Intellectual Property Protection via Watermarking: A Survey

Peter A. Jamieson

Winter Semester 2000

Department of Electrical and Computer Engineering
University of Toronto
Toronto, Ontario M5S 3G4, Canada

April 17, 2000

Abstract

Intellectual property protection (IPP) is a reasonably established field that attempts to protect peoples' creativity. In our present technological era, where the words *electronic* and *computer* enter all elements of our lives, intellectual property (IP) has to be protected in the new electronic medium. Essentially, products exist in an electronic medium, in which data can be easily copied, modified, and examined at the lowest implementation level. The technological world has adopted an old principle called watermarking, and has adapted it for use within the digital domain.

Watermarking, the addition of a recoverable mark that demonstrates authenticity or ownership, has been used to protect digital multimedia, and the question is, can watermarking be applied to any forms of media/data. Specifically, how can watermarking be used to protect the design property of software and hardware. In this paper the current state-of-the-art software and hardware watermarking schemes will be examined. The main goal of the watermark in hardware and software designs is to prove ownership of the original design; if a watermark is attacked, then the watermark scheme has a level of defense against the various types of attacks.

1 Introduction

The watermark was originally used to protect a countries currency from being counterfeited. Basically, unique watermarks were put onto paper currency so that each piece of paper could be identified as authentic. In the current electronic age this concept has been introduced to protect intellectual property (IP), and has been successfully applied to multimedia. This paper attempts to survey watermarking concepts, and specifically, how they have been applied to protect software and hardware.

1.1 Watermarking

Watermarking is the introduction of a unique object *O* into an item *I* such that *O* can be recovered at a later time, and *O* does not destroy *I* (Figure 1). We do not destroy *I* if *I'* is interpreted as equal to *I*, where *I'* has the watermark *O* embedded. For example, if a digital image has been watermarked, then a human cannot see a difference between the original and the watermarked image. Watermarking is an aspect of steganography, and therefore, can be analyzed based on its data-rate, stealth, and resilience [4]. These qualities will help us examine any watermarking scheme or algorithm.

1.2 Software and Hardware Watermarking

Now that we have a general idea of what watermarking is, we have to define software and hardware watermarking. Collberg and Thomborson [4] give a good definition for software watermarking as:

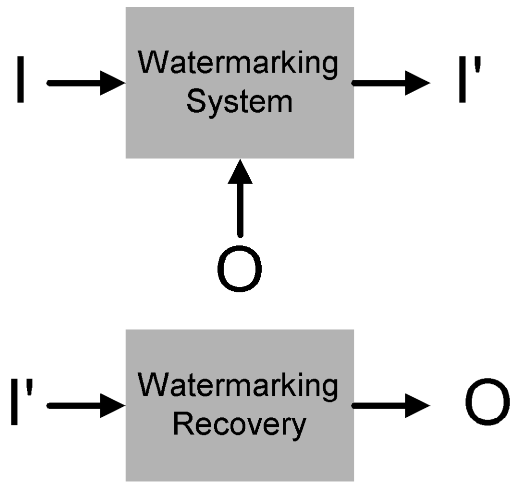

Figure 1: Basic Watermarking

”Embed a structure W into a program P such that: W can be reliably located and extracted from P even after P has been subjected to semantics preserving transformations such as code optimization and obfuscation; W is stealthy; W has a high data rate; embedding W into P does not adversely affect the performance of P; and W has a mathematical property that allows us to argue that its presence in P is the result of deliberate actions.”

The main concern in software is that algorithms that exist as IP are protected so that the original authors can prove that someone has copied their software. These issues are becoming more important since software has become part of modular design, and low level algorithms are reused by new system designs. Original authors of the low level algorithms or various modules of a system may want to identify that someone else has incorporated their original work, and potentially, software watermarking could identify the original authors code.

Hardware is also becoming a process in which modular design and reuse are key elements of the design process, so hardware also needs an element of protection such that owners can claim their IP. The definition of software watermarking can be used for hardware watermarking if we consider the ”program P” to be a design in a hardware description language (i.e. VHDL or Verilog); therefore, we are attempting to add a watermark to hardware through the various stages of design process. The basic concepts for hardware and software watermarking are similar, but the actual implementation for each is different, and each area of watermarking is treated separately.

