Opportunistic IP Birthmarking using Side Effects of Code Transformations on High-Level Synthesis

Hannah Badier, Christian Pilato, Jean-Christophe Le Lann, Philippe Coussy, Guy Gogniat

Opportunistic IP Birthmarking using Side Effects of Code Transformations on High-Level Synthesis

Hannah Badier*, Christian Pilato†, Jean-Christophe Le Lann*, Philippe Coussy‡, Guy Gogniat‡
*ENSTA Bretagne, Lab-STICC, France, †Politecnico di Milano, Italy, ‡Universite de Bretagne Sud, Lab-STICC, France
{hannah.badier,jean-christophe.le_lann}@ensta-bretagne.org, christian.pilato@polimi.it, {philippe.coussy,guy.gogniat}@univ-ubs.fr

***Abstract*—The increasing design and manufacturing costs are leading to globalize the semiconductor supply chain. However, a malicious attacker can resell a stolen Intellectual Property (IP) core, demanding methods to identify a relationship between a given IP and a potentially fraudulent copy.**

We propose a method to protect IP cores created with high-level synthesis (HLS): our method inserts a discrete *birthmark* in the HLS-generated designs that uses only intrinsic characteristics of the final RTL. The core of our process leverages the side effects of HLS due to specific source-code manipulations, although the method is HLS-tool agnostic. We propose two independent validation metrics, showing that our solution introduces minimal resource and delay overheads ($< 6\%$ and $< 2\%$, respectively) and the accuracy in detecting illegal copies is above 96%.

I. Introduction

Semiconductor design houses are increasingly relying on pre-designed Intellectual Property (IP) blocks, often designed by specialized companies with methods based on high-level synthesis (HLS). HLS starts from a high-level description and automatically creates the corresponding Register-Transfer Level (RTL) description [1]. The IP re-use paradigm, however, introduces security concerns. A rogue employee in the integration company can copy or resell illegal copies, undermining the market of the company designing the IP [2]. IP designers are forced to modify their design with proprietary information to claim IP ownership during litigation. *IP watermarking* is a popular protection method that can be applied in different design stages [3]. Low-level watermarking methods can create issues during physical design [4], while additional functions added for IP watermarking can be detected and removed with re-synthesis attacks [5], nullifying the function of the IP watermark. Reusing part of the IP logic to implement the watermark "signature" incurs high overhead [6].

Another approach uses intrinsic design properties (called *birthmarks*) to detect if two designs share the same source. While birthmarks have been used to detect software theft [7], we investigate how to extend this approach to hardware for semiconductor companies using commercial HLS tools.

We propose to integrate IP birthmarking during HLS. After analyzing the effects of compiler and HLS transformations [8], we observe that *reverting transient HLS transformations is never perfect in practice* [9]. A *transient transformation* is a modification of the C code (i.e., *before* HLS) that is reverted on the resulting RTL design (i.e., *after* HLS). Our process is shown in Fig. 1. We apply a set of transformations to the input C code so that performing HLS on the modified C code produces an RTL design that is affected by these transformations. For example, we introduce additional fake operations controlled by additional inputs that can affect the resource binding. Even if we remove the extra operations after HLS, the resulting RTL contains slight modifications that are impossible to match without replicating exactly the same sequence of transient transformations. These minimal changes constitute the *birthmark* for the IP design. We also propose two metrics to analyze the relationship between two designs. Our birthmarks are based on intrinsic properties of the circuit. So, it cannot be removed with re-synthesis attacks [6], [10]. Also, it is compatible with commercial HLS tools as it operates only on the IP descriptions before and after HLS.

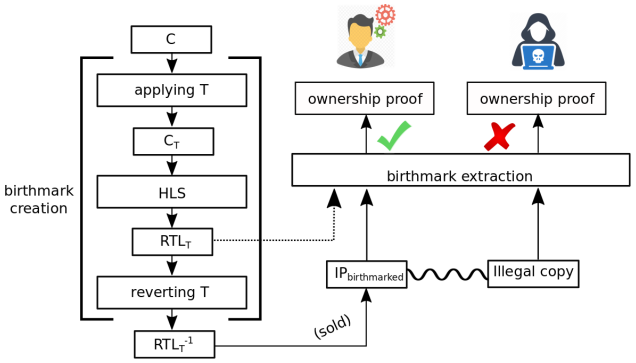


Fig. 1: IP protection with birthmarking based on transient transformations on the input C code.

