Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2019) 27: 3823 – 3836
© TÜBİTAK
doi:10.3906/elk-1903-14

A novel hardware-efficient spatial orientation tree-based image compression algorithm and its field programmable gate array implementation

Mohd Rafi LONE[*], **Najeeb-ud-Din HAKIM**
Department of Electronics and Communication, National Institute of Technology, Srinagar, India

Received: 04.03.2019 • **Accepted/Published Online:** 05.06.2019 • **Final Version:** 18.09.2019

Abstract: Set partitioning in hierarchical trees (SPIHT) has become a popular research topic for more than a decade now. This is because it is simple, besides having compression efficiency close to the state-of-the-art JPEG2000 standard. The main drawback of SPIHT is that it uses three lists to store addresses of coefficients during its operation. These lists are dynamic and in worst cases need to store more number of addresses than total coefficients. In this work, a novel hardware-efficient spatial orientation tree-based algorithm is proposed and its hardware implementation is carried out. The wavelet transformed image is partitioned into 2×2 blocks. Each node of spatial orientation tree (SOT) represent a block of coefficients, rather than a single coefficient. Two small state-tables are used in this algorithm to store the status of each block. In addition to this, two extremely small lists are used to store the node addresses of a single SOT tree. To store the state-tables and lists for 512×512 image, only 0.88% of the memory needed by SPIHT is required for five levels of dyadic decomposition. The peak-signal-to-noise ratio (PSNR) gain of 0.1 dB to 0.3 dB at low bit rates (below 1 bpp) and 0.6 dB to 1.2 dB at high bit rates (above 1.75 bpp) in comparison to SPIHT is observed for test image Lena. A field programmable gate array (FPGA) implementation targeted for Xilinx Zynq Z-7020 is presented in the paper. The proposed architecture saves 90% of the FPGA area used by SPIHT. The hardware efficiency of the proposed architecture is better in comparison to different existing architectures.

Key words: Image compression, set partitioning in hierarchical trees, discrete wavelet transform, progressive image coding, spatial orientation tree

1. Introduction

Wavelet-based image coding techniques have completely replaced the discrete cosine transform-based methods. The state-of-the-art JPEG2000 image compression technique is also based on wavelet transform [1] using Cohen–Daubechies–Feauveau (9/7). JPEG2000 supports transmission of information in progressive manner; however, it is based on binary arithmetic coder which is complex and requires large area and memory, thereby decreasing its throughput [2]. Two other algorithms that support embedded coding are embedded zerotree wavelet [3] and Set partitioning in hierarchical trees (SPIHT) [4]. Both algorithms partition the transformed image into bit-planes and then, significance check based on set partitioning is made in breadth first manner. However, SPIHT further identifies the sets as Type A or Type B, and hence reduce the bit rate further. Type A node means that the children and grand descendants of the node are to be tested for significance. Type B node is one whose grand descendants only need to be tested for significance. SPIHT stands close to JPEG2000 in compression efficiency [5]. Furthermore, SPIHT has better hardware utilization in comparison to JPEG2000.

*Correspondence: rafimiet@gmail.com

Due to these reasons SPIHT has become a popular research topic and different variants have been proposed to date.

Traditional SPIHT [4] is based on three lists that contain the addresses of the pixels and sets that need to be processed. Large dynamic memory is needed to store the lists. As the bit rate increases, SPIHT can have more list nodes in comparison to the total number of coefficients. For real-time application, the realization becomes difficult and throughput reduces due to the dynamic memory management [6]. Different approaches have been considered to reduce the memory size of the SPIHT algorithm. No list SPIHT (NLS) [7] algorithm reduces the memory needed for the implementation of SPIHT. However, NLS lacks parallel processing of the coefficients; hence, a low throughput is achieved. A modified SPIHT algorithm [8], uses 1D addressing and simplified scanning to reduce the complexity of SPIHT. Low memory zero-tree coding [9] uses top-bit scheme for status of coefficients to avoid the use of lists. Listless modified SPIHT [10] encodes the more significant coefficients of low frequency band to ensure better performance. A block-tree–based method known as wavelet block truncation coding (WBTC) [11] improves the compression performance for low bit rates. However, like SPIHT, it also uses auxiliary lists whose size increases exponentially, requiring large memory. The dynamic memory has been reduced to some extent in listless block-tree coding (LBTC) [12]. LBTC has exploited the use of markers to store the significance status of the coefficients. Some papers have considered strip-based implementation of SPIHT, partioning the input image into strips and encoding each strip separately [13–15]. The advantages of different set-partitioning coding algorithms have been combined in [16]. A general tree is used which simultaneously represents tree set and square set.

The hardware implementation of NLS is presented in [17]. A 4 $\times$ 4 bit-plane is processed in one clock cycle by a modified SPIHT algorithm [18]. However, the critical path in this algorithm limits the operating frequency of the design, which in turn limits the throughput. A bit-plane parallel SPIHT encoder is proposed by Fry et al. [19]. This modified SPIHT algorithm decomposes wavelet coefficients into different bit-planes. Multiple bit-planes are processed in parallel. Four pixels are processed in parallel to achieve large throughput. However, decoder of this algorithm cannot achieve such a large throughput. Moreover, the hardware cost for this design is very high. Jin et al. [20] provide a block-based pass parallel SPIHT (BPS) algorithm that processes a single bit-plane block of (4 $\times$ 4) in a single clock cycle. The throughput of the encoder and decoder is almost identical. The main drawback of this algorithm is that it uses a fixed block size due to hardware complexity and memory requirements. A one-dimensional SPIHT (1D SPIHT) implementation is provided in [21, 22], which exploits parallelism to achieve large throughput. While improving throughput, performance of SPIHT is compromised to some extent. In [23], DWT and 1D SPIHT are optimized to meet the fixed target compression ratio in display systems, which use raster scan system. It is aimed at reducing the frame memory for video data compression. A real-time SPIHT decoder is presented in [24]. In [24], a cost- and power-effective implementation of 1D SPIHT decoder is presented for processing electrocardiograph (ECG) signals used in mobile health (mHealth) applications.

