Watermarking While Preserving The Critical Path

Seapahn Meguerdichian, Miodrag Potkonjak
Computer Science Department, University of California, Los Angeles
seapahn@cs.ucla.edu, miodrag@cs.ucla.edu

Abstract

In many modern designs, timing is either a key optimization goal and/or a mandatory constraint. We propose the first intellectual property protection technique using watermarking that guarantees preservation of timing constraints by judiciously selecting parts of the design specification on which watermarking constraints can be imposed. The technique is applied during the mapping of logical elements to instances of realization elements in a physical library. The generic technique is applied to two steps in the design process: combinational logic mapping in logic synthesis and template matching in behavioral synthesis. The technique is fully transparent to the synthesis process, and can be used in conjunction with arbitrary synthesis tools. Several optimization problems associated with the application of the technique have been solved. The effectiveness of the technique is demonstrated on a number of designs at both logic synthesis and behavioral synthesis.

1. Introduction

Recently, design using Intellectual Property (IP) has been receiving a great deal of attention. As system engineers discover the benefits of design using tested and verified IP components, the need for Intellectual Property Protection (IPP) becomes increasingly apparent. One of the methods used for IPP is watermarking. With watermarking, a unique signature is encoded in the design such that it becomes an integral part of the system. In some cases the new constraints introduced by a watermark lead to inefficiencies in the original designs that can be undesirable. Here we show a new watermarking method with results that in addition to meeting the criteria of sound watermarks, preserve the critical path of the designs, eliminating possible delay overheads.

We motivate the key idea behind our approach using an example.

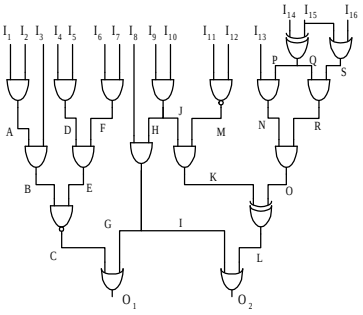


Figure 1 - Motivational Example

In order to create the physical hardware, the circuit in ***Figure 1*** must be mapped to a set of physical realization elements. For demonstration purposes, we use a *K-M-Macrocell* mapping approach where the final physical blocks will each have a maximum of *K* inputs and implement a Boolean function with at most *M* product terms. ***Figure 2*** shows a mapping solution of this circuit with the values of *K=4* and *M=5*. The maximum delay of this mapping solution is directly related to the total depth of the circuit. In this case, the circuit has a depth of *3* nodes. Therefore, any watermarking solution that attempts to preserve the critical path may not have a total depth of more than *3* nodes. Our watermarking algorithm produces the result shown in ***Figure 3***:

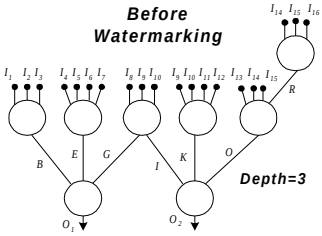


Figure 2 - Mapping Solution Before Watermarking

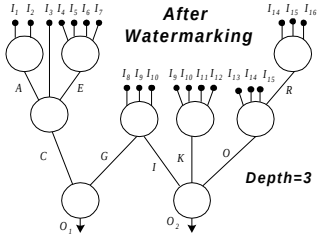


Figure 3 - Mapping Solution With Watermark

As shown later, a set of properly chosen constraints in the watermarking preprocessor forces the mapping algorithm to produce a new mapping solution. Although the two versions perform the same function, the watermarked result has a specific structure that is proof of authorship.

2. Related Work

Template Matching is a fundamental task and notion in a number of scientific and engineering domains. For example, it is the first axiomatic principle in theoretical immunology [Per79], and is widely considered as the key activity which will enable emerging computer aided drug-design [Per90]. A comprehensive survey on string and several pattern matching techniques in computer science is given in [Aho90].

The major impetus to widespread use of pattern matching was due to a code generation scheme suggested by Hoffman and McDonnel [Hof82]. The tree matching method has been used in a number of successful compiler projects [Aho90]. Through use of the tree processing language Twig [Aho89] it was also applied in logic synthesis for technology mapping [Keu87]. Template matching has attracted a great deal of interest in behavioral synthesis [Not91]. Other pattern matching approaches in logic synthesis were predating the application of tree matching methods [Dar81, DeG85].

