Hindawi Publishing Corporation
EURASIP Journal on Advances in Signal Processing
Volume 2009, Article ID 629285, 18 pages
doi:10.1155/2009/629285

Research Article

A New Frame Memory Compression Algorithm with DPCM and VLC in a 4×4 Block

Yongseok Jin, Yongje Lee, and Hyuk-Jae Lee

Department of Electrical Engineering and Computer Science, Inter-University Semiconductor Research Center, Seoul National University, Seoul 151-742, South Korea

Correspondence should be addressed to Hyuk-Jae Lee, hyuk_jae_lee@capp.snu.ac.kr

Received 11 January 2009; Revised 8 July 2009; Accepted 15 November 2009

Recommended by Gloria Menegaz

Frame memory compression (FMC) is a technique to reduce memory bandwidth by compressing the video data to be stored in the frame memory. This paper proposes a new FMC algorithm integrated into an H.264 encoder that compresses a 4×4 block by differential pulse code modulation (DPCM) followed by Golomb-Rice coding. For DPCM, eight scan orders are predefined and the best scan order is selected using the results of H.264 intra prediction. FMC can also be used for other systems that require a frame memory to store images in RGB color space. In the proposed FMC, RGB color space is transformed into another color space, such as YCbCr or G, R-G, B-G color space. The best scan order for DPCM is selected by comparing the efficiency of all scan orders. Experimental results show that the new FMC algorithm in an H.264 encoder achieves 1.34 dB better image quality than a previous MHT-based FMC for HD-size sequences. For systems using RGB color space, the transform to G, R-G, B-G color space makes most efficient compression. The average PSNR values of R, G, and B colors are 46.70 dB, 50.80 dB, and 44.90 dB, respectively, for 768×512-size images.

1. Introduction

Frame memory size and bandwidth requirements often limit the performance of a video processor designed for implementing a video compression standard such as MPEG-2, MPEG-4, and H.263 or H.264/AVC [1–4]. Frame memory compression (FMC) is a technique to reduce frame memory size by compressing the data to be stored in frame memory. Memory bandwidth requirement is also reduced by FMC because data access requirements are reduced. Figure 1 shows a video processor in which the encoder and decoder of an FMC algorithm are integrated inside the processor. A reference frame is, in general, stored in an off-chip memory. When the video processor stores the reference frame in the off-chip memory, the FMC encoder compresses the data. To access the reference frame from the off-chip memory, the video processor fetches compressed data from the off-chip memory and the FMC decoder decompresses and restores the original data.

Three properties, low latency, random accessibility, and low image quality degradation, are required for an efficient FMC algorithm. Video processor performance is significantly affected by the speed of the external memory, and FMC algorithm latency delays the access of external memory. Therefore, low latency in the FMC algorithm is required to minimize performance drop-off. Image compression algorithms like JPEG2000 are not suitable for FMC because they are too complex for low latency implementation, although their compression efficiency is high. The second property, random accessibility, is needed because frame memory can be accessed at an arbitrary address. Finally, FMC algorithms, in general, adopt lossy compression to maintain relatively high compression efficiency. Lossy compression typically degrades image quality, and therefore, additional image quality degradation may limit the practical use of FMC algorithms.

Extensive research efforts have been made to reduce the size and bandwidth requirements of frame memory [5–9]. A popular technique for FMC is a transform-based approach in which a frame is decomposed into small blocks that are transformed into a frequency domain by a simple transform, such as discrete cosine transform (DCT) [6], the Hadamard

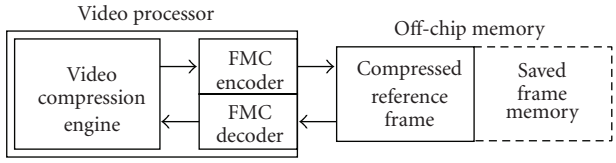


Figure 1: Video processor with an integrated FMC encoder and decoder.

Transform or its variations [7]. The frequency domain coefficients are then compressed by quantization followed by variable length encoding, such as Golomb-Rice coding. A transform-based approach achieves an efficient compression when the block size for a transform is large. For example, the block size in the algorithm in [6] is 16×16. As the transform block size increases, the hardware complexity of a transform as well as the compression latency also increases. Another approach is a spatial domain FMC that requires a relatively small amount of computation [8, 9]. The FMC in [8] is a variable-ratio compression which achieves an average of 40% memory reduction. The FMC in [9] is a DPCM-based approach which achieves 50%-constant compression with pattern matching and selective quantization. This FMC is implemented in software, but it is not verified in hardware. Due to the sequential nature of the pattern decision, a large latency is expected if this algorithm is implemented in hardware.

Frame memory compression techniques for specific applications have been proposed [10, 11]. For LCD, it is often the case that data are over-driven to compensate for the slow response time of an LCD panel. To detect the difference between the current and the previous frames, the previous frame is stored in a frame memory. FMC is used to reduce the frame memory space and aggressive techniques are employed at the sacrifice of the image quality because a reconstructed image is used only to detect the difference and slight image quality degradation is tolerable. Another example use of FMC is texture compression in graphics rasterization [12, 13]. In general, a slight image quality degradation is allowed in texture image rasterization. Therefore, texture compression often uses the dictionary-based approach that aims an aggressive compression ratio at the sacrifice of image quality. Both algorithms for the LCD over-drive and texture compression allows image quality degradation, and consequently, they may not be suitable for image compression integrated in an H.264 compression chip.

This paper proposes a new FMC algorithm that compresses frame data efficiently by using intraprediction information provided by an H.264/AVC encoder. The proposed algorithm divides an image frame into 4×4 blocks and compresses each block independently by a 50% constant compression ratio. For each 4×4 block, DPCM is performed along a predefined scan order. To achieve high compression efficiency, eight DPCM scan orders are predefined on an analog of the eight 4×4 intraprediction modes (excluding the DC prediction mode) for an H.264/AVC encoder. To select the best scan order, the FMC algorithm uses the information provided by H.264/AVC intraprediction because those predictions evaluate the correlations among neighboring pixels and provide information about the direction between highly correlated pixels. Once H.264 intraprediction mode is selected, the scan order is selected from the intraprediction mode, and DPCM is performed. The DPCM results are further compressed by Golomb-Rice coding. If the compression ratio does not reach 50%, the 4×4 block pixel data are quantized by 1-bit right shifting, repeat DPCM, and entropy coding.

Frame memory is used not only in a video compression processor but also in an LCD driver [10, 11] or a 2D/3D graphics processing chip [14]. A 50% compression of a reference frame can also be used for these chips to save the frame memory bandwidth and space. However, these chips do not include an intraprediction module, so that the best scan mode must be decided by the FMC algorithm itself. Furthermore, video compression standards usually employ the YCbCr 4:2:0 color format in the frame memory, but other chips often employ the RGB 4:4:4 color format. Therefore, the FMC algorithm for a video processor is not directly applicable for an LCD driver or a 2D/3D graphics chip. The second part of this paper modifies the FMC algorithm proposed for an H.264/AVC encoder to be used for the frame memory compression for these chips. This modification includes the transform of the RGB color space to another color space efficient for compression. Other modifications are the inclusion of the step to select the best scan mode and the combined packetization of three color components.

The paper is organized as follows. In Section 2, the proposed FMC algorithm is described. Then the FMC algorithm for RGB color space is presented in Section 3. Section 4 explains the hardware implementation of the proposed FMC algorithm. Section 5 compares image quality degradation of the proposed FMC algorithm with a previous algorithm. Conclusions are presented in Section 6.

2. FMC with H.264/AVC Video Compression

This section proposes an FMC algorithm that can be used to reduce frame memory for an H.264/AVC encoder.

2.1. Basic Idea. The proposed FMC algorithm was designed to compress a 4×4 block by 50% and generate a 64-bit packet. To achieve this aim, the proposed algorithm employs DPCM, which calculates differences between successively scanned data and uses those differences to represent the data. For efficient DPCM compression, the differences between successive data should be small so that the data can be represented by a small number of bits. The magnitude of the difference depends on the image contents as well as the scan order. For example, if a 4×4 block includes vertical stripes, a DPCM scan along the vertical direction results in a smaller difference than that along the horizontal direction. Therefore, it is important to select a scan order that minimizes the differences between data. To this end, the proposed FMC algorithm uses eight scan modes (see Figure 2). The eight modes are based on an analog of the

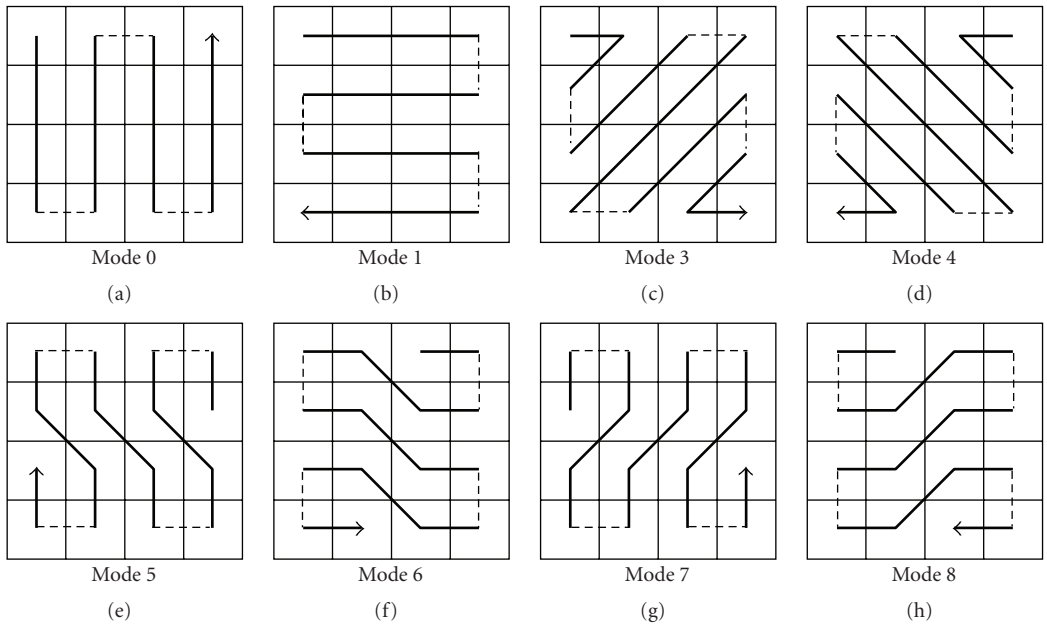


Figure 2: Eight scan modes for DPCM. Arrows indicate the scan order.