This paper proposes a spatial orientation tree-based algorithm and its FPGA implementation. The contribution of this work can be summarized as follows:

1. Compression efficiency of the proposed algorithm:

   (a) It provides higher peak-signal-to-noise ratio (PSNR) value in comparison to SPIHT at low bit rates (approx. less than 1 bpp).

   (b) It provides comparable PSNR to SPIHT at medium bit rates.

   (c) It provides higher PSNR value in comparison to SPIHT at high bit rates (approx. above 1.75 bpp).

2. Dynamic memory is reduced considerably.

3. FPGA area is reduced significantly.

4. Higher hardware utilization efficiency is achieved.

The paper is organized as follows. Section 2 provides a detailed description of the proposed algorithm. The memory requirement by the proposed algorithm is presented in Section 3 in comparison to other algorithms. The hardware architecture of the proposed algorithm is provided in Section 4. The performance and hardware results are presented in Section 5. Finally, Section 6 concludes the paper.

2. The proposed algorithm

The proposed method is based on state-table approach. The discrete wavelet transformed (DWT) image is partitioned into blocks of size 2 × 2. The spatial orientation of coefficient blocks at different dyadic levels is exploited. Two fixed-size state-tables are used to reflect the status of each block and its descendants (if they exist). The following are the definitions of the terms used in this algorithm:

(a) SIG_B: State-table for significance of block.

(b) SIG_D: State-table for significance of descendant blocks.

(c) LCB: List of child blocks.

(d) LPB: List of parent blocks.

(e) Type-A: All descendant nodes are insignificant and need to be checked at current threshold.

(f) Type-B: Some offspring nodes are significant already, others need to be checked at current threshold.

(g) O(x): Offspring of node x.

(h) D(x): Descendants of node x.

(i) I(p): Value of coefficient located at p.

(j) S_n(x): Significance of block at x at current threshold n.

The transformed image is saved in Morton Scan order. It is easy to navigate through a Spatial Orientation Tree (SOT) using Morton scanning method [25]. For any block node x, the offspring nodes are located in lower dyadic level at 4x, 4x+1, 4x + 2 and 4x + 3 respectively as shown in Figure 1. State-table, SIG_B is used to reflect the significance of each coefficient block. A node is assigned '1' if either at least one of the coefficients belonging to the node is found significant or any of its descendant nodes is found to be significant. State-table, SIG_D is used to reflect whether a node has any insignificant descendant or not. For an image of size N × N, SIG_B and SIG_D consists of $N^2/4$ bits and $N^2/16$ bits respectively. If all the descendant nodes are significant, SIG_D is assigned a '1' for that node. SIG_B and SIG_D are also arranged in Morton scanning

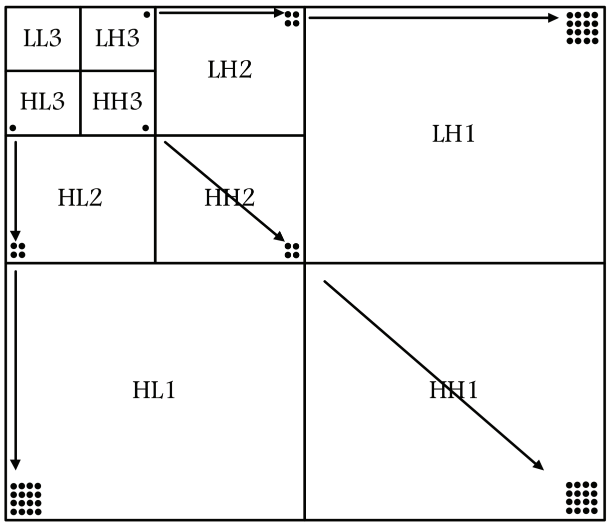

Figure 1. Spatial orientation in 3-level DWT subbands

order to easily locate the corresponding coefficients in the given image. Before starting the encoding process, SIG_B and SIG_D are initialized as under:

$$SIG\_B(x) = \begin{cases} 1, & x \ in \ LL \ band \\ 0, & Otherwise \end{cases} \tag{1}$$

$$SIG\_D(x) = \{0, \qquad for \ all \ x \tag{2}$$

Two passes, refinement pass (RP) and sorting pass (SP), are used in this algorithm as shown in Figure 2. Breadth-first search is used in RP, while depth-first search is used in SP. In RP, each coefficient in a significant block is compared to the current threshold. For a coefficient, which becomes significant at current threshold, a sign bit is appended along with the encoded bit stream. In depth-first mode (in SP), all the coefficients of a SOT tree are encoded for a given threshold before taking up the next SOT tree. This eliminates the need for storing coefficients other than current SOT tree. Two lists are used to hold the coefficient addresses during SP encoding process; list of child blocks (LCB) and list of parent blocks (LPB). These lists are very small in size and are initialized to be empty. The maximum size needed for a given level of DWT is fixed, as shown in Table 1. LCB is filled with the child nodes of the tree that are yet insignificant. While LPB consists of all respective parent nodes. The sizes of these lists are completely independent of the input image size.

