Personal Verification using Finger Vein Biometrics in FPGA-based System-on-Chip

M. Khalil-Hani, P.C. Eng
VLSI-eCAD Research Laboratory (VeCAD)
Faculty of Electrical Engineering, Universiti Teknologi Malaysia
81310 Skudai, Johor, Malaysia
khalil@fke.utm.my, peichee84@hotmail.com

Abstract— **As security and privacy protection have become important today, biometric-based personal verification has also increasingly gain popularity. However, biometric technology applies complex, compute-intensive image processing algorithms which demand to be executed on powerful computers for acceptable processing times. This prevents biometric systems to be used in applications that require low-cost embedded solutions. This paper presents a novel approach to personal verification that utilizes infrared finger vein biometrics implemented in an FPGA-based System-on-Chip platform. To improve performance, we apply the hardware-software co-design approach, using hardware acceleration for the image preprocessing tasks and image buffer management while leaving feature extraction and template matching to software. The system is prototyped on an Altera Nios2 FPGA development board running the Nios2-Linux RTOS at 100MHz clock. The proposed finger vein biometric system achieves a matching accuracy of EER (average) of 0.08% and a verification processing time of 1.27 seconds, demonstrating that the design approach taken can be effective for embedded vein biometric authentication system.**

1. Introduction

Personal verification systems based on biometrics are more reliable than the password-based systems, since biometric characteristics cannot be lost or forgotten. Also, biometric features are difficult to replicate, and the user is required to be present for the authentication process [1]. While biometric traits such as face, fingerprint, iris and voice [2-5] have been well studied, and are considered as the traditional ones, biometrics utilizing vein patterns are more recent and therefore less developed. However, with traditional biometric methods, it is relatively easy for another person to obtain unauthorized biometric data, since they achieve authentication by utilizing information from the external body.

Vein biometrics, which utilize the network of blood vessels underneath a person's skin, are proven to be unique to each finger and each individual, and are stable over a long period of time [6]. Since veins are hidden underneath the skin surface and are mostly invisible to human eye, they are not susceptible to external distortions and are extremely difficult for an intruder to acquire or duplicate the vein patterns as compared to other biometric traits. Besides, as vein biometrics involve sensing the flow of blood in the vessels, "aliveness" detection is inherent, thus ensuring that only live finger veins are capable of generating the biometric templates. Due to the uniqueness, stability, and high resistance to criminal tampering, vein patterns offer a more secure and reliable trait for a biometric authentication system.

There is now a demand for biometric technology to be deployed in low-cost embedded devices for portability and mobility. Implementations of biometric authentication systems in embedded hardware can also address the critical biometric information leakage issue [7]. This is because an embedded system can be designed effectively to provide a medium of secure communication, secure information storage, and tamper resistance against both physical and software attacks. However, embedded systems, which are typically constrained in resources and performance, hence the design of biometric systems in such embedded environments is challenging and therefore less developed, especially with vein biometrics [8].

In this paper, an infrared finger vein biometric verification system is developed targeted for an implementation in FPGA-based SoC platform. Our focus is to obtain an embedded system based on the SoC that achieves high performance and optimum matching accuracy. The system is prototyped on Altera Nios2 FPGA Stratix II EP2S180 development board running on Nios2-Linux RTOS with a 100MHz clock frequency. The rest of the paper is organized as follows. Section II presents the general description of the proposed finger vein biometric system, and section III provides the details of hardware acceleration of the image preprocessing tasks. Experimental results are given in section IV, and conclusions are made in section V.

2. Description & System Architecture

Fig. 1 shows the top-level conceptual design of the proposed finger vein biometric system. In this current version of the prototype, the vein pattern is captured by using a low-cost modified IR webcam connected to a PC. As human veins are hidden underneath skin, vein patterns cannot be observed in visible light. However, vein patterns can be acquired by the fact

