Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X
IMPACT FACTOR: 5.258

IJCSMC, Vol. 5, Issue. 3, March 2016, pg.540 – 549

Implementation of CAVLD Architecture Using Binary Tree Structures and Data Hiding for H.264/AVC Using CAVLC & Exp-Golomb Codeword Substitution

G. Sherlin Shobitha, Dept. Of ECE, SCETW, Hyderabad, India, Email: gshobitha@stanley.edu.in

K. Kishore Kumar, Dept. Of CSE, MCET, Hyderabad, India, Email: kishorkadari@gmail.com

Abstract - Data hiding is necessary in the encrypted video to maintain security and privacy. This paper presents, a novel scheme of data hiding in the encrypted version using CAVLC & Exp-Golomb codeword substitution and architecture to minimize memory space in CAVLD of H.264/AVC using Binary tree Structures. To solve the problem of decoding a great number of syntax elements based on look-up tables, an efficient decoding has been designed using binary tree structures. Experimental results have demonstrated the feasibility and efficiency of proposed scheme.

Keywords: H.264/AVC, CAVLD, Data Hiding, Codeword substitution

I. INTRODUCTION

With the increasing demands of providing video data security and privacy protection, data hiding in encrypted H.264/AVC videos will undoubtedly become popular in the near future. The ITU-T/ISO/IEC Joint Video Team [1] established a new, improved video coding standard known as H.264/AVC [2] in 2003. This standard aims at a wide range of applications such as multimedia short message, entertainment, videophone, storage, videoconference, HDTV broadcasting and Internet streaming. The capability of performing data hiding directly in encrypted H.264/AVC video streams would avoid the leakage of video content, which can help address the security and privacy concerns with cloud computing [3]. The proposed scheme can achieve excellent performance in the following three different prospects.

- The data hiding is performed directly in encrypted H.264/AVC video bitstream.
- The scheme can ensure both the format compliance and the strict file size preservation.
- The scheme can be applied to two different application scenarios by extracting the hidden data either from the encrypted video stream or from the decrypted video stream.

Context-adaptive variable-length decoding (CAVLD) is an inherently lossless compression technique. In H.264/AVC, it is used to decode residual, zig-zag order, blocks of transform coefficients.

The architecture designed in this work does uses less memory, aiming to save both hardware resources and power dissipation. The decoding of the syntax elements that used look-up tables was replaced by efficient tree structures.

This paper is organized as follows. Section II presents CAVLD architecture. Section III presents Binary tree structures. Section IV presents Data Embedding. Section V presents Results and Discussion. Section VI presents Conclusion. Section of this work.

II. CAVLD ARCHITECTURE

The CAVLD (Context Adaptive Variable Length Decoder) architecture (shown in Fig. 1) was designed through five main processing modules, one 32-bit buffer, one registers bank, one re-order module, router elements and supplementary modules. Each one of the processing modules is responsible for decoding one type of element, namely: Coeff_Token, Trailling Ones, Levels, Total_Zeros and Run_Before.

i) Coeff_token

To decode the coeff token, H.264 uses tree strcture depending on the number of non zero coefficients and trailing ones. The tree structure used to decode the coeff_token when nC=-1is shown in Fig 2. Based on the H.264/AVC CAVLD look-up tables, six binary trees were created to decode the coeff_token. When there are more than eight non zero coefficients in neighboring blocks, H.264 uses a fixed six bit coding. Depending on the nC parameter, corresponding total coefficient and trailing ones are detected.

ii) Trailing Ones

Number of trailing ones is calculated depending on the sign of the code. If code is '0' it is replaced with '+1'. If code is '1' it is replaced with '-1'. The sign of each trailing one is decoded using one bit from the bitstream. The trailing ones are inserted into the coefficient array. This stage is skipped if there are no trailing ones.

iii) Level

Level consists of a level prefix and a level suffix. The position of first one gives the level prefix. In earlier standards of H.264, the length of prefix was restricted to a maximum value of 15, but this is relaxed in recent amendments. In the proposed design, we only process 16 bits each time. This simplifies the interface of the input buffer, as all other stages require a maximum of 16 bits. We use multiple cycles in case prefix length is greater than 16. Since this is not a frequent case, there is no impact on throughput. The process to decode a level has a regular structure and it does not need tables. The process is carried out through the reading of bits from the input and a level value is generated according to a set of steps. A level code consists of a prefix and a suffix. The prefix is defined as a sequence of zeros that has its size determined by tam_prefix, until the first bit one. The suffix has a variable size, from 0 to 6 bits. The size of suffix is adapted as the level decoding process is performed, considering the previous levels magnitudes. After the prefix and suffix codes are read from input, the CodeNo is computed, as shown in eq (1).

