Journal of Real-Time Image Processing (2020) 17:1421–1446
https://doi.org/10.1007/s11554-019-00901-x

ORIGINAL RESEARCH PAPER

A memory and area-efficient distributed arithmetic based modular VLSI architecture of 1D/2D reconfigurable 9/7 and 5/3 DWT filters for real-time image decomposition

Anirban Chakraborty[1] · **Ayan Banerjee**[1]

Received: 20 November 2018 / Accepted: 15 July 2019 / Published online: 25 July 2019

Abstract
In this article, we have proposed the internal architecture of a dedicated hardware for 1D/2D convolution-based 9/7 and 5/3 DWT filters, exploiting bit-parallel 'distributed arithmetic' (DA) to reduce the computation time of our proposed DWT design while retaining the area at a comparable level to other recent existing designs. Despite using memory extensive bit-parallel DA, we have successfully achieved 90% reduction in the memory size than that of the other notable architectures. Through our proposed architecture, both the 9/7 and 5/3 DWT filters can be realized with a selection input, mode. With the introduction of DA, we have incorporated pipelining and parallelism into our proposed convolution-based 1D/2D DWT architectures. We have reduced the area by 38.3% and memory requirement by 90% than that of the latest remarkable designs. The critical-path delay of our design is almost 50% than that of the other latest designs. We have successfully applied our prototype 2D design for real-time image decomposition. The quality of the architecture in case of real-time image decomposition is measured by 'peak signal-to-noise ratio' and 'computation time', where our proposed design outperforms other similar kind of software- and hardware-based implementations.

Keywords DWT · Distributed arithmetic · Memory efficient · Digital VLSI design · Parallelism · Image decomposition · PSNR

1 Introduction

Now-a-days, 'discrete wavelet transform' (DWT) has gained popularity among signal-processing engineers and researchers. This is mainly because of its capability of presenting time-domain as well as frequency-domain information of a signal simultaneously [1]. Until date, various kinds of techniques have been proposed for the implementation of DWT algorithm in terms of a dedicated hardware which facilitates real-time signal processing. All these hardware implementations of DWT can be broadly classified into two heads, viz., designs based on convolution-based DWT and designs based on lifting-based DWT [2, 3]. These can again be categorized into several sub-heads. Some of the notable and recent hardware implementations of DWT are presented in the subsequent paragraphs.

In the work of [4], the authors proposed two DWT architectures for 1D 9/7 DWT filters. Those architectures are based on bit-serial and bit-parallel 'distributed arithmetic' (DA). Though their bit-serial DA-based 1D DWT architecture of 9/7 filter requires less hardware compared to other contemporary works, the architecture is very slow and thereby ineligible to be used for real-time applications demanding high-frame-processing rate. On the other hand, their bit-parallel DA-based design is faster than the bit-serial one. However, bit-parallel DA-based design consumes almost eight times more memory resources than that of bit-serial DA-based design. Therefore, their designs are either area efficient and slow or memory consuming and fast. Their designs fail to address some serious design-related inefficiencies which restricts their design's applicability in real-time signal processing.

Mahajan et al. [5] proposed a DA-based 1D DWT architecture. They used 'carry save full adder' (CSFA) and carry save accumulator to reduce their design's critical-path delay.

✉ Anirban Chakraborty
chakraborty.sab@gmail.com

[1] Electronics and Telecommunication Engineering Department, Indian Institute of Engineering Science and Technology, Shibpur, Howrah, India

In this way, they achieved a design, whose maximum operating frequency is satisfactory. Their design is also claimed to be area efficient. However, the speed of operation of their proposed architecture is not at all up to the mark as their 'computation time' (CT) and 'cycle period' (CP) are proportionally related to the input data width 'L'. Therefore, for achieving a good data precision, their design stumbles in terms of speed and maximum operating frequency. Moreover, there is also scope of reducing memory requirement further.

The authors of the work [6] presented a three-level 2D DWT architecture for Daubechies 9/7 as well as bi-orthogonal filters. They successfully managed to reduce memory size to a great extent. They eliminated frame buffer and introduced line buffer in place of that. They also achieved high speed. However, all their advantages come at the cost of severe area overhead and power requirement. For example, for three-level 2D DWT decomposition of 512×512 image, their Daub-4 architecture requires 152 more multipliers and 114 more adders just to reduce 82414 memory words. Therefore, in spite of their praiseworthy memory efficiency, their designs suffer from significant area and power inefficiency.

Meher et al. [7] proposed 9/7 and 5/3 filter architectures. They proposed pipelined, without pipelined and also reconfigurable 9/7 and 5/3 architectures. They focused on reducing the area and memory size significantly. Though their design is area efficient and the operating speed is also satisfactory, there is further scope of reducing their CP and thereby enhancing the maximum operating frequency which is an important design aspect for real-time signal processing.

Lai et al. [8] presented a lifting-based 2D DWT architecture for JPEAG 2000 applications. They used two 1D DWT cores and one transposing unit for realizing 2D DWT architecture. For single-level DWT decomposition of an $N \times N$ image, $4N$ temporal memory is required in their proposed design which can be considered as a memory inefficient design. Apart from this, their design consumes large silicon area, as their design is not multiplier less. Due to existence of multipliers, their proposed architecture's CP is also larger compared to other recent works.

In the work of [9], an optimized lifting-based 1D DWT architecture is proposed based on embedded decimation technique. They further extended their designs for 2D. They used a vertical and two horizontal filter modules working in parallel and pipelined for realizing 2D DWT. Though they put marks of novelty in their proposed design strategy, their design falls far behind recent DWT architecture in terms of some important performance parameters such as area requirements, memory requirement, CP, etc.

Mohanty et al. [10] designed a lifting-based multi-level 2D DWT architecture. The authors presented several innovative design techniques and thereby managed to reduce frame buffer size, total silicon area requirement. They also focused on reducing the area delay product by appropriate portioning and scheduling of the computations performed in different decomposition levels. Their architecture is also configurable for area constrained and high-throughput application. However, while comparing their designs numerically in terms of various performance parameters, the quality of their design is found to be of low level in comparison with other similar kind of works. Their design consumes a large number of multipliers, adders, and ROM words. The maximum operating frequency of their design is also not satisfactory.

Tian et al. [11] proposed a VLSI architecture of multi input/multi output 2D DWT architecture based on lifting DWT. They focused on reducing the computation time. The main disadvantage of their design is that the factor by which they reduced the computation time is the same to the factor by which the corresponding area requirement and memory consumption increase. Therefore, basically, they achieved to reduce the computation time at the cost of proportional increase in area and memory requirements.

In the work of [12], three generic architectures are proposed for constructing 2D DWT architectures by the usage of line-based method for any hardware implementation such as convolution-based or lifting-based implementations. First, 1D architecture is based on single-level decomposition and it introduces internal line buffers and the optimization of the buffer size. The second architecture is for multi-level DWT and it is based on recursive pyramid algorithm. The third architecture is the combination of the first two architectures. In spite of all such novelties, their design's performance is not satisfactory in comparison with other works, as it consumes large number of multipliers and ROM words.

In the paper of [13], the authors proposed a systolic modular architecture for 2D DWT. The data processing has two distinct phases, viz., column processing (stage 1) and row processing (stage 2). Both of these stages are performed concurrently by mean of innovative data access techniques. In this way, using a novel folding techniques and a new data access scheme, the authors presented a transposition-free architecture. According to numerical analysis of their work's performance, their design requires a number of frame buffers and multipliers. The CP of their design is also large of the order of one multiplier delay.

Darji et al. [14] proposed a multiplier less 1D/2D DWT architecture based on lifting DWT. They used innovative z-scanning method to reduce transposing buffer size to 0. However, their temporal buffer size is directly proportional to input data points. Their adder requirement is also high. Their architecture's performance in case of real-time image decomposition can also be outshined by other recent methods.

Dillen et al. [15] proposed combined line-based 5/3, 9/7 DWT filter architecture for JPEG2000 standard. Their design is claimed to be area and memory efficient and of

high speed. However, their work had been proposed long time ago. In recent times, there are many works which surpasses their design's specifications. Their architecture can operate at the maximum operating frequency of 110 MHz which is not at all considered as high operating frequency now-a-days.

Banerjee et al. [16] proposed lifting-based 3D DWT architecture for video processing. Their architecture has features such as low power consumption, area efficiency, high operating speed, and high maximum operating frequency. However, there is scope of enhancing those features further and overtaking their design's performance numerically.

In the work of Hegde et al. [17], the authors proposed one lifting- and flipping-based DWT architecture which is memory and power efficient. They used area consumption, critical-path delay, and power consumption as the main performance metrics. They proposed 'look-up table' (LUT)-based multiplier to reduce area and critical-path delay. They developed the architecture using gate-level HDL language and provided the ASIC implementation details. By proposing LUT-based multiplier, they successfully achieved to reduce the critical-path delay and area consumption of their multiplier than any conventional popular multiplier. However, they did not completely omit multipliers from their designs. Therefore, their design's critical-path delay and power consumption are greater than any other multiplierless design. Moreover, LUT-based design uses a lot of registers or memory. Therefore, their design is also memory extensive.

In the paper [18], a multiplier less lifting-based 2D DWT architecture was proposed. In the same paper [18], a flipping-based 2D DWT architecture was also proposed. It was established in the paper [18] that the inherent low critical-path delay of flipping-based architecture could also be achieved using lifting-based DWT design. Both the designs were compared with other existing works to substantiate the contributions. Though the designs presented in [18] are claimed to reduce the critical-path delay significantly, still, the critical-path delays of both the lifting- and flipping-based architectures are much higher than that of any convolutional DWT architecture. Therefore, there exists enough scope of further enhancing the timing performance.

Having described certain latest and benchmark works in the domain of DWT architecture design, we are now focused on briefly mentioning some of the most recent works in the same domain. In the work [19], the authors presented 1D/2D DWT architectures using floating-point 'multiply and accumulator circuit' (MAC) units. The design was implemented using 45 nm CMOS technology. Though the validation and verification of the work [19] are praiseworthy, the performance can be improved further in terms of critical-path delay, CT, and memory requirement. The work presented in [20] deals with DA-based DWT architecture of LeGall 5/3 DWT filter. The work had been implemented onto Altera FPGA and the quality of the design was compared with other DWT-based works to prove its superiority. However, there remains ample scope of further optimizing the DWT architecture in terms of area consumption, power consumption, and speed of operation. The authors of the work [21] presented a 1D DWT architecture of LeGall 5/3 DWT filter using 'canonical sign digit' (CSD)-based DA. The CSD-based DA method helped the authors proposing hardware efficient DWT architecture requiring only 7 adders, a few shift registers and multiplexers. However, from Fig. 5 of the paper [21], it is evident that their clock period is 100 ns. This indicates that their design's operating frequency is only 10 MHz which is not at all acceptable for many real-time applications. Another significant and most recent DWT architecture was presented in the work of [22]. In the paper [22], a dual-memory controller-based 2D DWT architecture had been presented with a focus on real-time image processing. The memory requirement of the design had been claimed to be optimized to facilitate real-time image processing. The architecture was also implemented using FPGA. Though the design was claimed to be area and memory efficient, the delay of the design was 11.577 ns which restricted the design to be used in high-speed applications. The power consumption of the design had been reported as 0.306 W which could also be reduced with proper design methodology.

Apart from all the hardware-based DWT implementations stated above, we have also surveyed certain software-based DWT realizations using 'graphics processing unit' (GPU). The software-based DWT implementations are presented in [23, 24]. The work of [23] is a standard work focused on accelerating the DWT part of JPEG2000 using GPU-based computations. The authors used 'compute unified device architecture' (CUDA) platform to realize the DWT implementation on NVidia GeForce GTX 295. The authors of the paper [23] suggested a DWT algorithm with proper memory treatment for enhancing the timing performance. The DWT implementation can decompose a 512×512 image in 0.12 ms. The work of [24] proposed a recent 2D DWT implementation on NVidia GeForce GTX TITAN Black GPU. A register-based strategy was followed by the authors of the work [24] to propose their DWT algorithm which was claimed to be four times faster than other GPU-based software implementations of DWT.

In our paper, we have focused on implementing Daubechies 9/7 and 5/3 DWT filters [2] based on convolution-based DWT. We have extended our design for 2D after initially implementing it into 1D. We have exploited bit-parallel DA to make our design multiplierless. We have optimized our memory requirement which is less than what generally is needed for any bit-parallel DA-based 2D DWT architectures. We have made our design tunable for 9/7 and 5/3 filters by means of a mode

selection input. We have also tested our prototype design for multi-level DWT analysis in case of real-time image decomposition. The supremacy of our design is established in terms of various design-related parameters as well as in case of real-time image decomposition.

The rest of this paper is organized as follows. In Sect. 2, overview of DWT algorithm has been discussed. The mathematical formula for DA-based DWT is also mentioned in that section. Then, our proposed 1D/2D DWT architectures are presented in Sect. 3. Section 4 deals with the analysis of the comparison results and discussion. In this section, the output-decomposed images from our prototype proposed DWT filter are also provided. Finally, we conclude in Sect. 5.

2 Overview of DWT

In DWT, an input signal is passed through a series of high-pass filters to analyze the high frequencies and through a series of low-pass filters to analyze the low frequencies. The resolution and scale of the signal are changed by this filtering operation. The DWT analyzes the signal at different frequency bands with different resolutions by decomposing the signal into a coarse approximation. The decomposition of the signal into different frequency bands is obtained by passing the original signal $x[n]$ through a half-band high-pass filter $g[n]$ and a low-pass filter $h[n]$. After the filtering, half of the samples can be eliminated according to the Nyquist's rule, since the signal now has the highest frequency which is half of that of the original signal. This decomposition can be mathematically expressed as

$$y_{\mathrm{h}}[k] = \sum_{n} x[n] \cdot g[2k - n], \tag{1}$$

$$y_{\mathrm{l}}[k] = \sum_{n} x[n] \cdot h[2k - n], \tag{2}$$

where $y_{\mathrm{h}}[k]$ and $y_{\mathrm{l}}[k]$ are the outputs of the high-pass and low-pass filters, respectively, after sub-sampling by 2.

The DWT of the original signal is then obtained by concatenating all coefficients starting from the last level of decomposition (remaining two samples). The DWT will then have the same number of coefficients as the original signal. The reconstruction formula for the DWT is

$$x[n] = \sum_{k=-\infty}^{\infty} \big(y_{\mathrm{h}}[k] \cdot g[2k - n]\big) + \big(y_{\mathrm{l}}[k] \cdot h[2k - n]\big). \tag{3}$$

2.1 Mathematical background of DWT

The high-pass and low-pass filters of the Daubechies 9/7 wavelet filter [2] with filter coefficients $(h(i), -3 \le i \le 3;\ l(i), -4 \le i \le 4)$ and the high-pass and low-pass filters of the Daubechies 5/3 wavelet filter [3] with filter coefficients $(h'(i), -1 \le i \le 1;\ l'(i), -2 \le i \le 2)$ satisfy the symmetry property $(h(n) = h(N - n - 1))$, where N is the filter order.

