ระบบรักษาความปลอดภัยโดยเครื่องตรวจสอบลายนิ้วมือ ผ่านเครือข่ายอีเทอร์เนต

SECURITY SYSTEM BY FINGERPRINT SCANNER VIA ETHERNET

โดย

นายวุฒิชัย อุทธา

นายสรชัย สมรรถานันท์

นายสิริพงศ์ สิทธิไชยนันท์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2548

ผ่านการตรวจชิ้นงานแล้ว

(ลงชื่อ).....ผู้ตรวจ

ผ่านการตรวจรูปเล่มแล้ว

(ลงชื่อ).....ผู้ตรวจ

ระบบรักษาความปลอดภัยโดยเครื่องตรวจสอบลายนิ้วมือ ผ่านเครือข่ายอีเทอร์เนต

SECURITY SYSTEM BY FINGERPRINT SCANNER VIA ETHERNET

โดย

นายวุฒิชัย อุทธา 45010734

นายสรชัย สมรรถานันท์ 45010802

นายสิริพงศ์ สิทธิไชยนันท์ 45010833

อาจารย์ที่ปรึกษา

รศ.ดร.ปราโมทย์ วาดเขียน

ดร.จีรสุดา โกษียาภรณ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2548

ปริญญานิพนธ์ปีการศึกษา 2548

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง **ระบบรักษาความปลอดภัยโดยเครื่องตรวจสอบลายนิ้วมือผ่านเครือข่ายอีเทอร์เน็ต**

SECURITY SYSTEM BY FINGERPRINT SCANNER VIA ETHERNET

ผู้จัดทำ

1. นายวุฒิชัย	**อุทธา**	**45010734**
2. นายสรชัย	**สมรรถานันท์**	**45010802**
3. นายสิริพงศ์	**สิทธิไชยนันท์**	**45010833**

ปราโมทย์ ..... อาจารย์ที่ปรึกษา

(รศ.ดร.ปราโมทย์ วาดเขียน)

Khongyapat ..... อาจารย์ที่ปรึกษา

(ดร.จีรสุดา โกษียาภรณ์)

ระบบรักษาความปลอดภัยโดยเครื่องตรวจสอบลายนิ้วมือผ่านเครือข่ายอีเทอร์เนต

SECURITY SYSTEM BY FINGERPRINT SCANNER VIA ETHERNET

โดย 1. นายวุฒิชัย อุทชา 45010734

2. นายสรชัย สมรรถานันท์ 45010802

3. นายสิริพงศ์ สิทธิไชยนันท์ 45010833

อาจารย์ที่ปรึกษา 1. รศ.ดร.ปราโมทย์ วาดเขียน

2. ดร.จีรสุดา โกษียาภรณ์

บทคัดย่อ

โครงงานนี้มีจุดประสงค์ คือ สร้างระบบรักษาความปลอดภัย โดยการแยกแยะลายนิ้วมือของแต่ละบุคคล เพื่อให้บุคคลที่ได้รับอนุญาตผ่านเข้ามาในระบบเท่านั้น โดยทำการสร้างฮาร์ดแวร์ที่รับข้อมูลลายนิ้วมือแล้วส่งผ่านเครือข่าย LAN ซึ่งถูกควบคุมด้วยไมโครคอนโทรลเลอร์ไปยังเครื่องคอมพิวเตอร์เซิร์ฟเวอร์แล้วตรวจสอบฐานข้อมูล โดยสามารถนำระบบนี้ไปติดตั้งที่ประตูเข้าออกสำนักงาน

Abstract

The objective of project is a construction security system by identifying personal fingerprint whom permit to access the security system. By building hardware to receive fingerprint's data and transmit through Local Area Network (LAN) , which controlled by Microcontroller. In order to compare fingerprint's data with database at computer server. We can apply this system for set up at office's entrance.

สารบัญ

สารบัญรูปภาพ

หน้า

บทที่ 2 ทฤษฎีหรือหลักการ

รูปที่

บทที่ 3 การคำนวณและการสร้างวงจร

รูปที่

บทที่ 4 การทดลองและผลการทดลอง

รูปที่

สารบัญตาราง

หน้า

บทที่ 1
บทนำ

ความเป็นมาและรายละเอียดต่างๆ ที่เกี่ยวกับโครงงานที่จะกล่าวถึงในบทนี้ เป็นข้อแนะนำประกอบในการอ่านรายงาน ทั้งนี้มุ่งหวังให้ผู้อ่านสามารถเข้าใจถึงส่วนประกอบและแนวทางในการดำเนินการต่อไป โดยจะกล่าวถึงความเป็นมาและข้อควรทราบต่างๆ ที่เกี่ยวข้องกับ ระบบรักษาความปลอดภัยโดยเครื่องตรวจสอบลายนิ้วมือผ่านเครือข่ายอีเทอร์เนต (SECURITY SYSTEM BY FINGERPRINT SCANNER VIA ETHERNET)

1.1 ความเป็นมา

1.1.1 ลายนิ้วมือ (Fingerprint)

ในปัจจุบันนี้ประชาการมีจำนวนมากขึ้น ต้องมีการติดต่อระหว่างบุคคลต่างๆ ถ้าเราสามารถมีระบบจำแนกลักษณะบุคคลต่างๆ ก็จะทำให้การติดต่อสื่อสารเป็นไปด้วยความสะดวก และปลอดภัยมากยิ่งขึ้น จึงมีการคิดค้นวิธีการต่างๆ เช่น รหัสผ่าน (Password) ลายนิ้วมือ เสนส์ตา เพื่อจำแนกบุคคล

การจำแนกลายนิ้วมือเป็นวิธีการหนึ่งที่ใช้จำแนกบุคคล ซึ่งนอกจากจะใช้ในระบบรักษาความปลอดภัยแล้ว ยังสามารถใช้ประโยชน์ในด้านอื่นๆ เช่น ระบบค้นหาประวัติของบุคคลหรืออาชญากร ระบบฝากถอนโดยใช้เครื่องอัตโนมัติ (ATM) ระบบทะเบียน เป็นต้น

1.1.2 อีเทอร์เนต (Ethernet)

อีเทอร์เนตนั้นเป็นมาตรฐานการส่งข้อมูลที่อนุญาตให้เครื่องคอมพิวเตอร์ใช้ช่องสัญญาณร่วมกันโดยผลัดกันใช้ อุปกรณ์ที่ใช้ในการส่งสัญญาณของอินเทอร์เนตนั้นก็คือ การ์ดแลน สายแลน และ อุปกรณ์รวมสัญญาณ ถ้าเป็นการเชื่อมต่อแบบบัสจะใช้สายสัญญาณกลางหรือแบ็คโบนเป็นตัวรวมสัญญาณ แต่ถ้าเป็นการเชื่อมต่อแบบสตาร์ (Star) จะใช้ฮับเป็นอุปกรณ์รวมสัญญาณ ในปัจจุบันนิยมใช้การเชื่อมต่ออีเทอร์เนตแบบสตาร์มากเนื่องจากความเร็วในการส่งข้อมูลและความสะดวกในการดูและรักษา

ระบบเครือข่ายอีเทอร์เนต หมายถึง มาตรฐานในการต่อเชื่อมคอมพิวเตอร์หลายเครื่อง (ตั้งแต่สองเครื่องขึ้นไป) เข้าด้วยกันเป็นระบบเครือข่าย สำหรับปฏิบัติการในแต่ละจุด (เช่นตามบ้าน หรือ สำนักงาน ต่าง ๆ) หรือ Local Area Network ซึ่งนิยมเรียกกันสั้น ๆ ว่า LAN รวมทั้งระบบการสื่อสารที่ช่วยให้คอมพิวเตอร์เหล่านั้น สามารถใช้ข้อมูลและโปรแกรมต่างๆ ร่วมกันได้ด้วย

ในกรณีที่ คอมพิวเตอร์หลายเครื่อง ในห้องเดียวกัน หรืออาคารเดียวกัน ระบบเครือข่ายอีเทอร์เนตจะสามารถช่วยเพิ่มประสิทธิภาพการทำงานของคอมพิวเตอร์เหล่านั้นได้ เพราะหลังจากการสร้างระบบเครือข่ายอีเทอร์เนตขึ้นมาแล้ว ข้อมูลจะสามารถถ่ายโอนระหว่าง PC และเครื่องเซิร์ฟเวอร์ได้รวดเร็วขึ้นมาก อีกทั้งยังสามารถสั่งพิมพ์งานผ่านพรินเตอร์ หรือใช้โปรแกรมต่าง ๆ รวมทั้งระบบต่อเชื่อมอินเตอร์เนตร่วมกันระหว่างคอมพิวเตอร์ทุกเครื่องในเครือข่ายนั้นด้วย

จนถึงบัดนี้ระบบเครือข่ายนี้ก็ยังนับเป็นระบบยอดนิยมสำหรับธุรกิจน้อยใหญ่ ทำให้เครือข่ายอีเทอร์เนต กลายเป็นระบบมาตรฐานอย่างหนึ่งในการสร้างระบบเครือข่ายคอมพิวเตอร์ในปัจจุบัน แต่ระบบเครือข่ายอีเทอร์เนตก็เป็นอะไรที่มากกว่าแค่อุปกรณ์ฮาร์ดแวร์ เพราะมันยังรวมถึงรูปแบบในการสื่อสาร และการถ่าย

ไอนข้อมูลต่าง ๆ ของคอมพิวเตอร์ที่เชื่อมโยงกันนั้นด้วย ทั้งนี้คอมพิวเตอร์ที่ต่อเชื่อมด้วยระบบอีเทอร์เนตนี้ จะส่งข้อมูลไปตามสาย ในรูปแบบของกลุ่มข้อมูลขนาดเล็กที่เรียกว่า แพ็กเก็ต (Packet) โดยในแพ็กเก็ตนั้นนอกจากมีข้อมูลต่างๆ แล้ว ยังมีข้อมูลเกี่ยวกับที่อยู่ของคอมพิวเตอร์ที่เกี่ยวข้องกับการรับ-ส่งข้อมูลต่างๆ ด้วย

โดยโครงงานนี้เป็นระบบตรวจสอบลายนิ้วมือโดยใช้สแกนเนอร์ส่งข้อมูลลายนิ้วมือของแต่ละบุคคลผ่านเครือข่ายอีเทอร์เนตแล้วนำภาพลายนิ้วมือของบุคคลนั้นมาประมวลผลแล้วเก็บลักษณะเด่นไว้ในฐานข้อมูล และสามารถตรวจเทียบลายนิ้วมือที่ต้องการจำแนกบุคคลกับฐานข้อมูล เพื่อระบุได้ว่าตรงกับลายนิ้วมือของผู้ใด หรือไม่ใช่บุคคลที่มีในฐานข้อมูล

1.2 วัตถุประสงค์ของโครงงาน

สร้างระบบรักษาความปลอดภัย โดยการแยกแยะลายนิ้วมือของแต่ละบุคคล เพื่อให้บุคคลที่ได้รับอนุญาตผ่านเข้ามาในระบบเท่านั้น โดยทำการสร้างฮาร์ดแวร์ที่รับข้อมูลลายนิ้วมือแล้วส่งผ่านเครือข่าย LAN ซึ่งถูกควบคุมด้วยไมโครคอนโทรลเลอร์ไปยังเครื่องคอมพิวเตอร์เซิร์ฟเวอร์แล้วตรวจสอบฐานข้อมูล โดยสามารถนำระบบนี้ไปติดตั้งที่ประตูเข้าออกสำนักงาน

1.3 วิธีการดำเนินงาน

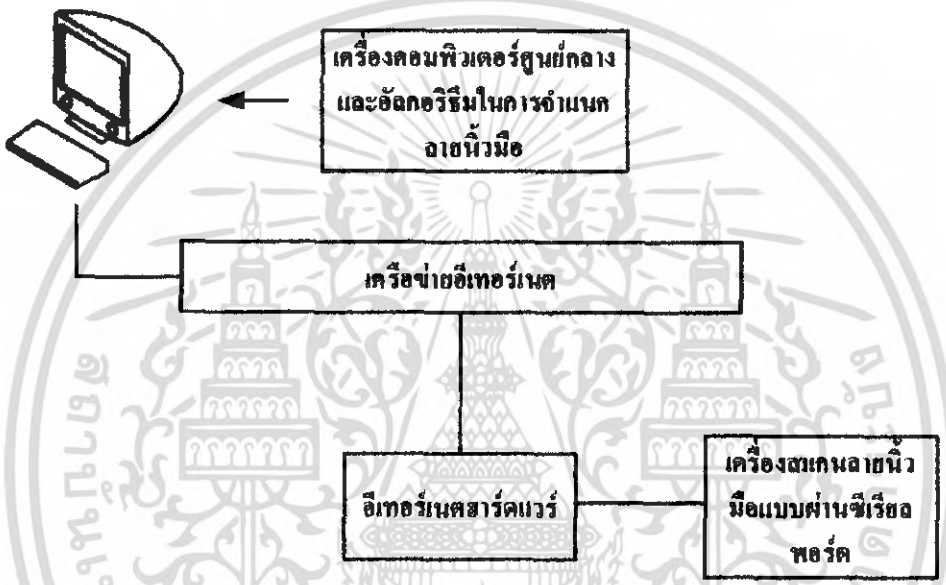
- ศึกษารวบรวมข้อมูลการทำงานของอีเทอร์เนต, อัลกอริธึมของการประมวลภาพ และวงจรต่างๆที่เกี่ยวข้อง
- เขียนโปรแกรมดึงลักษณะสำคัญของภาพลายนิ้วมือจากอัลกอริธึมที่ได้ศึกษา
- สร้างฮาร์ดแวร์ที่ใช้ในการรับและส่งข้อมูลผ่านวงแลน ซึ่งประกอบด้วย วงจรควบคุมการรับส่งข้อมูลผ่านอินเตอร์เนต และไมโครคอนโทรลเลอร์
- เขียนโปรแกรมทางด้านเครื่องคอมพิวเตอร์ศูนย์กลางเพื่อใช้ในการรับและส่งข้อมูล ระหว่างเครื่องคอมพิวเตอร์กับฮาร์ดแวร์ที่ได้กล่าวมาในข้อที่แล้ว
- สร้างฮาร์ดแวร์ที่ทำการรับภาพลายนิ้วมือของบุคคล แล้วทำการส่งข้อมูลภาพลายนิ้วมือให้กับฮาร์ดแวร์ที่เชื่อมต่อกับระบบแลน เพื่อจะนำภาพลายนิ้วมือไปประมวลผลที่เครื่องคอมพิวเตอร์ศูนย์กลาง
- ทำการเชื่อมต่อระบบทั้งหมดเข้าด้วยกัน
- เก็บผลการทดลอง
- วิเคราะห์และสรุปผล

บทที่ 2
ทฤษฎีหรือหลักการ

หลังจากที่ทราบถึงความเป็นมา, วัตถุประสงค์, และวิธีการดำเนินงานของบทที่ 1แล้ว ในบทที่ 2 นี้ จะเป็นทฤษฎีหรือหลักการที่เราได้ทำการศึกษาค้นคว้าเพื่อมาใช้ในการทำโครงงานนี้ โดยมีแบ่งออกเป็น 2 ส่วนใหญ่ดังนี้

1. การตรวจสอบลายนิ้วมือ
2. โพรโตคอล TCP/IP

โดยทฤษฎีทั้ง 3 ส่วน จะถูกนำมาใช้ในการอธิบายระบบโดยรวมของโครงงานนี้ ดังรูปที่ 2.1



รูปที่ 2.1 ส่วนประกอบโดยรวมของโครงงาน

เราจะทำการตรวจสอบลายนิ้วมือโดยการใช้เครื่องสแกนลายนิ้วมือแบบผ่านซีเรียลพอร์ตโดยจะทำการส่งข้อมูลลายนิ้วมือไปยังอีเทอร์เนตฮาร์ดแวร์ แล้วส่งผ่านเครือข่ายอีเทอร์เนตไปยังเครื่องคอมพิวเตอร์ศูนย์กลางเพื่อทำการจำแนกลายนิ้วมือ

2.1 การตรวจสอบลายนิ้วมือ

2.1.1 ไบโอเมตริกซ์ (Biometrics)

ไบโอเมตริกซ์ คือ การใช้ลักษณะทางกายภาพ หรือลักษณะทางพฤติกรรม ที่เป็นลักษณะเฉพาะของแต่ละคน ในการระบุตัวบุคคลโดยอัตโนมัติลักษณะทางกายภาพที่ใช้เป็นเกณฑ์ได้เช่น ลายนิ้วมือ, ใบหน้า, มือ, นิ้ว, หู, เรตินา และ Iris ภายในดวงตา เป็นต้น ส่วนลักษณะทางพฤติกรรมที่เป็นลักษณะเฉพาะของแต่ละบุคคล เช่น เสียงพูด, การพิมพ์, การเดิน เป็นต้น โดยสิ่งที่ใช้เป็นเกณฑ์ได้นั้นคือ จะต้องเป็นสิ่งที่มีการเปลี่ยนแปลงได้น้อยตามกาลเวลา และที่สำคัญจะต้องเป็นสิ่งที่เฉพาะไม่เหมือนกันในแต่ละบุคคล

คำว่า Biometricsนี้มาจากภาษากรีกสองคำคือ Bios (Life) ซึ่งหมายถึงชีวิตหรือความมีชีวิต กับคำว่า Metro (Measure) หมายถึงการวัดหรือมาตรวัด ซึ่งเมื่อรวมกันไบโอเมตริกซ์ จึงหมายถึงวิธีการหรือเทคนิคใน

การตรวจสอบแยกแยะสิ่งมีชีวิตโดยวัดจากคุณลักษณะของสิ่งมีชีวิตนั้นๆ คนเราสามารถที่จะแยกแยะเพื่อนออกจากคู่อริได้โดยการพิจารณาจากใบหน้า แต่ในกรณีของแฝดเหมือน (Identical Twins) สมองของคนเราที่ว่าแน่นักก็อาจจะต้องยอมแพ้เหมือนกัน คำจำกัดความที่ใช้กันโดยทั่วไปสำหรับไบโอเมตริกซ์ก็คือ เทคนิคอัตโนมัติต่างๆในการตรวจวัดคุณลักษณะทางกายภาพ (Physical Characteristics) พฤติกรรม (Behavior) ตลอดจนร่องรอยอื่นๆ ในชีวิตประจำวัน (Personal Traits) ของบุคคลที่มีชีวิต แล้วนำมาเปรียบเทียบกับคุณลักษณะนั้นๆ ที่ได้มีการบันทึกไว้ก่อนหน้านี้ในฐานข้อมูล เพื่อวัตถุประสงค์ในการแยกแยะ (Recognizing) บุคคลนั้นจากบุคคลอื่น สำหรับไบโอเมตริกซ์แล้วคุณลักษณะต่างๆ ไม่ว่าจะเป็นคุณลักษณะทางกายภาพ ทางพฤติกรรม หรือร่องรอย อื่นๆในชีวิตประจำวันของบุคคลจะต้องสามารถที่จะวัดในเชิงปริมาณได้ คุณลักษณะทางกายภาพนั้นส่วนใหญ่จะไม่แปรเปลี่ยนไปตามกาลเวลา ในขณะที่คุณลักษณะทางพฤติกรรมหรือร่องรอยในชีวิตประจำวันอาจมีการเปลี่ยนแปลงไปตามกาลเวลา ตามการเรียนรู้ของเจ้าของได้ ด้วยเหตุนี้ไบโอเมตริกซ์ที่ใช้คุณลักษณะทางกายภาพมาเป็นตัววัดจึงได้รับความเชื่อถือมากกว่า ตัวอย่างคุณลักษณะทางกายภาพที่ได้นำมาใช้เป็นไอดีไบโอเมตริกซ์อย่างแพร่หลาย ได้แก่ ลายนิ้วมือ ม่านตา เส้นเลือดที่ผนังลูกตาดำ รูปพรรณสัณฐานของมือ โครงสร้างรูปหน้า นอกจากนี้ยังมีคุณลักษณะทางกายภาพอื่นๆอีกที่ยังอยู่ระหว่างการศึกษาความเป็นไปได้ในการนำมาใช้เป็นไอดี และพัฒนาเป็นทางเลือกใหม่ให้กับเทคโนโลยีไบโอเมตริกซ์ เช่น ลักษณะรอยย่นของข้อนิ้ว (Knuckle Creases) คลื่นสมอง (Acoustic Head Resonance) หรือแม้แต่กลิ่นตัว (Body Odors) เป็นต้น

ข้อมูลไบโอเมตริกซ์จะต้องเก็บบันทึกจากบุคคลที่ยังมีชีวิตอยู่เท่านั้น โดยอุปกรณ์เก็บข้อมูลไบโอเมตริกซ์จะมีระบบตรวจเช็คความมีชีวิตของบุคคลด้วย ดังนั้นการใช้เทปบันทึกเสียงแทนการพูดด้วยเสียงจริงจะไม่สามารถหลอกลวงเครื่องแยกแยะเสียงพูดของคนตามหลักการไบโอเมตริกซ์ได้ หรือฉากหนึ่งในภาพยนตร์เรื่อง Demolition Man ที่มีการควักลูกตาผู้คุมนักโทษจิ้มติดปลายดินสอตบตาเครื่องอ่านเรตินา เพื่อหลบหนีออกจากเรือนจำนั้น ได้รับการยืนยันจากบริษัทผู้ผลิตอุปกรณ์นี้ (ซึ่งเป็นสปอนเซอร์ให้กับภาพยนตร์เรื่องดังกล่าวด้วย) ว่าจริงๆแล้วการกระทำเช่นนี้ไม่อาจจะหลอกลวงอุปกรณ์นี้ได้ เนื่องจากเครื่องอ่านเรตินามีคุณสมบัติในการตรวจเช็คความมีชีวิตของเส้นเลือดดำที่อยู่ในผนังด้านในของลูกตาได้ คุณสมบัติข้อนี้เองทำให้เทคโนโลยีไบโอเมตริกซ์ แตกต่างออกไปจากศาสตร์ทางด้านการชันสูตรศพ (Forensic Science) ไบโอเมตริกซ์ถูกนำมาใช้เพื่อวัตถุประสงค์ในการทำความรู้จัก หรือแยกแยะตัวบุคคล จากวัตถุประสงค์นี้เองสามารถแบ่งการใช้งานออกได้เป็นสองโอกาสคือ

2.1.1.1 เพื่อบ่งชี้ความเป็นตัวจริง (Verification) การใช้งานในกรณีนี้จะมีการเปรียบเทียบข้อมูลไบโอเมตริกซ์ที่เก็บได้ใหม่ (ณ จุดใช้งาน) กับข้อมูลของบุคคลนั้นที่ได้เคยลงทะเบียนไว้ เพื่อพิสูจน์ว่าบุคคลที่มากล่าวอ้างนี้เป็นตัวจริง ซึ่งถือว่าเป็นการเปรียบเทียบแบบ One-to-One

2.1.1.2 เพื่อระบุว่าบุคคลนั้นๆเป็นใคร (Identification) กรณีนี้จะต้องเทียบข้อมูลไบโอเมตริกซ์ที่เก็บใหม่ ณ จุดใช้งาน กับข้อมูลไบโอเมตริกซ์ทั้งหมดที่มีอยู่ในฐานข้อมูล เพื่อพิสูจน์ว่าบุคคลที่เป็นเจ้าของข้อมูลที่เก็บ ณ จุดใช้งานนั้น เป็นใครในฐานข้อมูล เป็นการเปรียบเทียบแบบ One-to-Many

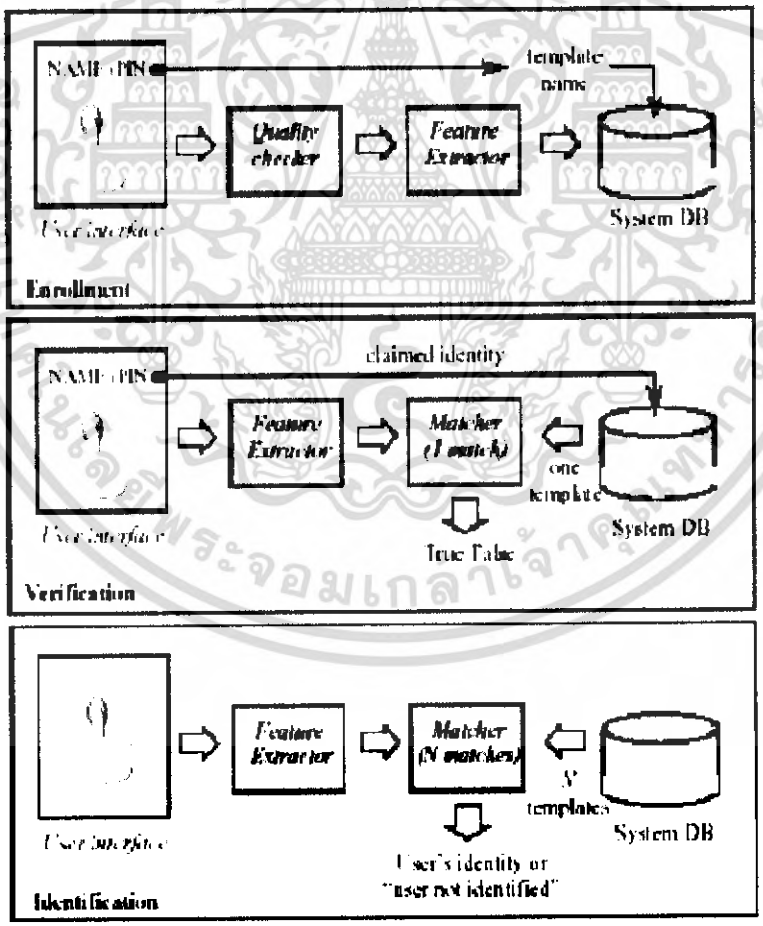
2.1.1.3 การลงทะเบียน (Enrollment) คือ ลักษณะการทำงานที่ทำการสแกนลายนิ้วมือ เพื่อบันทึกเป็นฐานข้อมูลลงไปในหน่วยความจำภายใน

เทคโนโลยีไบโอเมตริกซ์ จะใช้เทคนิคการทำงานแบบ อัตโนมัติ ซึ่งต้องมีขั้นตอนมาตรฐานดังนี้

1. เก็บตัวอย่างคุณลักษณะที่ต้องการวัด เช่น สแกนลายนิ้วมือออกมาเป็นภาพถ่ายลายนิ้วมือ
2. เก็บข้อมูลไบโอเมตริกซ์จากตัวอย่างที่สแกนได้ในข้อ 1 เช่น เก็บข้อมูลเชิงปริมาณจากภาพถ่ายลายนิ้วมือด้วยการคำนวณโดยใช้อัลกอริธึมเฉพาะ
3. เปรียบเทียบข้อมูลเชิงปริมาณที่วัดได้จากข้อ 2 กับข้อมูลที่ได้บันทึกเอาไว้ก่อนหน้านี้ ซึ่งอาจบันทึกไว้ในฐานข้อมูลกลาง หรือบันทึกไว้บนบัตรอัจฉริยะ
4. พิจารณาผลการเปรียบเทียบว่าถูกต้องตรงกันหรือไม่
5. ตัดสินว่าบุคคลนี้เป็นใคร (Identification) หรือเป็นตัวจริงตามที่มีการกล่าวอ้าง (Verification) หรือไม่

ก่อนที่ระบบจะสามารถระบุบุคคล และบ่งชี้ความเป็นตัวจริงของแต่ละบุคคลนั้น ระบบจะทำการร้องขอบางสิ่งบางอย่างเพื่อจะเทียบกับตัวมัน ดังนั้นประวัติและข้อมูลส่วนตัวทางด้านชีววิทยาจะถูกเก็บไว้ในระบบ

กระบวนการลงทะเบียน, การระบุบุคคล และการบ่งชี้ความเป็นตัวจริง แสดงดังรูปที่ 2.2



รูปที่ 2.2 แสดงกระบวนการลงทะเบียน, การระบุบุคคล และการบ่งชี้ความเป็นตัวจริง

2.1.2 หลักพื้นฐานในการพัฒนาเทคโนโลยีเพื่อระบุตัวบุคคล

พัฒนาการทางเทคโนโลยีในการระบุตัวบุคคลนั้นพัฒนามาจากหลักพื้นฐาน 3 หลัก ได้แก่

1) สิ่งที่รู้ (Something You Know) หมายถึง วิธีการยืนยันตัวบุคคลซึ่งเฉพาะบุคคลที่ตรวจสอบและบุคคลที่ถูกตรวจสอบเท่านั้นที่รู้เกี่ยวกับสิ่งที่ใช้ตรวจสอบ เช่น การใช้รหัสผ่าน (Password) รหัสประจำตัว (Personal Identification Number: PIN)

2) สิ่งที่มี (Something You Have) หมายถึง การที่บุคคลซึ่งถูกตรวจสอบมีสิ่งซึ่งใช้ยืนยันตัวบุคคล เช่น บัตรประจำตัวพนักงานในการบันทึกเวลาเข้า-ออกในการทำงานแต่ละวัน บัตรสมาร์ทการ์ด (Smart Card) หรือบัตรแถบแม่เหล็ก (Magnetic Card) เป็นต้น

3) สิ่งที่เป็น (Something You Are) หมายถึงการใช้ลักษณะเฉพาะของตัวบุคคลในการตรวจสอบและพิสูจน์ตัวบุคคลสิ่งที่ใช้ตรวจสอบมักเป็นเทคโนโลยีชีวภาพ เช่น ลายนิ้วมือ (Fingerprints) ม่านตา (Iris) เสียง (Voice Prints) หรือลักษณะของ DNA เป็นต้น

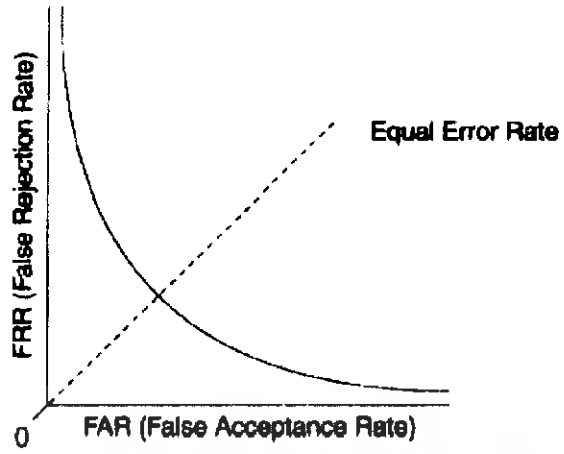
2.1.3 ความแม่นยำของไอดีไบโอเมตริกซ์

ความแม่นยำของไอดีไบโอเมตริกซ์ นอกจากจะพิจารณาจากประเภทของเทคโนโลยีที่ใช้แล้ว ยังพิจารณาได้จากตัววัดความแม่นยำทางเทคนิค ซึ่งผู้ผลิตอุปกรณ์จะต้องแจ้งให้ผู้ใช้ทราบขณะที่ทำการติดตั้งระบบ เพื่อการปรับแต่งตามสภาพการใช้งาน ประสิทธิภาพหรือความแม่นยำของอุปกรณ์ไบโอเมตริกซ์ จะวัดได้จากร้อยละของความผิดพลาดที่เกิดขึ้นขณะทำการตรวจสอบไอดี ดังต่อไปนี้

2.1.3.1 False Rejection Rate (FRR) หรืออัตราการปฏิเสธตัวจริง ค่า FRR จะบอกถึงร้อยละของความผิดพลาดที่เกิดขึ้นเมื่อตัวจริงถูกปฏิเสธ หากค่า FRR ของระบบต่ำแสดงว่าอัตราการปฏิเสธต่ำ นั่นคือไม่ว่าตัวจริงหรือตัวปลอมก็มีโอกาสได้รับอนุญาตให้ผ่านเข้าสู่ระบบได้ง่าย ระบบจะหละหลวม ฉะนั้นหากระบบต้องการความปลอดภัยค่อนข้างสูงก็ควรจะปรับแต่งให้ FRR มีค่าสูงขึ้น อย่างไรก็ตาม ข้อดีของการปรับให้ FRR มีค่าต่ำก็คือ ตัวจริงจะไม่ต้องหงุดหงิดกับการถูกปฏิเสธซ้ำแล้วซ้ำเล่าโดยอุปกรณ์อัตโนมัติ

2.1.3.2 False Acceptance Rate (FAR) หรืออัตราการยอมรับตัวปลอม เป็นร้อยละของความผิดพลาดที่เกิดขึ้นจากการที่ตัวปลอมได้รับการยอมรับ ค่า FAR ที่ต่ำจะหมายถึงอัตราการยอมปล่อยให้ผ่านเข้าสู่ระบบต่ำ ไม่ว่าตัวจริงหรือตัวปลอมก็มีโอกาสถูกปฏิเสธสูง ระบบจะเข้มงวดมาก ฉะนั้นหากต้องการให้ระบบมีความเป็นมิตรกับผู้ใช้ (User Friendliness) ก็ควรจะปรับให้ FAR มีค่าสูง อย่างไรก็ตามการปรับแต่งให้ FAR มีค่าต่ำจะช่วยป้องกันความเสียหายอันเนื่องมาจากการยอมรับตัวปลอมให้ผ่านเข้าสู่ระบบได้โดยง่าย

2.1.3.3 Equal Error Rate (ERR) คืออัตราความผิดพลาดที่ได้รอมชอมด้วยการปรับแต่งให้ FRR มีค่าใกล้เคียงกับ FAR ระบบที่มีค่า ERR ต่ำจะเป็นที่ต้องการมากกว่าระบบที่มีค่า ERR สูงกว่า ในบางครั้งจะเรียก ERR ว่า Crossover Rate



รูปที่ 2.3 แสดงความสัมพันธ์ระหว่าง FRR, FAR และ ERR

จะเห็นได้ว่าในทางปฏิบัติ หากเราต้องการให้ตัวจริงถูกปฏิเสธน้อย คือให้ค่า FRR ต่ำ เราจะต้องปล่อยให้ตัวปลอมได้รับการยอมรับมากขึ้น คือค่า FAR สูงขึ้น และในทางกลับกัน หากเราต้องการให้ตัวปลอมหลุดเข้ามาในระบบได้ยาก หรือ FAR ต่ำ เราก็ต้องแลกด้วยการปล่อยให้ตัวจริงต้องหงุดหงิดกับการถูกปฏิเสธบ่อยครั้ง หรือคือให้ FRR สูง นั่นคือ ตัววัดทั้งสองนี้จะแปรผกผันซึ่งกันและกัน ฉะนั้นทางสายกลางที่ใช้ในการพิจารณาก็คือ ดูจากค่า ERR ซึ่งเป็นค่าร้อยละที่ได้จากการรอมชอมระหว่างตัววัดสองตัวนี้

2.1.4 ระบบเครื่องสแกนลายนิ้วมือ

เครื่องสแกนลายนิ้วมือมือ (Fingerprint Scan) เป็นระบบที่ทำงานด้วยหลักเทคโนโลยีชีวภาค ใช้ลักษณะเฉพาะตัวของคนเราในการพิสูจน์ตัวบุคคลว่าคนคนนั้นคือใคร มีการนำไปใช้ควบคู่กับเทคโนโลยีสมัยใหม่หลายอย่าง อาทิ ระบบลงเวลาทำงานด้วยการสแกนลายนิ้วมือ ระบบควบคุมการเข้า-ออกอาคาร ระบบเปิด-ปิดประตูด้วยการสแกนลายมือ เป็นต้น

การนำระบบสแกนลายมือไปใช้ที่เห็นได้ชัดในระบบบันทึกเวลา ซึ่งปัจจุบันวิธีบันทึกเวลา ได้แก่ ระบบเซ็นชื่อ นาฬิกาตอกบัตร เครื่องรูดบัตรแบบบาร์โค้ด แถบแม่เหล็กไร้สัมผัส และเครื่องสแกนลายนิ้วมือ

2.1.5 ประวัติของการทำระบบตรวจสอบลายนิ้วมือ

ลายนิ้วมือ คือ รูปลายนูนที่ปรากฏบนนิ้วมือของมนุษย์ ลักษณะดังกล่าวเกิดขึ้นตั้งแต่เกิด ซึ่งมนุษย์ได้ใช้ลักษณะดังกล่าวเป็นตัวระบุบุคคลเป็นเวลานานมาก แต่ที่ค้นพบชิ้นแรกเริ่มต้นในศตวรรษที่ 16 โดยถูกพบตามวัตถุโบราณที่มนุษย์สร้างขึ้น และตามหลักฐานทางประวัติศาสตร์ต่างๆ ดังแสดงในรูปที่ 2.4

Neolithic Carvings (Gavrinis Island) [103]

Standing Stone (Goat Island, 2,000 B.C.) [85]

A Chinese clay seal (300 B.C.) [85]

An impression on a Palestinian lamp (400 A.D.) [103]

รูปที่ 2.4 แสดงลายมือนิ้วมือที่ถูกพบตามวัตถุโบราณและตามหลักฐานทางประวัติศาสตร์

ในปี 1684 นักวิทยาศาสตร์ด้านโครงสร้างสัตว์ และพืช ชื่อ N. grew ได้ตีพิมพ์รายงานเกี่ยวกับลายเส้นนูน และลายเส้นร่องของลายนิ้วมือ ซึ่งรายงานฉบับนี้ได้ถือว่าเป็นรายงานฉบับแรกของโลกที่เกี่ยวข้องกับลายนิ้วมือ หลังจากนั้นนักวิจัยจึงได้มุ่งความสนใจไปที่การศึกษาลายนิ้วมือมากขึ้น

รูปที่ 2.5 แสดงลายเส้นนูน และลายเส้นร่องของลายนิ้วมือตามรายงานของ N. grew

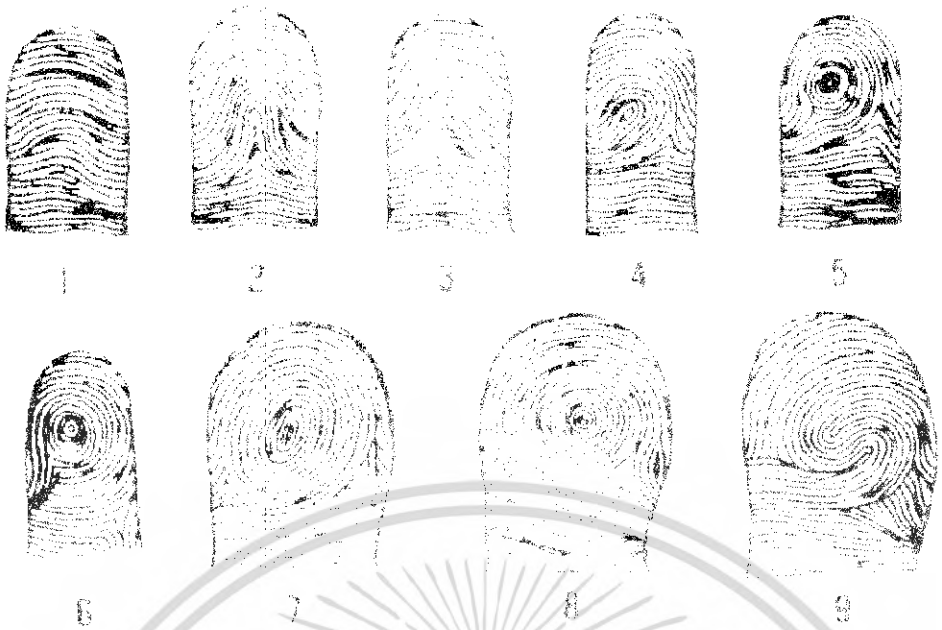
ในปี 1788 Mayor ได้ให้รายละเอียดด้านแบบจำลองกายวิภาคของโครงสร้างลายนิ้วมือ ดังแสดงในรูปที่ 2.6

รูปที่ 2.6 แสดงรายละเอียดด้านแบบจำลองกายวิภาคของโครงสร้างลายนิ้วมือ

ในปี 1809 T. Bewick เริ่มใช้ลายนิ้วมือของเขาแทนเครื่องหมายการค้า ดังแสดงในรูปที่ 2.7 และเชื่อกันว่า ผลในครั้งนั้นทำให้เกิดแรงกระตุ้นสำคัญในการใช้ลายนิ้วมือในการระบุบุคคล

รูปที่ 2.7 แสดงลายนิ้วมือของ T. Bewick ในการแทนเครื่องหมายการค้า

ในปี 1823 Purkinje เริ่มแบ่งกลุ่มลายนิ้วมือออกเป็น 9 ชนิด โดยแบ่งตามรูปร่างของรูปลายนูน ดังแสดงในรูปที่ 2.8



รูปที่ 2.8 แสดงการแบ่งกลุ่มลายนิ้วมือของ Purkinje

ในปี 1880 H. Fauld ได้เสนอความคิด ถึงความเป็นเอกลักษณ์ และโดดเด่นของลายนิ้วมือ ในเวลาเดียวกัน Herschel ได้เปิดเผยว่าเขาได้ทดลองการระบุตัวบุคคลด้วยลายนิ้วมือได้สำเร็จแล้ว โดยใช้เวลาทั้งสิ้น 20 ปี และการค้นพบของเขาก่อให้เกิดต้นแบบการระบุตัวบุคคลด้วยลายนิ้วมือในสมัยใหม่

ปลายศตวรรษที่ 18 Sir F. Galton ได้บรรยายผลการศึกษาของลายนิ้วมือ และเขาได้นำเสนอว่า จุดขาดคือ ลักษณะเด่นของลายนิ้วมือ ในปี 1888

ในปี 1889 E. Henry ได้พัฒนากระบวนการตรวจสอบลายนิ้วมือขึ้น เขามีชื่อเสียงอย่างมากขึ้นมาจากการสร้างระบบ Henry System ในการจำแนกกลุ่มของลายนิ้วมือ

ก่อนศตวรรษที่ 20 ได้มีความรู้ในลายนิ้วมือโดยสรุปได้ ดังนี้

- มีความเป็นเอกลักษณ์ในแต่ละคน และแต่ละนิ้ว
- ในความหลากหลายของลายนิ้วมือนี้เราสามารถแบ่งกลุ่มได้
- ลักษณะของจุดขาดของเส้นนูน และเส้นร่องจะคงที่

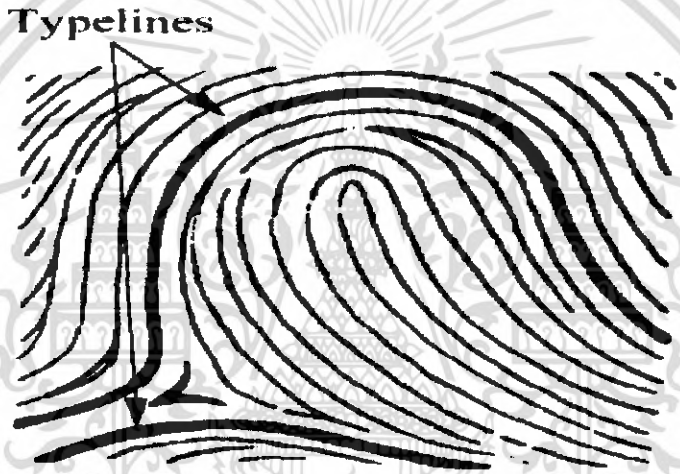
และนอกจากนี้ การใช้ลายนิ้วมือในการระบุตัวบุคคลยังได้รับการยอมรับอย่างเป็นทางการว่าให้ผลที่น่าเชื่อถืออีกด้วย ในเวลาต่อมาก็ได้พัฒนามาใช้ในทางกฎหมายอย่างเป็นทางการ และพัฒนาเพื่อการตรวจจับคนร้ายได้อีกด้วย

ก่อนปี 1960 Federal Bureau of Investigation (FBI) ได้ร่วมมือกับกรมตำรวจจากปารีส ในการพัฒนาระบบตรวจสอบลายนิ้วมืออัตโนมัติ (Automatic Fingerprint Identification System) ได้เป็นผลสำเร็จ และได้ทำมาใช้แทนที่ระบบเดิมที่ใช้ในด้านกฎหมาย และการตรวจจับคนร้าย ระบบดังกล่าวทำให้ได้ผลผลิตเพิ่มขึ้น และลดค่าจ้าง ค่าฝึกฝนพนักงานตรวจสอบเป็นอย่างมาก และต่อมาจึงเป็นที่แพร่หลายไปสู่ประชาชนทั่วไป ทำให้การตรวจสอบลายนิ้วมือนั้นได้แพร่หลายมากที่สุดในระบบตรวจสอบชีวภาพ

