VLSI CAD Tool Protection by Birthmarking Design Solutions

Lin Yuan, Gang Qu and Ankur Srivastava
Electrical and Computer Engineering Department and Institute for Advanced Computer Studies
University of Maryland, College Park, MD, 20742 USA

{yuanl, gangqu, ankurs}@eng.umd.edu

Abstract

Many techniques have been proposed in the past for the protection of VLSI design IPs (intellectual property). CAD tools and algorithms are intensively used in all phases of modern VLSI designs; however, little has been done to protect them. Basically, given a problem $\mathcal{P}$ and a solution $\mathcal{S}$, we want to be able to determine whether $\mathcal{S}$ is obtained by a particular tool or algorithm.

We propose two techniques that intentionally leave some *trace* or *birthmark*, which refers to certain easy detectable properties, in the design solutions to facilitate CAD tool tracing and protection. The *pre-processing* technique provides the ideal protection at the cost of losing control of solution's quality. The *post-processing* technique balances the level of protection and design quality.

We conduct a case study on how to protect a timing-driven gate duplication algorithm. Experimental results on a large set of MCNC benchmarks confirm that the pre-processing technique results in a significant reduction (about 48%) of the optimization power of the tool, while the post-processing technique has almost no penalty (less than 2%) on the tool's performance.

Categories and Subject Descriptors: K.5.1 [Legal Aspects of Computing]: Hardware/Software Protection.

General Terms: Legal Aspects, Security, Algorithms.

Keywords: CAD, protection, birthmarking, intellectual property.

1. INTRODUCTION

With the recent development of web-based CAD design environment and the intellectual property (IP) based reuse design method, the need for protocols to protect design IPs and design tools is soaring. Most of the existing IP protection standards target the protection of designs in the forms of GDSII files, netlist, or FPGA configuration bitstreams [5]. They are not developed to protect CAD tools and algorithms. Currently, traditional software protection mechanisms such as licensing agreements and encryption are used instead.

However, CAD tools and algorithms are obviously different from traditional software, their piracy takes different forms and thus have to be protected by different means. The value of CAD tools are materialized inside the designs synthesized or solved by them. Evidently, while most commonly found software piracy is illegal redistribution [2, 8], recent legal disputes involve CAD tools and algorithms are mainly on the misuse. Furthermore, the collaborated web-based design frameworks [3] make CAD tools more vulnerable than ever to unauthorized misuse.

Neither licensing agreement nor encryption is capable to detect the misuse of CAD tools. That is, a dishonest designer illegally uses a tool or an algorithm to generate an IP but claims that other tools or algorithms have been used. Given the important role that CAD tools and algorithms play in the EDA society, particularly in the IP-based design era and the collaborated web-based design environment, the need for effective CAD tools and algorithms protection becomes obvious and urgent.

Design Protection vs. Design Tool Protection

It is important to clarify the difference between protecting designs and protecting design tools. The goal of design protection is to provide a mechanism to verify, with certain degree of confidence, whether a (suspicious) piece of IP is a duplicate (either partial or identical) of another IP or contains other IPs. In design tool protection, the goal is to enable the tool and software developers to trace the usage of their licensed or sold product. In another word, we seek to answer the question that whether a particular CAD tool or algorithm has been used during the design and implementation of the suspicious IP.

Consider the scenario that engineer Alice uses Bob's placement-and-rout tool in her design. To protect the design itself, Alice can embed her watermark into the design. This watermark can be traced later on and enables her to establish her authorship of the design. However, if Alice illegally used Bob's tool (e.g, she did not renew the license agreement or she got the copy of the tool from her friend or the company that she used to work for) in her design, Bob's copyright of the tool would be violated. The goal of design tool protection is to help Bob efficiently and accurately detect whether his tool has been misused.

Existing Protection Approaches

Most CAD tools and algorithms are implemented and distributed as software programs. Typical source code protection methods include watermarking, tamper proofing, and obfuscating [2]. They will be helpful in proving the authorship on the software but cannot detect the misuse of the CAD tools and algorithms.

In the *constraint-based watermarking* mechanism [5], designers embed digital signatures or other traceable marks into IPs, as additional design constraints during the design and implementation phases. The watermark can be retrieved, if necessary, to help designers establishing their authorship or to trace the source of theft, but they do not help to verify whether a particular design tool has been used to create the IP.

