โปรแกรมสนทนาและควบคุมโทรศัพท์ผ่านเครือข่ายอินเตอร์เน็ท

Telephony and control telephone via internet program

โดย

นายสาริษฐ์ วีรมหาวงศ์ B4406877

นางสาวอัจฉรา ปรีดามีสุข B4408109

รายงานนี้เป็นส่วนหนึ่งของวิชา 427499 โครงงานวิศวกรรมโทรคมนาคม

ประจำภาคการศึกษาที่ 3/2548

สาขาวิชาวิศวกรรมโทรคมนาคม สำนักวิชาวิศกรรมศาสตร์

มหาวิทยาลัยเทคโนโลยีสุรนารี

ปริญญานิพนธ์ปีการศึกษา 2548

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี

เรื่อง โปรแกรมสนทนาโทรศัพท์ผ่านเครือข่ายอินเตอร์เน็ท

ผู้จัดทำ

1.นายสาริษฐ์ วีรมหาวงศ์ B4406877

2.นางสาวอัจฉรา ปรีดามีสุข B4408109

อาจารย์ที่ปรึกษา

(อาจารย์ ดร.ชุติมา พรหมมาก)

โครงงาน โปรแกรมสนทนาและควบคุมโทรศัพท์ผ่านเครือข่ายอินเตอร์เน็ท
Telephony and control telephone via internet program

ผู้ดำเนินงาน นายสาริษฐ์ วีรมหาวงศ์ B4406877
นางสาวอัจฉรา ปรีดามีสุข B4408109
อาจารย์ที่ปรึกษา อาจารย์ ดร.ชุติมา พรหมมาก
สาขาวิชา วิศวกรรมโทรคมนาคม
ภาคการศึกษา 3/2548

บทคัดย่อ

เมื่ออินเทอร์เน็ตมีการใช้งานกว้างขวางขึ้น ความต้องการประยุกต์แบบใหม่ ๆ บนอินเทอร์เน็ตจึงได้รับการพัฒนา โดยเฉพาะอย่างยิ่งการใช้อินเทอร์เน็ตเป็นโครงสร้างพื้นฐานเพื่อรองรับการสื่อสารรูปแบบต่าง ๆ เช่น การใช้โทรศัพท์บนเครือข่าย การติดต่อด้วยเสียง ระบบวิดีโอคอนเฟอเรนซ์ การกระจายสัญญาณเสียงหรือภาพบนเครือข่าย และสิ่งหนึ่งที่มีการพัฒนาการ ประยุกต์จนสามารถใช้งานได้ดี คือระบบการสื่อสารด้วยเสียงผ่านเครือข่าย IP การสื่อสารโดยใช้เทคโนโลยีนี้จะทำให้เราสามารถประหยัดค่าใช้จ่ายในการติดต่อสื่อสาร ช่วยประหยัดค่าโทรศัพท์ทางไกล ซึ่งก็เป็นที่มาของโครงงานนี้คือ จะเป็นการสร้างการควบคุมเครื่องโทรศัพท์ผ่านระบบอินเตอร์เน็ต หรือจะเป็นระบบเครือข่ายเน็ตเวิร์ค แนวความคิดเบื้องต้นคือ ต้องการควบคุมเครื่องโทรศัพท์โดยใช้อินเตอร์เน็ตเป็นสื่อกลาง และทำการส่งสัญญาณเสียงผ่านระบบอินเตอร์เน็ตด้วยเช่นกัน นั่นก็หมายความว่าเราสามารถที่จะคุยโทรศัพท์กับระบบโทรศัพท์พื้นฐานในระยะไกลโดยไม่เสียค่าทางไกล ผ่านระบบเครือข่ายอินเตอร์เน็ตและสามารถลดค่าใช้จ่ายได้ เช่นถ้าเราเดินทางไปต่างประเทศเราก็สามารถโทรศัพท์ผ่านระบบอินเตอร์เน็ตได้ราคาพื้นที่ที่เราได้ติดตั้งระบบของเราไว้ หรือจะนำไปประยุกต์ใช้ในงานต่างๆ เช่น สามารถให้บริการบุคคลภายนอกให้คุยโทรศัพท์กับบุคคลในระบบ
โทรศัพท์ภายในผ่านระบบอินเตอร์เน็ทได้

กิตติกรรมประกาศ

การทำปริญญานิพนธ์นี้สำเร็จได้ด้วยดี เนื่องจากได้รับความอนุเคราะห์ในการให้คำปรึกษาในด้านต่างๆ ในระหว่างการดำเนินการจากบุคคลหลายท่านที่ใดให้ความช่วยเหลืออย่างดีเสมอมา อันได้แก่

ผศ.ดร.รังสรรค์ วงศ์สรรค์ หัวหน้าสาขาวิชาวิศวกรรมโทรคมนาคมและอาจารย์ที่ปรึกษาทางด้านวิชาการที่ให้คำปรึกษาในด้านต่างๆ รวมทั้งให้คำเสนอแนะที่เป็นประโยชน์ต่อพวกเราอย่างมาก

อ.ดร.ชุติมา พรหมมาก อาจารย์ที่ปรึกษาในด้านต่างๆทั้งทางวิชาการและการปฏิบัติงานและคอยดูแล ควบคุมการทำงานอย่างใกล้ชิด และคอยให้กำลังใจเสมอมา

อาจารย์สมิงที่ช่วยให้คำปรึกษาเกี่ยวกับระบบไมโครคอนโทรลเลอร์

คณาจารณ์ประจำสาขาวิชาวิศวกรรมโทรคมนาคมทุกท่าน ที่ประสิทธ์ประสาทความรู้ทุกอย่างซึ่งสามารถนำมาใช้ดำเนินจนสำเร็จ

คุณประพล จาระตะคุ หัวหน้าอาคารศูนย์เครื่องมือ 3 ที่ช่วยดูแลและติดต่อประสานงานเรื่องการเบิกจ่ายงบประมาณและเอกสารต่าง ๆ

บุคลากรสาขาวิชาวิศวกรรมโทรคมนาคมที่สนับสนุนด้านอุปกรณ์และการดำเนินงาน

สุดท้ายนี้คณะผู้จัดทำขอขอบพระคุณท่านอาจารย์และบุคลากรทุกท่านที่ให้การสนับสนุนการทำโครงงานด้วยดีตลอดมา

นายสาริษฐ์ วีรมหาวงศ์
นางสาวอัจฉรา ปรีดามีสุข

สารบัญ

สารบัญ(ต่อ)

สารบัญตาราง

สารบัญรูปภาพ

หน้า

สารบัญรูปภาพ(ต่อ)

หน้า

สารบัญรูปภาพ(ต่อ)

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มา

ในปัจจุบันนี้คงต้องยอมรับว่าอินเทอร์เนตเข้ามาบทบาทในชีวิตประจำวันของเรามากขึ้น และได้มีจำนวนผู้ใช้เพิ่มขึ้นอย่างรวดเร็ว เนื่องจากผู้ใช้สามารถค้นหาข้อมูลที่ต้องการจากแหล่งข้อมูลขนาดใหญ่และสามารถติดต่อสื่อสารกับบุคคลจากที่ต่างๆได้อย่างสะดวกรวดเร็ว เนื่องมาจากความยืดหยุ่น ทำให้สามารถพัฒนาการให้บริการต่างๆได้ง่ายกว่าเครือข่ายในระบบอื่นๆ การให้บริการอย่างหนึ่งที่กำลังได้รับความสนใจอย่างมากคือการส่งเสียงผ่านเครือข่ายแบบเรียลไทม์ การให้บริการในรูปแบบเรียลไทม์นี้ช่วยทำให้ผู้ใช้ได้รับความสะดวกสบายเป็นอย่างมาก เนื่องจากเป็นการประหยัดค่าใช้จ่ายในการติดต่อสื่อสาร ช่วยประหยัดค่าโทรศัพท์ทางไกล และเมื่อพิจารณาในแง่ผู้ให้บริการหรือผู้ดูแลเครือข่าย การพัฒนาดูแลปรับปรุงเครือข่าย สามารถทำได้ง่ายและสะดวกกว่าการติดต่อสื่อสารในรูปแบบอื่นๆ

อย่างไรก็ตามเนื่องจากเครือข่ายอินเตอร์เน็ทเป็นเครือข่ายที่ไม่รับรองคุณภาพของการให้บริการ เพราะเครือข่ายอินเตอร์เน็ททั่วไปทำการส่งข้อมูลซึ่งไม่ต้องการคุณสมบัติแบบเรียลไทม์ แต่ในการส่งข้อมูลในรูปแบบสัญญาณเสียงนั้นจำเป็นต้องใช้คุณสมบัติแบบเรียลไทม์ ดังนั้นจำเป็นต้องมีโปรโตคอลที่สามารถรองรับคุณสมบัติดังกล่าวในการส่งข้อมูล ซึ่งได้แก่ โปรโตคอลซิพ (SIP) เป็นโปรโตคอลในชั้นแอพพลิเคชั่น และใช้บริการของโปรโตคอลในชั้นแอพพลิเคชั่นและใช้บริการแอพพลิเคชั่นในชั้นที่ต่ำกว่า สามารถขอรับบริการได้ทั้ง UDP และ TCP SIP เป็นโปรโตคอลมีหน้าที่สร้างและจบการทำงานของการเชื่อมต่อ และหาตำแหน่งของเครื่องปลายทาง แต่เนื่องจากฟังก์ชั่นของ SIP ยังมีข้อจำกัดอยู่ ดังนั้นจึงต้องนำโปรโตคอลอื่นมาช่วยในการทำงานด้วยได้แก่ RTP/RTCP เพื่อให้สามารถใช้งานได้อย่างมีประสิทธิภาพมากขึ้น

1.2 วัตถุประสงค์ของโครงงาน

1.2.1 เพื่อศึกษาการพัฒนาโปรแกรมบนระบบปฏิบัติการวินโดวส์

1.2.2 เพื่อศึกษาการประยุกต์ใช้งานไมโครคอลโทรเลอร์ MCS-51

1.2.3 เพื่อศึกษาการส่งข้อมูลแบบมัลติมีเดียชนิดวอยซ์ไปบนไอพีเน็ตเวิร์ค

1.2.4 เพื่อศึกษาการทำงานของโปรโตคอลบนอินเตอร์เน็ทชนิดต่างๆ

1.3 ประโยชน์ที่คาดว่าจะได้รับ

1.3.1 ได้ศึกษาการพัฒนาโปรแกรมบนระบบปฏิบัติการวินโดวส์

1.3.2 ได้ศึกษาการส่งข้อมูลแบบมัลติมีเดีย

1.3.3 ได้สร้างโปรแกรมต้นแบบในการควบคุมโทรศัพท์พื้นฐานผ่านอินเตอร์เน็ท

1.3.4 ได้พัฒนาปรับปรุงโปรแกรมสื่อสารด้วยเสียง

1.4 ขอบเขตของโครงงาน

1.4.1 พัฒนาโปรแกรมควบคุมโทรศัพท์ผ่านทางอินเตอร์เน็ทบนระบบปฏิบัติการวินโดวส์ด้วยโปรแกรม Microsoft visual c+ .net

1.4.2 ทำการประยุกต์ใช้ไมโครคอลโทรลเลอร์ควบคุมเครื่องโทรศัพท์

1.4.3 ปรับปรุงแก้ไขโปรแกรมส่งเสียงผ่านอินเตอร์เน็ทให้ตรงกับความต้องการ

1.4.4 สามารถควบคุมการทำงานโทรศัพท์ผ่านเครือข่ายได้

1.4.5 สามารถติดต่อสื่อสารกันด้วยเสียงได้

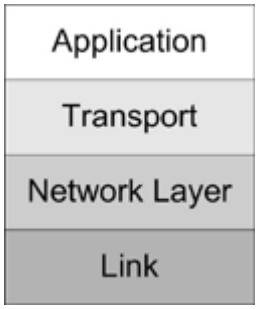
บทที่ 2
พื้นฐานโปรโตคอล TCP/IP และ VoIP

2.1 หลักการพื้นฐานของเครือข่ายอินเตอร์เน็ต

TCP/IP (Transmitsion Control Protocol/InternetProtocol) เป็นชุดของโปรโตคอลที่ถูกใช้ในการสื่อสารผ่านเครือข่ายอินเทอร์เน็ต ได้รับการพัฒนามาตั้งแต่ปี 1960 ซึ่งถูกใช้เป็นครั้งแรกในเครือข่าย ARPANET ซึ่งต่อมาได้ขยายการเชื่อมต่อไปทั่วโลกเป็นเครือข่ายอินเตอร์เน็ต ทำให้ TCP/IP เป็นที่ยอมรับอย่างกว้างขวางจนถึงปัจจุบัน

การแบ่งชั้นของ TCP/IP

TCP/IP แบ่งออกเป็น 4 เลเยอร์ ดังรูปที่ 2.1



รูปที่ 2.1 การแบ่งชั้นของ TCP/IP

ในแต่ละเลเยอร์จะมีหน้าที่ดังนี้

1. Link Layer- เลเยอร์นี้มีหน้าที่ควบคุมการรับส่งข้อมูลในระดับฮาร์ดแวร์ของเครือข่าย รับผิดชอบการรับส่งข้อมูลในระดับกายภาพ จนถึงการแปลความจากสัญญาณไฟฟ้าเป็นข้อมูลทางคอมพิวเตอร์

2. Network Layer -ทำหน้าที่รับข้อมูลจากชั้น Transport Layer และค้นหาและเลือกเส้นทางระหว่างผู้รับและผู้ส่ง เทียบได้กับ Network Layer ของ OSI Model โปรโตคอลในเลเยอร์นี้ได้แก่ IP, ICMP, IGMP

3. Transport Layer -รับผิดชอบการรับส่งข้อมูลระหว่างปลายด้านส่งและด้านรับข้อมูล และส่งข้อมูลขึ้นไปให้ Application Layer นำไปใช้งาน ต่อ เทียบได้กับ Session Layer และ Transport Layer ของ OSI Model

4. Application Layer -เป็นเลเยอร์ที่แอพพลิเคชั่นเรียกโปรโตคอลระดับล่างๆลงไป เพื่อให้บริการต่างๆ เช่น FTP, SMTP, Telnet, HTTP, POP

โครงสร้างของโปรโตคอล TCP/IP

เนื่องจาก TCP/IP เป็นชุดของโปรโตคอลประกอบด้วยโปรโตคอลหลายตัวทำงานร่วมกันในเลเยอร์ต่างๆ และมีหน้าที่แตกต่างกันออกไป ได้แก่

TCP: (Tranmission Control Protocol) - อยู่ใน Transport Layer ทำหน้าที่จัดการและควบคุมการรับส่งข้อมูล และมีกลไกความคุมการ รับส่งข้อมูลให้มีความถูกต้อง (reliable) และมีการสื่อสารอย่างเป็นกระบวนการ (connection-orient)

UDP: (User Datagram Protocol) - อยู่ใน Transport Layer ทำหน้าที่จัดการและควบคุมการรับส่งข้อมูล แต่ไม่มีกลไกความคุมการรับ ส่งข้อมูลให้มีเสถียรภาพและเชื่อถือได้ (unreliable, connectionless) โดยปล่อยให้เป็นหน้าที่ของแอพพลิเคชั่นเลเยอร์ แต่ UDP มีข้อได้เปรียบในการส่งข้อมูลได้ทั้งแบบ unicast, multicast และ broadcast อีกทั้งยังทำการติดต่อสื่อสารได้เร็วกว่า TCP เนื่องจาก TCP ต้องเสีย overhead ให้กับขั้นตอนการสื่อสารที่ทำให้ TCP มีความน่าเชื่อถือในการรับส่งข้อมูลนั่นเอง

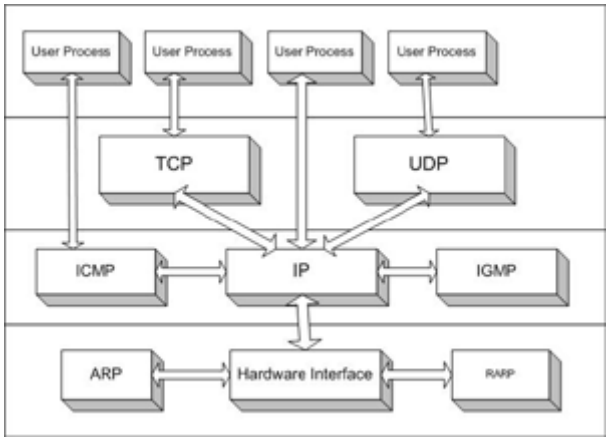
IP: (Internet Protocol) - อยู่ใน Internetwork Layer เป็นโปรตคอลหลักในการสื่อสารข้อมูล มีหน้าที่ค้นหาเส้นทางระว่างผู้รับและผู้ส่ง โดยใช้ IP Address ซึ่งมีลักษณะเป็นเลขสี่ชุด แต่ละชุดมีค่าตั้งแต่ 0-255 เช่น 172.17.3.12 ในการอ้างอิงโฮสต์ต่างๆ และกลไกการ Route เพื่อส่งต่อข้อมูลไปจนถึงจุดหมายปลายทาง

ICMP: (Internet Control Message Protocol) - อยู่ใน Internetwork Layer มีหน้าที่ส่งข่าวสารและแจ้งข้อผิดพลาดให้แก่ IP

IGMP: (Internet Group Management Protocol) - อยู่ในเน็ตเวิร์กเลเยอร์ ทำหน้าที่ในการส่ง UDP ดาต้าแกรมไปยัง กลุ่มของโฮสต์ หรือ โฮสต์หลายๆตัวพร้อมกัน

ARP: (Address Resolution Protocol) - อยู่ใน Link Layer ทำหน้าที่เปลี่ยนระหว่าง IP แอดเดรส ให้เป็นแอดเดรสของ Network Interface เรียกว่า MAC Address ในการติดต่อระหว่างกัน MAC Address คือหมายเลขประจำของ Hardware Interface ซึ่งในโลกนี้จะไม่มี MAC Address ที่ซ้ำกัน มีลักษณะเป็นเลขฐาน 16 ยาว 6 ไบต์ เช่น 23:43:45:AF: 3D:78 โดย 3 ไบต์แรกจะเป็นรหัสของผู้ผลิต และ 3 ไบต์หลังจะเป็นรหัสของผลิตภัณฑ์

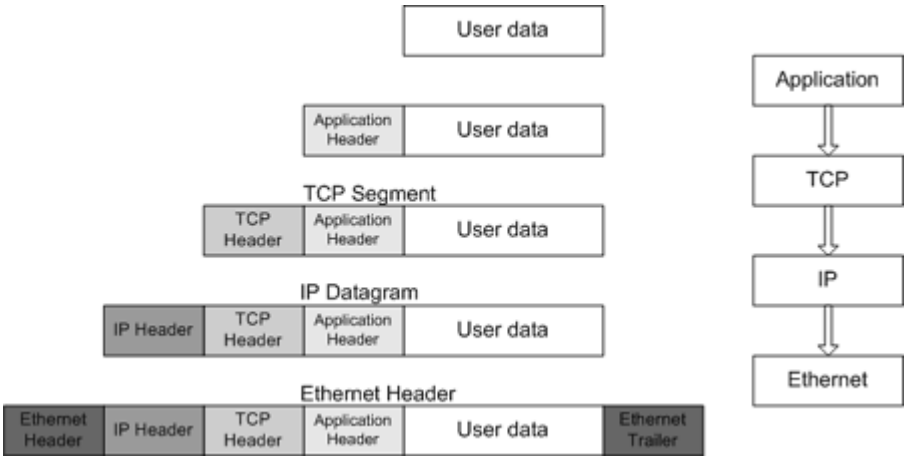
RARP: (Reverse ARP) - อยู่ในลิงค์เลเยอร์เช่นกัน แต่ทำหน้าที่กลับกันกับ ARP คือเปลี่ยนระหว่างแอดเดรสของ Network Interface ให้ เป็นแอดเดรสที่ใช้โดย IP Address



รูปที่ 2.2 แสดงให้เห็นถึงความสัมพันธ์ระหว่างโปรโตคอลต่างๆใน TCP/IP

Encapsulation/Demultiplexing

เวลาส่งข้อมูล เมื่อข้อมูลถูกส่งผ่านในแต่ละเลเยอร์ แต่ละเลเยอร์จะทำการประกอบข้อมูลที่ได้รับมา กับส่วนควบคุมซึ่งอยู่ส่วนหัวของข้อมูลเรียกว่า Header ภายใน Header จะบรรจุข้อมูลที่สำคัญของโปรโตคอลที่ทำการ Encapsulate เมื่อผู้รับ ได้รับข้อมูล ก็จะเกิดกระบวนการทำงานย้อนกลับคือ โปรโตคอลเดียวกัน ทางฝั่งผู้รับก็จะได้รับข้อมูลส่วนที่เป็น Header ก่อนและนำไปประมวลและทราบว่าข้อมูลที่ตามมามีลักษณะอย่างไร ซึ่งกระบวนการย้อนกลับนี้เรียกว่า Demultiplexing



รูปที่ 2.3 ขั้นตอนการ encapsulation เมื่อข้อมูลถูกส่งผ่านโปรโตคอลต่างๆ

ข้อมูลที่ผ่านการ Encapsulate ในแต่ละระดับมีชื่อเรียกแตกต่างกันข้อมูลที่มาจาก User หรือก็คือข้อมูลที่ User เป็นผู้ป้อนให้กับ Application เรียกว่า User Data เมื่อ Application ได้รับข้อมูลจาก user ก็จะนำมาประกอบกับส่วนหัวของ Appliction เรียกว่า Application Data และส่งต่อไปยังโปรโตคอล TCP เมื่อโปรโตคอล TCP ได้รับ Application Data ก็จะนำมารวมกับ Header ของโปรโตคอล TCP เรียกว่า TCP Segmentและส่งต่อไปยังโปรโตคอล IP เมื่อโปรโตคอล IP ได้รับ TCP Segment ก็จะนำมารวมกับ Header ของ โปรโตคอล IP เรียกว่า IP Datagramและส่งต่อไปยังเลเยอร์ Datalink Layer ในระดับ Datalink จะนำ IP Datagram มาเพิ่มส่วน Error Correction และ flag เรียกว่า Ethernet Frame ก่อนจะแปลงข้อมูลเป็นสัญญาณไฟฟ้า ส่งผ่านสายสัญญาณที่เชื่อมโยงอยู่ต่อไป

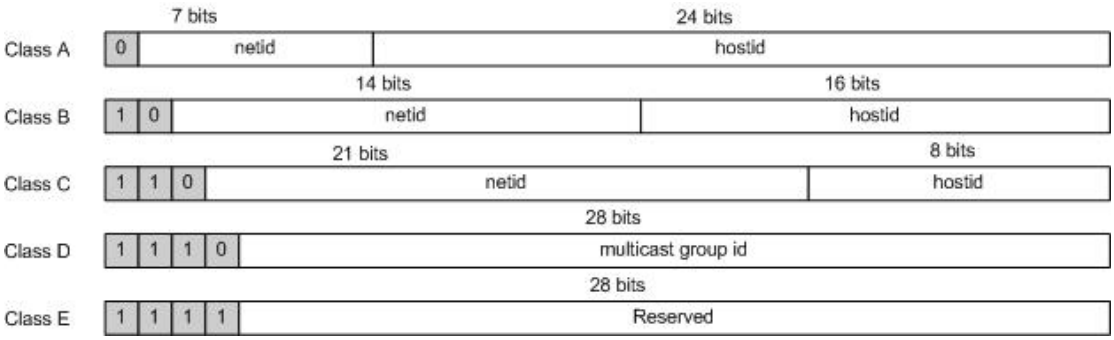
ไอพีโปรโตคอล (IP: Internet Protocol)

ด้วยเหตุที่ IP เป็นโปรโตคอลหลักในการสื่อสารข้อมูล และถือได้ว่าเป็นหัวใจสำคัญของโปรโตคอล TCP/IP จึงจะขอยกมาอธิบายก่อน เพื่อให้ง่ายต่อการอธิบายโปรตคอลตัวอื่นๆ ต่อไป ในบทนี้ท่านจะได้เรียนรู้เกี่ยวกับหน้าที่และลักษณะของโปรโตคอล IP , Internet Address , รูปร่างของ IP Header , การ Routing และ การจัดสรร IP ด้วย Subnet IP เป็นโปรโตคอลที่ทำหน้าที่รับภาระในการนำข้อมูลไปส่งยังผู้รับ ที่เชื่อมต่ออยู่ในระบบ network ซึ่งทั้งสองฝั่งอาจอยู่คนละเน็ตเวิร์คกันก็ได้ โปรโตคอลอื่นๆ ในระดับ network Layer ขึ้นไปทั้งTCP , UDP ,ICMP ต่างก็ต้องอาศัยโปรโตคอล IP ในการรับส่งข้อมูลทั้งสิ้น โปรโตคอล IP มีความสามารถในการค้นหาเส้นทางจากผู้รับไปยังผู้ส่ง มีกลไกที่ชาญฉลาดในการค้นหาเส้นทาง สามารถค้นหาเส้นทางไปถึงผู้รับได้เอง หากมีเส้นทางที่สามารถไปได้ แต่ไม่ได้ติดต่อระหว่างผู้รับกับผู้ส่งโดยตรง และไม่มีการยืนยันว่า ข้อมูลถึงผู้รับจริงหรือไม่ ทั้งนี้อาจเกิดจากหลายสาเหตุ เช่น ที่อยู่ของผู้รับไม่มีการเชื่อมต่ออยู่ในระบบ Internet กล่าวได้ว่า โปรโตคอล IP มีหน้าที่ในการค้นหาเส้นทางเท่านั้น ไม่มีการยืนยันผลสำเร็จในการส่งข้อมูล หากเกิดข้อผิดพลาดในการส่งข้อมูล แม้ว่าจะมีการส่ง icmp massage กลับมารายงานข้อผิดพลาด แต่ก็รับประกันไม่ได้อยู่ดีว่า icmp message จะกลับมาถึงเรียบร้อยหรือไม่ ด้วยเหตุนี้ จึงถือว่า IP เป็นโปรโตคอลที่ไม่มีความน่าเชื่อถือ (reliable)

IP Addressing

ทุกอินเตอร์เฟซที่ต่ออยู่บนอินเตอร์เน็ตจะต้องมีหมายเลขประจำตัวเพื่อใช้ในการสื่อสารข้อมูล เรียกว่า Internet Address หรือเรียกย่อๆว่า IP Address โดยค่า IP Address นี้จะเป็นหมายเลขจำนวน 32 บิต แต่แทนที่จะกำหนดให้เลขทั้ง 32 บิตนั้นถูกนับต่อเนื่องกันไป ก็จะใช้วิธีการแบ่ง

หมายเลขดังกล่าวออกเป็นกลุ่มของเลขขนาด 8 บิตจำนวน 4 ชุด และคั่นแต่ละชุดด้วยจุด ตัวอย่างเช่น 172.17.3.12 นอกจากนี้ใน IP Address นั้นยังถูกแบ่งออกเป็น 2 ส่วนคือ ส่วนที่ เป็นแอดเดรสของเน็ตเวิร์ก (Network ID) และส่วนที่เป็นแอดเดรสของโฮสต์ (Host ID) ซึ่งข้อมูลในส่วนนี้จะถูกใช้สำหรับ ค้นหาเส้นทางของ IP ในการที่จะขนส่งข้อมูลจากต้นทางให้ถึงปลายทางอย่างถูกต้อง เพื่อเป็นการกำหนดขนาดของเน็ตเวิร์ก สำหรับ IP Address ต่างๆดังนั้นจึงมีการจัด IP Address ในแต่ละช่วงออกเป็นคลาส (class) ต่างๆกันจาก A ถึง E เพื่อจะได้ทำการจัดสรร IP Address ได้อย่างเหมาะสมกับขนาดของเน็ตเวิร์ก



รูปที่ 2.4 การกำหนด IP Address ในคลาสต่างๆ

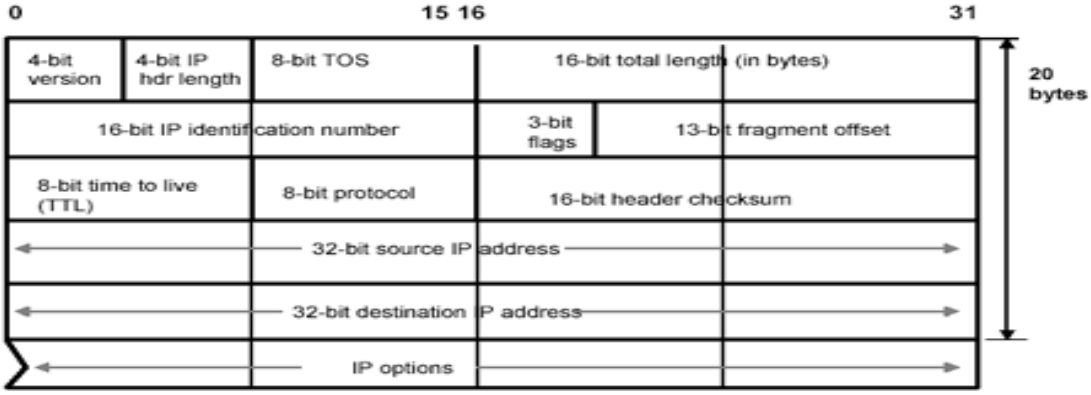
จากข้อกำหนดในการแบ่งคลาสของ IP Address หากลองนำบิตที่อยู่ในตอนต้นของ ip address ในแต่ละคลาสมาแปลงเป็น ip address ในเลขฐานสิบ จะเห็นว่าแต่ละคลาสครอบคลุม ip address ช่วงต่างๆ ดังตารางที่ 2.5

Class	IP Range
A	0.0.0.0 - 127.255.255.255
B	128.0.0.0 - 191.255.255.255
C	192.0.0.0 - 223.255.255.255
D	224.0.0.0 - 239.255.255.255
E	240.0.0.0 - 255.255.255.255

ตารางที่ 2.1 แสดงช่วงของ IP Adress ในแต่ละคลาส

IP Header

เมื่อข้อมูลถูกส่งลงมาจากชั้น Transport Layer สู่ชั้นNetwork Layer กระบวนการ Encapsulate ของ IP Protocol จะทำหารเพิ่มส่วน Header ลงไป Header ของ IP datagram มีขนาด 20-32 ไบต์ มีส่วนประกอบต่างๆ ดังแสดงในรูปที่ 2.5



รูปที่ 2.5 IP Header

ตำแหน่ง	ชื่อ	อธิบาย
0-3	Version	มีขนาด 4 บิตเป็นเวอร์ชั่นของ IP ปัจจุบันค่านี้ถูกกำหนดให้เป็น 4
4-7	length	มีขนาด 4 บิตเป็นค่าความยาวของ Header นี้ โดยปกติจะเป็น 5 หมายความว่า 5*32 บิต = 20 ไบต์
8-15	Type of Service	เป็นข้อมูลขนาด 8 บิต ปัจจุบันไม่ได้ใช้งานแล้ว
16-31	Total length	เป็นฟิลด์ที่บอกจำนวนไบต์ทั้งหมดของ IP Datagram ด้วยขนาด 16 บิตทำให้ Datagram มีขนาดสูงสุดไม่เกิน 65535 ไบต์ และมีขนาดเล็กสุดไม่ต่ำกว่า 512 ไบต์
32-47	Identification	ใช้ในกรณีที่มีการแบ่งดาต้าแกรมออกเป็นแฟรกเมนต์ เมื่อนำกลับมารวมกันใหม่จะได้รู้ว่ามาจากดาต้าแกรมเดียวกัน
48-50	Flag	ใช้ในกรณีที่มีการแบ่งข้อมูลออกเป็นแฟรกเมนต์ มีความหมาย

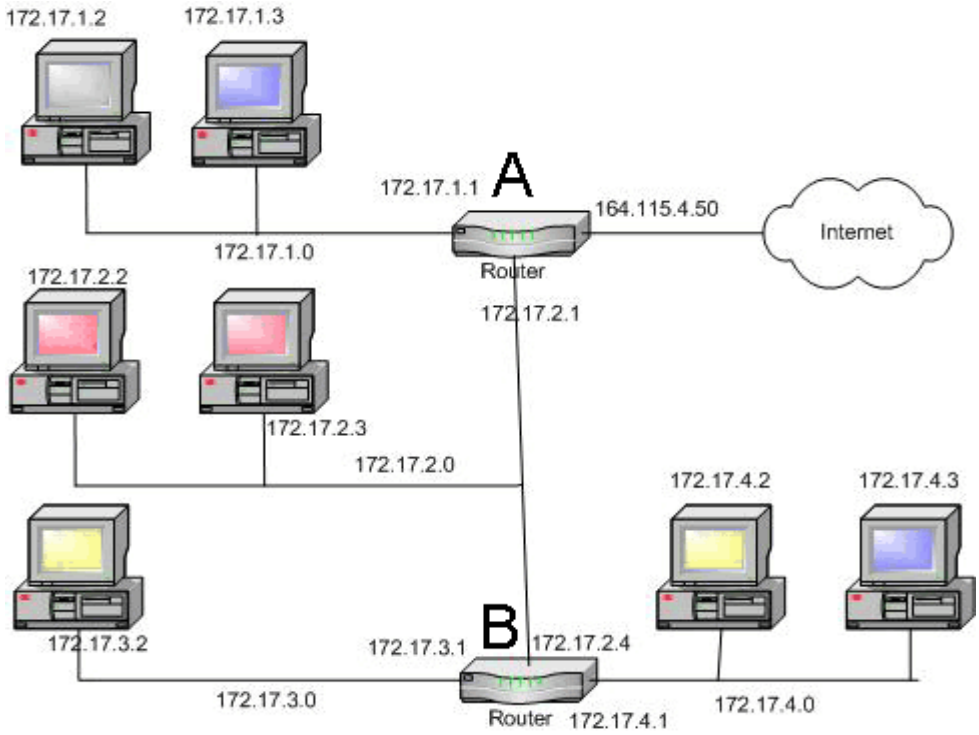
<table>
<tr><td></td><td></td><td>ดังนี้
บิต 0 : reserved เป็น 0 เสมอ
บิต 1 (DF) 0 = May Fragment, 1 = Don't Fragment
บิต 2 (MF) 0 = Last Fragment, 1 = More Fragments.</td></tr>
<tr><td>51-63</td><td>fragment offset</td><td>เป็นส่วนระบุข้อมูลที่ใช้แยกรวมข้อมูล เพื่อให้ขู้มูลที่ถูกแยกออกเป็นแฟรกเมนต์กลับมารวมกันได้อย่างถูกต้องตามลำดับ</td></tr>
<tr><td>64-71</td><td>Time to Live (TTL)</td><td>เป็นจำนวนครั้งสูงสุดที่ดาต้าแกรมนี้จะถูกส่งผ่านครือข่ายไปยังปลายทางได้ เพื่อ ป้องกันไม่ให้ดาต้าแกรมถูกเราต์ไปเรื่อยๆ อย่างไม่สิ้นสุด ปกติค่านี้จะเริ่มต้นที่ 32 และจะถูกลดค่าลงทีละ 1 เมื่อมีการเราต์ จนค่านี้มีค่าเป็น 0 ก็จะไม่ถูกเราต์อีกต่อไป</td></tr>
<tr><td>72-79</td><td>Protocol</td><td>เป็นข้อมูลที่ระบุโปรโตคอลที่ส่งดาต้าแกรมนี้มา ตัวอย่างโปรโตคอลที่ใช้บ่อยๆ ได้แก่
โปรโคอล ค่าในฟิลด์ Protocol อธิบาย
ICMP 1 Internet Comtrol Message Protocol
TCP 6 Transmission Control Protocol
UDP 17 User Datagram Protocol</td></tr>
<tr><td>80-95</td><td>Header Checksum</td><td>เป็นส่วนตรวจสอบความถูกต้องของข้อมลใน Header โดยไม่เกี่ยวกับส่วนข้อมูลที่อยู่ภายใน payload ค่านี้จะถูกคำนวนใหม่ทุกครั้งที่มีการเปลี่ยนแปลงข้อมูลใน Header (เช่น TTL ที่มีการเปลี่ยนแปลงทุกครั้งที่ IP datagram ถูกส่งผ่านเราเตอร์)</td></tr>
<tr><td>86-127</td><td>Source IP Address</td><td>คือ IP Address ของผู้ส่งดาต้าแกรม</td></tr>
<tr><td>128-163</td><td>Destination IP Address</td><td>คือ IP Address ของผู้รับดาต้าแกรม</td></tr>
<tr><td>ไม่แน่นอน</td><td>Option</td><td>มีขนาดข้อมูลไม่แน่นอน ใช้สำหรับกำหนดค่าพารามิเตอร์ปลีกย่อย ซึ่งส่วนใหญ่ไม่มีการนำไปใช้งาน</td></tr>
</table>

ขึ้นอยู่กับ Option	Padding	มีข้อมูลว่างเปล่า ใช้เป็นส่วนเติมเต็มของฟิลด์ Option ให้ครบ 32 ไบต์

ตารางที่ 2.2 อธิบาย IP Header

IP Routing

IP Routing เป็นกระบวนการค้นหาเส้นทางในการส่งผ่านข้อมูลจากต้นทางไปยังปลายทาง โดยผ่านการส่งต่อข้อมูลไปจนกว่าจะถึงปลายทาง นับเป็นกลไกสำคัญที่ทำให้ IP เป็นโปรโตคอลที่สามารถส่งข้อมูลจากโฮสต์หนึ่งไปอีกโฮสต์หนึ่งได้แม้ว่าจะอยู่ไกลแสนไกล ขอเริ่มต้นทฤษฎีของ IP Routing ด้วยการทำความรู้จักกับส่วนประกอบต่างๆ ของเน็ตเวิร์ค ในแง่ของ IP Routing กันก่อน Host โฮสต์เป็นอุปกรณ์ที่ทำหน้าที่ให้กำเนิดข้อมูลในกรณีเป็นผู้ส่ง หรือทำหน้าที่รับข้อมูลไปใช้งานในกรณีเป็นผู้รับ การสื่อสาร ข้อมูลใดๆจะต้องเป็นการสื่อสารจากโฮสต์ไปยังโฮสต์เสมอ สำหรับ IP Packet แล้วข้อมูลในเฮดเดอร์ที่ปรากฏอยู่ในฟิลด์ Source Address และ Destination Address ซึ่งเรียกว่า IP Address จะเป็นหมายเลขระบุตำแหน่งของโฮสต์ต้นทางและโฮสต์ ปลายทางเท่านั้น Network เน็ตเวิร์คเป็นเครือข่ายที่มีการเชื่อมต่อกันของโฮส 2 ตัวขึ้นไป โฮสต์แต่ละตัวในเน็ตเวิร์คเดียวกันสามารถเชื่อมต่อถึงกันได้โดยตรง Router เราเตอร์ทำหน้าที่ในการ ส่งผ่านข้อมูลจากเน็ตเวิร์กหนึ่งไปยังอีกเน็ตเวิร์กหนึ่ง ตำแหน่งของเราเตอร์จะอยู่ในจุดที่เชื่อมต่อระหว่างสองเน็ตเวิร์กเข้าด้วยกัน ด้วยข้อกำหนดของ IP ข้อมูลจะส่งไปถึงกันโดยตรงข้ามเน็ตเวิร์กไม่ได้ จะต้องอาศัยเราเตอร์เป็นผู้ทำหน้าที่ส่งผ่านข้อมูลไปให้ ใน Router จะมี Routing Table สำหรับเก็บข้อมูลเพื่อใช้ในการพิจารณาเลือกเส้นทางในการส่งดาต้าแกรม ซึ่งจะอธิบายกลไกการทำงานในหัวข้อถัดไป ในการอธิบายกระบวนการ Routing ให้เป็นที่เข้าใจในเบื้องต้น และขออธิบายจากเน็ตเวิร์คตัวอย่างที่แสดงในรูป