ถึงแม้ว่าระบบตรวจสอบลายนิ้วมืออัตโนมัติจะได้รับการพัฒนามาเป็นเวลา 30 ปีแล้ว แต่ยังไม่สามารถระบุกระบวนการที่ตายตัวได้ชัดเจน อันเนื่องมาจากปัจจัยด้านต่างๆ เช่น จำนวนรูปที่ตรวจสอบ คุณภาพรูปที่ตรวจสอบ ซึ่งก็ยังคงต้องพัฒนากันต่อไป

2.1.6 ลักษณะของลายนิ้วมือ

ผิวหนังบริเวณปลายนิ้วมือ ประกอบด้วยลายเส้นสองชนิด ชนิดหนึ่งเราเรียกว่าเส้นนูน (Ridge) อีกชนิดหนึ่งเราเรียกว่า รอบร่อง หรือเส้นร่อง (Furrow) จะอยู่สลับกันตลอดไป โดยถ้าเราใช้หมึกสีดำทาบนนิ้วมือและกดนิ้วมือลงบนกระดาษขาวจะได้ลายเส้นสีขาว และสีดำสลับกัน เราเรียกเส้นสีดำว่า เส้นนูน และเรียกเส้นสีขาวว่า เส้นร่อง ในการพิจารณาประเภทของลายนิ้วมือเราจะสนใจเพียงส่วนของลายนิ้วมือที่เรียกว่า บริเวณลายนิ้วมือที่อยู่ภายใน (Pattern Area) โดยบริเวณลายนิ้วมือที่อยู่ภายในของลายนิ้วมือประกอบไปด้วย เส้นขอบ (Type Line) ดังรูปที่ 2.9



รูปที่ 2.9 แสดงบริเวณลายนิ้วมือที่อยู่ภายใน (Pattern Area) และเส้นขอบ (Type Line)

บริเวณลายนิ้วมือที่อยู่ภายในของ Loop หรือ Whorl จะประกอบไปด้วยจุดสำคัญ 2 จุด คือ

2.1.6.1 จุดเดลต้า (Delta หรือ Ridge Dot) หมายถึง จุดบนลายนูนที่อยู่ตรงกลางที่มีลักษณะเป็นเดลต้า ซึ่งคือ บริเวณที่มีลักษณะของลายเหมือนรูปสามเหลี่ยม ดังรูปที่ 2.10 ซึ่งจะแสดงให้เห็นถึงบริเวณที่เป็นเดลต้า และจุดเดลต้าอยู่ตรงกลางของบริเวณที่มีลักษณะเป็น Delta

รูปที่ 2.10 แสดงตัวอย่างของรูปแบบของจุดเดลต้า

2.1.6.2 จุดคอร์ (Core) หมายถึง จุดบนเส้นโค้งของลายนูน โดยที่ลักษณะของเส้นโค้งนั้นต้องเป็นลักษณะที่โค้งขึ้น และเริ่มกลับตัวลง หรือเริ่มจะกลายเป็นโค้งลง แล้วจึงวิ่งสวนทางกันกับในตอนแรกก่อนที่จะ

โค้งลง เส้นโค้งลายนูนนี้จะต้องเป็นเส้นโค้งลายนูนที่อยู่ชั้นในสุดของบริเวณที่มีรูปแบบเป็นคอร์ ซึ่งเป็นลักษณะของบริเวณที่มีรูปแบบเป็นเส้นโค้งที่โค้งขึ้นมาซ้อนๆ กันหลายเส้น จากโค้งวงกลางลดลงมาเรื่อยๆ จนโค้งเล็กที่สุด (เส้นที่อยู่ในที่สุดอาจไม่เป็นเส้นโค้งก็ได้)

รูปที่ 2.11 แสดงตัวอย่างของรูปแบบของจุดคอร์

หลักการที่สำคัญในการแบ่งประเภทลายนิ้วมือ และการ Matching ลายนิ้วมือ คือ การนับเส้นนูน ซึ่งอาจกำหนดอย่างหยาบๆ ตามจำนวนของเส้นนูนที่สัมผัส หรือต่อผ่านลายเส้นระหว่างคอร์ และเดลต้าเนื่องจากรูปแบบของเส้นนูนมีความซับซ้อนสูง จึงทำให้การนับเส้นนูนทำได้ค่อนข้างยาก มี 3 ตัวอย่างในการนับเส้นนูน แสดงดังรูปที่ 2.12

รูปที่ 2.12 แสดงตัวอย่างในการนับเส้นนูน

2.1.7 ชนิดและรูปแบบของลายนิ้วมือ

รูปแบบของภาพลายนิ้วมือ สามารถแบ่งออกได้เป็น 3 กลุ่ม คือ

2.1.7.1 กลุ่มมัดหวาย (Loop)

มีลักษณะการวิ่งของลายนูนจากทางด้านใดด้านหนึ่งมาที่บริเวณกลางนิ้วมือ แล้ววิ่งโค้งขึ้นไปด้านบนแล้วจึงโค้งลง และวิ่งวนกลับไปตามทางเดิมที่ได้วิ่งมา กลุ่มมัดหวายนี้จะมี Delta อยู่หนึ่งจุด ประมาณ 60 – 65% ของมนุษย์จะมีลายนิ้วมือในกลุ่มนี้

รูปที่ 2.13 แสดงลายนิ้วมือของกลุ่มมัดหวาย (Loop)

กลุ่มมัดหวายที่พบในลายนิ้วมือทั่วไป สามารถแบ่งได้เป็น 2 ประเภท คือ

1. **กลุ่มมัดหวายเดี่ยว (Single Loop)** สามารถแบ่งได้เป็น 2 ชนิด คือ

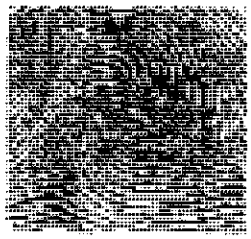
ก) มัดหวายซ้าย (Left Loop) มีลักษณะการวิ่งของลายนูนจากทางด้านซ้ายมาที่บริเวณกลางนิ้วมือ แล้ววิ่งโค้งขึ้นไปด้านบนแล้วจึงโค้งลง และวิ่งวนไปตามทางเดิมที่ได้วิ่งมา

รูปที่ 2.14 แสดงลักษณะของมัดหวายซ้าย

ข) มัดหวายขวา (Right Loop) มีลักษณะการวิ่งของลายนูนจากทางด้านขวามาที่บริเวณกลางนิ้วมือ แล้ววิ่งโค้งขึ้นไปด้านบนแล้วจึงโค้งลง และวิ่งวนไปตามทางเดิมที่ได้วิ่งมา

รูปที่ 2.15 แสดงลักษณะของมัดหวายขวา

2. **กลุ่มมัดหวายคู่ (Double Loop)** เป็นกลุ่มของรูปแบบที่เกิดจากการรวมกันของกลุ่มมัดหวายเดี่ยวสองกลุ่มรวมกันเป็นรูปแบบเดียว โดยลักษณะการวิ่งของลายนูนอาจมาจากทางด้านเดียวกันแล้วรวมตัวกันเป็นกลุ่มมัดหวายเดี่ยวสองกลุ่ม โดยที่กลุ่มหนึ่งจะมีรูปแบบเป็นลักษณะโค้งขึ้น และอีกกลุ่มจะมีลักษณะโค้งลง



รูปที่ 2.16 แสดงลักษณะของมัดหวายคู่

2.1.7.2 กลุ่มก้นหอย (Whorl)

ลักษณะการวิ่งของลายนูนมีรูปแบบเป็นลักษณะเป็นโค้งเป็นรูปวงกลมจากวงใหญ่ ค่อยๆ เล็กลงไปเรื่อยๆ จนเหลือวงกลมวงเล็กที่สุดอยู่ด้านใน กลุ่มก้นหอยนี้จะมี Delta ตั้งแต่สองกลุ่มขึ้นไป ประมาณ 30 – 35% ของมนุษย์จะมีลายนิ้วมือในกลุ่มนี้

รูปที่ 2.17 แสดงลายนิ้วมือของกลุ่มก้นหอย (Whorl)

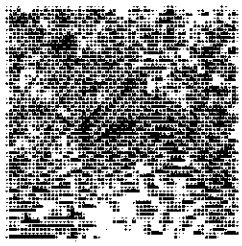
2.1.7.3 กลุ่มเส้นโค้ง (Arch)

มีลักษณะการวิ่งของลายนูน จากลักษณะที่ขนานกับพื้นราบแล้วพุ่งโค้งขึ้น แล้วจึงมีการวิ่งในลักษณะขนานกับพื้นราบอีกครั้ง กลุ่มเส้นโค้งนี้จะไม่มี Delta ให้เห็นประมาณ 5% ของมนุษย์จะมีลายนิ้วมือในกลุ่มนี้

รูปที่ 2.18 แสดงลายนิ้วมือของกลุ่มเส้นโค้ง (Arch)

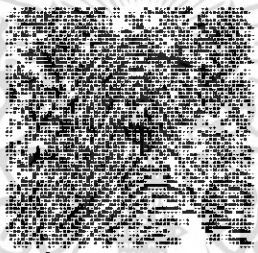
กลุ่มเส้นโค้งสามารถแบ่งรูปแบบได้เป็น 2 ประเภท คือ

1. **โค้งราบ (Plane Arch)** เป็นกลุ่มโค้งที่ลายเส้นวิ่ง หรือไหลออกไปข้างหนึ่งโดยไม่เกิดมุมแหลม หรือพุ่งขึ้นตรงกลาง



รูปที่ 2.19 แสดงลักษณะของโค้งราบ

2. **โค้งกระโจม (Tented Arch)** ลายเส้นตรงกลางเกิดเป็นลายเส้นพุ่งขึ้นโดยเป็นมุมแหลม หรือมุมฉาก

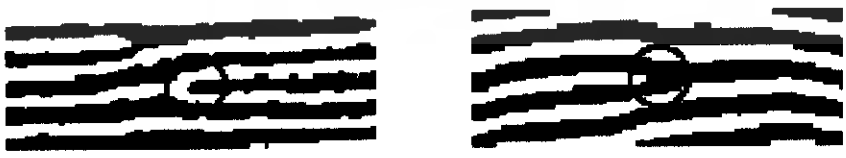


รูปที่ 2.20 แสดงลักษณะของโค้งกระโจม

2.1.8 ลักษณะเฉพาะ (Minutiae)

เป็นลักษณะเฉพาะในลายนิ้วมือแต่ละคน ซึ่งประกอบไปด้วยลักษณะสำคัญ 3 ชนิด คือ

1. จุดแยกลายนูน (Ridge Bifurcation) เป็นจุดที่อยู่บนลายนูนที่เกิดการแยกจากหนึ่งเส้นทางเป็นสองเส้นทาง
2. จุดปลายของลายนูน (Ridge Ending) เป็นจุดที่อยู่บนลายนูนในบริเวณปลายสุดของลายนูน
3. จุดพื้นที่ปิดล้อม (Enclosure) เป็นจุดที่อยู่บนลายนูนที่เกิดการแยกจากหนึ่งเส้นทางเป็นสองเส้นทาง แล้วสองเส้นทางนั้นกลับมาบรรจบเป็นเส้นทางอีกครั้งซึ่งทำให้เกิดพื้นที่ปิดล้อมที่ไม่มีลายนูนอยู่ภายในขึ้น



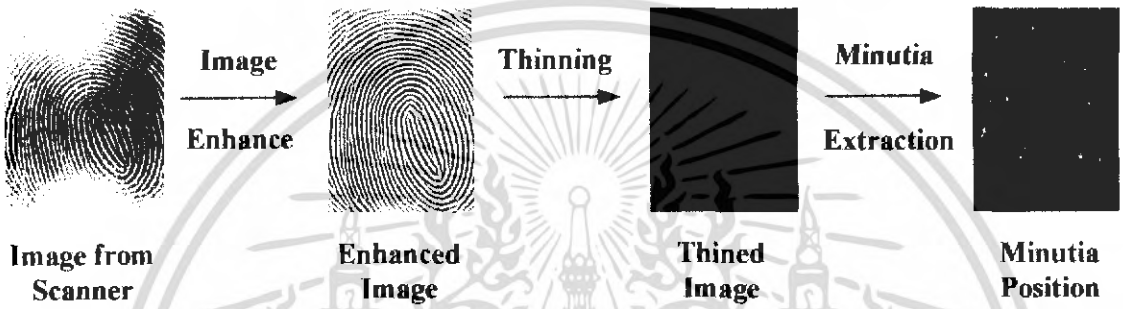
รูปที่ 2.21 แสดงลักษณะเฉพาะ (Minutiae)

2.1.9 การ Matching ลายนิ้วมือ

ลายนิ้วมือสองลายที่เหมือนกันนั้น เราสามารถทำการเปรียบเทียบกันว่าทั้งสองลายตรงกันหรือไม่โดยอาศัยปัจจัย 4 ข้อ คือ

1. ข้อตกลงเรื่องรูปร่างโดยทั่วไป
2. ระดับที่ตรงกัน
3. ปัจจัยในการแบ่งชนิด
4. ความสัมพันธ์ของลักษณะเฉพาะ (Minutiae)

2.1.10 กระบวนการประมวลผลหาลักษณะลายนิ้วมือ



รูปที่ 2.22 กระบวนการประมวลผลหาลักษณะลายนิ้วมือ

กระบวนการประมวลผลลายนิ้วมือ มีไว้เพื่อหาลักษณะจุดเด่นของลายนิ้วมือของแต่ละบุคคลเพื่อนำไปเก็บไว้ในฐานข้อมูล เพื่อใช้ในการระบุบุคคลเมื่อทำการตรวจลายนิ้วมือในครั้งต่อไป สามารถแบ่งกระบวนการคร่าวๆได้ดังนี้

1. การปรับคุณภาพของภาพ (Image Enhance)

เนื่องจากภาพจากเครื่องสแกนได้ภาพที่ไม่คมชัดอาจทำให้ การหาลักษณะเด่นเกิดข้อผิดพลาดได้จึงต้องมีการประมวลผลภาพในเบื้องต้น (Pre-Processing) ก่อน จากรูปจะเป็นการปรับคุณภาพของภาพ โดยวิธี Short Time Fourier Transform (STFT)

2. การทำภาพให้บาง, การทำภาพขาวกับดำ และกำจัดเส้นสะพาน (Binalization, Thinning and Imperfect Removal)

หลังจากได้ภาพที่ผ่านการปรับปรุงมาแล้วทำการปรับสีให้เหลืออยู่เพียง 2 ระดับ (Binalization) คือ ขาว กับ ดำ โดยการใช้ Otsu's Tredhold ตัดระดับสีของแต่ละพิเซล

หลังจากนั้นทำให้ภาพบาง (Thinning) โดยการใช้หลังของมอร์โฟโลยี นำพิเซลที่ไม่ต้องการออก และให้ภาพที่ผ่านกระบวนการทำให้บางแล้วจะอยู่ในรูป 8-Connectivity

แต่ทว่ารูปที่ผ่านการทำให้บางแล้วในความเป็นจริงจะเกิดเส้นกิ่งและเส้นสะพาน ข้อผิดพลาดเหล่านี้ถ้าหากปล่อยเอาไว้เมื่อถึงกระบวนการหาจุดเด่นลายนิ้วมือ จะทำให้เกิดข้อผิดพลาดได้ จึงต้องทำการกำจัดออกเรียกกระบวนการนี้ว่า การกำจัดเส้นกิ่งเส้นสะพาน (Imperfect Removal หรือ Post-Processing)

3. การหาจุดเด่นของลายนิ้วมือ (Minutia Extraction)

ในการบวนการนี้จะทำการค้นหาพิเซลที่เป็นจุดแยก และจุดปลายของลายนิ้วมือเพื่อเก็บค่าให้อยู่ในรูปของตำแหน่ง และมุมของรูปภาพ แล้วจึงนำค่าเก็บไว้ในฐานข้อมูล ซึ่งเรียกว่า เทมเพลต (Template) เพื่อรอการเรียกใช้งาน

4. การหาเปอร์เซ็นต์แมทช์ (Matching Minutia)

กระบวนการนี้จะนำลายนิ้วมือของแต่ละบุคคลมาผ่านกระบวนการเพื่อให้ได้เทมเพลต ซึ่งจะนำเทมเพลตนี้ไปทำการเทียบกับฐานข้อมูล เพื่อหาเปอร์เซ็นต์ความเหมือน ซึ่งเปอร์เซ็นต์ความเหมือนนี้เอง ผู้ดูแลระบบจะเป็นผู้ที่กำหนดว่า เปอร์เซ็นต์ความเหมือนต้องไม่ต่ำกว่ากี่เปอร์เซ็นต์จึงจะตัดสินใจว่าใช่บุคคลนั้นหรือไม่

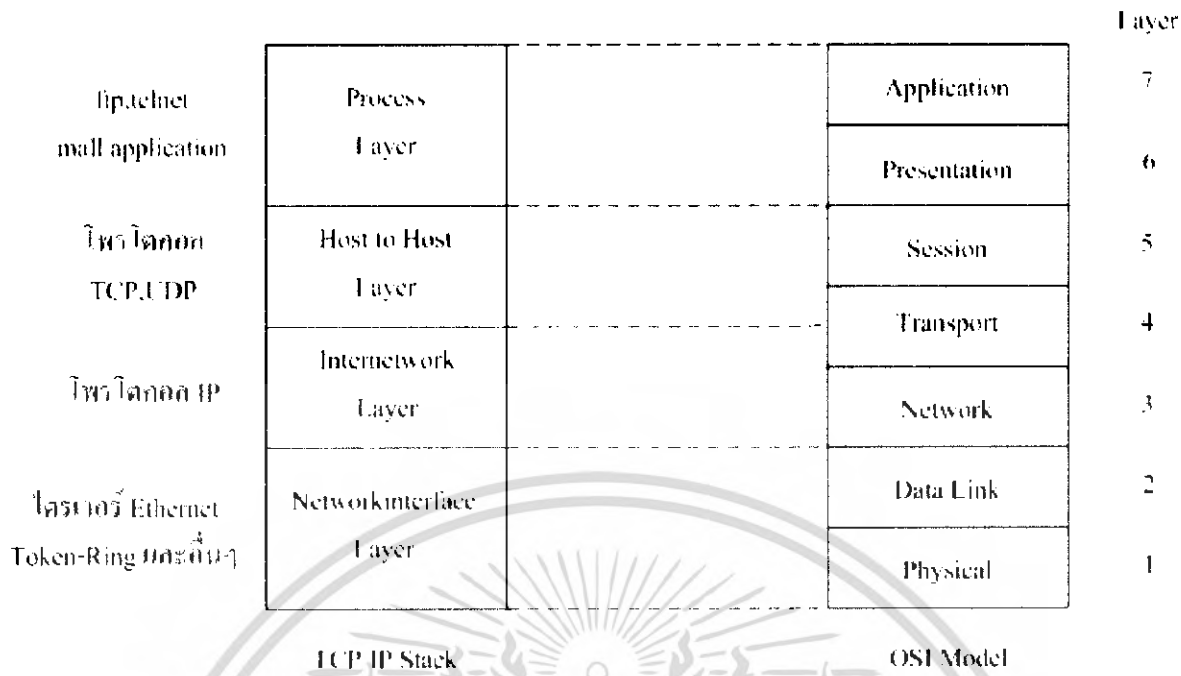
2.2 โพรโตคอล TCP/IP

2.2.1 โครงสร้างของโพรโตคอล TCP/IP

โพรโตคอล TCP/IP มีการจัดกลไกการทำงานเป็นชั้นหรือ Layer เรียงต่อกัน โดยในแต่ละ Layer จะมีการทำงานเทียบได้กับ OSI model มาตรฐาน แต่บาง Layer ของโพรโตคอล TCP/IP จะทำงานเทียบกับ OSI หลาย Layer ปนกัน ซึ่งในแต่ละ Layer ของโพรโตคอล TCP/IP จะประกอบด้วย

- Process Layer
- Host – to – Host Layer
- Internetwork Layer
- Network Interface Layer

โดยเมื่อเทียบกับมาตรฐาน OSI model ดังรูปที่ 2.23 ซึ่งเราจะเห็นว่าบางกลไกของโพรโตคอล TCP/IP เทียบได้กับมาตรฐาน OSI model สองชั้น หรือบางกลไกก็จะทำงานคาบเกี่ยวกันระหว่างบางชั้นของ OSI model ตัวอย่างเช่น กลไกการทำงานของโพรโตคอล TCP/IP ในส่วน Network Interface Layer เมื่อเทียบกับมาตรฐาน OSI model จะเทียบได้กับ Data Link Layer และ Physical Layer 2 ชั้นรวมกัน เป็นต้น ในแต่ละกลไกของโพรโตคอล TCP/IP และโพรโตคอลอื่น ๆ ในชุดของ TCP/IP ร่วมทำงานอยู่ด้วย



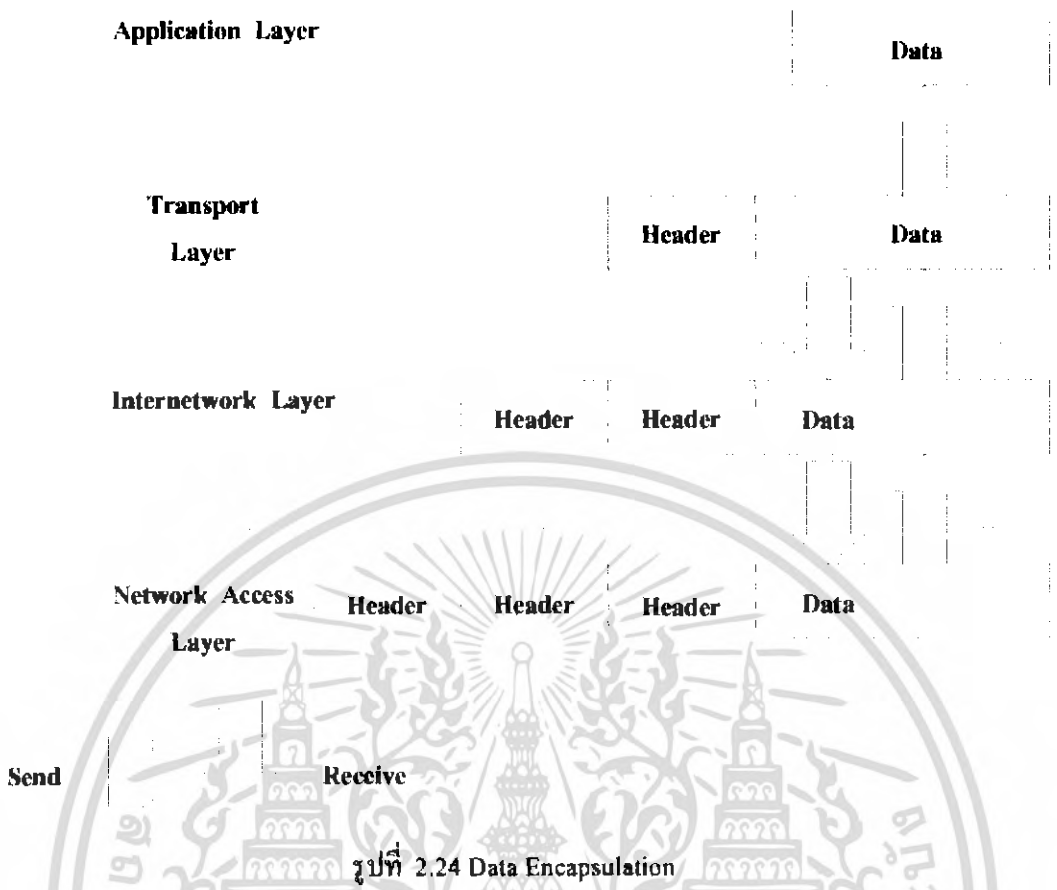
รูปที่ 2.23 แสดงกลไกของโพรโตคอลมาตรฐาน OSI model

สำหรับการศึกษาโพรโตคอล TCP/IP นั้นเราจะไม่อ้างอิง OSI Reference Model นี้เพราะจะเข้าใจได้ยาก ดังนั้นเราจึงจะสร้างโมเดลขึ้นมาใหม่โดยแบ่งออกเป็น 4 ชั้นดังนี้

4	Application Layer
3	Host - to - Host Transport Layer
2	Internet Layer
1	Network Access Layer

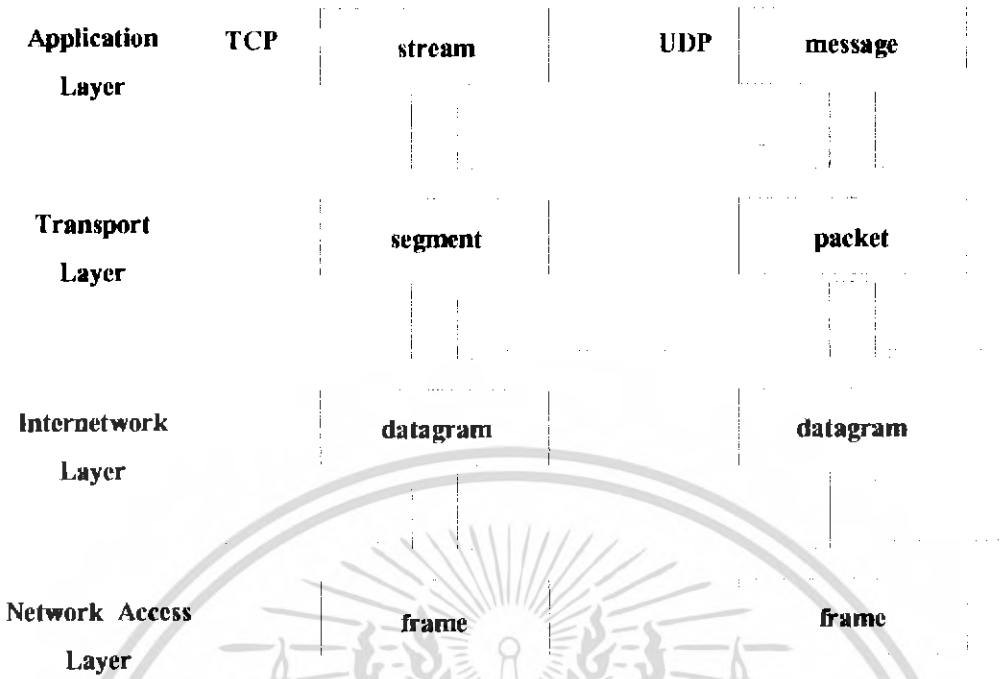
ลักษณะการทำงานของโมเดลในลักษณะนี้คือ ข้อมูลจะถูกส่งลงมาจากชั้นข้างบนลงมายังชั้นข้างล่างสุดซึ่งมีหน้าที่จัดการเกี่ยวกับการส่งข้อมูลผ่านสายสัญญาณไปยังจุดหมายปลายทาง เมื่อข้อมูลไปถึงจุดหมายแล้วก็จะกลับย้อนจากชั้นล่างขึ้นไปชั้นบนสุด ซึ่งเป็นชั้นที่โปรแกรมใช้งานต่างๆ ทำงานอยู่

ขณะที่ข้อมูลถูกส่งผ่านจากชั้นบนลงมายังชั้นล่าง แต่ละชั้นจะทำการเพิ่มข้อมูลควบคุมเข้าไปเพื่อให้การส่งข้อมูลถูกต้อง และเป็นการส่งพารามิเตอร์ที่จำเป็นไปให้กับชั้นของมันในเครื่องปลายทาง ข้อมูลควบคุมเหล่านี้เราเรียกว่า Header แต่ละชั้นจะมี Header ที่มีรูปแบบเป็นของตัวเอง การเพิ่มเฮดเดอร์เข้าไปกับข้อมูลนั้นเราเรียกว่า Data Encapsulation ซึ่งได้แสดงไว้ในรูปที่ 2.24



รูปที่ 2.24 Data Encapsulation

หน่วยของข้อมูลที่จะทำการส่งนั้นนั้นจะมีชื่อเรียกต่างกันเมื่อมันเดินทางผ่านแต่ละ Layer ดังแสดงไว้ในรูปที่ 2.25 ซึ่งจะเห็นว่าการส่งใน TCP/IP นั้นมีอยู่ 2 โพรโตคอล ย่อยคือ TCP กับ UDP ชื่อของข้อมูลที่ผ่านโพรโตคอลทั้งสองนั้นแตกต่างกันในชั้นบนๆ เท่านั้น ชื่อสำคัญๆ ที่เราจะต้องพบและใช้ต่อไปบ่อยๆ ได้แก่ คาต้าแกรม และเฟรม



รูปที่ 2.25 โครงสร้างของข้อมูล

2.2.2 Network Interface Layer

เนื่องจากในด้านกายภาพของเครือข่ายนั้น มีหลายวิธีการและหลายรูปแบบในการเชื่อมต่อระบบให้เป็นเครือข่าย แต่อย่างไรก็ตามในเครือข่ายอินเตอร์เนตนี้ ข้อมูลหรือ IP datagram จะถูกถ่ายทอดและส่งผ่านไปยังปลายทางโดยไม่คำนึงถึงรูปแบบการเชื่อมต่อทางกายภาพ ไม่ว่าจะเป็นการใช้เครือข่ายใยแก้วนำแสงหรือเครือข่ายสาย Unshielded Twist Pair (UTP) เชื่อมต่อเป็นแบบเครือข่ายอีเทอร์เนต ธรรมดาหรือเครือข่าย Token Ring, ATM, ISDN ฯลฯ ก็ตาม

2.2.2.1 อีเทอร์เนต

อีเทอร์เนตเป็นการใช้สายโคแอคเชียลเบสบัสเชื่อมต่อระบบเข้าด้วยกันเพื่อทำการส่งถ่ายข้อมูลในระบบดิจิทัลระหว่างคอมพิวเตอร์ โดยบริษัทอุปกรณ์ดิจิตอล บริษัท Intel และบริษัท Xerox ได้ใช้ระบบนี้อ้างอิงเรื่อยมา ซึ่งอีเทอร์เนตจะใช้เทคนิคการส่งเบสแบนด์ในการเข้าถึงข้อมูล และยังสามารถใช้บนสายโคแอคเชียลที่มี 2 ขนาด คือ สายอีเทอร์เนตแบบหนา และสายอีเทอร์เนตแบบบาง โดยสายเคเบิลทั้งสองนี้เป็นที่ใช้กันอย่างกว้างขวางในปัจจุบัน

ส่วนประกอบหลักที่สำคัญของเครือข่ายอีเทอร์เนต

ระบบเครือข่ายอีเทอร์เนต มีส่วนประกอบหลักซึ่งเมื่อทำงานด้วยกันแล้วก็จะเป็นเครือข่ายที่มีประสิทธิภาพการทำงานสูงดังนี้

1. ตัวเฟรมเป็นชุดรูปแบบของบิตข้อมูลข่าวสารที่ใช้ส่งผ่านมาบนระบบ หากไม่มีเฟรมเราจะไม่สามารถสื่อสารข้อมูลบนเครือข่ายได้โดยเด็ดขาด การรับส่งข้อมูลข่าวสารบนเครือข่ายอีเทอร์

เนต จะต้องเป็นไปในรูปแบบเฟรมมาตรฐาน 2 แบบ และเป็นแบบใดแบบหนึ่งเท่านั้น (การ์ด LAN เป็นผู้สร้างเฟรมนี้ขึ้นมา)

2. ชุดโปรโตคอลที่ใช้ในการควบคุมการแอกเซสเข้าไปที่เครือข่าย (Media Access Control Protocol) ซึ่งประกอบด้วยชุดของกฎกติกาที่อยู่ใน Ethernet Interface (เช่น การ์ด LAN เป็นต้น) ซึ่งเป็นกฎมาตรฐานที่จะยอมให้คอมพิวเตอร์ต่างๆสามารถเข้ามาที่เครือข่าย และแบ่งใช้ทรัพยากรต่างๆ บนเครือข่ายได้อย่างมีประสิทธิภาพ
3. อุปกรณ์ที่ใช้รับส่งสัญญาณบนเครือข่าย (Signaling Components) ประกอบด้วยชุดของอุปกรณ์ที่ใช้เชื่อมต่อและส่งสัญญาณเพื่อการรับส่งข้อมูลภายในเครือข่าย
4. สื่อที่ใช้ในการรับส่งสัญญาณข้อมูลบนเครือข่าย (Physical Medium) ประกอบด้วยสายสัญญาณ รวมทั้งอุปกรณ์ทางฮาร์ดแวร์อื่นๆ ที่จะช่วยในการนำพาข้อมูลข่าวสารต่างๆในรูปแบบดิจิทัลวิ่งไปมาบนเครือข่าย

2.2.2.2 เฟรมอีเทอร์เนต

อีเทอร์เนตได้ถูกระบุไว้อย่างแน่นอนไว้ในชั้น Physical Layer โดยมันจะแสดงรายละเอียดของรูปแบบเป็นแพ็กเก็ต ซึ่งโดยส่วนมากจะเรียกว่า เฟรม ซึ่งเป็นหัวใจสำคัญของระบบอีเทอร์เนต จากรูปที่ 2.26 ได้แสดงรูปแบบของเฟรมอีเทอร์เนต โดยเฟรมนี้มันจะ Encapsulates TCP/IP โปรโตคอล และสามารถทำการตอบสนองสำหรับส่งข้อมูลข้ามเข้าระบบที่เชื่อมต่อ ไปยังอีก Layer ได้โดย Gateway หรือ End Node

Preamble	Destination MAC Address (6 Byte)	Source MAC Address (6 Byte)	Type (2 Byte)	Data Field (1500 Byte Max)	Cyclic Redmdancy Check (4 Byte)

รูปที่ 2.26 ลักษณะของเฟรมอีเทอร์เนต

2.2.2.3 MAC Address

เนื่องจากคอมพิวเตอร์แต่ละเครื่องสามารถแชร์ข้อมูลกันได้ในระบบเครือข่ายเดียวกัน ดังนั้นแต่ละเครื่องควรมีสิ่งที่ชี้ลักษณะเฉพาะตัวของมัน เช่น เราต้องมีบัตรประจำตัวประชาชน ซึ่งในทางคอมพิวเตอร์นี้เราจะใช้เลขฐาน 16 จำนวน 12 digits เป็นตัวบ่งชี้ลักษณะเฉพาะนั้น ๆ ซึ่งเราเรียกว่า MAC Address เนื่องจาก MAC Address เป็นตัวบ่งชี้ลักษณะเฉพาะของแต่ละเครื่อง ดังนั้นจึงต้องเป็นค่าที่ไม่ซ้ำกัน (Unique) MAC Addressเป็นเลข 48 บิตโดยแบ่งออกเป็น 2 ส่วนโดย 24 บิตแรกเป็นค่าที่แสดงถึงบริษัทที่ผลิตการ์ดนั้น ๆ ส่วน 24 บิต หลังเป็น Serial Number ที่ทางบริษัทกำหนดให้ ซึ่งแต่ละตัวต้องไม่ซ้ำกัน เราเรียกเลข 24 บิต นี้ว่า OUI (Organizationally Unique Identifier) ซึ่ง OUI จะใช้เพียง 22 บิตเท่านั้น ส่วนอีก 2 บิตที่เหลือจะถูกใช้เพื่อวัตถุประสงค์อื่น โดยบิตหนึ่งจะใช้เพื่อแสดงว่าแอดเดรสนั้นเป็น Broadcast/Multicast Address ส่วนอีกบิตหนึ่งนั้นไว้แสดงว่า Adapter นั้นถูกกำหนด Locally Administered Address ซึ่ง Admin ของระบบจะทำการกำหนด MAC Address เพื่อความเหมาะสมของนโยบายระบบ เช่น MAC Address = 03 00 00 00 00 00 01 ซึ่ง

จะเห็นว่าไบต์แรก = 03 = 00000011 นั่นคือ ทั้ง 2 bits ถูก set (reset = 0) ซึ่งเอาไว้กรณี Multicast ให้ทุกเครื่องที่รันบนโปรโตคอล NetBEUI

2.2.2.4 Type field

ประเภทของฟิลด์จะใช้ระบุความแตกต่างของโปรโตคอล คอมพิวเตอร์จะดำเนินการให้โปรโตคอลหลายๆตัว สามารถเห็นความแตกต่างของพวกมันได้ง่ายและผ่านการยินยอมของเฟรมที่เกี่ยวเนื่องกับเครือข่ายระบบ TCP/IP โดยทั่วไปจะใช้อีเทอร์เนต 3 ฟิลด์ ซึ่งมีค่าดังตารางที่ 2.1 ประเภทของหมายเลขอีเทอร์เนตเป็นรีจีสเตอร์กับ IEEE โดยจะใช้หมายเลขที่เฉพาะเจาะจงไม่เหมือนใคร

ตารางที่ 2.1 Ethernet type fields

Type	Protocol
0x0800	IP
0x0806	ARP
0x0835	RARP

2.2.2.5 Cyclic Redundancy Check (CRC)

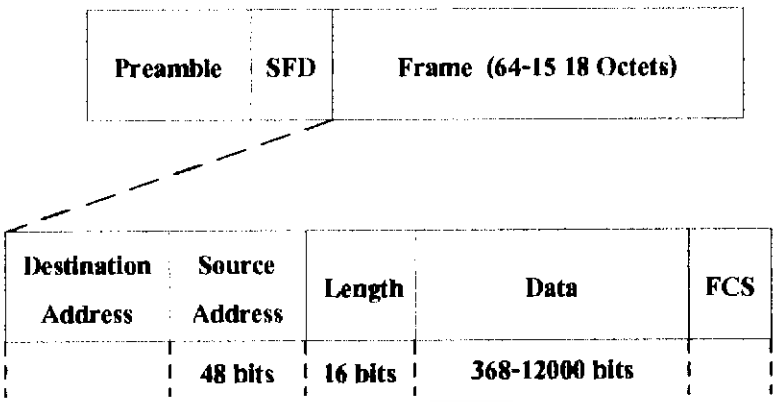
ที่ปลายสุดของเฟรม คือ Cyclic Redundancy Check (CRC) มีขนาด 32 บิต ที่คำนวณได้จากบิตทั้งหมดของอีเทอร์เนตเฟรม แต่จะไม่สนใจ Preamble ถ้าในเฟรมมีความผิดพลาดเกิดขึ้น ค่าที่คำนวณได้จะแตกต่างไปจากเฟรมเดิม ทำให้ข้อมูลไม่สามารถที่จะผ่านเฟรมไปยังชั้น Network layer ได้

2.2.2.6 IEEE 802.3 เฟรม

IEEE 802.3 หรือ อีเทอร์เนต (Ethernet) เป็นเครือข่ายที่มีความเร็วสูงการส่งข้อมูล 10 เมกะบิตต่อวินาที สถานีในเครือข่ายอาจมีโทโปโลยีแบบบัสหรือแบบดาว IEEE ได้กำหนดมาตรฐานอีเทอร์เนตซึ่งทำงานที่ความเร็ว 10 เมกะบิตต่อวินาทีไว้หลายประเภทตามชนิดสายสัญญาณเช่น

- 10Base5 อีเทอร์เนตโทโปโลยีแบบบัสซึ่งใช้สายโคแอกเชียลแบบหนา (Thick Ethernet) ความยาวของสายในเซกเมนต์หนึ่ง ๆ ไม่เกิน 500 เมตร
- 10Base2 อีเทอร์เนตโทโปโลยีแบบบัสซึ่งใช้สายโคแอกเชียลแบบบาง (Thin Ethernet) ความยาวของสายในเซกเมนต์หนึ่ง ๆ ไม่เกิน 185 เมตร
- 10BaseT อีเทอร์เนตโทโปโลยีแบบดาวซึ่งใช้ฮับเป็นศูนย์กลาง สถานีและฮับเชื่อมด้วยสายยูทีพี (Unshield Twisted Pair) ด้วยความยาวไม่เกิน 100 เมตร

เฟรมข้อมูลสำหรับระบบอีเทอร์เน็ตประกอบขึ้นด้วยกลุ่มของบิตที่เป็นข้อมูลและข่าวสารสำคัญ แบ่งออกเป็นขนาดสัดส่วนที่แน่นอนที่เรียกว่าช่อง Field ดังรูปที่ 2.27



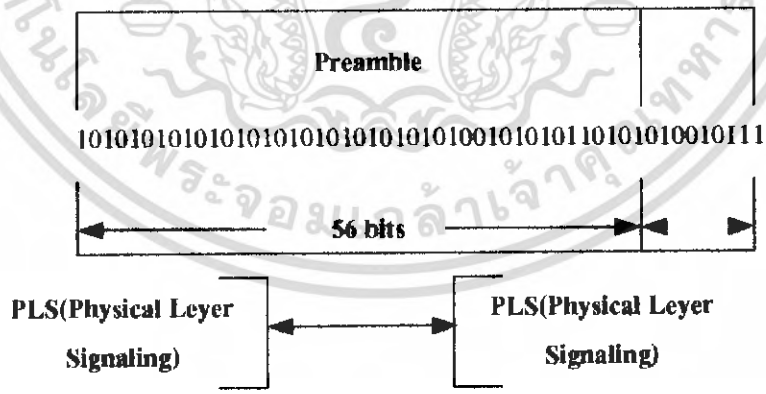
รูปที่ 2.27 ลักษณะโครงสร้างของเฟรมข้อมูลตามมาตรฐาน IEEE802.3

รูปที่ 2.27 แสดงให้เห็นรูปแบบของเฟรมข้อมูลที่ใช้บนอีเทอร์เนตตามมาตรฐาน IEEE802.3 ต่อไปเราจะมาทำความเข้าใจเฟรมมาตรฐานของ IEEE802.3

ช่อง Preamble

ช่อง Preamble ประกอบด้วยบิตข่าวสารที่เป็นเลข 1 และ 0 สลับกัน และสิ้นสุดที่ 11 ซึ่งเป็นบิตที่ 63 และ 64 เป็นบิตข่าวสารที่ยังไม่ใช่ข้อมูลจริงของผู้ส่ง Preamble ประกอบด้วยข่าวสารที่มีขนาด 7 หรือ 8 ไบต์ จุดประสงค์ของข่าวสารนี้ก็เพื่อใช้สร้างจังหวะการรับข้อมูลให้แก่ผู้รับโดยที่ส่วนนี้จะไปถึงตัวผู้รับก่อน ทำให้เครื่องคอมพิวเตอร์ของผู้รับสามารถปรับจังหวะความเร็วให้เข้ากับผู้ส่งได้ (Synchronize) สำหรับเฟรมแบบ Ethernet II จะมีขนาด 8 ไบต์ และถ้าเป็นมาตรฐาน IEEE802.3 แล้ว ช่องนี้จะถูกแบ่งออกเป็น 2 ส่วน (ดังรูปที่ 2.28) ได้แก่

1. Preamble
2. Start of Frame Delimiter



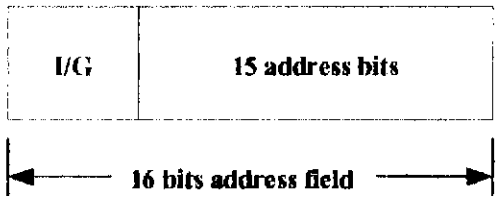
รูปที่ 2.28 ลักษณะส่วนการทำงานภายในของ Preamble

ช่อง Destination Address

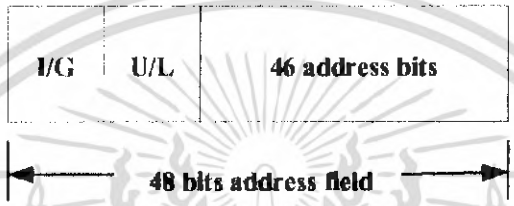
ในช่อง Destination Address ประกอบด้วยข้อมูลข่าวสารเกี่ยวกับแอดเดรสหรือที่อยู่ของผู้รับปลายทาง ดูเผินๆแล้วเป็นช่องที่เรียบง่ายไม่มีอะไรมาก แต่โดยความจริงแล้วช่องนี้ถูกแบ่งออกเป็นช่องย่อยๆที่

