J Electr Eng Technol Vol. 9, No. 4: 1411-1417, 2014
http://dx.doi.org/10.5370/JEET.2014.9.4.1411

ISSN(Print) 1975-0102
ISSN(Online) 2093-7423

Multi-symbol Accessing Huffman Decoding Method for MPEG-2 AAC

Eun-Seo Lee*, Kyoung-Cheol Lee, Kyou-Jung Son***, Seong-Pil Moon*** and Tae-Gyu Chang†**

Abstract – An MPEG-2 AAC Huffman decoding method based on the fixed length compacted codeword tables, where each codeword can contain multiple number of Huffman codes, was proposed. The proposed method enhances the searching efficiency by finding multiple symbols in a single search, i.e., a direct memory reading of the compacted codeword table. The memory usage is significantly saved by separately handling the Huffman codes that exceed the length of the compacted codewords. The trade-off relation between the computational complexity and the amount of memory usage was analytically derived to find the proper codeword length of the compacted codewords for the design of MPEG-2 AAC decoder. To validate the proposed algorithm, its performance was experimentally evaluated with an implemented MPEG-2 AAC decoder. The results showed that the computational complexity of the proposed method is reduced to 54% of that of the most up-to-date method.

Keywords: MPEG-2 AAC, Huffman decoding, Look-up table, Computational complexity, DSP implementation

1. Introduction

Huffman decoder, which occupies about 30% of overall computational complexity of an MPEG-2 AAC decoder, is known to be one of the major processing blocks [1-2]. This paper proposes a new Huffman decoding method for the purpose to improve the implementation efficiency of the MPEG-2 AAC decoder.

The proposed algorithm uses a reconstructed Huffman table in a form of a look-up table so that the symbol search can be performed by direct memory reading. Moreover, by concatenating multiple Huffman codes within a compacted codeword, the reconstructed Huffman table allows decoding of multiple symbols in a single search, resulting in a significant enhancement of searching efficiency.

The memory usage and the searching efficiency, which normally show a trade-off relation when the Huffman table is implemented in the direct look-up table form, are the major design parameters which affect the power consumption of the Huffman decoding algorithm [3-6].

In general, the conventional binary Huffman tree is not feasible to implement in a direct look-up table form because of the requirement of a huge memory space to represent the full address range corresponding to the maximum codeword length, i.e., 19 bits in MPEG-2 AAC. Moreover, poor bit resolving ratio and bit alignment overhead of the conventional binary tree aggravate the difficulty of applying a direct look-up based decoding [7].

The memory explosion problem has been considered as one of the major technical limitations that must be resolved for the application of a direct look-up table method. Hashemian search method [8], CHT search method [9], and Hybrid search method [7] are the typical examples of the effort to minimize the problem of memory sparsity by introducing the memory space grouping or the numerical interpretation of the Huffman codes for the direct memory accessing.

The Hashemian search method [8] groups Huffman codeword into many clusters according to the range of Huffman codeword length, e.g., the range in multiple of four bit, and searches a symbol in a cascaded way by applying direct look-up table to each group. The problem of memory sparsity still exists within the memory sub-group in Hashemian method, although the total memory size is significantly reduced by the memory sub-grouping technique.

In the CHT search method [9], the memory size is further reduced by sub-grouping of the memory in further condensed and detailed levels, i.e., one group of memory for the Huffman codes having the same bit length. The Hybrid search method [7] is another example of numerical interpretation of Huffman code and direct accessing of Huffman symbol. Among these methods, the Hybrid search method shows the smallest memory usage with a comparable level of searching speed to the other most up-to-date

† Corresponding Author: Dept. of Electrical and Electronics Engineering, Chung-Ang University, Korea. (tgchang@cau.ac.kr)
* Electronics and Telecommunications Research Institute (ETRI), Korea. (eslee@etri.re.kr)
** LIG Nex1 Co., LTd., Korea. (lejesk@gmail.com)
*** Dept. of Electrical and Electronics Engineering, Chung-Ang University, Korea. ({skj9865, mczz01}@dmc.cau.ac.kr)
Received: November 21, 2013; Accepted: January 15, 2014

Huffman decoding methods.

