FPGA Implementation of kNN Classifier Based on Wavelet Transform and Partial Distance Search

Yao-Jung Yeh, Hui-Ya Li, Wen-Jyi Hwang*, and Chiung-Yao Fang

Graduate Institute of Computer Science and Information Engineering,
National Taiwan Normal University, Taipei, 117, Taiwan
spmark@ice.ntnu.edu.tw,f3807391@ms8.url.com.tw,whwang@ntnu.edu.tw,
violet@ice.ntnu.edu.tw

Abstract. A novel algorithm for field programmable gate array (FPGA) realization of kNN classifier is presented in this paper. The algorithm identifies first k closest vectors in the design set of a kNN classifier for each input vector by performing the partial distance search (PDS) in the wavelet domain. It employs subspace search, bitplane reduction and multiple-coefficient accumulation techniques for the effective reduction of the area complexity and computation latency. The proposed implementation has been embedded in a softcore CPU for physical performance measurement. Experimental results show that the implementation provides a cost-effective solution to the FPGA realization of kNN classification systems where both high throughput and low area cost are desired.

Keywords: FPGA Implementation, Partial Distance Search, Image Processing, Pattern Recognition, Nonparametric Classification.

1 Introduction

Nonparametric classification techniques such as kNN [3,5,7] have been shown to be effective for applications of statistical pattern recognition. These techniques can achieve a high classification accuracy for problems of unknown distributions. Besides, when dealing with problems of nonnormal distributions, these techniques enjoy a lower classification error rate than that achieved by the commonly used parametric classification approaches, such as linear classifiers and quadratic classifiers. However, the nonparametric classification has a major drawback that a large amount of design vectors are required in classifiers. Large design sets, which result in high computation and storage complexities for classifiers, are necessary since the nonparametric methods require adequate statistical information. Therefore, it might be difficult to apply nonparametric classification techniques for the pattern recognition applications where real-time processing is desired.

* To whom all correspondence should be sent.

B.K. Ersbøll and K.S. Pedersen (Eds.): SCIA 2007, LNCS 4522, pp. 512–521, 2007.

One way to reduce the computational complexity is based on partial distance search (PDS) [2,3,4] in the original or transform domains. The PDS algorithms remove undesired vectors with less number of multiplications by calculating only the partial distance. An undesired vector is identified when its partial distance is larger than the full distance of the *current optimal vector*. These algorithms have been found to be effective for accelerating the encoding process of vector quantizers without increasing the distortion. Moreover, when applies to the kNN classifier, these algorithms do not increase the classification error rate. However, because the PDS is usually implemented by software, only a moderate acceleration can be achieved.

The objective of this paper is to present a novel hardware implementation for kNN classifier using FPGA technique. In addition to the employment of FPGAs, a subspace PDS with bitplane reduction and multiple coefficient accumulation is adopted for attaining higher throughput, higher flexibility, and lower area complexity. The PDS in the transform domain may have superior performance over the PDS in the original domain provided that the transform is orthogonal, and is able to compact the energy to few coefficients in the transform domain. In our implementation, a simple orthogonal wavelet, the Haar wavelet, is applied to each input test pattern because of its simplicity for VLSI realization. Coefficients in some highpass subbands will be truncated before fast search because they contain little energy of the input pattern. In addition, the bitplanes of the least significant bits (LSBs) of remaining coefficients can be removed because they have only a limited impact on results of the PDS. The partial distance can also be accumulated multiple coefficients at a time to further accelerate the throughput. The combination of subspace search, bitplane removal and multiple coefficient accumulation effectively enhances the search speed and reduce the codebook storage size at the expense of a slight degradation in performance.

The proposed implementation has been adopted as a custom logic block in the arithmetic logic unit (ALU) of the softcore NIOS processor running on 50 MHz. Custom instructions are also derived for accessing the custom logic block. The CPU time of the NIOS processor executing the PDS program with custom instructions is measured. Experiment results show that the CPU time is lower than that of 3.0-GHz Pentium processors executing the PDS programs without the support of custom hardware. In pratical applications, the proposed circuit may be beneficial for the implementation of smart camera for low network bandwidth consumption, which directly produces the classification results instead of delivering image raw data to main host computers for classification.

