A Lossless Color Image Compression Architecture Using a Parallel Golomb-Rice Hardware CODEC

Hong-Sik Kim, Joohong Lee, Hyunjin Kim, Sungho Kang, and Woo Chan Park, *Member, IEEE*

***Abstract*—In this paper, a high performance lossless color image compression and decompression architecture to reduce both memory requirement and bandwidth is proposed. The proposed architecture consists of differential-differential pulse coded modulation (DDPCM) and Golomb-Rice coding. The original image frame is organized as m by n sub-window arrays, to which DDPCM is applied to produce one seed and $m \times n - 1$ pieces of differential data. Then the differential data are encoded using the Golomb-Rice algorithm to produce losslessly compressed data. According to the experimental results on benchmark images, the proposed architecture can guarantee high enough compression rate and throughput to perform real-time lossless CODEC operations with a reasonable hardware area.**

***Index Terms*—Compression rate, DDPCM, Golomb-Rice coding, lossless image compression.**

I. Introduction

IN THIS PAPER, we consider the problem of hardware-based lossless compression of color image data. In requests for real-time image processing on massive multimedia jobs, high performance communication between processor and memory is required. In addition, increasing performance gaps between embedded processors and external memories have resulted in greater bandwidth requirement in the communication networks or data buses. There still exist critical limitations on the physical size of a bus due to the restricted operating speed from signal integrity issues and increased hardware area. Compressed memory architecture can be a good alternative solution to the above problems since it can significantly reduce requirements for increasing bus bandwidth [1], [2].

The compressed memory architecture should utilize a lossless compression algorithm because the original image should be reconstructed from the compressed one. In addition, a hardware-based, rather than a software-based, compression methodology is required in order to implement real-time features for data processing.

Manuscript received August 2, 2009; revised March 1, 2010 and May 24, 2010; accepted November 5, 2010. Date of publication March 17, 2011; date of current version November 2, 2011. This paper was recommended by Associate Editor M. Berekovic.

H.-S. Kim, H. Kim, and S. Kang are with the Department of Electrical and Electronic Engineering, Yonsei University, Seoul 120-749, Korea (e-mail: hongsik@yonsei.ac.kr; nagicman@soc.yonsei.ac.kr; shkang@yonsei.ac.kr).

J. Lee and W. C. Park are with the Department of Computer Engineering, Sejong University, Seoul 143-747, Korea (e-mail: jhlee@rayman.sejong.ac.kr; pwchan@sejong.ac.kr).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSVT.2011.2129350

Lossless image compression has been addressed with Lempel-Ziv code [3], Golomb-Rice code [4], JPEG-LS [5], and CALIC [6]. Most of these methodologies are software-based approaches targeting a high compression rate. They require complex hardware implementations to achieve sufficient performance of compression for color image data. In [7], in order to address the performance penalty which software-based approaches usually accompany, the working sets are selectively compressed according to the program phase in which the changes are detected.

In [8]–[11], the compressed memory schemes to improve the performance of embedded systems have been proposed. Operating system-based memory compression architecture is proposed to provide on-the-fly compression and decompression for embedded systems [8]. In [9], discrete cosine transform-based memory compression technique, which is a lossy compression technique, is applied to the motion-estimation applications. In [10], image compression hardware architecture based on Hadamard transform and Golomb-Rice coding is proposed. It, however, loses some data during the Hadamard transform and Golomb-Rice encoding. In [11], a new hardware-friendly adaptive decimation algorithm is proposed to achieve low bit rate and less image quality degradation for color images, which belongs to the lossy compression technique. So the approaches in [9]–[11] are not adequate for applications, such as most medical systems, where loss of fidelity cannot be tolerated in compression and decompression operations.

Many researches on hardware implementation of lossless compression and decompression have been proposed in [12]–[18]. In the previous schemes, since their complex coding algorithms employ serious data dependency, the hardware complexity and the performance degradation are increased so that they are not adequate for the high throughput applications.

In this paper, a new hardware architecture for lossless color image compression and decompression to improve throughput by exploiting massive parallelism is proposed. The proposed architecture consists of differential-differential pulse coded modulation (DDPCM) [19] and Golomb-Rice encoding. Original image frames are organized as m by n sub-window arrays, to which DDPCM is applied to produce one seed and $m \times n - 1$ pieces of differential data. Then, the Golomb-Rice algorithm encodes the differential data into variable length codeword. Experimental results on real benchmark images show that the proposed scheme can achieve high enough compression rate

