# We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

6,900
Open access books available

185,000
International authors and editors

200M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities




WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

## Interested in publishing with us?
## Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com



# 9

# FPGA Implementation for GHA-Based Texture Classification

Shiow-Jyu Lin[1,2], Kun-Hung Lin[1] and Wen-Jyi Hwang[1]
[1]*Department of Computer Science and Information Engineering, National Taiwan Normal University and*
[2]*Department of Electronic Engineering, National Ilan University*
*Taiwan*

## 1. Introduction

Principal components analysis (PCA) (Alpaydin, 2010; Jolliffe, 2002) is an effective unsupervised feature extraction algorithm for pattern recognition, classification, computer vision or data compression (Bravo et al., 2010; Zhang et al., 2006; Kim et al., 2005; Liying & Weiwei, 2009; Pavan et al., 2007; Qian & James, 2008). The goal of PCA is to obtain a compact and accurate representation of the data that reduces or eliminates statistically redundant components. Basic approaches for PCA involve the computation of the covariance matrix and the extraction of eigenvalues and eigenvectors. A drawback of the basic approaches is the high computational complexity and large memory requirement for data with high vector dimension. Therefore, these approaches may not be well suited for real time applications requiring fast feature extraction.

A number of fast algorithms (Dogaru et al., 2004; El-Bakry, 2006; Gunter et al., 2007; Sajid et al., 2008; Sharma & Paliwal, 2007) have been proposed to reduce the computation time of PCA. However, only moderate acceleration can be achieved because most of these algorithms are based on software. Although hardware implementation of PCA and its variants are possible, large storage size and complicated circuit control management are usually necessary. The PCA hardware implementation may therefore be possible only for small dimensions (Boonkumklao et al., 2001; Chen & Han, 2009).

An alternative for the PCA implementation is to use the generalized Hebbian algorithm (GHA) (Haykin, 2009; Oja, 1982; Sanger, 1989). The principal computation by the GHA is based on an effective incremental updating scheme for reducing memory utilization. Nevertheless, slow convergence of the GHA (Karhunen & Joutsensalo, 1995) is usually observed. A large number of iterations therefore is required, resulting in long computational time for many GHA-based algorithms. The hardware implementation of GHA has been found to be effective for reducing the computation time. However, since the number of multipliers in the circuit grows with the dimension, the circuits may be suitable only for PCA with small dimensions. Although analog GHA hardware architectures (Carvajal et al., 2007; 2009) can be used to lift the constraints on the vector dimensions, these architectures are difficult to be directly used for digital devices.

In light of the facts stated above, a digital GHA hardware architecture capable of performing fast PCA for large vector dimension is presented. Although large amount of arithmetic computations are required for GHA, the proposed architecture is able to achieve fast training with low area cost. The proposed architectures can be divided into three parts: the synaptic weight updating (SWU) unit, the principal components computing (PCC) unit, and memory unit. The memory unit is the on-chip memory storing synaptic weight vectors. Based on the synaptic weight vectors stored in the memory unit, the SWU and PCC units are then used to compute the principal components and update the synaptic weight vectors, respectively.

In the SWU unit, one synaptic weight vector is computed at a time. The results of precedent weight vectors will be used for the computation of subsequent weight vectors for expediting training speed. In addition, the computation of different weight vectors shares the same circuit for lowering the area cost. Moreover, in the PCC unit, the input vectors are allowed to be separated into smaller segments for the delivery over data bus with limited width. Both the SWU and PCC units can also operate concurrently to further enhance the throughput.

To demonstrate the effectiveness of the proposed architecture, a texture classification system on a system-on-programmable-chip (SOPC) platform is constructed. The system consists of the proposed architecture, a softcore NIOS II processor (Altera Corp., 2010), a DMA controller, and a SDRAM. The proposed architecture is adopted for finding the PCA transform by the GHA training, where the training vectors are stored in the SDRAM. The DMA controller is used for the DMA delivery of the training vectors. The softcore processor is only used for coordinating the SOPC system. It does not participate the GHA training process. As compared with its software counterpart running on Intel *i*7 CPU, our system has significantly lower computational time for large training set. All these facts demonstrate the effectiveness of the proposed architecture.

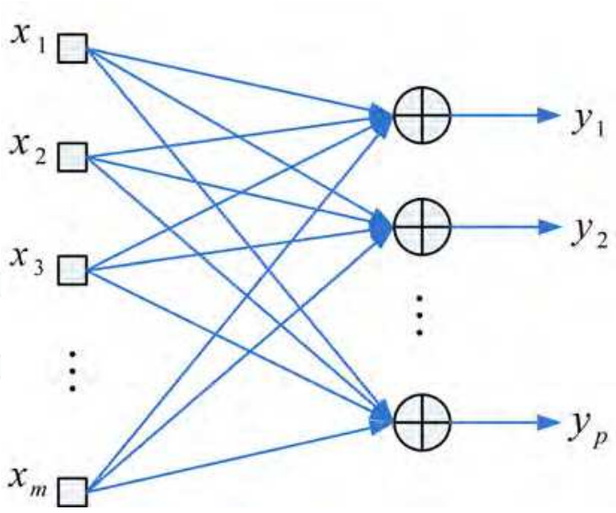



Fig. 1. The neural model for the GHA.

## 2. Preliminaries

Figure 1 shows the neural model for GHA, where $\mathbf{x}(n) = [x_1(n), ..., x_m(n)]^T$, and $\mathbf{y}(n) = [y_1(n), ..., y_p(n)]^T$ are the input and output vectors to the GHA model, respectively. The output

vector $\mathbf{y}(n)$ is related to the input vector $\mathbf{x}(n)$ by

$$y_j(n) = \sum_{i=1}^{m} w_{ji}(n)x_i(n), \tag{1}$$

where the $w_{ji}(n)$ stands for the weight from the $i$-th synapse to the $j$-th neuron at iteration $n$. Each synaptic weight vector $\mathbf{w}_j(n)$ is adapted by the Hebbian learning rule:

$$w_{ji}(n+1) = w_{ji}(n) + \eta[y_j(n)x_i(n) - y_j(n)\sum_{k=1}^{j} w_{ki}(n)y_k(n)], \tag{2}$$

where $\eta$ denotes the learning rate. After a large number of iterative computation and adaptation, $\mathbf{w}_j(n)$ will asymptotically approach to the eigenvector associated with the $j$-th principal component $\lambda_j$ of the input vector, where $\lambda_1 > \lambda_2 > ... > \lambda_p$. To reduce the complexity of computing implementation, eq.(2) can be rewritten as

$$w_{ji}(n+1) = w_{ji}(n) + \eta y_j(n)[x_i(n) - \sum_{k=1}^{j} w_{ki}(n)y_k(n)]. \tag{3}$$

A more detailed discussion of GHA can be found in (Haykin, 2009; Sanger, 1989)

## 3. The proposed GHA architecture

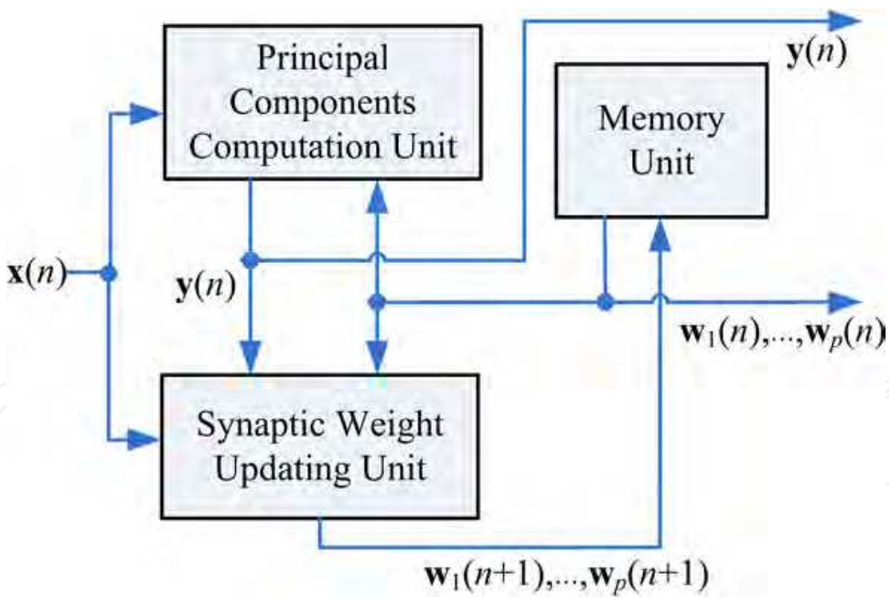



Fig. 2. The proposed GHA architecture.

