Turkish Journal of Electrical Engineering and Computer Sciences

Volume 26 | Number 6 Article 12

1-1-2018

FPGA implementation of a low-power and area-efficient state-table-based compression algorithm for DSLR cameras

MOHD RAFI LONE

NAJEEB UD DIN HAKIM

Turkish Journal of Electrical Engineering & Computer Sciences

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

Research Article

Turk J Elec Eng & Comp Sci
(2018) 26: 2927 – 2942

doi:10.3906/elk-1804-208

FPGA implementation of a low-power and area-efficient state-table-based compression algorithm for DSLR cameras

Mohd Rafi LONE*, **Najeeb-ud-Din HAKIM**[1]
Department of Electronics and Communication Engineering, National Institute of Technology, Srinagar, India

Received: 28.04.2018 • **Accepted/Published Online:** 07.09.2018 • **Final Version:** 29.11.2018

Abstract: Small image acquisition devices like digital single lens reflex (DSLR) cameras most commonly use Joint Photographic Expects Group (JPEG) coding standard for lossy compression. Although JPEG is a simple coding standard, its compression efficiency is very low as compared to any typical state-of-the-art image coding standards like set partitioning in hierarchical trees (SPIHT). In this paper, a novel state-table-based SPIHT (STS) algorithm and its field programmable gate array (FPGA) implementation is proposed. The STS uses two small state-tables and two extremely small lists. The STS not only provides better compression efficiency than the state-of-the-art JPEG 2000 at high bit rates but also requires very small memory to hold the state-tables and lists in comparison to SPIHT. On average STS requires 0.86% of the memory needed by SPIHT when evaluated for image sizes ranging from 4 Mpixels to 40 Mpixels. The implementation results show that STS consumes very less FPGA area in comparison to SPIHT-based architectures. The dynamic power dissipation of STS is also less than that of JPEG-like compression standards. This makes our proposed algorithm a better candidate for compression in low-power, low-memory digital image acquisition devices.

Key words: Set partitioning in hierarchical trees, very large-scale integration, image compression, dynamic power dissipation, field programmable logic array, digital single lens reflex

1. Introduction

The global camera industry has made significant growth in the last decade. The main focus has been given to high-end digital single lens reflex (DSLR) cameras. The areas of continuous improvement in camera technology include resolution, compactness, multiple scene modes, storage, and others. The central goal of all improvements in sophisticated camera technology is to improve the quality of the captured scene. One of the issues limiting the image quality in small digital cameras that has attracted less attention is image compression. As the image is captured by camera sensor, it is either compressed and then stored, or stored directly on a secure digital (SD) card. Many cameras have an option to store the captured images in uncompressed RAW format. Some cameras support a lossless Tagged Image File Format (TIFF) compression standard. Joint Photographic Experts Group (JPEG) [1] is the widely used lossy compression standard in modern DSLR cameras provided by leading manufacturers like Canon[1] and Nikon[2] [2]. A comparison of JPEG with RAW was given in [3]. Despite providing low performance in comparison to the state-of-the-art compression standards, JPEG is widely used

*Correspondence: rafimiet@gmail.com

[1]Image compression: Lossless and lossy compression, http://cpn.canon-europe.com/content/education/infobank/image_compression.do
[2]DSLR Camera Basics, http://imaging.nikon.com/lineup/dslr/basics/26/01.htm

in these devices. The reasons include:

(a) Simple coder;

(b) Low computational complexity;

(c) High compatibility with other devices, browsers, and graphics programs;

(d) Memory-efficient;

(e) Real-time processing.

Discrete cosine transform (DCT)-based image coding algorithms like JPEG have been replaced completely by discrete wavelet transform (DWT)-based image coding techniques. Embedded zerotree wavelet (EZW) has provided a new paradigm of image coding by introducing progressive, zerotree-based coding technique [4]. Set partitioning in hierarchical trees (SPIHT) [5] took it to a higher level, where the compression efficiency reaches that of the state-of the-art JPEG 2000 compression standard [6]. The baseline JPEG 2000 consists of four stages: DWT, quantization, and tier 1 and tier 2. The tier 1 JPEG 2000 consists of bit-plane coder (BPC) and arithmetic coder (AC). The BPC forms context and decision pairs from a code-block of quantized coefficients. These context and decision pairs are then encoded using an AC coder to generate a compressed bit stream. The tier 2 stage finally reorganizes the bit stream and places markers into it. The BPC has high computational complexity, as it has to process context-decision pairs at bit-level. On the other hand, the AC coder is highly sequential and restricts the throughput of the overall JPEG 2000 standard. Although modifications to the binary arithmetic encoder are proposed by many researchers [7, 8], the resources utilized and dynamic power dissipation are still high and the throughput achieved is low. High throughput is achievable in SPIHT. In SPIHT the complexity in its algorithm comes from the three lists that it uses to encode an image. Different variants of SPIHT have been proposed. The goals of these variants are to:

(i) Improve compression efficiency [9–11];

(ii) Reduce memory requirements for software and hardware implementations [12, 13];

(iii) Provide better throughput; and

(iv) Ensure less hardware utilization [14].

Furthermore, the cameras are mostly based on microprocessors. Microprocessor-based algorithms provide a low performance and consume more power in comparison to Application Specific Integrated Circuit (ASIC) implementations [15]. Despite being power-hungry devices, microprocessor-based implementation is preferred in modern cameras due to their reconfigurability, reusability, and low cost. On the other hand, field programmable gate arrays (FPGAs) provide performance very close to ASICs and the flexibility very close to that of microprocessors [16].

No-list SPIHT (NLS) [17] has been proposed to reduce the memory of SPIHT. The hardware implementation of the algorithm was presented in [18]. The memory is reduced successfully in NLS; however, it provides a low throughput. A 4×4 bit-plane is processed per clock cycle in modified SPIHT implementation [19]. However, the critical path reduces its frequency significantly. A bit-plane parallel SPIHT architecture was proposed

in [20]. Four pixels are processed in parallel in this architecture. The throughput of the encoder is improved; however, the decoder cannot reach a similar throughput. Also, the hardware cost of this architecture is very high. Block-based pass parallel SPIHT (BPS) was proposed in [21]. A bit-plane block of 4×4 is processed in a single clock cycle. A 1D SPIHT was presented in [22], which exploits parallelism to achieve high throughput. High throughput is reached; however, the performance is reduced as it considers 1D-DWT instead of 2D-DWT. A low-power FPGA implementation of the DCT-based Cordic-Loeffle (CL-DCT) algorithm in comparison to JPEG was provided in [23].

