Netlist-Level IP Protection by Watermarking for LUT-Based FPGAs

Moritz Schmid, Daniel Ziener and Jürgen Teich

Hardware/Software Co-Design
Department of Computer Science
University of Erlangen-Nuremberg, Germany
email: {daniel.ziener, teich}@cs.fau.de

Abstract

This paper presents a novel approach to watermark FPGA designs on the netlist level. We restrict the dynamically addressable part of the logic table, thus freeing space for insertion of signature bits into lookup tables (LUTs). In this way, we tightly integrate the watermark with the design so that simply removing mark carrying components would damage the intellectual property core. Converting functional LUTs to LUT-based RAMs or shift registers prevents deletion due to optimization. With this technique, we take watermark carrying components out of the scope of optimization algorithms to achieve complete transparency towards development environments. We can extract the marks from the bitfile of an FPGA. The method was tested on a Xilinx Virtex-II Pro FPGA and showed low overhead in terms of timing and resources at a reasonable number of watermarked cells.

1. Introduction

Recent years have shown an increasing demand for faster development of more and more complex electronic designs. A well accepted strategy is to use intellectual property (IP) cores in designs to aid an accelerated development of products. IP cores are mostly provided in a modular fashion. It has become very simple to copy and resell an IP core without even understanding how it works. IP protection is often presented as countermeasure against piracy of IP cores. It is grouped by the VSI Alliance IP protection development working group into three main approaches; deterrent, protection, and detection [6]. Watermarks are a suitable instrument to allow for detection of copyright infringement. Using steganographic methods, authors can insert their copyright into IP cores, thus enabling identification of these cores.

A design for an FPGA is usually developed according to the well known development flow, as it is shown on the left in Figure 1. Digital watermarks can be inserted at each level, the design resembles an individual core. To preserve maximum industry acceptance, it is necessary to keep the IP core as versatile as possible, and to provide independence of specific target hardware.

We present a method, to protect IP cores at the netlist level. Signature dependent watermarks are inserted into EDIF netlists to obtain watermarked netlists, which can then be distributed as protected IP cores (see Figure 1). The protected IP core can be used without restriction in designs. If a product is suspected to contain an unlicensed protected IP core, the owner can extract the bitfile by wire-tapping the communication between the PROM and the SRAM based FPGA. The watermark can verified by extracting LUT contents from the obtained bitfile and comparing them to the inserted watermarks. We have to note, that this scheme will only work, if the bitfile was not encrypted.

This paper is organized as follows: Section 2 discusses previous approaches to watermarking. Section 3 lays out a theoretical foundation about threats to watermarking for FPGAs and provides definitions to be met by a watermarking scheme to be secure. Section 4 discusses our overall approach to insert watermarks at the netlist level. Section 5 presents experimental results, and section 6 concludes the document.

2. Related Work

Most methods for watermarking IP cores focus on either introducing additional constraints on certain parts of the solution space of synthesis and optimization algorithms, or adding redundancies to the design.

Constraint-based watermarking was first introduced in [9]. It encodes an author's signature into an optimization or synthesis problem, by limiting the overall solutions space

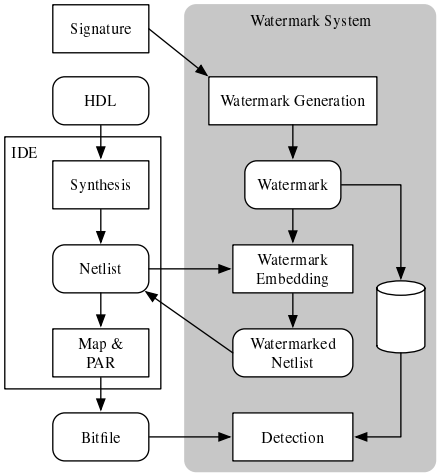

Figure 1. Overview of the proposed watermarking method. The FPGA development flow is shown on the left. On the right, the proposed flow for embedding watermarks is depicted.

to a certain area reflecting the given signature. Approaches include only allowing a certain number of inputs to a gate [14], modification of register ordering by graph coloring [7], imposing timing constraints on nets to achieve a distinct signature in [8], and only allowing a set of standard cells the design is mapped onto [10].

A very important aspect of watermarking is verification of the inserted marks. The major drawback of the above presented approaches is the limitation on verification possibilities of the watermarked core. An ideal watermarking strategy only requires the given product to verify inserted watermarks. Our method was developed focussing on the verification of the watermarks. There are four potential sources of information: bitfile, ports, power, and electromagnetic radiation.

