An Efficient VLSI Architecture of VLD for AVS HDTV Decoder

Bin Sheng, Wen Gao, *Member*, IEEE, Don Xie, and Di Wu

Abstract — *In this paper, we present a VLSI design of Variable Length Code Decoder for AVS video standard. As a co-processor of a RISC CPU, the design can decode Fixed Length Code, unsigned or signed k-th Exp-Golomb Code, and AVS 2-D Variable Length Code. Furthermore, it has a pre-processing submodule, which can perform Start Code Detection and De-stuffing for the input bitstream. The proposed architecture has been described in Verilog HDL, simulated with VCS digital simulator, and implemented using 0.18μm Artisan CMOS cells library by Synopsys Design Compiler. The circuit costs about 15k equivalent logic gates (not including 4kb on-chip SRAM). And the critical path is less than 6ns in the worst case. This design has been implemented in a single chip AVS HDTV decoder, AVS101, which can support real-time decoding for NTSC, PAL, 720p 60 frames/s or 1080i 60 fields/s programs. Although the architecture was originally designed for AVS video standard, it can be easily adapted to other coding standards*[1].

Index Terms — Variable Length Code Decoder, AVS, Video Decoder, VLSI

I. INTRODUCTION

Audio Video Coding Standard Workgroup of China is finalizing a national standard for the coding of video and audio. The new standard is known as AVS [1]. Comparing with other international coding standards, such as MPEG-2 [2], MPEG-4 [3] and H.264/AVC [4], the advantages of AVS include higher performance, lower complexity, lower implementation cost and licensing fees.

Variable Length Code (VLC), also called the Huffman Code [5], is an optimal code in theory with the average codeword length approximating the source entropy. Given the probability distribution of one set of source symbols, the idea is to assign shorter codewords to the more probable source symbols and longer codewords to the less frequent symbols, so that the average bit-rate is reduced. In real applications, the VLC can relax the transmission bandwidth and storage capacity requirements, especially for high-speed applications such as High Definition Televisions (HDTV), broadband video delivery, and real-time video storage and retrieval. Similar to many other video standards [2]-[4], AVS adopts VLC as entropy coding tool.

[1] This work was supported by the National High Technology Development 863 Program of China under Grant No. 2003AA1Z1290.

Bin Sheng is with the Department of Computer Science and Engineering, Harbin Institute of Technology, Harbin, China (e-mail: bsheng@jdl.ac.cn).

Wen Gao is with the Institute of Computing Technology, Chinese Academy of Sciences, China (e-mail: wgao@jdl.ac.cn).

Don Xie is with the Institute of Computing Technology, Chinese Academy of Sciences, China (e-mail: xdxie@jdl.ac.cn).

One of the most important modules in a real-time video decoder is the Variable Length Code Decoder (VLD). Because the length of codeword is variable, the codeword boundary can not be determined until the previous codeword has been decoded. This recursive dependency results in an upper limit on iteration speed and limits the possible throughput in a given VLSI technology and architectural approach.

There have been many researches on the architecture of VLD. These architectures can be divided into two classes. The first class is tree-based [6]. These architectures decode codewords according to a binary tree fixed in the decoding part. When receiving the data stream bit by bit, the source symbol can be obtained by traversing the tree from the root node to each leaf node. It is obvious that no codeword can be a prefix of any other codeword since the VLC codewords satisfy prefix condition. As the decoding process is bit by bit, this kind of architecture is not efficient for decoding one single bitstream. But the output throughput can be one codeword per cycle when multiple independent bitstreams share the same decoder.

The other class of architectures is lookup-table-based [7]-[10]. Although the codeword length is variable, the decoding process can still be implemented with a lookup table. The lookup table can be implemented using Programmable Logic Array (PLA), Random Access Memory (RAM), Read Only Memory (ROM) or hardwired logic. This kind of architecture can decode one codeword per cycle. Since the average bit per codeword is more than one bit, the architecture of the second class is more efficient and faster than that of the first one under the same transmission rate.

