Optimization and Hardware Implementation of Image and Video Watermarking for Low-Cost Applications

Konstantinos Pexaras, *Student Member, IEEE*, Irene G. Karybali, *Member, IEEE*, and Emmanouil Kalligeros, *Member, IEEE*

***Abstract*— The prevalence of wireless networks has made the long-term need for communications security more imperative. In various wireless applications, images and/or video constitute critical data for transmission. For their copyright protection and authentication, watermarking can be used. In many cases, the cost of wireless nodes must be kept low, which means that their processing and/or power capabilities are very limited. In such cases, low-cost hardware implementations of digital image/video watermarking techniques are necessary. However, to end up with such implementations, proper selection of watermarking techniques is not enough. For this reason, in this paper, we introduce computation optimizations of the implemented algorithm to keep the integer part of arithmetic operations at optimal size, and, hence, arithmetic units as small as possible. In addition, further analysis is performed to reduce quantization error. Three different hardware-architecture variants, two for image watermarking and one for video (pipelined), are proposed, which reutilize the already small arithmetic units in different computation steps, to further reduce implementation cost. The proposed designs compare favorably to already existing implementations in terms of area, power, and performance. Moreover, the watermarked images'/frames' errors, compared to their floating point counterparts, are very small, while robustness to various attacks is high.**

***Index Terms*— Watermarking, images, video, hardware implementation, digital systems, algorithmic optimizations.**

I. Introduction

SECURITY constitutes a timeless requirement in any type of communication. In the previous decades, the rapid spread of the Internet made this requirement more imperative, since a large amount of personal and sensitive data began to be exchanged across the world. The advent and prevalence of wireless networks has emphasized this long-term need for communications security. Apart from the sensitive nature of the transmitted data (e.g., traffic and surveillance cameras), wireless devices are vulnerable to physical attacks, while the wireless technologies used for their communications are susceptible to interference and interception [1], [2]. Since, in many applications, the critical data are images or video, image and video security is a major concern for such networks.

Two of the required security features are copyright protection and authentication. A malicious adversary may duplicate a node's data for profit [3], while the source of critical or security data must be verifiable. Digital watermarking has been successfully used for copyright protection and authentication of images, video or other kinds of data. Concerning digital images and video (i.e., a sequence of images), many watermarking techniques have been proposed in the literature (e.g., see [4], [5]). In many cases though, especially at network edge [6], the cost of wireless nodes has to be kept low, which means that their resources (mainly, processing power and memory) are very limited [1]. Therefore, a new low-cost environment emerges, in which the implementation, in software, of many of the proposed digital image/video watermarking algorithms is not possible.

In such cases, a hardware implementation is preferable. However, for wireless-network applications, a hardware-implemented watermarking scheme should fulfill two contradictory requirements: high robustness to attacks, due to the critical nature of the data, and small implementation cost. The latter is necessary because of the stringent power consumption constraints (i.e., battery life) of many wireless nodes, especially at the edge of the network [6].

To address both of these demands at the same time, in this paper we develop low-cost, robust-watermark embedders for both still images and video. First, a watermarking technique is properly selected, based on performance and cost criteria. In terms of cost, what matters is that the necessary arithmetic operations require, mainly, "cheap" arithmetic units. Moreover, since implementation cost comes primarily from these units, they should be as small as possible. To this end, we introduce computation modifications that maintain the integer part of operations at optimal size. On top of that, further analysis is performed so as to reduce quantization errors, i.e., errors that occur in the numbers' fractional part during arithmetic operations. Finally, hardware architectures are proposed, both for image watermarking (a serial, very low-cost one, and a more costly yet efficient, parallel one), and video watermarking (a pipelined one) that reutilize arithmetic units in different computation steps, to keep the overall cost as small as possible.

Manuscript received July 16, 2018; revised February 8, 2019; accepted March 14, 2019. Date of publication April 22, 2019; date of current version May 15, 2019. This paper was recommended by Associate Editor E. Blokhina. *(Corresponding author: Emmanouil Kalligeros.)*

The authors are with the Department of Information and Communication Systems Engineering, University of the Aegean, 83200 Samos, Greece (e-mail: icsd09119@icsd.aegean.gr; karybali@aegean.gr; kalliger@aegean.gr).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSI.2019.2907191

A. Related Work in the Literature

Various hardware architectures of image watermarking techniques have been presented in the literature, which, in general, perform watermark embedding in the spatial or frequency domain (Discrete Cosine Transform – DCT, Wavelet Transform, etc.). For their implementation, either Field Programmable Gate Arrays (FPGAs) or Application Specific Integrated Circuits (ASICs) have been employed. As an early work in the area, [7] can be mentioned, in which a chip capable of inserting invisible-robust watermarks in the DCT domain was designed. An FPGA implementation of an invisible-robust watermarking technique in the wavelet domain has been presented in [8]. The authors of [9] and [10] present FPGA- and ASIC-based architectures for robust and fragile watermark insertion in the spatial domain, while in [11] the first low-power watermarking chip in the literature was introduced. Hardware-based watermarking solutions for CMOS image sensors have been described in [12]–[14]. A spread spectrum spatial domain watermarking scheme that uses binary watermarks has been proposed in [15]. Lately, some implementations of reversible watermarking, targeting mainly medical and military images, have been presented, like [16] and [17], while even asynchronous implementations [18] have appeared in the literature.

Similarly, for video watermarking, a number of architectures realized in hardware can be found. The authors of [19] proposed the JAWS watermarking system, using for its implementation the Philips Trimedia very long instruction word (VLIW) processor, while in [20], [21] an ASIC architecture for the JAWS scheme was developed. The Millennium watermarking system for DVD copyright protection was presented in [22]. The authors of [23] developed a real-time video watermarking system using DSP and VLIW processors. An FPGA implementation of a real-time video watermark embedder based on the Haar wavelet transform was presented in [24]. In [25], Mohanty proposed the concept of a secure digital camera with a built-in watermarking and encryption facility, while in [26] a watermarking algorithm and the corresponding VLSI architectures that insert, in real-time, a broadcaster's logo (visible watermark) in video streams, were developed. The authors of [27] presented a hardware implementation of a digital watermarking system that inserts invisible, semi-fragile watermarks into video streams during compression. Watermark embedding in [27] is performed in the DCT domain. An invisible robust video-watermarking approach that also relies on the DCT has been presented in [28]. The architecture of [28] uses integer arithmetic units and is readily adaptable to the H.264 standard.

In this paper, the proposed watermark-embedding architecture targets low-cost, wireless applications. Because of this, contrary to the already presented designs, it combines/satisfies all of the following features/requirements: a) low cost (this is why algorithmic modifications are proposed), b) high robustness to compression and transmission losses. Even High Efficiency Video Coding (HEVC) has been considered for compression, although, due to its increased requirements, it is not applicable to low-cost devices, at least for the moment. c) Watermark embedding in the spatial domain, in order to be directly applicable to very low-cost devices without compression. d) Independence from the (potentially) utilized compression scheme, so that the proposed architecture can be employed in a wide range of cases, from the simple JPEG image-compression case, to the sophisticated HEVC video-compression, while keeping its low-cost characteristics. None of the aforementioned literature techniques offer all these features at the same time, and this constitutes the main contribution of the proposed approach.

The rest of the paper is organized as follows: in Section II, a brief description of the selected watermarking algorithm is given, along with the reasons that led us to its selection. The algorithmic optimizations for low-cost hardware implementation and quantization-error reduction are also provided. Sections III and IV present respectively the low-cost hardware implementations for still-image and video watermarking, while in Section V experimental results and comparisons with other implementations in the literature are provided, as well as evaluations of quantization-error optimizations and robustness to attacks. Finally, Section VI concludes the paper.

II. Watermark Embedding

A hardware-implemented watermarking technique for critical, low-cost applications should be robust, due to the sensitive nature of the data, and have a small footprint. So, we opt for spatial domain techniques based on perceptual masking. In this way, we avoid the cost of the direct and inverse transformations involved in frequency domain embedding, and, using an appropriate perceptual mask, we ensure the watermark robustness. By employing perceptual masks, which take into account the properties of the human visual system (HVS), the watermark energy can be properly adapted, achieving robustness and invisibility at the same time.

In this work, we adopt the NVF (Noise Visibility Function)-based mask presented in [29], denoted hereafter as M_{NVF}, for watermark embedding. This mask has been derived so as to enhance the robustness of the embedded watermark against the denoising attack, which causes high-frequency information loss. The watermark is robust to this loss, since it is amplified on the image/frame edges or textured areas corresponding to high frequencies. Note that the elimination of high-frequency information (which is visually less perceivable) is also the result of image or video compression, irrespective of the used compression standard [30]. So, it is expected that the adopted watermarking technique will be robust to compression too, which is actually true. It has been shown in [4] that M_{NVF}-based watermarking resists JPEG compression. We will show here that it also resists compression with modern standards, such as H.264/AVC [31] or HEVC [32]. Note though that, from the perspective of low cost applications, compression standards such as HEVC would not be the first choice, due to their high complexity and power consumption.

Additionally, some of the effects of compression, such as image blocking and blurring, are common artifacts introduced by wireless channel losses. Moreover, by employing a method robust to compression and compression-standard independent, we ensure that the proposed embedders can be used in image or video capturing systems with compression, preceding the compression module. For a thorough analysis of

M_{NVF}-based watermarking robustness, refer to [29], [4] and Section V-C.

According to [29], if x is the cover image and w the watermark, the watermarked image will be

$$y = x + \alpha M_{NVF} \odot w, \tag{1}$$

where $\odot$ stands for point-wise multiplication. The watermark is a Gaussian distributed pseudorandom pattern (it can be generated using a random key) of equal size and uncorrelated with the cover image. It has zero mean, while its strength (i.e., its standard deviation) is denoted by α, which means that w itself has variance equal to 1.

The NVF identifies textured and edge regions, where the watermark can be amplified. For the determination of the optimal NVF, the watermark is considered as noise, and the classical MAP (Maximum a Posteriori Probability) image denoising approach is used [29]. The perceptual mask derived, assuming a non-stationary Gaussian model, is defined as

$$M_{NVF}(i,j) = 1 - \frac{1}{1+\sigma_x^2(i,j)}, \tag{2}$$

where $\sigma_x^2(i,j)$ is the local variance of the cover image in a neighborhood centered on pixel (i,j), $1 \le i \le M$, $1 \le j \le N$ and $M \times N$ are the image dimensions.