$$\text{CodeNo} = ((\text{tam\_prefix} \times (2^{\text{table}})) + \text{suffix} \quad (1)$$

From CodeNo, it is possible to reach the level value. If the CodeNo is even the level is calculated by eq(2), otherwise it is calculated by eq(3).

$$\text{Level} = ((\text{CodeNo}+2)/2) \quad (2)$$

$$\text{Level} = -((\text{CodeNo} + 1)/2) \quad (3)$$

iv) Total_zeros

H.264 specifies 15 Total_zero binary trees, which is chosen depending on the number of nonzero coefficients.

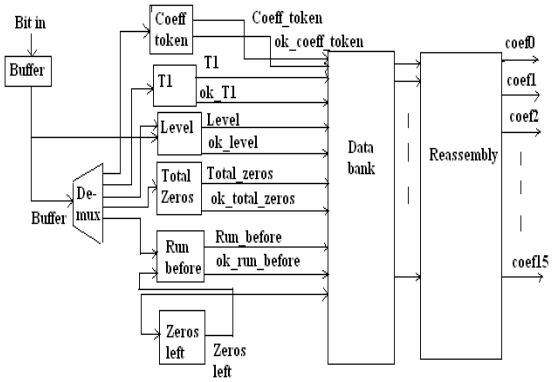

Fig. 1: Architecture of CAVLD

v) Run_before

H.264 specifies 7 Run_before binary trees. The number of zeros remaining determines the binary tree to be used. Out of 7 binary trees, 1 is used to decode run_before. 'zeros left' register is decremented by run_before. coeff register holds all previously decoded level values and trailing ones in the order they are decoded. Zeros have to be inserted between these coefficients according to run_before [4].Data bank consists of organized collection of data or information for storage purpose. Reassembly reorders the bitstream and sets in correct order. It is important to emphasize that the CAVLD architecture designed in this work uses less memory since the decoding of the syntax elements that used look-up tables were replaced by efficient binary tree structures. The architecture proposed and designed presents a high savings in hardware resources and memory accesses.

III. BINARY TREE STRUCTURES

The algorithm finds data by repeatedly dividing the number of ultimately accessible records in half until only one remains. Binary tree is searched from the root to the leaves as shown in the Fig 2. Each node of the tree has two paths: one to decode a bit '0' and other to decode a bit '1', such paths are traversed according to the bit stream input. When this search through the nodes reaches some of the leaves, this means that a symbol was decoded.

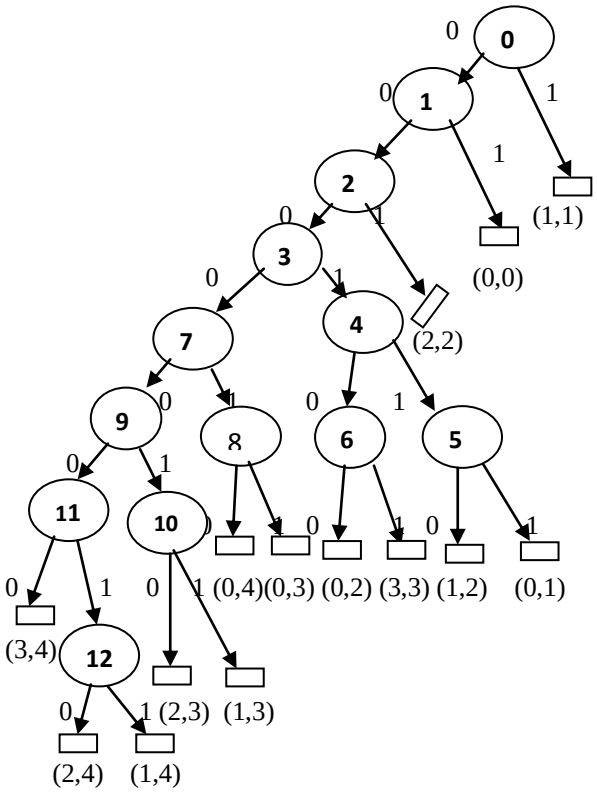

Fig. 2: Tree structure used to decode the coeff_token when nC=-1

In a tree, records are stored in locations called leaves. This name derives from the fact that records always exist at end points; there is nothing beyond them. Branch points are called nodes. The order of the tree is the number of branches (called children) per node. There are always two children per node in a binary tree, so the order is two. The number of trees in a binary tree is always a power of 2. The inputs of binary tree are only positive, single to double digit integers are allowed.