เรียกว่า Sub-Fields ซึ่งเก็บข้อมูลข่าวสารที่ใช้ดูแลการทำงานของเครือข่าย ทั้งนี้ขึ้นอยู่กับแอดเดรสที่ปรากฏอยู่ในช่องนี้ไม่ว่าช่องแอดเดรสนี้จะมีแอดเดรสที่ถูกเจาะจงเป็นรายบุคคลหรือเป็นกลุ่มของผู้รับก็ตาม

ก. ช่องข่าวสารขนาด 2 ไบต์



ข. ช่องข่าวสารขนาด 6 ไบต์



หมายเหตุ

หากค่าบิตในช่องย่อย I/G มีค่าเป็น "0" ก็หมายถึงแอดเดรสที่ระบุตัวคอมพิวเตอร์ผู้รับอย่างเฉพาะเจาะจง แต่ถ้ามีค่าเป็น "1" ก็หมายถึงคอมพิวเตอร์ที่ระบุแอดเดรสเป็นกลุ่ม ไม่เจาะจงเครื่องใดเครื่องหนึ่ง

หากค่าบิตในช่อง U/L มีค่าเป็น "0" ก็หมายความว่าแอดเดรสนี้ถูกกำหนดมาตรฐานโดย IEEE และถ้ามีค่าเป็น "1" ก็จะหมายถึงเป็นแอดเดรสมาตรฐานเฉพาะองค์กรที่ระบุมาตรฐานในซอฟต์แวร์ ซึ่งแอดเดรสมาตรฐานที่เราใช้กันอยู่ทุกวันนี้ถูกควบคุมโดย IEEE

ช่อง I/G

มีการจัดตั้งค่าบิตขึ้นในช่องนี้โดยการ์ด LAN ซึ่งหากค่านี้ถูกตั้งค่าไว้ที่ "0" ก็แสดงว่าตัวแอดเดรสที่ระบุอยู่ในช่อง Destination Address นั้นเป็นแอดเดรสที่ระบุตัวคอมพิวเตอร์ผู้รับแบบเฉพาะเจาะจง แต่ถ้าถูกตั้งค่าเป็น "1" ก็แสดงว่าแอดเดรสในช่อง Destination Address นี้เป็นแอดเดรสที่ใช้ติดต่อผู้รับที่เป็นกลุ่มคอมพิวเตอร์ทั้งหลาย เราเรียก Group Address ตัวอย่างของ Group Address ได้แก่ "FFFFFFFFFFFF" ซึ่งถือว่าเป็น Broadcasting Address หรือแอดเดรสที่ไม่เจาะจงผู้รับ โดยผู้รับเป็นกลุ่มหรือทั้งหมดก็สามารถรับข้อมูลข่าวสารนี้ได้

ช่องย่อย U/L

ช่องย่อย U/L มีไว้สำหรับช่องขนาด 6 ไบต์เท่านั้น ค่าที่ถูกตั้งไว้ในช่องย่อยนี้เป็นการบ่งบอกให้ทราบว่าแอดเดรสที่ปรากฏอยู่ในช่อง Destination Address นี้เป็นแอดเดรสที่ถูกกำหนดมาตรฐานโดย IEEE หรือองค์กรอย่างเฉพาะเจาะจง

ช่อง Source Address

สำหรับช่อง Source Address นี้มีไว้เพื่อแสดงตัวสถานีเครือข่ายต้นทางที่เป็นต้นทางส่งข้อมูลข่าวสารเข้ามาและเช่นเดียวกับช่อง Destination Address กล่าวคือ ช่อง Source Address สามารถมีช่องย่อยได้ทั้งแบบ 2 ไบต์หรือ 6 ไบต์อย่างใดอย่างหนึ่ง

ตารางที่ 2.2 เป็นตัวอย่างของ Source Address ที่แสดงรหัสแอดเดรสของผู้ผลิตดังนี้คือ

ผู้ผลิตการ์ด LAN	รหัสผู้ผลิตขนาด 3 ไบต์
Cisco	00-00-0C
Cabletron	00-00-1D
Intel	00-AA-00
3 Com	02-60-8C
Hewlett Packard	08-00-09
Sun	08-00-20
DEC	08-00-2B
Shiva	00-80-D3
Xerox	00-00-AA
IBM	08-00-5A

ช่องแสดง Type

ช่องแสดง Type มีขนาด 2 ไบต์ ใช้กับอีเทอร์เนตเฟรม เท่านั้น โดยช่องนี้ใช้เพื่อแสดงว่าโปรโตคอลการทำงานของเฟรมนี้เป็นแบบใด จุดประสงค์คือเพื่อต้องการให้ทราบว่าข้อมูลที่อยู่ในเฟรมนี้ จะทำงานภายใต้โปรโตคอลใด ซึ่งผู้รับจะได้เตรียมการแปลความหมายที่อยู่ในช่องข้อมูล (Data Field) ได้ถูกต้อง

ภายใต้ระบบเครือข่ายอีเทอร์เนตเราสามารถใช้โปรโตคอลได้หลายตัวพร้อมกันบนเครือข่าย LAN และบริษัท XEROX ทำหน้าที่เป็นผู้ให้บริการ กำหนดพิกัดระยะของแอดเดรสที่เป็นลิขสิทธิ์ให้แก่ผู้ผลิตการ์ด LAN ต่างๆรวมทั้งการกำหนดค่าที่ใช้แสดงแทนโปรโตคอลที่ใช้ในช่อง Type แห่งนี้

ตารางที่ 2.3 เป็นตัวอย่างของรหัสที่ใช้แสดงแทนโปรโตคอลที่ใช้ในช่อง Type 18 รายการดังนี้

โปรโตคอลที่ใช้	ค่าที่เป็นรหัสแบบเลขฐาน 16
IP	0800
X.75 Internet	0801
X.25 Level 3	0805
Address Resolution Protocol (ARP)	0806
Banyan Systems	0BAD
BBN Simnet	5208

DEC MOP Dump/Load	6001
DEC MOP Remote Console	6002
DEC DECNET Phase IV Route	6003
DEC LAT	6004
DEC Diagnostic Protocol	6005
DEC LANBridge	8038
DEC Ethernet Encryption	803D
Apple Talk	809B
IBM SNA Service on Ethernet	80D5
Apple Talk ARP	80F3
NetWare IPX/SPX	8137
SNMP	814C

ช่อง Length

ช่องนี้มีขนาดความยาวเพียง 2 ไบต์ใช้ได้กับเฟรมมาตรฐาน IEEE802.3 เท่านั้นเป็นช่องที่ใช้แสดงขนาดจำนวนของไบต์ที่มีปรากฏอยู่ในช่อง Data

ช่อง Data (Data Field)

ดังที่ได้กล่าวมาแล้วว่าช่องของ Data อย่างน้อยต้องมีขนาดไม่เล็กกว่า 46 ไบต์ เพื่อให้แน่ใจว่าเฟรมมีขนาดไม่ต่ำกว่า 64 ไบต์ซึ่งหมายความว่าการแพร่ข้อมูลขนาดหนึ่งไม่ว่า 1 หรือ 10 ไบต์ก็ตามต้องมาจาก 46 ไบต์นี้แต่ถ้าข้อมูลในช่องนี้เล็กกว่า 46ไบต์ แน่นอนว่าต้องมีการเพิ่มไบต์ลงไปอีกเพื่อให้ได้ขนาด 46 ไบต์อยู่ดี ขนาดของข้อมูลที่อยู่ใน Data จะต้องมีขนาดสูงสุดไม่เกิน 1,500 ไบต์

ช่องตรวจสอบความผิดพลาดของข้อมูลในเฟรม (Frame Check Sequence)

ช่อง Frame Check Sequence นี้ใช้ได้ในเฟรมมาตรฐาน ทั้งเฟรมมาตรฐาน ทั้งอีเทอร์เนตและ IEEE802.3 เป็นช่องที่ประกอบด้วยข้อมูลที่ใช้เป็นกลไกในการตรวจสอบความผิดพลาดของข้อมูลภายในเฟรม

หลักการทำงานมีอยู่ว่าก่อนที่เครื่องผู้ส่งจะส่งข้อมูลออกไปที่เครือข่าย การ์ด LAN ของมันจะคำนวณค่าต่างๆในช่องต่างๆซึ่งครอบคลุมตั้งแต่ช่อง Address ต่างๆของ Type และช่อง Length รวมทั้งช่อง Data การคำนวณค่าแบบนี้เรียกว่า Cyclic Redundancy Check (CRC) ซึ่งหลังจากที่ได้คำนวณค่าเสร็จสิ้นแล้ว ผลลัพธ์ที่คำนวณได้มีขนาด 4 ไบต์จะถูกนำไปใส่ไว้ในช่อง Frame Check Sequence แห่งนี้

2.2.3 Internetwork Layer

ในระดับล่างต่อมาในชั้น Internetwork Layer มีหน้าที่ส่งผ่านข้อมูลในระหว่างเครือข่าย โดยมีโพรโตคอลที่ทำงานเป็นกลไกสำคัญในการส่งผ่านข้อมูลไปยังเครือข่ายใด ๆ บนอินเตอร์เน็ต คือ โพรโตคอล IP (Internet Protocol) นอกจากนี้ในชั้น Internetwork Layer ยังมีโพรโตคอลทำงานอยู่ด้วยอีก 2 ชนิดคือ

โพรโตคอล Internet Control Message Protocol (ICMP) และโพรโตคอล Address Resolution Protocol (ARP) ซึ่งอยู่ภายในโพรโตคอล IP

2.2.3.1 โพรโตคอล IP (Internet Protocol)

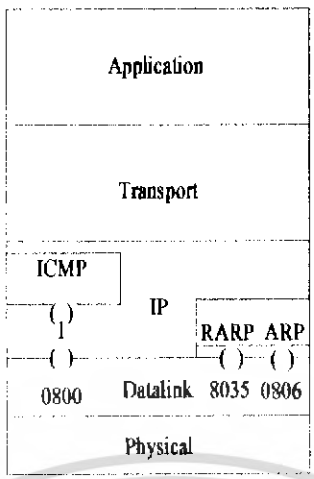
โพรโตคอล IP ทำหน้าที่ให้บริการส่งผ่านข้อมูลที่มาจาก Host – to – Host Layer เพื่อส่งข้ามไปยังเครือข่ายใด ๆ ได้อย่างถูกต้อง แม้ว่าจะมีเครือข่ายเชื่อมต่อกันอยู่ในอินเตอร์เน็ตเป็นล้าน ๆ เครือข่ายก็ตาม เนื่องจากโพรโตคอล IP มีข้อมูลตำแหน่ง IP ปลายทางที่จะส่งข้อมูลไปให้โดยทำงานร่วมกับอุปกรณ์ Router เพื่อส่งข้อมูลข้ามเครือข่ายออกไปได้ ตัวโพรโตคอล IP จะทำงานแบบ Packet Switching คือมีการส่งข้อมูลผ่านสวิทช์ (Switch) ไปยังปลายทาง โดยข้อมูลจะเดินทางไปยังเครือข่ายต่าง ๆ ผ่านสวิทช์นี้ไปเรื่อยๆ จนกว่าจะถึงปลายทาง ตัววงจรผ่านหรือสวิทช์นี้อาจเป็น Gateway หรือ Router ในระบบเครือข่ายก็ได้ ซึ่งในข้อมูลของโพรโตคอล IP จะมีข้อมูลของหมายเลข IP ที่จะส่งข้อมูลไปและเมื่อถึงเครือข่ายปลายทางแล้ว จะมีกลไกแปลงหมายเลข IPให้เป็นหมายเลขฮาร์ดแวร์ประจำเครื่องที่ถูกต้องอีกทีหนึ่งด้วยโพรโตคอล ARP

IP จะให้บริการชนิด Connectionless สำหรับ User ดังนั้น Data ที่ถูกส่งผ่าน IP โดยกระบวนการ Send ยังไม่แน่ว่าจะสามารถส่งถึง ข้อมูลของ IP หน่วยต่างๆที่ถูกส่ง เราเรียกว่า IP Datagram ซึ่งผ่านการ Encapsulation ข้อมูลที่ถูกส่งมาจาก Layer ที่สูงกว่าด้วย IP Header ถ้า IP Datagram ไม่ได้ถูกนำส่งภายในเวลาที่กำหนด (Time – to –Live) Datagram นั้นก็จะไม่ถูกส่งอีกเลย สำหรับช่วงเวลา Time – to –Live นั้นสามารถกำหนดไว้ในกระบวนการส่ง

2.2.3.2 ส่วนประกอบของ IP

IP Address จะต้องมีค่าเฉพาะเจาะจงไม่เหมือนใคร เพื่อที่จะใช้เชื่อมต่อเข้ากับเครือข่ายที่หมายเลขของเครือข่ายที่เฉพาะเจาะจงเช่นกัน โดยในรูปที่ 2.29 แสดงให้เห็นว่าใน IP Layer จะประกอบไปด้วย 3 โปรโตคอล ได้แก่

1. The Address Resolution Protocol (ARP)
2. The Reverse Address Resolution Protocol (RARP)
3. The Internet Control Message Protocol (ICMP)



รูปที่ 2.29 ส่วนประกอบของ IP

ARP และ RARP จะอยู่ที่ส่วนปลายสุดของ IP Layer เพราะทั้งสองโปรโตคอลไม่ใช้ IP และจะถูกแยกโปรโตคอลโดย Data link Layer ที่สนับสนุนตัวมันอยู่ ส่วน ICMP จะอยู่ในส่วนบนสุดของ IP Layer ซึ่งจะข้ามเข้าไปในเครือข่ายใน IP Datagram

2.2.3.3 IP Datagram

Datagram เป็น Basic Unit ของข้อมูลที่ IP จะทำการส่ง ซึ่งถูกออกแบบมาให้ทำงานกับเครือข่ายแบบ Packet Switch ซึ่งข้อมูลของผู้ใช้มักถูกแบ่งออกเป็นหลาย Datagram โดยแต่ละตัวจะมี Header ที่เก็บรายละเอียดเกี่ยวกับตัวมันเอง และปลายทางที่มันจะไป Datagram แต่ละตัวจะเดินทางโดยไม่เกี่ยวข้องกัน นั่นคือ Datagram แต่ละตัวอาจจะเดินทางไปยังปลายทางโดยใช้เส้นทางคนละเส้น และลำดับที่ของการไปถึงจุดหมายปลายทางก็ไม่แน่นอน เป็นหน้าที่ของ Internet Protocol ในเครื่องปลายทางที่จะต้องประกอบ Datagram เหล่านี้ให้กลายเป็นข้อมูลของผู้ใช้ที่สมบูรณ์อีกครั้ง

IP Datagram ประกอบด้วย IP Header และ IP Payload

2.2.3.3a <u>IP Header</u> เป็นขนาดที่เปลี่ยนแปลงได้ระหว่าง 20 และ 60 ไบต์ ในการเพิ่มขึ้น 4 ไบต์ มันจะจัดเตรียมการสนับสนุน Routing , การแสดงตัว Payload , การชี้ให้เห็นถึงขนาด IP Header และ Datagram , การสนับสนุน Fragmentation โดยมีโครงสร้างดังรูปที่ 2.30

Version	IP Header Length	Type of Service	Total Length	Identifier	Flags	Fragment Offset
4 bit	4 bit	8 bit	16 bit	16 bit	3 bit	13 bit

Time – to – Live	Protocol	Header Checksum	Source IP Address	Destination IP Address	IP Option and Padding
8 bit	8 bit	16 bit	32 bit	32 bit	32 bit

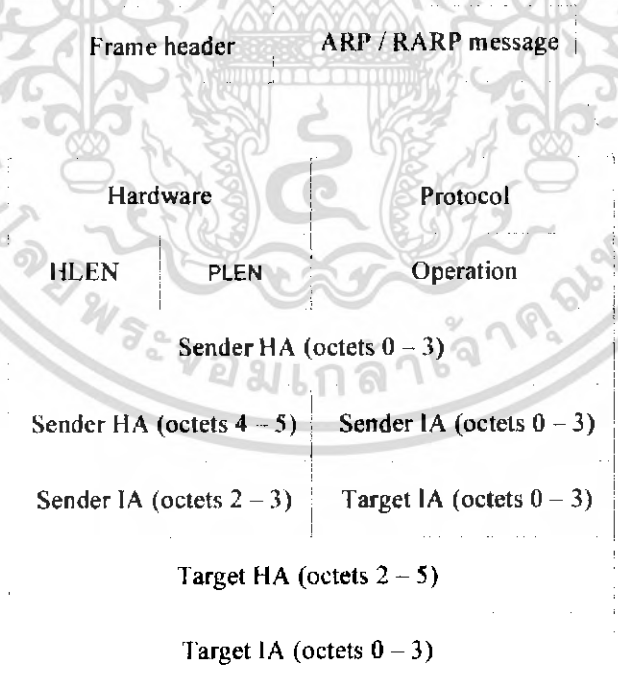
รูปที่ 2.30 แสดงโครงสร้าง IP Header

2.2.3.3b IP Payload เป็นขนาดที่เปลี่ยนแปลงโดยมีลำดับจาก 8 ไบต์ (68 ไบต์ IP Datagram กับ 60 ไบต์ IP Header) ถึง 65,515 ไบต์ (65,535 ไบต์ IP Datagram กับ 20 ไบต์ IP Header)

2.2.3.4 Address Resolution Protocol (ARP)

โพรโตคอล ARP (Address Resolution Protocol) ถูกเรียกใช้งานโดยโพรโตคอล IP เพื่อช่วยแปลงหมายเลข IP ไปเป็นหมายเลขฮาร์ดแวร์ปลายทาง ตัวอย่างเช่น เว็บเซิร์ฟเวอร์เครื่องหนึ่งเชื่อมต่ออยู่ในเครือข่ายอินเตอร์เน็ต และในการเชื่อมต่อนี้ต้องอาศัย Network Interface Card (NIC) หรือ LAN Card ติดตั้งอยู่ ที่LAN Card นี้เองจะมีหมายเลขเฉพาะประจำฮาร์ดแวร์ที่ไม่ซ้ำกับใคร เพื่อใช้อ้างอิงการส่งข้อมูลในเครือข่าย แต่เมื่อมาใช้งานในโพรโตคอล TCP/IP ก็จะต้องมีการกำหนดหมายเลข IP Address ประจำตัวเพื่อใช้อ้างอิงกัน และโพรโตคอล ARP จะทำหน้าที่แปลงค่าหมายเลข IP ให้เป็นหมายเลขฮาร์ดแวร์จริงให้ในระดับการทำงานที่ Internetwork Layer นี้ ซึ่งกลไกการแปลงนี้เรียกว่า Address Resolution

2.2.3.4a ARP Datagram



รูปที่ 2.31 แสดง ARP Datagram

ในรูปที่ 2.31 แสดง ARP Datagram โดยที่มันไม่สามารถนำมาใช้ได้ทั้งหมดสำหรับ TCP/IP หรือเครือข่ายใดเครือข่ายหนึ่งโดยเฉพาะ โดยมันแค่ถูกกำหนดให้เป็นตัวกลางที่มีความสามารถทำการส่งเฟรม Broadcast กระบวนการของ ARP จะดำเนินการโดยตรงเข้าไปอยู่เหนือ Datalink Layer และจะทำการ Encapsulate ในเฟรม Datalink โดย ARP Datagram จะประกอบไปด้วย

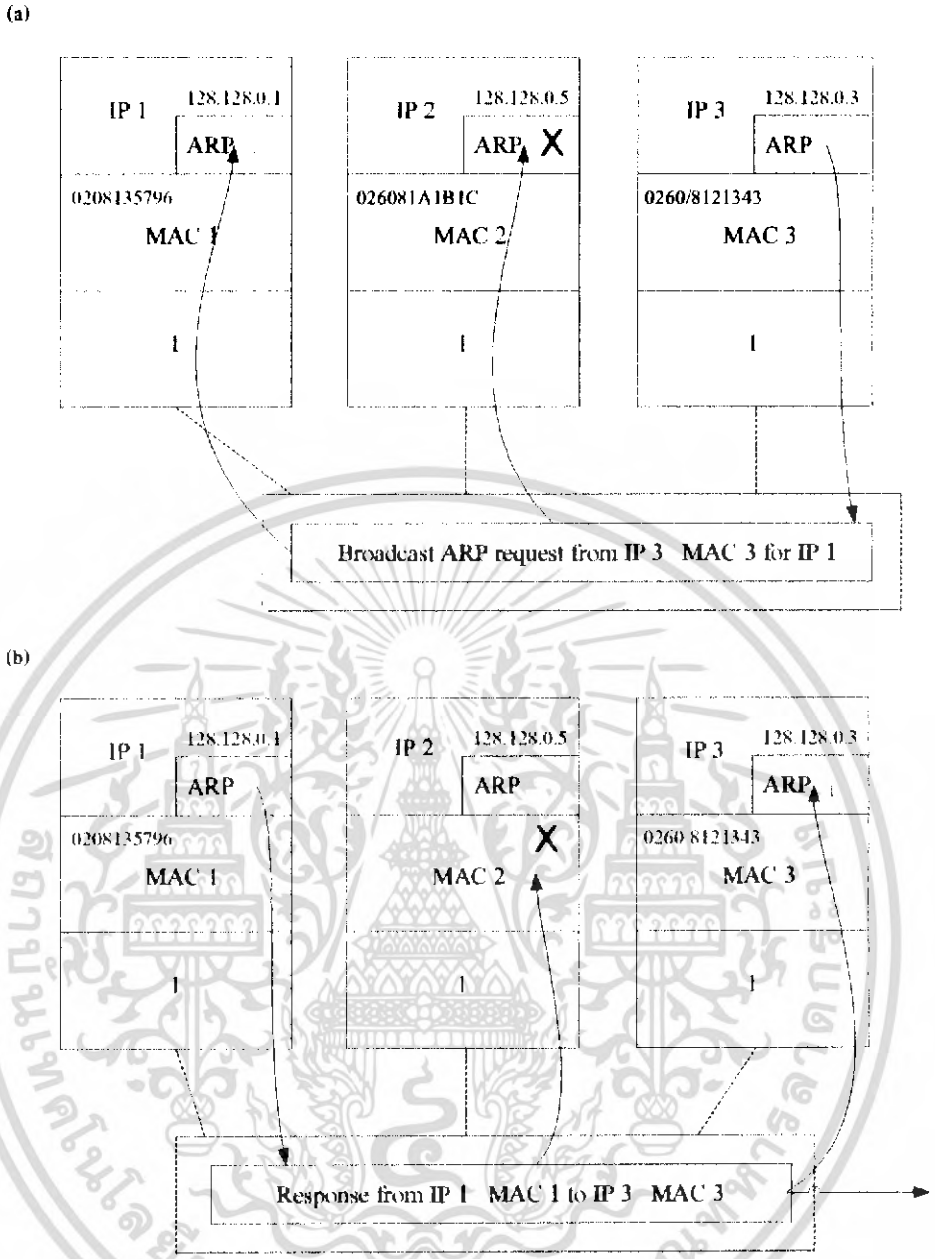
2.2.3.4b กระบวนการทำงานของ ARP

การติดต่อสื่อสารระหว่างคอมพิวเตอร์บนเครือข่ายนั้นจะต้องอาศัยค่า MAC Address เป็นสำคัญ ฉะนั้นหากว่าเครื่องส่งไม่ทราบค่า MAC Address ของเครื่องรับแล้ว การส่งข้อมูลก็ไม่สามารถเกิดขึ้นได้ ดังนั้นจำเป็นที่ต้องมีกลไกที่ใช้ค้นหาค่า MAC Address ของเครื่องรับ ในแบบจำลอง TCP/IP ได้มีการเลือกใช้ ARP ซึ่งเป็นกลไกที่อนุญาตให้ IP Protocol ค้นหาค่า MAC Address ที่สัมพันธ์กับ IP Address

การทำงานของ ARP ประกอบไปด้วยกระบวนการต่อไปนี้

1. เครื่อง Client จะใช้ค่าของ IP Address เพื่อค้นหา MAC Address ที่สอดคล้องกัน ซึ่งเก็บไว้ใน ARP Cache
2. ถ้าไม่พบค่า MAC Address สำหรับ IP Address ดังกล่าวใน ARP Cache แล้วเครื่อง Client ก็จะทำการส่ง ARP Request Packet ด้วยวิธีการ Broadcast ในระดับชั้นย่อย MAC ให้กับทุกๆ Host ใน LAN ทราบ
3. ถ้าหากมี Host ที่มีค่า IP Address สอดคล้องกับข้อมูล ARP Request Packet ดังกล่าวมันจะทำการส่ง ARP Reply Packet แบบ Unicast ในระดับชั้นย่อย MAC ซึ่งภายในมีข้อมูลของ MAC Address และ IP Address ของ Host ดังกล่าวกับไปที่เครื่อง Client
4. เครื่อง Client จะบันทึกข้อมูลค่า MAC Address และ IP Address ดังกล่าวลงบน ARP Cache

ในรูปที่ 2.32 แสดงตัวอย่างการการดำเนินการของ ARP โดย IP Address 128.128.0.3 และค่า MAC Address 0x0268121343 จะทำการร้องขอโดยตัวมัน IP Layer ก็จะค้นหาค่า MAC Address กับ IP Address 128.128.0.1 โดยเริ่มแรก 128.128.0.3 จะส่ง ARP Request ที่จะถูกรับโดย ARP Software บนทุกๆจุดในเครือข่าย 128.128.0.1 จะยอมรับ Address ของมันในการร้องขอ และจะส่ง ARP Reply กลับไป แต่จะใช้ค่า MAC Address ที่ ARP Request ส่งมา ARP Reply จะถูกดำเนินการ และถูกละทิ้งไปอย่างรวดเร็วโดย Card LAN ที่ทุกๆจุดบนเครือข่าย ขณะที่มันเป็น Unicast Frame กับ Destination Address ที่ผิด



รูปที่ 2.32 ตัวอย่างกระบวนการของ ARP, (a) ARP Request, (b) ARP Response

2.2.3.5 Reverse Address Resolution Protocol (RARP)

วิธีการ ARP ช่วยแก้ปัญหาในการค้นหาที่อยู่ของข้อมูลที่ใช้การกำหนดที่อยู่แบบฮาร์ดแวร์ แต่ถ้าทราบที่อยู่แบบฮาร์ดแวร์แล้วต้องการแปลงที่อยู่เป็น IP จะทำอย่างไร ปัญหานี้มักเกิดขึ้นกับเครื่อง Computer ที่เริ่มทำงานด้วยการอ่านข้อมูลทั้งหมดจากเครื่อง Host เครื่องประเภทนี้จะทราบเพียงที่อยู่ ของตนเองจากอุปกรณ์สื่อสารเครือข่ายเท่านั้น

การค้นหาคำตอบสามารถทำได้โดยวิธีควบคุมการสื่อสารแบบ ARP ย้อนกลับ หรือ RARP (Reverse Address Resolution Protocol) วิธีการนี้ Computer ที่เพิ่งจะเริ่มทำงาน (หรือเครื่องใดก็แล้วแต่) จะส่งคำถามออกไปในทำนอง "ที่อยู่ขนาด 48 บิตแบบฮาร์ดแวร์ของฉันคือ 14.04.05.18.01.25 มีใครทราบที่อยู่ IP ของฉัน

บ้าง" เครื่องที่ให้บริการ RARP จะตรวจดูข้อมูลในตารางข้อมูลของตนเองแล้วจึงส่งหมายเลข IP กลับไปให้ วิธีการนี้ช่วยให้เกิดความอ่อนตัวและเพิ่มประสิทธิภาพในการใช้หมายเลข IP เนื่องจากผู้ใช้ไม่มีหมายเลข IP เป็นของตนเอง ผู้ควบคุมระบบสามารถกำหนดหมายเลข IP ใดๆที่ไม่มีผู้ใช้งานในขณะนั้นให้ใช้ได้ หมายเลข IP ในที่นี้จึงเป็นเสมือนสมบัติส่วนกลางที่ทุกคนใช้ร่วมกัน

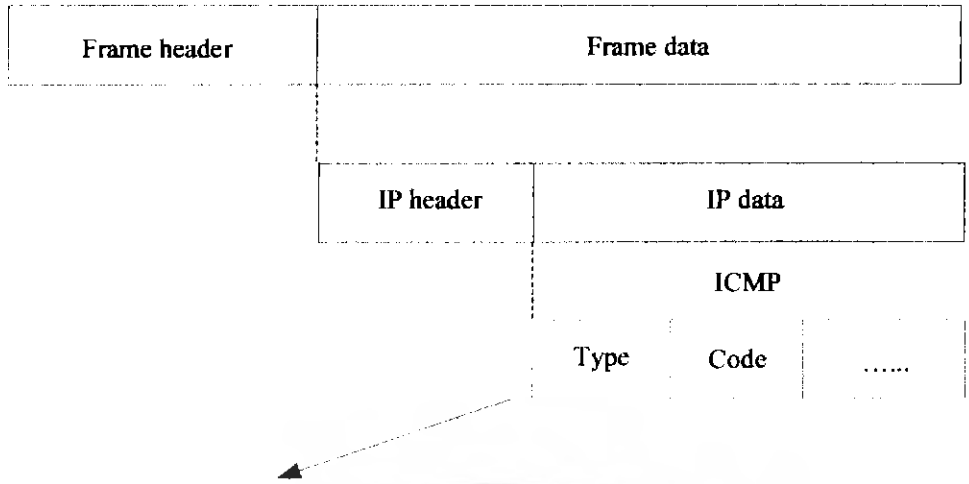
2.2.3.6 Internet Control Message Protocol (ICMP)

ถึงแม้ว่า IP จะเป็น Datagram Service และไม่มีการรับประกันรูปแบบการส่ง โดย Internet Control Message Protocol (ICMP) จะถูกจัดเตรียมไว้ภายใน IP ทำให้เกิด Error Messages ให้เข้าไปช่วย IP Layer ให้มีความสามารถในการส่งที่ดีที่สุด โดยหน้าที่หลักของโปรโตคอล ICMP (Internet Control Message Protocol) คือ การแจ้งหรือแสดงข้อความจากระบบ เพื่อบอกให้ผู้ใช้ ทราบว่า เกิดอะไรขึ้นในการส่งผ่านข้อมูลนั้น ซึ่งปัญหาส่วนมากที่พบคือ ส่งไปไม่ได้ หรือปลายทางรับข้อมูลไม่ได้ เป็นต้น นอกจากนี้ โปรโตคอล ICMP ยังถูกเรียกใช้งานจากเครื่อง Server และ Router อีกด้วย เพื่อแลกเปลี่ยนข้อมูลที่ใช้ควบคุม ส่วนรูปแบบการทำงานของโปรโตคอล ICMP นั้นจะทำควบคู่กับโปรโตคอล IP ในระบบเดียวกัน และข้อความต่าง ๆ ที่แจ้งให้ทราบจะถูกผนึกอยู่ภายในข้อมูล IP (IP datagram) อีกทีหนึ่ง

รูปที่ 2.33 แสดงรูปแบบพื้นฐานของ ICMP message encapsulated ใน IP Datagram ใน ICMP มีหมายเลข IP โปรโตคอลของตัวมันเอง ดังนั้น IP Layer รู้ว่ามันรับ ICMP แม้ว่า ICMP ใช้ IP Layer ที่เป็นตัวพิจารณาว่า IP ภายในเป็นของใคร เพราะว่ามันไม่สามารถจัดเตรียมให้กับ Layer ที่สูงกว่าได้

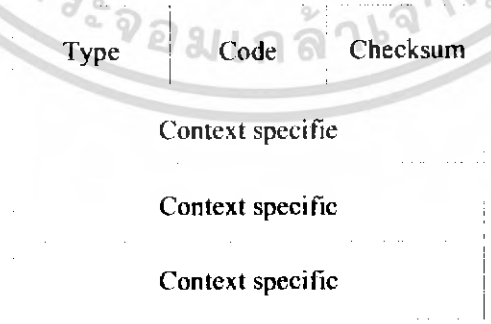
นับตั้งแต่ IP Message ที่ถูกขนย้ายใน IP มันจะถูกทิ้งไปเหมือนกับ IP Datagram โดยมันจะไม่สามารถรักษาสถานภาพไว้ได้ ICMP Message จะไม่ถูกทำให้เกิดขึ้นในกรณีที่ ICMP Message เกิดความผิดพลาดเกิดขึ้น

รูปแบบพื้นฐานของ ICMP Datagram แสดงไว้ในรูปที่ 2.34 โดยจะประกอบไปด้วยการแบ่งประเภทของ ICMP Message และ Code ที่ต้องจัดเตรียมรายละเอียด Checksum ต้องถูกนำมาใช้ เพราะ IP ไม่สามารถที่จะป้องกันข้อมูลของมันได้ เมื่อทำการดำเนินการมาอยู่เหนือ Physical Network ที่มี Frame Check Sequence ICMP Checksum ต้องเป็น 0 นั่นหมายความว่าจะไม่มีการคำนวณเกิดขึ้น



Type field	Message type
0	Echo reply
3	Destination unreachable
4	Source quench
5	Redirect (change route)
8	Echo request
11	Time exceeded for datagram
12	Parameter problem on datagram
13	Time stamp request
14	Time stamp reply
15	Information request
16	Information reply
17	Address mask request
18	Address mask response

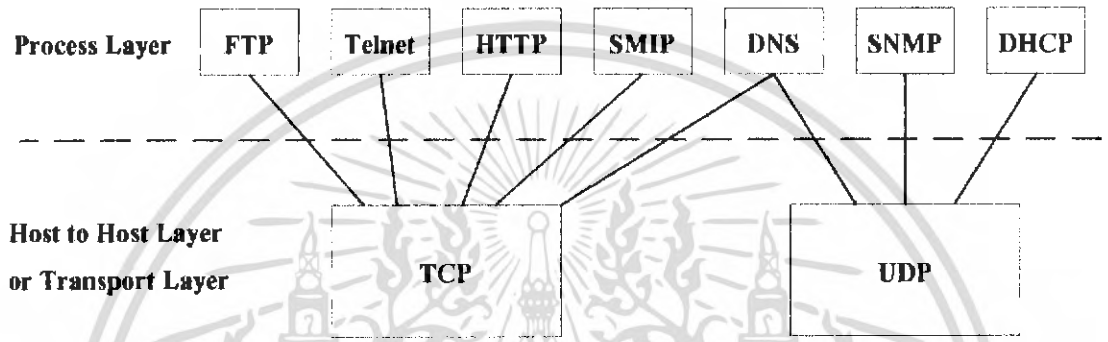
รูปที่ 2.33 ICMP encapsulated ใน IP และประเภทของ ICMP



รูปที่ 2.34 รูปแบบของ ICMP Datagram

2.2.4 Host-To-Host Layer

การทำงานที่ชั้นของ Host-to- Host Layer นี้จะมีบทบาทในการจัดการต่อจาก Process Layer บางครั้งเรามักเรียกชั้น Host-to-Host ว่าเป็น Transport Layer ซึ่งไม่ใช่ชั้นของ Transport Layer ในมาตรฐาน OSI Model การทำงานของ Host-to-Host Layer นี้จะมีการสร้าง Connection หรือการเชื่อมต่อกันระหว่างแอพพลิเคชันกับ Host – to – Host Layer โดยจุดที่เชื่อมกันเพื่อรับส่งข้อมูลที่เรียกว่า พอร์ต หรือ Socket (คำว่าพอร์ตในที่นี้ไม่ได้หมายถึง พอร์ตทางฮาร์ดแวร์) และในแต่ละแอพพลิเคชันก็จะสร้างการเชื่อมต่อผ่านพอร์ตได้พร้อมกันหลายแอพพลิเคชัน ซึ่งการใช้งานพอร์ตของแต่ละแอพลิเคชันที่อยู่ในชั้น Process Layer จะแตกต่างกันตามหมายเลขที่กำหนดไว้ และแต่ละโพรโตคอลจะมีการใช้งาน port หมายเลขต่าง ๆ ไม่ซ้ำกัน ดังรูปที่ 2.35

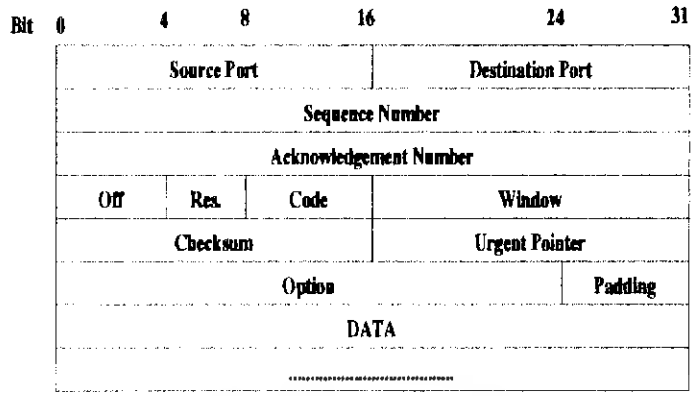


รูปที่ 2.35 แสดงการใช้งานพอร์ตของแต่ละโพรโตคอล

ในชั้น Host-to-Host หรือ Transport Layer ของ TCP/IP นี้จะมีโพรโตคอลทำงานอยู่ 2 โพรโตคอลที่แตกต่างกัน คือ โพรโตคอล TCP และโพรโตคอล UDP (User Datagram Protocol) ในการส่งผ่านข้อมูลลงไปที่ชั้นถัดๆ ไป เราจะเห็นว่าโพรโตคอล TCP และ UDP จะถูกผนึกเข้าไปในโพรโตคอล IP อีกทีหนึ่งและส่งต่อไปยังเครือข่ายอินเตอร์เน็ตต่อไป

2.2.4.1 โพรโตคอล TCP

โพรโตคอล TCP (Transmission Control Protocol) เป็นโพรโตคอลที่มีการรับส่งข้อมูลแบบ stream oriented protocol หมายความว่า การรับส่งข้อมูลจะไม่คำนึงถึงปริมาณข้อมูลที่จะส่งไป แต่จะแบ่งข้อมูลเป็นส่วนย่อย ๆ ก่อน แล้วจึงส่งไปยังปลายทางอย่างต่อเนื่องเป็นลำดับข้อมูล ในกรณีที่ข้อมูลส่วนใดส่วนหนึ่งสูญหายไป ก็จะส่งข้อมูลส่วนนั้นใหม่อีกครั้ง สำหรับปลายทางก็จะทำหน้าที่จัดเรียงส่วนของข้อมูล datagram ดังนั้นแอพพลิเคชันหรือโปรเซ็สใดที่อาศัยการส่งผ่านข้อมูลด้วยโพรโตคอล TCP จะต้องใช้หน่วยความจำและขนาดของช่องสัญญาณ (bandwidth) มากกว่า UDP



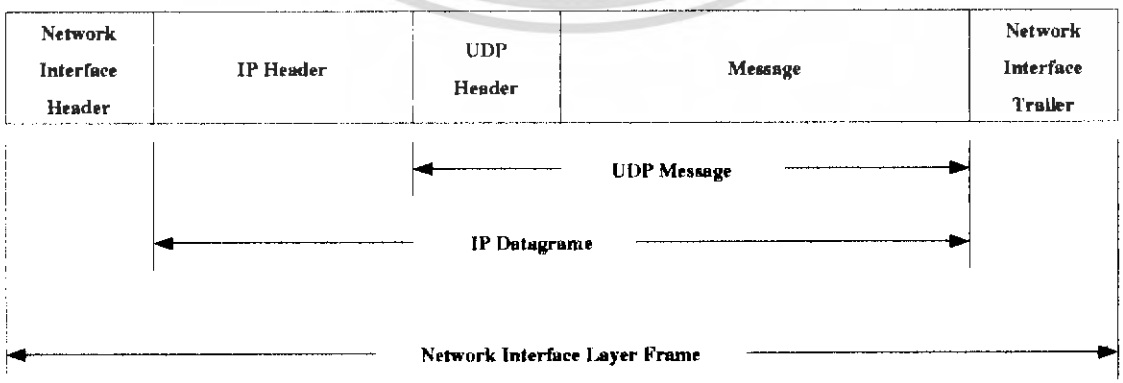
รูปที่ 2.36 แสดงโครงสร้างของโปรโตคอล TCP

2.2.4.2 โพรโตคอล UDP

ใน Host-to-Host Layer นอกจากจะมีโพรโตคอล TCP ทำงานแล้ว ก็ยังมีโพรโตคอล UDP (User Datagram Protocol) ในการส่งข้อมูลแต่ละครั้งและไม่มีการส่งข้อมูลใหม่อีกในกรณีที่เกิดความผิดพลาดของการส่งข้อมูล เมื่อเป็นเช่นนี้แอพพลิเคชันหรือโปรเซ็สใดที่ต้องอาศัยโพรโตคอล UDP จะเป็นแบบที่ทั้งสองด้านไม่จำเป็นต้องอาศัยการสร้างช่องทางเชื่อมต่อกัน (Connectionless) ระหว่างเครื่องเซิร์ฟเวอร์ให้บริการกับเครื่องที่ขอใช้บริการ โดยไม่ต้องแจ้งให้ฝ่ายรับข้อมูลเตรียมรับข้อมูลเหมือนโพรโตคอล TCP และไม่มีการตรวจสอบความถูกต้องครบถ้วนในการรับส่งข้อมูลนั้นๆ ด้วย เนื่องจากโพรโตคอล UDP ไม่มีสัญญาณสอบทานข้อมูล (acknowledgement) ในการส่งข้อมูลแต่ละครั้ง และไม่มีการส่งข้อมูลใหม่อีกในกรณีที่เกิดความผิดพลาดของการส่งข้อมูล เมื่อเป็นเช่นนี้แอพพลิเคชันหรือโปรเซ็สใดที่ต้องอาศัยโพรโตคอล UDP ในการส่งผ่านข้อมูลก็อาจจะต้องสร้างขบวนการตรวจสอบข้อมูลขึ้นมาเอง

UDP Message

UDP Message ประกอบด้วย UDP Header และ Message โดยจะนำไปรวมกับ IP Header เพื่อเข้าสู่ IP Datagram ซึ่งใช้ IP Protocol หมายเลข 17 (0x11) โดย message สามารถมีขนาดสูงสุดได้ถึง 65,535 ไบต์ และมีขนาดเล็กที่สุดเท่ากับ 65,507 ไบต์ โดยเล็กกว่า IP Header (20 ไบต์) และ UDP Header (8 ไบต์) IP Datagram ที่ได้มันจะเป็น encapsulated กับ Network Interface Layer ที่เหมาะสม



รูปที่ 2.37 UDP Messagc Encapsulation

UDP Header

มีขนาด 8 ไบต์ โดยประกอบด้วย 4 ส่วนดังรูปที่ 2.38

Source Port	Destination Port	Length	Checksum
2 ไบต์	2 ไบต์	2 ไบต์	2 ไบต์

รูปที่ 2.38 แสดงโครงสร้างของ UDP Header

- **Source Port** มี 2 ไบต์ใช้ระบุเป็น Source Application Layer Protocol ทำการส่ง UDP Message โดย Source Port เป็นพอร์ตที่ใช้ในการเลือก เมื่อใดที่ไม่ได้ใช้มัน มันจะตั้งค่าเป็น 0x00-00 IP Multicast Traffic เปรียบเสมือน Video casts ใช้ส่ง UDP สามารถใช้ค่า 0x00-00 เพราะจะไม่ตอบรับ Video Traffic เป็นเพียงการสมมุติ Application Layer ใช้ Source Port ในการนำ UDP Message เข้ามา Destination Port สำหรับการตอบรับ

- **Destination Port** มี 2 ไบต์ใช้ระบุเป็น Destination Application Layer Protocol การรวมของ Destination IP Address ของ IP Header และ Destination Port ของ UDP Header จะไม่เหมือนใครสำหรับกระบวนการที่จะส่งข้อมูล

