A Low-Cost Hardware Design of a 1-D SPIHT Algorithm for Video Display Systems

Xuan Truong Nguyen, Hyuk-Jae Lee, *Member, IEEE*, and Hyun Kim, *Member, IEEE*

***Abstract*—This paper presents low-cost compressors for video display systems, which compress video data to reduce the size of the frame memory in a display panel in a television or a smartphone. The hardware solution exploits the discrete wavelet transform (DWT) and 1-D set partitioning in hierarchical trees (SPIHT) to support the raster-scan processing order in a display system and meet the fixed target compression ratio (CR). However, the compression efficiency and hardware cost often increase in proportion to the block size and thus, designing a compressor with low cost but high fidelity is challenging. This paper proposes a generic partitioned SPIHT algorithm, which achieves a low-cost design for mobile devices by allowing various sizes of partitioned sub-blocks. By taking advantage of DWT decomposition properties, the proposed algorithm partitions a coding block into sub-blocks, each of which can be processed independently. The independent coding operations among multiple sub-blocks allow the size of the hardware buffer to be decreased, by storing the temporary results only for an individual sub-block. However, the proposed sub-block coding slightly decreases the compression efficiency when each sub-block is compressed with the same target CR. To reduce the drop-off in the compression efficiency, the boundary-pixel handling in DWT is exploited and the proposed partitioned SPIHT method adjusts the target CR for each sub-block depending on the potential quality loss. When compared with the previous 1-D design, experimental results show that the hardware gate count and internal memory are reduced by 59.34% and 75%, respectively. Furthermore, the proposed boundary handling and bit-allocation schemes mitigate the PSNR degradation due to the sub-block coding by 1.11 dB.**

***Index Terms*—1-D compression, discrete wavelet transform (DWT), frame memory compression, low-cost hardware design, set partitioning in hierarchical trees (SPIHT), video display systems.**

I. Introduction

IN RECENT years, video display systems, such as in smart and high-resolution televisions or mobile devices, have become increasingly popular. A display system usually stores video data for display in an off-chip memory space (e.g., DRAM) [1]. When the display system supports a high-resolution, a huge bandwidth is required for this external memory access, as well as a large memory storage space [1], [2]. For instance, 4K UHD (3840 × 2160) televisions with three color components with a bit width of eight per a pixel require a frame memory of 25MB. In order to reduce the memory cost and bandwidth, display panel designers often attempt to reduce the amount of data stored in frame memory by compressing frame data [3]–[6]. An integration of a frame memory compression in video display systems is presented in Fig. 1. Frame compression algorithms for display systems have different requirements from the standard video compression, such as H.264 [7], [8]. First, they require a relatively small fixed target compression ratio (CR), ranging between three and six, as the memory size in a video display system is fixed. Second, they require a low hardware complexity, as the hardware cost overhead should not weaken the advantages of reducing memory storage and bandwidth. Finally, it is necessary to process frame data in a raster-scan order.

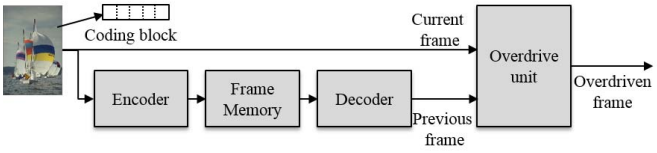

Fig. 1. LCD overdrive with a frame memory compression in display systems.

Several frame memory compression algorithms have been proposed for video display systems. In general, these are categorized into lossy and lossless compression methods [9]. Lossless compression approaches such as JPEG-LS [3] are simple and obviously achieve a high image/video quality. However, they mostly adopt variable length coding (VLC), and consequently, cannot meet a fixed target CR, as required in video display systems. To fit in a limited memory size, lossy compression approaches [3]–[6] have been proposed by exploiting an adaptive quantization threshold. In general, however, the threshold is unknown before encoding. Therefore, it is iteratively adjusted until its corresponding bit-stream meets a target bit length (TBL). Clearly, these iterative methods are not efficient for hardware implementation. Set partitioning in hierarchical trees (SPIHT) is a fast and high-fidelity compression algorithm for wavelet-transformed images [10]. It processes discrete wavelet transform (DWT) coefficients bit-plane by bit-plane in a descending order, from upper bit-planes

Manuscript received December 30, 2017; revised February 10, 2018; accepted February 15, 2018. Date of publication March 7, 2018; date of current version March 29, 2018. This work was supported in part by the Institute for Information and Communications Technology Promotion through the Korea Government (MSIT, Development of Intelligent Semiconductor Technology for Vision Recognition Signal Processing for Vehicle Based on Multi-Sensor Fusion) under Grant 2017-0-00721-001, and in part by the Research and Development Program of MOTIE/KEIT(Developing Processor-Memory-Storage Integrated Architecture for Low Power, High Performance Big Data Servers) under Grant 10077609. *(Corresponding author: Hyun Kim.)*

The authors are with the Inter-University Semiconductor Research Center, Department of Electrical and Computer Engineering, Seoul National University, Seoul 08826, South Korea (e-mail: truongnx@capp.snu.ac.kr; hyuk_jae_lee@capp.snu.ac.kr; snusbkh0@capp.snu.ac.kr).

Digital Object Identifier 10.1109/TCE.2018.2812059

to lower ones, and codes the more significant bits first. While achieving a high compression efficiency, it is easily terminated as the coded bit-stream reaches the TBL. In other words, SPIHT efficiently generates the coded bit-stream targeted for a fixed memory space. The conventional SPIHT [10] and its variations [11]–[15] target 2-D images, to take advantage of spatial data correlations in both horizontal and vertical directions. Hence, they assume that an image is partitioned into 2-D blocks or tiles. However, this assumption is not true in video display systems that display frames in the raster-scan order, because 2-D SPIHT-based approaches require a huge memory buffer to store additional lines and form 2-D blocks. For example, to compress a 16×16 block in a 4K UHD image requires storing 16 lines, or a buffer of 16×3840 pixels. Therefore, it is not efficient to employ these 2-D SPIHTs for video display systems.

The 1-D SPIHT algorithm, a modification of the conventional 2-D SPIHT, has been proposed for 1-D data, in order to reduce the large memory requirement described above. Originally, this method and its variations have been applied for wavelet electrocardiogram data [16], [17], as the input signal is naturally 1-D. Zhang and Zhang [18] investigated an unsymmetrical 1-D SPIHT algorithm for image processing. However, their work mainly focused on the algorithm, while a hardware implementation was not investigated. Kim *et al.* [19], [20] focused on the hardware implementations of 1-D SPIHT for video display systems. To improve the coding efficiency, they selected a relatively large block, with a size of 1×64. However, this large block size requires a lot of hardware resources (i.e., gate count and buffer) and is still a burden to be used as an embedded compression module in mobile devices.

