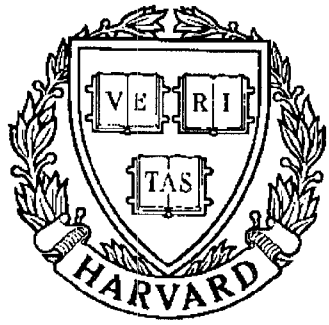
TECHNICAL RESEARCH REPORT

SYSTEMS RESEARCH CENTER



Supported by the National Science Foundation Engineering Research Center Program (NSFD CD 8803012), Industry and the University

VLSI Implementation of a Tree Searched Vector Quantizer

by R. Kolagotla, S.S. Yu, and J.F. JáJá

TR 90-74r1

To appear in the
IEEE Trans. on Signal Processing
1993

VLSI Implementation of a Tree Searched Vector Quantizer [1]

Ravi K. Kolagotla, Shu-Sun Yu and Joseph F. JáJá

Department of Electrical Engineering
Systems Research Center, and
Institute For Advanced Computer Studies
University of Maryland, College Park, MD 20742.

Abstract

The VLSI design and implementation of a Tree Searched Vector Quantizer is presented. The number of processors needed is equal to the depth of the tree. All processors are identical and data flow between processors is regular. No global control signals are needed. The processors have been fabricated using $2\mu m$ N-well process on a $7.9mm \times 9.2mm$ die. Each processor chip contains 25,000 transistors and has 84 pins. The processors have been thoroughly tested at a clock frequency of 20 MHz.

1 Introduction

Compression of images has been the subject of extensive studies due to its many applications. This has led to the development of several important techniques such as predictive coding, block transform coding, vector quantization (VQ), and subband coding [1]. Our goal here is the development of a hardware compression system to efficiently compress LANDSAT images for storage purposes. This system must be capable of handling an input data rate of 320 Mbits/sec.

In a block structured image coding scheme, an input image is segmented into blocks of equal size and each block is quantized independently. Vec-

[1]This work was partially supported by Ford Aerospace Corporation, MIPS contract # 121.45, and the Systems Research Center, contract # OIR–85–00108

tor Quantization provides the best performance among all block structured image coding schemes for a given blocksize and bit-rate. In order to make best use of inter-pixel correlation, large dimension vectors must be quantized. However, it is difficult to design and implement reasonable size codebooks for large dimension vectors. To solve this problem, a combined VQ-DCT-SQ image coding algorithm [2] was developed in the University of Maryland's Communications and Signal Processing Laboratory.

In a conventional block transform image coding system, each block is operated upon by a linear transformation, such as the 2-D Discrete Cosine Transformation (DCT), to remove inter-pixel correlation within the block. The transform coefficients are then quantized using scalar quantizers. In the combined VQ-DCT-SQ coding scheme, a high bit rate scalar quantizer is used to quantize the error vectors[2] from a low bit rate vector quantizer. A 2-D DCT is used to energy compact the error vectors prior to scalar quantization. In order to exploit the classified nature of vector quantization, we can use a different scalar quantizer for each codevector; each scalar quantizer optimized for its particular codevector. The performance improvement due to using different scalar quantizers for different codevectors is a strong function of the image statistics.

In this paper we describe the architecture, design and VLSI implementation of a Tree Search VQ (TSVQ) processor that will be used in the VQ-DCT-SQ system.

2 Architecture

Given an input vector $\mathbf{x}$, a VQ encoder chooses a reproduction vector $\hat{\mathbf{x}}$ from a predetermined set of reproduction vectors (or codevectors) that is closest to the input vector relative to a certain distortion measure. In a binary TSVQ, the codebook is organized in a tree structure. The input vector is compared

[2]Difference between the input vectors and their corresponding codevectors.