II. Background

We assume the **attacker** aims at stealing, replicating, and reselling an IP description. The attacker has access to the RTL design (*soft IP*) or can reverse the gate-level netlist (*firm IP*) to extract the corresponding RTL description [11]. The **design house**, instead, aims at protecting its IP by claiming ownership of a potentially stolen design. To create the IPs, the design house uses commercial HLS without security enhancements.

IP birthmarking can suggest whether an IP has been potentially stolen: *if two designs have the same birthmark, one has to be a copy of the other*. Designers already used to identify the illegal copies of software programs [12]. A hardware birthmark must include both functional and structural information about the circuit. *Let* IP_1 *and* IP_2 *be two circuits, and* hbm *a hardware* **birthmark extraction function**. *The function* hbm *is a valid hardware birthmark if* $hbm(IP_1) = hbm(IP_2)$ *only when* IP_1 *and* IP_2 *are copies of each other with a high degree of confidence.* This definition refers to circuit *similarity* and *containment* to differentiate the origin and the copy.

We aim at creating a hardware birthmark by exploiting HLS effects. Indeed, even slight changes in the source code can introduce considerable differences in the RTL design: the HLS process is notably sensitive to **syntactic variance** [13]. This effect is still largely visible even with canonical representations like static single assignment (SSA) and symbolic techniques [14]. Compiler optimizations and their ordering can significantly affect the area and latency of the HLS-generated circuits [8]. Also, the impact of optimization passes is application dependent. Software obfuscation techniques applied to source code can negatively affect HLS quality of results [15].

III. Proposed Approach

Fig. 1 shows our IP birthmarking method. Let C be the input C code to synthesize. To create unique features unknown to the attacker, we borrow the idea of C-based obfuscation [9], [16]. We identify a set T of C-level transformations with parameters known only to the design house. After applying T, we obtain the transformed code C_T that is fed into commercial HLS to obtain the corresponding RTL design RTL_T. After HLS, we obtain the final design $RTL_{T^{-1}}$ by reverting T with a corresponding set of RTL transformations T^{-1}. Designs $RTL_{T^{-1}}$ and RTL (obtained by directly applying HLS to the code C) have the same functionality but the micro-architecture of $RTL_{T^{-1}}$ contains unique features due to the combined effects of code transformations, their parameters, and HLS. We use these micro-architectural differences as *hardware birthmark*.

In case of IP copy, the attacker accesses only $RTL_{T^{-1}}$ (i.e., the sold IP), while the design house can compare this potentially stolen IP with its library of designs obtained before applying the reverse transformations. Verifying the birthmark can be reduced to the following problem: **proving that** $RTL_{T^{-1}}$ ***originates*** **from** RTL_T **allows the design house to claim IP ownership since no one else has access to** RTL_T.

IV. IP Birthmarking Creation

We create our birthmarking by applying transformations on the input C code, using commercial HLS to generate the RTL, and reverting the transformations on the output RTL.

We transform the input C code with key-based control flow splitting, already used for obfuscating software code [17] and logic locking [16]. After parsing, we list all eligible basic blocks, excluding the ones with multiple exit branches. For each basic block to transform, we introduce mutually-exclusive variants preceded by an `if/else` statement having a key-dependent condition. The "bogus blocks" have minor changes in the operators, operands, or instructions order. These modifications aim at uniquely impacting HLS resource binding and scheduling. The transformed C code (C_T) is rebuilt from the modified CFG representation. We use commercial HLS to obtain the corresponding RTL (RTL_T). Its interface has additional inputs for carrying the predicate keys, so HLS will not remove them. After HLS, we revert the transient transformations to 1) get a functionally equivalent design without specifying the predicate keys, and 2) minimize the overhead by removing the extra resources. With an RTL post-processing step, we apply the key values as constants to the extra inputs, propagating the values to existing signals. For example, we remove a multiplexer with a constant selector by connecting the output directly to the proper input. As the values are propagated, we also remove the parts that are no longer used. For example, the other input of the multiplexer previously removed becomes unconnected and we can remove the logic generating this value if it is unused in other parts of the circuit. Differently from [9], we apply RTL simplifications instead of leveraging logic synthesis to remove the extra logic. So, we can analyze the birthmark directly on the RTL design ($RTL_{T^{-1}}$).