The low-pass and high-pass components of Daubechies 9/7 wavelet filters, $v_{\mathrm{h}(9/7)}[n]$ and $v_{\mathrm{l}(9/7)}[n]$, respectively, are given by [7]

$$\begin{aligned} v_{\mathrm{h}(9/7)}[n] = {} & h[-3]x[n+3] + h[-2]x[n+2] + h[-1]x[n+1] \\ & + h[0]x[n] + h[1]x[n-1] + h[2]x[n-2] + h[3]x[n-3]. \end{aligned} \tag{4}$$

In addition

$$\begin{aligned} v_{l(9/7)}[n] = {} & l[-4]x[n+4] + l[-3]x[n+3] + l[-2]x[n+2] \\ & + l[-1]x[n+1] + l[0]x[n] + l[1]x[n-1] \\ & + l[2]x[n-2] + l[3]x[n-3] + l[4]x[n-4], \end{aligned} \tag{5}$$

where $x[n]$ is the input data stream. The 9/7 filter coefficients [7] are given in Table 1.

Using the symmetry property of the coefficients of 9/7 filters as shown in Table 1, the low-pass and high-pass DWT outputs can be expressed as

$$v_{\mathrm{h}(9/7)}[n] = h[0]s_n[0] + h[1]s_n[1] + h[2]s_n[2] + h[3]s_n[3]. \tag{6}$$

In addition

$$v_{\mathrm{l}(9/7)}[n] = l[0]s_n[0] + l[1]s_n[1] + l[2]s_n[2] + l[3]s_n[3] + l[4]s_n[4], \tag{7}$$

where

$$\begin{aligned} & s_n[0] = x[n],\ \ s_n[1] = x[n-1] + x[n+1], \\ & s_n[2] = x[n-2] + x[n+2], \\ & s_n[3] = x[n-3] + x[n+3], \\ & s_n[4] = x[n-4] + x[n+4]. \end{aligned} \tag{8}$$

Similarly, the low-pass and high-pass components of Daubechies 5/3 wavelet filters, $v_{\mathrm{h}(5/3)}$ and $v_{\mathrm{l}(5/3)}$, are expressed in the similar form of Eqs. 6 and 7 as

$$v_{\mathrm{h}(5/3)}[n] = h'[0]s_n[0] + h[1]s_n[1]. \tag{9}$$

In addition

$$v_{\mathrm{l}(5/3)}[n] = l'[0]s_n[0] + l[1]s_n[1] + l[2]s_n[2], \tag{10}$$

Table 1 High- and low-pass coefficients of 9/7 filter [7]

i	Low-pass filter, $l(i)$	High-pass filter, $h(i)$
0	0.6029490	1.1150870
±1	0.2668641	−0.5912718
±2	−0.0782233	−0.0575435
±3	−0.0168641	0.0912717
±4	0.0267487	

where the 5/3 filter coefficients [7] are given in Table 2.

Equations 6, 7 and 9, 10 may be rewritten in summation form as

$$v_{\mathrm{h}(9/7)}[n] = \sum_{i=0}^{3} h[i]s_n(i), \tag{11}$$

$$v_{\mathrm{l}(9/7)}[n] = \sum_{i=0}^{4} l[i]s_n(i), \tag{12}$$

$$v_{\mathrm{h}(5/3)}[n] = \sum_{i=0}^{1} h'[i]s_n(i), \tag{13}$$

$$v_{\mathrm{l}(5/3)}[n] = \sum_{i=0}^{2} l'[i]s_n(i). \tag{14}$$

For introducing DA, we have assumed the signal samples $(l(i), h(i), s_n(i))$ to be L-bit numbers in 2's complement representation. Each of the intermediate signals $s_n(i)$ for $0 \le i \le 4$ of Eq. 6 may thus be expressed in expanded form as

$$s_n(i) = -\big(s_n(i)\big)_0 + \sum_{l=1}^{L-1} 2^{-l}(s_n(i))_1, \tag{15}$$

Substituting Eqs. 15 in 11 and rearranging the order of summation, Eq. 11 can be rewritten as

$$v_{\mathrm{h}(9/7)}[n] = \sum_{l=1}^{L}\left[\sum_{i=0}^{3} h[i]\big(s_n(i)\big)_1\right]2^{-l} + \sum_{i=0}^{3} h[i]\big[-\big(s_n[i]\big)_1\big]. \tag{16}$$

Equation 16 may also be expressed as

$$v_{\mathrm{h}(9/7)} = \sum_{l=0}^{L-1} c_1\left[\sum_{i=0}^{3} h[i]\big(s_n(i)\big)_1\right]2^{-l}, \tag{17}$$

where $c_1 = 1$ for $1 \le l \le L-1$ and $c_1 = -1$ for $l = 0$.

Similarly, Eq. 12 can be rewritten as

$$v_{\mathrm{l}(9/7)} = \sum_{l=0}^{L-1} c_1\left[\sum_{i=0}^{4} l[i]\big(s_n(i)\big)_1\right]2^{-l}. \tag{18}$$

Table 2 High- and low-pass coefficients of 5/3 filter [7]

i	Low-pass filter, $l'(i)$	High-pass filter, $h'(i)$
0	0.75	1
1	0.25	−0.5
±2	−0.125	

In a similar way, Eqs. 13 and 14 can be rewritten as

$$v_{\mathrm{h}(5/3)} = \sum_{l=0}^{L-1} c_1\left[\sum_{i=0}^{1} h'[i]\big(s_n(i)\big)_1\right]2^{-l}, \tag{19}$$

$$v_{\mathrm{l}(5/3)} = \sum_{l=0}^{L-1} c_1\left[\sum_{i=0}^{2} l'[i]\big(s_n(i)\big)_1\right]2^{-l}. \tag{20}$$

The intermediate signals $s_n(i)_1, 0 \le i \le 4, 0 \le l \le L-1$ are fed as addresses to memory units, where the pre-computed partial inner product values are stored.

3 Proposed DWT architectures

We have realized Eqs. 17–20 in terms of a dedicated hardware which is capable of processing 1D/2D signals in real time. In this section, a memory efficient and low area architecture of 1D and 2D convolution-based 9/7 and 5/3 DWT filter is proposed. The novelty of our design is that we have exploited the inherent high-speed nature of convolution-based DWT approach while mitigating its disadvantages judiciously. The main disadvantages of convolution-based DWT approach are area inefficiency, incapability of being pipelined, and huge memory requirement. First, two of these demerits are dealt with using DA configured with bit-parallel mode. However, bit-parallel DA requires extensive memory. Therefore, it can be said that the solution for alleviating some of the disadvantages of convolution-based DWT design, in turn, exaggerating the other disadvantages of convolution-based DWT. Perhaps, that is the main reason as to why researchers focus on lifting-based design leaving the inherent advantages of convolution-based DWT unexplored. Another significant achievement of our design is that we have successfully reduced the huge memory requirement of bit-parallel DA-based convolutional DWT filters by our own innovative method and fully enjoyed the virtues of convolution-based DWT. Initially, we have designed a 1D DWT architecture exploiting bit-parallel DA. Then, we have extended the 1D DWT architecture for 2D using our proposed design strategy. We have efficiently introduced universality in our proposed architecture, so that the same architecture can be tuned for 9/7 as well as 5/3 DWT filters. We have mentioned our proposed design as dual mode in some places of rest of the paper. Our proposed 1D/2D architecture not only outperforms other convolution-based DWT architectures, but also it outshines most of the other lifting-based DWT architectures.

3.1 Design 1D DWT architecture

Our proposed convolution-based dual mode 1D DWT design is based on the essence of bit-parallel DA. Here, 'L' bits of the intermediate signal values, i.e., $s_n(i)$ for $0 \leq i \leq 4$, are used as address bits and applied in parallel to 'L' identical memory units, each of which containing the same data, i.e., the partial inner products. The partial products obtained from the memory unit are accumulated in the accumulator in the next clock cycle. After the accumulation has been done, we get the final low-pass and high-pass outputs. The advantage of this structure is its simplicity, modularity, and speed of operation. However, the main disadvantage of this type of design method is the size of the memory which increases linearly with the number of bits of intermediate signal value, i.e., $s_n(i)$. So far, all the works which have been proposed based on bit-parallel DA, suffer from extensive memory requirement. Though our proposed design is based on bit-parallel DA, we have successfully eradicated the extensive memory requirement by our own method. The block diagram of the overall VLSI architecture for 1D DWT is shown in Fig. 1. The dashed lines in Fig. 1 represent the decomposition of the structure into pipeline stages by incorporating registers to break the critical path. The overall block diagram of Fig. 1 is composed of three main sub-blocks, i.e., 'intermediate signal generation', 'memory', and 'pipelined and structured adders' (PSA). Each of these sub-blocks is again modularized into several units. All these sub-blocks and their constituent units are described in the following sub-sections.

3.1.1 'Intermediate signal generation' sub-block

The 'Intermediate Signal Generation' sub-block is comprised of a unit of shift registers, followed by an adder unit. The shift-register unit has nine 'parallel-in-parallel-out' (PIPO) shift registers. The input data sequence, i.e., $x[n]$ is fed to the shift-register unit. In every clock cycle, the outputs from the shift-register unit are fed to the adder unit to calculate the intermediate signal values $s_n(i)$, for $0 \leq i \leq 4$, according to Eq. 8. The adder unit consists of four L-bit 'parallel adders' (PA) and performs the addition operation concurrently such that the values of $s_n(i)$, for $0 \leq i \leq 4$, for a particular value of n are calculated in parallel. The internal architecture of the 'intermediate signal generation' sub-block is shown in Fig. 2.

3.1.2 'Memory' sub-block

In most of the bit-parallel DA-based architecture, the 'Memory' sub-block is used to store the partial products. In our case, 12 numbers of identical single-word ROMs

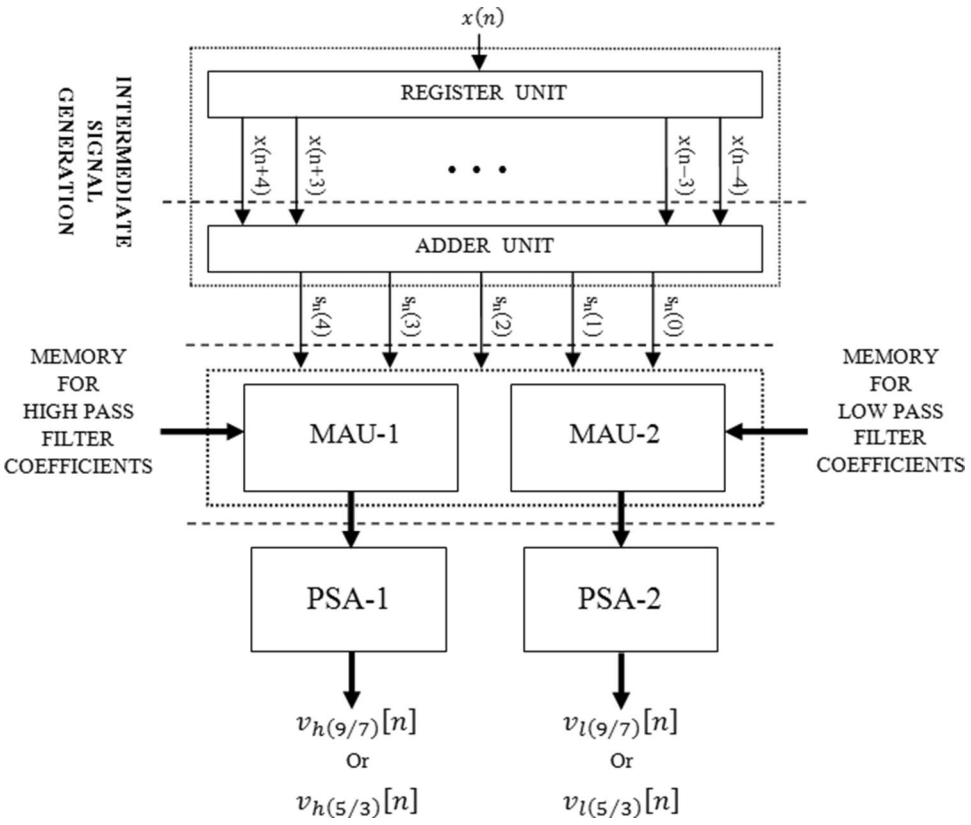


Fig. 1 Block diagram of the proposed bit-parallel DA-based 1D DWT architecture

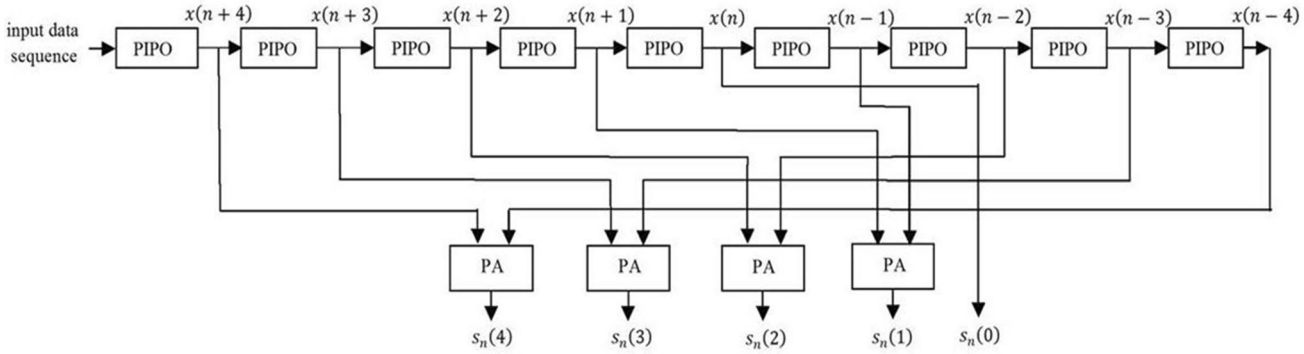


Fig. 2 Internal architecture of 'intermediate signal generation' sub-block

are used to store the partial inner products for computation of the high-pass filter output. The ROMs are numbered as ROM-0, ROM-1, ROM-2, and so on. ROM-0 to ROM-3 stores the possible partial inner products depending on the intermediate signals $s_n(0)$ and $s_n(1)$ for Daubechies 9/7 DWT filter. ROM-4 to ROM-7 stores the possible partial inner products depending on the intermediate signals $s_n(0)$ and $s_n(1)$ for Daubechies 5/3 DWT filter. Finally, ROM-8 to ROM-11 stores the possible partial inner products depending on the intermediate signals $s_n(2)$ and $s_n(3)$ for Daubechies 9/7 DWT filter. The organization of the ROMs and their respective data is shown in Fig. 3a.

Similarly, 12 numbers of identical single-word ROMs are used to store the partial inner- products for computation of the low-pass filter output. These ROMs are numbered as ROM-12, ROM-13, and so on. ROM-12 stores the partial product to be used for the signal $s_n(0)$, for Daubechies 9/7 DWT filter. ROM-13 to ROM-14 stores the possible partial inner products depending on the intermediate signals $s_n(1)$ and $s_n(2)$ for Daubechies 9/7 DWT filter. ROM-16 stores the partial product to be used for the signal $s_n(0)$, for Daubechies 5/3 DWT filter. ROM-17 to ROM-19 stores the possible partial inner products depending on the intermediate signals $s_n(1)$ and $s_n(2)$ for Daubechies 5/3 DWT filter. In addition, finally, ROM-20 to ROM-23 stores the possible partial inner products depending on the intermediate signals $s_n(3)$ and $s_n(4)$ for Daubechies 9/7 DWT filter. The organization of the ROMs and their respective data is shown in Fig. 3b.