- **Length** มี 2 ไบต์ที่ใช้ในการแสดงความยาวใน UDP Message มีความยาวน้อยที่สุด 8 ไบต์ (ขนาดของUDP Header) และมากที่สุด 65,515 ไบต์ (ค่าสูงสุด IP Datagram 65,535 ไบต์ น้อยกว่าค่าน้อยที่สุด IP Header 20 ไบต์) ความยาวมากที่สุดที่แท้จริงถูกจำกัดโดย MTU ซึ่งจะทำการเชื่อมโยงโดย UDP Message เป็นตัวส่ง ความยาว UDP สามารถคำนวณได้จากความยาวทั้งหมดและความยาวของ IP Header Field ใน IP Header

- **Checksum** มี 2 ไบต์ โดยจะทำการตรวจระดับของบิตอย่างสมบูรณ์สำหรับ UDP Message โดยที่ UDP Checksum คำนวณโดยใช้วิธีเดียวกันกับ IP Header Checksum

UDP Checksum

Checksum เป็น เลข 16 บิตถูกคำนวณด้วยวิธี 1's Complement โดยนำ Pseudo Header และข้อมูลทั้งหมดใน UDP Datagram มาคำนวณ

Pseudo Header เป็นข้อมูลที่อยู่ในส่วนของ IP Header ประกอบด้วยฟิลด์ Source IP Address, Destination IP Address, Zero, Protocol, UDP Length ดังแสดงในรูปที่ 2.39

16-bit Source IP address		
16-bit Destination IP address		
zero	8-bit protocol (17 for UDP)	16-bit length

รูปที่ 2.39 Pseudo Header

หากค่า Checksum ที่คำนวณออกมาเป็น 0 ค่า checksum จะถูกเซตเป็น 1 ทั้งหมดแทน (มีค่าเท่ากันในระบบ 1's complement) ทั้งนี้เพราะในบางแอพพลิเคชั่นที่ไม่ต้องการตรวจสอบค่า Checksum ในระดับ UDP จะเซตค่านี้เป็น 0 (Disable Checksum)

2.2.5 Process Layer

การแสดงลำดับชั้นการทำงานของโพรโตคอล TCP/IP เทียบกับมาตรฐาน OSI model นั้น ในชั้นบนสุดเรียกว่า Process Layer ทำงาน 2 หน้าที่เทียบได้กับ Application Layer และ Presentation Layer ในชั้นนี้จะรองรับการทำงานของแอพพลิชันต่าง ๆ ที่ทำงานเป็นโปรเซ็ส อยู่ในเครื่องเซิร์ฟเวอร์ที่ให้บริการและเครื่องที่ขอใช้บริการ หรือไคลเอนต์ (Client) ซึ่งจะติดต่อกันผ่านโพรโตคอลเฉพาะแอพพลิเคชันอีกทีหนึ่ง ตัวอย่างเช่น เมื่อผู้ใช้งานอินเตอร์เน็ตต้องการโอนถ่ายไฟล์หรือ Download ข้อมูลจากเครื่องเซิร์ฟเวอร์ที่ให้บริการ โดยอาจจะเรียกใช้โปรแกรม FTP Client ทั่วไป เช่น โปรแกรม WS_FTP ติดต่อกับโปรเซ็ส FTP ที่กำลังให้บริการอยู่ที่เครื่องเซิร์ฟเวอร์ จากนั้นตัวโปรเซ็ส FTP ก็จะเรียกใช้โพรโตคอล FTP (File Transfer Protocol) เพื่อทำการโอนถ่ายไฟล์นี้ หรือถ้าผู้ใช้ต้องการเรียกใช้งานคอมพิวเตอร์ที่อยู่ห่างไกลออกไปด้วยการใช้โปรแกรม Telnet ที่เครื่องเซิร์ฟเวอร์ให้บริการ ตัวโปรเซ็ส Telnet ที่ทำงานอยู่ก็จะเรียกใช้โพรโตคอล Telnet เพื่อติดต่อกัน หรือในกรณีที่มีการเรียกใช้โปรแกรม Web Browser เช่น Netscape Navigator เพื่อเรียกดูเว็บไซท์ CNN ที่เครื่องซึ่งให้บริการเว็บของ CNN ก็จะมีโปรเซ็ส HTTP (Hypertext Transfer Protocol) ทำงานอยู่และจะติดต่อกับผู้ใช้ผ่านโพรโตคอล HTTP เป็นต้น

การทำงานของแอพพลิเคชันต่าง ๆ จะอยู่ที่ Process Layer นี้ และมีการติดต่อกันตามแต่ละโพรโตคอลเฉพาะแล้วแต่แอพพลิเคชันที่ใช้งาน จากการที่ Process Layer ของ TCP/IP รองรับให้โพรโตคอลอื่นทำงานได้หลายโปรเซ็สและหลายโพรโตคอลได้พร้อมกันนั้น ทำให้ผู้ใช้สามารถเปิดโปรแกรมใช้งานได้หลาย ๆ โปรแกรมพร้อมกัน เช่น เปิดโปรแกรม Internet Explorer เพื่อเรียกดูเว็บเพจ พร้อมกับใช้งานโปรแกรม Outlook Express เพื่อรับส่งอีเมล์ไปพร้อมกันได้โดยไม่ต้องรอให้ทำงานอย่างหนึ่งอย่างใดเสร็จก่อน หรือในปัจจุบันมีการพัฒนาโปรแกรม Web Browser ให้สามารถเรียกใช้งานโพรโตคอลอื่น ๆ ได้มากขึ้น ทำให้เราสามารถใช้โปรแกรม Web Browser โอนถ่ายไฟล์ข้อมูลที่ใช้โพรโตคอล FTP ได้โดยไม่ต้องไปหาโปรแกรมอื่นมาใช้

โพรโตคอลหลัก ๆ ที่ทำงานใน Process Layer ซึ่งผู้ใช้มักจะคุ้นเคยกันดีได้แก่ FTP (File Transfer Protocol), Telnet, HTTP(Hyper Text Transfer Protocol), SMT(Simple Mail Transfer protocol) นอกจากนี้ยังมีโพรโตคอลอื่นที่อยู่เบื้องหลัง ซึ่งทำงานโดยที่ผู้ใช้ไม่ได้มีการใช้งานโดยตรง เช่น

- โพรโตคอล DNS (Domain Name System) ที่ทำหน้าที่แปลงข้อมูลชื่อ Domain Name หรือชื่อเว็บไซท์ทั้งหลายให้เป็นหมายเลข IP address
- โพรโตคอล DHCP (Dynamic Host Configuration Protocol) ทำหน้าที่แจกจ่ายข้อมูลพารามิเตอร์ของเครือข่ายให้กับเครื่องลูกข่ายที่เชื่อมต่ออยู่

บทที่ 3
การคำนวณและการสร้างวงจร

หลังจากที่ทราบถึงทฤษฎีหรือหลักการในบทที่ 2 แล้ว เราก็จะมาทำการคำนวณและทำการสร้างวงจร โดยในโครงงานนี้จะแบ่งส่วนประกอบหลักของระบบออกเป็น 3 ส่วนใหญ่ๆ ด้วยกันคือ

1. อัลกอริธึมในการตรวจสอบลายนิ้วมือ เทียบกับฐานข้อมูลในเครื่องคอมพิวเตอร์ระบบศูนย์กลาง
2. ฮาร์ดแวร์ของระบบที่ใช้ในการรับ - ส่งข้อมูล ผ่านวงแลน โดยมีซอฟท์แวร์รอรับข้อมูลที่เครื่องคอมพิวเตอร์ระบบศูนย์กลาง เพื่อทำการประมวลผลแล้วส่งผลการตรวจสอบลายนิ้วมือกลับมายังฮาร์ดแวร์
3. ฮาร์ดแวร์ที่ใช้ในการสแกนลายนิ้วมือ แล้วส่งข้อมูลต่อให้กับฮาร์ดแวร์ที่ใช้ในการรับ - ส่งข้อมูลผ่านวงแลน

3.1 อัลกอริธึมในการตรวจสอบลายนิ้วมือ

ในส่วนของอัลกอริธึมตรวจสอบลายนิ้วมือจะใช้โปรแกรมที่ประมวลบนเครื่องคอมพิวเตอร์ศูนย์กลาง โดยนำภาพลายนิ้วมือมาประมวลผลเพื่อหาลักษณะสำคัญของลายนิ้วมือ ซึ่งแบ่งออกเป็น 2 ส่วนใหญ่คือ การลงทะเบียน (Enrollment) และการระบุบุคคล (Identification)

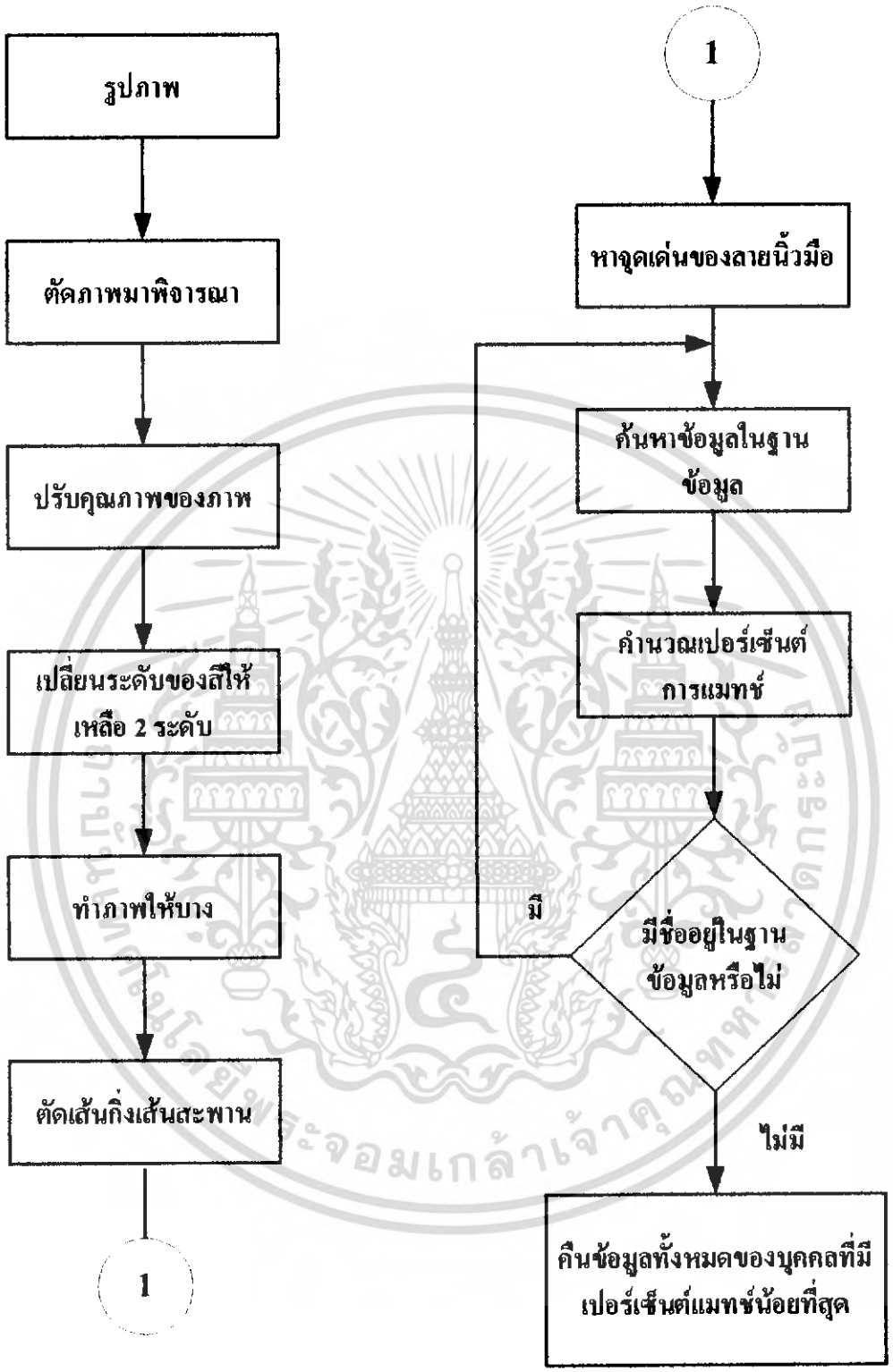
3.1.1 การลงทะเบียน (Enrollment)

การทำการลงทะเบียนดังรูปโฟว์ชาร์ตที่ 3.1 เริ่มต้นด้วยการนำภาพที่ได้จากเครื่องสแกนเนอร์มาตัดเพื่อนำพื้นที่ที่ต้องการมาประมวลผล แล้วทำการปรับคุณภาพของภาพ (Image Enhance) ทำการเปลี่ยนระดับของสีให้เหลือ 2 ระดับ แล้วทำการทำให้ภาพเป็นเส้นบาง (Thinning) และทำกำจัดเส้นกิ่งและเส้นสะพาน (Imperfect Removal) จากนั้นจึงทำการหาจุดเด่นลายนิ้วมือ (Minutia Extraction) จะได้ตำแหน่งและมุมของจุดเด่นลายนิ้วมือ หรือเทมเพลต ไปเก็บในฐานข้อมูลจึงจบกระบวนการลงทะเบียน

3.1.2 การระบุบุคคล (Identification)

การทำการระบุบุคคล ดังโฟว์ชาร์ต ที่ 3.2 ทำการประมวลผลภาพเหมือนกับกระบวนการลงทะเบียน หลังจากได้จุดเด่นลายนิ้วมือ หรือเทมเพลตแล้ว จะนำเทมเพลตมาตรวจสอบกับเทมเพลตกับทุกๆเทมเพลตในฐานข้อมูล ถ้าหากเทมเพลตใดมีค่าเปอร์เซ็นต์ความเหมือนมากที่สุด ให้โหลดข้อมูลที่เกี่ยวข้องกับเทมเพลตนั้น เช่น ชื่อ-นามสกุล หรือ ข้อมูลส่วนตัวที่เก็บในฐานข้อมูล เป็นอันจบกระบวนการระบุบุคคล

รูปที่ 3.1 โฟว์ชาร์ตของการลงทะเบียน (Enrollment)



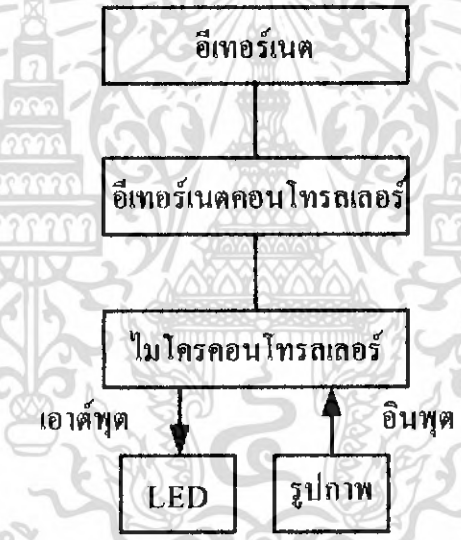
รูปที่ 3.2 โฟว์ชาร์ตของการระบุบุคคล (Identification)

3.2 ฮาร์ดแวร์ของระบบที่ใช้ในการรับ - ส่งข้อมูล ผ่านวงแลน

ในส่วนฮาร์ดแวร์ของระบบควบคุมนั้นจะใช้การควบคุมแบบฝังตัว (Embedded) ซึ่งจะประกอบด้วย วงจรควบคุมการรับ-ส่งข้อมูลผ่านอีเทอร์เนต ไมโครคอนโทรลเลอร์ และหลอดไฟ LED แสดงผลของข้อมูลที่ประมวลผลได้จากเครื่องคอมพิวเตอร์ระบบศูนย์กลาง รายละเอียดการออกแบบฮาร์ดแวร์ที่จะกล่าวถึงในที่นี้จึงประกอบด้วย การเชื่อมต่อฮาร์ดแวร์เข้ากับระบบเครือข่าย โดยอาศัยอุปกรณ์ควบคุมการเชื่อมต่อกับวงแลน (Embedded Ethernet Controller) และอธิบายถึงการควบคุมหน่วยการรับเข้าออกข้อมูล (I/O Port) ซึ่งถูกควบคุมด้วยไมโครคอนโทรลเลอร์ และกระบวนการรับ-ส่งข้อมูลของฮาร์ดแวร์

3.2.1 ส่วนประกอบของฮาร์ดแวร์ในการเชื่อมต่อกับอีเทอร์เนต

ส่วนประกอบหลักของฮาร์ดแวร์ทั้งหมดประกอบด้วย วงจรเชื่อมต่อกับระบบเครือข่าย โดยในส่วนนี้จะเชื่อมต่อกับอีเทอร์เนตและไมโครคอนโทรลเลอร์ โดยที่ไมโครคอนโทรลเลอร์จะไปควบคุมการทำงานจาก I/O พอร์ต ของอีเทอร์เนตคอนโทรลเลอร์ โดยจะทำควบคุมการรับข้อมูลอินพุต (Image) เข้ามา และแสดงผลของข้อมูลที่ประมวลผลได้ออกมายังหลอดไฟ LED ดังรูปที่ 3.3

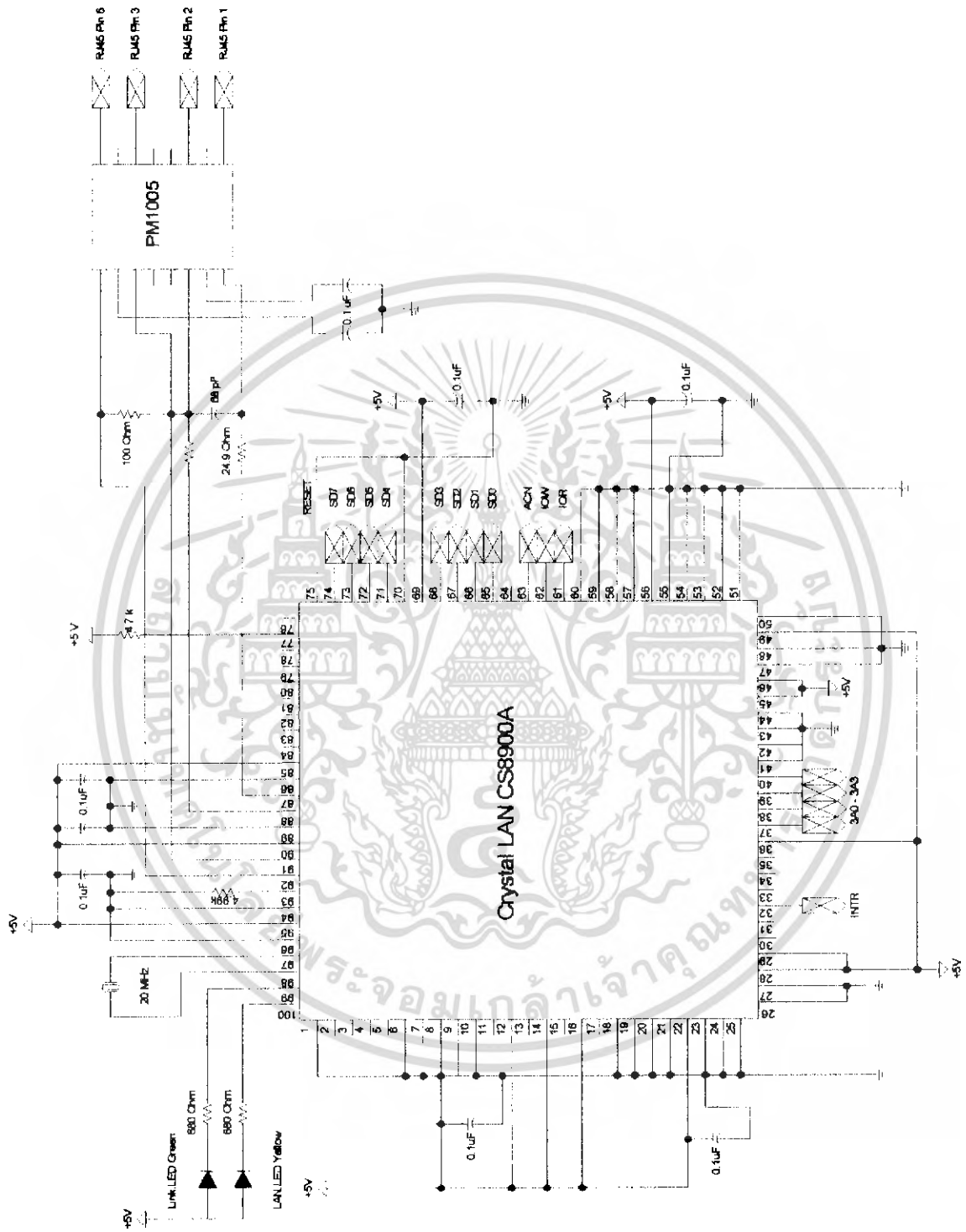


รูปที่ 3.3 แสดงส่วนประกอบหลักของฮาร์ดแวร์ที่ส่งข้อมูลผ่านวงแลน

3.2.2 ส่วนเชื่อมต่อระบบเครือข่าย

จะใช้อุปกรณ์ควบคุมการเชื่อมต่อระบบเครือข่าย (Ethernet Controller) โดยภายในวงจรจะประกอบไปด้วยส่วนประกอบหลักกับชิปควบคุมอีเทอร์เนต ในที่นี้จะทำการเลือกใช้ CS8900A-CQ, Isolator Transformer (PM1005) และ RJ-45 Connector

รูปที่ 3.4 แสดงส่วนเชื่อมต่อระบบเครือข่าย



วงจรควบคุมการเชื่อมต่อระบบเครือข่ายจะมี PIN OUT 18 ขา ดังรูปที่ 3.5

PIN 1	PIN 2	PIN 3	PIN 4	PIN 5	PIN 6	PIN 7	PIN 8	PIN 9
GND	VCC	INTR	SA0	SA1	SA2	SA3	/IOR	/IOW

PIN 10	PIN 11	PIN 12	PIN 13	PIN 14	PIN 15	PIN 16	PIN 17	PIN 18
/AEN	SD0	SD1	SD2	SD3	SD4	SD5	SD6	SD7

รูปที่ 3.5 PIN OUT

VCC - แหล่งจ่ายไฟตรง +5 V

GND - กราวด์อ้างอิง 0 V

INTR - สำหรับใช้งานในโหมดอินเทอร์รัพ

SA0 – SA3 - Address Bus เชื่อมต่อกับไมโครคอนโทรลเลอร์

/IOR - I/O Port Read (Active low)

/IOW - I/O Port Write (Active low)

/AEN - Chip Enable (Active low)

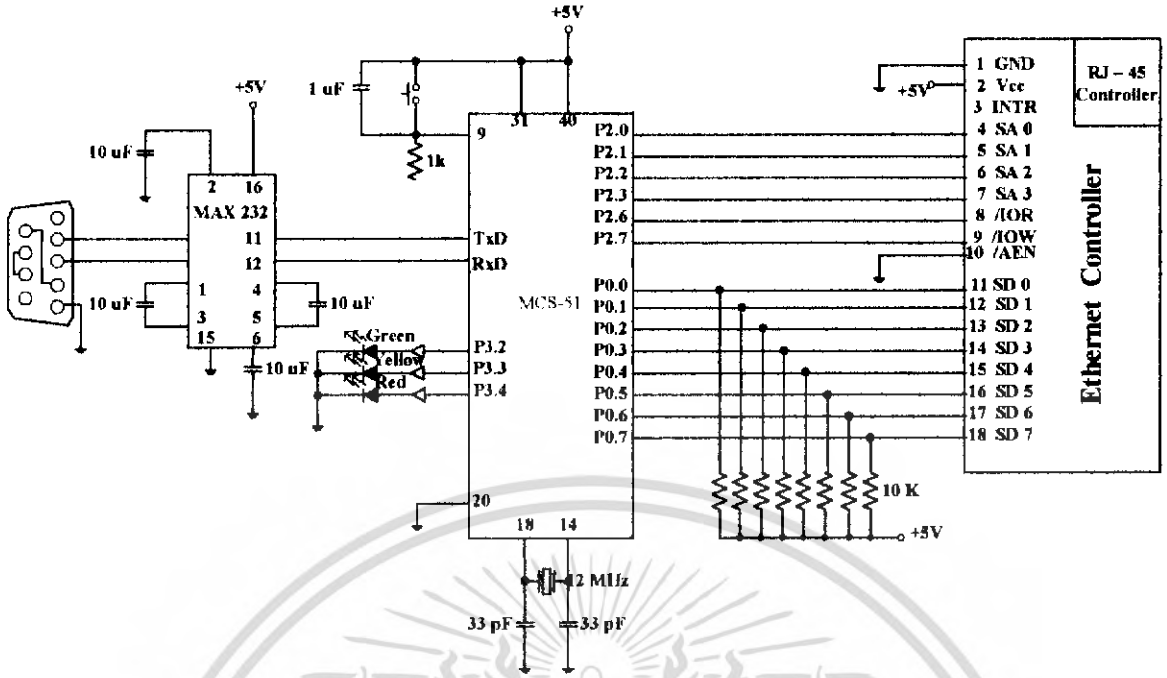
SD0 – SD7 - Data Bus เชื่อมต่อกับไมโครคอนโทรลเลอร์

จาก LED ในวงจรจะสว่างด้วยการจ่ายไฟแล้วเสียบสาย RJ-45 ที่ต่อในวงแลน เมื่อ

- Link LED (Green) จะกระพริบเมื่อมีเฟรมข้อมูลส่งออกจากอีเทอร์เนตคอนโทรลเลอร์
- LAN LED (Yellow) จะกระพริบเมื่อมีเฟรมข้อมูลเข้ามายังอีเทอร์เนตคอนโทรลเลอร์

3.2.3 การติดต่อกับอุปกรณ์ต่างๆ

ในการใช้ไมโครคอนโทรลเลอร์ ในที่นี้จะใช้ MCS-51 เบอร์ AT89C52 เป็นไมโครคอนโทรลเลอร์ที่ส่งข้อมูลทีละ 8 บิต ไปควบคุมอุปกรณ์ควบคุมการเชื่อมต่อระบบเครือข่ายโดย SD0 – SD7 จะเป็นดาต้าบัสเชื่อมต่อกับ P0.0 – P0.7, SA0 – SA3 จะเป็น Address Bus เชื่อมต่อกับ P2.0 – P2.3, /IOW ต่อกับ P2.7, /IOR ต่อกับ P2.6, /AEN จะต่อลงกราวด์, ขา INTR ปล่อยลอย เพราะไม่ใช้อินเทอร์รัพ ส่วนของ P1.0 – P1.3 จะต่อกับสวิตช์ เพื่อใช้ทำการส่งข้อมูลที่แตกต่างกัน 4 ข้อ และ P3.3 ต่อกับหลอดไฟ LED สีเหลือง เมื่อหลอด LED สว่างจะหมายถึงระบบพร้อมที่จะทำงาน P3.2 ต่อกับหลอดไฟ LED สีเขียว เมื่อหลอด LED สว่างจะหมายถึงข้อมูลที่ส่งไปมีค่าตรงกับข้อมูลที่มีอยู่ในฐานข้อมูลที่เครื่องคอมพิวเตอร์ระบบศูนย์กลาง และ P3.4 ต่อกับหลอดไฟ LED สีแดง เมื่อหลอด LED สว่างจะหมายถึงข้อมูลที่ส่งไปไม่ตรงหรือไม่มี เมื่อเปรียบเทียบกับข้อมูลที่มีอยู่ในฐานข้อมูลที่เครื่องคอมพิวเตอร์ระบบศูนย์กลาง โดยในการส่งข้อมูลอินพุต (Image) เราจะทำการส่งผ่านซีเรียลพอร์ตและแสดงผลของข้อมูลที่ประมวลผลได้ออกมายังหลอดไฟ LED ดังรูปที่ 3.6



รูปที่ 3.6 แสดงการติดต่อกับอุปกรณ์ต่างๆ

3.2.4 การเข้าถึงหน่วยความจำของระบบ

ในอุปกรณ์ควบคุมการเชื่อมต่อระบบเครือข่ายจะมีชิปประมวลผลที่สำคัญคือ CS8900A-CQ โดยชิปนี้มีรูปแบบการทำงาน 3 โหมด คือ Memory Mode, I/O Mode และ DMA Mode แต่ในที่นี้จะใช้เพียง I/O Mode เพียงอย่างเดียว ใน I/O Mode จะประกอบด้วย I/O พอร์ต อยู่ 8 พอร์ต แต่ละพอร์ตจะมีความยาว 16 บิต แต่เนื่องจากไมโครคอนโทรลเลอร์ทำงานทีละ 8 บิต จึงจำเป็นต้องส่งข้อมูล 2 รอบ จึงจะเข้าถึง I/O พอร์ต ได้ คำสั่งต่างๆใน I/O Mode จะประกอบด้วย

- **Receive / Transmit Data (พอร์ต 0, พอร์ต 1)**

  ใช้ในการรับส่งข้อมูลจากอีเทอร์เนต ส่วนมากจะใช้พอร์ต 0

- **TxCMD (Transmit Command)**

  ใช้ในการออกคำสั่งให้อุปกรณ์ควบคุมการเชื่อมต่อระบบเครือข่ายเตรียมตัวส่งข้อมูล

- **TxLength (Transmit Length)**

  ใช้ในการระบุความยาวของข้อมูลที่จะส่งเป็นไบต์

- **Interrupt Status Queue**

  ใช้ในการอินเทอร์รัพ

- **PacketPage Pointer**

  ใช้ในการระบุรีจีสเตอร์ภายในของ CS8900A-CQ สามารถหาดูได้จาก Datasheet

- **PacketPage Data (พอร์ต 0, พอร์ต 1)**

  ใช้ในการอ่านหรือเขียนรีจีสเตอร์ภายใน ที่ถูกระบุโดย PacketPage Pointer

ในการระบุคำสั่งต่างๆของ I/O พอร์ต ทำได้โดยการจ่ายไฟไปยัง Address Bus (SA0 – SA4) ดังตารางที่ 3.1

ตารางที่ 3.1 I/O พอร์ตของชิป CS8900A-CQ

Offset	Type	Description
0000h	Read/Write	Receive/Transmit Data (Port 0)
0002h	Read/Write	Receive/Transmit Data (Port 1)
0004h	Write-only	TxCMD (Transmit Command)
0006h	Write-only	TxLength(Transmit Length)
0008h	Read/Write	Interrupt Status Queue
000Ah	Read/Write	PacketPage Pointer
000Ch	Read/Write	PacketPage Data (Port 0)
000Eh	Read/Write	PacketPage Data (Port 1)

ตัวอย่างการเข้าถึงรีจิสเตอร์ภายในของ I/O Mode

ถ้าต้องการเข้าถึงรีจิสเตอร์ Receiver Event (Rx Event) ที่มี Address อยู่ที่ 0x0124 สามารถทำได้โดย

1. Write 0x24 (Least Significant 8 bit) to PacketPage Pointer นั่นคือ Address Bus = 0xa และ Data Bus = 0x24
2. Write 0x21 (Most Significant 8 bit) to PacketPage Pointer + 1 นั่นคือ Address Bus = 0xa + 1 = 0xb และ Data Bus = 0x01
3. เมื่อระบุรีจิสเตอร์แล้วก็จะสามารถ Read หรือ Write รีจิสเตอร์ได้โดยใช้ PacketPage Data พอร์ต 0 ทำได้โดย

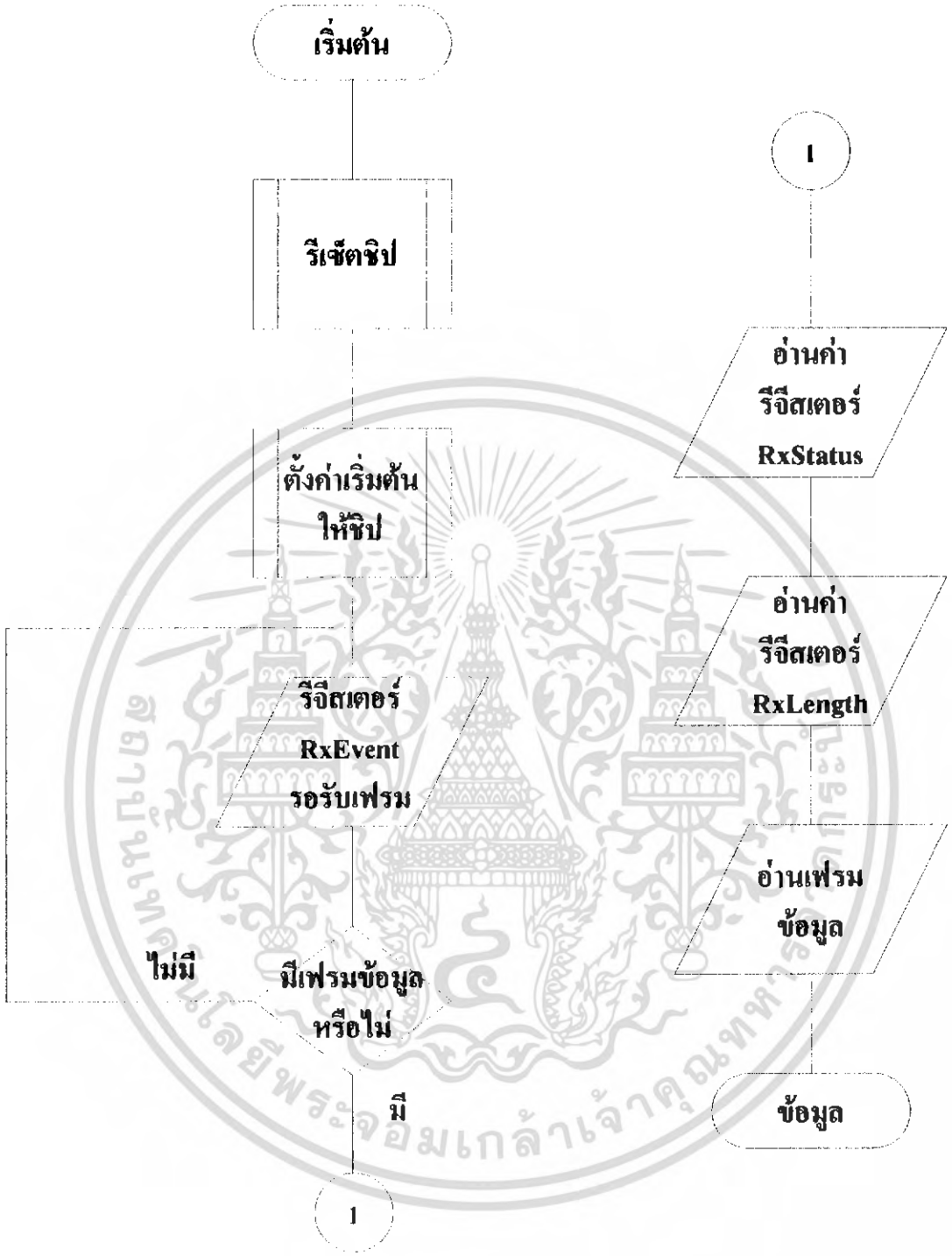
- Read / Write to PacketPage Data จะได้ Least Significant 8 บิต นั่นคือ Address Bus = 0xc และ Data Bus = Data Least Significant 8 บิต ที่ต้องการจะ Read หรือ Write
- Read / Write to PacketPage Data จะได้ Most Significant 8 บิต นั่นคือ Address Bus = 0xc + 1 = 0xd และ Data Bus = Data Most Significant 8 บิต ที่ต้องการจะ Read หรือ Write

3.2.5 กระบวนการส่งและรับข้อมูลของอีเทอร์เนตคอนโทรลเลอร์

ในกระบวนการส่งข้อมูลในอีเทอร์เนตคอนโทรลเลอร์ เริ่มต้นด้วยการรีเซ็ตเพื่อลบค่าเก่าออกจากอุปกรณ์ควบคุมการเชื่อมต่อระบบเครือข่ายก่อน จากนั้นทำการตั้งค่าเริ่มต้นของอุปกรณ์ควบคุมการเชื่อมต่อระบบเครือข่าย โดยใช้รีจีสเตอร์ RxCTL, LineCTL และค่า MAC Address ต่อมาตามด้วยการเขียนคำสั่งต้องการที่จะส่ง (TxCMD) และ เขียนความยาวของข้อมูล (TxLength) จากนั้นตรวจดูว่าบัสที่ต้องการส่งว่าว่างหรือไม่ โดยตรวจดูจากรีจีสเตอร์ BusST เมื่อ Bus ว่างก็จะสามารถส่งข้อมูลโดยเข้า I/O พอร์ต คือ Transmit Data (พอร์ต 0) ที่ Address 0x00 ดังในโฟว์ชาร์ตในการส่งข้อมูลของอีเทอร์เนตคอนโทรลเลอร์ ดังรูปที่ 3.7

ส่วนในกระบวนการรับข้อมูลใน Ethernet Controller เริ่มต้นทำการรีเซ็ตและตั้งค่าเริ่มต้นของอุปกรณ์ควบคุมการเชื่อมต่อระบบเครือข่าย และใช้รีจีสเตอร์ RxEvent รอรับ เฟรมเมื่อมีเฟรมเข้ามาให้ทำการอ่านค่า RxStatus และค่า RxLength จาก Receive Data พอร์ต 0 แต่มีข้อต้องระวังคือ RxStatus และ RxLength จะต้องอ่านจาก Most Significant bit ก่อนแล้วจึงค่อยอ่านจาก Least Significant bit คือ Set Address = 0x01 แล้วจึง Set Address = 0x00 จากนั้นก็ทำการอ่านค่าตามปกติจาก I/O Receive Data พอร์ต 0 คือ อ่าน Address = 0x00 แล้วจึงอ่านค่า 0x01 วนไปเรื่อยจนข้อมูลที่ส่งมาครบหมด ดังในโฟว์ชาร์ตในการรับข้อมูลของอีเทอร์เนตคอนโทรลเลอร์ ดังรูปที่ 3.8

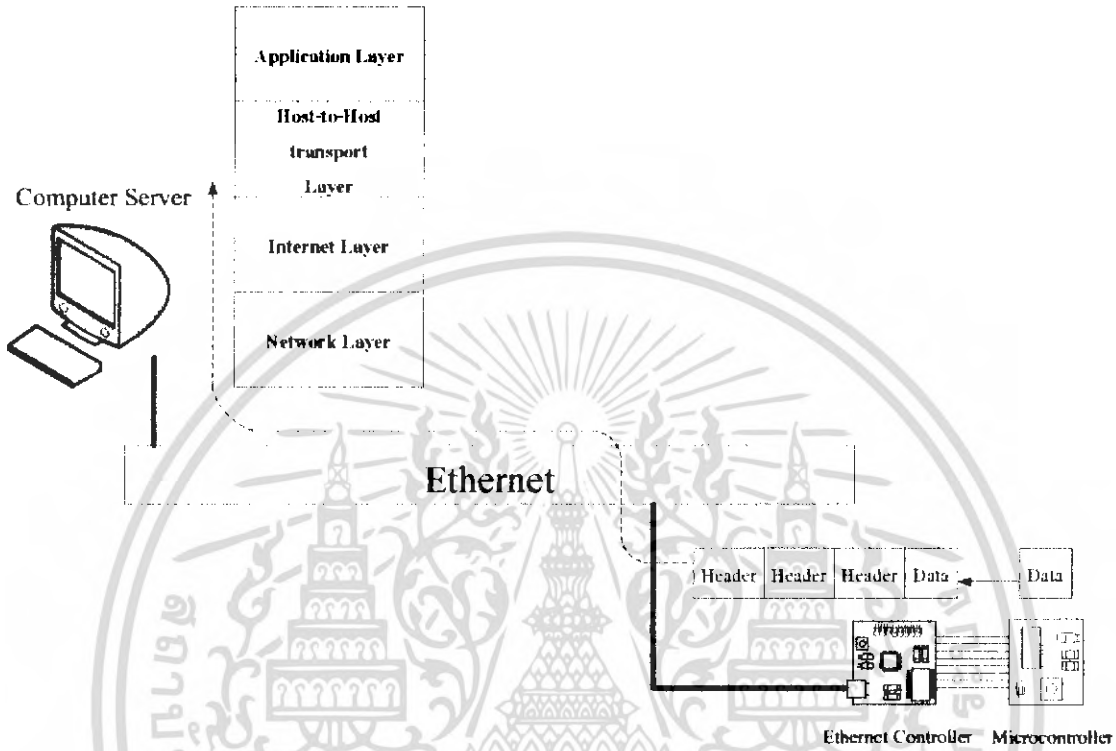
รูปที่ 3.7 แสดงโฟว์ชาร์ตในการส่งข้อมูลของอีเทอร์เนตคอนโทรลเลอร์



รูปที่ 3.8 แสดงโฟว์ชาร์ตในการรับข้อมูลของอีเทอร์เนตคอนโทรลเลอร์

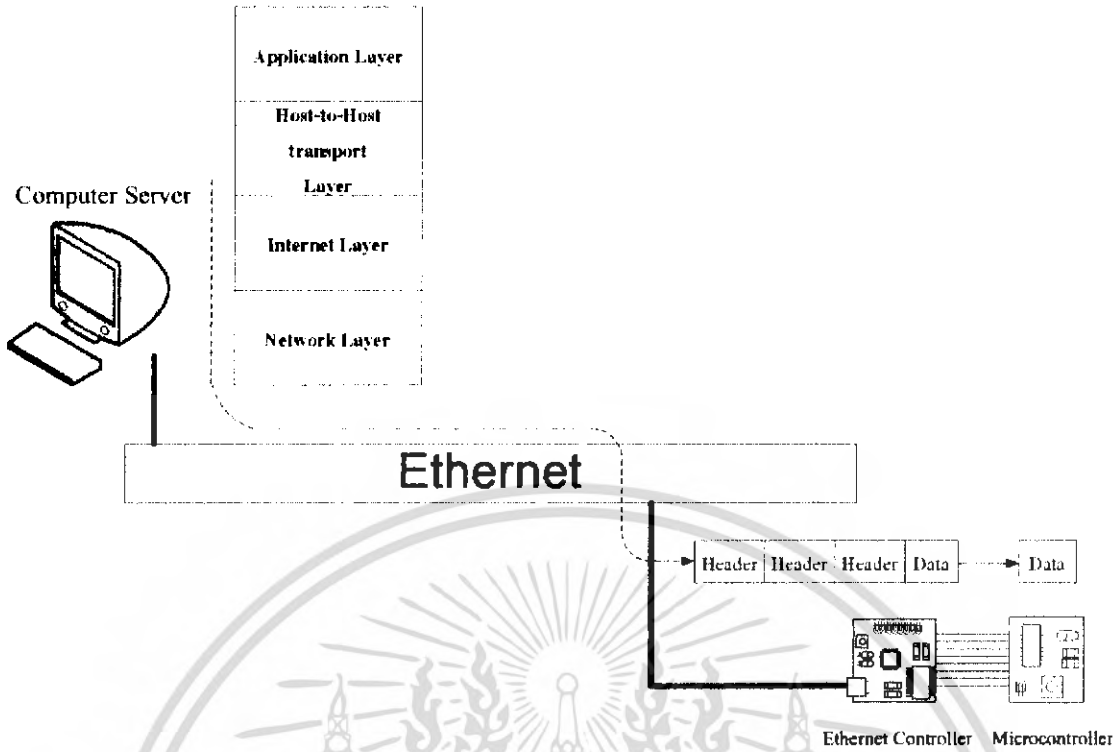
3.2.6 กระบวนการส่งและรับข้อมูลของระบบ

ในการส่งข้อมูลของฮาร์ดแวร์นั้น MCS-51 จะควบคุมการส่งข้อมูลให้กับอุปกรณ์ควบคุมการเชื่อมต่อระบบเครือข่าย แล้วส่งเฟรมข้อมูลที่ประกอบด้วยเฮดเดอร์และดาต้าส่งให้กับอุปกรณ์ควบคุมการเชื่อมต่อระบบเครือข่าย เพื่อส่งข้อมูลให้กับเครื่องคอมพิวเตอร์ศูนย์กลางเพื่อที่จะประมวลผลดังรูปที่ 3.9