2 PDS for kNN Classifier

First we briefly introduce some basic facts of the wavelet transform [6]. Let $\mathbf{X}$ be the n-stage discrete wavelet transform (DWT) of a $2^n \times 2^n$ pixel vector $\mathbf{x}$. The DWT $\mathbf{X}$ is also a $2^n \times 2^n$ pixel block containing blocks $\mathbf{x}_{L0}$ and $\mathbf{x}_{Vi}, \mathbf{x}_{Hi}, \mathbf{x}_{Di}, i = 0, .., n-1$. In the DWT, the blocks $\mathbf{x}_{L(m-1)}$ (lowpass blocks), and $\mathbf{x}_{V(m-1)}, \mathbf{x}_{H(m-1)}, \mathbf{x}_{D(m-1)}$ (vertical, horizontal, and diagonal orientation

selective highpass blocks), $m = 1, .., n$, are obtained recursively from $\mathbf{x}_{Lm}$ with $\mathbf{x}_{Ln} = \mathbf{x}$, where the blocks $\mathbf{x}_{Lm}, \mathbf{x}_{Vm}, \mathbf{x}_{Hm}$ and $\mathbf{x}_{Dm}, m = 0, .., n-1$ are with dimension $2^m \times 2^m$ pixels, respectively.

In the kNN classifier, suppose there are N classes: $\omega_1, ..., \omega_N$. Each class ω_i is associated with a design set $\mathcal{S}_i$. Let $\mathcal{S} = \{\mathbf{y}^j, j = 1, ..., t\} = \cup_{i=1}^{N} \mathcal{S}_i$ be the union of the design sets for kNN classification, where t is the total number of vectors in the union. In addition, let $\mathbf{x}$ be the input vector to be classified. Both $\mathbf{x}$ and $\mathbf{y}^j$ have the same dimension $2^n \times 2^n$ pixels. In the original kNN classifier, the first k closest vectors to $\mathbf{x}$ among the vectors in $\mathcal{S}$ are first identified. Let $k_i, 1 \le i \le N$ be the number of vectors that belong to $\mathcal{S}_i$ (i.e. the class ω_i) in these k closest vectors. The kNN algorithm classifies the input vector $\mathbf{x}$ to class ω_p iff $p = \arg\max_i k_i$.

Let $\mathbf{Y}^j$ be the DWT of $\mathbf{y}^j$. In addition, let $D(\mathbf{u}, \mathbf{v}) = \sum_i (u_i - v_i)^2$ be the squared distance between $\mathbf{u}$ and $\mathbf{v}$. It can be shown that, for an orthogonal DWT,

$$D(\mathbf{x}, \mathbf{y}^j) = D(\mathbf{X}, \mathbf{Y}^j) \tag{1}$$

Let $\mathcal{F}_k = \{f_1, f_2, ..., f_k\}$ be the indices of the first k closest vectors to $\mathbf{x}$ among the vectors in $\mathcal{S}$, where $D(\mathbf{X}, \mathbf{Y}^{f_1}) \le D(\mathbf{X}, \mathbf{Y}^{f_2}) \le ... \le D(\mathbf{X}, \mathbf{Y}^{f_k})$.

The objective of the PDS algorithm is to reduce the computation time for finding $\mathcal{F}_k$ for kNN classification. Let X_i and Y_i^j be the i-th coefficient of $\mathbf{X}$ and $\mathbf{Y}^j$, respectively, where the coefficients in the wavelet domain are indexed in the zig-zag order. Moreover, let $D^q(\mathbf{X}, \mathbf{Y}^j) = \sum_{i=1}^{q} (X_i - Y_i^j)^2$ be the partial distance between $\mathbf{X}$ and $\mathbf{Y}^j$. Since $D(\mathbf{X}, \mathbf{Y}^j) > D^q(\mathbf{X}, \mathbf{Y}^j)$, it follows that

$$D(\mathbf{x}, \mathbf{y}^j) > D^q(\mathbf{X}, \mathbf{Y}^j). \tag{2}$$

Let $D_i, i = 1, ..., k$, be the squared distance between $\mathbf{x}$ and *current* $\mathbf{y}^{f_i}$ during PDS process. Before PDS is started, we set the initial *current* $f_i = 0$, $i = 1, .., k$, and the corresponding initial D_i is set to be $D_i = \infty$.

