Finite State Vector Quantization with Multipath Tree Search Strategy for Image/Video Coding

Chen-Yi Lee, *Member, IEEE*, Shih-Chou Juan, and Yen-Juan Chao

***Abstract*—This paper presents a new vector quantization (VQ) algorithm exploiting the features of tree-search as well as finite state VQ's for image/video coding. In the tree-search VQ, multiple candidates are identified for ongoing search to optimally determine an index of the minimum distortion. In addition, the desired codebook has been reorganized hierarchically to meet the concept of multipath search of neighboring trees so that picture quality can be improved by 4 dB on the average. In the finite state VQ, adaptation to the state codebooks is added to enhance the hit ratio of the index produced by the tree-search VQ. Thus, compressed bits can be further reduced. An identifier code is then included to indicate to which output indexes belong. Therefore, this modified algorithm not only reaches a higher compression ratio, but also achieves better quality compared to conventional finite-state and tree-search VQ's. Finally, suitable VLSI architectures for real-time performance are proposed here 1) to remove the bottleneck of iteration bound in finite-state VQ and 2) to provide parallel computing structure for tree-search VQ to meet computational requirements.**

I. Introduction

Vector quantization or VQ is a technique for data compression. The key concept inherent in this technique is to replace an input vector by an index whose codevector has the minimum distortion compared to other codevectors of a designed codebook. To find such an index, several search schemes have been proposed in the literature [1], such as full search, tree-search, etc. In general, these schemes are evaluated *in terms of computation complexity, compression ratio, and picture quality*. Some minor modifications for storage space reduction on these schemes can also be found in the literature. For example, in [2], [3], the authors exploit transforms and then select key components for comparison. Thus both computational complexity and storage space can be reduced. However, the picture quality becomes degraded due to the selection of fixed transform components. To provide a high-quality picture service, a large codebook is often demanded. However, the drawback lies in the low compression ratio which can be defined by $(W \times N)/\log_2 M$, where M, N, and W stand for codebook size, input vector size, and input word length, respectively. It can be found that the compression ratio is very related to the codebook size M.

Manuscript received September 1, 1993; revised June 15, 1995. This paper was recommended by Associate Editor P. Pirsch. This work was supported by the National Science Council of Taiwan, R.O.C. under Grant NSC82-0404-E009-113. Also, the MPC support from Chip Implementation Center (CIC) of NSC for the prototype VQ chips is acknowledged.

The authors are with the Department of Electronics Engineering & Institute of Electronics, National Chiao Tung University, Hsinchu, Taiwan, R.O.C.

Publisher Item Identifier S 1051-8215(96)04110-9.

One way to improve the compression ratio while maintaining high picture quality is to exploit finite state machines for state tracing. These finite state vector quantization or FSVQ methods have been found to be an efficient technique for image compression [4]–[6]. Fig. 1 shows the block diagram of typical FSVQ's. Both encoder and decoder of an FSVQ have a finite state machine, which uses previously encoded vectors to decide current state and then selects one corresponding state codebook, which is a subset of master codebook, to quantize input vector. Since the state codebook is smaller than the master codebook, FSVQ can achieve both higher compression ratio and lower computational complexity than the full search VQ. And if the finite state machine can be designed well, the quality of this coding method will be better than that obtained by the full search VQ. For example, if the state codebooks contain 16 codevectors and the master codebook has 512 codevectors, then the computational complexity and the bit rate of FSVQ are 1/32 and 4/9, respectively, of those obtained by the full search VQ.

However, the state machine of FSVQ often does not produce the correct current state. In other words, the selected state codebook often does not contain the closest codevector and the encoder can only find suboptimal codevectors to encode the input vector. This causes larger distortion during the encoding process and since the next state is selected according to the previously decoded vector, the wrong selection of current state will influence the selection of the following state. Therefore, error becomes propagated and hence conventional FSVQ's only get 1–2 dB better coding quality than that obtained by the full search VQ at the same bit rate. In addition, since the current state codebook highly relies on previously decoded vectors, we cannot start to quantize the current input vector and select the state codebook from the state machines until the previous input vectors have been vector quantized. This feedback structure or iterative bound of FSVQ makes it difficult to develop real-time cost-effective hardwares for practical applications.

To solve the aforementioned problems, we propose a multipath tree search FSVQ algorithm and its VLSI architecture. In this novel approach, instead of waiting for the quantization of previous vectors and then selecting current state codebook to quantize input vector, we first use the tree search vector quantization or TSVQ method to find the indexes, I_{TSVQ}, of the two closest codevectors from the codebook, and then check whether the state codebook contains any of these two codevectors. If yes, the index, I_{state}, corresponding to one of the two closest codevectors in the state codebook will be transmitted,

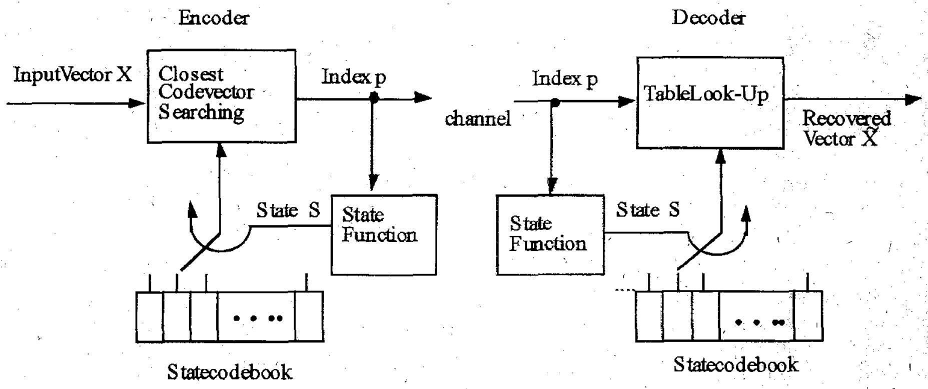


Fig. 1. Functional diagram of a typical FSVQ.