1.3 Organization of the Paper

The remainder of this paper examines both hardware and software watermarking separately. Section 2 examines the basic attacks against any watermark. Section 3 examines the software watermarking schemes and potential attacks; then, section 4 looks at hardware watermarking, and attacks, and finally, section 5 concludes the paper.

2 Watermarking Attacks

Collberg and Thomborson present subtractive attack, distortive attack, and additive attack as three possible ways of corrupting a watermark, where corruption means the watermark can no longer be used to prove ownership.

- Subtractive Attack: Eliminate the presence of the watermark,
- Distortive Attack: Introduce distortion via transformations such that the watermark can no longer be recovered.
- Additive Attack: Insert information to completely cover the original watermark or add information so that the original watermark cannot be trusted as the original watermark.

Different methods of watermarking will have varying abilities to withstand the three attacks; this is called resilience of the watermarking method. An attack on a watermark is only useful if the attack does not destroy the original watermarked object [4].

3 Software Watermarking

Software watermarking (SW) is an attempt to protect the designer's original work. A piece of software, when deployed in the market, is in a low level format such as assembly, bytecode or machine instructions, and this normally would protect the designer since large software systems are very hard to understand from their low level code. For example, a simple program in C like ”hello word” has seventeen assembly instructions when compiled in Linux, and this is only for one simple function call.

```
#include ”stdio.h”

void main()
printf(”Hello World”);
```

The reason software watermarking has become so important is because low level code can be decompiled back into higher levels. Specifically, the executable code for ”Hello World” can be transformed back into the C code, and now someone could potentially take the ideas' of the original software designer. This is particularly relevant to Java bytecode, since it can be easily transformed back into a Java program [3]. Watermarks can now be used to identify and prove that the designer's code has been reused. Figure 2 shows software watermarking as a subset of watermarking; additionally, the hierarchy shows that SW can be implemented either dynamically or statically, and that these implementations are subject to attacks. To better understand SW we should examine both static and dynamic methods, and how these methods are affected by attacks.

3.1 Static Software Watermarking Methods

Static SW means that the watermark is part of the executable; the watermark is data contained within the executable. Static watermarks are further decomposed into code watermarks and data

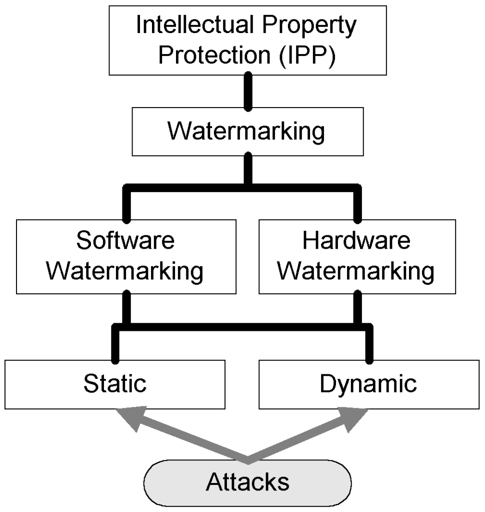

Figure 2: Hierarchy classification of Intellectual Property Watermarking

watermarks. Code watermarks are contained within the code of the executable as Figure 3 demonstrates, and data watermarks are contained anywhere else, in places like header files, resource files (MFC), or data files (Figure 3).

The advantages of the static watermark is that it is easy to implement, easy to uniquely identify, and they don't affect the performance of the program. The problem, however, is that the technique is so simple that it is easy to attack via distortive attacks, and in the case of data watermarks, a subtractive attack may also be successful.