IV. DATA EMBEDDING

To hide data, data Embedding is needed. Although few methods have been proposed to embed data into H.264/AVC bitstream directly [5]–[6], however, these methods cannot be implemented in the encrypted domain. Fig. 3(a) & Fig. 3(b) shows how data is embedded and extracted at sender's end and at receiver's end.

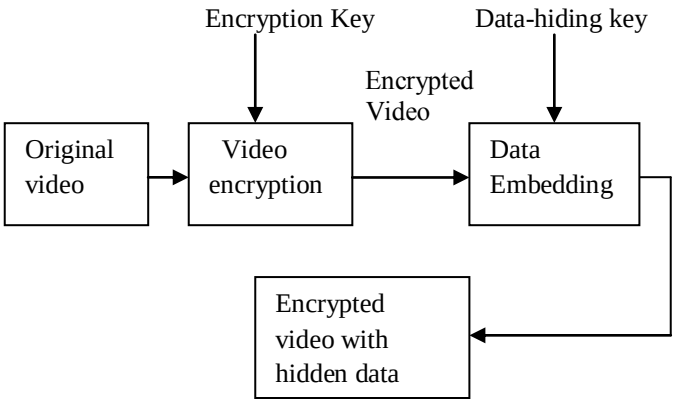

Fig. 3(a) Video Encryption and data Embedding at sender's end

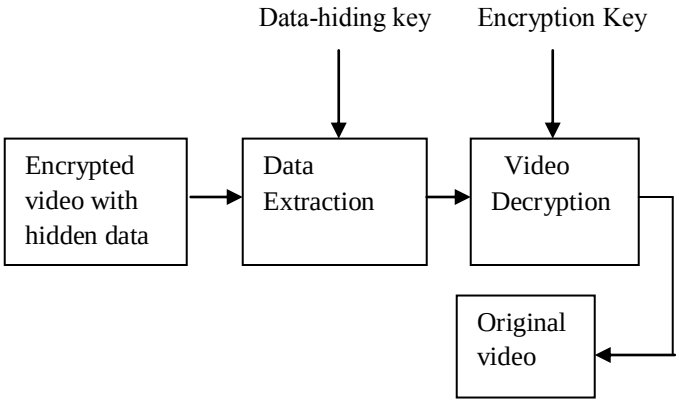

Fig.3(b) Video Decryption and data Extraction at receiver's end

In the encrypted bitstream of H.264/AVC, the proposed data embedding is accomplished by substituting eligible codewords of *Levels* in Table 1. Since the sign of *Levels* are encrypted, data hiding should not affect the sign of *Levels*. Besides, the codewords substitution should satisfy the following three limitations. First, the bitstream after codeword substituting must remain syntax compliance so that it can be decoded by standard decoder. Second, to keep the bit-rate unchanged, the substituted codeword should have the same size as the original codeword. Third, data hiding does cause visual degradation but the impact should be kept to minimum. That is, the embedded data after video decryption has to be invisible to a human observer. So the value of *Level* corresponding to the substituted codeword should keep close to the value of *Level* corresponding to the original codeword. In addition, the codewords of *Levels* within P-frames are used for data hiding, while the codewords of *Levels* in I-frames are remained unchanged. Because I-frame is the first frame in a group of pictures (GOPs), the error occurred in I-frame will be propagated to subsequent P-frames.

According to the analysis given above, we can see that there are no corresponding substituted codewords when *suffixLength* is equal to 0 or 1, as shown in Table 1. When *suffixLength* is equal to 0, we cannot find a pair of codewords with the same size.

