PROTECTING OWNERSHIP RIGHTS OF A LOSSLESS IMAGE CODER THROUGH HIERARCHICAL WATERMARKING

Hea Joung Kim
Electrical Eng.
U. of California
Los Angeles, CA

William H. Mangione-Smith
Electrical Eng.
U. of California
Los Angeles, CA

Miodrag Potkonjak
Computer Sci.
U. of California
Los Angeles, CA

Abstract - Current market forces make it necessary for designers to protect their work against illicit use. Digital watermarks can be used to sign a design and thus establish ownership. We present a hierarchical set of techniques for intellectual property protection of a linear predictive image coder. Watermarking techniques employed include switching entries in the Huffman coding table, applying zero cost hardware transformations, embedding a signature in the scheduling of shared hardware resources, and applying a watermark to the physical layout. Using these methods, it is possible to watermark complete ASIC or FPGA designs with little overhead in performance or achieved compression rates.

1 INTRODUCTION

Recently, there has been an increased interest in the area of intellectual property (IP) protection due to threat of piracy and counterfeiting. We have developed a hierarchical approach that can be used to watermark a complex block of IP at multiple levels of the design process. These sorts of techniques are evaluated in the context of a lossless image coder [1]. By watermarking the image coding hardware a designer will be able to assert ownership rights in the face of theft, even if sophisticated means are employed. The next few paragraphs will introduce the steps involved (Figure 1).

The assumptions made for the following hardware design is that the images are 256 by 256 arrays of 8-bit gray scale pixels. Furthermore, the training set used to construct the Huffman table for the errors of the linear prediction are F15, Bob, Lena, Football1, Football2, and Trit (Appendix A).

The linear predictive model employed here uses the pixel to the left, diagonal to left and above, above, and diagonal above to the right (Figure 1). This approach facilitates data re-use during raster scan processing. As shown in Figure 1, the first step in constructing a coder is to obtain the error(x,y) distribution of the training set using the model(x,y) and the image(x,y). A linear prediction code is used for the entire image except the first column, first row, and last column of the image, which are predicted using the previous row, column, and row, respectively. The linear prediction can easily be modified for the predictors used in JPEG [2] by performing similar operations on a different set of pixels as described in the JPEG standard.

0-7803-4997-0/98/$10.00

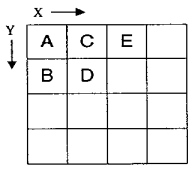


Figure 1. Pixels used for linear prediction error(x,y) = Image(x,y)-Model(x,y) = D – ¼*(A+B+C+E)

An identified set of representative training images is used to generate a probability density function (PDF) for the error values ranging from –128 to 127. Each error value, ranging from –128 to 127, has a probability of occurrence with the 0 error (i.e. the model is correct) having the highest probability. The probabilities are used to generate the Huffman tree as shown in Figure 2. The first step is to get the probabilities of all the error symbols. The second step is to combine symbols having lowest probability into a new symbol node with the original nodes as children. This step repeats until a single tree remains [3].

The Huffman tree encodes each error values to some specific number of code bits. This coding scheme allows low probability errors to be coded with a large number of bits while those with high probability (e.g. '0') get coded with a fewer number of bits. The upper-left pixel and a sequence of variable length error values represent coded images.

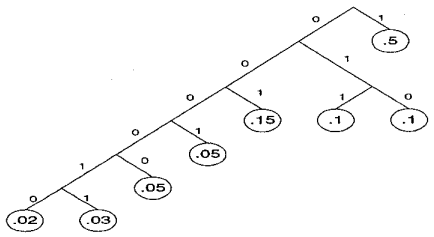


Figure 2. Huffman Tree Example

Watermarks are applied by modifying the table used to construct the Huffman tree, the labels applied to edges, the hardware for computing the model value, the order of computing intermediate terms, and finally the finite state machine (FSM)

1. Get probability distribution function of linear prediction errors for specified training set of images
2. Watermark the symbol (error) table by swapping specific entries or extending the code length and the code itself with edge labeling (depth first search)
3. Watermark hardware using functionally equivalent logic structure
4. Watermark controller using scheduling of shared hardware resources
5. Watermark state machine
6. Watermark physical FPGA design

Figure 3. Design Hierarchies Supporting Watermarks

used for control. Each technique will be discussed in the following sections.

2 RELATED WORK

With the wide spread use of the internet, many valuable digitized images require some form of title certification. Quite a few techniques have been developed for watermarking images [4], video [5], and audio [6]. Unlike this work, these techniques involve watermarking the media, and not the media codec.