รูปที่ 2.6 network ตัวอย่าง

การ Routing จะเป็นไปตามขั้นตอนดังนี้

1. ถ้าโฮสต์ต้นทางและปลายทางต่อเชื่อมร่วมอยู่ในเน็ตเวิร์กเดียวกัน มีการเชื่อมต่อถึงกันโดยตรง เช่น อีเธอร์เน็ตหรือโทเค็นริง ดังแสดงในภาพที่ 2.6 เป็นการติดต่อระหว่าง 172.17.2.2 และ 172.17.2.3 (เครื่องสีแดง) IP ดาต้าแกรมก็จะถูกส่งไปยังโฮสต์ปลายทางโดยตรง

2. หากโฮสต์ต้นทางและปลายทางไม่ได้อยู่ในเน็ตเวิร์กเดียวกัน IP ดาต้าแกรมจะถูกส่งไปยังดีฟอลต์เราเตอร์

3. เมื่อเราเตอร์ได้รับ IP ดาต้าแกรมจากข้อ 2 แล้วตรวจสอบดู หากพบว่าโฮสต์ปลายทางต่อร่วมอยู่บนเน็ตเวิร์กเดียวกันกับเราเตอร์ ให้ทำการส่งดาต้าแกรมไปที่โฮสต์นั้น เช่น หาก 172.17.3.2 ต้องการส่งดาต้าแกรมไปยัง 172.17.4.2 (เครื่องสีเหลือง) จะต้องส่งดาต้าแกรมไปที่ Router B Router B จะส่งดาต้าแกรมต่อไปยังโฮสต์ปลายทาง

4. หากไม่ได้ต่อร่วมกันก็ส่งดาต้าแกรมไปที่เราเตอร์ตัวต่อไป โดย Router จะเป็นผู้เลือกเส้นทาง ซึ่งมีอยู่ 2 กรณีคือ

1. ถ้ามีข้อมูลของโฮสปลายทางอยู่ใน Routing Table Router จะส่งดาต้าแกรมไปยัง router ตัวที่ระบุไว้ใน routing table

2. ถ้าไม่มีข้อมูลของโฮสปลายทางอยู่ใน Routing Table Router จะส่งดาต้าแกรมไปยัง default router และกลับไปที่ขั้นตอนในข้อ 3 ใหม่ จนกว่า IP ดาต้าแกรมจะเดินทางถึงปลายทางหรือหมดเวลาในการส่ง (TTL=0) สมมติว่าเครื่อง 172.17.1.3 ต้องการติดต่อกับ 172.17.4.3 จะต้องส่ง ip datagram ไปยัง Router A หาก Router A มีข้อมูลเกี่ยวกับ 172.17.4.3 อยู่ ก็จะรู้ว่าต้องส่งดาต้าแกรมไปยัง Router B คือ 172.17.2.4 และ Router B ก็จะส่ง ip datagram ไปยังโฮสต์ปลายทางได้สำเร็จ Subnet Adressing / Subnet Mask ในการใช้งาน โปรโตคอล TCP/IP ใน internet นั้นการแบ่ง IP Address ออกเป็นแอดเดรสของเน็ตเวิร์ก (netid) และแอดเดรสของ โฮสต์ตามที่ระบุของแต่ละคลาสค่อนข้างจะขาดประสิทธิภาพ คือในเน็ตเวิร์กคลาส A และ B แต่ละเน็ตเวิร์กนั้น สามารถมีจำนวนโฮสต์ได้มาก ซึ่งการที่จะนำ IP Address มาใช้อย่างทั่วถึงนั้นมีโอกาสเป็นไปได้ยากมากทั้งคลาส A และคลาส B เพราะมีโอกาสน้อยมากที่จะมีเน็ตเวิร์ก ใดในโลกมีจำนวนโฮสต์มากมายขนาดนั้นอยู่ภายในเน็ตเวิร์กเดียว ดังนั้น IP Address ที่จัดสรรให้ไปในแต่ละเน็ตเวิร์กของ คลาสเหล่านี้จึงถูกใช้ไม่หมดและไม่สามารถนำไปใช้ประโยชน์ที่อื่นได้เลย ดังนั้นเพื่อให้การจัดสรร ip เป็นไปอย่างมีประสิทธิภาพ จึงมีการนำส่วนของ hostid มาแบ่งย่อยเป็นสองส่วนคือ subnet id และ host id ทำให้ได้เน็ตเวิร์คย่อยหลายๆเน็ตเวิร์ค โดยในแต่ละเน็ตเวิร์ค มีจำนวนโฮสต์ไม่มากเกินไปและเพียงพอต่อการใช้งาน การแบ่ง Subnet ใช้เทคนิคที่เรียกว่า Subnet Mask ซึ่งเป็นตัวเลขมีความยาว 32 บิต แบ่งออกเป็นสี่ชุดเช่นเดียวกับ ip แต่ค่าของ subnet mask จะขึ้นอยู่กับความต้องการในการแบ่ง subnet ว่าต้องการจำนวน subnet เท่าใดและมีจำนวนโฮสต์เท่าใด หากนำ subnet mask มาเขียนเป็นเลขฐานสอง จะมีลักษณะพิเศษคือ ขึ้นต้นด้วยเลข 1 มีจำนวนกี่ตัวก็ได้ ตามแต่ความต้องการในการแบ่ง suubnet และตำแหน่งที่เหลือจะมีค่าเป็น 0 ความสัมพันธ์ระหว่าง ip address , subnet mask , host id , net id , จำนวน subnet และ จำนวน host จะเป็นดังนี้

• host id = (ip address) AND ~(subnet mask)

• net id = (ip address) AND (subnet mask)

• จำนวน host = ((2^จำนวนบิตที่เป็น 0 ของ subnet mask)-2) เนื่องจาก ip แรกของ subnet ถูกใช้เป็น net ip และ ip สุดท้ายของ subnet ถูกใช้เป็น broadcast id

• จำนวน subnet = (2^(จำนวนบิตที่เป็น 1 ของ subnet mask ในตำแหน่งที่เป็น host id ของ ip address))

ARP: Ethernet Address Resolution Protocol

ในการสื่อสารใดๆ ก็ตาม จำเป็นต้องมีการสื่อสารผ่านตัวกลางในระดับ Physical เสมอ ซึ่งเป็นระดับล่างสุดในการสื่อสาร สำหรับในโปรโตคอล TCP/IP ถือว่าเป็นชั้น Link Layer นั่นเอง การสื่อสารในระดับนี้เป็นการสื่อสารระหว่าง Hardware Interface ในเน็ตเวิร์คเดียวกัน ซึ่งมองข้อมูลเป็น Ethernet Frame เท่านั้น จะไม่สนใจว่าข้อมูลภายในเป็นอย่างไร มีต้นทางอยู่ที่ไหนหรือปลายทางไปหาใคร แต่สิ่งที่โปรโตคอลในชั้นนี้สนใจก็คือ ข้อมูลที่ Network Layer ส่งมาให้นั้นจะต้องส่งไปยัง Hardware Interface ไหน ซึ่งการระบุ Hardware Interface จะใช้เป็น MAC Address มีลักษณะเป็นเลขฐาน 16 ยาว 6 ไบต์ เช่น 23:43:AA:5B:32:2C ซึ่งจะไม่มีอุปกรณ์ที่มีหมายเลขนี้ซ้ำกันเด็ดขาด

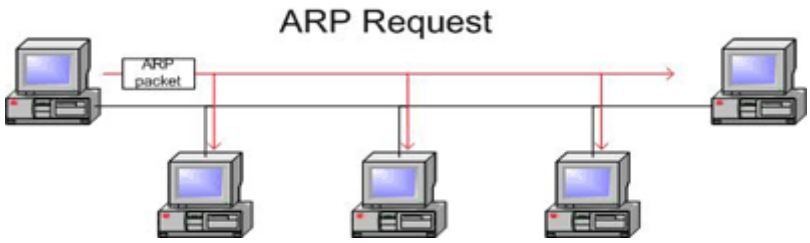
ทำไมต้องมี ARP?

ในกรณีที่มีการส่งข้อมูลจาก interface หนึ่ง ทุกๆ interface ที่อยู่ในเน็ตเวิร์คเดียวกันจะได้รับข้อมูล แต่มีเพียงอินเทอร์เฟสที่มี MAC Address ตรงกับ MAC Address ของผู้รับที่ระบุในเฟรมข้อมูลเท่านั้น ที่จะนำข้อมูลนั้นไปประมวลผล ดังนั้นในการส่งข้อมูลจากเครื่องหนึ่งไปยังอีกเครื่องหนึ่ง ฝั่งผู้ส่งจะต้องระบุ MAC Address ของผู้รับให้ถูกต้อง จึงจะสามารถส่งข้อมูลไปได้ สมมติว่า เครื่องคอมพิวเตอร์ ip 172.17.3.12 ต้องการติดต่อกับ 161.246.10.21 การทำงานในระดับ IP จะสั่งให้ ส่งข้อมูลไปยัง 172.17.3.1 ซึ่งเป็น default Router แต่ 172.17.2.12 จะรู้ได้อย่างไรว่า 172.17.2.12 มี MAC Address คืออะไร ? จุดนี้เองที่จะต้องมีการใช้ ARP ในการสอบถาม MAC Address จากเครื่องที่เราต้องการส่งข้อมูล เมื่อได้รับ MAC Address ของผู้รับมาแล้วจึงสามารถเชื่อมต่อกับ เครื่องอีกฝั่ง เพื่อการสื่อสารในระดับสูงขึ้นไปได้

กลไกการทำงานของ ARP

การทำงานของ ARP เป็นเรื่องไม่ซับซ้อน มีเพียง 2 ขั้นตอนเท่านั้นคือ

1. เครื่องที่ต้องการสอบถาม MAC Address ส่ง ARP packet เรียกว่า ARP Request ซึ่งบรรจุ IP, MAC Address ของตนเอง และ IP Address ของเครื่องที่ต้องการทราบ MAC Address ส่วน MAC Address ปลายทางนั้น จะถูกกำหนดเป็น FF:FF:FF:FF:FF:FF ซึ่งเป็น Broardcast Address เพื่อให้ ARP packet ถูกส่งไปยังเครื่องทุกเครื่องที่อยู่ในเน็ตเวิร์คเดียวกัน



รูปที่ 2.7 ARP Request จะถูกส่งไปยังเครื่องทุกเครื่องในเน็ตเวิร์ค

2. เฉพาะเครื่องที่มี IP Address ตรงกับที่ระบุใน ARP Packet จะตอบกลับมาด้วย ARP Packet เช่นกัน โดยใส่ MAC Address และ IP Address ของตนเองเป็นผู้ส่ง และใส่ MAC Address และ IP Address ของเครื่องที่ส่งมาเป็นผู้รับ packet ที่ตอบกลับนี้เรียกว่า **ARP Reply**

รูปที่ 2.8 ARP Reply จะถูกตอบกลับมาจากเครื่องที่มี IP Address เพื่อบอก MAC Address ของตนเอง

ARP Cache

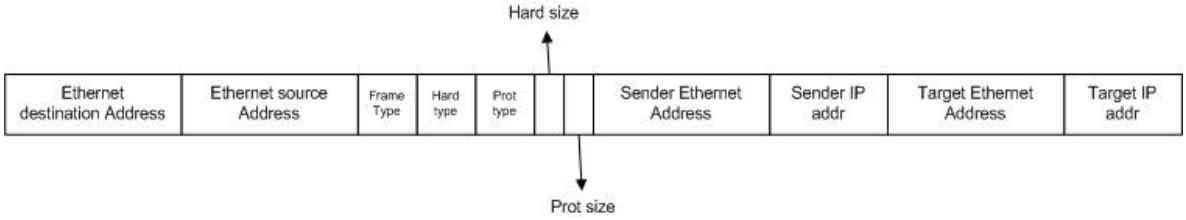
จากที่กล่าวมา จะเห็นว่ากระบวนการ ARP จะเกิดขึ้นทุกครั้งที่มีการส่ง IP datagram และกระบวการ ARP ก็กินเวลารับส่งข้อมูลและทรัพยากรในเน็ตเวิร์คพอสมควร โดยเฉพาะในจุดที่ต้องมีการ Broadcast ARP Request ซึ่งหากเป็นเช่นนั้น แบนวิธ์อันมีค่าของเน็ตเวิร์คคงหมดไปกับ ARP Packet ที่วิ่งพล่านในสายเคเบิ้ลแน่ๆ จึงมีการออกแบบบัฟเฟอร์เป็นตารางจับคู่ ระหว่าง ARP กับ IP Address เพื่อไม่ต้องส่ง ARP Request / Reply ทุกครั้งที่จะทำการส่ง IP datagram แต่ IP Address นั้นเป็นสิ่งที่ผู้ใช้กำหนดขึ้น เป็น Logical ซึ่งสามารถเปลี่ยนแปลงได้ ดังนั้น ข้อมูลในตารางนี้จึงต้องมีอายุการใช้งาน โดยทั่วไป กำหนดให้เป็นเวลา 20 นาที เมื่อหมดเวลาแล้ว หากจะส่ง IP Datagram ครั้งต่อไป จะต้องทำการส่ง ARP Request ใหม่ ท่านสามารถเรียกดู ARP cache ในระบบปฏิบัติการ Linux ได้ด้วยคำสั่ง

#ARP [Enter]

```
[root@Sandy root]# arp
Address                  HWtype  HWaddress           Flags Mask            Iface
172.17.0.1               ether   00:09:E9:95:D9:40   C                     eth0
172.17.3.28              ether   00:01:03:41:52:D5   C                     eth0
[root@Sandy root]#
```

รูปที่ 2.9 ผลของคำสั่ง arp แสดงตาราง arp cache สำหรับระบบปฏิบัติการ Linux

ARP Packet Format



รูปที่ 2.10 ARP Packet Format

ตำแหน่ง	ชื่อ	อธิบาย
ไบต์ 0-5	Ethernet Destination Address	ส่วนนี้อยู่ใน Header ของ Ethernet Frame ทั่วไป มีความหมายเป็น Address ปลายทาง ในกรณีของ ARP Request ข้อมูลในฟิลด์นี้จะเป็น FF:FF:FF:FF:FF:FF
ไบต์ 6-11	Ethernet Source Address	ส่วนนี้อยู่ใน Header ของ Ethernet Frame ทั่วไป มีความหมายเป็น Address ต้นทาง
ไบต์ 12-13	Ethernet Frame Type	ระบุถึงโปรโตคอลที่ Encapsulate อยู่ใน Ethernet Frame นี้ สำหรับ ARP จะต้องเป็น 0x0806
ไบต์ 14-15	Hard Type	ระบุถึงประเภทของ Hardware ที่โปรโตคอล ARP ถามอยู่ สำหรับกรณีนี้คือ Ethernet Address จะต้องเป็น 1
ไบต์ 16-17	Prot Type	ระบุชนิดของโปรโตคอลที่ถาม ว่าเป็น Hardware Address ของโปรโตคอลอะไร ในที่นี้คือ IP
ไบต์ 18	Hard Size	

ไบต์ 19		ระบุขนาดของแอดเดรสในโปรโตคอลที่ถาม ซึ่งมีค่าเป็น 4 สำหรับ IP
ไบต์ 20-21	OP Field	เป็นการระบุว่าเป็น ARP ชนิดใด
ไบต์ 22-27	Sender Ethernet Address	Ethernet Address ของผู้ส่ง มีค่าซ้ำกับในไบต์ที่ 6-11
ไบต์ 28-31	Sender IP Address	IP Address ของผู้ส่ง
ไบต์ 32-37	Target Ethernet Address	Ethernet Address ของผู้รับ ค่านี้จะว่างไว้ในกรณีของ ARP Request
ไบต์ 38-41	Target IP Address	IP Address ของผู้รับ

ตารางที่ 2.3 อธิบาย ARP Packet Format

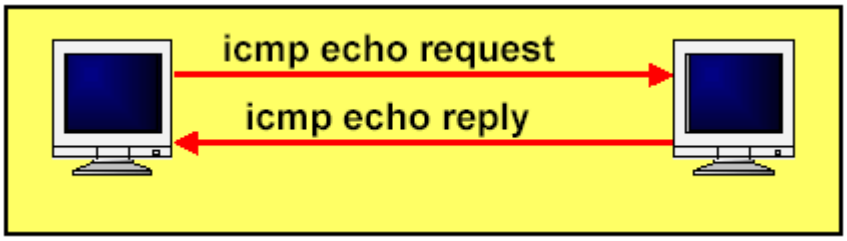
ICMP: Internet Control Message Protocol

ICMP เป็นโปรโตคอลที่ใช้ในการตรวจสอบและรายงานสถานภาพของดาต้าแกรม โปรโตคอลนี้ทำงานในระดับ Network Layer เช่นเดียวกับ IP ในกรณีที่เกิดปัญหากับดาต้าแกรม เช่น เราเตอร์ไม่สามารถส่งดาต้าแกรมไปถึงปลายทางได้ ICMP จะถูกส่งออกไปยังโฮสต้นทางเพื่อรายงานข้อผิดพลาด ที่เกิดขึ้น อย่างไรก็ดี ไม่มีอะไรรับประกันได้ว่า ICMP Message ที่ส่งไปนั้น จะถึงผู้รับจริงหรือไม่ หากมีการส่งดาต้าแกรมออกไปแล้วไม่มี ICMP Message ฟ้อง Error กลับมา ก็แปลความหมายได้สองกรณีคือ ข้อมูลถูกส่งไปถึงปลายทางอย่างเรียบร้อย หรืออาจจะมีปัญหา ในการสื่อสารทั้งการส่งดาต้าแกรม และ ICMP Message ที่ส่งกลับมาก็มีปัญหาระว่างทางก็ได้ ICMP จึงเป็นโปรโตคอลที่ไม่มีความน่าเชื่อถือ (unreliable) ซึ่งจะเป็นหน้าที่ของ โปรโตคอลในระดับสูงกว่า Network Layer ในการจัดการให้การสื่อสารนั้นๆ มีความน่าเชื่อถือ

การใช้งาน ICMP

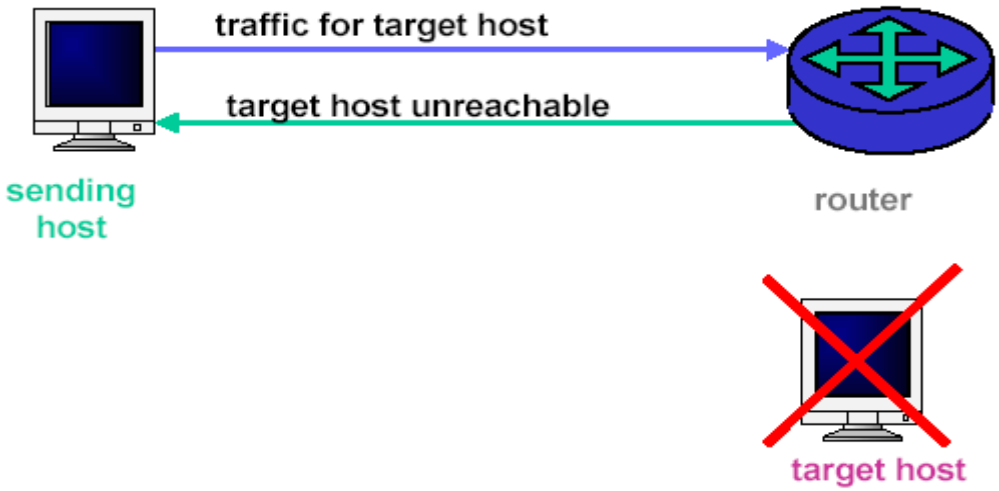
โดยทั่วไป ICMP มีการใช้งานในสองลักษณะคือ

1. Query ใช้สอบถามสถานะระหว่างกัน ในรูป เป็นการส่ง Echo request เพื่อถามสถานะของปลายทาง ซึ่งโฮสปลายทางอยู่ในสถานะปกติ สามารถทำการสื่อสารได้จะส่ง Echo Reply กลับมา



รูปที่ 2.11 การใช้งานโปรโตคอล ICMP เพื่อสอบถามสถานะระหว่างกัน

2. Error Report ใช้รายงานข้อผิดพลาดที่เกิดขึ้น เช่น หากไม่สามารถส่งดาต้าแกรมไปถึงปลายทางได้ เราเตอร์จะส่ง ICMP Message Host Unreachable กลับมารายงานโฮสต้นทาง



รูปที่ 2.12 การใช้งานโปรโตคอล ICMP เพื่อรายงานข้อผิดพลาดที่เกิดขึ้น

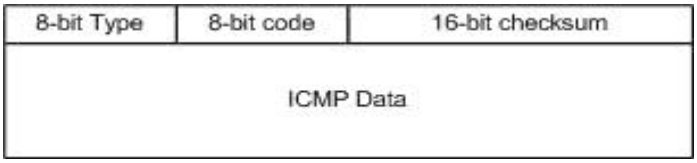
การรายงานข้อผิดพลาดของ ICMP นั้น จะ ไม่รายงานข้อผิดพลาดในบางกรณีคือ

1. เกิดความผิดพลาดในการส่ง ICMP เสียเอง
2. การส่งข้อมูลเป็นการส่งข้อมูลแบบ Multicast หรือ Broadcast

3. มีการแบ่งดาต้าแกรมออกเป็นส่วนย่อยๆ เรียกว่าแฟรกเมนต์ ในกรณีนี้ จะส่ง ICMP Message สำหรับแฟรกเมนต์แรกเพียงครั้งเดียว ไม่ต้องแจ้งกลับทุกแฟรกเมนต์ เพราะถือว่าเป็นดาต้าแกรมเดียวกัน

4. Address ต้นทางไม่ได้เจาะจงโฮส เช่น Address ที่เป็น 0 ทั้งหมด Address ที่เป็น loopback หรือ Address ที่เป็นแบบ Multicast หรือ Broadcast

ICMP Message



รูปที่ 2.13 แสดงรูปร่างของ ICMP Message

ตำแหน่ง	ชื่อ	อธิบาย
บิต 0-7	Type	ประเภทของ ICMP Message
บิต 8-15	Code	รหัสของ ICMP Message
บิต 16-31	Checksum	เป็นตัวตรวจสอบความถูกต้องของ ICMP Massage ทั้งหมด
ไม่แน่นอน	Data	ข้อมูลภายใน ICMP Message ขึ้นอยู่กับ Type *

ตารางที่ 2.4 อธิบาย ICMP Message

* ตำแหน่งต่างๆของข้อมูลภายใน Data จะขึ้นอยู่กับชนิดของ ICMP นั้นๆ ซึ่งมีรายละเอียดค่อนข้างมาก เพื่อไม่ให้เนื้อหายืดเยื้อจนเกินไป จะไม่นำมากล่าวถึงในเอกสารฉบับนี้ ท่านสามารถศึกษาโครงสร้างของ Data ของ ICMP แต่ละชนิดได้จาก RFC 792

ICMP Message Type

TYPES ของ ICMP มีความหมายดังนี้

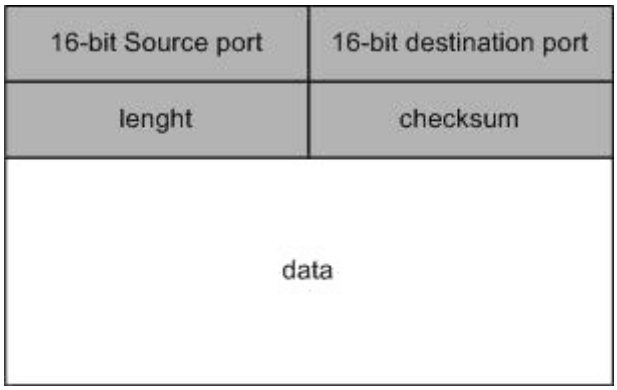
TYPE	Description
0	Echo Reply
3	Destination Unreachable
4	Source Quench
5	Redirect Message
8	Echo Request
11	Time Exceeded
12	Parameter Problem
13	Timestamp Request
14	Timestamp Reply
15	Information Request (No Longer Used)
16	Information Reply (No Longer Used)
17	Address Mask Request
18	Address Mask Reply

ตารางที่ 2.5 Description ICMP Message Type

UDP: User Datagram Protocol

UDP เป็นโปรโตคอลที่ถูกออกแบบมาให้ทำหน้าที่รับส่งข้อมูลโดยมีขั้นตอนการทำงานไม่ซับซ้อนและทำงานได้รวดเร็ว แต่มีจุดด้อยคือไม่มีความน่าเชื่อถือ (unreliable) และเป็นการสื่อสารแบบไม่ต่อเนื่อง (connectionless) โปรโตคอล UDP ทำงานในชั้น Transport Layer ซึ่งจะต้องพึ่งพาโปรโตคอล IP ในการรับส่งข้อมูล

UDP Header



รูปที่ 2.14 UDP Header

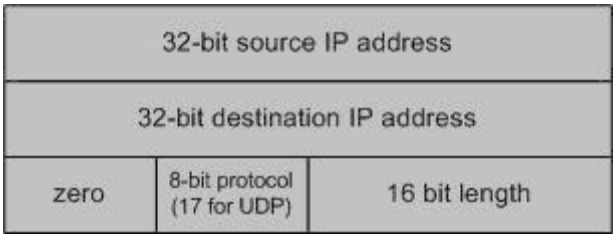
ตำแหน่ง	ชื่อ	อธิบาย
บิต 0-15	Source port number	หมายเลขพอร์ตต้นทางที่ส่งดาต้าแกรมนี้ มีความยาว 16 บิต
บิต 16-31	destination port number	หมายเลขพอร์ตปลายทางที่จะเป็นผู้รับดาต้าแกรม มีความยาว 16 บิตเช่นกัน
บิต 32-47	UDP length	ความยาวของดาต้าแกรม ทั้งส่วน Header และ data นั่นหมายความว่า ค่าที่น้อยที่สุดในฟิลด์นี้คือ 8 ซึ่งเป็นขนาดของ Header
บิต 48-63	Checksum	เป็นตัวตรวจสอบความถูกต้องของ UDP datagram และจะนำข้อมูลบางส่วนใน IP Header มาคำนวนด้วย

ตารางที่ 2.5 อธิบาย UDP Header

UDP Checksum

Checksum เป็น เลข 16 บิตถูกคำนวนด้วยวิธี one's complement โดยนำ Pseudo Header และข้อมูลทั้งหมดใน UDP Datagram มาคำนวน

Pseudo Header เป็นข้อมูลที่อยู่ในส่วนของ IP Header ประกอบด้วยฟิลด์ source ip address, destination ip address, zero, protocol, udp length ดังแสดงในรูป



รูปที่ 2.15 Pseudo Header

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

TCP: Transmistion Control Protocol

TCP เป็นโปรโตคอลที่ใช้สื่อสารระหว่างโฮสที่มีความน่าเชื่อถือ จะเห็นได้ว่าโปรโตคอลในระดับ IP หรือแม้กระทั่ง UDP จะสนใจ ข้อมูลเพียง 1 ดาต้าแกรม กลไกของโปรโตคอลจะมีหน้าที่ตรวจสอบความถูกต้องเพียงเฉพาะดาต้าแกรมนั้น ๆ เมื่อจะทำ การส่งดาต้าแกรมใหม่ก็จะถือว่าเป็นข้อมูลชุดใหม่ที่ไม่มีความสัมพันธ์ใด ๆ กับข้อมูลดาต้าแกรมอื่น (การสื่อสาร 1 ครั้ง จึงใช้เพียง 1 ดาต้าแกรม) แต่สำหรับ TCP แล้วจะเห็นว่าข้อมูลนั้นเป็น stream คือมีความสัมพันธ์ต่อเนื่องกัน มีกลไกในการตรวจสอบทั้งด้านส่ง และด้านรับเพื่อให้แน่ใจว่าสามารถสื่อสารกันได้จริงจึงจะมีการส่งรับข้อมูลเกิดขึ้น ตลอดจนการยกเลิกการติดต่อก็มีกลไกสำหรับแจ้งให้อีกฝั่งทราบ ทำให้การสื่อสารด้วย TCP จึงเสมือนว่าทั้ง 2 ฝ่ายคือฝ่ายรับและฝ่ายส่งได้ทำการต่อสาย เน็ตเวิร์กถึงกัน (connected) ตลอดเวลาที่การรับส่งข้อมูลจนกระทั่งการสื่อสารทั้งหมดเสร็จสิ้นจึงจะทำการยกเลิกการเชื่อมต่อนั้นเสีย

จุดเด่นประการสำคัญของ TCP ที่กล่าวถึงอยู่เสมอคือ ความมีเสถียรภาพและความถูกต้องของการสื่อสารซึ่งมีความเชื่อถือได้สูง คุณสมบัติที่ทำให้ TCP มีข้อดีดังกล่าวคือ

1. ข้อมูลที่จะส่งผ่าน TCP จะถูกนำมาแตกย่อยออกเป็นส่วน ๆ ให้มีขนาดเหมาะสมสำหรับการส่งข้อมูล โดย TCP มีกลไกในการพิจารณาว่าขนาดเท่าใดจะทำให้การรับ - ส่งนั้นมีประสิทธิภาพและน่าเชื่อถือสูงสุด โดยข้อมูลแต่ละชุดที่แบ่งออกและ ทำการส่งโดย TCP แต่ละครั้งจะเรียกว่า TCP เซกเมนต์

2. ในการส่งข้อมูลแต่ละครั้ง TCP จะมีการจับเวลาไว้เสมอ เพื่อรอการตอบรับจากผู้รับว่าได้รับข้อมูลถูกต้อง หากหมดเวลาแล้วไม่มีการตอบรับ TCP จะถือว่าข้อมูลไปไม่ถึงและทำการ

แก้ปัญหาที่เกิดขึ้น เช่น ยกเลิกการติดต่อ, ส่งข้อมูลซ้ำ ทำให้ Application ทราบสถานะการส่งข้อมูลตลอดเวลา

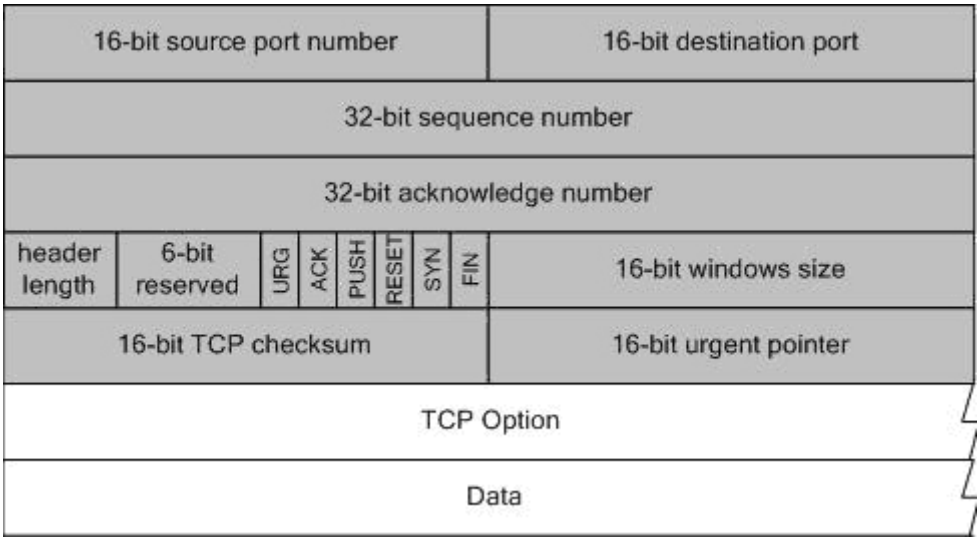
3.TCP มี checksum ซึ่งจะครอบคลุมทั้ง TCP Header และ TCP Data เพื่อเป็นการป้องกันและตรวจสอบว่าข้อมูล ที่ส่งมานั้นถูกต้อง และไม่ได้ถูกแก้ไขระหว่างทาง หาก TCP ได้รับข้อมูลที่ทำการตรวจสอบกับ checksum แล้วปรากฏว่า มีความผิดพลาดเกิดขึ้น TCP จะทิ้งข้อมูลที่ได้รับและจะไม่ทำการตอบรับข้อมูลนั้นกลับไปยังผู้ส่ง คือถือเสมือนว่าไม่ได้รับ ข้อมูลนั้น เพื่อให้ทางฝ่ายผู้ส่งทำการส่งใหม่หรือหาข้อบกพร่องและพยายามแก้ไขตามแต่แอพพลิเคชั่นทางฝ่ายผู้ส่งเห็นสมควร

4. เนื่องจาก TCP อาศัย IP ในการส่งข้อมูล ซึ่ง IP เองอาจจะถูกแฟรกเมนต์ได้ และทำให้ข้อมูลที่ถูกแฟรกเมนต์นั้นส่งถึง ปลายทางในลำดับที่ไม่ถูกต้องได้ หน้าที่ของ TCP เมื่อรับข้อมูลที่แฟรกเมนต์มานั้นจะต้องนำข้อมูลแต่ละส่วนมาประกอบ รวมกันให้ถูกต้องสมบูรณ์ก่อนจะส่งไปยัง Application Layer ต่อไป

5. การส่ง - รับข้อมูลด้วย IP อาจจะมีกรณีที่ IP Datagram นั้นถูกส่งซ้ำขึ้นได้ TCP ที่รับข้อมูลซ้ำดังกล่าวจะต้องทราบว่า เป็น IP Datagram ที่ซ้ำและไม่นำข้อมูลไปใช้งาน

6. TCP มีกลไกควบคุมการไหลของข้อมูล (Flow Control) โดยการควบคุมนี้จะต้องอาศัยลำดับของการรับส่งที่ถูกต้อง และสัมพันธ์กันทั้ง 2 ฝั่ง ในขณะเดียวกันข้อมูลที่ส่งนั้นจะต้องอาศัย IP หลายดาต้าแกรมจึงจะได้รับข้อมูลครบทั้งหมด ดังนั้นในการรับข้อมูลทางฝ่ายรับจึงต้องเตรียมบัฟเฟอร์ไว้จำนวนหนึ่งเพื่อรอรับข้อมูลและรวบรวมข้อมูลทั้งชุดให้อยู่ใน บัฟเฟอร์ก่อนที่จะทำการจัดเรียงข้อมูล ตรวจสอบความถูกต้องแล้วจึงส่งต่อไปยังแอพพลิเคชั่น ด้วยเหตุผลดังกล่าวจะเห็นได้ว่าขนาดของข้อมูลมิได้ถูกจำกัดที่ขนาดของดาต้าแกรมใด ๆ ข้อมูลที่ส่งอาจจะมีขนาดใหญ่มากอยู่ในหลายดาต้าแกรม ก็เป็นได้ ดังนั้นเพื่อป้องกันการส่งข้อมูลขนาดใหญ่เร็วเกินไปจนทำให้ทางฝ่ายรับไม่มีหน่วยความจำเพียงพอที่จะเป็น บัฟเฟอร์ที่พักข้อมูล การส่งข้อมูลจึงถูกจำกัดโดยจะอนุญาตให้ทำการส่งข้อมูลได้ เท่าที่ฝ่ายรับมีบัฟเฟอร์เพียงพอเท่านั้น

TCP Header



รูปที่ 2.16 TCP Header

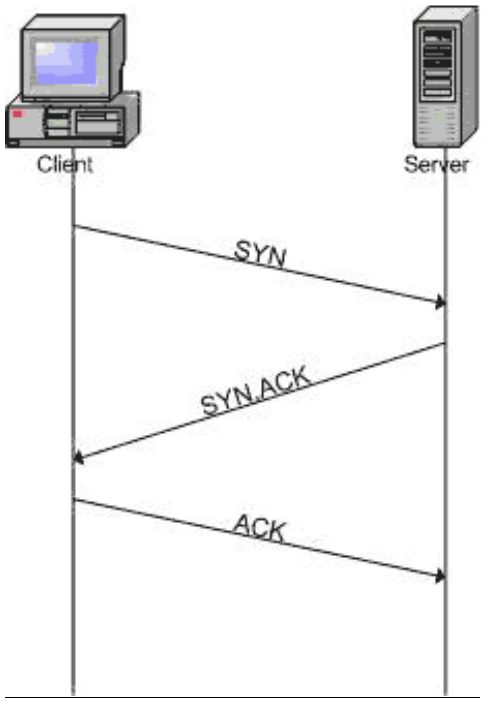
ชื่อ	อธิบาย
Source Port Number	หมายถึงพอร์ตที่โฮสต์ต้นทางใช้ในการสื่อสารกันของเซสชั่นนี้ และ TCP/IP จะใช้พอร์ตนั้นไป ตลอดตราบใดที่การสื่อสารในเซสชั่นนี้ยังไม่ยุติลง โดยทั่วไปพอร์ตนี้จะเรียกว่า "ไคลเอนต์พอร์ต" คือพอร์ตที่ไคลเอนต์เปิดขึ้น มาเพื่อรอการตอบรับจากเซิร์ฟเวอร์ (พิจารณาจากทิศทางของแพ็กเก็ตที่ส่งมาจากไคลเอนต์ไปยังเซิร์ฟเวอร์) ไคลเอนต์พอร์ต จะมีหมายเลขไม่แน่นอนและเปลี่ยนไปทุกครั้งที่มีการเริ่มการเชื่อมต่อใหม่ เป็นพอร์ตที่ถูกเปิดไว้ในระยะเวลาสั้น ๆ (ephenumeral port) ค่าที่เป็นไปได้ของพอร์ตนี้ขึ้นอยู่กับการจัดสรรของระบบปฏิบัติการ ในการกำหนดขอบเขตของพอร์ตเหล่านี้ส่วนใหญ่จะมีค่า อยู่ในช่วง 1024 – 5000
Destination Port Number	หมายถึงหมายเลขพอร์ตบนโฮสต์ปลายทางที่โฮสต์ต้นทางต้องการติดต่อด้วย โดยนัยแล้วจะ หมายถึงแอพพลิเคชั่นที่ให้บริการอยู่พอร์ตนั้นที่โฮสต์