A. Optimization for Minimum Integer Bit-Width

When transforming a high-level algorithmic description to its fixed-point implementation, two problems may arise. The first one is overflows, which can appear if the integer bit-width is too small. In our implementation, this problem is tackled via the computation modifications that will be presented in this subsection (they were first introduced in [33]). The second problem is the precision loss or quantization error caused by the finite fraction bit-width. Optimizations for keeping this error low are proposed in Section II-B.

Thus, in this subsection, we aim to maintain the integer portion of operations at optimal size. Since, after watermarking, the values of gray-scale images can be $\ge$256 or $<$0, the optimal size of their (signed) integer portion is 10 bits. Additionally, 10 bits are used for their fractional part to have sufficient precision. Specifically, with a 10-bit fractional part, we can represent, *before arithmetic operations*, all values with a precision of 10^{-3} ($2^{-10} = 1/1024$). We consider this 3-decimal-digit initial accuracy as a minimum requirement, since arithmetic operations will subsequently induce quantization errors that will reduce the results' accuracy. Below, we determine the peak value that each involved quantity can reach and we propose proper modifications, so as the involved computations fit in the 10.10 (20-bit wide) signed fixed-point representation.

The mask is computed using the unbiased sample variance:

$$\sigma_x^2(i,j) = \frac{1}{mn-1} \sum_{k=-\lfloor \frac{m}{2} \rfloor}^{\lfloor \frac{m}{2} \rfloor} \sum_{l=-\lfloor \frac{n}{2} \rfloor}^{\lfloor \frac{n}{2} \rfloor} [x(i+k, j+l) - \mu(i,j)]^2, \tag{3}$$

where

$$\mu(i,j) = \frac{1}{mn} \sum_{k=-\lfloor \frac{m}{2} \rfloor}^{\lfloor \frac{m}{2} \rfloor} \sum_{l=-\lfloor \frac{n}{2} \rfloor}^{\lfloor \frac{n}{2} \rfloor} x(i+k, j+l) \tag{4}$$

is the local mean value computed in a $m \times n$ neighborhood and $\lfloor \cdot \rfloor$ is round to the integer towards minus infinity.

The sums' result in (4) is upper bounded by $255mn$ (total white neighborhood), which needs $\lceil \log_2(255mn) \rceil$ bits for its representation ($\lceil \cdot \rceil$ is round to the integer towards plus infinity). We rearrange the operations in (4) as shown below:

$$\mu(i,j) = \sum_{k=-\lfloor \frac{m}{2} \rfloor}^{\lfloor \frac{m}{2} \rfloor} \sum_{l=-\lfloor \frac{n}{2} \rfloor}^{\lfloor \frac{n}{2} \rfloor} \frac{1}{mn} \cdot x(i+k, j+l). \tag{5}$$

We first multiply the pixels' values with $\frac{1}{mn}$ and then compute their sum, which now takes values in the interval [0, 255]. So, only 8 bits are needed for the representation of its integer part.

Continuing with (3), the difference of the image and the local mean values is in the range $(-255, 255)$ (the endpoint values can never occur), and the sums' result, which takes values in $[0, 255^2 mn)$, needs $\lceil \log_2(255^2 mn) \rceil$ bits for the representation of its integer part. Instead of $\sigma_x^2(i,j)$, we compute $\frac{\sigma_x^2(i,j)}{256}$ which is equal to

$$\sum_{k=-\lfloor \frac{m}{2} \rfloor}^{\lfloor \frac{m}{2} \rfloor} \sum_{l=-\lfloor \frac{n}{2} \rfloor}^{\lfloor \frac{n}{2} \rfloor} \frac{[x(i+k, j+l) - \mu(i,j)]}{mn-1} \cdot \frac{[x(i+k, j+l) - \mu(i,j)]}{256}. \tag{6}$$

We divide $[x(i+k, j+l) - \mu(i,j)]$ with $(mn-1)$ and with 256, we multiply the results and compute their sum. Since we use a 3×3 neighborhood in our implementation, $\frac{1}{mn-1}$ translates to 3 right shifts. Similarly, division by 256 corresponds to 8 right shifts. The range of values of $\frac{\sigma_x^2(i,j)}{256}$ is $[0, \frac{255^2}{(mn-1)256} \cdot mn)$, and for a 3×3 neighborhood, 10 bits are needed for the signed representation of its integer part.

Finally, the mask is computed by the following equation:

$$M_{NVF}(i,j) = 1 - \frac{\frac{1}{256}}{\frac{1}{256} + \frac{\sigma_x^2(i,j)}{256}}. \tag{7}$$

Concerning the desired image quality, it can be derived by adjusting the watermark strength, i.e. α, in eq. (1). The peak signal-to-noise ratio ($PSNR$) is used for this purpose:

$$PSNR = 10 \log_{10} \frac{\max(x)^2}{\|y - x\|^2} \tag{8}$$

where $\max(x) = 255$ and $\| \cdot \|$ is the Euclidean norm. Let $u \equiv M_{NVF} \odot w$ be the masked watermark. Then, from (8):

$$\alpha = \frac{A}{\sqrt{\|u\|^2}}, \quad A = \frac{255}{\sqrt{10^{\frac{PSNR}{10}}}}. \tag{9}$$

A is smaller than 9 when the $PSNR$ ranges from 30 to 45 dB. Higher values of $PSNR$ correspond to better image quality. Our implementation gives the option to select the quality level, by using a lookup table with different values of A.

TABLE I
WATERMARK EMBEDDING STEPS AND OPTIMIZATIONS

Computation Step	Original Equation [29]	Modified Equation
$\mu(i,j)$	Eq. (4)	Eq. (5) Eq. (13)
$\sigma_x^2(i,j)$	Eq. (3)	Eq. (6) Eq. (21)
$M_{NVF}(i,j)$	Eq. (2)	Eq. (7)
u	$u = M_{NVF} \odot w$	As is
$\|u\|^2$	Eq. (10)	Eq. (11)
α	Eq. (9)	$\alpha = \frac{(A \times 16)}{\sqrt{256 \cdot \|u\|^2}}$
y	Eq. (1)	As is

The computation of α involves the computation of

$$\|u\|^2 = \frac{1}{MN} \sum_{i=1}^{M} \sum_{j=1}^{N} u^2(i,j), \tag{10}$$

where $u^2(i,j) = M_{NVF}^2(i,j) \cdot w^2(i,j)$ is upper bounded by $w^2(i,j)$, since M_{NVF} takes values from 0 to 1. Since w is a Gaussian distributed pseudorandom pattern with zero mean and variance equal to 1, $\frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N} w^2(i,j)$ equals the variance of w, i.e. equals 1 [34]. This means that the sums' result in (10), i.e. $\|u\|^2 MN$, is upper bounded by MN, which needs $\lceil \log_2 MN \rceil$ bits for its representation (we remind that $M \times N$ are the image dimensions).

To reduce the number of bits required for the above representation, we modified the computation of $\|u\|^2$ as follows:

$$\|u\|^2 = \frac{1}{256} \sum_{q=1}^{J} \left[\left[\left(\sum_{p=P_{q-1}+1}^{P_q} \mathbf{u}^2(p) \right) / \left(\frac{M}{16}\right) \right] / \left(\frac{N}{16}\right) \right], \tag{11}$$

where vector $\mathbf{u}$ contains the elements of the masked watermark taken column-wise, $P_0 = 0$ and $P_q : \sum_{p=P_{q-1}+1}^{P_q} \mathbf{u}^2(p) \leq 462$. First, the elements of $\mathbf{u}$ are added until their sum reaches value 462. We remind that u is upper bounded by w, which takes values in $(\mu - 6\sigma, \mu + 6\sigma)$, i.e. $(-6, 6)$ in our case, with probability that approaches 1. In order to ensure that $\sum_{p=P_{q-1}+1}^{P_q} \mathbf{u}^2(p)$ will not exceed 511, we set a threshold equal to 462 $(511 - 7^2)$. This threshold was selected to cover the extreme case that w takes values out of $(-6, 6)$. Then, this sum is divided first with $M/16$ and then with $N/16$. Note that by using the 10.10 signed fixed-point representation, the image dimensions can be up to $2^{13} \times 2^{13}$. The above process is repeated for all elements of u. The total sum, i.e. $256 \cdot \|u\|^2$, is upper bounded by 256 and needs 10 bits for its signed, integer part representation. Its square root is then computed and $\alpha = (A \times 16)/\sqrt{256 \cdot \|u\|^2}$. Since $\|u\|^2$ is upper bounded by 1, α in (9) can take a bit higher values than A.

Finally, the watermarked image y in (1) takes values within the desired representation range, since $\alpha M_{NVF} \odot w$ is upper bounded by $\alpha \cdot w$, which takes values in $(-6\alpha, 6\alpha)$. The watermark embedding steps and the corresponding optimizations are summarized in Table I ($\|u\|^2$ and α are in the same step).

The described optimizations reduce the size of the adders required for the implementation of equations (4), (3) and (10) from 30 bits (for eq. (10) we consider 1280×720 image/frame size) to 20 bits (eq. (5), (6) and (11) – (5) and (6) share the same adders, as will be explained in Section III), and, most importantly, the size of the divider that will be used in the proposed designs from 40:27 bits (i.e., 40-bit dividend and 27-bit divisor) to 30:20 bits. The latter is vital, especially for low-cost designs, since the divider is a complicated, array-like structure that occupies a significant part of the implementation area.

B. Optimization for Low Quantization Error

A way to estimate the errors induced by the finite word-length is via simulations, which may be time consuming, or by precision analysis, which can be made dynamically [35] or statically. Affine arithmetic [36], which is an improved static analysis technique compared to the traditional interval arithmetic [37], has been used for precision analysis [38], [39]. In [38], statistical interpretations of affine arithmetics are given. In [40], a quantitative evaluation of various kinds of static analysis techniques is presented. The effect of the quantization error on the computations' precision is studied in [41] and a model to enhance and predict this precision is presented.

Here, we study the quantization errors of the first steps of our algorithm, since these affect all the computations that follow. First, we focus on the error occurring from the computation of the local mean value in eq. (5), which takes part in the computation of $\frac{\sigma_x^2(i,j)}{256}$ in eq. (6). For both equations, we propose modifications to keep quantization errors low.

Since 10.10 signed fixed-point representation is used in our implementation, the fractional part of the handled numbers consists of 10 bits. Let f denote the bit-width of the fractional part. The quantization error is bounded by $2^{-(f+1)}$ in case of rounding, or 2^{-f} in case of truncation. Rounding gives lower errors than truncation, but requires additional hardware. Truncation needs one extra bit to achieve the same error bound as round to the nearest [35]. In our hardware implementation, we use truncation to avoid the extra hardware. Therefore, the fixed-point representation v_f of a real number v is

$$v_f = v + 2^{-f}\varepsilon, \quad with\ \varepsilon \in [-1, 0]. \tag{12}$$

Since the sign of the error $v - v_f$ is the same as the sign of v, which means that there is no possibility of errors' cancellation, we have to keep the truncation errors as low as possible.

