Effective Iterative Techniques for Fingerprinting Design IP

Andrew E. Caldwell, Hyun-Jin Choi, Andrew B. Kahng, *Member, IEEE*, Stefanus Mantik, Miodrag Potkonjak, *Member, IEEE*, Gang Qu, *Associate Member, IEEE*, and Jennifer L. Wong, *Student Member, IEEE*

***Abstract*—Fingerprinting is an approach that assigns a unique and invisible ID to each sold instance of the intellectual property (IP). One of the key advantages fingerprinting-based intellectual property protection (IPP) has over watermarking-based IPP is the enabling of tracing stolen hardware or software. Fingerprinting schemes have been widely and effectively used to achieve this goal; however, their application domain has been restricted only to static artifacts, such as image and audio, where distinct copies can be obtained easily. In this paper, we propose the first generic fingerprinting technique that can be applied to an arbitrary synthesis (optimization or decision) or compilation problem and, therefore to hardware and software IPs.**

The key problem with design IP fingerprinting is that there is a need to generate a large number of structurally unique but functionally and timing identical designs. To reduce the cost of generating such distinct copies, we apply iterative optimization in an incremental fashion to solve a fingerprinted instance. Therefore, we leverage on the optimization effort already spent in obtaining previous solutions, yet we generate a uniquely fingerprinted new solution. This generic approach is the basis for developing specific fingerprinting techniques for four important problems in VLSI CAD: partitioning, graph coloring, satisfiability, and standard-cell placement. We demonstrate the effectiveness of the new fingerprinting-based IPP techniques on a number of standard benchmarks.

***Index Terms*—Fingerprint, Intellectual property protection, iterative optimization, VLSI, watermark.**

I. Introduction

With the rapid deployment of new process technologies, the shrinking time-to-market requirement, and the advances in CAD tool capabilities, core-based design and software reuse methodologies have attracted a great deal of industrial and academic interest. *Intellectual property protection* (IPP) techniques are an unavoidable prerequisite for the development and adoption of reuse-based system integration business models. In such reuse-based IP business models, as well as the related IPP model, there are two basic types of legal entities involved in an IP transaction: *provider* (*seller*, *owner*) and *buyer* (*user*). Another entity, *IP intruder*, which is illegal, will attempt to infringe upon the legal entities' rights[1]. Therefore, it is the goal of IPP to protect both the provider and the buyer.

Evidently, the IPP develop and working group of the Virtual Socket Interface Alliance (VSIA) has identified the followings as the goals of IPP: *enable IP providers to protect their IPs against unauthorized use; protect all types of design data used to produce and deliver IPs; detect and trace the use of IPs* [25]. An effective IPP scheme should provide IP owner the ability to determine that an unauthorized use has occurred and then, to trace the source of the theft to protect the owner himself and the legal IP users. The recently proposed *constraint-based* watermarking IPP technique, which we will survey in the next section, is effective in helping IP providers to detect their IPs and establish their authorship from illegal copies to discourage piracy and unauthorized IP redistribution. However, it offers little help in protecting IP *buyer's* legal ownership of a given piece of IP. IP buyers desire the protection from being "framed" by other dishonest buyers working in collusion, or by a dishonest IP provider who sells extra copies of the IP and then attempts to blame the buyer. This becomes an insurmountable task if all buyers receive identical copies of the IP. VSIA has also identified the need of such protection and predicted that fingerprinting methods would be the key enabling technique.

A simple symmetric scheme extends the idea of watermarking to fingerprinting for the protection of IP buyers. Each IP buyer gives IP provider his digital signature (encrypted using the buyer's public key). IP provider converts this signature into fingerprinting constraints and integrates them with the original design constraints as well as the provider's own watermarking constraints. The synthesis tools will then generate a piece of IP that satisfies all the original design constraints and has both IP provider's watermark and the specific IP buyer's fingerprint embedded. This allows IP provider to trace individual IP buyer since each IP becomes unique with the buyer's fingerprint. It also protects the buyer in the sense that IP provider can not resell this realization of the IP to another buyer since the embedded fingerprint can only be interpreted by the first buyer via his secret key.

The difficulty of such symmetric fingerprinting protection is that IP provider most often cannot afford to apply a given wa-

Manuscript received July 29, 2002; revised April 4, 2003. This paper was recommended by Associate Editor R. Gupta.

A. E. Caldwell and S. Mantik are with Cadence Design Systems, Inc., San Jose, CA 95134 USA (e-mail: caldwell@cadence.com; stefanus@cadence.com).

A. B. Kahng is with the Computer Science and Engineering Department and the Electrical and Computer Engineering Department at the University of California, San Diego, La Jolla, CA 92093 USA (e-mail: abk@cs.ucsd.edu).

M. Potkonjak and J. L. Wong are with the Computer Science Department, University of California, Los Angeles, CA 90095 USA (e-mail: miodrag@cs.ucla.edu; jwong@cs.ucla.edu).

G. Qu is with the Electrical and Computer Engineering Department and the Institute of Advanced Computer Study, University of Maryland, College Park, MD 20742 USA (e-mail: gangqu@eng.umd.edu).

Digital Object Identifier 10.1109/TCAD.2003.822126

[1]We mention that a third party IP intruder violates both IP provider and buyer's rights, however, any legal entity can also be the IP intruder and violates the other entity's rights. For example, if a legal buyer illegally redistributes the IP he purchased from an IP provider, or if an IP provider sells the IP bound to a specific user to other users, then IP piracy occurs.

termarking technique with each buyer's signature and repeat the entire design process: creating a large number of different high-quality solutions from scratch has a clear time and cost overhead. Hence, the challenge is to develop practical fingerprinting protocols that can provide a number of distinct realization of the same IP with reasonable amortized design effort.