The fundamental idea of additive watermarking is to add something to a design, that would not be present normally, yet is hard to detect and would ideally damage the design, if removed. Approaches propose hiding signature bits in unused LUTs by altering bitfiles [9] [11][15], or modifying LUTs on HDL level [4]. Here, the verification is done using the bitfile. Others add extra circuits to the design, making it possible to obtain the included signature at the output ports of the FPGA by triggering the added circuit through feeding a special signal sequence to the inputs [3], or detect a watermark solely probing the power pins of the device [18].

In [16], the authors propose to use the content of the lookup tables in an FPGA to show whether a core is included in the FPGA design. An overview and evaluation of existing watermark techniques is also given in [17].

3. Theoretical Foundation

Security issues of watermarking applications are often stated using natural language, which usually results in imprecise definitions. Li et. al. present a very simplistic and general, yet mathematically precise approach on the security issues of watermarking for digital multimedia in [13]. Here, we use their definitions to develop a precise model for watermarking FPGA designs on the netlist level.

3.1 Definitions

Logic designs for FPGAs usually go through the well known development cycle from specification in HDL to ready-to-use bitfile. We define the steps of this cycle as different abstraction levels, a work can be specified in. Furthermore, we define the overall process as a series of transformations from one abstraction level to another. An overview is given in Figure 2.

Denote a piece of IP specified on abstraction level A as a work $I_A = (x_{A_1}, \ldots, x_{A_k})$, where each $x_{A_i} \in \mathcal{U}_A$ is an element of the work, resided in a universe inherent to the abstraction level. $\mathcal{T}(\cdot)$ is an efficient transformation algorithm, capable of transforming a work of a specific abstraction level into a work of another abstraction level. Transformations from higher to lower abstraction levels are denoted by $\mathcal{T}_{Y \rightarrow Z}(\cdot)$. These are common to the development flow. The opposite direction, from lower to higher abstraction levels, as in $\mathcal{T}_{Y \leftarrow Z}(\cdot)$, can be achieved by reverse engineering. Reverse engineering transformations are not available, but have to be especially developed.

A key K, is a sequence of m binary bits, i.e. $K = \{0, \quad 1\}^m$. A watermark W_A, applicable to a work of abstraction level A, is defined as $W_A = (w_{A_1}, w_{A_2}, \ldots, w_{A_l})$, where each element $w_{A_i} \in (domain(W_A) := \mathcal{W}_A)$. The watermark has to be embedded into a work I_A, so the domain $\mathcal{W}_A$ of the elements w_{A_i} of W_A is dependent on both the abstraction level of I_A, and the watermark generation process. In order to compare two works of equal abstraction level, we define a distance function $Dist(\cdot, \cdot)$. If the distance between two works I_A and I'_A of the same technology level A is less than a specific threshold t ($Dist(I_A, I'_A) < t$), the two works are of similar quality in terms of electronic designs, i.e. functionality, efficiency, economic value, etc.

Our watermarking scheme for FPGA designs consists of three algorithms, a watermark generator $\mathcal{G}$, a watermark embedder $\mathcal{E}$, and a watermark detector $\mathcal{D}$. Each of these has to be fitted to the technology level it will be applied at. The generator $\mathcal{G}_A$ generates a watermark W_A according to some key K and the work I_A ($W_A = \mathcal{G}_A(K, I_A)$). The watermark is embedded into the work I_A by the embedding algorithm, creating a watermarked work $\widetilde{I_A} = \mathcal{E}_A(I_A, W_A)$.

There should not be any perceivable difference between I_A and $\widetilde{I}_A$, hence $Dist(I_A, \widetilde{I}_A) < t$. The detector receives as input a watermark W_A and a work I_A. It is capable of efficiently determining whether W_A is contained in I_A. The output of the detector is $\mathcal{D}_A(I_A, W_A) = true$, if W_A is contained in the work, and $\mathcal{D}_A(I_A, W_A) = false$ if it is not.

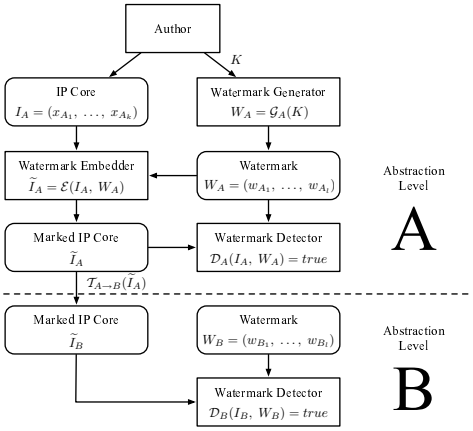

Figure 2. Overview of the watermarking procedure with explicit characterization of abstraction levels.

