VLSI Implementation of Visible Watermarking for a Secure Digital Still Camera Design

Saraju P. Mohanty, N. Ranganathan and Ravi K. Namballa
Dept. of CSE, University of South Florida, Tampa, FL 33620
{smohanty,ranganat,rnamball}@csee.usf.edu

Abstract

Watermarking is the process that embeds data called a watermark into a multimedia object for its copyright protection. The digital watermarks can be visible to a viewer on careful inspection or completely invisible and cannot be easily recovered without an appropriate decoding mechanism. Digital image watermarking is a computationally intensive task and can be speeded up sigificantly by implementing in hardware. In this work, we describe a new VLSI architecture for implementing two different visible watermarking schemes for images. The proposed hardware can insert on-the-fly either one or both watermarks into an image depending on the application requirement. The proposed circuit can be integrated into any existing digital still camera framework. First, separate architectures are derived for the two watermarking schemes and then integrated into a unified architecture. A prototype CMOS VLSI chip was designed and verified implementing the proposed architecture and reported in this paper. To our knowledge, this is the first VLSI architecture for implementing visible watermarking schemes.

1 Introduction

Watermarking is the process that embeds data called a watermark, tag or label into a multimedia object such that watermark can be detected or extracted later to make an assertion about the object. The object may be an image, audio, video, or text. In general, any watermarking scheme consists of three parts, such as, the watermark, the encoder and the decoder. The marking algorithm incorporates the watermark into the object, whereas the verification algorithm authenticates the object determining both the owner and the integrity of the object. The watermarks can be applied either in spatial domain or in frequency domain. According to human perception, the digital watermarks can be divided into four categories : visible watermark, invisible-robust, invisible-fragile and dual [1, 2]. A visible watermark is a secondary translucent overlaid into the primary image and appears visible to a viewer on careful inspection.

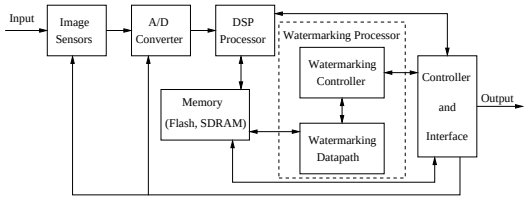


Figure 1: System Architecture of a Secure Digital Still Camera

Several software based watermarking schemes have been presented in the literature; however, only a few hardware schemes have been proposed. Strycker, et. al. [4] proposed the implementation of a real-time spatial domain watermark embedder and detector on a Trimedia TM-1000 VLIW processor. Mathai, et. al. [5] present a chip implementation of the same video watermarking algorithm. A DCT domain invisible watermarking chip is presented by Tsai and Lu [6]. Garimella, et. al. [7] proposed a VLSI architecture for invisible-fragile watermarking in spatial domain. Mohanty, et. al. [8] described a watermarking chip that has spatial domain invisible robust and fragile watermarking functionalities.

In this work, we focus on the VLSI implementation of two spatial domain visible watermarking schemes, one proposed by Braudaway, et. al. [9] and the other by Mohanty, et. al. [3]. The VLSI chip can insert either one or both the watermarks depending on the requirement of the user. The proposed watermarking chip can be integrated within any existing digital still camera. We provide the schematic view of a still camera that includes a watermarking module in Fig. 1, and call such a camera as a "secure digital still camera".

2 Watermarking Algorithms

In this section, we outline the watermarking algorithms in brief with the modifications needed for hardware implementation. The notations listed in Table 1 are needed for describing the algorithms.

Table 1: List of Variables used in Explanation

I	: Original (or host) grayscale image
W	: Watermark image (a grayscale image)
(m, n)	: A pixel location
I_W	: Watermarked image
$N_I \times N_I$	: Original image dimension
$N_W \times N_W$	: Watermark image dimension
i_k	: k^{th} block of the original image I
w_k	: k^{th} block of the watermark image W
$i_{W\,n}$	: k^{th} block of the watermarked image I_W
α_k	: Scaling factor for k^{th} block
β_k	: Embedding factor for k^{th} block
μ_I	: Mean gray value of the original image I
$\mu_{k\,I}$	: Mean gray value of image block i_k
$\sigma_{k\,I}$	: Variance of the original image block i_k
α_{max}	: The maximum value of α_k
α_{min}	: The minimum value of α_k
β_{max}	: The maximum value of β_k
β_{min}	: The minimum value of β_k
I_{white}	: Gray value corresponding to white pixel
α_I	: A global scaling factor
C_1, C_2	: Linear regression co-efficients
C_3, C_4	: Linear regression co-efficients