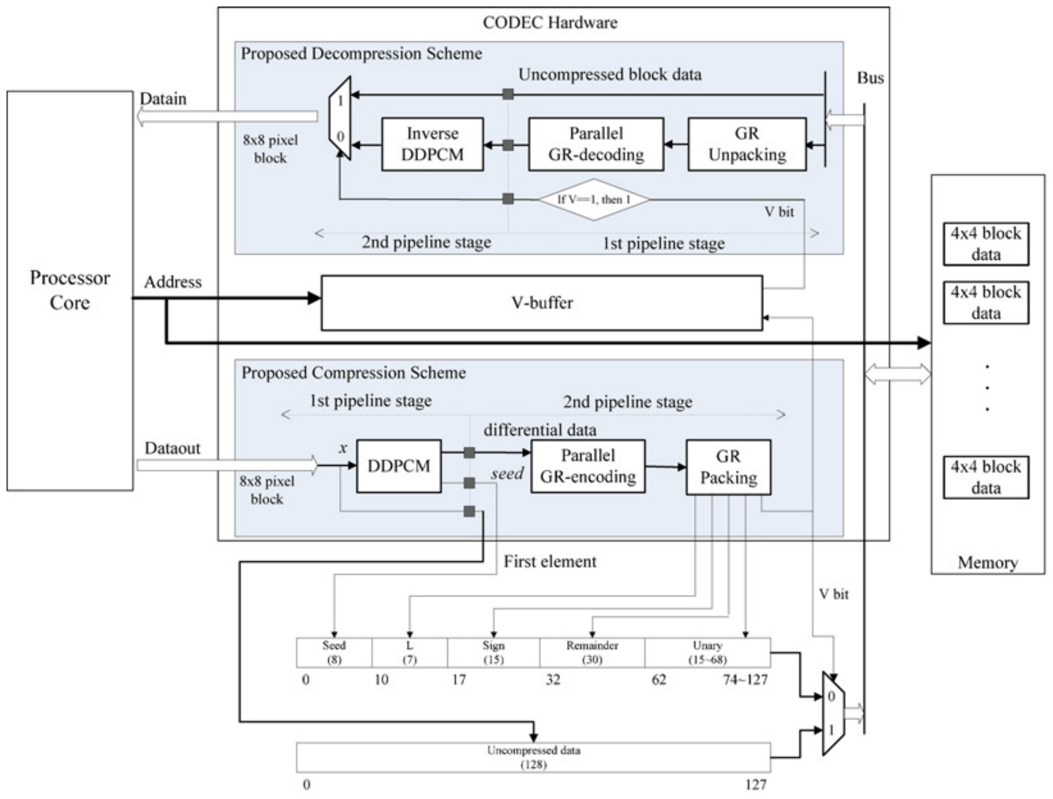


Fig. 1. Proposed hardware architecture for compression/decompression and segment data format.

and throughput to perform real-time lossless encoding and decoding with a reasonable hardware area.

This paper is organized as follows. Sections II and III describe the proposed compression and decompression hardware architectures, separately. Section IV provides experimental results on real benchmark image data and Section V concludes this paper.

II. Proposed Compression Architecture

Fig. 1 shows the architecture for the proposed compression and decompression algorithms and the segment data format. The proposed compression architecture consists of DDPCM step and Golomb-Rice encoding step. In the first step, DDPCM is applied to 4×4 block data (the sub-window size is assumed to be 4×4 throughout this paper unless otherwise described) to produce one piece of seed and 15 pieces of differential data. The seed is used in its uncompressed form while the differential data are encoded by Golomb-Rice coding algorithm to produce the compressed data with variable length.

If the size of the compressed data is smaller than the size of the uncompressed one, then the Golomb-Rice encoded data is stored in the memory. Otherwise, the uncompressed data is selected. Two types of segment data formats are used in order to distinguish the two cases as shown in Fig. 1. A V bit indicates whether the data is compressed or not. For example, if the V bit is “0,” the data is meant to be compressed. Otherwise, the data is uncompressed and the original data ($16 \times 8 = 128$ bits) follows. This V bit information is stored at the V-buffer of the codec hardware as shown in Fig. 1 and the remaining data (75–128 bit variable codeword when compressed and 128 bit original data when uncompressed) are stored in the memory.

In the *Seed* field, the seed generated by DDPCM operation is stored. The 15 bit *Sign* field contains information about the signs of the 15 pieces of differential data calculated during DDPCM step.

According to the Golomb-Rice algorithm, the original input data is divided by 2^k, to produce quotient data which will be unary coded into a variable length unary codeword (15–68 bits) and remainder data which will be used as a binary remainder codeword. In this paper, the Golomb-Rice parameter of k is assumed to be 2. In the *Remainder* field, fixed-length encoded data of 15×2 bits for 15 remainders is stored. In addition, the *Unary* field stores the variable length unary codeword for 15 quotients. Since the size of unary codeword is variable, its length should be identified by L field.

The compression step is divided into two pipeline stages as shown in Fig. 1 in order to reduce the critical path delay. In the first stage, DDPCM-based preprocessing operations are performed, and one piece of seed and 15 pieces of differential data are latched and delivered to the second stage. In the second stage, 15 pieces of differential data are compressed by Golomb-Rice encoder and the encoded data is packed according to the segment data format. In this section, the proposed compression structure including a Golomb-Rice encoder and a data packing module will be described in detail. For detailed implementation of DDPCM operations, please refer to [10].