To address this problem, this paper proposes a partitioned SPIHT method, which efficiently achieves a low-cost design. The proposed algorithm creates partitioned sub-blocks in coding operations, which are independent of each other. Therefore, the proposed algorithm compresses all bit-planes in a sub-block before it moves to the next sub-block, so that not all intermediate results for a sub-block are necessary to store after the sub-block coding is completed. In this manner, the buffer space for the intermediate results is shared by multiple sub-blocks, and consequently the total buffer size is significantly reduced. It should be noted that the proposed partitioned algorithm still maintains the raster-scan processing order of an input image and thus, it is suitable for low-cost designs because it does not require an input buffer.

Using small coding units can effectively reduce the buffer size and shorten the latency. However, encoding small coding units separately leads to local optimizations, and the coding efficiency may be degraded [11]. To address this problem, the method presented here utilizes two simple but efficient techniques. First, the proposed method utilizes the conventional 1-D DWT as in [19] and [20] instead of reducing the DWT block size into the sub-block size. Consequently, it efficiently performs the boundary handling with the boundary pixels of sub-blocks, which significantly improves the coding efficiency owing to sub-block coding. Second, this paper proposes a simple bit-allocation method, which effectively distributes the given TBL into four independent sub-blocks, and improves the coding efficiency. To this end, the proposed hardware implementation only requires 22.4K gates for an encoder and 25.9K gates for a decoder, which represents a reduction of 59.34% compared with the hardware design in [20]. Simulations with test images show that the two proposed schemes for coding efficiency enhancement mitigate the PSNR degradation due to the sub-block coding by 1.11 dB.

The remainder of this paper is organized as follows. Section II briefly introduces the DWT and SPIHT based compression algorithm, and Section III describes the proposed block-based SPIHT algorithm. Section IV presents an improvement of the block-based algorithm, by adjusting the TBL of each block according to its complexity. In Section V, the proposed algorithm is evaluated by presenting simulation results, and its complexity is estimated with a hardware implementation. Section VI presents the conclusions for this paper.

II. Conventional 1-D DWT and SPIHT Algorithms

This section briefly reviews 5/3 integer filtering 1-D DWT [21], [22] for the wavelet transform, and the direct modification of SPIHT for 1-D data [12] adopted in this paper.

A. Discrete Wavelet Transform

For an efficient hardware implementation of DWT, this paper adopts lifting-based filtering [20], which consists of a sequence of simple filtering operations. The lifting scheme consists of three stages: 1) splitting; 2) predicting; and 3) updating. In the first stage, the input sequence is divided into two subsets: 1) an even-indexed sequence and 2) an odd-indexed sequence. The even sequence is used to predict the odd one in the second stage. The difference between the odd sequence and the prediction value is calculated as the high coefficient. In the third stage, the even sequence is updated with the high-pass coefficient to compute the low-pass coefficient. In particular, the high-pass and low-pass coefficients [$H^j(i)$ and $L^j(i)$ at the jth level] are calculated from the low-pass coefficients at the previous level as follows:

$$H^j(i) = L^{j-1}(2i+1) - \left\lfloor \frac{L^{j-1}(2i) + L^{j-1}(2i+2)}{2} \right\rfloor \quad (1)$$

$$L^j(i) = L^{j-1}(2i) + \left\lfloor \frac{H^j(i) + H^j(i-1) + 2}{4} \right\rfloor \quad (2)$$

where i is the index of a coefficient for $0 \le i \le [(N/2^j) - 1]$ and $\lfloor a \rfloor$ indicates the largest integer not exceeding a.

When the processing unit of DWT is large, the output result is repeated with a given period, in the sense that the level and the index of the generated coefficient is same for each period. For example, the implementation of the three-level DWT with 32 input samples is illustrated in Table I. The first column shows the execution time, and the second column shows the input data, which is received at a rate of one data point per one cycle. The third column shows the generated output sequence forming a 32-pixel coding, which is similar to that in [19] and [20]. The index in the parenthesis represents the output DWT coefficient index. The fourth column shows

TABLE I
PERIODIC OPERATIONS IN THREE-LEVEL DWT

t	Input	Output	DWT Coeff.	t	Input	Output	DWT Coeff.
0	$x(0)$	-	-	18	$x(18)$	$c(1)$	$L^3(0)$
1	$x(1)$	$c(16)$	$H^1(0)$	19	$x(19)$	$c(22)$	$H^1(1)$
2	$x(2)$	-	-	20	$x(20)$	$c(10)$	$H^2(0)$
3	$x(3)$	$c(20)$	$H^1(1)$	21	$x(21)$	$c(26)$	$H^1(2)$
4	$x(4)$	$c(8)$	$H^2(0)$	22	$x(22)$	$c(6)$	$H^3(0)$
5	$x(5)$	$c(24)$	$H^1(2)$	23	$x(23)$	$c(30)$	$H^1(3)$
6	$x(6)$	$c(4)$	$H^3(0)$	24	$x(24)$	$c(14)$	$H^2(1)$
7	$x(7)$	$c(28)$	$H^1(3)$	25	$x(25)$	$c(20)$	$H^1(0)$
8	$x(8)$	$c(12)$	$H^2(1)$	26	$x(26)$	$c(2)$	$L^3(0)$
9	$x(9)$	$c(17)$	$H^1(0)$	27	$x(27)$	$c(23)$	$H^1(1)$
10	$x(10)$	$c(0)$	$L^3(0)$	28	$x(28)$	$c(11)$	$H^2(0)$
11	$x(11)$	$c(21)$	$H^1(1)$	29	$x(29)$	$c(27)$	$H^1(2)$
12	$x(12)$	$c(9)$	$H^2(0)$	30	$x(30)$	$c(7)$	$H^3(0)$
13	$x(13)$	$c(25)$	$H^1(2)$	31	$x(31)$	$c(31)$	$H^1(3)$
14	$x(14)$	$c(5)$	$H^3(0)$	32	$x(32)$	c(15)	$H^2(1)$
15	$x(15)$	$c(29)$	$H^1(3)$	33	-	-	-
16	$x(16)$	$c(13)$	$H^2(1)$	34	-	$c(3)$	$L^3(0)$
17	$x(17)$	$c(19)$	$H^1(0)$	-	-	-	-

the DWT decomposition level and the index in each level. The fifth to eighth columns also show the same as the first to the fourth columns, where the time starts from the 18th cycle in the fifth column. As the decomposition level is three, the output is repeated with a period of eight. For example, $H^1(0)$ is generated at cycles 1, 9, 17, and 25. The darkness level in Table I indicates the output in the same period. Owing to the repeating property, the same hardware generating the first eight coefficients is used repeatedly to generate the remaining sequence. This implies that the hardware cost does not increase, even though the length of the input image sequence does. Note that the repetition period depends on the DWT level. For four-level DWT, the repetition period is 16 ($= 2^4$). In general, n-level DWT generates an output sequence repeating with a period of 2^n.

