CAPITAL UNIVERSITY OF SCIENCE AND TECHNOLOGY, ISLAMABAD

Accelerating Fingerprint Identification using FPGA for Large Scale Applications

by

Mohsin Shafiq

A thesis submitted in partial fulfillment for the degree of Master of Science

in the

Faculty of Engineering

Department of Electrical Engineering

2018

This piece of research is dedicated to my loving and supportive family. And to all my teachers who have helped and guided me throughout my life.

CAPITAL UNIVERSITY OF SCIENCE & TECHNOLOGY
ISLAMABAD

CERTIFICATE OF APPROVAL

Accelerating Fingerprint Identification using FPGA for Large Scale Applications

by

Mohsin Shafiq CE141002

THESIS EXAMINING COMMITTEE

S. No.	Examiner	Name	Organization
(a)	External Examiner	Dr. Mubeen Ghafoor	CIIT, Islamabad
(b)	Internal Examiner	Dr. Muhammad Faisal Iqbal	CUST, Islamabad
(c)	Supervisor	Dr. Imtiaz Ahmed Taj	CUST, Islamabad

Dr. Imtiaz Ahmed Taj
Thesis Supervisor
May, 2018

Dr. Noor Muhammad Khan	Dr. Imtiaz Ahmed Taj
Head	Dean
Dept. of Electrical Engineering	Faculty of Engineering
May, 2018	May, 2018

Author's Declaration

I, **Mohsin Shafiq** hereby state that my MS thesis titled "**Accelerating Fingerprint Identification using FPGA for Large Scale Applications**" is my own work and has not been submitted previously by me for taking any degree from Capital University of Science and Technology, Islamabad or anywhere else in the country/abroad.

At any time if my statement is found to be incorrect even after my graduation, the University has the right to withdraw my MS Degree.

Mohsin Shafiq

Registration No: CE141002

Plagiarism Undertaking

I solemnly declare that research work presented in this thesis titled "*Accelerating Fingerprint Identification using FPGA for Large Scale Applications*" is solely my research work with no significant contribution from any other person. Small contribution/help wherever taken has been dully acknowledged and that complete thesis has been written by me.

I understand the zero tolerance policy of the HEC and Capital University of Science and Technology towards plagiarism. Therefore, I as an author of the above titled thesis declare that no portion of my thesis has been plagiarized and any material used as reference is properly referred/cited.

I undertake that if I am found guilty of any formal plagiarism in the above titled thesis even after award of MS Degree, the University reserves the right to withdraw/revoke my MS degree and that HEC and the University have the right to publish my name on the HEC/University website on which names of students are placed who submitted plagiarized work.

Mohsin Shafiq

Registration No: CE141002

Acknowledgements

All the praises and thanks be to Allah Almighty, who has enabled me to complete this research.

I would like to thank my respected supervisor Dr. Imtiaz Ahmed Taj and Dr. Mubeen Ghafoor for their kind support and guidance. I acknowledge that the fingerprint database used for purpose of this research is taken from fingerprint verification competition [1]. Also open source implementation of Hong's [2] and Chikkerur's [3][4] algorithm is used.

I am thankful to my parents who are, and always will be, the main reason behind my every success. I would like to thank my loving wife for her support and for dealing with my lack of presence for the last couple of years. In the end, I would like to mention my kids who motivate me to be better.

Abstract

Real time fingerprint identification for large scale applications is a very challenging task not only from computations point of view but also in terms of accuracy.In this research both these challenges are addressed and accurate minutiae matching algorithm specially designed for hardware based accelerator is proposed.

The research has several contributions; An accurate fingerprint matching algorithm based on minutiae pattern matching is initially implemented and tested on CPU. Modifications are made in the algorithm in order to make sure efficient use of hardware resources without any significant effect on accuracy. The RTL design for the algorithm is developed in order to port it to an FPGA. Further study is done to optimize the RTL level hardware design so that it consumes fewer resources and works efficiently for real-time implementation in larger AFIS. As a result, a hardware accelerator for minutiae pattern matching, that is capable of performing multimillion matches per second is developed. The design is scalable and can be customized according to the application needs. To the best of our knowledge, the design outperforms other fingerprint matching design implemented on FPGA. It is ensured through rigorous testing that the fingerprint-matching unit is not only fast, but it also gives accurate matching results.

In future, as an extension of this work, a complete system is envisioned to be developed that involves real-time fingerprint acquisition, enhancement, feature extraction, encoding and matching with database templates.

Contents

List of Figures

List of Tables

Abbreviations

PIN	Postal Index Number
AFIS	Automated Fingerprint Identification System
DNA	Deoxyribonucleic Acid
ROI	Region of Interest
FFT	Fast Fourier Transform
IFFT	Inverse Fast Fourier Transform
STFT	Short Time Fourier Transform
CBFS	Coupled Breath Fast Search
CPU	Central Processing Unit
GPU	Graphics Processing Unit
FPGA	Field Programmable Gate Array
ASIC	Application Specific Integrated Circuit
CLB	Configurable Logic Block
LUT	Look Up Table
DB	Database
FF	Flip Flop
HDL	Hardware Descriptive Language
DSP	Digital Signal Processing
RAM	Random Access Memory
DRAM	Distributed Random Access Memory
ROM	Read Only Memory
RTL	Register-transfer Level
EER	Equal Error Rare
MSB	Most Significant Bit

LSB	Least Significant Bit
UUT	Unit Under Test
MPS	Matches Per Second
PCIe	Peripheral Component Interconnect Express

Chapter 1

Introduction

1.1 Overview

Biometrics is a modern-day methodology for human identification, which is based on computable and communicative biological features of human beings. In early ages, of mankind, biometrics including handprints and fingerprints were used among human beings to mark their caves as an identification sign. Later in the fourteenth century, Chinese traders developed the methodology of implementing fingerprints as their signatures in business agreements. In current era of technology, use of e-banking and e-commerce has grown manifolds, and security features in such applications based on user identification have become vital. For such vigorous identification, use of biometrics is becoming the technology of choice as compared to conventional cards and PIN codes, which are more likely to be compromised. Biometrics are generally considered the most reliable and stable form of identification for security purposes [5]. Recently, Fingerprint based identification has also gained a lot of attention in commercial applications such as mobile phones or telecommunication networks and surveillance. In applications like e-banking, e-commerce and large scale security systems there is a need for a large scale AFIS system [6], that can work in real-time.

1.2 Biometrics Behaviors used for Identification

nowadays, several biometrics features are being utilized for the human identification. These biological characteristics include palm impressions, fingerprints pattern, DNA, voice forms, Iris identification and many more. Figure 1.1 shows the basic classification of biometrics and some of the common type of biometrics from each class. Biological features are considered more reliable for the purpose of identification as compared to the behavioral features. A brief discussion on the biological biometrics is presented in subsequent sections.

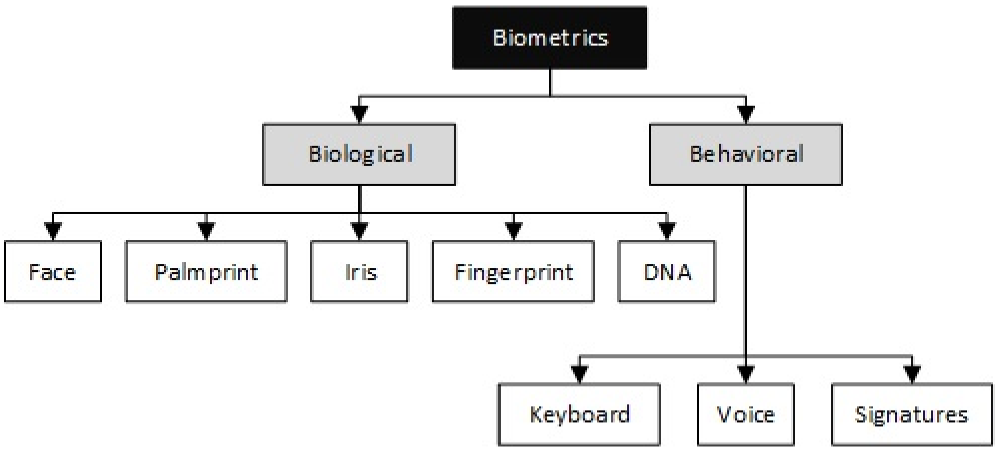


FIGURE 1.1: Biometric Features Used for Human Identification

1.2.1 Fingerprints

A fingerprint may be considered as a sinusoidal wave of positioned points having same frequency throughout the sinusoid image. These points are referred as ridges and furrows representing peaks and troughs in the wave. A sample is shown in figure 1.2.

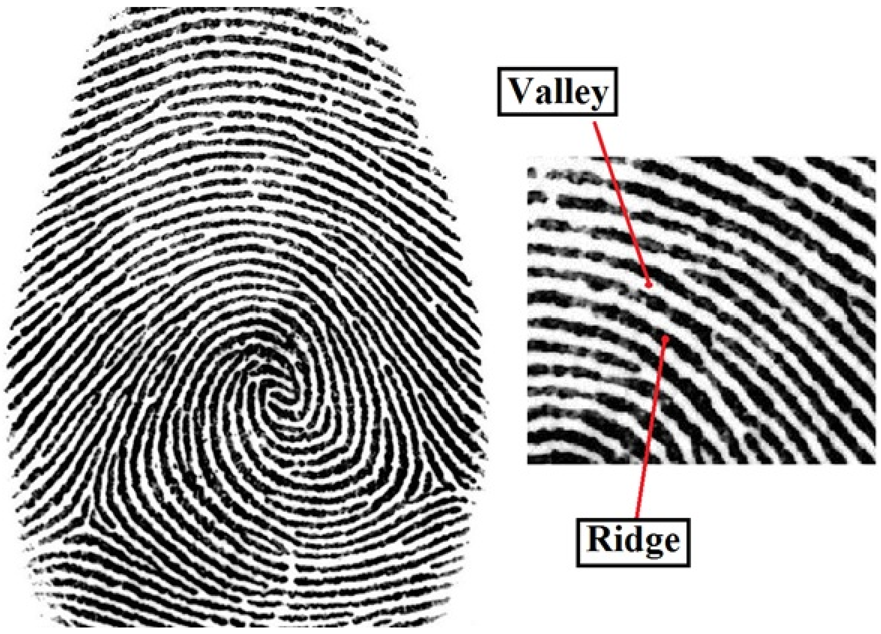


FIGURE 1.2: Finger Print Structure

The ridges in the fingerprint image have very high uniqueness due to certain biometric features called 'minutiae'. Minutiae is a feature, derived from the discontinuities in the pattern of ridges. Ridges have different type of discontinuities out of which, ridge bifurcation and ridge ending are considered most important. A fingerprint image showing these features is shown in figure 1.3 [7].

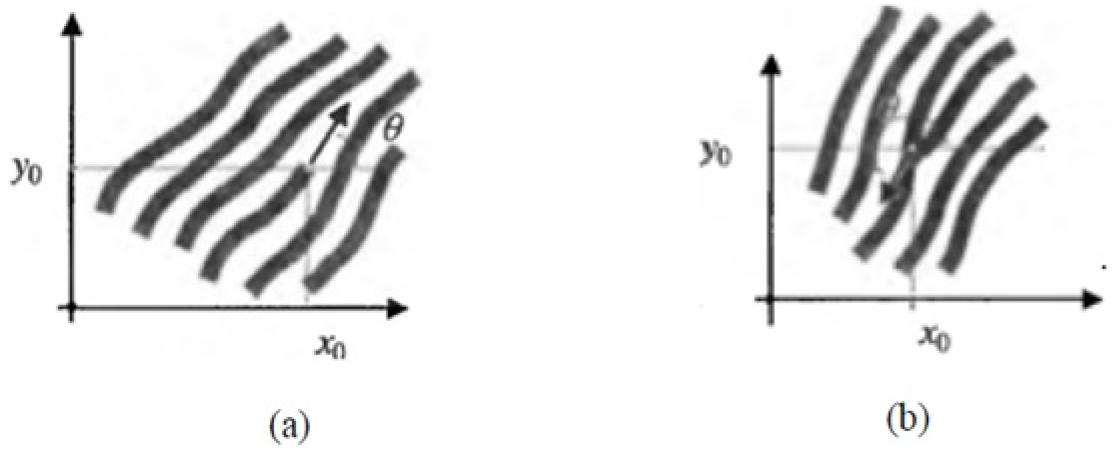


FIGURE 1.3: Common Minutia Types a) Ending b) Bifurcation

1.2.2 Face

Face is another important Biometric feature used in authentication and identification. Modern applications have high demand for facial recognition based security as it can be achieved non-intrusively. Face recognition is used in wide range of viable and forensic applications. Algorithm based on facial recognitions have been developed on large scale but there are still limitations and challenges due to which, face recognition is not being used as a major authentication feature. Challenges include pose variation, aging effects and background illumination [8].

1.2.3 Iris

Iris recognition feature is considered more stable than face recognition. Iris is the circular shape area between the retina and lens of the eyes. It includes many unique pattern features including furrows, freckles and coronas.

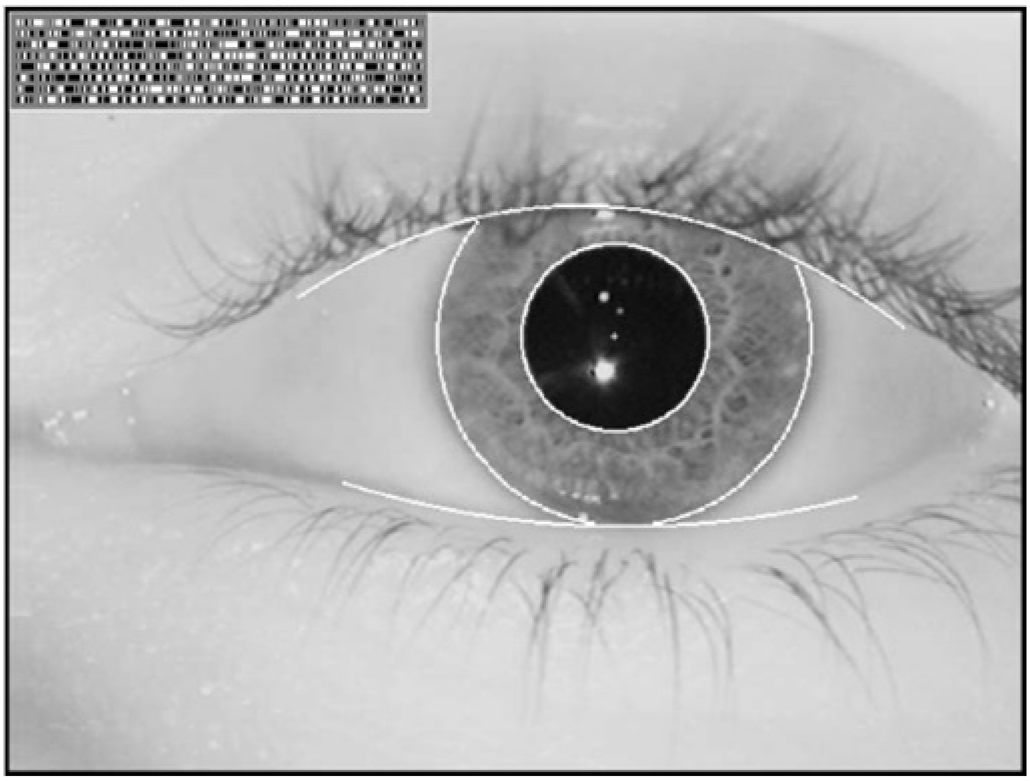


FIGURE 1.4: Example of Iris Pattern [9]

The important thing about Iris is that its patterns and features remain consistent throughout the life span as compared to face recognition. However, there are some challenges associated with using Iris as an identification metric too. Capturing the complex and unique patterns of Iris require very accurate, high-quality sensors, which are expensive, hence making this technique cost inefficient. Another issue regarding iris image acquisition is that Iris structure can only be captured from a specified distance, using the focal plane of camera. Therefore, this requires more cooperation from the user and might become a tiresome and complicated exercise.

1.2.4 Palm Impressions

Palm impressions are the inner portion of the human hand. It last from the roots of the fingers to the wrist. Palmprints provide a larger area as compared to fingerprints and it includes some extra features as well. These features include principal lines, minutia, ridges and textures. Principal lines consist of heart line, life line and headline. Palmprint is shown in the figure 1.5 [10].

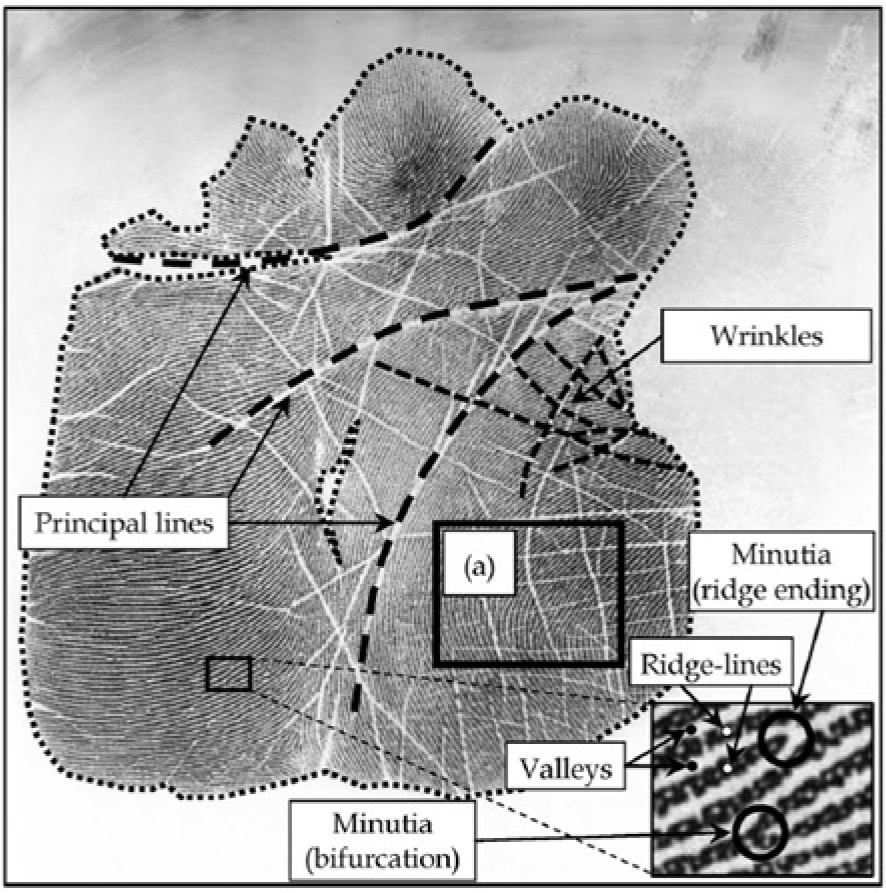


FIGURE 1.5: Palmprint of Human Hand

1.3 Comparison of Biometrics

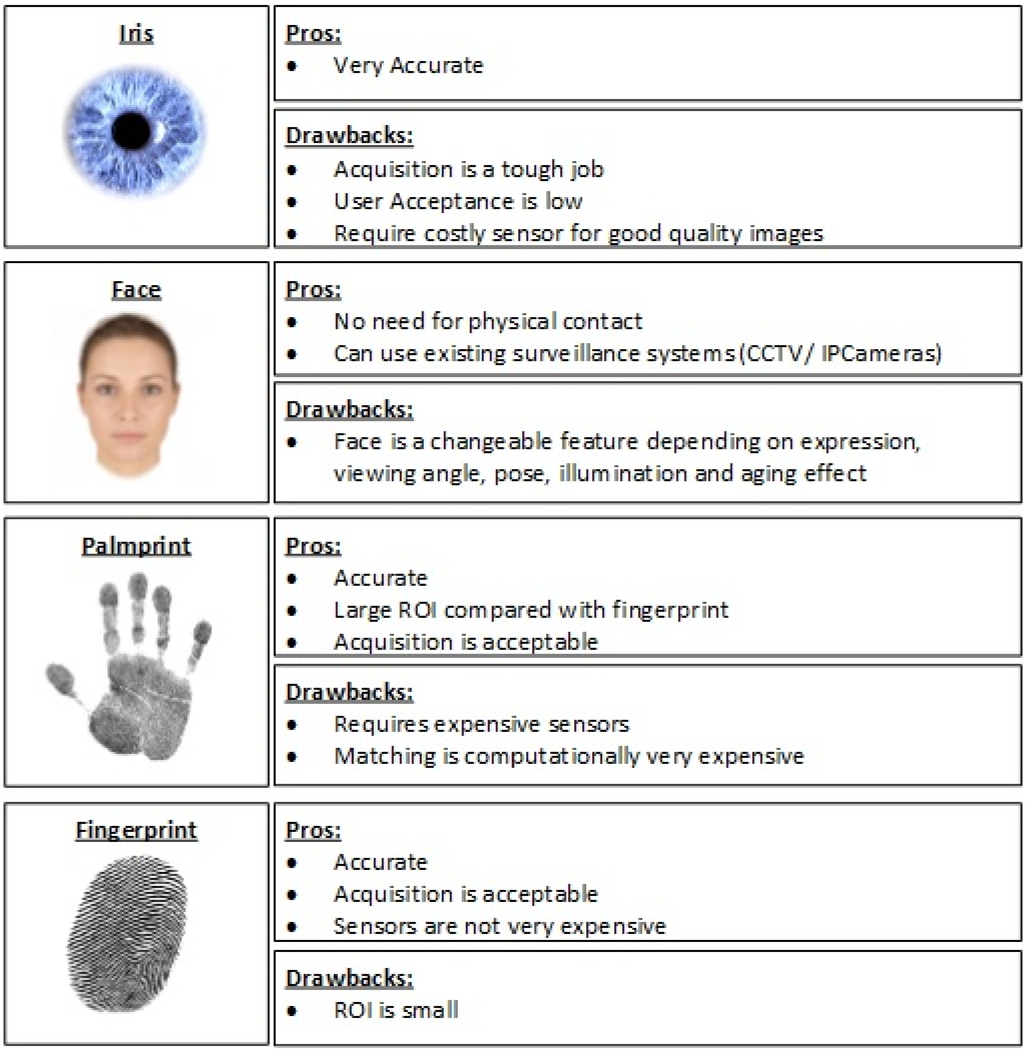
Biometric features such as finger impressions, palm patterns, Face recognition and Iris are used in authentication processes based on certain evaluations which include uniqueness of patterns, steadiness over time and adequacy for masses. Comparisons of these biometric features is discussed in this section.

Face recognition is widely used biometric feature of identification however; automated face recognition is not very reliable. Main reason behind this factor is that the face features are not constant and may change according to age, pose, brightness and expressions [8]. Identification processes based on Iris recognition is considered most reliable [9], but acquiring Iris image and its processing makes the process expensive and tiresome. Palmprint provides better patterns and more features compared to a fingerprint because of its large area. However, real time matching mechanism do not prefer palmprints as storing and matching palmprint templates require very large space and computation capability [10].

As an instance, VeriFinger [11], a fingerprint matcher machine performs 15000 matches per second for fingerprint matching however; it performs only three matches per second using palmprints. This means palmprint processing technique is five thousand times slower than fingerprint identification process.

Fingerprint biometric feature in comparison is fast, reliable and is accepted by masses. Therefore, for large databases like National Identity Cards (NIC), civil registration and even banking systems use fingerprints as the main identification metric [7]. Based on a study, fingerprint based systems generate more revenue compared to other biometric systems in market [12].

A comparison of these features is shown in figure 1.6



Iris	**Pros:** • Very Accurate **Drawbacks:** • Acquisition is a tough job • User Acceptance is low • Require costly sensor for good quality images
Face	**Pros:** • No need for physical contact • Can use existing surveillance systems (CCTV/ IPCameras) **Drawbacks:** • Face is a changeable feature depending on expression, viewing angle, pose, illumination and aging effect
Palmprint	**Pros:** • Accurate • Large ROI compared with fingerprint • Acquisition is acceptable **Drawbacks:** • Requires expensive sensors • Matching is computationally very expensive
Fingerprint	**Pros:** • Accurate • Acquisition is acceptable • Sensors are not very expensive **Drawbacks:** • ROI is small

FIGURE 1.6: Biometric Features Comparison

1.4 Components of Fingerprint Matching System

A general fingerprint identification system includes two processes namely:

1. Fingerprint encoding
2. Fingerprint matching

1.4.1 Fingerprint encoding

In this process, fingerprint impression is first obtained by a sensor after which the output image is used for feature mining. In the mining step, features like minutiae are extracted from the image. The most preferred way of extracting minutiae is preprocessing and enhancement before applying feature extraction. Preprocessing is applied to obtain the best results by removing unwanted deficiencies in the image. After minutiae extraction, minutiae is encoded where biometric code for image is computed and saved into a database.

1.4.2 Fingerprint Matching

In the matching process, biometric codes for unique finger prints are computed and matched with the database images. Following figure 1.8 shows the modular block diagram of fingerprint matching system. Image acquisition, preprocessing, enhancements and matching are parts of fingerprint matching system.