A. Golomb-Rice Encoder

The Golomb-Rice encoder receives 15 pieces of 10-bit differential data from the DDPCM module and generates

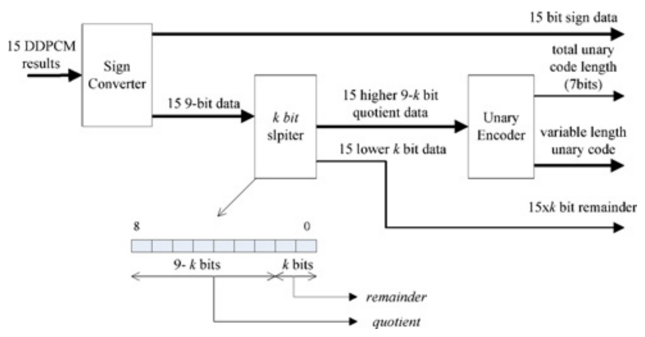


Fig. 2. Golomb-Rice encoder hardware structure.

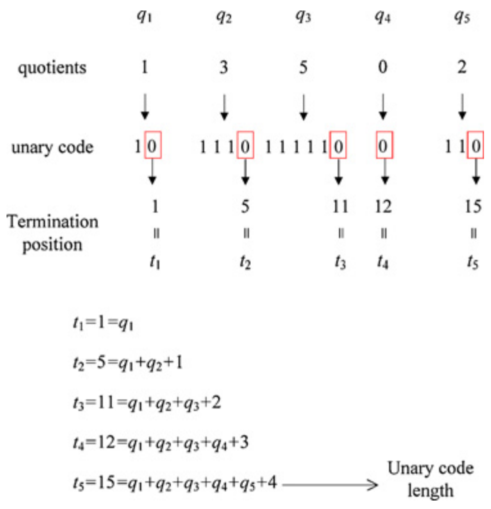


Fig. 3. Example of unary encoding.

four outputs which are sign values (15 bits), 15 remainder data (15×2 bits), variable length unary-encoded quotient data (15–68 bits), and unary code length (7 bits). In the DDPCM preprocessing step, 8-bit input data is processed by two successive subtractions so that the length of the DDPCM result data becomes 10 bits.

During unary encoding, if the sign of the input data is restricted to positive numbers, then unary encoding can be performed more efficiently since one unary code can stand for two symbols, e.g., +7 and −7 can be encoded by one unary code of 10 for $k = 2$. For this purpose, negative input values are converted into positive ones, which are encoded by the Golomb-Rice algorithm.

Fig. 2 describes the proposed Golomb-Rice hardware architecture. Since the Golomb-Rice algorithm uses only a power-of-two divider, the k-bit splitter can replace the complex divider as shown in Fig. 2. The splitter assigns the lower k bits of the input data to a remainder and the higher 10-k bits to a quotient. The unary encoder compresses the 15 pieces of quotient data with unary codes. For example, if a quotient is 5, then it is encoded with five successive 1s and one terminating 0, or 111110. The lengths of 15 pieces of encoded data can vary, which means that both the positions of the terminating 0s and the total length of the compression data for 15 quotients are variable.

The operation of the unary encoder will be described with a simple example in Fig. 3. In the example, there are five quotients to be unary encoded. The unary-encoded sequence shall be "1011101111100110." In order to generate this encoded sequence, the positions of the terminating 0s are calculated by adding all previous quotients including the current one and index numbers. For example, the third terminating position is calculated by $t_3 = q_1 + q_2 + q_3 + 2$. Therefore, without loss of generality, the i_{th} terminating position, t_i, can be calculated by (1) and the last terminating position, t_{15}, is to be the total length of the unary-encoded data as follows:

$$t_i = \sum_{j=1}^{i} q_j + i - 1, (1 \le i \le 15). \tag{1}$$

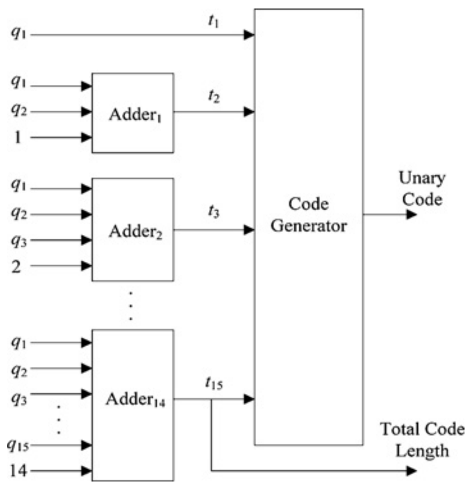


Fig. 4. Unary encoder hardware structure.