The entropy coding algorithm of AVS is different from that of other standards. It uses both *k*-th Exp-Golomb Code and 2-D VLC to improve the coding performance. Although some decoders of Exp-Golomb Code for H.264 standard have been proposed, few VLD designs which can decode both *k*-th Exp-Golomb Code and AVS 2-D VLC were reported. Li proposed a design of VLD, which can support AVS Standard Definition Televisions (SDTV) video decoding [11]. Adopting a combined parallel decode algorithm based on PLA mainly and serial decode algorithm, Li's design achieves a trade-off between area and speed for SDTV application.

In this paper, we propose a versatile VLD architecture, which can support real-time decoding for AVS HDTV video. The architecture can support the decoding of Fixed Length Code, unsigned or signed *k*-th Exp-Golomb Code, and AVS 2-D VLC. Furthermore, it has a bitstream pre-processing submodule, which can perform Start Code Detection and De-stuffing.

Di Wu is with the Institute of Computing Technology, Chinese Academy of Sciences, China (e-mail: dwu@jdl.ac.cn).

Contributed Paper
Manuscript received April 13, 2006

0098 3063/06/$20.00 © 2006 IEEE

The following sections are organized as follows. Section II briefly explains entropy coding algorithms adopted by AVS. The proposed architecture of VLD is described in Section III. Section IV presents simulation results and VLSI implementation. Conclusion and future work are given in the last section.

II. AVS Entropy Coding Algorithms

According to AVS video standard, syntax elements except coefficient residuals, such as MB type and MV (Motion Vector), are coded using either Fixed Length Code or 0th Exp-Golomb Code. Coefficient residuals are coded using 2-D VLC. At first, coefficient residuals are coded into 2-D data, (*level, run*), using Run Length Coding (RLC). Then each (*level, run*) pair is assigned to *CodeNumber* via lookup in a suitable code table, which is selected according to the context. Finally, the *CodeNumber* is coded again using *k*-th Exp-Golomb Code, where the rank *k* is context-based. Fixed Length Code can be directly decoded from the bitstream, since the code length is known. The decoding process of Exp-Golomb Code is much more complex, because the codeword length can not be determined before the codeword is found in the code book.

A. Exp-Golomb Code Structure

Exp-Golomb Code is a kind of variable length code, whose structure is very regular. The codeword consists of prefix and suffix. The number of codewords grows exponentially with the length of the suffix. An Exp-Golomb Code is constructed as following,

$$\overbrace{0\ldots0}^{m}1\overbrace{x_{n-1}x_{n-2}\ldots x_0}^{n}$$

The prefix bits consist of *m* leading "0's" and a "1", where $m \geq 0$. The last *n*-bit field is the suffix that carries information. For the *k*-th Exp-Golomb code, *n* is equal to *m* + *k*, and code length *L* can be represented with *m* and *k*, as in (1).

$$L = m + n + 1 = 2m + k + 1 \tag{1}$$

Table I describes the structure of *k*-th Exp-Golomb Codewords, where *INFO* is defined as

$$INFO = \sum_{i=0}^{n-1} x_i \cdot 2^i \tag{2}$$

B. Assignment of Syntax Element to CodeNumber

Except coefficient residuals and fixed length coded syntax elements, other syntax elements are all coded using 0th Exp-Golomb Codes. These syntax elements are either unsigned or signed. To the unsigned syntax element, the original value is equal to *CodeNumber*. And to the signed syntax element, the relationship between original value and *CodeNumber* is described as in (3), where *v* represents the original value.

$$CodeNumber = \begin{cases} 2|v| & (v < 0) \\ 2|v| - 1 & (v \geq 0) \end{cases} \tag{3}$$

Table II gives some examples of assigning signed syntax element to *CodeNumber*.