A distortive attack would attempt to change the syntax (grammatical structure) of the code while keeping the semantics (the meaning). If a thorough distortive attack is performed, then many of the data and code watermarks could be destroyed, since they heavily depend on syntax and defined structures. For example, any string could be broken up into a chain of characters, or any control flow could be altered, but the program would still execute as expected. A data watermark, which is embedded into the executable could be attacked by examining the code for obvious copyright identification and eliminating them. To strengthen these watermarks, encryption

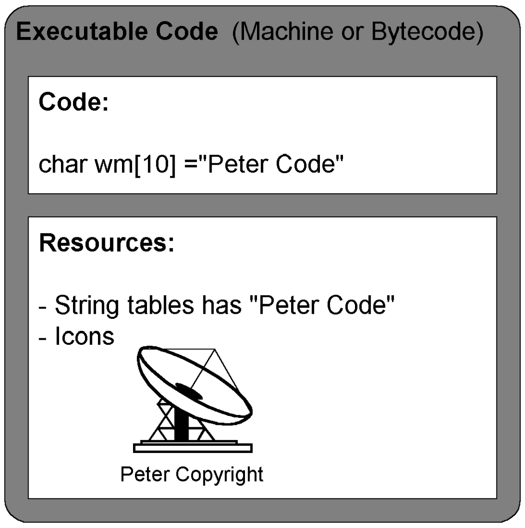
Executable Code (Machine or Bytecode)
Code:
char wm[10] ="Peter Code"
Resources:
- String tables has "Peter Code"
- Icons
Peter Copyright

Figure 3: Basic Watermarking

could be applied where feasible, but in general, distortive attacks are very effective against the static watermark [4].

The negative aspects of the static software watermark are important, but the introduction of these types of watermarks forces an attacker to perform actions to eliminate the watermark. These actions take time, and are considered a deterrent; therefore, there is value in using the static watermark, but it probably shouldn't be used as the only intellectual protecting mechanism.

3.2 Dynamic Software Watermarking Methods

So embedding information into the executable is reasonably effective, but does not generate a very strong watermark. A dynamic watermark does not embed the data within the executable, but uses the executable code. Basically, when code is executed it takes in input data and transforms it into output data, and this is what is taken advantage for dynamic SW. For a unique set of special inputs we use the output of the program as the unique watermark identification, where special inputs are considered inputs that would not occur during the normal use of the program. Collberg and Thomborson, identify three types of dynamic software watermarks: Easter Egg Watermarks, Data Structure Watermarks, and Execution Trace Watermarks (Figure 4). These are the stronger and more complicated software watermarks, so the implementation of each of the three types will be examined more thoroughly than static SW. Unfortunately, Collberg and Thomborson seem to be the only active members of the SW community and the following information is only studied from [4].