The equation to calculate each terminating zero position is implemented by separate adders to improve calculation throughput as shown in Fig. 4. After each adder calculates the position of the corresponding terminating 0 by accumulating adequate quotient values and index numbers, the code generator, which is a simple fully combinational circuit, produces the final unary code according to the terminating positions. Since the length of the unary code is variable, the length information should be delivered to recover the original data. The code length is exactly the same as the last terminating position, t_{15}, so that the position value of t_{15} is used to stand for the total unary code length.

B. Data Packing

Once the Golomb-Rice encoding operation is finished, the final compression data are produced by packing the encoded codeword and information data according to the format in Fig. 1. Sometimes, the length of the encoded data can be greater than the length of the uncompressed data. In this case, the original uncompressed data for 16 pixels are packed together with the flag bit, V, set to 1. Otherwise, a piece of seed data, a unary code length, the sign information of the DDPCM results, the binary coded remainders, and the unary encoded quotient data are packed together with the flag bit, V, set to 0. The corresponding V bits are stored at the V-buffer of the codec hardware and the remaining data (75–128 bit encoded codeword when compressed and 128 bit original data when uncompressed) are stored in the memory.

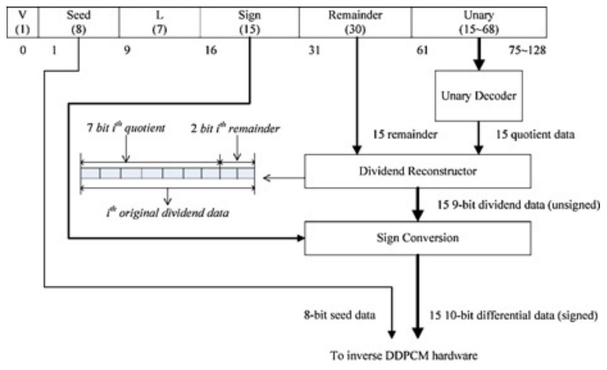


Fig. 5. Proposed decompression architecture.

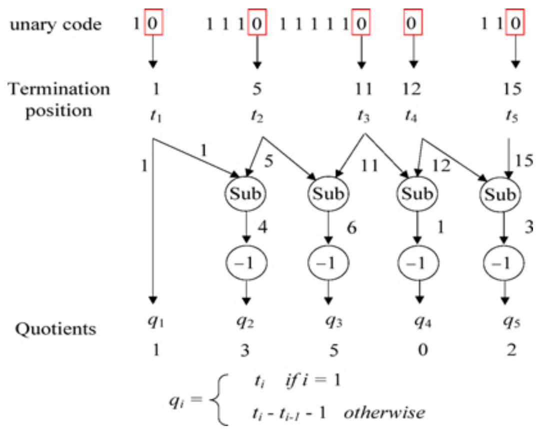


Fig. 6. Proposed unary decoder.

III. Proposed Decompression Architecture

The decompression process is simply the inverse of the compression process. As shown in Fig. 5, the whole decompression flow consists of three steps of unpacking, Golomb-Rice decoding, and inverse DDPCM. When a piece of sub-window array data reaches, the unpacking hardware module decides whether the data is a compressed one or not according to the value of the V bit stored in the V-buffer. If the V bit is 1, then the data is directly forwarded to the processor without any processing. Otherwise, Golomb-Rice decoding and inverse DDPCM operations are performed to reconstruct the original 4×4 sub-window image data. The Golomb-Rice decoder includes a unary decoder, a dividend reconstructor, and a sign converter as shown in Fig. 5.

The decompressor is divided into two pipeline stages in order to shorten the critical path delay. In the first stage, compressed data is decoded by a Golomb-Rice decoder to deliver one piece of seed and 15 pieces of differential data to the second stage. In the second stage, inverse DDPCM is applied to those pipelined data and original 4×4 sub-window array data are reconstructed.

A. Golomb-Rice Decoding

Two pieces of informative data such as a remainder binary codeword and a quotient unary codeword have been produced by the Golomb-Rice encoder. The remainder binary codeword does not need to be decoded, since the original remainder data were directly used for the remainder binary codeword. On the contrary, the quotient unary codeword should be decoded to reconstruct the original quotient data.

Once 15 pieces of quotient data are reconstructed by the unary decoder, 15 pieces of unsigned dividend data will be recovered by putting together remainders and quotients as shown in Fig. 5. Then the 15 pieces of unsigned differential data are converted into the original signed data according to the information in the *Sign* field. Finally, the 8-bit seed and 15 pieces of 10-bit differential data construct the DDPCM data which will be processed to reconstruct the original 4×4 sub-window image by the inverse DDPCM operation.

B. Unary Decoding

The unary decoder recovers the original 15 pieces of quotient data from the unary codeword, which has variable length between 15 bits and 68 bits. Each piece of decoded quotient data has a fixed length of 8 bits. Since a unary code word contains 15 pieces of compressed quotient data, there exist 15 terminating 0s within the variable length unary code word. At first, the proposed unary decoding hardware determines the position of terminating 0s and then reconstructs the original quotient data by subtracting the successive two position values.

