International Journal of Multimedia and Ubiquitous Engineering
Vol.11, No.3 (2016), pp.45-56
http://dx.doi.org/10.14257/ijmue.2016.11.3.05

Fast H.264 Video Decoding by Bit Order Reversing and Its Application to Real-time H.264 Video Encryption

Kang Yi[1], Yuo-Han Lee[1] and Jeong-Hyun Joo[2]
[1] School of Computer Science and Electrical Engineering, Handong Global University, Pohang, Korea
[2] Department of Computer Engineering, Seoul National University, Seoul, Korea
yk@handong.edu, incf1159@gmail.com, koo221@naver.com

Abstract

The demand on fast H.264 video processing technology is highly increasing since the electronics devices with encoding, transmission, and decoding high-resolution video are becoming popular in the consumer market. In this paper, we present a fast H.264 video decoding method by modifying the algorithm dealing with parsing bit-stream of variable-length code (VLC). The key idea of our approach is to reverse the bit order in a fast way through LUT (Look-Up-Table) in order to reduce the bit pattern extraction time for VLC stream. We demonstrated the speedup by employing the proposed idea for H.264 real-time video encryption application. The experimental results show about 11% execution time reduction in the part of video stream parsing function and 3.1% time reduction in the whole video encryption system.

Keywords*: H.264 video decoding, variable length coding, fast video stream parsing, accelerating H.264 decoder, Lookup table based decoding*

1. Introduction

Recently, enormously massive video data is created by everybody, everywhere, and everyday through many ways and reasons such as real-time video streaming service for entertainment business, surveillance cameras for social security purpose and personal wearable devices for life-logging purpose [8-10]. Video data should be stored and transmitted in a highly-compressed form because of its huge data volume. In order to process any recorded video data, the decoding (inverse of encoding) process is necessary. Many of the video application requires real-time processing performance. Especially, for the increasing interest in personal privacy protection and the social demand on safety and security protection against terrorism and natural disaster, fast real-time video encryption demand is crucial [1].

In a video processing system, according to our observation, the time portion of video decoding accounts most of the whole processing time. Here, we select an existing video encryption system from [2] as a reference to illustrate the importance of decoding time reduction and to demonstrate the impact of our proposed fast decoding method. Figure 1 shows the decoding ratio of a reference video encryption system. The input video data in the graph have HD resolution. The reference video encryption system targets fast encryption by employing simple computing such as bit flipping and bit pattern replacement as far as maintaining the structure of encoding video format. But, in order to comply the compatibility to video standard partial video decoding is required. Note that in this case the ratio of decoding part accounts more than 85% of the whole system time on average.

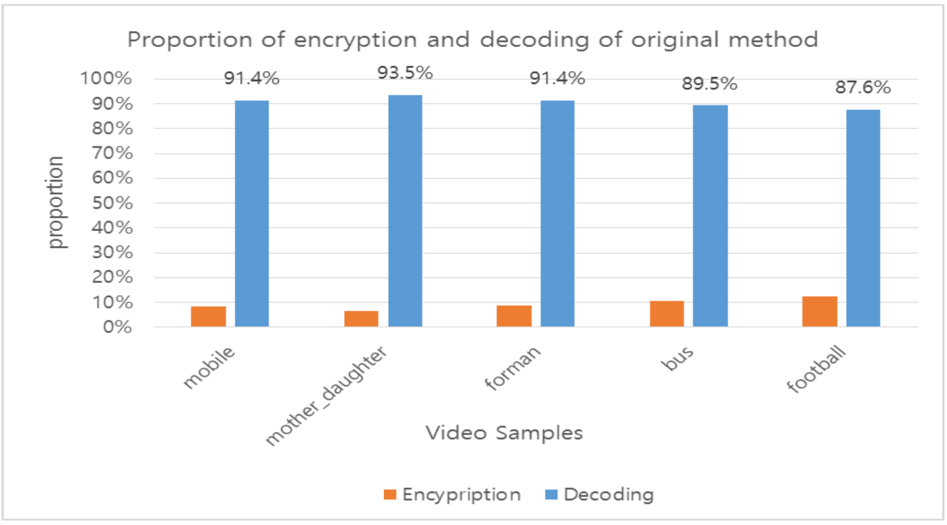

Figure 1. The Percentage of Encryption and Decoding Time Required for a Video Encryption System per HD Video Data

There are many efforts for faster video decoding. The existing related works can be categorized into two. One is software-oriented parallel processing based approach. The other is developing dedicated hardware to achieve real-time performance. Reference [3-5] falls into the former and [6] and [6] falls in the latter.

