Real-Time Video Compression Using Differential Vector Quantization

James E. Fowler, Jr., *Student Member, IEEE*, Kenneth C. Adkins, *Member, IEEE*, Steven B. Bibyk, *Member, IEEE*, and Stanley C. Ahalt, *Member, IEEE*

Abstract—**This paper describes hardware that has been built to compress video in real time using full-search vector quantization (VQ). This architecture implements a differential-vector-quantization (DVQ) algorithm and features a special-purpose digital associative memory, the VAMPIRE chip, which has been fabricated in 2 μm CMOS. We describe the DVQ algorithm, its adaptations for sampled NTSC composite-color video, and details of its hardware implementation. We conclude by presenting both numerical results and images drawn from real-time operation of the DVQ hardware.**

I. Introduction

VECTOR quantization has become well-known and widely studied since Shannon first established the merits of quantizing vectors rather than scalars [1]. Since that time, vector quantization (VQ) has been shown to be useful in the realm of data compression, particularly attracting attention for its efficient compression of digitized speech and image data. Meanwhile, as digital data has become more prevalent, the demand for real-time image-coding hardware has increased dramatically. The design of real-time vector quantizers for image coding has been particularly difficult due to the inherent computational complexity of VQ encoders and the extremely fast speeds demanded by real-time video applications. While much of the research effort directed at image compression via VQ has not been directly applicable to real-time hardware coding architectures, some recent proposals have been put forward ([2]–[5]). Unfortunately, the actual performance of VQ of real video signals in real time has not been sufficiently demonstrated. The work presented here represents a real-time demonstration of the merits of VQ.

In this paper, we begin with a review of VQ and previously proposed hardware VQ architectures. Next, we give a description of an algorithm for video compression called differential vector quantization (DVQ), a combination of differential-pulse-code modulation (DPCM) and full-search VQ. We present an architecture that, having been constructed in hardware, implements this algorithm in real time. This architecture is centered around a special-purpose digital associative memory, the VAMPIRE chip, which has been fabricated with a 2 μm CMOS n-well process and is described briefly here. We conclude with several examples drawn from the actual operation of the hardware.

Manuscript received October 13, 1993; revised April 22, 1994 and June 24, 1994. The work of J. E. Fowler, Jr. was supported by a NASA Space Grant/OAI Graduate Fellowship from the Ohio Space Grant Consortium. This work was supported in part by Grants from Cray Research, Inc. This material is based upon work supported by NASA under NAG-3-1 164 and NAG-3-1 802. This paper was recommended by Associate Editor L. Wu.

The authors are with the Department of Electrical Engineering, The Ohio State University, Columbus, OH 43210 USA.

IEEE Log Number 9408040.

II. Vector Quantization

Vector quantization is a well-known technique for data compression and has been discussed extensively elsewhere (see in particular [6] and [7]). What follows is a brief overview of the theory of VQ and some of the research directed toward hardware solutions of VQ encoding.

A. Vector-Quantization Theory

The philosophy of VQ stems from Shannon's rate-distortion theory which implies that, theoretically, better performance can always be obtained from coding vectors of information rather than scalars [1]. An extensive discussion of VQ techniques and applications is given in [6], and the basic theory is summarized in the context of image applications in [8].

In a typical application of VQ to image coding, the image is broken into blocks of pixels called tiles. Each image tile of $n \times m$ pixels can be considered a vector, $\mathbf{u}$, of dimension $k = mn$. For each image tile, the encoder selects the codeword $\mathbf{y}$ that yields the lowest distortion by some distortion measure $d(\mathbf{u}, \mathbf{y})$. The index, j, of that codeword is sent through the transmission channel. If the channel is errorless, the decoder retrieves the codeword $\mathbf{y}$ associated with index j and outputs $\mathbf{y}$ as the reconstructed image tile, $\hat{\mathbf{u}}$.

In the past, VQ has had limited use in image-compression applications because of the large computational expenses of both the encoding and training processes. In both processes, distortions are calculated for each codeword in the codebook and these distortions are compared to find the closest codeword. Since these calculations must be performed for each input vector, the overall operation is quite computationally expensive. Another disadvantage of VQ is that, because it encodes blocks, it tends to make the image edges "blocky" [9].

In an effort to reduce the amount of computation associated with full-search VQ encoding as discussed above, several suboptimal methods have been proposed. These suboptimal alternatives typically restrict the codeword search to a subset of the codebook (tree-structured VQ), or split the quantization into separate steps that each use smaller codebooks (multistage VQ and mean/residual VQ). These suboptimal approaches have been shown to effectively reduce computational expenses

while sacrificing some encoder performance. For an detailed survey of these techniques, refer to [7].

B. Associative Memories and VQ

Associative memories (AM's), also known as content-addressable memories (CAM's), are data storage devices which are accessed by the contents of the memory cells rather than the addresses of the cells. This property gives AM's inherent search capabilities which are nonexistent in conventional memories. VQ is well-suited to associative techniques; indeed, an AM that operates on the VQ distortion measure $d(\mathbf{u}, \mathbf{y})$ is a direct implementation of VQ into hardware [18]. The use of an AM is one of the few viable alternatives for performing video-rate VQ since the address space of a standard look-up table is too large to be practically implementable, and serial-search techniques are too time-consuming for real-time operation [18].

Although well-studied in the literature, relatively few AM's have been designed and implemented because they tend to be more application specific than ordinary RAM memory. Additionally, of those that do exist, most AM's are restricted to exact matches; that is, the input word must be exactly the same as the stored word for the match to be found [18]. For our implementation of VQ, we need an AM that can perform inexact matches governed by a suitable VQ distortion measure.

C. Hardware VQ Architectures

Despite the promising performance of VQ in theory, practical hardware VQ architectures for image coding have been somewhat scarce in literature. The first hardware VQ encoders were designed for speech; however, real-time video encoding requires significantly faster processing. Rather than attempt full-search VQ at these rates, several proposed architectures for real-time video VQ are based on the suboptimal techniques discussed above. For example, Dezhgosha *et al.* [2] propose an architecture based on mean/residual VQ and Ramamoorthy *et al.* [12] propose using multi-stage VQ.

Recent advances in VLSI technology have made full-search VQ at video rates possible. For example, Panchanthan and Goldberg [3] propose an architecture based on an exact-match CAM. More recently, analog VQ encoding chips have been fabricated by Fang *et al.* [4] and Tuttle *et al.* [5]. However, none of the above have actually demonstrated performance of real-time video VQ. A number of papers (e.g., [2], [3], and [12]) have presented *proposals* for hardware architectures, but few have actually fabricated their designs. The analog designs of [4] and [5] suffer from limited precision and conclusive operational behavior for real-time video has not been proven for either.

Below, we present an algorithm for VQ of real-time sampled NTSC video. The architecture which implements the algorithm is described and results which were obtained from real-time operation of the hardware are given. The heart of the design features a digital AM, the VAMPIRE chip, which is described below with the hardware implementation of the DVQ algorithm. The VAMPIRE chip is an inexact-match AM that calculates absolute distance between stored vectors and an input vector. The VAMPIRE chip delivers the quantization performance associated with full-search VQ without suffering from the limited precision of analog designs.

