An Adaptive Multialphabet Arithmetic Coding for Video Compression

Meng-Han Hsieh and Che-Ho Wei

Abstract—**In this paper, the hardware implementation issues for an adaptive multialphabet arithmetic coder are discussed. A simple weighted history model is proposed to encode the video data. This model uses a weighted finite buffer to model the cumulative density function of the arithmetic coder. The performance of the weighted history model is evaluated together with several other well-known models. To access, search, and update the cumulative frequencies corresponding to model symbols in real time, we present a low complexity multibase cumulative occurrence array structure that can offer the probability information for high-speed encoding and decoding. For the application in video compression, the multialphabet arithmetic coding with weighted history model can be a good choice as the variable length coding of the video symbols.**

Index Terms— **Data compression, multialphabet arithmetic coding, symbol probability models.**

I. Introduction

FOR many years, the Huffman coding technique has been regarded as an optimal entropy coding for data compression provided that the symbol statistics are known in advance [1]. However, the efficiency of the Huffman code is limited to the integer representation of the assigned bits, and when processing texts where one symbol has a probability of occurrence approximating unity, the performance of Huffman encoding becomes poor. Although the entropy associated with such symbols is extremely low, each symbol must still be encoded as an individual value. Arithmetic coding [2], [3] removes this limitation by representing the symbol as an interval of real numbers in the range 0–1. Initially, the range of values for coding a text is the whole interval [0, 1). As the message becomes longer, the interval needed to represent it becomes smaller, and the number of bits needed to specify this interval increases. Sequential symbols of message reduce the size of the interval according to the symbol probabilities set up by the model. The more probable symbols lower the range by less than the less probable symbols and therefore add fewer bits to the message. The performance of arithmetic coding is optimal without the need for blocking of input data, and it promotes a clear distinction between the model for representing data and the encoding of information with respect to that model.

In this paper, a simple adaptive model for a multialphabet arithmetic coder and its hardware implementation are introduced. Section II describes the constraints of the modeling unit for arithmetic code, and a weighted history model is presented to overcome the impairments which arose from these constraints. The hardware implementation issues of the arithmetic coder with adaptive model are investigated in Section III.

Manuscript received April 15, 1996; revised December 22, 1997. This paper was recommended by Associate Editor W. Li. This work was supported by the National Science Council of the Republic of China under Grant NSC84-0414-E009-010.

The authors are with the Department of Electronic Engineering, National Chiao Tung University, Hsinchu 30010, Taiwan, R.O.C.

Publisher Item Identifier S 1051-8215(98)02649-4.

II. Models for Arithmetic Coding

The arithmetic coding implementation by Witten *et al.* [2] is used with a model providing a cumulative frequency array $cum\_freq[\,]$. The requirements on the cumulative array are that:

- $cum\_freq[i-1] \geq cum\_freq[i]$;
- an attempt is never made to encode a symbol i for which $cum\_freq[i-1] = cum\_freq[i]$;
- $cum\_freq[0] \leq Max\_frequency$.

Provided that these conditions are satisfied, the values in this array need not bear any relationship to the actual cumulative symbol frequencies in message. Therefore, if the frequencies are accurate, encodings will occupy less space.

The arithmetic coding technique can use any statistical model to estimate the probability characteristic of the incoming data, and the compression performance is dominated by the model used. More sophisticated models such as the finite-context Markov models [4] can achieve better compression, but the time consumption for updating the model is enormous. To reach the requirement for high-speed applications, not only the arithmetic coder itself but also the model used to estimate the statistics of the incoming data must be fast enough.

A simple adaptive model presented by Witten *et al.* [2] uses the instantaneous statistics of the symbols. At the beginning, all frequency counts are the same. As an additional symbol is coded, the distribution of the cumulative probabilities is updated. After N symbols with p possible occurrences are coded, the distribution is obtained from $N+p$ occurrences. For a large N and relatively small p, the distribution is very close to that of N and therefore the performance of the arithmetic coding approaches that of the entropy of the source.

A limited past history model has been introduced by Ghanbari [5] to improve the performance of the adaptive model presented above. This model uses a limited number of past symbols instead of the whole history to estimate the probability distribution. A buffer is used to store only M previous data with $M < N$. This buffer is constructed by a first-in–first-out shift register, which stores the M most recent symbols of the incoming data. When a new symbol is coded it is stored in the buffer, and therefore the $(M+1)$th previous symbol is shifted out from the buffer. The limited past history model modifies the cumulative distribution to the local statistics of the source, which can increase the efficiency of the arithmetic coder especially in compressing the image data.

The limited past history model takes a relatively large buffer to achieve its optimal compression performance. For example, the optimum value for buffer size of limited past history is about 2–4k when compressing an image file, hence the hardware complexity will increase accordingly. The large buffer size also has the adverse effect that the local statistics are more difficult to pick up.