4×4 intraprediction modes for an H.264 4×4 intraprediction is performed in nine different scan modes, but Mode 2 (DC mode) is excluded from Figure 2 because Mode 2 did not provide information useful for scan order selection. An advantage resulting from Mode 2 exclusion is that only three bits are needed to represent the remaining eight modes. The eight modes in Figure 2 cover various image types for DPCM scans. For example, Mode 0 is suitable for an image with vertical stripes while Mode 1 is suitable for horizontal stripes, and an image with diagonal stripes may be best suited to one of the other modes.

2.2. Algorithm. The flowchart of the proposed algorithm is shown in Figure 3. A single 4×4 block is the input of the algorithm and the output is a 64-bit packet. As this FMC is designed to reduce frame memory for H.264/AVC compression, the H.264/AVC compression operations, including intraprediction, are performed with FMC. To select two scan modes from among the 8 modes shown in Figure 2, the 4×4 intraprediction result is assessed by the algorithm. The first mode is the same as that determined by intraprediction, excluding the DC mode. The horizontal and vertical modes, in general, produce efficient FMC results. Thus, one of these two modes is always selected as the second mode. For example, if modes 1, 3, 5, or 7 are selected first by H.264 intraprediction, then mode 0 is selected as the second mode, while if modes 0, 4, 6, or 8 are selected first, mode 1 is selected second. If the DC mode is selected by intraprediction, modes 0 and 1 are selected as the first and second modes, respectively.

The two selected scan orders are provided to the next step, which performs DPCM operations along the selected scan orders. The input 4×4 pixels are quantized with the quantization parameter (QP). For quantization, the input

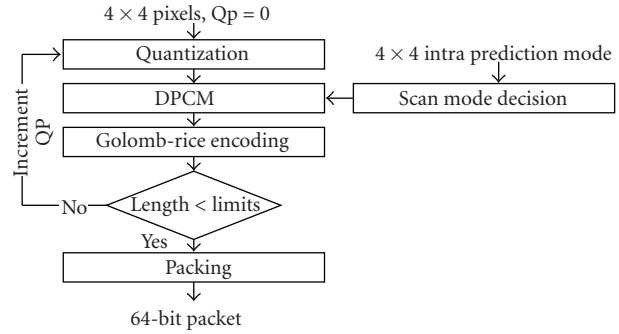


Figure 3: Flowchart of the proposed FMC algorithm.

data are right shifted by QP times. For example, if QP = 2, then the input data are shifted to the right twice. During this shift operation, the left most bit is replaced by 0. The quantization parameter is initially set to 0 and incremented later, if required. The DPCM results are compressed by Golomb-Rice coding and the required number of bits for a single packet is calculated. If this number is less than the limit (i.e., 64 bits), then the result of Golomb-Rice coding is packed into a 64-bit packet. Since two scan modes are selected and Golomb-Rice coding is performed for both modes, the one requiring the smaller number of bits is selected. If the Golomb-Rice coding result requires a larger number of bits than the limit, the QP is incremented by 1 and quantization, DPCM, and Golomb-Rice coding are performed a second time. The Golomb-Rice coding and packetizing steps are explained next.

In order to match the desired bit-rate, the proposed algorithm prequantizes the input pixels and then applies DPCM. However, in lossy DPCM usually, there is a feedback loop, and quantization is applied during (and not before)

the prediction. For a uniform quantizer, if the quantization step size Δ ($= 2^{QP}$) is sufficiently small, it is reasonable to assume that the quantization error is uniformly distributed in interval $[-\Delta/2, \Delta/2]$. Note that the QP value used in the proposed FMC is small (see Figure 20). Therefore, the quantization error is likely to be distributed uniformly. This implies that the quantization errors in both the feedback loop and prequantization approaches have similar distribution of quantization error and consequently the coding errors of the two DPCMs do not differ significantly.

On the other hand, the hardware complexity of the prequantization is just about a half of that required by the conventional feedback-loop approach because the conventional approach requires two adders in addition to the dequantizer for an encoder whereas the prequantization requires just a single adder. In summary, the prequantization DPCM is adopted in this paper because computational complexity is about a half of the feedback-loop DPCM although the prequantization DPCM increases slightly the coding error.

2.3. Golomb-Rice Coding. The Golomb-Rice coding [15, 16] accepts only a nonnegative number as input. However, a DPCM result can be negative. Therefore, for Golomb-Rice coding input, a negative DPCM result is converted into a nonnegative number by

$$\text{source} = \begin{pmatrix} 2|\text{diff}|, & \text{diff} > 0 \\ 2|\text{diff}| - 1, & \text{otherwise} \end{pmatrix}, \tag{1}$$

where diff represents a DPCM result and source represents the input to the Golomb-Rice coding.

For Golomb-Rice coding, source is divided by 2^k and the division quotient is represented in unary notation that represents a nonnegative integer, n, with n zeros, followed by a single one. The quotient and remainder in conventional k bit binary notation are then concatenated to form a Golomb-Rice codeword. The length of a Golomb-Rice codeword is

$$\text{length}_{\text{GR}} = k + 1 + \left\lfloor \frac{\text{source}}{2^k} \right\rfloor. \tag{2}$$

For a small *source*, a smaller k results in a smaller Golomb-Rice codeword length. As *source* increases, a larger k may produce a smaller *code* length. Thus, the choice of k depends on the value of *source*. For example, if $k = 0$, the length increase is too large for a large *source*. On the other hand, if $k > 2$, the length is too large for a small *source*, and a k greater than 2 is unacceptable for 50% compression because the minimum number of bits assigned to each pixel is 4. Therefore, the chosen value of k is either 1 or 2. For the eight modes shown in Figure 2, a difference along the dotted line is encoded with $k = 2$ while a difference along the solid line is encoded with $k = 1$. DPCM results along dotted lines may be large because the dotted lines cross edges. In this case, a large k may lead to a smaller number of bits to represent this large difference. By assigning the large k ($k = 2$) to the dotted line and the small k ($k = 1$) to the rest, the total number of bits generated by Golomb-Rice coding for all 16 pixels are, in general, reduced.

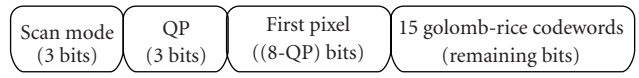


Figure 4: The format of a Golomb-Rice codeword packet.

2.4. Packetization. The Golomb-Rice codewords are packetized as a 64-bit packet. Figure 4 shows the packet format. The 8 scan modes are coded with 3 bits and stored in the leftmost position and the 3-bit QP is stored next. The first pixel requires (8−QP) bits stored next to the QP and the remaining bits store the Golomb-Rice codewords for the remaining 15 pixels.

Video compression standards, such as H.264/AVC, employ the 4 : 2 : 0 format in the YCbCr color space to represent an image. In general, the three color components are stored in separate spaces in frame memory. One reason for separate memory allocation is because the three components are not always accessed at the same time. For example, motion estimation in the H.264/AVC requires only the Y color component. Another reason is the difference in the amount of data in the Y and Cb (or Cr) color components. In the 4 : 2 : 0 format, Y color data are assigned to each pixel, while single Cb (or Cr) data are assigned to every 2×2 pixels [17]. Thus, the amount of data for the Cb (or Cr) color component is one fourth of that for the Y color component. As a result, the Y color component requires four times larger memory space than the Cb (or Cr) color components. As the three colors are stored separately and accessed independently, they are also compressed independently. Thus, the FMC algorithm in Figure 3 is performed independently three times for Y, Cb, and Cr colors.

2.5. Example. Consider a 4×4 block as shown in Figure 5(a), and assume that the intraprediction mode resulting from H.264/AVC is 1. Thus, the first scan order selected is mode 1 and the second scan order is mode 0. QP = 0 requires 91 bits for mode 1 and 212 bits for mode 0. Thus, QP = 0 is not acceptable for both modes. For QP = 1, mode 1 scans data as denoted by the arrow shown in Figure 5(b). The pixel values quantized with QP = 1 (i.e., shifted once to the right) are also shown in Figure 5(b). The scanned data along the dotted arrow are 121, 120, 118, 118, 109, 108, 104, 103, 110, 110, 107, 105, 110, 110, 108, and 107. Thus, the DPCM results are 121, −1, −2, 0, −9, −1, −4, −1, 7, 0, −3, −2, 5, 0, −2, and −1 in the scanned order shown in Figure 5(c). Table 1 shows the Golomb-Rice codewords for the DPCM results. For example, the fourth DPCM result, DPCM [4], is −9. From (1), the *source* for this value is 17. From $k = 2$, the quotient and remainder are 4 and 1, respectively. The quotient in unary notation is 00001 and the remainder in k-bit binary notation is 01. The final codeword is the concatenation of the quotient and remainder, that is, 0000101. Table 1 shows the codewords of all DPCM results. Fifty bits were required for all the words. In addition to these bits, 6 bits are necessary to store the mode and QP and 7 bits are required for the first datum. As a result, the packet in mode 1 with QP = 1 requires 63 bits. On the other hand, mode 0 requires 124 bits when QP =

242	241	237	236
206	209	216	219
221	221	214	211
215	216	220	221

(a)

121	120	118	118
103	104	108	109
110	110	107	105
107	108	110	110

(b)

121	−1	−2	0
−1	−4	−1	−9
7	0	−3	−2
−1	−2	0	5

(c)

Figure 5: An example of 4 × 4 block: (a) Input 4 × 4 pixel values, (b) 4 × 4 pixel values after quantization by QP = 1, and (c) DPCM results.

1. As mode 1 requires fewer bits than mode 0, it is chosen as the best scan mode. Figure 6 shows the FMC result. In Figure 6, the first three bits (001) and the next three bits (001) represent mode 1 and QP, respectively. The next seven bits represent the first datum quantized by QP = 1. The remaining bits are the Golomb-Rice codewords of the next 15 DPCM results.

Table 1: The Golomb-Rice Codewords of the 4 × 4 Block Shown in Figure 5.