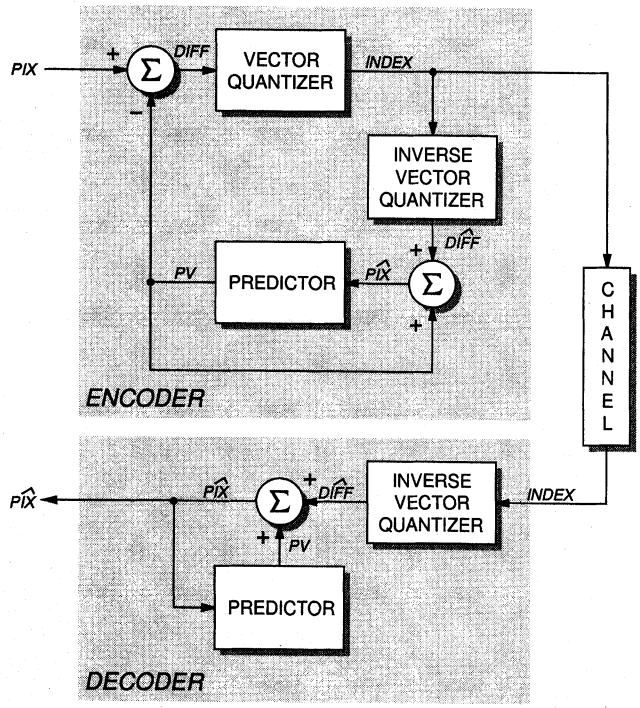


Fig. 1. Block diagram of DVQ algorithm

III. Algorithm Description

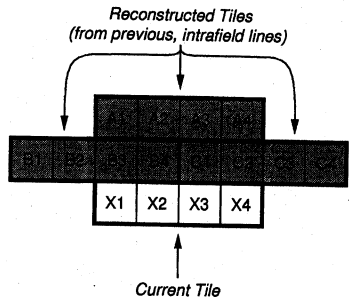
In this section, we discuss the algorithm for image compression which is the basis of the hardware implementation presented later in the paper. This algorithm has been explored in detail previously [8], so only a brief overview is given here. We conclude the algorithm description by presenting several practical considerations relevant to real-time processing of sampled NTSC video signals using our algorithm.

Our algorithm is called differential vector quantization (DVQ) and it combines the methods of VQ and DPCM. An artificial neural network (ANN) is used to train the VQ codebooks. A brief overview of these techniques follows.

A. Artificial Neural Networks and Vector Quantization

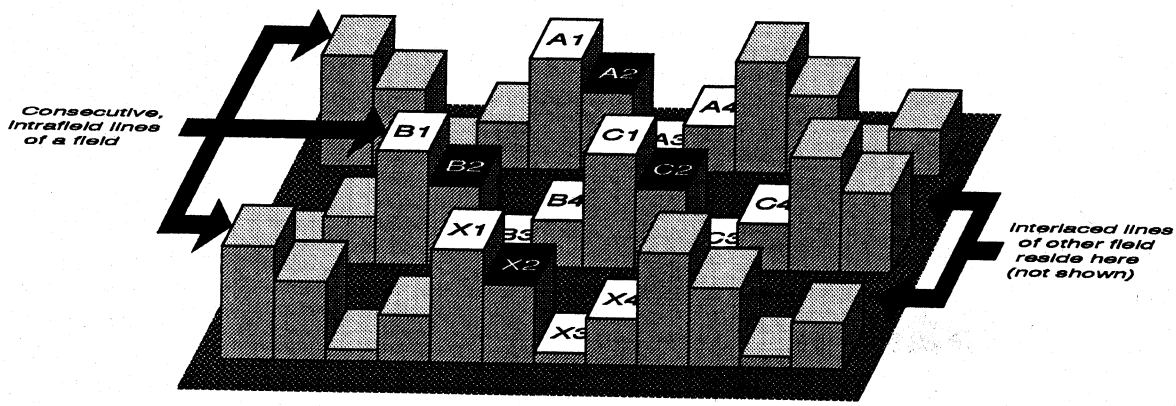
The computational complexity of traditional methods for the design of VQ codebooks has restricted their use in real-time applications [6], [13]. One such traditional approach is the Linde, Buzo, and Gray (LBG) algorithm [13], a locally optimal algorithm that has been used extensively in designing vector quantizers for speech and image encoding. It has been shown that ANN's can be used for design of VQ codebooks to circumvent the limitations of traditional algorithms [14].

ANN's consist of a large number of simple, interconnected computational units that can be operated in parallel. Also, ANN-codebook-design algorithms do not need access to the entire training data set at once during the training process. These features make ANN algorithms ideally suited for the design of adaptive vector quantizers.



Prediction: $\tilde{X}_i = ((B_i + C_i)/2 + A_i)/2$

(a)



(b)

Fig. 2. Prediction of sampled NTSC composite-color video. (a) Prediction from reconstructed tiles of previous, intrafield lines. (b) Phase considerations.

One ANN algorithm, frequency-sensitive competitive learning (FSCL) [15], [16] features a modified distortion measure that ensures all codewords in the codebook are updated equally frequently during iterations of the training process. It has been shown that codebooks designed with FSCL yield mean squared errors and signal-to-noise ratios comparable to those of the locally optimal LBG algorithm [14]. Also, the FSCL ANN yields codebooks with good mean-squared-error performance and with sufficient entropy so that entropy coding of the VQ indices would not provide significant additional compression [8].

B. The Differential-Vector-Quantization Algorithm

DVQ combines the methods of VQ and DPCM. DVQ replaces the scalar quantizer in the DPCM framework with a vector quantizer, and consequently has many of the compression advantages of both VQ and DPCM. DVQ has been presented previously in [9], where it was called vector DPCM, and in [7], where it was called predictive VQ (PVQ). One of the first applications of prediction to VQ for image coding was [17], which also featured a delayed-decision encoding tree.

Fig. 1 shows the general block diagram of our DVQ algorithm. In the encoding process, the predictor uses previously reconstructed tiles to predict the pixel values of the current tile. This predicted tile, PV, is subtracted pixel by pixel from the actual tile, PIX. The resulting difference tile, $DIFF$, is vector-quantized and the index, $INDEX$, is broadcast via the transmission channel to the decoder. The encoder inverse vector-quantizes $INDEX$, producing a reconstructed tile, $\widehat{PIX}$, to be used in later predictions. Note that, since the vector quantizer processes difference tiles, the VQ codebook must be appropriately derived from "difference images."

The decoder architecture is very similar to that of the encoder; in fact, a decoder is contained within the encoder. This replication exists so that the encoder tracks the performance of the decoder in order that the predictions in the encoder are identical to those in the decoder. Hence, in the absence of channel errors, the output of the decoder can be found in the encoder. Thus, for prototyping purposes, algorithm performance can be demonstrated by constructing only the encoder.

DVQ has several advantages over both scalar DPCM and VQ. As mentioned above, the quantization of vectors yields better compression performance than that of scalars. Additionally, since the VQ is performed on difference values rather than on the image itself, the resulting image is less "blocky" [9]. Finally, the codebooks for DVQ tend to be more robust and more representative of many images than codebooks designed for VQ because the difference tiles in a DVQ codebook are more generic than the image tiles in a VQ codebook [9].

There are many decisions to be considered in the design of a DVQ algorithm, such as tile size, distortion criterion, and method of prediction. These issues have been discussed in previous publications and so are omitted here. For more detail, refer to [8].

C. NTSC Considerations

The hardware implementation of our DVQ algorithm processes NTSC composite-color video signals. Generally, color video is comprised of three signals: one luminance signal and two separate color signals. In composite-color video, the two color signals are combined in quadrature, modulated by a specially chosen frequency called the color subcarrier, and added to the luminance signal. Finally, horizontal and vertical synchronization pulses are included to produce the baseband composite-color video signal.