In this paper, we lay out the basic requirements for fingerprinting techniques and propose a generic fingerprinting methodology that applies to arbitrary synthesis (optimization/decision) problems, and that combines existing watermarking techniques and iterative approaches to solving optimization problems. Our approach allows IP owner/provider to design the IP with his (provider's) watermark embedded, in order to obtain an initial "seed" solution. We may view this initial design as a "from-scratch optimization." Then, for each IP buyer, a new *fingerprinted* optimization instance is created based on the buyer's fingerprint and some knowledge of the current seed solution. Solving this new fingerprinted instance with an "incremental optimization" yields a different but functionally identical fingerprinted IP, and is inexpensive because it leverages the design optimization effort that is inherent in the seed solution.

II. Related Work

A. IP Watermarking

A major characteristic of IP watermarking, as distinguished from artifact watermarking, is that it must maintain the correct functionality of the IP. This is the main reason why IP watermarking is not trivially achievable. The *constraint-based watermarking* technique [13] translates a to-be-embedded signature into a set of additional constraints during the design and implementation of the IP, in order to uniquely encode the author's signature into the IP. The effectiveness of this technique lies in the large solution space of the optimization problem that corresponds to the design of the IP: 1) the author's signature is added via extra constraints that reduce the solution space and 2) ownership is typically proved via the exceptionally small probability of obtaining a given solution from the initial solution space without the benefit of the signature constraints. The methodology is mathematically sound [21] and has been shown to yield strong proofs of authorship with little or no loss of solution quality, at the level of behavior synthesis [11], logic synthesis [16], and physical design [14], as well as in FPGA design [17].

B. Fingerprinting

Fingerprints have been used for human identification for a long time because of their uniqueness. Protocols have been developed for adding fingerprint-like marks into digital data to protect both the provider and the buyers [3], [4], [20]. Such marks are made by introducing minute errors to the original copy, with such errors being so insignificant that their effect is negligible. However, this is not applicable for VLSI design IP fingerprinting: a minor error can change the functionality of the IP and render the entire design useless.

To the best of our knowledge, only two published works address VLSI design IP protection using fingerprinting. The first approach is [17] to partition the problem into small parts, and impose constraints as needed to make solutions for each part "connectable." Multiple solutions are found independently for each part, and a solution to the original problem can be constructed by mixing and matching these solutions according to the buyer's fingerprint. However, the method is relatively impractical (the problem must have a specific (usually, geometric) structure, the approach can affect solution quality significantly, and it is vulnerable to collusion since fingerprinted solutions all have the same structure). The second approach [22] introduces a set of independently relaxable constraints before solving the problem. Then, once a solution is found, relaxing each constraint independently guarantees that a number of distinct solutions can be derived. This is similar to the approach of [17] in that a fingerprinted solution is obtained by independently combining elements of the solution (either solutions to sub-parts, or independent relaxations of constraints). The run-time overhead is almost zero, but many similarities are expected among fingerprinted solutions, making the approach vulnerable to collusion.

Our approach combines the concepts of constraint manipulation and the iterative improvement method for solving hard optimization problems. It is different from the above in that: 1) it can be applied to any synthesis problem; 2) it requires significantly less, although still nontrivial, time to generate fingerprinted IP instances; 3) it maintains the quality of the large number of distinct copies of the same IP; and 4) it is more secure again collusion attacks.

1. **for** $i = 0$ to $+\infty$
2. Generate a new trial solution s' from the current solution s_i
3. Decide whether to set $s_{i+1} = s_i$ or $s_{i+1} = s'$
4. **if** a stopping condition is satisfied
5. **return** the best solution found

Fig. 1. Basic template for iterative global optimization.

C. Iterative Optimization Techniques

An instance of finite global optimization has a finite solution set S and a real-valued cost function $f : S \rightarrow \Re$. Without loss of generality, global optimization seeks a solution $s^* \in S$ that minimizes f, i.e., $f(s^*) \leq f(s)\ \forall s \in S$. This framework applies to most combinatorial domains (scheduling, coloring, partitioning, quadratic assignment, etc.); continuous optimizations can also be discretized to yield finite instances. Many optimization problems are NP-hard [9], and hence heuristic methods are often applied which use an iterative approach broadly described by the iterative global optimization template of Fig. 1.

Typically, s' in Line 2 is generated by a perturbation to s_i. That is, s' is selected from the *neighborhood* of s_i under a given neighborhood operator. Example operators include changing a vertex's color in graph coloring; swapping two cells in standard-cell placement; moving a vertex to a different partition in graph partitioning; etc. Lines 2–4 can be hierarchically applied to create very complicated metaheuristics. For example, the Kernighan–Lin [15] and Fiduccia–Mattheyses [7] graph partitioning heuristics are both greedy iterative optimizers with respect to a complicated *pass* move that is itself a move-based iterative optimization. The complexity of the metaheuristic and

its sensitivity to perturbations of the instance can be a vehicle for IPP: given a solution (say, an assignment of vertices to partitions) it is typically extraordinarily difficult to identify the instance (say, the weighted edges of a graph over the vertices) for which a given metaheuristic would return the solution.

The basic idea of the proposed fingerprinting technique is to embed IP buyer's signature as additional constraints during the iterative improvement process of solving the optimization problem. In addition, we discuss how to extend this idea to decision problems and other problems that are normally not solved by iterative optimization techniques.

III. Fingerprinting Objectives

A fingerprint, being the signature of the buyer, should satisfy all the requirements of any effective watermark (see [14], [21] for details): **1) High credibility.** The fingerprint should be readily detectable in proving legal ownership, and the probability of coincidence should be low. **2) Low overhead.** Once the demand for fingerprinted solutions exceeds the number of available good solutions, the solution quality will necessarily degrade. Nevertheless, we seek to minimize the impact of fingerprinting on the quality of the software or design. **3) Resilience.** The fingerprint should be difficult or impossible to remove without complete knowledge of the software or design. **4) Transparency.** The addition of fingerprints to software and designs should be completely transparent, so that fingerprinting can be used with existing design tools. **5) Part protection.** Ideally, a good fingerprint should be distributed all over the software or design in order to identify the buyer from any part of it.