From the literature, it follows that a low-power, performance-efficient algorithm must be put forward. The key contributions of this work include the following:

1. A state table-based SPIHT (STS) algorithm is proposed that provides:
   a. Performance similar to that of SPIHT at low bit rates.
   b. Performance higher than SPIHT at high bit rates.
   c. Much lower memory requirement than SPIHT.
2. FPGA implementation of STS algorithm that provides:
   a. FPGA area utilization less than SPIHT and SPIHT-based architectures.
   b. Dynamic power dissipation comparable to that of JPEG architectures.

The paper is organized as follows. Section 2 briefly describes the traditional SPIHT algorithm. The proposed algorithm is presented in Section 3. Section 4 provides the memory required by STS and other algorithms. Section 5 provides the FPGA implementation of the proposed algorithm. The results and discussion are provided in Section 6. Finally, the work is concluded in Section 7.

2. SPIHT algorithm

The SPIHT algorithm is a spatial orientation tree (SOT)-based compression scheme. As the image is decomposed into many subbands using DWT, there is a correlation between different subbands at different levels of decomposition. The correlation is such that a parent node in a higher dyadic level is said to be related to four offspring nodes in the next lower dyadic level. This correlation is exploited in SPIHT by observing that if a parent node is insignificant at a particular threshold, it is most probable that the descendant nodes are also insignificant. Each node is checked for significance at every threshold value. However, if descendants are found insignificant, not all the bits corresponding to each coefficient are generated, and only a single '0' may represent the whole tree or a branch of it.

Three lists are used by SPIHT while encoding/decoding an image: the list of significant pixels (LSP) list of insignificant pixels (LIP), and list of insignificant sets (LIS). LSP and LIP contain the addresses of the coefficients, while LIS contains the addresses of the parent nodes that represent its descendants. Initially each node in LIS is a Type-A node, which means that its descendants are all insignificant and need to be checked at the current threshold.

If a node in LIS has significant descendants, then the offspring nodes are moved to LSP or LIP according to their significance with respect to the threshold. The node in LIS is moved behind all entries in LIS and

treated as Type-B, which implies that all the descendants except the offspring are insignificant and need to be checked at the current threshold. If the descendants are found significant, a '1' is output and the offspring nodes are added to LIS as Type-A.

These lists are updated each time a node is checked. This makes SPIHT an undesirable compression technique, as the number of entries in these lists in certain cases exceeds the total number of coefficients. A large memory requirement is imposed by SPIHT, which makes its hardware implementation inefficient. Besides high memory requirements, power dissipation in SPIHT also increases as the memory size increases, because block RAMs in FPGAs consume a significant amount of power.

3. Proposed algorithm

The proposed algorithm is put forth primarily to eradicate the need for a large dynamic memory, which limits the use of SPIHT, despite providing very high compression efficiency. Our proposed algorithm is based on state-tables that preserve the status of each block of coefficients. The following terms are used in the 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) Sn(x): Significance of node x at current threshold.

The DWT transformed image is stored in Morton scan order [24]. Using Morton scan order, it is easier to navigate through a SOT tree. For any parent node located at x, the offspring nodes are located at locations 4x, 4x+1, 4x+2, and 4x+3. In STS, the image is divided into blocks of size 2 $\times$ 2. Unlike SPIHT, these blocks are treated as nodes of the SOT tree. Two state-tables, SIG_B and SIG_D, are used to hold the status of the nodes. SIG_B reflects if a node is significant or not at any given threshold. SIG_D reflects if any descendant node is insignificant or not. SIG_B and SIG_D are also arranged in Morton scan order. For an image of size R $\times$ C, the sizes of SIG_B and SIG_D in number of memory bits are given in Eq. (2) and Eq. (3), respectively. Two passes, a refinement pass (RP) and sorting pass (SP), are employed by the proposed algorithm. While encoding a bit-plane, the SP is always preceded by the RP. The blocks representing the highest level of DWT are initialized as significant blocks (SIG_B = '1'), while all other blocks are treated as insignificant (SIG_B = '0'). SIG_D is initialized as '0' for all blocks. The initial threshold is set as the highest power of 2, just below the largest coefficient in the transformed image.

3.1. Refinement pass

In RP, the significant blocks are encoded for the current bit-plane. Each coefficient is checked for significance at the current threshold. If a coefficient is significant at the current threshold, a bit '1' is output; otherwise, a bit '0' is output. If a coefficient was insignificant previously and becomes significant at the current threshold, a sign bit is also output. The image is encoded in RP in breadth-first search mode, i.e. the blocks in a higher level of DWT are always encoded before the blocks in a lower level of DWT.

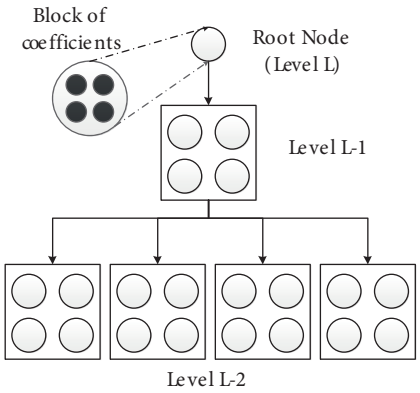


Figure 1. A SOT tree in STS for L-level DWT.

3.2. Sorting pass

In SP, a block is treated as significant if at least one of its coefficients is greater than the current threshold or if any of its descendant nodes are significant. It is observed that the initial few passes do not yield any significant coefficient in SP. The proposed algorithm skips SP for these initial passes wisely. The STS uses a depth-first search for SP. Depth-first limits the number of coefficients that actively take part in the encoding process at any given time. Each SOT is treated independently, and hence only the nodes of one SOT tree are considered for filling up the lists. Two small lists, a list of child blocks (LCB) and list of parent blocks (LPB), are employed to store the nodes of the SOT tree. The LCB contains the child nodes and its maximum size is fixed, given by Eq. (4). The LPB contains the parent nodes and its maximum size is also fixed, given by Eq. (5). Both of these lists are used to update the state-tables. Each SOT tree has its roots in the highest DWT level as shown in Figure 1. If SIG_D is '1' for a root node, then all the nodes of the SOT are significant already and are encoded in RP. Otherwise, the SOT tree is checked for the current threshold in SP. Each SOT tree is encoded in STS as follows:

A. If SIG_D(root) = '0'

a. add root to LCB

B. For each entry in LCB having SIG_B = '0'

a. If Sn(node) = '0' AND Sn(descendants) = '0'

i. Output '0'

ii. Remove the entry from LCB

b. Else

i. Output '1'

ii. Output encoded bits of node

iii. SIG_B(node) = '1'

iv. If Sn(descendants) = '1'

