Efficient K-Means VLSI Architecture for Vector Quantization

Hui-Ya Li, Wen-Jyi Hwang*, Chih-Chieh Hsu, and Chia-Lung Hung

Department of Computer Science and Information Engineering,
National Taiwan Normal University, Taipei, 117, Taiwan
royalfay@gmail.com, whwang@ntnu.edu.tw,
andy730215@msn.com, nicky730216@gmail.com

Abstract. A novel hardware architecture for k-means clustering is presented in this paper. Our architecture is fully pipelined for both the partitioning and centroid computation operations so that multiple training vectors can be concurrently processed. The proposed architecture is used as a hardware accelerator for a softcore NIOS CPU implemented on a FPGA device for physical performance measurement. Numerical results reveal that our design is an effective solution with low area cost and high computation performance for k-means design.

1 Introduction

Cluster analysis is a method for partitioning a data set into classes of similar individuals. The clustering applications in various areas such as signal compression, data mining and pattern recognition, etc., are well documented. In these clustering methods the k-means [9] algorithm is the most well-known clustering approach which restricts each point of the data set to exactly one cluster.

One drawback of the k-means algorithm is the high computational complexity for large data set and/or large number of clusters. A number of fast algorithms [2,6] has been proposed for reducing the computational time of the k-means algorithm. Nevertheless, only moderate acceleration can be achieved in these software approaches.

Other alternatives for expediting the k-means algorithm are based on hardware. As compared with the software counterparts, the hardware implementations may provide higher throughput for distance computation. Efficient architectures for distance calculation and data set partitioning process have been proposed in [3,5,10]. Nevertheless, the centroid computation is still conducted by software in some architectures. This may limit the speed of the systems. Although hardware dividers can be employed for centroid computation, the hardware cost of the circuit may be high because of the high hardware complexity for the divider design. In addition, when the usual multi-cycle sequential divider architecture is employed, the implementation of pipeline architecture for both clustering and partitioning process may be difficult.

* To whom all correspondence should be sent.

A.-B. Salberg, J.Y. Hardeberg, and R. Jenssen (Eds.): SCIA 2009, LNCS 5575, pp. 440–449, 2009.

The goal of this paper is to present a novel pipeline architecture for the k-means algorithm. The architecture adopts a low-cost and fast hardware divider for centroid computation. The divider is based on simple table lookup, multiplication and shift operations so that the division can be completed in one clock cycle. The centroid computation therefore can be implemented as a pipeline. In our design, the data partitioning process can also be implemented as a c-stages pipeline for clustering a data set into c clusters. Therefore, our complete k-means architecture contains $c + 2$ pipeline stages, where the first c stages are used for the data set partitioning, and the final two stages are adopted for the centroid computation.

The proposed architecture has been implemented on field programmable gate array (FPGA) devices [8] so that it can operate in conjunction with a softcore CPU [12]. Using the reconfigurable hardware, we are then able to construct a system on programmable chip (SOPC) system for the k-means clustering. The applications considered in our experiments are the vector quantization (VQ) for signal compression [4]. Although some VLSI architectures [1,7,11] have been proposed for VQ applications, these architectures are used only for VQ encoding. The proposed architecture is used for the training of VQ codewords. As compared with its software counterpart running on Pentium IV CPU, our system has significantly lower computational time for large training set. All these facts demonstrate the effectiveness of the proposed architecture.

2 Preliminaries

We first give a brief review of the k-means algorithm for the VQ design. Consider a full-search VQ with c codewords $\{y_1, ..., y_c\}$. Given a set of training vectors $T = \{x_1, ..., x_t\}$, the average distortion of the VQ is given by

$$D = \frac{1}{wt} \sum_{j=1}^{t} d(x_j, y_{\alpha(x_j)}), \tag{1}$$

where w is the vector dimension, t is the number of training vectors, $\alpha()$ is the source encoder, and $d(u, v)$ is the squared distance between vectors u and v. The k-means algorithm is an iterative approach finding the solution of $\{y_1, ..., y_c\}$ locally minimizing the average distortion D given in eq.(1). It starts with a set of initial codewords. Given the set of codewords, an optimal partition $T_1, T_2, ..., T_c$ is obtained by

$$T_i = \{x : x \in T, \alpha(x) = i\}, \tag{2}$$

where

$$\alpha(x) = \arg \min_{1 \le j \le c} d(x, y_j). \tag{3}$$

After that, given the optimal partition obtained from the previous step, a set of optimal codewords is computed by

$$y_i = \frac{1}{Card(T_i)} \sum_{x \in T_i} x. \tag{4}$$

The same process will be repeated until convergence of the average distortion D of the VQ is observed.