Starting from $\mathbf{Y}^1$, we check each vector (except initial *current* $\mathbf{Y}^{f_1}$ itself) until $\mathbf{Y}^t$ is reached. For each vector $\mathbf{Y}^j$ to be searched, we compute $|Y_1^j - X_1|$. Suppose $|Y_1^j - X_1| > \sqrt{D_k}$, then $\sqrt{D(\mathbf{x}, \mathbf{y}^j)} > \sqrt{D_k}$. Hence, j does not belong to $\mathcal{F}_k$ and $\mathbf{Y}_j$ can be rejected. If $|Y_1^j - X_1| < \sqrt{D_k}$, then we employ the following fast search process. Beginning with $q = 2$, for each value of q, $q = 2, .., 2^n \times 2^n$, we evaluate $D^q(\mathbf{X}, \mathbf{Y}^j)$. Suppose $D^q(\mathbf{X}, \mathbf{Y}^j) > D_k$, then $D(\mathbf{x}, \mathbf{y}^j) > D_k$ and $\mathbf{Y}^j$ can be rejected. Otherwise, we go to the next value of q and repeat the same process. Note that the partial distance $D^q(\mathbf{X}, \mathbf{Y}^j)$ can be expressed as

$$D^q(\mathbf{X}, \mathbf{Y}^j) = D^{q-1}(\mathbf{X}, \mathbf{Y}^j) + (X_q - Y_q^j)^2. \tag{3}$$

Therefore, the partial distance of the new q can use the partial distance of the previous q, and only the computation of $(X_q - Y_q^j)^2$ is necessary.

This PDS process is continued until $\mathbf{Y}^j$ is rejected or q reaches $2^n \times 2^n$. If $q = 2^n \times 2^n$, then we compare $D(\mathbf{x}, \mathbf{y}^j)$ with D_k. If $D(\mathbf{x}, \mathbf{y}^j) < D_k$, then we remove f_k from $\mathcal{F}_k$, insert j into $\mathcal{F}_k$, and re-sort elements in $\mathcal{F}_k$. After the final $\mathcal{F}_k$ is found, we then compute k_i for each class ω_i. The class having highest k_i

Table 1. The classification error rate and area complexity (LEs) of kNN design set for various l values with or without subspace search

Subspace	No	Yes	Yes
l	0	0	6
Classification error rate	5.41%	6.67%	7.08%
Design Set LEs	26706	15235	3912

is then identified as the class that $\mathbf{x}$ belongs to. This completes the PDS-based kNN classification.

3 The Architecture

3.1 PDS for Hardware Realization

The PDS adopted here for hardware realization features the subspace search, the bitplane reduction, and multiple-coefficient partial distance accumulation.

Subspace Search. Because the DWT is able to compact the energy of a vector to a lowpass subband, the PDS in the wavelet domain can be accelerated further by scanning only the coefficients in the lowpass subband. The kNN design set $\mathcal{C}$ for subspace search contains only $\mathbf{Y}_{Lm}^j$(i.e., the DWT coefficients of $\mathbf{y}_{Lm}^j$), $j = 1, ..., t$. Therefore, the subspace search is also beneficial for the VLSI realization of a kNN classifier since it significantly reduce the storage size of the design set.

Although the $\mathbf{Y}_{Lm}^j, j = 1, ..., t$, can be obtained offline, the $\mathbf{X}_{Lm}$ should be obtained online from a source vector $\mathbf{x}$. Therefore, the DWT computation is still necessary in the subspace PDS. We use the Haar wavelet for the DWT hardware realization because it has simple lowpass filter (i.e., impulse response $= \{\frac{1}{2}, \frac{1}{2}\}$) and highpass filer (i.e., impulse response $= \{-\frac{1}{2}, \frac{1}{2}\}$). Consequently, no multiplication is necessary for the DWT implementation.

