IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 30, 2019, accepted September 18, 2019, date of publication September 23, 2019, date of current version October 4, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2943194

An Effective Algorithm and Architecture for the High-Throughput Lossless Compression of High-Resolution Images

JAESHIN LEE[1], JUWON YUN[1], JINYOUNG LEE[1], IMJAE HWANG[1], DUKKI HONG[1], YOUNGSIK KIM[2], CHEONG GHIL KIM[3], (Member, IEEE), AND WOO-CHAN PARK[1], (Member, IEEE)

[1]Department of Computer Engineering, Sejong University, Seoul 05006, South Korea
[2]Department of Game and Multimedia Engineering, Korea Polytechnic University, Siheung 15073, South Korea
[3]Department of Computer Science, Namseoul University, Cheonan 31020, South Korea

Corresponding author: Woo-Chan Park (pwchan@sejong.ac.kr)

This work was supported in part by the Institute for Information and Communications Technology Promotion through the Korean Government (MSIP) under Grant 2016-0-00204, in part by the Development of Mobile GPU Hardware for Photo-Realistic Realtime Virtual Reality, in part by the National Research Foundation of Korea (NRF) Grant Funded by the Korean Government (MSIP) under Grant 2019R1A2C1005163, and in part by the Computer Aided Design (CAD) tools funded by the IC Design Education Center (IDEC) of South Korea.

ABSTRACT This paper proposes a high-throughput lossless image-compression algorithm based on Golomb–Rice coding and its hardware architecture. The proposed solution increases compression ratios (CRs) while preserving the throughput by taking advantage of a novel parallel variable-length sign coding (PVSC) algorithm that reduces the sign bits to achieve a higher CR. In addition, the proposed solution adopts and modifies the two existing compression algorithms to improve the overall compression performance. The experimental results show that the proposed solution yields an average CR of 3.12, which is higher than those achieved with the previous algorithms. The hardware implementation of the proposed solution for an 8×8 block unit achieves a throughput of 18 GBps and 24 GBps when encoding and decoding, respectively. This hardware performance is enough to handle 7680×4320@240-Hz image processing.

INDEX TERMS Variable length coding, lossless image compression, DDPCM, Golomb-Rice coding, UHD.

I. INTRODUCTION

In recent years, high-definition (HD) images, such as full HD (1920×1080), quad HD (QHD, 2560×1440), and ultra HD (UHD, 3840×2160 or 7680×4320) have been used in mobile devices, PCs, and TVs. To handle 4:2:0 YUV images at 30 Hz, full HD requires a processing capability of 93 MBps, QHD requires 166 MBps, and UHD requires 373 MBps or 1.5 GBps. These processing speeds increase to 3 GBps or 12GBps if the UHD scanning frequency is 240 Hz. With the rapid improvement in image resolution in the latest video systems, the bus bandwidth needed to refer to the images stored in the frame buffer has increased dramatically. In addition, the memory bandwidth requirement has become one of the most concerning issues in binocular video applications such as virtual reality systems, as they require twice the throughput. Therefore, there have been many studies on image-compression techniques to alleviate this problem.

Image-compression techniques are classified into two categories: lossy and lossless. The quantization in lossy methods increases compression ratios (CRs) but data loss can occur. Lossless compression methods have lower CRs than lossy methods, but they allow the original data to be perfectly reconstructed from the compressed data. As a result, this lossless compression could be very suitable as a frame buffer recompression algorithm that is applicable to liquid crystal display (LCD) overdrive [40]. Here, data redundancy is generally eliminated in the prediction stage, and the outcome is compressed via entropy coding. The available prediction methods are either spatial-based (e.g., CALIC [2], LOCO [3], DPCM [4], etc.) or transform-based (e.g., wavelet analysis [5]) in nature. The coding strategies commonly used for entropy coding include Golomb–Rice coding and Huffman coding [39].

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

To address the memory bandwidth problem in high-resolution images without quality degeneration, a number of lossless embedded compression (LEC) techniques have been proposed [7], [8], [18], [23]–[38], [42]. However, the studies in [8], [18] point out that the previous LEC schemes in [23], [27]–[31], [34]–[36], [33]–[35] are not sufficient to handle high-performance applications such as HD video sequences in real time due to heavy data dependency, high hardware complexity, and low throughput. The works in [7], [32] achieve higher throughput by employing line-based algorithms. In this approach, the images are displayed line by line and non-power-of-two 3D textures are supported. According to the findings in [37], block-based algorithms generally yield better compression performance than line-based algorithms.

The studies in [8], [18], [36]–[38], [42], [43] increase CRs by proposing block-based prediction algorithms or by enhancing entropy coding algorithms. These studies also implement hardware for high-resolution image processing. In [36], [38], the implemented hardware can process 4-K images with a high CR, but it is unable to process more than 3 pixels per cycle due to data-processing dependency. The methods in [18], [37], [43] achieve high CRs while processing 5.1 pixels/cycle, 10.7 pixels/cycle, and 10.67 pixels/cycle when encoding and 14.2 pixels/cycle, 21.3 pixels/cycle and 10.67 pixels/cycle when decoding. In other words, they still encounter the problem of data dependencies. During compression or decompression, they take an $N \times N$ block shape in a frame as a basic unit that cannot that cannot be applied to non-power-of-two 3D textures.

In [8], our previous work and the base model to be improved in this work, the differential–differential pulse-coded modulation (DDPCM) prediction is performed on various $M \times N$ blocks of the original image frame, and prediction errors are encoded using Golomb–Rice coding. The hardware architecture in [8] can perform massively parallel processing in the variable-length coding stage and in the prediction stage. It achieves lossless pixel throughput by compressing and decompressing blocks during every cycle with 6-12 times the performance improvement compared to the comparative models [13], [26]–[29]. However, from the point of CR, the model [8] still have the room for improvement.

In this paper, we propose a lossless compression solution (algorithms and architecture) that increases CRs while offering the massively parallel pixel-processing architecture suggested in [8]. For this purpose, following three techniques are utilized:

- sign-bit field compression
- efficient use of spatial locality in image data
- flexibility in the use of k parameter

The first is to develop a new parallel variable-length sign coding (PVSC) algorithm. The second and third are to adopt and modify the previous compression solution, which shall replace DDPCM with DPCM and adopt adaptive-k instead of fixed-k, respectively. Kim *et al.* [8] scanned the first left column pixels by vertical prediction method and the other pixels using horizontal prediction method. We follow this sequence in the same way.

The hardware architecture of the proposed algorithm enables each sign bit to be coded in parallel, called PVSC, thus allowing massively parallel processing. In addition, the proposed architecture eliminates the pipeline latency that can occur in variable-length sign decoding. In the experiments with six full-HD benchmarks, the proposed solution yields an average data-reduction ratio of 70%. The four-stage pipeline encoder and decoder implemented in the 55-nm fabrication process have a maximum clock frequency of 370 MHz and 286 MHz and a gate count of 83 K and 121 K, respectively. Due to the pipeline depth adjustment, logic optimization, and the high-end manufacturing, this implementation achieves a throughput of 24 GBps, which is higher than the hardware performance (13 GBps) reported in [8].

The proposed architecture has the following three characteristics. First, it exceeds the throughput requirement (about 12 GBps) necessary to provide a high-end screen refresh rate for 8-K images (e.g., 240 Hz). Second, its throughput in relation to the hardware size is higher than that in [37] by more than 1.8 times. Third, there is no tradeoff between hardware performance and compression efficiency. Despite improved hardware performance, the CR of the proposed solution is as good as those reported in the latest literature on lossless compression technologies.

The remainder of this paper is organized as follows. Section II reviews the previous works related to the topic. Section III introduces the proposed architecture for our parallel algorithm and the packing and unpacking parallel scheme of two variable-length coded data (unary and PVSC) without individual length information. Section IV presents the experimental results of algorithm and hardware performance. Section V consists of the conclusion.

II. RELATED WORK

In many LEC methods, Golomb algorithms or Golomb–Rice algorithms are used for entropy coding [8], [11], [12], [13], [22], [32]. Golomb–Rice coding divides a positive integer (an input value) into two parts: quotient q and remainder r. The quotient is sent in unary coding. Unary coding represents a natural number n, with n ones followed by a zero (a unique terminating symbol). The remainder r is redefined in truncated binary encoding as $2k$. In [8], [13], [32], a fixed-k value is used in Golomb–Rice coding.