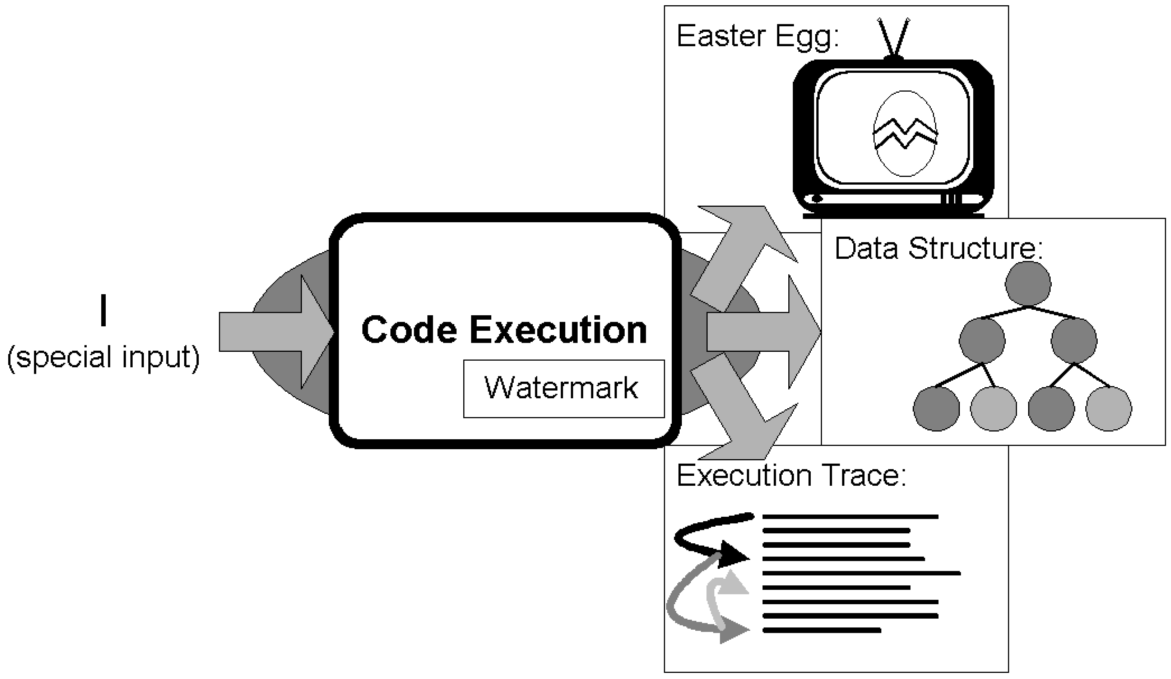

Figure 4: Illustrates the three dynamic software watermarking schemes based on the inputs and outputs

The Easter Egg watermark is probably one of the most exciting watermarks because the Easter Egg has been traditionally used as programmers' signatures in a program, where the signature is usually humorous. A website, for example, *www.eeggs.com* describes hundreds of known eggs in commercial software. An Easter Egg is activated when some special input is performed, and the program will alter its execution and perform some special operation like a flight simulator in

Microsofts' Excel. The problem with using this method as a software watermark, is that web sites, like the one described above, notify people of the watermark. From this information, a subtractive attack can be easily performed since there is known information about the watermark.

A dynamic data structure watermark is generated by the code as it is executed. The special inputs will generate the data structure such that when we examine the variables at run-time, the watermark can be extracted. The advantage of this type of watermark is that there is no unique output generated, and the existence of the watermark is hard to identify given the executable. The problem is that we rely on variables, and the variables can be attacked by distortion in which semantics are preserved, but syntax is modified. In this case, splitting of variables is a viable attack.

Finally, dynamic execution trace watermark is a trace (record of execution steps) of the executable given a specific input. This approach also keeps the watermark well hidden, but semantic preserving transformations will again eliminate the watermark.

It is apparent that the various SW techniques are susceptible to subtractive and distortive attacks, but distortive attacks, as obfuscating transformations, seem to be the best way to attack a watermark. To tamperproof a watermark we have to improve its resilience to all the possible attacks, and though this is probably not possible, watermarks that are resilient to most of the distortive attacks presently referenced in [4] and [3]. This renders the software watermarks as relatively tamperproof.

3.3 Example Software Watermarking Method

An example of one particular watermarking method is a modification of the Radix-k Encoding described in [5]. Consider that we have a linked list with its own linked list pointer called *llp1*.

The linked list also has two extra pointers; let us call them *p1* and *p2*. To encode watermark data in the dynamic structure we define a simple test. First off, the distance of a pointer is defined as the number of nodes in the linked list that must be crossed via llp1 pointers to get to the node pointed to by *p?*. With distance defined, we can encode watermark data in a variety of ways, but let us define a ternary system in which:

- a '1' is encoded if at a specific node the distance of *p1* equals the distance of *p2*.
- a '2' is encoded if at a specific node the distance of *p1* is greater than the distance of *p2* .
- a '3' is encoded if at a specific node the distance of *p1* is less than the distance of *p2*.

From this system we can encode a watermark using each of the nodes of the linked list excluding the first node. A graphical representation and example of the preceding method can be viewed in figure 5.

The strength of this watermarking technique, in which dynamic data structures are used with pointers as the data controlling entity, is that they are not easy to distortively attack. The pointers can't be removed or split easily, and the linked list, in its non-watermarking state, is an integral part of the code so it can not be subtractively attacked. This method, however, may be attacked by additive attacks. Collberg and Thomborson identify four ways that these dynamic data structures could be attacked.