	ปลายทางนั่นเอง พอร์ตนั้นจะเรียกอีกอย่างหนึ่งว่า "เซิร์ฟเวอร์พอร์ต" หมายเลขพอร์ตที่เปิดไว้จะขึ้นอยู่กับแอพพลิเคชั่นที่ให้บริการ โดยทั่วไปแอพพลิเคชั่นแต่ละประเภทจะมีหมายเลขพอร์ต เป็นมาตรฐานสำหรับให้ไคลเอนต์ได้เรียกใช้บริการ
Sequence Number	เป็นฟิลด์ที่ระบุถึงหมายเลขลำดับที่ใช้อ้างอิงในการสื่อสารข้อมูลแต่ละครั้ง เพื่อให้ทั้ง 2 ฝ่าย จะได้รับทราบตรงกันว่าเป็นข้อมูลของชุดใด การนำไปใช้งานจะได้ไม่ปะปนกัน และมีลำดับที่ถูกต้อง เนื่องจากการสื่อสารข้อมูลผ่าน TCP นั้นจังหวะและลำดับเป็นส่วนสำคัญของโปรโตคอลไม่ยิ่งหย่อนไปกว่าข้อมูลใน TCP Header รวมไปถึง การที่ข้อมูลในแต่ละ TCP Segment อาจจะถูกทำการแฟรกเมนต์ในเลเยอร์ของ IP ถัดลงไป ทำให้ข้อมูลถูกแบ่งออกและส่งไปในลำดับที่ไม่เรียงกัน หากไม่มีจุดอ้างอิงของข้อมูลก็จะไม่สามารถอ่านข้อมูลกลับใหม่ได้อย่างสมบูรณ์และถูกต้อง การส่งข้อมูลและการตอบรับจะใช้ฟิลด์ นี้เป็นตัวยืนยันระหว่างกันเสมอ
Acknowledge Number	ทำหน้าที่เช่นเดียวกับ Sequence Number ต่างกันตรงที่เป็น Sequence Number ซึ่งในการตอบรับ กล่าวคือ เนื่องจาก Sequence Number ที่ใช้ในการอ้างอิงนั้นผู้ที่เริ่มส่งข้อมูลจะเป็นผู้กำหนดเลขขึ้น มาและส่งไปพร้อมกับการสร้างการเชื่อมต่อครั้งใหม่แต่สำหรับฝ่ายที่ถูกติดต่อก็จำเป็นต้องกำหนดหมายเลขสำหรับใช้อ้างอิง ในการตอบรับเช่นกัน ค่าที่อยู่ใน Acknowledge Number ก็คือหมายเลขที่ใช้อ้างอิงในการตอบรับนี้
Header Length	โดยปกติความยาวของ TCP Header จะเท่ากับ 20 ไบต์ แต่ถ้าหากมีการใช้ Option อาจจะทำให้ ขนาดของเฮดเดอร์ยาวขึ้นตามข้อมูลที่ต้องเพิ่มมาจาก Option นั้น แต่ทั้งหมดแล้วจะไม่เกิน 60 ไบต์
Window Size	เป็นขนาดของการรับ - ส่งข้อมูลในแต่ละครั้งที่ทางฝ่ายผู้รับจะสามารถรับได้ เนื่องจากในการรับข้อมูลนั้น ทางผู้รับจะต้องจัดเตรียมหน่วยความจำในการพักข้อมูลที่มากจาก TCP และทำการ Demultiplex ออกมา หากไม่มีการตกลง ถึงขนาดที่ทางฝ่ายรับสามารถรับได้ ก็จะทำให้การสื่อสารข้อมูลไม่สมดุล และฝ่ายรับอาจจะประมวลผลทัน ซึ่งจะส่งผลให้ต้องส่ง ข้อมูลซ้ำหลายครั้ง
Checksum	ฟิลด์ที่ใช้ในการตรวจสอบความถูกต้องของข้อมูลใน TCP เซกเมนต์ใช้ระบุหมายเลข Sequence Number ของ TCP เซกเมนต์ล่าสุดที่อยู่ในโหมด

	Urgent
Urgent Pointer	ข้อมูลเพิ่มเติมซึ่งจะอยู่ใน TCP Header เมื่อมีการตั้งค่า option บางอย่างที่ต้องการข้อมูลเพิ่มเติมซึ่งไม่มีใน TCP Header เช่น MSS, Strict Route

ตารางที่ 2.6 อธิบาย TCP Header

Connection Establishment



รูปที่ 2.17 3-way handshakes

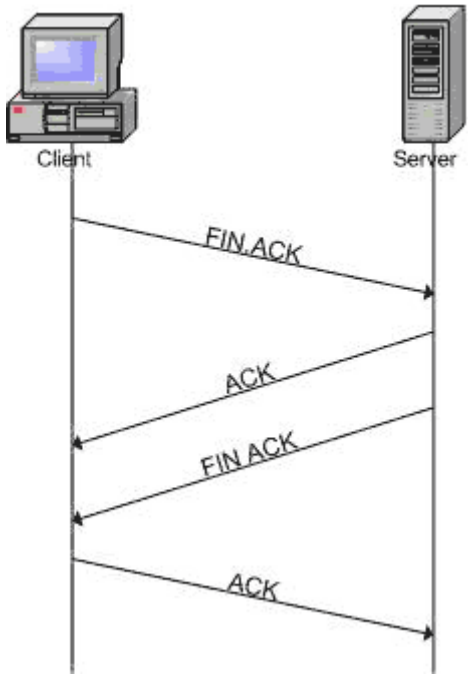
ก่อนที่จะเริ่มต้นการสื่อสาร จะต้องมีการส่งสัญญาณเพื่อบอกโฮสต์อีกฝั่งหนึ่งให้เตรียมตัวติดต่อ ซึ่งกระบวนการที่ใช้มีชื่อเรียกว่า 3-Way Hand Shake มีขั้นตอนคือ

1 .เครื่องไคลเอนต์จะทำการส่งเซกเมนต์ โดยเปิด SYN Flag ระบุหมายเลขพอร์ตที่ต้องการติดต่อบนเซิร์ฟเวอร์และระบุหมายเลข ลำดับของข้อมูล (ISN - Initial Sequence Number)

2. เครื่องเซิร์ฟเวอร์เมื่อได้รับข้อมูลเซกเมนต์จากข้อ 1 ก็จะตอบกลับด้วยการเพิ่มค่า ISN ที่ได้รับขึ้นอีก 1 พร้อมทั้งระบุหมายเลขลำดับ (ISN) ของตนเอง และเปิด SYN กับ ACK Flag

3. ไคล์เอนต์เมื่อได้รับการตอบกลับจากเซิร์ฟเวอร์ตามข้อ 2 ก็จะทำการตอบรับกลับไป โดยการเพิ่มค่า ISN ของเซิร์ฟเวอร์ขึ้นอีก 1 และเปิด ACK Flag เมื่อผ่านการสร้าง connection ทั้ง 3 ขั้นตอนแล้ว ตอนนี้ทั้งไคล์เอนต์ และเซิร์ฟเวอร์เปรียบเสมือนมีการเชื่อมต่อถึงกันแล้ว สถานะของการเชื่อมต่อในขณะนี้เรียกว่า **Established**

Connection Termination



รูปที่ 2.18 TCP Header

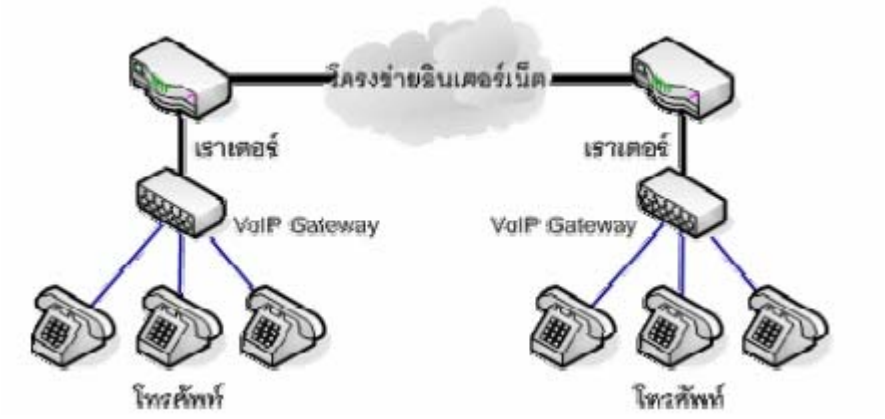
เมื่อการสื่อสารของทั้งสองฝั่งจบลง และไม่ต้องการรับส่งข้อมูลอีกต่อไป จะต้องทำตามขั้นตอนการยุติการสื่อสารเพื่อให้การสื่อสารจบลงอย่างสมบูรณ์ ซึ่งมีอยู่ 4 ขั้นตอนคือ

1. ไคลเอนต์ทำการส่ง ISN พร้อมกับ FIN ACK Flag ไปยังเซิร์ฟเวอร์
2. เซิร์ฟเวอร์ทำการตอบรับ ISN และบวกค่า ISN อีก 1 พร้อม ACK Flag
3. เซิร์ฟเวอร์ทำการส่ง ISN พร้อมกับ FIN ACK Flag ไปยังไคลเอนต์
4. ไคลเอนต์ทำการตอบรับ ISN และบวกค่า ISN อีก 1 พร้อม ACK Flag

การยุติการเชื่อมต่อ โดยส่ง FIN ACK ออกไปมีความหมายคือ โฮสต์ที่ส่งไม่มีข้อมูลจะส่งไปอีก มิใช่ต้องการปิดการสื่อสารทั้งหมดในทันที ดังนั้นจึงต้องทำทั้งสองทาง การสื่อสารจึงจะยุติลงอย่างสมบูรณ์ ในการใช้งานจริง อาจมีการยุติการสื่อสารเพียงด้านเดียว คือหยุดส่งข้อมูล แต่ยังคงเปิดพอร์ตไว้รอรับข้อมูลจากอีกด้านหนึ่ง ทั้งนี้ขึ้นอยู่กับลักษณะการใช้งาน การปิดพอร์ตสื่อสารเพียงด้านเดียวเช่นนี้ เรียกว่า Half-Close

2.2 หลักการพื้นฐาน Voice over IP

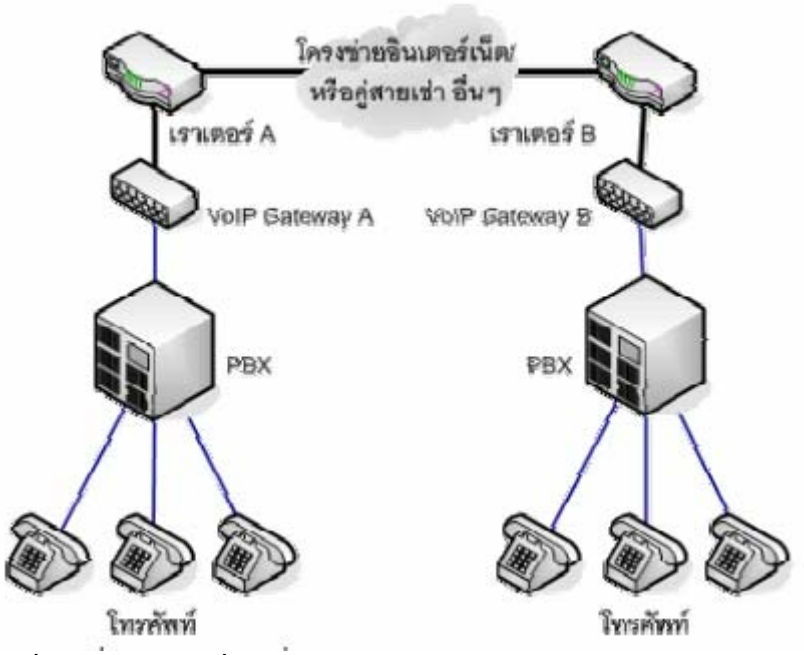
การส่งเสียงบนเครือข่ายไอพี หรือเรียกว่า Voice over IP เป็นระบบที่นำสัญญาณข้อมูลเสียงมาบรรจุลงเป็นแพ็กเก็ต ไอพี แล้วส่งไปโดยที่เราเตอร์มีวิธีปรับตัวเพื่อรับสัญญาณแพ็กเก็ตและยังแก้ปัญหาบางอย่างให้ เช่น การบีบอัดสัญญาณเสียง ให้มีขนาดเล็กลง การแก้ปัญหา เมื่อมีบางแพ็กเก็ตสูญหาย หรือได้มาล่าช้า



รูปที่ 2.19 ภาพการเชื่อมต่อเครือข่ายโทรศัพท์ผ่านอินเตอร์เน็ตโดยนำ VoIP มาใช้

ระบบ VoIP เป็นระบบที่นำสัญญาณเสียงที่ผ่านการดิจิไตซ์ โดยหนึ่งช่องเสียงเมื่อแปลงเป็นข้อมูลจะมีขนาด 64 Kbps การนำข้อมูลเสียงขนาดนี้ ต้องนำมาบีบอัด โดยทั่วไปจะเหลือประมาณ 10 Kbps ต่อช่องสัญญาณเสียงแล้วจึง บรรจุลงในแพ็กเก็ต เพื่อส่งผ่านทางเครือข่ายไอพี การสื่อสารผ่านทางเครือข่ายไอพีต้องมีเราเตอรีทำหนาที่พิเศษเพื่อประกันคุณภาพของช่องสัญญาณไอพีนี้ เพื่อให้ข้อมูลไปถึงปลายทางหรือกลับมาได้อย่างถกต้องและอาจมีการให้สิทธิพิเศษก่อนแพ็กเก็ตไอพีอื่น เพื่อการให้บริการที่ทำให้เสียงมีคุณภาพ

ช่องสัญญาณไอพีนี้เพื่อที่จะให้ข้อมูลไปถึงปลายทางหรือกลับมาได้อย่างถูกต้องและอาจมีการให้สิทธิพิเศษก่อนแพ็กเก็ตไอพีอื่น(Quality of service : QoS) เพื่อการให้บริการที่ทำให้เสียงมีคุณภาพจากระบบดังกล่าวนี้เอง จึงสามารถนำมาประยุกต์ใช้กับระบบเชื่อมโยงเครือข่ายโทรศัพท์ระหว่างสำนักงาน โดยแต่ละสำนักงานสามารถใช้ระบบสื่อสารโทรศัพท์ผ่านทางเครือข่ายไอพี (VoIP) รวมถึงยังสามารถรับส่งข้อมูล (data) ไปพร้อมๆกันได้



รูปที่ 2.20 ภาพการเชื่อมต่อระบบโทรศัพท์โดยใช้ VoIP กับ PABX มาใช้

ด้วยวิธีการสื่อสารแบบ VoIP จึงทำให้ระบบโทรศัพท์ที่เป็นตู้ชุมสายภายในขององค์กรสามารถเชื่อมกันถึงกันผ่านทางเครือข่ายไอพี การสื่อสารแบบนี้ทำให้สามารถใช้โทรศัพท์ข้ามถึงกันได้ในลักษณะ PABX กับ PABX และทำให้ประหยัดค่าใช้จ่ายได้มาก

ขั้นตอนการทำงานของ VoIP

ขั้นตอนการทำงานเบื้องต้น

1. เมื่อผู้พูดโทรศัพท์จากเครื่องโทรศัพท์ธรรมดา หรือพูดผ่านไมโครโฟนที่ถูกต่อเข้ากับการ์ดเสียงของเครื่องคอมพิวเตอร์คลื่นสัญญาณเสียง แบบอนาล็อกก็จะรับการแปลงเป็นสัญญาณดิจิตอล จากนั้นจะถูกบีบอัดด้วยตัวถอดรหัสผ่านอุปกรณ์ PABX (Private Box Exchange) หรือ VoIP Gateway

2. เมื่อผ่าน VoIP Gateway แล้วก็จะถูกส่งต่อไปยัง Gatekeeper เพื่อค้นหาเครื่องปลายทางที่จะรับการติดต่อ เช่น หมายเลขไอพี หมายเลข โทรศัพท์ เป็นต้น แล้วแปลงเป็นแพ็กเกจข้อมูลส่งออกไปบนระบบ เครือข่ายอินเตอรืเน็ต นั่นเอง

3. จะผ่านมาที่ VoIP Gateway ปลายทาง แล้วจึงทำการย้อนกระบวนการทั้งหมดเพื่อส่งให้กับผู้รับปลายทางต่อไป

กระบวนการทำงานของเทคโนโลยี VoIP มีรายละเอียดการทำงานดังนี้

Conversion to PCM (Pulse Code Modulation) จะเป็นการแปลงสัญญาณ Analog ให้ไปอยู่ในรูปแบบสัญญาณ Digital หรือที่เรียกว่า PCM

รูปที่ 2.21 ภาพขั้นตอนการแปลงสัญญาณเป็นดิจิตอล

Removal of Echo จะเป็นการมีการแยกสัญญาณออกเป็นส่วนๆ เพื่อทำการตัดสัญญาณ Echo ออก ซึ่งกระบวนการนี้จะถูกจัดการ โดย DSP (Digital Signal Processors)

รูปที่ 2.22 ภาพขั้นตอนการแยกสัญญาณออกเป็นส่วนๆ เพื่อทำการตัดสัญญาณ Echo ออก

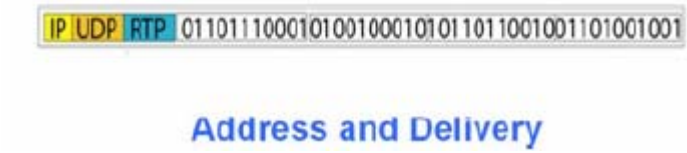
Framing ในส่วนของสัญญาณที่เหลือนั้น ก็จะถูกแบ่งและจัดรูปแบบขึ้นมาใหม่ในรูปของ Frame ซึ่งกระบวนการนี้จะถูกจัดการโดยรูปแบบการบีบอัดที่เรียกว่า CODEC หลังจากกระบวนนี้แล้ว Frame ของสัญญาณเสียงจะถูกสร้างขึ้น

รูปที่ 2.23 ภาพการจัดแบ่งและจัดรูปแบบขึ้นมาใหม่ในรูปแบบของ Frame

Packetisation ในกระบวนการนี้จะเป็นการแปลง Frame ของสัญญาณให้มาอยู่ในรูปของ Packet ซึ่งจะมีการเพิ่ม Header เข้าไปใน Packet โดยในส่วนของ Header นั้นก็จะประกอบไปด้วยข้อมูลที่เรียกว่า Sequence Number และ Time Stamp หลังจากนั้น Packet นี้จะถูกส่งต่อไปที่ Host Processor

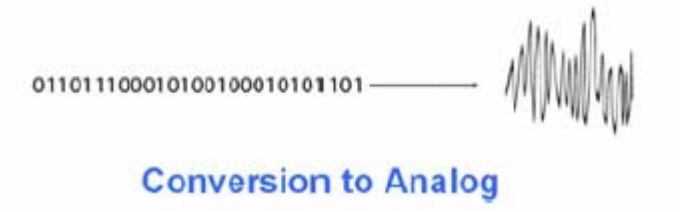
รูปที่ 2.24 ภาพการแปลง Frame ของสัญญาณให้มาอยู่ในรูปของ Packet

Address and Delivery หลังจากที่ได้แปลงสัญญาณให้อยู่ในรูปของ Packet แล้ว ข้อมูลนั้นจะถูกนำมาวิเคราะห์และใส่ค่า IP Address ปลายทาง



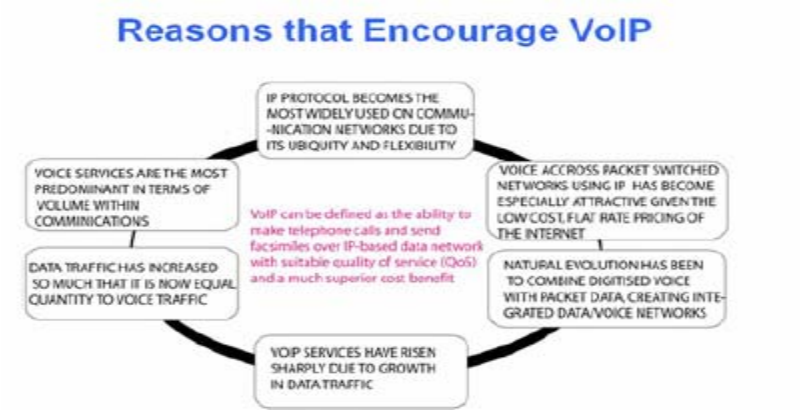
รูปที่ 2.25 ภาพการใส่ค่า IP Address ปลายทาง

Conversion to Analog หลังจากที่ได้ทำการใส่ค่าของ IPAddress ปลายทางไปใน Header ของ Packet แล้วนั้น เมื่อ Packet เหล่านั้นไปถึงด้านปลายทางข้อมูล Header เหล่านี้จะถูกแยกออกเพื่อให้เหลือแค่ Voice Frame หลังจากนั้นก็จะทำการแปลงสัญญาณ Digital PCM ให้กลับมาเป็นสัญญาณรูปแบบ analog ที่เป็นสัญญาณเสียงที่เราได้ยินกันอีกครั้งหนึ่ง



รูปที่ 2.26 ภาพการแปลงสัญญาณ Digital PCM ให้กลับมาเป็นสัญญาณรูปแบบ ที่เราสามารถได้ยินอีกครั้งหนึ่ง

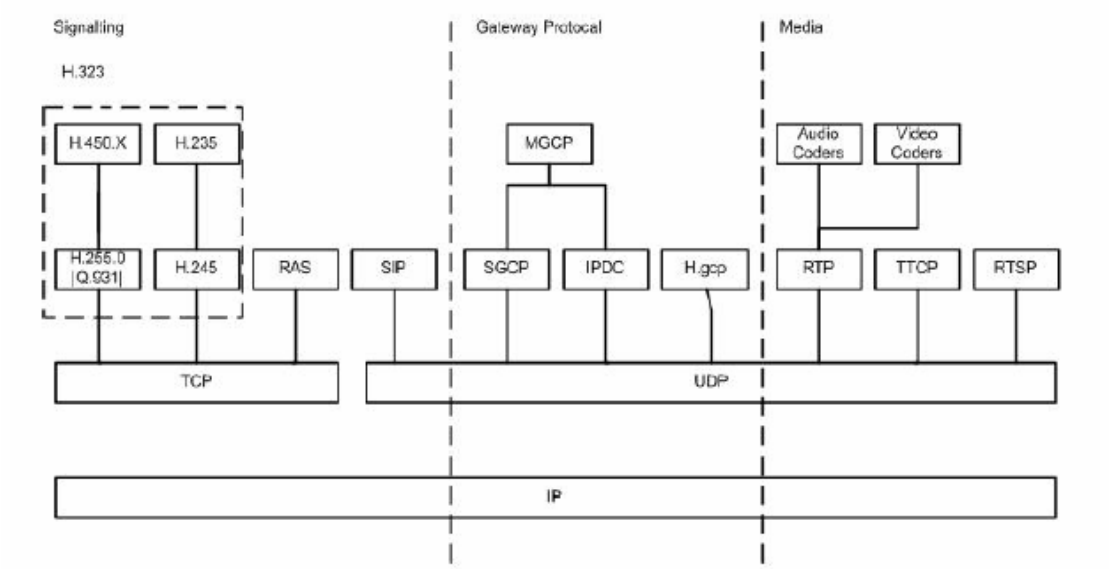
Error Correction กระบวนการนี้จะเป็นกระบวนการที่ใช้ในการตรวจสอบและแก้ไขข้อผิดพลาดซึ่งอาจจะเกิดขึ้นระหว่างการส่งสัญญาณและนำมาซึ่งความผิดเพี้ยนหรือความเสียหายของสัญญาณจนทำให้เราไม่สามารถทำการสื่อสารอย่างถูกต้องได้



รูปที่ 2.27 ภาพขั้นตอนการตรวจสอบความผิดพลาด

Standard of VoIP Technology

โปรโตคอลสำหรับ VoIP มีอยู่ 2 มาตรฐานคือ H.323[2] และ SIP(Session Initial Protocal)[1] โปรโตคอล H.323 เป็นโปรโตคอลที่พัฒนาโดย ITU-T(International Telecommunications Union-Telecommunications section) ส่วน SIP ถูกพัฒนาโดย IETF (Internet Engineering Task Force) มาตรฐานเหล่านี้ เราสามารถเรียกได้อีกอย่างหนึ่งว่า “Call Control Technologies” โปรโตคอลทั้งสองมีหน้าที่หลักในการสร้าง สิ้นสุด และการเปลี่ยนแปลงรายละเอียดของการเรียก ระหว่างผู้ใช้ VoIP รวมทั้งยังสามารถให้ฟังก์ชันเพิ่มเติมอื่นๆ โปรโตคอลทั้งสองเป็นโปรโตคอลสหรับ VoIP ซึ่งใช้บริการชุดโปรโตคอล TCP/IP ในชั้นต่ำกว่า และสามารถใช้งานร่วมกับโปรโตคอลอื่น เพื่อให้เกิดการบริการที่มีคุณภาพมากขึ้น



รูปที่ 2.28 VoIP Protocal Stack

โปรโตคอล H.323 และ SIP เป็นโปรโตคอลในชั้นแอปพลิเคชั่น (Application layer) และใช้บริการของโปรโตคอลในชั้นที่ต่ำกว่า SIP สามารถใช้ได้ทั้ง UDP และ TCP ส่วน H.323 ใช้ TCP เท่านั้น แต่เนื่องจากว่าฟังก์ชันของ H.323 และ SIP มีขอบเขตจำกัด ดังนั้นจึงได้นำโปรโตคอล อื่นมาช่วยในการทำงาน ซึ่งได้แก่ RTP/RTCP ซึ่งทำให้ VoIP สามารถให้บริการได้อย่างมีประสิทธิภาพมากขึ้น โปรโตคอลดังกล่าวเป็นโปรโคคอลในชั้นแอพปลิเคชั่นซึ่งทำงานอยู่บนชุดโปรโตคอล TCP/IP โปรโตคอลเหล่านี้ไม่ได้เป็นโปรโตคอลเฉพาะสำหรับ VoIP ดังนั้นจะกล่าวถึงอย่างคร่าวๆ ในส่วนนี้ แต่สำหรับโปรดตคอล SIP ซึ่งเป็นโปรโตคอลหลักสำหรับ VoIP จะกล่าวถึงรายละเอียดในหัวข้อถัดไป

Real Time Protocal

1.ใช้ในการส่งผ่านข้อมูลผ่านเครือข่ายสำหรับการส่งข้อมูลพวกใช้เวลาจริง

2.สามารถใช้งานร่วมกับโปรแกรมประยุกต์โครงข่ายมัลติมีเดียอื่นได้

3.อาร์ทีพีไม่เป็นแบบ Connection-Oriented

4.ไม่มีความผิดพลาดในการเรียงข้อมูล ซึ่งแตกต่างจากโปรโตคอล ยูดีพี เมื่อส่งแล้วมีปัญหาในการลำดับก่อนหลัง

5.ข้อมูลที่จะส่งถูกควบคุมด้วย RTCP: Real Time Control Protocal

6. RTP ไม่มีการรับประกันคุณภาพของการส่งข้อมูลที่ หมายความว่าไม่มีกลไกใหๆ ยืนยันว่าการส่งสำเร็จหรือไม่

บทที่ 3
ไมโครคอนโทรลเลอร์ MCS-51

3.1 MCS-51

ปัจจุบันไมโครคอนโทรเลอร์ได้เข้ามามีบทบาทมากกับการควบคุมในงานอุตสาหกรรม การพัฒนาอุปกรณ์สารกึ่งตัวนำอย่างต่อเนื่อง ผนวกกับเทคโนโลยีการผลิตที่วิวัตนาการมาเป็นหน่วยไมครอน นำไปสู่การสร้าง CHIP ที่มีขนาดเล็กและมากด้วยคุณภาพ ไมโครคอนโทรเลอร์ 8051 เป็นCHIP ที่นิยมมากตัวหนึ่งที่ใช้ในงานควบคุมเนื่องจากความสามารถที่สูงและ ง่ายต่อการใช้งาน ทั้งนี้ต้องขึ้นอยู่กับผู้ใช้งานที่จะมีวิธีการอย่างไรในการดึงความสามารถของ CHIP มาให้ใช้เต็มที่เพื่อให้เกิดประสิทธิภาพสูงสุด ซึ่งการที่รู้ถึงรายละเอียด และเรียนรู้สถาปัตยกรรมของ เป็นสิ่งแรกที่ต้องทำ ทางขณะผู้จัดทำหวังว่า web page ชุดนี้จะทำให้ผู้อ่านมีความรู้ความเข้าใจและสามารถนำ 8051 ไมโครคอน โทรเลอร์ CHIP ไปใช้งานให้เกิดประโยชน์สูงสุดต่อไป

แนะนำไมโครคอนโทรเลอร์ตระกูล 8051

ไมโครคอนโทรเลอร์ 8051ได้มีการผลิตครั้งแรกโดยบริษัท Intel และใช้ชื่อว่า MCS-51 ซึ่งปรากฏว่านิยมใช้กันอย่างแพร่หลายในไมโครคอนโทรเลอร์ตระกูล 16 บิตต่อมาได้มีหลายบริษัทที่รับลิขสิทธ์จากบริษัท intel ให้มีการนำ MCS-51 ไปผลิตเพื่อจำหน่าย ทำให้เกิด 8051 ไมโครคอนโทรเลอร์ CHIP หลาย SERIES จำนวนมากขึ้นมาซึ่งผู้บริโภคสามารถเลือกใช้ตามความเหมาะสม ของลักษณะงาน

EMBEDDED CONTROLLERS										
Feature	8051AH	8031AH	8751H	80C51BH	80C31BH	87C51	8052AH	8032AH	8752	8044H
Program Memory (Bytes)	4K	-	4K	4K	-	4K	8K	-	8K	4K
RAM Memory (Bytes)	128	128	128	128	128	128	256	256	256	192
Program Memory Expansion (Off Chip) (Bytes)	64k									
Data Memory Expansion(Off Chip) (Bytes)	64K									
Max Clock Frequency (MHz)	12	12	12	16	16	16	16	12	12	12
Typical instruction Time (uS)	1	1	1	0.75	0.75	0.75	1	1	1	1
16-Bit Timer/Counter	2	2	2	2	2	2	3	3	3	2
Serial Communications	Synchronous Mode, Asynchronous Mode, 9 or 10 -Bit Programmable									HDLC/SDLC
No. of I/O Lines	32	16	32	32	16	32	32	16	32	32
Interrupt Sources	5	5	5	5	5	5	6	6	6	5
(Two Priority Levels) Power Requirements 125	125	250	24	24	29	175	175	175	200	
(ICC Max. mA)										
Programmable Power Modes	-	-	-	4.0 mA	4.0 mA	4.0 mA	-	-	-	-
Idele PowerDown				50 uA	51 uA	52 uA				30 mA

ตารางที่ 3.1 แสดงไมโครคอนโทรเลอร์ตระกูล MCS-51 ของบริษัท Intel

การจัดการหน่วยความจำและการเชื่อมต่อ

หน่วยความจำของไมโครคอนโทรเลอร์ 8051 แบ่งเป็น 2 ประเภท คือ

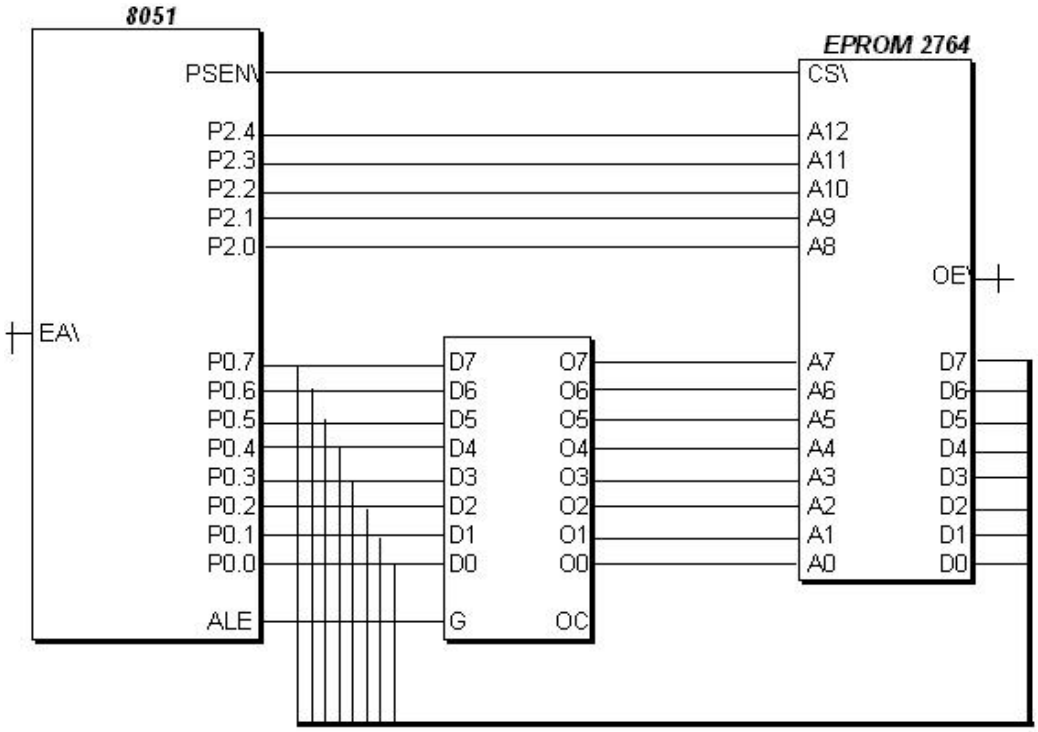
1. หน่วยความจำโปรแกรม(Program Memory) หน่วยความจำประเภทนี้ คือ ROM ใช้เก็บโปรแกรมที่ใช้ในการควบคุมระบบ ซึ่งเป็นหน่วยความจำประเภท non-volatile
2. หน่วยความจำข้อมูล (Data Memory) หน่วยความจำประเภทนี้ได้แก่ RAM

หน่วยความจำโปรแกรม

ใน 8051 จะแบ่งหน่วยความจำประเภทนี้เป็นอีก 2 ประเภท หน่วยความจำโปรแกรมภายนอก (external memory) กล่าวคือ ROM ที่มาต่อภายนอกตัว 8051 ส่วนอีกประเภทหนึ่ง คือ หน่วยความจำภายใน (internal memory) ได้แก่ ROM ภายในตัวไมโครคอนโทรเลอร์เอง การเลือกใช้หน่วยความจำโปรแกรมภายนอกหรือหน่วยความจำโปรแกรมภายในทำได้โดยการให้สัญญาณทางไฟฟ้าที่ขา EA/ โดย

- สัญญาณทางไฟฟ้าที่ขา EA/ เป็นลอจิก 0 หมายถึง หน่วยความจำโปรแกรมภายนอก
- สัญญาณทางไฟฟ้าที่ขา EA/ เป็นลอจิก 1 หมายถึง หน่วยความจำโปรแกรมภายใน

*หมายเหตุ หน่วยความจำโปรแกรมนั้น ไมโครคอนโทรเลอร์ตระกูล 8051 สามารถอ้างข้อมูลได้ 64 กิโลไบต์



รูปที่ 3.1 การเชื่อมต่อ 8051 กับหน่วยความจำโปรแกรมภายนอก

หน่วยความจำข้อมูล

หน่วยความจำข้อมูลมีหน้าที่สำหรับเก็บข้อมูล หรือตัวแปรที่เกิดขึ้นในขณะที่กำลังประมวลผลโปรแกรมไว้เป็นการชั่วคราว โดยพื้นฐานแล้วหน่วยความจำข้อมูลจัดเป็นหน่วยความจำ RAM แบบสแตติก ดังนั้นเมื่อไม่มีการ จ่ายไฟฟ้าให้กับระบบ ก็จะมีผลทำให้ข้อมูลที่จัดเก็บไว้ภายในหน่วยความจำนี้สูญหายไป พื้นที่ของหน่วยความจำข้อมูล ของ 8051 สามารถมีได้สูงสุดไม่เกิน 64 กิโลไบต์ และแยกประเภทออกเป็นสองลักษณะตามตำแหน่งที่ตั้งของหน่วยความจำนั้น

-หน่วยความจำโปรแกรมภายใน (Internal Data Memory) ซึ่ง เป็น RAM ที่อยู่ภายในตัวของไอซีไมโครคอนโทรลเลอร์เอง

-หน่วยความจำข้อมูลภายนอก (External Data Memory) ซึ่งเป็นการใช้ไอซีหน่วยความจำ RAM มาเพิ่มเติมเข้าไปในวงจร ลักษณะเดียวกับการนำไอซี EPROM มาใช้งานเป็นหน่วยความจำโปรแกรมนั่นเอง

หน่วยความจำข้อมูลภายใน

หน่วยความจำข้อมูลภายในของ 8051 มีจำนวนทั้งหมด 256 ไบต์ โดยจำแนกออกได้เป็นสองลักษณะ คือพื้นที่เฉพาะสำหรับตัวประมวลผลกลาง หรือเรียกว่า รีจิสเตอร์ R0-R7 และพื้นที่ใช้งานทั่วไปสำหรับโปรแกรมใช้งานที่ผู้ใช้สร้างขึ้นมา

หน่วยความจำขนาด 128 ไบต์แรก

บริเวณแอดเดรส 00H - 1FH จำนวน 32 ไบต์จำแนกออกเป็นกลุ่ม (Blank) 8 ไบต์ จำนวน 4 กลุ่ม ซึ่งมีชื่อเรียกว่า รีจีสเตอร์ R0 - R7 ดังตารางต่อไปนี้

แอดเดรส	รีจิสเตอร์แบงค์	ชื่อรีจิสเตอร์ใช้งาน
00H-07H	0	R0-R7
08H-0FH	1	R0-R7
10H-17H	2	R0-R7
18H-1FH	3	R0-R7

ตารางที่ 3.2 แสดงรีจีสเตอร์ R0 - R7

จะเห็นได้ว่าชื่อของรีจิสเตอร์ไม่ว่าจะอยู่ในรีจิสเตอร์แบงค์ใด ก็จะมีชื่อ R0 ถึง R7 เหมือนกันทั้งสิ้น ดังนั้นในการใช้งานผู้ใช้จะต้องให้ความระมัดระวังว่าต้องการรีจิสเตอร์นั้นๆ จากแบงค์ใด การสวิตช์ เลือกแต่ละกลุ่มของรีจิสเตอร์นี้ก็ทำได้ง่าย เพียงการกำหนดค่าของบิตที่อยู่ภายในรีจิสเตอร์ PSW เท่านั้นตามตาราง ต่อไปนี้

รีจิเตอร์แบงค์	บิต RS0	บิต RS1	ตำแหน่งหน่วยความจำ
0	0	0	0000H
1	0	1	0008H
2	1	0	0010H
3	1	1	0018H

ตารางที่ 3.3 แสดงรีจีสเตอร์ รีจิสเตอร์ PSW

อย่างไรก็ตามโดยทั่วไปก็มักจะมีการใช้งานรีจิสเตอร์ R0-R7 เฉพาะในแบงก์ 0 เท่านั้น ดังนั้นพื้นที่ของ แบงก์อื่นๆ ที่เหลือก็สามารถนำมาใช้ในลักษณะของหน่วยความจำข้อมูลภายในปกติด้วยการอ้างถึงหมายเลขของ แอดเดรสนั้นๆ โดยตรง