Table I
Structure of the *k*-th Exp-Golomb codeword

Rank, *k*	Codeword	Length, *L*	*CodeNumber*
0	1	1	$2^{(L-1)/2} - 1 + INFO$
	$01X_0$	3	
	$001X_1X_0$	5	
	...	...	
1	$1X_0$	2	$2^{L/2} - 2 + INFO$
	$01X_1X_0$	4	
	$001X_2X_1X_0$	6	
	...	...	
2	$1X_1X_0$	3	$2^{(L+1)/2} - 4 + INFO$
	$01X_2X_1X_0$	5	
	$001X_3X_2X_1X_0$	7	
	...	…	
3	$1X_2X_1X_0$	4	$2^{(L+2)/2} - 8 + INFO$
	$01X_3X_2X_1X_0$	6	
	$001X_4X_3X_2X_1X_0$	8	
	...	…	
…	…	…	…

Table II
Assignment of signed syntax element to *CodeNumber*

CodeNumber	Syntax Element
0	0
1	1
2	-1
3	2
4	-2
5	3
6	-3
…	…
N	$(-1)^{N+1}\lceil N/2 \rceil$

To the coefficient residuals, the coding algorithm is much more complex. The residuals are firstly coded into 2-D data (*level, run*), using RLC. Then each (*level, run*) pair is assigned to a *CodeNumber* via a suitable VLC table. At last, the *CodeNumber* is coded using 0th, 1st, 2nd or 3rd Exp-Golomb Codes according to context information. The suitable table is chosen from totally nineteen VLC tables according to the context, which are seven Luma Intra tables, seven Luma Inter tables and five Chroma tables. The VLC tables are built on a lot of context-based statistic information. The rank *k* of a coefficient residual codeword, which may be 0, 1, 2 or 3, is determined by the context-based selection of VLC table. The

first codeword of a block always selects Table 0, such as Luma Intra Table 0, Luma Inter Table 0, or Chroma Table 0 according to the type of current block (luma/chroma and intra/inter). The selection of VLC table for next codeword depends on the absolute value of the previous decoded *level*. If the value is larger than corresponding threshold, a table with bigger table number will be selected for the next codeword decoding. Otherwise, the current table will still be used to decode the next codeword.

In summary, AVS 2-D VLC is much more complex than traditional VLC. Especially, due to the result of code-table lookup is coded again using *k*-th Exp-Golomb Code where the selection of rank *k* is context based, the data path is much longer. The complexity directly increases the difficulty of VLSI implementation in terms of both speed and area.

III. The Proposed Architecture

The circuit of VLD is designed as a co-processor of an embedded RISC CPU. It can decode Fixed Length Code, unsigned or signed *k*-th Exp-Golomb Code, and AVS 2-D VLC. Furthermore, it has a pre-processing submodule, which can perform Start Code Detection and De-stuffing operations for the input bitstream.

A. The Overall Architecture

The overall architecture is shown in Fig. 1.

The functions of the major components are described as follows. RISC CPU communicates with VLD via the co-processor interface. Control Unit can receive commands from RISC CPU, control the operations of all submodules, and return the decoding result back to RISC CPU if necessary. R1, R0 are two 32-bit registers, in which the input bitstream is stored. The barrel shifter LBS can left-shift its 32-bit opening window from the 63-bit input to the next codeword according to the value of Acc, which records the number of decoded bits in R1 and accumulates the length of the current codeword. The barrel shifter RBS makes the next codeword, whose length is *code_length*, right-adjusted. According to AVS standard, the length of Variable Length Code is not more than 32. 16 bits are enough to contain the first "1" and the suffix bits of a valid codeword, as in (1). So the lower 16 bits of RBS output are stored in CW, a 16-bit register. Post-processing submodule performs the mapping from the value of CW to *CodeNumber* for both coefficient residuals and signed/unsigned syntax elements. When decoding a signed or unsigned codeword, the output of Post-processing should be returned to RISC CPU. Lookup Table contains the nineteen VLC tables implemented using hardwired logic. When decoding a coefficient residual codeword, (*level, run*) can be produced via table lookup among the VLC tables. *k* Generation contains an optimized lookup table for fast generation of *k*, which will be described in detail later. Table Number Generation can adaptively select VLC table number for next coefficient residual codeword according to context information, as mentioned in Section II-B. When decoding a Fixed Length Code, the Control Unit sends the code length to the accumulator Acc, and the 32-bit output of RBS is returned to RISC CPU. If the input bitstream is valid continuously, this VLD architecture can achieve its upper limit of decoding speed, which is one codeword per clock cycle.

