# Computational RAM Implementation of Vector Quantization for Image Compression

**Thinh M. Le, Sethuraman Panchanathan, and Martin Snelgrove***
**Department of Electrical Engineering, University of Ottawa.**
***Department of Electronics, Carleton University.**

***Abstract*:- Vector Quantization (VQ) is a powerful technique for low-bit rate image and video compression. However, VQ is a computational intensive technique. In this paper, a Computational RAM (C*RAM) implementation of VQ for real-time image compression is presented. The C*RAM architecture enables parallel search in the direction of codewords. The C*RAM implementation uses a simple multiplication-free distortion measure, and also has a possibility of early exit condition resulting in significant reductions in complexity. In addition, the proposed achitecture can be externally programmed in contrast to other dedicated hardware implementations. High-level functional simulations show that the C*RAM architecture can implement VQ in real-time for image compression.**

## 1.0 Introduction

Vector Quantization (VQ) [1] is a promising technique for low-bit rate image and video compression. Nasrabadi and King [2] have presented a review of vector quantization for image coding. In VQ, the input image is first decomposed into a set of vectors. It is assumed that a fixed universal codebook is available at both the transmitter and the receiver. The two basic steps in VQ are quantization and decoding. In the quantization process, for each input vector, the codebook is searched to obtain the closest codeword. Compression is achieved by transmitting the index of the codeword. Reconstruction of images can be implemented by simple table lookup techniques where the label is used as an address to a table containing the codewords.

Since VQ involves a search operation on the codebook to obtain the closest codeword for each input vector, the basic computation in a VQ encoder is the distance calculation which computes the distortion between an input vector and a codeword. Traditionally, the search mechanism is implemented sequentially: each vector is compared with the codewords one at a time. The search complexity of VQ for K input vectors of dimension L and a codebook of size N is O(KLN); this is computation intensive, and is therefore difficult to implement in real-time. Recently, architectures that implement VQ for speech and image coding applications have been reported in the literature. One class of architectures is based on efficient execution of the distortion computation by using fast processors [3] or a pipeline of fast processors [4]; in the latter case, each processor in the pipeline executes a portion of the distortion computation. While the speedup achieved is sufficient for real time speech coding, the search complexity is still O(KLN). A second class of architectures is based on parallel and pipeline execution of the VQ algorithm. For example, Sun and Hsu [5] have reported on an architecture paralleled in the direction of L and pipelined in the direction of N. Abut *et al.* [6] have proposed an architecture based on fuzzy associative memory (FAM) chips where multiple input vectors are processed in parallel, thus partially exploiting parallelism in the direction K. Dezhgosha *et al*. [7] have implemented an architecture paralleled in the direction of N, and have included on-chip memory for fast processing. A third class of architectures implements a modified form of VQ. For example, if a tree-searched VQ [6], or multistage VQ [8] is used, a smaller number of codewords can be searched, with resulting reductions in computational complexity in the direction of N. In theory, these implementations can provide real-time image compression.

VQ can be viewed as a pattern-matching process where each input pattern (vector) is compared with a finite set of templates (codeword) [9]. A promising architecture with a high degree of parallelism for pattern-matching is a Content-Addressable Memory (CAM) [10]. CAM consists of a collection of cells that can be searched simultaneously and in parallel based on their contents. The use of a CAM for both exact and inexact matching of patterns has been reported in the literature [11]. In VQ, the closest codeword for a given input vector is usually determined by employing the mean-square-error (MSE) distortion measure. However, this measure cannot be readily implemented in a CAM-based architecture and is hence replaced by the absolute difference measure. It has been shown [12] that the absolute difference measure results in little degradation in performance compared to the conventional MSE measure.

Recently, Computational RAM's (C*RAM) have been proposed as an attractive alternative for implementing CAM-based algorithms. C*RAM [13-14], is a conventional RAM with SIMD (Single Instruction stream, Multiple Data stream) processing elements added to achieve fast on-chip computation. In this configuration, a processing element (PE) is attached to each memory column which is capable of executing operations in parallel. All PE operations are sequenced by a controller which can be programmed to execute a variety of algorithms. Due to the parallel structure and programmable nature of C*RAM, this architecture is suitable for image processing operations.