There have been numerous papers on linear and non-linear prediction for lossless image compression [7,8] from different application domains. Most of these papers describe the different *x,y* coordinate pixels to use to do linear prediction more efficiently for the images of different types, e.g. medical images, synthetic aperture radar images, infrared images, and others. All these papers present a method to predict the errors from a smaller range so the prediction is easier. The techniques presented here are directly applicable to this broader domain.

Recent research results have investigated a range of hardware watermarking techniques [9,10] at a number of distinct levels. The work present here is the first multi-level system, and the first attempt to watermark an application algorithm (i.e. linear prediction image compression).

3 APPROACH

The following sections will describe all the steps involved in a complete top-down watermarking solution.

3.1 Watermarking the Compression Code

The first step is to create a look-up table for the Huffman coding of the error(x,y) distribution. The Huffman table is generated by the summed errors of the training set images. The probability density function is calculated to generate the Huffman table of the image collection. The PDF of each error (ranging from –128 to 127 given an 8-bit gray scale image) is found using a C program. Following the algorithm for constructing the Huffman code, the pair of the least probable errors is iteratively combined to generate the Huffman tree.

An arbitrary sequence of length 36 and 72 signature can embedded into the Huffman table. The simple approach is to take the first 8-bits of the signature to select one of the probabilities of the error symbols listed in decreasing order. Then if the ninth bit is a 1, the probabilities of the error below and above the 8-bit number are swapped. If the ninth bit is a 0, the probabilities of the selected entry and the immediately preceding locations are switched. If the code length is the same for the two probabilities that will be switched, the error code is extended by one bit to embed a '0' or '1'. Thus, this approach may result in increasing the code length of certain error symbols. Figure 4 shows the method described.

1. Take the first eight bits to get the location of decreasing probabilities of the error symbols
2. Take the ninth bit to swap the probabilities and error code (if the error code is the same length swap and extend 1 bit)
3. 01110100,0 (116,0) the 116^{th} and 117^{th} are switched 116^{th} probability code is 11 bits and is extended to 12 with the 12^{th} bit being a '0'

4. 00000011,1 (3 , 1) the 2nd and 4th are swapped and 3rd probability code is lengthened to 6bits with the 6th bit being a '1'
5. 10100000,1 (160,1)
6. 00000000,1 (neglect...too expensive to change location '0' or '1')
7. 11010000,0 (208,0)
8. 00000010,1 (2,1) ..neglect since duplicated
9. 00000111,0 (7,0)
10. 00011110,0 (30,0)

Figure 4. Watermarking the Compression Code

Loc	Size	error	trit	lena	fball2	fball1	F15	bob	prob
0	5	0	28795	27315	35140	34935	54045	101095	0.14308
1	5	255	9120	24160	14870	14710	10105	5270	0.03979
2	5	1	8910	23335	13535	14470	1510	8890	0.03593
3	5	254	8255	21580	13420	13440	2440	8270	0.03428
4	5	4	8425	12855	10280	10335	15845	8655	0.03377
5	5	5	7560	10120	9275	9075	24360	5580	0.03355
6	5	2	8525	19700	12590	12730	1905	6980	0.03175
7	5	250	8540	7910	8965	8630	19685	7460	0.03112
8	5	253	8585	17065	11875	12630	895	9915	0.03100
9	5	251	7530	9605	9985	10040	14565	6665	0.02969

Table 1. Original Huffman Table with no watermarking

Loc	Size	error	trit	lena	Fball2	fball1	F15	bob	prob
0	5	0	28795	27315	35140	34935	54045	101095	0.143
1	5	255	9120	24160	14870	14710	10105	5270	0.039
1	1	1	8925	12855	10280	10335	15845	8655	0.033
3	6*	254	10692	28002	16242	17364	1812	10668	0.034
1	1	1	9210	15335	13535	14470	1510	8890	0.035
5	5	5	7560	10120	9275	9075	24360	5580	0.033
7	6*	250	10248	9492	10758	10356	23622	8952	0.031
1	1	1	8525	19760	12590	12730	1905	6980	0.031
8	5	253	8585	17065	11875	12630	895	9915	0.031
9	5	251	7530	9605	9985	10040	14565	6665	0.029

Table 2. Huffman Table with Watermarking

Table 1 and Table 2 show the exact procedures taken to embed a signature in the Huffman table for the two examples listed in Figure 4. The first example is where the first 8-bits are a 3 and the 9th bit is a '1', while the second example is where the first 8-bits are a 7 and the 9th bit is a '0'.

