VNU Journal of Science: Comp. Science & Com. Eng., Vol. 31, No. 3 (2020) 1-9

An Implementation of PCA and ANN-based Face Recognition System on Coarse-grained Reconfigurable Computing Platform

Hung K. Nguyen*, Xuan-Tu Tran

University of Engineering and Technology, Vietnam National University, Hanoi, Vietnam

Abstract

In this paper, a PCA and ANN-based face recognition system is proposed and implemented on a Coarse Grain Reconfigurable Computing (CGRC) platform. Our work is quite distinguished from previous ones in two aspects. First, a new hardware-software co-design method is proposed, and the whole face recognition system is divided into several parallel tasks implemented on both the Coarse-Grained Reconfigurable Architecture (CGRA) and the General-Purpose Processor (GPP). Second, we analyzed the source code of the ANN algorithm and proposed the optimization solution to explore its multi-level parallelism to improve the performance of the application on the CGRC platform. The computation tasks of ANN are dynamically mapped onto CGRA only when needed, and it's quite different from traditional Field Programmable Gate Array (FPGA) methods in which all the tasks are implemented statically. Implementation results show that our system works correctly in face recognition with a correct recognition rate of approximately 90.5%. To the best of our knowledge, this work is the first implementation of PCA and ANN-based face recognition system on a dynamically CGRC platform presented in the literature.

Keywords: Coarse-grained Reconfigurable Architecture; Principal Components Analysis (PCA); Face Recognition; Artificial Neural Network (ANN); Reconfigurable Computing platform.

1. Introduction

Face recognition is one of the most common biometric recognition techniques that attract huge attention of many researchers in the field of computer vision since the 1980s. Today, face recognition has proven its important role and is widely used in many areas of life. Some important applications of face recognition are automatic criminal record checking, integration with surveillance cameras or ATM systems to increase security, online payment, tracking, and prediction of strange diseases in medicine.

The face recognition system gets an image, a series of photos, or a video as input and then processes them to identify whether a person is known or not. The system includes two phases which are the feature extraction and the classification as shown in

Figure 1.

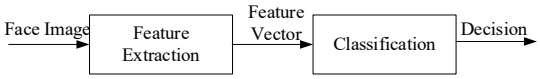

Figure 1. Processes in face recognition.

The problem we have to deal with when implementing a face recognition system is that the data set has a very large number of dimensionality resulting in a large amount of computation which takes a lot of processing time. Therefore, a significant improvement would be achieved if we could reduce the

*Corresponding author. E-mail.: kiemhung@vnu.edu.vn

dimensionality of data by mapping them to another space with a smaller number of dimensionality [16]. Especially, dimensionality reduction is indispensable for real-time face recognition system while processing high-resolution images. Feature extraction is a process to reduce the dimensionality of a set of raw data to more manageable groups for processing. Feature extraction selects and/or combines variables into features, effectively reducing the amount of data that must be processed, while still accurately and completely describing the original data set. The principal components analysis (PCA) [15] method is the appearance-based technique used widely for reducing dimensionality and achieved a great performance in face recognition.

Classification is a process in which ideas and objects are recognized, differentiated, and understood based on the extracted features by an appropriate classifier. The artificial neural networks (ANNs) are one of the most successful classification systems that can be trained to perform complex functions in various face recognition systems. State-of-the-art ANNs are demonstrating high performance and flexibility in a wide range of applications including video surveillance, face recognition, and mobile robot vision, etc.

Face recognition using PCA in combination with neural networks is a method to achieve high recognition efficiency by promoting the advantages of PCA and neural networks [11]. In this paper, a face recognition system based on the combination of PCA and neural network is implemented on the coarse-grained reconfigurable computing platform. The proposed system offers an improvement in the recognition performance over the conventional PCA face recognition system. The system operates stably and has high adaptability when the data input has a large variation. The system has been implemented and validated on the coarse-grained reconfigurable computing platform built around the CGRA called MUSRA that was proposed in our previous work [10].

The rest of this paper is organized as follows. Section 2 reviews some related works. In Section 3, the proposal of the MUSRA-based coarse-grained reconfigurable computing (CGRC) platform is introduced. Section 4 presents the implementation of the face recognition system on the CGRC platform. Evaluation of the proposed system in comparison with the related works are given in Section 5. Finally, some conclusions are drawn in Section 6.

2. Related works

2.1 PCA for Face Recognition

Principal Component Analysis (PCA) is a standard method for dimensionality reduction and feature extraction. It uses a mathematical method called orthogonal transformation to transform a large number of correlated variables into a smaller set of uncorrelated variables so that the newly generated variables are linear combinations of old variables [15].

In this paper, the PCA method is used to reduce the number of dimensionality of the image, helping to reduce the computation complexity of the training or identification process in the neural network later. The steps to perform PCA are as follows:

Step 1: Let's establish the training set of face images be $S = \{\Gamma_1, \Gamma_2, \ldots, \Gamma_M\}$. Each image in 2-dimension with size W×H is converted into a 1-dimension vector of W×H elements.

Step 2: Calculate the average image Ψ:

$$\Psi = \frac{1}{M}\sum_{i=1}^{M} \Gamma_i \quad (1)$$

Step 3: Calculate the deviation of input images from average image:

$$\Phi_i = \Gamma_i - \Psi \quad (2)$$

Step 4: Calculate the covariance matrix C:

$$C = \frac{1}{M}\sum_{i=1}^{M} \Phi_i \Phi_i^T = A.A^T \quad (3)$$

where $A = [\Phi_1, \Phi_2, \ldots, \Phi_M]$

Step 5: Because matrix C is too large in size (N×N), therefore, to find the eigenvector u_i of C we find the eigenvector and the eigenvalue of the matrix L:

$$L = A^T A \text{ với } L_{m,n} = \Phi_m^T \Phi_n \quad (4)$$

The size of the matrix L is M×M << N×N, so calculating eigenvector is faster.

Step 6: Let's set v_i as the eigenvector of L. The eigenvector of C is:

$$u_i = \sum_{i=1}^{M} v_{ik} \Phi_k \text{ , i } = \overline{1, M} \quad (5)$$

Because vectors u_i are the eigenvectors of the covariance matrix corresponding to the original face images, so they are referred as eigenfaces.

Step 7: After finding the eigenfaces, the images in the database will be projected onto these eigenfaces space to create the feature vectors. These vectors are much smaller than the image size but still carries the most key information contained in the image.

There is much research ([13][14][15][16][18][19][20]) on using PCA in scientific disciplines, some works have published the implementation of PCA for face recognition ([13][14]).