In this paper, we propose to implement VQ for image compression using C*RAM. The proposed architecture has the following advantages: parallel search in the direction of N, complexity reduction, and fast execution of VQ with minimal degradation in performance. This paper is organized as follows: The C*RAM architecture is reviewed in section 2. Section 3 presents the VQ implementation using C*RAM. Section 4 provides simulation results and performance analysis. Finally, the conclusions are presented in section 5, followed by the references.

## 2.0 Review of C*RAM

Computational RAM (C*RAM) [13] is a memory-SIMD hybrid architecture where each column of memory has an associated processing element (PE). C*RAM was primarily designed with the goal of augmenting conventional RAM's (used in computer main memory and video buffers) with computation capability. The C*RAM concept makes use of the large on-chip bandwidth to perform massively parallel bit-serial computations. There are two major functions in a C*RAM: memory and computation. When functioning as memory, C*RAM is read or written as part of the host processor address space. When functioning as a computing engine, all PE's execute operations (on its own local memory) in parallel, sequenced by a controller.

### 2.1 C*RAM Architecture

Recently, C*RAM has been fabricated in a 0.8μm BiCMOS chip having an SRAM core[16]. There are 64 processing elements (PE) attached to the 64Kbit memory column which may effectively be considered as 64 independent computing units having its own PE and 1Kbit of local memory (figure 1).

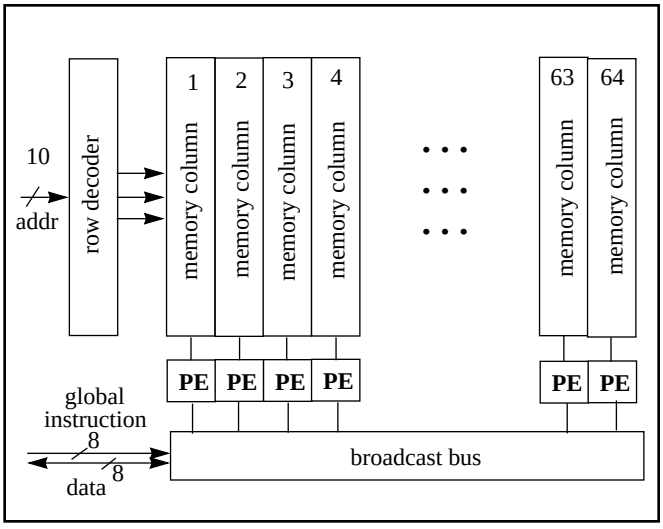


**FIGURE 1. C*RAM architecture**

In the conventional memory, each memory cell is addressed by the row and column decoders. However, in this architecture, the entire row is addressed at the same time. Each PE is addressed by the address stored in its local memory. The read-execute-write cycle time is 20ns. Communication in a C*RAM is done via the PE of each computing unit. Currently, only left-right communication is possible. We now describe the details of each PE.

### 2.2 Processing Element Model

A PE (figure 2) has two 1-bit registers, X and Y, and a 1-bit ALU which functions as a multiplexer. Inputs to the bit-arithmetic ALU can be contents of registers X, Y and a memory bit M. An 8-bit global instruction from the C*RAM controller chooses between the different functions of the ALU. After each computation, the results are written back into the memory or the registers. Each X, Y register has a left or right connection enabling data shifting and neighborhood communication between PE's.

In addition to the X,Y registers and ALU, the Write Enable (WE) register permits conditional operations. WE register functions as a mask, blocking undesired memory columns in various operations. In other words,

the WE register is set only when there is a Write to the corresponding memory column, or reset otherwise. We note that different masks may be stored in the scratch memory for different PE operations. This enables all the PE's in the C*RAM to participate in the global operations without modifying the masked memory contents.