B. Set-Partitioning in Hierarchical Trees

SPIHT is a fast and effective compression algorithm used for encoding DWT coefficients [10]. It processes in a bit-plane by bit-plane manner, from the most significant bit-plane down to the least significant. SPIHT performs a significance test on a set of wavelet coefficients organized in a tree structure. For a given set of coefficients T, the significance test $S_n(\cdot)$, which represents the result of the significance test for the nth bit-plane, is given as follows:

$$S_n(T) = \begin{cases} 1, & \max_{(c(i))\in T}\{|c(i)|\} \geq 2^n \\ 0, & \text{otherwise} \end{cases} \quad (3)$$

where $c(i)$ is the ith coefficient in T and 2^n is the threshold of the nth bit-plane (i.e., the significance of the bit-plane). The significance test classifies the set as a significant set, in which the maximum coefficient of the set is larger than the significance of the bit-plane. Otherwise, the set is classified as an insignificant set, and coded as a single bit "0." The significant set is divided into subsets, and the significance of each subset is tested again. When the subset has a single coefficient, the subset becomes a significant pixel or an insignificant pixel depending on the result of a significance test. The coding

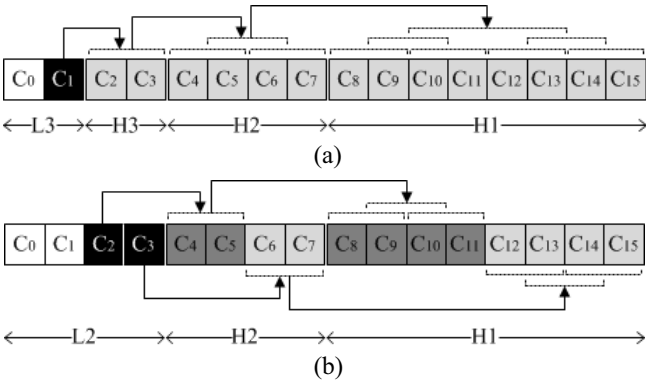

Fig. 2. Binary tree structure of DWT coefficients. (a) Decomposition level 3 and (b) decomposition level 2.

operation is terminated when all coefficients are coded or the encoded bit-stream length reaches the TBL.

SPIHT performs the significance test on the binary tree of coefficients. In this binary tree, a coefficient $c(i)$ has two offspring, which are the two coefficients $c(2i)$ and $c(2i+1)$. In turn, $c(2i)$ has two offspring $c(4i)$ and $c(4i+1)$. Therefore, $c(4i)$, $c(4i+1)$, $c(4i+2)$, and $c(4i+3)$ are also descendants of the coefficient $c(i)$. Fig. 2 shows the binary tree structure used in SPIHT for a block size of 1×16. Note that the input of SPIHT consists of DWT coefficients, and the tree structure depends on the DWT decomposition level. Fig. 2(a) and (b) shows the structures for levels three and two, respectively. In Fig. 2(a), the coefficient $c(0)$ corresponds to $L_3(0)$, which is the 0th low-band coefficient of level 3. The coefficient $c(1)$ corresponds to $L_3(0)$, which is the first low-band coefficient of level three. The coefficient $c(2)$ corresponds to $H_3(0)$, which is the 0th high-pass band coefficient of level three. The correspondences of the other coefficients are also presented in Fig. 2(a). The half of the coefficients in the low-pass band with the highest indexes form the root(s) of all the binary trees. For example, in Fig. 2(a), $L_3(1)$ is the only root of all the binary trees, which are indicated by the arrows in the figure. In Fig. 2(b), $L_2(2)$ and $L_2(3)$ [corresponding to $c(2)$ and $c(3)$, respectively] form the roots of the two binary trees.

The no-list SPIHT (NLS) algorithm shown in Fig. 3 is a modified version of SPIHT, in order to ease the hardware implementation by using a marker for each DWT coefficient to store its encoding state [12]. In NLS, every DWT coefficient is classified as one of three states, list of significant sets (LIS), list of insignificant pixels (LIP), or list of significant pixels (LSP). A coefficient is in the LIS state if the set consisting of itself and its descendent coefficients is insignificant. If a coefficient is not classified as an LIS state, then it is classified as either an LIP or LSP state. A coefficient is in the LIP state if it is insignificant. On the other hand, a coefficient is in the LSP state if it is significant. NLS processes in a bit-plane by bit-plane manner, and the algorithm shown in Fig. 3 presents the operation for a single bit-plane. In this algorithm, mark[i], magn[i], and sign[i] represent the state, magnitude, and sign of the ith coefficient $c(i)$, respectively.

NLS consists of three passes: 1) the refinement pass (RP); 2) insignificant pixel pass (IPP); and 3) insignificant set

```
Refinement Pass (RP)
1:    for i=0; i<L; i=i+1
2:        if mark[i] == LSP then
3:            output magn[i]                          // Magnitude bit

Insignificant Pixel Pass (IPP)
4:    for i=0; i<L; i=i+1
5:        if mark[i] == LIP then
6:            output magn[i]                          // Magnitude bit
7:            if magn[i] == 1 then
8:                output sign[i]                      // Sign bit
9:                mark[i] == LSP

Insignificant Set Pass (ISP)
10:   for i=0; i<L; i=i+2
11:       if mark[i] == LIS then
12:           output S(Ti)                            // Sorting bit
13:           if S(Ti) == 1 then                      // Partitioning
14:               mark[2i] = LIS
15:               mark[2i+2] = LIS
16:               for j= 0; j <2; j =j+1
17:                   output magn[i+j]                // Magnitude bit
18:                   if magn[i+j] == 1 then
19:                       output sign[i+j]            // Sign bit
20:                       mark[i+j] == LSP
21:                   else then
22:                       mark[i+j] == LIP
```

Fig. 3. 1-D NLS algorithm.

pass (ISP). RP processes the coefficients in the LSP state (lines 1–3 in Fig. 3). The magnitude bit of the current bit-plane for the coefficient is generated for the bit-stream (line 3). IPP processes coefficients in the LIP state (lines 4–9). The first step in this pass generates the magnitude bit (line 6), and then tests its significance (line 7). If it is significant, then the sign bit is generated (line 8) and the state is changed to LSP (line 9). Lines 14–23 describe the operation of ISP, which processes the coefficients in the LIS state. First, this tests the significance of the set that consists of the descendent coefficients of the current coefficient. Let $S(i)$ be defined as the significance of the set consisting of the coefficient $c(i)$ and all its descendant coefficients. Then, the first step of ISP tests $S(i)$. This value is "1" if the set is significant (i.e., the set includes at least one significant coefficient). On the contrary, this set is insignificant if all coefficients in the set are insignificant. If this set is insignificant, then only the value of $S(i)$ is generated, and the operation of ISP ends. If this set is significant, then its two offspring coefficients are marked as LIS (line 17). The next steps generate the magnitude bit and the sign bit, and change the status depending on the significance of the coefficient (lines 18–23). Note that the effective compression of SPIHT is achieved by ISP, which generates only a single bit for all the descendent coefficients if the significant test returns 0.