otherwise an identifier ID together with the index I_{TSVQ} of the closest codevector in the master codebook will be sent out. This method overcomes the problem that the state codebook, in some cases, does not contain the proper codevector to represent the input vector. In Section II we will describe this algorithm in detail. Also, some simulation results are given to highlight distinguished features of this algorithm. We then present a VLSI architecture for the proposed algorithm in Section III, where module design and removing iteration bound will be emphasized. Section IV outlines design issues of our approach and gives some comparisons with current VQ approaches for image and video coding.

II. The Multipath Tree Search FSVQ Algorithm

The functional diagram of the proposed algorithm is shown in Fig. 2. It contains two VQ phases: one is the TSVQ, and the other is the FSVQ. In the first phase of encoder design, we first exploit a multipath tree search VQ to find the nearly closest codevector. Since the computational complexity of TSVQ is almost as low as that of FSVQ, the advantage of lower computational complexity inherent in FSVQ is still reserved in this proposed algorithm. In the second phase, we then exploit a finite state machine to predict the possible index produced by the TSVQ. That is, the state machine selects a state codebook which may or may not contain the index I_{TSVQ} produced by the TSVQ. If the prediction is correct and the selected codebook really contains the I_{TSVQ}, the information needed to be transmitted to the decoder is the position I_{state} which specifies the position of I_{TSVQ} in the selected state codebook; otherwise an identifier code ID together with the I_{TSVQ} are transmitted to the decoder. This solves the problem that sometimes the state codebook does not contain the proper codevector to represent the input.

In the following, we will show how the TSVQ is designed to achieve better quality of decoded images and how the finite state machine is designed to reach lower bit rate.

A. Design of Multipath TSVQ

The first part of the proposed VQ algorithm is a TSVQ, which is exploited to find the nearly closest codevector to represent the input vector. The tree search VQ (TSVQ) or known as tree-structure VQ [7] is an alternative to full search VQ. In principle, the TSVQ performs a sequence of binary (or larger) searches instead of one large search as in conventional full search VQ's. As a result, *the encoding complexity increases as* $\log_2 M$ ($=$ *tree-depth* D) *instead of as* M, *where* M *is the number of leaves of the searching tree.* The performance of TSVQ will, in general, suffer some degradation over the performance of full search VQ at the same number of code words. In addition, the memory requirement in the encoder of TSVQ nearly doubles. However, the great advantage of reducing search complexity over some increase in distortion makes the TSVQ more attractive.

Before discussing the details of the TSVQ phase, we first briefly present two kinds of TSVQ's: balanced and unbalanced TSVQ's. Then we explain why one of them is more suitable for our TSVQ design.

1) Balanced TSVQ: Traditionally, TSVQ quantizers have been designed one layer at a time using the splitting method of the generalized Lloyd algorithm (GLA) [7]. The first layer of the tree is designed and then fixed. Each new layer of the tree is then designed by splitting each codevector of the previous level, i.e., the two new children of each node is produced by splitting the old codevector. The training sequence used to produce this old codevector will then be used to generate these new children. Thus, the TSVQ under such a growth scheme is a balanced tree, leading to a fixed rate code.

2) Unbalanced TSVQ: Makhoul [8] introduced an alternative TSVQ design algorithm. *The tree is designed one node at a time rather than one layer at a time by splitting the node that contributes most to the overall distortion of the coder.* This method results in an unbalanced tree because the node split can be at any depth. For a binary tree, the tree is grown until its number of leaves reaches a predetermined power of two. These leaves are then renumbered, and this mapping process is used as the codevector indexes. Although the tree is unbalanced, the code is fixed-rate. The advantage of this tree growing scheme is that there will be more codevectors available to code high distortion events which imply those trees will be split most of the time. Another approach to design an unbalanced tree is described in [9]. A balanced fixed rate TSVQ is grown and then optimally pruned back using the generalized Breiman, Friedman, Olshen, and Stone (BFOS) algorithm [10]. The resultant coder is a tree structure VQ, with leaves at varying depths, that results in a natural variable tree code. The coder is