As far as the authors are aware, the massive parallel pixel-processing architecture for Golomb–Rice coding was first proposed in [8]. This architecture consists of DDPCM and Golomb–Rice encoding with a fixed-k ($k = 2$) value. The original image frames are organized as $M \times N$ sub-window arrays, to which DDPCM is applied, thereby producing one seed and $M \times N - 1$ pieces of differential data. The Golomb–Rice algorithm then encodes the differential data into a variable-length codeword. The study in [8] noted that the position of a unique symbol in a variable-length codeword gave an indication of the original data; based on this,

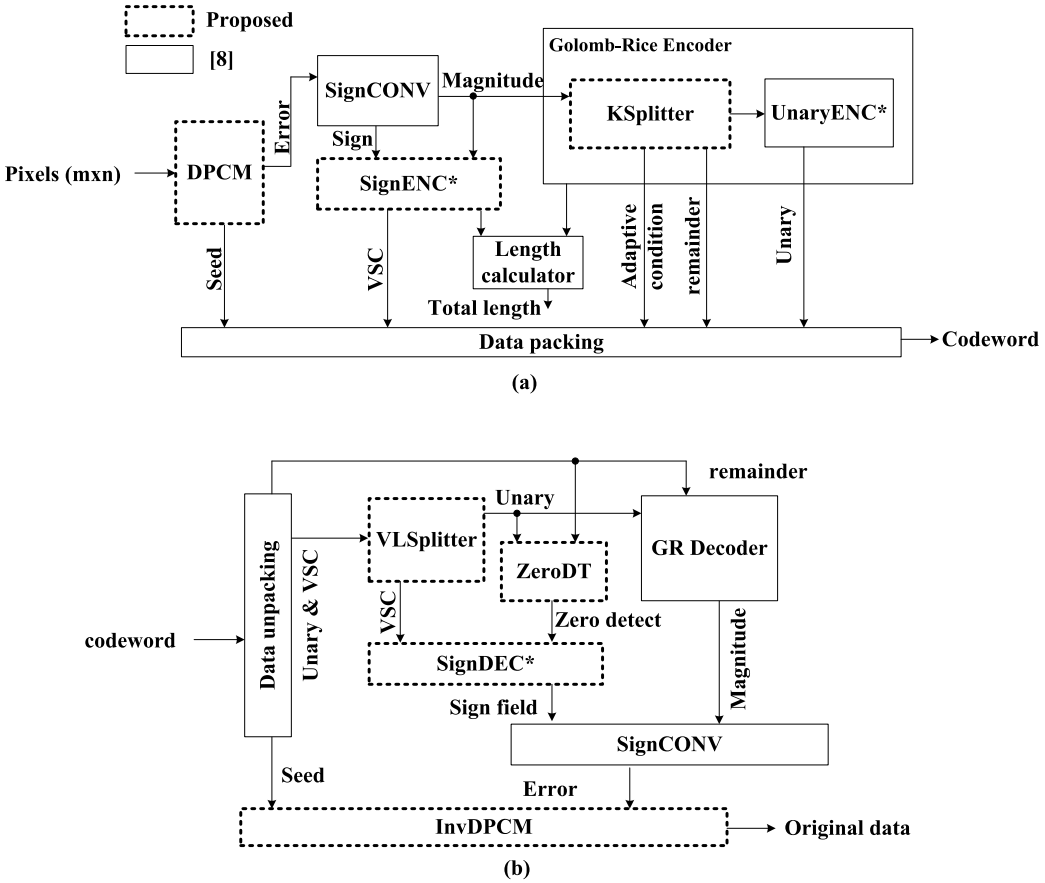

FIGURE 1. Proposed hardware architecture: (a) compression and (b) decompression. The dotted-line boxes denote newly proposed components, solid-line boxes denote legacy components, and ∗ indicates a parallel-processing component.

hardware architecture for parallel encoding and decoding was proposed. The experiments performed with 8 × 8 blocks show that this architecture achieves fully parallel processing of 64 pixels/cycle, but its CR is 1.52, which is rather low.

According to [37], a high-throughput hardware implementation for a variable-length coding algorithm is difficult to apply; therefore, its reference frame recompression scheme uses semi-fixed length (SFL) or significant bit truncation (SBT) algorithms that encode prediction errors in a group with the same number of bits and store the fixed-length data of each group. In [18], the hierarchical average and copy prediction (HACP) algorithm that processes $N \times N$ blocks at L-levels is used to create prediction residuals, and the prediction errors are entropy coded using SBT. Reference [37] suggests that the multi-mode DPCM and averaging prediction (MDA) algorithm for $N \times N$ blocks should combine the advantages of DPCM scanning and averaging and uses SFL for entropy coding. The compression algorithm in [18] has parallelism of 5.1 pixels/cycle in compression and 14.2 pixels/cycle in decompression for the 16 × 8 block unit. The algorithm in [37] yields parallelism of 10.7 pixels/cycle in compression and 21.3 pixels/cycle in decompression for the 8 × 8 block unit. Although the algorithms in [18], [37] are not fully parallel, they achieve a high CR of 2.2 and 2.49, respectively.

III. PROPOSED ARCHITECTURE

This section describes the proposed algorithm and architecture for the high-throughput lossless compression converting the overall architecture, pipeline stages, PVSC, adaptive-k parallel scheme of Golomb–Rice, and packing/unpacking codeword scheme including two variable-length data.

A. ARCHITECTURE OVERVIEW

Figure 1 shows the proposed architecture. The solid-line components are identical to those proposed in [8] and the dotted-line components are the newly proposed parts presented in this paper. The seven dotted-line boxes in the compressor and decompressor are classified into five groups: 1) DPCM/InvDPCM, 2) KSplitter, 3) SignENC/SignDEC, 4) ZeroDT, and 5) VLSplitter.

The DPCM/InvDPCM component, a replacement for the previous DDPCM/InvDDPCM prediction algorithms in [8], enables a slight increase in the CR. The KSplitter component replaces the previous fixed-k ($k = 2$) algorithm with a block-based adaptive-k algorithm in Golomb–Rice

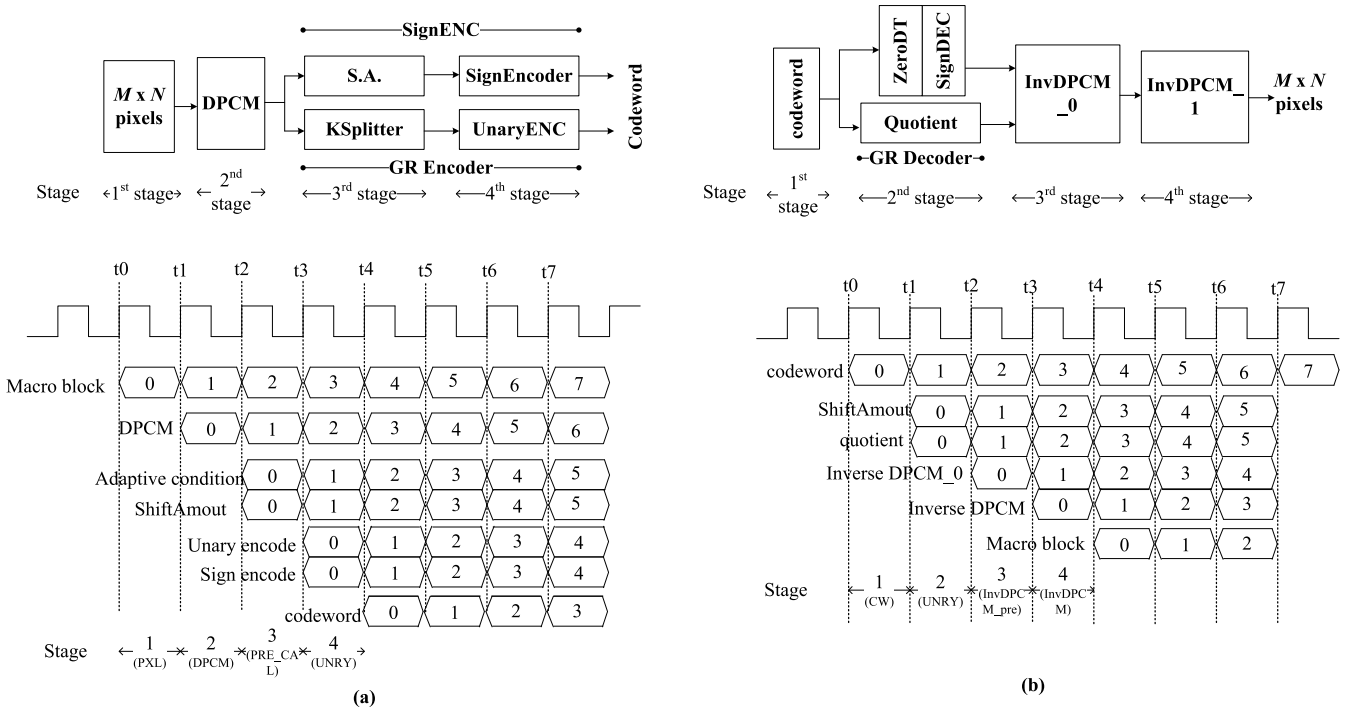

FIGURE 2. Pipeline stage: (a) encoder (b) decoder.

coding, compensating for the degradation in the CR related to the fixed-k algorithm. The SignENC/SignDEC component enables parallel processing by resolving the data-dependency problem in the proposed PVSC algorithm which is a variable-length algorithm. The ZeroDT component eliminates latency that occurs in PVSC decoding. It accelerates a restoration of the sign bits that are deleted during encoding. Finally, the VLSplitter splits the unary code and PVSC code from the packed data without requiring knowledge of their individual lengths.

The pipeline architecture consists of four stages, as shown in Figure 2. One stage is added to the compressor in [8] to perform the proposed algorithm. The adaptive condition of k and the shift amount (SA) are calculated in the added stage. If a macro block is given to the compressor at time *t0*, DPCM, adaptive condition/SA, and unary/sign encoding are performed from time *t1* to *t3*, respectively, and the codeword is completed at time *t4*. One stage is also added to the decompressor but alleviates the critical time path, unlike the compressor. If the codeword is given to the decompressor at time *t0*, the SA/quotient and two steps of inverse DPCM are performed from time *t1* to *t3*, respectively, and a macro block is reconstructed at time *t4*.