In order to achieve full transparency of the watermarking process towards development tools, it is an essential requirement, that a work, marked on any abstraction level, will retain the watermark, if transformed to a lower abstraction level. Hence, if $\mathcal{D}_Y(\widetilde{I}_Y, W_Y) = true$, so should $\mathcal{D}(\widetilde{I}_Z, W_Z) = true$, if $\widetilde{I}_Z = \mathcal{T}_{Y \to Z}(\widetilde{I}_Y)$, and W_Z is a representation of W_Y on abstraction level Z.

3.2 Threat Model for Watermarking FPGA Designs

A threat model consists of security goals, threats, and attacks. A threat is a potential event, or a sequence of events, that might lead to a violation of one or more security goals. The actual realization of a threat is called an attack. From an economical perspective, it can be considered a matter of cost and availability, whether a design will be purchased or developed from scratch. Setting business ethics aside, it is also a question, whether it is an option to obtain a valuable IP core and develop an attack capable of removing any authorship ensuring mechanisms. The overall security goal for watermarking schemes is the proof of authorship, its main threats are ownership deadlock, counterfeit ownership, theft [13], and forged authorship. These threats can be achieved by copy, removal, and ambiguity attacks, as it can be seen in Figure 3.

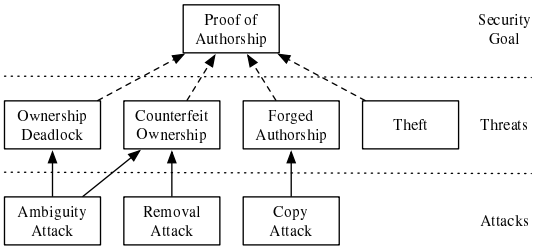

Figure 3. Attacks and threats to interfere with the proof of authorship.

In case of an ownership deadlock, an attacker is successful in threatening the proof of authorship by forcing the decision process of determining the author of a piece of IP into a deadlock. This can be accomplished, if an attacker can present his own watermark in the work, thus creating an ambiguity. If the watermark of the attacker can be considered a stronger proof than the watermark of the original owner, the ambiguity attack can implement counterfeit ownership. Another possibility for achieving this threat is, if an attacker is capable of removing the original watermark form the work by means of a removal attack. If the watermark of a credible author is used by an attacker to *sign* a piece of IP of possibly less quality than one would expect from the feigned author, the proof of authorship is threatened by forged authorship, using a copy attack. Theft is the case if an attacker is successful in presenting a stolen work as his own, where it is assumed, that the original author does not take part in the dispute.

Copy attacks target the key used to generate the watermarks. In [5], the authors present a simple method on how to use RSA to prevent signatures from being copied.

For multimedia works, it is generally required, that it has to be computationally infeasible (c.i.) to render the watermark scheme useless by any of the attacks defined above, with out reducing the work in quality. In case of electronic designs, the c.i. requirement does not necessarily need to hold. Instead, it is sufficient, if the costs for obtaining a watermarked version of a work and the development of an appropriate attacker, are greater than the overall cost for development or purchase of a design of equal abilities.

We define an attacker $\mathcal{A}$, capable of of efficiently transforming a protected work $\widetilde{I}$, into an unprotected work I ($\mathcal{A}(\widetilde{I}) = I$), as an algorithm of polynomial complexity. Let $C(\cdot)$ be a cost evaluation function to describe costs of purchase, denoted as $C_P(\cdot)$, development, denoted as $C_D(\cdot)$, and obtaining, $C_O(\cdot)$. The cost for obtaining a design may vary between copying the design from an arbitrary source and purchasing it. Instead of the c.i. requirement, it is

enough to fulfill

$$C_P(\widetilde{I}) < C_D(I) < (C_O(\widetilde{I}) + C_D(\mathcal{A})). \quad (1)$$

A watermark scheme for FPGA designs can be defined as resistant against removal attacks for any watermarked work $\widetilde{I}_A$ of a given abstraction level A, if it is of more cost to develop an appropriate attacker $\mathcal{A}$, for which it is not c.i. to compute a work $I'_A = \mathcal{A}(\ \widetilde{I}_A\ )$, with $Dist(\ \widetilde{I}_A\ ,\ I'_A\ ) < t$, and $\mathcal{D}(I'_A, W_A) = false$, than legal use would be.

To give an example, first of all, it should obviously be of less cost for a customer, to purchase a watermarked FPGA design or IP core in form of a netlist ($\widetilde{I}_N$) than to redevelop the design. Purchasing or redeveloping the design, should be of less cost than developing an attacker $\mathcal{A}$, capable of computing a work I_N from any $\widetilde{I}_N$ ($\mathcal{A}(\ \widetilde{I}_N\ ) = I_N$), for which $\mathcal{D}(I_N, W_N) = false$, and $Dist(\widetilde{I}_N, I_N) < t$. Such an attacker might be in the form of a transformation from netlist to HDL level ($\widetilde{I}_H = \mathcal{T}_{H \leftarrow N}(\widetilde{I}_N)$), that can compute a work for which it is possible to remove the watermarks, causing detection attempts after another transformation $I'_N = \mathcal{T}_{H \rightarrow N}(I'_H)$, to result in $false$.

