การสื่อสารข้อมูลเสียงผ่านเครือข่ายอินเตอร์เน็ตโดยใช้โครงข่ายโทรศัพท์

Internet Phone Gateway

โดย

นายคชาพฤทธิ์ บุรัมยากร

นายณัฐพงษ์ พัฒนพงษ์

นายปาลธรรม เกษมทรัพย์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2541

การสื่อสารข้อมูลเสียงผ่านเครือข่ายอินเตอร์เน็ตโดยใช้โครงข่ายโทรศัพท์

Internet Phone Gateway

โดย

นายคชาพฤทธิ์ บุรัมยากร 38014052

นายณัฐพงษ์ พัฒนพงษ์ 38014143

นายปาลธรรม เกษมทรัพย์ 38014291

อาจารย์ที่ปรึกษา

อ.นภัทร สระเอี่ยม

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2541

ปริญญานิพนธ์ปีการศึกษา 2541

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การสื่อสารข้อมูลเสียงผ่านเครือข่ายอินเตอร์เน็ตโดยใช้โครงข่ายโทรศัพท์ **Internet Phone Gateway**

ผู้จัดทำ

1. นายคชาพฤทธิ์ บุรัมยากร 38014052

2. นายณัฐพงษ์ พัฒนพงษ์ 38014143

3. นายปาลธรรม เกษมทรัพย์ 38014291

อาจารย์ที่ปรึกษา

(อ.นภัทร สระเอี่ยม)

การเชื่อมโทรศัพท์กับอินเตอร์เน็ต

Internet Phone Gateway

โดย นายคชาพฤทธิ์ บุรัมยากร 38014052
นายณัฐพงษ์ พัฒนพงษ์ 38014143
นายปาล ธรรม เกษมทรัพย์ 38014291

อาจารย์ที่ปรึกษา อ.นภัทร สระเอี่ยม

บทคัดย่อ

เนื่องจากในปัจจุบันการสื่อสารด้วยเสียงผ่านทางอินเตอร์เน็ตนั้นต้องทำบนระบบเครือข่ายอินเตอร์เน็ตเท่านั้น โครงงานชิ้นนี้ได้มีความพยายามที่จะลดข้อจำกัดนี้โดยการเชื่อมต่อข้อมูลทางเสียงเข้ากับโครงข่ายโทรศัพท์สาธารณะโดยมีการสร้างโปรแกรมเพื่อให้บริการและส่วนเชื่อมต่อกับระบบโทรศัพท์ ทั้งนี้ยังมีการลดขนาดข้อมูลโดยวิธีของอเด็ฟทีฟเดลต้ามอดูเลชันเพื่อลดอัตราการส่งข้อมูล

ABSTRACT

Nowsday internetphone is a voice communication between two computers on internet, so this project try to solve this deficiency by connecting voice data from internet into public service telephone network by making client-server program and implementing the server-to-telephone interface circuit.The voice data tranfered between client and server is compressed by Adaptive Delta Modulation method in order to increase data tranferring speed.

สารบัญ

หน้า

สารบัญรูปภาพ

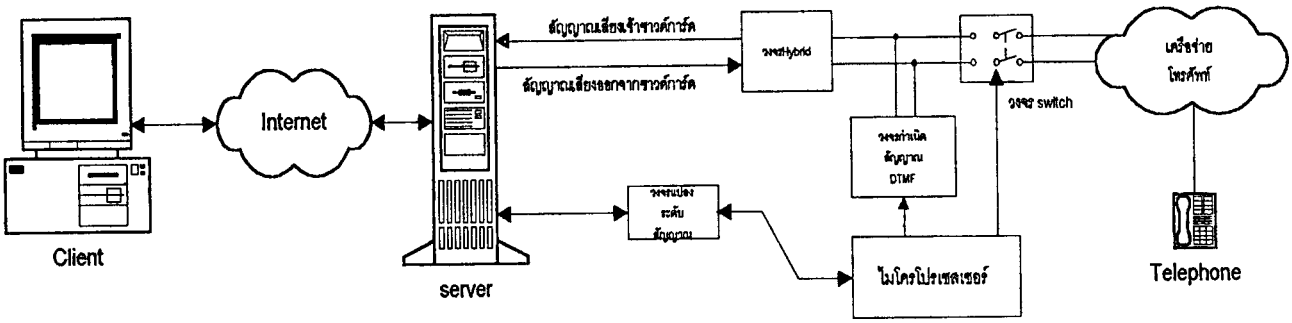
หน้า

บทที่1
บทนำ

ในปัจจุบันนี้เทคโนโลยีทางการสื่อสารได้พัฒนาทั้งรูปแบบและวิธีการไปอย่างรวดเร็วและมีความหลากหลายมากขึ้นซึ่งอินเตอร์เน็ตเองนั้นก็เป็นรูปแบบหนึ่งของการสื่อสารข้อมูลที่ได้รับความนิยมอย่างสูงในขณะนี้ ดังนั้นโครงงานนี้จึงได้ทำการสร้างแอปพลิเคชัน (Application) เพื่อใช้ในเครือข่ายอินเตอร์เน็ตขึ้นมา

อินเตอร์เน็ตโฟนนั้นเป็นแอปพลิเคชันอีกอันหนึ่งที่ใช้ในการส่งข้อมูลเสียงผ่านระบบเครือข่ายของอินเตอร์เน็ต ข้อดีของอินเตอร์เน็ตโฟนนั้นคือค่าใช้จ่ายที่ไม่แพงเมื่อเปรียบเทียบกับการโทรศัพท์ทางไกลไปต่างประเทศ แต่อินเตอร์เน็ตโฟนนั้นมีข้อจำกัดอยู่ที่เมื่อผู้ใช้ต้องการใช้ ผู้ใช้ทั้งสองฝ่ายจะต้องอยู่ต่อหน้าเครื่องคอมพิวเตอร์ที่ใช้งานซึ่งจะต้องอยู่บนเครือข่ายอินเตอร์เน็ตทั้งสองเครื่อง

โครงงานนี้จึงได้นำเสนอแอปพลิเคชันที่ทำหน้าที่เป็นส่วนเชื่อมต่อกันระหว่างอินเตอร์เน็ตโฟนกับโทรศัพท์ธรรมดา ทำให้สามารถที่จะเรียกเข้าไปในโทรศัพท์ธรรมดาได้จากอินเตอร์เน็ตโฟนทำให้ผู้ใช้บริการทั้งสองฝ่ายไม่จำเป็นต้องอยู่ต่อหน้าเครื่องคอมพิวเตอร์พร้อมกันซึ่งเป็นการทำให้ข้อจำกัดของการใช้อินเตอร์เน็ตโฟนหมดไป



รูปที่ 1.1 แสดงขอบข่ายของโครงงาน

บทที่ 2
ทฤษฎีหรือหลักการ

2.1 อินเตอร์เน็ต (Internet)

2.1.1 ระบบเครือข่ายคอมพิวเตอร์ และอินเทอร์เน็ตเวิร์คกิ้ง(Internetworking)

ระบบเครือข่ายคอมพิวเตอร์ (Computer Network) คือระบบการเชื่อมต่อระหว่างระบบปลายทาง (End-System) ซึ่งระบบปลายทางเป็นระบบอิสระต่อกัน (Autonomous) ระบบปลายทางสามารถเป็นได้ตั้งแต่ไมโครคอมพิวเตอร์ไปจนกระทั่งถึงซุปเปอร์คอมพิวเตอร์ (Supercomputer) ขนาดใหญ่เพื่อจุดมุ่งหมายในการแลกเปลี่ยนข้อมูลและการแบ่งปันทรัพยากรของระบบ เช่นไฟล์ (File),เครื่องพิมพ์ (Printer),โมเด็ม (Modem),ตลอดจนการให้บริการฐานข้อมูลร่วม (Sharing database)

อินเตอร์เน็ตเวิร์คกิ้ง หรืออินเตอร์เน็ต (internet) คือการเชื่อมต่อของระบบเครือข่าย 2 เครือข่ายขึ้นไป ดังนั้นคอมพิวเตอร์บนระบบเครือข่ายหนึ่งก็สามารถติดต่อกับคอมพิวเตอร์บนระบบเครือข่ายอื่นๆได้

องค์ประกอบของอินเตอร์เน็ต

อินเตอร์เน็ตเป็นเครือข่ายคอมพิวเตอร์ชนิดหนึ่งที่ใช้โปรโตคอล TCP/IP (Transmission Control Protocal/Internet Protocal) เป็นมาตรฐานการทำงานของระบบ ดังนั้นถ้ามีเครือข่ายคอมพิวเตอร์ที่ใช้โปรโตคอล TCP/IP อยู่แล้วก็จะเป็นการสะดวกและง่ายต่อการเชื่อมต่อเข้ากับระบบอินเตอร์เน็ต ระบบการทำงานของเครือข่ายโปรโตคอล TCP/IP โดยเฉพาะสำหรับเครือข่ายอินเทอร์เน็ตนั้นจะแบ่งกลุ่มของแฟกเตอร์หรือฟังก์ชั่นการทำงานออกเป็น 6 กลุ่มใหญ่ๆ ซึ่งการติดตั้งอุปกรณ์ต่างๆเองก็ต้องคำนึงถึงแฟกเตอร์ 6 กลุ่มด้วยกันคือ

ชนิดของสถานี ระบบเครือข่าย TCP/IP ส่วนใหญ่จะประกอบด้วยเครื่องคอมพิวเตอร์ที่ทำหน้าที่แตกต่างกันอยู่สองชนิด คือ เครื่องที่ทำหน้าที่ให้บริการที่เรียกว่าโฮสต์ (Host) หรือเซิร์ฟเวอร์ (Server) และคอมพิวเตอร์สำหรับผู้ใช้ทั่วไปเรียกว่าเทอร์มินัล (Terminal) หรือ ไคลเอ็นต์ (Client) โดยที่เครื่องคอมพิวเตอร์ที่ทำหน้าที่เป็นสถานีให้บริการนั้นจะเป็นเครื่องที่คอยให้บริการแก่ผู้ใช้ในด้านต่างๆ ไม่ว่าจะเป็นแหล่งเก็บรวบรวมข้อมูล (Data Sharing) การให้บริการโปรแกรมประยุกต์ต่างๆ (Application) หรือการให้บริการการใช้งานระบบประมวลผลกลาง (CPU Time Sharing) เป็นต้น ดังนั้นคุณสมบัติโดยทั่วไปทั้งด้านฮาร์ดแวร์และด้านซอฟต์แวร์ของเครื่องโฮสต์หรือเซอร์ฟเวอร์จึงมีคุณสมบัติที่ดีกว่าเครื่องคอมพิวเตอร์สำหรับผู้ใช้มาก

ระบบไอพีแอดเดรส (IP Address)

การสื่อสารข้อมูลในระบบเครือข่ายคอมพิวเตอร์ เป็นการสื่อสารในลักษณะที่เฟรมข้อมูลของแต่ละการสื่อสารเป็นคนกำหนดเส้นทางที่สื่อสารเอง คือเมื่อมีการขอติดต่อสื่อสารข้อมูลของเครื่องคอมพิวเตอร์คู่ใดเกิดขึ้น เครื่องคอมพิวเตอร์เครื่องนั้นก็จะทำการสร้างเฟรมข้อมูลขึ้นมาแล้วค่อยส่งออกไปในระบบเครือข่ายโดยที่เฟรมข้อมูลนั้นจะมีส่วนของแอดเดรสที่อยู่ในส่วนอ้างอิงทำกับการสื่อสาร (Header) ที่จะบอกว่า เฟรมข้อมูลนี้เป็นของเครื่องคอมพิวเตอร์เครื่องใดที่กำลังส่งและจะส่งไปยังเครื่องใด

ดังนั้นการที่เครื่องคอมพิวเตอร์ต่างๆจะติดต่อสื่อสารกันในระบบเครือข่ายโปรโตคอล TCP/IP จะต้องมีการกำหนดค่าไอพีแอดเดรสให้แต่ละสถานีที่จะสื่อสารกันด้วย นอกเหนือจากค่า MAC Address ที่มีอยู่ในแต่

ละเครื่อง เพราะค่าไอพีแอดเดรสนั้นจะเป็นค่าอ้างอิงในเฟรมข้อมูลที่สื่อสารในเครือข่าย ซึ่งจะมี 2 ชนิดคือ ไอพีแอดเดรสต้นทาง (Source IP Address) และไอพีแอดเดรสปลายทาง (Destination IP Address)

ระบบโปรโตคอลหาเส้นทาง (IP Routing Protocal)

ลักษณะการติดต่อสื่อสารกันระหว่างเครื่องคอมพิวเตอร์ใดๆในระบบเครือข่ายอินเตอร์เน็ต นั้นโดยทั่วไปแล้วมี 2 ลักษณะคือ

1. การเชื่อมต่อภายในเครือข่ายท้องถิ่น (LAN)
2. การเชื่อมต่อระหว่างเครือข่ายท้องถิ่นหนึ่งกับเครือข่ายท้องถิ่นอื่น โดยอาจมีการบริการของระบบเครือข่ายระยะไกล (WAN) เข้ามาเกี่ยวข้องด้วย

ในการเชื่อมต่อจำเป็นต้องใช้อุปกรณ์ที่เรียกว่าเราเตอร์ (Router) โดยเราเตอร์จะเกี่ยวข้องกับระบบทำงานที่เรียกว่า โปรโตคอลหาเส้นทาง (Routing Protocal) ซึ่งจะทำหน้าที่ตรวจสอบและจัดการเกี่ยวกับเส้นทางในการสื่อสารข้อมูลทั้งหมดของระบบ

ระบบชื่อกลุ่ม (Domain Name System)

มีการออกแบบระบบชื่อของสถานีบริการต่างๆบนเครือข่ายอินเทอร์เน็ตในรูปลักษณะตัวอักษร เพื่ออำนวยความสะดวกต่อการใช้งานของยูสเซอร์ ระบบ DNS เป็นระบบซอฟแวร์ที่ทำหน้าที่ในการจัดสรรและบริการในส่วนการเปรียบเทียบค่าระหว่างชื่อตัวอักษรกับค่าไอพีแอดเดรสของเครื่องสถานีต่างๆบนอินเตอร์เน็ต

โปรแกรมประยุกต์บนเครือข่ายอินเตอร์เน็ต (Application)

ระบบเครือข่ายอินเทอร์เน็ตเป็นระบบเครือข่ายขนาดใหญ่ที่มีการเชื่อมโยงกันทั่วโลก ดังนั้นการใช้งานโปรแกรมประยุกต์ต่างๆบนระบบอินเทอร์เน็ตจึงมีลักษณะพิเศษแตกต่างจากการใช้งานบนระบบเครือข่ายท้องถิ่นทั่วไป คือจะมีโปรแกรมประยุกต์มากมายหลายชนิด เช่น ระบบจดหมายอิเลคทรอนิคส์ (E-mail) ระบบข่าวสารรวม (Usenet) ระบบอุโมงค์อินเทอร์เน็ต (Gopher) และระบบเครือข่ายใยแมงมุม (World-Wide-Web) เป็นต้น โดยที่แต่ละชนิดมีการใช้งานที่แตกต่างกันมากระบบความปลอดภัย (Security) มีหน้าที่ป้องกันไม่ให้มีการลักลอบเข้ามาใช้หรือทำลาย ข้อมูลที่สำคัญ

2.1.2 โปรโตคอล TCP/IP

<u>ข้อแตกต่างระหว่างโปรโตคอล TCP/IP และรูปแบบของ OSI</u>

- ลำดับการติดต่อสื่อสารของชั้นเลเยอร์ ในรูปแบบ OSI นั้นจะกำหนดลำดับชั้นการสื่อสารที่เป็นลำดับขั้นตอนการติดต่อที่แน่นอน โดยเฉพาะการอินเตอร์เฟสระหว่างเลเยอร์ ซึ่งทำให้ระบบ OSI สามารถเป็นระบบเปิดสำหรับระบบเครือข่ายคอมพิวเตอร์ทั่วไป เพราะไม่ว่าจะมีการเปลี่ยนแปลงโปรโตคอลในเลเยอร์ชั้นใดก็ตามจะไม่มีผลกระทบต่อการสื่อสารเลเยอร์ชั้นถัดไป ในขณะที่ชุด โปรโตคอล TCP/IP จะไม่มีการกำหนดรูปแบบการติดต่อที่ตายตัว เพื่อให้ผู้ออกแบบสามารถออกแบบโครงสร้างของเครือข่ายได้อย่างอิสระ

- การติดต่อสื่อสารระหว่างเครือข่ายหรือการอินเทอร์เน็ต คือ การติดต่อสื่อสารข้อมูลระหว่างระบบคอมพิวเตอร์ 2 ระบบที่ไม่สามารถติดต่อสื่อสารกันได้โดยผ่านทางเครือข่ายการสื่อสารข้อมูลเพียงเครือข่ายเดียวได้ ต้องอาศัยเครือข่ายตั้งแต่ 2 เครือข่ายขึ้นไปในการติดต่อสื่อสารกัน และเครือข่ายเหล่านี้อาจจะมีลักษณะของเครือข่ายที่ต่างกันก็ได้

ความแตกต่างในเรื่องของอินเทอร์เน็ตระหว่างชุดโปรโตคอล TCP/IP กับรูปแบบ OSI ก็คือในชุดโปรโตคอล TCP/IP จะใช้โปรโตคอลสำหรับอินเทอร์เน็ตที่เรียกว่า โปรโตคอล IP (Internet Protocal) ซึ่งในรูปแบบ OSI จะเรียกว่าโปรโตคอลสำหรับการอินเตอร์เน็ตว่า โปรโตคอลเน็ตเวิร์ค

- การบริการการเชื่อมต่อข่าวสาร(connection service) ในชุดโปรโตคอล TCP/IP นั้นจะมีการบริการการเชื่อมต่อการสื่อสารระหว่างต้นทางและปลายทาง 2 แบบ คือการบริการแบบ คอนเนคชันเลส (Connectionless)และแบบ คอนเนคชันโอเรียนท์ (Connection-Oriented) ส่วนในรูปแบบ OSI จะให้ความสำคัญเฉพาะกับการบริการแบบ คอนเนคชันโอเรียนท์ เท่านั้น

- โปรโตคอลควบคุมการจัดการสื่อสาร ในชุดโปรโตคอล TCP/IP จะใช้โปรโตคอล TCP (Transmission Control Protocal) เป็นโปรโตคอลสำหรับควบคุมการสื่อสาร กำหนดตำแหน่งต้นทางและปลายทาง และอื่นๆกับข้อมูล ซึ่งในรูปแบบ OSI นั้นจะแบ่งแยกการควบคุมการสื่อสารออกจากกันโดยใช้โปรโตคอลเซสชั่นและโปรโตคอล ทรานสปอร์ตตามลำดับ

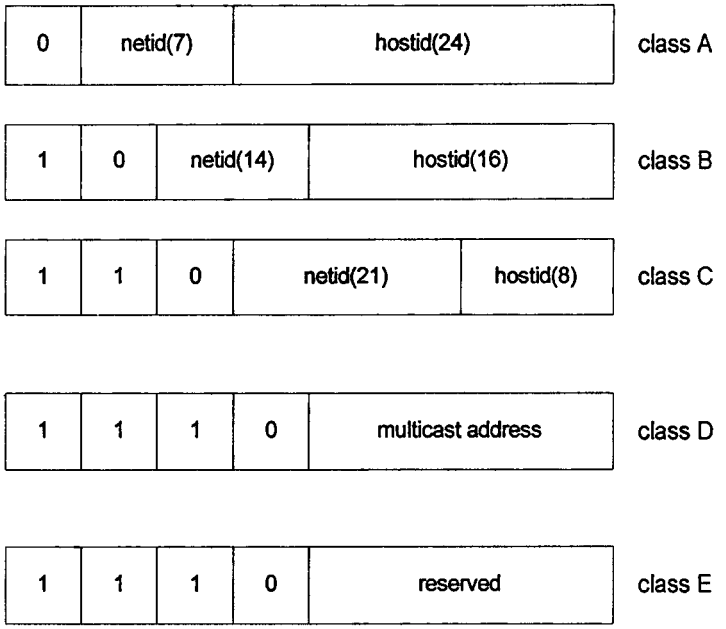
ลักษณะการติดต่อ แบ่งออกเป็น 2 ชนิดคือ

- คอนเนคชันโอเรียนท์ (Connection-Oriented)คือการติดต่อที่ต้องมีการเชื่อมโปรเซสที่จะทำการติดต่อก่อนที่จะมีการส่งหรือรับข้อมูล ซึ่งสามารถใช้คำว่าวงจรเสมือน (Virtual Circuit) เพราะว่าจะทำงานเสมือนมีวงจรต่ออยู่ระหว่างโปรเซส ถึงแม้ว่าข้อมูลนี้อาจจะผ่าน Packet-Switching Network บริการชนิดนี้ส่วนมากจะใช้ในกรณีที่มีข่าวสารต้องการมากกว่าหนึ่งข่าวสาร ดังนั้นสามารถแบ่งขั้นการทำงานออกเป็น

- ขั้นการสร้างการติดต่อ (Connection Establishment)
- ขั้น การส่งผ่านข้อมูล (Data Transfer)
- ขั้นยกเลิกการติดต่อ (Connection Termination)

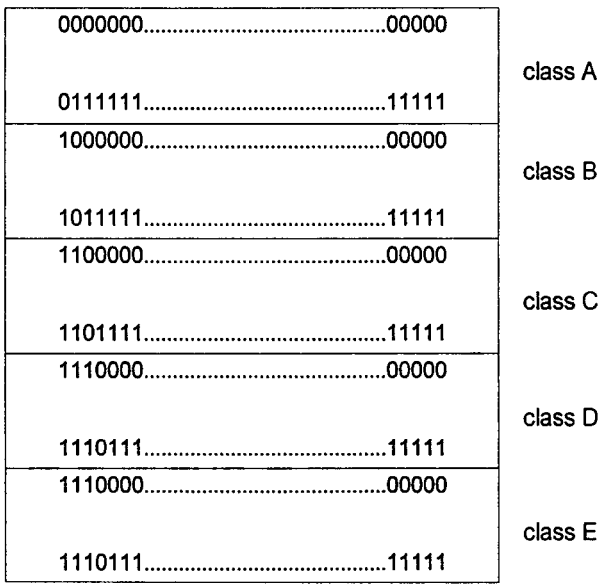
- คอนเนคชันเลส (connectionless) หรือ ดาต้าแกรม(Datagram) คือจะไม่มีการสร้างการติดต่อ และขั้นการยกเลิกการติดต่อ แต่จะมีขั้นการส่งผ่านข้อมูลเพียงอย่างเดียว โดยข้อมูลที่ซึ่งเรียกว่าดาต้าแกรมจะถูกส่งจากระบบหนึ่งไปสู่ระบบหนึ่งอย่างเป็นอิสระโดยไม่ขึ้นอยู่กับดาต้าแกรมอื่น

-ไอพีแอดเดรส (IP Address) เป็นหัวใจสำคัญของโปรโตคอล IP เพราะเป็นแอดเดรสที่บ่งบอกถึงสถานีปลายทางจริงๆ ภายในระบบเครือข่ายขนาดใหญ่ที่มีการเชื่อมต่อทั้ง LAN และ WAN ทำให้ผู้ใช้งานใช้ระบบเครือข่ายเสมือนเป็นระบบเครือข่ายเดียวกันได้ และมีวัตถุประสงค์ให้ผู้ใช้สามารถกำหนดแอดเดรสของสถานีได้ตามต้องการ



รูปที่ 2.1 แสดงประเภทของไอพีแอดเดรส

ไอพีแอดเดรสมีขนาด 32 บิท แบ่งออกเป็น 5 ประเภท คือ A,B,C,D,E มีใช้งานโดยทั่วไปอยู่ 3 ประเภทแรก ส่วนแบบ D ใช้ในกรณีพิเศษ ส่วน E ถูกสำรองไว้ในอนาคต
ช่วงของไอพีแต่ละประเภทแสดงได้ดังรูป



รูปที่ 2.2 แสดงช่วงของไอพีแต่ละประเภท

ไอพีแต่ละประเภทมีหมายเลขที่ไม่ซ้ำกัน ซึ่งก็หมายความว่า ถ้ากำหนดสถานีใดๆ ด้วยไอพีแอดเดรส หมายเลขของสถานีก็ไม่ซ้ำกันด้วย เมื่อดูแค่สามประเภทแรก จะพบว่าภายในไอพีแอดเดรส

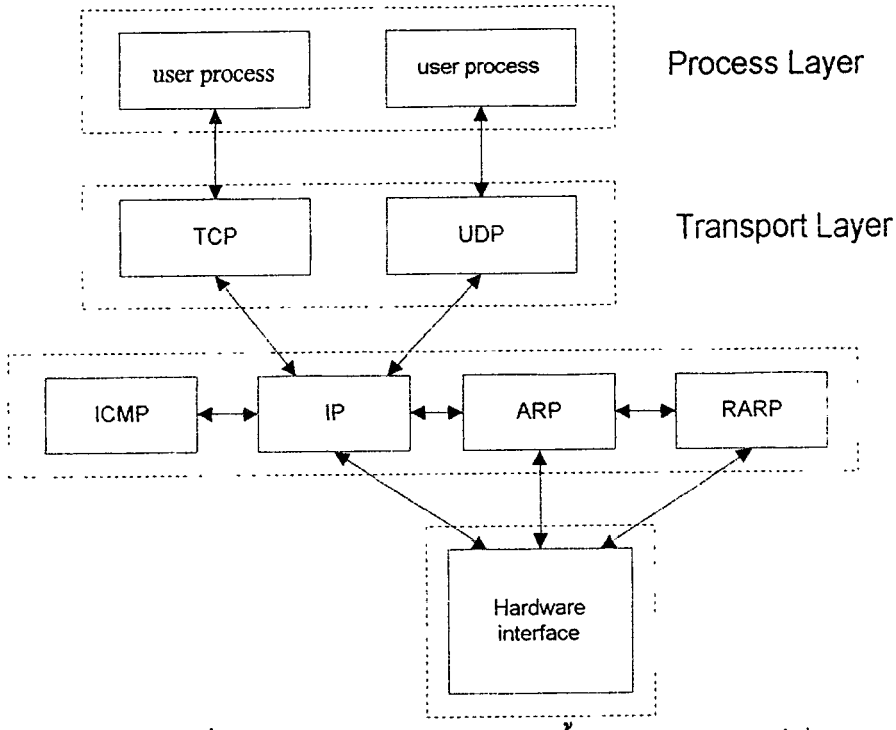
ขนาด 32 บิท แบ่งออกเป็น 2 ส่วนย่อย คือ หมายเลขเครือข่าย (network identification) และหมายเลขสถานี (host identification)

ในการอ่านไอพีแอดเดรสจะไม่มองเป็นเลขฐานสอง แต่จะมองเป็นเลขฐานสิบแทน โดยการแบ่ง 32 บิทออกเป็น 4 ไบต์ ย่อย แต่ละไบต์แทนด้วยเลขฐานสิบ 1 ตัว เรียกว่า"dotted decimal" ดังนั้นเลขแต่ละตัวของไอพีแอดเดรสมีค่า 0-255 เช่น $10011110011011000000010011100010_b$ เขียนในรูป dotted decimal ได้เป็น 158.108.4.226 ส่วนค่าหมายเลขเน็ตเวิร์คนั้นถ้าเป็นแบบ A จะต้องขึ้นต้นด้วย 0 แบบ B ต้องขึ้นต้นด้วย 10 และแบบ C ต้องขึ้นต้นด้วย 110

<u>โครงสร้างของชุดโปรโตคอล TCP/IP</u>

โครงสร้างของสถาปัตยกรรมของชุดโปรโตคอล TCP/IP นั้นแบ่งออกเป็น 3 ส่วนหลักๆ คือ ส่วนของกรรมวิธีปฏิบัติหรือโปรเซส (Process) โฮสต์ (Host) และเครือข่าย (Network) ในส่วนของโปรเซสก็ได้แก่แอปพลิเคชั่นหรือเอนทิลิตี้ที่ต้องการติดต่อสื่อสาร ทุกโปรเซสจะกระทำในเครื่องของโฮสต์ ซึ่งในแต่ละโฮสต์จะสามารถมีหลายเอนทิตี้ ได้พร้อมกัน การสื่อสารกันระหว่างเอนทิตี้ของโฮสต์เครื่องหนึ่งกับเอนทิตี้ของโฮสต์อีกเครื่องหนึ่ง หรือหลายเครื่องจะกระทำโดยผ่านทางเครือข่ายที่โฮสต์เชื่อมต่ออยู่

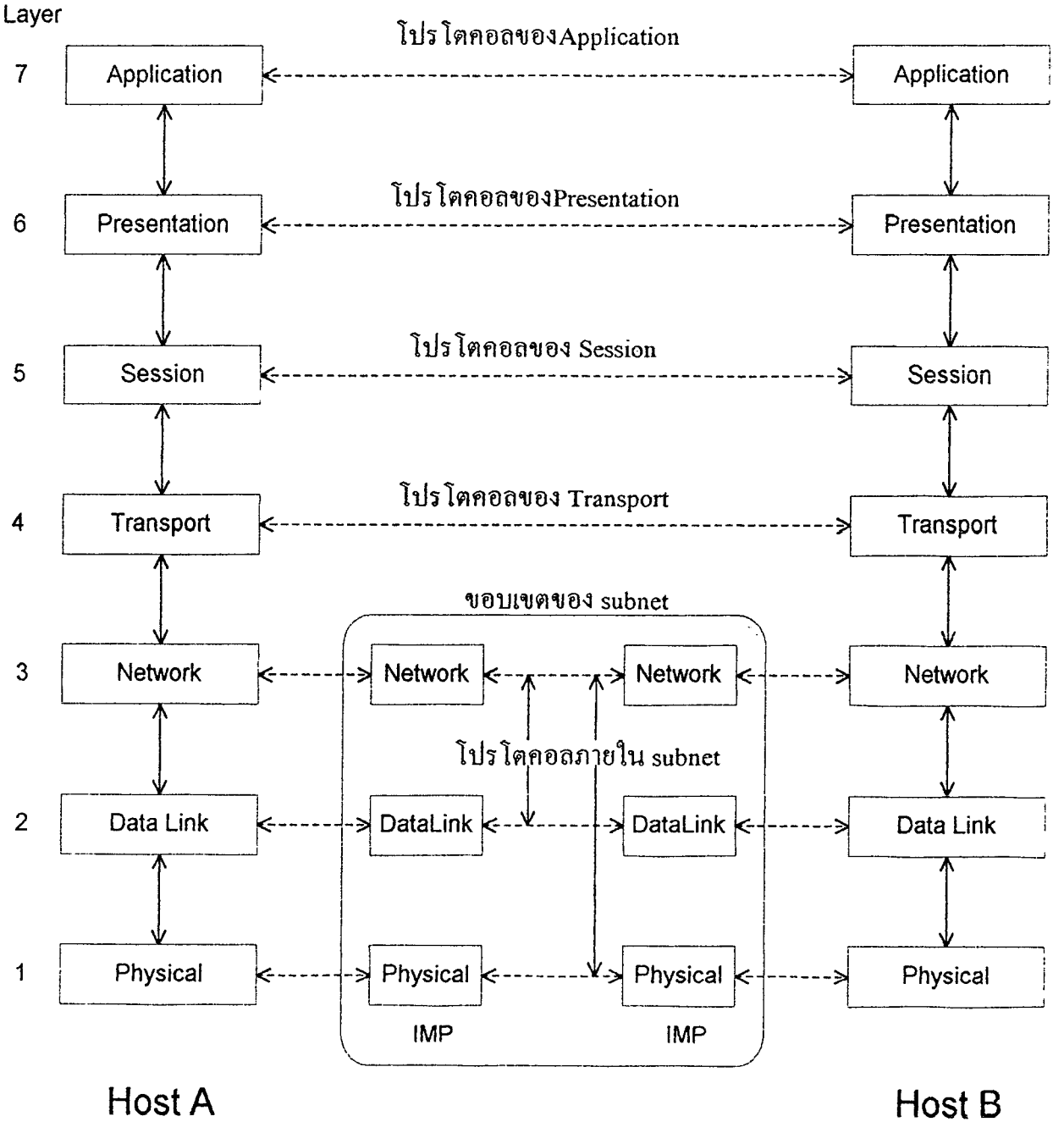
การทำงานที่สัมพันธ์กันระหว่างโปรเซส โฮสต์ และเครือข่ายของสถาปัตยกรรม TCP/IP ทำให้สามารถจัดรูปแบบของสถาปัตยกรรม TCP/IP ได้เป็น 4 ชั้น และสามารถกำหนดชนิดของโปรโตคอล ที่ทำงานในแต่ละชั้นได้เป็น 4 โปรโตคอลเช่นกัน ดังที่ได้กล่าวมาแล้วว่าในชุดโปรโตคอล TCP/IP นั้นเอนทิตี้แต่ละชั้นอาจจะติดต่อสื่อสารข้อมูลโดยผ่านเอนทิตี้ในชั้นเดียวกัน หรือเอนทิตี้ในชั้นล่างลงไปซึ่งไม่จำเป็นต้องเป็นชั้นที่ติดกัน



