VLSI Implementation of a Real-Time Video Watermark Embedder and Detector

Nebu John Mathai, Ali Sheikholeslami, and Deepa Kundur

Dept. of Elec. and Comp. Eng., University of Toronto,

10 Kings College Rd., Toronto, Canada M5S 3G4

Email: {mathai@ecf, ali@eecg, deepa@comm} .utoronto.ca

ABSTRACT

This paper describes the hardware design and implementation of the JAWS (Just Another Watermarking System) embedder and detector for watermarking of real-time uncompressed digital video. Our design employs a floating point datapath, maximizing dynamic range of the FFT, frame filtering, and correlation operations. The design, implemented in a 1.8V, 0.18µm CMOS process with a core area of 3.53 mm^2, is capable of watermarking video streams at a peak rate of over 3 Mpixels/s.

1. INTRODUCTION

Digital watermarking is the process of embedding a message, called the payload, within the content of a host message, where the host message can be audio, still images, or video. Watermarking systems for audio and still imagery are often implemented in software due to the low data rate of these signals. For video streams, however, real-time watermarking is too expensive to implement in software, and hence there is a strong motivation for hardware implementation.

This paper describes a VLSI implementation of the JAWS embedder and detector algorithm [1, 2, 3]. JAWS is a well-known and respected system that has found interest in both academic and industrial research circles. Thus, it is an ideal case study for our implementation.

A testchip is designed in 0.18µm CMOS technology to embed and detect watermarks in uncompressed real-time video streams at a rate of 30 frames/s and 320x320 pixels/frame. The testchip is capable of operating at 75 MHz, to process a peak pixel rate of over 3 Mpixels/s. As this work is the first step toward analyzing the relationship between watermarking algorithmic features and implementation cost for practical systems, this provides us with a useful vehicle for study without incurring excessive hardware costs.

2. WATERMARK EMBEDDER

The first stage of the JAWS embedder, shown in Figure 1,

This work was supported by NSERC.

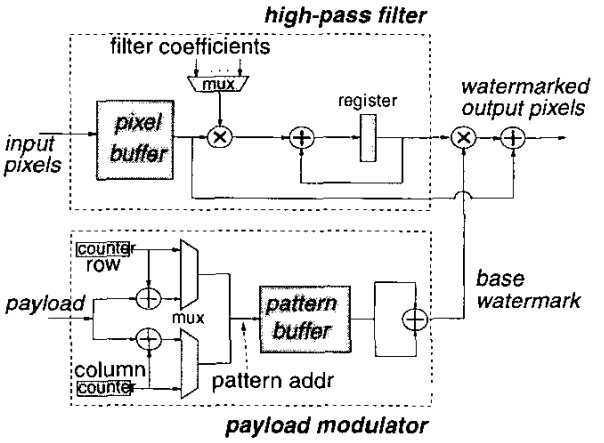

Figure 1. Watermark Embedder Architecture

high-pass filters each input frame to determine the regions of the image that can be imperceptibly distorted. The input payload is used to modulate a pseudo-random noise pattern (arranged as a 32x32 pixel matrix and called the primitive pattern) to generate a base watermark, which is then point-wise multiplied with the filtered output to create an imperceptible watermark containing the payload. This final watermark is then pixel-wise added to the original video frame to generate the output.

To achieve high throughput, we choose a pipelined architecture where the data stream through the pipeline is a stream of pixels forming a row-by-row raster of each frame.

For the filter, a buffer is used to store three rows of pixels from the input video frame: the current row and its two adjacent rows. The input port fills the pixel buffer. Once full, the input is stalled, and one row of filtered elements are computed. For each pixel, nine elements are read from the buffer, multiplied with the appropriate filter coefficient, and accumulated. After the entire filtered row has been forwarded to the rest of the embedder, the input port resumes filling the pixel buffer.