In typical discussions of image compression, compression is performed on red-green-blue pixel arrays (ppm-format images), and, consequently, nearest-neighbor pixels can be used in prediction. In contrast, the hardware implementation of our DVQ algorithm processes sampled NTSC composite-color video signals. To correctly perform prediction on sampled composite-color video, one must consider the phasing of the color subcarrier.

Fig. 2 illustrates the prediction scheme of our DVQ algorithm and how it accounts for the phase of the color subcarrier. Imagine that the video signal to be processed is a flat field of one color at a constant intensity. Thus, the luminance signal and both chrominance signals are constant (DC) values. However, due to the fact that the chrominance signals are modulated in quadrature by the color subcarrier, the resulting composite video signal is not DC, but rather shows sinusoidal oscillations at the color-subcarrier frequency, as shown in Fig. 2(b). Consequently, only those samples that have the same phase as the current pixel may be used in the prediction. Thus, our prediction scheme accounts for the phasing of the color information in the video signal at the expense of using pixels which are farther away from the current pixel than the nearest neighbors.

Fig. 2(a) shows the prediction used in the hardware implementation of our DVQ algorithm in relation to the tiling of the pixels by the vector quantizer. The NTSC video signal is sampled at four times the color-subcarrier frequency (14.31 818 MHz). The vectors for VQ are tiles of 4×1 samples. Note that, as shown in Fig. 2(b), consecutive intrafield lines of NTSC video have a phase difference of 180°.

IV. The VAMPIRE Chip

The Vector-quantizing Associative Memory Processor Implementing Real-time Encoding (VAMPIRE) is a special-purpose, digital associative memory designed for video-rate vector quantization. The details of the design and operation of this chip are found elsewhere [18], so only a brief overview is given here. Fig. 3 shows the general structure of the VAMPIRE chip.

The VAMPIRE chip is designed to quantize vectors at video rates. The input to the chip is 32 bits representing a 4-D vector with each vector component having 8 bits of resolution. Since these vectors are composed of four video samples, the designed throughput is that of the NTSC colorburst (3.579 545 MHz, or one vector every 280 ns). Each VAMPIRE chip holds 32 codewords. The chips can be operated alone (for codebooks of 32 or less codewords) or can be linked together to accommodate codebooks of greater than 32 codewords.

TABLE I
Example Absolute-Distance Calculation Using 3-bit Values

(i)	Original Problem:	$\|011 - 101\|$
(ii)	Complement the larger and add:	$011 + 010 = 101$
(iii)	Complement the result:	$101 \rightarrow 010$

The VAMPIRE chip calculates the l_1 metric (also known as absolute distance or city-block distance) between the input vector and each of the 32 codewords stored in its memory. These distortion calculations are done digitally and in parallel by 256 computation cells (8 computation cells for each of the 32 codewords, see Fig. 3(a)). A priority encoder selects the codeword with the lowest distortion and places the address on the output bus. Additionally, the distortion is placed on a wired-NOR compare bus. This bus is used to compare each chip's minimum internal distortion to the overall minimum distance as broadcast among chips when several chips are connected together for codebooks of greater than 32 codewords. Each chip "disqualifies" itself if it doesn't hold the winning codeword; the address of the winning codeword is then placed on the address bus.

Fig. 3(a) shows the basic layout of the VAMPIRE chip. The computational cells, CC_{ij}, receive the 32-bit input vector, VECTOR_IN(31:0), and calculate the absolute distance between the input vector and each stored codeword. The winning codeword, which is the codeword with the smallest absolute distance, is determined by the computation cells in conjunction with the priority-encoder circuitry. This smallest distance is output to the compare lines while the address of the winning codeword is output to the ADDR_OUT lines.

A. Absolute-Distance Calculation

The algorithm for calculating the absolute-distance metric is as follows. Let C_i^j be vector component i of codeword j and I_i be vector component i of the input vector. The absolute distance, D^j, for codeword j is

$$D^j = \sum_i D_i^j \tag{1}$$

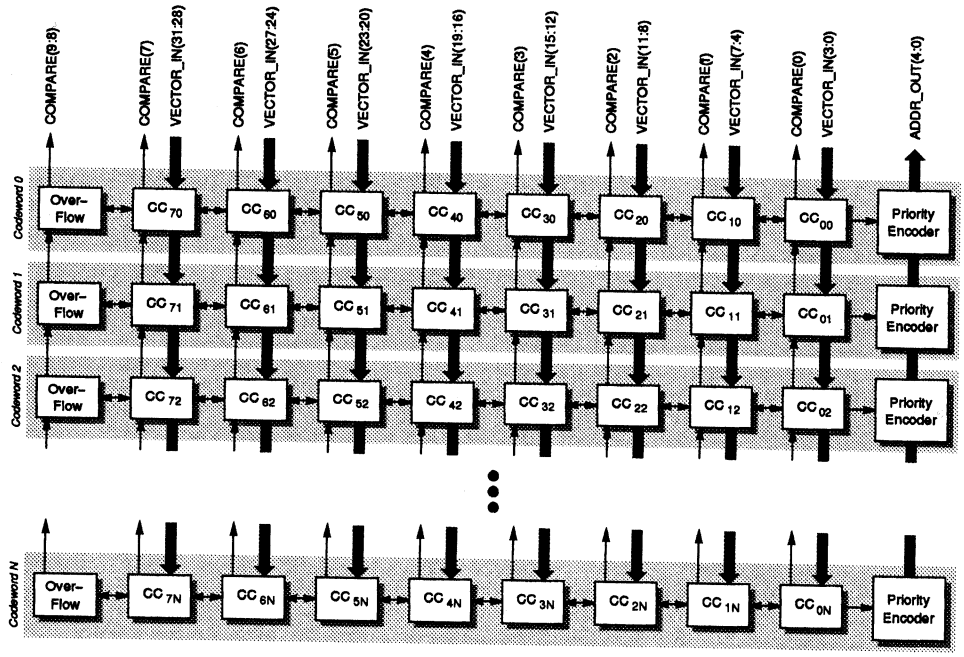
where

$$D_i^j = |C_i^j - I_i| \tag{2}$$

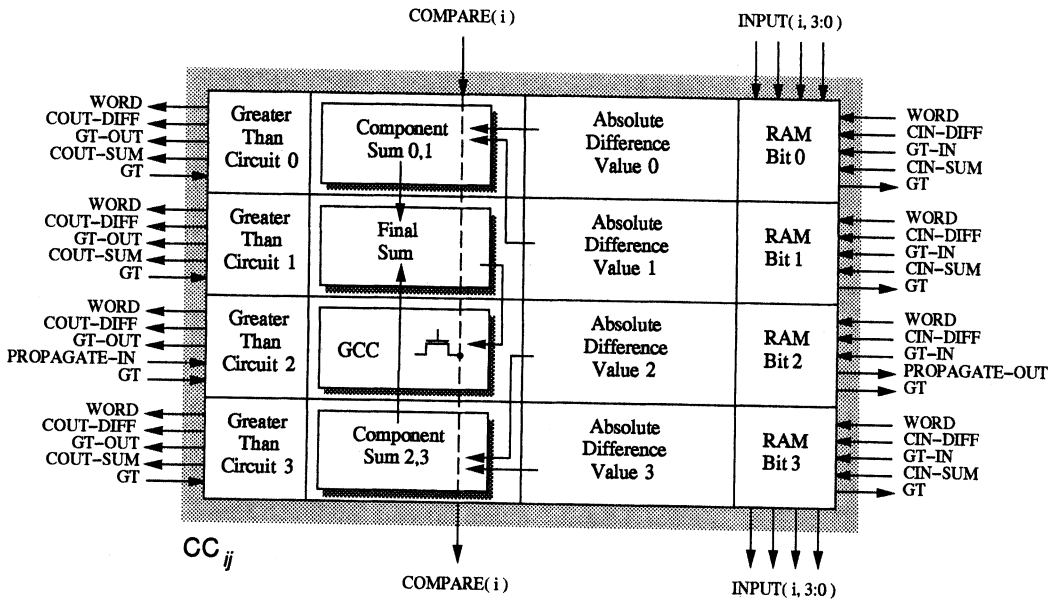
The method for calculating D_i^j is as follows. First, determine which value is larger, C_i^j or I_i. Form the 1's complement of this larger value and add it to the other value. Then 1's complement the result. For an example of the calculation, see Table I. Although this approach is unconventional, it is easily implemented in silicon and involves less layout area than other methods involving 2's complement arithmetic [18].