บริเวณแอดเดรส 20H - 2FH จำนวน 16ไบต์ บริเวณพื้นที่เป็นส่วนสำหรับผู้ใช้ซึ่งจะมีความพิเศษต่างไปหน่วยความจำส่วนอื่นๆ เนื่องจากผู้ใช้อาจสามารถอ้างถึงหน่วยความจำบริเวณนี้ได้ทั้งในลักษณะของ ไบต์ข้อมูล เช่น ปกติหรืออาจจะเป็น บิตข้อมูล ได้โดยตรง ดังนั้นหากว่ามองในลักษณะบิตข้อมูลแล้ว ก็จะมีพื้นที่ตัวแปรแบบบิตให้ใช้ งานได้มากถึง128 บิต โดยตำแหน่งแรกของบิตจะเป็นบิตซึ่งเริ่มต้นนับจากบิตนัยสำคัญต่ำสุด(LSB) ของแอดเดรส 20H เรื่อยไปจนกระทั่งถึงบิตที่ 127 ซึ่งเป็นบิตนัยสำคัญสูงสุด (MSB) ของแอดเดรส 2FH ความสามารถในการใช้งานพื้นที่ส่วนนี้แบบบิตข้อมูลโดยตรงนี้นับว่าน่าสนใจมาก และถือเป็นการใช้งาน 8051 อย่างเต็มประสิทธิภาพทีเดียว เนื่องจากว่า 8051 ได้รับการออกแบบมาก็มักจะเป็นเพียงการอ่านค่าสถานะลอจิก ของเส้นสัญญาณ หรือกรณีการส่งออกข้อมูลก็จะเป็นการกำหนดสภาวะลอจิกให้กับวงจรภายนอกผ่านทางบิตใดบิต หนึ่งอยู่แล้ว ดังนั้นหากว่ามีการกำหนดบิตหรืออ่านค่าของบิตมาโดยตรงแทนที่จะต้องทำลอจิกขั้นต้นกับข้อมูลทั้งไบต์ เพื่อต้องการทราบผลเพียงหนึ่งบิตเช่นที่กระทำกันในโปรเซสเซอร์โดยทั่วไป ก็จะเพิ่มความสะดวกและรวดเร็วในการ เขียนโปรแกรมควบคุมมากรายละเอียดในส่วนนี้จะได้กล่าวถึงอีกครั้งหนึ่งเมื่อศึกษาถึงการใช้งานพอร์ต อินพุต/เอาต์พุตต่อไป

บริเวณแอดเดรส 30H - 7FH เป็นบริเวณที่สามารถนำไปใช้งานได้อย่างอิสระ โดยสามารถอ้างถึงได้เฉพาะ ในลักษณะของไบต์ข้อมูลตามปกติเท่านั้น

หน่วยความจำขนาด 128 ไบต์ถัดไป

พื้นที่ตั้งแต่บริเวณตั้งแต่แอดเดรส 80H-FFH เป็นบริเวณของหน่วยความจำที่มีการใช้งานเฉพาะจาก8051เท่านั้น โดยจะนำมาใช้เป็นตำแหน่งของ รีจิสเตอร์หน้าที่พิเศษ (Special-Function Register หรือ SFR) จำนวน 20 ตำแหน่ง ดังแสดงแผนภาพในรูป สำหรับ
ไมโครคอนโทรลเลอร์เบอร์ 8051 จะมีหน่วยความจำข้อมูลภายในสำหรับการใช้งานเพิ่มมากขึ้นกว่าเบอร์อื่นๆ เช่น 8031 หรือ 8751 อีก 128 ไบต์ โดยจะอยู่บริเวณช่วงแอดเดรส 80H ถึง FFH เช่นกันซึ่งแม้ว่า จะเป็นพื้นที่ที่มีหมายเลขแอดเดรสเดียวกับส่วนของรีจิสเตอร์หน้าที่พิเศษ แต่ในความเป็นจริงแล้วจะเป็นพื้นที่หน่วย ความจำอีกบริเวณหนึ่ง ซึ่งมีการซ้อนเกย (Overlap) กันให้อยู่ในบริเวณแอดเดรสส่วนนี้ ซึ่งหากว่าผู้ใช้งานต้องการ จะเก็บข้อมูลในพื้นที่บริเวณนี้ก็จะต้องใช้การอ้างถึงหน่วยความจำแบบโดยอ้อม (Indirect Addressing) เท่านั้น

รีจิสเตอร์หน้าที่พิเศษ

รีจิสเตอร์หน้าที่พิเศษ (SFR) เป็นรีจิสเตอร์สำหรับการควบคุมหน้าที่และการทำงานของอุปกรณ์หรือพอร์ตของ 8051 ทั้งหมด โดยมีตำแหน่งอยู่ในบริเวณแอดเดรส 80H-FFH การใช้งานรีจิสเตอร์หน้าทีเศษเหล่านี้สามารถทำได้ทั้งการระบุถึงชื่อของรีจิสเตอร์หรือตำแหน่งแอดเดรสที่เป็นของรีจิสเตอร์นั้นก็ได้ ตารางต่อไปนี้แสดงเห็นลักษณะการจัดพื้นที่หน่วยความจำ สำหรับรีจิสเตอร์หน้าที่พิเศษเหล่านี้โดยมีข้อ สังเกตว่ารีจิสเตอร์ที่อยู่ตำแหน่งแอดเดรสที่เป็นจำนวนทวีคูณของค่า8จะสามารถอ้างถึงในระดับบิตได้ด้วย (นั่นคือ แอดเดรส 80H, 88H, 90H, 98H,A0H, A8H, B0H
, B8H, D0H, E0H และ F0H)

แอกคิวมูเลเตอร์ (Accumulator) หรือ ACC เป็นรีจิสเตอร์ขนาด 8 บิตทำหน้าที่ในการเก็บข้อมูลที่จะส่งให้กับหน่วยทำงานภายในซีพียูและเก็บผลลัพธ์ ที่ได้จากการทำงานนั้น การทำงานของรีจิสเตอร์นี้มีลักษณะเช่นเดียวกับตัวแอกคิวมูเลเตอร์ของโปรเซสเซอร์ทั่วไปการใช้งานภายในโปรแกรมจะเรียกว่า รีจิสเตอร์ A

รีจิสเตอร์ B เป็นรีจิสเตอร์ที่ใช้สำหรับการทำคำสั่งการคูณแลหารตัวเลข ในกรณีที่ไม่ใช้ในการคำนวณทางด้าน คณิตศาสตร์ ก็สามารถนำไปใช้งานเช่นเดียวกับรีจิสเตอร์ทั่วไปได้

โปรแกรมเคาน์เตอร์ (Program Counter) เป็นรีจิสเตอร์ที่ใช้ในการชี้ตำแหน่งแอดเดรสของหน่วยความจำโปรแกรม ซึ่งจะต้องไปทำงานในลำดับถัด ไป การใช้งานภายในโปรแกรมจะเรียกว่า รีจิสเตอร์ PC

สแต็กพอยน์เตอร์ (Stack Pointer) เป็นรีจิสเตอร์ขนาด 8 บิตทำหน้าที่เก็บตำแหน่งของตัวชี้หรือพอยน์เตอร์ (Pointer) ของบริเวณสแต็ก (Stack) สำหรับเก็บขอ้มูลแอกคิวมูเลเตอร์ รีจิสเตอร์ต่างๆ รวมทั้งข้อมูลจากโปรแกรมโดยปกติแล้วเมื่อทำการเริ่มต้น ระบบใหม่ภายหลังจากการเริ่มจ่ายไฟฟ้า หรือมีการรีเซต (Reset) เกิดขึ้นค่าภายในสแต็กพอยน์เตอร์จะมีค่า 07H ซึ่ง เป็นตำแหน่งแอดเดรสภายในบริเวณเนื้อที่ 128 ไบต์แรกของหน่วยความจำข้อมูลภายใน การใช้งานภายในโปรแกรม จะเรียกว่า รีจิสเตอร์ SP

ตัวชี้ข้อมูล หรือ ดาต้าพอยน์เตอร์ (Data Pointer) เป็นรีจิสเตอร์ขนาด 16 บิต ซึ่งเรียกว่า DPTR และสามารถใช้งานแยกออกเป็นรีจิสเตอร์ขนาด 8 บิตสองตัวคือ รีจิสเตอร์ DPH และ DPL เพื่อเก็บค่าของแอดเดรสของหน่วยความจำที่ต้องใช้งานภายในโปรแกรมหรืออาจจะเป็นแอดเดรสของอุปกรณ์ภายนอก ซึ่งกำหนดให้ติดต่อกันโดยใช้ตำแหน่งของหน่วยความจำภายในโปรแกรม

โปรแกรมสเตตัสเวิร์ด (PSW) รีจิสเตอร์นี้ทำหน้าที่บอกถึงแฟล็กสภาวะการทำงานต่างๆ รวมทั้งบิตสำหรับการกำหนดเลือกแบงก์(Bank) ของรีจิสเตอร์ที่ใช้งานด้วย ดังแสดง

*ชื่อบิต: PSW ตำแหน่ง: D0h ค่าบิตเริ่มต้น: 0000 0111

ชื่อบิต	ตำแหน่ง	ความหมาย
CY	PSW.7	Carry Flag
AC	PSW.6	Auxiliary Carry Flag
F0	PWS.5	Flag 0
RS1	PSW.4	Select Blank bit 1
RS0	PSW.3	Select Blank bit 0
OV	PSW.2	Overflow Flag
_ P	PSW.1 PSW.0	Parity Flag

ตารางที่ 3.4 แสดงโปรแกรมสเตตัสเวิร์ด (PSW)

รีจิสเตอร์ที่เกี่ยวข้องกับพอร์ต (Port Register)

รีจิสเตอร์เหล่านี้จะมีความเกี่ยวข้องกับการทำงานของพอร์ตอินพุต/เอาท์พุตโดยตรง ซึ่งแต่ละตัวจะเป็นรีจิสเตอร์ขนาด 8 บิต สามารถใช้งานได้ทั้งในลักษณะของการอินพุต หรือการเอาต์พุตข้อมูลได้ การดำเนินการใดๆ ที่ เกี่ยวกับพอร์ตทั้งสี่นี้จะมีผลทำให้ข้อมูลที่ตำแหน่งของพอร์ตเหล่านี้เปลี่ยนแปลงไปเช่นกัน นอกจากนี้พอร์ต P0 และ P2 ยังสามารถนำมาใช้ในการติดต่อกับหน่วยความจำโปรแกรมหรือหน่วยความจำข้อมูลภายนอกได้ โดยพอร์ต P2 จะเป็นค่าของแอดเดรส 8 บิตบนของหน่วยความจำ ส่วนพอร์ต P0 นั้นในช่วงเริ่มแรกจะเป็นค่าของแอดเดรส 8 บิต ล่างของหน่วยความจำช่วงเวลาต่อมาจึงจะนำพอร์ต P0 ไปใช้เป็นบัสสำหรับการรับหรือส่งข้อมูลกับหน่วยอุปกรณ์ ภายนอก สำหรับพอร์ต P3นั้นนอกเหนือจากจะใช้ในฐานะของพอร์ตอินพุต/เอาต์พุตเช่นปกติแล้ว ยังนำมาใช้ในฐานะ บัสควบคุมเกี่ยวกับสัญญาณอินเตอร์รัปต์ได้อีกด้วย

รีจิสเตอร์ SBUF เป็นบัฟเฟอร์ขนาด 8 บิต สำหรับการสื่อสารข้อมูลแบบอนุกรมทั้งการรับและส่งข้อมูล ซึ่งตามความเป็นจริงแล้วบัฟเฟอร์นี้มีอยู่ด้วยกันสองชุดและแยกจากกันอย่างชัดเจนสำหรับการส่งและการรับ โดยซีพียูจะทำการจัด การเลือกบัฟเฟอร์ที่เหมาะสมให้โดยอัตโนมัติ

รีจิสเตอร์ PCON เป็นรีจิสเตอร์ที่ใช้ในการควบคุมหน้าที่การทำงานในสามลักษณะ ซึ่งได้แก่ การควบคุมการทำงานของโปรโซสเซอร์ (บิต IDL และ PD) การกำหนดอัตราการทวีคูณของอัตราเร็วในการสื่อสารข้อมูลอนุกรม (บิต SMOD) และแฟล็กสภาวะสำหรับการใช้งานทั่วไป (บิต GR0 และ GR1)

*ชื่อบิต: PCON ตำแหน่ง: 97h ค่าบิตเริ่มต้น: 0xxx 0000

ชื่อ	ตำแหน่ง	ความหมาย
SMOD	PCON.7	บิตทวีคูณของอัตราบอดปรกติ
	PCON.6	
	PCON.5	
	PCON.4	
GF1	PCON.3	แฟลกสำหรับให้ผู้ใช้ ใช้งานทั่วไป
Flag 0		
GF0	PCON.2	แฟลกสำหรับให้ผู้ใช้ ใช้งานทั่วไป Flag 1
PD	PCON.1	บิตสำหรับการกำหนด Power down

IDL	PCON.0	บิตสำหรับการกำหนด idel mode

ตารางที่ 3.5 แสดงรีจิสเตอร์ PCON

บิต PD (Power down) เป็นการกำหนดให้ลดกำลังไฟฟ้าที่จ่ายให้กับส่วนของโปรเซสเซอร์ภายในลงโดยยังคงมีกำลังไฟฟ้าจ่ายให้กับส่วนหน่วยความจำข้อมูลภายในผ่านทางขาสัญญาณ RST วิธีการนี้มักนำมาใช้ในกรณีที่มีการตรวจสอบการไม่มีกำลังไฟฟ้า (Power failure) โดยวงจรตรวจสอบภายนอกจะต้องมีการอินเตอร์รัปต์เข้ามา เพื่อทำการเก็บข้อมูลที่กำลังประมวลผลอยู่ก่อนและเมื่อมีกระแสไฟฟ้าจ่ายให้เป็นปกติแล้ว จึงค่อยนำข้อมูลนั้นมาประมวลผลต่อไป

บิต IDL (Idle Mode) เป็นการกำหนดให้โปรเซสเซอร์หยุดการทำงานชั่วขณะ (Sleep) และจะกลับมาอยู่ในสภาพปกติอีกครั้งเมื่อ ทำการรีเซตทางฮาร์ดแวร์ หรือมีการอินเตอร์รัปต์อย่างใดอย่างหนึ่งเกิดขึ้นการทำงานในลักษณะนี้สามารถเกิดขึ้นได้ ก็เนื่องจากว่าสภาวะการหยุดการทำงานชั่วขณะนั้น เป็นเพียงการห้ามไม่ให้มีสัญญาณนาฬิกาจ่ายให้ส่วนของ โปรเซสเซอร์เท่านั้น ส่วนของวงจรการอินเตอร์รัปต์พอร์ตอนุกรมและวงจรนับ/จับเวลา ยังคงมีสัญญาณนาฬิกาอยู่ เป็นปกติ

รีจิสเตอร์ IP, IE, TMOD, TMOD, SCON

เป็นกลุ่มของรีจิสเตอร์ที่ทำหน้าที่กำหนดการควบคุม และการทำงานของการอินเตอร์รัปต์ต่างๆ ของ 8051

พอร์ตอินพุต/เอาต์พุตของ 8051

พอร์ต มีความหมายถึงแอดเดรสหนึ่งที่ได้รับการกำหนดไว้เพื่อการโอนย้ายข้อมูลระหว่างไมโคร คอนโทรลเลอร์กับอุปกรณ์ภายนอก การกำหนดประเภทของการติดต่อขึ้นอยู่กับทิศทางการไหลของข้อมูลเมื่อพิจารณาจากไมโครคอนโทรลเลอร์เป็นหลัก ดังนั้นการนำเข้าข้อมูลจากวงจรภายนอกจึงเรียกว่า การอินพุต (input)และในกรณีตรงกันข้ามเพื่อส่งออกข้อมูลก็จะเรียกว่า การเอาต์พุต (output) เมื่อพิจารณาถึงวิธีการส่งข้อมูลภายในพอร์ตจะสามารถแยกประเภทของพอร์ตออกได้เป็นสองลักษณะ คือพอร์ตแบบขนาน (Parallel port) ซึ่งทำการส่งจำนวนบิตข้อมูลทั้งหมดออกมาหรือนำเข้าไปพร้อมกันในคราวเดียว และพอร์ตแบบอนุกรม (Serial port) ซึ่งทำการโอนย้าย

ข้อมูลคราวละบิตๆ จนครบจำนวน แต่สำหรับในบทนี้จะกล่าว ถึงเฉพาะในส่วนของพอร์ตแบบขนานเท่านั้น สำหรับการทำงานของพอร์ตแบบอนุกรมจะได้กล่าวภายหลัง

พอร์ตแบบขนานของ 8051

8051 มีโครงสร้างของพอร์ตที่สามารถใช้งานแบบขนานได้จำนวนทั้งหมดสี่พอร์ต เรียกชื่อเรียงตามลำดับว่าพอร์ต0,1,2 และ 3 และเป็นพอร์ตขนาด 8 บิตทั้งหมด การใช้งานพอร์ตสามารถทำได้ทั้งในลักษณะของเส้น สัญญาณเดี่ยวๆหรือกลุ่มของสัญญาณได้ นอกจากนี้พอร์ต 0,2 และ 3 ยังสามารถนำไปใช้งานอื่นๆ ที่ไม่ใช่เป็นพอร์ต อินพุต/เอาต์พุตได้โดยพอร์ต 0 จะทำหน้าที่มัลติเพล็กซ์ ระหว่างบัสแอดเดรสไบต์ต่ำและบัสข้อมูลสำหรับการติดต่อ กับวงจรประกอบร่วม ข้อมูลบัสแอดเดรสไบต์สูงซึ่งจะส่งออกมาทางพอร์ต 2 สำหรับพอร์ต 3 นั้น นอกเหนือไปจากความสามารถเช่นพอร์ตปกติแล้วสามารถนำไปเป็นขาสัญญาณของการอินเตอร์รัปต์ต่างๆ ซึ่งรวมทั้งการสร้างสัญญาณ ควบคุม RD\ และ WR\ เพื่อทำหน้าที่อ่านหรือเขียนหน่วยความจำข้อมูลภายนอกด้วย การใช้งานพอร์ตลักษณะงานแบบ อื่นๆที่ไม่ใช่เป็นพอร์ต/เอาต์พุทนี้จะดำเนินการโดย 8051 เองโดยอัตโนมัติ

โ

ครงสร้างการทำงานของพอร์ต 8051

จากลักษณะโครงสร้างของแต่ละบิตภายในพอร์ตทั้งหมดของ 8051 จะเห็นว่ามีความคล้ายคลึงกันตามลักษณะโครงสร้างที่เรียกว่า Quasi-bidirectional port ยกเว้นพอร์ต 0 ซึ่งเพียงแต่ไม่มี ตัวต้านทานทำหน้าที่ Pull-up สัญญาณไว้ภายในเท่านั้น วงจรประกอบอื่นภายในยังมีฟลิปฟลอปแบบ D ซึ่งมีผลทำให้ พอร์ตสามารถแลตช์หรือค้างสภาวะของสัญญาณได้ นอกจากนี้ในส่วนเอาต์พุตของฟลิปฟลอปเฉพาะของพอร์ต 0 และพอร์ต 2 จะมีโครงสร้างที่ทำหน้าที่คล้ายกับสวิตช์เพิ่มเติมขึ้น เพื่อควบคุมให้เอาต์พุตนี้ต่อเข้ากับส่วนของ ทรานซิสเตอร์ในระหว่างที่ไม่ได้มีการทำงานในลักษณะของบัสแอดเดรสหรือบัสข้อมูลด้วย สำหรับบัฟเฟอร์จำนวนสองตัวของทุกบิตในพอร์ตนั้นมีการทำงานแยกกันโดยอิสระโดยตัวที่อยู่ทางด้านบนจะยอมให้สัญญาณผ่านได้ก็ต่อ เมื่อมีการอ่านค่าข้อมูลที่ค้างไว้ส่วนอีกตัวหนึ่งซึ่งอยู่ทาง
ด้านล่างจะถูกใช้งานเฉพาะเมื่อได้มีการอ่านสถานะของขา สัญญาณเท่านั้น

การใช้งานพอร์ตเป็นการอินพุต

การใช้งานพอร์ตเป็นการอินพุตข้อมูลจะต้องเริ่มด้วยการส่งข้อมูลที่มีค่าเป็น 1 ออกมาทางบิตของพอร์ต นั้นก่อนเป็นลำดับแรก เพื่อหยุดการทำงานของทรานซิสเตอร์ที่ทำหน้าที่ขับสัญญาณเอาต์พุตของบิตนั้น ทำให้ขาสัญญาณของบิตถูกต่อเข้ากับตัวต้านทานซึ่งทำหน้าที่ Pull-up ภายในซึ่งมีผลให้บิตนั้นๆของพอร์ต 1,2 และ 3 เป็น สภาวะของลอจิกสูง ตัวต้านทานนี้มีค่าประมาณ 50 K โอห์ม ซึ่งเป็นค่าที่สูงมาก และทำให้อุปกรณ์ภายนอกสามารถขับสัญญาณของพอร์ตเหล่านี้เป็นลอจิกต่ำได้ง่าย สำหรับบิตของพอร์ต 0 นั้น แม้ว่าจะมีหลักการทำงานที่คล้ายคลึงกัน กับบิตของพอร์ตอื่นๆ แต่เนื่องจากการที่ไม่มีตัวต้านทานทำหน้าที่ Pull-up ภายในไว้ ทำให้เมื่อทรานซิสเตอร์ที่ทำหน้าที่ ขับสัญญาณเอาต์พุตนั้นหยุดการทำงาน ก็จะเป็นผลให้ขาสัญญาณนี้อยู่ในสภาวะอิมพีแดนซ์สูงแทน

การใช้งานพอร์ตเป็นการเอาต์พุต

เมื่อมีการส่งข้อมูลที่มีค่าเป็น 0 ให้กับแต่ละบิตของพอร์ตทุกพอร์ต ข้อมูลนี้จะถูกส่งให้กับฟลิปฟลอปซึ่งจะค้างค่านี้ไว้ และมีผลทำให้ทรานซิสเตอร์ที่ทำหน้าที่ขับสัญญาณเอาต์พุตนั้นทำงาน ดังนั้นขาสัญญาณก็จะมีสภาวะ ลอจิกเป็นลอจิกต่ำส่วนการส่งข้อมูลที่มีค่าเป็น 1 ออกมานั้น ในกรณีที่เป็นการทำงานในแต่ละบิตของพอร์ต 1,2 หรือ 3 จะทำ ให้ทรานซิสเตอร์ที่ทำหน้าที่ขับสัญญาณเอาต์พุตนั้นหยุดการทำงาน มีผลทำให้ขาของสัญญาณเป็นลอจิกสูงด้วยตัว ต้านทานที่ Pull-up อยู่ภายในนั้น แต่สำหรับการทำงานในแต่ละบิตทางพอร์ต 0 นั้นจะมีผลที่แตกต่างออกไป โดยขา สัญญาณจะเป็นสภาวะอิมพีแดนซ์สูงแทน เนื่องจากไม่มีตัวต้านทานภายในเชื่อมต่ออยู่นั่นเอง ดังนั้นในการใช้งานพอร์ต 0 เป็นการเอาต์พุตข้อมูล จึงจำเป็นต้องใช้ตัวต้านทานภายนอก Pull-up สัญญาณไว้กับลอจิกสูงแทน ความสามารถอีกประการหนึ่งเกี่ยวกับพอร์ตอินพุต/เอาต์พุตของ 8051 เป็นวิธีการอ่านลิจิกจากพอร์ตซึ่งมีได้สองวิธีคือ การอ่านค่าลอจิกที่ขาสัญญาณ (Port pin) และวิธีการอ่านลอจิกของการแลตช์ที่พอร์ต (Port latch) วิธีการอ่านค่าจากพอร์ต ทั้งสองแบบนี้จะช่วยให้ระบบทำงานได้ด้วยความถูกต้องมาก ยิ่งขึ้น ยกตัวอย่างเช่น หากว่าพอร์ตถูกนำไปต่อกับขาเบสของทรานซิสเตอร์แบบ NPN และขาอิมิตเตอร์ต่อกับกราวด์ ของระบบ เมื่อมีการส่งค่า 1 ออกไปจะมีผลทำให้ทรานซิสเตอร์ทำงาน ในขณะนั้นถ้าซีพียูมีการอ่านค่าลิจิกจากขา สัญญาณของพอร์ตนี้ก็จะได้ค่าลอจิกต่ำเนื่องจากมองเห็นค่าศักย์ไฟฟ้าระหว่างขาเบสและขาอิมิตเตอร์ซึ่งมีค่าประมาณ 0.6 โวลต์แทนดังนั้นในกรณีเช่นนี้หากว่าเป็นการอ่านค่าจากลอจิกของการแลตช์ ก็จะได้รับค่าระดับลอจิกสูงซึ่งเป็นค่า ที่ถูกต้องสภาพที่เป็นจริง

ลักษณะสมบัติของพอร์ตอินพุต/เอาต์พุต

ดังได้กล่าวแล้วว่าพอร์ต 1,2 และ 3 ของ 8051 มีตัวต้านทาน (ซึ่งสร้างขึ้นจาก FET) ทำหน้าที่ Pull-up ขาสัญญาณไว้และมีค่าประมาณ 50 K โอห์ม ซึ่งถือว่ามีค่าที่สูงมาก เป็นผลให้การเปลี่ยนแปลงระดับสัญญาณลอจิก จากสูงไปต่ำทำได้อย่างรวดเร็ว แต่ในกรณีตรงข้ามจะใช้เวลาการเปลี่ยนแปลงระดับสัญญาณนานกว่ามาก ทั้งนี้เนื่อง จากว่ากระแสจะไหลผ่านตัวต้านทานนี้ได้น้อยมาก ดังนั้นในการแก้ปัญหาจึงได้มีการออกแบบตัวต้านทานเพิ่มขึ้นอีกหนึ่ง ตัวขนานไว้โดยมีค่าประมาณ 1K โอห์ม เรียกว่า Speed-up resistor ซึ่งยอมให้กระแสไหลผ่านได้มากขึ้นประมาณ 50-100 เท่า และจะมีการเชื่อมต่อตัวต้านทานที่เพิ่มขึ้นนี้เฉพาะเมื่อมีการเปลี่ยนระดับสัญญาณจากลอจิกต่ำไปเป็นลอจิก สูงเท่านั้น โดยใช้เวลาประมาณ 2 คล็อกไซเคิล 4.6 คำสั่งการใช้งานพอร์ตอินพุต/เอาต์พุต เนื่องจาก 8051 ใช้หลักการที่เรียกว่า Memory mapped system กล่าวคือ การอ้างถึงพอร์ตรีจิสเตอร์ หรืออุปกรณ์ต่างๆ ภายในระบบ จะเป็นการติดต่อกับหน่วยความจำตำแหน่งหนึ่งเท่านั้น ดังนั้นในการดำเนินการเพื่อนำ เข้าหรือส่งออกข้อมูลกับพอร์ต จึงใช้คำสั่งการอ่านค่าจากหน่วยความจำซึ่งถูกออกแบบให้เป็นตำแหน่งของพอร์ตหรือ คำสั่งการเขียนค่าข้อมูลไปยังตำแหน่งหน่วยความจำนั้นแทน ดังนั้นจะสังเกตเห็นได้ว่าในตารางชุดคำสั่งของ 8051 จะไม่มีคำสั่งที่เกี่ยวกับการทำงานพอร์ตแต่ประการใด เช่น คำสั่ง IN (นำเข้าข้อมูลจากพอร์ต)หรือคำสั่ง OUT(ส่ง ข้อมูลออกจากพอร์ต) เป็นต้น นอกจากนี้ 8051 ยังมีชุดคำสั่งที่จัดการข้อมูลแบบบิตได้โดยตรง (Single-bit Operation) ดังนั้นเรา สามารถที่จะใช้คำสั่งนี้จัดการพอร์ตอินพุต/เอาต์พุตทั้งหมดแบบเส้นสัญญาณเดี่ยวได้โดยการใช้คำสั่ง SETB เพื่อ กำหนดค่าเป็น 1 หรือคำสั่ง CLR เพื่อทำให้บิตมีค่าเป็น 0 คำสั่งเหล่านี้มีประโยชน์มากและทำให้ลดความซับซ้อนในการ ใช้คำสั่งภายในโปรแกรมลงได้มาก

การ Interupt ใน 8051

ประเภทของการ Interrupt

- External interrupt การตรวจสอบสัญญาณที่มา interrupt นี้ จะสามารถกำหนดให้มีการตรวจสอบในลักษณะเมื่อได้มีการเปลี่ยนแปลงระดับสัญญาณ (Level-sensitive) ไปแล้ว หรือในช่วงเวลาขณะเริ่มมี การเปลี่ยนแปลงสัญญาณจาก logic สูงไปต่ำ (Edge-sensitive)

- Internal inetrrupt แหล่งกำเนิดสัญญาณนี้จะเป็นวงจรภายใน Microcontroller เอง เช่น วงจรนับ/จับเวลาวงจรเชื่อต่อสัญญาณอนุกรมเป็นต้นโครงสร้างการ interrupt เกิดได้ 5 ลักษณะคือ

- INT0 สัญญาณ interrupt จากภายนอก ทางขาสัญญาณ P3.2 โดย 8051 จะทำการสุ่มตัวอย่าง สัญญาณเมื่อสิ้นสุดทุก Machine Cycle

- INT1 สัญญาณ interrupt จากภายนอกทางขาสัญญาณ P3.3 โดย 8051 จะทำการสุ่มตัวอย่าง สัญญาณเมื่อสิ้นสุดทุก Machine Cycle

- Timer0 สัญญาณการเกิด Overflow ของ Timer 0

- Timer1 สัญญาณการเกิด Overflow ของ Timer 1

- Serial Port การเกิด interrupt ที่เกิดขึ้นจาการรับ/ส่งข้อมูลอนุกรม ทำให้มีผผลต่อ flag interrupt RI และ TI ตามลำดับ

การกำหนดให้ 8051 สามารถตอบรับการ interrupt แต่ละประเภท ทำได้โดยกำหนด bit ของ ข้อมูลที่เกี่ยวข้อง ซึ่งมักอยู่ภายใน register TCON และ SCON หากได้ว่ามีการกำหนด ค่าของ bit ซึ่งอยู่ภายใน register IE (Intereupt Enable Register) ด้วยแล้ว ก็สามารถตอบรับการ interrupt ของสัญญาณนั้นๆ ได้ นอกจากหนี้ตามแผนภาพในรูป ยังแสดงให้เห็นว่าสัญญาณ interrupt แต่ละประเภท ยังสามารถกำหนด piority ของการ interrupt ได้ 2 ลักษณะ คือ High Low piority กล่าว คือ ขณะที่ประมวลผลอยู่ภายในส่วนของ program ย่อย บริเวณ interrupt ของสัญญาณที่มีระดับความสำคัญต่ำอยู่ ก็สามารถถูก interrupt ที่มี piorrity สูงกว่าได้ แต่หากว่าเป็นสัญญาณ interrupt ที่มี piority เดียวกันหรือต่ำกว่าแล้วก็จะต้องรอให้เสร็จสิ้นการประมวลผลที่ดำเนินอยู่ก่อน

การควบคุม Interrupt

ตามโครงสร้างที่ด้านการจัดการ interrupt ของ 8051 สามารถกำหนดเรียกเพื่อยินยอมหรือไม่ยิน ยอม (Enable/Disabble) ให้มีการ interrupt แต่ละสัญญาณได้ โดยใช้วิธีการกำหนด ค่าของ bit ภาย ใน register IE

วงจรนับ/จับเวลา

8051 ประกอบด้วย register ขนาด 16 bit จำนวน 2 ตัว คือ T0 (Timer0) และ T1 (Timer1) ซึ่งสามารถนำไปใช้งานได้อย่างอิสระ โดยสามารถควบคุมให้ทำหน้าที่เป็นตัวจับเวลา (Timer) เพื่อนับจำนวน plus สัญญาณนาฬิกาภายใน หรือควบคุมให้ทำหน้าที่เป็นตัวนับ (Counter) เพื่อนับจำนวน plus ของระบบ ได้ ภายใน register แต่ละตัวยังสามารถแยกออกได้เป็น register ขนาด 8 bit คือ TH0, TL0, TH1 และ TL1 โดยการทำงานของ register ทั้ง 2 ตัวนี้มีผลมาจากการกำหนดค่าของ bit ที่อยู่ภายใน TMOD (Timer mode control register) และ TCON (Timer/Counter control register) bit ต่างๆภายใน register TMOD bit ต่างๆภายใน register TCON

การ Interupt วงจรนับตรวจ/จับเวลา

จากกระบวนการทำงานของวงจรนับ/จับเวลาของ 8051 จำเป็นต้องกำหนดค่าเริ่มต้นให้กับ register T0 หรือ T1 ค่านี้เป็นค่าจำนวน plus ภายในที่จะต้องนับหรือค่าของจำนวน plus ภายนอก ที่เข้ามาทางขาสัญญาณสัญญาณ T0 หรือ T1 ค่าตัวเลขภายใน register นี้จะต้องลดให้มีค่าที่น้อยกว่าค่าที่ต้องการอยู่หนึ่งค่า ทั้งนี้เนื่องจากการทำงานของ register จะเพิ่มค่าจากที่กำหนดไปเรื่อยๆ จนถึง ค่าสูงสุดของ register และกลับไปเป็นค่า 0 เมื่อมีการเกิด Overflow เกิดขึ้น ทำให้เกิดการกำหนดค่า flag เพื่อแจ้งให้ CPU ได้รับทราบ ดังนั้นโปรแกรมทั่วไปจึงมักใช้สภาวะของ flag นี้(TF0 และ TF1) ซึ่งเป็น bit อยู่ภายใน register TCON เพื่อตรวจสอบว่ากระบวนการนั้นได้เสร็จสิ้นลงแล้ว หรือใช้เพื่อ ทำการ interrupt program ต่อไป ส่วนควบคุมการทำงานของวงจรนับ/จับเวลา ซึ่งประกอบด้วยส่วนของการกำหนดที่มาของสัญญาณ (Timer) หรือ (Counter) และ bit หรือขาสัญญาณสำหรับการหยุดหรือทำงานของวงจรนับ

การทำงานเป็นตัวจับเวลา

ก่อนที่ทำงานเป็นวงจรจับเวลา ต้องมีการกำหนดค่าให้อยู่ในสภาวะดังตารางก่อน

การจับเวลาใน mode 0 : การทำงานใน mode 0 วงจรนับฝจับเวลาจะทำหน้าที่เป็นตัวนับขนาด 13 bit(โดยใช้ register TH0 หรือ TH1 เป็นตัวนับขนาด 8 bit และ register TL0 หรือ TL1 มีขนาด 5 bit)

การจับเวลาใน mode 1 : การทำงานใน mode 1 มีความคล้ายคลึงใน mode 0 มาก แตกต่างกันที่ mode 1 เป็นตัวนับขนาด 16 bit เต็ม ดังรูป

การจับเวลาใน mode 2 : การทำงานใน mode 2 ของวงจรนับ/จับเวลาแตกต่างกันออกไป เพียงใช้ register TL0(TL1) เป็น ตัวนับขนาด 8 bit ส่วน register TH0(TH1) เก็บค่าเริ่มต้นของการนับไว้ ดังรูป

การจับเวลาใน mode 3 : การทำงานใน mode 3 จะสามารถใช้ได้เฉพาะกับ Timer 0 เท่านั้น หากว่านำไปกำหนดให้กับ Timer 1 จะทำให้หยุดการทำงานไป เมื่อ Timer 0 ได้รับการกำหนดทำงานใน mode 3 จะมีผลทำให้ register ของมันแยกการทำงานเป็นอิสระ โดย register TL0 จะถูกควบคุมจาก bit ภายใน register TCON และ ขาสัญญาณ INT0 ดังแสดงในรูป และเมื่อมีการ Overflow เกิดขึ้น จากค่า 0FFH เป็น 00H ก็ จะมีผลให้ flag TF0 มีการเปลี่ยนแปลงเกิดขึ้น สำหรับ register TH0 จะถูกกำหนดให้ทำงานในแบบ ของตัวจับเวลาภายใต้การควบคุมของ bit TR1ใน registerTCON
เท่านั้น และหากเกิด Overflow จะมีผลลเฉพาะต่อ flag TF1 ในส่วน Timer 1 ขณะเมื่อ Timer 0 ถูกกำหนดให้ทำงานmode3 ก็ยังจะสามารถทำงานใน mode อื่นๆ ที่ไม่ใช้ mode 3 ได้ เช่นเดิม ยกเว้น

จะไม่มีการ interrupt เกิดขึ้นเท่านั้น (เนื่องจากflag TF1 ถูก ใช้โดย Timer 0 ไปแล้ว) รูปการทำงานใน mode 3

การทำงานเป็นตัวนับสัญญาณ

การใช้งานในลักษณะตัวนับ(Counter) โดยหลักแล้วจะเหมือนกับลักษณะการทำงานเป็น ตัวจับเวลา (Timer) ดังได้กล่าวในหัวข้อที่ผ่านมา ข้อแตกต่างประการเดียวคือ แทนที่จะนับ plus สัญญาณภายในและผ่านวงจรหาร 12 มาเป็นการนับ plus สัญญาณทางขาสัญญาณ T0(P3.4) ให้ กับ Timer0 หรือขาสัญญาณ T1(P3.5) ให้กับ Timer1 เท่านั้น นอกจากนี้ก่อนการเริ่มต้นใช้งานจะ ต้องกำหนดค่าของ bit C/T ภายใน register TCON ให้มีค่าเป็น 1 เสียก่อน

วงจรนับ/จับเวลา2(Timer2)

mode การทำงานของ Timer2 ประกอบด้วย

Capture mode: สามารถเลือกใช้งานได้ 2 ลักษณะ ด้วยการกำหนดให้กับ bit EXEN2 ของ register T2CON ดังนี้

1. เมื่อกำหนด bit EXEN2 เป็น 0 Timer 2 ยังทำงานเป็นวงจรนับ/ตรวจจับเวลา เมื่อมีการ overflow ขึ้น bit ใน register TF2 จะถูกเซต และสามารถนำไปสร้างการ interrupt ขึ้นได้

2. เมื่อกำหนดค่า bit EXEN2 เป็น 1 การทำงานจะครอบคลุมการทำงานลักษณะข้างต้น แต่จะเพิ่มเมื่อมีการเปลี่ยนแปลงระดับ สัญญาณทางขาสัญญาณ T2EX จาก logic สูง ไปเป็น logic ต่ำ จะมีผลทำให้ค่าข้อมูลภายใน register ของ Timer2 คือ TL2 และ TH2 ถูกนำไปใส่ (Capture) ให้กับ register RCAP2L และ RCAP2H ซึ่งเป็น register หน้าที่พิเศษ หรือ SFR ที่มีใน Microcontroller เบอร์ 8052 เท่านั้น นอกจากนี้จะมีผลทำให้ bit EXF2 ภายใน register T2CON มีค่าเป็น 1 สามารถนำไปใช้งานในการ interrupt ได้เช่นกัน