Data watermarking embeds hidden data into an object for the purpose of identification, annotation, and copyright. Two main directions in watermarking have emerged. The first direction aims to protect static artifacts, such as text, image, audio, and video, by leveraging on the inabilities of human preceptor systems in detecting minute changes in the object. Comprehensive surveys on this direction include [Voy99, Nik99].

A few years ago a conceptually new direction in watermarking was proposed [Hon98]. This set of techniques aim to embed a signature into hardware, software, or other functionally dynamic artifacts, by altering the structure of the objects while fully

DAC 2000, Los Angeles, California

preserving their functionality. The two main strategies that have been proposed are *horizontal* and *vertical* watermarking. In the horizontal watermarking, the design/software is structurally altered by a preprocessing or a post-processing step during its creation [Kah98, Oli99]. In the vertical approach, the watermark is embedded during a synthesis or compilation step. The specification of the design at one process level is augmented with additional design constraints that correspond to a message. Therefore, after the synthesis step, the design has structural properties that correspond to the signature constraints [Hon98, Cha98].

3. Preliminaries

A Boolean network can be represented as a directed acyclic graph (DAG). Each node represents a logic gate, and the directed edge *(i,j)* is in the DAG if the output of gate *i* is an input of gate *j*. We denote any node with no incoming edges as a Primary Input (*PI)* and any node with no outgoing edges as a Primary Output *(PO)*.

We use *input(i)* to denote the set of all the nodes that fanin to node *i*. A *K-bounded* network is defined as a Boolean network with $|input(i)| \leq K$ for each node *i*. A *K-LUT* is a special logic block that has *K* inputs, *one* output, and can implement any K-input Boolean function. *K-LUTs* are the basic design blocks found in most FPGA architectures. Similarly, A *K-M-Macrocell* is a special logic block that has *K* inputs, *one* output, and can implement any K-input Boolean function that requires at most *M* product terms. A PLA architecture can easily be modeled using *K-M-Macrocells.*

A *K-LUT* or *K-M-Macrocell* based mapping covers a given Boolean network with *K-LUTs* or *K-M-Macrocells.* Note that for delay optimization, the blocks corresponding to the LUTs or Macrocells are allowed to overlap, resulting in logic duplication. In any such mapping, attempts can be made to reduce the area, the depth [Con94], or both area and depth of the final mapping solution.

The USERID string used in the algorithm is obtained by the RSA public key encryption using the PGP software package. A specific text file that is provided by the owner is encrypted to provide a cryptographically strong pseudorandom bit stream that represents the signature being encoded.

In our algorithm for technology mapping, we assume that we start with a *2-bounded* Boolean network. However, this method can be applied to any general logic network. According to [Con94], there are a number of ways to transform any Boolean network into *K-bounded* networks. Our watermarking algorithm can be applied to any general logic network that can be mapped for optimal delay constraints. Throughout this paper, we use the *K-M-Macrocell mapping* approach in all our examples. The watermarking technique used in our approach is independent of the type of mapping that is used. We assume that delay minimization is the primary mapping constraint.

4. Approach

The algorithm for watermarking a given logic network in the technology mapping phase, while preserving the critical path, can be summarized with the following five major preprocessing steps:

1. Obtaining an initial mapping solution.
2. Identifying the ϵ-critical path in the mapped solution.
3. Determining the nodes in the original network that are mapped to the ϵ-critical path in the mapping solution.
4. Selecting a set of signals that are not on the ϵ-critical path to encode the watermark.
5. Applying the watermark constraints.

After these preprocessing steps, the network can be mapped to obtain the watermarked solution. The final step involves a simple post-processing routine that removes the extra constraints added in the preprocessing step. The following pseudo code describes the algorithm:

G(N,E) is a DAG that describes the given network.