The algorithm proposed for the decompressor is performed in parallel at time *t1* when the quotient is reconstructed (see Figure 2). Inverse DPCM is separated into two stages. We compute and store half of the mathematical operations in the first stage, and then calculate other half in the half-result stored in the next step. This technique can be applied to mathematical operations which don't have feedback or branch path such as unary or DPCM in our algorithm. We consider the tradeoff between area and performance and decide to apply it to the inverse DPCM.

Our compressor and decompressor performed four cycles because of the four-stage pipeline. There is no throughput drop for [8] because the macroblock is compressed or decompressed every cycle after the first pipeline latency.

The data-compression flow of the proposed architecture is as follows. The DPCM component takes the pixel image data of an $M \times N$ block to be compressed and produces residual data for the $M \times N$ block by eliminating data redundancy. The residual data are split into the first element (seed) that is exempted from compression and the prediction error field (consisting of $(M \times N) - 1$ prediction errors) that is to be compressed. The SignCONV component takes the prediction error field and splits it into the sign field consisting of $(M \times N) - 1$ sign bits and the magnitude field consisting of $(M \times N) - 1$ magnitudes. The SignENC component takes the sign field and produces variable-length sign data by referring to the magnitude data (see Section 3.1). The SignENC component performs massively parallel bit processing to achieve a throughput that is as high as that achieved using the previous algorithms (see Section 3.2). The Golomb–Rice encoder takes the magnitude field and produces a variable-length codeword.

During Golomb–Rice encoding, the KSplitter finds the k that is optimized for the code length (see Section 3.4) and produces the remainder of $(M \times N - 1) \times k$ bits. The UnaryENC component performs the massively parallel processing proposed in [8] for the magnitude field that is split by k and produces a variable-length unary code. The produced variable-length unary code, seed, variable-length sign

data, and $(M \times N - 1) \times k$ remainder bits are packed, creating a final codeword (see Section 3.5).

The decoding flow of the proposed architecture, which is the opposite of the encoding flow, is as follows. The compressed variable-length codeword is unpacked and split into the seed, remainder, and variable-length data. The VLSplitter takes the variable-length data and divides it into the variable-length sign code and the unary code. In the SignDEC component, the variable-length sign code along with the sign bit data that is partially restored in the ZeroDT using the unary code (see Section 3.3) is restored to the sign field. At the same time, the Golomb–Rice decoder restores the magnitude field using the inputted unary code and the remainder. The restored sign field and magnitude field are reconstructed into the $(M \times N) - 1$ signed residuals. Finally, the InvDPCM component generates the original data of an $M \times N$ block with the signed residual data and the seed.

B. PROPOSED PVSC ALGORITHM

The Golomb–Rice coding widely used in existing LEC algorithms is a positive integer-based compression technique. To use Golomb–Rice coding, prediction errors are generally changed to positive numbers in the prediction stage via mapping, as in JPEG-LS [1], [10], [12], [28], or via preprocessing, as in FELICS [13], [32]. In [8], the sign field after the prediction stage is stored without being compressed. In this paper, the PVSC algorithm that compresses the sign field is proposed.

The proposed PVSC algorithm is based on the fact that the sign bits of $+0$ and -0 are redundant in signed magnitude number representations. The proposed PVSC algorithm concatenates the non-zero magnitude sign bits when encoding. After passing through the PVSC encoding stage, the zero-magnitude sign bits are removed, which contributes to increasing the CRs. During decoding, the restored zero-magnitude sign bits are automatically restored to zero, and the non-zero magnitude sign bits are restored from the inputted PVSC code.

Figure 3 shows an example of PVSC coding that exhibits five prediction errors when $k = 1$. Figure 3 (a) illustrates PVSC encoding. Among the prediction errors a0–a4, those with both a zero quotient and a zero remainder are found. This indicates that the prediction error is 0 and its sign bit is unnecessary (removable). In Figure 3 (a), a1 and a3 have both a zero quotient and a zero remainder, so their sign bits can be removed. Therefore, the inputted sign bits "1, 0, 0, 0, 1" become "1, 0, 1" after passing through PVSC encoding.

Figure 3 (b) presents the decoding process. During PVSC decoding, the prediction errors with both a zero quotient and a zero remainder are found, and their sign bits are reconstructed to 0. The sign bits of the other data are recovered from the PVSC code. In Figure 3 (b), a1 and a3 satisfy zero detection (i.e., the output of the zero-detection stage is "true"), so their sign bits are restored to 0. The sign bits of a0, a2, and a4 are restored from the PVSC code "1,0,1" in a sequential manner. Finally, the restored sign field is "1,0,0,0,1."

	a_0	a_1	a_2	a_3	a_4
sign (1-bit)	1	0	0	0	1
quotient (7-bit)	1	0	3	0	1
remainder (1-bit)	0	0	1	0	1
zero detection	F	T	F	T	F
Sign bit encoding (VSC)	1	0	1		

(a)

	a_0	a_1	a_2	a_3	a_4
VSC	1	0	1		
quotient (7-bit)	1	0	3	0	1
remainder (1-bit)	0	0	1	0	1
zero detection	F	T	F	T	F
Reconstructed sign		0		0	
VSC re-ordering	1		0		1
Sign bit decoding (1-bit)	1	0	0	0	1

(b)

FIGURE 3. Example of sign bit coding: (a) encoding and (b) decoding.

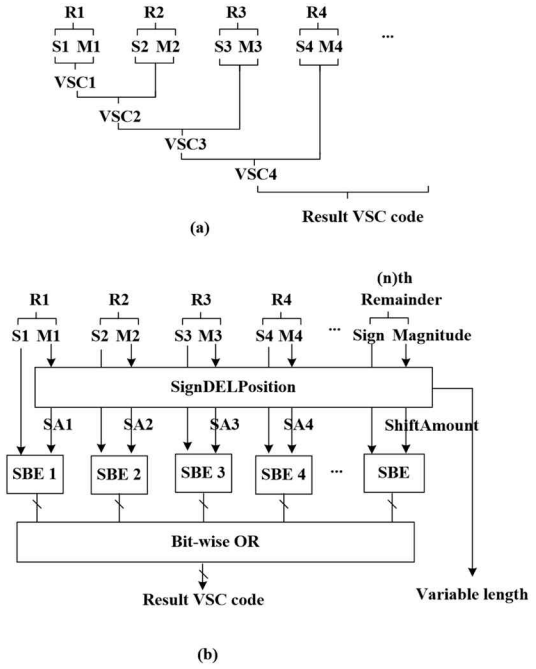

FIGURE 4. PVSC processing flow: (a) conventional sequential processing and (b) proposed parallel processing. (R#: Residual data, S#: Sign data, M#: Magnitude data).

C. SignENC/SignDEC: PARALLEL ARCHITECTURE FOR PVSC

The PVSC algorithm described in subsection 3 B either deletes the unnecessary sign bits or reorders the remaining sign bits using shifts during compression. This can be done in a sequential manner, as shown in Figure 4 (a). During sequential processing, concatenating the sign bit of each residual with the previously encoded variable-length sign code is repeated sequentially, which gives rise to long latency.

As represented in Figure 4 (b), this paper proposes an architecture that enables the parallel processing of the

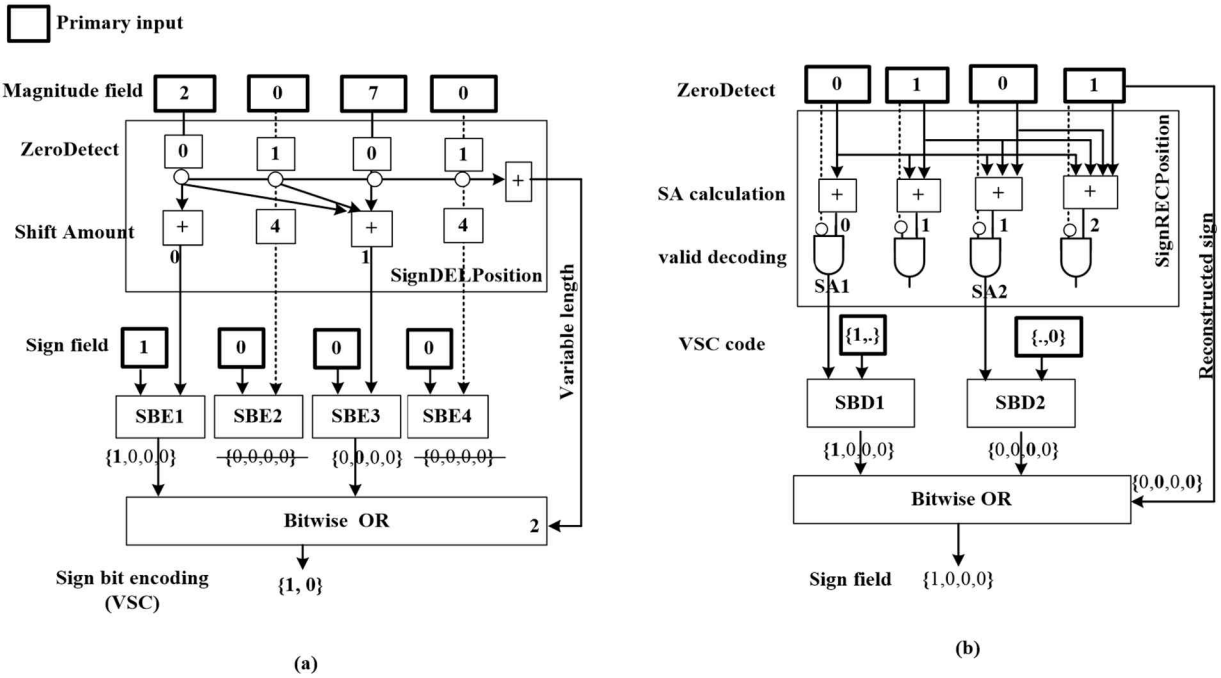

FIGURE 5. Example of PVSC parallel processing: (a) SignENC and (b) SignDEC (o: inverter, +: adder, SBE: sign bit encoder, PVSC: parallel variable-length sign coding, and SA: shift amount).

proposed PVSC algorithm. To provide parallelism during compression, two different types of components are introduced. A single SignDELPosition component produces bit position information simultaneously, and multiple SignBitEncoder (SBE) components perform the bit encoding.