At the same time, the IPP business model implies that fingerprints have additional mandatory attributes:

- **Collusion-secure.** Different users will receive different copies of the solution with their own fingerprints embedded. These fingerprints should be embedded in such a way that it is not only difficult to remove them, but also difficult to forge a new fingerprint from existing ones (i.e., the fingerprinted solutions should be structurally diverse).
- **Runtime.** The (average) run-time for creating a fingerprinted solution should be much less than the run-time for solving the problem from scratch. The complexity of synthesis problem and the need for large quantity of fingerprinted solutions make it impractical to solve the problem from scratch for each individual buyer.
- **Preserving watermarks.** Fingerprinting should not diminish the strength of the author's watermark. Ideally, not only should the fingerprinting constraints not conflict with the watermarking constraints, any hint on the watermark from fingerprints should also be prevented as well.

From the above objectives, we extract the following key requirements for fingerprinting protocols:

- A fingerprinting protocol must be capable of generating solutions that are "far away" from each other. If solutions are too similar, it will be difficult for the seller to identify distinct buyers and it will be easy for dishonest buyers to collude. In most problems, there exist generally accepted definitions for distance or similarity between different solutions.

1. Create a watermarked instance I_0 by embedding the IP provider's watermark into the initial instance I;
2. Generate a (watermarked) initial solution S_0 from I_0 by a *from-scratch* optimization;
3. **for** $j = 1$ to n (n is the number of IP buyers)
4. Create a fingerprinted instance I_j with the j-th buyer's fingerprint F_j added into I_0;
5. Using S_0 as the initial solution, apply an *incremental* optimization to generate a fingerprinted solution S_j for I_j;

Fig. 2. The generic iterative approach for generating fingerprinted solutions.

- A fingerprinting protocol should be nonintrusive to existing design optimization algorithms, so that it can be easily integrated with existing software tool flows.
- The cost of the fingerprinting protocol should be kept as low as possible. Ideally, it should be negligible compared to the original design effort.

IV. New Fingerprinting Approach

To maintain reasonable run-time while producing a large number of fingerprinted solutions, we will exploit the availability of iterative heuristics for difficult optimizations. Notably, we propose to apply such heuristics: 1) in an *incremental* fashion and 2) to design optimization instances that have been perturbed according to a buyer's signature (or fingerprint).

Fig. 2 outlines the proposed approach. Lines 1 and 2 generate an initial watermarked solution S_0 using an (iterative) optimization heuristic in "from-scratch" mode. Then we use this solution as the "seed" to create fingerprinted solutions as follows: Line 3 embeds the buyer's signature into the design as a fingerprint (e.g., by perturbing the weights of edges in a weighted graph) to yield a fingerprinted instance. This fingerprinted instance is then solved by an *incremental* iterative optimization using S_0 as the initial solution.

This generic approach provides the fingerprinting desiderata we present above. Comparing to the symmetric fingerprinting method presented in Section 1, it has the following advantages.

- **Shortened run-time.** We leverage the design optimization effort that is inherent in the high-quality "seed" solution S_0, by using it as the starting point, to reduce the run-time for reaching the stopping criterion.
- **Distinct solutions.** The addition of fingerprinting constraints will subtly change the problem instance to break the local minimality of the starting solution S_0 and to help the iterative optimizer to find the fingerprinted solution, a new local minimum.
- **Improved solution quality.** Adding fingerprinting constraints also changes the *optimization cost surface* and can actually lead to improved solution quality, as noted in the metaheuristics literature [19], [24].
- **Alternate starting point.** Alternatively, we could use S_{j-1} as the initial solution in Line 5 of Fig. 2. This will more likely to make all the fingerprinted solutions to be "far away" from each other and ultimately reduce the chance of collusion.

Next, we will present specific fingerprinting techniques for four classes of VLSI CAD problems to show how to apply this

1	Obtain an initial partitioning solution S_0 by finding the best solution out of 40 starts of CLIP FM for the original instance I_0;
2	Select a subset $E' \subset E$ of size equal to $r\%$ of the total number of hyperedges in H according to the j-th user's fingerprint;
3	Create the fingerprinted instance I_j for the j-th user
3.1	Reset all hyperedge weights to 20;
3.2	for the i-th hyperedge $e_i \in E'$
3.3	if (the i-th bit of user's fingerprint is 0)
3.4	increase the weight of hyperedge e_i by 19;
3.5	else
3.6	decrease the weight of hyperedge e_i by 19;
4	Partition the hypergraph instance I_j using a single start of CLIP FM with S_0 as the starting solution to obtain the fingerprinted solution S_j;
5	Goto step 2 if another fingerprinted solution is needed;

Fig. 3. Pseudocode of the iterative fingerprinting approach on the partitioning problem.

approach to: 1) classic iterative optimization algorithms; 2) optimization problems that may not be solved by iterative improvement; and 3) decision problems.

A. Partitioning and Standard-Cell Placement

Given a hyperedge- and vertex-weighted hypergraph $H = (V, E)$, a *k-way partitioning* of V assigns the vertices to k disjoint nonempty partitions. The *k-way partitioning problem* seeks to minimize a given objective function such as the *cut size*, i.e., the number of hyperedges whose vertices are not all in a single partition. In our partitioning testbed, we use the recent CLIP FM variant [6] and the net cut cost function.

Fig. 3 depicts how to iteratively construct a sequence of fingerprinted solutions for the partitioning problem. Step 1 finds a "seed" solution by CLIP FM with multiple starts. This not only gives IP provider a high quality watermarked solution, but also demonstrates that our approach can efficiently generate, from a high quality solution, new solutions of the same or better quality. Steps 2 and 3 construct a fingerprinted instance based on user's fingerprint. Note that for simplicity, we reset all the hyperedge weights to 20 and change the weights of the selected hyperedges to 39 or 1 to embed a bit "0" or "1," accordingly. When the weights are tightly constrained by the performance and other design requirements, we should perform such fingerprinting process on the set of hyperedges with loose constraints and modify their weights in a less dramatic fashion. Step 4 solves for a fingerprinted solution. We use only one start since our CLIP FM implementation is deterministic. This results in a much-reduced run-time to obtain a new solution, comparing to that for the multiple-start CLIP FM in Step 1. Our experiments will validate the quality of the new solution, which is expected to be high because it is based on a carefully selected high-quality "seed" solution.