A weighted history model has been proposed [6] to overcome these drawbacks of the limited past history model. Suppose that there are p possible occurrences and the alphabet used in arithmetic coding is defined as $S_1, \cdots, S_p$. The buffer size used in the limited past history model is M, and the occurrence of S_i in the buffer is represented by O_i for all index i lies between 1 and p. Adding all occurrence in the buffer, we obtain the buffer size as $M = O_1 + O_2 + O_3 + \cdots + O_p$, and the relative frequency of symbol S_i as $freq(S_i) = (O_i + 1)/(p + M)$. Therefore, the corresponding cumulative frequency of symbol S_i is $cum\_freq(S_i) = \sum_{k=1}^{i} freq(S_k)$. The major disadvantages of the limited past history model are caused by the requirement that the occurrence of each symbol is at least one for arithmetic coding. The limited past history model overestimates the probability of each symbol by $1/(p + M)$, and the total overhead probability is equal to $p/(p + M)$. When the buffer size M is small, the overhead probability is almost one. That is, the probability distribution obtained by the limited past history buffer is nearly invariant to occurrence in the history buffer, and the statistical property of the source data is not reflected by this model. To enforce the relations between the probability distribution and the occurrence in the buffer, we can simply induce a weight to the buffer. Therefore, the frequency of the ith symbol is $freq(S_i) = (O_i \cdot W + 1)/(p + M \cdot W)$. The total overhead probability of the weighted history model is $p/(p + M \cdot W)$, which is much smaller than that of the limited past history model, especially when the buffer size is small.

The performance of the arithmetic coding with weighted history model for various buffer sizes and various weights is investigated as shown in Fig. 1. Fig. 1(a) uses a picture from the video sequence "missa" as the source data. The limited past history model can be regarded as a special case of the weighted history model with a unity weight. From this figure, it can be seen that the weighted history model really outperforms the limited past history model, especially when the buffer size is small. Fig. 1(b) shows the performance of the weighted history model with source data from the coded data of the pyramid vector quantizer (VQ) [7]. This diagram shows that the weighted history model uses less buffer and has a better compression performance than that of the limited past history model.

From these two figures we see that if the weight is too large, the performance of the weighted history model will degrade. The reason is that the large weight will reduce the probability of the symbols that are not in current buffer, and if the next symbol is not in the history buffer, a long codeword will be assigned to represent this symbol. From our experiments, an appropriate weight for the weighted history model is in the range from 16–128. It should be noted that the limited past history model with infinite buffer size is basically a low-complexity adaptive model of [2]. In [16], good initial models and infinite buffer size are used as the probability model. The so-called syntax-based arithmetic coding (SAC) incorporates the switching of probability models for corresponding data. If the syntax of the data source is known, the local statistics will be exploited more. However, *a priori* knowledge of the data source has to be given, and thus restrict the applicable range of the syntax-based model. The adaptivity of the adaptive model shown in [2] can be further improved by decreasing the cumulative frequency limit. For a multialphabet arithmetic coder, the occurrence of each alphabet must be at least one. Consequently, the adaptive model with a lower cumulative frequency limit has large overhead probability when the alphabet size is large. A similar solution is to weight the occurrences, and the overhead probability will be reduced. However, decreasing the cumulative frequency limit will make the updating procedure of the cumulative frequency count more often, and thus decrease the coding speed.

To evaluate the performance of the weighted history model, some adaptive models are compared. The experimental results are shown in Fig. 2. There are five data sources used in the comparison of the compression ratio. The first one is a 256 gray-level image file from the video sequence called "Miss America" ("missa") and the size of this image is 360 $\times$ 288. The other sources are output data from vector quantization methods [7] containing the interframe and intraframe components. Two methods of VQ, the classified VQ and pyramid VQ, are used. The input data of these VQ methods are video sequences "Miss America" and "sales man." The Lempel–Ziv model shown in Fig. 2 is the LZC algorithm, which is used in the *UNIX compress* program [4]. This algorithm uses a dictionary technique rather than a statistical model to compress data. The adaptive model in Fig. 2 is written by Witten *et al.* [2]. The dependency model is written by Abrahamson [8]. This model reflects the accurate probability of each symbol's occurrence following the previously encoded source character. The Q-Coder has been implemented by Mitchell [9]. The adapter of the Q-Coder is a finite-state machine multiplexed to maintain a separate state record for each context. Symbol probabilities are updated only when renormalization occurs. In the weighted history model, weights and buffer sizes are carefully adjusted such that the denominator of the cumulative frequency is a power of two. Therefore, the division operations used in calculating the frequency can be represented by shift operations. Various weights, buffer sizes, and symbols used in arithmetic coding are evaluated.

The performances of various probability models shown in Fig. 2 are evaluated by the proportion remaining [4], which is calculated by dividing the output file size by the original file size. From this table, we can see that the weighted history model with weight = 16 and buffer size = 112 performs very well in all kinds of test sources, especially when coding the data after pyramid VQ.

From the experiments, we see that the performance of the arithmetic coding with the weighted history model is quite