In the local mean value computation, truncation errors arise from the fixed-point representations of $\frac{1}{mn}$ and its product with $x(i+k, j+l)$. Our study showed that by shifting $\frac{1}{mn}$ appropriately, we can minimize the produced errors. So, we modify eq. (5), as shown below:

$$\mu(i,j) = \sum_{k=-\lfloor \frac{m}{2} \rfloor}^{\lfloor \frac{m}{2} \rfloor} \sum_{l=-\lfloor \frac{n}{2} \rfloor}^{\lfloor \frac{n}{2} \rfloor} \frac{2^\delta}{mn} \cdot \frac{x(i+k, j+l)}{2^\delta}, \tag{13}$$

where $\delta = 0, 1, 2, \ldots, \delta_{max}$. The value of δ_{max} is determined by the fixed-point representation used. Note that since each image value $x(i+k, j+l) \in [0, 255]$, there is not truncation error in the 10.10 fixed-point representation of $\frac{x(i+k,j+l)}{2^\delta}$. The fixed-point representations of $\frac{2^\delta}{mn}$ and its product

with $\frac{x(i+k,j+l)}{2^\delta}$ are

$$\left(\frac{2^\delta}{mn}\right)_f = \left(\frac{2^\delta}{mn}\right) + 2^{-f}\varepsilon \tag{14}$$

and

$$\left[\left(\frac{2^\delta}{mn}\right)_f \cdot \frac{x(i+k,j+l)}{2^\delta}\right]_f = \frac{2^\delta}{mn}\cdot\frac{x(i+k,j+l)}{2^\delta} + E, \tag{15}$$

respectively, where

$$E = \frac{x(i+k,j+l)}{2^\delta}\cdot 2^{-f}\varepsilon + 2^{-f}\varepsilon. \tag{16}$$

So, the fixed-point representation of the local mean value, as this is computed in (13), is

$$\mu_f(i,j) = \mu(i,j) + E_\mu \tag{17}$$

where

$$E_\mu = \sum_{k=-\lfloor\frac{m}{2}\rfloor}^{\lfloor\frac{m}{2}\rfloor}\sum_{l=-\lfloor\frac{n}{2}\rfloor}^{\lfloor\frac{n}{2}\rfloor}\left[\frac{x(i+k,j+l)}{2^\delta}\cdot 2^{-f}\varepsilon + 2^{-f}\varepsilon\right]. \tag{18}$$

The sums in the μ_f computation do not introduce errors, since the result has the same fraction bit-width as each of the added terms [38]. However, error E is accumulated in eq. (18) and it is necessary to keep it as low as possible.

It is obvious from the above equations that, for finite f, for each image value, the errors E and E_μ decrease as δ increases, which has been also verified experimentally.

The upper bound of E, denoted as $UB(E)$, is computed by substituting $x(i+k,j+l)$ and ε with their maximum absolute values, which are 255 and 1, respectively, and is given by:

$$UB(E) = \left(\frac{255}{2^\delta} + 1\right)\cdot 2^{-f}. \tag{19}$$

The error computed by equation (15) is

$$E_c = \frac{2^\delta}{mn}\cdot\frac{x(i+k,j+l)}{2^\delta} - \left[\left(\frac{2^\delta}{mn}\right)_f \cdot \frac{x(i+k,j+l)}{2^\delta}\right]_f, \tag{20}$$

and its sign is positive (corresponds to $-E$ of equation (15)).

We have computed E_c, for each value of $x(i+k,j+l) \in [0,255]$ and for each value of δ allowed by the utilized fixed-point representation. In our implementation, where f equals 10, δ_{max} is 10. The 10.10 signed fixed-point representation allows the representation of $\frac{2^{\delta_{max}}}{mn}$, and $\frac{x(i+k,j+l)}{2^{\delta_{max}}}$ corresponds to 10 right shifts of $x(i+k,j+l)$. In all cases, E_c takes lower values than the upper bound $UB(E)$, as shown in Figures 1 and 2. In Fig. 1, E_c for image value 255 versus $UB(E)$ is depicted, for each value of δ. In Fig. 2, E_c is shown for all the image values from 0 to 255. For every image value, E_c is lower than $UB(E)$ and takes the same values for $\delta \geq 8$.

Note that $UB(E) \in (0.00122, 0.25]$ and $E_c \in [0, 0.19368)$. As it has already been mentioned, E is accumulated in eq. (18). For neighborhood size $m \times n$, the values of E_μ can be mn times higher, i.e. $E_\mu \in [0, 1.74312)$ for the 3×3 neighborhood used here. Similarly, $UB(E_\mu) \in (0.01098, 2.25]$.

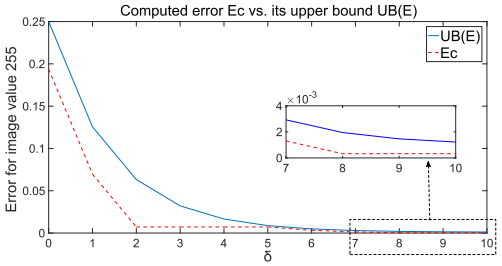


Fig. 1. Comparison of E_c for image value 255 with the upper bound of E.

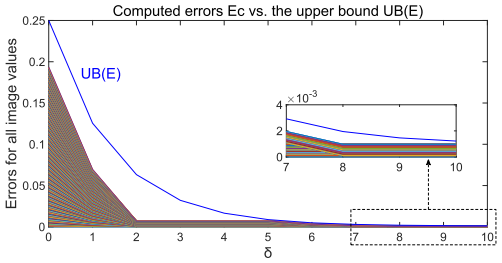


Fig. 2. Comparison of E_c for all image values with the upper bound of E.

Based on the above results, in our implementation we use $\delta = 8$ to avoid ending up with very small numbers from the computation of $\frac{x(i+k,j+l)}{2^\delta}$. Then, for $\delta = 8$, $E_\mu \in [0, 0.00879)$ and $UB(E_\mu) \in (0.01098, 0.01754]$.

Continuing with eq. (6), $[x(i+k,j+l) - \mu(i,j)]$ is divided by $(mn-1)$, which equals 2^3 in our implementation, and with 2^8. The computation of $\left[\frac{\sigma_x^2(i,j)}{256}\right]_f$ accommodates errors such as E_μ, truncation errors from the fixed point representations of the shifted terms of the product [42] and error due to the multiplication. This results in a non-linear error function, which thereafter participates in the computations that follow. The detailed study of such errors requires non-linear optimization, which is out of the scope of this paper.

The optimization proposed here is derived by simulations and aims to keep the error in the $\left[\frac{\sigma_x^2(i,j)}{256}\right]_f$ computation low, by balancing the size of the product terms. This is achieved by shifting $[x(i+k,j+l) - \mu(i,j)]$ properly, as shown below:

$$\sum_{k=-\lfloor\frac{m}{2}\rfloor}^{\lfloor\frac{m}{2}\rfloor}\sum_{l=-\lfloor\frac{n}{2}\rfloor}^{\lfloor\frac{n}{2}\rfloor}\frac{[x(i+k,j+l)-\mu(i,j)]}{(mn-1)2^d}\cdot\frac{[x(i+k,j+l)-\mu(i,j)]}{2^{8-d}}. \tag{21}$$

The lowest errors are obtained for $d = 2$.

We demonstrate the effect of the above modifications by applying them to the image shown in Fig. 3, which is watermarked for PSNR values 30, 35, 40 and 45 dB. The mean absolute error in the computation of its local mean values, as compared to those of Matlab with floating-point double-precision representation, is 0.70946 for $\delta = 0$ and 0.00491 for $\delta = 8$.

Then, for $\delta = 8$, $\left[\frac{\sigma_x^2(i,j)}{256}\right]_f$ is computed, for $d = 0$ and $d = 2$, and is compared to its floating-point double-precision

Fig. 3. Test image.

TABLE II
MEAN SQUARED ERRORS OF THE WATERMARKED IMAGE COMPUTATIONS

PSNR	30dB	35dB	40dB	45dB
MSE_1	1.26136	0.39882	0.12613	0.03988
MSE_2	0.40601	0.12839	0.04059	0.01283

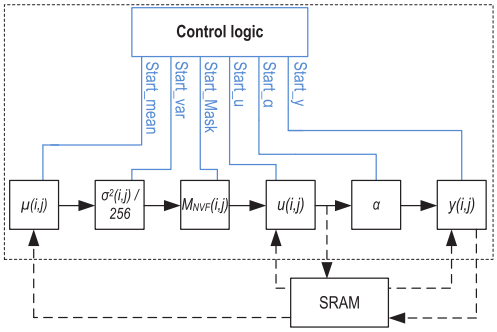


Fig. 4. Architecture of the implemented image-watermarking module.

representation. The mean absolute errors are approximately equal to 0.00364 and 0.00357, respectively.

Finally, the mean squared errors in the whole computation of the watermarked image, as compared to the corresponding ones generated with floating-point double-precision representation, denoted as MSE_1 for $\delta = 0$ and $d = 0$ and as MSE_2 for $\delta = 8$ and $d = 2$, are shown in Table II.

Concluding, for the examined image, the error for the local mean values computation is reduced by more than two orders of magnitude, a small error reduction is achieved in the computation of $\left[\frac{\sigma_x^2(i,j)}{256}\right]_f$ and MSE is reduced by about 67%. The most important is that these improvements [eq. (13) and eq. (21)] are achieved without any additional hardware as compared to equations (5) and (6), since they can be applied with just value shifts.

III. IMAGE WATERMARK EMBEDDER DESIGN

The architectural diagram of the implemented watermark-embedding module for still images [33] is shown in Fig. 4. The computation blocks are drawn as squares and correspond to the respective steps of the embedding algorithm of [29], as optimized in Section II (Table I). To keep area and power consumption low, in the image watermarking case we decided to process only one pixel at a time. In this way, only one of the blocks of Fig. 4 is used at every instant, which allows us to share arithmetic units among the blocks.