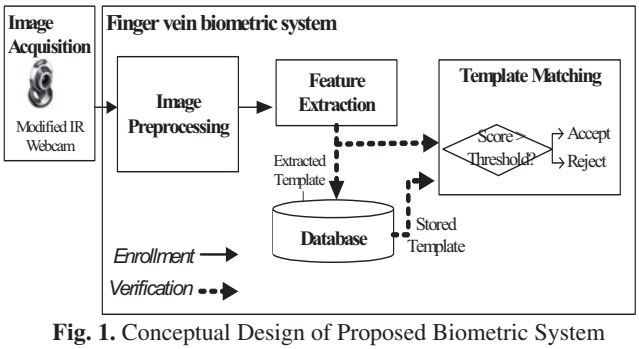

Fig. 1. Conceptual Design of Proposed Biometric System

that the infrared light (IR) with wavelength 700nm-1000nm can pass through human tissues and the haemoglobin in the blood can absorb infrared light fully [9]. In this paper, the NIR imaging technique, rather than far-infrared (FIR), is applied since it is more suitable for our purposes here, according to the study in [10]. A light transmission method as illustrated in Fig. 2 is used in which, the finger is placed in between an array of infrared light-emitting diode (IR LED) and a low cost image acquisition module using a modified webcam with an attached IR filter. In the resulting image, the vein patterns capture appear darker. Image captured is 320x240 pixels (width x height) in size, with 3 bytes per pixel. The raw finger vein image, which is in coloured *bmp* format is first converted to grayscale image, to reduce the size from 3 bytes to 1 byte per pixel and for easier image manipulation.

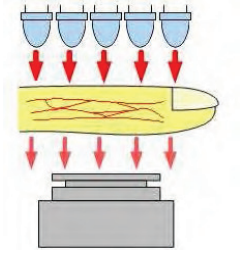

Fig. 2. Finger vein NIR imaging technique

The captured image is then transfered to an FPGA-based SoC platform for biometric processing, which involves the process flow of image preprocessing, feature extraction and template matching. As shown in Fig. 1, in the enrollment stage of the process, users register their biometric data with the system. The biometric templates are stored locally in the template database in the on-board flash memory, thus providing strong security. In the verification stage, users present their biometric traits as well as the information of the claimed identity to the system. A feature set is then extracted from the query biometric input, and the system performs one-to-one comparison of this feature set with the claimed database template. Based on the result of this comparison, the system classifies the user as either a genuine user or as an imposter.

The software-based feature extraction module utilizes the minutiae-based features extracted from the vein patterns for recognition as proposed in [11]. In this work, the minutiae points chosen include bifurcation points and ridge ending points. The most widely used method for minutiae feature extraction is the Cross Number (CN) concept [12-13]. CN is defined as number of transition from 0 to 1 (and vice versa) for a pixel P_0 to the surrounding pixels, P_1 to P_8 in a 3*3 window. The biometric template is generated by storing the minutaie type as well as the *x* and *y* coordinates of the minutiae point. Table 1 shows the correspondence of the cross numbers to the minutaie types.

Table 1. CN and Minutiae types

CN	Minutaie type
0, 1	Isolated point
2, 3	Ridge ending point
4, 5	Connecting point
6, 7	Bifurcation point
8	Crossing point

In this paper, template matching applies the Hausdorff Distance (HD) [14] to measure the dissimilarity of two images. Given two sets of points A= $\{a_1, \ldots, a_m\}$ and B= $\{b_1, \ldots, b_n\}$, the Hausdorff Distance is defined as:

$$H(A,B) = \max(h(A,B), h(B,A)) \quad , \quad h(A,B) = \|\text{a-b}\| \qquad (1)$$

where a is max element in A, b is min element in B.

h(A,B) is called the directed Hausdorff Distance from A to B. It measures the distance from point in set A to its nearest neighbor in set B, and identifies the point that is farthest from any point of B. Thus, the Hausdorff Distance H(A,B) measures the degree of mismatch between two sets of points as it reflects the distance of A that is farthest from any point of B and vice versa. A smaller value of H(A,B) would indicate a better similarity of the two sets. The directed Hausdorff Distance based on the work in [9] is very sensitive to outlier points. The Modified Hausdorff Distance (MHD) introduced in [15] overcomes this problem. In MHD, the directed Hausdorff Distance is defined as:

$$h(A,B) = \frac{1}{m}\sum_{a\in A}\min_{b\in B}\|a-b\| \qquad (2)$$

By taking the average value of all distances from a point in A to its nearest neighbour in B, rather than taking the farthest point as defined in [9], MHD decreases the impact of outlier point in the set.