Element	Value	Source	k value	Codeword
Diff [1]	−1	1	1	11
Diff [2]	−2	3	1	011
Diff [3]	0	0	1	10
Diff [4]	−9	17	2	0000101
Diff [5]	−1	1	1	11
Diff [6]	−4	7	1	00011
Diff [7]	−1	1	1	11
Diff [8]	7	14	2	000110
Diff [9]	0	0	1	10
Diff [10]	−3	5	1	0011
Diff [11]	−2	3	1	011
Diff [12]	5	10	2	00110
Diff [13]	0	0	1	10
Diff [14]	−2	3	1	011
Diff [15]	−1	1	1	11

3. FMC of Frame Memory in RGB Color Space

There exist a number of applications other than H.264/AVC video compression that store video data in frame memory. For instance, an LCD display driver needs frame memory to store its output video [10, 11]. For another example, a 2D or 3D graphics processor also requires frame memory [14]. The FMC algorithm proposed in Section 2 is not directly applicable to these other applications because they cannot use the information obtained by H.264/AVC intraprediction. Moreover, these other applications, in general, store video in the RGB color space while the algorithm in Section 2 is developed for video in the YCbCr color space. This section extends the algorithm proposed in the previous section and proposes the FMC algorithm suitable for video in the RGB color space.

3.1. FMC in the 4 : 4 : 4 Format and Combined Packetization. In an LCD display driver or 2D/3D graphic processor, an image is stored in the RGB 4 : 4 : 4 format in which each pixel is represented by R, G, and B color components. Unlike the YCbCr colors in the 4 : 2 : 0 format, RGB color components in the 4 : 4 : 4 format are, in general, accessed at the same time [10–14]. Thus, an effective memory access is possible by storing three color components for one pixel in consecutive memory addresses. As three color components are stored consecutively and accessed at the same time, these components can also be compressed at the same time to be packetized into a single combined packet. The combined packet allows more efficient compression than the separate packet because the scan mode and QP can be shared by these three colors. The format of the combined packet is shown in Figure 7. The 4 × 4 block in the 4 : 4 : 4 format consists of 16 pixels of three colors, so that total 384 bits are required to store a single 4 × 4 block. By 50% compression, the compressed packet size is less than or equal to 192 bits.

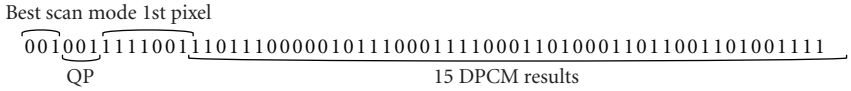


Figure 6: The packetized result of the example shown in Figure 5 and Table 1.

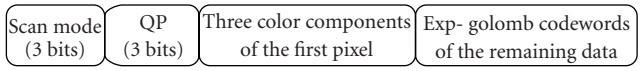


Figure 7: The format of a combined Exp-Golomb codeword packet.

The scan mode and QP are stored in the leftmost 6 bits. Note that only one scan mode and QP are required for three colors. The first pixel data of three colors are stored next followed by remaining pixels. For the compression of the remaining data, it is observed experimentally that the Exp-Golomb coding is more efficient than the Golomb-Rice coding (see details in the next subsection).

3.2. Exp-Golomb Coding. Golomb-Rice codewords used in Section 2 are efficient when the value of source is not large. Recall that the length of a Golomb-Rice codeword increases in proportion to its value. On the other hand, another entropy coding, the length of an Exp-Golomb codeword, is

$$\text{length}_{\text{EG}} = k + 1 + 2\left\lfloor \log_2\left(\frac{\text{source}}{2^k + 1}\right)\right\rfloor. \tag{3}$$

The details about Exp-Golomb coding are presented in [17]. The length of an Exp-Golomb codeword increases in proportion to log(source). Therefore, Exp-Golomb coding generates a shorter codeword than Golomb-Rice coding when the value of *source* is large. It is observed by experiments that Exp-Golomb coding is more efficient than Golomb-Rice coding for combined packetization (see Figure 20). Similar to Golomb-Rice coding, a large k generates a short codeword when the value of *source* is large. On the other hand, a small k is preferable for a small *source*. Thus, the value of k is chosen in the same manner as for Golomb-Rice coding in Section 2; that is, 2 is chosen for the *source* (DPCM results) represented by the dotted line's shown in Figure 2 while 1 is chosen for the rest DPCM results.

3.3. Scan Mode Decision. Among the eight possible scan modes shown in Figure 2, the mode that generates the smallest packet size must be selected. In Section 2, two candidate scan modes are determined from the intraprediction mode in H.264/AVC. Then, the results of the two modes are compared and the best mode is selected between the two candidate modes by comparing their packet sizes. For the FMC in the RGB color space, the information from H.264/AVC is not available. Thus, all eight scan modes are compared and the best mode is selected among them. To this end, the parameter QP is set to 0 and the lengths of fifteen *sources* (DPCM results) are evaluated and then added to obtain the packet size. The packet size must be evaluated for the whole eight scan modes, so that a large amount of computation is required for the selection of the best scan mode.

The computation for best mode selection is reduced by taking advantage of the fact that there exist many DPCM results that are shared by multiple scan modes. For instance, in Figure 2, the first DPCM results of mode 1 and 2 are identical (i.e., they are the difference between the leftmost top pixel and its next pixel to the right). For the eight scan modes with fifteen DPCM results each, the code lengths of 120 DPCM results need to be evaluated. Among these 120 DPCM results, 57 results are shared by more than one scan modes. Thus, 63 DPCM results in total are necessary for the evaluation of the code lengths for eight scan modes.

To obtain the accurate packet size, the evaluation of the lengths of *sources* must be repeated until the packet size is less than 192. However, the repeated evaluations require too much computation. Therefore, only the evaluation with QP = 0 is used to choose the best scan mode. Experiments show that the order of the packet size chosen with QP = 0 is almost the same as the order with the best QP.

3.4. Color Transform. With experiments, it is observed that the compression efficiency is improved when the RGB color space is first transformed into the YCbCr color space and then the FMC is applied to the image in the YCbCr space (see Section 5 for details on the experimental results). Note that the transformed image in this case is in the 4 : 4 : 4 format instead of the 4 : 2 : 0 format as in Section 2. Thus, all three color components are available for each pixel, and they are packetized in the same format shown in Figure 7. One of the reasons why the YCbCr color space is more efficient than the RGB color space is because the data in the Cb and Cr colors vary more slowly than those in the R and B colors, respectively. As a result, the DPCM results in the Cb and Cr colors are smaller than those in the R and B colors, respectively. The combined packetization of Y, Cb, and Cr colors allow increased bits assigned to the Y color thanks to the reduced bits assigned for Cb and Cr colors. The increased bits assigned to the Y color decreases the error in the Y color, and consequently, the error in the Cb and Cr is also reduced because Y is used to derive Cb and Cr. Moreover, Y color affects the subjective quality greater than Cb or Cr color. As a result, image quality is, in general, improved by the color space transform. The transform coefficients between the RGB color space and the YCbCr color space are given by ITU-R recommendation BT.601 [18].

One source of quality degradation with the YCbCr color space transform is the round-off error in the transform. For instance, consider the pixel with its value $\{R, G, B\} = \{128, 128, 128\}$. Suppose that this pixel is transformed into the YCbCr color space. This pixel is transformed into

$\{Y, Cb, Cr\} = \{142.592, -8.46, -10.695\}$. By rounding off these values to integers to store in memory, this pixel becomes $\{143, -8, -11\}$. Suppose that this pixel is transformed back to the original RGB color space. $\{R, G, B\} = \{131.784, 132.284, 124.058\}$ is obtained. By rounding off these values again to integers, the pixel becomes $\{R, G, B\} = \{132, 132, 124\}$ which is significantly different from the original value $\{128, 128, 128\}$. This example shows that a significant error is caused by the transformation.

For the FMC in the RGB color space, it is not mandatory to use the YCbCr color space. In the JPEG2000 standard for image compression, a modified YCbCr color space is used for the removal of the transform error [19]. The FMC algorithm can be applied to the JPEG2000 YCbCr color space just in the same way as the original YCbCr color space. The transformation error is reduced because the transformation is reversible. In JPEG2000, 9 bits are used to store each of Cb and Cr components so that no error is created by the transform. Thus, the image quality with the JPEG YCbCr space is better than that with the original YCbCr space.

A number of demosaicing algorithms [20–23] as well as digital display interface such as low-voltage differential signaling (LVDS) adopt the color space consisting of G, R-G, B-G instead of the YCbCr color space. One of the main advantages is a simple transformation from/to the RGB color space because only subtraction operations are needed for the transformation. Another advantage comes from the fact that the error in R-G or B-G does not affect the G color space so that the error in the G color space is less than that in R-G or B-G color space. This property can reduce the quality degradation by color transform because human eyes are more sensitive to the G color than the R or B color. For simplicity, Dr and Db are used hereafter to denote R-G and B-G spaces, respectively. Instead of the original YCbCr color space, the JPEG2000 color space or GDbDr color space can also be used for FMC. Section 5 presents experimental comparisons among these color spaces.

In the packet shown in Figure 7, the first three pixels are stored from the 7th bit. In the RGB color space, $(8 - QP)$ bits are necessary to store one color component of the first pixel. Thus, to store three colors, $3 \cdot (8 - QP)$ bits are required. For the original YCbCr color space, 8 bits are required to represent Y, Cb, Cr color components. Thus, $3 \cdot (8 - QP)$ bits are necessary to store the first pixel in the packet shown in Figure 7. In the JPEG2000 YCbCr color space, $(8 - QP)$ bits are needed for the first pixel. On the other hand, $(8 - QP + 1)$ bits are needed for the Cb color of the first pixel because they include the sign bit. Similarly, Cr also requires $(8 - QP + 1)$ bits. Therefore, $(8 - QP) + 2 \cdot (8 - QP)$ bits are required to store Y, Cb, and Cr colors of the first pixel. For the GDbDr color space, G requires 8 bits while Dr or Db requires 9 bits. Thus, $(8 - QP) + 2 \cdot (8 - QP)$ bits are also necessary for the first pixel.

3.5. Algorithm. Figure 8 shows the flow chart of the FMC algorithm discussed in this section. This algorithm processes three color components in the YCbCr or GDbDr space simultaneously, so that the number of bits for the input 4×4 pixels is 384 and that for the output packet is reduced to 192 by 50% compression. When compared with the algorithm shown in Figure 3, the first step is added to the transform from the RGB color space to YCbCr (or GDbDr) color space. The scan mode decision step is different from that in Figure 3 because the best scan mode is decided by comparing all 8 scan modes. The Golomb-Rice coding is replaced by Exp-Golomb Coding, and Quantization, DPCM steps are the same as those in Figure 3.