In [3], macro block level scheduling to utilize many-core processor feature was studied resulting 60% speedup over greedy scheduling. In [4] the use GPU of Xbox 360 as parallel processing unit to accelerate the H.264 decoding in high-definition video is proposed toward implementing real-time software-only video decoder. In [5], authors presented H.264 multiprocessing proposal. They combined coarse-grained frame-level parallel decoding with fine-grained macro block macroblock level parallelism. And, they also converted input stream into meta-format to have forward-oriented and self-contained format making the processing of meta- stream independent of the original stream. On the other hand, in [6], hardware architecture for variable-length decoder of H.264/AVC by analysis of inherent parallelism of CAVLC algorithm. Especially, authors adopted Bit-position VLC decoding approach to decode Multiple symbols concurrently in a critical step in CAVLC. In [7], authors presented two case studies of different architecture for H.264 decoder by optimal hardware module placement using the profiling traffic data among H.264 modules.

Most of these existing approaches does not present any implementation to achieve real-time performance in embedded processor environment by pure software-only method. But, our target application, a real-time video encryption, requires software-oriented implementation because none of existing hardware encoder which is usually encoder ASIC, does not allow to provide intermediate internal data flow information that is crucial to guide efficient video stream encryption. In this paper, we aim to present a video stream decoding method toward software-oriented fast video decoder. The presented approach is a simple but very effective way to accelerate the video stream decoding procedure based on converting input stream format into forward-oriented stream format. In order to decode the encoded video stream, the extraction of specific bit patterns consisting of each token is necessary. One of the key for fast bit pattern extraction requires to consider bits and bytes orders in each bytes and words. We achieved the fast decoding by rearranging bits order in each byte resulting in forward-oriented stream format. We achieved this conversion in a fast way through a look-up table that adjust the input bit pattern into properly ordered patterns.

2. The Background and Problem Definition

2.1. H.264 Video Standard and Overview of Decoder

Variable length codes (VLC) are used many part of H. 264 coding. The image data residuals and some headers of video stream are coded as VLC. The main point of decoder is entropy decoder and prediction part. The efficient entropy decoder is important to achieve a fast speed decoder. Figure 2 shows the overall structure of H.264 decoder. It decodes the input VLC code in entropy decoding stage and performs de-quantization followed by inverse transform. Our decoder accelerates the process of entropy decoding stag which is shown as highlighted box in Figure 2.

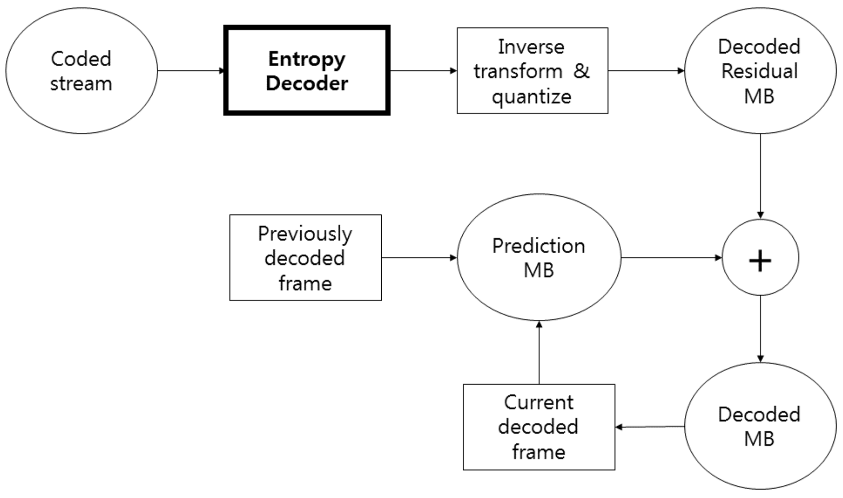

Figure 2. H.264 Decoder Block Diagram

An H.264 stream is set of NAL (Network Abstraction Layer) unit. The first byte in NAL unit is a header. This header contains information about the type of NAL unit such as SPS, PPS, and SLICE where, SPS stands for sequence parameter set and PPS stands for picture parameter set. Both unit contains the parameter information which is used by decoder. Actual image data is in slice unit. Slice NAL packet is composed of slice header and many macro blocks. Macroblock is the main source of picture information. They contains the most important information like luminance and chrominance data of pixels in each macroblock. Macro block has its header, prediction type, coded block pattern, quantization parameter and actual data. Overall structure of H.264 is shown at Figure 3.

