Received May 13, 2019, accepted June 19, 2019, date of publication June 24, 2019, date of current version July 12, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2924490

SMFrWF: Segmented Modified Fractional Wavelet Filter: Fast Low-Memory Discrete Wavelet Transform (DWT)

MOHD TAUSIF[1], (Student Member, IEEE), EKRAM KHAN[1], (Senior Member, IEEE), MOHD HASAN[1], (Senior Member, IEEE), AND MARTIN REISSLEIN[2], (Fellow, IEEE)
[1]Department of Electronics Engineering, Aligarh Muslim University, Aligarh 202002, India
[2]Goldwater Center, School of Electrical, Computer, and Energy Engineering, Arizona State University, Tempe, AZ 85287-5706, USA

Corresponding author: Martin Reisslein (reisslein@asu.edu)

The work of M. Tausif was supported by the Ministry of Electronics and Information Technology, Government of India, through the Visvesvaraya Ph.D. Scheme.

ABSTRACT This paper proposes a novel algorithm to compute the 2-D discrete wavelet transform (DWT) of high-resolution (HR) images on low-cost visual sensor and Internet of Things (IoT) nodes. The main advantages of the proposed segmented modified fractional wavelet filter (SMFrWF) are reduced computation (time) complexity and energy consumption compared to the state-of-the-art low-memory 2-D DWT computation methods. In particular, the conventional convolution-based DWT is very fast but requires large random access memory (RAM), as the entire image needs to be in the system memory. The fractional wavelet filter (FrWF) requires only a small RAM but has high complexity due to multiple readings of image lines. The proposed SMFrWF avoids the multiple readings of image lines, thus reducing the memory read access time and, thereby, the complexity. We evaluated the proposed SMFrWF through MATLAB simulations with 70 popular gray-scale test images of dimensions ranging from 256×256 up to 8192×8192 pixels. The results show that for images of size 2048×2048 pixels, the proposed SMFrWF (with four segments per line) has 16.8% and 53.6% lower time complexities than the conventional DWT and FrWF, respectively. The proposed SMFrWF has also been modeled in a hardware description language (HDL) and implemented on an Artix-7 field-programmable gate array (FPGA) platform to evaluate the hardware performance. We observed that the proposed SMFrWF has 65% lower energy consumption than the FrWF (both implemented on the same board). Thus, the proposed SMFrWF appears suitable for computing the wavelet transform coefficients of HR images on low-cost visual sensors and IoT platforms.

INDEX TERMS Discrete wavelet transform (DWT), low memory, low complexity, low-cost portable devices, visual sensors.

I. INTRODUCTION

A. MOTIVATION

High-resolution (HR) images generally incorporate more details of a scene or phenomenon than low-resolution images; hence, HR images are used in many applications, such as satellite imaging, medical imaging, and intelligent surveillance [1]–[5]. HR images are preferred for medical diagnosis due to their higher information content, which may be critical in many situations. The performance of pattern recognition systems can be enhanced through HR images [6].

The associate editor coordinating the review of this manuscript and approving it for publication was Tie Qiu.

Remote sensing and LANDSAT applications [7]–[9] capture images of geographical areas, whereby object recognition is easier in HR satellite images [10]. Due to the increasing demand for HR images, there is a need to develop efficient coders for HR images on low-cost visual sensors.

Most existing image coding algorithms are either based on the Discrete Cosine Transform (DCT) or the Discrete Wavelet Transform (DWT). Due to its excellent decorrelation, energy compaction, and symmetry properties, the DCT has been widely adopted in image and video coding standards, such as JPEG, MPEG-2, H.264, and HEVC [11]. The DCT is generally applied on image blocks, thereby reducing the required memory, but coding block-based DCT coefficients at very low bit-rates leads to blocking artifacts. Furthermore, the block-based DCT only exploits the correlations within the blocks, limiting the coding efficiency. In contrast, the DWT is applied on the entire image; thus, DWT based coders can achieve higher coding efficiency than DCT based coders [12]. Recently, the DWT has become very popular in image coding applications, including the JPEG 2000 coder, real-time processing, and biomedical signal processing [11].

The conventional DWT computation method requires the complete image to be kept in memory which is challenging for memory-constrained devices, such as digital cameras, personal digital assistants (PDAs) [13] and wireless videophones [14]. These low-cost mass-market consumer devices have only limited memory [12]. Moreover, there are many imaging oriented applications, such as visual sensors (sensor nodes with a small camera), with very limited on-board memory. A network of visual sensors, called visual sensor network (VSN) or wireless multimedia sensor network (WMSN), has several applications, such as visual surveillance of wild animals, automatic road traffic monitoring, and mobile multimedia [15]. The sensor nodes can capture both multimedia and non-multimedia data [16]. These nodes can also be used in IoT applications, such as smart home monitoring, smart cities, and smart health care systems [17]–[19]. However, the sensor nodes have limited RAM memory and power. For example, the available memory on many low-cost sensor nodes is only of the order of 10 kB [20]. The conventional DWT computation method requires 524 kB of RAM for a gray-scale image of size 256×256 pixels [21] and the memory requirement increases linearly with the image size. Hence, the conventional DWT computation method cannot be implemented in memory-constrained environments. Various low-memory methods to compute the DWT have been developed, such as the line-based DWT [12], the block-based DWT [22], the stripe-based DWT [23], and the Fractional Wavelet Filter (FrWF) [21]. The FrWF is the most memory efficient, requiring only 2.304 kB, 4.608 kB, and 9.216 kB memory for images of size 256×256, 512×512, and 1024×1024 pixels, respectively [20].

Although the FrWF has a low memory requirement, its memory requirement increases with the image size and therefore for HR images (images of size larger than 1024×1024), the FrWF memory requirement may exceed the memory available on typical sensor nodes (which is of the order of 10 kB). Furthermore, the FrWF achieves the low memory requirement at the expense of increased time and computational complexity, as it requires 3.25 times more multiplication operations and 2.89 times more addition operations than the conventional DWT [21]. Low-cost hand-held multimedia devices and visual sensors are constrained in terms of memory and computational capabilities as well as battery power. Therefore, from the complexity point of view, the FrWF may not be suitable for these devices. Hence, there is a need to develop a technique that can compute the DWT with low memory as well as lower computational complexity than the existing techniques.

For image transmission over bandwidth-limited communication channels, e.g., in wireless networks, the transform coefficients computed by the DWT need to be compressed (coded) using an efficient image coding algorithm. Assuming that the two operational stages (transform and coding) are performed serially, the overall memory requirement of the image coder is the maximum of the memory requirements of the two stages and the overall time complexity of the coder is the sum of the complexities of the individual stages. Among the various wavelet-based image coding algorithms, the Zero Memory Set Partitioned Embedded Block Coder (ZM-SPECK) [24], the Low Memory Block Tree Coder (LMBTC) [25], and the Wavelet Image two line coder (Wi2l) [26] satisfy the memory constraint of low-cost visual sensor nodes. Among these, ZM-SPECK [24] requires the least memory and is theoretically a memory-less coder. ZM-SPECK does not require any memory for its implementation (except for a few buffers and storage for variables), yet it needs to be combined with a suitable transform to design an image coder. Thus, the transform memory determines the overall codec memory; therefore, the transform memory needs to be reduced to design a low-cost image coder.

B. RELATED WORK

The first effort to reduce the memory for the one-dimensional (1-D) DWT was made in [27]. The various popular low-memory 2-D DWT techniques fall into three categories: line-based approaches [12], [13], [28], block-based approaches [22], [29]–[31], and stripe-based approaches [23], [32]–[36]. For the line-based DWT, only the rows necessary for computing the DWT are kept in RAM. The line-based approach implemented using the lifting scheme in [13] is faster than the conventional DWT approach but requires 12 image lines to be stored in RAM [21].

Architectures based on line based scanning techniques have been proposed in [37]–[41]. The architectures reported in [37], [38] require memory greater than $5.5N$, for an $N \times N$ dimension image, whereas the design of [39] uses 9 N storage cells. The architecture proposed in [40] needs a constant transposition memory of 4 words only, but requires a temporal memory of 5.5 N. The architecture proposed in [41] has reduced the temporal memory to 3 N and needs fewer hardware resources; however, it still requires a transposition memory of size N. Further reduction in hardware resources can be achieved at the cost of a higher number of computation cycles.

Block-based approaches apply the wavelet transform on image blocks, rather than on the complete image. Generally, the line-based and block-based DWT computation approaches require the same amount of memory [20]. DWT architectures for block-based scanning have been proposed in [42]–[44]. The architecture presented in [42] has high throughput at the expense of large temporal and transposition memories and extensive arithmetic resources. The design proposed in [43] requires large memory (of the order of $N^2 + 4N$ words as reported in [45]). Moreover, the design of [43] mainly emphasizes speed (fast computation) and does not appropriately consider the energy and area constraints [46]. The architecture proposed in [44] reduces the internal memory size, but increases the external bandwidth and computation time.

The stripe-based DWT is equivalent to the line-based DWT applied on wider "line" of blocks. The memory requirement of all these stripe-based approaches for a typical 512 $\times$ 512 gray-scale image is about 26 kB [20], which is more than the available RAM of many low-cost visual sensor nodes. Architectures for computing the DWT based on stripe-based scanning methods have been proposed in [47]–[49]. Although the architecture proposed in [47] needs no temporal memory, it requires a large line buffer and a complex control scheme. The architecture proposed in [49] has been developed for computing 1-D transform coefficients only. The modified stripe-based architecture presented in [50] has a long computation time. Furthermore, the memory requirements of other state-of-the-art DWT architectures, such as [51], [52] are more than the RAM available on most of the low-cost sensor nodes.

The lifting scheme [53] is another popular method for computing the DWT. The lifting scheme uses in-place computations, which interleave low frequency and high frequency transform coefficients to reduce the required memory. For the line-based DWT computation, the lifting scheme needs to store only 12 image lines; whereas, without lifting, 18 image lines need to be stored [21]. In order to compensate for this interleaving, the coefficients need to be reordered, which increases the complexity [13]. The execution time required by the lifting scheme is almost the same as that of the conventional DWT [13].

Overall, the lowest transform memory to date has been achieved by the line-based FrWF [20], [21]. However, even the FrWF is not suitable for transforming high-resolution (HR) images on low-cost memory-constrained platforms [54], as the available memory is less than that required for HR images. In order to further reduce the FrWF memory requirement, a Segmented FrWF (SFrWF) has been proposed [54]. The SFrWF partitions an image line into different segments and then applies the FrWF to these segments. The SFrWF needs less memory than the FrWF, but has high complexity. Therefore from the complexity point of view, the SFrWF is not suitable for low-cost visual sensor nodes.

C. CONTRIBUTION OF THIS PAPER

In this paper, we propose a novel technique to compute the DWT of images with much lower time complexity than the SFrWF. The proposed technique enables the implementation of HR image coding on low-cost resource-constrained devices, such as VSNs and IoT platforms.

The proposed Modified FrWF (MFrWF) uses the conventional DWT for computing the horizontal 1-D DWT. However, for subsequent column-wise filtering, a novel approach is proposed in which a line is read only once, and the line is filtered partially and values are stored and subsequently updated. The novelty of the proposed MFrWF is that the column-wise filtering is performed in such a way that a line needs to be accessed only once and required multiplications with corresponding filter coefficients are performed and results are stored in buffers. The difference between the FrWF and the proposed MFrWF is that the FrWF reads an image line multiple times because an image line is part of multiple vertical filter areas as explained in Sections II-B and III-A. The MFrWF avoids the multiple readings of an image line by using three different processes with intermediate buffers, as explained in detail in Section III-B. Experimental results demonstrate that the proposed MFrWF significantly reduces the time complexity (typically to a half or a third) compared to the FrWF.

Due to the additional storage of intermediate values, the MFrWF memory requirement is slightly increased compared to the FrWF. The MFrWF memory requirement can be reduced by the proposed SMFrWF, which partitions an image line into multiple segments. Only one segment is read at a time into RAM and the subband coefficients of the segment are computed using the MFrWF procedure. The SMFrWF requires somewhat larger memory than the corresponding SFrWF, but the SMFrWF does enable the transformation of HR images of size 4096 $\times$ 4096 pixels on nodes with 10 kB of RAM. The SMFrWF time complexity for such HR image transformations is less than half of the corresponding SFrWF time complexity. To the best of our knowledge, the proposed SMFrWF is the first attempt to develop an alternative to the FrWF that reduces the computation complexity for HR image wavelet transforms on low-memory portable multimedia devices.

The rest of the paper is organized as follows. Section II gives brief overviews of the conventional DWT and FrWF. Section III introduces the proposed MFrWF and SMFrWF along with the pseudo-code as well as memory and complexity analyses. Section IV presents evaluation results for memory, time complexity, and quality of the reconstructed image, as well as a hardware implementation. Section V concludes the paper and outlines future research directions.

II. BACKGROUND

In this section, we briefly review the conventional DWT and FrWF schemes for computing the wavelet transform coefficients of an image. Throughout, we assume, as is common in low-memory image transform and coding studies, that the original image and its transformed coefficients are stored in an external secure digital (SD) card. Data is read from the SD card into the system's limited RAM memory as and when needed.

A. OVERVIEW OF CONVENTIONAL 2-D DWT

The conventional 2-D DWT approach first filters (convolves) all the rows of an image by a low pass filter (LPF) and a high pass filter (HPF), followed by downsampling by a factor of two, resulting in two subbands, namely the approximation

TABLE 1. Summary of main notations.