with two codevectors at each node. Based on this comparison, one of the two branches is chosen and the codebook search space is reduced in half. This process is repeated until a leaf node is reached. Let $\mathbf{x} = (x_1, \ldots, x_L)^T$ represent the L-dimensional input vector, and $\mathbf{c_1} = (c_{1,1}, \ldots, c_{1,L})^T$, and $\mathbf{c_2} = (c_{2,1}, \ldots, c_{2,L})^T$ represent the two vectors in the codebook of a given node. The processing performed at each node is reduced to testing the condition:

$$d(\mathbf{x}, \mathbf{c_1}) \geq d(\mathbf{x}, \mathbf{c_2}), \tag{1}$$

where $d(\mathbf{x}, \mathbf{c_i}), \mathbf{i} = 1, 2$ are the distortion measures. For the general case of the weighted mean-squared error distortion,

$$d(\mathbf{x}, \mathbf{c_i}) = (\mathbf{x} - \mathbf{c_i})^T \mathbf{W} (\mathbf{x} - \mathbf{c_i}), \quad \mathbf{i} = 1, 2,$$

where $\mathbf{W}$ is the weighting matrix. Equation (1) can be expressed as:

$$(\mathbf{x} - \mathbf{c_1})^T \mathbf{W} (\mathbf{x} - \mathbf{c_1}) - (\mathbf{x} - \mathbf{c_2})^T \mathbf{W} (\mathbf{x} - \mathbf{c_2}) \quad \geq 0 \tag{2}$$

If equation (2) is satisfied, the input vector $\mathbf{x}$ is closer to codeword $\mathbf{c_2}$. Otherwise $\mathbf{x}$ is closer to $\mathbf{c_1}$. We expand equation (2) to obtain [3]:

$$\sum_{j=1}^{L} \{\alpha_j x_j\} + \beta \quad \geq 0 \tag{3}$$

where $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_L) = 2(\mathbf{c_2} - \mathbf{c_1})^T \mathbf{W}$, and $\beta = \mathbf{c_1}^T \mathbf{W} \mathbf{c_1} - \mathbf{c_2}^T \mathbf{W} \mathbf{c_2}$. For the special case of the mean-squared error distortion measure, $\mathbf{W} = \mathbf{I}$, and hence $\alpha_j = 2(c_{2,j} - c_{1,j})$, and $\beta = \sum_{j=1}^{L} (c_{1,j}^2 - c_{2,j}^2)$.

Instead of using the raw codebook online, we can determine these α and β coefficients off-line and store them in memory chips[3]. This algorithm is

[3]Some applications use a weighting matrix $\mathbf{W}(\mathbf{x})$ that depends on the input vector $\mathbf{x}$. Equation (3) is still valid in this case, but a preprocessor is needed to compute the α and β coefficients in real-time. [**Note added in proof:** We recently learned of a similar architecture developed by Yan and McCanny [4]. They report that equation (3) is valid for the case of the Itakura-Saito distortion measure as well.]

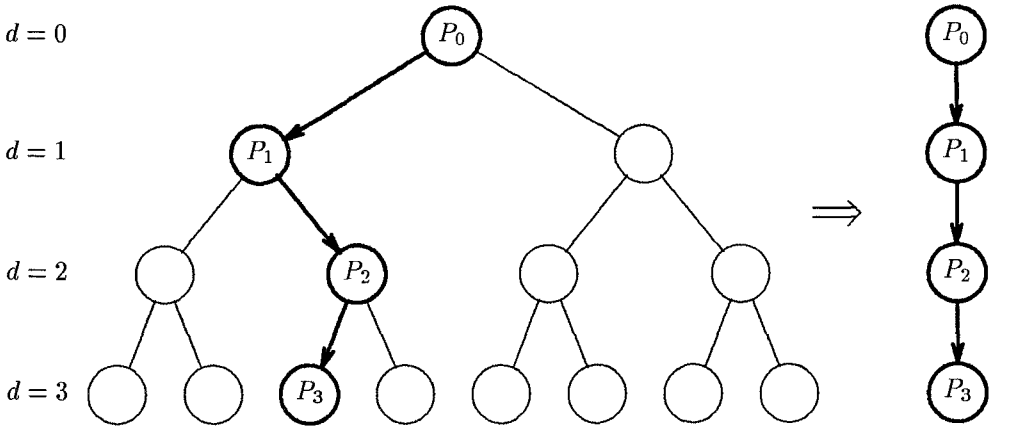


Figure 1: Traversal of a binary tree of depth 4, and its mapping onto a linear array of processors.

based on Binary Hyperplane Testing [5]. Directly implementing equation (1) requires $2(L^2 + L)$ multiplications, $2(L^2 - 1)$ additions and $L^2 + L$ words of memory storage, while implementing equation (3) requires only L multiplications, L additions, and $L + 1$ words of memory storage[4].

The computations performed by a TSVQ can be viewed as finding a path from the root to a leaf in a binary tree. While traversing a binary tree, only one node is encountered at each level. Hence, the computations at each level can be performed by a single processor. A tree of depth d can be mapped onto a linear array of d processors as shown in Fig. 1.