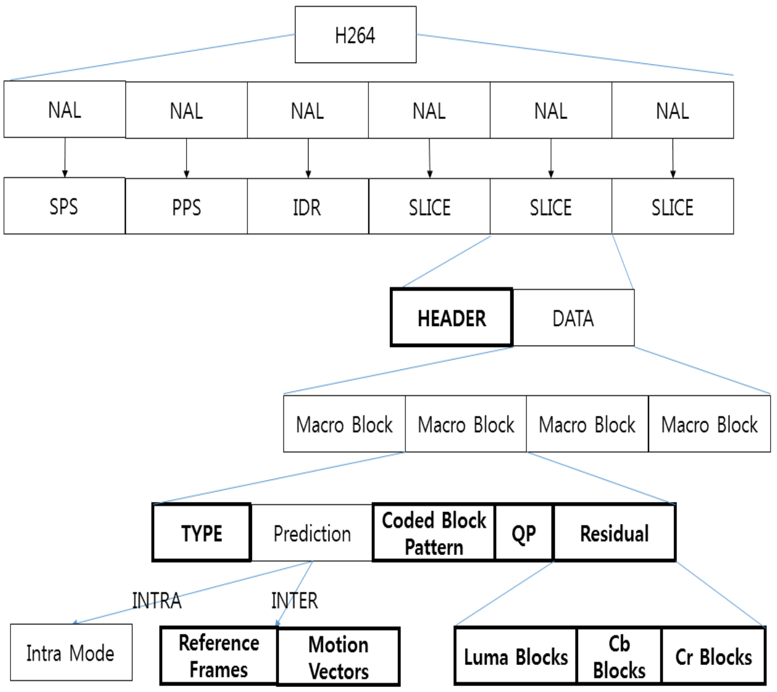

Figure 3. Overall Structure of H.264 Stream

In Figure 3, bolded blocks represent that these parts are coded in VLC. These coded data have advantages to make compact data. However, it requires additional computation to find out information from bit streams. Since, H.264 have many coded parts, extracting data from coded stream is the most time-consuming job to decode H.264 stream.

2.2. Conventional Video Decoding Procedure for Variable Length Code

The NAL units of H.264 data are stored in memory as a stream. Then binary value stream is divided into bytes. However, length of binary elements in H.264 are not fixed and there is no guarantee that size is multiple of byte size. This means that each element may not fit into byte borders. Therefore, the size of H.264 data element is not fixed and it is not padded to byte unit. This varying length makes a mismatch between data boundary and memory position. Depending on the length of code and context coded data element can be positioned within two or more bytes.

Figure 4 shows an example of coded stream data. There are five elements A-E and their binary value is 110, 1001, 1000011, 1010, 111111, respectively. The most significant bit of the first element A become the most significant bit of the lowest address byte. The least significant bit of the A is followed by the most significant bit of B which is the next element of A. If the least significant bit of an element is exactly at the boundary of a byte, the most significant bit of next element should be placed at the most significant bit of the next byte. However, in Figure 4, the element C does not end at the byte boundary and is placed in both Byte1 and Byte2. Since H.264 structure starts from most significant bit toward least significant bit order, 2^{nd} bit of element C starts from most significant bit of Byte2. Therefore, element C is composed with least significant bit part of Byte1 and most significant bit part of Byte2 as shown in Figure 4 (b) and (c). Due to varying length of data, decoder doesn't know the actual size of C when it get the Byte1. Decoder have to look next byte and iterate same process until it finds complete coded digits.