B. The Computation Cell

Fig. 3(b) is a detailed diagram of the computation cell for bit i of codeword j. The 4 bits of RAM of this computation cell represent the same bit position of the 4 vector components of codeword j. Thus, 8 of these computation cells are placed side by side, as in Fig. 3(a), to form a complete codeword consisting of four 8-bit vector components.



(a)



(b)

Fig. 3. General structure of the VAMPIRE chip. (a) Detailed floorplan showing $N = 32$ codewords. (b) Structure of the computation cell, CC_{ij}, for bit i of codeword j.

For each bit of the computation cell, a greater-than circuit determines whether the stored codeword is greater than the input vector (signals INPUT(i,3:0) in Fig. 3(b)). This calculation cascades between adjacent computation cells, via GT_IN and GT_OUT, from the least-significant to the most-significant bit of the codeword. At the most-significant bit, the final greater-than result GT is the fed back to all the computation cells of the codeword. Using the GT information, the four absolute-difference values are calculated for the codeword. As discussed above, this calculation involves 1's complements and an addition. Carries are cascaded between computation cells (CIN-DIFF and COUT-DIFF).

The four absolute-difference values are added together by the two component sum circuits and the final sum circuit shown in Fig. 3(b). The final sum is the absolute-distortion metric which is then passed to the global compare circuit (GCC).

C. The Global Compare Circuit

The GCC determines which codeword on the chip has the smallest absolute distortion. Again, this calculation is done bit-wise in the computation cells. Fig. 4 shows the GCC of each computation cell. The internal compare lines

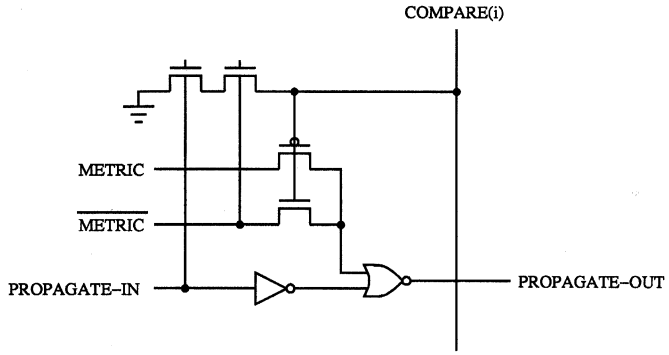


Fig. 4. The GCC within a computation cell of the VAMPIRE chip. METRIC is one bit of the absolute-distance metric.

form a wired-NOR bus. The PROPAGATE signal ripples from the most-significant to the least-significant bit of the final absolute distance. As long as PROPAGATE-IN is asserted, then that particular codeword has not been eliminated from the winner-selection process. For a given bit, the COMPARE line is driven low if any codeword's absolute distance is low in that bit position. If the COMPARE line is low for that bit, and the codeword's distance is high, then the codeword is eliminated from the winner-selection process and PROPAGATE-OUT is driven low. Otherwise, PROPAGATE-OUT remains high and the processing continues to the next bit position.

Fig. 5 shows one bit-slice of the internal compare bus and each codeword's GCC for that bit. When the winner-selection process has completed rippling through each bit, the final, winning absolute distance remains on the internal COMPARE bus. The interchip-winner-selection circuit extends the wired-NOR COMPARE bus to external compare pins so that a winner may be determined between multiple chips. For a given chip, the interchip-winner-selection circuit functions as follows. If an internal COMPARE line is low, then the chip drives the external COMPARE pin low. If the internal COMPARE line is high, then the pin becomes an input. At any progressively lower bit positions, a chip is disqualified from competition if the internal and external COMPARE states differ. Disqualification occurs by setting the $\overline{\text{CHIP-VALID-OUT}}$ line high. This $\overline{\text{CHIP-VALID-OUT}}$ line ripples through each bit of the internal COMPARE bus.

D. Fabrication and Testing

Twelve VAMPIRE chips were fabricated by the MOSIS service using a 2 μm CMOS n-well process. Preliminary low-speed testing indicated that the chip functioned correctly on a codebook of 32 codewords. However, there was a minor design error that prevented connecting chips to expand processing to codebooks of more than 32 codewords: the NOR gate of the interchip-winner-selection circuitry (see Fig. 5) was inadvertently fabricated as a NOT, thus preventing correct operation of the external COMPARE bus.

Another problem was found during high-speed testing: the maximum processing time of the chip was determined to be 380 ns (35% slower than the desired speed of 280 ns). It was proposed that the pull-down transistors on the internal compare bus (see Fig. 4) in the GCC's were too small to pull these lines down fast enough.

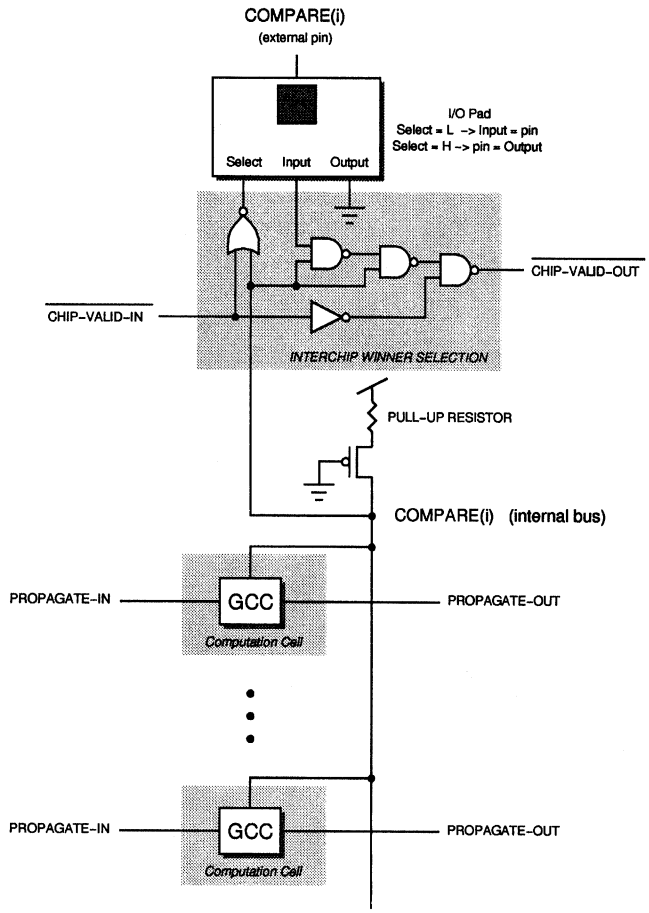


Fig. 5. One bit-slice of the winner-selection circuitry of the VAMPIRE chip. Each codeword's GCC connects to the internal COMPARE bus. The interchip-winner-selection circuit extends the internal COMPARE bus off the chip to select a winning codeword from multiple chips.