Forensic engineering techniques proposed by Kirovski et al. en-

GLSVLSI'05, April 17–19, 2005, Chicago, Illinois, USA.

able the identification of solutions generated by strategically different tools and algorithms [4]. They first collect statistical data from the solutions generated by a pool of algorithms. They then study certain problem-dependent properties of the solutions to put the pool of algorithms into clusters. To detect which algorithm has been applied to obtain a given solution, they simply check the given solution for the properties that the algorithm clustering has been performed and claim that the solution is obtained by the algorithm that has the best fit.

The forensic engineering technique has several limitations. First, it can only used to distinguish strategically different algorithms. Because strategically similar algorithms will most likely find solutions with similar property, therefore clustering will not be able to separate these algorithms. Second, it requires a pool of candidate algorithms, a large number of problem instances, and the computing resource to run each algorithm on each problem instance. Moreover, it is non-trivial to put algorithms into clusters based on the characterization of solutions. If the solution's properties are not selected properly, the clustering may not be effective and could be very misleading.

Problem, Objectives, and Contributions

In this paper, we propose protection techniques to address the following problem: Given a (design, synthesis, or general) problem $\mathcal{P}$ and a solution $\mathcal{S}$, how to determine whether $\mathcal{S}$ is obtained by a particular (design, synthesis) tool or algorithm.

An effective CAD tool and algorithm protection technique must be able to 1) identify with high accuracy that whether the given solution is generated by the target tool and algorithm; 2) retain the performance (e.g., CPU and memory requirements) of the tool and algorithm as well as the quality of the solutions it provides; 3) be robust against attempts to remove the protection from the tool and algorithm or from the solutions obtained by them.

We first demonstrate that, although it is possible to apply the same constraint-based watermarking method [5] to protect CAD tools and algorithms, one cannot guarantee the quality of the solution provided by such "watermarked" tools and algorithms. This will have great impact on the performance and therefore the market value of the tool. Furthermore, this method cannot be used to protect tools and algorithms that are already in use. We then propose a new technique that can add protection to both existing and new CAD tools and algorithms while maintaining the quality of the solution.

2. GENERIC APPROACH FOR TOOL AND ALGORITHM PROTECTION

CAD tools and algorithms usually leave plenty of traces in their design solutions. For example, a graph coloring algorithm (such as greedy approaches) may have very unbalanced independent sets; and a SAT solver may prefer to assign more true values than false to variables. Such trace reflects the strategies of the tools and algorithms and will be inherent in any design solutions they find.

Our proposed birthmarking approach leverage this property and intentionally leave some traces in the design and implementation of the tools/algorithms, we call "birthmarks". Birthmark can be created either before or after the design solution is obtained, which we refer to as pre-processing and post-processing approaches respectively. The birthmarking technique has the following advantages:

- Low detection cost: "birthmark" is implemented as a certain property, so we only need to detect the existence of such property in the design solution. There is no need to conduct any of the expensive operation required by the forensic engineering method.
- Low implementation cost: we suggest to create the birthmark either before or after the design solution is obtained as a pre- or post-processing. This avoids the intrusion to the original CAD tool and algorithm.
- Good applicability: birthmarking approach can be used to protect both new and existing CAD tools. It can also be used to protect different implementation of the same algorithm, which is impossible for any known methods.
- Compatibility with watermarking: both the birthmarking and watermarking techniques enforce certain structures into the design solution for protection of different objectives. The proposed birthmarking approach can be combined with existing watermarking techniques.
- Control of solution's quality: the proposed post-processing approach maintains the high quality solution generated by the protected CAD tools and algorithms.

2.1 Pre-processing Approach

The most straightforward, and yet, arguably the most secure way to protect a tool or an algorithm is to leave trace in its strategical approach during the tool development phase. Tool developers know the implementation details of the tool and algorithm better than anyone else. This gives them advantages in embedding information (trace, birthmark) into the tool that may not be easily discovered by others. The constraint-based watermarking technique is applicable in this case. Specifically, a large part of CAD tools are heuristic algorithms for NP-hard problems. These heuristics need to make many (local) decisions to reach a good solution. Some of these local decision makings may have very little impact in the quality of the solution. Therefore, one can use additional *constraints* to force a decision, or subtly change the objective functions to guide the heuristic to solutions with special structures.