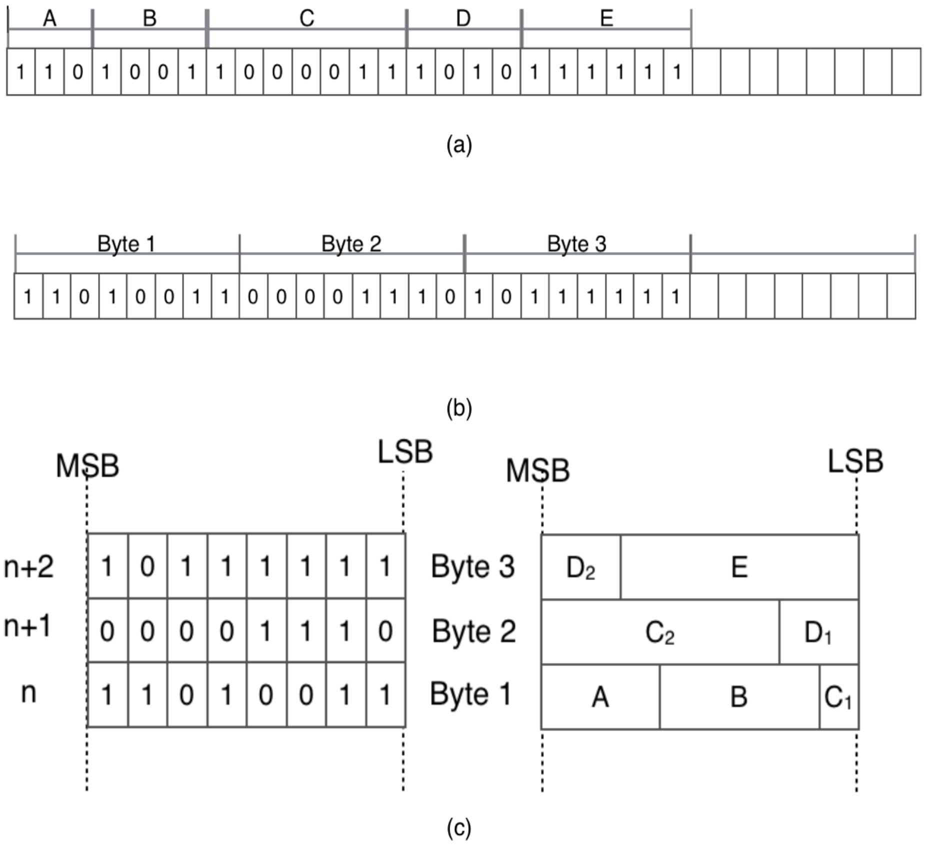

Figure 4. Example of H.264 Stream Data. (a) Data Token of H.264; (b) Byte Boundary; (c) Data Token in Actual Memory Space

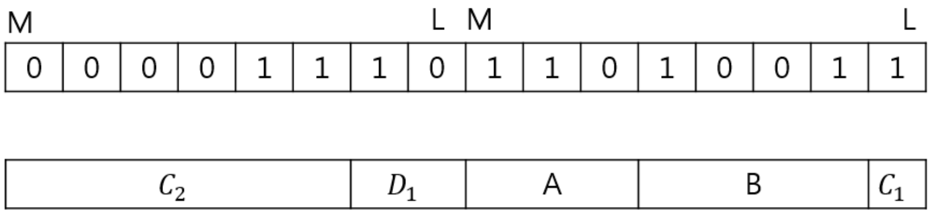

Figure 5. Stored Data in Register

In order to minimize the number of iterations through many bytes, decoders may try to read multi-bytes into 32-bits or 64-bits registers. But, Figure 5 shows what happens when decoder handles many bytes at one time to prevent iteration. In Figure 6, M and L denote most significant bit and least significant bit in each byte, respectively. , the 1st bit of register is the rightmost L position and the 16th bit is the leftmost M of the Figure 5. We assume a little-endian system which maps lower address byte in memory in a lower byte in a data word. The read data bytes, Byte1 and Byte2 from the two consecutive locations of memory in Figure 4 (c) are placed in 16-bit register as shown in Figure 5 lower part. The sub-elements C_1 and element C_2 that consist of element C are separated in a register and the flow continuation is lost in reading bit by bit manner. Furthermore, the flow of data reading in the register is right to left. However, actual data location does not match with it. This means that decoder have to iterate to parse the data.

2.3 H.264 Video Encryption Algorithm

Video encryption algorithms are divided into two classes. The first one consider the video just as a stream of data and encrypt data directly using traditional cipher algorithms. This is full encryption using standard encryption algorithm such as AES and RSA. The first one requires heavy computational power and is not adequate to real-time processing, especially, for mobile devices that operate with battery. Another one is selective encryption. Unlike the first one, these algorithms choose the part of compressed video data to encrypt. In this paper, we choose one of the most efficient selective video encryption algorithms, which is published in [2].

The encryption algorithm in [2] firstly classifies the input video according to the texture and motion complexity. The texture and motion intensity complexity is measured as shown in Figure 6. The input video is classified into one of six categories according to the two-class of texture complexity measurement and three-class of motion complexity measurement.

Figure 7 shows the overall process of the video encryption in [2]. The selective encryptions are applied in three ways: intra_pred_mode (intra prediction mode of macro block) encryption, LowCoeffSign (DCT coefficient codeword parameter) encryption, and MVD (motion vector difference) codeword encryption. Based on the video classified results, one of the six partial encryption options is selected. For example, the complex-texture and high-intensity motion video needs all of the ‘intraPredMode’ and ‘non-zero DCT’ and ‘MVD of P-frame’ encryption, and the non-complexity texture and low-intensity motion video needs 3 ‘low-band non-zero DCT’ and 33% of ‘MVD in P frame’. Note in all the process of classification and encryption operations, the compressed video need to be partially decompressed. As shown in the Figure 1, more than 90% of time is consumed by decoding. Thus, even for the selective video encryption method, minimizing the decryption time is crucial for the real time implementation of the video encryption system.