Fig. 6 illustrates the unary decoding process of the proposed decoder with the same example used in Fig. 4. The unary decoding process is exactly the inverse of unary encoding. In the example, the 16-bit unary codeword contains five quotients so that five terminating 0s are included in the unary codeword. The locations of the 0s are calculated and then the final quotients are generated according to the equation by the unary decoder hardware as shown in Fig. 6. In order to improve the throughput of the whole system, all 15 positions are calculated in parallel.

IV. Experimental Results

A. Simulation Results

For the experiment, several benchmark images are selected from real video data (benchmarks 1–4) and Quake 3 game images (benchmarks 5–7) as shown in Fig. 7. The image sizes of the benchmarks 1–4 are 352×288 and the image sizes of the benchmark 5–7 are 320×240. The proposed compression and decompression algorithms are implemented in C++. Each piece of benchmark image data is constructed according to a 4:2:0 YUV format. Therefore, YUV image frames are separately organized as $m \times n$ sub-window array, to which DDPCM is applied to produce one seed and $m \times n - 1$ pieces of differential data. For the experiments, four different combinations of sub-window arrays have been considered, such as 4×4, 8×4, 8×8, and 16×8 sub-window arrays. Then the differential data are encoded by the Golomb-Rice algorithm to produce losslessly compressed data. The final compression ratio is calculated by dividing the number of bits to store the original input image with the number of bits to store the compressed data.

In Table I, the compression ratios of the proposed algorithm for the benchmark images in case of four sub-window arrays

TABLE I
COMPRESSION RATIO OF THE PROPOSED LOSSLESS COMPRESSION ALGORITHM

	Compression Ratio															
	4×4 Array				8×4 Array				8×8 Array				16×8 Array			
	Y	U	V	Sum	Y	U	V	Sum	Y	U	V	Sum	Y	U	V	Sum
Benchmark 1	1.46	1.65	1.69	1.52	1.53	1.77	1.80	1.60	1.56	1.83	1.86	1.65	1.60	1.87	1.92	1.68
Benchmark 2	1.44	1.65	1.68	1.50	1.50	1.76	1.79	1.58	1.54	1.82	1.86	1.63	1.57	1.86	1.90	1.66
Benchmark 3	1.53	1.66	1.67	1.57	1.62	1.76	1.78	1.66	1.67	1.83	1.84	1.72	1.71	1.86	1.88	1.76
Benchmark 4	1.22	1.60	1.62	1.33	1.24	1.69	1.72	1.37	1.26	1.75	1.79	1.39	1.26	1.78	1.81	1.40
Benchmark 5	1.52	1.60	1.63	1.55	1.59	1.68	1.71	1.62	1.62	1.74	1.77	1.66	1.62	1.76	1.80	1.67
Benchmark 6	1.59	1.65	1.63	1.60	1.67	1.74	1.72	1.69	1.71	1.81	1.79	1.74	1.73	1.84	1.81	1.76
Benchmark 7	1.56	1.61	1.58	1.57	1.64	1.70	1.66	1.65	1.68	1.76	1.71	1.70	1.70	1.79	1.74	1.72
Average	1.47	1.63	1.64	1.52	1.54	1.73	1.74	1.60	1.58	1.79	1.81	1.64	1.60	1.83	1.83	1.67

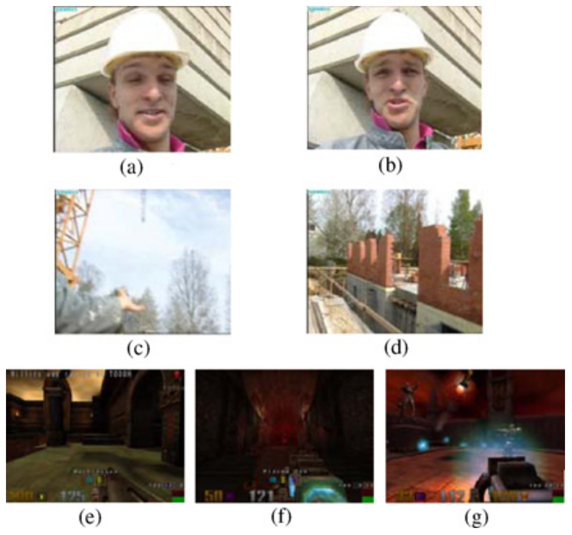


Fig. 7. Benchmark images. (a) Benchmark 1. (b) Benchmark 2. (c) Benchmark 3. (d) Benchmark 4. (e) Benchmark 5. (f) Benchmark 6. (g) Benchmark 7.

are presented. As the size of sub-window array increases, the compression ratio also increases. For example, the average compression ratios of 4×4, 8×4, 8×8, and 16×8 arrays are 1.52 1.59, 1.64, and 1.67, respectively. Since the spatial redundancy of the adjacent data at the sub-window boundaries cannot be exploited, the compression ratio of smaller sub-window array will decrease. The compression ratio for benchmark 4 was relatively lower than the compression ratio for other benchmark images. With benchmark 4, the sharp variance in the brightness component (Y) reduces spatial redundancies so that the compression ratio on Y value was relatively lower than the other benchmark images. However, on average, the proposed algorithm could guarantee more than 35% reduction in storage compared to the uncompressed image, which means that the proposed compression architecture can significantly reduce the storage requirement.