Bitplane Reduction. In addition to its simplicity, another advantage of the Haar wavelet is that the coefficients are of finite precision. All coefficients can be precisely stored in the ROM for PDS operations. On the other hand, it may not be necessary to store all bitplanes in $\mathbf{X}_{Lm}$ since the removal of LSB bitplanes have limited impact on the PDS performance. Table 1 shows the classification error rate of kNN classifier based on subspace PDS with/without bitplane reduction. Area complexities of the ROM are also included for comparison purpose. The classification error rate is defined as the number of test vectors that are misclassified divided by the total number of test vectors. The area complexity is defined as the number of logic elements (LEs) required for the FPGA implementation of the ROM containing the design sets for kNN classification. The FPGA used for the measurement of the storage size is the Altera Stratix [8]. The bitplane reduction level, denoted by l, is defined as the number of LSB bitplanes removed from the subspace PDS.

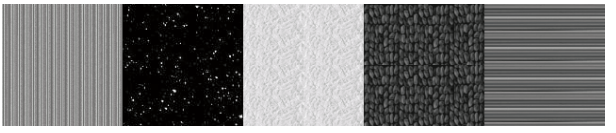

Fig. 1. Five textures for classification

Table 2. The area complexity (LEs) of VSDC unit for various l and ä values

	l	**0**	**2**	**4**	**6**
	1	46	40	34	28
	2	123	107	92	75
δ	**4**	278	242	206	170
	8	589	513	437	361
	16	1212	1056	900	744

In this experiment, there are 5 classes (i.e., $N = 5$). Each class represents a texture as shown in Fig. 1. The design set $\mathcal{C}_i$ associated with each class contains 256 vectors. Therefore, there are 1280 vectors ($t = 1280$) in the design set $\mathcal{C}$. The dimension of the vectors is 8×8 pixels (i.e., $n = 3$). For the subspace search, the dimension of $\mathbf{X}_{Lm}$ and $\mathbf{Y}_{Lm}^j$ is 4×4 pixels (i.e., $m = 2$).

As compared with the basic PDS process, the classification error rate is only slightly increased when using subspace PDS and removing 6 LSB bitplanes ($l = 6$). However, the deduction in ROM area complexity is quite substantial. In fact, the classification error rate is only increased by 1.67% (from 5.41% to 7.08%), but the deduction in area complexity is 85.35% (from 26706 LEs to 3912 LEs).

Multiple-Coefficient Partial Distance Accumulation. From eq.(3), it follows that the partial distance is accumulated one coefficient at a time in the basic PDS. Therefore, the hardware realization of the basic PDS requires only one multiplier. The speed of partial distance computation can be accelerated for enhancing the throughput by accumulating δ coefficients at a time. The partial distance is then computed by

$$D^q(\mathbf{X}_{Lm}, \mathbf{Y}_{Lm}^j) = D^{q-\delta}(\mathbf{X}_{Lm}, \mathbf{Y}_{Lm}^j) + \sum_{i=q-\delta+1}^{q} (X_i - Y_i^j)^2, \qquad (4)$$

where X_i and Y_i^j are the i-th coefficient of $\mathbf{X}_{Lm}$ and $\mathbf{Y}_{Lm}^j$, respectively. The size of the lowpass subband $2^m \times 2^m$ pixels should be a multiple of δ.

Table 2 shows the area complexity of the vector squared distance computation (VSDC) units with various δ and l values. The FPGA used for the area complexity measurement is also the Altera Stratix. It can be observed from the table that, although VSDC units with larger δ values have higher encoding throughput, their area complexity is also very large. Therefore, the employment