Fig. 2 shows the architecture of a TSVQ using d processors. The coefficients necessary for each processor's computations are stored in memories and will in general depend on the distortion measure used. Processor P_i adds the results of its computations to a partial index datapath and generates a Go signal to initiate processing by processor P_{i+1}. This Go signal is used to

[4]The same conclusion was independently reached by Tom Lookabaugh [6] and Wai-Chi Fang *et. al.* [7].

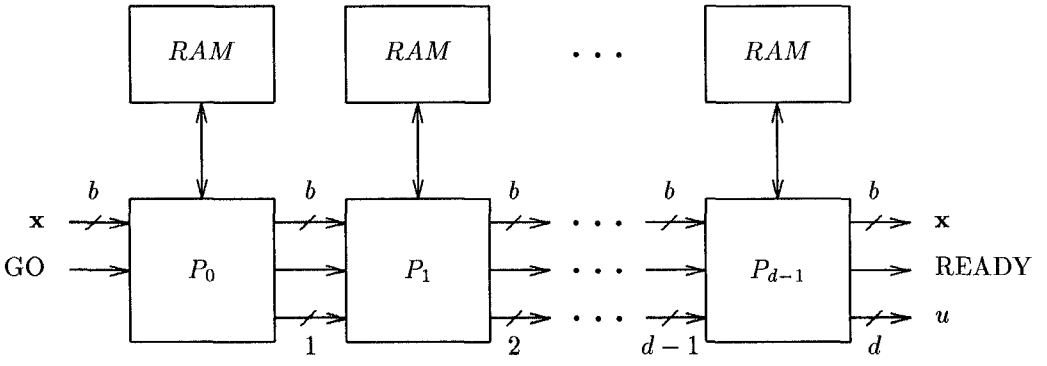


Figure 2: Systolic architecture for computing TSVQ. Each processor adds its partial index to the index data-path, and generates a control signal to initiate processing by its neighbor down the tree. No global control signals are needed.

reset the accumulator in processor P_{i+1}. The final processor, P_{d-1}, returns the complete index u. The size of the memory is different for different processors. The first processor needs a memory of $L+1$ words to store β' and the L components of α_j. Processor P_{i+1} needs twice as much memory as processor P_i. The last processor needs a memory of $2^{d-1}(L+1)$ words. The throughput of this scheme is one L-dimensional vector per L clock cycles.

Each TSVQ Processor performs the computations stated in equation (3). Its output is a '0' if equation (3) is satisfied and a '1' otherwise. Our implementation of the TSVQ processor uses the pipelined parallel multiplier developed by McCanny and McWhirter [8]. We do not need a comparator unit in the processor. The most significant bit (MSB) of the accumulated products directly represents the processor's output.

Fig. 3 shows a block diagram of the TSVQ processor. Input data is skewed and all internal operations are performed in a bit-skewed word-parallel mode. The multiplier takes two b-bit numbers α_j and x_j, and a $2b$-bit number[5] β', and returns a $2b$-bit number $p_j = \alpha_j x_j + \beta'$. The bits of

[5]We define $\beta' = \beta/L$ and add it during each of the L multiplication steps. This can

$p_j = p_{j,2b}, p_{j,2b-1}, \ldots, p_{j,1}$ are available in a skewed fashion, least significant bit (LSB) first. The latency of the multiplier depends on the bit position; it is b for the LSB bit $p_{j,1}$, and $3b$ for the MSB bit $p_{j,2b}$. The accumulator must have a precision of

$$n = 2b + \lceil \log L \rceil$$

bits, to prevent overflow when L $2b$-bit numbers are added together. The output of the multiplier is sign extended by $\lceil \log L \rceil$ bits and is directly applied to the accumulator.

The accumulator consists of a linear array of cells, and operates on skewed input data as shown in Fig. 4. Each cell consists of a full adder and three latches. Carry is propagated to the neighboring cell and sum is stored within the cell. The accumulator computes

$$A = \sum_{j=1}^{L} p_j,$$

and returns the sign of A. The sign of A is available at the carry output pin of the last cell in the accumulator array. It is denoted by L/R in Fig. 4. A Reset signal is generated once every L clock cycles. Reset is propagated along the array and each cell is reset in turn. This allows the next set of L numbers to be accumulated immediately after the last number of the current set is applied to the accumulator. The latency of the accumulator is $n + L$ clock cycles. This is the number of clock cycles between the time $p_{1,1}$ is applied to cell A_1 and the time L/R is ready at cell A_n. Hence, the latency of each processor is

$$b + n + L = 3b + \lceil \log L \rceil + L.$$

For example, if the word size $b = 8$, and the vector dimension $L = 64$, we have a latency of 94 clock cycles.