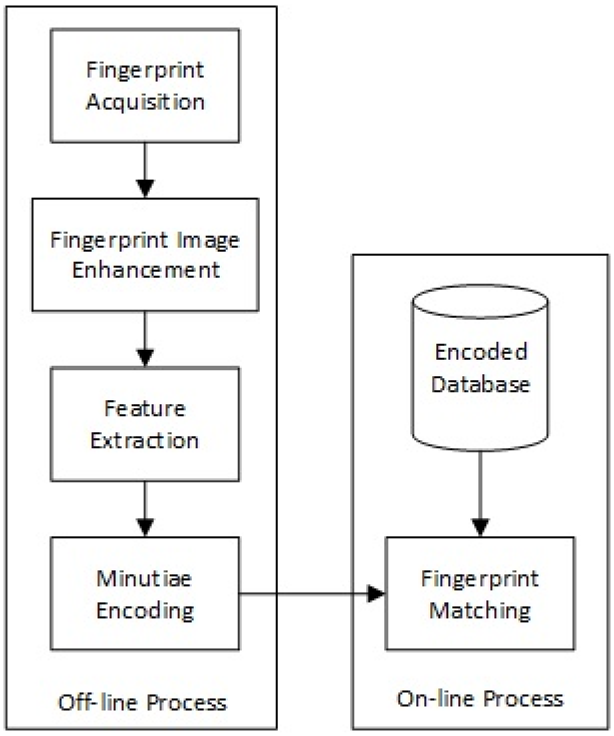


Figure 1.7: Modular Diagram of Fingerprint Identification Process

1.5 FPGA for Hardware acceleration

FPGA stands for Field Programmable Gate Array and as the name itself suggests, it has an array of logic elements that can be programmed by the designer. FPGAs architecture has also evolved from the start and moreover each FPGA might have a slightly different architecture. The two most prominent FPGA vendors are Xilinx and Altera. Each of these vendors further offer different families (variants) of FPGA devices. Mostly all FPGAs consist of programmable logic blocks (CLBs in Xilinx). These CLBs contain number of logic blocks called LUTs (Look up tables) or Logic units and Flip-flops/ registers [13]. FPGAs nowadays also come with dedicated block RAM units that can be used for storage. All of these are interconnected and user logic is mapped on to these hardware elements to give ASIC like performance. Initially FPGAs were used for ASIC prototyping but later because of their configurability and lower time to market, they have become an ideal choice for product development.

Due to an open field of programmable elements, FPGA is an ideal choice for implementing Image processing algorithms that often require parallel computations. Though CPU often do provide some level of parallelism, it cannot compete with the design freedom on an FPGA device. An FPGA can perform 10 times better compared to a CPU with four cores [14]. Algorithms that comprise of operations that are independent of each other and can be performed in parallel are ideal for FPGAs. GPUs with a lot of parallelism and extremely fast clock can compete with FPGA's performance. However, GPUs can give best results on simple algorithms in which all pixel operations can be executed in parallel [14]. FPGA offer extreme level of customization and a deterministic latency.

Biggest advantage of FPGAs over GPU is the low power consumption and customizable IOs. This makes FPGA a preferred choice when making a standalone application specific hardware. FPGAs now also come with 1000's of dedicated DSP slices that can easily handle complex mathematical operations. FPGAs are programmed with the help of hardware descriptive languages (HDL) like Verilog or VHDL.

1.6 Problem Statement

Fingerprint based biometric authentication systems are very common nowadays as fingerprint is the most reliable and cheap to work with biometric feature. It only requires a low cost sensor and fingerprint algorithms are most mature compared to the other biometrics. Depending upon the application, either a system requires verification or identification. Verification means to verify if the input fingerprint matches to the fingerprint samples stored in database for that particular person only. In verification process, the person is known. In case of identification, the person is not known and fingerprint is used to identify the person. This usually require searching through the database of millions of fingerprints. So the identification process requires a lot of time to provide a result, and hence it require a solution that can reduce time as much as possible.

As discussed earlier, FPGA naturally becomes an ideal candidate in order to perform the task of accelerating the fingerprint identification system. FPGAs are very commonly used in real-time implementation of algorithms belonging to different fields. The goal of this research is to provide a fast and accurate fingerprint matching design capable of performing multi-million matches per second.

1.7 Research Objective

All the steps involved around fingerprint biometric systems have been under study for quite a while and they are more mature compared to any other biometric feature. There is still room for improvement when it comes to the hardware acceleration of fingerprint algorithms. With the advancement of technology, performing high speed computations are possible with applications in every field of study. Main goals/ purpose of this research can be divided into two parts:

1. First objective is to study and analyze all the different stages involved in fingerprint base biometric systems. Goal is to study different techniques

used for fingerprint enhancement, feature extraction and matching and to see the combination that gives accurate matching results.

2. Second goal is to develop a FPGA powered system that is capable of performing ultra-fast fingerprint matching which can be used in several applications. Another related goal of the research is to optimize the HDL design and to find out that which design approach yield better results in terms of FPGA resource utilization and speed.

1.8 Research Approach

In this research initially fingerprint encoding and minutiae based fingerprint matching algorithm is implemented. By varying the parameters, performance of algorithm is tested. Next the encoding and matching algorithm are tweaked so that the implementation on hardware is efficient. Hardware implementation is done to make sure results on CPU and FPGA are identical. Two different approaches were used for designing the hardware architecture. One approach follows a rather conventional approach to design while the second approach is the proposed approach which is hardware resource efficient. In the last step, performance is evaluated with special attention towards possible matches per second. Figure below shows the steps involved in research methodology.

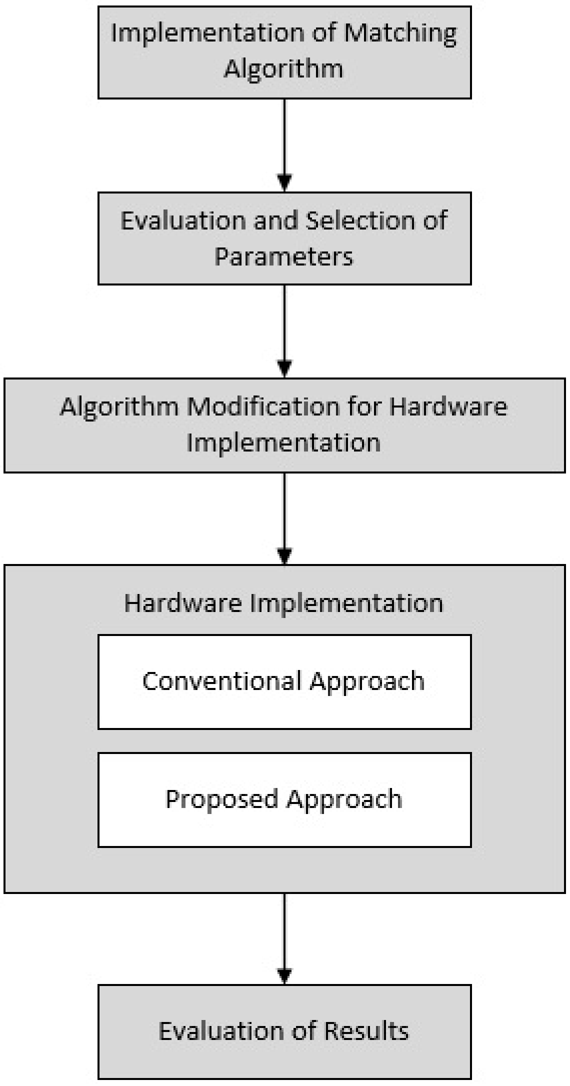

Implementation of Matching Algorithm
Evaluation and Selection of Parameters
Algorithm Modification for Hardware Implementation
Hardware Implementation
Conventional Approach
Proposed Approach
Evaluation of Results

FIGURE 1.8: Steps involved in Research Methodology

1.9 Organization of thesis

Thesis consists a total of six chapters. This chapter has already given basic introduction to the topic. It has also provided the problem statement that will be addressed in the later chapters. Chapter 2 is the literature review of different techniques used for handling different steps of a fingerprint based biometric system. Chapter 3 explains the finger print encoding and matching algorithm implemented on CPU in order to get accurate matching results. Chapter 4 and 5 provide two different HDL designs to perform fingerprint matching. The comparison of two approaches are also discussed and a better approach is identified. Chapter 6 takes the better of the two approaches to the selected hardware device and shows results of real-time performance of design. It discusses important points like how to test the design, matches per second and performance of design on other FPGA devices. Chapter 7 gives the conclusion and discusses what future work can be done to take this study forward.

1.10 Summary

This chapter gives an insight into the dissertation. Biometric features have been introduced in brief with their comparative analysis showing how fingerprint matching is better than other available biometrics. Remaining part of the chapter provides overview of the research objectives and goals. Subsequent chapter present the complete literature review.

Chapter 2

Literature Review

2.1 Overview

These days, finger print impression is a very significant biometric feature. It is being used for human identification in the banking sector and security applications. A lot of study has already been done on fingerprint authentication, but more work can be done on its large-scale implementation. Finger print methodology includes two steps namely encoding and matching. In the first step, an image is acquired and feature extraction is performed. Maio and Maltoni extracted minutiae from gray image generated by ridge line following algorithm [15]. Though the results were very good, but these techniques were not adopted due to there high computational cost [7].

Watson et al [16], at NIST (National Institute of Standards and Technology) developed an algorithm for minutiae extraction. The software developed was named as MINDTCT. The algorithm operates on each image by application of binary value to individual pixels based on the ridge and minutiae values of neighboring pixels. Afterwards minutia searching is applied for minutiae mining. The method was faster in minutiae extraction, but it faced many limitations due to which it was not adopted. Limitations included non-uniform gray scales, broken ridges and creases. Ultimately, the method failed.

Insufficiencies of minutiae points may result in wrong matching. The better approach is to include a preprocessing and enhancement stage combination before applying feature extraction. This stage is necessary before feature extraction as it will remove unwanted imperfections from the image. In encoding, after feature extraction, biometric code for the finger impression is computed and saved in database to be used as a reference for matching.

In the matching stage, sample of the reference image is loaded from database and features are matched. Steps involved in the whole process from image acquisition to print matching is shown in the following text.

2.2 Finger Print Acquisition

According to experts, ridges and minutiae points are important in fingerprint identification since these are mainly used in crime investigation and are acceptable by law as well. Minutia matching is considered the most reliable and accurate method of identification. For good identification of fingerprints, an image of 400dpi or more is required. This can be achieved through any image sensor available which can now acquire an image of resolution as low as 500dpi.

2.3 Pre-processing and Enhancements

In most of the matching cases, quality of input image ensures success and efficiency of the matching algorithm. To achieve better feature extraction, fingerprint impressions are passed through two stages: preprocessing and enhancement. The purpose of these stages is to remove any unwanted imperfections in the input image. The mentioned process is depicted in figure 2.1.

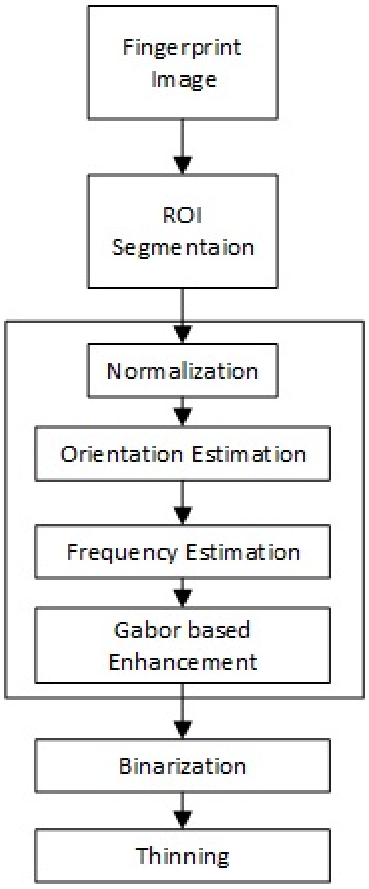


FIGURE 2.1: Preprocessing and Enhancement Technique

From figure 2.1, stages included in preprocessing and enhancement stages are region of interest mask recognition, binarization and thinning. At ROI (region of interest) mask segmentation stage, finger impression image in mined from background, based on the local variance of the image and directional coherence. After ROI, various techniques are utilized for image enhancement [17][18][19]. These techniques do enhance the fingerprint image however they may not improve the ridge structure which may be affected due to broken ridges, creases and scars. In order to enhance the ridges structure, contextual filtering may be applied.

Contextual filters are developed based on the ridges frequency and local orientation. In order to improve the ridge structure, frequency and orientation of ridges are computed in valid region and the contextual filtering is applied to enhance the image. Background knowledge based filtering is utilized to enhance the image quality.

Finally, the thinned image is used to extract minutiae. For this, image is first binarized and then thinned in the preprocessing stage.

2.3.1 Region of Interest (ROI) segmentation

This technique is used to mine the exact finger impression from the image and discarding rest of the contents of the image including background and other details. The same process can be utilized to extract the background as well. ROI is mined so that only designated features may be extracted at the time of feature extraction. For quick and fast decision making in ROI segmentation, a non-overlapping block technique may be utilized. ROI segmentation in finger print images of lower contrast can be implemented using gradient magnitude as suggested by Maio and Maltoni [15]. Figure 2.2 shows the experimental results of applying variance-based segmentation. Another technique for ROI segmentation was given by Bazen [20], which is based on coherence of ridge direction.

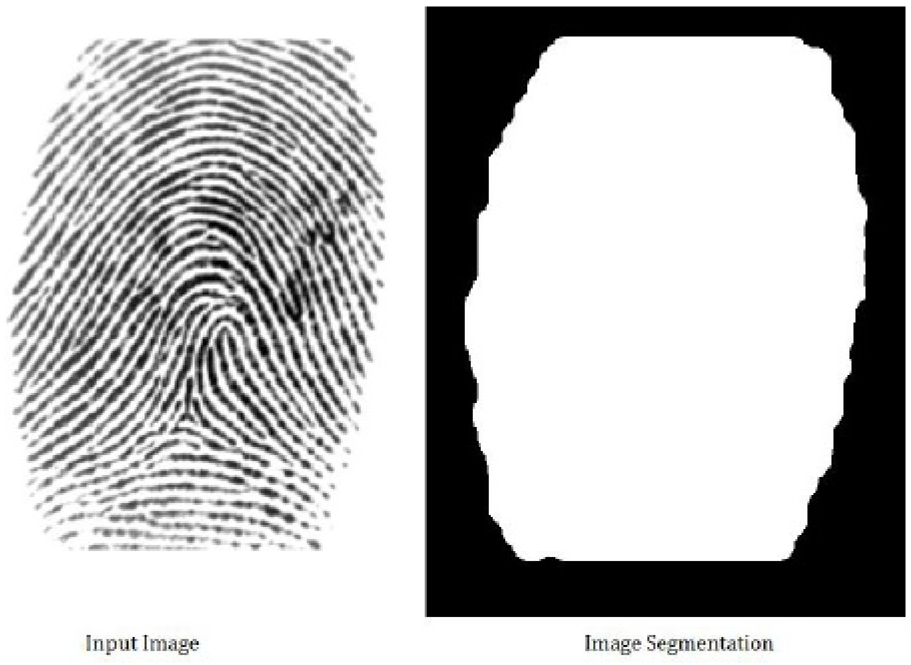


FIGURE 2.2: ROI segmentation based on Variance technique

2.3.2 Initial Enhancement

Image of the fingerprint acquired may have many imperfections which can include noise, sensor error, sweating due to ink density and others. These unwanted parameters affect the ridge structure and valley patterns. Kim [17] proposed a methodology to deal with these unwanted noise and illumination issue by normalizing the variance and means. In this normalization, gray level adjustments are ensured so that the image has the standard variance all over the image.

Greenberg et al. [18] suggested wiener filtering for the fingerprint image improvement. Willis [19] suggested technique of enhancing the image and removing unwanted scars and noises in frequency domain enhancement. In this methodology, Fourier transform of an image block is multiplied by the Kth power of the power spectrum, taking the inverse Fourier transform of the result computes the enhanced fingerprint image.

Chandra et al. [21] put forward a methodology of image enhancement by median filtering. Noise from the fingerprint image and overall scars can be eliminated using these enhancement techniques however, ridge structures and broken images cannot be recovered by these techniques. Fingerprint images are generally improved using contextual filtering which improves scars, broken ridges and creases. For this, ridge orientation and frequency images are computed before the filtering process.

2.3.3 Orientation Estimation

In this method, overriding path of underlying local ridge structure is created by preparing a positioning map or image O(x,y) [7]. This map is actually the orientation map. The orientation image O(i,j) shows the foremost orientation of ridge with horizontal axis in figure 2.3 [7].

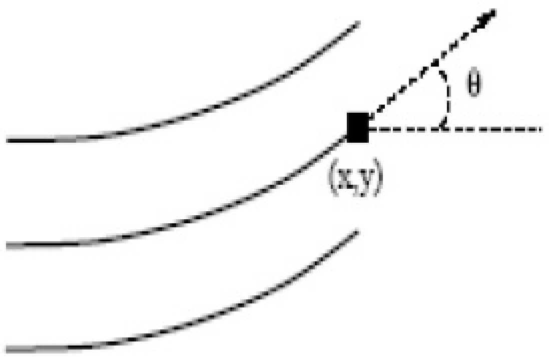


FIGURE 2.3: The angle of orientation

An effective orientation estimation is very necessary since the enhancement stage is purely dependent on it. Oriented image is also utilized in feature extraction and encoding stage. For low quality images, Turroni et al. [22] gave methodology based on adaptive alignment approximation. Common methodologies in use for estimation are the ones based on gradient computation [23]. In contrast, simple gradient methods are not utilized since sines and cosines are discontinuous and nonlinear. Bazen et al. [24], solved this issue by using principal component analysis of the gradient vector to compute the orientation.

Kass and witkin [25] proposed orientation estimation based on doubling the angles. Ratha et al. [26] put forward methodology of estimation by computing numerous gradients within 17x17 space to estimate the main ridge positioning.

2.3.4 Ridge frequency estimation

In this method, ridge frequency image is computed showing ridge density in local region. Hong et al. [2] suggested the method of frequency estimation in spatial domain within orientation window using x-signatures of ridges. This is shown in figure 2.4 [2].

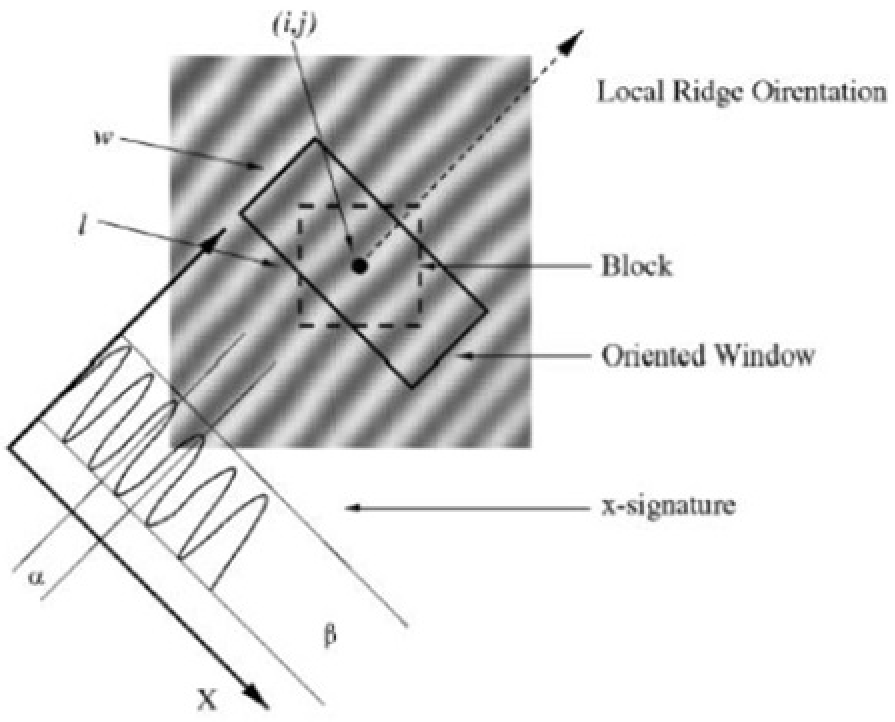


FIGURE 2.4: X-Signature based frequency estimation

In this technique, image is divided into multiple blocks. In the orientation window, ridges are divided by the distance between first and last ridge. This computes the frequency of the block. The methodology is different however it is not suitable for fingerprint images having noise. Chikkerur et al. [3], developed the algorithm for frequency and orientation estimation of fingerprint images based on short time Fourier transform (STFT)

2.3.5 Enhancement based on Contextual filtering

Filter parameters are computed adaptively based on the local characteristics of the image contents. Filters are tuned on the local frequency and orientation since the main concern is to enhance the ridge structure [27]. All enhancement techniques focus on enhancing the ridge structure. Filtering is applied along the ridge direction to remove the noise and to enhance the fingerprint image. O'Gorman and Nickerson [28] were first to propose the contextual filter. The method convolved each of the pixel with the filter which was bell shaped and extended along the ridge direction and designed for a constant ridge frequency as shown in figure 2.5 [28].

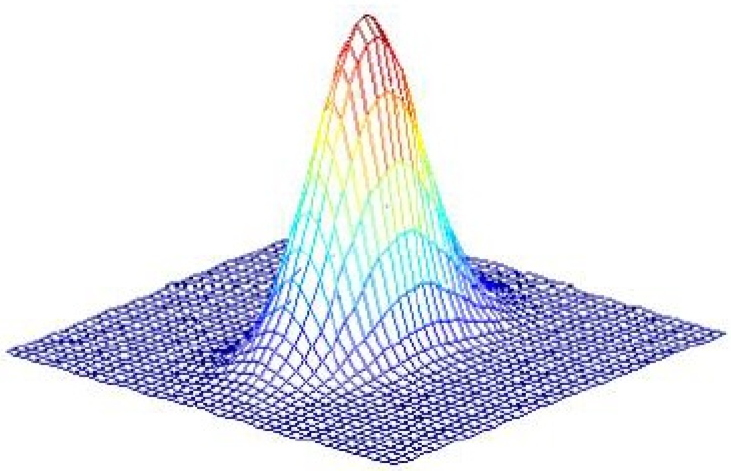


FIGURE 2.5: Contextual Filtering

In finger print image for diverse ridge orientations, a set of 16 rotated filters was derived from the suggested filter. A filter is selected from the set of filters whose orientation matches the orientation of the pixel to perform convolution. Filter is premeditated for a specific ridge frequency so it does not exploit the ridge pattern of the fingerprint image.

Hong et al. [2], proposed a technique for fingerprint image enhancement. The technique used the Gabor filters which is easy and straightforward. The technique is implemented by convolving each pixel with the Gabor filter, which is tuned with the local frequency and orientation. One flaw in the Gabor filtering is that it consists of a DC component, which reduces the distinction between ridge, and valley. This is why, enhancement done by this method is not very efficient.

Improvement in Gabor filter was proposed by Wang et al. [27] in the form of Log-Gabor filters. Figure 2.6 [27] below shows the 2-D log Gabor.

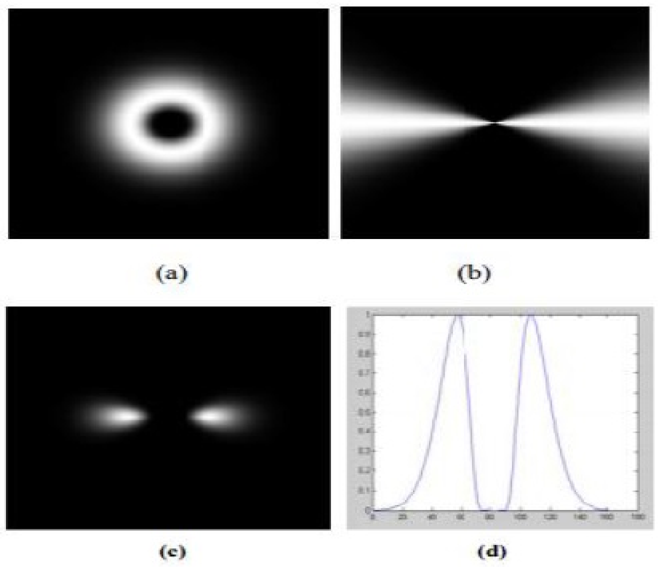


Figure 2.6: a)2D Log Band Pass Filter b) 2D log Fan Filter c) 2D log Gabor filter d) 1D frequency response of Log Gabor

Unlike Gabor filters, Log Gabor filter has no DC component, which makes it better than Gabor filter. In Log Gabor method, frequency and orientation is computed using the Gabor filter suggested by Hong et al. [2], however, enhancement is performed by Chikkurer [3] that utilizes localized frequency domain filtering. A two fold enhancement methodology was proposed by Mubeen et al. [29] that processes both in spatial and frequency domain. Block diagram for this approach is shown in figure 2.7 [29].