- *Output '1'*

- *Move node entry to LPB from LCB*
- *Add offsprings to LCB*

v. *Else*

- *Output '0'*
- *Remove entry from LCB*

C. *For each entry in LCB having SIG_B = '1'*

a. *For each offspring having SIG_B = '0'*

i. *If Sn(descendants) = '1'*

- *Output '1'*
- *Move entry from LCB to LPB*

ii. *Else*

- *Output '0'*
- *Remove entry from LCB*

b. *For each offspring having SIG_B = '1'*

i. *If SIG_D(offspring) = '0'*

- *Move entry from LCB to LPB*

ii. *Else*

- *Remove from LCB*

D. *For each entry in LPB*

a. *If SIG_D(each offspring) = '1'*

i. *SIG_D(entry) = '1'*

b. *Remove entry from LPB*

4. Memory requirement

The input image is first transformed into DWT coefficients. For a transform block size of R × C and each coefficient represented by W bits, the memory (bits) required is presented in Eq. (1). The memory required by the two state tables is given in Eqs. (2) and (3), respectively. The maximum fixed size of the two lists is given in Eqs. (4) and (5), respectively. L represents the DWT transformation level.

$$M_{Transformed\_Image} = RCW \tag{1}$$

$$M_{SIG\_B} = RC/4 \tag{2}$$

$$M_{SIG\_D} = RC/16 \tag{3}$$

Table 1. Dynamic memory requirements (Mbits) by different algorithms in addition to the memory required by the image itself corresponding to different image sizes.

Algorithm	Size (Mpixels)									
	4	8	12	16	20	24	28	32	36	40
SPIHT	18.13	36.25	54.38	72.50	90.63	108.75	126.88	145.00	163.13	181.25
WBTC	17.22	34.44	51.66	68.88	86.09	103.31	120.53	137.75	154.97	172.19
LBTC	2.00	4.00	6.00	8.00	10.00	12.00	14.00	16.00	18.00	20.00
JP2K*	0.46	0.65	0.80	0.92	1.03	1.13	1.22	1.30	1.38	1.46
BTCA*	0.71	1.01	1.23	1.42	1.59	1.74	1.88	2.01	2.13	2.25
STS	0.15	0.31	0.47	0.62	0.78	0.93	1.09	1.25	1.40	1.56

*At bitrate = 1 bpp.

$$M_{LCB} = (4 + 3(L-2)) \times log_2(RC) \tag{4}$$

$$M_{LPB} = (1 + \sum_{i=1}^{L-2} 4^n) \times log_2(RC) \tag{5}$$

The total number of memory bits needed by the algorithm in addition to the image itself is given in Eq. (6) below:

$$M_{STS} = (5 + 3(L-2) + \sum_{i=1}^{L-2} 4^n) \times log_2(RC) + RC \times (21/6) \tag{6}$$

Here, $log_2(RC)$ gives the number of bits needed to address the image.

The memory requirements by STS in comparison to different algorithms is presented in Table 1 for various image sizes. It is obvious from the table that, as the image size increases, the memory size also increases. Column 2 to Column 11 show the memory requirement corresponding to different image sizes ranging from 4 Mpixels to 40 Mpixels. It should be noted that the memory requirement for JPEG 2000 and BTCA are given for the bitrate of 1 bpp. Surely, the memory requirement for these algorithms will increase as the bitrate increases. It is evident that the memory size required by STS is very small in comparison to the algorithms. On average, STS requires only 0.86%, 0.88%, 7.6%, 45%, and 82% of the memory required by SPIHT, WBTC, LBTC, BTCA, and JPEG 2000, respectively.

5. Hardware Design

In this section, the hardware design of the proposed image compressor is presented. The implementation has been targeted for FPGAs. The FPGA platform has been chosen given that it provides reconfigurability besides high performance. The top level architecture of the STS compressor is shown in Figure 2. The image to be encoded is either input as a whole or it is first partitioned into many subimages called tiles and then input to the compressor. The DWT stage takes a three level 2-D DWT transformation using Cohen–Daubechies–Feauveau (5,3). The transformed block from DWT is then compressed using the proposed STS encoder. In the

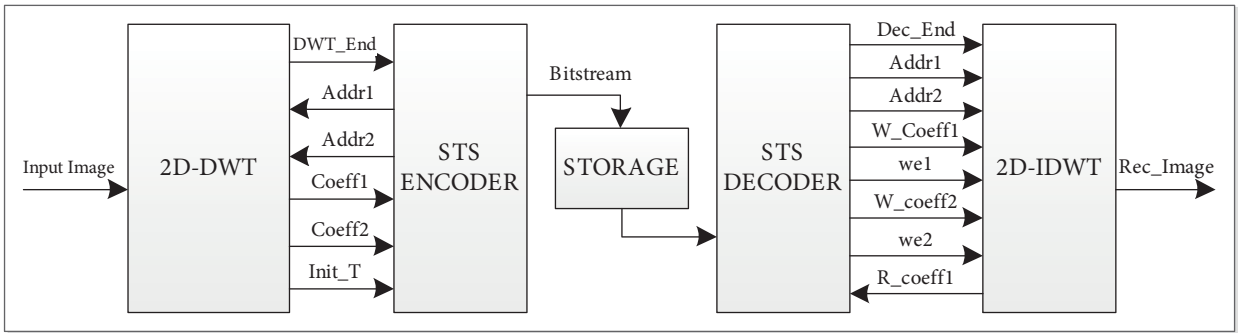


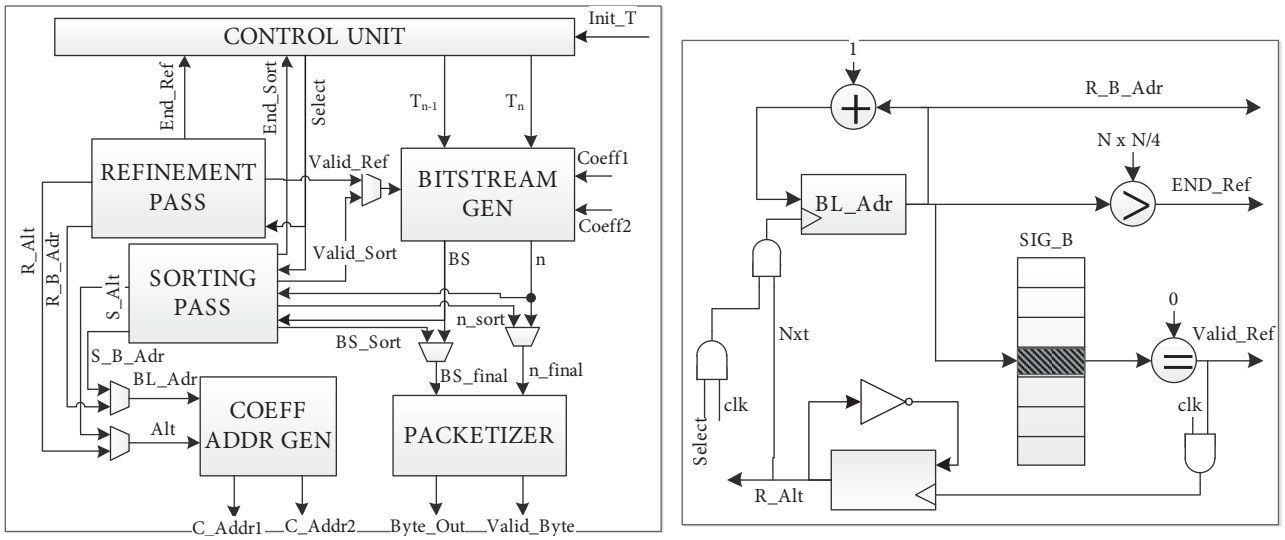
Figure 2. Top-level architecture of STS compressor.