- Obtain a mapping solution: M = DoMapping(G)
- Mark critical nodes:
  For each node $n_i \in M$
  If n_i is on the ϵ-critical path of M, set $Critical(n_i)=true$.
- Extrapolate critical nodes from M to G:
  For each node $n_i \in G$
  Find node $n' \in M$ such that n_i is mapped to node n'.
  If $Critical(n') = true$, then set $Critical(n_i)=true$.
- Assign sequential labels to each non critical signal:
  For each internal non-critical node $n_i \in G$
  Assign unique sequential label to all fanin signals of n_i.
- Select the encoding signals:
  S = *Select_Watermark_Signals*(USERID)
- Introduce new constraints by breaking the selected signals into a set of primary input/output pairs.
  For each $e_i \in S$
  Add primary output POe_i from starting node of e_i.
  Add primary input PIe_i at the ending node of e_i.
  Delete e_i from G.
- Obtain the final mapping solution: M = DoMapping(G)
- Remove all the constraints added in the watermarking step.
  For each $e_i \in S$
  Short Primary Outputs POe_i and Primary Input PIe_i

Determining the *ϵ-critical path* is the first step for preserving the critical path. It is essential that ϵ is chosen such that the addition of constraints to any part of the design other than the *ϵ-critical path* will not result in a mapped network that has a longer delay than the original mapped network. Also, if ϵ is larger than necessary, too many signals will be marked as critical and there will not be sufficient room to embed a sound watermark. In most cases, ϵ values range between *1-3*.

When selecting the set of non critical signals that will be used to introduce our mapping constraints, we use the USERID bit stream discussed earlier. It is important to note that *the total number (n) of the chosen signals on the same path (P) must not be larger than the difference of the length of the path (L_P) and the length of the critical path (L_{CP}), i.e.* $\boldsymbol{n+L_P \leq L_{CP}}$. If this condition is not held, after watermarking, the mapped circuit is guaranteed to have a critical path that is longer than the original mapped circuit.

To verify the watermark, it is sufficient to show that the signature is encoded in the mapped solution and that it corresponds to the text file (or the PGP public key) of the real owner. It is possible to demonstrate that a signature is encoded in a design if there are enough watermarking constraints to show an unusual match. PGP can be used to show that a specific watermark corresponds to the real owner's text file. Mathematically, the probability of a watermark being present in a design can range between 10^{-6} and 10^{-50} depending on the number of constraints and the type of design that is being watermarked. The complexity of the algorithm presented here is bound by the technology mapping step.

We now illustrate this algorithm with a simple example. Consider the graph in ***Figure 4*** that represents a portion of a 2-bounded network. This portion of our sample circuit has *12* primary inputs and *2* primary outputs. Each node in the graph implements a logic function. For simplicity, we show the rest of the network as a single node (…) and assume that the critical path passes through that node.

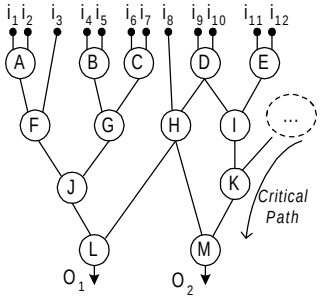


Figure 4 - Algorithm Example

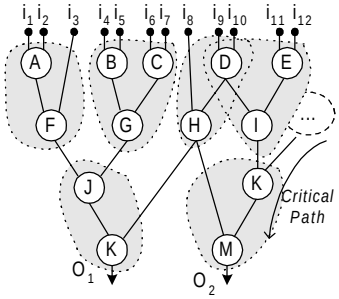


Figure 5 - Mapping Solution With No Watermark

After running the *K-M Mapping* algorithm on our network, we get a solution that is shown in ***Figure 5***. Note that the shaded areas represent a macrocell with *K* inputs implementing a function of at most *M* product terms. Here we have chosen *K=4* and *M=5*.

We identify the *∈-critical path* and assign labels to the non-critical signals ***(Figure 6)***. After assigning a unique label to each non-critical signal in our circuit, we can start the actual watermarking process. Our watermark encoding algorithm identifies the signals that are used to encode the message. For demonstration purposes, let us assume that S_1, S_3, and S_9 are chosen to be marked. We introduce the watermark constraints by breaking the selected signals as a pair of primary output and primary input signals as shown in ***Figure 7***.