Each sort tree has its roots in the highest dyadic level, as indicated by Figure 2. If SIG_D is '1' for any root node, then the next SOT tree is taken. For a Type-A node, a '0' bit is output if all its descendant nodes remain insignificant at current threshold. Otherwise, a '1' is output, followed by the encoded bits from each of the four offspring nodes. Once any of the offspring nodes is significant, the parent node is considered Type-B and each offspring node having SIG_D = '0' is checked separately. SIG_B and SIG_D are updated while processing LCB and LPB, respectively.

2.1. The proposed algorithm vs traditional SPIHT

It will be better to highlight some main differences between the traditional SPIHT and the proposed algorithm, before proceeding forward. The main difference that separates the two algorithms into two different paradigms

```
    REFINEMENT PASS
1   ❖ for each SIG_B(x) = 1
2     • if $S_n$(I(p)) = 1
3       o if $S_{n+1}$(I(p)) = 1
4         ➢ Output [1]
5       o else
6         ➢ Output [1 sgn]
7     • else
8       o Output [0]
    SORTING PASS
9   ❖ for each (i) in $LL_L$
10    • for each (k) ∈ O(i)
11      o for each SIG_D(k) = 0
12        ➢ if $S_n$(D(k)) = 0
13          ▪ Output [0000]
14        ➢ else
15          ▪ Move (k) to LPB
16          ▪ for each SIG_D(x) = 0 (x ∈ O(k))
17            ✓ Move (x) to LCB
18      o while LCB ≠ Ø
19        ➢ for each SIG_B{LCB(x)} = 0
20          ▪ for each I(p) ∈ LCB(x)
21            ✓ if $S_n$(I(p)) = 1
22              ♣ add [1 sgn] to Temp
23            ✓ else
24              ♣ add [0] to Temp
25          ▪ if SIG_B{LCB(x)} = 0 {[x] not in level 1
26            ✓ if $S_n${D{LCB(x)}} = 1
27              ♣ Output [1 Temp 1(type A)]
28              ♣ Add [x] to LPB
29              ♣ Add each [r] to LCB ;
30              ♣ Remove [x] from LCB
31              ♣ SIG_B{LCB(x)} = 1
32            ✓ else
33              ♣ if max(Temp) = 1
34                ♦ Output [1 Temp 0]
35                ♦ SIG_B{LCB(x)} = 1
36                ♦ Remove [x] from LCB
37              ♣ else
38                ♦ Output [0]
39                ♦ Remove [x] from LCB
40          ▪ If SIG_B{LCB(x)} = 0 {[x] in level 1}
41            ✓ if max(Temp) = 1
42              ♣ Output [1 Temp]
43              ♣ SIG_B{LCB(x)} = 1
44            ✓ else
45              ♣ Output [0]
46            ✓ Remove [x] from LCB
47        ➢ for each SIG_B{LCB(x)} = 1
48          ▪ if each SIG_B{O{LCB(x)}} = 0 (type A)
49            ✓ if $S_n${D{LCB(x)}} = 1
50              ♣ Output [1]
51              ♣ Add [x] to LPB
52              ♣ Add each [r] to LCB; {r ∈ O(x)}
53            ✓ else
54              ♣ Output [0]
55            ✓ Remove [x] from LCB
56          ▪ else (type B)
57            ✓ add [x] to LPB
58            ✓ for each [r] ∈ O(x)
59              ♣ if SIG_B(r) = 0 OR SIG_D(r) = 0
60                ♦ Add [r] to LCB;
61            ✓ Remove [x] from LCB
62      o while LPB ≠ Ø
63        ➢ for each (x) in LPB
64          ▪ if each SIG_D{O{LPB(x)}} = 1
65            ✓ SIG_D{LPB(x)} = 1
66          ▪ Remove [x] from LPB
```

Figure 2. The proposed compression algorithm.

Table 1. Size of lists verses No. of DWT levels

Level of DWT	LCB size	LPB size
3	7	5
4	10	21
5	13	85

is that SPIHT uses breadth first manner coding in sorting pass, while the proposed algorithm uses depth-first manner coding. As SPIHT encodes in breadth-first manner, some coefficients from different SOT trees are kept in the three lists of SPIHT and are not removed completely (until a zerotree is observed or the tree is exhausted). This is because, the coding algorithm returns to encode an SOT tree again and again to encode leftover portions of the tree (from root to leaves). While the proposed algorithm takes one SOT tree at a time and encodes it completely. The coefficients from other SOT trees are not processed until current SOT tree is encoded, thereby reducing the memory. The SPIHT encodes the transformed image by taking one coefficient at a time, while the proposed algorithm merges four SOT trees to encode them at a time. The traditional SPIHT uses three lists while the proposed algorithm uses only two lists with two state-tables.

Table 2. Dynamic memory required by different algorithms in addition to the memory required for the image itself.

Algorithm	SPIHT [4]	WBTC [11]	LBTC [12]	Proposed
memory required (kB)	1160	1100	128	10.2

(Image size: 512 × 512; L = 5; W = 18)

3. Memory Requirement

As discussed in Section 1, the large dynamic requirement by SPIHT is the main bottleneck of the algorithm. The proposed algorithm requires small fixed memory. The memory is needed to store the state-tables and the lists. Suppose R and C represent the number of rows and columns of a wavelet transformed image with L number of dyadic levels and each coefficient represented by W bits. Then the number of memory bits needed by the proposed algorithm are:

$$M_{Transformed\_Image} = R \times C \times W, \tag{3}$$

$$M_{SIG\_B} = (R \times C)/4, \tag{4}$$

$$M_{SIG\_D} = (R \times C)/16, \tag{5}$$

$$M_{LCB} = (4 + 3(L-2)) \times log_2(R \times C), \tag{6}$$

$$M_{LPB} = (1 + \sum_{n=1}^{L-2} 4^n) \times log_2(R \times C). \tag{7}$$

Therefore, the total number of bits that needs to be stored by the proposed algorithm in addition to transformed image itself are:

$$M_{Proposed} = (R \times C) \times (5/16) + log_2(R \times C) \times (5 + 3(L-2) + \sum_{n=1}^{L-2} 4^n), \tag{8}$$

where $\log_2(R \times C)$ gives the number of addressing bits needed for the image.