Here, we can see that ROM-0 to ROM-3 and ROM-4 to ROM-7 stores the partial inner products for the high-pass filter output of 9/7 and 5/3 wavelet filters, respectively, for the same set of inter mediate signals, i.e., $s_n(0)$ and $s_n(1)$. Since we use only one mode, i.e., either 9/7 or 5/3 filter at a time, so the output port of the corresponding ROMs, for example, ROM-0 and ROM-4, ROM-1 and ROM-5, and so on, are passed through separate multiplexers, to select the required one. The select line of this multiplexer is denoted as 'mode'.

3.1.2.1 Multiplexing and addition unit (MAU) Separate multiplexing units are used in our proposed architecture for high-pass and low-pass filters, denoted by MAU-1 and MAU-2, respectively. The internal architecture of this 'multiplexing and addition unit' MAU is shown in Fig. 4, where the dashed line represents the decomposition of the structure into pipelined stages.

For high-pass filter, 'L' numbers of 'multiplexing subcells' (MUXSC) are used. They are denoted as MUXSC-1, MUXSC-2, and so on. The bit of a particular position of intermediate signals, i.e., $\left(s_n(i)\right)_1$, for $0 \le i \le 3$ is used as the select inputs of the MUXSC. The MUXSCs are organized, such that the $(l+1)$th MUXSC receives lth bit of the intermediate signals, i.e., $\left(s_n(i)\right)_1$, for $0 \le i \le 3$, such that the first MUXSC processes the MSBs of intermediate signals, i.e., $\left(s_n(i)\right)_0$ for $0 \le i \le 3$, and the Lth MUXSC, processes the LSBs of intermediate signals, i.e., $\left(s_n(i)\right)_L$ for $0 \le i \le 3$. Each of the MUXSCs contains two 4×1 multiplexers and one adder. The 4×1 multiplexers are denoted as MUX-1 and MUX-2.

The select lines of the MUX-1 are $\left(s_n(0)\right)_1$ and $\left(s_n(1)\right)_1$. This MUX-1 selects one of the four partial products corresponding to these signals depending on the combination of the select inputs. The select lines of the MUX-2 are $\left(s_n(2)\right)_1$ and $\left(s_n(3)\right)_1$. Therefore, MUX-2 selects one of the four partial products corresponding to these signals depending on the combination of the select inputs. The intermediate signals $s_n(0)$ and $s_n(1)$ are used for both 9/7 and 5/3 filters. Therefore, MUX-1 is active for the computation of both 9/7 as well as 5/3 DWT filters. However, the intermediate signals $s_n(2)$ and $s_n(3)$ are not used for 5/3 filter computation. Therefore, MUX-2 is activated only for computation using 9/7 wavelet filter. The 'chip select' (CS) input of MUX-2 is an active low input. Therefore, the 'mode' signal is connected to the 'CS' input of MUX-2. The 'mode' signal is '0' during the computation using 9/7 filter, enabling the MUX-2. However, during the computation using 5/3 filter; 'mode' signal is '1', disabling the MUX-2. In this way, though we

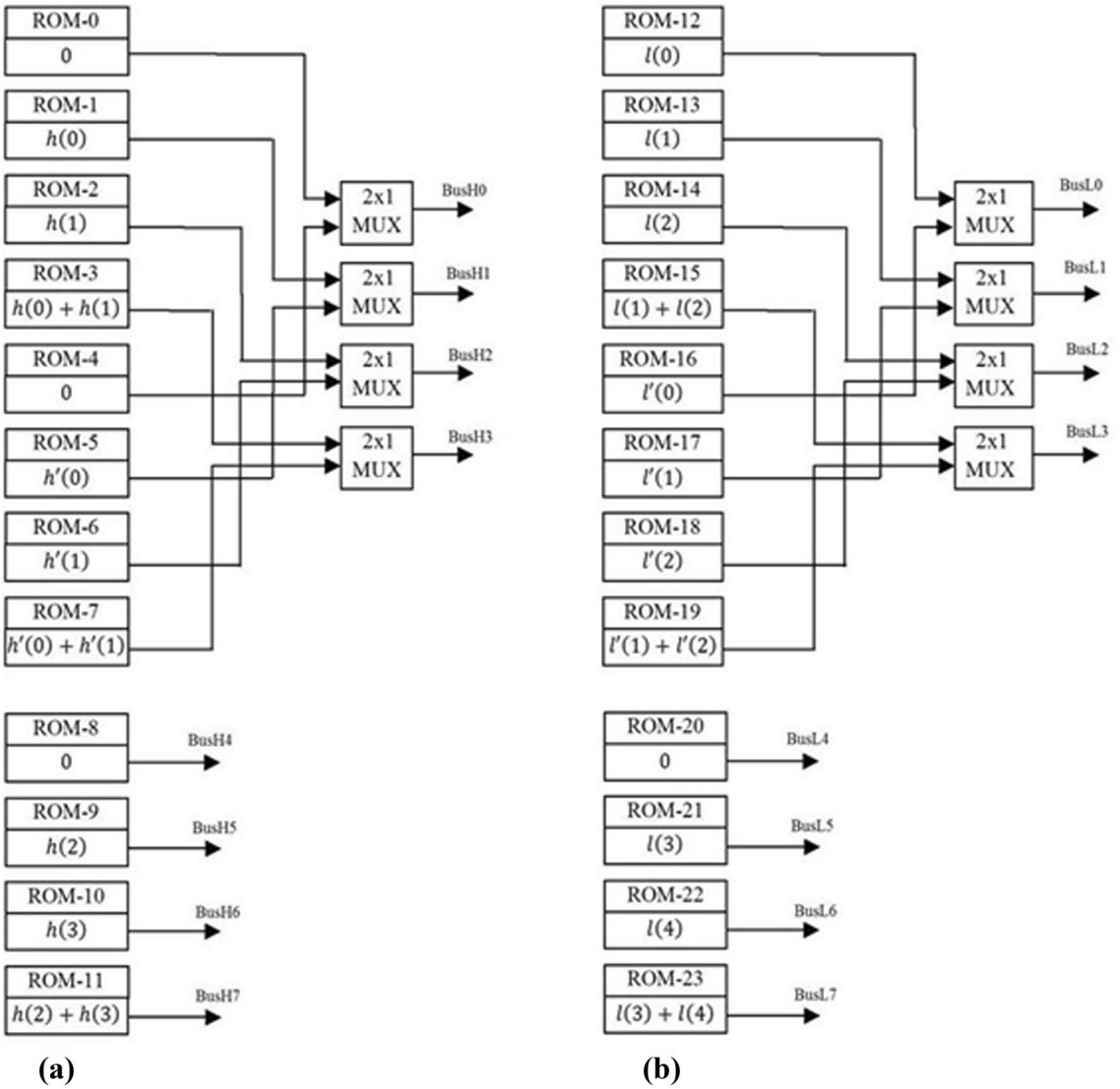


Fig. 3 Organization of the ROMs and their respective data; **a** for high-pass filter, **b** for low-pass filter

have proposed a tunable design for both 9/7 and 5/3 DWT filters, we have retained the power consumption as required by a single non-tunable design.

The output of the MUX-1 and MUX-2 is added using a simple two input adder before they are transferred to the PSA sub-block. The structure of the $(l+1)$th MUXSC for high-pass filter is shown in Fig. 5. The unit of MUX-1 and MUX-2 and the adder are connected in pipeline, as shown by the dashed line in Fig. 5.

Similarly, for low-pass filter, 'L' numbers of multiplexing sub-cells (MUXSC) are used and each sub-cell contains MUX-1 and MUX-2, the same as that for the high-pass filter. Here, the select lines of the MUX-1 are $\left(s_n(1)\right)_1$ and $\left(s_n(2)\right)_1$. In addition, the select lines of the MUX-2 are $\left(s_n(3)\right)_1$ and $\left(s_n(4)\right)_1$. Here, the signal $\left(s_n(0)\right)_1$ is not connected to the select lines of any of the two MUXes. It is connected to the select input of 'selector', a simple combinational logic block. The data bus from the ROM-12/ ROM-16 is fed to this selector. If $\left(s_n(0)\right)_1$ is '0', the output of this 'selector' is also '0'. In addition, when $\left(s_n(0)\right)_1$ is '1', the data, i.e., partial product that should be generated due to $\left(s_n(0)\right)_l$, is obtained at the output. The VLSI architecture of the $(l+1)$th MUXSC for low-pass filter is shown in Fig. 6.

The next part of this MAU is an adder. In case of high-pass filter, the data obtained from the two multiplexers, MUX-1 and MUX-2, are added using a two input adder. This adder adds two 'L' bit 2's complement numbers. In case of low-pass filter, the data obtained from the two multiplexers, MUX-1 and MUX-2, and from the 'selector'

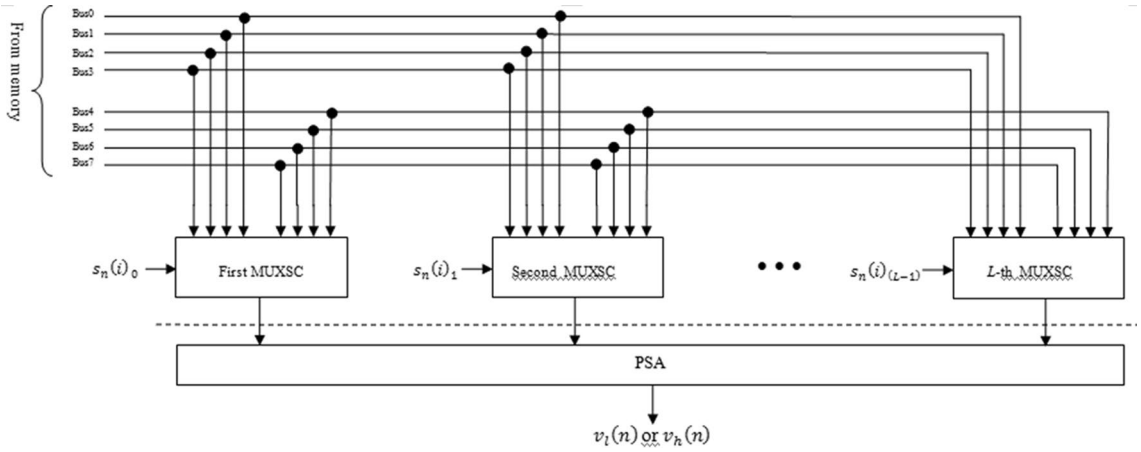


Fig. 4 Internal architecture of the MAU

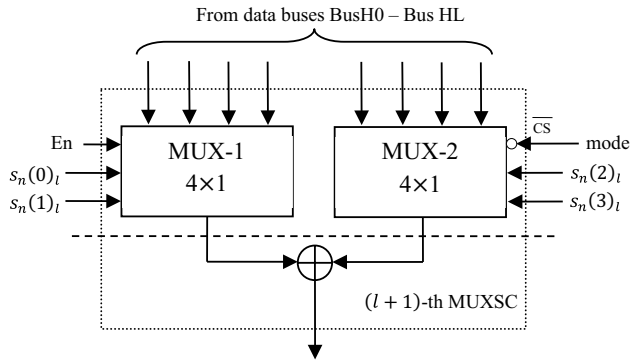


Fig. 5 Structure of $(l+1)$th MUXSC for high-pass filter

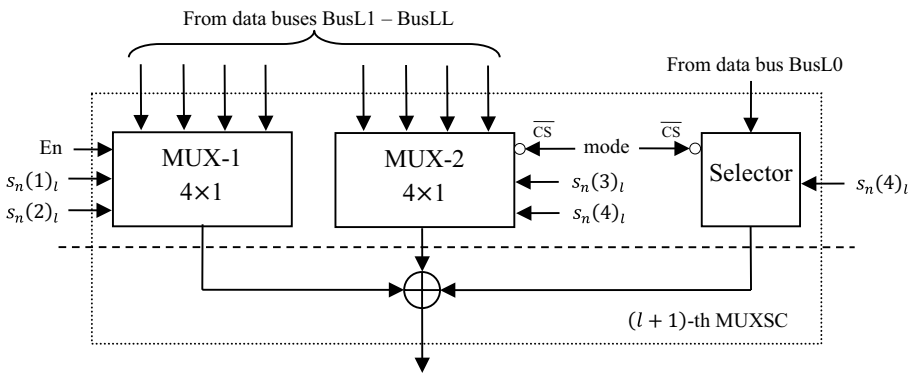
are added using a three input adder. This adder adds three 'L' bit 2's complement numbers. The output of the adder unit is delivered to the PSA sub-block

3.1.3 PSA sub-block

Each of the MUXSCs delivers the required partial inner products to the PSA. Since each of the MUXSCs corresponds to different positions of bits of the intermediate signals, the partial inner products have increasing weightage starting from LSB to MSB. Therefore, during accumulation, according to Eqs. 17–20, these partial inner products must be right shifted to the proper bit positions. Internal architecture of the PSA sub-block is shown in Fig. 7b. Accumulation is done in the adder tree comprising of $K = \log_2 L$ stages. The adders in each of the stages are connected with the adders of their adjacent stages in pipelined manner. Stage-1 consists of $\left(\left(L/2\right) - 1\right)$ numbers of 'adder cells' (ADDC) and one 'subtractor-cell' (SUBC). Stage-2 consists of $\left(L/4\right)$ numbers of ADDC, Stage-3 consists of $\left(L/8\right)$ numbers of ADDC, and so on. In addition, the last stage, i.e., Kth stage contains only one ADDC. Function of the ADDC in the kth pipelined stage is shown in Fig. 7a.

The ADDC in kth stage, where, $1 \le k \le K$, performs right-shift of the right operand by k bit positions, and add that with

Fig. 6 Structure of $(l+1)$th MUXSC for low-pass filter



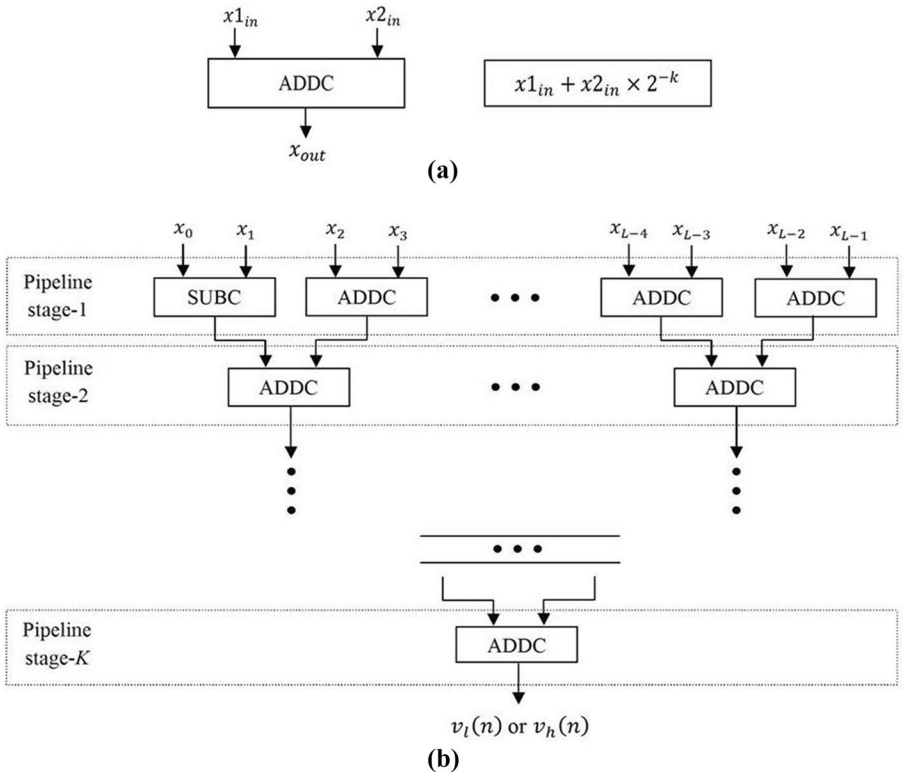


Fig. 7 **a** Function of the ADDC. **b** Internal architecture of a PSA sub-block

the left operand. That is, each of the adder cells in the first stage, right shifts their right operand by 1 bit position, and add that with the left operand. In addition, the adder cells in the second-stage right shifts their right operand by two-bit position, and add that with the left operand and so on. The subtractor cell in the first-stage performs the same operation as the adder cell, except that it performs subtraction of its operands instead of addition. The SUBC subtracts its left operand, i.e., the partial product obtained due to the sign bit, i.e., MSB, from that one, obtained due to the next MSB.

