อุปกรณ์ตรวจวัดความชื้นและอุณหภูมิผ่านเครือข่ายอีเทอร์เนต

HUMIDITY AND TEMPERATURE MEASURING EQUIPMENT THROUGH ETHERNET NETWORK

โดย

นายปฐมพงศ์ อุ่นเรือน

นายประสงค์ ประชาชนะชัย

นายพงศธร คงศรีแก้ว

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2549

อุปกรณ์ตรวจวัดความชื้นและอุณหภูมิผ่านเครือข่ายอีเทอร์เนต

HUMIDITY AND TEMPERATURE MEASURING EQUIPMENT THROUGH ETHERNET NETWORK

โดย

นายปฐมพงศ์ อุ่นเรือน 46010405

นายประสงค์ ประชาชนะชัย 46010415

นายพงศธร คงศรีแก้ว 46010478

อาจารย์ที่ปรึกษา

รศ.ดร.ปราโมทย์ วาดเขียน

ผศ.ดร.จิรสุดา โกษียาภรณ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2549

ปริญญานิพนธ์ปีการศึกษา 2549

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง **อุปกรณ์ตรวจวัดความชื้นและอุณหภูมิผ่านเครือข่ายอีเทอร์เนต**

HUMIDITY AND TEMPERATURE MEASURING EQUIPMENT THROUGH ETHERNET NETWORK

ผู้จัดทำ

1. นายปฐมพงศ์ อุ่นเรือน 46010405

2. นายประสงค์ ประชาชนะชัย 46010415

3. นายพงศธร คงศรีแก้ว 46010478

..... อาจารย์ที่ปรึกษา

(รศ.ดร. ปราโมทย์ วาดเขียน)

..... อาจารย์ที่ปรึกษา

(ผศ.ดร.จีรยุดา โกษียาภรณ์)

อุปกรณ์ตรวจวัดความชื้นและอุณหภูมิผ่านเครือข่ายอีเทอร์เน็ต

HUMIDITY AND TEMPERATURE MEASURING EQUIPMENT THROUGH ETHERNET NETWORK

โดย นายปฐมพงศ์ อุ่นเรือน 46010405
นายประสงค์ ประชาชนะชัย 46010415
นายพงศธร คงศรีแก้ว 46010478

อาจารย์ที่ปรึกษา รศ.ดร.ปราโมทย์ วาดเขียน
ผศ.ดร.จีรสุดา โกษียาภรณ์

บทคัดย่อ

โครงงานนี้เป็นการนำเสนออุปกรณ์ตรวจวัดความชื้นและอุณหภูมิผ่านเครือข่ายอีเทอร์เน็ตโดยใช้ไอซีเบอร์ SHT-15 เป็นเซนเซอร์สำหรับวัดค่าความชื้นและอุณหภูมิในห้องหรือระบบหนึ่งๆ ซึ่งจะใช้ไมโครคอนโทรลเลอร์ในการประมวลผลแล้วแสดงค่าที่ได้บนจอแอลซีดี พร้อมทั้งส่งข้อมูลดังกล่าวไปยังเครื่องคอมพิวเตอร์ซึ่งมีหน้าที่เป็นศูนย์กลางผ่านอีเทอร์เน็ตแบบฝังตัว โดยฮาร์ดแวร์ดังกล่าวประกอบด้วยไอซี crystal LAN เบอร์ CS8900A-CQ , หม้อแปลงเบอร์ PM1005 รวมทั้งไมโครคอนโทรลเลอร์อีกตัวหนึ่งสำหรับควบคุมการรับและส่งข้อมูลระหว่างตัวเซนเซอร์กับคอมพิวเตอร์

ABSTRACT

This project presents the equipment which can be used to sensor the humidity and the temperature within particular room or specified areas through Ethernet Network by using IC SHT-15. The microcontroller also has been installed for processing and displaying the acquired humidity and temperature information on LCD monitor. Furthermore, these information can be sent to the centered computer through hardware called " Embedded Ethernet ". This hardware consists of IC crystal LAN CS8900A-CQ , transformer PM1005 and another microcontroller in order to control receiving and sending data between the sensor and the computer.

กิตติกรรมประกาศ

ขอขอบพระคุณ รศ.ดร.ปราโมทย์ วาดเขียน อาจารย์ที่ปรึกษาและพี่ๆห้อง project ทุกคนที่คอยช่วยให้คำปรึกษาและช่วยเหลือได้ตลอดเวลาไม่ว่าจะดึกแค่ไหน ขอขอบคุณเพื่อนร่วมกลุ่ม รวมทั้งร้านอาหารชงโคที่พวกเรารับประทานอาหารยามหิว

สารบัญ

หน้า

บทที่ 1 บทนำ 1

บทที่ 2 ทฤษฎีหรือหลักการ 4

2.1 แบบอ้างอิง OSI 4

2.1.1 ชั้นประยุกต์ (Application Layer) 5

2.1.2 ชั้นนำเสนอ (Presentation Layer) 5

2.1.3 ชั้นเซสชั่น (Session Layer) 5

2.1.4 ชั้นเคลื่อนย้ายข้อมูล (Transport Layer) 6

2.1.5 ชั้นเครือข่าย (Network Layer) 6

2.1.6 ชั้นเชื่อมโยงข้อมูล (Data Link Layer) 7

2.1.7 ชั้นกายภาพ (Physical Layer) 7

2.2 โปรโตคอล TCP/IP 8

2.3 Network Interface Layer 11

2.3.1 อีเทอร์เนต 11

2.3.2 เฟรมอีเทอร์เนต 13

2.3.3 MAC Address (Media Access Control Address) 13

2.3.4 Type field 13

2.3.5 Cyclic Redundancy Check (CRC) 14

2.3.6 IEEE 802.3 เฟรม 14

2.4 Internetwork Layer 22

2.4.1 โปรโตคอล IP (Internet Protocol) 22

2.4.2 ส่วนประกอบของ IP 22

2.4.3 IP Datagram 23

2.4.3a IP Header 24

2.4.3b IP Payload 25

2.4.4 Fragmentation และ Reassembly 26

2.4.5 Address Resolution Protocol (ARP) 26

2.4.5a ARP Datagram 27

2.4.5b กระบวนการทำงานของ ARP 28

2.4.6 Reverse Address Resolution Protocol (RARP) 30

2.4.7 Internet Control Message Protocol (ICMP) 31

2.5 Host – To – Host Layer 34

สารบัญ(ต่อ)

หน้า

2.5.1 พอร์ต 35

2.5.2 โปรโตคอล TCP 36

2.5.3 โปรโตคอล UDP 37

2.6 Process Layer 41

2.7 เซนเซอร์วัดความชื้นสัมพัทธ์และอุณหภูมิ 42

2.7.1 คุณสมบัติของเซนเซอร์ SHT-15 42

2.7.2 ขาสัญญาณสำหรับการสื่อสารข้อมูลของเซนเซอร์ SHT15 42

2.7.3 รูปแบบการสื่อสารข้อมูลของ SHT15 42

2.7.4 รีเซตการเชื่อมต่อ (Connection reset sequence) 43

2.7.5 ขั้นตอนการอ่านอุณหภูมิและความชื้นสัมพัทธ์ 44

2.7.6 การคำนวณค่าอุณหภูมิ 44

2.7.7 คำนวณค่าความชื้นสัมพัทธ์ 45

2.8 MICROCONTROLLER 45

2.8.1 โครงสร้างและรายละเอียด 45

2.8.2 การใช้พอร์ตอนุกรม 47

2.8.3 การใช้งานเป็นตัวตั้งเวลา (Timer) 48

2.8.4 การใช้งานเป็นเคาท์เตอร์ (Counter) 49

2.9 แอลซีดี (LCD) 50

บทที่ 3 การคำนวณและการสร้างวงจร 52

3.1 ส่วนประกอบฮาร์ดแวร์ของระบบ 52

3.2 ส่วนเชื่อมต่อเซนเซอร์วัดความชื้นและอุณหภูมิ SHT-15 และแสดงผลที่จอ LCD 53

3.3 ส่วนเชื่อมต่อระบบเครือข่าย 56

3.4 การเข้าถึงหน่วยความจำของระบบ 58

3.5 กระบวนการส่งและรับข้อมูลของ Ethernet Controller 59

3.6 กระบวนการส่งและรับข้อมูลของระบบ 63

3.7 การควบคุม Ethernet Controller และเครื่องคอมพิวเตอร์ 64

บทที่ 4 การทดลองและผลการทดลอง 68

4.1 การทดลองที่ 1 เป็นการทดลองการเชื่อมต่อของวงจรทั้งหมดซึ่งประกอบด้วยส่วนของวงจรเชื่อมต่อไมโครคอนโทรลเลอร์ตัวที่ 1 กับเซนเซอร์ และไมโครคอนโทรลเลอร์ตัวที่ 2 กับ Ethernet Controller พร้อมส่งไปยังคอมพิวเตอร์ ที่มี IP Address อยู่ในวง LAN เดียวกัน 68

สารบัญ(ต่อ)

หน้า

4.2 การทดลองที่ 2 เป็นการทดลองการเชื่อมต่อของวงจรทั้งหมดซึ่งประกอบด้วยส่วนของวงจรเชื่อมต่อไมโครคอนโทรลเลอร์ตัวที่ 1 กับเซนเซอร์ และไมโครคอนโทรลเลอร์ตัวที่ 2 กับ Ethernet Controller พร้อมส่งไปยังคอมพิวเตอร์ที่มี IP Address ไม่ได้อยู่ในวง LAN เดียวกัน 78

บทที่ 5 บทวิจารณ์และบทสรุป 87

ภาคผนวก

กิตติกรรมประกาศ

หนังสืออ้างอิง

สารบัญรูปภาพ

หน้า

รูปที่ 2.1 แสดงชั้นของการเชื่อมต่อระหว่างคอมพิวเตอร์ 2 เครื่อง 4
รูปที่ 2.2 แสดงกลไกของโปรโตคอลมาตรฐาน OSI model 9
รูปที่ 2.3 Data Encapsulation 10
รูปที่ 2.4 แสดงโครงสร้าง Data Structures 11
รูปที่ 2.5 ลักษณะของเฟรมอีเทอร์เนต 13
รูปที่ 2.6 อีเทอร์เนตประเภทต่าง ๆ 15
รูปที่ 2.7 แสดงลักษณะโครงสร้างของเฟรมข้อมูลตามมาตรฐาน IEEE 802.3 16
รูปที่ 2.8 แสดงลักษณะส่วนการทำงานภายในของ Preamble 17
รูปที่ 2.9 แสดงส่วนประกอบของ IP 23
รูปที่ 2.10 แสดง IP Datagram ซึ่งประกอบด้วย IP Header และ IP Payload 23
รูปที่ 2.11 แสดงโครงสร้าง IP Header 24
รูปที่ 2.12 แสดง ARP Datagram 27
รูปที่ 2.13 แสดงตัวอย่างกระบวนการของ ARP , (a) ARP Request , (b) ARP Response 30
รูปที่ 2.14 แสดง ICMP encapsulated ใน IP และประเภทของ ICMP 33
รูปที่ 2.15 แสดงรูปแบบของ ICMP Datagram 33
รูปที่ 2.16 แสดงการใช้งาน port ของแต่ละโปรโตคอล 34
รูปที่ 2.17 แสดงการส่งข้อมูลจาก Application ไปยัง Host – to – Host Layer 35
รูปที่ 2.18 แสดง Ports ที่ใช้กับ UDP และ TCP 35
รูปที่ 2.19 แสดงโครงสร้างของโปรโตคอล TCP 36
รูปที่ 2.20 แสดงกลไกการส่งข้อมูลด้วยโปรโตคอล UDP 37
รูปที่ 2.21 UDP Message encapsulation 38
รูปที่ 2.22 แสดงโครงสร้างของ UDP Header 38
รูปที่ 2.23 แสดง Pseudo Header 40
รูปที่ 2.24 แสดงรูปร่างเซนเซอร์ SHT15 และการจัดขาเพื่อต่อใช้งาน 42
รูปที่ 2.25 แสดงรูปแบบสัญญาณกระตุ้นผ่านขาสัญญาณ SCK และ DATA (Transmission start) 43
รูปที่ 2.26 แสดงรายละเอียดคำสั่งและข้อมูลสำหรับควบคุมการทำงานของเซนเซอร์ SHT-15 43
รูปที่ 2.27 แสดงขาของ MCS-51 47
รูปที่ 2.28 แสดงภาพแอลซีดีโมดูลที่นำมาใช้ในโครงงานนี้ 50
รูปที่ 3.1 แสดงส่วนประกอบหลักของฮาร์ดแวร์ทั้งหมด 52
รูปที่ 3.2 แสดงวงจรที่ใช้เชื่อมต่อไมโครคอนโทรลเลอร์และเซนเซอร์ SHT-15 53

สารบัญรูปภาพ(ต่อ)

หน้า

รูปที่ 3.3 แสดง Flow Chart ในการรับส่งค่าข้อมูลความชื้นและอุณหภูมิระหว่าง ไมโครคอนโทรลเลอร์กับเซนเซอร์ SHT-15 และแสดงผลที่จอ LCD 54

รูปที่ 3.4 แสดง Flow Chart ในการรับส่งค่าข้อมูลความชื้นและอุณหภูมิระหว่าง ไมโครคอนโทรลเลอร์กับเซนเซอร์ SHT-15 และแสดงผลที่จอ LCD พร้อมทั้งส่งข้อมูล ผ่านพอร์ตอนุกรม 55

รูปที่ 3.5 แสดงวงจรส่วนเชื่อมต่อระบบเครือข่าย 56

รูปที่ 3.6 แสดง PIN OUT ของวงจรควบคุมการเชื่อมต่อระบบเครือข่าย 57

รูปที่ 3.7 แสดงวงจรที่เชื่อมต่อกันระหว่างไมโครคอนโทรลเลอร์กับ Ethernet Controller 57

รูปที่ 3.8 แสดง Flow Chart ในการส่งข้อมูลของ Ethernet Controller 60

รูปที่ 3.9 แสดง Flow Chart ในการรับข้อมูลของ Ethernet Controller 61

รูปที่ 3.10 แสดงรูปวงจรรวมทั้งหมด 62

รูปที่ 3.11 แสดงกระบวนการส่งข้อมูลของระบบ 63

รูปที่ 3.12 แสดงกระบวนการรับข้อมูลของระบบ 64

รูปที่ 3.13 แสดง Flow Chart ของโปรแกรมที่ใช้ใน Ethernet Controller 66

รูปที่ 3.14 แสดง Flow Chart ของโปรแกรมที่ใช้ในเครื่องคอมพิวเตอร์ 67

รูปที่ 4.1 แสดงกระบวนการรับส่งข้อมูลระหว่าง Ethernet Controller กับคอมพิวเตอร์ที่มี IP อยู่ในLAN เดียวกัน 68

รูปที่ 4.2 แสดง MAC Address ใน Cache Memory 69

รูปที่ 4.3 แสดงผลของเฟรม ICMP Echo Request ที่ Ethernet Controller (IP 161.246.18.157) ส่งมาตรวจสอบสถานะการเชื่อมต่อกับคอมพิวเตอร์ (IP 161.246.18.158) 70

รูปที่ 4.4 แสดงผลของเฟรม ARP Request ที่คอมพิวเตอร์ส่งแบบ Broadcast มาเพื่อถาม MAC Address ของ Ethernet Controller IP 161.246.18.157 71

รูปที่ 4.5 แสดงผลของเฟรม ARP Reply ที่ Ethernet Controller ซึ่งมี IP 161.246.18.157 ส่งไปยังเครื่องคอมพิวเตอร์เพื่อบอก MAC Address (00-11-D8-67-1F-1D) ของตัวมันเอง 72

รูปที่ 4.6 แสดงผลของเฟรม ICMP Echo Reply ที่คอมพิวเตอร์ส่งกลับไปให้ Ethernet Controller เพื่อยืนยันเชื่อมต่อ และพร้อมที่จะรับข้อมูลจาก Ethernet Controller แล้ว 73

รูปที่ 4.7 แสดงเฟรม UDP ที่ถูกส่งจาก Ethernet Controller ไปยังคอมพิวเตอร์ซึ่งมีข้อมูลของ อุณหภูมิและความชื้นแนบอยู่ 74

รูปที่ 4.8 แสดง MAC Address ใน Cache Memory ของ Ethernet Controller (IP 161.246.18.157) หลังจากได้รับ ARP Reply 75

สารบัญรูปภาพ(ต่อ)

หน้า

รูปที่ 4.9 แสดงโปรแกรม Ethernet_Controller.java ที่รันออกมาบนเครื่องคอมพิวเตอร์ (IP 161.246.18.158 Port 6000) ในขณะที่ยังไม่มีการเชื่อมต่อกับ Ethernet Controller 76

รูปที่ 4.10 (ก) , (ข) และ (ค) แสดงค่าของอุณหภูมิและความชื้นที่เปลี่ยนแปลงเมื่อได้ทำการเชื่อมต่อคอมพิวเตอร์กับ Ethernet Controller 77

รูปที่ 4.11 แสดงกระบวนการรับส่งข้อมูลระหว่าง Ethernet Controller กับคอมพิวเตอร์ที่มี IP ไม่ได้อยู่ในวง LAN เดียวกัน 78

รูปที่ 4.12 แสดงผลของเฟรม ICMP Echo Request ที่ Ethernet Controller (IP 161.246.18.157) ส่งมาตรวจสอบสถานะการเชื่อมต่อกับคอมพิวเตอร์ (IP 161.246.31.103) 79

รูปที่ 4.13 แสดงผลของเฟรม ICMP Echo Reply ที่คอมพิวเตอร์ส่งกลับไปให้ Ethernet Controller เพื่อยืนยันการเชื่อมต่อ และพร้อมที่จะรับข้อมูลจาก Ethernet Controller แล้ว 80

รูปที่ 4.14 แสดงเฟรม UDP ที่ถูกส่งจาก Ethernet Controller ไปยังคอมพิวเตอร์ซึ่งมีข้อมูลของอุณหภูมิและความชื้นแนบอยู่ 81

รูปที่ 4.15 แสดงโปรแกรม Ethernet_Controller.java ที่รันออกมาบนเครื่องคอมพิวเตอร์ (IP 161.246.31.103 Port 6000) ในขณะที่ยังไม่มีการเชื่อมต่อกับ Ethernet Controller 82

รูปที่ 4.16 (ก) , (ข) และ (ค) แสดงค่าของอุณหภูมิและความชื้นที่เปลี่ยนแปลงเมื่อได้ทำการเชื่อมต่อคอมพิวเตอร์กับ Ethernet Controller 83

รูปที่ 4.17 แสดงบอร์ดวงจรทั้งหมดขณะลงกล่อง 84

รูปที่ 4.18 (ก) , (ข) แสดงรูปชิ้นงาน 85

รูปที่ 4.19 แสดงการเชื่อมต่อชิ้นงานกับคอมพิวเตอร์ 85

รูปที่ 4.20 แสดงรูปผลบนหน้าจอคอมพิวเตอร์ 86

รูปที่ 4.21 แสดงรูปผลบนหน้าจอ LCD ของชิ้นงาน 86

สารบัญตาราง

หน้า

ตารางที่ 2.1 แสดงชุดโปรโตคอล OSI ที่ทำงานในแต่ละเลเยอร์ 4

ตารางที่ 2.2 แสดง Model ของโปรโตคอล TCP/IP 9

ตารางที่ 2.3 Ethernet type field 14

ตารางที่ 2.4 แสดงตัวอย่างของ Source Address ที่แสดงรหัสแอดเดรสของผู้ผลิต 19

ตารางที่ 2.5 แสดงตัวอย่างของรหัสที่ใช้แสดงแทนโปรโตคอลที่ใช้ในช่อง Type 18 รายการ 20

ตารางที่ 2.6 แสดง UDP Port Numbers 40

ตารางที่ 2.7 แสดงรายละเอียดคำสั่งและข้อมูลสำหรับควบคุมการทำงานของเซนเซอร์ SHT-15 43

ตารางที่ 2.8 แสดงค่าเวลาที่เซนเซอร์ SHT-15 ต้องใช้ในการประมวลผลข้อมูล 44

ตารางที่ 2.9 แสดงการกำหนดค่าคงที่ทางอุณหภูมิตัวที่ 1และตัวที่ 2เพื่อคำนวณค่าอุณหภูมิจริงที่วัดได้ 44

ตารางที่ 2.10 แสดงการกำหนดค่าคงที่ซึ่งต้องใช้ในการคำนวณค่าความชื้นสัมพัทธ์จริงที่วัดได้ 45

ตารางที่ 2.11 แสดงคำสั่งที่ใช้ควบคุมแอลซีดี 50

ตารางที่ 3.1 แสดง I/O Mode Mapping 58

บทที่ 1
บทนำ

ความเป็นมาและคำจำกัดความต่างๆที่กล่าวถึงในบทนี้ ซึ่งเป็นข้อแนะนำประกอบการอ่านรายงาน Project เพื่อให้ผู้อ่านสามารถเข้าใจถึงส่วนประกอบต่างๆโดยจะกล่าวถึงคำจำกัดความและข้อควรทราบที่เกี่ยวข้องกับอุปกรณ์ตัววัดอุณหภูมิและความชื้น ซึ่งเชื่อมต่อเครือข่ายอีเทอร์เนต (Ethernet) โดยจะทำการเก็บค่าที่วัดได้และแสดงผลบนคอมพิวเตอร์

ความเป็นมา

ในปัจจุบันโรงงานเกษตรกรรมและอุตสาหกรรมส่วนใหญ่ได้มีการติดตั้งอุปกรณ์ตรวจวัดอุณหภูมิและความชื้นเพื่อป้องกันอันตรายต่าง ๆ ที่เกิดขึ้นเช่น ในแปลงเกษตร จะต้องการอุณหภูมิและความชื้นค่าหนึ่ง ถ้าเกินค่านี้จะทำให้เกิดความเสียหายของพืชผักสวนครัวได้ หรือการผลิตอุปกรณ์ทางอิเล็กทรอนิกส์ต้องการอุณหภูมิและความชื้นที่เหมาะสมในการทำงาน ถ้าเกินค่าดังกล่าวแล้วอุปกรณ์จะไม่สามารถทำงานได้ โดยค่าที่อ่านได้ในอุปกรณ์ตรวจวัดอุณหภูมิและความชื้นนี้สามารถที่จะใช้ข้อมูลร่วมกันได้หลายๆเครื่อง เนื่องจากมีประโยชน์ที่สามารถดูค่าตรวจวัดอุณหภูมิและความชื้นจากหลายๆเครื่องซึ่งเชื่อมต่อกับคอมพิวเตอร์หลัก (Computer Server) คือ เซิร์ฟเวอร์ที่สามารถจัดการเกี่ยวกับการเข้าดูค่าตรวจวัดอุณหภูมิและความชื้นของเครือข่าย ซึ่งเซิร์ฟเวอร์ไม่จำเป็นต้องเป็นเครื่องคอมพิวเตอร์ก็ได้ แต่อาจจะเป็นเพียงอุปกรณ์ที่มีขนาดเล็กและสามารถเชื่อมต่อกับอุปกรณ์ตรวจวัดอุณหภูมิและความชื้นกับเครือข่ายได้ง่าย โดยมีไมโครคอนโทรลเลอร์เป็นตัวควบคุมแทน ซึ่งอุปกรณ์นี้จะอยู่ในรูปของระบบควบคุมแบบฝังตัว (Embedded system)

Embedded system เป็นระบบอิเล็กทรอนิกส์ที่ใช้สำหรับงานควบคุมรวมถึงการแสดงผลการทำงานต่างๆ โดยที่ระบบเหล่านี้ถูกใช้เป็นส่วนหนึ่งของระบบและอุปกรณ์ควบคุมเครื่องมือ เครื่องจักรต่างๆ การที่ใช้คำว่า “ระบบแบบฝังตัว” เนื่องจากระบบเหล่านี้เป็นส่วนหนึ่งของระบบใหญ่ ในหลายกรณีที่ผู้ใช้ทั่วไปอาจไม่ทราบว่าอุปกรณ์ควบคุมเครื่องมือ เครื่องจักรรวมถึงระบบใดระบบหนึ่งที่ใช้งานเป็นประจำเหล่านั้นเป็นระบบแบบฝังตัว ในบางครั้งแม้แต่ผู้ที่มีความรู้ทางด้านเทคนิคก็ไม่สามารถระบุได้แน่ชัดว่าอุปกรณ์ควบคุมใดมีระบบแบบฝังตัวอยู่ จนกว่าจะมีการทำงานและตรวจสอบกับระบบและอุปกรณ์ควบคุมนั้นระยะหนึ่งเลยทีเดียว

ระบบแบบฝังตัว (Embedded system) นี้แม้จะไม่ใช่เครื่องคอมพิวเตอร์แต่ก็มีระบบคอมพิวเตอร์อยู่ภายใน อาจจะเป็นเพียงไมโครโพรเซสเซอร์ (Microprocessor) หรือชิป (Chip) ธรรมดาหรือโพรเซสเซอร์ (Processor) ที่ประกอบไปด้วย ชิป (Chip) ที่มีวงจรซับซ้อน โดยจะมีหลักการทำงานคือมีสัญญาณข้อมูลเข้า (Input) จากอุปกรณ์เซนเซอร์ (Sensor) เข้าสู่ระบบ และมีสัญญาณผลลัพธ์ (Output) ของระบบไปควบคุมบังคับสวิตซ์เครื่องควบคุมต่างๆ เช่น สวิตซ์เครื่องจักร หรือ วาล์วควบคุมทิศทางการไหลของท่อน้ำในทิศทางต่างๆ นอกจากนี้แบบและรุ่นของระบบแบบฝังตัว (Embedded system) ก็มีมากมายมีทั้งระบบที่เป็นแบบง่ายๆ การทำงานไม่ซับซ้อนตลอดจนแบบระบบที่ซับซ้อน ซึ่งขึ้นอยู่กับประเภทและจำนวนไมโครโพรเซสเซอร์ รวมไปถึงงานโปรแกรมควบคุมในระบบ

ประเภทของ Embedded system

1. แบบไมโครโพรเซสเซอร์เดี่ยวเป็นระบบซึ่งใช้อยู่ในอุปกรณ์ขนาดเล็ก เช่น อุปกรณ์ไฟฟ้า วงจรไฟฟ้าต่างๆ เครื่องตรวจจับต่างๆ

2. แบบไมโครโพรเซสเซอร์หลายตัวรวมกันในวงจร ซึ่งเป็นระบบที่ใช้อยู่ในอุปกรณ์ควบคุมที่ซับซ้อน เช่น อุปกรณ์ควบคุมการไหลของแก๊สของเหลว กระแสไฟฟ้า อุปกรณ์ขยายสัญญาณต่าง ๆ อุปกรณ์ปิดวาล์ว เครื่องควบคุมเครื่องจักรในโรงงาน ซึ่งจะมีทั้งที่ไม่ทำหน้าที่เกี่ยวกับเวลาและทำหน้าที่เกี่ยวกับเวลา

เนื่องจากระบบควบคุมแบบฝังตัวเป็นระบบที่เหมาะสมในการควบคุมงานเฉพาะทาง แต่ระบบเหล่านี้มักจะเป็นระบบปิด ทำให้การควบคุมและใช้งานต้องกระทำผ่านเครื่องมือ หรืออุปกรณ์เฉพาะของผู้ออกแบบ หรือผู้ผลิตแต่ละรายเท่านั้น แต่อย่างไรก็ตามการควบคุมจำเป็นจะต้องมีซอฟต์แวร์พิเศษซึ่งมักยึดติดกับระบบนั้น หรือระบบปฏิบัติการที่ใช้ควบคุม

Ethernet เป็นมาตรฐานการส่งข้อมูลที่อนุญาตให้เครื่องคอมพิวเตอร์ใช้ช่องสัญญาณร่วมกันโดยผลัดกันใช้ อุปกรณ์ที่ใช้ในการส่งสัญญาณของอินเทอร์เน็ตนั้นก็คือการ์ดแลน สายแลนและอุปกรณ์รวมสัญญาณ ถ้าเป็นการเชื่อมต่อแบบ Bus จะใช้สายสัญญาณกลางหรือแบ็คโบนเป็นตัวรวมสัญญาณ แต่ถ้าเป็นการเชื่อมต่อแบบ Star จะใช้ฮับเป็นอุปกรณ์รวมสัญญาณ ในปัจจุบันนิยมใช้การเชื่อมต่อ Ethernet แบบ Star มากเนื่องจากความเร็วในการส่งข้อมูลและความสะดวกในการดูแลรักษา

ระบบเครือข่าย Ethernet หมายถึง มาตรฐานในการเชื่อมต่อคอมพิวเตอร์หลายเครื่อง (ตั้งแต่ 2 เครื่องขึ้นไป) เข้าด้วยกันเป็นระบบเครือข่าย สำหรับปฏิบัติการในแต่ละจุด (เช่นตามบ้านหรือสำนักงานต่างๆ) หรือ Local Area Network ซึ่งนิยมเรียกกันสั้นๆว่า LAN รวมทั้งระบบการสื่อสารที่ช่วยให้คอมพิวเตอร์เหล่านั้นสามารถใช้ข้อมูลและโปรแกรมต่างๆร่วมกันได้ด้วย

ในกรณีที่คอมพิวเตอร์หลายเครื่องในห้องเดียวกันหรืออาคารเดียวกัน ระบบเครือข่าย Ethernet จะสามารถช่วยเพิ่มประสิทธิภาพการทำงานของคอมพิวเตอร์เหล่านั้นได้ เพราะหลังจากการสร้างระบบเครือข่าย Ethernet ขึ้นมาแล้ว ข้อมูลจะสามารถถ่ายโอนระหว่างเครื่องคอมพิวเตอร์ทั่วไปกับเครื่องเซิร์ฟเวอร์ได้รวดเร็วขึ้นมาก อีกทั้งยังสามารถสั่งพิมพ์งานผ่านพรินเตอร์หรือใช้โปรแกรมต่างๆ รวมทั้งระบบต่อเชื่อมอินเตอร์เน็ตร่วมกันระหว่างคอมพิวเตอร์ทุกเครื่องในเครือข่ายนั้นด้วย

จนถึงบัดนี้ระบบเครือข่ายนี้ก็ยังนับเป็นระบบยอดนิยมสำหรับธุรกิจน้อยใหญ่ทำให้เครือข่าย Ethernet กลายเป็นระบบมาตรฐานอย่างหนึ่งในการสร้างระบบเครือข่ายคอมพิวเตอร์ในปัจจุบัน แต่ระบบเครือข่าย Ethernet ก็เป็นอะไรที่มากกว่าแค่อุปกรณ์ฮาร์ดแวร์เพราะมันยังรวมถึงรูปแบบในการสื่อสารและการถ่ายโอนข้อมูลต่างๆของคอมพิวเตอร์ที่เชื่อมโยงกันนั้นด้วย ทั้งนี้คอมพิวเตอร์ที่ต่อเชื่อมด้วยระบบ Ethernet นี้จะส่งข้อมูลไปตามสายในรูปแบบของกลุ่มข้อมูลขนาดเล็กที่เรียกว่า packets โดยใน packet นั้นนอกจากจะมีข้อมูลต่างๆ แล้ว ยังมีข้อมูลเกี่ยวกับที่อยู่ของคอมพิวเตอร์ที่เกี่ยวข้องกับการรับ-ส่งข้อมูลต่างๆด้วย

คำจำกัดความที่ใช้ใน Project

คำเหล่านี้คือ คำจำกัดความที่จะกล่าวถึงภายใน Project นี้ โดยบางคำอาจจะมีความหมายครอบคลุมถึงระบบที่ใกล้เคียงหรือระบบที่เฉพาะเจาะจง

อีเทอร์เนต (Ethernet)

มาตรฐานในการเชื่อมต่อระบบเครือข่ายท้องถิ่นที่ได้รับความนิยมมากที่สุด ซึ่งใน Project นี้ จะถือว่าการเชื่อมต่อเครือข่ายคือ การเชื่อมต่อผ่านระบบสายสัญญาณตามมาตรฐาน Ethernet ลักษณะโดยทั่วไปของอีเทอร์เนตคือ การทำงานในลักษณะ Broadcast ที่จุดเชื่อมต่อต่างๆ สามารถรับส่งข้อมูลได้ตลอดเวลา

PROTOCOL

โปรโตคอล (Protocol) เป็นมาตรฐานในการสื่อสารข้อมูลของคอมพิวเตอร์ หรืออาจกล่าวได้ว่าโปรโตคอลเป็น "ภาษา" ที่คอมพิวเตอร์ใช้สื่อสารกัน ดังนั้นคอมพิวเตอร์ที่ต้องการสื่อสารกันจำเป็นที่ต้องใช้ "ภาษา" หรือโปรโตคอลเดียวกัน เพราะไม่เช่นนั้นคอมพิวเตอร์ก็จะสื่อสารกันไม่ได้

TCP/IP

Transmission Control Protocol / Internet Protocol เป็นโปรโตคอลที่กำหนดการรับส่งข้อมูลระหว่างจุดบนระบบเครือข่าย ซึ่งคุณลักษณะเฉพาะ TCP/IP คือ ความสามารถในการยืนยันความถูกต้องของการรับส่ง และการได้รับข้อมูลบนระบบเครือข่าย

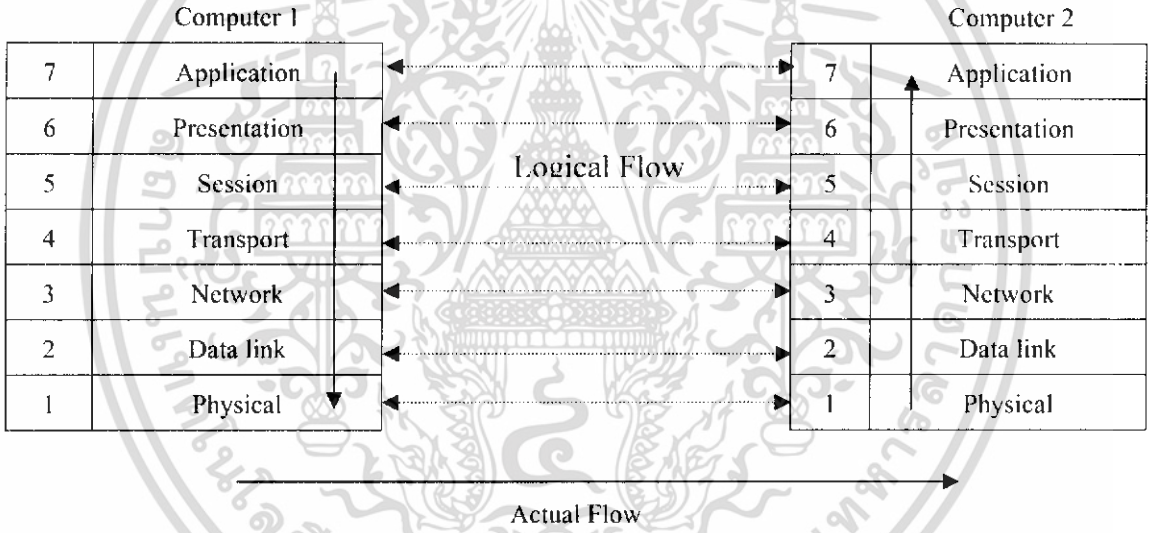
UDP

UDP เป็นโปรโตคอลที่ถูกออกแบบมาให้ทำหน้าที่รับส่งข้อมูลโดยมีขั้นตอนการทำงานไม่ซับซ้อนและทำงานได้รวดเร็ว แต่มีจุดด้อยคือไม่มีความน่าเชื่อถือ คือ เป็นการสื่อสารที่ไม่ต้องทำการตอบกลับ (connectionless) ทำให้ไม่สามารถยืนยันความถูกต้องของข้อมูลระหว่างเครื่องเซิร์ฟเวอร์ที่ให้บริการกับเครื่องที่ขอใช้บริการได้ โปรโตคอล UDP ทำงานในชั้น Transport Layer ซึ่งจะต้องพึ่งพาโปรโตคอล IP ในการรับส่งข้อมูล

บทที่ 2
ทฤษฎีหรือหลักการ

2.1 แบบอ้างอิง OSI

องค์กรมาตรฐานนานาชาติ (The International Organization for Standardization) และใช้อักษรย่อว่า ISO เป็นองค์กรที่ออกแบบโปรโตคอล OSI (Open System Interconnect) หรือโปรโตคอลการเชื่อมเครือข่ายแบบเปิด จุดมุ่งหมายของการพัฒนามาตรฐานนี้เพื่อให้คอมพิวเตอร์และอุปกรณ์เครือข่ายที่ผลิตโดยบริษัทต่างๆสามารถทำงานร่วมกันได้ ชุดโปรโตคอลนี้ส่วนใหญ่จะเรียกว่า แบบอ้างอิง OSI (OSI Reference Model) เหตุที่เรียกแบบอ้างอิงนี้เนื่องจากโปรโตคอลนี้ไม่ได้เป็นที่ถูกใช้กันอย่างแพร่หลาย เช่น ในเครือข่ายอินเตอร์เน็ตแต่เนื่องจากแบบอ้างอิง OSI มีการออกแบบโครงสร้างที่ค่อนข้างสมบูรณ์ด้วยเหตุนี้จึงใช้โปรโตคอลชุดนี้เป็นแบบอ้างอิงในการพัฒนาโปรโตคอลชุดอื่นๆ อีกทั้งยังเป็นระบบที่ง่ายต่อการอธิบายกลไกการทำงานของโปรโตคอลในเครือข่าย ดังนั้นเมื่อกล่าวถึงเครือข่ายส่วนใหญ่จะใช้โปรโตคอลนี้เป็นแบบอ้างอิงในการอธิบายซึ่งก็เป็นที่มาของการเรียกโปรโตคอลชุดนี้ว่าแบบอ้างอิง OSI นั่นเอง



รูปที่ 2.1 แสดงชั้นของการเชื่อมต่อระหว่างคอมพิวเตอร์ 2 เครื่อง

ชุดโปรโตคอล OSI ประกอบด้วยโปรโตคอลมาตรฐานหลายโปรโตคอล โปรโตคอลเหล่านี้เป็นส่วนหนึ่งของโครงการนานาชาติเพื่อพัฒนาโปรโตคอลและมาตรฐานอื่นๆ เพื่อเอื้ออำนวยให้อุปกรณ์เครือข่ายที่ผลิตจากบริษัทต่างๆสามารถทำงานร่วมกันได้ ข้อกำหนดมาตรฐาน OSI ถูกจัดทำโดย 2 องค์กรคือ ISO และ ITU-T

ตารางที่ 2.1 แสดงชุดโปรโตคอล OSI ที่ทำงานในแต่ละเลเยอร์

OSI Reference Model		**OSI Protocols**
7	Application	CMIP, DS, FTAM, MHS, VTP

6	Presentation	Presentation Service/ Presentation Protocol
5	Session	Session Service/ Session Protocol
4	Transport	TP0, TP1, TP2, TP3, TP4, TP5
3	Network	CONP/CMNS, CLNP/CLNS, IS-IS, ES-IS
2	Data link	IEEE 802.2, IEEE 802.3, IEEE 802.5, FDDI, X.25
1	Physical	IEEE 802.2 Hardware, IEEE 802.3 Hardware, IEEE 802.5 Hardware, FDDI, X.25 Hardware

2.1.1 ชั้นประยุกต์ (Application Layer)

โปรโตคอลชั้นที่อยู่บนสุดของแบบอ้างอิง OSI คือ ชั้นประยุกต์ (Application Layer) ถึงแม้ว่าชื่อจะเป็นแอพพลิเคชั่นเลเยอร์แต่ก็ไม่ได้รวมเอาแอพพลิเคชั่นของผู้ใช้ด้วย (User Application) แต่โปรโตคอลชั้นนี้จะเป็นจุดเชื่อมต่อระหว่างแอพพลิเคชั่นของผู้ใช้กับกระบวนการการสื่อสารผ่านเครือข่าย ชั้นนี้อาจจะถือได้ว่าเป็นชั้นที่เริ่มกระบวนการติดต่อสื่อสาร เช่น เมื่อผู้ใช้ต้องการส่งอีเมล์ โปรแกรมที่ผู้ใช้ใช้ส่งอีเมล์จะทำการติดต่อกับโปรโตคอลในชั้นประยุกต์เพื่อเริ่มกระบวนการทั้งหมด

2.1.2 ชั้นนำเสนอ (Presentation Layer)

โปรโตคอลในชั้นนี้จะรับผิดชอบเกี่ยวกับรูปแบบของข้อมูลที่รับส่งผ่านเครือข่าย เนื่องจากคอมพิวเตอร์ที่ต้องการแลกเปลี่ยนข้อมูลกันนั้นอาจมีวิธีการเข้ารหัส (Encoding) ที่ต่างกัน เช่น คอมพิวเตอร์บางเครื่องอาจใช้การเข้ารหัสแบบ ASCII (American Standard Code for Information Interchange) หรือบางเครื่องอาจใช้การเข้ารหัสแบบ EBCDIC (Extended Binary Coded Decimal Interchange Code) ดังนั้นก่อนการส่งข้อมูลโปรโตคอลในเลเยอร์นี้ก็จะแปลงข้อมูลให้อยู่ในรูปแบบที่เป็นมาตรฐาน ส่วนทางด้านฝ่ายรับก็จะทำการแปลงกลับไปให้เป็นรูปแบบที่คอมพิวเตอร์เครื่องนั้นเข้าใจ นอกจากนี้ เลเยอร์นี้ยังรับผิดชอบในการทำให้ข้อมูลที่เข้ารหัสเลขฐานิยมที่ต่างกันสามารถแลกเปลี่ยนข้อมูลกันได้

2.1.3 ชั้นเซสชั่น (Session Layer)

เซสชั่นเลเยอร์ ทำหน้าที่ควบคุมการสื่อสารผ่านเครือข่ายที่กำลังเกิดขึ้นระหว่างสองฝั่ง การสื่อสารที่กำลังเป็นไปในช่วงชั่วขณะใดขณะหนึ่งจะเรียกว่า "เซสชั่น (Session)" แอพพลิเคชั่นทั้งสองฝั่งสามารถแลกเปลี่ยนข้อมูลหรือรับส่งแพ็กเก็ต (packet) ถึงกันและกันได้ในช่วงเวลาที่เซสชั่นยังอยู่ โดย

เซสชั่นเลเยอร์จะรับผิดชอบเกี่ยวกับการสร้างเซสชั่น ควบคุมการแลกเปลี่ยนข้อมูล และยกเลิกเซสชั่นเมื่อการสื่อสารสิ้นสุด

เซสชั่นเลเยอร์สามารถกำหนดรูปแบบการสื่อสารว่าเป็นแบบทางเดียว (Unidirectional) หรือแบบสองทาง (Bi-directional) ในการสื่อสารแบบสองทางนั้นยังสามารถกำหนดได้ว่าการไหลของข้อมูลนั้นเป็นไปได้สองทางพร้อมกันหรือไหลได้ทางเดียวในขณะใดขณะหนึ่ง ในการจัดการเกี่ยวกับการที่ให้ข้อมูลไหลได้ทางเดียวในขณะใดขณะหนึ่งนั้นก็โดยการใช้โทเคน (Token) เฉพาะฝั่งที่มีโทเคนเท่านั้นถึงจะมีสิทธิ์ส่งแพ็กเก็ต อีกฟังก์ชั่นหนึ่งก็คือ การควบคุมจังหวะการรับส่งข้อมูล (Synchronization) ฟังก์ชั่นนี้มีประโยชน์ในกรณีอย่างเช่น สมมติว่ากำลังมีการถ่ายโอนไฟล์ระหว่างสองเครื่อง แล้วเครื่องใดเครื่องหนึ่งเกิดล้มเหลวกะทันหัน การถ่ายโอนไฟล์นั้นอาจต้องเริ่มใหม่ แต่เซสชั่นเลเยอร์มีฟังก์ชั่นที่กำหนดจุดเริ่มต้นของกระบวนการถ่ายโอนไฟล์ได้ โดยเมื่อเปิดเครื่องใหม่ก็เริ่มกระบวนการถ่ายโอนไฟล์ต่อจากเมื่อคราวก่อนหน้านี้ได้

2.1.4 ชั้นเคลื่อนย้ายข้อมูล (Transport Layer)

ชั้นเคลื่อนย้ายข้อมูล จะรับผิดชอบในการเคลื่อนย้ายข้อมูลระหว่างโพรเซสส์ของผู้รับกับโพรเซสส์ของผู้ส่ง โพรเซสส์ในที่นี้จะหมายถึงโปรแกรมที่กำลังรันบนเครื่องคอมพิวเตอร์ ซึ่งในขณะใดขณะหนึ่งอาจมีหลายโพรเซสส์ที่กำลังรันอยู่ ดังนั้นชั้นนี้จะรับผิดชอบในการรับส่งข้อมูลให้ถึงโพรเซสส์ที่ต้องการ หน้าที่อีกอย่างของโปรโตคอลในชั้นนี้คือ การตรวจเช็คแพ็กเก็ตที่ละทิ้งโดยเราท์เตอร์ และทำการส่งข้อมูลใหม่อีกครั้ง นอกจากนี้หน้าที่ที่สำคัญอีกอย่างหนึ่งของชั้นนี้คือ การจัดเรียงแพ็กเก็ตข้อมูลที่อาจเดินทางถึงฝ่ายรับโดยไม่เป็นลำดับ การที่แพ็กเก็ตเดินทางถึงปลายทางไม่เป็นลำดับนี้เกิดขึ้นได้เนื่องจากแพ็กเก็ตแต่ละแพ็กเก็ตอาจจะเดินทางคนละเส้นทาง หรือแพ็กเก็ตบางแพ็กเก็ตอาจสูญหายระหว่างทางแล้วมีการส่งใหม่อีกครั้ง อย่างไรก็ตามชั้นเคลื่อนย้ายข้อมูลนี้สามารถตรวจเช็คลำดับและจัดเรียงแพ็กเก็ตเหล่านี้ให้เหมือนเดิมก่อนที่จะส่งต่อชุดข้อมูลไปให้ชั้นเซสชั่นต่อไป โดยโปรโตคอลในชั้นนี้จะแบ่งออกเป็น 2 ประเภท คือ คอนเน็กชั่นโอเรียนเต็ด (Connection-Oriented) และ คอนเน็กชั่นเลสส์ (Connectionless)

2.1.5 ชั้นเครือข่าย (Network Layer)

ชั้นเครือข่ายจะรับผิดชอบในการจัดเส้นทางให้กับข้อมูลระหว่างสถานีส่งและสถานีรับ ถ้ามีเส้นทางเดียว เช่น ถ้ามีคอมพิวเตอร์แค่สองเครื่องเชื่อมต่อกันโดยตรง การจัดเส้นทางคงไม่ยากเพราะมีแค่เส้นทางเดียว แต่ถ้าเป็นเครือข่ายที่ซับซ้อนการจัดเส้นทางก็ไม่ใช่เรื่องที่ง่ายนัก ชั้นนี้จะไม่มีกลไกใดๆ ในการตรวจสอบข้อผิดพลาดของข้อมูล ดังนั้นฟังก์ชันนี้จึงเป็นหน้าที่ของชั้นเชื่อมโยงข้อมูล

ชั้นเครือข่ายจะรับผิดชอบในการกำหนดเส้นทางข้อมูลระหว่างสถานีส่งและสถานีรับที่อยู่คนละเครือข่าย การที่จะทำเช่นนี้ได้ต้องมีระบบการจัดการที่อยู่ (Addressing) ที่ไม่ขึ้นอยู่กับที่อยู่ที่ใช้ในชั้นเชื่อมโยงข้อมูล

การให้บริการ (Service) ในเลเยอร์นี้จะแบ่งออกเป็น 2 ประเภทคือ

- Connectionless Network Service : การให้บริการแบบคอนเน็กชั่นเลสส์ หมายถึง การส่งข้อมูลแบบไม่มีการสร้างการเชื่อมต่อก่อน โดยหวังว่าแพ็กเก็ตจะส่งถึงปลายทางแน่นอน ดังนั้นจึงไม่สามารถรับรองได้ว่าข้อมูลจะส่งถึงปลายทางสำเร็จ ดังนั้นโปรโตคอลชั้นที่อยู่สูงกว่าจะต้องรับผิดชอบในการตรวจสอบข้อผิดพลาดเอง