If the image size is 512 × 512, the dyadic decomposition level will be 5 and number of bits per coefficient will be 18, then the dynamic memory size required in addition to that required by the image itself is provided in Table 2. It is evident from Table 2 that the memory required by the proposed algorithm is only 0.88%, 0.93%, and 7.97% of that needed by SPIHT, WBTC, and LBTC, respectively. In other words, SPIHT, WBTC, and LBTC respectively need 114, 108, and 12.5 times more memory than that needed by the proposed algorithm.

4. Hardware design

The top-level architecture of the proposed compressor is shown in Figure 3. The first stage performs 2D-DWT on the input. The transformed data is then made available for the second stage to perform encoding. In some image coders like JPEG2000, the input image is divided into multiple subimages called tiles or transform-blocks. Instead of processing the whole image, these small sized transform-blocks are processed. Each transform-block

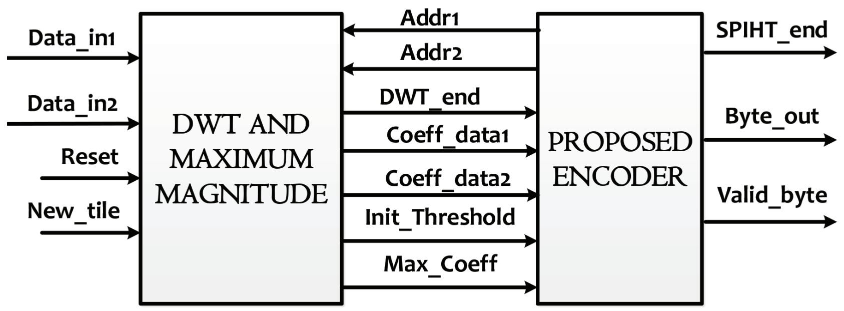

Figure 3. Top-level architecture of the proposed compressor.

is processed independently and separately, one after the other. The advantage is the reduction in FPGA area utilization and complexity of the compressor. This is because, the compressors usually have a great dependence on image size. However, the tiling results in blocking artifacts, which are much visible at low bit rates and higher levels of DWT. Different techniques for reducing the blocking artifacts are presented in [26]. The proposed algorithm has significantly reduced the dependence of its architecture on image size. Small increase in hardware area is observed for a large change in image size. As the *New_tile* signal goes high, the DWT module starts processing the input data. When all the desired pixels are transformed in the current tile, *DWT_end* is set and the transformed coefficients are made available for encoding by the second stage.

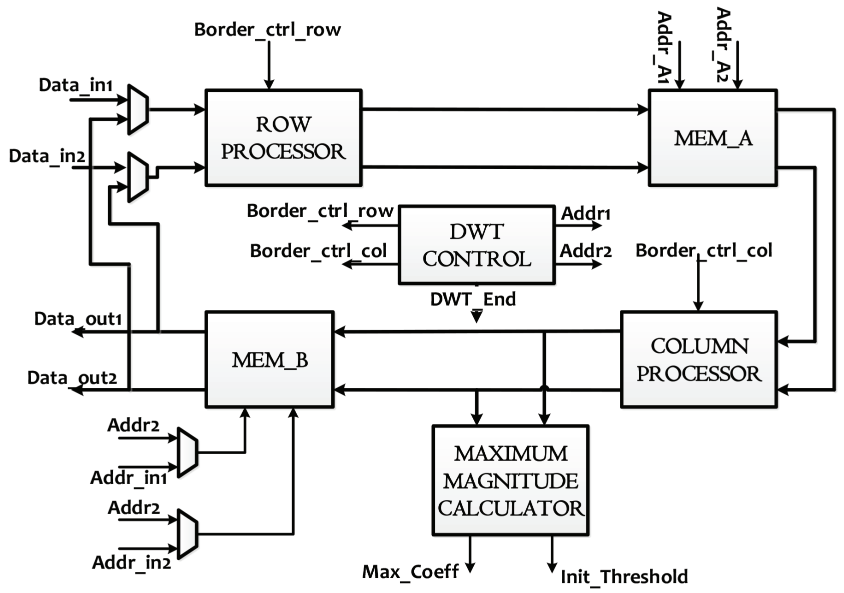

Figure 4. Discrete wavelet transform architecture.