The proposed method in this paper tries to achieve both high memory efficiency and high search efficiency by capitalizing the highly skewed statistical distribution of VLC (variable length code). In the proposed method, the amount of memory usage can be significantly saved by applying the direct look-up table technique only for short Huffman codes having high probability of occurrence. The Huffman codes which exceed the length of the compacted codeword are exceptionally treated for decoding based on a binary tree search method. The trade-off relation between the memory usage and the searching efficiency of the proposed algorithm is also analytically investigated to provide a design guide to select a proper size of the look-up table.

To verify the performance of the proposed method, it is implemented for MPEG-2 AAC decoder and its processing speed, i.e., computational complexity is measured and compared with those of the other three conventional methods, i.e., the sequential search method, the binary tree search method, and the up-to-date Hybrid method [7]. The average savings of computational complexity, when tested for 63 MPEG-2 AAC files, are shown as 84%, 63% and 46%, respectively to those of the three methods. Such complexity reductions can be considered as a significant contribution, especially for the purpose of low power implementation of MPEG2 AAC audio decoder.

2. Multi-symbol Accessing Huffman Decoding Method

In the proposed method, the conventional Huffman binary tree is reconstructed into the form of a complete binary tree and multiple number of Huffman codes are allocated in its leaf nodes. A compacted codeword table is constructed based on the leaf nodes of the fully expanded Huffman tree.

In the reconstructed table, the compacted codewords themselves are the addresses of the compacted codeword table, equivalently the addresses of the leaf nodes in the tree. It is a structure of content addressable memory where symbol decoding can be performed by direct reading of the table with the compacted codeword as its address. In this way, the use of comparison and branch instruction, which is considered as one of the major complexity increasing factors because of its pipeline stalling effect, is eliminated in the decoding algorithm.

For each address, the corresponding Huffman symbols are stored together with two values of side information. The side information, i.e., the number of Huffman symbols and the occupied bit length by the Huffman codes in the compacted codeword are needed for bit alignment operation in decoding process. A compacted codeword may contain a certain number of the truncated bits of a Huffman code at its tail part. Therefore at each decoding iteration of a compacted codeword, unused bits must be concatenated with the incoming bit stream to construct a bit-aligned new compacted codeword.

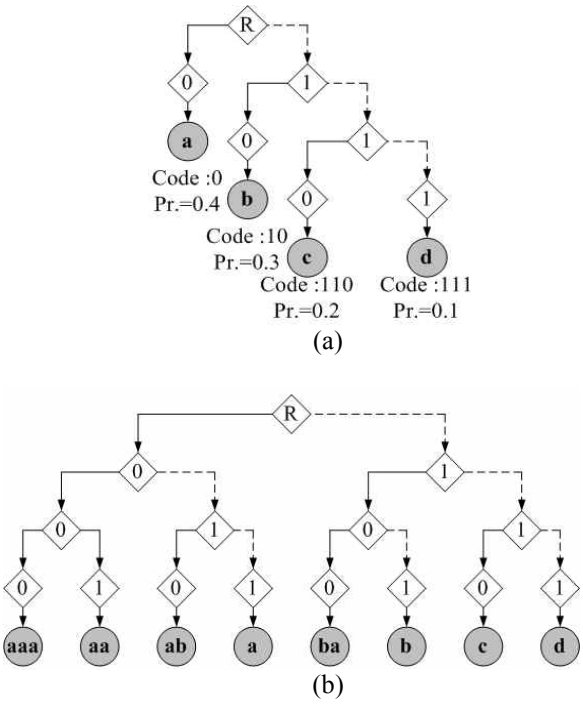

Fig. 1. An example of (a) Huffman tree and (b) its fully expanded Huffman tree to generate a compacted codeword table.

Table 1. Reconstructed compacted codeword table.

Index (Compacted codeword)	Number of symbols	Sequence of symbols	Occupied bit length
0 (0 0 0)	3	**a**, **a**, **a**	3
1 (0 0 1)	2	**a**, **a**	2
2 (0 1 0)	2	**a**, **b**	3
3 (0 1 1)	1	**a**	1
4 (1 0 0)	2	**b**, **a**	3
5 (1 0 1)	1	**b**	2
6 (1 1 0)	1	**c**	3
7 (1 1 1)	1	**d**	3

An example of compacted codeword table and its constructions are shown in the following. A conventional skewed Huffman binary tree in Fig. 1(a) is fully expanded to a complete binary tree in Fig. 1(b). Each leaf node in Fig. 1(b) corresponds to a compacted codeword having three bit word-length. The compacted codeword table is constructed using the fully expanded Huffman tree of Fig. 1(b) together with the side information as shown in Table 1.