3 The Proposed Architecture

As shown in Fig. 1, the proposed k-means architecture can be decomposed into two units: the partitioning unit and the centroid computation unit. These two units will operate concurrently for the clustering process. The partitioning unit uses the codewords stored in the register to partition the training vectors into c clusters. The centroid computation unit concurrently updates the centroid of clusters. Note that, both the partitioning process and centroid computation process should operate iteratively in software. However, by adopting a novel pipeline architecture, our hardware design allows these two processes to operate in parallel for reducing the computational time. In fact, our design allows the concurrent computation of c+2 training vectors for the clustering operations.

Fig. 2 shows the architecture of the partitioning unit, which is a c-stage pipeline, where c is the number of codewords (i.e., clusters). The pipeline fetch one training vector per clock from the input port. The i-th stage of the pipeline compute the squared distance between the training vector at that stage and the i-th codeword of the codebook. The squared distance is then compared with the current minimum distance up to the i-th stage. If distance is smaller than the current minimum, then the i-th codeword becomes the new current optimal codeword, and the corresponding distance becomes the new current minimum distance. After the computation at the c-th stage is completed, the current optimal codeword and current minimum distance are the actual optimal codeword and the actual minimum distance, respectively. The index of the actual optimal codeword and its distance will be delivered to the centroid computation unit for computing the centroid and overall distortion.

As shown in Fig. 2, each pipeline stage i has input ports *training_vector_in, codeword_in, D_in, index_in*, and output ports *training_vector_out, D_out, index_out*. The *training_vector_in* is the input training vector. The *codeword_in* is the i-th codeword. The *index_in* contains index of the current optimal codeword up to the stage i. The *D_in* is the current minimum distance. Each stage i first computes the squared distance between the input training vector and the i-th codeword (denoted by D_i), and then compared it with the *D_in*. When

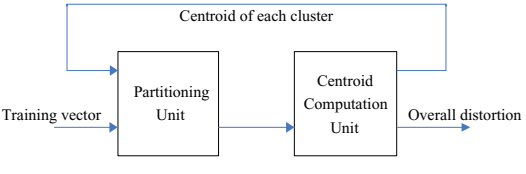

Fig. 1. The proposed k-means architecture

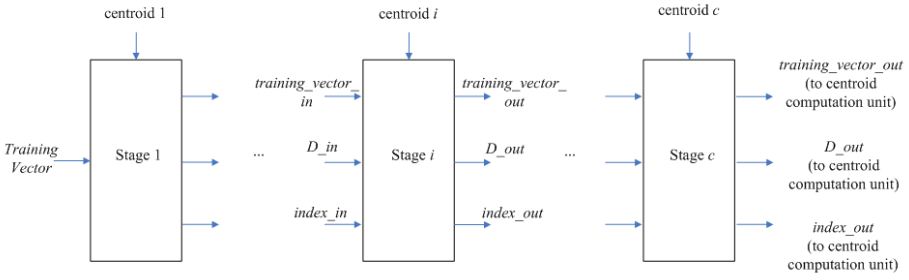

Fig. 2. The architecture of the partitioning unit

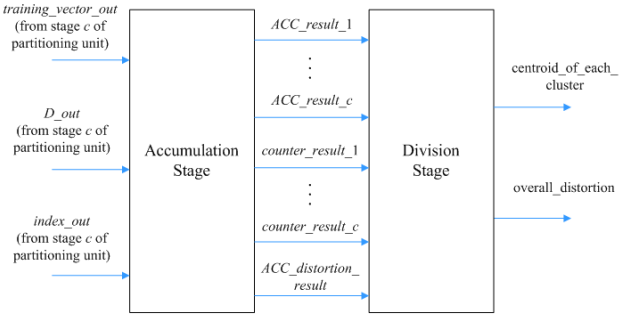

Fig. 3. The architecture of the centroid computation unit

the squared distance is greater than *D_in*, we have *index_out* $\leftarrow$ *index_in* and *D_out* $\leftarrow$ *D_in*. Otherwise, *index_out* $\leftarrow i$, and the *D_out* $\leftarrow D_i$. Note that the output ports *training_vector_out, D_out* and *index_out* at stage i are connected to the input ports *training_vector_in, D_in*, and *index_in* at the stage i+1, respectively. Consequently, the computational results at stage i at the current clock cycle will propagate to stage i+1 at the next clock cycle. When the training vector reaches the c-th stage, the final *index_out* indicates the index of the actual optimal codeword, and the *D_out* contains the corresponding distance.