C. VLSI Design of SPIHT Algorithm and Its Challenges

Fig. 4 illustrates the conventional design for an embedded codec engine that uses DWT and SPIHT [11] with two main modifications. First, its input is under a YUV422 format instead of the general pixel in [11]. At every cycle, four pixels are continuously processed in the raster-scan order by the system. Therefore, an input buffer is not required. Second, the

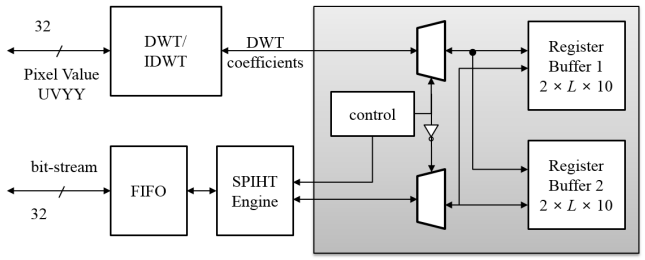

Fig. 4. General system architecture for an embedded codec engine.

size of the buffers between DWT and SPIHT is determined by parameter L, which refers to the size of a coding unit. Similar to [11], dual buffers are used to avoid bubble cycles. In addition, each buffer has a doubled size to simultaneously handle the block with luminance (Y) and chrominance (UV) components. The design challenge lies in how to define the value L when considering the tradeoff between the coding efficiency and hardware cost. The block size L is set to 64 in [19] and [20], which achieves a high coding efficiency, but suffers from a high hardware cost. However, this paper addresses the hardware cost problem in designing a small coding unit (e.g., $L = 16$). In other words, the coding efficiency gap among different coding block units is addressed. Note that in [11] this gap is not clearly shown, as the small coding unit is relatively large, having 64 ($= 8 \times 8$) coefficients, while the CR is two or four.

III. Proposed Partitioned SPIHT Algorithm

This section presents the partitioned SPIHT algorithm to reduce the buffer size and the corresponding scheduling.

A. Partitioned SPIHT Algorithm

Fig. 5 shows the proposed block-based NLS algorithm. The input sample is partitioned into sub-blocks of size b, which are processed independently. The operation is identical for all sub-blocks (lines 2–26), so that this is simply repeated as the outer-most loop (line 1). The partitioned SPIHT method in this paper can be considered as a generalized version of the four set-partitioning trees in [11]. The sub-block size is determined by the DWT decomposition level, so that sub-blocks have an identical format. For a given DWT decomposition level of n, the target sub-block size of the proposed hardware implementation is 1×2^n, because the hardware resources can be efficiently utilized with this size (see the detailed explanation in Section III-B). The proposed partitioned algorithm is also different from the previous block-based pass-parallel SPIHT [15], in which each bit-plane is partitioned into smaller coding units. In particular, an 8×8 bit-plane is divided into four 4×4 bit-planes, and processed sequentially.

B. Modified Data Structure and Buffer Reduction

This section proposes a method to reduce the buffer size. The first buffer is used for storing the marker in the SPIHT module in Fig. 4. One way to reduce the buffer size is to reuse the buffer of one sub-block for the other sub-blocks. The

```
Outer loop over sub-block
1:    for k=0; i<L/b; k=k+1

Refinement Pass (RP)
2:    for m=0; m<b; m=m+1
3:        i = k*b+m
4:        if mark[i] == LSP then
5:            output magn[i]                          // Magnitude bit

Insignificant Pixel Pass (IPP)
6:    for m=0; m<b; m=m+1
7:        i = k*b+m
8:        if  mark[i] == LIP then
9:            output magn[i]                          // Magnitude bit
10:           if magn[i] == 1 then
11:               output sign[i]                      // Sign bit
12:               mark[i] == LSP

Insignificant Set Pass (ISP)
13:   for m=0; m<b; m=m+2
14:       i = k*b+m
15:       if mark[i] == LIS  then
16:           output S(Ti)                            // Sorting bit
17:           if S(Ti) == 1 then                      // Partitioning
18:               mark[k*b+2*m] = LIS
19:               mark[k*b+2*m+2] = LIS
20:               for j= 0; j <2; j =j+1
21:                   output magn[i+j]                // Magnitude bit
22:                   if magn[i +j] == 1 then
23:                       output sign[i +j]           // Sign bit
24:                       mark[i +j] == LSP
25:                   else then
26:                       mark[i +j] == LIP
```

Fig. 5. Proposed portioned-SPIHT algorithm.

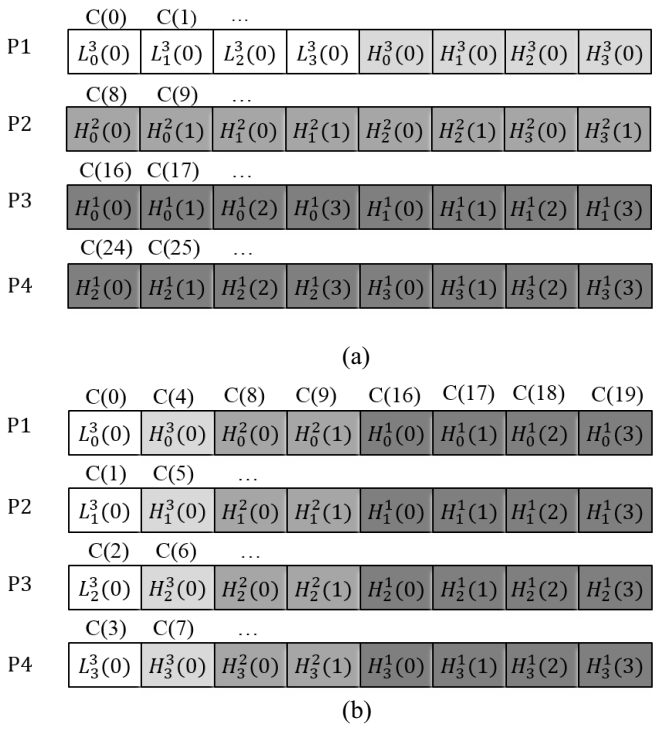

Fig. 6. Partitioning of a coding block into four sub-blocks. (a) Conventional partitioning [19], [20] and (b) proposed partitioning.