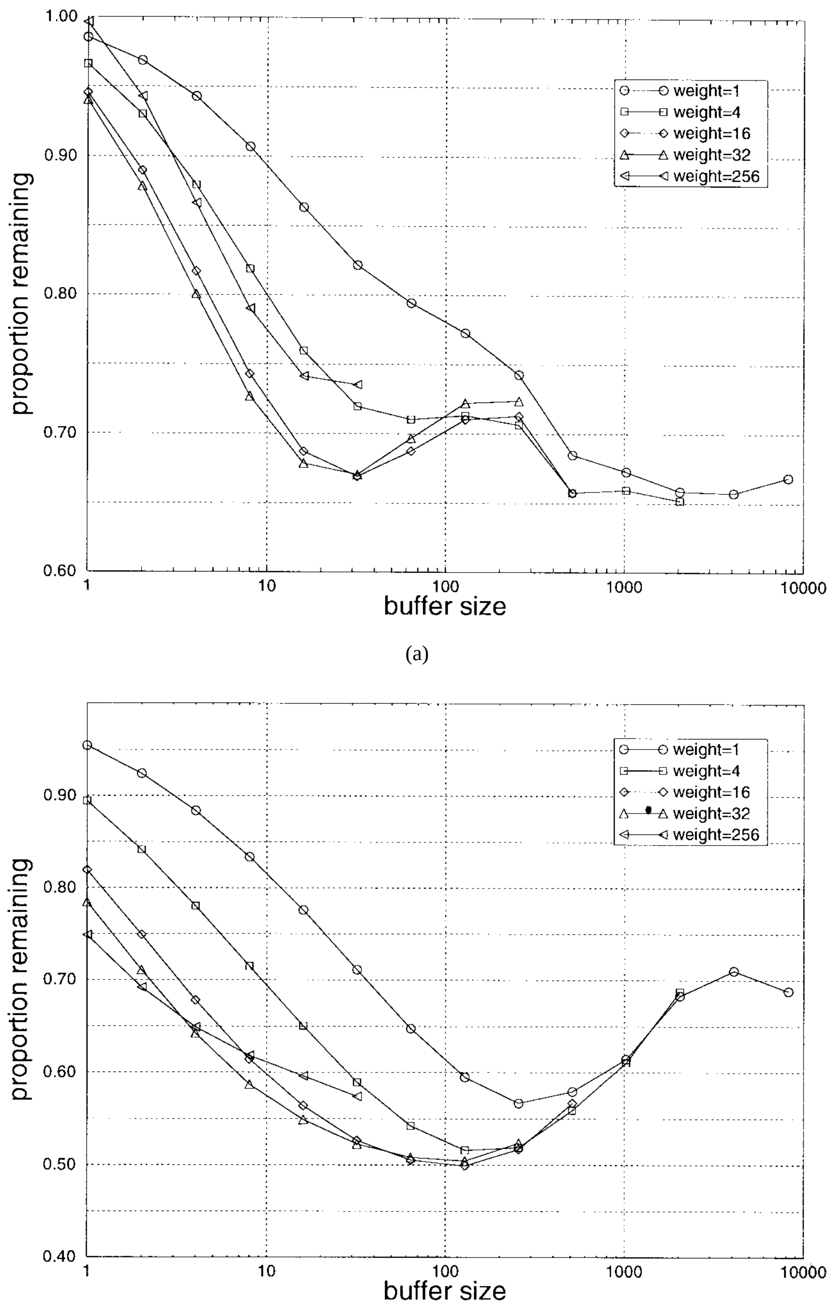

Fig. 1. Performance of arithmetic coding with weighted history model. Data source: (a) an image from “missa” and (b) coded data from pyramid VQ.

Models / Data sources	Size (bytes)	Lempel-Ziv model	Adaptive model	Dependency model	Q-Coder	Weighted history model			
						256 symbols		16 symbols	
						$M=112, W=16$	$M=24, W=32$	$M=127, W=16$	$M=30, W=8$
Image file	103,026	0.6486	0.6702	0.5429	1.0719	0.7109	0.6683	0.8434	0.7889
Classified VQ(1)	23,107	0.8924	0.7084	0.8361	1.0486	0.7619	0.8652	0.9108	0.9676
Classified VQ(2)	35,256	0.7876	0.6813	0.7979	0.9365	0.7263	0.7530	0.9396	1.0656
Pyramid VQ(1)	51,513	0.6188	0.6591	0.5786	0.8913	0.4986	0.5293	0.6768	0.6852
Pyramid VQ(2)	51,513	0.6121	0.7126	0.5524	0.9409	0.5241	0.5346	0.7115	0.7018

Fig. 2. Performance of probability models (proportion remaining).

good. Since the structure of the weighted history model is very simple, it is suitable for compressing image data in real time.

III. Hardware Implementation

The most well-known hardware implementation of arithmetic coding is the Q-Coder [10] reported by the IBM Research Center. The Q-Coder is an optimal binary arithmetic coder which uses a finite-state table to model the incoming data and performs well when the binary sources are used as input. A multiplication-free multialphabet arithmetic code devised by Rissanen and Mohiuddin [11] requires no multiplication or division, thus admitting a simple and fast hardware implementation. This code also accommodates the corresponding occurrence counts as well as the symbol probabilities. The code efficiency is typically in the range of 97–99%. Several recent publications also discuss the multiplication-free approximation of arithmetic coding and make some improvements [12]–[15]. A nonadaptive 256-symbol arithmetic encoder implemented by field programmable gate array (FPGA) technology has been presented in [14]. In this paper, a hardware structure for an arithmetic code based on [11] is introduced for its simpler decoder structure. The structure of an arithmetic codec can be divided in three parts: encoder, decoder, and modeling unit.

A. Encoder

The main parts of the arithmetic encoder can be divided into registers, a dynamic lookup table, adders, and the renormalizer. The architecture of the encoder is shown in Fig. 3. Detail operations of each part are described as follows.