Algorithm 1 (Proposed in [9]) : The watermarked image is obtained by adding a scaled gray value of the image to the host image. The amount of scaling is done in such a way that the alternation of each original image pixel occurs perceptually by equal degree. The original formula is simplifed as follows, where the scaling factor α_I determines the strength of the watermark [10].

$$I_W(m,n) = \begin{cases} I(m,n) + W(m,n)\left(\frac{I_{white}}{38.667}\right)\left(\frac{I(m,n)}{I_{white}}\right)^{\frac{2}{3}}\alpha_I \\ \qquad\qquad \text{for } \frac{I(m,n)}{I_{white}} > 0.008856 \\ I(m,n) + W(m,n)\left(\frac{I(m,n)}{903.3}\right)\alpha_I \\ \qquad\qquad \text{for } \frac{I(m,n)}{I_{white}} \leq 0.008856 \end{cases} \quad (1)$$

The above equation is further simplified to make the hardware implementation easier. At the same time, care is taken to make sure that the hardware is as accurate as the software implementations. We assume $I_{white} = 255$ and simplify the above equations to the following.

$$I_W(m,n) = \begin{cases} I(m,n) + \left(\frac{\alpha_I}{6.0976}\right) W(m,n)(I(m,n))^{\frac{2}{3}} \\ \qquad\qquad \text{for } I(m,n) > 2.2583 \\ I(m,n) + \left(\frac{\alpha_I}{903.3}\right) W(m,n) I(m,n) \\ \qquad\qquad \text{for } I(m,n) \leq 2.2583 \end{cases} \quad (2)$$

We further simplify the above expressions and remove the cubic root function with a piecewise linear model. We divide the gray values range $[0,\, I_{white}]$ to four ranges, such as $\left[0,\, \frac{I_{white}}{4}\right]$, $\left[\frac{I_{white}}{4},\, \frac{I_{white}}{2}\right]$, $\left[\frac{I_{white}}{2},\, \frac{3I_{white}}{4}\right]$, and $\left[\frac{3I_{white}}{4},\, I_{white}\right]$. We fit four linear regression co-efficients that best approximates the cubic root in each of these ranges. Moreover, we roundup the fraction involved in the comparison operation and the final simplified expression that is implemented using hardware is as follows.

$$I_W(m,n) = \begin{cases} I(m,n) + \left(\frac{\alpha_I}{903.3}\right)\ W(m,n)\ I(m,n) \\ \qquad\qquad \text{for } I(m,n) \leq 2 \\ I(m,n) + \left(\frac{\alpha_I C_1}{6.0976}\right)\ W(m,n)\ I(m,n) \\ \qquad\qquad \text{for } 2 < I(m,n) \leq 64 \\ I(m,n) + \left(\frac{\alpha_I C_2}{6.0976}\right)\ W(m,n)\ I(m,n) \\ \qquad\qquad \text{for } 64 < I(m,n) \leq 128 \\ I(m,n) + \left(\frac{\alpha_I C_3}{6.0976}\right)\ W(m,n)\ I(m,n) \\ \qquad\qquad \text{for } 128 < I(m,n) \leq 192 \\ I(m,n) + \left(\frac{\alpha_I C_4}{6.0976}\right)\ W(m,n)\ I(m,n) \\ \qquad\qquad \text{for } 192 < I(m,n) < 256 \end{cases} \quad (3)$$

Algorithm 2 (Proposed in [3]) : The pixel gray values are modified based on the local and global statistics. The watermaking insertion process consists of the following steps. Both the host image (one to be watermarked) I and the watermark (image) W are divided into blocks of equal sizes (the two images may be of unequal size). Let i_k denote the k^{th} block of the original image I and w_k denote the k^{th} block of the watermark W. For each block (i_k), the local statistics; mean $\mu_{k\,I}$ and variance $\sigma_{k\,I}$ are computed. The image mean gray value μ_I is also found out. The watermarked image block is obtained by modifying i_k as follows.

$$i_{W\,k} = \alpha_k\, i_k\ +\ \beta_k\, w_k \qquad k = 1, 2... \quad (4)$$

Where, α_k and β_k are scaling and embedding factors respectively, depending on $\mu_{k\,I}$ and $\sigma_{k\,I}$ of each host image block.