architecture, it can be observed that the wavelet transformed block is stored in a dual-port memory within the DWT stage and the corresponding coefficients are made available to the STS encoder, as and when it needs. Also, the initial threshold is calculated in the DWT stage and made available to the STS encoder. When the tile is completely transformed, the *DWT_End* signal is set high and the STS starts processing the transform block. The decoder works in the reverse way. The STS decoder decodes the bitstream generated by the encoder and builds the transform block, which is then inverse transformed to recover the image tile. The transform block is stored in the memory located in the inverse DWT stage, 2D_IDWT. Coefficients are read from the memory using the signal *R_coeff1* and then updated and written back using the signals *W_coeff1*, *W_coeff2*, *we1*, and *we2*.

5.1. STS encoder

The top level architecture for the STS encoder is shown in Figure 3a. It mainly consists of two passes, REFINEMENT PASS (RP) and SORTING PASS (SP). The resources used by the modules BITSTREAM GEN, COEFF ADDR GEN, and PACKETIZER are shared between the two passes. These resources are utilized by one of the two passes at a time, decided by the CONTROL UNIT. As mentioned in Section 3, the proposed encoder encodes the image in a block-based manner. The specific block addresses corresponding to those blocks that need to the encoded as per the algorithm are produced by the two passes. The COEFF ADDR GEN module produces the desired coefficient addresses, which are then sent to the transform block memory to retrieve the corresponding coefficients. A *Valid* signal is sent to the BITSTREAM GEN module to prevent it from encoding the coefficients that do not meet the requirements to be encoded. The BITSTREAM GEN receives the coefficients and compares them with the current threshold T_n and previous threshold T_{n-1} to encode them. Two coefficients are encoded per clock cycle and hence a bitstream of width up to 4 bits is generated. If the encoding control is with SP, the bitstream BS is made available to the SORTING PASS module, where the addressing bits are appended to the bitstream to make it possible for the decoder to recognize the coefficients that were encoded. Finally, the PACKETIZER module reorganizes the bitstream and produces a byte-long output, which can then be stored in off-chip memory.

The RTL schematic for REFINEMENT PASS is given in Figure 3b. *Select* is used to enable any of the two passes at a time. In RP, if the SIG_B value corresponding to the block address is '0', then the signal *Valid_Ref* is set high so that the coefficients belonging to the block be encoded by the subsequent modules. Otherwise, the address is incremented by 1 and checked again. If the last block address is reached, *End_Ref* is set high, so that the CONTROL UNIT can activate the SP. In SP, the transform block is encoded by taking one SOT tree at a time, as shown in Figure 4. A SOT tree encapsulates blocks from all three levels of DWT.



(a) STS ENCODER (b) REFINEMENT PASS

Figure 3. STS encoder: (a) top level architecture, (b) RTL schematic for refinement pass.

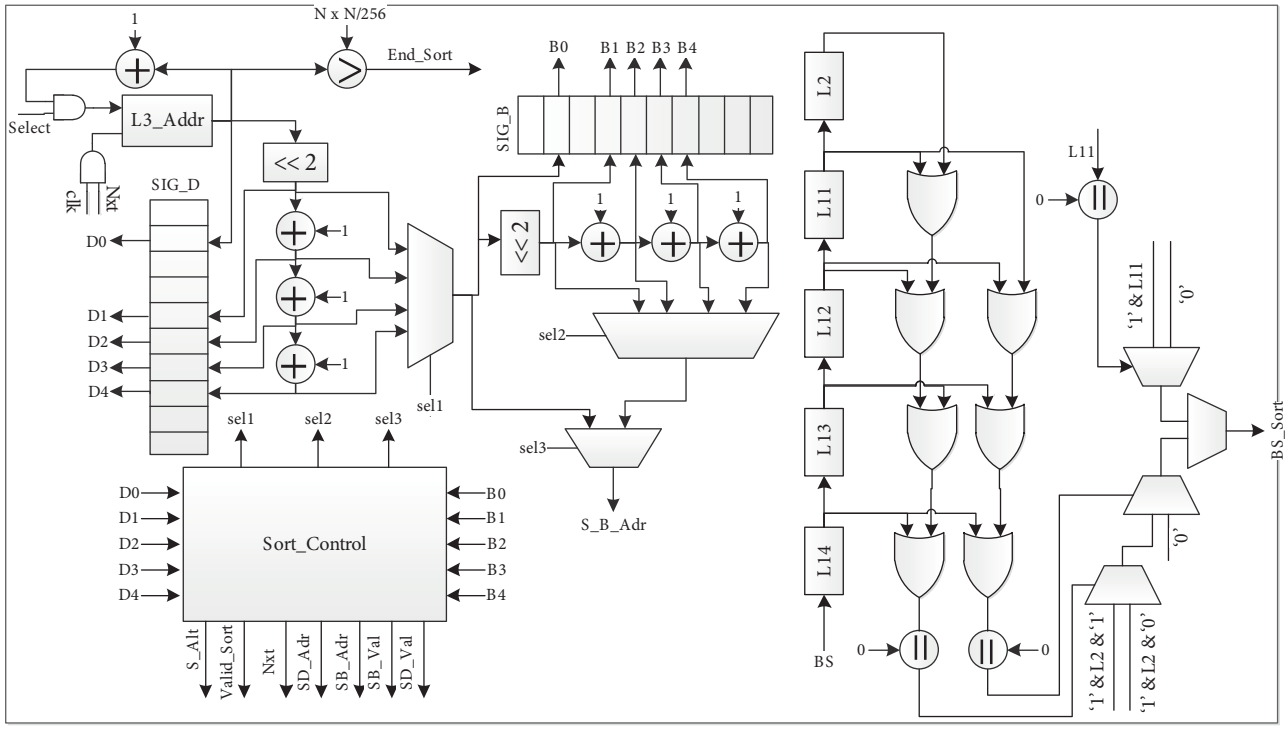
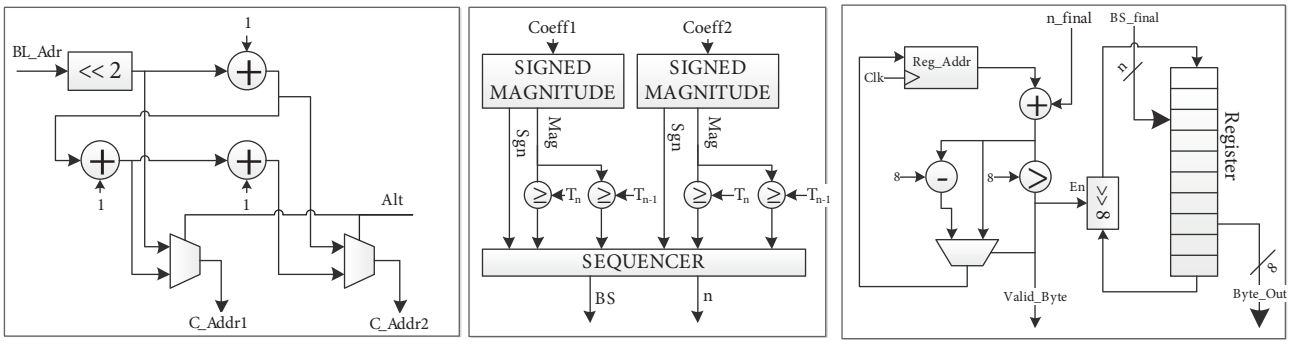