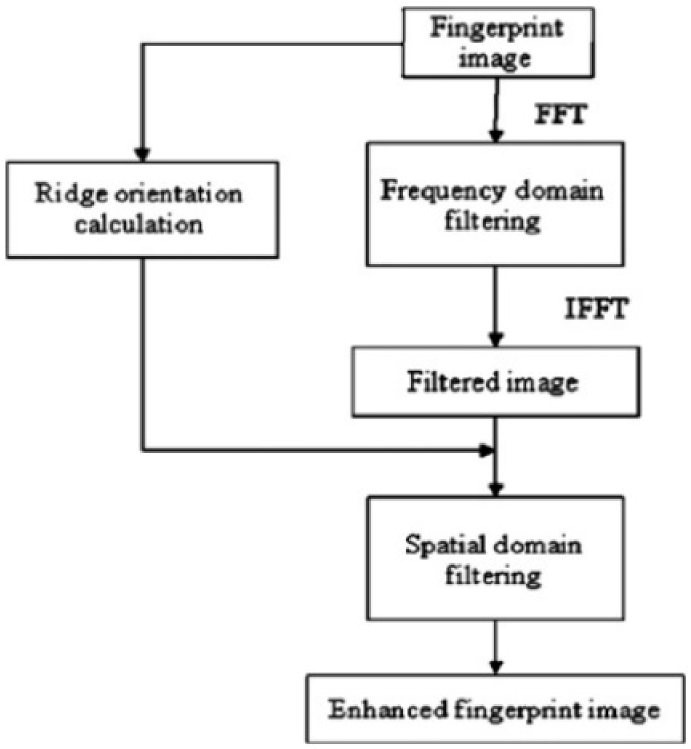

Fingerprint image
FFT
Ridge orientation calculation
Frequency domain filtering
IFFT
Filtered image
Spatial domain filtering
Enhanced fingerprint image

Figure 2.7: Two Fold Contextual Filtering

In this process, fingerprint image is first converted to frequency domain by applying Fast Fourier Transform (FFT) after which it is passed through various band pass filters and then converted back to spatial domain using inverse Fourier transform. In the spatial domain, filtering is applied to remove broken ridges and scars. For this purpose, ridge orientation map is developed and local smoothing is applied to all blocks of unified band-pass filtered image. Ultimately enhanced image is obtained by combining all the blocks.

2.3.6 Binarization and Thinning

After the enhancement and pre-processing, fingerprint image is binarized before feature extraction. The process converts gray image to binary image in which white color represents ridges and black color shows valleys.

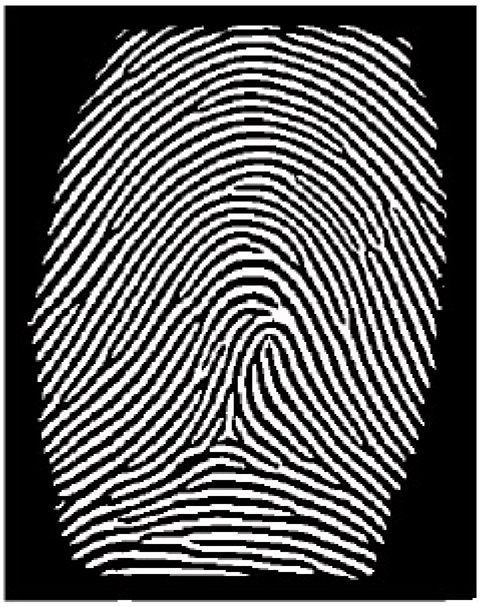


Figure 2.8: Binarization of Image

In binarization a threshold is selected and all pixels below that threshold are turned black and other are white. Selecting threshold is a cumbersome process which is resolved by image enhancement using filter banks. Filters decrease the illumination variation and reject all lower frequencies. Thus DC component can be used as a universal threshold for the image. Figure 2.8 shows the result of binarization.

Last stage before feature extraction is the thinning stage in which binarized image is morphologically changed so that the width of the ridges is reduced to single pixel [28, 29]. The process retains the ridges structure of the binary image. Finally, these thin images are used for feature extraction. Result of thinning process is shown in figure 2.9.

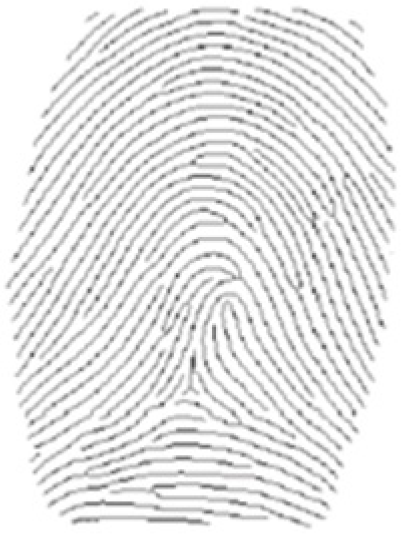


FIGURE 2.9: Thinning process results

2.4 Feature Extraction

Fingerprint impressions contain many features that are used in image matching. These features include minutiae, ridges, single core and others. Single core feature may be defined as "a core is located at the innermost re-curving ridge of the ridge pattern. If a ridge is considered as part of a circle, core is the core of that circle. Similarly an elliptical ridge will have focal point as its core." Core point is normally considered the reference point of fingerprint image and is extensively used in fingerprint impression matching [30][31][32].

Apart from this, presence of core point in partial fingerprint impression is not sure. Detection of core point in such images may lead to an error thus increasing the overall error of the system. The better way to match the print impression is to use minutiae which contains the most details. Francis Galton first detected minutiae points by observing the discontinuities of local ridges [33]. "Galton details" are referred to minutiae points. Minutiae has various types each one of them contain their own location and angle. Most common minutiae points are shown below for reference in figure 2.10 [7].

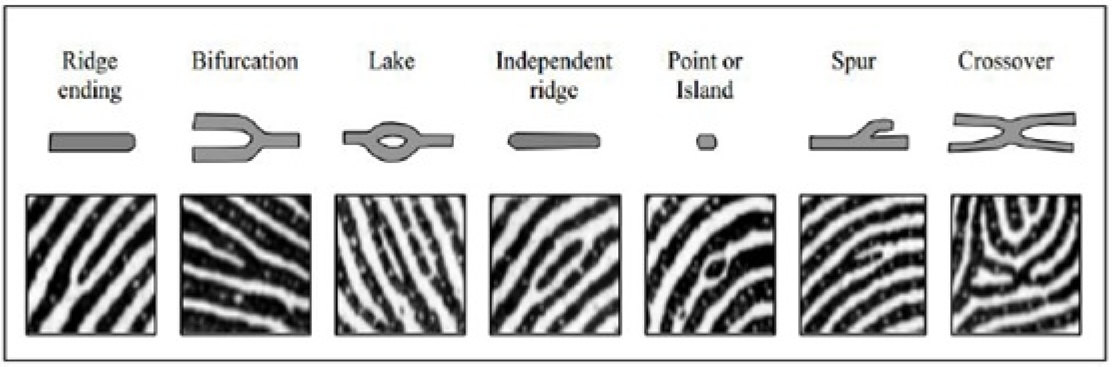


FIGURE 2.10: Most common Minutiae points

Out of the seven most common types shown in figure 2.10, ridge ending and ridge bifurcation types are mostly used since their detection is very stable and accurate as compared to other types. Other types of minutiae may be considered as combination of ridge ending and ridge bifurcation so there is no such need of extracting other features.

Minutiae points are extracted from thinned image which is obtained by thinning algorithm. However, errors are also produced in thinning algorithm which may create some false minutiae points by changing the ridge patters. Such false points are eliminated using post processing technique after minutiae detection [34][35][36]. The algorithm is described in subsequent details.

2.4.1 Minutiae Extraction

In this step of extracting minutiae point, local neighboring of each pixel is scanned in 3x3 window. During scanning of 8 neighboring pixels in clock wise direction, ridge is considered as 0 to 1 transition. If number of 0 to 1 transition are 3, pixel is considered as ridges bifurcation. If the transition number is equal to one only, the point is ridge ending. Figure 2.11 [7] illustrate ridge ending and bifurcation.

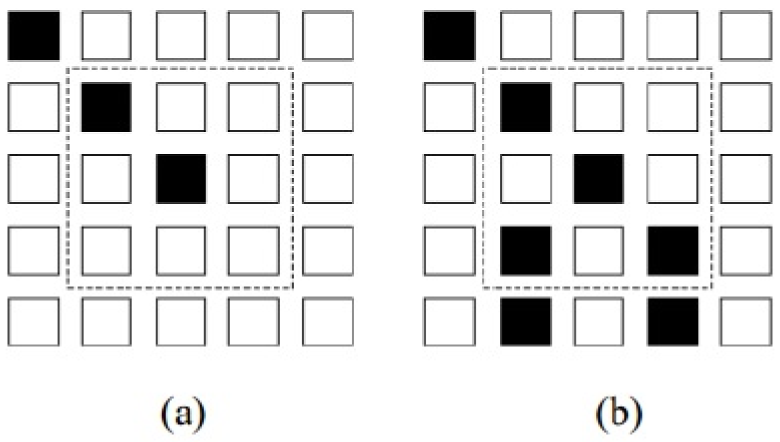


FIGURE 2.11: a) Ridge ending b) Ridge Bifurcation

2.4.2 False Minutiae Removal

False minutiae points created during thinning algorithm application are removed during post processing stage. Types of the false minutiae points are illustrated in figure 2.12 [7] below.

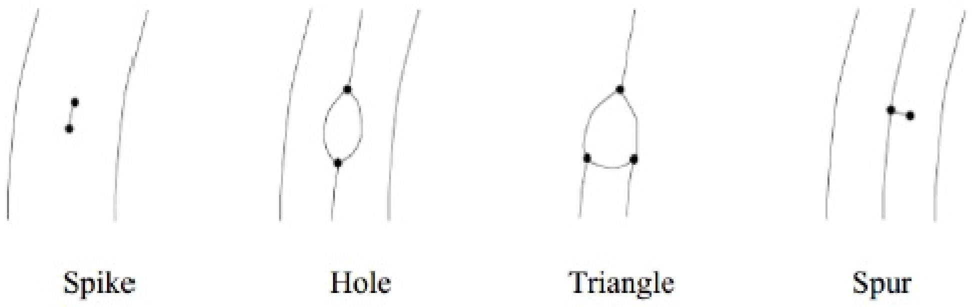


FIGURE 2.12: False Minutiae Points

Various techniques have been suggested to eliminate these types of false minutiae points. Marius Tico and Pauli Kusomanen [35] put forward a post processing techniques in which each candidate minutiae is scanned in a WxW window. Figure 2.13 [7] show output image with highlighted minutiae points.

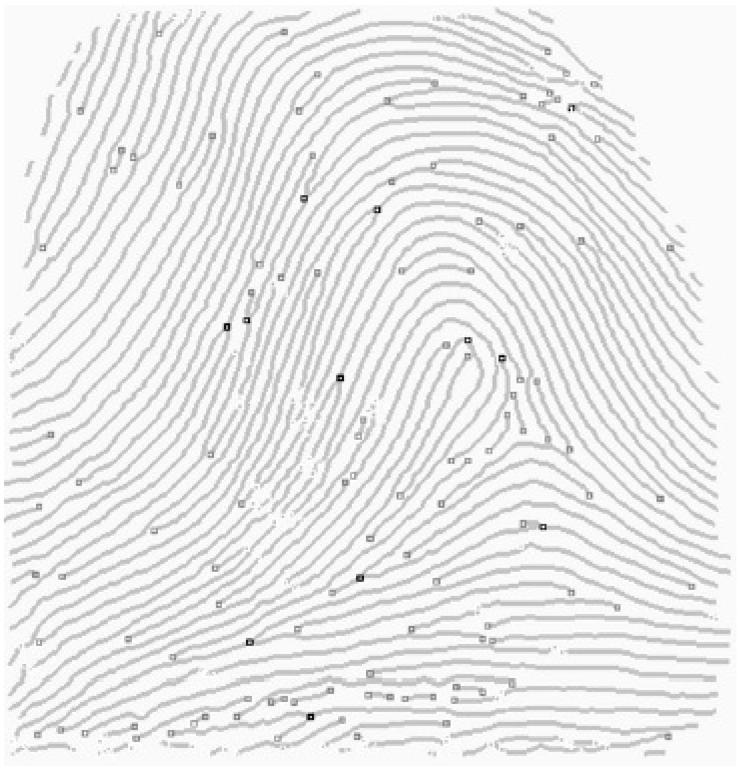


FIGURE 2.13: Result of minutiae extraction

A minutiae list is attained after minutiae withdrawal and false minutiae points exclusion. Each minutiae point is epitomized using its own angle and location. Finding the types of minutiae is very inaccurate and is thus not included in the features list.

2.5 Features Encoding and Matching

2.5.1 General Methodology of encoding

Suppose there is a fingerprint image having M minutiae points that are signified by a list (x_i, y_i, θ_i) where 'i' ranges from 1 to M. Each minutiae point is a set of (x,y,θ) and denoted by 'Mi'. Here position coordinates are signified by x and y while θ embodies the angle with respect to the x-axis. Each minutiae point is designated as reference point and then by using geometrical transformation, its angle and distance are measured with respect to its neighboring N minutia points. Figure 2.14 shows two minutiae points and their orientation.

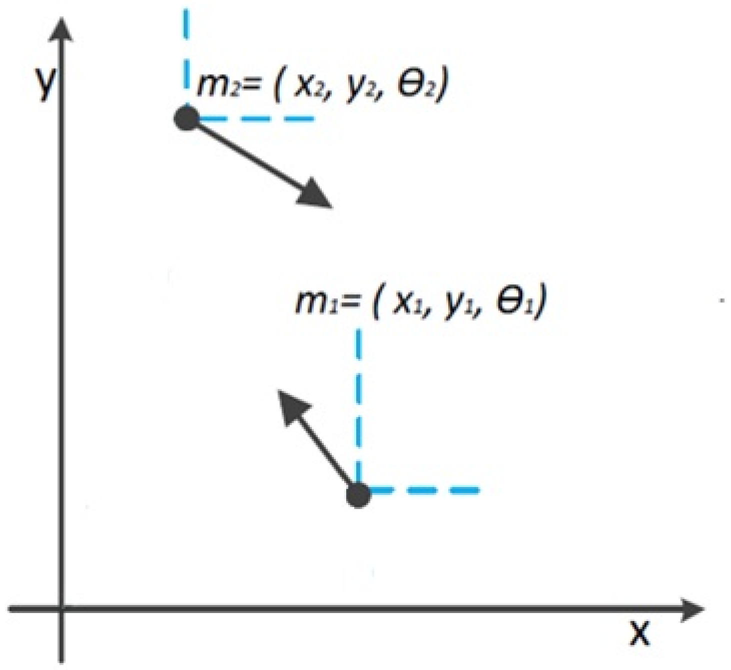


FIGURE 2.14: Fingerprint Minutiae Points representation

Minutiae is then represented as:

$$F = \{(\Delta x_1, \Delta y_1, \Delta\theta_1)^T, ..., (\Delta x_N, \Delta y_N, \Delta\theta_N)^T\}$$

Here F contains the set of neighboring minutiae points of selected reference minutia. Biometric codes are computed for the template image. A biometric code is the encoding value of such minutiae of fingerprint image. These computed codes are saved in database.

2.5.2 General Methodology of Matching

To match the finger print of candidate image with encoded database template, candidate image is processed through all the steps mentioned in the context i.e. from acquiring image to encoding till its biometric codes are computed. The computed codes are then compared with the biometric codes of the template image stored in the database. The biometric codes computed for the candidate image is compared with the codes of the template. Each minutia code is compared with all the minutia codes of the template. Since absence of any neighboring minutia in the template may change the order of the minutiae points, it is mandatory to compare the minutiae code of the candidate picture with every minutiae code in the template. If the reference minutiae code has n neighboring, comparison of this minutia with the template will need n2 comparisons. Greater the number of neighboring minutia, more will be the accuracy of the matching algorithm. Contrarily, number of comparisons and biometric code sizes also increase.

If M1 is the minutiae points in the template and M2 represents the same in candidate picture and each minutia point in candidate and template picture has n neighboring, complexity of the algorithm is computed to be M1xM2xn2. Number of successful minutia pairs matching adds to the overall score of similarity.

2.5.3 Contemporary Minutia encoding and matching technique

In fingerprint processing, encoding and matching methodologies may differ, but their basics remain the same. Encoding and matching is performed on each minutia based on its distances from neighboring minutia points, as put forward by Euclidean [37] [4] [6]. Jiang and Yau [4] suggested an encoding and matching mechanism based on minutia-triplet structure. The method is translational and rotational invariant. In its encoding phase, each minutia in template image is taken as reference and based on its distances with the neighboring minutiae, a polar vector is formed. Figure 2.15 [4] shows the encoding methodology suggested by jiang and yau where a minutia point $m_i(x_i, y_i, \theta_i)$ is encoded based on its distance from neighboring minutia points n_0 and n_1 which are represented as $n_0(x_{n0}, y_{n0}, \theta_{n0})$ and $n_1(x_{n1}, y_{n1}, \theta_{n1})$.

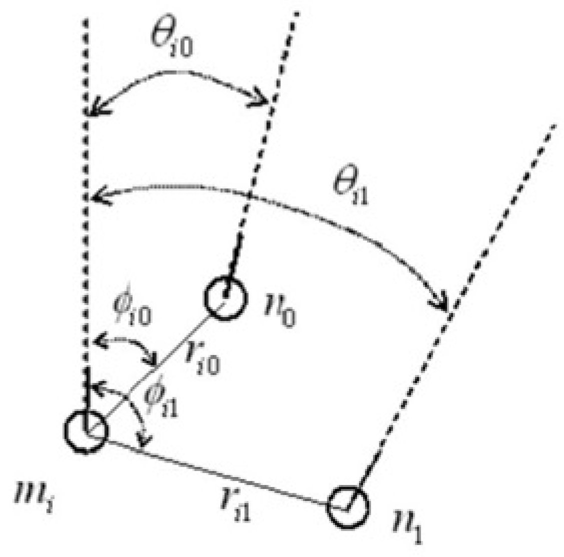


FIGURE 2.15: Fingerprint encoding method based on minutia triplet structure

Polar vector is represented as follows:

$$F_i = (r_{i0}, r_{i1}, \theta_{i0}, \theta_{i1}, \phi_{i0}, \phi_{i1}, n_{i0}, n_{i1}, t_i, t_0, t_1)^T$$

where r_{i0} *and* r_{i1}: distance of minutiae m_i with its neighboring minutia points n_0 and n_1 in polar coordinates.

θ_{i0} *and* θ_{i1}: difference between angle of reference minutiae and angles of neighboring minutiae n_0 and n_1.

ϕ_{i0} *and* ϕ_{i1}: Difference between reference minutiae orientation θ_i and orientation of line section which is connecting m_i and neighbors n_0 and n_1.

n_{i0} *and* n_{i1}: The number of ridges between minutiae m_i and its neighboring minutia n_0 and n_1.

Minutiae-triplet of "t_i, t_0, t_1" represents the type of minutiae point, e.g. ridge ending or bifurcation.

Jiang and Yau [4] proposed a matching mechanism too which calculates the weighted distance between the minutiae pairs in corresponding polar vectors. Similarity is computed between all possible minutia pairs. Number of successfully matched minutiae pairs add to the overall similarity score of the matching. Later a consolidation stage is applied to improve the matching done at local level.

Jea et al. [6] suggested a matching technique for partial matching of finger prints. He proposed the use of neural network to compute the normalized matching score within overlying region of aspirant and template fingerprint image.

Ratha et al. [38] proposed fingerprint matching techniques to find the false and lost minutiae points within the image on a fixed radius. This method of encoding is applied to each minutiae point and creates a vector containing encoding of all neighboring minutia points in that radius. The method increases the complexity of the algorithm however it creates a variable length of code for each minutiae reference point which helps in finding the false and lost points.

Tico et al. [39] suggested encoding and matching techniques for minutia based on the positioning of close juxtaposition of each minutia. Feng et al. [40], put forward a hybrid scheme of matching which performs matching on fixed radius and on orientation field.

Chikkerur [37] proposed an encoding scheme for fingerprint images bases on a graph of local structure. The structure was named as k-plet. In this method, encoding of selected minutiae point in performed in such a way that reference minutiae are taken as the central minutiae. The neighboring minutiae points fall in four quadrants. The encoding scheme retains the connectivity of minutiae and is less error sensitive in the presence of false or lost points. With k=4 and minutiae=3, k-plet example is illustrated in figure 2.16 [7].

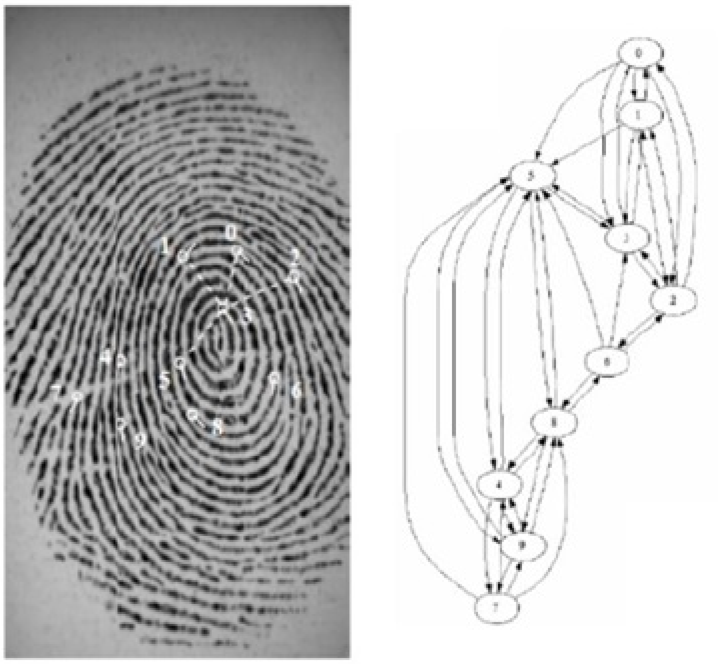


FIGURE 2.16: (left) k-plet local structure for minutiae 3 (right) graph based on k-plet local structure

Chikkerur [37] presented a matching scheme as well based on k-plet structures. The method uses the dynamic programming after which global search is performed by novel algorithm called CBFS (coupled breath first search). During the matching process, pair of nodes from both reference and template images are initialized for matching based on graph. During traversal of graphs, locally matched points are searched in CBFS manner. In the end, matching score is achieved for the pair of nodes.

Cappelli et al. [41] proposed Minutiae cylindrical code (MCC) for encoding and matching of the images on fixed length. In this scheme, a minutiae is selected as reference and 3D structure based on its neighboring is generated. However, matching process of this scheme is quite intensive and time consuming on normal computer due to its sequential computing. Cappelli et al. [42] put forward a method of using hash based indexing to speed up the matching process. Locality sensitive hashing (LSH) is based on MCC which maps minutiae points on to fixed length transformation invariant binary vector. Cappelli et al. [43] proposed optimized MCC method for GPU as well which uses the computation power of GPU based on careful design of data structure and memory transfers.

2.6 FPGA for Fingerprint authentication process

A lot of work has been done on implementing different steps of fingerprint authentication process on FPGA. Some of the research revolved on simple dedicated hardware based solution and some of the research was done to accelerate the processes. It was important to review all the previous work that has been done in this same direction. Lopez et al. [44] worked on implementing fingerprint minutiae extraction process on FPGA but the goal was not acceleration but rather providing a dedicated low cost solution. Soft core processor was used in design for the implementation. Khan et al. [45] implemented fingerprint image enhancement on FPGA. The research used anisotropic Gaussian filter in order to perform fingerprint enhancement. Ratha et al. [46] focused on acceleration of minutiae based fingerprint matching and used NIST-9 [47] database. Their research claimed that the FPGA design can achieve up to 110000 matches per second. Lindoso et al. [48] worked on correlation based fingerprint matching rather than minutiae based matching. The research was focused towards fast matching and less emphasis on accuracy. Matching two fingerprints took 0.145 milliseconds in the best scenario. Fons et al. [49] focused on implementing a low cost minutiae based matching system. The aim of research was not speed as it used FPGA + Microcontroller based design that could only give result of matching two fingerprint in 723.9 milliseconds. Lindoso et al. [50] did research on finding optimum minutiae based matching algorithm for FPGAs. The research involved comparing two different fingerprint matching algorithms on hardware. The two algorithms were Pre-alignment based and local structure based. Experimentation was done on personal data base of 56 images and Pre-alignment based algorithm was declared to be the faster as it could match two fingerprints in 0.14 milliseconds. Fons et al. [51] designed a complete automatic fingerprint-based authentication system using a single FPGA device. The research revolved around using the run-time reconfigurable hardware. All the stages that are part of automated fingerprint identification system were implemented and were sequentially used. 20.7 millisecond was the time taken for matching two images. Danese et al. [52] proposed an FPGA design using the embedded processor. The research was performed using FVC2002 [1] database and a speed of 0.66 milliseconds per match was achieved. The research predicted a time as low as 0.07 millisecond per match in case the design is run at a much higher clock speed. Jiang et al. [53] work was solely focused on getting maximum number of matches per second and the proposed work could perform 1.22 million matches per second. Drawback of the approach was that fingerprint minutiae alignment was not catered for. The approach might give high speed matching as the computations are quiet less but at the same time, it means inefficient matching results.

2.7 Summary

The chapter gives an insight in the literature review of fingerprint processing techniques. Contemporary methods of contextual based enhancement, feature extraction and efficient encoding and matching techniques have been discussed. In the subsequent chapter, encoding and matching algorithm implementation on CPU is discussed in detail.