The choice of α_k and β_k are governed by certain characteristics of human visual system (HVS) and mathematical models are proposed so that the perceptual quality of the image are not degraded due to watermark addition. The α_k and β_k are obtained as follows. The α_k and β_k for edge blocks are taken to be α_{max} and β_{min} respectively. The α_k and β_k are found out using the following equations.

$$\begin{aligned} \alpha_k &= \tfrac{1}{\hat{\sigma_{k\,I}}} exp\left(-(\hat{\mu_{k\,I}} - \hat{\mu_I})^2\right) \\ \beta_k &= \hat{\sigma_{k\,I}}\left(1 - exp\left(-(\hat{\mu_{k\,I}} - \hat{\mu_I})^2\right)\right) \end{aligned} \quad (5)$$

Where, $\hat{\mu_{k\,I}}$ and $\hat{\mu_I}$ are normalised values of $\mu_{k\,I}$ and μ_I, and $\hat{\sigma_{k\,I}}$ are normalised logarithm values of $\sigma_{k\,I}$. The α_k and β_k are scaled to the ranges (α_{min},α_{max}) and (β_{min}, β_{max}) respectively, where α_{min} and α_{max} are minimum and maximum values of scaling factor, and β_{min} and β_{max} are minimum and maximum values of embedding factor. These parameters determine the extent of watermark insertion. A linear transformation is used to scale current α_k and β_k values to the ranges (α_{min}, α_{max}) and (β_{min}, β_{max}), respectively.

Let current values of α_k be written as α_k^c, and α_{min}^c and α_{max}^c, respectively denote the current minimum and maximum values. Similarly, let current values of β_k be written as β_k^c, and β_{min}^c and β_{max}^c, respectively denote the current minimum and maximum values. The α_k and β_k values are scaled as follows.

$$\alpha_k = \left(\frac{\alpha_{max}-\alpha_{min}}{\alpha_{max}^c-\alpha_{min}^c}\right)\alpha_k^c + \left(\alpha_{max} - \left(\frac{\alpha_{max}-\alpha_{min}}{\alpha_{max}^c-\alpha_{min}^c}\right)\alpha_{max}^c\right)$$
$$\beta_k = \left(\frac{\beta_{max}-\beta_{min}}{\beta_{max}^c-\beta_{min}^c}\right)\beta_k^c + \left(\beta_{max} - \left(\frac{\beta_{max}-\beta_{min}}{\beta_{max}^c-\beta_{min}^c}\right)\beta_{max}^c\right) \quad (6)$$

We used first-order derivatives for edge detection. For horizontal edge detection, we compute the horizontal gradient as : $G_h(m,n) = I(m,n) - I(m+1,n)$. The vertical gradient is computed as $G_v(m,n) = I(m,n) - I(m,n+1)$ for vertical edge detection. The amplitude of an edge is calculated as, $G(m,n) = |G_h(m,n)| + |G_v(m,n)|$. The mean amplitude for a block is computed as,

$$G_\mu = \frac{1}{N_B \times N_B} \sum_m \sum_n G(m,n) \quad (7)$$

When the mean amplitude for a block exceeds a predefined threshold, we declare it as an edge block. The values of m and n correspond to the pixel locations of individual blocks with reference to the original image pixel location.

The mean gray value of a block is calculated as the average of the gray values of all pixels in the image block. The mean gray values are normalized with pure white pixel gray value. Thus, the normalized mean gray values of a block is,

$$\hat{\mu_k}_I = \frac{1}{N_B \times N_B}\left(\frac{1}{I_{white}}\right)\sum_m \sum_n I(m,n) \quad (8)$$

Where, m and n are the pixel locations of the k^{th} image block, same as their locations in the original image. The normalized standard deviation of gray values for the k^{th} block is calculated as follows.

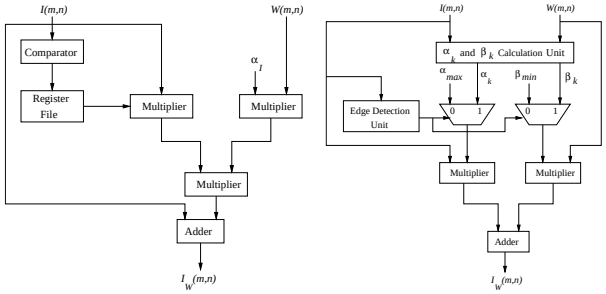
$$\hat{\sigma_k}_I = \frac{1}{N_B \times N_B}\left(\frac{2}{I_{white}}\right)\sum_m \sum_n \left|I(m,n) - \frac{I_{white}}{2}\right| \quad (9)$$

The exponential term in Eqn. 5 is approximated as a Taylor series upto the square term.