shared use of a single marker for every sub-block requires that all the coding information is reset after each sub-block coding is completed. This implies that each sub-block compression is performed independently. This independent sub-block compression requires the reorganization of the data structure. Fig. 6(a) shows the conventional linear data structure of the coefficients generated by DWT. This figure presents the example of 1×32 coefficients with a three-level filtering operation. The 1×32 coefficients are partitioned into four sub-blocks, each of size 1×8. The first coefficient $c(0)$ is $L_0^3(0)$ which is generated from the first eight input samples $\{x(0), x(1), \ldots, x(7)\}$ in addition to $x(8)$, which is one of the second eight samples. The subscript 0 indicates that a coefficient is generated for the first eight samples, and the superscript 3 represents the DWT decomposition level of three. The second coefficient $c(1)$ is $L_1^3(0)$, which is generated from the second eight input samples $\{x(8), x(9), \ldots, x(5)\}$ as well as $x(6)$ and $x(7)$ from the first eight samples and $x(16)$ from the third eight samples. If coefficients are indexed as in Table I, then $L_1^3(0)$ is $c(8)$. The third and fourth coefficients $c(2)$ and $c(3)$ are $L_2^3(0)$ and $L_3^3(0)$, respectively, which are generated mostly from the third and fourth sets of eight input samples, respectively. The fifth coefficient $c(4)$ is $H_0^3(0)$, from the first eight samples $\{x(0), x(1), \ldots, x(7)\}$. The six, seventh, and eighth coefficients $c(5)$, $c(6)$, and $c(7)$ are $H_1^3(0)$, $H_2^3(0)$, and $H_3^3(0)$, from the second, third, and fourth sets of eight input samples, respectively. As a result, the first sub-block corresponds to the DWT coefficients $\{L_0^3(0), L_1^3(0), L_2^3(0), L_3^3(0), H_0^3(0), H_1^3(0), H_2^3(0), H_3^3(0)\}$, and the second sub-block corresponds to the DWT coefficients $\{H_0^2(0), H_0^2(1), H_1^2(0), H_1^2(1), H_2^2(0), H_2^2(1), H_3^2(0), H_3^2(1)\}$. Note that the marker status of the second sub-block depends on the coding result of the first sub-block. For example, the first entry in the second sub-block $H_0^2(0)$ requires the information generated as a result of $H_0^3(0)$ in the first block. Therefore, the dependence between the first and second sub-blocks makes it impossible to reset the marker status between the coding operations of the first and second sub-blocks. Similarly, the dependence between the second and third sub-blocks prohibits the marker status from being reset.

The proposed sub-block structure can also reduce the buffer size, to store the results of DWT coefficients in the coefficient-to-bit-plane (C2B) module in addition to the buffer in the SPIHT module. This is because the DWT coefficients are generated in a periodical manner, as shown in Table I, which in fact follows the same order as the proposed sub-block structure. Therefore, it is not necessary to store all the DWT operations. Instead, only the first set of DWT coefficients is stored and forwarded to the SPIHT module, because this first set corresponds to the first sub-block for SPIHT operation. These DWT and SPIHT operations are processed in a pipelined manner, and the storage space in the C2B module can be significantly reduced. To this end, two buffers, each storing a sub-block, are required to store and forward the DWT results to SPIHT in a ping-pong manner. One buffer is necessary to store the DWT coefficients used as the input to SPIHT, while the other buffer is used to store the output of DWT operations for the next pipeline stage.

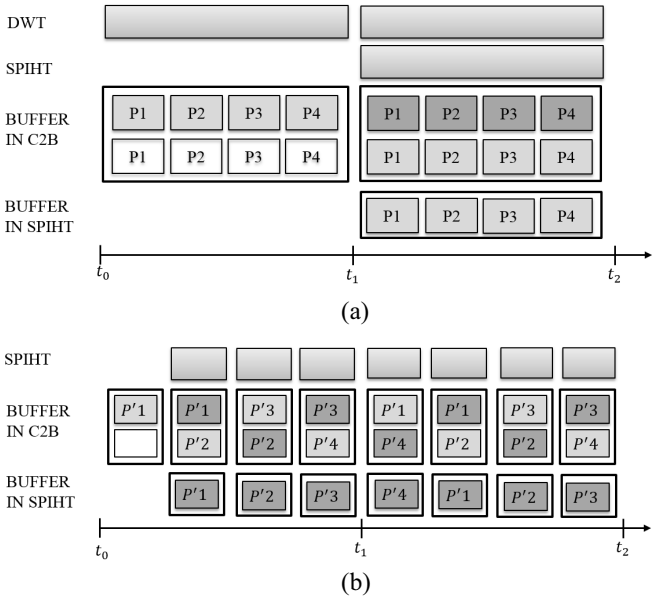

Fig. 7. Pipelined execution of the partitioned sub-blocks. (a) Conventional partitioning and (b) proposed partitioning.

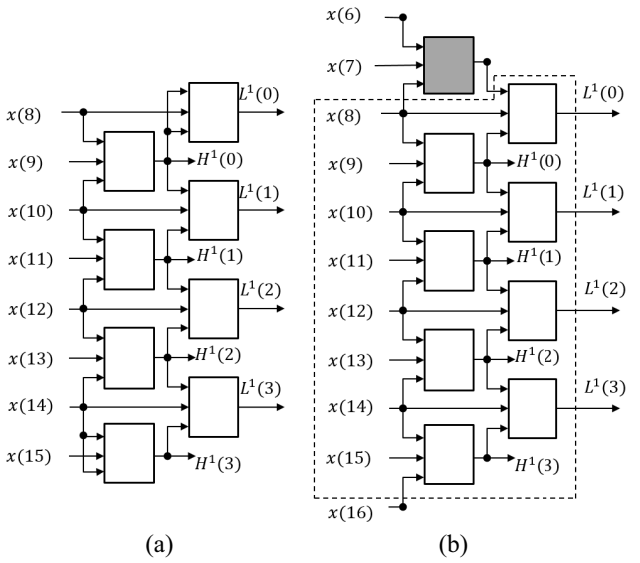

Fig. 8. 1-D DWT analysis of the second 1×8 block. (a) Conventional operation and (b) proposed operation with boundary extension.

Fig. 7 presents an example when the block size is 1×32, which is partitioned into four sub-blocks of size 1×8. Fig. 7(a) shows the conventional buffer, which stores the entire 1×32 blocks in the C2B module. For a pipelined execution in a ping-pong manner, two blocks need to be stored in the buffer with a size of 64 coefficients. The marker buffer also stores all the information for the entire 1×32 block. In Fig. 7(b), the space is reduced using the proposed scheme. When DWT transforms P1, eight output coefficients are stored in the first part in the memory block. Note that the coefficients in P1 are $L^3(0)$, $H^3(0)$, $H^2(0)$, $H^2(1)$, $H^1(0)$, $H^1(1)$, $H^1(2)$, and $H^1(3)$. When the second part P2 is transformed in DWT, the next eight output coefficients are stored in the second part of the memory in the C2B module. At the same time, C2B sends the first sub-block bit-plane by bit-plane to the SPIHT block. When the third part P3 is transformed in DWT, the first part P1 in the memory block is no longer used. The coefficients of P3 are saved in the first part of the memory. Concurrently, C2B transfers the bit-planes of the second memory to SPIHT. This process is repeated until DWT transforms the last sub-block. As a result, the memory size in the C2B module is significantly reduced to store only two sub-blocks, which each consist of 16 coefficients. Thus, the buffer size is reduced to only a quarter of the original size, achieving a 75% buffer reduction.

It should be noted that the DWT module in our design is similar to that in [19] and [20]. In particular, the block size for DWT is 1×64 instead of 1×16. Thanks to the DWT properties, the hardware cost of DWT module is relatively small as the DWT level are still three. In addition, the adoption efficiently handles the block artifact in the boundary pixels, thus, it significantly reduces the coding degradation due to the sub-block coding.