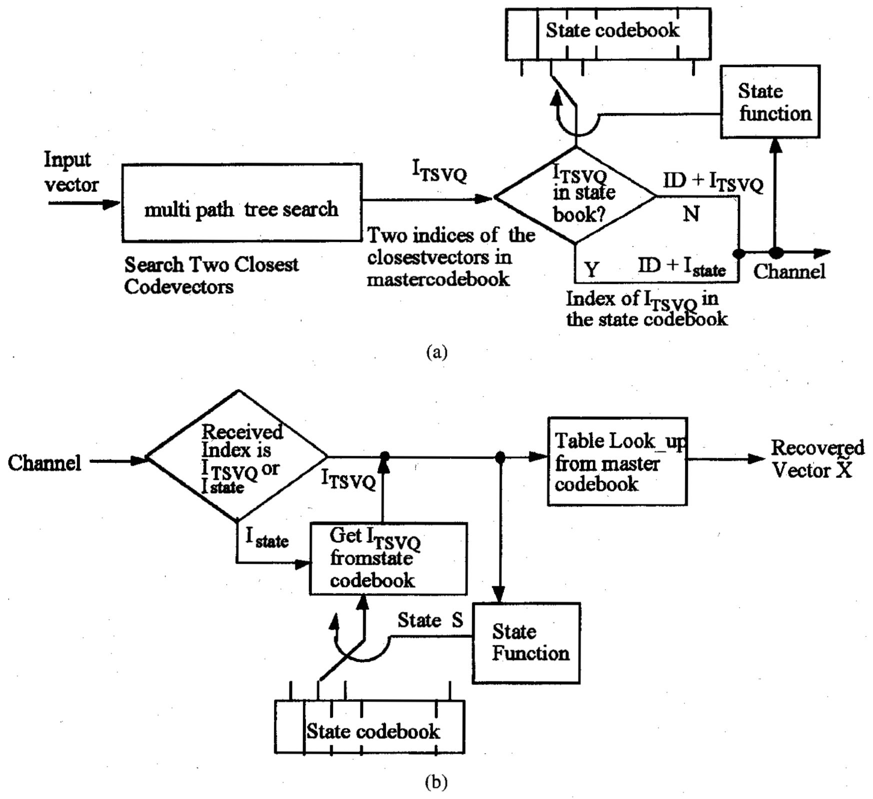

State codebook
State function
Input vector
multi path tree search
I_{TSVQ}
I_{TSVQ} in state book?
ID + I_{TSVQ}
N
Y
ID + I_{state}
Channel
Search Two Closest Codevectors
Two indices of the closestvectors in mastercodebook
Index of I_{TSVQ} in the state codebook
(a)
Channel
Received Index is I_{TSVQ} or I_{state}
I_{TSVQ}
I_{state}
Get I_{TSVQ} fromstate codebook
Table Look_up from master codebook
Recovered Vector X
State S
State Function
State codebook
(b)

Fig. 2. Functional diagram of the proposed FSVQ system: (a) encoder and (b) decoder.

called a pruned TSVQ (PTSVQ). The PTSVQ has the ability to out-perform the balanced TSVQ by being able to devote more bits to high distortion events.

Although the performance of unbalanced TSVQ is, in general, better than the balanced TSVQ, the uncertain searching depth makes it difficult for hardware implementation, which also degrades the advantage of lower computational complexity inherent in TSVQ. In addition, our experiment results show that sometimes the performance of the unbalanced TSVQ is not good enough, especially for images outside the training sequences. The reason for this phenomenon is that unbalanced TSVQ uses more codevectors to encode commonly used clusters while training the codebook, and for the less used clusters, fewer codevectors are used to represent them. Therefore, for the outside training images, if most input vectors belong to the less used clusters, the performance of this encoder becomes very bad. Due to this consideration, the balanced TSVQ is exploited here in the first phase to reach a reasonable computational complexity and make the algorithm more suitable for general images. Moreover, some strategies are added to improve the performance of the balanced TSVQ.

3) Strategy to Improve TSVQ: The TSVQ is a sequence of binary search operations which can be described as follows.

1) First, identify a searching node as the root.
2) Calculate the distortion between input vector and two children of the searching node. The new searching node is the children which is closer to the input vector.
3) If the new searching node is the leaf, the searching process is over and this leaf is identified as the codevector to represent the input vector; otherwise, go to step 2) for ongoing search.

Sometimes, the TSVQ cannot find the closest codevector because the "father" of the closest codevector is not always closer than the other nodes in the "ancestor" level. This phenomenon becomes more serious when the tree size is larger and the codevectors become closer to each other. Then the "ancestors" of the closest codevector are not selected as the new search node, because sometimes the brothers of these "ancestors" are closer to the input vector. If one of the "ancestors" is not selected, the obtained codevector will never be the closest one and then cause large distortion. This is why the performance of TSVQ is always lower than full search VQ.

To reduce the effect of this problem, we use the following strategies.

Multi-Path TSVQ: Chang *et al.* [11] have proposed a multipath TSVQ to improve the performance of TSVQ. Instead of finding one nearest node from two searching nodes, the multipath TSVQ finds two nearest nodes from the four searching nodes. The next four search nodes are the children of the two found nearest nodes. The computation of the multipath TSVQ is the double of that of conventional TSVQ's. The reason that multipath TSVQ searches more nodes than the TSVQ is because sometimes the "ancestor" of the closest codevector is not closer than other search nodes, then the tree search will cause some errors. Therefore, the multipath TSVQ

TABLE I
THE IMPROVEMENT OF MULTIPATH TSVQ VERSUS CONVENTIONAL TSVQ'S AND FULL-SEARCH VQ

schemes	*test images (SNR)*			
	lena	pepper	girl	jet
TSVQ	25.837	26.732	24.732	25.428
multi-path TSVQ	29.09	29.619	31.97	29.505
full-search VQ	29.718	30.296	32.79	30.616

TABLE II
THE EXPERIMENT RESULTS OF SEARCHING THE NEIGHBORS OF THE CLOSEST CODEVECTOR

schemes	*test images (SNR)*			
	lena	pepper	girl	jet
multi-path TSVQ with 8 neigh.	29.311	29.844	31.82	29.929
multi-path TSVQ with 4 neigh.	29.232	29.706	31.611	29.776
multi-path TSVQ without neigh.	29.09	29.619	31.57	29.505

searches more nodes to reduce the effect. Table I shows the improvement of the multipath TSVQ. This experiment uses a codebook of 256 codevectors and four test images.