The SignDELPosition checks the magnitude field to determine whether each sign bit should be deleted or reordered and calculates the corresponding SA and total sign bit length. If a sign bit needs to be deleted, its SA becomes a fixed value that is the field length $(M \times N) - 1$. If the sign bit needs to be reordered, the SA is the sum of sign bits that have been deleted up to the current bit position. Each SBE encodes the 1-bit sign data by shifting it as much as its SA and generates $(M \times N) - 1$ length codes. If a SA is $(M \times N) - 1$, its sign bit is shifted out of range and is eventually deleted. Finally, a bitwise OR operation is performed on all the encoded sign data of $(M \times N) - 1$ length, thus producing PVSC code with variable length that is calculated by SignDELPosition.

$$\mathrm{SA} = \begin{cases} \sum_{i=1}^{n} \overline{M}_i - 1; & if\ M_i = 0 \\ n; & otherwise \quad (i = 1, 2, 3, \ldots, n) \end{cases} \tag{1}$$

$$\text{PVSC length} = \sum_{i=1}^{n} \overline{M}_i \tag{2}$$

Figure 5 depicts the encoding and decoding of four sign data items in the proposed parallel-processing architecture. The SignENC represented in Figure 5 (a) encodes a 4-bit sign field into a variable-length sign code. The SignDELPosition takes an input from the magnitude field, determines whether to delete or shift the sign bit of each data item, and calculates the SA for each data item using the "(1)" and total variable length using the "(2)". (1) is a SA calculation formula. Here, M is a message, and the - symbol means inversion. That is, the i-th SA is the total number of magnitude (M) of 0 values from 0th to the i-th when the i-th M is 0. The i-th SA is n when the i-th M is 1. (2) is the formula for the PVSC length, which is the total number of 0 Ms for the magnitude field. If the magnitude field {2,0,7,0} is inputted, the SAs of the second and fourth data items with a zero magnitude are the field length n (i.e., n = 4). The SAs of the first and third data items with non-zero magnitude are the accumulated number of removed sign bits, that is, 0 and 1, respectively. As a result, the SAs are {0, 4, 1, 4}.

Each SBE encodes an individual sign bit of the inputted sign field. When a given sign field is {1, 0, 0, 0}, each bit of the sign field is sent to each SBE (SBE1, SBE2, SBE3, SBE4) in order. In SBE2 and SBE4 where the SA is 4, the sign bits are shifted out of range and thus deleted. The SAs of the first and third SBEs are 0 and 1, so they produce {1, 0, 0, 0} and {0, 0, 0, 0}, respectively. A bitwise OR operation is performed on all SBE outputs, creating {1, 0, 0, 0}. Finally, a PVSC code becomes {1, 0} with a 2-bit length that comes from SignDELPosition.

Figure 5 (b) illustrates how the SignDEC component restores the variable-length sign code {1,0} into the 4-bit sign field {1, 0, 0, 0}. To restore the sign field, the zero-detection result of the magnitude field, {0, 1, 0, 1}, is used. The magnitudes of the second and fourth data items are 0, so their sign bits are reconstructed to 0. To restore the sign bits of the first and third data items, the PVSC code is decoded.

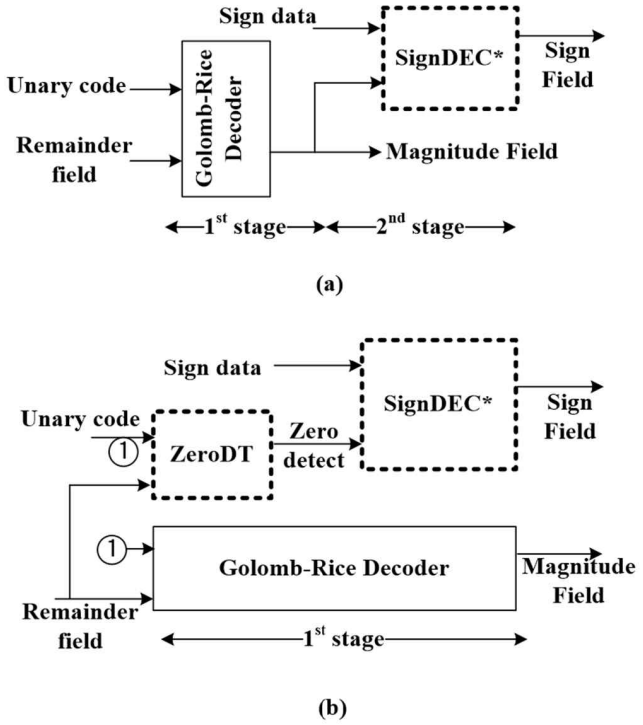

FIGURE 6. ZeroDT component: (a) conventional PVSC decoding flow and (b) the proposed architecture with an additional component for latency reduction.

The SignRECPosition component accumulates the number of the zero sign data, which makes the SA value of the first and third data items become 0 and 1, respectively. The SignBitDecoder1 (SBD1) creates {1, 0, 0, 0} by shifting the PVSC code bit "1" zero times. SBD2 produces {0, 0, 0, 0} by shifting the PVSC code bit "0" one time. A bitwise OR operation is then performed on all the created sign fields and on the reconstructed zero-magnitude sign field, restoring the final 4-bit sign field {1, 0, 0, 0}.

D. ZeroDT: AN ADDITIONAL COMPONENT FOR LATENCY REDUCTION

In the proposed PVSC algorithm, decoding is performed in two stages. As shown in Figure 6 (a), the Golomb–Rice decoder restores the magnitude field in the first stage. In the second stage, the sign field is restored using the zero-detection output of the magnitude field and the PVSC code. That is, 2-stage processing is needed to restore the original data.

As shown in Figure 6 (b), the proposed architecture introduces an additional component called ZeroDT that avoids dependencies between the first and second decoding stages. The basic idea is that zero detection is possible by identifying a code segment consisting only of symbols (i.e., terminating zero symbols) in the unary code. For example, a 16-bit unary code 1011101111100110 is decoded into {1, 3, 5, 0, 2}. The fourth code bit that involves successive symbols ("00") can be pinpointed during the zero detection. With ZeroDT, sign-field restoration and magnitude-field restoration can be performed independently and simultaneously at a single pipeline.

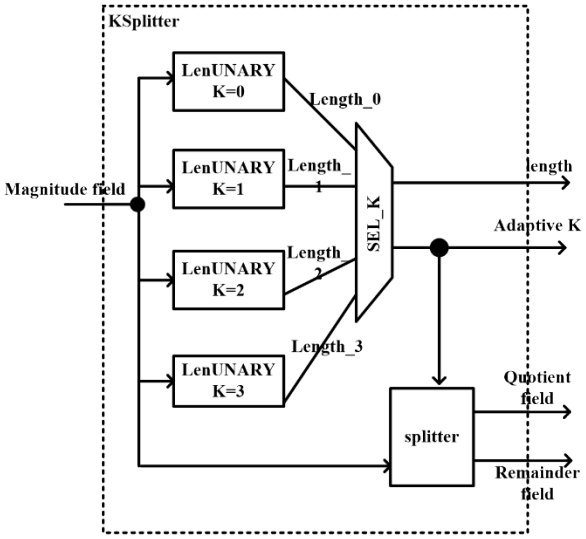

FIGURE 7. KSplitter hardware architecture.

E. KSplitter: AN ADAPTIVE-k BIT SPLITTER

According to [13], the FELICS algorithm uses a simple and efficient method for k parameter selection in the Golomb–Rice code (GR), but it also gives rise to heavy data dependencies that limit parallelism during compression. There is another reference [41] improves the compression efficiency by using the adaptive divisor k. It determines the k of the current block by using the previous k. So that it has dependency between blocks when calculating the current k. Due to this block dependency issue, only sequential processing is possible, and random access cannot be implemented. In [8], [13], the implemented hardware uses a fixed-k value ($k = 2$) for Golomb–Rice coding. The KSplitter of the proposed architecture replaces the fixed-k ($k = 2$) algorithm for Golomb–Rice coding in [8] with a block-based adaptive-k algorithm. This allows for compensating for a loss of compression efficiency (CRs) related to the fixed-k algorithm. Note that k is still fixed within a block to avoid data-dependency issues.