Although we use a deterministic CLIP FM implementation, it is very unlikely for the iterative fingerprinting approach to generate identical solutions at Step 4 because user's signature gives different fingerprinting constraints. In fact, we can use S_{j-1} instead of S_0 as the starting solution to guarantee the uniqueness of every fingerprinted solution as in the following standard-cell placement problem.

1	Obtain an initial placement solution S_0 for the instance I_0 in LEF/DEF format;
2	Select a subset $N' \subset N$ of the signal nets in the design according to the j-th user's fingerprint;
3	Create the fingerprinted instance I_j for the j-th user
3.1	Reset the weights of all signal nets to 1;
3.2	Set the weight of each net in N' to 10;
4	Incrementally re-place the design, starting from the current solution S_{j-1} and using the new net weights, to obtain the fingerprinted placement solution S_j;
5	Goto step 2 if another fingerprinted solution is needed;

Fig. 4. Pseudocode of the iterative fingerprinting approach on the standard-cell placement problem.

The standard-cell placement problem seeks to place each cell of a gate-level netlist onto a legal site, such that no two cells overlap and the wirelength of the interconnections is minimized. We iteratively construct, as shown in Fig. 4, a sequence of fingerprinted placement solutions according to the following steps (note that our approach is compatible with the LEF/DEF and Cadence QPlace based constraint-based watermarking flow presented in [14]). Note that when we create the new solution in Step 4, we start with the current solution S_{j-1} rather than the "seed" solution S_0 by invoking the *Incremental Mode* of the Cadence QPlace tool (version 4.1.34). S_{j-1}, a local optimal solution for the fingerprinted instance I_{j-1}, loses its local optimality after we reset the weights of the selected nets in the new fingerprinted instance I_j (Step 3). This enables us to find solutions "far away" from the "seed" and increases the resilience against collusion attacks. However, all previous fingerprinting constraints are inherent in the new solution and affect its quality.

B. Graph Coloring

The graph vertex coloring (**GC**) optimization seeks to color a undirected graph with as few colors as possible, such that no two adjacent vertices receive the same color. Most GC algorithms can be classified into three categories: exact [5], constructive [10], and iterative improvement [8], [12]. It has been shown that iterative improvement methods (such as simulated annealing and generic tabu search) are the most effective for random graphs while exact coloring is better for real-life CAD-related graphs [5]. It becomes important and interesting to study whether the proposed iterative fingerprinting technique is applicable when the underlying optimization algorithm (e.g., exact algorithm) does not possess the nature of iterative improvement.

Given a graph $G(V, E)$ and an algorithm $\mathcal{A}$, a coloring solution is essentially a partition of vertices into disjoint independent sets (IS) where all vertices in an IS will be assigned the same color. Fig. 5 illustrates our approach to iteratively construct a sequence of coloring solutions from a known (watermarked) solution S_0. Note that the algorithm $\mathcal{A}$ does not necessarily need to be an iterative improvement. To reduce the run-time of finding a high-quality solution, we create a fingerprinted graph $G'(V', E')$ in Step 3 by: 1) deleting the maximal ISs from the selected ISs; 2) collapsing the nonmaximal ISs to a single vertex; and 3) embedding user's fingerprint as additional constraints. Graph G' will be colored in Step 4 and the fingerprinted solution can be easily constructed in Step 5.

1 Obtain a coloring solution $S_0 = \{C_1, \ldots, C_k\}$ by applying algorithm $\mathcal{A}$ to graph G, where C_i is an independent set and all the vertices in a given C_i receive the same color;
2 Select $\{D_1, \ldots, D_l\} \subset \{C_1, \ldots, C_k\}$ (according to the i-th user's fingerprint);
3 Create the fingerprinted graph $G'(V', E')$
3.0 $V' = V; E' = E$;
3.1 for $(j = 1, \cdots, l)$
3.2 { if (D_j is a maximal independent set)
3.3 $V' = V' \setminus D_j$ = {vertices in V' but not in D_j};
3.4 $E' = E' \setminus \{(u, v) | u \in D_j$ or $v \in D_j\}$
= {edges in E' with no endpoints in D_j};
3.5 else /* collapse the set D_j to one vertex v^j */
3.6 { select $v^j \in D_j$ randomly;
3.7 $V' = (V' \setminus D_j) \cup \{v^j\}$;
3.8 $E' = (E' \setminus \{(u, v) | u \in D_j$ or $v \in D_j\}) \cup \{(u, v^j) | u \notin D_j$ and $\exists v \in D_j, s.t. (u, v) \in E\}$;
}
}
3.9 G' = *watermarking* (G', user's fingerprints);
4 Obtain a coloring solution for graph G';
5 Create the fingerprinted solution for graph G
5.1 for $(j = 1, \cdots, l)$
5.2 { if (D_j is a maximal independent set)
5.3 assign all vertices in D_j a new color;
5.4 else
5.5 assign all vertices in D_j the same color of v^j;
5.6 }
6 Go to step 2 if another fingerprinted solution is needed;

Fig. 5. Pseudocode of the iterative fingerprinting approach on the graph coloring problem.

We first mention that the algorithm $\mathcal{A}$ can be any graph coloring algorithm, not necessarily one to follow the iterative improvement approach. Secondly, the fingerprinted solution we obtain in Step 5 will be different from the initial solution S_0 because S_0 does not guarantee the satisfaction of the user's fingerprinting constraints, which are added in Step 3.9[2]. Finally, the run-time in obtaining a fingerprinted solution should be less than that of solving the problem from scratch because we are coloring a smaller graph where the presumably high-quality maximal ISs from the original solutions are preserved.