Considering a graph related problem, when there is a tie in selecting a vertex, instead of breaking the tie randomly as most of the algorithms do, one can embed one bit information by choosing the vertex with smaller index to embed a bit '0' and choosing the one with higher index for a bit '1'. This leaves a trace in the tool that will assign a birthmark to each solution obtained from the tool. The birthmark is a special structure in the solution that indicates the underlying algorithm being used to obtain the solution. It comes from, in this example, the tie breaking policy.

We call this **pre-processing** protection approach because the trace is left in the design solution before or during its generation. This approach is easy to implement during the tool/algorithm development phase where there are normally numerous choices. It will be secure and robust against all kinds of attacks; without the full knowledge (e.g., understanding the source code line-by-line) of the tool or the algorithm, it will be difficult to distinguish whether a decision is made based on the strategy of the tool/algorithm or on the embedded information. As usual, this information is encrypted using the developer's key. Therefore, even if an attacker manages to locate all or part of the message, it appears to be a random sequence without the developer's key.

However, there are two problems with this approach. First, the impact of an additional constraint to the quality of the solution is unpredictable due to the complexity of the problem. Thus the design solution's quality may be out of control. This could be fatal in most cases in the highly competitive CAD tool industry. Second, it may not be clear how a piece of constraint will affect the solution's structure and makes birthmark detection a challenging problem.

2.2 Post-processing Approach

In the **post-processing** approach, the protected CAD tool or algorithm is developed in two steps and it will generate the birthmarked solutions internally in two phases. First, we develop the tool or algorithm to optimize the solution's quality. Second, we

modify the developed tool or algorithm to add a birthmark to the solution. This separation of tool development and birthmark generation solves both problems associated with the pre-processing approach. We do not require the full knowledge of the tool or algorithm to generate the birthmark. Therefore, second step can be applied to protect existing CAD tools and algorithms.

In the first step, we will be able to obtain a high quality solution, which we can use as the guideline to *balance the quality of the design solution and the protection level of the design tool*. Specifically, we can identify places in this solution to hide information without causing large degradation of the solution's quality. Note that this cannot be achieved by the pre-processing approach because we do not know the best quality of a solution that the original tool can provide. In the second step, we can birthmark the solution by manipulating its structure such that the quality of solution is minimally affected. Such changes in the solution property will be much easier to control and detect than those imposed by the pre-processing approach.

The tool protected by post-processing approach reports only the birthmarked solution without releasing any information about the solution internally generated in the first step. From this reported solution, one can detect the birthmark and thus identify whether a given tool or algorithm has been used to generate this solution. Unlike the pre-processing approach, robustness becomes the key concern for the post-processing approach. Attacker may massage, tweak, or conduct other local changes to the birthmarked design solution in the attempt to remove or damage the birthmark. However, carefully designed post-processing technique such as spreading small "birthmarks" all over the design can reduce, if not completely eliminate, the threat from this attack.

3. PROTECTING A GATE DUPLICATION ALGORITHM

In this section, we illustrate the proposed CAD tool and algorithm protection approach by developing pre- and post-processing techniques to protect a timing driven gate duplication algorithm [7].

Given a combinational circuit, a gate duplication problem is to find a gates duplication strategy such that the overall circuit delay is minimized. Recently, Srivastava et al. have shown this problem to be NP-Complete and proposed an algorithm *DUP_EPSILON* in the post-technology mapping phase [7]. Their algorithm first traverses the network from primary outputs (PO) to primary inputs (PI) in topologically sorted order, evaluating the following pair at the input pins of each gate: the required times at the input pin if the gate is duplicated; and the ones if the gate is not duplicated. Then the algorithm traverses the network from PI to PO in topological order deciding which gate(s) to be duplicated. The last and final network traversal goes from PO to PI again in which the gates are physically duplicated. To mitigate the area increase due to the duplicated gates, in *DUP_EPSILON*, only gates on non-critical paths are to be considered for duplication. For more details of the algorithm, please refer to paper [7].