3.2 Design of 2D DWT architecture

Computation of 2D DWT of the elements of a 2D matrix is performed in two stages. In the first stage, 1D DWT is performed rowwise; the output of the first stage is stored in an intermediate buffer, i.e., a delay unit, to provide the necessary column delay. Then, 1D DWT of these intermediate output is performed in columnwise manner in the second stage. The proposed bit-parallel DA-based 2D DWT architecture is shown in Fig. 8.

The bit-parallel DA-based design consists of two stages. Input data sequence is entered to Div-1 in the first stage. Div-1 consists of register unit, adder unit to produce the intermediate signals, multiplexing and addition unit, to generate the partial inner products corresponding to the high-pass and low-pass 1-D DWT outputs, and finally the PSA to accumulate the inner products. Div-1 produces one low-pass $v_l(n)$ and one high-pass $v_h(n)$ intermediate output in every clock cycle, and delivers them to Div-2 and Div-3, respectively, in the second stage. Each of the Div-2 and Div-3 consists of delay unit, adder unit, multiplexing and addition unit, and finally, the PSA. Div-2 produces low-pass $v_{ll}(n)$ and high-pass $v_{lh}(n)$ outputs of the intermediate low-pass signal $v_l(n)$. Div-3 produces low-pass $v_{hl}(n)$ and high-pass $v_{hh}(n)$ outputs of the intermediate high-pass signal $v_h(n)$. Here, partial inner products for the high-pass filter coefficients are stored in nine single-word ROMs, and partial inner products for the high-pass filter coefficients are stored in 11 single-word ROMs. These ROMs are always 'active' and data buses from these ROMs are connected to the MAU of both the stages. The different parts of this design are discussed in the following sub-section.

3.2.1 Memory unit and data distribution

We have assumed an arbitrary 16×16 matrix as our input 2D sample data. Each element of the matrix can be represented by an 8-bit binary number. These elements are stored in 16 numbers of RAMs, each having 16 locations. All of these RAMs are single-port RAMs. Suppose the rows are represented as R_i, $0 \le i \le 15$, and the columns are represented as C_i, $0 \le i \le 15$. We have used four DWT channels to provide an optimization between the numbers of hardware required and throughput rate of the system.

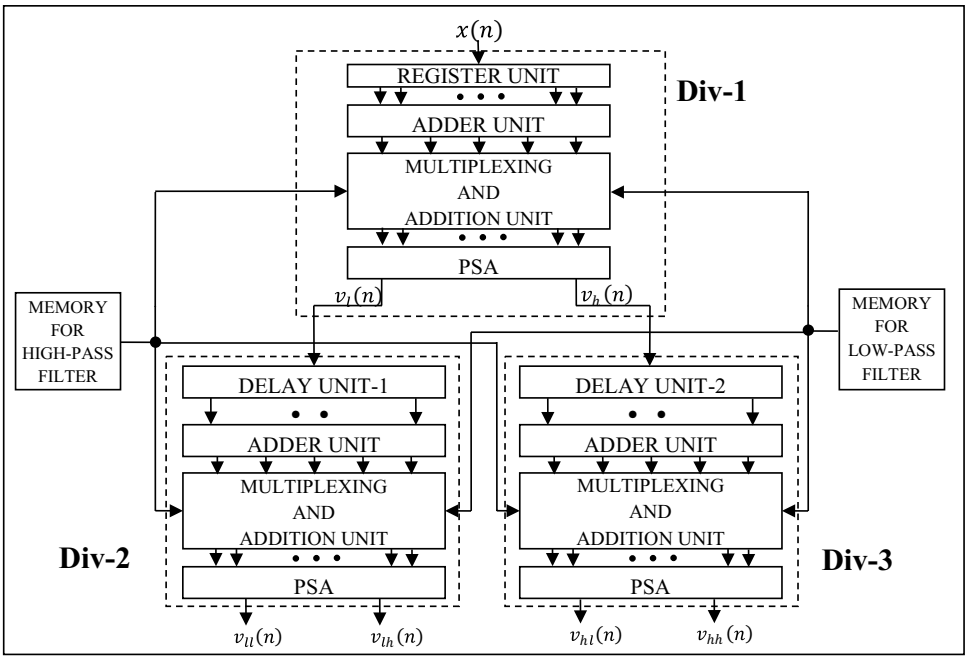


Fig. 8 Block diagram of the proposed bit-parallel 2D DWT architecture

Now, following Eqs. 4 and 5, we can see that, to compute 1D DWT for one point, we have to take four points from the left and four points from the right of that point from the input data sequence, according to the structure of the DWT filter, i.e., according to symmetry about the origin. Therefore, while calculating the 1D DWT of the first column C_0, taking the first element of the first row-first column R_0C_0, as the origin point, we have four elements in the right side of the origin, which are the elements of the first row of the second, third, fourth, and fifth columns, respectively. However, in the left side, we do not have any element. We can use zero padding, but multiplying filter coefficients with zero, gives zero, resulting in unnecessary loss of boundary information. Therefore, we have considered elements from the columns C_{12}, C_{13}, C_{14}, and C_{15}, which are stored in RAM-12, RAM-13, RAM-14, and RAM-15, respectively, as the elements of the left side of the origin, performing modulo N operation, which takes care of cyclic extension. This data padding can be done by inserting four extra RAMs in the left side of the original matrix. However, this will increase the memory requirement. The memory unit and data distribution is shown in Fig. 9.

Therefore, we have to put the elements from the columns C_0, C_2, C_4, $C_{6,}$ and so on as the center point of the filter. While processing, we have to process all the rows of the column taken as the center of the filter, then have to start the processing of a new column. While processing the column C_0, the set of columns required to be connected to the register unit are C_{12}, C_{13}, C_{14}, C_{15}, C_0, C_1, C_2, $C_{3,}$ and C_4. While processing the column C_2, the set of columns required to be connected to the register unit are C_{14}, C_{15}, C_0, C_1, C_2, C_3, C_4, C_5, and C_6. While processing the column C_4, the set of columns required to be connected to the register unit are C_0, C_1, C_2, C_3, C_4, C_5, C_6, C_7, and C_8. This procedure continues up to the processing of the column C_{14}. Therefore, we can see that each input of the register unit requires eight different sets of columns while processing the entire 16×16 matrix. Therefore, nine 8×1 multiplexers are used for each of the inputs to the register unit, to select the eight different columns during the processing.

3.2.2 Memory address generation unit

Now comes the one of the most important part of our design, the memory address generation unit. We have seen that data padding is required while processing the elements of the boundary columns to prevent the loss of boundary information. The same data padding is required while processing the elements of boundary rows during the computation in the second DWT unit. While processing the elements of different rows of the same column, processing of 1D dwt results of the first row requires four additional rows of data, which we can obtain from the processing of elements of the rows R_{12}, R_{13}, $R_{14,}$ and $R_{15,}$ respectively. Therefore, these rows must be processed before R_0. We can store these data points in four extra memory locations, but these will unnecessarily increase the memory size. Therefore, we have designed our memory address generation unit, so that it generates addresses to select the row in the order R_{12}, R_{13}, R_{14}, R_{15}, R_0, R_1, $R_{2,}$ and so on.

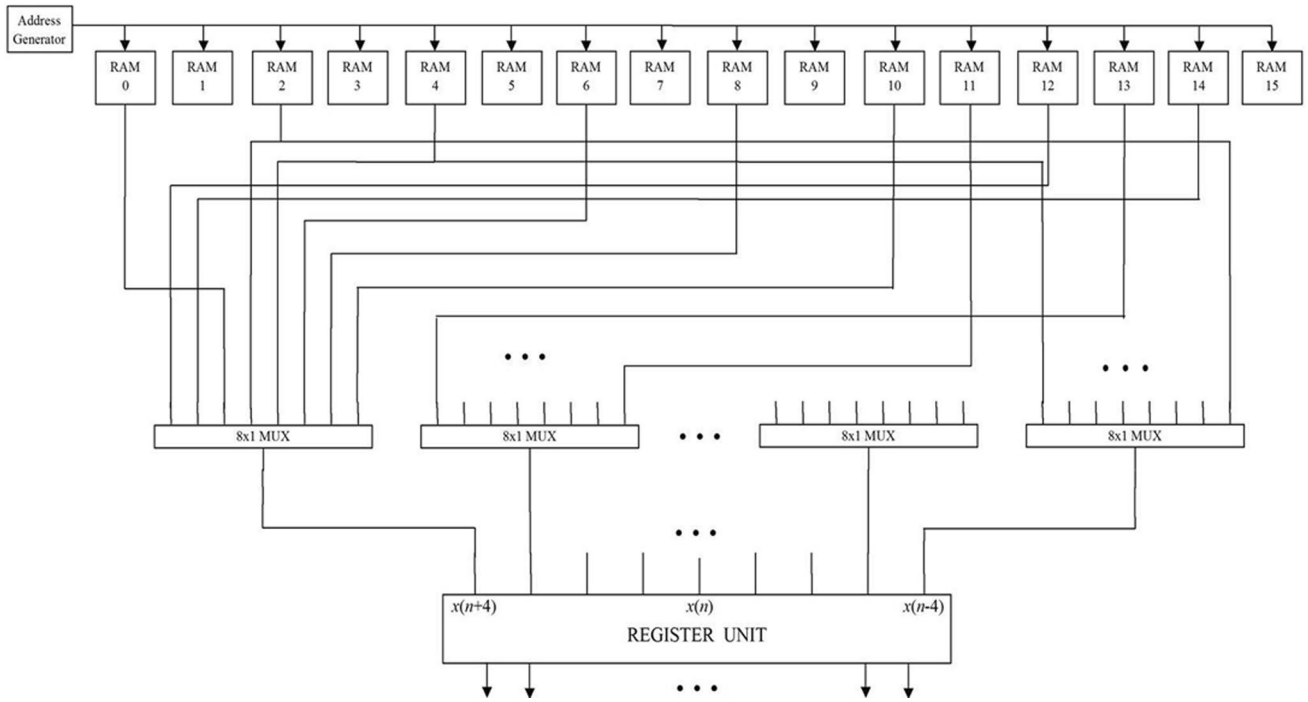


Fig. 9 Memory unit and data distribution system

This address generation unit is nothing, but a 4-bit asynchronous upcounter using positive edge triggered T-flip-flop, providing the states of '0000', '0001', up to '1111'. Then, it returns back to the initial state '0000' and counting starts again. According to our scheme, we have to start processing from 13th row, i.e., R_{12}. To select that row of each of the 16 RAMs, the output bits of the counter must be '1100' at the beginning of the operation. If we denote the output bits of the counter of the address generation block as Q_i, then Q_3 and Q_2 these two bits should be '1'. This can be done by making the asynchronous 'SET' input of T-FF_3 and T-FF_2 as '1' while making the asynchronous 'CLR' input of T-FF_0 and T-FF_1 as '1'. Thereafter, as the clock signal oscillates, the counter proceeds through its states.

Similar data padding is required while processing the 16th row, R_{15}. Now, the elements from R_0, R_1, R_2, and R_3 are to be padded. Therefore, during the processing of first set of columns, the address generator generates addresses, as '1100', '1101', '1110', and '1111' then proceeds through '0000' to '1111', again returns back to '0000', and proceeds up to '0011'. Then, the address bits are reset to '1100', and starts processing the second set of columns. To exemplify the proposed data-scanning/memory address generation procedure, mentioned above, we have taken an example of one arbitrary 16×16 2D data matrix, as shown in Fig. 10a. The data padded version of this 2D data matrix is presented in Fig. 10b.

In Fig. 10b, the original 16×16 2D data matrix is depicted within thick black borders. From Fig. 10b, we can see that data have been padded in all the four sides of the original 2D data matrix. In each side, (4×16) numbers of data have been padded. Therefore, total $4 \times (4 \times 16)$, i.e., 256 data have been padded incurring 256 extra memory locations. Likewise, it can undoubtedly be stated that total numbers of extra memory locations needed for storing the padded data are $[4 \times (4 \times N)] = 16N$ for any $(N \times N)$ data matrix. Therefore, in memory-based data-scanning techniques, the numbers of extra memory locations needed for storing the padded data increase linearly with the size of original 2D data matrix. In Fig. 10b, the elements inside red circles indicate two exemplary boundary elements that require data padding before being processed by DWT filter. For the element '2' inside the top left red circle, data from the last four rows of the first column of are required to be padded which are [1, 3–5], as shown in Fig. 10b. The processing window for the element '2' is highlighted in Fig. 10b by green rectangular section. Similarly, for the boundary element '3' inside the bottom left red circle, the processing window, which requires four data points to be padded from the top four rows of the first column, is highlighted using green rectangular section. For performing 1D DWT of the first column of the 2D data matrix, total data points needed to be scanned sequentially ranges from the extreme top element of the upper green rectangular box to the extreme bottom element of the lower green rectangular box, as shown

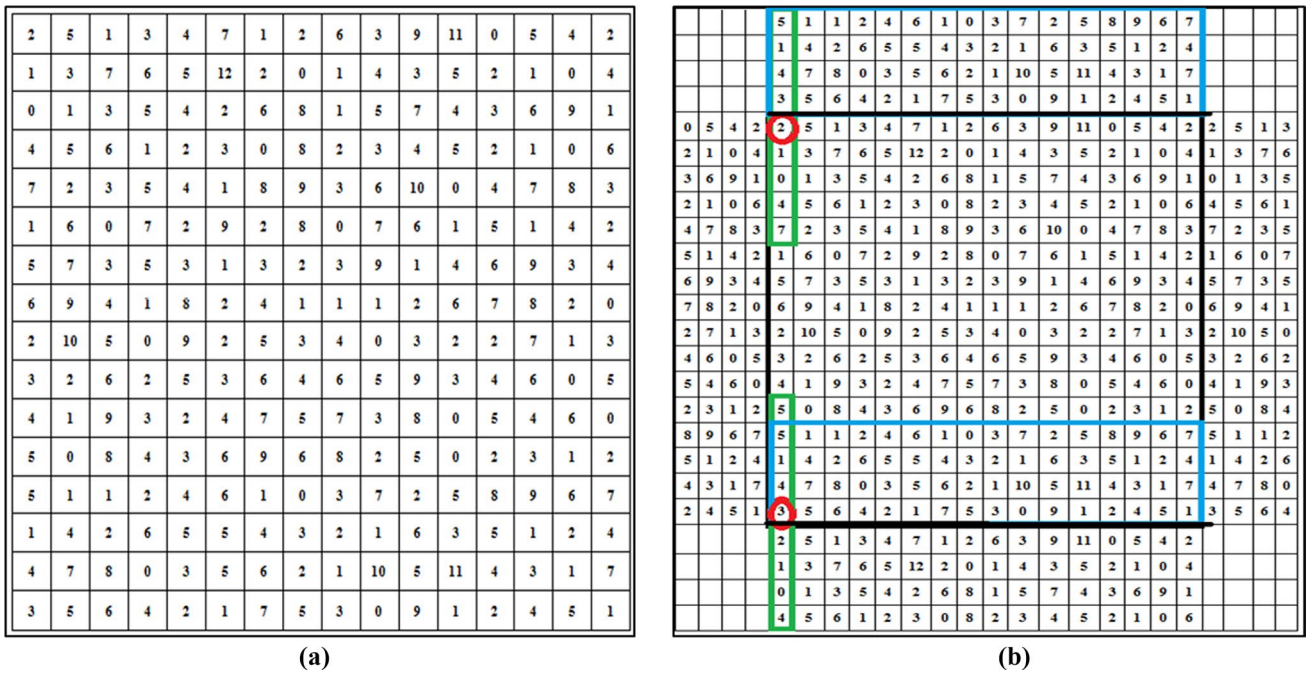


Fig. 10 **a** Exemplary 2D data matrix; **b** data padded matrix

in Fig. 10b. Therefore, total 24 memory locations need to be scanned sequentially. This 1D DWT of the first column can also be performed by scanning only 16 memory locations in a cyclic manner using our proposed method. Our proposed cyclic data-scanning technique for all row elements of the first column is depicted in Fig. 11a.