IV. Coding Efficiency Improvement for Individual Sub-Block Coding

A. Boundary Handling

The proposed sub-block partitioning method differs from the conventional DWT+SPIHT method for a small block size. For example, conventional DWT+SPIHT can be performed with a 1×8 block as the basic processing block. Recall that the proposed partitioned algorithm uses 1×8 as the sub-block size, while the basic processing block size is 1×32. There exists an important difference between the conventional algorithm for a 1×8 block and the proposed algorithm for a 1×8 sub-block partitioned from a 1×32 block. Fig. 8 illustrates this difference, where the DWT operation of the second 1×8 block $(x(8), \ldots, x(15))$ is presented. Fig. 8(a) shows the operations for the conventional approach, whereas the operation for the proposed algorithm is shown in Fig. 8(b). Comparing with Fig. 8(a), the output is the same, but the input is slightly different. In Fig. 8(b), $x(6)$, $x(7)$, and $x(16)$ are used for the operation, but they are not in Fig. 8(a). This indicates that the conventional approach for a 1×8 block does not make use of the image characteristics on the boundary pixels, which degrades the compression efficiency. The sub-block compression for P2 uses 11 input samples from $x(6)$ to $x(16)$ for the generation of DWT coefficients of the 1×8 sub-block, whereas the conventional 1×8 operations use only eight input samples from $x(8)$ to $x(15)$. Thus, this misses three out of 11 input samples, causing a substantial degradation of the compression efficiency (see the detailed numerical results in Section V-B).

B. Adjustment of the Target Bit Length for Individual Sub-Blocks

Assume that a block is partitioned into four sub-blocks P1, P2, P3, and P4. Fig. 9(a) shows an example in which the four sub-blocks, P1, P2, P3, and P4 may have different lengths for the generated bit-stream when they are encoded in

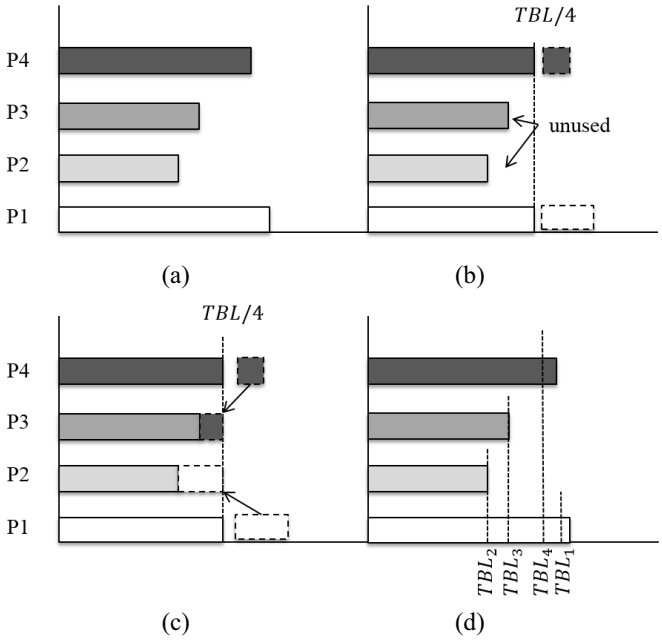

Fig. 9. Adjustment of the TBLs of sub-blocks. (a) Generated bit-stream length for lossless compression. (b) Truncated bit-stream to meet the identical TBL. (c) Reallocation of the truncated bit-stream to the other sub-block. (d) Adjusted TBL for reallocation.

a lossless manner. If the four sub-blocks are encoded independently, then a quarter of the TBL is assigned to each sub-block. Fig. 9(b) shows an example in which the sub-blocks P1 and P4 are truncated by the given TBL. Meanwhile, the sub-blocks P2 and P3 have wasted spaces. If these wasted spaces are used to store the truncated bit-streams of P1 and P4, then the data loss in P1 and P4 can be avoided [see Fig. 9(c)]. This reallocation of bit-stream storage is implemented if the TBL of each sub-block is adjusted such that a relatively large TBL is assigned to a complex sub-block (P1 or P4 in this example), whereas a small TBL is assigned to a simple sub-block (P2 or P3). Fig. 9(d) illustrates an adjustment of the sub-block TBL. Through this TBL adjustment, a wasted space is reduced, and consequently the coding efficiency can be improved. The challenge in the TBL adjustment lies in the fact that an appropriate TBL needs to be selected for each sub-block. Let TBL_1, TBL_2, TBL_3, and TBL_4 be the TBLs assigned to the sub-blocks, P1, P2, P3, and P4, respectively. The remainder of this section discusses how to select appropriate values for TBL_1, TBL_2, TBL_3, and TBL_4.

An SPIHT-based algorithm has an advantage in the estimation of the loss of image quality on the fly, while the algorithm is running. Let BL_1, BL_2, BL_3, and BL_4 be the bit-lengths given by the sub-blocks P1, P2, P3, and P4, respectively. Let Δ_i be the adjusted part for the sub-block Pi, with $i \in \{1, 2, 3, 4\}$. In this paper, Δ_1, Δ_2, Δ_3, and Δ_4 are simply defined as follows:

$$\Delta_1 = -\Delta_3 = \frac{BL_1 - BL_3}{2} \tag{4}$$

$$\Delta_2 = -\Delta_4 = \frac{BL_2 - BL_4}{2}. \tag{5}$$

Note that the adjusted length must be encoded as a header to be valid to the decoder. The length is represented by four bits consisting of one sign bit and three value bits. For example, (0)100 is translated to an adjusted length $\Delta = +16(=$ (0)10000).

V. Experimental Results

This section evaluates the compression efficiency of the proposed partitioned SPIHT algorithm and the cost of the hardware that implements the proposed algorithm.

A. VLSI Implementations

The proposed algorithm is implemented in hardware with Verilog HDL programming, which is synthesized with a 0.13-um library for ASIC fabrication. Table II shows the hardware cost for three implementations: the straight-forward design of NLS in Fig. 3, with the block size $L = 64$, and two proposed partitioned SPIHT methods with the sub-block size $L = 16$, with and without bit-allocation. The proposed method using boundary handling and bit-allocation schemes is marked with O, while the other (unused) is marked with $\times$. The third column shows the hardware costs for the NLS encoder and decoder, respectively, shown in Fig. 3. In this implementation, the block size is determined as 1×64, and all the hardware modules are designed to process a 1×64 block as the basic processing unit. Note that the input image is in the color YUV-422 format, so that each block has a total of 128 coefficients, with 64 luminance pixels for Y and 32 chrominance pixels each for U and V. In the proposed module with $L = 64$, the gate count of the DWT unit (6.1% for the encoder and 9.9% for the decoder) is much smaller than that of the SPIHT unit. Therefore, the block size reduction for the SPIHT unit is much more efficient. The upper parts of the fourth and fifth columns show the gate counts of the encoder without two additional schemes and those with two additional schemes, respectively. Compared with the hardware cost of the nonpartitioned implementation (i.e., $L = 64$) shown in the third column, the proposed partitioned SPIHT encoder without the boundary handling and the TBL adjustment achieves 62.73% of total hardware cost reduction. As explained, for the boundary handling and the TBL adjustment, extra hardware resources are necessary. As a result, the total hardware cost of the proposed partitioned SPIHT encoder with two additional schemes is 22.4K gates, which still achieves a large reduction of 58.67%. The lower parts of the fourth and fifth columns show the hardware costs of the proposed decoder designs without two additional schemes and those with two additional schemes, respectively. Compared with the hardware cost in the third column, the gate count of the proposed partitioned SPIHT decoder with two additional schemes is 25.9K gates, which achieves a large reduction of 52.48%.