- *Lookup table*
  Because of the approximation used in the encoder, a lookup table between input data and the modeling unit is used to keep the symbol with the largest count as the last model symbol. In general cases, the adaptive models are used in the arithmetic encoder and, therefore, the lookup table must be dynamic to coincide with the modeling unit.
- *Registers and adders*
  The range register A and the code register C are used to perform the encoding process. We update A and C simultaneously to increase the speed. In implementation we use a 16-b register for A and a 64-b register for C. The higher-order 48 b of C act as a 48-b guard register to prevent the carry-propagation problem. It can reduce the probability of carry-over problem and, therefore, speed up the encoding operation. The guard register can be implemented as a counter whose input is the carry of the adder connected to C. By this variation, we use a 16-b adder instead of a 64-b adder to update C. The bit-stuff scheme used in the Q-Coder can be applied to C if the carry propagates over the guard register.
- *Renormalizer*
  The renormalizer shown in Fig. 3 is used to bring A in the range [0.75, 1.5). The hardware structure of the renormalizer is shown in Fig. 4. To renormalize A, we first find the position of the first "1" of A from most significant bit (MSB) to least significant bit (LSB), and shift A left such that the MSB is the first "1" of the original A. That is, we first renormalize A into the range [1, 2). The notation A' is used to represent the renormalized A. After step 1, we adjust A' according to the first bit below MSB. If it is 1, we must shift A right by one bit to bring A in the range [0.75, 1.5). If this bit is zero, that means A' is already in the range [0.75, 1.5) and no further shift is needed. A cellular approach of bus arbitration logic shown in Fig. 5 is used to perform the operation of step 1 because of regularity and modularity. The basic cell of bus arbitration logic shown in Fig. 6 is designed with a pass transistor logic circuit to achieve high speed. The shifter part of this renormalizer is implemented by two barrel shifters, one is for A and the other is for C. Each input is passed through only one NMOS to obtain the shifted result. After renormalization, the bits shifted out must be stored in the buffer. An extra programmable logic array (PLA) is needed to convert the control signal into shift count so that we can do buffer control.

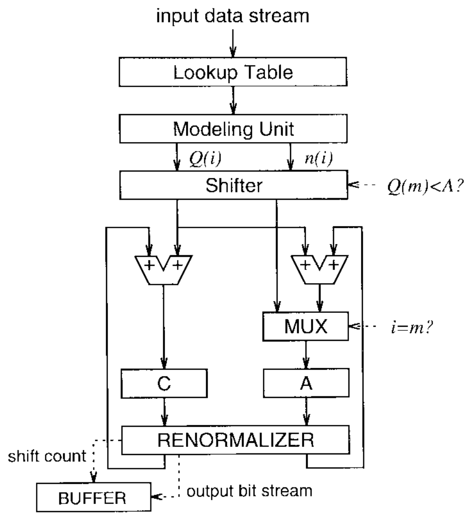

Fig. 3. Hardware architecture of multialphabet arithmetic encoder.

B. Decoder

Fig. 7 shows the architecture of the decoder. The coded stream is stored in buffer and is sent to the renormalizer. The renormalizer of decoder is the same as that in encoder, but we filled the bits to be shifted in C with the incoming code stream rather than zeros. No guard register is needed in decoder, thus both A and C are stored in 16-b registers. The update rules for A and C in decoder are the same as those in encoder. From the decoding algorithm described in [11], we need to find the largest symbol j such that $2^{-\beta}\overline{Q}(j) \leq C$, and decode the symbol as the so-found largest symbol i. To implement this procedure in hardware, one comparator and one shifter are needed for each Q. To reduce the hardware complexity, we modify the decoding algorithm to find the largest symbol j such that $\overline{Q}(j) \leq 2^{\beta} \cdot C$, and decode the symbol as the so-found largest symbol i. That means we only need a comparator for each Q and one shifter is used to shift C before comparison is made. The notation C_{model} is used to

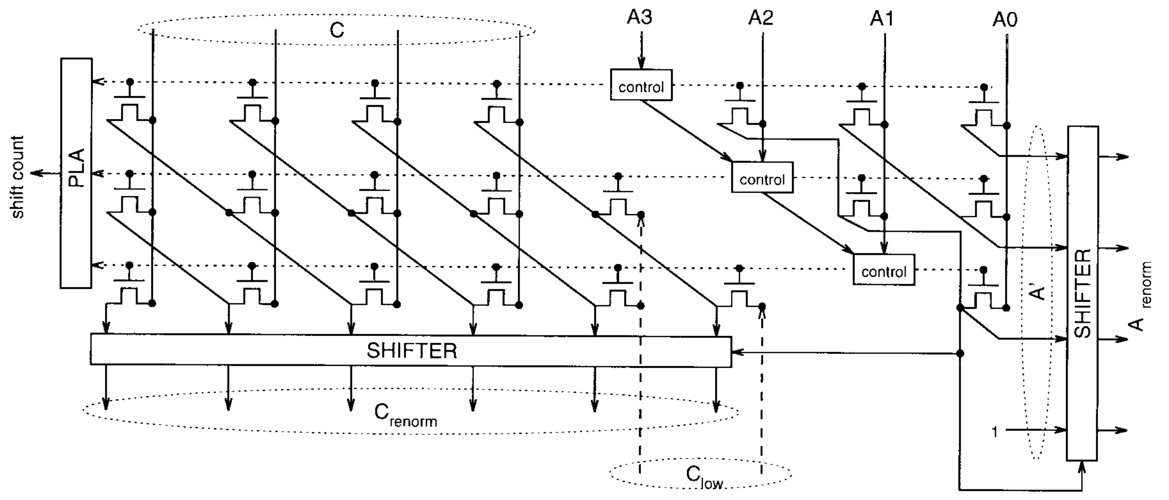

Fig. 4. Hardware architecture of renormalizer.