Fig. 3 shows the system architecture of the proposed system in a FPGA-based SoC platform. We have applied the hardware-software partitioning that leads to an effective cost-speed tradeoff. Speed performance is achieved by the application of hardware-software co-design methodology, using hardware acceleration for the compute-intensive image preprocessing tasks and image buffer management while leaving feature extraction and template matching processes to software. The on-board SDRAM and flash memory units will contain the Nios2-Linux RTOS and the template database respectively.

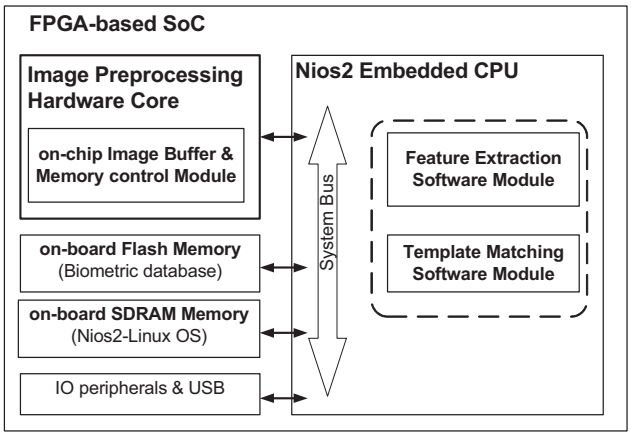

Fig. 3. System Architecture of Proposed Biometric System

3. Image Preprocessing Accelerator Engine

As mentioned earlier, the image preprocessing tasks and image buffer control and address generation are performed in a dedicated hardware accelerator engine (HW core). As shown in Fig. 4, the image preprocessing tasks include grayscale median filter, ROI (region-of-interest) extraction, image alignment and resizing, Gaussian low pass filter, image thresholding, binary median filter and thinning.

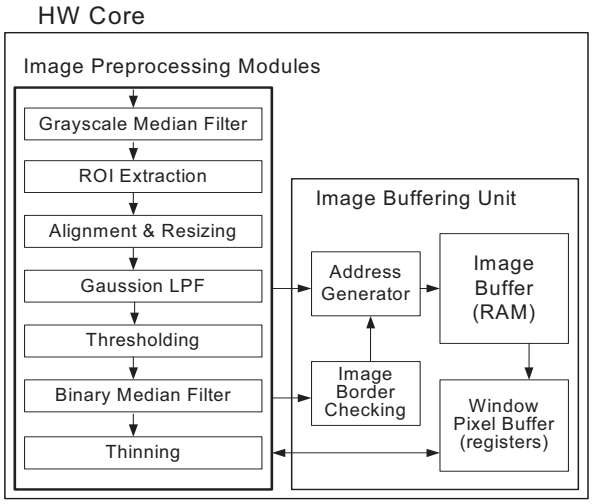

Fig. 5. Functional Block Diagram of Image Preprocessing HW

The image buffer is a shared memory implemented as an 18 bit address RAM which can contain up to 2^18=262,144 location of image pixels. The first half of the memory is reserved for input pixels, while the second half stores the output pixels. The image processing modules can take any size of image with the condition the image size (width x height) does not exceed (2^18)/2=131,072.

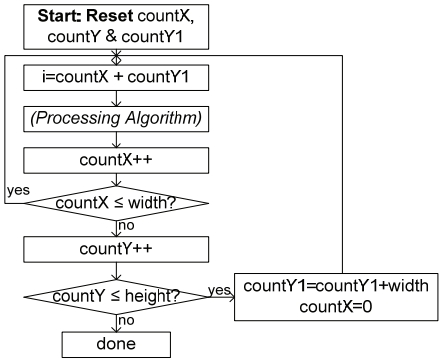

Fig. 6. Control Flowchart of Pixel address hardware

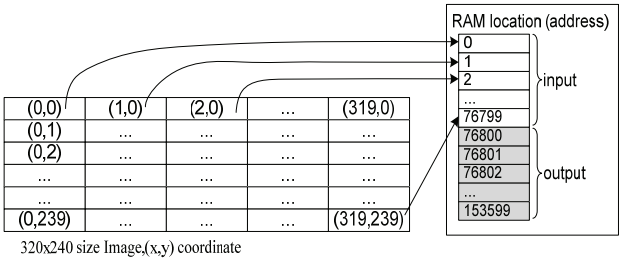

Fig. 7. Storing image pixels in RAM (example of 320x240 image size)