Neighbors Searching: After obtaining the nearly closest codevector by the multipath TSVQ, we can search the neighbors of the obtained codevector. Since the codevector is very close to the input vector, the closest codevector must be very close to the obtained codevector. In other words, the closest code word may be in the neighbors of the obtained codevector. This method is done by the following steps.

1) Find the n neighbors of each code word, the neighbors of each codevector is listed in a table.
2) For each input vector, use multipath TSVQ to find the nearly closest codevector.
3) Then search the neighbors of the obtained codevector; if any neighbor is closer to the input vector, the final obtained codevector is the closest one.

Table II shows the results of applying this method which achieves 0.2–0.3 dB higher compared to those without searching the neighbors.

Codebook Design: The codebook of traditional balanced TSVQ is designed with the splitting method of the GLA [7]. But the codebook generated by this method cannot ensure that the "ancestor" of the closest codevector will be closer to the input vector than its "brothers." To achieve better coding quality, the codebook used in the TSVQ has to be modified. In this paper, we propose the following scheme to construct the tree structure codebook.

1) Use the LBG algorithm [12] to produce the codebook until the codebook size reaches the desired figure M. These M codevectors are the leaves of the tree.
2) For the obtained N-size codebook, we first identify the pairs whose leaves are closest to each other. This location process begins at looking for the furthest pair (i.e., pair distance is the longest). The codevectors in the same pair become the "brothers" in the searching tree. And the "father" is constructed by averaging the two codevectors in the pair. Then we obtain the level of $M/2$-node of the searching tree. This process will continue until the root level is reached.

TABLE III
THE IMPROVEMENT OF MULTIPATH TSVQ WITH THE "AVERAGE TREE CONSTRUCTION" METHOD

schemes	*test images (SNR)*		
	lena	pepper	jet
TSVQ with new codebook	29.267	29.852	29.794
TSVQ with LGB codebook	29.09	29.619	29.505
full-search VQ	29.718	30.296	30.616

We call this method "average tree construction," because the "father" is always the average of the two children. Experiment results (given in Table III) show that codebook constructed by this method will achieve higher quality (0.2–0.3 dB in SNR) compared to codebook generated by LGB. This improvement results from the fact that the father is the "center" of the two children, therefore the "ancestor" of the closest code word will be closer to the input vector than the conventional TSVQ whose codebook is generated the GLA.

B. The Second Phase—Finite State Machine

The second part of the proposed algorithm is a state machine which uses the information of previously decoded vectors to predict the possible index I_{TSVQ} generated by the TSVQ. That is, the state machine will select a proper state codebook which may contain the newly generated index I_{TSVQ}. *In other words, the elements of the state codebook are the possible indexes of I_{TSVQ}, rather than the entries of the codevectors used in conventional FSVQ's.* This strategy leads to the saving of large memory space and to the reduction of computational complexity. If the selected state codebook really contains the new index I_{TSVQ}, an index I_{state} which indicates the position of the I_{TSVQ} in the selected state codebook is transmitted to decoder. Since the selected state codebook of the decoder is the same as that of the encoder, the decoder can use this I_{state} to recover the corresponding I_{TSVQ} and then reconstruct images from the closest codevectors. Because the size of the state codebook is smaller than the larger codebook of the TSVQ, fewer bits for I_{state} instead of the larger size I_{TSVQ} are needed for transmission. However, if the selected state codebook does not contain the newly generated index I_{TSVQ}, an identifier code ID together with the I_{TSVQ} will be transmitted to the decoder. With this ID code, the decoder knows how to assemble the decoded images by using either the I_{TSVQ} or the I_{state}.

In the meantime, we can adaptively change the content of selected state codebooks to reflect the local statistics of input image/video sequences. This is done by inserting the recently used index on the top of state codebooks. This results in an adaptive version of multipath tree search based FSVQ algorithm whose output is a variable rate. If the state machine is designed well, and selected state codebooks usually hit the I_{TSVQ}, then the average bit rate becomes lower than that obtained from FSVQ without adaptation.