In our implementation, we assumed that the captured images are stored as raw, gray-scale data in a RAM, from which

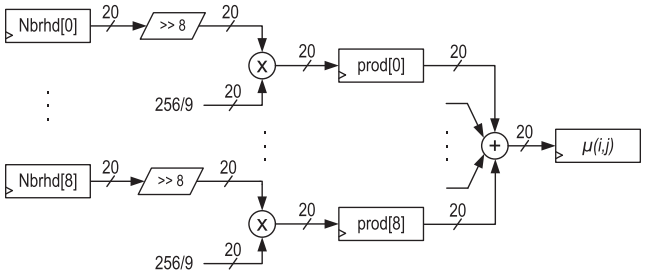


Fig. 5. μ-block datapath - parallel implementation ($>>$ denotes right shift).

they are retrieved. Specifically, SRAM was considered due to its simplicity in interfacing. We require an SRAM read or write cycle to fit in one system clock cycle, a constraint that can be easily met, since the clock frequencies of our designs are not extremely high. Of course, all SRAM transactions are registered. We note that, to keep Fig. 4 simple, we do not show the implemented SRAM interface but only the necessary SRAM accesses. This is why the corresponding lines are dashed. We should also mention that the u-block reads the watermark w from the SRAM (we consider it stored there) and writes back the $u(i, j)$ values. The latter is done because all $u(i, j)$ values are needed first for the calculation of the watermark strength α, and then for computing the watermarked image y. In all our designs (including the video-watermarking one), the required SRAM capacity is that for storing the initial image (x), w and u. The final, watermarked image y is stored over u in the case of image watermarking, or is shifted out of the embedder in the video case. Of course, if a watermark generation module is used, the corresponding SRAM space is not needed. In the following, we provide a description of the datapath of every calculation block. Note that we postpone the description of the M_{NVF}- and y-blocks for Section IV. The reason is that their design, which is relatively simple in the image-watermarking case, requires some special treatment for the pipelined video-watermarking embedder, as will be explained later.

A. Implementation of the μ-block

In Fig. 5, the parallel implementation of the μ-block datapath is shown. Actually, we have developed two different versions of the μ- and the $\sigma^2/256$-blocks: a serial, targeting very low cost, and a parallel for improved performance. In the parallel version of the μ-block, nine multipliers are employed for multiplying, in a single clock cycle, every (right-shifted by 8) pixel of a neighborhood (Nbrhd[0]-Nbrhd[8]) with 256/9 (eq. (13), $\delta = 8$). The resulting products are then added by a tree of eight adders. On the contrary, in the serial version, only one multiplier and one adder are used, but the mean value's computation is performed in nine successive clock cycles. We note that after being read from the SRAM, every pixel's value is augmented with twelve 0's (2 on the left and 10 on the right side) so as to be transformed to the desired 10.10 fixed-point format. This is why registers Nbrhd[0]-Nbrhd[8] are marked as 20-bit wide (we remind that the pixels of the original image are 8-bit values). Also, any neighborhood pixels outside the image boundaries are considered as 0's. As far as the multiplications are concerned, from their 40-bit results, we keep the middle 20 (10 integer and 10 fractional

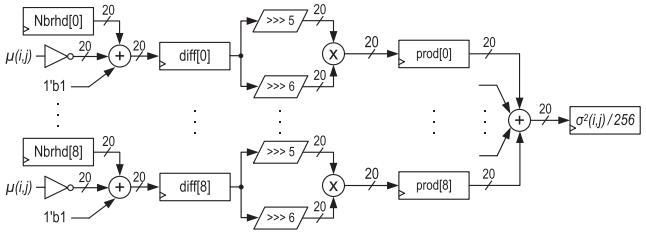


Fig. 6. $\sigma^2/256$-block datapath - parallel implementation ($<<<$, $>>>$ denote arithmetic shifts).

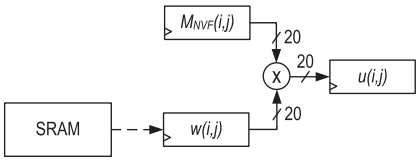


Fig. 7. The u-block.

bits with truncation). Registers prod[0]-prod[8] are used for clock-speed optimization purposes.

B. Implementation of the $\sigma^2/256$-block

The datapath of the parallel version of the $\sigma^2/256$-block is shown in Fig. 6. First, the difference of the neighborhood's mean value $\mu(i, j)$ from every neighborhood pixel is calculated (by 2's complement addition) and the results are stored in registers (diff[0]-diff[8]). This latching is done, again, to optimize clock speed. Then, the multiply-add structure of the μ-block (9 multipliers and 8 adders) is re-used to compute the sum of products of eq. (21). This structure is fed with the arithmetically right-shifted versions by 5 (divide by 32) and by 6 (divide by 64) of the *diff* registers, as explained in Section II-B (eq. (21), $d = 2$). The serial version of the $\sigma^2/256$-block needs, instead of nine, just one additional adder (but extra clock cycles) for performing the subtractions. In this case too, the multiplier and the adder of the serial μ-block are re-used by the serial $\sigma^2/256$-block.

C. Implementation of the u-block

The u-block, shown in Fig. 7, is very simple, since it involves just a multiplication between the calculated mask value ($M_{NVF}(i, j)$) and the corresponding watermark value ($w(i, j)$) retrieved from the SRAM.

D. Implementation of the α-block

The block calculating the watermark strength α is the most complex of our implementation and is shown in Fig. 8. Each $u(i, j)$ value is first multiplied by itself so as to be squared and is then added to the value of a register that keeps the temporary sum of the squared $u(i, j)$'s ($u^2\_sum\_temp$). When the value of that register becomes equal or greater than 462, as explained in Section II-A, it is sent through the upper multiplexer of Fig. 8 to the divider, and is divided first by $M/16$ and then by $N/16$. The final quotient is added to a register that keeps the partial $256 \cdot \|u\|^2$ value. The same process is repeated for all $M \times N$ $u(i, j)$'s and, when completed, the total $256 \cdot \|u\|^2$ value is first square-rooted and then fed to the divider (through the lower multiplexer) as a divisor of $A \times 16$. Note that we have pre-computed 16 different values of A, which correspond

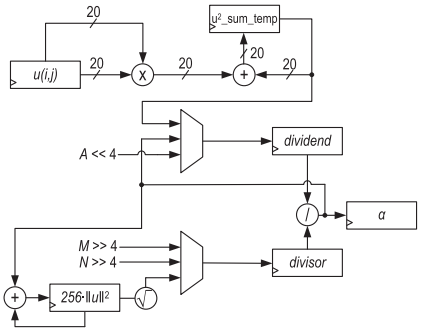


Fig. 8. α-block datapath ($<<$, $>>$ denote shifts).

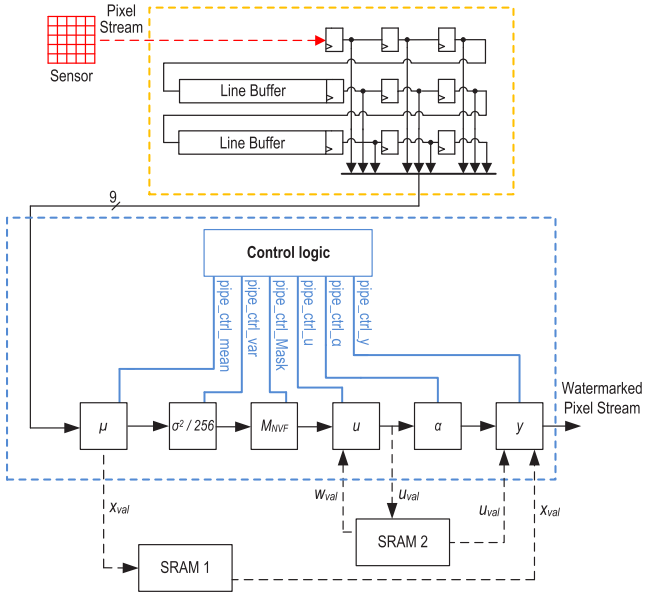


Fig. 9. Architecture of the implemented video-watermarking module.

to 16 different $PSNRs$ $(30-45\ dB)$. These values are stored in a small lookup table of our system.

As far as the arithmetic units are concerned, the square root gets a 20-bit input and outputs a 10-bit value, which corresponds to a 5.5 fixed-point integer in our implementation. For that reason, its outputs are augmented with 10 0's (5 on the left and 5 on the right side) to be transformed to the proper internal representation. As for the divider, to yield results with 10 fractional bits, the dividend should be padded with 10 0's on the right side ($\equiv \times 1024$). Hence, the corresponding register should be 30-bit wide. The quotient needs 30 bits as well, from which we keep the 20 least significant that correspond to the required 10.10 fixed-point result. Since division is an expensive operation, we pipelined the divider in order to optimize throughput. The same holds for the square root unit (please refer to Section V for details). Finally note that, since a division operation is also needed for the mask calculation (M_{NVF}-block), a single divider is shared between the M_{NVF}- and the α-block.

IV. Video Watermark Embedder Design

The diagram of the video watermark embedder is shown in Fig. 9. Its main difference, compared to the embedder for still images, is that it is pipelined, i.e., more than one frame pixels are processed concurrently. Since a 3×3 neighborhood

is used for the calculation of μ and $\sigma^2/256$, a window of the same size should effectively slide over each frame. Taking into account that pixel processing should be performed in real time, instead of sliding the window over a frame after storing it to a memory, its pixels are directly fed into shift registers, called line buffers. Line buffers constitute the most frequently utilized solution in such cases [43]. Due to the selected window dimensions (3×3), two line buffers along with three registers are sufficient for our implementation, as explained in [43] and shown in Fig. 9. By shifting the frame pixels in the line buffers, the required 3×3 window is always placed in the registers at the right side of the buffer block, as shown in Fig. 9.

The most important decision, when implementing our pipelined video architecture, was how often a new pixel will enter the embedder. First, in order to have high throughput, the parallel implementation of the μ- and $\sigma^2/256$-blocks were chosen (Figures 5 and 6). To keep implementation cost low, we again decided to share the required multiply-add structure of 9 multipliers and 8 adders between these two blocks. Hence, in our pipelined scheme we cannot compute concurrently the μ and the $\sigma^2/256$ values for the neighborhoods of two different pixels, which means that, according to this restriction, a new pixel should be fed to the embedder every two clock cycles.

Another restriction though comes from the number of memory accesses required for every processed pixel, which are totally five: two writes (the related x and u values) and three reads (the w, u and x values). For convenience, these accesses are shown in Fig. 9. Note that in Fig. 9, the two x and the two u values are different (they correspond to different pixels) due to the different pipeline stage the respective accesses happen. The reason for writing the x value and the u value to the SRAM is that they will be needed again later (this explains the corresponding reads), after α's computation, for calculating the final, watermarked pixel value. This practically means that a pixel that enters the pipeline does not go through all of its length at once, but it stops at the α-block and returns to the pipeline later (from the SRAM), for the final pixel calculation. Note though that this does not affect the pipeline's throughput, as will be explained in a while.