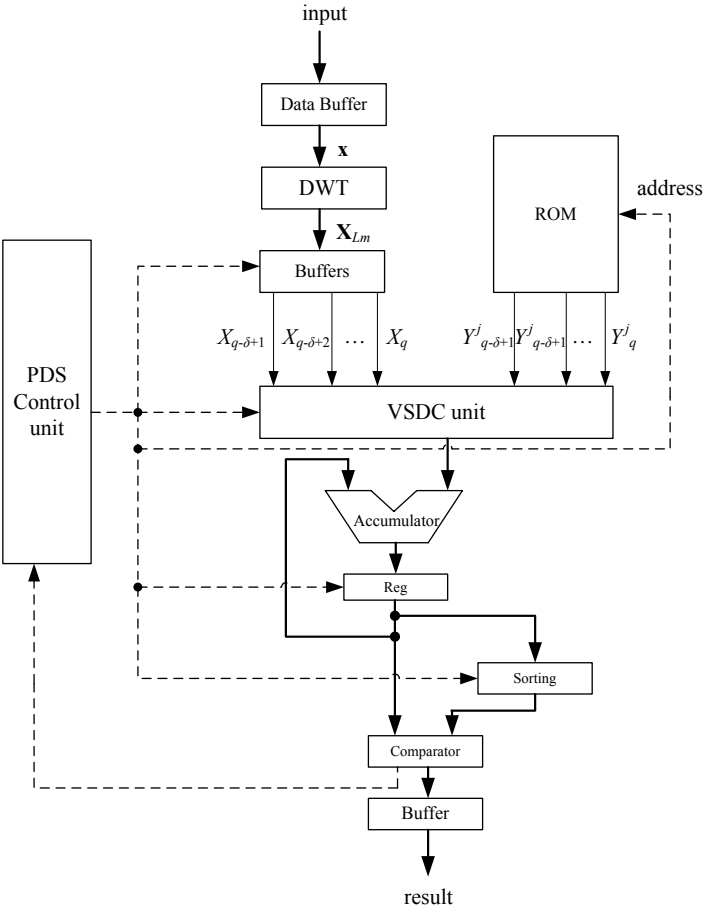

Fig. 2. The VLSI architecture of the proposed PDS algorithm

of VSDC with very large δ values may be difficult for some applications where the cost is an important concern. On the contrary, smaller δ values (e.g., $\delta = 4$) requires significantly lower area complexity while improving the throughput.

3.2 FPGA Implementation

The basic VLSI architecture for realizing the proposed PDS is shown in Fig. 2, which contains a ROM, a DWT unit, a VSDC unit, an accumulator, a comparator, a control unit, a sorting circuit, and a number of registers storing the values of intermediate results. The ROM contains $\mathbf{Y}^1_{Lm}, ..., \mathbf{Y}^t_{Lm}$ for the fast search. The DWT unit is used for computing $\mathbf{X}_{Lm}$ of an input vector $\mathbf{x}$. The VSDC and accumulator are used to compute and store the partial distance $D^q(\mathbf{X}_{Lm}, \mathbf{Y}^j_{Lm})$ for $q = \delta, 2\delta, ..., 2^m \times 2^m$. The comparator then compares $D^q(\mathbf{X}_{Lm}, \mathbf{Y}^j_{Lm})$ with D_k. The comparison results are reported to the control unit, which determines

Table 3. The latency of the kNN classifier for various ä values

δ	1	2	4	8	16
$\mathcal{L}$	1.4836	1.2096	1.0864	1.0307	1

whether the reset of the accumulator and the update of $\mathcal{F}_k$ are necessary. When $\mathcal{F}_k$ is required to be updated, the sorting circuit will be activated. The circuit first replaces current f_k by j and updates corresponding D_k. After that, the circuit sorts the elements of $\mathcal{F}_k$ in accordance with $D_i, i = 1, ..., k$. Each of the new $f_i, i = 1, ..., k$, and its corresponding D_i after sorting operations will be stored in registers for subsequent PDS comparisons.

Table 3 shows the latency of the VLSI architecture for various δ values. Let $\mathcal{T}(\mathbf{x})$ be the number of clock cycles required for the completion of the subspace PDS given an input vector $\mathbf{x}$. The latency $\mathcal{L}$ is then defined as

$$\mathcal{L} = \frac{1}{tW} \sum_{j=1}^{W} \mathcal{T}(\mathbf{x}^j), \tag{5}$$

where W is the total number of input vectors and t is the total number of codewords. For each input vector, the latency $\mathcal{L}$ then represents the average number of clock cycles required for the computation of each input vector. The design sets used in this experiment are identical to those in Table 1. Therefore, $N = 5$, $t = 1280$, $n = 3$, and $m = 2$.