Fig. 3 depicts the architecture of the centroid computation unit, which can be viewed as a two-stage pipeline. In this paper, we call these two stages, the accumulation stage and division stage, respectively. Therefore, there are $c + 2$ pipeline stages in the k-means unit. The concurrent computation of c+2 training vectors therefore is allowed for the clustering operations.

As shown in Fig. 4, there are c accumulators (denoted by $ACCi, i = 1, .., c$) and c counters for the centroid computation in the accumulation stage. The i-th accumulator records the *current* sum of the training vectors assigned to cluster i. The i-th counter contains the current number of training vectors mapped to cluster i. The *training_vector_out, D_out* and *index_out* in Fig. 4 are actually the outputs of the c-th pipeline stage of the partitioning unit. The *index_out* is used

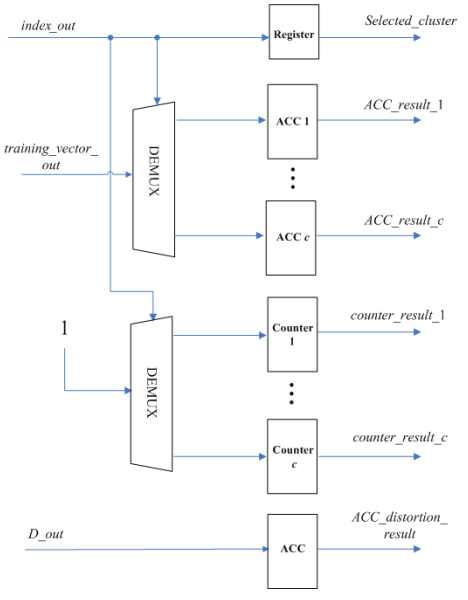

Fig. 4. The architecture of accumulation stage of the centroid computation unit

as control line for assigning the training vector (i.e. *training_vector_out*) to the optimal cluster found by the partitioning unit.

The circuit of division stage is shown in Fig. 5. There is only one divider in the unit because only one centroid computation is necessary at a time. Suppose the final *index_out* is i for the j-th vector in the training set. The centroid of the i-th cluster then need to be updated. The divider and the i-th accumulator and counter are responsible for the computation of the centroid of the i-th cluster. Upon the completion of the j-th training vector at the centroid computation unit, the i-th counter records the number of training vectors (up to j-th vector in the training set) which are assigned to the i-th cluster. The i-th accumulator contains the sum of these training vectors in the i-th cluster. The output of the divider is then the mean value of the training vectors in the i-th cluster.

The architecture of the divider is shown in Fig. 6, which contains w units (w is the vector dimension). Each unit is a scalar divider consisting of an encoder, a ROM, a multiplier and a shift unit. Recall that the goal of the divider is to find the mean value as shown in eq.(4). Because the vector dimension is w, the sum of vectors $\sum_{x \in T_i} x$ has w elements, which are denoted by $S_1, ..., S_w$ in the Fig. 6.(a). For the sake of simplicity, we let S be an element of $\sum_{x \in T_i} x$, and $Card(T_i) = M$. Note that both S and M are integers. It can then be easily observed that

$$\frac{S}{M} = S \times \frac{2^k}{M} \times 2^{-k}, \tag{5}$$

for any integer $k > 0$. Given a positive integer k, the ROM in Fig. 6.(b) in its simplest form have 2^k entries. The m-th, $m = 1, ..., 2^k$, entry of the ROM

