DECLARATION OF THESIS / UNDERGRADUATE PROJECT PAPER AND COPYRIGHT

MOHD ANNUAR BIN SUHAINI

Author’s full name : __________________________________________________________

Date of birth : 28 JUNE 1986

Title : FINGERPRINT BIOMETRIC EMBEDDED SYSTEM FOR SECURITY APPLICATION

Academic Session: 2008/2009

I declare that this thesis is classified as :

☐ CONFIDENTIAL (Contains confidential information under the Official Secret Act 1972)*

☐ RESTRICTED (Contains restricted information as specified by the organisation where research was done)*

☐ OPEN ACCESS I agree that my thesis to be published as online open access (full text)

I acknowledged that Universiti Teknologi Malaysia reserves the right as follows :

1. The thesis is the property of Universiti Teknologi Malaysia.
2. The Library of Universiti Teknologi Malaysia has the right to make copies for the purpose of research only.
3. The Library has the right to make copies of the thesis for academic exchange.

Certified by :

______________________________ ________________________________
SIGNATURE SIGNATURE OF SUPERVISOR

860628-23-6001 Prof. Dr. Mohamed Khalil Mohd Hani
(NEW IC NO. /PASSPORT NO.) NAME OF SUPERVISOR

Date : 7 MAY 2009 Date : 7 MAY 2009

NOTES : * If the thesis is CONFIDENTIAL or RESTRICTED, please attach with the letter from the organisation with period and reasons for confidentiality or restriction.
“I hereby declare that I have read this thesis and in my opinion this thesis is sufficient in term of scope and quality for the award of the degree of Bachelor of Engineering (Electrical-Electronics)”

Signature : .................................
Name of Supervisor :  Prof. Dr. Mohamed Khalil Mohd Hani
Date : 7 MAY 2009
FINGERPRINT BIOMETRIC EMBEDED SYSTEM FOR SECURITY APPLICATION (FRONT-END)

MOHD ANNUAR BIN SUHAINI

A report submitted in fulfillment of the requirements for the award of the degree of Bachelor of Engineering (Electrical-Electronics)

Faculty of Electrical Engineering
Universiti Teknologi Malaysia

MAY 2009
I declare that this thesis entitled “Fingerprint Biometric Embedded System for Security Application” is the result of my own research except as cited in the references. The thesis has not been accepted for any degree and is not concurrently submitted in candidature of any other degree.

Signature : .....................................
Name : Mohd Annuar bin Suhaini
Date : 7 MAY 2009
To my beloved mother and father
ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to my supervisor, Prof. Dr. Mohammed Khalil Mohd Hani for his encouragements, patience, guidance, critics and friendship. Without his continued support and insights throughout this work, this thesis could not have been realized.

I would also like to thanks my mentor in ECAD Research Laboratory; Rabia, Jasmin, Vishnu, and Pei Chee for their endless guidance throughout my research journey. Many thanks also conveyed to the technicians in ECAD laboratory, En. Zulkifli and En. Jefri.

My utmost thanks go to my families for their caring encouragement and moral support throughout my academic year.

Lastly I would like to acknowledge, with many thanks to all my friends and whoever involve directly or indirectly, in making this thesis successful.
ABSTRACT

A very dependable personal identification is required due to the growing fact of technology abuse in the necessity of protection and access restriction. A system based on the application of statistical analysis to biological data possessed by the user can verifies user identity and is one of the most successful techniques, because of its high level of reliability and decreasing cost of fingerprint sensing devices. The main idea is to verify the person is who he claims to be. This project proposed an online fingerprint biometric embedded system that involved software development and interface design in an embedded processor. The 256 gray-scale PNM format images with the size of 192 x 192 pixels are used for this project as input from the fingerprint scanner. The system consists of two components; the Usertest embedded software and the fingerprint biometric embedded software which is executed in Altera Cyclone II Nios processor on the Field Programmable Gate Array (FPGA) platform. The Nios II system module will be generated and compiled before it is configured into the development board. Then a real time operating system (uClinux) will be downloaded and run on the development board along with the fingerprint biometric embedded software which has been cross compiled for the uClinux kernel. The fingerprint biometric embedded software contains three stages; image processing, feature extraction and matching stage. The image processing stage will enhance the fingerprint image while the feature extraction stage will extract the valid minutiae points in template based. The matching stage will match the minutiae template with a previously stored template. The fingerprint image is acquired from a fingerprint scanner device connected to the development board via USB and is controlled by a preconfigured scanning software (Usertest) which has been cross compiled for uClinux kernel. The scanning software is responsible to communicate with
the device scanner and acquire the fingerprint image from user. An embedded software system that can verify the minutiae templates extracted a fingerprint image is resulted from this project.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>CHAPTER</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>DECLARATION</td>
<td></td>
<td>li</td>
</tr>
<tr>
<td>DEDICATION</td>
<td></td>
<td>iii</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENT</td>
<td></td>
<td>iv</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td></td>
<td>v</td>
</tr>
<tr>
<td>TABLE OF CONTENTS</td>
<td></td>
<td>vii</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td></td>
<td>xi</td>
</tr>
<tr>
<td>LIST OF ABBREVIATIONS</td>
<td></td>
<td>xii</td>
</tr>
<tr>
<td>LIST OF APPENDICES</td>
<td></td>
<td>xiii</td>
</tr>
</tbody>
</table>

1 INTRODUCTION

1.1 Background 1
1.2 Problem Statement 2
1.3 Objectives 4
1.4 Scope of Works 4
1.5 Project Contribution 5
1.6 Thesis Organization 5

2 LITERATURE REVIEW

2.1 Biometric Technology 7
2.2 Fingerprint 10
2.3 Fingerprint Recognition System 12
2.4 Image Processing Stage 13
  2.4.1 Segmentation 13
  2.4.2 Binarization 14
  2.4.3 Noise Elimination 15
  2.4.4 Smoothing 16
  2.4.5 Thinning 17
2.5 RTOS 18

3 RESEARCH METHODOLOGY 20
3.1 Project Workflow 20
3.2 Tools and Software Used 22
  3.2.1 Altera Installer 22
  3.2.2 Quartus II Subscription Edition Software 23
  3.2.3 Nios II Embedded Design Suite 23
  3.2.4 Ubuntu released 8.04 (hardy) 24
  3.2.5 Linux for Nios II 24
  3.2.6 Binary toolchain 24
  3.2.7 Microsoft Visual Basic 6.0 24
  3.2.8 MATLAB 25
  3.2.9 Altera Development and Education Board 25
  3.2.10 Microsoft Visual C++ 6.0 25
3.3 Embedded System Design Flow 26
3.5 Chapter Summary 27

4 DESIGN AND IMPLEMENTATION 28
4.1 System Architecture 28
4.2 Embedded System Hierarchy 31
4.3 Fingerprint Matching Workflow 32
4.4 Fingerprint Scanner Configurations 34
4.5 Fingerprint scanner setup 35
4.6 Fixing pixel size 44
4.7 Chapter Summary 45

5 RESULTS AND DISCUSSIONS 46
5.1 Test Vector 46
5.2 Image Processing Results 47
5.3 Full Embedded System Implementation Test Result 50

6 CONCLUSIONS 52
6.1 Recommendation for Future Work 52

REFERENCES 54

APPENDIX A - C 56-105
## LIST OF TABLES

<table>
<thead>
<tr>
<th>TABLE NO.</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Comparison of Biometric Technologies</td>
<td></td>
</tr>
<tr>
<td>5.1</td>
<td>Full Embedded system implementation test results</td>
<td></td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIGURE NO.</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Ridge Ending and Ridge Bifurcation</td>
<td></td>
</tr>
<tr>
<td>2.2.1</td>
<td>Core and delta points</td>
<td></td>
</tr>
<tr>
<td>2.2.2</td>
<td>Different kinds of minutiae points</td>
<td></td>
</tr>
<tr>
<td>2.3.1</td>
<td>Fingerprint recognition system</td>
<td></td>
</tr>
<tr>
<td>2.4</td>
<td>Process in image processing stage</td>
<td></td>
</tr>
<tr>
<td>2.4.3</td>
<td>Structural element used in noise elimination process</td>
<td></td>
</tr>
<tr>
<td>2.4.5a</td>
<td>eight-connectivity neighborhoods</td>
<td></td>
</tr>
<tr>
<td>2.4.5b</td>
<td>Structuring elements used in thinning process</td>
<td></td>
</tr>
<tr>
<td>3.1</td>
<td>Project Workflow</td>
<td></td>
</tr>
<tr>
<td>4.1</td>
<td>Development board upgrade of Fingerprint Biometric Embedded System</td>
<td></td>
</tr>
<tr>
<td>4.2</td>
<td>Fully Embedded System</td>
<td></td>
</tr>
<tr>
<td>4.3</td>
<td>Embedded System Hierarchy</td>
<td></td>
</tr>
<tr>
<td>4.4</td>
<td>Embedded System Workflow</td>
<td></td>
</tr>
<tr>
<td>4.5</td>
<td>Misc Device Configurations</td>
<td></td>
</tr>
<tr>
<td>5.1</td>
<td>First and second test images (pic.pnm and pic2.pnm)</td>
<td></td>
</tr>
<tr>
<td>5.2</td>
<td>Image Segmentation</td>
<td></td>
</tr>
<tr>
<td>5.3</td>
<td>Image binarization</td>
<td></td>
</tr>
<tr>
<td>5.4</td>
<td>Image smoothing</td>
<td></td>
</tr>
<tr>
<td>5.5</td>
<td>Image thinning</td>
<td></td>
</tr>
</tbody>
</table>
### LIST OF ABBREVIATIONS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADO</td>
<td>ActiveX Data Object</td>
</tr>
<tr>
<td>AFIS</td>
<td>Automated Fingerprint Identification System</td>
</tr>
<tr>
<td>CN</td>
<td>Crossing Number</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>DAO</td>
<td>Data Access Object</td>
</tr>
<tr>
<td>EDA</td>
<td>Electronic design Automation</td>
</tr>
<tr>
<td>FPGA</td>
<td>Field Programmable Gate Array</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphic User Interface</td>
</tr>
<tr>
<td>ODBC</td>
<td>Open Database Connector</td>
</tr>
<tr>
<td>OLE DB</td>
<td>Object Linking and Embedding Database</td>
</tr>
<tr>
<td>PC</td>
<td>Personal Computer</td>
</tr>
<tr>
<td>PDA</td>
<td>Personal Digital Assistant</td>
</tr>
<tr>
<td>PIO</td>
<td>Parallel Input Output</td>
</tr>
<tr>
<td>PLD</td>
<td>Programmable Logic Device</td>
</tr>
<tr>
<td>RISC</td>
<td>Reduced Instruction Set Computer</td>
</tr>
<tr>
<td>SDK</td>
<td>System Development Kit</td>
</tr>
<tr>
<td>SoC</td>
<td>System-on-Chip</td>
</tr>
<tr>
<td>SOPC</td>
<td>System-on-a-Programmable-Chip</td>
</tr>
<tr>
<td>UART</td>
<td>Universal Asynchronous Receiver Transmitter</td>
</tr>
<tr>
<td>RTOS</td>
<td>Real Time Operating System</td>
</tr>
</tbody>
</table>
# LIST OF APPENDICES

<table>
<thead>
<tr>
<th>APPENDIX</th>
<th>TITLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Altera Design Software 8.1 Installation Manual for the Ubuntu Linux Environment</td>
<td>56</td>
</tr>
<tr>
<td>B</td>
<td>Compiling and Using uClinux with the Altera Nios II</td>
<td>66</td>
</tr>
<tr>
<td>C</td>
<td>Main Program Source Code for uClinux-Fingerprint.c</td>
<td>90</td>
</tr>
</tbody>
</table>
CHAPTER 1

INTRODUCTION

This report proposes the FPGA implementation of fingerprint biometric embedded system. The design incorporates the embedded processor to produce a complete system that can perform fingerprint image processing, feature extraction and matching process.

1.1 Background

“Traditionally, knowledge-based security with using password and token based security such as badges have been used to restrict access to secure systems. However, security can be easily breached in these systems when a password is divulged to an unauthorized user or a badge is stolen by an impostor. The emergence of biometrics has addressed the problems that plague traditional verification methods. Biometrics refers to the automatic identification or verification of an individual or a claimed identity by using certain physiological or behavioral traits associated with the person such as
fingerprints, hand geometry, iris, retina, face, hand vein, facial thermograms, signature, voiceprint, etc. Biometric indicators have an edge over traditional security methods in that these attributes cannot be easily stolen or shared. Among all the biometric indicators, fingerprint-based identification is the oldest and most popular method among all the biometric techniques being used today, which has been successfully used in numerous applications. The main reason for the popularity is of course its high level of reliability. This is because, every one is known to possess a unique fingerprint and its features remain invariant with age (Ariff, 2008).