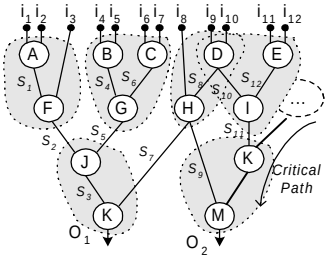


Figure 6 - Assign Labels To Non Critical Signals

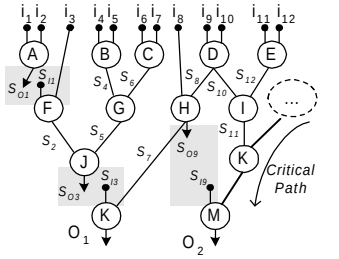


Figure 7 - Introduce Mapping Constraints

We now run the mapping algorithm to obtain the solution in ***Figure 8.*** The new mapping solution produces macrocells that do not contain the signals that we broke in our watermarking step. We can now remove the constraints by patching the signals S_1, S_3, and S_9 to get the result in ***Figure 9.***

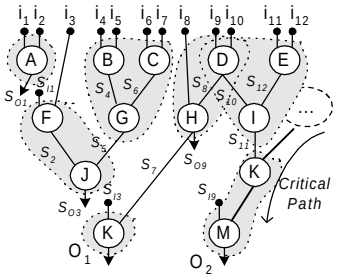


Figure 8 – Technology Mapping After Watermark

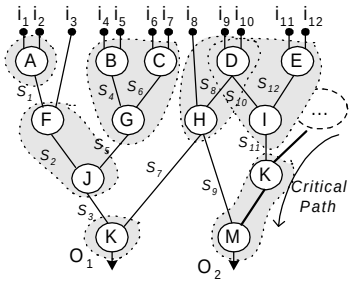
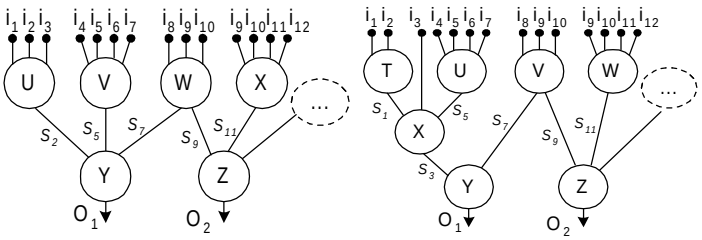


Figure 9 - Final Watermarked Solution



Before watermarking ***After Watermarking***

Figure 10 - Comparison Of Mapping Solutions

There is a well know analogy between the technology mapping step in logic synthesis and the template matching step in behavioral synthesis and compilation. Still, there are two main differences that have important ramifications when a technique from one domain is adapted for application in the other. First, in behavioral synthesis there exists a notion of cycle time, dictated by the critical path of the slowest operation which has to be executed in a single clock cycle. The second difference is that due to time (hardware) sharing the impact of template matching on area is more difficult to predict than in technology mapping.

We applied the new critical path-sensitive watermarking in high level synthesis after two modifications in order to address the differences discussed above. First, we lowered the probability of selecting the nodes in the computation as part of the signature, if as a consequence, a local template with strong area savings is matched. Second, constraints, corresponding to all instances of occurrences of templates, that are most frequent in a computation, get higher correlation of simultaneously receiving or not receiving watermarking constraints. Mathematical analysis indicates that the strength of ownership is improved if the correlation factor is selected to be 0.75.

Our watermarking method for technology mapping in logic synthesis is implemented using C as part of the UC Berkeley SIS package. The source files that implement this algorithm contain about 540 lines of code. The template matching watermarking for behavioral synthesis is implemented using the HYPER package [Rab91].

5. Experimental Results

Table-1 summarizes the experimental results of watermarking the mapping solutions for several different circuits. All the example circuits listed in ***Table-1*** are MCNC benchmark circuits. These circuits have unmapped internal node counts ranging from 696 to 1566 nodes and unmapped depths ranging from 11 to 108 nodes. In addition to depth and node counts, ***Table-1*** also lists the number of primary input / output signals, number of signals that were deemed critical or not, number of added constraints, and the percentage of area increase due to watermark constraints. Some of the circuits contain several registers. Although the registers do not effect the watermarking algorithm, their input and output signals are counted as pseudo primary input/output signals. Each added constraint represents several bits of information that are obtained from the USERID string discussed earlier. The number of bits corresponding to each selected signal depends on the number of non-critical signals determined in the algorithm.