In JAWS, payload modulation is accomplished by encoding the payload data in the positional displacement between two patterns generated from the primitive pattern (in the pattern

I-7803-7761-3/03/$17.00 ©2003 IEEE

(a) (b) (c)

Figure 2. (a) Unwatermarked image, (b) strongly watermarked image (with no filtering) and (c) imperceptibly watermarked image

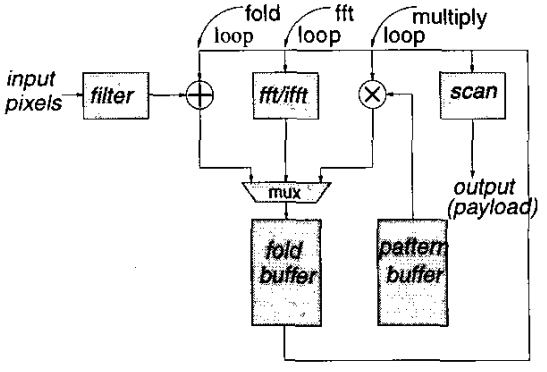

Figure 3. Watermark Detector Architecture

buffer). The first pattern is the primitive pattern itself. In the second pattern, each pixel of the primitive pattern is displaced a certain pixel offset row- and column-wise (these offsets are a function of the payload). As the figure shows, two counters are used to address the primitive pattern elements. The payload data word is then separated into row and column components, and added to the row and column counters to access the displaced element. The two elements are added together to create the modulated pattern which is then replicated over the span of the video frame (by counter rollover).

Figure 2(a) shows hardware simulation results of the embedder. The first image is the original (unwatermarked) image input to the embedder. Figure 2(b) shows the output of the embedder with the filter disabled; the regular pattern tiled across the image is the result of the payload modulation circuit. Finally, Figure 2(c) shows the output of the watermark embedder with the filter enabled; the embedded watermark is imperceptible (the PSNR between this image and the original is 40 dB).

3. WATERMARK DETECTOR

The detector, shown in Figure 3, filters all frames with a whitening filter to improve detector performance [1]. The topology of this filter is identical to the high-pass filter of the embedder, with the only difference being the filter coefficient values. After being filtered, the input stream enters the fold

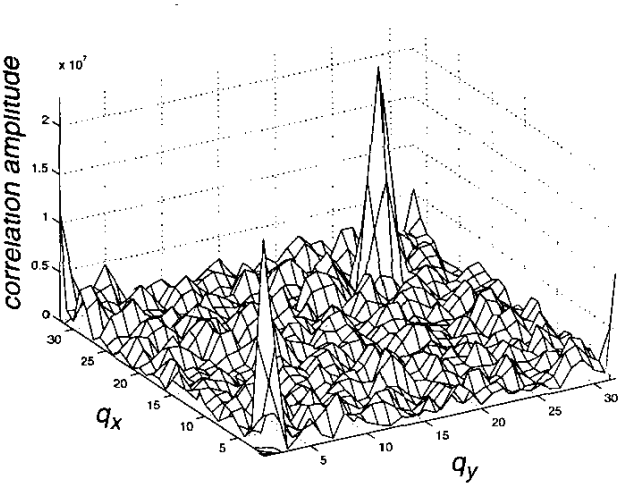

Figure 4. IFFT Computation Output

loop, consisting of an adder and the fold buffer RAM. This loop causes primitive pattern-sized blocks of the filtered frame to be accumulated into the fold buffer. Once this process is complete, the input port is stalled and the rest of the detection process begins.

The Fast Fourier Transform (FFT) loop is first selected and the 2D FFT of the fold buffer is computed. A single decimation-in-time butterfly circuit is used for all FFT passes. The frequency-domain result is then pixel-wise multiplied with the frequency-domain primitive pattern. Finally, the FFT loop is selected, but this time the "twiddle" coefficients are conjugated to calculate the inverse FFT (IFFT). In parallel with the last pass through the IFFT, the scan hardware is enabled. This is a set of two registers that keep track of the two highest correlation peaks output by the IFFT. These peaks represent the detection of the primitive pattern and its displaced version.

Figure 4 shows the output of the IFFT computation where q_x and q_y denote the number of pixel shifts of the primitive pattern in the x and y directions. Two detection "peaks" are observed: one at $q_x = q_y = 0$ corresponding to the detection of the primitive pattern, and another corresponding to the detection of the displaced pattern. The difference in position between these two peaks are used to calculate the payload.

The detector was run with all three inputs shown in Figure 2. The detector correctly detected the absence of payload in the unmarked image. For the two marked images, the correct payload was detected.