“The embedded systems have become increasingly popular as advances in ICtechnology and processor architecture allow for flexible computational parts and high-performance modules integrated on a single carrier. Embedded system interacts with the physical world. It executes on machines that are not, first and foremost, computers. They are cars, airplanes, telephones, audio equipment, robots, appliances, toys, security systems, pacemakers, heart monitors, weapons, television sets, printers, scanners, climate control systems, manufacturing systems, and so on. They performed function carefully partitioned in software and hardware to strike the balance between flexibility, reusability, performance and cost (Ariff, 2008)”.

1.2 Problem Statement

As mention earlier, it is important to have reliable personal identification due to growing importance of information technology. A biometric security system is the key task to the automated security system that use a specific physiological or behavioral characteristic possessed by a user.
There are two main aspects to be considered to design a fingerprint biometric system which are the technique to be used and the system designed. As for the first one, the challenge is to have reliable algorithm to extract the biometric data from a fingerprint image. This is because the fingerprint image slightly varies each time the image is taken due to the factor of the user’s finger position placed on the scanner even though the minutiae point remain unchanged. Moreover, there still could be other factors preventing a good image capture such as oily finger and anonymous substance on fingerprint. In addition, there are two types of features to be extracted from fingerprints: local and global. The global features are the cores and deltas points while the local representations predominantly based on ridge endings or bifurcations. The two local features are collectively known as minutiae as shown in Figure 1.1.

Figure 1.1  Ridge Ending and Ridge Bifurcation

The second aspect that should be considered is that the system design can be trusted. The embedded system must not compromise the identification of the protected individual and must not fail for security breach.
1.3 Objectives

From the discussion above, this thesis sets out two objectives:

1. To enhance the previous fingerprint biometric embedded system into online fingerprint recognition system that use fingerprint scanner with database to store the minutiae templates.

2. To implement the complete fingerprint biometric system in an embedded processor on Altera FPGA (Cyclone II) DE2 development board as embedded design prototype.

1.4 Scope of Works

1. System is online

2. Able to Receive fingerprint images from the fingerprint scanner

3. Uses 256 gray-scales PNM images, 192 x 192 pixel input

4. Implemented on Altera Cyclone II FPGA; DE2 development board

5. Running RTOS on embedded system

6. Generated image files in each stages to display output
1.5  Project Contribution

1. An upgrade to the new development board (DE2 board) was successfully implemented and now the project can use all the features the board has to offer.

2. The fingerprint device scanner was successfully integrated to the development board and user can get the fingerprint image directly from the scanner without the host PC interface. This’ll allow the implementation of fully embedded system.

3. The fingerprint biometric embedded software was change to run on top of Operating System instead of operating as firmware on Nios II. It means the same code can be used for PC simulation as well as the real deal on embedded system. This will allow the same source code to be compiled for different system architecture, Nios II or IA-32 without any changes to the code.

4. The resolution of the captured image was increased to 192 x 192. With larger resolution, more minutiae point can be captured.

1.6  Thesis Organization

This thesis is organized into six chapters. The first chapter introduced the motivation, research objectives, scope of work and contribution of this project.
Chapter 2 reviews the background of the project. Algorithms used in developing the system are also explained in this chapter.

Chapter 3 presents the research methodology, system design procedures and application tool that have been used in this project.

Chapter 4 described the software development of the system and implementation of the software in embedded system.

Chapter 5 presents the result for each stage in the system and discussion of the overall result.

In the final chapter, the research work is summarized and the potential future works are given.

All the critical steps for reimplementation of this project are included in the Appendix.
LITERATURE REVIEW

This chapter discusses the background of biometric technologies, fingerprints and minutiae point. The algorithms used in a fingerprint recognition system are also explained in detail. Finally is the introduction on Field Programmable Gate Array (FPGA) and Altera DE2 development board.

2.1 Biometric Technology

In the science and technology of measuring and analyzing biological data, Biometric technologies are defined as “automated method of identifying or authenticating the identity of a living person based on a physiological or behavioral characteristic.” (Bowman, 2000)
The term biometric device in the access control industry implies that three major components are present:

- A mechanism to scan and capture a digital or analog image of a living personal characteristic
- Compression, processing and comparison of the image.
- Interface with application system

These pieces can be configured in a variety of ways for different situations. The stored images or reference templates can reside on a card, in the device or at a host. Identification and authentication is the most important aspect of biometrics. Identification occurs when an individual’s characteristics are being selected from a group of stored images. The question put to the machine is “do I know you?”. It is called a “one-to-many” search. The search algorithm will search a database and return with a likely list of candidates in a matter of minutes. Law enforcement is the most popular application of identification device. The AFIS (automated fingerprint identification system) can perform over 100,000 fingerprint match attempts in a second (Maio et al, 2003).

Authentication occurs when an individual makes a claim of identity by presenting a code or a card. The question put to the machine is “Are you who you claim to be?”. It is called a “one-to-one” search. The individual characteristics are being measured against an enrolled image that stored on a token or in a local database with the image presented. The search time and subsequent authentication are much faster than identification because the person must present a pin or password as an index for authentication (Maio et al, 2003).
A physiological characteristic is a relatively stable physical characteristic, such as fingerprint, hand silhouette, iris pattern or blood vessel on the back of the eye. This type of measurement is basically unchanging and unalterable without significant force.

A behavioral characteristic is more a reflection of an individual’s psychological makeup although general physical traits such as size and sex have a major influence. The most common behavioral traits used are signature. Other behavior that can be used are how one type at the keyboard and how one speak.

This physiological or behavioral characteristic used for identification actually follows some requirements. Table 2.1 shows a comparison of biometric technologies using these requirements.

- Universality: Every person must have the characteristic.

- Uniqueness: Two distinct people can not have the same characteristic.

- Permanence: This characteristic can not change according to the time.

- Collectability: This characteristic can be measured quantitatively.

- Performance: The identification process must present an acceptable result.

- Acceptability: Indicates to what extent people are willing to accept the biometric system.

- Circumvention: Refers to the ability to get destroyed.
Table 2.1: Comparison of Biometric Technologies

<table>
<thead>
<tr>
<th>Biometrics</th>
<th>Universality</th>
<th>Uniqueness</th>
<th>Permanence</th>
<th>Collectability</th>
<th>Performance</th>
<th>Acceptability</th>
<th>Circumvention</th>
</tr>
</thead>
<tbody>
<tr>
<td>Face</td>
<td>High</td>
<td>Low</td>
<td>Medium</td>
<td>High</td>
<td>Low</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Fingerprint</td>
<td>Medium</td>
<td>High</td>
<td>High</td>
<td>Medium</td>
<td>High</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>Hand Geom.</td>
<td>Medium</td>
<td>Low</td>
<td>Low</td>
<td>High</td>
<td>Low</td>
<td>High</td>
<td>Medium</td>
</tr>
<tr>
<td>Hand Vein</td>
<td>Medium</td>
<td>Medium</td>
<td>Medium</td>
<td>Medium</td>
<td>Medium</td>
<td>Medium</td>
<td>Low</td>
</tr>
<tr>
<td>Iris</td>
<td>High</td>
<td>High</td>
<td>High</td>
<td>Medium</td>
<td>High</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>Retina</td>
<td>High</td>
<td>High</td>
<td>Medium</td>
<td>Low</td>
<td>High</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>Ear</td>
<td>Medium</td>
<td>Medium</td>
<td>High</td>
<td>Medium</td>
<td>Medium</td>
<td>High</td>
<td>Medium</td>
</tr>
<tr>
<td>Signature</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
<td>High</td>
<td>Low</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Voice</td>
<td>Medium</td>
<td>Low</td>
<td>Low</td>
<td>Medium</td>
<td>Low</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Thermogram</td>
<td>High</td>
<td>High</td>
<td>Low</td>
<td>High</td>
<td>Medium</td>
<td>High</td>
<td>High</td>
</tr>
</tbody>
</table>

2.2 Fingerprint

“A fingerprint can be looked at from different levels: the global level, the local level and the very fine level. At the global level, you find the singularity points, called core and delta points. These singularity points are very important for fingerprint classification, but they are not sufficient for accurate matching. Figure 2.2.1 shows the core and delta points of two fingerprint’s pattern; loop and whorl. Loops have one delta, whorl have two (Farizan, 2006)”.
“At the local level, you find the minutiae details or sometimes called the minutiae points. Minutiae mean small details. In the context of fingerprint, minutiae refer to various ways that the ridges in a fingerprint can be discontinuous. For example, a ridge can suddenly come to an end. Such a feature is naturally called a ridge ending. A ridge that divides into two ridges is called a bifurcation. Figure 2.2.2 shows four different kind minutiae points (Farizan, 2006).”
“In order to establish comparison between fingerprint images, Automated Fingerprint Identification System’s often rely on procedures based on local features. Global features are mainly employed to reduce the computational cost associated to fingerprint comparison procedure (Farizan, 2006).”

2.3 Fingerprint Recognition System

“The architecture of a fingerprint recognition system is shown in Figure 2.3.1 A typical fingerprint recognition system consists of four components: User Interface, system database, enrollment module and authentication module (Farizan, 2006).”

![Figure 2.3.1: Fingerprint recognition system](image)

“The user interface provides a mechanism for a user to indicate his/her identity and input his/her fingerprint into the system. The system database consists of a collection of records, each of which corresponds to an authorized person that has access to the system. The task of enrollment module is to enroll persons and their fingerprints
into the system database. The task of the authentication module is to authenticate the identity of the person who intends to access the system (Farizan, 2006)

### 2.4 Image Processing Stage

The goal of image processing stage is to filter, binarize, enhance and skeletonized the original gray-level image. Five different processes are sequentially applied to achieve this goal. Figure 2.4 shows the process required in this stage.

![Image Processing Stage Diagram](image)

**Figure 2.4:** Process in image processing stage

#### 2.4.1 Segmentation

The first step of the image processing stage is image segmentation. Image segmentation is the process of separating the foreground regions in the image from the background regions. In a fingerprint image, the background regions generally exhibit a very low grey-scale variance value, whereas the foreground regions have a very high variance. Hence, a method based on variance thresholding can be used to perform the segmentation. (Mehtre, 1993)
“Firstly, the image is divided into blocks and the grey-scale variance is calculated for each block in the image. If the variance is less than the global threshold, then the block is assigned to be a background region; otherwise, it is 17 assigned to be part of the foreground. The grey-level variance for a block of size $W \times W$.

$W$ is defined as:

$$V(k) = \frac{1}{W^2} \sum_{i=0}^{W-1} \sum_{j=0}^{W-1} (I(i,j) - M(k))^2$$

(2.1)

Where $V(k)$ is the variance for block $k$, $I(i,j)$ is the grey-level value at pixel $(i,j)$, and $M(k)$ is the mean grey-level value for the block $k$ (Farizan, 2006).

### 2.4.2 Binarization

Binarization is the process that converts a gray scale image, which has 256 of gray-level (0 to 255) to a black and white (0 and 1) binary images. The average gray value can be used as a threshold value for the binarization process. If the gray value of a particular pixel is lower than the threshold value, then the new value assigned for that pixel is ‘0’ (representing the ridges), else the new value is set to ‘0’ (representing the valleys). The average gray value is determined by

$$V_{mean} = \frac{1}{r \times c} \sum_{x=0}^{r} \sum_{y=0}^{c} V(x,y)$$

(2.2)
2.4.3 Noise Elimination

“Noise elimination is the process that removes all the undesired pixels in the image (black pixel that occur as noise in the image). Noise elimination process uses a combination of two morphological method, erosion and dilation. Eroding the image using structuring element, identifies and mark the location of the noise in the image. After all the location have been identified and marked, the noise will be removed by dilating the image with the same structuring element (Farizan, 2006)”.

Erosion can be determined by;
\[
A \Theta B = \{z | (B)_z \subseteq A \} \tag{2.3}
\]

Dilation can be determined by;
\[
A \oplus B = \{z | (B)_z \cap A \neq \emptyset \} \tag{2.4}
\]
2.4.4 Smoothing

“Smoothing process seeks to fill up all the holes that exist in the ridges while at the same time preserving all the essential details that an observer wish to see in the original image. Smoothing used a combination of two morphological methods, opening and closing.

An opening is erosion following by dilation with the same structuring element. Opening is determined by

\[ A \circ B = (A \ominus B) \oplus B \]  

(2.5)
A closing is dilation following by erosion with the same structuring element. Closing is determined by

\[ A \ast B = (A \oplus B) \ominus B \]  

(2.6)

(Farizan, 2006)”.

### 2.4.5 Thinning

“Thinning process is used to skeletonized the binary image by reducing all lines to a single pixel thickness. The approach iteratively deletes edge point pixels from the region until just the skeleton remains.

A pixel p0 is defined to have at least one pixel in its eight-connectivity neighborhood that belongs to the background. The eight-connectivity neighborhood is shown in Figure 2.4.5(a).

