematically in [2], by applying the basic concepts of FSM equivalence. It was proved that under the given condition the initial state in both designs will behave identically. And then proved that the output functions (λ and λ_{wm}) will produce the same values for any given set of input sequence.

B. Reliability and Attack Analysis

The level of reliability can be divided into two main aspects: *robustness*, which measures the strength of the hidden mark against attacks, and *false positive*, which defines the probability a watermark detector can find an ownership mark in a non-watermarked design. In the case of public-key operation, the level of reliability should include more measures for *asymmetry robustness*. In the multimedia domain, the robustness of a watermark is usually measured usually using benchmarks, e.g., Stirmark [14] for images. Benchmarking an IP watermarking scheme is much harder as the watermark might be spread in many design levels, given the different nature through the design span.

1) Attack Analysis: Removal attacks aim at the removal of the watermark information. This is attempted without breaking the security of the watermark. Removal attacks are divided into either *elimination attacks* or *masking attacks*. In elimination attacks, the intruder tries to completely eliminate the watermark. On the other hand, masking attacks do not aim at remove the watermark itself, but aim at distorting the watermark detector such that it will not be able to detect the availability of the watermark.

Coinciding transitions are considered as supraliminal channels, attacking such transitions or changing any of their values will directly result in destroying the design under investigation. Using this fact, we define the probability of watermark deletion (P^s) as the "*probability to delete at least one added transition without deleting any coinciding ones*". This will differ according to the operation mode and is calculated as ($P^s_m = \frac{m_1}{n+m_1}$) for symmetric operation mode, and as ($P^a_m = \frac{m_1}{m_1+m_2} = \frac{m_1}{m_1+m_2}$) in the public-key (asymmetric) mode, where n is the total number of transitions, m_1 is the total number of extra added transitions and m_2 is the total number of coinciding transitions.

For removal attacks, we defined the removal probability of the watermark (P_r) as "*probability any attack would change or delete all extra added transitions without deleting one original transition*". As discussed above this probability depends on the mode of operation (symmetric or asymmetric).

This will differ according to the operation mode and is calculated as ($P^s_r = \frac{1}{C^{m_1}_{n+m_1}}$) for symmetric operation mode, and as ($P^a_r = \frac{1}{C^{m_1}_{m_1+m_2}}$) in the public-key (asymmetric) mode.

2) Detecting False Positives: In [17], the probability of coincidence (P_u) for an FSM is defined as the "*odds that an unintended watermark is detected in a design*". This probability was defined in [17] under the approximation that

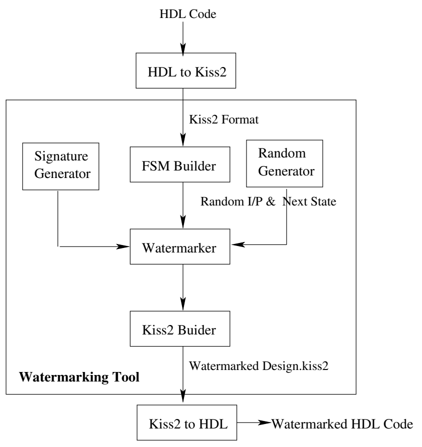


Fig. 6. Watermarking Prototype Tool

all the transitions have the same probability of occurrence as follows:

$$P_u = \frac{1}{[2^{|\Delta|}]^x - 1} \quad n \geq 1$$

where x is the number of extra added transitions and $\mid \Delta \mid$ is the total number of possible outputs. In our system, we are only adding m_2 extra transitions, but the owner still needs a sequence of length m to detect the watermark. In the case of using the MD5 hash function, a constant number of bits for the watermarking sequence (128 bits) is introduced. We can calculate the lower bound of the coinciding probability, the worst achievable case, is equal to 2.938×10^{-39}. This means that we can safely state, that our probability of coincidence is nearly constant and is larger than the above value for a 128 bits signature.

C. Watermarking Overhead

Watermark approaches rely on embedding signatures generated from hash functions in order to decrease the watermarking process overhead. In our particular case, the number of bits added by using MD5 hash function is equal to 128 bits. This amount can be increased at the expense of the number of input bits as well as the area and extra logic added to the system. Mapping more coinciding transitions directly means that we will have less overhead. The watermarking overhead is divided into three different issues: area, delay, and power. We have measured the percentage of increase on a number of benchmark designs before and after watermarking. Also, we have measured the time needed for watermark insertion. The results are shown and discussed in next section.