Finally, a watermark scheme is resistant to ambiguity attacks for any watermarked work $\widetilde{I}_A$ of a given abstraction level A, if it is of more cost to develop an appropriate attacker $\mathcal{A}$ for which it is not c.i. to compute a watermark W'_A, with $Dist(\ \widetilde{I}_A\ ,\ I'_A\ ) < t$, and $\mathcal{D}(\widetilde{I}_A, W'_A) = true$, than legal use would be.

4. Concept of Watermarking for Netlists

Besides extractability, another of our top concerns for watermarking is transparency towards design tools. These tools usually contain mechanisms to improve a given circuit to achieve optimal placement and routing of logic on specific hardware. Xilinx CAD tools, for example, perform a global optimization step right before mapping the design to the FPGA, which will remove any logic, the algorithm determines to be redundant.

Most additive watermarking methods for netlists watermark the design by introducing redundant logic to the circuit, which will not alter its functionality. In case of IP cores distributed as a netlist, redundant logic might be removed by global optimization. It is necessary to take watermark carrying components out of the scope of optimization algorithms, so that they will pass unchanged.

4.1 LUT to Register Conversion Approach

Logic elements (LEs) on Xilinx FPGAs, that are used as LUTs can also be used as either shift register LUTs (SRLs) or distributed RAM (as an example, see [2] for Xilinx Virtex-II and Virtex-II Pro devices). A LUT can be considered as a RAM, with the inputs specifying the address of the bit to read from the stored value. If used as a LUT, the design tools interpret the stored value as a logic function, which is therefore subject to optimization. If the component is specified, for example, as a shift register, the optimization tools loose the context of it being a logic function, and the component will remain unchanged.

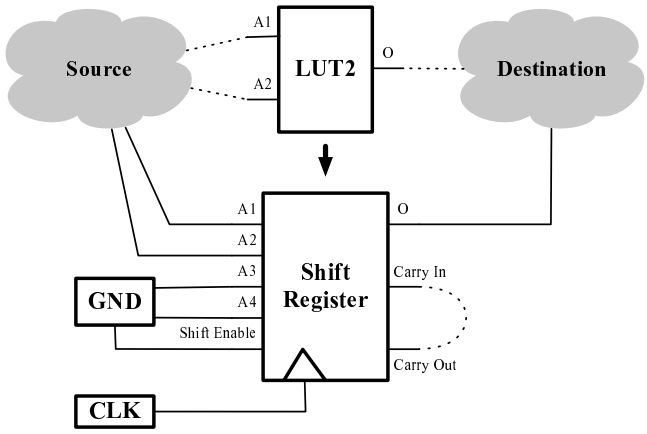

Figure 4. Example showing how to replace a LUT2 by a shift register for Xilinx devices. The schematics represents the logic blocks as seen from a netlist point of view.

EDIF netlists define LUT-cells with an arbitrary number of inputs. Replacing a LUT with, for example, only 2 inputs in a netlist with a register having 4 inputs, as shown in Figure 4, the actual logic table will take up only a small portion of the overall available table. Setting the remaining inputs after conversion to constant signals, e.g. a constant low signal, it is possible to restrict the dynamically addressable part of the logic table to just the part that is needed to fulfill the intended logic function. This frees the rest of the table for insertion of watermarks.

We demonstrate this idea in Figure 5. A two-input boolean *AND* function is implemented using a shift register. Dynamically addressable storage is restricted by connecting the unused inputs A3 and A4 to a static low signal.

Let $Ext_A(I_A) = \{x_{A_1}, \ldots, x_{A_k}\}$ denote an extractor function, capable of extracting the content of a work I_A on a given abstraction level A, to elements x_{A_i} of the set of all contents of the work on the same technology level.

The work to be watermarked is denoted by I_N, the subset of markable contents can be obtained by the extractor, $Ext_N(I_N) = (x_{N_1}, \ \ldots, \ x_{N_k})$, N representing the abstraction level netlist. The elements of the vector, x_{N_i}, are of the type EDIF cell. Markable are

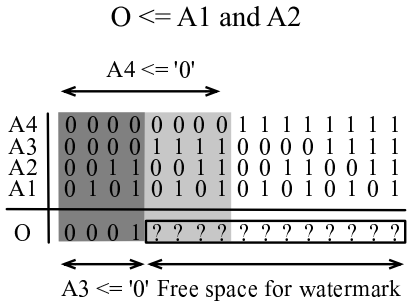

Figure 5. Example of implementing a two input AND function using an SRL. Addressable storage is restricted by connecting inputs A3 and A4 to static low signals.