N	Image size, $N \times N$ pixels
Q	Number of segments per line
s	Input buffer for line (dimension $1 \times N$) or line segment (dimension $1 \times N/Q$)
n_l	Number of low pass filter coefficients
n_h	Number of high pass filter coefficients
r, w	Time required for a read operation, a write operation
a, m	Time required for an addition, a multiplication
c	Time required for a comparison (incl. modulo-4 op.)
lev	Wavelet transform level, $lev \geq 1$

subband L, and the detail subband H. The process is then repeated on all the columns of subbands L and H, resulting in the LL, LH, HL, and HH subbands. The downsampling combined with the convolution operations with the LPF and HPF (for symmetric filters) can be expressed as [20]:

$$L(i) = \sum_{j=-\lfloor \frac{n_l}{2} \rfloor}^{j=\lfloor \frac{n_l}{2} \rfloor} x_{2i+j} l_j \qquad i = 0, 1, \ldots, \frac{N}{2} - 1 \tag{1}$$

$$H(i) = \sum_{j=-\lfloor \frac{n_h}{2} \rfloor}^{j=\lfloor \frac{n_h}{2} \rfloor} x_{2i+j+1} h_j \quad i = 0, 1, \ldots, \frac{N}{2} - 1, \tag{2}$$

where l_j and h_j denote coefficient j of the LPF and HPF, respectively, x_{2i+j} denotes the $(2i + j)^{th}$ sample of the signal, and n_l and n_h are the lengths of the LPF and HPF, respectively. Table 1 summarizes the main notations used in this article. The symbol $\lfloor x \rfloor$ denotes the largest integer less than or equal to x. $L(i)$ and $H(i)$ are the i^{th} coefficient of subbands L and H, respectively. The LPF center is aligned with the odd signal values, i.e., $x_0, x_2, \ldots, x_{\frac{N}{2}-2}$, and the HPF center is aligned with the even signal values, i.e., $x_1, x_3, \cdots, x_{\frac{N}{2}-1}$. Border discontinuities are avoided by symmetrically extending $\lfloor \frac{n_l}{2} \rfloor$ samples on both sides. If the original image is of dimension $N \times N$, the resulting subbands LL, LH, HL, and HH are of dimension $\frac{N}{2} \times \frac{N}{2}$ each. For a gray-scale image (8 bits per pixel) of dimension $N \times N$, the conventional DWT computation method requires the entire image to be kept in RAM. The memory requirement is $4N^2$ bytes for floating point filter values, see Appendix A.

B. FrWF OVERVIEW

The FrWF uses three different buffers, namely **s**, LL_HL, and LH_HH, each of dimension $1 \times N$, where N is the number of pixels in a row (line). Only one pixel row of the image is read into buffer **s** at a time. The 1-D DWT of this line is calculated with the row-wise conventional 1-D DWT approach, while the column-wise DWT coefficients are computed through consecutive updates. In the FrWF, a particular image area, called vertical filter area (VFA), is selected at a time. The number of rows in the VFA is equal to the LPF length. The VFA is slided vertically by two lines to achieve vertical downsampling. The filtered and downsampled rows in a VFA are multiplied by filter coefficients and then added together to obtain the final subbands [20], see Appendix B for the FrWF computational complexity analysis. The FrWF memory requirement (in bytes) for gray-scale-images using floating point arithmetic for lev wavelet transform levels is

$$Mem_{\text{FrWF\_float}} = \begin{cases} 9N, & lev = 1 \\ \dfrac{12N}{2^{lev-1}}, & lev \geq 2. \end{cases} \tag{3}$$

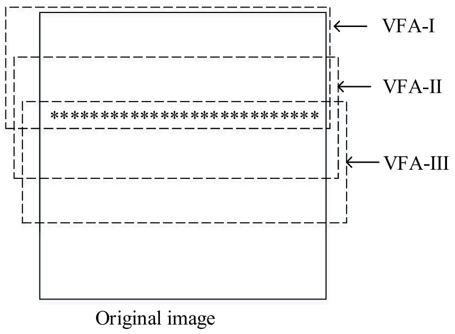


FIGURE 1. Illustration of an image line being part of multiple vertical filter areas (VFAs).

One of the problems in the FrWF is that a line is typically part of multiple VFAs (since the VFA is shifted vertically by 2 lines) as depicted in Fig. 1. In the figure, dashed rectangles labeled as VFA-I, VFA-II, and VFA-III depict three consecutive VFAs. Clearly, to implement vertical filtering in the FrWF, an image line needs to be read multiple times, which increases the computation time. As the filter length increases, the number of lines in the VFA increases and thus the number of repetitive readings of a line from the SD card increases, further increasing the FrWF computation time. Although the FrWF is memory efficient, it is very time-consuming and hence requires fast processors. Therefore, the challenge is to modify the FrWF such that it may be implemented on low-cost microcontrollers (which are widely used in VSNs and the IoT) to compute the DWT of HR images.

III. PROPOSED MODIFIED FRACTIONAL WAVELET FILTER

The proposed Modified Fractional Wavelet Filter (MFrWF) algorithm is based on the FrWF, but modified to reduce the computational complexity. Block-diagrams highlighting the key difference between the FrWF and the modified FrWF (MFrWF) are shown in Fig. 2. As depicted in Fig. 2,

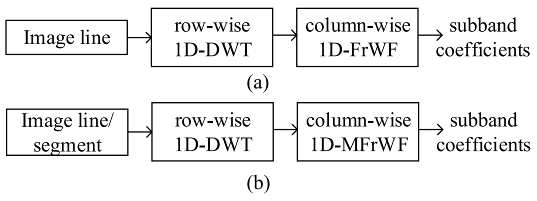


FIGURE 2. Block diagram of steps in (a) FrWF and (b) proposed MFrWF.

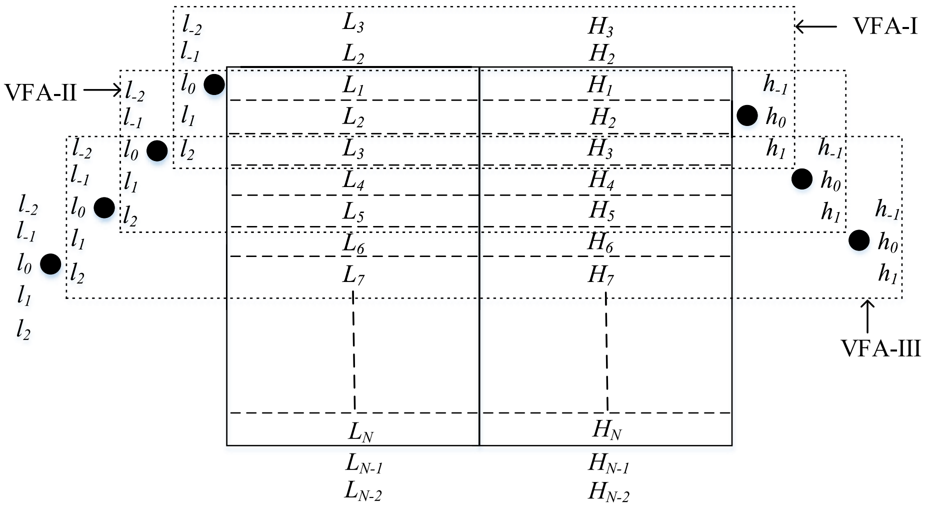

VFA-I
VFA-II
VFA-III
L_1 … L_7, L_N, L_{N-1}, L_{N-2}, L_3, L_2
H_1 … H_7, H_N, H_{N-1}, H_{N-2}, H_3, H_2
l_{-2} l_{-1} l_0 l_1 l_2
h_{-1} h_0 h_1

FIGURE 3. Position of filter coefficients for computing *LL* and *HH* subbands in different VFAs for 5/3 filter bank.

the difference lies in the way the column-wise filtering is performed. The proposed MFrWF avoids multiple readings of a line belonging to different VFAs while computing the vertical filtering, thereby reducing the computation time. This process requires some additional memory, which can be compensated for by applying the algorithm on segmented lines, rather than on the whole line. This section presents the details of the MFrWF method as well as its memory requirements and complexity characteristics.

A. PROBLEM FORMULATION

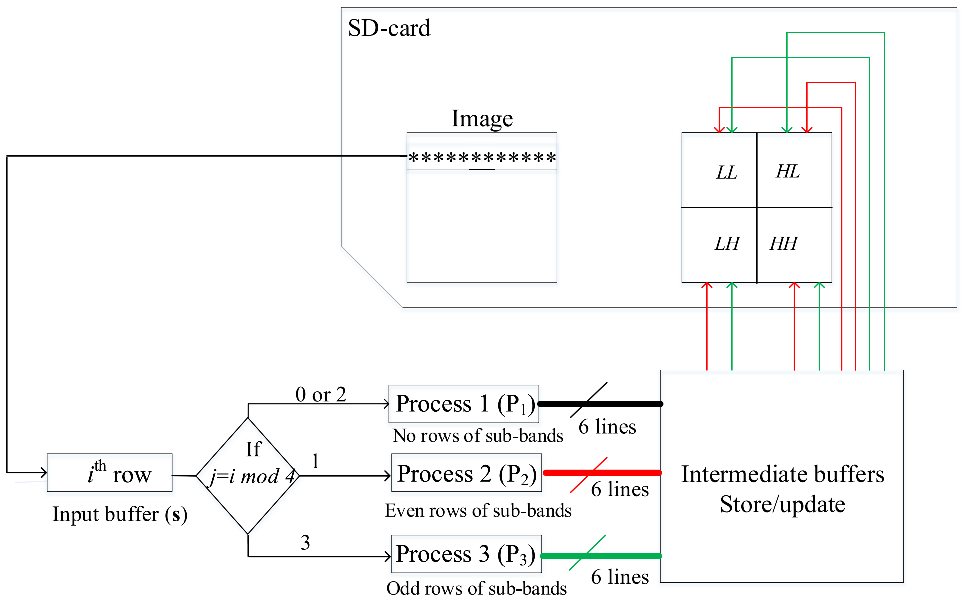
In order to facilitate the understanding of the proposed MFrWF method, it is helpful to analyze the problems in the column-wise filtering in the FrWF in detail. Let us assume that an image (of size $N \times N$) has already undergone the row-wise (horizontal) low-pass and high-pass filtering using 5/3 filter banks, resulting in the L and H subbands (each of size $\frac{N}{2} \times N$), respectively, as depicted in Fig. 3. (Other filters can be accommodated in analogous fashion.) In Fig. 3, L_i and H_i represent row i of the L and H subbands, respectively. The rows with index $i < 0$ or $i > N$ are the symmetrically extended rows beyond the image boundaries. To apply column-wise (vertical) low-pass and high-pass filtering, the columns of subbands L and H (within a VFA) need to be convolved with the impulse responses of the corresponding filters. The black dots in Fig. 3 represent the positions of the central LPF and HPF coefficient (l_0 and h_0), respectively. In the FrWF, only one line is read at a time and all its elements are multiplied by the filter coefficients shown (for subbands LL and HH only) adjacent to the corresponding rows in Fig. 3 and the results are stored in buffers. The process is repeated for each row within a VFA and the previous results are subsequently updated. The completion of this process for a VFA results in one row of each of the LL, LH, HL, and HH subbands. The VFA is then shifted by two lines, to achieve vertical downsampling by a factor of two.

As is evident from Fig. 3, for computing the LL subband coefficients, all elements of odd indexed rows of the L subband (except L_1) are multiplied by l_{-2}, l_0, and l_2; whereas, all elements of even indexed rows of the L subband are multiplied by l_1 and l_{-1}. Likewise, the HH subband coefficients are obtained by multiplying the elements of the H subband either with h_0 (for even indexed rows) or with h_1 and h_{-1} (for odd indexed rows), as shown in Fig. 3. Similarly, the LH and HL subband coefficients are computed by multiplying the rows of subbands L and H with the HPF and LPF coefficients, respectively. The mathematical justifications of this FrWF process is given in Appendix B. In order to implement the vertical filtering in FrWF, a row is read as many times as the number of different filter coefficients it has to be multiplied with. Furthermore, since each VFA is processed independently, a line (row) is read at least once in each VFA. However, since a row may be part of more than one VFA, the FrWF requires multiple readings of lines. Thus, the FrWF consumes a lot of computation time due to multiple readings of lines (which increase the memory read/access time) while implementing vertical filtering. Reducing the computation time due to multiple readings of lines during vertical filtering is the main contribution of this paper.

B. MODIFIED FrWF (MFrWF)

1) MFrWF OVERVIEW

In order to reduce the FrWF complexity, we modify the process of column-wise filtering in the FrWF so as to obtain the proposed MFrWF. Fig. 4 shows the schematic diagram of the proposed MFrWF. The MFrWF implementation requires eight buffers (for a 5/3 filter): an input buffer **s** of dimension $1 \times N$ (to store a line of an image of size $N \times N$); a buffer $b_{L/H}$ of dimension $1 \times \frac{N}{2}$ (to store the coefficients of subbands L or H), and six intermediate buffers, namely b_{1_ll}, b_{2_ll}, b_{lh}, b_{1_hl}, b_{2_hl} and b_{hh}; each of dimension $1 \times \frac{N}{2}$ (to store newly



Red lines denote the output of Process 2 from which even rows of subbands are computed.
Green lines denote the output of Process 3 from which odd rows of subbands are computed.

FIGURE 4. **Schematic diagram illustrating the modified FrWF (MFrWF) technique for a 5/3 filter.**

TABLE 2. **Filter coefficients and buffers to be used in different processes to compute DWT using 5/3 filter. The $*$ symbol indicates that after the corresponding subband coefficients have been computed and saved in the external SD card, the buffer is reset.**

Process	filter coefficients					buffers				
	f_1	f_2	f_3	f_4	f_5	b_1	b_2	b_3	b_4	b_5
$P_1(L)$	l_{-1}	l_1	h_0	–	–	b_{1_ll}	b_{2_ll}	b_{lh}	–	–
$P_1(H)$	l_{-1}	l_1	h_0	–	–	b_{1_hl}	b_{2_hl}	b_{hh}	–	–
$P_2(L)$	l_0	l_2	h_1	l_{-2}	h_{-1}	b_{1_ll}	$b_{2\_ll}*$	$b_{lh}*$	b_{2_ll}	b_{lh}
$P_2(H)$	l_0	l_2	h_1	l_{-2}	h_{-1}	b_{1_hl}	$b_{2\_hl}*$	$b_{hh}*$	b_{2_hl}	b_{hh}
$P_3(L)$	l_0	l_2	h_1	l_{-2}	h_{-1}	b_{2_ll}	$b_{1\_ll}*$	$b_{lh}*$	b_{1_ll}	b_{lh}
$P_3(H)$	l_0	l_2	h_1	l_{-2}	h_{-1}	b_{2_hl}	$b_{1\_hl}*$	$b_{hh}*$	b_{1_hl}	b_{hh}

generated or updated intermediate coefficients of subbands LL, LH, HL, and HH).