Chapter 3

Minutiae Encoding and Matching algorithms in AFIS

In the initial stage, finger print encoding and matching algorithm is implemented on CPU. Purpose of this implementation is to first verify the algorithm's functionality on CPU before porting it to a hardware. After the basic implementation, downscaling is performed in order to make the algorithm more hardware friendly. This chapter discusses the implementation of finger print encoding and matching algorithm on CPU. Different steps prior to encoding and matching are covered in the literature review chapter. To summarize, after acquiring a finger print image, first the ROI is detected. After ROI segmentation images are enhanced to get rid of any distortions. Further binarization and thining is performed before the final feature extraction part. Minutiae points are extracted and each finger print is represented as a list of minutiae. Suppose if a finger print consists of M minutiae it will be represented in the form of:

$$MinutiaeList = ((x_1, y_1, \theta_1), (x_2, y_2, \theta_2), ..., (x_M, y_M, \theta_M))^T$$

Where M is the total number of minutiae in the image, x and y are the position of the particular minutiae and θ is its angle along x-axis.

3.1 Finger Print Encoding

Once the minutiae list is complete after feature extraction, encoding is performed. Each minutiae in the minutiae list is encoded with respect to its neighboring minutiae. The encoded minutiae list usually contain encoded features with N nearest neighbors. The three features that are calculated are r, θ and ϕ. To encode a minutiae in the minutiae list, first a list of N nearest neighbors is required.

$$Minutiae\_List = ((x_1, y_1, \theta_1), (x_2, y_2, \theta_2)......(x_M, y_M, \theta_M)) \tag{3.1}$$

$$Nearest\_neigh\_i = ((x_{i1}, y_{i1}, \theta_{i1}), (x_{i2}, y_{i2}, \theta_{i2})......(x_{iM}, y_{iM}, \theta_{iM})) \tag{3.2}$$

Where i is the minutiae number and N is the total number of nearest neighbors used for encoding purpose. Features r, θ and ϕ are calculated with respect to all N neighbors for all M minutiae.

Final encoded minutiae list can be represented as:

$$Encoded\_Minutia\_List = (M_1, M_2, M_3, ...M_M) \tag{3.3}$$

Where M is the total number of minutiae. Each minutiae M_i can be represented as:

$$M_i = ((r_{i1}, \theta_{i1}, \phi_{i1}), (r_{i2}, \theta_{i2}, \phi_{i2}), ...(r_{iN}, \theta_{iN}, \phi_{iN})) \tag{3.4}$$

Each feature r, θ and ϕ can be represented with the help of equations:

$$r_{ij} = \sqrt{(disX_{ij})^2 + (disY_{ij})^2} \tag{3.5}$$

$$\theta_{ij} = tan^{-1}\frac{(disY_{ij})}{(disX_{ij})} \tag{3.6}$$

$$\phi_{ij} = Minutiae\_list(\theta_i) - Nearest\_neigh\_i(\theta_{ij}) \tag{3.7}$$

Where

$$disX_{ij} = Minutiae\_list(x_i) - Nearest\_neigh\_i(x_{ij}) \tag{3.8}$$

$$disY_{ij} = Minutiae\_list(y_i) - Nearest\_neigh\_i(y_{ij}) \tag{3.9}$$

3.2 Finger Print Matching

One to one matching of two finger prints is done by matching the encoded minutiae list discussed in the previous section. In case of verification, input finger print is directly matched with the stored finger print(s) of the same person in data base. In case of identification, the input image is matched with all of the stored images in database. The match resulting in maximum score indicates who the fingerprint belong to. The input image is called a reference image and it is matched with the database images.

Matching the reference image to the database image require matching each minutiae of reference image with each minutiae of database image. Each minutiae itself is a list of encoded features (r,θ,ϕ) with reference to its neighboring minutiae points. So in order to match two minutiae, each encoded feature of minutiae1 is matched with each encoded feature of minutiae2.

For example if reference image has M1 number of minutiae and database image has M2 number of minutiae and we match N neighboring features of each minutiae, the total number of matches will be M1 x M2 x N x N x 3. Result of each minutiae matched is placed in a matrix of size M1 x M2 as shown in Figure 3.2.

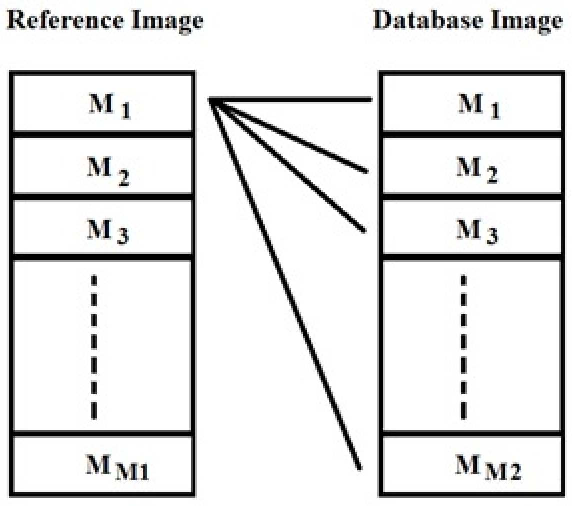
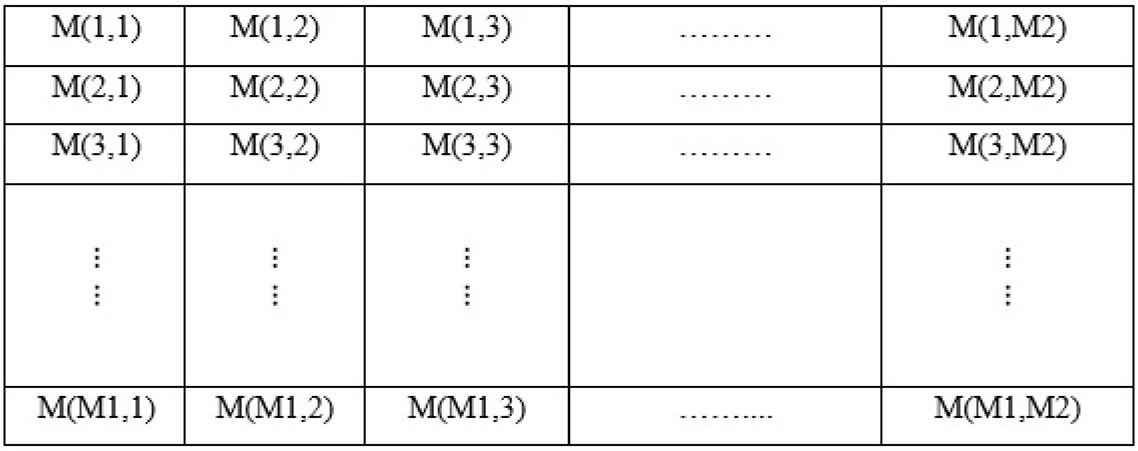


Figure 3.1: Fingerprint one on one matching



M(1,1)	M(1,2)	M(1,3)	....	M(1,M2)
M(2,1)	M(2,2)	M(2,3)	....	M(2,M2)
M(3,1)	M(3,2)	M(3,3)	....	M(3,M2)
⋮	⋮	⋮		⋮
M(M1,1)	M(M1,2)	M(M1,3)	....	M(M1,M2)

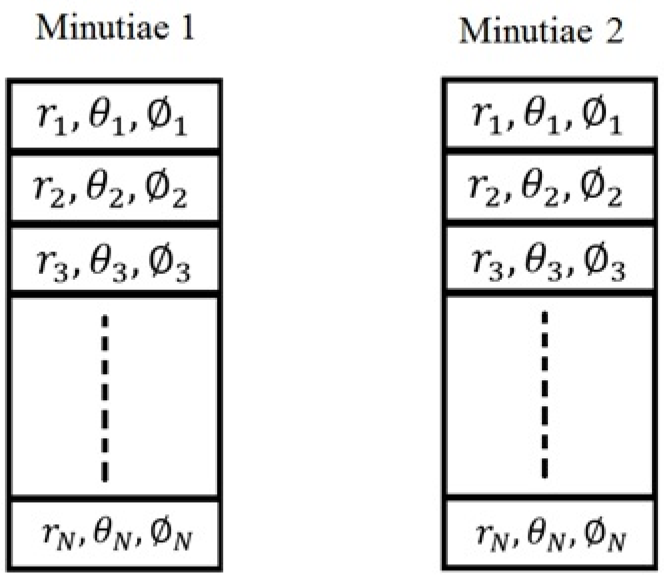
Figure 3.2: Minutiae match result matrix

Where M (1, 1) means the result of matching M1 Minutiae of Reference image with M1 Minutiae of Database image.

M (i, j) will be either 1 or 0. 1 meaning matched and 0 meaning not matched. Matching of Minutiae is explained below in the form of equations. If feature match score is greater than the neighbor threshold value, the two minutiae are considered to be a match.

$$M(i,j) = \begin{cases} 1 & if feat\_match\_score(i,j) \geq neigh\_thr \\ 0 & Otherwise \end{cases} \tag{3.10}$$

As discussed previously, minutiae matching require matching N neighbors so N x N values are compared and $feat\_match\_score(i,j)$ is incremented by 1 for every positive match.



Minutiae 1
Minutiae 2
$r_1, \theta_1, \emptyset_1$
$r_2, \theta_2, \emptyset_2$
$r_3, \theta_3, \emptyset_3$
$r_N, \theta_N, \emptyset_N$

FIGURE 3.3: Minutiae list with respect to N neighboring minutiae

$$feat\_match\_score(i,j) = \sum_{k=1}^{N} \sum_{l=1}^{N} Feat\_match(k,l) \tag{3.11}$$

In order to declare a match between features, all 3 features (r,θ,ϕ) must match. $feat_match(k,l)$ will also be a binary value, 1 meaning a match in all 3 features.

$$feat\_match(k,l) = R\_match(k,l) \ \&\& \ Theta\_match(k,l) \ \&\& \ Phi\_match(k,l) \tag{3.12}$$

Criteria for r, θ and ϕ to be declared match vary slightly.

$$R\_match(k,l) = abs(r_k - r_l) \ \leq \ R\_thr \tag{3.13}$$

$$Theta\_match(k,l) = (abs(\theta_k - \theta_l) \ \leq \ T\_thr) \ \| \ (abs(\theta_k - \theta_l) \ \geq \ 360 - T\_thr) \tag{3.14}$$

$$Phi\_match(k,l) = (abs(\phi_k - \phi_l) \ \leq \ P\_thr) \ \| \ (abs(\phi_k - \phi_l) \ \geq \ 360 - P\_thr) \tag{3.15}$$

R_thr, *T_thr* and *P_thr* are all different threshold values for the features r, θ and ϕ. The OR condition in case of Theta and Phi is because of the fact that angle is a cyclic value where 0 and 360 degrees are considered equal.

Once all the minutiae are matched and the score matrix is fully populated, score is calculated. Before score calculation the matrix require one last operation to make sure that any single minutiae should only have positive match with only 1 other minutiae. In case a minutiae had positive match with multiple minutiae, it will only be considered as single match while calculating final score. In order to do this, the score matrix is traversed and value of 1 is searched. In case a 1 is located at a location (i, j), the remaining values of i row and j column are forced to zero.

$$miutiae\_score = \sum_{i=1}^{M1} \sum_{j=1}^{M2} score\_matrix(i, j) \tag{3.16}$$

Percentage match between the two finger print images is calculated:

$$percentage\_match = \frac{minutiae\_score}{min(M1, M2)} * 100 \tag{3.17}$$

3.3 Downscaling for hardware

The algorithm for matching explained above is a perfect candidate for hardware implementation as we can use the parallel processing capabilities of the FPGA to perform all the matches in parallel. Implementation on hardware is discussed in the next section but some modifications in the algorithm are made in order to make it more suitable for hardware implementation.

The proposed design for FPGA benefit more if the threshold values *R_thr*, *T_thr* and *P_thr* are equal to '1'. This will decrease the time required to load reference image into the FPGA finger print matching block. Details of this are explained later in the chapter. Another added benefit of this approach is that in order to get good results at such lower threshold value, the r, θ and ϕ values must be downscaled. Downscaling results in reducing the number of bits required to hold values of r, θ and ϕ. Further the size of arithmetic units get smaller that benefits both area and speed.

r, θ, ϕ were downscaled to give a good result with R_thr, T_thr and $P\_thr = 1$. θ and ϕ which were earlier in the range 0 to 360 degrees were downscaled to the range of 0 to 45 degrees. The r, θ and ϕ matching equations (3.13, 3.14 and 3.15) in the previous section can now be written like:

$$R\_match(k,l) = abs(r_k - r_l) \ \le \ 1 \tag{3.18}$$

$$Theta\_match(k,l) = (abs(\theta_k - \theta_l) \ \le \ 1) \ \| \ (abs(\theta_k - \theta_l) \ \ge \ 45 - 1) \tag{3.19}$$

$$Phi\_match(k,l) = (abs(\phi_k - \phi_l) \ \le \ 1) \ \| \ (abs(\phi_k - \phi_l) \ \ge \ 45 - 1) \tag{3.20}$$

3.4 Algorithm Performance

Efficiency of finger print matching algorithm is calculated in terms of EER. The testing protocol used for experimentation is FVC [1]. Equal error rate (EER) is used as an indicator to evaluate performance. The data base used for carrying out experimentation is FVC2002 DBI-A [1]. There are total of 100 finger print subject and each has 8 samples. In order to calculate EER, first the False Non-Matching Rate (FNMR) is computed. This is also known as genuine matching where samples of a same subject are matched only with each other. For 8 samples, there will be 28 different 1 on 1 matches. Total number of matches required are:

$$8C2 \ * \ 100 = 2800$$

After this, imposter matching is performed and false matching rate (FMR) is computed. In this case, only first sample of each 100 subjects are matched with each other. With total of 100 samples and performing 1 on 1 matches, there are total of 100C2 combinations in which matching will take place. Total number of matches for imposter matching are:

$$100C2 \;=\; 4950$$

EER was calculated for different values of 'n' nearest neighbors matched and 'neighbor threshold'. Also the threshold values for feature matching were also changed in order to get to the lowest EER value. The table 3.1 below shows EER values for the best combination of $'R\_thr'$, $'T\_thr'$ and $'P\_thr'$ which were 6, 9 and 9 respectively.

Table 3.1: EER Calculation by varying number of neighbors matches and neigh thresh

	N-Neighbors matched							
neigh_thresh	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
1	18.692	20.726	22.334	25.094	27.006	29.447	30.566	32.485
2	3.4416	2.9491	3.1169	3.4719	3.5433	4.0660	4.3503	5.2031
3	3.7422	2.4847	1.9798	1.8579	1.7157	1.7561	1.6777	1.8858
4	15.091	5.7011	2.5902	1.7457	1.8757	1.6016	1.3375	1.4214
5	X	21.671	10.431	5.0916	2.7624	2.0451	1.7471	1.7379
6	X	X	76.964	15.099	8.6172	5.3137	3.7065	2.7446
7	X	X	X	82.053	19.438	12.099	8.2422	5.9030
8	X	X	X	X	85.446	24.153	15.974	11.010
9	X	X	X	X	X	88.482	78.553	19.760
10	X	X	X	X	X	X	90.678	81.803

3.4.1 Effect of downscaling on algorithm's efficiency

As mentioned earlier, all the values of (r,θ, ϕ) features were downscaled in order to make algorithm more hardware friendly. $'R\_thr'$, $'T\_thr'$ and $'P\_thr'$ were all fixed to value of 1 and the downscaling was performed to reduce the r, θ, ϕ values to 6-bits. 'r' was downscaled to be in the range of 0 to 63. Best EER was achieved by downscaling θ, ϕ values to be in range of 0 to 45 degrees. Each downscaling case was tested by trying all the different combinations of 'n' nearest neighbors matched and 'neighbor threshold'. Table 3.2 shows EER achieved after downscaling. Best results were achieved for 9 neighbors with *'neigh_thresh'* set at 4.

Table 3.2: EER Calculations after downscaling features

	N-Neighbors matched					
neigh_thresh	**5**	**6**	**7**	**8**	**9**	**10**
1	26.515	29.453	31.096	33.279	36.654	36.598
2	4.8427	5.3274	5.6854	6.4823	7.4975	8.5996
3	2.3712	2.3223	2.1499	2.2866	2.7208	2.9592
4	3.7642	2.5335	1.9316	2.0047	1.6575	1.7662
5	17.510	6.7547	2.9582	2.2372	2.5025	1.9565
6	X	22.117	10.064	5.0994	3.2951	2.4257
7	X	X	77.000	14.081	7.9565	4.885
8	X	X	X	80.125	18.242	10.581
9	X	X	X	X	82.750	22.046

Chapter 4

Hardware Design for Minutiae Matching

The algorithm used for finger print matching is an ideal algorithm that should be ported to hardware. Too many computations are required in order to match two fingerprints. There were total of M1 x M2 x N x N x 3 values that are compared and score is calculated. In software a sequential approach is used by using multiple loops. Though some level of parallelism can be implemented on a CPU using multithreading but CPU cannot compete with the parallel computation capability of an FPGA. FPGA provide an open field of logic arrays that give control to the developer. Developer is free to use the available resources to achieve what the application requires.

4.1 Overview

For this study, two different approaches are proposed and compared for implementation on FPGA. Both approaches make use of resources and perform all matches in parallel. Both approaches are pipelined to give high speed matching as matches per second is a very important measure for performance. The difference between the two approaches is:

- One approach follow the conventional style of matching that require arithmetic units (This Chapter).
- Second proposed approach make use of DRAMs as look up tables to perform matching (Discussed in next chapter).

Research methodology involves implementing and testing both approaches and comparing the two approaches in terms of hardware resource utilization and possible matches per second.

DRAM based RTL design is explained in the next chapter. This chapter explains the proposed RTL design that follows the conventional style of implementation. In this style of implementation, features are matched using arithmetic operations. Two values are subtracted and the difference is compared with the threshold. A bottom up approach is used to explain the functionality of all the blocks before reaching the top module. Further protocol is discussed which is adapted for feeding reference and database images to finger print matching unit.

4.2 Single neighbor matching

This block is used to match single feature of one minutiae with single feature of the other minutiae. This module only takes input and provides output declaring if the two feature sets match or not. Block design for single neighbor matcher is shown in figure 4.1.

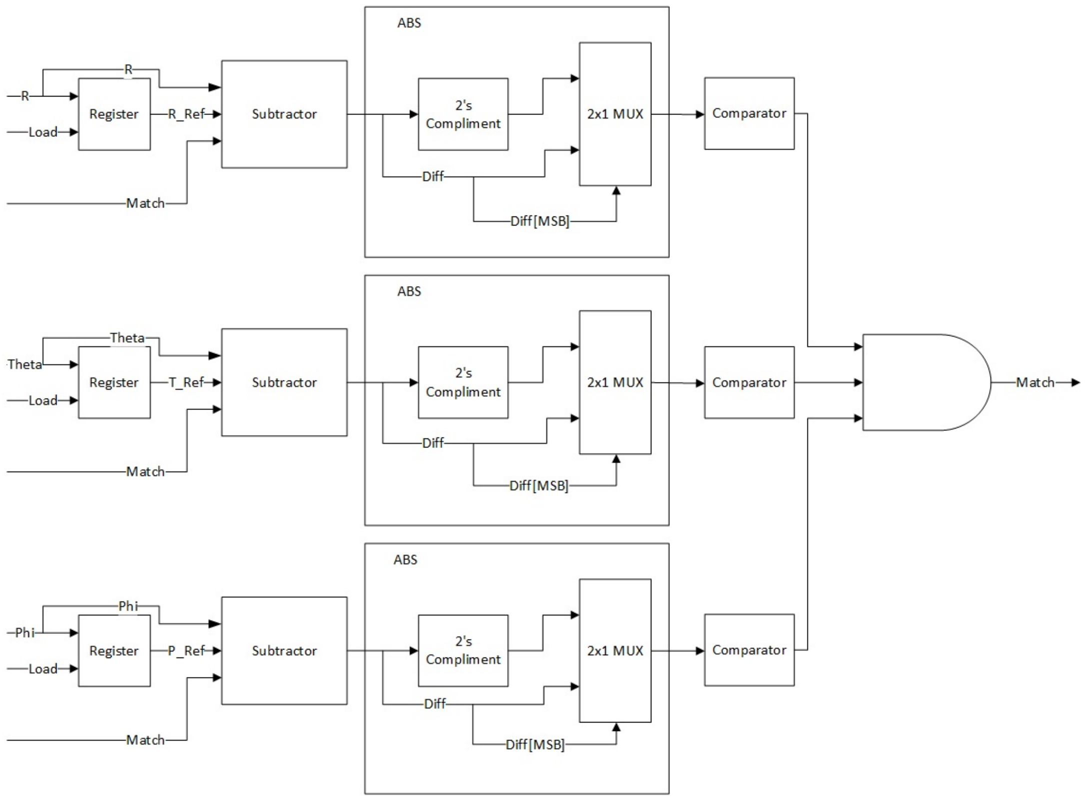


FIGURE 4.1: Single neighbor matcher block

As already discussed a single feature set consists of three values (R, Theta and Phi). In order to perform matching first the block expects reference values to be loaded. R, theta and phi are loaded into a register using "load" write enable signal. In order to match the loaded value with input value, the input value is provided on the same bus with "match" signal asserted. The two values are subtracted to find the difference between them.

Absolute operation doesn't exist in FPGA directly so it is implemented using two's complement and a multiplexer. After subtraction, the subtractor output is observed. Most significant bit of output tells if it is a negative value or a positive value. If the subtractor's output is positive, the result is passed directly to comparator. If the subtractor's output is negative, two's complement of the value is fed to comparator. Two's complement of a value is taken by inverting all the bits and adding 1 to it.

$$2's\_complement(value) \ = \sim(value) + 1 \tag{4.1}$$

$$abs(value) = \begin{cases} value, & if\ MSB\ of\ value = 0 \\ 2's\_complement(value), & if\ MSB\ of\ value = 1 \end{cases} \tag{4.2}$$

Comparator operation is also performed in an optimized fashion. As a general implementation rule, the greater than (>) and less than (<) checks should be avoided as much as possible. As explained earlier, the comparator needs to check for " ≤ 1" and " ≥ 44" conditions (see equations 3.18, 3.19 and 3.20). Value that is needed to be checked is available in 6-bits, 5 being the most significant bit and 0 being the least significant bit. The " ≤ 1" condition was handled by checking bits 5 down to 1. If any of these bits is 1, it means the value is greater than 1. Similarly " ≥ 44" condition was handled by checking the same bits. If value of most significant 5 bits is equal to 22, it means that complete value is either 44 or 45. As absolute difference of value that are in the range of 0 to 45 cannot be greater than 45 so if the most significant 5 bits represent 22, the " ≥ 44" check is satisfied.

$$comparator(abs\_diff\_R) = \begin{cases} 1, & if\ abs\_diff\_R[5:1] = 0 \\ 0, & Otherwise \end{cases} \tag{4.3}$$

$$comparator(abs\_diff\_Theta) = \begin{cases} 1, & if\ abs\_diff\_Theta[5:1] = 0\ or\ 22 \\ 0, & Otherwise \end{cases} \tag{4.4}$$

$$comparator(abs\_diff\_Phi) = \begin{cases} 1, & if\ abs\_diff\_Phi[5:1] = 0\ or\ 22 \\ 0, & Otherwise \end{cases} \tag{4.5}$$

At the end the two feature sets are declared to be matching, if all three comparators result is 1.

4.3 Minutiae matching

As previously discussed, each minutiae is represented in a form of list of features (R, theta and phi) with reference to its neighboring minutiae. So in order to match two minutiae, all the neighboring features of minutiae 1 are matched with all neighboring features of minutiae 2. There are nine neighboring feature for each minutiae to be matched in hardware. The block diagram of minutiae matching module in figure 4.2 demonstrates the design.

There are nine single neighbor matcher modules instantiated inside the minutiae matching module. While loading, the nine features are loaded sequentially. A de multiplexer is used in order to select where to load the feature values. "Neighbor Sel" can have a value from 0 to 8. While matching, a single set of R, theta and Phi is matched with all 9 loaded values in parallel. As a result, after inputting only nine values for matching, result of 9x9 matches can be calculated.

While matching, the database minutiae is fed to the minutiae matcher sequentially. One feature vector is matched with nine features of reference minutiae. The result of these nine matches is a nine bit vector. Bit adder adds all the bits of this vector gives us number of reference features that match with the input feature. The remaining eight features are also matched in similar fashion. Output of bit adder is fed to an accumulator which can give us the score of 9x9 matches.