2.2. Artificial Neural Networks

Artificial neural networks take their inspiration from a human brain's nervous system. Figure 2 depicts a typical neural network with a single neuron explained separately. Similar to human nervous system, each neuron in the ANN collects all the inputs and performs an operation on them. Lastly, it transmits the output to all other neurons of the next layer to which it is connected. A neural network is composed of three layer types:

- **Input Layer:** takes input values and feeds them to the neurons in the hidden layers.
- **Hidden Layers:** are the intermediate layers between input and output which help the neural network learn the complicated relationships involved in data.
- **Output Layer:** presents the final outputs of the network to the user.

Computation at each neuron in hidden layers and output layer is modeled by the expression:

$$y_i = f(\sum_{j=1}^{R} W_{ij} \times x_j + b_i) \quad (6)$$

where W_{ij}, b_i, x_j and y_i are the weights, bias, input activations, and output activations, respectively, and $f(\cdot)$ is a nonlinear activation function such as Sigmoid [5], Hyperbolic Tangent [5], Rectified Linear Unit (RELU) [6], etc.

Just like in human brain, an ANN needs to be trained to perform its given tasks. This training involves determining the value of the weights (and bias) in the network. After that, the ANN can perform its task by computing the output of the network by using the weights determined during the training process. This process is referred to as inference. Training and inference must be considered during the development of hardware platform for ANN. Training generally requires high-computing performance, high-precision arithmetic, and programmability to support different deep learning models. In fact, training is usually performed offline on workstations or servers. Some research efforts have been looking for incremental training solutions [7] and a reduction in precision training [8] to decrease the computation complexity.

Many ANN frameworks are implemented on GPU (Graphic Processing Unit) platforms such as Caffe [1], Torch [2], and Chainer [3]. These fast and friendly frameworks are developed for easily modifying the structures of neural networks. However, from the performance point of view, dedicated architectures for ANNs have a higher throughput as well as higher energy efficiency. In recent decades, interest in the hardware implementation of artificial neural networks (ANN) by using FPGA and ASIC has grown. This is mainly due to the rapid development of semiconductor technology that is used for implementing digital ANN. Previous FPGA/ASIC architectures already achieved a throughput of several hundreds of Gop/s. These architectures are easily scalable to get a higher performance by leveraging parallelism. However, one problem that most of these designs are still faced with is: ASIC solution are usually suffering from a lack of the flexibility to be reconfigured for the various parameters of ANN. With deep ANN comprising many layers with different characteristics, it is impossible to use heterogeneous architectures for the different layers. In this paper, we propose an implementation of ANN on the coarse-grained reconfigurable architecture.

2.3 Reconfigurable Hardware

The reconfigurable hardware is generally classified into the Field Programmable Gate Array (FPGA) and coarse-grained dynamically reconfigurable architecture (CGRA). A typical

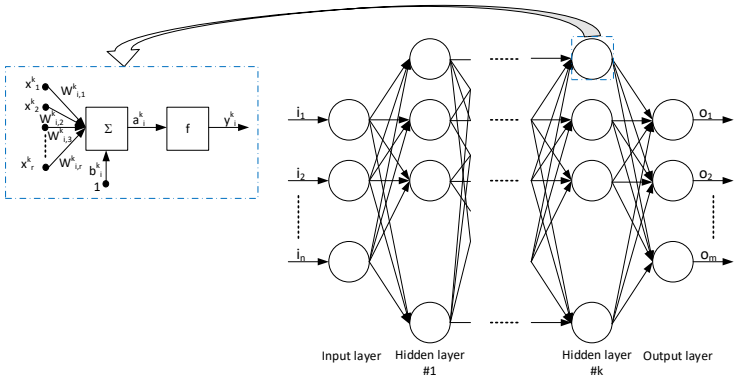

Figure 2. An artificial neuron and an ANN model.

example of the FPGA-based reconfigurable SoC is Xilinx Zynq-7000 devices [21]. Generally, FPGAs support the fine-grained reconfigurable fabric that can operate and be configured at bit level. FPGAs are extremely flexible due to their higher reconfigurable capability. However, the FPGAs consume more power and have more delay and area overhead due to greater quantity of routing required per configuration [22]. This limits the capability to apply FPGA to embedded applications. To overcome the limitation of the FPGA-like fine-grained reconfigurable devices, coarse-grained reconfigurable architectures (CGRAs) focus on data processing and configuration at bit-group with complex functional blocks (e.g. Arithmetic Logic unit (ALU), multiplier, etc.). These architectures are often designed for a specific domain of applications. CGRAs achieve a good trade-off between performance, flexibility, and power consumption.

Many CGRAs have been proposed with the unique features that is dedicated to a specific domain of applications. Typical two of them are REMUS[23] and ADRES[24]. ADRES (Architecture for Dynamically Reconfigurable Embedded System) is a reconfigurable system template, which tightly couples a VLIW (Very Long Instruction Word) processor and a coarse-grained reconfigurable matrix into a single architecture. Here, coarse-grained reconfigurable matrix plays a role of a co-processor in the VLIW processor. Coupling CGRA directly with the processor increases the performance at the expense of decrease in flexibility because the CGRA architecture has to be compatible with the given processor architecture. By contrast, in the REMUS-II (REconfigurable MUltimedia System version II) architecture - a coarse-grained dynamically reconfigurable heterogeneous computing SoC for multimedia and communication baseband processing, the CGRA is implemented as an IP core that is attached to the system bus of the processor. The REMUS-II consists of one or two coarse-grained dynamically reconfigurable processing units (RPUs) and an array of RISC processors (μPU) coupled with a host ARM processor via the AHB bus. Designing the CGRA as an IP core in the REMUS makes it easy to reuse design in the various systems with no dependence on any particular processor architecture.

In [10], we developed and modeled a coarse-grained dynamically reconfigurable architecture, called MUSRA (Multimedia Specific Reconfigurable Architecture). The MUSRA is a high-performance, flexible platform for a domain of applications in multimedia processing. In contrast with FPGAs, the MUSRA aims at reconfiguring and manipulating on the data at word-level. The MUSRA was proposed to exploit high data-level parallelism (DLP), instruction-level parallelism (ILP) and TLP (Task Level Parallelism) of the computation-intensive loops of an application. The MUSRA also supports the capability of dynamic reconfiguration by enabling the hardware fabrics to be reconfigured into different functions even if the system is working.