B. Standard Cell-Based Implementation

The proposed hardware architecture of compression and decompression was designed with Verilog hardware description language and synthesized with *Synopsys Design Vision* using a 0.15 μm complementary metal-oxide-semiconductor standard cell library. Table II includes the synthesis results in terms of area, critical path delay, and power consumption for various sub-window cases of 4×4, 8×4, 8×8, and 16×8 pixels. The hardware area is given in terms of gate equivalents (GEs), assuming that a two-input NAND gate is one GE. In all cases, the same number of pipeline stages and design constraints were applied. Since the proposed architecture exploits massive parallelism for compression and decompression operations, the area increases exponentially as the sub-window array size increases but the critical path delay slightly increases. For example, comparing 4×4 array and 8×8 array cases, the area increased by about three to four times but the critical path delay increased by about 10%. According to the critical path delay, in all cases, the encoder and decoder hardware can operate well at clock frequency of higher than 100 MHz. The dynamic power and static consumptions are calculated by using Synopsys Design Vision under 1.2 V operating voltage and the maximum operating clock frequency (such as, 170 MHz for 4×4 case, 150 MHz for 8×4 case, 130 MHz for 8×8 case, and 100 MHz clock frequency for 16×8 case). The power consumption increases exponentially according to the size of the sub-window. The throughputs of the proposed CODEC hardware are provided in the fifth column. In order to see the tradeoffs between different metrics, we have calculated the throughput per unit area and the throughput per unit power at the sixth and seventh columns of Table II, respectively. According to the results, the proposed scheme guarantees the most efficient throughput for the 4×4 sub-window case in terms of both the power consumption and the hardware area.

C. Performance Analysis for Different Bus Bandwidths

The actual performance improvement of the proposed hardware CODEC is limited by bus bandwidth. So we have performed experiments for different bus bandwidths such as 1-byte, 2-byte, and 4-byte bandwidths. Fig. 8 illustrates the experimental results for the performance improvement of the proposed CODEC. We have normalized the performance of the proposed CODEC to the performance of the system without any compression. According to the results, the proposed CODEC hardware could improve the system performance by about 66.47% on average. In addition, for the narrow bandwidth case, such as 1 byte bus, the contribution of the proposed CODEC was more significant.

TABLE II

STANDARD CELL-BASED IMPLEMENTATION

		Area (GEs)	Delay (ns)	Power (mW)		Throughput (Gpixels/s)	Throughput Per Unit Area (Kpixels/s)	Throughput Per Unit Power (Mpixels/s×mW)
				Dynamic	Static			
4 × 4 array	Encoder	15 998	4.23	4.48	0.004	2.72	41.91	308.42
	Decoder	16 456	5.78	4.33	0.005			
8 × 4 array	Encoder	54 560	4.59	14.84	0.013	5.02	22.83	16.92
	Decoder	55 381	6.34	11.67	0.014			
8 × 8 array	Encoder	202 095	4.95	55.51	0.049	8.64	9.97	7.65
	Decoder	231 011	7.38	57.38	0.068			
16 × 8 array	Encoder	639 195	5.26	176.53	0.157	13.31	4.96	3.64
	Decoder	701 205	9.55	179.36	0.207			

TABLE III

PERFORMANCE COMPARISON BETWEEN THE PROPOSED SCHEME AND THE PREVIOUS ENCODERS

		[15]	[16]	[17]	[18]	[20]	Proposed Encoder (For 4×4 Array)
Algorithm		X-Match ProRli	SPIHT	JPEG-LS	Arithmetic Coding	FELICS	DDPCM + Golomb Rice
Technology		Altera FPGA EP20K100	0.18 μm	0.18 μm	Xilinx Vertex 4	0.13 μm	0.15 μm
Operating frequency (MHz)		34.7	30	40	123	273	170
Throughput (MB/s)		275	13.82	9.9	123	546	2720
Compression ratio		1.51	2.00	NA	1.76	2.35	1.52
Parallelism		2	1	1	1	2	16
Area	Logic (GEs)	245 K	26 K	17.6	NA[a]	12.97 K	16.00 K
	Memory (Byte)	Not used	1.28 K	2.25 K	7.7 K	1.9 K	Not used
Power (mW)		NA	3.36	7.17	NA	33.93	4.48
Power efficiency (MB/mW)		NA	4.11	1.38	NA	16.0	607.14

[a]In [18], the hardware area is expressed as the number of the logic and FF slices and LUT, so that we could not provide exact GE.