Figure 7 presents the hardware architecture of the KSplitter that finds an adaptive-k in block-based Golomb–Rice encoding. In the KSplitter component, k values are in the range of 0–3. This is because experimental results show that CRs are not significantly affected when the parameter k is greater than 4. Each LenUNARY computes the length of the Golomb–Rice code that is created when a given k is applied to the inputted magnitude field. The Golomb–Rice code length is the sum of the unary code length and the remainder length. The unary code length is proportional to the sum of quotients, and the remainder length is a fixed length determined according to k. The splitter separates the quotient field from the magnitude field using the k value

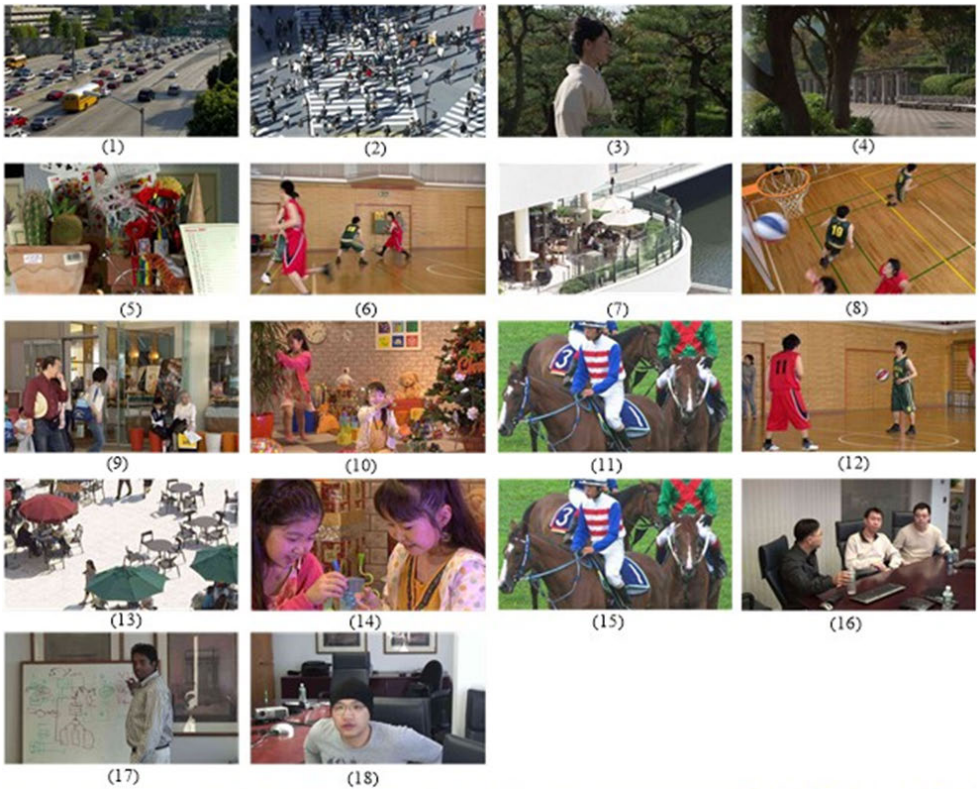

FIGURE 8. Benchmark sequences: (1) HEVC, CLASS A TRAFFIC, (2) HEVC, Class A PeopleOnStree, (3) HEVC, Class B Kimono, (4) HEVC, Class B ParkScene, (5) HEVC, Class B Cactus, (6) HEVC, Class B BasketballDrive, (7) HEVC, Class B BQTerrace, (8) HEVC, Class C BasketballDrill, (9) HEVC, Class C BQMall, (10) HEVC, Class C PartyScene, (11) HEVC, Class C RaceHorses, (12) HEVC, Class D BasketballPass, (13) HEVC, Class D BQSquare, (14) HEVC, Class D BlowingBubble, (15) HEVC, Class D RaceHorses, (16) HEVC, Class E Vidyo1, (17) HEVC, Class E Vydyo3, (18) HEVC, Class E Vidyo4.

that is determined in the SEL_K component and sends the quotient field to the UnaryENC component.

We can determine k for each block just taking one cycle without increasing latency, unlike the sequential processing case, as the proposed technique simultaneously obtains all unary lengths for the given k. In addition, k is determined before unary encoding because the proposed architecture consists of the KSplitter component followed by the UnaryENC component.

F. PACKING OF VARIABLE-LENGTH DATA

Figure 8 shows two types of data-pack formats. Figure 8 (a) presents the format used in the previous algorithm. It contains one variable-length field, Unary data, and stores the length of the entire data in the Length field. Figure 8 (b) shows the data-pack format of the proposed solution. There are two variable-length data fields, Unary data and Sign data. There is also an additional fixed-length field, adaptive-k (AK). To prevent a decrease in the CR, the individual lengths of the two variable-length codes are not stored. Only the total length of the data is stored in the Length field. If the value of the length is greater or equal to the original data length, the original data is stored. In this case, since a separate indicator for recording whether the data is compressed is not necessary, the total length of the codewords can be prevented from exceeding the frame buffer size. As represented in Figure 8 (c), the unary code and the sign code that compose a variable-length data item place their first bits at one of the two opposite ends of their fields.

When decoding packed data with the format shown in Figure 8 (b), the fixed-length data are separated and sent to the appropriate decoding components. The fixed-length data include the 2-bit AK field, the 8-bit seed field, and the remainder field (the length of which is determined by AK), with the last one consisting of two variable-length data items. In the VLSplitter, the last fixed-length data item is split into two variable-length data items (i.e., a unary code and a sign code) by retrieving their bits that start from one of each of their field ends. If the value of the length field is equal to the original data length, the data at the location of the unary data is the quotient data.

IV. EXPERIMENTAL RESULTS

This section introduces the simulation results of the proposed algorithm and the implementation of the proposed hardware architecture. The effects of the PVSC algorithm on data compression are analyzed, and the proposed algorithms are compared with others [8], [18], [37] in terms of CR. The hardware implementation is designed with Verilog HDL and its evaluation is expressed in terms of clock frequency, throughput and unit/total area in a 55-nm cell library.

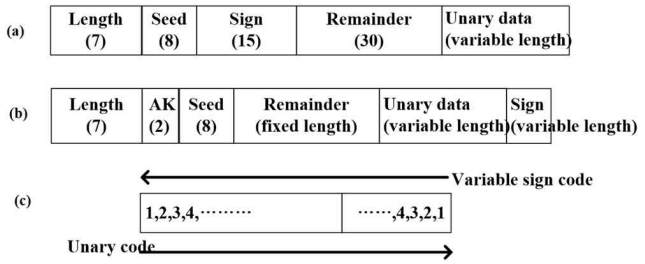

FIGURE 9. Data packing for variable-length data: (a) previous packing format, (b) proposed packing format, and (c) bit order of the variable-length data.

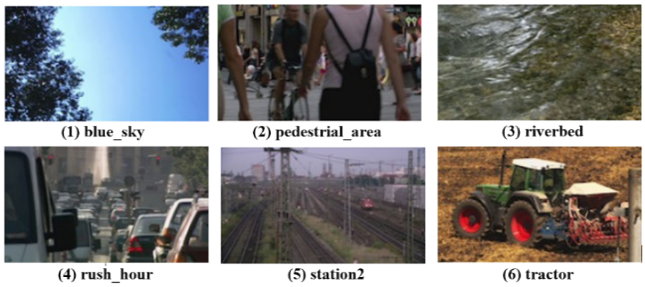

FIGURE 10. Full-High Definition (FHD) Benchmark images: (1) blue_sky, (2) pedestrial_area, (3) riverbed, (4) rush_hour, (5) station, and (6) tractor.

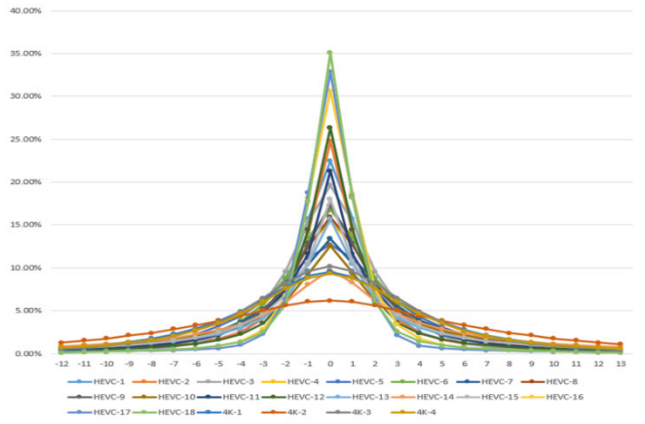

FIGURE 11. Histogram of residual data (HEVC-1: CLASS A TRAFFIC, HEVC-2: Class A PeopleOnStree, HEVC-3: Class B Kimono, HEVC-4: Class B ParkScene, HEVC-5: Class B Cactus, HEVC-6: Class B BasketballDrive, HEVC-7: Class B BQTerrace, HEVC-8: Class C BasketballDrill, HEVC-9: Class C BQMall, HEVC-10: Class C PartyScene, HEVC-11: Class C RaceHorses, HEVC-12: Class D BasketballPass, HEVC-13: Class D BQSquare, HEVC-14: Class D BlowingBubble, HEVC-15: Class D RaceHorses, HEVC-16: Class E Vidyo1, HEVC-17: Class E Vydyo3, HEVC-18: Class E Vidyo4, 4K-1: CrowdRun, 4K-2:DucksTakeOff, 4K-3: InToTree, and 4K-4: ParkJoy).