In step three of the insertion algorithm, scaling needs to be done using a linear transformation, to find the current minimum and maximum values for both α_k and β_k over all the blocks. The hardware performance is going to be severely degraded since it has to wait till all the pixels of the images are processed to derive the local statistics of all the blocks. So, we modify the above Eqn. 5 to ensure that the performance of the hardware is improved with no compromise on the quality. We find α_k and β_k using the following equations.

$$\alpha_k = \alpha_{min} + (\alpha_{max} - \alpha_{min})\frac{1}{\hat{\sigma_k}_I} exp\left(-(\hat{\mu_k}_I - \hat{\mu}_I)^2\right)$$
$$\beta_k = \beta_{min} + (\beta_{max} - \beta_{min})\hat{\sigma_k}_I\left(1 - exp\left(-(\hat{\mu_k}_I - \hat{\mu}_I)^2\right)\right) \quad (10)$$

Extensive simulations for various images show that the α_k and β_k obtained using Eqn. 6 and Eqn. 10 are comparable (maximum difference is 5% [1]).



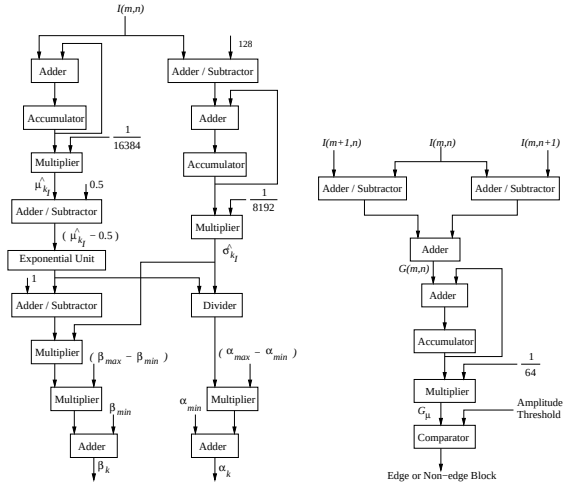
(a) For Algorithm 1 (b) For Algorithm 2

Figure 2: Datapath Architectures for the Algorithms

3 VLSI Architecture

We develop an architecture for the first algorithm shown in Fig. 2(a). A register file is used to store the constants needed to scale the image-watermark product in Eqn. 3. We store the constants $\frac{1}{903.3}$, $\frac{C_1}{6.0976}$, $\frac{C_2}{6.0976}$, $\frac{C_3}{6.0976}$, and $\frac{C_4}{6.0976}$ and the other constant α_I is assumed as a parameter. The comparator is used to determine the range in which a particular pixel gray value lies, such that an appropriate constant can be picked up from the register file. The left side multiplier calculates appropriate constant times the host image pixel gray values and the right side multiplier is used to find α_I times the watermark image pixel gray value. The results of the above two multipliers are fed to the third multiplier which effectively calculates the product of constants, α_I, the host image pixel gray value, and the watermark image pixel gray value, respectively. The product is added to the host image pixel gray values using the adder to obtain watermarked image pixel gray values. The above described process is carried out for all the pixels to obtain the watermarked image.

The architecture being proposed for the second algorithm is shown in Fig. 2(b) which present the operation at pixel level. The "α_k and β_k calculation unit" computes the α_k and β_k values for the k^{th} non-edge block using expression in Eqn. 10. The "edge detection unit" determines if a block is an edge block or non-edge block. If the G_μ exceeds an user defined threshold, then it is an edge-block. Larger the threshold more are the blocks declared as edge-blocks. The multiplexors help in selecting the scaling and embedding factors between the edge and non-edge blocks. The multiplier on the left calculates the scaling factors times the host image pixel gray value. The right side multiplier multiplies the embedding factor with the watermark image pixel gray value. The products from these two multipliers are added using an adder to find the watermarked image pixel gray value. This process is repeated for all pixels in a block, and subsequently for all the blocks in the image.