As shown in Figure 2, the proposed GHA architecture consists of three functional units: the memory unit, the synaptic weight updating (SWU) unit, and the principal components computing (PCC) unit. The memory unit is used for storing the *current* synaptic weight vectors. Assume the *current* synaptic weight vectors $\mathbf{w}_j(n), j = 1, ..., p$, are now stored in the memory unit. In addition, the input vector $\mathbf{x}(n)$ is available. Based on $\mathbf{x}(n)$ and $\mathbf{w}_j(n), j = 1, ..., p$, the goal of PCC unit is to compute output vector $\mathbf{y}(n)$. Using $\mathbf{x}(n)$, $\mathbf{y}(n)$

and $\mathbf{w}_j(n), j = 1, ..., p$, the SWU unit produces the new synaptic weight vectors $\mathbf{w}_j(n+1), j = 1, ..., p$. It can be observed from Figure 2 that the new synaptic weight vectors will be stored back to the memory unit for subsequent training.

### 3.1 SWU unit

The design of SWU unit is based on eq.(3). Although the direct implementation of eq.(3) is possible, it will consume large hardware resources. To further elaborate this fact, we first see from eq.(3) that the computation of $w_{ji}(n+1)$ and $w_{ri}(n+1)$ shares the same term $\sum_{k=1}^{r} w_{ki}(n)y_k(n)$ when $r \leq j$. Consequently, independent implementation of $w_{ji}(n+1)$ and $w_{ri}(n+1)$ by hardware using eq.(3) will result in large hardware resource overhead.

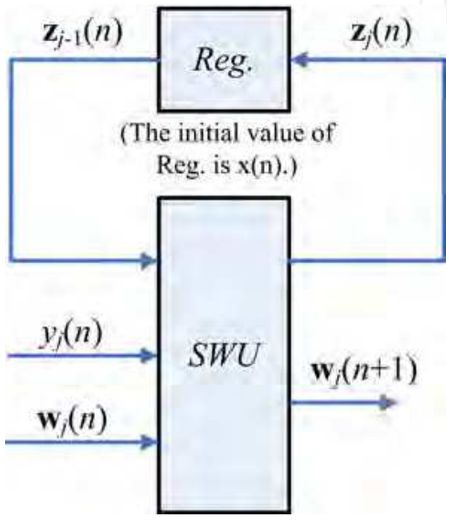



Fig. 3. The hardware implementation of eqs.(5) and (6).

To reduce the resource consumption, we first define a vector $z_{ji}(n)$ as

$$z_{ji}(n) = x_i(n) - \sum_{k=1}^{j} w_{ki}(n)y_k(n), j = 1, ..., p, \tag{4}$$

and $\mathbf{z}_j(n) = [z_{j1}(n), ..., z_{jm}(n)]^T$. Integrating eq.(3) and (4), we obtain

$$w_{ji}(n+1) = w_{ji}(n) + \eta y_j(n)z_{ji}(n), \tag{5}$$

where $z_{ji}(n)$ can be obtained from $z_{(j-1)i}(n)$ by

$$z_{ji}(n) = z_{(j-1)i}(n) - w_{ji}(n)y_j(n), j = 2, ..., p. \tag{6}$$

When $j = 1$, from eq.(4) and (6), it follows that

$$z_{0i}(n) = x_i(n). \tag{7}$$

Figure 3 depicts the hardware implementation of eqs.(5) and (6). As shown in the figure, the SWU unit produces one synaptic weight vector at a time. The computation of $\mathbf{w}_j(n+1)$, the $j$-th weight vector at the iteration $n+1$, requires the $\mathbf{z}_{j-1}(n)$, $\mathbf{y}(n)$ and $\mathbf{w}_j(n)$ as inputs.

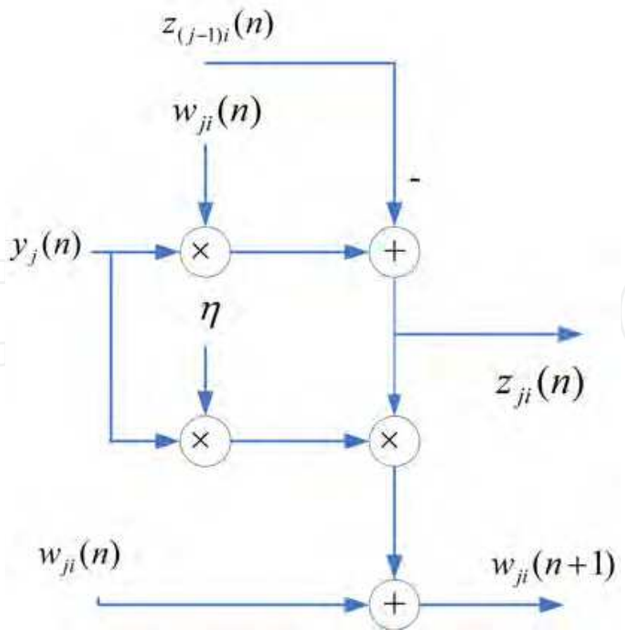



Fig. 4. The architecture of each module in SWU unit.

In addition to $\mathbf{w}_j(n+1)$, the SWU unit also produces $\mathbf{z}_j(n)$, which will then be used for the computation of $\mathbf{w}_{j+1}(n+1)$. Hardware resource consumption can then be effectively reduced.

One way to implement the SWU unit is to produce $\mathbf{w}_j(n+1)$ and $\mathbf{z}_j(n)$ in one shot. In SWU unit, $m$ identical modules may be required because the dimension of vectors is $m$. Figure 4 shows the architecture of each module. The area cost of the SWU unit then will grows linearly with $m$. To further reduce the area cost, each of the output vectors $\mathbf{w}_j(n+1)$ and $\mathbf{z}_j(n)$ are separated into $b$ segments, where each segment contains $q$ elements. The SWU unit only computes one segment of $\mathbf{w}_j(n+1)$ and $\mathbf{z}_j(n)$ at a time. Therefore, it will take $b$ clock cycles to produce complete $\mathbf{w}_j(n+1)$ and $\mathbf{z}_j(n)$.