4.1. Discrete-wavelet transformation architecture

The inputs to the compressor are two pixels: *Data_in1* and *Data_in2*. A 3-level DWT transform based on Cohen–Daubechies–Feauveau 5/3 wavelet filter is used here. The forward DWT arrangement is shown in Figure 4. Row and column processors are used to take 1-D DWT on horizontal and vertical directions respectively. The border management for the image is controlled by the signals *Border_ctrl_row* and *Border_ctrl_col*, as illustrated in [27]. Two dual-port memory banks, MEM_A and MEM_B are used to hold the intermediate

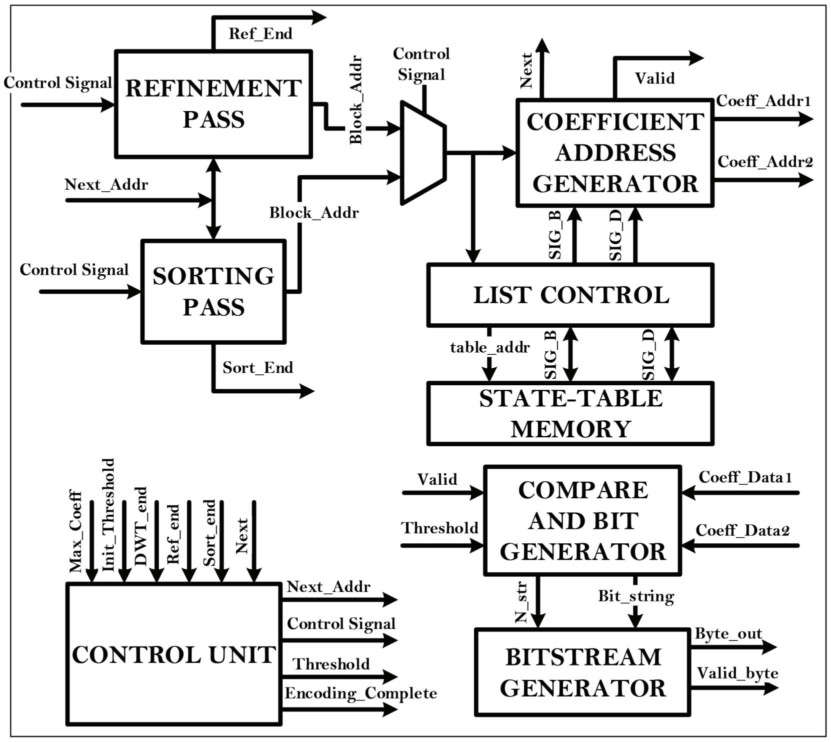

Figure 5. The proposed encoder architecture.

coefficients. The word length taken is 16 bits per coefficient. After first level of DWT, the input to row processor accepts data from MEM_B. The initial threshold *Init_Threshold* and maximum coefficient *Max_Coeff* among high-frequency bands are calculated in the DWT stage. It has been observed that the initial few sorting passes do not yield any information as no coefficient in higher frequency bands is greater than the threshold. Encoding in these sorting passes results in loss of compression efficiency. After each RP, a comparison is made between threshold and *Max_Coeff*; if it is greater than the threshold, then SP is taken, otherwise it is skipped and the threshold is halved. When the three levels of transformation are completed, the desired transformed coefficients are stored in MEM_B in Morton scan order. *DWT_end* is set and the contents of the memory are made available to the SPIHT encoder. The data is output through *Data_out1* and *Data_out2* for the addresses *Addr_in1* and *Addr_in2*.

4.2. The proposed encoder architecture

As the *DWT_end* goes high, the control shifts to the encoding stage. The FPGA architecture for the proposed encoder is presented in Figure 5. The two passes, refinement pass and sorting pass, are processed one after the other, starting with refinement pass. For three-level transformed image, 7 and 5 addresses are required by LCB and LPB, respectively. As these lists are very small in size, no separate memory is used. However, list control is required to manage the addresses and update SIG_B and SIG_D table. In refinement pass, the block addresses are calculated and then the list control checks if the SIG_B is ‘1’ or not. If it is ‘1’ then the corresponding coefficients are checked, otherwise the next block address is sought. If a block is to be encoded, then coefficient addresses, *Coef_Addr1* and *Coeff_addr2*, are sent to MEM_B. Corresponding coefficients are then compared with the threshold and output bits are generated accordingly. If a coefficient becomes significant in current pass, then bits ‘1’ and ‘sgn’ are generated. Otherwise, only bit ‘1’ or ‘0’ is generated for significant and insignificant coefficients, respectively. Thus, a coefficient may be encoded either using one or two bits. This makes it impossible for the decoder to predict where the sorting pass starts and hence makes parallelism difficult. When the refinement pass is completed, *Ref_end* is set high and sorting pass commences. In sorting pass, SIG_D determines whether we have to encode a SOT tree or not. If it is ‘1’ then it means all the descendants are already significant and can be further refined in refinement pass only. If a SOT tree is taken for encoding, SIG_D and SIG_B are further used to determine whether a particular node of the SOT tree and its corresponding descendants need to be checked or not. If the coefficients corresponding to the current block need to be checked, Valid is set high. The coefficient addresses are sent to MEM_B which returns corresponding coefficient values, as was done in refinement pass. These coefficient values are compared with current threshold and accordingly *Bit_string* and *N_str* are generated. *N_str* gives the number of bits in *Bit_string* that are valid at any time. When sorting pass is completed, *Sort_end* is set high. Finally, a byte long output bit string is generated in the form of *Byte_out*. *Valid_byte* reflects whether the *Byte_out* is valid or invalid in a particular clock cycle.

5. Results and discussion

The main focus of this paper is to reduce the FPGA area and provide better hardware utilization efficiency besides improving the performance in comparison to SPIHT and its different variants. For performance evaluation, test images from two databases are used, these are: The USC-SIPI Image Database [1] and JPEG2000 test set. In this section, performance evaluation and hardware utilization of the proposed work is presented.

5.1. Performance evaluation

The performance of the proposed algorithm is evaluated using PSNR. Two sets of standard 8 bit per pixel monochrome images are used. The USC-SIPI Image Database includes Lena, Barbara, Baboon, Goldhill, Boat, Pirate, and Aeroplane. The size of each image in this set is 512 × 512. The images of the second set are from JPEG2000 test set. These include high-resolution images of size 2560 × 2048, namely Bike, Woman, and Cafe.