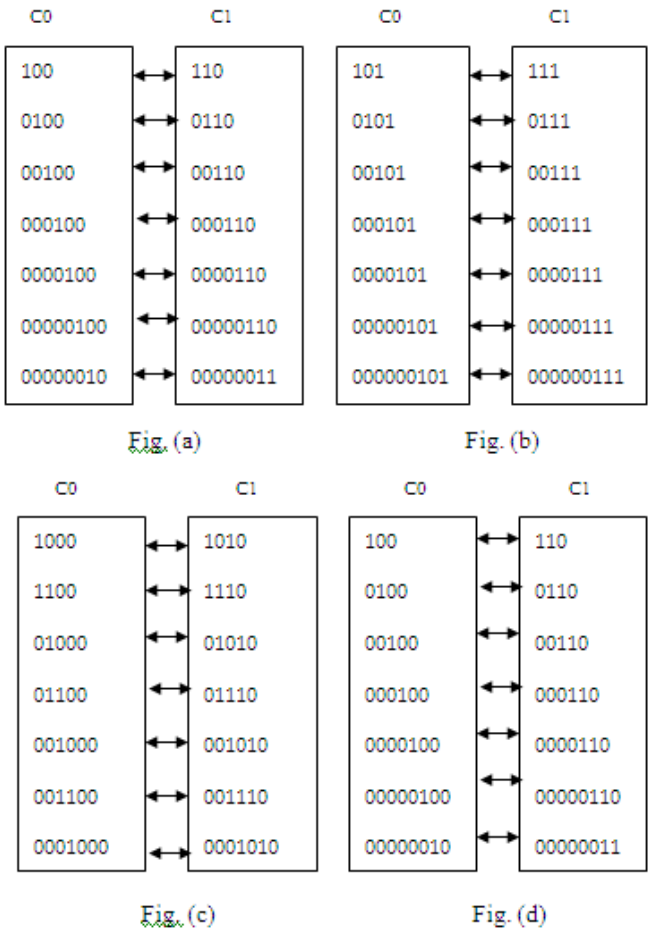

Fig. 4 CAVLC codeword mapping. (a) Suffix Length = 2 &Level > 0 (b) Suffix Length = 2 & Level < 0 (c) Suffix Length = 3 & Level > 0 (d) Suffix Length = 3 & Level < 0

When *suffixLength* is equal to 1, one codeword also cannot be substituted by another codeword with the same size, since this substitution would change the sign of *Level*. Then the codewords of *Levels* which *suffixLength* is 2 or 3 would be divided into two opposite codespaces denoted as *C*0and *C*1 as shown in Fig. 4. The codewords assigned in *C*0 and *C*1 are associated with binary hidden information "0" and "1".

Suppose the additional data that we want to embed is a binary sequence denoted as $B = \{b(i)|i = 1, 2, \cdots, L, b(i) \in \{0, 1\}\}$. Data hiding is performed directly in encrypted bit-stream through the following steps.

Step1. In order to enhance the security, the additional data is encrypted with the chaotic pseudo-random sequence $P = \{p(i)|i = 1, 2, \cdots, L, p(i) \in \{0, 1\}\}$[7] to generate the to-be-embedded sequence $W = \{w(i)|i = 1, 2, \cdots, L, w(i) \in \{0, 1\}\}$. The sequence P is generated by using logistic map with an initial value [7] i.e., data hiding key. It is very difficult for anyone who does not retain the data hiding key to recover the additional data.

Step2. The codewords of *Levels* are obtained by parsing the encrypted H.264/AVC bitstream.

Step3. If current codeword belongs to codespaces *C*0 or *C*1, the to-be-embedded data bit can be embedded by codeword substituting. Otherwise, the codeword is left unchanged. The detailed procedure of codeword substituting for data hiding is shown in Fig. 5. For example, when *Level* is positive 1 and its *sufflxLength* is 3, then its corresponding codeword is "1000" which belongs to *C0* as shown in Fig. 4(c). If the data bit "1" needs to be embedded, the codeword "1000" should be replaced with "1010". Otherwise, if the data bit "0" needs to embedded, the codeword "1000" will keep unchanged.

Step4. Choose the next codeword and then go to Step3 for data hiding. If there are no more data bits to be embedded, the embedding process is stopped.

Suppose the to-be-embedded data is "1001", the *CAVLC* codeword of *Level* parsing from H.264/AVC bitstream is "01 010 00100 00100 0001011 0000100" and the encryption stream is "10111", an example of data embedding based on codeword mapping is shown in Fig. 6.


```
Procedure
if (data bit= =0)
{
        if (the codeword belongs to C0)
           The codeword is unmodified;
        else if (the codeword belongs to C1)
                The codeword is replaced with the
                 corresponding codeword in C0.

}
else if (data bit= =1)
{
         if ( the codeword belongs to C1)
             The codeword is unmodified;
         else if(the codeword belongs to C0)
                The codeword is replaced with the
                 corresponding codeword in C1.
}
```