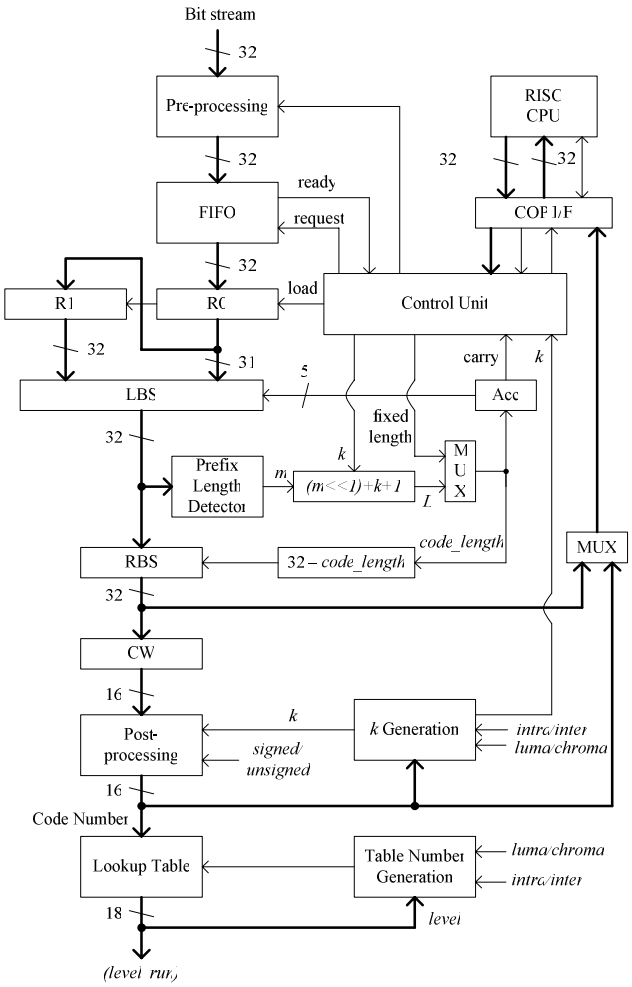

Fig. 1. Micro-architecture of the VLD

B. Pre-processing Submodule

The Pre-processing submodule shown in Fig. 2 performs de-stuffing for the input bitstream.

De-stuffing is the process of removing the stuffing bits, which are inserted into the bitstream to prevent Start Code emulation in the encoder. Start Codes are used to denote some special positions in the bitstream via their eight special information bits, such as picture header, slice header and so on. Start Code is defined as four byte-aligned bytes in the bitstream, such as 0x000001B0, which consists of twenty-three bits of "0", one bit of "1", and eight special information bits "0xB0". In some cases, pseudo Start Code may appear during encoding, which consists of twenty-three bits of "0", one bit of "1" and eight arbitrary binary bits too, just like real Start Code. Not really denoting position information, pseudo Start Code may lead decoder to find wrong position of the Start Code in

the bitstream and even make decoding failed. To prevent the emulation, AVS standard stuffs two bits of "10", so-called stuffing bits, after any twenty-two bits of "0" which are not bits of a real Start Code. For example, twenty-four bits string "0000 0000 0000 0000 0000 xxxx" will be stuffed to twenty-six bits string "0000 0000 0000 0000 0000 10xx xx". So in the decoder, the stuffed bits should be removed before decoding process. The Pre-processing submodule can detect and remove stuffing bits from the bitstream. Its input is 32-bit wide, and de-stuffed results are concatenated into 32-bit words again for output.