It can be observed from the Table 3 that, the average latency of the PDS with a relatively small $\delta > 1$ for detecting an undesired codeword is close to 1. In addition, it also follows from Tables 1 and 2 that the subspace PDS with small δ and large l can lower the area complexity. Therefore, with a lower area complexity and a faster clock rate, the proposed PDS is able to achieve an average latency close 1 for squared distance calculation and undesired codeword detection.

3.3 The PDS Architecture as a Custom User Logic in a Softcore CPU

To physically measure the performance of the proposed architecture, the proposed PDS architecture has been implemented as a user logic in the ALU of the softcore NIOS CPU, which is a general purpose, pipelined, single-issued reduced instruction set computer (RISC) processor. The major distinction between normal microprocessors and softcore processors is that softcore processors have a re-configurable architecture. Therefore, custom VLSI architectures can be embedded in the softcore processors for physical performance measurement.

Fig. 3 shows the position of the kNN architecture in the ALU of the NIOS CPU. It can be observed from the figure that the kNN architecture and the basic ALU unit share the same input and output buses. The kNN architecture can be accessed by the custom instructions associated with the architecture.

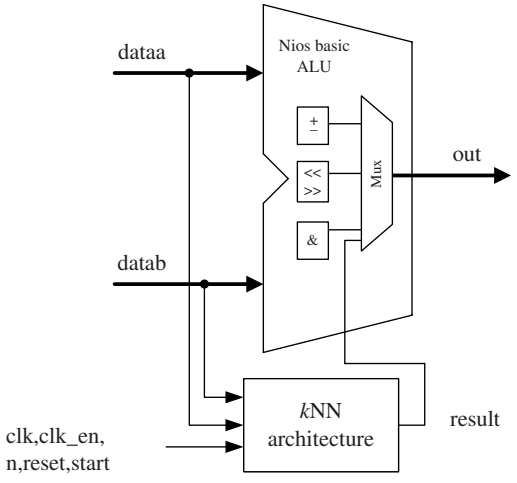

Fig. 3. The position of the kNN architecture in the ALU of the NIOS CPU

The custom instructions accessing the buffers and status register of the kNN architecture can be called using macros in C/C++. Therefore, the C program for kNN classification running on the NIOS CPU involves only the input vector loading and kNN results collection operations. The C code implementation is straightforward, and requires few computations. This can be beneficial for embedded systems where the CPU has limited computational capacity for high level language execution.

4 Experimental Results and Conclusions

This section presents some physical performance measurements of the proposed FPGA implementation. In our experiments, the dimension of vectors in the design set is 8×8 pixels (i.e., $n = 3$). Only $\mathbf{Y}_{Lm}^1, ..., \mathbf{Y}_{Lm}^t$ are stored for PDS, where the total number of vectors t in the design set is 1280. We set $m = 2$ so that the dimension of $\mathbf{Y}_{Lm}^j$ is 4×4 pixels. The bitplane reduction level is $l = 6$. The VSDCs are implemented with $\delta = 4$. There are $N = 5$ textures for classification as shown in Fig. 1. We also set $k = 5$ for the kNN classification.

The whole kNN system consisting of the NIOS CPU embedded by the proposed kNN architecture is implemented by the Altera Stratix 1P1S40 FPGA, which has maximum number of 41250 LEs [8]. The operating frequency of the system is 50 MHz, and the system consumes only 6391 LEs for the implementation of both NIOS CPU and subspace PDS circuit. Therefore, the area complexity of the proposed circuit is significantly less than the maximum capacity of the target FPGA device.