Table-2 lists the results of our watermarking method for template matching behavioral synthesis. The designs include elementary functions such as sine, linear controllers, and various other structures [Rab91]. Although on average the area requirements increased by *1.9%* for template matching (***Table-2***) with a median of *1.2%*, it is interesting to note that about *1/3* of the cases had a decrease in the area requirements. This outcome is a consequence of the complexity of the effects constraints have on template

matching results. In general, it is difficult to predict what effects the addition of the constraints will have on the final area requirements. Consequently, with template matching, although the watermarking algorithm introduces new constraints to the design, the final solution may be more area-efficient than before.

For each circuit listed in ***Table-1***, our algorithm attempted to add a maximum of *20* watermarking constraints. However, as discussed earlier, in some cases this task is impossible. The *apex5* circuit for example, contains only *7* signals that were marked as non-critical, *4* of which could be marked without disturbing the maximum delay of the final network. The area occupied by the mapped circuits increased on average by *2.86%* with a median of *2.04%.*

6. Conclusion

We presented the first watermarking-based IPP technique which guarantees preservation of critical timing constraints in a design. The effectiveness of the technique is demonstrated on technology mapping and template matching. Low area overhead is achieved on all designs, while strong proofs of ownership are established through difficult to detect and remove watermarks.

7. References

[Aho89] A.V. Aho, M. Ganapathi, S.W.K. Tjiang: "Code Generation Using Tree Matching and Dynamic Programming", *ACM Trans. on Prog. Languages and Systems*, Vol. 11, No. 4, pp. 491-516, 1989.

[Aho90] A.V. Aho, "Algorithms for Finding Patterns in Strings." *Handbook of Theoretical Computer Science*, Vol. A, ed. Jan van Leeuwen, Chap. 5, 255-300, 1990.

[Cha98] Charbon, E.: "Hierarchical watermarking in IC design", *Proc. of the IEEE 1998 Custom Integrated Circuits Conference*, 1998. p. 295-8.

[Con94] Cong, J. and Y. Ding, "An Optimal Technology Mapping Algorithm for Delay Optimization in LUT Based FPGA Designs," *IEEE Trans. On Computer-Aided Design*, Vol. 13, pp. 1-12, Jan. 1994.

[Dar81] Darringer, J.A.; Joyner, W.J., Jr.; Berman, C.L.; Trevillyan, L. "Logic synthesis through local transformations (VLSI CAD)". *IBM Journal of Research and Development*, July 1981, vol.25, (no.4):272-80.

[DeG85] Gregory, D.; Bartlett, K.; de Geus, A.; Hachtel, G. "SOCRATES: A system for automatically synthesizing and optimizing combinational logic". *23rd Design Automation Conference.* Proceedings 1986, p. 79-85.

[Hof82] C.W. Hoffman, M.J. O'Donnel: "Pattern Matching in Trees", *Journal of the ACM*, Vol. Vol.29, No. 1, pp. 68-95, 1980.

[Hon98] Hong, I.; Potkonjak, M. "Techniques for intellectual property protection of DSP designs". *Proc. of the 1998 IEEE ICASSP '98*. p.3133-6, vol.5, 1998

[Kah98] Kahng, A.B.; Mantik, S.; Markov, I.L.; Potkonjak, M.; and others. "Robust IP watermarking methodologies for physical design". *Proceedings of 35th Design and Automation Conference*, p. 782-7, 1998

[Keu87] K. Keutzer: "DAGON: Technology binding and local optimization by dag matching", *24th Design Automation Conference*, pp. 341-347, 1987.

[Nik99] Nikolaidis, N.; Pitas, I. "Digital image watermarking: an overview". *Proceedings IEEE International Conference on Multimedia Computing and Systems*. IEEE, 1999. p.1-6 vol.1.