In Fig. 11a, two boundary elements of the first column are highlighted by green circle. The top boundary element requires four data from last four rows of the first column which are denoted by double underline in red. Similarly, the bottom boundary element requires four data from top four rows of the first column which are denoted by single underline in red. Now, instead of storing the padded data, we choose to scan the data cyclically in the clockwise direction from the starting point, as mentioned in Fig. 11a. The end point of the proposed cyclic scanning is also denoted in the same figure, i.e., Fig 11a.

The proposed cyclic data-scanning technique has been realized through our proposed memory address generation unit, the internal architecture of which is depicted in Fig. 11b. From Fig. 11b, it can be visualized that the proposed cyclic data scanning incurs some extra hardware

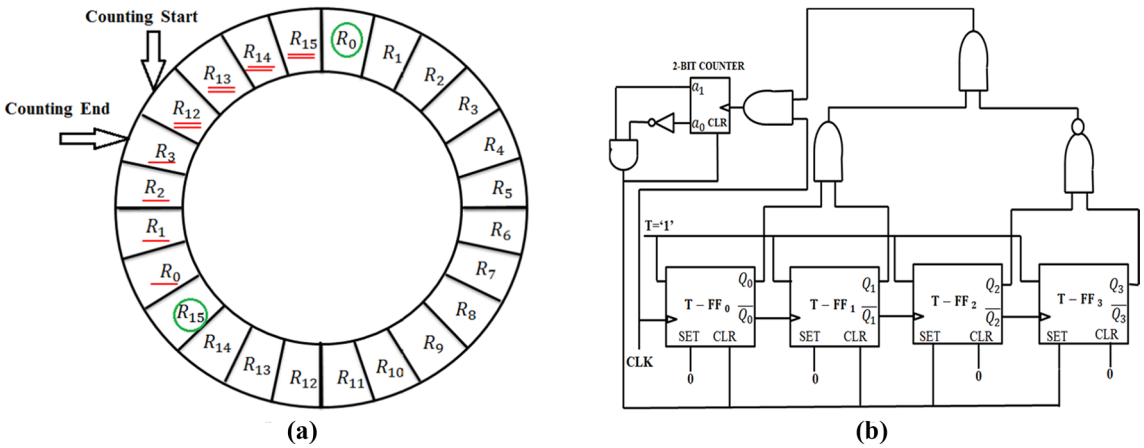


Fig. 11 **a** Proposed cyclic data-scanning technique; **b** proposed memory address generating unit

elements (4 AND gate, 1 NAND gate and a 2 bit up counter) along with a 4-bit asynchronous up counter which is required by any conventional memory address generation technique. Therefore, we have successfully reduced the memory size required by other conventional memory address generation techniques at the cost of few extra hardware elements. The comparative analysis between our proposed cyclic data-scanning-based memory address generation method and the conventional memory address generation method, in terms of hardware resources and extra memory locations required for data padding is presented in Table 3 for varying image size.

From Table 3, it is clear that by proposing the cyclic data-scanning-based memory address generation technique, we have saved huge memory resources, which increases linearly with image size, at the cost of some mere logic gates and one extra asynchronous up counter.

3.2.3 Delay unit

In the block diagram of Fig. 8, we can see that after the PSA unit of Div-1, two delay units are placed at each of the Div-2 and Div-3. PSA unit of Div-1 contains an accumulator which produces two outputs simultaneously ($v_l(n)$ and $v_h(n)$). These two outputs get transferred to the Div-2 and Div-3, respectively. As our proposed VLSI architecture of 2D DWT is a pipelined design, there should be two pipeline registers between Div-1, Div-2 and Div-1, Div-3 to break the critical path. Delay units inside Div-2 and Div-3 essentially serve the purpose of the pipeline registers. Div-2 and Div-3 process the intermediate results, i.e., low-pass and high-pass 1D DWT outputs ($v_l(n)$ and $v_h(n)$, respectively) produced by Div-1, respectively. The structure of the delay unit is the same, as shown in Fig. 12. The delay unit is nothing, but a combination of nine word-level 'parallel-in-parallel-out shift registers' (PIPOSR), connected in series. Processing of the first DWT unit requires 1 clock cycle. First-processed data of R_{12} is obtained after 1 clock pulse. Second-processed data from the next row, i.e., from $R_{13,}$ are obtained after the second clock pulse. Similarly, the processed data from R_{14} are obtained after the third clock pulse. These PIPOSRs are designed to get triggered at the negative edge of the first, second, third, etc. clock pulses. After nine clocks, all the rows required to be processed to start the second-level processing of R_0 is done. After initial nine clock pulses, nine

Table 3 Comparison of our proposed address generation method with conventional memory-based address generation method

Image size	Resources utilized	Proposed method	Conventional method
16×16	1. Logic gates	4 AND, 1 NAND	0
	2. Counter	One 2 bit and one 4 bit up counter	One 4 bit up counter
	3. Extra memory for data padding	0	256
256×256	1. Logic gates	4 AND, 1 NAND	0
	2. Counter	One 5 bit and one 8 bit up counter	One 8 bit up counter
	3. Extra memory for data padding	0	4096
512×512	1. Logic gates	4 AND, 1 NAND	0
	2. Counter	One 6 bit and one 9 bit up counter	One 9 bit up counter
	3. Extra memory for data padding	0	8192
1024×1024	1. Logic gates	4 AND, 1 NAND	0
	2. Counter	One 7 bit and one 10 bit up counter	One 10 bit up counter
	3. Extra memory for data padding	0	16,384

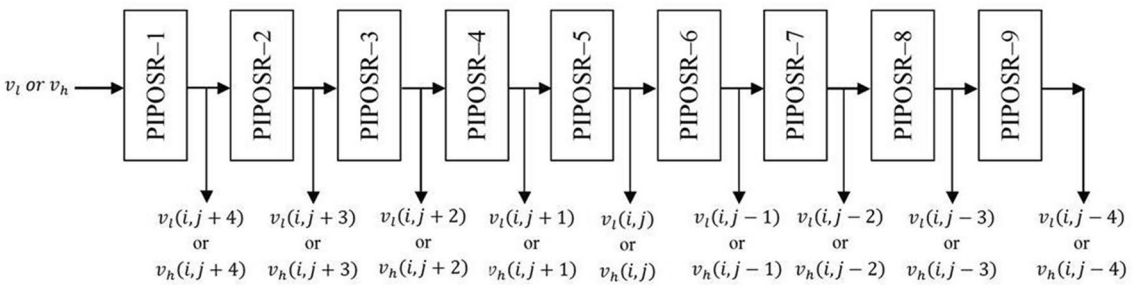


Fig. 12 Internal architecture of the delay unit

intermediate data elements, constituting a processing window, get stored in the nine PIPOSR. Then, these nine intermediate data are passed to the second DWT unit simultaneously in the immediate next clock cycle. Thereafter, in every clock pulses, the processing window gets shifted further by one element and the present nine constituent elements are passed to the second DWT unit simultaneously. Therefore, after an initial delay of 9 clock pulses, the second DWT unit gets a set of 9 elements in every clock pulses. Therefore, it can be inferred, from the above discussion, that the proposed delay units serve both the purpose of the pipeline registers by breaking the critical path between Div-1, Div-2, and Div-1, Div-3 as well as the purpose of the shift registers by storing and shifting intermediate data coming from Div-1, thus providing inputs to the subsequent unit. By catering such joint purposes, the proposed delay units successfully save a lot of hardware resources. It also eradicates the one clock pulse delay as required by a separate pipeline register. Two delay units are used for the low-pass and high-pass outputs of the first DWT unit.

4 Results and discussion

The proposed VLSI architecture of 1D/2D dual mode DWT filter has been initially designed as a prototype with 16 bit fixed-point binary data format in which 9 bits are used for integer part and 7 bits are used for fractional part of the data. At a later stage, for enhancing the data precision while retaining the area and memory overload at a reasonable level, we have extended the design for 32 bit fixed-point binary data format, where 24 bits and 8 bits are, respectively, used for representing the integer and fractional part of the data. Though we could have increased the data precision further by exploiting single precision (32 bit) floating-point binary representation (IEEE-754 format) [25], still, the proposed design has been implemented in a fractional fixed-point format instead of its floating-point counterpart for enhancing the performance in terms of hardware resources, latency, and power consumption. The range and precision of a fractional fixed-point binary representation mainly depend upon its fractional bit width and integer bit width, respectively. Suppose, in any arbitrary unsigned binary, fixed-point representation x bits are allocated for integer part and y bits are allocated for fractional part. Then, the range of values representable by that binary format will be 0 to $[(2^x - 1) + \sum_{i=1}^{y} 2^{-i}]$ and the corresponding precision will be 2^{-y}. Likewise, in our case, the 32 bit fixed-point representation has a range of 0 to $\left[(2^{24} - 1) + \sum_{i=1}^{8} 2^{-8}\right]$ and precision of 2^{-8}. If we want to increase the precision further, then more number of bits must be provided to represent the fractional part which, in turn, decreases the bit width representing integer part, thus reducing the range. Therefore, in fractional fixed-point binary format, there exists a trade-off between the range and the precision. On the other hand, in case of 32 bit normalized floating-point representation, as described in IEEE-754 format, the range is 0 to 3.4028×10^{38} and the precision is dynamic, i.e., changing. Using 32 bit normalized floating-point format, the value as low as 1.1755×10^{-38} can be represented. The characteristic of dynamic precision of 32 bit floating-point representation is represented in Fig. 13. In Fig. 13, the precision decreases in the direction of the positive y axis.

From the above discussion, it is clear that the range and precision, achieved using 32 bit normalized floating-point format, outshine the range and precision corresponds to our 32 bit fixed-point representation, where 24 bits are used to represent the integer part and remaining bits are utilized for denoting the fractional part. Therefore, naturally, it can be inferred that the accuracy of any VLSI architecture using 32 bit normalized floating-point arithmetic is more lucrative than that of its 32 bit fractional fixed-point counterpart. However, in terms of hardware resources, timing, and power efficiency, the 32 bit fractional fixed-point binary representation is far superior to that of the 32 bit normalized floating-point representation which is evident from Fig. 14a–c.

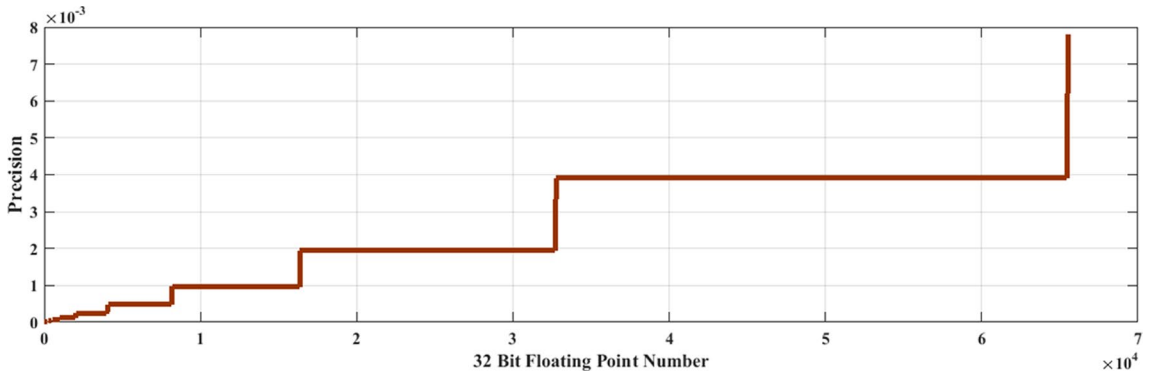


Fig. 13 Dynamic precision of 32 bit floating-point number in IEEE-754 format

In Fig. 14, we have displayed the comparative evaluations between the 32 bit fixed-point representation (24 bit integer and 8 bit fraction) and 32 bit floating-point representation (IEEE-754) by means of a single addition operation using a 32 bit adder. As the main basic arithmetic unit in our proposed 1D/2D DWT architecture is adder, we compared the performance of a 32 bit adder both using fixed-point and floating-point arithmetic. Other basic units in our proposed 1D/2D DWT design are multiplexers, PIPO registers, flip-flops, and memory which are less or not at all sensitive to fixed to floating-point transition as long as the total bit width remains the same. From the above discussion, it is clear that unless the high accuracy is of utmost importance, fixed-point arithmetic based VLSI architectures are preferred for any real-time application. DWT is being predominantly utilized in case of image compression or image fusion, where high accuracy is desirable, but not at the cost of other performance metrics such as area, latency, and power. Naturally, for any dedicated hardware of DWT, fixed-point arithmetic gets priority over its floating-point counterpart. Therefore, we have chosen fixed-point representation for designing our proposed 1D/2D DWT architecture.

After we have prototyped our proposed VLSI architecture of 1D/2D DWT in 32 bit fractional fixed-point representation, we have analytically compared the performance of our proposed 1D/2D DWT architecture with other notable works of similar kinds, as shown in Tables 4 and 5. After accomplishing the analytical evaluation, we have reconfigured our 2D DWT design using 32 bit normalized floating-point representation (IEEE-754) also. Then, we have implemented both the proposed fixed-point as well as floating-point 2D DWT architectures onto Spartan-6 and Zynq UltraScale + MPSoC FPGA. All the fixed- as well as floating-point-based implementation reports for both the FPGAs are reported and compared with that of the other notable works to justify the superiority of our 32 bit fixed-point-based 2D DWT design in terms of hardware complexity. Finally, we have applied our prototype 2D-DWT architectures (32 bit fixed-point and floating point) in case of real-time image decomposition. The performance of our design in case of real-time application is presented both quantitatively as well as qualitatively.