The MFrWF algorithm works as follows. It reads one row (say row i) of the image at a time into buffer **s**. Depending on the row index i, it calls one of three processes, namely Process 1 (P_1), Process 2 (P_2), or Process 3 (P_3), to perform the 2-D filtering. For even indexed rows, Process 1 (P_1) is called; whereas, for odd-indexed rows, the process is selected as follows: If the value of $j = i \mod 4$ (result of modulo-4 operation on line index i) is 1, then Process 2 (P_2) is selected; on the other hand, if the value of $j = i \mod 4$ is 3, then Process 3 (P_3) is selected. Processes 2 and 3 compute the even and odd rows of the subbands, respectively, as shown in Fig. 4.

2) MFrWF P1, P2, AND P3 PROCESSES

Each of the three processes consists of two stages. In the first stage (equivalent to horizontal filtering), the row stored in buffer **s** is low/high-pass filtered (conventional convolution) and the values, after down-sampling by 2, are stored in buffer $b_{L/H}$. In the second stage (equivalent to the vertical filtering), the contents of buffer $b_{L/H}$ are multiplied with low/high-pass filter coefficients and stored (after update, if required) in the corresponding intermediate buffers, as shown in Fig. 5(a) for Process 1 and Fig. 5(b) for Processes 2 and 3. In this figure, f_1, f_2, f_3, f_4, and f_5 are filter coefficients, while b_1, b_2, b_3, b_4, and b_5 are buffers. The filter coefficients and the buffers to be used in the different processes are listed in Table 2.

We note that in the first stage, input lines are first low-pass filtered and sent to the second stage through buffer $b_{L/H}$. After completing the second stage, the input line from buffer **s** is high-pass filtered in the first stage and processed in the second stage. In this way, a single buffer $b_{L/H}$ is used to store the output of both low and high-pass filters in the first stage. Furthermore, the filter coefficients used for the multiplications in the second stage in a process remain the same, irrespective of the type of filtering in the first stage; however, the intermediate buffers used are different as

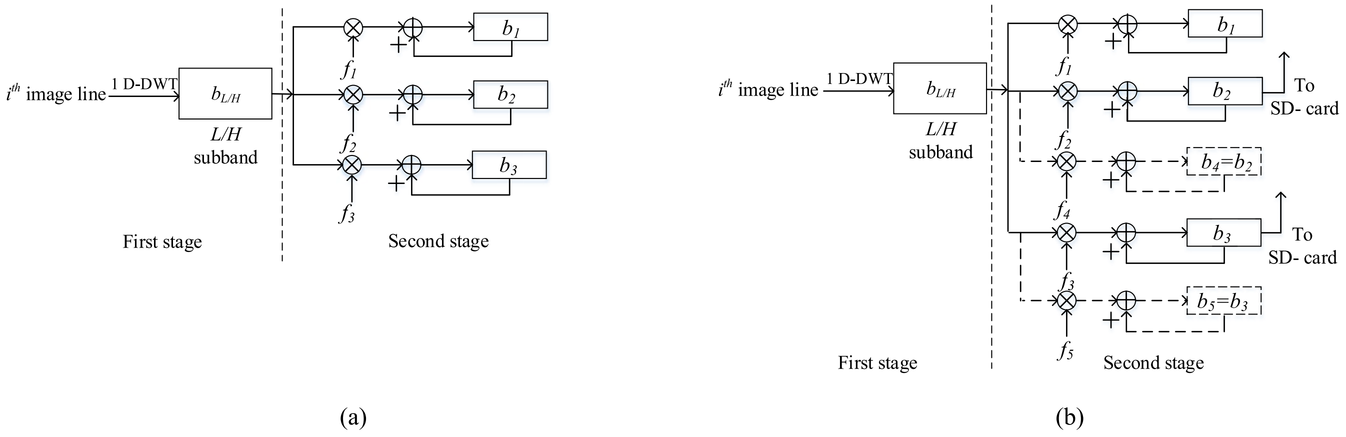


FIGURE 5. Schematic process representation showing filter coefficients to be multiplied with and buffers to be saved in for (a) Process 1, and (b) Processes 2 and 3.

evident from Table 2. This is because the low-pass and high-pass filtering outputs in the first stage contribute to different subbands in the second stage. Since two different data (one at a time) are being processed in the second stage, each of the processes can be divided into two subprocesses, namely $\{P_q(L)$ and $P_q(H), q = 1, 2, 3\}$, corresponding to low-pass and high-pass filtering in the first stage, respectively.

The three processes operate similarly and differ only in the second stage filter coefficients and buffers, as depicted in Table 2. We describe the operation of the processes as follows. Consider process P_1 for an even-indexed line i that is currently in buffer **s**: The line is first processed through $P_1(L)$, which convolves the image line with an LPF using Eqn. (1) (which incorporates downsampling) and stores the result in buffer $b_{L/H}$. The coefficients in buffer $b_{L/H}$ are then multiplied by the filter coefficients l_{-1}, l_1, and h_0 and stored (after adding to the previous values) in buffers b_{1_ll}, b_{2_ll}, and b_{lh}, respectively. The two stages are repeated for process $P_1(H)$, except that in the first stage the elements of buffer **s** are now convolved with an HPF, and the results of the multiplications with the same set of filter coefficients in the second stage are added to the previous values of buffers b_{1_hl}, b_{2_hl}, and b_{hh}, respectively. This completes the processing of one even-indexed image line. We note that the values in these buffers will be used and updated in Processes 2 and 3, to compute the final subband values.

The next (odd-indexed) image line is read into buffer **s**, and either process P_2 or P_3 is applied depending on the line index. Suppose, the algorithm selects process P_2 for this line: the line in buffer **s** is first convolved by a LPF (corresponding to process $P_2(L)$) in the first stage and the result is stored in buffer $b_{L/H}$. In the second stage of the process, the elements of buffer $b_{L/H}$ are multiplied by l_0, l_2, and h_1 and added/stored to the previous values in buffers b_{1_ll}, b_{2_ll}, and b_{lh}, respectively. Whenever a line is multiplied by l_2 and h_1 in the second stage, it means that the processing of all rows in a VFA is complete and the contents of buffers b_{2_ll} and b_{lh}, which correspond to even lines of subbands LL and LH, respectively, are flushed to the SD card and are reset. In the proposed algorithm, in order to reduce the additional memory, these two buffers b_{2_ll} and b_{lh} can be reused to store the intermediate results of multiplication of the contents of buffer $b_{L/H}$ with other filter coefficients within the same process.

After resetting the contents of the two flushed buffers b_{2_ll} and b_{lh}, the elements of buffer $b_{L/H}$ are multiplied by the remaining coefficients, i.e., l_{-2} and h_{-1}, and the results are stored in buffers b_{2_ll} and b_{lh}, respectively, as depicted in Table 2, $P_2(L)$ process rows. This completes process $P_2(L)$. We emphasize that in Table 2, buffers b_2 and b_3 are reused as buffers b_4 and b_5 after flushing their contents to the SD card. The dashed boxes and lines in Fig. 5(b) show how buffers b_2 and b_3 can be reused (after flushing their previous contents to the SD card and resetting). After completing process $P_2(L)$, process $P_2(H)$ is applied on the same line stored in buffer **s**. Process $P_2(H)$ is similar to process $P_2(L)$, except that in the first stage, the image line is convolved with the HPF and after multiplication in the second stage, the results are stored/updated in a different set of buffers, as shown in Table 2. Process $P_2(H)$ computes the even lines of subbands HL and HH.

Process 3 (P_3) operates similarly to process 2 (P_2), with only a minor difference in the second stage, in that the roles of b_{1_ll} and b_{1_hl} are interchanged with b_{2_ll} and b_{2_hl}, respectively, as shown in Table 2. Sub-process $P_3(L)$ of process P_3 computes the odd rows of subbands LL and LH; whereas, sub-process $P_3(H)$ computes the odd rows of subbands HL and HH. In this way, processes P_2 and P_3 alternately compute the even and odd rows of four wavelet subbands, and allow the 2-D DWT of an image to be computed by transferring only one image row to the processor RAM at a time and reading every line only once.

We note that the proposed MFrWF computes the 2-D DWT (with a 5/3 filter bank) of an image by reading each image line

Algorithm 1 SMFrWF Steps

```
1: Initialize buffers: s, $b_{L/H}$, $b_{1\_ll}$, $b_{2\_ll}$, $b_{lh}$, $b_{1\_hl}$, $b_{2\_hl}$, and $b_{hh}$
2: for $q = 1, 2, \ldots, Q$
3:   for $i = 1, 2, \ldots, N$ Read segment $q$ of line $i$ to buffer s
4:    $j = i \mod 4$ (modulo-4operation)
5:     if $j = 0$ or 2: call $Process1$ ($P_1$);
6:      if $j = 1$: call $Process2$ ($P_2$);
7:       if $j = 3$: call $Process3$ ($P_3$);
8:   end for
9: end for
10: Function $Process1$ ($P_1$):
    {$b_{L/H} = conv(\mathbf{s}, l)$
    $(b_{1\_ll}, b_{2\_ll}, b_{lh}) = update(l_{-1}, l_1, h_0; b_{L/H})$
    $b_{L/H} = conv(\mathbf{s}, h)$
    $(b_{1\_hl}, b_{2\_hl}, b_{hh}) = update(l_{-1}, l_1, h_0; b_{L/H})$}
11: Function $Process2$ ($P_2$) :
    {$b_{L/H} = conv(\mathbf{s}, l)$
    $(b_{1\_ll}, b_{2\_ll}, b_{lh}) = update(l_0, l_2, h_1; b_{L/H})$
    $(LL(\frac{i-1}{2}), LH(\frac{i-1}{2})) = store(b_{2\_ll}, b_{lh})$
    $(b_{2\_ll}, b_{lh}) = update(l_{-2}, h_{-1}; b_{L/H})$
    $b_{L/H} = conv(\mathbf{s}, h)$
    $(b_{1\_hl}, b_{2\_hl}, b_{hh}) = update(l_0, l_2, h_1; b_{L/H})$
    $(HL(\frac{i-1}{2}), HH(\frac{i-1}{2})) = store(b_{2\_hl}, b_{hh})$
    $(b_{2\_hl}, b_{hh}) = update(l_{-2}, h_{-1}; b_{L/H})$}
12: Function $Process3$ ($P_3$):
    {$b_{L/H} = conv(\mathbf{s}, l)$
    $(b_{1\_ll}, b_{2\_ll}, b_{lh}) = update(l_2, l_0, h_1; b_{L/H})$
    $(LL(\frac{i-1}{2}), LH(\frac{i-1}{2})) = store(b_{1\_ll}, b_{lh})$
    $(b_{1\_ll}, b_{lh}) = update(l_{-2}, h_{-1}; b_{L/H})$
    $b_{L/H} = conv(\mathbf{s}, h)$
    $(b_{1\_hl}, b_{2\_hl}, b_{hh}) = update(l_2, l_0, h_1; b_{L/H})$
    $(HL(\frac{i-1}{2}), HH(\frac{i-1}{2})) = store(b_{1\_hl}, b_{hh})$
    $(b_{1\_hl}, b_{hh}) = update(l_{-2}, h_{-1}; b_{L/H})$}
13: Function $update$:
        $(b_1, b_2, b_3) = update\ (f_1, f_2, f_3; b_{L/H})$;
    {$b_1 += f_1 * b_{L/H}$
    $b_2 += f_2 * b_{L/H}$
    $b_3 += f_3 * b_{L/H}$ }
14: Function $store$:
        $(SB_1(i), SB_2(i)) = store\ (b_1, b_2)$;
    { $SB_1(i) = b_1$; Store values in buffer $b_1$ as row $i$ of subband $SB1$ in SD card and reset buffer $b_1$.
    $SB_2(i) = b_2$; Store values in buffer $b_2$ as row $i$ of subband $SB2$ in SD card and reset buffer $b_2$.}
```

only once and performs all required operations using eight buffers (one with N elements, and the remaining seven with $N/2$ elements each). On the other hand, the FrWF computes the 2-D DWT by reading every odd image row three times and every even row twice and uses three buffers (each of size N elements). The memory access time and hence the total computation time of the proposed MFrWF is expected to be substantially reduced compared to the FrWF; however, the MFrWF has a slightly higher memory requirement.

3) SEGMENTED MFrWF (SMFrWF)

In case only lower memory is available, we propose to apply the MFrWF algorithm on line segments, rather than on the entire line: an image line is first partitioned into multiple segments and then the MFrWF is independently applied on these segments in the order depicted in Fig. 6. The results obtained for each line segment are concatenated to obtain the rows of the subbands. We refer to the MFrWF combined with line segmentation as Segmented Modified FrWF (SMFrWF). The memory requirement of the SMFrWF decreases as the number of line segments increases, at the expense of increased computation time. Therefore, the number of segments can be selected to trade off memory requirement and computational complexity, according to the available resources and intended application, as analyzed in detail in Appendix C.

For clarity of exposition, we have explained the proposed MFrWF for computing one-level of the 2-D DWT with 5/3 filter banks; however, the proposed MFrWF is generic and can be applied for any filter bank (including 9/7 filter banks) and

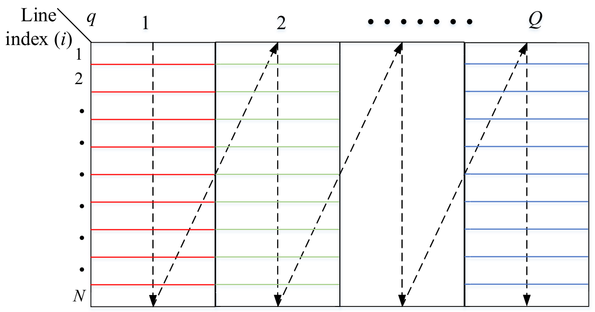


FIGURE 6. Diagram showing the order in which different segments are read.

for any level of the wavelet decomposition. The number of intermediate buffers will depend on the length of the longest filter in the filter bank.

C. SMFrWF PSEUDO-CODE

This section presents the pseudo-code of the proposed SMFrWF algorithm for 5/3 filters. The algorithm for computing the first level of the 2-D wavelet decomposition of an image is summarized in the pseudo-code in Algorithm 1. Note that $Q = 1$ corresponds to the MFrWF without line segmentation. The same pseudo-code can be extended to compute higher decomposition levels by considering subband LL as input. The functions $conv(\mathbf{s}, l)$ and $conv(\mathbf{s}, h)$ denote the convolution of elements of buffer $\mathbf{s}$ with the LPF (l) and HPF (h), respectively. The algorithm processes segment q of each image line before processing segment $q+1$, as shown in

Fig. 6. The vertical scanning of segments reduces the size of each buffer by a factor of Q (the number of segments in each line).