Let

$$\hat{\mathbf{w}}_{j,k}(n) = [w_{j,(k-1)q+1}(n), ..., w_{j,(k-1)q+q}(n)]^T, k = 1, ..., b. \tag{8}$$

and

$$\hat{\mathbf{z}}_{j,k}(n) = [z_{j,(k-1)q+1}(n), ..., z_{j,(k-1)q+q}(n)]^T, k = 1, ..., b. \tag{9}$$

be the $k$-th segment of $\mathbf{w}_j(n)$ and $\mathbf{z}_j(n)$, respectively. The computation $\mathbf{w}_j(n)$ and $\mathbf{z}_j(n)$ take $b$ clock cycles. At the $k$-th clock cycle, $k = 1, ..., b$, the SWU unit computes $\hat{\mathbf{w}}_{j,k}(n+1)$ and $\hat{\mathbf{z}}_{j,k}(n)$. Because each of $\hat{\mathbf{w}}_{j,k}(n+1)$ and $\hat{\mathbf{z}}_{j,k}(n)$ contains only $q$ elements, the SWU unit consists of $q$ identical modules. The architecture of each module is also shown in Figure 4. The SWU unit can be used for GHA with different vector dimension $m$. As $m$ increases, the area cost therefore remains the same at the expense of large number of clock cycles $b$ for the computation of $\hat{\mathbf{w}}_{j,k}(n+1)$ and $\hat{\mathbf{z}}_{j,k}(n)$.

Figures 5, 6 and 7 show the operation of the $q$ modules. For the sake of simplicity, the computation of the first weight vector $\mathbf{w}_1(n+1)$ (i.e., $j = 1$) and the corresponding $\mathbf{z}_1(n)$ are considered in the figures. Based on eq.(7), the input vector $\mathbf{z}_{j-1}(n)$ is actually the training

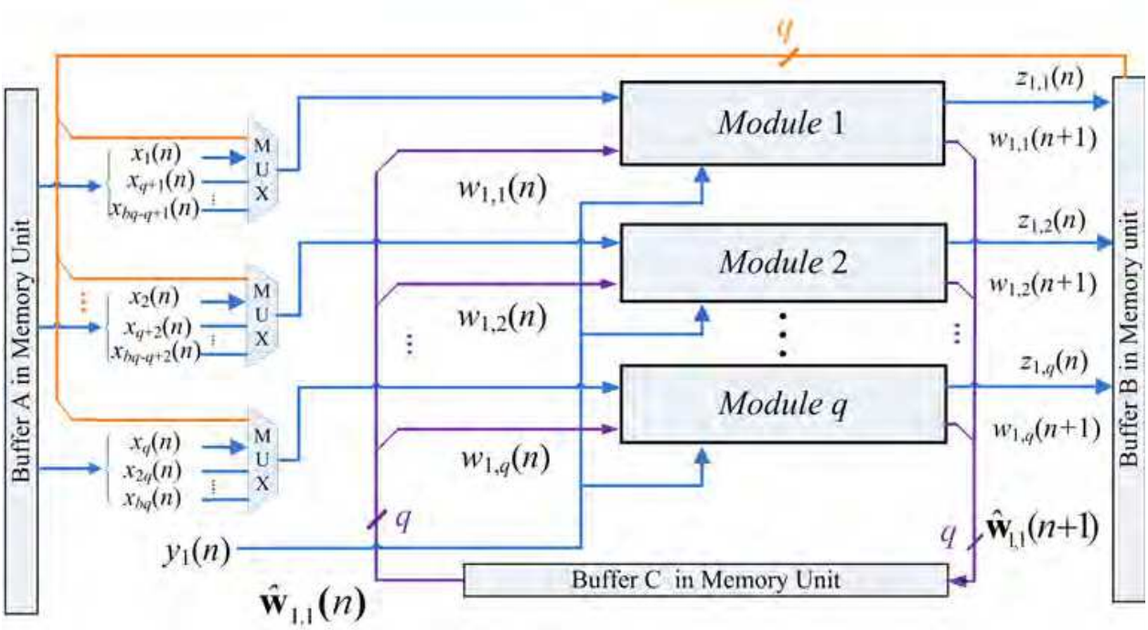



Fig. 5. The operation of SWU unit for computing the first segment of $\mathbf{w}_1(n+1)$.

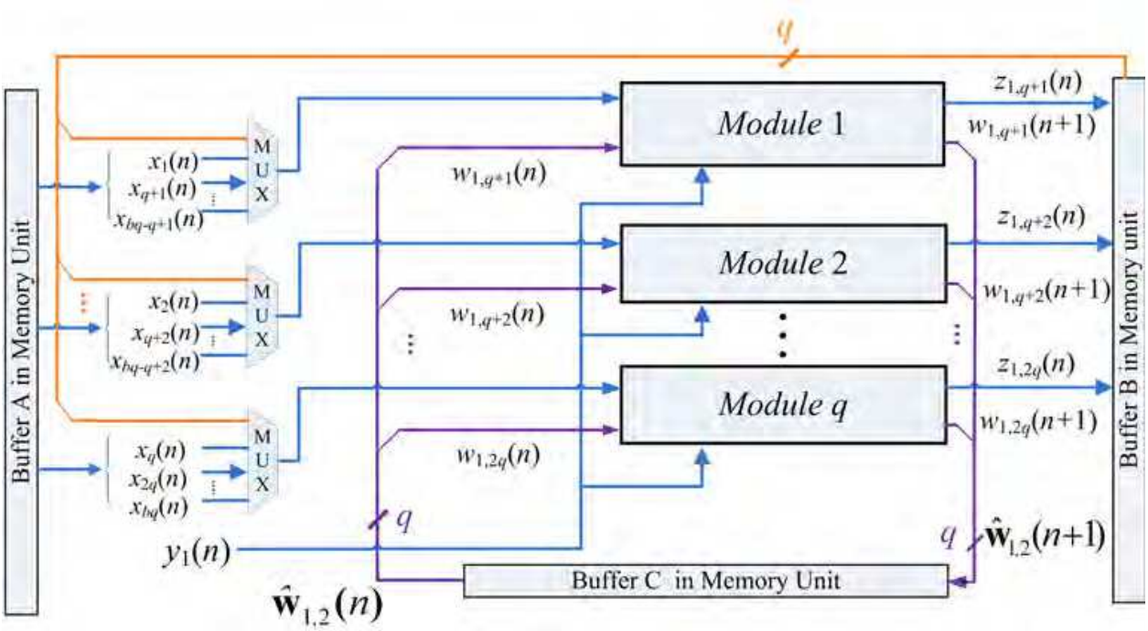



Fig. 6. The operation of SWU unit for computing the second segment of $\mathbf{w}_1(n+1)$.

vector $\mathbf{x}(n)$, which is also separated into $b$ segments, where the $k$-th segment is given by

$$\hat{\mathbf{z}}_{0,k}(n) = [x_{(k-1)q+1}(n), ..., x_{(k-1)q+q}(n)]^T, k = 1, ..., b. \tag{10}$$

They are then multiplexed to the $q$ modules. The $\hat{\mathbf{z}}_{0,1}(n)$ and $\hat{\mathbf{w}}_{1,1}(n)$ are used for the computation of $\hat{\mathbf{z}}_{1,1}(n)$ and $\hat{\mathbf{w}}_{1,1}(n+1)$ in Figure 5. Similarly, the $\hat{\mathbf{z}}_{0,k}(n)$ and $\hat{\mathbf{w}}_{1,k}(n)$ are