In the Table 1, the shaded part in each compacted codeword represents the bits to be translated to their corresponding symbols. The rest bits, un-shaded part, are concatenated with the next incoming bits by the bit-alignment work. For example, in the codeword '0 0 1' of index 2, the first two '0's are translated to the two symbols 'a, a' and rest '0' is handed over for the next compacted codeword.

Table 2. The example of memory structure of conventional binary tree search method.

Index (name of node)	Leaf node	Left-child's address or return value	Right-child's address
0 (R)	False	1	2
1 (a)	True	A	-
2	False	3	4
3 (b)	True	B	-
4	False	5	6
5 (c)	True	C	-
6 (d)	True	D	-

The proposed method has advantages in terms of both searching efficiency and memory usage when compared to the conventional binary search method. The conventional binary search method performs the comparison and branch instructions for the every single incoming bit to follow the tree structure. This results in larger amount of processing time relatively to the proposed method where only the comparable amount of comparison and branch instructions are performed during one time iteration for the multiple symbols decoding. [10]

The complexity comparison of the proposed method and the conventional method can be simply demonstrated by counting the average numbers of searching operation per one symbol decoding in their example Huffman tables. In Table 1, the proposed Huffman decoding method requires average 0.615 operation to decode a single symbol as 8codeword/13symbols. In Table 2, the example of memory structure of conventional binary tree search method is shown. The conventional binary tree search method requires average 1.9 times search to decode one symbol as (1×0.4)+(2×0.3)+(3×0.2)+(3×0.1). These examples show that the number of searching times of proposed method is less as much as 1/3 than those of the conventional Huffman decoding method with only 38% (29words / 21words) increase of memory occupation in the proposed method.

In the Table 2, the index is assigned for each node, and each node contains the information where the current node is a leaf node or not. If current node is a leaf node, 'return value' indicates the Huffman symbol data. If it is not a leaf node, 'left-child's address' and 'right-child's address' are selected by next input bit (0 or 1) to indicate the index to the followed branch.

In the following section, the computational complexities of the both algorithms are calculated in more detail way with their algorithm flow-charts and compared to investigate the trade-off relation between the complexity and the memory usage.

3. Analysis of the trade-off relation between the Computational Complexity and the Memory Usage

In this section, the trade-off characteristics between the computational complexity and the amount of memory usage are investigated. In this analysis, the computational complexity and the amount of memory usage of the binary tree search method are used as the references of comparison.