![eight-connectivity neighborhoods](image)

**Figure 2.4.5(a):** eight-connectivity neighborhoods
As mentioned above, the algorithm had to be modified to apply to fingerprint ridge thinning. The problem lies in what is defined to be a one-pixel width skeleton. In the case of fingerprint ridges a ridge point that is not a minutia is only allowed to have two neighbors that belong to the ridge. This fact conflicts with the second condition in the original thinning algorithm. The problem arises in 16 special cases where not all neighbor pixels, which belong to the background, are connected but where the pixel still should be deleted (Farizan, 2006)’

A more detailed explanation of the fingerprint image processing can be found on the index and was discuss in the previous project thesis.

2.5 RTOS

“A Real-Time Operating System (RTOS) is a multitasking operating system intended for real-time applications. Such applications include embedded systems (programmable thermostats, household appliance controllers), industrial robots, spacecraft, industrial control and scientific research equipment.
A RTOS facilitates the creation of a real-time system, but does not guarantee the final result will be real-time; this requires correct development of the software. An RTOS does not necessarily have high throughput; rather, an RTOS provides facilities which, if used properly, guarantee deadlines can be met generally (soft real-time) or deterministically (hard real-time). An RTOS will typically use specialized scheduling algorithms in order to provide the real-time developer with the tools necessary to produce deterministic behavior in the final system. An RTOS is valued more for how quickly and/or predictably it can respond to a particular event than for the amount of work it can perform over a given period of time. Key factors in an RTOS are therefore minimal interrupt latency and a minimal thread switching latency. (Wikipedia)"
CHAPTER 3

RESEARCH METHODOLOGY

This chapter describes the methodology used in this project. It begins with the discussion of the project workflow, followed by the system design and tools that are applied in this work.

3.1 Project Workflow

This project involved the effort of embedded system design process, which involves interface design and software development. Hence, it calls for embedded software system design, in which the software are designed and downloaded in an embedded system. The overall project workflow is shown in Figure 3.1.

It all began with the search for motivation to do this project. Hence the research problems were identified. The need of an automated security system that applied the
biometric technologies on person identification to protect the information has been the problems that need to be solved in this project.

![Project Workflow Diagram]

**Figure 3.1:** Project Workflow

Once the problems have been identified, a plan was developed to find the most conductive solutions to a favorable outcome. The works was divided into three parts literature, implementation and documentation and these works were done concurrently.
The work on literature involved studies on previous thesis, algorithm involved, C programming, tutorial on CAD tools and understanding the development boards used in previous work and in the future work.

The implementation of this project involved the recreation of the previous prototype. Then all the results related was collected and analyze to recreate the prototype on new environment followed by a test on the new prototype. If the prototype fails the test, the problem will be studied and the process of recreate the prototype continues, else the system will be refine.

The documentation of this project involved the proposal, weekly progress report and report writing.

### 3.2 Tools and Software used

In order to aid in the design and implementation, several development tools, EDA tools and a computer is needed to conduct this project.

#### 3.2.1 Altera Installer

To reduce the size of device family download file by up to 50 percent, and also to enable installation in multiple computers.
3.2.2 Quartus II Subscription Edition Software

Altera Quartus II is a design entry, compilation, simulation and implementation software tool. It is a comprehensive environment for system-on-a-programmable-chip (SoPC) design, which fully automates the process of configuring processor features by generating the interconnect logic to integrate the components in the hardware system. SoPC Builder specifies the Nios processor core(s), memory, and other peripherals our system requires. SoPC Builder can also import a designer’s HDL design files, providing an easy mechanism to integrate custom logic into a Nios processor system. (Altera, 2003d)

3.2.3 Nios II Embedded Design Suite

The Nios II Embedded Design Suite (EDS) is a collection of cutting-edge software tools, utilities, libraries, and drivers to help design with the Nios® II embedded processor. Included in the design suite is the Nios II IDE which is the primary software development tool for the Nios II family of embedded processors. Nios II IDE key features are new project wizards and software templates, Compiler for C and C++ (GNU), Source navigator, editor, and debugger, Eclipse project-based tools, Software build tools, Nios II C-to-Hardware (C2H) acceleration compiler and complete documentation and training that helps reduce the design time.
3.2.4 Ubuntu released 8.04 (Hardy)

Ubuntu is a community developed, Linux based operating system that is perfect for laptops, desktops and servers. It contains all the common applications.

3.2.5 Linux for Nios II

uClinux was a derivative of Linux 2.0 kernel intended for microcontrollers without Memory Management Units (MMUs). Later the patches of uClinux were merged back to the mainstream Linux 2.6 kernel. Today's uClinux as an operating system includes Linux kernel releases for 2.0 2.4 and 2.6 as well as a collection of user applications, libraries and tool chains.

3.2.6 Binary Toolchain

Debian build toolchain is a collection of software utilities used to create Debian source package (.dsc) and debian binary packages (.deb) from upstream source tarballs.

3.2.7 Microsoft Visual Basic 6.0

A window-based compiler used to develop the GUI. The developed GUI will get data from the fingerprint scanner and form the fingerprint image before sending it to the development board. Each button in the GUI will invoke an action to be done on the host pc or a signal to be sent to FPGA development board and invoke a subroutine in
FingerPrint.c. The GUI will also create database from the templates for each enrolment stages.

### 3.2.8 MATLAB

The MATLAB provides a high-level programming language, an interactive technical computing environment, and functions for algorithm development, data analysis and visualization and numeric computation. It is mainly used as a test tool for the study of the image processing algorithm. Any new algorithm discovered was tested first in MATLAB environment and if the algorithm suitable for the system then the algorithm will be written in C programming language.

### 3.2.9 Altera DE2 Development and Education Board

The Altera® DE2 development and education board provides an ideal vehicle for learning about digital logic, computer organization, and FPGAs. Featuring an Altera Cyclone® II FPGA, the DE2 board offers state-of-the-art technology suitable for university and college laboratory use, a wide range of design projects, as well as sophisticated digital system development can be use as hardware platform to prototype embedded systems.

### 3.2.10 Microsoft Visual C++ 6.0

A window-based compiler used to develop the GUI. The developed GUI is pending for the user to load an input image and display the result of each processing modules. Each
button in the GUI will invoke an action to be done on the host pc or a signal to be sent to FPGA development board and invoke a subroutine in FingerPrint.c.

3.3 Embedded System Design Flow

The Nios embedded processor is a soft core CPU optimized for in system-on a programmable-chip (SOPC) designs implemented in programmable logic devices (PLD). Nios processor designs are created using Altera's SOPC Builder system design tool. This processor is a five-stage pipeline with separate instruction and data memory masters RISC microprocessor. The Nios processor supports both 32-bit and 16-bit architectural variants and both 16 and 32-bit variants use 16-bit instructions. (Altera, 2003a). In this work, a standard 32-bit Nios configuration has been chosen.

The system design procedure applied in this work starts with generating standard 32-bit Nios II system. After the standard 32-bit Nios II system has been pre-configured, a linux distribution, uClinux was configured for the system. This includes the installation for the fingerprint scanner driver. Then the fingerprint biometric embedded software was changed to run on top of uClinux. After that, the fingerprint scanning software compiled for uClinux. These softwares then included into /bin directory of uClinux followed by generation of zImage of the RTOS. Finally, the uClinux zImage file was downloaded into the Nios II system. This will be discuss further in Chapter 4.
3.4 Chapter Summary

In conclusion, this chapter has briefly introduced the overall project workflow and described the tools and software needed in designing the fingerprint biometric system. Besides, Nios system module has been explained with the embedded system design flow followed by the software development.
CHAPTER 4

DESIGN AND IMPLEMENTATION

This chapter will discuss about the front end implementation of the system in Nios II embedded processor, the hardware design of the system in the new development board and the software design on image processing stage which is executed in Nios soft core as embedded design.

4.1 System Architecture

The fingerprint biometric embedded system proposed in this project is the automatic fingerprint recognition software. It is executed in Nios II soft core processor programmed in the Altera’s Cyclone II FPGA on the DE2 development board.

For the initial work, the architecture of the Fingerprint Biometric Embedded System will follow the previous system. The details on each processing modules in
terms of design, implementation and source codes can be referred at the thesis from Ariff, 2008. The change made is the development board. This is the first step to change the environment of the system before a more advanced upgrade implemented. As before, the Nios II system will be loaded into the board. Then the fingerprint biometric template system (FingerPrint.c) is downloaded onto Nios II CPU into the system module. The DE2 development board will communicate with host PC through UART serial communication channel. A fingerprint scanner is connected to the host PC to make it an online system. The architecture is shown in figure 4.1.

![Figure 4.1: Development board upgrade of Fingerprint Biometric Embedded System](image)

After the upgrade in development board environment successfully achieve, then only the works on the fully embedded system can continue. For a fully embedded system, the fingerprint scanner must be connected to the DE2 development board and the fingerprint database must be stored into a storage outside the host PC preferably in case of this prototype, into the DE2 development board. The fully embedded system is shown in figure 4.2.
As mention before, the storage for this prototype is on the development board. However, it only provides a small storage only suitable for a small database. It is more suitable as a testing ground for a prototype but in a real implementation designer may want to consider a bigger storage for the database.

The fully embedded system also takes images directly from the fingerprint scanner via the standard USB interface. This is done using the uCLinux as the Linux kernel for microcontrollers with support for USB and Nios architecture. For this, the kernel version 2.4.0 is needed. However USB 2.0 support is available in kernel 2.5.2 or later. The kernel will be downloaded then run by the Nios CPU. Linux USB does have a generic scanner driver that provides the communication link between the device and user space. But in this design, the original scanner driver was modified to work with uCLinux and to produce 192 x 192 pixel PNM image file.
4.2 Embedded System Hierarchy

Before move on to the details on RTOS applications, one must understand the hierarchical position of these applications. The design complexity reduced in the higher level and vice versa for lower level. Shown below is the embedded system hierarchy.

Figure 4.3: Embedded System Hierarchy

As seen above, Nios II is the soft-core processor used for this embedded system. Their related file is SoC.sof. The file is downloaded into DE2 board to run as embedded system hardware that uses Altera Cyclone II FPGA as system on chip. On top of Nios II, a real time operating system known as uClinux resides. RTOS is much more like a manager to an embedded system. It manages all the system resource such as processor, memory, I/O ports and fingerprint scanner.
The top levels of this embedded system are the Fingerprint matching algorithm and scanner interface. These applications can be run separately by just calling it in the uClinux terminal or user can write a script to run them procedurally. Script is a very powerful tool in linux. It’s more like a batch file in Windows. Both have the same function. Script is used in this project to run the applications. Since the applications are compiled separately, script is used to run them automatically.

4.3 Fingerprint Matching Workflow

This section discuss about the workflow of the test program for fingerprint matching used in this project. The figure below shows the software module on the lefthand side and its output image produced on the right hand side.
Begin with test, it will interface with the fingerprint scanner and get the first fingerprint image named pic.pnm (first test image). Then test2 also will get the second fingerprint image named pic2.pnm (second test image). The reason there are two applications to get these image is because these two applications will named the file differently. User can also change usertest.c so that with one application, it can produce two different file names for two test image. These images will be the input for uClinux-Fingerprint application. This application will read these two images and show its matching percentage. This is where the image processing algorithm is. This Application also produced the output images for each steps in image processing. The files are image1-segmentation.pnm, image1-binarization.pnm, image1-smoothing.pnm, image1-thinning.pnm, image2-segmentation.pnm, image2-binarization.pnm, image2-smoothing.pnm, image2-thinning.pnm.

**Figure 4.4:** Embedded System Workflow
thinning.pnm, image2-segmentation.pnm, image2-binarization.pnm, image2-smoothing.pnm and image2-thinning.pnm. These images is created in /home directory in when running uClinux. Finally, the matching percentage result is displayed in the terminal.

As stated above, the embedded system software (Fingerprint.c) was change to run on top of operating system. This change is necessary so that the software that contains the images processing algorithm can run on uClinux. Furthermore, with these changes the same code can be used for PC simulation as algorithm testing ground without actually having the embedded system.

This is done by changing some flow of the program. The input from UART communication has been removed and replaced with the input from the terminal. The input image is taken directly from the saved location in /home directory. The output is the image produced after each images processing stage was done.

4.4 **Fingerprint Scanner Configurations**

The fingerprint scanner used in this project is Authentec-AES2510B codename FR-25 Sensor. This fingerprint scanner used integrated USB interface. It works well with Linux operating system, tested on Ubuntu 8.10 (Hardy). The driver of this scanner is fully supported by Authentec and can be downloaded at their website.
As for the scanner configuration, the driver provided is for PC used. The objective is to use the fingerprint scanner on embedded system. This can easily be done by having an operating system (uClinux) on the embedded system itself. But still there are lots of configurations required. To make it work on embedded system, the driver must be recompiled for the uClinux kernel. Then the scanning software (usertest.c) to access the scanner must be configured and recompiled for the uClinux kernel too. Only then, the scanner can be use on embedded system.