---

be done without any additional hardware and eliminates the need for a comparator unit to compare the accumulated sums with β.

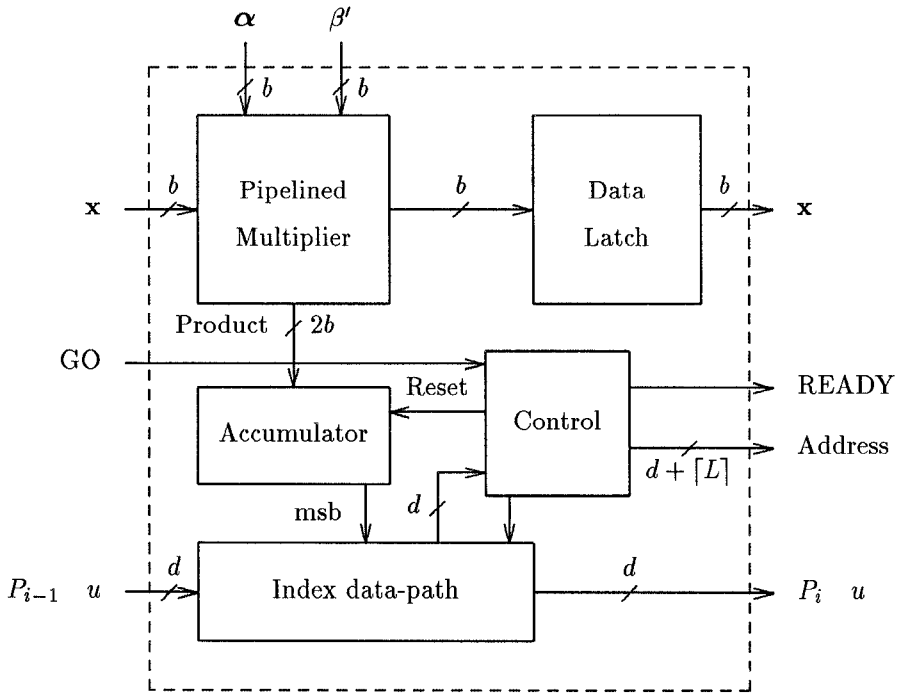


Figure 3: Detailed block diagram of each processor. Each processor's READY output must be connected to the GO input of its neighbor. Only the most significant b bits of β' are applied to the processor. The least significant b bits are set to zero internally.