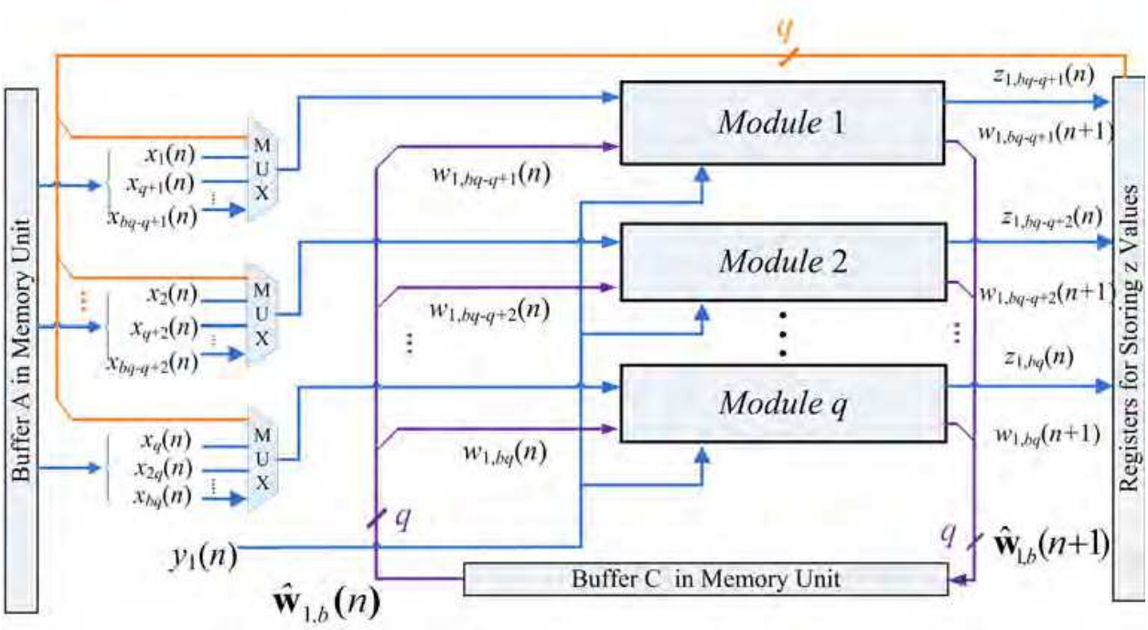



Fig. 7. The operation of SWU unit for computing the $b$-th segment of $\mathbf{w}_1(n+1)$.

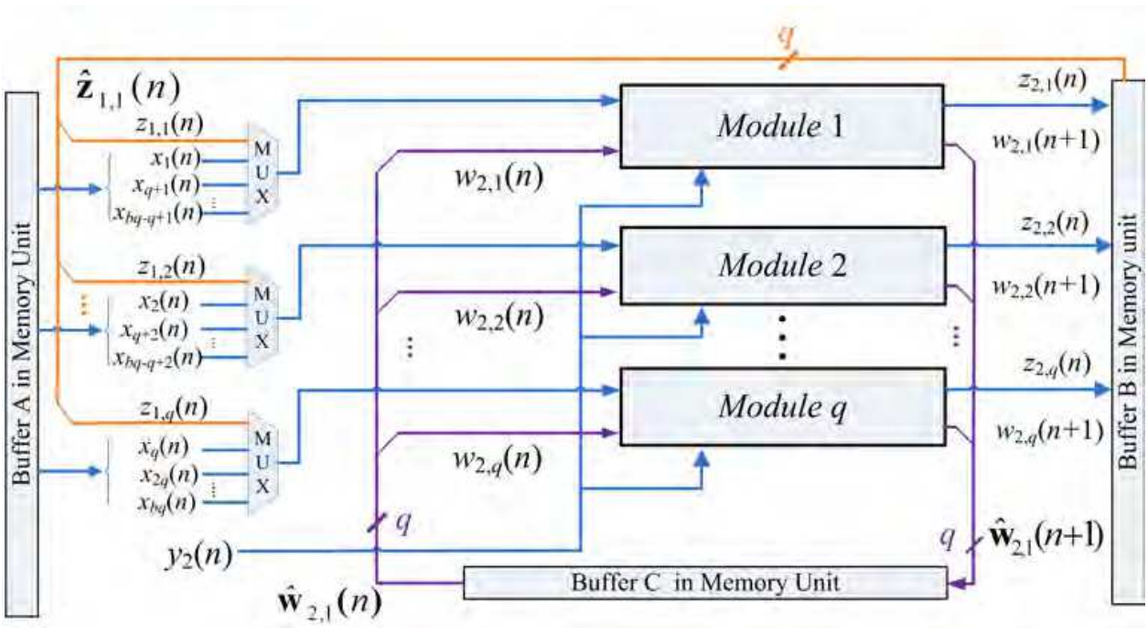



Fig. 8. The operation of SWU unit for computing the first segment of $\mathbf{w}_2(n+1)$.

used for the computation of $\hat{\mathbf{z}}_{1,k}(n)$ and $\hat{\mathbf{w}}_{1,k}(n+1)$ in Figures 6 and 7 for $k=2$ and $k=b$, respectively.

After the computation of $\mathbf{w}_1(n+1)$ is completed, the vector $\mathbf{z}_1(n)$ is available as well. The vector $\mathbf{z}_1(n)$ is then used for the computation of $\mathbf{w}_2(n+1)$. Figure 8 shows the computation of the first segment of $\mathbf{w}_2(n+1)$ (i.e., $\hat{\mathbf{w}}_{2,1}(n+1)$) based on the first segment of $\mathbf{z}_1(n)$ (i.e., $\hat{\mathbf{z}}_{1,1}(n)$). The same process proceeds for the subsequent segments until the computation of

the entire vectors $\mathbf{w}_2(n+1)$ and $\mathbf{z}_2(n)$ are completed. The vector $\mathbf{z}_2(n)$ is then used for the computation of $\mathbf{w}_3(n+1)$. The weight vector updating process at the iteration $n+1$ will be completed until the SWU unit produces the weight vector $\mathbf{w}_p(n+1)$.

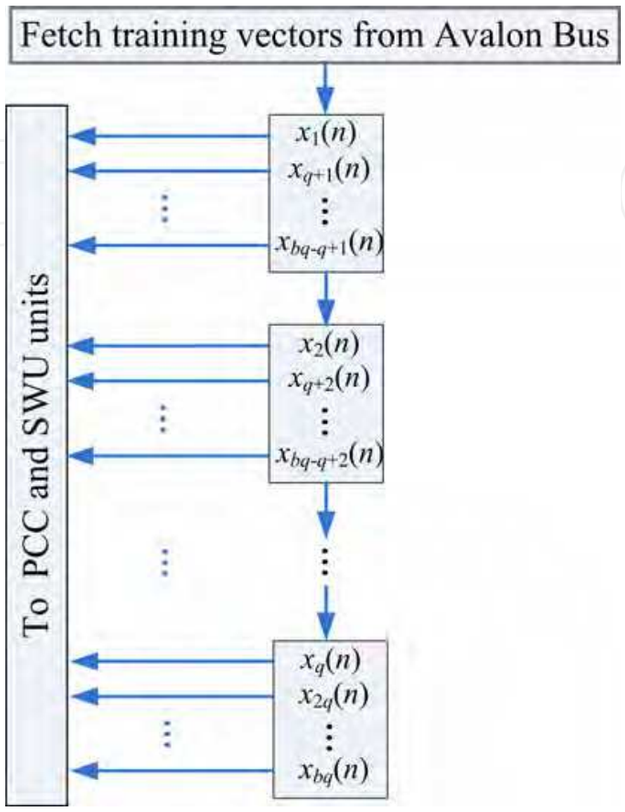



Fig. 9. The architecture of Buffer A in memory unit.