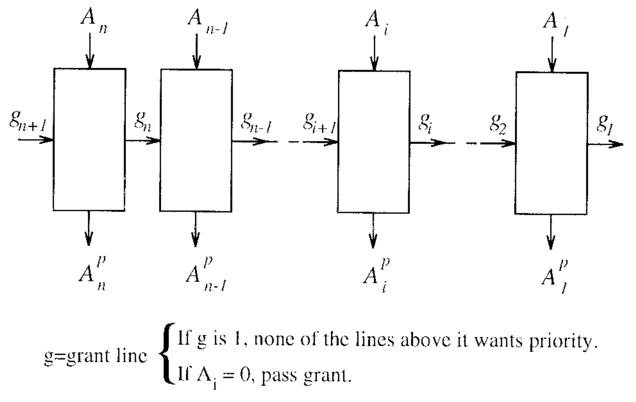

Fig. 5. Cellular approach for bus arbitration logic.

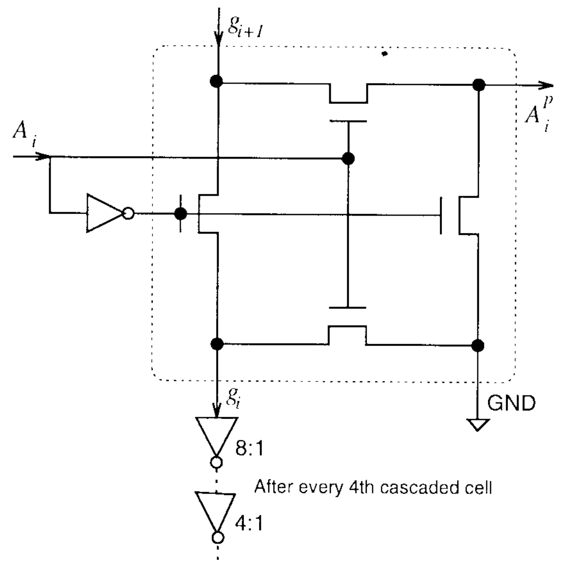

Fig. 6. Circuit of basic cell of bus arbitration logic.

represent $2^{\beta} \cdot C$. The comparators and cumulative probability registers are combined together to reduce the interconnection area. To obtain the model symbol i, a PLA is used to convert the output of comparators into a model symbol. The structure of the modeling unit without update structure is shown in Fig. 8. The update rule of the modeling unit in decoder must be the same as that in encoder.

C. Modeling Unit

The modeling unit of an arithmetic coder provides the statistical information to the coder. For encoding operation, the modeling unit acts as a lookup table and provides the probability and the cumulative probability of the current incoming symbol. For decoding operation, the modeling unit searches the symbol that has the largest cumulative frequency among all the symbols with cumulative frequencies smaller than current code point. The compression performance is highly dependent on the modeling unit, and the modeling unit is usually adaptive to gain better compression ratio. Most adaptive models devised are suitable for software implementations, but they usually are difficult to realize in hardware form. The more skillful methods such as the Markov methods or the Lempel–Ziv algorithms take more memory and increase the hardware complexity.

The weighted history model introduced in this paper uses a small history buffer to model the cumulative density function of the arithmetic coder and smaller counters to record the cumulative frequencies. Each occurrence in the history buffer is multiplied by a weight, thus all bits below the weight are not changed if the weight is an integer power of two. An example is shown in Fig. 9. The maximal cumulative frequency count is set to 512, and 256 symbols are used in the arithmetic coder. For a limited past history model or adaptive model of [2], a 9-b counter is needed for each symbol, whereas a weighted history model with weight $= 16$ only needs a 5-b counter for each symbol. The least significant four bits of the cumulative frequency counter are not changed while updating the model because the binary representation of the weight 16 is 10 000. A simple block diagram of the weighted history model is shown in Fig. 10. A shift register is used to store M most recent symbols. Initially, the history buffer is filled with symbols uniformly and the cumulative frequency

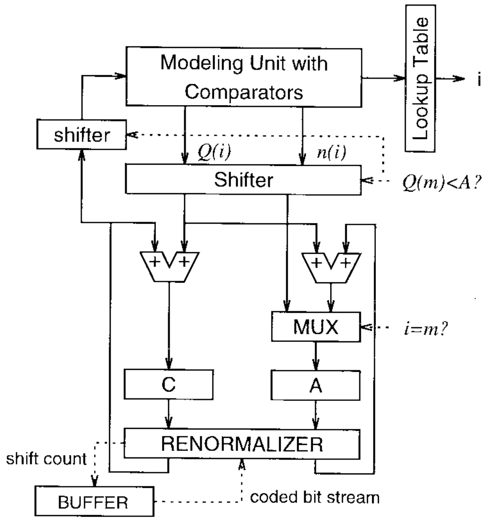

Fig. 7. Hardware architecture of multialphabet arithmetic decoder.