Auto-reload mode: สามารถทำงานได้ 2 ลักษณะเช่นเดียวกัน

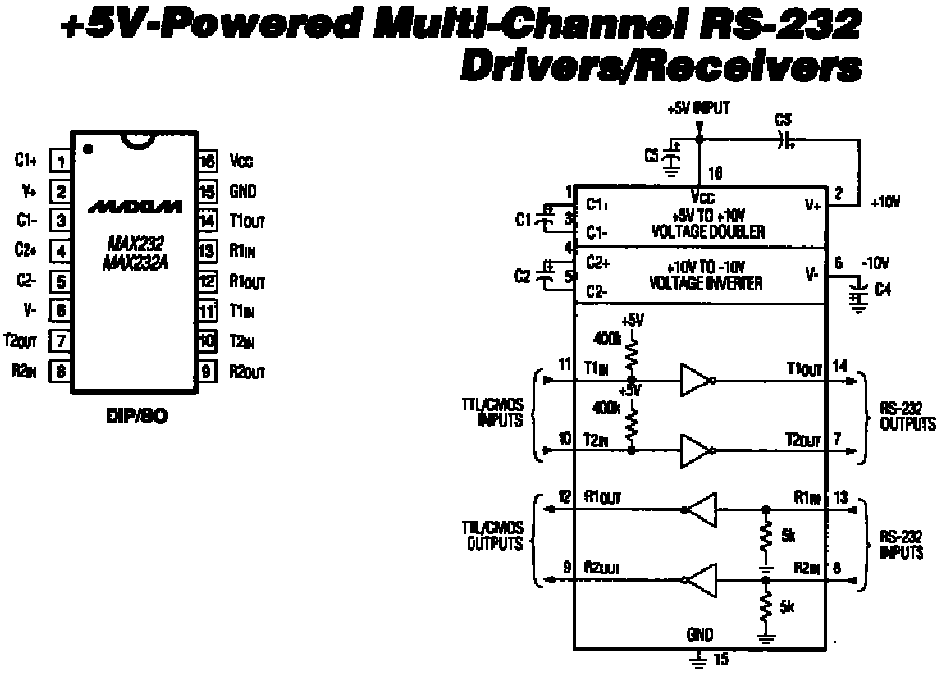
Baud rate Generator: ของ Timer 2 จะมีความแตกต่างจาก Timer 0 และ Timer 1 โดยวงจรรับ และการส่ง สามารถเป็นค่าที่ต่างกันได้ ขึ้นอยู่กับการกำหนดค่าให้กับ bit TCLK และ RCLK ของ Timer 2

การทำงานของโหมดนี้คล้ายคลึงกับการทำงานใน Auto-reload mode กล่าวคือค่าใน register TH2 เปลี่ยนแปลงจากค่า0FFH ไปเป็นค่า 0 หรือที่เรียกว่า overflow จะมีผลให้มีการโหลด

ข้อมูลขนาด 16 bit จาก register RCAP2H และ RCAP2L ซึ่งมีการเตรียมค่าล่วงหน้าแล้วโดยอัตโนมัติการจับเวลาใน

การติดต่อระหว่างไมโครคอนโทรเลอร์กับคอมพิวเตอร์ผ่านพอร์ตอนุกรม

พอร์ตอนุกรมของเครื่องคอมพิวเตอร์จะใช้สัญญาณแรงดันไฟฟ้าไม่เป็นไปตามรูปแบบ TTL มีขนาดสัญญาณ 15Volts เพราะต้องการให้สามารถติดต่ออุปกรณ์ต่อพ่วงที่ระยะไกล แต่ในการใช้งานติดต่อกับไมโครคอนโทรลเลอร์จะต้องเป็นไปตามรูปแบบ TTL 5 Volts ดังนั้นจึงต้องใช้ชิพMAX232 มาเป็นตัวบัฟเฟอร์



รูปที่ 3.2 การต่อชิพMAX232 เพื่อใช้งาน

บทที่ 4
การโปรแกรมส่งข้อมูลผ่านอินเตอร์เน็ทด้วย Microsoft Visual C+ .net

4.1 Winsock

Winsock เป็น API เปิดด้านเน็ตเวิร์คที่เป็นมาตรฐานโดยที่ Winsock ถูกออกแบบมาครั้งแรกเพื่อสร้างการโปรแกรมอินเตอร์เฟสที่เป็นมาตรฐานสำหรับ TCP/IP ในทุกเวอร์ชันของระบบปฏิบัติการ Windows รวมทั้ง Windows XP, Windows 2000, Windows NT และ Windows 98 ซึ่งจะเป็น Winsock เวอร์ชัน 2.2 แต่ถ้าเป็นระบบปฏิบัติการวินโดวส์รุ่นดั้งเดิม เช่น Windows 95 และ Windows CE นั้นจะใช้วินซ็อกเวอร์ชัน 1.1

Winsock เป็นเน็ตเวิร์คแอพพลิเคชันโปรแกรมมิงอินเตอร์เฟสไม่ใช่โปรโตคอล ซึ่ง Winsock นั้นมีรากฐานเดียวกับ Socket ของยูนิกซ์ตระกูล BSD (Berkeley Software Distribution) เวอร์ชัน 4.3 จากมหาวิทยาลัยแคลิฟอร์เนีย วิทยาเขต Berkeley รายละเอียด (Specification) ได้รวมทั้งรูทีนซ็อกเก็ตสไตล์ BSC และการขยายสเปคมาใช้กับ Windows

การใช้ Winsock อนุญาตให้แอพพลิเคชันของเราทำการติดต่อสื่อสารข้ามเน็ตเวิร์คใดๆ ก็ได้ที่กระทำกับ Winsock API แพลตฟอร์ม Win 32 ซึ่ง Winsock ได้ให้เธรดที่ปลอดภัย (thread safety) รวมทั้งมีอีกสองเหตุผลหลักในการใช้ Winsock การคอนโทรลและความมีประสิทธิภาพ

คลาส MFC สนับสนุนการโปรแกรม Winsock API โดยการใช้คลาส CAsyncSocket และ CSocket โดยที่คลาส CAsyncSocket ห้อมล้อมด้วยวินโดวส์ซ็อกเก็ต API ที่ระดับล่าง ซึ่งต้องมีความรู้เกี่ยวกับการสื่อสารข้อมูลผ่านเน็ตเวิร์ค แต่ถ้าเราต้องการอินเตอร์เฟสที่ง่ายกว่าคลาสนี้แล้วควรจะใช้คลาส CSocket คลาส

CSocket สืบทอดจากคลาส และรวมทั้งสืบทอดการห้อมล้อมของ Winsock APIออบเจ็กต์ CSocket แสดงในระดับสูง (Hith level) กว่าออบเจ็กต์ CAsyncSocket

ออบเจ็กต์ CSocket ทำงานร่วมกับคลาส CSocketFile และคลาส CArchive เพื่อจัดการการส่งและการรับข้อมูล

4.2 การใช้ออบเจ็กต์ CAsyncSocket

ก่อนที่จะกล่าวถึงการใช้ออบเจ็กต์นี้ เราควรทำความเข้าใจกับเมมเบอร์ฟังก์ชันที่สำคัญของคลาส CAsyncSocket ซึ่งนำมาใช้ในการสร้างแอพพลิเคชัน ดังตาราง

ชื่อเมมเบอร์ฟังก์ชัน	รายละเอียด
Create	ใช้สำหรับสร้างซ็อกเก็ตใหม่
GetLastError	เอาค่าสถานะความผิดพลาดการกระทำครั้งสุดท้ายที่ล้มเหลว
GetPeerName	เอาค่าที่อยู่(Address) ของซ็อกเก็ตอื่นๆ ที่เชื่อมต่อกัน
GetSockName	เอาค่าที่อยู่ของซ็อกเก็ตโลคอล
Accept	ยอมรับการเชื่อมต่อบนซ็อกเก็ต
Bind	ทำการไบนด์ที่อยู่โลคอลกับซ็อกเก็ต และใช้กับสตรีมซ็อกเก็ตเท่านั้น
Close	ปิดซ็อกเก็ตและถ้าออบเจ็กต์นี้ไม่ได้ใช้งานดีสตรัคเตอร์จะเรียกฟังก์ชันนี้ให้
Connect	สร้างการเชื่อมต่อกับซ็อกเก็ตอื่นๆ และใช้กับสตรีมซ็อกเก็ต
Listen	สร้างซ็อกเก็ตเพื่อฟังการเข้ามา การขอร้อง การเชื่อมต่อและใช้กับสตรีมซ็อกเก็ต
Receive	รับข้อมูลซ็อกเก็ตใช้กับสตรีมซ็อกเก็ต และดาต้าแกรมซ็อกเก็ต
ReceiveFrom	รับข้อมูลดาต้าแกรมและเก็บที่อยู่มาไว้ในโครงสร้าง SOCKADDR
Send	ส่งข้อมูลบนซ็อกเก็ตที่ถูกเชื่อมต่อ ใช้ได้ทั้งสตรีมและดาต้าแกรมซ็อกเก็ต
SendTo	ส่งข้อมูลไปยังปลายทางโดยเฉพาะ ใช้ได้ทั้งสตรีมและดาต้าแกรมซ็อกเก็ต
ShutDown	ทำให้การเรียกฟังก์ชัน Send และหรือ Receive ไม่สามารถใช้ได้บนซ็อกเก็ต
OnAccept	ถูกเรียกบนซ็อกเก็ตที่กำลังฟัง (Listening Socket) เพื่อให้สัญญาณว่าการขอร้องการเชื่อมต่อจากแอพพลิเคชันอื่นๆ กำลังรอคอยพร้อมที่จะถูกยอมรับ
OnConnect	ถูกเรียกบนซ็อกเก็ตที่กำลังเชื่อมต่อ (Connecting Socket) เพื่อให้สัญญาณว่าการช่อมต่อกับแอพพลิเคชันอื่นๆ ได้เรียบร้อยสมบูรณ์แล้ว และแอพพลิเคชันพร้อมส่งรับเมสเสจบนซ็อกเก็ตทั้งสอง
OnSend	แจ้งให้ทราบว่าซ็อกเก็ตนี้พร้อมจะส่งข้อมูลได้โดยการเรียกฟังก์ชัน Send
OnReceived	แจ้งให้ทราบว่าซ็อกเก็ตนี้มีข้อมูลอยู่ในบัฟเฟอร์ (Buffer) และพร้อมที่จะถูกดึงมาใช้โดยการเรียกฟังก์ชัน Receive
OnClose	แจ้งให้ทราบว่าซ็อกเก็ตนี้มีแอพพลิเคชันอีกฝั่งหนึ่งของการเชื่อมต่อมีการปิดแล้ว หรือเกิดการสูญเสียการเชื่อมต่อ

ตารางที่ 4.1 แสดงคลาสเมมเบอร์ของ **CAsyncSocket**

4.3 การสร้างและการใช้ออบเจ็กต์ CAsynSocket

ก่อนที่จะทำการสร้างแอพพลิเคชัน เราจะทำความเข้าใจเกี่ยวกับการใช้ออบเจ็กต์ CAsynSocket และ ขั้นตอนต่างๆ ดังต่อไปนี้

1. สร้างออบเจ็กต์ CAsynSocket และใช้ออบเจ็กต์เพื่อสร้างการจัดการซ็อกเก็ต การสร้างออบเจ็กต์นี้มีด้วยกันสองรูปแบบ แบบแรกมีรูปแบบดังนี้

```
CAsyncSocket  mysock;
Mysock.Create( );  //ใช้ดีฟอลต์พารามิเตอร์
```

หรือแบบนี้

```
CAsyncSocket* pMySocket = new CAsyncSocket;
Int nPort = 2545;
pMySocket->Create( nPort, SOCK_DGRAM );
```

รูปแบบแรก สร้างออบเจ็กต์ CAsyncSocket บนสแต็ค (Stack) ส่วนรูปแบบทั้งสองสร้างออบเจ็กต์บนฮีป (Heap) หรือจองหน่วยความจำไว้ในเมมโมรี ทั้งสองแบบจะใช้ฟังก์ชัน Create แต่ในแบบแรกใช้ดีฟอลต์พารามิเตอร์ซึ่งจะเป็นสตรีมซ็อกเก็ต และในแบบที่สองกำหนดพารามิเตอร์ให้เป็นดาต้าแกรมซ็อกเก็ตด้วยการเจาะจงหมายเลขพอร์ตด้วยการใช้พารามิเตอร์ nPort ซึ่งเป็น short integer หรือมีขนาด 16 บิต

สำหรับซ็อกเก็ตเซิร์ฟเวอร์แล้วเราต้องระบุหมายเลขพอร์ต (ในคลาส MFC กำหนดไว้จะเป็นหมายเลขพอร์ตที่มากกว่า 1,024 ขึ้นไป)

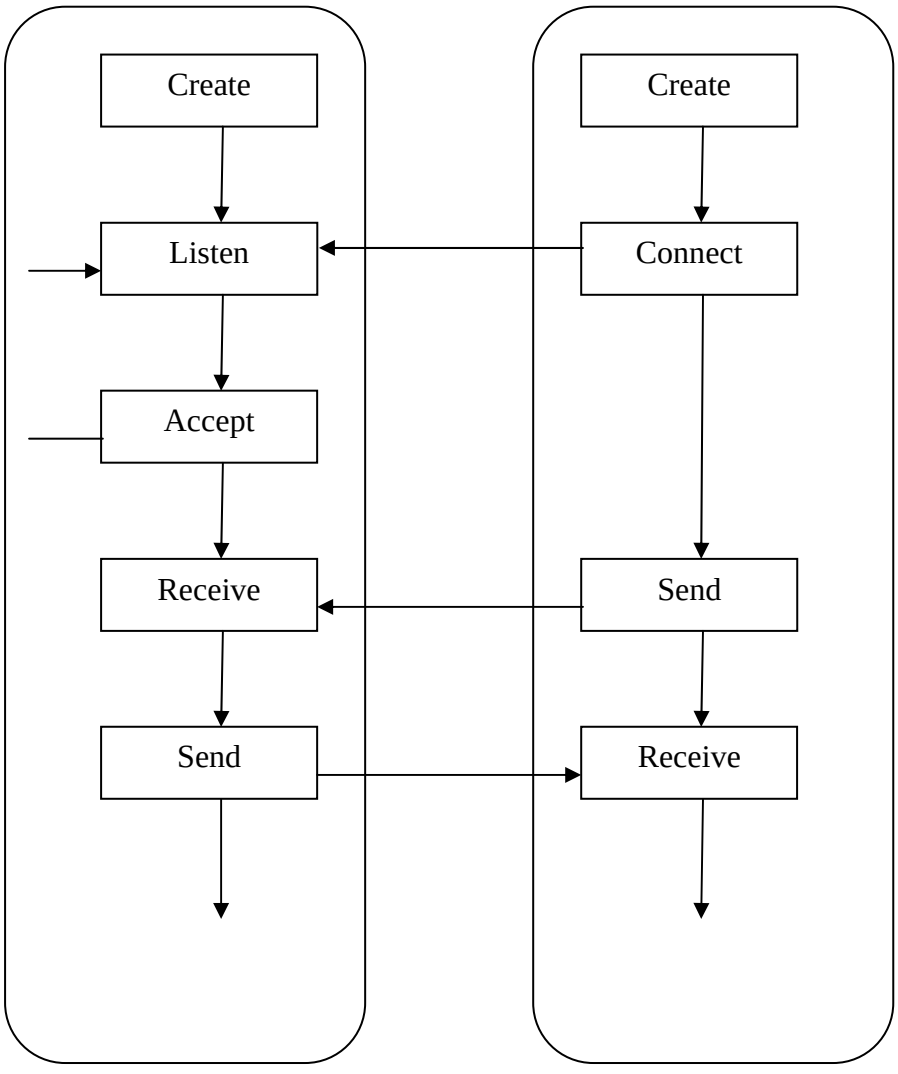
ส่วนซ็อกเก็ตไคลเอนท์แล้วเราต้องใช้ค่าดีฟอลต์ในพารามิเตอร์นี้ ซึ่งจะให้ Winsock เลือกพอร์ตพารามิเตอร์เอง ส่วนพารามิเตอร์ที่สองจะเป็นชนิดซ็อกเก็ตถ้าเป็นดีฟอลต์จะเป็น

SOCK_STREAM หรืออีกชนิดหนึ่งเป็น SOCK_DGRAM พารามิเตอร์ที่สามเป็น IP Address ซ็อกเก็ตมีขนาด 32 บิต โดยปกติจะเป็นการใช้ค่าดีฟอลต์ในพารามิเตอร์นี้

2.ถ้าซ็อกเก็ตเป็นไคลเอนท์ ต้องเชื่อมต่อออบเจ็กต์ซ็อกเก็ต กับเซิร์ฟเวอร์ซ็อกเก็ตด้วยการใช้เมมเบอร์ฟังก์ชัน Connect ของคลาส CAsyncSocket หรือถ้า CAsyncSocketเป็นเซิร์ฟเวอร์ต้องทำการกำหนด CAsyncSocketเพื่อเริ่มต้นการฟัง (ด้วยการใช้เมมเบอร์ฟังก์ชัน Listen) จากไคลเอนท์ในการได้รับการขอร้องการเชื่อมต่อจะยอมรับก็ต่อเมื่อมีการเรียกเมมเบอร์ฟังก์ชัน Accept (ในการใช้เมมเบอร์ฟังก์ชันAccept จะมีอาร์กิวเมนต์อ้างอิงกับออบเจ็กต์ใหม่ซึ่งเป็นออบเจ็กต์ว่างเราต้องสร้างออบเจ็กต์นี้ก่อนที่จะเรียกฟังก์ชัน Accept และออบเจ็กต์นี้จะไม่มีการใช้คำสั่ง Create)

3.ทำการส่งและรับข้อมูลโดยการใช้ฟังก์ชัน Send และReceive ตามลำดับทั้งเซิร์ฟเวอร์และไคลเอนท์

4.ทำลายออบเจ็กต์ CAsyncSocket ถ้าเราสร้างออบเจ็กต์บนสแต็ค โดยดีสตรัคเตอร์จะทำลายออบเจ็กต์นี้เมื่อออบเจ็กต์ไม่ได้ถูกใช้งาน แต่สร้างออบเจ็กต์บนฮีปด้วยการชโอเปอเรเตอร์ new เราต้องเพิ่มโอเปอเรเตอร์ delete เพื่อทำลายออบเจ็กต์ในดีสตรัคเตอร์ด้วย (แต่ถ้าเราลืมใช้ delete ฟังก์ชันดีสตรัคเตอร์จะไปเรียกฟังก์ชัน Close แล้วจึงค่อยทำลาย)



รูปที่ 4.1 Flow chat การทำงานโดยออบเจกต์ CAsyncSocket

4.4 การสร้างแอพพลิเคชั่นฝั่งเซิร์ฟเวอร์ของโปรแกรมควบคุม

การสร้างโปรแกรมฝั่งเซิร์ฟเวอร์ประกอบด้วยการใช้ ใช้ MFC Application Wizard ของ Microsoft Visual C+ สร้างโค้ดหลักให้ และการเพิ่มคลาสใหม่ที่จำเป็นต่อการใช้งาน

การเพิ่มโค้ดเพื่อฟังการขอร้องและการยอมรับการเชื่อมต่อจากไคลเอนท์

ฟังก์ชัน Listen เป็นฟังก์ชันที่ถูกเรียกขึ้นมาเพื่อฟังการขอร้องจากไคลเอ็นท์ ฟังก์ชัน AcceptClient จะทำงานก็ต่อเมื่ออีเวนต์ OnAccept เกิดขึ้นแล้วก็จะเรียกฟังก์ชันนี้เพื่อยอมรับการเชื่อมต่อของไคลเอนท์ เมื่อยอมรับการเชื่อมต่อก็จะแสดง IP Address ทั้งเซิร์ฟเวอร์และไคลเอนท์ รวมทั้งหมายเลขพอร์ตด้วย และพร้อมที่จะรับส่งข้อมูลแล้วหรืออีเวนต์ OnReceive และการส่งข้อมูลจะเกิดขึ้น

การเพิ่มโค้ดเพื่อแสดง IP Address และหมายเลขพอร์ต

ต่อไปจะแสดง IP Address ของเซิร์ฟเวอร์และหมายเลขพอร์ตด้วยการใช้ฟังก์ชัน GetSockName เนื่องจากหมายเลขพอร์ตเป็นจำนวนเต็ม แต่ตัวแปรที่ใช้ในการแสดงข้อมูลบนไดอะล็อกเป็นสตริง ดังนั้นจึงต้องแปลงเลขจำนวนเต็มให้เป็นสตริงด้วยการใช้ฟังก์ชัน Format

การเพิ่มโค้ดเพื่อรับข้อมูลจากไคล์เอนท์

ฟังก์ชัน Receive จะรับข้อมูลแล้วเก็บไว้ในตัวแปรบัฟเฟอร์ (buffer) ขนาด 4 KB เมื่อฟังก์ชัน Receive รับข้อมูลจะต้องมีการตรวจสอบการรับข้อมูลเพื่อรับประกันว่าข้อมูลส่งมาแล้วได้รับถูกต้อง 100 เปอร์เซ็นต์ โดยใช้ switch สเตทเมนต์

การเพิ่มโค้ดเมื่อเลิกใช้งาน

การเลิกการใช้งานสามารถแบ่งได้เป็น 2 กรณี เช่นเดียวกับแอพพลิเคชันฝั่งเซิร์ฟเวอร์คือเมื่อเซิร์ฟเวอร์ยกเลิกการเชื่อมต่อ และเมื่อไคลเอนท์ต้องการปิดการใช้งาน กรณีแรกเมื่อเซิร์ฟเวอร์

ยกเลิกการเชื่อมต่อ อีเวนต์ OnClose ของไคลเอนท์จะเกิดขึ้น และใช้โค้ดเช่นเดียวกับแอพพลิเคชันฝั่งเซิร์ฟเวอร์ และไปเรียกฟังก์ชัน Closed

การเพิ่มโค้ดเพื่อส่งข้อมูลผ่าน RS-232 ไปควบคุมโครคอมพิวเตอร์

จะต้องใช้หลักการเขียนโปรแกรมในแบบการอ่านและเขียนไฟล์ ข้อมูลแทนทั้งนี้อาจจะมองว่าข้อมูลที่จะรับ/ส่งออกพอร์ตนั้นเปรียบได้กับการอ่านและเขียน ข้อมูลบนไฟล์

4 ขั้นตอนเพื่อติดต่อ Serial Port

1. กำหนดตัวแปรสำหรับเก็บ file names,data read (Buffer),file handles,ค่าของการรับ/ส่งข้อมูล

```
HANDLE hComm;          // handle สำหรับ COM port
DCB dcb;               // data structure ที่เก็บค่าสำหรับติดต่อ serial port
char * chCommPort = "COM1"; // ในที่นี้เราใช้ com port 1
int valReturn;         // ใช้เก็บคืนค่ากลับในบ้างฟังก์ชั่น
char chBuffer;         // ไว้เก็บตัวอักษรในการรับและส่งข้อมูล
unsigned long nReadPort;   // ใช้เก็บคืนค่ากลับในฟังก์ชั่น ReadFile
unsigned long nWrittenPort;  // ใช้เก็บคืนค่ากลับในฟังก์ชั่น WriteFile
```

2. ใช้คำสั่ง CreateFile ในการอ้างอิงไฟล์ข้อมูลขึ้นใหม่

```
hComm = CreateFile( chCommPort,
                    GENERIC_READ | GENERIC_WRITE,
                    // กำหนดแบบอ่านและเขียนไฟล์
                    0,
                    NULL,              // ไม่กำหนดการป้องกัน attributes
                    OPEN_EXISTING, // เพื่อเปิดช่องติดต่อสื่อสาร
                    0,                 // ไม่ใช้ verlapped I/O
                    NULL       // เราจะใช้ NULL สำหรับติดต่อช่องทางสื่อสาร
                    );

if (hComm == INVALID_HANDLE_VALUE) {
```

```
  // ตรวจสอบข้อผิดผลาดของโปรแกรม
  m_text1="การสร้างไฟล์ใหม่ผิดผลาด";       //แสดงข้อความออกที่หน้าจอEdit Box
  UpdateData(FALSE);
  exit(1);
  }
// ตรวจสอบสถานะที่จะใช้ Communicate
valReturn = GetCommState(hComm, &dcb);

 if (!valReturn) {
  //แสดงข้อความออกที่หน้าจอใน Edit Box
  m_text1="การตรวจสอบสถานะ Com port ผิดผลาด ";
  UpdateData(FALSE);
  exit (2);
}
```

3. กำหนดค่าสำหรับติดต่อกับ Seiral Port โดยใช้ data structure ชื่อ DCB

```
//  ในส่วนนนี้จะเป็นการตั้งค่าการติดต่อกับCom port
//  DCB เป็น structure สำหรับใช้ควบคุม การติดต่อผ่าน serial port
//  ค่าคงที่ใน DCB ของ baudrate
// CBR_110 CBR_19200
// CBR_300 CBR_38400
// CBR_600 CBR_56000
// CBR_1200 CBR_57600
// CBR_2400 CBR_115200
// CBR_4800 CBR_128000
// CBR_9600 CBR_256000
// CBR_14400
// 9,600 bps เราจะใช้ ,
// 8 data bits, no parity, and 1 stop bit.
```

```
  dcb.BaudRate = CBR_9600;              // กำหนดค่า baud rate
  dcb.ByteSize = 8;                     // ขนาดของจำนวนข้อมูล
  dcb.Parity = NOPARITY;                //  กำหนดให้ไม่มี parity bit //มีรายละเอียดของค่าคงที่
ดังนี้
// ค่าคงที่                 รายละเอียด
// EVENPARITY       Even
// MARKPARITY       Mark
// NOPARITY         No parity
// ODDPARITY        Odd
// SPACEPARITY    Space

 dcb.StopBits = ONESTOPBIT;       // 1 stop bit มีรายละเอียดของค่าคงที่
//ค่าคงที่                      รายละเอียด
//ONESTOPBIT       1 stop bit
//ONE5STOPBITS  1.5 stop bits
//TWOSTOPBITS     2 stop bits

//กำหนดค่าในการ Communcate
valReturn = SetCommState(hCom, &dcb);

if (!valReturn) {
// เงื่อนไขตรวจสอบความผิดผลาด แสดงข้อความออกที่หน้าจอใน Edit Box
  m_text1="การตั้งค่าของ Com Port ผิดผลาด";
  exit (3);
  UpdateData(FALSE);
  }
}
```

4. ใช้คำสั่ง ReadFile และ WriteFile เพื่ออ่านหรือเขียนตัวอักษรผ่าน Serial Port

```
// รูปแบบการอ่านหรือรับข้อมูลผ่านทาง com port
// ในส่วนของ &ch คือตัวแปรที่หน้าที่เป็น Buffer เก็บตัวอักษร //และเลขแปดคือจำนวนของ
ตัวอักษรที่รับเข้ามาแต่ละครั้ง

ReadFile(hCom,&ch,8,&nReadPort,NULL);
m_text1=ch; // แสดงค่าตัวแปรที่รับข้อมูลได้
UpdateData(FALSE);
// รูปแบบการเขียนข้อมูลเพื่อส่งผ่านทาง com port
// ซึ่งจะนำตัวอักษรใน Edit Box ที่จะส่งออกไปเก็บไว้ในตัวแปร m_text2

UpdateData(TRUE);
WriteFile(hCom,m_text2,8,&nWrittenPort, NULL);
```

4.5 การสร้างแอพพลิเคชั่นฝั่งไคลเอนท์ของโปรแกรมควบคุม

การสร้างโปรแกรมฝั่งไคลเอนท์ประกอบด้วยการใช้ MFC Application Wizard ของ Microsoft Visual C+ สร้างโค้ดหลักให้ และการเพิ่มคลาสใหม่ที่จำเป็นต่อการใช้งาน เช่นเดียวกับโปรแกรมทางฝั่งเซิร์ฟเวอร์

การเพิ่มโค้ดเมื่อต้องการติดต่อกับเซิร์ฟเวอร์

ฟังก์ชัน Connect จะทำงานก็ต่อเมื่อผู้ใช้ต้องการติดต่อหรือเชื่อมต่อกับเซิร์ฟเวอร์ หน้าที่หลักของฟังก์ชันนี้คือ สร้างซ็อกเก็ตโดยฟังก์ชัน Create จากนั้นจะเรียกฟังก์ชัน Connect โดยเอาพารามิเตอร์ IP Address และหมายเลขพอร์ตส่งไปเพื่อเชื่อมต่อกับเซิร์ฟเวอร์

การเพิ่มโค้ดเมื่อต้องการส่งข้อมูลให้กับเซิร์ฟเวอร์

กรณีที่ต้องการส่งข้อมูล เราจะนำข้อมูลมาเก็บไว้ในตัวแปร m_strSendMessages โดยการการกำหนดค่าให้โดยตรง ก่อนที่จะมีการส่งข้อมูลไปยังไคลเอนท์ จะทำการตรวจสอบก่อนว่ามีข้อมูลที่จะส่งหรือไม่ ถ้าไม่มีฟังก์ชันนี้ก็จะออกหรือไม่มีการทำงาน ถ้ามีก็จะส่งข้อมูลโดยการใช้ฟังก์ชัน Send เพื่อส่งข้อมูลที่เก็บไว้ในตัวแปร m_strSendMessage ซึ่งในที่นี้จะเป็นสตริงหรือข้อความ

การเพิ่มโค้ดเมื่อเลิกใช้งาน

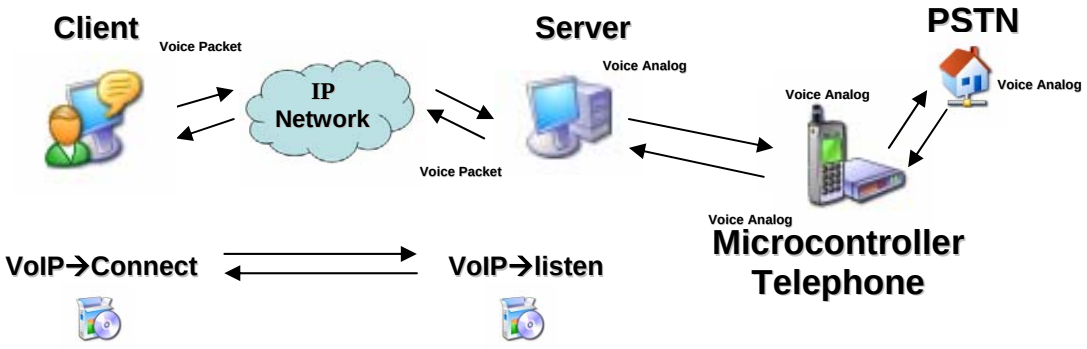
เมื่อไคลเอนท์ต้องการปิดการใช้งาน จะใช้โค้ดเช่นเดียวกับแอพพลิเคชันฝั่งเซิร์ฟเวอร์ และไปเรียกฟังก์ชัน Closed ให้ทำงาน

บทที่ 5
การออกแบบและการสร้าง

5.1 การออกแบบโปรแกรม

โครงสร้างของโปรเจคประกอบด้วยการทำงาน 3 ส่วนคือ

1. โปรแกรมรับส่งคอนโทรล 2 ฝั่ง คือทางด้านClient และทางด้านSever
2. ไมโครคอนโทรลเลอร์ซึ่งมีหน้าที่รับคำสั่งและควบคุมการทำงานโทรศัพท์
3. โปรแกรม VoIP รับส่งสัญญาณเสียง



รูปที่ 5.1 บล็อกไดอะแกรมการออกแบบการทำงานของโครงงาน

โปรแกรมส่งคอนโทรลควบคุมโทรศัพท์ฝั่งClient

โปรแกรมทางด้าน Client จะทำการConnect เชื่อมต่อกับ โปรแกรมทางด้าน Server จากนั้นจะเรียกโปรแกรม VoIP ทางด้าน Server ขึ้นมาเพื่อรอการเชื่อมต่อสัญญาณเสียง และ

ในโปรแกรมจะต้องมีคอนโทรลต่างๆเพื่อที่จะสามารถควบคุมการสั่งงานโทรศัพท์ได้ หลักการทำงานดังรูปที่

รูปที่ 5.2 หลักการทำงานในการคอนโทรลต่างๆเพื่อที่จะสามารถควบคุมการสั่งงานโทรศัพท์

Control ประกอบด้วย Connect, DisConnect, VoIP, (Number 0-9,*,#), Call, Hang

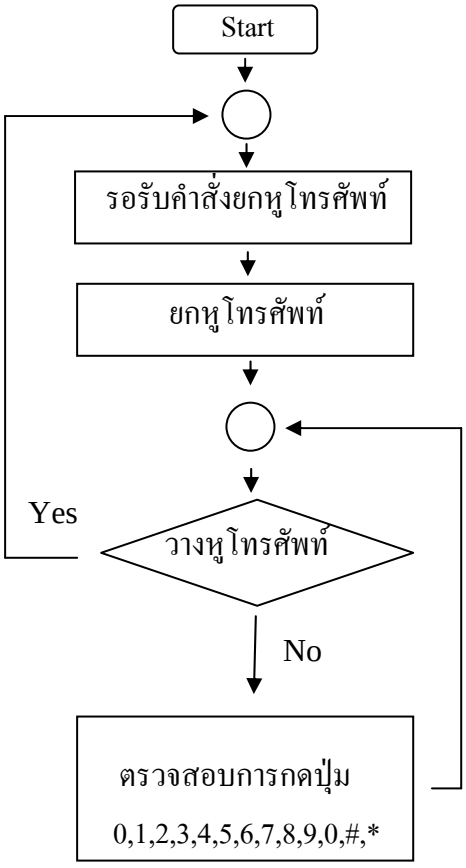
โปรแกรมรับคอนโทรลควบคุมโทรศัพท์ฝั่ง Server

โปรแกรมทางด้าน Server จะรอคอยการเชื่อมต่อจาก Client และเมื่อมีการเชื่อมต่อเข้ามาก็จะต้องเปิดโปรแกรม VoIP ขึ้นมารอคอยการเชื่อมต่อจากโปรแกรม VoIP จากทาง Client ด้วย และเมื่อมีข้อมูลการคอนโทรลมาจาก Client ก็จะส่งออกไปทางพอร์ตอนุกรมให้ไมโครคอนโทรลเลอร์ควบคุมการทำงานโทรศัพท์ และเมื่อมีการใช้งานโทรศัพท์หากมีการสุญเสียการเชื่อมต่อก็จะทำการวางสายอัติโนมัติเพื่อเป็นการป้องกันการยกหูโทรศัพท์ค้าง
หลักการทำงานดังนี้

รูปที่ 5.3 หลักการทำงานโปรแกรมรับคอนโทรลควบคุมโทรศัพท์ฝั่ง Server

ไมโครคอนโทรลเลอร์ควบคุมการทำงานโทรศัพท์

ไมโครคอนโทรลเลอร์มีหน้าที่รับข้อมูลที่ถูกส่งมาทางพอร์ตอนุกรม และควบคุมการทำงานเครื่องโทรศัพท์ มี Flowchart ดังรูปที่ 5.4



รูปที่ 5.4 Flowchart แสดงการควบคุมการทำงานเครื่องโทรศัพท์โดยไมโครคอนโทรเลอร์

โปรแกรม VoIP รับส่งสัญญาณเสียง

ในการพัฒนาโปรแกรมส่งสัญญาณเสียงได้นำซอร์สโค้ดที่เป็นโอเพ่นซอร์ส มาประยุกต์ใช้งาน จาก Objective Systems, Inc. และเป็นโปรแกรมที่ถูกพัฒนาขึ้นด้วยภาษาซี และใช้มาตรฐานโปรโตคอลที่เรียกว่า H.323 ซึ่งเป็นโปรโตคอลที่พัฒนาโดย ITU-T(International Telecommunication Union – Telecommunications section) โดยหน้าที่หลักคือการสร้าง และสิ้นสุดการทำงาน และยังเรียกใช้ชุดโปรโตคอล TCP/IP ในชั้นที่ต่ำกว่าและใช้ร่วมงานกับโปรโตคอลอื่นๆ เพื่อให้เกิดประสิทธิภาพและคุณภาพมากยิ่งขึ้น

ในการพัฒนาใช้งานก็จะเพิ่มโค้ดให้โปรแกรมปิดตัวเองเมื่อทางฝั่งใดฝั่งหนึ่งปิดโปรแกรมใช้งานลงไปอีกฝั่งหนึ่งก็จะปิดตัวเองลงอัติโนมัติเพื่อ การเชื่อมต่อครั้งใหม่ และในส่วนของโปรแกรมเราสามารถใช้ voices vocoder ได้หลายชนิด ซึ่งในการทำโปรเจคได้ทดลอง vocoder ชนิด G.711 ทั้งชนิด a-law และ u-law G.729, G.729A, G.7231, และมาตรฐาน GSM เพื่อทำการใช้งานเนื่องมาจากแต่ละชนิดจะใช้บิทเรทขนาดไม่เท่ากันดังนั้น คุณภาพเสียงก็จะไม่เท่ากัน และความดีเลย์ของสัญญาณก็จะแตกต่างกันด้วย

ในการสร้างตัวโปรแกรม เราก็จะต้องใช้ Microsoft Visual c+ มาเป็นตัวสร้างโปรแกรม เพราะซอร์ซโปรแกรมนี้ได้ถูกพัฒนามาด้วยภาษา C+ และจะเป็นโปรแกรมแบบ win32 ไม่มีหน้าต่างวินโดวส์ แต่จะใช้ความสามารถต่างๆ ของวินโดวส์ รันบนวินโดวส์ได้ แต่จะต้องอยู่ในดอสโหมด

5.2 การสร้างโครงงาน

5.2.1 การเขียนโปรแกรมรับคอนโทรลจากเครื่อง Client ควบคุมเครื่องโทรศัพท์โดยใช้ไมโครคอนโทรลเลอร์ควบคุม

การกำหนดค่าเริ่มต้นให้ไมโครคอนโทรลเลอร์ทำงานร่วมกับพอร์ตอนุกรมได้

เป็นการกำหนดค่าเริ่มต้นให้ไมโครคอนโทรลเลอร์กำหนดค่าไทเมอร์ และค่าซีเรี่ยลเพื่อที่จะทำการรับข้อมูลจากพอร์ตอนุกรมของเครื่องคอมพิวเตอร์

```
INIT_SERIAL:  MOV   TMOD,#00100000B
              MOV   SCON,#01010000B
              MOV   TH1,#0FBH
              SETB  TR1
              RET
```

การรอรับข้อมูลจากพอร์ตอนุกรม

การรอรับข้อมูลจากพอร์ตอนุกรมโดยใช้ไมโครคอนโทรลเลอร์ เราจะใช้วิธีการตรวจสอบบิต RI ถ้ามีข้อมูลถูกส่งมาทางพอร์ตอนุกรมและไมโครคอนโทรลเลอร์ได้รับข้อมูลครบถ้วนแล้วจะเกิดการ Set bit RI ให้เท่ากับ 1 จากนั้นเราจึงทำการอ่านข้อมูลจากรีจิสเตอร์ SBUF โดยทำการย้ายข้อมูลผ่านทางรีจิสเตอร์แอกคิวมูเลเตอร์ A และเมื่อรับข้อมูลเสร็จแล้ว เพื่อจะทำการรับข้อมูลใหม่ทุกครั้ง เราจะต้องทำการเคลีย์บิต RI เสมอ

```
CLR   RI        :   Clear bit RI (RI=0)
JNB   RI,$      :   Jump Here if  RI=0
MOV   A,SBUF    :   Move data  Register SBUF to Register A
CLR   RI        :   Always Clear bit RI after receive data
```