4.5 Fingerprint scanner setup

This step by step tutorial shows you the details for setting up the fingerprint scanner. First of all we will go through the requirement for this setup.

Requirements:

1. Fingerprint scanner driver (in the DVD)
2. Authentec AES2501B fingerprint scanner
3. Have done tutorial on Altera Software Installation (Appendix A) and Compiling and using uClinux (Appendix B)
4. Enabling USB and file system support for uClinux (Appendix B)

Compiling scanning software for uClinux

In order to use the fingerprint scanner, appropriate software needed to interface with the device. This software is included with the fingerprint scanner driver in the
DVD. The software, usertest.c must be compiled for uCLinux so that user can interface with the fingerprint scanner in embedded system environment. This tutorial assumes that user have already understood the process and requirement of configuring and compiling source code for uCLinux which are taught in Appendix B in the section Cross Compiling Program for uCLinux.

User must change the scanning software source code to change the preconfigured settings. The scanning software used in this project was configured to access the device in /dev/aes25010 of uCLinux directory and store the captured image in /home directory. Shown below are the source code and its command line of configurations:

```c
/**
 * AES2501 Device Driver
 * Userspace test program
 *
 * Compile with:
 * gcc -I. -o usertest usertest.c
 */
#include <aes2501.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>

#define BUFFER_SIZE 256

int main(int argc, char *argv[]) {
```
int fd_in;
int fd_out;
char buffer[BUFFER_SIZE];
size_t len;

fd_in = -1;
fd_out = -1;

int data, rdata;

//point to the corresponding device node
fd_in = open("/dev/aes25010", O_RDONLY);

if (fd_in == -1) {
    perror("open() -- ");
    goto err;
}

//choose the place to save the fingerprint image
fd_out = open("/home/nuar/Pictures/pic.pnm", O_CREAT | O_WRONLY);

if (fd_out == -1) {
    perror("open() -- ");
    goto err;
}

data = 0x55555555;
ioctl(fd_in, AES2501_IOC_TEST, data);
//ioctl(fd_in, CASE2, &rdata);

printf("IOCTL test: written: \"%x\" - received: \"%x\n\", data, rdata);
/* Write the fingerprint */
while ((len = read(fd_in, buffer, BUFFER_SIZE)) > 0)
    write(fd_out, buffer, len);

system ("chmod 777 /home/nuar/Pictures/pic.pnm");

close(fd_in);
close(fd_out);

return EXIT_SUCCESS;

err:

if (fd_in != -1)
    close(fd_in);

if (fd_out != -1)
    close(fd_out);

return EXIT_FAILURE;
}

After preferred configurations have been made, the source code is now ready for cross compilation.

nios2-linux-uclibc-gcc usertest.c –o test –elf2flt

This command will cross compile usertest.c and produce an executable file named test. Test then copied into ~/uClirunx-dist/romfs/bin directory. When the uClirunx
is properly downloaded into the board and the terminal has been invoked, the test program can be access directly anywhere in any directory.

The installation process

1. Copy these files into ~/nios2-linux/linux-2.6/drivers/misc. In this project it was placed in home/nuar/Program/nios2-linux/linux-2.6/drivers/misc.

   • aes2501.c
   • aes2501.h
   • aes2501_regs.h

2. Open Kconfig in text editor (right click>Open width text editor).

3. Put these configuration into the file right under ‘if MISC_DEVICES’ line.

   ```
   config AES2501
      tristate "example AES2501 module"
      help
      Enable example AES2501 module.
   ```

4. Open Makefile and add this command into the last line of the file.
Save and close the file.

5. Open /uClinux-dist directory in terminal then run make menuconfig.

```
make menuconfig
```


6. The Linux Kernel Configuration interface will appear. Enter Device Drivers option. Enable the Misc devices by checking the box (e.g. [*] Misc devices). Then enter the Misc devices menu. After it is been done, you can see the AES2501 module driver listed.

```
obj-$(CONFIG_AES2501) += aes2501.o
```
Choose the driver then exit and save your new kernel configuration.

7. All the settings are now done. You can begin to compile the uClinux kernel. Run this code in terminal at ~/uClinux-dist directory.

```
make
```

The compiling process takes some time, please calmly wait. There should be no error while compiling. After the process is done, make the zImage file.

```
make image
```

As stated before, you must ensure that no error occurs while producing the zImage file or you would have to check your steps.
8. When the zImage file is produces you’re now ready to download it into the development board. First of all, please configure/download the related sof file into the development board and connects the fingerprint scanner to the DE2 board. Then, change directory to ~/uClinux-dist/images. You can directly download the zImage file into the board with the commad:

```
nios2-download -g zImage
```

9. Invoke the terminal using this command

```
nios2-terminal
```

While uClinux is loading, you can see the list of device it loads. If you already attach the fingerprint scanner to the DE2 board, you can see the device details while loading uClinux. You can also attach the fingerprint scanner after uClinux finished loading. If you do so, the terminal will show the device is being connected.

10. Make a device node for the scanner. This depends on the configuration made in usertest.c. If user decide to access the scanner at /dev/aes25010 location, the device node name aes25010 must be created in /dev directory. There are two options to create the device node. First, you can create it while running uClinux. Second, you can create it before creating zImage. First choice is temporary. The node created will be deleted after the power is out or uClinux is reloaded. On the other hand, the second choice is permanent. The node created in ~/romfs/dev directory so that every time zImage is created, the node is already there.

Temporary-created after uClinux is loaded.

```
mknod /dev/aes25010 c 180 192
```
Permanent-created in user romfs directory. Please point to the right ~/romfs/dev directory. In this project it is created as

```
mknod /home/nuar/Program/nios2-linux/uClinux-dist/romfs/dev/aes25010 c 180 192
```

After the node is created for permanent node, the zImage must be remake so that the node can be in the zImage file. Then load the zImage into DE2 board and run uClinux terminal.

Where:
aes25010 is the node name (can be anything, but must be same with the one in usertest.c).
   c is device characteristic
   180 is device major number
   192 is device minor number

Detail is in the manual page of mknod command.

11. To use the scanner, simply enter test (name of the scanning software. Should be in /bin uClinuxdirectory) in the uClinux terminal and it’ll start the process to scan. You may want to scan your finger slowly about 10 seconds because the system need more time to capture the image on a slower system (embedded). The image file is saved in the directory configured in usertest.c in this project, it is in /home uClinux directory.

This installation steps refer to the nios wiki web site at:
www.nioswiki.com/OperatingSystems/UClinux/ModuleProgramming
4.6 Fixing pixel size

As stated above, this project used 192 x 192 pixel PNM image as input. Initially, the scanner allows user to capture 192 x h pixel PNM image where h is the height of the image depending on the number of strips the scanner able to capture. The image size is fixed to be perfect square because the algorithm can only accept this type of image only. The fixed pixel size can be implemented in various ways, but in this case the changes were made to the scanner’s driver.

The scanner’s driver (aes2501.c) is studied and there is a function to calculate the height of the fingerprint image. The function will return the height of the fingerprint image based on the number of strips calculated. The number of strips calculated depends on the time taken to swap the finger. Neglecting all the details, by fixing the height (192) as the function’s return, the image size will always be 192 x 192.

Actually it is rather simple if you know the function that calculates the image height. The function name is “static unsigned assemble”. This function assembles frames to single image and return image height. When the function returns, it will return image_height value to its caller. Thus, by changing the image height to the value you want, the driver will return the desired image height.
4.7 Chapter Summary

This chapter discusses the front end implementation of this project. The system architecture of this project implements an online fully embedded system that uses uClinux as the real-time operating system that runs on Nios II system where the system is downloaded to the DE2 board and the uClinux terminal is shown on the host PC. The fingerprint scanner is directly connected to the embedded system via USB. The scanner driver was changed and preconfigured to run for uClinux. There are also some changes made to the embedded system software so that it can run on uClinux.
CHAPTER 5

RESULTS AND DISCUSSION

This chapter reveals the results and discussions for the fingerprint biometric embedded system. This is mainly to test the new system in term of functionality and trustworthy. The objective is to test the new system by comparing the same fingerprint image captured twice in online mode.

5.1 Test Vector

The test vector will be two fingerprint images. For the first test image, it is captured using ‘test’ program on uClinux. This program will produce a fingerprint image named pic.pnm. For the second test image, it is captured using the ‘test2’ program on uClinux. This program will produce a fingerprint image named pic2.pnm. These two images will be use in uClinux-Fingerprint program as the input image for matching test.
Figure 5.1: First and second test images (pic.pnm and pic2.pnm)

5.2 Image Processing Result

In uClinux-Fingerprint application, an output image will be produced each time the image is processed in all critical stages. Starting with the first input image (pic.pnm) followed by the second input image (pic2.pnm). After this process is completed, the matching process begins. It compares the extracted minutiae point from pic.pnm and pic2.pnm and finds any similarity and records it. This data then used to calculate the matching percentage. Below are the output images.
Figure 5.2: Image Segmentation

Figure 5.2 show the image output from the segmentation process. We can clearly see that the unwanted black background has been eliminated.

Figure 5.3: Image binarization

Figure 5.3 show the image output from the binarization process. In this process, all the 256 gray scale level were converted to 2 black and white scale level.
Figure 5.4: Image smoothing

Figure 5.4 shows the image after smoothing process. We can see that the holes in the fingerprint image are now covered.

Figure 5.5: Image thinning

Figure 5.5 show the output image from the thinning process. In this process, the image edge was skinned until 1 pixel thick.
5.3 Full Embedded System Implementation Test Result

These are the results from the experiment of the new architecture of fingerprint biometric embedded system.

Table 5.1: Full Embedded system implementation test results

<table>
<thead>
<tr>
<th>TEST VECTORS</th>
<th>PERCENTAGE OF MATCHING</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Live scan of the same two fingerprint images</td>
<td>65.25345861</td>
</tr>
<tr>
<td>2. Live scan of different two fingerprint images</td>
<td>23.783965733</td>
</tr>
<tr>
<td>3. Offline matching between the same fingerprint image file.</td>
<td>100.00000000</td>
</tr>
<tr>
<td>4. Offline matching between two slightly different image positions cropped from the same fingerprint image.</td>
<td>83.65829503</td>
</tr>
<tr>
<td>5. Offline matching between two completely different fingerprint image</td>
<td>35.45793487</td>
</tr>
</tbody>
</table>

From table 5.1 we can see the results are as expected. One thing that good to keep in mind is that this is the best result obtain from the test, meaning that the result can always change for worst. The most reliable result obtains were from the offline matching process. It gives results that follow the expected pattern to its test vector. So in offline matching process, the same fingerprint images will give higher matching percentage and the different fingerprint image will give lower matching percentage and
this pattern is reliable. However in live test, the test result can always change. Sometimes a faulty matching percentage obtained.

There is only one good explanation for this phenomenon. It is because the image processing algorithm does not have alignment feature. Image alignment problem happens in live scanning. It is because of the placing of the test finger on the fingerprint device. Each time image is taken, the finger is placed differ from its original position and angels. This problem does not occur in offline matching between the fingerprint image file because the same image file was used and cropped to produce another file. Thus, in order to have robust fingerprint scanning software, the alignment problem must be eliminated.
It is concluded that the system in this project is a front-end fully embedded online fingerprint biometric system. It is an automated fingerprint recognition system implemented on Altera DE2 development board and the board is connected directly to the fingerprint scanner via USB and the outputs are the processed images files. This new system works but unfortunately the fingerprint image processing algorithm does not have the feature to make the system more robust for online implementation.

This project also intended to improve the image processing software to produce a more robust system. However it is only achievable if the alignment problem is eliminated.

6.1 Recommendation for future works

There are two major problems in this project that can be repaired. First is the execution speed of the embedded software. The program takes around 1 minute to
complete the processing and matching process. It is a little too long to wait. The best thing to do here is to migrate the image processing algorithm from software into hardware. With the algorithm running as hardware, the process could be much faster and consumes less clock cycles.

Secondly, the alignment problem must be eliminated. This is to ensure the system is robust and reliable. This can be done by adding the alignment features into the feature extraction stage so that the minutiae point extracted does not affected by the alignment problem.

There are also some redundancies in the algorithm coding. If we look at the coding, it is long and tedious. A good program is a program that is optimized. So, an optimized program works well and faster plus uses little resources. This is very important if the program is created for embedded system because embedded system’s resource is limited.
REFERENCES


APPENDIX A
(Altera Design Software 8.1 Installation Manual for the Ubuntu Linux Environment)
veCAD Technical Report

Altera Design Software 8.1 Installation Manual for the Ubuntu Linux Environment

Vishnu Paramasivam
Mohamed Khalil-Hani, PhD

Version Date: 24 Nov 2008 (Issue 1.4)

Abstract/Summary:
This tutorial shows a step-by-step guide on how to install the Altera Design Software 8.1 in Ubuntu Linux, including all the necessary hacks and modifications of the install scripts.