Therefore, basically, we have evaluated the performance of our prototype DWT architecture in three layers. First, we have analytically assessed the performance of our prototype 1D/2D DWT architectures configured in 32 bit fractional fixed-point representation. Second, our prototype 2D DWT architecture is reconfigured using 32 bit normalized floating-point representation (IEEE 754 format). Then, both the 32 bit fixed-point as well as floating-point-based proposed 2D DWT architectures are dumped onto Spartan-6 and Zynq UltraScale + MPSoC FPGAs for on-board testing which ensures the feasibility of our proposed 2D DWT architecture to be successfully utilized in case of real-time image decomposition. The results of on-board testing are compared with some latest and notable works of similar kind. Finally, we have also compared the performance of our prototype FPGA-based hardware implementations of 2D DWT with that of the 'general purpose processor' (GPP) and 'digital signal processor' (DSP)-based software realisation (in Python) of 2D DWT in case of real-time image decomposition. All these three layers of rigorous performance evaluations are detailed in the following.

4.1 Analytical evaluation

The proposed bit-parallel DA-based 1D DWT structure involves one intermediate signal generation sub-block; two multiplexing and addition unit, one for high-pass filter and the other for low-pass filter; two memory sub-blocks, one for storing high-pass filter coefficients and the other for storing low-pass filter coefficients; and two PSA. The intermediate signal generation sub-block consists of a shift-register unit, followed by an adder unit. As discussed in the previous section, the shift-register unit is comprised of nine PIPO, while the adder unit is comprised of four L-bit PA. Here, 12 single-word ROMs are used for storing the partial inner products for the high-pass filter coefficients and 12 identical single-word ROMs are used for storing the partial inner products for the low-pass filter coefficients. Therefore, total

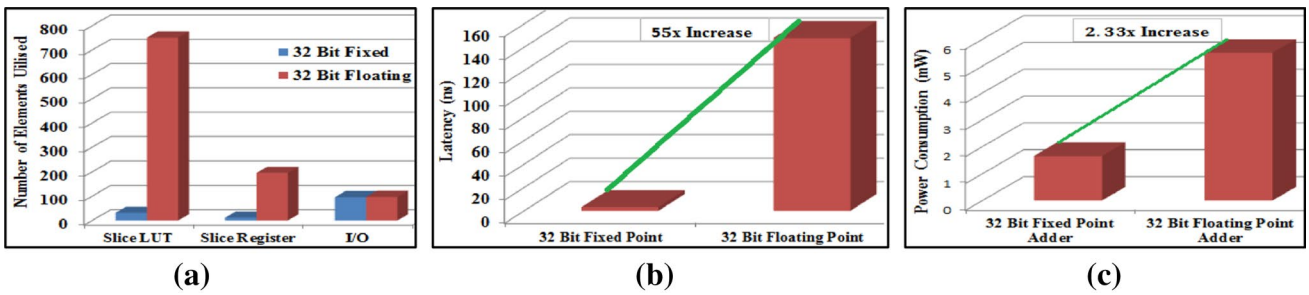


Fig. 14 32 bit fixed vs. floating-point arithmetic; **a** hardware resources; **b** latency; **c** power consumption

Table 4 Comparison of existing 1D DWT architectures with ours

Structures	ROM	Multiplier	Adder	MUX	REG	Cycle period	Cycle period (ns)	TR	CT	Mode
Mohanty et al. [4] (bit serial)	48	0	9	11	2	$L(T_{MR}+T_{FA})$	$32\times3.243=103.776$	$2/L$	$NL/2$	Single
Mohanty et al. [4] (bit parallel)	384	0	21	0	12	T_A	3.243	2	$N/2$	Single
Anurag et al. [5]	48	0	$2+(2L+7)$	4	10	$L[\max(T_{MR},T_{FA})]$	$(32\times6.3)=201.6$	2	$N/2$	Single
Mohanty et al. [6] (without pipelining)	0	9	14	0	0	$T_M+T_A+T_{FA}$	35	2	$N/2$	Single
Mohanty et al. [6] (with pipelining)	0	9	14	0	9	T_M	25	2	$N/2$	Single
Meher et al. (9/7 structure) [7] (without pipelining)	0	0	19	0	0	T_A+7T_{FA}	39	2	$N/2$	Single
Meher et al. (9/7 structure) [7] (with pipelining)	0	0	19	0	8	T_A+3T_{FA}	19	2	$N/2$	Single
Meher et al. (5/3 structure) [7] (without pipelining)	0	0	6	0	0	T_A+2T_{FA}	14	2	$N/2$	Single
Meher et al. (reconfigurable structure) (with pipelining) [7]	0	0	20	2	10	T_A+3T_{FA}	19	2	$N/2$	Dual
[20]	–	0	14	1	–	$4T_{FA}$	28	0.5	$2N$	Single
[21]	–	0	7	1	–	$3T_{FA}$	21	0.4	$5N/2$	Single
Propose bit-parallel (with pipelining)	20	0	18	32	12	T_A	2.54	2	$N/2$	Dual

'Mode' indicates which filter does the design use for computation of DWT coefficients, where, 'Single Mode' refers to that the design operates using only one, 9/7 or 5/3 wavelet filter, and 'Dual Mode' refers to that the design can operate using either 9/7 or 5/3 wavelet filter in the same architecture

TR throughput rate, *CT* computation time in number of clock cycles, *REG* number of data registers and pipeline registers, *L* input word length, *N* size of the input data, T_A addition time of *L*-bit-parallel adder, T_{FA} full adder time, T_{MR} memory read time, T_M time required for one multiplication operation

memory requirement for our proposed 1D DWT architecture is 24 words. All the stages, i.e., intermediate signal generation sub-block, adder unit, multiplexing unit, and PSA, are connected in pipeline. In addition, the adders in the PSA are connected in pipeline. Therefore, in this bit-parallel DA-based DWT design, duration of one bit-cycle is, $T_b = T_A$, and since in bit-parallel architecture, all of the L bits of the intermediate signal values are processed concurrently, so the duration of one cycle period is $T = T_b$, i.e., $T = T_A$. The performance comparison of our proposed 1D/2D DWT architectures with other existing architectures in terms of ROM words, number of adders, multipliers, registers, multiplexers, cycle period, throughput rate, and computation time are given in Tables 4 and 5.

Table 4 depicts the comparison of our proposed 1D DWT architecture with other existing works. From Table 4, it is clear that our proposed work outperforms other recent works in terms of various design-related parameters. Our bit-parallel DA-based 1D DWT design uses 32 numbers of multiplexers, whereas this number is 0 for [4], but our memory requirement, in number of words, is only 5% of that of the design in [4] and can provide the same throughput rate for both of the 9/7 and 5/3 wavelet filter. Our requirement of ROM words and multiplexers are more than that of the architecture presented in [7], but our critical-path delay, i.e., cycle period is nearly one-fourth than that of the design presented in [7]. Though the hardware complexity of the work of [20, 21] is less than that of our proposed 1D DWT architecture, the cycle period, throughput rate and computation time of our proposed architecture is far superior to the work of [20, 21]. The cycle period (in nano seconds), as shown in Table 4, indirectly indicates our proposed design's superiority in terms of maximum operating frequency over other existing works.

Table 5 provides the comparative analysis of our proposed 2D DWT architecture with other notable similar kind of designs. Our multiplierless bit-parallel DA-based 2D DWT design involves 160 numbers of multiplexers which is 82% more than that of [6], but our proposed architecture involves 49 numbers of adders, which is only 17% of that of [6], only 28% of that of [10] and 38.3% of that of [11]. Total memory requirement is much less than all of the designs listed above (only 4% and 6% of that of the designs in [6, 10], respectively. Critical-path delay, i.e., cycle period of this design is also much less (nearly half) than the multiplier-based designs [8–11], because the path delay of an adder is much less than that of any fastest multiplier. And last but not the least; our proposed design is made a dual mode one, unlike the other designs, as listed in Table 5.

To full proof the analytical evaluation of our proposed 32 bit fixed-point-based DWT design, we have compared the cycle period or critical-path delay of our proposed design with some most recent DWT designs [17–19] which

Table 5 Comparison of existing 2D DWT architectures with ours

Structures	DWT scheme	Multiplier	Adder	ROM words	Frame buffers	MUX/DEMUX	Cycle period	Cycle period (ns)	Mode
Lai [8] (bi-ortho)	LF	10	16	108	104	12	T_M+T_A	33.71	Single
Xiong [9] (bi-ortho)	LF	18	32	94	80	4	T_M+2T_A	39	Single
Mohanty [10] (bi-ortho)	LF	99	176	336	0	78	T_M+2T_A	34	Single
Tian [11] (bi-ortho)	LF	96	128	352	104	96	T_M+2T_A	37	Single
Huang [12] (Daub)	CV	16	16	44	80	4	T_M	27	Single
Meher [13] (Daub)	CV	16	12	6	128	16	T_M	24.59	Single
Mohanty [6] (bi-ortho)	CV	144	14	16	0	24	T_M	25	Single
Mohanty [6] (Daub)	CV	168	126	306	0	88	T_M	25	Single
[19]	CV	6 MACs	0	–	$\frac{5N^2}{4}$	$\frac{N^2}{2}+2$	T_{FMAC}	13.78	Single
Proposed bit-parallel (bi-ortho)	CV	0	49	20	0	160	T_A	2.54	Dual

'Mode' indicates which filter does the design use for computation of DWT coefficients, where, 'Single Mode' refers to that the design operates using only one, 9/7 or 5/3 wavelet filter, and 'Dual Mode' refers to that the design can operate using either 9/7 or 5/3 wavelet filter in the same architecture

LF lifting scheme, *CV* convolution scheme, *Bi-ortho* bi-orthogonal, *Daub* Daubechies, L input word length, $N\times N$ size of the input 2D data, T_A addition time of L-bit-parallel adder, T_{MR} memory read time, T_M time required for one multiplication operation, T_{FMAC} critical-path delay of floating-point MAC of [19]

specifically concentrated on reduction of the critical-path delay. The comparison details are listed in Table 6. From Table 6, it can be realized that the critical-path delay of our proposed design is much less than that of the designs of [17–19]. Naturally, our proposed design's maximum operating speed should also be much greater than those designs of [17–19]. Any real-time signal-processing application demands a dedicated VLSI architecture having low critical-path delay and high-frame-processing rate. Therefore, in this respect, our proposed design outshines the designs of [17–19]. In terms of other performance parameter, such as area consumption and power consumption, our design outperforms those designs [17–19] as our design is totally multiplierless and has not utilized any LUTs. We have not numerically showed the comparison of our proposed architecture with [17, 18] in terms of the area and power consumption, because they presented their results in terms of ASIC implementation, while we have implemented our design using FPGA.

Table 6 Comparison of critical-path delay

Architectures	Critical-path delay	Critical-path delay (ns)	Design type
Design 1 [17]	$T_{\mathrm{LUT}}+\left(\frac{W}{4}+3\right)\cdot T_{\mathrm{FA}}+T_{\mathrm{A}}$	53 (for $W=32$)	Lifting
Design 2 [17]	$T_{\mathrm{LUT}}+\left(\frac{W}{4}-1\right)\cdot T_{\mathrm{FA}}+2\cdot T_{\mathrm{A}}$	42 (for $W=32$)	Lifting
[17]	$T_{\mathrm{LUT}}+\left(\frac{W}{4}\right)\cdot T_{\mathrm{FA}}+T_{\mathrm{A}}$	39 (for $W=32$)	Flipping
[18]	$T_{\mathrm{BE}}+T_{\mathrm{P}}+\left(\frac{W}{2}-1\right)\cdot T_{\mathrm{FA}}+T_{\mathrm{A}}$	95 (for $W=32$)	Flipping
[18]	$T_{BE}+T_{P}+\left(\frac{W}{2}+1\right).T_{FA}+T_{A}$	103 (for $W=32$)	Lifting
[19]	T_{FMAC}	13.78	Convolution
Proposed	T_{A}	2.54	Convolution

T_{BE} delay of Booth's encoder, T_P delay of partial product generation, T_{FA} delay of full adder, T_A delay of an adder, T_{LUT} delay of LUT (2^5 locations for lifting-based design, 2^4 locations for flipping-based design), T_{FMAC} delay of an MAC unit of [19], W word length

4.2 Performance in real-time image decomposition

After performing the post-routing simulation using Xilinx ISE 14.7, we have implemented our proposed 2D DWT design onto Spartan-6 series FPGA, XC6LS45T. We have also simulated our proposed architecture using Xilinx Vivado 18.2 and implemented the prototype 2D DWT design onto Zynq UltraScale + MPSoC series FPGA, XCZU7EV-2FFVC1156. For implementing our prototype 2D DWT architecture onto both the FPGAs, we have connected the individual FPGA board with host 'Personal Computer' (PC) by means of a JTAG cable. We have configured both the FPGAs in boundary scan mode configuration in which the bit file of the proposed 2D DWT architecture is directly transferred from host PC to FPGA in a serial fashion. The maximum data transfer rate through the JTAG cable is only 66 MHz. This rate is inconsequential, because the bit file of the proposed design is required to be downloaded in the FPGA only once as long as the power supply is in the 'ON' mode. After the FPGAs get configured in accordance with the bit file of our proposed design, we have applied our prototype 2D DWT design implemented on both the FPGAs, mentioned above, to real-time image decomposition. The test setup for supplying input images to the individual FPGA and displaying the output-decomposed image from the FPGA is detailed later in this section. Here, we have estimated the quality of our proposed 2D DWT architecture by comparing it with that of the other existing hardware-based works, in case of real-time image decomposition. The comparison is performed for the proposed architectures configured in both the 32 bit fractional fixed-point and floating-point formats (IEEE-754 format). We have also reported the comparative results of the performance of our prototype 2D DWT architecture implemented onto both the Spartan-6 and Zynq UltraScale + MPSoC series FPGA with that of the software-based implementations of 2D DWT using GPP and DSP. This comparative performance evaluation with software-based 2D DWT realisation is also performed and reported for both the 32 bit fractional fixed-point representation (24 bit for integer part and 8 bit for fractional part) as well as 32 bit floating-point representation (IEEE-754 format).