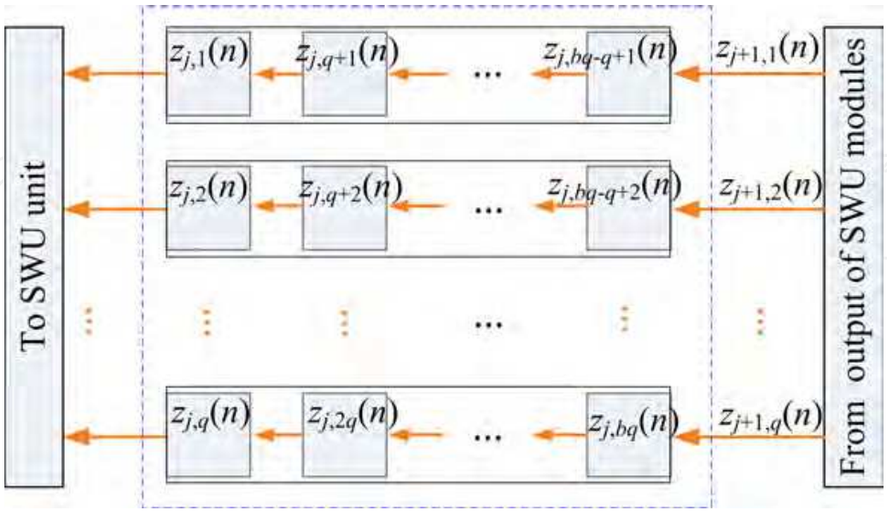



Fig. 10. The architecture of Buffer B in memory unit.

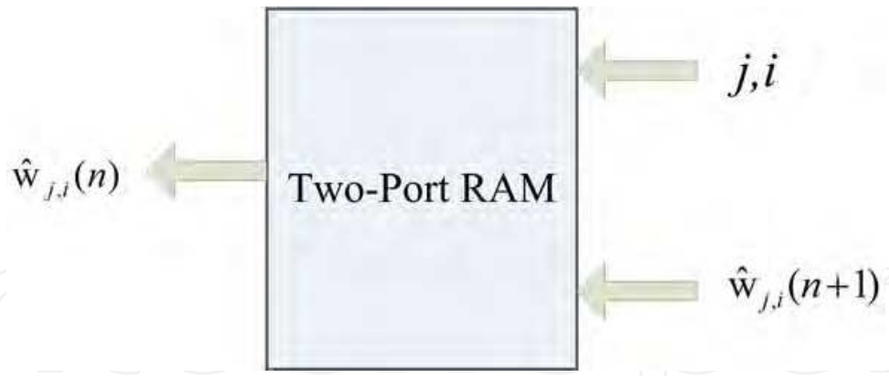



Fig. 11. The architecture of Buffer C in memory unit.

### 3.2 Memory unit

The memory unit contains three buffers: Buffer A, Buffer B and Buffer C. As shown in Figure 9, Buffer A stores training vector $\mathbf{x}(n)$. It consists of $q$ sub-buffers, where each sub-buffer contains $b$ elements. All the sub-buffers are connected to the SWU and PCC units.

The architecture of Buffer B is depicted in Figure 10, which holds the values of $\mathbf{z}_j(n)$. Each segment of $\mathbf{z}_j(n)$ computed from SWU unit is stored in Buffer B. After all the segments are produced, the Buffer B then deliver the segments of $\mathbf{z}_j(n)$ to SWU unit in the first-in-first-out (FIFO) fashion.

The Buffer C is used for storing the synaptic weight vectors $\mathbf{w}_j(n)$, $j = 1, ..., p$. It is a two-port RAM for reading and writing weight vectors, as revealed in Figure 11. The address for the RAM is expressed in terms of indices $j$ and $i$ for reading or writing the $i$-th segment of the weight vector $\mathbf{w}_j(n)$.

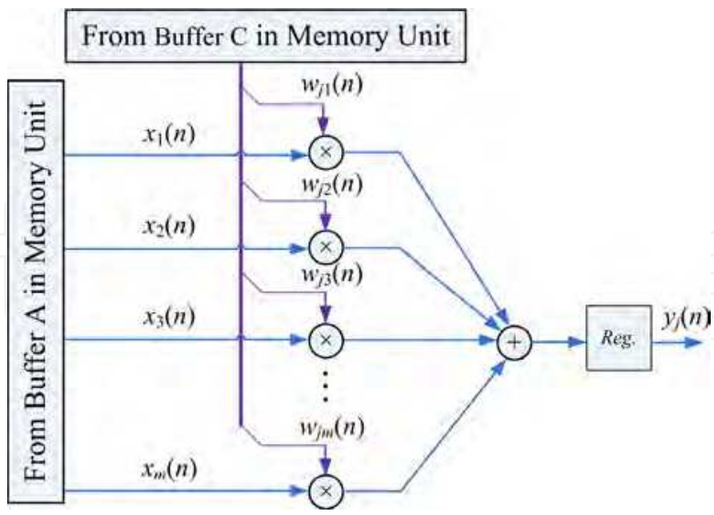



Fig. 12. The architecture of PCC unit.

### 3.3 PCC unit

The PCC operations are based on eq.(1). Therefore, the PCC unit of the proposed architecture contains adders and multipliers. Figure 12 shows the architecture of PCC. The training vector

$\mathbf{x}(n)$ and synaptic weight vector $\mathbf{w}_j(n)$ are obtained from the Buffer A and Buffer C of the memory unit, respectively. When both $\mathbf{x}(n)$ and $\mathbf{w}_j(n)$ are available, the proposed PCC unit then computes $y_j(n)$. Note that, after the $y_j(n)$ is obtained, the SWU unit can then compute $\mathbf{w}_j(n+1)$. Figure 13 reveals the timing diagram of the proposed architecture. It can be observed from Figure 13 that both the $y_{j+1}(n)$ and $\mathbf{w}_j(n+1)$ are computed concurrently. The throughput of the proposed architecture can then be effectively enhanced.

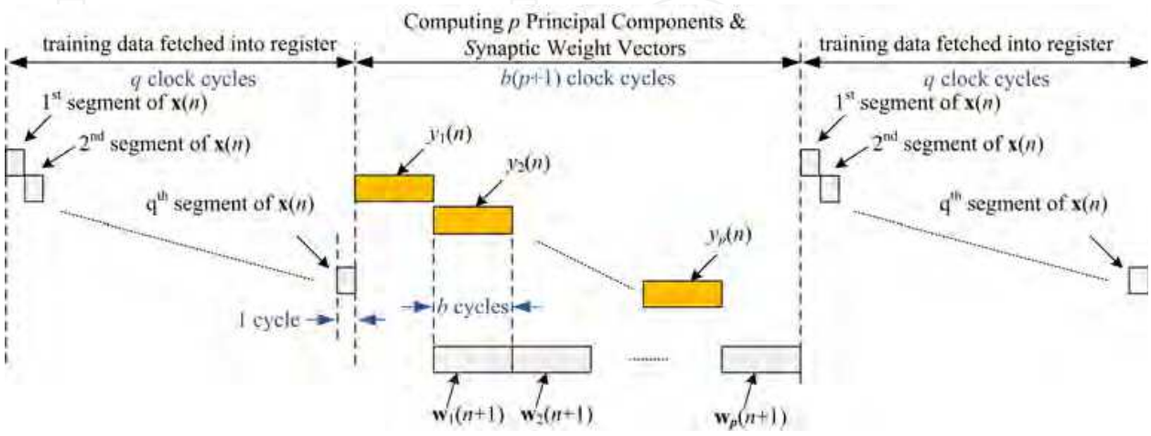



Fig. 13. The timing diagram of the proposed architecture.