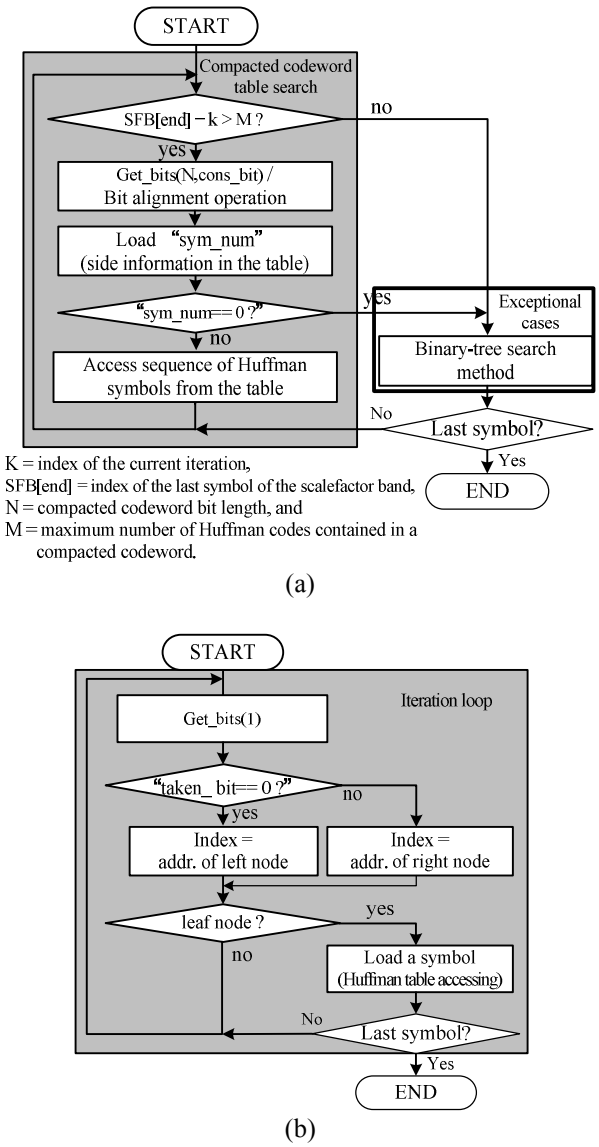

Fig. 2. The flow charts of (a) the proposed Huffman decoding method and (b) the conventional binary search method.

The flow charts of the proposed compacted codeword search method and the binary tree search method are shown in Figs. 2 (a) and (b), respectively. The decoding computational complexity was obtained by counting the number of 'load' operations and 'compare-and-branch' operations needed for each symbol decoding.

The Huffman decoding of MPEG-2 AAC is performed in the unit of each scalefactor band by associating one codebook among the eleven codebooks. The iteration of the Huffman decoding starts from the first spectral coefficient in the scalefactor band, i.e., SFB [start], to the last spectral coefficient, i.e., SFB[end] [11].

In Fig. 2 (a), the proposed Huffman algorithm performs each iteration with the multiple bits of the compacted codeword instead of the bit-wise iteration operation. At the beginning of each iteration, if the test for sufficiency of margin to the end of scalefactor band is passed, one compacted codeword of a fixed number of bits is constructed by loading and concatenating the number of bits decoded in the previous iteration. The loading operation is indicated as 'Get_bits(N, cons_bit)' in the figure. By addressing with the compacted codeword, the reconstructed Huffman codebook table is accessed to retrieve the associated Huffman symbols.

The reconstructed Huffman table also contains the information about the number of symbols to retrieve, i.e., 'sym_num'. If the value of 'sym_num' is zero, it means that the length of the codeword for the next symbol exceeds that of the compacted codeword. This case is handled exceptionally. The Huffman table also contains the number of bits consumed to decode in the current iteration, i.e., 'cons_bits'.

In Fig. 2(b), the conventional binary tree search method performs one depth search with one time of loading operation, i.e., 'Get_bits(1)', and two times of comparison operations, i.e., one to check for a leaf node and the other for the index direct to a child node. Where, if the result of the 'Get_bit(1)' is zero, i.e., 'taken_bit=0', the algorithm goes down to the left-side node. This one depth search is repeated from the root until the iteration index reaches to a leaf node, consequently to complete the acquisition of one Huffman symbol data.

The counted 'load' (L) and 'compare-and-branch' (C) operations are shown in Eqs. (1) and (2), respectively for the proposed method and the binary tree search method.

$$C_{proposed} = ((3L+2C) + \sum_{i=K+1}^{N} p(s_i)\cdot(3L+2C)\cdot(l(s_i)-D))/\alpha \quad (1)$$

$$C_{binary} = \sum_{i=1}^{N} p(s_i)\cdot(3L+2C)\cdot l(s_i) \quad (2)$$

where D : compacted codeword length,

N : total number of symbols in the Huffman codebook,
K : number of symbols for which the code length is shorter than or equal to D,
L, C : numbers of instruction cycles for a 'load' operation and a 'compare-and-branch' operation, respectively,
$p(s_i)$, $l(s_i)$: the i-th symbol s_i's occurrence probability and its bit length, respectively, and
α : average number of symbols decoded by one search.