รูปที่ 2.3 แสดงความสัมพันธ์ระหว่างชั้นต่างๆ ของอินเตอร์เน็ต

2.2 แบบจำลอง OSI (OSI model)

แบบจำลอง OSI (Open System Interconnection model) เป็นรูปแบบของเครือข่ายที่ถูกเสนอขึ้นโดยองค์กร ISO (International Standard Organization) ในปี 1983 ซึ่งมีจุดประสงค์ใหญ่เพื่อให้ระบบคอมพิวเตอร์ต่างๆสามารถเชื่อมต่อเป็นระบบเดียวกันได้ ไม่ว่าเครื่องนี้จะเป็นของบริษัทผู้ผลิตรายใด แต่ว่ารูปแบบในการเชื่อมต่อกันเป็นเครือข่ายนั้นจะปฏิบัติตามแนวทางเหมือนกันตามแบบจำลอง OSI นอกจากนี้แล้วในแต่ละชั้นของแบบจำลอง OSIก็ต้องใช้โปรโตคอลในแบบเดียวกันด้วย ปัจจุบันแบบจำลอง OSIมีรูปแบบเป็นลำดับขั้นโปรโตคอล (protocal hierarchies)โดยแบ่งเป็นชั้นทั้งหมด 7 ชั้นด้วยกัน



รูปที่ 2.4 แสดง OSI โมเดลและการเชื่อมต่อ

ชั้นฟิสิคัล (physical layer)

ชั้นฟิสิคัลนี้เป็นชั้นที่เกี่ยวข้องกับการติดต่อสื่อสารในระดับเบื้องต้นที่เรียกว่าบิทดิบ จุดประสงค์ใหญ่ในการออกแบบหน้าที่ของชั้นคือเมื่อผู้ส่งส่งบิทที่มีค่าเป็น 1 ออกไป ทางด้านผู้รับก็จะต้องรับค่าได้เป็น1 เช่นเดียวกัน จากความต้องการดังกล่าวทำให้เกิดเป็นข้อกำหนดต่างๆ เช่น การกำหนดแรงดันไฟฟ้าที่มีค่าเป็น1 และ 0 ช่วงเวลาในการส่งบิท 0 ที่ติดกันนั้นห่างกันได้น้อยแค่ไหน ความเป็นไปได้ในการส่งสัญญาณใน 2 ทิศทางโดยมีช่องสัญญาณเดียวกันและในช่วงเดียวกัน การเริ่มต้นการเชื่อมต่อและยกเลิกการเชื่อมต่อเมื่อการติดต่อสิ้นสุดลงว่าเป็นอย่างไร ตลอดจนจำนวนของพิน (pin) และหน้าที่ของแต่ละพินว่าเป็นอย่างไร จะเห็นว่าในชั้นนี้ต้องเกี่ยวข้องกับคุณสมบัติด้าน ไฟฟ้า เครื่องกล เป็นส่วนใหญ่

ชั้นดาต้าลิงค์(data link layer)

หน้าที่หลักของชั้นดาต้าลิงค์คือการจัดส่งข้อมูลผ่านไปยังชั้นชั้นฟิสิคัล รวมทั้งจัดการเกี่ยวกับการตรวจจับความผิดพลาดและการแก้ไขความผิดพลาด ถ้าจะสรุปก็คือเมื่อมองจากชั้นที่อยู่เหนือชั้นดาต้าลิงค์ถัดขึ้นไปซึ่งก็คือชั้นเน็ตเวิร์กนั้น ตัวของชั้นเน็ตเวิร์กจะต้องมองเป็นว่าการส่งผ่านข้อมูลจากตัวเองไปยังชั้นเน็ตเวิร์กของคอมพิวเตอร์ตัวอื่นนั้นเป็นการส่งผ่านข้อมูลที่ถูกต้องสมบูรณ์ไม่มีข้อผิดพลาดเลย ซึ่งก็เป็นหน้าที่ของชั้นดาต้าลิงค์ที่ต้องให้บริการแก่ชั้นเน็ตเวิร์ก

เมื่อชั้นดาต้าลิงค์ได้รับข้อมูลที่ส่งผ่านจากชั้นเน็ตเวิร์กแล้ว ก็จะดำเนินการแบ่งข้อมูล จากนั้นจึงทำการส่งไปยังชั้นดาต้าลิงค์ของคอมพิวเตอร์ทางด้านรับ ด้านรับเมื่อได้รับเฟรมข้อมูลแล้วก็จะส่งเฟรมตอบรับ (acknowledgement frame) กลับไปยังเครื่องคอมพิวเตอร์ทางด้านส่ง หากพิจารณาว่าข้อมูลได้ถูกส่งเรียงกันไปโดยไม่สนใจว่าบิทที่ต่อกันนั้นจะประกอบกันเป็นเฟรมหรือไม่ ดังนั้นจะเป็นหน้าที่ของชั้นดาต้าลิงค์ที่จะต้องกำหนดขอบเขตและขนาดของเฟรมข้อมูลเอาเอง ซึ่งชั้นดาต้าลิงค์จะทำโดยการแทรกบิทพิเศษเอาไว้ที่ต้นเฟรมและที่ท้ายเฟรม

ชั้นเน็ตเวิร์ก (network layer)

ชั้นเน็ตเวิร์กนี้มีหน้าที่หลักเกี่ยวกับการควบคุมการทำงานของซับเน็ต ซึ่งต้องกำหนดว่าแพ็กเกจใดจะถูกส่งไปยังเส้นทางใด (route) จากด้านส่งไปยังด้านรับ โดยเส้นทางนี้อาจจะถูกกำหนดออกมาตายตัวเลยระหว่างตัวประมวลผลเชื่อมต่อข่าวสาร เรียกว่า IMP (Interface Message Processors) กับ IMP อีกตัวหนึ่ง หรือการกำหนดแบบไดนามิค(dynamic) ซึ่งทำให้การติดต่อระหว่าง IMP แต่ละคู่นั้นจะใช้เส้นทางในการติดต่อแตกต่างกัน ทั้งนี้เพื่อลดปัญหาด้านทราฟฟิคในเครือข่าย

ชั้นทรานสปอร์ต (transport layer)

หน้าที่หลักของชั้นทรานสปอร์ต นี้คือการทำการรับข้อมูลจากชั้นเซสชัน จากนั้นทำการแยกข้อมูลออกเป็นหน่วยที่เล็กลง จากนั้นจึงทำการส่งหน่วยต่างๆเหล่านี้ลงไปยังชั้นเน็ตเวิร์กและต้องกระทำการอันเป็นที่แน่ใจว่าหน่วยต่างๆจะต้องส่งไปถึงจุดหมายอย่างถูกต้องตามลำดับ ในสภาวะปกตินั้นชั้นทรานสปอร์ตจะทำหน้าที่คอยสร้างเส้นทางในการติดต่อสื่อสารตามที่ชั้นเซสชันร้องขอมาเมื่อชั้นเซสชันต้องการส่งผ่านข้อมูล และทางชั้นทรานสปอร์ตก็ต้องการให้การไหลเวียนของข้อมูลเป็นไปอย่างรวดเร็ว ก็อาจจะสร้างเส้นทางขึ้นมาหลายเส้นทางแล้วแยกข้อมูลออกเป็นหน่วยย่อยๆเพื่อที่จะส่งข้อมูลแต่ละหน่วยแยกทางกันไป

ชั้นทรานสปอร์ตนั้นจะต้องกำหนดชนิดของบริการที่จะบริการให้แก่ชั้นเซสชัน ชนิดของการเชื่อมต่อในชั้นนี้เช่น แบบ ช่องสัญญาณปราศจากความผิดพลาดชนิดจุดต่อจุด (error free point-to-point channel) ซึ่งจะทำการส่งผ่านข้อมูลเรียงตามลำดับตามที่ถูกส่งออกมาจากต้นทาง

ชั้นเซสชัน (session layer)

ชั้นเซสชันนี้จะคอยทำหน้าที่ในการสร้างเซสชันระหว่างผู้ใช้บริการที่อยู่คนละเครื่องคอมพิวเตอร์กันให้เกิดการติดต่อสื่อสารกันได้ ความหมายคือ การที่จะยอมให้มีการลำเลียงข้อมูลเป็นลำดับตามมา เช่นในระบบโทรศัพท์ หน้าที่ในการบริการต่อผู้เรียกโดยส่งสัญญาณไปยังผู้รับ จนผู้รับรับขึ้นมาจนทำให้เกิดการสนทนาเริ่มต้นขึ้นได้ หน้าที่ดังกล่าวจะเป็นหน้าที่ของชั้นเซสชัน แต่หน้าที่ในการส่งสัญญาณแต่ละคำพูดจะเป็นหน้าที่ของชั้นทรานสปอร์ต

บริการอีกอย่างหนึ่งคือ การบริหาร โทเคน (token management) โดยโปรโตคอลบางตัวนั้นจะไม่อนุญาตให้เครื่องคอมพิวเตอร์ทั้งสองด้านที่ทำการติดต่อสื่อสารกันอยู่นั้นทำการปฏิบัติการที่เหมือนกันในเวลาเดียวกันได้ นอกจากนี้ยังมีหน้าที่อีกอย่างหนึ่งคือการซิงโครนัสเครื่องคอมพิวเตอร์สองเครื่องเข้าด้วยกันเพื่อให้การติดต่อเป็นไปอย่างคล้องจองกัน

ชั้นพรีเซนเตชัน (presentation layer)

ชั้นพรีเซนเตชันนี้มีหน้าที่จัดการเกี่ยวกับวากยสัมพันธ์ (syntax) และรูปแบบต่างๆ ของข่าวสารที่จะถูกส่งจากเครื่องคอมพิวเตอร์ออกไป เช่น การเข้ารหัสข้อมูลให้อยู่ในรูปที่เข้าใจกันทั้งผู้รับและผู้ส่ง ตัวอย่างก็คือเมื่อชั้นนี้รับข้อความจากชั้นแอปพลิเคชันซึ่งเป็นข้อความมาแล้ว ก็จะต้องมาทำการเข้ารหัสอักขระ (character) แต่ละตัวในข้อความนั้นให้อยู่ในรูปที่ทางผู้รับรับแล้วสามารถแปลงกลับเป็นข้อความที่ถูกต้องได้เหมือนเดิม เช่นการเข้ารหัสข้อมูลแบบแอสกี เป็นต้น

ชั้นแอปพลิเคชัน (application layer)

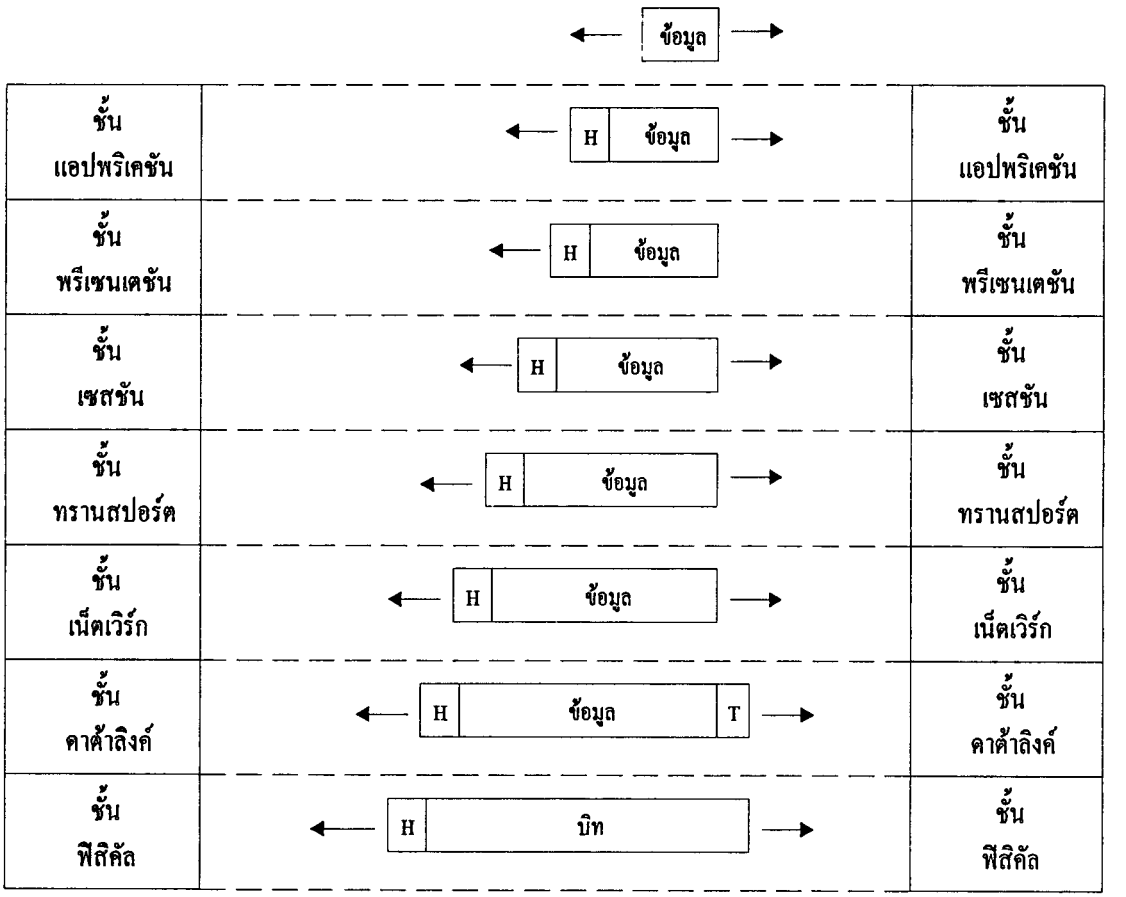
ชั้นแอปพลิเคชันนี้เป็นชั้นที่มีโปรโตคอลหลายหลากมากมาย โดยที่ชั้นนี้เป็นส่วนที่ติดต่อกับผู้ใช้โดยตรง ได้แก่ โฮสคอมพิวเตอร์ เทอร์มินัลหรือคอมพิวเตอร์ส่วนบุคคล เป็นต้น แอปพลิเคชันในชั้นนี้สามารถนำเข้าหรือออกจากระบบเครือข่ายได้โดยไม่ต้องสนใจว่าจะมีขั้นตอนการทำงานอย่างไรเพราะจะมีชั้นพรีเซนเตชันรับผิดชอบอยู่แล้ว

2.2.1 การส่งข้อมูลในแบบจำลอง OSI

ในรูปที่ 2.5 แสดงตัวอย่างให้เห็นถึงการส่งผ่านข้อมูลโดยใช้เครือข่ายตามแบบจำลอง OSI โดยเริ่มจากข้อมูลถูกป้อนจากผู้ใช้บริการเข้าไปยังชั้นแอปพลิเคชัน จากนั้นชั้นแอปพลิเคชันก็อาจจะเพิ่มส่วนที่เป็นข้อมูลสำหรับการควบคุมบางอย่างเพิ่มเติมเข้าไปยังข้อมูลเดิมที่ส่วนหัว จากนั้นจึงส่งผ่านต่อลงมายังชั้นพรีเซนเตชัน เมื่อชั้นพรีเซนเตชันได้รับข้อมูล ก็อาจจะแปลงข้อมูลให้อยู่ในรูปแบบมาตรฐาน เช่นแปลงข้อความที่อยู่ในรูปอักษร (text) ให้อยู่ในรูปแอสกี และอาจจะเพิ่มเติมส่วนหัวเข้าไปด้วย แล้วจึงส่งต่อลงมาให้ชั้นเซสชัน นอกจากนี้ตัวของชั้นพรีเซนเตชันเองจะไม่สนใจด้วยว่าข้อมูลที่มันรับมาจากชั้นแอปพลิเคชันนั้นประกอบด้วยอะไรบ้าง โดยจะมองข้อมูลที่ส่งออกมาเป็นเนื้อเดียวกันหมดไม่สนว่าอันไหนเป็นข้อมูลจริงอันไหนเป็นส่วนหัวที่เติมเข้ามา ในทำนองเดียวกันกับชั้นบนข้อมูลจะถูกส่งผ่านลงมาเรื่อยๆจนถึงชั้นฟิสิคัล ซึ่งเป็นชั้นที่จะกระทำให้เกิด

การส่งข้อมูลที่แท้จริงเพื่อส่งผ่านไปถึงเครื่องคอมพิวเตอร์ตัวรับ แล้วข้อมูลก็จะถูกส่งย้อนขึ้นไปข้างบนกระบวนการก็จะกระทำย้อนกลับกับตอนที่ส่งมาจนถึงชั้นแอปพลิเคชัน

ถึงแม้ว่าการส่งผ่านข้อมูลจริงๆนั้นจะกระทำในแนวตั้งลงมา แต่ก็สามารถมองได้เสมือนว่าแต่ละชั้นที่อยู่ระดับเดียวกันรับส่งข้อมูลกันได้โดยตรงตามแนวนอน แนวคิดนี้เป็นจุดประสงค์ในการจะทำให้การนำไปใช้ที่ชั้นต่างๆง่ายขึ้น โดยเสมือนว่าติดต่อกับชั้นในระดับเดียวกันของเครื่องคอมพิวเตอร์ตัวอื่น โดยไม่ต้องสนใจเลยว่าแท้จริงแล้วข้อมูลจะถูกส่งผ่านไปอย่างไร

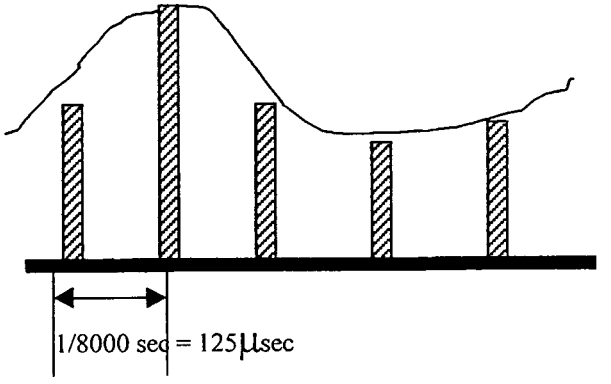


รูปที่ 2.5 แสดงการข้อมูลในเลย์เยอร์ต่างๆ

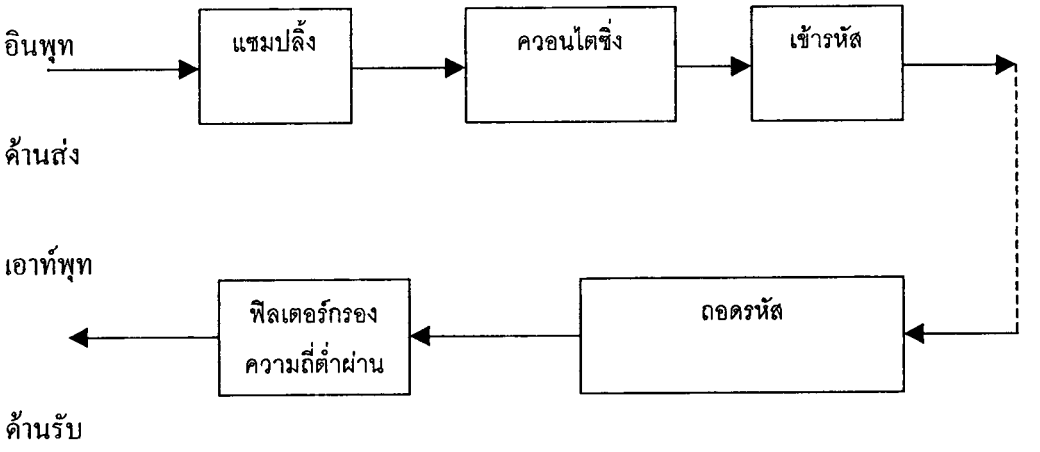
2.3 พัลส์โค้ดมอดดูเลชั่น (Pulse Code Modulation)

ในปัจจุบันระบบพัลส์โค้ดมอดดูเลชั่น ถูกนำไปใช้อย่างกว้างขวางโดยเฉพาะสัญญาณเสียงและในระยะหลังๆนี้ การพัฒนาระบบนี้เพื่อการนำไปใช้กับสัญญาณภาพก็มีบทบาทขึ้นด้วย ดังนั้นในบทนี้จะกล่าวถึงขั้นตอนของกระบวนการ พัลส์โค้ดมอดดูเลชั่น อย่างคร่าวๆ

แซมปลิ้งเรต = 8KHz



รูปที่ 2.6 แสดงการแซมปลิ้งสัญญาณ



รูปที่ 2.7 กระบวนการเข้าและถอดรหัสของพัลส์โค้ดมอดดูเลชั่น

จากรูปที่2.7 เป็นการแสดงขั้นตอนการประมวลเพื่อให้ได้รับพัลส์โค้ดมอดดูเลชั่น อย่างกว้างๆ ก็คือการเข้ารหัส(coding) และการนำสัญญาณไปแปลงกลับซึ่งเรียกว่าการถอดรหัส (decoding) เพื่อให้ได้สัญญาณเดิม

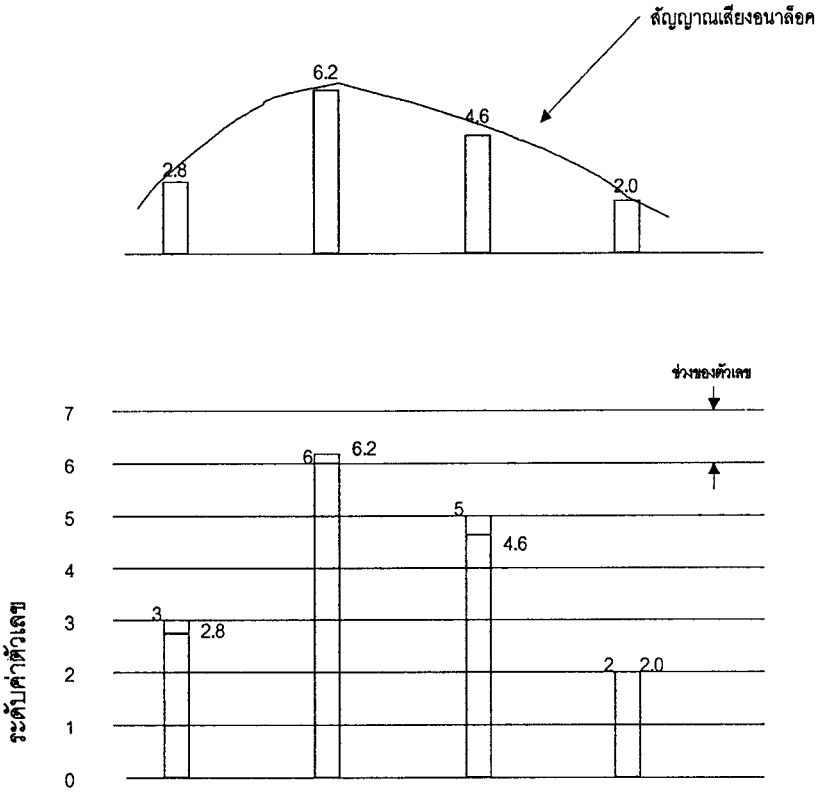
2.3.1 แซมปลิ้ง

แซมปลิ้ง คือการทำสัญญาณซึ่งมีค่าต่อเนื่องให้เป็นดิสครีท(discrete)ในช่วงเวลาที่เท่าๆกัน โดยใช้ทฤษฎีแซมปลิ้งที่ว่า ถ้าเก็บแซมปลิ้งด้วยอัตรา 2 เท่าหรือมากกว่าความถี่สูงสุดของสัญญาณอนาล็อกแล้ว จะสามารถทำให้สัญญาณเดิมกลับคืนมาได้อย่างถูกต้อง สัญญาณที่ผ่านการแซมปลิ้ง แล้วจะได้สัญญาณ พัลส์แอมปลิจูดมอดดูเลชั่น (PAM)

2.3.2 ควอนไตซิ่ง

ขบวนพัลส์แอมปลิจูดมอดดูเลชั่น ที่ผ่านการแซมปลิ้งแล้ว ยังถือว่าเป็นสัญญาณอนาล็อกอยู่ คือมันจะมีขนาดเปลี่ยนอย่างต่อเนื่องไปกับเวลา การจัดระดับคือ กระบวนการเปลี่ยนขนาดของพัลส์แอมปลิจูดมอดดู เลชั่น เหล่านั้นให้เป็นตัวเลขเป็นขั้นๆตามค่าที่กำหนดไว้

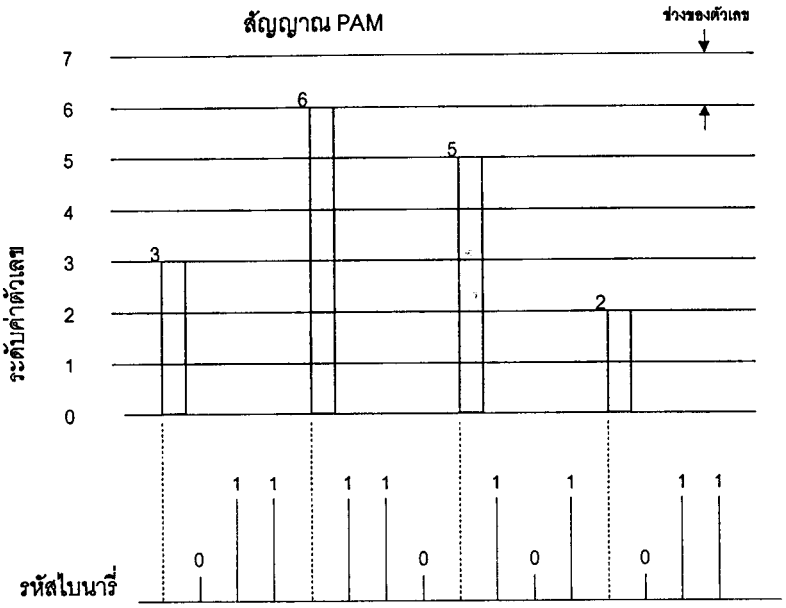
แอมปลิจูดของแซมปลิ้งทุกตัวจะถูกจัดให้เป็นระดับ ซึ่งเรียกว่า ควอนไตซิ่งลีเวล โดยมีระยะห่างระหว่างระดับข้างเคียงเรียกว่า ควอนไตซิ่งอินเทอร์วัลกรณีนี้เรียกว่าการจัดระดับแบบยูนิฟอร์มหรือแบบลิเนียร์



รูปที่ 2.8 แสดงการแปลงสัญญาณพัลส์แอมปลิจูดมอดดูเลชั่น ให้เป็นตัวเลข(การจัดระดับ)

ขนาดของแซมปลิ้งทุกตัวจะแสดงด้วยค่าระดับควอนไตซ์ที่ใกล้เคียงที่สุด จะเห็นว่าสัญญาณพัลส์แอมปลิจูดมอดดูเลชั่น ที่ถูกจัดระดับแล้วนี้จะเพียงค่าโดยประมาณเท่านั้น ดังนั้นส่วนเกินและส่วนขาดจากการจัดระดับจึงเป็นค่าผิดพลาดระหว่างสัญญาณเดิมและค่าที่ได้จัดระดับซึ่งค่าผิดพลาดนี้เรียกว่า ควอนไตซิ่งน้อยส์

ในการให้ทำควอนไตซิ่งน้อยส์นี้ลดลงในเบื้องต้นคือการลดควอนไตซิ่งอินเทอร์วัลให้แคบลง เช่นถ้าลดควอนไตซิ่งอินเทอร์วัลลงครึ่งหนึ่งปริมาณขอควอนไตซิ่งน้อยส์จะลดลง1/4 และการลดควอนไตซิ่งอินเทอร์วัลลงครึ่งหนึ่งจะสอดคล้องกับการเพิ่มจำนวนบิทอีก1บิท นั่นคือควอนไตซิ่งน้อยส์จะลดลง 1dB ทุกๆการเพิ่ม 1 บิท



รูปที่2.9 การนำสัญญาณพัลส์แอมปลิจูดมอดดูเลชั่นมาเข้ารหัส

2.3.3 การเข้ารหัส

หลังจากขบวนพัลส์แอมปลิจูดมอดดูเลชั่น ได้ผ่านการจัดระดับมาแล้ว จะต้องเปลี่ยนขนาดเหล่านั้นเป็นรหัสไบนารี่ กรณีที่เป็นการส่งสัญญาณเสียงทางโทรศัพท์ จะถูกเปลี่ยนเป็นรหัส 8 บิท ซึ่งสามารถแสดงค่าแอมปลิจูดได้ 2^8 ระดับ ระบบการเข้ารหัสมีหลายแบบ แต่ส่วนมากจะใช้กัน 3 แบบดังแสดงไว้ในตาราง

ระดับการควอนไตซ์	รหัสธรรมดา	รหัสแบบเกรย์	รหัสแบบสมมาตร
0	000	000	011
1	001	001	010
2	010	011	001
3	011	010	000
4	100	110	100
5	101	111	101
6	110	101	110
7	111	100	111

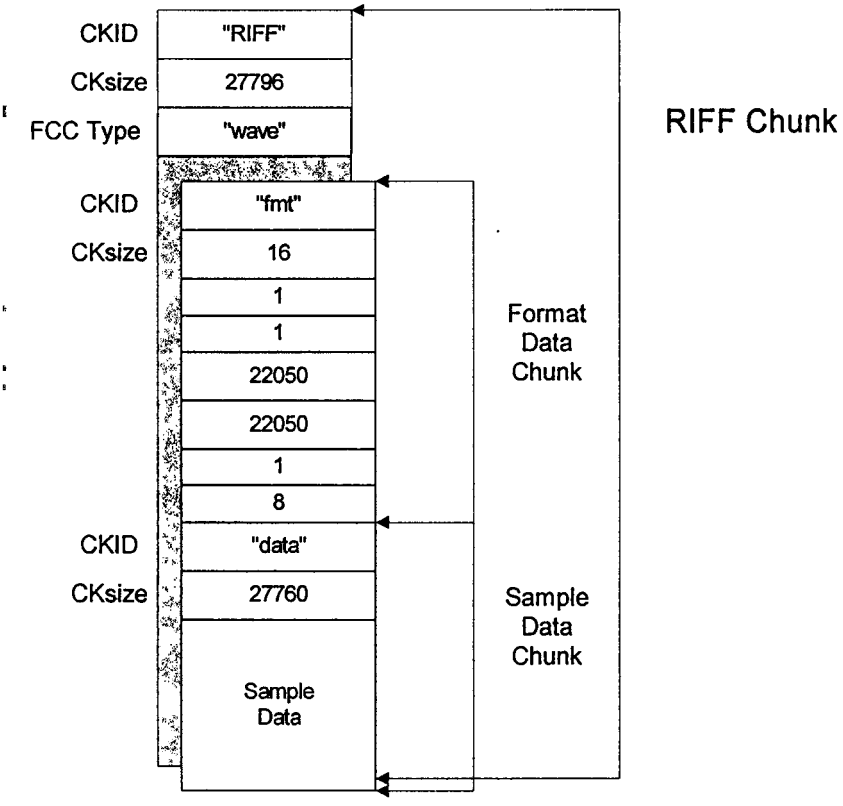
ตารางที่ 2.1 รหัสไบนารีแบบต่างๆ

2.4 ไฟล์สกุล .wav

<u>รูปแบบการบันทึกคลื่นเสียง</u>

สำหรับการบันทึก 1 ช่องสัญญาณเสียงทั่วไปจะบันทึกเป็นโหมดโมโน และถ้าต้องการบันทึกเป็น 2 ช่องสัญญาณเสียงแยกออกเป็น ช่องสัญญาณเสียงข้างซ้ายและขวาจะบันทึกในโหมดสเตอริโอ ซึ่งเหมาะกับการบันทึกเสียงดนตรี