1. Add extra pointers like *p3*, *p4* and *p5* that have their own distances.
2. Rename and reorder the structure fields so that it is hard to identify *p1* and *p2*.
3. Split nodes so that *llp1* is broken up into a series of steps.
4. Add "bogus" nodes pointing into the structure (Does not affect this particular example)

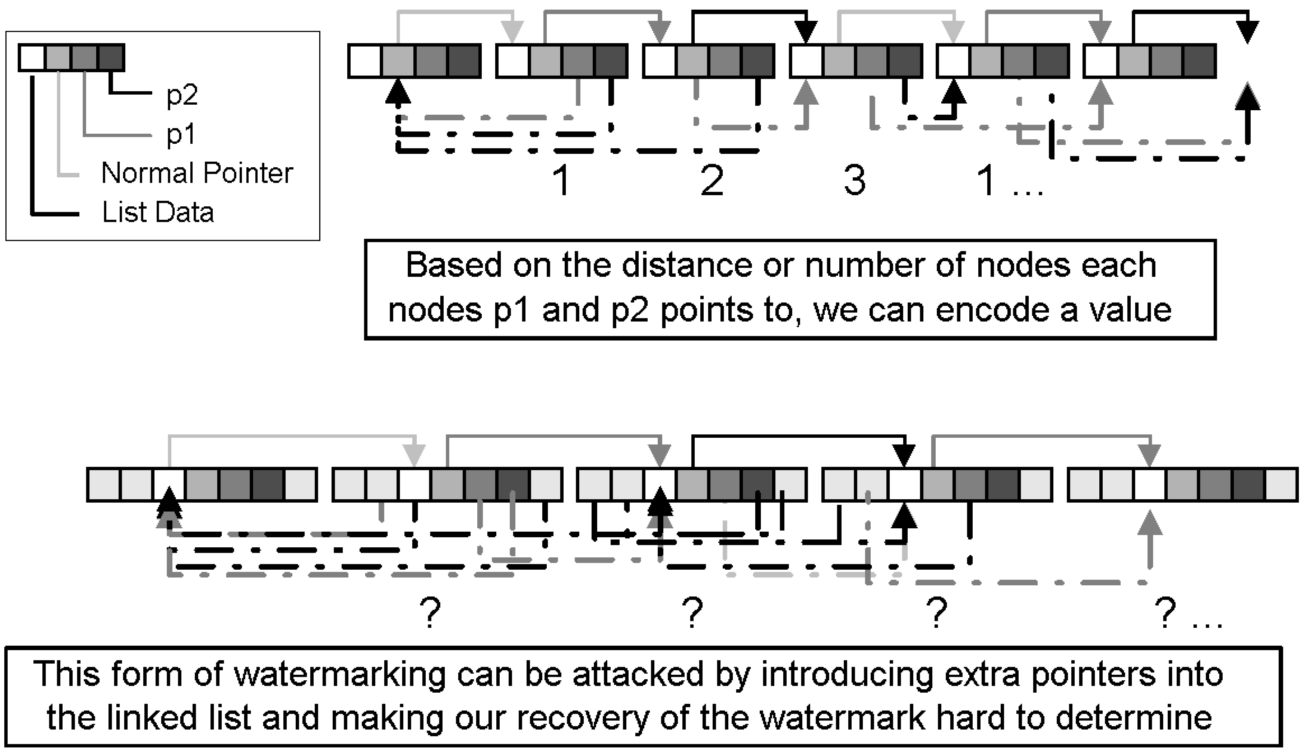
p2
p1
Normal Pointer
List Data
1
2
3
1 …
Based on the distance or number of nodes each nodes p1 and p2 points to, we can encode a value
?
?
?
? …
This form of watermarking can be attacked by introducing extra pointers into the linked list and making our recovery of the watermark hard to determine

Figure 5: Illustration of how the linked list watermarking would look like, and one attack that would destroy the watermark

If the constraints of *p1* and *p2* are further restricted then the watermark becomes more resilient tp the above attacks. For example, the distances between *p1* and *p2* can be forced to differ by a maximum of one, and *p1* and *p2* can be forced to point in the opposite direction of *llp1* (where direction is the way in which a pointer traverses the list). These constraints will solve some of the pointer identification and addition problems, but the proposed watermark is still not perfect (Figure 5).

The example demonstrates that a watermark can be resilient against many attacks, but there currently is no such thing as a perfect software watermark. The best thing to introduce in software is a number of different types of watermarks. This would add more protection to the watermarks because multiple watermarking attacks would need to be performed on the code, which would hopefully cause the code to become very complicated with high overhead as the watermark is destroyed.