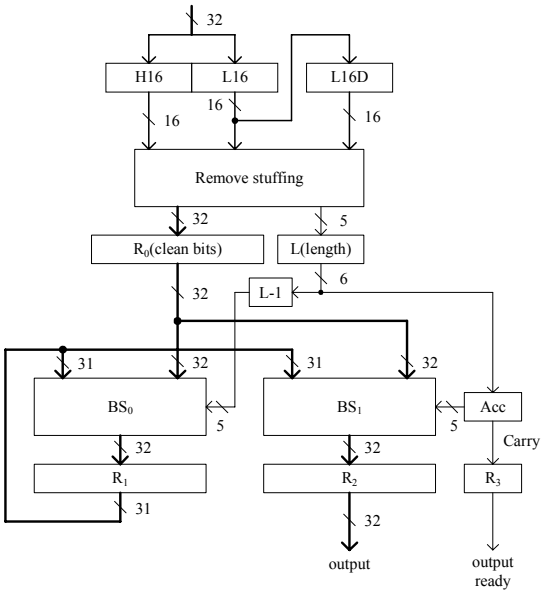

Fig. 2. Micro-architecture of Pre-processing submodule

The functions of some major circuit components in Fig. 2 are described as follows. The 32-bit input data are respectively stored into two 16-bit registers, H16 and L16. The 48 bits data from H16, L16 and L16D (a 16-bit register, in which stores the last value of L16), are sent into Remove Stuffing module, where stuffing bits are detected and removed. Register R0 and register L store the outputs of Remove Stuffing, which are clean bits and their length respectively. The clean bits in R0 are left-adjusted and stuffed with 0's on the right if necessary. Register R1 stores the concatenated previous bits which have not yet been output. R2 is the output register. Acc is an accumulator, which records the number of remainder bits in R1 and accumulates the length of the current input bits. Carry is latched in R3 and indicates whether the 32-bit output in R2 is available.

BS0 and BS1 both provide 32-bit wide windows on their 63 input bits. BS0 is controlled by the value of L-1 and left-shifts the input clean bits from R0 into R1, so that the rightmost bit of R1 is the last bit of the input clean bits. Consequently, the bits stored in R1 are concatenated with the next left-adjusted input bits via BS1. BS1 is controlled by the number of remainder bits in R1, which is recorded by Acc. If the length sum of remainder bits and current input bits is more than 32, the output of BS1 are just the first 32 bits, which are ready for output. At the same time, a carry signal will be sent out to denote that the output is ready.

C. Prefix Length Detector

The first task for decoding a variable length codeword is to determine its code length. If the rank k is given, the code length L can be determined by m, as in (1). The Prefix Length Detector is used to determine the prefix length m of an Exp-Golomb Code. Because the length of Variable Length Code supported by AVS standard is not more than 32, m is not bigger than 15. So, the higher 16 bits of the LBS output must contain the prefix zeros and the middle one of a valid codeword. The micro-architecture of Prefix Length Detector is described as Fig. 3.

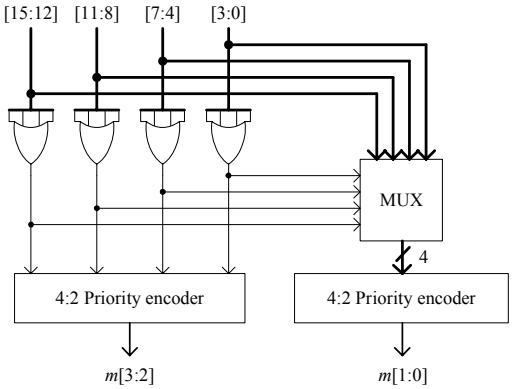

Fig. 3. Micro-architecture of Prefix Length Detector

D. Post-processing Submodule

The main function of Post-processing module is to translate Codeword to *CodeNumber*. Furthermore, for the signed or unsigned syntax elements, *CodeNumber* should be decoded once more to get the original values.

According to Table I, *CodeNumber* can be derived from Codeword directly, as in

$$CodeNumber = Codeword - 2^k \tag{4}$$

According to Section II-B, for unsigned syntax elements,

$$v = CodeNumber \tag{5}$$

Whereas for signed syntax elements,

$$v = \begin{cases} CodeNumber >> 1 & (CodeNumber[0] = 0) \\ -\left((CodeNumber >> 1) + 1\right) & (CodeNumber[0] = 1) \end{cases} \tag{6}$$

Here, *CodeNumber*[0] denotes the lowest bit of binary *CodeNumber*.