3. Proposed Architecture of CGRC Platform

3.1 Coarse-Grained Reconfigurable Computing Platform

In this paper, we developed a high-performance Coarse-Grained Reconfigurable Computing Platform (CGRC) for experimentally evaluating and validating the applications of multimedia processing. The platform's hardware is a system-on-chip based on the MUSRA (Multimedia Specified coarse-grained Reconfigurable Architecture) [10], the ARM processor, and the other IP cores from the Xilinx's library as shown in Figure 3. The CGRC platform was synthesized and implemented on the Xilinx ZCU106 Evaluation Kit [25]. The ARM processor functions as the central processing unit (CPU) that takes charge of managing and scheduling all activities of the system. The external memory is used for communicating data between tasks on the CPU and tasks on the MUSRA. Cooperation between MUSRA, CPU, and DMACs (Direct Memory Access Controllers) are synchronized by the interrupt mechanism. When the MUSRA finishes the assigned task, it generates an interrupt via IRQC (Interrupt Request Controller) unit to signal the CPU and returns bus control to the CPU. In order to run on the platform, the C-program of the application is compiled and loaded into the Instruction Memory of the platform. Meanwhile, the data is copied into the Data Memory.

Execution and data-flow of the MUSRA are reconfigured dynamically under controlling of the CPU. After resetting, the operation of the system is briefly described as follows:

① **Context Memory Initialization**: CPU writes the necessary control parameters and then grant bus control to CDMAC in Context Memory. CDMAC will copy a context from the instruction memory to context memory. At the same time, CPU executes another function.

② **Context Parser Initialization**: CPU writes the configuration words to the context parser.

③ **RCA Configuration** and **Data Memory Initialization**: After configured, parser reads one proper context from the context memory, decode it and configure RCA. Concurrently, CPU initializes DDMAC that will copy data from the external data memory to the internal data memory. DDMAC is also used for writing the result back to the external data memory.

④ **RCA Execution**: RCA performs a certain task right after it has been configured.

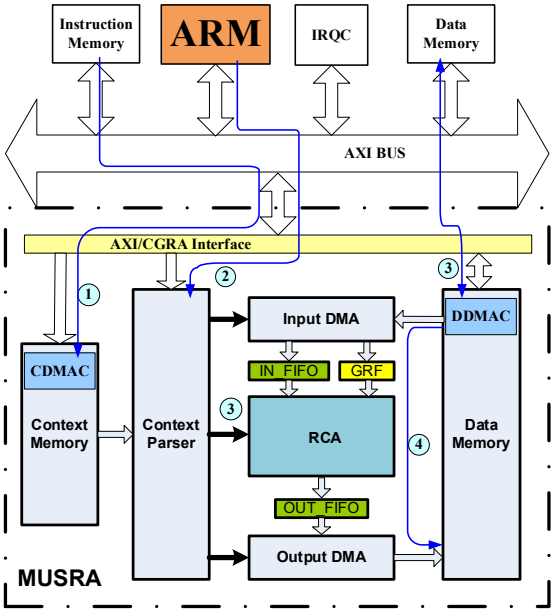

Figure 3. Coarse-Grained Reconfigurable Computing Platform (CGRC).

3.2 MUSRA architecture

The MUSRA [10] is composed of a Reconfigurable Computing Array (RCAs), Input/Output FIFOs, Global Register File (GRF), Data/Context memory subsystems, and DMA (Direct Memory Access) controllers, etc. Data/Context memory subsystems consist of storage blocks and DMA controllers (i.e. CDMAC and DDMAC). The RCA is an array of 8×8 RCs (Reconfigurable Cells) that can be configured partially to implement computation-intensive tasks. The input and output FIFOs are the I/O buffers between the data memory and the RCA. Each RC can get the input data from the input FIFO or/and GRF, and store the results back to the output FIFO. These FIFOs are all 512-bit in width and 8-row in depth, and can

load/store sixty-four bytes or thirty-two 16-bit words per cycle. Especially, the input FIFO can broadcast data to every RC that has been configured to receive the data from the input FIFO. This mechanism aims at exploiting the reusable data between several iterations. The interconnection between two neighboring rows of RCs is implemented by a crossbar switch. Through the crossbar switch, an RC can get results that come from an arbitrary RC in the above row of it. The Parser decodes the configuration information that has been read from the Context Memory, and then generates the control signals that ensure the execution of RCA accurately and automatically.

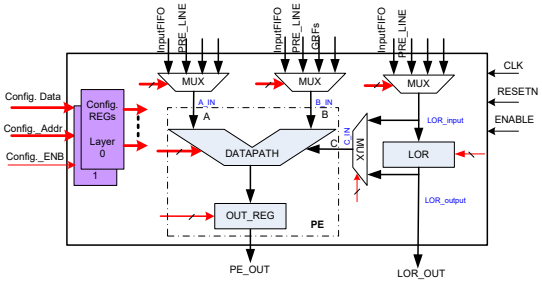

Figure 4. RC architecture.

RC (Figure 4) is the basic processing unit of RCA. Each RC includes a data-path that can execute signed/unsigned fixed-point 8/16-bit operations with two/three source operands, such as arithmetic and logical operations, multiplier, and multimedia application-specific operations (e.g. barrel shift, shift and round, absolute differences, etc.). Each RC also includes a local register called LOR. This register can be used either to adjust operating cycles of the pipeline or to store coefficients when a loop is mapped onto the RCA. A set of configuration registers, which stores configuration information for the RC, is called a layer. Each RC contains two layers that can operate in the ping-pong fashion to reduce the configuration time.

The configuration information for the MUSRA is organized into the packets called context. The context specifies a particular operation of the RCA core (i.e. the operation of each RC, the interconnection between RCs, the input source, output location, etc.) as well as the control parameters that control the operation of the RCA core. The total length of a context is 128 32-bit words. An application is composed of one or more contexts that are stored into the context memory of the MUSRA.

The MUSRA architecture is basically the such-loop-oriented one. By mapping the body of the kernel loop onto the RCA, the RCA just needs configuring one time for executing multiple times, therefore it can improve the efficiency of the application execution. Executing model of the RCA is the pipelined multi-instruction-multi-data (MIMD) model. In this model, each RC can be configured separately to a certain operation, and each row of RCs corresponds to a stage of a pipeline. Multiple iterations of a loop are possible to execute simultaneously in the pipeline.