Returning back to the memory accesses, it is clear that with five accesses per pixel, we would need to feed our pipeline with, at max, one pixel per five cycles. Since this was unacceptable in terms of throughput, we decided to use two different SRAM modules (SRAM1 and SRAM2), in order to be able to access them simultaneously. Note that this design choice does not affect the total memory capacity that our architecture requires; SRAM1 stores the initial frame's (x) values, while SRAM2 stores the watermark (w) and the u values. With this arrangement, we can fit all five memory accesses into three clock cycles (the write of a x value and the read of a u value are done in the same clock cycle, while this applies to the read of a x value and the write of a u value). Consequently, we can push a new pixel to our pipeline every three clock cycles, which yields the desired throughput figures and, also, satisfies the previous requirement set for not duplicating the multiply-add structure of the μ- and $\sigma^2/256$-blocks.

In Fig. 10, the pipeline stages of the implemented video architecture, under "normal" operation, are shown.

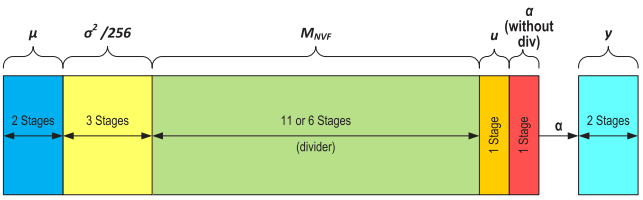


Fig. 10. Pipeline stages of the video watermarking architecture (normal operation).

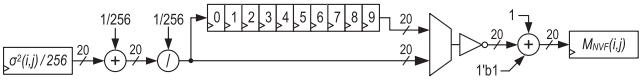


Fig. 11. M_{NVF}-block datapath for the video embedder.

The number of stages of the M_{NVF}-block depend on the pipeline stages of the divider, which are different for the two different implementations of our design, the FPGA- (11 stages) and the ASIC-based one (6 stages). With the term "normal" operation we refer to the case that no division is required by the α-block, which is the most frequent one. In the rare case that divisions should be performed by the α-block, we stall the pipeline and we use the same divider (in order to reduce cost), as will be explained in detail in Section IV-A.

As already mentioned, the computations for all $x(i, j)$ values of a frame go all the way up to the α-block, but do not directly proceed to the y-block. After α's calculation, the $x(i, j)$ values, along with the corresponding $u(i, j)$'s, are retrieved from the SRAM modules, where they have been stored, to compute the final, watermarked pixels' values ($y(i, j)$'s). This is why, in Fig. 10 we show the stages of the α- and the y-blocks disconnected, with an arrow between them. Note though that this has no effect on the pipeline's performance since, when a new frame starts entering the pipeline, the α's computation for the current one is being concluded. This means that a frame starts being shifted out of the pipeline when the next one starts entering it, which ensures a continuous data flow throughout the pipeline (apart, of course, from a few stalls due to divisions).

The datapath designs of the μ-, $\sigma^2/256$-, u- and α-blocks of the video watermark embedder are the same as those of Figures 5, 6, 7 and 8 for the image embedder. In the following, we discuss the implementation of the M_{NVF}- and the y-blocks that have some interesting aspects in the pipelined video case.

A. Implementation of the M_{NVF}-block for the Pipelined Video Architecture

The datapath of the M_{NVF}-block for the video embedder is shown in Fig. 11. As can be seen from eq. (7), in terms of arithmetic operations, it requires an addition, a division and a subtraction (i.e., addition). For the image watermark embedder, this block is actually constructed by the corresponding arithmetic units, with the divider pipelined as already explained. However, for still images, the divider is always available when required by either the M_{NVF}- or the α-block, because, as explained in Section III, only one of the computation blocks is used every time. This is not the case though for the video architecture; when the α-block needs the divider, the pipeline operates in normal mode and, hence, the divider is already used by the M_{NVF}-block.

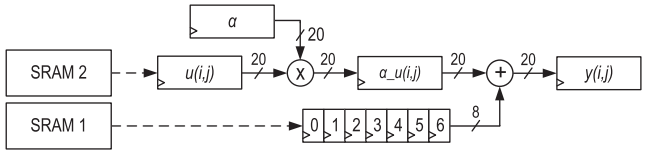


Fig. 12. The y-block for the video embedder.

As mentioned above, in such cases the pipeline is stalled to execute the α-block's divisions. Nevertheless, the divider's pipeline registers contain data of the M_{NVF}-block. Our decision was to capture the results of the corresponding divisions with the shift register placed at the output of the divider that is shown in Fig. 11. Therefore, when the α-block's divisions are over and the pipeline resumes its normal operation, what is sent to the second adder of the M_{NVF}-block through the multiplexer, are the contents of the shift register, which are the data that should have been sent to this adder if the stall had not happen. At the same time, the next normal-operation data are placed at the inputs of the divider. Since the length of the shift register is equal to the divider's internal-register levels (10 for our FPGA implementation, as shown in Fig. 11, and 5 for the ASIC), when all of its contents have been shifted out, the divider has new normal-operation results at its output, which are sent to the adder by complementing the control signal of the multiplexer. With this solution, we avoid backtracking the embedder when the remainder of the pipeline has to be stalled, so that the α-block uses the divider.

B. Implementation of the y-block for the Pipelined Video Architecture

The arithmetic-unit structure of the y-block is fairly simple, since it comprises a multiplier and an adder. There are two points though, in its video-architecture version (Fig. 12), that need to be discussed. The first one concerns the shift register, acting as a FIFO buffer, that we use between SRAM1 and the adder. This buffer holds a small number of x values before the final, watermarked pixel calculation. The reason for its existence, is that, due to the characteristics of the implemented algorithm, there is a very small overlap between the current and the next frame. This buffer is used for "absorbing" the overlapping pixels, i.e., the pixel of the current frame that overlaps with the corresponding pixel of the next frame is sent from SRAM1 to the buffer, while that of the next frame is stored in SRAM1. The size of this FIFO buffer depends on the pipeline depth, and for our FPGA implementation, which has the deepest pipeline, is equal to 7.

The second thing that we should mention concerns the latching of the $\alpha \cdot u(i, j)$ multiplication's result in register $a\_u(i, j)$, before its final addition with $x(i, j)$. This latching does not have to do with the clock speed, but with the fact that, according to our pipeline design, we need to perform a u value read and a x value read in two (consecutive) clock cycles before the computation of the final $y(i, j)$.

V. Experimental Results and Comparisons

A. Hardware Evaluation

In terms of image watermarking, both the serial and the parallel version of the described embedder were implemented in Verilog and mapped to the Cyclone IV E EP4CE115F29C7 FPGA device, using Intel's (ex Altera) Quartus Prime tool, and to an industrial 90nm standard-cell library (ASIC), using a commercial synthesis tool. The Cyclone IV FPGA is the main chip of Terasic's DE2-115 board, which we used, apart from post-synthesis simulations, for verifying our image-watermarking designs. Thus, along with the embedder and the necessary SRAM interface, we also implemented a board interface in order to be able to control all system's operations during our hands-on experiments. We should also mention that all simulation results were cross-checked and verified against a fixed-point Matlab model of the optimized watermarking algorithm that we developed. The synthesis results of the implemented image watermark embedders, along with some important design information are presented in Table III. Please note that the reported FPGA results are after the "Place & Route" step (i.e., we ran the whole compilation flow in Quartus Prime), whereas for our ASIC implementations we report synthesis results (pre-layout). The same holds for the video-embedder results that will be presented in a while.

From Table III, it can be seen that the total number of arithmetic units has been kept very low: 10 (33) overall in the serial (parallel) implementation, 7 (22) of which are adders. Observe that we have limited the usage of the costly division unit to just 1 instance, which is shared among different blocks. The same holds for the multipliers, the total number of which depends on the serial or parallel implementation of the μ- and $\sigma^2/256$- blocks (any other multipliers needed are from this set). In our designs, we have not restricted only the adders' volume, due to their small size (20 bits). As for the pipelined units (divider and square root), different stage volumes were chosen for each implementation, depending on performance criteria.

Throughput results for the various versions of our image watermark embedder are shown in the upper part (above the double horizontal line) of Table V. These figures have been calculated by averaging the clock cycles needed for various 640×480 and 1280×720 grayscale images (small discrepancies exist, since the number of divisions during α's computation is not fixed). It is interesting to observe that the parallel ASIC version of our image embedder achieves a throughput of 30.1 $images/sec$ (i.e., video rate) for images of size 640×480. Note that in our architecture, all SRAM read/write cycles, apart from those in the y's calculation, are overlapped with computation cycles.

The results for the video watermark embedder are shown in Table IV. For our FPGA implementation we provide the highest operating frequency result, while for the ASIC case we give three different options (100, 140.8 and 181.8MHz – the highest frequency achieved by our design) that have different characteristics in terms of performance and area/power consumption. All designs though offer frame rates well above 30 frames per second, for frame sizes of 640×480 and 1280×720, as shown in Table V. Note that the reported frame rates are for watermark embedding in every raw video frame.

From Table IV we can see that, apart from three multipliers, we have not increased the number of utilized arithmetic units, as compared to the parallel image embedder. These three extra

TABLE III
DESIGN CHARACTERISTICS AND SYNTHESIS RESULTS FOR THE IMAGE WATERMARK EMBEDDERS

	Cyclone IV E		**ASIC**	
	Serial Impl.	Parallel Impl.	Serial Impl.	Parallel Impl.
Area [FPGA: # Logic Elements, ASIC: mm^2]	4399	4582	1.52	2.89
Total 1-bit Registers (FPGA only)	1969	1968	-	-
Clock frequency (MHz)	88.69	79.84	181.82	166.7
Power Consumption (mW)	201.72	205.16	4.69	4.23
# Integer arith. units (add, mul, div, sqrt)	7, 1, 1, 1	22, 9, 1, 1	7, 1, 1, 1	22, 9, 1, 1
# Pipeline stages (div, sqrt)	12, 4	11, 2	5, 2	6, 2

TABLE IV
DESIGN CHARACTERISTICS AND SYNTHESIS RESULTS FOR THE VIDEO WATERMARK EMBEDDER

	Cyclone IV E	**ASIC**		
		Low Speed	Med. Speed	High Speed
Area [FPGA: # Logic Elements, ASIC: mm^2]	2362	2.22	2.43	2.68
Total 1-bit Registers (FPGA only)	1291	-	-	-
Clock frequency (MHz)	88.03	100	140.8	181.8
Power Consumption (mW)	227.13	3.27	4.79	7.8
# Integer arith. units (add, mul, div, sqrt)	22, 12, 1, 1	22, 12, 1, 1	22, 12, 1, 1	22, 12, 1, 1
# Pipeline stages (div, sqrt)	11, 2	6, -	6, -	6, -