3.1 Protection Methods

We incorporate our protection schemes with the original gate duplication algorithm such that for a given gate-level circuit, we will be able to tell whether the gate duplication algorithm has been used to optimize the circuit delay or not.

We first pick a message in plain text and encrypt it into a 128-bit binary sequence. We then convert the sequence into a set of constraints that can be embedded into the algorithm. In pre-processing approach, we will add those additional constraints to influence the strategical decision in the algorithm. For every gate in the network, the first decision to be made is whether it should be duplicated or not. If this gate were duplicated, the next decision is how to partition its fanouts. Since only critical gates are considered for duplication, we use the critical gates with more than three fanouts as candidates for embedding. We choose a gate according to the first k bits in the signature (k is the base-2 logarithm of the number of candidate gates) , and keep a record of this gate's ID for detection. We then use the next two bits in signature to decide whether or not to duplicate this gate and if duplicate, how to partition its fanouts. For example, we use bit '0' to decide to duplicate the gate; and bit '1' to not to duplicate it. If a gate is duplicated, we use bit '0' to enforce that there must be a balanced partition on fanouts of this gate, and bit '1' to enforce a staggered partition. We repeat this procedure until all the signature bits have been resolved and translated into constraints. Note that the selected gates are not physically duplicated, instead these constraints restrict the freedom of how to duplicate the gates in the gate duplication algorithm. We refer to this modified algorithm as *DUP_PREWM*.

The post-processing protection approach, *DUP_POSTWM*, can be described as follows: we first obtain a temporary gate duplication solution by applying *DUP_EPSILON* algorithm to the circuit for optimization. This gives us a circuit with minimized delay. We then modify this solution by duplicating gates off the critical paths. This is guided by the signature binary sequence in a similar way as in the *DUP_PREWM* approach. The obtained new gate-level circuit will be reported as the final timing-driven gate duplication solution. Recall that *DUP_EPSILON* only considers gates on critical path for duplication because duplicating gates off critical path will less likely help to reduce the circuit delay. *DUP_POSTWM* duplicates certain non-critical gates to create "birthmark" for the solution and thus leaves trace for the gate duplication solution. Note that this whole process is integrated with the original *DUP_EPSILON* algorithm. Since it is not trivial and desirable for tool users to understand the source code of the tool line-by-line, the birthmark can be safe.

3.2 Birthmark Detection and Security

Given a gate-level circuit, the detection process is as follows: we topologically traverse the circuit from PO to PI, checking each gate whose ID matches our record and see if it has been duplicated and partitioned as the way as our pre- or post-processing protection algorithm does. If most of the gates match, we can claim that the circuit has been optimized using our protected tool and algorithm. In current integrated circuit designs, the number of gates are usually very large ranging from a few thousands to millions. The probability of a designer duplicating a single gate exactly as the way we do is very small, which provides us with a strong proof of the authorship.

There are two possible ways for an attacker to remove or corrupt the embedded birthmark:

1) The attackers locate and merge the duplicated gates. This form of attack is weak. First of all, it is not easy to identify the duplicated gates; even if somehow the attacker finds all the duplicated gates, he/she cannot tell which gate is duplicated by the birthmark embedding part in the algorithm or by the normal routine. If he/she just merge most of the duplicated gates, the delay saving achieved by gate duplication algorithm would be greatly impaired.

2) The attackers randomly choose to duplicate some gates. In this case, part of the birthmark will probably be tampered because the attacker may change the fanout partition of certain gates. However, the remaining part can still be used to show the ownership by approaches proposed in [1].