[Not91] S. Note, W. Geurts, F. Catthoor, H. De Man: "Cathedral-III: Architecture-Driven High Level Synthesis for High Throughput DSP Applications", *ACM/IEEE DAC'91*, pp. 597-602, 1991.

[Oli99] A. L. Oliveira, "Robust Techniques For Watermarking Sequential Circuit Design". *1999 DAC Proceedings*, pp. 873-842, 1999.

[Per79] A.S. Perelson, G.F. Oster: "Theoretical Studies of Clonal Selection: Minimal Antibody Repertoire Size and Reliability of Self-Non-Self Discrimination", *Journal of Theoretical Biology*, Vol. 81, pp. 645-670, 1979.

[Per90] A.S. Perelson: "Theoretical Immunology", *1989 Lectures in Complex Systems*, pp. 465-499, Addison-Wesley, Redwood City, CA. Vol. 8, No. 2, pp. 40-51, June 1991.

[Rab91] J. Rabaey, C. Chu, P. Hoang, M. Potkonjak, "Fast Prototyping of Datapath-Intensive Architectures", *IEEE Design and Test of Computers*, Vol. 8, No. 2, pp. 40-51, June 1991.

[Voy99] Voyatzis, G.; Pitas, I. "The use of watermarks in the protection of digital multimedia products". *Proceedings of the IEEE*, July 1999, vol.87, (no.7):1197-207.

Design	# of Nodes	Critical Path cycles	Critical Path ns	Area Increase	Constr. Added
Linear5mat	21	11 / 6	682 / 496	12.7%	7
Wdf8	23	10 / 8	308 / 255	7.2%	8
Volterra	24	14 / 7	830 / 648	4.2%	10
Iir7	33	10 / 6	870 / 570	-2.9%	10
Modem	39	20 / 18	1480 / 630	-1.8%	12
Parallel8	39	9 / 8	612 / 312	6.3%	14
conv5	45	10 / 7	770 / 518	-3.2%	15
Sine	49	16 / 23	992 / 621	0.9%	17
Ladder8	50	34 / 47	2448 / 1457	-2.2%	16
Wavelet	53	14 / 9	1078 / 513	4.3%	22
DS25	62	15 / 7	462 / 255	-3.4%	23
Differ	80	15/7	1406 / 423	3.2%	27
Winfft11	104	11 / 10	847 / 590	4.7%	34
Winfft13	116	11 / 10	847 / 590	1.5%	35
DSKaiser	137	30 / 11	2670 / 748	-0.6%	47
fir100	302	103 / 28	9064 / 2408	0.8%	98

Table 2 – Experimental Results - Watermarking Template Matching For Behavioral Synthesis

Circuit	Unmapped Depth	Unmapped Nodes	Unmapped Input	Unmapped Output	Mapped Depth	Mapped Nodes Before WM	Mapped Nodes After WM	Mapped Critical Signals	Mapped Critical Non-Signals	Mapped Constr. Added	Mapped Area Incr.	Mapped Run Time *
Apex5	11	828	117	88	4	525	527	1158	7	4	0.38%	1.70
Frg2	12	695	143	139	4	442	451	961	50	14	2.04%	1.60
X3	12	768	135	99	4	338	358	218	800	18	5.92%	1.50
I8	12	917	133	81	5	581	586	1295	74	15	0.86%	2.00
Apex6	16	714	135	99	6	338	376	306	634	20	11.24%	1.40
Pair	18	1556	173	137	6	798	801	1682	466	19	0.38%	3.00
Rot	22	696	135	107	8	391	397	465	458	20	1.53%	1.40
9234	22	928	172	175	7	460	472	399	817	19	2.61%	1.90
S5378	24	1322	196	210	9	653	670	366	1403	20	2.60%	2.70
C2670	26	1300	233	64	9	510	523	1500	517	19	2.55%	2.50
MM30A	108	1500	124	121	28	884	896	1227	1235	20	1.36%	3.30

** Run-times reported in seconds on a Sun UltraSparc 5 (333Mhz 128MB RAM)*

Table 1- Experimental Result - Watermarking Technology Mapping For Logic Synthesis

This research was supported in part by NSF under grant CCB-9734166