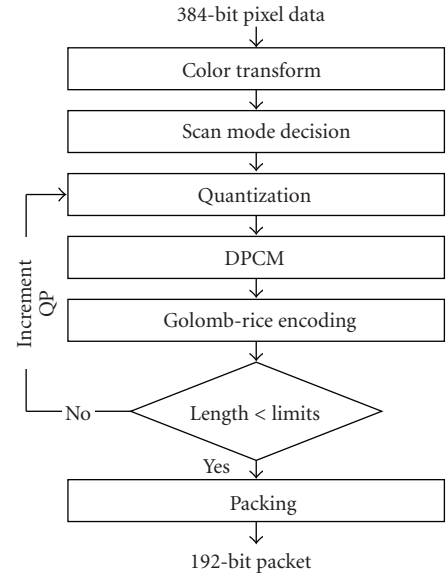


Figure 8: Flowchart of the FMC for the RGB color space.

3.6. FMC by 75%. The data in the RGB color space can be compressed by 75% with the combination of color transform, subsampling, and the FMC proposed in Section 2. Subsampling from the 4:4:4 format to the 4:2:0 format achieves 50% compression. Recall that the FMC algorithm in Section 2 is applied to the subsampled data in the 4:2:0 format to achieve another 50% compression. Color transform to another color space like YCbCr is necessary because the subsampling of the Cb and Cr colors does not severely deteriorate the visual quality of an image because human eyes are more sensitive to the Y color than Cb and Cr colors. The original YCbCr color space may create a round-off error. To reduce this error, the JPEG2000 YCbCr color space or the GDbDr color space is also considered as the target color space. The effectiveness of three color spaces are evaluated by experiments as presented in Section 5.

4. Hardware Implementation

This section explains the hardware implementation of the proposed FMC algorithm in Section 2.

4.1. Encoder. The pipeline architecture of the FMC encoder is shown in Figure 9(a). To increase the throughput, the encoder operation is pipelined in four stages. In pipeline Stages 1 and 2, quantization, DPCM, and Golomb-Rice

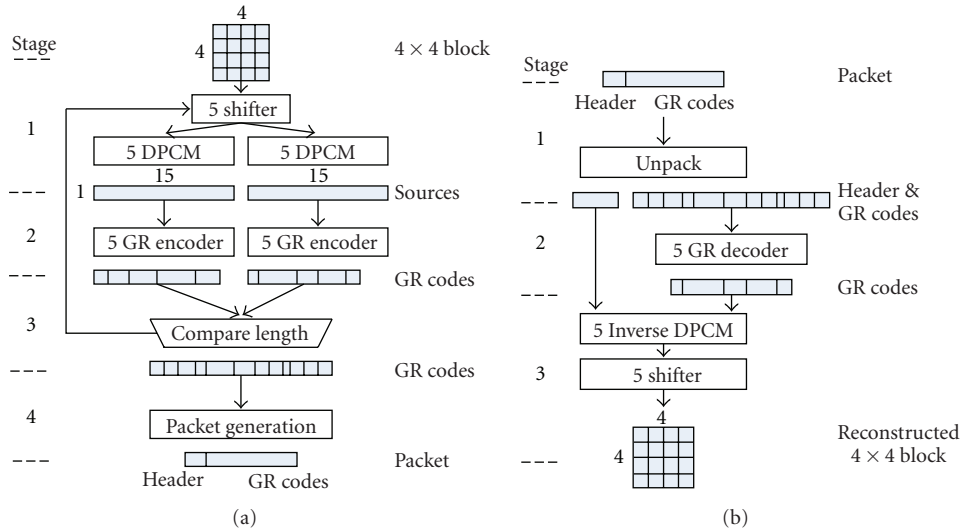


Figure 9: Block diagram of the FMC encoder and decoder.

encoding are performed for codeword generation. Initially, QP is chosen 0 and the codeword is generated. In Stage 3, if the codeword size is less than or equals to 64 bits, the pipeline moves to the next stage. Otherwise, the QP is incremented and Stages 1, 2 and 3 are repeated. The codeword generation and QP increment are repeated until the codeword size is less than or equal to 64. Five cycles are needed to complete a single iteration of Stages 1, 2, and 3. The total execution time is 5(QP + 1) + 1 cycles because Stages 1, 2, and 3 take 5 cycles. If QP is 0, a new 4×4 block is processed at every 5 cycle. The gate count of the FMC encoder is 19.8 K.

4.2. Decoder. In general, the execution time of an FMC encoder is not critical because the compressed data are not used immediately but they are stored in a frame memory for use in some time later. However, the execution time of an FMC decoder is critical because its result is immediately used. Therefore, an optimized hardware design is needed to minimize the execution time of a decoder. Figure 9(b) shows the proposed pipelined architecture of an FMC decoder. In Stage 1, a 64-bit packet is read from the frame memory. The proposed FMC decoder needs 5 cycles to complete one 4×4 block and processes a new 4×4 block for every 3 cycles. Assuming that the memory bandwidth is allowed to transmit 32 bits per a cycle, the throughput of the FMC decoder is larger than that of the frame memory. Therefore, the memory bandwidth is the bottleneck of the overall throughput and the addition of the FMC decoder does not decrease the data access throughput. The gate count of the FMC decoder is 11.3 K.

4.3. Complexity Comparison. The complexity of the proposed algorithm is compared with the previous work based on Modified Hadamard Transform [7]. Table 2 shows the numbers of additions (or subtractions) and shifts required for both encoding and decoding operations of FMC. For the proposed FMC, N represents the number of iterations. The Golomb-Rice coding is not considered for this comparison because it is common for both FMCs. Experiments show that the average number of N is equal to 2.43. If this number is used for the equation in Table 2, the proposed FMC encoding requires 72.9 additions (or subtractions) and 22.88 shifts for each 4×4 block (16 pixels). The MHT-based FMC requires 27 additions (or subtractions) and 68 shifts for each 1×8 block (8 pixels). To process 16 pixels (two 1×8 blocks), the MHT-based FMC requires 54 additions (or subtractions) 136 shifts. Thus, the proposed FMC requires a comparable amount of computation. For decoding, the proposed FMC also requires less computation than the MHT-based FMC. The complexity reduction is possible by the proposed FMC because it makes use of the information given by an H.264 encoder.

Table 2: Complexity comparison (FMC encoding/decoding).

	Block size	Addition (or Subtraction)	Shift
Proposed FMC in Section 2	4×4	$30N/15$	$16 \cdot (N-1)/16$
MHT-based FMC	1×8	27/27	68/36

4.4. Integration into an H.264 Encoder Chip. The proposed FMC encoder and decoder are integrated with H.264 encoder [24]. Figure 10 shows a block diagram of the encoder. The hardware accelerators for motion estimation, deblocking filter, intraprediction, and variable length coder are implemented in hardware and the remaining part of computation is processed by the ARM7TDMI processor. VIM (Video Input Module) accepts image data from an image sensor and SPI interface outputs the encoded stream. Memory Controller is designed for efficient data communication with an external SRAM. Two AMBA AHB buses are used for

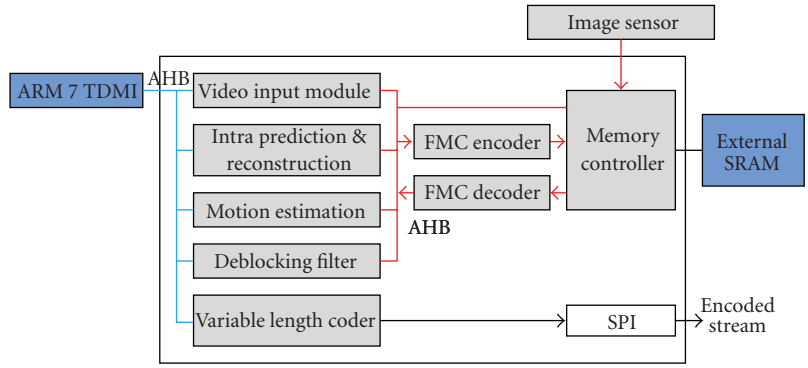


Figure 10: Block diagram of the H.264/AVC encoder integrated with the FMC encoder and decoder.

the communication between modules. One AHB bus is mainly used for the control of the hardware modules by ARM7TDMI processor and the other AHB bus is mainly used for data communication between hardware modules and external memory. The FMC encoder and decoder are placed between the AHB bus and the memory controller. Figure 11 shows the layout and the chip photograph of the H.264/AVC encoder shown in Figure 10. The die area of the H.264/AVC encoder is 5 mm × 5 mm using the Dongbu 1P6M 0.13 um CMOS technology.

5. Experimental Results

5.1. FMC Algorithm in an H.264 Encoder. Software implementation of the proposed algorithm in Section 2 is integrated with H.264/AVC JM reference software version 13.2 [25] so that the reference frame is compressed by the proposed FMC. Previous work, based on Modified Hadamard Transform (MHT) proposed in [7], is also implemented and the results are compared. The two algorithms are evaluated with three CIF-size (352 × 288) video sequences: Foreman, Mobile Calendar, and Table Tennis; as well as with two HD-size (1920 × 1080) sequences: Blue sky and Pedestrian area. For every sequence, 100 frames are used and the encoding speed is 30 frames per second. For experiments, the test sequence is encoded as a Baseline profile stream with the intraframe interval of 10, 3 reference frames for motion estimation, deblocking filter turned on, rate-distortion optimization also turned on, and four QP values, 20, 24, 28, and 32.

The rate distortion performances for Y component are shown in Figure 12. The average PSNR degradations, by the FMC algorithms, are measured and shown in Table 3. These values are obtained by Bjontegaard's method presented in [26]. For the three CIF-size sequences, the average PSNR degradations are 0.77 dB and 2.39 dB by the proposed and MHT-based FMCs, respectively. For the two HD-size sequences, the average PSNR degradations are 0.38 dB and 1.72 dB by the proposed and MHT-based FMCs, respectively. For both CIF-size and HD-size video sequences, the proposed FMC makes a significant improvement over the previous MHT-based FMC. The results also show that quality degradation of HD-size video is less than that of CIF-size video. This is because spatial correlation of a 4 × 4 block generally increases as image size increases, so that compression with minimal loss of information is possible.

Table 3: Average BD-PSNR(dB) degradation compared with the original H.264.