เสียงพูดจะใช้การแซมปลิ้งที่ความถี่ 11025 Hz ส่วนความถี่แซมปลิ้งที่ 22050 Hz จะให้คุณภาพใกล้เคียงกับเทปบันทึกเสียง และความถี่ 44100 Hz จะเป็นความถี่ที่คุณภาพเสียงเดียวกับเพลงที่บันทึกในแผ่น CD ขนาดของแต่ละแซมเปิลจะมีสองระดับ คือ 8 บิต และ 16 บิต



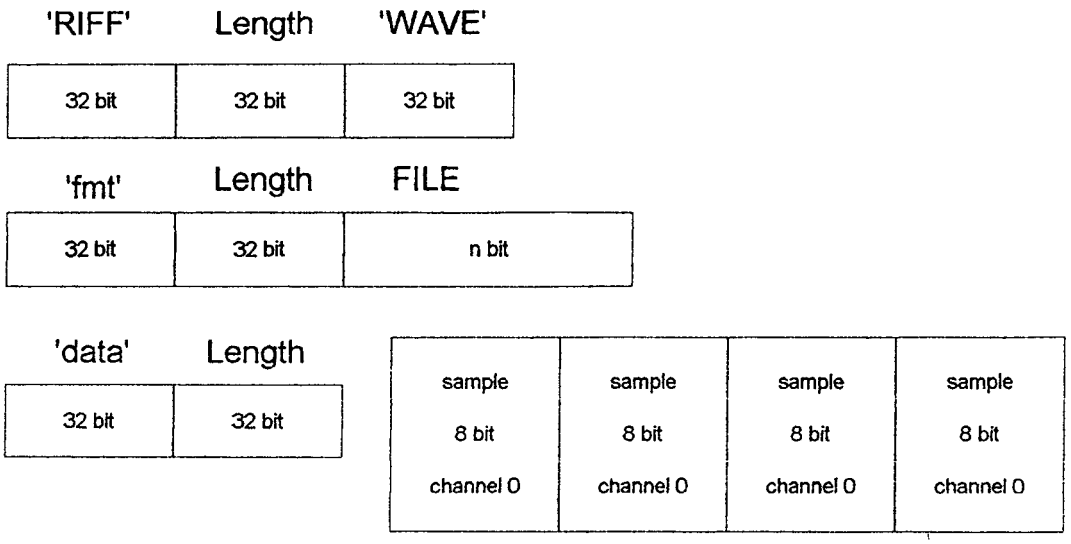
รูปที่ 2.10 โครงสร้างของไฟล์ .wav

รูปแบบของไฟล์ .wav มีอยู่ 4 ประเภทคือ

1. ไฟล์ 8บิตโมโน(8-bit mono)

ตัวอย่างแบบ 8บิตโมโนนี้ ไบท์ (byte) ทั้งหมดถูกเก็บเรียงตามลำดับโดยช่องสัญญาณที่ 0 (channel 0) จะถูกนำมาใช้สำหรับตัวอย่าง

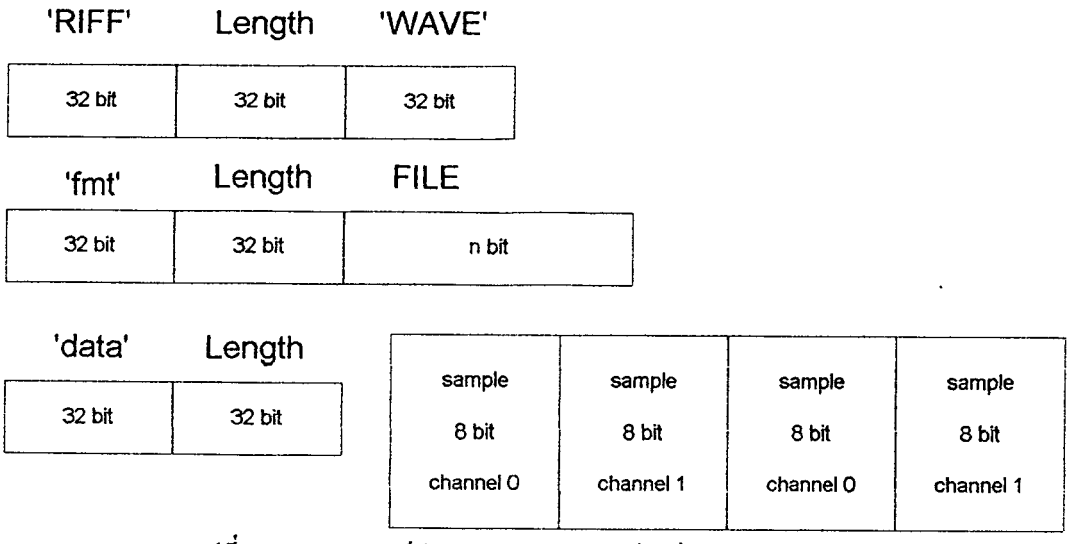
แบบ โมโน



รูปที่ 2.11 แสดงองค์ประกอบของส่วนหัวไฟล์ .wav แบบ 8 บิตโมโน

2.ไฟล์ 8บิตสเตอริโอ (8-bit stereo)

ตัวอย่างแบบสเตอริโอนี้ ช่องสัญญาณ 0 คือ ช่องสัญญาณทางด้านซ้าย และช่องสัญญาณที่ 1 คือ ช่องสัญญาณทางด้านขวา ข้อมูลจะเก็บอยู่ตามรูปแบบนี้ คือ 8 บิต สำหรับช่องสัญญาณที่ 0 , 8 บิตสำหรับช่องสัญญาณ ที่ 1 , 8 บิตสำหรับช่องสัญญาณที่ 0 , 8 บิตสำหรับช่องสัญญาณที่ 1 ..... ซึ่งหมายความว่า 1 แซมเปิล ประกอบด้วย 2 ไบท์ ไบท์แรกสำหรับช่องสัญญาณทางซ้ายและ อีกไบท์สำหรับช่องสัญญาณทางขวา



รูปที่ 2.12 แสดงองค์ประกอบของส่วนหัวไฟล์ .wav แบบ 8 บิต สเตอริโอ

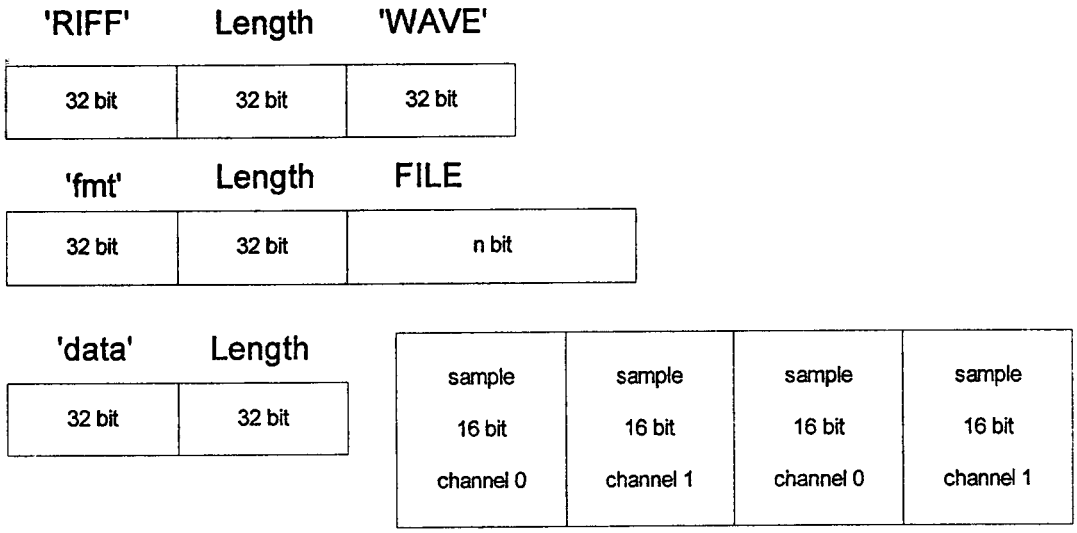
3.ไฟล์16 บิตโมโน(16-bit mono)

การแสดงตัวอย่างแบบ 16 บิตโมโน ในหน่วยความจำนี้ ถูกใช้สำหรับการบันทึกแต่ละแซมเปิลแบบเรียงลำดับของไบท์ข้อมูลเหมือนกับแบบ 16 บิตโมโน

4.ไฟล์ 16 บิต สเตอริโอ (16-bit stereo)

ตัวอย่างแบบ 16 บิต สเตอริโอ ต้องการหน่วยความจำจำนวนมากสำหรับแต่ละแซมเปิล ซึ่งต้องการถึง 4 ไบท์ (2 ไบท์สำหรับช่องสัญญาณทางซ้าย และอีก 2 ไบท์สำหรับช่องสัญญาณทางขวา) ลำดับของการเรียงไบท์ข้อมูล คือ 1 ไบท์ต่ำ (low byte) และ 1 ไบท์สูง (high byte) สำหรับแต่ละช่องสัญญาณ

ในการเก็บแบบแซมเปิลขนาด 8-bit ค่าของแต่ละไบท์ อยู่ระหว่าง 0 ถึง 255 ดังนั้น ค่า 128 แสดงค่ากลางของรูปคลื่นแบบดิจิตอล (median of the digitized waveform)สำหรับตัวอย่างแบบ 16-bit พิสัยของค่าอยู่ระหว่าง -32768 ถึง 32767 ในกรณีนี้ค่า 0 เป็นค่ากลาง



รูปที่ 2.13 แสดงองค์ประกอบของส่วนหัวไฟล์ .wav แบบ 16 บิต สเตอริโอ

2.5 เดลต้ามอดูเลชั่น (Delta Modulation)

2.5.1 หลักการทำงานของเดลต้ามอดูเลชั่น(DM)

สัญญาณที่ได้จากเดลต้ามอดูเลชั่นจะมีลักษณะเป็นสัญญาณขั้นบันได (Stair Case) ซึ่งระดับของขั้นบันไดจะหาได้จากสัญญาณความแตกต่างระหว่างระดับ สัญญาณอินพุท(Input Signal) กับสัญญาณที่ได้จากการประมาณ (Approximate) โดยค่าความแตกต่างที่ได้จะถูกเปลี่ยนเป็นสัญญาณ 2 ระดับคือ +d และ -d เท่านั้น หลักในการหาค่าของสัญญาณประมาณคือ ถ้าสัญญาณอินพุทมีค่าสูงกว่าสัญญาณที่ได้จากการประมาณค่าที่ตำแหน่งของการแซมปลิ้ง (Sampling) ครั้งก่อนก็จะทำการเพิ่มระดับสัญญาณค่าประมาณขึ้นอีก +d หาก สัญญาณค่าประมาณมีค่ามากกว่าสัญญาณอินพุทมันก็จะถูกลดระดับสัญญาณลง -d หากการเปลี่ยนระดับของสัญญาณอินพุทไม่เร็วเกินไป จะพบว่าค่าที่ประมาณไว้เป็นขั้นบันได (Stair Case Approximate) จะมีค่าสูงหรือต่ำกว่าสัญญาณอินพุทไม่เกิน +d หรือ -d

สัญญาณอินพุทไม่เร็วเกินไป จะพบว่าค่าที่ประมาณไว้เป็นขั้นบันได (Stair Case Approximate) จะมีค่าสูงหรือต่ำกว่าสัญญาณอินพุทไม่เกิน +d หรือ -d

หลักการดังกล่าวสามารถแสดงในรูปสมการได้ดังนี้

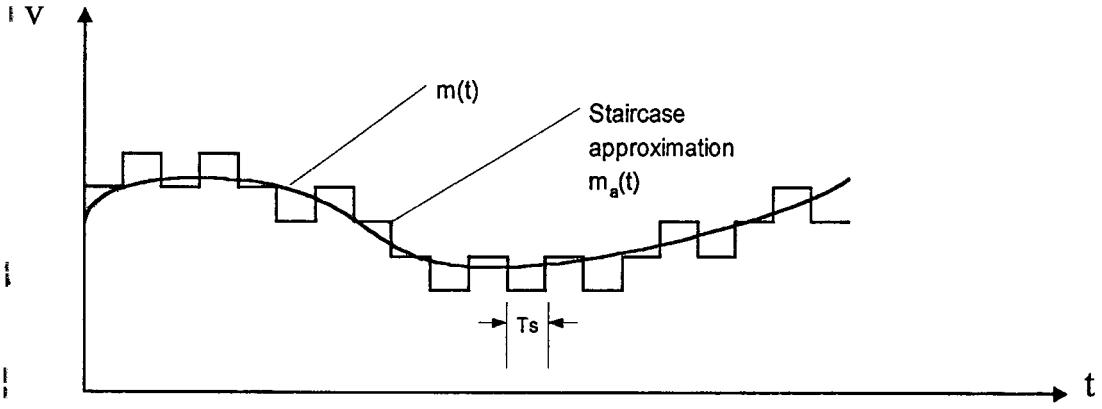
$$b_n = \mathrm{sgn}\left[m(n.Ts) - m_a(n.Ts - Ts)\right] \qquad (2.1)$$
$$m_a(n.Ts) = m(n.Ts - Ts) + d.b_n$$

m(t) คือ สัญญาณอินพุท

$m_a(t)$ คือ ขั้นบันไดการประมาณ

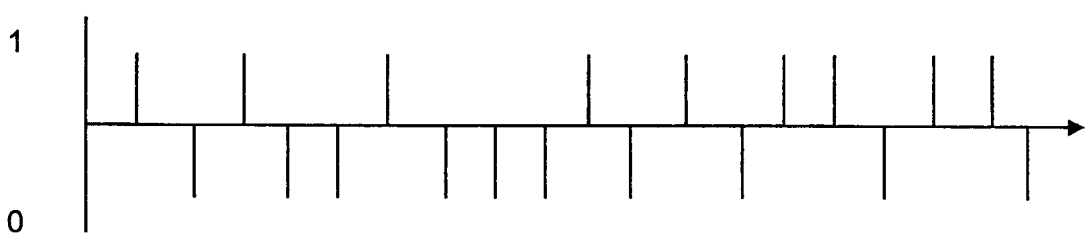
b_n คือ เครื่องหมายที่ได้จากการเปรียบเทียบค่า

d คือ ขนาดของขั้นบันไดที่ใช้



รูปที่ 2.14 แสดงสัญญาณที่ได้จากเดลต้ามอดูเลชั่น

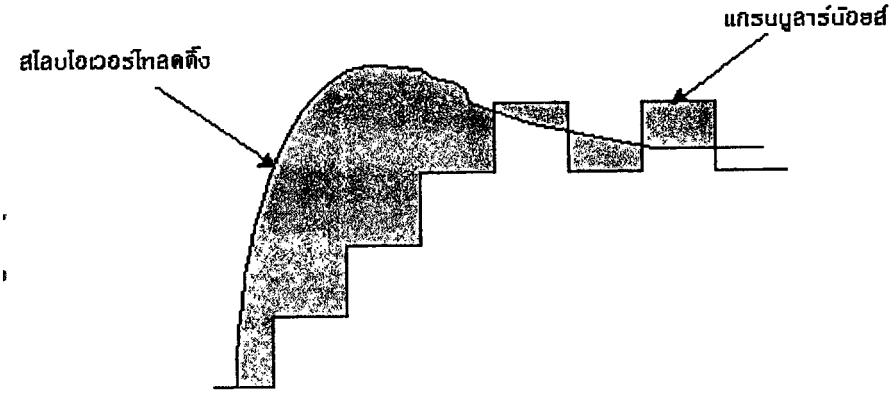
ในการส่งข้อมูลจะส่ง b_n แทนข้อมูล 1 ตัวอย่าง (Sample) ซึ่ง b_n จะถูกแทนด้วยข้อมูลขนาด 1 บิท โดยอัตราการส่งข้อมูลจะเท่ากับอัตราการแซมปลิ้ง (Sampling Rate)



รูปที่ 2.15 แสดงข้อมูลที่ส่งแทน 1 ตัวอย่าง

2.5.2 ความผิดพลาดที่เกิดในระบบ

ความผิดพลาด หรือ ค่าคลาดเคลื่อน ที่เกิดขึ้นในระบบเดลต้ามอดูเลชั่น เรียกว่า ความผิดพลาดควอนไตซิ่ง (Quantizing Error) จะเกิดจากความผิดพลาด 2 อย่าง คือ สโลปโอเวอร์โหลดดิสทอร์ชั่น (Slope Overload Distortion) และ แกรนนูล่าร์น้อยส์ (Granular Noise) ดังแสดงในรูป



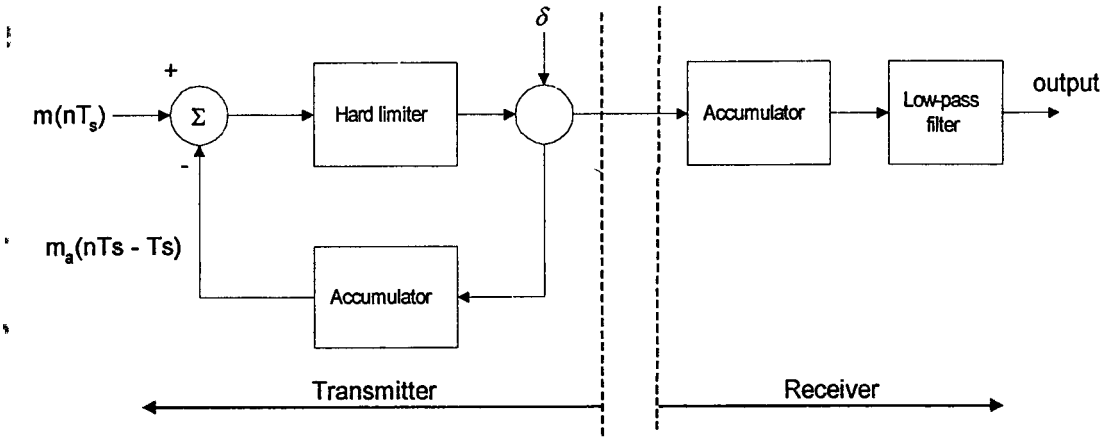
รูปที่ 2.16 แสดงการเกิด ความผิดพลาดควอนไตซิ่ง

สโลปโอเวอร์โหลดดิสทอชั่น เกิดขึ้นจากขนาดของเสต็ป (Step) d มีค่าเล็กเกินกว่าที่จะทำให้สัญญาณขั้นบันได $m_a(t)$ เปลี่ยนค่าตามสัญญาณอินพุท m(t)ได้ทัน

แกนูล่าน้อยส์ จะเกิดเมื่อขนาดของสเต็ป d ใหญ่เกินไป ทำให้สัญญาณขั้นบันไดการประมาณมีค่ามากกว่าสัญญาณอินพุท

สำหรับหลักการของเดลต้ามอดูเลชั่น แบบธรรมดานั้น สามารถแสดงในบล็อกไดอะแกรมดังในรูป ซึ่งประกอบด้วยตัวบวก(summer),ฮาร์ดลิมิเตอร์(hard limiter)และตัวเก็บค่า(accumulator)หรืออินทีเกรเตอร์ (integrator) นั่นเอง โดยตัวบวก จะเปรียบเทียบความแตกต่างระหว่างสัญญาณอินพุท m (n.Ts) และสัญญาณ เอาท์พุท m_a(n.Ts - Ts) จากตัวเก็บค่า ค่าความแตกต่างนี้จะผ่านเข้าไปในฮาร์ดลิมิเตอร์ ซึ่งจะให้เอาท์พุทออกมาเป็น +1 หรือ -1 ขึ้นอยู่กับค่าความแตกต่างแล้วเอาท์พุทจากฮาร์ดลิมิเตอร์จะถูกคูณด้วย d และจะถูกส่งเข้าไปสู่ accumulator ซึ่งทำหน้าที่สร้างสัญญาณค่าประมาณดังสมการ

$$m_a(n.Ts - Ts) = \sum_{i=1}^{n-1} db_i \qquad (2.2)$$



รูปที่ 2.17 แสดงบล็อคไดอะแกรมของระบบเดลต้ามอดูเลชั่น

ดังนั้นทุกชั่วขณะของการแซมปลิ้ง ตัวเก็บค่าจะเพิ่มสัญญาณค่าประมาณด้วยขนาดสเต็ป (d) ตามทิศทางของ b_i ถ้าสัญญาณแซมปลิ้ง m(n.Ts) มากกว่าสัญญาณค่าประมาณ m_a(n.Ts) ค่า +d ก็จะถูกเพิ่มให้กับสัญญาณค่าประมาณ ถ้าสัญญาณตัวอย่าง m(n.Ts) มีค่าน้อยกว่าค่าประมาณ m_a(n.Ts- Ts) ค่า -d จะถูกรวมเข้ากับตัวเก็บค่า โดยวิธีนี้ตัวเก็บค่าจะทำการสร้างสัญญาณใหม่ตามสัญญาณอินพุทโดยใช้ 1 สเต็ปในแต่ละครั้งของการแซมปลิ้งพบว่าพัลส์ที่ได้จะประกอบกันเป็นสัญญาณดิจิตอลเพื่อส่งไปยังด้านรับ ในทางด้านรับก็จะสร้างขั้นบันได m_a(t) ออกมา โดยนำพัลส์บวกและลบที่ได้รับผ่านตัวเก็บค่าซึ่งมีการทำงานเช่นเดียวกับทางด้านส่งส่วน ควอนไตซิ่งน้อยส์ที่มีอยู่ใน m_a(t) จะถูกกำจัดออกไปได้โดยผ่านวงจรกรองความถี่ต่ำผ่าน (low pass filter) ซึ่งมีแบนด์วิด(bandwidth) เท่ากับแบนด์วิดของสัญญาณเบสแบนด์

2.5.3 สโลปโอเวอร์โหลดดิ้ง(Slope Overloading)

ปัญหาใหญ่ในระบบ DM คือการเกิดสโลปโอเวอร์โหลด เมื่อสัญญาณอินพุท m(t) เปลี่ยนแปลงสัญญาณค่าประมาณ m_a(t) จะสเต็ปตามสัญญาณอินพุทไปตลอดตราบเท่าที่สัญญาณตัวอย่างยังมากหรือน้อยกว่าสัญญาณ m_a(t)ซึ่งสเต็ปตามสัญญาณ m(t) ไม่ทันนั้น ก็จะเกิดเป็นความผิดพลาดที่เรียกว่าโอเวอร์โหลด (overload) นี้เกิดขึ้นเนื่องจากความชันของสัญญาณ m(t) เราเรียกว่าสโลปโอเวอร์โหลด (slopeoverload)

เพื่อที่จะหาเงื่อนไขสำหรับป้องกันสโลปโอเวอร์โหลดในระบบ DM เราสมมุติว่าอินพุท

$m(t) = A\cos(2\pi f_m t)$

ดังนั้นความชันสูงสุดของสัญญาณคือ

$$[dm(t)/dt]_{max} = A.2\pi f_m \qquad (2.3)$$

การเปลี่ยนแปลงมากที่สุดในช่วงระหว่างแซมปลิ้งของสัญญาณอินพุทคือ $A.2\pi f_m.Ts$ เพื่อหลีกเลี่ยงการเกิดสโลปโอเวอร์โหลด การเปลี่ยนแปลงของแซมปลิ้งต้องน้อยกว่า d นั่นคือ

$$A.2\pi f_m.Ts < d \qquad (2.4)$$

หรือขนาด (amplitude) ที่ซึ่งสโลปโอเวอร์โหลด จะเกิดขึ้นคือ

$$A=(df_s)/(2\pi f_m) \qquad (2.5)$$

ซึ่ง $f_s=1/T_s$ เป็นอัตราการแซมปลิ้งของระบบเดลต้ามอดูเลชั่นได้มีการตรวจสอบโดยการทดลองพบว่า DM จะส่งสัญญาณเสียงโดยปราศจากการเกิดสโลปโอเวอร์โหลดนั้น ขนาดของสัญญาณจะต้องไม่เกินกว่าขนาดมากที่สุดของสัญญาณในสมการที่ (2.4) ซึ่งใช้กับ f_m 800Hz ปัญหาของการเกิดสโลปโอเวอร์โหลดในระบบเดลต้ามอดูเลชั่น สามารถแก้ไขได้โดยการใช้วงจร กรองสัญญาณ หรือโดยสัญญาณเพื่อจำกัดอัตราสูงสุดของการเปลี่ยนแปลงการเพิ่มขนาดของสเต็ปหรือเพิ่มอัตราการแซมปลิ้งจะทำให้ต้องใช้แบนด์วิดมากขึ้น

หนทางที่ดีที่สุดในการหลีกเลี่ยงสโลปโอเวอร์โหลด โดยการตรวจสอบเงื่อนไขการเกิดสโลปโอเวอร์โหลดและทำให้สเต็ปมีขนาดมาขึ้น เมื่อตรวจสอบได้ว่าเกิดสโลปโอเวอร์โหลดซึ่งเป็นแบบที่เรียกว่า "อะแด็ปทีฟเดลต้ามอดูเลชั่น (ADM)"

2.5.4 สัญญาณรบกวนในระบบเดลต้ามอดูเลชั่น

เอาท์พุทที่ได้ทางด้านดีโคดเดอร์ (decoder) จะมีความแตกต่างจากด้านอินพุท เนื่องจากผลของควอนไตซิ่งน้อยส์ $n_q(t)$ และสัญญาณรบกวนที่เกิดขึ้นในขณะทำการส่ง no (t) ดังนั้น

$$m_r(t) = m_0(t) + n_0(t) + n_q(t) \qquad (2.6)$$

ซึ่ง $m_r(t)$ เป็นเอาท์พุทของดีมอดูเลเตอร์ที่ผ่านวงจรกรองความถี่ต่ำผ่านแล้ว

$m_0(t)$ เป็นสัญญาณเอาท์พุท สมมุติเท่ากับ m(t)

$n_0(t)$ และ $n_q(t)$ เป็นสัญญาณรบกวนที่เอาท์พุทเมื่อผ่านวงจรกรองความถี่แล้ว

คุณภาพของสัญญาณทั้งหมดในระบบ DM ถูกวัดในเทอมของอัตราส่วนสัญญาณต่อสัญญาณรบกวน (signal-to-noise ratio (S/N)) ซึ่งสามารถหาได้จาก

$$(S/N)_0 = \left(E\left[\{m_0(t)\}^2\right]\right) / \left(E\left[\{n_q(t)\}^2\right] + E\left[\{n_0(t)\}^2\right]\right) \qquad (2.7)$$

พลังงานเฉลี่ยที่มีสัญญาณรบกวนอยู่สามารถคำนวณได้ดังต่อไปนี้

2.5.5 ควอนไตซิ่งน้อยซ์ในระบบ เดลต้ามอดูเลชั่น

เราให้ $m(t) = m_a(t) + e_q(t)$ ซึ่ง $|e_q(t)| = |m(t) - m_a(t)| < d$ ในขณะที่เกิดสโลปโอเวอร์โหลด ควอนไตซิ่งน้อยส์ $n_q(t)$ เป็นผลของฟิลเตอร์แก่ $e_q(t)$ ถ้าเราสมมติยูนิฟอร์ม สำหรับ $e_q(t)$ ดังนั้น

$$E\left[\{e_q(t)\}^2\right] = \int \left(\frac{1}{2d}\right).e^2 de = d^2/3 \qquad (2.8)$$

จากการทดลองพบว่า normalize power ของสัญญาณ $e_q(t)$ กระจายอย่างสม่ำเสมอในช่วงความถี่ 0 ถึง f_s เป็นอัตรากการแซมปลิ้ง ดังนั้น power spectral density $G_{eq}(f)$ ของ $e_q(t)$ ถูกกำหนดโดย

$$G_{eg}(f)=\begin{cases} d^2/6f_s & ,f<f_s \\ 0 \end{cases} \qquad (2.9)$$

เพราะว่า n_q (t) เป็นผลตอบสนองของเบสแบนด์ฟิลเตอร์ต่อ $e_q(t)$ พลังงานเฉลี่ยในภาวะปกติของสัญญาณ n_q (t) ถูกกำหนดโดย

$$E\left[\left\{n_q(t)\right\}^2\right]=\int G_{eq}(f)df$$
$$=(d^2/3).(f_m/f_s) \qquad (2..10)$$

ในการที่จะคำนวณอัตราส่วนระหว่างค่าสัญญาณต่อควอนไตซิ่งน้อยส์(signal to quantizing noise power ratio) เราต้องคำนวณหากำลังของสัญญาณ(signal power $E[m_0^2(t)]$)เพื่อความง่ายในการคำนวณ เราจะคิดกรณีที่แย่ที่สุดสำหรับ DM ซึ่งกำลังงานของสัญญาณทั้งหมดมารวมกันที่ขอบบนของเบสแบนด์ ดังนั้นเราให้

$$m(t) = A\cos 2f_m t \qquad (2.11)$$

ดังนั้น

$$m_0(t) = A\cos 2f_m t \qquad (2.12)$$

และ

$$E[m_0^2(t)] = A^2/2 \qquad (2.13)$$

และเพื่อหลีกเลี่ยงการเกิดสโลปโอเวอร์โหลดเรามี

$$A = (d/2\pi).(f_s/f_m) \qquad (2.14)$$

เราจะได้ อัตราส่วนระหว่างค่าสัญญาณต่อควอนไตซิ่งน้อยส์ คือ

$$\{E[m_0^2(t)]\}/\{E[n_q^2(t)]\} = (3/8\pi^2).(f_s/f_m)^3 \qquad (2.15)$$

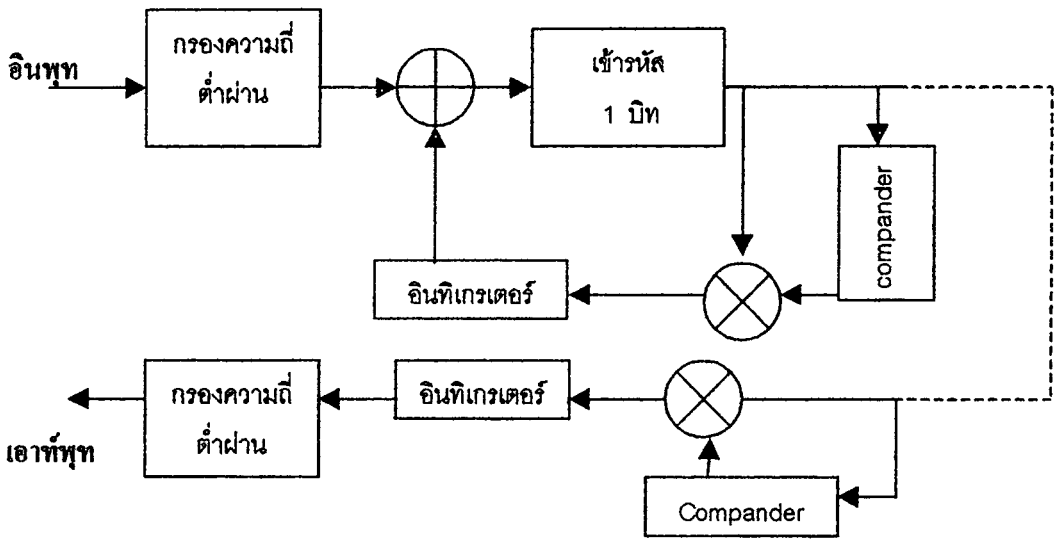
2.6 อะแด็ปทีฟเดลต้ามอดูเลชั่น (Adaptive delta modulation)

