Design and Implementation of an FPGA-based Real-Time Face Recognition System

Janarbek Matai, Ali Irturk and Ryan Kastner
Dept. of Computer Science and Engineering, University of California, San Diego
La Jolla, CA 92093, United States
{jmatai, airturk, kastner}@cs.ucsd.edu

***Abstract*—Face recognition systems play a vital role in many applications including surveillance, biometrics and security. In this work, we present a complete real-time face recognition system consisting of a face detection, a recognition and a downsampling module using an FPGA. Our system provides an end-to-end solution for face recognition; it receives video input from a camera, detects the locations of the face(s) using the Viola-Jones algorithm, subsequently recognizes each face using the Eigenface algorithm, and outputs the results to a display. Experimental results show that our complete face recognition system operates at 45 frames per second on a Virtex-5 FPGA.**

***Keywords*-Face recognition; Eigenface; Complete face recognition system; face detection; FPGA; real-time processing.**

I. Introduction

Face recognition is a challenging research area in terms of both software (developing algorithmic solutions) and hardware (creating physical implementations). A number of face recognition algorithms have been developed in the past decades [1] with various hardware implementations [2], [3], [4], [5], [6], [7]. All previous hardware implementations assume that the input to the face recognition system is an unknown face image. Current hardware based face recognition systems are limited since they fail if the input is not a face image. A practical face recognition system should not require the input to be a face, instead would recognize face(s) from any arbitrary video which may or may not contain face(s) potentially in the presence of other objects. Therefore, an ideal face recognition system should first have a face detection subsystem which is necessary for finding a face in an arbitrary frame, and also a face recognition subsystem which identifies the unknown face image.

We define the *complete face recognition system* as a system which interfaces with a video source, detects all face(s) images in each frame, and sends only the detected face images to the face recognition subsystem which in turn identifies the face images. We designed and implemented a real-time and complete face recognition system consisting of a face detection subsystem, a downsampling module and a face recognition subsystem. The face detection subsystem uses our previously developed hardware implementation [8], [9], which is publicly available at [10]. The face recognition subsystem uses the Eigenface algorithm [1]. The complete system interfaces with a camera, sends the video data to the face detection subsystem, which in turn sends detected faces to the face recognition subsystem via the downsampling module as shown in Figure 1. Our face recognition system automatically identifies or verifies a person from a digital image, a video frame or a video source while previous works [2], [3], [4], [5], [6], [7] simply implemented what we describe as the face recognition subsystem.

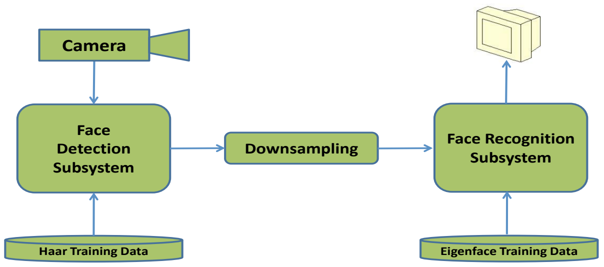

Figure 1. Overview of our complete face recognition system on an FPGA. Video data is received from the camera and sent to the face detection subsystem which finds the location of the face(s). These face(s) can be any size. The architecture then performs downsampling of the detected face to 20 × 20, and sends these 400 pixel values to the recognition subsystem. The face recognition subsystem identifies the person.

In this work, we describe the design and implementation of a face recognition architecture using Eigenface algorithm. We design and implement a face recognition subsystem on an FPGA using both pipelined and non pipelined architectures. In each case, we evaluate system performance on a different number of images. Then we show how to integrate face recognition and face detection using a downsampling module which is responsible for preprocessing detected face images from the detection subsystem to satisfy the requirements of the face recognition subsystem.

II. Face Recognition Subsystem

A. Architecture

The block diagram of the face recognition subsystem is shown in Figure 2.

1) Image Reader: The image reader module reads a 20×20 image and stores each pixel of this unknown image in the Image Frame Buffer. These are the pixels of the unknown image that need to be recognized. Previously stored pixels are sent to the Normalized Image Calculator Module in order to start normalized image calculation.

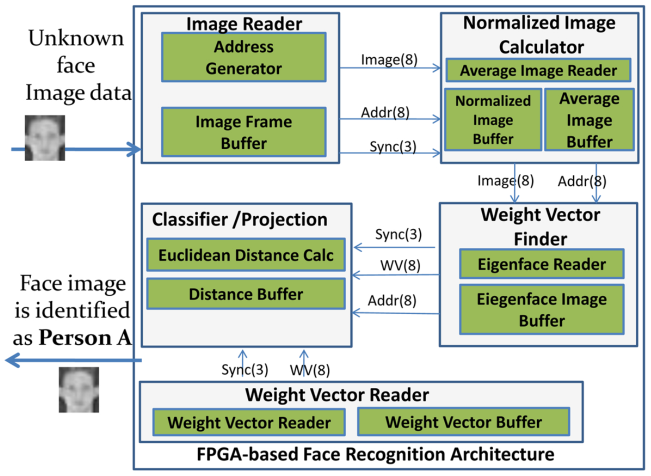

Figure 2. A block diagram of the face recognition subsystem implemented on Virtex-5 FPGA.