Performance evaluation of Lena, Barbara, and Goldhill is provided in Table 3 for bit rates in range 0.25 and 1. The improvement in PSNR for Lena ranges from 0.2 dB to 0.5 dB and 0.1 dB to 0.3 dB against SPIHT [4] and SPIHT-ZTR [14], respectively. The PSNR gain for Barbara ranges from 0.6 dB to 1.2 dB against both SPIHT and SPIHT-ZTR. For Goldhill, the PSNR does not show much improvement. Table 4 shows the PSNR comparison for high-resolution images, Woman and Bike, for the bit rates in range 0.0625 bpp and 2 bpp. From the table, it is evident that the PSNR in the proposed algorithm shows improvement for higher and lower bit rates. In the middle bit rates, the PSNR does not show much improvement and in some cases lags behind other algorithms. One of the reasons for performance improvement at lower bit rates is that initial sorting passes are skipped in proposed algorithm, unless it is confirmed that higher subbands have some coefficients higher than threshold. In a bit-plane encoding pass, the bits generated are almost the same as that of SPIHT. However, the proposed algorithm uses depth-first search while as other encoders use breadth-first search. Breadth-first has an advantage over depth-first in that the chances of encoding more significant coefficients early in initial passes are more in breadth-first. This is the reason that in middle bit rates, the performance of the proposed algorithm decreases. At high bit rates, depth-first search may result in higher performance as compared to breadth-first because the high-frequency bands also have a sufficient number of coefficients that are significant at these low

[1]The USC-SIPI (1977). Image Database [online]. Website http://sipi.usc.edu/database [accessed 23 April 2019].

Table 3. Performance of the proposed algorithm in comparison to SPIHT and SPIHT-ZTR algorithms in terms of peak-signal-to-noise ratio (dB) versus bit-rate (bpp) for various test images of size 512 $\times$ 512 (without arithmetic coding).

Bit Rate	SPIHT	SPIHT-ZTR	Proposed	SPIHT	SPIHT-ZTR	Proposed	SPIHT	SPIHT-ZTR	Proposed
(bpp)	LENA			BARBARA			GOLDHILL		
0.25	32.91	32.98	33.11	26.14	26.16	26.84	29.91	29.84	30.05
0.50	36.07	36.17	36.37	29.60	29.65	30.57	32.33	32.33	32.30
0.80	38.34	38.46	38.74	32.86	32.95	34.15	34.41	34.62	34.54
1.00	39.31	39.49	39.61	34.29	34.46	35.40	35.66	35.77	35.61

Table 4. Performance of the proposed algorithm in comparison to SPIHT, NLS, WBTC, and LBTC algorithms in terms of PSNR (dB) for test images of size 2560 $\times$ 2048 (without arithmetic coding).

Images	Algorithm	Bit rate (bpp)					
		0.0625	0.125	0.25	0.50	1.00	2.00
Woman	SPIHT [4]	25.07	26.91	29.43	32.93	37.73	43.21
	NLS [7]	26.61	28.34	30.56	33.30	36.99	42.60
	WBTC [11]	25.29	27.08	29.71	33.23	37.91	43.62
	LBTC [12]	26.71	28.38	30.59	33.31	37.00	42.62
	Proposed	25.18	27.03	29.13	32.44	37.09	43.62
Bike	SPIHT [4]	22.86	25.29	28.51	32.36	36.98	42.90
	NLS [7]	23.40	25.86	28.75	32.15	36.46	42.53
	WBTC [11]	23.14	25.48	28.76	32.51	37.12	43.13
	LBTC [12]	23.49	25.91	28.77	32.17	36.47	42.55
	Proposed	23.24	25.52	28.61	32.41	36.73	42.51

Table 5. Performance comparison for two sets of images with sizes 512 $\times$ 512 and 2560 $\times$ 2048, for higher bit rates (without arithmetic coding).

Bit rate (bpp)	PSNR (dB) (512 $\times$ 512)		PSNR (dB) (2560 $\times$ 2048)		Average PSNR (dB)	
	SPIHT [4]	Proposed	SPIHT [4]	Proposed	SPIHT [4]	Proposed
1.50	38.56	38.75	40.58	40.20	39.57	39.48
1.75	39.69	39.94	41.76	41.50	40.72	40.72
2.00	40.87	41.08	42.78	43.08	41.83	42.08
2.25	42.00	42.14	44.27	44.22	43.13	43.18
2.50	43.10	43.42	45.27	45.10	44.19	44.26
2.75	44.22	44.57	46.19	46.27	45.20	45.42
3.00	45.21	45.93	47.08	47.72	46.15	46.82
3.25	46.45	47.06	48.10	49.47	47.28	48.26
3.50	47.52	48.14	49.58	50.34	48.55	49.24
3.75	48.60	49.61	50.35	51.23	49.48	50.42
4.00	49.72	51.01	51.04	52.43	50.38	51.72

Table 6. Hardware resource utilization of the proposed architecture for different transform block sizes.

Transform block	Slices		Block RAM (18 Kb)	
Size	Encoder	Decoder	Encoder	Decoder
16 × 16	159	166	0	0
32 × 32	167	171	0	0
64 × 64	170	182	1	1
128 × 128	175	232	1	1
256 × 256	190	237	2	2

Table 5 presents the comparison of performance between SPIHT and the proposed algorithm for higher bit rates. Both sets of images are used for evaluation at these bit rates. The average performance for the image sets are listed in the table. Finally a total average is presented in the last two columns of the table. It is very clear that the performance of the proposed algorithm improves in comparison to SPIHT as the bit rate increases. At around 1.75 bpp, the proposed algorithm surpasses SPIHT and the PSNR difference starts growing as bit rate goes up. The PSNR improvement is around 0.60 dB at approximately 3 bpp. As the bit rate further increases, the PSNR differences grows to 1.3 dB at 4 bpp.