D. MFrWF COMPUTATIONAL COMPLEXITY

This section summarizes the evaluation (for details see Appendix D) of the time required to compute the 2-D DWT with the proposed MFrWF (with and without line segmentation) and compares with the conventional DWT, FrWF, and segmented FrWF (SFrWF). The total time consumed (C_{Total}) for computing the 2-D DWT is the sum of the time consumed for accessing (read/write) the SD card data ($C_{\text{SD\_Access}}$) and the time required to perform arithmetic operations (C_{Arithm}).

The SD card access time $C_{\text{SD\_Access}}$ depends on the number of read and write operations, whereas the computation time C_{Arithm} depends on the number of additions, multiplications, and comparisons. The time required for memory access in the MFrWF is:

$$C_{\text{SD\_Access\_MFrWF}} = N^2 r + N^2 w, \tag{4}$$

where r and w denote the time required to read and write one pixel, respectively. The numbers of read and write operations required for computing the 2-D DWT of images using various methods are listed in Table 3. We note from Table 3 that the number of write operations of the conventional DWT is $2N^2$ (as explained in Appendix A), whereas the other methods need only N^2 write operations. The numbers of read operations in the proposed MFrWF and SMFrWF are less than in the conventional DWT, FrWF, and SFrWF. In fact, the numbers of read operations in the FrWF and SFrWF are the highest among the considered DWT computation approaches. The numbers of read operations in the MFrWF and SMFrWF are less than in the conventional DWT, despite the fact that the number of read operations in the SMFrWF increases with increasing Q (see Appendix E).

As detailed in Appendix D, the MFrWF computation time is

$$C_{\text{Arithm\_MFrWF}} = N^2\,[2(n_l-2)a + n_l m] + Nc, \tag{5}$$

where a, m, and c denote the time required to perform one addition, one multiplication, and one comparison (including the modulo-4), respectively, and n_l is the LPF length. The N comparisons (one for each image line) are required to decide which process is to be performed on the current line. From Eqns. (4) and (5) for the MFrWF in comparison to the time complexities in Table 3 for the conventional DWT and FrWF, we observe that the MFrWF substantially reduces the number of read operations (by a factor of $n_l/2$ compared to the FrWF). However, the MFrWF slightly increases the computation time by increasing the number of operations (one comparison operation per line). However, we note that saving read operations (which depend on the number of pixels N) is more economical than increasing the number of arithmetic operations (comparison operation performed only once for each line). Therefore, due to the greatly reduced number of read operations, the MFrWF has lower computational complexity compared to the conventional DWT and FrWF, as numerically verified in Section IV-B.

In summary, we observe from Table 3 that the conventional DWT requires $2N^2$ write operations and the other DWT computation methods require N^2 write operations. The numbers of read operations of the various DWT computation methods differ, with the FrWF and SFrWF requiring by far the most read operations. The MFrWF and SMFrWF require the same numbers of additions and multiplications as the conventional DWT, but less than the FrWF. The SMFrWF has relatively higher complexity than the MFrWF due to the increased number of read operations.

E. MEMORY REQUIREMENT

The memory requirement of the proposed MFrWF can be evaluated by considering an image of size $N \times N$ and the sizes of the various buffers used in the algorithm. The MFrWF uses buffer $\mathbf{s}$ of dimension $1 \times N$ for temporary storage of one image line consisting of N pixels, each of size one byte (unsigned integer). After the first stage of the row-wise convolution and downsampling, the real-valued filtered coefficients are stored in buffer $b_{L/H}$, with size $1 \times N/2$. Finally, in the second stage, computing the coefficients of each of the LL and HL subbands requires $\frac{n_l-1}{2}$ intermediate buffers, while computing the coefficients of each of the LH and HH subbands requires $\frac{n_h-1}{2}$ intermediate buffers. These buffers are of size $1 \times N/2$ elements and each element has four bytes to store floating-point coefficients. For computing higher DWT levels, the filtering and downsampling operations are performed on the LL subband, and the coefficients in the input buffer $\mathbf{s}$ are also of floating type. For each higher DWT level, the sizes of the buffers are reduced by a factor of two. Therefore, the total MFrWF memory requirement for computing one and multiple DWT levels is given in terms of the LPF filter length n_l (whereby $n_h = n_l - 2$ for most bi-orthogonal filters) and the number lev of wavelet decomposition levels as

$$Mem_{\text{MFrWF}} = \begin{cases} N(4n_l-5)\ \text{bytes}, & lev = 1 \\ \dfrac{2N(2n_l-1)}{2^{lev-1}}\ \text{bytes}, & lev \geq 2. \end{cases} \tag{6}$$

For the segmented MFrWF (SMFrWF), an input image line is partitioned into Q segments, with $\frac{N}{Q} + 2\lfloor\frac{n_l}{2}\rfloor$ coefficients each, see Appendix C. Thus, the size of the input buffer $\mathbf{s}$ is $1 \times (\frac{N}{Q} + 2\lfloor\frac{n_l}{2}\rfloor)$ (each coefficient has one byte for the first transform level). The dimensions of the other buffers are $1 \times \frac{N}{2Q}$ coefficients of four bytes each. Thus, the SMFrWF memory requirements are

$$Mem_{\text{SMFrWF}} = \begin{cases} \dfrac{N(4n_l-5)}{Q} + 2\left\lfloor\dfrac{n_l}{2}\right\rfloor \text{bytes}, & lev = 1 \\ \dfrac{2N(2n_l-1)}{Q2^{lev-1}} + 2\left\lfloor\dfrac{n_l}{2}\right\rfloor \text{bytes}, & lev \geq 2. \end{cases} \tag{7}$$

We note that the overall memory requirement for computing the DWT coefficients for more than one level is equivalent to the memory requirement of the first level, since the same buffers may be reused.

TABLE 3. Comparison of computational complexity (in terms of number of read, write, add, multiplication, and comparison operations) and memory requirements (in bytes) of different 2-D DWT computation methods, as derived in Appendices A–E. Parameters: Image dimension $N \times N$ pixels; length of LPF n_l (HPF length $n_h = n_l - 2$ for typical bi-orthogonal filters); Q line segments.

	Number of operations					
Method	read	write	add	multipl.	comp.	Memory (bytes)
Conv. DWT	$2N^2$	$2N^2$	$2N^2(n_l-2)$	$N^2 n_l$	–	$4N^2$
FrWF	$\frac{n_l N^2}{2}$	N^2	$\frac{N^2}{2}(n_l^2-4)$	$\frac{N^2}{4}(n_l^2+4n_l-4)$	–	$9N$
SFrWF	$\frac{n_l N^2}{2}$	N^2	$\frac{N^2}{2}(n_l^2-4)$ $+n_l N(Q-1)(n_l-2)$	$\frac{N^2}{4}(n_l^2+4n_l-4)$	–	$\frac{9N}{Q}+12(n_l-1)$
MFrWF	N^2	N^2	$2N^2(n_l-2)$	$N^2 n_l$	N	$N(4n_l-5)$
SMFrWF	$N^2+2N(Q-1)\lfloor\frac{n_l}{2}\rfloor$	N^2	$2N^2(n_l-2)$	$N^2 n_l$	N	$\frac{N(4n_l-5)}{Q}+2\lfloor\frac{n_l}{2}\rfloor$

TABLE 4. Memory required in kBytes for computing wavelet transform using various DWT computation methods for 5/3 filter for images of different resolutions (sizes); the number in brackets is the number Q of line segments.

Memory (kB)	Image size N ($N \times N$ pixels)							
Method	256	512	768	1024	2048	2304	4096	8192
Conv. DWT	262.144	1048.576	2359.296	4194.304	18874.368	21233.664	67108.864	268435.456
FrWF	2.304	4.608	6.912	9.216	18.432	20.736	36.804	73.728
SFrWF (2)	1.200	2.352	3.504	4.656	9.264	10.416	18.480	36.912
SFrWF (4)	0.624	1.200	1.776	2.352	4.656	5.232	9.264	18.480
SFrWF (8)	0.336	0.624	0.912	1.200	2.352	2.640	4.656	9.264
MFrWF	3.840	7.680	11.520	15.360	30.720	34.56	61.440	122.880
SMFrWF (2)	1.924	3.844	5.764	7.684	15.364	17.284	30.724	61.444
SMFrWF (4)	0.964	1.924	2.884	3.844	7.684	8.644	15.364	30.724
SMFrWF (8)	0.484	0.964	1.444	1.924	3.844	4.324	7.684	15.364

IV. RESULTS AND DISCUSSION

This section evaluates and compares the memory requirement and computational complexity of the proposed MFrWF (with and without line-segmentation) with the conventional DWT and FrWF (with and without line-segmentation). We consider 70 standard gray-scale test images of dimensions ranging from $N \times N = 256 \times 256$ to 8192×8192 (eight different dimensions); specifically ten images each of the dimensions 256×256, 512×512, 768×768, 1024×1024, 2048×2048, and 2304×2304, as well as five images each of dimensions 4096×4096 and 8192×8192. The results presented are the averages of 25 independent observations for each image. We also demonstrate the compatibility of the proposed MFrWF and SMFrWF with state-of-the-art wavelet based image coding algorithms and evaluate the coding efficiency in terms of the peak-signal-to-noise-ratio (PSNR). We implemented all the transforms and the coding algorithms using MATLAB2018 on a Pentium I3 computer system equipped with a 2.4 GHz processor and 4 GB RAM. Furthermore, we implemented the architecture of the proposed MFrWF and the state-of-the-art FrWF architecture [55] on a field-programmable gate array (FPGA), namely the Xilinx Artix-7 board, and compared their hardware performance metrics.

A. MEMORY REQUIREMENT

The memory required for computing five levels of the wavelet transform (with 5/3 filter banks) using the conventional DWT, FrWF, SFrWF, MFrWF, and SMFrWF for gray-scale images of various sizes are shown in Table 4. We observe from Table 4 that the memory requirement for computing the DWT coefficients increases in general with the image size. Table 4 also shows that the memory requirement of the conventional DWT is the highest among all considered methods as it requires the whole image to be stored in the system memory. The other techniques require only one image line or its segments to be stored in the system memory. However, the memory requirements of the FrWF and MFrWF differ due to the numbers and sizes of the buffers used to implement these algorithms. We observe from Table 4 that the MFrWF requires slightly higher memory sizes than the FrWF. The slightly higher MFrWF memory requirement is due to the increased number of buffers in its implementation. However, applying the MFrWF on segments of an image line (as in SMFrWF), reduces the memory requirement as the number of segments increases. This is because of the reduced SMFrWF buffer sizes. Nevertheless, we observe from Table 4 that the segmented FrWF (SFrWF) requires less memory than the segmented MFrWF (SMFrWF) for the same number of image line partitions.

Table 4 reveals that the segmented FrWF or MFrWF can be implemented on low-cost sensor nodes with up to 10 kB of on-board memory [20] and can transform even HR images. For example, for an image of size 2048×2048, the SFrWF with $Q = 4$ and $Q = 8$ requires 4.656 kB and 2.352 kB of memory; whereas, the SMFrWF requires 7.684 kB and

TABLE 5. Time required in seconds for computing five levels of wavelet transform for different image sizes using different methods to compute the DWT using 5/3 filter; the numbers in brackets is the number Q of line segments.

Time (s)	Image size N ($N \times N$ pixels)							
Method	256	512	768	1024	2048	2304	4096	8192
Conv. DWT	0.0164	0.0533	0.1213	0.2170	0.8349	1.0391	3.236	14.938
FrWF	0.0233	0.0694	0.1526	0.3427	1.4985	1.8219	6.489	30.121
SFrWF (2)	0.0348	0.0913	0.1842	0.3953	1.5392	1.8226	6.625	35.112
SFrWF (4)	0.0512	0.1232	0.3250	0.5104	1.7712	2.1034	6.909	35.513
SFrWF (8)	0.0793	0.1784	0.4466	0.6313	1.9867	2.4124	7.528	37.372
MFrWF	0.0124	0.0335	0.0713	0.1342	0.5532	0.6809	2.284	11.171
SMFrWF (2)	0.0154	0.0488	0.0915	0.1677	0.6190	0.7641	2.592	13.133
SMFrWF (4)	0.0317	0.0655	0.1192	0.1978	0.6946	0.8579	2.745	13.872
SMFrWF (8)	0.0608	0.1061	0.1688	0.2585	0.8245	1.0129	3.134	14.744

3.844 kB of on-board RAM memory. On the other hand for the same resolution image, the conventional DWT, MFrWF, and FrWF require 18.87 MB (megabyte), 30.72 kB, and 18.432 kB of RAM, respectively. Thus, from a memory point of view the SFrWF and proposed SMFrWF satisfy the memory constraint of low-cost visual sensors even for HR-images.

B. COMPLEXITY ANALYSIS

In this section, the MFrWF computational complexity in terms of time (seconds) consumed in computing the 2-D DWT coefficients is measured and compared with the conventional DWT, FrWF, and SFrWF. The measured computation times (in seconds) for computing five levels of the 2-D DWT with the 5/3 filter bank for different size images are given in Table 5. The table gives the total times required to compute the DWT coefficients, including read/write access time, arithmetic (addition and multiplication) computation time, and comparison time (including modulo-4 operation in MFrWF). We observe the following from Table 5:

1) The computation time of each method increases with the image resolution and with the number of segments (for SFrWF and SMFrWF).
2) The complexity of the proposed MFrWF is the lowest among all the considered methods of computing the 2-D DWT for all considered image resolutions. The conventional DWT has a complexity of almost the same order as the MFrWF, but the conventional DWT requires very large memory, particularly for HR images. The MFrWF achieves substantial complexity reductions compared to the FrWF, mainly due to the reduced line read operations in the MFrWF.
3) The segmented MFrWF (SMFrWF) is less complex than the segmented FrWF (SFrWF) for a given number of line segments. The complexity of the SMFrWF (4) is higher than that of the conventional DWT for images of size up to 512×512; however, for higher resolution images, the conventional DWT and FrWF (with and without segmentation) have higher time complexity than the SMFrWF (4).