(a) α_k and β_k Calculation (b) Edge Detection

Figure 3: Individual Datapath Units for Algorithm 2

α_k and β_k calculation unit: The architectural details of the "α_k and β_k calculation unit" are shown in Fig. 3(a). The hardware implements the calculation for α_k and β_k represented as Eqn. 10 for one block at a time. The left side adder-accumulator combination finds the sum of all the image pixel gray values for a block. After the sum is multiplied with $\left(\frac{1}{N_B \times N_B} * \frac{1}{I_{white}}\right)$, we get the normalised mean gray value of k^{th} block denoted by $\hat{\mu_{kI}}$. Since we have assumed block size of 8×8, and I_{white} as 256, this evaluates to $\frac{1}{16384}$. It may be noted that I_{white} is 255, but using 256 makes hardware implementation easier, the latter being representable as a power of two. In the original algorithm $(\hat{\mu_{kI}} - \hat{\mu_I})$ is the deviation of a mean gray value of a block from the image mean gray value. We are evaluating the deviation of mean block gray value from mid-intensity of $\frac{I_{white}}{2}$ for simplicity, . Thus, $(\hat{\mu_{kI}} - \hat{\mu_I})$ is computed as $(\hat{\mu_{kI}} - 0.5)$, when normalised with I_{white}. This assumption accelerates the hardware performance to a great extent since the block-by-block watermarking can be performed without waiting for the global image statistics computed over the whole image before the watermark insertion can be performed. The expression $exp\left(-(\hat{\mu_{kI}} - \hat{\mu_I})^2\right)$ is computed using the "exponential unit".

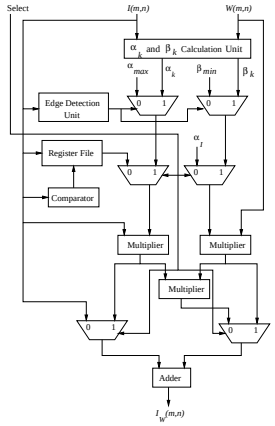
The adder/subtractor unit finds the image pixel gray value absolute deviation from $\frac{I_{white}}{2}$. The adder-accumulator following this accumulate the $\sum_m \sum_n \left| I(m,n) - \frac{I_{white}}{2} \right|$ for a block. When this sum is multiplied with $\left(\frac{1}{N_B \times N_B}\right) * \left(\frac{2}{I_{white}}\right)$, which is 8192 for our case, we get the normalised standard deviation $\hat{\sigma_{kI}}$. The right side divider divides exponential value computed before by $\hat{\sigma_{kI}}$. The quotient is then multiplied with $\alpha_{max} - \alpha_{min}$. The above product is added to α_{min} to evaluate α_k expressed in Eqn. 10. The exponential unit result is fed to a adder/subtractor on left side which finds its difference from 1. The result is then multiplied with $\hat{\sigma_{kI}}$ obtained from the computations performed before. The product obtained is then multiplied with $\beta_{max} - \beta_{min}$. This product is then added to β_{min} which in turn gives the required β_k as per Eqn. 10.

Edge detection unit: The architecture for determining if a block is an edge or non-edge block is shown in Fig. 3(b). The absolute values of the horizontal gradient $|G_h(m,n)|$ and the absolute value of vertical gradient $|G_v(m,n)|$ are calculated. The amplitude of an edge $G(m,n)$ is calculated using the first adder. Then, the adder-accumulator combination finds the sum of $G(m,n)$ for all pixels of a block. The above sum when multiplied with $\left(\frac{1}{N_B \times N_B}\right)$ $(= 64)$, we get the mean amplitude G_μ for a block. The comparator compares the G_μ values with an user defined threshold and declares the block as an edge or non-edge block.

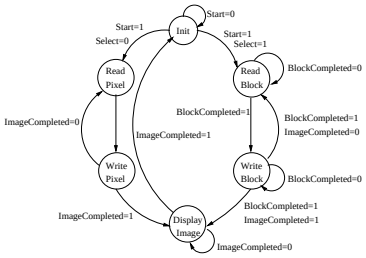
The individual datapaths for both the algorithms are stitched together using multiplexors and a combined datapath shown in Fig. 4(a) is obtained. Both the datapaths share the same multipliers, as it is evident from Fig. 4(a), the multiplexors help in selecting input for the multipliers (When Select is "0" algorithm 1 is used). The controller that drives the datapath is shown in Fig. 4(b). The controller has six states, such as Init, ReadBlock, WriteBlock, ReadPixel, WritePixel, and DisplayImage. When the Start signal is "1" the watermarking process is initiated. Depending on the Select signal one of the watermarking schemes is chosen and the corresponding datapath needs to be driven to carry out the watermarking process.