The functions mentioned in (4), (5) and (6) are not complex and can be easily implemented by hardware circuits.

E. Critical Path Optimization

As mentioned in Section II-B, during the decoding of coefficient residuals, the length of next codeword depends on

the rank *k* of next Exp-Golomb Code, which can be derived from the selection of VLC tables. Four steps are needed to produce *k*. At first, current codeword is Exp-Golomb decoded to *CodeNumber*. Then the residual level is produced via table lookup with *CodeNumber*. Thirdly, the VLC table for next residual codeword is selected, using the residual *level* and related context information. At last, *k* is derived from the selected table number.

To achieve high decoding speed for coefficient residuals, which is decoding one residual per clock cycle, we have to produce *k* for next codeword in the same cycle with the decoding of current codeword. Thus the serial four steps become the critical path of the design. In order to shorten the critical path, we optimized Step 3 and 4 together. *k* Generation contains an optimized lookup table for fast generation of *k*, which is much smaller than VLC Lookup Table. Taking advantage of this optimization, the critical path is reduced from 6.9ns to 6ns in the worst case.

IV. Simulation Results

We have described the architecture mentioned above in Verilog HDL at RTL level, which is synthesizable. According to AVS Reference Model [12], C-program models of RISC CPU and VLD are both developed to generate test vectors for VCS digital simulator. By testing with many corner case bitstreams, over 300 frames per bitstream (including NTSC, PAL, 720p, and 1080i), VCS simulation results show that the function of the Verilog design is completely correct.

The verified RTL design is synthesized using 0.18μm Artisan CMOS cells library by Synopsys Design Compiler. The whole circuit costs about 15k equivalent logic gates (not including a 4kb on-chip SRAM), and the critical path is less than 6ns in the worst case. Table III shows the comparison of synthesized results between Li's design in [11] and ours.

Table III
Comparison of Synthesized Results

	Li's Architecture	Our Architecture
Technology	0.18μm Artisan CMOS	0.18μm Artisan CMOS
Critical path	10 ns	6 ns
Working frequency	100MHz	148.5MHz
Gate count	13.6k	15k
Decoding speed	Less than 1 code per cycle	1 code per cycle
Capacity	SDTV	HDTV (720p 60 frames/s or 1080i 60 fields/s)

The proposed VLD architecture has been implemented in AVS101, which is the first single chip HDTV decoder for AVS video and audio. The decoder chip is physically implemented on a 6-mental 0.18μm Artisan CMOS technology. The area of the chip is 6.9mm by 6.9mm and the clock cycle is 6.7ns in the worst case. The power consumption is about 2.0-2.5 watt. At 148.5MHz working frequency, the decoder chip can support real-time decoding of NTSC, PAL or HDTV (720p 60 frames/s or 1080i 60 fields/s) bitstream. Fig.4 shows the chip photograph.

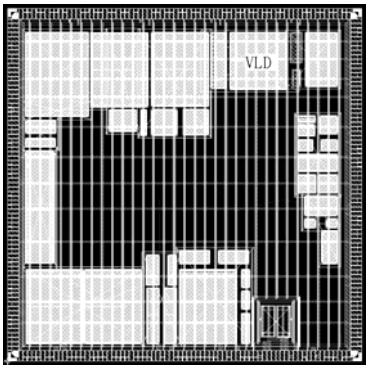

Fig. 4. Chip photograph

V. Conclusion and Future Work

In this paper, we present a versatile VLD design for AVS, which has been implemented in a single chip AVS HDTV real-time decoder. Under the control of an embedded RSIC CPU in the decoder chip, the design can decode Fixed Length Code, unsigned or signed *k*-th Exp-Golomb Code, and AVS 2-D VLC. Moreover, the design can perform Start Code Detection and De-stuffing operations. This VLD architecture can be easily adapted to other coding standards, though it was originally designed for AVS.

Our future work includes supporting real-time decoding for multiple AVS bitstreams, furthermore supporting multiple standards, such as MPEG-2, H.264/AVC. Moreover, we should reduce circuit area and power consumption for mobile applications.