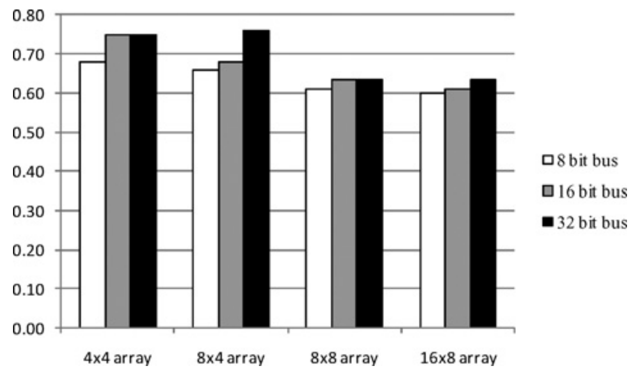


Fig. 8. Performance evaluation for different bus bandwidth configurations.

D. Comparison with the Previous Encoder Schemes

Table III provides the comparison between the proposed scheme and the various previous lossless compression hardware approaches [15]–[18], [20] in terms of the algorithm, the technology, the operating frequency, the throughput, the compression ratio, the parallelism, the hardware area, and the power consumption. Except for the compression ratio, the proposed scheme shows the best results in terms of hardware complexity, performance, and power consumption. The compression ratio of the proposed scheme was relatively lower than the compression ratios of [16], [18], and [20] since the proposed scheme adopts much simpler preprocessing stage based on DDPCM algorithm. The usability of the proposed scheme would be limited due to its lower compression ratio. The DDPCM algorithm, however, is better for parallel implementation so that the proposed encoder hardware could guarantee much higher throughput than the other schemes due to its extensive parallelism exploitation. Also, in terms of the power efficiency the proposed architecture demonstrates superior performance. Therefore, the proposed system would be more adequate for the high throughput applications.

TABLE IV

DECODER PERFORMANCE COMPARISON

	Technology	Operating Frequency	Logic Area (GEs)	Throughput (MB/s)
Proposed decoder	0.15 μm	170 MHz	16.46 K	2720
Lossless JPEG decoder [21]	0.18 μm	200 MHz	39 K	500

E. Comparison with the Previous Decoder Schemes

Table IV shows the comparison between the proposed decoder scheme and the lossless JPEG decoder. Since we could not find any decoder hardware implementation for the previous schemes [15]–[18], [20], the comparison with the lossless JPEG decoder implementation [21] is provided in Table IV. According to the results in Table IV, the proposed scheme outperforms the lossless JPEG decoder in terms of hardware complexity and performance.

V. Conclusion

In this paper, hardware architecture and implementation of a new lossless image compression and decompression scheme were presented. The proposed algorithm consists of DDPCM and Golomb-Rice coding. The input image frame is organized as $n \times m$ sub-window arrays, to which DDPCM is applied to produce one piece of seed and $(n \times m - 1)$ pieces of differential data. Then Golomb-Rice coding algorithm is applied to the differential data in order to produce losslessly compressed data. New hardware architecture for compression and decompression are proposed to contribute to memory bandwidth reduction by exploiting parallelism of operations. Experimental results on real benchmark image data show that the proposed architecture can guarantee adequate performance enhancement for real-time lossless encoding and decoding operations with moderate hardware area and power consumption.

References

[1] E. G. Hallnor and S. K. Reinhardt, "A unified compressed memory hierarchy," in *Proc. 11th Int. Symp. High-Performance Comput. Architect.*, 2005, pp. 201–212.

[2] S. Roy, R. Kumar, and M. Prvulovic, "Improving system performance with compressed memory," in *Proc. 15th Int. Parallel Distributed Process. Symp.*, 2001, pp. 630–636.

[3] J. Ziv and A. Lempel, "Compression of individual sequences via variable rate coding," *IEEE Trans. Inform. Theory*, vol. IT-24, no. 5, pp. 530–536, Sep. 1978.

[4] S. W. Golomb, "Run-length codings," *IEEE Trans. Inform. Theory*, vol. 12, no. 7, pp. 399–401, Jul. 1966.

[5] *JPEG LS Image Coding System*, document ISO/IEC JTC1/SC29/WG1 N399-WD14495, ISO/IEC, Jul. 1996.

[6] X. Wu and N. D. Memon, "Context-based adaptive lossless image coding," *IEEE Trans. Commun.*, vol. 45, no. 4, pp. 437–444, Apr. 1997.

[7] D. Nakar and S. Weiss, "Selective main memory compression by identifying program phase changes," in *Proc. 3rd Workshop Memory Performance Issues*, 2004, pp. 96–101.