The 8-bit width size of the RAM is designed to operate with any grayscale image (8 bit per pixel) or binary image (1 bit per pixel). The pixels are loaded into the RAM according control flowchart shown in Fig. 6. Fig. 7 depicts how the pixels are stored in the RAM for a sample 320x240 image size. The looping path, *x* then by *y,* is followed when writing the image pixels value to the RAM. After the image is processed, the output data is stored, starting at the next address location after the last address of the input image. Therefore, the relationship between the (x,y) coordinates and the RAM addresses is as illustrated in Fig. 8.

x / y	0	1	2	...	...	319
0	0	1	2	...	...	319
1	320	321	322	...	...	639
2	640	641	642	...	...	959
...	...	...	...	...	...	...
...	...	...	...	...	...	...
237	75840	...	...	...	76799	76799
238	76160	...	...	...	76799	76799
239	76480	...	...	...	76799	76799

Fig. 8. Pixel coordinate (x,y) relation to RAM address

Let us now look into the design of the image prepocessing modules. But, due to the lack of space in this paper, only the Grayscale Median filter, ROI Extraction with Canny algorithm, Gaussian Low Pass Filter and Binary Median filter are presented.

Grayscale Median Filter— This filter is employed to remove background noise in the captured vein image whilst preserving the edges. The algorithm applied is based on the *odd-even mergesort* sorting. A 7x7 pipelined median filter hardware is designed based on the work in [16]. Fig. 9 shows an example of the median filter network for 13 elements, where the horizontal lines represent the inputs, a vertical arrow represents a comparator with the arrow pointing towards the larger value of its two inputs. The proposed median filter network for a 7x7 processing window requires a total of 342 comparators.

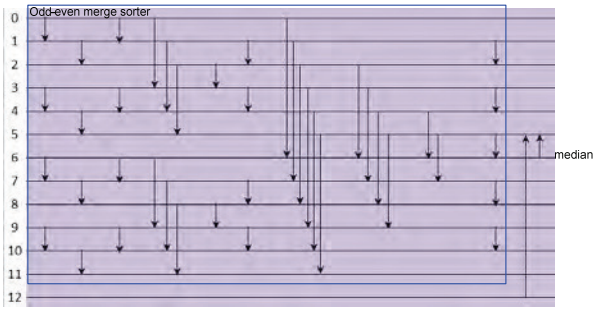

Fig. 9. Example of Median filter network for 13 elements

ROI Extraction with Canny edge detection algorithm— This involves extracting the finger region by applying Canny edge detection algorithm, edge smoothing with morphological dilation, and finger region filling. The Canny edge detection algorithm is as follows: (1) Image smoothing is performed, applying grayscale median filter to remove noise so as to be mistaken for an edge, (2) a 2-D first derivative of the Gaussian operator is convolved with the smoothed image to obtain the *dx* (horizontal gradient) and *dy* (vertical gradient), (3) The edge strength G (gradient magnitude) and direction θ (gradient phase) is then determined from the *dx* and *dy*, (4) a non-maxima suppression process is then performed, which eliminates the pixels that have no local maximum G, and (5) hysteresis thresholding is applied to which edges are significant, classifying them into strong edges and weak edges. Edge

tracking is then performed to find the weak edges which are connected to strong edges and classified them as strong edges, resulting in removing any broken edges.

The 2-D convolution is described by the equation (3), where O(x,y) is the output of the convolution, I(x,y) is the original image, f(x,y) is the convolution kernel, and M is the size of convolution kernel in both dimensions:

$$O(x,y) = I(x,y) * f(x,y) = \sum_{i=-M/2}^{M/2} \sum_{j=-M/2}^{M/2} I(i,j).f(x-i,y-j) \quad (3)$$

The first derivative of Gaussian convolution filter f(x,y) is:

$$f(x) = -\frac{x}{\sigma^2}\exp(-\frac{x}{\sigma^2}) \quad (4)$$

For sigma, σ=1.0, we will get the following convolution kernel for both horizontal and vertical convolution,

$$f_{dGx} = (f_{dGy})^T = [\ 0.0133\ 0.1080\ 0.2420\ \ 0\ -0.2420\ -0.1080\ -0.133] \quad (5)$$