ลักษณะโดยทั่วไปของอะแด็ปทีฟเดลต้ามอดูเลชั่น

สโลปโอเวอร์โหลด จะเกิดขึ้นเมื่อความชันของสัญญาณสูงมากและต่ำมาก ปัญหานี้สามารถแก้ไขได้โดยการปรับขนาดของสเต็ป โดยลดขนาดของสเต็ปเมื่อการเปลี่ยนแปลงของสัญญาณเกิดขึ้นอย่างช้าๆ หรือในช่วงที่ความชันของสัญญาณมีค่าต่ำ และเพิ่มขนาดของสเต็ปเมื่อการเปลี่ยนแปลงของสัญญาณเกิดขึ้นอย่างรวดเร็ว หรือในช่วงที่ความชันของสัญญาณมีค่าสูง

ในขณะที่ความถี่ในการแซมปลิ้งสูงขึ้นผลต่างระหว่างค่าแซมปลิ้งข้างเคียงจะน้อยลงระบบการเข้ารหัสแบบอะแด็ปทีฟเดลต้ามอดูเลชั่น จะพิจารณาจากจุดนี้คือใช้ความถี่ในการแซมปลิ้งให้สูงขึ้นและเข้ารหัสของผลต่างของสัญญาณเพื่อส่งออกไปด้วย 1 บิท

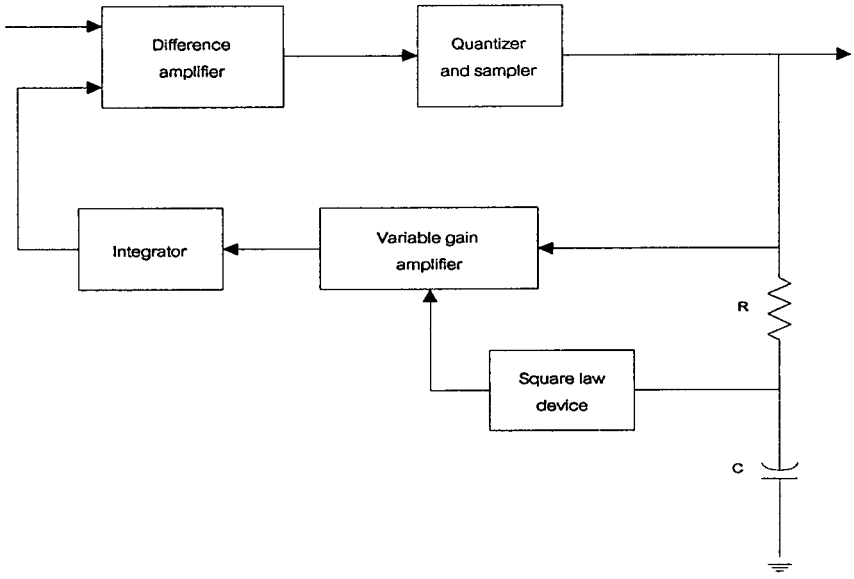
โครงสร้างของการเข้ารหัส/ถอดรหัสแบบอะแด็ปทีฟเดลต้ามอดูเลชั่น คอมพาราเตอร์, อินทิเกรเตอร์, D/Aคอนเวอร์เตอร์ และวงจรโลจิกที่จำเป็น



รูปที่ 2.18โครงสร้างของการเข้ารหัส/ถอดรหัสแบบอะแด็ปทีฟเดลต้ามอดูเลชั่น คอมพาราเตอร์, อินทิเกรเตอร์, D/Aคอนเวอร์เตอร์ และวงจรโลจิกที่จำเป็น

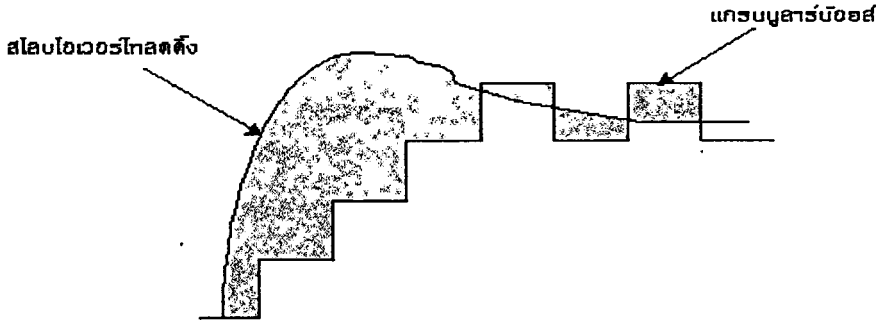
กรณีที่เข้ารหัสโดยใช้เพียง 1 บิท จะกำหนดขนาดของสเต็ปจากขบวนพัลส์ที่เข้ามาก่อน กรณีที่มีพัลส์เหมือนกันอย่างต่อเนื่องจะกำหนดว่าเกิดโอเวอร์โหลดขึ้นและจะเพิ่มขนาดของสเต็ปให้กว้างขึ้น ในทางตรงกันข้ามถ้าเกิดมีพัลส์สลับกันอย่างต่อเนื่องก็จะต้องลดขนาดของสเต็ปให้แคบลง

บล็อกไดอะแกรมแบบหนึ่งของระบบอะแด็ปทีฟเดลต้ามอดูเลชั่นแสดงดังรูปที่2.19 ที่สามารถปรับขนาดของสเต็ปตามลักษณะของสัญญาณขนาดของสเต็ปจะเปลี่ยนแปลงได้โดยการควบคุมเกนของอินทิเกรเตอร์ ซึ่งเกนจะมีค่าต่ำเมื่อแรงดันเป็น0 และเกนจะเพิ่มขึ้นเมื่อแรงดันควบคุมเพิ่มมากขึ้น

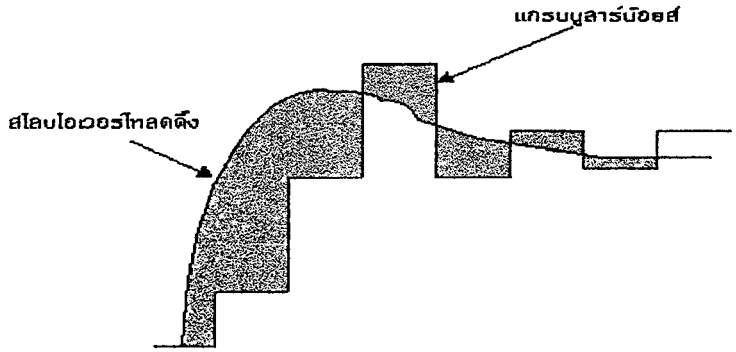


รูปที่ 2.19 บล็อกไดอะแกรมแบบหนึ่งของ อะเด็ปทีฟเดลต้ามอดูเลชั่น

วงจรควบคุมเกนประกอบด้วย R, C และอุปกรณ์อื่นๆ เมื่อสัญญาณอินพุทคงที่หรือเปลี่ยนแปลงช้าๆ เอาท์พุทของมอดูเลเตอร์จะเป็นพัลส์บวกและลบสลับกันตลอดเวลา พัลส์เหล่านี้เมื่ออินทิเกรตโดยR, C แล้วจะได้เอาท์พุทออกมาเฉลี่ยเป็น0 V. เกนคอนโทรลจะต่ำ ดังนั้นสเต็ปของ ตัวเก็บค่า(accumulator) จะต่ำด้วย



รูปที่2.20 ความผิดพลาดในการจัดระดับสัญญาณแบบลีเนียร์เดลต้ามอดูเลชั่น (LDM)



รูปที่2.21 ความผิดพลาดในการจัดระดับอะแด็ปทีฟเดลต้ามอดูเลชั่น

ในกรณีที่จะเกิดสโลปโอเวอร์โหลด เอาท์พุทของมอดูเลเตอร์จะเป็นพัลส์บวกหรือลบต่อเนื่องกัน อินทิเกรเตอร์จะอินทิเกรตให้เอาท์พุทโวลเตจออกมา จะไปเพิ่มเกนของวงจรขยายทำให้ขนาดของสเต็ปเพิ่มขึ้น ทำให้เกิดสโลปโอเวอร์โหลดลดลง

สำหรับดีมอดดูเลเตอร์ในระบบอะแด็ปทีฟเดลต้ามอดูเลชั่น จะมีวงจร อะแด็ปทีฟเกน(adaptive gain)เช่นเดียวกับด้านตัวเข้ารหัส(encoder) ทำให้เอาท์พุทมีลักษณะเดียวกับด้านตัวเข้ารหัส

การเข้ารหัสแบบอะแด็ปทีฟเดลต้ามอดูเลชั่น สามารถทำได้หลายวิธี เช่น ไฮบริดเดลต้ามอดูเลชั่น(Hybrid Companding Delta Modulation)

2.7 ไฮบริดเดลต้ามอดูเลชั่น(Hybrid Companding Delta Modulation)

ลักษณะโดยทั่วไปของ ไฮบริดเดลต้ามอดูเลชั่น(HCเดลต้ามอดูเลชั่น)

ระบบการเข้ารหัสดิจิตอลสำหรับส่งเสียงพูดต่างๆนั้น เดลต้ามอดูเลชั่น เป็นวิธีการหนึ่งที่มีประสิทธิภาพมาก เดลต้ามอดูเลชั่นเป็นระบบการจัดระดับที่คาดการณ์อย่างง่าย หรือกล่าวอีกอย่างก็คือ เดลต้ามอดูเลชั่น ก็คือ ดิฟเฟอเรนเชียลพัลส์โค้ดมอดดูเลชั่นบิทนั่นเอง แต่ระบบ เดลต้ามอดูเลชั่น นี้ง่ายกว่าและถูกกว่าพัลส์โค้ดมอดดูเลชั่น และ ดิฟเฟอเรนเชียลพัลส์โค้ดมอดดูเลชั่น

ทั้งที่เดลต้ามอดูเลชั่น มีข้อได้เปรียบหลายประการ แต่เหตุผลหนึ่งซึ่งทำให้ เดลต้ามอดูเลชั่น ไม่ได้รับการยอมรับอย่างกว้างขวางในอดีตคือ พิสัยทางพลศาสตร์(dynamic)แคบ จากความยุ่งยากในการจัดระดับซึ่งทำให้ขนาดของสเต็ปคงที่ ทำให้เกิด noise 2 ชนิดคือ

- แกรนนูลาร์น้อยส์(granular noise) เกิดจากการจำกัดสเต็ปไซส์(step size) ของตัวเข้ารหัส
- สโลปโอเวอร์โหลดดิ้ง(slope overload noise) เกิดเมื่อความชันของสัญญาณมากกว่าที่เดลต้ามอดูเลชั่น จะตามทัน

ปัญหานี้สามารถบรรเทาได้โดยการจัดระดับขั้นของการจัดระดับที่เหมาะสมก็คือ ขนาดของระดับการเปลี่ยนแปลงตามขนาดและความชันของสัญญาณอินพุท มีการวิจัยโดยการ คอมแพนดิ้ง ทางพยางค์(syllabic) หรือ คอมแพนดิ้งชั่วขณะ(instantaneous) ในการเปลี่ยนแปลงระดับการจัดระดับ(quantizing)

ขั้นตอนในการเปลี่ยนแปลงให้เหมาะสมที่ใช้ในการคอมแพนดิ้งทางพยางค์จะปรับเปลี่ยนขนาดของสเต็ปในการจัดระดับค่อนข้างช้าที่อัตราของพยางค์(syllabic rate) ตัวอย่างคือ CSVD

การคอมแพนดิ้งชั่วขณะนั้น ขนาดของขั้นในการจัดระดับจะเปลี่ยนแปลงอย่างทันที ณ ช่วงเวลาแซมปลิ้ง ซึ่งขึ้นอยู่กับการเปลี่ยนแปลงของอินพุทตัวอย่างคือ CFDM และ HIDM

ทั้งระบบอะแด็ปทีฟเดลต้ามอดูเลชั่น ที่การคอมแพนดิ้งทางพยางค์และการคอมแพนดิ้งชั่วขณะ มีคุณลักษณะดีกว่าระบบเดลต้ามอดูเลชั่น เชิงเส้นที่มิได้มีการดัดแปลง(nonadaptive linear delta modulation LDM) แต่ก็ยังมีข้อบกพร่องบางอย่าง เช่น อะแด็ปทีฟเดลต้ามอดูเลชั่นไม่สามารถตามอินพุทที่มีการเปลี่ยนแปลงอย่างทันทีทันใด เนื่องจากขนาดของระดับขึ้นอยู่กับความชันเฉลี่ยของอินพุทในช่วงเวลาหนึ่ง

ในทางตรงกันข้ามถ้าใช้การคอมแพนดิ้งชั่วขณะสำหรับการเปลี่ยนแปลงของการจัดระดับและมีการจำกัดขนาดพื้นฐานของแต่ละขั้น ระบบอะแด็ปทีฟเดลต้ามอดูเลชั่นจะทำงานดีสำหรับเสียงพูดเพียง 1ส่วน แต่จะเกิด น้อยส์ที่ยอมรับไม่ได้ในส่วนอื่นๆของเสียงพูด เนื่องจากโดยทั่วไปจะมีความแตกต่างอย่างมากในทางพิสัยทางพลศาสตร์ของสัญญาณเสียง และขนาดของระดับขั้นของอะแด็ปทีฟเดลต้ามอดูเลชั่น ไม่สามารถหาขีดจำกัดที่ได้ผลดีที่สุดสำหรับสัญญาณต่างๆ และการเปลี่ยนแปลงขนาดของแต่ละขั้นเกิดขึ้นอย่างรวดเร็ว ทำให้เกิดสภาวะไม่ดีในส่วนของเสียงพูดในภาวะคงที่

ในโครงงานนี้จะแสดงระบบอะแด็ปทีฟเดลต้ามอดูเลชั่น ซึ่งใช้การคอมแพนดิ้งแบบพยางค์และแบบชั่วขณะ ระบบนี้คือ HCDM (Hybrid Companding Delta Modulation) พบว่าHDCMมีข้อดีกว่าระบบ CVSD และ CFDM โดยไม่คำนึงถึงสภาวะของช่วงสัญญาณ

ขั้นตอนของHCDM

อินพุทแซมปลิ้ง(input sampling – s_n)จะถูกเปรียบเทียบกับค่าประมาณของตัวมันซึ่งได้จากการเพิ่มหรือลดค่าโดยประมาณก่อนหน้าที่เวลาการสุ่มตัวอย่าง(sampling time) แต่ละช่วงด้วยขนาดระดับหนึ่งขั้นระดับ โดยขึ้นอยู่กับเครื่องหมายของความแตกต่างระหว่างสัญญาณ และค่าโดยประมาณข่าวสารทางเครื่องหมาย(sign information) ขนาด1บิทต่อการสุ่มตัวอย่างแต่ละครั้งจะถูกส่งผ่านช่องสัญญาณไบนารี่ไปยังเครื่องรับ ความสามารถพิเศษของระบบHCDMคือการใช้งานคอมแพนเดอร์แบบพยางค์และแบบชั่วขณะพร้อมกัน ส่วนแรกใช้เพื่อปรับปรุงขนาดของระดับขั้นพื้นฐานของควอนไตเซอร์ ขึ้นอยู่กับการเปลี่ยนแปลงของสัญญาณอินพุท และส่วนหลังใช้เพื่อการเปลี่ยนแปลงระดับขั้นทุกๆการสุ่มตัวอย่าง

ขนาดของระดับพื้นฐาน A หาได้จากค่ารากที่2 ของกำลังสองเฉลี่ยของสโลปเอนเนอจี้(slope energy) E ของสัญญาณเสียงที่ลาดหมายโดยการผ่านตัวกรองความถี่ต่ำ จะถูกเปลี่ยนแปลงครั้งหนึ่งทุกๆ Mตัวอย่างซึ่งคือ

$$A=\alpha E \qquad (2.16)$$

เมื่อ α คือ scale factor

$$E = [\sum_{i=1}^{M-1} (s_i - s_{i-1})^2 / M - 1]^{1/2} \qquad (2.17)$$

เมื่อs_i คือตัวอย่าเสียงที่ถูกประมาณเป็นลำดับที่i

ด้วยขนาดระดับขั้นพื้นฐานA การคอมแพนดิ้งชั่วขณะถูกกระทำโดยกฎทางตรรกขึ้นอยู่กับเอาท์พุท3บิทที่ตามกันมาตามตารางที่2.2 output bit(b_n)ถูกสร้างในตัวเข้ารหัสของHCDM

$$b_n = \mathrm{sgn}(s_n - d_n) \qquad (2.18)$$

โดย $s_n = \exp(-\beta T)s_{n-1} + b_{n-1}d_{n-1}$

$d_n = A\gamma_n$

$\gamma_n = k_n\gamma_{n-1}$

และ $k_n = f(b_n, b_{n-1}, b_{n-2})$ (2.19)

เมื่อ β คือส่วนกลับของค่าคงตัวเวลารั่วไหลของวงรอบการประมาณค่า(leakage time constant of the prediction loop)

d_n คือ step sizeของการสุ่มตัวอย่างลำดับที่ n

k_n คือ ตัวประกอบการคูณ ขึ้นอยู่กับค่าปัจจุบันและอีก 2 บิทก่อนหน้า

d_n (HCDM step size) ในทางปฏิบัติขึ้นอยู่กับค่า E (signal slope energy) ซึ่งเปลี่ยนแปลงอย่างช้าๆที่อัตราพยางค์ และตัวประกอบระดับขั้นγ_nเปลี่ยนแปลงอย่างทันทีทันใดทุกๆการสุ่มตัวอย่าง

b_{n-2}	b_{n-1}	b_n	Multiplication Factor (k)
+	+	+	1.50
-	-	-	1.50
-	-	+	1.00
+	+	-	1.00
-	+	+	0.66
+	-	-	0.66
-	+	-	0.66
+	-	+	0.66

ตารางที่2.2 คอมแพนดิ้งโลจิก(Companding Logic)ของ HCDM

สามารถประมาณค่าการเปลี่ยนแปลงความชันของอินพุทด้วยแบบการไปข้างหน้าโดยตรงในกรณีนี้ตัวเข้ารหัสจะซับซ้อนกว่าเพราะข่าวสารทางความชันจะถูกส่งกลับเอาท์พุทแบบไบนารี่ของHCDM หลังจากการเข้ารหัสและการมัลติเพล็กซ์ การได้เปรียบในการประมาณค่าแบบไปข้างหน้าจะน้อยลงเมื่อเปรียบเทียบกับแบบป้อนกลับ(feedback mode)แต่กระนั้น codec ของเคลต้ามอดูเลชั่น เป็นแบบพื้นฐานซึ่งตามความชันของอินพุทมากกว่าขนาดของมัน การใช้งานของข่าวสารทางความชันจะถูกใช้งานมากกว่า ทั้งๆที่ขั้นตอนของHCDM จะถูกอธิบายในรูปแบบของดิจิตอลแต่ก็สามารถทำงานในรูปแบบของอนาล็อกได้ด้วย

2.8 การหาขีดจำกัดที่ดีที่สุดของพารามิเตอร์และการพิจารณาคุณสมบัติของระบบ (Parameter Optimization and System Performance)

มาพิจารณาถึงค่าตัวแปรที่สำคัญของ HCDM ด้วยการจำลองระบบโดยใช้คอมพิวเตอร์ เพื่อให้ได้คุณสมบัติของระบบที่ดีที่สุด เราจะใช้ค่า SQNR เพื่อทำการวัดคุณสมบัติของระบบ

$$SQNR = \frac{\sum_{i=1}^{N} (s_i)^2}{\sum_{i=1}^{N} (s_i - s'_i)^2} \qquad (2.20)$$

เมื่อ s_i คือ สัญญาณอินพุท

s'_I คือ สัญญาณที่ถูกถอดรหัส

N คือ จำนวนสัญญาณอินพุททั้งหมดที่ถูกสุ่มตัวอย่าง

การหา SQNR พบว่าค่า สเกลแฟคเตอร์(scale factor (α))ของ สเต็ปไซส์(A) มีบทบาทสำคัญในการควบคุม ควอนไตซิ่งน้อยส์ ในระบบนี้ว่าขนาดของระดับเบื้องต้นถูกกำหนดโดยวงรอบปิดถ้า α ถูกกำหนดไว้ต่ำมาก เอาท์พุทของดีโคดเดอร์จะไม่สามารถติดตามสัญญาณอินพุทได้ดี ทำให้เกิด สโลปโอเวอร์โหลดดิ้ง มาก ในทางตรงกันข้าม ถ้าตั้ง α ไว้สูงมากทำให้ขนาดของระดับเบื้องต้นมีขนาดใหญ่ ทำให้เอาท์พุทของตัวถอดรหัสมีค่ามากเมื่อเปรียบเทียบกับอินพุท และประสิทธิภาพของตัวถอดรหัสจะต่ำ

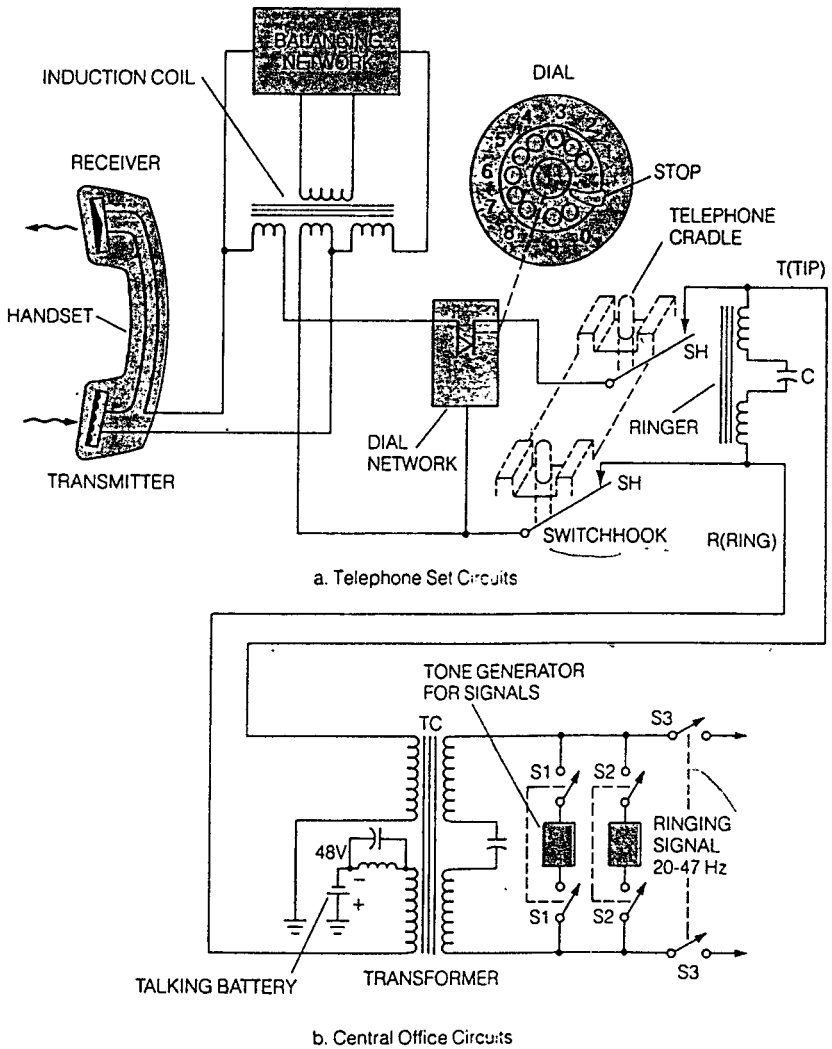
2.9 ระบบโทรศัพท์

2.9.1 โลคัลลูป (Local Loop)

คำจำกัดความของโลคัลลูป หมายถึงการเชื่อมต่อแบบที่ใช้คู่สาย 1 คู่สายเชื่อมต่อระหว่างโทรศัพท์กับส่วนกลาง

โทรศัพท์แต่ละเครื่องนั้นจะติดต่อกับส่วนกลาง (Central Office) ซึ่งที่ส่วนกลางนี้จะมีอุปกรณ์สวิทช์ชิ่ง อุปกรณ์ส่งสัญญาณ และมีแบตเตอรี่ที่จ่ายไฟกระแสตรงไปเลี้ยงเครื่องโทรศัพท์ ดังแสดงในรูปที่ 2.22

โทรศัพท์แต่ละเครื่องนั้นจะต่อกับส่วนกลางผ่านทาง โลคัลลูป ซึ่งมีสายอยู่สองสาย เรียกว่า " คู่สาย " โดยใน 1 คู่สายจะมีสายหนึ่งเรียกว่า ทิป (เขียว) (Tip) และอีกสายหนึ่งคือ ริง (แดง) (Ring) ซึ่งหมายส่วนทิปและริงของปลั๊กที่ใช้ในสวิทช์บอร์ด (switch board) แบบแมนนวล โดยริงต่อกับไฟ -48V(DC) ส่วน ทิปต่อลงกราวด์ที่ชุมสายปลายทาง สวิทช์ในส่วนกลางนั้นมีหน้าที่ในการเชื่อมต่อการเรียกจากโทรศัพท์เครื่องหนึ่งไปยังเครื่องที่ถูกเรียก เมื่อสถาปนาการเชื่อมต่อได้แล้วโทรศัพท์ทั้งสองเครื่องจะสื่อสารกันโดยใช้ไฟเลี้ยงจากส่วนกลางโดยมีหม้อแปลงเป็นตัวคัปปลิ้ง



รูปที่2.22การเชื่อมต่อระหว่างโทรศัพท์กับส่วนกลาง

ขั้นตอนในการโทร

1. เมื่อวางแฮนด์เซ็ตของโทรศัพท์บนตัวแคร่ น้ำหนักของมันจะทำให้ปุ่มสวิตช์สุดต่ำลงและเป็นการเปิด (Open) วงจร ตอนนี้เรียกว่าออนสุด(On hook)วงจรระหว่างโทรศัพท์และส่วนกลางจะเปิดวงจร

ส่วนวงจรกระดิ่ง(Ringer Circuit) จะยังคงต่ออยู่กับส่วนกลาง เมื่อแฮนด์เซ็ตถูกยกขึ้นจากแคร่ จะทำให้เกิดการครบวงจรทำให้เกิดกระแสไหลในวงจร ตอนนี้เรียกว่า Off hok ซึ่งในขณะยกหูนี้ระดับแรงดันไฟฟ้าระหว่าทิปกับริงมีค่าลดลงเหลือ 5 ถึง 10 โวลท์แล้วแต่ชนิดของโทรศัพท์ และจะมีกระแสไฟตรง ขนาด 20 mA ไหลวนอยู่ในลูป(คำว่า On hook และ Off hook มาจากในยุคแรกของโทรศัพท์ ตัวรับจะแยกออกต่างหากและแขวนอยู่บนสวิทช์สุด ปัจจุบันเลิกใช้ แล้วแต่ยังเรียกกันอยู่)

2.ส่วนชุมสายโทรศัพท์ สามารถตรวจจับการยกหู และตรวจหาตำแหน่งของผู้ที่เป็นฝ่ายเรียกได้ จากนั้นชุมสายส่งสัญญาณให้หมุนหรือกดหมายเลข สัญญาณนี้เรียกว่า ไดอัลโทน (dial tone)

3.จากนั้นเมื่อชุมสายได้รับหมายเลขที่หมุนหรือกดเข้ามาแล้ว ชุมสายทำหน้าที่ให้อุปกรณ์สวิตช์ชิ่งในชุมสายหาตำแหน่งของวงจรของหมายเลขที่ต้องการติดต่อโดยชุมสายทำการพิจารณาก่อนว่าเครื่องโทรศัพท์ของผู้ถูกเรียกตามหมายเลขนั้น กำลังใช้งานอยู่หรือไม่ ถ้าอยู่ในขณะกำลังใช้งาน ชุมสายจะส่งสัญญาณสายไม่ว่างให้ผู้เรียกทราบว่าไม่สามารถทำการติดต่อได้ สัญญาณสายไม่ว่างนี้เรียกว่า บิซซี่โทน (busy tone)

ในกรณีที่เครื่องโทรศัพท์ตามหมายเลขไม่อยู่ในการใช้งาน ชุมสายจะทำการส่งสัญญาณออกไปสองทิศทางดังนี้

- ส่งสัญญาณเรียกให้รับ (ringing tone) เพื่อแจ้งให้ผู้ถูกเรียกมารับโทรศัพท์

-ส่งสัญญาณเรียกกลับ (ringback tone) เพื่อแจ้งให้ผู้เรียกรับทราบว่า ได้ทำการติดต่อสำเร็จแล้ว

2.9.2 การส่งหมายเลขโทรศัพท์

มีการส่งแบ่งได้เป็น 2 แบบ

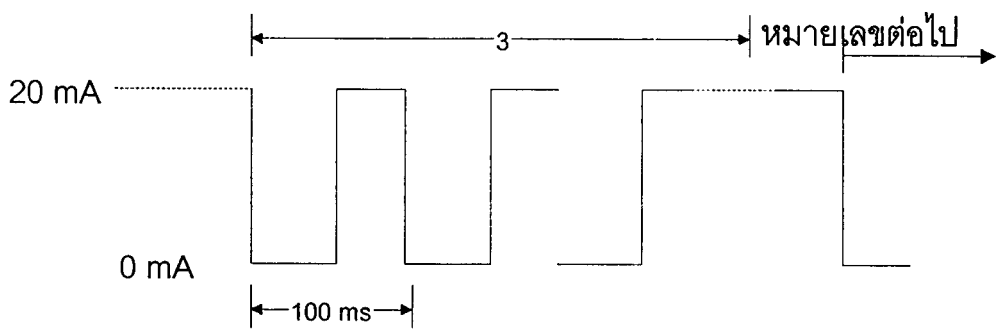
1.แบบเป็นพัลส์ ที่เกิดจากกระแสลูป ซึ่งมีในโทรศัพท์แบบหมุน

2.แบบเป็นความถี่ของแต่ละหมายเลข ซึ่งมีในโทรศัพท์แบบกดปุ่ม

ซึ่งในแต่ระแบบมีรายละเอียดดังนี้

(1)สัญญาณจากโทรศัพท์แบบหมุน (rotary dial)

โทรศัพท์ชนิดนี้สร้างสัญญาณจากกระแสลูป โดยต่อเข้ากับอุปกรณ์สวิทช์ชิ่งทำหน้าที่ เปิด และ ปิด เข้ากับกลไกการหมุนหมายเลขในเครื่อง ทำให้กระแสพัลส์ตอบสนองเข้ากับกระแสที่หมุน