- Connection-Oriented Network Service : เป็นการให้บริการเครือข่ายโดยมีการรับรองว่าข้อมูลจะส่งถึงปลายทางแน่นอน ซึ่งก่อนที่จะมีการส่งข้อมูลแต่ละครั้งนั้นจะมีการสร้างเส้นทางการเชื่อมต่อระหว่างสองสถานีก่อน และเมื่อรับส่งข้อมูลสำเร็จก็จะมีการยกเลิกเส้นทางการเชื่อมต่อดังกล่าว

2.1.6 ชั้นเชื่อมโยงข้อมูล (Data Link Layer)

เลเยอร์ที่สองของแบบอ้างอิง OSI มีชื่อว่าชั้นเชื่อมโยงข้อมูล ชั้นนี้ก็มีหน้าที่เหมือนกับชั้นอื่นๆ คือ รับและส่งข้อมูล ซึ่งชั้นนี้จะรับผิดชอบในการส่งข้อมูลและมีการตรวจสอบความถูกต้องของข้อมูลด้วย ทางด้านสถานีที่ส่งข้อมูลจะส่งข้อมูลให้เป็นเฟรม (Frame) ซึ่งในเฟรมจะมีข้อมูลที่ทำให้เฟรมสามารถส่งไปยังสถานีรับผ่านเครือข่ายท้องถิ่น (LAN) อย่างถูกต้องและสำเร็จ การส่งข้อมูลที่สำเร็จในที่นี้หมายถึงการที่เฟรมข้อมูลส่งถึงปลายทางที่สถานีส่งต้องการโดยที่เฟรมข้อมูลไม่มีข้อผิดพลาด ดังนั้นในเฟรมต้องมีข้อมูลที่ใช้ในการตรวจสอบข้อผิดพลาดของเฟรมข้อมูลนั้นๆด้วย การส่งข้อมูลสำเร็จนั้นเหตุการณ์เหล่านี้ต้องเกิดขึ้น

- สถานีเมื่อได้รับเฟรมแล้วต้องตรวจสอบข้อผิดพลาดของข้อมูล แล้วแจ้งให้สถานีส่งทราบ
- สถานีส่งต้องได้รับการตอบรับจากสถานีรับว่าได้รับเฟรมข้อมูลถูกต้องแล้ว

ในระหว่างการรับส่งข้อมูลอาจมีเหตุการณ์ต่างๆเกิดขึ้น เช่น ข้อมูลส่งไปไม่ถึงปลายทาง หรือบางส่วนของเฟรมเสียหายหรือเกิดข้อผิดพลาดขึ้น ชั้นเชื่อมโยงข้อมูลจะรับผิดชอบในการตรวจสอบและแก้ไขข้อผิดพลาดต่างๆเหล่านี้ ชั้นเชื่อมโยงข้อมูลนี้ยังรับผิดชอบในการจัดบิตต่อเนื่องที่ส่งผ่านมาจากชั้นกายภาพให้เป็นเฟรมเหมือนเดิม การจัดบิตต่อเนื่องให้กลับเป็นเฟรมเหมือนเดิมจะใช้บัฟเฟอร์ (Buffer) แล้วใส่ข้อมูลทีละบิตจนครบเฟรม ชั้นกายภาพและชั้นเชื่อมโยงข้อมูลนี้เป็นขั้นตอนที่มีในการสื่อสารข้อมูลทุกประเภท โดยไม่สนใจว่าจะเป็น LAN และ WAN โปรโตคอลมาตรฐานที่ทำงานในชั้นนี้ก็เหมือนกับชั้นกายภาพคือ มี IEEE 802.2 LLC , IEEE 802.3 (อีเทอร์เนต) , IEEE 802.5 (โทเคนริง), FDDI และ X.25

2.1.7 ชั้นกายภาพ (Physical Layer)

เลเยอร์ที่อยู่ล่างสุดคือ ชั้นกายภาพ (Physical Layer) เลเยอร์นี้จะรับผิดชอบเกี่ยวกับการส่งข้อมูลที่เป็นบิต หรือ "0" กับ "1" ในระบบเลขฐานสอง (Binary) ชั้นนี้จะรับข้อมูลจากเลเยอร์ที่ 2 หรือชั้นเชื่อมโยงข้อมูล (Data Link Layer) ซึ่งข้อมูลชุดหนึ่งจะเรียกว่า "เฟรม (Frame)" และทำการส่งเฟรมของข้อมูลนี้ทีละบิตแบบเรียงลำดับ เหตุการณ์นี้จะเกิดขึ้นทางฝั่งสถานีส่งข้อมูลส่วนทางฝั่งสถานีรับ

ข้อมูล ชั้นกายภาพต้องทำการรับข้อมูลที่ส่งมาทีละบิตและจัดส่งผ่านข้อมูลที่เป็นบิตนี้ต่อไปยังชั้นเชื่อมโยงข้อมูลเพื่อทำการโพรเซสส์ต่อไป

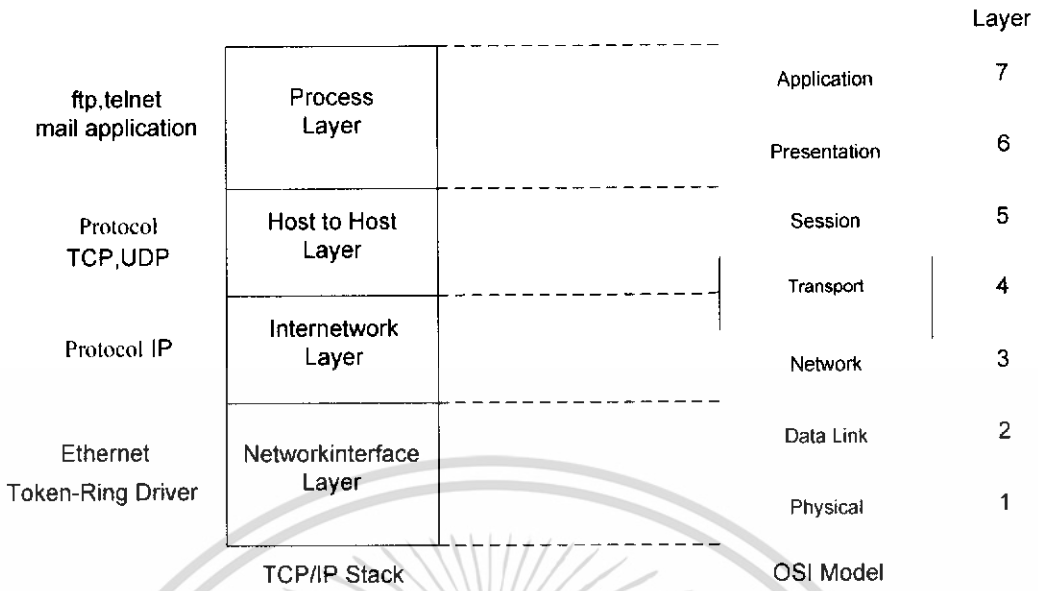
ชั้นนี้จะเห็นข้อมูลเป็นเลขฐานสอง หรือเลข 1 กับ 0 เท่านั้น ซึ่งจะไม่สนใจความหมายของข้อมูลเลย เลเยอร์นี้จะสนใจเฉพาะการที่จะแปลงข้อมูลให้เป็นสัญญาณไฟฟ้าหรือแสงขึ้นอยู่กับสื่อกลางที่ใช้ และสามารถส่งไปบนสื่อกลางหรือสายสัญญาณได้เท่านั้น ซึ่งมาตรฐานเกี่ยวกับความดันไฟฟ้า (Voltage) ที่เป็นตัวนำสัญญาณ ประเภทของสายสัญญาณและความต้านทานของสายสัญญาณ แม้กระทั่งลักษณะของหัวเชื่อมต่อสายสัญญาณด้วย เลเยอร์ที่หนึ่งนี้เป็นขั้นตอนหรือกลไกที่จำเป็นในการส่งสัญญาณข้อมูลไปบนสายสัญญาณและรับสัญญาณนั้นจากสายสัญญาณ

2.2 โปรโตคอล TCP/IP

โปรโตคอล TCP/IP (Transmission Control Protocol / Internet Protocol) มีการจัดกลไกการทำงานเป็นชั้นหรือ Layer เรียงต่อกัน โดยในแต่ละ Layer จะมีการทำงานเทียบได้กับ OSI model มาตรฐาน แต่บาง Layer ของโปรโตคอล TCP/IP จะทำงานเทียบกับ OSI หลาย Layer ปนกัน ซึ่งในแต่ละ Layer ของโปรโตคอล TCP/IP จะประกอบด้วย

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

โดยเมื่อเทียบกับมาตรฐาน OSI model ดังรูปที่ 2.2 ซึ่งเราจะเห็นว่าบางกลไกของโปรโตคอล TCP/IP เทียบได้กับมาตรฐาน OSI model สองชั้น หรือบางกลไกก็จะทำงานคาบเกี่ยวกันระหว่างบางชั้นของ OSI model ตัวอย่างเช่น กลไกการทำงานของโปรโตคอล TCP/IP ในส่วน Network Interface Layer เมื่อเทียบกับมาตรฐาน OSI model จะเทียบได้กับ ชั้นเชื่อมโยงข้อมูล (Data Link Layer) และชั้นกายภาพ (Physical Layer) 2 ชั้นรวมกัน เป็นต้น ในแต่ละกลไกของโปรโตคอล TCP/IP จะมีโปรโตคอลอื่นๆในชุดของ TCP/IP ร่วมทำงานอยู่ด้วย



รูปที่ 2.2 แสดงกลไกของโปรโตคอลมาตรฐาน OSI model

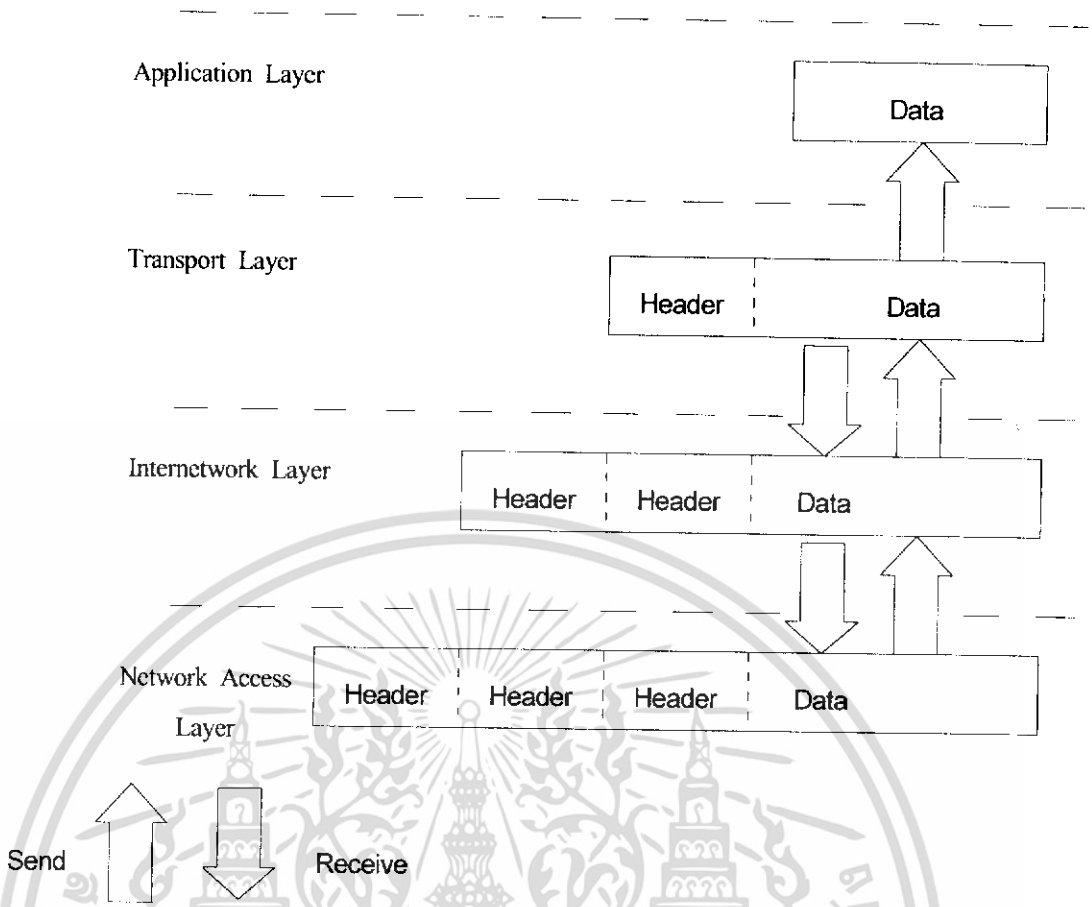
สำหรับการศึกษา โปรโตคอล TCP/IP นั้นเราจะไม่อ้างอิง OSI Reference Model นี้เพราะจะเข้าใจได้ยาก ดังนั้นเราจึงจะสร้าง Model ขึ้นมาใหม่โดยแบ่งออกเป็น 4 ชั้นดังนี้

ตารางที่ 2.2 แสดง Model ของโปรโตคอล TCP/IP

4	Application Layer
3	Host – to – Host Transport Layer
2	Internetwork Layer
1	Network Access Layer

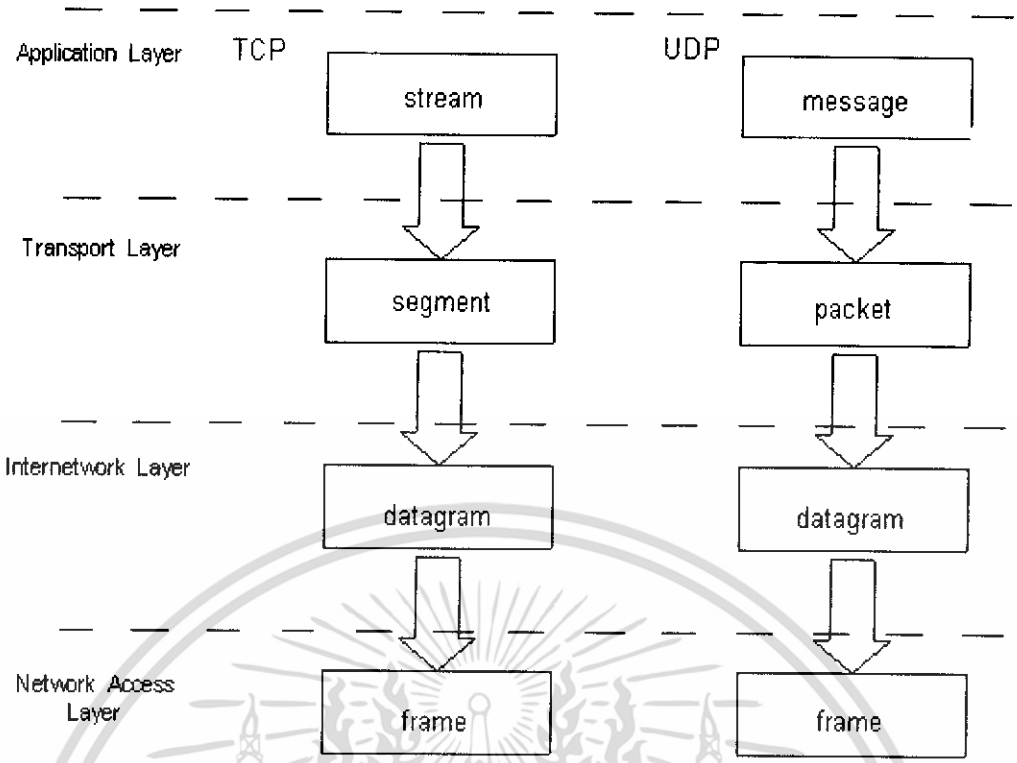
ลักษณะการทำงานของ Model ในลักษณะนี้คือ ข้อมูลจะถูกส่งลงมาจากชั้นข้างบนลงมายังชั้นข้างล่างสุดซึ่งมีหน้าที่จัดการเกี่ยวกับการส่งข้อมูลผ่านสายสัญญาณไปยังจุดหมายปลายทาง เมื่อข้อมูลไปถึงจุดหมายแล้วก็จะกลับย้อนจากชั้นล่างขึ้นไปชั้นบนสุด ซึ่งเป็นชั้นที่โปรแกรมใช้งานต่างๆทำงานอยู่

ขณะที่ข้อมูลถูกส่งผ่านจากชั้นบนลงมายังชั้นล่าง แต่ละชั้นจะทำการเพิ่มข้อมูลควบคุมเข้าไปเพื่อให้การส่งข้อมูลถูกต้อง และเป็นการส่งพารามิเตอร์ที่จำเป็นไปให้กับชั้นของมันในเครื่องปลายทาง ข้อมูลควบคุมเหล่านี้เราเรียกว่า "Header" แต่ละชั้นจะมี Header ที่มีรูปแบบเป็นของตัวเอง การเพิ่ม Header เข้าไปกับข้อมูลนั้นเราเรียกว่า การผนึกข้อมูล (Data Encapsulation) ซึ่งได้แสดงไว้ในรูปที่ 2.3



รูปที่ 2.3 Data Encapsulation

หน่วยของข้อมูลที่จะทำการส่งนั้นนั้นจะมีชื่อเรียกต่างกันเมื่อมันเดินทางผ่านแต่ละ Layer ดังแสดงไว้ในรูปที่ 2.4 ซึ่งจะเห็นว่าการส่งใน TCP/IP นั้นมีอยู่ 2 โปรโตคอลย่อยคือ TCP กับ UDP ชื่อของข้อมูลที่ผ่านโปรโตคอลทั้งสองนั้นแตกต่างกันในชั้นบนๆเท่านั้น ชื่อสำคัญๆที่เราจะต้องพบและใช้ต่อไปบ่อยๆ ได้แก่ datagram และ frame



รูปที่ 2.4 แสดงโครงสร้าง Data Structures

2.3 Network Interface Layer

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

2.3.1 อีเทอร์เนต

อีเทอร์เนตเป็นการใช้สาย coaxial – base bus เชื่อมต่อระบบเข้าด้วยกันเพื่อทำการส่งถ่ายข้อมูลในระบบดิจิตอลระหว่างคอมพิวเตอร์ โดยบริษัทอุปกรณ์ดิจิตอล บริษัท Intel และบริษัท Xerox ได้ใช้ระบบนี้อ้างอิงเรื่อยมา ซึ่งอีเทอร์เนตจะใช้เทคนิคการส่ง base – band ในการเข้าถึงข้อมูลและยังสามารถใช้บนสาย coaxial ที่มี 2 ขนาด คือ สายอีเทอร์เนตแบบหนา และสายอีเทอร์เนตแบบบาง โดยสายเคเบิลทั้งสองนี้เป็นที่ใช้กันอย่างกว้างขวางในปัจจุบัน ซึ่งระบบเครือข่ายอีเทอร์เนตมีลักษณะพิเศษดังนี้

- เป็นระบบเครือข่ายที่มีความเร็วในการส่งข้อมูลในรูปแบบดิจิตอลที่มีความเร็วตั้งแต่ 10 Mbps จนถึง 10 Gbps

- เป็นเครือข่ายที่มีขนาด Diameter ตั้งแต่ 205 เมตร จนถึง 4,000 เมตร

- ใช้โปรโตคอลการทำงานที่เรียกว่า CSMA/CD (Carrier Sense Multiple Access with Collision Detection) ซึ่งเป็นมาตรฐานของ IEEE 802.3 นอกจากนี้ก็ยังมีมาตรฐาน IEEE 802.3u สำหรับ 100 Mbps ที่เรียกว่า Fast Ethernet , IEEE 802.3z สำหรับ 1 Gbps ที่เรียกว่า Gigabit Ethernet รวมทั้ง IEEE 802.3ae สำหรับ 10 Gbps ที่เรียกว่า 10 Gigabit Ethernet

- หนึ่งเครือข่าย Ethernet สามารถมีอุปกรณ์เชื่อมต่อ เช่น คอมพิวเตอร์ลูกข่าย อุปกรณ์ Repeater เป็นต้น ได้มากมายถึง 1,024 รายการหรือเรียกว่า Node

- เป็นเครือข่ายที่สามารถใช้สายสัญญาณได้หลายแบบ เช่น สาย Coaxial ทั้งแบบหนาและแบบบาง สาย Twisted Pair ทั้งแบบ Shield และ Unshield รวมทั้งสาย Optical Fiber แบบและขนาดต่างๆ นอกจากนี้ยังสามารถใช้สื่อที่ใช้รับส่งข้อมูลแบบไร้สาย เช่น คลื่นวิทยุที่มีความถี่ Spread Spectrum รวมทั้งไมโครเวฟ (Microwave) ที่ใช้ความถี่ในช่วง 14 GHz และอินฟราเรด (infrared) เป็นต้น

- เป็นระบบเครือข่ายที่มีการเชื่อมต่อในรูปแบบ Bus และ Star Topology

- อุปกรณ์ที่ใช้มีราคาประหยัด

- มีความน่าเชื่อถือสูงโดยเฉพาะหากใช้สื่อที่เป็นสาย Optical Fiber

- มีเครื่องมือในรูปแบบของซอฟต์แวร์ที่ใช้บริหารจัดการเครือข่ายมาคมายที่ทำงานภายใต้ SNMP (Simple Network Management Protocol)

ส่วนประกอบหลักที่สำคัญของเครือข่ายอีเทอร์เนต

ระบบเครือข่ายอีเทอร์เนตมีส่วนประกอบหลักซึ่งเมื่อทำงานด้วยกันแล้วก็จะเป็นเครือข่ายที่มีประสิทธิภาพการทำงานสูงดังนี้

1. ตัวเฟรมเป็นชุดรูปแบบของบิตข้อมูลข่าวสารที่ใช้ส่งผ่านมาบนระบบ หากไม่มีเฟรมเราจะไม่สามารถสื่อสารข้อมูลบนเครือข่ายได้โดยเด็ดขาด การรับส่งข้อมูลข่าวสารบนเครือข่ายอีเทอร์เนตจะต้องเป็นไปในรูปแบบเฟรมมาตรฐาน 2 แบบ และเป็นแบบใดแบบหนึ่งเท่านั้น (การ์ด LAN เป็นผู้สร้างเฟรมนี้ขึ้นมา)

2. ชุดโปรโตคอลที่ใช้ในการควบคุมการแอกเซสเข้าไปที่เครือข่าย (Media Access Control Protocol) ซึ่งประกอบด้วยชุดของกฎกติกาที่อยู่ใน Ethernet Interface (เช่น การ์ด LAN เป็นต้น) ซึ่งเป็นกฎมาตรฐานที่จะยอมให้คอมพิวเตอร์ต่างๆสามารถเข้ามาที่เครือข่าย และแบ่งใช้ทรัพยากรต่างๆบนเครือข่ายได้อย่างมีประสิทธิภาพ

3. อุปกรณ์ที่ใช้รับส่งสัญญาณบนเครือข่าย (Signaling Components) ประกอบด้วยชุดของอุปกรณ์ที่ใช้เชื่อมต่อและส่งสัญญาณเพื่อการรับส่งข้อมูลภายในเครือข่าย

4. สื่อที่ใช้ในการรับส่งสัญญาณข้อมูลบนเครือข่าย (Physical Medium) ประกอบด้วยสายสัญญาณรวมทั้งอุปกรณ์ทางฮาร์ดแวร์อื่นๆที่จะช่วยในการนำพาข้อมูลข่าวสารต่างๆในรูปแบบดิจิตอลวิ่งไปมาบนเครือข่าย

2.3.2 เฟรมอีเทอร์เนต

อีเทอร์เนตได้ถูกระบุไว้อย่างแน่นอนไว้ในชั้นกายภาพ (Physical Layer) โดยมันจะแสดงรายละเอียดของรูปแบบเป็นแพ็กเก็ต (packet) ซึ่งโดยส่วนมากจะเรียกว่า "เฟรม (Frame)" ซึ่งเป็นหัวใจสำคัญของระบบอีเทอร์เนต จากรูปที่ 2.5 ได้แสดงรูปแบบของเฟรมอีเทอร์เนต โดยเฟรมนี้มันจะ encapsulates TCP/IP โปรโตคอล และสามารถทำการตอบสนองสำหรับส่งข้อมูลข้ามเข้าระบบที่เชื่อมต่อไปยังอีก 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 Redundancy Check (4 Byte)

รูปที่ 2.5 ลักษณะของเฟรมอีเทอร์เนต

2.3.3 MAC Address (Media Access Control Address)

เนื่องจากคอมพิวเตอร์แต่ละเครื่องสามารถแชร์ข้อมูลกันได้ในระบบเครือข่ายเดียวกัน ดังนั้นแต่ละเครื่องควรมีสิ่งที่ชี้ลักษณะเฉพาะตัวของมัน เช่น เราต้องมีบัตรประจำตัวประชาชน ซึ่งในทางคอมพิวเตอร์นี้เราจะใช้เลขฐาน 16 จำนวน 12 digits เป็นตัวบ่งชี้ลักษณะเฉพาะนั้นๆ ซึ่งเราเรียกว่า MAC Address เนื่องจาก MAC Address เป็นตัวบ่งชี้ลักษณะเฉพาะของแต่ละ machine ดังนั้นจึงต้องเป็นค่าที่ไม่ซ้ำกัน (unique) MAC Address เป็นเลข 48 bits โดยแบ่งออกเป็น 2 ส่วนโดย 24 bits แรกเป็นค่าที่แสดงถึงบริษัทที่ผลิตการ์ดนั้นๆ ส่วน 24 bits หลังเป็น serial number ที่ทางบริษัทกำหนดให้ซึ่งแต่ละตัวต้องไม่ซ้ำกัน เราเรียกเลข 24 bit นี้ว่า OUI (Organizationally Unique Identifier) ซึ่ง OUI จะใช้เพียง 22 bits เท่านั้น ส่วนอีก 2 bits ที่เหลือจะถูกใช้เพื่อวัตถุประสงค์อื่น โดย bit หนึ่งจะใช้เพื่อแสดงว่า address นั้นเป็น broadcast/multicast address ส่วนอีก bit หนึ่งนั้นไว้แสดงว่า adapter นั้นถูกกำหนด locally administered address ซึ่ง admin ของระบบจะทำการกำหนด MAC Address เพื่อความเหมาะสมของนโยบายระบบ เช่น MAC Address = 03 00 00 00 00 00 01 ซึ่งจะเห็นว่า byte แรก = 03 = 00000011 นั่นคือ ทั้ง 2 bits ถูก set (reset = 0) ซึ่งเอาไว้กรณี multicast ให้ทุกเครื่องที่ run บนโปรโตคอล NetBEUI

2.3.4 Type field

ประเภทของ field จะใช้ระบุความแตกต่างของโปรโตคอล คอมพิวเตอร์จะดำเนินการให้โปรโตคอลหลายๆตัวสามารถเห็นความแตกต่างของพวกมันได้ง่ายและผ่านการยินยอมของเฟรมที่เกี่ยวเนื่องกับเครือข่าย

ระบบ TCP/IP โดยทั่วไปจะใช้อีเทอร์เนต 3 field ซึ่งมีค่าดังตารางที่ 2.3 ประเภทของหมายเลขอีเทอร์เนตเป็นรีจีสเตอร์กับ IEEE โดยจะใช้หมายเลขที่เฉพาะเจาะจงไม่เหมือนใคร

ตารางที่ 2.3 Ethernet type field

Type	Protocol
0x0800	IP
0x0806	ARP
0x0835	RARP

2.3.5 Cyclic Redundancy Check (CRC)

ที่ปลายสุดของเฟรมคือ Cyclic Redundancy Check (CRC) มีขนาด 32 บิตที่คำนวณได้จากบิตทั้งหมดของอีเทอร์เนตเฟรมแต่จะไม่สนใจ Preamble ถ้าในเฟรมมีความผิดพลาดเกิดขึ้นค่าที่คำนวณได้จะแตกต่างไปจากเฟรมเดิม ทำให้ข้อมูลไม่สามารถที่จะผ่านเฟรมไปยังชั้น Network Layer ได้

2.3.6 IEEE 802.3 เฟรม

IEEE 802.3 หรือ อีเทอร์เนต (Ethernet) เป็นเครือข่ายที่มีความเร็วสูงการส่งข้อมูล 10 เมกกะบิตต่อวินาที สถานีในเครือข่ายอาจมีโทโปโลยีแบบบัส (Bus) หรือแบบดาว (Star) IEEE ได้กำหนดมาตรฐานอีเทอร์เนตซึ่งทำงานที่ความเร็ว 10 เมกกะบิตต่อวินาทีไว้หลายประเภทตามชนิดสายสัญญาณ เช่น

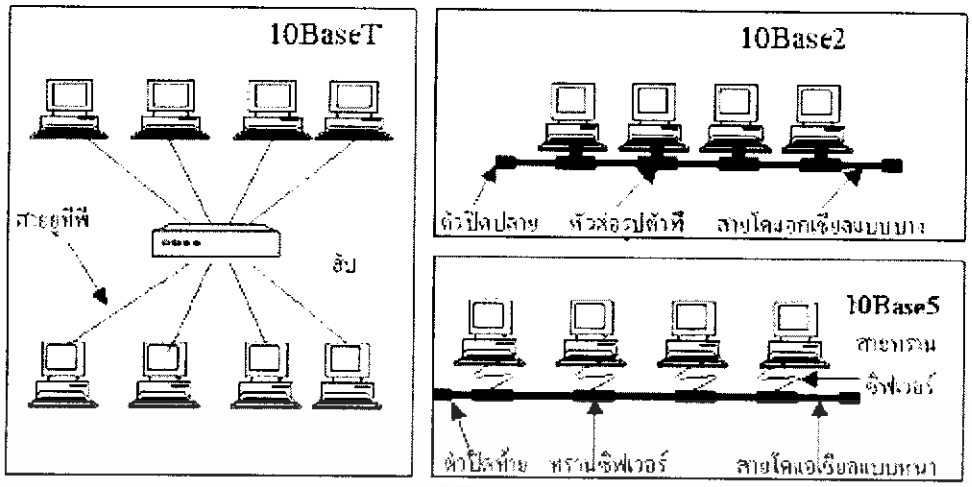
- 10 Base 5 อีเทอร์เนตโทโปโลยีแบบบัสซึ่งใช้สายโคแอ็กซ์เชียลแบบหนา (Thick Ethernet) ความยาวของสายในเซกเมนต์หนึ่งๆไม่เกิน 500 เมตร

- 10 Base 2 อีเทอร์เนตโทโปโลยีแบบบัสซึ่งใช้สายโคแอ็กซ์เชียลแบบบาง (Thin Ethernet) ความยาวของสายในเซกเมนต์หนึ่งๆไม่เกิน 185 เมตร

- 10 Base-T อีเทอร์เนตโทโปโลยีแบบดาวซึ่งใช้ฮับเป็นศูนย์กลาง สถานีและฮับเชื่อมด้วยสาย UTP (Unshield Twisted Pair) ด้วยความยาวไม่เกิน 100 เมตร

รูปที่ 2.6 แสดงถึงลักษณะเครือข่ายอีเทอร์เนตแยกตามประเภทของสายสัญญาณ รหัสขึ้นต้นด้วย "10" หมายถึงความเร็วสายสัญญาณ 10 เมกกะบิตต่อวินาที คำว่า "Base" หมายถึงสัญญาณชนิด "Base" รหัสถัดมาหากเป็นตัวเลขหมายถึงความยาวสายต่อเซกเมนต์ในหน่วยหนึ่งร้อยเมตร (5=500 , 2 แทนค่า 185) หากเป็นอักษรจะหมายถึงชนิดของสาย เช่น T คือ Twisted pair หรือ F คือ Fiber optics

ส่วนมาตรฐานอีเทอร์เนตความเร็ว 100 เมกกะบิตต่อวินาทีที่นิยมใช้ในปัจจุบันได้แก่ 100 Base-TX และ 100 Base-FX สำหรับอีเทอร์เนตความเร็วสูงแบบกิกะบิตอีเทอร์เนตเริ่มแพร่หลายมากขึ้น ตัวอย่างของมาตรฐานกิกะบิตอีเทอร์เนตในปัจจุบันได้แก่ 1000 Base-T , 1000 Base-LX และ 1000 Base-SX เป็นต้น



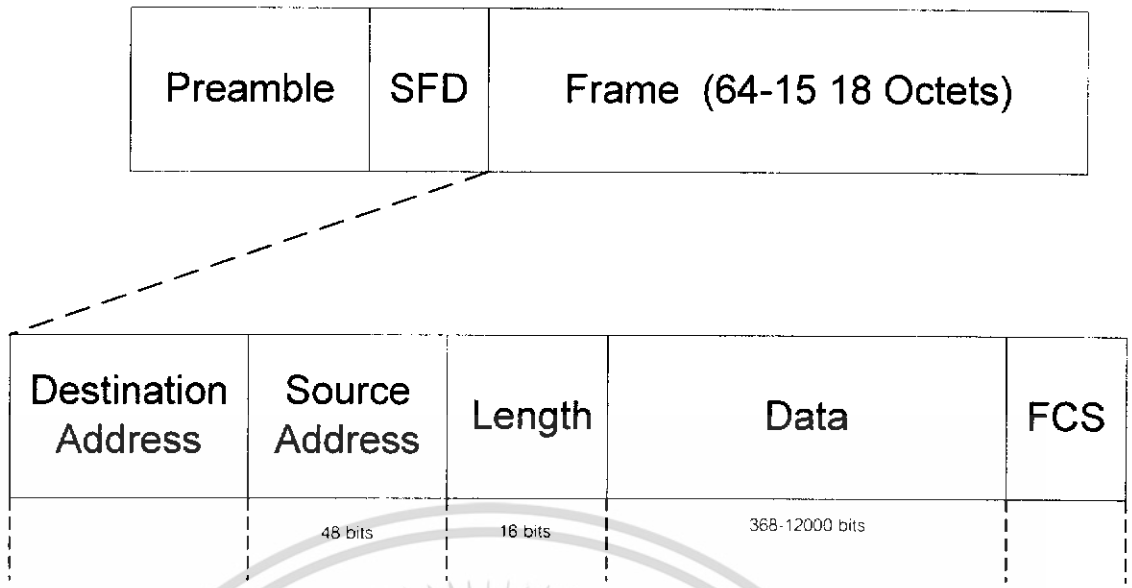
รูปที่ 2.6 อีเทอร์เนตประเภทต่าง ๆ

อีเทอร์เนตใช้โปรโตคอล ซีเอสเอ็มเอ/ซีดี (CSMA/CD : Carrier Sense Multiple Access with Collision Detection) เป็นตัวกำหนดขั้นตอนให้สถานีเข้าครอบครองสายสัญญาณ ในขณะเวลาหนึ่งจะมีเพียงสถานีเดียวที่เข้าครองสายสัญญาณเพื่อส่งข้อมูล

สถานีที่ต้องการส่งข้อมูลต้องการตรวจสอบสายสัญญาณว่ามีสถานีอื่นใช้สายอยู่หรือไม่ ถ้าสายสัญญาณว่างก็ส่งข้อมูลได้ทันที หากไม่ว่างก็ต้องคอยจนกว่าสายสัญญาณว่างจึงจะส่งข้อมูลได้ ขณะที่สถานีหนึ่งๆกำลังส่งข้อมูลก็ต้องตรวจสอบสายสัญญาณไปพร้อมกันด้วยเพื่อตรวจว่าในจังหวะเวลาที่ใกล้เคียงกันนั้นมีสถานีอื่นซึ่งพบสายสัญญาณว่างและส่งข้อมูลมาหรือไม่ หากเกิดกรณีเช่นนี้ขึ้นแล้ว ข้อมูลจากทั้งสองสถานีจะผสมกันหรือเรียกว่า การชนกัน (Collision) และนำไปใช้ไม่ได้ สถานีจะต้องหยุดส่งและสุ่มหาเวลาเพื่อเข้าใช้สายสัญญาณใหม่

ในเครือข่ายอีเทอร์เนตที่มีสถานีจำนวนมากมักพบว่าการส่งข้อมูลจะล่าช้าเพราะแต่ละสถานีพยายามยึดช่องสัญญาณเพื่อส่งข้อมูลและเกิดการชนกันเกือบตลอดเวลา โดยไม่สามารถกำหนดว่าสถานีใดจะได้ใช้สายสัญญาณเมื่อเวลาใด อีเทอร์เนตจึงไม่เหมาะกับการใช้งานในระบบเครือข่ายขนาดใหญ่

เฟรมข้อมูลสำหรับระบบ Ethernet ประกอบขึ้นด้วยกลุ่มของบิตที่เป็นข้อมูลและข่าวสารสำคัญแบ่งออกเป็นขนาดสัดส่วนที่แน่นอนที่เรียกว่าช่อง Field ดังรูปที่ 2.7



รูปที่ 2.7 แสดงลักษณะโครงสร้างของเฟรมข้อมูลตามมาตรฐาน IEEE 802.3

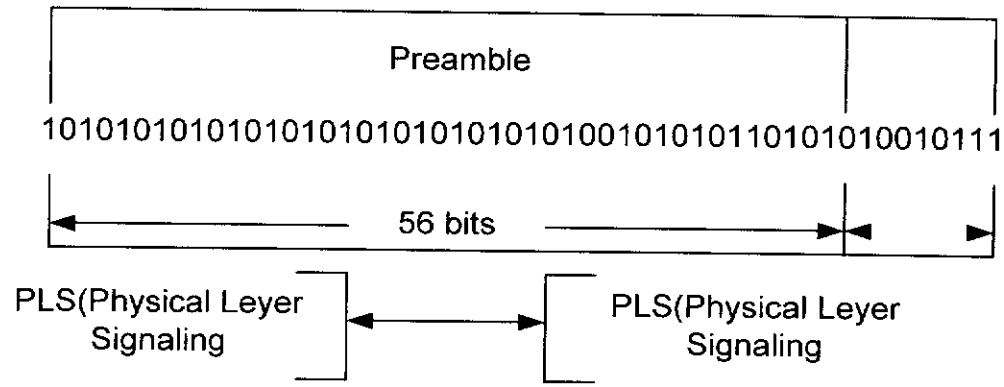
รูปที่ 2.7 แสดงให้เห็นรูปแบบของเฟรมข้อมูลที่ใช้บน Ethernet ตามมาตรฐาน IEEE 802.3 ต่อไปเราจะมาทำความเข้าใจเฟรมมาตรฐานของ IEEE802.3

ช่อง Preamble

ช่อง Preamble ประกอบด้วยบิตข่าวสารที่เป็นเลข 1 และ 0 สลับกันและสิ้นสุดที่ 11 ซึ่งเป็นบิตที่ 63 และ 64 เป็นบิตข่าวสารที่ยังไม่ใช่ข้อมูลจริงของผู้ส่ง Preamble ประกอบด้วยข่าวสารที่มีขนาด 7 หรือ 8 ไบต์ จุดประสงค์ของข่าวสารนี้ก็เพื่อใช้สร้างจังหวะการรับข้อมูลให้แก่ผู้รับโดยที่ส่วนนี้จะไปถึงตัวผู้รับก่อน ทำให้เครื่องคอมพิวเตอร์ของผู้รับสามารถปรับจังหวะความเร็วให้เข้ากับผู้ส่งได้ (Synchronize) สำหรับเฟรมแบบ Ethernet II จะมีขนาด 8 ไบต์ และถ้าเป็นมาตรฐาน IEEE802.3 แล้วช่องนี้จะถูกแบ่งออกเป็น 2 ส่วน (ดังรูปที่ 2.8) ได้แก่

1. Preamble
2. Start of Frame Delimeter

จุดประสงค์อีกประการหนึ่งของ Preamble ได้แก่การทำให้แน่ใจว่าระยะเวลาความห่างระหว่างเฟรมที่ 1 กับเฟรมถัดมาจะมีความห่างอยู่ที่ 9.6 ไมโครวินาทีโดยมาตรฐาน ทั้งนี้ก็เพื่อการปฏิบัติการตรวจสอบความผิดพลาดและกู้สถานการณ์เมื่อเกิดปัญหา



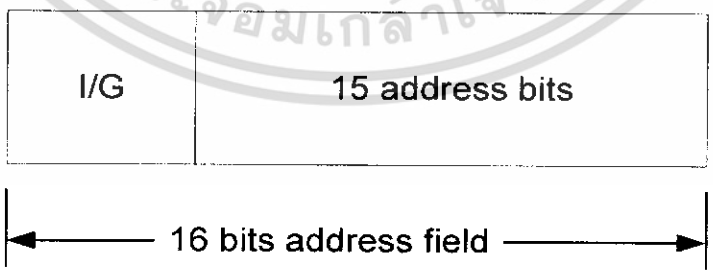
รูปที่ 2.8 แสดงลักษณะส่วนการทำงานภายในของ Preamble

ช่อง Preamble รวมทั้ง Start of Frame Delimeter นี้จะถูกถอดออกไปเป็นอันดับแรกหลังจากคอมพิวเตอร์ของผู้รับได้เฟรมข้อมูลนี้เป็นที่เรียบร้อยแล้ว โดยการ์ด LAN ของผู้รับจะทำการตรวจสอบว่าจำนวนบิตในช่อง Preamble มีครบ 64 บิตหรือไม่ หากไม่ครบก็แสดงว่าเป็นเฟรมที่ไม่สมบูรณ์โดยอาจเป็นเฟรมที่หลงเหลือมาจากการชนกับเครือข่ายก็ได้ ลักษณะเช่นนี้การ์ด LAN ของผู้รับจะปฏิเสธที่จะรับและโยนทิ้ง (Frame Drop) ในทันที โดยสามารถเปรียบเทียบ Preamble ไม่สมบูรณ์ได้กับรถยนต์หรูๆที่มีกันชนหน้าบุบซึ่งเจ้าของที่เป็นผู้รับทำใจไม่ได้เนื่องจากเป็นรถใหม่

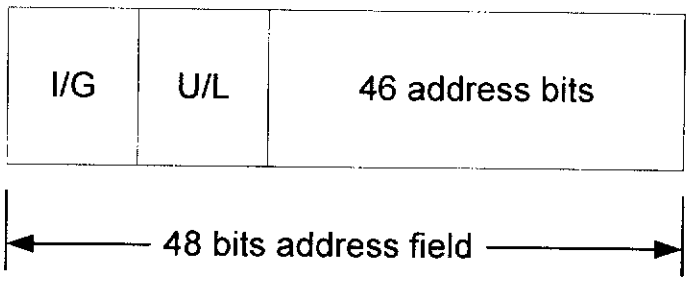
ช่อง Destination Address

ในช่อง Destination Address ประกอบด้วยข้อมูลข่าวสารเกี่ยวกับแอดเดรสหรือที่อยู่ของผู้รับปลายทาง ดูเผินๆแล้วเป็นช่องที่เรียบง่ายไม่มีอะไรมากแต่โดยความจริงแล้วช่องนี้ถูกแบ่งออกเป็นช่องย่อยๆที่เรียกว่า Sub-fields ซึ่งเก็บข้อมูลข่าวสารที่ใช้ดูแลการทำงานของเครือข่ายทั้งนี้ขึ้นอยู่กับแอดเดรสที่ปรากฏอยู่ในช่องนี้ไม่ว่าช่องแอดเดรสนี้จะมีแอดเดรสที่ถูกเจาะจงเป็นรายบุคคลหรือเป็นกลุ่มของผู้รับก็ตาม

ก. ช่องข่าวสารขนาด 2 ไบต์



ข. ช่องข่าวสารขนาด 6 ไบต์



หมายเหตุ

หากค่าบิตในช่องย่อย I/G มีค่าเป็น "0" ก็หมายถึงแอดเดรสที่ระบุตัวคอมพิวเตอร์ผู้รับอย่างเฉพาะเจาะจง แต่ถ้ามีค่าเป็น "1" ก็หมายถึงคอมพิวเตอร์ที่ระบุแอดเดรสเป็นกลุ่มไม่เจาะจงเครื่องใดเครื่องหนึ่ง

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

จากกรอบแผนผังในข้อ ก. และข้อ ข. เป็นการแสดงรูปภาพในช่องย่อยๆของ Destination Address ซึ่งมีทั้งแบบขนาด 2 ไบต์สำหรับเฟรมแบบ IEEE 802.3 เท่านั้น ส่วนแบบ 6 ไบต์สำหรับเฟรมแบบ Ethernet และ IEEE 802.3 ผู้ใช้งานสามารถเลือกใช้ช่องย่อยใน Destination Address ทั้งแบบ2 ไบต์ และ 6 ไบต์อย่างใดอย่างหนึ่งและเนื่องจากอุปกรณ์ที่ใช้เป็นแบบ IEEE 802.3 ดังนั้นทุกสถานีเครือข่ายทุกเครื่องจะต้องใช้โครงสร้างการอ้างอิงแอดเดรสแบบเดียวกันเสมอ เช่น หากเลือกใช้แบบ 2 ไบต์ก็หมายความว่าคอมพิวเตอร์ทุกเครื่องที่เชื่อมต่อเข้ากับเครือข่ายจะต้องใช้แบบ 2 ไบต์ทั้งหมด

ทุกวันนี้เครือข่ายเกือบทั้งหมดใช้มาตรฐานแบบ 6 ไบต์เพื่อการอ้างอิงแอดเดรสถึงกัน แต่ก็ยังร่วมช่องขนาด 2 ไบต์นี้เข้าไปเป็นทางเลือกอีกด้วย

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

ช่องย่อย Source Address แบบ 2 ไบต์ใช้กับมาตรฐาน IEEE 802.3 และต้องใช้แอดเดรสปลายทางขนาด 2 ไบต์เท่านั้น รวมทั้งทุกสถานีลูกข่ายจะต้องใช้ช่องแอดเดรสขนาด 2 ไบต์ส่วนช่องย่อยขนาด 6 ไบต์สามารถใช้ได้ทั้งมาตรฐาน Ethernet ทั่วไปและ IEEE 802.3 และเมื่อมีการเลือกใช้ช่องย่อย 6 ไบต์ก็จะมีการกำหนดให้ 3 ไบต์แรกเป็นแอดเดรสที่ IEEE กำหนดให้ผู้ผลิตต่างๆ ซึ่งแอดเดรสนี้จะถูกฝังตัวอยู่ในไมโครชิปบนการ์ด LAN ส่วนที่เหลืออีก 3 ไบต์ก็จะเป็นแอดเดรสที่ผู้ผลิตการ์ด LAN แต่ละแห่งนำไปกำหนดกันเองต่อไป

ตัวอย่างของ Source Address ที่แสดงรหัสแอดเดรสของผู้ผลิตดังนี้คือ

ตารางที่ 2.4 แสดงตัวอย่างของ Source Address ที่แสดงรหัสแอดเดรสของผู้ผลิต

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 ไบต์ใช้กับ Ethernet Frame เท่านั้น โดยช่องนี้ใช้เพื่อแสดงว่าโปรโตคอลการทำงานของเฟรมนี้เป็นแบบใด จุดประสงค์คือเพื่อต้องการให้ทราบว่าข้อมูลที่อยู่ในเฟรมนี้จะทำงานภายใต้โปรโตคอลใด ซึ่งผู้รับจะได้เตรียมการแปลความหมายที่อยู่ในช่องข้อมูล (Data Field) ได้ถูกต้อง

ภายใต้ระบบเครือข่าย Ethernet เราสามารถใช้โปรโตคอลได้หลายตัวพร้อมกันบนเครือข่าย LAN และบริษัท XEROX ทำหน้าที่เป็นผู้ให้บริการกำหนดพิกัดระยะของแอดเดรสที่เป็นลิขสิทธิ์ให้แก่ผู้ผลิตการ์ด LAN ต่างๆ รวมทั้งการกำหนดค่าที่ใช้แสดงแทนโปรโตคอลที่ใช้ในช่อง Type แห่งนี้

ตารางที่ 2.5 แสดงตัวอย่างของรหัสที่ใช้แสดงแทนโปรโตคอลที่ใช้ในช่อง Type 18 รายการ

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 LAN Bridge	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 IEEE 802.3 เท่านั้นเป็นช่องที่ใช้แสดงขนาดจำนวนของไบต์ที่มีปรากฏอยู่ในช่อง Data

ภายใต้มาตรฐาน Ethernet และ IEEE 802.3 ขนาดของเฟรมจะมีขนาดเล็กที่สุดไม่ต่ำกว่า 64 ไบต์นับตั้งแต่ชุดแรกสุดคือ Preamble จนถึงช่องสุดท้าย ได้แก่ FCS และการกำหนดให้มีขนาดเล็กที่สุดไม่น้อยกว่า 64 ไบต์นี้จุดประสงค์ก็เพื่อให้แน่ใจว่าช่วงระยะการส่งข้อมูลมีมากพอที่จะทำให้การ์ด LAN สามารถตรวจพบการเกิดการชนกัน (Collision) บนสายสัญญาณที่มีขนาดความยาวที่สุดของเครือข่าย หากขนาดเฟรมเล็กกว่า 64 ไบต์ก็อาจเกิดปัญหา Collision ซึ่งจะนำไปสู่ปัญหาบนเครือข่ายได้