Contents:
1. Foreword
2. The Installation Process
3. Setting up the Paths and Licensing
4. USB-Blaster Setup
5. Miscellaneous Fixes

Revision history:

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Author</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issue 1.1</td>
<td>2007-10-28</td>
<td>Vishnu</td>
<td>Updated for v7.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Added ModelSim Installation</td>
</tr>
<tr>
<td>Issue 1.2</td>
<td>2008-1-31</td>
<td>Vishnu</td>
<td>Updated extra fixes.</td>
</tr>
<tr>
<td>Issue 1.3</td>
<td>2008-6-20</td>
<td>Vishnu</td>
<td>Updated for Quartus 8.0</td>
</tr>
<tr>
<td>Issue 1.4</td>
<td>2008-11-24</td>
<td>Pei chee</td>
<td>Updated for Quartus 8.1</td>
</tr>
</tbody>
</table>

Links:
http://nioswiki.jot.com/WikiHome/OperatingSystems/QuartusforLinux

Please send comments and inquiries to:
VLSI-ECAD Research Laboratory (veCAD)
Faculty of Electrical Engineering
University Teknologi Malaysia (UTM)
81310 Skudai, Johor.
vecad@fke.utm.my
www.fke.utm.my/vecad

UTM
Foreword

- Altera Design Software for Linux was designed to work on a very old version of Red Hat Linux. So if you still haven't chosen a Linux to work with, CentOS is a good Linux distribution which is based on Red Hat Linux but modern and compatible.

- Some changes are required to be done before it will work on a more modern distribution of Linux, like Ubuntu.

- Altera Quartus II for Linux currently only accepts a floating license. There is no web-edition version, nor does it support a T-guard license or any kind of hardware based fixed PC license.

- It is recommended to use Quartus tools in Linux for developing systems that use uClinux. Also, the Altera design tools run faster in Linux.
The Installation Process

1. Download the installation files (for Linux, not UNIX) from the Altera website:
   - Altera Installer (81_altera_installer.tar)
   - Quartus II Subscription Edition Software (81_quartus_linux.tar)
   - Nios II Embedded Design Suite (81_nios2eds_linux.tar) (if required)
   - ModelSim-Altera (81_modelsim_ae_linux.tar) (if required)

2. Extract the 81_altera_installer.tar into a folder. Please make sure the whole path for this folder does not contain any spaces.

3. Next, put all the other three installation files into the Altera Installer folder. See the picture below for clarification.

   Note: The installation of ModelSim-Altera will fail, because it is essentially the same old version from before. If you need to use Modelsim, please replace the file 'install_download' file with the patched one provided in the nioswiki.
Open up a terminal. Type:

```bash
sudo apt-get remove csh
dsudo apt-get install tcsh
```

To ensure everything works, we need to change the default shell from dash to bash. This will also improve compatibility of other scripts, such as compiling uCllinux and buildroot. Type:

```bash
sudo rm /bin/sh
dsudo ln -s /bin/bash /bin/sh
```

Next, cd to the 81_altera_installer folder. Then type:

```bash
sudo ./install_download
```

Follow the on-screen instructions. Make sure:

- The Altera install folder is `/opt/altera8.1`

  *If you want to use a different folder, please make the appropriate adjustments.*

- All three tar files (81_quartus_linux.tar, 81_nios2eds_linux.tar, 81_modelsim_aci_linux.tar) is inside the Altera installer folder

- Platform selection is 'linux'

Choose whichever software you want to install, and the installation will proceed. This will take a few minutes.

Once the installation is complete, set up the environment to provide commonly used command line tools available. Type this:

```
# Nios II command line utilities
QUARTUS_ROOTDIR=/opt/altera8.1/quartus
PATH=/opt/altera8.1/nios2eds/bin:$QUARTUS_ROOTDIR/bin:/opt/altera8.1/nios2eds/bin/nios2-gnutools/H-i686-pc-linux-gnu/bin:$PATH
export QUARTUS_ROOTDIR
```

Next create a shortcut to the Nios II SDK shell:

```bash
sudo gedit /bin/n2sh
```
12. Gedit will open up. Put this inside the file:

```bash
#!/bin/sh

# Nios II SDK Shell

/opt/altera8.1/nios2eds/sdk_shell
```

13. Make sure the n2sh script is executable.

```
sudo chmod +x /bin/n2sh
```

14. Log off and log in back into your system for the environment settings to take effect. Now you will be able to use the Nios II SDK shell. This shell contains all the paths and environment variables required to use all the Altera tools.

15. Type 'n2sh' in any terminal to start a Nios II SDK shell. Now you can run any program you want from the terminal.
   - Quartus II: type `quartus`.
   - Nios II IDE: type `nios2-ide`.
   - All the command line tools: `nios2-download`, `nios2-terminal` and so on.
   - `nios2-configure-sof` can be used to download a .sof file into the board.
   - `lmgrd -c /home/myname/mylicensefile` can be used to set up your license.

16. You can still run several of the programs without using the Nios II SDK shell, such as `nios2-download`, and `nios2-configure-sof`. But programs such as Quartus II and Nios II IDE must be executed through the Nios II SDK shell.
Setting up the Licensing

- If you are setting up a floating license server in Linux and need to use the program `lmgrd` program, it needs to be run during startup. The path for this file needs to be in the global environment, which is in `/etc/environment`.

- Follow the Altera instructions in their website, and then insert the path `“:/opt/altera8.1/quartus/linux”` inside the PATH variable in `/etc/environment` file. The `lmgrd` command can be set up to run when the computer boots up. This is done by adding the command in `System > Preferences > Sessions`.

- Optionally, it is better to put the following four files into a separate folder `/opt/flexlm`. All the following files (excluding the license file) can be found in `/opt/altera8.1/quartus/linux`.
  - `lmgrd`
  - `alterad`
  - `lmutil`
  - `yourlicensefile.dat`

- Then, append “`:opt/flexlm” into the PATH variable in the `/etc/environment` file. The `lmgrd` command can be set up to run when the computer boots up. This is done by adding the whole command in `System > Preferences > Sessions`. 
USB-Blaster Setup

1. To use the USB-Blaster, we need to enable USB filesystem in Ubuntu.

2. Create an empty file in your home directory named “.jtag.conf”

3. Type this in a terminal:

   ```
   sudo gedit /etc/fstab
   ```

   at the end of the file:

   ```
   usbfs /proc/bus/usb usbfs devmode=0666 0 0
   ```

4. Put this line

5. This gives permission to anyone to use the USB file system. This is effective after rebooting Linux, or by remounting the filesystem:

   ```
   sudo mount -a
   ```

6. That's it. You should be able to use the USB-Blaster now.
Miscellaneous Fixes

1. Sometimes problems might occur. V8.1 seems to solve many of these problems, but these fixes are left here just in case.

2. Quartus stops responding, and hangs. To solve this, just type “**killall nautilus**” in a terminal.

3. To fix the language issue with Perl, type this in a terminal (case sensitive):
   ```
   cd /usr/lib/locale
   cp -r en_US.utf8 en_US
   ```

4. To fix the “Unknown Linux Processor” issue, copy the provided file ‘arch’ and put it in `/bin`. You will need administrator access to do this, so type “**sudo nautilus**” in a terminal.

5. The ‘arch’ file is a self written script that uses the 'uname' command instead of the obsolete 'arch' command. This because Quartus is written for Red Hat v9, a now ancient Linux distro.

6. The biggest problem in Quartus is the Licences menu will not open. This can be solved by copying the provided file 'libX11.so.6.2' into `/opt/alteraX.X/quartus/linux`. You will need administrator privileges.

7. Then type this in a terminal (case sensitive):
   ```
   cd /opt/alteraX.X/quartus/linux
   sudo ln -s libX11.so.6.2 libX11.so.6
   ```

8. Please note that X.X is the quartus version, change the commands above to suit the version used. Also note that most of the commands don't report anything if the command is typed correctly.
APPENDIX B

(Compiling and Using uClinux with the Altera Nios II)
Compiling and Using µClinux with the Altera Nios II

Vishnu Paramasivam
Mohamed Khalil-Hani, PhD

Version Date: 24 Nov 2008 (Issue 1.2)

Abstract/Summary:

This documentation is about compiling and using µClinux with the Altera Nios II system. The examples inside are targeted for the Altera DE2 board, but should work on other development boards as well.

Contents:

6. Introduction
7. µClinux Overview
8. Compiling the µClinux Kernel
9. Using µClinux

Revision history:

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Author</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issue 1.0</td>
<td>2007-10-10</td>
<td>Vishnu</td>
<td>First draft version</td>
</tr>
<tr>
<td>Issue 1.1</td>
<td>2008-5-7</td>
<td>Vishnu</td>
<td>Updates and added information</td>
</tr>
<tr>
<td>Issue 1.2</td>
<td>2008-11-24</td>
<td>Pei Chee</td>
<td>Updates and added information</td>
</tr>
</tbody>
</table>

Links: [http://nioswiki.jot.com/%C2%B5Clinux](http://nioswiki.jot.com/%C2%B5Clinux)

Please send comments and inquiries to:

VLSI-ECAD Research Laboratory (veCAD)
Faculty of Electrical Engineering
University Teknologi Malaysia (UTM)
81310 Skudai, Johor.
vecad@fke.utm.my
www.fke.utm.my/vecad
Introduction

μClinux stands for "MicroController Linux". It is a fork of the Linux kernel for microcontrollers without a memory management unit (MMU), such as the Nios II processor. The supported architectures include M68K/Coldfire, ARM, Xilinx MicroBlaze, Opencores OR32, and Altera Nios II. μClinux must be patched first to work properly on an Altera Nios II system.

While Xilinx actively supports μClinux to include Microblaze in the mainstream μClinux kernel, Altera Nios II support is provided by the community. Altera actively supports MicroC/OS II RTOS because it is more profitable to them. μClinux on the other hand, is free. Initially, Microtronix ported μClinux for the Altera Nios II system, but the project was never updated and is outdated. In addition to that, it was not free.

The latest μClinux distribution is patched at various files for it to work on the Altera Nios II system. At the time of writing, the latest version is μClinux-dist-20080131. The server for the Nios II branch for μClinux is maintained by the National Taiwan University of Science and Technology. The server uses Git, a popular source code control system used by kernel developers.

This documentation describes how to patch, compile, and configure the μClinux source code so that it works on the Nios II platform. It also describes how to compile and add custom programs by modifying the filesystem. Additionally, it also describes how add USB and networking functionality to μClinux.

This documentation does not describe how to use Linux. Using Linux is outside the scope of this document. Learning how to work and use Linux is essential, because compiling μClinux is done on a host pc running Linux, such as Ubuntu or a similar modern Linux distribution. Since μClinux is a Linux distribution, knowledge in using Linux will help in the long run. Take a short course or learn through online tutorials on Linux basics and terminal usage before attempting to use μClinux.
μClinux Requirements

This documentation focuses the use of μClinux on the Altera DE2 board from Terasic, but it also works on other boards. The following is a summary of requirements for running μClinux on hardware:

- Two files are required to be generated using Quartus II: the hardware .sof file and the SOPC builder .ptf file.
- The hardware must be generated in SOPC builder using a Nios II Full Featured CPU.
- A minimum of **8 MB SDRAM/DDR SDRAM**. Flash memory must be present in the SOPC Builder .ptf file, but is not required.
- The CPU interrupt vector must be set to the SDRAM/DDR SDRAM.
- The CPU reset vector must be set to the Flash memory.
- Do not assign IRQ 0 to any hardware. Use other interrupt values.
- Include at least one full-featured timer. Name it “**timer_0**”
- For Ethernet functionality, use the DM9000A chip on the DE2. Refer to the “Designing the Davicom DM9000A Ethernet Chip SOPC Builder component” technical report. It must be named “**DM9000A**”.
- For USB functionality, use the ISP1362 chip on the DE2. Refer to the “Designing the Philips ISP1362 USB Chip SOPC Builder component” technical report. It must be named “**ISP1362**”.

The following is a summary of software required to compile and use μClinux:

17. To compile μClinux, it must be done on a Linux PC, preferably Ubuntu. A virtual machine may be used, such as VMWare, but is not recommended.
18. The PC must have a working Quartus and Nios II EDS installation in Linux. Refer to the “Altera Design Software 8.1 Installation Manual for the Ubuntu Linux Environment” technical report.
19. Get these files from the lab (latest at time of writing):
   - **nios2-linux-updated.tar.gz**: The latest Nios2-Linux installation package
   - **nios2gcc-20080203.tar.bz2**: The latest pre-build binary toolchain for cross compilation.
- **system.ptf**, **system.sof**, and **zImage**: These are pre-compiled binaries. Use these to get a feel of µLinux on hardware before actually doing any compilation.
µClinux Overview

µClinux is a Linux based RTOS. This means that it can run and compile Linux based programs and libraries. It also is a terminal based OS. Once µClinux is downloaded into the hardware, it will transmit and receive ASCII characters through the JTAG cable. These ASCII characters actually make up the µClinux terminal session. On the PC side, the nios2-terminal will be able to receive and transmit these ASCII characters as well. To a casual observer, this seems as if the PC is the one running a typical Linux terminal session, when it is actually the terminal session for µClinux which is running on the board.