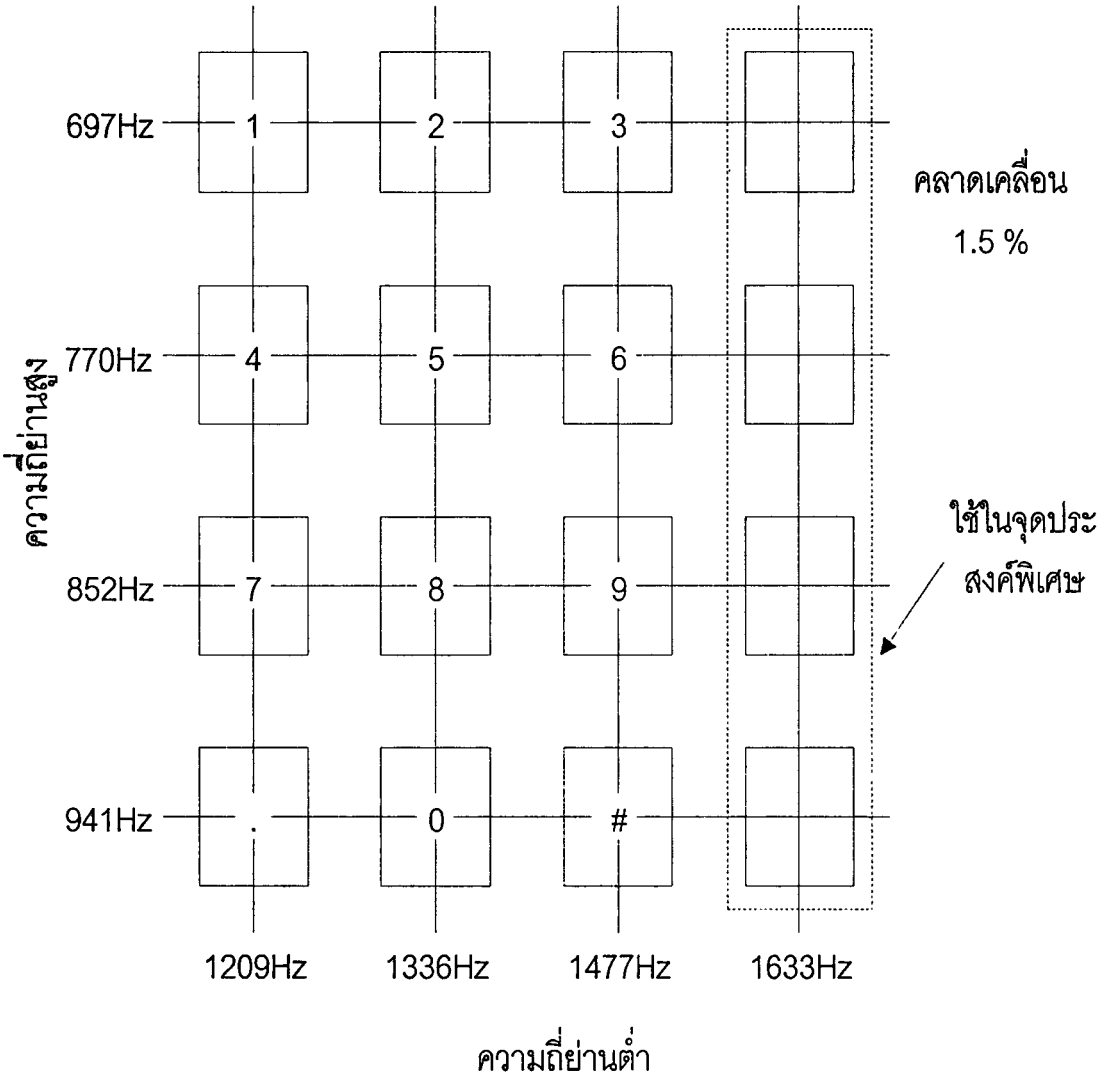


รูปที่ 2.23 แสดงการหมุนหมายเลข 3

จากรูป 2.23 แต่ละช่วงเวลาของสัญญาณพัลส์ 1 ลูก มีค่า 100 มิลลิวินาที และมีค่าดิวตี้ไซเคิลเท่ากับ 40 เปอร์เซ็นต์ และจากการใช้มือหมุนพบว่าช่วงเวลาเฉลี่ยก่อนที่จะหมุนหมายเลขแต่ละค่า มีค่าประมาณ 0.5 วินาที - 3 วินาที

(2) สัญญาณจากการโทรศัพท์แบบกดปุ่ม (touch-tone)

โทรศัพท์ชนิดนี้ใช้การสร้างสัญญาณ DTMF (Dual - Tone Multiple Frequency) จากการกดปุ่มของแต่ละหมายเลข และแต่ละปุ่มที่กดมีความถี่เฉพาะสร้างขึ้นมา 2 ความถี่นำมารวมกันและส่งออกไปพร้อมกัน เช่นถ้า

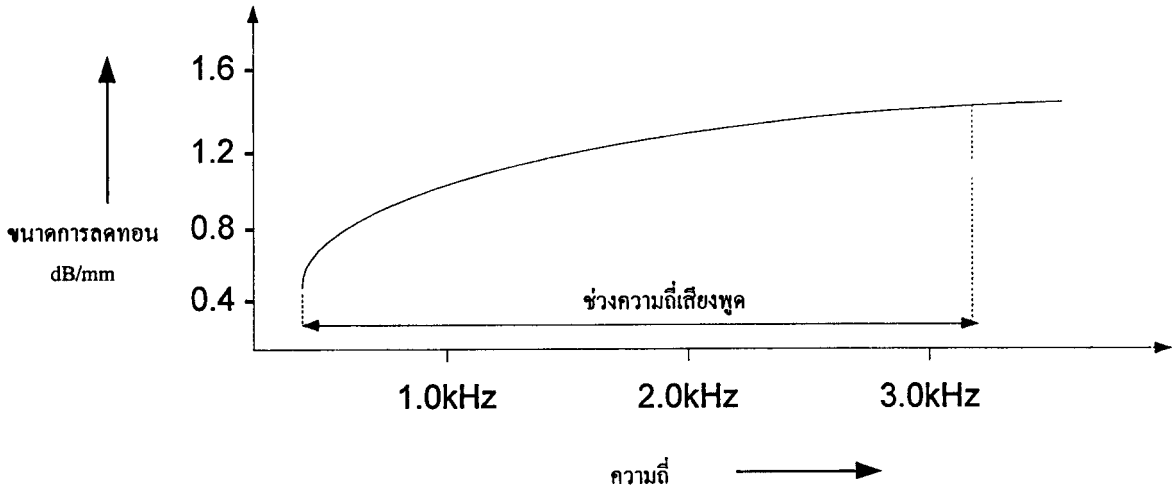


รูปที่ 2.24 แสดงตำแหน่งของปุ่มและความถี่

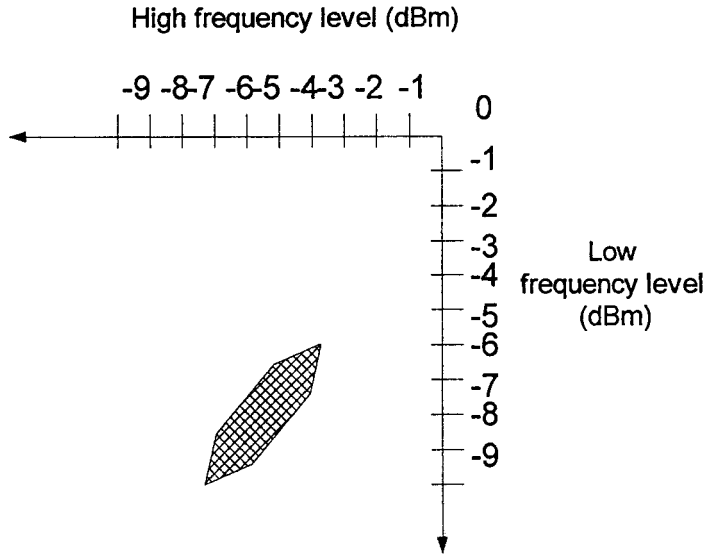
กดหมายเลข 5 ก็จะมีความถี่ 1336 Hz และ 770 Hz ส่งออกมา

ในการสร้างสัญญาณนั้นจะให้ด้านความถี่สูงมีขนาดใหญ่กว่าความถี่ต่ำ 2 dB เพื่อชดเชยการสูญเสียในการส่งเพราะสัญญาณที่มีความถี่สูงจะถูกลดทอนในขนาดที่มากกว่าความถี่ต่ำ ดังรูปที่2.25 แสดงกราฟการลดทอน-ความถี่สำหรับเคเบิลขนาด 0.9 มิลลิเมตร

คู่ความถี่ที่ส่งออกไปนั้นมีค่าอยู่ประมาณ 40 มิลลิวินาที และช่วงเวลาระหว่างหมายเลขมีค่า 60 มิลลิวินาที เป็นอย่างต่ำ โทรศัพท์แบบกดปุ่มทำงานเร็วกว่าแบบหมุนอยู่ถึง 10 เท่า



รูปที่2.25 แสดงกราฟการลดทอน-ความถี่สำหรับเคเบิลขนาด 0.9 มิลลิเมตร



รูปที่2.26แสดงถึงระดับในการส่งสัญญาณ DTMF

จากรูปที่ 2.26 แสดงถึงระดับในการส่งสัญญาณ DTMF พื้นที่แลเงาคือระดับของคู่สัญญาณที่ดีที่สุด (Optimum) และ ระดับแตกต่างกันได้มากที่สุด

การตอบรับการเรียก

เมื่อทางฝ่ายผู้ถูกเรียกยกหูขึ้นแล้วทำให้เกิดกระแสไหลเข้าไปในโทรศัพท์ ส่วนกลางจะยกเลิกสัญญาณริงกิง(ringing signal) และสัญญาณริงแบค(ring back)จากวงจรการสนทนา

ผู้พูดจะพูดในส่วนที่เรียกว่า ด้านส่ง (Transmitter) ซึ่งทำหน้าที่เปลี่ยนเสียงให้กลายเป็นไฟฟ้า เวลาฟังจะฟังจาก ด้านรับ (Receiver) ซึ่งจะเปลี่ยนจากสัญญาณไฟฟ้าเป็นเสียง

ในการพูดจะมีสัญญาณส่วนหนึ่งป้อนกลับเข้าไปยังส่วนรับฟังของผู้พูดเอง ซึ่งเราเรียกว่า ไซด์โทน (side tone) ไซด์โทนมีความสำคัญมากในการทำให้ผู้พูดตัดสินใจว่าจะพูดให้ดังขึ้นหรือเบาลงเพราะถ้าไซด์โทนมีขนาดมากคนพูดจะพูดเบาลง ในทางกลับกันก็จะพูดดังขึ้นถ้าสัญญาณตัวนี้น้อยไป

การยกเลิกการเรียก

การติดต่อจะยุติลงเมื่อคู่สนทนาวางหูลง (On hook) ส่วนกลางก็จะทำการยกเลิกการติดต่อ ในบางแห่งจะยุติเมื่อทั้งคู่วางหู บางแห่งจะยุติเมื่อผู้เรียกวางหูเพียงฝ่ายเดียว

2.9.3สัญญาณรบกวนในช่องสัญญาณเสียง

ในระบบสื่อสารนั้นจะต้องเผชิญกับสัญญาณที่ไม่ต้องการทั้งหลาย เรียกว่า สัญญาณรบกวน (noise) ซึ่งอาจทำให้ข่าวสารบิดเบือนไป อาจมีเหตุจาก ฟ้าผ่า,การเหนี่ยวนำของสายไฟฟ้ากำลัง,สัญญาณรบกวนจากทรานซิสเตอร์,จุดเชื่อมหลุด สิ่งเหล่านี้จะถูกรวมเข้าไปในวงจรขยายพร้อมกับสัญญาณเสียงพูดด้วย

สัญญาณรบกวนแยกได้เป็น 3 ประเภท คือ

(1)เทอร์มอลและช็อตน็อยส์ (Thermal and shot noise) เป็นสัญญาณรบกวนที่เกิดจากการเปลี่ยนแปลงของสัญญาณพาหะเมื่อผ่านอุปกรณ์ต่างๆในเน็ตเวิร์ค

(2) อินเตอร์มอดูเลชั่นและครอสทอล์ค น็อยส์ (Intermodulation and cross-talk noise) เป็นสัญญาณรบกวนที่เกิดจากการแทรกแซงระหว่างสัญญาณที่ต้องการกับสัญญาณอื่นๆในเน็ตเวิร์ค หรือเกิดจากการมอดดูเลทสัญญาณอื่นที่ใกล้กับสัญญาณพาหะเข้าไปในวงจรโทรศัพท์ด้วย

(3) อิมพัลส์น็อยส์ (Impulse noise) เกิดจากระดับแรงดันไฟฟ้าเพิ่มขึ้นและลดลงอย่างรวดเร็ว อาจเรียกว่าทรานเซียนส์ (transients) ก็ได้ ถ้ามีสัญญาณรบกวนแบบนี้ในวงจรโทรศัพท์ มีสาเหตุได้หลายอย่างคือเกิดจากอุปกรณ์สวิช์ชิ่งแบบกลไกในชุมสาย(เช่นจากรีเลย์) การรั่วไหลในสายไฟฟ้า หรือเกิดจากฟ้าผ่าก็ได้

(4)ปัญหาเรื่องเสียงสะท้อน (Echo)

-เมื่อฝ่ายพูด พูดออกมา สัญญาณเสียงถูกส่งผ่านคู่สายบนจนไปถึงชุมสายด้านรับที่วงจรภายในต่อถึงกัน เสียงพูดของผู้พูดส่วนหนึ่งไปถึงผู้ฟัง ขณะเดียวกันจะมีสัญญาณอีกส่วนหนึ่งที่หลุดออกมาและวนกลับไปวงจรด้านผู้พูดใหม่อีก ดังแสดงในรูปที่ 2.27

สัญญาณที่วนกลับนี้ เราเรียกว่า " เสียงสะท้อน " (Echo) และถูกภาคขยายของคู่สายด้านล่างขยายสัญญาณให้สูงเกือบเท่ากับเสียงที่พูดออกมา แต่ปรากฏว่าผู้พูดจะไม่ได้ยินเสียงสะท้อนนี้ เพราะว่าค่าหน่วงเวลา (Delay) ระหว่างหูฟังของโทรศัพท์มีค่าน้อยมากๆ จนฟังได้ว่าเป็นเสียงที่เกิดขึ้นในเวลาเดียวกัน

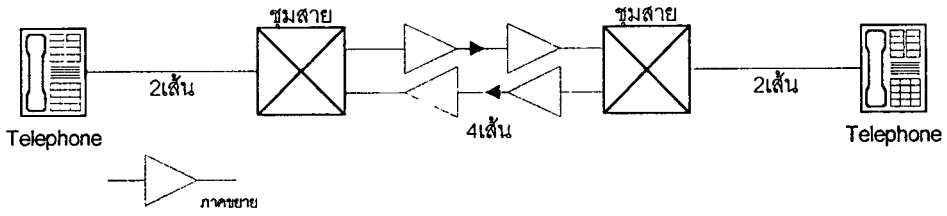
ถ้าเป็นการโทรระหว่างประเทศที่ระยะทางไกลมากๆ เช่น

-การสื่อสารผ่านดาวเทียมจะมีค่าหน่วงเวลาประมาณ 0.5 วินาที

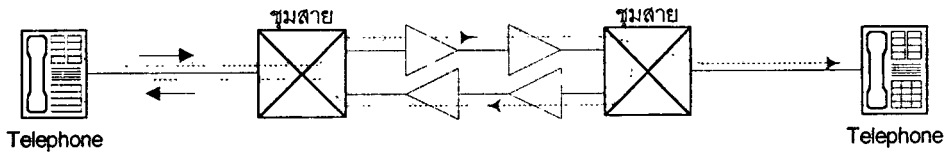
-การสื่อสารผ่านคลื่นวิทยุมีค่าหน่วงเวลา 0.25 วินาที

ในกรณีเหล่านี้มีผลให้ได้ยินเสียงพูดของตัวเองหลังจากที่ได้พูดไปแล้ว สัญญาณสะท้อนนั้นเกิดจากการไม่แมทช์กันของอิมพีแดนซ์ของสายส่ง ซึ่งมักจะปรากฏที่ส่วนวงจรไฮบริด ที่ส่วนอินเตอร์เฟสระหว่างวงจร 2 คู่สาย กับวงจร 4 คู่สาย และยังเกิดจากความไม่สมดุลย์ในเน็ตเวิร์ค การแก้ปัญหาทำโดยการใช้อุปกรณ์ลดเสียงสะท้อน (echo suppressor) เพื่อช่วยให้เสียงสะท้อนหมดไป

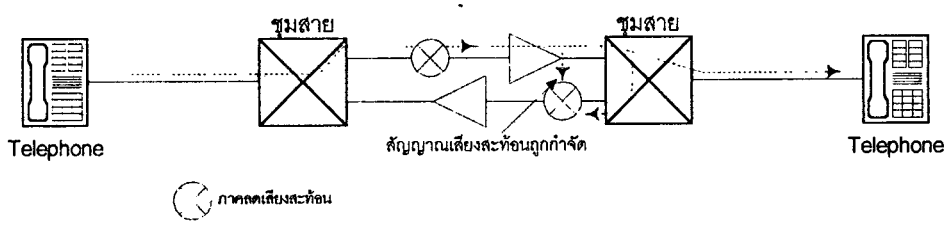
a)เครือข่ายโทรศัพท์ท้องถิ่น



b)เครือข่ายโทรศัพท์ทางไกล



c)การเกิดเสียงสะท้อน



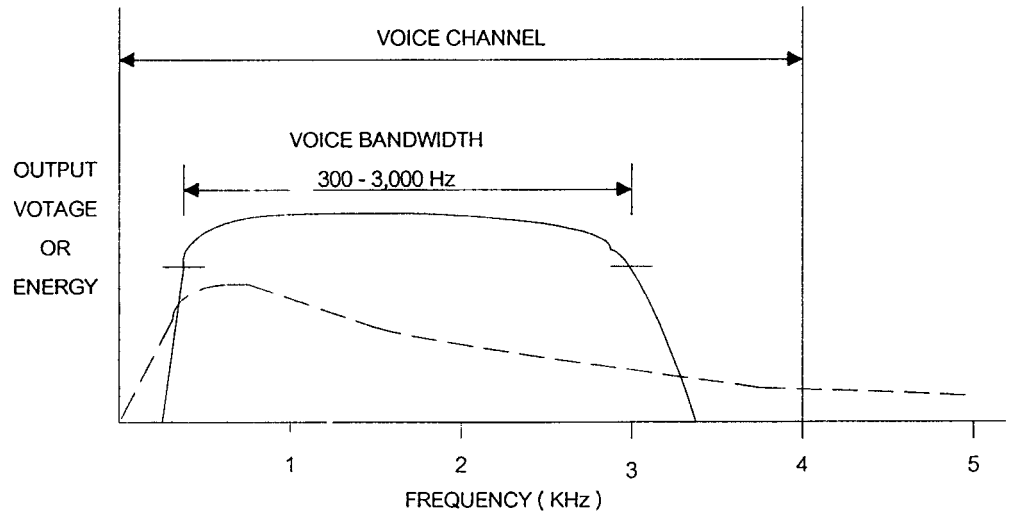
d)การลดเสียงสะท้อน

รูปที่ 2.27แสดงการเกิดและการลดสัญญาณสะท้อนระหว่างคู่สายโทรศัพท์

2.9.4สัญญาณในระบบโทรศัพท์

1.การส่งสัญญาณเสียง

สัญญาณเสียงจัดเป็นสัญญาณอนาล็อกซึ่งมีความถี่ตั้งแต่ 100 Hz ถึง มากกว่า 6,000 Hz ซึ่งมีขนาดและความถี่เปลี่ยนแปลงตามในรูปที่ 2.28



รูปที่2.28แสดงแบนด์วิดท์ของสัญญาณเสียง

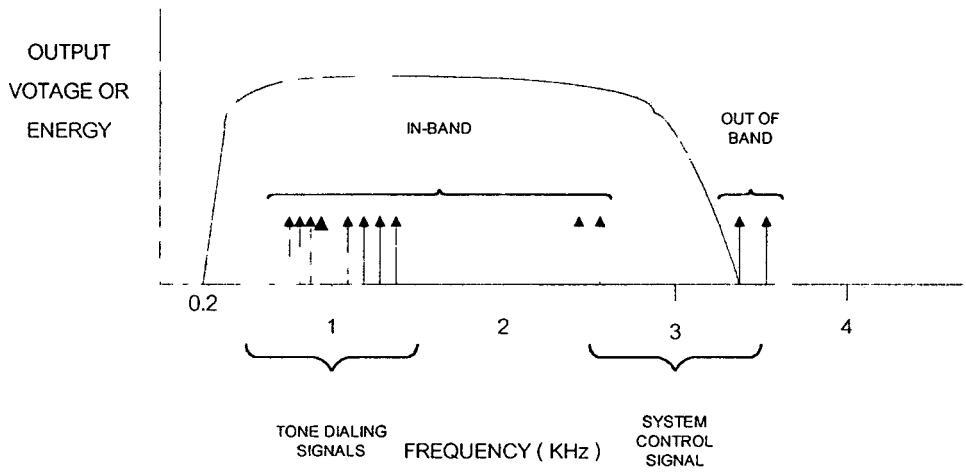
แต่ว่าสัญญาณเสียงที่มีผลต่อการตีความนั้นส่วนใหญ่มีความถี่ในช่วง 200 Hz ถึง 4,000 Hz ในระบบโทรศัพท์จะใช้ช่องสัญญาณเสียงในช่วง 0 - 4,000 Hz (VF channel)

2. ชนิดของสัญญาณ

2.1 In-band signal คือสัญญาณในระบบโทรศัพท์ที่มีความถี่ 300 ถึง 3,000 Hz ซึ่งเป็นช่วงที่ให้มีความถี่เสียงผ่านไปได้

2.2 Out-of-band signal คือสัญญาณที่อยู่ในช่อง VF แต่ไม่อยู่ในช่วง 300Hz ถึง 3,000Hz

สัญญาณเสียงทั้งหมดและสัญญาณ DTMF จัดเป็นสัญญาณ อินแบนด์ ส่วนสัญญาณอื่นบางตัวเป็นสัญญาณอินแบนด์บางตัวเป็นสัญญาณเอาท์ออฟแบนด์



รูปที่ 2.29แสดงการจำแนกประเภทของสัญญาณ

3. สัญญาณโทน (tone signalling)

สัญญาณโทนเป็นสัญญาณที่ใช้แสดงสภาวะของเครื่องที่ถูกเรียก และเพื่อการควบคุม อาจเป็นสัญญาณความถี่เดียวหรือเกิดจากการรวมกันของสัญญาณหลายความถี่ก็ได้ มีทั้งแบบต่อเนื่องและแบบ เบิร์ส (bursts) คือมีและหยุดในอัตราที่ต่างๆกันเช่น ไดอัลโทน เป็นสัญญาณที่ต่อเนื่องเกิดจากการรวมกัน (combination) กันของสัญญาณความถี่ 350Hz และ 4440Hz

สัญญาณต่างๆที่ใช้ดังแสดงในตาราง 2.3

Tone	Frequency(Hz)	On Time(s)	Off Time(s)
Dial	350 + 440	Continuous	
Busy	480 + 620	0.5	0.5
Ringback, Normal	440 + 480	2	4
Ringback, PBX	440 + 480	1	3
Congestion(Toll)	480 + 620	0.2	0.3
Recoder(Local)	480 + 620	0.3	0.2
Receiver Off – hook*	1400 + 2060	0.1	0.1
	2450 + 2600		
No such number	200 to 400	Continuous, Frequency modulated at 1Hz Rate	

* Receiver Off – hook is a very loud tone, 0 dBm per frequency.

ตารางที่2.3แสดงคุณสมบัติของสัญญาณในระบบโทรศัพท์

2.9.5 วงจรไฮบริด (Hybrid)

หน้าที่ของวงจรไฮบริดคือการอินเตอร์เฟสระหว่างวงจร 2 สาย กับวงจร 4 สาย เพื่อทำให้การสื่อสารเป็นแบบฟูลดูเพล็กซ์ (Full duplex) และนอกจากนี้แล้วยังสามารถใช้วงจรขยายสัญญาณในการขยายสัญญาณได้ในกรณีที่ต้องส่งสัญญาณเป็นระยะทางไกลเพราะสัญญาณจะมีขนาดลดลงตามระยะทางที่เพิ่มขึ้น

การทำงานของวงจร Hybrid

วงจร Hybrid ประกอบด้วย ขดลวดเหนี่ยวนำจำนวนมาก ใช้หลักการของการเปลี่ยนแปลงกระแสในขดลวด ทำให้เกิดการเปลี่ยนแปลงสนามแม่เหล็ก และเหนี่ยวนำขดลวดอีกขดทำให้เกิดกระแสไหลในวงจร

จากรูปถ้ามีสัญญาณส่งมาที่ 7 , 8 เกิดกระแสไหลผ่านขดลวด A , B เกิดการเหนี่ยวนำไปยังขดลวด C , D การเหนี่ยวนำที่ขดลวด D ทำให้เกิดกระแสไหลไปยังวงจร 2-wire ซึ่งติดกับเทอร์มินอล 1 - 2

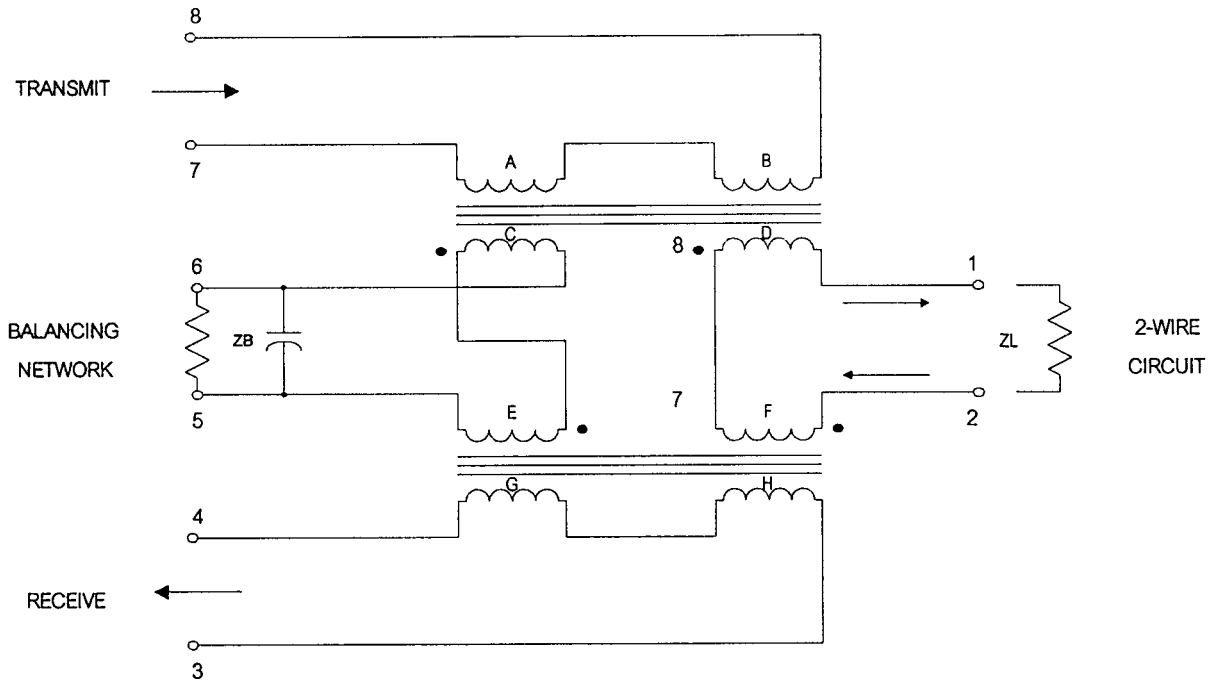
กระแสที่เกิดจากขดลวด D ที่ไหลผ่านขดลวด F เหนี่ยวนำให้เกิดศักย์ไฟฟ้าในขดลวด H และในทำนองเดียวกัน มีการเหนี่ยวนำในขดลวด C ทำให้กระแสไหลเข้าวงจรสมดุลย์(Balance Network) และขดลวด E Zbของวงจรสมดุลย์มีค่าเท่ากับ Zl ของ line impedance

เมื่อให้จำนวนรอบของขดลวด C กับ D เท่ากัน จำนวนรอบของขดลวด E กับ F เท่ากัน และจำนวนรอบของขดลวด G กับ H เท่ากัน

ขดลวด C กับ E มีการต่อกลับขั้วกับขดลวด D กับ F เพราะฉะนั้นสัญญาณที่เหนี่ยวนำในขดลวด G จะมีเฟสตรงกันข้ามกับขดลวด H ดังนั้นเมื่อสัญญาณมีขนาดเท่ากัน แต่มีเฟสตรงกันข้าม สัญญาณจึงหักล้างกันหมดไป ผลก็คือสัญญาณที่ถูกส่งมาจะปรากฏที่เทอร์มินอล 7- 8 และถูกส่งไปยังเทอร์มินอล 1 - 2 แต่จะไปปรากฏที่เทอร์มินอล 3 - 4 ซึ่งจะส่งไปยังด้านรับ แต่ระดับสัญญาณที่เทอร์มินอล 1 - 2 จะเป็นครึ่งหนึ่งจะถูกส่งไปยังวงจรสมดุลย์ (Balance network)

เมื่อมีสัญญาณมาปรากฏที่เทอร์มินอล 1-2 จะมีกระแสไหลผ่านขดลวด B และขดลวด F กระแสที่ไหลในขด F จะเหนี่ยวนำให้เกิดกระแสไหลในขด H ซึ่งทำให้เกิดกระแสไหลไปยังด้านรับและกระแสตัวเดียวกันนี้ได้ไหลผ่านขด G ทำให้เกิดการเหนี่ยวนำเกิดกระแสในขด E จะไหลไปยังวงจรสมดุลย์ และไปยังขด C และเหนี่ยวนำให้เกิดกระแสในขด A

โวลท์เตจที่เกิดขึ้นที่ขดลวด A จะมีเฟสตรงข้าม 180^0 กับโวลท์เตจที่เกิดในขด B เนื่องจากการต่อสลับกันของขด C กับ E ทำให้ไม่มีสัญญาณปรากฏที่ 7-8 ซึ่งเป็นด้านส่ง



รูปที่2.30 แสดงวงจรไฮบริด

2.9.6 การกำหนดเลขหมายโทรศัพท์

ปัจจุบันการใช้โทรศัพท์นั้นสามารถเรียกติดต่อทางไกลได้แม้ในต่างประเทศ ดังนั้นจึงมีการกำหนดเลขหมายโทรศัพท์ รหัสหมายเลขของพื้นที่ รวมทั้ง prefix ต่างๆจะต้องกำหนดให้อยู่ในมาตราฐานเดียวกันทั้งหมด

จึงจะทำให้โทรศัพท์ทุกๆเลขหมายทั่วโลกสามารถติดต่อถึงกันได้ มาตราฐานการกำหนดเลขหมายดังกล่าวจะถูกกำหนดโดยผ่านหน่วยงาน CCITT (The International Telegraph and Telephone Consultative Committee) ซึ่งได้กำหนดไว้ดังนี้

-Station Number เป็นเลขหมายของผู้เช่าโทรศัพท์ที่อยู่ในชุมสายท้องถิ่นประกอบด้วยเลขหมาย 4 ตัว เขียนเป็นสัญลักษณ์ได้คือ xxxx

-Office Code (หรือ Exchange code) เป็นรหัสของชุมสายท้องถิ่น ประกอบด้วยเลขหมาย 1 - 3 ตัว สำหรับประเทศไทยใช้เลขหมาย 3 ตัว ในเขตโทรศัพท์นครหลวงและใช้เลขหมาย 2 ตัวในเขตโทรศัพท์ภูมิภาค เขียนเป็นสัญลักษณ์ คือ AAA

-Area Code (หรือ trunk code) เป็นรหัสทางไกลที่ถูกกำหนดให้ใช้ใน Local Network หนึ่งๆ ประกอบด้วยเลขหมาย 1 - 3 ตัว เขียนเป็นสัญลักษณ์ คือ OOO

-Country Code เป็นรหัสทางไกลของประเทศ ประกอบด้วยเลขหมาย 1 - 3 ตัว เขียนเป็นสัญลักษณ์ คือ CCC

-Trunk Prefix เป็นเลขหมายนำ ที่ใช้สำหรับการเรียกทางไกลภายในประเทศ ประกอบด้วยเลขหมาย 1 - 3 ตัว เขียนเป็นสัญลักษณ์ คือ TTT

-International Prefix เป็นเลขหมายนำ ที่ใช้สำหรับการเรียกทางไกลต่างประเทศ ประกอบด้วยเลขหมาย 1 - 3 ตัว เขียนเป็นสัญลักษณ์ คือ III

-Directory Number (หรือ Subscriber number) คือเลขหมายโทรศัพท์ที่ใช้สำหรับการเรียกภายใน Local Network เดียวกัน ประกอบด้วย Office Code กับ Station Number

Directory Number : AAA-XXX

-National Number คือหมายเลขโทรศัพท์ที่ใช้สำหรับการเรียกทางไกลภายในประเทศ ประกอบด้วย Area Code กับ Directory Number

Number : OOO-AAA-XXX

-International Number คือหมายเลขโทรศัพท์ที่ใช้สำหรับการเรียกทางไกลต่างประเทศ ประกอบด้วย Country Code กับ National Number

International Number : CCC-OOO-AAA-XXX

บทที่ 3
การคำนวณและการสร้าง

ในโครงงานนี้ได้สร้างโปรแกรมที่ทำหน้าที่เป็นเซิร์ฟเวอร์และไคล์เอนด์ โดยการส่งข้อมูลระหว่างไคล์เอนด์และเซิร์ฟเวอร์จะใช้โปรโตคอล TCP/IP โดยจะใช้พอร์ตหมายเลข 2543 เป็นพอร์ตส่งข้อความควบคุม และพอร์ตหมายเลข 2541 และ 2542 เป็นพอร์ตที่ใช้ส่งข้อมูลเสียง