LUT cells which will be converted to shift register cells. $LUT_N = (lut_{N_1}, \ldots, lut_{N_m})$ denotes a subset of I_N, containing only LUT cells. The watermark generator needs information about the LUTs to be converted, alongside the key to generate the watermarks. The watermark, $W_N = (srl_{N_1}, \ldots, srl_{N_n})$, is created by a generator $\mathcal{G}(K, LUT_N)$. The marks are inserted by replacing the chosen subset of the set suitable cells $LUT_N = (lut_{N_1}, \ldots, lut_{N_m})$ with the set of watermark cells $W_N = (srl_{N_1}, \ldots, srl_{N_n})$, so that $Dist(I_N, \widetilde{I}_N) < t$. This is done by an embedding algorithm, which results in the watermarked work $\widetilde{I}_N = \mathcal{E}(I_N, W_N)$.

The inserted watermarks are to be extracted from a bitfile by exhaustive search on the memory contents of the LEs. Without any additional knowledge, the only information available are the watermarks themselves, for which it is a priori impossible to know the final locations in the bitfile. By interconnecting the mark-carrying registers using carry ports, it is possible to force neighboring placements on the FPGA, thus enabling the detector to search for values in relative close proximity.

The detector algorithm $\mathcal{D}_N$, receives a potentially watermarked work $\widehat{I}_B$ on the abstraction level bitfile, denoted by subscript B. It was transformed by another party, using a transformation algorithm $\mathcal{T}_{N\to B}(\widetilde{I}_N, \{I'_{N_1}, \ldots, I'_{N_k}\}) = \widehat{I}_B$, to combine the watermarked work and possible other works of the same abstraction level. The other works are denoted by the set $\{I'_{N_1}, \ldots, I'_{N_k}\}$. The approach used by $\mathcal{D}_N$ can be described as follows. Let $Dec_{Type, Y\leftarrow X}(\{x_{X_1}, \ldots, x_{X_k}\}) = \{x_{Y_1}, \ldots, x_{Y_k}\}$ be a decoding function, decoding elements of a lower technology level X to a higher technology level Y. The decoding is performed according to a decoding rule, defined by $Type$. The detector algorithm can then be described as $\mathcal{D}_N(W_N, Dec_{SRL,N\leftarrow B}(Ext_B(I_B)))$. After having performed the necessary operations to make the elements of the given work comparable to the elements of the watermark, the detector algorithm can decide on whether the marks can be considered as contained in the given work, denoted by $\mathcal{D}_N(W_N, Dec_{SRL,N\leftarrow B}(Ext_B(I_B))) = true$, and $false$ otherwise. An overview is provided in Figure 6.

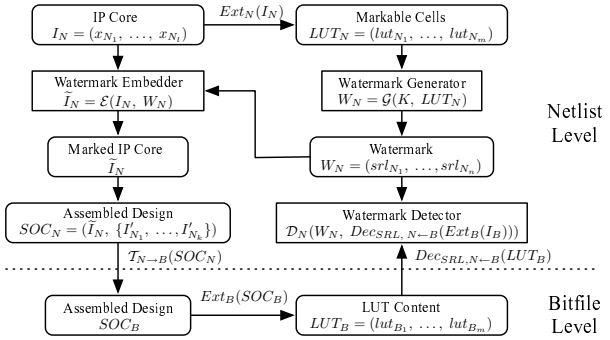

Figure 6. Overview of the proposed watermarking technique in terms of the theoretical definitions.

4.2 Security Analysis

Due to the fact, that the inserted watermarks are very difficult to detect on the bitfile level, we assume, that removal attacks will most likely be attempted on the netlist level. If an attacker is able to decide, which of the contained shift registers in the marked netlist was converted from a LUT, it is rather easy to reverse the process. The complexity of identifying converted LUTs can be increased by constantly changing the ports for the logic function, as well as installing bogus constant signal generating cells. Furthermore, identifiers of components in the netlist can be scrambled to decrease human readability.

Ambiguity attacks are necessary, if the watermark was not detected by a pirate, and an author tries to proof his authorship. Here, it involves finding a fake watermark in either the netlist or the bitfile. The chance of finding such a watermark in the netlist was not further explored. We assume, that an attacker can only create an ambiguity, using the exact same values from the bitfile, as are used by the extraction program to identify the watermark. The effort necessary to construct such a forged watermark can be compared to breaking the mechanisms, the watermarks were created by. Using cryptography, this can be hardened to a very high extent. Furthermore, it was proposed that a low false positive rate is the key to non-invertible watermark schemes, thus resistance to ambiguity attacks [12]. Our approach is compatible to this watermarking scheme. We have enabled extraction of individual watermark chains, instead of all inserted watermarks. This increases the chance of a success-

ful proof of authorship.