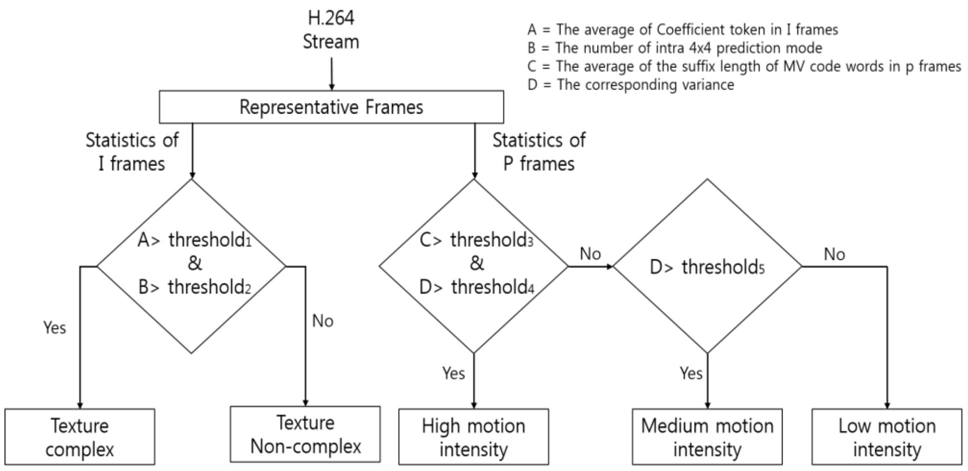
H.264 Stream
A = The average of Coefficient token in I frames
B = The number of intra 4x4 prediction mode
C = The average of the suffix length of MV code words in p frames
D = The corresponding variance
Representative Frames
Statistics of I frames
Statistics of P frames
A> threshold1 & B> threshold2
C> threshold3 & D> threshold4
D> threshold5
Yes
No
Yes
No
Yes
No
Texture complex
Texture Non-complex
High motion intensity
Medium motion intensity
Low motion intensity

Figure 6. Input Video Frame Classification Criteria of [2]

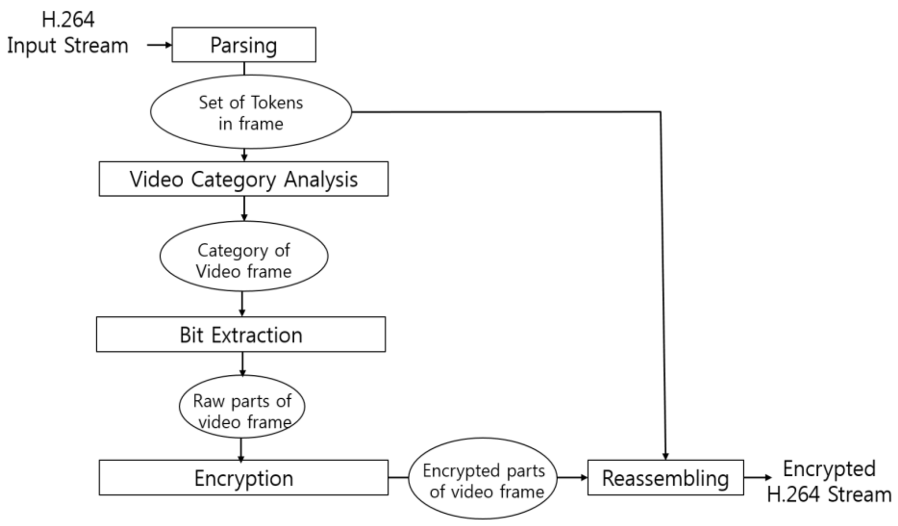

Figure 7. The Overall Procedure of Video Encryption Of [2]

3. Proposed Method Using Bit-Reordering

3.1 General Case

As illustrate by Figure 5 in Section 2, the fragmentation of element is the main problem that makes the additional iterative computation and it is unavoidable as long as element size is variable. Here, we found a solution to avoid this discontinuation in the byte boundary. Our idea is based on a very simple bit order reversing in each byte when bringing the data from memory.