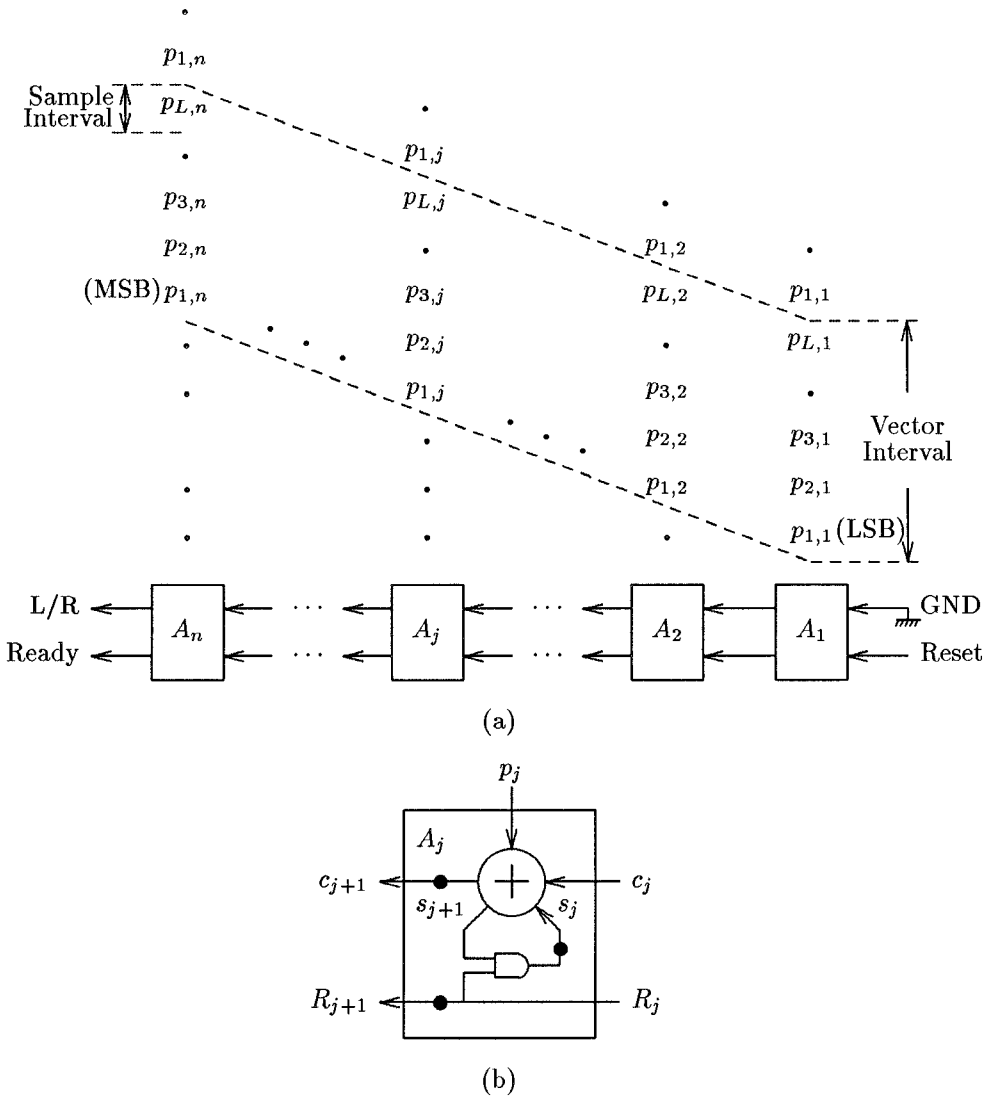


Figure 4: Detailed diagram of the accumulator (a) Linear array of cells. Input data is applied in a skewed fashion and carry is propagated between cells. Reset is applied to the first cell and is propagated down the array. Cells in this array are reset in a staggered fashion. (b) Detail of each cell. Solid circles are unit delay elements.

3 Layout and VLSI Implementation

The detailed block diagram of each processor is shown in Fig. 3. Each processor consists of a pipelined parallel multiplier, a bit-level accumulator, a data vector register, a partial index register, and a local control unit. The multiplier computes $a \times b + c$, and can process a different set of inputs each clock cycle. The products are output in skewed fashion, LSB first, every clock cycle. A bit-level accumulator adds these partial products in bit-serial fashion. The MSB of the accumulated partial products represents the processor's partial index. One of the advantages of this architecture is the absence of any comparator unit. We don't need a comparator because the multiplier can perform addition without any extra hardware. Hence we can directly implement equation (3) in hardware. It is not necessary to add any correction terms to the accumulator's output. The control unit keeps track of each input block of size $k \times k$ pixels and sends a reset signal to the accumulator once every k^2 clock cycles. The reset signal propagates through the accumulator and each of its cells resets in succeeding clock cycles. This scheme allows for the next block of skewed partial products to be accumulated immediately after the last block is applied to the accumulator. Input block sizes of 4×4 or 8×8 pixels can be quantized by this processor. An external control signal is used to select between these two modes.

A separate datapath is used to propagate the partial index through the pipeline. Each block of input vectors has a partial index tag associated with it. This partial index moves along with the input synchronously. An address for the off-chip RAM is generated from this partial index and the output of the on-chip counter. There are 8 pins in the index data path. This allows for trees of depth up to 8 to be easily constructed using these processors. These processors can also be used, together with some external logic, to build trees of depth larger than 8.