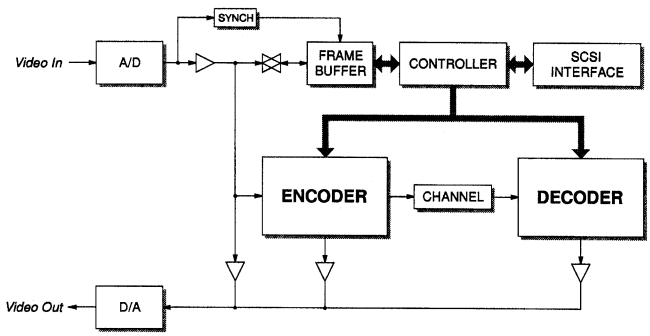


Fig. 6. Block diagram of the DVQ system

Another batch of VAMPIRE chips was fabricated. In the design of these new chips, the NOR-gate error of the interchip-winner-selection circuitry was corrected, and the pull-down transitors in the GCC's were doubled in size. It has been verified that these new chips correctly connect together to implement codebooks of more than 32 codewords. However, doubling the size of the pull-down transistors did not fix the speed inadequacy. Further investigation is underway to determine why this fix did not work, although it is suspected that the problem may lie in the off-chip interface circuitry that drives the external pins. If this is the case, it may be

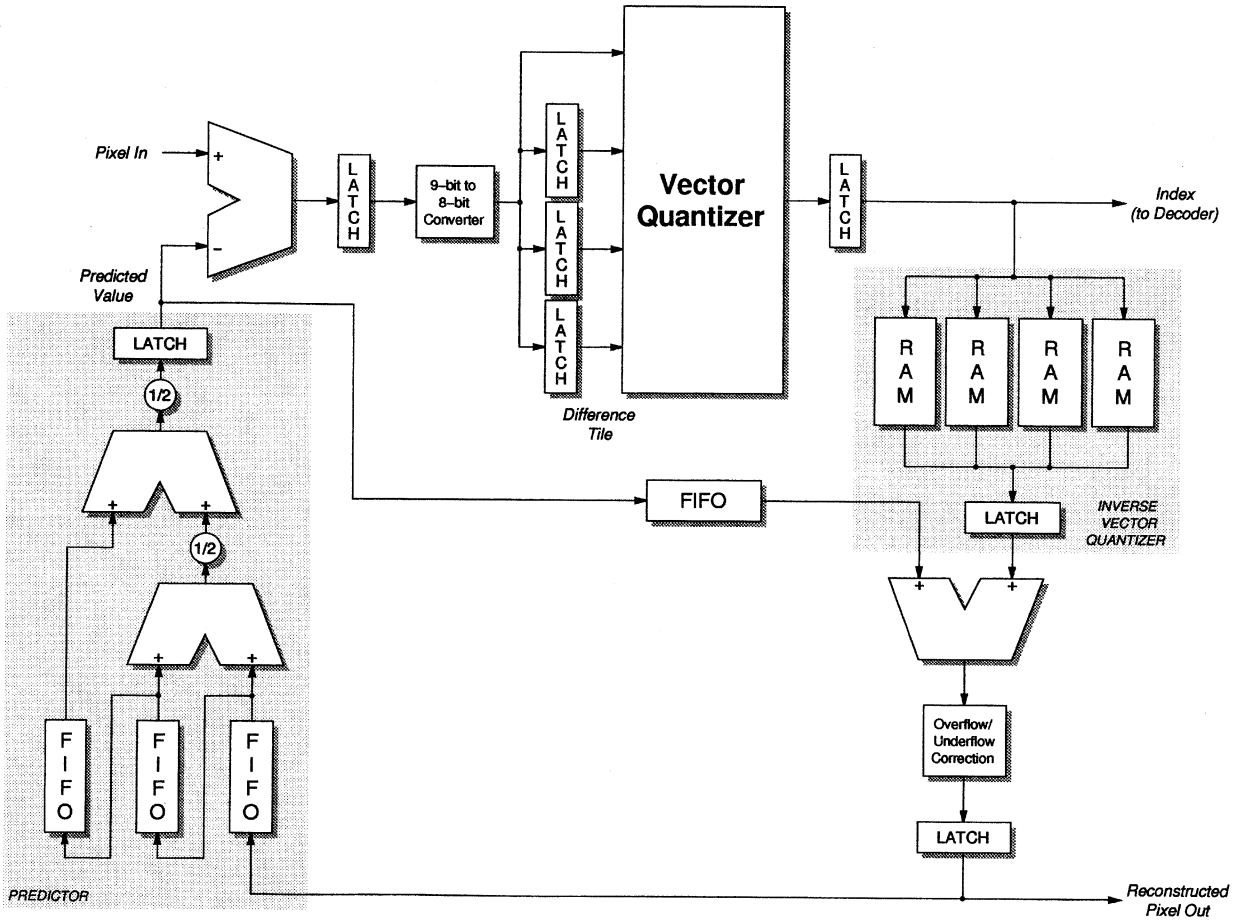


Fig. 7. Encoder block diagram

necessary that the chips be fabricated in 0.5 μm CMOS; this was not done before because of the greater fabrication cost of this faster technology. However, the slow speed of the VAMPIRE chips does not impede their usefulness in our DVQ system. Indeed, in the next section, we present a workaround that enables the DVQ system to operate in real-time with a 128-codeword codebook.

V. System Details

Fig. 6 shows the organization of the hardware constructed to implement our DVQ video-compression algorithm. The system is composed of the following logical units: controller and interface; A/D converter and frame buffer; encoder; and decoder. Discussion of these units follows. To meet the speed requirements of real-time operation, FAST Advanced Schottky TTL logic was used in most of the units. Figs. 9–11 are photographs of the DVQ system.

A. Controller and Interface

The controller routes the flow of data between the units of the system. In addition, it drives a SCSI-bus interface which allows communication with a PC. The SCSI bus is used to transfer commands to the system and also to upload or download single frames of video.

B. Sampling and Frame Buffer

The A/D converter samples the incoming video at four times the color-subcarrier frequency (14.31 818 MHz). In addition to the active-video portion of the signal, all horizontal and vertical synchronization pulses are sampled and processed. Thus, one frame of sampled video consists of 910 $\times$ 526 samples.

The frame buffer consists of a 512 $\times$ 8 dynamic RAM and associated addressing and synchronization circuitry. The frame buffer is used to store a frame of video from the A/D converter for output through the SCSI bus. Additionally, it can receive a frame from the SCSI bus and output it repeatedly to the D/A converter or to the encoder. The synchronization circuitry analyses the incoming video and provides the frame buffer with information indicating the starting and stopping points of a frame.

C. Encoder

Fig. 7 shows a block diagram of the architecture of the encoder. The encoder consists of the following units: predictor, vector quantizer, and inverse vector quantizer. The encoder runs in real-time without buffering, processing input pixel samples and outputting reconstructed pixels at the sample rate (14.31 818 MHz).