C. Experiment Results

We have used this scheme to implement the side match VQ (SMVQ) [5], which is one of the FSVQ's, to observe the coding quality. Simulation results are obtained from two different approaches: one is the multipath tree search FSVQ or called the modified SMVQ, and the other is the adaptive version which is called the adaptive modified SMVQ. The adaptive modified SMVQ adaptively updates the state codebook by exploiting the least recently used (LRU) strategy. This LRU is implemented by placing the most recently used element on the top of selected state codebooks. Here, the codebook of the TSVQ is generated by the "average tree construction" technique mentioned earlier, where a four-path TSVQ is used. We have changed both sizes of the codebook of the TSVQ and the state codebook of the state machine. *The larger the codebook size of TSVQ is, the higher will be the signal-to-noise ratio (SNR) coding results obtained. For a given TSVQ codebook size, there is an optimal state codebook size to reach a minimum bit rate.* The more complex the encoded image is, the larger the optimal state codebook size is. Fig. 3 shows the comparison of the SNR versus bit-rate of this proposed algorithm with that obtained from the original SMVQ. Note that each point is derived at an optimal state codebook size for different TSVQ codebook sizes. Simulation results show that the modified SMVQ implemented by our proposed approach does achieve higher SNR than that obtained from the original SMVQ. The reason why this modified SMVQ improves the coding quality is because our proposed algorithm eliminates the disadvantage that the nearest codevector is not in the state codebook as found in conventional FSVQ's. And the reason the adaptive modified SMVQ out-performs the others is because the state codebook of the adaptive modified SMVQ can trace the local statistics of input image/video sequences so that the prediction of the I_{TSVQ} is more accurate and the average output bit rate becomes lower.

Fig. 4 shows the results of our algorithm for two test images. Output bit-rates for "Lena" and "Pepper" are 0.321 and 0.305 b/pixel, respectively. Picture quality obtained from this method is very close to that from full-search VQ, as shown in Table III. However, the output bit-rate becomes less than 65% of that from full-search VQ.

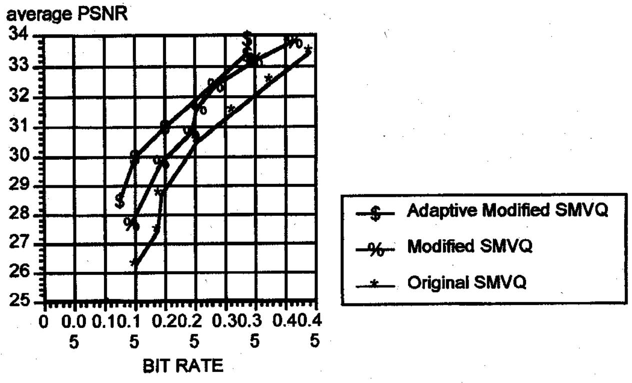


Fig. 3. SNR versus different bit rates. Simulation results on Claire sequence are obtained, respectively from original SMVQ and our modified SMVQ implemented by the proposed architecture. The adaptive one modifies state codebook by using LRU strategy. The master codebook is generated by using LBG algorithm from the luminance of four JPEG test images: "balloons," "Barbara," "boats," and "chairlady."

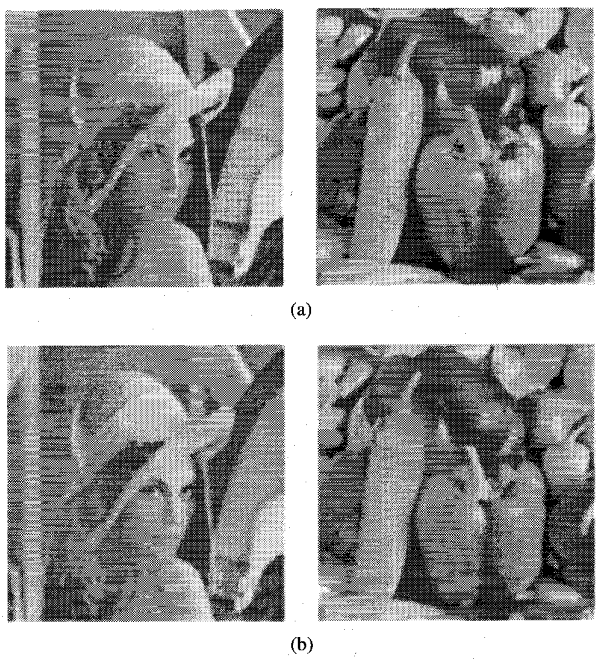


Fig. 4. Simulation results for two test images—Lena and Pepper—(a) shows the original images and (b) shows the decoded images with output bit-rates at 0.321 and 0.305 b/pixel, respectively.

III. VLSI Design for Real-Time Performance

It can be seen from the algorithm description that the recursive part is only in the finite state machine. Therefore, to speed up coding performance, we have to break this bottleneck in order to meet real-time requirements. In this section, we first deal with a pipeline structure for implementing the TSVQ. We then show a parallel structure for removing the iteration bound of the FSVQ. Thus, the complete architecture, which is also named "multipath tree-search based architecture" here [13], can be pipelined to implement the proposed algorithm.

A. Architecture for the TSVQ

The required operations in different levels of the tree structure are very related because only when the upper level is determined can the next level be performed. However, there is no feedback loop between two adjacent levels. Therefore, each level can be performed independently if its upper is already determined. Thus, the structure of tree search VQ is inherently pipelined. An overall hardware organization of the encoder is shown in Fig. 5. The TSVQ contains several stages, each of which has different sizes of codebook depending on its depth in the complete tree structure. The indexes of the two nearest codevectors are transmitted to the address generator which

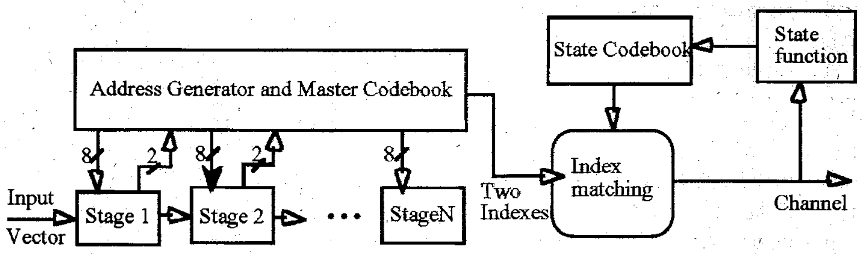


Fig. 5. Block diagram of the encoder.