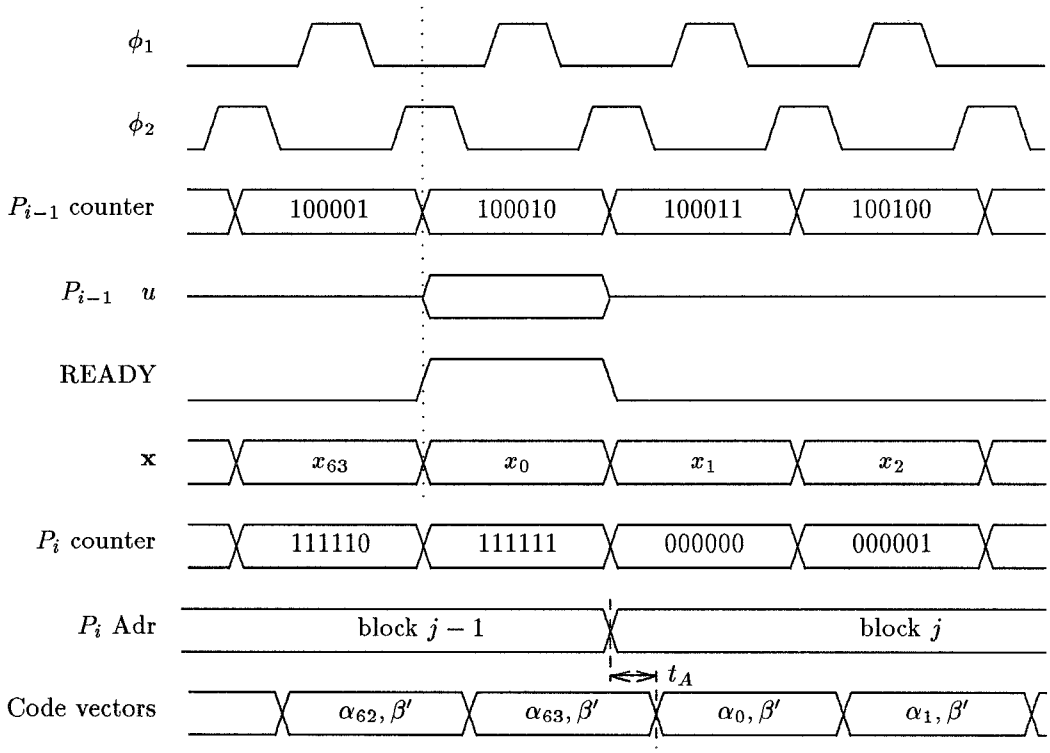


Figure 5: Timing diagram of signal flow between processors for input block size of 8×8. Dotted line shows the boundary between adjacent vectors. Coefficient memory chips must have an access time smaller that t_A.

3.1 Timing

This TSVQ implementation consists of one processor for each level of the tree. Interconnection and data flow between processors is simple and requires no global control signals. Fig. 5 illustrates the timing of all local signals between processors for the case when the block size is 8×8. The system requires a two phase non-overlapping clock. Two phase clocks avoid race conditions and permit simple logic level design. The latency time of each processor is 100 clock cycles. This includes the 64 cycles needed to read each block. If the block size is 4×4, the latency per processor is 52. Each

processor generates a READY signal when its computation is completed. This READY signal also indicates the start of the delayed input vector and its partial index. This signal is used by the neighboring processor to reset its control unit. The partial index is also used as an address for the coefficient memory.

3.2 Fabrication and Testing

The TSVQ processors have been fabricated using $2\mu m$ N-well fabrication process. MOSIS' $7.9mm \times 9.2mm$ standard frame was used as the die size. The chip is packaged in a 84-pin PGA package. A single processor fits in this standard frame, although it does not require the entire area. Fig. 6 shows a plot of the fabricated chip.

This chip was tested using a IMS HS 1000 tester using 500 randomly generated test vectors. It was found to be fully functional at a frequency of 20 MHz.

4 Conclusions

An architecture and VLSI implementation of a TSVQ has been presented. This TSVQ architecture uses identical processors at each level of the binary tree. The architecture is fully pipelined, and latency is 100 clock cycles per processor when the block size is 8×8 pixels. These processors have been fabricated using $2\mu m$ N-well process through MOSIS. The die size is $7.9mm \times 9.2mm$. The processor chips have been thoroughly tested and found to be fully functional at a frequency of 20 MHz.

Using these TSVQ processors, the VQ-DCT-SQ system can process 1 pixel/clock or 160 Mbits/sec. Using two such systems in parallel, we can achieve a data rate of 320 Mbits/sec.