Sequence	8-mode FMC	Proposed FMC	1-mode FMC	MHT-based FMC
Foreman	0.45	0.69	1.08	2.72
Mobile and calendar	0.76	1.00	1.32	2.41
Table tennis	0.49	0.61	0.93	2.05
CIF average	0.57	0.77	1.11	2.39
Blue sky	0.47	0.65	1.05	2.05
Pedestrian area	0.06	0.11	0.24	1.38
HD average	0.27	0.38	0.64	1.72

The simulation also evaluates the efficiency of the scan mode decision step in Figure 3. The mode selected by the scan mode decision step may not always be the scan mode that maximizes the compression efficiency. Thus, all 8 modes are used by the FMC algorithm and the best scan mode is then selected. In Figure 12, "8-mode FMC" presents the results when the best scan mode is selected from among all 8 modes. Another simulation uses the scan mode selected by the H.264 intraprediction, "1-mode FMC" (Figure 12). The computational complexity of 1-mode is half of that using the proposed algorithm because only one mode is evaluated while the proposed algorithm evaluates two modes. The 1-mode quality degradation is larger than that using the proposed algorithm. Comparing the average of the three CIF-size sequences, the 8-mode algorithm was 0.20 dB better than the proposed algorithm while the 1-mode algorithm is 0.34 dB worse than the

Table 4: Ratio of the difference along the dotted line scan over that along the solid line scan.

	Foreman	mobile	Table tennis	Blue sky	Pedestrian area	Average
Dotted/solid line	177.6%	140.2%	151.5%	180.1%	312.7%	153.4%

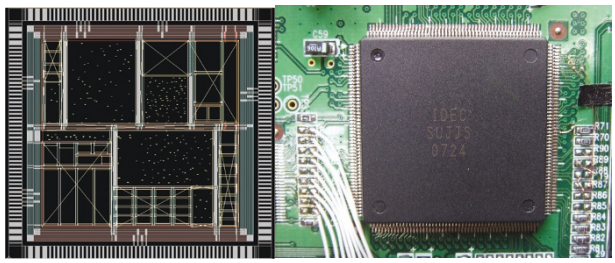


Figure 11: Chip layout and photograph.

proposed algorithm. For the two HD-size sequences, the 8-mode and 1-mode algorithms average 0.11 dB better and 0.26 dB worse, respectively, than the proposed algorithm. These results show that the proposed algorithm produces a reasonable trade-off between complexity and quality.

Figure 13 shows the subjective quality comparison. As shown in the figure, the MHT-based FMC suffers from the blur around the numbers while the number blurring is significantly reduced by the proposed FMC.

Within the 60 frames of the Foreman sequence, the PSNR of each frame is shown in Figure 14. Three lines show the proposed FMC, the MHT-based FMC, and the original H.264 encoder with no FMC. An intraframe is inserted once in every 10 frames, and the peaks in the graph represent the intraframes. The MHT-based FMC significantly drops the PSNR for all frames while the proposed algorithm produces notably less quality degradation.

Since the frame compression is lossy, this raises the issue of drift, as there may be a mismatch between the encoded frame written in the compressed file, and the decoded frame stored in the memory and used later for the prediction of successive frames. The decrease of PSNR is observed in Figure 14 as the PSNR of a frame distant from an intraframe is less than that close to the intraframe not only with the proposed FMC but also with the H.264 encoder. This result shows that the drift by the propose FMC does not affect significantly the PSNR drop. In order to precisely measure the additional PSNR drop caused by the proposed FMC, the PSNR difference between the original H.264 encoder without the FMC and the integrated H.264 encoder with the FMC is shown in Figure 15. As shown in this figure, the PSNR difference does not vary significantly regardless of the distance from an intraframe. This result also shows that the additional PSNR drop caused by the proposed FMC is not very significant. This experiment is performed with various intervals of intraframe period, and the results are similar to that shown in Figure 15. Thus, the additional experimental results are not presented in this paper.

The eight scanning modes given in Figure 2 are employed based on an analog of the 4×4 intraprediction modes for an H.264 encoder. Among the eight scanning modes, the best mode is selected to minimize the DPCM error. For the selected scanning mode, the scan along the solid line is the major scanning direction whereas the scan along the dotted line is, in general, perpendicular to the major scanning direction. Therefore, the difference along the solid line is likely to be smaller than that along the dotted line. For example, consider the case when a 4×4 block includes a virtual stripe pattern so that scanning mode 0 is selected. In this case, the scan along the dotted line crosses the vertical stripe and the chance is very high that the difference along the dotted line is larger than that along the solid line. Therefore, the "source" along the dotted line is expected to have a large value.

The expectation is supported by experimental results given in Table 4. The numbers given in this table are the ratios of the average difference along the dotted line over that along the solid line. This table shows that the difference along the dotted line is about 153.4% of that along the solid line.

In an H.264 encoder, deblocking filter is the only module that stores the reference frame. Figure 16 shows a 16×16 macroblock (lightly shaded blocks) that is the current macroblock to be filtered. To perform deblock filtering, the 4×16 pixels (dark shaded blocks) above the current macroblock and 16×4 pixels in the left of the current macroblock are necessary. Note that the 4×16 pixels are already processed by the above macroblock and they are compressed before they are stored. Then, for the current macroblock, the above 4×16 pixels are read again from the reference memory and filtered and then written back again. Thus, these pixels are stored into reference memory twice. As they are compressed whenever they are stored into reference memory, they are compressed twice. The successive compressions increase the PSNR degradation.

One way to reduce the PSNR degradation is to store the data without compression for the first write of the 4×16 pixels. These 4×16 pixels are read again and then compressed in the second write. As the second write finally stores the reference frame which is to be used by the next frame, the goal of memory size reduction is achieved even though only the second write is compressed.

Table 5 shows the BD-PSNR difference between the two approaches. The numbers in the table show the BD-PSNR drop (i.e., the difference in the BD-PSNR between the original H.264 encoder and the integrated H.264 encoder with the proposed FMC). The first column shows test video sequences and the second column shows the case when both the first and second writes compress the 4×16 data whereas the third column shows the BD-PSNR drop when only the second write by deblocking filter is compressed.

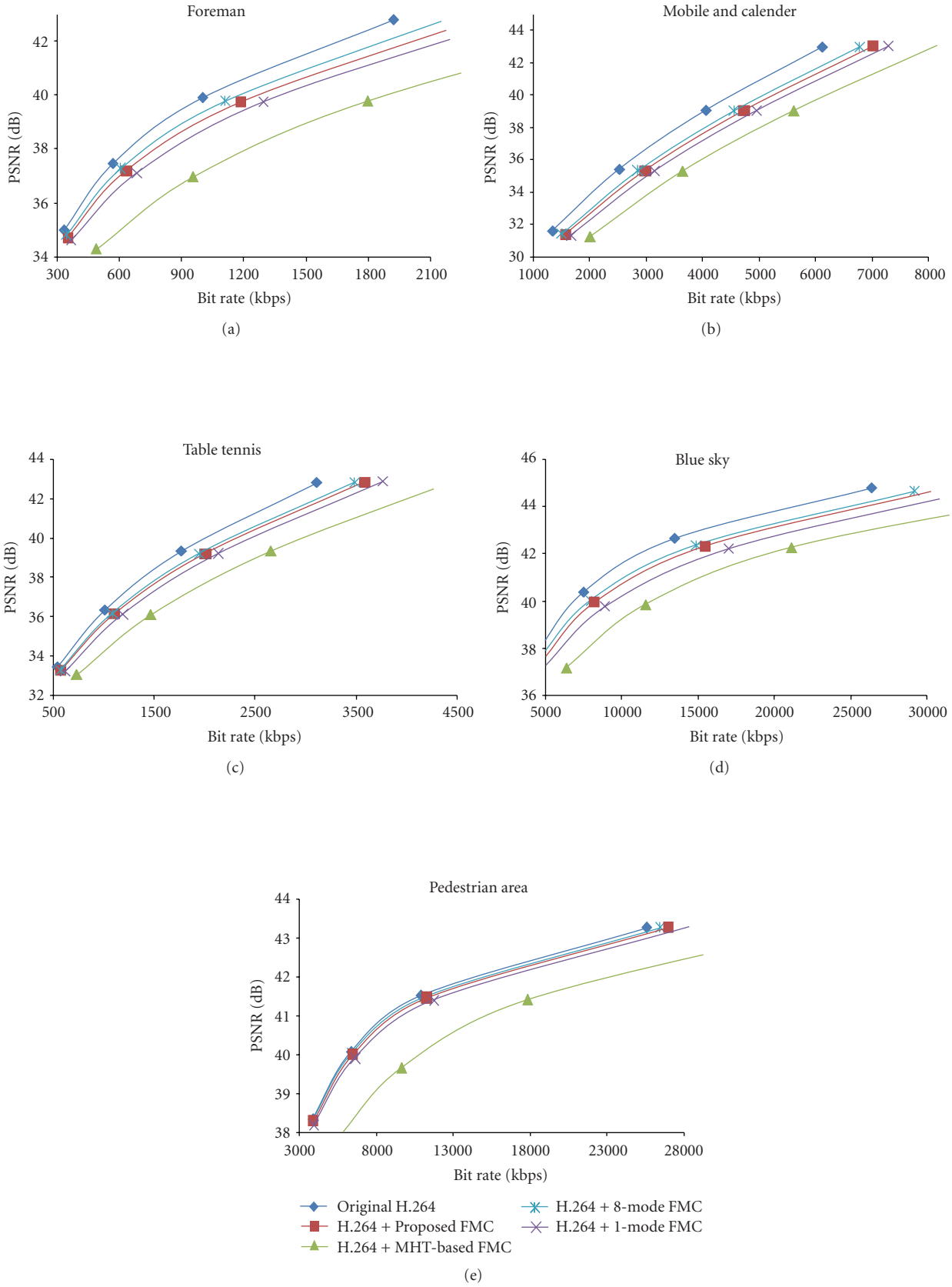


Figure 12: Rate distortion performance comparison of various FMC algorithms integrated into an H.264 Encoder.

(a) (b) (c)

Figure 13: Subjective quality comparison for Mobile Calendar sequence: (a) original H.264, (b) H.264 + Proposed FMC, and (c) H.264 + MHT-based FMC.

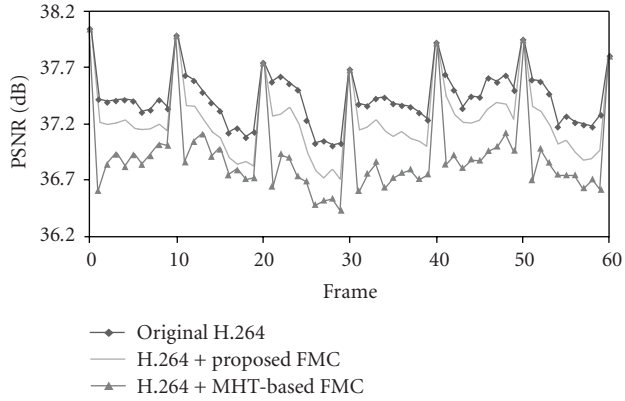


Figure 14: PSNR variations in the Foreman sequence over 60 frames.