Converting these floating point values in equation (5) to integers with a multiply mask of 10000, we obtain d_x and d_y as follows:

$$d_x = I(x, y-3) \text{ x } (-133) + I(x, y-2) \text{ x } (-1080) + ... \quad (6)$$
$$d_y = I(x-3, y) \text{ x } (-133) + I(x-2, y) \text{ x } (-1080) + ...$$

Implementing multiplication in hardware is slow and consumes more logic, hence in this work, we simplify the multiplication by using shift left and add operators. For example, since 133 = 128+4+1=27+22+1, multiplication by 133 can be written as (*in*<<7) + (*in*<<2) + *in*, where *in* is the input we wish to multiply by 133. The gradient magnitude, G, and phase are determined by:

$$G = \sqrt{d_x^2 + d_y^2} \quad \text{and} \quad \theta = \arctan(d_y/d_x) \quad (7)$$

G is simplified by using only shift and add operations, by applying $G = \sqrt{(a^2 + b^2)} = \max\{7x/8 + y/2, x\}$ where $x = \max\{|a|, |b|\}$ and $y = \min\{|a|, |b|\}$. The gradient direction is quantizing into four discrete directions, *Discrete Angle*. Table 2 shows the values assigned during quantization:

Table 2. Value assigned for gradient direction quantization

Gradient Direction (θ)	Discrete Angle
0 ≤ θ < 45 or 180 ≤ θ < 225	1
45 ≤ θ < 90 or 225 ≤ θ < 270	2
90 ≤ θ < 135 or 270 ≤ θ < 315	3
135 ≤ θ < 180 or 315 ≤ θ < 360	4

Since we have to multiply the Gaussian mask when finding the *dx* and *dy*, any value using these parameters should be divided by 10000. The square root in Equation (7) is removed since the gradient magnitude is only use for comparison. The division with 10000 can be simplified by using shift and add operation. 10000 can be approximate as:

13421/134217728=(213+212+210+26+25+23+22+1)/227.

Fig. 10 shows the hardware module *divideBy10000*.

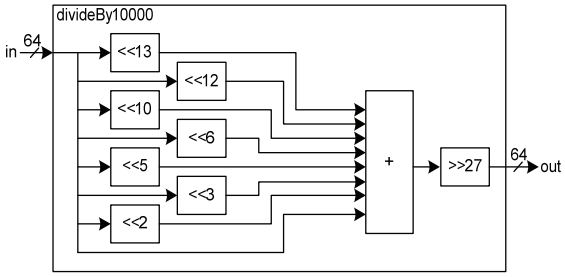

Fig. 10. Hardware module *divideBy10000*

The gradient magnitude G for all the 9 pixels in a 3x3 window can be determined using the hardware design given in Fig. 11.

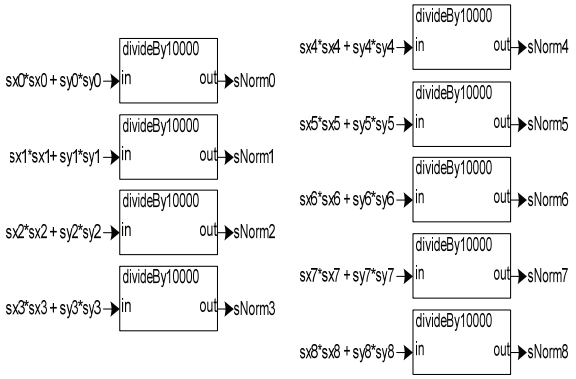

Fig. 11. Computing G for pixels in a 3x3 window

Gaussian Low Pass Filter— A spatial low pass filter with the selected convolution mask given in Fig. 12 is applied to the image to smooth out the sharp transitions in gray level and remove high frequency noise in the image. The convolution of the input image with the convolution mask is performed as shown in Fig. 13. The operation is applied to each pixel in the image. Z is the input image pixel, and W is the selected convolution mask.

$$\frac{1}{273} \times \begin{matrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{matrix}$$