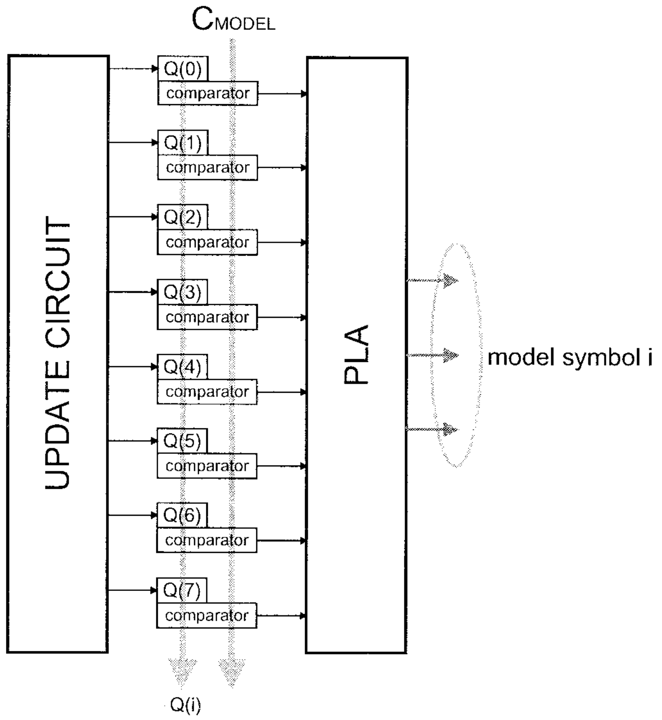

Fig. 8. Hardware structure of modeling unit.

counters are initialized according to the contents of the history buffer and the requirement of minimal occurrence. While a new symbol is encoded, this symbol is stored in the buffer and the $(M+1)$th previous symbol is shifted out. Only the first and the last symbols are used to update the cumulative frequency. The combinational logic gates in Fig. 10 first compare these two symbols. If the *current_symbol* is larger than the *previous_symbol*, all cumulative frequency counters corresponding to *previous_symbol* toward *current_symbol*-1 are counted up by one. This operation is equivalent to increasing the occurrence of *current_symbol* by a weight, and decreasing the occurrence of the *previous_symbol* by a weight. While the *present_symbol* is smaller than the *previous_symbol*, all cumulative frequency counters of *current_symbol* toward *previous_symbol*-1 are counted down by one. Thus, the cumulative frequency counter must be an up-and-down counter, and a parallel counter is recommended to increase the speed.

The complexity of modeling unit shown in Fig. 10 is acceptable while the alphabet size is several tens, but if we enlarge the alphabet size to 256, which is commonly used in digital storage and transmission systems, the control circuit becomes too complex for practical realization. As discussed in previous section, the compression performance is highly dependent on the alphabet size. Reducing the size of alphabet will degrade the compression performance.

The two main problems in implementing the modeling unit with large alphabet are the updating operation and the searching operation. The problem of the updating operation arises from the fact that the arithmetic coder uses the cumulative frequency of the symbol to generate its code point. If the occurrence of one symbol is changed, the cumulative frequency counts of symbols with alphabet order less than that symbol must be changed accordingly. Then all cumulative frequency registers must be accessible concurrently if we want to use parallel structure for implementation. An alternative structure is the serial structure, which updates a cumulative frequency register of one symbol at a time. The speed of the serial structure is much slower than the parallel one and has a drawback that the input rate is not constant. The decoding operation for multialphabet arithmetic coding involves the searching operation over the cumulative frequency registers. The searching operation has the same kind of problems that the updating operation has: too complex if parallel structure is used and too slow if serial structure is used. Therefore, a multibase cumulative frequency array is proposed in this paper to solve these two problems.

Our implementation focuses on the adaptive 256-symbol arithmetic coding. First, the 256 symbols are divided into 16 banks. Each bank has 16 registers to store the cumulative frequency information. The top register of each bank, called the "base" of the bank, stores the corresponding cumulative occurrence. The remaining 15 registers store the extent of the difference between the corresponding cumulative occurrences and the base, as shown in Fig. 11. The reason for storing the difference extent instead of the true value is shown in the updating procedure. Because the alphabet size is 256, we use 8 b to represent each symbol. When the occurrence of a symbol is changed, the four MSB's of this symbol are used to determine which bank this symbol belongs to, and the four LSB's are used to obtain the location of this symbol in the bank. In updating the cumulative occurrence array, we can update the bases of banks and the contents of the bank that the symbol belongs to simultaneously. Instead of the 8-input–256-output PLA used in the direct implementation of the modeling unit, two 4-input–16-output PLA's are used to implement the updating circuit of the multibase cumulative frequency array.

To see how the multibase cumulative occurrence array is applied to the model, we take two examples: the adaptive model used in [2] and the proposed weighted history model. The adaptive model described in [2] increases a symbol's occurrence by one when a symbol is coded. When the total

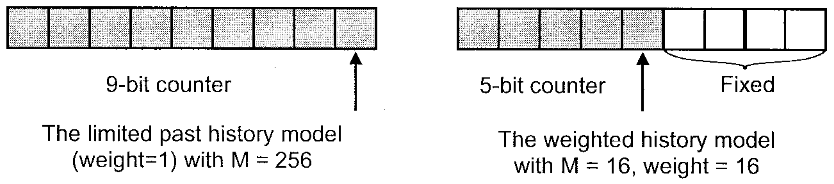
9-bit counter
The limited past history model (weight=1) with M = 256
5-bit counter
Fixed
The weighted history model with M = 16, weight = 16

Fig. 9. Relation between the cumulative frequency counter and the weight.