Figure 4. RTL schematic for sorting pass.

From the figure, it can be seen that the block addresses from all the three levels are calculated and checked. The block address from level L3, *L3_Addr*, is used to calculate its offspring in level L2, which are located at $4 \times L3\_Addr$, $4 \times L3\_Addr + 1$, $4 \times L3\_Addr + 2$, and $4 \times L3\_Addr + 3$. The entries in SIG_D corresponding to the level 2 and 3 addresses are checked by Sort_Control and accordingly a decision is made on whether to take the SOT tree or not. If the SOT tree is taken, it is also decided which of the branches originating from



(a) COEFF ADDR GEN (b) BITSTREAM GEN (c) PACKETIZER

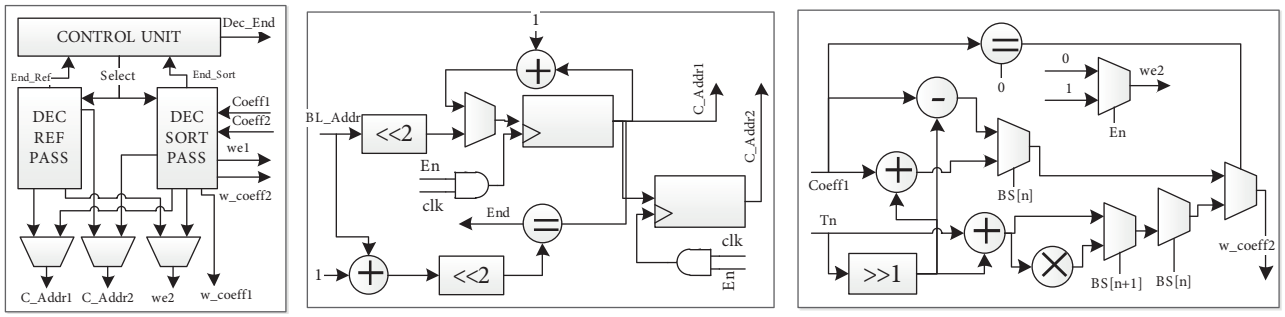
Figure 5. RTL schematic.

level 2 is to be taken. When a branch is taken, SIG_B is used to decide which of the blocks need to be encoded. A block needs to be encoded if it is not previously significant. The control signals *sel1*, *sel2*, and *sel3* are generated by Sort_Control to manage the right sequence of encoding the SOT tree. When a branch is encoded, the bitstream corresponding to the level L2 node and its four offspring nodes from level L1 are obtained from BITSTREAM GEN. Consider the corresponding BS values to be stored in registers L2, L11, L12, L13, and L14. If any of the bitstreams are found to contain values other than zero (significant coefficient), then a '1' followed by BS corresponding to L2 are output. Furthermore, if all of the level L1 bitstreams L11, L12, L13, and L14 are found to be insignificant, then a '0' follows. Otherwise, a '1' follows, which is further followed by encoded bits from each of the level L1 blocks. For convenience, the encoding of only one block from level L1 (L11) is shown in the RTL schematic.

The RTL corresponding to COEFF ADDR GEN is shown in Figure 5a. The coefficient addresses are generated from the block address *BL_Addr*. The *BL_Addr* is shifted by two bits to the left to get the coefficient located at 4 × BL_Addr. Two coefficient addresses are generated per clock cycle. The *Alt* generated by RP and SP is used to select between the pair of coefficient addresses. In one clock cycle, one pair of addresses is selected, while in the next clock cycle, the remaining pair of coefficient addresses are selected. The bitstream BS is generated from the BITSTREAM GEN module with the RTL shown in Figure 5b. The coefficient value is first converted into signed magnitude form and then its magnitude is compared with the current and previous thresholds to encode the coefficient. If the coefficient was previously insignificant and becomes significant at the current threshold, then a sign bit *'sgn'* is appended with the significance bit. The encoded bits from the two coefficients are attached together and output in BS. The number of valid bits in BS is given by n, which at maximum can be 4. The PACKETIZER receives the bitstreams from BITSTREAM GEN or SP and then reorganizes the bits to produce output in byte-long packets, as shown in Figure 5c. The BS values are buffered into a temporary register. When the contents exceed 8 bits, a byte is output and the rest of the contents are shifted by 8 bits. The *Reg_Addr* pointing to the highest valid bit in the register is subtracted by 8 as the address exceeds 8.