4 Hardware Watermarking

Hardware watermarking (HW) has the same goals as SW, but the intellectual property that must be protected for hardware exists as a physical entity, as opposed to the electronic format of software. This partition between hardware and software is not black and white since many aspects of hardware design are entering the software domain. Hardware description languages, like VHDL or Verilog, have been built so designers can specify a circuit in a language, and synthesize the circuit into a physical manifestation of the description. The first thing to address is what aspect of the hardware needs to be watermarked. If there is a concern that VHDL modules (similar to software procedures and functions) may be used by others, then the VHDL code must be watermarked, and we can use the SW techniques. Unfortunately, HDL does not usually employ dynamic structures, so there is a limit to the strength of the watermarks that can be applied at the HDL design. This problem does not really exist in the hardware design world since HDL code is not normally distributed, and the only thing that is distributed is the physical hardware.

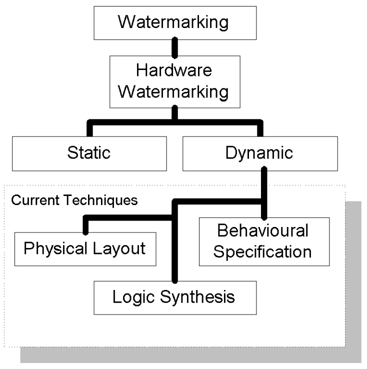

Figure 6: Hierarchy classification of current developments in Hardware Watermarking

Since the physical hardware is distributed, the watermark must be applied to the physical design. The watermark manifests itself in the physical hardware. The elements that potentially can be used to enter a watermark are the routing of connections, and the placement of components. These elements of design are built by software tools that attempt to find minimizations with respect to time, size, and various other circuit requirements. A tool is used to simplify the minimization problem, but this means a watermark can not be introduced by just excepting a particular minimum routing. To add the watermark constraints must be applied at the design stage to introduce the watermark. These design constraints have currently been introduced at three levels: logic synthesis [6], physical design [2], and behavioral specification [7].

Paper [2] gives the best introduction of the constraint watermarking concept via the satisfying (SAT) problem, and this general constraint based scheme is applied in all the specific HW found in this survey. The constraint based schemes have one characteristic in common; the circuits have problems that are NP-complete, and this means that the hardware circuits are solving an optimization [2]. This suggests that the intellectual property of hardware that needs to be protected is a special class of circuits in which there is an aspect of optimization. Imagine that there are a series of constraints that are already present with current design. A watermark is introduced by adding more constraints, where these additional constraints do not alter the hardware, but still narrow the range of possible optimizations. The interesting thing to notice here is that if software solves an optimization based on constraints, then the software can be watermarked by the same principles [2].

Since the HW schemes fall into one general class, then only the Combinational Logic Synthesis Method [7] will be examined in depth. Various other methods can be found in [2], [1], [6], and [8].

4.1 Example Hardware Watermarking Method

A similar example to [6] appears in Figure 7(a). This is much simpler than the system presented in [6], but the concept can be illustrated. The problem is, given the logic network and the mapping cells available (Figure 7(b)), what is the minimum mapping? The minimum in this case is four mappings, with a total of ten possible mapping arrays. To introduce a watermark, additional constraints must be added onto the logic network so that we still get the same optimization, but the problem has been solved for a more unique problem. If the output of each individual gate is enumerated (Figure 7(a)), then to further constrain the problem we can choose new external outputs from outputs inside the logic network. For example, if the number 3 is to be encoded, then we hook an external output off of 3 (Figure 7(c)). The minimum mapping for the network remains four, but given the constraint, there is only three possible ways to get this minimum. Therefore, the probability that another different optimization technique would come up with the same organization is 3/10.