The predictor calculates predicted values at the sample rate. At this rate, the predictor has 69 ns to generate each

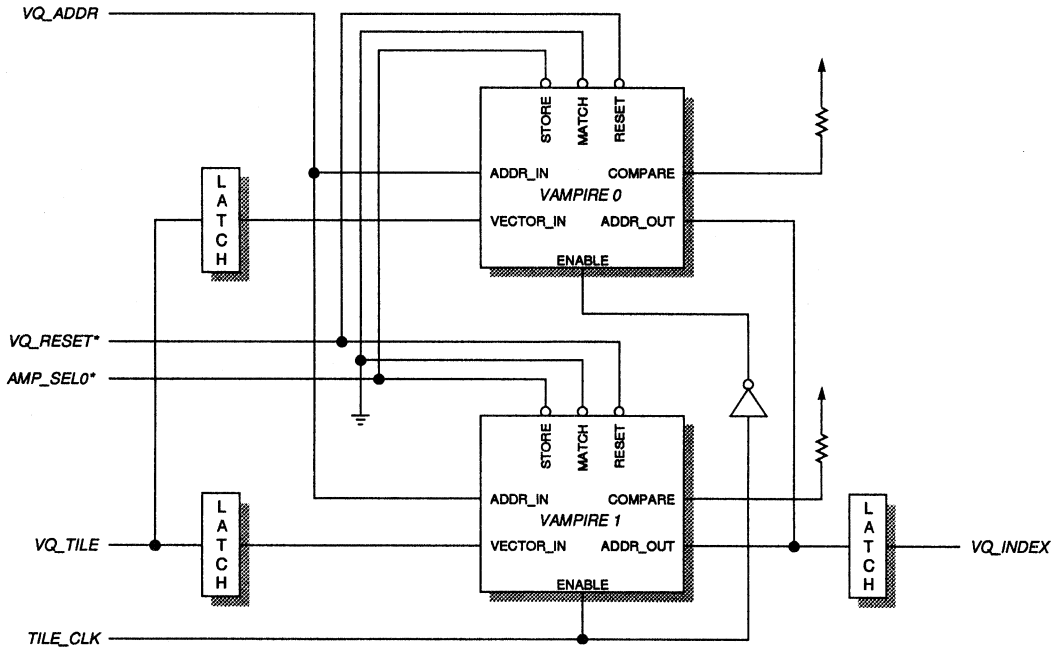


Fig. 8. Schematic diagram of the vector quantizer showing the VAMPIRE chips in "flip-flop" configuration.

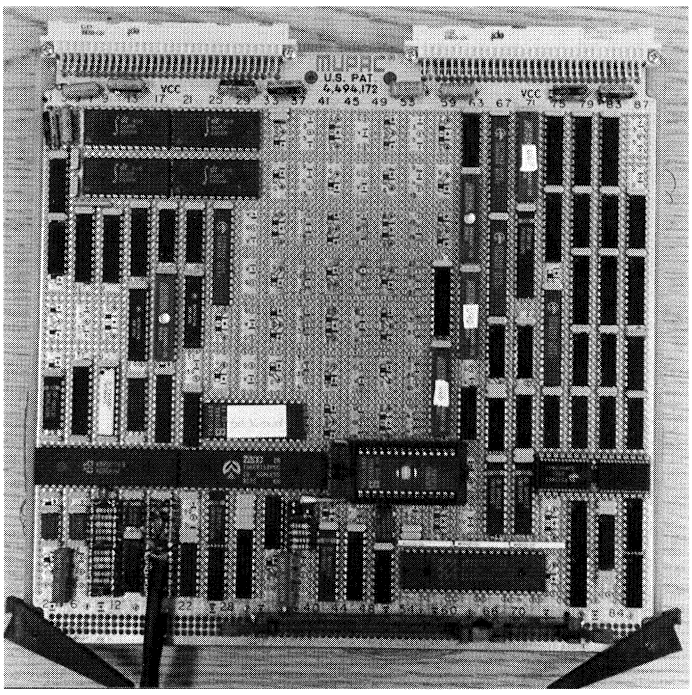


Fig. 9. Photograph of board 1 of the DVQ system. This board contains the A/D converter, the SCSI interface, and the system controller. The predictor resides on the upper-right of the board, and the inverse VQ RAM's are on the upper-left.

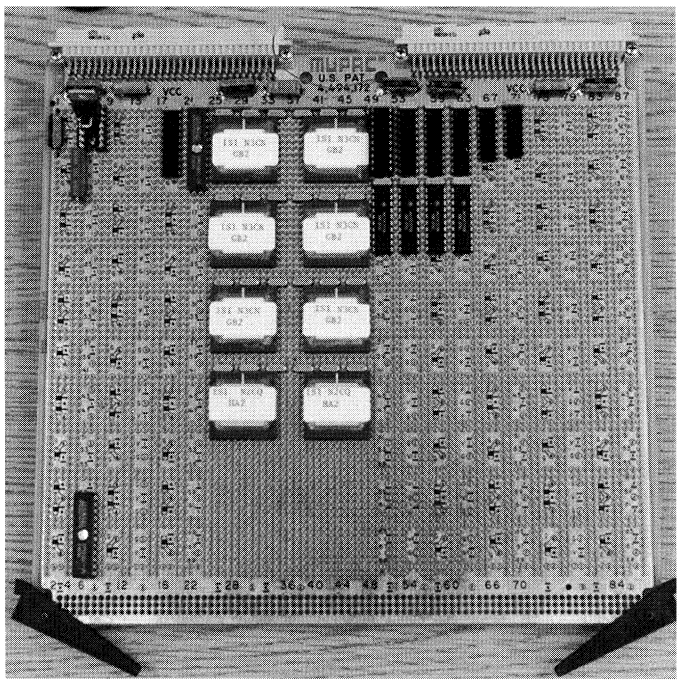


Fig. 10. Photograph of board 2 of the DVQ system. This board does all the VQ processing. The VAMPIRE chips are the large chips in the middle. Processing "flip-flops" between the left and right columns of the VAMPIRE chips, effectively doubling the allowable VQ-processing time.

predicted value. The prediction process involves extracting three 8-bit reconstructed pixel values from FIFO memory and performing two 8-bit additions and two divisions-by-2. High-speed, CMOS FIFO's (size = 2 048 × 9, access time = 10 ns) are used. Each 8-bit addition is accomplished with two 74F283 4-bit adders. The divisions result from ignoring the least-significant bit.

The inverse vector quantizer is composed of 4 dynamic RAM's, one for each of the four vector components. The VQ codebook is stored in both the vector quantizer and the inverse vector quantizer (This storage is done via the SCSI bus by circuitry not shown in Fig. 7). The inverse vector quantizer performs a table lookup into the codebook given the index value generated by the vector quantizer.

D. Vector Quantizer

As discussed in Section IV, the vector quantizer that has been constructed operates with up to 128 codewords. Since the

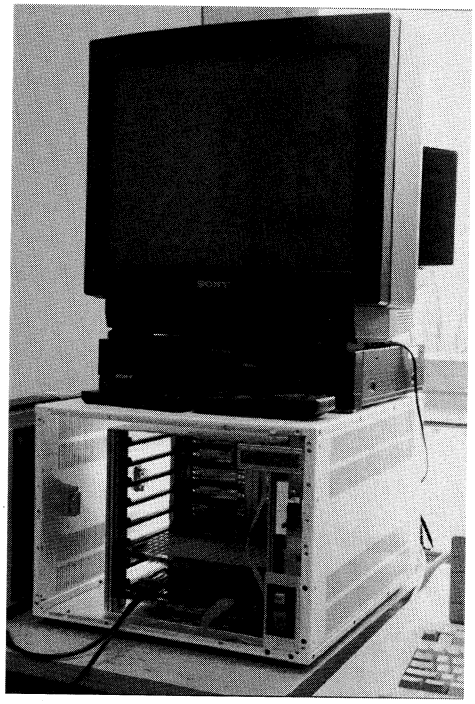


Fig. 11. Photograph of the entire DVQ system.

speed of the VAMPIRE chips is 35% too slow, it was necessary to use two sets VAMPIRE chips in the vector quantizer, as shown in Fig. 8 for the case of a 32-codeword codebook. The processing in Fig. 8 is staggered so that each chip quantizes every-other vector. This "flip-flop" arrangement allows each chip twice the time to process (560 ns instead of 280 ns) while ensuring every vector is quantized. In the current configuration of the DVQ system, two sets of 4 VAMPIRE chips use this "flip-flop" arrangement to implement a codebook of 128 codewords and the 7-bit VQ indices are output at 3.57 954 MHz.