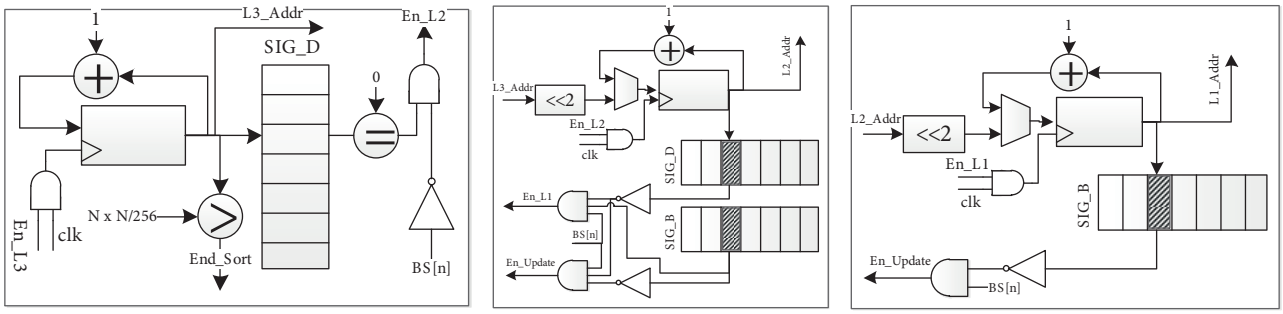
5.2. STS decoder

The architecture of the STS decoder is the reverse of the STS encoder, as shown in Figure 6a. Here the bitstream is used to reconstruct the transform block. Figure 6b shows the schematic for the STS decoder refinement address generator. *C_Addr1* is used to access the coefficient from the memory located in the IDWT module. In the next clock cycle, the coefficient is updated and written back into the same location from where



(a) STS DECODER (b) DEC REF ADDR (c) DEC REF UPDATE

Figure 6. STS decoder: (a) top level architecture, (b) RTL scheme for block address generator in refinement pass, (c) RTL scheme for coefficient update in refinement pass.



(a) L3 ADDR (b) L2 ADDR (c) L1 ADDR

Figure 7. RTL scheme for block address generation sorting pass.

it was accessed. That is why *C_Addr2* is a delayed version of *C_Addr1*. From Figure 6c, it is clear that the updating of a coefficient depends on the bitstream. BS[n] is the current bit from the bitstream that is to be exhausted by updating the coefficient, and BS[n+1] is the next coefficient that follows BS[n]. Depending on the coefficient value (whether it is already significant or not) and these two BS bits, the coefficient may be either written back as it is, added or subtracted with $0.5 \times T_n$, or replaced with $1.5 \times T_n$ or $-1.5 \times T_n$. When a coefficient becomes significant, it is replaced by $1.5 \times T_n$ or $-1.5 \times T_n$ and afterwards bits '0' and '1' correspond to subtracting and adding the coefficient magnitude with $0.5 \times T_n$ value, respectively.

Like the encoder sorting pass, three addresses are needed in the decoder sorting pass, as well, as shown in Figure 7. Only one of the three address generators L3_ADDR, L2_ADDR, and L1_ADDR is enabled at any time. Whether a SOT tree is selected or not depends on the SIG_D and the next bit from BS, as shown in Figure 7a. When a particular block is selected, a block address to coefficient address generation takes place in the same way as in the STS encoder. However, the coefficient is not updated the same way as in the decoder refinement pass. In a sorting pass, the coefficients to be updated are all previously insignificant. Hence, the coefficient locations are written only with $1.5 \times T_n$ or $-1.5 \times T_n$, as shown in the Figure 8. Two coefficients are updated in a single clock cycle. The first coefficient, *w_coeff1*, depends on the two next BS bits (BS[n] and BS[n+1]), while the second coefficient, *w_coeff2*, depends on two more following bits as well (BS[n], BS[n+1], BS[n+2], and BS[n+3]).

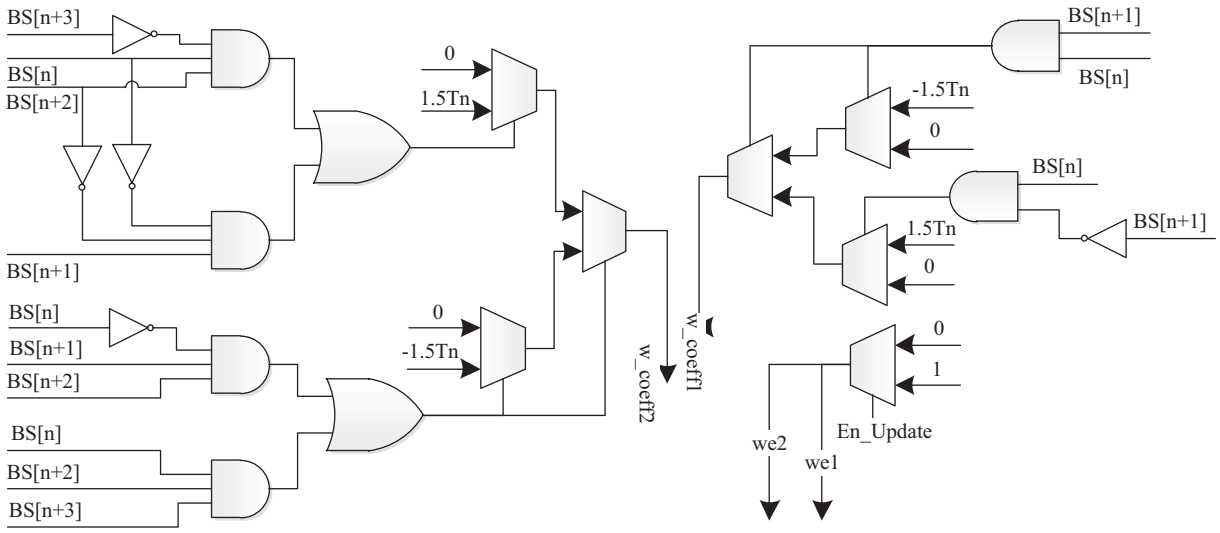


Figure 8. RTL scheme for DEC SORT UPDATE.

Table 2. FPGA resources of proposed STS in comparison with other implementations.

Identifier	Name	# Images	Resolution	Size (Mpixels)
Data Set 1	Canon	14	2432 × 3648	8.5
Data Set 2	Sony	14	1944 × 2592	4.8
Data Set 3	Pixabay	5	4000 × 6000	22.9

6. Results and discussion

The main attraction of this work is to reduce the FPGA area and dynamic power dissipation of the proposed STS architecture below that of JPEG and provide compression efficiency better than SPIHT. For performance evaluation, images from two databases are used. These are the DSLR Photo Enhancement Dataset (DPED)[1] [25] and Pixabay[2], as shown in Table 2. The colored images are converted to monochrome images before encoding. For coding performance, peak signal-to-noise ratio (PSNR) has been compared against bitrate. The PSNR and bitrate are expressed by the following relations [26]:

$$PSNR(dB) = 10\ log_{10}(255^2/MSE) \tag{7}$$

$$MSE = \frac{\sum_{i=1}^{R}\sum_{j=1}^{|l|}(x_{i,j} - y_{i,j})}{R \times C} \tag{8}$$

$$Bitrate(bpp) = \frac{Number\ of\ output\ bits}{R \times C} \tag{9}$$

Table 3 shows the PSNR of the proposed algorithm in comparison to SPIHT, JPEG, and JPEG 2000 averaged over the images from Test Set 2. It is evident from the table that initially the PSNR value of STS

[1]http://people.ee.ethz.ch/ ihnatova/
[2]https://pixabay.com/en/

Table 3. Average PSNR (dB) of proposed STS in comparison to JPEG, JPEG 2000, and SPIHT for Test Set 2 (STS without arithmetic encoding).