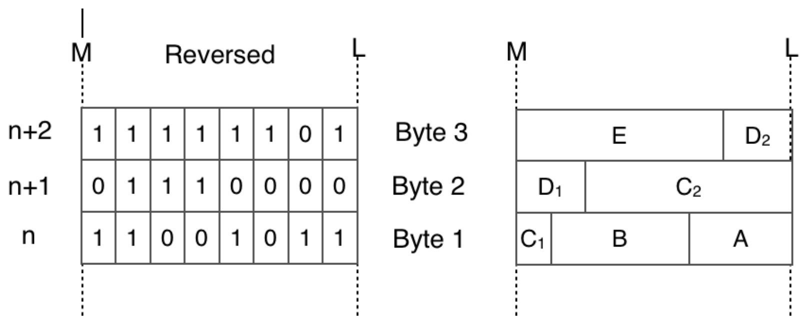

Figure 8. The Memory Structure after Bit Reversion

We depict our method in Figure 8 in which shows the bit map after bit reversing. In the Figure 8, we assume that there are three symbols A, B, C and each symbol is stored in a memory sequentially in a compact form beginning address n to n+2. The bits are reversed and the order of symbols are also reversed in each byte. For example, element A comes least significant bit place and C_1 comes the most significant bit place in Byte1, which is contrast to the case in Figure 5. The symbol C has disconnection point between Byte1 and Byte2 in the Figure 5, but in the Figure 6, after bit revering in each byte the sub-element C_1 and C_2 are placed next each other. That is, the lower part of element C, C_1 is placed the most significant bit of Byte1 and the upper part of element C, C_2, is placed at the least significant bit of Byte2 resulting in making element C value in a connected form. Now, it is possible to read element C without iteration through two bytes when fetched into register. Figure 9 shows the register contents example after bit order reversing. Compared to Figure 5, symbol data are aligned as data flow direction of register. Decoder doesn’t need to iterate. Bit mask operation can exactly extract the given data just by using shift-right operation and bitwise-AND operations.

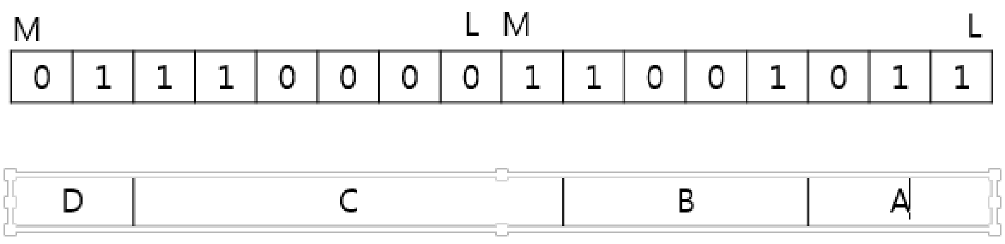

Figure 9. Stored Data in Register (Reversed Data Version)

It is noteworthy to understand the symbols is not separated in a register. Since, the bit order is reserved, reading order of right-to-left in bit level should be guaranteed to find exact value of the original symbol. But, the computers treats the right part as least significant bit inhibiting right-to-left order in any selected part of bits. Thus, one more bit-order reversing is required when identifying the symbol name of bits.

3.2 Endian Consideration

Endianness means byte ordering method in computer memory. There are two kind of endianness, big-endian and little-endian systems. Little-endian places the MSB at the highest-address byte and the LSB is stored at the lowest-address memory. Big-endian does in opposite manner. With big-endian, MSB is at the lowest and LSB is at the highest byte. We mentioned that bit reversed elements are connected at 4-byte variable. It is true when the bit ordering system is little endian. But, Big-endian is quite different as shown in Figure 10.