TABLE V
IMAGE AND VIDEO EMBEDDERS THROUGHPUT

Implementation	**640×480**	**1280×720**
FPGA (Image – Serial)	6.9 $im./sec$	2.3 $im./sec$
FPGA (Image – Parallel)	11.8 $im./sec$	3.9 $im./sec$
ASIC (Image – Serial)	16.9 $im./sec$	5.6 $im./sec$
ASIC (Image – Parallel)	30.1 $im./sec$	10 $im./sec$
FPGA (Video)	93.7 fps	31.2 fps
ASIC (Video – 100MHz)	106.5 fps	35.4 fps
ASIC (Video – 140.8MHz)	150 fps	49.9 fps
ASIC (Video – 181.8MHz)	193.6 fps	64.4 fps

multipliers are necessary in the pipelined video case, due to the fact that multiple blocks of the embedder are concurrently active. Another observation is that in the pipelined arithmetic units, we have kept the same number of stages as in our parallel image implementation, except for the square root unit in the ASIC video implementation, whose pipelining did not offer any additional advantages in terms of clock speed. Finally, it can be seen that the video watermark embedder is less area consuming than its parallel image counterpart. The reason for this is twofold: first, as explained in the beginning of this Section, the image embedder was built as a stand-alone system in the DE2-115 board, which means that there is a lot of logic for controlling irrelevant with the embedder parts of the system (7-segment displays, LEDs, switches, etc.). Most importantly though, in the image embedder there is a complex control logic for reading 3×3 pixel neighborhoods from the SRAM. In the video embedder, such a logic does not exist since every neighborhood is readily available in the line buffers.

In Table VI we summarize the main characteristics of various already existing image watermarking architectures, along with those of the proposed ones. For convenience, in the first 7 rows of the table, FPGA implementations are reported, while in the remaining 6 rows (after the double horizontal line) we provide the characteristics of ASIC-based architectures. Similarly, the characteristics of various video watermarking architectures that can be found in the literature, implemented in both FPGA and ASIC devices, are presented in Table VII.

Since the existing literature designs are mapped to various FPGA devices or ASIC technologies, comparisons with them are not straightforward. Regarding the FPGA devices in Tables VI and VII, notice that different kinds of logic blocks (LABs, LUTs, LEs or LCs) are reported. This is due to the different architectures and features of the various FPGA devices. Device capacity though, is often measured in terms of logic cells (LCs) or logic elements (LEs), which are the logic equivalents of a classic four-input LUT and a single-bit register. This configuration actually applies to the FPGA device of this work, as well as to those of [15], [16], [26] and [27]. Work [8] reports the number of LABs of the CPLD device it uses, with each LAB consisting of ten logic cells. The device in [17] features six-input LUTs, with four times more SRAM cells than a four-input one. Works [9] and [23] do not report area utilization, whereas [22] provides results in gate count (KGates), which is not usually used to characterize the size of FPGA implementations, and without mentioning a particular device. As far as FPGA embedded blocks are concerned, the utilized Cyclone IV device contains only embedded multipliers, which is also the case for all the compared techniques in Tables VI and VII, expect for [8] and [27]. From this discussion it is clear that the LEs of our implementations, occupy the same or less area than the logic blocks of the works we compare with, and that their corresponding volumes are representative and, hence, suitable for cost comparison.

Concerning ASIC implementations, the situation is more complex. Almost all works shown in Tables VI and VII provide area results only in mm^2 and not in gate equivalents, while many of them ([10]–[12] and [21]) report post-layout results (as mentioned above, we report synthesis results). Hence, the ASIC results of Tables VI and VII are not directly comparable. We present them though, to provide an overview of the characteristics of the ASIC-based works in the literature.

Thus, among the FPGA-based designs in Tables VI and VII, we observe that the proposed one offers the smallest power consumption and the second smallest area overhead in the image-watermarking case. This is due to the few, small-sized arithmetic units it incorporates, and demonstrates that

TABLE VI
COMPARISONS WITH OTHER IMAGE WATERMARKING ARCHITECTURES

Research Work	Design Type	Processing Domain	Watermark Type	Area (Logic blocks or mm^2)	Clock Freq. (MHz)	Power (mW) (Voltage)
[8]	FPGA (Altera APEX20KC)	Wavelet	Invisible Robust	4037 LABs	82	–
[9]	FPGA (Xilinx Virtex-II)	Spatial	Invisible Robust	–	50.398	–
[15]	FPGA (Xilinx Virtex-II Pro)	Spatial	Invisible Robust	1669 LUTs	82.26	1300
[16]	FPGA (Xilinx Spartan-3E)	Spatial	Reversible	11291 LUTs	98.76	750
[17]	FPGA (Xilinx Virtex-7)	Spatial	Reversible	50124 LUTs	445.33	1215
Proposed (ser.)	FPGA (Altera Cyclone IV E)	Spatial	Invisible Robust	4399 LEs	88.69	201.72
Proposed (par.)	FPGA (Altera Cyclone IV E)	Spatial	Invisible Robust	4582 LEs	79.84	205.16
[7]	ASIC (0.35μ)	DCT	Invisible Robust	3.064	50	62.78 (3.3V)
[10]	ASIC (0.35μ)	Spatial	Invisible Robust/Fragile	–	545	2.055 (3.3V)
[11]	ASIC (0.25μ)	DCT	Invisible Robust Visible	16.2	70, 280	0.3 (1.5, 2.5V)
[12]	ASIC (0.18μ)	Spatial	Invisible Robust	1.46	–	– (1.8/3.3V)
Proposed (ser.)	ASIC (0.09μ)	Spatial	Invisible Robust	1.52	181.82	4.69 (1V)
Proposed (par.)	ASIC (0.09μ)	Spatial	Invisible Robust	2.89	166.7	4.23 (1V)

TABLE VII
COMPARISONS WITH OTHER VIDEO WATERMARKING ARCHITECTURES

Research Work	Design Type	Processing Domain	Watermark Type	Area (Logic blocks or mm^2)	Clock (MHz)	Power (mW) (Voltage)	Frame size	Frame rate (fps)
[22]	FPGA	Spatial	Invis. Robust	17 KG	–	–	–	–
[23]	FPGA	Fractal	Invis. Robust	–	50	–	512×512	–
[26]	FPGA (Altera Cyclone-II)	DCT	Visible	16566 LEs	100	–	320×240	43
[27]	FPGA (Altera Cyclone)	DCT	Semifragile	9263 LCs	40	270	640×480	130
Proposed	FPGA (Altera Cyclone IV E)	Spatial	Invis. Robust	2362 LEs	88.03	227.13	1280×720	31.2
[21]	ASIC (0.18μ)	Spatial	Invis. Robust	3.53	75	60 (1.8V)	320×320	30
[22]	ASIC	Spatial	Invis. Robust	14 KG	–	–	–	–
[28]	ASIC (0.18μ)	DCT	Invis. Robust	5.25	120	104.8 (–)	720×486	–
Prop. (Lo Sp.)	ASIC (0.09μ)	Spatial	Invis. Robust	2.22	100	3.27 (1V)	1280×720	35.4
Prop. (Hi Sp.)	ASIC (0.09μ)	Spatial	Invis. Robust	2.68	181.8	7.8 (1V)	1280×720	64.4

the main target of this work, i.e. low cost, has been achieved. Moreover note that the aforementioned area/power advantages are offered with quite high clock frequencies (88.7 and 79.8MHz). Concerning video, the advantages of our bare, video watermark embedder are obvious: it offers the smallest area and power-consumption overheads, while it operates at a high enough clock frequency to support a frame rate above 30 frames per second, for frames of size 1280×720, which are the largest among the compared works. As for our ASIC implementations, we can note that they support fast clock speeds, offering high throughput with fairly small area and power overhead.

B. Verification of Low Quantization Error Optimizations

In order to verify the quantization error optimizations proposed in Section II-B, experiments were conducted for 100 different images of size 1280×720. First, the local mean values of each image were computed via eq. (13), for $\delta = 0$ and $\delta = 8$, and were compared to those generated in Matlab with floating-point double-precision representation. The mean absolute errors were computed for all images and their mean value was approximately equal to 0.77429, for $\delta = 0$, and 0.00487, for $\delta = 8$ (actually reduced by more than two orders of magnitude).

Then, with $\delta = 8$ and $d = 2$, 100 different watermarks were embedded in each of the examined images, for $PSNR$ values 30, 35, 40 and 45 dB, resulting in 40000 watermarked images (10000 for each $PSNR$ value – we remind that, contrary to other architectures, in the proposed one, all $PSNR$ options between 30 and 45 dB are available to the user to choose from). We computed the mean squared error (MSE) among the images watermarked as proposed in this paper and those generated in Matlab with floating-point double-precision representation. Some error results are shown in Fig. 13, for 100 instances of one image. Each of them is computed after embedding a different watermark in the image. Also, each watermark is embedded with four different strength values (α), corresponding to the $PSNR$ values of 30, 35, 40 and 45 dB. Note that the case of watermark embedding with $PSNR$ equal to 30 dB is not very common, since the image will be heavily watermarked, which leads to perceptual distortions. $PSNR$ values higher than 35 dB are, usually, more acceptable. However, a 30 dB $PSNR$ is used here for testing purposes and, as shown in Fig. 13, the errors are quite low, considering

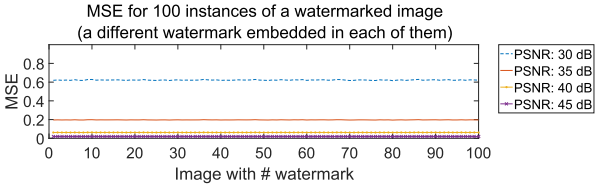


Fig. 13. Mean squared errors for different PSNR values.

the utilized 10.10 signed fixed-point representation. For the remaining $PSNR$ values, the errors are even smaller. The mean $MSEs$, computed for all the images' instances and $PSNR$ values 30, 35, 40 and 45 dB, are 0.35891, 0.11348, 0.03588 and 0.01134, respectively, which are fairly small.

C. Robustness Against Attacks

As it has been already mentioned in Section II, the M_{NVF}-based watermarking was designed to be robust to the denoising attack. Such a technique is expected to be also robust against compression, since both denoising and compression affect the high-frequency information. The technique of [29] that we are based on, has been extensively tested in [4] for comparison reasons, and has been shown to exhibit particularly good behavior against JPEG compression (and obviously against MJPEG that is used for video compression). For a thorough analysis of the M_{NVF}-based image watermarking robustness, refer to [29] and [4].