[8] L. Yang, H. Lekatsas, R. Dick, and S. Chakradhar, "Operating system-based memory compression for embedded systems," U.S. Patent US2007005911A1, 2007.

[9] R. R. Osorio and J. D. Bruguera, "A combined memory compression and hierarchical motion estimation architecture for video encoding in embedded systems," in *Proc. 9th EUROMICRO Conf. Digital Syst. Des.: Architect., Methods, Tools*, 2006, pp. 269–274.

[10] T. Y. Lee, "A new frame-recompression algorithm and its hardware design for MPEG-2 video decodes," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 6, pp. 529–534, Jun. 2003.

[11] A. Wu and P. W. M. Tsang, "VLSI implementation of computation efficient color image compression scheme based on adaptive decimation for portable device application," *J. Signal Process. Syst.*, vol. 59, no. 3, pp. 267–279, 2010.

[12] K. Denecker, D. 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, 2002.

[13] L. Brooks and K. Fife, "Hardware efficient lossless image compression engine," in *Proc. IEEE Int. Conf. Acou., Speech, Signal Process.*, 2004, pp. 17–21.

[14] X. Chen, N. Canagarajah, and J. L. Nunez-Yanez, "lossless multi-mode interband image compression and its hardware architecture," in *Proc. Conf. Des. Architect. Signal Image Process.*, 2008, pp. 208–215.

[15] M. Milward, J. L. Nunez, and D. Mulvaney, "Design and implementation of a lossless parallel high-speed data compression system," *IEEE Trans. Parallel Distributed Syst.*, vol. 15, no. 6, pp. 481–490, Jun. 2004.

[16] 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.

[17] L. Xiaowen, 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.

[18] 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.

[19] S. Morein, "ATI Radeon: HyperZ technology," in *Hot3D Session Eurographics Workshop Graphics Hardware*, Aug. 2000.

[20] 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. Syst.*, vol. 18, no. 1, pp. 39–52, Jan. 2010.

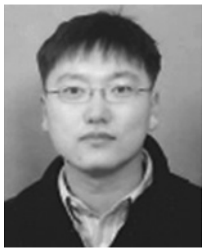
[21] CAST. *Image, Video, and Audio Interface IP Cores* [Online]. Availble: http://www.cast-inc.com/ip-cores/images

Hong-Sik Kim received the B.S., M.S., and Ph.D. degrees in electrical and electronic engineering from Yonsei University, Seoul, Korea, in 1997, 1999, and 2004, respectively.

In 2005, he was a Post-Doctoral Fellow with the Virginia Institute of Technology, Blacksburg. In 2006, he was a Senior Engineer with System LSI Group, Samsung Electronics Company, Gwangju, Korea. He is currently a Research Professor with Yonsei University. His current research interests include design for testability, test cost reduction, lossless data compression, and 3-D graphics rendering hardware design.

Joohong Lee received the B.S. degree in Internet engineering and the M.S. degree in computer science from Sejong University, Seoul, Korea, in 2008 and 2010, respectively.

He is currently an Assistant Engineer with the Research and Development Team, Digital Appliances, Samsumg Electronics Company, Ltd. His current research interests include 3-D rendering processor architecture, high performance computer architecture, real-time ray tracing, embedded systems, system-on-chip design, lossless image compression, and software engineering.



Hyunjin Kim received the B.S., M.S., and Ph.D. degrees in electrical engineering from Yonsei University, Seoul, Korea, in 1997, 1999, and 2010, respectively.

From 2002 to 2004, he was a Research Engineer with the Research and Development Center, Samsung Electro Mechanics, Suwon, Korea. His current research interests include parallel string matching, reconfigurable computing, interconnection networks, micro architecture, and compilers.

Sungho Kang received the B.S. degree from Seoul National University, Seoul, Korea, and the M.S. and Ph.D. degrees in electrical and computer engineering from the University of Texas at Austin, Austin.

He was a Post-Doctorial Fellow with the University of Texas at Austin, a Research Scientist with the Schlumberger Laboratory for Computer Science, Schlumberger Inc., Austin, and a Senior Staff Engineer with Semiconductor Systems Design Technology, Motorola Inc., Austin. Since 1994, he has been an Associate Professor with the Department of Electrical and Electronic Engineering, Yonsei University, Seoul. His current research interests include very large scale integration (VLSI) design, VLSI computer-aided design, and VLSI testing and design for testability.

Woo-Chan Park (M'11) received the B.S., M.S., and Ph.D. degrees in computer science from Yonsei University, Seoul, Korea, in 1993, 1995, and 2000, respectively.

From 2001 to 2003, he was a Research Professor with Yonsei University. He is currently an Associate Professor of computer engineering, Sejong University, Seoul. His current research interests include ray tracing processor architecture, 3-D rendering processor architecture, real-time rendering, advanced computer architecture, computer arithmetic, lossless image compression hardware, and application-specific integrated circuit design.