2) Normalized Image Calculator: The normalized image calculator module finds the differences between the average image and the input image. The average image reader reads the image pixels from the average image buffer, and then the input image pixels are subtracted to find a normalized image. The normalized image is stored in the normalized image buffer.

3) Weight Vector Finder: The weight vector finder module calculates weight values for input image using the previously calculated normalized image and Eigenvector values. The Eigenvector values are read by the Eigenface reader from the Eigenface image buffer. The Eigenvector values are stored in block RAM. The weight vector finder is the most computationally expensive step in the face recognition algorithm.

4) Weight Vector Reader: The weight vector reader is used by the classifier/projection module for retrieving the weight vector values that are generated in the training stage and stored in the block RAM.

5) Classifier/Projection: The classifier module utilizes weight vectors (from the weight vector reader module) and the weight vector for the unknown image (from the weight vector finder module). Then the classifier finds the distance between each weight vector from the weight vector reader module and the weight vector of the unknown image. For each calculation of distance, it compares the current distance value with the previous one. If the current value is smaller, then it is stored to the distance buffer. Finally, the index of the identified face, which corresponds to a minimum distance in distance buffer, is sent to the display (or other output device) as an identified face.

B. FPGA Implementation

The implementation of the face recognition subsystem is performed in two steps. The first step generates the training data and the second step is face recognition.

1) Training Data Generation: The training data is generated using the OpenCV library [11]. We used two different face image databases as training data. First, we evaluated the feasibility of the face recognition subsystem using the ORL database [12]. We refer to the ORL database as "*set1*" throughout the remainder of the article. We generated training data using 100 images from 10 different individuals from *set1*. We also collected 60 images from 6 individuals in our lab which we call "*set2*". In the following sections, we introduce the details of the implementation based on *set2*.

The training data provides us with an average image Ψ, weight vectors for each image Ω_i, and Eigenvectors μ_i. Assuming $\Gamma_1, \Gamma_2 ... \Gamma_{60}$ represent the initial 60 images provided for training, the following data is generated:

- An average image Ψ of size 20×20
- A weight vector $\Omega_1, \Omega_2, ...\Omega_{60}$ for each image. The size of the weight vector Ω_i for image i is 59. In total, we have to save a 59×60 matrix for each Ω_i in the form of $\Omega = \omega_1, \omega_1, ..., \omega_{59}$.
- 59 Eigenvectors of size 20×20. The set of Eigenvectors is $\mu_1, \mu_2, ..., \mu_{59}$

In essence, the training data transforms the sixty images into a linear combination of weight vectors and Eigenvectors. For instance, Γ_1 can be represented as:

$$\Gamma_1 - \Psi = \omega_1 * \mu_1 + \omega_2 * \mu_2 + ... + \omega_{59} * \mu_{59} \qquad (1)$$

2) Face recognition: After generating the training data, we store the average image Ψ (size of Ψ is 20×20), the weight vectors for each image Ω_i (size of Ω_i is 59×1), and the Eigenvectors μ_i (size of μ_i is 20×20) in a block RAM. Then, we implement face recognition in three steps: normalization, weight vector calculation and projection to find if $\Gamma_{unknown}$ belongs to any of the six individual's faces in training data.

Normalization: Given a 20×20 unknown input image $\Gamma_{unknown}$, the first step of the face recognition is the calculation of the normalized image Φ. Given the training data, it is straightforward to calculate the normalized image. The average image Ψ is subtracted from unknown image pixel by pixel; both the average image and the unknown input image have the same size (20×20). The average image buffer is stored in a block RAM, and the input image buffer is implemented for storing the unknown input image. Since there are 400 operations and each operation is independent, the subtraction of the average image pixels from unknown image pixels can be performed in parallel. After the normalized image is calculated, the resulting pixels are sent to the weight vector calculation step.

Weight vector calculation step: Here we find the weight vector for an unknown image using Equation 2. In this step, the $\mu_1, \mu_2, ..., \mu_{59}$ Eigenvectors and normalized image Φ are used to calculate the weight vector of the unknown input image. At this point, the normalized image Ψ is calculated and stored in a register. Therefore, the calculation of each of the 59 weight vector elements ω_i can be parallelized since the Eigenvectors are read from the block RAM independently of each other. The Eigenvector buffer has 23,600 16 bit elements (the 59 Eigenvectors each contain $20 \times 20 = 400$ 16 bit elements). The normalized image is a 20×20 matrix.

$$\omega_i = \mu_i^T \Phi \tag{2}$$

where $i = 1$ to 59.