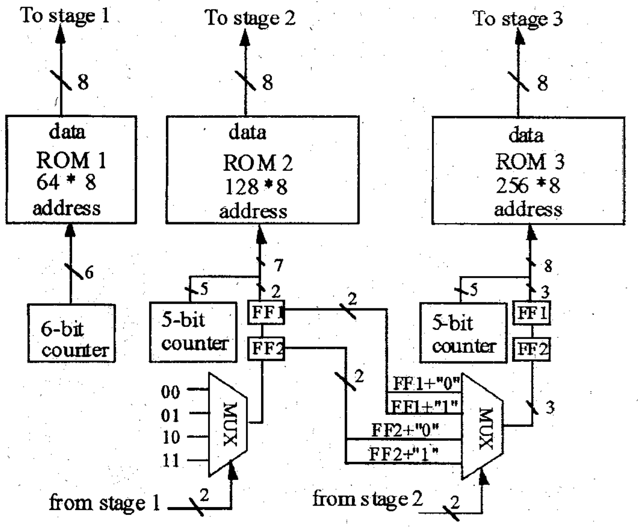


Fig. 6. Block diagram of the first three stages for address generator and master codebook.

produces the address for the codevectors needed at the next level. Fig. 6 shows the architecture of the first three stages of the address generator. Each stage of the address generator uses the indexes produced by the previous stage of TSVQ to determine the searching node of current stage. The two indexes produced by the last stage of the TSVQ are the indexes of the two identified nearly closest codevector I_{TSVQ}'s which will be passed to the FSVQ.

From this architecture description, it can be found that only (D-2) levels are needed for tree searching, where D is tree depth. This is because four nodes are required simultaneously at each level, implying the first two stages, i.e., D_0 and D_1, are never used. The TSVQ architecture mainly consists of two parts, one is the storage and address generation for codevectors and the other is the distortion estimation and index selection. It is clear that for tree depth of D, $[2^{(D+1)}-4]$ codevectors are needed to be stored. Also the addresses for four different codevectors have to be provided by the address generators whose part of the MSB's are determined by its previous stage. As for the distortion estimation and index selection, it can be easily realized on a few adders, registers, and multiplexers [13].

B. Architecture for the FSVQ

From the algorithm description, we know that the only calculation in the finite state machine is to check whether

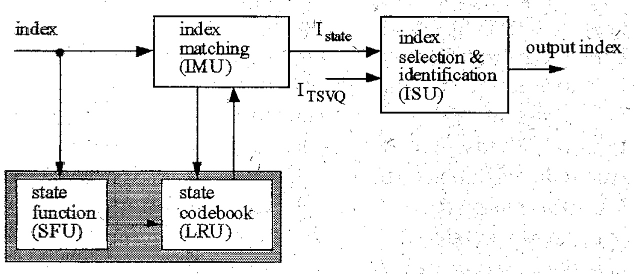


Fig. 7. Overall hardware organization of the FSVQ. Note that SFU and LRU can be realized on PLA and RAM, respectively.

the two I_{TSVQ}'s obtained from the TSVQ are in the selected state codebook. Since only index instead of codevectors has to be compared, matching can be done easily by reading contents from state codebook. Here, the state codebook is selected by a finite state function which can be easily realized on simple hardwares, e.g., using table look-up for the state function. For other FSVQ's, we only need to change the state function of state machine and the content of the state codebook, therefore, this architecture can implement various types of FSVQ. Fig. 7 shows the overall hardware organization of this part. It contains an IMU for index matching, an LRU for adaptation of the content of state machine, an SFU for state function, and an ISU for selection of index to be sent.

Fig. 8 shows the details of the IMU and ISU. This part checks whether the two I_{TSVQ}'s obtained from the TSVQ are in the selected state codebook. In addition, it provides addresses for the state codebook which are selected by state function. If the *match* flag becomes activated, then the shorter index I_{state} accompanied by an ID is sent; otherwise, the longer index of the TSVQ I_{TSVQ} together with another ID are sent to the channel over the parallel/serial converter.

C. VLSI Implementation

A prototype VLSI chip (as shown in Fig. 9) for the proposed algorithm has been fabricated and tested using 0.8 μm CMOS double-metal process. Results show that at clock rate of 33 MHz, an image/video sequence of 30 frames with size of 1024 $\times$ 1024 pixels can be handled by one single chip. In order to allow the use of different codebooks for picture quality verification, about 44 Kb SRAM size is included in this chip. Below is the summary of this VQ chip.

- Algorithm: multipath tree-search (seven-stage) FSVQ algorithm;
- Throughput: 1 pixel per cycle;

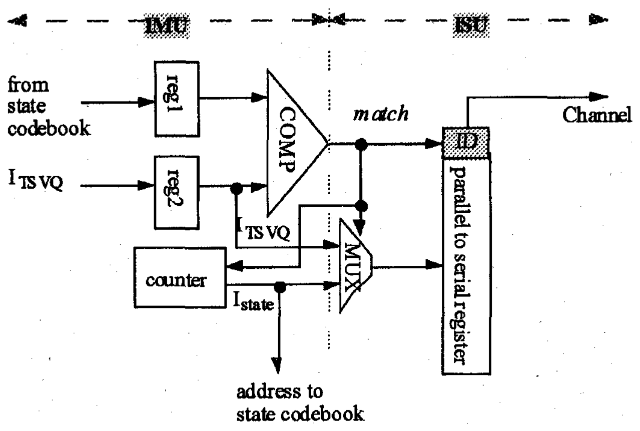


Fig. 8. Block diagram of index matching and selection units.