The assignment of the zero and one to the Huffman tree edges can be used to watermark the codes of the error symbols. We set a rule that two nodes emerging from an interior node will be sorted so that the sub-tree with the lowest probability will be assign an edge value of zero. If the actual edge assignment follows this

rule it encodes a '0', otherwise it encodes a '1'. Thus, if the training set is known, a signature can be retrieved through systematic probing.

3.2 Watermarking the Datapath

The hardware is watermarked at multiple stages of the design (Appendix B). The first step involves transformations as shown Figure 2. The model(x,y) is obtained by ¼*(Image(x-1,y-1) + Image(x-1,y) + Image(x+1,y) + Image(x,y-1)). The basic operations needed are to shift and add or to add and shift.

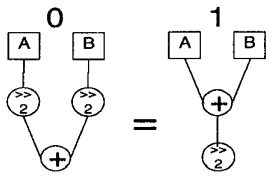


Figure 5. Transformations for embedding 0 or 1

Figure 5 shows that a shift-add operation designates a '0' and an add-shift operation designates a '1'. A parallel datapath can be used to embed a 10-bit signature as shown on Figure 6. The hardware markings discussed in sections 3.2, 3.3, and 3.4 are more difficult to extract after a design appears in the field. Unlike modifications to the coding structure, hardware marks cannot easily be extracted through external probing. Nonetheless, techniques do exist for extracting RTL and datapath information using optical inspection [11].

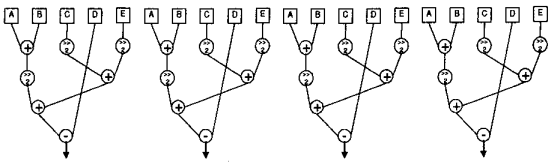


Figure 6. Signature '10101010'

3.3 Watermarking the Control Structure

The next form of signature embedding is to use scheduling to insert artificial dependencies between operations in the datapath (Figure 7). An operation delayed even when it could have been executed in parallel embeds a '1' (operations –gray filled - Figure 7), while an operation that follows the normal execution flow embeds a '0'.

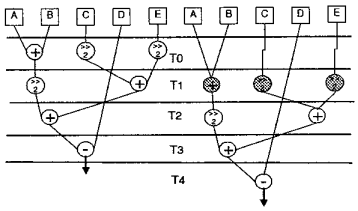


Figure 7. Scheduling to embed "000111" a signature

3.4 Watermarking the Finite State Machine

Following hardware transformations and datapath scheduling, another signature can be embedded using the states for the finite state machine that controls the flow graph of the operations shown above. The state numbers for the state machine can be created from an arbitrary sequence of numbers.

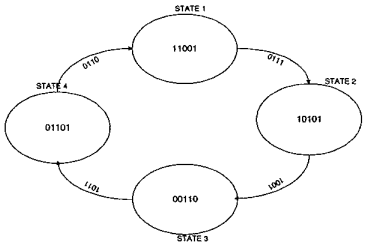


Figure 8. Finite State Machine used to embed signature

Figure 8 shows exactly how another signature can be embedded in the design. This is not a very rigorous or undetectable signature but it would be easy to demonstrate ownership if the design is copied with no thought. The signature would read '11001->->10101->->00110->->01101'.

The entire design would be specified in Verilog and synthesized to an FPGA.

4 EXPERIMENTAL RESULTS

The experimental results available are for the Huffman table used for compression of the image data and the cost associated with embedding a signature. The raw performance rates are not interesting given that any hardware implementation will be faster than a software implementation. The Verilog code embeds signatures using scheduling, transformations, states in a FSM, and jump conditions signatures. Thus, the focus of the experimental results is on the cost of coding an image using the Huffman table with a signature embedded in the table.

The first set of data shows the best compression possible assuming that errors with value greater than 20 can be coded using 20 bits. Despite the fact that it is possible to code any error greater than 10 with a fixed bit-length of 9, the method of coding errors greater that 20 bits can be just as effective in showing the cost of embedding a signature in the Huffman table.

Figure 9 shows the different sizes of the compressed images for the different types of images. The best code represents a Huffman tree generated using each images probability density function to compress the same image. The summed Huffman is the code generated from the sum of all the PDFs of the images. The 36-bit signature is the size of the compressed image given that 36 bits of data has been embedded into the summed Huffman code using the method described in Figure 4. Figure 9 shows a graph comparing the results.