4 Prototype Chip Implementation

The implementation of the watermarking datapath and controller was carried out in the physical domain using the Cadence Virtuoso layout tool using bottom-to-top hierarchical design approach. The design involved the construction of four main units, such as the exponential unit, the edge detection unit, the α_k and β_k calculation unit, register file, and the accumulator. All of the above units have multipliers, adders, adder/subtractor, divider, comparator, and so on. These small functional units are laid out individually through modularization and later interfaced with each other to get the four above mentioned units. The datapath and the controller are constructed using the main units and the functional units. The layouts of the gates at the lowest level of hierarchy are drawn using the CMOS standard cell design approach. We designed our own standard cell library containing basic gates, such as AND, OR, NOT. The complete layout of the watermarking

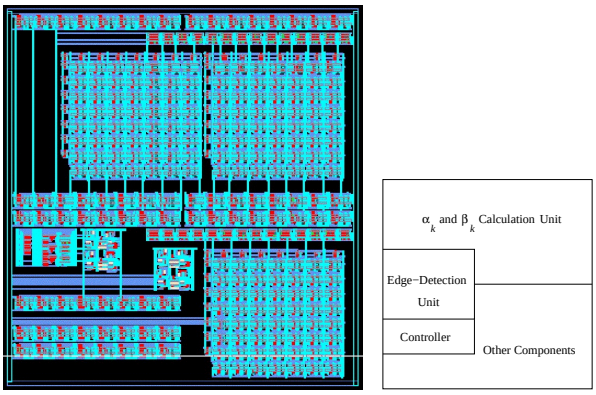


(a) Datapath for Algorithms 1 and 2



(b) Controller for Merged Datapath

Figure 4: Architecture for the Proposed Processor



(a) Chip Layout (b) Floor Plan

Figure 5: Proposed Watermarking Chip

Table 2: Overall Statistics of the Watermarking Chip

Area	$3.34 \times 2.89mm^2$
Number of gates	28469
Clock frequency	$292.27MHz$
Number of I/O pins	72
Power	$6.9286mW$

chip is given in Fig. 5(a) and the floor plan of the chip is provided in Fig. 5(b). Table 2 shows the overall design details of the chip and the corresponding pin diagram is shown in Fig. 6. The chip statistics are obtained using HSPICE for 0.35μ MOSIS SCN3M SCMOS technology.

The datapath construction involves the implementation of the proposed architecture in the previous section. The functional units are 8-bit ripple carry adders, 8-bit multipliers and 8-bit adder/subtractor. The adder/subtractor unit is obtained from the adder using XOR gates [11]. An 8-bit parallel array multiplier is obtained from full-adders and AND gates to implement multiplication operations with reduced delay [12]. The divider is implemented using the shift and subtract logic for the division [11]. The comparator was designed to compare values of two 8-bit numbers for greater-than, equal to, or less-than relations. First, a single-bit comparator was designed to compare the values of two single-bit numbers, and later, instances of this module were cascaded to compare two 8-bit numbers, starting from the most-significant bit position and proceeding towards the least-significant bit position.

The accumulator is implemented as a 14-bit register to accommodate a maximum value of 64×256. The maximum value occurs when each pixel in a 8×8 block assumes the value of pure white pixel gray value. The register file is an addressable array of 8-bit registers (words) [12]. Based on the address specified and a Read/Write select line, at any time, a value can be either written to or read from the register file. Here, we used a 5-word register file to store the five different constants, such as $\frac{1}{903.3}$, $\frac{C_1}{6.0976}$, $\frac{C_2}{6.0976}$, $\frac{C_3}{6.0976}$, and $\frac{C_4}{6.0976}$, in Eqn. 3. Multiplexors are used at appropriate places in the design to select one of the incoming lines. Each of such multiplexor is implemented using a combination of transmission gates. Three asynchronously resettable registers are designed to encode the five states of the controller depicted in Fig. 4(b). The three registers could be reset by the user to return the controller to its intial state at any time and from there, the watermarking function could be started afresh.