บนพื้นฐานของเฟรมขนาดเล็กที่สุดคือ 64 ไบต์ และมีการใช้ช่องแอดเดรสขนาด 2 ไบต์ หมายความว่าช่องสำหรับ Data จะต้องมีขนาดเล็กสุดไม่ต่ำกว่า 46 ไบต์ (เมื่อหักขนาดและจำนวนช่องต่างๆออกไปหมดแล้ว)

เมื่อใดที่ข้อมูลมีขนาดเล็กกว่า 46 ไบต์ช่องของ Data ที่อยู่ในเฟรมจะถูกใส่ค่าเพิ่มให้ได้อย่างน้อยเป็น 46 ไบต์

ช่อง Data (Data Field)

ดังที่ได้กล่าวมาแล้วว่าช่องของ Data อย่างน้อยต้องมีขนาดไม่เล็กกว่า 46 ไบต์ เพื่อให้แน่ใจว่าเฟรมมีขนาดไม่ต่ำกว่า 64 ไบต์ซึ่งหมายความว่าการแพร่ข้อมูลขนาดหนึ่งไม่ว่า 1 หรือ 10 ไบต์ก็ตามต้องมาจาก 46 ไบต์นี้แต่ถ้าข้อมูลในช่องนี้เล็กกว่า 46 ไบต์ แน่นอนว่าต้องมีการเพิ่มไบต์ลงไปอีกเพื่อให้ได้ขนาด 46 ไบต์อยู่ดี

ขนาดของข้อมูลที่อยู่ใน Data จะต้องมีขนาดสูงสุดไม่เกิน 1,500 ไบต์

ช่องตรวจสอบความผิดพลาดของข้อมูลในเฟรม (Frame Check Sequence)

ช่อง Frame Check Sequence นี้ใช้ได้ในเฟรมมาตรฐานทั้งเฟรมมาตรฐาน Ethernet และ IEEE 802.3 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

เมื่อเฟรมถูกส่งมาถึงผู้รับแล้ว ตัวการ์ด LAN ของผู้รับจะทำการตรวจสอบค่าที่อยู่ในช่อง Preamble เพื่อดูว่ามีความถูกต้องหรือไม่ เพื่อให้แน่ใจว่าเฟรมที่ทำการตรวจสอบอยู่นี้ไม่ได้เป็นเฟรมที่เหลือรอดจาก Collision หรือการชนกันของสัญญาณในเครือข่าย และหาก Preamble ไม่มีปัญหาเรื่องความถูกต้องก็จะมีการคำนวณค่าที่อยู่ในช่อง Frame Check Sequence ต่อไป หากมีความผิดพลาดกล่าวคือ ค่าที่ได้ไม่ตรงกับค่าที่ได้จากการที่คำนวณได้จากต้นทางแสดงว่าเป็นเฟรมที่มีปัญหา ซึ่งก็มักเป็นปัญหาจากสายสัญญาณรวมทั้งสัญญาณรบกวน ซึ่งในที่สุดการ์ด LAN ก็จะปฏิเสธที่จะรับเฟรมที่เข้ามานี้ในที่สุด

ข้อกำหนดเกี่ยวกับขนาดของ Data Frame

ขนาดของ Data Frame มีมาตรฐานดังต่อไปนี้

- ขนาดเล็กที่สุดต้องไม่น้อยกว่า 64 ไบต์ โดยมี 12 ไบต์สำหรับแอดเดรส 2 ไบต์สำหรับช่อง Length 46 ไบต์สำหรับเก็บข้อมูล และ 4 ไบต์สำหรับตรวจสอบความผิดพลาดข้อมูลหรือ Frame Check Sequence

- ขนาดใหญ่ที่สุดต้องไม่เกิน 1,518 ไบต์ โดยแบ่งออกเป็น 12 ไบต์สำหรับแอดเดรส 2 ไบต์สำหรับ Length 1,500 ไบต์สำหรับข้อมูล และ 4 ไบต์สำหรับช่องตรวจสอบความผิดพลาดข้อมูล

- เฟรมที่มีขนาดเล็กที่สุด 64 ไบต์นี้เทียบกับ 512 บิต (Bit/Byte) โดยที่ระบบ Ethernet มีค่า Bit Time อยู่ที่ 0.1 ไมโครวินาที (Bit Time เป็นช่วงเวลาที่ใช้ในการส่งข้อมูลขนาด 1 บิต) ดังนั้น การส่งข้อมูลขนาดเล็กที่สุดคือ 64 ไบต์จะต้องใช้เวลาอยู่ที่ 51.2 ไมโครวินาที

2.4 Internetwork Layer

ในระดับล่างต่อมาในชั้น Internetwork Layer มีหน้าที่ส่งผ่านข้อมูลในระหว่างเครือข่าย โดยมีโปรโตคอลที่ทำงานเป็นกลไกสำคัญในการส่งผ่านข้อมูลไปยังเครือข่ายใดๆบนอินเตอร์เน็ต คือ โปรโตคอล IP (Internet Protocol) นอกจากนี้ในชั้น Internetwork Layer ยังมีโปรโตคอลที่ทำงานอยู่ด้วยอีก 2 ชนิด คือ โปรโตคอล Internet Control Message Protocol (ICMP) และโปรโตคอล Address Resolution Protocol (ARP) ซึ่งอยู่ภายในโปรโตคอล IP

2.4.1 โปรโตคอล IP (Internet Protocol)

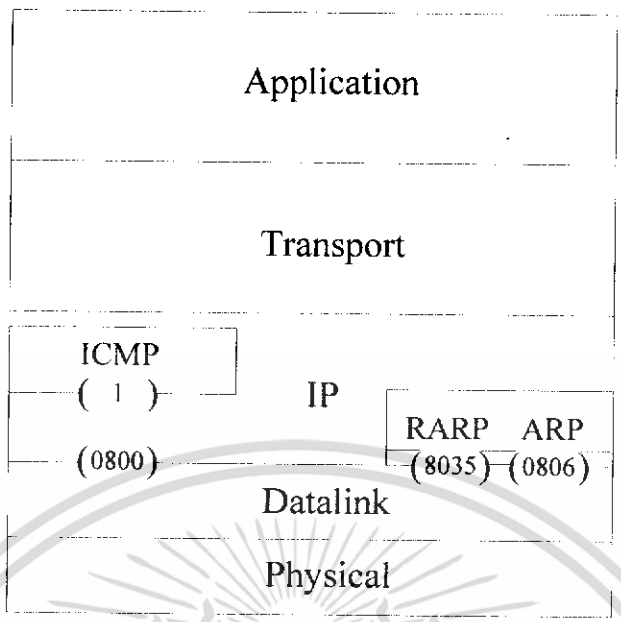
โปรโตคอล IP ทำหน้าที่ให้บริการส่งผ่านข้อมูลที่มาจาก Host – to – Host Layer เพื่อส่งข้ามไปยังเครือข่ายใดๆได้อย่างถูกต้อง แม้ว่าจะมีเครือข่ายเชื่อมต่อกันอยู่ในอินเตอร์เน็ตเป็นล้านๆเครือข่ายก็ตาม เนื่องจากโปรโตคอล IP มีข้อมูลตำแหน่ง IP ปลายทางที่จะส่งข้อมูลไปให้โดยทำงานร่วมกับอุปกรณ์ router เพื่อส่งข้อมูลข้ามเครือข่ายออกไปได้ ตัวโปรโตคอล IP จะทำงานแบบ packet switching คือมีการส่งข้อมูลผ่านสวิทช์ (switch) ไปยังปลายทางโดยข้อมูลจะเดินทางไปยังเครือข่ายต่างๆผ่านสวิทช์นี้ไปเรื่อยๆจนกว่าจะถึงปลายทาง ตัววงจรผ่านหรือ switch นี้อาจเป็น gateway หรือ router ในระบบเครือข่ายก็ได้ ซึ่งในข้อมูลของโปรโตคอล IP จะมีข้อมูลของหมายเลข IP ที่จะส่งข้อมูลไปและเมื่อถึงเครือข่ายปลายทางแล้วจะมีกลไกแปลงหมายเลข IP ให้เป็นหมายเลขฮาร์ดแวร์ประจำเครื่องที่ถูกต้องอีกทีหนึ่งด้วยโปรโตคอล ARP

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

2.4.2 ส่วนประกอบของ IP

IP Address จะต้องมีค่าเฉพาะเจาะจงไม่เหมือนใคร เพื่อที่จะใช้เชื่อมต่อเข้ากับเครือข่ายที่หมายเลขของเครือข่ายที่เฉพาะเจาะจงเช่นกัน โดยในรูปที่ 2.9 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 2.9 แสดงส่วนประกอบของ IP

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

2.4.3 IP Datagram

Datagram เป็น basic unit ของข้อมูลที่ IP จะทำการส่ง ซึ่งถูกออกแบบมาให้ทำงานกับเครือข่ายแบบ Packet Switch ซึ่งข้อมูลของผู้ใช้มักถูกแบ่งออกเป็นหลาย Datagram โดยแต่ละตัวจะมี Header ที่เก็บรายละเอียดเกี่ยวกับตัวมันเองและปลายทางที่มันจะไป Datagram แต่ละตัวจะเดินทางโดยไม่เกี่ยวข้องกัน นั่นคือ Datagram แต่ละตัวอาจจะเดินทางไปยังปลายทางโดยใช้เส้นทางคนละเส้นและลำดับที่ของการไปถึงจุดหมายปลายทางก็ไม่แน่นอน เป็นหน้าที่ของ Internet Protocol ในเครื่องปลายทางที่จะต้องประกอบ Datagram เหล่านี้ให้กลายเป็นข้อมูลของผู้ใช้ที่สมบูรณ์อีกครั้ง

รูปที่ 2.10 แสดง IP Datagram ซึ่งประกอบด้วย IP Header และ IP Payload

2.4.3a IP Header เป็นขนาดที่เปลี่ยนแปลงได้ระหว่าง 20 และ 60 ไบต์ ในการเพิ่มขึ้น 4 ไบต์ มันจะจัดเตรียมการสนับสนุน routing , การแสดงตัว payload , การชี้ให้เห็นถึงขนาด IP Header และ Datagram , การสนับสนุน fragmentation โดยมีโครงสร้างดังรูปที่ 2.11

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 2.11 แสดงโครงสร้าง IP Header

- *Version*

ใช้แสดง Version ของ IP ที่สร้าง Header

- *Header Length*

Header Length ใช้แสดงความยาว Header ในหน่วยของ 32 บิต ให้สังเกตว่า Header จะมีความยาวอย่างน้อย 5 words

- *Type of Service*

Type of Service (TOS) มีขนาด 8 บิต ใช้ในการแสดงคุณสมบัติของ Service ที่ทำการส่ง Datagram โดยใช้ Internetwork Routers โดย TOS ประกอบด้วย Sub – Field และ Flags ที่ต้องการแสดงถึง precedence , delay , throughput , reliability และ cost characteristics TOS จะทำการตั้งค่าโดย Sending Host และไม่สามารถแก้ไขได้โดย Routers

- *Total Length*

Total Length มีขนาด 2 ไบต์ ใช้ในการแสดงขนาดของ IP Datagram (IP Header และ IP Payload) สำหรับขนาดสูงสุดของ Total Length จะเหมือนกับ IP MTU สำหรับ Network Interface Layer ระหว่าง Header Length กับ Total Length สามารถคำนวณหาค่า IP Payload ได้จาก

IP Payload Length (ไบต์) = Total Length (ไบต์) - 4* Header Length (32 bit words)

- *Identification*

Identification มีขนาด 2 ไบต์ ใช้ในการแสดงลักษณะเฉพาะในการส่ง IP Packet ระหว่าง Source กับ Destination Node โดย Sending Host จะเป็นตัวตั้งค่า Identification และจะทำการต่อเข้ากับ IP Datagram

- *Flags*

Flags มีขนาด 3 บิต ประกอบด้วย 2 Flags สำหรับ Fragmentation โดย Flags ตัวแรกใช้แสดง IP Datagram ว่ามีลักษณะที่เหมาะสมสำหรับ Fragmentation หรือไม่ และ Flags อีกตัวหนึ่งใช้แสดงว่ามี Fragment ไปตาม Fragment IP Datagram หรือไม่

- *Fragment Offset*

Fragment Offset มีขนาด 13 บิต ใช้ในการแสดง Offset ที่เป็น Fragment เริ่มต้นที่เกี่ยวข้องกับ IP Payload เดิม

- *Time To Live*

Time To Live มีขนาด 1 ไบต์ ใช้ในการแสดงการเชื่อมต่อของ IP Datagram โดยสามารถที่จะเคลื่อนย้ายก่อนที่ IP Router จะทิ้งมันไป

- *Protocol*

Protocol มีขนาด 1 ไบต์ ใช้ในการแสดง Protocol ใน Layer ที่สูงกว่าซึ่งอยู่ใน IP Payload ค่าที่ใช้โดยปกติใน IP Protocol เป็น 1 สำหรับ ICMP , 6 สำหรับ TCP และ 17 (0x11) สำหรับ UDP

- *Header Checksum*

Header Checksum มีขนาด 2 ไบต์ จะแสดงการตรวจสอบ bit-level ที่สมบูรณ์ครบถ้วนบน IP Header เท่านั้น ไม่รวม IP Payload โดย Sending Host จะแสดง Checksum เริ่มต้นในการส่ง

- *Source Address*

Source Address มีขนาด 4 ไบต์ ประกอบด้วย IP Address ของ Source Host โดย Network Address Translation (NAT) เป็นการเคลื่อนย้าย IP Datagram โดย NAT ใช้เคลื่อนย้ายระหว่าง Public กับ Private Address เมื่อทำการเชื่อมต่อกับอินเตอร์เน็ต

- *Destination Address*

Destination Address มีขนาด 4 ไบต์ ประกอบด้วย IP Address ของ Destination Host โดย IP Datagram จะเคลื่อนย้ายโดย NAT

- *IP Option*

IP Option เป็นตัวเพิ่มมาตรฐานของ IP Address ถ้า IP Option ไม่กำหนดแต่ละ IP Header ก็ต้องกำหนด IP Option มาให้บางที IP Option มีจุดประสงค์เพื่อใช้ในการทดสอบ Network

2.4.3b 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)

2.4.4 Fragmentation และ Reassembly

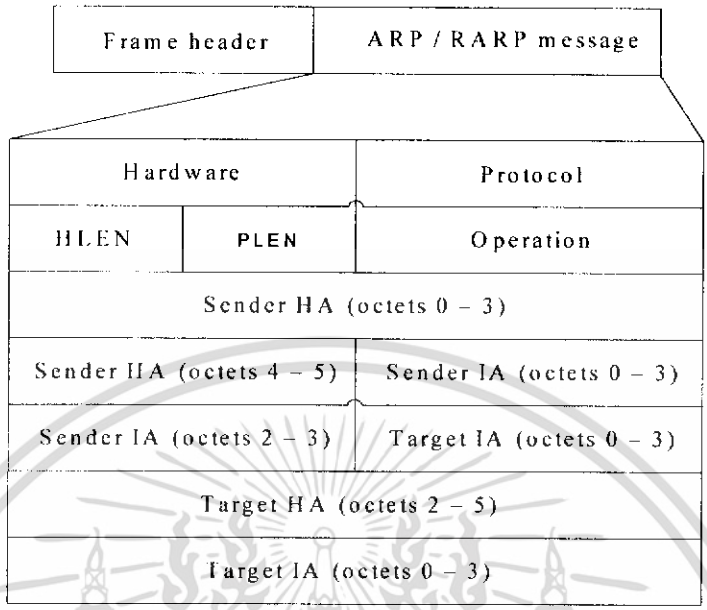
Network และ Technology ต่างๆมักจะมีการกำหนดขนาดสูงสุดของ Packet ไว้ไม่เท่ากันเพื่อเป็นการลด Header/Trailer Overhead จำเป็นต้องส่ง Packet ที่มี Data field ที่มีขนาดใหญ่ที่สุดเท่าที่จะเป็นไปได้ อย่างไรก็ตามเมื่อ Packet มีขนาดใหญ่การครอบครอง Bus จะใช้เวลานาน รวมทั้ง Error Rate จะสูงขึ้นตามไปด้วย ดังนั้น IP จึงถูกออกแบบยินยอมให้ Router สามารถแบ่ง IP Datagram ที่มีขนาดใหญ่กว่าที่จะส่งไปที่ Network ปลายทางได้

การ Fragmentation สามารถทำได้โดยการเอา Header ออกจาก IP Datagram ชุดเริ่มต้น จากนั้นแบ่ง Datagram Data Field เป็นจำนวนที่เหมาะสม แล้วใส่ IP Header ค่าใหม่ลงในแต่ละส่วนที่ถูกแบ่งในรูปของ Datagram ใหม่จากนั้นส่วนแบ่งใหม่นี้ก็จะถูกดำเนินการอย่างเป็นอิสระต่อกันและถูกนำส่งไปจนถึงปลายทาง

2.4.5 Address Resolution Protocol (ARP)

โปรโตคอล ARP (Address Resolution Protocol) ถูกเรียกใช้งานโดยโปรโตคอล IP เพื่อช่วยแปลงหมายเลข IP ไปเป็นหมายเลขฮาร์ดแวร์ปลายทาง ตัวอย่างเช่น เว็บเซิร์ฟเวอร์เครื่องหนึ่งเชื่อมต่ออยู่ในเครือข่ายอินเตอร์เน็ต และในการเชื่อมต่อนี้ต้องอาศัย Network Interface Card (NIC) หรือ LAN card ติดตั้งอยู่ที่ LAN card นี้เองจะมีหมายเลขเฉพาะประจำฮาร์ดแวร์ที่ไม่ซ้ำกับใครเพื่อใช้อ้างอิงการส่งข้อมูลในเครือข่าย แต่เมื่อมาใช้งานในโปรโตคอล TCP/IP ก็จะต้องมีการกำหนดหมายเลข IP Address ประจำตัวเพื่อใช้อ้างอิงกัน และโปรโตคอล ARP จะทำหน้าที่แปลงค่าหมายเลข IP ให้เป็นหมายเลขฮาร์ดแวร์จริงให้ในระดับการทำงานที่ Internetwork Layer นี้ ซึ่งกลไกการแปลงนี้เรียกว่า address resolution

2.4.5a ARP Datagram



รูปที่ 2.12 แสดง ARP Datagram

ในรูปที่ 2.12 แสดง ARP Datagram โดยที่มันไม่สามารถนำมาใช้ได้ทั้งหมดสำหรับ TCP/IP หรือเครือข่ายใดเครือข่ายหนึ่งโดยเฉพาะ โดยมันแค่ถูกกำหนดให้เป็นตัวกลางที่มีความสามารถทำการส่งเฟรม Broadcast กระบวนการของ ARP จะดำเนินการโดยตรงเข้าไปอยู่เหนือ Data Link Layer และจะทำการ Encapsulate ในเฟรม Datalink โดย ARP Datagram จะประกอบไปด้วย

- *Hardware*

ในส่วนนี้จะเป็นตัวบ่งชี้ประเภทของ Network Hardware ของ Datagram ที่ถูกทำให้เกิดขึ้น ซึ่งมีอยู่หลายประเภทดังนี้

Type	**Description**
1	Ethernet (10 Mbps)
2	Experimental Ethernet (3 Mbps)
3	Amateur radio AX.25
4	Proteon PRONET Token Ring
5	Chaos
6	IEEE 802 networks
7	ARCNET

8	Hyperchannel
9	Lanstar
10	Autonet short address
11	LocalTalk
12	LocalNET (IBM PCNet or Sytek Inc. LocalNET

- *Protocol*

ในส่วนนี้จะทำการระบุตัวโปรโตคอลตัวที่ร้องขอที่จะแสดงตัว ซึ่งค่าที่ใช้จะเหมือนกับในประเภทของอีเทอร์เนตที่อยู่ในเฟรมอีเทอร์เนต โดยจะใช้ค่า 0x0800 สำหรับ IP

- *HLEN*

จะเป็นตัวกำหนดความยาวของ Hardware Address ใน Octets โดยปกติจะมีค่าเป็น 6 สำหรับ IEEE LAN MAC Address

- *PLEN*

จะเป็นตัวกำหนดความยาวของ Network Layer Address ใน Octets โดยปกติจะมีค่าเป็น 4 สำหรับ IP

- *Operation*

จะมีค่าเป็น 1 สำหรับ ARP Request และ 2 สำหรับ ARP Reply ส่วน RARP จะมีค่าเป็น 3 สำหรับ RARP Request และ 4 สำหรับ RARP Reply

- *Address*

2.4.5 b 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

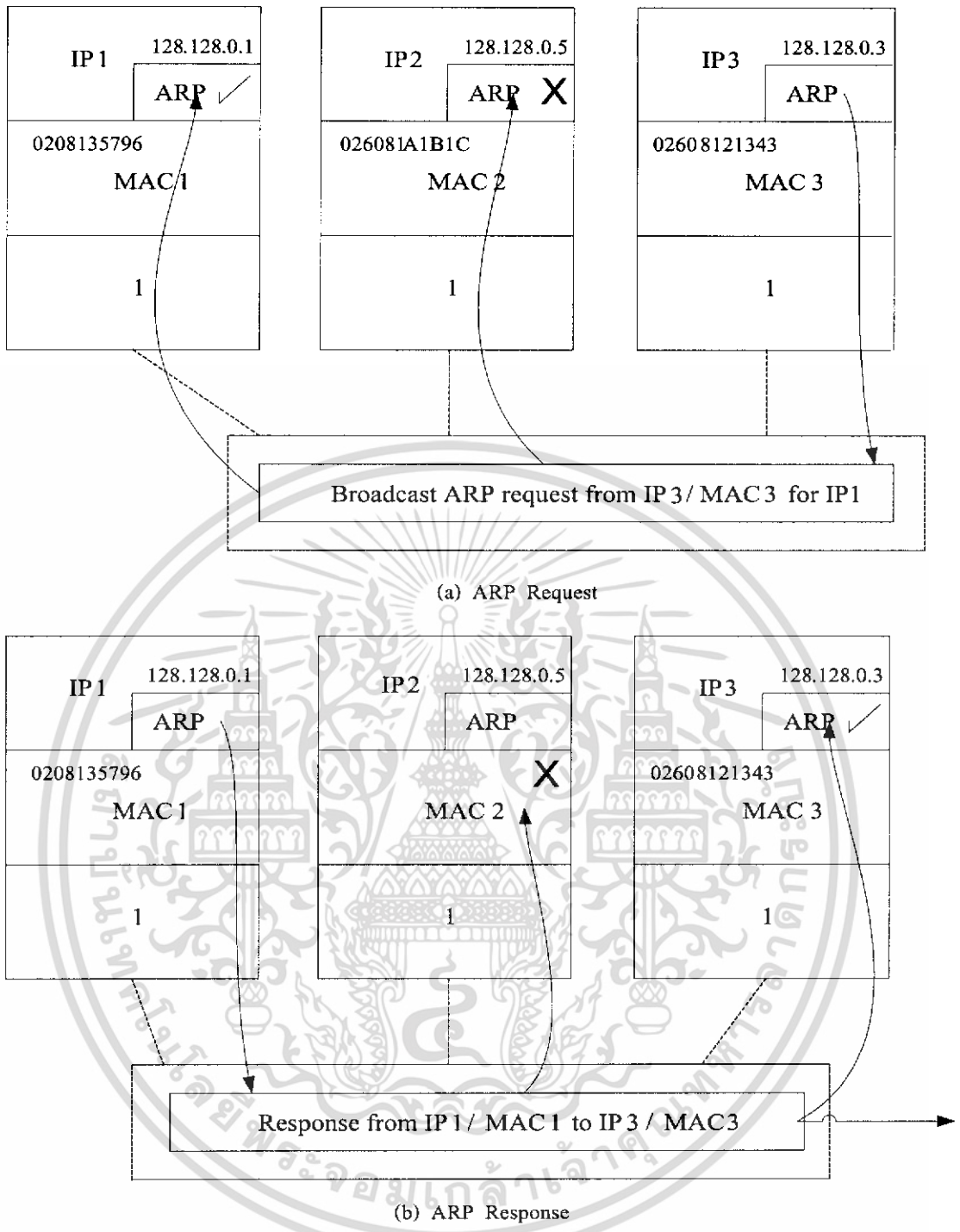
1. เครื่อง Client จะใช้ค่าของ IP Address เพื่อค้นหา MAC Address ที่สอดคล้องกัน ซึ่งเก็บไว้ใน ARP Cache

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 ดังกล่าวมัน จะทำการส่ง 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 2.13 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 ที่จะถูกรับ โดย ARP Software บนทุกๆจุดในเครือข่าย 128.128.0.1 จะยอมรับ Address ของมันในการร้องขอ และจะส่ง ARP Reply กลับไป แต่จะใช้ค่า MAC Address ที่ ARP Request ส่งมา ARP Reply จะถูกดำเนินการและถูกละทิ้งไปอย่างรวดเร็วโดยการ์ด LAN ที่ทุกๆจุดบนเครือข่ายขณะที่มันเป็น Unicast frame กับ Destination Address ที่ผิด



รูปที่ 2.13 แสดงตัวอย่างกระบวนการของ ARP, (a) ARP Request , (b) ARP Response

2.4.6 Reverse Address Resolution Protocol (RARP)

วิธีการ ARP ช่วยแก้ปัญหาในการค้นหาที่อยู่ของข้อมูลที่ใช้การกำหนดที่อยู่แบบฮาร์ดแวร์ แต่ถ้าทราบที่อยู่แบบฮาร์ดแวร์แล้วต้องการแปลงที่อยู่เป็น IP จะทำอย่างไร ปัญหานี้มักเกิดขึ้นกับเครื่องคอมพิวเตอร์ที่เริ่มทำงานด้วยการอ่านข้อมูลทั้งหมดจากเครื่อง Host เครื่องประเภทนี้จะทราบเพียงที่อยู่ของตนเองจากอุปกรณ์สื่อสารเครือข่ายเท่านั้น

การค้นหาคำตอบสามารถทำได้โดยวิธีควบคุมการสื่อสารแบบ ARP ย้อนกลับ หรือ RARP (Reverse Address Resolution Protocol) วิธีการนี้ Computer ที่เพิ่งจะเริ่มทำงาน (หรือเครื่องใดก็แล้วแต่) จะส่งคำถามออกไปในทำนอง " ที่อยู่ขนาด 48 Bits แบบฮาร์ดแวร์ของฉันคือ 14.04.05.18.01.25 มีใครทราบที่อยู่ IP ของฉันบ้าง " เครื่องที่ให้บริการ RARP จะตรวจดูข้อมูลในตารางข้อมูลของตนเองแล้วจึงส่งหมายเลข IP กลับไปให้ วิธีการนี้ช่วยให้เกิดความอ่อนตัวและเพิ่มประสิทธิภาพในการใช้หมายเลข IP เนื่องจากผู้ใช้ไม่มีหมายเลข IP เป็นของตนเอง ผู้ควบคุมระบบสามารถกำหนดหมายเลข IP ใดๆที่ไม่มีผู้ใช้งานในขณะนั้นให้ใช้ได้ หมายเลข IP ในที่นี้จึงเป็นเสมือนสมบัติส่วนกลางที่ทุกคนใช้ร่วมกัน

ข้อด้อยของวิธี RARP คือการที่ผู้ใช้จะส่งคำถามโดยใช้หมายเลข "1" จำนวน 48 ตัวเป็นที่อยู่ของผู้ให้บริการ หมายเลขนี้เป็นหมายเลขพิเศษที่ Router จะไม่ยอมส่ง Packet ผ่านไปยังเครือข่ายอื่นเลย ฉะนั้นผู้ให้บริการ RARP จะต้องมีอยู่ประจำทุกเครือข่าย อย่างไรก็ตาม โปรโตคอลแบบ BOOTP ได้รับการพัฒนาขึ้นมาเพื่อแก้ปัญหานี้โดยการใช้ Packet UDP แทน Packet ชนิดนี้สามารถส่งไปได้ทั่วทุกเครือข่ายและยังให้ข้อมูลอื่นเพิ่มเติม เช่น หมายเลข IP ของผู้ให้บริการแฟ้มข้อมูล หมายเลข IP ของ Router อัตโนมัติ และตารางข้อมูลเครือข่ายย่อยเป็นต้น

2.4.7 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 คือ การแจ้งหรือแสดงข้อความจากระบบเพื่อบอกให้ผู้ใช้ทราบว่าเกิดอะไรขึ้นในการส่งผ่านข้อมูลนั้น ซึ่งปัญหาส่วนมากที่พบคือ ส่งไปไม่ได้หรือปลายทางรับข้อมูลไม่ได้ เป็นต้น นอกจากนี้โปรโตคอล ICMP ยังถูกเรียกใช้งานจากเครื่อง Server และ Router อีกด้วยเพื่อแลกเปลี่ยนข้อมูลที่ใช้ควบคุม ส่วนรูปแบบการทำงานของโปรโตคอล ICMP นั้นจะทำควบคู่กับโปรโตคอล IP ในระบบเดียวกันและข้อความต่างๆที่แจ้งให้ทราบจะถูกผนึกอยู่ภายในข้อมูล IP (IP datagram) อีกทีหนึ่ง

ข้อความที่โปรโตคอล ICMP ส่งนั้นแบ่งออกได้เป็น 2 แบบ คือ ICMP Error Message หรือข้อความแจ้งข้อผิดพลาด และ ICMP Query หรือข้อความเรียกขอข้อมูลเพิ่ม ตัวอย่างกลไกการทำงานของโปรโตคอล ICMP เช่น เมื่อมีการส่งผ่านข้อมูลจากผู้ใช้ไปยังปลายทางที่ไม่ถูกต้องหรือขณะนั้นเครื่องปลายทางเกิดปัญหาจนไม่สามารถรับข้อมูลได้ ที่ Router จะส่งข้อความแจ้งเป็น ICMP Message ที่ชื่อ Destination Unreachable ให้กับผู้ส่งข้อมูลนั้น นอกจากนี้ตัวข้อมูลที่แจ้งข้อความก็จะมีส่วนของข้อมูล IP Datagram ที่เกิดปัญหาด้วย ดังนั้นเมื่อผู้ส่งข้อมูลได้รับข้อความแจ้งแล้วก็จะทราบได้ว่าจุดที่เกิดปัญหานั้นอยู่ที่ใด

ICMP จะใช้ IP Datagram ช่วยในการขนส่งข้อมูลกลับและจะเดินทางระหว่างส่วนที่เกี่ยวเนื่องกัน ICMP error messages ถูกทำให้เกิดขึ้นโดย node ที่ยอมรับว่ามีปัญหาส่งขึ้น และมันจะทำการส่งกลับมาที่ Address เริ่มต้นของ Datagram ที่เกิดปัญหา โดย ICMP สามารถ Destination node ตัว

สุดท้ายสุดหรือระหว่างกลาง Routers และ Hosts ทั้งคู่สามารถที่เป็นแหล่งกำเนิดของ IP Datagram โดยสรุปได้ว่าโปรโตคอลนี้จะอาศัย IP Datagram ในการส่งสัญญาณควบคุมเหล่านั้น คำสั่งควบคุมที่สำคัญของ ICMP ได้แก่

- *Flow Control*

จะถูกส่งโดยเครื่องปลายทางหรือเครื่อง gateway ที่อยู่ในเส้นทางเดินของ Datagram สัญญาณนี้จะส่งกลับไปยังต้นทางเมื่อมี Datagram ถูกส่งมาเร็วเกินไปจนมันประมวลผลไม่ทัน ซึ่งจะส่งผลให้เครื่องต้นทางหยุดส่ง Datagram ชั่วขณะ

- *Detecting Unreachable Destinations*

สัญญาณนี้จะถูกส่งกลับไปยังต้นทางเพื่อบอกว่าหาปลายทางไม่พบ

- *Redirecting Routes*

สัญญาณนี้จะเกิดขึ้นเฉพาะใน Network ที่มี gateway อยู่มากกว่าหนึ่งตัวโดย gateway ตัวใดตัวหนึ่งอาจส่งสัญญาณนี้ไปยังต้นทางเพื่อบอกให้มันเปลี่ยนเส้นทางการส่ง Datagram ไปยัง gateway ตัวอื่นโดยอาจมีสาเหตุมาจากเส้นทางขัดข้อง

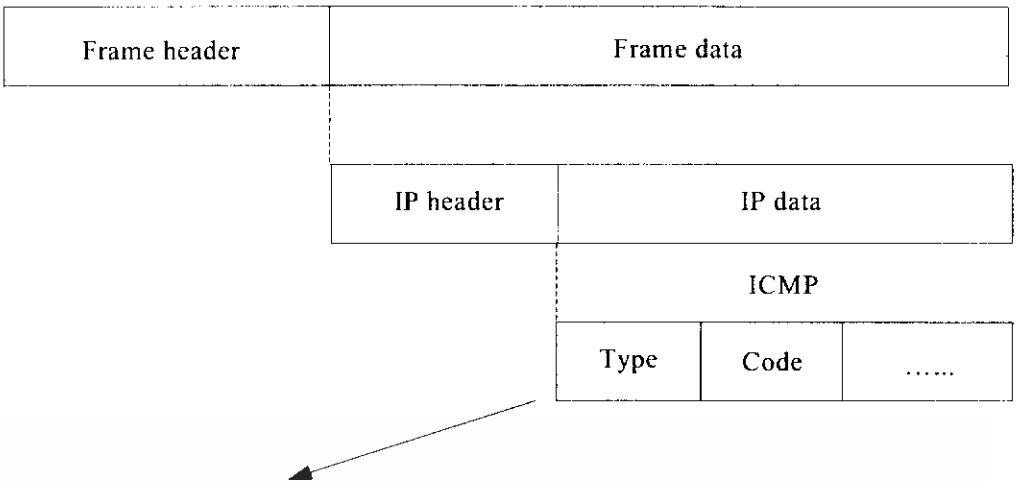
- *Checking Remote Hosts*

เครื่องๆหนึ่งสามารถทำการส่ง Echo Message ไปยังเครื่องใดๆเพื่อทดสอบว่า Internet Protocol ของเครื่องนั้นทำงานอยู่หรือไม่ เมื่อเครื่องปลายทางได้รับสัญญาณนี้ก็จะส่งสัญญาณเดิมนี้กลับมา ตัวอย่างของการใช้ Message นี้คือ คำสั่ง Ping ของ UNIX ซึ่งเรามักใช้ตรวจสอบเครื่องปลายทางว่าทำงานอยู่หรือไม่

รูปที่ 2.14 แสดงรูปแบบพื้นฐานของ ICMP message encapsulated ใน IP Datagramใน ICMP มีหมายเลข IP โปรโตคอลของตัวมันเอง ดังนั้น IP Layer รู้ว่ามันรับ ICMP แม้ว่า ICMP ใช้ IP Layer ที่เป็นตัวพิจารณาว่า IP ภายในเป็นของใครเพราะว่ามันไม่สามารถจัดเตรียมให้กับ Layer ที่สูงกว่าได้

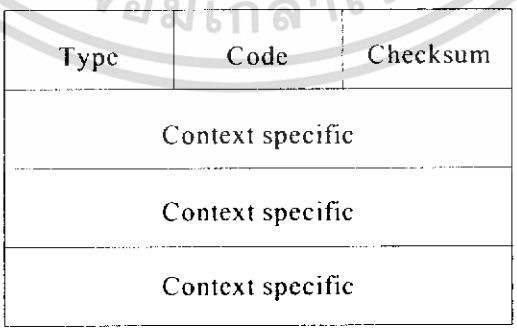
นับตั้งแต่ IP Message ที่ถูกขนย้ายใน IP มันจะถูกทิ้งไปเหมือนกับ IP Datagram โดยมันจะไม่สามารถรักษาสถานภาพไว้ได้ ICMP Message จะไม่ถูกทำให้เกิดขึ้นในกรณีที่ ICMP Message เกิดความผิดพลาดเกิดขึ้น

รูปแบบพื้นฐานของ ICMP Datagram แสดงไว้ในรูปที่ 2.15 โดยจะประกอบไปด้วยการแบ่งประเภทของ ICMP Message และ Code ที่ต้องจัดเตรียมรายละเอียด Checksum ต้องถูกนำมาใช้เพราะ 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

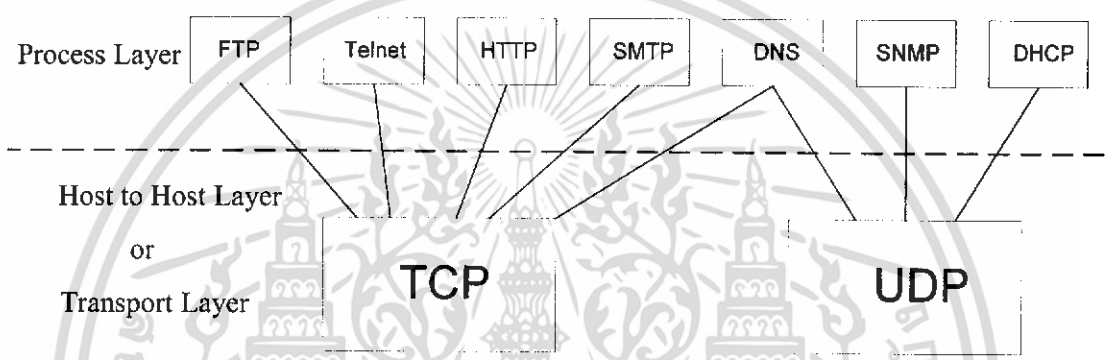
รูปที่ 2.14 แสดง ICMP encapsulated ใน IP และประเภทของ ICMP



รูปที่ 2.15 แสดงรูปแบบของ ICMP Datagram

2.5 Host – To – Host Layer

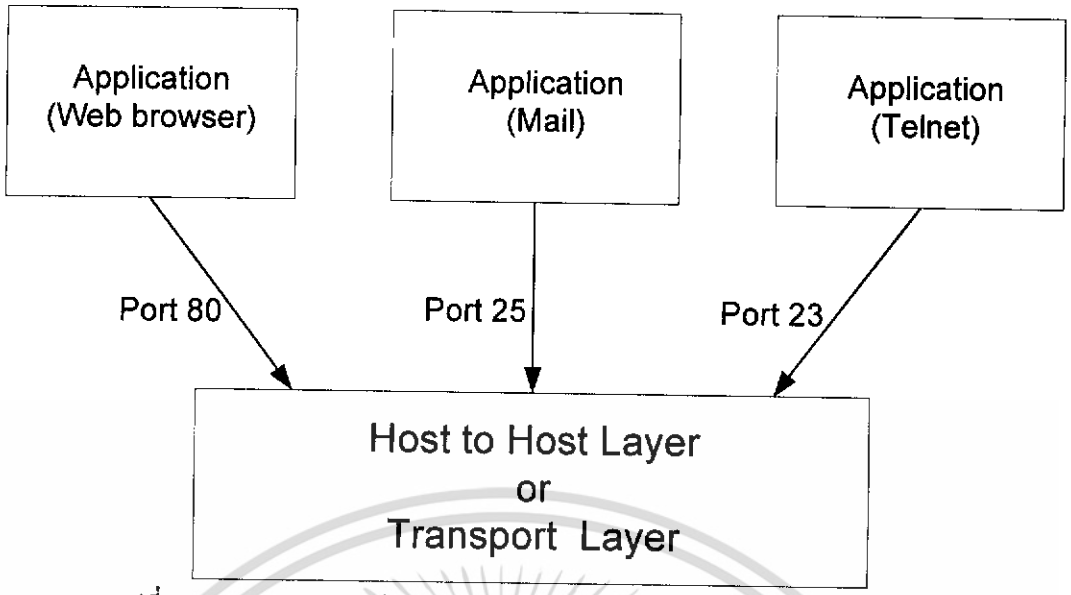
การทำงานที่ชั้นของ Host – to – Host Layer นี้จะมีบทบาทในการจัดการต่อจาก Process Layer บางครั้งเรามักเรียกชั้น Host – to – Host ว่าเป็น Transport Layer ซึ่งจะไม่ใช่ชั้นของ Transport Layer ในมาตรฐาน OSI model การทำงานของ Host – to Host Layer นี้จะมีการสร้าง connection หรือการเชื่อมต่อกันระหว่างแอพพลิเคชั่นกับ Host – to – Host Layer โดยจุดที่เชื่อมกันเพื่อรับส่งข้อมูลที่เรียกว่า port หรือ socket (คำว่า port ในที่นี้ไม่ได้หมายถึง port ทางฮาร์ดแวร์) และในแต่ละแอพพลิเคชั่นก็จะสร้างการเชื่อมต่อผ่าน port ได้พร้อมกันหลายแอพพลิเคชั่น ซึ่งการใช้งาน port ของแต่ละแอพพลิเคชั่นที่อยู่ในชั้น Process Layer จะแตกต่างกันตามหมายเลขที่กำหนดไว้ และแต่ละโปรโตคอลจะมีการใช้งาน port หมายเลขต่างๆไม่ซ้ำกัน ดังรูปที่ 2.16



รูปที่ 2.16 แสดงการใช้งาน port ของแต่ละโปรโตคอล

เมื่อแอพพลิเคชั่นทำงานผ่านโปรโตคอลในชั้น Process Layer จะมีการส่งผ่านข้อมูลไปยัง Host – to – Host Layer ที่ชั้นนี้จะมีการเชื่อมต่อผ่าน port ที่กำหนด ทำให้การรับส่งข้อมูลในแต่ละโปรโตคอลทำได้ถูกต้อง ถึงแม้ว่าในเครื่องเซิร์ฟเวอร์ที่ให้บริการจะมีการทำงานอยู่หลายโพรเซสส์ที่แตกต่างกันก็ตาม หรือมีผู้ใช้บริการเข้ามาใช้งานพร้อมกันจำนวนมากและหลายแอพพลิเคชั่นในเวลาเดียวกัน ในชั้น Host – to – Host หรือ Transport Layer ของ TCP/IP นี้จะมีโปรโตคอลทำงานอยู่ 2 โปรโตคอลที่แตกต่างกัน คือ โปรโตคอล TCP และ โปรโตคอล UDP (User Datagram Protocol) ในการส่งผ่านข้อมูลลงไปที่ชั้นถัดๆไป เราจะเห็นว่าโปรโตคอล TCP และ UDP จะถูกผนึกเข้าไปในโปรโตคอล IP อีกทีหนึ่งและส่งต่อไปยังเครือข่ายอินเตอร์เน็ตต่อไป

รูปที่ 2.18 แสดง Ports ที่ใช้กับ UDP และ TCP



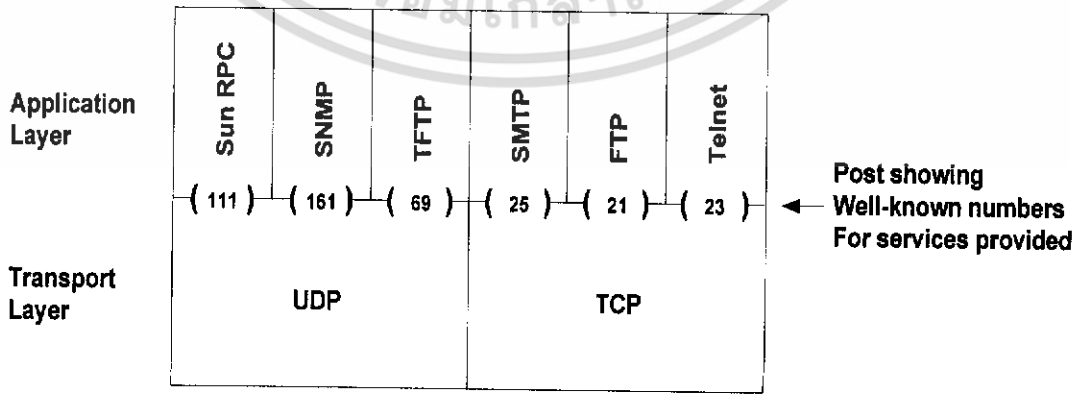
รูปที่ 2.17 แสดงการส่งข้อมูลจาก Application ไปยัง Host – to – Host Layer

ตัวโปรโตคอล TCP และ โปรโตคอล UDP จะมีแอพพลิเคชั่นเฉพาะเพื่อเรียกใช้งานแยกกันคือ แอพพลิเคชั่นที่ใช้โปรโตคอล FTP , Telnet , HTTP และ SMTP จะมีการส่งผ่านข้อมูลโดยเรียกใช้โปรโตคอล TCP ส่วนแอพพลิเคชั่นที่ใช้โปรโตคอล SNMP และ DHCP จะส่งผ่านข้อมูลโดยเรียกใช้โปรโตคอล UDP และสำหรับโปรโตคอล DNS นั้น จะสามารถเรียกใช้งานได้ทั้ง TCP และ UDP

2.5.1 พอร์ต

UDP และ TCP จะใช้ Ports ในการส่งข่าวสารไปยังชั้น Application Layer โดยที่ Ports เป็น 16 บิตแอดเดรสที่เป็นหมายเลขที่เป็นที่รู้จักซึ่งอยู่ในช่วง 0 ถึง 255 ดังแสดงในรูปที่ 2.18 ใน words อื่นๆ หมายเลข Ports จะถูกแบ่งสำหรับการเข้า Application Layer ถ้า Application เป็นตัวพัฒนาในการสร้างโปรแกรมให้ทำงานเหนือ UDP หรือ TCP

ทางเลือกในการที่จะกำหนด Ports โดย NFS จะใช้ Portmapper ที่อนุญาตให้ Ports ใหม่ที่มีค่าชัดเจนและถูกเก็บค่า หรือ Ports ที่ถูกใช้โดยเฉพาะเจาะจงที่ถูกค้นพบบนการร้องขอ



รูปที่ 2.18 แสดง Ports ที่ใช้กับ UDP และ TCP

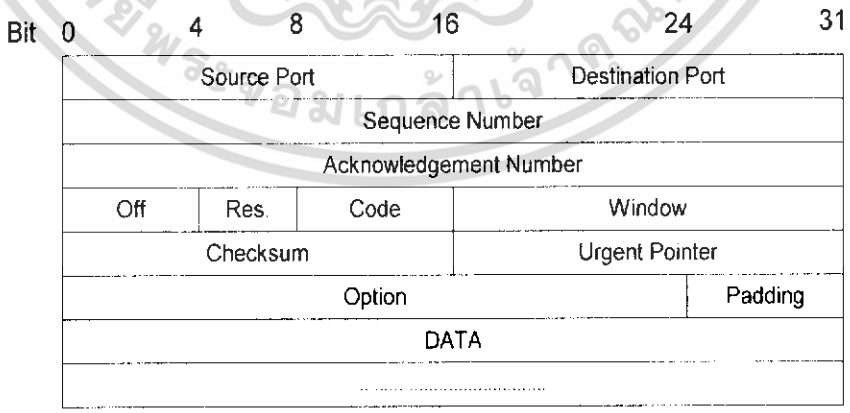
2.5.2 โปรโตคอล TCP

โปรโตคอล TCP (Transmission Control Protocol) เป็นโปรโตคอลที่มีการรับส่งข้อมูลแบบ stream oriented protocol หมายความว่าการรับส่งข้อมูลจะไม่คำนึงถึงปริมาณข้อมูลที่จะส่งไป แต่จะแบ่งข้อมูลเป็นส่วนย่อยๆก่อนแล้วจึงส่งไปยังปลายทางอย่างต่อเนื่องเป็นลำดับข้อมูล ในกรณีที่ข้อมูลส่วนใดส่วนหนึ่งสูญหายไปก็จะส่งข้อมูลส่วนนั้นใหม่อีกครั้ง สำหรับปลายทางก็จะทำหน้าที่จัดเรียงส่วนของข้อมูล datagram ดังนั้นแอพพลิเคชั่นหรือโพรเซสส์ใดที่อาศัยการส่งผ่านข้อมูลด้วยโปรโตคอล TCP จะต้องใช้หน่วยความจำและขนาดของช่องสัญญาณ (bandwidth) มากกว่า UDP