Table III compares the hardware resources of the proposed implementation with those of previous works in [19] and [20]. Clearly, the proposed implementation consumes significantly less hardware costs and memory bits. The cost reduction can be explained as follows. As mentioned in the previous section, the gate count of the DWT unit is much smaller than that of the SPIHT unit and thus, this paper focuses on the block size

TABLE II
GATE COUNTS OF THE PROPOSED HARDWARE IMPLEMENTATIONS (ASIC)

	Module	L=64	L=16 (×)	L=16 (O)
Encoder	Total	**54.2K**	**20.2K**	**22.4K**
	Forward DWT Unit	3.3K	3.3K	3.3K
	Encoder Core Unit	15.4K	8.2K	8.2K
	C2B Unit	22.2K	3.2K	3.2K
	Packetizer Unit	13.3K	5.5K	7.7K
Decoder	Total	**54.5K**	**22.2K**	**25.9K**
	Inverse DWT Unit	5.4K	5.4K	5.4K
	Decoder Core Unit	18.0K	8.9K	9.1K
	B2C Unit	27.6K	7.0K	7.0K
	Parser Unit	3.5K	0.9K	4.4K

TABLE III
GATE COUNTS COMPARISON OF HARDWARE IMPLEMENTATIONS (ASIC)

Algorithm		Gate count (ASIC 0.13μm)	Memory (bits)
Hybrid SPIHT[19]	Encoder	42.8K	1980
	Decoder	57.7K	2249
1D SPIHT [20]	Encoder	50.0K	1984
	Decoder	68.8K	1280
The proposed work with L=16 (O)	Encoder	**22.4K**	496
	Decoder	**25.9K**	320

reduction for the SPIHT unit and proposes the generic partitioned SPIHT algorithm. It should be noted that the coding block unit for the proposed partitioned SPIHT is only 1×16; while that of [19] and [20] is 1×64. The proposed algorithm compresses all the bit-planes in a sub-block before it moves on to the next sub-block, so that it is not necessary to store all the intermediate results for one sub-block after the sub-block coding is completed. In this manner, the buffer space for the intermediate results is shared by multiple sub-blocks, and the total gate count and buffer size are significantly reduced. To this end, compared to [19] and [20], the proposed implementation achieves a memory bit reduction of 80.7% and 75%, respectively, and a total hardware cost reduction of 51.94% and 59.34%, respectively. These results indicate that the proposed design is very suitable for mobile devices.

B. Compression Efficiency

To evaluate the objective image quality, the peak signal-to-noise ratio (PSNR) is measured for the 24 test images used as test images [23]. The PSNR is calculated by

$$\mathrm{PSNR} = 10 \times \log_{10}\left(\frac{255^2}{(\mathrm{MSE}_R + \mathrm{MSE}_G + \mathrm{MSE}_B)/3}\right) \quad (6)$$

where MSE_R, MSE_G, and MSE_B represent the mean squared errors of the red, green, and blue components, respectively. Similar to [19] and [20], the images are transformed into Y, Cb, and Cr components, and then subsampled in the 4:2:2 format.

Table IV reports the average PSNR results of the proposed design, the previous 1-D DWT+SPIHT designs in [19] and [20]. Compared to the previous designs, the proposed design has achieved the significant hardware reduction by reducing the coding block unit of the proposed SPIHT from

TABLE IV
QUALITY COMPARISON OF THE IMAGES WITH SPIHT VARIATIONS

Algorithm	PSNR (dB)	ΔPSNR (dB)
1D SPIHT [20] w/o additional schemes	43.41	-
1D SPIHT [20] w/ additional schemes	44.37	+0.96
1D Hybrid SPIHT[19]	46.04	+2.63
NLS L=64	43.62	+0.21
Proposed NLS L=16	41.68	-1.73
+ Boundary Handling	42.49	-0.92
+ Bit allocation	42.79	-0.62

1×64 to 1×16, but it accompanies the PSNR degradation inevitably. The second row of Table IV shows the PSNR of the baseline high-throughput 1-D SPIHT [20], which achieves 43.41 dB with the same 24 test images [23]. As [20] includes additional schemes as dummy bit reuse, half-block processing, and relocation of sorting bits, [20] can achieve the PSNR of 44.37 dB on average as shown in third row of Table IV. In [19], the coding efficiency can be considerably improved as the DWT low coefficients are further encoded by a lossless compression algorithm. However, as described above, the PSNR enhancement in [19] and [20] suffers from very huge hardware costs and consequently, these are not suitable for mobile devices. In the sixth row of Table IV, the proposed NLS with $L = 16$ shows the PSNR degradation of 1.73 dB compared to the baseline high-throughput 1-D SPIHT [20]. The previous study in [24] shows that when the processing unit of the baseline high-throughput 1-D SPIHT [20] changes from 1×64 to 1×16, the average performance degrades by 3.15 dB because the processing units of DWT and SPIHT are all reduced to 1×16. However, the proposed design effectively reduces the hardware costs and memory bits with relatively small PSNR degradation of 1.73 dB by utilizing 1×64 DWT units. Furthermore, the proposed boundary handling increases the average PSNR from 41.68 dB to 42.49 dB (i.e., +0.81 dB); while the TBL adjustment further increases the PSNRs from 42.49 dB to 42.79 dB (i.e., +0.3 dB). As a result, thanks to the boundary handling and the TBL adjustment, the proposed design improves the coding efficiency by 1.11 dB, resulting in the PSNR degradation of 0.62 dB compared to the baseline high-throughput 1-D SPIHT [20]. Although the PSNR of the proposed design is still lower than that of the previous works, the proposed design alleviates the considerable performance degradation and significantly reduces the hardware resources.

VI. CONCLUSION

As various commercial video display systems, including high-resolution televisions and mobile devices, have become increasingly popular, frame memory compression is becoming more important for reducing memory storage and bandwidth effectively. There are two main contributions in this paper for supporting efficient compression with 1-D DWT and SPIHT schemes which are widely used for frame memory compression. First, the proposed partitioned algorithm is designed to handle various sub-blocks, depending on the decomposition level of DWT. If the decomposition level is n, then any

multiple of 1×2^n can be used as the sub-block size. The proposed algorithm compresses all the bit-planes in a sub-block before it moves on to the next sub-block, so that it is not necessary to store all the intermediate results for one sub-block after the sub-block coding is completed. In this manner, the buffer space for the intermediate results is shared by multiple sub-blocks, and the total buffer size is significantly reduced. The required gate count is 48.3K, achieving up to 59.34% hardware cost reduction compared with the previous 1-D DWT and SPIHT design. Second, to reduce the degradation of the compression efficiency by the partitioned coding scheme, a boundary handler of DWT coefficients is utilized, and the TBL of each sub-block is adjusted according to its predicted complexity. In summary, the proposed approach makes a good tradeoff between the cost and the coding efficiency, which makes it applicable for low-cost commercial devices such as smartphones.