Regarding the benchmark sequence, we use two kinds of benchmark groups consisting of 18 HEVC benchmark sequences, and 4 4-K × 2-K sequences shown in Figure 13. The HEVC benchmark sequences are from the Joint Collaborative Team on Video Coding (JCT-VC) and the 4-K × 2-K sequences are from the Xiph.Org Foundation. They are shown in Table 1.

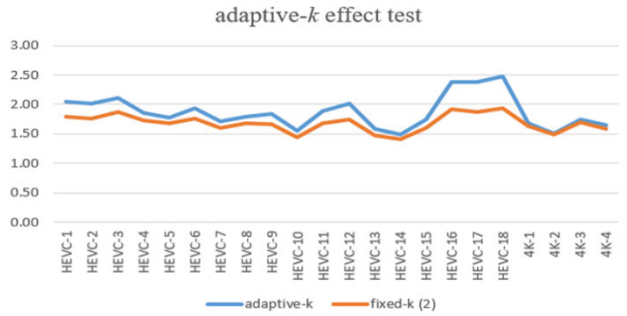

FIGURE 12. Histogram of adaptive-*k* and fixed-*k* on GR coding (HEVC-1: CLASS A TRAFFIC, HEVC-2: Class A PeopleOnStree, HEVC-3: Class B Kimono, HEVC-4: Class B ParkScene, HEVC-5: Class B Cactus, HEVC-6: Class B BasketballDrive, HEVC-7: Class B BQTerrace, HEVC-8: Class C BasketballDrill, HEVC-9: Class C BQMall, HEVC-10: Class C PartyScene, HEVC-11: Class C RaceHorses, HEVC-12: Class D BasketballPass, HEVC-13: Class D BQSquare, HEVC-14: Class D BlowingBubble, HEVC-15: Class D RaceHorses, HEVC-16: Class E Vidyo1, HEVC-17: Class E Vydyo3, HEVC-18: Class E Vidyo4, 4K-1: CrowdRun, 4K-2:DucksTakeOff, 4K-3: InToTree, and 4K-4: ParkJoy).

FIGURE 13. 4-K × 2-K Benchmark images: (a) CrowdRun, (2) DucksTakeOff, (c) InToTree and (4) ParkJoy.

A. COMPRESSION RATIOS

Figure 11 shows the distribution of the residual data obtained after DPCM processing. The distribution of 0 data was the highest in all test benches. When our PVSC was applied to these test benches, the sign bits decreased by up to 35%.

The compression efficiency of the proposed algorithm is evaluated with the CR using (3). The CR evaluation is performed on all benchmark sequences without and with quantization by applying the quantization parameter, QP, values of 22, 27, 32, and 37.

$$\mathrm{CR} = \left(\frac{\mathrm{Originaldatasize}}{\mathrm{Compresseddatasize}} \right) \tag{3}$$

Figure 12 shows the results of CR comparisons for all test benches between the coding algorithms of the adaptive-*k* GR and the fixed-*k* GR. The experiment was performed by limiting the adaptive-*k* to a range of 0 to 3 and by fixing the fixed-*k* to 2. As a result, the compression ratio increased in

TABLE 1. Bench mark list and compression condition.

	FHD image group	HEVC benchmark sequence group	4-K x 2-K test sequence group
Test sequence	1. blue_sky	Class A (Traffic, PeopleOnStreet)	1. CrowdRun
	2. pedestrian_area	Class B (Kimono, ParkScene, Cactus, BasketballDrive, BQTerrace)	2. DucksTakeOff
	3. riverbed	Class C (BasketballDrill, BQMall, partyScene, RaceHorses)	3. InToTree
	4. rush_hour	Class D (BasketballPass, BQSquare, BlowingBubble, RaceHorses)	4. ParkJoy
	5. station	Class E (vydyo1, vydyo3, vydyo4)	
	6. tractor		
QP parameter	0, 22, 27, 32, 37	22, 27, 32,37	22, 27, 32,37

TABLE 2. CRs (1: blue_sky, 2:pedestrian_area, 3: riverbed, 4: rush_hour, 5: station2, 6: tractor).

Test bench	CR	
	Luma	4:2:0
1	2.16	2.39
2	2.23	2.52
3	1.90	2.16
4	2.15	2.44
5	1.98	2.24
6	1.92	2.05
Avr.	2.06	2.30

TABLE 3. CRs (1: blue_sky, 2: pedestrian_area, 3: riverbed, 4: rush_hour, 5: station2, 6: tractor).

No	CR			
	QP=22	QP=27	QP=32	QP=37
1	3.19	3.30	3.48	3.78
2	3.43	3.71	4.11	4.69
3	2.42	2.71	3.27	4.18
4	3.59	3.86	4.28	4.84
5	2.72	3.03	3.57	4.35
6	2.28	2.45	2.78	3.48
Avr.	2.94	3.18	3.58	4.22

every test bench by 9.87% on average. The compression ratio increasing effect was high in HEVC-16 and HEVC-17 where the color and pattern are relatively simple.

Table 2 shows the simulation results of the CR of the proposed solution with an average CR of 2.06 sin Luma sample frames. It yielded an average CR of 2.30 in 4:2:0 frames.

Table 3 presents the results of the experiment that examined how the CR was influenced by QP. For this experiment, the FHD images were transformed into 8 × 8 block images through a discrete cosine transform (DCT) with four different QP values. The images were restored then via inverse DCT and inverse quantization. Next, the restored images were compressed using the proposed compression solution. The proposed solution achieved an average CR of 3.48. The CRs with regard to each QP value (i.e., 22, 27, 32, and 37) were 2.94, 3.18, 3.58, and 4.22, respectively. The lossless compression of images with a quality loss with a high QP parameter setting shows higher CR.

Table 5 and Table 4 show the CRs of the proposed and existing algorithms with HEVC sequence and 4-K sequences, respectively. In Table 5, the data of [37] was reused for the CRs of the existing algorithm. The test results with HEVC test sequences showed an average CR of 2.78, which is higher than the 1.7, 2.06, and 2.33 CRs of the previous studies. In Table 4, those with 4-K test sequences showed an average CR of 2.71, which is higher than the 1.7, 2.06, and 2.23 CRs of the previous studies.

B. PARALLEL HARDWARE IMPLEMENTATION

This paper proposes the parallel architecture for PVSC to code each sign bit in parallel and adaptive-k condition of Golomb-Rice coding algorithm. Their performance comparisons are made with other works using bytes per cycle and cycles per 8 × 8 blocks. In Table 6, the lower section shows the comparison results. The proposed parallel architecture achieves 64 bytes compression and decompression per cycle. That is, one clock cycle is required for 8 × 8 block data processing. This parallelism is at the same level as [8] and is higher than [18], [37].

The proposed hardware architecture with a four-stage pipeline was designed for the encoder and decoder with Verilog HDL (Hardware Description Language). It was implemented with a 55-nm standard cell library up to the synthesis

TABLE 4. CRs of the proposed and previous algorithms (1: CrowdRun, 2:DucksTakeOff, 3: InToTree, 4: ParkJoy).

No	CR, QP=0				CR, QP=22				CR, QP=27				CR, QP=37			
	[8]	[18]	[37]	**Prop.**	[8]	[18]	[37]	**Prop.**	[8]	[18]	[37]	**Prop.**	[8]	[18]	[37]	**Prop.**
1	1.52	1.62	1.66	**1.69**	1.76	1.96	2.18	**2.46**	1.82	2.07	2.33	**2.74**	1.87	2.23	2.51	**3.65**
2	1.28	1.44	1.45	**1.51**	1.52	1.91	2.05	**2.23**	1.70	2.15	2.33	**2.70**	1.89	2.28	2.48	**3.82**
3	1.53	1.67	1.69	**1.74**	1.76	1.54	1.63	**1.83**	1.84	2.17	2.36	**2.36**	1.90	2.44	2.65	**4.33**
4	1.49	1.58	1.61	**1.65**	1.71	1.86	1.97	**2.39**	1.78	2.66	2.91	**3.01**	1.85	3.47	3.82	**5.24**
Avr.	1.46	1.58	1.60	**1.65**	1.69	1.82	1.96	**2.23**	1.79	2.26	2.48	**2.70**	1.88	2.61	2.88	**4.26**

TABLE 5. CRs of the proposed and previous algorithms (HEVC-1: CLASS A TRAFFIC, HEVC-2: Class A PeopleOnStree, HEVC-3: Class B Kimono, HEVC-4: Class B ParkScene, HEVC-5: Class B Cactus, HEVC-6: Class B BasketballDrive, HEVC-7: Class B BQTerrace, HEVC-8: Class C BasketballDrill, HEVC-9: Class C BQMall, HEVC-10: Class C PartyScene, HEVC-11: Class C RaceHorses, HEVC-12: Class D BasketballPass, HEVC-13: Class D BQSquare, HEVC-14: Class D BlowingBubble, HEVC-15: Class D RaceHorses, HEVC-16: Class E Vidyo1, HEVC-17: Class E Vydyo3, HEVC-18: Class E Vidyo4).