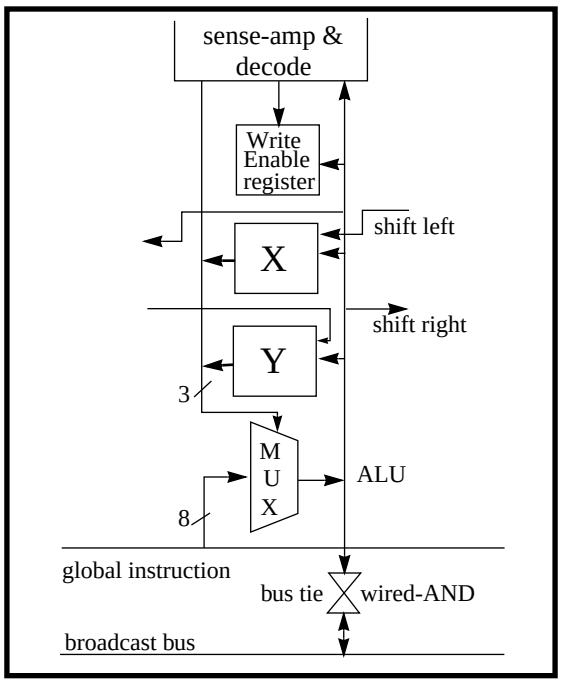


**FIGURE 2. PE model**

A bidirectional bus-tie circuit performs the wired-AND logic of all PE's local results. This global wired-AND is fed back to all the PE's. The same line may also be used by the C*RAM controller to broadcast a 1-bit constant value to all the PE's.

## 3.0 C*RAM Implementation of VQ

C*RAM implementation of VQ has been performed at the functional level. Simulations have been carried out using 6 standard images each of size 512x512 pixels with a universal codebook of size 256 codewords. The codebook was generated from a set of 10 training images using the LGB algorithm[15]. Images are sub-divided into blocks of 4x4 pixles to form 16-D vectors.

The codewords are pre-loaded into the C*RAM where all 16 pixels of each codeword are arranged in one unit (a PE and its local memory). A codebook of 256 code-words will therefore occupy 256 units which correspond to a module of 4 C*RAM chips. The encoding process is thus executed in parallel over the entire codebook. The input vectors are broadcast to all units of the C*RAM module one at a time (figure 3). A scheduler is designed to take care of the input vector sequencing. We note that for real-time implementation, additional modules may be required to increase the level of parallelism. When the scheduler finds a module in a busy state, it will redirect the input vector to the next module for compression.

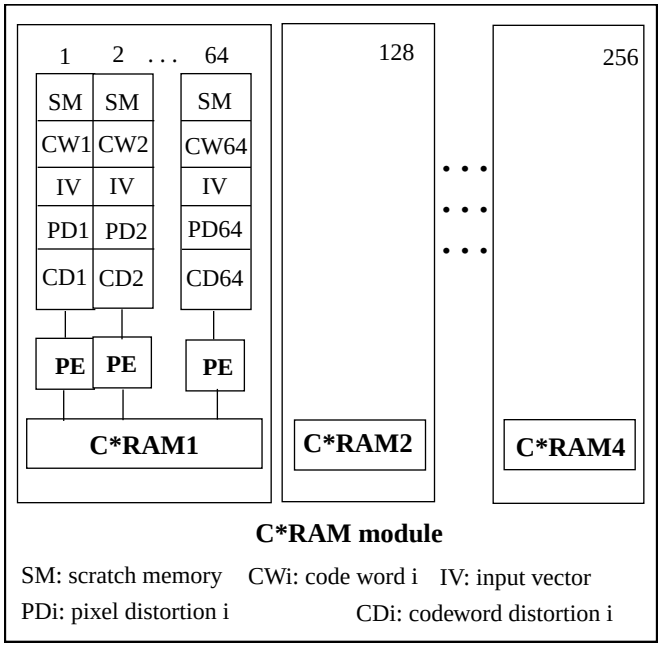


**FIGURE 3. Codeword arrangement in C*RAM**

In each memory column of a unit, the space is occupied by the following:

1. Scratch memory (SM): for PE address, temporary results, and masks for different PE operations;

2. Codeword i(CWi): comprising of 16, 8-bit pixels;

3. Input vector (IV): also comprising of 16, 8-bit pixels. (Note that, at a particular time, all memory columns process the same input vector);

4. Pixel distortion i (PDi): comprising of 16, 8-bit distortions betweeen the pixels of CWi and the corresponding pixels of IV; and

5. Codeword distortion i (CDi): which is the sum of all 16 pixel distortions of a particular codeword i.

The encoding process can be executed in at most three stages:

i. Pixel distortion calculation,

ii. Pattern-matching search, and

iii. Absolute-difference search.

1.Pixel distortion calculation: In this stage, the pixel distortion of each pixel of every codeword and the corresponding pixels of the input vector are calculated. With a codebook of N codewords each of size L, the pixel distortion $\vec{R} = R_{ij}$, for a given input vector $\vec{X} = [X_j]$, and a codeword $\vec{C} = [C_{ij}]$ is given by:

$$R_{ij} = |X_j - C_{ij}| \quad ,i = 0 to N-1 \quad j = 0 to L-1 \qquad (1)$$

As shown in equation (1), the absolute value of the pixel distortion is normally required to obtain the closest codeword for each input vector. However, the C*RAM architecture is defined for fast implementation of logic rather than arthmetic operations.