### 3.4 SOPC-based GHA training system

The proposed architecture is used as a custom user logic in a SOPC system consisting of softcore NIOS CPU (Altera Corp., 2010), DMA controller and SDRAM, as depicted in Figure 14. All training vectors are stored in the SDRAM and then transported to the proposed circuit via the Avalon bus. The softcore NIOS CPU runs on a simple software to coordinate different components, including the proposed custom circuit in the SOPC. The proposed circuit operates as a hardware accelerator for GHA training. The resulting SOPC system is able to perform efficient on-chip training for GHA-based applications.

## 4. Experimental results

This section presents some experimental results of the proposed architecture applied to texture classification. The target FPGA device for all the experiments in this paper is Altera Cyclone III (Altera Corp., 2010). The design platform is Altera Quartus II with SOPC Builder and NIOS II IDE. Two sets of textures are considered in the experiments. The first set of textures, shown in Figure 15, consists of three different textures. The second set of textures is revealed in Figure 16, which contains four different textures. The size of each texture in Figures 15 and 16 is $320 \times 320$.

In the experiment, the principal component based $k$ nearest neighbor (PC-$k$NN) rule is adopted for texture classification. Two steps are involved in the PC-$k$NN rule. In the first step, the GHA is applied to the input vectors to transform $m$ dimensional data into $p$ principal components. The synaptic weight vectors after the convergence of GHA training are adopted to span the linear transformation matrix. In the second step, the $k$NN method is applied to the principal subspace for texture classification.

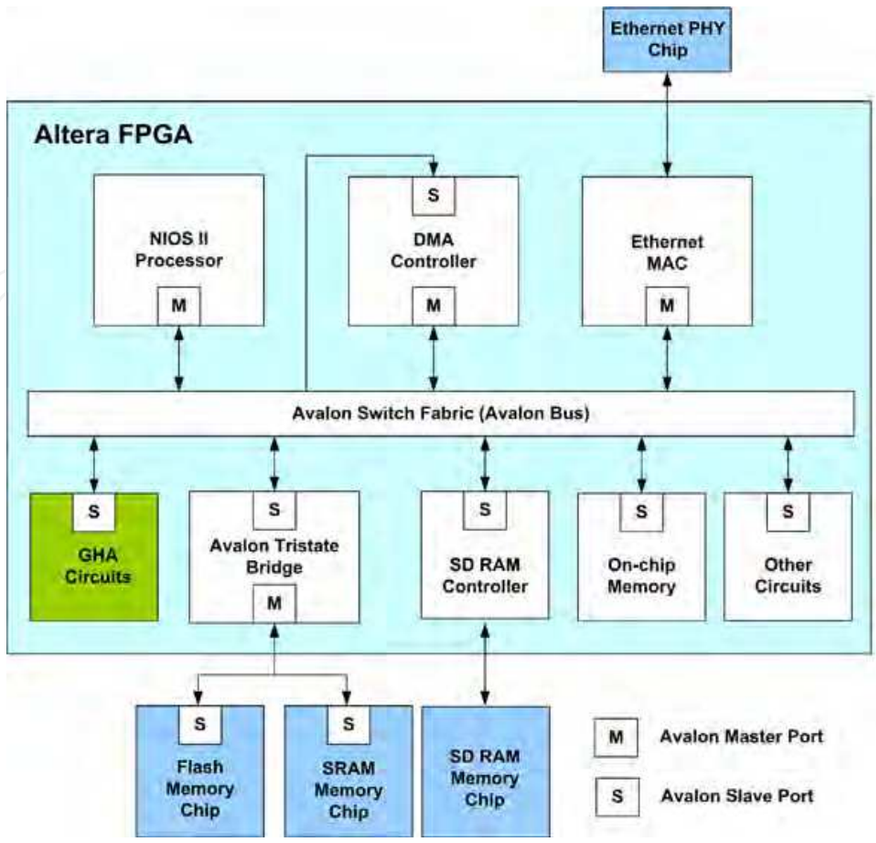



Fig. 14. The SOPC system for implementing GHA.

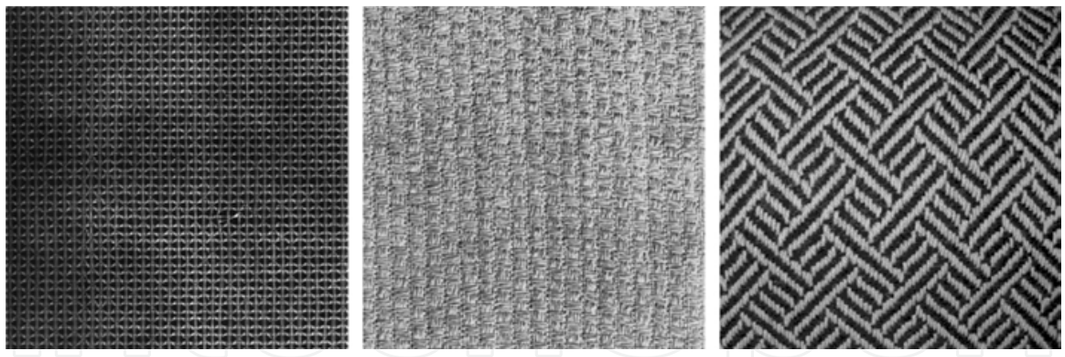


Fig. 15. The first set of textures for the experiments.

Figures 17 and 18 show the distribution of classification success rates (CSR) of the proposed architecture for the texture sets in Figures 15 and 16, respectively. The classification success rate is defined as the number of test vectors which are successfully classified divided by the total number of test vectors. The number of principal components is $p = 4$. The vector dimension is $m = 16 \times 16$. The distribution is based on 20 independent GHA training processes. The distribution of the architecture presented in (Lin et al., 2011) with the same $p$ is also included for comparison purpose. The vector dimension for (Lin et al., 2011) is $m = 4 \times 4$.

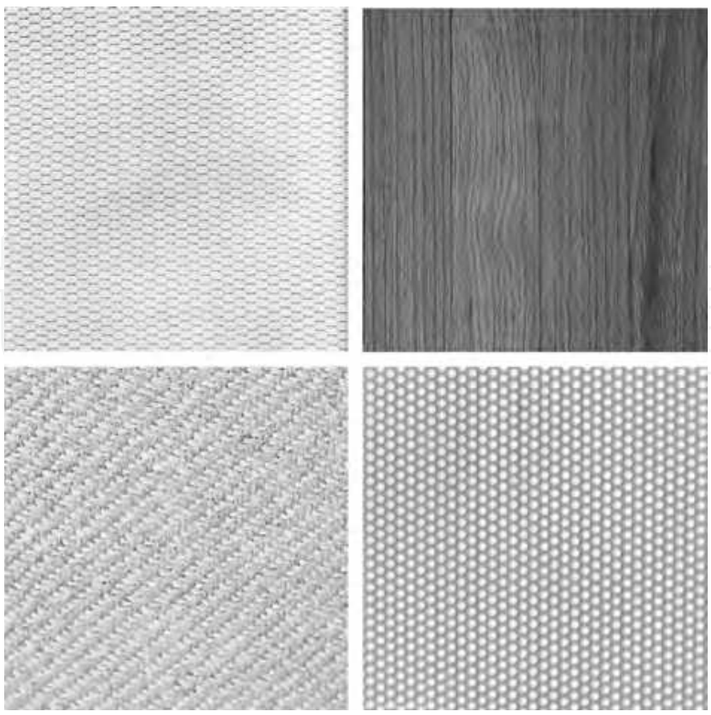


Fig. 16. The second set of textures for the experiments.