โปรแกรมที่สร้างขึ้นในโครงงานนี้ สร้างขึ้นโดยใช้เดลไฟล์ 2.0 เป็นเครื่องมือในการสร้าง ซึ่งแต่ละโปรแกรมที่สร้างขึ้นโดยเดลไฟจะเรียกว่าโปรเจค(project) โดยใน หนึ่งโปรเจคจะประกอบด้วยไฟล์ สกุล .DFM ที่ทำหน้าที่เก็บข้อมูลส่วนที่เป็นหน้าต่างของโปรเจคนั้น และไฟล์สกุล .PAS ทำหน้าที่เป็นไฟล์หลักในการทำงานของโปรเจค นอกจากนี้ในหนึ่งโปรเจคอาจจะประกอบด้วยไฟล์สกุล .PAS อื่นๆ ที่ช่วยการทำงานของไฟล์ที่ทำหน้าที่ไฟล์

ในบทนี้จะอธิบายลำดับขั้นตอนการทำงานของโปรเจคทั้งหมดในโครงงาน พร้อมทั้งภาพประกอบขณะที่โปรแกรมทำงานบนหน้าจอและผังการทำงาน(flow chart) ส่วนโค้ดของโปรแกรมทั้งหมด ในโครงการนี้ได้สร้างโปรเจคหลักที่ทำหน้าที่เป็นเซิร์ฟเวอร์และไคลเอนด์ ของการส่งข้อมูลเสียง โดยข้อมูลเสียงทั้งหมดที่ใช้ในโครงงานนี้เป็นข้อมูลของสัญญาณเสียงที่ถูกแซมปลิ้งด้วยความถี่ 11025 Hz และ เก็บข้อมูล แบบ 8 บิตต่อหนึ่งแซมเปิ้ล และเป็นสัญญาณเสียง 1 ช่องสัญญาณ(โมโน) และโปรเจคที่ทำหน้าที่วาดกราฟของแซมเปิ้ลจากไฟล์ และส่วนของโปรแกรมเซิร์ฟเวอร์ และไคล์เอนด์ ที่ทำหน้าที่บีบอัดและขยายข้อมูลที่ถูกบีบอัด

โปรแกรมในโครงงานนี้ ได้แก่โปรแกรมต่างๆดังนี้

1.โปรแกรมเซิร์ฟเวอร์ ทำหน้าที่รอรับการร้องขอ และเชื่อมต่อสัญญาณเสียงจากคู่สายโทรศัพท์ไปให้โปรแกรมไคล์เอนด์ รวมทั้งควบคุมการรับส่งข้อมูลเสียงขณะสนทนา

2.โปรแกรมไคล์เอนด์ ทำหน้าที่ร้องขอการติดต่อไปยังเซิร์ฟเวอร์ และควบคุมการรับส่งข้อมูลเสียงขณะสนทนา

3.โปรแกรมดูข้อมูลในไฟล์สกุล .wav ทำหน้าที่แสดงค่าทุกแซมเปิ้ลในไฟล์ และสามารถเปรียบเทียบให้เห็นลักษณะของข้อมูลเสียงก่อนถูกบีบอัดและข้อมูลเสียงที่ผ่านกระบวนการบีบอัดข้อมูลได้

4.ส่วนของโปรแกรมเซิร์ฟเวอร์ และไคล์เอนด์ ทำหน้าที่บีบอัดและขยายข้อมูลโดยวิธีการอเด็ฟทีฟเดลต้ามอดดูเลชั่นใช้บีบอัดและขยายข้อมูลเสียง

3.1 โปรแกรมเซิร์ฟเวอร์

โปรแกรมนี้ทำหน้าที่รอรับการติดต่อจากโปรแกรมไคล์เอนด์ โดยเมื่อให้บริการกับไคล์เอนด์ จะรับหมายเลขโทรศัพท์ที่ไคล์เอนด์ต้องการจะติดต่อ ส่งให้กับไมโครโปรเซสเซอร์ ที่ควบคุมวงจรโทรศัพท์ผ่านทางพอร์ตอนุกรม จากนั้นไมโครโปรเซสเซอร์ ที่ควบคุมวงจรโทรศัพท์ จะหมุนหมายเลขโทรศัพท์ และส่งผลให้กับโปรแกรมเซิร์ฟเวอร์ทราบ หลังจากนั้นจะสั่งให้รีเลย์เชื่อมต่อ เสียงจากคู่สายโทรศัพท์ผ่านวงจรไฮบริดจ์ และผ่านไปยังซาวด์การ์ดของเซิร์ฟเวอร์ เพื่อแปลงข้อมูลเสียงด้วยวิธีการพัลซ์โค้ดมอดดูเลชั่น ในรูปแบบของไฟล์แบบ .wav

เนื่องจากซาวด์การ์ดไม่สามารถอัดเสียงหรือเล่นเสียงพร้อมกันได้ดังนั้นโปรแกรมเซิร์ฟเวอร์ จะทำงานร่วมกับโปรแกรมไคล์เอนด์ในการควบคุมการส่งข้อมูลเสียง และการทำงานของซาวด์การ์ดทั้งเครื่องเซิร์ฟเวอร์และไคล์เอนด์ โดยการติดต่อผ่านซอกเก็ต 3 ซอกเก็ต คือ ซอกเก็ตหมายเลข 2541ใช้รับข้อมูลเสียง ซอกเก็ต 2542 ใช้ส่งข้อมูลเสียง ซอกเก็ต 2543 ใช้ส่งข้อความควบคุมโปรแกรมเซิร์ฟเวอร์และไคล์เอนด์ใช้อัลกอริทึมที่สัมพันธ์กันในการรับและส่งข้อมูลเสียงระหว่างกันนั่นคือ ขณะสนทนา ทั้ง 2 โปรแกรม จะคอยตรวจจับสัญญาณเสียงที่รับผ่านไมโครโฟนตลอดเวลาหากพบว่ามีระดับสัญญาณเสียงจะแจ้งผ่านซอกเก็ต 5243 ให้อีกฝ่ายยกเลิกการรับเสียงและให้รอรับข้อมูลเสียงแทน จนกระทั่งฝ่ายส่งหยุดส่งข้อมูล ฝ่ายที่รับข้อมูลจะยกเลิกการรับและเปลี่ยนไปตรวจจับสัญญาณเสียงดังเดิม

ก่อนการส่งข้อมูลผ่านเครือข่ายอินเตอร์เน็ตจะบีบอัดข้อมูลเสียงด้วยวิธีอะเด็ปทีฟเดลต้ามอดดูเลชั่น ซึ่งจะทำให้ข้อมูลมีขนาดเล็กลง 8 เท่า และทางฝ่ายรับ จะขยายก่อนมูลเสียงที่รับเข้ามาก่อนเล่นออกลำโพงการรับส่งข้อมูลผ่านทางพอร์ตอนุกรมสามารถปรับค่าได้ทั้งหมายเลขพอร์ต อัตราบอด จำนวนบิตข้อมูลและพาริตี้บิต โดยจะต้องเลือกให้สัมพันธ์กับค่าที่กำหนดไว้ในไมโครโปรเซสเซอร์

ในการสร้างโปรแกรมเซิร์ฟเวอร์นี้จะต้องใช้ไฟล์ต่างๆ มาใช้ร่วมกันเมื่อสร้างเป็นโปรแกรมเซิร์ฟเวอร์ ได้แก่

1.ไฟล์ Server2.pas เป็นไฟล์หลักในการทำงานของโปรแกรมนี้ จะเรียกใช้ฟังก์ชั่นต่างๆในไฟล์ ต่างๆ ที่จะกล่าวต่อไปเพื่อใช้ในการควบคุมด้านต่างๆ

2.ไฟล์ MmioWrap.pas ทำหน้าที่เก็บฟังก์ชั่นควบคุมการทำงานการอ่านข้อมูลจากไฟล์สกุล .wav และการย้ายข้อมูลไปใส่หน่วยความจำ

3.ไฟล์ Socket.pas และ Winsock.pas เป็นไฟล์ที่เก็บฟังก์ชั่นที่ควบคุมการทำงานเกี่ยวกับ โปรโตคอล TCP/IP ของ Windows95 โดยเฉพาะการรับส่งข้อมูลผ่านทางซ็อกเก็ต

4.ไฟล์ comdrv32.pas เป็นไฟล์ที่เก็บฟังก์ชั่นที่ควบคุมการทำงานเกี่ยวกับการรับส่งข้อมูล ผ่านพอร์ตอนุกรม

5.ไฟล์ Waveplay.pas ทำหน้าที่เก็บฟังก์ชั่นควบคุมการเล่นไฟล์เสียงของซาวด์การ์ด

6.ไฟล์ WaveWrap.pas ทำหน้าที่เก็บฟังก์ชั่นควบคุมซาวด์การ์ด

3.2 โปรแกรมไคล์เอนด์

โปรแกรมนี้จะทำหน้าที่เรียกและสั่งให้เซิร์ฟเวอร์ หมุนโทรศัพท์ ไปยังหมายเลขที่ผู้ใช้ต้องการ และทำการควบคุมการรับส่งข้อมูล ขณะสนทนาด้วยอัลกอริทึมที่สัมพันธ์กับโปรแกรมเซิร์ฟเวอร์ ซึ่งได้กล่าวมาแล้วในหัวข้อ 3.1

โปรแกรมไคล์เอนด์ จะใช้การบีบอัดข้อมูลและขยายข้อมูลแบบเด็ฟทีฟเดลต้ามอดดูเลชั่น เช่นเดียวกับโปรแกรมเซิร์ฟเวอร์

ในการสร้างโปรแกรมไคล์เอนด์นี้จะต้องใช้ไฟล์ต่างๆ มาใช้ร่วมกันเมื่อสร้างเป็นโปรแกรมไคล์เอนด์ ได้แก่

1.ไฟล์ Client2.pas เป็นไฟล์หลักในการทำงานของโปรแกรมนี้ จะเรียกใช้ฟังก์ชั่นต่างๆที่จะกล่าวต่อไปเพื่อใช้ในการควบคุมด้านต่างๆ

2.ไฟล์ MmioWrap.pas ทำหน้าที่เก็บฟังก์ชั่นควบคุมการทำงานการอ่านข้อมูลจากไฟล์สกุล .wav และการย้ายข้อมูลไปใส่หน่วยความจำ

3.ไฟล์ Socket.pas และ Winsock.pas เป็นไฟล์ที่เก็บฟังก์ชั่นที่ควบคุมการทำงานเกี่ยวกับโปรโตคอล TCP/IP ของ Windows95 โดยเฉพาะการรับส่งข้อมูลผ่านทางซ็อกเก็ต

4.ไฟล์ Waveplay.pas ทำหน้าที่เก็บฟังก์ชั่นควบคุมการเล่นไฟล์เสียงของซาวด์การ์ด

5.ไฟล์ WaveWrap.pas ทำหน้าที่เก็บฟังก์ชั่นควบคุมซาวด์การ์ด

3.3 การทำงานของไมโครโปรเซสเซอร์ ที่ควบคุมวงจรโทรศัพท์

ไมโครโปรเซสเซอร์ที่ทำหน้าที่ควบคุมวงจรโทรศัพท์จะรอรับข้อมูลที่ส่งมาจากโปรแกรมเซิร์ฟเวอร์ โดยโปรแกรมเซิร์ฟเวอร์จะส่งข้อมูลเป็นตัวอักษรตามมาตรฐานแอสกีออกมา ไมโครโปรเซสเซอร์จะตรวจว่าตัวอักษรนั้นคือคำสั่งให้ทำชับรูทีนใด และจะกระโดดไปทำงานชับรูทีนนั้น จนกระทั่งทำงานเสร็จไมโครโปรเซสเซอร์จะส่งตัวอักษรนั้นกลับมาให้เซิร์ฟเวอร์ทราบว่าทำงานที่ชับรูทีนนั้นเสร็จเรียบร้อยแล้ว ตัวอักษรที่ใช้สั่งให้ไมโครโปรเซสเซอร์ ทำงานมีดังนี้

ตัวเลข 0-9 สั่งให้ไมโครโปรเซสเซอร์ ควบคุม ไอซี 5088 ให้สร้างสัญญาณ DTMF ตามตัวเลข 0-9

อักษร C สั่งให้ รีเลย์ทำงาน โดยให้เชื่อมคู่สายโทรศัพท์เข้ากับวงจรโทรศัพท์

อักษร D สั่งให้ รีเลย์เลิกทำงาน โดยให้ตัดคู่สายโทรศัพท์ออกจากวงจรโทรศัพท์

3.4 โปรแกรมดูข้อมูลในไฟล์สกุล .wav

โปรแกรมถูกสร้างเพื่อเปรียบเทียบข้อมูลจากไฟล์เสียงสกุล .wav ที่ยังไม่ถูกบีบอัด กับไฟล์เสียงที่ถูกผ่านขยายหลังจากเป็นข้อมูลที่ถูกบีบอัด โดยจะอ่านข้อมูลจากไฟล์สกุล .wav ในแบบ 8 บิต โมโน และถูกแซมปลิ้งด้วยความถี่ 11025 Hz เท่านั้น โดยจะอ่านข้อมูลทีละไบท์แล้วนำข้อมูลแต่ละไบท์มาลากเส้นต่อจุดจนกระทั่งหมดข้อมูลในไฟล์

3.5 การบีบอัดและขยายข้อมูลโดยวิธีการอะแด็ปทีฟเดลต้ามอดดูเลชั่น

การบีบอัดและขยายข้อมูลด้วยวิธีการอะแด็ปทีฟเดลต้ามอดดูเลชั่นจะทำงานเฉพาะไฟล์ .wav ที่แซมปลิ้งด้วยความถี่11025Hz 8บิตต่อแซมเปิ้ล ที่เป็นโมโน

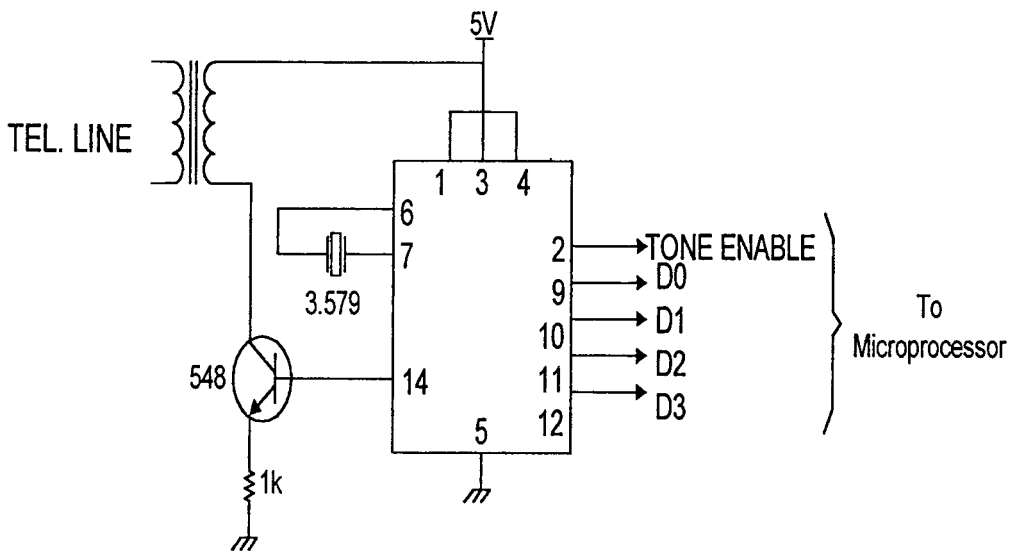
วิธีการอะแด็ปทีฟเดลต้ามอดดูเลชั่นวิธีการนี้จะทำให้แซมเปิ้ลขนาด1ไบต์เหลือเพียง1บิตที่จะเก็บในคอมเพรสไฟล์ ในแต่ละบิตจะแสดงค่าว่าเป็นการเพิ่มหรือลดลงจากสัญญาณเดิมซึ่งขนาดของระดับที่เพิ่มหรือลดนั้นเรียกว่า สเต็ปไซส์(STEP SIZE)

การคอมเพรสจะเริ่มต้นการทำงานโดยหาค่าสโลปเอเนจี้ (SLOPE ENERGY) และจากนั้นใช้ค่าสเกลแฟคเตอร์(SCALE FACTOR)ค่า 0.1 , 0.2 , 0.3 .... 2.5 มาเป็นตัวคูณกับค่าสโลปเอเนจี้แล้วหาค่าอัตราส่วนระหว่างสัญญาณเสียงต่อสัญญาณรบกวน(SQNR) และนำแต่ละค่าอัตราส่วนระหว่างสัญญาณเสียงต่อสัญญาณรบกวนมาเทียบกันแล้วใช้ค่าที่ทำให้ค่าดีที่สุดเป็นตัวคูณกับค่าสโลปเอเนจี้และใช้เป็นค่าสเต็ปไซส์ และสเต็ปไซส์นี้ปรับค่าได้ คือถ้าเกิดสโลบโอเวอร์โหลดดิง ติดต่อกันจะปรับขนาดของสเต็ปไซส์ขึ้น และถ้าเกิดพัลส์สลับกันต่อเนื่องก็จะลดขนาดของสเต็ปไซส์ลง ไฟล์ที่ได้จะมีขนาดลดลง8เท่า

ส่วนของการดีโค้ด จะอ่านข้อมูลทีละบิตและเมื่อมีข้อมูล1หรือ 0 ต่อเนื่องกันจะเพิ่มขนาดของสเต็ปไซส์ ถ้ามีข้อมูล1สลับ0ต่อเนื่องกันจะลดขนาดของสเต็ปไซส์ และนำสเต็ปไซส์ที่ได้จากการเปรียบเทียบค่ามาบวกกับสัญญาณที่ประมาณค่าไว้ แล้วจะได้ไฟล์.wav กลับคืนมา

3.6 วงจรกำเนิดสัญญาณคู่ความถี่(DTMF)

ไอซีที่นำใช้ทำหน้าที่กำเนิดสัญญาณคู่ความถี่ใช้ไอซีเบอร์ TP5088 ที่สามารถนำมาประยุกต์ใช้ในงานโทรศัพท์ที่ถูกควบคุมด้วยไมโครโปรเซสเซอร์ โดยไม่จำเป็นต้องผ่านคีย์บอร์ด ในช่วงที่ขาโทนเอนนาเบิล (TONE ENABLE) อยู่ในสภาวะแรงดันไฟต่ำจะทำให้วงจรออลซิเลเตอร์ไม่ทำงานและจะไม่รับข้อมูลอินพุทที่เข้ามา ขณะที่มีการเปลี่ยนแปลงสภาวะจากแรงดันไฟต่ำไปแรงดันไฟสูงที่ขาโทนเอนนาเบิล ข้อมูลจะถูกป้อนเข้าไปในไอซีและสัญญาณคู่ความถี่ จะถูกสร้างจากความถี่ DTMF มาตรฐาน ซึ่งจะมีกลุ่มความถี่ต่ำและกลุ่มความถี่สูง



รูปที่3.1 แสดงวงจรกำเนิดสัญญาณคู่ความถี่

การกำเนิดสัญญาณคู่ความถี่ใช้ไมโครโปรเซสเซอร์ควบคุมมทางขาอินพุทและขาโทนเอนนาเบิลโดยจะป้อนข้อมูลเข้าทางขา D0 - D3 ก่อน แต่ว่า TP5088 จะยังไม่ทำงานและจะไม่รับข้อมูลอินพุทที่เข้ามา

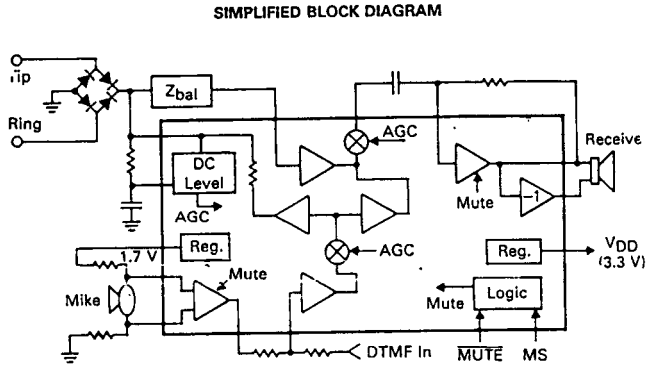
เพราะขาโทนเอนนาเบิลอยู่ในสภาวะแรงดันไฟต่ำ จากนั้นจะเปลี่ยนแปลงสภาวะจากแรงดันไฟต่ำไปแรงดันไฟสูงที่ขาโทนเอนนาเบิล เพื่อให้มีการรับข้อมูลทางขา D0 - D3 และสร้างสัญญาณคู่ความถี่ออกไปตามข้อมูลอินพุท D0 - D3 ตามตาราง

หมายเลข	Low Frequency	High Frequency	D3	D2	D1	D0
1	697	1209	0	0	0	1
2	697	1336	0	0	1	0
3	697	1477	0	0	1	1
4	770	1209	0	1	0	0
5	770	1336	0	1	0	1
6	770	1477	0	1	1	0
7	852	1209	0	1	1	1
8	852	1336	1	0	0	0
9	852	1477	1	0	0	1
0	941	1336	1	0	1	0

ตารางที่ 3.1 แสดงสัญญาณDTMFที่ได้จาก ไอซีเบอร์ 5088 เมื่อป้อนที่ขา D0 , D1 , D2 และ D3ด้วยค่าต่างๆ

3.7 วงจรควบคุมเสียงพูด MC34114

วงจรควบคุมเสียงพูดแบบสองทิศทาง (two way speech circuit) เป็นอีกส่วนหนึ่งภายในเครื่องโทรศัพท์ที่จัดว่ามีความสำคัญต่อการทำงานของตัวเครื่องโทรศัพท์ เพราะเป็นส่วนที่จะต้องทำงานเกี่ยวกับสัญญาณเสียงพูดที่เราพูดผ่านไมโครโฟน หรือสัญญาณเสียงที่จะได้ยินจากคู่สนทนา ข้อสำคัญของการออกแบบวงจรนี้ คือ การแมตชิ่งอิมพีแดนท์ของสายส่งสัญญาณ จากชุมสายกับอิมพีแดนท์ของวงจร ซึ่งจะต้องมีความใกล้เคียงกันมากที่สุด เพื่อประสิทธิภาพของการส่งสัญญาณ

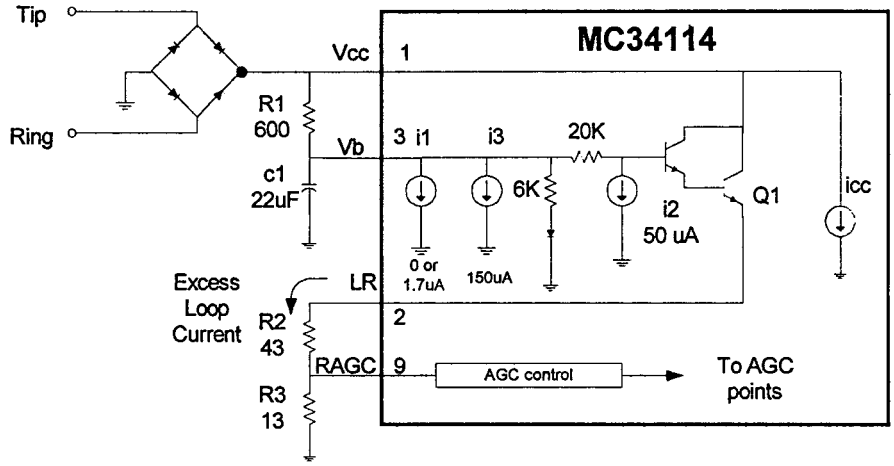


รูปที่ 3.2 แสดงบล็อกไดอะแกรมภายในของ MC34114

ไอซี MC34114 ประกอบด้วยวงจรควบคุมเสียงพูดที่มีวงจรไฮบริด (วงจรแยกระบบสายส่ง จาก 2W เป็น 4W) วงจรเชื่อมต่อกระแสไฟตรงที่ต่ออยู่กับสายทิปกับริง สามารถปรับแต่งอัตราการขยายสัญญาณของด้านส่ง ด้านรับและไซด์โทน (sidetone : การที่เสียงพูดของผู้พูดสามารถได้ยินในส่วนของผู้ฟัง เพื่อให้ทราบได้ว่าเราควรจะพูดดังค่อยขนาดไหนในการติดต่อกัน) มีส่วนวงจรชดเชยผลอันเนื่องมาจากความยาวของสายส่งสัญญาณ (line length compensatation) ที่อัตราการขยายเปลี่ยนแปลงตามกระแสในลูป รวมทั้งวงจรขยายไมโครโฟนแบบผลต่างเพื่อที่จะลดการรบกวน เนื่องจากความถี่วิทยุ

3.7.1 วงจรเชื่อมต่อกับไฟตรง

วงจรเชื่อมต่อกับไฟตรง (ขา 1, 2, 3) จะกำหนดคุณสมบัติของไฟตรงจากกระแสในลูป จากรูปที่ 3.3 ระดับแรงดันไฟตรงที่ V_{cc} ถูกจำกัดโดยการยกระดับแรงดันของขา 1 กับ ขา 2 บวกกับแรงดันตกคร่อม



รูปที่ 3.3 วงจรสมมูลย์ของการอินเตอร์เฟสกับคู่สายโทรศัพท์

R_2 และ R_3 ไอซี MC34114 ต้องการ I_{cc} เป็นกระแสไบอัสภายใน ซึ่งปกติมีค่าประมาณ 10 มิลลิแอมป์ เราสามารถที่จะลดกระแส I_{cc} หากจำเป็น โดยการเพิ่มค่า R_{12}

ในระหว่างการพูดและการส่งสัญญาณแบบพัลส์ ตัวกำเนิดกระแส I_1 ไม่ทำงาน การยกระดับแรงดันจะตกลงไป เนื่องมาจากขา B และ E ของทรานซิสเตอร์ Q_1 (ประมาณ 1.4 โวลท์) 1 โวลท์คร่อมความต้านทาน 20 กิโลโอห์ม และแรงดันตกคร่อม R_1 ซึ่งทำให้ V_{cc} จะเปลี่ยนแปลงตั้งแต่ 0.15 ไปจนถึงประมาณ 1.0 โวลท์ เมื่อกระแสลูปที่มาจากขั้วทิปกับริงมีค่าเกินกว่า I_{cc} จะต้องการ กระแสที่เกินจะไหลผ่าน Q_1, R_2 เพื่อให้เป็นไปตามคุณสมบัติของความสัมพันธ์ระหว่างกระแสกับแรงดัน

ในการส่งสัญญาณแบบโทน แหล่งจ่ายกระแส I_1 ทำงาน ทำให้มีกระแสไหลผ่าน R_1 เพิ่มขึ้น 1.7 มิลลิแอมป์ ยกระดับแรงดันขึ้นอีกประมาณ 1.0 โวลท์ (เมื่อ R_1 มีค่า 600 โอห์ม) คุณสมบัติพิเศษนี้เป็นการประกันได้ว่า เมื่อกระแสลูปมีค่าน้อย จะมีแรงดันที่ V_{cc} มากพอสำหรับสัญญาณ DTMF และแหล่งจ่ายไฟ

V_{DD} จะสามารถจ่ายแรงดันที่พอเพียงไปให้ส่วนแป้นกดสัญญาณภายนอก กระแส I_{cc} ในการทำงานแบบนี้จะเพิ่มขึ้นไปประมาณ 1.3 มิลลิแอมป์

ความต้านทาน R_1 ใช้ได้ตั้งแต่ 100 ไปจนถึง 1800 โอห์ม ถ้าใช้ค่าที่มากเกินไป กระแสที่ไหลไปยัง V_R จะมีค่าไม่เพียงพอ แต่ถ้าน้อยเกินไป การกรองที่ V_R จะไม่เป็นผล ถึงแม้ว่าจะมีการเพิ่มค่า C_1 ก็ตาม (สัญญาณเสียงพูดจะถูกกรองโดย V_R)

แรงดันตกคร่อม R_3 เป็นตัวควบคุมการทำงานของวงจร AGC (เป็นส่วนชดเชยผลอันเนื่องมาจากความยาวของสายส่งสัญญาณ) เมื่อความต้านทานที่ตกคร่อม RAGC เพิ่มขึ้นจากประมาณ 0.4 โวลท์ ไปเป็น 1.2 โวลท์ ส่วนควบคุมการทำงานของ AGC จะเปลี่ยนแปลงอัตราการขยายกระแสของ AGC ตั้งแต่ 1.0 ไปจนถึง 0.5 (ซึ่งจะลดอัตราการขยายของส่วนรับและส่งไปประมาณ 6 dB)

ค่าของ R_2 และ R_3 สามารถที่จะเปลี่ยนแปลงได้ เมื่อมีการเพิ่มเติมวงจรที่กระแสจากลูป อาทิเช่น ไมโครโปรเซสเซอร์ต่าง ๆ หรือเพื่อเปลี่ยนแปลงจุดเริ่มต้นการทำงานของวงจร AGC หากไม่มีการใช้งาน AGC ควรจะต่อขา 9 เข้ากับกราวน์เพื่อให้ได้อัตราการขยายที่สูงสุด หรือเข้ากับ V_R เพื่ออัตราการขยายต่ำสุด

3.7.2 ตัวจ่ายแรงดันคงที่

ไอซี MC34114 มีตัวจ่ายแรงดันคงที่ 2 ตัว เพื่อจ่ายแรงดันให้แก่ทั้งวงจรภายในและวงจรภายนอก ตัวจ่ายแรงดันคงที่ V_R จ่ายแรงดัน 1.7 โวลท์ ที่แรงดันสูงสุด 500 ไมโครแอมป์ ซึ่งผลที่ได้นี้จะนำไปใช้ไบอัสขา 10 (TXI) และไบอัสไมโครโฟน โดยปกติ V_R มีค่าน้อยกว่า V_{cc} ประมาณ 0.3 โวลท์ เมื่อ V_{cc} มีค่าน้อยกว่า 2.0 โวลท์

ตัวจ่ายแรงดันคงที่ V_{DD} จ่ายแรงดัน 3.3 โวลท์ ที่กระแสสูงสุด 1.0 มิลลิแอมป์ ในขณะใช้พูดแบบปกติ และกระแสสูงสุด 2.5 มิลลิแอมป์ ในการส่งสัญญาณแบบพัลส์หรือโทน ปกติเราใช้ V_{DD} ในการจ่ายพลังงานให้กับวงจรแป้นกดที่อยู่ภายนอก รวมทั้งวงจรอื่นที่ต่ออยู่ด้วยกัน ปกติ V_{DD} จะมีค่าน้อยกว่า V_{cc} ประมาณ 0.5 โวลท์ V_{DD} เป็นตัวจ่ายกระแสคงที่แบบขนาน ซึ่งจะเปลี่ยนไปเป็นค่าความต้านทานสูงโดยอัตโนมัติ เมื่อ V_{cc} มีค่าต่ำกว่า 1.4 โวลท์ คุณลักษณะนี้จะช่วยป้องกันการกินกระแสจากแบตเตอรี่ ช่วยคงหน่วยความจำของวงจรแป้นกด เมื่อ V_{cc} มีค่าเป็น 0 กระแสรั่วไหลจะมีค่า 0.02 ไมโครแอมป์ เมื่อป้อนค่าแรงดันไม่เกิน 6.0 โวลท์ เข้าที่ V_{DD} โดยที่ขา 17 เปิดวงจรหรือต่อกับ V_{DD} หากขา 17 ต่อลงกราวด์ กระแสหลายร้อยไมโครแอมป์ จะไหลเข้า V_{DD} และไหลลงกราวด์ที่ขา 17