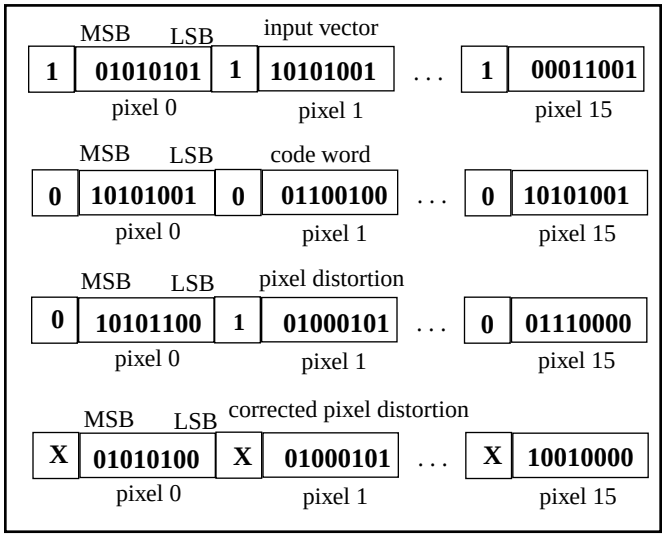


**FIGURE 4. Pixel distortion calculation in a unit**

In order to facilitate the computation process, 1's are appended to the $2^N$ position of each input vector pixel, and 0's are appended to the $2^N$ position of each codeword pixel, where N is the number of bits representing the gray levels (figure 4). The pixels of the codewords are then subtracted from those of the modified input vector. If the Nth bit of the result $\vec{R}$ is 0, then result $\vec{R}$ is negative, is therefore 2's complemented.

b.Pattern-matching search: In this stage, a minimum search is performed throughout the bit planes of the pixel distortions to find the closest match codewords for early detection. This stage takes advantage of the fact that a significant number of input vectors which do not greatly deviate from the codewords can be detected at this stage. This can be done by conducting a minima search on all L pixel distortions of the result $\vec{R}$ with a search threshold being set from the bit planes $B_{N-1}$ to $B_i$ (figure 5). This search threshold can be adjusted depending on the nature of the image. For low-detailed images, i may be set to 1, and for other images i can be set to 2.

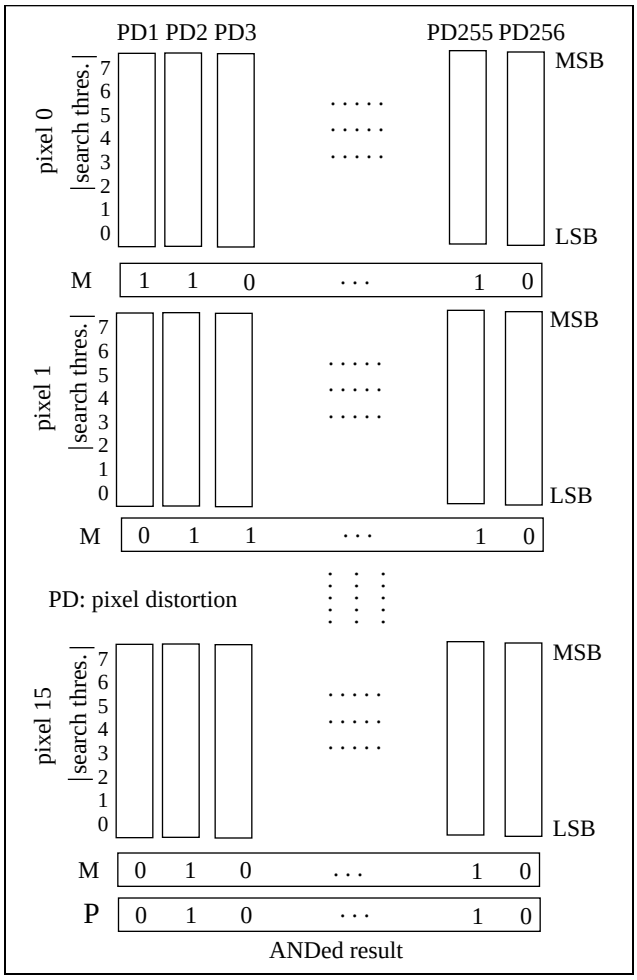


**FIGURE 5. Pattern-matching search**

The minimum search is carried out as follows: starting from the most significant bit plane $B_{N-1}$ of each pixel distortion, if the bit of a unit is 0, the corresponding unit is considered as the smaller one; if the bit is 1, the