การรับคำสั่งและสั่งการทำงาน

หลังจากที่ได้รับข้อมูลจากพอร์ตอนุกรมเรียบร้อยแล้ว จะทำการเปรียบเทียบข้อมูลที่ได้รับมาว่าเป็นการกดปุ่มหมายเลขใด หรือเป็นการสั่งยกหูโทรศัพท์, วางหูโทรศัพท์ และเมื่อเปรียบเทียบพบแล้วจะสั่งให้ไฟสัญญาณการกดปุ่มกระพริบ และสั่งให้ทำการเซตและเคลีย์บิตโดยใช้เวลาดีเลย์ 0.2 วินาที ตามตำแหน่งขาพอร์ตต่างๆ โดยมีหมายเลขที่กำหนดประจำขาพอร์ต เพื่อให้เสมือนกับการกดปุ่มเครื่องโทรศัพท์ และถ้าเป็นการยกหูโทรศัพท์ก็จะทำการเซตบิตค้างไว้จนเมื่อมีการวางหูโทรศัพท์ก็จะทำการเคลีย์บิต

ตัวอย่างการเปรียบเทียบค่าปุ่ม

```
CH0:  CJNE  A,#'1',CH1  :  Compare A equal number 1
      SETB  P0.2        :  Set  P0.2  =  1
      CALL  DELAY       :  Delay time 200ms
      JMP   LOOP_CH     :  Return to wait data control
```

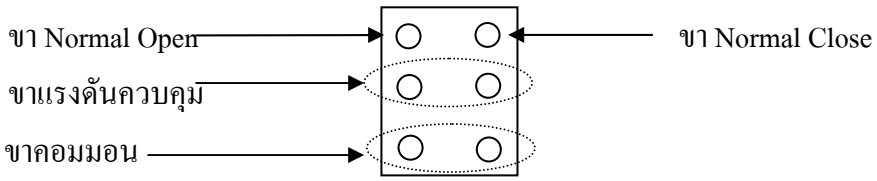
5.2.2 การควบคุมการทำงานของโทรศัพท์โดยใช้รีเลย์เป็นตัวควบคุมหน้าสัมผัสเครื่องโทรศัพท์

รีเลย์ (Relays)

เป็นอุปกรณ์ทำหน้าที่เป็นสวิตช์มีหลักการทำงานคล้ายกับ ขดลวดแม่เหล็กไฟฟ้าหรือโซลินอยด์ (solenoid)รีเลย์ใช้ในการควบคุมวงจร ไฟฟ้าได้อย่างหลากหลาย รีเลย์เป็นสวิตช์ควบคุมที่ทำงานด้วยไฟฟ้า ซึ่งในโครงงานเราจะใช้ รีเลย์ควบคุม ซึ่งมีขนาดเล็กกำลังไฟฟ้าต่ำ ใช้ในวงจรควบคุมทั่วไปที่มีกำลังไฟฟ้าไม่มากนัก

รูปที่ 5.5 รีเลย์ควบคุม

โดยรีเลย์ที่เราใช้ในโครงงานจะเป็นรีเลย์ควบคุมชนิด 6 ขา 5 Volts ซึ่งจะมีขาไฟควบคุมการทำงานของรีเลย์ 2 ขา ขาคอมมอน 2 ขา ขา Normal Open และขา Normal Close อย่างละ 1 ขา



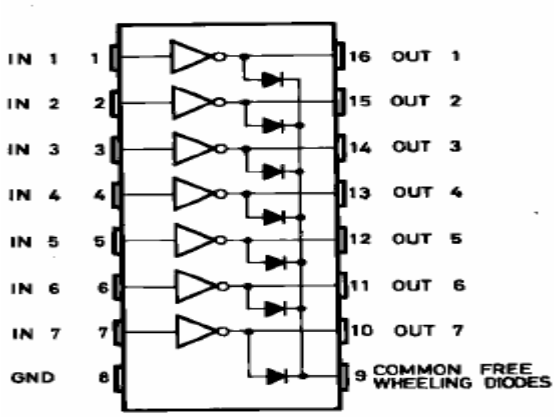
รูปที่ 5. 6 จำลองขาของรีเลย์

การทำงานของรีเลย์จะต้องใช้กระแสไฟมากกว่าที่ไมโครคอนโทรลเลอร์จะขับให้รีเลย์ทำงานได้โดยตรง ดังนั้นจึงต้องมีการใช้ชิพ ULN2003A มาต่อเป็นบัฟเฟอร์ระหว่างรีเลย์กับไมโครคอนโทรลเลอร์ เพื่อที่จะให้สามารถควบคุมการทำงานของรีเลย์ด้วยไมโครคอนโทรเลอร์ได้

รูปที่ 5.7 ULN2003A

ULN2003A

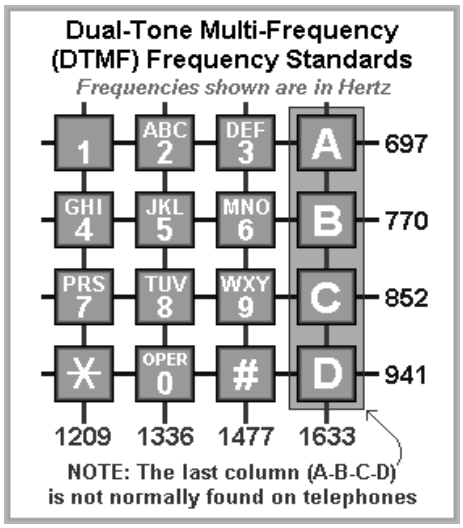
เป็นชิพดาร์ลิงตัน อาร์เรย์ เจ็ดขา ชนิด Open Collector สามารถขับกระแส และแรงดันได้สูงสามารถขับกระแสได้แต่ละช่อง 500 mA เอาท์พุตสูงสุด 50Volts อินพุตคือสัญญาณ 5V TTL



รูปที่5.8 แสดงวงจรภายในชิพ ULN2003A

การควบคุมการกดปุ่มของเครื่องโทรศัพท์

เครื่องโทรศัพท์ปกติจะมีคอนแทคสำหรับปุ่มกดเป็นไปตามรูปแบบ DTMF (Dual Tone Multiple Frequency) โดยการกดปุ่มหมายเลขหนึ่งหมายเลขจะมีไมโครชิพทำการมอดดูเลตสัญญาณ 2 ความถี่ก่อนจะส่งสัญญาณออกไปเพื่อให้ชุมสายรู้ว่าเป็นหมายเลขใด

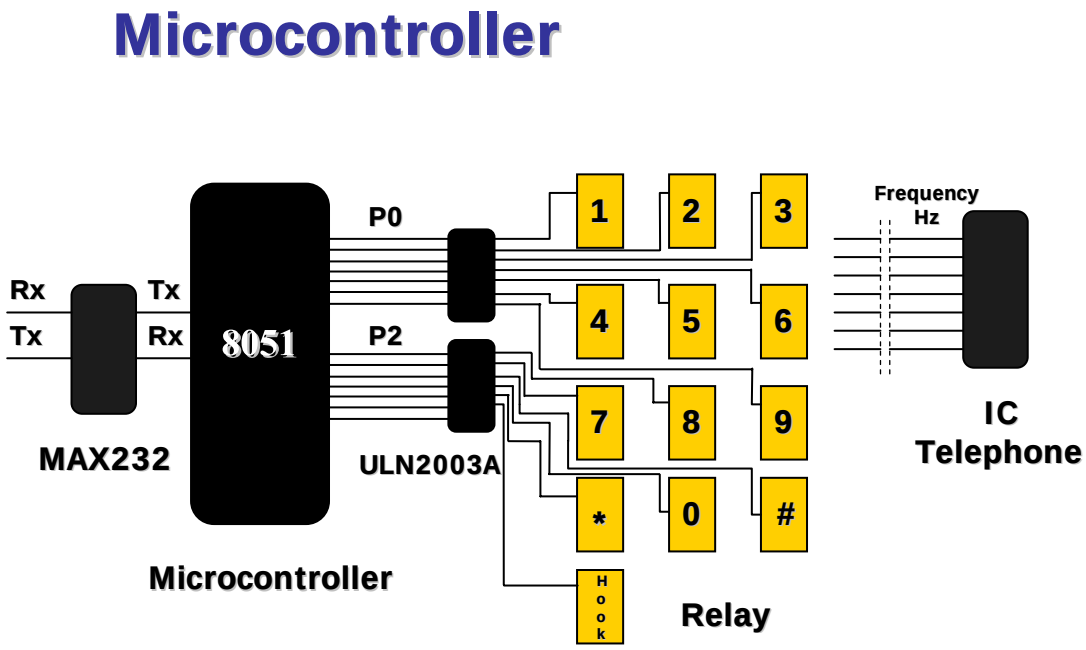


รูปที่5.9 ตาราง Dual-Tone Multi Frequency Standards

ในตัวเครื่องโทรศัพท์ คอนแทคสำหรับปุ่มกดจะซ้ำกันโดยความถี่ เป็นคอลัมน์และแถว เราก็จะสามารถนำความถี่แยกออกมาเป็นแต่ละความถี่ได้ และเมื่อนำความถี่มารวมกันเป็นคู่ตามมาตรฐานการมอดูเลตสัญญาณก็จะทำให้เราสามารถควบคุมการกดปุ่มของเครื่องโทรศัพท์ได้

ในโครงงานนี้จะใช้รีเลย์จำนวน 12 ตัว เป็นสวิตซ์ ใช้แทนการกดหมายเลขทั้ง 0 – 9 และ * กับ # โดยนำความถี่แต่ละแถวแต่ละคอลัมน์มาเรียงและจัดเป็นคู่ให้ตรงตามมาตรฐาน DTMF

5.2.3 ไดอะแกรมของอุปกรณ์ควบคุมโทรศัพท์



รูปที่ 5.10 ไดอะแกรมของอุปกรณ์ควบคุมโทรศัพท์

ขาที่	P0.2	P0.1	P0.0	P0.3	P0.5	P0.4	P2.6	P2.7	P0.6	P2.3	P2.4	P2.5	P2.2
หมายเลข	1	2	3	4	5	6	7	8	9	*	0	#	ยกหู, วางหู

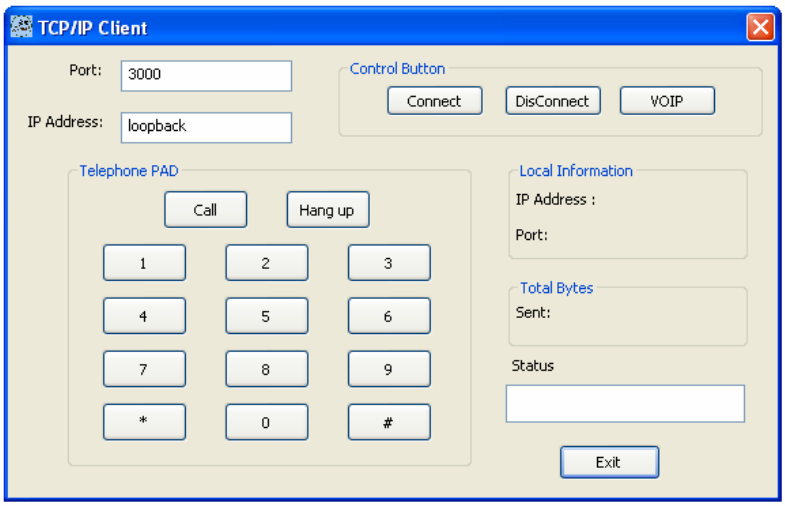
ตารางที่ 5.1 แสดงการต่อขาของไมโครคอนโทรเลอร์

เมื่อ User ทำการกดปุ่มยกหูโทรศัพท์ ไมโครคอนโทรลเลอร์จะได้รับคำสั่งให้เซตบิตขาที่ P2.2 ค้างไว้ และจะรอรับหมายเลขจาก User เมื่อได้รับหมายเลขแล้วจะทำการเซตและเคลีย์บิตตามตำแหน่งขาที่กำหนดไว้ตามหมายเลขนั้นๆ โดยมีการหน่วงเวลาไว้เป็นเวลา 0.2 วินาที เพื่อเปรียบเสมือนการกดปุ่ม จนกระทั่ง User ทำการกดปุ่มวางสายโทรศัพท์ ไมโครคอนโทรลเลอร์จะทำการเคลียร์บิตที่ขา P2.2

5.2.4 การสร้างโปรแกรมควบคุมเครื่องโทรศัพท์ผ่านอินเตอร์เน็ต

ในการสร้างโปรแกรมทั้งทางด้านเซิร์ฟเวอร์และไคลเอนท์ เราจะใช้ Application Wizard ของ Microsoft Visual c+ .net สร้างโค้ดหลัก ซึ่งเป็นหน้าต่างและปุ่มควบคุมให้ และทำการเพิ่มโค้ดที่มีหน้าที่ ทำการร้องขอการเชื่อมต่อ การส่งข้อมูล ในโปรแกรมฝั่งไคล์เอนท์ และโค้ดที่มีหน้าที่รอฟังการเชื่อมต่อ, รับการเชื่อมต่อ และรับข้อมูลจากไคล์เอนท์ ในโปรแกรมฝั่งเซิร์ฟเวอร์ ในความสามารถทางด้านเน็ตเวิร์คที่ใช้ จะเป็น Winsock API ซึ่งเป็นมาตรฐานของการพัฒนาโปรแกรมเน็ตเวิร์คบนวินโดวส์ เพื่อที่จะสร้างอินเตอร์เฟสสำหรับ TCP/IP

การสร้างโปรแกรมควบคุมทางฝั่งไคลเอนท์



รูปที่ 5.11 โปรแกรมควบคุมเครื่องโทรศัพท์ผ่านอินเตอร์เน็ตฝั่งไคลเอนท์

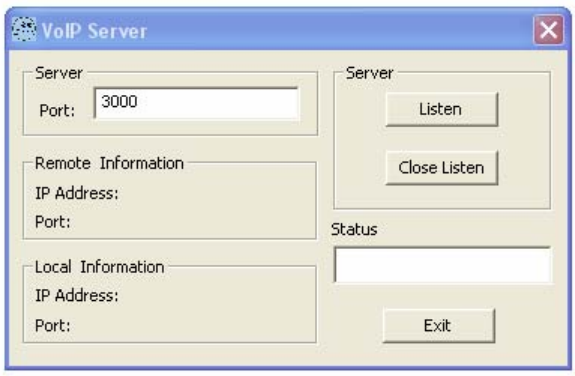
ในโปรแกรมจะประกอบไปด้วยช่องใส่หมายเลข IP Address และหมายเลขพอร์ท ซึ่งเป็นตำแหน่งของเครื่องเซิร์ฟเวอร์ ปุ่ม Control ต่างๆเพื่อใช้ในการควบคุม และช่องสำหรับแสดงข้อมูล

ต่างๆ เช่นหมายเลข IP Address, Port ของเครื่องไคลเอนท์ จำนวนข้อมูลที่ส่ง และ หน้าต่างสถานการณ์ส่งข้อมูล

ในการสร้างโปรแกรมจะต้องมีปุ่ม Control ต่างๆดังนี้

1. ปุ่ม Connect เพื่อทำการร้องขอการเชื่อมต่อกับโปรแกรมฝั่งเซิร์ฟเวอร์ และสั่งให้โปรแกรมส่งสัญญาณเสียงผ่านอินเตอร์เน็ตทำงานขึ้นมาในฝั่งเซิร์ฟเวอร์ ในสถานะรอการเชื่อมต่อ
2. ปุ่ม Disconnect เพื่อทำการยกเลิกการเชื่อมต่อ และในกรณีที่กำลังใช้งานอยู่จะสั่งให้โทรศัพท์วางสายอัตโนมัติ
3. ปุ่ม VoIP เพื่อสั่งให้โปรแกรมส่งสัญญาณเสียงผ่านอินเตอร์เน็ตทางฝั่งไคลเอนท์ขึ้นมาทำงาน ในสถานะเชื่อมต่อ
4. ปุ่มทั้งหมดใน Telephone Pad เพื่อทำการควบคุมโทรศัพท์
5. ปุ่ม Exit เพื่อทำการออกจากโปรแกรม

การสร้างโปรแกรมควบคุมทางฝั่งเซิร์ฟเวอร์



รูปที่ 5.12 โปรแกรมควบคุมเครื่องโทรศัพท์ผ่านอินเตอร์เน็ตฝั่งเซิร์ฟเวอร์

ในโปรแกรมจะประกอบด้วยช่องใส่หมายเลขพอร์ท ปุ่มControl และช่องแสดงข้อมูลหมายเลข IPAddress และหมายเลขพอร์ท ของเครื่องเซิร์ฟเวอร์ และไคลเอนท์ และช่องแสดงการรับข้อมูลต่างๆเข้ามา

ในการสร้างโปรแกรมจะต้องมีปุ่ม Control ต่างๆดังนี้

1. ปุ่ม Listen ซึ่งเป็นปุ่มควบคุมให้โปรแกรมอยู่ในสถานะรอการเชื่อมต่อจากไคลเอนท์
2. ปุ่ม Close Listen ซึ่งเป็นปุ่มควบคุมให้โปรแกรมทำการหยุดการเชื่อมต่อ

การสร้างโปรแกรมส่งสัญญาณเสียงทั้งทางด้านเซิร์ฟเวอร์และไคลเอนท์

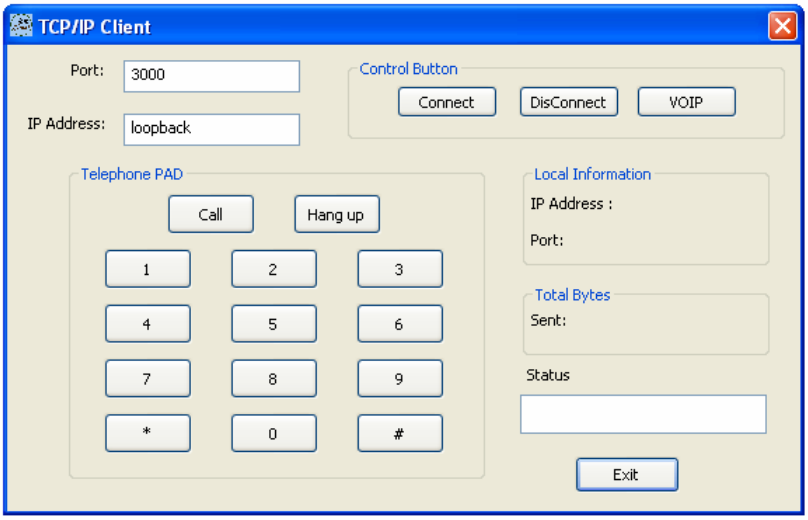
โปรแกรมที่ใช้เป็นโอเพ่นซอร์ส ซึ่งพัฒนาโดยใช้ภาษาซี แสต็ก ไม่มีปุ่มควบคุมคอนโทรล ลักษณะเป็นหน้าจอเหมือนโปรแกรมรันบนดอส ในโปรแกรมเราสามารถเลือกการใช้งาน Speech Vocoders ได้หลายมาตรฐานในโปรแกรมเราจะใช้มาตรฐาน G.711 และในส่วนของทางด้านเซิร์ฟเวอร์จะเพิ่มโค้ดให้โปรแกรมปิดตัวเองเมื่อการเชื่อมต่อถูกตัดขาด

บทที่ 6
การใช้งาน

6.1 การติดตั้งโปรแกรมลงบนไคลเอนท์และเซิร์ฟเวอร์

การติดตั้งโปรแกรมลงบนไคลเอนท์

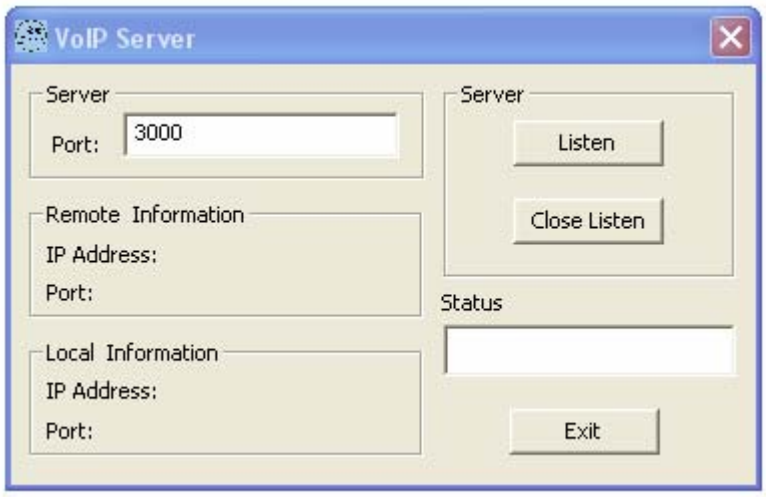
นำโปรแกรมควบคุมโทรศัพท์ผ่านทางอินเตอร์เน็ท(VoIP Client) มาติดตั้งลงบนคอมพิวเตอร์ฝั่งไคลเอนท์



รูปที่ 6.1 โปรแกรมควบคุมโทรศัพท์ผ่านทางอินเตอร์เน็ท(VoIP Client)

การติดตั้งโปรแกรมลงบนเซิร์ฟเวอร์

นำโปรแกรมควบคุมโทรศัพท์ผ่านทางอินเตอร์เน็ท(VoIP Server) มาติดตั้งลงบนคอมพิวเตอร์ฝั่งเซิร์ฟเวอร์



รูปที่ 6.2 โปรแกรมควบคุมโทรศัพท์ผ่านทางอินเตอร์เน็ท(VoIP Server)

6.2 การต่อกล่องควบคุมที่ฝั่งเซิร์ฟเวอร์

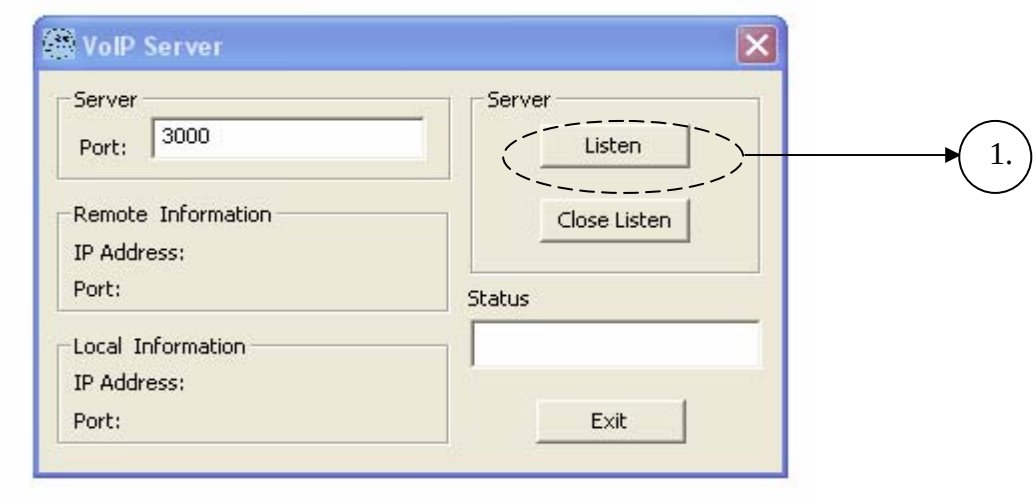
นำกล่องควบคุมโทรศัพท์ผ่านทางอินเตอร์เน็ทมาต่อกับเครื่องคอมพิวเตอร์ฝั่งเซิร์ฟเวอร์ซึ่งได้ผ่านการติดตั้งโปรแกรมควบคุมโทรศัพท์ผ่านทางอินเตอร์เน็ท(VoIP Server) เรียบร้อยแล้ว โดยนำพอร์ตอนุกรมต่อเข้ากับพอร์ตอนุกรมของคอมพิวเตอร์ (com1)ที่ต่ออยู่กับกล่องควบคุมโทรศัพท์ผ่านทางอินเตอร์เน็ท แล้วนำสายต่อช่อง Speaker Out ของซาวการ์ดที่เครื่องคอมพิวเตอร์ฝเข้ากับช่อง mic ของกล่องควบคุมโทรศัพท์ผ่านทางอินเตอร์เน็ท

รูปที่ 6.3 กล่องควบคุมโทรศัพท์ผ่านทางอินเตอร์เน็ท

6.3 การใช้โปรแกรมเพื่อเริ่มทำงานของคอมพิวเตอร์ฝั่งเซิร์ฟเวอร์

หลังจากที่ทำการติดตั้งโปรแกรมควบคุมโทรศัพท์ผ่านทางอินเตอร์เน็ท(VoIP Server) ติดลงบนคอมพิวเตอร์ฝั่งเซิร์ฟเวอร์แล้ว

1. คลิกแท็บ Listen เพื่อรอรับการติดต่อจากฝั่งไคลเอนท์

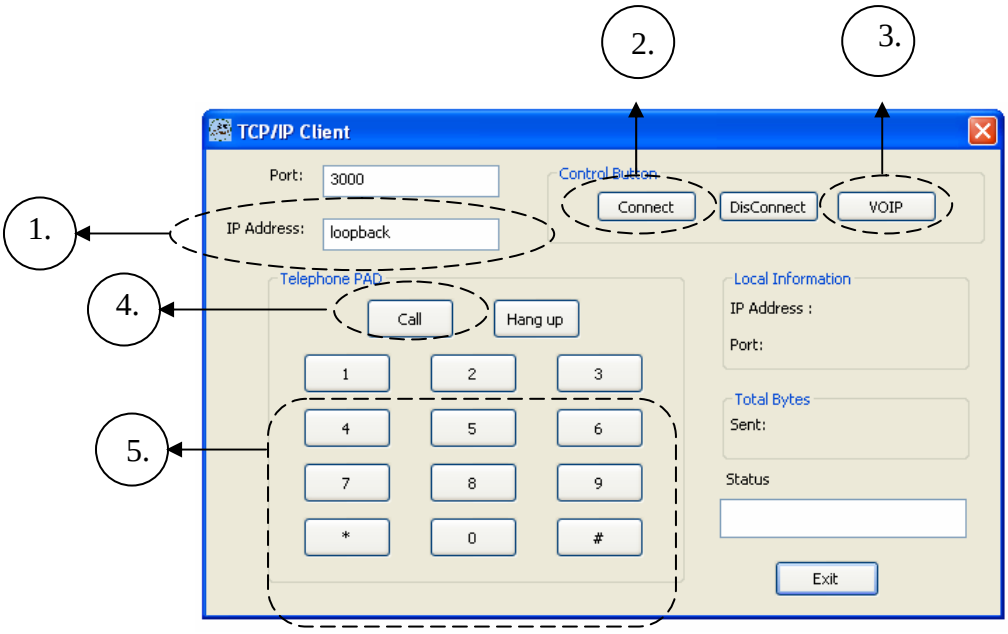


รูปที่ 6.4 โปรแกรมควบคุมโทรศัพท์ฝั่งเซิร์ฟเวอร์

6.4 การใช้โปรแกรมเพื่อติดต่อการควบคุมผ่านทางอินเตอร์เน็ท

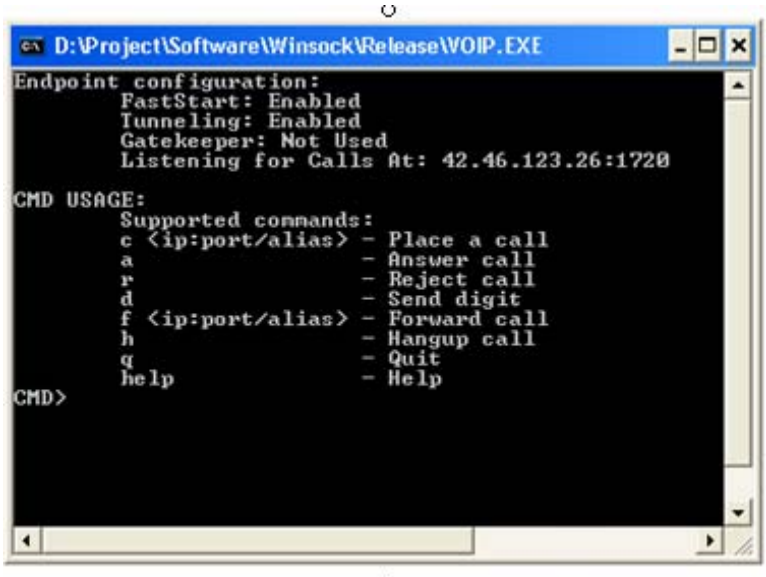
6.4.1 ฝั่งไคลเอนท์

1. ทำการติดตั้ง IP Address ของคอมพิวเตอร์ฝั่ง Server ที่เราทำการติดตั้งกล่องควบคุมโทรศัพท์ผ่านทางอินเตอร์เน็ท (VoIP Server) และโปรแกรมถูกตั้งให้อยู่ที่สถานะ Listen เรียบร้อยแล้ว



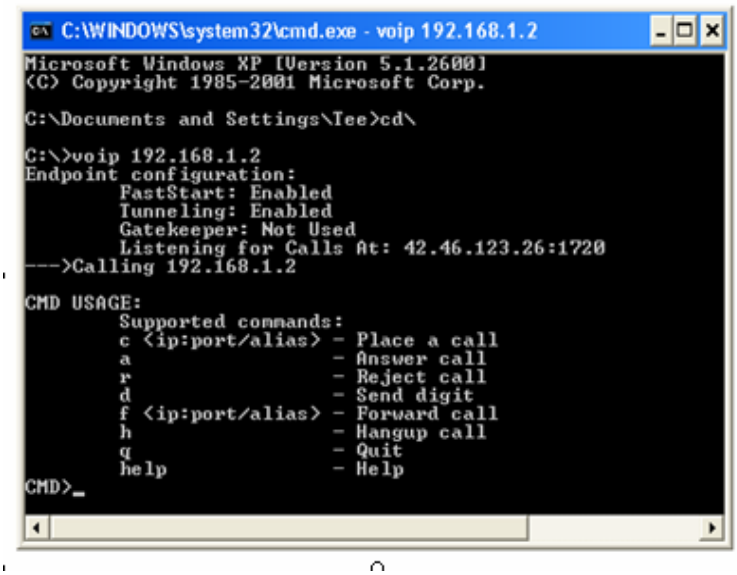
รูปที่ 6.5 โปรแกรมควบคุมโทรศัพท์ฝั่งไคลเอนท์

2. คลิกปุ่ม Connect เพื่อควบคุมให้โปรแกรม VoIP ฝั่งเซิร์ฟเวอร์ถูกเรียกขึ้นมาดังรูปที่6.5 และอยู่ในสถานะ Listen หรือสถานะรอรับคำสั่งจากฝั่งไคลเอนท์

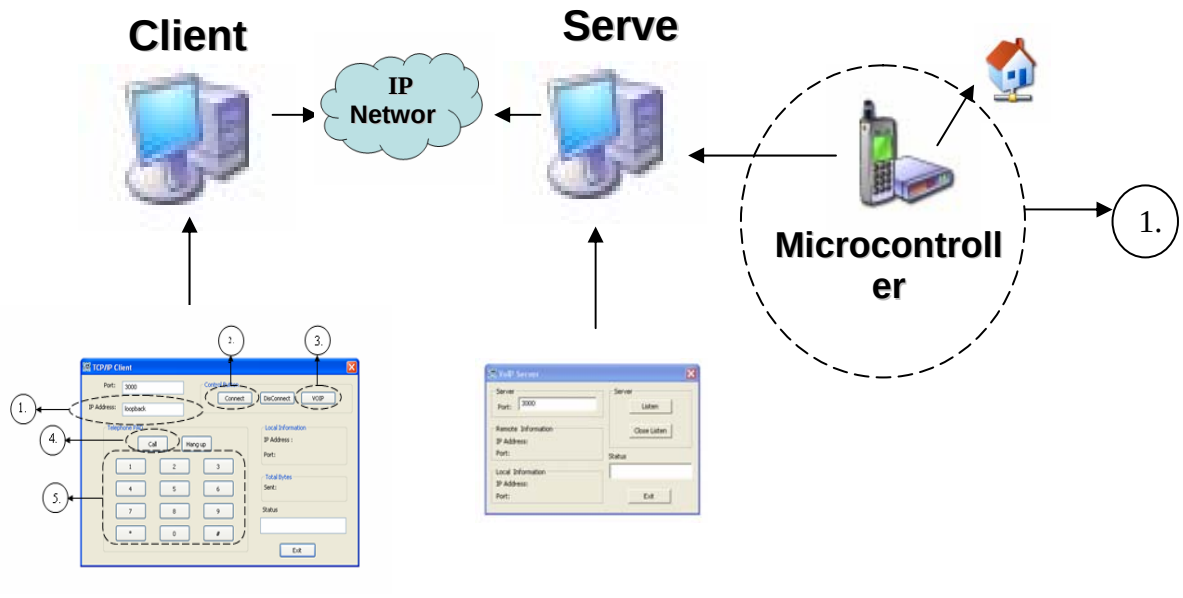


รูปที่ 6.6 โปรแกรม VoIP ฝั่งเซิร์ฟเวอร์

3. คลิกปุ่ม VoIP โปรแกรม VoIP ฝั่งไคลเอนท์จะถูกเรียกขึ้นมาดังรูปที่ 6.6 และทำการเชื่อมต่อไปยังเซิร์ฟเวอร์เพื่อให้ฝั่งไคลเอนท์และเซิร์ฟเวอร์สามารถส่งสัญญาณเสียงติดต่อกันได้



รูปที่ 6.7 โปรแกรม VoIP ฝั่งไคลเอนท์



รูปที่ 6.8 ไดอะแกรมขั้นตอนการติดต่อโดยควบคุมผ่านทางอินเตอร์เน็ท

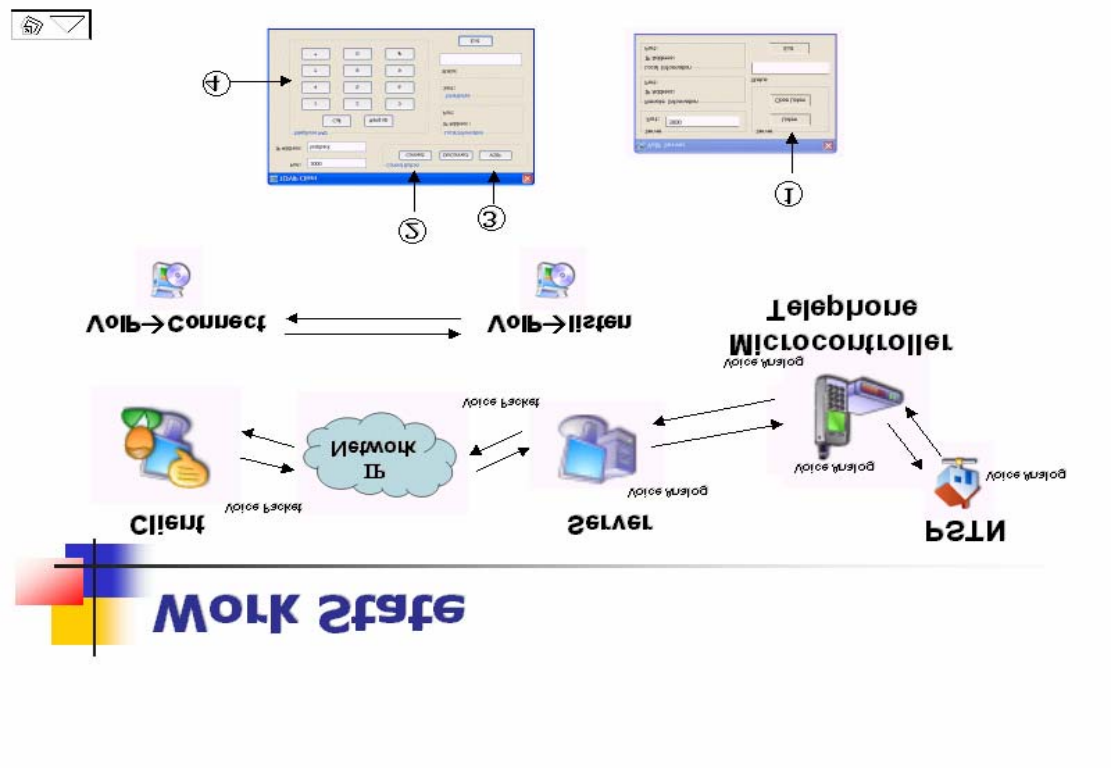
4. คลิกปุ่ม Call เพื่อควบคุมให้โทรศัพท์ฝั่งเซอร์ฟเวอร์ยกหู
5. กดหมายเลขปลายทางที่ต้องการติดต่อ

ฝั่งเซอร์ฟเวอร์

1. นำเลขที่ได้จากการกดฝั่งไคลเอนท์ไปประมวลผลโดยมีโปรแกรมไมโครคอนโทรลเลอร์ซึ่งอยู่ในส่วนของกล่องควบคุมฝั่งเซร์ฟเวอร์เป็นตัวตรวจสอบหมายเลขและควบคุมการกดเบอร์โทรศัพท์

ขั้นตอนการติดต่อโดยควบคุมผ่านทางอินเตอร์เน็ต

1. ติดตั้งโปรแกรมคอนโทรลทั้งฝั่งไคลเอนท์และเซอร์ฟเวอร์
2. กดปุ่ม listen บนโปรแกรมคอนโทรลฝั่งเซอร์ฟเวอร์ เพื่อรอฟังการติดต่อจากไคลเอนท์
3. กดปุ่ม connect บนโปรแกรมคอนโทรลฝั่งไคลเอนท์ เพื่อทำการเชื่อมต่อกับเซอร์ฟเวอร์ โปรแกรมการส่งเสียงผ่านอินเตอร์เน็ต หรือ VoIP ทางฝั่งไคลเอนท์ก็จะถูกรันขึ้นมา
4. กดปุ่ม VoIP บนโปรแกรมคอนโทรลฝั่งไคลเอนท์โปรแกรมการส่งเสียงผ่านอินเตอร์เน็ต หรือ VoIP ทางฝั่งไคลเอนท์ก็จะถูกรันขึ้นมา และพร้อมที่จะส่งเสียงผ่านอินเตอร์เน็ตได้
5. กดปุ่มเพื่อควบคุมโทรศัพท์ซึ่งอยู่ในส่วน Telephone Pad บนโปรแกรมคอนโทรลฝั่งไคลเอนท์
6. ฝั่งเซอร์ฟเวอร์ทำการตรวจสอบเบอร์ด้วยไมโคลคอนโทลเลอร์ และทำการกดเบอร์ปลายทาง
7. ฝั่งไคลเอนท์ทำการส่งเสียงซึ่งอยู่ในรูป Annalog ออกมา VoIP ทำการแปลงสัญญาณ Annalog ให้อยู่ในรูป Digital เรียกว่า Voice Packet
8. Voice Packet ถูกส่งออกไปบนเครือข่ายและไปยัง เซอร์ฟเวอร์
9. เซอร์ฟเวอร์ทำการแปลงข้อมูลกลับจาก Voice Packet เป็น Voice Annalog
10. เซอร์ฟเวอร์และไคลเอนท์สามารถสนทนากันผ่านเครือข่ายอินเตอร์เน็ตได้