Table 4 compares the execution time of NIOS with that of the Pentium IV for various PDS operations, where the execution time is defined as the average

Table 4. The CPU time of various kNN systems

CPU type	PentiumIV 3.0 GHz		PentiumIV 1.8 GHz		NIOS 50 MHz
algorithm	*k*NN	Subspace PDS	*k*NN	Subspace PDS	Subspace
Implementation	Software	Software	Software	Software	Hardware/ Software Codesign
CPU time (μs)	16079.2	262.5	24912.5	497.5	137.77

Table 5. Comparison between kNN and HVQ

FPGA Implementation	FPGA Device	Number of codewords	Vector Dimension	Number of LEs	Maximum Clock Rate
Subspace *k*NN (*k*=5)	Altera Stratix EP1S40	1280	64	2415	50 MHz
Hierarchical VQ[1]	Xilinx Vertex 400	256	64	6276	16MHz

CPU time (in μs) required for identifying the final class to each input vector. The NIOS CPU is running with the support of the proposed hardware; whereas, the Pentium IV CPU is executing solely on C codes. Note that, for the NIOS systems with software/hardware codesigns, the measurements in fact cover the complete execution process for codeword search including the memory accesses (the fetching of NIOS instructions and source vectors), buffer filling, executions of NIOS instructions, executions of PDS subspace hardware, and the retrieval of encoding results.

It can be observed from Table 4 that the average CPU time of the NIOS and Pentium IV 3.0 GHz are 137.77 μs and 262.5 μs, respectively. Therefore, although the operating frequency of the NIOS CPU is only 50 MHz, its average execution time is still lower than that of the Pentium IV CPU operating at 3.0 GHz. Table 4 also includes the CPU time of the basic kNN algorithm executed in Pentium IV without PDS. In this case, the CPU time is 16079.2 μs for Pentium IV 3.0 GHz CPU, which is 116.71 times longer then the NIOS CPU with hardware acceleration.

To further access the performance of the proposed implementation, Table 5 gives a comparison of our subspace PDS design with the FPGA implementation of the vector quantizer (VQ) presented in [1], which aims to reduce the computational cost for VQ encoding. We can view the VQ encoding circuit as a special kNN classification circuit with $k = 1$. The softcore CPU is not adopted as test bed in [1]. Therefore, only the kNN classfication circuits are considered for comparison. Consequently, in Table 5, the area complexity of the subspace PDS implementation is only 2415 LEs. Although our circuit contains larger number of vectors in the design set, we can see from the table that our subspace PDS architecture requires significantly lower area complexity. The FPGA implementation in [1] has higher area cost because it is based on the hierarchical VQ (HVQ), which needs extra intermediate vectors/nodes to build a tree structure

accelerating the VQ encoding process. The tree structure requires large area overhead for the FPGA implementation. By contrast, our design achieves fast computation while reducing the area complexity by the employment of subspace search and bitplane reduction. All these facts demonstrate the effectiveness of the proposed architecture and implementation.

References

1. Bracco, M., Ridella, S., Zunino, R.: Digital Implementation of Hierarchical Vector Quantization. IEEE Trans. Neural Networks 14, 1072–1084 (2003)
2. 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)
3. Hwang, W.J., Wen, K.W.: Fast kNN Classification Algorithm Based on Partial Distance Search. Electronics letters. 34, 2062–2063 (1998)
4. Mcnames, J.: Rotated Partial Distance Search for Faster Vector Quantization Encoding. IEEE Signal Processing Letters pp. 244–246 (2000)
5. Ridella, S., Rovetta, S., Zunino, R.: K-Winner Machines for Pattern Classification. IEEE Trans. Neural Networks 12, 371–385 (2001)
6. Vetterli, M., Kovacevic, J.: Wavelets and Subband Coding. Prentice Hall, New Jersy (1995)
7. Xie, A., Laszlo, C.A., Ward, R.K.: Vector Quantization Technique for Nonparametric Classifier Design. IEEE Trans. Pattern Anal. Machine Intell 15, 1326–1330 (1993)
8. Stratix Device Handbook `http://www.altera.com/literature/lit-stx.jsp` (2005)
9. Custom Instructions for NIOS Embedded Processors, Application Notes 188 (2002) `http://www.altera.com/literature/lit-nio.jsp`