Once the result for all 9x9 neighbors matching is complete, the score is fed to a comparator. After experimentation on algorithm it is established that the comparator should declare a match in minutiae if the score is greater than or equal to 4. Similar approach is used to implement this comparison as was done in the single neighbor matcher. The 9x9 result is represented by a 7 bit vector (6 down to 0) and if any bit from 6 to 2 is high, it means the value is greater than or equal to 4.

$$comparator(Sum\_9x9) = \begin{cases} 1, & if\ OR(Sum\_9x9[6:2]) = 1 \\ 0, & Otherwise \end{cases} \tag{4.6}$$

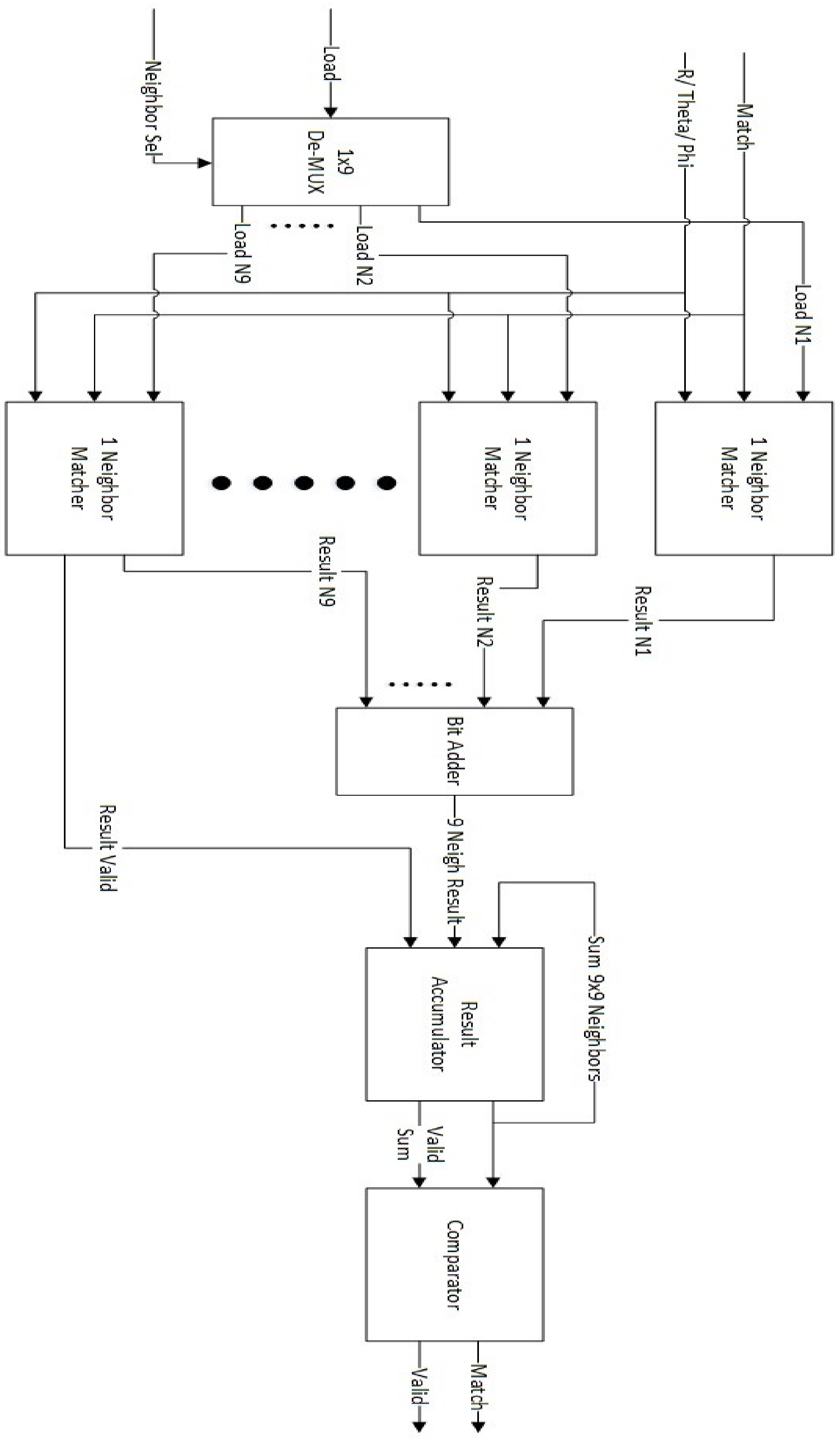


Figure 4.2: Minutiae matching unit

4.4 Matching module Top

In order to match reference and database finger print images, all minutiae must be matched. There will be M1 x M2 minutiae matches before computing the final score. The top level module for fingerprint matching is demonstrated with the help of figure 4.3.

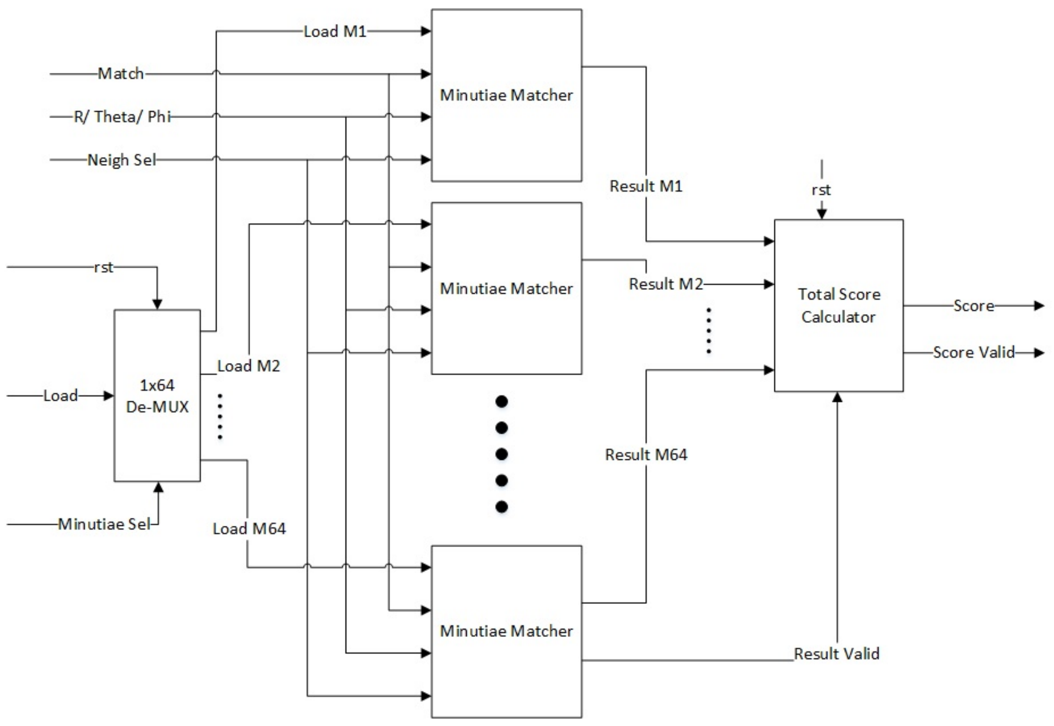


Figure 4.3: Top level fingerprint matcher

The number of minutiae can vary and each finger print will have different minutiae count. However for the sake of hardware implementation an upper limit is set. The maximum number of minutiae for any finger print image is 64. For this reason, the lop level matching module has 64 minutiae matchers. A special purpose de multiplexer is used to forward the load enable signal to the appropriate minutiae matcher. When Reset and load both are asserted, all the matchers are loaded at once. This case is used to reset all the reference values inside matchers. The result from 64 minutiae matchers form a 64 bit vector. This vector along with a valid signal is passed to the total score calculator. Total score calculator calculates the total number of matching minutiae among the two fingerprint templates. Before feeding another database image for matching, the total score is reset.

4.4.1 Total score calculator

The total score calculator block gets result from minutiae matching. Single minutiae of database image is matched with all minutiae of reference image at once. The result is provided to total score calculator. Once all M2 minutiae of database image are matched, the total score calculator provides the matching score.

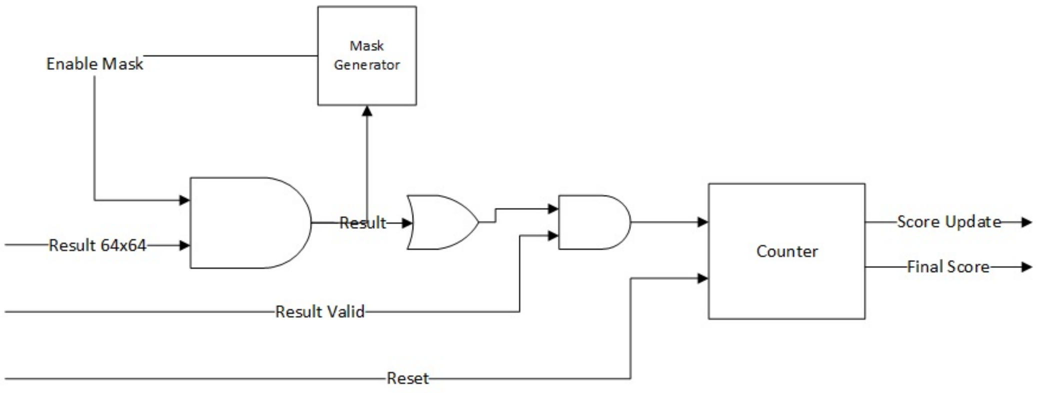
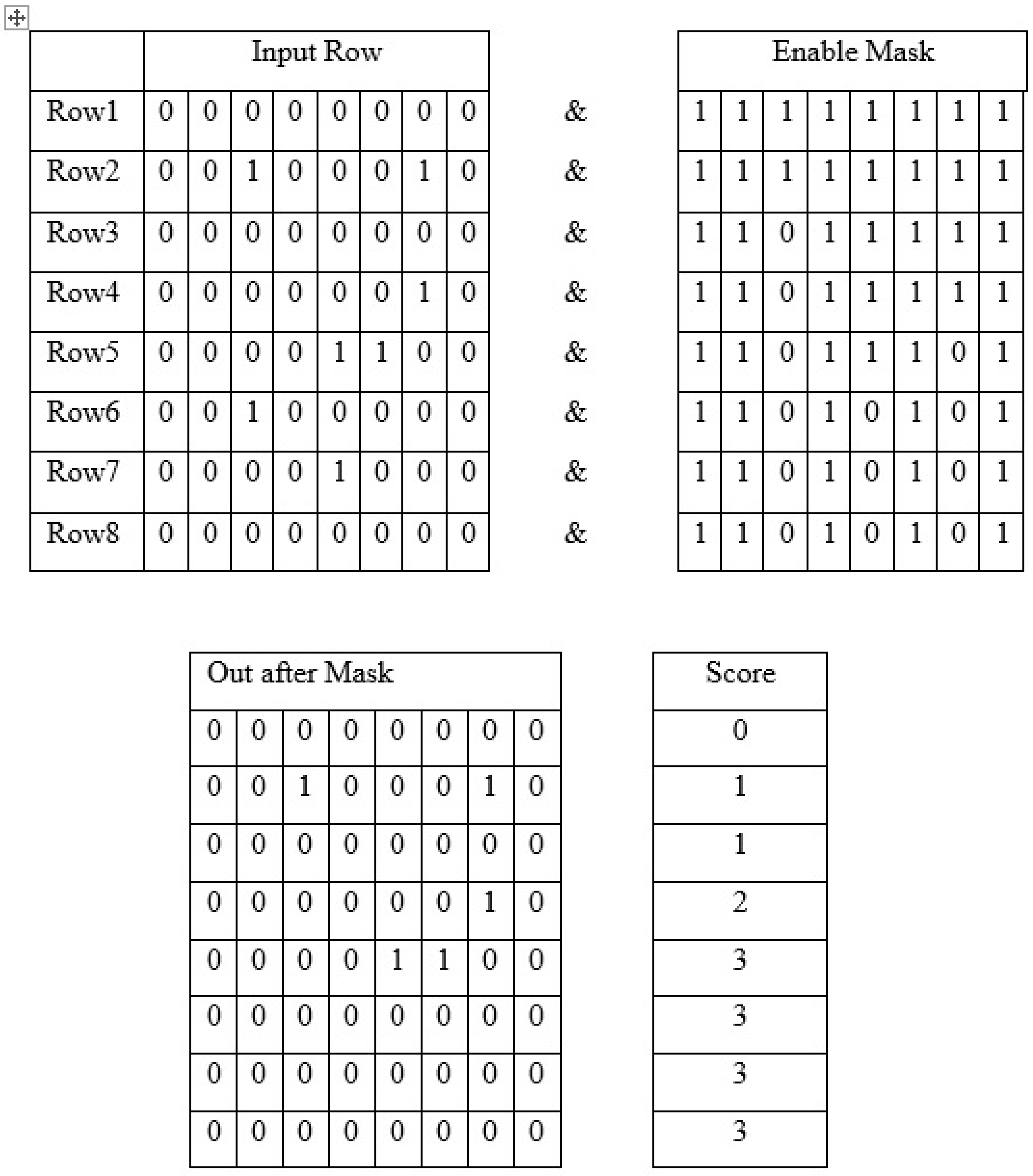


FIGURE 4.4: Total score calculator

As mentioned in section 3.2 (Finger print matching), before calculating the score, it is mandatory to make sure that a one minutiae of database image gives a positive match with only one minutiae of reference image. Implementation of this step is not straight forward in FPGA as not the whole matrix is available at once. A single row arrives into the block at one time so the row operation is simple. No matter how many bits out of the 64 bit register are one, they are counted only once using the OR operator. Second task is to ignore any other positive match at the column location of the first match. This task is handled using a mask. Initially all bits of the 64 bit mask are high. Now suppose that a minutiae of data base image matched with the 3rd minutiae of reference image. In this case, the 3rd bit of mask is changed to 0. So in future even if any other minutiae of data base image matches with the 3rd minutiae of reference image, it will become zero after the enable mask is applied to it. Let's consider an example where both M1 and M2 are equal to 8. Example is shown in figure 4.5.



	Input Row									Enable Mask							
Row1	0	0	0	0	0	0	0	0	&	1	1	1	1	1	1	1	1
Row2	0	0	1	0	0	0	1	0	&	1	1	1	1	1	1	1	1
Row3	0	0	0	0	0	0	0	0	&	1	1	0	1	1	1	1	1
Row4	0	0	0	0	0	0	1	0	&	1	1	0	1	1	1	1	1
Row5	0	0	0	0	1	1	0	0	&	1	1	0	1	1	1	0	1
Row6	0	0	1	0	0	0	0	0	&	1	1	0	1	0	1	0	1
Row7	0	0	0	0	1	0	0	0	&	1	1	0	1	0	1	0	1
Row8	0	0	0	0	0	0	0	0	&	1	1	0	1	0	1	0	1

Out after Mask								Score
0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	1
0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0	2
0	0	0	0	1	1	0	0	3
0	0	0	0	0	0	0	0	3
0	0	0	0	0	0	0	0	3
0	0	0	0	0	0	0	0	3

FIGURE 4.5: Example to ignore multiple positive minutiae matches

The 1's at row 6 and 7 are ignored because they lie in a column location that has already had a match so the enable mask disables them.

4.5 Software simulation test

After implementation, it is important to verify if the FPGA RTL design is giving accurate results or not. Initial test is to run software simulations and comparing simulation results with the results in CPU. Xilinx Vivado software has simulator in order to perform software simulation. Vivado allows the user to generate a blank test fixture with top level design module instantiated. A text fixture consists on an instance of module to be tested which is normally called unit under test (UUT). All the rest of the logic is to provide, UUT with the required inputs and to observe or compare its outputs. The code inside the test fixture doesn't need to be synthesizable as it will never be ported to a hardware. Figure 4.6 shows the concept in visual form.

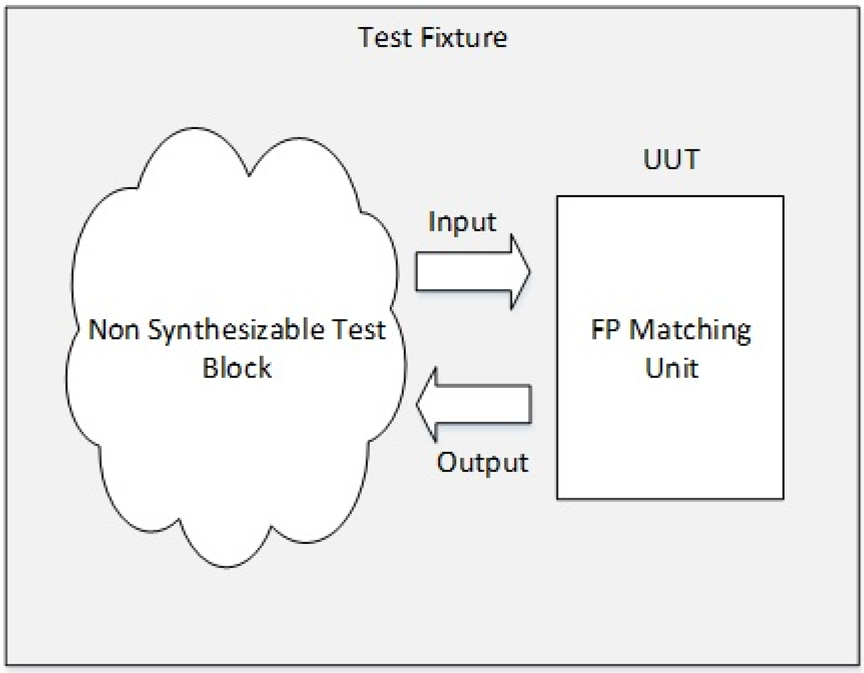
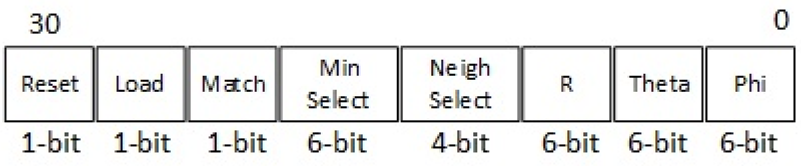


FIGURE 4.6: Test Fixture design concept

Data to be fed to UUT was taken from the CPU code and was arranged in the format required. For the design, a 31 bit data bus was used to provide data to UUT. The 31 bit data bus comprised of all the control and data signals required as input to UUT. Detailed purpose of each entry on bus has already been explained in the earlier sections of the chapter.



In case of loading a reference image into the design, Load (bit 29) will be high. R, theta and Phi will be holing 6-bit values of the feature while Min select and Neigh select are used to specify which neighbor of which minutiae is being loaded. Match bit is ignored when Load bit is high.

While matching, Reset and Load bits must be 0. Match will be 1 and R, theta and Phi will hold the feature set to be matched. Min select and neigh select will be ignored during matching as matching is performed in parallel on all the minutiae and all neighbors. After sending the whole data base image to the design, score provides the total number of minutiae matched between the two images.

In order to clear the loaded reference image from the registers, Reset and Load are both set to 1. All minutiae are loaded at once and Neigh select is set to 15 which means all neighbors will be loaded too. Only a single cycle is required to clear the complete reference image.

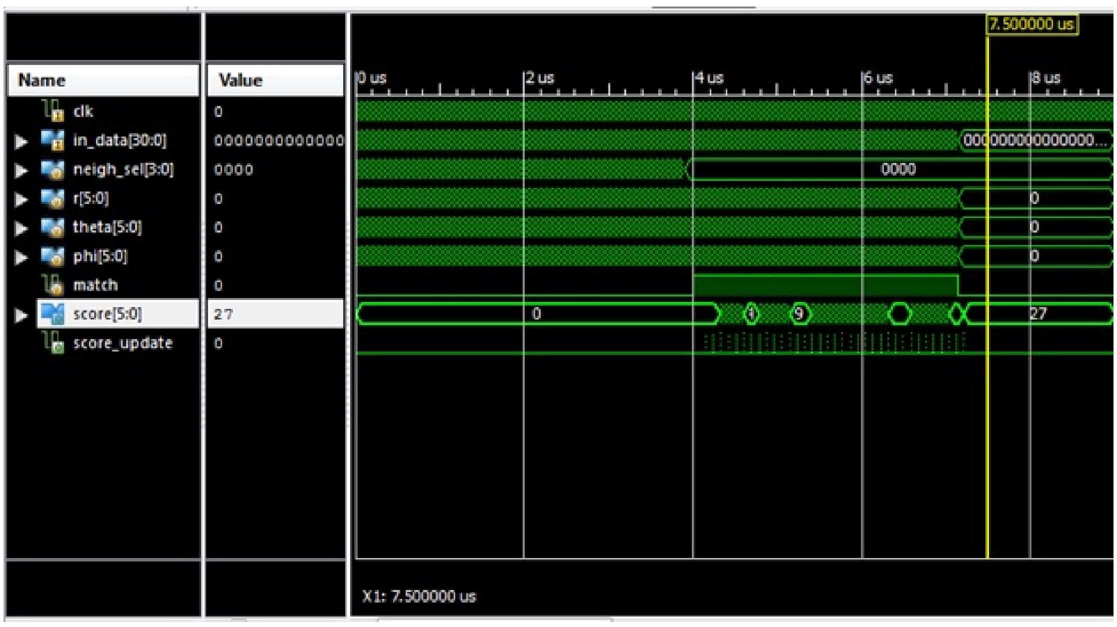


FIGURE 4.7: Simulation testing of RTL design

4.6 Hardware Area and Speed report

After verifying the design accuracy, design was synthesized in order to get area and speed estimate.

4.6.1 Resource utilization

Synthesis report provides the detailed resource utilization by the design on selected FPGA. FPGA used for the study was Xilinx Kintex Ultrascale XCKU040 [54]. Table 4.1 below shows the total available resources in the FPGA and resources utilized by the design.

TABLE 4.1: RTL design resource utilization on XCKU040

Resource Type	**Available**	**Used**	**%age**
CLB Registers	484800	21941	4.5%
CLB LUTs	242400	25447	10.5%
CLB LUTs as logic	242400	25446	10.5%
CLB LUTs as memory	112800	1	0%

4.6.2 Speed of the design

Vivado tool which was used for all the experimentation does not provide the speed at which the design can run after synthesizing. The only way to find out if the design works at a certain frequency is to provide timing constraints and implementing the design. If the implementation steps is successful without any timing errors, it means that the design can perform at that particular frequency.

In order to check the speed estimate at synthesis stage, the design was also synthesized on Xilinx ISE software because it does provide maximum frequency for the sequential design. FPGA used for synthesis on Xilinx ISE was Kintex 7 XC7K70T FPGA [55] which had much less resources compared to the XCKU040 but had same speed grade(-2). Reason behind changing the FPGA device is that Xilinx

ISE is an older tool which does not support Ultrascale devices. Whatever speed is achieved in Kintex 7 device will be lower than the speed possible on a Kintex Ultrascale.

Table 4.2: Speed estimation of design on KC7K70T

Synthesis	**Maximum Possible Frequency**
RTL Design	489.081 MHz

Chapter 5

Proposed DRAM based RTL design for Fingerprint Matching

5.1 Overview

This chapter explains in detail, the second design approach. In this approach, matching is performed by loading reference values into a RAM. These RAMs act as a look up table during the matching phase. The RAMs used for implementation can store 64 1-bit values. While loading a 6-bit reference value in to the RAM, the value itself is treated as an address. 1-bit binary 1 is written on that address. Now at the matching phase, again values to be matched are treated as read addresses. So if the value is equal to the reference value, the output of RAM will be 1. All the blocks in design are discussed in this section. A bottom up approach is used for explaining the design starting from the lowest level module.

5.2 Single neighbor matching

This block is used to match single feature of one minutiae with single feature of the other minutiae. This module takes reference and data base values as inputs and declare if the two match or not.

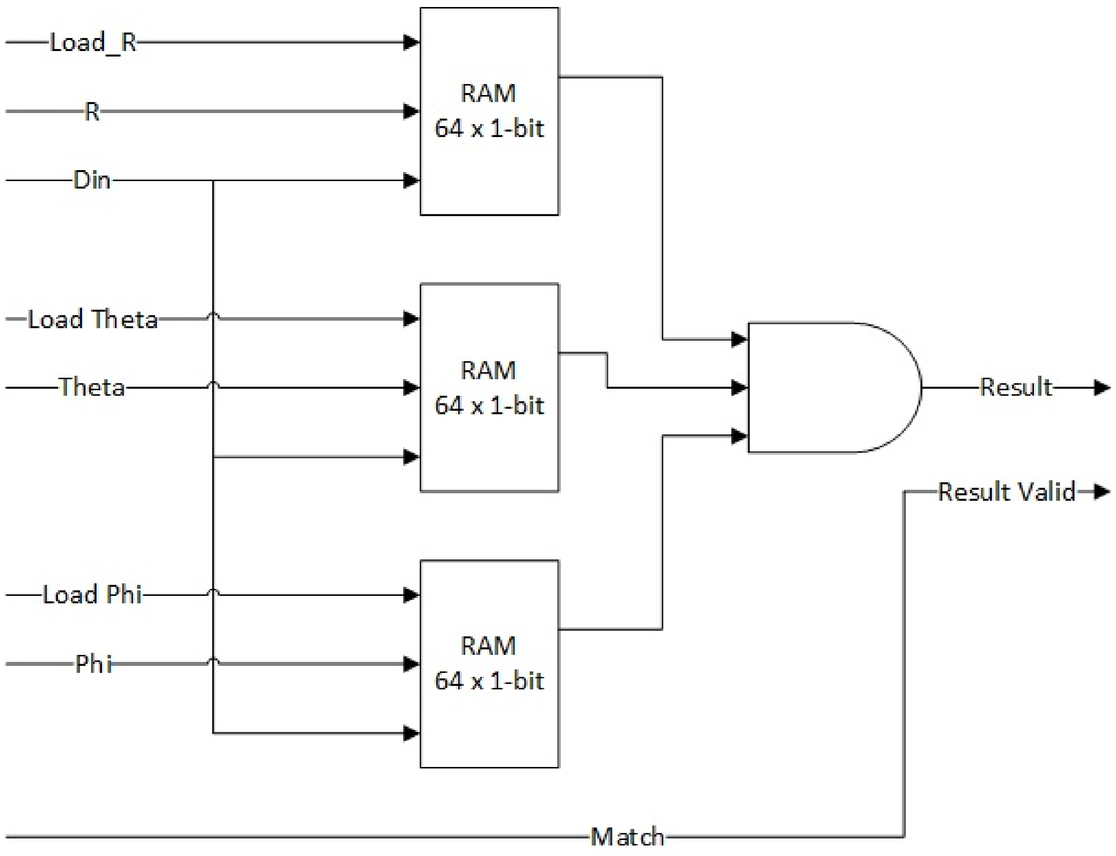
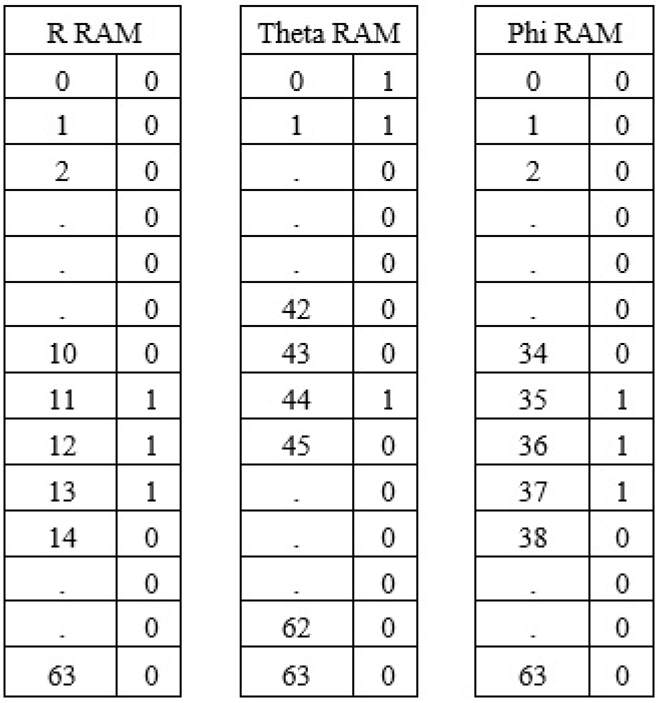


FIGURE 5.1: DRAM based single neighbor matcher

A single feature set consists of three values (r, θ and ϕ). So a single neighbor matcher incorporates three 64 x 1-bit RAMs. Address range is from 0 to 63 and Din can be 0 or 1. Writing to the RAMs is a synchronous operation while reading is asynchronous.