รูปที่ 3.9 กระบวนการส่งข้อมูลของระบบ

ส่วนในการรับข้อมูลเครื่องคอมพิวเตอร์ศูนย์กลางจะทำการส่งเฟรมข้อมูลที่ประกอบด้วยเฮดเดอร์และดาต้าเพื่อควบคุมอุปกรณ์ที่ทำการเชื่อมต่อมายังฮาร์ดแวร์ แล้วในส่วนของอุปกรณ์ควบคุมการเชื่อมต่อระบบเครือข่ายจะส่งแต่เพียงดาต้ามายัง MCS-51 ดังรูปที่ 3.10



รูปที่ 3.10 กระบวนการรับข้อมูลของระบบ

3.2.7 กระบวนการทำงานของเครื่องคอมพิวเตอร์ที่ทำการส่งข้อมูลภาพให้กับฮาร์ดแวร์

ในการส่งข้อมูลภาพให้กับฮาร์ดแวร์จะมีการส่งข้อมูลภาพทีละแถว และมีการใส่เฮดเดอร์2 ไบต์ที่หัวแถวแต่ละแถวหลังจากใส่เฮดเดอร์แล้วจะทำการส่งข้อมูลไบต์แรกของแถวไปยังฮาร์ดแวร์ แล้วรอการตอบกลับจากฮาร์ดแวร์เพื่อส่งทำการส่งพิเซลต่อไป แล้วจะทำกระบวนนี้ต่อไปเรื่อยๆจนส่งข้อมูลภาพครบ ดังโฟว์ชาร์ตรูปที่ 3.11

3.2.8 กระบวนการทำงานของฮาร์ดแวร์ และเครื่องคอมพิวเตอร์ประมวลผลส่วนกลาง

ในการควบคุมอีเทอร์เนตคอนโทรลเลอร์โดยการใช้ไมโครคอนโทรลเลอร์ เพื่อแสดงถึงการเชื่อมต่อการรับส่งข้อมูล และการแสดงผลของข้อมูลที่เปรียบจากฐานข้อมูลจากเครื่องคอมพิวเตอร์ส่วนกลาง โดยมีขั้นตอนดังนี้

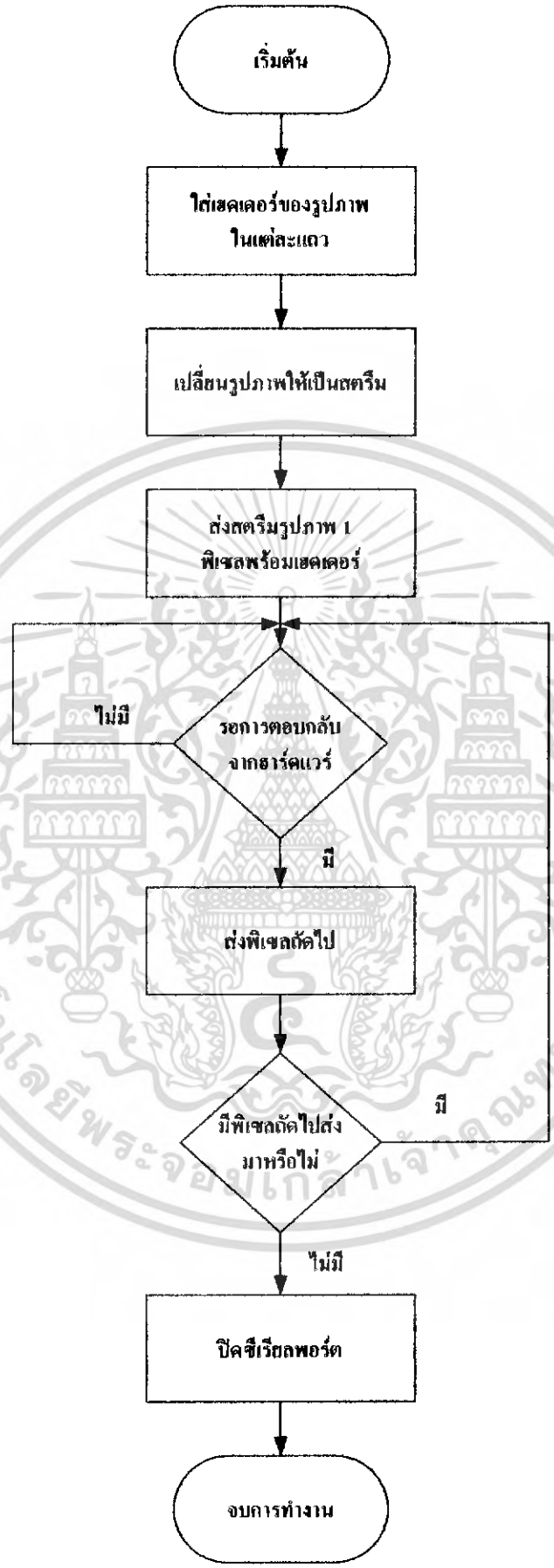
1. ทำการรีเซ็ตชิปและตั้งค่าให้กับอีเทอร์เนตคอนโทรลเลอร์
2. ส่งเฟรมข้อมูล Ping Request ซึ่งมี IP ปลายทางเป็นหมายเลข IP ของเครื่องคอมพิวเตอร์ส่วนกลาง
3. ทำการรอรับเฟรมที่เครื่องคอมพิวเตอร์ส่วนกลางส่งกลับมา ถ้าเฟรมที่ส่งกลับมาเป็น ARP Request ฮาร์ดแวร์จะทำการส่ง ARP Reply ส่งกลับไปยังเครื่องคอมพิวเตอร์ส่วนกลาง เมื่อเครื่องคอมพิวเตอร์ส่วนกลางทราบ MAC Address ของฮาร์ดแวร์แล้ว จะส่งเฟรม Ping Reply กลับมายังฮาร์ดแวร์ทำการตรวจสอบข้อมูลในเฟรมว่าตรงกับ Ping Request หรือไม่ ถ้าตรงให้ทำการส่ง

เฟรม UDP ให้กับเครื่องคอมพิวเตอร์ส่วนกลางเพื่อแสดงว่าการเชื่อมต่อเรียบร้อย ในขณะนี้ไฟ LED สีเหลืองจะสว่าง

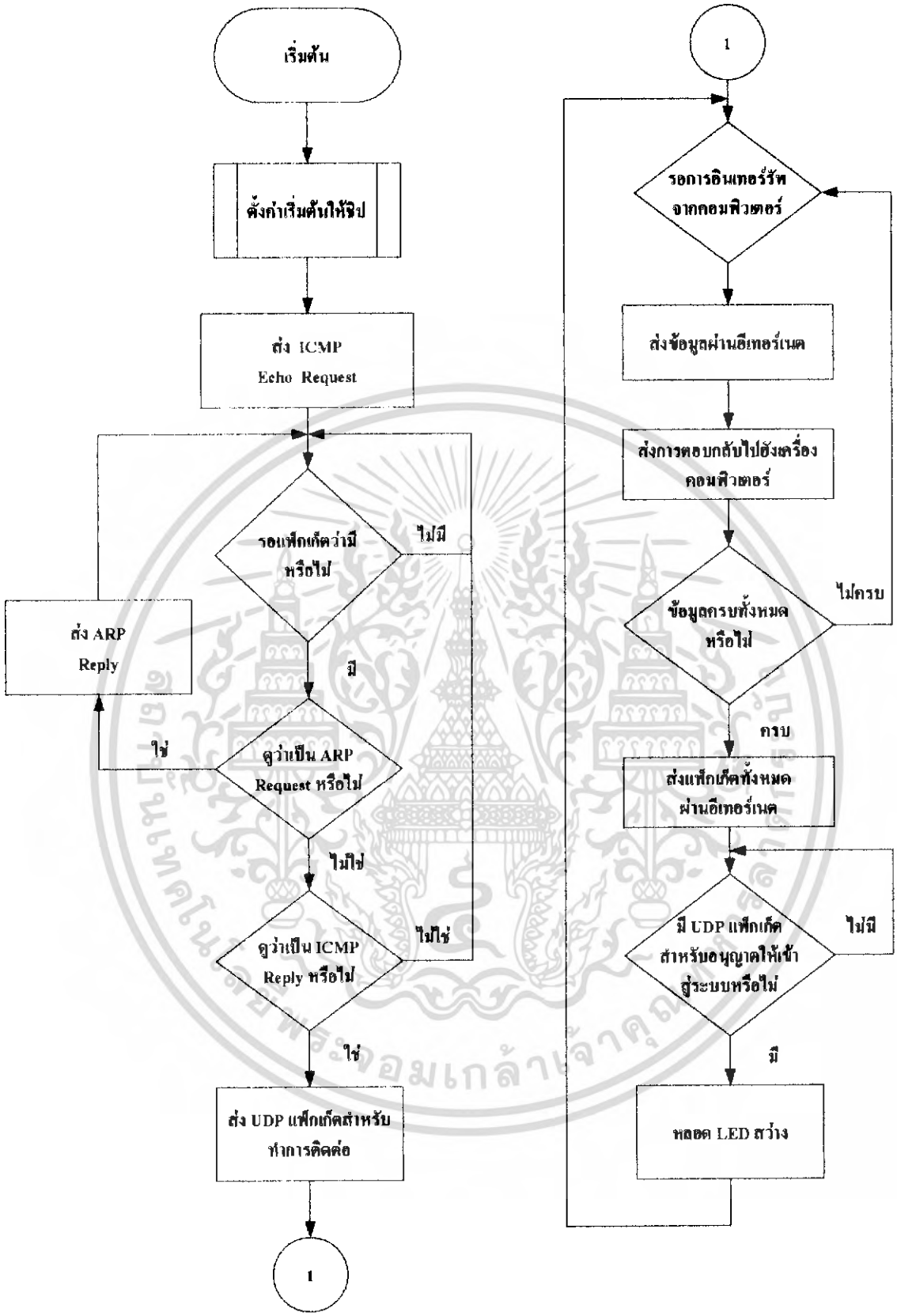
4. ฮาร์ดแวร์ทำการรอการอินเทอร์รัพจากเครื่องคอมพิวเตอร์ที่ส่งข้อมูลภาพ หลังจากนั้นเครื่องคอมพิวเตอร์ที่ส่งข้อมูลภาพก็จะทำการส่งข้อมูลภาพไป เมื่อทางฮาร์ดแวร์ได้รับข้อมูลแล้วก็จะทำการส่งสัญญาณ ACK กลับไป ทางเครื่องคอมพิวเตอร์ที่ส่งข้อมูลภาพก็จะส่งข้อมูลภาพไบต์ถัดไป ทำเช่นนี้ไปเรื่อยๆ จนข้อมูลภาพครบ แล้วจึงทำการส่งข้อมูลภาพทั้งหมดไปยังเครื่องคอมพิวเตอร์ส่วนกลาง
5. รอรับผลการตรวจสอบจากเครื่องคอมพิวเตอร์ส่วนกลาง ถ้าเป็นข้อมูลที่ตรงกับฐานข้อมูลจะส่วนเฟรมที่ทำให้หลอดไฟ LED สีเขียวสว่าง ถ้าเป็นข้อมูลที่ไม่ตรงหรือไม่มีในฐานข้อมูลจะส่งเฟรมที่ทำให้หลอดไฟ LED สีแดงสว่าง แล้วกลับไปรอรับผู้ใช้งานกดสวิตช์ต่อไป ดังโฟว์ชาร์ตรูปที่ 3.12

และในส่วนของเครื่องคอมพิวเตอร์ประมวลผลส่วนกลาง จะเขียนโปรแกรมโดยใช้ภาษา JAVA เพื่อทำการรับข้อมูลจากฮาร์ดแวร์แล้วนำไปเปรียบเทียบกับฐานข้อมูล เมื่อเปรียบเทียบเสร็จแล้วจะส่งผลการเปรียบเทียบให้กับฮาร์ดแวร์ โดยมีขั้นตอนการทำงานดังนี้

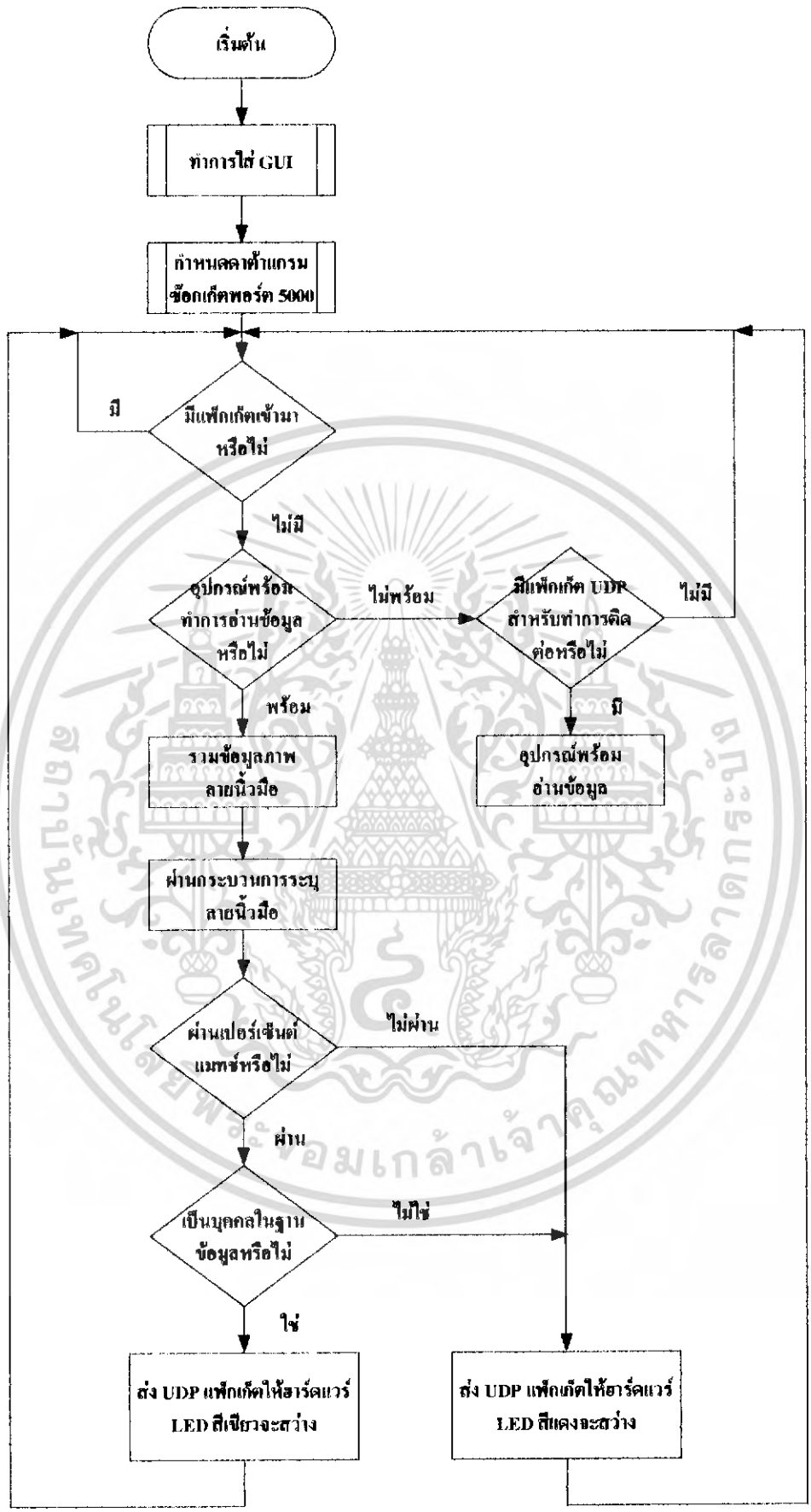
1. ทำการสร้าง GUI (Graphic User Interface) ชนิด J เฟรม แล้วเปิด UDP Socket ซึ่งมีค่าของพอร์ตเท่ากับ 5000 แล้วทำการรอรับเฟรมข้อมูลชนิด UDP จากฮาร์ดแวร์ (ในส่วนของเฟรม Ping และ ARP นั้นการ์ดแลนจะเป็นตัวจัดการในการส่งเฟรม ไม่เกี่ยวข้องกับตัวโปรแกรม)
2. รอทำการรับเฟรม UDP ที่ฮาร์ดแวร์ส่งมาเพื่อแสดงถึงการเชื่อมต่อ ถ้ายังไม่ได้รับเฟรมนี้ตัวโปรแกรมจะไม่ยอมให้ทำการตรวจสอบข้อมูลกับฐานข้อมูลได้ เมื่อได้รับเฟรมนี้แล้วตัวโปรแกรมจะการประกอบภาพลายนิ้วมือ แล้วทำการหาลักษณะเด่นของลายนิ้วมือ
3. นำข้อมูลลายนิ้วมือที่ได้มาเปรียบเทียบเปอร์เซ็นต์การแมทช์ถ้าผ่านเกณฑ์ที่ตั้งเอาไว้ โปรแกรมจะส่งเฟรมข้อมูลที่ทำให้หลอดไฟ LED สีเขียวสว่าง แต่ถ้าข้อมูลที่ได้รับไม่ผ่านเกณฑ์ที่ตั้งเอาไว้ โปรแกรมจะส่งเฟรมที่ทำให้หลอดไฟ LED สีแดงสว่าง ดังโฟว์ชาร์ตรูปที่ 3.13



รูปที่ 3.11 แสดงโฟว์ชาร์ตของการทำงานของฮาร์ดแวร์ และเครื่องคอมพิวเตอร์ประมวลผลส่วนกลาง



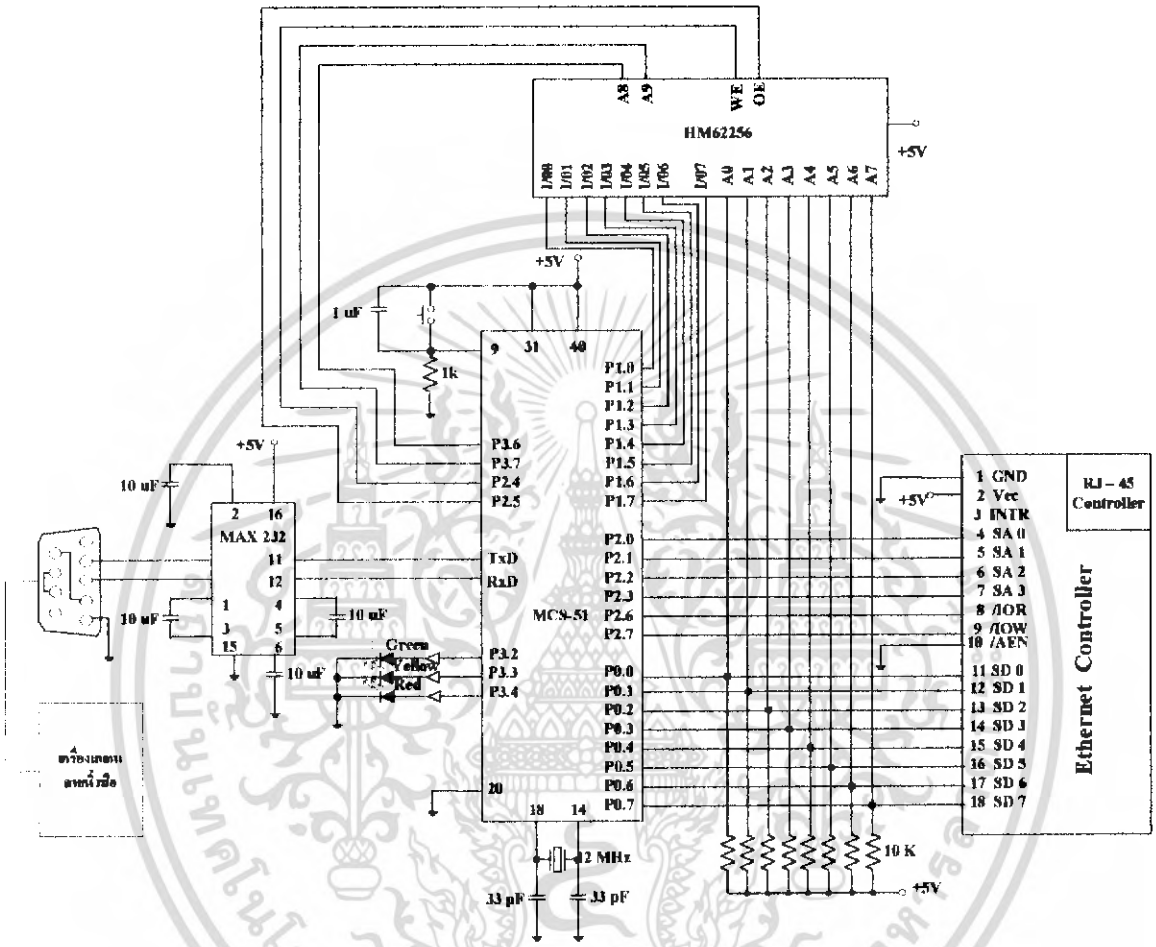
รูปที่ 3.12 แสดงไฟว์ชาร์ตของการทำงานของฮาร์ดแวร์



รูปที่ 3.13 แสดงโฟว์ชาร์ตของการทำงานโปรแกรมของเครื่องคอมพิวเตอร์ประมวลผลส่วนกลาง

3.3 เครื่องสแกนลายนิ้วมือ

ในส่วนนี้จะนำโมดูลสแกนลายนิ้วมือและเมมโมรี่ภายนอก มาติดเข้ากับวงจรรูปที่ 3.6 โดย โมดูลสแกนลายนิ้วมือ (FCP301) จะรับส่งข้อมูลผ่านทางซีเรียลพอร์ต เพื่อส่งข้อมูลรูปภาพมาให้ไมโครคอนโทรลเลอร์มาเก็บไว้ในเมมโมรี่ภายนอก (HM62256) ดังรูปที่ 3.14



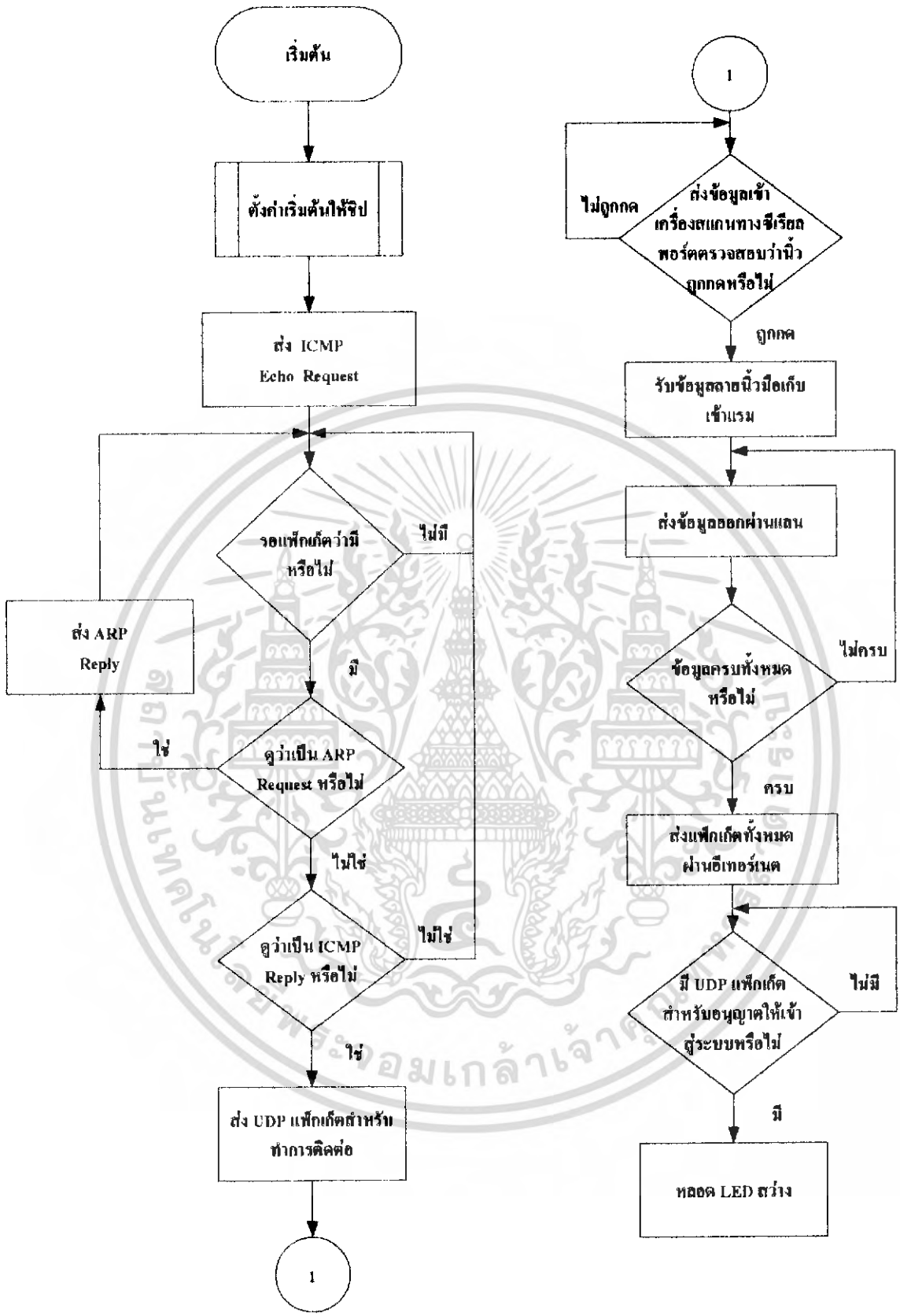
รูปที่ 3.14 แสดงการติดต่อกับอุปกรณ์ต่างๆ

โดยโมดูลสแกนลายนิ้วมือนี้ จะมีโปรโตคอลในการติดต่อเป็นเฟรมโดยใช้ ซีเรียลพอร์ต เป็นอินเตอร์เฟสในการติดต่อระหว่างไมโครคอนโทรลเลอร์กับโมดูลสแกนลายนิ้วมือ โดยที่ไมโครคอนโทรลเลอร์จะส่งคำสั่งให้แล้วรอการส่งข้อมูลกลับของไมโครคอนโทรลเลอร์ ซึ่งระบบของฮาร์ดแวร์จะมีการทำงานดังโฟว์ชาร์ตรูปที่ 3.15

1. ทำการรีเซ็ตชิปและตั้งค่าให้กับอีเทอร์เนตคอนโทรลเลอร์
2. ส่งเฟรมข้อมูล Ping Request ซึ่งมี IP ปลายทางเป็นหมายเลข IP ของเครื่องคอมพิวเตอร์ส่วนกลาง
3. ทำการรอรับเฟรมที่เครื่องคอมพิวเตอร์ส่วนกลางส่งกลับมา ถ้าเฟรมที่ส่งกลับมาเป็น ARP Request ฮาร์ดแวร์จะทำการส่ง ARP Reply ส่งกลับไปยังเครื่องคอมพิวเตอร์ส่วนกลาง เมื่อเครื่องคอมพิวเตอร์ส่วนกลางทราบ MAC Address ของฮาร์ดแวร์แล้ว จะส่งเฟรม Ping Reply กลับมายัง

ฮาร์ดแวร์ทำการตรวจสอบข้อมูลในเฟรมว่าตรงกับ Ping Request หรือไม่ ถ้าตรงให้ทำการส่งเฟรม UDP ให้กับเครื่องคอมพิวเตอร์ส่วนกลางเพื่อแสดงว่าการเชื่อมต่อเรียบร้อย ในขณะนี้ไฟ LED สีเหลืองจะสว่าง

4. ส่งข้อมูลเข้าเครื่องสแกนลายนิ้วมือทางซีเรียลพอร์ตตรวจสอบดูว่านิ้วถูกกดหรือไม่ ถ้าถูกกดให้รับข้อมูลมาเก็บที่แรมแล้วทำการส่งผ่านแลน ส่งแพ็กเก็ตทั้งหมดผ่านอีเทอร์เนตแล้วดูว่ามี UDP แพ็กเก็ตที่อนุญาตให้เข้าสู่ระบบ ถ้ามีหลอด LED สีเขียวจะสว่าง



รูปที่ 3.15 โฟว์ชาร์ตการทำงานของฮาร์ดแวร์

บทที่ 4
การทดลองและผลการทดลอง

หลังจากที่ทราบถึงทฤษฎีหรือหลักการในบทที่ 2 และการคำนวณและทำการสร้างวงจรในบทที่ 3 เราก็สามารถที่จะทำการทดลองโครงงาน โดยจะแบ่งการทดลองออกเป็น 4 ส่วนใหญ่ๆ ด้วยกัน คือ อัลกอริทึมในการตรวจสอบลายนิ้วมือ ฮาร์ดแวร์ของระบบที่ใช้รับและส่งข้อมูลผ่านวงแลน การส่งข้อมูลลายนิ้วมือออกสู่วงแลนเพื่อไปประมวลผล และการประมวลผลโดยรวมฮาร์ดแวร์โมดูลสแกนลายนิ้วมือเข้ากับระบบ โดยมีขั้นตอนการทดลอง และผลการทดลองดังนี้

4.1 ผลการทดลองตอนที่ 1 อัลกอริทึมในการตรวจสอบลายนิ้วมือ

จุดประสงค์การทดลองตอนนี้คือ ทำการประมวลผลภาพเชิงดิจิตอลจากภาพลายนิ้วมือที่ส่งมาจากฮาร์ดแวร์เพื่อที่จะนำไปหาจุดเด่นของลายนิ้วมือ เพื่อที่จะนำไปสร้างเทมเพลตเพื่อเก็บข้อมูล หรือตรวจสอบลายนิ้วมือกับฐานข้อมูลของเครื่องคอมพิวเตอร์ศูนย์กลางต่อไป

รูปภาพปกติที่ได้จากเครื่องสแกนเนอร์จะเป็นดังรูป 4.1 ซึ่งไม่ค่อยมีความคมชัด ถ้านำไปประมวลผลอาจทำให้เกิดความผิดพลาดขึ้นได้

รูปที่ 4.1 แสดงรูปลายนิ้วมือที่ได้จากเครื่องสแกนเนอร์

ต่อมาจึงนำภาพจากสแกนเนอร์มาเข้ากระบวนการปรับคุณภาพของภาพ (Image Enhancement) ซึ่งจะทำให้ได้ภาพที่คมชัดขึ้น โดยใช้กระบวนการ Short Time Fourier Transform จะได้ภาพดังรูปที่ 4.2

รูปที่ 4.2 รูปที่ผ่านกระบวนการปรับคุณภาพของภาพ

เมื่อได้รูปที่เหมาะสมต่อการหาจุดเด่นลายนิ้วมือแล้ว กระบวนการถัดมาคือ การทำภาพให้บาง (Thinning) ซึ่งจะทำให้เส้นลายนิ้วมือบางลง และการกำจัดเส้นกิ่งเส้นสะพาน (Imperfect Removal) จะช่วยกำจัดเส้นกิ่งและเส้นสะพานเพื่อความถูกต้องในการหาจุดเด่นลายนิ้วมือ ดังรูปที่ 4.3

รูปที่ 4.3 รูปที่ผ่านกระบวนการทำภาพให้บางและการกำจัดเส้นกิ่งเส้นสะพาน

เมื่อได้รูปที่เหมาะสมต่อการหาจุดเด่นแล้วก็เข้าสู่กระบวนการหาจุดเด่นของลายนิ้วมือ (Minutia Extraction) ซึ่งจะได้ข้อมูลของจุดเด่นลายนิ้วมือเป็นตำแหน่งและมุม เพื่อนำไปเก็บในฐานข้อมูลต่อไปดังรูปที่ 4.4

รูปที่ 4.4 แสดงจุดเด่นของลายนิ้วมือ

ดังนั้นจึงทำการวัดประสิทธิภาพของอัลกอริทึมในการจำแนกและตัดสินบุคคลโดยมีค่าผลการที่จะชี้วัด 2 ค่าได้แก่

1.False Rejection Rate (FRR) คือ อัตราการปฏิเสธตัวจริง จะบอกถึงร้อยละของความผิดพลาดที่เกิดขึ้นเมื่อตัวจริงถูกปฏิเสธ โดยทำการทดลองดังนี้

- เก็บเทมเพลตลายนิ้วมือของบุคคลที่จะนำมาทดสอบในฐานข้อมูล
- นำลายนิ้วมือของบุคคลนั้นมาทำการตรวจสอบว่าสามารถระบุตัวได้หรือไม่ ในการทดลองนี้ ได้ทำการตรวจสอบภาพลายนิ้วมือบุคคลนั้นจากการสแกนทั้งหมด 300 ครั้ง พบว่า สามารถระบุตัวได้ทั้งหมด 232 ภาพ และถูกปฏิเสธทั้งหมด 68 ภาพ ซึ่งคิดเป็นค่า FRR เท่ากับ 22.667 เปอร์เซ็นต์ ดังตารางที่ 4.1

ตารางที่ 4.1 แสดงผลของอัตราการปฏิเสธตัวจริงจากการตรวจสอบ 300 ครั้ง

ลายนิ้วมือบุคคลอ้างอิง											
ลำดับ	เปอร์เซ็นต์	ลำดับ	เปอร์เซ็นต์	ลำดับ	เปอร์เซ็นต์	ลำดับ	เปอร์เซ็นต์	ลำดับ	เปอร์เซ็นต์	ลำดับ	เปอร์เซ็นต์
1	52	26	48	51	72	76	76	101	60	126	60
2	80	27	36	52	56	77	60	102	64	127	64
3	48	28	44	53	40	78	60	103	68	128	60
4	60	29	52	54	52	79	48	104	40	129	40
5	72	32	48	55	64	80	76	105	68	132	56
6	64	31	64	56	68	81	48	106	60	131	52
7	56	32	32	57	72	82	52	107	60	132	52

8	64	33	44	58	64	83	52	108	60	133	56
9	64	34	64	59	60	84	68	109	44	134	40
10	100	35	44	60	48	85	60	110	68	135	52
11	52	36	72	61	72	86	48	111	48	136	64
12	64	37	52	62	48	87	68	112	56	137	36
13	60	38	36	63	72	88	72	113	68	138	56
14	60	39	56	64	36	89	24	114	40	139	64
15	52	40	36	65	48	90	48	115	48	140	68
16	72	41	60	66	72	91	60	116	60	141	48
17	60	42	56	67	44	92	44	117	52	142	68
18	28	43	40	68	52	93	60	118	48	143	40
19	32	44	28	69	60	94	60	119	56	144	60
20	48	45	56	70	64	95	68	120	52	145	52
21	48	46	28	71	48	96	56	121	48	146	56
22	48	47	48	72	76	97	60	122	68	147	48
23	60	48	52	73	64	98	64	123	56	148	64
24	72	49	44	74	64	99	64	124	48	149	44
25	32	50	64	75	52	100	48	125	52	150	56

ลายนิ้วมือบุคคลอ้างอิง

ลำดับ	เปอร์เซ็นต์	ลำดับ	เปอร์เซ็นต์	ลำดับ	เปอร์เซ็นต์	ลำดับ	เปอร์เซ็นต์	ลำดับ	เปอร์เซ็นต์	ลำดับ	เปอร์เซ็นต์
151	56	176	60	201	76	226	40	251	52	276	44
152	68	177	56	202	60	227	52	252	56	277	52
153	52	178	56	203	48	228	48	253	48	278	44
154	64	179	60	204	48	229	56	254	52	279	72
155	44	180	64	205	32	232	52	255	56	280	72
156	44	181	68	206	32	231	48	256	64	281	64
157	52	182	52	207	44	232	68	257	60	282	48

158	60	183	56	208	36	233	48	258	52	283	36
159	52	184	68	209	48	234	52	259	52	284	64
160	64	185	52	210	52	235	56	260	64	285	52
161	48	186	52	211	64	236	48	261	52	286	40
162	76	187	68	212	68	237	56	262	64	287	60
163	56	188	68	213	48	238	64	263	68	288	36
164	52	189	44	214	44	239	56	264	40	289	36
165	32	190	56	215	48	240	52	265	44	290	48
166	56	191	40	216	72	241	52	266	60	291	36
167	44	192	32	217	36	242	52	267	52	292	48
168	76	193	44	218	72	243	72	268	52	293	36
169	56	194	48	219	52	244	48	269	48	294	40
170	48	195	28	220	44	245	48	270	48	295	56
171	80	196	44	221	48	246	48	271	40	296	44
172	64	197	32	222	44	247	48	272	60	297	36
173	44	198	32	223	40	248	52	273	44	298	60
174	60	199	84	224	48	249	80	274	60	299	44
175	48	200	64	225	44	250	52	275	36	300	52
						68					

2.False Acceptance Rate (FAR) คือ อัตราการยอมรับตัวปลอม เป็นร้อยละของความผิดพลาดที่เกิดขึ้นจากการที่ตัวปลอมได้รับการยอมรับ โดยทำการทดลองดังนี้

- ใช้เทมเพลตลายนิ้วมือชุดเดิมจากการทดลองหาค่า FRR เป็นลายนิ้วมืออ้างอิง

- นำลายนิ้วมือของบุคคลอื่นมาทำการตรวจสอบกับลายนิ้วมืออ้างอิงว่าสามารถผ่านเข้าไปในระบบได้หรือไม่ ในการทดลองนี้ได้ทำการตรวจสอบกับลายนิ้วมือตัวอย่างของบุคคลอื่นๆ ทั้งหมด 18 คน คนละ 10 ภาพ ผลที่ออกมาคือ มีภาพที่ได้รับการยอมรับทั้งหมด 20 ภาพ จากทั้งหมด 180 ภาพ ซึ่งคิดเป็นค่า FAR เท่ากับ 11.11 เปอร์เซ็นต์ ดังตารางที่ 4.2

ตารางที่ 1.2 แสดงผลของอัตราการยอมรับตัวปลอมจากบุคคล 18 คน คนละ 10 ภาพ

ลายนิ้วมือบุคคลทดสอบ									
ลำดับ	บุคคลที่1	บุคคลที่2	บุคคลที่3	บุคคลที่4	บุคคลที่5	บุคคลที่6	บุคคลที่7	บุคคลที่8	บุคคลที่9
1	36	36	32	28	48	28	32	32	36
2	32	32	28	24	48	32	24	32	32
3	36	32	28	24	48	32	44	56	36
4	40	28	28	36	52	36	40	36	40
5	28	36	28	36	36	40	28	36	32
6	32	28	32	32	48	32	36	44	36
7	32	36	32	28	48	24	44	28	40
8	32	32	32	32	44	28	28	48	44
9	36	40	28	32	36	32	52	40	32
10	28	32	32	32	44	36	40	40	44
	0	0	0	0	6	0	1	2	0

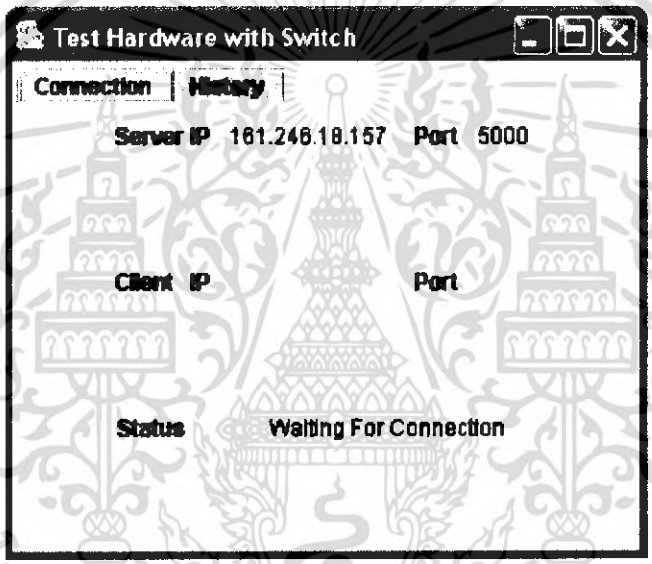
ลายนิ้วมือบุคคลทดสอบ									
ลำดับ	บุคคลที่10	บุคคลที่11	บุคคลที่12	บุคคลที่13	บุคคลที่14	บุคคลที่15	บุคคลที่16	บุคคลที่17	บุคคลที่18
1	32	40	32	24	44	28	36	32	36
2	32	40	36	40	48	36	48	36	48
3	40	44	32	24	36	20	44	44	40
4	40	44	48	24	44	40	40	28	48
5	28	48	36	28	44	32	56	32	56
6	32	32	40	24	44	40	36	28	44
7	24	36	36	20	40	44	32	40	36
8	28	40	32	32	40	48	40	36	48
9	28	36	28	24	44	40	36	28	32
10	36	32	40	36	48	36	36	32	32
	0	1	1	0	2	1	2	0	4

4.2 ผลการทดลองตอนที่ 2 ฮาร์ดแวร์ของระบบที่ใช้รับและส่งข้อมูลผ่านวงแลน

จุดประสงค์ของการทดลองตอนนี้คือ สร้างฮาร์ดแวร์ต้นแบบที่สามารถ รับส่งข้อมูลผ่านกับคอมพิวเตอร์ศูนย์กลาง โดยที่ฮาร์ดแวร์จะมีสวิตซ์ 4 ปุ่ม เมื่อกดสวิตซ์ฮาร์ดแวร์จะส่งข้อมูลที่ต่างกันเข้าสู่คอมพิวเตอร์ศูนย์กลาง แล้วคอมพิวเตอร์ศูนย์กลางจะทำการเปรียบเทียบค่าที่ส่งมากับฐานข้อมูลแล้วส่งสัญญาณให้กับฮาร์ดแวร์แสดงผลออกทางหลอดไฟ LED ว่าข้อมูลที่ส่งไปตรงกับข้อมูลที่อยู่ในฐานข้อมูลหรือไม่

4.2.1 โปรแกรมที่ใช้รันบนเครื่องคอมพิวเตอร์ศูนย์กลาง

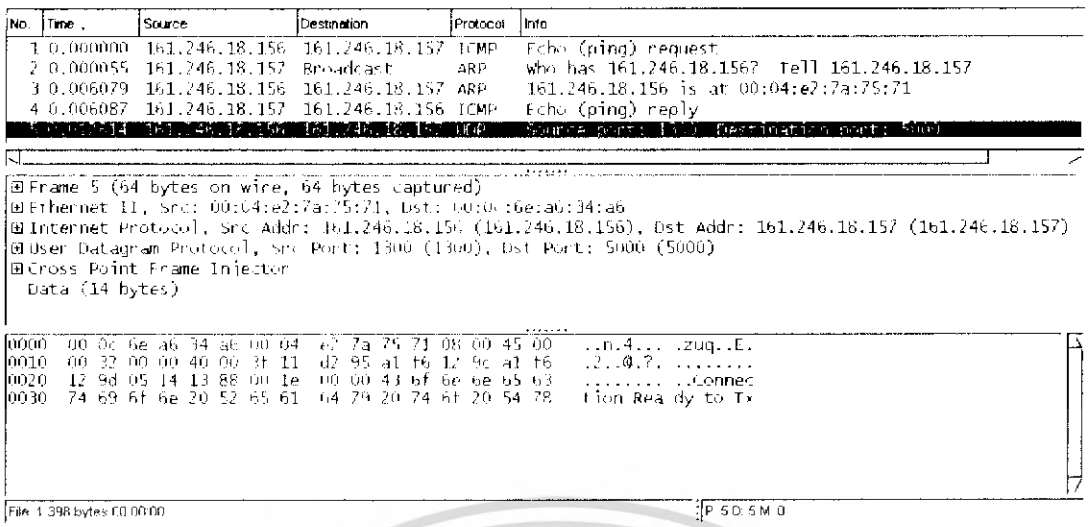
โปรแกรมที่ใช้บนเครื่องคอมพิวเตอร์ศูนย์กลางนั้นจะใช้ภาษา Java ดังรูป 4.5 เมื่อเปิดโปรแกรม จะแสดงหมายเลข IP ของเครื่องคอมพิวเตอร์ศูนย์กลาง และหมายเลขพอร์ตที่กำหนดไว้เพื่อที่จะติดต่อกับฮาร์ดแวร์