The first term in (1), i.e., $3L+2C$ indicates the three loading and two comparison operations needed for the iteration of the main loop in Fig. 2(a). The second term in (1), i.e., $\sum_{i=K+1}^{N} p(s_i)\cdot(3L+2C)\cdot(l(s_i)-D)$ reflects the operations to handle the two exceptional cases. One case is when the Huffman code length exceeds the length of the compacted codeword, i.e., 'sym_num==0'. The other case is to avoid the margin violation in decoding of each unit of scalefactor band (SFB) of MPEG-2 AAC. The direct search loop is terminated when the number of remaining spectral coefficients in the SFB is less than the maximum possible number of coefficients contained in a single compacted codeword. The left coefficients are decoded by the binary tree search method as an exceptional case. The overall $C_{proposed}$ is obtained by dividing the complexity of one iteration by the average number of symbols retrieved from one iteration, i.e., α. It is obtained using the information of probability distributions computed from the Huffman codebooks.

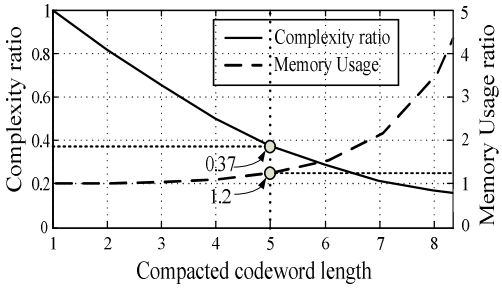

Fig. 3. The trade-off relation between the processing complexity and the memory usage for different sizes of compacted codeword *D*. The ratios of complexity and memory usage are obtained in reference to those of binary tree search.

In the binary tree search method, the main body iteration loop in Fig. 2(b), which corresponds 3L+2C operations, is repeated as much as the bit length of each Huffman codeword, i.e., $l(s_i)$. Therefore, the average number of operations per symbol search is $\sum_{i=1}^{N} p(s_i)\cdot(3L+2C)\cdot l(s_i)$, where N indicates the total number of Huffman symbols in each codebook and *p(si)* indicates the i-th symbol *si'*s occurrence probability.

The compacted codeword length D is a direct parameter effecting the trade-off relation between the computational complexity and the usage of memory. By varying the value of D, the decoding complexity ratios, $C_{proposed}/C_{binary}$, are computed and shown in Fig. 3. The averages of memory usage are also computed for different values of D by dividing the total required memory space computed for all twelve MPEG-2 AAC Huffman tables with the number of symbols in the tables.

The memory usage ratios, i.e., the ratio between the average memory usages of the proposed method to those of the binary search method are also shown in Fig. 3 to illustrate the trade-off relationships.

The result shown in Fig. 3 can provide a useful design guide to determine a proper codeword length considering the memory and processing speed of the implementation

environment. As marked in the Fig. 3, in the case of 5-bit compacted codeword, the processing complexity of the algorithm is significantly reduced to 37% while the average memory usage is increased to 120%. The increase of 20% memory space is considered as reasonably marginal compared to the significant improvement in computational complexity.

4. Measurement and Results

The performance of the proposed Huffman decoding algorithm is evaluated by measuring the instruction cycles of the proposed Huffman decoder. The MPEG-2 AAC decoder, including the Huffman decoding block, is implemented on a typical 32-bit DSP processor. The number of instruction cycles for 'load', 'branch' and the other arithmetic instructions are counted as five (5), six (6), and one (1), respectively, according to the specifications of the DSP processor.

In Fig. 4, the measured computational complexities by varying the bit length of compacted codeword from 4 to 8 bits are plotted together with those of the analytically derived Eqs. (1) and (2). Where, the computational complexity of the binary tree search is used as a comparison reference of competing method. It is confirmed that the analytically derived complexities agree well with those of the experimentally measured results.

The relative performance is defined as Eq. (3) and is used in the verification of the analytical derivation.

$$\text{Relative computational complexity} = \frac{\text{total number of instruction cycles of proposed method}}{\text{total number of instruction cycles of competing method}} \quad (3)$$

By using total of 63 MPEG-2 AAC files, the proposed method is tested and its performance results are compared with those of the other three conventional methods, i.e., the sequential search method, the binary tree search method,

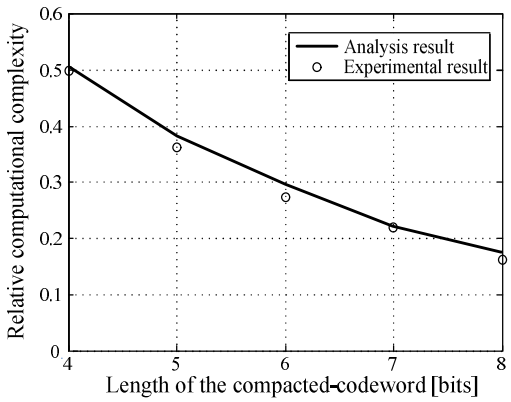

Fig. 4. Comparison of the measured computational complexities with those of the analytically derived results.