Fig. 5 The procedure of codeword mapping

Table 1: Levels and Corresponding Words

suffix Length	Level(>0)	Codeword	Level(<0)	Codeword
0	1	1	-1	01
	2	001	-2	0001
	3	00001	-3	000001
	4	0000001	-4	00000001
1	1	10	-1	11
	2	010	-2	011
	3	0010	-3	0011
	4	00010	-4	00011
	5	000010	-5	000011
	6	0000010	-6	0000011
	7	00000010	-7	00000011
	8	000000010	-8	000000011
2	1	100	-1	101
	2	110	-2	111
	3	0100	-3	0101
	4	0110	-4	0111
	5	00100	-5	00101
	6	00110	-6	00111
	7	000100	-7	000101
	8	000110	-8	000111
	9	0000100	-9	0000101
	10	0000110	-10	0000111
	11	00000100	-11	00000101
	12	00000110	-12	00000111
	13	000000100	-13	000000101
	14	000000110	-14	000000111
3	1	1000	-1	1001
	2	1010	-2	1011
	3	1100	-3	1101
	4	1110	-4	1111
	5	01000	-5	01001
	6	01010	-6	01011
	7	01100	-7	01101
	8	01110	-8	01111
	9	001000	-9	001001
	10	001010	-10	001011
	11	001100	-11	001101
	12	001110	-12	001111
	13	0001000	-13	0001001
	14	0001010	-14	0001011

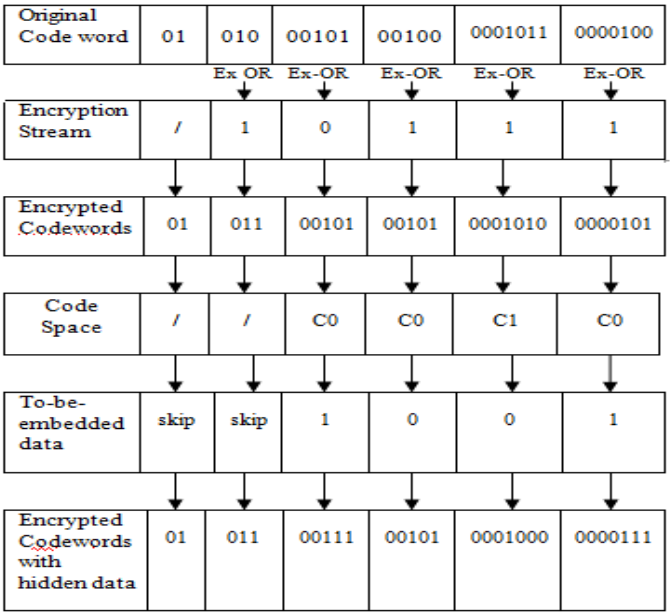

Fig. 6. An example of data embedding

The proposed scheme encrypts IPM (Intra Prediction Mode) Encryption, MVD (Motion Vector Difference) Encryption and Residual data encryption, which keeps security of the encrypted video. Stefen, Table (high motion sequences), Tempete, Mobile(high texure sequences), Hall and News(low motion sequences) are considered as six video sequences in QCIF format (176x144) at the rate of 30 frames/s. The GOP (Group Of Pictures) structure is "IPPPP: one I Frame followed by four P frames. The demonstration is shown in Figs. 7 and 8. An original frame from each video is depicted in Fig. 7, and their corresponding encrypted results are depicted in Fig. 8. The encrypted and decrypted video frames with hidden data are shown in Figs 9 and 10 respectively. In the experiments, no visible artifacts have been observed in all of the decrypted video frames with hidden data.

Fig. 7. Original video frames

Fig. 8. Encrypted video frames

Fig. 9. Encrypted video frames with hidden data

Fig.10 Decrypted Video frames with hidden data

In addition to code words of Levels, Exp-Golomb codewords of Motion Vector Difference (MVD) also can be used for data hiding. Data hiding and extraction procedure are the same as the previously described. For high motion sequences (such as *Stefan*, *Table*) and high texture sequences (such as *Tempete* and *Mobile*), the degradation in video quality caused by *MVD*'s Exp-Golomb codeword substituting is more serious than the previous *Level's* CAVLC codeword substituting method. Therefore, only for low motion sequences (such as Hall, News), Exp-Golomb codeword substituting is appropriate.

V. RESULTS AND DISCUSSION

The proposed CAVLD architecture performance with less memory occupation by using binary tree structures in terms of number of gates utilized are listed in the Table.4 and experimental results shown in the Table.2 and Table.3 for non-stego frames & stego (data hiding) using CAVLC codeword and Exp- Golomb codeword mapping has good privacy protection and data security, whereas the degradation in quality of the video caused by data hiding is quite small. Peak Signal to Noise Ratio (PSNR) is widely used objective video quality metric. A higher QP (Quantization Parmeter) will result in lower video quality. Data hiding payload can be assessed in kilobits per sec (kbits/s) [9]. The maximum payload capacity in each video is encoded with different QP values.