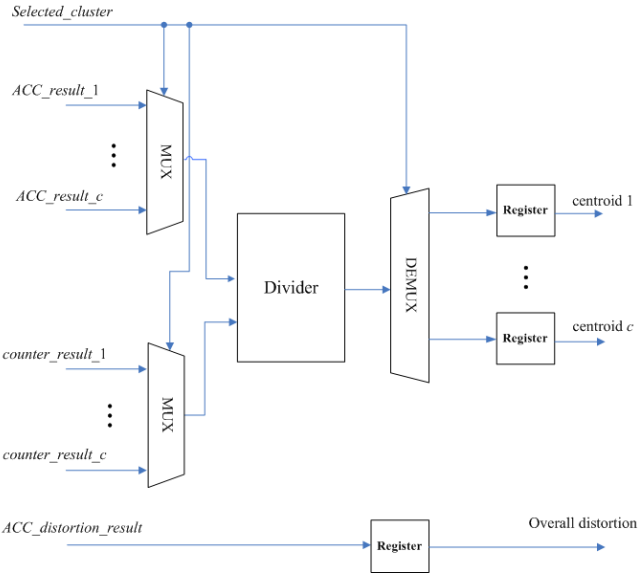

Fig. 5. The architecture of division stage of the centroid computation unit

contains the value $\frac{2^k}{m}$. Consequently, for any positive $M \leq 2^k$, $\frac{2^k}{M}$ can be found by a simple table lookup process from the ROM. The output of the ROM is then multiplied by S, as shown in the Fig. 6.(b). The multiplication result is then shifted right by k bits for the completion of the division operation $\frac{S}{M}$.

In our implementation, each $\frac{2^k}{m}, m = 1, ..., 2^k$, has only finite precision with fixed-point format. Since the maximum value of $\frac{2^k}{m}$ is 2^k, the integer part of $\frac{2^k}{m}$ has k bits. Moreover, the fractional part of $\frac{2^k}{m}$ contains b bits. Each $\frac{2^k}{m}$ therefore is represented by $(k + b)$ bits. There are 2^k entries in the ROM. The ROM size therefore is $(k + b) \times 2^k$ bits.

It can be observed from the Fig. 6 that the division unit also evaluates the overall distortion of the codebook. This can be accomplished by simply accumulating the minimum distortion associated with each training vector after the completion of the partitioning process. The overall distortion is used for both the performance evaluation and the convergence test of the k-means algorithm.

The proposed architecture is used as a custom user logic in a SOPC system consisting of softcore NIOS CPU, DMA controller and SDRAM, as depicted in Fig. 7. The set of training vectors is stored in the SDRAM. The training vectors are then delivered to the proposed circuit one at a time by the DMA controller for k-means clustering. The softcore NIOS CPU only has to activate the DMA controller for the training vector delivery, and then collects the clustering results after the DMA operations are completed. It does not participate in the partitioning and centroid computation processes of the k-means algorithm. The computational time for k-means clustering can then be lowered effectively.

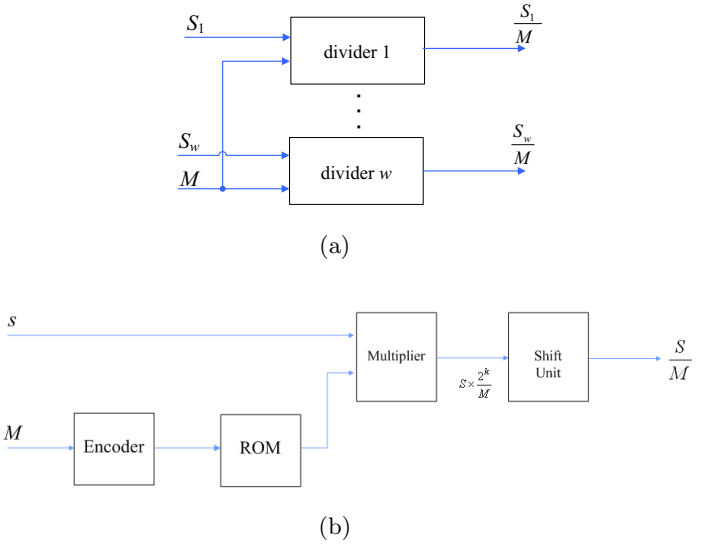

Fig. 6. The architecture of divider: (a) The divider contains w units; (b) Each unit is a scalar divider consisting of an encoder, a ROM, a multiplier, and a shift unit

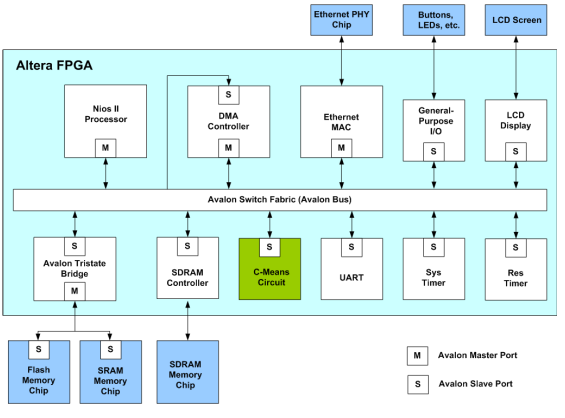

Fig. 7. The architecture of the SOPC using the proposed k-means circuit as custom user logic