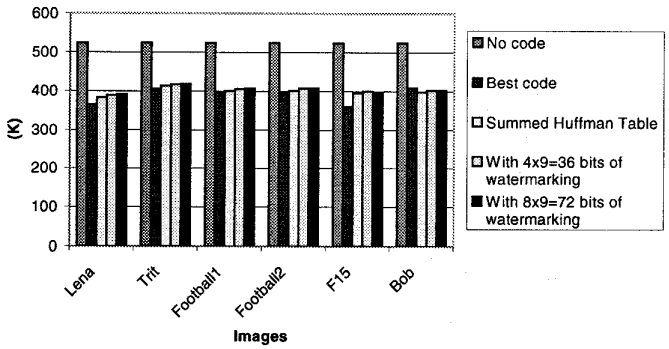


Figure 9. Comparison of image coders (uncompressed, compressed, and watermarked)

Figure 9 and Figure 10 show the percentages that the image sizes increases or decreases relative to the summed Huffman table, non-watermarked, and non-compressed images. It is important to note that the cost of watermarking is less than 2% of the compressed image size and in some cases even better since the summed Huffman table is not optimal for a specific image. In some instances, the size is smaller after the watermarking due to the switched probabilities from watermarking the code table.

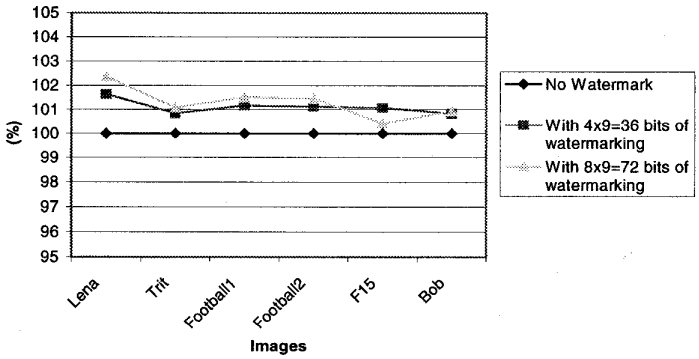


Figure 10. Cost of Watermarking the Huffman codes

Finally, the layout generated on an FPGA by synthesizing the Verilog code is shown in Figure 11. The critical path is 40ns without optimizations, and the clocking frequency is 25MHz. The empty CLBs (gray spots) can be used for embedding signatures as describe in [9,10].

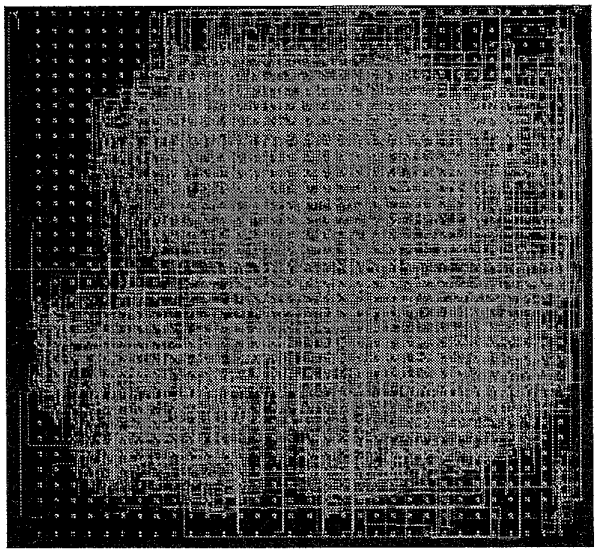


Figure 11. Layout from the Xilinx FloorPlanner for Linear Prediction Coder Circuit

Watermarking Method	Number of Signature	Cost
PDF manipulation for Huffman code	32 or 72	Maximally 2% inc. file size
Huffman code edge labeling	20 ~ 256 (theor. max)	None
Hardware Datapath Transformation	8 (4 pixels processed) ~??	No perform. loss (more gates)
Hardware Datapath Scheduling	6(1 bit per depend)	20% performance loss
State Machine	40 bits (5 bits per state)	More registers
FPGA physical layout	16 bits per CLB LUT	Reserved or unused CLBs