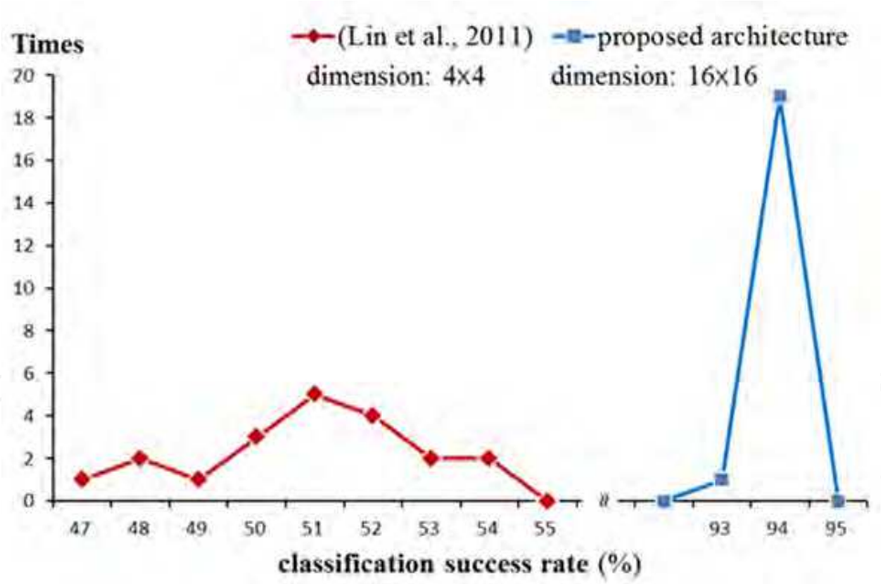



Fig. 17. The distribution of CSR of the proposed architecture for the texture set in Figure 15.

It can be observed from Figures 17 and 18 that the proposed architecture has better CSR. This is because the vector dimension of the proposed architecture is higher than that in (Lin et al., 2011). Spatial information of textures therefore is more effectively exploited for improving CSR by the proposed architecture. In fact, the vector dimension in the proposed architecture is $m = 16 \times 16$. The proposed architecture is able to implement hardware GHA for larger

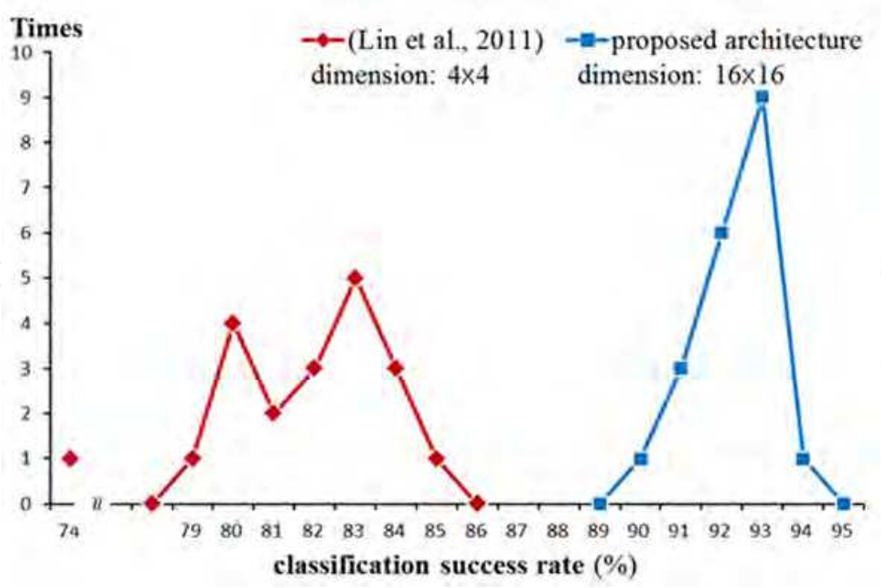



Fig. 18. The distribution of CSR of the proposed architecture for the texture set in Figure 16.

vector dimension because the area cost of the SWU unit in the architecture is independent of vector dimension. By contrast, the area cost of the SWU unit in (Lin et al., 2011) grows with the vector dimension. Therefore, only smaller vector dimension (i.e., $m = 4 \times 4$) can be implemented.

To further elaborate these facts, Tables 1 and 2 show the hardware resource consumption of the proposed architecture and the architecture in (Lin et al., 2011) for vector dimensions $m = 4 \times 4$ and $m = 16 \times 16$. Three different area costs are considered in the table: logic elements (LEs), embedded memory bits, and embedded multipliers. It can be observed from Tables 1 and 2 that given the same $m = 4 \times 4$ and the same $p$, the proposed architecture consumes significantly less hardware resources as compared with the architecture in (Lin et al., 2011). Although the area costs of the proposed architecture increase as $m$ becomes $16 \times 16$, as shown in Table 1, they are only slightly higher than those of (Lin et al., 2011) in Table 2. The proposed architecture therefore is well suited for GHA training with larger vector dimension due to better spatial information exploitation.

| | Proposed GHA with $m = 4 \times 4$ | | | Proposed GHA with $m = 16 \times 16$ | | |
|---|---|---|---|---|---|---|
| $p$ | LEs | Memory Bits | Embedded Multipliers | LEs | Memory Bits | Embedded Multipliers |
| 3 | 3942 | 1152 | 36 | 63073 | 1152 | 569 |
| 4 | 4097 | 1152 | 36 | 65291 | 1152 | 569 |
| 5 | 4394 | 1280 | 36 | 70668 | 1280 | 569 |
| 6 | 4686 | 1280 | 36 | 75258 | 1280 | 569 |
| 7 | 4988 | 1280 | 36 | 79958 | 1280 | 569 |

Table 1. Hardware resource consumption of the proposed GHA architecture for vector dimensions $m = 4 \times 4$ and $m = 16 \times 16$.

| $p$ | GHA in (Lin et al., 2011) with $m = 4 \times 4$ | | |
|---|---|---|---|
| | LEs | Memory Bits | Embedded Multipliers |
| 3 | 22850 | 0 | 204 |
| 4 | 31028 | 0 | 272 |
| 5 | 38261 | 0 | 340 |
| 6 | 45991 | 0 | 408 |
| 7 | 53724 | 0 | 476 |

Table 2. Hardware resource consumption of the GHA architecture (Lin et al., 2011) for vector dimension $m = 4 \times 4$.

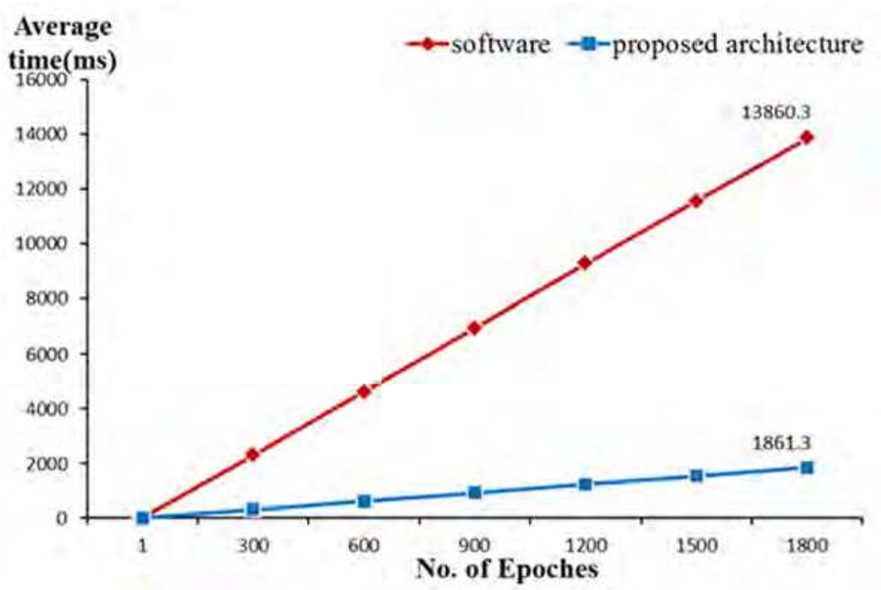



Fig. 19. The CPU time of the NIOS-based SOPC system using the proposed architecture as the hardware accelerator for various numbers of training iterations with $p = 4$.