Importantly, the proposed SMFrWF achieves significant complexity reductions while having sufficiently low memory requirements to make the processing of large HR images feasible on low-memory nodes. For instance, for HR images of dimension 4096×4096 pixels, the SMFrWF with $Q = 8$ segments needs only 7.684 kB of RAM (compared to 4.656 kB for the corresponding SFrWF, see Table 4) thus being readily feasible on nodes with 10 kB RAM. As specified in Table 5, the corresponding SMFrWF time complexity of 3.134 s is less than half of the SFrWF time complexity of 7.528 s, thus the SMFrWF has less than half the complexity of the SFrWF.

Although the MFrWF requires some comparisons to decide the particular operation to be performed on an image line, its time complexity is substantially reduced due to the nearly halved memory access time. The MFrWF has even lower time complexity than the conventional DWT. Therefore, we conclude that the proposed MFrWF (with and without segmentation) provides a good trade-off between memory and computational complexity and is thus suitable for transforming/coding of HR images on low-cost visual sensors.

C. CODING EFFICIENCY

Similar to other methods of computing the 2-D DWT, the MFrWF (with and without segmentation) results in the same set of transform coefficients, which may be quantized and encoded using any state-of-the-art wavelet image coding algorithm. Since the different DWT computation methods generate the same subband coefficients, they give the same peak-signal-to-noise-ratio (PSNR) at a particular bit-rate when they are combined with a wavelet-based image coding algorithm. When DWT, FrWF (with and without line segmentation), and MFrWF (with and without line segmentation) are combined with ZM-SPECK, to encode the *Bike*, 2048×2048 image, the PSNR values obtained at bit rates 0.2, 0.3, and 0.5 bit per pixel (bpp) are 23.7551 dB, 24.4655 dB, and 29.7488, respectively. The original image "*Bike*" with 2048×2048 pixels and the reconstructed images for the different bitrates when the transform is computed using the SMFrWF with $Q = 8$ segments are shown in Fig. 7. We observe from Fig. 7 that there are no blocking artifacts in the reconstructed image at any of the bitrates. The segmentations in SFrWF and SMFrWF employ overlaps (see

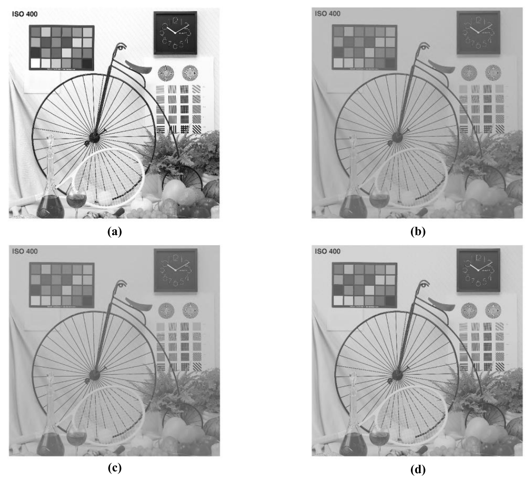


FIGURE 7. (a) Original image '*Bike*' (2048 × 2048) and reconstructed images after SMFrWF transform (five levels, 5/3 filter) with $Q = 8$ segments and ZM-SPECK encoding at (b) 0.1 bpp, (c) 0.3 bpp, and (d) 0.5 bpp.

Appendices C. and E.) to avoid border continuities, which could lead to artifacts.

In order to further demonstrate the compatibility of the proposed MFrWF with the coding algorithms, we combined the MFrWF (five levels, 5/3 filter) with several coding algorithms, namely SPIHT [56], SPECK [57], WBTC [58], and their low-memory versions namely NLS [59], LSK [60], ZM-SPECK [24], and LMBTC [25]. Since the transformed coefficients are independent of the DWT computation method, the bitstream of a specific coding algorithm and the quality of the decoded images at a particular bit rate are the same, irrespective of the DWT computation method. Considering the memory requirement of the coding algorithms, we note that LMBTC [25] and ZM-SPECK [24] satisfy the memory constraint of low-cost sensor nodes. ZM-SPECK is almost memoryless (requires only a few registers) and requires negligibly small memory. Therefore, keeping the memory constraint of low-cost sensor nodes, it is advisable to combine the proposed MFrWF with the ZM-SPECK coding algorithm.

TABLE 6. BD-PSNR gains in dB in the range of 0.01–0.1 bits per pixel of ZM-SPECK for 2048 × 2048 *Bike* image.

SPECK	LSK	SPIHT	NLS	WBTC	LMBTC
1.052	0.170	0.749	0.732	0.643	0.064

We present the Bjontegaard delta PSNR (BD-PSNR) [61] gain of ZM-SPECK with respect to other coding algorithms for the *Bike* image of dimension 2048 × 2048 in Table 6. We observe from Table 6 that ZM-SPECK outperforms the other coders at very low bit rates. This is mainly due to the one-pass coefficient encoding in ZM-SPECK (instead of two passes, namely sorting and refinement passes in SPIHT, SPECK, WBTC, NLS, and LSK) which mixes the sorting, sign, and refinement bits. Although ZM-SPECK encodes fewer new significant coefficients than SPIHT, SPECK, WBTC, NLS, and LSK, it encodes more refinement bits, and the overall PSNR gain due to refinement bits is typically slightly higher than the PSNR degradation due to fewer new significant coefficients at very low bit rates [24]. The coding efficiency of ZM-SPECK is higher than that of LMBTC due to the block-based nature of ZM-SPECK [24].

Since the overall memory requirement is the maximum of the memory required for MFrWF and ZM-SPECK, the overall memory required is still governed by that of MFrWF, as ZM-SPECK consumes negligible memory. Thus, the proposed MFrWF combined with ZM-SPECK is attractive for image coding within the constraints of low-cost visual sensors even for higher resolution images.

D. HARDWARE IMPLEMENTATION

1) DESIGN

Fig. 8 shows the top-level block diagram of the proposed MFrWF hardware architecture. The input data, i.e., image pixels, are read serially into the buffer B. From this buffer, five pixels (say x_0, x_1, x_2, x_3, and x_4) are fed in sliding window fashion into the 1-D DWT computation block to perform horizontal filtering. The 1-D DWT computation block (shown in Fig. 9) performs the 1-D DWT on these pixels to generate one coefficient each of the L and H subbands. The window is then slided by two pixels in a step, until the end of the

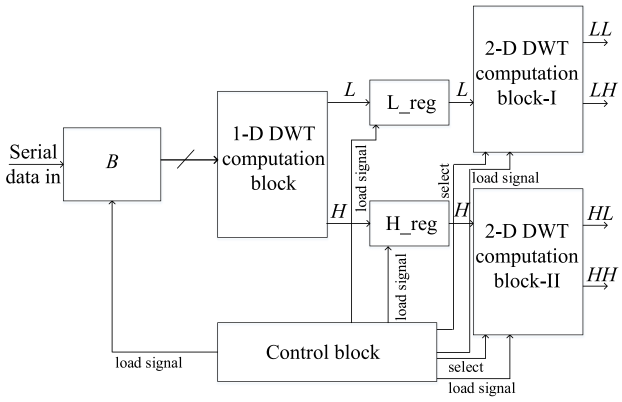


FIGURE 8. Block diagram of top-level FPGA architecture of MFrWF.

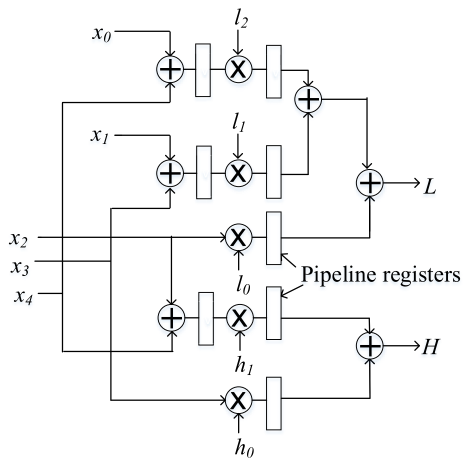


FIGURE 9. Schematic diagram of 1-D DWT computation block.

line. The same process is repeated on each line to generate the L and H subband coefficients according to Eqns. (1) and (2), respectively. For the symmetric 5/3 filter bank, $l_{-2} = l_2, l_{-1} = l_1$, and $h_{-1} = h_1$, thus Eqns. (1) and (2) can be simplified for the computation of a single L and H subband coefficient for a given line $i = 1$ to

$$L(1) = (x_0 + x_4)l_2 + (x_1 + x_3)l_1 + x_2 l_0 \qquad (8)$$

$$H(1) = (x_2 + x_4)h_1 + x_3 h_0, \qquad (9)$$

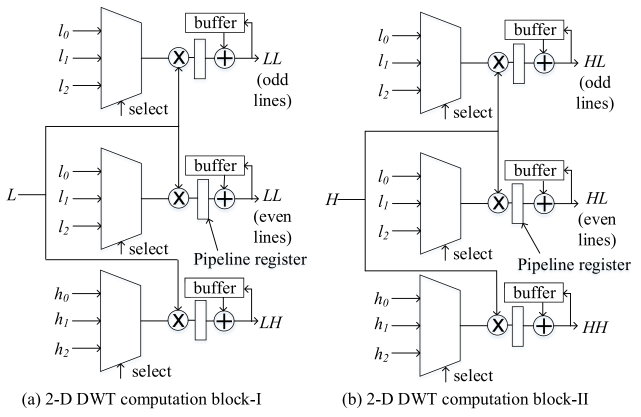


FIGURE 10. Schematic diagram of 2-D DWT computation blocks.

as implemented in Fig. 9. The computed L and H subband coefficients are saved in registers L_reg and H_reg, respectively. The L subband coefficient of L_reg is used as input for the 2-D DWT computation block-I (see Fig. 10(a)) to compute the LL and LH subband coefficients. Similarly, the coefficient of H_reg is used as input for the 2-D DWT computation block-II (see Fig. 10(b)) to compute the HL and HH subband coefficients. In Fig. 10(a), even and odd lines of the LL subband are computed through Processes 2 and 3, respectively (see Section III-B.2). Similarly, the even and odd lines of the HL subband are computed through Processes 2 and 3 (see Fig. 10(b)). By inserting pipeline registers after the multipliers and adders shown in Figs. 9–10, the critical path delay (CPD) of the proposed MFrWF architecture equals the multiplier delay T_m. The FrWF architecture [55] also uses pipeline registers to achieve a CPD of T_m.

The described MFrWF hardware architecture is independent of the segment size, i.e., accommodates the SMFrWF in a straightforward fashion. The segmentation does not require any extra pipeline registers. However, the segmentation reduces the required input pixel buffer size.

2) EVALUATION

We compare the proposed MFrWF architecture with the existing FrWF architecture [55]. We implemented both architectures on the same Xilinx FPGA platform (Family: Artix-7, Device: xc7a100t, Package: csg324, Speed: −3) using identical multipliers, adders, and other components provided by the Xilinx Artix-7 FPGA family. Both architectures use an input pixel width of 8 bits and a data-path width of 16 bits.

Comparing the hardware utilizations of both architectures for the 5/3 filter, we found that both architectures require four digital signal processors (DSPs), and do not require any look up table RAM (LUTRAM). The MFrWF architecture needs only one global buffer (BUFG), whereas the FrWF architecture needs two BUFGs. Moreover, the MFrWF architecture has 12 adders and 11 multipliers, while the FrWF architecture has 8 adders and 10 multipliers. As both architectures have the same CPD of T_m, they can both operate at the maximum frequency of 400 MHz. Additional hardware utilization characteristics are summarized in Table 7, indicating similar numbers of look up tables, flip flops, and input/output pins.

TABLE 7. Comparison of post-implementation hardware utilization for MFrWF for image dimension 2048 × 2048 with 5/3 filter bank.

Parameters	FrWF [55]	MFrWF
Number of Look Up Tab.	215	219
Number of Flip Flops	305	308
Number of IO pins	173	170
Total number of cycles	5,250,048	2,101,248
Power consumption [W]	0.695	0.603
Energy consumption [mJ]	9.486	3.294

TABLE 8. Compute cycles as well as power and energy consumption of SMFrWF for different segment numbers Q for image dimension 2048 × 2048 with 5/3 filter bank.

Param.	$Q = 2$	$Q = 4$	$Q = 8$
# cycles	2,105,344	2,113,536	2,129,920
Power [W]	0.615	0.640	0.661
Energy [mJ]	3.366	3.516	3.660

We evaluated the total number of cycles and the average power consumption for the 2048 × 2048 image size and the 5/3 filter with the Xilinx Vivado software suite, version 2018.2. We evaluated the consumed energy by multiplying the number of cycles with the average power consumption and the clock duration of 2.6 ns. We observe from Table 7 that the MFrWF architecture reduces the energy consumption almost down to a third of the FrWF energy consumption, which is mainly due to the reduction of the cycles to less than half with the MFrWF. We observe from Table 8 that for an increasing number of segments Q, the SMFrWF has slightly increasing cycle counts and energy consumption. This is mainly because the MFrWF reads a line in one cycle, whereas the SMFrWF reads a line in Q segments over Q cycles. The number of computation cycles remains essentially unaffected by the line segmentation.

E. MEMORY-COMPLEXITY TRADE-OFF

The memory and time complexity to compute five levels of the wavelet decomposition using the MFrWF and the FrWF (without segmentation ($Q = 1$) and with $Q = 2, 4$, and 8 segments) as well as the conventional DWT for image dimension 2048 × 2048 are shown in Fig. 11. We observe from Fig. 11 that for an increasing number of segments Q,

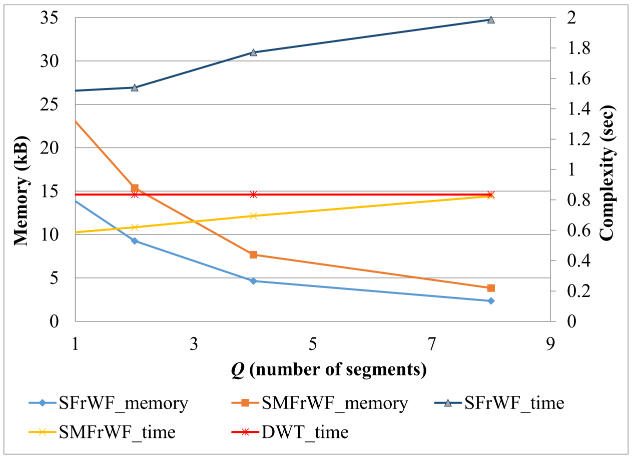


FIGURE 11. Memory-computational complexity (time) trade-off of SMFrWF and SFrWF for image size 2048 × 2048: Segmentation enables flexible memory size vs. computation complexity trade-offs. For $Q = 1$, SFrWF corresponds to FrWF and SMFrWF corresponds to MFrWF.

the FrWF and MFrWF memory requirements decrease, while the time complexities increase. Thus, there exists a trade-off between memory and complexity. This is a valuable feature since WMSNs/IoT devices are heterogeneous with varying memory and computational capabilities. The proposed MFrWF provides the flexibility to select the number of line segments Q according to the available on-board memory and computational power of the WMSNs/IoT devices.