circuit	orignal	DUP_EPSILON	DUP_PREWM improve			DUP_POSTWM improve		
	delay(ns)	improve	64-bit	128-bit	256-bit	64-bit	128-bit	256-bit
c880	26.58	7.24%	5.39%	2.54%	6.24%	7.24%	7.24%	7.22%
apex6	13.69	5.48%	3.35%	-1.14%	3.96%	5.48%	5.48%	4.50%
C5315	25.17	4.22%	3.26%	1.19%	-1.83%	4.22%	4.22%	4.22%
x1	6.63	2.72%	-1.53%	1.92%	0.78%	2.72%	2.72%	2.72%
x3	11.38	6.42%	5.74%	5.40%	2.21%	6.42%	6.42%	6.42%
i7	7.66	4.79%	2.36%	-0.66%	-0.34%	4.79%	4.79%	4.79%
des	18.41	5.92%	2.82%	4.70%	1.70%	5.92%	5.92%	5.92%
i10	39.42	8.91%	6.67%	2.23%	3.16%	8.91%	8.91%	8.73%
pair	21.08	6.91%	6.58%	4.97%	0.92%	6.91%	6.91%	6.48%
frg2	15.47	9.41%	8.65%	6.95%	6.38%	9.41%	9.41%	9.41%
ttt2	12.79	7.46%	7.46%	7.46%	7.46%	6.73%	7.46%	7.46%
f51m	18.89	4.50%	6.01%	4.49%	5.58%	4.50%	4.50%	4.50%
i9	13.55	0.75%	-1.55%	-2.00%	-3.81%	0.75%	0.75%	0.75%
i8	12.85	5.01%	3.42%	1.47%	0.70%	5.01%	5.01%	5.01%
rot	16.11	4.80%	0.58%	2.09%	-1.38%	4.80%	4.80%	4.51%
term1	9.65	4.47%	0.74%	-1.85%	-2.38%	4.15%	4.47%	4.47%
too_large	11.55	5.44%	3.36%	-1.16%	5.59%	5.44%	5.44%	5.44%
apex7	10.01	1.64%	1.66%	0.83%	1.66%	1.64%	1.64%	1.64%
alu2	21.05	4.49%	0.73%	1.52%	0.10%	3.86%	4.49%	4.49%
alu4	27.29	5.76%	4.32%	3.85%	1.44%	5.76%	5.30%	5.30%
average improve		5.32%	3.50%	2.24%	1.91%	5.23%	5.29%	5.20%

Table 1: Delay improvement achieved by original gate duplication algorithm, birthmarked preprocessing and postprocessing algorithms with 64, 128 and 256 bits messages embedded respectively.

circuit	DUP_EPSILON	DUP_PREWM area increase			DUP_POSTWM area increase		
	area	64-bit	128-bit	256-bit	64-bit	128-bit	256-bit
c880	545200	-1.02%	-3.06%	-0.26%	0.43%	1.62%	1.45%
apex6	923360	-3.62%	-2.52%	-4.22%	0.10%	0.60%	1.46%
c5315	2209568	-3.38%	-3.99%	-6.26%	0.06%	0.00%	0.36%
x1	396256	-7.26%	-6.21%	-5.39%	0.35%	0.82%	1.17%
x3	965584	-1.59%	-1.92%	-2.64%	0.10%	0.96%	1.49%
i7	806432	-2.88%	-3.05%	-1.21%	0.86%	0.35%	0.98%
des	4848800	-0.08%	-2.00%	-3.93%	0.02%	0.11%	0.38%
i10	3283264	-1.63%	-3.52%	-4.95%	0.03%	0.16%	0.54%
pair	2044384	-1.29%	-3.59%	-7.54%	0.05%	0.11%	0.59%
frg2	1099680	-0.93%	-0.21%	-0.76%	0.08%	0.59%	0.76%
ttt2	274224	0.00%	0.00%	0.00%	2.37%	2.03%	0.85%
f51m	198128	-8.90%	-13.58%	-11.24%	0.94%	0.47%	0.47%
i9	788336	-3.06%	-3.65%	-3.00%	0.47%	0.24%	0.35%
i8	1315440	-0.49%	-3.03%	-4.69%	0.00%	0.53%	0.92%
rot	1001312	-2.18%	-5.93%	-8.43%	0.28%	0.37%	1.53%
term1	325728	-13.53%	-10.26%	-8.83%	0.43%	0.00%	0.00%
too_large	473280	-6.96%	-4.71%	-4.12%	0.69%	1.47%	2.55%
apex7	337328	-2.48%	-6.88%	-3.85%	1.24%	1.65%	2.34%
alu2	789728	-10.69%	-11.05%	-12.69%	0.59%	0.24%	0.76%
alu4	1342352	-2.63%	-4.87%	-10.85%	0.14%	0.76%	1.28%
average area increase		-3.73%	-4.70%	-5.24%	0.46%	0.65%	1.01%