5.2. Hardware implementation

The FPGA implementation of the proposed architecture is targeted for Xilinx Zynq Z-7020 board [2]. The synthesis and simulation tool for its implementation is Vivado 2015.4 using VHDL hardware description language. The implementation is designed for 16 × 16 transform block, DWT level 3 and word length of 13 bits per DWT transformed coefficient. However, it can easily be modified for different sized transform block, DWT level, or word-length. The main focus of this work is to reduce the FPGA area utilized for its implementation and at the same time keep the area per unit normalized throughput as low as possible

Table 6 presents the FPGA resource utilization against transform block size. Different block sizes are used as indicated by the first column of the table. The second and third columns provide the number of slices used by the proposed encoder and decoder respectively. It is very clear that the proposed decoder consumes more resources in comparison with Encoder. The reason is that resource sharing in decoder is difficult as predicting the input bit stream is not possible. As the transform block size increases, the number of slices utilized also increases, but the increase is very slow. The reason for this slow increase against large increase in transform block size is that the architecture has a very small dependence on transform block size as discussed in Section 5. Columns 4 and 5 refer to the number of 18 kb Block RAMs used for storing the state-tables. At lower transform block sizes, the state-table size is so small that using a BRAM will be a mere waste of resources, FPGA logic is used instead.

Table 7 shows the comparison of FPGA implementation for the proposed architecture with existing variants of SPIHT. The proposed encoder and decoder designs are shown in the seventh and eighth columns of the table. The first row shows the different technologies used by the previous architectures. The logic size utilized by the designs is presented in the second row of the table in terms of number of slices. To make the comparison fair and independent of technology, the comparison is made using equivalent logic cells in third row

[2]Xilinx Zynq-7000 SoC (2017). Technical Reference Manual [online]. Website https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html [accessed 01 March 2018].

Table 7. Comparison of FPGA implementation for the proposed architecture with previous architectures.

Architecture	NLS [17]*	Bit-plane parallel SPIHT [19]*	BPS [20]		Proposed	
			Encoder	Decoder	Encoder	Decoder
Technology	Xil. Virtex 2-4	Xil. Virtex 2000E	Xil. Virtex 5-LX330		Xil. Zynq Z-7020	
Slice (no.)	4500	11,904+6528+18,816	3592	3431	159	166
Logic Cell (no.)	$\frac{10,125}{11,520}$	$\frac{83,808}{43,200}$	$\frac{22,996}{331,876}$	$\frac{21,901}{331,876}$	$\frac{1,017}{85,000}$	$\frac{1,064}{85,000}$
Frequency (MHz)	100	56	150	110	253	195
Throughput (Gbits/s)	0.29	3.58	2.34	1.72	0.3	0.26
Norm. Thr. (Mbits/(s $\times$ MHz)	2.9	63.9	15.6	15.6	1.19	1.33
Logic Cell/Norm Thr	3491	1312	1472	1404	854	800
Memory (Kbit)	432	N/A	8.3	6.7	0	0

*Encoder only

of the table. From the table, it is evident that the logic cells utilized by the proposed encoder are 10%, 1.21%, 4.42%, and 16.34% of that used by NLS, bit-plane-parallel SPIHT, BPS, and 1D SPIHT, respectively. Similarly, the decoder of the proposed design uses 20.58% and 20.27% of the logic used by BPS and 1D SPIHT, respectively. The operating frequency achieved by the architectures is provided in the fourth row. The proposed architecture gives the maximum frequency reached by any design. The fifth and sixth rows provide the throughput and normalized throughput of the designs. As the proposed architecture has a sequential behavior, comparatively a low throughput is achieved. However, the throughput is sufficient to encode 42 and decode 37 frames of high definition resolution (1280 $\times$ 720) gray scale images per second. The hardware efficiency parameter used in terms of logic cell/normalized throughput is shown in the seventh row. It is clear that the proposed architecture has used minimum logic to achieve a certain normalized throughput in comparison to NLS, Bit-plane parallel SPIHT, and BPS. 1D SPIHT has provided even better hardware efficiency; however, it needs 16 times more logic for encoder and 25 times more logic for decoder than proposed architecture to achieve such a high throughput. Finally, the memory used by an architecture is provided in the eighth row. The memory size is dependent on the transform block size. The proposed architecture, BPS, and 1D SPIHT use 16 $\times$ 16 transform block size while NLS uses 64 $\times$ 64. To compare with NLS, Table 6 depicts that the proposed architecture requires only one 18 Kb RAM for a transform block size of 64 $\times$ 64.

6. Conclusion

A novel spatial-orientation based compression algorithm has been proposed in this paper. The main focus of this algorithm is to reduce the hardware complexity in comparison to SPIHT and its variants. Two very small, state-tables and lists are used to achieve the goal. The memory requirement has been reduced significantly. The proposed algorithm achieves better PSNR at low (below 1 bpp) and high (above 1.75 bpp) bit rates. The FPGA area utilized by the proposed architecture is considerably reduced and a moderate throughput that could encode 42 and decode 37 gray scale images of size 1280 $\times$ 720 (high definition) per second is reached.