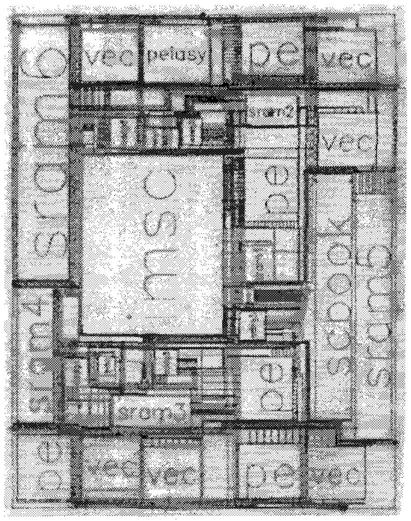
- Maximum clock rate: 35 MHz;
- Transistor-count: 420K;
- Internal memory size: 40 Kb (22.8 mm^2);
- Die size: 95 mm^2;
- Power consumption: 0.95 W at (33 MHz, 5 V);
- Process: TSMC 0.8 μm CMOS SPDM process;
- Package: 84-pin CLCC;
- Design style: full-custom + cell-based approach.

IV. Evaluation and Discussions

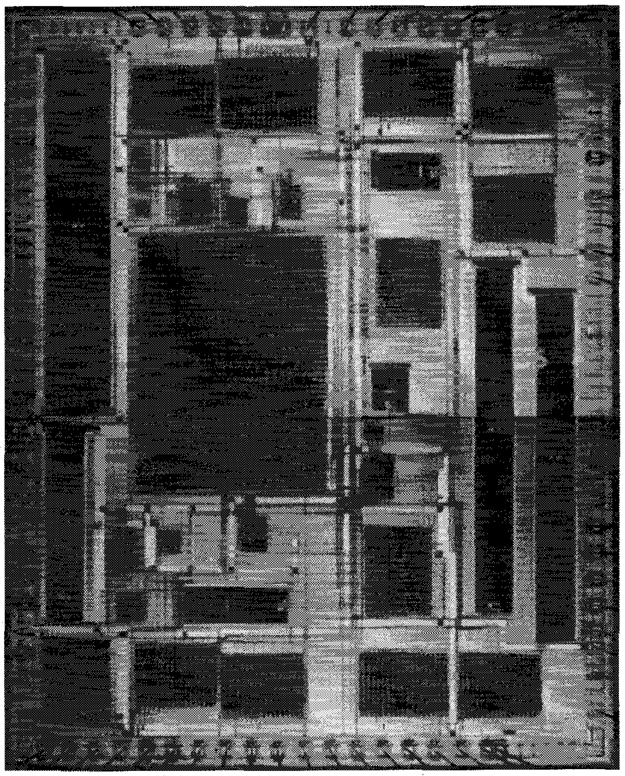
Compared to other FSVQ algorithms [5], [6], our approach offers higher picture quality at the same bit-rate. In the meantime, the iteration bound is now removed in our algorithm because only one input data has to be compared instead of an input vector. Although the multipath TSVQ is added and, hence, computational requirements become higher, we can exploit the pipelining inherent in the TSVQ to improve speed. Unlike other available VQ solutions [14]–[17] which are dedicated to tree-search or full-search algorithms, our approach does reach a more optimal solution in terms of picture quality, bit-rate, and hardware complexity.

From both algorithm and architecture descriptions, we know that pipelining can be applied to this new algorithm where computational complexity can be reduced and compression ratio can be enhanced. Below, we highlight some practical design issues for real-life applications.

1) The required memory space can be estimated by (2M-4)NW + 1024 W, where the first term is for TSVQ and the second term is for FSVQ.
2) Input sample rates for TSVQ and FSVQ are different. Since the TSVQ deals with block image data while the FSVQ deals with index data, data rate of the latter is only $1/N$ of the former. Thus, the TSVQ is more computation intensive.
3) Parallel computation or shared hardware data-path can be considered by trading off area and performance at the multipath TSVQ. For example, if only one 8-b data bus is available, at least $4N$ cycles are needed for each stage. On the other hand, if the bus number increases, parallel datapaths are needed and, hence, cycle-count can be reduced.



(a)



(b)

Fig. 9. (a) Shows floor plan of the VQ encoder chip, and (b) shows microphoto of the chip.

V. Conclusion

In this paper, we have proposed a new VQ algorithm and architecture which combines the advantages of both tree-search and finite-state VQ's. This so-called multipath tree-search FSVQ architecture does solve the problem that sometimes the selected state codevector does not contain the better codevector to represent the input vector. Simulation results have shown that good picture quality can be achieved at lower output bit-rate. Moreover, we have also presented a VLSI solution for the proposed algorithm, where tested results show that 30 image/video frames per second (1024 $\times$ 1024 pixels) can be handled at 33 MHz.