No	CR, QP=0				CR, QP=22				CR, QP=27				CR, QP=37			
	[8]	[18]	[37]	**Prop.**	[8]	[18]	[37]	**Prop.**	[8]	[18]	[37]	**Prop.**	[8]	[18]	[37]	**Prop.**
1	1.70	1.86	2.03	**2.04**	1.79	2.13	2.38	**2.72**	1.81	2.21	2.47	**3.02**	1.86	2.29	2.53	**4.01**
2	1.70	1.89	2.08	**2.01**	1.77	1.98	2.21	**2.55**	1.79	2.08	2.34	**2.83**	1.83	2.17	2.43	**3.54**
3	1.76	2.01	2.05	**2.11**	1.87	2.65	2.91	**3.28**	1.88	2.73	2.99	**3.66**	1.89	2.73	2.97	**4.94**
4	1.62	1.39	1.82	**1.86**	1.77	2.00	2.20	**2.57**	1.81	2.13	2.37	**2.92**	1.87	2.28	2.50	**4.16**
5	1.56	1.27	1.73	**1.77**	1.74	2.15	2.39	**2.59**	1.78	2.34	2.65	**2.96**	1.84	2.51	2.83	**3.96**
6	1.62	1.41	1.90	**1.94**	1.78	2.41	2.76	**3.10**	1.81	2.62	3.05	**3.58**	1.85	2.74	3.15	**4.53**
7	1.46	1.33	1.73	**1.71**	1.60	1.77	1.97	**2.27**	1.69	2.03	2.34	**2.83**	1.78	2.22	2.53	**3.59**
8	1.55	1.69	1.79	**1.80**	1.69	1.78	1.97	**2.29**	1.74	1.96	2.19	**2.62**	1.83	2.24	2.50	**3.74**
9	1.56	1.72	1.87	**1.84**	1.69	2.01	2.32	**2.44**	1.72	2.08	2.41	**2.64**	1.79	2.14	2.43	**3.41**
10	1.33	1.46	1.56	**1.55**	1.44	1.42	1.54	**1.82**	1.49	1.50	1.65	**1.99**	1.65	1.65	1.84	**2.71**
11	1.57	1.77	1.88	**1.88**	1.66	1.89	2.12	**2.31**	1.70	1.95	2.16	**2.54**	1.81	2.10	2.34	**3.64**
12	1.62	1.78	2.02	**2.02**	1.70	1.94	2.17	**2.48**	1.73	2.04	2.29	**2.73**	1.82	2.23	2.51	**3.62**
13	1.36	1.50	1.58	**1.59**	1.45	1.39	1.53	**1.93**	1.49	1.49	1.66	**2.16**	1.60	1.73	1.95	**2.78**
14	1.30	1.42	1.52	**1.49**	1.42	1.38	1.51	**1.70**	1.48	1.44	1.59	**1.84**	1.67	1.66	1.84	**2.62**
15	1.50	1.64	1.81	**1.74**	1.61	1.68	1.84	**2.00**	1.66	1.74	1.92	**2.20**	1.80	2.03	2.26	**3.31**
16	1.78	2.23	2.43	**2.39**	1.83	2.77	3.29	**3.40**	1.84	2.85	3.37	**3.67**	1.87	2.85	3.26	**4.60**
17	1.73	2.21	2.42	**2.37**	1.78	2.93	3.48	**3.42**	1.79	3.01	3.59	**3.58**	1.83	2.94	3.45	**4.11**
18	1.77	2.29	2.49	**2.48**	1.83	2.84	3.30	**3.50**	1.84	2.96	3.44	**3.80**	1.87	2.98	3.43	**4.67**
Avr.	1.58	1.72	1.93	**1.92**	1.69	2.06	2.33	**2.58**	1.73	2.18	2.47	**2.87**	1.80	2.31	2.60	**3.77**

step (synopsys dc). After synthesis, the maximum operating frequency was 370 MHz and 286 MHz for the encoder and the decoder, respectively. Both performed massively parallel processing, yielding a throughput of 24 GBps and 18 GBps for 8 × 8 blocks, a total gate count of 83 K and 121 K, and a gate count per pixel of 1.3 K and 1.9 K, respectively.

Table 7 summarizes the hardware implementation results in terms of hardware performance, total area, and unit area. The proposed solution achieved better hardware performance and lower power consumption than the previous algorithms.

The proposed hardware architecture had a throughput of 24 GBps during encoding and 18 GBps during decoding, which was the highest rate among the compared algorithms. This can be explained when considering two key points. First, the proposed architecture provides the massively parallel processing of [8], whereas the algorithms in [18], [37] have data-processing dependencies during compression and decompression. Second, the proposed hardware implementation has a higher operating frequency than the implementation in [8] thanks to pipeline depth adjustment, logic optimization, and high-end manufacturing.

The hardware of the proposed solution requires the smallest gate counts per pixel, which leads to the lowest power consumption per pixel. All these characteristics make the proposed compression solution suitable for high-performance mobile applications.

Our decompressor is larger than our compressor. Our proposal uses adders for parallel processing. For unary parallel

TABLE 6. Comparisons of CR and hardware parallelism.

Algorithm		[18] HACP+SBT	[37] MDA+SFL	[8] DDPCM+ fixed-k GR	Proposed LEC DPCM+ adaptive-k GR + PVSC	
Compression performance	QP condition	22,27,32,37	22,27,32,37	0	22,27,32,37	0
	CR	2.20	2.49	1.52	3.12	2.06
Hardware parallelism	Bytes per cycle	5.1/14.2	10.7/21.3	64	64	
	Cycles per 8x8 block	6/2	6/3	1/1	1/1	

TABLE 7. Hardware implementation summary.

Algorithm	[18] HACP+SBT	[37] MDA+SFL	[8] DDPCM+ fixed-k GR	Proposed LEC DPCM+ adaptive-k GR + PVSC
Technology	180 nm	90 nm	150 nm	55 nm
Block Size	16x8	8x8	8x8	8x8
max Frequency (MHz)	180	300	202/136	370/286
Throughput (Mbyte/sec)	900/2,556	3,200/6,400	12,928/8,704	23,704/18,286
Unit Area ((Gate x cycle) /pixel)	7K/2.5K	4.2K/1.6K	3.2K/3.2K	1.3K/1.9K
Logic (GC)	36.1K	45K/34K	202K/231K	83K/121K
Power consumption (mW)	N.A.*	N.A.*	N.A.*	23/17.6

N.A.*: non available information. We mark N.A for [18], [37], and [8] because there are no power results.

processing, encoding requires $M \times N - 1$ adders to compute the termination position. Decoding requires a number of adders by unary length to reconstruct quotients. As a result, the number of adders makes the decompressor larger than the compressor.

Even though the proposed method adds some hardware modules for SignENC/SignDEC, zeroDT, and KSplitter, the hardware cost is reduced compared with those in [8]. This result occurred for two reasons. First, the subtraction operation step has been shortened from two to one because we replaced the DDPCM algorithm adopted in [8] with the DPCM algorithm. Second, the logic complexity for the adders on unary coding or inverse DPCM has been reduced by performing the pipeline depth adjustment of [8].

V. CONCLUSION

This paper proposed a lossless compression solution by developing novel algorithms for frame-buffer recompression and by extending some of the previous compression methods. It also proposed hardware architecture that allows massively parallel processing for the compressor and decompressor. The proposed solution was implemented in a lossless embedded compressor. This hardware implementation has an average CR of 3.12 and an operating frequency of 370 MHz. Its throughput is 24 GBps, which exceeds the throughput requirement for 8-K UHD image processing at 240 Hz (i.e., 12 GBps). It occupies 1.3K gate counts for single pixel processing, which leads to lower power consumption (23mW for the compressor and 17.6mW for the decompressor). Therefore, the proposed solution is suitable for use in mobile applications where energy efficiency is a significant factor. In addition, the size of the compression unit ($M \times N$ blocks) can be adjusted, so the proposed solution can be used in line-based applications and in block-based applications.

The work presented in this paper focuses on compression algorithms for the entropy coding stage and parallel-processing architecture. In the future, prediction-stage compression algorithms will be studied to further improve compression efficiency. In addition, the current architecture offering block-level random access will be extended to provide intra-block random access. We also need to study that a hybrid compression algorithm to ensure fixed bandwidth requirements. The hybrid algorithm can be a mixture of both lossy and lossless algorithms.

REFERENCES

[1] A. Savakis and M. Piorun, "Benchmarking and hardware implementation of JPEG-LS," in *Proc. Int. Conf. Image Process. (ICIP)*, Rochester, NY, USA, Sep. 2002, p. 2.