V. Implementation and Experimental Results

To validate and test the performance of the watermarking algorithms, we have implemented a prototype tool [4] in C++ under Unix environment. IP FSM designs can be provided in VHDL or Verilog, which we translate to kiss2 format [16]. The tool is composed of four main blocks (Figure 6). We started by building a tree for the FSM representation using the *FSM builder* block. The *signature generation* block provides the signature to the watermarker block after hashing it, while the random input and next states needed are provided using a *random generator* built in our tool. In the *watermarker* block, the user can choose the number of iterations the algorithms can run to watermark his/her design. The watermarker block also includes a decision unit that chooses between iteration results using the highest probability P_r^a that is smaller than a certain value defined by the user (10^{-6} in our case). The choice takes into consideration the lowest overhead possible (least added transitions and lowest number of added bits). Finally, using the *Kiss-to-HDL* block [3], a synthesizeable watermarked VHDL code is generated.

To evaluate our approach, the presented watermarking algorithms were applied on the IWLS93 benchmark set [11] using the FSMs generated by the available SIS tool. The benchmark circuits were synthesized using Synopsys Design Analyzer. The tables below describe the results obtained with the output mapping algorithm. Details on results with the input mapping also can be found in [1]. All experimental results was conducted on on a Sun Sparc Ultra 5 machine with a 256 MHz Processor and 512 MB of memory. Table I shows for each design, the number of inputs/outputs, and transitions/states. The total number of added transitions (m), coinciding transition (m_1), extra added transitions (m_2), and extra inputs needed to add the watermark (e_w) are also shown. C/NC in the table defines if the design under investigation is completely specified (C), so at least one input is needed, or not (NC). (N) represents the number of iterations needed, and (t) is the time needed to insert the watermark in each design is given in ms. The time for inserting the watermark is extremely short, hardly exceeded 8 seconds in the case of *scf* with 54 iterations involved (Table I), which gives a good indication about the low overhead the algorithm can introduce in the design cycle.

Table II provides the removal probabilities (P_r^s and P_r^a) discussed earlier (for the output mapping case), as well as the area, power, and delay overhead percentage as compared to the synthesized original circuit. The experimental results demonstrate that our approach has a very low effect on delay and power, especially when the design get larger. On the other hand, the area overhead is high for small designs, then decreases as the designs get larger. This is due to the large signature size compared to the original circuit. As for the robustness of the system, the algorithm failed to watermark some designs efficiently, either it could not coincide transitions, such as for S1488, where watermarking cannot operate in public-mode. This can be solved by changing the signature generated or re-watermark the design, but this will result in a higher design overhead.

VI. Conclusions

Digital watermarking has emerged as a candidate solution at the detection phase of copyright protection for IP blocks. In this paper, we presented a short tutorial on how to analyze, and implement an approach for watermarking sequential IP designs. The approach is based on the utilization of coinciding

TABLE I
EXPERIMENTAL RESULTS FOR OUTPUT MAPPING ALGORITHM

Circuit	**♯I[O]**	**♯T[S]**	e_{wm}	***m***	m_1	m_2	**C/NC**	**N**	**t**
MC	3[5]	10[4]	2	33	17	16	C	3	455
LION	2[1]	11[4]	1	128	116	12	C	1	35
DK27	1[2]	14[7]	2	64	48	16	C	1	85
EX4	6[9]	21[14]	0	26	13	13	NC	6	416
OPUS	5[6]	22[10]	0	22	9	13	NC	4	149
EX1	9[19]	138[20]	1	8	3	5	NC	17	1001
S208	11[2]	153[18]	1	64	43	21	C	1	87
STYR	9[10]	166[30]	1	15	7	8	C	8	566
SCF	27[56]	166[121]	0	3	0	3	NC	51	8021
SAND	11[9]	184[32]	0	7	0	7	NC	7	425
SAND	11[9]	184[32]	1	14	6	9	NC	7	425
S820	18[19]	232[25]	1	7	0	7	C	17	1480
S832	18[19]	245[25]	1	7	0	7	C	17	743
S1494	8[19]	250[48]	1	8	7	1	C	17	646
S1488	8[19]	251[48]	1	7	0	7	C	17	1407
KIRKMAN	12[6]	370[16]	0	22	11	11	NC	10	201
S298	3[6]	1096[218]	1	22	3	19	C	4	154
TBK	6[3]	1569[32]	1	43	18	25	C	1	144

TABLE II
EXPERIMENTAL RESULTS FOR OUTPUT MAPPING ALGORITHM (CONT'D)

Circuit	P_r^s	P_r^a	$Area\%$	$Power\%$	$Delay\%$
MC	3.770e-12	8.570e-10	323	28.6	2.4
LION	1.500e-17	4.214e-17	240	17.307	0
DK27	5.814e-17	2.046e-15	153.2	19	3.6
EX4	7.1084e-12	9.614e-8	29.3	23.4	5.8
OPUS	1.926e-11	2.010e-6	34.9	13.2	6.3
EX1	1.938e-9	0.0178	17.391	14.516	3.5
S208	1.193e-29	2.432e-17	12.6	13.2	2.7
STYR	4.094e-14	1.554e-4	13.4	15.4	1.8
SCF	1.265e-6	1	3.5	9.8	1.4
SAND	6.074e-13	1	9.2	6.3	2.6
SAND	4.373e-11	2.997e-3	41.5	12.7	1.8
S820	3.385e-11	1	8.7	7.2	1.9
S832	1.236e-13	1	6.8	6.4	0
S1494	8.494e-14	0.125	11.6	3.2	1.8
S1488	0.0038	1	14.6	3.2	0
KIRKMAN	1.369e-21	1.417e-6	2.1	1.6	0.5
S298	1.436e-41	6.493e-4	5.4	5.8	0.7
TBK	1.223e-55	1.643e-12	1.7	3.2	0.2