Table 2: Area variation in birthmarked gate duplication algorithm

4. EXPERIMENTAL RESULTS

We implement both the pre- and post-processing techniques to protect the timing driven gate duplication algorithm *DUP_EPSILON*. The source code of *DUP_EPSILON* is modified to embed birthmark and are integrated in the SIS [6] environment. For each of the 20 MCNC benchmark circuits, we first optimize it via *script.rugged* followed by technology mapping. We then optimize its delay by algorithm *DUP_EPSILON* and its protected versions *DUP_PREWM* and *DUP_POSTWM*, respectively. An original message in plain text has been hashed into 64-bit, 128-bit, and 256-bit signatures and then embedded into the solution as birthmarks. We observe that these three algorithms give different circuits and we are able to recover the embedded signature bits successfully.

To measure the quality of the birthmarked solution obtained by the protected algorithms, we report the delay improvements over the original circuit achieved by the three algorithms in Table 1. On average, *DUP_EPSILON* gives 5.32% improvement. The preprocessing approach *DUP_PREWM* can only improve the delay by 3.5%(for 64-bit signature) to 1.91%(for 256-bit signature), depending heavily on the size of the signature. This implies that the capability of the gate duplication algorithm has been significantly weakened. On the other hand, the post-processing *DUP_POSTWM* approach can still maintain at least 5.2% delay improvement and it is less dependent on the signature's size. This shows that the capability of the algorithm has been preserved by the post-processing approach. We also notice that for some circuits, *i9* for example, the pre-processing approach *DUP_PREWM* finds solutions with lower quality than the original circuit (the entries with negative delay improvement in the table). This is because that the pre-processing approach embeds the birthmark by selecting certain gates for duplication and how to duplicate them. The longer is the signature, the less freedom is left for the gate duplication algorithm to operate for delay minimization.

Table 2 reports the circuit's area increase caused by gate duplication. As reported in [7], gate duplication will increase the area by 8.1% on average. In *DUP_PREWM*, the area penalty gets reduced by up to 5.24% compared to the output of *DUP_EPSILON* . This is because that the birthmark embedding procedure will mark certain gates, which would have been duplicated by the original algorithm, "not duplicable". This reduces the number of duplicated gates and therefore limits the area increase. The post-processing approach *DUP_POSTWM* will duplicate gates off critical path. Such gates will not be considered for duplication by the original algorithm. This results in area increase as shown in the last three columns of Table 2.

5. CONCLUSION

In this paper, we explore the possibility of protecting tools and algorithms directly during the design process. Pre-processing and post-processing protection techniques are proposed. The former embeds *trace* of the tool into the design solutions as their "birthmark" by constraining the tool development. High assurance is achieved in this case, but the detectability of the "birthmark" is low and the degradation of design quality could be high. The latter finds a design solution with the best possible quality first. Then it conducts an additional design step to "birthmarking" the design solution by changing it locally without affecting the design quality. A gate-level timing-driven gate duplication tool has been used as example to illustrate our approach and extensive experimental results have confirmed our conclusion.

6. REFERENCES

[1] E. Charbon. "Hierarchical Watermarking in IC Design", *CICC 98*, pp. 295-298.

[2] C.S. Collberg and C. Thomborson, "Watermarking, tamper-proofing, and obfuscation – tools for software protection", IEEE Transactions on Software Engineering, Vol. 8, **8**, 2002.

[3] A. Fin and F. Fummi. "A Web-CAD Methodology for IP-Core Analysis and Simulation", *DAC 00*, pp. 597-600.

[4] D. Kirovski, D. Liu, J.L. Wong, and M. Potkonjak. "Forensic Engineering Techniques for VLSI CAD Tools", *DAC 00*, pp. 581-586.

[5] G. Qu and M. Potkonjak. *Intellectual Property Protection in VLSI Designs: Theory and Practice*, Kluwer Academic Publishers, 2003.

[6] E. M. Sentovich et al., *SIS: A System for Sequential Circuit Synthesis*, Memorandum No. UCB/ERL M92/41, Department of EECS. UC Berkeley, May 1992.

[7] A. Srivastava, R. Kastner, C. Chen and M. Sarrafzadeh, "Timing Driven Gate Duplication," *IEEE Transactions on Very Large Scale Integrated Systems*, Jan 2004

[8] http://eet.com/news/97/946news/evidence.html