4. NUMERICAL REPRESENTATION

One of the main challenges for watermarking algorithm

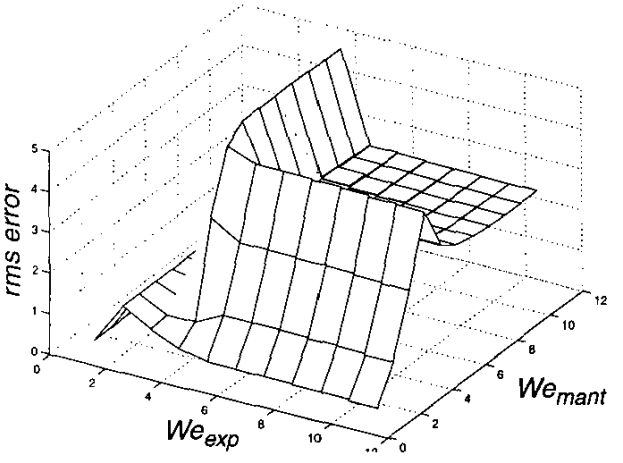

Figure 5. The effect of floating point precision on the perceptibility of the embedded watermark

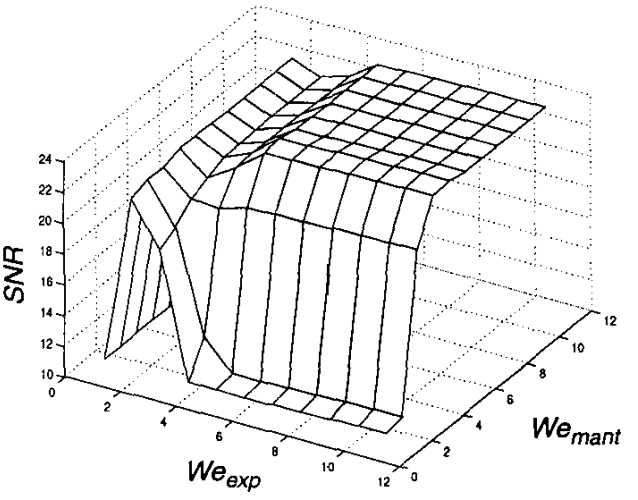

Figure 6. The effect of floating point precision on the detectability of the embedded watermark

design is determining suitable numerical representations for the datapath [2]. Due to the large dynamic range required for watermarking computations, the word size of the internal datapath must be carefully chosen to maximize algorithm performance and minimize hardware cost (the area and power of the datapath being roughly proportional to floating point word size). For flexibility, we choose a floating point datapath and use simulation results to determine appropriate word sizes. Algorithm performance is evaluated based on the imperceptibility of the watermark (measured quantitatively by the rms error between the unmarked and marked frames), and the detection SNR (where the IFFT detection peaks are considered to be the signal, and background detection values to be noise).

To evaluate the effect of word size on the embedder, we vary the mantissa width, We_{mant}, and exponent width, We_{exp}, independently, and measure the perceptibility of the watermark (Figure 5), and the detectability of the embedded watermark (Figure 6). These plots show three regions with common characteristics. First, there is a region where the watermark is most imperceptible (and undetectable). This corresponds to where the dynamic range is insufficient to represent a strong watermark. Second, there is a region where detection is much better, but the watermark is perceptible. This corresponds to where floating-point numbers behave like fixed point numbers; the poor granularity of the represented embedding depths results in a stronger embedded watermark. Finally, there is a third region where perceptibility is minimized, and detection performance maximized. Also note that these performance metrics are roughly constant throughout the region. The border of this region and the

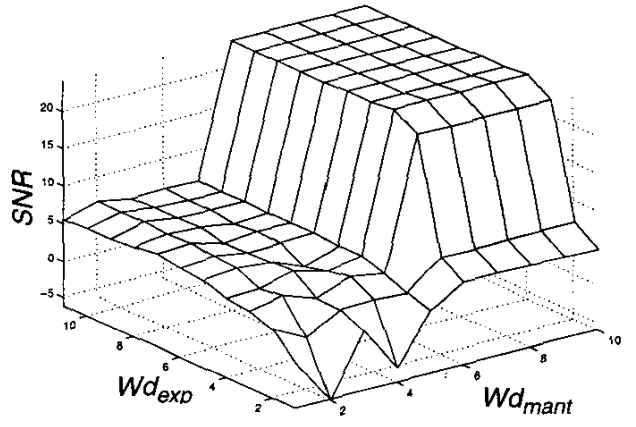

Figure 7. The effect of floating point precision on detection

second defines the point beyond which increasing word size ceases to enhance performance.