C. Satisfiability

As the final example, we show that the proposed iterative fingerprinting approach can also be applied to hard decision problems such as the NP-complete boolean satisfiability (SAT) problem. The SAT problem seeks to decide, for a given formula, whether there exists a truth assignment for the variables that makes the formula true. For a satisfiable formula $\mathcal{F}$, a solution is an assignment of 0 (false), 1 (true), or—(don't care) to each of the variables[3]. We necessarily assume that the given SAT instance is satisfiable and that it has a sufficiently large solution space to accommodate multiple fingerprinted solutions.

[2]We also mention that it is not required to color the fingerprinted graph G' by the same GC algorithm $\mathcal{A}$ in step 4. This could pull the new solution further away from the initial solution S_0.

[3]While some SAT solvers give only the truth variables and assume the rest are all false, other solvers do give don't care value to variables. If k variables are assigned don't cares in a solution, essentially this solution is equivalently to 2^k distinct solutions.

1 Solve $\mathcal{F}$ for an initial solution S_0: $\{v_1 = b_1, \cdots, v_n = b_n\}$, where $b_i \in \{0, 1, -\}$;
2 According to the i-th user's fingerprint, select a subset of variables: $V' = \{v_{i_1}, \cdots, v_{i_l}\} \subset V$;
3 Create the fingerprinted formula $\mathcal{F}'$
3.0 $\mathcal{F}' = \mathcal{F}$;
3.1 for $(j = 1, \cdots, l)$
3.2 { if $(b_{i_j} = 1)$
3.3 $\mathcal{F}' = \mathcal{F}'_{v_{i_j}}$;
3.4 else if $(b_{i_j} = 0)$
3.5 $\mathcal{F}' = \mathcal{F}'_{v'_{i_j}}$;
3.6 else $\mathcal{F}' = \mathcal{F}' \setminus v_{i_j}$;
}
3.7 $\mathcal{F}'$ = watermarking ($\mathcal{F}'$, user's fingerprints);
4 Solve $\mathcal{F}'$ for an assignment to the variables in $V \setminus V'$, variables that are in V but not in V';
5 Construct the fingerprinted solution for formula $\mathcal{F}$
5.1 for $(j = 1, \cdots, l)$
5.2 $v_{i_j} = b_{i_j}$;
6 Go to step 2 if another fingerprinted solution is needed;

Fig. 6. Pseudocode of the iterative fingerprinting approach for the Boolean satisfiability problem. $\mathcal{F}'_{v_{i_j}}$ (Step 3.3) and $\mathcal{F}'_{v'_{i_j}}$ (Step 3.5) are the *cofactors* of $\mathcal{F}'$ w.r.t. variable v_{i_j}. $\mathcal{F}' \setminus v_{i_j}$ (Step 3.6) is obtained by removing both v_{i_j} and v'_{i_j} from $\mathcal{F}'$.

Iterative improvement techniques cannot be applied to generate new SAT solutions from an existing one[4]. Our fingerprinting goal is to efficiently construct a sequence of distinct solutions from a given solution. We achieve this by iteratively building and solving "new" SAT instances with (much) smaller size. Fig. 6 outlines our approach on a formula $\mathcal{F}$ over boolean variables $V = \{v_1, \ldots, v_n\}$. Here we only mention that the reduction on run-time is a result of (1) the cofactoration in steps 3.3 and 3.5 as well as in 3.6 which reduce the size of the (fingerprinted) SAT instance and (2) the preservation of the values for a selected subset of variables which reuses the effort in finding the initial solution.

V. Experimental Results

We have conducted experiments on benchmark data for the above four problems. The goal is to verify that the proposed iterative fingerprinting approach meets the fingerprinting objectives and requirements as we discussed in Section III. In particular, we focus our analysis on: 1) the run-time for creating multiple fingerprinted solutions; 2) the quality of the fingerprinted solutions (except for the SAT problem as we have explained earlier); and 3) the distinctness among the fingerprinted solutions.

A. Partitioning

We test our fingerprinting method on three standard test cases corresponding to internal IBM designs from the ISPD-98 Benchmark Suite [1]. Table I reports the size of these designs

[4]One can develop local search heuristics (e.g., flip the assignment to some variables and then modify the assignments of others whenever necessary to keep all the clauses in the formula to be true) to find other solutions by taking advantage of the known truth assignment. However, it is not clear whether any kind of such heuristics is better than resolve the problem from scratch. This is due to the intrinsic nature of the SAT problem and the well-known fact that "local search is as hard as global search."

TABLE I
PARTITIONING EXPERIMENTS TEST CASES AND RESULTS

Test Cores	IBM01		IBM02		IBM03	
# of Vertices	12752		19601		23136	
# of Hyperedges	14111		19584		27401	
	Max	Ave	Max	Ave	Min	Ave
S_0 Cost	308	252.2	296	272.0	1047	881.5
S_0 CPU Time	261	187.6	379	329.8	808	695.5
S_i Cost	307.2	253.8	278.5	273.1	912.5	867.5
S_i CPU Time	3.25	2.42	15.2	8.4	36.1	18.4
Hamming Dist.	16.3	71.1	7.3	41.2	66.6	263.4

TABLE II
TEST CASES FOR STANDARD-CELL PLACEMENT EXPERIMENT

Test Case:	Test1	Test2	Test3	Test4
Number of Cells	3286	12133	12857	20577
Number of Nets	2902	11828	10880	25634

and the average results of the 20 independent trials (i.e., we go through Step 2 of the method in Fig. 3 to generate 20 fingerprinted solutions). The *Max* and *Ave* columns give the maximum and average solution cost and the CPU times (on a 300-MHz Sun Ultra-10 running Solaris 2.6.) to generate a solution. The last row shows the minimum and average Hamming distances (i.e., number of transpositions required to transform one solution into another) over all pairs among the solutions $S_0, S_1, \ldots, S_{20}$.

The data show that the fingerprinted solutions: 1) require much less CPU to generate than the original solutions (by factors ranging from 18 to 77); 2) are reasonably distinct from each other and from the original solutions; and 3) can even have better average quality than the original solutions (which we attribute to the similarity between our fingerprinting methodology and the problem-space iterative optimization metaheuristic [19]).