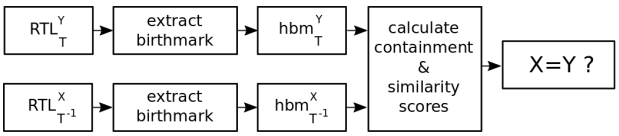


Fig. 2: Birthmark Extraction Flow

V. IP Birthmarking Extraction and Evaluation

A given IP design τ has been allegedly copied from a proprietary design with HLS-based birthmark ($RTL_{T^{-1}}$) when τ and $RTL_{T^{-1}}$ both derive from the same design RTL_T (i.e., same C code, transformations and HLS optimizations) with high structural *similarity*. Moreover, since the transformations from RTL_T to $RTL_{T^{-1}}$ only simplify the design, τ is a copy of $RTL_{T^{-1}}$ if they are both *contained* into RTL_T.

We propose two birthmarking metrics to prove this relationship (see Fig. 2). Each metric has specific definitions of *similarity* (SIM) and *containment* (CNT). Let X and Y be two designs and hbm a birthmark metric. Containment is defined as $\Gamma(X,Y) = CNT(hbm(X), hbm(Y))$ while similarity Σ is defined as $\Sigma(X,Y) = SIM(hbm(X), hbm(Y))$. So, τ is a copy of $RTL_{T^{-1}}$ if and only if

$$\Gamma(\tau, RTL^T) = 100\%$$
$$\forall i \in \mathbb{N} : \Sigma(\tau, RTL^T) \gg \Sigma(\tau, RTL_i^T)$$

i.e. τ is completely contained into RTL_T and is much more similar to RTL_T than to any other candidate design RTL_i^T. We should detect only direct IP copies (**Credibility** property), and not designs that originate from the same original C code but were transformed or synthesized differently. Our technique should match the two designs even if the attackers adds their own modifications (**Resilience** property).

A. Dataflow-based Birthmark

The dataflow graph (DFG) of RTL_T contains extra nodes and edges due to the "bogus code". After reversing the transformations, the DFG of $RTL_{T^{-1}}$ is different from the RTL generated directly from the input code. Let $DFG = (V, E)$ be the DFG of an RTL design, where V is set of circuit elements and E the set of connections. Let DFG_{T_i} be the DFG of the design transformed with key parameters i, and $DFG_{T_i}^{-1}$ the DFG of the design after reverting the transformations. Reverting the transformations does not add vertices. However, HLS choices may impact the edges because of changes in the interconnections. So, $DFG_{T_i}^{-1}$ should be *similar* to but also *contained* into DFG_{T_i}.

Since comparing two graphs is NP-complete, we approximate similarity and containment metrics on the two DFGs. We focus on individual paths from inputs to outputs. We assign a specific letter to each operator, register, and signal to convert each input/output path into a string [18]. So, we compute *string similarity*, which is a well-known problem in bioinformatics and natural language processing.

1) Similarity: Given two RTL designs X and Y, we compute the corresponding DFGs and in turn the set of strings S^X and S^Y. For every pair of strings, we compute the similarity metric $SIM(s^x, s^y)$, obtaining $|S^X| \times |S^Y|$ values. For the similarity of two strings, we use Levenshtein distance and Longest Common Subsequence (LCS). We select the T values with highest scores, where $T = min(|S^X|, |S^Y|)$, and we average them to obtain the final similarity score $\Sigma(X, Y)$.

2) Containment: The containment score is computed with the same procedure, except for the containment metric between two strings $CNT(s^x, s^y)$. We compute the metric as the number of letters in s^x that are present (in the same order) also in s^y. This metric is not commutative: $CNT(s^x, s^y) == CNT(s^y, s^x)$ only when the two strings are identical.