การติดต่อระหว่างกันจะต้องเป็นแบบ connection-oriented คือต้องมีการสร้างการติดต่อกันเป็น session ทั้ง 2 ด้านเสียก่อนแล้วจึงจะรับส่งข้อมูลไปได้พร้อมกัน (full duplex) เหมือนกับการใช้โทรศัพท์ติดต่อกัน เมื่อผู้ติดต่อต้นทางเรียกให้ฝ่ายตรงข้ามรับสายแล้วจึงเริ่มการสนทนา เช่น พูดคำว่า “สวัสดี” หรือ “ฮัลโหล” กันก่อนเพื่อให้แน่ใจว่าฝ่ายตรงข้ามพร้อมจะติดต่อด้วยจากนั้นจึงเริ่มต้นติดต่อกัน และเมื่อต้องการจะยกเลิกการติดต่อก็จะมีการพูดคำว่า “สวัสดี” ให้ฝ่ายตรงข้ามทราบว่าจะยกเลิกการติดต่อและวางสายไป ซึ่งในระหว่างการติดต่อกันนั้นแม้ว่าฝ่ายใดฝ่ายหนึ่งหรือทั้งสองฝ่ายจะเงียบไป คือ ไม่พูดอะไรเป็นเวลานานๆแต่การเชื่อมโยงระหว่างทั้งสองด้านยังคงมีอยู่ไม่ขาดไปจนกว่าฝ่ายใดฝ่ายหนึ่งจะวางสาย เช่นเดียวกับการติดต่อกันด้วยกลไกโปรโตคอล TCP เมื่อแอพพลิเคชั่นต้องการส่งผ่านข้อมูลจะใช้โปรโตคอลที่เหมาะสมในชั้น Process Layer ติดต่อไปและมีการสร้างช่องส่งข้อมูลผ่าน port ที่กำหนดเพื่อส่งผ่านข้อมูลไปยังโปรโตคอล TCP

ในระหว่างการรับส่งข้อมูลนี้โปรโตคอล TCP จะเพิ่มขบวนการสอบทานข้อมูลเพื่อให้ข้อมูลมีความถูกต้องไม่ผิดพลาดไปจากเดิม โดยการส่งสัญญาณสอบทานข้อมูล (acknowledgment) และส่งข้อมูลให้ใหม่อีกครั้งถ้าปลายทางไม่ได้รับหรือเกิดความผิดพลาดขึ้น

ความน่าเชื่อถือของการส่งผ่านข้อมูลโดยโปรโตคอล TCP จะมีมากกว่าแต่ก็ต้องอาศัยทรัพยากรของระบบมากกว่าในการทำงานเช่นกัน



รูปที่ 2.19 แสดงโครงสร้างของโปรโตคอล TCP

2.5.3 โปรโตคอล UDP

ใน Host – to – Host Layer นอกจากจะมีโปรโตคอล TCP ทำงานแล้วก็ยังมีโปรโตคอล UDP (User Datagram Protocol) ในการส่งข้อมูลแต่ละครั้งและไม่มีการส่งข้อมูลใหม่อีกในกรณีที่เกิดความผิดพลาดของการส่งข้อมูล เมื่อเป็นเช่นนี้แอพพลิเคชั่นหรือโพรเซสส์ใดที่ต้องอาศัยโปรโตคอล UDP จะเป็นแบบที่ทั้งสองด้านไม่จำเป็นต้องอาศัยการสร้างช่องทางเชื่อมต่อกัน (connectionless) ระหว่างเครื่องเซิร์ฟเวอร์ให้บริการกับเครื่องที่ขอใช้บริการ โดยไม่ต้องแจ้งให้ฝ่ายรับข้อมูลเตรียมรับข้อมูลเหมือนโปรโตคอล TCP และไม่มีการตรวจสอบความถูกต้องครบถ้วนในการรับส่งข้อมูลนั้นๆด้วยเนื่องจากโปรโตคอล UDP ไม่มีสัญญาณสอบทานข้อมูล (acknowledgement) ในการส่งข้อมูลแต่ละครั้งและไม่มีการส่งข้อมูลใหม่อีกในกรณีที่เกิดความผิดพลาดของการส่งข้อมูล เมื่อเป็นเช่นนี้แอพพลิเคชั่นหรือโพรเซสส์ใดที่ต้องอาศัยโปรโตคอล UDP ในการส่งผ่านข้อมูลก็อาจจะต้องสร้างขบวนการตรวจสอบข้อมูลขึ้นมาเอง

ตัวอย่างขั้นตอนกลไกการทำงานโดยใช้โปรโตคอล UDP มีดังต่อไปนี้

1. ในชั้นของ Process Layer เมื่อโปรแกรมควบคุมอุปกรณ์เครือข่าย เช่น โปรแกรม Network Management ต้องการส่งข้อมูลไปยังอุปกรณ์ที่ต้องการ แอพพลิเคชั่นนั้นจะติดต่อผ่านโปรโตคอล SNMP ในชั้น Process Layer

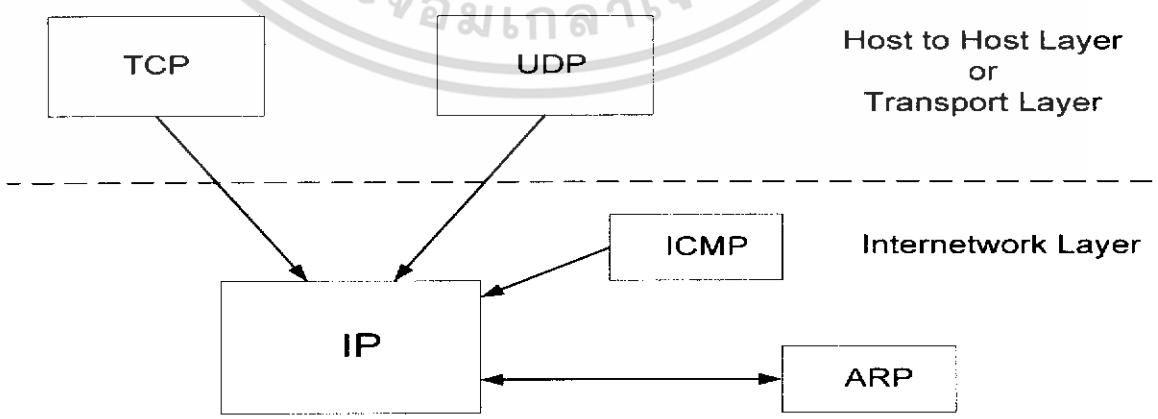
2. โปรโตคอล SNMP จะติดต่อกับโปรโตคอล UDP ในชั้นถัดไปเพื่อขอติดต่อผ่าน port ที่กำหนด

3. โปรโตคอล SNMP เตรียมข้อมูลที่จะส่งรวมทั้งที่อยู่ปลายทาง

4. โปรโตคอล SNMP ส่งผ่านข้อมูลให้โปรโตคอล UDP ที่อยู่ในชั้น Host – to – Host Layer

5. โปรโตคอล UDP ทำหน้าที่ผนึกข้อมูลหรือ datagram นั้นไปกับโปรโตคอล IP ในชั้นถัดลงไปเพื่อส่งข้อมูลออกจากเครื่อง

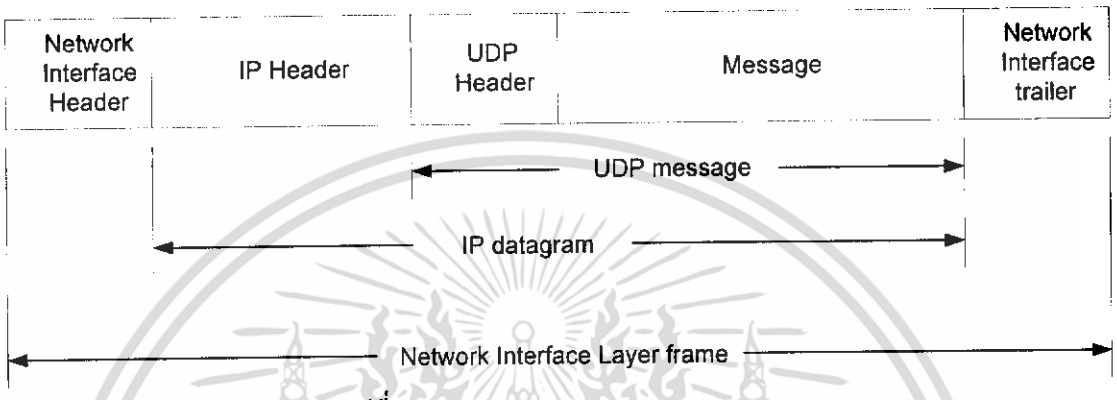
ซึ่งจะเห็นว่ามีกลไกที่ต่างจากการส่งข้อมูลด้วยโปรโตคอล TCP ซึ่งจะต้องมีการติดต่อกันก่อนและทั้งสองฝ่ายรับทราบการรับส่งข้อมูลของช่องการส่งข้อมูลนั้น



รูปที่ 2.20 แสดงกลไกการส่งข้อมูลด้วยโปรโตคอล UDP

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 2.21 UDP Message encapsulation

UDP Header

มีขนาด 8 ไบต์ โดยประกอบด้วย 4 ส่วนดังรูปที่ 2.22

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

รูปที่ 2.22 แสดงโครงสร้างของ UDP Header

- *Source Port*

มี 2 ไบต์ใช้ระบุเป็น Source Application Layer Protocol ทำการส่ง UDP Message โดย Source Port เป็น port ที่ใช้ในการเลือก เมื่อใดที่ไม่ได้ใช้มัน มันจะตั้งค่าเป็น 0x00-00 IP multicast traffic เปรียบเสมือน videocasts 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 (ขนาดของ UDP 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

ตำแหน่ง	ชื่อ	อธิบาย
บิต 0-15	Source port number	หมายเลขพอร์ตต้นทางที่ส่ง datagram นี้ มีความยาว 16 บิต
บิต 16-31	destination port number	หมายเลขพอร์ตปลายทางที่จะเป็นผู้รับ datagram มีความยาว 16 บิตเช่นกัน
บิต 32-47	UDP length	ความยาวของ datagram ทั้งส่วน Header และ data นั่นหมายความว่า ค่าที่น้อยที่สุดในฟิลด์นี้คือ 8 ซึ่งเป็นขนาดของ Header
บิต 48-63	Checksum	เป็นตัวตรวจสอบความถูกต้องของ UDP datagram และจะนำข้อมูลบางส่วนใน IP Header มาคำนวณด้วย

UDP Port

UDP Port จะแสดงที่ตั้งหรือแถวของ message ที่ชัดเจนสำหรับการส่ง message ถึง Application Layer protocol โดยใช้ UDP services รวมถึงในแต่ละตัวของ UDP message เป็น Source Port และ Destination Port ซึ่ง Internet Assigned Number Authority (IANA) จะเป็นตัวกำหนดหมายเลข Port

ตารางที่ 2.6 แสดง UDP Port Numbers

Port Numbers	Application Layer Protocol
53	Domain Name System (DNS)
67	BOOTP client (Dynamic Host Configuration Protocol [DHCP])
68	BOOTP server (DHCP)
69	Trivial File Transfer Protocol (TFTP)
137	NetBIOS Name Service
138	NetBIOS Datagram Service
161	Simple Network Management Protocol (SNMP)
520	Routing Information Protocol (RIP)
445	Direct hosting of server Message Block (SMB) datagram over TCP/IP
1812 , 1813	Remote Authentication Dial-In User Service (RADIUS)

UDP Checksum

Checksum เป็น เลข 16 บิตถูกคำนวณด้วยวิธี one's complement โดยนำ Pseudo Header และข้อมูลทั้งหมดใน UDP Datagram มาคำนวณ

Pseudo Header เป็นข้อมูลที่อยู่ในส่วนของ IP Header ประกอบไปด้วยฟิลด์ source IP address , destination IP address , zero , protocol , UDP length ดังแสดงในรูปที่ 2.23

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

รูปที่ 2.23 แสดง Pseudo Header

หากค่า Checksum ที่คำนวณออกมาเป็น 0 ค่า checksum จะถูกเซ็ตเป็น 1 ทั้งหมดแทน (มีค่าเท่ากันในระบบ 1's complement) ทั้งนี้เพราะในบางแอพพลิเคชั่นที่ไม่ต้องการตรวจสอบค่า checksum ในระดับ UDP จะเซ็ตค่านี้เป็น 0 (disable checksum)

2.6 Process Layer

การแสดงลำดับชั้นการทำงานของโปรโตคอล TCP/IP เทียบกับมาตรฐาน OSI model นั้น ในชั้นบนสุดเรียกว่า Process Layer ทำงาน 2 หน้าที่เทียบได้กับ Application Layer และ Presentation Layer ของมาตรฐาน OSI model ซึ่งในชั้นนี้จะรองรับการทำงานของแอพพลิเคชั่นต่างๆที่ทำงานเป็นโพรเซสส์อยู่ในเครื่องเซิร์ฟเวอร์ที่ให้บริการและเครื่องที่ขอใช้บริการหรือไคลเอนต์ (client) ซึ่งจะติดต่อกันผ่านโปรโตคอลเฉพาะแอพพลิเคชั่นอีกทีหนึ่ง ตัวอย่างเช่น เมื่อผู้ใช้งานอินเตอร์เน็ตต้องการโอนถ่ายไฟล์หรือ download ข้อมูลจากเครื่องเซิร์ฟเวอร์ที่ให้บริการ โดยอาจจะเรียกใช้โปรแกรม FTP client ทั่วไป เช่น โปรแกรม WS_FTP ติดต่อกับโพรเซสส์ FTP ที่กำลังให้บริการอยู่ที่เครื่องเซิร์ฟเวอร์ จากนั้นตัวโพรเซสส์ FTP ก็จะเรียกใช้โปรโตคอล FTP (File Transfer Protocol) เพื่อทำการโอนถ่ายไฟล์นี้ หรือถ้าผู้ใช้ต้องการเรียกใช้งานคอมพิวเตอร์ที่อยู่ห่างไกลออกไปด้วยการใช้โปรแกรม Telnet ที่เครื่องเซิร์ฟเวอร์ให้บริการ ตัวโพรเซสส์ Telnet ที่ทำงานอยู่ก็จะเรียกใช้โปรโตคอล Telnet เพื่อติดต่อกัน หรือในกรณีที่มีการเรียกใช้โปรแกรม web browser เช่น Netscape Navigator เพื่อเรียกดูเว็บไซท์ CNN ที่เครื่องซึ่งให้บริการเว็บของ CNN ก็จะมีโพรเซสส์ HTTP (Hyper Text Transfer Protocol) ทำงานอยู่ และจะติดต่อกับผู้ใช้ผ่านโปรโตคอล HTTP เป็นต้น

การทำงานของแอพพลิเคชั่นต่างๆจะอยู่ที่ Process Layer นี้ และมีการติดต่อกันตามแต่ละโปรโตคอลเฉพาะแล้วแต่แอพพลิเคชั่นที่ใช้งาน จากการที่ Process Layer ของ TCP/IP รองรับให้โปรโตคอลอื่นทำงานได้หลายโพรเซสส์และหลายโปรโตคอลได้พร้อมกันนั้น ทำให้ผู้ใช้สามารถเปิดโปรแกรมใช้งานได้หลายๆโปรแกรมพร้อมกัน เช่น เปิดโปรแกรม Internet Explorer เพื่อเรียกดูเว็บเพจพร้อมกับใช้งานโปรแกรม Outlook Express เพื่อรับส่งอีเมล์ไปพร้อมกันได้โดยไม่ต้องรอให้ทำงานอย่างหนึ่งอย่างใดเสร็จก่อน หรือในปัจจุบันมีการพัฒนาโปรแกรม web browser ให้สามารถเรียกใช้งานโปรโตคอลอื่นๆได้มากขึ้นทำให้เราสามารถใช้โปรแกรม web browser โอนถ่ายไฟล์ข้อมูลที่ใช้โปรโตคอล FTP ได้โดยไม่ต้องไปหาโปรแกรมอื่นมาใช้

โปรโตคอลหลักๆที่ทำงานใน Process Layer ซึ่งผู้ใช้มักจะคุ้นเคยกันดีได้แก่ FTP (File Transfer Protocol) , Telnet , HTTP (Hyper Text Transfer Protocol) , SMTP(Simple Mail Transfer protocol) นอกจากนี้ยังมีโปรโตคอลอื่นที่อยู่เบื้องหลังซึ่งทำงานโดยที่ผู้ใช้ไม่ได้มีการใช้งานโดยตรง เช่น

- โปรโตคอล DNS (Domain Name System) ที่ทำหน้าที่แปลงข้อมูลชื่อ domain name หรือชื่อเว็บไซท์ทั้งหลายให้เป็นหมายเลข IP address

- โปรโตคอล DHCP (Dynamic Host Configuration Protocol) ทำหน้าที่แจกจ่ายข้อมูลพารามิเตอร์ของเครือข่ายให้กับเครื่องลูกข่ายที่เชื่อมต่ออยู่

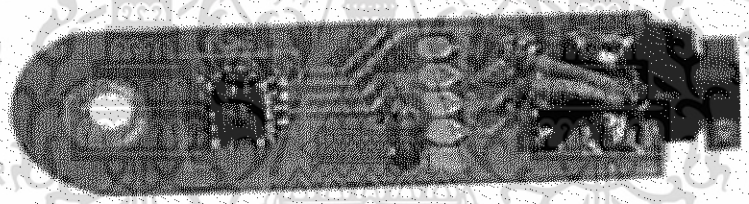
2.7 เซนเซอร์วัดความชื้นสัมพัทธ์และอุณหภูมิ

2.7.1 คุณสมบัติของเซนเซอร์ SHT-15

เป็นเซนเซอร์วัดความชื้นและอุณหภูมิจาก Sensirion มีขนาดเล็กและเพื่อความสะดวกในการใช้งานจึงได้ทำการติดตั้งลงบนแผ่นวงจรพิมพ์และต่อคอนเน็คเตอร์ 8 ขา เพื่อให้สามารถติดตั้งลงบนแผงต่อวงจรหรือเบรกบอร์ดเพื่อทำการทดลองได้ง่าย รวมไปถึงการนำไปประยุกต์ใช้งานจริงด้วย

คุณสมบัติทางด้านเทคนิค

- ทำหน้าที่เป็นทั้งตัววัดความชื้นและอุณหภูมิได้ในตัวถังเดียวกัน
- สามารถวัดความชื้นสัมพัทธ์ได้ในช่วง 0-100 % และอุณหภูมิช่วง -40 ถึง 120 องศาเซลเซียส
- สามารถกำหนดความละเอียดของย่านการวัดได้
- มีขนาดเล็กและกินพลังงานต่ำ
- ทำงานในย่านแรงดันไฟเลี้ยง +2.4 ถึง +5.5 V
- เสถียรภาพในการทำงานสูง



รูปที่ 2.24 แสดงรูปร่างเซนเซอร์ SHT15 และการจัดขาเพื่อต่อใช้งาน

2.7.2 ขาสัญญาณสำหรับการสื่อสารข้อมูลของเซนเซอร์ SHT15

- ขาสัญญาณนาฬิกา (SCK)

ทำหน้าที่รับสัญญาณนาฬิกา เพื่อกำหนดจังหวะในการสื่อสารข้อมูล

- ขาสัญญาณรับ/ส่งข้อมูล (DATA)

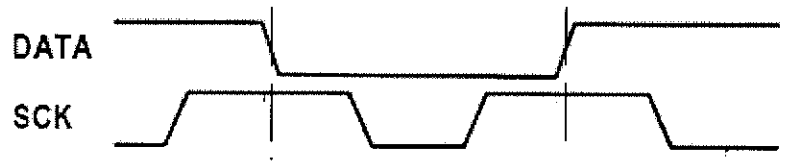
เป็นขาสัญญาณสำหรับรับส่งข้อมูล ในการใช้งานควรต่อความต้านทาน 10k พูลอัพที่ขานี้

2.7.3 รูปแบบการสื่อสารข้อมูลของ SHT15

การส่งคำสั่ง (Sending a Command)

ในสภาวะเริ่มต้นก่อนการส่งข้อมูลคำสั่งจากไมโครคอนโทรลเลอร์ไปยัง SHT-15 จำเป็นต้องสร้างรูปแบบสัญญาณกระตุ้นผ่านขาสัญญาณ SCK และ DATA เพื่อให้ตรงกับเงื่อนไขที่เรียกว่า Transmission start หรือสภาวะเริ่มต้นการส่งสัญญาณ นั่นคือขา DATA ต้องถูกทำให้เป็นลอจิก “0” นาน

อย่างน้อย 1 ไซเคิลของสัญญาณนาฬิกา SCK หลังจากนั้น SHT-15 จะทราบทันทีว่า ข้อมูลต่อจากนี้เป็นคำสั่ง



รูปที่ 2.25 แสดงรูปแบบสัญญาณกระตุ้นผ่านขาสัญญาณ SCK และ DATA (Transmission start)

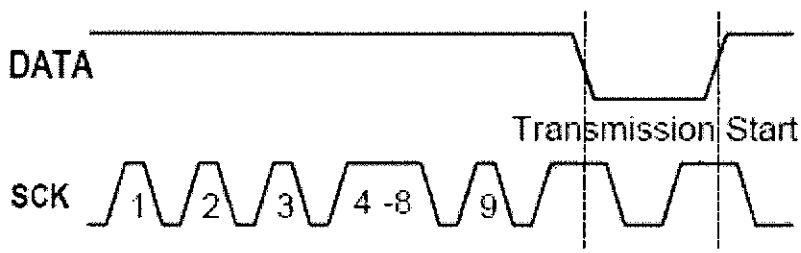
ตารางที่ 2.7 แสดงรายละเอียดคำสั่งและข้อมูลสำหรับควบคุมการทำงานของเซนเซอร์ SHT-15

คำสั่ง	ข้อมูลคำสั่ง
สงวนไว้	0000x
อ่านค่าอุณหภูมิ(Measure Temperature)	000011
อ่านค่าความชื้นสัมพัทธ์(Measure Humidity)	00101
อ่านค่ารีจิสเตอร์กำหนดสถานะ(Read Status Register)	00111
สงวนไว้	0101x ถึง 1110x
รีเซตการทำงาน(Soft reset) ทำให้รีจิสเตอร์กำหนดสถานะกลับไปสู่ค่า Default และต้องใช้เวลาในการทำงานอย่างน้อย 11 มิลลิวินาที จึงสามารถรับคำสั่งถัดไปได้	11110

หลังจากสร้างเงื่อนไข Transmission start แล้ว สามารถส่งคำสั่งไปยัง SHT-15 เพื่อกำหนดการทำงานได้ทันที โดยข้อมูลคำสั่งต่างๆสำหรับการทำงานตามตารางที่กำหนด

2.7.4 รีเซตการเชื่อมต่อ (Connection reset sequence)

เมื่อต้องการเริ่มต้นการเชื่อมต่อระหว่างไมโครคอนโทรลเลอร์กับเซนเซอร์ SHT-15 ต้องสร้างสัญญาณรีเซตขึ้นมาก่อน โดยทำให้ขา DATA มีสถานะลอจิกเป็น "1" นานเท่ากับช่วงเวลาที่ป้อนสัญญาณนาฬิกาที่ขา SCK จำนวน 9 ลูกติดต่อกัน แล้วตามด้วยการสร้างสภาวะเริ่มต้นการส่งสัญญาณ



รูปที่ 2.26 แสดงสัญญาณรีเซตและการสร้างสภาวะเริ่มต้นการส่งสัญญาณ

2.7.5 ขั้นตอนการอ่านอุณหภูมิและความชื้นสัมพัทธ์

การอ่านข้อมูลดิบของอุณหภูมิหรือความชื้นสัมพัทธ์นั้น ทำได้ภายหลังจากที่สร้างสภาวะเริ่มต้นที่เรียกว่า Transmission start แล้วตามด้วยการส่งข้อมูลคำสั่งอ่านอุณหภูมิหรือความชื้นสัมพัทธ์อย่างใดอย่างหนึ่งไปยัง SHT-15 เซนเซอร์ SHT-15 ต้องใช้เวลาในการประมวลผลเพื่อให้ได้ผลลัพธ์ที่ต้องการ ซึ่งใช้เวลามากหรือน้อยขึ้นกับความละเอียดของข้อมูลที่ต้องการแสดงในตาราง

ตารางที่ 2.8 แสดงค่าเวลาที่เซนเซอร์ SHT-15 ต้องใช้ในการประมวลผลข้อมูล

ความละเอียดของข้อมูลที่ประมวลผล	เวลาที่เซนเซอร์ SHT15 ใช้ในการประมวลผล (ผิดพลาด 15%)
14 บิต	210 มิลลิวินาที
12 บิต	55 มิลลิวินาที
8 บิต	11 มิลลิวินาที

2.7.6 การคำนวณค่าอุณหภูมิ

ในการอ่านค่าอุณหภูมิจากเซนเซอร์ SHT-15 ผู้พัฒนาสามารถเลือกความละเอียดในการอ่านได้ในแบบ 14 บิต หรือ 12 บิต โดยที่ความละเอียด 14 บิตเป็นค่าตั้งต้น โดยผู้ที่พัฒนาจำเป็นที่จะต้องอ่านข้อมูลดิบจากเซนเซอร์ SHT-15 เข้ามาก่อน จากนั้นจึงใช้กระบวนการทางคณิตศาสตร์เพื่อให้ได้ค่าอุณหภูมิออกมา โดยสามารถคำนวณได้จากสมการที่กำหนดมาจาก Sensirion ผู้ผลิตเซนเซอร์ SHT-15 ดังนี้

$$Temperature = d1 + (d2 * SO_T) \quad \text{.....1}$$

โดยที่ $Temperature$ คือค่าอุณหภูมิจริง

$d1$ คือค่าคงที่ ขึ้นอยู่กับไฟเลี้ยงที่ป้อนให้กับขา V_{DD} ของ SHT-15

$d2$ คือค่าคงที่ ขึ้นอยู่กับความละเอียดของอุณหภูมิที่ต้องการจาก SHT-15

SO_T คือค่าอุณหภูมิที่อ่านได้จากเซนเซอร์ SHT-15

ตารางที่ 2.9 แสดงการกำหนดค่าคงที่ทางอุณหภูมิตัวที่ 1 และตัวที่ 2 เพื่อคำนวณค่าอุณหภูมิจริงที่วัดได้

ไฟเลี้ยง	ค่าคงที่ทางอุณหภูมิตัวที่ 1(d1)	
	ในหน่วย C	ในหน่วย F
+5V	-40.00	-40.00
+4V	-39.75	-39.50
+3.5V	-39.66	-39.35
+3V	-39.60	-39.28
+2.5V	-39.55	-39.23

ความละเอียด	ค่าคงที่ทางอุณหภูมิตัวที่ 2(d2)	
	ในหน่วย C	ในหน่วย F
14 บิต	0.01	0.018
12 บิต	0.04	0.072

2.7.7 คำนวณค่าความชื้นสัมพัทธ์

สำหรับการอ่านค่าความชื้นสัมพัทธ์จากเซนเซอร์ SHT-15 สามารถเลือกความละเอียดในการอ่านได้ในแบบ 12 บิตหรือ 8 บิต โดยที่ความละเอียด 12 บิตเป็นค่าตั้งต้นหลัก โดยที่ผู้ที่จะพัฒนาจะต้องอ่านข้อมูลดิบจากเซนเซอร์ SHT-15 เข้ามาก่อน จากนั้นจึงใช้กระบวนการทางคณิตศาสตร์เพื่อให้ได้ค่าความชื้นสัมพัทธ์ออกมา โดยสามารถคำนวณได้จากสมการที่กำหนดมาจาก Sensirion ผู้ผลิตเซนเซอร์นี้ดังนี้

$$RH_{true} = (T-25)*(t1+(t2*SO_{RH}))+RH_{linear} \quad \text{.....2}$$

$$RH_{linear} = c1+(c2*SO_{RH})+(c3*(SO_{RH})^2) \quad \text{.....3}$$

โดยที่ RH_{true} คือค่าความชื้นสัมพัทธ์จริง

T คือค่าอุณหภูมิจริงที่คำนวณได้จากสมการที่ 1

$t1$ และ $t2$ คือค่าคงที่โดยขึ้นกับความละเอียดของความชื้นสัมพัทธ์ที่ต้องการจาก SHT-15

$c1, c2, c3$ คือค่าคงที่ขึ้นอยู่กับความละเอียดของความชื้นสัมพัทธ์ที่ต้องการจากเซนเซอร์ SHT-15

SO_{RH} คือค่าความชื้นสัมพัทธ์ดิบที่อ่านได้จาก SHT-15

ตารางที่ 2.10 แสดงการกำหนดค่าคงที่ซึ่งต้องใช้ในการคำนวณค่าความชื้นสัมพัทธ์จริงที่วัดได้

ความละเอียด	ค่าคงที่	
	$t1$	$t2$
12 บิต	0.01	0.00008
8 บิต	0.01	0.00128

ความละเอียด	ค่าคงที่		
	$c1$	$c2$	$c3$
12 บิต	-4	0.648	$-2.8*10^{-6}$
8 บิต	-4	0.0405	$-7.2*10^{-4}$

2.8 MICROCONTROLLER

2.8.1 โครงสร้างและรายละเอียด

ไมโครคอนโทรลเลอร์ที่ใช้ในโครงงานนี้เป็นตระกูล MCS-51 เบอร์ AT89xx (เบอร์ AT89C55) ซึ่งเป็นของบริษัท Atmel Corporation โดยมีคุณสมบัติดังนี้

- เป็นไมโครคอนโทรลเลอร์ที่ใช้ซีพียูขนาด 8 บิต
- ภายในมีหน่วยความจำโปรแกรมเป็นแบบแฟลช (flash memory) ขนาด 20 กิโลไบต์ สามารถลบและเขียนใหม่ได้นับพันครั้ง
- หน่วยความจำข้อมูลพื้นฐานเป็นแบบแรม (RAM) 256 ไบต์
- ขาพอร์ตเป็นแบบสองทิศทาง สามารถใช้งานเป็นทั้งอินพุตและเอาท์พุต
- มีวงจรสื่อสารอนุกรมแบบฟูลดูเพล็กซ์ (Full Duplex)
- ไทเมอร์และเคาน์เตอร์ขนาด 16 บิต จำนวน 3 ตัว
- สามารถรองรับแหล่งกำเนิดอินเตอร์รัปต์ (Interrupt) ได้ 6 ประเภท
- สามารถขยายหน่วยความจำภายนอกเพิ่มเติมได้สูงสุด 64 กิโลไบต์

- มีวงจรกำเนิดสัญญาณนาฬิกาอยู่ภายในชิป

การจัดขาของไมโครคอนโทรลเลอร์ MCS-51

ไมโครคอนโทรลเลอร์ MCS-51 ทุกเบอร์จะมีสถาปัตยกรรมและขาใช้งานพื้นฐานเหมือนกัน โดยมีรายละเอียดดังนี้

ขา Vcc ใช้สำหรับต่อไฟเลี้ยง +5V

ขา GND เป็นขากราวด์ สำหรับต่อกับกราวด์ของระบบ

ขาพอร์ต 0 (P0.0-P0.7) มี 8 ขา แต่ละขาสามารถกำหนดให้เป็นได้ทั้งอินพุตและเอาท์พุต สำหรับใช้งานทั่วไป ถ้าหากต้องการกำหนดให้ขาพอร์ต 0 ขาใดขาหนึ่งเป็นอินพุตสามารถทำได้โดยการเขียนข้อมูล "1" ไปยังแต่ละบิตของพอร์ตที่ต้องการติดต่อด้วย ส่งผลให้ขาพอร์ตนั้นมีสถานะปล่อยลอย (float) จึงมีอินพุตอิมพีแดนซ์สูงสามารถใช้งานเป็นขาพอร์ตอินพุตได้ นอกจากนั้นขาพอร์ตนี้ยังถูกใช้งานในการติดต่อกับขาแอดเดรสไบต์ต่ำของหน่วยความจำภายนอก (A0-A7) และขาข้อมูล (D0-D7) โดยใช้กระบวนการมัลติเพล็กซ์เข้าช่วย เพื่อสลับการทำงานเป็นได้ทั้งขาติดต่อแอดเดรสและขาข้อมูล

ขาพอร์ต 1 (P1.0-P1.7) มี 8 ขา แต่ละขาสามารถกำหนดให้เป็นได้ทั้งอินพุตและเอาท์พุต สำหรับใช้งานทั่วไป ถ้าหากต้องการกำหนดให้ขาพอร์ตใดเป็นอินพุตสามารถทำได้โดยการเขียนข้อมูล "1" ไปยังแต่ละบิตของพอร์ตที่ต้องการติดต่อด้วยเช่นเดียวกันกับขาพอร์ต 0

ขาพอร์ต 2 (P2.0-P2.7) มี 8 ขา แต่ละขาสามารถกำหนดให้เป็นได้ทั้งอินพุตและเอาท์พุต สำหรับใช้งานทั่วไป ถ้าหากต้องการกำหนดให้ขาพอร์ตใดเป็นอินพุตสามารถทำได้โดยการเขียนข้อมูล "1" ไปยังแต่ละบิตของพอร์ตที่ต้องการติดต่อด้วย ส่งผลให้ขาพอร์ตนั้นมีสถานะปล่อยลอย (float) จึงมีอินพุตอิมพีแดนซ์สูงสามารถใช้งานเป็นขาพอร์ตอินพุตได้ นอกจากนั้นขาพอร์ตนี้ยังถูกใช้งานในการติดต่อกับขาแอดเดรสไบต์สูงของหน่วยความจำภายนอก (A8-A15)

ขาพอร์ต 3 (P3.0-P3.7) มี 8 ขา แต่ละขาสามารถกำหนดให้เป็นได้ทั้งอินพุตและเอาท์พุต สำหรับใช้งานทั่วไป ถ้าหากต้องการกำหนดให้ขาพอร์ตใดเป็นอินพุตสามารถทำได้โดยการเขียนข้อมูล "1" ไปยังแต่ละบิตของพอร์ตที่ต้องการติดต่อด้วย ส่งผลให้ขาพอร์ตนั้นมีสถานะปล่อยลอย (float) จึงมีอินพุตอิมพีแดนซ์สูงสามารถใช้งานเป็นขาพอร์ตอินพุตได้ นอกจากนั้นขาพอร์ต 3 ยังเป็นขาที่มีหน้าที่การใช้งานพิเศษ ดังนี้

- *P3.0* ใช้เป็นขาอินพุตสำหรับรับข้อมูลจากการสื่อสารแบบอนุกรม หรือขา RxD
- *P3.1* ใช้เป็นขาอินพุตสำหรับส่งข้อมูลจากการสื่อสารแบบอนุกรม หรือขา TxD
- *P3.2* ใช้เป็นขาอินพุตรับสัญญาณอินเตอร์รัปต์จากภายนอกช่อง 0 หรือขา $\overline{\text{INT0}}$
- *P3.3* ใช้เป็นขาอินพุตรับสัญญาณอินเตอร์รัปต์จากภายนอกช่อง 1 หรือขา $\overline{\text{INT1}}$
- *P3.4* ใช้เป็นขาอินพุตสำหรับรับสัญญาณไทเมอร์จากภายนอกช่อง 0 หรือขา T0
- *P3.5* ใช้เป็นขาอินพุตสำหรับรับสัญญาณไทเมอร์จากภายนอกช่อง 1 หรือขา T1
- *P3.6* ใช้เป็นขาสัญญาณ $\overline{\text{WR}}$ ในกรณีที่ใช้เชื่อมต่อกับหน่วยความจำภายนอก
- *P3.7* ใช้เป็นขาสัญญาณ $\overline{\text{RD}}$ ในกรณีที่ใช้เชื่อมต่อกับหน่วยความจำภายนอก

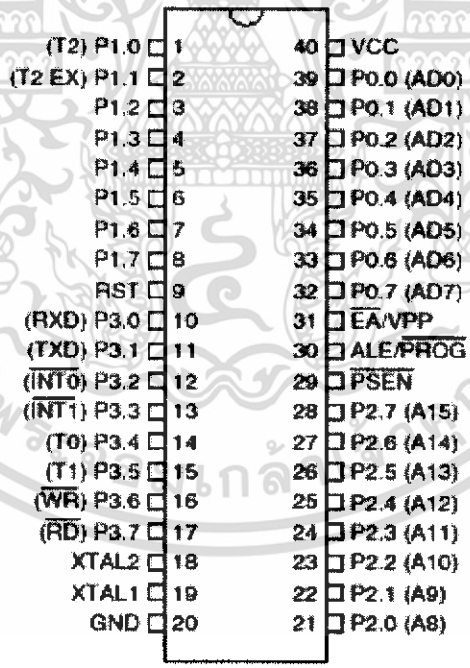
ขารีเซต (Reset) ใช้ในการรีเซตการทำงานของไมโครคอนโทรลเลอร์ โดยในการป้อนสัญญาณเพื่อรีเซตสถานะที่ขานี้ต้องอยู่ในระดับรีเซตอย่างน้อย 2 แมชชีนไซเคิล โดยที่วงจรกำเนิดสัญญาณนาฬิกายังคงทำงานต่อเนื่องไปอย่างเป็นปกติ

ขา $\overline{\text{ALE}}$ */PROG (Address Latch Enable/Program pulse input)* เป็นขาที่ใช้ในการควบคุมการแลตช์ของขาพอร์ต 0 เมื่อมีการใช้งานหน่วยความจำภายนอก

ขา $\overline{\text{PSEN}}$ *(Program Store Enable)* ขานี้ใช้ในการส่งสัญญาณเพื่อร้องขอติดต่อกับหน่วยความจำโปรแกรมภายนอก เมื่อไมโครคอนโทรลเลอร์ต้องการอ่านข้อมูลจากหน่วยความจำโปรแกรมภายนอก ตัวไมโครคอนโทรลเลอร์จะส่งสัญญาณออกมาที่ขานี้ 2 ครั้งในแต่ละแมชชีนไซเคิล แต่ถ้าหากติดต่อกับหน่วยความจำข้อมูลภายนอกขานี้จะไม่มีการส่งสัญญาณใดๆออกมา

ขา $\overline{\text{EA}}$*/Vpp (External Access enable/Programming voltage input)* ใช้สำหรับเลือกการติดต่อหน่วยความจำโปรแกรมจากภายนอกหรือภายในไมโครคอนโทรลเลอร์ ถ้าหากขานี้เป็น "0" เป็นการเลือกให้ไมโครคอนโทรลเลอร์ติดต่อกับหน่วยความจำโปรแกรมภายนอก แต่ถ้าหากขานี้เป็น "1" เป็นการเลือกให้ไมโครคอนโทรลเลอร์ติดต่อกับหน่วยความจำภายในตัวไมโครคอนโทรลเลอร์

ขา *XTAL1 และ XTAL2* เป็นขาสำหรับต่อคริสตอลเพื่อสร้างสัญญาณนาฬิกาในการกำหนดจังหวะการทำงานของไมโครคอนโทรลเลอร์



รูปที่ 2.27 แสดงขาของ MCS-51

2.8.2 การใช้พอร์ตอนุกรม

ในโครงงานนี้ได้เลือกใช้พอร์ตอนุกรมโหมด 1 (เลือกโหมดการทำงานจากรีจิสเตอร์ SCON) ซึ่งเป็นการสื่อสารอนุกรม 10 บิต เป็นข้อมูล 8 บิต บิตเริ่มต้น (Start bit) 1 บิต และบิตหยุด (Stop bit) 1 บิต โดยสามารถเลือกอัตราบอดได้ ซึ่งคำนวณจาก

$$B=\frac{2^{SMOD}\times Oscillator(freq)}{32\times12\times\left[256-(TH1)\right]}$$

รายละเอียดของรีจิสเตอร์ SCON

บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SM0-SM1 (Serial port mode bit 0 -1) : ใช้ในการเลือกโหมดการทำงานของพอร์ตอนุกรมภายในไมโครคอนโทรลเลอร์ (ที่ใช้เป็นโหมด 1 เซ็ตค่า SM0=0 , SM1=1)

SM2 : ใช้ในการเอ็นเอเบิลการสื่อสารในแบบมัลติโปรเซสเซอร์ (multiprocessor) ในการทำงานโหมด 2 และ 3 ของพอร์ตอนุกรมในไมโครคอนโทรลเลอร์ MCS-51 ถ้าบิตนี้เป็น "1" บิต RI จะไม่แอกตีฟถ้าบิตที่ 9 ที่รับเข้ามาเป็น "0" (ข้อมูลบิตที่ 9 เก็บไว้ที่บิต RB8) ในการทำงานโหมด 1 ถ้าบิตนี้เซต บิต RI จะไม่แอกตีฟถ้ายังไม่ได้รับบิตหยุด ส่วนในโหมด 0 บิตนี้จะไม่มีการใช้งาน

REN (Enable serial reception) : ใช้ในการเอ็นเอเบิลการรับข้อมูลของพอร์ตอนุกรม ทำการเซตและเคลียร์ด้วยกระบวนการทางซอฟต์แวร์ ถ้าต้องการให้มีการรับข้อมูลต้องเซตบิตนี้ให้เป็น "1"

TB8 : ใช้สำหรับเก็บข้อมูลบิตที่ 9 ที่ต้องการส่งออกไปในการทำงานโหมด 2 และ 3 ของพอร์ตอนุกรมในไมโครคอนโทรลเลอร์ MCS-51 ทำการเซตและเคลียร์ด้วยกระบวนการทางซอฟต์แวร์

RB8 : ใช้สำหรับเก็บข้อมูลบิตที่ 9 ที่ต้องการส่งออกไปในการทำงานโหมด 2 และ 3 ของพอร์ตอนุกรมในไมโครคอนโทรลเลอร์ MCS-51 แต่ถ้าหากพอร์ตอนุกรมทำงานอยู่ในโหมด 1 และบิต SM2 เป็น "0" ข้อมูลที่บิต RB8 คือข้อมูลของบิตหยุด (Stop bit) สำหรับในการทำงานโหมด 0 บิตนี้จะไม่ใช้งาน บิต RB8 นี้สามารถทำการเซตและเคลียร์ด้วยกระบวนการทางซอฟต์แวร์

TI (Transmit Interrupt flag) : ใช้ในการแสดงการเกิดอินเตอร์รัปต์เมื่อมีการส่งข้อมูลออกจากพอร์ตอนุกรมของไมโครคอนโทรลเลอร์ MCS-51 สามารถเซตได้ด้วยกระบวนการทางฮาร์ดแวร์ เมื่อทำการส่งข้อมูลบิตที่ 8 ไปเรียบร้อยแล้วในการทำงานโหมด 0 ส่วนในการทำงานโหมดอื่นบิตนี้จะเซตเมื่อมีการเริ่มต้นส่งบิตหยุดออกไป การเคลียร์บิตนี้ต้องใช้กระบวนการทางซอฟต์แวร์เท่านั้น

RI (Receive Interrupt flag) : ใช้ในการแสดงการเกิดอินเตอร์รัปต์เมื่อมีการรับข้อมูลเข้าสู่พอร์ตอนุกรมของไมโครคอนโทรลเลอร์ MCS-51 สามารถเซตได้ด้วยกระบวนการทางฮาร์ดแวร์ เมื่อทำการรับข้อมูลบิตที่ 8 เรียบร้อยแล้วในการทำงานโหมด 0 ส่วนในการทำงานโหมดอื่นบิตนี้จะเซตเมื่อสามารถรับบิตหยุดของข้อมูลอนุกรมไปได้ครึ่งทางแล้ว ยกเว้นในกรณีที่บิต SM2 มีการเซต บิตนี้จะเซตได้ก็ต่อเมื่อการรับบิตหยุดหรือบิตที่ 9 เกิดขึ้นอย่างสมบูรณ์แล้ว การเคลียร์บิตนี้ต้องใช้กระบวนการทางซอฟต์แวร์เท่านั้น

2.8.3 การใช้งานเป็นตัวตั้งเวลา (Timer)

ทำงานโดยในทุกๆแมชชีนไซเคิล ค่าของรีจิสเตอร์จะถูกเพิ่มขึ้น ก็คือการนับค่าแมชชีนไซเคิลนั่นเอง โดย 1 แมชชีนไซเคิลจะเท่ากับ 1 ส่วน 12 ของสัญญาณนาฬิกา ซึ่งรีจิสเตอร์ที่ควบคุมการทำงาน

ของไทม์เมอร์ก็คือ TCON(Timer/Counter Control Register)และเลือกโหมดของไทม์เมอร์ด้วยรีจิสเตอร์ TMOD

2.8.4 การใช้งานเป็นเคาท์เตอร์ (Counter)

ค่าของรีจิสเตอร์จะถูกเพิ่มขึ้นเมื่อมีการเปลี่ยนแปลงของระดับสัญญาณลอจิก(ขาลง 1 เป็น 0 หรือขาขึ้น 0 เป็น 1)ที่ขาอินพุตของวงจร โดยความถี่สูงสุดที่เคาท์เตอร์ของไมโครคอนโทรลเลอร์จะนับได้คือความถี่ของสัญญาณนาฬิกาหารด้วย 24 (จากกระบวนการตรวจสอบการเปลี่ยนแปลงของลอจิกต้องใช้ 2 แมชชีนไซเคิล)ซึ่งควบคุมการทำงานโดยใช้รีจิสเตอร์ TCON(Timer/Counter Control Register)และเลือกโหมดโดย TMOD เหมือนตัวของไทม์เมอร์

รายละเอียดการ Set รีจิสเตอร์ TCON

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1 : แฟลกซ์แสดงการเกิดโอเวอร์โฟลว์ของไทม์เมอร์1 (จะเซตเมื่อเกิดโอเวอร์โฟลว์)

TR1 : ควบคุมการนับของไทม์เมอร์ 1 โดยถ้าเป็น “1” ทำงานและถ้าเป็น “0” จะหยุดทำงาน

TF0 : แฟลกซ์แสดงการเกิดโอเวอร์โฟลว์ของไทม์เมอร์ 0

TR0 : ควบคุมการนับของไทม์เมอร์ 0 โดยถ้าเป็น “1” ทำงานและถ้าเป็น “0” จะหยุดทำงาน

IE1 : แสดงการเกิดอินเตอร์รัปต์ที่ $\overline{\text{INT1}}$

IT1 : ใช้เลือกประเภทการอินเตอร์รัปต์ของ $\overline{\text{INT1}}$ ว่าจะนับที่ขอบขาขึ้น (เซตเป็น “0”) หรือขอบขาลง (เซตเป็น “1”)

IE0 : แสดงการเกิดอินเตอร์รัปต์ที่ $\overline{\text{INT0}}$

IT0 : ใช้เลือกประเภทการอินเตอร์รัปต์ของ $\overline{\text{INT0}}$ ว่าจะนับที่ขอบขาขึ้น (เซตเป็น “0”) หรือขอบขาลง (เซตเป็น “1”)

รายละเอียดการ Set รีจิสเตอร์ TMOD

Timer1				Timer0			
GATE	C/T	M1	M0	GATE	C/T	M1	M0

GATE : เลือกการควบคุมจากซอฟต์แวร์ (“0”) หรือจากฮาร์ดแวร์ (“1”)

C/T : เลือกว่าจะใช้เป็นไทม์เมอร์ (“0”) หรือจะใช้เป็นเคาท์เตอร์ (“1”)

M1 : ใช้เลือกโหมดการทำงาน (ที่ใช้เป็นโหมด 1 เซตค่า M1=0 , M0=0)

M0 : ใช้เลือกโหมดการทำงาน (ที่ใช้เป็นโหมด 1 เซตค่า M1=0 , M0=0)

2.9 แอลซีดี (LCD)

ในโครงงานนี้จะใช้แอลซีดีโมดูลเบอร์ BC1602HGREH ของบริษัท BOLYMIN ซึ่งจะแสดงผลได้ 2 บรรทัด บรรทัดละ 16 ตัวอักษร มีขาให้ใช้งานทั้งหมด 14 ขา ซึ่งแต่ละขามีรายละเอียดดังนี้

ขา 1 V_{SS} : ต่อกราวด์

ขา 2 V_{DD} : ต่อไฟเลี้ยง +5 V

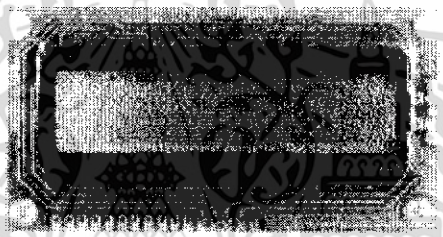
ขา 3 : เป็นขาปรับความสว่างของจอ LCD

ขา 4 RS : เป็นขาอินพุตใช้ในการแยกชนิดของข้อมูลที่ทำการประมวลผลในขณะนั้นว่าเป็นคำสั่งหรือข้อมูล ถ้าเป็น "0" แสดงว่าเป็นคำสั่ง ถ้าเป็น "1" แสดงว่าเป็นข้อมูล

ขา 5 R/W : เป็นขาที่ใช้เลือกว่าจะเขียนหรืออ่านข้อมูลจากโมดูลแอลซีดี ถ้าเป็น "0" จะเป็นการเขียนข้อมูล ถ้าเป็น "1" จะเป็นการอ่านข้อมูล