Figure 19 shows the CPU time of the NIOS-based SOPC system using the proposed architecture as the hardware accelerator for various numbers of training iterations with $p = 4$. The clock rate of NIOS CPU in the system is 50 MHz. The CPU time of the software counterpart also is depicted in the Figure 19 for comparison purpose. The software training is based on the general purpose 2.67-GHz Intel *i7* CPU. It can be clearly observed from Figure 16 that the proposed hardware architecture attains high speedup over its software counterpart. In particular, when the number of training iterations reaches 1800, the CPU time of the proposed SOPC system is 1861.3 ms. By contrast, the CPU time of Intel *i7* is 13860.3 ms. The speedup of the proposed architecture over its software counterpart therefore is 7.45.

## 5. Concluding remarks

Experimental results reveal that the proposed GHA architecture has superior speed performance over its software counterparts. In addition, the architecture is able to attain higher classification success rate for texture classification as compared with other existing

GHA architectures. The architecture also has low area cost for PCA analysis with high vector dimension. The proposed architecture therefore is an effective alternative for on-chip learning applications requiring low area cost, high classification success rate and high speed computation.

## 6. References


Alpaydin, E. (2010). *Introduction to Machine Learning*, second ed., MIT Press, Massachusetts, USA.

Altera Corporation (2010). *NIOS II Processor Reference Handbook ver 10.0*. http://www.altera.com/literature/lit-nio2.jsp

Altera Corporation (2010). *Cyclone III Device Handbook*. http://www.altera.com/products/devices/cyclone3/cy3-index.jsp

Boonkumklao, W., Miyanaga, Y., and Dejhan, K. (2001). Flexible PCA architectura realized on FPGA. *International Symposium on Communications and Information Technologies*, pp. 590 - 593.

Bravo, I., Mazo, M., Lazaro, J.L., Gardel, A., Jimenez, P., and Pizarro, D. (2010). An Intelligent Architecture Based on FPGA Designed to Detect Moving Objects by Using Principal Component Analysis. *Sensors*. 10(10), pp. 9232 - 9251.

Carvajal, G., Valenzuela, W., Figueroa, M. (2007). Subspace-Based Face Recognition in Analog VLSI. In: *Advances in Neural Information Processing Systems*, 20, pp. 225 - 232, MIT Press, Cambridge.

Carvajal, G., Valenzuela, W., and Figueroa M. (2009). Image Recognition in Analog VLSI with On-Chip Learning. *Lecture Notes in Computer Science* (ICANN 2009), Vol.5768, pp. 428 - 438.

Chen, D., and Han, J.-Q. (2009). An FPGA-based face recognition using combined 5/3 DWT with PCA methods. *Journal of Communication and Computer*, Vol. 6, pp.1 - 8.

Chengcui, Z., Xin, C., and Wei-bang, C. (2006). A PCA-Based Vehicle Classification Framework. Proceedings of the 22nd International Conference on Data Engineering Workshops, pp. 17.

Dogaru, R., Dogaru, I., and Glesner, M. (2004). CPCA: a multiplierless neural PCA. Proceedings of IEEE International Joint Conference on Neural Networks, vol.2684, pp. 2689 - 2692.

El-Bakry, H.M. (2006). A New Implementation of PCA for Fast Face Detection. *International Journal of Intelligent Systems and Technologies*,1(2), pp. 145 - 153.

Gunter S., Schraudolph, N.N., and Vishwanathan, S.V.N. (2007). Fast Iterative Kernel Principal Component Analysis, *Journal of Machine Learning Research*, pp. 1893 - 1918.

Haykin, S. (2009). *Neural Networks and Learning Machines*, third Ed., Pearson.

Jolliffe, I.T. (2002). *Principal component Analysis*, second Ed., Springer, New York.

Karhunen, J. and J. Joutsensalo (1995). Generalizations of principal component analysis, optimization problems, and neural networks. *Neural Networks*, 8(4), pp. 549 - 562.

Kim, K., Franz, M.O., and Scholkopf, B. (2005). Iterative kernel principal component analysis for image modeling, *IEEE Trans. Pattern Analysis and Machine Intelligence*, pp. 1351 - 1366.

Lin, S.-J., Hung, Y.-T., and Hwang, W.-J. (2011). Efficient hardware architecture based on generalized Hebbian algorithm for texture classification. *Neurocomputing* 74(17), pp. 3248 - 3256.

Liying, L. and Weiwei, G. (2009). The Face Detection Algorithm Combined Skin Color Segmentation and PCA. International Conference on Information Engineering and Computer Science, pp. 1 - 3.

Oja, E. Simplified neuron model as a principal component analyzer. *Journal of Mathematical Biology* 15(3), pp. 267-273.

Pavan Kumar, A., Kamakoti, V., and Das, S. (2007). System-on-programmable-chip implementation for on-line face recognition. *Pattern Recognition Letters* 28(3), pp. 342 - 349.

Qian Du and J.E.F. (2008). Low-Complexity Principal Component Analysis for Hyperspectral Image Compression. *International Journal of High Performance Computing Applications*.

Sajid, I., Ahmed M.M., and Taj, I. (2008). Design and Implementation of a Face Recognition System Using Fast PCA. IEEE International Symposium on Computer Science and its Applications, pp. 126 - 130.

Sanger,T.D. (1989). Optimal unsupervised learning in a single-layer linear feedforward neural network. *Neural Networks*, vol. 12, pp. 459 - 473.

Sharma, A. and Paliwal, K.K. (2007). Fast principal component analysis using fixed-point algorithm, *Pattern Recognition Letters*, pp. 1151 - 1155.

IntechOpen




**Principal Component Analysis**
Edited by Dr. Parinya Sanguansat

ISBN 978-953-51-0195-6
Hard cover, 300 pages
**Publisher** InTech
**Published online** 02, March, 2012
**Published in print edition** March, 2012

This book is aimed at raising awareness of researchers, scientists and engineers on the benefits of Principal Component Analysis (PCA) in data analysis. In this book, the reader will find the applications of PCA in fields such as image processing, biometric, face recognition and speech processing. It also includes the core concepts and the state-of-the-art methods in data analysis and feature extraction.

**How to reference**
In order to correctly reference this scholarly work, feel free to copy and paste the following:

Shiow-Jyu Lin, Kun-Hung Lin andWen-Jyi Hwang (2012). FPGA Implementation for GHA-Based Texture Classification, Principal Component Analysis, Dr. Parinya Sanguansat (Ed.), ISBN: 978-953-51-0195-6, InTech, Available from: http://www.intechopen.com/books/principal-component-analysis/fpga-implementation-for-gha-based-texture-classification

INTECH
open science | open minds

**InTech Europe**
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

**InTech China**
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

© 2012 The Author(s). Licensee IntechOpen. This is an open access article distributed under the terms of the Creative Commons Attribution 3.0 License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

IntechOpen

IntechOpen