B. Scheduling-based Birthmark

We also evaluate the HLS effects on operation scheduling. The transformations T force the HLS tool to change binding or execution order to optimize the corresponding RTL_T, potentially leading to different schedules. The final RTL reflects these differences since reverting the transformations removes extra logic without adding operation or modifying the scheduling. Let $RTL^X_{T_i}$ and $RTL^Y_{T_i}$ be two designs obtained from the same code C but with different transformations T_i^X and T_j^Y. Any changes in the scheduling lead to different birthmarks.

1) Similarity: Given two designs RTL^X and RTL^Y, we average the similarity scores between each pair of states:

$$SIM(S^X, S^Y) = \frac{\sum_{j=1}^{R} SIM(s_j^X, s_j^Y)}{R}$$
$$where\ R = \max(R^X, R^Y) \quad (1)$$

where R^i is the number of RTL states. If one design has less states, we pad it out with empty states. We use operation types for comparing the states instead of RTL names. We use Jaccard coefficient on the operation sets to compute similarity [19]:

$$SIM(s^x, s^y) = \frac{|OP(s^X) \cap OP(s^Y)|}{|OP(s^X) \cup OP(s^Y)|} \quad (2)$$

where $OP(s^i)$ contains the operation types in state s^i. $OP(s^i)$ is a *multiset* since it can contain multiple instances of the same operation type. Similarity of two states/designs is symmetric.

2) Containment: We compute the containment score by averaging the containment metric of the states. However, this metric aims at determining whether one state (schedule) is *contained* into another (i.e., it has equal or less operations). So, the containment metric of two states is defined as:

$$CNT(s^x, s^y) = \frac{|OP(s^x) \cap OP(s^y)|}{|OP(s^x)|} \quad (3)$$

$CNT(s^x, s^y)$ and the containment property are not symmetric.

VI. Experimental Evaluation

We tested our method with five randomly-generated C designs and two real benchmarks. We used Crokus[1] to generate the random programs, while the real benchmarks are from CHStone and MachSuite. We used Python tools to transform the C code based on input parameters and revert the transformations with the same key values. HLS is performed with Xilinx Vivado HLS. We used Yosys to extract the DFG of an RTL design, and the Networkx Python library to extract the shortest path from each input-output pair. We used a pre-defined alphabet to represent them and a combination of Python libraries and custom scripts to compute the metrics on them.

We composed the dataset by applying C (i.e., combinations of transformation and parameters) to D designs. We obtained $M = D \cdot C$ designs that we synthesized and reverted. The pairs of designs before and after reverting the transformations are *positive pairs* (P) because our birthmarking extraction process should confirm the relationship. Our metrics are expected to show no relationships for any other pair (*negative pairs*). We used 15 configurations for each random program and 3 for each real benchmark. So, we obtain 75 positive and 5,550 negative pairs for the synthetic benchmarks, along with 15 positive and 210 negative pairs for each real benchmark. In all cases, we apply transformations to at least 50% of the basic blocks.

Classification Accuracy. We use the similarity and containment scores as binary classifiers with two thresholds k_s and k_c, respectively. Above the threshold, the pair is classified as positive. To evaluate prediction accuracy, we use traditional classification metrics: *recall*, *precision*, and *balanced accuracy*.

Fig. 2a and Fig. 2b show the results for the metric scores in dataflow- and scheduling-based birthmarking, respectively. In all cases, *the average score for positive pairs (blue bars) is significantly higher than the average for negative pairs (red bars)*. However, only for the scheduling scores there is a clear distinction between the values for positive and negative pairs. So, a threshold value in between allows us to correctly classify the pairs. Conversely, dataflow scores are partially overlapping, leading to incorrect classifications depending on the threshold.

Table I reports more details on correct/incorrect classifications and the corresponding metrics. While *scheduling scores lead to perfect classifications in all cases*, the balanced accuracy for dataflow-based containment is the lowest (75.21%), while LCS similarity outperforms Levenshtein distance. In all three cases, however, we have high false positives.