Table 3. Comparisons of the computational complexity of the proposed method with the three conventional Huffman decoding methods. Those average computational complexities are normalized by the complexity of the proposed method.

	Proposed method	Sequential search method	Binary search method	Hybrid search method
Average computational complexity (normalized)	1	6.135	2.756	1.863
Relative complexity of the proposed method (%)	-	16.3%	36.3%	53.7%

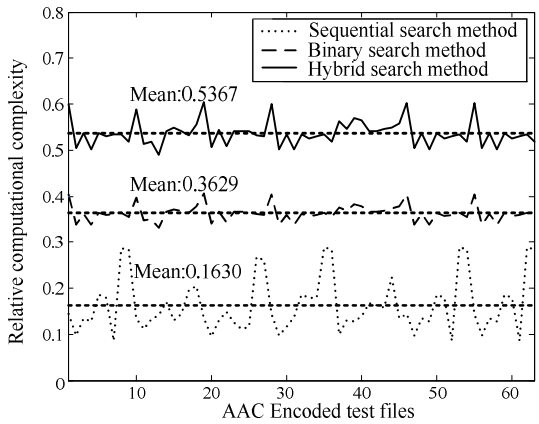

Fig. 5. Comparisons of the proposed method's computational complexity to those of the other typical Huffman decoding methods. (The length of the compacted codeword is 5 bits.)

and the Hybrid search method. The MPEG-2 AAC files are encoded utilizing the total twelve Huffman codebooks. Where, the length of the compacted word used in the comparison is five (5) bits.

The measured computational complexity of the proposed method is reduced to 16.3%, 36.3% and 53.67% in relative to those measured for the other three decoding methods as shown in Table 3. In the table, the average complexities of the tested algorithms normalized for that of the proposed method are shown together.

Fig. 5 shows the relative performances for each 63 MPEG-2 AAC files that are calculated for the three conventional decoding methods.

As shown in the simulation results, the average relative performance for the Binary search method agree with the analytical results obtained with the Eqs. (1) and (2). This result confirms that the proposed method reduces the processing complexity to 36.3% while the average memory usage is increased as much as only 20%. Also, the complexity of the proposed algorithm is improved to be less than 54% even when it is compared with that of the Hybrid Huffman decoding algorithm, which is known as one of the most efficient methods.

5. Conclusion

This paper presents an MPEG-2 AAC Huffman decoder which is based on the construction of a compacted codeword table. Where each codeword may contain multiple number of Huffman codes. The table's direct memory look-up structure and the multiple codes allocating structure allow a significant improvement of searching efficiency. The average computational complexity of the proposed method measured with 63 MPEG-2 AAC test files shows only 54 % even when it is compared with that of the Hybrid Huffman decoding algorithm, which is known as one of the most efficient ones. The computational complexity of the proposed approach is also analytically derived and its trade-off relationship with the amount of memory usage is also provided. The proposed Huffman decoding method is expected to be utilized broadly especially for low power implementation of portable multimedia platforms.

Acknowledgements

This research was supported by the Korea National Research Foundation under Grant 20100786, and by the Chung-Ang University Excellent Student Scholarship.

References

[1] M.A. Watson and P. Buettner, "Design and implementation of AAC decoders", *IEEE Transactions on Consumer Electronics*, vol. 46, no. 3, pp. 819-824, Aug. 2000.

[2] K. Sayood, "Introduction to Data Compression", 1996, Morgan Kaufmann.

[3] R. Freking and K. Parhi, "Low-memory, fixed-latency Huffman encoder for unbounded-length codes", in Proc. 34th Asilomar Conf. Signals, Syst., Comput., vol. 2, pp. 1031-1034, Pacific Grove, CA, Nov. 2000.