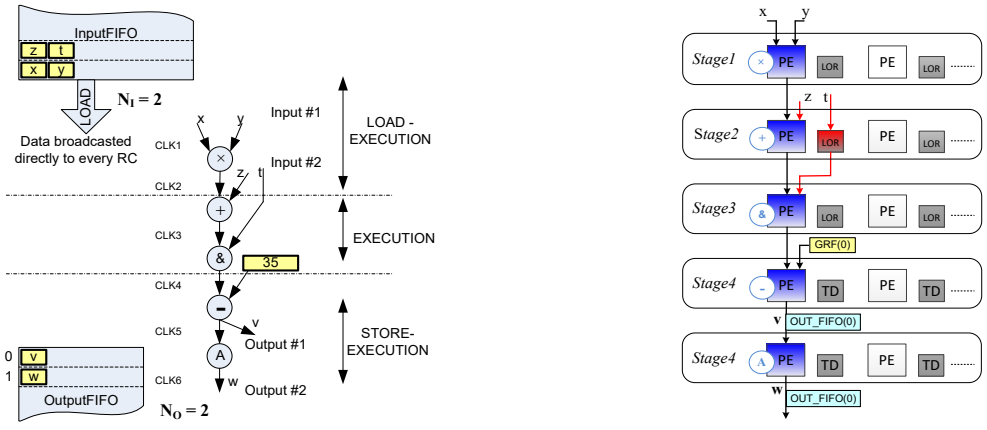

Figure 5. (a) DFG representation of a simple loop body, and (b) its map onto RCA.

For purpose of mapping, a kernel loop is first analyzed and loop transformed (e.g. loop unrolling, loop pipelining, loop blocking, etc.) in order to expose inherent parallelism and data locality that are then exploited to maximize the computation performance on the target architecture. Next, the body of the loop is represented by data-flow graphs (DFGs) as shown in Figure 5. Thereafter, DFGs are mapped onto RCA by generating configuration information, which relates to binding nodes to the RCs and edges to the interconnections. Finally, these DFGs are scheduled in order to execute automatically on RCA by generating the corresponding control parameters for the CGRA's controller. Once configured for a certain loop, RCA operates as the hardware dedicated for this loop. When all iterations of loop have completed, this loop is removed from the RCA, and the other loops are mapped onto the RCA.

4. Implementation of Face Recognition System

4.1. Face recognition system

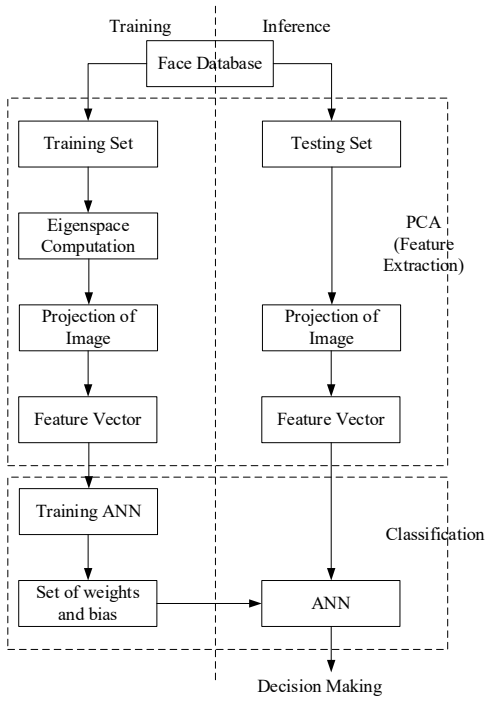

Figure 6. Face recognition based on the combination between ANN and PCA.

The face recognition system is based on the combination of PCA and an artificial neural network called the PCA-ANN system. The PCA-ANN face recognition system is divided into 3 processes: feature extraction, training, and recognition as shown in Figure 6. In the feature extraction process, an eigenfaces space is established from the training images using the PCA feature extraction method. The ANN requires the training process where the weights connecting the neurons in consecutive layers are calculated based on the training images and target classes. After generating the eigenvectors using PCA methods, the projection of face images in the training set is calculated and then used to train the neural network on how to classify images for each person. In the recognition process, each input face image in the testing set is also projected to the same eigenfaces space and classified by the trained ANN.

4.2. Hardware/Software Partition

Instead of implementing the system entirely by hardware or software, this paper proposes a system-level model for the realization of the PCA-ANN face recognition system, including hardware and software tasks, as shown in Figure 7.

In PCA feature extraction, calculating eigenvalues and eigenvectors for eigenfaces space requires very complicated algebraic methods like QR or Jacobi [12]. The hardware architecture for implementing a PCA algorithm is often very complex. Because of the complexity of the PCA algorithm, in the scope of this paper it will be implemented as software running on the CPU.

In ANN-based classification, two aspects must be considered, including training and inference. Training still requires high-performance computing, high-precision arithmetic, and programmability to support different deep learning models. The training process is time-consuming and involves a lot of power consumption. Therefore, it is usually done offline on the server's GPU. In particular, the training is performed in software using MATLAB running on the server. Matlab program includes one function to calculate the

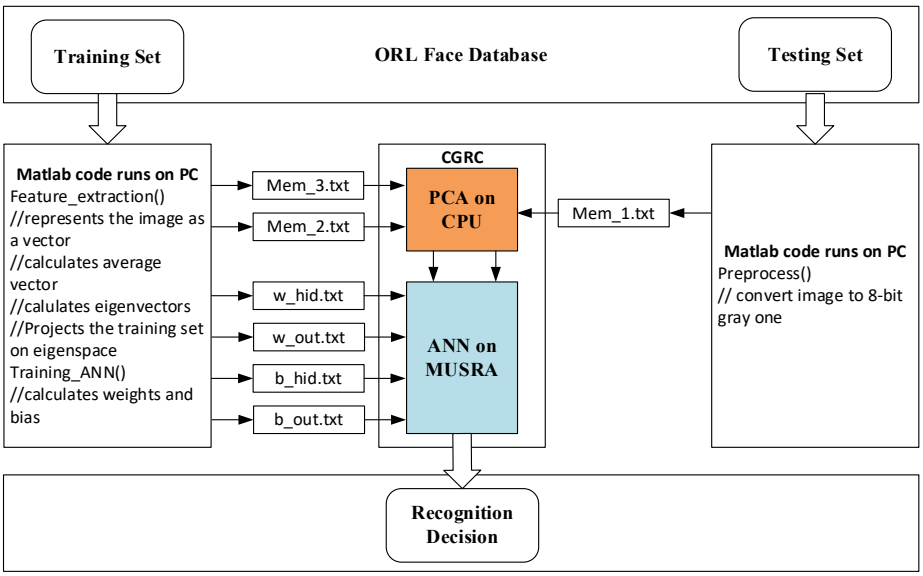

Figure 7. Hardware/Software partition.

eigenvectors using the built-in functions and another for training the neural network. The results are the average vector, the eigenvectors, the weights and biases of the neural network after being trained. These parameters are then saved in text files (.txt) and will be written to the memory on the CGRC platform while the system is operating.