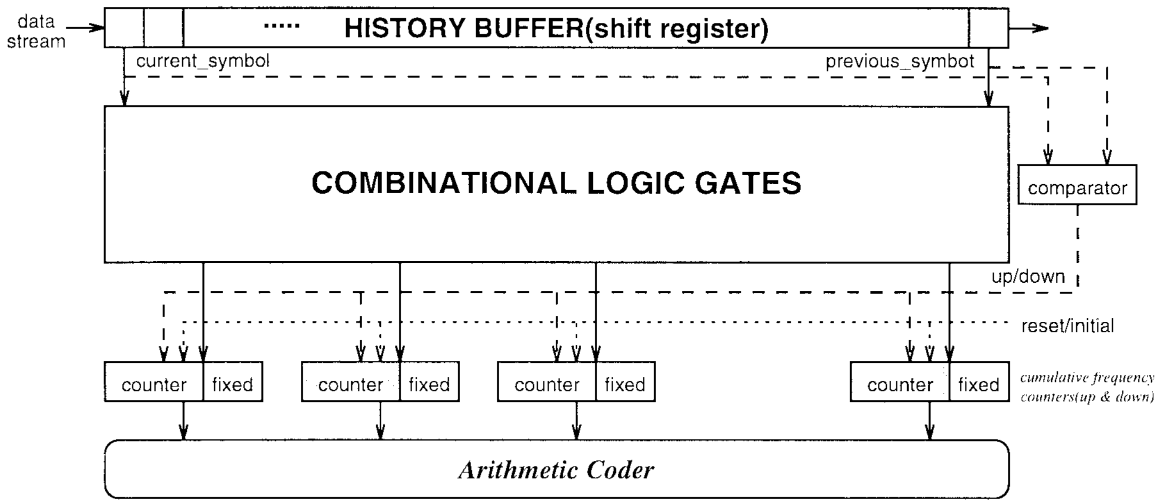

Fig. 10. Block diagram of weighted history model.

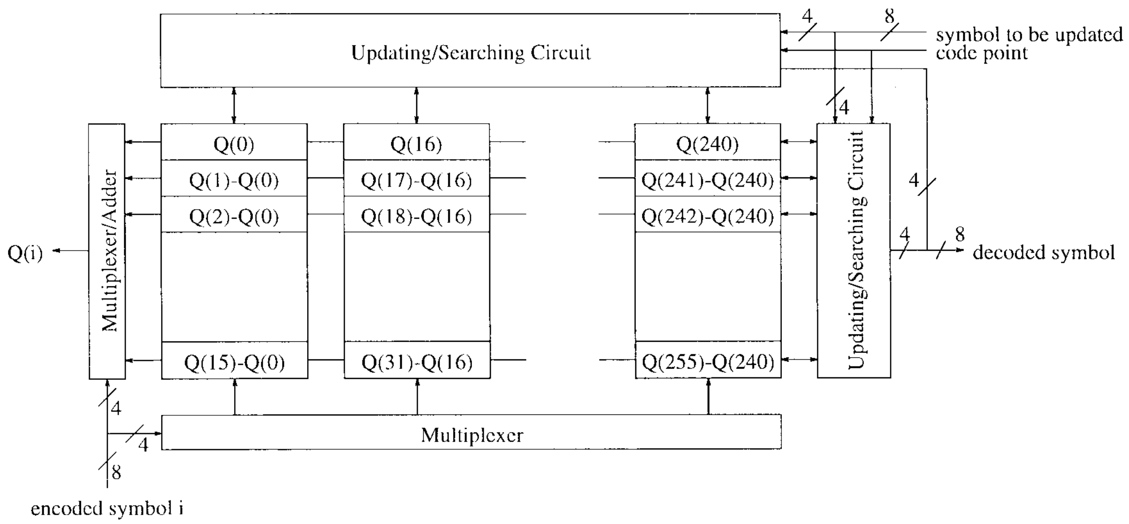

Fig. 11. Hardware structure of multibase cumulative frequency array.

occurrence equals the cumulative frequency limit, the occurrences of all symbols are halved. To implement the update circuit for this adaptive model, we need 32 counters and two PLA's. The two PLA's are used to determine which banks and which contents of the bank should be updated. Sixteen counters are used as the bases of banks, and the other 16 counters are used for the updating of the bank that the current symbol belongs to. To update the bank, we download the contents of the bank into counters and update these counters. The updated values are saved into the bank hereafter. Therefore, the contents of the bank can be implemented as RAM rather than counters. When the total occurrence achieves the cumulative frequency limit, the bases and the contents of each bank are halved. This action can be done serially because there are only 16 banks to be updated. If the maximal occurrence is as large as several tens of thousands, the overhead time is negligible. The cumulative frequency limit is dependent on the bit length of bases and the contents of banks. The performance of this model also depends on the maximal occurrence used [2]. A wordlength of 14 b is suitable for the cumulative frequency array. The performance of this model is shown in Fig. 2. The weighted history model avoids the halving problem in the adaptive model. However, because two symbols are used to update the cumulative frequency array, the updating operation of one symbol described above should be done twice. The word-length of the cumulative frequency array is 7 b if the weighted history model with history buffer length 112 and weight 16 is used. Another weighted history model shown in Fig. 2 for performance evaluation uses a history buffer of length 24 and weight of 32, and a 5-b memory is sufficient for each cumulative occurrence. The fixed part of the cumulative frequency of the weighted history model can be implemented by the LSB's of the address lines of symbols without extra hardware. We can see that although the proposed weighted history model needs one more cycle to update the model, the memory required is less than the adaptive model in [2].