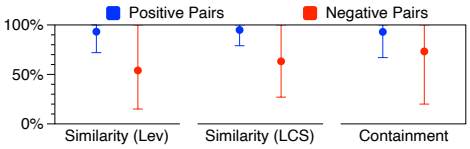
Credibility. We compare designs with the same input C code but different transformations. All metrics have significantly lower average scores (below 50%) and the pairs are correctly classified as negatives. The classification improves when increasing the number of transformations.

Resilience. We tested our metrics against design modifications by randomly adding a 5% of random "noise" operators in the final RTL designs. We tested only the scheduling metrics as they perform better. Scores and classification results are

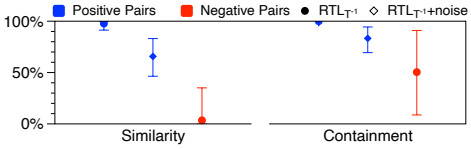
[1] https://github.com/JC-LL/crokus

TABLE I: Results with dataflow containment and similarities as classifiers. T: threshold, Lev: Levenshtein distance, LCS: Longest Common Subsequence.

	Dataflow-based birthmark			**Scheduling-based birthmark**			
	Similarity		Contain.	Similarity		Contain.	
	(Lev.)	(LCS)			(noise)		(noise)
	T = 0.81	T = 0.82	T = 0.94	T = 0.36	T = 0.36	T = 0.92	T = 0.75
True Positive (TP)	50	55	44	75	75	75	74
False Negative (FN)	6	1	12	0	0	0	1
True Negative (TN)	2,507	2,252	2,213	5,550	5,550	5,550	5,248
False Positive (FP)	573	828	867	0	0	0	302
Precision	**8.00%**	**6.20%**	**4.80%**	**100.00%**	**100.00%**	**100.00%**	**19.70%**
Recall	**89.30%**	**98.20%**	**78.60%**	**100.00%**	**100.00%**	**100.00%**	**98.70%**
Balanced accuracy	**85.34%**	**85.67%**	**75.21%**	**100.00%**	**100.00%**	**100.00%**	**96.61%**



(a) Dataflow-based birthmark.



(b) Scheduling-based birthmark and noise.

Fig. 3: Average-Max-Min graphs.

reported in Fig. 2a, Fig. 2b, and Table I (label *noise*). Final RTL designs ($RTL_{T^{-1}}$) are no longer completely contained into transformed RTL designs (RTL_T) but *metric scores for positive pairs are still higher than the ones for negative pairs, allowing a correct classification of most designs*.

Real Benchmarks and Overhead. We tested our approach on ADPCM and AES only for the scheduling-based birthmarking. For both benchmarks, *average scores are above 90% for positive pairs and below 45% for negative pairs*. Similarity metrics outperform containment ones. *The balanced accuracy is around 93% for ADPCM (with some false positives but no false negatives) and above 99% for AES*. Comparing the RTL designs with and without our birthmarks, *resource overhead was below 6%, while delay overhead was under 2%*.

VII. Related Work

Obfuscation thwarts reverse engineering by locking the IP functionality with a proper key. It can be performed at logic [20] and behavioral levels [21]. HLS-based locking requires custom EDA flows [16]. Watermarking aims at certifying the ownership of an IP after chip fabrication [3]. Circuit watermarking can be achieved by embedding secret information into the IP test circuit [22], the input and output signals [4], or in the finite state machines. They can be invalidated with resynthesis or removal attacks [5]. Birthmarks don't use any extra functionality, nullifying such attacks. Identifying illegal copies at behavioral RTL is similar to determine if two programs share a common origin [12]. Software birthmarking principles were first used to detect the theft of Java programs [7]. Hardware birthmarking was introduced to suggest similar designs for reusability [18]. We borrow the key idea to identify whether an IP has been illegally copied. High-level security techniques can bring significant advantages [23] but may require custom tools [6]. Our approach is compatible with commercial HLS.

VIII. Concluding Remarks

This paper aims at identifying illegal IP copies for HLS-generated designs. It leverages the side effects of *transient transformations* on the HLS input code. Our method is compatible with commercial HLS tools and creates IP birthmarks that are hard to replicate without knowing the sequence of optimizations and parameters. Extensive analysis on synthetic and real benchmarks confirm we can identify IP relationships with an accuracy above 96% (100% for similarity metrics) with less than 6% resource overhead.