[2] X. Wu and N. Memon, "CALIC-a context based adaptive lossless image codec," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP)*, vol. 4, May 1996, pp. 1890–1893.
[3] M. J. Weinberger, G. Seroussi, and G. Sapiro, "LOCO-I: A low complexity, context-based, lossless image compression algorithm," in *Proc. Data Compress. Conf.*, Mar./Apr. 1996, pp. 140–149.
[4] A. D. Mitra and P. K. Srimani, "Differential pulse-code modulation," *Int. J. Electron.*, vol. 46, pp. 633–637, Jun. 1972.
[5] S. Mallat, "A theory for multiresolution signal decomposition: The wavelet representation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 11, no. 7, pp. 674–693, Jul. 1989.
[6] H. Gao, F. Qiao, and H. Yang, "Lossless memory reduction and efficient frame storage architecture for HDTV video decoder," in *Proc. Int. Conf. Audio Lang. Img. Process.*, Jul. 2008, pp. 593–598.
[7] S.-H. Lee, M.-K. Chung, S.-M. Part, and C.-M. Kyung, "Lossless frame memory recompression for video codec preserving random accessibility of coding unit," *IEEE Trans. Consum. Electron.*, vol. 55, no. 4, pp. 2105–2113, Nov. 2009.
[8] H.-S. Kim, J. Lee, H. Kim, S. Kang, and W. Park, "A lossless color image compression architecture using a parallel golomb-rice hardware CODEC," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 21, no. 11, pp. 1581–1587, Nov. 2011.
[9] S. A. Martucci, "Reversible compression of HDTV images using median adaptive prediction and arithmetic coding," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 1990, pp. 1310–1313.
[10] M. J. Weinberger, G. Seroussi, and G. Sapiro, "The LOCO-I lossless image compression algorithm: Principles and standardization into JPEG-LS," *IEEE Trans. Image Process.*, vol. 9, no. 8, pp. 1309–1324, Aug. 2000.
[11] J. Kim, J. Kim, and C. Kyung, "A lossless embedded compression algorithm for high definition video coding," in *Proc. IEEE Int. Conf. Multimedia Expo*, New York, NY, USA, Jun./Jul. 2009, pp. 193–196.
[12] M. Papadonikolakis, V. Pantazis, and A. P. Kakarountas, "Efficient high-performance ASIC implementation of JPEG-LS encoder," in *Proc. Int. Design Autom. Test Eur. Conf Exhibit.*, Apr. 2007, pp. 1–6.
[13] T. H. Tsai, Y. H. Lee, and Y. Y. Lee, "Design and analysis of high-throughput lossless image compression engine using VLSI-oriented FELICS algorithm," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 18, no. 1, pp. 39–52, Jan. 2010.
[14] 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.
[15] Y. Li, W. Wang, and G. Zhang, "Hybrid pixel encoding: An effective display frame compression algorithm for HD video decoder," in *Proc. IEEE 15th Int. Conf. Comput. Sci. Eng.*, Dec. 2012, pp. 303–309.
[16] Y. Jiang, Y. Li, D. Ban, and Y. Xu, "Frame buffer compression without color information loss," in *Proc. IEEE 12th Int. Conf. Comput. Inf. Technol.*, Oct. 2012, pp. 12–17.
[17] Y. Li, Y. Jiang, and H. Meng, "Adaptive pixel encoding: An effective algorithm for frame buffer compression," in *Proc. IEEE 12th Int. Conf. Comput. Inf. Technol.*, Oct. 2012, pp. 5–11.
[18] J. Kim and C.-M. Kyung, "A lossless embedded compression using significant bit truncation for HD video coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 20, no. 6, pp. 848–860, Jun. 2010.
[19] W.-Y. Chen, L.-F. Ding, P.-K. Tsung, and L.-G. Chen, "Architecture design of high performance embedded compression for high definition video coding," in *Proc. IEEE Int. Conf. Multimedia Expo*, Hannover, Germany, Jun./Apr. 2008, pp. 825–828.
[20] S. Lee, N. Eum, M.-K. Chung, and C.-M. Kyung, "Low latency variable length coding scheme for frame memory recompression," in *Proc. IEEE Int. Conf. Multimedia Expo*, Jul. 2010, pp. 232–237.
[21] R. Moussalli, W. Najjar, X. Luo, and A. Khan, "A high throughput no-stall Golomb-Rice hardware decoder," in *Proc. IEEE 21st Annu. Int. Symp. Field-Program. Custom Comput. Mach.*, Seattle, WA, USA, Apr. 2013, pp. 65–72.
[22] 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. 2, pp. 1277–1291, Oct. 2010.
[23] K. Denecker, M. V. D. Ville, F. Habils, W. Meeus, M. Brunfaut, and I. Lemahieu, "Design of an improved lossless halftone image compression codec," *Signal Process., Image Commun.*, vol. 17, no. 3, pp. 277–292, Mar. 2002.
[24] L. Brooks and K. Fife, "Hardware efficient lossless image compression engine," in *Proc. IEEE Int. Conf. Acous., Speech, Signal Process.*, May 2004, pp. 17–21.
[25] X. Chen, N. Canagarajah, and J. L. Nunez-Yanez, "Lossless multimode interband image compression and its hardware architecture," in *Proc. Algorithm-Architecture Matching Signal Image Process.*, 2008, pp. 208–215.
[26] M. Milward, J. L. Nunez, and D. Mulvaney, "Design and implementation of a lossless parallel high-speed data compression system," *IEEE Trans. Parallel Distrib. Syst.*, vol. 15, no. 6, pp. 481–490, Jun. 2004.
[27] C.-C. Cheng, P.-C. Tseng, C.-T. Huang, and L.-G. Chen, "Multi-mode embedded compression codec engine for power-aware video coding system," in *Proc. IEEE Workshop Signal Process. Syst.*, Nov. 2005, pp. 532–537.
[28] X. Li, X. Chen, X. Xie, G. Li, L. Zhang, C. Zhang, and Z. Wang, "A low power, fully pipelined JPEG-LS encoder for lossless image compression," in *Proc. IEEE Int. Conf. Multimedia EXPO*, Jul. 2007, pp. 1906–1909.
[29] X. Chen, N. Canagarajah, J. L. Nunez-Yanez, and R. Vitulli, "Hardware architecture for lossless image compression based on context-based modeling and arithmetic coding," in *Proc. IEEE Int. SOC Conf.*, Sep. 2007, pp. 251–254.
[30] T. Song and T. Shimamoto, "Reference frame data compression method for H.264/AVC," *IEICE Electron. Express*, vol. 4, pp. 121–126, Jan. 2007.
[31] Y.-H. Lee, Y.-Y. Lee, H.-Z. Lin, and T.-H. Tsai, "A high-speed lossless embedded compression codec for high-end LCD applications," in *Proc. IEEE Asian Solid-State Circuits Conf.*, Nov. 2008, pp. 185–188.
[32] Y.-Y. Lee, Y.-H. Lee, and T.-H. Tsai, "An efficient lossless embedded compression engine using compacted-FELICS algorithm," in *Proc. IEEE Int. SOC Conf.*, Sep. 2008, pp. 233–236.
[33] 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.
[34] T.-C. Chen, Y.-H. Chen, K.-C. Wu, and L.-G. Chen, "Hybrid-mode embedded compression for H.264/AVC video coding system," in *Proc. Intl. Symp. Intell. Signal Process. Commun. Syst. (ISPACS)*, Dec. 2005, pp. 257–260.
[35] Y.-X. Lee and T.-H. Tsai, "An efficient embedded compression algorithm using adjusted binary code method," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2008, pp. 2586–2589.
[36] D. Zhou, J. Zhou, X. He, J. Zhu, J. Kong, P. Liu, and S. Goto, "A 530 mpixels/s 4096×2160@60fps H.264/AVC high profile video decoder chip," *IEEE J. Solid-State Circuits*, vol. 46, no. 4, pp. 777–788, Apr. 2011.
[37] L. Guo, D. Zhou, and S. Goto, "A new reference frame recompression algorithm and its VLSI architecture for UHDTV video codec," *IEEE Trans. Multimedia*, vol. 16, no. 8, pp. 2323–2332, Dec. 2014.
[38] X. Lian, Z. Liu, W. Zhou, and Z. Duan, "Lossless frame memory compression using pixel-grain prediction and dynamic order entropy coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 26, no. 1, pp. 223–235, Jan. 2016.
[39] S. W. Golomb, "Run-length Codings," *IEEE Trans. Inf. Theory*, vol. 12, no. 7, pp. 399–401, 1966.
[40] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proc. Inst. Radio Eng.*, vol. 40, no. 9, pp. 1098–1101, Sep. 1952.
[41] 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," *Display J.*, vol. 40, pp. 68–77, Dec. 2015.
[42] Y.-Z. Kao, K.-H. Heung, S.-S. F. Jiang, and Y.-H. Lee, "A novel lossless embedded compression algorithm for video coding for wireless sensor node applications," in *Proc. IEEE Int. Conf. Consum. Electron. Taiwan (ICCE-TW)*, Jun. 2017, pp. 103–104.
[43] C. Gu, X. Zeng, and Y. Fan, "A 5.3 Gpixels/s frame memory recompression method for QHD video coding," in *Proc. 14th IEEE Int. Conf. Solid-State Integr. Circuit Technol. (ICSICT)*, Oct./Nov. 2018, pp. 1–3.