While loading the reference value, respective Load signal is asserted which acts as a write enable signal for RAM. Value itself is treated as address of RAM. Din is always set to 1 during reference image storing. Din is only 0 when RAMs are being cleared before loading a new reference image. Keeping in mind that while matching the features, a difference of 1 between values is also considered as a match. In order to take care of this, value 1 is also loaded to the previous and next address location. Means loading a value R into the RAM, 1 bit binary 1 is written on location R, R+1 and R-1. So for each single value, three write operations are required.

Another important thing is that Theta and Phi are angles. Angles 0 and 360 are considered equal and in our case as downscaling forced values in range of 0 to 45 degrees, 0 and 45 are considered equal. 0 and 45 will have positive match for four different value i.e. 44, 45, 0 and 1. This meant four write operation at this corner case. This case is avoided by forcing feeding protocol to send value 0 in case the actual value is 0 or 45. So effectively Theta and Phi values will be in range 0 to 44. Consider loading feature set (r = 12, $\theta = 0$ and $\phi = 36$), RAMs will be populated like the figure 5.2 suggests.



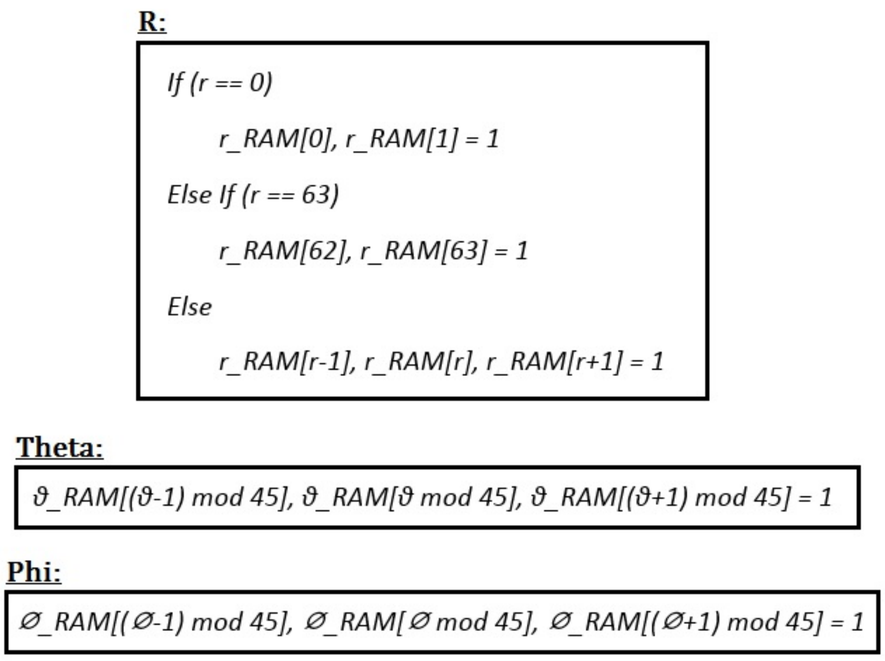
R RAM	
0	0
1	0
2	0
.	0
.	0
.	0
10	0
11	1
12	1
13	1
14	0
.	0
.	0
63	0

Theta RAM	
0	1
1	1
.	0
.	0
.	0
42	0
43	0
44	1
45	0
.	0
.	0
.	0
62	0
63	0

Phi RAM	
0	0
1	0
2	0
.	0
.	0
.	0
34	0
35	1
36	1
37	1
38	0
.	0
.	0
63	0

Figure 5.2: DRAM population based on feature value

For value (r = 12), 1 is written at address 11, 12 and 13. For value ($\theta = 0$), 1 is written at address 1, 0 and 44. For value ($\phi = 36$), 1 is written at address 35, 36 and 37. Total of nine write operations are performed in order to store three reference values. Pseudo code for loading reference r, θ and ϕ values to the respective RAMs is shown in figure 5.3



R:

```
If (r == 0)
    r_RAM[0], r_RAM[1] = 1
Else If (r == 63)
    r_RAM[62], r_RAM[63] = 1
Else
    r_RAM[r-1], r_RAM[r], r_RAM[r+1] = 1
```

Theta:

```
ϑ_RAM[(ϑ-1) mod 45], ϑ_RAM[ϑ mod 45], ϑ_RAM[(ϑ+1) mod 45] = 1
```

Phi:

```
∅_RAM[(∅-1) mod 45], ∅_RAM[∅ mod 45], ∅_RAM[(∅+1) mod 45] = 1
```

Figure 5.3: Pseudo Code for populating DRAMs with reference feature values

Now while matching, the database image feature values are again fed to RAM as addresses. An output value of 1 simply means that the value matches reference value. In case output of all three RAMs are 1, it means that the two feature sets match completely.

5.3 Minutiae matching

In order to match reference minutiae with database minutiae all nine of their neighboring features must match. This is why the minutiae matching block consists of nine single neighbor matchers. Loading mechanism of proposed design is completely different and complex compared to the conventional design.

A data distributor block is shown in the figure 5.4 below, which is responsible to provide R, Theta and Phi values to single neighbor matchers. Din also originates from data distributor.

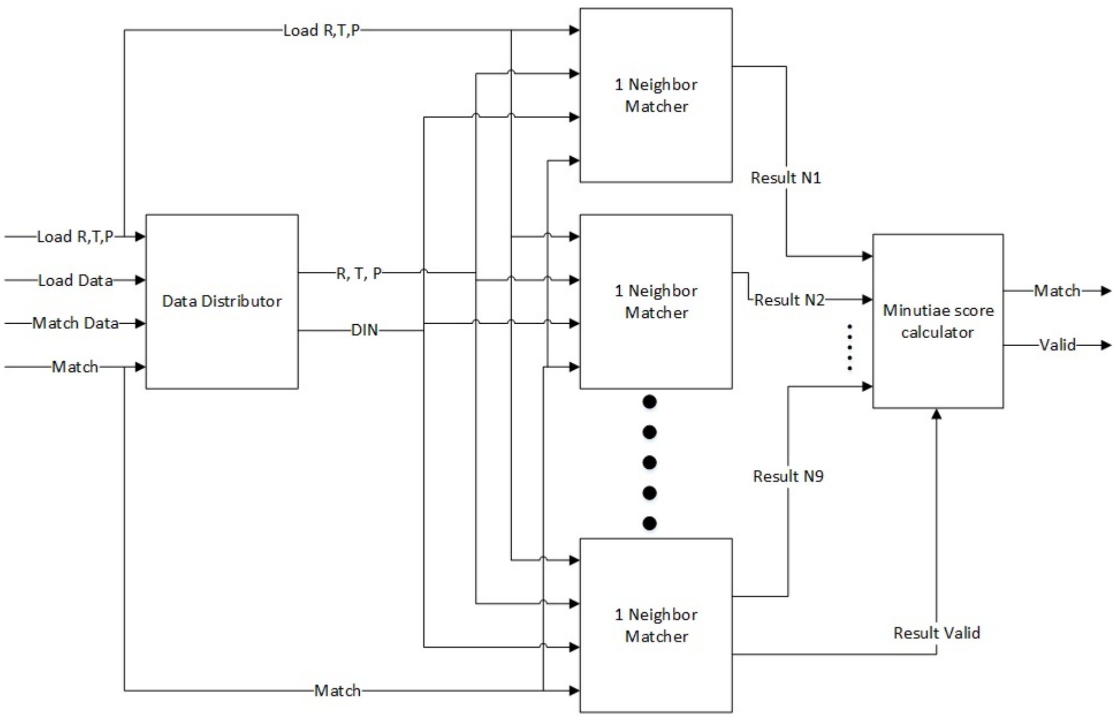


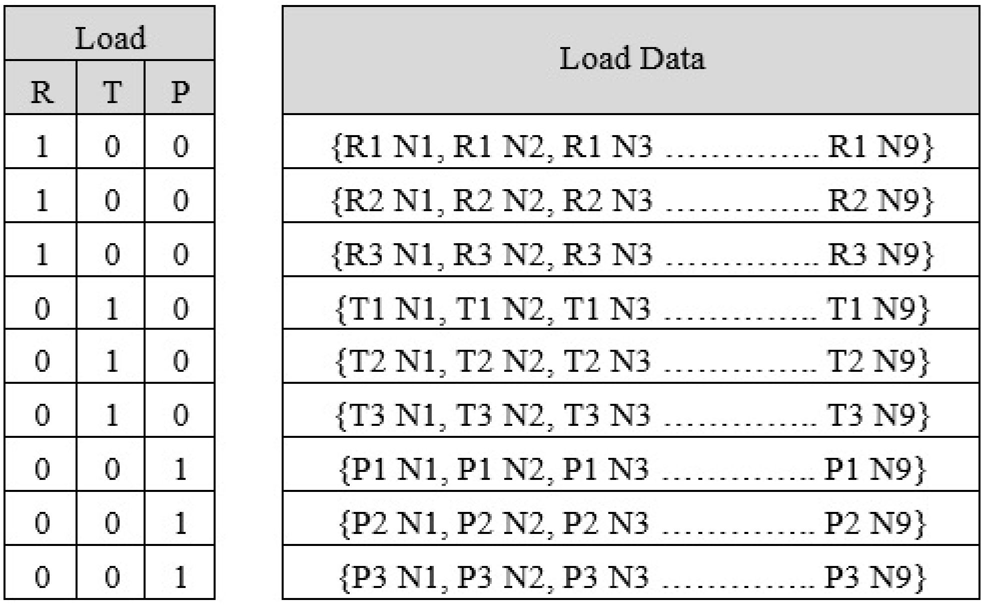
FIGURE 5.4: Minutiae matching unit for DRAM based design

5.3.1 Data Distributor

Single neighbor matcher expects reference and database values of R, Theta and Phi on a single bus but they arrive in finger print matching unit differently. R, Theta and Phi values are arranged differently on load data and match data.

Load data input is a 54 bit bus on which nine 6-bit values are placed. Each of these nine values belong to R, Theta or Phi of the nine separate single neighbor matcher. So in a single cycle, nine write operations are performed in parallel. This reduces the overall reference loading time and a whole minutiae is loaded in only 9 cycles.

Pattern to write nine (r, θ and ϕ) values to minutiae matcher is shown in figure 5.5.



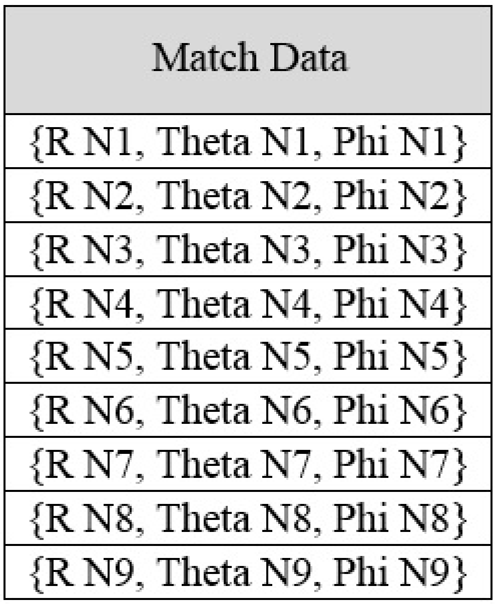
Load			Load Data
R	T	P	
1	0	0	{R1 N1, R1 N2, R1 N3 .... R1 N9}
1	0	0	{R2 N1, R2 N2, R2 N3 .... R2 N9}
1	0	0	{R3 N1, R3 N2, R3 N3 .... R3 N9}
0	1	0	{T1 N1, T1 N2, T1 N3 .... T1 N9}
0	1	0	{T2 N1, T2 N2, T2 N3 .... T2 N9}
0	1	0	{T3 N1, T3 N2, T3 N3 .... T3 N9}
0	0	1	{P1 N1, P1 N2, P1 N3 .... P1 N9}
0	0	1	{P2 N1, P2 N2, P2 N3 .... P2 N9}
0	0	1	{P3 N1, P3 N2, P3 N3 .... P3 N9}

FIGURE 5.5: Values on load data depending upon R,T,P signals

Three cycles are required to write the three values of R to nine single neighbor matchers. Same is done for Theta and Phi. Data distributor fetches data from load data bus and dispatches information to the appropriate bus. Output of data distributor is all nine 6-bit R, theta and phi values for the respective single neighbor matcher.

In a reset state when all loaded reference values are cleared, Load R, Load T and Load P are all high. In this case, the data distributor sets Din to be 0. Otherwise in normal loading operation, Din remains 1.

While matching, data is placed on 18-bit match data bus. These 18 bits contain three 6-bit R, theta and Phi values. All M2 (Number of minutiae in data base image) x 9 values are fed to matcher in sequence. Pattern to match nine (r, θ and ϕ) values of single minutiae is shown in figure 5.6. Each R, Theta and Phi value is fed to all nine single neighbor matchers in parallel with the help of data distributor.



Match Data
{R N1, Theta N1, Phi N1}
{R N2, Theta N2, Phi N2}
{R N3, Theta N3, Phi N3}
{R N4, Theta N4, Phi N4}
{R N5, Theta N5, Phi N5}
{R N6, Theta N6, Phi N6}
{R N7, Theta N7, Phi N7}
{R N8, Theta N8, Phi N8}
{R N9, Theta N9, Phi N9}

FIGURE 5.6: Feature values on match data bus

5.3.2 Minutiae Score Calculator

Match result of each neighbor is fed to minutiae score calculator which accumulates the number of neighbor matches and compares it with a threshold in order to declare positive or negative match. Figure 5.7 shows the flow of design.

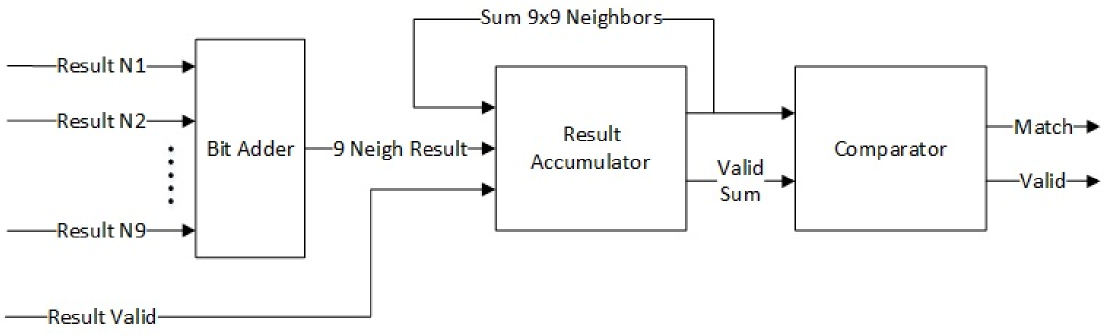


FIGURE 5.7: Minutiae score calculator

The input 9 bit vector represent number of positive matches between single neighboring feature of database minutiae and nine neighboring features of reference minutiae. Bit adder adds all the bits of this vector. This process is repeated nine times for each feature of database minutiae. Output of bit adder is fed to an accumulator which can give us the score of 9x9 matches.

Once the result for all 9x9 neighbors matching is complete, the score is fed to a comparator. After experimentation on algorithm it is established that the comparator should declare a match in minutiae if the score is greater than or equal to 4. The 9x9 result is represented by a 7 bit vector (6 down to 0) and if any bit from 6 to 2 is high, it means the value is greater than or equal to 4.

$$comparator(Sum\_9x9) = \begin{cases} 1, & if\ OR(Sum\_9x9[6:2]) \ = \ 1 \\ 0, & \qquad Otherwise \end{cases} \tag{5.1}$$

5.4 Matching module Top

In order to match reference and database finger print images, all minutiae must be matched. There will be M1 x M2 minutiae matches before computing the final score. The top-level module for fingerprint matching is shown in the figure 5.8.

The lop level matching module has 64 minutiae matchers. A special purpose de multiplexer is used to forward the load R, T, P signal to the appropriate minutiae matcher. When Reset and load both are asserted, all the matchers are loaded at once. This case is used to reset all the reference values inside matchers. The result from 64 minutiae matchers form a 64 bit vector. This vector along with a valid signal is passed to the total score calculator. Total score calculator calculates the total number of matching minutiae among the two fingerprint templates. Before feeding another database image for matching, the total score is reset. Detailed explanation of total score calculator is already covered in section 4.4.1.

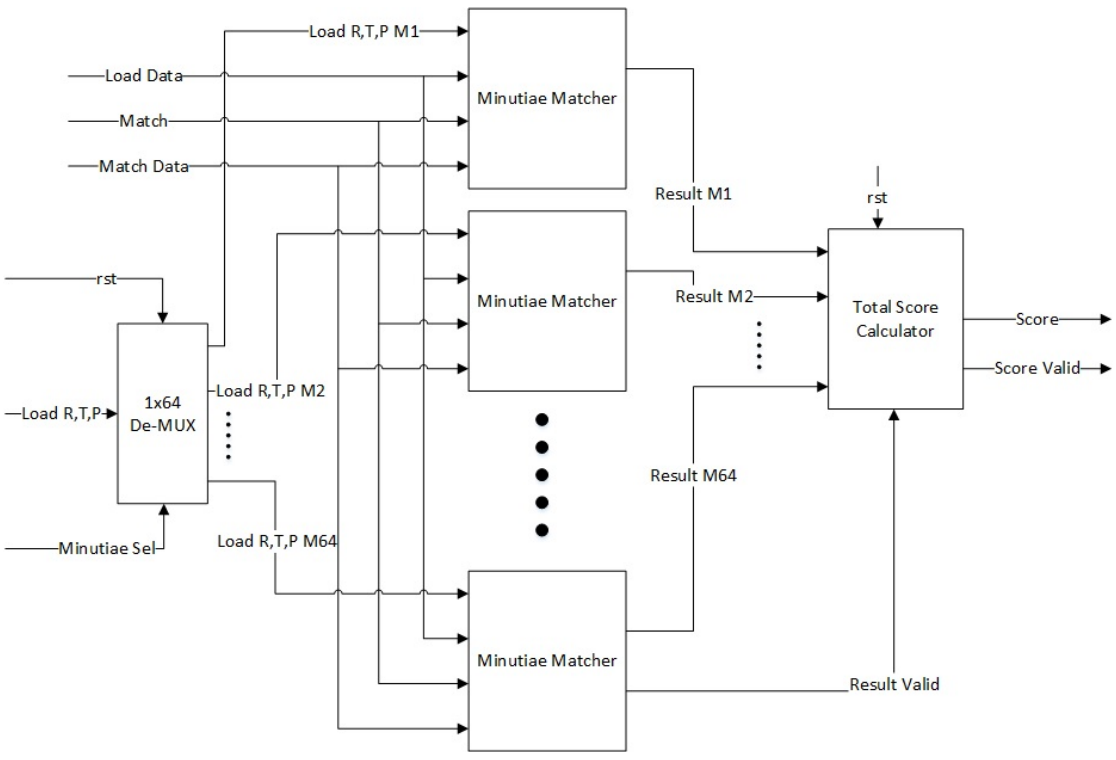


FIGURE 5.8: Top-level design for DRAM based fingerprint matcher

5.5 Pipelining in Design

All the design explained in chapter 4 and 5 are explained according to the logical flow. Clock signal is not shown in the designs to reduce complexity of the block diagrams and only focus is to explain the functionality of blocks. However throughout the design, all modules operate synchronously and pipelining is applied in order to make the design run as fast as possible.

Purpose of pipelining is to increase the maximum clock frequency at which the design can run. Pipelining may cause an initial latency but when the overall speed of execution increase. Pipelining can be explained with the help of a simple example. Consider a small design shown in figure 5.9, which require adding three values.

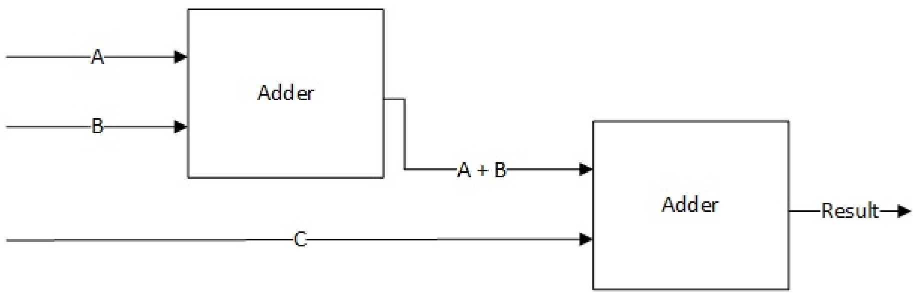


FIGURE 5.9: Example design without pipelining

Consider both adders have a combinational delay of D nanoseconds. So the total combinational delay of the given design would be 2D nanoseconds. This means the maximum frequency at which the design can work is 1/2D Hz. So if we need to perform (A + B + C) operation on 100 different values of A, B and C, it will take 100 x 2D nanoseconds. Now let's see the advantage of pipelining on the same design.

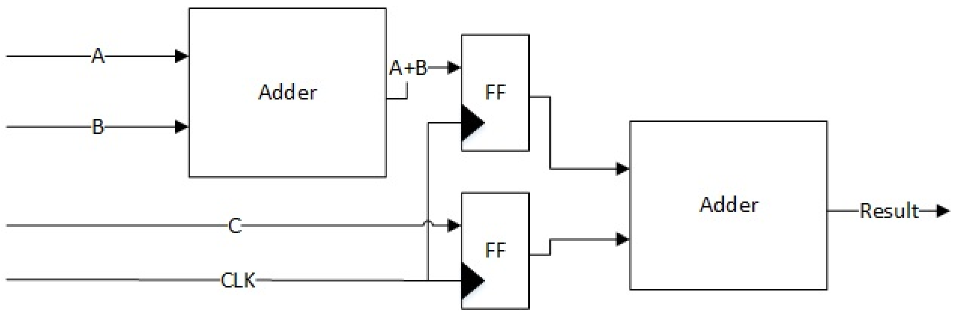


FIGURE 5.10: Pipelined example design of figure 5.8

By using flip-flops, the combinational delay of the design is now equal to the combination delay of single adder i.e. D nanoseconds. Pipelining will cause an initial latency which means result of first (A+B+C) will arrive in D + D nanoseconds. But after that each new addition will be performed in only D nanoseconds. So again in case we need to perform (A + B + C) operation on 100 different values of A, B and C, it will take (100 x D) + D nanoseconds. For example if D is equal to 5, 100 additions will take 1000 nanoseconds on non-pipelined design and only 505 nanoseconds on the pipelined design.

Pipelining in the implemented design was done in order to optimize the design for both area and performance. Each FPGA slice consist of multiple look up tables and flip flops. Logic is mapped on the look up tables (LUT) and flip flops (FF) are used for pipelining and adding delays. A good design often has a high percentage of LUT-FF pair utilization. Pipelined version of minutiae matcher is presented in next section.