ขา 6 E : เป็นขาสำหรับรับสัญญาณพัลส์เอ็นเอเบิลโมดูลแอลซีดีให้ทำงาน

ขา 7-14 D0-D7 : เป็นขาที่ใช้เป็นทางผ่านของข้อมูลระหว่างแอลซีดีกับอุปกรณ์ภายนอกขนาด 8 บิต



รูปที่ 2.28 แสดงภาพแอลซีดีโมดูลที่นำมาใช้ในโครงงานนี้

ตารางที่ 2.11 แสดงคำสั่งที่ใช้ควบคุมแอลซีดี

คำสั่ง (HEX)	การทำงาน
01	เคลียร์หน่วยแสดงผล
02	ให้เคอร์เซอร์กลับสู่ตำแหน่งซ้ายสุด
04	แสดงผลโดยเลื่อนเคอร์เซอร์ไปทางซ้าย
05	เลื่อนไปทางขวา
06	แสดงผลโดยเลื่อนเคอร์เซอร์ไปทางขวา
07	เลื่อนไปทางซ้าย
08	ปิดการแสดงผลและไม่แสดงเคอร์เซอร์
0A	ปิดการแสดงผลแต่แสดงเคอร์เซอร์
0C	แสดงผลแต่ไม่แสดงเคอร์เซอร์
0E	แสดงผลและแสดงเคอร์เซอร์
0F	แสดงผลและแสดงเคอร์เซอร์กระพริบ

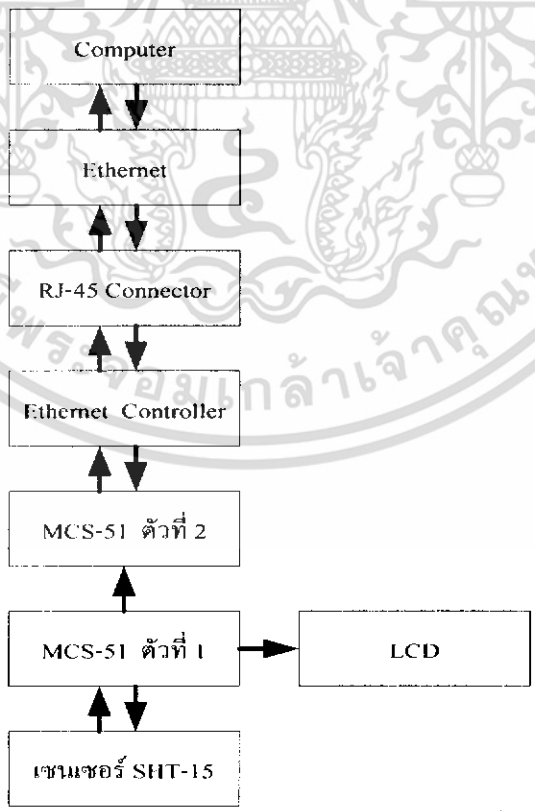
คำสั่ง (HEX)	การทำงาน
10	เลื่อนเคอร์เซอร์ไปทางซ้าย
14	เลื่อนเคอร์เซอร์ไปทางขวา
18	เลื่อนตัวอักษรตัวใหม่ไปทางซ้าย
1C	เลื่อนตัวอักษรตัวใหม่ไปทางขวา
80	ตำแหน่งเริ่มต้นบรรทัดที่ 1
C0	ตำแหน่งเริ่มต้นบรรทัดที่ 2
38	เป็นแบบ 2 บรรทัด ขนาดตัวอักษร 5 x 7 จุด

บทที่ 3
การคำนวณและการสร้างวงจร

หลังจากที่ทราบถึงทฤษฎีหรือหลักการในบทที่ 2 แล้ว เราก็จะมาทำการคำนวณและทำการสร้างวงจร ในส่วนของฮาร์ดแวร์ของระบบควบคุมแบบฝังตัว (Embedded system) ประกอบไปด้วยวงจรควบคุมการรับ-ส่งข้อมูลผ่านอีเทอร์เนต ไมโครคอนโทรลเลอร์ (AT89C55) รวมทั้งเซนเซอร์วัดความชื้นสัมพัทธ์และอุณหภูมิ (SHT-15) รายละเอียดการออกแบบฮาร์ดแวร์ที่จะกล่าวถึงในที่นี้จึงประกอบด้วยการเชื่อมต่อไมโครคอนโทรลเลอร์ตัวที่ 1 เข้ากับเซนเซอร์ SHT-15 พร้อมทั้งแสดงผลที่จอแอลซีดี และการเชื่อมต่อไมโครคอนโทรลเลอร์ตัวที่ 2 เข้ากับระบบเครือข่าย โดยอาศัยอุปกรณ์ควบคุมการเชื่อมต่อระบบเครือข่าย (Ethernet Controller) รวมทั้งอธิบายถึงการควบคุมหน่วยการรับเข้าออกข้อมูล (I/O Port) และกระบวนการรับ-ส่งข้อมูลของฮาร์ดแวร์

3.1 ส่วนประกอบฮาร์ดแวร์ของระบบ

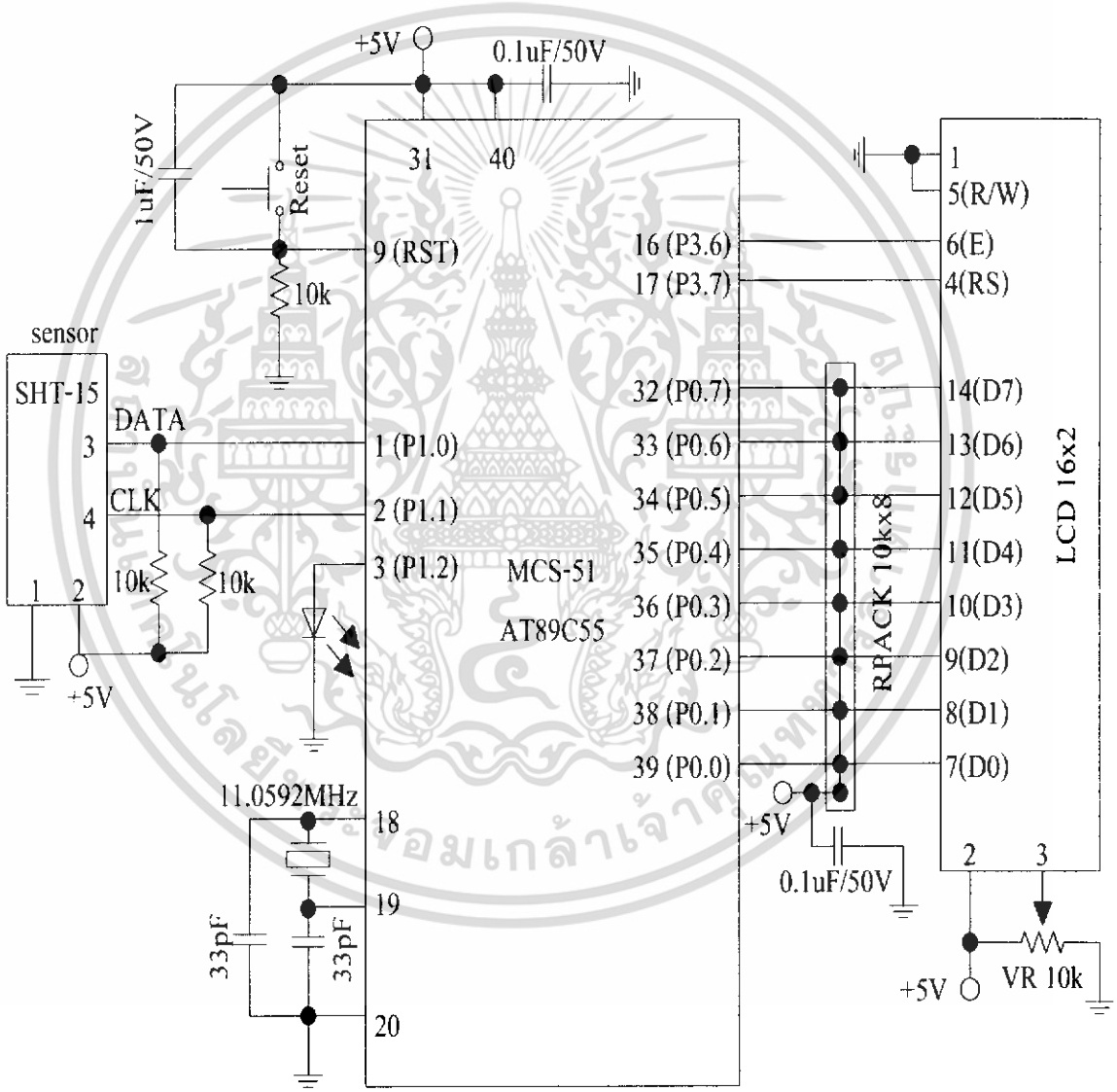
ส่วนประกอบหลักของฮาร์ดแวร์ทั้งหมดประกอบด้วย วงจรเชื่อมต่อกับเซนเซอร์ SHT-15 และจอแอลซีดีซึ่งควบคุมโดยไมโครคอนโทรลเลอร์ตัวที่ 1 รวมทั้งวงจรเชื่อมต่อกับระบบเครือข่ายโดยในส่วนนี้จะเชื่อมต่อกับอีเทอร์เนตและไมโครคอนโทรลเลอร์ตัวที่ 2 โดยไมโครคอนโทรลเลอร์ตัวนี้จะไปควบคุม I/O Port ของ Ethernet Controller ซึ่งข้อมูลจากเซนเซอร์จะถูกส่งจากไมโครคอนโทรลเลอร์ตัวที่ 1 ไปยังไมโครคอนโทรลเลอร์ตัวที่ 2 ผ่านพอร์ตอนุกรม โครงสร้างโดยรวมของฮาร์ดแวร์แสดงดังรูปที่ 3.1



รูปที่ 3.1 แสดงส่วนประกอบหลักของฮาร์ดแวร์ทั้งหมด

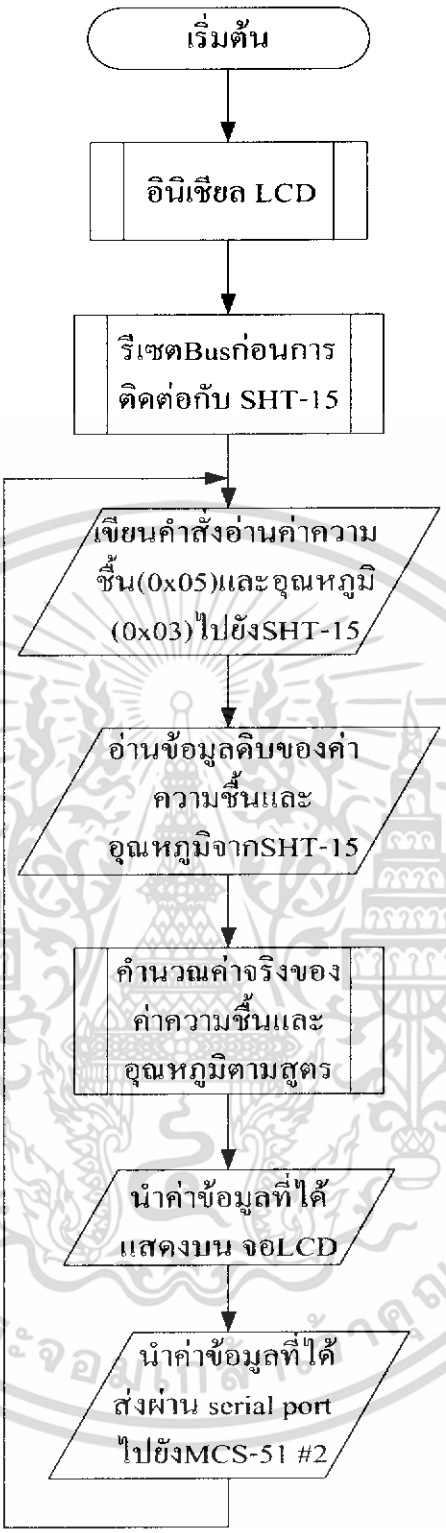
3.2 ส่วนเชื่อมต่อเซนเซอร์วัดความชื้นและอุณหภูมิ SHT-15 และแสดงผลที่จอ LCD

การเชื่อมต่อไมโครคอนโทรลเลอร์กับเซนเซอร์วัดความชื้นสัมพัทธ์และอุณหภูมิ SHT-15 จะเป็นการรับส่งข้อมูลกันแบบ Serial Interface (Bidirectional 2-Wire) โดยมีสัญญาณนาฬิกาเพื่อให้มัน Synchronous กัน ซึ่งการรับส่งข้อมูลกันของไมโครคอนโทรลเลอร์และเซนเซอร์เพื่อไปแสดงผลที่จอแอลซีดีจะเป็นไปตาม Flow chart ที่แสดงในรูปที่ 3.3 และหลังจากนั้นก็ส่งข้อมูลดังกล่าวไปยังไมโครคอนโทรลเลอร์อีกตัวหนึ่งผ่านพอร์ตอนุกรมโหมด 1 ด้วยอัตราบอด 19.2 kbps ดัง Flow chart ในรูปที่ 3.4 (โดยค่าข้อมูลที่รับมาจากเซนเซอร์จะเป็นข้อมูลดิบซึ่งจะต้องนำไปหาค่าความชื้นสัมพัทธ์และอุณหภูมิจริงๆจากการใช้สูตรดังที่กล่าวไปแล้วในบทที่ 2)



รูปที่ 3.2 แสดงวงจรที่ใช้เชื่อมต่อไมโครคอนโทรลเลอร์และเซนเซอร์ SHT-15

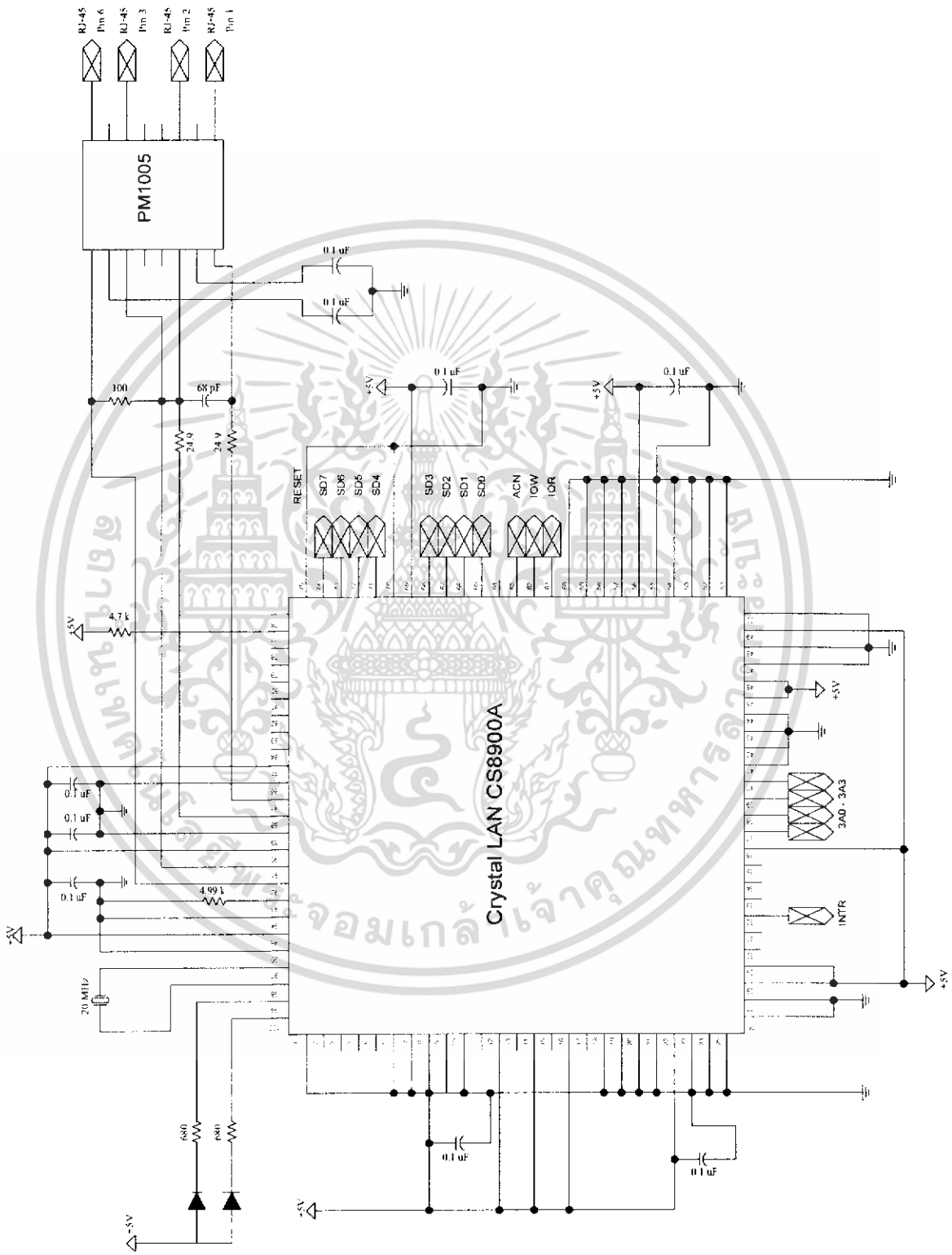
รูปที่ 3.3 แสดง Flow Chart ในการรับส่งค่าข้อมูลความชื้นและอุณหภูมิระหว่างไมโครคอนโทรลเลอร์กับเซนเซอร์ SHT-15 และแสดงผลที่จอ LCD



รูปที่ 3.4 แสดง Flow Chart ในการรับส่งค่าข้อมูลความชื้นและอุณหภูมิระหว่างไมโครคอนโทรลเลอร์กับเซนเซอร์ SHT-15 และแสดงผลที่จอ LCD พร้อมทั้งส่งข้อมูลผ่านพอร์ตอนุกรม

3.3 ส่วนเชื่อมต่อระบบเครือข่าย

จะใช้อุปกรณ์ควบคุมการเชื่อมต่อระบบเครือข่าย (Ethernet Controller) โดยภายในวงจรจะประกอบไปด้วยส่วนประกอบหลัก คือ ชิป (Chip)ควบคุมอีเทอร์เนตในที่นี้จะทำการเลือกใช้ CS8900A-CQ , Transformer (PM1005) และ RJ-45 Connector ดังรูปที่ 3.5



รูปที่ 3.5 แสดงวงจรส่วนเชื่อมต่อระบบเครือข่าย

วงจรควบคุมการเชื่อมต่อระบบเครือข่ายจะมี PIN OUT 18 ขา ดังรูปที่ 3.6

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	$\overline{IOR}$	$\overline{IOW}$

PIN 10	PIN 11	PIN 12	PIN 13	PIN 14	PIN 15	PIN 16	PIN 17	PIN 18
$\overline{AEN}$	SD0	SD1	SD2	SD3	SD4	SD5	SD6	SD7

รูปที่ 3.6 แสดง PIN OUT ของวงจรควบคุมการเชื่อมต่อระบบเครือข่าย

VCC : แหล่งจ่ายไฟตรง +5 V

GND : กราวด์อ้างอิง 0 V

INTR : ใช้ในโหมดอินเตอร์รัปต์ (ซึ่งไม่ได้ใช้)

SA0 – SA3 : Address Bus เชื่อมต่อกับ MCS-51

$\overline{IOR}$: I/O Port Read (Active low)

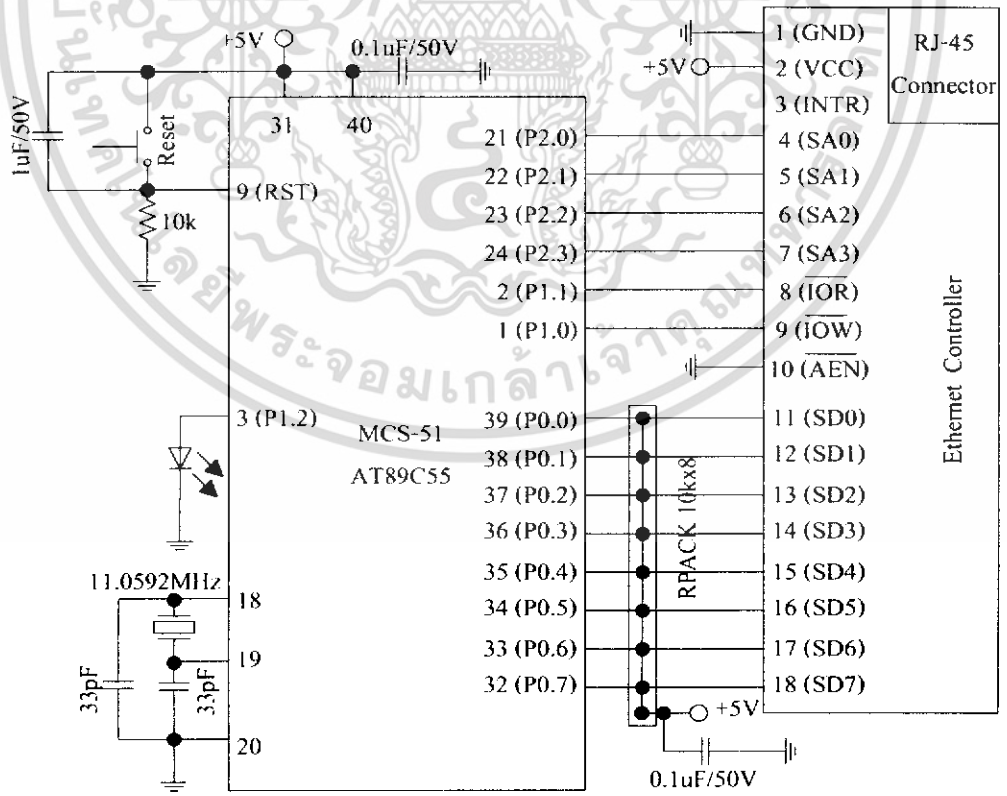
$\overline{IOW}$: I/O Port Write (Active low)

$\overline{AEN}$: Chip Select (Active low)

SD0 – SD7 : Data Bus เชื่อมต่อกับ MCS-51

จาก LED ในวงจรจะสว่างด้วยการจ่ายไฟแล้วเสียบสาย RJ-45 ที่ต่อในวง LAN เมื่อ

- Link LED (Green) จะกระพริบเมื่อมี Datagram ส่งออกจาก Ethernet Controller
- LAN LED (Yellow) จะกระพริบเมื่อมี Datagram เข้ามายัง Ethernet Controller



รูปที่ 3.7 แสดงวงจรที่เชื่อมต่อกันระหว่างไมโครคอนโทรลเลอร์กับ Ethernet Controller

3.4 การเข้าถึงหน่วยความจำของระบบ

ในอุปกรณ์ควบคุมการเชื่อมต่อระบบเครือข่ายจะมีชิป (Chip) ประมวลผลที่สำคัญคือ CS8900A-CQ โดยชิปตัวนี้มีรูปแบบการทำงาน 3 โหมด คือ Memory Mode , I/O Mode และ DMA Mode แต่ในที่นี้จะใช้เพียง I/O Mode เพียงอย่างเดียว ใน I/O Mode จะประกอบด้วย I/O พอร์ตอยู่ 8 พอร์ต แต่ละพอร์ตจะมีความยาว 16 บิต แต่เนื่องจากไมโครคอนโทรลเลอร์ทำงานได้ครั้งละ 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*

ใช้ในการอินเตอร์รัปต์

- *PacketPage Pointer*

ใช้ในการระบุรีจีสเตอร์ภายในของ CS8900A-CQ สามารถหาดูได้จาก Datasheet

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

ใช้ในการอ่านหรือเขียนรีจีสเตอร์ภายใน ก็จะถูกระบุโดย PacketPage Pointer

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

ตารางที่ 3.1 แสดง I/O Mode Mapping

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)

ตัวอย่างการเข้าถึงรีจีสเตอร์ภายในของ I/O Mode

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

1. จะต้องสั่งให้เขียน (Write) ค่า 0x24 ซึ่งเป็น Address 8 บิตล่าง (Least Significant 8 bit) ของ Receiver Event (Rx Event) ไปยัง PacketPage Pointer ที่มี Address อยู่ที่ 0x0A เพราะฉะนั้น Address Bus = 0x0A และ Data Bus = 0x24

2. เขียน (Write) ค่า 0x01 ซึ่งเป็น Address 8 บิตบน (Most Significant 8 bit) ของ Receiver Event (Rx Event) ไปยัง PacketPage Pointer + 1 นั่นคือ Address Bus = 0x0A + 1 = 0x0B และ Data Bus = 0x01

3. เมื่อระบุรีจิสเตอร์แล้วก็จะสามารถอ่าน (Read) หรือ เขียน (Write) รีจิสเตอร์ได้โดยใช้ PacketPage Data พอร์ต 0 ซึ่งทำได้โดย

- Read / Write ไปยัง PacketPage Data ที่มี Address อยู่ที่ 0x0C จะได้ Least Significant 8 บิต นั่นคือ Address Bus = 0x0C และ Data Bus = Data Least Significant 8 บิต ที่ต้องการจะ Read หรือ Write

- Read / Write ไปยัง PacketPage Data จะได้ Most Significant 8 บิต นั่นคือ Address Bus = 0x0C + 1 = 0x0D และ Data Bus = Data Most Significant 8 บิต ที่ต้องการจะ Read หรือ Write

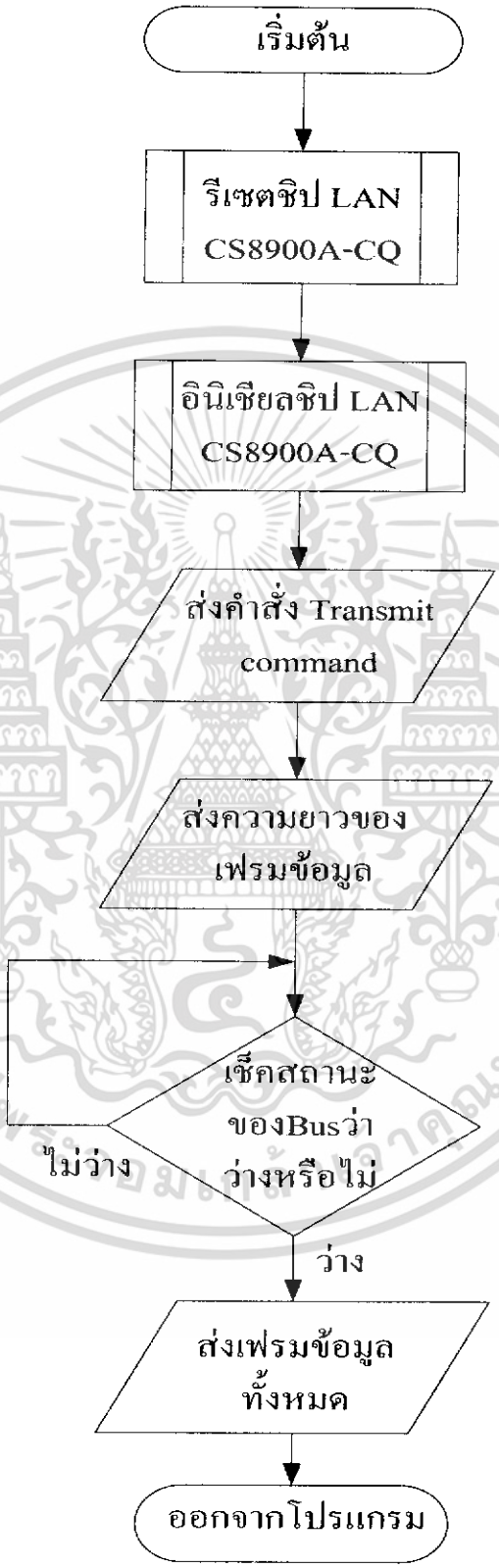
หมายเหตุ 0x... : หมายถึงเลขฐานสิบหก เช่น 0x12 ก็คือ เลข 12 ของฐานสิบหกซึ่งเท่ากับ 18 ในฐานสิบ

3.5 กระบวนการส่งและรับข้อมูลของ Ethernet Controller

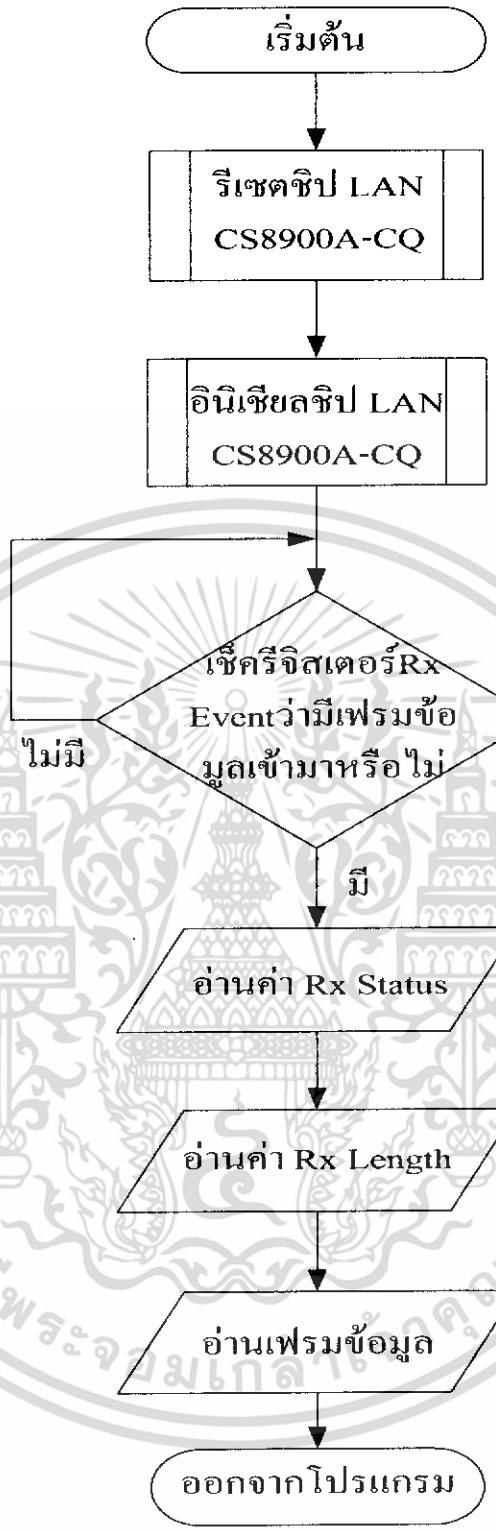
ในกระบวนการส่งข้อมูลใน Ethernet Controller เริ่มต้นด้วยการรีเซตเพื่อลบค่าเก่าออกจากอุปกรณ์ควบคุมการเชื่อมต่อระบบเครือข่ายก่อน จากนั้นทำการตั้งค่าเริ่มต้นของอุปกรณ์ควบคุมการเชื่อมต่อระบบเครือข่าย โดยใช้รีจิสเตอร์ Receiver Control (RxCTL) , Line Control (LineCTL) และค่า MAC Address ต่อมาตามด้วยการ เขียน (Write) คำสั่งที่ต้องการจะส่ง Transmit Command (TxCMD) และ Write ความยาวของข้อมูล Transmit Length (TxLength) จากนั้นตรวจดูว่า Bus ที่ต้องการส่งว่างหรือไม่ โดยตรวจดูจากรีจิสเตอร์ Bus Status (BusST) เมื่อ Bus ว่างก็จะสามารถส่งข้อมูลโดยเข้า I/O พอร์ต คือ Transmit Data (พอร์ต 0) ที่ Address 0x00 ดังใน Flow Chart ในการส่งข้อมูลของ Ethernet Controller ดังรูปที่ 3.8

ส่วนในกระบวนการรับข้อมูลใน Ethernet Controller เริ่มต้นทำการรีเซตและตั้งค่าเริ่มต้นของอุปกรณ์ควบคุมการเชื่อมต่อระบบเครือข่ายและใช้รีจีสเตอร์ Receiver Event (RxEvent) รอรับ frame เมื่อมี frame เข้ามาให้ทำการอ่านค่า Receiver Status (RxStatus)และค่า Receiver Length (RxLength) จาก Receive Data พอร์ต 0 แต่มีข้อควรระวังคือ RxStatus และ RxLength จะต้องอ่านจาก Most Significant bit ก่อนแล้วจึงค่อยอ่านจาก Least Significant bit คือ เซต Address = 0x01 แล้วจึงเซต Address = 0x00 จากนั้นก็ทำการอ่านค่าตามปกติจาก I/O Receive Data พอร์ต 0 คือ อ่าน Address =

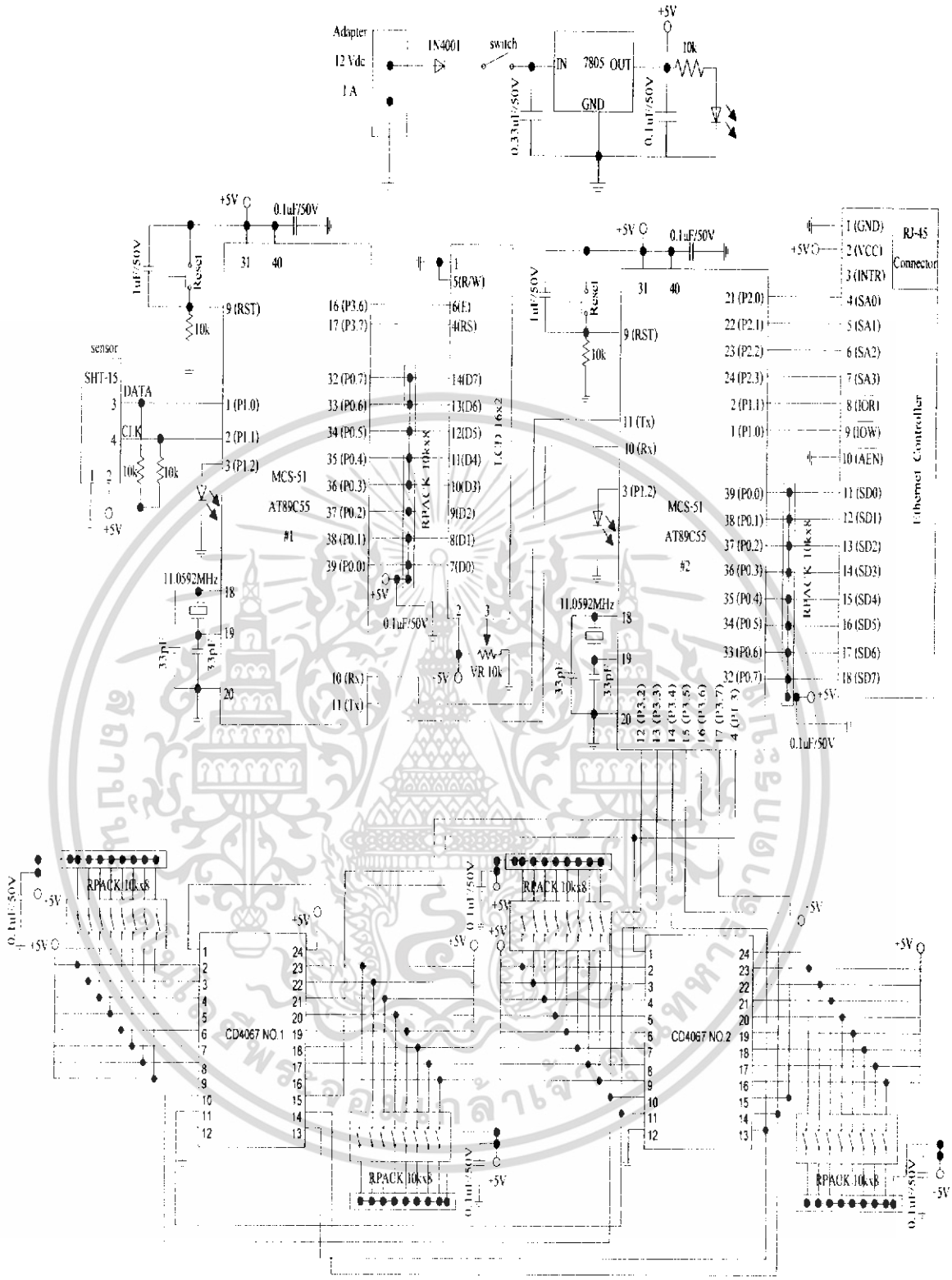
0x00 แล้วจึงอ่านค่า 0x01 วนไปเรื่อยจนข้อมูลที่ส่งมาครบหมด ดังใน Flow Chart ในการรับข้อมูลของ Ethernet Controller ดังรูปที่ 3.9



รูปที่ 3.8 แสดง Flow Chart ในการส่งข้อมูลของ Ethernet Controller



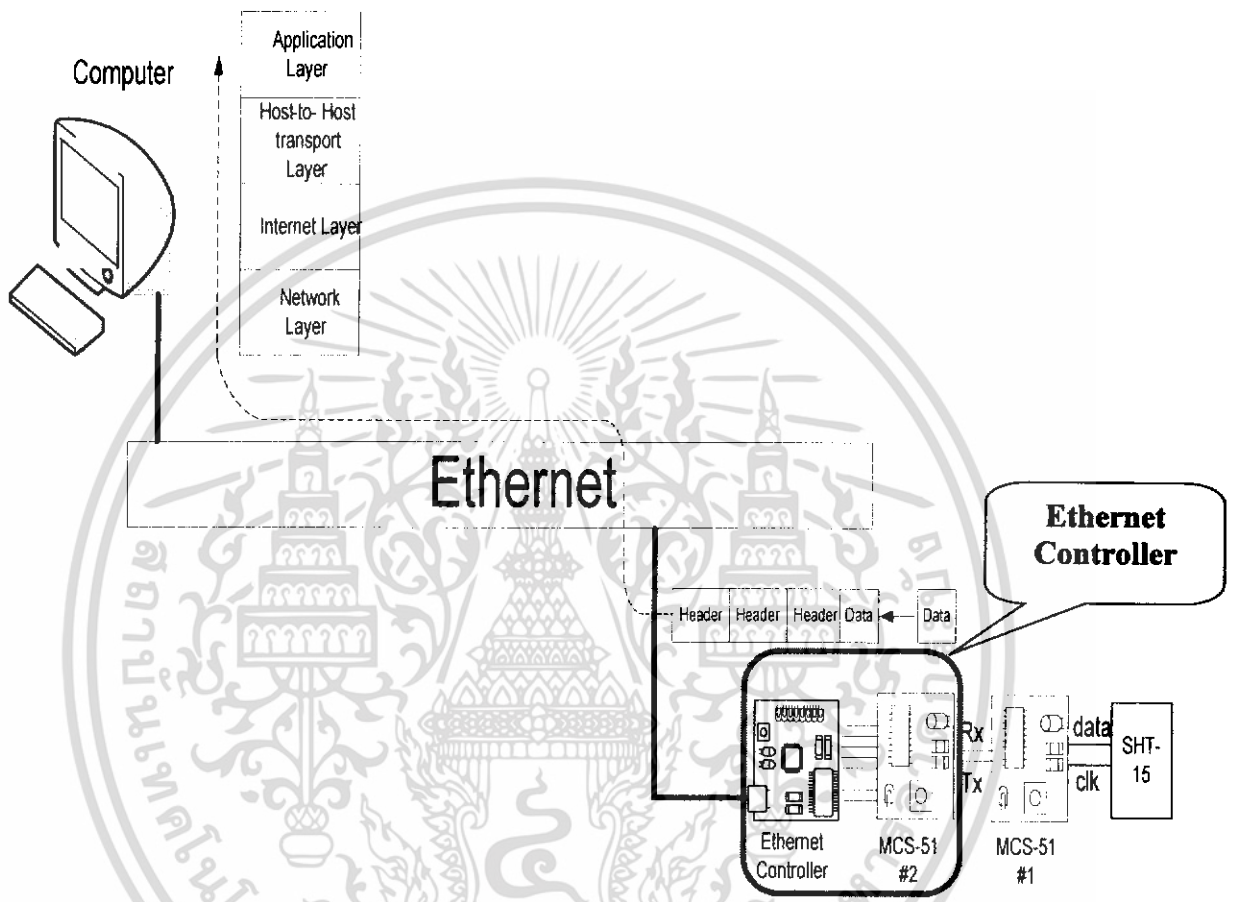
รูปที่ 3.9 แสดง Flow Chart ในการรับข้อมูลของ Ethernet Controller



รูปที่ 3.10 แสดงรูปวงจรรวมทั้งหมด

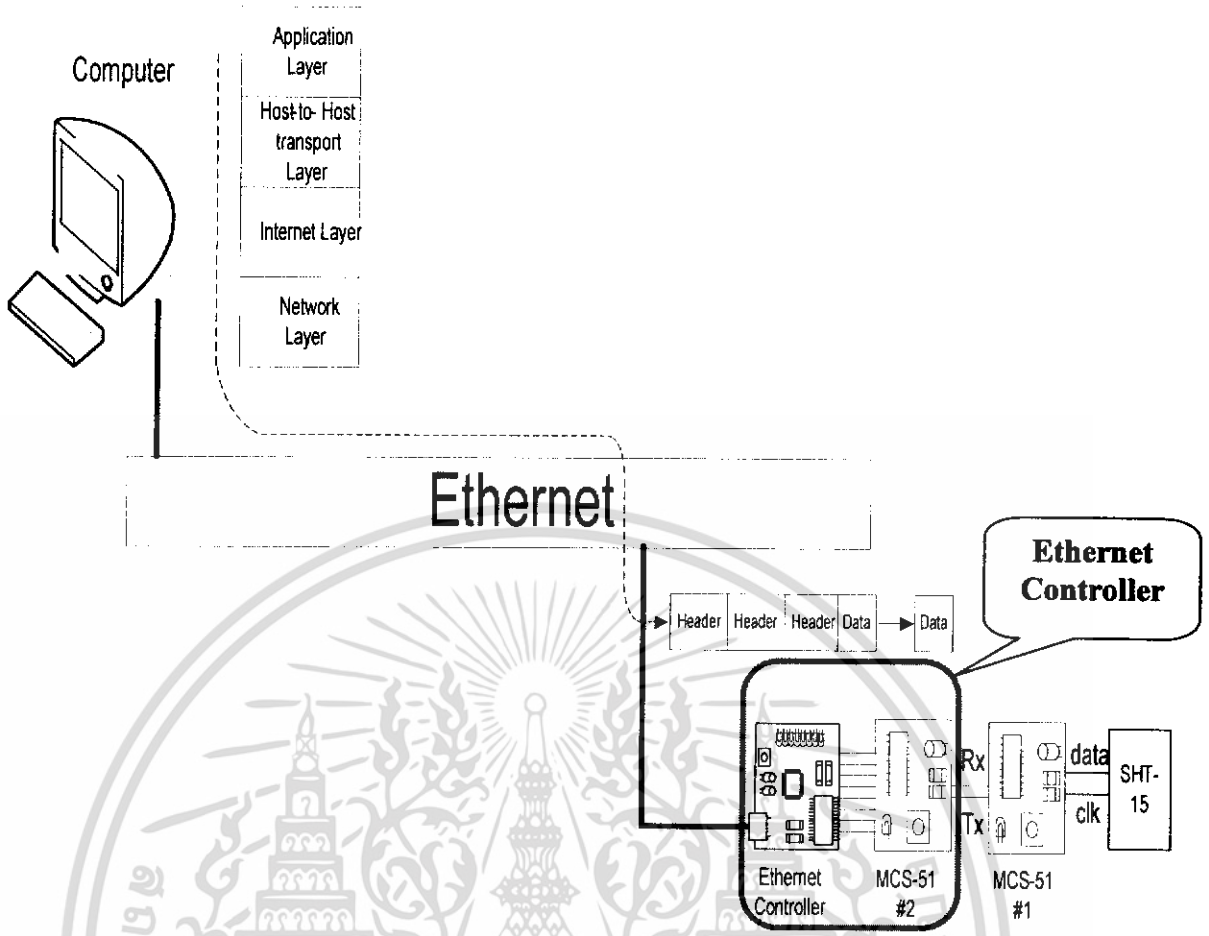
3.6 กระบวนการส่งและรับข้อมูลของระบบ

ในการส่งข้อมูลของฮาร์ดแวร์นั้น MCS-51 จะควบคุมการส่งข้อมูลให้กับอุปกรณ์ควบคุมการเชื่อมต่อระบบเครือข่าย แล้วส่งเฟรมข้อมูลที่ประกอบด้วย Header และ Data ส่งให้กับอุปกรณ์ควบคุมการเชื่อมต่อระบบเครือข่าย ต่อจากนั้นจะส่งข้อมูลให้กับเครื่องคอมพิวเตอร์เพื่อที่จะแสดงผล ดังรูปที่ 3.11



รูปที่ 3.11 แสดงกระบวนการส่งข้อมูลของระบบ

ส่วนในการรับข้อมูล เครื่องคอมพิวเตอร์จะทำการส่งเฟรมข้อมูลที่ประกอบด้วย Header และ Data เพื่อส่งข้อมูลการเชื่อมต่อมายัง Ethernet Controller แล้วในส่วนของอุปกรณ์ควบคุมการเชื่อมต่อระบบเครือข่ายจะส่งแต่เพียง Data มายัง MCS-51 ดังรูปที่ 3.12



รูปที่ 3.12 แสดงกระบวนการรับข้อมูลของระบบ

3.7 การควบคุม Ethernet Controller และเครื่องคอมพิวเตอร์

ในการควบคุม Ethernet Controller โดยการใช้ MCS-51 เพื่อที่จะให้ได้กระบวนการส่งและรับข้อมูลของระบบตามที่เราต้องการจะแบ่งเป็น 2 กรณี ดังนี้

- กรณีที่ 1 ไอพีแอดเดรสของคอมพิวเตอร์อยู่ในวง LAN เดียวกันกับ Ethernet Controller

1. ทำการรีเซตและตั้งค่าให้กับ Ethernet Controller พร้อมทั้งรับค่าการเซ็ตไอพีแอดเดรสของ Ethernet Controller จาก Dip switch

2. ส่งเฟรม ICMP Echo Request ไปยังเครื่องคอมพิวเตอร์เพื่อตรวจสอบสถานะการเชื่อมต่อ

3. รอรับเฟรมข้อมูลที่เข้ามา ถ้ายังไม่มีเฟรมข้อมูลเข้ามาให้รอรับไปเรื่อยๆ ถ้าเฟรมที่เข้ามาเป็น ARP Request ที่มี IP ตรงกับ Ethernet Controller ให้ทำการส่ง ARP Reply กลับไปยังเครื่องคอมพิวเตอร์

4. รอรับเฟรมข้อมูลที่เข้ามา ถ้ายังไม่มีเฟรมข้อมูลเข้ามาให้รอรับไปเรื่อยๆ ถ้าเฟรมที่เข้ามาเป็น ICMP Reply แสดงว่า Ethernet Controller และคอมพิวเตอร์สามารถเชื่อมต่อกันได้

5. รอรับข้อมูลค่าความชื้นและอุณหภูมิจากเซนเซอร์ จากนั้นก็ส่งเฟรมข้อมูลดังกล่าวไปยังคอมพิวเตอร์แล้วกลับมารอรับค่าข้อมูลต่อไปเรื่อยๆ (วนทำข้อ 5 ไปเรื่อยๆ)

- *กรณีที่ 2 ไอพีแอดเดรสของคอมพิวเตอร์ไม่ได้อยู่ในวง LAN เดียวกัน กับ Ethernet Controller*