Projection step: The Euclidean distance between the weight vector of the unknown input image and the weight vectors of the trained image are calculated using a nearest neighbor search. There are 60 weight vectors corresponding to each training image each of which containing 59 values. We calculate the Euclidean distances $d_1, d_2, ..., d_{60}$ between each of the training images and the unknown image using the weight vectors. The weight vectors of the training images are stored in the weight vector buffer in a block RAM. The size of the weight vector buffer is 59×60. The Euclidean distance calculation is performed on all 60 weight vectors Ω_i. The Euclidean distance calculation are independent operations, and therefore these two operations are performed in parallel. For each calculation of the distance value, we compare the new calculated distance value with the distance value in the distance buffer (old distance value). If the newly calculated value is smaller than the old distance value, we overwrite the newly calculated distance value and index. This process continues 60 times. Finally, the index of smallest value among $d_1, d_2, ..., d_{60}$ is returned from the distance buffer as the index of the person identified.

C. Experimental Results

We present experimental results from *set1* and *set2*. Figure 3 shows the performance comparisons between the software and hardware implementations of the face recognition subsystem using 10, 20, 25, 50 and 100 images from *set1*. When using 100 images, the face recognition subsystem achieves an average speed up of 15X over the equivalent software implementation. The software experiment was done on multi-core machine machine with Core2 Duo CPU running at 3.33 GHz with 4 GB RAM.

Figure 4 (a) and (b) shows the latency and the latency cycles respectively for 40, 50 and 60 face images from *set2* with pipelined and non-pipelined implementations. The device utilization summary when using *set2* with pipelined and non-pipelined implementations is also shown in Figure 4 (c) in number of slices, LUTs, RAMs (BRAMs), and DSP48s.

III. Implementation of the Complete Face Recognition System

In this section, we present the downsampling module used to connect the detection and the recognition subsystems. Then we describe the complete face recognition system

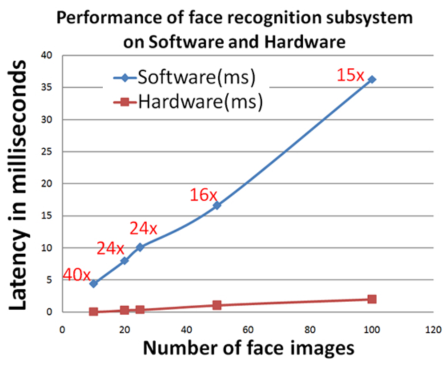

Figure 3. Performance comparisons between software and hardware implementations of the face recognition subsystem.

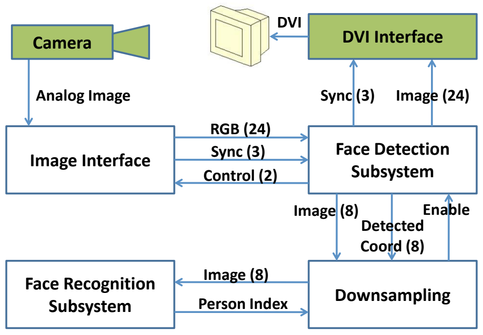

Figure 5. The architecture for the complete face recognition system consisting of the face detection and face recognition subsystems.

which is a combination of all of these subsystems. Figure 5 provides an overview of the architecture for the complete face recognition system.

The downsampling module is notified when a face is detected by the face detection subsystem. After being notified, the downsampling module reads the face image data using the coordinates, width and height given by the face detection subsystem. According to the size of the detected face image data, the downsampling module reduces the detected face to 20×20 and sends these 400 pixel values to the face recognition subsystem. The downsampling module resizes each detected face so that they are suitable as input into the face recognition subsystem.

We introduce a factor which is used to calculate how many pixels we should skip in order to downsample a $x \times x$ image into a 20×20 image. The factor depends on the size of detected face. For instance, if the size of the detected face is 60×60, then the factor would be 3. We can find the factor using $factor = detected\_face\_size/20$. Finally, when the detected face is appropriately downsampled, the downsampling module checks if the face recognition subsystem is busy. If the face recognition subsystem is available, it reads 20×20 image and returns the index of a person which belongs to the detected face. According to the returned