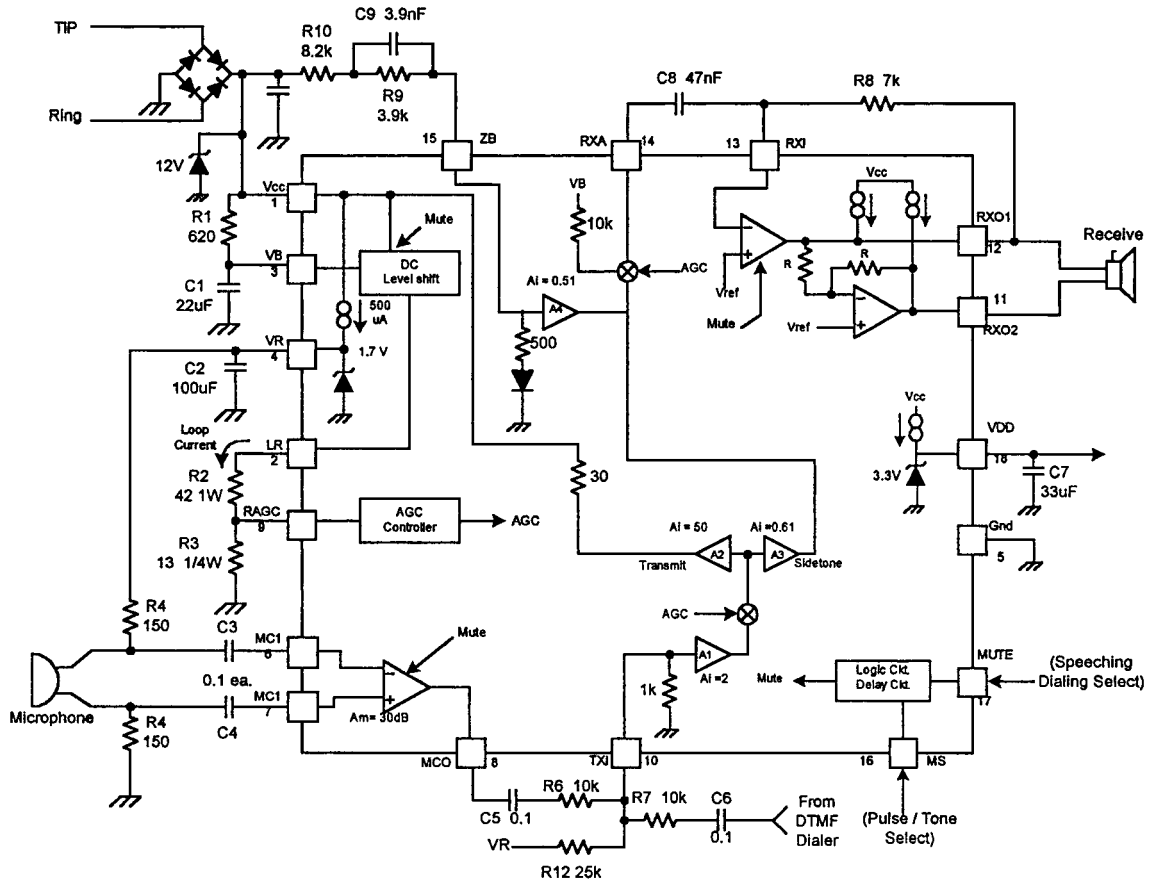
3.7.3 วงจรขยายสัญญาณจากไมโครโฟน

วงจรขยายสัญญาณจากไมโครโฟน (ขา 6, 7, 8) มีสัญญาณเข้าแบบผลต่าง(differential) ,สัญญาณออกแบบซิงเกิลเอนด์ และอัตราขยายภายในคงที่ +30 dB เอาต์พุตตรงเฟสกันกับ MC2 และกลับเฟสกันกับ MC1 มีความต้านทาน 20 กิโลโอห์ม และแมตชิ่งเป็นอย่างดีเพื่อ CMRR (Common mode Rejection) ที่สูง (ประมาณ 26 dB) เพื่อที่จะมีการขจัดสัญญาณจากการเหนี่ยวนำจากสายนำสัญญาณที่ไม่ต้องการ (CMRR มีค่าสูง) ไมโครโฟนจึงมีการไบอัสจากความต้านทานที่มีค่าเท่ากัน

เอาต์พุต (MCO) มีแรงดันไบอัสตรงอยู่ประมาณ 1.1 โวลท์ (เมื่อ V_{cc} มีค่ามากกว่า 3.0 โวลท์) มีอัตราการแกว่งประมาณ 2.0 โวลท์ (แกว่ง 500 มิลลิแอมป์ เมื่อ V_{cc} มีค่า 1.2 โวลท์) เอาต์พุตอิมพีแดนซ์

มีค่าประมาณ 270 โอห์ม และมีกระแสสูงสุดประมาณ 160 ไมโครแอมป์ ที่ 5 % ของ THD (Total Harmonic Distortion)

เมื่อ MC34114 อยู่ในระหว่างการส่งสัญญาณหมุน วงจรขยายไมโครโฟนจะถูกลดกำลังการส่งลงไปประมาณ 70 dB (300 – 4000 กิโลเฮิรตซ์) ซึ่งเพียงพอในการหยุดการทำงานของไมโครโฟนระดับแรงดันไฟตรงที่ MCO มีค่าประมาณ 80 มิลลิโวลท์ เมื่อถูกลดกำลังส่งลง



รูปที่ 3.4 แสดงบล็อกไดอะแกรมและอุปกรณ์ภายนอกของ MC34114

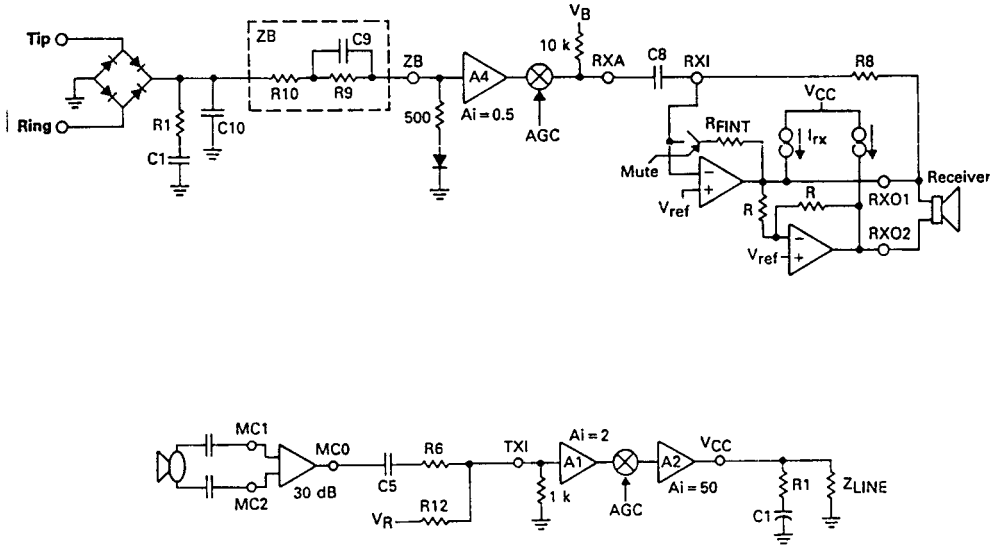
3.7.4 วงจรในการส่งสัญญาณ

วงจรที่ใช้ในการส่งสัญญาณออกไปมีอุปกรณ์ดัง แสดงในรูปแรงดันเอาต์พุตที่ MCO ถูกเปลี่ยนไปเป็นกระแสเข้า TXI โดย C_5 , R_6 และความต้านทานภายในของ TXI 1 กิโลโอห์ม A_1 และ A_2 เป็นอุปกรณ์ขยายกระแสที่มีอัตราขยายรวมกันเป็น 100 AGC ที่เข้ามามีค่าเป็น 1 เมื่อมีกระแสลูปน้อย และลดลงเป็น 0.5 เมื่อกระแสลูปมีค่าเพิ่มขึ้น ดังนั้นจะทำให้อัตราการขยายจาก TXI ไปจนถึง V_{cc} มีค่าตั้งแต่ 100 ถึง 50 เป็นผลทำให้กระแสที่ V_{cc} กระทำต่อ R_l และอิมพีแดนซ์ของสายส่ง(ประมาณ 600 โอห์ม) ก่อให้เกิดแรงดันที่ V_{cc} และเช่นกันที่ขั้วทิปกับริง ระดับแรงดันระหว่างขาMC1 - MC2 และขั้วทิปกับริงมีค่าตามสมการ

$$G_{TX} = (Am * 100 * AGC * R_l / Z_{LINE}) / (R_6 + 1000) \qquad (3.1)$$

เมื่อ Am เป็นอัตราการขยายของอุปกรณ์ขยายไมโครโฟน (31.1 V/V) ที่กระแสลูปมีค่าน้อยๆ G_{TX} ที่ค่าเป็น 84 V/V (38.5 dB) และมีค่าเป็น 42 V/V (32.5 dB) ที่กระแสลูปมีค่ามาก ๆ

สัญญาณที่ V_{cc} กลับเฟสกันกับสัญญาณที่ TXI แต่มีเฟสเดียวกันกับสัญญาณที่ MC1



รูปที่ 3.5 แสดงเส้นทางของสัญญาณทั้งทางด้านรับและด้านส่ง

3.7.5 วงจรในการรับสัญญาณ

วงจรที่ใช้รับสัญญาณเข้ามามีอุปกรณ์ดังแสดงในรูปที่ 3.5 ซึ่งโดยปกติมีค่า 600 โอห์ม จะเป็นตัวกำหนดจุดสิ้นสุดของสายส่ง(return loss) ของสัญญาณที่ส่งมาจากขั้วทิปและริง สัญญาณที่ได้รับจะสร้างกระแสไฟสลับผ่าน ZB Network (Balanced Impedance Network) และความต้านทาน 500 โอห์ม ที่ขา ZB A_4 จะลดกระแสลงครึ่งหนึ่งแล้วส่งต่อไปให้ AGC แล้วผ่าน C_8 ไปยัง RXI (จุดรวมอัตราการขยาย ซึ่งถ้า C_8 มีค่ามาก RXA จะเปรียบเป็นกราวน์เสมือนและไม่มีกระแสสลับไหลผ่านความต้านทานภายใน 10 กิโลโอห์ม) แรงดันที่ RXO_1 ถูกกำหนดโดยกระแสจาก C_8 และความต้านทานป้อนกลับ R_8 ออปแอมป์ตัวที่สอง (ที่ขา RXO_2) มีการกำหนดไว้แล้วว่าให้มีการขยายแบบกลับขั้ว และมีการขยายเป็น 1 (Inverting Unility Gain) อัตราการขยายแรงดันจากขั้วทิปกับริงไปยัง $RXO_1 - RXO_2$ มีค่าตามสมการต่อไปนี้

$$G_{RX} = (R_8 * AGC) / (ZB + 500) \qquad (3.2)$$

เมื่อ $ZB = R_{10} + R_9 // C_9 = R_{10} + R_9$

เมื่อใช้ค่าของอุปกรณ์ตามรูป อัตราการขยายจะมีค่าประมาณ 0.495 V/V (-6.1 dB) เมื่อกระแสในลูปมีค่าน้อย และอัตราการขยายกลายเป็น 0.25 V/V (-12 dB) เมื่อมีกระแสในลูปสูง

เมื่อ MC34114 อยู่ในระหว่างการส่งสัญญาณเลขออก (MUTE มีค่าเป็น 0) อัตราการขยายของวงจรภาครับจะลดลงด้วย เพราะมีการต่อ R_{FINT} ที่มีค่า 1.0 กิโลโอห์มขนานกับ R_8 อัตราการลดลงของสัญญาณจะมีค่าดังสมการต่อไปนี้

$$G_{RXM} = 20 * Log ((R_8 + R_{FINT}) / R_{FINT}) \qquad (3.3)$$

เมื่อขา MUTE กลับไปสู่สภาวะ 1 อีกครั้ง จะมีการหน่วงเวลาประมาณ 11 mSec ก่อนที่ความต้านทานจะถูกทำให้กลับไปเป็นสภาวะเดิม เพราะเหตุที่ว่า จะได้ป้องกันสัญญาณทรานเซียนส์อันเนื่องมาจากการส่งสัญญาณแบบพัลส์ อันเป็นเหตุให้เกิดเสียงคลิ๊กขึ้นที่หูฟัง

แรงดันไบอัสที่ขา RXI , RXO_1 แ ละ RXO_2 มีค่าประมาณ 0.65 โวลท์ กระแสไบอัสที่ขา RXI มีค่าประมาณ 50 นาโนแอมป์ แรงดันสูงสุดที่ RXO_1 และ RXO_2 อยู่ในเทอมของความต้านทานของหูฟัง และกระแส I_{tx} โดยค่านี้หาได้จากสมการ $I_{tx} = (V_R * 50 * AGC) / (R_{12} + 100)$

3.7.6 วงจรขจัดไซด์โทน

การขจัดไซด์โทนสามารถทำได้โดยการนำเอาตัวขยายกระแส A_3 มาสร้างสัญญาณที่คล้ายคลึงกันสัญญาณทางด้านส่ง แล้วนำมาขจัดไซด์โทนที่ผ่านเข้ามาทาง ZB และ A_4 เพื่อที่จะได้การขจัดสัญญาณที่สมบูรณ์ (ไม่มีกระแสสลับออกมาทาง RXA) จำเป็นที่จะต้องให้ ZB มีค่าตามสมการ

$$ZB = (40 * R_1 // Z_{LINE}) - 500 \qquad (3.4)$$

ซึ่ง ZB เป็นวงจรที่ประกอบขึ้นด้วย R_9 , R_{10} และ C_9 และ ZB เป็นความต้านทานทางACของสายส่ง อุปกรณ์ที่มีปฏิกิริยาตอบสนองต่อความต้านทานของสายส่ง สามารถชดเชยได้ด้วยการใช้วงจร ZB ที่มีปฏิกิริยาตอบสนองอย่างเปรียบเทียบกันได้ ในรูปที่ 3.5 C_9 จะเป็นตัวชดเชยการเลื่อนของเฟสอันเนื่องมาจากสายส่ง

เนื่องจากตามปกติในสายส่งสัญญาณที่เชื่อมต่ออยู่ระหว่าง ชุมสาย กับ เครื่องโทรศัพท์ จะมีค่าความต้านทาน ตัวเก็บประจุ และขดลวดเหนี่ยวนำอยู่ โดยเฉลี่ยแล้ว ทุกๆ ระยะ 1 ไมล์ ที่เพิ่มขึ้นของสายส่งจะเสมือนว่ามีตัวเก็บประจุค่าประมาณ 0.07 uF ต่อคร่อมอยู่ระหว่างสายส่ง และมีตัวต้านทาน 42 โอห์ม กับขดลวดเหนี่ยวนำ 1 mH ต่ออนุกรมกันอยู่ ดังนั้น จึงต้องมีวงจรที่จะสามารถรับรู้ความผิดพลาดเหล่านี้ได้

3.7.7 การเชื่อมต่อกันของสัญญาณลอจิก

ขาอินพุตลอจิก 2 ขา ของ MC34114 ถูกใช้ในการเปลี่ยนแปลงโหมดการทำงานดังตาราง ที่ 3.2 ต่อไปนี้

MUTE	MS	Mode
High	X	Speech
Low	High	Pulse Dialing
Low	Low	Tone Dialing

ตารางที่ 3.2 แสดงความสัมพันธ์ระหว่างโหมดการทำงานกับลอจิกของ MUTE และ MS

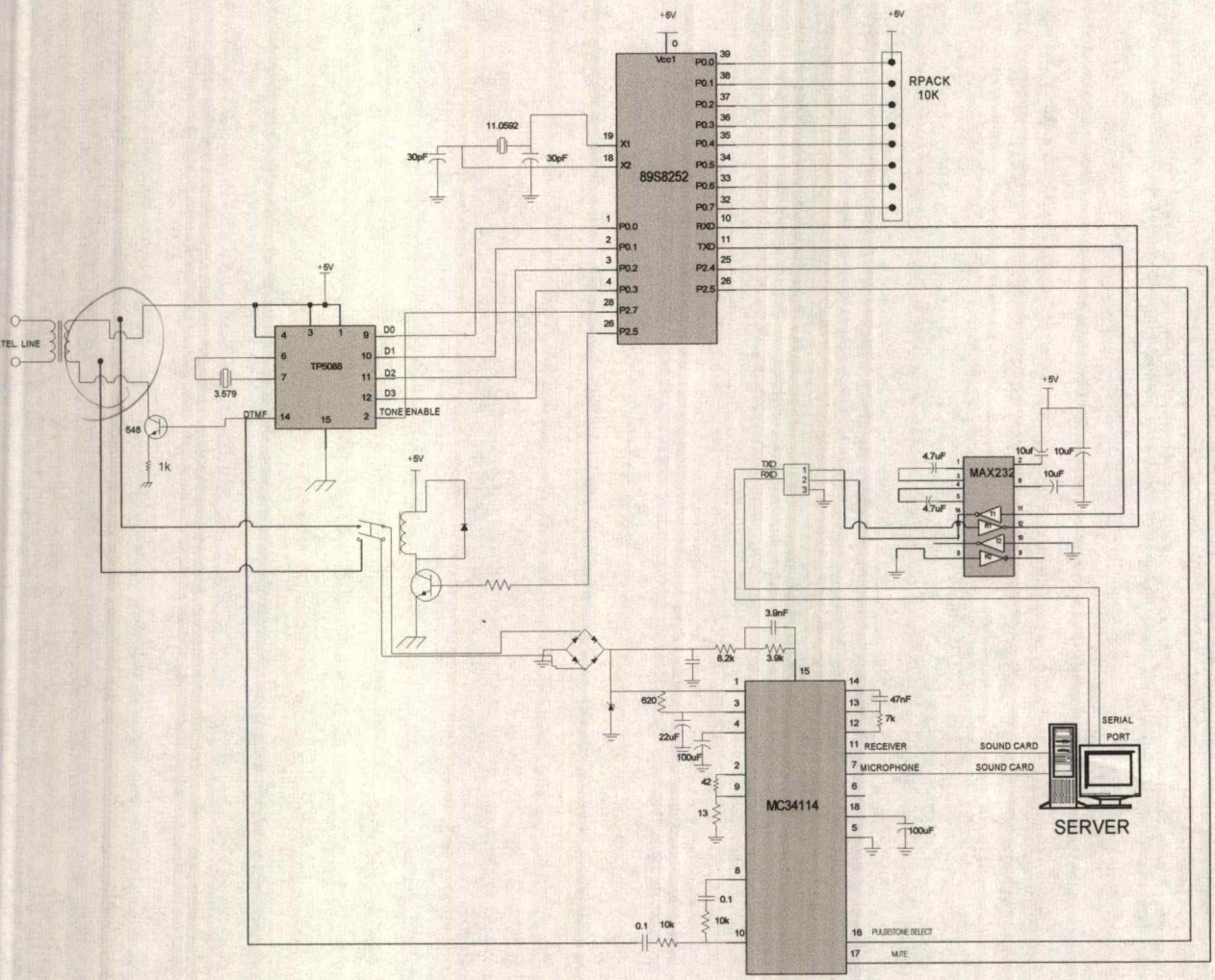
ค่าของลอจิก 1 ของขา MUTE มีค่าระหว่าง V_{DD} – 0.5 จนถึง V_{DD} ส่วนค่าลอจิก 0 ของขา MUTE มีค่าระหว่าง 0 – 1 โวลท์ การเปลี่ยนแปลงลอจิกต้องมากกว่าค่าเทรสโฮล 2.3 โวลท์ เมื่อขา MUTE เปลี่ยนไปเป็นค่า 0 หรือขา MS เกิดการเปลี่ยนแปลงลอจิก การเปลี่ยนแปลงภายในวงจรจะเกิดขึ้นภายใน 10 us แต่ถ้าขา MUTE เปลี่ยนไปเป็นค่า 1 จะเปลี่ยนแปลงหลังจากมีการหน่วง 11 ms เนื่องมาจากมีการป้องกันสัญญาณทรานเซียนส์ที่เกิดขึ้นจากสัญญาณพัลส์อันจะทำให้ได้ยินเสียงคลิ๊กที่หูฟัง

ขา MS จะทำงานเมื่อขา MUTE มีลอจิก 0 หน้าที่ที่แท้จริงของ MS ก็คือ การให้ค่าแรงดันเลเวลชิฟท์แก่ V_{cc} และ LR ในการส่งสัญญาณแบบโทน ค่าลอจิก 0 มีค่าระหว่าง 0 – 0.3 โวลท์ ค่าลอจิก 1 มีค่าระหว่าง 2 - V_{DD} โวลท์ ค่าเทรสโฮลมีค่า 0.75 โวลท์ เมื่อไม่มีการเลือกการทำงานระหว่างการส่งแบบพัลส์หรือโทน ให้ต่อลงกราวน์หรือ V_{DD} ห้ามไม่ให้ปล่อยลอยไว้เป็นอันขาด

เมื่ออยู่ในสภาวะออนฮุก และมีแรงดันไม่เกิน 6 โวลท์ต่ออยู่กับ MUTE กระแสรั่วไหล 0.02 ไมโครแอมป์ จะไหลเข้าขา MUTE และ V_{DD} แรงกันเท่ากัน แต่ถ้าแรงดันมีค่าไม่เท่ากันแล้ว กระแสจะไหลผ่านตัวต้านทานภายในและไดโอด หากมีแหล่งจ่ายไฟ เพื่อคงหน่วยความจำของวงจรที่แป้นกดต่ออยู่ และปรากฏว่ามีแรงดันของแหล่งจ่าย เพื่อคงหน่วยความจำของวงจรแป้นกดที่ V_{DD} ขา MUTE จะต้องต่ออยู่กับ V_{DD} หรือกราวน์ มิฉะนั้นแล้ว กระแส 100 – 200 ไมโครแอมป์ จะไหลผ่าน V_{DD} และออกทางขา MUTE

เมื่อ V_{cc} มีค่าเท่ากับ 0 และมีแรงดันไม่เกิน 6 โวลท์ ต่ออยู่ที่ขา MS จะมีกระแสรั่วไหลเกิดขึ้น 0.01 ไมโครแอมป์ ตลอดเวลาที่ขา MUTE ปล่อยลอยหรือต่อกับ V_{DD} หากขา MUTE ต่อลงกราวน์ จะมีความต้านทาน 3.5 กิโลโอห์ม เกิดขึ้นระหว่าง MS และ MUTE

หาก V_{cc} < 1.5 โวลท์ ขา MUTEจะไม่ทำงาน เป็นเหตุให้ MC34114 อยู่ในการทำงานโหมดสนทนา

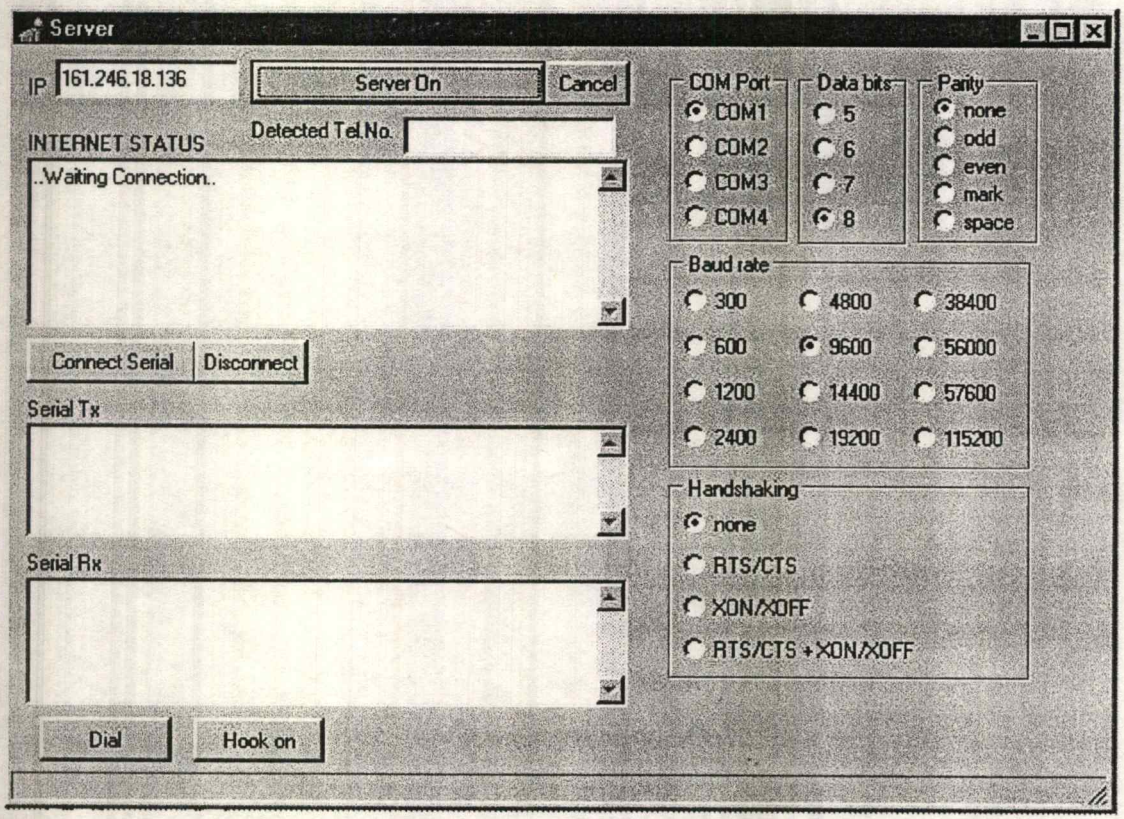


รูปที่ 3.6 แสดงวงจรรวม

บทที่ 4
การทดลองและผลการทดลอง

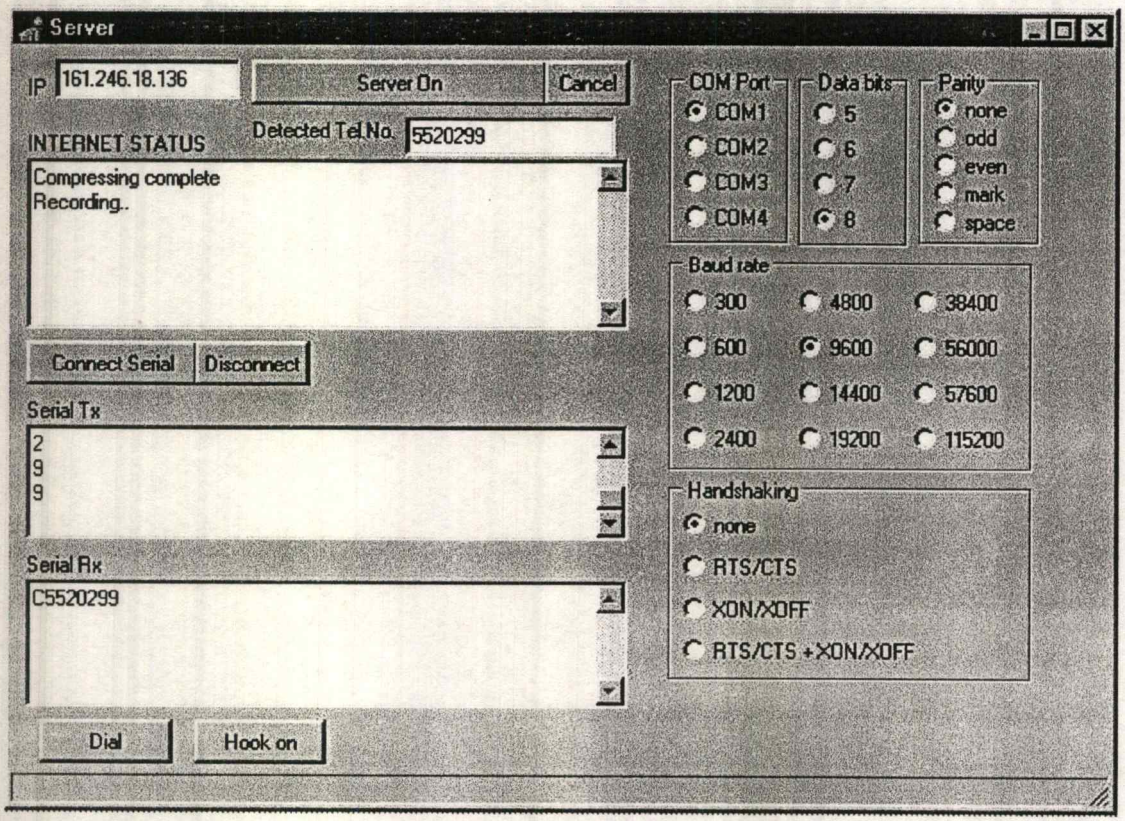
4.1 ผลการทดลองโปรแกรมเซิร์ฟเวอร์

โปรแกรมเซิร์ฟเวอร์จะถูกคอมไพล์เป็นไฟล์ชื่อ Server_2.exe ซึ่งเมื่อจะเริ่มต้นให้เซิร์ฟเวอร์ทำงานต้องตั้งค่าต่างๆ เกี่ยวกับการส่งข้อมูลทางพอร์ตอนุกรมให้ตรงกับค่าที่ไมโครโปรเซสเซอร์ใช้ก่อน แล้วจึงกดปุ่ม connect serial ถ้าปัญหาจะแสดงผลทางขอบล่างของโปรแกรมว่า Error จากนั้นก็กดปุ่ม Server on เพื่อสั่งให้เริ่มรอรับการติดต่อจากไคล์เอนต์ โดยเซิร์ฟเวอร์จะอยู่ในสถานะพร้อม ซึ่งจะได้ผลดังรูป



รูปที่ 4.1 แสดงโปรแกรมเซิร์ฟเวอร์ที่พร้อมรอรับการติดต่อจากไคล์เอนต์

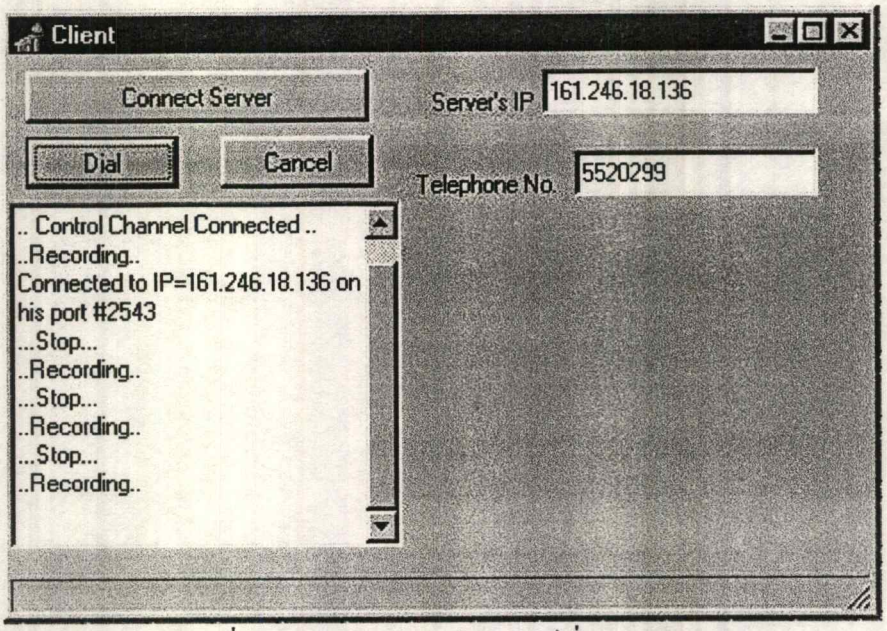
ถ้าหากมีโปรแกรมไคล์เอนต์เรียกเข้ามาโปรแกรมเซิร์ฟเวอร์ก็จะรับเบอร์โทรศัพท์ที่ไคล์เอนต์ต้องการติดต่อแล้วหมุนโทรศัพท์เรียกไปยังเครื่องโทรศัพท์ปลายทาง ถ้าหากมีผู้รับก็จะสามารถสนทนากันได้โดยเมื่อมีเสียงพูดจากผู้ใช้โทรศัพท์ โปรแกรมเซิร์ฟเวอร์จะตรวจจับและบีบอัดข้อมูล ซึ่งจะแสดงสถานะการทำงานให้เห็นดังรูป 4.2