After compilation, µClinux is downloaded into the SDRAM, or whatever suitable type of memory. The compilation process packs all the information into one file which is called the Linux kernel image, usually named zImage. The FPGA on the other hand, is programmed with an Altera Nios II system, which must conform to the requirements stated in the previous chapter. Any C/C++ program that is written for µClinux must be included inside the µClinux image. This is explained in later chapters.

µClinux does not lock hardware access like a real OS. Therefore by using volatile variables pointing to an address we can directly access the hardware or coprocessors in our programs. Since µClinux is an embedded OS with the target processor set to Altera Nios II, we must cross-compile the programs that we want to run on it. This is done by using the binary toolchain and libraries provided (refer to the next chapter).

Multitasking is also possible. By running an application with the ' & ' at the end of the command, the application will run in the background. The “ps” command (process show) will display a list of all processes running, like any typical Linux distribution. The “kill n” command can be used to destroy a process numbered 'n'.

The picture in the next page explains the connection between µClinux, the FPGA, the Nios II CPU, and the external hardware such as USB and Ethernet.
Abstract diagram showing the connection between μClinux, the FPGA, the Nios II CPU, and the external hardware such as USB and Ethernet.
Compiling the µClinux Kernel

**Before beginning**

- In Ubuntu, type this in a terminal to install dependencies for µClinux:

  ```
  sudo apt-get install ncurses-dev bison flex gawk gettext libncurses-dev curl git-core build-essential
  make gcc ccache zlib1g-dev libx11-dev texinfo liblzo-dev
  ```

- Set your terminal so that the backspace key generates “Control-H”. This is to make sure a nios2-terminal session works properly.

  ```
  Edit > Profile Preferences > Compatibility (in terminal menu)
  ```

- Check if the default shell is bash:

  ```
  ls -l /bin/sh
  ```
  This should give: `/bin/sh` -> `/bin/bash`

  Otherwise, link sh (gnome-terminal) to run bash instead of dash:

  ```
  sudo rm /bin/sh
  sudo ln -s /bin/bash /bin/sh
  ```

  Dash is default terminal for Ubuntu. Bash is slower than dash, but it is more compatible with scripting and programming makefiles.

- Extract the nios2-linux-updated.tar.gz into your working dir (eg. ~/Program).

  ```
  Please make necessary adjustment if you use different folder.
  ```

  ```
  tar xfvz nios2-linux-updated.tar.gz
  ```

- Checkout the source, need to do once only. This might take several minutes. Nios2 uClinux and GNU tools are now ready.

  ```
  ./checkout
  ```
Installing the Binary Toolchain

1. Extract the cross gcc compiler `nios2gcc-20080203.tar.bz2` into dir `opt/nios2`.
   
   Please make necessary adjustment if you use different folder.
   
   ```bash
   tar xfvj nios2gcc-20080203.tar.bz2
   ```

   *need administrative privilege to extract at this location*

   *just simply execute the command*

2. Now setup the PATH for cross gcc compiler.
   
   i. Use gedit to open `~/.profile`
   
   ```bash
   gedit ~/.profile
   ```

   ii. Add this at the end of the file
   
   ```bash
   PATH=$PATH:/opt/nios2/bin
   ```

   iii. Log out and log out to your system for the settings to take effect.

3. Now the gcc cross compiler for Nios II can be invoked by typing `nios2-linux-uclibc-gcc`.

Testing a pre-compiled µClinux image

(To try using µClinux before going in too deep)

- Cd to the directory containing `system.ptf`, `system.sof` and `zImage` file. Download the Nios II CPU (`system.sof`) into the FPGA

  ```bash
  nios2-configure-sof system.sof
  ```

  This `.sof` file is meant for an Altera DE2 Educational Board. Once the `.sof` file is downloaded, leave it running and open another terminal.

- Download the µClinux image (`zImage`) into the SDRAM

  ```bash
  nios2-download -g zImage
  ```

- There is a `system.ptf` file also included. This `.ptf` file is the file used to generate the Nios II CPU system through SOPC builder in Quartus II. It can be used as a reference. The `.ptf` and `.sof` file must be compatible.

- Once the image has been downloading onto the board, type this to begin µClinux:

  ```bash
  nios2-terminal
  ```
The µClinux inside the FPGA will boot up and send ASCII messages to the PC through the JTAG cable. This gives the illusion that the PC is running µClinux, which is a misconception. It is actually a terminal session running on the board itself, which is typical in Linux.

**Compiling the kernel**

- Cd to dir uClinux-dist
  ```
  cd ~/Program/nios2-linux/uClinux-dist
  ```

- Perform the `menuconfig` to configure the Linux kernel. This menu is to set the library and target processor.
  ```
  make menuconfig
  ```

- In the menu, select as follow:
  ```
  Vendor/Product Selection --> #<select>
  -- Select the Vendor you wish to target
  Vendor (Altera) --> #should have default to Altera
  -- Select the Product you wish to target
  Altera Products (nios2) --> #should have default to nios2
  #<exit>
  Kernel/Library/Default Selection --> #<select>
  ---Kernel is linux-2.6.x
  Libe Version (None) --> #should have default to None – very important
  [*] Default all settings (lose changes) (NEW) #<select>
  [ ] Customize Kernel Settings (NEW)
  [ ] Customize Application/Library Settings (NEW)
  [ ] Update Default Vendor Settings (NEW)
  - Exit and save changes. <exit> <exit> <yes>
```

- Next, setup the memory and IO port address map of Nios II board. Select nios2 cpu and sdram
  ```
  make vendor_hwselect SYSPTF=~/Program/system.ptf
  ```
  If you have your own .ptf and .sof files that complies with the stated requirements (see previous chapter) you can use the .ptf file here instead.
  ```
  make vendor_hwselect SYSPTF=/pathToYourHardwareProject/yourSystem.ptf
  ```

  Select nios2 cpu and SDRAM:
--- Please select which CPU you wish to build the kernel against:
(1) cpu_0 - Class: altera_nios2 Type: f Version: 7.07
Selection: 1
--- Please select a device to execute kernel from:
(3) sdram_0
Selection: 3
*(optional) --- Device to upload the kernel to : Flash Memory

- Compile the kernel source code: (this might take several minutes)
  
  `make`

- Make the image file, `zImage`, where the compressed kernel is stored and is in ELF form.
  
  `make image`

  The new Linux image will be in
  
  `/~Program/nios2-linux/uClinux-dist/images`

- Download `.sof` into FPGA, the image file `zImage` into the SDRAM and start `nios2-terminal`
  :
  
  `nios2-configure-sof system.sof`
  `nios2-download -g zImage`
  `nios2-terminal`

  The steps involved is similar to the previous section “Testing a pre-compiled µClinux image”, except this time use the new `zImage` file. If you used your own .ptf file, then you must also download the corresponding .sof file onto the board.

- To summarize, these are the steps required to compile µClinux :
  i. `make menuconfig`
  ii. `make vendor_hwselect SYSPTF=/pathToYourHardwareProject/yourSystem.ptf`
  iii. `make`
  iv. `make image`
Using µClinux

µClinux Root Filesystem

After running the “make image” command, the whole root filesystem of µClinux will be in
~/.µClinux-dist/romfs. Here we will be able to see the home, mount, boot, usr, and bin folders, to
say the least. The filesystem used by is typical of all Linux systems, so even average Linux users will
feel at home. Modifying any file in the romfs folder and running the make “make image” command
again will include all the changes made into the Linux image, zImage. When loaded into the
hardware, the changes made can be seen. For example, a cross-compiled program should be put in
the ../romfs/bin folder. Then the “make image” command must then be executed again to produce a
new zImage file. When downloaded into the hardware, the program can be executed directly.

The whole romfs folder should preferably be less than 2MB in size, so care must be taken
when inserting custom programs and media. It is recommended to be less than 2MB because this is
the size of a typical flash ROM. The zImage file that will be put into the flash ROM (or SDRAM
when testing) is actually the romfs folder compressed and packed into a single file.

The “make clean” command will delete this folder entirely. The whole compiling process
will have to be repeated once the “make clean” command is used.

µClinux Boot Sequence

To edit the µClinux boot sequence, modify the ../romfs/etc/rc file. For example, to setup a
static ip address or to use DHCP automatically, the commands to do so can be put at the end of the
../romfs/etc/rc file. Any command that needs to be executed when µClinux boots up can be put in
here.

When µClinux boots up there will be a welcome message “Welcome to µClinux”. This is
actually the command “cat /etc/modt” in the last line of the ../romfs/etc/rc file. This means that if
the "./romfs/etc/modt" file is modified, we will be able to display a custom welcome message. This is just an example of modifying the boot sequence.

**Cross-Compiling Programs for µClinux**

To compile a simple program, just add `-elf2flt` to link flag. For example, a simple hello.c file:

```c
#include <stdio.h>
int main(void)
{
    printf("hello world\n");
}
```

To compile hello.c, just type: `nios2-linux-uclibc-gcc hello.c -o hello -elf2flt`

The compiled object format is FLAT. It can be checked with: `nios2-linux-uclibc-flthdr hello`

Then copy hello to the romfs's bin dir. Rebuild the kernel image for initramfs.:

```
cp hello ~/uClinux-dist/romfs/bin
cd ~/uClinux-dist
make image
```

Download the new `zImage` file into the hardware. The hello world program can now be executed directly in the console. Read the previous sections to understand the µClinux root filesystem more.

The default stack size of application is 4KB, you can change it with `-elf2flt="-s <new stack size>"` option. For example, to increase the hello world program stack size, type this:

```
nios2-linux-uclibc-gcc hello.c -o hello -elf2flt="-s 16000"
```

The hello program will now have stack size as 16KiB.

**Cross-Compiling Programs using Eclipse for µClinux**

To compile a program using Eclipse, the following settings need to be done.

7. Right click on your project, choose:

```
Properties > C/C++ Build > Settings
```

8. For all compiler, change:
9. Add directories for complier:

```
C++ : /opt/nios2/include
/opt/nios2/include/c++/3.4.6
C   : /opt/nios2/include
```

10. In all Miscellaneous, add the flag option:

```
-elf2flt="-s 32000"
```

(stack size 32KB, you may change to your own preferable stack size)

Build your project, copy the “flt” format file in your project workspace into romfs' bin folder and rebuild the kernel image. This steps is similar to the previous section “Cross-Compiling Programs for µClinux”.

```
cp project ~/uClinux-dist-test/romfs/bin
cd ~/uClinux-dist
make image
```

### IO programming in user space

Note, we are dealing with user space which is with the uclibc library that is included in /opt/nios2/include. It is not kernel space, therefore interrupts can not be used. Nios II HAL can not be used either, such as the IORD() and IOWR() functions. To use interrupts, a Linux driver must be written. These are known as modules. Modules can run in user space and are the best method to access a hardware or a coprocessor.

To access IO ports, we can define them as memory pointer access (uncached). µClinux does not lock access to the memory like normal Linux distros, so it is not a problem:

```
(*volatile unsigned *)(port)   //for read, replace port with the hardware address
(*volatile unsigned *)(port)=(d) //for write, replace port with the hardware address
```

To maintain the IORD and IOWR standard, which is used in Nios II systems, these macros can be defined at the beginning of the source code to ease portability:

```
#define IORD(address,offset) (*volatile unsigned *)((address)|0x80000000)+4*(offset))
#define IOWR(address,offset,value) (*volatile unsigned *)((address)|0x80000000)+4*(offset))=(value)
```
**Important** : The Nios II CPU uses cache. A cached IO access is different from a regular IO access. It most cases, it will give problems such as wrong or delayed reads/writes. This only applies for Nios II Full version with Data Cache. Nios II Standard version is uncached, so there should be no problems. You can define memory pointer access, and you can make it uncached by setting address bit 31.

*Example : LED at address 0x00004000 (cached) -> 0x80004000 (uncached)*

An IO read/write to address 0x00004000 will have problems, however an IO read/write to address 0x80004000 will have no problems. This is why the defined IORD/IOWR macro above sets the 31st bit high.
**Customizing the Kernel and Bundled Applications**

The “**make menuconfig**” command is actually used to configure the kernel (drivers, modules and hardware) and also to configure bundled software. To configure the kernel tick “**Customize Kernel Settings**” or to customize the available applications tick “**Customize Vendor/User Settings**”:

- Kernel/Library/Default Settings
- [*] Customize Kernel Settings
- [*] Customize Vendor/User Settings

The kernel configuration menu will appear. In this menu, you can enable any driver or module that is supported by µClinux. There is a wide range of drivers available here. After saving any changes made, a new menu will appear. This is the application menu. In this menu, we can include a wide variety of applications available in µClinux, from libraries to simple games.