5 Experimental Results

The approach was implemented for the Xilinx Virtex-II Pro FPGA XC2VP7, featuring 4.928 slices [2]. We have evaluated the proposed method with respect to timing and resource overhead, by analysis of three public domain cores: Cordic, DES56, and RSA [1]. The designs were implemented on the FPGA using a balanced strategy. Resources are measured in terms of utilized slices, timing is reflected by minimum clock cycle duration. Properties of the unmarked cores are shown in Table 1.

Core	Timing (ns)	Resources (Slices)
Cordic	5.909	369
DES56	6.220	965
RSA	13.557	540

Table 1. Critical path delay (Timing) and used slices (Resources) of the unmarked cores, synthesized for the Xilinx Virtex-II Pro FPGA.

5.1 Effect of Amount of Watermarks on Timing and Resources at Fixed Interconnection Length

#SRLs	32	64	128	256	384
Cordic					
% Timing	39.19	32.53	64.05	77.81	78.76
% Resources	4.97	9.39	17.98	35.23	52.57
DES56					
% Timing	10.34	6.98	24.175	18.708	29.786
% Resources	-1.28	0.28	0.48	-0.17	-3.83
RSA					
% Timing	10.11	55.42	57.20	29.93	44.37
% Resources	-0.74	2.78	1.91	9.07	9.75

Table 2. Effect of number of watermarks on timing and resources at a fixed I/L of 4.

In this test, we measure the impact of various numbers of inserted watermarks on timing and resource overhead at a fixed interconnection length (I/L). Interconnection length specifies how many of the inserted SRLs were interconnected to form a cyclic register. We convert an equal number functional LUT2 and LUT3 cells to SRL cells. The cells to be converted were chosen according to a random number, depending on the signature. On average, this provided 10 bits for watermark insertion in each cell. Results are shown in Table 2. Timing and resource overhead are provided in percent difference to the values of unmarked cores from Table 1.

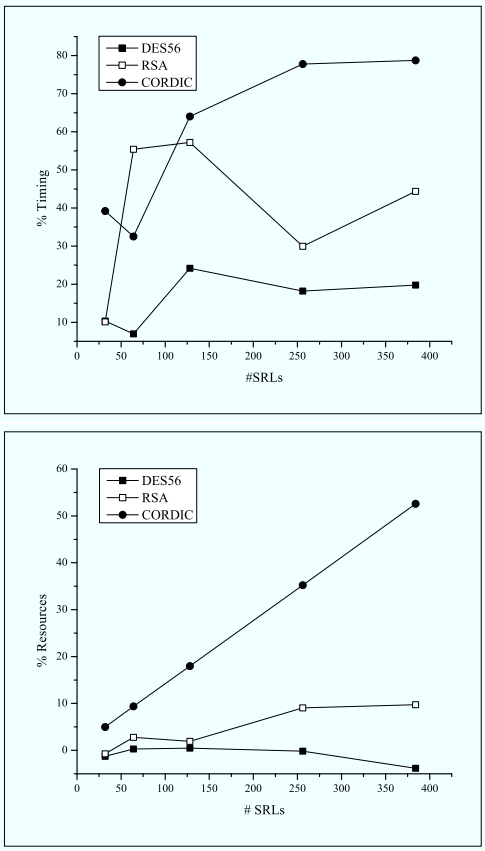

Figure 7. Plot of the timing and resource overhead according to results from Table 2.

Plots of the timing and resource overhead can be seen in Figure 7. The non-linear behavior might be due to heuristic optimization algorithms and random, but signature dependent placement of the watermark resources. Decrease of resource usage might be due to SRLs being packed more densely into the available slices, than it would be the case with LUTs. The very high decrease in timing quality and the explicit differences between different kinds of cores, might be explained by the fact that the conversion from LUT to SRL prohibits design tools from performing timing optimization. If applied to a core alike the DES56 core, the results are promising. For example, a reasonable amount of signature bits, i.e. 500 bits, can be inserted into a core by converting 50 LUTs to SRLs, on average. If we pick out the DES56 core, resources used remain unchanged, and timing overhead would not exceed 10%.