V. CONCLUSION

We have developed and evaluated a novel approach for low-memory computation of the two-dimensional (2-D) discrete wavelet transform (DWT) with low computational (time) complexity. The proposed Modified Fractional Wavelet Filter (MFrWF) reduces the computational complexity of the underlying FrWF by avoiding multiple readings of image lines, thus reducing the memory access time. Once a line is read from the external SD card, all necessary operations (for convolution) that are required for computing the 2-D DWT are performed and the results are stored in multiple buffers. Thus, the reduction in memory access time is achieved at the cost of slightly increased buffer memory. To compensate for the increased memory, we incorporated line segmentation into the MFrWF, resulting in the Segmented MFrWF (SMFrWF).

We have verified through extensive simulations and hardware implementations that the proposed MFrWF has substantially lower complexity than the conventional DWT and FrWF, while generating exactly the same wavelet transform coefficients. Moreover, the complexities of the SMFrWF for large HR images are less than for the FrWF and the corresponding SFrWF. We observed a trade-off between the memory requirement and computational complexity that is controlled by the number of line segments, which is a useful feature of the SMFrWF. We examined a MFrWF hardware architecture and observed reduced energy consumption compared to an equivalent FrWF architecture. Although, this paper focused on reducing the computational complexity of the forward DWT, the proposed methods are equally applicable to the inverse DWT.

There are several interesting future research directions related to DWT computation methods that require only low memory and incur low time complexity. One interesting future research direction is to examine image transformation and coding in the context of emerging network function virtualization [62]. With network function virtualization, a given physical IoT node or WMSN can support multiple virtual sensing services in parallel. Our low-memory and low complexity image transform could enable multiple parallel image sensing functions to be performed on a physical sensor node with limited hardware resources. Moreover, the sensing and computing in sensor nodes can become a ‘‘sensing layer’’ in emerging layered architectures that integrate the communication and computing functions in wireless networks [63], [64]. Future research should examine whether the computing resources that have been freed up with the proposed low-complexity DWT method could be utilized for more advanced processing, such as object detection [65], in the sensing layer.

This study has focused on the image wavelet transform, which is a preliminary step towards image compression (source coding). Often, the encoded images need to be transmitted over error-prone wireless networks, which require some coding for error resilience. For complex wireless networks with limited coordination, such as wireless sensor networks, network coding has recently emerged as an attractive coding approach [66]–[69]. An interesting future research direction is to jointly examine image wavelet transform, image coding, and network coding on resource-constrained sensor nodes.

APPENDIX A
ANALYSIS OF CONVENTIONAL DWT

A. FILTERING AN IMAGE LINE

In this and the subsequent appendices we derive the computational complexities in terms of the number of arithmetic (addition and multiplication) operations as well as the number of read and write operations listed in Table 3 for the different computational methods (conventional DWT, FrWF, SFrWF, MFrWF, and SMFrWF) for the 2-D DWT of images. We note that all methods perform the horizontal filtering of image lines in the same way. Therefore, the number of additions and multiplications required for computing one line of the L and H subbands is the same for all considered DWT computation methods. A coefficient of the L subband is obtained by convolving n_l coefficients of an image line with the LPF according to Eqn. (1), which requires $n_l - 1$ additions and $\frac{n_l+1}{2}$ multiplications (for symmetrical filters). Similarly, the computation of one H subband coefficient (obtained by convolution with a symmetrical HPF of length n_h) requires $n_h - 1$ additions and $\frac{n_h+1}{2}$ multiplications. Each line of the L and H subbands contains $N/2$ coefficients. The complexity of the horizontal filtering of one image line is the sum of the numbers of the respective arithmetic operations required for the LPF and HPF filtering operations. Thus, the total computation time required for one line of the L and H subbands (assuming $n_h = n_l - 2$) is

$$C_{\text{Line}} = \frac{N}{2}\left[2(n_l - 2)a + n_l m\right], \tag{10}$$

where a and m denote the time required for one addition and one multiplication, respectively.

B. CONVENTIONAL DWT OF AN IMAGE

In the conventional DWT, each of the N image lines is first horizontally filtered, which requires NC_{Line} computations. The subsequent column-wise vertical filtering requires also NC_{Line} computations, for a total computation time of $2\,NC_{\text{Line}}$ for the 2-D DWT.

During the horizontal filtering in the conventional DWT, each image line of N coefficients needs to be read once; thus, N^2 read operations are required to compute the L and H subbands. For the low-memory version of the conventional DWT, the L and H subbands (each subband with $N/2$ columns, and each column with N coefficients) resulting from the horizontal filtering are written to the SD card (N^2 write operations). For the vertical filtering, each column (consisting of N coefficients) of the L and H subbands (total of N columns) needs to be read from the SD card (N^2 read operations) to compute the LL, LH, HL, and HH subbands via vertical filtering. The final 2-D DWT transform coefficients are written to the SD card. Thus, a total of $2N^2$ read operations and $2\,N^2$ write operations are required for the low-memory version of the conventional 2-D DWT computation method, which requires N^2 memory elements for transform coefficients, equivalent to $4N^2$ bytes of memory (for floating point computations). An alternative version of the conventional DWT computation avoids the writing of the L and H subbands to the SD card, by storing them in N^2 additional memory elements for transform coefficients ($4N^2$ bytes), for a total of $8\,N^2$ bytes of memory. Since this study focuses on low-memory settings, we consider the low-memory version of the conventional DWT with $4N^2$ bytes of memory for comparisons.

We note that in the other DWT computation methods, the lines after horizontal filtering are multiplied by filter coefficients and saved in RAM buffers. The buffers are updated through successive operations to compute the LL, LH, HL, and HH subbands, and the final results are written to the SD card. Thus, these other DWT computation methods avoid the writing of the L or H subbands to the SD card and require therefore only N^2 write operations.

APPENDIX B FrWF ANALYSIS

A. ANALYSIS OF FrWF ARITHMETIC OPERATIONS

The FrWF works on the basis of a VFA consisting of n_l lines; the VFA is shifted by two image lines at a time. We first evaluate the time required for the arithmetic operations in a VFA. The horizontal filtering of a VFA requires $n_l C_{\text{Line}}$ operations, with C_{Line} given by Eqn. (10). From a VFA, one line each of the LL, LH, HL, and HH subbands will be computed. For computing a line of the LL and HL subbands, n_l rows (after 1-D horizontal filtering) need to be multiplied by the LPF coefficients (depending on the line index in the VFA, each row will be multiplied by a specific filter coefficient) and then added together. Thus, computing one line of the LL and HL subbands requires

$$C_{\text{LL\_HL}} = N\left[(n_l - 1)a + n_l m\right]. \tag{11}$$

Similarly, for computing one line of the LH and HH subbands, n_h rows (after 1-D horizontal filtering) need to be multiplied by the HPF filter coefficients (depending on the line index in VFA, each row will be multiplied by a specific filter coefficient) and then added together. The computation time required for computing one line of LH and HH subbands (considering $n_h = n_l - 2$) is

$$C_{\text{LH\_HH}} = N[(n_l - 3)a + (n_l - 2)m]. \tag{12}$$

The total computation time (for arithmetic operations) required for a VFA is the sum of $n_l C_{\text{Line}}$ as well as Eqns. (11) and (12). The VFA is shifted $N/2$ times in order to cover the entire image, resulting in the total arithmetic computation time

$$C_{\text{Arithm\_FrWF}} = \frac{N^2}{2}(n_l^2 - 4)a + \frac{N^2}{4}(n_l^2 + 4n_l - 4)m. \tag{13}$$

B. ANALYSIS OF FrWF READ OPERATIONS

During vertical filtering in the 2-D DWT, row g of the LL subband is computed according to

$$LL(g) = \sum_{j=-\lfloor \frac{n_l}{2} \rfloor}^{j=\lfloor \frac{n_l}{2} \rfloor} l_j L(2g + j - 1); \quad g = 1, 2, \ldots, \frac{N}{2}, \tag{14}$$

whereby we denote row $2g+j-1$ of the L subband by $L(2g+j-1)$ and denote LPF coefficient j by l_j. In the FrWF, n_l lines of the L subband are required to compute one line of the LL subband. In order to verify that even these lines are to be read multiple times, consider the computation of three consecutive rows, say rows g, $g+1$, and $g+2$ of the LL subband using a 5/3 filter ($n_l = 5$). For these three lines, Eqn. (14) may be written in expanded form as

$$\begin{aligned} LL(g) &= l_{-2}L(2g-3) + l_{-1}L(2g-2) \\ &\quad + l_0 L(2g-1) + l_1 L(2g) + l_2 L(2g+1) \qquad (15) \\ LL(g+1) &= l_{-2}L(2g-1) + l_{-1}L(2g) + l_0 L(2g+1) \\ &\quad + l_1 L(2g+2) + l_2 L(2g+3) \qquad (16) \\ LL(g+2) &= l_{-2}L(2g+1) + l_{-1}L(2g+2) + l_0 L(2g+3) \\ &\quad + l_1 L(2g+4) + l_2 L(2g+5). \qquad (17) \end{aligned}$$

From these expanded Eqns. (15)–(17), it is clear that the (odd-indexed) row $2g+1$ of the L subband is read thrice, once for the computation of each equation, and its elements are multiplied by l_2, l_0, and l_{-2}, respectively. Similarly, the (even-indexed) row $2g + 2$ of the L subband is read twice, to be multiplied by l_1 and l_{-1}. The HL subband is computed in a similar way and Eqns. (14)–(17) can be re-used, except for replacing the L subband coefficients by the H subband coefficients.

Similarly, it can be demonstrated that the LH and HH subband coefficients are computed by convolving the L and H subbands with the HPF of length n_h ($n_h = 3$ for 5/3 filter bank), for which odd-indexed lines are read twice (once to be multiplied by h_{-1} and once by h_1) and even indexed lines are read once and multiplied by the h_0 filter coefficient.

Thus, we conclude that the FrWF requires multiple readings of image lines for the computation of the vertical convolution. The number of times a line is read varies between $(n_h - 1)/2$ and $(n_l + 1)/2$, according to the filter lengths.

A VFA encompasses n_l horizontal image lines. The VFA is shifted down by two lines to achieve vertical downsampling by a factor of two. Overall, the VFA is shifted down $N/2$ times to cover all N image lines [20]. Thus, the FrWF needs to read $\frac{n_l N}{2}$ lines, each with N coefficients for computing the 2-D DWT of an image, resulting in a total of $\frac{n_l N^2}{2}$ read operations.

APPENDIX C SFrWF ANALYSIS

The SFrWF implements the FrWF on segmented image lines [54]. The number of multiplication operations in the SFrWF is the same as in the FrWF [54]. However, apart from the additions needed by the FrWF, the SFrWF needs $n_l N(Q-1)(n_l-2)$ extra additions due to the overlap and add (OLA) method [70], [71] as analyzed next.

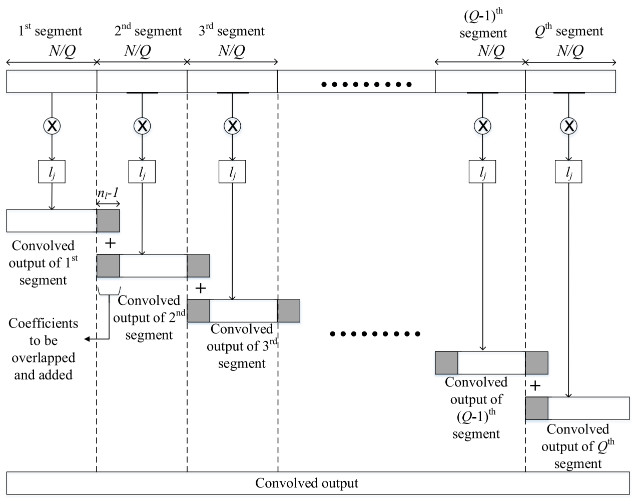

1st segment
N/Q
2nd segment
N/Q
3rd segment
N/Q
(Q-1)th segment
N/Q
Qth segment
N/Q
X
l_j
n_l-1
+
Convolved output of 1st segment
Convolved output of 2nd segment
Convolved output of 3rd segment
Convolved output of (Q-1)th segment
Convolved output of Qth segment
Coefficients to be overlapped and added
Convolved output

FIGURE 12. Illustration of the overlap-add method (OLA) for convolution [54].

The segmented FrWF (SFrWF) [54] individually filters each segment (with a filter of length n_l). In order to overcome the boundary discontinuities, the SFrWF uses the OLA method, as illustrated in Fig. 12. We observe from Fig. 12 that when the filter outputs corresponding to the shaded (additional $n_l - 1$ terms of previous segment obtained after the convolution) overlap with the first $(n_l - 1)$ terms of the current (adjacent) segment are added together, we get the same convolved output as would have been obtained if the original image line had been convolved with the filter with coefficients l_j, $j = -n_l/2, \ldots, n_l/2$. However, due to the OLA method, the SFrWF performs some extra additions at segment boundaries, while keeping the number of multiplications and read operations the same as required in the FrWF. As evident from Fig. 12, an image line partitioned into Q segments requires $Q - 1$ OLA operations.

Overall, the separate filtering by an LPF and by an HPF requires $(Q - 1)[(n_l - 1) + (n_h - 1)]$ extra additions for processing the OLA in one VFA. Considering $n_h = n_l - 2$ (for most bi-orthogonal filters), a total of $2(Q - 1)(n_l - 2)$ extra additions are needed due to the OLA method for one VFA line. Since each of the n_l VFA lines is read and processed (one segment at a time) in the same way, the processing of each VFA requires $2n_l(Q-1)(n_l-2)$ extra additions. Considering the vertical downsampling, an image is covered with $N/2$ VFAs. Therefore, the SFrWF performs a total of $Nn_l(Q - 1)(n_l - 2)$ extra additions due to the OLA method. The overall arithmetic computation time required for the SFrWF is the FrWF computation time Eqn. (13) plus $Nn_l(Q - 1)(n_l - 2)a$.