corresponding unit is eliminated. After each comparison, the remaining units become the candidates for the search at the next bit plane. The search at each pixel distortion is continued to bit plane $B_i$. The result of each minimum search of each pixel distortion are stored in scratch memory of the corresponding unit, where 1 indicates the minimum value, and 0 indicates that the vector is not qualify (please see vector M's in figure 5). This process is repeated L times over the L pixel distortions. The ANDed result of the corresponding units of L minima searches will determine whether there is a match. In other words, if all L pixel distortions are minimum, the corresponding unit is the closest match. In the case of multiple matches, any one of the closest codewords can be designated as the best match.

c.Absolute-difference search: If no match has been found in the pattern-matching search, (that is there are no 1's in vector P in figure 5) the closest match of the input vector will finally be searched using the absolute-difference search. In this stage, all L pixel distortions $\hat{R}$ are added up to form the codeword dirtortion $CD_i$, and the index of the codeword which has smallest codeword distortion will be chosen as the best match. The distortion of each codeword is given by:

$$CD_i = \sum_{j=0}^{L-1} R_{ij} \qquad i = 0 to N-1 \tag{2}$$

# 4.0 Performance

## 4.1 Computational complexity

Let p be the probability of encoding input vectors using the pattern-matching search, M1 be the number of memory cycles required to detect the closest match using the pattern-matching search, M2 be the number of memory cycles required to find the best match using the absolute-difference search. M is the weighted sum of M1 and M2 (as shown in equation 3), and the total computational complexity using the C*RAM implementation for an image of K input vectors with dimension L and a codebook of size N is reduced to order O(KL). M is proportional to L and can be expressed as follows:

$$M = p \cdot M_1 + (1-p) \cdot M_2 \qquad M_1 < M_2 \tag{3}$$

where p is image dependent. When p=0, $M=M_2$, the image is coded in the mean-absolute-difference sense. When p=1, $M=M_1$ the image is coded using the pattern-matching technique. For low-detailed images, p is large, therefore, M decreases. Thus the computational complexity is further reduced.

## 4.2 Simulation results

The coding performance of VQ is evaluated using the peak signal-to-noise ratio (PSNR), which is defined as:

$$PSNR = 10 \cdot \log \frac{255^2}{MSE} \tag{4}$$

Note that, for an MxM image, the mean-square-error (MSE) is defined as:

$$MSE = \left(\frac{1}{M}\right)^2 \sum_{k=1}^{M} \sum_{l=1}^{M} (x_{kl} - \hat{x}_{kl})^2 \tag{5}$$

where $x_{kl}$ and $\hat{x}_{kl}$ denote the original image pixel and encoded image pixel, respectively.

The simulation results of 6 images are tabulated in Table 1. In the first 6 cases, the search threshold was set from bit planes 7 to bit planes 2 (figure 5). In the last 2 cases, the search threshold was set from bit planes 7 to bit planes 1.

**TABLE 1. VQ using universal codebook of size 256**

| Image | PSNR C*RAM (dB) | PSNR full search (dB) | no. of mod. used | % pattern matching |
|---|---|---|---|---|
| Sailboat | 27.18 | 27.42 | 17 | 21.0 |
| Lena | 28.82 | 29.05 | 17 | 22.5 |
| Moon | 32.25 | 32.64 | 15 | 50.2 |
| Airplane | 29.04 | 29.53 | 14 | 54.2 |
| Chest | 35.12 | 37.96 | 12 | 82.6 |
| Ball | 34.09 | 38.88 | 10 | 97.5 |
| Chest* | 37.72 | 37.96 | 15 | 42.6 |
| Ball* | 37.74 | 38.88 | 12 | 69.7 |

*These simulations were carried out with the search threshold being set from bit plane 7 to bit plane 1 of the pixel distortions. We note that the time to process a frame is 33ms (real-time).

From Table 1, the following observations can be made:

1. For medium- to high-detailed images as in the case of Sailboat and Lena, the number of C*RAM modules used is large because of the low percentage of pattern-matching.