B. Standard-Cell Placement

For standard-cell placement, we have applied our fingerprinting technique to the four industry designs listed in Table II. For each test case, we generate an initial solution S_0 and a sequence of 20 different fingerprinted solutions $S_1, \ldots, S_{20}$; for each fingerprinted solution, the previous fingerprinted solution is used as the initial solution for QPlace Incremental Mode. Table III gives the summary of results for all four test cases where we change the weighted sets from 1% to 5%. The "Original" lines refer to the initial solutions S_0 and all other lines refer to fingerprinted solutions. From this table we can see that we can reduce the time to generate the next fingerprinted solution while maintaining the quality as well as producing a unique solution. Detailed analysis of the solutions can be found in the technical report [2].

C. Graph Coloring

The proposed GC fingerprinting technique has been tested on both real-life benchmarks and the DIMACS challenge graph. The real-life benchmark graphs are converted from register allocation problem of variables in real codes with known optimal solutions [28]. They are easy to color and almost all the original and fingerprinted graphs are colored instantaneously with no extra colors. However, the DIMACS challenge graph,

TABLE III
SUMMARY OF RESULTS FOR FINGERPRINTING OF ALL FOUR STANDARD-CELL PLACEMENT INSTANCES. *CPU (mm:ss)*: RUN-TIME, *COST*: TOTAL WIRELENGTH NORMALIZED TO THAT OF THE INITIAL SOLUTION S_0, *DIST.*: MANHATTAN DISTANCES (IN 10^6 MICRONS) OF S_j FROM S_0

Test cases	weighted nets	Normalized Cost		CPU (sec)		Manhattan Dist.	
		Max	Ave	Max	Ave	Min	Ave
Test1	Original	1.000	1.000	97	97	0	0
	1%	1.009	0.986	53	51	0.530	0.861
	2%	1.003	0.987	54	50	0.509	0.886
	5%	1.024	0.992	58	53	0.801	1.084
Test2	Original	1.000	1.000	384	384	0	0
	1%	0.994	0.976	220	205	3.532	5.829
	2%	1.001	0.982	214	202	3.757	5.662
	5%	1.023	1.004	210	198	3.946	5.992
Test3	Original	1.000	1.000	344	344	0	0
	1%	1.000	0.988	168	164	3.232	4.048
	2%	1.007	0.995	165	162	3.239	4.123
	5%	1.015	1.009	166	163	3.503	4.423
Test4	Original	1.000	1.000	960	960	0	0
	1%	1.006	0.983	510	499	11.39	15.53
	2%	0.997	0.987	506	485	13.06	17.44
	5%	1.031	1.018	514	496	12.35	18.02

TABLE IV
RESULTS FOR COLORING THE DIMACS CHALLENGE GRAPH WITH ITERATIVE FINGERPRINTING

% of ISs Recolored	Nodes	Edges	Edge Prob.	solution		Run-time (hours)
				Ave.	Best	
Original	1000	249826	0.5002	86	86	15.45
20%	222	13630	0.5556	86	86	1.10
30%	319	26908	0.5305	86	86	1.46
40%	397	40712	0.5179	86	86	2.02
50%	495	62726	0.5130	87	87	3.07
60%	593	89006	0.5071	87	87	4.55
70%	694	121591	0.5056	87.2	87	6.42

which is a random graph with 1000 vertices and an edge probability slightly larger than 0.5, is hard and the optimal solution is still open [26]. Results on this graph demonstrates not only the tradeoff between solution quality and fingerprint's credibility, but also the run-time saving of the proposed iterative fingerprinting approach in generating new solutions.

We first color the graph once to obtain an 86-color "seed" solution and then apply the approach in Fig. 5 to build fingerprinted solutions. The fingerprinting constraints are embedded by the watermarking method "adding edges" [21]. We color each fingerprinted graph five times. Parameters of the fingerprinted graphs and solutions, along with the average run-time, are reported in Table IV. The first column gives the percentage of independent sets to be recolored; the next three columns are the number of vertices, edges, and the edge probability of the fingerprinted graph; the "solution" columns show the average number of colors to color the fingerprinted graph and the best coloring solution we have found in five tries; the last column is the average run-time for finding one solution.

We can see that as the number of recolored ISs goes up from 20% to 70%, the fingerprinted graph will have more vertices to accommodate more fingerprinting constraints. This consequently increases the credibility of the fingerprint. However, the solution quality, measured by the number of colors used to color the graph, degrades despite more time being spent to find a solution. The degradation of solution quality is a direct result of adding more fingerprinting constraints. The longer run-time is

TABLE V
NUMBER OF UNDETERMINED VARIABLES (*VAR.*), AVERAGE DISTANCE FROM ORIGINAL SOLUTION (*DISTANCE*), AND AVERAGE CPU TIME (IN $1/100$THS OF A SECOND) FOR FINGERPRINTING SAT BENCHMARKS

Original Solution			20% Preserved			30% Preserved			50% Preserved		
File	Var.	CPU	Var.	Dist.	CPU	Var.	Dist.	CPU	Var.	Dist.	CPU
ii8a1	66	8.0	52	20	6.6	46	16	7.2	33	21	6.8
ii8a2	180	9.2	144	35	7.0	126	33	6.8	90	24	7.0
ii8a3	264	9.4	211	48	8.4	184	43	7.4	132	38	7.4
ii8a4	396	12.8	316	78	8.4	277	60	7.8	198	56	9.2
ii8b1	336	9.6	268	80	9.0	235	53	7.4	168	55	8.4
ii8b2	576	13.8	460	146	7.6	403	138	7.2	288	147	7.0
ii8b3	816	18.8	652	203	7.8	571	198	8.4	408	224	8.0
ii8b4	1068	30.8	854	272	8.2	747	246	9.0	534	277	7.4
ii8c1	510	12.4	408	92	8.0	357	103	8.4	255	72	7.8
ii8c2	950	17.4	760	218	8.6	665	238	7.8	475	246	7.4
ii8d1	530	12.2	424	68	8.6	371	60	8.8	265	71	7.8
ii8d2	930	17.2	744	246	7.8	651	251	8.0	465	212	7.2
ii8e1	520	12.0	416	82	8.4	364	56	8.6	260	68	6.8
ii8e2	870	17.2	696	165	7.4	609	223	8.0	435	98	8.2
Ave. Distance (%)			-	22%	-	-	20%	-	-	19%	-
Ave. CPU Saving (%)			-	-	38%	-	-	39%	-	-	41%

due to the fact that the size of the fingerprinted graph becomes larger and more structural information from the "seed" solution is removed as we are recoloring more ISs. Still, we see significant run-time savings over the original from-scratch run-time (15+ h) in all cases.