5 Results and Conclusions

The functional units are simulated individually before they are integrated to develop the whole chip. The functional verification of the whole chip is done by performing watermarking on various test images. The test images are borrowed

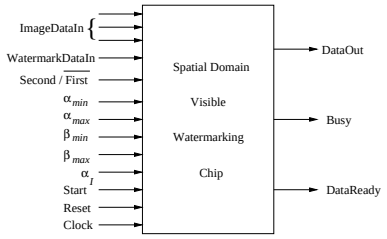


Figure 6: Pin diagram for the Proposed Chip

(a) Original Lena (b) Watermark Image

(c) Using Algorithm 1 (d) Using Algorithm 2

Figure 7: Original and Watermarked Images

from [3, 1] and of dimension 256×256. Fig. 7 shows one test image, a watermark image used, and watermarked images. The watermarked image is also shown in Fig. 7. For first algorithm, the values of α_{min}, α_{max}, β_{min}, and β_{max} are assumed as $0.95, 0.98, 0.02$, and 0.07, respectively, and for second algorithm α_I is 0.03. The visual inspection of the watermarked images proves that watermarking is able to preserve the quality of the image while explicitly proving the ownership. Of the various quantitative measures available to quantify the quality of the watermarked images, we used signal-to-noise ratio (SNR) as suggested by [5, 3, 1]. We calculated the SNR using the original and the watermarked image using a software simulator. Simulation results show that the SNR for various watermarked images is in the range of $20dB$ to $25dB$.

In this paper, we have presented a watermarking chip that can be integrated within a digital camera framework for watermarking images. The chip has two different types of watermarking capabilities, both in spatial domain. Out of the two watermarking schemes implemented, the first one does pixel-by-pixel processing and the second one is a block-by-block processing algorithm. Additional work needs to be done to develop block-by-block operation for the first algorithm so that high performance hardware can be designed. However, both the algorithms are comparable from the SNR point of view.

References

[1] S. P. Mohanty, "Watermarking of Digital Images," M.S. thesis, Indian Institute of Science, Bangalore, India, 1999.

[2] N. Memon and P. W. Wong, "Protecting Digital Media Content," *Comm. of the ACM*, vol. 41, no. 7, pp. 34-43, Jul 1998.

[3] S. P. Mohanty, K. R. Ramakrishnan, and M. S. Kankanhalli, "A Dual Watermarking Technique for Images," in *Proc. of the 7th ACM Intl. MM Conf. (Vol. 2)*, 1999, pp. 49-51.

[4] L. D. Strycker, et. al., "Implementation of a Real-Time Digital Watermarking Process for Broadcast Monitoring on Trimedia VLIW Processor," *IEE Proc. on Vision, Image and Signal Processing*, vol. 147, no. 4, pp. 371-376, Aug 2000.

[5] N. J. Mathai, D. Kundur, and A. Sheikholeslami, "Hardware Implementation Perspectives of Digital Video Watermarking Algortithms," *IEEE Trans. on Signal Processing*, 2003.

[6] T. H. Tsai and C. Y Lu, "A System Level Design for Embedded Watermark Technique using DSC System," in *Proc. of the IEEE Intl. Workshop on Intelligent Signal Processing and Communication System*, 2001.

[7] A. Garimella, et. al., "VLSI Impementation of Online Digital Watermarking Techniques With Difference Encoding for the 8-bit Gray Scale Images," in *Proc. of the Intl. Conf. on VLSI Design*, 2003, pp. 792-796.

[8] S. P. Mohanty, N. Ranganathan, and R. K. Namballa, "VLSI Implementation of Invisible Digital Watermarking Algorithms Towards the Developement of a Secure JPEG Encoder," in *Proc. of the IEEE Workshop on Signal Processing Systems*, 2003, pp. 183-188.

[9] G. W. Braudaway, K. A. Magerlein, and F. Mintzer, "Protecting Publicly Available Images with a Visible Image Watermark," in *Proc. of the SPIE Conf. on Optical Security and Counterfiet Deterrence Technique*, 1996, pp. 126-132.

[10] J. Meng and S. F. Chang, "Embedding Visible Video Watermarks in the Compressed Domain," in *Proc. of the Intl. Conf. on Image Processing (Vol. 1)*, 1998, pp. 474-477.

[11] V. P. Nelson, et. al., *Digial Logic Analysis and Design*, Prentice Hall, Upper Saddle River, New Jersey, USA, 1995.

[12] N. H. E. Weste and K. Eshraghian, *Principles of CMOS VLSI Design : A Systems Perspective*, Addison Wesley, Boston, MA, USA, 1999.