We examine here the robustness of our implementation against video compression with modern standards, such as H.264/AVC and HEVC. For our experiments' demonstration, we used a standard test video sequence of resolution 1280×720, *FourPeople* [44]. We took a subsequence consisting of the first 10 frames, which was converted and watermarked according to the utilized in this paper 10.10 signed fixed-point representation. The watermark was embedded only in the first frame (in its luminance component, Y), for PSNR equal to 40 dB. For the compression of the watermarked video sequence, FFmpeg [45] was used, employing the x264 encoder for H.264/AVC and the x265 one for HEVC, with their predefined parameters.[1] In both cases, the first frame was encoded as I frame, while the rest ones as P or B frames. Due to the way of P or B frames encoding (inter-prediction), we expect that the watermark will be present to these frames, too. Note that it is common to watermark only the I frames, in order to avoid visual artifacts during decompression [27].

Then, the compressed video was converted to raw (.yuv) format, again with FFmpeg, and the Y component of each frame was examined for watermark existence. Blind detection was used, which means that the original video was considered unavailable and only the key for the watermark generation was known. The detection task can be formulated as a binary hypothesis testing problem, where hypothesis H1 corresponds to watermark presence, while hypothesis H0 corresponds to watermark absence. We conducted 100 experiments. In each of them, a different watermark was embedded in the video sequence. Then, the detection process was performed for all the frames, with the right key and with a wrong one. The detection scheme employed is the one proposed in [4], which consists of two steps, a prewhitening (conducted with the least-squares prediction error filter), applied to the received frames and the estimated masked watermark, and the computation of the normalized correlation for the prewhitened parts.

[1]Constant Rate Factor (CRF) and Quantization Parameter (QP) are two of them, which take values from 0 for lossless compression, to 51 for the worst quality. The default CRF is 23 for x264 and 28 for x265. QP varies among frames to maintain a certain level of perceived quality. For the conducted experiments, the average QP was about 27 for x264 and 34 for x265.

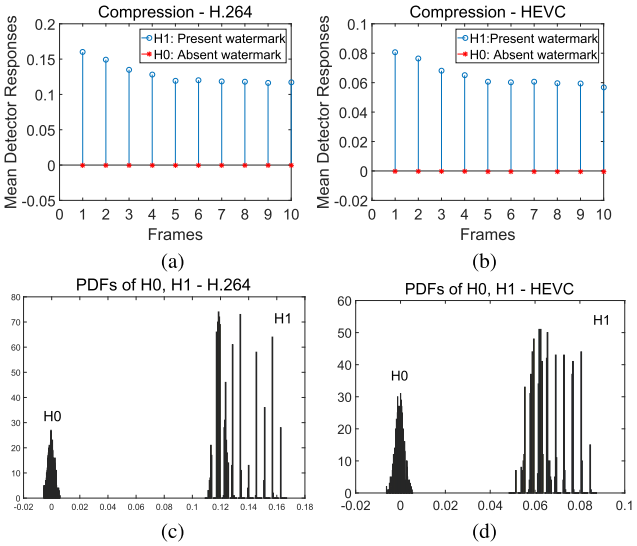


Fig. 14. Robustness to compression under H.264 and HEVC.

Fig. 15. Frame (Y component) with missing blocks (after compression).

The mean detector responses for the aforementioned experiments are shown in Figures 14a and 14b for H.264 and HEVC, respectively. The watermark is detected in all 10 frames and also the probability of detection equals 1. This is also demonstrated in Figures 14c and 14d (for H.264 and HEVC, respectively), where the probability density functions (PDFs) of H0 and H1 are shown to not overlap each other. Hence, our implementation resists compression with H.264/AVC and HEVC. Note that robustness against HEVC implies also robustness to BPG [46], which is an image compression format, based on HEVC coding of intraframes.

We continue with transmission losses and, specifically, with frames' missing blocks. The compressed video sequences (with H.264 and HEVC) were converted to raw format and we considered that their frames (Y components) suffered from missing blocks corresponding to 30% of their pixels. Such a frame example is shown in Fig. 15. In our experiments, black boxes of dimensions 40×40 were randomly inserted into each frame. As shown in Fig. 16, where the mean detector responses are depicted, the watermark resists this attack, which comprises compression (with H.264/HEVC) and missing blocks on top of compression.

The sequence of the 10 frames used in our experiments could correspond to a Group of Pictures (GOP). The watermark is embedded only in frame I, but is also present in frames P or B of the same GOP. This can be helpful in some cases such as frame dropping, since the watermark can be detected to the remaining frames of the GOP. Also, frame swapping will not affect the watermark detection in a GOP.

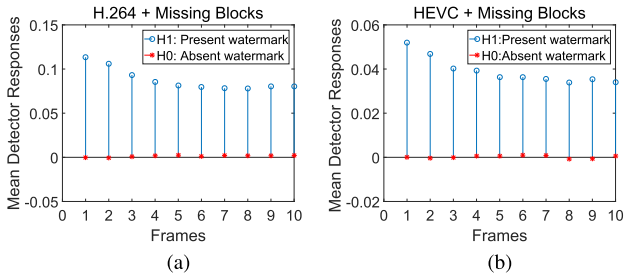


Fig. 16. Robustness to compression plus missing blocks.

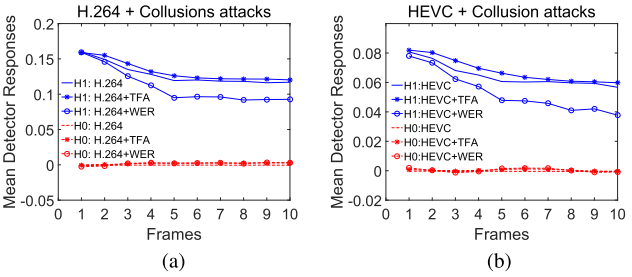


Fig. 17. Robustness to compression plus collusion attacks.

If the frame rate is doubled by inserting frames between the existing ones, the watermark will be detected every second frame. Of course, proper considerations should be made by the watermark detection process.

Since we detect the watermark in all the frames of the examined sequence, we tested its robustness against collusion attacks. We began with the temporal frame averaging (TFA) attack. Since neighboring video frames are similar, temporal lowpass filtering can be performed without introducing much visual distortion. A 3-frame temporal averaging filter was used in our experiments for performing the attack, as in [47]. TFA failed, because the frames belonging to the same GOP are correlated and contain the same watermark. In such cases, TFA attack strengthens rather than destroys the watermark [30], [47]. This is actually depicted in Fig. 17, where the mean detector responses are shown for the H.264 and HEVC compressed video sequences, as compared to the responses obtained when they were additionally attacked by TFA or watermark estimation re-modulation (WER – see below).

In the case of WER attack, an estimate of the watermark is obtained by averaging initial watermark estimations, extracted from a large number of frames in a video sequence. These initial estimations are obtained as differences between the frames and their low-pass filtered versions (3×3 averaging filter was used in our experiments). Then, the watermark estimate is properly remodulated so as to introduce distortion similar to that of the watermarking process [47]. The watermark here, as shown in Fig. 17, resists WER attack. Note that in our experiments the length of the used sequence (i.e. the number of frames with the same watermark) is not large. In any case, GOP length can be adjusted so as to prevent watermark estimation and extraction (and also different watermarks could be embedded to different GOPs). Moreover, recall that watermark embedding is developed to resist denoising. So, it resists lowpass filtering that is used in this attack.

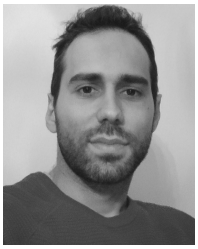
VI. Conclusion

We have presented image and video watermarking hardware architectures that suit well the needs of low cost applications. The proposed designs offer low area overhead and power consumption due to the introduced computation optimizations that keep the utilized integer arithmetic units small, and due to the reuse of the latter in different computation steps that limits their number. Moreover, a quantization error reduction analysis for the first two steps (local mean and local variance computation) of the implemented algorithm is provided, which contributes to the improvement of the final fixed-point results. The application of this analysis' results to our hardware architectures is effortless and imposes no additional area cost. The low area and power consumption characteristics of the proposed architectures have been verified with comparisons against various different implementations in the literature, while the proposed pipelined video watermarking architecture offers good to excellent frame rates (31.2 fps for a relatively low-end FPGA device, and 35.4 – 64.4 fps for ASIC implementations with different clock speeds), for frames of size 1280×720. Finally, robustness to the most up-to-date compression standards and various other attacks has been demonstrated.

References

[1] Q. M. Ashraf and M. H. Habaebi, "Autonomic schemes for threat mitigation in internet of things," *J. Netw. Comput. Appl.*, vol. 49, pp. 112–127, Mar. 2015.
[2] D. D. O. Gonçalves and D. G. Costa, "A survey of image security in wireless sensor networks," *J. Imag.*, vol. 1, no. 1, pp. 4–30, 2015.
[3] R. Xiao, X. Sun, and Y. Yang, "Copyright protection in wireless sensor networks by watermarking," in *Proc. Int. Conf. Intell. Inf. Hiding Multimedia Signal Process.*, Aug. 2008, pp. 7–10.
[4] I. G. Karybali and K. Berberidis, "Efficient spatial image watermarking via new perceptual masking and blind detection schemes," *IEEE Trans. Inf. Forensics Security*, vol. 1, no. 2, pp. 256–274, Jun. 2006.
[5] A. K. Singh, B. Kumar, G. Singh, and A. Mohan, *State-of-the-Art Techniques of Image Watermarking: New Trends and Future Challenges*. Cham, Switzerland: Springer, 2017, pp. 227–244.
[6] G. Vrettos, E. Logaras, and E. Kalligeros, "Towards standardization of MQTT-alert-based sensor networks: Protocol structures formalization and low-end node security," in *Proc. IEEE 13th Int. Symp. Ind. Embedded Syst. (SIES)*, Jun. 2018, pp. 1–4.
[7] T. H. Tsai and C. Y. Lu, "A systems level design for embedded watermark technique using DSC systems," in *Proc. IEEE Int. Workshop Intell. Signal Process. Commun. Syst.*, Nov. 2001, pp. 20–23.
[8] Y.-H. Seo and D.-W. Kim, "Real-time blind watermarking algorithm and its hardware implementation for motion JPEG2000 image codec," in *Proc. 1st Workshop Embedded Syst. Real-Time Multimedia*, 2003, pp. 88–93.
[9] S. P. Mohanty, C. R. Kumara, and S. Nayak, "FPGA based implementation of an invisible-robust image watermarking encoder," in *Proc. Int. Conf. Intell. Inf. Technol.*, vol. 3356, 2004, pp. 344–353.
[10] S. P. Mohanty, E. Kougianos, and N. Ranganathan, "VLSI architecture and chip for combined invisible robust and fragile watermarking," *IET Comput., Digit. Techn.*, vol. 1, no. 5, pp. 600–611, Sep. 2007.
[11] S. P. Mohanty, N. Ranganathan, and K. Balakrishnan, "A dual voltage-frequency VLSI chip for image watermarking in DCT domain," *IEEE Trans. Circuits Syst. II, Express Briefs*, vol. 53, no. 5, pp. 394–398, May 2006.
[12] G. R. Nelson, G. A. Jullien, and O. Yadid-Pecht, "CMOS image sensor with watermarking capabilities," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2005, pp. 5326–5329.
[13] Y. Shoshan, A. Fish, G. A. Jullien, and O. Yadid-Pecht, "Hardware implementation of a DCT watermark for CMOS image sensors," in *Proc. IEEE Int. Conf. Electron., Circuits Syst.*, Aug./Sep. 2008, pp. 368–371.
[14] S. D. Roy and O. Yadid-Pecht, "Design and implementation of hardware based watermarking solutions for CMOS image sensors," in *Proc. 10th IEEE Int. NEWCAS Conf.*, Jun. 2012, pp. 341–344.