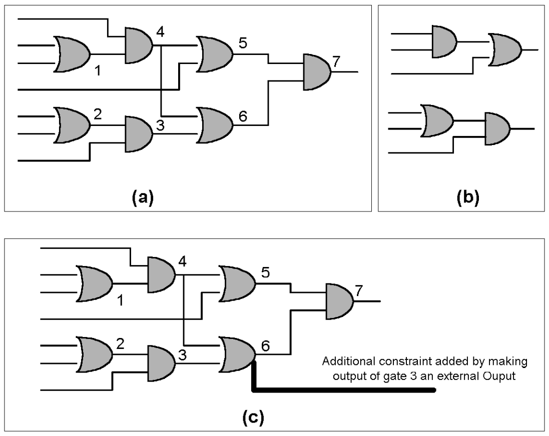
1
2
3
4
5
6
7
(a)
(b)
1
2
3
4
5
6
7
Additional constraint added by making output of gate 3 an external Ouput
(c)

Figure 7: An example of the Logic Synthesis Optimization problem. (a) contains the logic network. (b) shows the possible cells that are avialable to map onto the network. (c) shows the constrained problem that will optimize a specific watermark into the design.

This example is very small, and as the logic circuit becomes more complex the optimizations become more difficult, and probability of the same solution decreases. The nicest aspect of this method is that the design is functionally equivalent, and the minimization is still the best possible. I suspect that some numbered output choices, like output 1 or output 2, may cause the minimal solution to be greater than the original unconstrained problem, but the watermarking tool could be designed to take this into account, and modify the watermarking method.

The possible attacks against this method of watermarking are subtractive attack, and additive attack. The subtractive attack requires that outputs are sufficiently altered so that the output no longer corresponds to the original watermark with reasonably low probability. To change the outputs, the attacker has to rebuild the circuit or basically make a new optimization algorithm. The additive attack consists of the attacker attempting to find another watermark with sufficiently low probability that he/she can claim that his/her watermark protects the intellectual property.

5 Conclusion

The main conceptual difference between multimedia watermarking and software/hardware watermarking is that in the second case an element of functionality is introduced. The functionality in SW/HW must be kept; in the case of software this is the semantics, and in the case of hardware this is output equivalence ([2] further elaborates on this under the section Artifact vs. IP Protection Watermarking). This restriction makes it harder to add a watermark, but on the other hand there is potential to integrate the watermark as part of the functionality. If this were the case, then the watermark would be coupled with the hardware/software so that it would almost be impossible to eliminate or transform.

This paper has attempted to survey the current state of the art watermarking techniques on both software and hardware. These two areas, software and hardware, have different methods applied to them, but there is a fine line between the two disciplines, and at some point these divisions may blur and become part of a superset - IPP Watermarking. The individual watermarking methods were examined mainly on their implementation, and their resilience against the various attacks. The conclusions reached or implied by each research group is that IP watermarks can be built to be resilient, but not resilient to every possible attack. This suggests that a hierarchy of watermarks will perform the best as compared to watermarks by themselves.

References

[1] W. H. M.-S. S. I. L. M. M. P. P. T. H. W. A. B. Kahng, J. Lach and G. Wolfe. Robust ip watermarking methodologies for physical design. Technical report, UCLA, 1998.

[2] W. H. M.-S. S. I. L. M. M. P. P. T. H. W. A. B. Kahng, J. Lach and G. Wolfe. Watermarking techniques for intellectual property protection. In *DAC-98 35th ACM/IEEE DAC Design Automation Conference*, San Francisco, CA, June 1998.

[3] D. L. C. Collberg and C. Thomborson. A taxonomy of obfuscating transformations. Technical report, University of Aukland, 1998.

[4] C. Collberg and C. Thomborson. On the limits of software watermarking. Technical Report 164, University of Aukland, Aug. 1998.

[5] C. Collberg and C. Thomborson. Software watermarking: Models and dynamic embeddings. Technical report, University of Aukland, 1998.

[6] M. P. n. J. C. D. Kirovski, Y. Hwang. Intellectual property protection by watermarking combinational logic synthesis solutions. In *International Conference on Computer-Aided Design*, pages 194–198, 1998.

[7] I. Hong and M. Potkonjak. Behavioral synthesis techniques for intellectual property protection. Technical report, UCLA, 1997.

[8] G. Qu. Watermarking graph partitioning solutions. *ACM Trans. Comput. Syst.*, 6(4):393–431, Nov. 1998.