Fig. 12. Convolution mask

$$\begin{matrix} Z_0 & Z_1 & Z_2 & Z_3 & Z_4 \\ Z_5 & Z_6 & Z_7 & Z_8 & Z_9 \\ Z_{10} & Z_{11} & Z_{12} & Z_{13} & Z_{14} \\ Z_{15} & Z_{16} & Z_{17} & Z_{18} & Z_{19} \\ Z_{20} & Z_{21} & Z_{22} & Z_{23} & Z_{24} \end{matrix} * \begin{matrix} W_0 & W_1 & W_2 & W_3 & W_4 \\ W_5 & W_6 & W_7 & W_8 & W_9 \\ W_{10} & W_{11} & W_{12} & W_{13} & W_{14} \\ W_{15} & W_{16} & W_{17} & W_{18} & W_{19} \\ W_{20} & W_{21} & W_{22} & W_{23} & W_{24} \end{matrix}$$

Fig. 13. Convolution of input image with the convolution mask

Suppose Z_{12} is the pixel we wish to perform the convolution, so Z_0 to Z_{24} are the surrounding pixels. The corresponding equation is shown in (8), and the matrix operation is given in

The operation results in the final pixel being the weighted sum of the neighboring pixels

$$Z_{12} = \sum_{i=0}^{24} W_i \times Z_i \qquad (8)$$

Binary Median Filter— This filter functions to eliminate noise after thresholding stage. Since we are operating on a binary image, which consists only pixels of values '0' or '255', we can replace the sorting algorithm by using a simple count operation. The window size used in our system is 5x5 with iteration of 3. First, the algorithm will look at all pixels in the w*w size window, and count up one if the value is equal to '0'. If the total count value us greater than $\left(\frac{w \times w - 1}{2}\right)$, then the output is assign as '0' or else is assign as '255'. By doing so, the median filter for binary image is actually replacing the center pixel by the dominant pixel value in a w*w window. As a result, by using this counter-based design, the the median filtering process has been significantly speeded up by 80% compared to conventional method.

4. Experimental Results

Fig. 14 illustrates the output image of each module in the image preprocessing HW core. The figures show clearly the vein pattern extracted. Fig. 15 illustrates minutaie template extracted from a sample finger vein pattern.

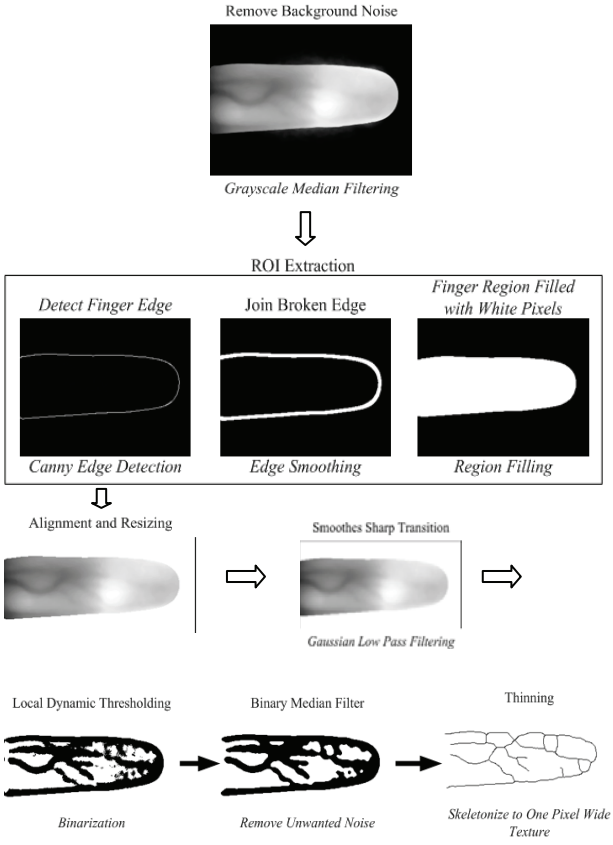

Fig. 14. Output image of each module in Image Preprocessing

Experimental work is performed on a database consisting of 100 finger vein images from 20 different fingers (5 samples for each finger).