4 Experimental Results

This section presents some experimental results of the proposed architecture. The k-means algorithm is used for VQ design for image coding in the experiments. The vector dimension is $w = 2 \times 2$. There are 64 codewords in the VQ. The target FPGA device for the hardware design is Altera Stratix II 2S60.

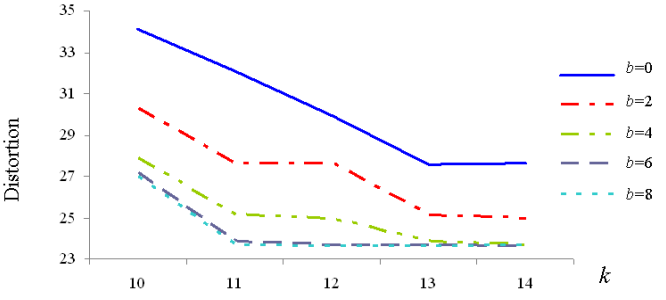

Fig. 8. The performance of the proposed k-means circuit for various sets of parameters k and b

We first consider the performance of the divider for the centroid computation of the k-means algorithm. Recall that our design adopts a novel divider based on table lookup, multiplication and shift operations, as shown in eq.(5). The ROM size of the divider for table lookup is dependent on the parameters k and b. Higher k and b values may improve the k-means performance at the expense of larger ROM size.

Fig. 8 shows the performance of the proposed circuit for various sets of parameters k and b. The training set for VQ design contains 30000 training vectors drawn from the image "Lena" [13]. The performance is defined as the average distortion of the VQ defined in eq.(1). All the VQs in the figure starts with the same set of initial codewords. It can be observed from the figure that the average distortion is effectively lowered as k increases for fixed b. This is because the parameter k set an upper bound on the number of vectors (i.e., M in eq.(5)) in each cluster. In fact, the upper bound of M is 2^k. Higher k values reduce the possibility that actual M is larger than 2^k. This may enhance the accuracy for centroid computation. We can also see from Fig. 8 that larger b can reduce the average distortion as well. Larger b values increase the precision for the representation of $\frac{2^k}{m}$; thereby improve the division accuracy.

The area cost of the proposed k-means circuit for various sets of parameters k and b is depicted in Fig. 9. The area cost is measured by the number of adaptive logic modules (ALMs) consumed by the circuit. It can be observed from the figure that the area cost of our circuit reduces significantly when k and/or b becomes small. However, improper selection of k and b for area cost reduction may increase the average distortion of the VQ. We can see from Fig. 8 that the division circuit with $b = 8$ has performance less susceptible to k. It can be observed from Fig. 8 and 9 that the average distortion of the circuit with $(b = 8, k = 11)$ is almost identical to that of the circuit with $(b = 8, k = 14)$. Moreover, the area cost of the centroid computation unit with $(b = 8, k = 11)$ is significantly lower than that of the circuit with $(b = 8, k = 14)$. Consequently, in our design, we select $b = 8$ and $k = 11$ for the divider design.

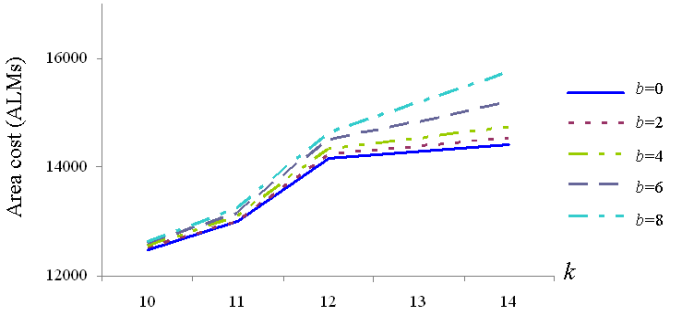

Fig. 9. The area cost of the k-means circuit for various sets of parameters k and b