On the other side, the inference is performed by both software and hardware on the high-performance CGRC platform. Here, PCA feature extraction is performed by the CGRC platform's CPU, and ANN is mapped onto the CGRC platform's MUSRA. The face image, which is considered for recognition, is firstly pre-processed by a MATLAB program on the server, then passed through the PCA module to extract the features, and finally sent to the ANN module for making recognition decision.

4.3 Mapping ANN onto MUSRA

Let's examine a generic ANN that has L layers with one input layer, one output layer, and L-2 hidden layers. At the layer k^{th}, the input vector X^k is forwardly transferred through the neurons to generate an output vector Y^k that then becomes the input vector X^{k+1} for the next layer $(k+1)^{th}$. The pseudo-code in Algorithm 1 describes ANN computation.

Algorithm 1. ANN Computation

1	X^1 = input
2	For k in 1 to L – 1 loop
3	$A^k = X^k W^k$
4	$Y^k = f(A^k)$
5	$X^{k+1} = Y_k$
6	End For
7	Output = X^L

Where, input = $(i_1, i_2, \ldots i_n)$ is the input vector, and output = $(o_1, o_2, \ldots o_m)$ is the output vector.

Let's set N_k is the number of neurons in k^{th} layer, where k = 1, 2, …, L-1. Since the output of each layer forms the input of the next layer, therefore, the input vector of the k^{th} layer is $X^k = [x_0^k, x_1^k, \ldots, x_{N_{k-1}-1}^k]$ and its dimension is $1 \times N_{k-1}$. The output vector of the k^{th} layer is $Y^k = [y_0^k, y_1^k, \ldots, y_{N_k-1}^k]$, which has $1 \times N_k$ elements.

W^k is the weight matrix at the layer k^{th}.

$$W^k = \begin{pmatrix} w_{0,0}^k & \cdots & w_{N_k-1,0}^k \\ \vdots & \ddots & \vdots \\ w_{0,N_{k-1}-1}^k & \cdots & w_{N_k-1,N_{k-1}-1}^k \end{pmatrix}$$

Algorithm 1 can be expanded to some loops, as shown in Algorithm 2.

Algorithm 2. ANN Computation

1	X^1 = input
2	For k in 1 to L – 1 loop
3	// loop_k runs through all layers
4	For i = 0 to N_k – 1 loop
5	//loop_i runs through all neurons
6	$A_i^k = b_i$
7	For j = 0 to N_{k-1} – 1 loop //loop_j
8	$A_i^k = A_i^k + x_j^k W_{i,j}^k$
9	End for j
10	$Y_i^k = f(A_i^k)$ //activation function
11	$X_i^{k+1} = Y_i^k$
12	End for i
13	End For k
14	Output = X^L

The loop *loop_j* (line 7 to line 9 in Algorithm 2) is unrolled and implemented by the adder tree with the output that is the accumulation of the products of the inputs and weights as shown in Figure 8. The structure of adder-tree for different layers varies in the number of inputs, weights, and bias.

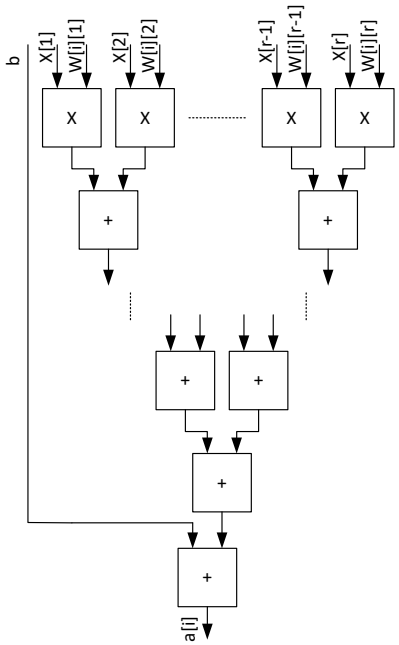

Figure 8. DFG of the loop loop_j.

The sum of the weighted inputs will be fed to the activation function to calculate the output value at each neuron (line 10 in Algorithm 2). The activation function used here is the Sigmoid function, as shown in expression (7):

$$f(x) = \frac{1}{1+e^{-x}} \quad (7)$$

There some different methods for the implementation of the sigmoid activation function, including Look-up table (LUT), Taylor transformation, piecewise linear approximation [9]. The method of Taylor transformation requires many multiplications. The LUT method uses a large memory to store the table of possible values of the target function. As a result, both of these methods are not realizable for the implementation on CGRA. Therefore, in this paper, we adopted the method of a piecewise linear approximation to perform the sigmoid activation function. The chosen approximation method has a great influence on the accuracy and performance of the neural network.

For x in the range [0, +∞), the sigmoid function is approximated by expression:

$$\bar{f}(x) = \begin{cases} 1, & x \geq 4 \\ -0.03125 * x^2 + 0.25 * x + 0.5, & 0 \leq x < 4 \end{cases} \quad (8)$$

For negative values of x, the sigmoid function can be calculated using the expression:

$$\bar{f}(x) = 1 - \bar{f}(-x) \quad (9)$$

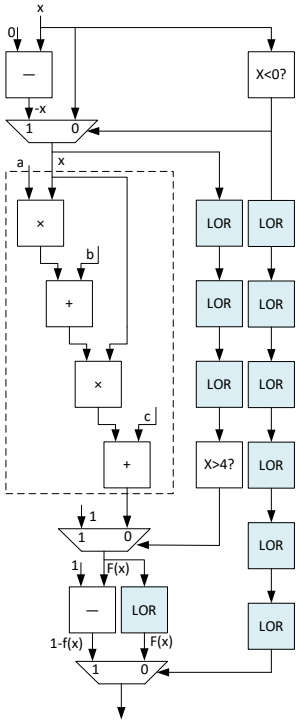

Figure 9. DFG of activation function.

Figure 9 shows a DFG for implementing the sigmoid activation function, where, a = -0.03125, b = 0.25, and c = 0.5. Because of the reconfigurability of MUSRA, the activation function can be easily configured to different ones for the hidden layers and the output layer.

5. Experiment and Evaluation

This section presents the validation of the PCA-ANN face recognition system on the CGRC platform. The performance of our implementation is compared with the other works.

5.1 Validation script

The face database used for the experiment is the image set from the AT&T ORL (Olivetti Research Laboratory). This database consists of 400 images of 40 people, each with 10 different images. Each image has dimensions of 80 × 110 pixels. The face database are divided into training set and testing set. After PCA feature extraction, each face image is represented by a feature vector of size 1×30.