REFERENCES

[1] H.-C. Kuo and Y.-L. Lin, "A hybrid algorithm for effective lossless compression of video display frames," *IEEE Trans. Multimedia*, vol. 14, no. 3, pp. 500–509, Jun. 2012.

[2] J. Someya, N. Okuda, and H. Sugiura, "The suppression of noise on a dithering image in LCD overdrive," *IEEE Trans. Consum. Electron.*, vol. 52, no. 4, pp. 1325–1332, Nov. 2006.

[3] T.-H. Tsai and Y.-H. Lee, "A 6.4 Gbit/s embedded compression codec for memory-efficient applications on advanced-HD specification," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 20, no. 10, pp. 1277–1291, Oct. 2010.

[4] Y. Lee, C.-E. Rhee, and H.-J. Lee, "A new frame recompression algorithm integrated with H.264 video compression," in *Proc. IEEE Int. Symp. Circuits Syst.*, New Orleans, LA, USA, May 2007, pp. 1621–1624.

[5] Y. Jin, Y. Lee, and H.-J. Lee, "A new frame memory compression algorithm with DPCM and VLC in a 4×4 block," *EURASIP J. Adv. Signal Process*., vol. 2009, no. 629285, p. 18, Dec. 2009.

[6] J.-W. Han, M.-C. Hwang, S.-G. Kim, T.-H. You, and S.-J. Ko, "Vector quantizer based block truncation coding for color image compression in LCD overdrive," *IEEE Trans. Consum. Electron.*, vol. 54, no. 4, pp. 1839–1845, Nov. 2008.

[7] D. Marpe, T. Wiegand, and G. J. Sullivan, "The H.264/MPEG4 advanced video coding standard and its applications," *IEEE Commun. Mag.*, vol. 44, no. 8, pp. 134–143, Aug. 2006.

[8] D. Woo, C.-E. Rhee, and H.-J. Lee, "A cache-aware motion estimation organization for a hardware-based H.264 encoder," *IEEE Trans. Consum. Electron.*, vol. 60, no. 1, pp. 83–91, Feb. 2014.

[9] K. Sayood, *Introduction to Data Compression*, 3rd ed. New York, NY, USA: Morgan Kaufmann, 2005.

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

[11] C.-C. Cheng, P.-C. Tseng, and L.-G. Chen, "Multimode embedded compression codec engine for power-aware video coding system," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 19, no. 2, pp. 141–150, Feb. 2009.

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

[13] P. Corsonello, S. Perri, G. Staino, M. Lanuzza, and G. Cocorullo, "Low bit rate image compression core for onboard space applications," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 16, no. 1, pp. 114–128, Jan. 2006.

[14] T. W. Fry and S. A. Hauck, "SPIHT image compression on FPGAs," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 15, no. 9, pp. 1138–1147, Sep. 2005.

[15] Y. Jin and H.-J. Lee, "A block-based pass-parallel SPIHT algorithm," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 7, pp. 1064–1075, Jul. 2012.

[16] Z. Lu, D. Y. Kim, and W. A. Pearlman, "Wavelet compression of ECG signals by the set partitioning in hierarchical trees algorithm," *IEEE Trans. Biomed. Eng.*, vol. 47, no. 7, pp. 849–856, Jul. 2000.

[17] S. Ktata, K. Ouni, and N. Ellouze, "A novel compression algorithm for electrocardiogram signals based on wavelet transform and SPIHT," *Int. J. Signal Process.*, vol. 5, no. 4, pp. 253–258, Sep. 2009.

[18] Z.-H. Zhang and J. Zhang, "Unsymmetrical SPIHT codec and 1D SPIHT codec," in *Proc. Int. Conf. Elect. Control Eng.*, Wuhan, China, Jun. 2010, pp. 2498–2501.

[19] S. Kim, D. Lee, H. Kim, N. X. Truong, and J. S. Kim, "An enhanced one-dimensional SPIHT algorithm and its implementation for TV systems," *Displays*, vol. 40, pp. 68–77, Dec. 2015.

[20] S. Kim, D. Lee, J.-S. Kim, and H.-J. Lee, "A high-throughput hardware design of a one-dimensional SPIHT algorithm," *IEEE Trans. Multimedia*, vol. 18, no. 3, pp. 392–404, Mar. 2016.

[21] D. Le Gall and A. Tabatabai, "Sub-band coding of digital images using symmetric short kernel filters and arithmetic coding techniques," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, New York, NY, USA, Apr. 1988, pp. 761–764.

[22] P.-Y. Chen, "VLSI implementation for one-dimensional multilevel lifting-based wavelet transform," *IEEE Trans. Comput.*, vol. 53, no. 4, pp. 386–398, Apr. 2004.

[23] *Kodak Lossless True Color Image Suite.* Accessed: Jul. 2, 2017. [Online]. Available: http://r0k.us/graphics/kodak/

[24] H. Kim and H.-J. Lee, "A low-power surveillance video coding system with early background subtraction and adaptive frame memory compression," *IEEE Trans. Consum. Electron.*, vol. 63, no. 4, pp. 359–367, Nov. 2017.

Xuan Truong Nguyen received the B.S. degree in electrical engineering from the Hanoi University of Science and Technology, Hanoi, Vietnam, in 2011 and the M.S. degree in electrical engineering and computer science from Seoul National University, Seoul, South Korea, in 2015, where he is currently pursuing the Ph.D. degree in electrical engineering and computer science.

His current research interests include algorithm and very large scale integration design for computer vision.

Hyuk-Jae Lee (M'03) received the B.S. and M.S. degrees in electronics engineering from Seoul National University, Seoul, South Korea, in 1987 and 1989, respectively, and the Ph.D. degree in electrical and computer engineering from Purdue University, West Lafayette, IN, USA, in 1996.

From 1998 to 2001, he was with the Server and Workstation Chipset Division, Intel Corporation, Hillsboro, OR, USA, as a Senior Component Design Engineer. From 1996 to 1998, he was a Faculty Member with the Department of Computer Science, Louisiana Tech University, Ruston, LA, USA. In 2001, he joined the School of Electrical Engineering and Computer Science, Seoul National University, where he is currently a Professor. He is the Founder of Mamurian Design, Inc., Seoul, a fabless SoC design house for multimedia applications. His current research interests include computer architecture and SoC design for multimedia applications.

Dr. Lee currently serves as an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS for Video Technology.

Hyun Kim (M'16) received the B.S., M.S., and Ph.D. degrees in electrical engineering and computer science from Seoul National University, Seoul, South Korea, in 2009, 2011 and 2015, respectively.

Since 2015, he has been with the BK21 Creative Research Engineer Development for IT, Seoul National University, where he is currently an Assistant Professor. His current research interests include low-power algorithm and SoC design for video coding such as HEVC and H.264/AVC and configurable video coding for real time systems.