However, including too many drivers or software in µClinux will cause it to become too big, and it might not fit into the SDRAM. So caution must be exercised.

**i) Enabling Networking Support**

In the kernel configuration menu, all the necessary drivers must be enabled. To activate networking support through the onboard Ethernet chip (DM9000A on the DE2 or SMC91C111 on Altera Nios II development boards) enable these options:

- Networking -->
  [*] Networking support
- Networking -->
  Networking options -->
  [*] Packet socket
  [*] Unix domain sockets
  [*] TCP/IP networking
- Device Drivers -->
  Network device support -->
  [*] Network device support
- Device Drivers -->
  Network device support -->
  Ethernet (10 or 100Mbit) -->
  [*] Ethernet (10 or 100Mbit)
  [*] SMC 91C9x/91C1xx support # only if you use SMC91C111 (Stratix I/II/III)
[*] DM9000

use

# only if you use the Davicom DM9000A. Do not
# the DM9000A with checksum offloading driver.
First, setup a custom MAC address. This step is not necessary.

```
ifconfig eth0 hw ether 00:07:ed:0a:03:29       # hardware MAC address
00:07:ed:0a:03:random
```

Then configure the IP address and router. If DHCP is available then skip this step.

```
ifconfig eth0 192.168.1.10                 # put in your static IP
route add default gw 192.168.1.254        # put in your gateway IP
```

Or, use a DHCP client if you have a DHCP server on the local network (it won't work if you don't have a dhcp server):

```
ifconfig eth0 up
dhcpcd &
```

To check the Ethernet connection, try "ping" the board to and from your PC or gateway. e.g.,

```
ping 192.168.1.254
```

**ii) Enabling USB and filesystem support**

To activate USB support in µCLinux, we need to specify the USB host controller that we use, and the hardware that we want to use with it. In this example, we want to use a USB drive. So we need filesystem support such as FAT32, EXT3, and NTFS. Note that using NTFS is not recommended.

File systems -->
[*] Ext3 journalling file system support   # Only if you want to use EXT3 (recommended)

File systems --> DOS/FAT/NT Filesystems -->
[*] MSDOS fs support
[*] VFAT (Windows-95) fs support
[*] NTFS file system support               # Only if you want to use NTFS (not recommended)

[*] NTFS write support                    # Only for write support (not recommended)
(437) Default codepage for FAT
(iso8859-1) Default iochard for FAT

File systems --> Native Language Support -->
(iso8859-1) Default NLS Option
[*] Codepage 437 (United States, Canada)

[*] NLS ISO 8859-1 (Latin 1; Western European Languages)
Next, enable the USB host controller and the hardware it works with. In this case we use the Philips ISP1362 USB chip which is available on the DE2.

Device Drivers --> SCSI device support -->
[*] SCSI device support
[*] SCSI disk support
USB support -->
[*] Support for Host-side USB
[*] ISP1362 HCD support
[*] USB Mass Storage support
[*] USB device filesystem

When a USB drive is inserted, we can determine which device it is by typing: `fdisk -l /dev/sda` in the terminal. This will list all USB mass storage devices. Typically, the device should be `/dev/sda1`. The following is an example output of the command:

```
Disk /dev/sda: 131 MB, 131072000 bytes
16 heads, 32 sectors/track, 500 cylinders
Units = cylinders of 512 * 512 = 262144 bytes

Device Boot Start End Blocks Id System
/dev/sda1 * 1 499 127728 6 FAT16
```

As can be seen in the last sentence above, the device is `/dev/sda1` with a FAT16 filesystem and can be mounted with a `mount` command: `mount /dev/sda1 /mnt -t vfat`

A simple `ls` command in the `/mnt` folder will display all the available files in the USB drive.
iii) Advanced : Using Buildroot to cross-compile libraries/applications

This is a topic that keeps changing over time. It is recommended to look at the nioswiki on the topic “buildroot” for the latest information. Buildroot is a set of Makefiles and patches that help to build the cross compiler, libraries, applications, Linux kernel and root filesystem. Buildroot was used to create the cross compiler and uClibc for Nios II.

To run buildroot, you must have a Linux desktop with software development packages. Login as root or use sudo to install these packages.

```
sudo apt-get install subversion make gcc ncurses-dev bison flex gawk
```

Next, grab a copy of buildroot from the lab server and unpack it into `~/buildroot`. The latest version can be downloaded from [http://buildroot.uclibc.org/downloads/snapshots/](http://buildroot.uclibc.org/downloads/snapshots/). Then configure for Nios II CPU.

```
cd ~/buildroot
make menuconfig
```

Please note, if this is your first try on buildroot, you should unselect everything in 'Toolchain' first, then only select the options you want. You should enable fewer options until you got first build successfully. Set the following:

Target Architecture (nios2) --->
Toolchain --->
Toolchain type (Buildroot toolchain) --->
--- Kernel Header Options
   Kernel Headers (Linux 2.6.23.x kernel headers) --->
--- uClibc Options
   uClibc C library Version (uClibc 0.9.29) --->
   (toolchain/uClibc/uClibc-0.9.29.config) uClibc configuration file
   [ ] Thread library debugging # optional, if you want to debug thread
   [ ] Build/install c++ compiler and libstdc++ # optional, if you want c++
   [ ] Build gdb for the Host # optional
      GDB debugger Version (gdb 6.6)
   [*] Enable elf2flt support
   [ ] Enable multilib support # optional, if you want various processor features
   [ ] Enable IPv6 # optional
   [*] Enable RPC
[*] Enable WCHAR support  # optional
Package Selection for the target  --->
# unselect everything
Target filesystem options  --->
# unselect everything
Kernel ---> Kernel type (none) --->
# none and unselect everything

Now build everything, it will automatically download all the sources and compile. This will take a very long time, as it will download the required source code for you.

```make
# If this process is taking too long, copy the “dl” folder
# in the µClinux/buildroot folder from the lab server, put it in
# the buildroot directory, and try again. This might not work
# if you use a later version of buildroot.
```

After building successfully, you need to update $PATH, remove old binary toolchain (/opt/nios2/bin) and add the new build. So that the cross compiler will be in the PATH when you login Linux. Edit /etc/profile and add the following (only do this if you want to use the toolchain you compiled yourself; if not, please ignore this step):

```
PATH="${PATH}":$HOME/buildroot/build_nios2/staging_dir/bin
```

Logout and login again. You can use the tools now. Run this to verify that you have it in your command search path,

```
nios2-linux-uclibc-gcc -v
```

Now you can compile numerous applications and libraries that are not included by default in µClinux. Just tick any application you want in “Package Selection” during “make configure”. Please note not everything will work magically, some applications and libraries need extra patching and tweaking before it will compile for the Nios II CPU. Such an example is the OpenSSL libraries.
Futher Reference

1. The best documentation is at the Nios community wiki:
   http://nioswiki.jot.com/WikiHome/OperatingSystems/%C2%B5Linux
2. Questions can be asked at the Nios community forum:
   http://www.niosforum.com
3. For further understanding on embedded Linux:
4. For a comprehensive and free book on writing drivers for Linux:
5. A recommended web based tutorial on writing Linux device drivers:
   http://www.freesoftwaremagazine.com/articles/drivers_linux
APPENDIX C

(Main Program Source Code for uClinux-Fingerprint.c)
int main()
{
    int answer;
    BYTE test[1];
    BYTE test1[1];
    BYTE **the_image;
    BYTE **image_data;
    long height, width;
    char c;
    // general variable for uart operation
    BYTE image[LEN];
    BYTE image2[LEN];
    // array to store characters received from uart
    char first = 0, first_bar = 0, second = 0, second_bar = 0,
    third = 0, third_bar = 0, fourth = 0;
    int data_len = 0;
    int image_len = 0;
    BYTE data[LEN] = {0};
    BYTE cryptogram[LEN] = {0};
    BYTE cryptogram1[LEN] = {0};
    BYTE image_decode[LEN] = {0};
    char operation = '0';
    int pad = 0;
    int k, j, p, q;
    int i = 0;  // general purpose variable for index looping
    int minutiae_count1 = 0;
    int minutiae_count2 = 0;
    int enroll_minutiae_count = 0;
    int part1a = 0;
    int match_count1 = 0;
    int verify_minutiae_count = 0;
    int part2a = 0;
    int match_count2 = 0;
    int n;
    double peratus = 0;
    FILE *fp;
    int pos;
    int end;
    int imagesize;
    struct record minutiae1[400];
    struct record valid_minutiae1[400];
    struct record minutiae2[400];
    struct record valid_minutiae2[400];
    struct record valid_minutiae[400];
    struct record minutiae_enroll[400];

    int upper = 0;
    int lower = 0;
    double divide = 0;
    double similarity_M = 0;
    double percentage = 0;

    char filename[100];
int ch;
char dat;

//np_pio *pio = na_button_pio;
//nr_pio_lcdinit(na_lcd_pio);

height = HGHT;
width = WTH;
image_len = LEN;

MAIN_MENU:

//test[0]=getchar();

printf("the process starts here \n");
//printf("please insert filename: ");
//gets(filename);
//printf("filenam selected is : %s \n",filename);
fp = fopen("./pic.pnm", "rb");

pos = ftell(fp);
fseek(fp, 0, SEEK_END);
end = ftell(fp);
fseek(fp, pos, SEEK_SET);

image_len = end;
width = 192;
height = end/width;

printf("no of char is %d \n", end);
printf("no of image_len is %d \n", image_len);
printf("no of width is %ld \n", width);
printf("no of height is %ld \n", height);

i=0;
while(!feof(fp))
{
    ch = fgetc(fp);
    dat = ch;
    image2[i] = dat;
    i++;
}

i=100;
j=0;
for (i=100; i<image_len; i++)
{
    image[j]=image2[i];
j++;
}
image_len=j;

fclose(fp);
printf ("closing file. \n");

for (i=0; i<=image_len; i++)
{
    image_decode[i]=image[i];
}

printf ("arrange 1-D image array into 2-D image array... \n");
// arrange 1-D image array into 2-D image array
// remove the padding
image_data = allocate_image_array(height,width);
imagesize = sizeof (image_decode);
printf ("size of image decode is %d \n", imagesize);

k = 0;
for (i = 0; i < height; i++)
{
    for (j = 0; j < width; j++)
    {
        image_data[i][j] = image_decode[k];
        k++;
    }
    pad = calculate_pad(width);
    if (pad != 0)
    {
        do
        {
            pad--;
            k++;
        }
        while (pad != 0);
    }
}

flip_image_array(image_data, height, width);

printf ("arrange 1-D image array into 2-D image array completed. \n");

printf ("separate the foreground region from the background...
\n");
//separate the foreground region from the background

the_image = allocate_image_array(height, width);
for (i = 0; i < height; i++)
{
    for (j = 0; j < width; j++)
    {
        the_image[i][j] = image_data[i][j];
    }
}
image_segmentation(the_image, height, width);
for (i = 0; i < height; i++)
{


for (j = 0; j < width; j++)
{
    image_data[i][j] = the_image[i][j];
}

//flip_image_array(the_image, height, width);
//free_image_array(the_image,width);
printf("separate the foreground region from the background completed. \n");
/*
//create an output file
// arrange 2-D image array into 1-D image array
k = 0;
for (i = 0; i < height; i++)
{
    for (j = 0; j < width; j++)
    {
        image_decode[k] = the_image[i][j];
        k++;
    }
}
//merge the image with its image header
i=0;
j=100;
for (i=0; i<=image_len; i++)
{
    image2[j]=image_decode[i];
    j++;
}
//calculate total image length
i=0;
i=image_len+100;
//write the image into a file
fp = fopen ("./image1-segmentation.pnm", "wb");
fwrite (image2, sizeof(image2[0]), i / sizeof(image2[0]), fp);
close(fp);
*/
printf("convert the image from 256 gray value to 2 gray value... \n");
//convert the image from 256 gray value to 2 gray value
the_image = allocate_image_array(height, width);
for (i = 0; i < height; i++)
{
    for (j = 0; j < width; j++)
    {
        the_image[i][j] = image_data[i][j];
    }
}
binarization(the_image, height, width);
for (i = 0; i < height; i++)
{
    for (j = 0; j < width; j++)
    {

image_data[i][j] = the_image[i][j];

convert(the_image, height, width);
//flip_image_array(the_image, height, width);
//free_image_array(the_image,width);
printf("convert the image from 256 gray value to 2 gray value completed. \n");
/*
//create an output file
// arrange 2-D image array into 1-D image array
k = 0;
for (i = 0; i < height; i++)
{
    for (j = 0; j < width; j++)
    {
        image_decode[k] = the_image[i][j];
        k++;
    }
}
//merge the image with its image header
i=0;
j=100;
for (i=0; i<=image_len; i++)
{
    image2[j]=image_decode[i];
    j++;
}
//calculate total image length
i=0;
i=image_len+100;
//write the image into a file
fp = fopen("./image1-binarization.pnm", "wb");
fwrite(image2, sizeof(image2[0]), i / sizeof(image2[0]), fp);
fclose(fp);
*/
printf("eliminate unwanted noise... \n");
//eliminate unwanted noise
the_image = allocate_image_array(height, width);
for (i = 0; i < height; i++)
{
    for (j = 0; j < width; j++)
    {
        the_image[i][j] = image_data[i][j];
    }
}
noise_elimination(the_image, height, width);
for (i = 0; i < height; i++)
{
    for (j = 0; j < width; j++)
    {
        image_data[i][j] = the_image[i][j];
    }
}
convert(the_image, height, width);
printf ("eliminate unwanted noise completed. \n");
//flip_image_array(the_image, height, width);
//free_image_array(the_image, width);

printf ("fill up the holes... \n");
//fill up the holes
the_image = allocate_image_array(height, width);
for (i = 0; i < height; i++)
{
    for (j = 0; j < width; j++)
    {
        the_image[i][j] = image_data[i][j];
    }
}
smoothing(the_image, height, width);
for (i = 0; i < height; i++)
{
    for (j = 0; j < width; j++)
    {
        image_data[i][j] = the_image[i][j];
    }
}
convert(the_image, height, width);
printf ("fill up the holes completed. \n");
//flip_image_array(the_image, height, width);
//free_image_array(the_image, width);