The neural network consists of three layers and is configured as follows:

- *Input Layer:* includes 30 neurons corresponding to the number of elements of the feature vector of size 1×30.
- *Hidden Layer:* consists of 120 neurons, each has 30 inputs, 30 weights and one bias.
- *Output Layer:* is made of three neurons corresponding to three outputs. Each of neuron has 120 inputs, 120 weights and one bias.
- *Number of training times*: 10000 times.
- *Learning coefficient*: 0.01
- *Permissible error*: 10^{-5}

5.2 Experimental Results

1) Sigmoid function and its approximation

Figure 10 shows the chart of Sigmoid function and its approximation by (8) and (9) in the range (-8, 8). Where, the orange line depicts the sigmoid function, and the blue line depicts the sigmoid function's approximation. Experiment estimation shows that the average error and the maximum error of approximation of the sigmoid function are $\varepsilon_{average} = 0.00774$ và $\varepsilon_{maximum} = 0.02163$.

2) Functional verification of face recognition system

To functionally verify the face recognition system, the training set and testing set are extracted from the database as follows. First, choose three image sets of three people in the ORL database to build the training set. Next, take three images of these three people, each person one image, and flip that image to create the testing set. The purpose of flipping the image is to make it different from the image in the training set. The training set and testing set is shown in Figure 11 and Figure 12, respectively.

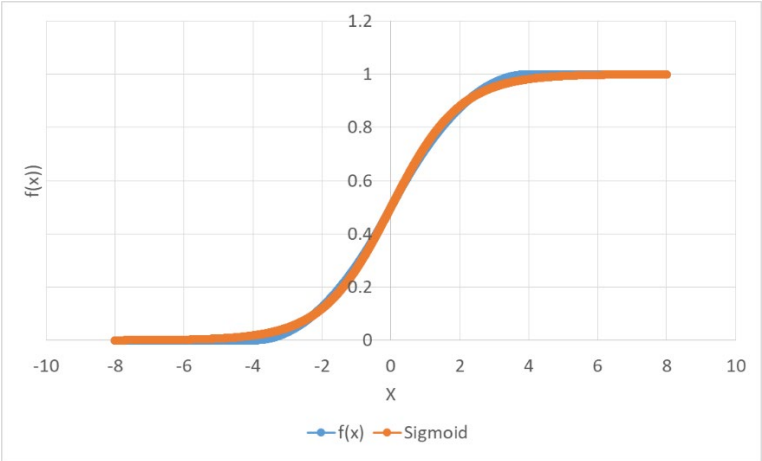

Figure 10. Chart of Sigmoid function and its approximation.

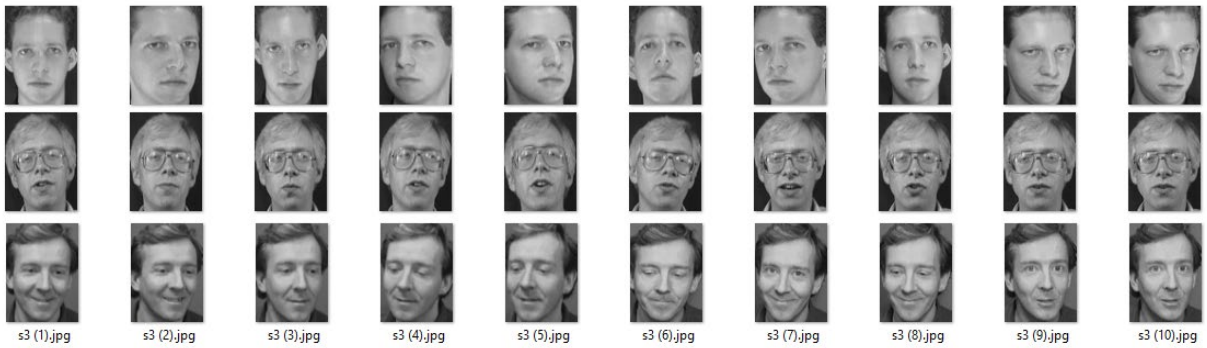

Figure 11. Training set including images of three first persons in the ORL database.

(a) images in the training set (b) image not in the training set

Figure 12.Testing set.

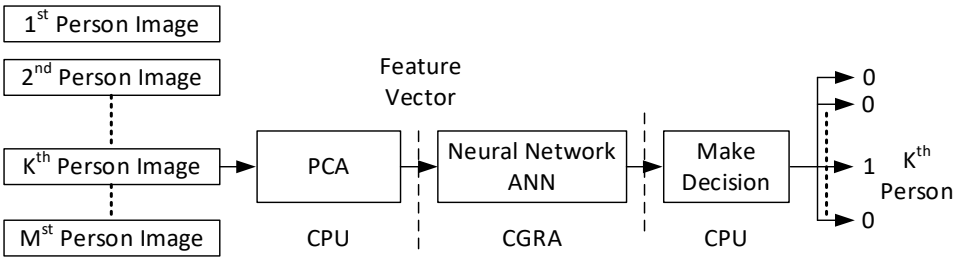

Figure 13. Inference Processing.

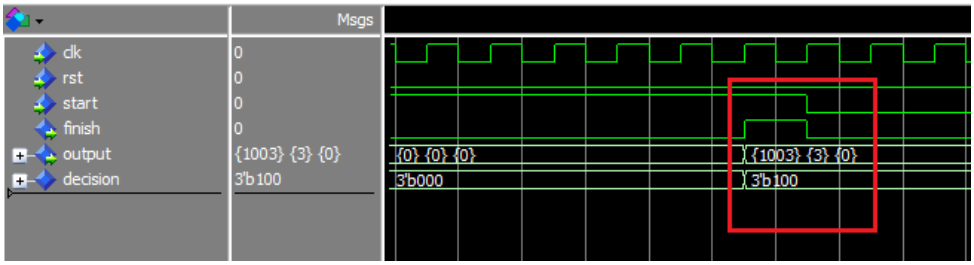

Figure 14. Simulation results for the first image in the training set.

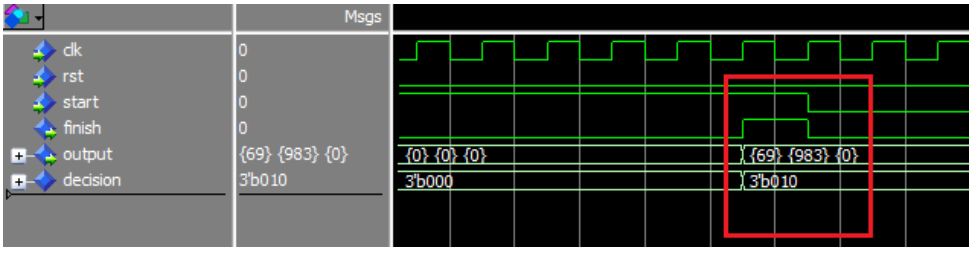

Figure 15. Simulation results for the second image in the training set.