รูปที่ 6.9 ไดอะแกรมขั้นตอนการติดต่อโดยควบคุมผ่านทางอินเตอร์เน็ต

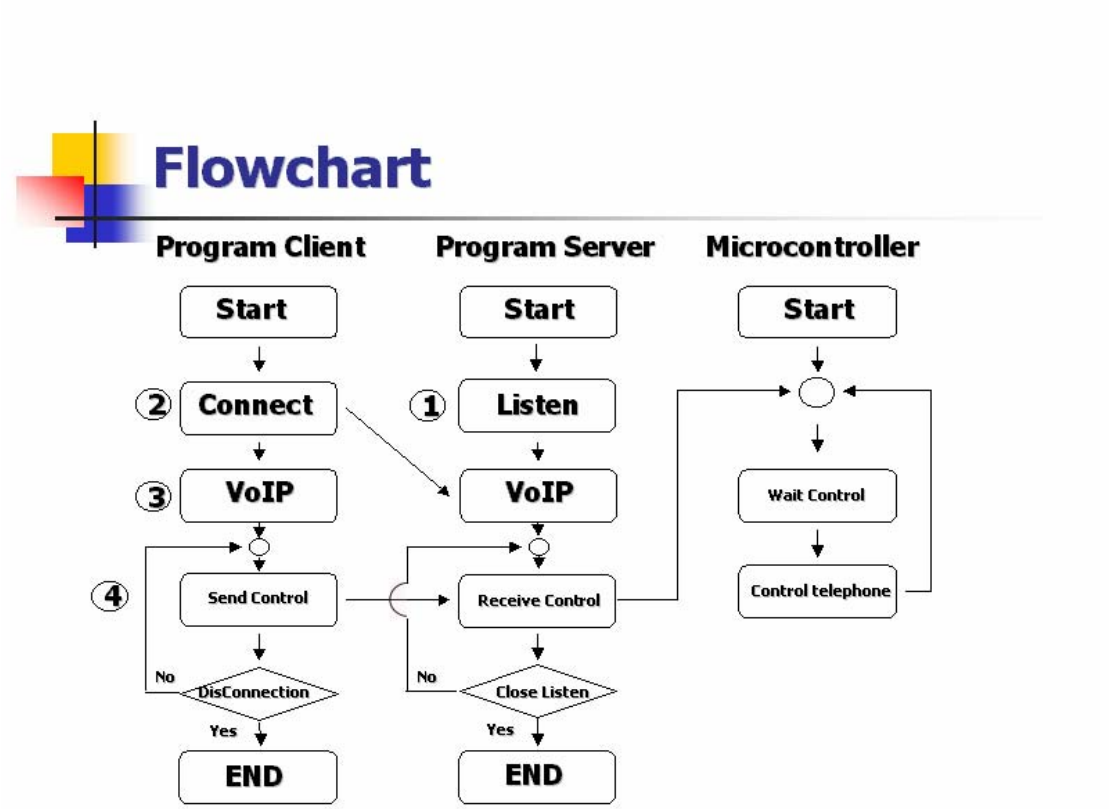
แสดงการทำงานตาม Flowchart

1. ฝั่งโปรแกรม Server จะเริ่มทำงานก่อนด้วยการรอฟังการร้องขอ การติดต่อมาจากทางด้านClient

2. เมื่อฝั่งClient ได้ทำการคอนเน็กต์ จะสั่งให้โปรแกรม VoIP ทางด้าน Server ทำการเปิดตัวเองขึ้นมาและอยู่ในสถานะรอฟังการเชื่อมต่อจากโปรแกรม VoIP ทางฝั่งClient

3. เมื่อเปิดโปรแกรมVoIP ฝั่ง Clientขึ้นมา โปรแกรม VoIP ด้านClient จะทำการเชื่อมต่อระหว่าง VoIP ทั้งสองด้านก็จะเกิดการเชื่อมต่อ Voice ก็จะถูกส่งผ่านNetwork

4. ทางฝั่งClient ก็จะมีการส่งคอนโทรลออกมาและถูกส่งผ่านไปยัง Network เมื่อโปรแกรมด้าน Server ได้รับคอนโทรล ก็จะทำการส่งข้อมูลออกทางพอร์ตอนุกรมไปควบคุมไมโครคอนโทรลเลอร์อีกที เมื่อแต่ละโปรแกรมได้รับคำสั่งและทำงานจบลงแล้วก็จะเข้าสู่สถานะรอฟังคำสั่งครั้งใหม่เสมอ จนกระทั่งมีคำสั่งวางสายหรือดิสคอนเน็กต์โดยไคลเอ็นต์หรือการยกเลิกการเชื่อมต่อจาก Server



รูปที่ 6.10 Flowchart แสดงการทำงาน

บทที่ 7
บทวิจารณ์และสรุป

บทสรุป

ปัจจุบันการติดต่อสื่อสารกระทำกันบนอินเตอร์เน็ตมากขึ้น เนื่องจากมีความสะดวก รวดเร็ว และประหยัด แต่การสร้างส่วนของการติดต่อได้นั้น เราจะต้องมีความรู้เรื่องโปรโตคอล เพื่อที่จะใช้กำหนดแนวทางในการเชื่อมต่อให้เพื่อให้ไปในทางเดียวกัน และมาตรฐานที่ใช้คือ H.323 ซึ่งก็สนับสนุนในการทำงานแบบ VoIP และนอกจากนี้ก็ยังต้องอาศัยโปรโตคอลตัวอื่นๆ ด้วย และสิ่งสำคัญคือตัวเข้ารหัสเสียงว่าจะนำชนิดใดมาใช้งานเพราะ คุณภาพเสียงที่ได้จะแตกต่างกันในแต่ละตัว ถ้าเลือกตัวเข้ารหัสที่มีคุณภาพเสียงที่ดี ก็อาจจะต้องใช้เครือข่ายที่มีความเร็วพอที่จะรองรับการใช้งานได้ ไม่เช่นนั้นก็จะเกิดการดีเลย์ของสัญญาณได้

วิจารณ์สิ่งที่ได้จากโครงงาน

สิ่งที่ได้จากการทำโครงงานนี้ คือการเข้าใจในหลักการพื้นฐานในการพัฒนาโปรแกรมบนระบบปฏิบัติการวินโดวส์ ด้วยโปรแกรม Microsoft Visual C+ .net ความเข้าใจการเขียนโปรแกรมด้วยภาษาแอสเซมบลี้เพื่อประยุกต์ใช้งานในการควบคุมไมโครคอนโทรลเลอร์ และยังมีความรู้ความเข้าใจในโปรโตคอลที่เกี่ยวข้องกับ VoIP ได้ศึกษาการเข้ารหัสเสียงตามมาตรฐานต่างๆของ ITU เช่น G.711 ชนิด u-law และ a-law , G.729 และอื่นๆ ซึ่งแต่ละอันจะให้คุณภาพเสียงที่แตกต่างกัน และมีความสามารถแตกต่างกัน เช่น G.711 สามารถลดเสียงเอ็กโค่ลงได้ เป็นต้น ในการทำโปรเจคได้ทดลองการเข้ารหัสแบบ G.711, G.7231, G.729, G.729A,GSM ปรากฏว่า การเข้ารหัสแบบ G.711 ในบางครั้งจะเกิดการดีเลย์ของสัญญาณมากกว่าชนิดอื่นๆ เนื่องมาจากความต้องการของขนาดบิทเรทที่มากกว่าชนิดอื่นๆ และการเข้ารหัสแบบอื่นๆ ก็จะให้คุณภาพเสียงที่ต่ำลงมาและมีเสียงก้อง แต่ไม่เกิดการดีเลย์ของสัญญาณ

ปัญหาอุปสรรคและแนวทางแก้ไข

ปัญหาที่สำคัญคือระบบโทรศัพท์ในโปรเจคจะได้รับสัญญาณรบกวนทางไฟฟ้าเข้ามาทำให้เราได้ยินเสียงรบกวนได้อย่างชัดเจน ในฝั่งปลายทางที่เราได้ทำการโทรหาแต่ไม่พบเสียงเอ็กโค่และ

ทางด้านต้นทางที่ควบคุมการโทรออกจะได้ยินเสียงเอ็กโค่อย่างชัดเจน และจะพบเสียงเอ็กโค่ในฝั่งต้นทาง ในกรณีนี้ได้ลองทำการเปลี่ยนว้อยซ์โวโคเด่อร์หลายๆแบบ พบว่าไม่สามารถทำให้เสียงเอ็กโค่หายไปได้แต่ จะทำให้เสียงเอ็กโค่มาเร็วขึ้นอีกเล็กน้อย และปัญหาอีกอย่างในการสร้างโปรเจ็คคือเราไม่สามารถที่จะหาข้อมูล เนื้อหาที่เกี่ยวข้องเพื่อที่จะนำมาพัฒนาโปรแกรมส่งสัญญาณเสียงบนไอพี ได้ง่ายและหลากหลาย จึงต้องนำโปรแกรมที่เป็นรูปแบบโอเพ่นซ์ซอร์สนำมาแก้ไขปรับปรุงให้เข้ากับโครงงาน และปัญหาทางด้านความไม่เสถียรของตัวไมโครคอนโทรลเลอร์ซึ่งเมื่อนำขาพอร์ตมาใช้งานจำนวนมากๆ การใช้งานในบางครั้งก็ไม่สามารถสั่งการบางขาให้ใช้งานได้จนคาดว่าโครงงานส่วนอื่นเสียก็ทำให้ปรับส่วนอื่นและตัวโปรแกรมหลายรอบ จึงเป็นการเสียเวลาซึ่งได้ปัญหานี้พบได้บ่อยกับไมโครคอนโทรเลอร์หลายๆตัว และจะมีโอกาสพบปัญหาเป็นครั้งคราว สุดท้ายปัญหาเรื่อง การวางแผนการทำงานได้ไม่ค่อยดี เพราะการทำงานบางครั้งพบปัญหาและความต้องการที่จะแก้ไขงานบางส่วนและงานออกมาให้สมบูรณ์ จึงไม่สามารถทำให้บรรลุจุดมุ่งหมายได้ทั้งหมด

แนวทางการพัฒนาต่อ

ในการทำโปรเจ็คซึ่งเกี่ยวกับระบบสัญญาณทางด้านเสียงนั้น สิ่งที่ต้องระวังคือสัญญาณรบกวนที่จะเข้ามาในระบบ ซึ่งต้องหาทางป้องกันและแก้ไขปัญหาเสียงรบกวนไม่ให้เข้ามาในระบบ เพราะจะทำให้คุณภาพเสียงที่ได้ต่ำลงมาก และสิ่งที่สำคัญในการพัฒนาอีกด้านคือตัวโปรแกรมด้าน VoIP ซึ่งมีโปรโตคอล 2 ชนิด คือ H.323 และ SIP ในโปรเจคนี้ได้ใช้โปรโตคอลชนิด H.323 ซึ่ง SIP จะเป็นโปรโตคอลของทางไมโครซอร์ฟ และมีตัวโปรแกรมชนิด SDK ช่วยให้เราสามารถเขียนโปรแกรมได้ง่ายขึ้น จะทำให้เราสามารถพัฒนาขึ้นมาใช้งานได้เองไม่ต้องใช้โอเพ่นซอร์ส และในส่วนโทรศัพท์อาจเปลี่ยนเป็นการใช้โมเด็มที่มีคุณสมบัติสนับสนุนทางเสียงมาใช้งานแทน แต่การพัฒนาโปรแกรมไปในทางด้านนี้ผู้พัฒนาจะต้องมีข้อมูลและความเข้าใจในการพัฒนาโปรแกรมค่อนข้างมาก

บรรณานุกรม

ยุทธนา ลีลาศวัฒนกุล, คู่มือการเขียนโปรแกรมและใช้งาน Visual C++ .NET , อินโฟเพรส,2546

ปรเมษฐ์ ประนยานันทน์, คู่มือและการประยุกต์ใช้งานไมโครคอนโทรลเลอร์ MCS-51, ซีเอ็ดยูเคชั่น, 2536

Douskalis,Bill, IP telephony : the integration of Robust VoIP services, Prentice Hall PTR, c2000

สุวัฒน์ ปุณณชัยยะ, เปิดโลกของ TCP/IP และโปรโตคอลของอินเตอร์เน็ต, โปรวิชั่น, 2543

http://www.dcomputer.com

http://www.obj-sys.com/index.shtml

http://www.thaiio.com

http://www.ku.ac.th

ประวัติผู้เขียน

นายสาริษฐ์ วีรมหาวงศ์ ภูมิลำเนาอยู่ที่ บ้านเลขที่ 22/221 ถนนบางนา-ตราด เขตบางนา จังหวัดกรุงเทพมหานคร จบการศึกษาระดับมัธยมศึกษาตอนปลายจากโรงเรียนราชดำริ กรุงเทพมหานคร ปีการศึกษา 2543 ปัจจุบันกำลังศึกษาอยู่ชั้นปีที่ 5 สาขาวิศวกรรมโทรคมนาคม สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี จังหวัดนครราชสีมา

นางสาวอัจฉรา ปรีดามีสุข ภูมิลำเนาอยู่ที่ บ้านเลขที่ 166 หมู่ 8 ต.เขากระปุก อ.ท่ายาง จังหวัดเพชรบุรี จบการศึกษาระดับมัธยมศึกษาตอนปลายจากโรงเรียนหนองชุมแสงวิทยา ปีการศึกษา 2544 ปัจจุบันกำลังศึกษาอยู่ชั้นปีที่ 5 สาขาวิศวกรรมโทรคมนาคม สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี จังหวัดนครราชสีมา

ภาคผนวก ก. Visual C++ .Net

การเขียนโปรแกรมด้วย Visual c++ .NET

Visual C++ .NET นั้นสามารถเขียนโปรแกรมเพื่อสร้างแอพพลิเคชั่นได้แทบทุกรูปแบบ ไม่ว่าจะเป็นโปรแกรมธรรมดาที่รันบน Windows เรื่อยไปจนถึงโปรแกรมที่ทำงานภายใต้แนวความคิด .NET แต่แนวทางที่เหมาะสมก็คือ การเขียนโปรแกรมแบบดั้งเดิม หรือเรียกว่า Native Code หรือ Unmanaged Code ซึ่งก็มีข้อดีก็คือ โปรแกรมที่ได้จะมีขนาดเล็ก ทำงานได้อิสระและรวดเร็วมาก เมื่อรันภายใต้ ดอสหรือวินโดวส์

การสร้างโปรแกรม

ในหัวข้อนี้เราจะสร้างแอพพลิเคชั่นอย่างง่ายโดยใช้เครื่องมือ ApplicationWizard (exe) ซึ่งจะทำให้ได้แอพพลิเคชั่นตามต้องการ แอพพลิเคชั่นที่ได้นี้จะประกอบด้วยคลาสพร้อมกับโค้ดต่างๆ

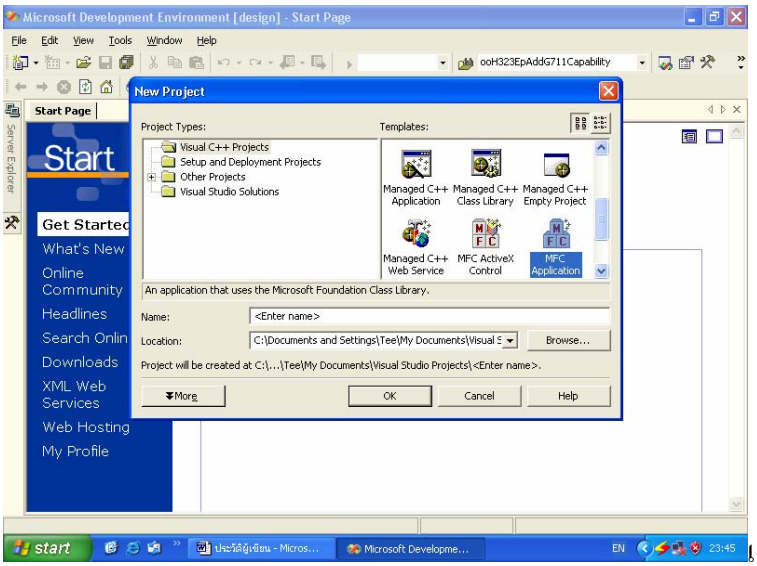
ข้อดีของการใช้ AppWizard (exe) ก็คือ ทำให้เราประหยัดเวลาในการเขียนโค้ดจำนวนมาก เพราะหลังจากที่มันสร้างโค้ดให้เราแล้ว เราก็เพียงแต่เขียนโค้ดเพิ่มอีกไม่กี่บรรทัด แล้วคอมไพลล์ไฟล์ทั้งหมด และถ้าไม่มีอะไรผิดพลาดเราก็เพียงแค่คลิกปุ่มรัน โปรแกรมก็สามารถใช้งานได้ทันที

สำหรับรายละเอียดการสร้างแอพพลิเคชั่นด้วย AppWizard(exe) มีรายละเอียดดังนี้

การใช้ AppWizard สร้างโค้ดหลักให้

การที่สร้างโปรแกรมใหม่ เราต้องสร้างโปรเจ็กต์ใหม่ เราเริ่มต้นด้วยการเรียก Microsoft Developer Studio และทำตามหัวข้อดังต่อไปนี้

1. คลิกเมนู File > New ใน Visual Studio .NET (หรือกดปุ่ม < Ctrl + Shift + N>) เราจะสร้างไดอะล็อกที่ชื่อว่า New Project ขึ้นมา
2. เลือกชนิดโปรเจ็กต์เป็น Visual C++ Projects ในช่อง Project Types
3. เลือกชนิดแอพพลิเคชั่นเป็น MFC Application ในช่อง Templates
4. พิมพ์ชื่อโปรเจ็คใหม่ และกำหนดโฟลเดอร์ตามต้องการ ตามรูป



รูปที่ ก.1 สร้างโปรแกรมใหม่

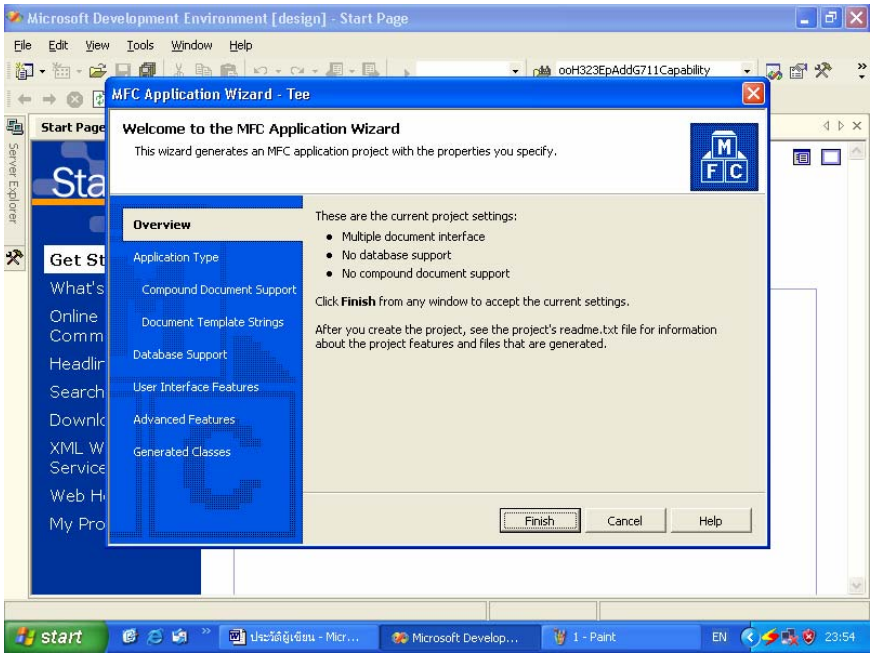
5. คลิกปุ่ม OK จะขึ้นไดอะล็อกซ์เป็น MFC Application Wizard
6. ที่แท็บ Overview จะสรุปรายละเอียดต่างๆ ดังรูป โดยดีฟอลต์ที่จะนำมาใช้ในการสร้างโปรเจ็กต์นี้เมื่อคลิกปุ่ม Finish รายละเอียดมีดังนี้

สร้างโปรแกรมเป็นแบบ Multiple Document Interface (MDI)

ไม่มีการสนับสนุนในการติดต่อกับฐานข้อมูล

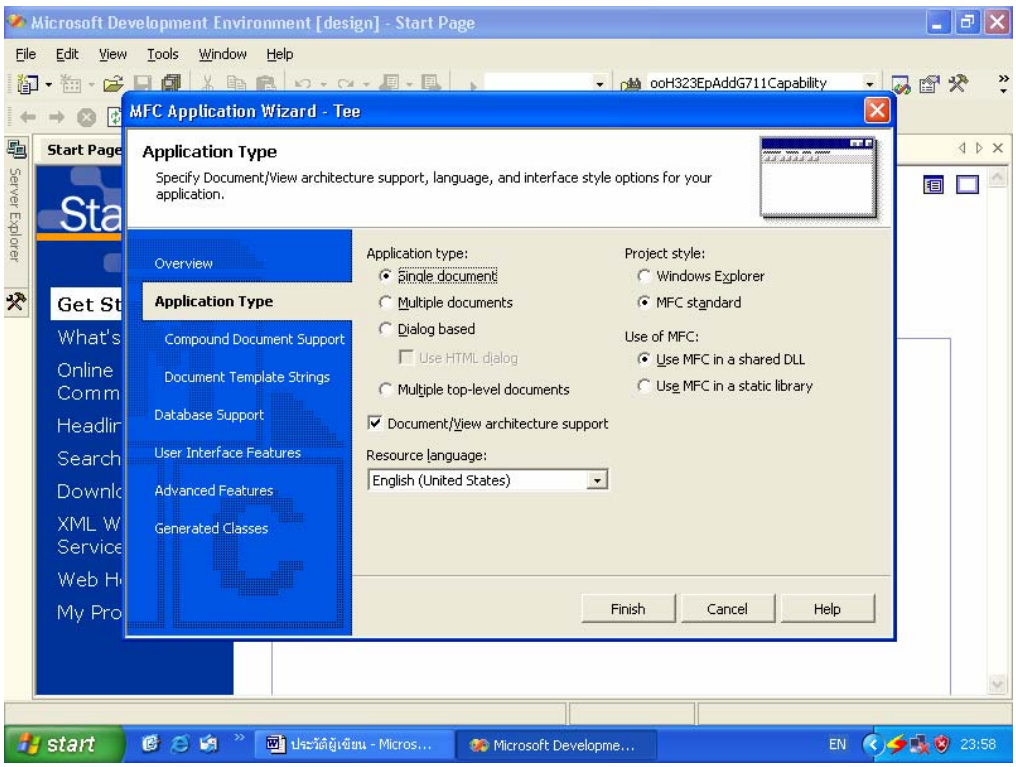
ไม่สนับสนุน Compound Document เช่น Object Linking and Embedding(OLE)

เป็นต้น



รูปที่ ก.2 แท็บ Overview

7. คลิกแท็บ Application Type เพื่อเลือกเกี่ยวกับการสนับสนุนสถาปัตยกรรมด็อกคิวเม็นต์/วิว ภาษาที่ต้องการใช้ และรูปแบบอินเตอร์เฟสที่ต้องการนำมาใช้ในโปรแกรมที่เราต้องการ



รูปที่ ก.3 แท็บ Application Type

รายละเอียดแต่ละหัวข้อมีดังนี้

ที่หัวข้อ Application Type จะเป็นตัวเลือกรูปแบบแอพพลิเคชั่นที่จะสร้างขึ้นมา

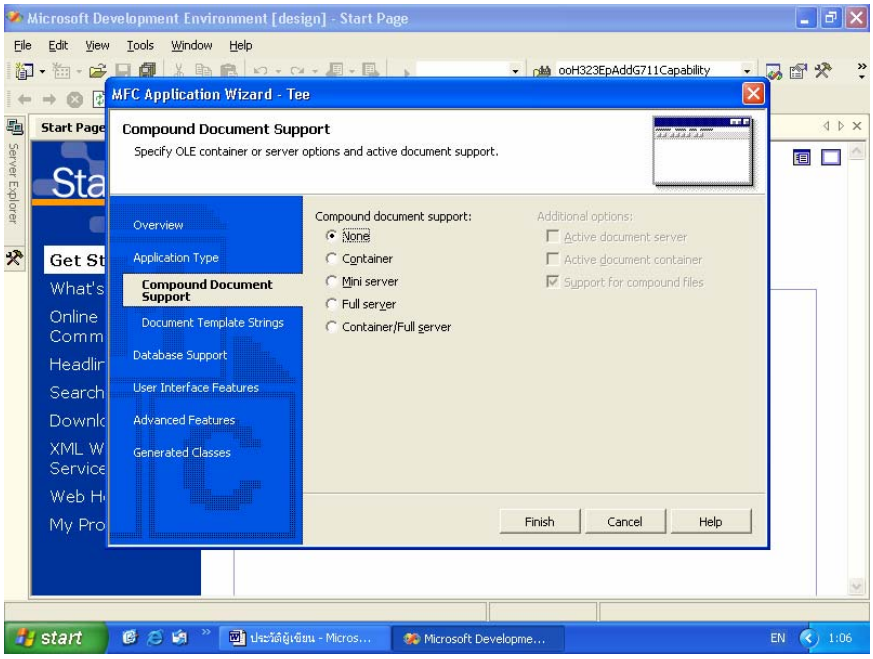
- Single Document ซึ่งเป็นแอพพลิเคชั่นที่สามารถแสดงผลได้ทีละหนึ่งด็อกคิวเม็นต์ เช่นโปรแกรม Notepad เป็นต้น
- Multiple document จะเป็นการสร้างแอพพลิเคชัน ท่าสมารถแสดงผลได้ทีละหลายๆ ด็อกคิวเม็นต์ในเวลา เดียวกัน เช่น ไมโครซอฟท์เวิร์ด
- Dialog based จะเป็นการสร้างแอพพลิเคชั่นที่เป็นไดอะล็อก ซึ่งก็คือมีวินโดวส์เดียวที่ทำหน้าที่โต้ตอบตอบกับผ้ใช้แบบง่ายๆ เช่นบการรับข้อมูล การบอกข้อมูล หรือเป็น การเตือนให้ผู้ใช้งานทราบ เป็นต้น

สำหรับหัวข้อ Document/View architecture support จะเป็นการกำหนดให้แอพพลิเคชั่นที่สร้างขึ้นมาใหม่นี้ใช้คลาสด็อกคิวเม็นต์ และคลาสวิวเป็นคลาสวิวมีหน้าที่หลักในการนำข้อมูลจากข้อมูลจากคลาสด็อกคิวเม็นต์ หรืออื่นๆ มาแสดงบนจอภาพ

ส่วนหัวข้อ Resource language จะเป็นการเลือกภาษาสำหรับใช้ใน Resource ของโปรแกรมนี้ในที่นี้จะเป็นภาษาอังกฤษตลอด

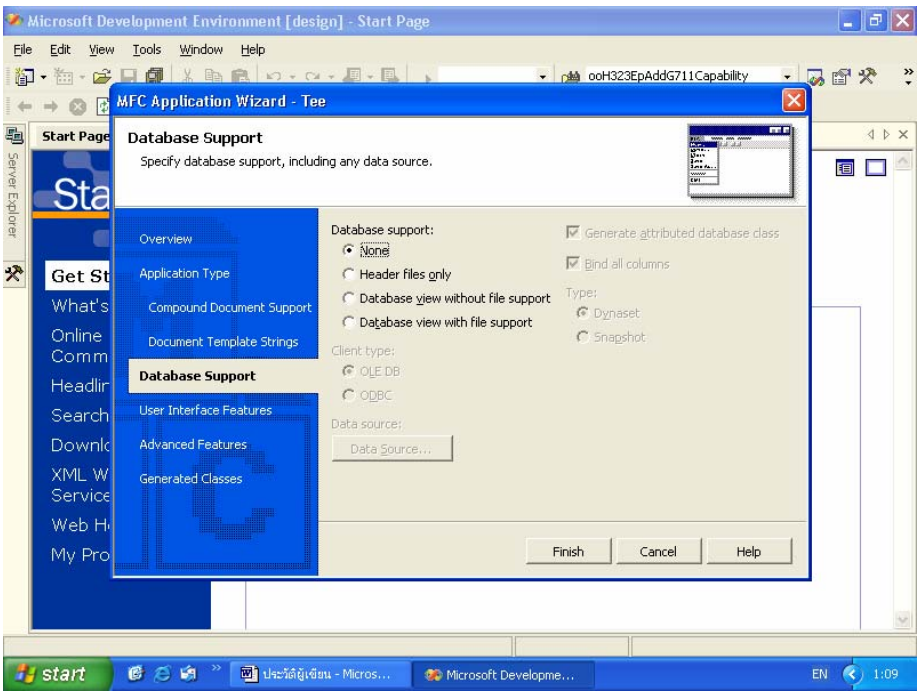
ที่หัวข้อ Project style 0เป็นการเลือกรูปแบบสถาปัตยกรรม Windows Explorer หรือ MFC โดยดีฟอลต์จะเป็น MFC ส่วน Windows Explorer จะคล้ายกับโปรแกรม Windows Explorer คือมีหน้าต่างซ้ายขวา ซึ่งหน้าต่างด้านซ้ายจะเป็นคลาส CtreeView และด้านขวาจะเป็น Clistview

8. คลิกที่แท็บ Compound Document Support เพื่อให้แอพพลิเคชั่น สนับสนุนเกี่ยวกับ OLE ในที่นี้จะใช้ค่าดีฟอลต์คือ คลิกที่ตัวเลือก none ดังรูป



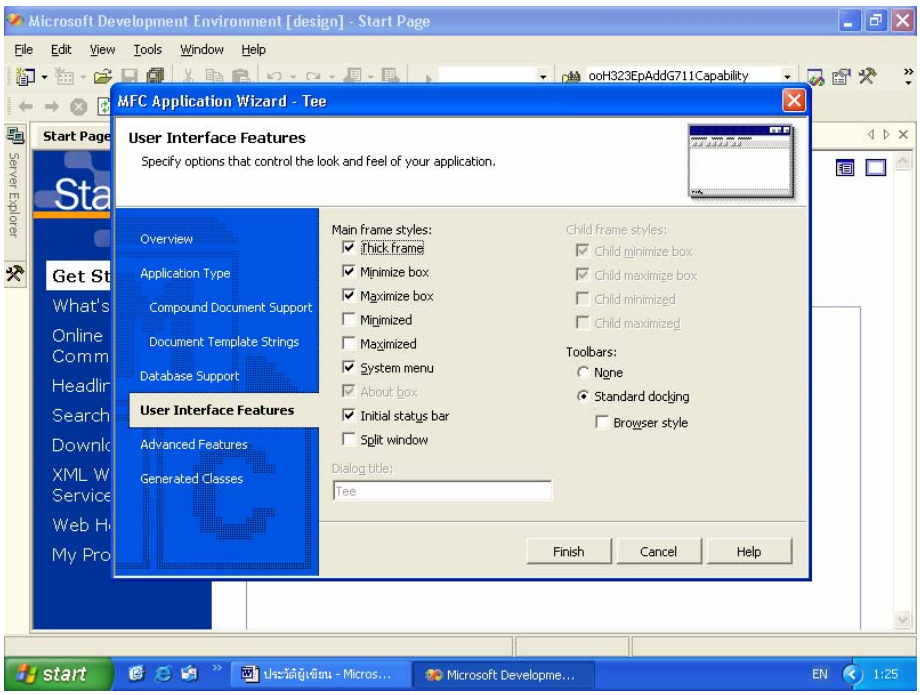
รูปที่ ก.4 แท็บ Compound Document Support

9. คลิกแท็บ Document Template String เพื่อตั้งชื่อนามสกุลสำหรับด็อกคิวเมนต์ เมื่อตั้งชื่อเสร็จแล้ว จะถูกบันทึกลงรีจิสเตอร์ของระบบปฏิบัติการวินโดวส์
10. แท็บ Database support จะเป็นเรื่องการสนับสนุนฐานข้อมูลต่าง ซึ่งค่าดีฟอลต์ ซึ่งก็คือรายการ None



รูปที่ ก.5 แท็บ Database support

11. แท็บ User Interface Features เป็นการเลือกรูปแบบในการติดต่อกับผู้ใช้ เช่นกลุ่มเกี่ยวกับเฟรม ได้แก่กรอบแบบหนา มีบ็อกซ์ Minimize และ Maximize เป็นต้น Main frame style จะเป็นการกำหนดคุณลักษณะของเฟรมหลักในแอพพลิเคชั่น ได้แก่ Thick frame กรอบของโปรแกรมเป็นเส้นหนา, Minimize box มีบ็อกซ์ทำให้โปรแกรมมีขนาดเล็กสุด Maximize box มีบ็อกซ์ทำให้โปรแกรมขนาดมีใหญ่สุด System menu มีเมนูระบบ About box มีไดอะล็อกซ์, Initial status bar ,สเตตัส, Split window สามารถแยกวินโดวส์ เป็นต้น ที่กลุ่ม Child frame styles จะเป็นการกำหนดกรอบสำหรับเฟรมลูก และจะใช้ได้ก็ต่อเมื่อเรา คลิกเลือก Application Type เป็นแบบ MDI เท่านั้น ที่กลุ่ม Toolbars จะเป็นการกำหนดรูปแบบทูลบาร์ โดยดีฟอลต์แล้วจะเป็นแบบ Standard docking ซึ่งจะมีปุ่มกด new , save , open เป็นหลัก



รูปที่ ก.6 แท็บ User Interface Features

12. แท็บ Advanced Features เพื่อเพิ่มการสนับสนุน เช่น ดารเพิ่ม Help, ActiveX Control เป็นต้น ซึ่งมีรายการละเอียดดังนี้

    Context-Sensitive help จะสร้างระบบความช่วยเหลือที่เกิดจากการกดปุ่ม F1

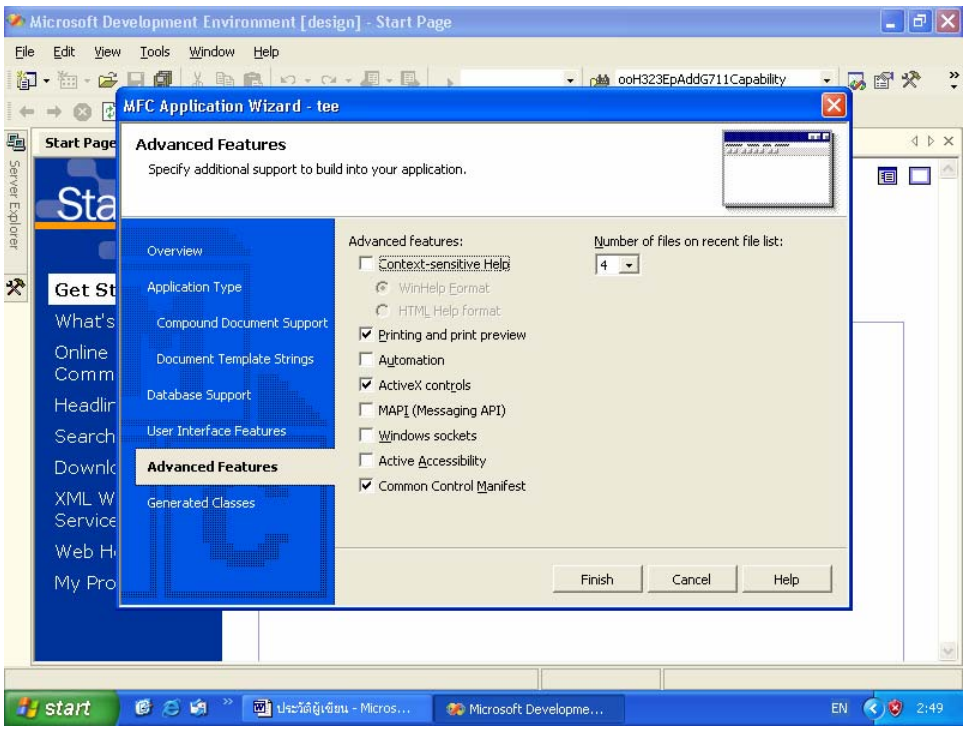
    Printing and print preview จะสร้างโค้ดสำหรับจัดการกับการพิมพ์

    ActiveX controls จะเพิ่มโค้ดเพื่อสนับสนุนการใช้คอนโทรล ActiveX

    Windows socket จะเพิ่มโค้ดเพื่อสนับสนุนการใช้งานคลาส Socket

    Common Control Manifest จะสร้างรูปแบบคอนโทรลต่างๆ ที่ติดต่อกับผู้ใช้ให้เหมือนกับที่แสดงใน Windows XP

    Number of files on recent file list จะแสดงจำนวนไฟล์ในลิสต์รายการเมนู file ในที่นี้เราใช้ค่าดีฟอลต์หมด



รูปที่ ก.7 แท็บ Advanced Features

13. แท็บ Classes ซึ่งเป็นขั้นตอนสุดท้ายเพื่อเลือกคลาสหลักที่จะใช้ในคลาสวิว และเปลี่ยนชื่อคลาส ไฟล์ต่างๆ เป็นต้น ในขั้นตอนนี้ให้ใช้ค่าดีฟอลต์ หมดเช่นกันดังรูป โดยที่หัวข้อ Generated Classes บนลิสต์บ็อกซ์ หมายถึง Appwizard จะสร้างคลาสหลักๆ ซึ่งได้แก่

คลาสแอพพลิเคชั่น (CFirstAppApp)

คลาสเมนเฟรม (CMainFrame)

คลาสด็อกคิวเม็นต์ (CFirstAppDoc)

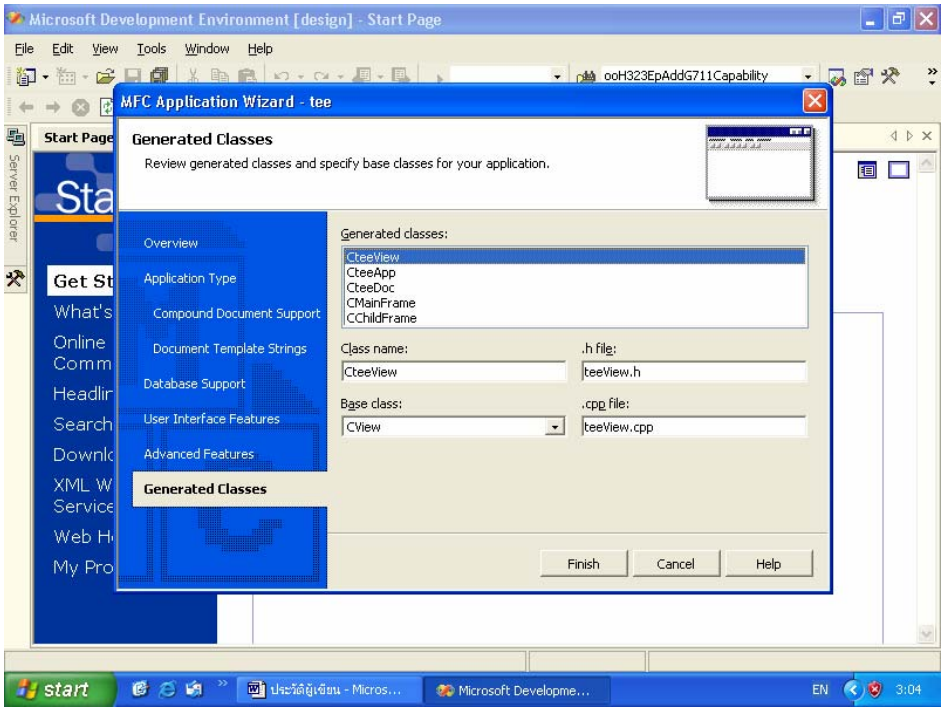
คลาสวิว (CFirstAppView)