Table 2: Embedding Capacity, PSNR in Decrypted Videos using CAVLC codeword substitution

Sequence	QP	Maximum Capacity (kbits/s)	PSNR(dB)	
			Non-Stego	Stego
Stefan	24	17.80	39.60	38.33
	28	3.63	35.84	35.50
	32	0.57	31.68	31.62
Table	24	8.27	38.44	37.91
	28	3.45	35.15	34.87
	32	0.82	32.31	32.17
Tempete	24	7.89	38.68	38.16
	28	1.13	34.83	34.74
	32	0.14	30.88	30.87
Mobile	24	1.81	38.44	38.32
	28	0.22	34.51	34.59
	32	0.01	30.63	30.63
Hall	24	0.60	40.30	40.26
	28	0.13	37.92	37.90
	32	0.02	34.98	34.98
News	24	0.50	40.82	40.75
	28	0.11	37.78	37.76
	32	0.02	34.57	34.56

According to Table 2, for low motion sequence (such as *Hall*, *News*), the embedding capacity is low if only the codewords of Levels are used for data hiding. In this case, both the *CAVLC* codewords of *Levels* and the Exp-Golomb codewords of *MVD*s can be used for data hiding. . The test results based on the combination of the *CAVLC* codewords of *Levels* and the Exp-Golomb codewords of *MVD*s are also given in Table 3. Compared with Table 2, the embedding capacity is improved only for low motion sequences (such as Hall, News), but the video quality degradation is also negligible. So the combination is entirely feasible.

Table 3: Test results based on combination of CAVLC codeword and Exp-Golomb codeword mapping

Sequence	QP	Maximum Capacity (kbits/s)	PSNR(dB)	
			Non-Stego	Stego
Hall	24	1.17	40.33	38.97
	28	0.61	37.92	37.09
	32	0.40	34.98	34.62
News	24	1.27	40.82	37.85
	28	0.73	37.78	36.12
	32	0.50	34.57	33.84

Table 4: Comparison of gate count

Different methods	Gate Count
[8]	17,586
Proposed	2,926

VI. CONCLUSION

This work presents adoption of an efficient decoding of syntax elements through binary tree structures to solve the problem of memory usage which saves hardware resources consumption & power dissipation. Also data-hider can embed additional data into the encrypted bitstream using CAVLC codeword mapping & Exp- Golomb codeword substituting, even though he does not know the original video content. Since data hiding is completed entirely in the encrypted domain, this method can preserve the confidentiality of the content completely.

REFERENCES

[1] Joint Video Team of ITU-T, and ISO/IEC JTC 1: Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification (ITU-T Rec. H.264 or ISO/IEC 14496-10 AVC). JVT Document, JVT-G050r1, 2003.

[2] INTERNATIONAL TELECOMMUNICATION UNION.ITU-T Recommendation H.264 (03/05): Advanced Video Coding for Generic Audiovisual Services. 2005.

[3] W. J. Lu, A. Varna, and M. Wu, "Secure video processing: Problems and challenges," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, Prague, Czech Republic, May 2011, pp. 5856–5859.

[4] M. Alle, J. Biswas and S. K. Nandy, "High Performance VLSI Architecture Design for H.264 CAVLC Decoder," in Proc. ASAP, 2006, pp. 317–322.

[5] D. K.Zou and J.A.Bloom,"H.264 sream replacement watermarking with CABAC encoding", in Proc. IEEE ICME, Singapore, Jul. 2010, pp. 117-121.

[6] D. W. Xu and R. D. Wang, "Watermarking in H.264/AVC compressed domain using Exp-Golomb code words mapping," *Opt. Eng.*, vol. 50, no. 9, p. 097402, 2011.

[7] D. W. Xu, R. D. Wang, and J. C. Wang, "Prediction mode modulated data-hiding algorithm for H.264/AVC," *J. Real-Time Image Process.*, vol. 7, no. 4, pp. 205–214, 2012.

[8] Taisa Leal da Silva,Joao Alberto Vortmann"Low cost memoryless CAVLD architecture for H.264/AVC Decoder"IEEE Trans 2009.

[9] T.Shanableh. "Data hiding in MPEG video files using ultivariate regression and flexiblemacro block odering",IEEE Trans. nf,Forensics Security,vol.7,n0.2,455-464,Apr.2012.