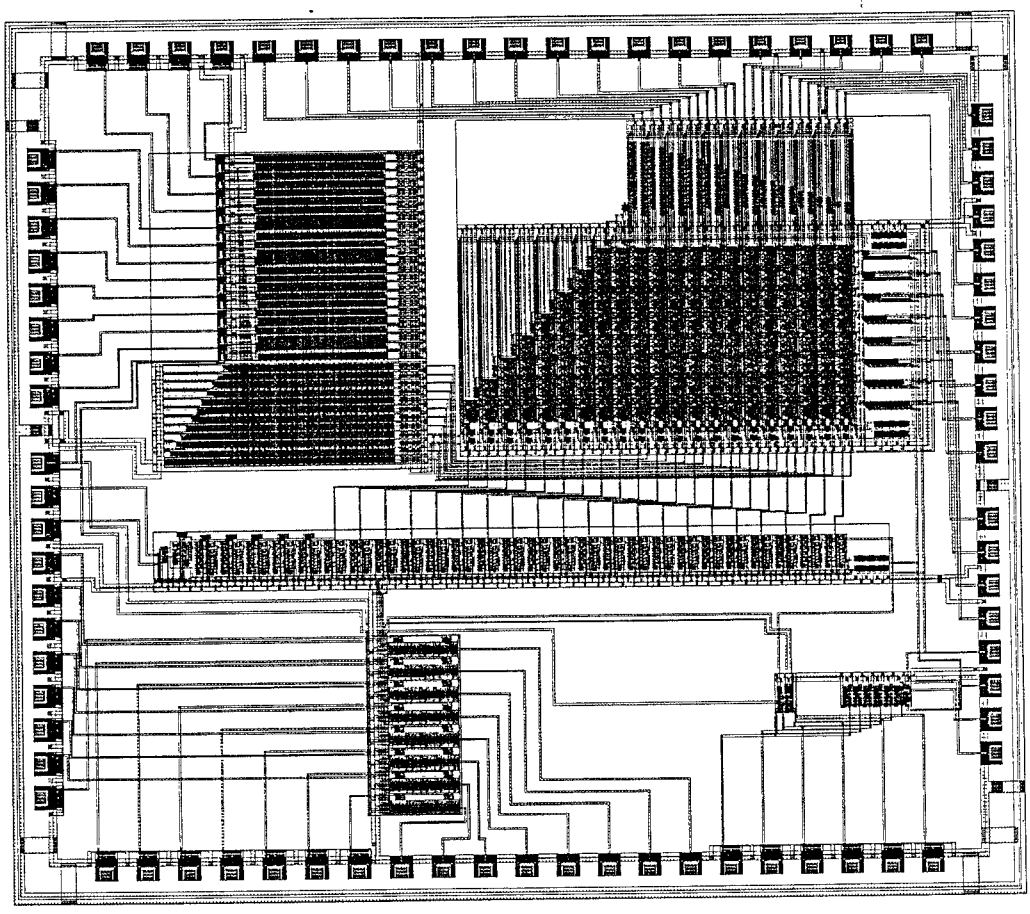


Figure 6: Plot of the TSVQ processor chip. Die size is $7.9mm \times 9.2mm$.

References

[1] N. Nasrabadi and R. King, "Image coding using Vector Quantization: A review," *IEEE Trans. Commun.*, vol. COM-36, pp. 957–971, Aug. 1988.

[2] X. Ran and N. Farvardin, "Combined VQ-DCT coding of images using interblock noiseless coding," in *Proc. IEEE Int'l. Conf. on Acoustics, Speech and Signal Processing*, pp. 2281–2284, 1990.

[3] G. Davidson, P. Cappello, and A. Gersho, "Systolic architectures for vector quantization," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-36, pp. 1651–1664, Oct. 1988.

[4] M. Yan and J. McCanny, "A bit-level systolic architecture for implementing a VQ tree search," *Journal of VLSI Signal Processing*, vol. 2, pp. 149–158, Nov. 1990.

[5] D. Y. Cheng and A. Gersho, "A fast codebook search algorithm for nearest-neighbor pattern matching," in *Proc. IEEE Int'l. Conf. on Acoustics, Speech and Signal Processing*, pp. 265–268, 1986.

[6] T. Lookabaugh, "Architectures for tree structured vector quantization." Unpublished work, May 1987.

[7] W. C. Fang, C. Y. Chang, and B. J. Sheu, "Systolic tree-structured vector quantizer for real-time image compression." Private communication, Oct. 1990.

[8] J. V. McCanny and J. G. McWhirter, "Completely iterative, pipelined multiplier array suitable for VLSI," *IEE Proc.*, vol. 129, pp. 40–46, Apr. 1982.