รูปที่ 4.5 แสดงโปรแกรมที่รันบนเครื่องคอมพิวเตอร์ศูนย์กลาง

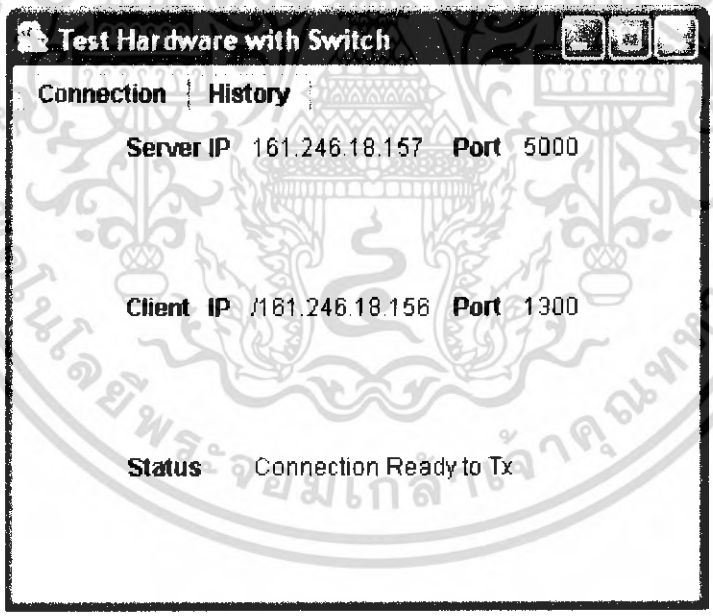
4.2.2 ฮาร์ดแวร์ทำการส่งเฟรมเชื่อมต่อกับเครื่องคอมพิวเตอร์ศูนย์กลาง

เมื่อฮาร์ดแวร์เปิดเครื่องแล้วจะทำการส่งสัญญาณ ping ให้กับเครื่องคอมพิวเตอร์ศูนย์กลาง ถ้าเครื่องคอมพิวเตอร์ศูนย์กลางยังไม่รู้ค่า MAC Address ของฮาร์ดแวร์จะทำการส่ง ARP Request ไปก่อนแล้วรอรับ ARP Reply จากฮาร์ดแวร์ จากนั้นจึงส่งสัญญาณ ping กลับไปยังฮาร์ดแวร์ เมื่อฮาร์ดแวร์ได้รับสัญญาณ ping แล้ว จะทำการส่งเฟรม UDP เพื่อยืนยันการเชื่อมต่อกับเครื่องคอมพิวเตอร์ศูนย์กลาง ดังรูปที่ 4.6 เป็นผลของการตรวจจับเฟรมที่อยู่ในวงแลน



รูปที่ 4.6 แสดงผลของการตรวจจับเฟรมที่พบในวงแลนเมื่อฮาร์ดแวร์ทำการเชื่อมต่อกับเครื่องคอมพิวเตอร์ศูนย์กลาง

เมื่อเครื่องคอมพิวเตอร์ศูนย์กลางได้รับเฟรม UDP ที่ยืนยันการเชื่อมต่อแล้วจะแสดงหมายเลข IP และพอร์ตที่มาเชื่อมต่อกัน ของฮาร์ดแวร์ ดังรูปที่ 4.7 ในตอนนี้เครื่องคอมพิวเตอร์ศูนย์กลางจะยอมให้รับเฟรมแล้วตรวจสอบกับฐานข้อมูล

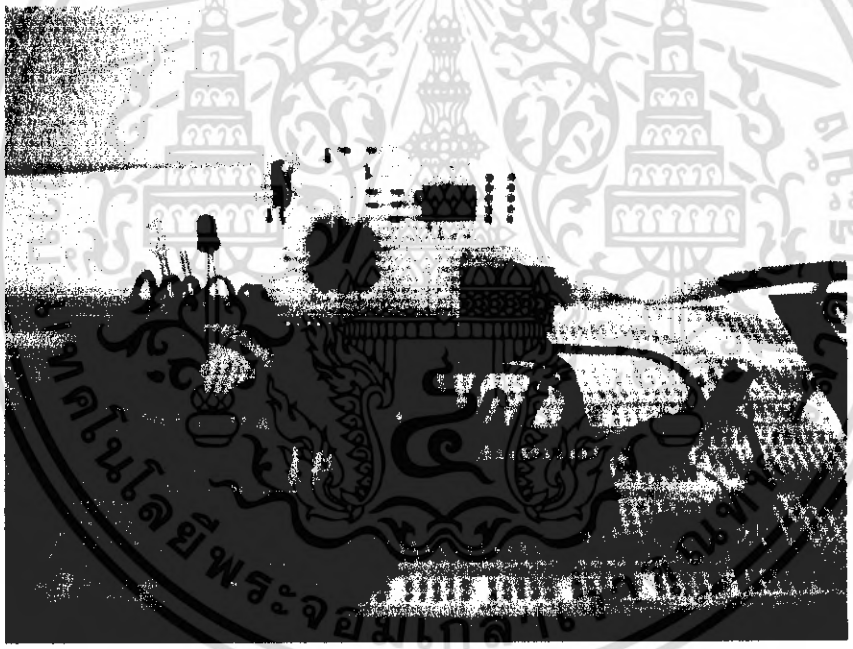


รูปที่ 4.7 แสดงโปรแกรมที่รันบนเครื่องคอมพิวเตอร์ศูนย์กลางหลังจากรับเฟรม UDP ที่ยืนยันการเชื่อมต่อ

และในเครื่องคอมพิวเตอร์ศูนย์กลางจะต้องสร้างฐานข้อมูลเตรียมไว้ก่อน ดังรูปที่ 4.8 โดยข้อมูลของคนที่ 1 จะตรงกับข้อมูลของสวิตช์ตัวที่ 1 ของฮาร์ดแวร์ ข้อมูลของคนที่ 2 จะตรงกับข้อมูลของสวิตช์ตัวที่ 2 ของฮาร์ดแวร์ และ ข้อมูลของคนที่ 3 จะตรงกับข้อมูลของสวิตช์ตัวที่ 3 ของฮาร์ดแวร์ แต่จะไม่อนุญาต ให้ผ่านเข้าสู่ระบบได้ (AUT - NO)

รูปที่ 4.8 แสดงฐานข้อมูลที่เก็บไว้ในเครื่องคอมพิวเตอร์ศูนย์กลาง

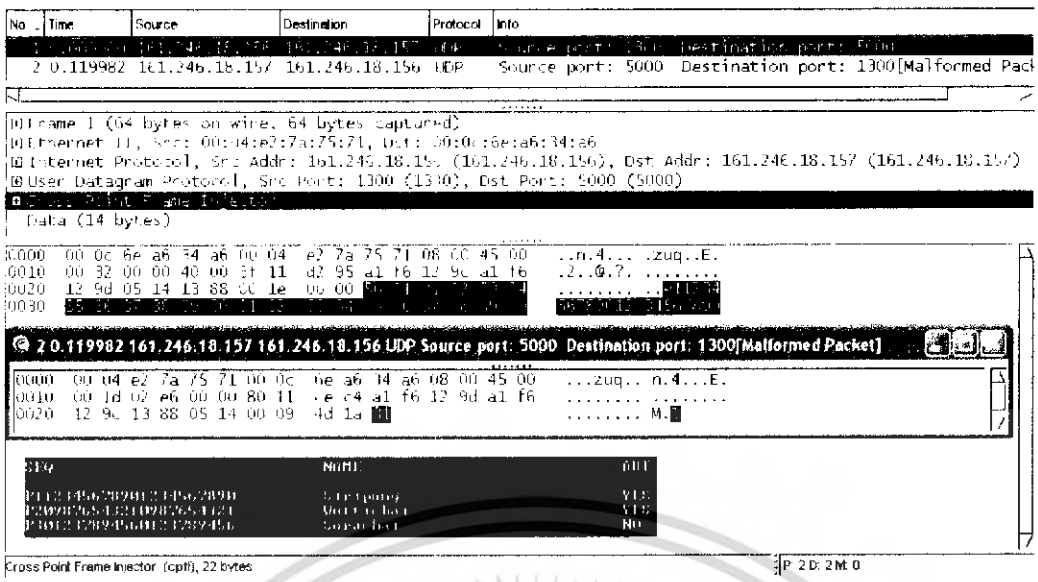
เมื่อทุกอย่างเชื่อมต่อเรียบร้อยแล้วพร้อมที่จะกดสวิตช์ ฮาร์ดแวร์จะแสดงไฟสีเหลือง ดังรูปที่ 4.9



รูปที่ 4.9 แสดงรูปฮาร์ดแวร์ที่พร้อมใช้งาน

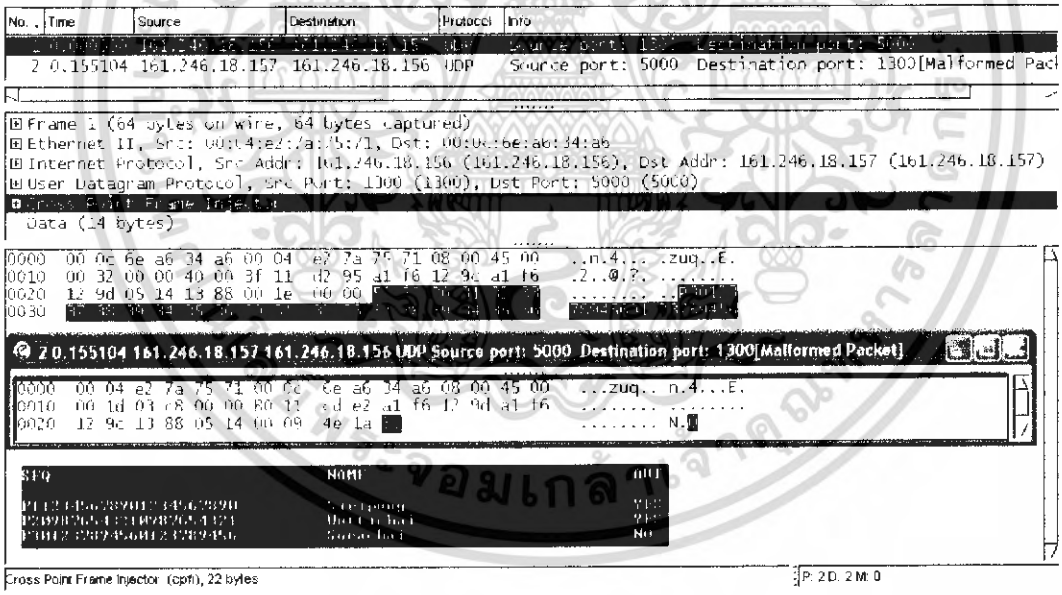
4.2.3 ทดลองกดสวิตช์แต่ละตัว

สวิตช์ตัวที่ 1 และ 2 จะเหมือนกันคือส่งเฟรมข้อมูลไปยังเครื่องคอมพิวเตอร์ศูนย์กลาง และ พบว่าข้อมูลตรงกับฐานข้อมูลแล้ว อนุญาตให้เข้าสู่ระบบได้ เครื่องคอมพิวเตอร์ศูนย์กลางจะทำการส่งเฟรมข้อมูลที่มีความหมายว่า อนุญาต ให้กับฮาร์ดแวร์ ดังรูปที่ 4.10 แล้วฮาร์ดแวร์จะติดไฟสีเขียว



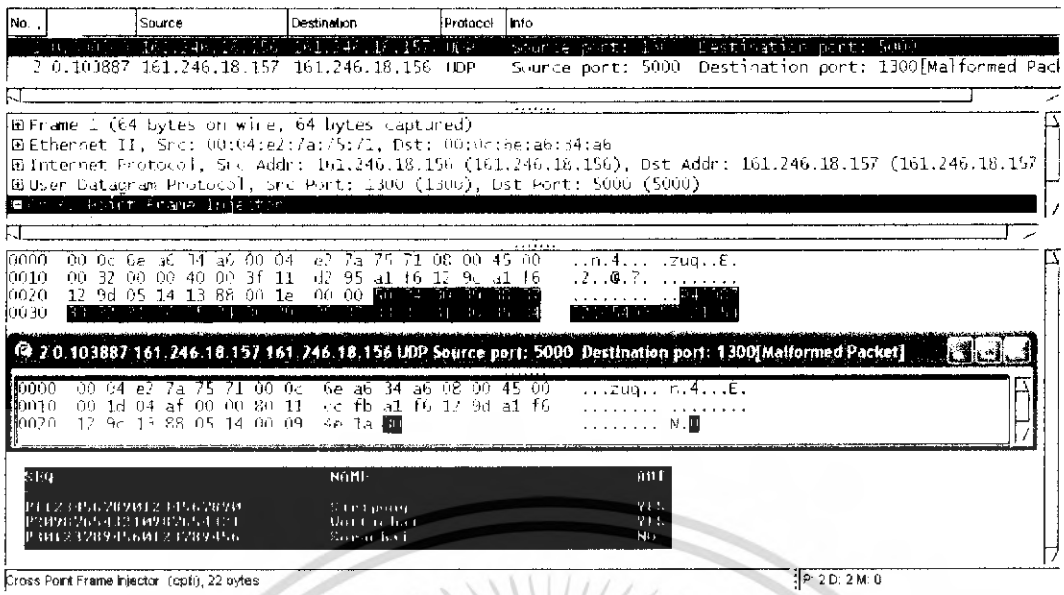
รูปที่ 4.10 แสดงผลของการตรวจจับเฟรมที่พบในวงแลน เมื่อกดสวิตช์ตัวที่ 1

ส่วนสวิตช์ตัวที่ 3 จะส่งเฟรมข้อมูลไปยังเครื่องคอมพิวเตอร์ศูนย์กลาง และ พบว่าข้อมูลตรงกับฐานข้อมูลแล้ว แต่ไม่อนุญาตให้เข้าสู่ระบบได้ เครื่องคอมพิวเตอร์ศูนย์กลางจะทำการส่งเฟรมข้อมูลที่มีความหมายว่า ไม่อนุญาต ให้กับฮาร์ดแวร์ ดังรูปที่ 4.11 แล้วฮาร์ดแวร์จะติดไฟสีแดง



รูปที่ 4.11 แสดงผลของการตรวจจับเฟรมที่พบในวงแลน เมื่อกดสวิตช์ตัวที่ 3

และสวิตช์ตัวที่ 4 จะส่งเฟรมข้อมูลไปยังเครื่องคอมพิวเตอร์ศูนย์กลาง แต่ข้อมูลไม่ตรงกับฐานข้อมูล เครื่องคอมพิวเตอร์ศูนย์กลางจะทำการส่งเฟรมข้อมูลที่มีความหมายว่า ไม่อนุญาต ให้กับฮาร์ดแวร์ ดังรูปที่ 4.12 แล้วฮาร์ดแวร์จะติดไฟสีแดง

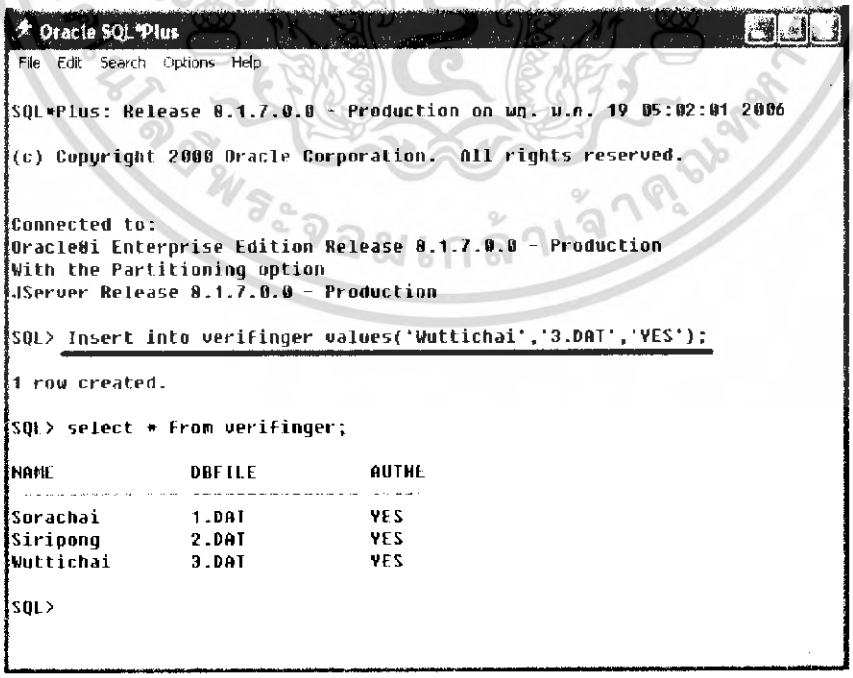


รูปที่ 4.12 แสดงผลของการตรวจจับเฟรมที่พบในวงแลน เมื่อกดสวิตช์ตัวที่ 4

4.3 ผลการทดลองตอนที่ 3 การส่งข้อมูลลายนิ้วมือออกสู่วงแลนเพื่อไปประมวลผล

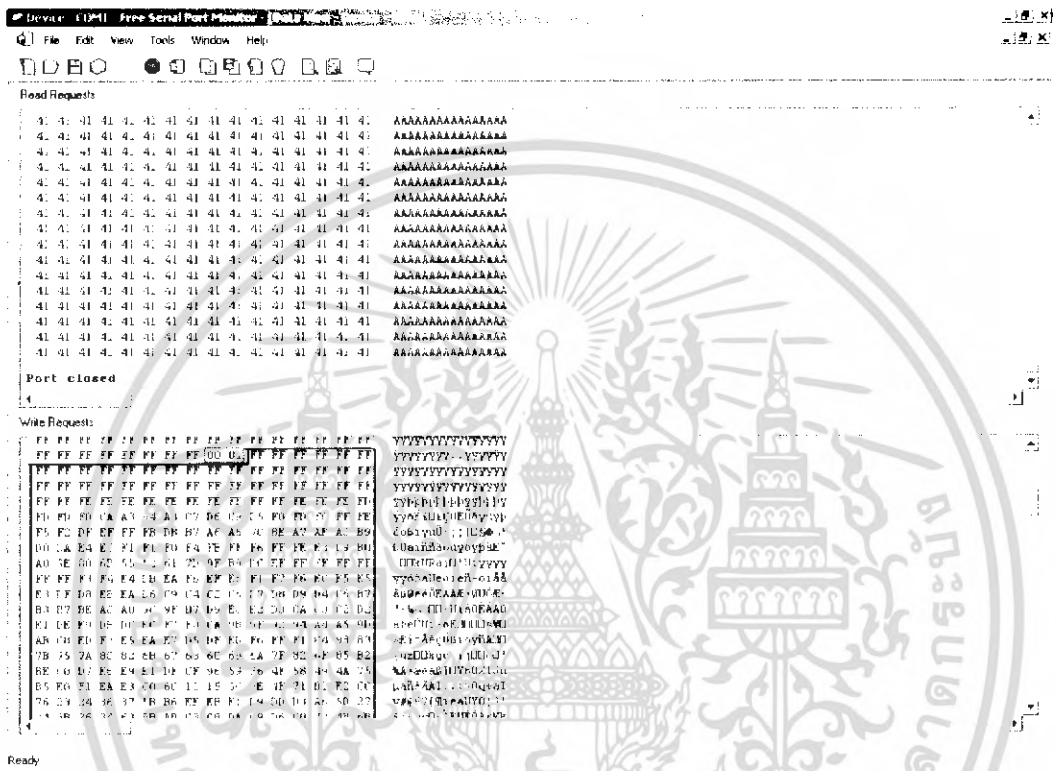
ในการทดลองนี้จะทำการส่งข้อมูลลายนิ้วมือออกสู่วงแลนเพื่อไปประมวลผล โดยส่งภาพออกทางซีเรียลพอร์ตเข้าสู่อีเทอร์เนตฮาร์ดแวร์แล้วไปประมวลผลที่เครื่องคอมพิวเตอร์ศูนย์กลาง แล้วแสดงผลที่คอมพิวเตอร์แล้วส่งค่าการประมวลผลกลับมายังฮาร์ดแวร์

โดยขั้นแรก ทำการเก็บข้อมูลลายนิ้วมือ (Template) ในฐานข้อมูลโดยใส่รายละเอียดต่างๆ เช่น ชื่อ สงไปด้วย ดังรูปที่ 4.13 ซึ่งทำการเก็บข้อมูลของ นายวุฒิชัย ชื่อไฟล์ที่เก็บเทมเพลต คือ 3.DAT และ อนุญาตให้ผ่านเข้าสู่ระบบได้ถ้าผ่านการตรวจสอบ



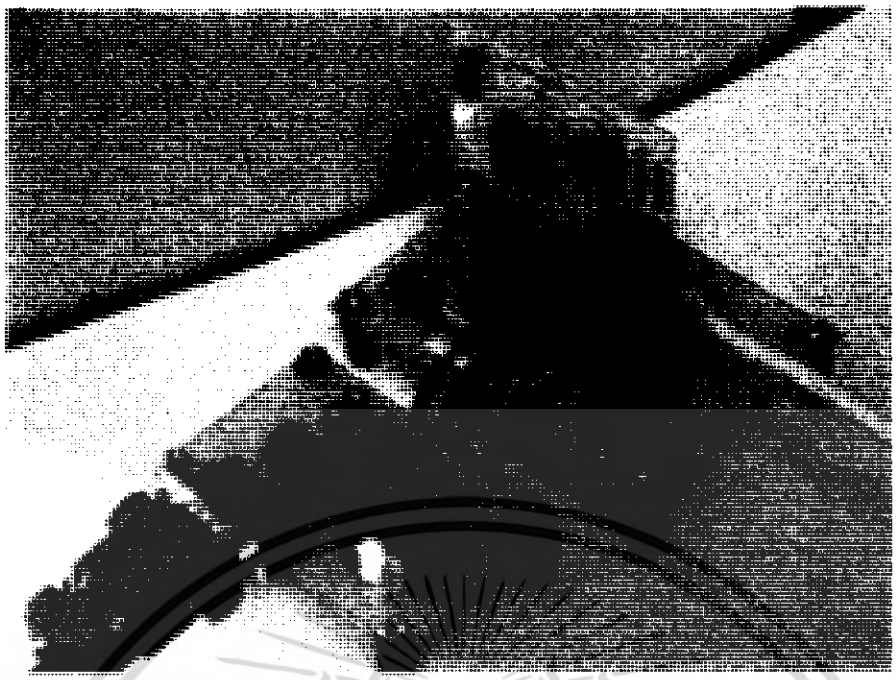
รูปที่ 4.13 แสดงการเก็บรายละเอียดของบุคคลในฐานข้อมูล

ต่อมาจะเป็นการส่งข้อมูลภาพเข้าซีเรียลพอร์ตไปยังอีเทอร์เนตฮาร์ดแวร์ซึ่งแต่ละแถวของรูปจะมีการใส่เฮดเดอร์จำนวน 2 ไบต์ เหตุที่ต้องใส่เฮดเดอร์ เพราะว่าเมื่อส่งข้อมูลจากวงแลนเป็นแบบ UDP ซึ่งเป็นแบบ Connectionless และ Packet Switch เฟรมหลังๆอาจจะไปถึงก่อนเฟรมต้นๆก็ได้ ทำให้เวลารวมภาพในเครื่องคอมพิวเตอร์ศูนย์กลางแล้วเกิดความผิดพลาดได้ จึงต้องใส่เฮดเดอร์ ในรูปที่ 4.14 เป็นรูปที่ Capture ซีเรียลพอร์ตของคอมพิวเตอร์ที่ส่งข้อมูลรูปภาพให้กับอีเทอร์เนตคอนโทรลเลอร์ในกรอบสีฟ้าจะเป็นเฮดเดอร์และสีแดงจะเป็นข้อมูลรูปภาพ



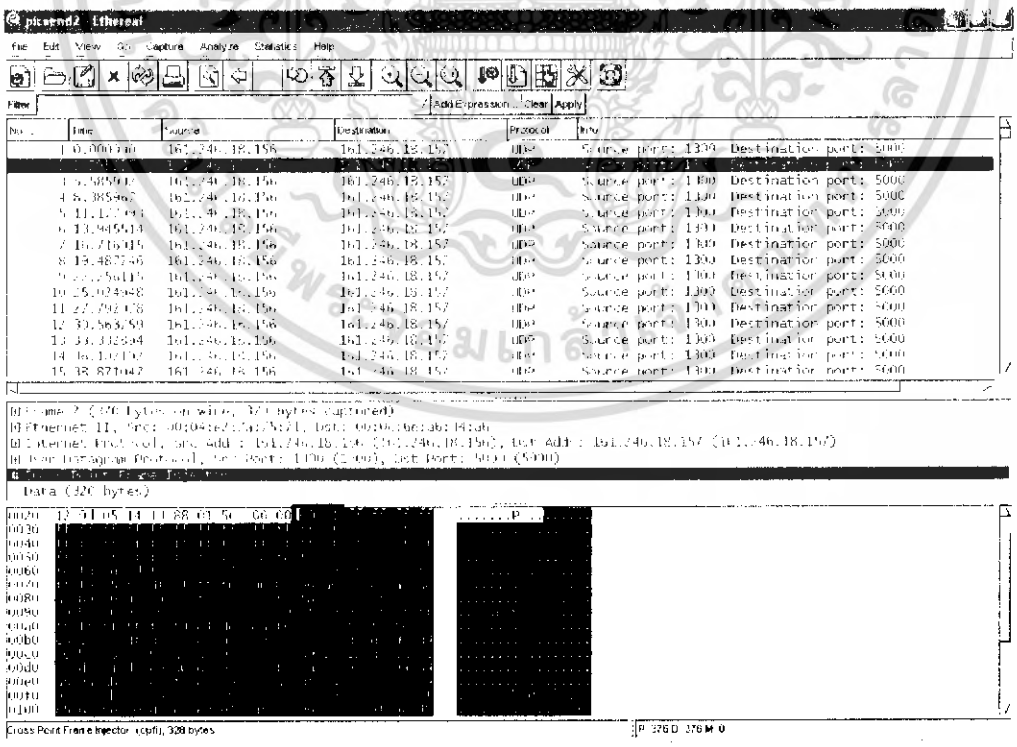
รูปที่ 4.14 แสดงการ Capture ข้อมูลที่ส่งออกให้กับอีเทอร์เนตคอนโทรลเลอร์

ในการทดลองนี้อีเทอร์เนตคอนโทรลเลอร์จะรับอินพุตเข้าที่ซีเรียลพอร์ตและเอาท์พุต จะออกที่ RJ-45 ไปยังวงแลน ดังรูปที่ 4.15



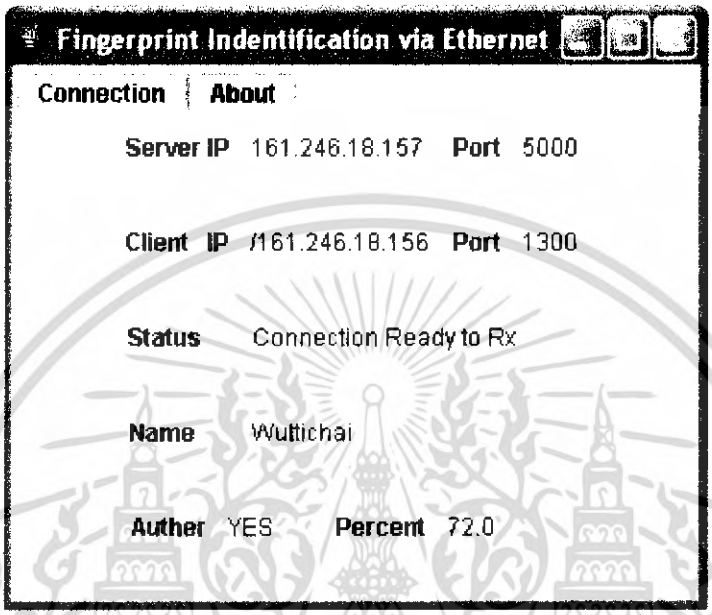
รูปที่ 4.15 แสดงวงจรที่ใช้ในการทดลองซึ่งต่อกับซีเรียลพอร์ตและ RJ-45

หลังจากนั้นอีเทอร์เนตคอนโทรลเลอร์จะส่งข้อมูลภาพออกเป็นเฟรม เฟรมละ 1 แถวของรูปภาพลายนิ้วมือ แล้วคอมพิวเตอร์ศูนย์กลางจะทำการรับเฟรมเหล่านั้น แล้วนำมาเก็บเป็นภาพเพื่อการนำไปประมวลผลต่อไป ซึ่งเฟรมที่ส่งไปนั้นจะเป็น UDP Frame Format ที่มีข้อมูลภาพซึ่งมีเฮดเดอร์ระบุแถวดังรูปที่ 4.16 ข้อมูลที่ถูกวงสีแดงไว้คือ เฮดเดอร์



รูปที่ 4.16 แสดง UDP เฟรมข้อมูลจากอีเทอร์เนตคอนโทรลเลอร์ซึ่ง Capture จากเครื่องคอมพิวเตอร์ศูนย์กลาง

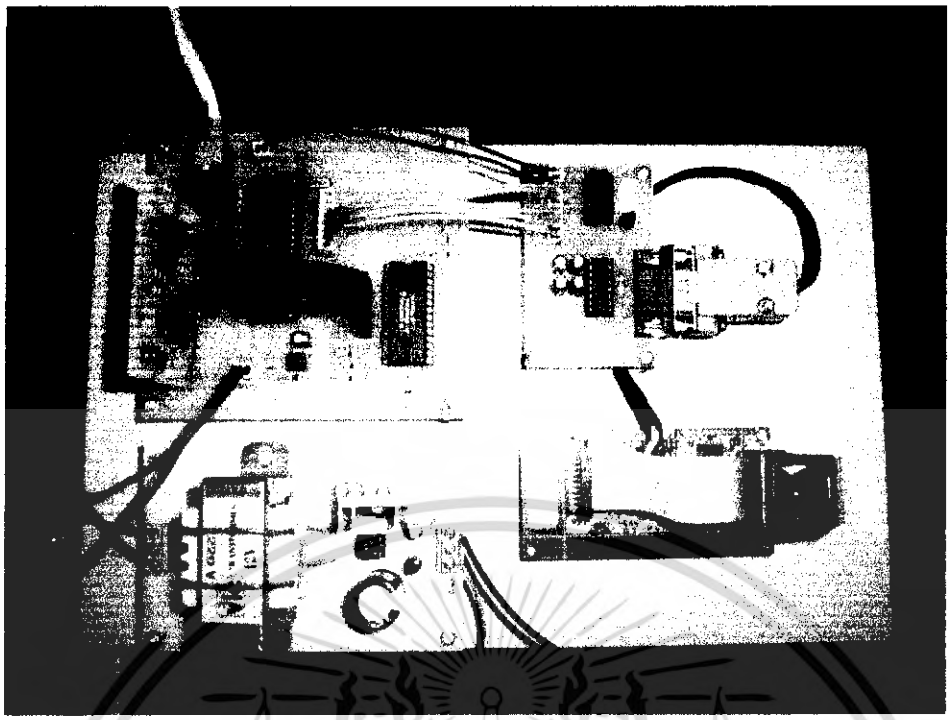
เมื่อเฟรมรูปภาพเข้าไปยังเครื่องคอมพิวเตอร์ศูนย์กลางครบแล้ว จะทำการประมวลผลเพื่อหาจุดเด่นลายนิ้วมือแล้วนำไปเทียบกับฐานข้อมูล เพื่อใช้ในการระบุบุคคล ดังรูปที่ 4.17 แสดงชื่อของเจ้าของลายนิ้วมือ แสดงสิทธิ์ในการเข้าสู่ระบบ และค่าความเหมือนของลายนิ้วมือ ถ้าบุคคลนั้นมีสิทธิ์ในการเข้าสู่ระบบเครื่องคอมพิวเตอร์ศูนย์กลางจะส่งสัญญาณให้อีเทอร์เนตคอนโทรลเลอร์เปิดไฟเขียว แต่ถ้าไม่อนุญาตก็จะส่งสัญญาณให้อีเทอร์เนตคอนโทรลเลอร์เปิดไฟสีแดง



รูปที่ 4.17 แสดงข้อมูลต่างๆของบุคคลที่ถูกระบุจากฐานข้อมูล

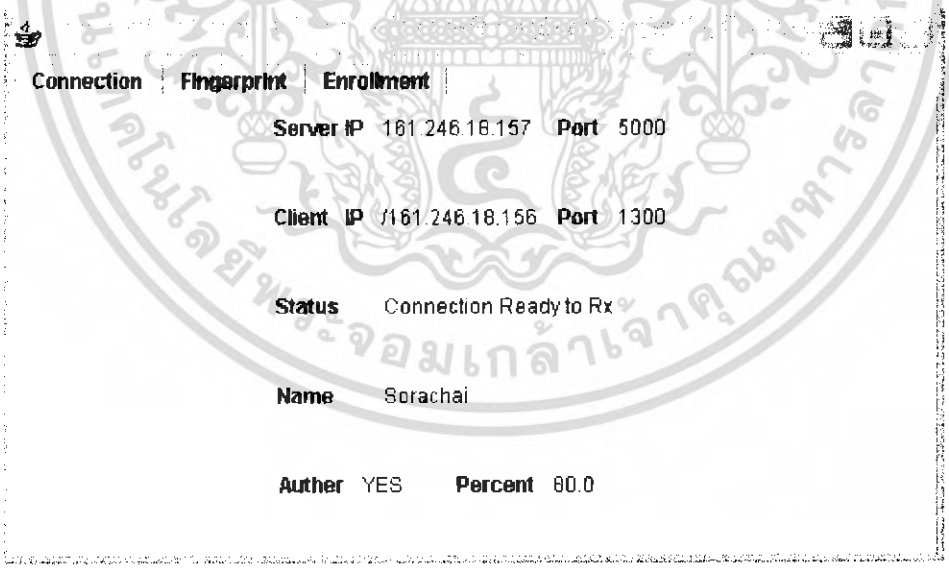
4.4 ผลการทดลองตอนที่ 4 รวมฮาร์ดแวร์โมดูลสแกนลายนิ้วมือเข้ากับระบบ

ในการทดลองตอนนี้จะนำโมดูลสแกนลายนิ้วมือมาต่อเข้ากับระบบผ่านทางซีเรียลพอร์ตแล้วส่งข้อมูลภาพออกไปยังวงแลน แล้วไปประมวลผลที่เครื่องคอมพิวเตอร์ศูนย์กลางเพื่อไปประมวลผลและส่งค่ากลับมายังฮาร์ดแวร์ดังรูปที่ 4.18 แสดงฮาร์ดแวร์ของระบบ



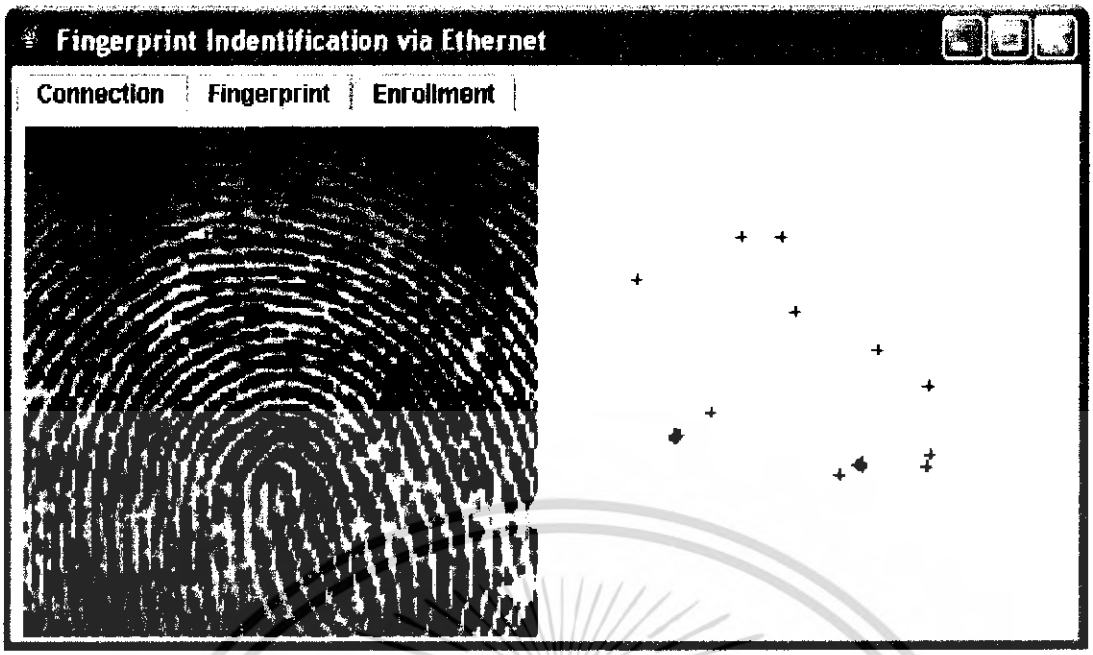
รูปที่ 4.18 แสดงการเชื่อมต่อฮาร์ดแวร์ของระบบทั้งหมด

เมื่อสแกนลายนิ้วมือแล้วข้อมูลภาพจะถูกส่งออกวงแลน ประมวลผลและแสดงผลที่ซอฟต์แวร์ของเครื่องคอมพิวเตอร์ศูนย์กลาง ดังรูปที่ 4.19 แท็ปแรกของซอฟต์แวร์จะแสดงชื่อ สิทธิในการเข้าสู่ระบบ และเปอร์เซ็นต์ความเหมือน เมื่อเทียบกับฐานข้อมูล



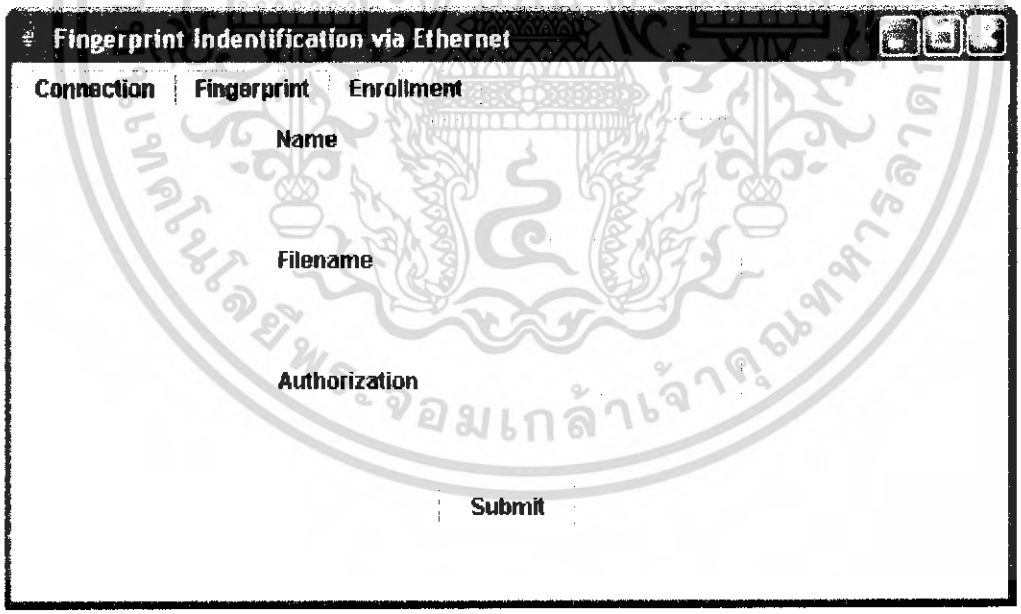
รูปที่ 4.19 แสดงรายละเอียดต่างๆของซอฟต์แวร์ในแท็ปแรก

ในส่วนแท็ปที่สองจะแสดงภาพของลายนิ้วมือที่รับเข้ามาจากโมดูลสแกนลายนิ้วมือ และรูปที่ผ่านกระบวนการปรับปรุงรูปภาพ (Image Enhancement) และจุดเด่นของลายนิ้วมือ (Minutia) ดังรูปที่ 4.20



รูปที่ 4.20 แสดงภาพของลายนิ้วมือในแท็ปที่สองของซอฟต์แวร์

ในส่วนแท็ปที่สาม ซึ่งเป็นแท็ปสุดท้ายมีไว้เพื่อเก็บจุดเด่นลายนิ้วมือ (Enrollment) ที่สแกนครั้งล่าสุดเก็บในฐานข้อมูล ซึ่งจะต้องกรอกชื่อ ไฟล์ที่เก็บจุดเด่นลายนิ้วมือ และสิทธิในการเข้าสู่ระบบ แล้วกดปุ่ม 'Submit' ดังรูปที่ 4.21



รูปที่ 4.21 แสดงส่วนของโปรแกรมที่ใช้ในการเก็บจุดเด่นลายนิ้วมือเข้าสู่ฐานข้อมูล

และหลังจากที่ประมวลผลเสร็จเรียบร้อยแล้วในการทดลองนี้จากรูปที่ 4.19 พบว่าสแกนพบชื่อ 'Sorachai' ซึ่งมีสิทธิที่จะเข้ามาสู่ระบบโปรแกรมจะส่งเฟรมไปยังฮาร์ดแวร์ ทำให้ไฟเขียวติดดังที่ได้แสดงแล้วในรูปที่ 4.18

บทที่ 5
บทวิจารณ์และบทสรุป

จากผลการทดลองในบทที่ 4 พบว่า ถ้าค่า อัตราการปฏิเสธตัวจริง (FRR) และอัตราการยอมรับตัวปลอม (FAR) ยิ่งมีเปอร์เซ็นต์น้อยเท่าไหร่ระบบนั้นระบบก็จะมีประสิทธิภาพในการตรวจสอบลายนิ้วมือมากยิ่งขึ้น

ในส่วนของ ฮาร์ดแวร์ที่ใช้รับและส่งข้อมูลผ่านวงแลน และซอร์ฟแวร์ที่เครื่องคอมพิวเตอร์ศูนย์กลาง มีความสามารถในการรับส่งข้อมูล ตรวจสอบข้อมูลในฐานข้อมูล และส่งผลที่ได้จากการตรวจสอบกับฐานข้อมูลในเครื่องคอมพิวเตอร์ศูนย์กลาง มายังฮาร์ดแวร์เพื่อบอกสถานะของผู้ใช้งานที่จะเข้ามาในระบบ

ในส่วนของโมดูลสแกนลายนิ้วมือที่เพิ่มเข้ามาให้กับระบบ เมื่อเวลาใช้งานจริงๆพบว่าเวลาที่ใช้ในการสแกนมีความล่าช้าอยู่มาก บวกกับเวลาที่ใช้ในการประมวลผลของเครื่องคอมพิวเตอร์ศูนย์กลาง ซึ่งเป็นจุดอ่อนของระบบนี้ อาจจะแก้ไขโดยการปรับปรุงเพิ่ม Baud Rate ของการรับส่งข้อมูลให้เพิ่มมากขึ้น

แนวทางในการพัฒนาต่อโปรเจคนี้ อาจจะพัฒนาในส่วนของอัลกอริทึมในการตรวจสอบลายนิ้วมือให้ดีขึ้น หรือจะพัฒนาเครื่องคอมพิวเตอร์ศูนย์กลางให้รับฮาร์ดแวร์ด้านลูกข่ายให้มากขึ้น หรืออาจจะพัฒนาเป็น Web Application ฝากไว้ที่โฮสและทางด้านฮาร์ดแวร์จะต้องพัฒนาให้ส่งข้อมูลผ่านอินเตอร์เนตได้