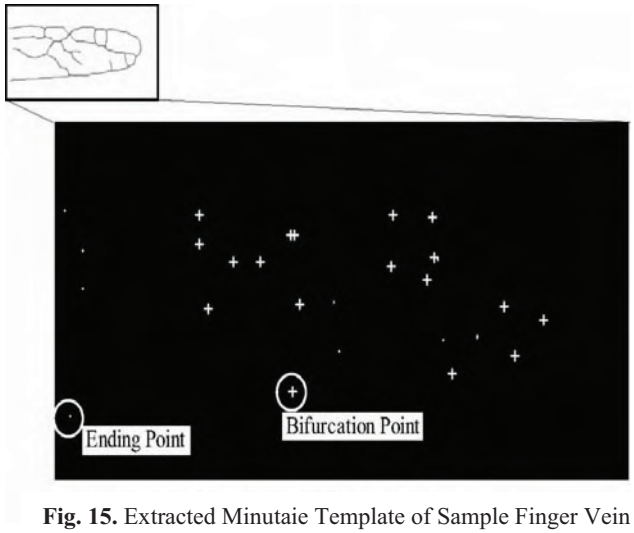

Fig. 15. Extracted Minutaie Template of Sample Finger Vein

Fig. 16 illustrates the distribution of the client and imposter access attempts against the MHD matching scores. It can be observed from the plot that the matching score distribution discriminates well between clients and imposters. We use different threshold values for different users in the database. The experiment on this database gives us an Equal Error Rate (EER) of 0.08% average. The matching accuracy achieved is encouraging and, the system can achieve higher accuracy by adding noise removal after the binary median filter or the thinning process. However, this would burden system resources already limited by the capabilities of the 50MHz fixed-point embedded processor.

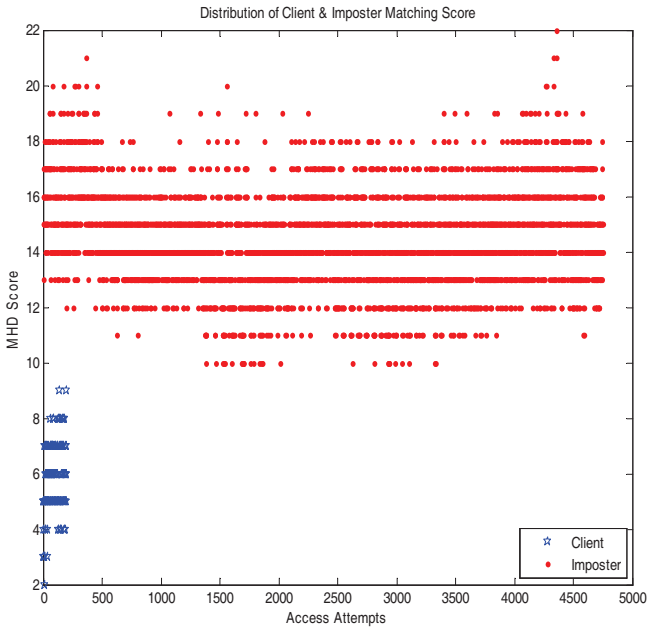

Fig. 16. Distribution of client and imposter MHD matching score of the proposed

The SoC is prototyped on an Altera Nios2 FPGA development board running the Nios2-Linux OS at 50MHz clock. Table 3 gives the speed performance (execution times) of the image preprocessing modules for the embedded SW execution (on Nios2 CPU) and the corresponding HW execution in the HW Core. The hardware accelerator core achieve an average of 27 speed gain over the firmware version. Table 4 gives the system performances, showing that the biometric verification process in the SoC-based implementation is about 20 times (=24.23/1.27) faster than the software-based equivalent. This performance can further be improved if the image preprocessing accelerator core employs pipelining and other high performance architectures in its design, and feature extraction is also performed in hardware.