The algorithm proposed here is highly sequential in nature. Throughput can be improved by employing parallelism in the design. As the algorithm is block-based, parallelism can be achieved very easily by processing

2 × 2 blocks of coefficients. Moreover, multiple blocks of a specific SOT tree can be processed in parallel easily as the algorithm encodes sorting pass in depth-first manner.

References

[1] JPEG2000 Image Coding System, document ISO/IEC 15444-1, 2000.

[2] Pastuszak G. A novel architecture of arithmetic coder in JPEG2000 based on parallel symbol encoding. In: International Conference on Parallel Computing in Electrical Engineering; Dresden, Germany; 2004. pp. 303-308.

[3] Shapiro JM. Embedded Image coding using zerotrees of wavelet coefficients. IEEE Transactions on Signal Processing 1991; 41(12): 3445-3462.

[4] Said A, Pearlman W. A new, fast, and efficient image codec based on set partitioning in hierarchical trees. IEEE Transactions on Circuits and Systems for Video Technology 1996; 6(3): 243-250.

[5] Pearlman WA, Islam A, Nagaraj N, Said A. Efficient, low complexity image coding with a set-partitioning embedded block coder. IEEE Transactions on Circuits and Systems for Video Technology 2004; 14(11): 1219-1235.

[6] Ritter J, Fey G, Molitor P. SPIHT implemented in a XC4000 device. In: Midwest Symposium on Circuits and Systems; Tulsa, OK, USA; 2002. pp. 239-242.

[7] Wheeler F, Pearlman W. SPIHT image compression without lists. In: IEEE International Conference on Acoustics, Speech, and Signal Processing; Istanbul, Turkey; 2000. pp. 2047-2050.

[8] Jyotheswar J, Mahapatra S. Efficient FPGA implementation of DWT and modified SPIHT for lossless image compression. Journal of Systems Architecture 2007; 53(1): 369-378.

[9] Su CY, Wu BF. A low memory zerotree coding for arbitrarily shaped objects. IEEE Transactions on Image Processing 2003; 12(3): 271-282.

[10] Pan H, Siu WC, Law NF. A fast and low memory image coding algorithm based on lifting wavelet transform and modified SPIHT. Signal Processing: Image Communication 2008; 23(1): 146-161.

[11] Moinuddin AA, Khan E, Ghanbari M. Efficient algorithm for very low bit rate embedded image coding. IET Image Process 2008; 2(2): 59-71.

[12] Senapati RK, Pati UC, Mahapatra KK. Listless block-tree set partitioning algorithm for very low bit rate embedded image compression. International Journal of Electronics and Communications 2012; 66(12): 985-995.

[13] Chew LW, Ang LM, Seng KP. New virtual SPIHT tree structures for very low memory strip-based image compression. IEEE Signal Processing Letters 2008; 15(1): 389-392.

[14] Chew LW, Chia WC, Ang L, Seng KP. Very low-memory wavelet compression architecture using strip-based processing for implementation in wireless sensor networks. EURASIP Journal on Embedded Systems 2009; (12): 1:16.

[15] ZainEldin H, Elhosseini MA, Ali HA. A modified listless strip based SPIHT for wireless multimedia sensor networks. Computers and Electrical Engineering 2016; 56(11): 519-532.

[16] Li Q, Chen D, Jiang W, Liu B, Gong J. Generalization of SPIHT: set partition coding system. IEEE Transactions on Image Processing 2016; 25(2): 713-725.

[17] Corsonello P, Perri S, Staino G, Lanuzza M, Cocorullo G. Low bit rate image compression core for onboard space applications. IEEE Transactions on Circuits and Systems for Video Technology 2006; 16(1): 114-128.

[18] Cheng CC, Tseng PC, Chen LG. Multimode embedded compression codec engine for power-aware video coding system. IEEE Transactions on Circuits and Systems for Video Technology 2009; 19(2): 141-150.

[19] Fry TW, Hauck SA. SPIHT image compression on FPGAs. IEEE Transactions on Circuits and Systems for Video Technology 2005; 15(9): 1138-1147.

[20] Jin Y, Lee HJ. A block-based pass-parallel SPIHT algorithm. IEEE Transactions on Circuits and Systems for Video Technology 2012; 22(7): 1064-1075.

[21] Kim S, Lee D, Kim JS, Lee HJ. A high-throughput hardware design of a one-dimensional SPIHT algorithm. IEEE Transactions on Multimedia 2016; 18(3): 392-404.

[22] Kim S, Lee D, Kim H, Truong NX, Kim JS. An enhanced one-dimensional SPIHT algorithm and its implementation for TV systems. Displays 2015; 40(12): 68-77.

[23] Nguyen XT, Lee H, Kim H. A low-cost hardware design of a 1-D SPIHT algorithm for video display systems. IEEE Transactions on Consumer Electronics 2018; 64(1): 44-52.

[24] Hsieh J, Shih M, Huang X. Algorithm and VLSI architecture design of low-power SPIHT decoder for mHealth applications. IEEE Transactions on Biomedical Circuits and Systems 2018; 12(6): 1450-1457.

[25] Artyomov E, Rivenson Y, Levi G, Pecht OY. Morton (Z) scan based real-time variable resolution CMOS image sensor. IEEE Transactions on Circuits and Systems for Video Technology 2005; 15(7): 947-952.

[26] Liang J, Tu C, and Tran TD. Optimal block boundary pre/post-filtering for wavelet-based image and video compression. IEEE Transactions on Image Processing 2005; 14(12): 2151-2158.

[27] Lone MR, Hakim N. A novel arrangement for efficiently handling image border in FPGA filter implementation. In: 3rd International Conference on Signal Processing and Integrated Networks; Noida, India; 2016. pp. 163-168.