ส่วนรายละเอียดอื่นๆ ของขั้นตอนนี้ ได้แก่

- ช่อง Class name: แสดงชื่อคลาสเมื่อคลิกชื่อคลาสในช่อง List Box แล้ว และถ้าต้องการเปลี่ยนชื่อคลาสให้แก้ไขที่ช่องนี้

- ช่อง .h file: และ .cpp file: แสดงชื่อไฟล์เมื่อคลิกชื่อคลาสในช่อง List Box แล้ว และถ้าต้องการเปลี่ยนชื่อไฟล์ให้แก้ไขที่ช่องนี้ ยกเว้นคลาสแอพพลิเคชั่นซึ่งไม่สามารถเปลี่ยนแปลงชื่อไฟล์ได้

- ช่อง Base class: ใช้ในกรณีที่คลิกชื่อคลาสวิวในช่อง List Box แล้วเท่านั้น ซึ่งจะแสดงรายชื่อคลาสต่างๆ เช่น คลาส CScrollView คลาส CHtmlView หรือถ้าเป็นดีฟอลต์แล้วจะเป็นคลาส CView เป็นต้น รายละเอียดการใช้คลาสเหล่านี้จะกล่าวอีกครั้งในบทต่อไป



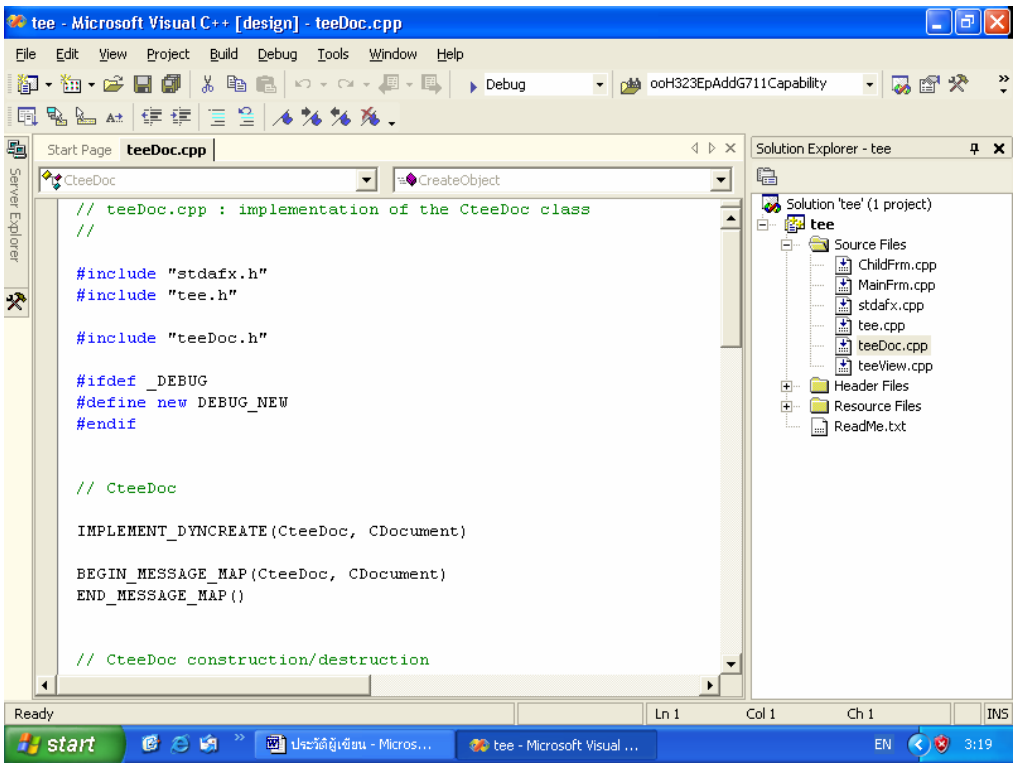
รูปที่ ก.8 แท็บ Generated Classes

ในกรณีที่เราคลิก หรือเลือกรายการผิด หรือต้องการกลับไปดูให้แน่ใจอีกครั้ง ก็สามารถทำได้โดยคลิกแท็บนั้นๆ

14. ท้ายสุดคลิกปุ่ม Finish ซึ่ง Visual C++ .NET จะสร้างโค้ดหลักให้ตามค่าที่เราเลือกและกำหนดมา

การแก้ไขและการเพิ่มซอร์สโค้ด

หากเราต้องการแก้ไข หรือเพิ่มซอร์สโค้ดอีกเล็กน้อย เพื่อให้แอพพลิเคชั่นทำงานสมบูรณ์ยิ่งขึ้น สามารถทำได้โดยการเพิ่มโค้ดที่คลาสด็อกคิวเม็นต์ ซึ่งจะเป็นเรื่องเกี่ยวกับข้อมูล ส่วนการเพิ่มโค้ดที่คลาสวิว ซึ่งจะนำข้อมูลในคลาสด็อกคิวเม็นต์มาแสดงออกบนจอภาพ



รูปที่ ก.9 การแก้ไขและการเพิ่มซอร์สโค้ด

Resource และคอนโทรลเบื้องต้น

คุณลักษณะเด่นที่เห็นได้อย่างชัดเจนในการเขียนโปรแกรม Windows คือการใช้ Resource ซึ่ง Resource จะอนุญาตให้เราจัดการกับส่วนติดต่อกับผู้ใช้โปรกแกรมของเรา

Resource สามารถถูกเปลี่ยนแปลงได้โดยปราศจากผลกระทบโดยตรงกับโค้ดในแอพพลิเคชั่นด้วยคุณลักษณะนี้ ทำให้การใช้และการทำความเข้าใจเกี่ยวกับ Resource เป็นสิ่งสำคัญสำหรับโปรแกรมเมอร์ที่ต้องการสร้างแอพพลิเคชั่นเพื่อทำงานกับ Windows

Resource เป็นส่วนสำคัญของแอพพลิเคชั่น โดย Resource จะมีชุดของการติดต่อกับผู้ใช้ต่างๆ เช่น เมนู, ไอคอน, ไดอะล็อกซ์ และอื่นๆ ซึ่งถูกใช้เพื่อทำงานกับผู้ใช้เพื่อแสดงผล และตีความข้อมูลจากโปรแกรม

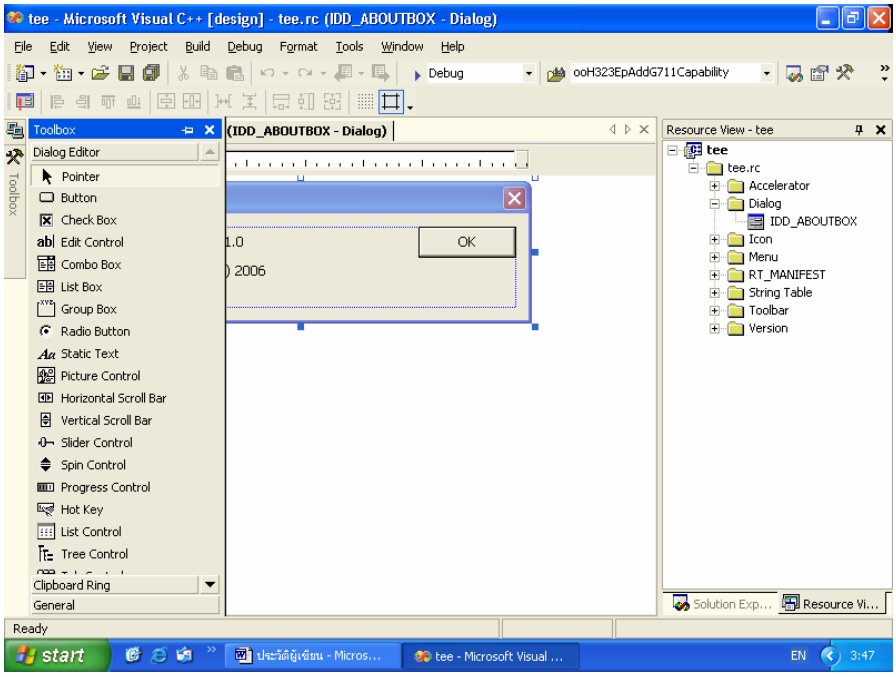
ชนิดของ Resource

Resource ใน Windows มีหลายชนิด ดังนี้

Dialog Box	เป็นฟอร์มที่ประกอบไปด้วยคอนโทรลต่างๆ เพื่อสามารถให้ผู้ใช้ป้อนข้อมูล เข้าไปในตัวโปรแกรม หรือรับข้อมูลเข้ามา
Menu	เป็นรายการคำสั่งต่างๆ ที่อยู่ด้านบนของวินโดวส์
String Table	เป็นรายการค่าคงที่ String ที่จะนำมาใช้ในโปรแกรมของเรา
Accelerator Key	เอาไว้สร้าง Short key
Cursor	เป็นออบเจ็กต์ตัวชี้ที่แสดงอยู่บนจอภาพ
Icon	เป็นเครื่องหมายของโปรแกรม
Bitmap	เป็นรูปภาพที่นำมาใช้กับแอพพลิเคชั่นของเรา

แนะนำคอนโทรลเบื้องต้น

คอนโทรลมักใช้คู่กับไดอะล็อกเสมอ คอนโทรลสามารถใช้กับคลาสวิวก็ได้ คอนโทรลสามารถแบ่งได้เป็นคอนโทรลมาตรฐาน และคอนโทรลทั่วไป โดยที่คอนโทรลมาตรฐานเป็นคอนโทรลที่ถูกสร้างภายในระบบปฏิบัติการวินโดวส์



รูปที่ ก.9 แนะนำคอนโทรลเบื้องต้น

ภาคผนวก ข. Source code

Client Programs

ASock.cpp : implementation file

```
#include "stdafx.h"
#include "WinSock.h"
#include "ASock.h"
#include "WinSockDlg.h" //

CASock::CASock()
{
}

CASock::~CASock()
{
}

void CASock::OnClose(int nErrorCode)
{
        if ( nErrorCode == 0 )
                ((CWinSockDlg*)m_pDwnd)->Closed();
        CAsyncSocket::OnClose(nErrorCode);

}
void CASock::OnReceive(int nErrorCode)
{
        CString strRecvMsg;
        char buffer[4096];
        int nRecv;
        nRecv = Receive(buffer, 4096);
```

```
	switch (nRecv)
	{
		case 0:
			Close();
				break;
		case SOCKET_ERROR: // Error occured
			if (GetLastError() != WSAEWOULDBLOCK)
			{
			AfxMessageBox ("Error occurred!!!");
			Close();
			}
			break;
		default: // No Error
			buffer[nRecv] = NULL
				CString strTemp(buffer);
				strRecvMsg = "R: " + strTemp;
				((CWinSockDlg*)m_pDwnd)-
	>m_listMessage.AddString(strRecvMsg);
	}
		CAsyncSocket::OnReceive(nErrorCode);
	}

void CASock::SetParent(CDialog* pDwnd)
{
	m_pDwnd = pDwnd;
}

void CASock::OnConnect(int nErrorCode)
{
	the base class
```

```
if (0 != nErrorCode)
{
	switch( nErrorCode )
	{
		case WSAEADDRINUSE:
			AfxMessageBox("IP Address นี้มีการใช้ไปแล้ว");
			break;
		case WSAEADDRNOTAVAIL:
			AfxMessageBox("IP Address นี้ไม่ว่างในเครื่องโลคอล");
			break;
		case WSAECONNREFUSED:
			AfxMessageBox("มีการพยามยามเชื่อมต่อ แต่ถูกปฏิเสธ หรือ
เซอร์เวอร์ยังไม่ทำงาน");
			break;
		case WSAEDESTADDRREQ:
			AfxMessageBox("ต้องการ IP Address  ที่ปลายทาง");
			break;
		case WSAEISCONN:
			AfxMessageBox("ซ็อคเก็ตถูกเชื่อมต่อไว้แล้ว");
			break;
		case WSAENETUNREACH:
			AfxMessageBox("เน็คเวิร์คไม่สามารถไปได้จากโฮสต์ ณ เวลานี้
หรือไม่มีเส้นทางที่จะไป");
			break;
		case WSAENOBUFS:
			AfxMessageBox("ไม่มีบัฟเฟอร์ หรือบัฟเฟอร์เต็ม ซ็อคเก็ตไม่
สามารถถูกเชื่อมต่อได้");
			break;
		case WSAENOTCONN:
			AfxMessageBox("ซ็อคเก็ตไม่ได้ถูกเชื่อมต่อ");
```

```
                    break;
                case WSAETIMEDOUT:
                    AfxMessageBox("พยามยามที่จะเชื่อมต่อโดยใช้ระยะเวลาหนึ่ง
แล้ว ไม่สามารถสร้างการเชื่อมต่อได้");
                    break;
                default:
                    char strError[256];
                    wsprintf(strError, "Connect error, Error Conde: %d",
nErrorCode);
                    AfxMessageBox(strError);
                    break;
            }
            AfxMessageBox("Plese click DisConnect and try connect again.");
        }
        else    ((CWinSockDlg*)m_pDwnd)->Connected();
        CAsyncSocket::OnConnect(nErrorCode);
}
```

ASock.h : implementation file

```
#pragma once
class CASock : public CAsyncSocket
{
public:
	CASock();
	virtual ~CASock();
	virtual void OnClose(int nErrorCode);
	virtual void OnReceive(int nErrorCode);
	void SetParent(CDialog* pDwnd);
protected:
	CDialog* m_pDwnd;
public:
	virtual void OnConnect(int nErrorCode);
};
```

WinSockDlg.cpp : implementation file

```
#include "stdafx.h"
#include "WinSock.h"
#include "WinSockDlg.h"
#include ".\winsockdlg.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#endif

class CAboutDlg : public CDialog
{
public:
	CAboutDlg();
	enum { IDD = IDD_ABOUTBOX };
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
protected:
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
}
```

```
BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
END_MESSAGE_MAP()

CWinSockDlg::CWinSockDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CWinSockDlg::IDD, pParent)
	, m_intPorts(3000)
	, m_strloAddIP(_T(""))
	, m_strloPort(_T(""))
	, m_strSendTotalMessages(_T(""))
	, m_strSendMessages(_T(""))
	, m_intCountSentByte(0)
	, m_strAddIP(_T("loopback"))
{
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CWinSockDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	DDX_Control(pDX, IDC_LISTMESSAGES, m_listMessage);
	DDX_Text(pDX, IDC_PORTS, m_intPorts);
	DDX_Text(pDX, IDC_LOADDIP, m_strloAddIP);
	DDX_Text(pDX, IDC_LOPORT, m_strloPort);
	DDX_Text(pDX, IDC_SENTTOTALBYTES, m_strSendTotalMessages);
	DDX_Text(pDX, IDC_SENDMESSAGE, m_strSendMessages);
	DDX_Text(pDX, IDC_ADDIP, m_strAddIP);
}
```

```
BEGIN_MESSAGE_MAP(CWinSockDlg, CDialog)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_BN_CLICKED(IDCANCEL, OnBnClickedCancel)
	ON_BN_CLICKED(IDC_BUTTON_SEND, OnBnClickedButtonSend)
	ON_BN_CLICKED(IDC_CONNECT, OnBnClickedConnect)
	ON_BN_CLICKED(IDC_DISCONNECT, OnBnClickedDisconnect)
	ON_BN_CLICKED(IDC_CALL, OnBnClickedCall)
	ON_BN_CLICKED(IDC_HANG, OnBnClickedHang)
	ON_BN_CLICKED(IDC_ONE, OnBnClickedOne)
	ON_BN_CLICKED(IDC_TWO, OnBnClickedTwo)
	ON_BN_CLICKED(IDC_THREE, OnBnClickedThree)
	ON_BN_CLICKED(IDC_FOUR, OnBnClickedFour)
	ON_BN_CLICKED(IDC_FIVE, OnBnClickedFive)
	ON_BN_CLICKED(IDC_SIX, OnBnClickedSix)
	ON_BN_CLICKED(IDC_SEVEN, OnBnClickedSeven)
	ON_BN_CLICKED(IDC_EIGHT, OnBnClickedEight)
	ON_BN_CLICKED(IDC_NINE, OnBnClickedNine)
	ON_BN_CLICKED(IDC_STAR, OnBnClickedStar)
	ON_BN_CLICKED(IDC_ZERO, OnBnClickedZero)
	ON_BN_CLICKED(IDC_SHARP, OnBnClickedSharp)
	ON_BN_CLICKED(IDC_VOIP, OnBnClickedVoip)
END_MESSAGE_MAP()

BOOL CWinSockDlg::OnInitDialog()
{
	CDialog::OnInitDialog();
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);
```

```
	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		CString strAboutMenu;
		strAboutMenu.LoadString(IDS_ABOUTBOX);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
strAboutMenu);
		}
	}
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon
	m_connectSock.SetParent(this);
	return TRUE;  // return TRUE  unless you set the focus to a control
}
void CWinSockDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialog::OnSysCommand(nID, lParam);
	}
}
```

```
void CWinSockDlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this);
		SendMessage(WM_ICONERASEBKGND,
reinterpret_cast<WPARAM>(dc.GetSafeHdc()), 0);
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;
		dc.DrawIcon(x, y, m_hIcon);
	}
	else
	{
		CDialog::OnPaint();
	}
}
HCURSOR CWinSockDlg::OnQueryDragIcon()
{
	return static_cast<HCURSOR>(m_hIcon);
}

void CWinSockDlg::OnBnClickedCancel()
{
	Closed();
	OnCancel();
}
```

```
void CWinSockDlg::OnBnClickedButtonSend()
{
	int	BytesSent = m_strSendMessages.GetLength();
	if (m_strSendMessages != "")
	{
		int dwBytes;
		dwBytes = m_connectSock.Send((LPCTSTR)m_strSendMessages,
			BytesSent);
				if (dwBytes == SOCKET_ERROR)
		{
			if (m_connectSock.GetLastError() != WSAEWOULDBLOCK)
			{

				char strError[256];
				wsprintf(strError, "Socket failed to send, Error Code: %d",
					m_connectSock.GetLastError());
				Closed();
				AfxMessageBox (strError);
			}
		}
		else
		{
			m_listMessage.ResetContent();
			if (!( m_strSendMessages == "Connecting...." || m_strSendMessages ==
"Calling...." ||  m_strSendMessages == "Hanging...." ))
			 { CString strTemp = "Send Number: ";
			m_intCountSentByte += BytesSent;
			m_strSendTotalMessages.Format("%lu", m_intCountSentByte);
			strTemp +=    m_strSendMessages;
			m_listMessage.AddString(strTemp);
```

```
                    UpdateData(FALSE);  }
                    else
                    {m_intCountSentByte += BytesSent;
                    m_strSendTotalMessages.Format("%lu", m_intCountSentByte);
                    CString strTemp ;
                    strTemp += m_strSendMessages;
                    m_listMessage.AddString(strTemp);
                    UpdateData(FALSE); }
                }
                m_strSendMessages       = _T("");
                UpdateData(FALSE);
        }

}

void CWinSockDlg::Closed()
{
        m_connectSock.Close();
        m_strloAddIP    = _T("");
        m_strloPort                 = _T("");
        m_strSendTotalMessages          = _T("");
        GetDlgItem(IDC_BUTTON_SEND)->EnableWindow(FALSE);
        GetDlgItem(IDC_CONNECT)->EnableWindow(TRUE);
        UpdateData(FALSE);
}

void CWinSockDlg::OnOK()
{
        OnBnClickedButtonSend();
```

```
}

void CWinSockDlg::Connected()
{
	GetDlgItem(IDC_BUTTON_SEND)->EnableWindow(TRUE);
	GetDlgItem(IDC_CONNECT)->EnableWindow(FALSE);
	UINT temploport = 0;
	m_connectSock.GetSockName(m_strloAddIP, temploport);
	m_strloPort.Format("%lu", temploport);
	UpdateData(FALSE);
}



void CWinSockDlg::OnBnClickedConnect()
{
	m_intCountSentByte = 0;
	UpdateData(TRUE);
	GetDlgItem(IDC_CONNECT)->EnableWindow(FALSE);
	m_connectSock.Create();
	m_connectSock.Connect(m_strAddIP, m_intPorts);
	m_strSendTotalMessages.Format("%lu", m_intCountSentByte);
	m_listMessage.ResetContent();
	m_strSendMessages = "Connecting...." ;
	OnBnClickedButtonSend();
}

void CWinSockDlg::OnBnClickedDisconnect()
{
	m_strSendMessages = "Hanging...." ;
```

```
            OnBnClickedButtonSend();
            m_listMessage.ResetContent();
            UpdateData(FALSE);
            Closed();
}

void CWinSockDlg::OnBnClickedCall()
{
             m_strSendMessages = "Calling...." ;
            OnBnClickedButtonSend();
}



void CWinSockDlg::OnBnClickedHang()
{
             m_strSendMessages = "Hanging...." ;
            OnBnClickedButtonSend();
}

void CWinSockDlg::OnBnClickedOne()
{       m_strSendMessages = "1" ;
                        OnBnClickedButtonSend();
}

void CWinSockDlg::OnBnClickedTwo()
{
              m_strSendMessages = "2" ;
              OnBnClickedButtonSend();
}
```

```
void CWinSockDlg::OnBnClickedThree()
{
	m_strSendMessages = "3" ;
	OnBnClickedButtonSend();
}

void CWinSockDlg::OnBnClickedFour()
{
	m_strSendMessages = "4" ;
	OnBnClickedButtonSend();
}

void CWinSockDlg::OnBnClickedFive()
{
	m_strSendMessages = "5" ;
	OnBnClickedButtonSend();
}

void CWinSockDlg::OnBnClickedSix()
{
	m_strSendMessages = "6" ;
	OnBnClickedButtonSend();
}

void CWinSockDlg::OnBnClickedSeven()
{
	m_strSendMessages = "7" ;
	OnBnClickedButtonSend();
}
```

```
void CWinSockDlg::OnBnClickedEight()
{
	m_strSendMessages = "8" ;
	OnBnClickedButtonSend();
}

void CWinSockDlg::OnBnClickedNine()
{
	m_strSendMessages = "9" ;
	OnBnClickedButtonSend();
}

void CWinSockDlg::OnBnClickedStar()
{
	m_strSendMessages = "*" ;
	OnBnClickedButtonSend();
}

void CWinSockDlg::OnBnClickedZero()
{
	m_strSendMessages = "0" ;
	OnBnClickedButtonSend();
}

void CWinSockDlg::OnBnClickedSharp()
{
	m_strSendMessages = "#" ;
	OnBnClickedButtonSend();
}
```

```
void CWinSockDlg::OnBnClickedVoip()
{
	UpdateData(TRUE);
	CString m_VOIP = "VOIP.EXE ";
		m_VOIP += m_strAddIP ;
		WinExec(m_VOIP,SW_SHOW);
}
```

WinSockDlg.h : header file

```
#pragma once
#include "afxwin.h"
#include "asock.h"
class CWinSockDlg : public CDialog
{
public:
	CWinSockDlg(CWnd* pParent = NULL);// standard constructor
	enum { IDD = IDD_WINSOCK_DIALOG };
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV support
protected:
	HICON m_hIcon;
	virtual BOOL OnInitDialog();
		afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
	afx_msg void OnPaint();
afx_msg HCURSOR OnQueryDragIcon();
	DECLARE_MESSAGE_MAP()
```

```
public:
        afx_msg void OnBnClickedCancel();
        afx_msg void OnBnClickedButtonSend();
        CListBox m_listMessage;
        UINT m_intPorts;
        CString m_strloAddIP;
        CString m_strloPort;
       CString m_strSendTotalMessages;
        CString m_strSendMessages;
        void Closed();

protected:
        CASock m_connectSock;
        int m_intCountSentByte;
        virtual void OnOK();

public:
        void Connected();
        afx_msg void OnBnClickedConnect();
        afx_msg void OnBnClickedDisconnect();
        CString m_strAddIP;
        afx_msg void OnBnClickedButtonSend2();
        afx_msg void OnBnClickedCall();
        afx_msg void OnBnClickedHang();
        afx_msg void OnBnClickedOne();
        afx_msg void OnBnClickedTwo();
        afx_msg void OnBnClickedThree();
        afx_msg void OnBnClickedFour();
        afx_msg void OnBnClickedFive();
        afx_msg void OnBnClickedSix();
```

```
        afx_msg void OnBnClickedSeven();
        afx_msg void OnBnClickedEight();
        afx_msg void OnBnClickedNine();
        afx_msg void OnBnClickedStar();
        afx_msg void OnBnClickedZero();
        afx_msg void OnBnClickedSharp();
        afx_msg void OnBnClickedVoip();

};
```

S e r v e r P r o g r a m s

ASock.cpp : implementation file

```
#include "stdafx.h"
#include "WinSock.h"
#include "ASock.h"
#include "WinSockDlg.h" //
#include ".\asock.h"

HANDLE hComp ;
DCB dcb ;
char  *chCommPort = "COM1";
int valReturn ;
char chBuff;
unsigned long nWritePort;
int a = 1;

CASock::CASock()
{
}

CASock::~CASock()
{
}

void CASock::OnAccept(int nErrorCode)
{
        if(a<=1)
   COM();
        a=2;
```

```
        if ( nErrorCode == 0 )
        ((CWinSockDlg*)m_pDwnd)->AcceptClient();
        CAsyncSocket::OnAccept(nErrorCode);
}

void CASock::OnClose(int nErrorCode)
{
        if ( nErrorCode == 0 )
        {
                CString str;
                        str="/";
                        WriteFile(hComp,str,8,&nWritePort, NULL);
                ((CWinSockDlg*)m_pDwnd)->Closed();
        CAsyncSocket::OnClose(nErrorCode);
        }
}

void CASock::OnReceive(int nErrorCode)
{
        CString strRecvMsg;
        char buffer[4096];
        int nRecv;
        nRecv = Receive(buffer, 4096);
        switch (nRecv)
        {
                case 0:
                        {
                        Close();}
```

```
                break;
        case SOCKET_ERROR:
                if (GetLastError() != WSAEWOULDBLOCK)
                {
                        AfxMessageBox ("Error occurred!!!");
                        Close();
                }
                break;
        default
                buffer[nRecv] = NULL;
                CString strTemp(buffer);
                ((CWinSockDlg*)m_pDwnd)->m_listMessage.ResetContent();
                if ( strTemp == "Connecting...." || strTemp == "Calling...." ||  strTemp
== "Hanging...." )
                {       strRecvMsg =+ strTemp;
                ((CWinSockDlg*)m_pDwnd)->m_listMessage.AddString(strRecvMsg);
                }
                else
                {       strRecvMsg = "Receive Number: " + strTemp;
                ((CWinSockDlg*)m_pDwnd)->m_listMessage.AddString(strRecvMsg);
                }
                        if(strTemp=="Calling....")
                        strTemp=".";
                else if(strTemp=="Hanging....")
                        strTemp="/";
                else
                   { }
                WriteFile(hComp,strTemp,8,&nWritePort, NULL);
                if(strTemp=="Connecting....")
```

```
                VOIP();
        }

        CAsyncSocket::OnReceive(nErrorCode);
}
void CASock::SetParent(CDialog* pDwnd)
{
        m_pDwnd = pDwnd;
}
int CASock::COM(void)
{
 hComp = CreateFile( chCommPort,
                GENERIC_READ | GENERIC_WRITE,
                0,
                NULL,
                OPEN_EXISTING,
                0,
                NULL
                );
  dcb.BaudRate = CBR_9600;
  dcb.ByteSize = 8;
  dcb.Parity = NOPARITY;
  dcb.StopBits = ONESTOPBIT;
  valReturn = SetCommState(hComp, &dcb);
        return 0;
}
void CASock::VOIP(void)
{
        WinExec("VOIP.EXE --listen --auto-answer",SW_SHOW);
}
```

Asock.h : implementation file

```
#pragma once
class CASock : public CAsyncSocket
{
public:
	CASock();
	virtual ~CASock();
	virtual void OnAccept(int nErrorCode);
	virtual void OnClose(int nErrorCode);
	virtual void OnReceive(int nErrorCode);
	void SetParent(CDialog* pDwnd);
protected:
	CDialog* m_pDwnd;
public:
	int COM(void);
	void VOIP(void);
};
```

WinSockDlg.cpp : implementation file

```
#include "stdafx.h"
#include "WinSock.h"
#include "WinSockDlg.h"
#include ".\winsockdlg.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#endif
class CAboutDlg : public CDialog
{
public:
        CAboutDlg();
        enum { IDD = IDD_ABOUTBOX };
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
protected:
        DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
}
```

```
BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
END_MESSAGE_MAP()
CWinSockDlg::CWinSockDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CWinSockDlg::IDD, pParent)
	, m_intPorts(3000)
	, m_strloAddIP(_T(""))
	, m_strloPort(_T(""))
	, m_strReAddIP(_T(""))
	, m_strRePort(_T(""))
	, m_strSendTotalMessages(_T(""))
	, m_strSendMessages(_T(""))
	, m_intCountSentByte(0)
	, m_bConnectFlag(false)
{
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CWinSockDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	DDX_Control(pDX, IDC_LISTMESSAGES, m_listMessage);
	DDX_Text(pDX, IDC_PORTS, m_intPorts);
	DDX_Text(pDX, IDC_LOADDIP, m_strloAddIP);
	DDX_Text(pDX, IDC_LOPORT, m_strloPort);
	DDX_Text(pDX, IDC_READDIP, m_strReAddIP);
	DDX_Text(pDX, IDC_READDIP, m_strReAddIP);
	DDX_Text(pDX, IDC_REPORT, m_strRePort);
	DDX_Text(pDX, IDC_SENTTOTALBYTES, m_strSendTotalMessages);
}
```

```
BEGIN_MESSAGE_MAP(CWinSockDlg, CDialog)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_BN_CLICKED(IDCANCEL, OnBnClickedCancel)
	ON_BN_CLICKED(IDC_LISTEN, OnBnClickedListen)
	ON_BN_CLICKED(IDC_CLOSELISTEN, OnBnClickedCloselisten)
	ON_BN_CLICKED(IDC_VOIP, OnBnClickedVoip)
END_MESSAGE_MAP()
BOOL CWinSockDlg::OnInitDialog()
{
	CDialog::OnInitDialog();
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);
	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		CString strAboutMenu;
		strAboutMenu.LoadString(IDS_ABOUTBOX);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
strAboutMenu);
		}
	}
	SetIcon(m_hIcon, TRUE);
	SetIcon(m_hIcon, FALSE);
```

```
	m_listenSock.SetParent(this);
	m_connectSock.SetParent(this);
	m_bConnectFlag = FALSE;
	return TRUE;
}

void CWinSockDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialog::OnSysCommand(nID, lParam);
	}
}

void CWinSockDlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this);
		SendMessage(WM_ICONERASEBKGND,
reinterpret_cast<WPARAM>(dc.GetSafeHdc()), 0);
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
```

```
                CRect rect;
                GetClientRect(&rect);
                int x = (rect.Width() - cxIcon + 1) / 2;
                int y = (rect.Height() - cyIcon + 1) / 2;
                dc.DrawIcon(x, y, m_hIcon);
        }
        else
        {
                CDialog::OnPaint();
        }
}
HCURSOR CWinSockDlg::OnQueryDragIcon()
{
        return static_cast<HCURSOR>(m_hIcon);
}

void CWinSockDlg::OnOK()
{
        //CDialog::OnOK();
}

void CWinSockDlg::OnBnClickedCancel()
{
        OnBnClickedCloselisten();
        OnCancel();
}
```

```
void CWinSockDlg::OnBnClickedListen()
{
		m_intCountSentByte = 0;
	UpdateData(TRUE);
	GetDlgItem(IDC_LISTEN)->EnableWindow(FALSE);
	if (!m_listenSock.Create(m_intPorts))
	{
		char strError[256];
		wsprintf(strError, "Socket failed to create, Error Code: %d",
			m_listenSock.GetLastError());
		m_listenSock.Close();
		AfxMessageBox (strError);
		GetDlgItem(IDC_LISTEN)->EnableWindow(TRUE);
	}
	m_listenSock.Listen();
	m_strSendTotalMessages.Format("%lu", m_intCountSentByte);
	m_listMessage.ResetContent();
	UpdateData(FALSE);
}

void CWinSockDlg::OnBnClickedCloselisten()
{
	Closed();
	m_listenSock.ShutDown();
	m_listenSock.Close();
	m_strloAddIP = _T("");
	m_strloPort = _T("");
	m_strReAddIP   = _T("");
```

```
	m_strRePort		= _T("");
	m_strSendMessages	= _T("");
	m_strSendTotalMessages		= _T("");
	GetDlgItem(IDC_LISTEN)->EnableWindow(TRUE);
	UpdateData(FALSE);
}

void CWinSockDlg::AcceptClient()
{
	if (m_bConnectFlag)
	{
		CAsyncSocket RejectSock; // New socket
		CString strInfo;
		strInfo = "มีการเชื่อมต่อมากเกินไป โปรดรอสักครู่ และลองพยายามอีกครั้ง";
		m_listenSock.Accept(RejectSock);
		RejectSock.Send(LPCTSTR(strInfo), strInfo.GetLength());
		RejectSock.Close();
	}
	else
	{
		UINT tempport = 0;
		UINT tempReport = 0;
		m_bConnectFlag = TRUE;
		m_listenSock.Accept(m_connectSock);
		m_connectSock.GetSockName(m_strloAddIP, tempport);
		m_strloPort.Format("%lu", tempport);
		m_connectSock.GetPeerName(m_strReAddIP, tempReport);
		m_strRePort.Format("%lu", tempReport);
		UpdateData(FALSE);
```

```
		}
}

void CWinSockDlg::Closed()
{
		m_bConnectFlag = FALSE;
		m_connectSock.Close();
}

void CWinSockDlg::OnBnClickedVoip()
{
		WinExec("VOIP.EXE --listen --auto-answer",SW_SHOW);
}
```

WinSockDlg.h : header file

```
#pragma once
#include "afxwin.h"
#include "asock.h"
class CWinSockDlg : public CDialog
{
public:
        CWinSockDlg(CWnd* pParent = NULL);// standard constructor
        enum { IDD = IDD_WINSOCK_DIALOG };
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);       // DDX/DDV support
protected:
        HICON m_hIcon;
        virtual BOOL OnInitDialog();
        virtual void OnOK();
        afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
        afx_msg void OnPaint();
        afx_msg HCURSOR OnQueryDragIcon();
        DECLARE_MESSAGE_MAP()
public:
        afx_msg void OnBnClickedCancel();
        afx_msg void OnBnClickedListen();
        afx_msg void OnBnClickedCloselisten();
        CListBox m_listMessage;
        UINT m_intPorts;
        CString m_strloAddIP;
        CString m_strloPort;
        CString m_strReAddIP;
        CString m_strRePort;
        CString m_strSendTotalMessages;
```

```
        CString m_strSendMessages;
        void Closed();
        void AcceptClient();
public:
protected:
        CASock m_connectSock;
        CASock m_listenSock;
        int m_intCountSentByte;
        bool m_bConnectFlag;
        afx_msg void OnEnChangeListmessages();
        afx_msg void OnLbnSelchangeListmessages();
public:
        afx_msg void OnBnClickedVoip();
};
```

Telephone Control by Microcomputer mcs-51

```
        ORG     0000H
        MOV     P0,#00H
        MOV     P2,#00H
        MOV     P0,#00H
        MOV     P2,#00H
        MOV     P0,#00H
        MOV     P2,#00H
        CALL    INIT_SERIAL
        CLR     RI
        CLR     P3.6

OPEN:   MOV     P0,#00H
        MOV     P2,#00H
        JNB     RI,$
        MOV     A,SBUF
        CLR     RI
        CJNE    A,#2EH,OPEN
        SETB    P2.2

LOOP_CH:MOV             P0,#00H
        MOV     P2,#00000100B
        SETB    P2.2
        JNB     RI,$
        MOV     A,SBUF
        CLR     RI
        CLR     P3.7
        CALL    DELAY
        SETB    P3.7
```

```
CLOSE:      CJNE    A,#2FH,CH0
            CLR     P2.2
            JMP     OPEN

CH0:    CJNE    A,#'1',CH1
            SETB    P0.2
            CALL    DELAY
            JMP     LOOP_CH

CH1:    CJNE    A,#'2',CH2
            SETB    P0.1
            CALL    DELAY
            JMP     LOOP_CH

CH2:    CJNE    A,#'3',CH3
            SETB    P0.0
            CALL    DELAY
            JMP     LOOP_CH

CH3:    CJNE    A,#'4',CH4
            SETB    P0.3
            CALL    DELAY
            JMP     LOOP_CH

CH4:    CJNE    A,#'5',CH5
            SETB    P0.5
            CALL    DELAY
            JMP     LOOP_CH

CH5:    CJNE    A,#'6',CH6
```

```
        SETB   P0.4
        CALL   DELAY
        JMP    LOOP_CH

CH6:    CJNE   A,#'7',CH7
        SETB   P2.6
        CALL   DELAY
        JMP    LOOP_CH

CH7:    CJNE   A,#'8',CH8
        SETB   P2.7
        CALL   DELAY
        JMP    LOOP_CH

CH8:    CJNE   A,#'9',CH9
        SETB   P0.6
        CALL   DELAY
        JMP    LOOP_CH

CH9:    CJNE   A,#'*',CH10
        SETB   P2.3
        CALL   DELAY
        JMP    LOOP_CH

CH10:   CJNE   A,#'0',CH11
        SETB   P2.4
        CALL   DELAY
        JMP    LOOP_CH

CH11:   CJNE   A,#23H,CH12
```

```
        SETB   P2.5
        CALL   DELAY
CH12:   JMP    LOOP_CH


INIT_SERIAL:MOV    TMOD,#00100000B
        MOV    SCON,#01010000B
        MOV    TH1,#0FBH
        SETB   TR1
        RET


DELAY:          CALL DELAY2
          CALL DELAY2
          RET
DELAY2:         MOV R6,#200          ;100MS
DLY100MS:  CALL    DELAY500U
          DJNZ R6,DLY100MS
          RET
DELAY500U: MOV     R5,#230
          DJNZ R5,$
          RET
END
```