[15] S. Ghosh, S. Talapatra, N. Chatterjee, S. P. Maity, and H. Rahaman, "FPGA based implementation of embedding and decoding architecture for binary watermark by spread spectrum scheme in spatial domain," *Bonfring Int. J. Adv. Image Process.*, vol. 2, no. 4, pp. 1–8, Dec. 2012.
[16] H. K. Maity and S. P. Maity, "FPGA implementation of reversible watermarking in digital images using reversible contrast mapping," *J. Syst. Softw.*, vol. 96, pp. 93–104, Oct. 2014.
[17] S. Hazra, S. Ghosh, S. De, and H. Rahaman, "FPGA implementation of semi-fragile reversible watermarking by histogram bin shifting in real time," *J. Real-Time Image Process.*, vol. 14, no. 1, pp. 193–221, Jan. 2018.
[18] H. R. Lakshmi and B. Surekha, "Asynchronous implementation of reversible image watermarking using mousetrap pipelining," in *Proc. IEEE 6th Int. Conf. Adv. Comput. (IACC)*, Feb. 2016, pp. 529–533.
[19] L. De Strycker *et al.*, "Implementation of a real-time digital watermarking process for broadcast monitoring on a TriMedia VLIW processor," *IEE Proc.–Vis., Image Signal Process*, vol. 147, no. 4, pp. 371–376, Aug. 2000.
[20] N. J. Mathai, D. Kundur, and A. Sheikholesami, "Hardware implementation perspectives of digital video watermarking algorithms," *IEEE Trans. Signal Process.*, vol. 51, no. 4, pp. 925–938, Apr. 2003.
[21] N. J. Mathai, A. Sheikholesami, and D. Kundur, "VLSI implementation of a real-time video watermark embedder and detector," in *Proc. IEEE Int. Symp. Circuits Syst.*, vol. 2, May 2003, pp. 772–775.
[22] M. Maes, T. Kalker, J.-P. M. G. Linnartz, J. Talstra, F. G. Depovere, and J. Haitsma, "Digital watermarking for DVD video copy protection," *IEEE Signal Process. Mag.*, vol. 17, no. 5, pp. 47–57, Sep. 2000.
[23] G. Petitjean, J. L. Dugelay, S. Gabriele, C. Rey, and J. Nicolai, "Towards real-time video watermarking for system-on-chip," in *Proc. IEEE Int. Conf. Multimedia Expo*, Aug. 2002, pp. 597–600.
[24] Y.-J. Jeong, K.-S. Moon, and J.-N. Kim, "Implementation of real time video watermark embedder based on Haar wavelet transform using FPGA," in *Proc. 2nd Int. Conf. Future Gener. Commun. Netw. Symposia*, Dec. 2008, pp. 63–66.
[25] S. P. Mohanty, "A secure digital camera architecture for integrated real-time digital rights management," *J. Syst. Archit.*, vol. 55, nos. 10–12, pp. 468–480, Oct./Dec. 2009.
[26] S. P. Mohanty and E. Kougianos, "Real-time perceptual watermarking architectures for video broadcasting," *J. Syst. Softw.*, vol. 84, pp. 724–738, May 2011.
[27] S. D. Roy, X. Li, Y. Shoshan, A. Fish, and O. Yadid-Pecht, "Hardware implementation of a digital watermarking system for video authentication," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 23, no. 2, pp. 289–301, Feb. 2013.
[28] A. M. Joshi, V. Mishra, and R. M. Patrikar, "Design of real-time video watermarking based on integer DCT for H.264 encoder," *Int. J. Electron.*, vol. 102, no. 1, pp. 141–155, 2015.
[29] S. Voloshynovskiy, A. Herrigel, N. Baumgaertner, and T. Pun, "A stochastic approach to content adaptive digital image watermarking," in *Proc. 3rd Int. Workshop Inf. Hiding*, Sep. 1999, pp. 211–236.
[30] M. Asikuzzaman and M. R. Pickering, "An overview of digital video watermarking," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 28, no. 9, pp. 2131–2153, Sep. 2018.
[31] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, pp. 560–576, Jul. 2003.
[32] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the high efficiency video coding (HEVC) standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1649–1668, Dec. 2012.
[33] K. Pexaras, C. Tsiourakis, I. G. Karybali, and E. Kalligeros, "Optimization and hardware implementation of image watermarking for low cost applications," in *Proc. 24th IEEE Int. Conf. Electron., Circuits Syst. (ICECS)*, Dec. 2017, pp. 347–350.
[34] A. Papoulis and S. U. Pillai, *Probability, Random Variables and Stochastic Processes*, 4th ed. New York, NY, USA: McGraw-Hill, 2002.
[35] A. A. Gaffar, O. Mencer, and W. Luk, "Unifying bit-width optimisation for fixed-point and floating-point designs," in *Proc. 12th Annu. IEEE Symp. Field-Program. Custom Comput. Mach.*, Apr. 2004, pp. 79–88.
[36] J. Stolfi and L. H. De Figueiredo, "Self-validated numerical methods and applications," in *Brazilian Mathematics Colloquium Monograph*. Rio De Janeiro, Brazil: IMPA, 1997.
[37] R. E. Moore, *Methods and Applications of Interval Analysis*. Philadelphia, PA, USA: SIAM, 1979.
[38] C. F. Fang, R. A. Rutenbar, and T. Chen, "Fast, accurate static analysis for fixed-point finite-precision effects in dsp designs," in *Proc. Int. Conf. Comput. Aided Design (ICCAD)*, Nov. 2003, pp. 275–282.
[39] D.-U. Lee, A. A. Gaffar, O. Mencer, and W. Luk, "Minibit: Bit-width optimization via affine arithmetic," in *Proc. 42nd Design Autom. Conf.*, Jun. 2005, pp. 837–840.
[40] J. Cong *et al.*, "Evaluation of static analysis techniques for fixed-point precision optimization," in *Proc. 17th IEEE Symp. Field Program. Custom Comput. Mach.*, Apr. 2009, pp. 231–234.
[41] A. Benkrid and K. Benkrid, "A statistical framework to minimise and predict the range values of quantisation errors in fixed-point FIR filters architectures," *Digit. Signal Process.*, vol. 23, pp. 453–469, Jan. 2013.
[42] T. Hilaire and B. Lopez, "Reliable implementation of linear filters with fixed-point arithmetic," in *Proc. IEEE Workshop Signal Process. Syst. (SiPS)*, 2013, pp. 401–406.
[43] C. Pal, A. Kotal, A. Samanta, A. Chakrabarti, and R. Ghosh, "Design space exploration for image processing architectures on FPGA targets," *CoRR*, vol. abs/1404.3877, pp. 1–19, Apr. 2014. [Online]. Available: https://arxiv.org/abs/1404.3877
[44] *Xiph.org Video Test Media (Derf's Collection)*. Accessed: Feb. 6, 2019. [Online]. Available: https://media.xiph.org/video/derf/
[45] *FFmpeg*. Accessed: Feb. 6, 2019. [Online]. Available: https://www.ffmpeg.org/
[46] (Apr. 21, 2018). *BPG Image Format*. [Online]. Available: https://bellard.org/bpg/
[47] G. Doerr and J.-L. Dugelay, "Security pitfalls of frame-by-frame approaches to video watermarking," *IEEE Trans. Signal Process.*, vol. 52, no. 10, pp. 2955–2964, Oct. 2004.



Konstantinos Pexaras (S'18) received the Diploma degree from the Department of Information and Communication Systems Engineering, University of the Aegean, Greece, in 2019, where he is currently pursuing the Ph.D. degree. He is also a Verification Engineer with HDL Design House, Greece. His research interests are in the area of digital circuits' design for signal and image processing.

Irene G. Karybali (M'06) received the Diploma degree in computer engineering and informatics, the M.Sc. degree in signal and image processing systems, and the Ph.D. degree in efficient image registration and image watermarking techniques from the University of Patras, Patras, Greece, in 1999, 2001, and 2005, respectively. From 2002 to 2003, she was a Researcher with the R&D Department, Computer Technology Institute (CTI), Patras. She has been an Adjunct Lecturer with the Technological Educational Institute of Larissa, Universities of Thessaly and Patras, and the Department of Information and Communication Systems Engineering, University of the Aegean, where she is currently. Her research interests include image watermarking, image registration, optimization of digital image processing algorithms for efficient hardware implementation, and hardware security. She is a member of the Technical Chamber of Greece.

Emmanouil Kalligeros (M'06) received the Diploma degree in computer engineering and informatics, the M.Sc. degree in computer science and technology, and the Ph.D. degree in embedded testing techniques from the Computer Engineering and Informatics Department, University of Patras, Patras, Greece, in 1999, 2001, and 2005, respectively. From 2006 to 2008, he was an Adjunct Professor with the University of Patras, the University of Peloponnese, and the University of the Aegean, Samos, Greece, teaching courses related to electronics and digital circuits design. Since 2008, he has been a Faculty Member with the Information and Communication Systems Engineering Department, University of the Aegean, where he is currently an Assistant Professor. His main research interests include digital circuits and systems design and test, hardware security, network-on-chip (NoC) architectures, and design for testability. In these areas, he has published more than 40 papers in prestigious international journals and conferences and has also coauthored a book. He is a member of the Technical Chamber of Greece. In 2015, he received the Award for Excellence in Teaching of the School of Sciences, University of the Aegean.