ภาคผนวก

1. Source Code ของฮาร์ดแวร์

Main.c

```
#include <REG52.H>
#include <absacc.H>
#include <string.h>
#include <io.h>
#include <initial.h>
#include <packet.h>
#include <recieve.h>
#include <FCP301.h>
unsigned char sbuffer,ACK;
unsigned char checkbit;
unsigned int count,row;

void uart_init(void)
{
       PCON=0x00;
       SCON=0x50;
       TMOD=0x20;
       TH1=0xFD;
       TR1=1;
       RI=0;
       TI=0;
}
void main()
{
              uart_init();
              count = 0;
              row = 0;
              Green = OFF;
              Yellow = OFF;
              Red = OFF;
              initial();

              txicmp();

              checkbit = 0;
              while(checkbit == 0)
              {
                     rxpacket();
                     // For recieve ARP Request and send ARP reply

       if((RxBuffer[13]==0x08)&&(RxBuffer[14]==0x06)&&(RxBuffer[39]==0xa1)

       &&(RxBuffer[40]==0xf6)&&(RxBuffer[41]==0x12)&&(RxBuffer[42]==0x9c))
                     {
                            txarp();
                     }
                     // For recieve ICMP Echo form and passing while loop

       if((RxBuffer[13]==0x08)&&(RxBuffer[14]==0x00)&&(RxBuffer[24]==0x01)

       &&(RxBuffer[27]==0xa1)&&(RxBuffer[28]==0xf6)&&(RxBuffer[29]==0x12)

       &&(RxBuffer[30]==0x9d)&&(RxBuffer[35]==0x00)&&(RxBuffer[36]==0x00)

       &&(RxBuffer[41]==0x00)&&(RxBuffer[42]==0x01)&&(RxBuffer[43]==0x01))
                     {
                            checkbit = 1;
                            Yellow = ON;
                     }
              }
              txudp();
              ioWrite(RxTxData+1,'x');
              FCP301();
              CheckPacket();
```

```
            while(1){}
}
```

io.h

```
#define addr P2
#define data P0
#define Dbit P3

#define RxTxData    0xF0        // Receive/Transmit data (port 0)
#define RxTxData1   0xF2        // Receive/Transmit data (port 1)
#define TxCmd       0xF4        // Transmit Command
#define TxLength    0xF6        // Transmit Length
#define ISQ         0xF8         // Interrupt status queue
#define PPPtr       0xFA        // PacketPage pointer
#define PPData      0xFC        // PacketPage data (port 0)
#define PPData1     0xFE        // PacketPage data (port 1)

sbit write   = P2^7;            // IOW active low
sbit read    = P2^6;            // IOR active low

//------DELAY1ms-------//
      void DELAY1ms(unsigned char round)
      {
            unsigned char X;
                  TMOD = (TMOD|0x01);
                  IE     = (IE|0x80);

            for (X=0; X<round; X++)
            {
                  TH0 = 0xFC;
                  TL0 = 0x66;
                  TF0 = 0;
                  TR0 = 1;
                  while(TF0 == 0);
                  TR0 = 0;
            }
      }
//--------------------//

//--Function ioRead Used to Read from the Data Bus--//
      unsigned char ioRead(unsigned char address)
      {
            unsigned char value;
            data  = 0xFF;              // Set P0 = 0xFF for recieve data
            addr  = (address&0xFF);
            addr  = (address&0xBF);    // read = 0 Active ioread
            value = data;              // Keep data to register
            addr  = (address&0xF0);    // read = 1 Deactive ioread
            return value;              // Return to ioread


      }
//--Funtcion ioWrite Used to write to the Data Bus--//
      void ioWrite(unsigned char address, unsigned char value)
      {
            data  = value;
            addr  = (address&0xFF);
            addr  = (address&0x7F);    // write = 0 Active iowrite
            addr  = (address&0xF0);    // write = 1 Deactive iowrite

      }
```

initial.h

```
#define addr P2
#define data P0
#define Dbit P3

#define RxTxData    0xF0        // Receive/Transmit data (port 0)
```

```
#define RxTxData1   0xF2          // Receive/Transmit data (port 1)
#define TxCmd       0xF4          // Transmit Command
#define TxLength    0xF6          // Transmit Length
#define ISQ         0xF8          // Interrupt status queue
#define PPPtr       0xFA          // PacketPage pointer
#define PPData      0xFC          // PacketPage data (port 0)
#define PPData1     0xFE          // PacketPage data (port 1)

sbit write    = P2^7;             // IOW active low
sbit read     = P2^6;             // IOR active low

//------DELAY1ms------//
      void DELAY1ms(unsigned char round)
      {
            unsigned char X;
                  TMOD = (TMOD|0x01);
                  IE      = (IE|0x80);

            for (X=0; X<round; X++)
            {
                  TH0 = 0xFC;
                  TL0 = 0x66;
                  TF0 = 0;
                  TR0 = 1;
                  while(TF0 == 0);
                  TR0 = 0;
            }
      }
//--------------------//

//--Function ioRead Used to Read from the Data Bus--//
      unsigned char ioRead(unsigned char address)
      {
            unsigned char value;
            data  = 0xFF;               // Set P0 = 0xFF for recieve data
            addr  = (address&0xFF);
            addr  = (address&0xBF);     // read = 0 Active ioread
            value = data;               // Keep data to register
            addr  = (address&0xF0);     // read = 1 Deactive ioread
            return value;               // Return to ioread


      }
//--Funtcion ioWrite Used to write to the Data Bus--//
      void ioWrite(unsigned char address, unsigned char value)
      {
            data  = value;
            addr  = (address&0xFF);
            addr  = (address&0x7F);     // write = 0 Active iowrite
            addr  = (address&0xF0);     // write = 1 Deactive iowrite

      }
```

packet.h

```
unsigned char BusST0,BusST1;
unsigned int Status,Length;
unsigned char idata RxBuffer[160];
unsigned char DataL,DataH,i,j,k,l,buff,temp;
unsigned int x,y,z;
void txicmp()
{
      ////////////////////////
      // Transmit  Datagram //
      ////////////////////////


      // Send Transmit Command (Data Sheet P.70 Setbit 7,6)
            ioWrite(TxCmd,     0xc0);
```

```
        ioWrite(TxCmd + 1,  0x00);

    // 64 bytes to be sent
    // Ethernet Header(14) + IP Header(20) + ICMP Header(8) + Data(22)
        ioWrite(TxLength,   0x40);
        ioWrite(TxLength+1, 0x00);

    // Check Bus status (Data Sheet P.67)
        ioWrite(PPPtr,     0x38);
        ioWrite(PPPtr + 1, 0x01);

    // Check Bus status,Is ready for transmit (Check bit 7)
        do {
            BusST0 = ioRead(PPData);
            BusST1 = ioRead(PPData+1);
        }while(!(BusST1==0x01));

    //--Ready to Transmit,Transmit Datagram in RxTxData Port--//

    ////////////////////////
    //  Ethernet Header  // (14 bytes)
    ////////////////////////

    // Send Destination (6 bytes)
    // MAC Address Microcomputer Top (00-08-02-f4-f8-5b) Loon(00-0c-6e-a6-34-
a6)
        ioWrite(RxTxData,   0x00);
        ioWrite(RxTxData+1, 0x0c);
        ioWrite(RxTxData,   0x6e);
        ioWrite(RxTxData+1, 0xa6);
        ioWrite(RxTxData,   0x34);
        ioWrite(RxTxData+1, 0xa6);
    // Send Source (6 bytes)
    // MAC Address Embedded Device (00-04-e2-7a-75-71)
        ioWrite(RxTxData,   0x00);
        ioWrite(RxTxData+1, 0x04);
        ioWrite(RxTxData,   0xe2);
        ioWrite(RxTxData+1, 0x7a);
        ioWrite(RxTxData,   0x75);
        ioWrite(RxTxData+1, 0x71);

    // Send Protocol Type (2 bytes)
    // IP=0x0800
        ioWrite(RxTxData,   0x08);
        ioWrite(RxTxData+1, 0x00);

    /////////////////
    //  IP Header  // (20 bytes)
    /////////////////

    // Version Header Length (1 byte)
    // Using IPv4
        ioWrite(RxTxData,   0x45);

    // Service (1 byte)
    // Normally set to zero because not used
        ioWrite(RxTxData+1, 0x00);

    // Length (2 bytes)
    // 50 byte Length (IP Header(20) + ICMP Header(8) + Data(22))
        ioWrite(RxTxData,   0x00);
        ioWrite(RxTxData+1, 0x32);

    // Identifier (2 bytes)
    // 0x0000
        ioWrite(RxTxData,   0x00);
        ioWrite(RxTxData+1, 0x00);
```

```
// Flags Fragment offset (2 bytes)(no flag)
// 0x0000
      ioWrite(RxTxData,   0x00);
      ioWrite(RxTxData+1, 0x00);

// Time to Live (1 byte)
// 63
      ioWrite(RxTxData,   0x3f);

// Protocol (1 byte)
// ICMP 1 = 0x01
      ioWrite(RxTxData+1, 0x01);

// Checksum IP Header (2 bytes)
// 0x12a6
      ioWrite(RxTxData,   0x12);
      ioWrite(RxTxData+1, 0xa6);

// Source Address (4 bytes)
// IP address 161.246.18.156 (Embedded Device)
      ioWrite(RxTxData,   0xa1);    // 161
      ioWrite(RxTxData+1, 0xf6);    // 246
      ioWrite(RxTxData,   0x12);    // 18
      ioWrite(RxTxData+1, 0x9c);    // 156

// Destination Address (4 bytes)
// IP address 161.246.18.157 (Labtop)
      ioWrite(RxTxData,   0xa1); // 161
      ioWrite(RxTxData+1, 0xf6); // 246
      ioWrite(RxTxData,   0x12); // 18
      ioWrite(RxTxData+1, 0x9d); // 157

//////////////////
//  ICMP Header// (8 bytes)
//////////////////

// Type (1 bytes)
// 0x08
      ioWrite(RxTxData,   0x08);

// Code (1 byte)
// 0x00
      ioWrite(RxTxData+1, 0x00);

// Checksum (2 bytes)
// 0xECF3
      ioWrite(RxTxData,   0xec);
      ioWrite(RxTxData+1, 0xf3);

// Identifier (2 bytes)
// 0x0000
      ioWrite(RxTxData,   0x00);
      ioWrite(RxTxData+1, 0x00);

// Sequence (2 bytes)
// 0x0001
      ioWrite(RxTxData,   0x00);
      ioWrite(RxTxData+1, 0x01);

//////////////
//  Data  // (22 bytes)
//////////////

// Write Message (22 bytes)
      ioWrite(RxTxData,   0x01);
      ioWrite(RxTxData+1, 0x01);
      ioWrite(RxTxData,   0x01);
      ioWrite(RxTxData+1, 0x01);
```

```
            ioWrite(RxTxData,   0x01);
            ioWrite(RxTxData+1, 0x01);
            ioWrite(RxTxData,   0x01);
            ioWrite(RxTxData+1, 0x01);
            ioWrite(RxTxData,   0x01);
            ioWrite(RxTxData+1, 0x01);
            ioWrite(RxTxData,   0x01);
            ioWrite(RxTxData+1, 0x01);
            ioWrite(RxTxData,   0x01);
            ioWrite(RxTxData+1, 0x01);
            ioWrite(RxTxData,   0x01);
            ioWrite(RxTxData+1, 0x01);
            ioWrite(RxTxData,   0x01);
            ioWrite(RxTxData+1, 0x01);
            ioWrite(RxTxData,   0x01);
            ioWrite(RxTxData+1, 0x01);
            ioWrite(RxTxData,   0x01);
            ioWrite(RxTxData+1, 0x01);

}

void txudp()
{
       ////////////////////////
       // Transmit  Datagram //
       ////////////////////////


       // Send Transmit Command (Data Sheet P.70 Setbit 7,6)
            ioWrite(TxCmd,      0xc0);
            ioWrite(TxCmd + 1,  0x00);

       // 64 bytes to be sent
       // Ethernet Header(14) + IP Header(20) + UDP Header(8) + Data(22)
            ioWrite(TxLength,   0x40);
            ioWrite(TxLength+1, 0x00);

       // Check Bus status (Data Sheet P.67)
            ioWrite(PPPtr,      0x38);
            ioWrite(PPPtr + 1, 0x01);

       // Check Bus status,Is ready for transmit (Check bit 7)
            do {
                   BusST0 = ioRead(PPData);
                   BusST1 = ioRead(PPData+1);
            }while(!(BusST1==0x01));

       //--Ready to Transmit,Transmit Datagram in RxTxData Port--//

       ///////////////////////
       //  Ethernet Header  // (14 bytes)
       ///////////////////////

       // Send Destination (6 bytes)
       // MAC Address Microcomputer Top (00-08-02-f4-f8-5b) Loon(00-0c-6e-a6-34-
a6)
            ioWrite(RxTxData,   0x00);
            ioWrite(RxTxData+1, 0x0c);
            ioWrite(RxTxData,   0x6e);
            ioWrite(RxTxData+1, 0xa6);
            ioWrite(RxTxData,   0x34);
            ioWrite(RxTxData+1, 0xa6);

       // Send Source (6 bytes)
       // MAC Address Embedded Device (00-04-e2-7a-75-71)
            ioWrite(RxTxData,   0x00);
            ioWrite(RxTxData+1, 0x04);
```

```
        ioWrite(RxTxData,   0xe2);
        ioWrite(RxTxData+1, 0x7a);
        ioWrite(RxTxData,   0x75);
        ioWrite(RxTxData+1, 0x71);

// Send Protocol Type (2 bytes)
// IP=0x0800
        ioWrite(RxTxData,   0x08);
        ioWrite(RxTxData+1, 0x00);


//////////////////
//  IP Header  // (20 bytes)
//////////////////

// Version Header Length (1 byte)
// Using IPv4
        ioWrite(RxTxData,   0x45);

// Service (1 byte)
// Normally set to zero because not used
        ioWrite(RxTxData+1, 0x00);

// Length (2 bytes)
// 50 byte Length (IP Header(20) + UDP Header(8) + Data(22))
        ioWrite(RxTxData,   0x00);
        ioWrite(RxTxData+1, 0x32);

// Identifier (2 bytes)
// 0x0000
        ioWrite(RxTxData,   0x00);
        ioWrite(RxTxData+1, 0x00);

// Flags Fragment offset (2 bytes)(no flag)
// 0x4000
        ioWrite(RxTxData,   0x40);
        ioWrite(RxTxData+1, 0x00);

// Time to Live (1 byte)
// 63
        ioWrite(RxTxData,   0x3f);

// Protocol (1 byte)
// UDP 17 = 0x11
        ioWrite(RxTxData+1, 0x11);

// Checksum IP Header (2 bytes)
// 0xd295
        ioWrite(RxTxData,   0xd2);
        ioWrite(RxTxData+1, 0x95);

// Source Address (4 bytes)
// IP address 161.246.18.156 (Embedded Device)
        ioWrite(RxTxData,   0xa1);    // 161
        ioWrite(RxTxData+1, 0xf6);    // 246
        ioWrite(RxTxData,   0x12);    // 18
        ioWrite(RxTxData+1, 0x9c);    // 156

// Destination Address (4 bytes)
// IP address 161.246.18.157 (Labtop)
        ioWrite(RxTxData,   0xa1); // 161
        ioWrite(RxTxData+1, 0xf6); // 246
        ioWrite(RxTxData,   0x12); // 18
        ioWrite(RxTxData+1, 0x9d); // 157

//////////////////
//  UDP Header // (8 bytes)
//////////////////
```

```
        // Source Port (2 bytes)
        // Port 1300 (0514)
              ioWrite(RxTxData,   0x05);
              ioWrite(RxTxData+1, 0x14);

        // Destination Port (2 bytes)
        // Port 5000 (0x1388)
              ioWrite(RxTxData,   0x13);
              ioWrite(RxTxData+1, 0x88);

        // Message Length (2 bytes)
        // 30 bytes (UDP Header(8) + Data(22))
              ioWrite(RxTxData,   0x00);
              ioWrite(RxTxData+1, 0x1e);

        // Checksum (2 bytes)
        // 0x0000 (no checksum used)
              ioWrite(RxTxData,   0x00);
              ioWrite(RxTxData+1, 0x00);

        ////////////
        //  Data  // (22 bytes)
        ////////////

        // Write Message (22 bytes)
              ioWrite(RxTxData,'C');
              ioWrite(RxTxData+1,'o');
              ioWrite(RxTxData,'n');
              ioWrite(RxTxData+1,'n');
              ioWrite(RxTxData,'e');
              ioWrite(RxTxData+1,'c');
              ioWrite(RxTxData,'t');
              ioWrite(RxTxData+1,'i');
              ioWrite(RxTxData,'o');
              ioWrite(RxTxData+1,'n');
              ioWrite(RxTxData,' ');
              ioWrite(RxTxData+1,'R');
              ioWrite(RxTxData,'e');
              ioWrite(RxTxData+1,'a');
              ioWrite(RxTxData,'d');
              ioWrite(RxTxData+1,'y');
              ioWrite(RxTxData,' ');
              ioWrite(RxTxData+1,'t');
              ioWrite(RxTxData,'o');
              ioWrite(RxTxData+1,' ');
              ioWrite(RxTxData,'R');
              //Addition out Function ioWrite(RxTxData+1,'x');
}


void txarp()
{
        ////////////////////////
        // Transmit  Datagram //
        ////////////////////////


        // Send Transmit Command (Data Sheet P.70 Setbit 7,6)
              ioWrite(TxCmd,      0xc0);
              ioWrite(TxCmd + 1,  0x00);

        // 42 bytes to be sent
        // Ethernet Header(14) + IP Header(20) + UDP Header(8) + Data(2)
              ioWrite(TxLength,   0x2a);
              ioWrite(TxLength+1, 0x00);

        // Check Bus status (Data Sheet P.67)
```

```
        ioWrite(PPPtr,     0x38);
        ioWrite(PPPtr + 1, 0x01);

    // Check Bus status,Is ready for transmit (Check bit 7)
        do {
            BusST0 = ioRead(PPData);
            BusST1 = ioRead(PPData+1);
        }while(!(BusST1==0x01));

    //--Ready to Transmit,Transmit Datagram in RxTxData Port--//

    ////////////////////////
    //  Ethernet Header  // (14 bytes)
    ////////////////////////

    // Send Destination (6 bytes)
    // MAC Address Microcomputer Top (00-08-02-f4-f8-5b) Loon(00-0c-6e-a6-34-
a6)
        ioWrite(RxTxData,   0x00);
        ioWrite(RxTxData+1, 0x0c);
        ioWrite(RxTxData,   0x6e);
        ioWrite(RxTxData+1, 0xa6);
        ioWrite(RxTxData,   0x34);
        ioWrite(RxTxData+1, 0xa6);

    // Send Source (6 bytes)
    // MAC Address Embedded Device (00-04-e2-7a-75-71)
        ioWrite(RxTxData,   0x00);
        ioWrite(RxTxData+1, 0x04);
        ioWrite(RxTxData,   0xe2);
        ioWrite(RxTxData+1, 0x7a);
        ioWrite(RxTxData,   0x75);
        ioWrite(RxTxData+1, 0x71);

    // Send Protocol Type (2 bytes)
    // ARP=0x0806
        ioWrite(RxTxData,   0x08);
        ioWrite(RxTxData+1, 0x06);


    ////////////////
    //  ARP Header // (28 bytes)
    ////////////////

    // Hardware Type (2 byte)
    // Ethernet
        ioWrite(RxTxData,   0x00);
        ioWrite(RxTxData+1, 0x01);

    // Protocol type (2 bytes)
        ioWrite(RxTxData,   0x08);
        ioWrite(RxTxData+1, 0x00);

    // Hardware size (1 bytes)
        ioWrite(RxTxData,   0x06);
    //Protocol size (1 byte)
        ioWrite(RxTxData+1, 0x04);

    // Opcode (2 bytes)
    // reply 02
        ioWrite(RxTxData,   0x00);
        ioWrite(RxTxData+1, 0x02);
    //sender MAC hardware (6bytes) (Embedded Device 00-04-e2-7a-75-71)
        ioWrite(RxTxData,   0x00);
        ioWrite(RxTxData+1, 0x04);
        ioWrite(RxTxData,   0xe2);
        ioWrite(RxTxData+1, 0x7a);
        ioWrite(RxTxData,   0x75);
```

```
            ioWrite(RxTxData+1, 0x71);
      // Source Address (4 bytes)
      // IP address 161.246.18.156 (Embedded Device)
            ioWrite(RxTxData,   0xa1);   // 161
            ioWrite(RxTxData+1, 0xf6);   // 246
            ioWrite(RxTxData,   0x12);   // 18
            ioWrite(RxTxData+1, 0x9c);   // 156
      //Dest MAC computer (6bytes) Top (00-08-02-f4-f8-5b) Loon(00-0c-6e-a6-34-
a6)
            ioWrite(RxTxData,   0x00);
            ioWrite(RxTxData+1, 0x0c);
            ioWrite(RxTxData,   0x6e);
            ioWrite(RxTxData+1, 0xa6);
            ioWrite(RxTxData,   0x34);
            ioWrite(RxTxData+1, 0xa6);
      // Destination Address (4 bytes)
      // IP address 161.246.18.157 (Labtop)
            ioWrite(RxTxData,   0xa1); // 161
            ioWrite(RxTxData+1, 0xf6); // 246
            ioWrite(RxTxData,   0x12); // 18
            ioWrite(RxTxData+1, 0x9d); // 157
}
```

receive.h

```
#define ON 1
#define OFF 0
sbit Green  = P3^2;
sbit Yellow = P3^3;
sbit Red    = P3^4;
unsigned char passloop;
void rxpacket()
{
      do {
      ioWrite(PPPtr,     0x24);
      ioWrite(PPPtr + 1, 0x01);
      BusST0 = ioRead(PPData);
      BusST1 = ioRead(PPData+1);
      }while((BusST0==0x04)&(BusST1==0x00));
      // frame is present //
      ioWrite(PPPtr,     0x02);
      ioWrite(PPPtr + 1, 0x04);
      DataL=ioRead(PPData);
      DataH=ioRead(PPData+1);
      x=DataL;
      y=DataH;
      y=y<<8;
      Length=x+y;
      //Protect Loop more than Buffer Array
      if (Length>159)
            Length=159;
      j=0;
      k=0;
      for (i=1;i<=Length;i++)
      {
            ioWrite(PPPtr,     0x04 + j);
            ioWrite(PPPtr + 1, 0x04);
            RxBuffer[i]=ioRead(PPData + k);
                  if(i%2==1) k=1;
                        else {
                        j=j+2;
                        k=0;
                        }
      }

}
void CheckPacket()
{
            passloop = 0;
```

```
            while(passloop == 0)
            {
                  rxpacket();

      if((RxBuffer[13]==0x08)&&(RxBuffer[14]==0x00)&&(RxBuffer[24]==0x11)
      //161.246.18.157
      &&(RxBuffer[27]==0xa1)&&(RxBuffer[28]==0xf6)&&(RxBuffer[29]==0x12)
      &&(RxBuffer[30]==0x9d)
      //161.246.18.156
      &&(RxBuffer[31]==0xa1)&&(RxBuffer[32]==0xf6)&&(RxBuffer[33]==0x12)
      &&(RxBuffer[34]==0x9c)
      //Src port 5000 Dest Port 1300
      &&(RxBuffer[35]==0x13)&&(RxBuffer[36]==0x88)&&(RxBuffer[37]==0x05)
      &&(RxBuffer[38]==0x14))
            {
                  if(RxBuffer[43]==0x31) {
                        Green = ON;
                        passloop = 1;
                              }
                  if (RxBuffer[43]==0x30) {
                  Red = ON;
                  passloop = 1;
                        }
                        }
            }
}
```

extram.h

```
#define io P1
sbit A8 = P3^6;
sbit A9 = P3^7;
sbit rd = P2^4;
sbit wr = P2^5;
void memwr(unsigned int add,unsigned char dat)
{
      unsigned char addrhigh,addrlow;
      addr = 0xF0;
      addrlow = add & 0x00FF;
      addrhigh = (add & 0xFF00)>>8;
      if (addrhigh==0x00) {A8=0;A9=0;}
      if (addrhigh==0x01) {A8=1;A9=0;}
      if (addrhigh==0x02) {A8=0;A9=1;}
      if (addrhigh==0x03) {A8=1;A9=1;}
      io = dat;
      data = addrlow;
      addr = 0xD0;
      addr = 0xF0;
}
unsigned char memrd(unsigned int add)
{
      unsigned char addrhigh,addrlow;
      addr = 0xF0;
      addrlow = add & 0x00FF;
      addrhigh = (add & 0xFF00)>>8;
      if (addrhigh==0x00) {A8=0;A9=0;}
      if (addrhigh==0x01) {A8=1;A9=0;}
      if (addrhigh==0x02) {A8=0;A9=1;}
      if (addrhigh==0x03) {A8=1;A9=1;}
      io = 0xFF;
      data = addrlow;
      addr = 0xE0;
      return(io);
      addr = 0xF0;
}
```

FCP301.h

```
#include <extram.h>
unsigned char framebuff[14];
```

```
unsigned char frameRXbuff[14];
unsigned char Rxloop=0;
unsigned char temp;
void TXframe()
{
	for (i=1;i<(framebuff[0]+1);i++)
	{
		if (TI==0)
		{
			SBUF=framebuff[i];
			while(TI==0){}
			TI=0;
		}
	}
}
void RXframe()
{
	i=1;
	while(i<Rxloop+1)
	{
		Yellow = ON;
		while(RI==0){}
			frameRXbuff[i] = SBUF;
			i++;
			RI=0;
			Yellow = OFF;
	}
}
void SetBuad()
{
	//length 8 bytes
	//header
	framebuff[0]=0x08;
	//data 02 00 D4 00 01 00 04 DB
	framebuff[1]=0x02;
	framebuff[2]=0x00;
	framebuff[3]=0xD4;
	framebuff[4]=0x00;
	framebuff[5]=0x01;
	framebuff[6]=0x00;
	framebuff[7]=0x04;
	framebuff[8]=0xDB;
	Rxloop=9;
}
void Connect()
{
	//length 7 bytes
	//header
	framebuff[0]=0x07;
	//data 02 00 07 00 00 04 0D
	framebuff[1]=0x02;
	framebuff[2]=0x00;
	framebuff[3]=0x07;
	framebuff[4]=0x00;
	framebuff[5]=0x00;
	framebuff[6]=0x04;
	framebuff[7]=0x0D;
	Rxloop=9;
}
void CreateCaptureHandle()
{
	//length 7 bytes
	//header
	framebuff[0]=0x07;
	//data 02 00 38 00 00 04 3E
	framebuff[1]=0x02;
	framebuff[2]=0x00;
	framebuff[3]=0x38;
```

```
        framebuff[4]=0x00;
        framebuff[5]=0x00;
        framebuff[6]=0x04;
        framebuff[7]=0x3E;
        Rxloop=9;
}
void Capture()
{
        //length 7 bytes
        //header
        framebuff[0]=0x07;
        //data 02 00 36 00 00 04 3C
        framebuff[1]=0x02;
        framebuff[2]=0x00;
        framebuff[3]=0x36;
        framebuff[4]=0x00;
        framebuff[5]=0x00;
        framebuff[6]=0x04;
        framebuff[7]=0x3C;
        Rxloop=9;
}
void GetImg()
{
        //length 9 bytes
        //header
        framebuff[0]=0x09;
        //data 02 00 22 00 02 08 0B 04 3D
        framebuff[1]=0x02;
        framebuff[2]=0x00;
        framebuff[3]=0x22;
        framebuff[4]=0x00;
        framebuff[5]=0x02;
        framebuff[6]=0x08;
        framebuff[7]=0x0B;
        framebuff[8]=0x04;
        framebuff[9]=0x3D;
        Rxloop=13;
}
void ContImg()
{
        //length 7 bytes
        //header
        framebuff[0]=0x07;
        //data 02 00 24 00 00 04 2A
        framebuff[1]=0x02;
        framebuff[2]=0x00;
        framebuff[3]=0x24;
        framebuff[4]=0x00;
        framebuff[5]=0x00;
        framebuff[6]=0x04;
        framebuff[7]=0x2A;
}
void CompImg()
{
        //length 7 bytes
        //header
        framebuff[0]=0x07;
        //data 02 00 68 00 00 04 6E
        framebuff[1]=0x02;
        framebuff[2]=0x00;
        framebuff[3]=0x68;
        framebuff[4]=0x00;
        framebuff[5]=0x00;
        framebuff[6]=0x04;
        framebuff[7]=0x6E;
        Rxloop=7;
}
void DisConnect()
```

```
{
	//length 7 bytes
	//header
	framebuff[0]=0x07;
	//data 02 00 09 00 00 04 0F
	framebuff[1]=0x02;
	framebuff[2]=0x00;
	framebuff[3]=0x09;
	framebuff[4]=0x00;
	framebuff[5]=0x00;
	framebuff[6]=0x04;
	framebuff[7]=0x0F;
	Rxloop=7;
}
void TxPic()
{
		//Setup Frame And Put That Byte to Embedded memory
			// Send Transmit Command (Data Sheet P.70 Setbit 7,6)
				ioWrite(TxCmd,	0xc0);
				ioWrite(TxCmd + 1,  0x00);

			// 171 bytes to be sent
			// Ethernet Header(14) + IP Header(20) + UDP Header(8) +
Data(129)
				ioWrite(TxLength,   0xAB);
				ioWrite(TxLength+1, 0x00);

			// Check Bus status (Data Sheet P.67)
				ioWrite(PPPtr,	0x38);
				ioWrite(PPPtr + 1, 0x01);

			// Check Bus status,Is ready for transmit (Check bit 7)
				do {
					BusST0 = ioRead(PPData);
					BusST1 = ioRead(PPData+1);
				}while(!(BusST1==0x01));

			//--Ready to Transmit,Transmit Datagram in RxTxData Port--//
			// Ethernet Header (14 bytes)
			// Send Destination (6 bytes)
			// MAC Address Microcomputer Loon(00-0c-6e-a6-34-a6)
				ioWrite(RxTxData,   0x00);
				ioWrite(RxTxData+1, 0x0c);
				ioWrite(RxTxData,   0x6e);
				ioWrite(RxTxData+1, 0xa6);
				ioWrite(RxTxData,   0x34);
				ioWrite(RxTxData+1, 0xa6);

			// Send Source (6 bytes)
			// MAC Address Embedded Device (00-04-e2-7a-75-71)
				ioWrite(RxTxData,   0x00);
				ioWrite(RxTxData+1, 0x04);
				ioWrite(RxTxData,   0xe2);
				ioWrite(RxTxData+1, 0x7a);
				ioWrite(RxTxData,   0x75);
				ioWrite(RxTxData+1, 0x71);

			// Send Protocol Type (2 bytes)
			// IP=0x0800
				ioWrite(RxTxData,   0x08);
				ioWrite(RxTxData+1, 0x00);

			// IP Header (20 bytes)
			// Version Header Length (1 byte)
			// Using IPv4
				ioWrite(RxTxData,   0x45);

			// Service (1 byte)
```

```
                    // Normally set to zero because not used
                        ioWrite(RxTxData+1, 0x00);

                    // Length (2 bytes)
                    // 157 byte Length (IP Header(20) + UDP Header(8) +
Data(129))
                        ioWrite(RxTxData,   0x00);
                        ioWrite(RxTxData+1, 0x9D);

                    // Identifier (2 bytes)
                    // 0x0000
                        ioWrite(RxTxData,   0x00);
                        ioWrite(RxTxData+1, 0x00);

                    // Flags Fragment offset (2 bytes)(no flag)
                    // 0x4000
                        ioWrite(RxTxData,   0x40);
                        ioWrite(RxTxData+1, 0x00);

                    // Time to Live (1 byte)
                    // 63
                        ioWrite(RxTxData,   0x3f);

                    // Protocol (1 byte)
                    // UDP 17 = 0x11
                        ioWrite(RxTxData+1, 0x11);

                    // Checksum IP Header (2 bytes)
                    // 0xd295
                        ioWrite(RxTxData,   0xd2);
                        ioWrite(RxTxData+1, 0x2a);

                    // Source Address (4 bytes)
                    // IP address 161.246.18.156 (Embedded Device)
                        ioWrite(RxTxData,   0xa1);   // 161
                        ioWrite(RxTxData+1, 0xf6);   // 246
                        ioWrite(RxTxData,   0x12);   // 18
                        ioWrite(RxTxData+1, 0x9c);   // 156

                    // Destination Address (4 bytes)
                    // IP address 161.246.18.157 (Labtop)
                        ioWrite(RxTxData,   0xa1); // 161
                        ioWrite(RxTxData+1, 0xf6); // 246
                        ioWrite(RxTxData,   0x12); // 18
                        ioWrite(RxTxData+1, 0x9d); // 157

                    //  UDP Header (8 bytes)
                    // Source Port (2 bytes)
                    // Port 1300 (0514)
                        ioWrite(RxTxData,   0x05);
                        ioWrite(RxTxData+1, 0x14);

                    // Destination Port (2 bytes)
                    // Port 5000 (0x1388)
                        ioWrite(RxTxData,   0x13);
                        ioWrite(RxTxData+1, 0x88);

                    // Message Length (2 bytes)
                    // 131 bytes (UDP Header(8) + Data(129))
                        ioWrite(RxTxData,   0x00);
                        ioWrite(RxTxData+1, 0x89);

                    // Checksum (2 bytes)
                    // 0x0000 (no checksum used)
                        ioWrite(RxTxData,   0x00);
                        ioWrite(RxTxData+1, 0x00);

                    //Data
```

```
                        for (k=0;k<129;k++)
                        {
                                if (k%2==0)
                                ioWrite(RxTxData, RxBuffer[k]);
                                else
                                ioWrite(RxTxData+1, RxBuffer[k]);
                        }
}
void FCP301()
{
                SetBuad();
                TXframe();
                RXframe();

                Connect();
                TXframe();
                RXframe();

                CreateCaptureHandle();
                TXframe();
                RXframe();

                Capture();
                TXframe();
                RXframe();
                while((frameRXbuff[6]!=0x00)&(frameRXbuff[7]!=0x00))
                {
                        Capture();
                        TXframe();
                        RXframe();
                }
                Red = ON;
                Green = ON;

                GetImg();
                TXframe();
                RXframe();

                //Send Picture Via Etherner
                j = 1;
                //initial Header
                RxBuffer[0]=0;
                for (l=0;l<16;l++)
                {

                        ContImg();
                        TXframe();
                        //Recieve 1011 bytes and kept in RAM
                                x=0;
                                while(x<1011)
                                {
                                        Yellow = ON;
                                        while(RI==0){}
                                        temp = SBUF;
                                        memwr(x,temp);

                                        x++;
                                        RI=0;
                                        Yellow = OFF;
                                }
                                Red = ON;
                        //Send Picture Packet via Ethernet
                                for (z=9;z<1009;z++)
                                {
                                        RxBuffer[j] = memrd(z);
                                        j++;
                                        if (j==129)
                                        {
```

```
                                TxPic();
                                j = 1;
                                RxBuffer[0]++;
                        }
                }
                Red = OFF;
}
    //Recieve Picture
       ContImg();
       TXframe();
              x=0;
              while(x<395)
              {
                     Yellow = ON;
                     while(RI==0){}
                     temp = SBUF;
                     memwr(x,temp);

                     x++;
                     RI=0;
                     Yellow = OFF;
              }
        //Send Picture
              for (z=9;z<393;z++)
              {
                     RxBuffer[j] = memrd(z);
                     j++;
                     if (j==129)
                     {
                            TxPic();
                            j = 1;
                            RxBuffer[0]++;
                     }
              }
              Red = OFF;
Green = OFF;
Yellow =ON;

Complmg();
TXframe();
RXframe();

DisConnect();
TXframe();
RXframe();

Yellow = ON;
TR1=0;
//Send Complete Frame
txudp();
ioWrite(RxTxData+1,'R');

}
```

2. Source Code ในส่วน Fingerprint Identification

```
import java.io.*;
import java.net.*;
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.sql.*;
import java.awt.image.*;
public class IdentFinger extends JFrame
{
 private String status,incomingdata,incomingStatus,txdata,Seq,auther,name;
 private double percent;
 private byte[] framebuff1=new byte[129];
 private int[] framebuff2=new int[129];
 private int[][] pic =new int[128][128];
 private JLabel a0,b0,c0,d0,e0,f0,g0,h0,i0,j0,k0;
 private JTextField a1,b1,c1,d1,e1,f1,g1,h1,i1,j1,k1;
 private JPanel panel0,panel1,panel2,panel3,panel4,panel5,panel6,panel7,panel8;
 private FlowLayout
layout0,layout1,layout2,layout3,layout4,layout5,layout6,layout7,layout8;
 private DatagramSocket socket;
 private DatagramPacket recievePacket;
 private InetAddress ClientIP;
 private int ClientPort,comb,recog =0;
 Image Img1,Img2;
 int[] tempimg1=new int[256*256];
 int[] tempimg2=new int[256*256];
 JLabel ImgLabel1,ImgLabel2;
 JPanel card2,ImgPanel1,ImgPanel2;
 ImageIcon ImgIcon1,ImgIcon2;
 JButton submit;
 String nam,file,aut;
  public IdentFinger()throws Exception
 {
  super("Fingerprint Indentification via Ethernet");
  Container pane =getContentPane();

  JTabbedPane tabbedPane =new JTabbedPane();

  JPanel card1=new JPanel();
     card1.setLayout(new GridLayout(5,1,1,1));

     panel0=new JPanel();
     layout0=new FlowLayout();
     layout0.setAlignment(FlowLayout.CENTER);
     panel0.setLayout(layout0);
     a0=new JLabel("Server IP ");
     panel0.add(a0);
     a1=new JTextField("",8);
     a1.setEditable(false);
     panel0.add(a1);
     b0=new JLabel(" Port ");
     panel0.add(b0);
     b1=new JTextField("5000",3);
     b1.setEditable(false);
     panel0.add(b1);
     card1.add(panel0);
```

```
panel1 = new JPanel();
layout1 = new FlowLayout();
layout1.setAlignment(FlowLayout.CENTER);
panel1.setLayout(layout1);
c0 = new JLabel("Client    IP ");
panel1.add(c0);
c1 = new JTextField("", 8);
c1.setEditable(false);
panel1.add(c1);
d0 = new JLabel(" Port ");
panel1.add(d0);
d1 = new JTextField("", 3);
d1.setEditable(false);
panel1.add(d1);
card1.add(panel1);

panel2 = new JPanel();
layout2 = new FlowLayout();
layout2.setAlignment(FlowLayout.CENTER);
panel2.setLayout(layout2);
e0 = new JLabel("Status       ");
panel2.add(e0);
e1 = new JTextField("", 15);
e1.setEditable(false);
status = "  Waiting For Connection      ";
e1.setText(status);
panel2.add(e1);
card1.add(panel2);

panel3 = new JPanel();
layout3 = new FlowLayout();
layout3.setAlignment(FlowLayout.CENTER);
panel3.setLayout(layout2);
f0 = new JLabel("Name       ");
panel3.add(f0);
f1 = new JTextField("", 15);
f1.setEditable(false);
panel3.add(f1);
card1.add(panel3);

panel4 = new JPanel();
layout4 = new FlowLayout();
layout4.setAlignment(FlowLayout.CENTER);
panel4.setLayout(layout4);
g0 = new JLabel("Auther ");
panel4.add(g0);
g1 = new JTextField("", 4);
g1.setEditable(false);
panel4.add(g1);
h0 = new JLabel("Percent ");
panel4.add(h0);
h1 = new JTextField("", 6);
h1.setEditable(false);
panel4.add(h1);
card1.add(panel4);

    card2 = new JPanel();
```

```
    card2.setLayout(new GridLayout(1,2,1,1));

JPanel card3 = new JPanel();
    card3.setLayout(new GridLayout(4,1,1,1));
    panel5 = new JPanel();
    layout5 = new FlowLayout();
    layout5.setAlignment(FlowLayout.CENTER);
    panel5.setLayout(layout5);
    i0 = new JLabel("Name                    ");
    panel5.add(i0);
    i1 = new JTextField("", 15);
    i1.setEditable(true);
    panel5.add(i1);
    card3.add(panel5);

    panel6 = new JPanel();
    layout6 = new FlowLayout();
    layout6.setAlignment(FlowLayout.CENTER);
    panel6.setLayout(layout6);
    j0 = new JLabel("Filename          ");
    panel6.add(j0);
    j1 = new JTextField("", 15);
    j1.setEditable(true);
    panel6.add(j1);
    card3.add(panel6);

    panel7 = new JPanel();
    layout7 = new FlowLayout();
    layout7.setAlignment(FlowLayout.CENTER);
    panel7.setLayout(layout7);
    k0 = new JLabel("Authorization");
    panel7.add(k0);
    k1 = new JTextField("", 15);
    k1.setEditable(true);
    panel7.add(k1);
    card3.add(panel7);

    panel8 = new JPanel();
    layout8 = new FlowLayout();
    layout8.setAlignment(FlowLayout.CENTER);
    panel8.setLayout(layout8);
    submit = new JButton("Submit");
    submit.setEnabled(false);
//Listener
    submit.addActionListener(new ActionListener()
{
 public void actionPerformed(ActionEvent e)
 {
  try
  {
          nam = i1.getText();
          file = j1.getText();
          aut  = k1.getText();
          ImgProcess App3 = new ImgProcess();
          App3.Enrollment(nam,file,aut);
  }
  catch(Exception exc){ System.out.println(exc); }
 }
});
```

```
    panel3.add(submit);
    card3.add(panel3);

 try
 {
 InetAddress host = InetAddress.getLocalHost();
    a1.setText(""+host.getHostAddress());
 }
 catch(UnknownHostException e)
 {
           e1.setText(e.toString());
 }

        tabbedPane.addTab("Connection", card1);
        tabbedPane.addTab("Fingerprint", card2);
    tabbedPane.addTab("Enrollment", card3);
        pane.add(tabbedPane, BorderLayout.CENTER);
        pack();
    setSize(340,320);
        setVisible(true);
}

private void CreateDatagrameSocket()
{
//Create Datagram Socket
 try
 {
      socket =new DatagramSocket(5000);
 }
 catch(SocketException socketException )
 {
         socketException.printStackTrace();
  System.exit(1);
 }
}

private void waitForPackets()throws Exception
 {
 while(true)
 {
  try
  {
  byte data_rx[]=new byte[200];
  DatagramPacket receivePacket =
   new DatagramPacket(data_rx, data_rx.length );
        socket.receive(receivePacket );

        c1.setText(""+receivePacket.getAddress());
        ClientIP =receivePacket.getAddress();
        d1.setText(""+receivePacket.getPort());
        ClientPort =receivePacket.getPort();
  if(status.equals("Connection Ready to Rx"))
  {
          txdata="0";
          incomingdata =new String(receivePacket.getData(),
  0, receivePacket.getLength());
   if(incomingdata.equals("Connection Ready to Rx"));
   //Function
   else
   {
```

```
if(incomingdata.length()==129)
{
            comb =(int)((0<<24)+(0<<16)+(0<<8)+(data_rx[0]&0xff));
 System.out.print(comb+" ");
 //Copy to row
 for(int j=0;j<128;j++)
 {
              pic[127-comb][j]=(int)((0<<24)+(0<<16)+(0<<8)+(data_rx[j+1]&0xff));
  System.out.print(pic[127-comb][j]+" ");
 }
 System.out.println();
}
//Complete Sending
else
{
            Pixcrop App2=new Pixcrop();
            App2.Pixcrop(pic);

            tempimg1=App2.ReturnImg1();
            Img1=createImage(new MemoryImageSource(256,256,tempimg1,0,256));
            ImgPanel1 = new JPanel();
            ImgIcon1 = new ImageIcon(Img1);
            ImgLabel1=new JLabel(ImgIcon1);
            ImgPanel1.add(ImgLabel1);
            card2.removeAll();
            card2.add(ImgPanel1);

            tempimg2=App2.ReturnImg2();
            Img2=createImage(new MemoryImageSource(256,256,tempimg2,0,256));
            ImgPanel2=new JPanel();
            ImgIcon2= new ImageIcon(Img2);
            ImgLabel2=new JLabel(ImgIcon2);
            ImgPanel2.add(ImgLabel2);
            card2.add(ImgPanel2);
 //Enable Enrollment
            submit.setEnabled(true);

            name =App2.ReturnName();
            auther =App2.ReturnAuther();
            percent =App2.Returnpercent();
            f1.setText(name);
            g1.setText(auther);
            h1.setText(""+percent);
 if(auther.equals("YES"))
 {
              txdata="1";
              SendPacket();
              SendPacket();
              SendPacket();
              txdata="0";
 }
 if(auther.equals("NO"))
 {
              txdata="0";
              SendPacket();
              SendPacket();
              SendPacket();
              txdata="0";
 }
```

```
                recog =0;
    }
   }
  }
  if(status.equals("  Waiting For Connection      "))
  {
          incomingStatus = new String(receivePacket.getData(),
 0, receivePacket.getLength());
  if(incomingStatus.equals("Connection Ready to Rx"))
  {
          e1.setText(incomingStatus);
          status =incomingStatus;
  }
  }
 }
 catch(IOException ioException )
 {
           e1.setText(ioException.toString());
           ioException.printStackTrace();
 }
}
}

private void SendPacket()
{
 try{
  byte data[]= txdata.getBytes();
  DatagramPacket sendPacket =new DatagramPacket
  (data, data.length,ClientIP,ClientPort);
       socket.send(sendPacket );
 }
 catch(IOException ioException )
 {
       e1.setText(ioException.toString()+"\n");
       ioException.printStackTrace();
 }
}
public static void main(String[] args)throws Exception
{
    IdentFinger App =new IdentFinger();
    App.CreateDatagrameSocket();
    App.addWindowListener(
 new WindowAdapter(){
  public void windowClosing (WindowEvent e)
  {
   System.exit(0);

  }
 }
);
    App.waitForPackets();
 }
}
```