Table 3. Execution Times of each module in Image Preprocessing core, running at 50MHz

Image Preprocessing Modules	Execution Time (sec)		Speed Gain (A/B)
	SW in Nios2 (A)	HW accelerator (B)	
Grayscale Median filter (7x7)	9.43	0.02	470
ROI extraction	7.01	0.39	18
Image alignment & resizing	0.95	0.01	95
Gaussian low pass filter (5x5)	1.04	0.01	104
Local Thresholding (19x19)	1.59	0.03	53
Binary Median filter (5x5, 3)	1.47	0.04	37
Thinning	2.53	0.41	6.2
Total:	**24.02**	**0.91**	**27**

Table 4. Performance – SoC against SW-based Implementation*

Process	Processing Time (sec)	
	SW-based	SoC-based
Image Preprocessing	24.02	0.91
Feature Extraction & Template Matching (SW)	0.21	0.21
IO data transfer	-	0.15
Total:	**24.23**	**1.27**

*SoC Implementation: Nios2 cpu + Image Preproc. HW Core
SW-based Implementation: Embedded Software on Nios2 cpu

5. Conclusion

Based on the results, it can be concluded that a finger vein authentication system targeted for embedded system, which is implemented on Altera Nios II FPGA prototyping board, running on RTOS Nios2-Linux has been successfully developed. The experimental results have shown the performance bottlenecks, and opportunities for optimizing the system have been identified. The proposed finger vein biometric system achieves a matching accuracy of EER (average) of 0.08% and a verification processing time of 1.27 seconds, demonstrating that the design approach taken can be effective for embedded vein biometric authentication system.

Acknowledgement

This work is supported by the Ministry of Higher Education Malaysia (MoHE) and Universiti Teknologi Malaysia (UTM) under University Grant Vote No. Q.J130000.7123. 02H39.

References

[1] U. Uludag, S. Pankanti, S. Prabhakar, and A. K. Jain, "Biometric cryptosystems: issues and challenges," Proc. of the IEEE, vol. 92, pp. 948-960, 2004.
[2] T. Matthew and P. Alex, "Eigenfaces for recognition," J. of Cognitive Neuroscience, Vol. 3, pp. 71-86, 1991.
[3] A. K. Jain, H. Lin, and R. Bolle, "On-line fingerprint verification," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 19, pp. 302-314, 1997.
[4] S. Lim, K. Lee, O. Byeon and T. Kim. "Efficient iris recognition through improvement of feature vector and classifier", Electronics and Telecommunications Research Institute Journal, vol. 23, no. 2, pp. 61-70, 2001.
[5] A. M. Judith, "Voice biometrics," Communications of the ACM, vol. 43, pp. 66-73, 2000.
[6] Y. H. Ding, D. Y. Zhuang, and K. J. Wang, "A study of hand vein recognition method," in IEEE Int. Conference on Mechatronics and Automation, pp. 2106-2110, 2005.
[7] S. L. Yang, K. Sakiyama, and I. M. Verbauwhede, "A compact and efficient fingerprint verification system for secure embedded devices," in Asilomar Conference on Signals, Systems and Computers, pp. 2058-2062, 2003.
[8] N.Aaraj *et.al*, "Architectures for efficient face authentication in embedded systems," in Proceedings Design, Automation and Test in Europe, pp.6, 2006.
[9] D. Mulyono and H. S. Jinn, "A study of finger vein biometric for personal identification," in Int. Symposium on Biometrics and Security Technologies, pp. 1-8, 2008.
[10] L. Y. Wang and G. Leedham, "Near- and Far- Infrared Imaging for Vein Pattern Biometrics," in IEEE Int Conf on Video and Signal Based Surveillance, pp. 52-52, 2006.
[11] L. Y. Wang, G. Leedham, and D. S. Y. Cho, "Minutiae feature analysis for infrared hand vein pattern biometrics," Pattern Recognition, vol. 41, pp. 920-929, 2008.
[12] R. Thai, "Fingerprint image enhancement and minutiae extraction", School of Computer Science and Software Engineering, the University of Western Australia, 2009.
[13] X. Sun and Z.M. Ai, "Automatic feature extraction and recognition of fingerprint images", Proc of ICSP', 1996.
[14] O.Jesorsky *et.al*, "Robust face detection using Hausdorff Distance", Lecture Notes In Computer Science, Springer-Verlag , Vol. 2091, Halmstad, Sweden, pp.90-95, 2001.
[15] M.P. Dubuisson & A.K Jain, "A modified Hausdorff distance for object matching", Proc. IAPR Int. Conf. on Pattern Recognition, vol.1, Jerusalem, pp.566-568, October, 1994.
[16] K.L. Chung, "A fast Pipelined Median Filter Network", Signal Processing, 51(2), pp.133-136, 1996.