D. Satisfiability

The SAT instances in our experiments, which are generated from the problem of inferring the logic in an 8-input, 1-output "blackbox," are from DIMACS [26]. All instances that we use are satisfiable and WalkSAT [27] is used as the satisfiability solver. As described earlier, we start with solving each original instance to obtain the "seed" solution. Then part of this solution is preserved according to the user's fingerprint. This is done by selecting a subset of the variables, fixing their assignments, and removing them (along with their complements) from the formula. We find a solution for the smaller fingerprinted instances and compare the Hamming distance between these solutions and the original solution. The Hamming distance between two solutions $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$ and $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$ is defined as: $\mathrm{dist}(\mathcal{S}, \mathcal{T}) = \sum_{i=1}^{n} |s_i - t_i|$.

We experiment maintaining the assignments to $k\%$ of the variables in the "seed" solution, where k goes from 5% to 60% with an increment of 5%. Table V reports the cases when k is set to be 20%, 30%, and 50%. As indicated from the last two rows, we are able to find solutions that are 20% different from the seed with a CPU saving around 40%. Intuitively, as we increase the percentage of preserved variables, one may expect more run-time reduction and smaller Hamming distance. However, statistical analysis on the experimental results suggests that both numbers stay rather stable regardless of the percentage of variables we preserve. This is due to the following reason. Although the size of the instances, reflected by the "Var." columns in the table representing the number of variables in the fingerprinted instances, decreases with the percentage of the variables preserved, the structural difficulty of these instances increases. The important observation is that the original instances of the problem have large number of solutions. The difficulty in solving the fingerprinted instances of smaller size actually increases because the preserved variables are selected randomly and many initially feasible solutions become infeasible. Therefore, solvers will spend more time to locate a solution, which reduces the CPU time saving we are expecting from making the size of the formula small.

VI. CONCLUSION

Fingerprinting-based intellectual property (IP) protection mechanism has major advantages over the watermarking-based IP protection because it provides protection to both the buyer and seller. The key problem related to the use of fingerprinting for IP protection is the tradeoff between collusion resilience and the run-time overhead to generate large number of distinct IP instances. We have introduced a new generic fingerprinting technique applicable to the protection of solutions to optimization/decision problems and, therefore, of hardware and software IPs. By judiciously exploiting partial solution reuse and the incremental application of iterative optimizers, we have developed fingerprinting-based IP protection techniques for the problems of partitioning, graph coloring, satisfiability and standard-cell placement. The proposed fingerprinting protocols provide high collusion resilience low run-time overhead simultaneously as we have demonstrated from the experiments.

REFERENCES

[1] C. J. Alpert, "The ISPD-98 circuit benchmark suite," in *Proc. ACM/IEEE International Symposium on Physical Design*, Apr. 98, pp. 80–85.

[2] A. E. Caldwell, H. Choi, A. B. Kahng, S. Mantik, M. Potkonjak, G. Qu, and J. L. Wong, "Effective Iterative Techniques for Fingerprinting Design IP," University of Maryland Institute for Advanced Computer Studies (UMIACS), Technical Report UMIACS-TR-2003-79, July 2003.

[3] I. Biehl and B. Meyer, "Protocols for collusion-secure asymmetric fingerprinting," in *Proc. 14th Annu. Symp. Theoretical Aspect of Computer Science*, 1997, pp. 399–412.

[4] D. Boneh and J. Shaw, "Collusion-secure fingerprinting for digital data," in *Proc. 15th Annu. Int. Cryptology Conf.*, 1995, pp. 452–465.

[5] O. Coudert, "Exact coloring of real-life graphs is easy," in *Proc. 34th Design Automation Conf.*, June 1997, pp. 121–126.

[6] S. Dutt and W. Deng, "VLSI circuit partitioning by cluster-removal using iterative improvement techniques," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1996, pp. 194–200.