5.2 Effect of Interconnection Length and Number of Inserted Watermarks on Timing and Resource Overhead

We only use the DES56 core to examine the effects of various interconnection lengths on timing and resource overhead. Watermarks are inserted in relation to the percentage of the overall convertible content. Table 3 shows the obtained results. Values are provided in percent difference from those properties obtained from the unmarked design. Figure 8 shows plots of the results from the table for timing and resource overhead, respectively. According to the

%SRLs	10	20	30	40	50	70	90
Chain Length 1							
% Timing	18.2	12.3	9.1	36.8	39.9	36.2	14.0
% Resources	0.2	3.0	4.3	5.1	6.7	8.6	14.5
Chain Length 4							
% Timing	15.4	12.3	3.2	27.7	25.5	37.7	23.2
% Resources	1.2	-0.5	0.3	-0.2	-5.0	-7.0	-5.7
Chain Length 8							
% Timing	15.2	8.3	5.4	26.3	25.7	29.4	20.8
% Resources	0.9	1.0	2.4	-0.4	-2.5	-5.7	-4.9
Chain Length 16							
% Timing	14.4	9.5	5.7	23.7	28.4	27.3	19.2
% Resources	2.6	1.8	2.8	-2.5	-3.3	-5.6	-5.2
Chain Length All							
% Timing	15.7	13.2	7.3	30.0	25.5	36.8	20.1
% Resources	1.5	1.8	3.4	-3.6	-4.1	-6.8	-6.3

Table 3. Effect of I/L and number of inserted watermarks on timing and resource overhead.

obtained results, the best timing can be achieved, if about 30% of the total amount of available LUTs are converted to SRLs, which roughly corresponds to about 200 inserted watermarks. This 30% ditch was carefully reviewed, but could not be verified in other scenarios. Additionally, it can be observed, that interconnecting each SRL to itself leads to the worst resource overhead. For intermediate interconnection lengths, there is only a slight difference in terms of timing and resource overhead. It can be concluded, that interconnecting the SRLs might help the CAD tools to pack the components more densely into the available slices.

5.3 Extraction Test

We have also analyzed, whether the inserted marks can be extracted from a bitfile. Measures of interest for each interconnection length include, how many of the inserted marks can be clearly identified and how many indeterminable duplicates there are. By indeterminable duplicates,

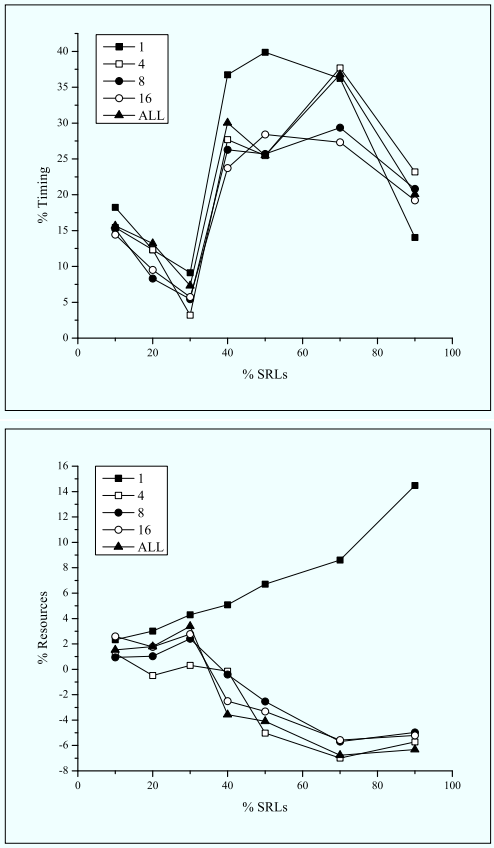

Figure 8. Plot of the timing and resource overhead according to results from Table 3.

we mean values, that can be found more than once in the bitfile, so that it is impossible to determine, which one is the watermark. Results are presented in Table 4 and Figure 9. The test shows, that apart from self-interconnection, almost all lengths of interconnection will produce less than two duplicates.

6 Conclusions

We have presented a novel approach to watermark FPGA designs on the netlist level, by restricting the dynamically addressable part of the logic table and using the space to insert signature bits. We tightly integrate the watermark with the LUTs of the design, so that simply removing the mark carrying components would damage the IP core. Converting functional LUTs to LUT-based RAMs or shift registers prevents deletion due to optimization, thus we were able to take watermark carrying components out of the scope of optimization algorithms to achieve transparency towards development environments. We have also shown, how the watermarks can be extracted from the bitfile of an FPGA.

%SRLs	10	20	30	40	50	70	90
I/L	% Duplicates						
1	4.41	6.62	5.76	3.47	3.48	2.32	1.76
4	1.10	1.29	2.21	1.30	0.71	0.91	0.66
8	0	0.37	1.47	1.13	0.64	0.76	0.59
16	0	0.18	0.49	1.39	0.43	0.60	0.43
All	0	0	0.37	0.70	0.50	0.61	0.20

Table 4. Percentage of irresolvable duplicates compared to inserted marks at different I/L.