4.2.1 Experimental setup

We have implemented our proposed design onto two different FPGAs, as discussed above. Therefore, we have two different experimental setups. Moreover, for comparing the performance of our proposed VLSI architecture with its software counterpart, we have run the Python program of 2D DWT on GPP and DSP. For GPP-based implementation, we have used the inbuilt quad-core Arm Cortex-A53 processor of Zynq UltraScale + MPSoC. For DSP-based implementation, we have used inbuilt DSP unit of STM32F446RE microcontroller. For floating-point-based software implementations, we make use of the 'floating-point processing unit' (FPU) of the above-mentioned GPP and DSP. The experimental setups for the FPGA-based implementations are depicted in Fig. 15

In Fig. 15a, the experimental setup using Spartan-6 FPGA (XC6LS45T) is shown. After the configuration of Spartan-6 FPGA has been accomplished using JTAG cable, we have focussed on connecting the FPGA chip with the host PC. Our aim is to verify the usefulness of our proposed design in terms of a tangible real-time application. For that, we have planned to utilise the prototype hardware of the 2D DWT in real-time image decomposition. We have placed the test images, i.e. ,images to be decomposed, in MATLAB bin directory of host PC. Then, we have established the connection between the host PC and the external RAM of the Spartan-6 FPGA board with the help of a high-speed ethernet connection. The speed of this connection can be approximately up to 1Gbps. We also configure the inbuilt GTP trans-receivers and PCI Express interface of the FPGA to achieve a high-speed data transfer between external RAM and FPGA block RAM. Fig 15a represents a pictorial description of data transfer between host PC, external RAM, and FPGA. In general, the size of the real-life images surpasses the maximum size of block

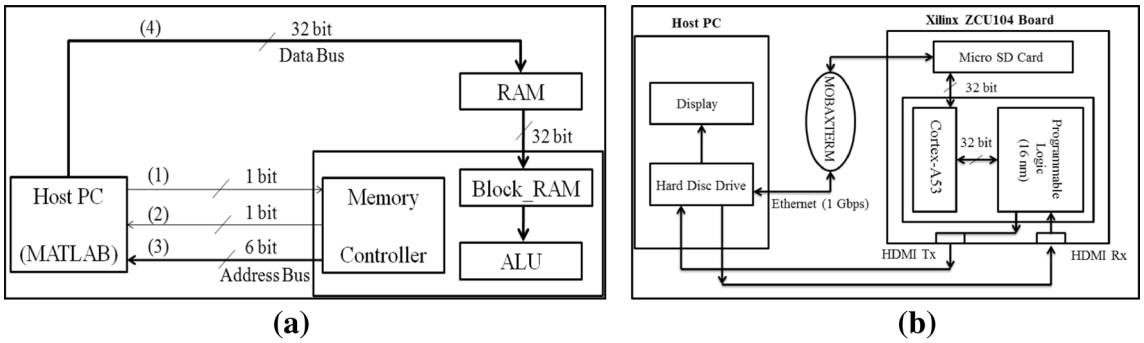


Fig. 15 Experimental setups. **a** for Spartan-6 FPGA XC6LS45T; **b** for Zynq UltraScale + MPSoC FPGA

RAM resources of the FPGA. To mitigate the problem, we tune the inbuilt interfaces of the FPGA, so that we may use the external RAM block of the FPGA board also. The sequence of operation for this external RAM-based data transfer is indicated by numerals in Fig. 15a inside bracket; '()'. The sequences are described below using the sequence number given in Fig. 15a. (1) Seeking permission for starting data transfer from host PC to external RAM. (2) A flag is sent from memory controller block of FPGA to host PC indicating the permission of data transfer. (3) 6-bit starting address of the external RAM, from where the data need to be stored, is sent by Memory controller block to host PC. (4) Upon receiving the starting address, host PC starts sending 32 bit data to external RAM through 32 bit data bus.

In Fig. 15b, the experimental setup using Zynq UltraScale + MPSoC FPGA is presented. The same setup is utilized for implementing Python-based 2D DWT program in the Arm Cortex-A53 GPP of the Zynq UltraScale + MPSoC board. However, the connection establishment and data transfer procedures between host PC and the Zynq UltraScale + MPSoC are different for both the FPGA-based hardware implementation and GPP-based software implementation. Both of these procedures are explained in the following.

- Connection to GPP in Zynq UltraScale + MPSoC board from host PC.

(a) Debian linux-based 'operating system' (OS) is loaded in micro SD card and put it on micro SD card slot of Zynq UltraScale + MPSoC evaluation board.
(b) JTAG cable is connected between the Zynq UltraScale + MPSoC evaluation board and host PC for accessing the board through terminal in host PC via UART (serial) protocol.
(c) The Zynq UltraScale + MPSoC board is booted with Debian linux using appropriate command through terminal. Through this step, Debian linux-based OS is essentially installed onto Arm Cortex A-53 GPP.
(d) 1Gbps Ethernet connection is established between the Zynq UltraScale + MPSoC board and host PC by connecting Ethernet cable and providing appropriate command through terminal in host PC.
(e) MOBAXTERM software is utilized to send the test images from host PC to micro SD card in Zynq UltraScale + MPSoC board via 'file transfer protocol' (FTP).
(f) Then, the Python file containing 2D DWT program is sent to micro SD card from host PC along with a C++ programing file containing instructions of running the Python file onto Arm Cortex A-53 processor.
(g) The C++ file also contains instructions for providing the test images to the inputs of the Arm Cortex-A53 GPP from microSD card and for taking the outputs (decomposed images) from the GPP back to the micro SD card.
(h) After that, the output-decomposed images are sent back from micro SD card to host PC using command from terminal.

- Connection to FPGA in Zynq UltraScale + MPSoC board from host PC.

1. Steps A–E stated above are performed invariably.
2. The bit file generated by Xilinx Vivado 18.2 for the proposed 2D DWT architecture is transferred to the FPGA of the Zynq UltraScale + MPSoC by the JTAG connection.
3. C++ programming file is sent to the micro SD card. The C++ file contains instructions for providing the test images to the inputs of the Zynq UltraScale + MPSoC FPGA from microSD card and for taking the outputs from the FPGA back to the micro SD card.
4. Decomposed images are sent back to the host PC from micro SD card.

4.2.2 Performance evaluation and comparison

After the Spartan-6 and Zynq UltraScale + MPSoC FPGAs have been configured with the bit file of our proposed 2D DWT design, test images are sent to the inputs of the FPGAs through the procedures mentioned above. Those test images get decomposed by the implemented designs on both the FPGAs. The outputs from both the FPGAs are taken back to the host PC by the procedures stated in the above sub-section. Then the DWT coefficients of the decomposed images are used to reconstruct the test images using Python coding on PyCharm IDE in host PC. Obviously, the reconstructed test images and the original test images will not be exactly same, as quantization noise and binary data truncation error will be added, while processing by the proposed VLSI architectures implemented on the FPGAs. The accuracy of our proposed implemented designs are estimated in terms of 'peak signal-to-noise ratio' (PSNR) [26] between restored and original test images following equation:

$$\mathrm{PSNR} = 20\log_{10}\frac{\mathrm{MAX}_z}{\sqrt{\frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}|z(x,y)-\hat{z}(x,y)|^2}}\ \mathrm{dB}, \tag{21}$$

where z and $\hat{z}$ are the original test image and the reconstructed test image, respectively. (x, y) denotes the image coordinates and $(M \times N)$ is the image size.

This PSNR calculation is also performed in host PC using Python on PyCharm IDE. In Table 7, we have compared the accuracy of our prototype 2D DWT designs implemented using both Spartan-6 and Zynq UltraScale + MPSoC, in terms of PSNR, with the latest notable existing designs of [14, 19, 22] for real-time image decomposition for some popular test images (256×256). This performance comparison is performed using both 32 bit fixed point (24 bit for the integer part and 8 bit for the fractional part) as well as 32 bit normalized floating-point representation (IEEE-754 format).

From Table 7, it is seen that our proposed 2D DWT architectures (both in 32 bit fixed and floating-point format) implemented on Spartan-6 and Zynq UltraScale + MPSoC FPGAs produce higher PSNR in comparison to the notable and latest works of [14, 19, 22]. The lowest PSNR obtained for our proposed architecture configured in 32 bit fixed-point format and implemented on Spartan-6 FPGA is 39.48 which is still 16.40% higher than the work of [14], 6.27% higher than the work of [19] and 6.64% higher than the work of [22]. The original test images (256×256), decomposed images by proposed 2D DWT design in 32 bit fixed-point format implemented on Spartan-6 FPGA and also the reconstructed images are shown in Fig. 16.

After implementing our proposed 2D DWT architecture, we have also compared our prototype design with other prominent existing designs in terms of various design-related parameters which are obtained directly from FPGA resource utilization report. We have also used the implemented 2D DWT designs for real-time three-level image decomposition of 512×512 sized test images and then compared the results with that of the other significant designs. This comparison results are presented in a tabular form in Table 8. Table 9 describes the comparison in terms of resource utilization details of FPGA.

Therefore, we have vigorously compared the performance of our proposed 1D as well as 2D DWT architectures with other prominent designs of similar kind. We have analyzed the quality of our proposed design in terms of its performance in real-time image decomposition. The resource utilization details of our proposed work are also taken into consideration after implementing it onto Spartan 6 and Zynq UltraScale + MPSoC series FPGA. We have presented the numerical results of comparison in tabular format and we have also provided the outputs of real-time image decomposition operation for visual inspection. In this way, we have extensively evaluated the working quality of our proposed architecture for both the 32 bit fixed-point and floating-point representations and established our prototype 32 bit fixed-point-based 2D DWT design to be the most suitable for real-time applications.

Having compared the performance of our proposed VLSI architecture of 2D DWT in both 32 bit fixed and floating-point configuration with other similar hardware implementations of 2D DWT, we are now focused on comparing the performance with that of the 32 bit fixed-point and floating-point-based software implementations of 2D DWT. For that, we have used PyWavelets package of Python to decompose test images. The pywt.dwt2() function of the PyWavelet package is modified using 32 bit floating-point-based arithmetic (IEEE-754 format) for the implementation of 32 bit floating-point-based 2D DWT. We also have converted the data type of 512×512 sized test image 'Lena' into 32 bit floating-point format before being decomposed using floating-point arithmetic based 2D DWT. Then, we have run the Python code of 2D DWT in 32 bit fixed-point as well as floating-point format in 'central processing unit' (CPU), GPP and DSP for decomposing the 512×512 sized test image 'Lena'. We have run the python code in inter(R) Core(TM) i3-2310M CPU operating at 2.10 GHz, in the quad-core Arm Cortex-A53 GPP operating at 1.5 GHz inside Zynq UltraScale + MPSoC evaluation board and also in DSP units of STM32F446RE Microcontroller operating at maximum 180 MHz. The comparative analysis of the performance of these software-based implementations with that of our proposed VLSI architectures in terms of 'computation time' (CT) and PSNR is presented in Table 10 as well as in Fig. 17.

Table 7 Real-time performance comparison of proposed 2D DWT architecture

Image (256×256)	PSNR (dB)						
	[14]	[19]	[22]	Proposed (32 bit fixed) in Spartan-6	Proposed (32 bit floating) in Spartan-6	Proposed (32 bit fixed) in Zynq UltraScale + MPSoC	Proposed (32 bit floating) in Zynq UltraScale + MPSoC
Lena	34.89	38.12	37.29	41.35	45.87	42.97	47.82
Cameraman	34.50	39.36	38.12	41.08	44.96	40.93	45.77
Pepper	34.53	40.03	37.43	40.85	44.01	41.04	45.53
Boat	34.80	40.59	38.89	41.81	46.12	42.33	46.17
Satellite image	33.92	37.15	37.02	39.48	43.64	38.79	44.43

Fig. 16 **a** Original images; **b** 2D DWT decomposed images using Spartan-6, and **c** reconstructed images

Table 8 Analysis of our proposed design for real-time three-level 2D DWT decomposition for 512×512 images

Design parameters	Architectures								
	[13]	[8]	[6]	[14]	[22]	Proposed (32 bit fixed) in Spartan-6	Proposed (32 bit floating) in Spartan-6	Proposed (32 bit fixed) in Zynq UltraScale + MPSoC	Proposed (32 bit floating) in Zynq UltraScale + MPSoC
DWT scheme	CV	LF	CV	LF	LF	CV	CV	CV	CV
Multiplier	16	10	189	0	–	0	0	0	0
Adder	12	16	294	34	–	49	49	49	49
ROM	6	65,565	81,920	65,565	73,728	24	24	24	24
Critical path/ cycle period	T_M	T_M	T_M	T_A	–	T_A	T_A	T_A	T_A
Critical path/ cycle period (ns)	7.897	6.543	6.896	2.830	11.577	2.720	159.640	2.540	147.972

Table 9 Comparison in terms of implemetation details of FPGA

Item	[15]	[16]	[14]	[22]	Proposed (32 bit fixed)	Proposed (32 bit floating)	Proposed (32 bit fixed)	Proposed (32 bit floating)
FPGA	Virtex-E	Virtex-4	Virtex-4	Artix-7	Spartan-6	Spartan-6	Zynq UltraScale + MPSoC	Zynq UltraScale + MPSoC
Device	XCV1000E-8	XCVFX140-12	XCVFX100-12	XC7Z020	XC6SLX25T	XC6SLX25T	XCZU7EV-2FFVC1156	XCZU7EV-2FFVC1156
Slice	1741 (14%)	973	630 (1%)	1124 (2%)	470 (12.50%)	752 (20%)	557 (0.0012%)	952 (0.0018%)
Memory	–	$\frac{10N}{2}$	$4N$	73,728	24	24	24	24
Maximum operating frequency	110 MHz	321 MHz	353.12 MHz	300 MHz	367.19 MHz	6.26 MHz	393.7 MHz	6.76 MHz
Power	–	–	139 mW	306 mW	115 mW	220 mW	97 mW	185 mW

Table 10 Comparison of CT (s) and PSNR (dB) for 512×512 image ('LENA') decomposition

Parameter	Binary representation	Different implementations			
		CPU	Cortex-A53 GPP	DSP unit in STM-32F446RE	Proposed architecture in Zynq UltraScale + MPSoC FPGA
CT (s)	32 bit fixed	0.288	0.059	0.043	0.0000188
	32 bit floating	0.389	0.107	0.086	0.0000491
PSNR (dB)	32 bit fixed	44.01	43.19	42.87	42.97
	32 bit floating	48.85	48.69	47.96	47.82

From Table 10 and Fig. 17, it is clear that our proposed VLSI architectures configured in both fixed-point and floating-point format outperform all the corresponding software implementations in terms of speed (in CT) at an almost comparable PSNR.

We have also compared the performance of our proposed VLSI design with two existing notable software-based implementations [23, 24] of 2D DWT realized on 'graphics processing unit' (GPU). The performance comparison results are listed down in Table 11. From Table 11, it is clear that the performance of our proposed VLSI architecture of 2D DWT is the best with respect to [23, 24] in terms of CT (in second) and PSNR (in dB) for the case of single-level 2D DWT decomposition of a standard 512×512 image, 'Lena'. The results presented in Table 11 establish that the timing performance of our proposed VLSI architecture outshines the performance of [23, 24] at a comparable PSNR.