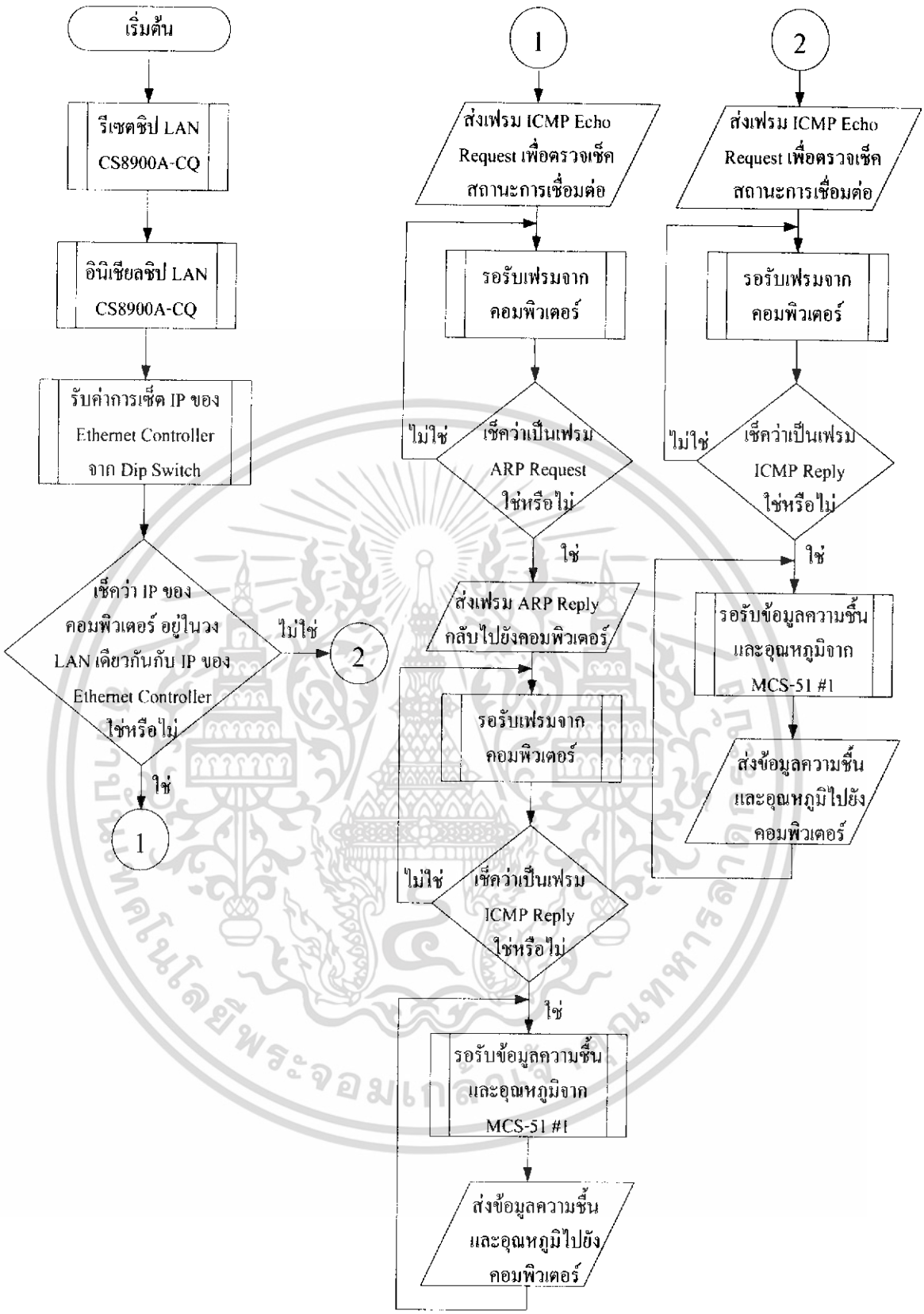
1. ทำการรีเซตและตั้งค่าให้กับ Ethernet Controller พร้อมทั้งรับค่าการเซ็ตไอพีแอดเดรสของ Ethernet Controller จาก Dip switch

2. ส่งเฟรม ICMP Echo Request ไปยังเครื่องคอมพิวเตอร์เพื่อตรวจสอบสถานะการเชื่อมต่อ

3. รอรับเฟรมข้อมูลที่เข้ามา ถ้ายังไม่มีเฟรมข้อมูลเข้ามาให้รอรับไปเรื่อยๆ ถ้าเฟรมที่เข้ามาเป็น ICMP Reply แสดงว่า Ethernet Controller และคอมพิวเตอร์สามารถเชื่อมต่อกันได้

4. รอรับข้อมูลค่าความชื้นและอุณหภูมิจากเซนเซอร์ จากนั้นก็ส่งเฟรมข้อมูลดังกล่าวไปยังคอมพิวเตอร์แล้วกลับมารอรับค่าข้อมูลต่อไปเรื่อยๆ (วนทำข้อ 4 ไปเรื่อยๆ)

โดยกระบวนการของทั้งสองกรณีจะเป็นไปดัง Flow Chart ในรูปที่ 3.13



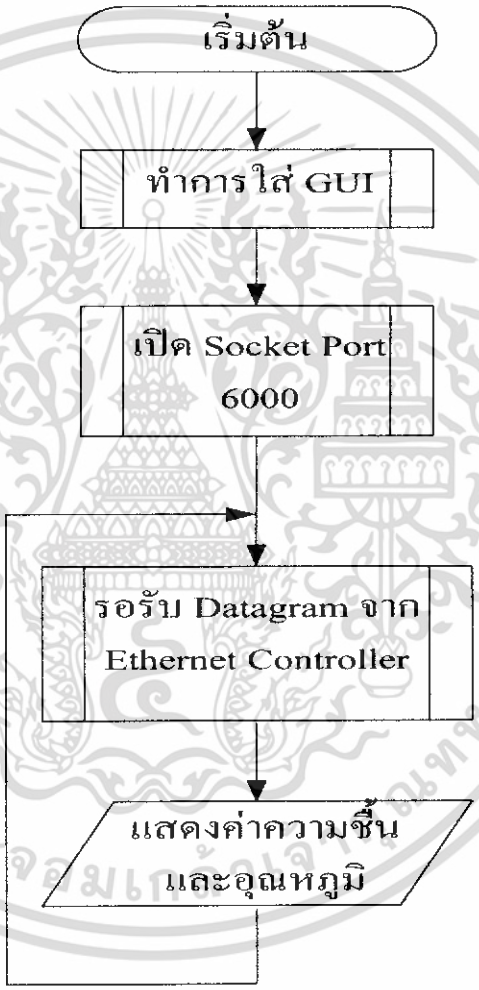
รูปที่ 3.13 แสดง Flow Chart ของโปรแกรมที่ใช้ใน Ethernet Controller

และในส่วนของเครื่องคอมพิวเตอร์จะใช้โปรแกรมภาษาจาวา (JAVA)สำหรับการรับค่าข้อมูลความชื้นและอุณหภูมิมาแสดงผลบนหน้าจอ window ซึ่งกระบวนการส่งและรับข้อมูลของระบบ ตามที่เราต้องการจะต้องทำดังนี้

1. ทำการสร้าง GUI (Graphic User Interface) ชนิด J เฟรม และเปิด Socket Port ของเครื่องคอมพิวเตอร์ ในที่นี้เราใช้พอร์ต 6000

2. รอรับเฟรมข้อมูล UDP จาก Ethernet Controller (ในส่วนของเฟรม ICMP และ ARP การ์ดแลนจะเป็นตัวจัดการในการส่งเฟรมเอง ไม่เกี่ยวข้องกับโปรแกรมนี้)

3. นำค่าข้อมูลที่รับมาแสดงผลไปเรื่อยๆ ซึ่งกระบวนการดังกล่าวเป็นดัง Flow Chart รูปที่ 3.14



รูปที่ 3.14 แสดง Flow Chart ของโปรแกรมที่ใช้ในเครื่องคอมพิวเตอร์

บทที่ 4

การทดลองและผลการทดลอง

หลังจากทราบถึงทฤษฎีหรือหลักการในบทที่ 2 และการคำนวณรวมไปถึงการสร้างวงจรในบทที่ 3 เราก็สามารถที่จะทำการทดลอง Project ชิ้นนี้ด้วยการทำการทดสอบการติดต่อกันระหว่างระบบคอมพิวเตอร์กับ Ethernet Controller เพื่อทำการแสดงผลของอุณหภูมิและความชื้นที่รับมาจากตัวอุปกรณ์เซนเซอร์ ซึ่งมีขั้นตอนและผลการทดลองดังนี้

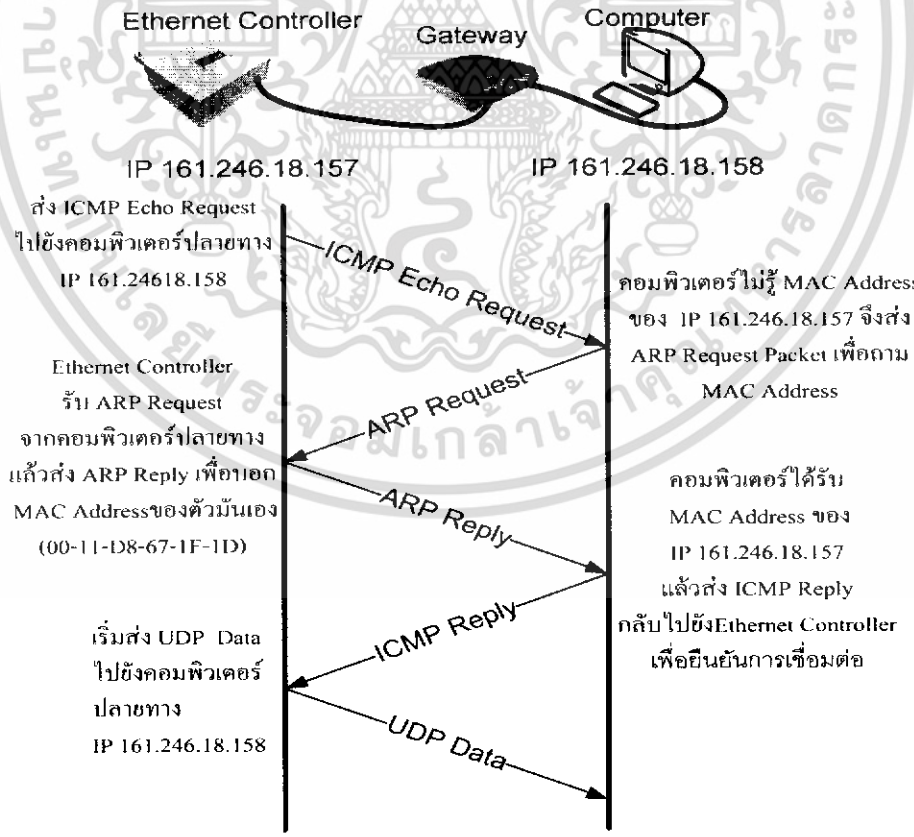
4.1 การทดลองที่ 1 เป็นการทดลองการเชื่อมต่อของวงจรทั้งหมดซึ่งประกอบด้วยส่วนของวงจรเชื่อมต่อไมโครคอนโทรลเลอร์ตัวที่ 1 กับเซนเซอร์ และไมโครคอนโทรลเลอร์ตัวที่ 2 กับ Ethernet Controller พร้อมส่งไปยังคอมพิวเตอร์ ที่มี IP Address อยู่ในวง LAN เดียวกัน

ขั้นตอนการทดลอง

1. ทำการเขียนโปรแกรมลงบนไมโครคอนโทรลเลอร์ตัวที่ 1 ที่เชื่อมต่อกับเซนเซอร์ดัง Flow chart รูปที่ 3.4 และเขียนโปรแกรมลงบนไมโครคอนโทรลเลอร์ตัวที่ 2 ที่เชื่อมกับ Ethernet Controller ดัง Flow chart รูปที่ 3.13 แล้วทำการเขียนโปรแกรม (Ethernet_Controller.java) ลงบนเครื่องคอมพิวเตอร์ดัง Flow chart รูปที่ 3.14

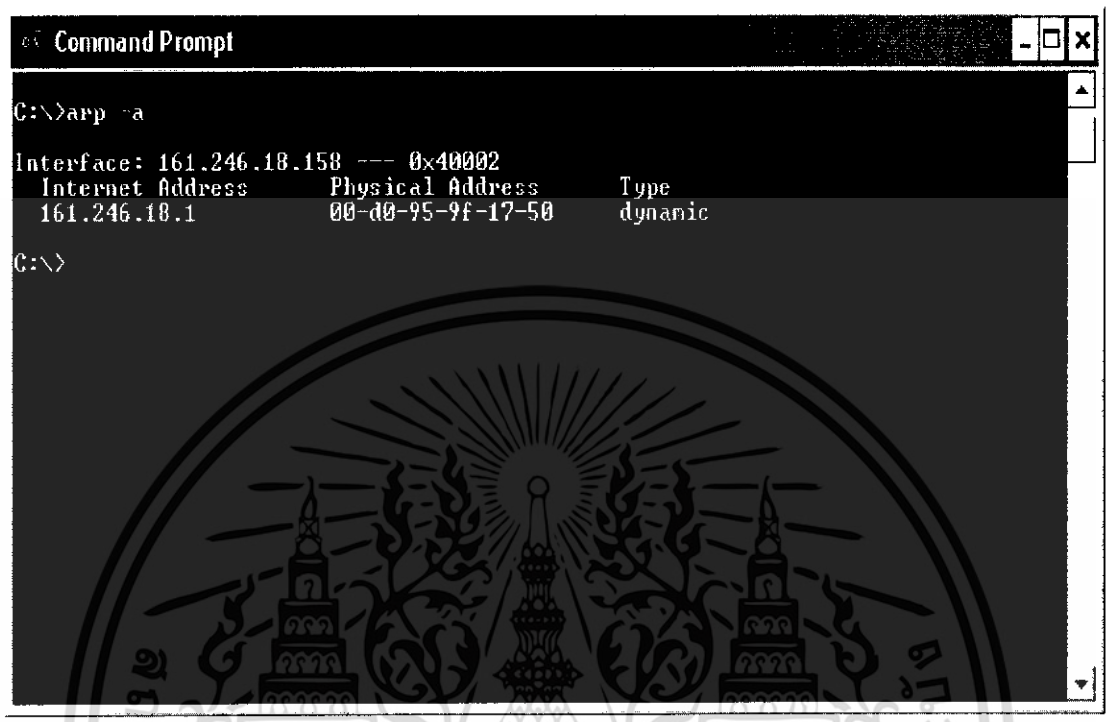
2. ทำการเชื่อม Ethernet Controller และ เครื่องคอมพิวเตอร์เข้าด้วยกันผ่าน Default Gateway รันโปรแกรม Ethernet_Controller.java บนเครื่องคอมพิวเตอร์ (IP 161.246.18.158) ซึ่งใช้ Port 6000

3. สังเกตการทำงานของ Datagram ดังรูปที่ 4.1 แล้วเก็บผลการทดลอง



รูปที่ 4.1 แสดงกระบวนการรับส่งข้อมูลระหว่าง Ethernet Controller กับคอมพิวเตอร์ที่มี IP อยู่ในวง LAN เดียวกัน

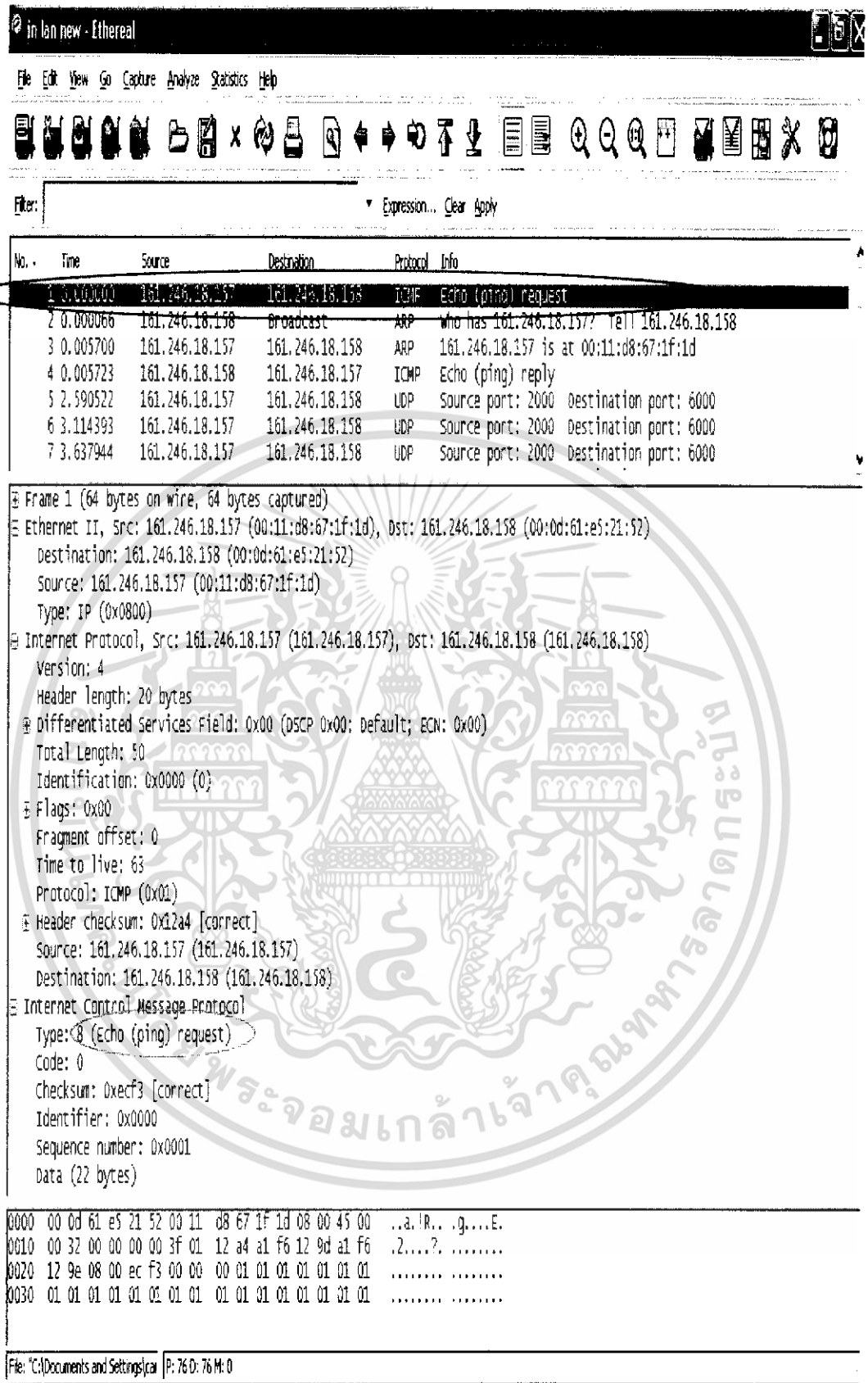
ก่อนการทดลองเมื่อทำการเช็ค Cache Memory ของ Network Interface Card เพื่อทำการตรวจสอบ MAC Address ของอุปกรณ์ที่เชื่อมต่อกับคอมพิวเตอร์ โดยใช้คำสั่ง arp –a ใน MS-DOS Command Prompt จะปรากฏผลดังรูปที่ 4.2



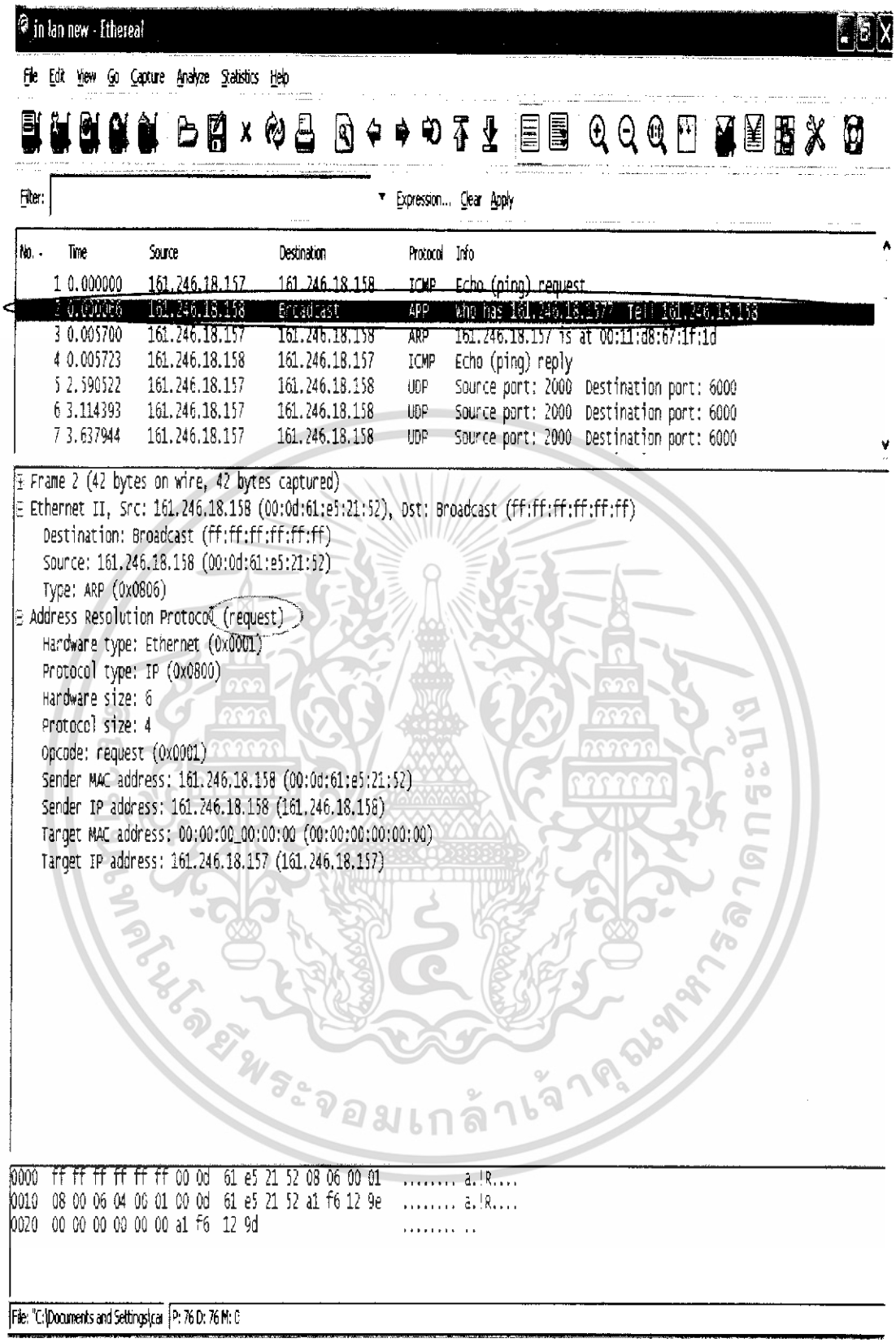
รูปที่ 4.2 แสดง MAC Address ใน Cache Memory

จะเห็นว่าในตอนแรกยังไม่มี MAC Address ของ Ethernet Controller จะพบเพียง MAC Address ของ Default Gateway (IP 161.246.18.1) ซึ่งทำให้เครื่องคอมพิวเตอร์จะยังไม่สามารถส่งเฟรมข้อมูลได้ เพราะว่ายังขาด Destination Address ของ Ethernet Controller

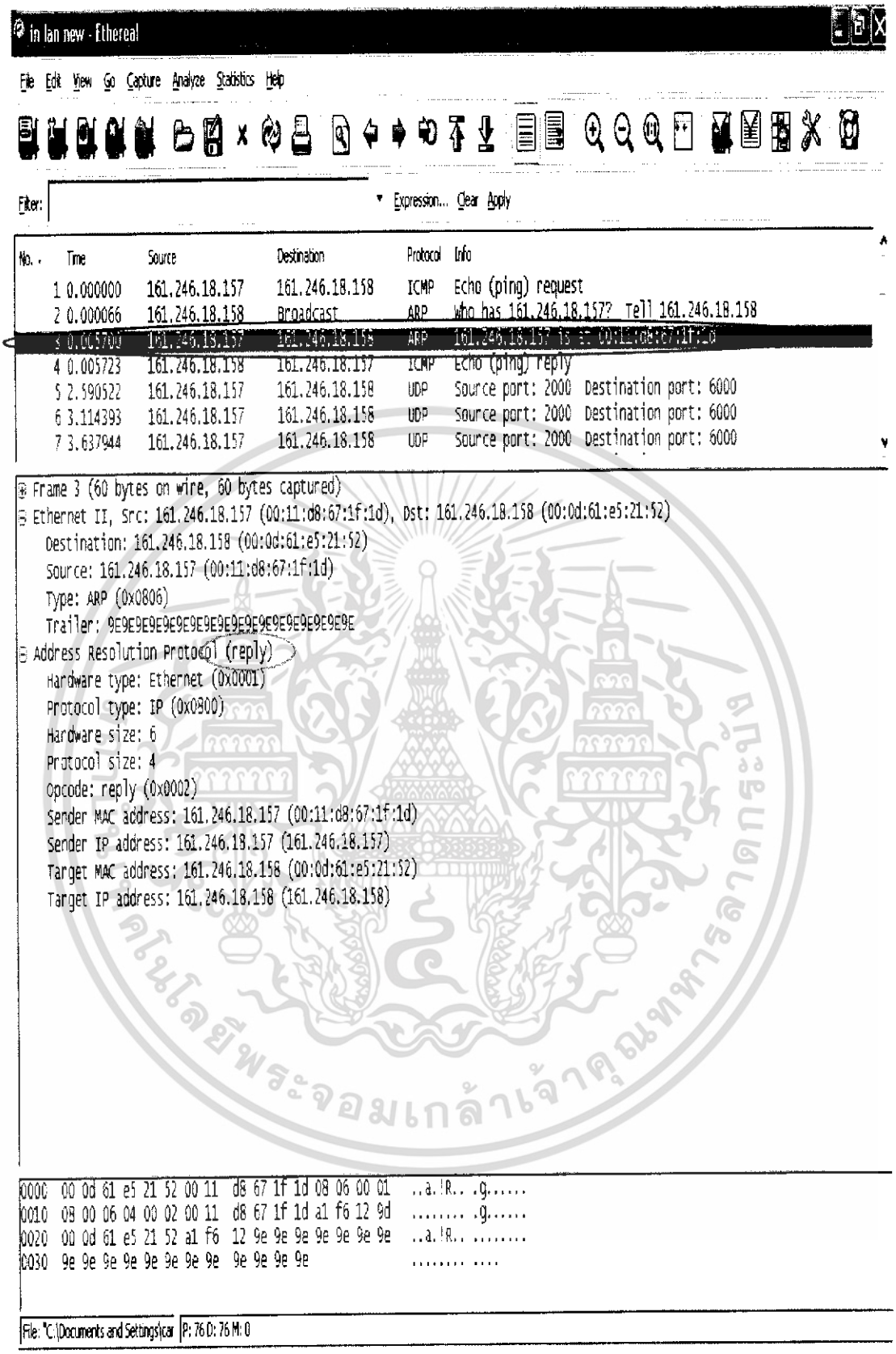
จึงต้องส่งเฟรมข้อมูลเริ่มแรกไปก่อนซึ่งเป็นเฟรม ICMP Echo Request เพื่อตรวจสอบสถานะการเชื่อมต่อของ Ethernet Controller กับคอมพิวเตอร์ (เริ่มแรกอุปกรณ์ทั้งสองยังไม่รู้จักกัน) ซึ่งเครื่องคอมพิวเตอร์จะส่ง ARP Request มายัง Ethernet Controller และให้ Ethernet Controller ส่ง ARP Reply กลับมายังเครื่องคอมพิวเตอร์ จากนั้นคอมพิวเตอร์ก็จะส่ง ICMP Reply กลับไปเพื่อบอกว่าขณะนี้อุปกรณ์ทั้งสองได้เชื่อมต่อกันแล้ว Ethernet Controller ก็จะเริ่มส่งข้อมูลที่ต้องการได้ทันที ซึ่งได้ผลการ Capture Datagram ดังรูปที่ 4.3 , 4.4 , 4.5 , 4.6 , 4.7



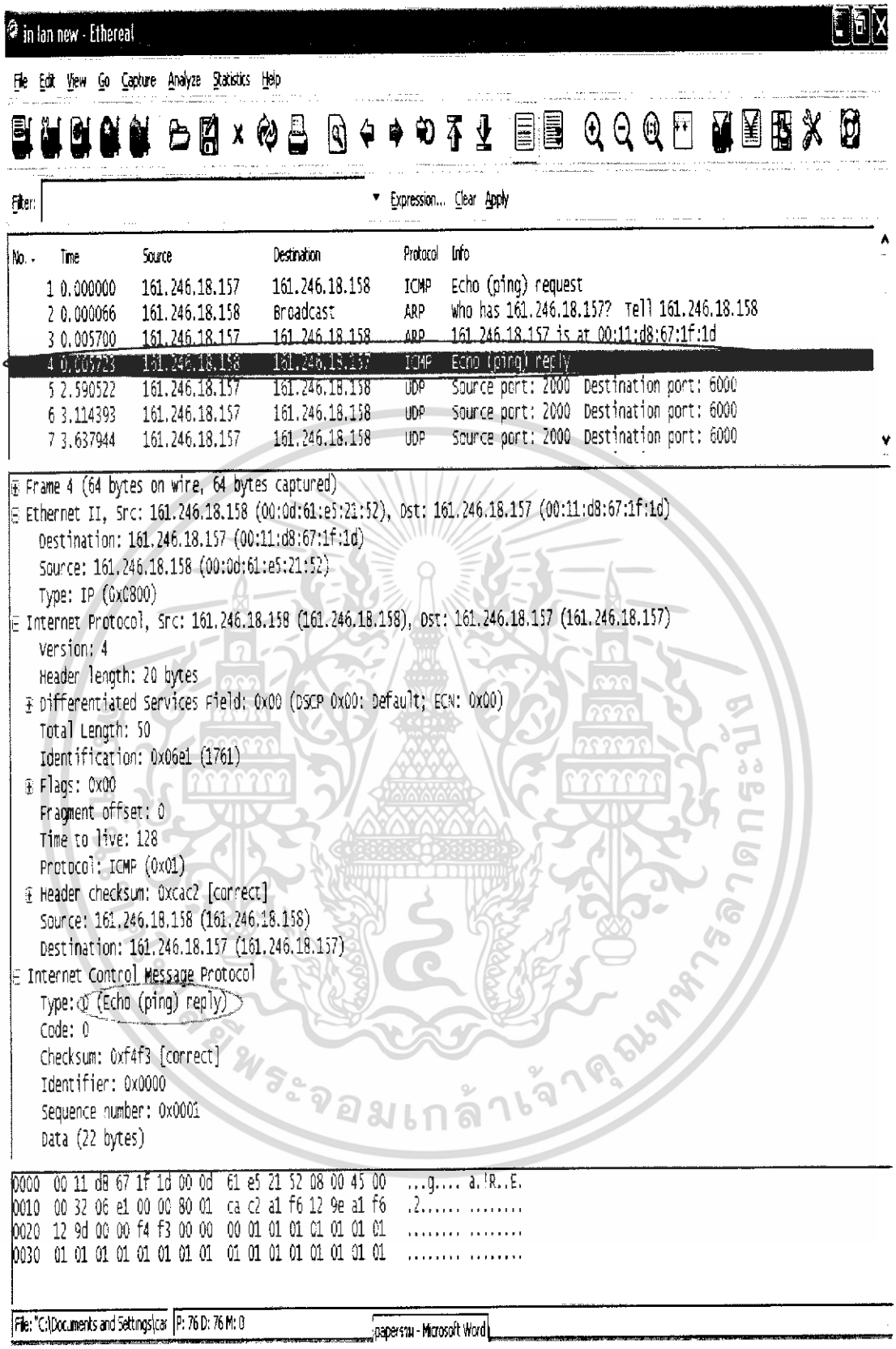
รูปที่ 4.3 แสดงผลของเฟรม ICMP Echo Request ที่ Ethernet Controller (IP 161.246.18.157) ส่งมาตรวจสอบสถานะการเชื่อมต่อกับคอมพิวเตอร์ (IP 161.246.18.158)



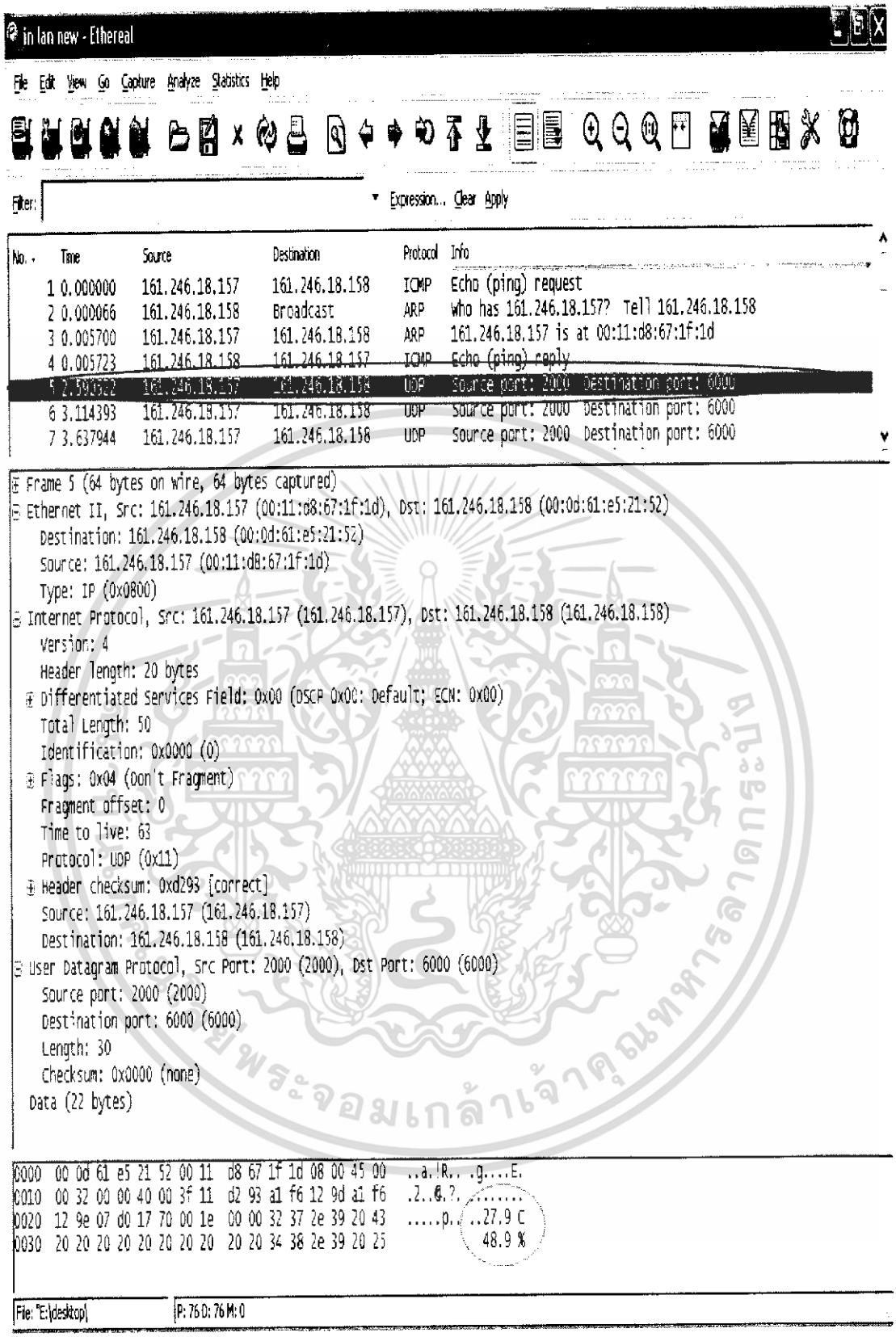
รูปที่ 4.4 แสดงผลของเฟรม ARP Request ที่คอมพิวเตอร์ส่งแบบ Broadcast มาเพื่อถาม MAC Address ของ Ethernet Controller IP 161.246.18.157



รูปที่ 4.5 แสดงผลของเฟรม ARP Reply ที่ Ethernet Controller ซึ่งมี IP 161.246.18.157 ส่งไปยังเครื่องคอมพิวเตอร์เพื่อบอก MAC Address (00-11-D8-67-1F-1D) ของตัวมันเอง

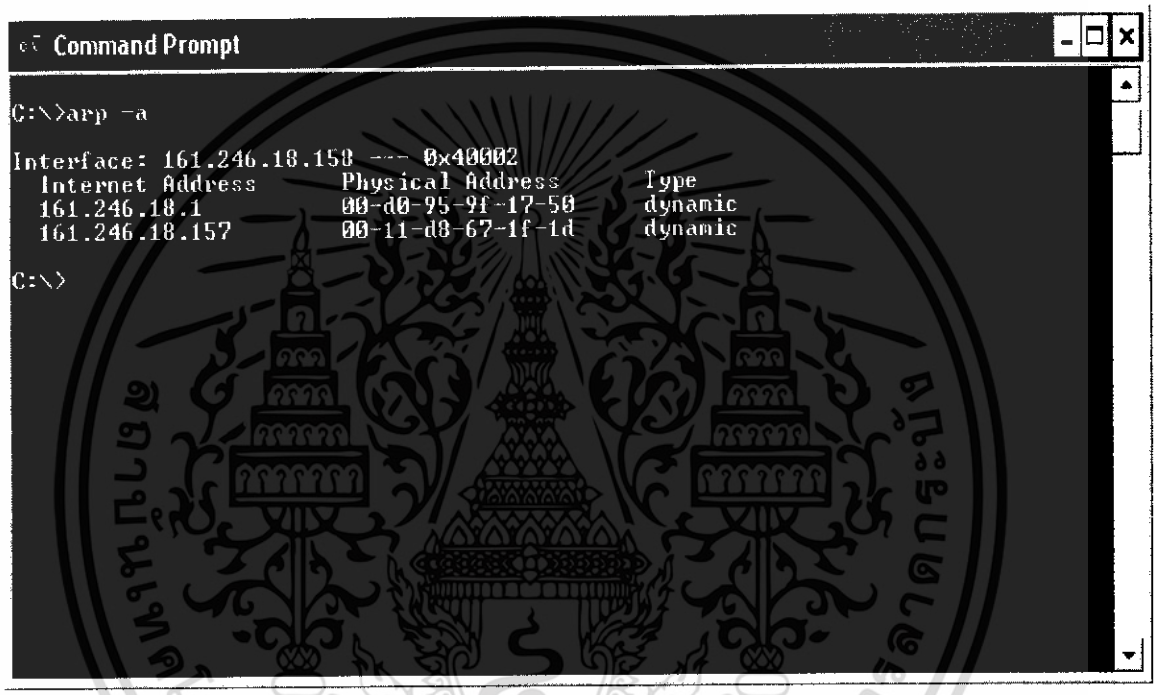


รูปที่ 4.6 แสดงผลของเฟรม ICMP Echo Reply ที่คอมพิวเตอร์ส่งกลับไปให้ Ethernet Controller เพื่อยืนยันการเชื่อมต่อ และพร้อมที่จะรับข้อมูลจาก Ethernet Controller แล้ว

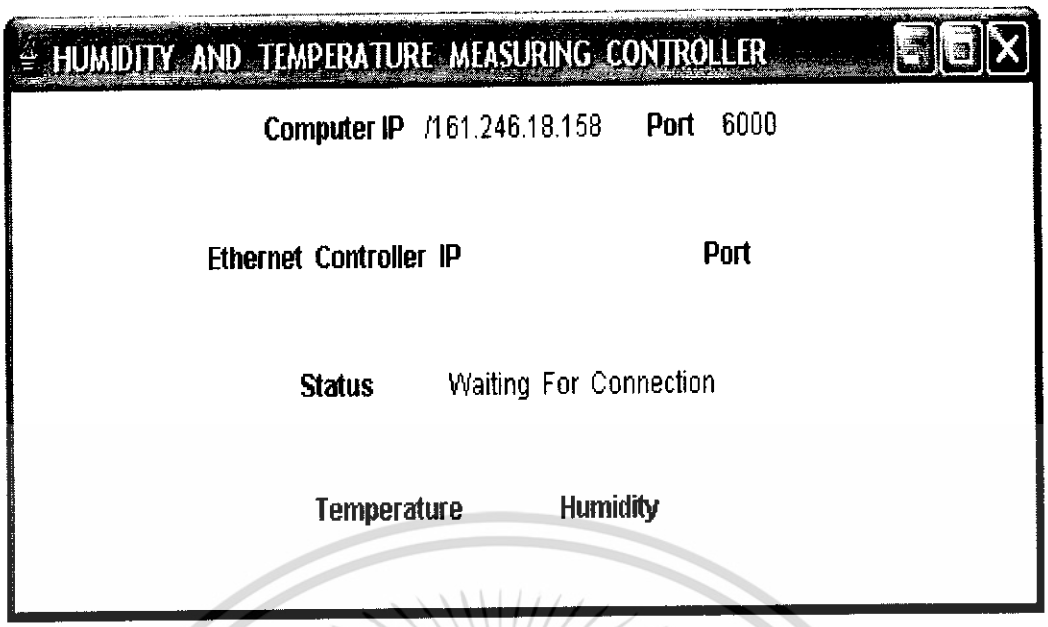


รูปที่ 4.7 แสดงเฟรม UDP ที่ถูกส่งจาก Ethernet Controller ไปยังคอมพิวเตอร์ซึ่งมีข้อมูลของอุณหภูมิและความชื้นแนบอยู่

จะเห็นว่าเมื่อเครื่องคอมพิวเตอร์ได้รับเฟรม ICMP Echo Request จาก Ethernet Controller แล้วก็จะส่ง ARP Request แบบ Broadcast Datagram ออกไป โดยการถามหา IP Address 161.246.18.157 (Ethernet Controller) ซึ่งเครื่องคอมพิวเตอร์ทุกเครื่องในระบบเครือข่ายจะได้รับเฟรมนี้ทุกเครื่องถ้าเครื่องใดมี IP Address 161.246.18.157 ก็ให้ทำการตอบ ARP Reply กลับไปยังเครื่องที่ส่ง ARP Request มาซึ่งในกรณีนี้ Ethernet Controller จะทำการตอบ ARP Reply กลับไปยังคอมพิวเตอร์ เป็นผลให้ Network Interface Card รู้ว่า IP Address 161.246.18.157 มีค่า MAC Address เท่าไร เมื่อทำการเช็ค Cache Memory ของ Network Interface Card ก็จะปรากฏ MAC Address ของ Ethernet Controller (IP 161.246.18.157) ดังรูปที่ 4.8

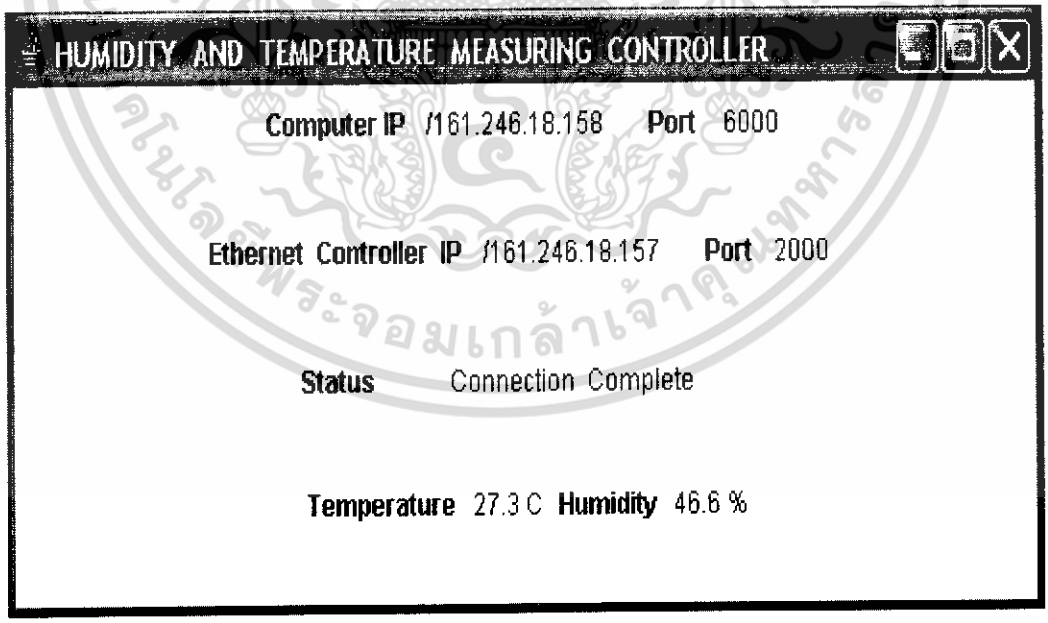


รูปที่ 4.8 แสดง MAC Address ใน Cache Memory ของ Ethernet Controller (IP 161.246.18.157) หลังจากได้รับ ARP Reply

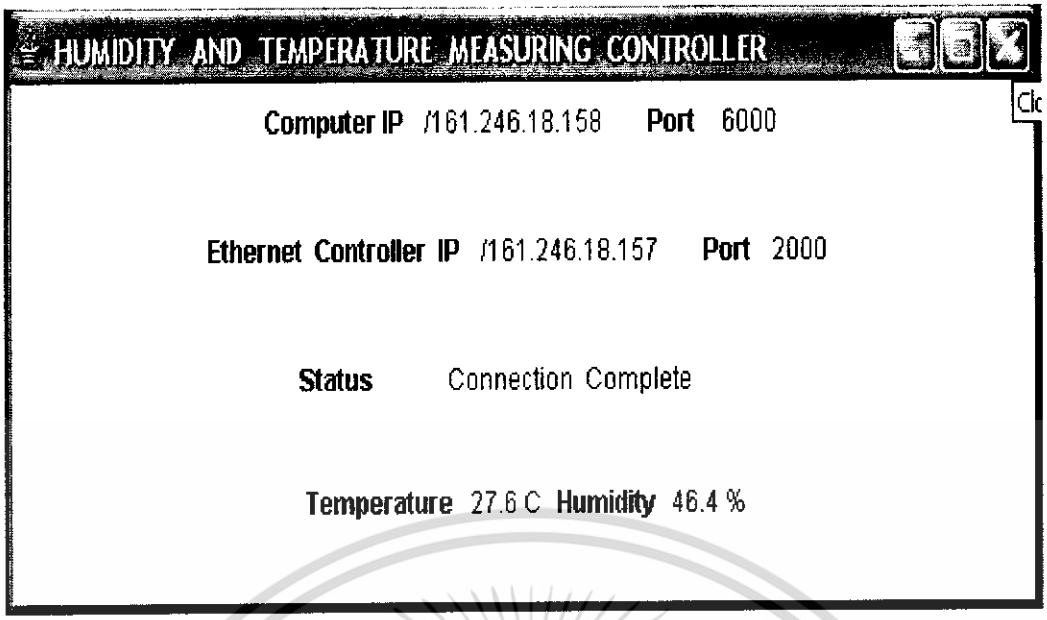


รูปที่ 4.9 แสดงโปรแกรม Ethernet_Controller.java ที่รันออกมาบนเครื่องคอมพิวเตอร์ (IP 161.246.18.158 Port 6000) ในขณะที่ยังไม่มีการเชื่อมต่อกับ Ethernet Controller

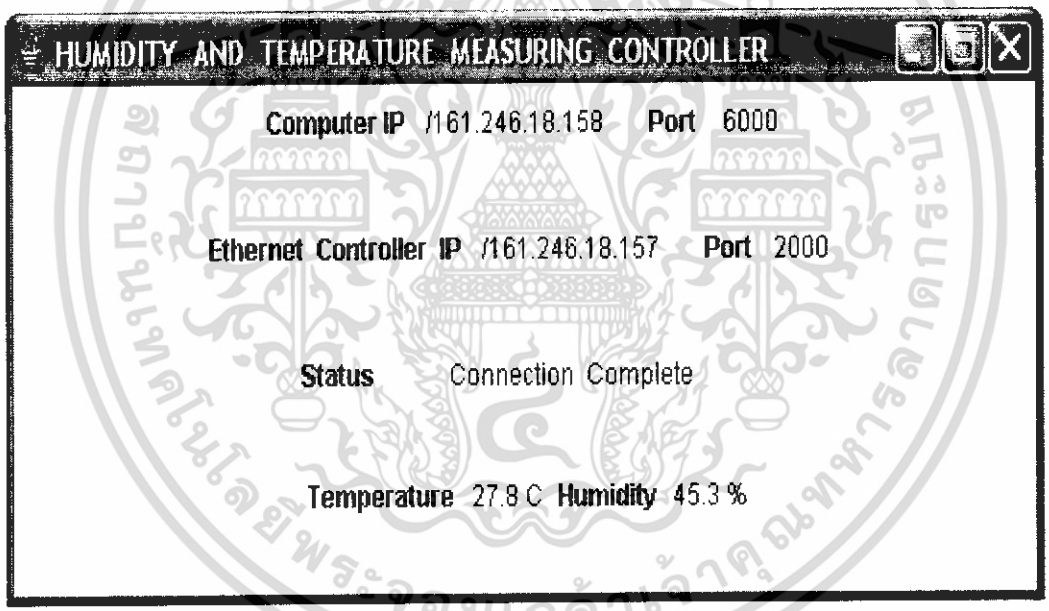
รอการเชื่อมต่อระหว่างคอมพิวเตอร์กับ Ethernet Controller ถ้าเชื่อมต่อกันแล้วที่หน้าจอจะขึ้น IP 161.246.18.157 Port 2000 และที่ช่อง Status จะปรากฏคำว่า Connection Complete พร้อมทั้งปรากฏค่าข้อมูลของอุณหภูมิและความชื้น ดังรูปที่ 4.10 (ก) , (ข) และ (ค)