In the encoding process, the modeling unit provides the probability and the cumulative probability of the current encoded symbol to the encoder. Because of the multibase cumulative occurrence array used in our implementation, the base of the bank should be added to the content of the bank to obtain the actual cumulative occurrence. Therefore, an additional adder is needed for accessing the cumulative occurrence of the modeling unit. In the decoding process, the decoder offers the current code point to the modeling unit, and the modeling unit scans the cumulative occurrence array to find the suitable symbol. Our structure has the advantage that there is no need to search over all the symbols in the array. Instead, we first find out the bank that the code point lies in and then search over this bank to find the corresponding symbol. The searching process for banks and that inside the bank can be parallel because there are only 16 items to be searched. The hardware complexity is reduced significantly since only 32, instead of 256, comparators are needed for the parallel searching structure. Thus, the multibase cumulative occurrence array makes the hardware implementation of the adaptive 256-symbol arithmetic decoder feasible. Note that the complexity of the modeling unit is highly dependent on the wordlength of the cumulative occurrence array. All the counters, comparators, memories, and adders have the same wordlength in the modeling unit. Thus, comparing the adaptive model in [2] that uses 14-b word-length, the proposed weighted history model with 7- or 5-b wordlength is easier to implement. The hardware complexity of our weighted history model is about 1/2 or 1/3 of that using the adaptive model in [2].

IV. Conclusion

A weighted history model proposed in this paper can solve the disadvantages of the limited past history model for arithmetic codes. The weighted history model can provide better performance with a smaller history buffer. Experimental results show that the arithmetic coding with weighted history model is good for image coding. A simple hardware block diagram for the weighted history model is also presented in this paper. The memory size is small and the hardware structure of this model can be easily implemented when the alphabet size is not too large. While the alphabet size is as large as several hundreds, a multibase cumulative occurrence array is proposed to decrease the hardware complexity of the updating and decoding circuits. The combination of the multibase cumulative occurrence array, modeling unit, encoder, and decoder can be easily implemented as VLSI for video compression.

References

[1] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proc. IRE,* vol. 40, pp. 1098–1101, Sept. 1952.

[2] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression," *Commun. ACM,* vol. 30, no. 6, pp. 520–540, June 1987.

[3] G. G. Langdon, "An introduction to arithmetic coding," *IBM J. Res. Develop.,* vol. 28, no. 2, pp. 135–149, Mar. 1984.

[4] R. N. Williams, *Adaptive Data Compression.* Norwell, MA: Kluwer, 1991.

[5] M. Ghanbari, "Arithmetic coding with limited past history," *Electron. Lett.,* vol. 27, no. 13, pp. 1157–1159, June 1991.

[6] M.-H. Hsieh and C.-H. Wei, "A multialphabet arithmetic coding with weighted history model," in *IEEE Int. Symp. Information Theory,* Whistler, B.C., Canada, Sept. 17–22, 1995, p. 393.

[7] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression.* Norwell, MA: Kluwer, 1992.

[8] D. M. Abrahamson, "An adaptive dependency source model for data compression," *Commun. ACM,* vol. 32, no. 1, pp. 77–83, Jan. 1989.

[9] J. L. Mitchell and W. B. Pennebaker, "Software implementations of the Q-Coder," *IBM J. Res. Develop.,* vol. 32, no. 6, pp. 753–774, Nov. 1988.

[10] R. B. Arps, T. K. Truong, D. J. Lu, R. C. Pasco, and T. D. Friedman, "A multi-purpose VLSI chip for adaptive data compression of bilevel images," *IBM J. Res. Develop.,* vol. 32, no. 6, pp. 775–794, Nov. 1988.

[11] J. Rissanen and K. M. Mohiuddin, "A multiplication-free multialphabet arithmetic code," *IEEE Trans. Commun.,* vol. 37, pp. 93–98, Feb. 1989.

[12] D. Chevion, E. D. Karnin, and E. Walach, "High efficiency, multiplication free approximation of arithmetic coding," in *Proc. Data Compression Conf.,* 1991, pp. 43–52.

[13] G. Feygin, P. G. Gulak, and P. Chow, "Minimizing error and VLSI complexity in the multiplication free approximation of arithmetic coding," in *Proc. Data Compression Conf.,* 1993, pp. 118–127.

[14] H. Printz and P. Stubley, "Multialphabet arithmetic coding at 16 MBytes/sec," in *Proc. Data Compression Conf.,* 1993, pp. 128–137.

[15] S. M. Lei, "Efficient multiplication-free arithmetic codes," *IEEE Trans. Commun.,* vol. 43, pp. 2950–2958, Dec. 1995.

[16] X. Ran and C. Y. Choo, "Syntax-based arithmetic video coding for very low bitrate visual telephony," in *Proc. IEEE Int. Conf. Image Processing,* 1995, vol. 2, pp. 410–413.