3. Source Code ในส่วน Image Process

```
import jmatlink.*;
import java.sql.*;
public class  ImgProcess{
 String name ="Unknown Person",auther ="NO",file;
 double tempmatchper,matchper=0;
```

```
int hh =(int)(256-12)/12);
int ww =(int)(256-12)/12);
int hhh =hh*12;
int www =ww*12;
int x,y;
int[][] ArrayTemp =new int[256][256];
double[][] image1 =new double[hhh][www];
int[] OputImg;
int HiBranch,WeBranch,HiEnd,WeEnd;
public void ImgPreProcess(int pixel[],int w,int h)throws Exception
{
int[] result =new int[www*hhh];
   ProgTool tool =new ProgTool();
   JMatLink engine =new JMatLink();
   ImgAlgo Algo =new ImgAlgo();
//Convert Stream[] to Array[][]
int[][] inttemp =new int[h][w];
   inttemp =tool.StrmtoAry(pixel,w,h);
//Convert int to double
double[][] doubletemp =new double[h][w];
   doubletemp =tool.InttoDoub(inttemp,w,h);
//Use Matlab to Enhance Image By STFT Method
   engine.engOpen();
   engine.engEvalString(tool.setWorkdir("Matlabsrc"));
   engine.engPutArray("I",doubletemp);
   engine.engEvalString("[enhimg]= fftenhance(I);");
   engine.engEvalString("t =otsu_threshold(enhimg);");
   engine.engEvalString("Img =double(enhimg)");
double[][] enhimg =engine.engGetArray("Img");
int t =(int)engine.engGetScalar("t");
//Convert double to int
   inttemp =tool.DoubtoInt(enhimg,w,h);
//Otsu Threadhold
int passloop=0;
int[][] OtsuTemp =new int[h][w];
while(passloop==0)
{
     passloop=1;
 for(int i=0;i<h;i++)
 {
  for(int j=0;j<w;j++)
  {
   if(inttemp[i][j]<t)
            OtsuTemp[i][j]=255;
   else OtsuTemp[i][j]=0;
  }
 }
 for(int i=0;i<6;i++)
 {
  for(int j=0;j<6;j++)
  {
   if(OtsuTemp[i][j]==255)
          passloop=0;
  }
 }
     t=t-1;
}
   inttemp=OtsuTemp;  //end Otsu's Threadhold
//Cut pixel
```

```
for(int i=0;i<hhh;i++)
{
 for(int j=0;j<www;j++)
 {
        image1[i][j]=inttemp[i+6][j+6];
 }
}
int[][] resizeInt=new int[hhh][www];
    resizeInt = tool.DoubtoInt(image1, www, hhh);

    engine.engPutArray("image1", image1);
    engine.engEvalString("[o1Bound, o1Area]=direction(image1,16);");
    engine.engEvalString("[o2, o1Bound, o1Area]=drawROI(image1, o1Bound, o1Area, 1)");
    engine.engEvalString("[A, B]=size(o2);");
    y =(int)engine.engGetScalar("A");
    x =(int)engine.engGetScalar("B");
    engine.engEvalString("o1=im2double(bwmorph(o2, 'thin', Inf));");
double[][] o1 = new double[y][x];
    o1=engine.engGetArray("o1");
int[][] Resize1ArrayInt = new int[y][x];
    Resize1ArrayInt = tool.DoubtoInt(o1, x, y);
int[] Resize1StrmInt = new int[x*y];
    Resize1StrmInt = tool.ArytoStrm(Resize1ArrayInt, x, y);
//Imperfect Removal
    Resize1StrmInt = ImgProcess(Resize1StrmInt, x, y);
    Resize1ArrayInt = tool.StrmtoAry(Resize1StrmInt, x, y);
    o1=tool.InttoDoub(Resize1ArrayInt, x, y);
    engine.engPutArray("o1", o1);
    engine.engEvalString("o1=im2double(bwmorph(o1, 'clean'));");
    engine.engEvalString("o1=im2double(bwmorph(o1, 'hbreak'));");
    engine.engEvalString("o1=im2double(bwmorph(o1, 'spur'));");
//Minutia Extraction

engine.engEvalString("[end_list1, branch_list1, ridgeMap1, edgeWidth]=mark_minutia(o1, o1B
ound, o1Area,16);");

engine.engEvalString("[pathMap1, real_end1, real_branch1]=remove_spurious_Minutia(o1, end
_list1, branch_list1, o1Area, ridgeMap1, edgeWidth);");

    engine.engEvalString("[hibr, webr]=size(real_branch1);");
    HiBranch =(int)engine.engGetScalar("hibr");
    WeBranch =(int)engine.engGetScalar("webr");
double[][] DoubBranch = new double[HiBranch][WeBranch];
int[][] Branch = new int[HiBranch][WeBranch];
    DoubBranch =engine.engGetArray("real_branch1");
    Branch = tool.DoubtoInt(DoubBranch, WeBranch, HiBranch);

    engine.engEvalString("[hiend, weend]=size(real_end1);");
    HiEnd =(int)engine.engGetScalar("hiend");
    WeEnd =(int)engine.engGetScalar("weend");
double[][] DoubEnd = new double[HiEnd][WeEnd];
int[][] End = new int[HiEnd][WeEnd];
    DoubEnd =engine.engGetArray("real_end1");
    End = tool.DoubtoInt(DoubEnd, WeEnd, HiEnd);

//Save Template
//Insert Filename
    engine.engEvalString("save(char('temp.DAT'),'real_end1','pathMap1','-ASCII');");
```

```
//Part DataBase
   engine.engEvalString("finger2=load(char('temp.DAT'))"); //Load i/p template
Class.forName("oracle.jdbc.OracleDriver");
 Connection c =
DriverManager.getConnection("jdbc:oracle:thin:@127.0.0.1:1521:oracle","test","test");
 Statement s = c.createStatement();
 ResultSet r = s.executeQuery("select name,DBFile,auther from verifinger");
  while(r.next())
  {
           file = r.getString(2);
  //Matching
  //Template in database
           engine.engEvalString("finger1=load(char('"+file+"'))");
  //Input Template And Set Torrerance
           engine.engEvalString("percent_match=match_end(finger1,finger2,15);");
           tempmatchper = engine.engGetScalar("percent_match");
           engine.engEvalString("clear finger1;");
           engine.engEvalString("clear percent_match;");
  //Cut Percent Match
  if((tempmatchper>matchper)&(tempmatchper>=70))
  {
             matchper = tempmatchper;
             name = r.getString(1);
             auther = r.getString(3);
  }
  }
     s.close();
     c.close();
//Close MATLAB Engine
   engine.engClose();
System.out.print(name+" "+matchper);
//Show Enhance Image
for(int i=0;i<hhh;i++)
{
 for(int j=0;j<www;j++)
 {
  if(resizeInt[i][j]==0)
          resizeInt[i][j]=255;
  else resizeInt[i][j]=0;
 }
}
//Mark Branch
for(int i=0;i<HiBranch;i++)
{
 int hipoint = Branch[i][0]+15;
 int wepoint = Branch[i][1]+15;
 for(int x=(hipoint-3);x<(hipoint+3);x++)
 {
       resizeInt[x][wepoint]=125;
 }
 for(int y=(wepoint-3);y<(wepoint+3);y++)
 {
       resizeInt[hipoint][y]=125;
 }

}
//Mark End
for(int i=0;i<HiEnd;i++)
{
```

```
 int hipoint = End[i][0]+15;
 int wepoint = End[i][1]+15;
 for(int x=(hipoint-3); x<(hipoint+3); x++)
 {
       resizeInt[x][wepoint]=125;
 }
 for(int y=(wepoint-3); y<(wepoint+3); y++)
 {
       resizeInt[hipoint][y]=125;
 }
}
//Enlarge to 256*256
int[][] EnlargeImg = new int[256][256];
for(int i=0; i<256; i++)
{
 for(int j=0; j<256; j++)
 {
       EnlargeImg[i][j]=255;
 }
}
for(int i=0; i<hhh; i++)
{
 for(int j=0; j<www; j++)
 {
       EnlargeImg[i+8][j+8]=resizeInt[i][j];
 }
}
//Convert to Stream
   OputImg = tool.ArytoStrm(EnlargeImg, 256,256);
}
public void Enrollment(String name,String file,String aut)throws Exception
{
    ProgTool tool = new ProgTool();
    JMatLink engine = new JMatLink();
    engine.engOpen();
    engine.engEvalString(tool.setWorkdir("Matlabsrc"));
    engine.engEvalString("tempfinger=load(char('temp.DAT'));");
    engine.engEvalString("save(char('"+file+"'),'tempfinger','-ASCII');");
    engine.engClose();
 Class.forName("oracle.jdbc.OracleDriver");
  Connection c =
DriverManager.getConnection("jdbc:oracle:thin:@127.0.0.1:1521:oracle","test","test");
  Statement s = c.createStatement();
      s.executeUpdate("Insert into verifinger values('"+name +"','"+file +"','"+aut
+"')");
      s.close();
      c.close();
}
public String ReturnName()
{
 return name;
}
public String ReturnAuther()
{
 return auther;
}
public double Returnpercent()
{
 return matchper;
}
public int[] ReturnImg()
```

```
{
 return OputImg;
}
public int[] ImgProcess(int pixel[],int w,int h)
{
 int[]result =new int[w*h];
    ImgAlgo Algo =new ImgAlgo();
    ProgTool tool =new ProgTool();
 //Decode and Convert 0 to 255: to 0
 for(int i=0;i<(w*h);i++)
 {
  if(pixel[i]==0)pixel[i]=255;
  else pixel[i]=0;
 }
 //Clear 4 Border
 int[][] inttemp =new int[h][w];
    inttemp =tool.StrmtoAry(pixel,w,h);
 for(int i=0;i<h;i++)
 {
       inttemp[i][0]=255;
       inttemp[i][w-1]=255;
 }
 for(int j=0;j<w;j++)
 {
       inttemp[0][j]=255;
       inttemp[h-1][j]=255;
 }
    pixel =tool.ArytoStrm(inttemp,w,h);
    result =Algo.ImperRemove(pixel,w,h);
 for(int i=0;i<w*h;i++)
 {
  if(result[i]==255)result[i]=0;
  else result[i]=1;
 }
 return result;
 }
}
```

4. Source Code ในส่วน Image Algorithm

```
public class ImgAlgo
{
//Imperfect Removal
 public int[] ImperRemove(int[]Thinning,int w,int h)
 {
  //Branch Eliminate
   int wh =w*h;
   int[]branchE =new int[wh];
   int[]bg1=new int[wh];
   int[]bg2=new int[wh];
   int Index,Index2,nextpixel;
   //copy Thinning[] to bg1[] and coding 0 to 1255 to 0
   for(int i=0 ; i<wh ; i++)
   {
    if(Thinning[i]==0)
            bg1[i]=1;
    else bg1[i]=0;
   }

    //Scan for Endpoint exclude border
```

```
for(int i =w ; i <=((wh)-(w+1)); i++)
{
if((!((i%w ==0) | (i%w ==(w-1))))&(bg1[i]==1))
 {
          Index =bg1[i-w-1]+((bg1[i-w])*2)+((bg1[i-w+1])*4)+((bg1[i-1])*8)+((bg1[i+1])*16)
  +((bg1[i+w-1])*32)+((bg1[i+w])*64)+((bg1[i+w+1])*128);
 //Found initial ending
 if((Index==1)|(Index==2)|(Index==4)|(Index==8)|
 (Index==16)|(Index==32)|(Index==64)|(Index==128))
 {
            nextpixel=i;
 if(Index==1) nextpixel=i-w-1;   if(Index==2) nextpixel=i-w;
 if(Index==4) nextpixel=i-w+1;   if(Index==8) nextpixel=i-1;
 if(Index==16) nextpixel=i+1;     if(Index==32) nextpixel=i+w-1;
 if(Index==64) nextpixel=i+w;     if(Index==128) nextpixel=i+w+1;
            bg2[i]=1;
            bg2[nextpixel]=1;
 //Tracking line
 for(int th=0; th<5 ; th++)
 {
              Index =bg1[nextpixel-w-1]+((bg1[nextpixel-w])*2)+((bg1[nextpixel-w+1])*4)
  +((bg1[nextpixel-1])*8)+((bg1[nextpixel+1])*16)+((bg1[nextpixel+w-1])*32)
  +((bg1[nextpixel+w])*64)+((bg1[nextpixel+w+1])*128);

 //If topology is connectivity
 if((Index==66)|(Index==36)|(Index==129)|(Index==24)|
 (Index==130)|(Index==34)|(Index==68)|(Index==65)|
 (Index==17)|(Index==48)|(Index==12)|(Index==136))
 {
                Index2=bg2[(nextpixel-w-1)]+((bg2[nextpixel-w])*2)
  +((bg2[nextpixel-w+1])*4)+((bg2[nextpixel-1])*8)
  +((bg2[nextpixel+1])*16)+((bg2[nextpixel+w-1])*32)
  +((bg2[nextpixel+w])*64)+((bg2[nextpixel+w+1])*128);
 int diff =Index-Index2;
 if(diff==1)nextpixel =nextpixel-w-1;
 if(diff==2)nextpixel =nextpixel-w;
 if(diff==4)nextpixel =nextpixel-w+1;
 if(diff==8)nextpixel =nextpixel-1;
 if(diff==16)nextpixel =nextpixel+1;
 if(diff==32)nextpixel =nextpixel+w-1;
 if(diff==64)nextpixel =nextpixel+w;
 if(diff==128)nextpixel =nextpixel+w+1;
              bg2[nextpixel]=1;
 }//end connectivity

 //If topology is nearing final junction
 if(th>0)
 {
 //Direction 1
 if((Index==13)|(Index==25)|(Index==137)|
 (Index==44)|(Index==56)|(Index==168))
 {
                nextpixel =nextpixel-1;
                bg2[nextpixel]=1;
 }
 //Direction 2
 if((Index==225)|(Index==226)|(Index==228)|
 (Index==193)|(Index==194)|(Index==196)|
 (Index==97)|(Index==98)|(Index==100))
```

```
    {
                    nextpixel =nextpixel-w;
                    bg2[nextpixel]=1;
    }
    if((Index==145)|(Index==152)|(Index==176))
    {
                    nextpixel =nextpixel-1;
                    bg2[nextpixel]=1;
    }
    //Direction 3
    if((Index==21)|(Index==28)|(Index==52))
    {
                    nextpixel = nextpixel+1;
                    bg2[nextpixel]=1;
    }
    if((Index==39)|(Index==71)|(Index==135)|
    (Index==35)|(Index==67)|(Index==131)|
    (Index==38)|(Index==70)|(Index==134))
    {
                    nextpixel =nextpixel-w;
                    bg2[nextpixel]=1;
    }
   }//End nearing Junction

   //If topology is final junction
   if((Index==39)|(Index==45)|(Index==57)|(Index==71)|
   (Index==135)|(Index==149)|(Index==156)|(Index==169)|
   (Index==180)|(Index==225)|(Index==226)|(Index==228)|
   (Index==26)|(Index==37)|(Index==49)|(Index==50)|
   (Index==69)|(Index==76)|(Index==81)|(Index==88)|
   (Index==133)|(Index==138)|(Index==140)|(Index==161)|
   (Index==162)|(Index==164))
   {
                bg2[nextpixel]=0;
    //Delete Tracking line from bg1[]
    for(int j=0 ; j<wh ; j++)
                    bg1[j]=bg1[j]-bg2[j];
    //Clear bg2[]
    for(int j=0 ; j<wh ; j++)
                    bg2[j]=0;
    //Exit for loop
                th=5;
   }
   }//end Tracking line
   //Clear bg2[]
   for(int j=0 ; j<wh ; j++)
                bg2[j]=0;
  }//end found junction
 }
}//end scan endpoint
//Decode bg1[]to bridgeE[]
for(int i=0; i<wh ; i++)
{
 if(bg1[i]==0)
         branchE[i]=255;
 else branchE[i]=0;
}
//Bridge Eliminate
int[]bridgeE =new int[wh];
```

```
//copy Thinning[] to bg1[] and coding 0 to 1255 to 0
for(int i=0 ; i<wh ; i++)
{
 if(branchE[i]==0)
         bg1[i]=1;
 else bg1[i]=0;
}
 //Scan Round 1 exclude border
 for(int i =w ; i <=((wh)-(w+1)); i++)
 {
 if((((i%w ==0) | (i%w ==(w-1))))&(bg1[i]==1))
  {
            Index =bg1[i-w-1]+((bg1[i-w]*2)+((bg1[i-w+1])*4)+((bg1[i-1])*8)+((bg1[i+1]*16)
   +((bg1[i+w-1])*32)+((bg1[i+w])*64)+((bg1[i+w+1])*128);
  //Found initial junction
  if((Index==39)|(Index==45)|(Index==57)|(Index==71)|
  (Index==135)|(Index==149)|(Index==156)|(Index==169)|
  (Index==180)|(Index==225)|(Index==226)|(Index==228)|
  (Index==26)|(Index==37)|(Index==49)|(Index==50)|
  (Index==69)|(Index==76)|(Index==81)|(Index==88)|
  (Index==133)|(Index==138)|(Index==140)|(Index==161)|
  (Index==162)|(Index==164))
  {
             nextpixel=i;
  if(Index==39) nextpixel=i+w-1;   if(Index==45) nextpixel=i-w+1;
  if(Index==57) nextpixel=i+1;      if(Index==71) nextpixel=i+w;
  if(Index==135) nextpixel=i+w+1;   if(Index==149) nextpixel=i-w-1;
  if(Index==156) nextpixel=i-1;     if(Index==169) nextpixel=i+w+1;
  if(Index==180) nextpixel=i+w-1;   if(Index==225) nextpixel=i-w-1;
  if(Index==226) nextpixel=i-w;     if(Index==228) nextpixel=i-w+1;
  if(Index==26) nextpixel=i-1;      if(Index==37) nextpixel=i-w+1;
  if(Index==49) nextpixel=i+1;      if(Index==50) nextpixel=i-1;
  if(Index==69) nextpixel=i-w+1;    if(Index==76) nextpixel=i-w+1;
  if(Index==81) nextpixel=i+1;      if(Index==88) nextpixel=i+1;
  if(Index==133) nextpixel=i-w+1;   if(Index==138) nextpixel=i+w+1;
  if(Index==140) nextpixel=i+w+1;   if(Index==161) nextpixel=i+w+1;
  if(Index==162) nextpixel=i+w+1;   if(Index==164) nextpixel=i+w+1;
             bg2[i]=1;
             bg2[nextpixel]=1;
  //Tracking line
  for(int th=0; th<8 ; th++)
  {
              Index =bg1[nextpixel-w-1]+((bg1[nextpixel-w])*2)+((bg1[nextpixel-w+1])*4)
   +((bg1[nextpixel-1])*8)+((bg1[nextpixel+1])*16)+((bg1[nextpixel+w-1])*32)
   +((bg1[nextpixel+w])*64)+((bg1[nextpixel+w+1])*128);

  //If topology is connectivity
  if((Index==66)|(Index==36)|(Index==129)|(Index==24)|
  (Index==130)|(Index==34)|(Index==68)|(Index==65)|
  (Index==17)|(Index==48)|(Index==12)|(Index==136))
  {
                Index2=bg2[(nextpixel-w-1)]+((bg2[nextpixel-w])*2)
   +((bg2[nextpixel-w+1])*4)+((bg2[nextpixel-1])*8)
   +((bg2[nextpixel+1])*16)+((bg2[nextpixel+w-1])*32)
   +((bg2[nextpixel+w])*64)+((bg2[nextpixel+w+1])*128);
  int diff =Index-Index2;
  if(diff==1)nextpixel =nextpixel-w-1;
  if(diff==2)nextpixel =nextpixel-w;
```

```
 if(diff==4)nextpixel =nextpixel-w+1;
 if(diff==8)nextpixel =nextpixel-1;
 if(diff==16)nextpixel =nextpixel+1;
 if(diff==32)nextpixel =nextpixel+w-1;
 if(diff==64)nextpixel =nextpixel+w;
 if(diff==128)nextpixel =nextpixel+w+1;
              bg2[nextpixel]=1;
              bg2[i]=0;
}//end connectivity

//If topology of connectivity is nearing initial junction
if(th==1)
{
 if((Index==13)|(Index==44))
 {
                nextpixel =nextpixel-w+1;
                bg2[nextpixel]=1;
 }
 if((Index==25)|(Index==56))
 {
                nextpixel =nextpixel+1;
                bg2[nextpixel]=1;
 }
 if((Index==137)|(Index==168))
 {
                nextpixel =nextpixel+w+1;
                bg2[nextpixel]=1;
 }
}//end nearing junction

//If topology is nearing final junction
if(th>2)
{
 //Direction 1
 if((Index==13)|(Index==25)|(Index==137)|
 (Index==44)|(Index==56)|(Index==168))
 {
                nextpixel =nextpixel-1;
                bg2[nextpixel]=1;
 }
 //Direction 2
 if((Index==225)|(Index==236)|(Index==228)|
 (Index==193)|(Index==194)|(Index==196)|
 (Index==97)|(Index==98)|(Index==100))
 {
                nextpixel =nextpixel-w;
                bg2[nextpixel]=1;
 }
 if((Index==145)|(Index==152)|(Index==176))
 {
                nextpixel =nextpixel+1;
                bg2[nextpixel]=1;
 }
 //Direction 3
 if((Index==21)|(Index==28)|(Index==52))
 {
                nextpixel =nextpixel+1;
                bg2[nextpixel]=1;
 }
 if((Index==99)|(Index==71)|(Index==135)|
```

```
   (Index==35)|(Index==67)|(Index==131)|
   (Index==38)|(Index==70)|(Index==134))
   {
                    nextpixel =nextpixel-w;
                    bg2[nextpixel]=1;
   }
  }//End nearing Junction

  //If topology is final junction
  if((Index==39)|(Index==45)|(Index==57)|(Index==71)|
   (Index==135)|(Index==149)|(Index==156)|(Index==169)|
   (Index==180)|(Index==225)|(Index==226)|(Index==228)|
   (Index==26)|(Index==37)|(Index==49)|(Index==50)|
   (Index==69)|(Index==76)|(Index==81)|(Index==88)|
   (Index==133)|(Index==138)|(Index==140)|(Index==161)|
   (Index==162)|(Index==164))
   {
                  bg2[nextpixel]=0;
   //Delete Tracking line from bg1[]
   for(int j=0 ; j<wh ; j++)
                    bg1[j]=bg1[j]-bg2[j];
   //Clear bg2[]
   for(int j=0 ; j<wh ; j++)
                    bg2[j]=0;
   //Exit for loop
                  th=8;
   }
  }//end Tracking line
  //Clear bg2[]
  for(int j=0 ; j<wh ; j++)
                bg2[j]=0;
 }//end found junction
}
}//end round 1
//Scan Round 2 exclude border
for(int i =w ; i <=((wh)-(W+1)); i++)
{
if(((((i%W == 0) | (i%W ==(W-1))))&(bg1[i]==1))
 {
          Index = bg1[i-W-1]+((bg1[i-W])*2)+((bg1[i-W+1])*4)+((bg1[i-1])*8)+((bg1[i+1])*16)
   +((bg1[i+W-1])*32)+((bg1[i+W])*64)+((bg1[i+W+1])*128);
 //Found junction
 if((Index==26)|(Index==37)|(Index==49)|(Index==50)|
 (Index==69)|(Index==76)|(Index==81)|(Index==88)|
 (Index==133)|(Index==138)|(Index==140)|(Index==161)|
 (Index==162)|(Index==164))
 {
            nextpixel=i;
 if(Index==26) nextpixel=i-w;     if(Index==37) nextpixel=i-W-1;
 if(Index==49) nextpixel=i-W-1;   if(Index==50) nextpixel=i-w;
 if(Index==69) nextpixel=i-W-1;   if(Index==76) nextpixel=i-1;
 if(Index==81) nextpixel=i-W-1;   if(Index==88) nextpixel=i-1;
 if(Index==133) nextpixel=i-W-1;  if(Index==138) nextpixel=i-w;
 if(Index==140) nextpixel=i-W+1;  if(Index==161) nextpixel=i-W-1;
 if(Index==162) nextpixel=i-w;    if(Index==164) nextpixel=i-W+1;
            bg2[i]=1;
            bg2[nextpixel]=1;
 //Tracking line
```

```
for(int th=0; th<8 ; th++)
{
            Index =bg1[nextpixel-W-1]+((bg1[nextpixel-W])*2)+((bg1[nextpixel-W+1])*4)
 +((bg1[nextpixel-1])*8)+((bg1[nextpixel+1])*16)+((bg1[nextpixel+W-1])*32)
 +((bg1[nextpixel+W])*64)+((bg1[nextpixel+W+1])*128);

#If topology is initial connectivity
if((Index==66)|(Index==36)|(Index==129)|(Index==24)|
(Index==130)|(Index==34)|(Index==68)|(Index==65)|
(Index==17)|(Index==48)|(Index==12)|(Index==136))
{
              Index2=bg2[(nextpixel-W-1)]+((bg2[nextpixel-W])*2)
  +((bg2[nextpixel-W+1])*4)+((bg2[nextpixel-1])*8)
  +(bg2[nextpixel+1])*16)+((bg2[nextpixel+W-1])*32)
  +((bg2[nextpixel+W])*64)+((bg2[nextpixel+W+1])*128);
 int diff =Index-Index2;
 if(diff==1)nextpixel =nextpixel-W-1;
 if(diff==2)nextpixel =nextpixel-W;
 if(diff==4)nextpixel =nextpixel-W+1;
 if(diff==8)nextpixel =nextpixel-1;
 if(diff==16)nextpixel =nextpixel+1;
 if(diff==32)nextpixel =nextpixel+W-1;
 if(diff==64)nextpixel =nextpixel+W;
 if(diff==128)nextpixel =nextpixel+W+1;
              bg2[nextpixel]=1;
              bg2[i]=0;
}//end connectivity

#If topology of connectivity is nearing initial junction
if(th==1)
{
 if((Index==225)|(Index==193)|(Index==97))
 {
              nextpixel =nextpixel-W-1;
              bg2[nextpixel]=1;
 }
 if((Index==226)|(Index==194)|(Index==98))
 {
              nextpixel =nextpixel-W;
              bg2[nextpixel]=1;
 }
 if((Index==228)|(Index==196)|(Index==100))
 {
              nextpixel =nextpixel-W+1;
              bg2[nextpixel]=1;
 }
 if(Index==145)
 {
              nextpixel =nextpixel-W-1;
              bg2[nextpixel]=1;
 }
 if(Index==152)
 {
              nextpixel =nextpixel-1;
              bg2[nextpixel]=1;
 }
 if(Index==176)
 {
              nextpixel =nextpixel+W-1;
```

```
                bg2[nextpixel]=1;
 }
}//end nearing junction

//If is nearing final junction
if(th>2)
{
 //Direction 1
 if((Index==13)|(Index==25)|(Index==137)|
 (Index==44)|(Index==56)|(Index==168))
 {
                nextpixel =nextpixel-1;
                bg2[nextpixel]=1;
 }
 //Direction 2
 if((Index==225)|(Index==226)|(Index==228)|
 (Index==193)|(Index==194)|(Index==196)|
 (Index==97)|(Index==98)|(Index==100))
 {
                nextpixel =nextpixel+w;
                bg2[nextpixel]=1;
 }
 if((Index==145)|(Index==152)|(Index==176))
 {
                nextpixel =nextpixel+1;
                bg2[nextpixel]=1;
 }
 //Direction 3
 if((Index==21)|(Index==28)|(Index==52))
 {
                nextpixel =nextpixel+1;
                bg2[nextpixel]=1;
 }
 if((Index==39)|(Index==71)|(Index==135)|
 (Index==35)|(Index==67)|(Index==131)|
 (Index==38)|(Index==70)|(Index==134))
 {
                nextpixel =nextpixel-w;
                bg2[nextpixel]=1;
 }
}//end nearing junction

//If topology is final junction
if((Index==39)|(Index==45)|(Index==57)|(Index==71)|
(Index==135)|(Index==149)|(Index==156)|(Index==169)|
(Index==180)|(Index==225)|(Index==226)|(Index==228)|
(Index==26)|(Index==37)|(Index==49)|(Index==50)|
(Index==69)|(Index==76)|(Index==81)|(Index==88)|
(Index==133)|(Index==138)|(Index==140)|(Index==161)|
(Index==162)|(Index==164))
{
              bg2[nextpixel]=0;
 //Delete Tracking line from bg1[]
 for(int j=0 ; j<wh ; j++)
                bg1[j]=bg1[j]-bg2[j];
 //Clear bg2[]
 for(int j=0 ; j<wh ; j++)
                bg2[j]=0;
 //Exit for loop
```

```
                th=8;
  }
 }//end Tracking line
 //Clear bg2[]
 for(int j=0 ; j<wh ; j++)
              bg2[j]=0;
}//end found junction
}
}//end round 2
//Scan Round 3 exclude border
for(int i =w ; i <=((wh)-(w+1)); i++)
{
if(((i%w ==0) | (i%w ==(w-1)))&(bg1[i]==1))
{
          Index =bg1[i-w-1]+((bg1[i-w])*2)+((bg1[i-w+1])*4)+((bg1[i-1])*8)+((bg1[i+1])*16)
  +((bg1[i+w-1])*32)+((bg1[i+w])*64)+((bg1[i+w+1])*128);

 //Found junction
 if((Index==26)|(Index==37)|(Index==49)|(Index==50)|
 (Index==69)|(Index==76)|(Index==81)|(Index==88)|
 (Index==133)|(Index==138)|(Index==140)|(Index==161)|
 (Index==162)|(Index==164))
 {
            nextpixel=i;
 if(Index==26) nextpixel=i-1;     if(Index==37) nextpixel=i+w-1;
 if(Index==49) nextpixel=i+w-1;   if(Index==50) nextpixel=i+w-1;
 if(Index==69) nextpixel=i+w;      if(Index==76) nextpixel=i+w;
 if(Index==81) nextpixel=i+w;      if(Index==88) nextpixel=i+w;
 if(Index==133) nextpixel=i+w+1;   if(Index==138) nextpixel=i-1;
 if(Index==140) nextpixel=i-1;     if(Index==161) nextpixel=i+w-1;
 if(Index==162) nextpixel=i-w-1;   if(Index==164) nextpixel=i+w-1;
            bg2[i]=1;
            bg2[nextpixel]=1;
 //Tracking line
 for(int th=0; th<8 ; th++)
 {
             Index =bg1[nextpixel-w-1]+((bg1[nextpixel-w])*2)+((bg1[nextpixel-w+1])*4)
  +((bg1[nextpixel-1])*8)+((bg1[nextpixel+1])*16)+((bg1[nextpixel+w-1])*32)
  +((bg1[nextpixel+w])*64)+((bg1[nextpixel+w+1])*128);

  //If topology is initial connectivity
  if((Index==66)|(Index==36)|(Index==129)|(Index==24)|
  (Index==130)|(Index==34)|(Index==68)|(Index==65)|
  (Index==17)|(Index==48)|(Index==12)|(Index==136))
  {
              Index2=bg2[(nextpixel-w-1)]+((bg2[nextpixel-w])*2)
   +((bg2[nextpixel-w+1])*4)+((bg2[nextpixel-1])*8)
   +((bg2[nextpixel+1])*16)+((bg2[nextpixel+w-1])*32)
   +((bg2[nextpixel+w])*64)+((bg2[nextpixel+w+1])*128);
  int diff =Index-Index2;
  if(diff==1)nextpixel =nextpixel-w-1;
  if(diff==2)nextpixel =nextpixel-w;
  if(diff==4)nextpixel =nextpixel-w+1;
  if(diff==8)nextpixel =nextpixel-1;
  if(diff==16)nextpixel =nextpixel+1;
  if(diff==32)nextpixel =nextpixel+w-1;
  if(diff==64)nextpixel =nextpixel+w;
  if(diff==128)nextpixel =nextpixel+w+1;
```

```
                bg2[nextpixel]=1;
                bg2[i]=0;
}//end connectivity

//If topology of connectivity is nearing initial junction
if(th==1)
{
 if((Index==39)|(Index==35)|(Index==38))
 {
                nextpixel =nextpixel+w-1;
                bg2[nextpixel]=1;
 }
 if((Index==71)|(Index==67)|(Index==70))
 {
                nextpixel =nextpixel+w;
                bg2[nextpixel]=1;
 }
 if((Index==135)|(Index==131)|(Index==134))
 {
                nextpixel =nextpixel+w+1;
                bg2[nextpixel]=1;
 }
 if(Index==21)
 {
                nextpixel =nextpixel-w-1;
                bg2[nextpixel]=1;
 }
 if(Index==28)
 {
                nextpixel =nextpixel-1;
                bg2[nextpixel]=1;
 }
 if(Index==52)
 {
                nextpixel =nextpixel+w-1;
                bg2[nextpixel]=1;
 }
}//end nearing junction

//If topology is nearing final junction
if(th>2)
{
 //Direction 1
 if((Index==13)|(Index==25)|(Index==137)|
  (Index==44)|(Index==56)|(Index==168))
 {
                nextpixel =nextpixel-1;
                bg2[nextpixel]=1;
 }
 //Direction 2
 if((Index==225)|(Index==226)|(Index==228)|
  (Index==193)|(Index==194)|(Index==196)|
  (Index==97)|(Index==98)|(Index==100))
 {
                nextpixel =nextpixel+w;
                bg2[nextpixel]=1;
 }
 if((Index==145)|(Index==152)|(Index==176))
 {
                nextpixel =nextpixel+1;
```

```
                    bg2[nextpixel]=1;
    }
    //Direction 3
    if((Index==21)|(Index==28)|(Index==52))
    {
                    nextpixel =nextpixel+1;
                    bg2[nextpixel]=1;
    }
    if((Index==39)|(Index==71)|(Index==135)|
    (Index==35)|(Index==67)|(Index==131)|
    (Index==38)|(Index==70)|(Index==134))
    {
                    nextpixel =nextpixel-w;
                    bg2[nextpixel]=1;
    }
   }//end nearing junction

   //If topology is junction
   if((Index==39)|(Index==45)|(Index==57)|(Index==71)|
   (Index==135)|(Index==149)|(Index==156)|(Index==169)|
   (Index==180)|(Index==225)|(Index==226)|(Index==228)|
   (Index==26)|(Index==37)|(Index==49)|(Index==50)|
   (Index==69)|(Index==76)|(Index==81)|(Index==88)|
   (Index==133)|(Index==138)|(Index==140)|(Index==161)|
   (Index==162)|(Index==164))
   {
                bg2[nextpixel]=0;
    //Delete Tracking line from bg1[]
    for(int j=0 ; j<wh ; j++)
                    bg1[j]=bg1[j]-bg2[j];
    //Clear bg2[]
    for(int j=0 ; j<wh ; j++)
                    bg2[j]=0;
    //Exit for loop
                th=8;
    }

   }//end Tracking line
   //Clear bg2[]
   for(int j=0 ; j<wh ; j++)
                bg2[j]=0;
  }//end found junction
 }
}//end round 3
//Decode bg1[]to bridgeE[]
for(int i=0; i<wh ; i++)
{
 if(bg1[i]==0)
        bridgeE[i]=255;
 else bridgeE[i]=0;
}
//Island Removal
int[]islandR =new int[wh];
//copy bridgeE[]to bg1[]and coding 0 to 1255 to 0
for(int i=0 ; i<wh ; i++)
{
 if(bridgeE[i]==0)
        bg1[i]=1;
 else bg1[i]=0;
```

```
}
//Scan for Endpoint exclude border
for(int i =w ; i <=((wh)-(w+1)); i++)
{
if(((((i%w ==0) | (i%w ==(w-1))))&(bg1[i]==1))
 {
          Index =bg1[i-w-1]+((bg1[i-w])*2)+((bg1[i-w+1])*4)+((bg1[i-1])*8)+((bg1[i+1])*16)
  +((bg1[i+w-1])*32)+((bg1[i+w])*64)+((bg1[i+w+1])*128);
 //Found initial ending
 if((Index==1)|(Index==2)|(Index==4)|(Index==8)|
 (Index==16)|(Index==32)|(Index==64)|(Index==128))
 {
            nextpixel=i;
 if(Index==1) nextpixel=i-w-1;   if(Index==2) nextpixel=i-w;
 if(Index==4) nextpixel=i-w+1;   if(Index==8) nextpixel=i-1;
 if(Index==16) nextpixel=i+1;     if(Index==32) nextpixel=i+w-1;
 if(Index==64) nextpixel=i+w;     if(Index==128) nextpixel=i+w+1;
            bg2[i]=1;
            bg2[nextpixel]=1;
 //Tracking line
 for(int th=0; th<10 ; th++)
 {
             Index =bg1[nextpixel-w-1]+((bg1[nextpixel-w])*2)+((bg1[nextpixel-w+1])*4)
  +((bg1[nextpixel-1])*8)+((bg1[nextpixel+1])*16)+((bg1[nextpixel-w-1])*32)
  +((bg1[nextpixel-w])*64)+((bg1[nextpixel+w+1])*128);

 //If topology is connectivity
 if((Index==66)|(Index==36)|(Index==129)|(Index==24)|
 (Index==130)|(Index==34)|(Index==68)|(Index==65)|
 (Index==17)|(Index==48)|(Index==12)|(Index==136)|
 (Index==160)|(Index==33)|(Index==5)|(Index==132))
 {
              Index2=bg2[(nextpixel-w-1)]+((bg2[nextpixel-w])*2)
  +((bg2[nextpixel-w+1])*4)+((bg2[nextpixel-1])*8)
  +((bg2[nextpixel+1])*16)+((bg2[nextpixel+w-1])*32)
  +((bg2[nextpixel+w])*64)+((bg2[nextpixel+w+1])*128);
 int diff =Index-Index2;
 if(diff==1)nextpixel =nextpixel-w-1;
 if(diff==2)nextpixel =nextpixel-w;
 if(diff==4)nextpixel =nextpixel-w+1;
 if(diff==8)nextpixel =nextpixel-1;
 if(diff==16)nextpixel =nextpixel+1;
 if(diff==32)nextpixel =nextpixel+w-1;
 if(diff==64)nextpixel =nextpixel+w;
 if(diff==128)nextpixel =nextpixel+w+1;
              bg2[nextpixel]=1;
 }//end connectivity

 //If topology is final ending
 if((Index==1)|(Index==2)|(Index==4)|(Index==8)|
 (Index==16)|(Index==32)|(Index==64)|(Index==128))
 {
              bg2[nextpixel]=1;
 //Delete Tracking line from bg1[]
 for(int j=0 ; j<wh ; j++)
                bg1[j]=bg1[j]-bg2[j];
 //Clear bg2[]
 for(int j=0 ; j<wh ; j++)
```

```
                    bg2[j]=0;
     //Exit for loop
                    th=10;
     }
    }//end Tracking line
    //Clear bg2[]
    for(int j=0 ; j<wh ; j++)
                bg2[j]=0;
   }//end found junction
   }
  }//end scan endpoint
 //Decode bg1[] to bridgeE[]
 for(int i=0; i<wh ; i++)
 {
  if(bg1[i]==0)
          islandR[i]=255;
  else islandR[i]=0;
 }
 return islandR;
}
}
```

5. Source Code ในส่วน Pixcrop

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import java.awt.image.*;
public class Pixcrop {
 int w,h,wh;
 double percent;
 int[] Showimg1 = new int[256*256];
 int[] Showimg2 = new int[256*256];
 String name,auther;
 public void Pixcrop(int[][] temppix2) throws Exception{
    ImgProcess image = new ImgProcess();
    ProgTool tool = new ProgTool();
    w = 256;
    h = 256;
    wh = w*h;
 int[][] temppix3 = new int[256][256];
 int[] pixel = new int[256*256];
 int temp=0;
 //Resixe Image from 128*128 to 256*256
 for(int i=0;i<256;i++)
 {
  for(int j=0;j<256;j++)
  {
        temppix3[i][j]=temppix2[(int)(i/2)][(int)(j/2)];
  }
 }
    pixel = tool.ArytoStrm(temppix3,w,h);
 for(int i=0;i<wh;i++)
 {
      Showimg1[i] = (255<<24)|(pixel[i]<<16)|(pixel[i]<<8)|(pixel[i]);
 }
    image.ImgPreProcess(pixel,w,h);
 int[] OputImg = new int[256*256];
    OputImg = image.ReturnImg();
```

```
 for(int i=0;i<(256*256);i++)
 {
  if(OputImg[i]==0)
         Showimg2[i] = (255<<24)|(0<<16)|(255<<8)|(0);
  if(OputImg[i]==125)
         Showimg2[i] = (255<<24)|(255<<16)|(0<<8)|(0);
  if(OputImg[i]==255)
         Showimg2[i] = (255<<24)|(255<<16)|(255<<8)|(255);
 }
    name = image.ReturnName();
    auther = image.ReturnAuther();
    percent = image.Returnpercent();
}

public String ReturnName()
{
 return name;
}
public String ReturnAuther()
{
 return auther;
}
public double Returnpercent()
{
 return percent;
}
public int[] ReturnImg()
{
 return Showimg;
}
public int[] ReturnImg2()
{
 return Showimg2;
}
}
```

5. Source Code ในส่วน ProgTool

```
import java.io.*;
import jmatlink.*;
public class ProgTool{
//Function Set Work Directory of MATLAB
public String setWorkdir(String dir)
{
 String file = "ProgTool.class";
 File findpath = new File(file);
 int L1 = file.length();
 String directory = findpath.getAbsolutePath();
 int L2 = directory.length();
 String path = directory.substring(0, L2-L1);
 String path1 = path.concat(dir);
    path1 = "cd "+path1+";";
 return path1;
}
//Function Change Stream to Array[][]
public int[][] StrmtoAry(int pixel[], int w, int h)
{
 int[][] temp = new int[h][w];
 for(int x=0 ; x<h ; x++)
 {
```

```
 for(int y=0 ; y<w ; y++)
 {
        temp[x][y]=pixel[(x*w)+y];
 }
 }
return temp;
}
//Function Change Array[][] to Stream
public int[] ArytoStrm(int pixel[][],int w,int h)
{
 int[]temp =new int[w*h];
 for(int x=0 ; x<h ; x++)
 {
  for(int y=0 ; y<w ; y++)
  {
        temp[(x*w)+y]=pixel[x][y];
  }
 }
return temp;
}
//Function Convert Int to Double
public double[][] InttoDoub(int inttemp[][],int w,int h)
{
 double[][]doubletemp =new double[h][w];
 for(int i=0 ; i<h ;i++)
 {
  for(int j=0 ; j<w ; j++)
  {
        doubletemp[i][j]=(double)inttemp[i][j];
  }
 }
return doubletemp;
}
//Function Convert Double to Int
public int[][] DoubtoInt(double doubletemp[][],int w,int h)
{
 int[][]inttemp =new int[h][w];
 for(int i=0 ; i<h ;i++)
 {
  for(int j=0 ; j<w ; j++)
  {
        inttemp[i][j]=(int)doubletemp[i][j];
  }
 }
return inttemp;
}
}
```