References

[1] Audio Video Coding Standard Workgroup of China (AVS), Advanced Coding of Audio and Video - Part 2: Video, December 2004.
[2] ISO/IEC IS 13818, General Coding of Moving Picture and Associated Audio Information, 1994.
[3] Information technology – Coding of audio-visual objects – Part 2: Visual (ISO/IEC FCD 14496), July 2001.
[4] Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification (ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC), May 2003
[5] A. Huffman, "A method for the construction of minimum redundancy codes," Proc. of IRE, vol. 40, pp. 1098-1101, Sept. 1952.
[6] A. Mukherjee, N. Rangnanthan, J.W. Flieder, and T. Acharya, "MARVLE: a VLSI chip for data compression using tree-based codes," IEEE Trans. on VLSI System, pp. 203-214, 1993.
[7] S. M. Lei, M. R. Sun, K. Ramachandran, and S. Palaniraj, "VLSI implementation of an entropy coder and decoder for advanced TV applications," Proc. of ISCAS, pp. 3030-3033, 1990.
[8] Z. Aspar, Z. M. Yusof, and I. Suleiman, "Parallel Huffman decoder with an optimized lookup table option on FPGA," IEEE Proc. of TENCON, pp73-76, 2000.

[9] K. Y. Min, J.W. Chong, "A memory-efficient VLC decoder architecture for MPEG-2 application," IEEE Workshop on Signal Processing Systems, pp. 43-49, 2000.
[10] S. B. Choi, M. H. Lee, "High speed pattern matching for a fast Huffman decoder," IEEE Trans. on Consumer Electronics, 41, 2, pp. 97-103, 1995.
[11] D. Li, L. Yu, and J. Dong, "A decoder architecture for advanced video coding standard," SPIE Proc. of Visual Communications and Image Processing, pp. 2299-2308, 2005.
[12] AVS 1.0 RM5, December 2004.

Bin Sheng is a Ph.D. candidate at Harbin Institute of Technology. He received his MS degree in Computer Science from Harbin Institute of Technology, China, 2001. His research interests include: VLSI design, image processing, multimedia data compression.

Wen Gao (M'99) received the M.S. degree and the Ph.D. degree in computer science from Harbin Institute of Technology, Harbin, China, in 1985 and 1988, respectively, and the Ph.D. degree in electronics engineering from the University of Tokyo, Tokyo, Japan, in 1991. He was a Research Fellow with the Institute of Medical Electronics Engineering, University of Tokyo, in 1992, and a Visiting Professor with the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, in 1993. From 1994 to 1995, he was a Visiting Professor with the Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge. Currently, he is the Director of the Joint R&D Lab (JDL) for Advanced Computing and Communication, Chinese Academy of Sciences, a Professor with the Institute of Computing Technology, a Professor of computer science with the Harbin Institute of Technology, and an honor Professor of computer science at City University of Hong Kong. He has published seven books and over 200 scientific papers. His research interests are in the areas of signal processing, image and video communication, computer vision, and artificial intelligence. Dr. Gao chairs the Audio Video coding Standard (AVS) workgroup of China. He is the head of the Chinese National Delegation to MPEG working group (ISO/SC29/WG11). He is also the Editor-in-Chief of the Chinese Journal of Computers and the general Co-Chair of the IEEE International Conference on Multi-model Interface in 2002.

Don Xie received his Ph.D. degree in Electrical Engineering, University of Rochester, USA. He was a Senior Scientist at Eastman Kodak Company, New York, USA, from 1994 to 1997; a Principal Scientist at Broadcom Corporation, California, USA, from 1997 to 2002. Now he is a Researcher of Institute of Computing Technology, Chinese Academy of Sciences. His research interests include multimedia SoC design, embedded system. He has 23 U.S. Patents.

Di Wu is a Ph.D. at Harbin Institute of Technology. He received his MS degree in Computer Science from Harbin Institute of Technology, China, 1998. His research interests include: computer architecture, video compression, VLSI design.