References

[1] R. Nane *et al.*, "A survey and evaluation of FPGA high-level synthesis tools," *IEEE TCAD*, vol. 35, no. 10, 2016.

[2] U. Guin *et al.*, "Counterfeit Integrated Circuits: A rising threat in the global semiconductor supply chain," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1207–1228, Aug. 2014.

[3] A. T. Abdel-Hamid *et al.*, "A survey on IP watermarking techniques," *Design Automation for Embedded Syst.*, vol. 9, no. 3, pp. 211–227, 2004.

[4] B. Le Gal and L. Bossuet, "Automatic low-cost IP watermarking technique based on output mark insertions," *DAES*, vol. 16, no. 2, 2012.

[5] A. Cui *et al.*, "A robust FSM watermarking scheme for IP protection of sequential circuit design," *IEEE TCAD*, vol. 30, no. 5, 2011.

[6] C. Pilato *et al.*, "High-level synthesis of benevolent Trojans," in *Proc. of DATE*, 2019, pp. 1124–1129.

[7] G. Myles and C. Collberg, "Detecting software theft via whole program path birthmarks," in *Proc. of ISC*, 2004, pp. 404–415.

[8] Q. Huang *et al.*, "The effect of compiler optimizations on high-level synthesis for FPGAs no," in *Proc. of FCCM*, 2013, pp. 89–96.

[9] H. Badier *et al.*, "Transient key-based obfuscation for HLS in an untrusted cloud environment," in *Proc. of DATE*, 2019, pp. 1118–1123.

[10] A. T. Abdel-Hamid *et al.*, "IP watermarking techniques: Survey and comparison," in *Proc. of IWSOC*, 2003, pp. 60–65.

[11] J. Rajendran *et al.*, "Belling the CAD: Toward security-centric electronic system design," *IEEE TCAD*, vol. 34, no. 11, pp. 1756–1769, 2015.

[12] D. Grover, "Program identification," in *The protection of computer software–its technology and applications*. Cambridge Univ. Press, 1989.

[13] V. Chaiyakul *et al.*, "High-level transformations for minimizing syntactic variances," in *Proc. of DAC*, 1993, pp. 413–418.

[14] A. A. Kountouris and C. Wolinski, "Efficient scheduling of conditional behaviors for high-level synthesis," *ACM TODAES*, vol. 7, no. 3, pp. 380–412, 2002.

[15] N. Veeranna and B. C. Schafer, "Efficient behavioral intellectual properties source code obfuscation for HLS," in *Proc. of LATS*, pp. 1–6.

[16] C. Pilato *et al.*, "TAO: Techniques for algorithm-level obfuscation during high-level synthesis," in *Proc. of DAC*, 2018, pp. 1–6.

[17] C. Collberg *et al.*, "A taxonomy of obfuscating transformations," The University of Auckland, New Zealand, Tech. Rep., 1997.

[18] K. Zeng and P. Athanas, "Discovering reusable hardware using birthmarking techniques," in *Proc. of IRI*, 2015, pp. 106–113.

[19] A. Z. Broder, "On the resemblance and containment of documents," in *Proc. of SEQUENCES*, 1997.

[20] J. Roy, F. Koushanfar, and I. Markov, "EPIC: Ending piracy of integrated circuits," in *Proc. of DATE*, 2008, pp. 1069–1074.

[21] F. Koushanfar, "Provably secure active IC metering techniques for piracy avoidance and digital rights management," *IEEE Trans. on Information Forensics and Security*, vol. 7, no. 1, pp. 51–63, Feb 2012.

[22] Y. C. Fan and H. W. Tsao, "Watermarking for intellectual property protection," *Electronics Letters*, vol. 39, no. 18, pp. 1316–1318, 2003.

[23] C. Pilato *et al.*, "Securing hardware accelerators: A new challenge for high-level synthesis," *IEEE ESL*, vol. 10, no. 3, 2018.