The SFrWF reduces the FrWF memory requirement by a factor of Q due to the line segmentation (while only requiring $12(n_l - 1)$ additional bytes of memory for temporary buffers [54]). The SFrWF requires more additions due to the OLA method. Since the complexity of the FrWF is already relatively high due to multiple line reads, the additional add operations due to the OLA method in the SFrWF further increase the complexity.

APPENDIX D MFrWF ANALYSIS

The MFrWF filters the N image lines horizontally as per the conventional approach, requiring NC_{Line} arithmetic operations. After horizontal filtering, the lines are multiplied by different filter coefficients and saved in buffers followed by successive update operations to compute the LL, LH, HL, and HH subbands, as described in Section III-B. For the computation of these subbands, all the N coefficients of the even and odd numbered lines require $\frac{n_l-1}{2}$ and $\frac{n_l+1}{2}$ multiplications, respectively. Thus, Nn_l multiplications are needed for computing one line of the four subbands. Since each of the four subbands has $N/2$ lines, a total of $\frac{N^2 n_l}{2}$ multiplications are needed to compute one level of the 2-D DWT.

In the three MFrWF processes, namely P_1, P_2, and P_3, the odd and even indexed lines are multiplied by different filter coefficients and the results are either saved in intermediate buffers or added to the previous values in buffers to update their content. The multiplication of a line by l_2 or h_1 (for a 5/3 filter) indicates that the VFA processing is complete, and the buffer values after the update will correspond to a particular subband line depending on the process performed on the line. For every image line to be processed, the update operations require $n_l - 2$ additions for each line element. For the N coefficients in an image line, $N(n_l - 2)$ add operations are needed to implement the update operation of an image line. For an image with N lines, the update operations require a total of $N^2(n_l - 2)$ additions.

The total number of additions and multiplications needed for the MFrWF is the sum of NC_{Line} and the computations needed for computing the LL, LH, HL, and HH subbands, resulting in Eqn. (5).

APPENDIX E SMFrWF ANALYSIS

A. ANALYSIS OF SMFrWF ARITHMETIC OPERATIONS

Since the SMFrWF uses a slightly different line segmentation approach than the OLA method of SFrWF (as analyzed in detail in the next subsection), the numbers of arithmetic operations needed for the SMFrWF are exactly the same as for the MFrWF.

B. ANALYSIS OF SMFrWF READ OPERATIONS

In order to avoid the extra computations (specifically, additions) due to the OLA method in the SFrWF, we propose a different segmentation strategy for the SMFrWF: Let $\mathbf{X} = \{x_1, x_2, \ldots, x_N\}$ denote an N-pixel image line that is to be convolved with a filter of length n_l. For the purpose of memory reduction, we partition the line into Q segments, such that a given segment is overlapping for $\lfloor n_l/2 \rfloor$ samples with the adjacent segments at each boundary so as to avoid the border discontinuities. Let $\mathbf{X}_1$, $\mathbf{X}_q$, $q = 2, 3, \ldots, Q-1$, and $\mathbf{X}_Q$ denote the first, q^{th}, and Q^{th} (the last) segments of the image line, respectively. The elements of these segments are

$$\mathbf{X}_1 = \left\{x_1, \ldots, x_{\left(\frac{N}{Q}+\lfloor\frac{n_l}{2}\rfloor\right)}\right\}, \tag{18}$$

$$\mathbf{X}_q = \left\{x_{\left(\frac{(q-1)N}{Q}-(\lfloor\frac{n_l}{2}\rfloor-1)\right)}, \ldots, x_{\left(\frac{qN}{Q}+\lfloor\frac{n_l}{2}\rfloor\right)}\right\}, \tag{19}$$

$$\mathbf{X}_Q = \left\{x_{\left(\frac{(Q-1)N}{Q}-(\lfloor\frac{n_l}{2}\rfloor-1)\right)}, \ldots, x_N\right\}, \tag{20}$$

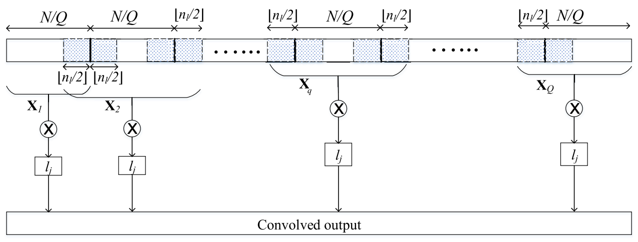


FIGURE 13. Segmentation of an image line in SMFrWF.

as illustrated in Fig. 13. In Fig. 13, the shaded portions represent the overlapping segment regions. During the filtering (convolution), the filter coefficients slide in such a way that the center of a filter is allowed to align with that portion of a segment that would have been obtained if the line had been partitioned with non-overlapping segments, i.e., the center of the filter slides only over N/Q samples bounded with solid vertical lines in Fig. 13 (and not over the samples in the shaded regions). Furthermore, the left side of the first segment and the right side of last (Q^{th}) segment are symmetrically extended to avoid border discontinuities (these extensions are not shown in Fig. 13). The extra $\lfloor n_l/2 \rfloor$ samples, shown by shaded regions on both sides of the intermediate segments $q = 2, 3, \ldots, Q-1$ in Fig. 13 are needed to perform the convolution when the center of the filter moves towards the segment boundaries and some filter coefficients extend beyond the segment boundaries.

By the design of the SMFrWF line segmentation, no additional arithmetic (add or multiply) operations need to be performed due to the line segmentation. However, due to the overlapping nature of the segments, $\lfloor n_l/2 \rfloor$ additional samples from adjacent segments need to be read. The number of samples that need to be read to process each segment (except the first and last segments) of an image line are $\frac{N}{Q} + 2\left\lfloor\frac{n_l}{2}\right\rfloor$; whereas, for the first and last segment, $\frac{N}{Q} + \left\lfloor\frac{n_l}{2}\right\rfloor$ samples are read (since the left side of the first segment and the right side of last segment are symmetrically extended). Thus, in total $N + 2(Q-1)\left\lfloor\frac{n_l}{2}\right\rfloor$ read operations are required for processing an image line in SMFrWF. For the N image lines in an $N \times N$ size image, a total of

$$N^2 + 2N(Q-1)\left\lfloor\frac{n_l}{2}\right\rfloor \tag{21}$$

read operations are needed in SMFrWF.

We emphasize that the number of additions and multiplications of the MFrWF and SMFrWF (and the conventional DWT) are the same since the centers of the filters are *not* shifted over the terms which are overlapping. However, there is small increase in the number of read operations (equal to $2N(Q-1)\left\lfloor\frac{n_l}{2}\right\rfloor$) for the SMFrWF, as compared to the MFrWF, due to the repetitive reading of some image pixels at the segment boundaries.

C. READ OPERATIONS IN SMFrWF VS. CONVENTIONAL DWT

Although the MFrWF (without line segmentation) requires significantly fewer read operations than the FrWF or the conventional DWT, the number of read operations for the SMFrWF is proportional to Q, the number of line segments, see Table 3. Since the number read operations in SMFrWF increases with Q, one may wonder whether the number of SMFrWF read operations increases above the number of read operations for the conventional DWT for large segment numbers Q. We prove in this appendix that the number of SMFrWF read operations is always less than for the conventional DWT.

We observe from Table 3 that the number of read operations required by the SMFrWF and conventional DWT, respectively, for an $N \times N$ size image are $N^2 + 2N(Q-1)\left\lfloor\frac{n_l}{2}\right\rfloor$ and $2N^2$, whereby n_l denotes the length of the highest order filter in the filter bank (which is generally the LPF). Thus, it is sufficient to prove that

$$(Q-1)2\left\lfloor\frac{n_l}{2}\right\rfloor < N. \tag{22}$$

Considering the largest possible number of segments $Q = \frac{N}{n_l+2\lfloor\frac{n_l}{2}\rfloor}$ for a given image size (with N pixels per line),

$$\left(\frac{N}{n_l+2\lfloor\frac{n_l}{2}\rfloor}-1\right)2\left\lfloor\frac{n_l}{2}\right\rfloor < N \tag{23}$$

$$\Leftrightarrow -2\left\lfloor\frac{n_l}{2}\right\rfloor\left(n_l+2\left\lfloor\frac{n_l}{2}\right\rfloor\right) < Nn_l, \tag{24}$$

which holds true for all positive integers N and n_l.

REFERENCES

[1] J. Jiang, X. Ma, C. Chen, T. Lu, Z. Wang, and J. Ma, "Single image super-resolution via locally regularized anchored neighborhood regression and nonlocal means," *IEEE Trans. Multimedia*, vol. 19, no. 1, pp. 15–26, Jan. 2017.

[2] M. Elhoseny, G. Ramírez-González, O. M. Abu-Elnasr, S. A. Shawkat, A. N, and A. Farouk, "Secure medical data transmission model for IoT-based healthcare systems," *IEEE Access*, vol. 6, pp. 20596–20608, 2018.

[3] R. J. Mstafa, K. M. Elleithy, and E. Abdelfattah, "A robust and secure video steganography method in DWT-DCT domains based on multiple object tracking and ECC," *IEEE Access*, vol. 5, pp. 5354–5365, 2017.

[4] Y. Shu, C. Han, M. Lv, and X. Liu, "Fast super-resolution ultrasound imaging with compressed sensing reconstruction method and single plane wave transmission," *IEEE Access*, vol. 6, pp. 39298–39306, 2018.

[5] H. Zheng, K. Zeng, D. Guo, J. Ying, Y. Yang, X. Peng, F. Huang, Z. Chen, and X. Qu, "Multi-contrast brain MRI image super-resolution with gradient-guided edge enhancement," *IEEE Access*, vol. 6, pp. 57856–57867, 2018.

[6] G. Latif, D. N. F. A. Iskandar, J. M. Alghazo, and N. Mohammad, "Enhanced MR image classification using hybrid statistical and wavelets features," *IEEE Access*, vol. 7, pp. 9634–9644, 2019.

[7] S. A. Azzouzi, A. Vidal-Pantaleoni, and H. A. Bentounes, "Monitoring desertification in Biskra, Algeria using Landsat 8 and Sentinel-1A images," *IEEE Access*, vol. 6, pp. 30844–30854, 2018.

[8] G. Chander and B. Markham, "Revised Landsat-5 TM radiometric calibration procedures and postcalibration dynamic ranges," *IEEE Trans. Geosci. Remote Sens.*, vol. 41, no. 11, pp. 2674–2677, Nov. 2003.

[9] S. Li and G. Jiang, "Land surface temperature retrieval from Landsat-8 data with the generalized split-window algorithm," *IEEE Access*, vol. 6, pp. 18149–18162, 2018.

[10] S. C. Park, M. K. Park, and M. G. Kang, "Super-resolution image reconstruction: A technical overview," *IEEE Signal Process. Mag.*, vol. 20, no. 3, pp. 21–36, May 2003.

[11] A. Madanayake, R. J. Cintra, V. Dimitrov, F. Bayer, K. A. Wahid, A. Edirisuriya, U. Potluri, S. Madishetty, and N. Rajapaksha, "Low-power VLSI architectures for DCT/DWT: Precision vs approximation for HD video, biomedical, and smart antenna applications," *IEEE Circuits Syst. Mag.*, vol. 15, no. 1, pp. 25–47, 1st Quart., 2015.

[12] C. Chrysafis and A. Ortega, "Line-based, reduced memory, wavelet image compression," *IEEE Trans. Image Process.*, vol. 9, no. 3, pp. 378–389, Mar. 2000.

[13] J. Oliver and M. Perez Malumbres, "On the design of fast wavelet transform algorithms with low memory requirements," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 18, no. 2, pp. 237–248, Feb. 2008.

[14] P. Cosman and K. Zeger, "Memory constrained wavelet based image coding," *IEEE Signal Process. Lett.*, vol. 5, no. 9, pp. 221–223, Sep. 1998.

[15] I. F. Akyildiz, T. Melodia, and K. R. Chowdhury, "Wireless multimedia sensor networks: Applications and testbeds," *Proc. IEEE*, vol. 96, no. 10, pp. 1588–1605, Oct. 2008.

[16] M. A. Jan, M. Usman, X. He, and A. U. Rehman, "SAMS: A seamless and authorized multimedia streaming framework for WMSN-based IoMT," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 1576–1583, Apr. 2019.

[17] H. Tao, M. Z. A. Bhuiyan, A. N. Abdalla, M. M. Hassan, J. M. Zain, and T. Hayajneh, "Secured data collection with hardware-based ciphers for IoT-based healthcare," *IEEE Internet Things J.*, vol. 6, no. 1, pp. 410–420, Feb. 2019.

[18] M. M. Rana and W. Xiang, "IoT communications network for wireless power transfer system state estimation and stabilization," *IEEE Internet Things J.*, vol. 5, no. 5, pp. 4142–4150, Oct. 2018,

[19] O. Elijah, T. A. Rahman, I. Orikumhi, C. Y. Leow, and M. N. Hindia, "An overview of Internet of Things (IoT) and data analytics in agriculture: Benefits and challenges," *IEEE Internet Things J.*, vol. 5, no. 5, pp. 3758–3773, Oct. 2018.

[20] S. Rein and M. Reisslein, "Low-memory wavelet transforms for wireless sensor networks: A tutorial," *IEEE Commun. Surveys Tuts.*, vol. 13, no. 2, pp. 291–307, 2nd Quart., 2011.

[21] S. Rein and M. Reisslein, "Performance evaluation of the fractional wavelet filter: A low-memory image wavelet transform for multimedia sensor networks," *Ad Hoc Netw.*, vol. 9, no. 4, pp. 482–496, Jun. 2011.

[22] Y. Chung-Hsien, W. Jia-Ching, W. Jhing-Fa, and C.-W. Chang, "A block-based architecture for lifting scheme discrete wavelet transform," *IEICE Trans. Fundam. Electr., Commun. Comput. Sci.*, vol. 90, no. 5, pp. 1062–1071, May 2007.

[23] L. W. Chew, W. C. Chia, L.-M. Ang, and K. P. Seng, "Very low-memory wavelet compression architecture using strip-based processing for implementation in wireless sensor networks," *EURASIP J. Embedded Syst.*, vol. 2009, no. 479281, pp. 1–16, Dec. 2009.