(ก)



(ข)



(ค)

รูปที่ 4.10 (ก), (ข) และ (ค) แสดงค่าของอุณหภูมิและความชื้นที่เปลี่ยนแปลงเมื่อได้ทำการเชื่อมต่อคอมพิวเตอร์กับ Ethernet Controller

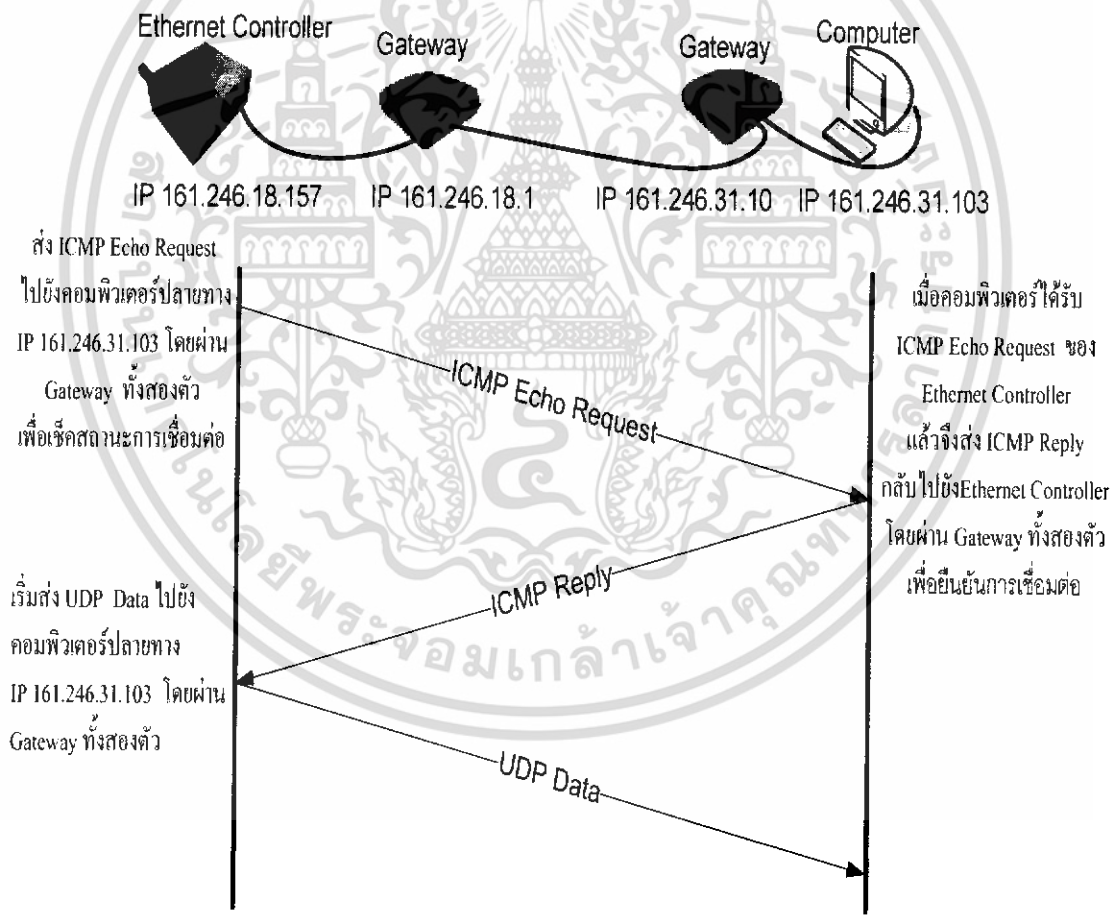
4.2 การทดลองที่ 2 เป็นการทดลองการเชื่อมต่อของวงจรทั้งหมดซึ่งประกอบด้วยส่วนของวงจรเชื่อมต่อไมโครคอนโทรลเลอร์ตัวที่ 1 กับเซนเซอร์ และไมโครคอนโทรลเลอร์ตัวที่ 2 กับ Ethernet Controller พร้อมส่งไปยังคอมพิวเตอร์ที่มี IP Address ไม่ได้อยู่ในวง LAN เดียวกัน

ขั้นตอนการทดลอง

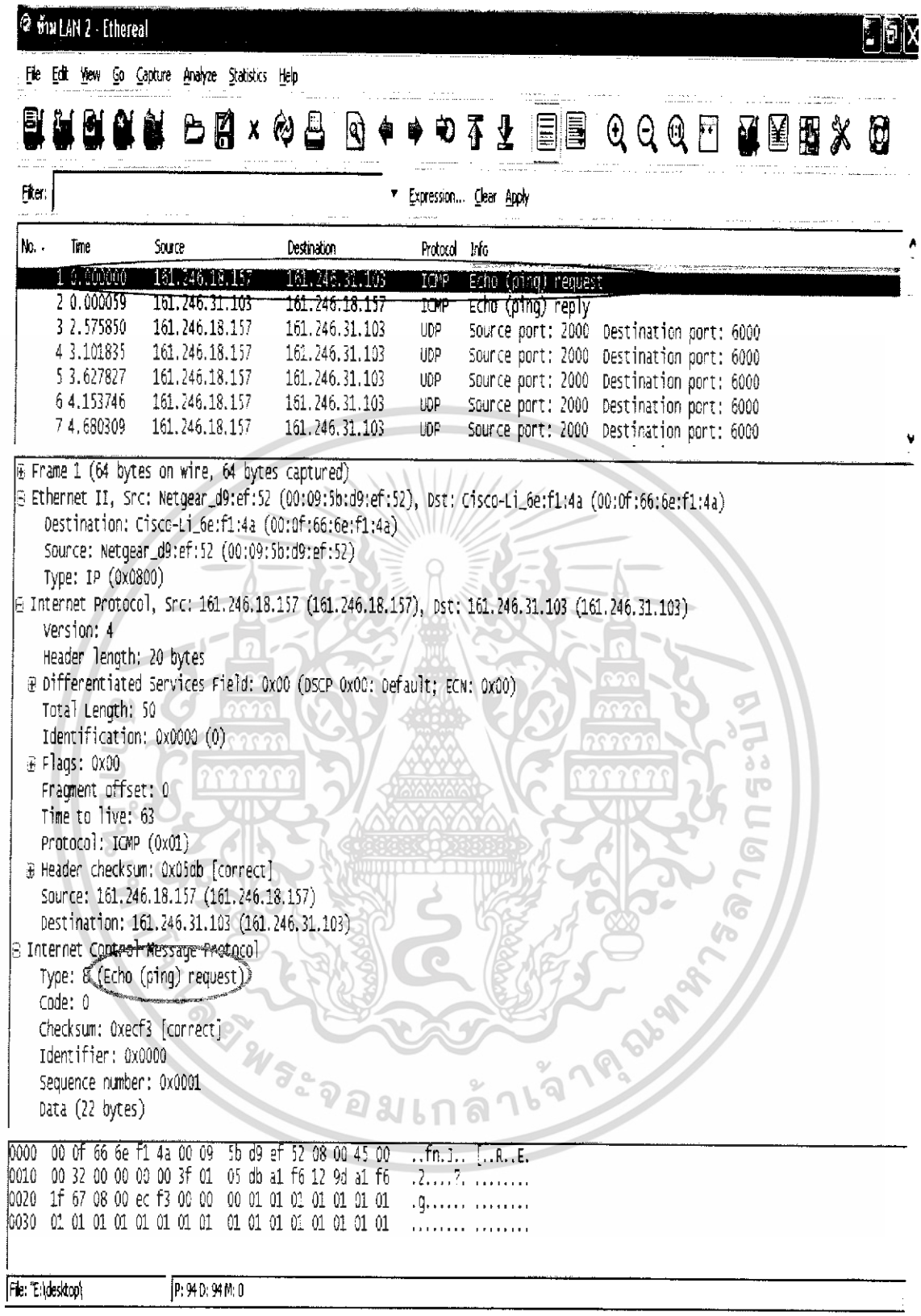
1. ทำการเขียนโปรแกรมลงบนไมโครคอนโทรลเลอร์ตัวที่ 1 ที่เชื่อมต่อกับเซนเซอร์ดัง Flow chart รูปที่ 3.4 และเขียนโปรแกรมลงบนไมโครคอนโทรลเลอร์ตัวที่ 2 ที่เชื่อมกับ Ethernet Controller ดัง Flow chart รูปที่ 3.13 แล้วทำการเขียนโปรแกรม (Ethernet_Controller.java) ลงบนเครื่องคอมพิวเตอร์ดัง Flow chart รูปที่ 3.14

2. ทำการเชื่อม Ethernet Controller เข้ากับ Default Gateway (IP 161.246.18.1) และเชื่อมคอมพิวเตอร์เข้ากับ Default Gateway (IP 161.246.31.10) รันโปรแกรม Ethernet_Controller.java บนเครื่องคอมพิวเตอร์ (IP 161.246.31.103) ซึ่งใช้ Port 6000

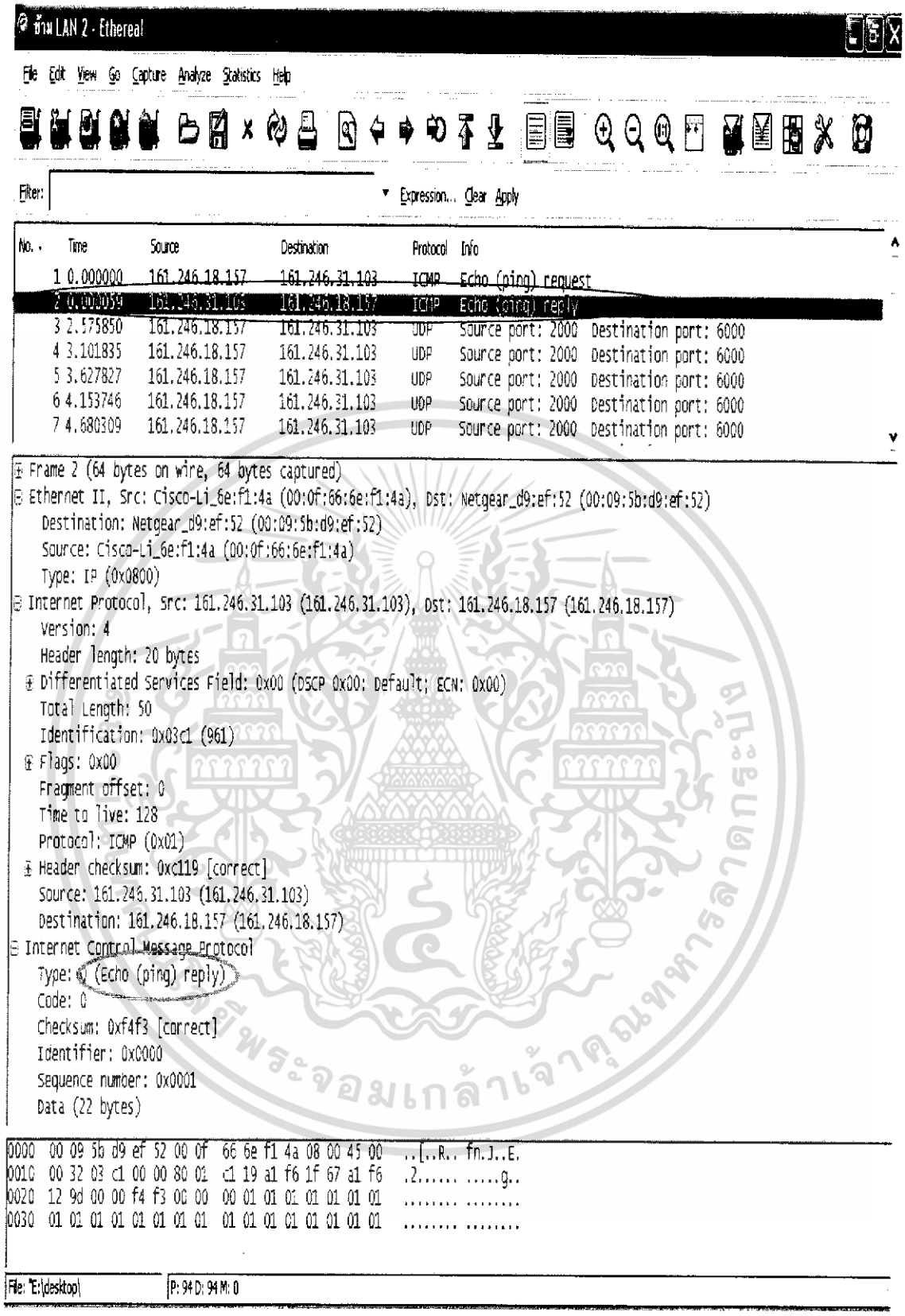
3. สังเกตการทำงานของ Datagram ดังรูปที่ 4.11 แล้วเก็บผลการทดลอง



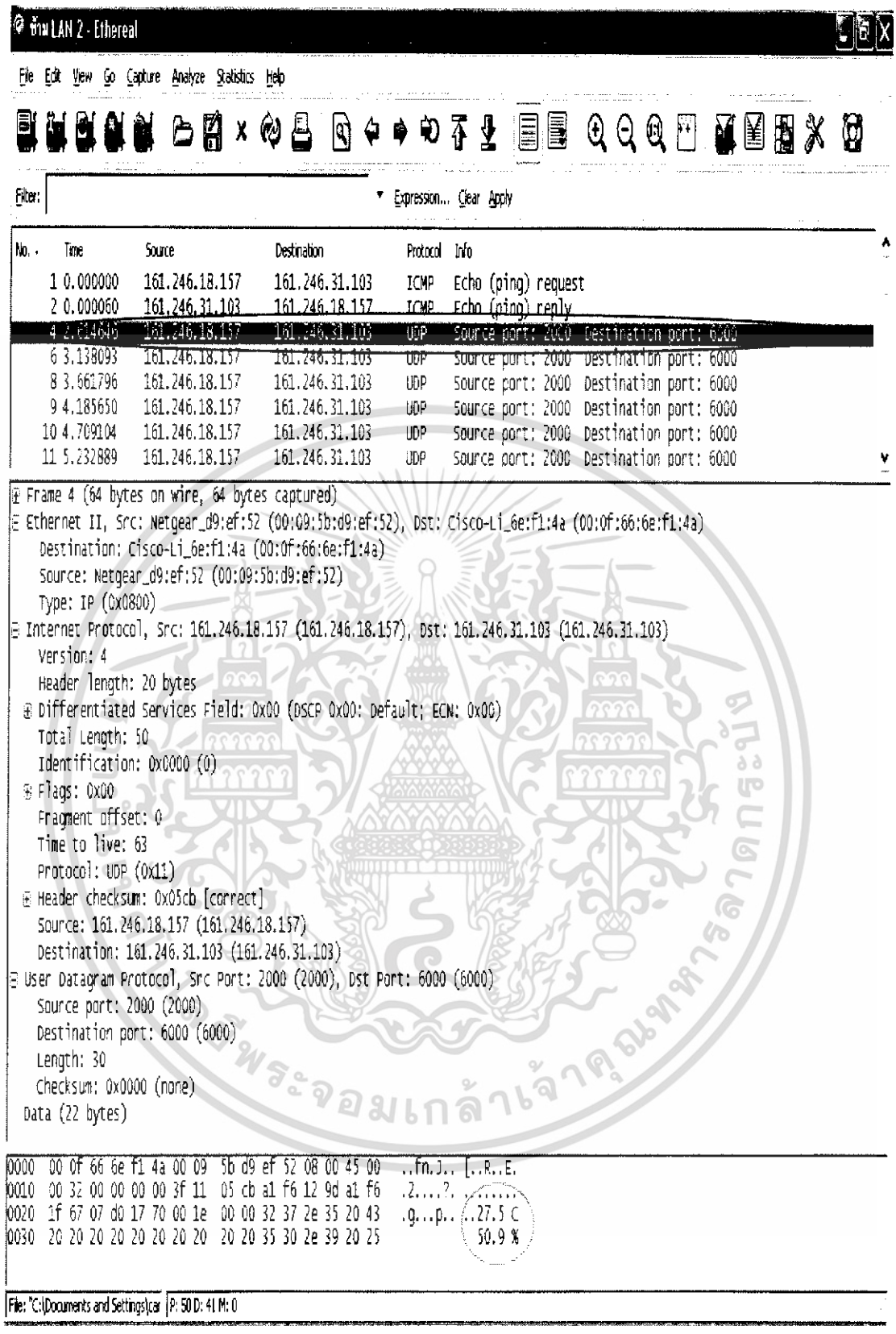
รูปที่ 4.11 แสดงกระบวนการรับส่งข้อมูลระหว่าง Ethernet Controller กับคอมพิวเตอร์ที่มี IP ไม่ได้อยู่ในวง LAN เดียวกัน



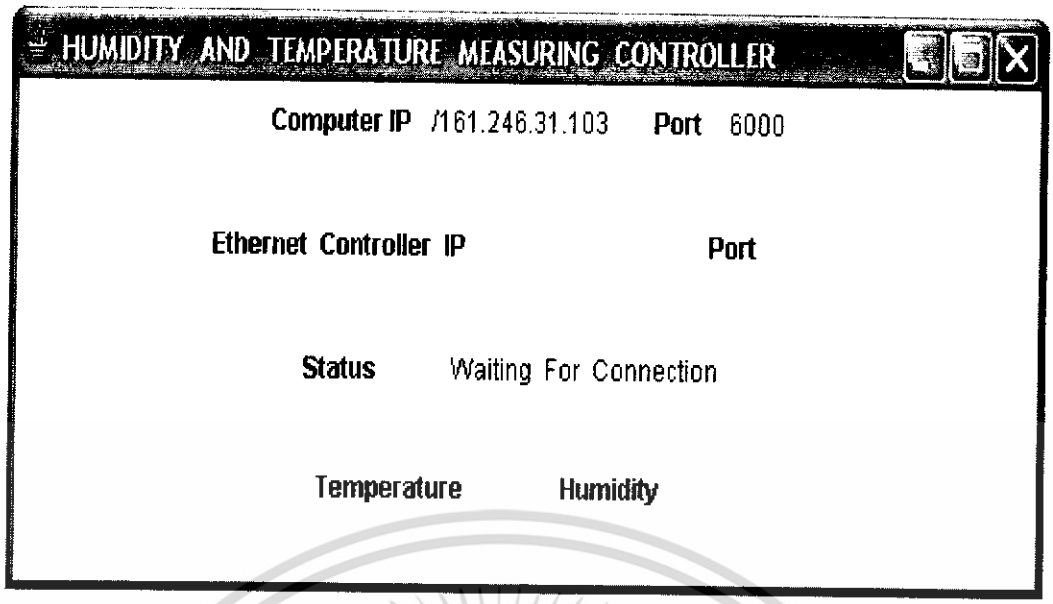
รูปที่ 4.12 แสดงผลของเฟรม ICMP Echo Request ที่ Ethernet Controller (IP 161.246.18.157) ส่งมาตรวจสอบสถานะการเชื่อมต่อกับคอมพิวเตอร์ (IP 161.246.31.103)



รูปที่ 4.13 แสดงผลของเฟรม ICMP Echo Reply ที่คอมพิวเตอร์ส่งกลับไปให้ Ethernet Controller เพื่อยืนยันการเชื่อมต่อ และพร้อมที่จะรับข้อมูลจาก Ethernet Controller แล้ว

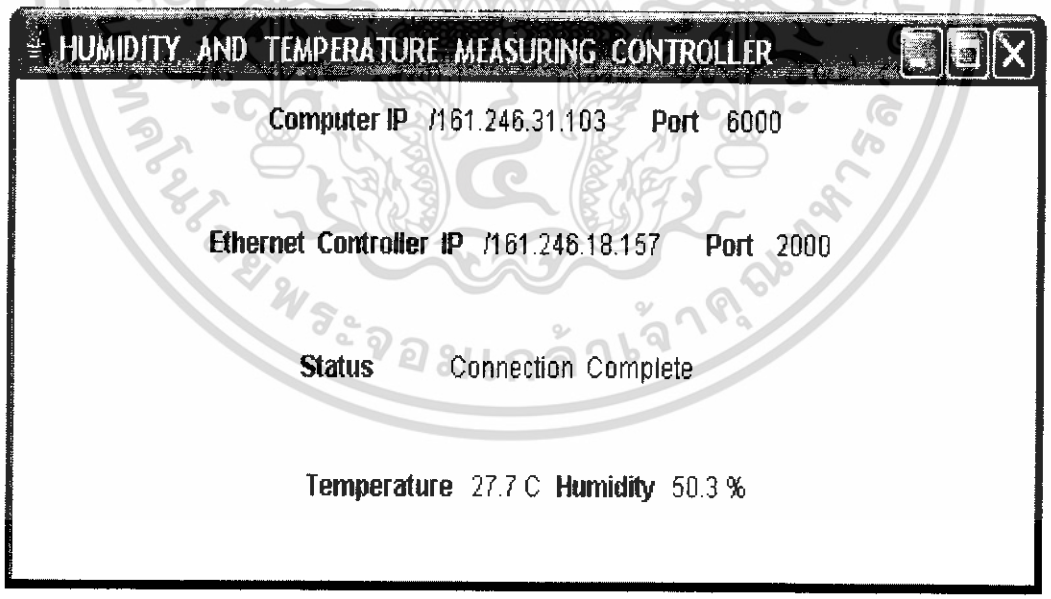


รูปที่ 4.14 แสดงเฟรม UDP ที่ถูกส่งจาก Ethernet Controller ไปยังคอมพิวเตอร์ซึ่งมีข้อมูลของอุณหภูมิและความชื้นแนบอยู่

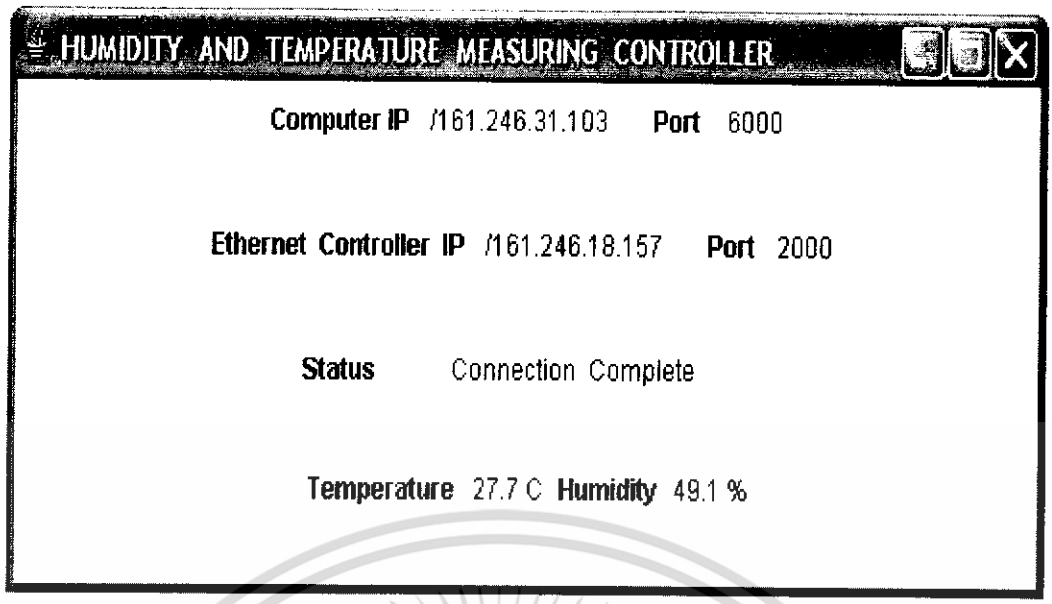


รูปที่ 4.15 แสดงโปรแกรม Ethernet_Controller.java ที่รันออกมาบนเครื่องคอมพิวเตอร์ (IP 161.246.31.103 Port 6000) ในขณะที่ยังไม่มีการเชื่อมต่อกับ Ethernet Controller

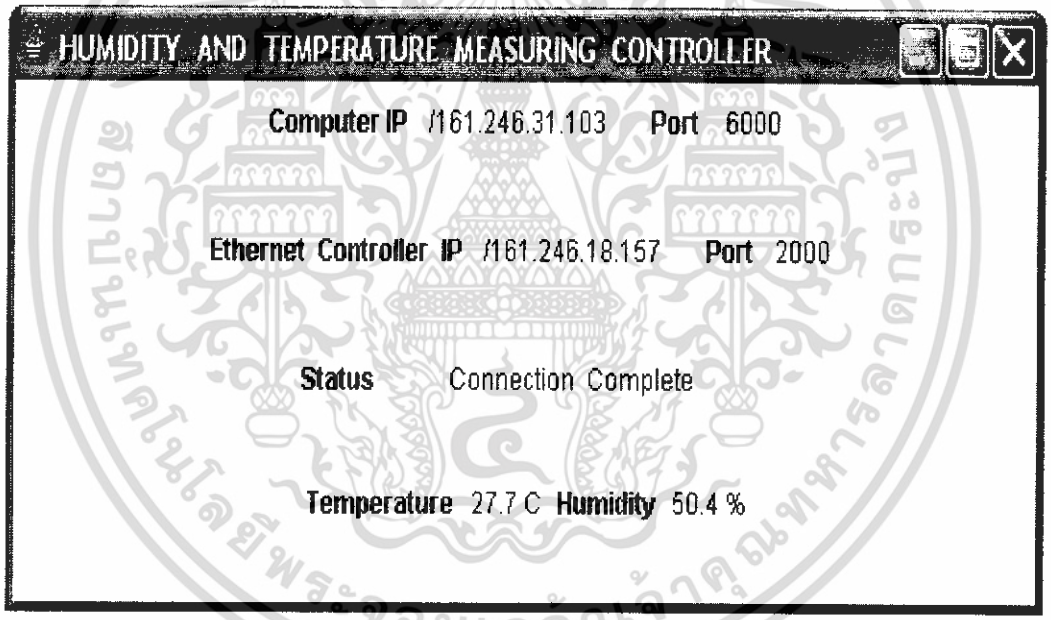
รอการเชื่อมต่อระหว่างคอมพิวเตอร์กับ Ethernet Controller ถ้าเชื่อมต่อกันแล้วที่หน้าจอจะขึ้น IP 161.246.18.157 Port 2000 และที่ช่อง Status จะปรากฏคำว่า Connection Complete พร้อมทั้งปรากฏค่าข้อมูลของอุณหภูมิและความชื้น ดังรูปที่ 4.16 (ก) , (ข) และ (ค)



(ก)

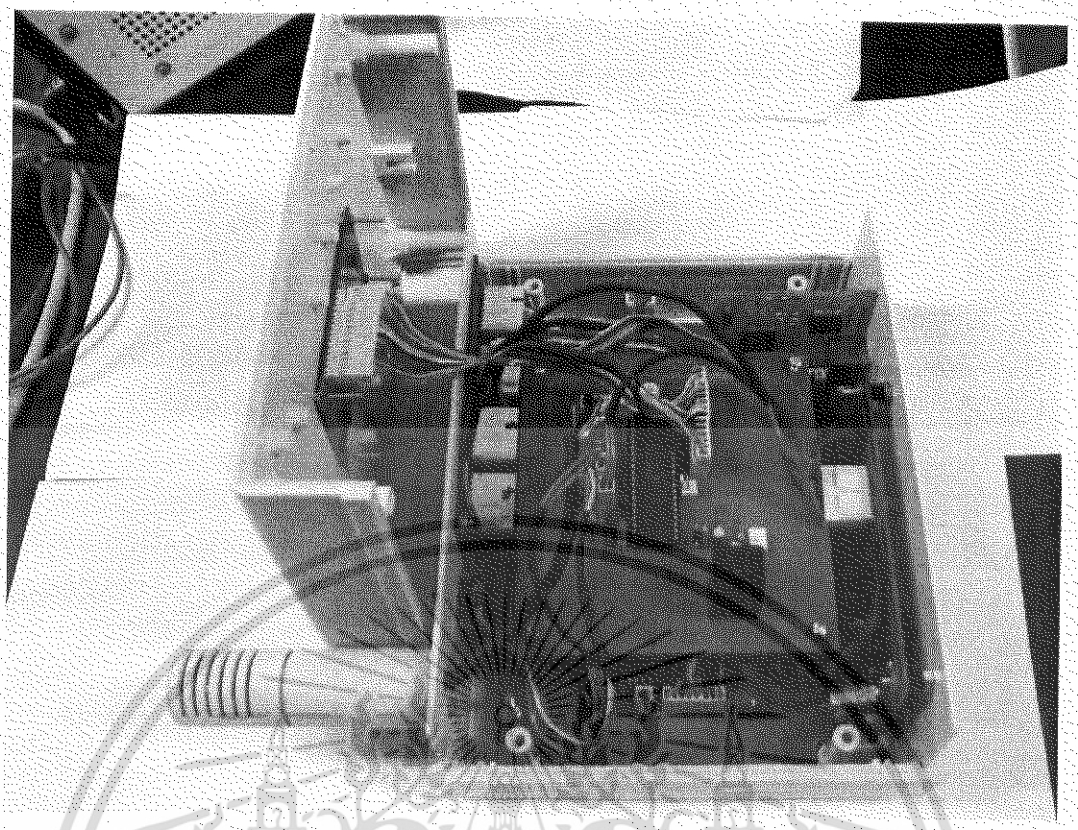


(ข)



(ค)

รูปที่ 4.16 (ก), (ข) และ (ค) แสดงค่าของอุณหภูมิและความชื้นที่เปลี่ยนแปลงเมื่อได้ทำการเชื่อมต่อคอมพิวเตอร์กับ Ethernet Controller

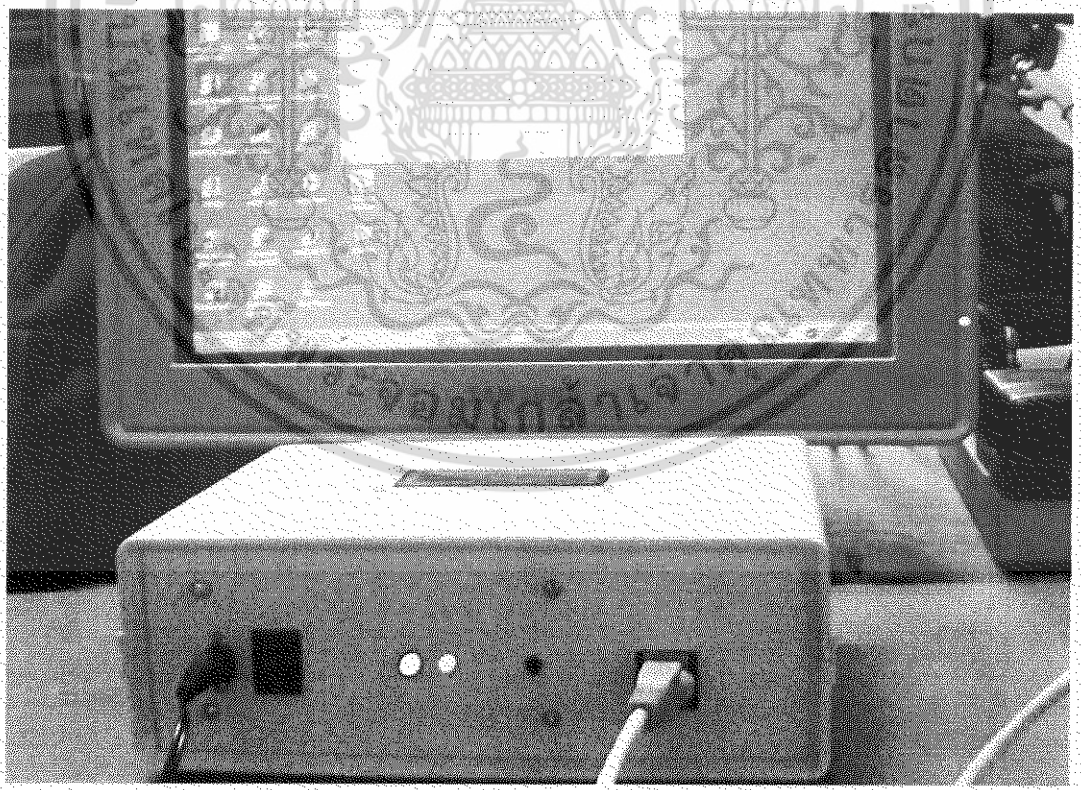


รูปที่ 4.17 แสดงบอร์ดวงจรทั้งหมดขณะลงกล่อง

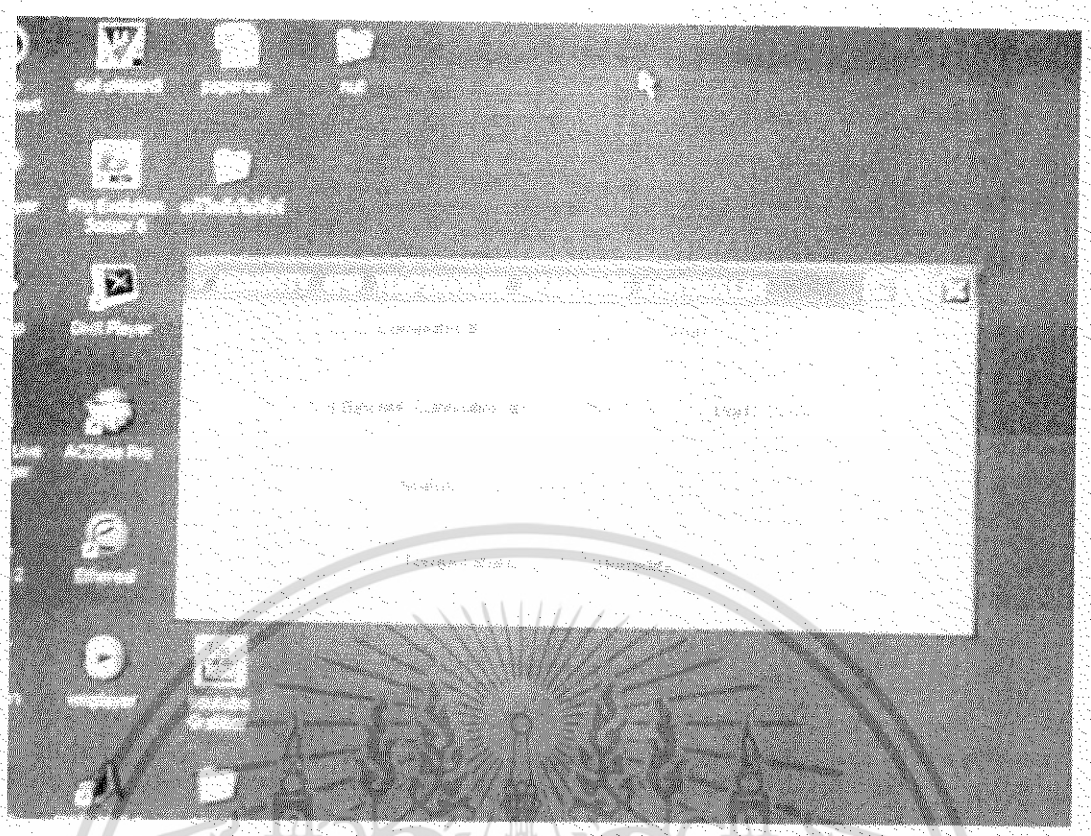
(ก)

(ข)

รูปที่ 4.18 (ก), (ข) แสดงรูปชิ้นงาน



รูปที่ 4.19 แสดงการเชื่อมต่อชิ้นงานกับคอมพิวเตอร์



รูปที่ 4.20 แสดงรูปผลบนหน้าจอคอมพิวเตอร์

รูปที่ 4.21 แสดงรูปผลบนหน้าจอ LCD ของชิ้นงาน

บทที่ 5
บทวิจารณ์และบทสรุป

โครงงาน Project ชิ้นนี้เป็นการสร้างเครื่องวัดอุณหภูมิและความชื้นมาแสดงผลบนเครื่องคอมพิวเตอร์ผ่านเครือข่ายอีเทอร์เนต (Ethernet Network) โดยใช้ฮาร์ดแวร์ที่เรียกว่า "Ethernet Controller" โดยระบบที่ใช้ในการเชื่อมต่อเข้ากับเครือข่ายอีเทอร์เนตถูกเรียกว่า " ระบบฝังตัว (Embedded System) " ซึ่งจะทำการทดสอบฮาร์ดแวร์นี้โดยการเชื่อมต่อกับเซนเซอร์วัดอุณหภูมิและความชื้นและทำการวัดค่า จากนั้นเชื่อมต่อกับคอมพิวเตอร์เพื่อมาแสดงค่าที่วัดได้บนหน้าจอแสดงผล

จากผลการทดลองในบทที่ 4 จะเห็นว่า Ethernet Controller สามารถรับ-ส่งข้อมูลระหว่างเซนเซอร์กับ hardware และระหว่าง hardware กับเครื่องคอมพิวเตอร์ได้ ซึ่งโปรโตคอลที่ใช้คือโปรโตคอล UDP ซึ่งมีความรวดเร็วในการส่งเนื่องจากไม่ต้องมีการจองเส้นทางและไม่มีการตรวจสอบเฟรมข้อมูล โดยข้อมูลที่รับมาจากเซนเซอร์เป็นดิจิตอลและจะถูกแปลงให้เป็นรหัสแอสกีย์ส่งผ่าน Ethernet Controller และทำป้อนเข้าสู่โปรแกรมที่รันบนเครื่องคอมพิวเตอร์และแสดงผลออกมาเป็นตัวเลขได้เป็นองศาเซลเซียสและเปอร์เซนต์ความชื้นสัมพัทธ์ นอกจากนี้ในการทดลองการรับ-ส่งข้อมูลกันระหว่าง Ethernet Controller และคอมพิวเตอร์สามารถกระทำได้ทั้งในวง LAN เดียวกัน และคนละวง LAN ได้

ปัญหาที่พบในโครงงานชิ้นนี้คือในการทดลองได้ใช้คอมพิวเตอร์เพียงเครื่องเดียว แต่ในความเป็นจริงอาจจะมีการเชื่อมต่อคอมพิวเตอร์หลายๆเครื่องเข้าไปในระบบดังนั้นอาจจะเกิดการชนกันของข้อมูลได้ และถ้าเป็นไปได้ควรจะต้องมีการพัฒนาไปใช้โปรโตคอล TCP เพื่อให้มีการตรวจสอบเฟรมข้อมูลระหว่างการส่งตลอดเวลา ทำให้ระบบน่าจะมีความน่าเชื่อถือมากขึ้นกว่าเดิมด้วย

การประยุกต์การใช้งานสามารถนำไปพัฒนาต่ออย่างเช่น การสร้างวงจรมาควบคุมที่อาจจะเพิ่มในเรื่องของการส่งสัญญาณแจ้งเตือนผู้ดูแลระบบให้ทราบเมื่อค่าอุณหภูมิและความชื้นถึงค่าที่จำกัดหรือค่าที่ต้องการซึ่งอาจนำไปใช้ในโรงงานอุตสาหกรรมหรือพื้นที่โรงเรือนการเกษตรที่ต้องการการควบคุมสภาพแวดล้อมให้เหมาะสม นอกจากนั้นอาจนำเอาหลักการทางด้านระบบ LAN ไปเป็นแนวทางในการศึกษาระบบทางด้านเนตเวิร์กในระดับต่างๆทั้งในระดับเครือข่ายแบบย่อยๆรวมไปถึงระดับเครือข่ายองค์กรขนาดกลางและขนาดใหญ่ที่สนับสนุนการบริหารการใช้ทรัพยากรในองค์กรร่วมกัน ทำให้การใช้และปรับปรุงข้อมูลการส่งข้อมูลเป็นไปอย่างรวดเร็วและมีประสิทธิภาพมากที่สุด

หนังสืออ้างอิง

[1] K. Washburn , J.T. Evans . TCP/IP Running a Successful Network . Addision-Wesley Publishing Company ,1996.

[2] ผศ. ธีรวัฒน์ ประกอบผล . การพัฒนาไมโครคอนโทรลเลอร์ด้วยภาษาซี . สมาคมส่งเสริมเทคโนโลยี (ไทย-ญี่ปุ่น) ,2549.

[3] กิตติ ภักดีวัฒนะกุล . JAVA ฉบับพื้นฐาน . ไทยเจริญการพิมพ์ ,2541.

[4] วรรณิกา เนตรงาม . คู่มือการเขียนโปรแกรมภาษา JAVA ฉบับผู้เริ่มต้น . Infopress Developer Book ,2545.

[5] ดร. วิรินทร์ เมฆประดิษฐ์สิน . คัมภีร์ระบบเครือข่าย . ซีเอ็ด ,2540.

[6] Using the Crystal 8900A in 8-Bit Mode , www.cirrus.com/en/pubs/appNote/an181.pdf

[7] CS8900A Product Data Sheet , www.cirrus.com/en/pubs/proDatasheet/CS8900A_F2.pdf

[8] SHT-15 datasheet , http://www.silaresearch.com/data/sht15.pdf

[9] ดร.วีระศักดิ์ ชีงฉาวร . JAVA PROGRAMMING VOLUME I,II,III ซีเอ็ด ,2548.