Algorithm	Bitrate (bpp)							
	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0
JPEG	36.52	41.14	43.82	45.87	49.41	52.24	55.34	57.10
SPIHT	38.10	43.65	46.99	49.34	51.28	54.36	57.08	58.36
JPEG 2000	39.11	44.84	47.99	49.64	51.98	55.02	58.30	61.68
STS	37.67	43.42	47.21	50.11	52.69	55.73	59.13	63.14

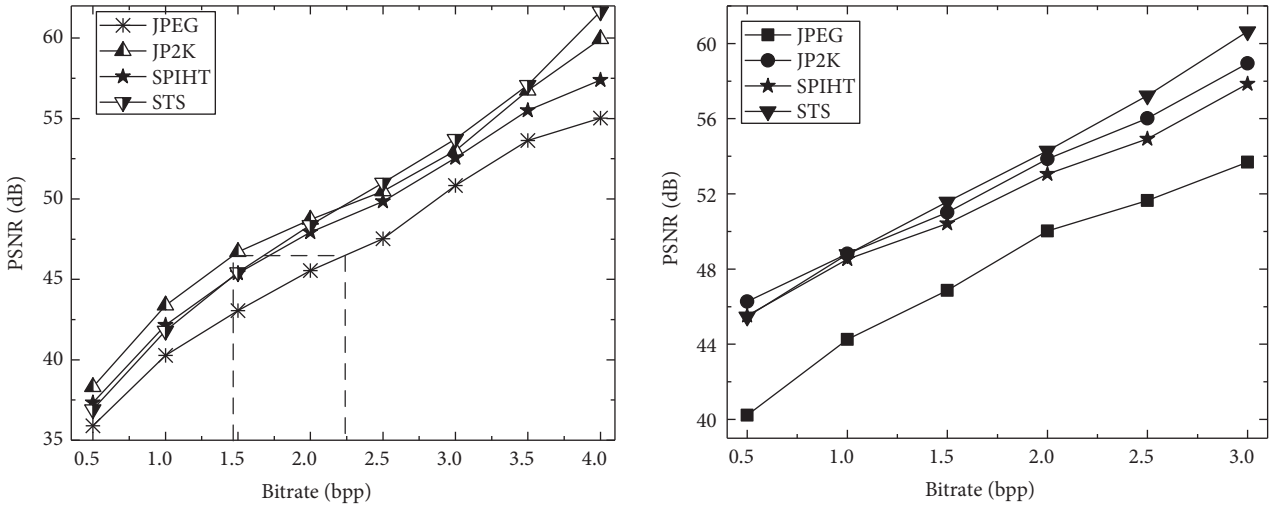


Figure 9. Average PSNR (dB) comparison of proposed STS algorithm in comparison to JPEG, JPEG 2000, and SPIHT algorithms for (a) Test Set 1 and (b) Test Set 3 (STS without arithmetic encoding).

Table 4. FPGA resources of proposed STS in comparison with different implementations.

Architecture	Technology	Logic cell (no.)	Memory (Kbit)
NLS [17]	Virtex 2-4	10,125	432
Bit-plane Parallel SPIHT [20]	Virtex 2000E	83,808	N/A
BPS [21]	Virtex 5-LX330	22,996	8.3
1D SPIHT [22]	Virtex 6-LX75T	16,621	0
MQ Coder [7]	Virtex 4-LX80	15,692	4.17
ABRC [8]	Virtex 4-LX80	1688	0
JPEG [1]	Spartan 3-S200	2711	N/A
CL-DCT [23]	Spartan 3-S200	2385	N/A
STS [this work]	Zynq Z-7020	1017	0

is better than JPEG but lower than SPIHT and JPEG 2000. As the bitrate increases, STS surpasses SPIHT and JPEG 2000 one after another. It should be noted that the results for STS are evaluated without using the

Table 5. Dynamic power dissipation for proposed STS in comparison to JPEG and JPEG 2000-based architectures.

Architecture	MQ Coder	ABRC	CL-DCT	STS
Frequency (MHz)	48.30	105.92	66.4	96
Dynamic power (mW)	488.67	127.05	96	74
Normalized power (mW/MHz)	10.117	1.19	1.45	0.77
Power density (μW/(MHz $\times$ Logic cell no.))	0.65	0.71	0.60	0.75

arithmetic encoder, which can further increase the PSNR by 0.3 dB to 0.6 dB.

Figure 9 shows the performance of STS against JPEG, JPEG 2000, and SPIHT for Test Sets 1 and 3 respectively. The STS surpasses SPIHT and JPEG 2000 at around 1.5 bpp and 2.25 bpp, respectively, for Test Set 1. From the figure, it is evident that the PSNR of STS is 2 dB to 6 dB higher than JPEG. It can also be seen that for any PSNR value, STS requires a bitrate approximately 0.75 bpp lower than JPEG. In other words, 22.5 Mbits of memory can be saved in order to store a 30 Mpixel image with STS instead of JPEG. This shows how important of a role STS can play in digital cameras, if it replaces JPEG.

Table 4 provides the FPGA implementation results of STS in comparison to different architectures. The first column gives the different architectures, based on JPEG 2000, SPIHT, and JPEG. The second column provides the technology used. All of the architectures are implemented on Xilinx FPGA boards. The third column provides the details about the area used. Number of logic cells used is the best parameter available to fairly compare the resources used by different FPGA technologies. The STS and other variants of SPIHT use 16 $\times$ 16 transform block size, except NLS, which uses 64 $\times$ 64 transform block size. The JPEG 2000-based architectures also use a 16 $\times$ 16 code block. Any increase in image size does not affect the hardware resources of STS at fixed transform block size. From the results, it is evident that STS not only consumes less resources than JPEG 2000 and SPIHT-based architectures, but also less than JPEG-based architectures. JPEG requires 2.7 times the logic cells used by STS, while no-list SPIHT (NLS) requires approximately 10 times more area than STS. The logic required by the arithmetic encoders used in JPEG 2000, MQ Coder [7] and ABRC [8], are 15 and 1.6 times more than STS. It should be noted that the logic required to implement the entire JPEG 2000 will be more than that required by MQ Coder and ABRC.