Word size also plays an important role in the detector where, due to the iterative nature of the detection computations, the numbers may grow in size. To illustrate the effect of word size on the detector, we vary the mantissa width, Wd_{mant}, and the exponent width, Wd_{exp}, and measure the performance of the detector on a frame where the watermark is imperceptible. In Figure 7, two regions can be seen: one where the detector fails due to insufficient dynamic range, and another where the detector succeeds and performs the same with further increases in precision.

Table 1. Watermarking testchip features

clock	75MHz
process	0.18μm CMOS
core size	1.88mm x 1.88mm
power (embedder)	60 mW
power (detector)	100 mW
payload/frame	4 bits
each frame	320 x 320 pixels
frame rate	30 frames/s
max. pixel rate	3 Mpixels/s
pixel buffer size	960 word by 16-bits
fold buffer size	1024 word by 16-bits
pattern buffer size	1024 word by 16-bits

The results of these simulations require a floating point representation with a 6-bit exponent and 8-bit mantissa to attain the highest performance (close to ideal). Since our available SRAM macro was 16-bits wide, we used 16-bit floating point words (6-bit exponent, 10-bit sign and magnitude mantissa) for the entire datapath of the testchip.

CONCLUSION

We have demonstrated a watermarking embedder and detector that can process uncompressed digital video in real time. The testchip die photo indicating the blocks of the embedder and detector is shown in Figure 8, along with a summary of the testchip features in Table 1.

ACKNOWLEDGMENTS

The authors would like to thank CMC for facilitating the chip fabrication, and NSERC of Canada for financial support.

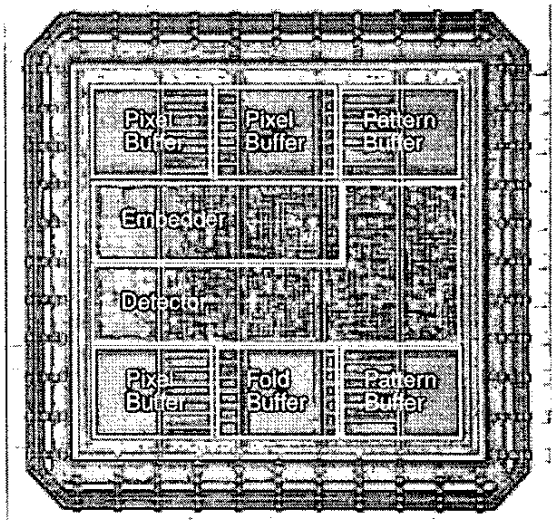

Figure 8. Watermarking testchip die photo

REFERENCES

[1] T. Kalker, G. Depovere, J. Haitsma, and M. Maes, "A video watermarking system for broadcast monitoring," *Proc. SPIE, Security and Watermarking of Multimedia Contents*, Vol. 3657, pp. 103-112, Jan. 1999.

[2] L.De Strycker, P. Termont, J. Vandewege, J. Haitsma, T. Kalker, M. Maes, and G. Depovere, "Implementation of a real-time digital watermarking process for broadcast monitoring on a TriMedia VLIW processor", *IEE Proc. Vision, Image and Signal Processing*, Vol. 147, August 2000.

[3] M. Maes, T. Kalker, J. Haitsma, and G. Depovere, "Exploiting shift invariance to obtain a high payload in digital image watermarking," *Proc. IEEE Int. Conference on Image Processing*, Vol. 2, pp. 7-12, October 1999

[4] G. Petitjean, J. L. Dugelay, S. Gabriele, C. Rey and J. Nicolai, "Towards Real-Time Video Watermarking for System-on-Chip", *Proc. IEEE Int. Conf. on Multimedia and Expo*, 2002, to appear.