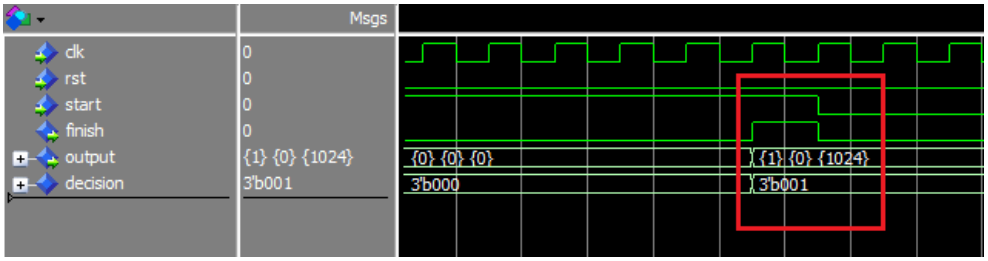

Figure 16. Simulation results for the third image in the training set.

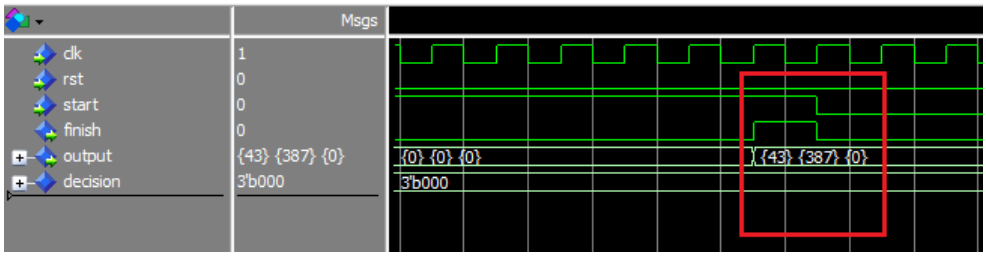

Figure 17. Simulation results for the image not in the training set.

Testing images are represented in feature vectors before feeding to ANN and the ANN's outputs are compared with the given threshold by CPU to make a recognition decision as shown in Figure 13. The simulation results of ANN are shown in Figure 14, Figure 15, and Figure 16.

Similarly, take an image that is not included in the training set as input to face recognition system. Simulation results are shown in Figure 17.

The computation result of ANN is represented by 16-bit fixed-point numbers where 6 bits represent the integer part, and another 10 bits represent the fractional part. This result is displayed on the port output in the waveform window while simulating. For easy viewing, this result is left-shifted 10 bits (i.e. multiplied by 2^{10}) to convert to 16-bit integers, as shown in the figures Figure 14 - Figure 17 above. The value at the output is compared to the threshold by the CPU to determine which face is detected. The thresholds chosen for comparison are 0.9 and 0.1 (values greater than 0.9 are determined to be 1, less than 0.1 is decided as 0). To compare with the simulation results, these threshold values are also left-shifted 10 bits to become values 921 and 102, respectively. The rules for making identification decisions are shown in Table 1.

Based on the rules in Table 1 and the simulation results in figures Figure 14 - Figure 17, we get the results of identification, as shown in Table 2. This result proves that the system is functional correctness as expected.

Table 1. The Rule for making decision

OUPUT	Decision	Identification results
output(2) > 921 output(1) < 102 output(0) < 102	[1; 0; 0]	First Person
output(2) < 102 output(1) > 921 output(0) < 102	[0; 1; 0]	Second Person
output(2) < 102 output(1) < 102 output(0) > 921	[0; 0; 1]	Third Person
Others	[0; 0; 0]	Stranger

3) Performance Analysis of face recognition system

To evaluate the performance of the system, we use the training and testing set which are built as follows. We use 30 images of the first three people in the ORL database as a training set. The testing set has 400 images, including 30 flipped images of three selected people (to make the difference from the training images) and 370 images of the remaining 37 people in the ORL database.

Table 2. Identification results

	First Image	Second image	Third Image	Fourth image
Displayed result (16-bit)	[1003; 3; 0]	[69; 983; 0]	[1; 0; 1024]	[43; 387; 0]
*Real result	[0.9794; 0.0029; 0]	[0.0673; 0.9599; 0]	[0.0009; 0; 1]	[0.0419; 0.3779; 0]
Decision	[1; 0; 0]	[0; 1; 0]	[0; 0; 1]	[0; 0; 0]
Identification results	First person	Seconf person	Third person	Stranger

**Real result = Displayed result/*$\mathbf{2^{10}}$.

Table 3. Performance Evaluation.

	Number of images	Correct Identification	Wrong Identification
First person	10	10	0
Second person	10	8	2
Third persom	10	9	1
Stranger	370	332	38
Total	400	359	41
Ratio		90.5%	9.5%

The validation results are recorded in Table 3. Here, “strangers” are those who are not in the training set, and "acquaintances” are those who are in the training set. The result is considered as “Correct Identification” (a) when inputting the image of an “acquaintance”, the system correctly identifies which of the three selected persons the image is; or (b) when inputting an image of a stranger, the system returns "stranger". On the contrary, the result is considered as "Wrong identification" when the system mistakenly recognizes "stranger" as "acquaintance", "acquaintance" as "stranger", or confuses an acquaintance with each other.

Table 4. Performance Comparison

PCA-ANN			PCA	
Our	[13]	[14]	[13]	[14]
90.5%	85%	88%	78%	86%

Evaluation results show that the system has quite high recognition efficiency, achieving correct recognition rate of 90.5%. The experimental results are compared with the results of other works in Table 4.

Compared to the results of other works using the same method, our system achieved 5.5% and 2. 5% higher correct recognition rate than [13] and [14], respectively. Compared with using only PCA method for identification, the recognition results when using the combination of two methods are 12.5% and 4.5% more accurate than [13] and [14], respectively.

Table 5 shows the change in recognition efficiency when changing the number of hidden layer neurons. When the number of hidden layer neurons increased from 120 to 150, the difference in performance was very small. However, when the number of hidden layer neurons is reduced, especially when it is reduced to 30, the performance is greatly reduced, showing that the number of hidden layer neurons

Table 5. Recognition performance vs. number of hidden layer neurons.

Number of neurons	150	120	90	60	30
Performance	89.75%	90.5%	88.75%	85.25%	80.5%

has a large impact on the identification performance of the neural network. This can be explained as follows. When the number of hidden layer neurons is too small, the network cannot learn deeply enough, resulting in poor recognition performance. Conversely, increasing the number of hidden layer neurons will make the network model more complicated and the possibility of "over-matching" becomes higher. "Over-matching" occurs when the trained network matches with the training samples so much, therefore it answers exactly what has been learned, and does not care what is not learned.