Finally, Table 5 provides the power dissipation comparison between STS, JPEG-based CL-DCT, and arithmetic coders MQ Coder and ABRC. For dynamic power dissipation comparison, both STS and CL-DCT are implemented on a Spartan 3 FPGA board. The second row gives the maximum operating frequency, which is 96 MHz for STS and only 66.4 MHz for CL-DCT. The third and fourth rows give the dynamic power dissipation and normalized power dissipation, respectively. It is evident that the power dissipation and its normalized value is lower in STS than that of CL-DCT, ABRC, and MQ Coder. STS has 96%, 35%, and 48% less normalized power dissipation in comparison to MQ Coder, ABRC, and CL-DCT, respectively. Finally, the power density is provided in the fifth row. STS has slightly more power density than CL-DCT but very close to that of ABRC.

It should be noted that the comparison with JPEG and CL-DCT is only made in order to understand the power and area efficiency of STS; otherwise, the compression efficiency of both these algorithms is very low in comparison to STS. Comparison is also made with the binary arithmetic encoders used in JPEG 2000 and it is clear that STS requires less FPGA area and dissipates less dynamic power. Unfortunately, no variant of SPIHT has provided power dissipation for FPGA implementation to date. The area used by JPEG is 2.7 times that of

STS, while it is 10 to 82 times that of SPIHT-based architectures. It is analogous that the power dissipation in SPIHT-based designs will be accordingly high.

7. Conclusion

A novel state-table-based SPIHT (STS) algorithm is proposed in this paper. The main focus of this algorithm is to provide high performance and low memory requirements. The performance of our proposed algorithm is higher than SPIHT at higher bitrates. The memory requirement is 0.88% that of SPIHT. The algorithm is implemented on Xilinx FPGAs. The FPGA area utilized by SPIHT-based architectures is approximately 10 to 80 times more than STS. Also, JPEG occupies approximately 2.7 times more area than STS. Most importantly, the power dissipation in JPEG is more than that in STS. All these features make the proposed compression algorithm a better candidate for DSLR and other cameras. STS provides better quality than JPEG, with less FPGA area and dynamic power dissipation.

References

[1] Wallace G. The JPEG still picture compression standard. IEEE T Consum Electr 1992; 38: 18-34.

[2] Hanhart P, Ebrahim T. Evaluation of JPEG XT for high dynamic range cameras. Signal Process-Image 2017; 50: 9-20.

[3] Yuan L, Sun J. High quality image reconstruction from RAW and JPEG image pair. In: IEEE 2011 International Conference on Computer Vision; 6–13 November 2011; Barcelona, Spain. New York, NY, USA: IEEE. pp. 2158-2165.

[4] Shapiro JM. Embedded image coding using zerotrees of wavelet coefficients. IEEE T Signal Proces 1993; 41: 3445-3462.

[5] Said A, Pearlman WA. A new, fast, and efficient image codec based on set partitioning in hierarchical trees. IEEE T Circ Syst Vid 1996; 6: 243-250.

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

[7] Liu K, Zhou Y, Li YS, Ma JF. A high performance MQ encoder architecture in JPEG 2000. Integration 2010; 43: 305-317.

[8] Belyaev E, Liu K, Gabbouj M, Li Y. An efficient adaptive binary range coder and its VLSI architecture. IEEE T Circ Syst Vid 2015; 25: 1435-1446.

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

[10] Pearlman WA, Islam A, Nagaraj N, Said A. Efficient, low-complexity image coding with a set-partitioning embedded block coder. IEEE T Circ Syst Vid 2004; 14: 1219-1235.

[11] Senapati RK, Pati UC, Mahapatra KK. Listless block-tree set partitioning algorithm for very low bit rate embedded image compression. AEU-Int J Electron C 2012; 66: 985-995.

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

[13] Su CY, Wu BF. A low memory zerotree coding for arbitrarily shaped objects. IEEE T Image Process 2003; 12: 271-282.

[14] Jyotheswar J, Mahapatra S. Efficient FPGA implementation of DWT and modified SPIHT for lossless image compression. J Syst Architect 2007; 53: 369-378.

[15] Philip D. VLSI architecture for real-time image and video processing systems. J Real-Time Image Pr 2006; 1: 57-62.

[16] Kuon I, Rose J. Measuring the gap between FPGAs and ASICs. IEEE T Comput Aid D 2007; 26: 203-215.

[17] Wheeler FW, Pearlman WA. SPIHT image compression without lists. In: IEEE 2000 International Conference on Acoustics, Speech, and Signal Processing; 5–9 June 2000; İstanbul, Turkey. New York, NY, USA: IEEE. pp. 2047-2050.

[18] Corsonello P, Perri S, Staino G, Lanuzza M, Cocorullo G. Low bit rate image compression core for onboard space applications. IEEE T Circ Syst Vid 2006; 16: 114-128.

[19] Cheng CC, Tseng PC, Chen LG. Multimode embedded compression codec engine for power-aware video coding system. IEEE T Circ Syst Vid 2009; 19: 141-150.

[20] Fry TW, Hauck SA. SPIHT image compression on FPGAs. IEEE T Circ Syst Vid 2005; 15: 1138-1147.

[21] Jin Y, Lee HJ. A block-based pass-parallel SPIHT algorithm. IEEE T Circ Syst Vid 2012; 22: 1064-1075.

[22] Kim S, Lee D, Kim JS, Lee HJ. A high-throughput hardware design of a one-dimensional SPIHT algorithm. IEEE T Multimedia 2016; 18: 392-404.

[23] Kaddachi ML, Soudani A, Lecuire V, Torki K, Makkaoui L, Moureaux JM. Low power hardware-based image compression solution for wireless camera sensor networks. Comp Stand Inter 2012; 34: 14-23.

[24] Artyomov E, Rivenson Y, Levi G, Pecht OY. Morton (Z) scan based real-time variable resolution CMOS image sensor. IEEE T Circ Syst Vid 2005; 15: 947-952.

[25] Ignatov A, Kobyshev N, Timofte R, Vanhoey K, Gool LV. DSLR-quality photos on mobile devices with deep convolutional networks. In: IEEE 2017 International Conference on Computer Vision; 22–29 Oct 2017; Venice, Italy. New York, NY, USA: IEEE. pp. 3297-3305.

[26] Granrath DJ. The role of human visual models in image processing. P IEEE 1981; 69: 552-561.