transitions as well as the unused transitions of the design FSM in order to give higher robustness. We also extended this approach to cover HFSMs making it possible to watermark a larger level of industrial size designs.

We defined different parameters needed to evaluate such approach and tested it using experimental results with the IWLS93 benchmark. The implemented algorithms are fast and have a comparatively low overhead on the design, which would ease their integration in the design cycle. We have also compared between our proposed approach and other approaches already available in the open literature .

REFERENCES

[1] A. T. Abdel-Hamid, S. Tahar, and E.M. Aboulhamid, "A Public-Key Watermarking Technique for IP Designs"; IEEE/ACM Design Automation and Test in Europe (DATE'05), Munich, Germany, March 2005, pp. 330-336.

[2] A. T. Abdel-Hamid, S. Tahar, and E.M. Aboulhamid, "A Survey on IP Watermarking Techniques"; International Journal on Design Automation for Embedded Systems, Springer-Verlag press, Vol. 9, No. 3, July 2005, pp. 211-227.

[3] A. T. Abdel-Hamid, M. Zaki, and S. Tahar, "A tool for Converting Finite State Machine to VHDL", Proc. of IEEE Canadian Conference on Electrical & Computer Engineering, Niagara Falls, Ontario, Canada, Vol. 4, May 2004, pp. 1907-1910.

[4] A.T. Abdel-Hamid, S. Tahar and E.M. Aboulhamid, "A Tool for Automatic Watermarking of IP Designs", Proc. IEEE Northeast Workshop on Circuits and Systems (NEWCAS'04), Montreal, Quebec, Canada, June 2004, pp. 381-384.

[5] S. Cravar, "On Public-key Steganography in the Presence of an Active Warden", Technical Report RC20931. IBM Research Division, T. J. Watson Research Center, July 1997.

[6] W. Diffie and M.E. Hellman, "New Directions in Cryptography", *IEEE Transactions on Information Theory*, Vol. 22, No. 6, 1976, pp. 644-654.

[7] A. Girault, B. Lee, and E. A. Lee,"Hierarchical Finite State Machines with Multiple Concurrency Models", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 18 , No. 6 , June 1999, pp. 742-760

[8] D. Harel, "Statecharts: A Visual Formalism for Complex Systems", *ACM Science of Computer Programming*, Vol. 8 , No. 3, June 1987, pp. 231-274.

[9] Intellectual Property Protection Development Working Group, "Intellectual Property Protection: Schemes, Alternatives and Discussion", VSI Alliance, White Paper, Version 1.1, August 2001.

[10] A. Kerckhoffs, "La cryptographie militaire", *Journal des sciences militaires*, vol. IX, Janvier 1883, pp. 538, Fvrier 1883, pp. 161191.

[11] K. McElvain, "LGSynth93 Benchmark Set: Version 4.0", Http://www.cbl.ncsu.edu/pub/Benchmark_dirs/LGSynth93/, 1993.

[12] A. L. Oliveira, "Techniques for the Creation of Digital Watermarks in Sequential Circuit Designs", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 20, No. 9, September 2001, pp. 1101-1117.

[13] F. A. P. Petitcolas,"Watermarking Schemes Evaluation", IEEE Magazine of Signal Processing, Vol. 17, No. 5, September 2000, pp. 58-64.

[14] F. A. P. Petitcolas, R.J. Anderson, and M. G. Kuhn, "Attacks on copyright marking systems", Information Hiding, LNCS 1525, Springer-Verlag, 1998, pp. 219-239.

[15] R. Rivest, "RFC 1321: The MD5 Message-Digest Algorithm", Network Working Group, 1992.

[16] E. M. Sentovich, *et al.*, "SIS: A System for Sequential Circuit Synthesis", Technical Report 94720, Dept. of Electrical Engineering and Computer Science, University of California, Berkeley, USA, 1992.

[17] I. Torunoglu, and E. Charbon, "Watermarking-Based Copyright Protection of Sequential Functions", IEEE Journal of Solid-State Circuits, Vol. 35, No. 3, February 2000, pp. 434-440.

[18] M. Von Der Beeck, A Comparison of Statecharts Variants, *In Proc. Formal Techniques in Real Time and Fault Tolerant Systems*, LNCS 863, Springer-Verlag 1994, pp. 128148.