The top level RTL schematic of our proposed 1D and 2D DWT architectures configured in 32 bit fixed-point format are shown in Fig. 18a, b, respectively. Figure 19 depicts the post-routing simulation results of our 32 bit fixed-point 2D DWT architecture in Xilinx Vivado 18.2

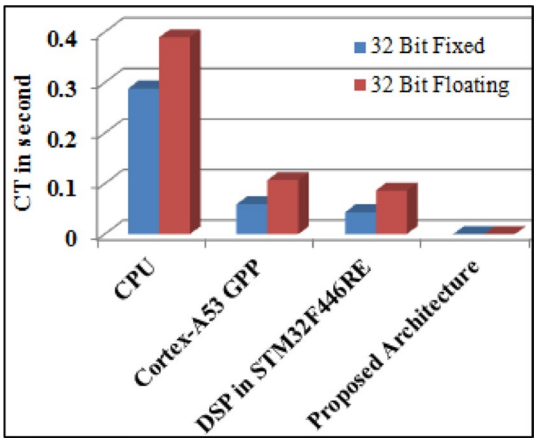


Fig. 17 Comparison of CT (in s) for different software implementations with proposed VLSI design

5 Conclusion

In this article, we have demonstrated our proposed 1D/2D DWT architecture. A dedicated VLSI architecture for real-time DWT decomposition is a much sought after thing now-a-days. As DWT is used in many signal-processing algorithms, therefore, our proposed DWT architecture's applicability is easily realizable. Though not sufficient in accordance with requirement, till date, a number of DWT architectures have been proposed. After studying the existing architectures, we have pointed out that each one of the architecture is designed for catering certain purposes while overlooking other purposes. For example, some of the well-known architectures are designed only for applications that require high-speed dedicated hardware. In that hardware memory requirement and area consumption is totally overlooked. We have tried to mitigate this problem by proposing our own DWT designs where we have tried to balance all the performance parameters, so that our architecture can be used as general purpose DWT hardware. In our architecture, we have introduced DA-based design strategy to make it multiplierless, thereby area efficient. We have chosen bit-parallel DA, so that the proposed architecture can be parallelized and pipelined. The memory requirement is also taken into account successfully. We have also introduced provisions to reconfigure

Table 11 Performance comparison of proposed VLSI architecture with [23, 24]

Parameter	Binary representation	[23]	[24]	Proposed architecture in Zynq UltraScale + MPSoC FPGA
CT (s)	32 bit fixed	0.0000795	0.0000315	0.0000188
	32 bit floating	0.00011	0.0000583	0.0000491
PSNR (dB)	32 bit fixed	41.59	42.07	42.97
	32 bit floating	46.23	48.54	47.82

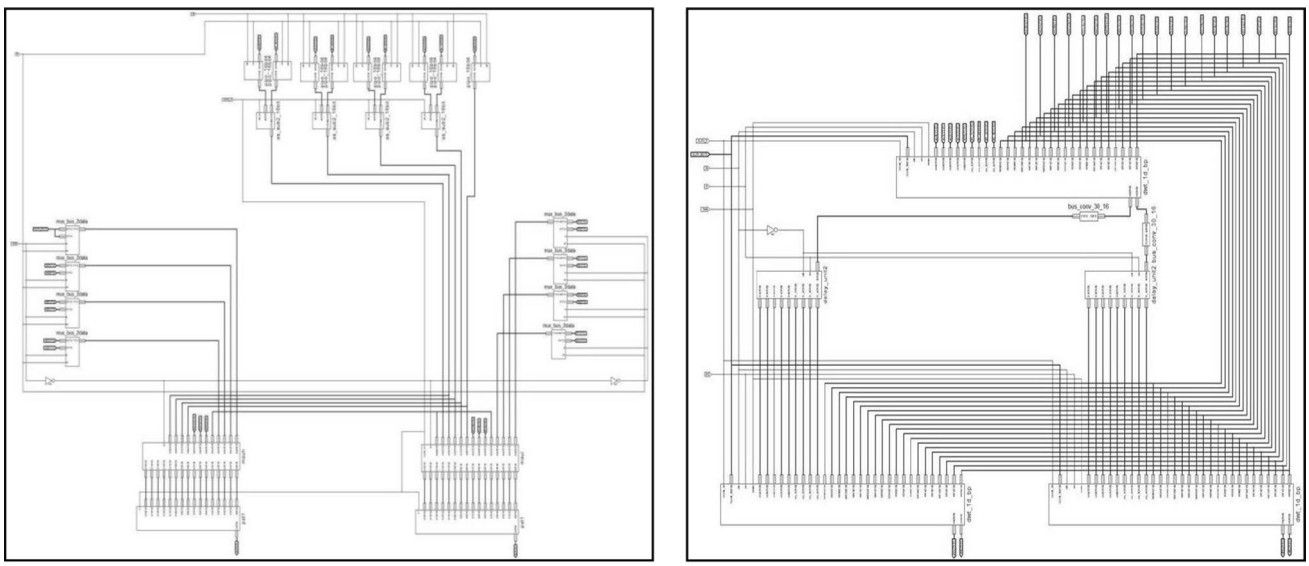


Fig. 18 **a** Top level RTL schematic of 1D DWT; **b** top level RTL schematic of 2D DWT

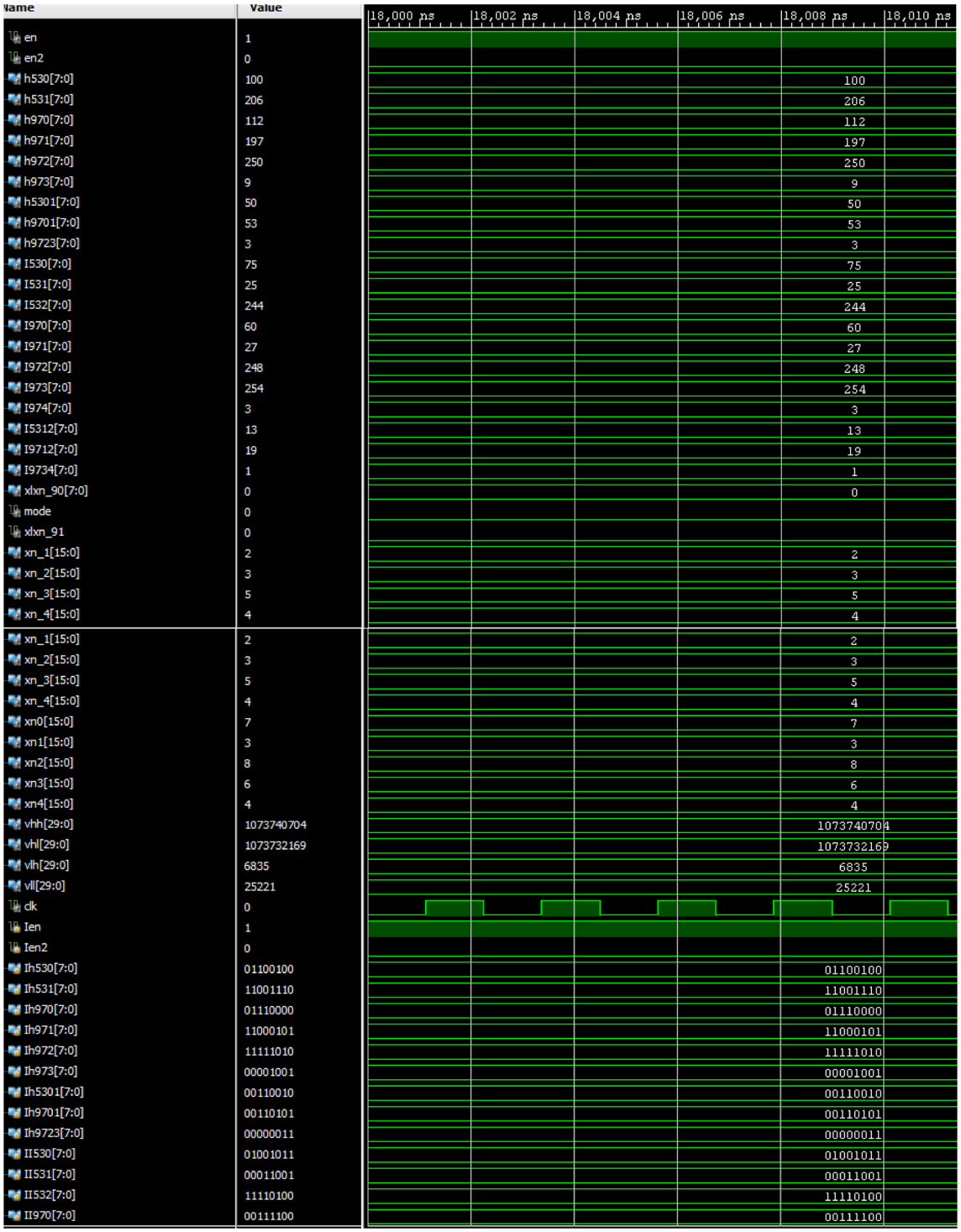


Fig. 19 Post-routing simulation of proposed 32 bit fixed-point architecture of 2D Dwt in Xilinx Vivado 18.2

our proposed architecture for 9/7 as well as 5/3 DWT filter by providing mode selection input. The supremacy of our proposed architecture has been established in terms of various design parameters by comparing it with other recent DWT architectures. While our proposed DWT design outshines all other architectures in some design parameters, it provides comparable results with respect to other design parameters. After statistical performance analysis, we have also applied our prototype DWT hardware in case of real-time image decomposition. The comparison results

for this real-time image decomposition also establish the superiority of our architecture.

References

1. Meyer, Y.: Wavelets: Algorithms and Applications. SIAM, Philadelphia (1993)
2. Daubechies, I., Sweldens, W.: J. Fourier Anal. Appl. **4**, 247 (1998)
3. Jou, J.M., Shiau, Y.-H., Liu, C.-C.: Efficient VLSI architectures for the biorthogonal wavelet transform by filter bank and lifting scheme. In: ISCAS 2001. The 2001 IEEE International Symposium on Circuits and Systems (Cat. No. 01CH37196), Sydney, vol. 2, pp. 529–532 (2001)
4. Mohanty, B.K., Meher, P.K.: Efficient multiplierless designs for 1-D DWT using 9/7 filters based on distributed arithmetic. In: Proceedings of the 2009 12th International Symposium on Integrated Circuits, Singapore, pp. 364–367 (2009)
5. Mahajan, A., Mohanty, B.K.: Efficient VLSI architecture for implementation of 1-D discrete wavelet transform based on distributed arithmetic. In: IEEE Asia Pacific Conference on Circuits and Systems, Kuala Lumpur, pp. 1195–1198 (2010)
6. Mohanty, B.K., Meher, P.K.: Memory-efficient high-speed convolution-based generic structure for multilevel 2-D DWT. IEEE Trans. Circ. Syst. Video Technol. **23**(2), 353–363 (2013)
7. Meher, P.K., Mohanty, B.K., Swamy, M.M.S.: Low-area and low-power reconfigurable architecture for convolution-based 1-D DWT using 9/7 and 5/3 filters. In: 28th International Conference on VLSI Design, Bangalore, pp. 327–332 (2015)
8. Lai, Y., Chen, L., Shih, Y.: A high-performance and memory-efficient VLSI architecture with parallel scanning method for 2-D lifting-based discrete wavelet transform. IEEE Trans. Consum. Electron. **55**(2), 400–407 (2009)
9. Xiong, C., Tian, J., Liu, J.: Efficient architectures for two-dimensional discrete wavelet transform using lifting scheme. IEEE Trans. Image Process. **16**(3), 607–614 (2007)
10. Mohanty, B.K., Meher, P.K.: Memory efficient modular vlsi architecture for high throughput and low-latency implementation of multilevel lifting 2-D DWT. IEEE Trans. Signal Process. **59**(5), 2072–2084 (2011)
11. Tian, X., Wu, L., Tan, Y., Tian, J.: Efficient multi-input/multi-output VLSI architecture for two-dimensional lifting-based discrete wavelet transform. IEEE Trans. Comput. **60**(8), 1207–1211 (2011)
12. Huang, C.-T., Tseng, P.-C., Chen, L.-G.: Generic RAM-based architectures for two-dimensional discrete wavelet transform with line-based method. IEEE Trans. Circ. Syst. Video Technol. **15**(7), 910–920 (2005)
13. Meher, P.K., Mohanty, B.K., Patra, J.C.: Hardware-efficient systolic-like modular design for two-dimensional discrete wavelet transform. IEEE Trans. Circ. Syst. II Express Briefs **55**(2), 151–155 (2008)
14. Darji, A., Arun, R., Merchant, S.N., Chandorkar, A.: Multiplierless pipeline architecture for lifting-based two-dimensional discrete wavelet transform. IET Comput. Dig. Tech. **9**(2), 113–123 (2015)
15. Dillen, G., Georis, B., Legat, J.D., Cantineau, O.: Combined line-based architecture for the 5-3 and 9-7 wavelet transform of JPEG2000. IEEE Trans. Circ. Syst. Video Technol. **13**(9), 944–950 (2003)
16. Das, A., Hazra, A., Banerjee, S.: An efficient architecture for 3-D discrete wavelet transform. IEEE Trans. Circ. Syst. Video Technol. **20**(2), 286–296 (2010)
17. Hegde, G., Reddy, K.S., Ramesh, T.K.S.: A new approach for 1-D and 2-D DWT architectures using LUT based lifting and flipping cell. AEU Int. J. Electron. Commun. **97**, 165–177 (2018)
18. Mohanty, B.K., Meher, P.K., Srikanthan, T.: Critical-path optimization for efficient hardware realization of lifting and flipping DWTs. In: IEEE International Symposium on Circuits and Systems (ISCAS), Lisbon, pp. 1186–1189 (2015)
19. Mohamed Asan Basiri, M., Noor Mahammad, S.: An efficient VLSI architecture for convolution based DWT using MAC. In: 31st International Conference on VLSI Design and 2018 17th International Conference on Embedded Systems (VLSID), Pune, pp. 271–276 (2018)
20. Aziz, F., Javed, S., Iftikhar Gardezi, S.E., Jabbar Younis, C., Alam, M.: Design and implementation of efficient DA architecture for LeGall 5/3 DWT. In: International Symposium on Recent Advances in Electrical Engineering (RAEE), Islamabad, pp. 1–5 (2018)
21. Gardezi, S.E.I., Aziz, F., Javed, S., Younis, C.J., Alam, M., Massoud, Y.: Design and VLSI implementation of CSD based DA architecture for 5/3 DWT. In: 16th IEEE International Bhurban Conference on Applied Sciences and Technology (IBCAST), pp. 548–552 (2019)
22. Naik, P., Guhilot, H., Tigadi, A., Ganesh, P.: Reconfigured VLSI architecture for discrete wavelet transform. In: Soft Computing and Signal Processing. Springer, Singapore, pp. 709–720 (2019)
23. Matela, J.: GPU-based DWT acceleration for JPEG2000. In: Annual Doctoral Workshop on Mathematical and Engineering Methods in Computer Science, pp. 136–143(2009)
24. Enfedaque, P., Auli-Llinas, F., Moure, J.C.: Implementation of the DWT in a GPU through a register-based strategy. IEEE Trans. Parallel Distrib. Syst. **26**(12), 3394–3406 (2014)
25. https://www3.ntu.edu.sg/home/ehchua/programming/java/datarepresentation.html. Accessed 6 July 2019
26. Al-Najjar, Y.A., Soong, D.C.: Comparison of image quality assessment: PSNR, HVS, SSIM, UIQI. Int. J. Sci. Eng. Res. **3**(8), 1–5 (2012)

Anirban Chakraborty is pursuing Ph.D. in the Dept. of Electronics and Telecommunication Engineering from IIEST, Shibpur, India. He has completed his M.E. with specialization in Nuclear Engineering from Jadavpur University, India. He has obtained B. Tech from Netaji Subhash Engineering College, Kolkata, India in Electronics and Communication Engineering. His research interest includes Digital Signal, Image Processing and VLSI design.

Ayan Banerjee is presently working as an Associate Professor in the Department of Electronics and Telecommunication Engineering at IIEST, Shibpur, India. He has obtained B.E. on 1994 from Bengal Engineering College, Shibpur, Calcutta University and M. Tech. from IIT, Kharagpur in Electronics and Electrical Communication Engineering with specialization in Integrated Circuits and Systems Engineering on 1999. He has completed his Ph.D. from IIT, Kharagpur on 2013 successfully. His research area includes Digital Signal/Image Processing, VLSI Architectures for Communication and Biomedical Engineering, CORDIC-based DSP architectures etc. He has authored a number of Journal and Conference papers of International repute.