5.5.1 Pipelined design of Minutiae matcher

The figures 5.11 and 5.12 below demonstrate how pipelining is performed on minutiae matcher which has already been covered in section 5.3

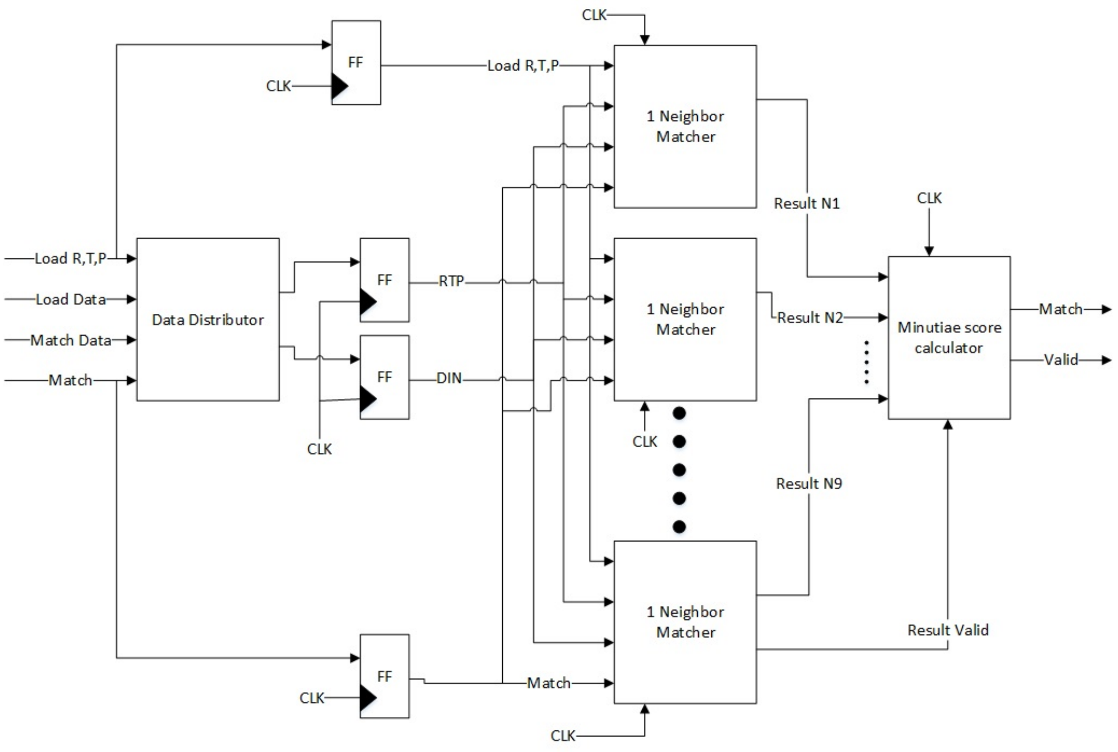


FIGURE 5.11: Pipelined version of minutiae matching unit

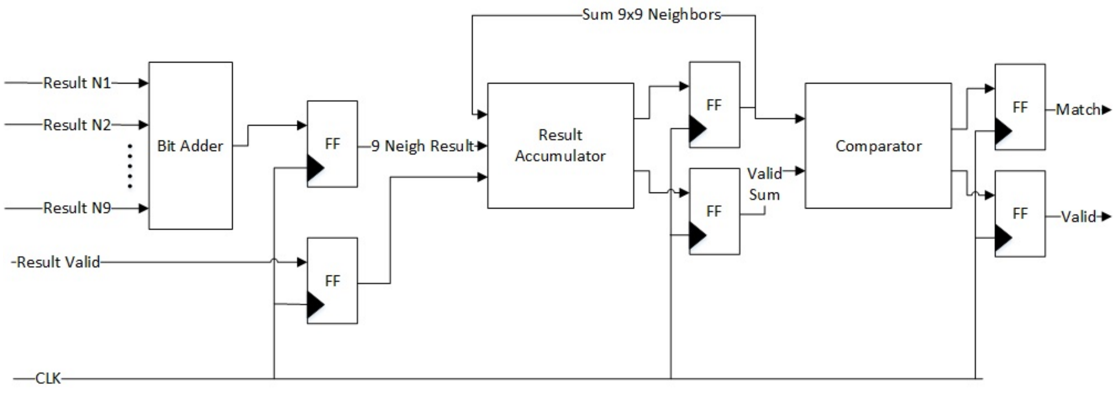
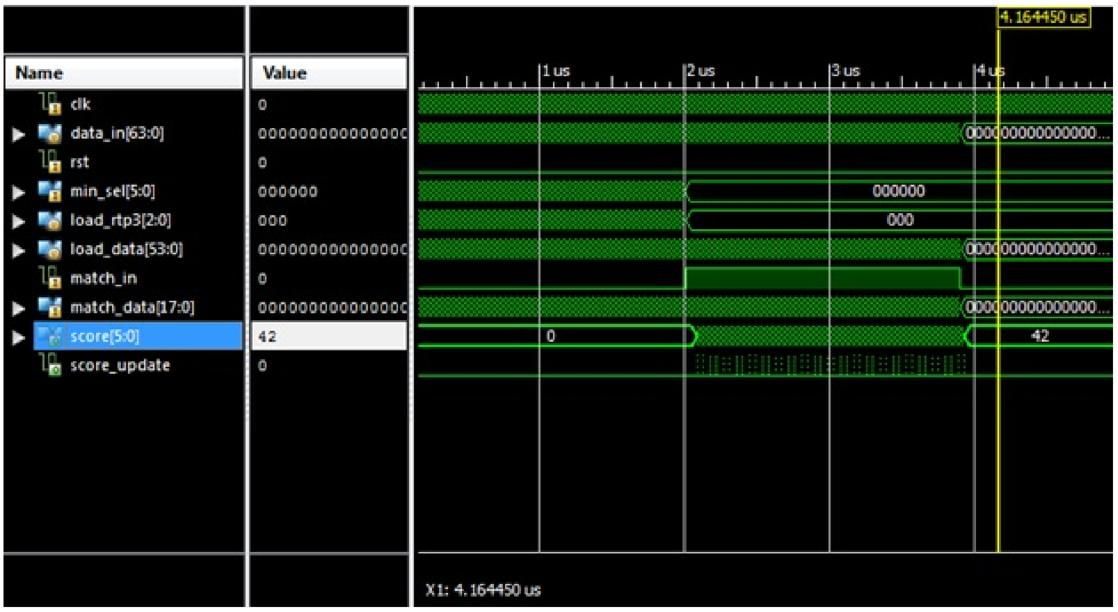


FIGURE 5.12: Pipelined minutiae score calculator

5.6 Software Simulation test

Just like the first design approach, this design was also initially verified. The simulation results showed that the DRAM based RTL design was yielding the exact same result as CPU implementation. Data was fed to UUT as required in order to load reference image and to match gallery images.



Name	Value
clk	0
data_in[63:0]	000000000000000
rst	0
min_sel[5:0]	000000
load_rtp3[2:0]	000
load_data[53:0]	000000000000000
match_in	0
match_data[17:0]	000000000000000
score[5:0]	42
score_update	0

FIGURE 5.13: Simulation testing of DRAM based design

5.7 Hardware Area and Speed report

One important goal of the research is to come up with an approach that utilizes least number of hardware resources. In order to make the comparison, DRAM based design was also synthesized to get area and speed estimate.

5.7.1 Hardware resource utilization

FPGA used for the study was Xilinx Kintex Ultrascale XCKU040 [54]. The table 5.1 below shows the total available resources in the FPGA and resources utilized by the DRAM based design and the conventional RTL design.

TABLE 5.1: Resource comparison between conventional and DRAM based design

		Conventional Design		**DRAM Based Design**	
Resources	**Available**	**Used**	**%age**	**Used**	**%age**
CLB Registers	484800	21941	4.5%	**12771**	**2.63%**
CLB LUTs	242400	25447	10.5%	**14713**	**6%**
CLB LUTs as logic	242400	25446	10.5%	**12981**	**5.3%**
CLB LUTs as mem	112800	1	0%	**1729**	**1.5%**

The table clearly shows that the proposed DRAM based approach is a much better as it require a lot lesser number of CLBs in order to pack all the design. The reason behind the better performance can be explained by explaining the FPGA architecture a little. As explained earlier in the chapter, the current design make use of RAMs in order to match feature values while the normal method required arithmetic and logical operations in order to perform comparison.

In FPGAs the logic is implemented on configurable logic blocks (CLBs). Architecture of each FPGA family differ slightly so here only the architecture of XCKU040 is discussed which is used for the experimentation. Every CLB consists of one slice with eight 6-input LUTs and sixteen storage elements (FFs) [56]. The slice can either be SLICEL or SLICEM. L in SLICEL stand for logic as the LUTs in SLICEL can only be configured for logical operation. The LUTs in SLICEM however can be used as look up table, 64-bit DRAM or 32-bit Shift register. Difference between SLICEL and SLICEM is shown in the table 5.2.

TABLE 5.2: Difference between SLICEL and SLICEM

Resources	**LUTs**	**FFs**	**DRAM**	**Shift Registers**
SLICEL	8	16	X	X
SLICEM	8	16	512 bits	256 bits

As the table tells us, there are no distributed RAM and shift registers available in SLICEL. There are total of 14,100 SLICEM in XCKU040 and 16,200 SLICEL. In the normal implementation the comparison is being performed by arithmetic and logic operations only. This means all the design will be implemented with the help of mostly LUTs and FFs with some little use of shift registers. Which results in almost no utilization of the added resources available in SLICEMs and hence total number of slices required is high.

In the proposed DRAM based design, feature matching is performed with the help of distributed RAMs. This takes off a lot of load from LUTs. Design is placed into the FPGA by making full use of the SLICEM additional resources which results in almost half number of total required CLBs.

5.7.2 Speed of the design

In order to check the speed estimate at synthesis stage, the DRAM based design was also synthesized on Xilinx ISE software. FPGA used for synthesis on Xilinx ISE was Kintex 7 XC7K70T FPGA [55] which had much less resources compared to the XCKU040 but had same speed grade(-2). Reason behind changing the FPGA device is that Xilinx ISE is an older tool which does not support Ultrascale devices. Whatever speed is achieved in Kintex 7 device will be lower than the speed possible on a Kintex Ultrascale. The table below shows maximum possible clock speed for DRAM based design. Performance of conventional design is also shown for reference.

Table 5.3: Speed comparison between the two proposed designs

Design	**Maximum Frequency**
Conventional	489.081 MHz
DRAM based	**560.711 MHz**

Chapter 6

Performance Evaluation and Discussion

After verifying both design approaches and comparing synthesis reports, final step is to shift to an actual hardware and test the design. Moving forward all the tests are performed for the DRAM based design which clearly has much less resource utilization and is capable of running at a much faster clock.

6.1 Hardware Testing Wrapper

In hardware the test fixture cannot be used as it is not synthesizable code. Only synthesizable code can be implemented on FPGA. So the task in hand is to feed information to the fingerprint matching module. In a complete implemented product, FPGA will be getting all the information via some external interface. But for hardware testing without the involvement of any external hardware, we had to create a testing mechanism inside the FPGA as shown in figure 6.1.

To provide input to finger print matching module, a ROM was instantiated inside the FPGA. This ROM was programmed to hold all the data which was required to be fed to matching module. ROM held loading data for reference image as well as the matching data for database images. Purpose of counter in the design is to provide read addresses to ROM that will in turn provide input to the matching module. ROM was 64 bit wide and it held data in the format explained in the next section.

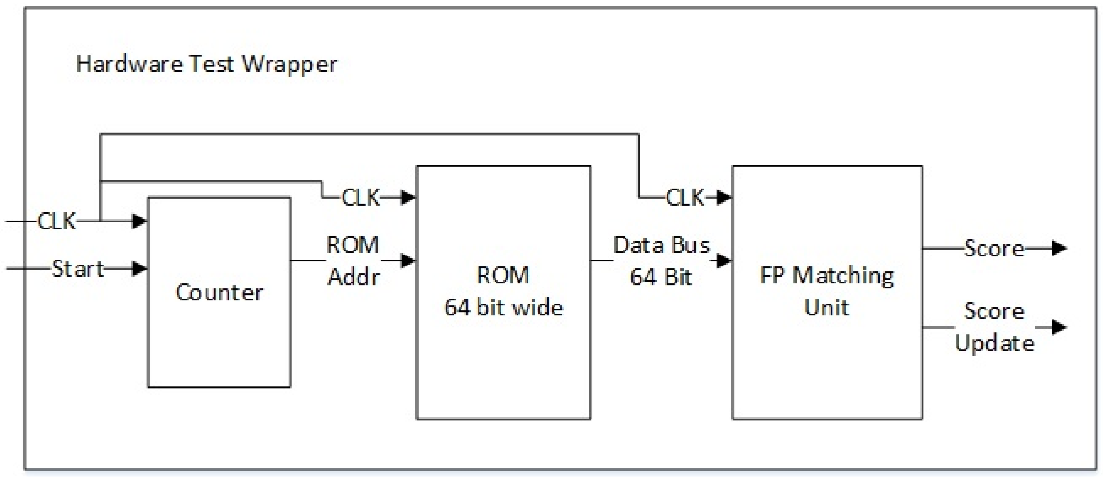
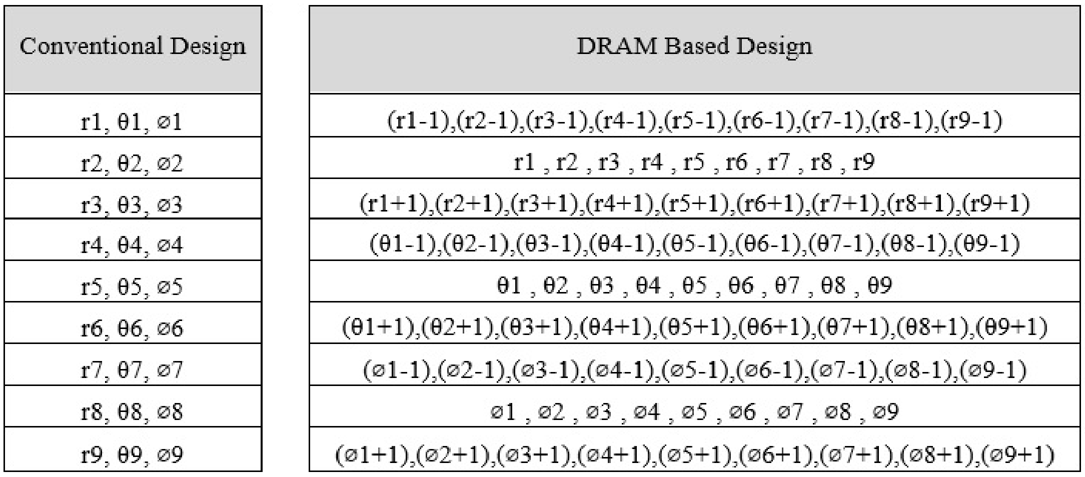


Figure 6.1: Hardware design for testing FP Matching unit

In order to perform matching test on hardware, the ROM was filled with one reference image at a time. Information of 800 images from FVC2002 [1] were also stored in ROM which were all matched with the reference image. Multiple tests were performed by changing the reference image. Whichever reference image is loaded into the design, it is matched with all database images and scores are calculated.

6.1.1 ROM Output Bus format

Before explaining the contents of 64-bit bus lets recap the loading requirement in DRAM based design. As explained in the design, for each value of R, Theta and Phi, three values are required to be loaded to the DRAM. This would have caused a three times increase in loading time. Therefore, the loading mechanism was changed. In a single cycle, nine values of R, Theta or Phi were loaded at once. The input data taken from the CPU code was modified in the pattern, which can be stored into the RTL design. This is explained with the help of an example for single minutiae. In conventional design, the reference image was loaded in nine cycles where each of the nine neighboring r, θ, and ϕ are stored in a single cycle. In the proposed design all the nine values of r, θ, or ϕ are loaded at once and because there are three different values for each r, θ, and ϕ, total of only 9 cycles are needed to load the whole minutiae.



Conventional Design	DRAM Based Design
r1, θ1, ø1	(r1-1),(r2-1),(r3-1),(r4-1),(r5-1),(r6-1),(r7-1),(r8-1),(r9-1)
r2, θ2, ø2	r1 , r2 , r3 , r4 , r5 , r6 , r7 , r8 , r9
r3, θ3, ø3	(r1+1),(r2+1),(r3+1),(r4+1),(r5+1),(r6+1),(r7+1),(r8+1),(r9+1)
r4, θ4, ø4	(θ1-1),(θ2-1),(θ3-1),(θ4-1),(θ5-1),(θ6-1),(θ7-1),(θ8-1),(θ9-1)
r5, θ5, ø5	θ1 , θ2 , θ3 , θ4 , θ5 , θ6 , θ7 , θ8 , θ9
r6, θ6, ø6	(θ1+1),(θ2+1),(θ3+1),(θ4+1),(θ5+1),(θ6+1),(θ7+1),(θ8+1),(θ9+1)
r7, θ7, ø7	(ø1-1),(ø2-1),(ø3-1),(ø4-1),(ø5-1),(ø6-1),(ø7-1),(ø8-1),(ø9-1)
r8, θ8, ø8	ø1 , ø2 , ø3 , ø4 , ø5 , ø6 , ø7 , ø8 , ø9
r9, θ9, ø9	(ø1+1),(ø2+1),(ø3+1),(ø4+1),(ø5+1),(ø6+1),(ø7+1),(ø8+1),(ø9+1)

FIGURE 6.2: Difference between 'loading' in the two proposed designs

A 64 bit bus is used in case of the DRAM based design in order to store reference image to the design, send data base images for matching and to reset the loaded reference image. Detailed purpose of each entry in the bus has already been explained in the chapter 5. Data organization on the bus is shown using figures 6.3, 6.4 and 6.5.

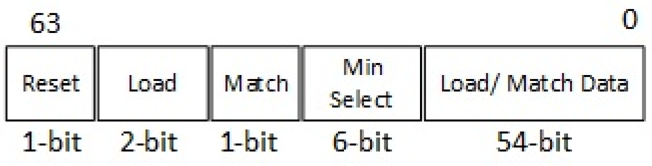


FIGURE 6.3: Organization of 64-bit input data bus

Load is 2 bit in the proposed design as it is used to select if the values being loaded belong to R, Theta or Phi. While loading the reference image, Reset is set to 0, Load is set according to the data being loaded, Match is ignored, Min select tells which minutiae is being loaded and Load data is as shown in figure 6.2 values of either R, Theta or Phi(6-bit each).

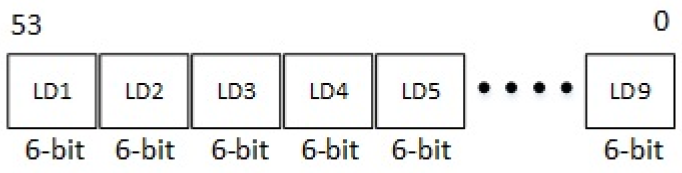


FIGURE 6.4: Load data information

While matching, Reset and Load must be set to 0. Match will be 1, Min select is ignored in this case. Match data will only have 18 least significant bits filled with R, Theta and Phi of data base image.

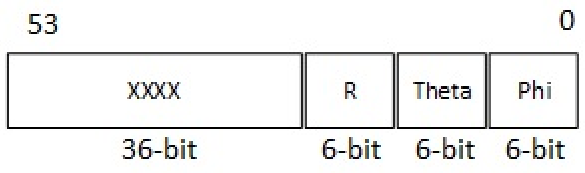


FIGURE 6.5: Data base image matching information

In order to clear the loaded reference image Reset is set to 1, Load is set to 3, Match and Min Select are ignored in this scenario. Load data will contain addresses of RAMs that are required to be cleared. So LD1 to LD9 values will move from 0 to 63 in order to clear the whole RAM. 63 cycles are required to reset all the 1728 BRAMs because all BRAMs of all minutiae are cleared in parallel.

6.2 Hardware platform

Hardware used for this experiment was Xilinx Kintex UltraScale FPGA Evaluation Kit KCU105 [57] shown in figure 6.6. The available FPGA on the development kit was a Kintex Ultrascale XCKU040-2FFVA1156E. Clock was provided by an oscillator on board and start signal was provided by an on-board push button.

FIGURE 6.6: Xilinx Kintex Ultrascale evaluation board KCU105

And in order to see the output, ILA core was added to the design. ILA core helps in capturing real time internal waveforms from FPGA and displays them on a testing PC. ILA uses JTAG interface in order to display data from FPGA to a testing PC. The results were verified on hardware as well after successful testing.

6.3 Matching unit capability: Matches per second

The matches per second will only be discussed for the DRAM based design, which is better in all respects. Total number of cycles required to feed a data base template with M number of minutiae to the matching unit are:

$$Cycles\ for\ Feeding = (M * 9)$$

Where 9 is the number of neighboring features matched for each minutiae. Once all the information is fed to the matching unit, final score is updated on bus after a few cycles. Next data base template can only be fed to the matching unit once score of previous template is computed. These few cycles delay must be included while calculating matches per second. This delay was 6 cycles in our case and it is shown in figure 6.7.

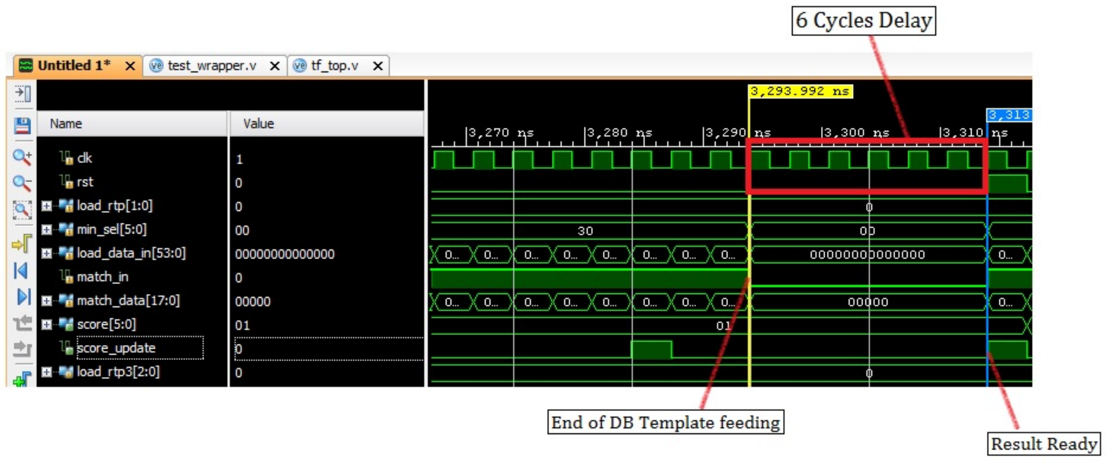
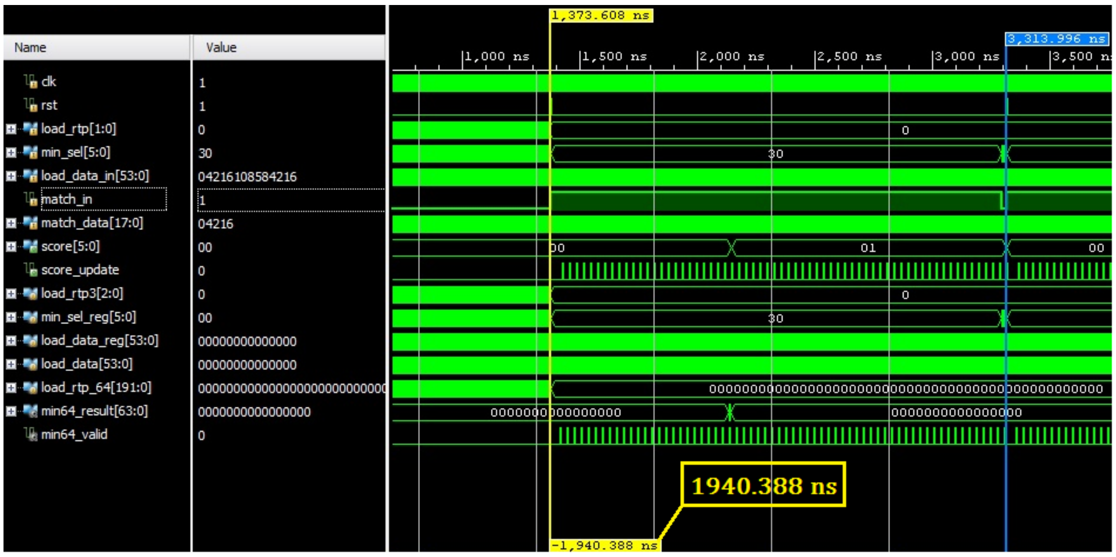


FIGURE 6.7: Delay after feeding complete DB template information

This 6 cycles delay is independent of minutiae count and will always remain constant. So total number of cycles required to match a single data base template of M minutiae are:

$$Cycles\ for\ Matching\ =\ (M * 9)\ +\ 6$$

The hardware test was performed at 300 MHz successfully which means one clock cycle is equal to 3.33 nanoseconds. In order to calculate matches per second, we used a scenario where minutiae in DB image is equal to the maximum possible minutiae count i.e. 64. Test results are shown in Figure 6.8



Name	Value
clk	1
rst	1
load_rtp[1:0]	0
min_sel[5:0]	30
load_data_in[53:0]	04216108584216
match_in	1
match_data[17:0]	04216
score[5:0]	00
score_update	0
load_rtp3[2:0]	0
min_sel_reg[5:0]	00
load_data_reg[53:0]	00000000000000
load_data[53:0]	00000000000000
load_rtp_64[191:0]	000000000000000000000000000
min64_result[63:0]	0000000000000000
min64_valid	0

FIGURE 6.8: Match time for DB template with 64 Minutiae

Only 1,940.388 nanoseconds are required to match the image which means 515,360 matches per second. The average minutiae count is 45 in the FVC2002 [1] database so matches per second will be much less on average. Table 6.1 show matches per second for both the scenarios.