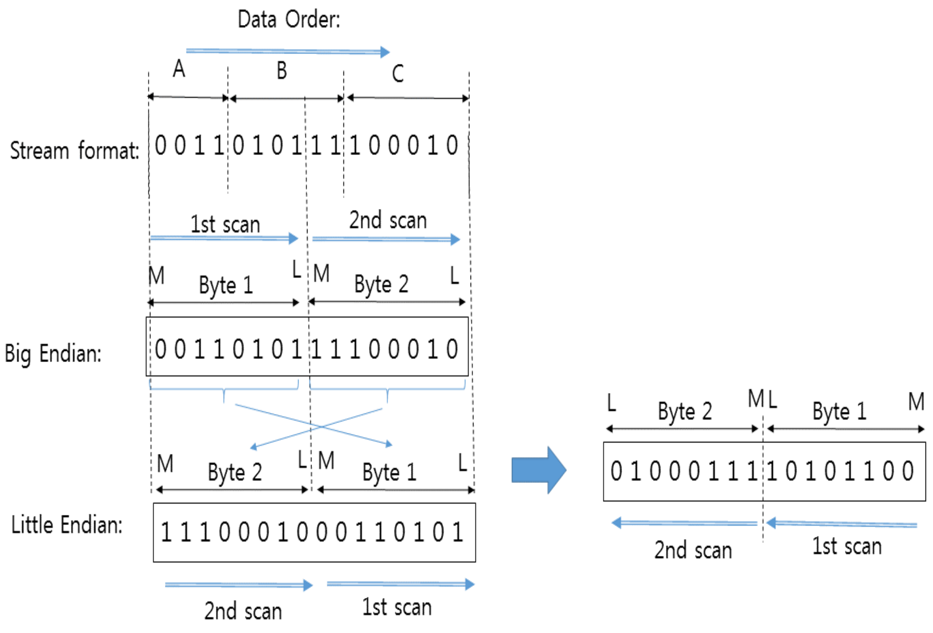

Figure 10. Problem of Little Endian and Its Correction by Bit Reordering

Figure 10 shows what is differences when bytes are ordered as big-endian and little-endian. We assume 3 symbols A(0011), B(010111) and C(100010) are in a stream. Value in memory is same for big-endian and little-endian but meaning is different. In case of big endian, byte at lowest position means MSB and byte at the highest position represents the LSB. Unlike the little endian which need bit reverse operation to be continuous at register, the big-endian doesn't need to do that. With big-endian, you need only left-to-right scanning and no need to reverse bit order in each of byte.

4. Experimental Results

We implemented the reference H.264 video encryption method [2] using JM H.264 standard reference software and modified the code to include our proposed decoding method that utilizes bit reversing. We performed experiments with several video files. We measured the elapsed time of bit extraction function, decoder part including bit extraction, and whole video encryption system including all parts of decoding and encryption. In the Table 1, we show the details of experimental results. We analyzed the computing time and compared the results between original and modified by our bit-stream decoding approach. For the bit extraction part itself, ours has better speed than the original one by 11.45% on average. While, for the decoding part which includes the bit extraction part, our decoding speed-up is about 3.6% on average compared to the original decoding time. On the other hand, in terms of the whole system, our modified system achieved 3.26% computing time saving.

Table 1. The Analysis of the Computing Time Between Original Video Encoding System and the Modified Video Encoding System with Our Approach in JPEG Decoding

Time	mobile	Mother_ daughter	foreman	bus	football	average
Original Bit Extraction (A)	1.873	0.568	1.854	2.038	0.550	1.377
Reduced Bit Extraction (B)	1.634	0.514	1.659	1.797	0.49	1.219
(B-A)/A * 100 (%)	12.76%	9.37%	10.50%	11.81%	10.97%	11.45%
Original Decoding (C)	5.832	2.018	5.889	6.338	1.845	4.385
Reduced Decoding (D)	5.593	1.965	5.694	6.097	1.785	4.227
(C-D)/C * 100 (%)	4.10%	2.64%	3.31%	3.80%	3.27%	3.60%
Encryption (E)	0.546	0.140	0.554	0.754	0.261	0.451
Original Total (F=E+C)	6.379	2.158	6.444	7.092	2.107	4.836
Reduced Total (G=E+D)	6.140	2.105	6.249	6.851	2.046	4.679
(F-G)/F * 100 (%)	3.75%	2.46%	3.02%	3.40%	2.87%	3.26%

In the Figure 11, we compared the conventional and proposed method integrated with referenced video encryption system [2]. The proposed method achieved 12.7% to 9.3% reduction in bit extraction function time compared to conventional method. The 2.64% to 4.1% time reduction is achieved in decoding part including bit extraction. And, 3.75% to 2.46% time reduction is achieved in the reference video encryption system integrated with our decoder method.