2. For low-detailed images as in the case of Chest and Ball, the number of C*RAM modules used is small because of the high percentage of pattern-matching. However, this results in a lower objective quality compared to full search VQ. A reduced search threshold (to bit plane 1 from bit plane 2) results in better quality images.

## 5.0 Conclusions

We have presented a C*RAM implementation of VQ for image compression. VQ involves a search operation on the codebook to obtain the best match. The search mechanism when implemented sequentially results in a complexity of O(KLN). The proposed C*RAM-based implementation of VQ results in a complexity of order O(KL). In addition, further reduction in complexity is made possible by the use of early exit condition. Simulations demonstrate that real-time VQ encoding is possible by the proposed 64K C*RAM implementation. A 256K C*RAM is presently under fabrication. This would reduce the number of C*RAM chips used by a factor of 4.

**Acknowledgement**

This work was supported by the National Sciences and Engineering Research Council of Canada (NSERC) and Microelectronics Network (MicroNet) under the Network Centers of Excellence (NCE) program of the Government of Canada. The authors wish to thank Christian Cojocaru and Duncan G. Elliott for their help.

**References**

[1] R. M. Gray, "Vector Quantization" *IEEE ASSP Mag*., pp. 4-29, Apr. 1984.

[2] N.M. Nasrabadi and R. A. King, "Image coding using vector quantization: A review," IEEE Trans. Commun., vol. COM-36, no. 8, pp. 957-971, Aug. 1988.

[3] B. P. M. Tao, H. Abut, and R. M. Gray, "Hardware realization of waveform vector quantization," IEEE J. Select. Areas Commun., vol. SAC-2, pp. 343-352, Mar. 1985.

[4] G. Davidson and A. Gersho, "Application of a VLSI vector quantization processor to real-time speech coding," IEEE Select. Areas Commun., vol. SAC-4, no. 1. pp. 112-124, Jan. 1986.

[5] H. Sun and H. Hsu, "A VLSI wavefront array for image vector quantization," in Proc. 30th Midwest Symp. Circuits Syst., pp. 1230-1233, Aug. 1987.

[6] H. Abut, B. P. M. Tao, and J. L. Smith, "Vector quantizer architectures for speech and image coding," in Proc. IEEE ICASSP'87, vol. 2 (Dallas, TX) pp. 756-759, Apr. 1987.

[7] K. Dezhgosha, M. M. Jamali, S. C. Kwatra, "A VLSI Architecture for Real-Time Image Coding Using a Vector Quantization Based Algorithm", IEEE Transactions on Signal Processing, Vol. 40, No. 1, pp. 181-189, Jan. 1992.

[8] P. A. Ramamoorthy and B. Potu, "Bit-serial systolic chip set for real-time image coding", in Proc. IEEE ICASSP'87, vol. 2, pp.789-792, Apr. 1987.

[9] A. Gersho and V. Cuperman, "Vector quantization: A pattern matching technique for speech coding", IEEE Commun. Mag., pp 15-21, Dec. 1983.

[10] C. C. Foster, *Content Addressable Parallel Processors* (Computer Science Series) New York: Van Nostrand Reinhold, 1976.

[11] S. S. Yau and C. C. Yang, "Pattern recognition by using an associative memory", IEEE Trans. Electron. Comput., vol. 15, pp. 944-947, Dec. 1966.

[12] S. Panchanathan, M. Goldberg, "A Content-Addressable Memory Architecture for Image Coding Using Vector Quantization", *IEEE Trans. on Signal Processing*, Vol. 39, No. 9, Sept. 1991.

[13] D. G. Elliott, M. Snelgrove, and M. Stumm, "Computational-RAM: A Memory-SIMD Hybrid and Its Application to DSP", CICC May 1992.

[14] C. Cojocaru, M. Snelgrove, and D. G. Elliott, "A BATMOS Processing Element for 256Kb Computational RAM", int. rep., DOE Carleton University.

[15] Y. Linde, A. Buzo, and R. M. Gray, "An Algorithm for Vector Quantizer Design", *IEEE Trans. Commun*, Vol. COM-28, pp. 84-95, Jan. 1980.

[16] Allan L. Silburt et al, "A 200MHz 0.8um BiCMOS modular memory family of DRAM and multi-port SRAM's", proceedings of the IEEE 1992, CICC May 1992.