The training of the VQ codebooks was done offline using the FSCL ANN algorithm. Six frames of video were sampled using the frame buffer and were uploaded via the SCSI bus to a Sun SPARC workstation for training. The resulting codebooks were then downloaded to the DVQ architecture via the SCSI bus.

E. Decoder

As seen in Fig. 1, the architecture of the decoder is simply a replication of a subset of the encoder architecture. In the absence of channel errors, the output of the decoder is the same as the reconstructed values found within the encoder. Thus, for proof of principle, only the encoder was constructed. The results shown below are generated by the encoder. The decoder is a simple extension of the system hardware, and we plan to construct it in the near future so that tests over an error-prone channel can be performed.

VI. Results

Figs. 12 and 13 present the results obtained during real-time operation of our hardware DVQ implementation on NTSC video. Fig. 12(a) shows one frame from the original video sequence. Fig. 12(b) shows that same frame from the output video sequence, which has been compressed and reconstructed by the DVQ hardware using 32 codewords (a compression ratio of 6.4:1, channel rate of 17.9 Mbits/s). Some "blocky" effects on the edges, along with some color distortion, are visible. Fig. 13 shows these same results obtained for another frame of the video sequence. Table II shows how the image distortion varies with the compression rate by presenting MSE values calculated by computer simulations of the DVQ algorithm for several codebooks of varying size.

TABLE II
MSE Values from Computer Simulations of DVQ on Video Image with Codebooks of Varying Size

	Codebook Size			
Image	256	128	64	32
banal*	12.4	20.5	29.3	57.1
fbi	5.5	7.7	11.1	20.2
kate	4.9	7.2	9.5	18.8
laura*	5.5	7.5	10.3	20.6
paris*	4.8	6.7	9.7	15.3
pete	7.2	9.8	14.2	25.9
piano	6.8	9.4	13.8	26.9
rhino*	8.9	12.9	20.0	31.5
rio	9.0	14.1	19.7	47.4
rr_diner*	9.2	13.0	20.2	34.1
snake*	4.8	6.6	9.4	18.6
talk	11.8	18.3	24.7	42.7

*Images used in training the vector quantizer.

TABLE III
Summary of Vampire Chip Characteristics

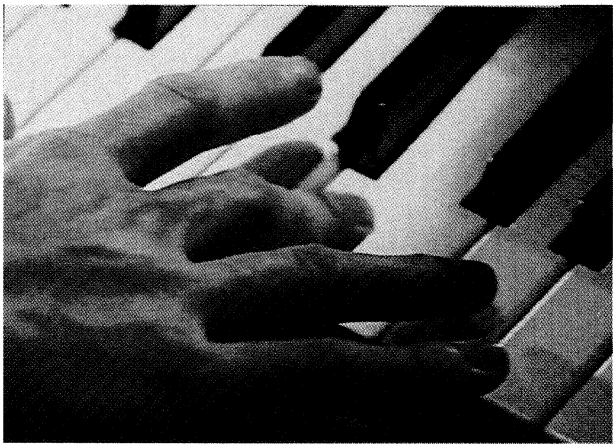
Die size	4.6 × 6.8mm
Technology	2μm CMOS n-well
Vector Rate	2.63×10^6 vectors/s
Encoding delay*	approx. 1μs
Codebook size	32 codewords on one chip; expandable to 256 with 8 chips
Vector dimension	Four 8-bit components
Power supply	5V
Operating Rate**	1.01×10^9 ops/s

*Encoding delay is for the VAMPIRE chip operating in the DVQ architecture

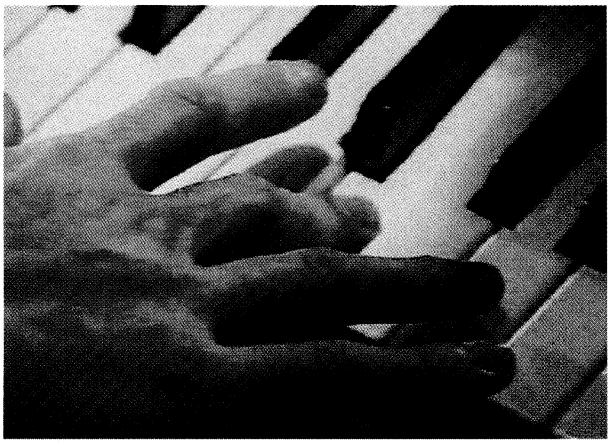
**Indicates number of mathematical operations per second

VII. Conclusions and Future Work

In this paper, we have described our DVQ algorithm, presented a hardware architecture implementing it, and demonstrated real-time operation of this hardware on NTSC video. The heart of this compression system is the VAMPIRE chip, whose characteristics are summarized in Table III. Digital associative memories, such as the VAMPIRE chip, will certainly play a role in bringing VQ to prominence in practical, real-time applications. It should be noted that the vector dimension provided by the VAMPIRE chip, 4 components, may be too small for some applications. Proposals have been made for modifications to the VAMPIRE design to expand this maximum number of vector components [18]. It has



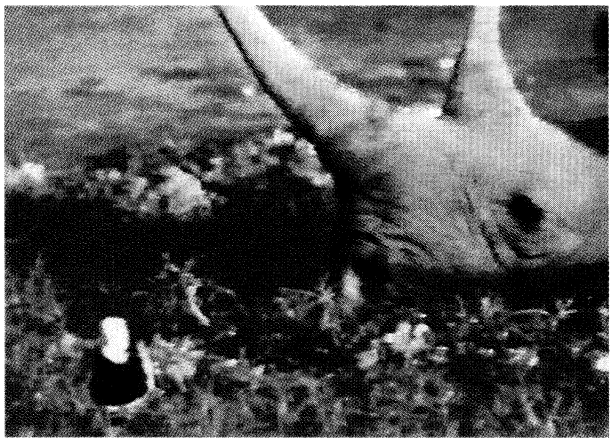
(a)



(b)

Fig. 12. Output of the DVQ hardware on real-time video. (a) Original video frame. (b) Frame from output video compressed using VQ with 32 codewords.

(a)



(b)

Fig. 13. Output of the DVQ hardware on real-time video. (a) Original video frame. (b) Frame from output video compressed using VQ with 32 codewords.

been proposed to cascade several chips together to obtain longer vectors in a fashion similar to the way the current VAMPIRE design allows several chips to connect to form larger codebooks. An alternate strategy calls for a synchronous design in which each vector component is input individually and the distortion is successively calculated and added to an accumulator.

The system implementation presented is here is simple and straightforward: Simplicity and speed was emphasized over flexibility so that real-time operation was achieved. The main goal of this work has been to demonstrate VQ in real time on real video, not to provide a design immediately applicable to commercial broadcasting. Since in this implementation we were limited to 128 codewords, the quality we obtained, although reasonable and as expected from computer simulations, was not sufficient for broadcast applications.