[7] C. M. Fiduccia and R. M. Mattheyses, "A linear time heuristic for improving network partitions," in *Proc. ACM/IEEE Design Automation Conf.*, 1982, pp. 175–181.
[8] C. Fleurent and J. A. Ferland, "Generic and hybrid algorithms for graph coloring," *Ann. Oper. Res.*, vol. 63, pp. 437–461, 1996.
[9] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W.H. Freeman, 1979.
[10] M. M. Halldorsson, "A still better performance guarantee for approximate graph coloring," *Inform. Processing Lett.*, vol. 45, no. 1, pp. 19–23, 1995.
[11] I. Hong and M. Potkonjak, "Behavioral synthesis techniques for intellectual property protection," in *Proc. 36th ACM/IEEE Design Automation Conf.*, June 1999, pp. 849–854.
[12] D. S. Johnson *et al.*, "Optimization by simulated annealing: An experimental evaluation II. Graph coloring and number partitioning," *Oper. Res.*, vol. 39, no. 3, pp. 378–406, 1991.
[13] A. B. Kahng, J. Lach, W. H. Mangione-Smith, S. Mantik, I. L. Markov, M. Potkonjak, P. Tucker, H. Wang, and G. Wolfe, "Watermarking techniques for intellectual property protection," in *Proc. ACM/IEEE Design Automation Conf.*, June 1998, pp. 776–781.
[14] A. B. Kahng, S. Mantik, I. L. Markov, M. Potkonjak, P. Tucker, H. Wang, and G. Wolfe, "Robust IP watermarking methodologies for physical design," in *Proc. ACM/IEEE Design Automation Conf.*, June 1998, pp. 782–787.
[15] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Tech. J.*, vol. 49, pp. 291–307, 1970.
[16] D. Kirovski, Y. Hwang, M. Potkonjak, and J. Cong, "Intellectual property protection by watermarking combinational logic synthesis solutions," in *Proc. IEEE/ACM Int. Conf. Computer Aided Design*, 1998.
[17] J. Lach, W. H. Mangione-Smith, and M. Potkonjak, "FPGA fingerprinting techniques for protecting intellectual property," in *Proc. CICC*, 1998.
[18] J. P. M. Silva and K. A. Sakallah, "Boolean satisfiability in electronic design automation," in *37th ACM/IEEE Design Automation Conf.*, June 2000, pp. 675–680.
[19] I. H. Osman and J. P. Kelly, Eds., *Meta-Heuristics: Theory and Applications*: Kluwer, 1996.
[20] B. Pfitzmann and M. Schunter, "Asymmetric fingerprinting," in *Proc. Int. Conf. Theory and Application of Cryptographic Techniques*, 1996, pp. 84–95.
[21] G. Qu and M. Potkonjak, "Analysis of watermarking techniques for graph coloring problem," in *Proc. IEEE/ACM Int. Conf. Computer Aided Design*, Nov. 1998, pp. 190–193.
[22] ——, "Fingerprinting IP's using constraint-addition: Approach and graph coloring case study," in *Proc. 37th ACM/IEEE Design Automation Conf.*, June 2000, pp. 587–592.
[23] G. Qu, J. L. Wong, and M. Potkonjak, "Optimization-intensive watermarking techniques for decision problems," in *Proc. 36th ACM/IEEE Design Automation Conf.*, June 1999, pp. 33–36.
[24] R. H. Storer, S. D. Wu, and R. Vaccari, "New search spaces for sequencing problems with application to job shop scheduling," *Management Science*, vol. 38, pp. 1495–1509, 1992.
[25] *Intellectual Property Protection White Paper: Schemes, Alternatives, and Discussion Version 1.0*, Sept. 2000.
[26] DIMACS Graph Coloring Benchmark [Online]. Available: ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/color/.
[27] SATLIB-The Satisfiability Library [Online]. Available: http://www.satlib.org/.
[28] Graph Coloring Instances [Online]. Available: http://mat.gsia.cmu.edu/COLOR/instances.html

Andrew E. Caldwell is a Member of the Consulting Staff in the X-Architecture Engineering group at Cadence Design Systems, Inc. His research interests include timing and power aware physical design, as well as large-scale combinatorial optimization. He is on leave from the UCLA Ph.D. program in computer science.

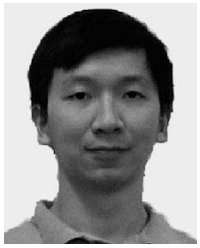
Mr. Caldwell is a member of the ACM.

Hyun-Jin Choi, photograph and biography not available at the time of publication.

Andrew B. Kahng (A'89–M'03) received the A.B. degree in applied mathematics from Harvard College, and the M.S. and Ph.D. degrees in computer science from the University of California, San Diego.

From 1989 to 2000, he was a member of the UCLA computer science faculty. He is Professor of CSE and ECE at UC San Diego. He has published over 200 papers in the VLSI CAD literature, receiving three Best Paper awards and an NSF Young Investigator award. His research is mainly in physical design and performance analysis of VLSI, as well as the VLSI design-manufacturing interface. Other research interests include combinatorial and graph algorithms, and large-scale heuristic global optimization.

Since 1997, Professor Kahng has defined the physical design roadmap for the International Technology Roadmap for Semiconductors (ITRS), and since 2001 has chaired the U.S. and international working groups for Design technology for the ITRS. He has been active in the MARCO Gigascale Silicon Research Center since its inception. He was also the founding General Chair of the ACM/IEEE International Symposium on Physical Design, and co-founded the ACM Workshop on System-Level Interconnect Planning.



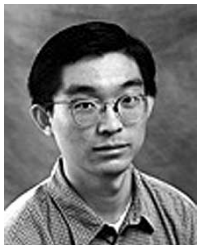
Stefanus Mantik received the B.S. degree in information and computer science from the University of California at Irvine, the M.S. and the Ph.D. degrees in computer science from the University of California, Los Angeles.

He is currently a Senior Member of Technical Staff at Cadence Design Systems, Inc. His research interests include VLSI layout design and optimization.

Miodrag Potkonjak (S'86–M'92) received the Ph.D. degree in electrical engineering and computer science from the University of California, Berkeley, in 1991.

His watermarking-based Intellectual Property Protection research formed a basis for the Virtual Socket Initiative Alliance standard. His research interests include system design, embedded systems, computational security, and intellectual property protection.

In 1991, Dr. Potkonjak joined C&C Research Laboratories, NEC USA, Princeton, NJ. Since 1995, he has been with the Computer Science Department at UCLA. He received the NSF CAREER award, OKAWA foundation award, UCLA TRW SEAS Excellence in Teaching Award and a number of best paper awards.



Gang Qu (S'98–A'00) received the B.S. and M.S. degrees in mathematics from the University of Science and Technology of China in 1992 and 1994, respectively. In 2000, he received the Ph.D. degree in computer science from the University of California, Los Angeles.

In 2000, he joined the Electrical and Computer Engineering Department in the University of Maryland, College Park. He became a member of the University of Maryland Institute of Advanced Computer Studies in 2001. His research interests include intellectual property reuse and protection, low power system design, applied cryptography, and computer-aided synthesis.

Dr. Qu has won several awards including the 36th Design Automation Conference Graduate Scholarship, the Dimitris N. Chorafas Foundation Award, and 2002 George Corcoran Award for Significant Contributions to ECE Education.

Jennifer L. Wong (S'99) received the M.S. degree in computer science and engineering from the University of California, Los Angeles, in 2002. She is working toward the Ph.D. degree at the University of California, Los Angeles.

Her research interests include intellectual property protection, embedded systems, and *ad hoc* sensor networks.