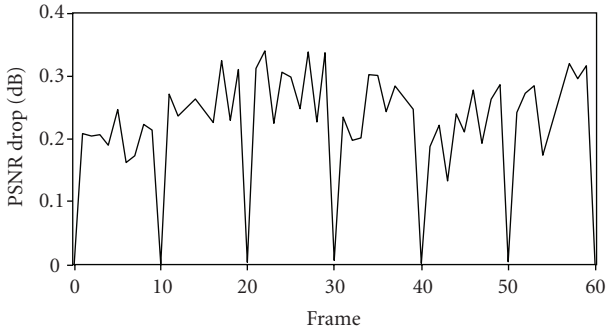


Figure 15: PSNR difference between the original H.264 encoder and the integrated H.264 encoder with the proposed FMC.

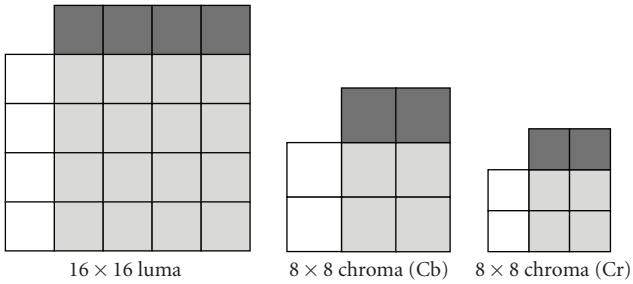


Figure 16: The pixels to be written twice for deblocking filter.

Table 5: Effect of compression by the first write of deblocking filter in BD-PSNR degradation (dB).

Sequence	Compression in both the first and second writes	Compression only in the second write
Foreman	0.69	0.65
Mobile and Calendar	1.00	0.79
Table Tennis	0.61	0.47
CIF average	0.77	0.65
Blue sky	0.65	0.49
Pedestrian Area	0.11	0.08
HD average	0.38	0.29

By storing the first write without compression, an average of about 0.12 dB improvement is achieved for CIF-size videos and an average of 0.09 dB improvement for HD size videos.

The H.264 Encoder has four modules that access the external memory. They are image sensor interface, video input module, motion estimation, and deblocking filter. The image sensor module receives pixel data from an image sensor in the YUV 4 : 2 : 0 format and stores it in the external memory. The video input module reads the input data to process H.264 encoding. The memory bandwidth to access the input data is as follows:

$$\mathrm{BW}_{\text{current frame store/load}} = H \times W \times 1.5 \times 2, \tag{4}$$

where W and H represents width and height of a frame. For the reference frame, both deblocking filter and motion estimation modules access the reference frame. The bandwidth required by deblocking filter is as follows:

$$\mathrm{BW}_{\text{DB store}} = \left(\frac{H}{16} \times \frac{W}{16}\right) \times (16 \times 16 \times 1.5 + 16 \times 4 \times 2),$$
$$\mathrm{BW}_{\text{DB load}} = \left(\frac{H}{16} \times \frac{W}{16}\right) \times (16 \times 4 \times 2). \tag{5}$$

The memory bandwidth requirement by motion estimation depends on the search range, search algorithm and data reuse scheme. The H.264 encoder in this paper adopts the full search algorithm and level C data reuse scheme [27]. Thus, the required memory bandwidth is

$$\mathrm{BW}_{\mathrm{ME\ luma}} = \frac{H}{16} \times (W + SR_H - 1) \times (16 + SR_V - 1) \times f_{\mathrm{ref}}, \quad (6)$$

where SR_H and SR_V represent the horizontal and vertical search ranges, respectively, and f_{ref} is the number of reference frame. The memory requirement for chrominance components by motion estimation is as follows [28]:

$$\mathrm{BW}_{\mathrm{ME\ chroma}} = \left(\frac{W}{16}\right) \times \left(\frac{H}{16}\right) \times (16 \times 3 \times 3 \times 2). \quad (7)$$

Thus, the total memory requirement is

$$\mathrm{BW}_{\mathrm{total}} = (\mathrm{BW}_{\mathrm{current\ frame(store/load)}} + \mathrm{BW}_{\mathrm{DB\ store}} + \mathrm{BW}_{\mathrm{DB\ load}} + \mathrm{BW}_{\mathrm{ME\ luma}} + \mathrm{BW}_{\mathrm{ME\ chroma}}) \times \mathrm{FrameRate}. \quad (8)$$

Figure 17 shows the required memory bandwidth that depends on the frame size and search range. The frame rate is 30 frames per second and all frames are encoded as P-frame. The bar graphs given in Figure 17 show the required bandwidth when search ranges ($\mathrm{SR}_H/\mathrm{SR}_V$) are 64/32 (H[−32, +31], V[−16, +15]), 128/64 (H[−64, +63], V[−32, +31]), and 196/128 (H[−98, −97], V[−64, +63]), respectively.

To support this memory bandwidth, the required operating frequency is

$$\mathrm{Freq}_{\mathrm{min}} = \frac{\mathrm{BW}_{\mathrm{total}}}{(\text{memory bus bit width} \times \text{memory bus utilization})}. \quad (9)$$

Assuming that memory bus bit width is 32 and the memory bus utilization is 100%, the line graphs show the required operating clock frequency of the external memory. The solid line graph shows the frequency for the original H.264 encoder whereas the dotted line graph shows that for the integrated H.264 encoder with the proposed FMC. Figures 17(a) and 17(b) show the cases when the number of reference frames is 1 and 3, respectively. With the proposed FMC, the total memory bandwidth is reduced to about 50% whereas the bandwidth required by the current frame remains the same. The performance of the H.264 encoder is limited when the memory bandwidth cannot meet the required bandwidth. For example, if the number of reference frames is 3, the frame size is 1920 × 1080, and the search range is 64 × 32, then required clock frequency is 233.3 MHz. For most commercially available SDRAMs (not DDR-SDRAM), this clock frequency is impossible. With the integration of the proposed FMC, the clock frequency is reduced to 138.9 MHz which is in the range of the normal operating frequency of an SDRAM. The reduction of memory traffic also makes decreases the power consumption.

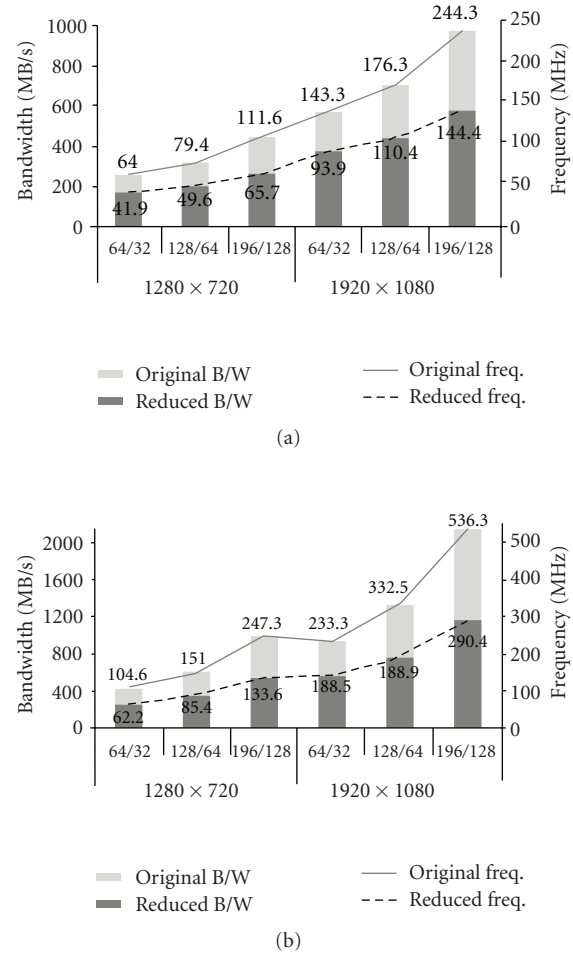


Figure 17: Reduction of the bandwidth requirement by the proposed FMC. (a) 1 reference frame. (b) 3 reference frames.

5.2. FMC for the RGB Color Space. This subsection presents the experimental results to evaluate the FMC algorithm for the RGB color space proposed in Section 3. To this end, twenty-three images of size 768 × 512 in the RGB color space shown in Figure 18 are used. Image degradations by the four FMCs with RGB 4:4:4 format, GDbDr 4:4:4 format, JPEG2000 YCbCr 4:4:4 format, and the standard YCbCr 4:4:4 format are compared. The quality degradation represented by PSNR is presented in Table 6. The boldface letters represent the best results among the five FMCs. In general, GDbDr-based FMC outperforms the others for G and R color components while the JPEG2000 YCbCr-based FMC outperforms for B color component. In general, the correlation between B and G colors (Db) is less than the correlation between B and Y colors (Cb), and consequently, the JPEG2000 YCbCr achieves the better PSNR for B color than GDbDr does.

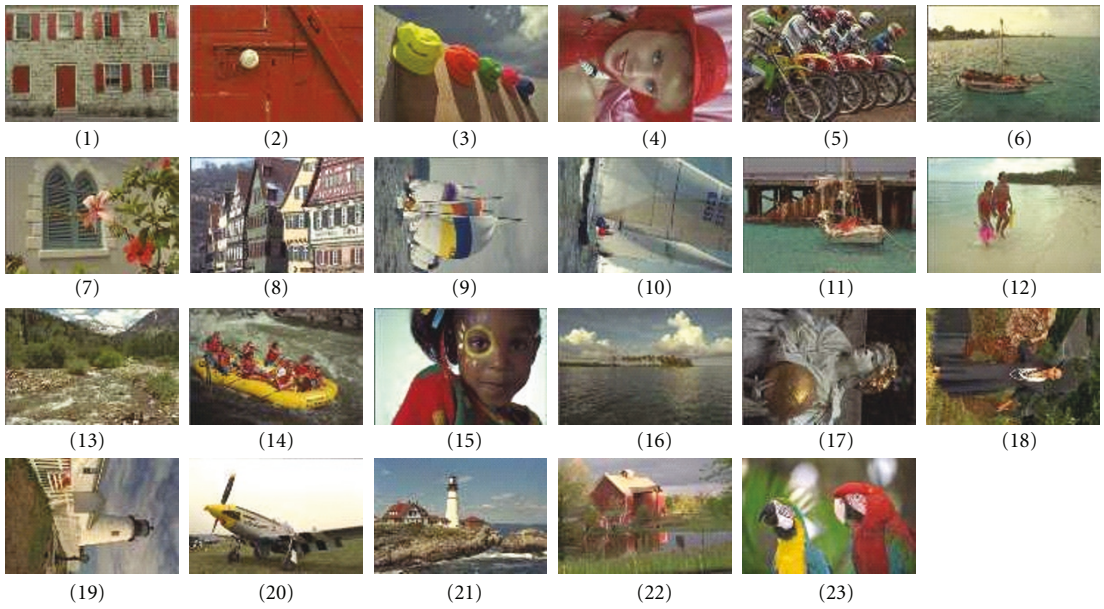


Figure 18: Test RGB bitmap images.