There are several modifications that would improve the picture quality and compression performance. First, compression should be performed on only the active-video portion of the signal. To this end, a digital television chip set, such as the one made by Philips, could be used to strip out the sync information. The sync information should be removed from the signal before compression and then restored to the signal by the decoder. Not only would greater compression be obtained, since only the active video is processed, but also the decompressed signal would be of better quality, since the sync information would not endure the distortion due to quantization. Additionally, the digital TV chips could demodulate the color signals from the luminance. In composite-color video, the color signals appear as high-frequency ''noise'' in the luminance signal. Consequently, the predictor tends to distort this color ''noise,'' despite the compensatory measures taken during the design of the predictor to ensure that prediction is done using only same-phase samples. For better edge and color fidelity, the results presented here indicate that it is imperative that the color signals be demodulated and processed separately. Finally, the predictor should incorporate temporal information in the form of motion estimation. These ideas are the topics of further research which we hope will extend the utility of our architecture.

ACKNOWLEDGMENT

The authors would like to express their sincere thanks to M. Hanes for his photographic assistance and M. Mokheimer for his help with the hardware construction.

REFERENCES

[1] C. E. Shannon, "A mathematical theory of communication," in *Key Papers in The Development of Information Theory,* D. Slepian, Ed. New York: IEEE Press, 1948, pp. 5–18.

[2] K. Dezhgosha, M. M. Jamali, and S. C. Kwatra, "A VLSI architecture for real-time image coding using a vector quantization based algorithm," *IEEE Trans. Signal Processing*, vol. 40, pp. 181–189, Jan. 1992.

[3] S. Panchanathan and M. Goldberg, "A content-addressable memory architecture for image coding using vector quantization," *IEEE Trans. Signal Process.*, vol. 39, pp. 2066–2078, Sept. 1991.

[4] W.-C. Fang, B. J. Sheu, O. T.-C. Chen, and J. Choi, "A VLSI neural processor for image data compression using self-organization networks," *IEEE Trans. Neural Networks*, vol. 3, pp. 506–518, May 1992.

[5] G. T. Tuttle, S. Fallahi, and A. A. Abidi, "A low-power analog CMOS vector quantizer," in *Proc. IEEE Data Compression Conf.* J. A. Storer and M. Cohn, Eds. Snowbird, UT: IEEE Computer Society Press, 1993, pp. 410–419.

[6] R. M. Gray, "Vector quantization," *IEEE ASSP Mag.*, vol. 1, pp. 4–29, Apr. 1984.

[7] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression (Kluwer international series in engineering and computer science).* Norwell, MA: Kluwer Academic, 1992.

[8] J. E. Fowler, M. R. Carbonara, and S. C. Ahalt, "Image coding using differential vector quantization," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 3, pp. 350–367, Oct. 1993.

[9] C. W. Rutledge, "Vector DPCM: Vector predictive coding of color images," in *Proc. IEEE Global Telecommun. Conf.*, Sept. 1986, pp. 1158–1164.

[10] D.-M. Chiang and L. C. Potter, "Minimax non-redundant channel coding for vector quantization," in *Proc. Int. Conf. on Acoustics, Speech, and Signal Process.*, April 1993, pp. V-617–V-620.

[11] K. A. Zeger and A. Gersho, "Pseudo-Gray coding," *IEEE Trans. Commun.*, vol. 38, pp. 2147–2158, Dec. 1990.

[12] P. A. Ramamoorthy, B. Potu, and T. Tran, "Bit-serial VLSI implementation of vector quantizer for real-time image coding," *IEEE Trans. Circuits Syst.*, vol. 36, pp. 1281–1290, Oct. 1989.

[13] Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design," *IEEE Trans. Commun.*, vol. 28, pp. 84–95, Jan. 1980.

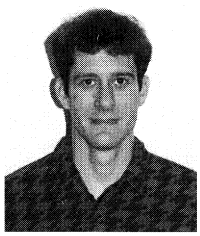
[14] S. C. Ahalt, P. Chen, and A. K. Krishnamurthy, "Performance analysis of two image vector quantization techniques," in *Proc. Int. Joint Conf. Neural Networks*, vol. I, June 1989, pp. 169–175.

[15] S. C. Ahalt, A. K. Krishnamurthy, P. Chen, and D. E. Melton, "Competitive learning algorithms for vector quantization," *Neural Networks*, vol. 3, no. 3, pp. 277–290, 1990.

[16] A. K. Krishnamurthy, S. C. Ahalt, D. Melton, and P. Chen, "Neural networks for vector quantization of speech and images," *IEEE J. Selected Areas in Commun.*, vol. 8, no. 3, pp. 1449–1457, Oct. 1990.

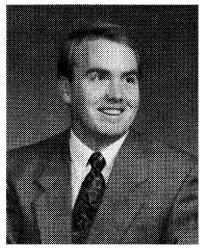
[17] H.-M. Hang and J. W. Woods, "Predictive vector quantization of images," *IEEE Trans. Commun.*, vol. 33, pp. 1208–1219, Nov. 1985.

[18] K. C. Adkins, *The VAMPIRE Chip: A Vector-quantizer Associative Memory Processor Implementing Real-time Encoding.* Ph.D. dissertation, The Ohio State University, 1993.



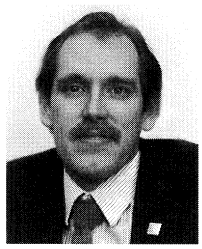
James E. Fowler, Jr. received the B.S. degree in computer and information science engineering and the M.S. degree in electrical engineering from the Ohio State University in 1990 and 1992, respectively. He is currently pursuing the Ph.D. degree in electrical engineering also at the Ohio State University.

He has worked as a Computer Programmer at Intergraph, Huntsville, AL, and at the Ohio Department of Transportation, Columbus. Since 1990, he has been a Graduate Researcher at the Ohio State University. His research interests include image and signal processing algorithms and hardware, video technology, and neural network algorithms for signal processing.



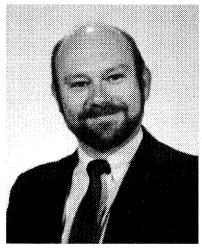
Kenneth C. Adkins (S'88–M'88) received the B.S. degree in 1988, the M.S. degree in 1991, and the Ph.D. degree in 1993, all in electrical engineering from the Ohio State University. His doctoral work focused on the development of VLSI associative memories for real-time video data compression using vector quantization.

He is presently with National Semiconductor, Santa Clara, CA where he works in the Memory Products Division developing application-specific FLASH memory circuits and high-performance, low-voltage EEPROM devices.



Steven B. Bibyk (S'78–M'80) received a combined B.S. and M.S. degree in 1980 and the Ph.D. degree in 1983, all in electrical engineering and applied physics, from Case Western Reserve University.

He joined the Department of Electrical Engineering of the Ohio State University in 1984 where he is presently an Associate Professor. His research has been in the area of device operation for nonvolatile memories and the reliability and system limitations of CMOS VLSI at liquid nitrogen temperatures. During 1991-1992, he was a visiting scientist at the NASA Lewis Research Center and the Ohio Aerospace Intstitute where he worked on electronics for Space Communication Systems. At present, he is working on the development of VLSI associative memories for pattern recognition and demodulation of digital communication signals.



Stanley C. Ahalt (S'77–M'79) received the B.S.E.E. and M.S.E.E. degrees from Virginia Polytechnic Institute in 1978 and 1980, and the Ph.D. degree from Clemson University in 1986.

He was a member of the Technical Staff at Bell Telephone Laboratories from 1980 to 1981. He is currently an Associate Professor with the Department of Electrical Engineering at the Ohio State University. His research interests are image processing, special purpose computer architectures, neural networks, and parallel computing.