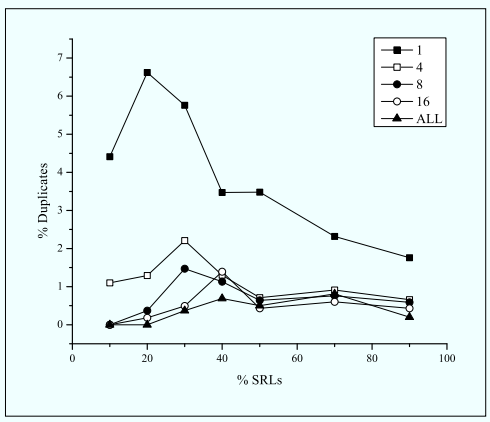

Figure 9. Percentage of irresolvable duplicates compared to inserted marks, according to Table 4.

We can detect the authorship of IP cores without having to request additional information from the producer. The proposal was tested on a Xilinx Virtex-II Pro FPGA and showed low overhead in terms of timing and resources at a reasonable number of watermarked cells.

References

[1] Opencores.org. *http://www.opencores.org*.

[2] *Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Complete Data Sheet*, Oct. 2005.

[3] F. Bai, Z. Gao, Y. Xu, and X. Cai. A Watermarking Technique for Hard IP Protection in Full-custom IC Design. *Communications, Circuits and Systems, 2007. ICCCAS 2007. International Conference on*, pages 1177–1180, 11-13 July 2007.

[4] E. Castillo, A. Garcia, L. Parrilla, D. Morales, A. Lloris, and U. Meyer-Baese. Digital Signature Embedding Technique for IP Core Protection. *Programmable Logic, 2007. SPL '07. 2007 3rd Southern Conference on*, pages 143–148, 28-26 Feb. 2007.

[5] N. Ferguson and B. Schneier. *Practical Cryptography*. Wiley Publishing, Inc., Indianapolis, Indiana, USA, 2003.

[6] V. A. I. P. D. W. Group. Intellectual Property Protection: Schemes, Alternatives and Discussion. Version 1.1. Whitepaper IPPWP 1 1.1, VSI Alliance, 2001.

[7] I. Hong and M. Potkanjak. Behavioral Synthesis Techniques for Intellectual Property Protection. In *Design Automation Conference*, pages 849–854, 1999.

[8] A. K. Jain, L. Yuan, P. R. Pari, and G. Qu. Zero Overhead Watermarking Technique for FPGA Designs. In *GLSVLSI '03: Proceedings of the 13th ACM Great Lakes symposium on VLSI*, pages 147–152, New York, NY, USA, 2003. ACM Press.

[9] 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 *Design Automation Conference*, pages 776–781, 1998.

[10] D. Kirovski, Y.-Y. Hwang, M. Potkonjak, and J. Cong. Intellectual Property Protection by Watermarking Combinational Logic Synthesis Solutions. In *ICCAD*, pages 194–198, 1998.

[11] Lach, Mangione-Smith, and Potkonjak. Fingerprinting Techniques for Field-Programmable Gate Array Intellectual Property Protection. *IEEETCAD: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20, 2001.

[12] Q. Li and E.-C. Chang. Zero-Knowledge Watermark Detection Resistant to Ambiguity Attacks. In *MM&Sec '06: Proceedings of the 8th workshop on Multimedia and security*, pages 158–163, New York, NY, USA, 2006. ACM.

[13] Q. Li, N. Memon, and H. T. Sencar. Security Issues in Watermarking Applications - A Deeper Look. In *MCPS '06: Proceedings of the 4th ACM international workshop on Contents protection and security*, pages 23–28, New York, NY, USA, 2006. ACM Press.

[14] S. Meguerdichian and M. Potkonjak. Watermarking While Preserving the Critical Path. In *Design Automation Conference*, pages 108–111, 2000.

[15] D. Saha and S. Sur-Kolay. Fast Robust Intellectual Property Protection for VLSI Physical Design. *Information Technology, (ICIT 2007). 10th International Conference on*, pages 1–6, 17-20 Dec. 2007.

[16] D. Ziener, S. Aßmus, and J. Teich. Identifying FPGA IP-Cores based on Lookup Table Content Analysis. In *Proceedings of 16th International Conference on Field Programmable Logic and Applications*, pages 481–486, Madrid, Spain, Aug. 2006.

[17] D. Ziener and J. Teich. Evaluation of Watermarking Methods for FPGA-Based IP-cores. Technical Report 01-2005, University of Erlangen-Nuremberg, Department of CS 12, Hardware-Software-Co-Design, Am Weichselgarten 3, D-91058 Erlangen, Germany, Mar. 2005.

[18] D. Ziener and J. Teich. Power Signature Watermarking of IP Cores for FPGAs. *Journal of Signal Processing Systems, Volume 51, Number 1*, pages 123–136, April 2008.