[4] K. Chung and J. Wu, "Level-compressed Huffman decoding", *IEEE Trans. Commun.*, vol. 47, no. 1-, pp. 1455-1457, Oct. 1999.

[5] S. Ho and P. Law, "Efficient hardware decoding method for modified Huffman code", *Electron. Lett.*, vol. 27, no. 10, pp. 855-856, May 1991.

[6] Lee, Eun-Seo and Lee, Jae-Sik and Son, Kyou-Jung and Chang, Tae-Gyu, "Compacted codeword Huffman decoding method for MPEG-2 AAC decoder", in IEEE International Conference on Consumer Electronics (ICCE), 2013, pp. 478-479, 2013.

[7] J.S. Lee, J.H. Jeong, and T.G. Chang, "An Efficient Method of Huffman Decoding for MPEG-2 AAC and Its Performance Analysis", *IEEE Transactions on Speech and Audio Processing*, vol. 13, no. 6, pp 1206-1209, Nov. 2005.

[8] R. Hashemian, "Memory efficient and high-speed search Huffman coding", *IEEE Trans. Commun.*, vol. 43, pp. 2576-2581, Oct. 1995.

[9] R. Hashemian, "Condensed table of Huffman coding, a new approach to efficient decoding", *IEEE Trans. Commun.*, vol. 52, no. 1, pp. 6-8, Jan. 2004.

[10] T.-H. Tsai, C.-N. Liu and J.-H. Hung, "VLIW-aware software optimization of AAC decoder on parallel architecture core DSP (PACDSP) processor", *IEEE Transactions on Consumer Electronics*, vol. 54, no. 2, pp. 933-939, May 2008.

[11] T. Tsai, C. Liu, "Low-Power System Design for MPEG-2/4 AAC Audio Decoder Using Pure ASIC Approach", *IEEE Transactions on Circuits and Systems*, vol. 56, no. 1, Jan. 2009, pp. 144-155, 2009.

Eun-Seo Lee He received the B.S., M.S., and Ph.D degrees in electrical and electronics engineering from Chung-Ang University, Korea in 2003, 2005, and 2008 respectively. Since 2009, he has been a researcher at Electronics and Telecommunications Research Institute (ETRI), where he has worked on home network middleware especially device control and management. Also, he has been participated in standardization activities on ISO/IEC JTC1 SC6. His recent research interests are smart home appliances, device auto configuration, device control and management system.

Kyoung-Cheol Lee He received the B.S., M.S. and Ph.D. degrees from the Chung-Ang University, Seoul, Korea, in 1998, 2000, and 2004, respectively, all in electrical engineering. Since 2009, he has been a research engineer in Communication R&D Lab. at LIGNex1, Republic of Korea. His research interests include satellite communication system, satellite antenna control, and digital communication system.

Kyou-Jung Son He received his B.S. degrees in Electrical and Electronics Engineering from the Chung-Ang University, Seoul, Korea in 2012. He is currently pursuing the M.S. degree in Electrical and Electronics Engineering from the Chung-Ang University. His research interests are in the area of multimedia signal processing.

Seong-Pil Moon He received his B.S. and M.S. degrees in Electrical and Electronics Engineering from the Chung-Ang University, Seoul, Korea in 2010 and 2012, respectively. He is currently pursuing the Ph.D. degree in Electrical and Electronics Engineering from the Chung-Ang University. His research interests are in the area of adaptive signal processing, digital communications, and multimedia signal processing.

Tae-Gyu Chang He received the B.S. degree from the Seoul National University, Seoul, Korea, in 1979, and M.S. degree from Korea Advanced Institute of Science and Technology, Seoul, in 1981, and Ph.D. degree from University of Florida, Gainesville, in 1987, all in electrical engineering. From 1981 to 1984, he was with the Hyundai Engineering/ Electronics Inc., Seoul, as a Systems Design Engineer. From 1987 to 1990, he was a Faculty Member of Tennessee State University, Nashville, as a Research Assistant Professor at the Center of Excellence in Information Systems Engineering. In March 1990, he joined the faculty of the Chung-Ang University, Seoul, where he is currently a Professor at the Department of Electrical and Electronics Engineering. He is a Senior Member of the IEEE Signal Processing Society. His research interests include adaptive signal processing, multimedia signal processing and communications.