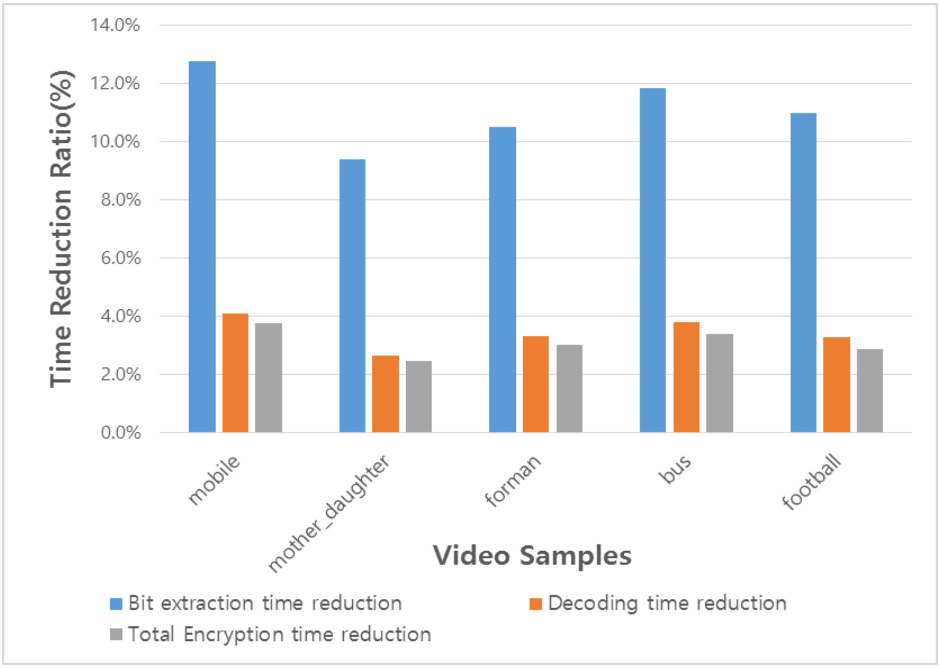

Figure 11. Experimental Results of Proposed Execution Time Reduction in Comparison With Conventional Decoder in Terms of Bit Extraction Function, Decoder, and Whole Encryption System Perspectives, Respectively

5. Conclusion

In this paper, we proposed a fast H.264 decoder by effective bit pattern extraction method. The proposed method performs bit reordering in each byte to make bit pattern matching simple. In addition, we developed look-up table based fast bit reordering method. We have applied the proposed fast decoding method to a real-time video encryption system that receives H.264-coded video stream and generates encrypted video stream compliant to H.264 standard. Experimental shows about 11.04% time reduction in comparison to existing method for the entry decoding part and 3.1% time reduction from the viewpoint of whole video encryption system.

Acknowledgement

This work was supported by the CISS funded by MSIP as Global Frontier Project (CISS- 2013M3A6A6073718).

References

[1] J. shah and V. Saxena, “Video Encryption: A Survey”, International Journal of Computer Science, vol. 8, no. 2, **(2011)**, pp. 525-534.
[2] Y. Zhao and L. Zhuo, “A Content-Based Encryption Scheme for Wireless H.264 Compressed Video”, Proceedings of International Conference on Wireless Communication & Signal Processing, **(2012)**, pp. 1-6.
[3] J. Chong, N. Satish and B. Catnzarom, “Efficient Parallelization of H.264 Decoding with Macro Block Level Scheduling”, Proceedings of IEEE Conference on Multimedia and Expo, **(2007)**, pp. 1874-1877.
[4] C. A. B. Juuan, C. William, C. Eric, B. Daniel and F. Barry, “Real-Time high definition H.264 video decode using the Xbox 360 GPU”, Proceedings of Applications of Digital Image Processing, **(2007)**.
[5] H. Richter, B. Stabernack and V. Kuhn, “Architectural decomposition of video decoders for many core architecture”, Proceedings of Conference on Design and Architectures for Signal and Image Processing, **(2012)**, pp. 1- 8.
[6] Y. N. Wen, G. L. Wu, S. J. Chen and Y. H. Hu, “Multiple-Symbol Parallel CAVLC Decoder for H.264/AVC”, IEEE Asia Pacific Conference on Circuit and Systems, **(2006)**, pp. 1240-1243.

[7] Y. Kim and E. William, "H.264 Video Decoder Design: Beyond RTL Design Implementation", IEEE Workshop on Signal Processing Systems Design and Implementation, **(2006)**, pp. 107-112.
[8] X. Xiong and B. Choi, "Improvement on Image Rotation for Relative Self-Localization Estimation", International Journal of Multimedia and Ubiquitous Engineering, vol. 8, no. 3, **(2013)**, pp. 285-294.
[9] D. Tian, "A Survey of Refining Image Annotation Techniques", International Journal of Multimedia and Ubiquitous Engineering, vol. 9, no. 3, **(2014)**, pp.117-128.
[10] C. Zhang, C. Wang and B. Jiang, "Color Image Compression Based on Directional All Phase Biorthogonal Transform", International Journal of Multimedia and Ubiquitous Engineering, vol. 10, no. 1, **(2015)**, pp. 247-254.

Authors

Kang Yi, He received the B.S, M.S. and Ph.D. in computer engineering from Seoul National University, Seoul, Korea in 1990, 1992, 1997, respectively. From 1999, he is with school of computer science and electrical engineering of Handong Global University in Pohang, Korea as a faculty member. His current research interest includes image processing for vehicular application, video/image encryption and integrity assurance.

Yuo-Han Lee, He is currently a senior undergraduate student at school of computer science and electrical engineering in Handong Global University, Korea. His major is computer engineering and pursuing B.S. of Engineering in Computer Engineering. His interests includes multimedia computing and machine learning.

Jeong-Hyun Joo, He received the B.S in Computer Engineering and Electrical Engineering from Handong Global University in 2014. He is pursuing MS degree in Computer Engineering in Seoul National University, Korea. His research interests includes algorithm design and analysis, and H.264 video encoding and encryption.