Table 6: PSNR (db) of frame memory compression for 23 Images with various color transformations.

Image no.	RGB 4 : 4 : 4			GDbDr 4 : 4 : 4			JPEG YCbCr 4 : 4 : 4			YCbCr 4 : 4 : 4		
	R	G	B	R	G	B	R	G	B	R	G	B
(1)	39.63	39.65	39.64	**45.42**	**49.52**	43.94	43.82	43.76	**46.51**	41.80	46.63	39.69
(2)	45.96	45.93	46.05	**48.33**	**52.60**	47.78	47.69	50.77	**48.99**	43.32	47.23	42.83
(3)	46.97	46.92	47.06	**48.62**	**53.40**	47.29	47.09	47.16	**48.11**	44.20	46.67	43.03
(4)	44.97	44.91	44.96	**47.18**	**51.36**	46.67	47.08	49.05	**47.50**	42.78	46.51	41.94
(5)	39.20	39.14	39.21	42.07	**47.06**	40.34	**42.64**	43.10	**42.67**	40.86	45.48	39.22
(6)	41.16	41.23	41.29	**46.17**	**50.17**	44.27	44.61	43.77	**45.89**	41.95	46.58	40.92
(7)	45.99	46.06	45.97	**48.24**	**52.42**	45.97	45.42	45.78	**46.50**	43.60	46.91	42.50
(8)	38.77	38.77	38.92	43.53	**47.84**	42.26	**43.69**	44.20	**43.79**	41.31	45.60	39.60
(9)	45.80	45.83	45.84	**49.59**	**53.56**	**48.11**	47.40	46.30	47.12	44.40	46.95	42.32
(10)	45.12	45.14	45.24	**48.29**	**52.92**	47.83	47.36	45.98	46.84	44.21	47.30	42.34
(11)	42.39	42.49	42.47	**46.83**	**50.64**	44.39	45.64	45.72	**45.61**	42.58	46.04	41.25
(12)	46.68	46.56	46.67	**50.15**	**54.13**	**49.32**	47.18	46.41	47.72	44.35	47.01	43.61
(13)	36.20	36.20	36.32	**41.27**	**45.76**	38.32	40.86	40.42	**41.59**	39.94	44.26	38.61
(14)	41.46	41.54	41.48	**44.50**	**48.45**	41.11	43.24	43.92	**42.88**	41.58	46.03	40.02
(15)	44.76	44.64	44.78	45.85	**51.09**	45.25	**46.72**	46.40	**46.91**	42.99	46.81	41.71
(16)	44.88	44.90	44.86	**49.76**	**53.47**	**49.54**	45.66	45.98	47.72	43.97	48.03	41.65
(17)	44.14	44.39	44.43	**48.64**	**51.62**	45.15	46.10	46.75	**46.38**	43.49	47.27	41.70
(18)	39.73	39.82	39.87	**43.07**	**46.87**	39.61	42.19	42.28	**42.23**	40.81	45.25	39.68
(19)	43.02	43.17	43.15	**47.19**	**50.99**	45.31	43.90	45.47	**46.88**	42.26	46.83	41.40
(20)	44.49	44.65	44.69	**48.37**	**52.04**	46.07	44.96	46.03	**46.86**	43.29	47.79	42.51
(21)	41.12	41.19	41.27	**46.74**	**50.23**	43.87	43.88	44.79	**45.29**	42.18	46.02	41.14
(22)	42.97	43.15	43.21	**45.55**	**49.06**	42.52	43.59	44.22	**44.81**	41.86	46.28	40.67
(23)	46.72	46.91	46.94	**48.67**	**53.10**	**47.81**	47.04	47.34	47.46	43.88	46.95	42.88
Avg.	43.14	43.18	43.23	**46.70**	**50.80**	44.90	45.12	45.46	**45.92**	42.68	46.54	41.36

(a) (b)

(c) (d)

(e) (f)

Figure 19: (a), (c), and (e) are the original images of test 5, 13, 18 and (b), (d), and (f) are their GDbDr-based FMC compressed images.

As the human eyes are more sensitive to the G color than the B color, it is reasonable to choose the GDbDr-based FMC rather than the JPEG2000 YCbCr-based FMC. Note that the average PSNRs for R, G, and B achieved by the GDbDr-based FMC are 46.70 dB, 50.80 dB, and 44.90 dB, respectively. As the PSNR is very large, the quality degradation is hardly observed. Among the twenty-three test images, three images with the lowest PSNR degradation are chosen and the images with the GDbDr-based FMC are compared with their original images. Figure 19 shows these images and it is very hard to distinguish the original image from the compressed image.

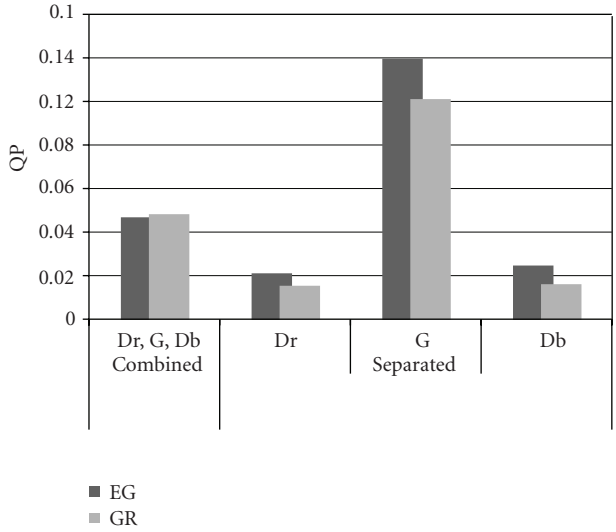
The FMC algorithm proposed in Section 3.6 is also evaluated with the twenty-three RGB images. Recall that

Table 7: PSNR (db) of 75% frame memory compression for 23 Images with various color transformations.

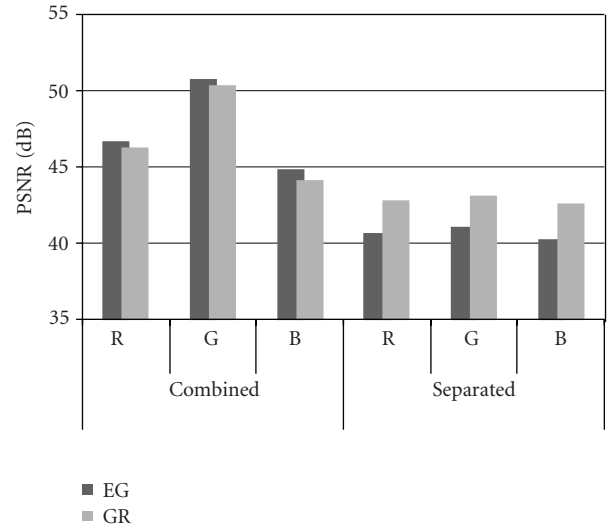
Image no.	RGB 4 : 2 : 0			GDbDr 4 : 2 : 0			JPEG YCbCr 4 : 2 : 0			YCbCr 4 : 2 : 0		
	R	G	B	R	G	B	R	G	B	R	G	B
(1)	22.10	38.25	21.94	**35.79**	**39.73**	**36.98**	35.35	36.85	36.15	36.03	39.34	35.75
(2)	28.33	44.90	29.28	33.69	**46.17**	**41.53**	**35.28**	42.38	39.63	34.92	42.43	36.63
(3)	29.73	45.58	30.61	37.90	**46.94**	36.25	**38.74**	41.26	**37.57**	38.23	43.82	36.52
(4)	28.44	43.71	28.33	33.96	**45.26**	**41.12**	**35.82**	40.83	40.60	35.25	41.86	37.55
(5)	21.83	37.68	22.12	33.75	**39.06**	32.65	**34.13**	35.65	**33.40**	34.81	38.55	33.12
(6)	23.23	39.82	23.64	**38.23**	**41.39**	**36.68**	36.97	38.07	36.46	37.38	41.15	35.47
(7)	27.46	44.61	27.47	37.70	**45.85**	36.35	**38.02**	40.49	**37.46**	37.88	43.39	35.87
(8)	19.27	37.41	19.42	34.06	**38.30**	34.35	**34.31**	36.03	**34.51**	34.58	38.06	33.52
(9)	27.34	44.37	27.73	**39.66**	**45.69**	37.79	39.41	41.13	**38.33**	38.12	43.93	36.62
(10)	27.96	43.80	27.96	39.01	**44.88**	38.27	**39.20**	40.89	**38.71**	38.05	43.49	36.75
(11)	24.82	41.06	25.29	36.31	**42.54**	39.02	**36.68**	39.08	**37.94**	36.48	41.48	36.41
(12)	28.32	45.25	27.99	38.86	**46.54**	39.23	**39.60**	41.54	**39.95**	38.42	44.11	37.63
(13)	20.02	34.90	20.06	**34.77**	**36.39**	32.05	33.10	33.50	32.03	34.79	36.72	**32.25**
(14)	24.05	40.13	25.02	32.59	**41.63**	32.57	**33.18**	37.20	**32.85**	33.90	39.55	32.88
(15)	27.41	43.38	27.72	33.65	**44.80**	37.85	**35.53**	39.65	**38.66**	35.28	41.52	36.88
(16)	27.19	43.62	27.67	**41.69**	**45.05**	**40.27**	39.78	41.26	39.61	39.25	44.28	37.77
(17)	27.45	42.85	27.15	**40.12**	**44.25**	**38.08**	38.75	40.87	37.94	38.53	43.41	36.42
(18)	23.56	38.34	23.74	**35.24**	**39.74**	33.50	34.71	36.09	**33.78**	35.43	39.24	33.27
(19)	23.59	41.82	24.12	**38.35**	**42.92**	**38.03**	37.51	39.84	37.97	37.59	42.21	36.16
(20)	25.86	43.19	25.89	**39.97**	**44.20**	36.18	38.15	40.32	**37.01**	38.95	43.52	35.51
(21)	23.95	39.75	24.29	**38.00**	**41.37**	**36.25**	36.58	38.16	35.93	37.10	41.09	35.15
(22)	26.39	41.76	26.09	**35.65**	**43.23**	35.20	35.42	38.40	**35.88**	36.07	41.15	34.67
(23)	29.77	45.34	29.38	37.29	**47.10**	37.47	**37.82**	41.39	**38.21**	37.45	44.18	36.30
Avg.	25.57	41.81	25.78	**36.79**	**43.18**	36.86	36.70	39.17	**36.98**	36.72	41.67	35.61