[24] N. R. Kidwai, E. Khan, and M. Reisslein, "ZM-SPECK: A fast and memoryless image coder for multimedia sensor networks," *IEEE Sensors J.*, vol. 16, no. 8, pp. 2575–2587, Apr. 2016.

[25] M. Tausif, N. R. Kidwai, E. Khan, and M. Reisslein, "FrWF-based LMBTC: Memory-efficient image coding for visual sensors," *IEEE Sensors J.*, vol. 15, no. 11, pp. 6218–6228, Nov. 2015.

[26] S. A. Rein, F. H. Fitzek, C. Gühmann, and T. Sikora, "Evaluation of the wavelet image two-line coder: A low complexity scheme for image compression," *Signal Process. Image Commun.*, vol. 37, pp. 58–74, Sep. 2015.

[27] M. Vishwanath, "The recursive pyramid algorithm for the discrete wavelet transform," *IEEE Trans. Signal Process.*, vol. 42, no. 3, pp. 673–676, Mar. 1994.

[28] L. Ye, J. Guo, B. Nutter, and S. Mitra, "Low-memory-usage image coding with line-based wavelet transform," *Opt. Eng.*, vol. 50, no. 2, pp. 027005-1–027005-11, Feb. 2011.

[29] Y. Bao and C.-C. J. Kuo, "Design of wavelet-based image codec in memory-constrained environment," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 11, no. 5, pp. 642–650, May 2001.

[30] C.-K. Hu, W.-M. Yan, and K.-L. Chung, "Efficient cache-based spatial combinative lifting algorithm for wavelet transform," *Signal Process.*, vol. 84, no. 9, pp. 1689–1699, 2004.

[31] V. Ratnakar, "TROBIC: Two-row buffer image compression," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Proc. (ICASSP)*, vol. 6, Mar. 1999, pp. 3133–3136.

[32] L. W. Chew, W. C. Chia, L.-M. Ang, and K. P. Seng, "Low–memory video compression architecture using strip–based processing for implementation in wireless multimedia sensor networks," *Int. J. Sensor Netw.*, vol. 11, no. 1, pp. 33–47, Jan. 2012.

[33] W. C. Chia, L. W. Chew, L.-M. Ang, and K. P. Seng, "Low memory image stitching and compression for WMSN using strip-based processing," *Int. J. Sensor Netw.*, vol. 11, no. 1, pp. 22–32, Jan. 2012.

[34] L. W. Chew, L. M. Ang, and K. P. Seng, "New virtual SPIHT tree structures for very low memory strip-based image compression," *IEEE Signal Process. Lett.*, vol. 15, pp. 389–392, 2008.

[35] R. K. Bhattar, K. R. Ramakrishnan, and K. S. Dasgupta, "Strip based coding for large images using wavelets," *Signal Process., Image Commun.*, vol. 17, no. 6, pp. 441–456, Jul. 2002.

[36] L. Ye and Z. Hou, "Memory efficient multilevel discrete wavelet transform schemes for JPEG2000," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 25, no. 11, pp. 1773–1785, Nov. 2015.

[37] C.-T. Huang, P.-C. Tseng, and L.-G. Chen, "Generic RAM-based architectures for two-dimensional discrete wavelet transform with line-based method," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 15, no. 7, pp. 910–920, Jul. 2005.

[38] Y.-H. Seo and D.-W. Kim, "VLSI architecture of line-based lifting wavelet transform for motion JPEG2000," *IEEE J. Solid-State Circuits*, vol. 42, no. 2, pp. 431–440, Feb. 2007.

[39] W.-H. Chang, Y.-S. Lee, W.-S. Peng, and C.-Y. Lee, "A line-based, memory efficient and programmable architecture for 2D DWT using lifting scheme," in *Proc IEEE Int. Symp. Circuits Syst. (ISCAS)*, vol. 4, May 2001, pp. 330–333.

[40] C.-Y. Xiong, J.-W. Tian, and J. Liu, "Efficient high-speed/low-power line-based architecture for two-dimensional discrete wavelet transform using lifting scheme," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 16, no. 2, pp. 309–316, Feb. 2006.

[41] B. K. Mohanty, A. Mahajan, and P. K. Meher, "Area- and power-efficient architecture for high-throughput implementation of lifting 2-D DWT," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 59, no. 7, pp. 434–438, Jul. 2012.

[42] C. Cheng and K. K. Parhi, "High-speed VLSI implementation of 2-D discrete wavelet transform," *IEEE Trans. Signal Process.*, vol. 56, no. 1, pp. 393–403, Jan. 2008.

[43] K. Andra, C. Chakrabarti, and T. Acharya, "A VLSI architecture for lifting-based forward and inverse wavelet transform," *IEEE Trans. Signal Process.*, vol. 50, no. 4, pp. 966–977, Apr. 2002.

[44] B.-F. Wu and C.-F. Lin, "Memory-efficient architecture for JPEG 2000 coprocessor with large tile image," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 53, no. 4, pp. 304–308, Apr. 2006.

[45] A. D. Darji, S. S. Kushwah, S. N. Merchant, and A. N. Chandorkar, "High-performance hardware architectures for multi-level lifting-based discrete wavelet transform," *EURASIP J. Image Video Process.*, vol. 2014, nos. 1–19, p. 47, Oct. 2014.

[46] A. Mason, J. Li, K. Thomson, Y. Suhail, and K. Oweiss, "Design optimization of integer lifting DWT circuitry for implantable neuroprosthetics," in *Proc. 3rd IEEE/EMBS Special Topic Conf. Microtechnol. Med. Biol.*, May 2005, pp. 136–139.

[47] B. K. Mohanty and P. K. Meher, "Memory-efficient high-speed convolution-based generic structure for multilevel 2-D DWT," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 23, no. 2, pp. 353–363, Feb. 2013.

[48] Y. Hu and C. C. Jong, "A memory-efficient high-throughput architecture for lifting-based multi-level 2-D DWT," *IEEE Trans. Signal Process.*, vol. 61, no. 20, pp. 4975–4987, Oct. 2013.

[49] W. Jiang and A. Ortega, "Lifting factorization-based discrete wavelet transform architecture design," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 11, no. 5, pp. 651–657, May 2001.

[50] C.-T. Huang, P.-C. Tseng, and L.-G. Chen, "Analysis and VLSI architecture for 1-D and 2-D discrete wavelet transform," *IEEE Trans. Signal Process.*, vol. 53, no. 4, pp. 1575–1586, Apr. 2005.

[51] B. K. Mohanty and P. K. Meher, "Memory efficient modular VLSI architecture for high throughput and low-latency implementation of multilevel lifting 2-D DWT," *IEEE Trans. Signal Process.*, vol. 59, no. 5, pp. 2072–2084, May 2011.

[52] W. Zhang, Z. Jiang, Z. Gao, and Y. Liu, "An efficient VLSI architecture for lifting-based discrete wavelet transform," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 59, no. 3, pp. 158–162, Mar. 2012.

[53] W. Sweldens, "The lifting scheme: A construction of second generation wavelets," *SIAM J. Math. Anal.*, vol. 29, no. 2, pp. 511–546, 1998.

[54] M. Tausif, E. Khan, M. Hasan, and M. Reisslein, "SFrWF: Segmented fractional wavelet filter based DWT for low memory image coders," in *Proc. IEEE Uttar Pradesh Section Int. Conf. Electr., Comput. Electr. (UPCON)*, Oct. 2017, pp. 593–597.

[55] M. Tausif, A. Jain, E. Khan, and M. Hasan, "Efficient architectures of fractional wavelet filter (FrWF) for visual sensors and wearable devices," in *Proc. IEEE SENSORS*, Oct. 2018, pp. 1–4.

[56] A. Said and W. A. Pearlman, "A new, fast, and efficient image codec based on set partitioning in hierarchical trees," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 6, no. 3, pp. 243–250, Jun. 1996.

[57] W. A. Pearlman, A. Islam, N. Nagaraj, and A. Said, "Efficient, low-complexity image coding with a set-partitioning embedded block coder," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 14, no. 11, pp. 1219–1235, Nov. 2004.

[58] A. A. Moinuddin, E. Khan, and M. Ghanbari, "Efficient algorithm for very low bit rate embedded image coding," *IET Image Process.*, vol. 2, no. 2, pp. 59–71, Apr. 2008.

[59] F. W. Wheeler and W. A. Pearlman, "SPIHT image compression without lists," in *Proc. IEEE Conf. Acoust., Speech Signal Process.*, vol. 4. Jun. 2000, pp. 2047–2050.

[60] M. V. Latte, N. H. Ayachit, and D. K. Deshpande, "Reduced memory listless SPECK image compression," *Digit. Signal Process.*, vol. 16, no. 6, pp. 817–824, Nov. 2006.

[61] G. Bjontegaard, "Calculation of average PSNR differences between RD curves," ITU–Telecommun. Standardization Sector, Video Coding Experts Group, Austin, TX, USA, Tech. Rep. VCEG-M33, ITU-T SG16/Q6, Apr. 2001, pp. 1–4.

[62] K. Han, S. Li, S. Tang, H. Huang, S. Zhao, G. Fu, and Z. Zhu, "Application-driven end-to-end slicing: When wireless network virtualization orchestrates with NFV-based mobile edge computing," *IEEE Access*, vol. 6, pp. 26567–26577, 2018.

[63] G. Mokhtari, A. Anvari-Moghaddam, and Q. Zhang, "A new layered architecture for future big data-driven smart homes," *IEEE Access*, vol. 7, pp. 19002–19012, 2019.

[64] P. Shantharama, A. S. Thyagaturu, N. Karakoc, L. Ferrari, M. Reisslein, and A. Scaglione, "LayBack: SDN management of multi-access edge computing (MEC) for network access services and radio resource sharing," *IEEE Access*, vol. 6, pp. 57545–57561, 2018.

[65] J. Ren, Y. Guo, D. Zhang, Q. Liu, and Y. Zhang, "Distributed and efficient object detection in edge computing: Challenges and solutions," *IEEE Netw.*, vol. 32, no. 6, pp. 137–143, Nov./Dec. 2018.

[66] Y. Gao, X. Xu, Y. L. Guan, and P. H. J. Chong, "V2X content distribution based on batched network coding with distributed scheduling," *IEEE Access*, vol. 6, pp. 59449–59461, 2018.

[67] D. E. Lucani, M. V. Pedersen, D. Ruano, C. W. Sørensen, F. H. P. Fitzek, J. Heide, O. Geil, V. Nguyen, and M. Reisslein, "Fulcrum: Flexible network coding for heterogeneous devices," *IEEE Access*, vol. 6, pp. 77890–77910, 2018.

[68] A. Naeem, M. H. Rehmani, Y. Saleem, I. Rashid, and N. Crespi, "Network coding in cognitive radio networks: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1945–1973, 3rd Quart., 2017.

[69] X. Shao, C. Wang, C. Zhao, and J. Gao, "Traffic shaped network coding aware routing for wireless sensor networks," *IEEE Access*, vol. 6, pp. 71767–71782, 2018.

[70] J. S. Lim, *Two-Dimensional Signal and Image Processing*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1990.

[71] J. G. Proakis and D. Manolakis, *Digital Signal Processing: Principles, Algorithms and Applications*, 4th ed. Delhi, India: Pearson, 2006.

MOHD TAUSIF received the B.Tech. degree in electronics engineering from Gautam Buddha Technical University, Lucknow, India, in 2012, and the M.Tech. degree in communication and information system from Aligarh Muslim University (AMU), Aligarh, India, in 2014, where he is currently pursuing the Ph.D. degree. His research interests include signal processing and low-complexity image coding algorithms. He has qualified for the Junior Research Fellowship (JRF) conducted by University Grants Commission, India, in 2015. He was a recipient of the prestigious Visvesvaraya Fellowship from the Ministry of Information and Technology, Government of India, for pursuing the Ph.D. degree.

EKRAM KHAN received the B.Sc.(Eng.) and M.Sc.(Eng.) degrees in electronics engineering from Aligarh Muslim University (AMU), Aligarh, India, in 1991 and 1994, respectively, and the Ph.D. degree in electronics engineering from the University of Essex, Colchester, U.K., in 2003. In 1993, he joined the Department of Electronics Engineering, AMU, where he has been a Professor, since 2009. He has successfully completed several research projects funded by various agencies in India and U.K. He spent the summer of 2005 and 2006, as an Academic Visitor, at the University of Essex (funded by the Royal Society, U.K). He has authored or coauthored over 90 papers in refereed academic journals and international conference proceedings. His research interests include low-complexity image/video coding, video transmission over wireless networks, and biomedical image processing. He is also a Life Member of the Institution of Electronics and Telecommunication Engineers, India, and the Systems Society of India. He received the Commonwealth Scholarship to pursue the Ph.D. degree with the University of Essex. He also received the Research Award from the IBM J. T. Watson Research Laboratory, USA, for the best disruptive idea presented at the IEEE ICME 2002, held at EPFL, Switzerland.

MOHD HASAN (M'10–SM'13) received the B.Tech. degree in electronics engineering from Aligarh Muslim University (AMU), India, the M.Tech. degree in integrated electronics and circuits from the IIT Delhi, India, and the Ph.D. degree from The University of Edinburgh, U.K. He worked as a Visiting Postdoctoral Researcher on a project funded by the prestigious Royal Academy of Engineering, U.K., on low-power field programmable gate array architecture with the School of Engineering, The University of Edinburgh. He has been a Full Professor with AMU, since 2005. He has authored over 137 research papers in reputed journals and conference proceedings. His research interests include low-power VLSI design, nanoelectronics, spintronics, and battery-less electronics.

MARTIN REISSLEIN (S'96–M'98–SM'03–F'14) received the Ph.D. degree in systems engineering from the University of Pennsylvania, Philadelphia, in 1998. He is currently a Professor with the School of Electrical, Computer, and Energy Engineering, Arizona State University (ASU), Tempe. He also chairs the Steering Committee of the IEEE Transactions on Multimedia. He also serves as an Associate Editor for the IEEE Transactions on Mobile Computing, IEEE Transactions on Education, IEEE Access, and *Computer Networks*. He is also an Associate Editor-in-Chief of the IEEE Communications Surveys & Tutorials and a Co-Editor-in-Chief of *Optical Switching and Networking*.