Table 3. Signatures and cost of signature for specified algorithm

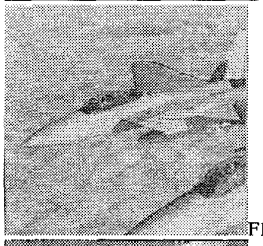
Table 3 is a brief summary of all the bits embedded in the hierarchical design. The Huffman table is manipulated to embed either a 36 or 72 bit signature. The edge labeling embeds another 20 or more bits depending on the maximum code length for the error symbol with no negative impact. The transformations embed 8 bits of signature that can embed more bits if more image pixels are processed in parallel. The hardware scheduling decreases the performance due to the execution delay resulting from the artificial dependencies. More bits are embedded with more parallel execution. The finite state machine is used to embed 40 bits and the associated cost is more registers. The FPGA configurable logic blocks (CLB) are used to embed as many bits a possible in the FPGA physical layout.

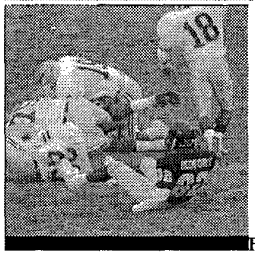
5 CONCLUSION

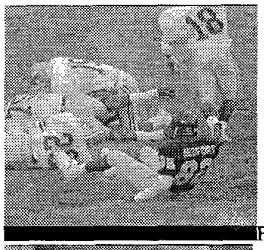
Based on the results, we can cheaply and efficiently watermark a lossless linear prediction hardware using the Huffman table. The cost of adding a 72-bit signature in the Huffman code is at most 2% of the compressed image. The technique is simple enough to incorporate. Furthermore, the hardware methods allow for a "without reasonable doubt" conclusion that the design is that of the owners. The techniques used here are not limited to FPGAs and can be applied to ASICs. Given that it took some engineers a few weeks to reverse engineer an Intel 386 [11], it would be easy for others to copy one's hardware. By using the techniques developed here, one may embed a 20-bit signature. This 20-bit signature may be sufficient to prove in the court of law that the design is proprietary. Using 20-bits, there is a 1 in 1,000,000 chance that the next designer could use the same set of transformations, scheduling, and state machine watermarking. These efforts show that watermarking a lossless linear prediction for lossless image compression hardware and the Huffman table is simple and effective.

Appendix A: Images

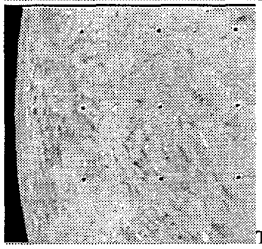

Bob


F15


Fball1


Fball2


Lena


Trit

References

[1] H.G. Mussman. Predictive Image Coding. *Advances in Electronics and Electron Physics, Supplement. 12*, pages 73-112. Academic Press Inc. 1979.

[2] G.K. Wallace. The JPEG Still Picture Compression Standard. *Communications of the ACM*. 34(4):31-44, 1991.

[3] M. Nelson and J-L. Gailly. *The Data Compression Book (second edition)*. M&T Books, 1996.

[4] I.J. Cox, J. Kilian, T. Leighton, T. Shamoon, "Secure spread spectrum watermarking for images, audio and video", *International Conference on Image Processing*, 1996, vol. 3, pp. 243-246.

[5] F. Hartung and B. Girod,"Watermarking of MPEG-2 encoded video without decoding and re-encoding", *Multimedia Computing and Networking*,1997, pp. 264-274.

[6] L. Boney, A.H. Tewfik, and K.N. Hamdy, "Digital watermarks for audio signals", *International Conference on Multimedia Computing and Systems*,1996, pp. 473-480.

[7] P. G Howard and J.S. Vitter. Fast and efficient lossless compression. *Proceedings of the Data Compression Conference*, pages 351-360. IEEE Computer Society Press, 1993.

[8] N. D. Memon, K. Sayood, and S. S. Magliveras. "Lossless Image Compression – a comparative study." *Still Image Compression*, pages 8-20. SPIE Proceedings Volume 2418, 1995.

[9] J. Lach, W. H. Mangione-Smith, and M. Potkonjak. "FPGA Fingerprinting Techniques for Protecting Intellectual Property," *Custom Integrated Circuits Conference*, 1998.

[10] J. Lach, W. H. Mangione-Smith, and M. Potkonjak. "Fingerprinting Digital Circuits On Programmable Hardware." *International Workshop in Information Hiding*, 1998.

[11] R. Anderson and M Kuhn, "Tamper Resistance – A Cautionary Note", *USENIX Workshop on Electronic Commerce*, Nov. 1996.