In summary, this fully pipelinable multipath tree-search based architecture not only suits the implemention of various FSVQ's for real-time image/video coding, but also achieves higher SNR than conventional FSVQ's in terms of bit-rate and picture quality.

Acknowledgment

The authors would like to thank their colleagues within the SI2 group of NCTU for many discussions and fruitful suggestions.

References

[1] R. M. Gray and A. Gersho, *Vector Quantization and Signal Processing.* Norwell, MA: Kluwer, 1991.
[2] C. Y. Lee and S. C. Juan, "An ASIC architecture for real-time image/video coding based on fixed-basis distortion vector quantization," in *Proc. ISCAS'92,* San Diego, CA, May 10–13, 1992.
[3] ——, "VLSI implementation of a modified-VQ encoder suitable for image/video applications," in *Proc. EURSIPCO'92,* Brussels, Belgium, Aug. 26–28, 1992.
[4] J. Foster, R. M. Gray, and M. O. Dunham, "Finite state vector quanitization for waveform coding," *IEEE Trans. Info. Theory,* vol. IT-31, pp. 348–355, May 1985.
[5] T. Kim, "Side match and overlap match vector quantizers for image," *IEEE Trans. Image Processing,* vol. 1, no. 2, Apr. 1992,
[6] W. T. Chen, R. F. Chang, and J. S. Wang, "Image sequence coding using adaptive finite-state vector quantization," *IEEE Trans. Circuits Syst. Video Technol.,* vol. 2, no. 1, Mar. 1992.
[7] A. Buzo, A. H. Gray, Jr., R. M. Gray, and J. D. Markel, "Speed coding based upon vector quantization," *IEEE Trans. Acoust., Speech, Signal Processing,* vol. ASSP-28, pp. 562–574, Oct. 1980.
[8] J. Makhoul, S. Roucos, and H. Gish, "Vector quantization in speech coding," *IEEE Trans. Commun.,* vol. COM-36, no. 8, pp. 957–971, Aug. 1988.
[9] P. A. Chou, T. Lookabaugh, and R. M. Gray, "Optimal pruning with applications to tree-structured source coding and modeling," *IEEE Trans. Inform. Theory,* vol. 35, no. 2, pp. 299–315, Mar. 1989.
[10] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees.* Belmont, CA: Wadsworth, 1984.
[11] F. Chang, W. T. Chen, and J. S. Wang, "Image sequence coding using adaptive tree-structure vector quantization with multipath searching," in *IEEE Int. Conf. Acoust., Speech, and Signal Processing,* 1991, pp. 2281–2284.
[12] Y. Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design," *IEEE Trans. Commun.,* vol. COM-28, no. 1, pp. 84–95, 1980.
[13] Y. J. Chao and C. Y. Lee, "A new multipath tree-search FSVQ architecture for image/video sequence coding," in *Proc. ISCAS'95,* Seattle, WA, Apr. 29–May 3, 1995, pp. 1628–1631.
[14] G. A. Davidson, P. R. Cappello, and A. Gersho, "Systolic architectures for vector quantization," *IEEE Trans. Acoust., Speech, Signal Processing,* vol. 36, pp. 1651–1664, Oct. 1988.
[15] P. A. Ramamoorthy, B. Potu, and T. Tran, "Bit-serial VLSI implementation of vector quantizer for real-time image coding," *IEEE Trans. Circuits Systems,* vol. 36, pp. 1281–1290, Oct. 1989.
[16] H. Park and V. K. Prasanna, "Modular VLSI architectures for real-time full-search-based vector quantization," *IEEE Trans. Circuits Syst. Video Technol.,* vol. 3, pp. 309–317, Aug. 1993.
[17] W. C. Fang, C. Y. Chang, B. J. Sheu, O. T. C. Chen, and J. C. Curlander, "VLSI systolic binary tree-searched vector quantizer for image compression," *IEEE Trans. VLSI Syst.,* vol. 2, no. 1, pp. 33–43, Mar. 1994.

Chen-Yi Lee (S'89–M'90) received the B.S. from National Chiao Tung University, Taiwan, in 1982, the M.S. and Ph.D. degrees from the Katholieke University Leuven (KUL), Belgium, in 1986 and 1990, respectively, all in electrical engineering.

From 1986 to 1990, he was with IMEC/VSDM, working in the area of architecture synthesis for DSP. Since February 1991, he has been an Associate Professor in the Department of Electronics Engineering at the National Chiao Tung University, Hsinchu, Taiwan. His research interests mainly include video/image coding, high-speed networking, VLSI architecture, and system-level synthesis.

Shin-Chou Juan received the B.S. and M.S. degrees from the Department of Electronics Engineering from the National Chiao Tung University (NCTU), Hsinchu, Taiwan, in 1990 and 1992, respectively. He is currently working toward the Ph.D. degree in the Institute of Electronics, NCTU.

His research interests include image coding, ATM networking, and high-speed digital integrated circuits.

Yen-Juan Chao was born in Rangoon, Burma, on March 18, 1971, and moved to Taiwan with her family in 1974. She received the B.S. and M.S. degrees from the Department of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan, in June 1993 and June 1995, respectively.

She is now with the Silicon Integrated Systems Corp., Taiwan, as a Design Engineer working on PCI interface designs. Her research interests include low-power VLSI circuits, image/video coding, and high-speed architecture performance analysis.