6. Conclusion

In this paper, we presented our work on the proposal, implementation and evaluation of PCA and ANN-based face recognition system. The feature vectors obtained through the PCA method are used as the input for training and testing the ANN. The combination of PCA method and neural network is to improve the system's identification efficiency. The face recognition system has been hardware/software co-designed and implemented on a coarse-grained reconfigurable computing platform. We analyzed the source code of the ANN algorithm and proposed the optimization solution to explore its multi-level parallelism in order to improve the performance of the application on the CGRC platform. Our implementation has been simulated and validated by the CGRC platform of the MUSRA on the Xilinx ZCU106 Evaluation Kit. The verification process has confirmed that the system works correctly in face recognition. The correct recognition rate is approximately 90.5%. The proposed system gets an improvement on the recognition rates over classical PCA face recognition system. In addition, the recognition performance of our system is higher than the PCA-ANN system proposed by other works. It is also easy to reconfigure the MUSRA to support different ANN configuration (for example, number of layer, number of neurons per layer, activation function, etc.).

Our method on CGRC platform could be extended to the algorithm of the other applications. In the future work, some aspects such as hardware/software partitioning, DFG extracting, and scheduling, etc., will continue to be optimized according to the architecture of the MUSRA to achieve a better performance.

Acknowledgement

References

[1] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, “Caffe,” in the ACM International Conference on Multimedia - MM ’14, 2014, pp. 675–678.

[2] R. Collobert, “Torch7: A matlab-like environment for machine learning,” *BigLearn, NIPS Workshop*, 2011.

[3] S. Tokui, K. Oono, S. Hido, C. S. A. S. Mateo, and J. Clayton, “Chainer: a Next-Generation Open Source Framework for Deep Learning,” *learningsys.org*.

[4] Sze, V., Chen, Y. H., Yang, T. J., & Emer, J. S. (2017). Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, *105*(12), 2295-2329.

[5] Tsmots, I., Skorokhoda, O., & Rabyk, V. (2019). Hardware Implementation of Sigmoid Activation Functions using FPGA. In 2019 IEEE 15th International Conference on the Experience of Designing and Application of CAD Systems (CADSM) (pp. 34-38). IEEE.

[6] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines,” in Proc. ICML, 2010, pp. 807–814.

[7] Istrate, R.; Malossi, A.C.I.; Bekas, C.; Nikolopoulos, D.S. Incremental Training of Deep Convolutional Neural Networks. arXiv 2018, arXiv:1803.10232.

[8] Guo, S.; Wang, L.; Chen, B.; Dou, Q.; Tang, Y.; Li, Z. FixCaffe: Training CNN with Low Precision Arithmetic Operations by Fixed Point Caffe. In Proceedings of the APPT 2017, Oslo, Norway, 14–15 September 2017.

[9] Beiu, V., Peperstraete, J. A., Vandewalle, J., & Lauwereins, R. (1994). Closse approximations of sigmoid functions by sum of step for vlsi implementation of neural networks. Sci. Ann. Cuza Univ., 3, 5-34.

[10] Nguyen, H. K., & Phan, M. T. (2017, November). RTL design of a dynamically reconfigurable cell array for multimedia processing. In 2017 4th NAFOSTED Conference on Information and Computer Science (pp. 189-194). IEEE.

[11] B.-J. Oh, “Face recognition by using neural network classifiers based on PCA and LDA,” in *2005 IEEE International Conference on Systems, Man and Cybernetics*, 2005, vol. 2, pp. 1699-1703 Vol. 2.

[12] Golub, Gene H., and Henk A. Van der Vorst. "Eigenvalue computation in the 20th century", Journal of Computational and Applied Mathematics 123.1-2 (2000): 35-65.

[13] Alaa Eleyan, and Hasan Demirel. Pca and lda based neural networks for human face recognition. Vol. 558. INTECH Open Access Publisher, 2007.

[14] MP. Rajath Kumar, and K. M. Aishwarya. "Artificial neural networks for face recognition using PCA and BPNN." TENCON 2015-2015 IEEE Region 10 Conference. IEEE, 2015.

[15] Abdi, Hervé, and Lynne J. Williams. "Principal component analysis." Wiley interdisciplinary reviews: computational statistics 2.4 (2010): 433-459.

[16] P Valarmathie, MV Srinath, K Dinakaran, An increased performance of clustering high dimensional data through dimensionality reduction technique. Theoretical and Applied Information Technology 5(6), 731–733 (2005).

[17] A.A.S. Ali, A. Amira, F. Bensaali, M. Benammar, Hardware PCA for gas identification systems using high level synthesis on the Zynq SoC, in IEEE International Conference on Electronics, Circuits, and Systems, (2013), pp. 707–710.

[18] T.C. Chen,W.Liu,L.G. Chen,VLSI architecture of leading eigenvector generation for on-chip principal component analysis spike sorting system, in International Conference of the IEEE Engineering in Medicine and Biology Society, (2008), pp. 3192–3195.

[19] A. Das, S. Misra, S. Joshi, J. Zambreno, G. Memik, A. Choudhary, An efficient FPGA implementation of principle component analysis based network intrusion detection system, in Proceedings of the Conference on Design, Automation and Test in Europe, (2008), pp. 1160–1165.

[20] 11. T. Karnthak P.Kumhom, A hardware implementation of PCAbased-on the networks-on-chip paradigm, in International Symposium on Communications and Information Technologies, (2012), pp. 834–839

[21] http://www.xilinx.com/products/silicon-devices/soc/zynq-7000.htm.

[22] G. Theodoridis, D. Soudris and S. Vassiliadis, “A Survey of Coarse-Grain Reconfigurable Architectures and Cad Tools Basic Definitions, Critical Design Issues and Existing Coarse-grain Reconfigurable Systems”, Springer, 2008.

[23] M. Zhu, L. Liu, S. Yin, et al., "A Cycle-Accurate Simulator for a Reconfigurable Multi-Media System," IEICE Transactions on Information and Systems, Vol. 93, pp. 3202-3210, 2010.

[24] Frank Bouwens, Mladen Berekovic, Bjorn De Sutter, and Georgi Gaydadjiev: “Architecture Enhancements for the ADRES Coarse-rained Reconfigurable Array” HiPEAC 2008, LNCS 4917, pp. 66-81, 2008.

[25] https://www.xilinx.com/products/boards-and-kits/zcu106.html