TABLE 6.1: Matches per second achieved using single matcher at 300 MHz

Minutiae Count	**Clock Speed (MHz)**	**Clock Period (ns)**	**Match Time (microsecond)**	**Matches/ sec**
Max (64)	300	3.33	1.940	515,360
Avg (45)			1.370	729,925

6.4 Multiple FP matching units in parallel

The free available space in the FPGA allows us to use multiple finger print matchers in parallel. Same reference image can be loaded to all of the matcher modules while the database images can be fed to separate matching modules. This means a direct increase in the hardware resource utilization but it also means a direct increase in matches per second.

In order to feed data to matching units, Separate ROMs were used for Loading and Matching phases. In our experiment, we worked with 8 matching units in parallel. Testing was performed by loading 100 gallery images into the ROM and the design was verified on simulation as well as hardware. Details of test will be discussed in later section.

6.4.1 Calculating percentage score and best match

The output score coming out of each matching unit represents the number of minutiae matched between the respective gallery image and the database image. In order to compare the scores, first a percentage score is required as mentioned in equation 3.17. Percentage score calculation and best match search is performed in a manner which is explained in figure 6.9

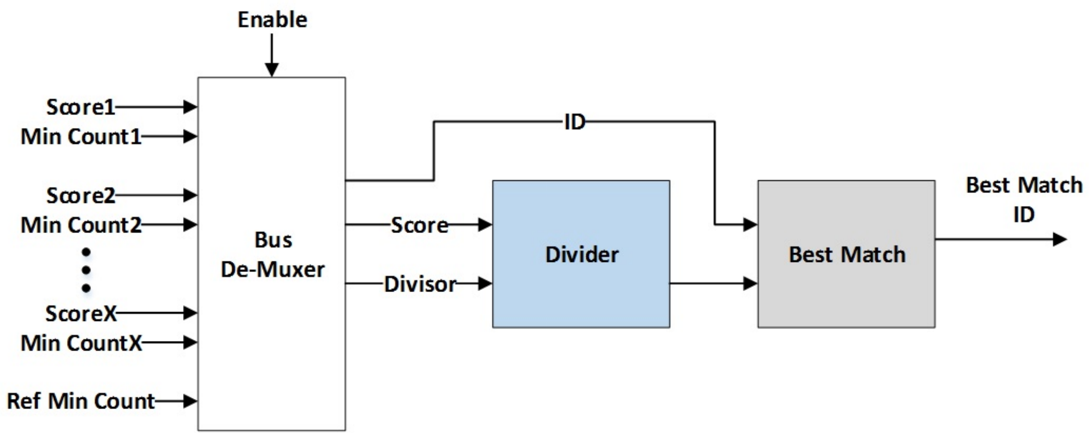


FIGURE 6.9: Percentage score calculation and best match searching

Where X is the number of matching units working in parallel. When minutiae score from all three matching units are available on bus, the scores are captured and transmitted in a sequential manner using a bus multiplexer. This demultiplexer also finds the divisor for respective score which will be used to calculate percentage. Divisor for any particular gallery image is simply the minimum of gallery image and reference image minutiae count (see 3.17). The demultiplexer generates IDs and send it along with the score and divisor.

Benefit of transmitting information of three matching units in series is that only single divider block is enough for processing. Divider computes the percentage score which then enters into the "best match" block synchronized with respective ID. Best match block initially hold '0' value as reference score. For each score entering the block, it is matched with reference score. If the new score is greater than reference score, it becomes the new reference score and it's ID is stored as best match.

6.4.2 Effect on resources utilization

The effect of this using multiple matching units can be seen in the table 6.2. The table gives the resource utilization on Kintex Ultrascale XCKU040 FPGA which was used for all the development and testing.

Table 6.2: Resources utilized after using 8 matchers in parallel

Resources	**Available**	**Used**	**%age**
Slice Registers	484800	102923	21.23%
Slice LUTs	242400	118776	49.01%
LUTs used as Logic	242400	104474	43.10%
LUTs used as Memory	112800	14336	12.71%

The resource utilization increase linearly as we increase the number of matching units. Trend of increase in resource utilization is shown in table 6.3. Table shows resource utilization in case of 1, 3 and 8 matching units.

Table 6.3: Resource utilization in case of using 1,3 or 8 FP Matching units

Resources	**Available**	**1 Unit**	**3 Units**	**8 Units**
CLB Registers	484800	2.65%	7.8%	21.23%
CLB LUTs	242400	6.1%	18.17%	49.01%
CLB LUTs as logic	242400	5.39%	16.03%	43.10%
CLB LUTs as mem	112800	1.5%	4.6%	12.71%

An important thing to consider while increasing number of matching units is that the loading and matching data will be fed from some external interface and that will become bottleneck of the design. Achievable matches per second (MPS) and Input data rate required for different number of matching units and different frequencies are shown in the two table below.

Table 6.4: Matches per second possible depending on number of units and operating frequencies

Number of	**Operating Speed**		
Matching Units	**100 MHz**	**200 MHz**	**300 MHz**
1	171,609 MPS	343,669 MPS	515,360 MPS
2	343,218 MPS	687,338 MPS	1,030,558 MPS
3	514,827 MPS	1,031,007 MPS	1,546,082 MPS

Table 6.5: Required bandwidth for different number of units and operating frequencies

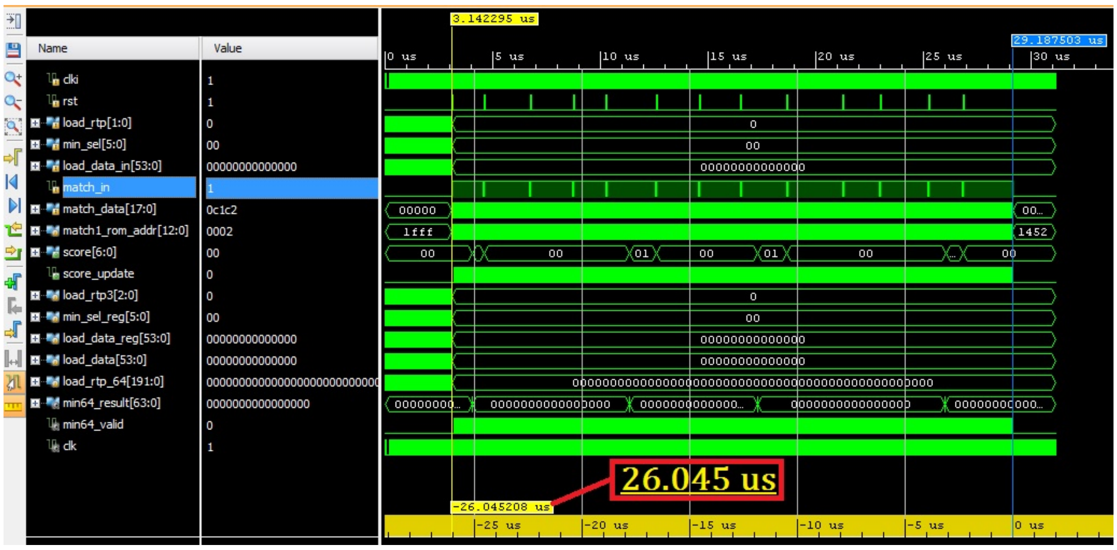
Number of	**Operating Speed**		
Matching Units	**100 MHz**	**200 MHz**	**300 MHz**
1	381 MB/s	763 MB/s	1,144 MB/s
2	572 MB/s	1,144 MB/s	1,715 MB/s
3	763 MB/s	1,526 MB/s	2,289 MB/s

The communication interfaces are also getting faster. The requirement by our design can easily be met by using the 8-lane PCIe interface available on the KCU105. We can get around 8 Gigabytes/sec input data by using the 8 lane PCIe 3.0 [58].

6.5 Evaluating Hardware Performance: Matches per Second

The maximum possible speed we can achieve on KCU105 depends on multiple factors. From tables 6.4 and 6.5 we can see that increasing number of matching units is a better way to increase matches per second compared to increasing operating frequency. Required bandwidth will be the bottleneck in this case and not the FPGA and design capability. We increased the number of matching units and implemented the design using Vivado after applying timing constraints. The number of matching units which we were able to implement were 8. The design was successfully implemented on Kintex ultrascale FPGA when constrained to run at 200 MHz.

In order to test the design's functionality, 100 gallery images were loaded into the matching ROMs and were fed to 8 matching units. This meant that for first 4 matching units, 13 images were fed and for last 4 matching units, 12 images were sent. The scores were verified on simulation and hardware. Matches per second was observed from Vivado ILA. As in parallel systems, the speed will be equal to the speed of matching unit that is last to finish all the assigned matches. Matching unit 2 had to match images which had on average larger minutiae count so it was used to calculate the matches per second capability of our hardware. Figure 6.10 shows the time required to match 100 images while using 8 matching units running at 200 MHz.



Name	Value
clki	1
rst	1
load_rtp[1:0]	0
min_sel[5:0]	00
load_data_in[53:0]	00000000000000
match_in	1
match_data[17:0]	0c1c2
match1_rom_addr[12:0]	0002
score[6:0]	00
score_update	0
load_rtp3[2:0]	0
min_sel_reg[5:0]	00
load_data_reg[53:0]	00000000000000
load_data[53:0]	00000000000000
load_rtp_64[191:0]	0000000000000000000000000000
min64_result[63:0]	0000000000000000
min64_valid	0
clk	1

FIGURE 6.10: Time taken by Hardware to match 100 images

As shown in the figure, we require 26.045 microseconds to match 100 images. This figure translate to 3,839,508 matches per second. This is because the average minutiae count for data base images used is around 45.

We also performed testing in order to compute matches per second in the worst case scenario which means if each data base image has minutiae count equal to 64. For this purpose we stored a 64 minutiae count image and respectively provided the same image to all matches working in parallel. This gave us the figure of 2,749,140 matches per second.

Keeping this figure as target, hardware tests were performed to get maximum matches per second. We were able to successfully run hardware test using 6 matching units in parallel, which means 2,577,319 matches per second as shown in table 6.6 below. Design with 12 matching units was also synthesized successfully but running on hardware require some work at the logic placement stage.

TABLE 6.6: Achieved Matches Per Second on Hardware

Test Type	**Clock Speed**	**Matching Units**	**Matches/ sec**
For Max Minutiae Count	200 MHz	8	2,749,140
Test on 100 DB images			3,839,508

6.6 Comparison with previously published FPGA matchers

The table 6.8 below compares the performance of our design with already published work on fingerprint matching acceleration using FPGA. The details of these designs have already been covered in literature review (Section 2.6) and here it is only summarized in tabular form.

TABLE 6.7: Comparison with other published FPGA based Fingerprint matchers [50][52][53]

Method	EER	FPGA	MPS	Drawback
Lindoso et al.	5.35%	Xilinx Virtex 4	7,067	- Low EER - Non Standard DB
Danese et al.	6.16%	Altera Stratix II Nios II Processor	1,515 (Current) 14,285 (Predicted)	- Low EER
Jiang et al.	Not mentioned	Xilinx Virtex E	1,220,000	- Doesn't cater fingerprint alignment - High MPS at cost of accuracy
Proposed	1.65%	Xilinx Kintex Ultrascale	2,749,140	

6.7 Synthesis report on other FPGAs

The design did not use any resource that was dedicated for Kintex UltraScale FPGAs. All the RTL level programming was done to make sure platform independence. Though he concept used for optimization was used by studying the architecture of Xilinx FPGAs. The Distributed RAM instance used is a primitive 64x1 DRAM block that is available in all Xilinx FPGAs. The resource utilization for the proposed DRAM based design was also checked for several other Xilinx FPGAs. This included some old and mostly newer FPGA families. The results are shown only for a single matching instance. All the devices selected have a -2 speed grade.

Table 6.8: Resource utilization on some other Xilinx FPGAs

Speed and Area	**FPGA Family and Device**			
	Spartan 6	**Virtex 5**	**Kintex 7**	**Artix 7**
	XC6SLX45T	**XC5VLX110T**	**XC7K70T**	**XC7A100T**
Slice Registes	22%	18%	14%	9%
Slice LUTs	54%	22%	35%	22%
Max CLK Freq	235.4 MHz	351.2 MHz	560.71 MHz	419.38 MHz

Chapter 7

Conclusion and Future Work

The research process was able to give many positive outcomes. We were able to implement an encoding and matching algorithm which provide good accurate results. Hardware design approach was proposed which consumes almost half resources compared to the conventional design approach. The matching algorithm was ported to hardware which accelerated identification process, capable of handling multi-million matches per second. Total of 2,749,140 matches per second were achieved on KCU105 board by using 8 Fingerprint matching units in parallel running at 200MHz.

TABLE 7.1: Achieved Matches Per Second on KCU105

Matching Units	**Resource Utilization**	**Operating Speed**	**Matches per sec**
8	49.01%	200 MHz	2,749,140

An AFIS depends on design of matching unit as well as design of interface. The current research work focused only on design of matching unit. For future work, a complete system can be developed in which a server can feed reference and database images to KCU105 using some external interface. In case we increase the matching units to 12, keeping FPGA utilization to around 73%, we will get 2,061,855 matches per second at 100 MHz. But this will mean 2.69 Gigabytes/

sec required bandwidth. Assuming we can get around 7 Gigabytes/ sec bandwidth easily from the available PCIe interface, we can ran our design at 250 MHz frequency, which will give 5,154,639 matches per second.

Table 7.2: Achievable MPS on KCU105 using the proposed design

Matching Units	**Resource Utilization**	**Operating Speed**	**Matches per sec**	**Required Bandwidth**
12	72.8%	250 MHz	5,154,639	6.7 GB/s

Future work include achieving the maximum possible matches per second on hardware which as estimated in table 7.2. There is still a need of fast and low-cost fingerprint matching solutions in many field. The design can be incorporated in any design and ported to any FPGA. This design is scalable for the need of application. In case of low computational requirement, the design can be ported to a low cost FPGA device, running at a much lower data rate. Further work can be done on testing the design on other databases as well and further optimizing the accuracy of algorithm.

Due to customizable IOs, FPGA is capable of interfacing with any chip directly. Furthermore, FPGA consume much less power compared to GPUs. This makes FPGA a candidate for making a standalone-dedicated hardware, which is capable of performing multimillion matches per second. FPGA can directly interface with any standard fingerprint acquisition device and any storage device used for storing database template images. So in future, work can be done on a standalone FPGA based AFIS that is portable, consumes very less power and can perform multimillion matches per second.

Bibliography

[1] "Fingerprint verification competition 2002," *http://bias.csr.unibo.it/fvc2002/*.

[2] L. Hong, Y. Wan, and A. K. Jain, "Fingerprint image enhancement: Algorithms and performance evaluation," *IEEE Transactions on Pattern Analysis Machine Intelligence*, vol. 20, no. 8, pp. 777–789, 1998.

[3] S. Chikkerur, A. N. Cartwright, and V. Govindaraju, "Fingerprint enhancement using stft analysis," *Pattern Recognition*, vol. 40, no. 1, pp. 198–211, 2007.

[4] X. Jiang and W.-Y. Yau, "Fingerprint minutiae matching based on the local and global structures," *15th International conference on Pattern Recognition*, vol. 2, pp. 1038–1041, 2000.

[5] K. Jain, A. Ross, and S. Prabhakar, "An introduction to biometric recognition," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 14, pp. 4–20, 2004.

[6] T. Y. Jea and V. Govindaraju, "A minutia-based partial fingerprint recognition system," *Pattern Recognition*, vol. 38, pp. 1672–1684, 2005.

[7] D. Maltoni, D. Maio, A. K. Jain, and S. Prabhakar, "Handbook of fingerprint recognition," *2nd Edition, Springer, New York*, 2009.

[8] W. Zhao, R. Chellappa, A. Rosenfeld, and P. J. Phillips, "Face recognition: A literature survey," *ACM Computing Surveys*, vol. 35, no. 4, pp. 399–458, 2003.

[9] J. Daugman, "How iris recognition works," *IEEE Transactions on Circuits and systems for Video Technology*, vol. 14, no. 1, pp. 21–30, 2004.

[10] R. Cappelli, M. Ferrara, and D. Maio, "A fast and accurate palmprint recognition system based on minutiae," *IEEE Transactions on Systems, Man and Cybernetics - Part B*, vol. 42, no. 3, pp. 956–962, 2012.

[11] "Neurotechnology, verifinger," *http://www.neurotechnology.com*.

[12] "Homeland security news wire, 18 january 2011," *http://www.homelandsecuritynewswire.com/biometrics-market-expected-hit-12-billion-2015-0*.

[13] "Basic fpga architecture and its applications," *https://www.edgefx.in/fpga-architecture-applications/*.

[14] Y. Yamaguchi, "Performance comparison of fpga, gpu and cpu in image processing," *19th International conference on Field Programmable Logic and Application*, 2009.

[15] D. Maio and D. Maltoni, "Direct gray-scale minutiae detection in fingerprints," *IEEE Transactions on Pattern Analysis Machine Intelligence*, vol. 19, no. 1, pp. 27–40, 1997.

[16] C. I. Watson, M. D. Garris, E. Tabassi, C. L. Wilson, R. M. McCabe, S. Janet, and K. Ko, "User's guide to nist biometric image software (nbis)," *National institute of standards and technology*, 2002.

[17] B. G. Kim and D. J. Park, "Adaptive image normalization based on block processing for enhancement of fingerprint image," *Electronics Letters*, vol. 38, no. 14, pp. 696–698, 2002.

[18] S. Greenberg, M. Aladjem, D. Kogan, and I. Dimitrov, "Fingerprint image enhancement using filtering techniques," *International conference on Pattern Recognition (15th)*, vol. 3, pp. 326–329, 2000.

[19] A. J. Willis and L. Myers, “A cost–effective fingerprint recognition system for use with low-quality prints and damaged fingertips,” *Pattern Recognition*, vol. 34, no. 2, pp. 255–270, 2001.

[20] A. M. Bazen and S. H. Gerez, “Segmentation of fingerprint images,” *Workshop on Circuits Systems and Signal Processing (ProRISC)*, 2001.

[21] E. Chandra and K. Kanagalakshmi, “Noise elimination in fingerprint images using median filter,” *International Journal of Advanced Networking and Applications*, vol. 2, no. 6, pp. 950–955, 2011.

[22] F. Turroni, D. Maltoni, R. Cappelli, and D. Maio, “Improving fingerprint orientation extraction,” *IEEE Transactions on Information Forensics Security*, vol. 6, no. 3, pp. 1002–1013, 2011.

[23] A. Grasselli, “On the automatic classification of fingerprints,” *Methodologies of Pattern Recognition, (S. Watanabe Edition), Academic, New York*, pp. 253–273, 1969.

[24] A. M. Bazen and S. H. Gerez, “Systematic methods for the computation of the directional fields and singular points of fingerprints,” *IEEE Transactions on Pattern Analysis Machine Intelligence*, vol. 24, no. 7, pp. 905–919, 2002.

[25] M. Kass and A. Witkin, “Analyzing oriented patterns,” *Computer Vision Graphics and Image Processing*, vol. 37, no. 3, pp. 362–385, 1987.

[26] N. K. Ratha, S. Y. Chen, and A. K. Jain, “Adaptive flow orientation-based feature extraction in fingerprint images,” *Pattern Recognition*, vol. 28, no. 11, pp. 1657–1672, 1995.

[27] W. Wang, J. W. Li, F. F. Huang, and H. L. Feng, “Design and implementation of log-gabor filter in fingerprint image enhancement,” *Pattern Recognition Letters*, vol. 29, pp. 301–308, 2008.

[28] L. O’Gorman and J. V. Nickerson, “An approach to fingerprint filter design,” *Pattern Recognition*, vol. 22, no. 1, pp. 29–38, 1989.

[29] M. Ghafoor, I. A. Taj, W. Ahmed, and M. N. Jafri, "Efficient 2-fold contextual filtering approach for fingerprint enhancement," *IET Image Processing*, vol. 8, no. 7, pp. 417–425, 2014.

[30] A. K. Jain, S. Prabhakar, L. Hong, and S. Pankanti, "Filterbank - based fingerprint matching," *IEEE Transactions on image processing*, vol. 9, no. 5, pp. 846–859, 2000.

[31] Y. Jie, Y. Y. Fang, Z. Renjie, and S. Qifa, "Fingerprint minutiae matching algorithm for real time system," *Pattern Recognition*, vol. 39, no. 1, pp. 143–146, 2006.

[32] S. Chikkerur and N. Ratha, "Impact of singular point detection on fingerprint matching performance," *Automatic Identification Advanced Technologies*, pp. 207–212, 2005.

[33] A. K. Jain, L. Hong, and R. Bolle, "On-line fingerprint verification," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 4, pp. 302–314, 1997.

[34] J. Zhou and J. Gu, "A model-based method for the computation of fingerprints' orientation field," *IEEE Transactions on Image Processing*, vol. 13, no. 6, pp. 821–835, 2004.

[35] M. Tico and P. Kuosmanen, "An algorithm for fingerprint image post processing," *34th Asilomar Conference on Signals, Systems and Computers*, 2000.

[36] A. K. Jain, J. Feng, A. Nagar, and K. Nandakumar, "On matching latent fingerprints," *Computer Vision and Pattern Recognition Workshop*, 2008.

[37] S. Chikkerur, A. N. Cartwright, and V. Govindaraju, "K-plet and cbfs: A graph based fingerprint representation and matching algorithm," *Proceedings of the International Conference on Biometrics, Hong King*, 2006.

[38] N. K. Ratha, V. D. Pandit, R. M. Bolle, and V. Vaish, "Robust fingerprint authentication using local structural similarity," *Workshop on Applications of Computer Vision*, pp. 29–34, 2000.

[39] M. Tico and P. Kuosmanen, "Fingerprint matching using an orientation-based minutia descriptor," *IEEE Transactions on Pattern analysis and machine intelligence*, vol. 25, no. 8, pp. 1009–1014, 2003.

[40] J. Feng, "Combining minutiae descriptors for fingerprint matching," *Pattern Recognition*, vol. 41, pp. 342–352, 2008.

[41] R. Cappelli, M. Ferrara, and D. Maltoni, "Minutia cylinder-code: A new representation and matching technique for fingerprint recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 12, pp. 2128–2141, 2010.

[42] R. Cappelli, D. Maltoni, and M. Ferrara, "Fingerprint indexing based on minutia cylinder code," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 5, pp. 1051–1057, 2012.

[43] R. Cappelli, M. Ferrara, and D. Maltoni, "Large-scale fingerprint identification on gpu," *Information Sciences 306*, pp. 1–20, 2015.

[44] M. Lopez and E. Canto, "Fpga implementation of a minutiae extraction fingerprint algorithm," *IEEE International Symposium on Industrial Electronics*, 2008.

[45] T. M. Khan, D. G. Bailey, M. A. U. Khan, and Y. Kong, "Efficient hardware implementation for fingerprint image enhancement using anisotropic gaussian filter," *IEEE Transactions on Image processing*, vol. 26, no. 5, 2017.

[46] N. Ratha, K. Karu, S. Chen, and A. Jain, "A real-time matching system for large fingerprint databases," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 18, no. 8, pp. 799–813, 1996.

[47] "Nist special database 9," *https://www.nist.gov/srd/nist-special-database-9*.

[48] A. Lindoso, L. Entrena, C. Lpez-Ongil, and J. Liu, "Correlation-based fingerprint matching using fpgas," *IEEE International conference on Field-Programmable Technology*, 11-14 Dec 2005.

[49] M. Fons, F. Fons, and E. Canto, “Design of fpga-based hardware accelerators for on-line fingerprint matcher systems,” *Research in Microelectronics and Electronics*, 12-15 June 2006.

[50] A. Lindoso, L. Entrena, and J. Izquierdo, “Fpga-based acceleration of fingerprint minutiae matching,” *3rd Southern Conference on Programmable Logic*, 28-26 Feb 2007.

[51] M. Fons, F. Fons, and E. Cant, “Fingerprint image processing acceleration through run-time reconfigurable hardware,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 57, no. 12, pp. 991–995, 2010.

[52] G. Danese, M. Giachero, F. Leporati, and N. Nazzicari, “A multicore embedded processor for fingerprint recognition,” *13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools*, 1-3 Sept. 2010.

[53] R. M. Jiang and D. Crookes, “Fpga-based minutia matching for biometric fingerprint image database retrieval,” *Journal of Real-Time Image Processing*, vol. 3, no. 3, pp. 177–182, 2008.

[54] “Ultrascale architecture and product data sheet: Overview,” *Xilinx, DS890.*

[55] “7 series fpgas data sheet: Overview,” *Xilinx, DS180.*

[56] “Ultrascale architecture configurable logic block,” *Xilinx, UG574.*

[57] “Kcu105 board user guide,” *Xilinx, UG917.*

[58] “Pci express 3.0 frequently asked questions,” *https://web.archive.org/web/20140201172536/http://www.pcisig.com/news_room/faqs/pcie3.0_faq/*.