/*
// create an output file
// arrange 2-D image array into 1-D image array
k = 0;
for (i = 0; i < height; i++)
{
    for (j = 0; j < width; j++)
    {
        image_decode[k] = the_image[i][j];
        k++;
    }
}
// merge the image with its image header
i=0;
j=100;
for (i=0; i<=image_len; i++)
{
    image2[j]=image_decode[i];
j++;
}
// calculate total image length
i=0;
i=image_len+100;
// write the image into a file
*/
fp = fopen("./image1-smoothing.pnm", "wb");
fwrite(image2, sizeof(image2[0]), i / sizeof(image2[0]), fp);
fclose(fp);

printf("skeletonized the smoothed image to 1-pixel thick...

\n");

//skeletonized the smoothed image to 1-pixel thick
//nr_pio_lcdwritescreen("IMAGE PROCESSING => THINNING");

the_image = allocate_image_array(height, width);
for (i = 0; i < height; i++)
{
    for (j = 0; j < width; j++)
    {
        the_image[i][j] = image_data[i][j];
    }
}
thinning(the_image, height, width);
for (i = 0; i < height; i++)
{
    for (j = 0; j < width; j++)
    {
        image_data[i][j] = the_image[i][j];
    }
}

convert(the_image, height, width);
printf("skeletonized the smoothed image to 1-pixel thick completed. \n");

//create an output file
// arrange 2-D image array into 1-D image array
k = 0;
for (i = 0; i < height; i++)
{
    for (j = 0; j < width; j++)
    {
        image_decode[k] = the_image[i][j];
        k++;
    }
}
//merge the image with its image header
i=0;
j=100;
for (i=0; i<=image_len; i++)
{
    image2[j]=image_decode[i];
j++;
}
//calculate total image length
i=0;
i=image_len+100;
//write the image into a file
fp = fopen("./image1-thinning.pnm", "wb");
fwrite (image2, sizeof(image2[0]), i / sizeof(image2[0]), fp);
fclose(fp);

* /
printf ("extract ridge ending and ridge bifurcation...
\n") ;
//extract ridge ending and ridge bifurcation
the_image = allocate_image_array(height, width);
for (i = 0; i < height; i++)
{
    for (j = 0; j < width; j++)
    {
        the_image[i][j] = image_data[i][j];
    }
}
minutiae_count1 = minutiae_extraction1(the_image,
hight, width, minutiae1);
printf ("minutiae count1 completed. \n");
enroll_minutiae_count = minutiae_validation1(the_image,
hight, width, minutiae_count1, minutiae1, valid_minutiae1);
printf ("enroll minutiae count completed. \n");
part1a = enroll_minutiae_count;
printf ("part1a calculation completed. \n");
macth_count1 = enroll_minutiae_count;
printf ("match count calculation completed. \n");
for (p = part1a; p < 400; p++)
{
    valid_minutiae1[p].type = 0;
    valid_minutiae1[p].direction = 0;
    valid_minutiae1[p].x = 0;
    valid_minutiae1[p].y = 0;
}
convert(the_image, height, width);
flip_image_array(the_image, height, width);
free_image_array(the_image,width);
printf ("extract valid minutiae completed. \n");
printf ("extract ridge ending and ridge bifurcation completed. \n");

PERCENT: 

// received data from uart (2nd image) 
// store in array

//printf ("please insert filename2: ");
//gets (filename);
//printf ("filename2 selected is : %s \n",filename);
fp = fopen ("./pic2.pnm", "rb");

pos = ftell (fp);
seek (fp, 0, SEEK_END);
end = ftell (fp);
seek (fp, pos, SEEK_SET);

image_len = end;
width = 192;
height = end/width;

printf ("no of char is %d \n", end);
printf ("no of image_len is %d \n", image_len);
printf ("no of width is %ld \n", width);
printf ("no of height is %ld \n", height);

i=0;
while(!feof(fp))
{
    ch = fgetc(fp);
    dat = ch;
    image2[i] = dat;
    i++;
}

i=100;
j=0;
for (i=100; i<image_len; i++)
{
    image[j]=image2[i];
    j++;
}
image_len=j;
fclose(fp);

for (i=0; i<=image_len; i++)
{
    image_decode[i]=image[i];
}

printf ("arrange 1-D image array into 2-D image array... \n");
// arrange 1-D image array into 2-D image array (2nd image)
// remove the padding
image_data = allocate_image_array(height,width);
j=0;
k=0;
for (i = 0; i < height; i++)
{
    for (j = 0; j < width; j++)
    {
        image_data[i][j] = image_decode[k];
        k++;
    }
    pad = calculate_pad(width);
    if (pad != 0)
    {
        do
        {
            pad--;
            k++;
        }
        while (pad != 0);
    }
flip_image_array(image_data, height, width);

printf ("arrange 1-D image array into 2-D image array completed. \n");

printf ("separate the foreground region from the background...
");

//image2: image_segmentation
the_image = allocate_image_array(height, width);
for (i = 0; i < height; i++)
{
    for (j = 0; j < width; j++)
    {
        the_image[i][j] = image_data[i][j];
    }
}
image_segmentation(the_image, height, width);
for (i = 0; i < height; i++)
{
    for (j = 0; j < width; j++)
    {
        image_data[i][j] = the_image[i][j];
    }
}
//flip_image_array(the_image, height, width);
//free_image_array(the_image, width);
printf ("separate the foreground region from the background completed. \n");

/*
//create an output file
// arrange 2-D image array into 1-D image array
k = 0;
for (i = 0; i < height; i++)
{
    for (j = 0; j < width; j++)
    {
        image_decode[k] = the_image[i][j];
        k++;
    }
}
//merge the image with its image header
i=0;
j=100;
for (i=0; i<=image_len; i++)
{
    image2[j]=image_decode[i];
    j++;
}
//calculate total image length
i=0;
i=image_len+100;
//write the image into a file
fp = fopen ("./image2-segmentation.pnm", "wb");
fwrite (image2, sizeof(image2[0]), i / sizeof(image2[0]), fp);
fclose(fp);
*/
printf ("convert the image from 256 gray value to 2 gray value...
");
//image2: binarization
the_image = allocate_image_array(height, width);
for (i = 0; i < height; i++)
{
    for (j = 0; j < width; j++)
    {
        the_image[i][j] = image_data[i][j];
    }
}
binarization(the_image, height, width);
for (i = 0; i < height; i++)
{
    for (j = 0; j < width; j++)
    {
        image_data[i][j] = the_image[i][j];
    }
}
convert(the_image, height, width);
//flip_image_array(the_image, height, width);
//free_image_array(the_image, width);
printf ("convert the image from 256 gray value to 2 gray value completed. \n");
*/
//create an output file
// arrange 2-D image array into 1-D image array
k = 0;
for (i = 0; i < height; i++)
{
    for (j = 0; j < width; j++)
    {
        image_decode[k] = the_image[i][j];
k++;
    }
}
//merge the image with its image header
i=0;
j=100;
for (i=0; i<=image_len; i++)
{
    image2[j]=image_decode[i];
j++;
}
//calculate total image length
i=0;
i=image_len+100;
//write the image into a file
fp = fopen("./image2-binarization.pnm", "wb");write(image2, sizeof(image2[0]), i / sizeof(image2[0]), fp);close(fp);

/*
printf("eliminate unwanted noise... \n");
//image2: noise_elimination
the_image = allocate_image_array(height, width);
   for (i = 0; i < height; i++)
      {for (j = 0; j < width; j++)
         {the_image[i][j] = image_data[i][j];
         }
      }noise_elimination(the_image, height, width);
      for (i = 0; i < height; i++)
         {for (j = 0; j < width; j++)
            {image_data[i][j] = the_image[i][j];
            }
         }
      }
convert(the_image, height, width);
printf("eliminate unwanted noise completed. \n");
//flip_image_array(the_image, height, width);
//free_image_array(the_image,width);

printf("fill up the holes... \n");
//image2: smoothing
the_image = allocate_image_array(height, width);
   for (i = 0; i < height; i++)
      {for (j = 0; j < width; j++)
         {the_image[i][j] = image_data[i][j];
         }
      }smoothing(the_image, height, width);
      for (i = 0; i < height; i++)
         {for (j = 0; j < width; j++)
            {image_data[i][j] = the_image[i][j];
            }
         }
      }
convert(the_image, height, width);
printf("fill up the holes completed. \n");
//flip_image_array(the_image, height, width);
//free_image_array(the_image,width);
/*
//create an output file
// arrange 2-D image array into 1-D image array
k = 0;
for (i = 0; i < height; i++)
{
    for (j = 0; j < width; j++)
    {
        image_decode[k] = the_image[i][j];
        k++;
    }
}

// merge the image with its image header
i=0;
j=100;
for (i=0; i<=image_len; i++)
{
    image2[j]=image_decode[i];
    j++;
}

// calculate total image length
i=0;
i=image_len+100;

// write the image into a file
fp = fopen ("./image2-smoothing.pnm", "wb");
fwrite (image2, sizeof(image2[0]), i / sizeof(image2[0]), fp);
close(fp);
*

printf ("skeletonized the smoothed image to 1-pixel thick...
\n");

// image2: thinning
the_image = allocate_image_array(height, width);
for (i = 0; i < height; i++)
{
    for (j = 0; j < width; j++)
    {
        the_image[i][j] = image_data[i][j];
    }
}
thinning(the_image, height, width);
for (i = 0; i < height; i++)
{
    for (j = 0; j < width; j++)
    {
        image_data[i][j] = the_image[i][j];
    }
}

convert(the_image, height, width);
printf ("skeletonized the smoothed image to 1-pixel thick completed. \n");
/*

// create an output file

// arrange 2-D image array into 1-D image array
k = 0;
for (i = 0; i < height; i++)
{
    for (j = 0; j < width; j++)
    {
        image_decode[k] = the_image[i][j];
        k++;
    }
}

//merge the image with its image header
i=0;
j=100;
for (i=0; i<=image_len; i++)
{
    image2[j]=image_decode[i];
    j++;
}

//calculate total image length
i=0;
i=image_len+100;

//write the image into a file
fp = fopen("./image2-thinning.pnm", "wb");
fwrite(image2, sizeof(image2[0]), i / sizeof(image2[0]), fp);
fclose(fp);

//image2: minutiae_extraction2 and minutiae_validation2
the_image = allocate_image_array(height, width);
for (i = 0; i < height; i++)
{
    for (j = 0; j < width; j++)
    {
        the_image[i][j] = image_data[i][j];
    }
}

minutiae_count2 = minutiae_extraction2(the_image, height, width, minutiae2);
verify_minutiae_count = minutiae_validation2(the_image, height, width, minutiae_count2, minutiae2, valid_minutiae2);
part2a = verify_minutiae_count;
match_count2 = verify_minutiae_count;

// delete excess structure
for (q = part2a; q < 400; q++)
{
    valid_minutiae2[q].type = 0;
    valid_minutiae2[q].direction = 0;
    valid_minutiae2[q].x = 0;
    valid_minutiae2[q].y = 0;
}

flip_image_array(the_image, height, width);
free_image_array(the_image, width);
peratus = minutiae_matching(valid_minutiae2, valid_minutiae1, enroll_minutiae_count, verify_minutiae_count);
printf("percentage matching is %lf \n", peratus);
```c
END:

    return 0;
}
```