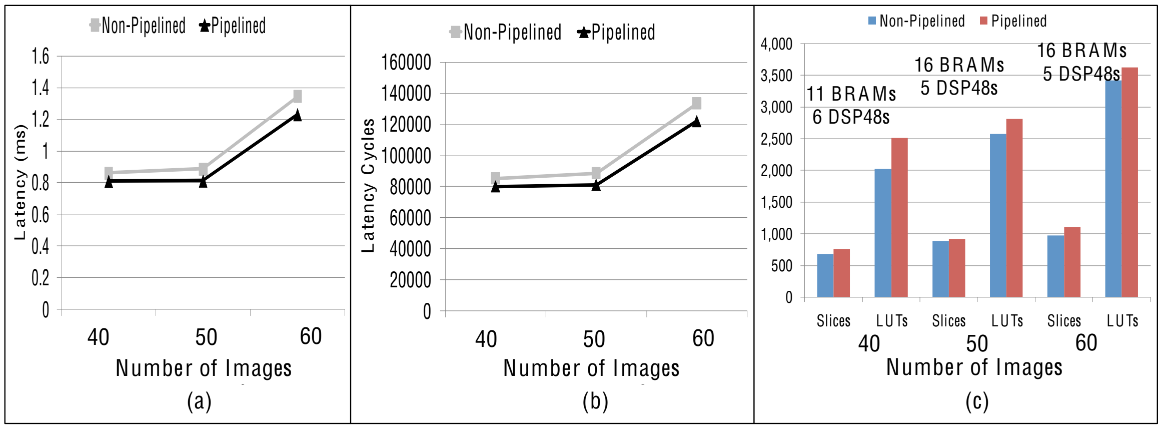

Figure 4. Part (a) shows the latency of our face recognition subsystem implementation results on an FPGA in milliseconds using both pipelined and non-pipelined implementations. Part (b) shows the latency cycles for pipelined and non-pipelined implementations. Part (c) shows the device utilization summary for the number of slices, LUTs, block RAMs (BRAMs) and DSP48s for both pipelined and non-pipelined implementations.

Table I
Device Utilization Table for the Complete System

Logic Utilization	Used	Available	Utilization
Slices	8,683	11,200	77%
Number of Slice LUTs	32,480	44,800	72%
Number of Block RAMs	84	148	56%
Number of DSP48s	11	128	8%

index of a person, we draw a box around the detected face with predefined color. Each individual's face in the set is represented by an index and each index is associated with a color.

The implementation was simulated/verified with ModelSim, and then implemented on a Virtex-5 FPGA. Table I shows the device utilization of the complete face recognition system on a Virtex-5 FPGA board. According to the experimental results, the complete face recognition system runs at 45 frames per second on VGA data.

IV. Conclusion

This paper presented the design and implementation of a complete FPGA-based real-time face recognition system which runs at 45 frames per second. This system consists of three subsystems: face detection, downsampling and face recognition. All of the modules are designed and implemented on a Virtex-5 FPGA. We presented the architectural integration of the face detection and face recognition subsystems as a complete system on physical hardware. Different experimental results of the face recognition subsystem are presented for pipelined and non-pipelined implementations.

References

[1] M. Turk and A. Pentland, "Eigenfaces for recognition," *Journal of Computer and System Sciences*, vol. 3, no. 55, pp. 119–139, 1991.

[2] I. Sajid, M. M. Ahmed, I. Taj, M. Humayun, and F. Hameed, "Design of high performance fpga based face recognition system," in *PIERS Proceedings, Cambridge*, July 2 2008.

[3] D. Chen and H. Jiu-qiang, "An fpga-based face recognition using combined 5/3 dwt with pca methods," in *Journal of Communication and Computer*, vol. 6, Oct 2009.

[4] H. T. Ngo, R. Gottumukkal, and V. K. Asari, "A flexible and efficient hardware architecture for real-time face recognition based on eigenface," in *IEEE Computer Society Annual Symposium on VLSI*, 2005.

[5] R. Gottumukkal, H. T. Ngo, and V. K. Asari, "Multi-lane architecture for eigenface based real-time face recognition," in *Microprocessors and Microsystems*, 2006, p. 216224.

[6] A. P. Kumar, V. Kamakoti, and S. Das, "System-on-programmable-chip implementation for on-line face recognition," *Pattern Recognition Letters*, vol. 28, pp. 342–349, 2007.

[7] S. Visakhasart and O. Chitsobhuk, "Multi-pipeline architecture for face recognition on fpga," in *International Conference on Digital Image Processing*, March 2009.

[8] J. Cho, S. Mirzaei, J. Oberg, and R. Kastner, "Fpga-based face detection system using haar classifiers," in *International Symposium on Field Programmable Gate Arrays*, February 2009.

[9] J. Cho, B. Benson, and R. Kastner, "Hardware acceleration of multi-view face detection," in *IEEE Symposium on Application Specific Processors*, July 2009.

[10] J. Cho and R. Kastner, "Face detection (haar classifiers) source code, http://ercbench.ece.wisc.edu."

[11] G. Bradski and A. Kaehler, *Learning OpenCV*. O'Reilly Media, Inc., 2008.

[12] "The orl database from cambridge university computer laboratory." [Online]. Available: http://www.cl.cam.ac.uk/research/dtg/attarchive