ภาคผนวก

โปรแกรม Ethernet_Controller.java : สำหรับรับค่าและแสดงผลข้อมูลความชื้นสัมพัทธ์และอุณหภูมิบนคอมพิวเตอร์

```
//Ethernet_Controller.java
import java.io.*;
import java.lang.*;
import java.net.*;
import java.awt.*;
import javax.swing.text.*;
import javax.swing.border.*;
import java.awt.event.*;
import javax.swing.*;
public class Ethernet_Controller extends JFrame
{
      private JLabel a1,b1,c1,d1,e1,f1,g1;
      private JTextField a2,b2,c2,d2,e2;
      private JTextArea f2,g2;
      private JPanel p0,p1,p2,p3;
      private GridLayout grid;
      private DatagramSocket socket;
      private DatagramPacket recievePacket;
      private InetAddress ClientIP;
      private int ClientPort;
      public Ethernet_Controller()
      {
      super( "HUMIDITY  AND  TEMPERATURE  MEASURING  CONTROLLER" );
            grid = new GridLayout(4,1,1,1);

            Container c = getContentPane();
            c.setLayout( grid );

            p0=new JPanel();
            p0.setLayout( new FlowLayout() );
            a1=new JLabel(" Computer IP ");
            p0.add(a1);
            a2=new JTextField("",9);
            a2.setEditable(false);
            p0.add(a2);
            b1=new JLabel("  Port ");
            p0.add(b1);
            b2=new JTextField(" 6000",3);
            b2.setEditable(false);
            p0.add(b2);
            c.add(p0);

            p1=new JPanel();
            p1.setLayout( new FlowLayout() );
            c1=new JLabel(" Ethernet  Controller  IP ");
            p1.add(c1);
            c2=new JTextField("",9);
            c2.setEditable(false);
```

```
p1.add(c2);
d1 = new JLabel("  Port ");
p1.add(d1);
d2 = new JTextField("",3);
d2.setEditable(false);
p1.add(d2);
c.add(p1);

p2 = new JPanel();
p2.setLayout( new FlowLayout() );
e1 = new JLabel(" Status     ");
p2.add(e1);
e2 = new JTextField("",15);
e2.setEditable(false);
e2.setText("     Waiting  For  Connection       ");
p2.add(e2);
c.add(p2);

p3 = new JPanel();
p3.setLayout( new FlowLayout() );
f1 = new JLabel(" Temperature ");
p3.add(f1);
f2 = new JTextArea("",1,3);
p3.add(f2);
g1 = new JLabel(" Humidity ");
p3.add(g1);
g2 = new JTextArea("",1,3);
p3.add(g2);
c.add(p3);


try
{
InetAddress host = InetAddress.getLocalHost();
a2.setText(""+ "/" + host.getHostAddress());
}
catch(UnknownHostException e)
{
e2.setText(e.toString());
}

setLocation(240,240);
setSize(520,240);
show();
```



```
public static void main(String[] args)
{
```

```
//Create Datagram Socket
        try
        {
        socket = new DatagramSocket( 6000);
        }
        catch( SocketException socketException ) {
                    socketException.printStackTrace();
                    System.exit( 1);
        }
}

private void RxPackets()
{
while ( true )
   {
        try
        {
        byte data_rx[] = new byte[100];
        DatagramPacket receivePacket =
        new DatagramPacket( data_rx, data_rx.length );
        socket.receive( receivePacket );

        c2.setText(""+ receivePacket.getAddress());
        ClientIP = receivePacket.getAddress();
        d2.setText(""+ receivePacket.getPort());
        ClientPort = receivePacket.getPort();

        e2.setText("    Connection  Complete     ");
        f2.setText(""+ new String(receivePacket.getData(),0, 6));
        g2.setText(""+ new String(receivePacket.getData(),16, 6));
        }
        catch( IOException ioException )
        {
         e2.setText(ioException.toString());
         ioException.printStackTrace();
        }
   }
}

public static void main(String[] args)
{
        Ethernet_Controller app = new Ethernet_Controller();
        app.addWindowListener(
                    new WindowAdapter() {
                      public void windowClosing(WindowEvent e)
                      {
                         System.exit(0);
                      }
                    }
        );
        app.RxPackets();
}
}
```

โปรแกรมการรับค่าข้อมูลความชื้นและอุณหภูมิจากเซนเซอร์ SHT-15 ในไมโครคอนโทรลเลอร์ตัวที่ 1

//main.c

```
#include <REGX55.H>  //Microcontroller specific library,port definitions
#include <intrins.h> //Keil library (is used for _nop()_ operation)
#include <math.h>    //Keil library
#include <stdio.h>   //Keil library
#include <SHT_15.h>  //Command for SHT-15
#include <lcd.h>     //Display LCD

void main()
{ value humi_val,temp_val;
  unsigned char error,checksum;
  unsigned int i;

  lcd_init();
  led = 0;
  s_connectionreset();
  lcd_puts(0x80,"  TEMPERATURE    ");      // Show Display before value
  lcd_puts(0xC0,"   &  HUMIDITY    ");     // Show Display before value
  delay(250);
  lcd_puts(0x80,"    MEASURING    ");      // Show Display before value
  lcd_puts(0xC0,"THROUGH ETHERNET");       // Show Display before value
  delay(250);
  lcd_puts(0x80," Temp :    .  C ");       // Show temperature value
  lcd_puts(0xC0,"Humid :    .  % ");       // Show humidity value
  while(1)
  {
  led = 0;
  error=0;

  //---measure humidity----//
  error+=s_measure((unsigned char*) &humi_val.i,&checksum,HUMI);
  //---- measure temperature----//
  error+=s_measure((unsigned char*) &temp_val.i,&checksum,TEMP);

  if(error!=0) s_connectionreset(); //if error: connection reset
  else
    { humi_val.f=(float)humi_val.i; //converts integer to float
      temp_val.f=(float)temp_val.i; //converts integer to float

      //----calculate humidity, temperature---//
      calc_sth15(&humi_val.f,&temp_val.f);
      led = 1;
      delay(7);
    }
       for (i=0;i<800;i++);
    //-------------------------------------------------------------//
  }
}
```

```
//SHT_15.h
typedef union
{ unsigned int i;
  float f;
} value;

enum {TEMP,HUMI};
sbit  DATA    = P1^0 ;
sbit  SCK     = P1^1 ;
sbit  led     = P1^2;

unsigned char d = 0;
unsigned char p = 0;

#define noACK 0
#define ACK   1
                //    adr           command         r/w
#define MEASURE_TEMP 0x03   //000     0001      1
#define MEASURE_HUMI 0x05   //000     0010      1
//-----------------------------------------------------//


//---writes a byte on the bus and checks the acknowledge---//
char s_write_byte(unsigned char value)

{
  unsigned char i,error=0;
  for (i=0x80;i>0;i/=2)       //shift bit for masking
  {
    if (i & value) DATA=1;    //masking value with i ,write to BUS
    else DATA=0;
    SCK=1;                    //clk for BUS
    _nop_();_nop_();_nop_(); //pulswidth approx. 5 us
    SCK=0;
  }
  DATA=1;                     //release DATA-line
  SCK=1;                      //clk #9 for ack
  error=DATA;                 //check ack (DATA will be pulled down by SHT15)
  SCK=0;
  return error;               //error=1 in case of no acknowledge
}
//-----------------------------------------------------------------//

//---reads a byte form the bus and gives an acknowledge if "ack=1"---//

char s_read_byte(unsigned char ack)
{
  unsigned char i,val=0;
  DATA=1;                           //release DATA
  for (i=0x80;i>0;i/=2)             //shift bit for masking
  { SCK=1;                          //clk for BUS
    if (DATA) val=(val | i);        //read bit
    SCK=0;
  }
  DATA=!ack;                        //if "ack==1" pull down DATA-line
  SCK=1;                            //clk #9 for ack
  _nop_();_nop_();_nop_();          //pulswidth approx. 5 us
```

```
  SCK=0;
  DATA=1;                              //release DATA-line
  return val;
}
//---------------------------------------------------------------//

//--generates a transmission start---//
void s_transstart(void)

{
   DATA=1; SCK=0;                      //Initial state
   _nop_();
   SCK=1;
   _nop_();
   DATA=0;
   _nop_();
   SCK=0;
   _nop_();_nop_();_nop_();
   SCK=1;
   _nop_();
   DATA=1;
   _nop_();
   SCK=0;
}
//---------------------------------------------------------------//

/*--- communication reset: DATA-line=1 and at least 9 SCK cycles
followed by transstart---*/

void s_connectionreset(void)
{
  unsigned char i;
  DATA=1; SCK=0;                      //Initial state
  for(i=0;i<9;i++)                    //9 SCK cycles
  { SCK=1;
    SCK=0;
  }
  s_transstart();                     //transmission start
}

//--makes a measurement (humidity/temperature) with checksum--//

char s_measure(unsigned char *p_value, unsigned char *p_checksum,
unsigned char mode)
{
  unsigned error=0;
  unsigned int i;

  s_transstart();                    //transmission start
  switch(mode){                      //send command to sensor
    case TEMP : error+=s_write_byte(MEASURE_TEMP); break;
    case HUMI : error+=s_write_byte(MEASURE_HUMI); break;
    default     : break;
  }
```

```
//--wait until sensor has finished the measurement--//
for (i=0;i<65535;i++) if(DATA==0) break;
if(DATA) error+=1;       // or timeout (~2 sec.) is reached
  *(p_value)  =s_read_byte(ACK);   //read the first byte (MSB)
  *(p_value+1)=s_read_byte(ACK);   //read the second byte (LSB)
  *p_checksum =s_read_byte(noACK); //read checksum
  return error;
}
//-----------------------------------------------------------//

//----calculates temperature [°C] and humidity [%RH]----//
//---input :  humi  (12 bit),temp  (14 bit)---//
//--output:  humi [%RH], temp [°C]--//


void calc_sth15(float *p_humidity ,float *p_temperature)
{ const float C1=-4.0;              // for 12 Bit
  const float C2=+0.0405;           // for 12 Bit
  const float C3=-0.0000028;        // for 12 Bit
  const float T1=+0.01;             // for 14 Bit @ 5V
  const float T2=+0.00008;          // for 14 Bit @ 5V
  const float D3 =-0.00000002;      // for 14 Bit @ 5V

  float rh=*p_humidity;     // rh:   Humidity  12 Bit
  float t=*p_temperature;   // t:    Temperature 14 Bit
  float rh_lin;             // rh_lin:  Humidity linear
  float rh_true;            // rh_true: Temperature compensated humidity
  float t_C;                // t_C   :  Temperature [°C]
  int t_s1,rh_s1,s_t,s_rh;

  t_C=(t*0.01)+(D3*(t-7000)*(t-7000))- 40; //calc. temperature
  rh_lin=C3*rh*rh + C2*rh + C1;            //calc. humidity
  rh_true=(t_C-25)*(T1+T2*rh)+rh_lin;      //calc. temperature
  if(rh_true>100)rh_true=100;
  if(rh_true<0.1)rh_true=0.1;
  *p_temperature=t_C;                      //return temperature [°C]
  *p_humidity=rh_true;                     //return humidity[%RH]
  inttolcd(0x89,t_C);              // Display temperature value(int)
  inttolcd(0xC9,rh_true);          // Display humidity value(int)
  //---Compute decimal---//
  t_s1 = t_C * 10;
  s_t=t_s1;
  t_s1 %=10;
  rh_s1= rh_true * 10;
  s_rh=rh_s1;
  rh_s1 %=10;
  inttolcd(0x8C,t_s1);             //Display decimal of temperature
  inttolcd(0xCC,rh_s1);            //Display decimal of humidity
  while(d==0)
  {
      {
      for (p=0;p<10;p++);
      }
      d=d+1;
  }
  serial(s_t,s_rh);                //send data to MCS-51 #2
}
```

```
//lcd.h
/*-------------------------------------------------------------*/
#define lcd_clear() lcd_command(1) //function clear display LCD
#define lcd_origin() lcd_command(2)//function set origin LCD
#define NUMBER_OF_DIGITS 16        //function convert integer
sbit e  = P3^6;                    // Define P3.6 for Enable
sbit rs = P3^7;                    // Define P3.7 for RS

/***** Function for delay time ms scale ****/
void delay(unsigned int ms)
{
      unsigned int x,a;            // Keep for counter loop
      for(x=0;x<ms;x++)
      {
      for(a=0;a<908;a++);  // Loop for delay 1 millisec per unit
      }
}

/*********** Send command 1 byte to LCD *********/
void lcd_command(unsigned char com)
{
      rs = 0;
      e = 1;
      P0 = com;
      delay(1);
      e = 0;
      delay(1);
}

/********** Send character 1 byte to LCD ***********/
void lcd_text(unsigned char text)
{
      rs = 1;
      e = 1;
      P0 = text;
      delay(1);
      e = 0;
      delay(1);
}

/*********** Display String to LCD **************/
void lcd_puts(char addr, char *ptr)
{
      lcd_origin();              // Set origin LCD
      lcd_command(addr);         // Set address LCD
      while(*ptr)                // check 0(NULL) condition
      {
            lcd_text(*ptr);  // Send data of ptr pointer to LCD
            ptr++;           // Increase address 1 time
      }
}
```

```
/****** Convert long integer to ascii for display on LCD ******/
void _ultoa(unsigned long value, char* string, unsigned char radix)
{
      unsigned char index;                  // Counter of digit
      char buffer[NUMBER_OF_DIGITS];        // Data buffer
      index = NUMBER_OF_DIGITS;             // Load counter by digit count

      do
      {
      //----Convert configuration by radix(10 or 16)----//
      buffer[--index] = '0' + (value % radix);
      //----- For base over 10(base 16)----//
      if ( buffer[index] > '9') buffer[index] += 'A' - '9' - 1;
      value /= radix;       // Div by to calculate into next digit
      } while (value != 0);                 // End for convert?



      do
      {
      //---Load convert value to string buffer---//
      *string++ = buffer[index++];
      } while ( index < NUMBER_OF_DIGITS );   // Over of digit count?

      *string = 0;                  // Place null for end string
}

/***** Convert long integer to ascii for display on LCD ******/

void _ltoa(long value_1, char * string_1, unsigned char radix_1)
{
      if (value_1 < 0 && radix_1 == 10)    // For value < 0 (base 10)
      {
      *string_1++ = '-';                   // Load sign '-' for display
      _ultoa(-value_1, string_1, radix_1);// Convert long integer
      }
      else
      {
      _ultoa(value_1, string_1, radix_1); // Convert long integer
      }
}

/******** Function Convert integer display on LCD *****/
void inttolcd(unsigned char posi, int value)
{
      char buff[12];                    // For keep string send to LCD
      _ltoa(value,&buff[0],10);         // Convert value base 10
      lcd_puts(posi,&buff[0]);          // Send integer to LCD
}
```

```
/********* Function Initial LCD ******************/
void lcd_init()
{
      delay(100);              // Delay for initial LCD
      lcd_command(0x38);       // on display ,8 bit display ,5*7 dot
      lcd_command(0x0C);       // None cursor
      lcd_command(0x01);       // Clear screen
}

void delay1()
{
      int a;
      for(a=0;a<75;a++);
}

/******** Function Send serral***************/
serial(int s_1  , int s_2)
{
  unsigned char s_t1,s_t2, s_rh1,s_rh2;
  unsigned char text1[2];
  unsigned char k,f =0 ;
  unsigned char t = 0x30;

      //---convert to ASCII---//
      s_t1 = s_1/100;
      text1[2] = s_t1+t;
      s_1 = s_1 % 100;
      s_t2 = s_1/10;
      text1[3] = s_t2+t;
      s_t2 = s_1%10;
      text1[4] = s_t2+t;
      s_rh1 = s_2/100;
      text1[5] = s_rh1+t;
      s_2 = s_2 % 100;
      s_rh2 = s_2/10;
      text1[6] = s_rh2+t;
      s_rh2 = s_2%10;
      text1[7] = s_rh2+t;

      PCON |= 0x80;
      TMOD |= 0x20;            // Timer1 Mode2(8 bit auto reload)
      SCON |= 0x40;            // Mode serial port TX data(none RX)
      TH1 = 0xFD;              // Set19200 bps Timer1 default
      TF1 = 0;                 // Clear bit over flag Timer1
      TI = 0;                  // Clear bit over flag TX
      TR1 =1;                  // Start Timer1
      text1[0]= 0x01;          //Header#1
      text1[1]= 0x10;          //Header#2

      for(k=0;k<8;k++)         // For loop send data to serial port
      {
            SBUF = text1[k];         // TX data to serial by array type
            while(~TI);              // Wait for send data finish(TI = 1)
            TI = 0;                  // Clear bit flag TI = 0(Ready TX)
            delay1();
      }
}
```

โปรแกรมควบคุมการรับส่งข้อมูลคอมพิวเตอร์กับ Ethernet Controller ในไมโครคอนโทรลเลอร์ตัวที่ 2

//main.c

```
#include <REGX55.H>
#include <initial.h>
#include <frame_Eth.h>
#include <manual_IP.h>

main()
{
led = 0;
chk = 0;
initial();
manual_IP();
if(sel_IP_D==0)
{
      While(chk==0)
      {
      icmp_in();
      receive();
      if((Buffer[13]==0x08)&&(Buffer[14]==0x06)&&(Buffer[39]==0xA1)
      &&(Buffer[40]==0xF6)&&(Buffer[41]==0x12)&&(Buffer[42]==0x9D))
            { arp(); }
      receive();
      if((Buffer[13]==0x08)&&(Buffer[14]==0x00)&&(Buffer[24]==0x01)
      &&(Buffer[31]==IP_S[0])&&(Buffer[32]==IP_S[1])
      &&(Buffer[33]==IP_S[2])&&(Buffer[34]==IP_S[3])
      &&(Buffer[43]==0x01))
            { chk = 1; }
      }
      while(1)
      {
            led=0;
            delay(3);
            rec();
            tx_inlan();
      }
}
else
{
      While(chk==0)
      {
      icmp_cross();
      receive();
      if((Buffer[13]==0x08)&&(Buffer[14]==0x00)&&(Buffer[24]==0x01)
      &&(Buffer[31]==IP_S[0])&&(Buffer[32]==IP_S[1])
      &&(Buffer[33]==IP_S[2])&&(Buffer[34]==IP_S[3])
      &&(Buffer[43]==0x01))
            {
            chk = 1;
            }
      }
```

```
        while(1)
        {
                led=0;
                delay(3);
                rec();
                tx_crosslan();
        }
}
}
//-------------------------------------------------------//
```

//initial.h

```
#define ADDR P2
#define DATA P0

/*
                 7     6     5     4     3     2     1     0
                +-----+-----+-----+-----+-----+-----+-----+-----+
        addr    | N/A | N/A | N/A | N/A |  A3 |  A2 | A1  |  A0 |
                +-----+-----+-----+-----+-----+-----+-----+-----+
*/

//---------------------------------------//

#define RxTxData        0x00        // Receive/Transmit data (port 0)
#define RxTxData1       0x02        // Receive/Transmit data (port 1)
#define TxCmd           0x04        // Transmit Command
#define TxLength        0x06        // Transmit Length
#define ISQ             0x08        // Interrupt status queue
#define PPPtr           0x0A        // PacketPage pointer
#define PPData          0x0C        // PacketPage data (port 0)
#define PPData1         0x0E        // PacketPage data (port 1)

//---------------------------------//
//------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;
                }
        }
//-----------------------------------------------//
```

```
sbit Write    = P1^0;          // IOW active low
sbit Read     = P1^1;          // IOR active low
sbit led      = P1^2;
sbit mux_o    = P1^3;
sbit sel_P0   = P1^4;
sbit sel_IP_D = P1^5;
sbit con_A    = P3^2;
sbit con_B    = P3^3;
sbit con_C    = P3^4;
sbit con_D    = P3^5;
sbit en_1     = P3^6;
sbit en_2     = P3^7;


//--Function Eth_Read Used to Read from the Data Bus--//
       unsigned char Eth_Read(unsigned char address)
      {
              unsigned char value;
              sel_P0 = 0;
              ADDR  = (address&0x0F);  // Get address only 4 bits
              Read  = 0;               // Active ioread
              value = DATA;            // Keep data to register
              Read  = 1;               // Deactive ioread
              return value;            // Return to ioread

}
//-------------------------------------------------//

//--Funtcion Eth_Write Used to write to the Data Bus--//
      void Eth_Write(unsigned char address, unsigned char value)
      {
              sel_P0 = 1;
              DATA  = value;
              ADDR  = (address&0x0F); // Get address only 4 bits
              Write = 0;              // Active iowrite
              Write = 1;              // Deactive iowrite
      }

//-------------------------------------------------//
void delay(unsigned int ms)
{
      unsigned int x,a;    // Keep for counter loop
      for(x=0;x<ms;x++)
      {
      for(a=0;a<908;a++); // Loop for delay 1 millisec per unit
      }
}
//--------------------------//

void delay1()
{
      unsigned char c;
      for(c=0;c<50;c++);
}

//------------------------//
```

```
void initial()
{
      //-----Initialize Registers--------- //

      //------Reset Chip------//
      // (1) Write 0x0114 to PacketPage Pointer (Data Sheet P.64)
      // (2) Write 0x0040 to PacketPage Data Port (Set bit 6)
            Eth_Write(PPPtr,       0x14);
            Eth_Write(PPPtr + 1,   0x01);
            Eth_Write(PPData,      0x40);
            Eth_Write(PPData + 1,  0x00);
      // Delay time about 125 ms for chip resetting in progress
            DELAY1ms(126);

      //---Configure Receiver Control fo Promiscious mode, RxOK---//
      // (1) Write 0x0104 to PacketPage Pointer (Data Sheet P.56)
      // (2) Write 0x2D00 to PacketPage Data Port (Set bit 8,A,B,D)
            Eth_Write(PPPtr,       0x04);
            Eth_Write(PPPtr + 1,   0x01);
            Eth_Write(PPData,      0x00);
            Eth_Write(PPData +1,   0x2D);

      //--Set 10BaseT, SerRxOn, SerTxOn in Line Control--//
      // (1) Write 0x0112 to PacketPage Pointer (Data Sheet P.62)
      // (2) Write 0x00C0 to PacketPage Data Port (Set bit 6,7)
            Eth_Write(PPPtr,       0x12);
            Eth_Write(PPPtr + 1,   0x01);
            Eth_Write(PPData,      0xC0);
            Eth_Write(PPData + 1,  0x00);


      /-- MAC Adrress for Ethernet Controller(00-11-D8-67-1F-1D)--//
      //Write MAC(IEEE) Address (Data Sheet P.71)
            Eth_Write(PPPtr,       0x58);
            Eth_Write(PPPtr + 1,   0x01);
            Eth_Write(PPData,      0x00);
            Eth_Write(PPData+1 ,   0x11);

            Eth_Write(PPPtr,       0x5A);
            Eth_Write(PPPtr + 1,   0x01);
            Eth_Write(PPData,      0xD8);
            Eth_Write(PPData+1 ,   0x67);

            Eth_Write(PPPtr,       0x5C);
            Eth_Write(PPPtr + 1,   0x01);
            Eth_Write(PPData,      0x1F);
            Eth_Write(PPData+1 ,   0x1D);

}
//---------------------------------------//
```

```
//frame_Eth.h
unsigned char Bus0,Bus1;
unsigned int Length;
unsigned char idata Buffer[70];
unsigned char IP_S[4];
unsigned char DataL,DataH,i,j,k,a,chk;
unsigned int  x,y;
unsigned char text1[6];
unsigned long cal_chk[2];
unsigned char buff,chk_s[2];
unsigned long temp[4];

void rec()
{
      unsigned char dat0 = 0; // For keep data TX/RX
      unsigned char dat1 = 0; // For keep data TX/RX
      PCON |= 0x80;
      TMOD |= 0x20;          // Timer1 Mode2(8 bit auto reload)
      SCON |= 0x50;          // Mode serial port TX/RX data
      TH1 = 0xFD;            // Set 19200 bps Timer1 default
      RI = 0;                // Clear RI flag
      TR1 =1;                // Start Timer1
      do
            {
            while(~RI);    // Wait until recive data from serial port
            RI = 0;        // Clear RI flag
            dat0 = SBUF;   // Load data keep to dat
            while(~RI);    // Wait until recive data from serial port
            RI = 0;        // Clear RI flag
            dat1 = SBUF;   // Load data keep to dat
            }
            while((dat0!=0x01)||(dat1!=0x10));


            for(a=0;a<6;a++)
            {
            while(~RI);    // Wait until recive data from serial port
            RI = 0;        // Clear RI flag
            text1[a] = SBUF;       // Load data keep to dat
            }
            led = 1;
            delay(3);
}

//----------------------------------------//

void arp()
{
      //---------Transmit  Datagram---------//


      // Send Transmit Command (Data Sheet P.69 Setbit 7,6)
            Eth_Write(TxCmd,       0xC0);
            Eth_Write(TxCmd + 1,   0x00);
```

```
// 42 bytes to be sent
// Ethernet Header(14) + IP Header(20)+ UDP Header(8)+ Data(2)
        Eth_Write(TxLength,    0x2A);
        Eth_Write(TxLength+1,  0x00);

// Check Bus status (Data Sheet P.66)
        Eth_Write(PPPtr,       0x38);
        Eth_Write(PPPtr + 1,   0x01);



// Check Bus status,Is ready for transmit (Check bit 8)
        do {
                    Bus0 = Eth_Read(PPData);
                    Bus1 = Eth_Read(PPData+1);
        }while(!(Bus1==0x01));

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


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


// Send Destination (6 bytes)
// MAC Address  computer (00-0D-61-E5-21-52)
        Eth_Write(RxTxData,    0x00);
        Eth_Write(RxTxData+1,  0x0D);
        Eth_Write(RxTxData,    0x61);
        Eth_Write(RxTxData+1,  0xE5);
        Eth_Write(RxTxData,    0x21);
        Eth_Write(RxTxData+1,  0x52);


// Send Source (6 bytes)
// MAC Address Ethernet Controller (00-11-D8-67-1F-1D)
        Eth_Write(RxTxData,    0x00);
        Eth_Write(RxTxData+1,  0x11);
        Eth_Write(RxTxData,    0xD8);
        Eth_Write(RxTxData+1,  0x67);
        Eth_Write(RxTxData,    0x1F);
        Eth_Write(RxTxData+1,  0x1D);



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


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

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

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

        // Hardware size (1 bytes)
                Eth_Write(RxTxData,    0x06);

        //Protocol size (1 byte)
                Eth_Write(RxTxData+1,  0x04);

        // Opcode (2 bytes)
        // reply 02
                Eth_Write(RxTxData,    0x00);
                Eth_Write(RxTxData+1,  0x02);

        //sender MAC hardware(6bytes)
        //(Ethernet Controller 00-11-D8-67-1F-1D)
                Eth_Write(RxTxData,    0x00);
                Eth_Write(RxTxData+1,  0x11);
                Eth_Write(RxTxData,    0xD8);
                Eth_Write(RxTxData+1,  0x67);
                Eth_Write(RxTxData,    0x1F);
                Eth_Write(RxTxData+1,  0x1D);

        // Source Address (4 bytes)
        // IP address  (Ethernet Controller)
                k=0;
                for(a=0;a<4;a++)
                {
                 if(a%2==0) k=0;
                else
                {
                k=1;
                }
                Eth_Write(RxTxData+k, IP_S[a]);
                }

        // MAC Address computer (00-0D-61-E5-21-52)
                Eth_Write(RxTxData,    0x00);
                Eth_Write(RxTxData+1,  0x0D);
                Eth_Write(RxTxData,    0x61);
                Eth_Write(RxTxData+1,  0xE5);
                Eth_Write(RxTxData,    0x21);
                Eth_Write(RxTxData+1,  0x52);


        // Destination Address (4 bytes)
        // IP address 161.246.12.158(Computer)
                Eth_Write(RxTxData,    0xA1);       // 161
                Eth_Write(RxTxData+1,  0xF6);       // 246
                Eth_Write(RxTxData,    0x12);       // 12
                Eth_Write(RxTxData+1,  0x9E);       // 158
}


//---------------------------------//
```

```
void tx_inlan()
{
      //---------Compute  Checksum-------------//
       temp[0] = IP_S[0];
       temp[0] = temp[0]<<8;
       temp[0] = temp[0]+IP_S[1];
       temp[1] = IP_S[2];
       temp[1] = temp[1]<<8;
       temp[1] = temp[1]+IP_S[3];


cal_chk[0]=0x4500+0x0032+0x4000+0x3F11+temp[0]+temp[1]+0xA1F6+0x129E;
       cal_chk[1]=cal_chk[0];
       cal_chk[1]=cal_chk[1]>>16;
       buff=cal_chk[1];
       cal_chk[0]=~(cal_chk[0]+(buff+0x01));
       chk_s[0]= cal_chk[0];
       cal_chk[0]=cal_chk[0]>>8;
       chk_s[1]= cal_chk[0];


      //-----------Transmit  Datagram------------//


      // Send Transmit Command (Data Sheet P.69 Setbit 7,6)
            Eth_Write(TxCmd,      0xC0);
            Eth_Write(TxCmd + 1,  0x00);

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

      // Check Bus status (Data Sheet P.66)
            Eth_Write(PPPtr,      0x38);
            Eth_Write(PPPtr + 1,  0x01);

      // Check Bus status,Is ready for transmit (Check bit 8)
            do {
                        Bus0 = Eth_Read(PPData);
                        Bus1 = Eth_Read(PPData+1);
            }while(!(Bus1==0x01));


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

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


      // Send Destination (6 bytes)
      // MAC Address computer (00-0D-61-E5-21-52)
            Eth_Write(RxTxData,   0x00);
            Eth_Write(RxTxData+1, 0x0D);
            Eth_Write(RxTxData,   0x61);
            Eth_Write(RxTxData+1, 0xE5);
            Eth_Write(RxTxData,   0x21);
            Eth_Write(RxTxData+1, 0x52);
```

```
// Send Source (6 bytes)
// MAC Address Ethernet Controller (00-11-D8-67-1F-1D)
        Eth_Write(RxTxData,    0x00);
        Eth_Write(RxTxData+1,  0x11);
        Eth_Write(RxTxData,    0xD8);
        Eth_Write(RxTxData+1,  0x67);
        Eth_Write(RxTxData,    0x1F);
        Eth_Write(RxTxData+1,  0x1D);

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

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

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

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

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


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

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


// Time to Live (1 byte)
// 63
        Eth_Write(RxTxData,    0x3F);

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

// Checksum IP Header (2 bytes)
        Eth_Write(RxTxData,    chk_s[1]);
        Eth_Write(RxTxData+1,  chk_s[0]);
```

```
// Source Address (4 bytes)
// IP address  (Ethernet Controller)
        k=0;
        for(a=0;a<4;a++)
        {
         if(a%2==0) k=0;
        else
        {
        k=1;
        }
        Eth_Write(RxTxData+k, IP_S[a]);
        }
        // IP address 161.246.18.158(Computer)
        Eth_Write(RxTxData,    0xA1);       // 161
        Eth_Write(RxTxData+1,  0xF6);       // 246
        Eth_Write(RxTxData,    0x12);       // 18
        Eth_Write(RxTxData+1,  0x9E);       // 158


//-----------UDP Header---------// (8 bytes)

// Source Port (2 bytes)
// Port 2000 (07D0)
        Eth_Write(RxTxData,    0x07);
        Eth_Write(RxTxData+1,  0xD0);

// Destination Port (2 bytes)
// Port 6000 (0x1770)
        Eth_Write(RxTxData,    0x17);
        Eth_Write(RxTxData+1,  0x70);

// Message Length (2 bytes)
// 30 bytes (UDP Header(8) + Data(22))
        Eth_Write(RxTxData,    0x00);
        Eth_Write(RxTxData+1,  0x1E);

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


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


// Write data text1
        Eth_Write(RxTxData,          text1[0]);
        Eth_Write(RxTxData+1,        text1[1]);
        Eth_Write(RxTxData,          '.');
        Eth_Write(RxTxData+1,        text1[2]);
        Eth_Write(RxTxData,          ' ');
        Eth_Write(RxTxData+1,        'C');
        Eth_Write(RxTxData,          ' ');
        Eth_Write(RxTxData-1,        ' ');
        Eth_Write(RxTxData,          ' ');
        Eth_Write(RxTxData+1,        ' ');
        Eth_Write(RxTxData,          ' ');
```

```
                Eth_Write(RxTxData+1,       ' ');
                Eth_Write(RxTxData,         ' ');
                Eth_Write(RxTxData+1,       ' ');
                Eth_Write(RxTxData,         ' ');
                Eth_Write(RxTxData+1,       ' ');
                Eth_Write(RxTxData,         text1[3]);
                Eth_Write(RxTxData+1,       text1[4]);
                Eth_Write(RxTxData,         '.');
                Eth_Write(RxTxData+1,       text1[5]);
                Eth_Write(RxTxData,         ' ');
                Eth_Write(RxTxData+1,       '%');
}

//--------------------------------------//

void icmp_in()
{
      /*----------Compute CheckSum----------*/
       temp[0] = IP_S[0];
       temp[0] = temp[0]<<8;
       temp[0] = temp[0]+IP_S[1];
       temp[1] = IP_S[2];
       temp[1] = temp[1]<<8;
       temp[1] = temp[1]+IP_S[3];

       cal_chk[0]=0x4500+0x0032+0x3F01+temp[0]+temp[1]+0xA1F6+0x129E;
       cal_chk[1]=cal_chk[0];
       cal_chk[1]=cal_chk[1]>>16;
       buff=cal_chk[1];
       cal_chk[0]=~(cal_chk[0]+(buff+0x01));
       chk_s[0]= cal_chk[0];
       cal_chk[0]=cal_chk[0]>>8;
       chk_s[1]= cal_chk[0];


      //---------Transmit  Datagram---------------//



      // Send Transmit Command (Data Sheet P.69 Setbit 7,6)
            Eth_Write(TxCmd,      0xC0);
            Eth_Write(TxCmd + 1,  0x00);

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

      // Check Bus status (Data Sheet P.66)
            Eth_Write(PPPtr,      0x38);
            Eth_Write(PPPtr + 1,  0x01);

      // Check Bus status,Is ready for transmit (Check bit 8)
            do {
                        Bus0 = Eth_Read(PPData);
                        Bus1 = Eth_Read(PPData+1);
            }while(!(Bus1==0x01));
```

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

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

// Send Destination (6 bytes)
// MAC Address computer (00-0D-61-E5-21-52)
        Eth_Write(RxTxData,    0x00);
        Eth_Write(RxTxData+1,  0x0D);
        Eth_Write(RxTxData,    0x61);
        Eth_Write(RxTxData+1,  0xE5);
        Eth_Write(RxTxData,    0x21);
        Eth_Write(RxTxData+1,  0x52);

// Send Source (6 bytes)
// MAC Address Ethernet Controller(00-11-D8-67-1F-1D)
        Eth_Write(RxTxData,    0x00);
        Eth_Write(RxTxData+1,  0x11);
        Eth_Write(RxTxData,    0xD8);
        Eth_Write(RxTxData+1,  0x67);
        Eth_Write(RxTxData,    0x1F);
        Eth_Write(RxTxData+1,  0x1D);

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

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

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

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

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

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

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

// Time to Live (1 byte)
// 63
        Eth_Write(RxTxData,    0x3F);
```

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

// Checksum IP Header (2 bytes)
        Eth_Write(RxTxData,    chk_s[1]);
        Eth_Write(RxTxData+1,  chk_s[0]);

// Source Address (4 bytes)
// IP address (Ethernet Controller)
        k=0;
        for(a=0;a<4;a++)
        {
         if(a%2==0) k=0;
        else
        {
        k=1;
        }
        Eth_Write(RxTxData+k, IP_S[a]);
        }

// IP address 161.246.18.158(Computer)
        Eth_Write(RxTxData,    0xA1);       // 161
        Eth_Write(RxTxData+1,  0xF6);       // 246
        Eth_Write(RxTxData,    0x12);       // 18
        Eth_Write(RxTxData+1,  0x9E);       // 158


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

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

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

// Checksum (2 bytes)
// 0xECF3
        Eth_Write(RxTxData,    0xEC);
        Eth_Write(RxTxData+1,  0xF3);
// Identifier (2 bytes)
// 0x0000
        Eth_Write(RxTxData,    0x00);
        Eth_Write(RxTxData+1,  0x00);

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

```
    //----------Data----------// (22 bytes)
    // Write Message (22 bytes)
            Eth_Write(RxTxData,     0x01);
            Eth_Write(RxTxData+1,   0x01);
            Eth_Write(RxTxData,     0x01);
            Eth_Write(RxTxData+1,   0x01);
            Eth_Write(RxTxData,     0x01);
            Eth_Write(RxTxData+1,   0x01);
            Eth_Write(RxTxData,     0x01);
            Eth_Write(RxTxData+1,   0x01);
            Eth_Write(RxTxData,     0x01);
            Eth_Write(RxTxData+1,   0x01);
            Eth_Write(RxTxData,     0x01);
            Eth_Write(RxTxData+1,   0x01);
            Eth_Write(RxTxData,     0x01);
            Eth_Write(RxTxData+1,   0x01);
            Eth_Write(RxTxData,     0x01);
            Eth_Write(RxTxData+1,   0x01);
            Eth_Write(RxTxData,     0x01);
            Eth_Write(RxTxData+1,   0x01);
            Eth_Write(RxTxData,     0x01);
            Eth_Write(RxTxData+1,   0x01);
            Eth_Write(RxTxData,     0x01);
            Eth_Write(RxTxData+1,   0x01);
}

//-------------------------------------//

void tx_crosslan()
{
    /*--------------Compute  CheckSum----------*/
     temp[0] = IP_S[0];
     temp[0] = temp[0]<<8;
     temp[0] = temp[0]+IP_S[1];
     temp[1] = IP_S[2];
     temp[1] = temp[1]<<8;
     temp[1] = temp[1]+IP_S[3];


cal_chk[0]=0x4500+0x0032+0x4000+0x3F11+temp[0]+temp[1]+0xA1F6+0x1F67;
     cal_chk[1]=cal_chk[0];
     cal_chk[1]=cal_chk[1]>>16;
     buff=cal_chk[1];
     cal_chk[0]=~(cal_chk[0]+(buff+0x01));
     chk_s[0]= cal_chk[0];
     cal_chk[0]=cal_chk[0]>>8;
     chk_s[1]= cal_chk[0];


    //------------Transmit  Datagram-------------//

    // Send Transmit Command (Data Sheet P.69 Setbit 7,6)
           Eth_Write(TxCmd,       0xC0);
           Eth_Write(TxCmd + 1,   0x00);
```

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

// Check Bus status (Data Sheet P.66)
        Eth_Write(PPPtr,       0x38);
        Eth_Write(PPPtr + 1,   0x01);

// Check Bus status,Is ready for transmit (Check bit 8)
        do {
                    Bus0 = Eth_Read(PPData);
                    Bus1 = Eth_Read(PPData+1);
        }while(!(Bus1==0x01));

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

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

// Send Destination (6 bytes)
        // MAC Address Gateway wireless (00-09-5B-D9-EF-52)
        Eth_Write(RxTxData,    0x00);
        Eth_Write(RxTxData+1,  0x09);
        Eth_Write(RxTxData,    0x5B);
        Eth_Write(RxTxData+1,  0xD9);
        Eth_Write(RxTxData,    0xEF);
        Eth_Write(RxTxData+1,  0x52);

// Send Source (6 bytes)
// MAC Address Ethernet Controller(00-11-D8-67-1F-1D)
        Eth_Write(RxTxData,    0x00);
        Eth_Write(RxTxData+1,  0x11);
        Eth_Write(RxTxData,    0xD8);
        Eth_Write(RxTxData+1,  0x67);
        Eth_Write(RxTxData,    0x1F);
        Eth_Write(RxTxData+1,  0x1D);

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

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

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

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

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

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

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

// Time to Live (1 byte)
// 63
        Eth_Write(RxTxData,    0x3F);

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

// Checksum IP Header (2 bytes)
        Eth_Write(RxTxData,    chk_s[1]);
        Eth_Write(RxTxData+1,  chk_s[0]);

// Source Address (4 bytes)
// IP address (Ethernet Controller)
        k=0;
        for(a=0;a<4;a++)
        {
         if(a%2==0) k=0;
        else
        {
        k=1;
        }
        Eth_Write(RxTxData+k, IP_S[a]);
        }
// IP address 161.246.31.103(Computer)
        Eth_Write(RxTxData,    0xA1);       // 161
        Eth_Write(RxTxData+1,  0xF6);       // 246
        Eth_Write(RxTxData,    0x1F);       // 31
        Eth_Write(RxTxData+1,  0x67);       // 103

//---------UDP Header---------------// (8 bytes)

// Source Port (2 bytes)
// Port 2000 (07D0)
        Eth_Write(RxTxData,    0x07);
        Eth_Write(RxTxData+1,  0xD0);

// Destination Port (2 bytes)
// Port 6000 (0x1770)
        Eth_Write(RxTxData,    0x17);
        Eth_Write(RxTxData+1,  0x70);

// Message Length (2 bytes)
// 30 bytes (UDP Header(8) + Data(22))
        Eth_Write(RxTxData,    0x00);
        Eth_Write(RxTxData+1,  0x1E);
```

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


        //-------------Data---------------//  (22 bytes)
        // Write data text1
                Eth_Write(RxTxData,    text1[0]);
                Eth_Write(RxTxData+1,  text1[1]);
                Eth_Write(RxTxData,    '.');
                Eth_Write(RxTxData+1,  text1[2]);
                Eth_Write(RxTxData,    ' ');
                Eth_Write(RxTxData+1,  'C');
                Eth_Write(RxTxData,    ' ');
                Eth_Write(RxTxData+1,  ' ');
                Eth_Write(RxTxData,    ' ');
                Eth_Write(RxTxData+1,  ' ');
                Eth_Write(RxTxData,    ' ');
                Eth_Write(RxTxData+1,  ' ');
                Eth_Write(RxTxData,    ' ');
                Eth_Write(RxTxData+1,  ' ');
                Eth_Write(RxTxData,    ' ');
                Eth_Write(RxTxData+1,  ' ');
                Eth_Write(RxTxData,    text1[3]);
                Eth_Write(RxTxData+1,  text1[4]);
                Eth_Write(RxTxData,    '.');
                Eth_Write(RxTxData+1,  text1[5]);
                Eth_Write(RxTxData,    ' ');
                Eth_Write(RxTxData+1,  '%');
}
//-----------------------------------//

void icmp_cross()
{
        /*-------------Compute  CheckSum------------*/
         temp[0] = IP_S[0];
         temp[0] = temp[0]<<8;
         temp[0] = temp[0]+IP_S[1];
         temp[1] = IP_S[2];
         temp[1] = temp[1]<<8;
         temp[1] = temp[1]+IP_S[3];

         cal_chk[0]=0x4500+0x0032+0x3F01+temp[0]+temp[1]+0xA1F6+0x1F67;
         cal_chk[1]=cal_chk[0];
         cal_chk[1]=cal_chk[1]>>16;
         buff=cal_chk[1];
         cal_chk[0]=~(cal_chk[0]+(buff+0x01));
         chk_s[0]= cal_chk[0];
         cal_chk[0]=cal_chk[0]>>8;
         chk_s[1]= cal_chk[0];

        //-----------Transmit  Datagram----------//
        // Send Transmit Command (Data Sheet P.69 Setbit 7,6)
                Eth_Write(TxCmd,       0xC0);
                Eth_Write(TxCmd + 1,   0x00);
```

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

// Check Bus status (Data Sheet P.66)
        Eth_Write(PPPtr,       0x38);
        Eth_Write(PPPtr + 1,   0x01);

// Check Bus status,Is ready for transmit (Check bit 8)
        do {
                    Bus0 = Eth_Read(PPData);
                    Bus1 = Eth_Read(PPData+1);
        }while(!(Bus1==0x01));

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

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

// Send Destination (6 bytes)
// MAC Address Gateway wireless (00-09-5B-D9-EF-52)
        Eth_Write(RxTxData,    0x00);
        Eth_Write(RxTxData+1,  0x09);
        Eth_Write(RxTxData,    0x5B);
        Eth_Write(RxTxData+1,  0xD9);
        Eth_Write(RxTxData,    0xEF);
        Eth_Write(RxTxData+1,  0x52);

// Send Source (6 bytes)
// MAC Address Ethernet Controller(00-11-D8-67-1F-1D)
        Eth_Write(RxTxData,    0x00);
        Eth_Write(RxTxData+1,  0x11);
        Eth_Write(RxTxData,    0xD8);
        Eth_Write(RxTxData+1,  0x67);
        Eth_Write(RxTxData,    0x1F);
        Eth_Write(RxTxData+1,  0x1D);

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

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

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

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

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

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

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

// Time to Live (1 byte)
// 63
        Eth_Write(RxTxData,    0x3F);

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

// Checksum IP Header (2 bytes)
        Eth_Write(RxTxData,    chk_s[1]);
        Eth_Write(RxTxData+1,  chk_s[0]);

// Source Address (4 bytes)
// IP address (Ethernet Controller)
        k=0;
        for(a=0;a<4;a++)
        {
         if(a%2==0) k=0;
        else
        {
        k=1;
        }
        Eth_Write(RxTxData+k, IP_S[a]);
        }
// IP address 161.246.31.103(Computer)
        Eth_Write(RxTxData,    0xA1);       // 161
        Eth_Write(RxTxData+1,  0xF6);       // 246
        Eth_Write(RxTxData,    0x1F);       // 31
        Eth_Write(RxTxData+1,  0x67);       // 103


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

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

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

// Checksum (2 bytes)
// 0xECF3
        Eth_Write(RxTxData,    0xEC);
        Eth_Write(RxTxData+1,  0xF3);
```

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

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

        //------------Data-----------// (22 bytes)
        // Write Message (22 bytes)
                Eth_Write(RxTxData,     0x01);
                Eth_Write(RxTxData+1,  0x01);
                Eth_Write(RxTxData,     0x01);
                Eth_Write(RxTxData+1,  0x01);
                Eth_Write(RxTxData,     0x01);
                Eth_Write(RxTxData+1,  0x01);
                Eth_Write(RxTxData,     0x01);
                Eth_Write(RxTxData+1,  0x01);
                Eth_Write(RxTxData,     0x01);
                Eth_Write(RxTxData+1,  0x01);
                Eth_Write(RxTxData,     0x01);
                Eth_Write(RxTxData+1,  0x01);
                Eth_Write(RxTxData,     0x01);
                Eth_Write(RxTxData+1,  0x01);
                Eth_Write(RxTxData,     0x01);
                Eth_Write(RxTxData+1,  0x01);
                Eth_Write(RxTxData,     0x01);
                Eth_Write(RxTxData+1,  0x01);
                Eth_Write(RxTxData,     0x01);
                Eth_Write(RxTxData+1,  0x01);
                Eth_Write(RxTxData,     0x01);
                Eth_Write(RxTxData+1,  0x01);
}
//--------------------------------//

void receive()
{
                do
                {
                Eth_Write(PPPtr,        0x24);
                Eth_Write(PPPtr + 1,   0x01);
                Bus0 = Eth_Read(PPData);
                Bus1 = Eth_Read(PPData+1);
                }while((Bus0==0x04)&(Bus1==0x00));
                // frame is present //
                        Eth_Write(PPPtr,        0x02);
                        Eth_Write(PPPtr + 1,   0x04);
                        DataL=Eth_Read(PPData);
                        DataH=Eth_Read(PPData+1);
                        x=DataL;
                        y=DataH;
                        y=y<<8;
                        Length=x+y;
```

```
                        j=0;
                        k=0;
                        for (i=1;i<=Length;i++)
                        {
                          Eth_Write(PPPtr,     0x04 + j);
                          Eth_Write(PPPtr + 1, 0x04);
                          Buffer[i]=Eth_Read(PPData + k);
                          if(i%2==1) k=1;
                          else
                             {
                                   j=j+2;
                                   k=0;
                             }
                        }
}

/*-----------------------------------------------*/

//manual_IP.h
void manual_IP()
{
      unsigned char buff1;

      en_2=1;
      en_1=1;
//Mux #1//
//Dip SW1//
      con_A =0;
      con_B =0;
      con_C =0;
      con_D =0;
      en_1=0;
      buff1=mux_o;
      delay1();
      en_1=1;
      IP_S[0]=IP_S[0]+(buff1);
      buff1=0;
      con_A =1;
      con_B =0;
      con_C =0;
      con_D =0;
      en_1=0;
      buff1=mux_o;
      delay1();
      en_1=1;
      IP_S[0]=IP_S[0]+(buff1<<1);
      buff1=0;
      con_A =0;
      con_B =1;
      con_C =0;
      con_D =0;
      en_1=0;
      buff1=mux_o;
      delay1();
      en_1=1;
      IP_S[0]=IP_S[0]+(buff1<<2);
      buff1=0;
```

```
        con_A =1;
        con_B =1;
        con_C =0;
        con_D =0;
        en_1=0;
        buff1=mux_o;
        delay1();
        en_1=1;
        IP_S[0]=IP_S[0]+(buff1<<3);
        buff1=0;
        con_A =0;
        con_B =0;
        con_C =1;
        con_D =0;
        en_1=0;
        buff1=mux_o;
        delay1();
        en_1=1;
        IP_S[0]=IP_S[0]+(buff1<<4);
        buff1=0;
        con_A =1;
        con_B =0;
        con_C =1;
        con_D =0;
        en_1=0;
        buff1=mux_o;
        delay1();
        en_1=1;
        IP_S[0]=IP_S[0]+(buff1<<5);
        buff1=0;
        con_A =0;
        con_B =1;
        con_C =1;
        con_D =0;
        en_1=0;
        buff1=mux_o;
        delay1();
        en_1=1;
        IP_S[0]=IP_S[0]+(buff1<<6);
        buff1=0;
        con_A =1;
        con_B =1;
        con_C =1;
        con_D =0;
        en_1=0;
        buff1=mux_o;
        delay1();
        en_1=1;
        IP_S[0]=IP_S[0]+(buff1<<7);
        IP_S[0]=~IP_S[0];


//Dip SW2//
        buff1=0;
        con_A =0;
        con_B =0;
        con_C =0;
```

```
con_D =1;
en_1=0;
buff1=mux_o;
delay1();
en_1=1;
IP_S[1]=IP_S[1]+(buff1);
buff1=0;
con_A =1;
con_B =0;
con_C =0;
con_D =1;
en_1=0;
buff1=mux_o;
delay1();
en_1=1;
IP_S[1]=IP_S[1]+(buff1<<1);
buff1=0;
con_A =0;
con_B =1;
con_C =0;
con_D =1;
en_1=0;
buff1=mux_o;
delay1();
en_1=1;
IP_S[1]=IP_S[1]+(buff1<<2);
buff1=0;
con_A =1;
con_B =1;
con_C =0;
con_D =1;
en_1=0;
buff1=mux_o;
delay1();
en_1=1;
IP_S[1]=IP_S[1]+(buff1<<3);
buff1=0;
con_A =0;
con_B =0;
con_C =1;
con_D =1;
en_1=0;
buff1=mux_o;
delay1();
en_1=1;
IP_S[1]=IP_S[1]+(buff1<<4);
buff1=0;
con_A =1;
con_B =0;
con_C =1;
con_D =1;
en_1=0;
buff1=mux_o;
delay1();
en_1=1;
IP_S[1]=IP_S[1]+(buff1<<5);
buff1=0;
```

```
        con_A =0;
        con_B =1;
        con_C =1;
        con_D =1;
        en_1=0;
        buff1=mux_o;
        delay1();
        en_1=1;
        IP_S[1]=IP_S[1]+(buff1<<6);
        buff1=0;
        con_A =1;
        con_B =1;
        con_C =1;
        con_D =1;
        en_1=0;
        buff1=mux_o;
        delay1();
        en_1=1;
        IP_S[1]=IP_S[1]+(buff1<<7);
        IP_S[1]=~IP_S[1];

//Mux #2//
//Dip SW3//
        buff1=0;
        con_A =0;
        con_B =0;
        con_C =0;
        con_D =0;
        en_2=0;
        buff1=mux_o;
        delay1();
        en_2=1;
        IP_S[2]=IP_S[2]+(buff1);
        buff1=0;
        con_A =1;
        con_B =0;
        con_C =0;
        con_D =0;
        en_2=0;
        buff1=mux_o;
        delay1();
        en_2=1;
        IP_S[2]=IP_S[2]+(buff1<<1);
        buff1=0;
        con_A =0;
        con_B =1;
        con_C =0;
        con_D =0;
        en_2=0;
        buff1=mux_o;
        delay1();
        en_2=1;
        IP_S[2]=IP_S[2]+(buff1<<2);
        buff1=0;
        con_A =1;
        con_B =1;
        con_C =0;
```

```
        con_D =0;
        en_2=0;
        buff1=mux_o;
        delay1();
        en_2=1;
        IP_S[2]=IP_S[2]+(buff1<<3);
        buff1=0;
        con_A =0;
        con_B =0;
        con_C =1;
        con_D =0;
        en_2=0;
        buff1=mux_o;
        delay1();
        en_2=1;
        IP_S[2]=IP_S[2]+(buff1<<4);
        buff1=0;
        con_A =1;
        con_B =0;
        con_C =1;
        con_D =0;
        en_2=0;
        buff1=mux_o;
        delay1();
        en_2=1;
        IP_S[2]=IP_S[2]+(buff1<<5);
        buff1=0;
        con_A =0;
        con_B =1;
        con_C =1;
        con_D =0;
        en_2=0;
        buff1=mux_o;
        delay1();
        en_2=1;
        IP_S[2]=IP_S[2]+(buff1<<6);
        buff1=0;
        con_A =1;
        con_B =1;
        con_C =1;
        con_D =0;
        en_2=0;
        buff1=mux_o;
        delay1();
        en_2=1;
        IP_S[2]=IP_S[2]+(buff1<<7);
        IP_S[2]=~IP_S[2];

//Dip SW4//
        buff1=0;
        con_A =0;
        con_B =0;
        con_C =0;
        con_D =1;
        en_2=0;
        buff1=mux_o;
        delay1();
```

```
en_2=1;
IP_S[3]=IP_S[3]+(buff1);
buff1=0;
con_A =1;
con_B =0;
con_C =0;
con_D =1;
en_2=0;
buff1=mux_o;
delay1();
en_2=1;
IP_S[3]=IP_S[3]+(buff1<<1);
buff1=0;
con_A =0;
con_B =1;
con_C =0;
con_D =1;
en_2=0;
buff1=mux_o;
delay1();
en_2=1;
IP_S[3]=IP_S[3]+(buff1<<2);
buff1=0;
con_A =1;
con_B =1;
con_C =0;
con_D =1;
en_2=0;
buff1=mux_o;
delay1();
en_2=1;
IP_S[3]=IP_S[3]+(buff1<<3);
buff1=0;
con_A =0;
con_B =0;
con_C =1;
con_D =1;
en_2=0;
buff1=mux_o;
delay1();
en_2=1;
IP_S[3]=IP_S[3]+(buff1<<4);
buff1=0;
con_A =1;
con_B =0;
con_C =1;
con_D =1;
en_2=0;
buff1=mux_o;
delay1();
en_2=1;
IP_S[3]=IP_S[3]+(buff1<<5);
buff1=0;
con_A =0;
con_B =1;
con_C =1;
con_D =1;
```

```
        en_2=0;
        buff1=mux_o;
        delay1();
        en_2=1;
        IP_S[3]=IP_S[3]+(buff1<<6);
        buff1=0;
        con_A =1;
        con_B =1;
        con_C =1;
        con_D =1;
        en_2=0;
        buff1=mux_o;
        delay1();
        en_2=1;
        IP_S[3]=IP_S[3]+(buff1<<7);
        IP_S[3]=~IP_S[3];
}
```