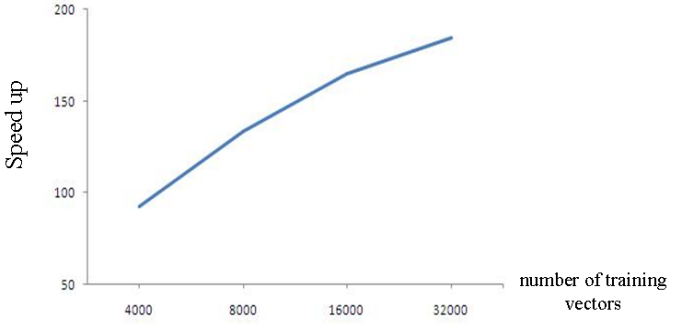

Fig. 10. Speedup of the proposed system over its software counterpart

Our SOPC system consists of softcore NIOS CPU, DMA controller, 10 M bytes SDRAM and the proposed k-means circuit. The k-means circuit consumes 13253 ALMs, 8192 embedded memory bits and 288 DSP elements. The NIOS softcore CPU of our system also consumes hardware resources. The entire SOPC system uses 17427 ALMs and 604928 memory bits.

Fig. 10 compares the CPU time of our system with its software counterpart running on 3 GHz Pentium IV CPU for various sizes of training data set. It can be observed from the figure that the execution time of our system is significantly lower than that of its software counterpart. In addition, gap in CPU time enlarges as the the training set size increases. This is because our system is based on efficient pipelined computation for partitioning and centroid operations. When the training set size is 32000 training vectors, the CPU time of our system is only 3.95 mini seconds, which is only 0.54% of the CPU time of its software counterpart. The speedup of our system over software implementation is 185.18.

5 Concluding Remarks

The proposed architecture has been found to be effective for k-means design. It is fully pipelined with simple divider for centroid computation. It has high

throughput, allowing concurrent partitioning and centroid operations for $c + 2$ training vectors. The architecture can be efficiently used as an hardware accelerator for a general processor. As compared with the software k-means running on Pentium IV, the NIOS-based SOPC system incorporating our architecture has significantly lower execution time. The proposed architecture therefore is beneficial for reducing computational complexity for clustering analysis.

References

1. Bracco, M., Ridella, S., Zunino, R.: Digital implementation of hierarchical vector quantization. IEEE Trans. Neural Networks, 1072–1084 (2003)
2. Elkan, C.: Using the triangle inequality to accelerate K-Means. In: Proc. International Conference on Machine Learning (2003)
3. Estlick, M., Leeser, M., Theiler, J., Szymanski, J.J.: Algorithmic transformations in the implementation of K- means clustering on reconfigurable hardware. In: Proc. of ACM/SIGDA 9th International Symposium on Field Programmable Gate Arrays (2001)
4. Gersho, A., Gray, R.M.: Vector Quantization and Signal Compression. Kluwer, Norwood (1992)
5. Gokhale, M., Frigo, J., Mccabe, K., Theiler, J., Wolinski, C., Lavenier, D.: Experience with a Hybrid Processor: K-Means Clustering. The Journal of Supercomputing, 131–148 (2003)
6. Hwang, W.J., Jeng, S.S., Chen, B.Y.: Fast Codeword Search Algorithm Using Wavelet Transform and Partial Distance Search Techniques. Electronic Letters 33, 365–366 (1997)
7. Hwang, W.J., Wei, W.K., Yeh, Y.J.: FPGA Implementation of Full-Search Vector Quantization Based on Partial Distance Search. Microprocessors and Microsystems, 516–528 (2007)
8. Hauck, S., Dehon, A.: Reconfigurable Computing. Morgan Kaufmann, San Francisco (2008)
9. MacQueen, J.: Some Methods for Classi cation and Analysis of Multivariate Observations. In: Proc. of the 5th Berkeley Symposium on Mathematical Statistics and Probability, pp. 281–297 (1967)
10. Maruyama, T.: Real-time K-Means Clustering for Color Images on Reconfigurable Hardware. In: Proc. 18th International Conference on Pattern Recognition (2006)
11. Wang, C.L., Chen, L.M.: A New VLSI Architecture for Full-Search Vector Quantization. IEEE Trans. Circuits and Sys. for Video Technol., 389–398 (1996)
12. NIOS II Processor Reference Handbook, Altera Corporation (2007), `http://www.altera.com/literature/lit-nio2.jsp`
13. USC-SIPI Lab, `http://sipi.usc.edu/database/misc/4.2.04.tiff`