this algorithm achieves 75% compression by combining the 50% FMC algorithm and another 50% compression by color transform and subsampling from RGB 4 : 4 : 4 format into YCbCr (or GDbDr) 4 : 2 : 0 format. Table 7 shows PSNR values when the images are compressed by 75%. For the transform into the standard YCbCr color space and the JPEG YCbCr color space are evaluated and compared with the GDbDr color space. As shown in Table 7, the FMC with GDbDr color space achieves the best quality for the G and R color component while the FMC with JPEG2000 YCbCr color space achieves the best quality for B color components. The FMC with the standard YCbCr color space does not outperform for any image. When compared with the image quality by the 50% FMC algorithm presented in Table 6, the image quality is significantly degraded as PSNR is much less than that in Table 6. The quality degradation is caused by large compression ratio.

Figure 20 shows the efficiency of the combined packetization which is explained in Section 3.1. Figure 20(a) shows the average QPs for the combined and separate packetizations. Note that QP + 1 corresponds to the number of iteration in FMC encoding. To compare the efficiencies of Golomb-Rice coding and Exp-Golomb Coding, both are used for the compression and their results are compared. In this figure, GR and EG stand for Golomb-Rice and Exp-Golomb codings, respectively. The combined packetization reduces the average QP of G color, but it increases the average QP of Dr and Db colors. This implies that the degradation of G color is substantially reduced while those of R and B colors may increase. Figure 20(b) compares the PSNR of combined packetization with that of separated packetization. As shown in this figure, the PSNR of G colors as well as R and B colors increase when combined packetization is adopted. It is because the error reduction of the G color also affects the error reduction of R and B colors, consequently resulting in the improvement of PSNR for all three color components. It is also shown that Golomb-Rice coding is efficient for separate packetization while Exp-Golomb coding is efficient for combined packetization. This is shown in Figure 20(a) because the QP of Exp-Golomb coding for combined packetization is slightly less than that of Golomb-Rice coding while the QP is substantially increased by Exp-Golomb coding for separate packetization. In Figure 20(b), it is shown that Exp-Golomb coding achieves better PSNR than Golomb-Rice coding for combined packetization, but less PSNR for separate packetization.



(a)



(b)

Figure 20: QP and PSNR comparison between combined and separate packetization. (a) The QP averaged over twenty-three test images shown in Figure 18. (b) The PSNR averaged over twenty-three test images shown in Figure 18.

6. Conclusions

This paper proposes an FMC algorithm that compresses video data to be stored into frame memory. The proposed FMC algorithm achieves lower image degradation than other transform based algorithms. The computational complexity is relatively small because the information given by H.264/AVC video encoding is used. By using the pixel correlation information, which comes from the H.264/AVC intraprediction, an efficient DPCM scan order is selected without a significant increase in the amount of computation. The proposed algorithm performs the compression in a 4×4 block, so that both horizontal and vertical correlations are exploited. As a result, higher compression efficiency is achieved than with an MHT-based algorithm, which exploits only the horizontal correlations. As a result, compared to an MHT-based algorithm, image quality is improved by an average of 1.62 dB and 1.34 dB for CIF-size and HD-size images, respectively.

The proposed FMC algorithm is modified for the system without an H.264/AVC encoder. As the intraprediction result from H.264/AVC is not available, an additional step to select the best scan order is necessary. This system, in general, stores RGB colors instead of YCbCr colors as in H.264/AVC compression. For improved compression efficiency, the RGB color space is transformed into another color space and then compression algorithm is performed for the transformed domain. Experiments with various color spaces show that the most efficient result is obtained with the G, R-G, B-G color space.

Acknowledgments

This work was sponsored by ETRI System Semiconductor Industry Development Center, Human Resource Development Project for IT-SoC Architect, and CAD tools were supported by the IDEC.

References

[1] Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, "Draft ITU-T recommendation and final draft international standard of Joint Video Specification (ITU-T Rec. H264—ISO/IEC 14496-10 AVC)," in *Proceedings of the 7th Meeting on Document JVT-G050d35*, Pattaya, Thailand, March 2003.

[2] T.-C. Chen, S.-Y. Chien, Y.-W. Huang, et al., "Analysis and architecture design of an HDTV720p 30 frames/s H.264/AVC encoder," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 16, no. 6, pp. 673–688, 2006.

[3] T.-M. Liu, T.-A. Lin, S.-Z. Wang, et al., "A 125 μW, fully scalable MPEG-2 and H.264/AVC video decoder for mobile applications," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 1, pp. 161–169, 2007.

[4] Y. Chen, C. Cheng, T. Chuang, C. Chen, S. Chien, and L. Chen, "Efficient architecture design of motion-compensated temporal filtering/motion compensated prediction engine," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 18, no. 1, pp. 98–109, 2008.

[5] V. G. Moshnyaga, "Reduction of memory accesses in motion estimation by block-data reuse," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '02)*, vol. 3, pp. 3128–3131, Orlando, Fla, USA, May 2002.

[6] 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 *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME '08)*, pp. 825–828, Hannover, Germany, June 2008.

[7] T. Y. Lee, "A new frame-recompression algorithm and its hardware design for MPEG-2 video decoders," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 6, pp. 529–534, 2003.

[8] T. Song and T. Shimamoto, "Reference frame data compression method for H.264/AVC," *IEICE Electronics Express*, vol. 4, no. 3, pp. 121–126, 2007.

[9] Y. V. Ivanov and D. Moloney, "Reference frame compression using embedded reconstruction patterns for H.264/AVC decoders," in *Proceedings of the 3rd International Conference on Digital Telecommunications (ICDT '08)*, Bucharest, Romania, July 2008.

[10] J. Someya, A. Nagase, N. Okuda, K. Nakanishi, and H. Sugiura, "Development of single chip overdrive LSI with embedded frame memory," in *Proceedings of the International Symposium Digest of Technical Papers (SID '08)*, vol. 39, pp. 464–467, Los Angeles, Calif, USA, May 2008.

[11] J. Someya, N. Okuda, and H. Sugiura, "The suppression of noise on a dithering image in LCD overdrive," *IEEE Transactions on Consumer Electronics*, vol. 52, no. 4, pp. 1325–1332, 2006.

[12] J. Strom and T. Akenine-Moller, "PACKMAN: texture compression for mobile phones," in *Proceedings of the 2nd International Conference on Computer Graphics and Interactive Techniques*, p. 66, Singapore, August 2004.

[13] J. Strom and T. Akenine-Moller, "iPACKMAN: highquality, low-complexity texture compression for mobile phones," in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, pp. 63–70, Los Angeles, Calif, USA, 2005.

[14] D. Kim, K. Chung, C.-H. Yu, et al., "An SoC with 1.3 Gtexels/s 3-D graphics full pipeline for consumer applications," *The IEEE Journal of Solid-State Circuits*, vol. 41, no. 1, pp. 71–84, 2006.

[15] S. W. Golomb, "Run-length encodings," *IEEE Transactions on Information Theory*, vol. 12, pp. 399–401, 1966.

[16] R. F. Rice, "Some practical universal noiseless coding techniques," Tech. Rep., Jet Propulsion Laboratory, California Institute of Technology, Pasadena, Calif, USA, 1979.

[17] I. E. G. Richardson, *H.264 and MPEG-4 Video Compression: Video Coding for Next-Generation Multimedia*, John Wiley & Sons, New York, NY, USA, 2003.

[18] ITU-R BT.601-5, "Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios," ITU-T, 1995.

[19] C. Christopoulos, A. Skodras, and T. Ebrahimi, "The jpeg2000 still image coding systemml: an overview," *IEEE Transactions on Consumer Electronics*, vol. 46, no. 4, pp. 1103–1127, 2000.

[20] S. C. Pei and I. K. Tam, "Effective color interpolation in CCD color filter arrays using signal correlation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 6, pp. 503–513, 2003.

[21] K. Hirakawa and T. W. Parks, "Adaptive homogeneity-directed demosaicing algorithm," *IEEE Transactions on Image Processing*, vol. 14, no. 3, pp. 360–369, 2005.

[22] D. Menon, S. Andriani, and G. Calvagno, "Demosaicing with directional filtering and a posteriori decision," *IEEE Transactions on Image Processing*, vol. 16, no. 1, pp. 132–141, 2007.

[23] X. Li, "Demosaicing by successive approximation," *IEEE Transactions on Image Processing*, vol. 14, no. 3, pp. 370–379, 2005.

[24] J.-S. Jung, G. Jin, and H.-J. Lee, "Early termination and pipelining for hardware implementation of fast H.264 intraprediction targeting mobile HD applications," *EURASIP Journal on Advances in Signal Processing*, vol. 2008, Article ID 542735, 19 pages, 2008.

[25] Joint Model (JM)—H.264/AVC Reference Software, http://iphome.hhi.de/suehring/tml/download/.

[26] G. Bjontegaard, "Calculation of average PSNR differences between RD curves," in *Proceedings of the 13th VCEG Meeting in Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG*, Documents VCEG-M33, Austin, Tex, USA, March 2001.

[27] J.-C. Tuan, T.-S. Chang, and C.-W. Jen, "On the data reuse and memory bandwidth analysis for full-search block-matching VLSI architecture," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, no. 1, pp. 61–72, 2002.

[28] H. Hongqi, X. Jiadong, D. Zhemin, and S. Jingnan, "High efficiency Synchronous DRAM controller for H.264 HDTV encoder," in *Proceedings of the IEEE Workshop on Signal Processing Systems (SiPS '07)*, pp. 373–376, Shanghai, China, October 2007.