รูปที่ 4.2 แสดงโปรแกรมเซิร์ฟเวอร์ขณะรับส่งข้อมูลเสียง

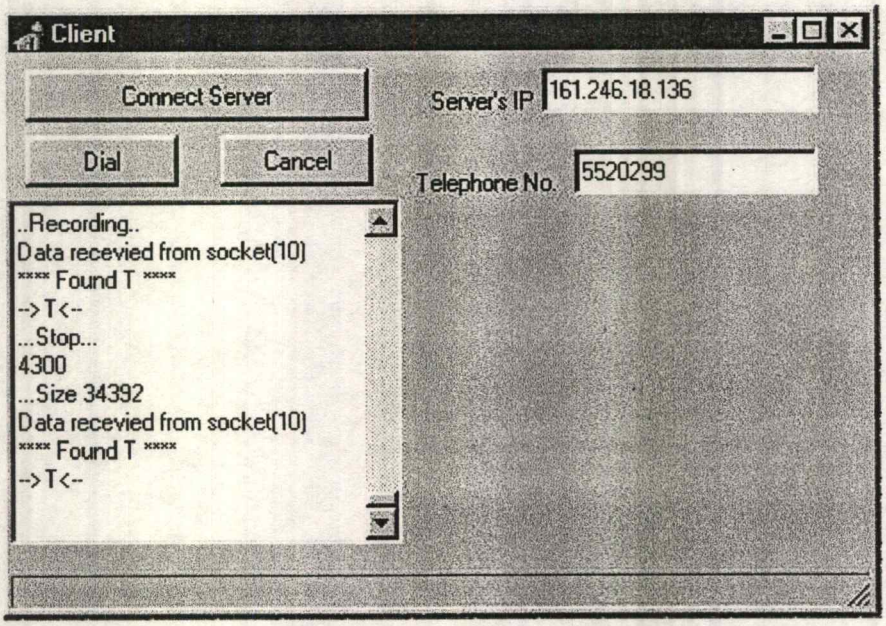
4.2 ผลการทดลองโปรแกรมไคล์เอนด์

โปรแกรมไคล์เอนด์จะถูกคอมไพล์เป็นไฟล์ชื่อ Client_2.exe ซึ่งเมื่อจะเริ่มต้นให้ไคล์เอนด์ทำงานต้องป้อนค่าหมายเลขโทรศัพท์แล้วกดปุ่ม Connect Server และ จะแสดงผลดังรูป



รูปที่ 4.3 แสดงโปรแกรมไคลเอนต์เมื่อสนทนา

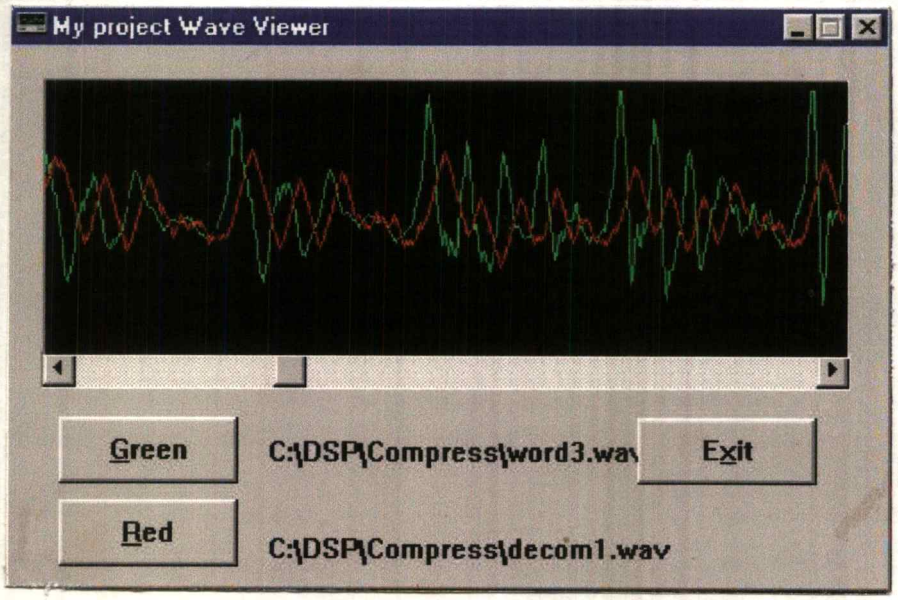
โดยเมื่อได้รับข้อมูลเสียงจะแสดงข้อความให้ ทราบว่ามีข้อมูลเสียงเข้ามาและ แสดงให้เห็นขนาด ไฟล์ที่รับเข้ามาและขนาดไฟล์หลังการขยายข้อมูลออก ดังรูป



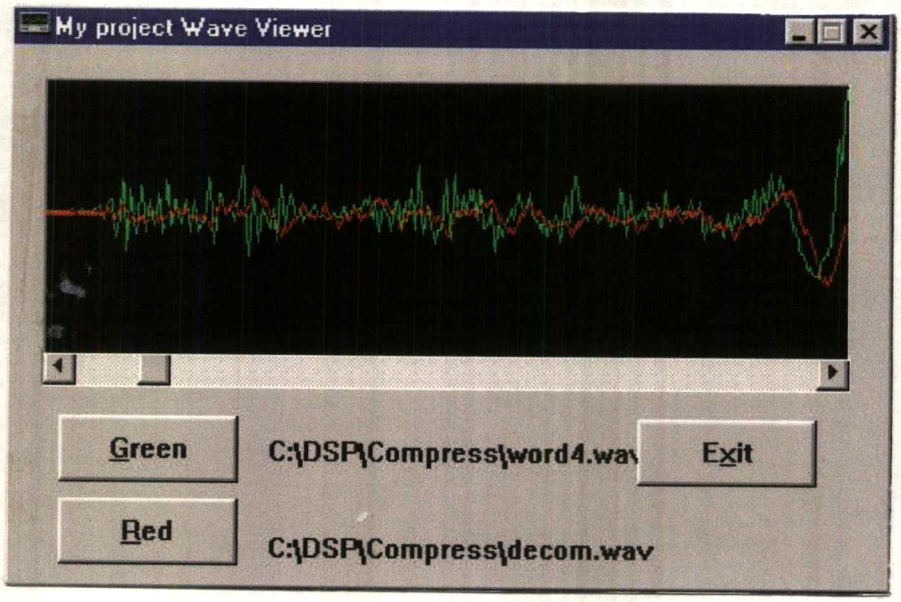
รูปที่ 4.4 แสดงโปรแกรมไคลเอนต์เมื่อรับข้อมูลเสียงเข้ามา

4.3 ผลการทดลองการบีบอัดข้อมูลด้วยวิธีอะแด็ปทีฟเดลต้ามอดดูเลชั่น

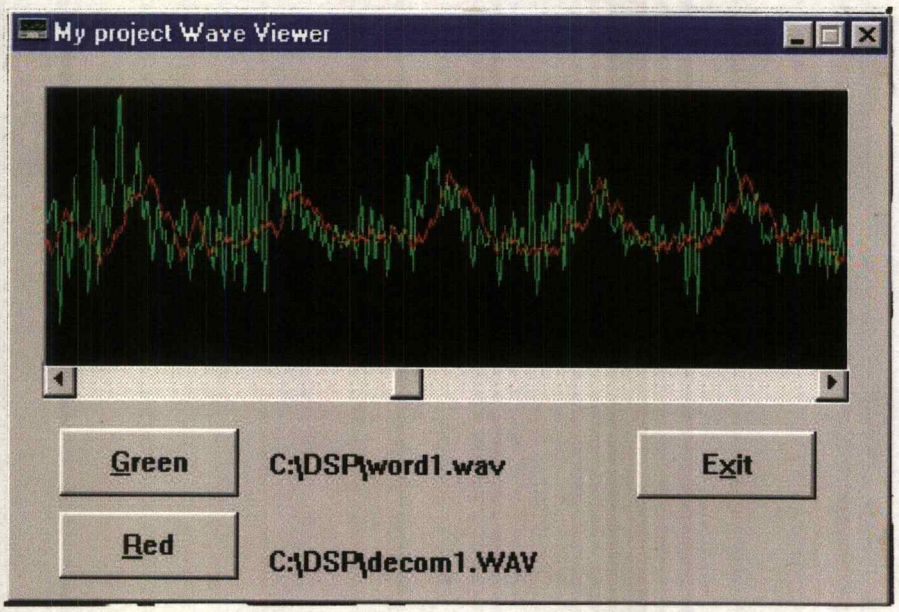
จากการเก็บตัวอย่างเสียงจากผ่านการบีบอัดมาเปรียบเทียบกับเสียงยังไม่ผ่านการบีบอัดได้ผลมาดูสัญญาณในแกนเวลาดังนี้



รูปที่ 4.5 แสดงการเปรียบเทียบสัญญาณเสียงก่อนการบีบอัด(กราฟสีเขียว)และหลังจากผ่านการบีบอัดข้อมูล(กราฟสีแดง) ตัวอย่างเสียงพูดที่ 1

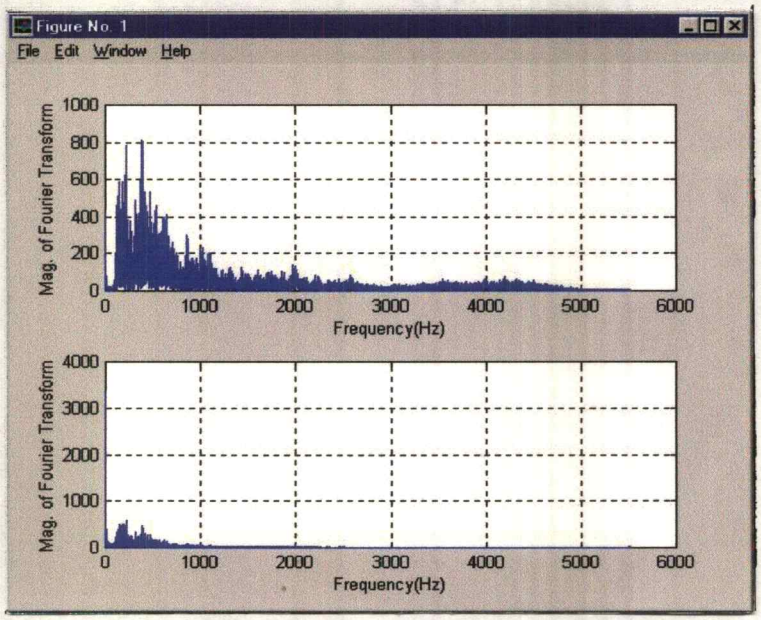


รูปที่ 4.6 แสดงการเปรียบเทียบสัญญาณเสียงก่อนการบีบอัด(กราฟสีเขียว)และหลังผ่านการบีบอัดข้อมูล(กราฟสีแดง) ตัวอย่างเสียงพูดที่ 2

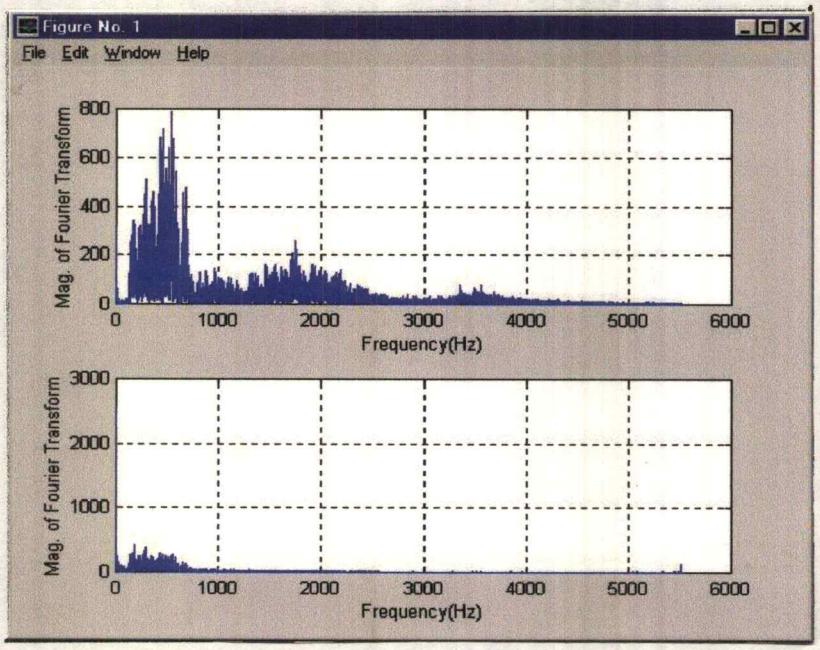


รูปที่ 4.7 แสดงการเปรียบเทียบสัญญาณเสียงก่อนการบีบอัด(กราฟสีเขียว)และหลังผ่านการบีบอัดข้อมูล (กราฟสีแดง)ตัวอย่างเสียงพูดที่ 3

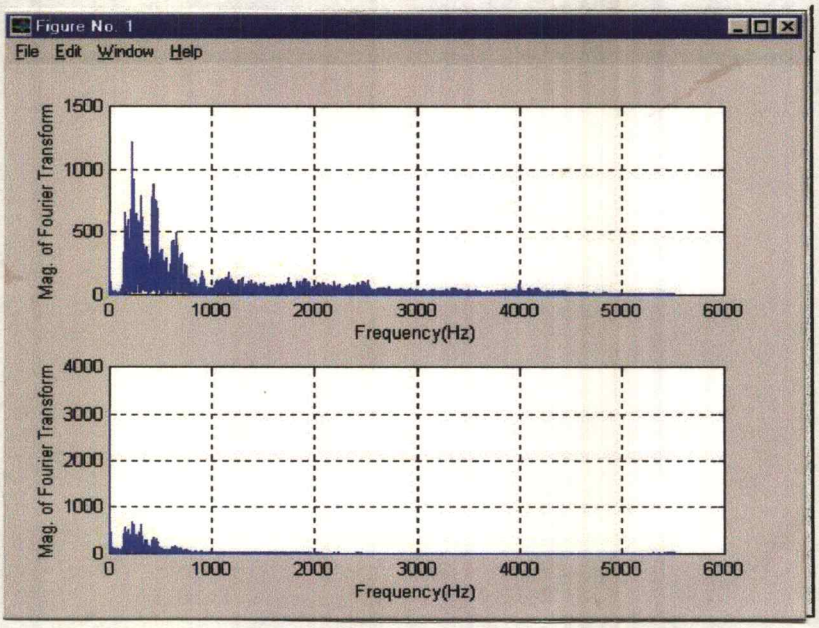
นำตัวอย่างเสียงทั้ง 3 มาผ่านการแปลงโดยใช้ฟังก์ชั่น ฟาส ฟูเรียห์ ทรานส์ฟอร์ม (Fast Fourier Transform) ของ MATLAB ได้ผลดังนี้



รูปที่ 4.8 สเปกตรัมสัญญาณเสียงก่อนบีบอัดข้อมูล(กราฟรูปบน) และหลังผ่านการบีบอัดข้อมูล (กราฟรูปล่าง) ของตัวอย่างเสียงพูดที่ 1



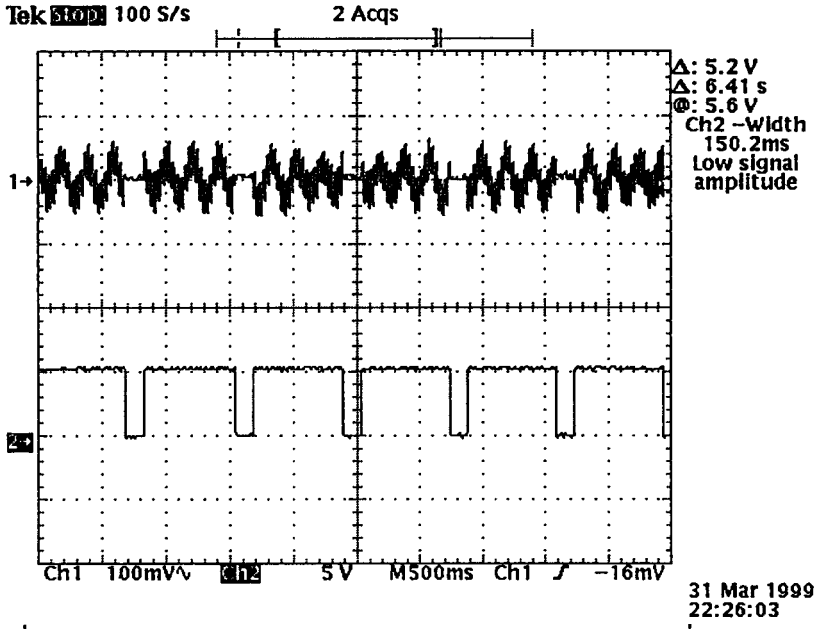
รูปที่ 4.9 สเปกตรัมสัญญาณเสียงก่อนบีบอัดข้อมูล(กราฟรูปบน) และหลังผ่านการบีบอัดข้อมูล (กราฟรูปล่าง) ของตัวอย่างเสียงพูดที่ 2



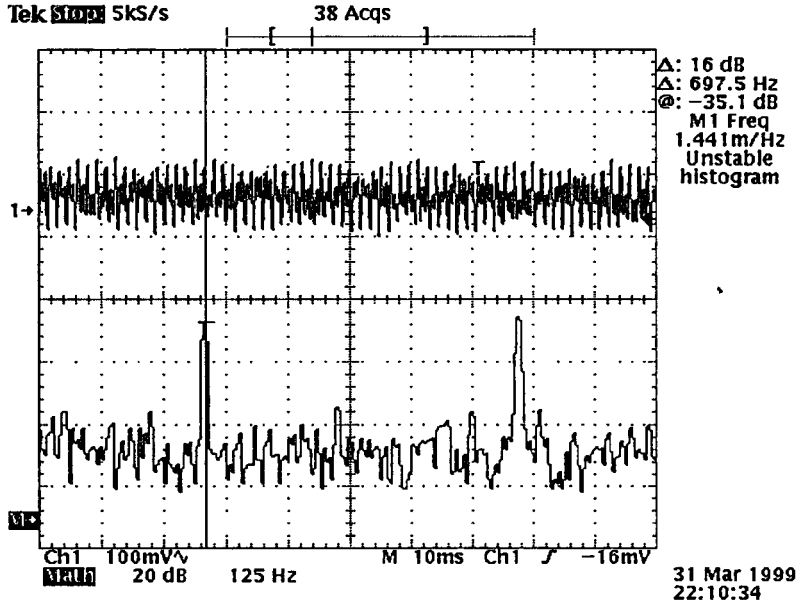
รูปที่ 4.10 สเปกตรัมสัญญาณเสียงก่อนบีบอัดข้อมูล(กราฟรูปบน) และหลังผ่านการบีบอัดข้อมูล (กราฟรูปล่าง) ของตัวอย่างเสียงพูดที่ 3

4.4 การทดสอบส่วนกำเนิดสัญญาณ DTMF

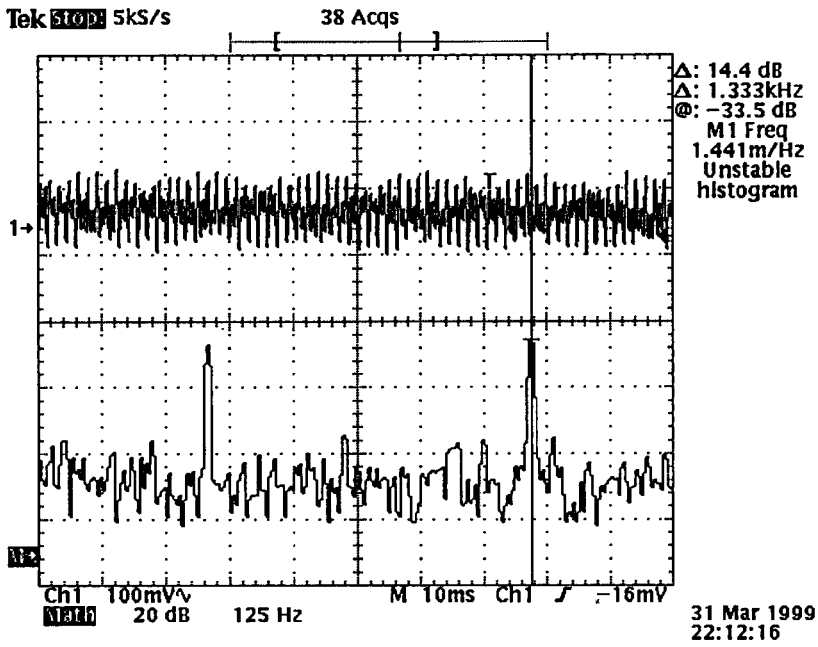
จากหลักการทำงานของส่วนกำเนิดสัญญาณ DTMF ไอซีเบอร์ TP5088 จะทำการผลิตสัญญาณ DTMF ออกมาเมื่อไมโครโปรเซสเซอร์ ส่งหมายเลขค่าหมายเลขจากพอร์ต P1.0 –P1.3 และส่งค่า 1 ออกทางขาP2.7 ซึ่งทำหน้าที่เป็น TONE ENABLE เข้าไปยังไอซี TP5088 จะได้ผลดังรูปที่ 4.10



รูปที่ 4.11 แสดงการสร้างสัญญาณ DTMF (สัญญาณ 1 หมายเลข) ที่ควบคุมด้วยสัญญาณ TONE ENABLE (สัญญาณหมายเลข 2)



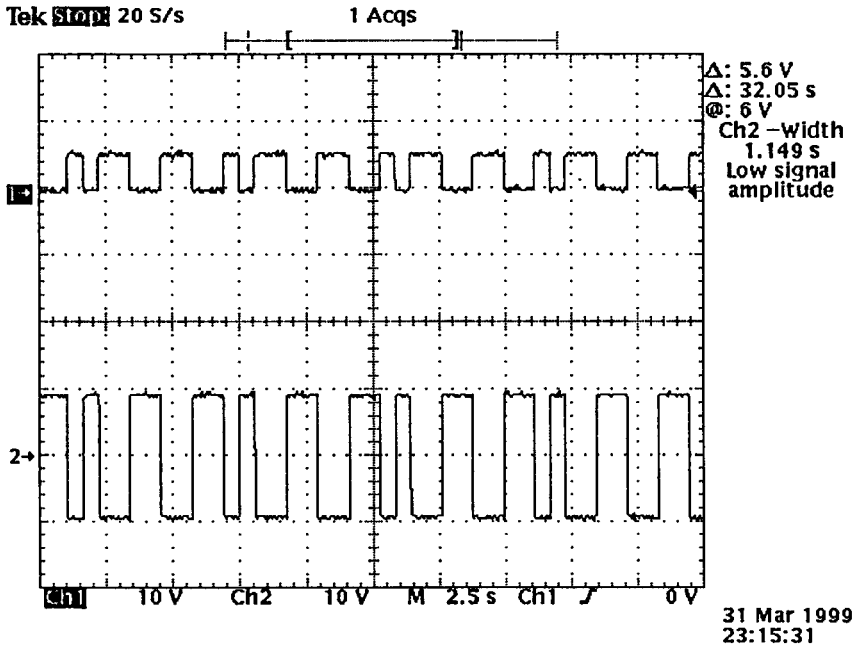
รูปที่ 4.12 แสดงสัญญาณ DTMF ของเลข 2 ประกอบด้วยความถี่ 697.5 Hz



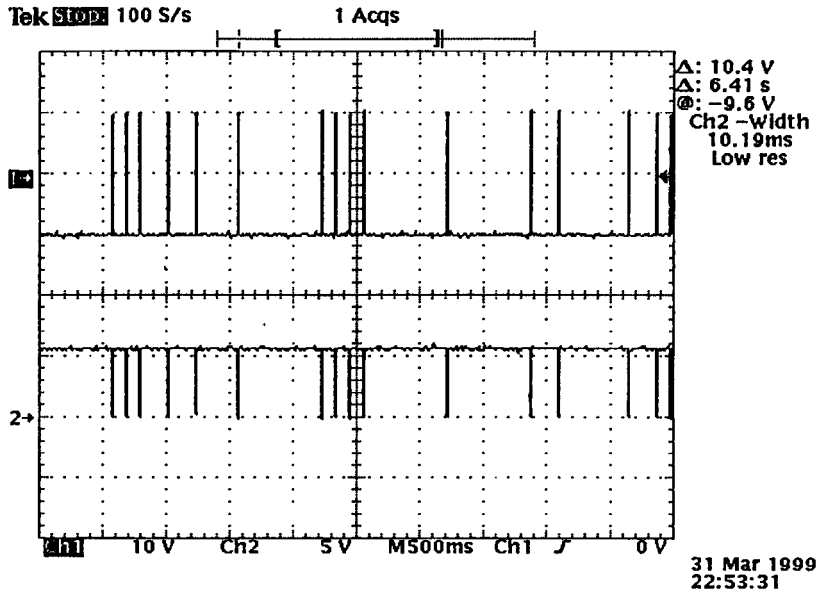
รูปที่ 4.13 แสดงสัญญาณ DTMF ของเลข 2 ประกอบด้วยความถี่ 1.333 KHz

4.5 การทดสอบส่วนแปลงสัญญาณที่รับส่งทางพอร์ตอนุกรม

ในการส่งสัญญาณข้อมูลจากพอร์ตอนุกรมของเซิร์ฟเวอร์ไปยังไมโครโปรเซสเซอร์นั้นจะ ต้องแปลงระดับสัญญาณ โดยใช้ไอซี MAX232 การส่งสัญญาณจากเซิร์ฟเวอร์(-10 โวลต์ ถึง +10 โวลต์) ไปยังไมโครโปรเซสเซอร์ต้องส่งสัญญาณระดับ –10 โวลต์ ถึง +10 โวลต์ เข้าขา 13 ของ MAX232 แล้วจะได้สัญญาณที่มีระดับ 0 ถึง 5โวลต์ ออกทางขา 12 เพื่อส่งเข้าขา Rx ของไมโครโปรเซสเซอร์ ดังรูป ที่ 4.13



รูปที่ 4.14 แสดงการแปลงระดับสัญญาณที่จากเซิร์ฟเวอร์ (สัญญาณหมายเลข 2) เข้าขา13 ของ MAX232 แล้วให้ผลของสัญญาณ เป็นระดับ 0 ถึง 5 โวลต์ทางขา 12 (สัญญาณหมายเลข 1)



รูปที่ 4.15 แสดงการแปลงระดับสัญญาณที่จากไมโครโปรเซสเซอร์(สัญญาณหมายเลข 2) เข้าขา 11 ของ MAX232 แล้ว ให้ผล ของสัญญาณ เป็นระดับ 0 ถึง 5 โวลต์ทางขา 14(สัญญาณหมายเลข 1)

ในทางกลับกัน การส่งสัญญาณข้อมูลจากไมโครโปรเซสเซอร์ไปยังพอร์ตอนุกรมของเซิร์ฟเวอร์นั้น จะ ต้องแปลงระดับสัญญาณ โดยสัญญาณจากไมโครโปรเซสเซอร์ จะส่งเข้าที่ขา 11 ของไอซี MAX232 แล้วจะได้สัญญาณที่มีระดับ –10 โวลต์ ถึง +10 โวลต์ออกทางขา 14 เพื่อส่งเข้าพอร์ตอนุกรมของเซิร์ฟเวอร์ต่อไป

บทที่ 5
บทสรุปและวิจารณ์

จากผลการทดลองปรากฏว่าส่วนต่างๆของโครงงานทำงานได้อย่างเป็นที่น่าพอใจ ผู้ใช้สามารถใช้สนทนากันได้เป็นอย่างดี แต่ยังมีข้อด้อยในด้านประสิทธิภาพทางด้านการสนทนา อาทิเช่น ขณะทำการสนทนาจะมีการหน่วงเวลาเกิดขึ้นเนื่องจากทั้งการทำงานของโปรแกรมเองและยังอาจมีการหน่วงของเวลาเพิ่มขึ้นหากว่าในขณะที่สนทนานั้นเครือข่ายมีการจราจรหนาแน่น จึงทำให้เป็นอุปสรรคในการสนทนาทำให้เกิดความเบื่อหน่ายหรือทำให้การรับข่าวสารผิดพลาดได้

สำหรับในโครงงานนี้ได้จัดทำไว้เพื่อรองรับผู้ใช้เพียง 1 รายเท่านั้น เมื่อมีผู้ต้องการใช้มากกว่า 1 รายขึ้นไปจะต้องรอจนกว่าผู้ใช้รายแรกจะยกเลิกการติดต่อเสียก่อน

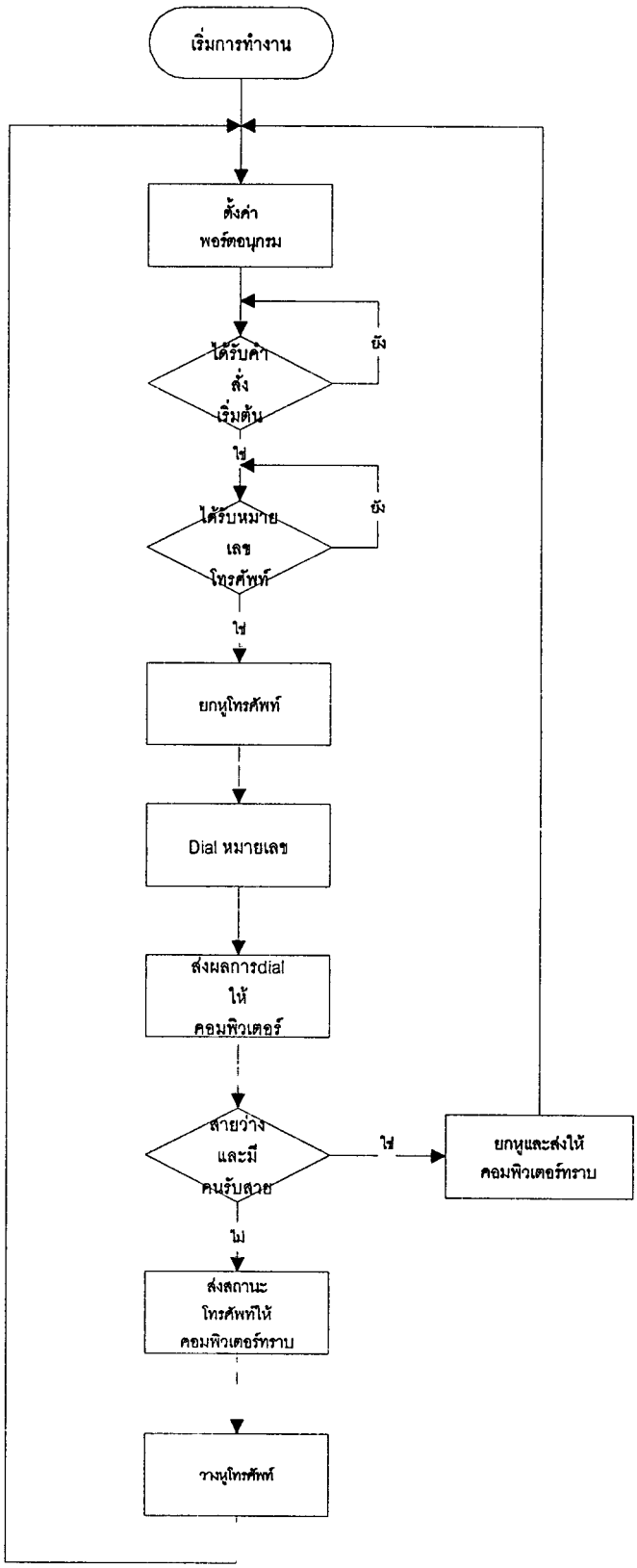
ในการพัฒนาต่อไปจะต้องทำการลดสัญญาณรบกวนและความล่าช้าที่เกิดจากการทำงานของโปรแกรม

ภาคผนวก

กิตติกรรมประกาศ

ขอขอบพระคุณ อาจารย์ นภัทร สระเอี่ยม ที่ให้ความช่วยเหลือในด้านต่างๆเสมอมา
ขอขอบพระคุณ ร.ศ.สมยศ จุณณะปิยะ ที่กรุณาให้ข้อมูลเกี่ยวกับไมโครโปรเซสเซอร์
ขอบคุณ ปิยะบุตร บุญอร่ามเรือง และปานวิทย์ ฐุวะนุติ ที่ให้ยืมใช้อุปกรณ์ด้วยความเต็มใจ

โฟล์ชาร์ตแสดงการทำงานของไมโครโปรเซสเซอร์



โปรแกรมควบคุมไมโครโปรเซสเซอร์

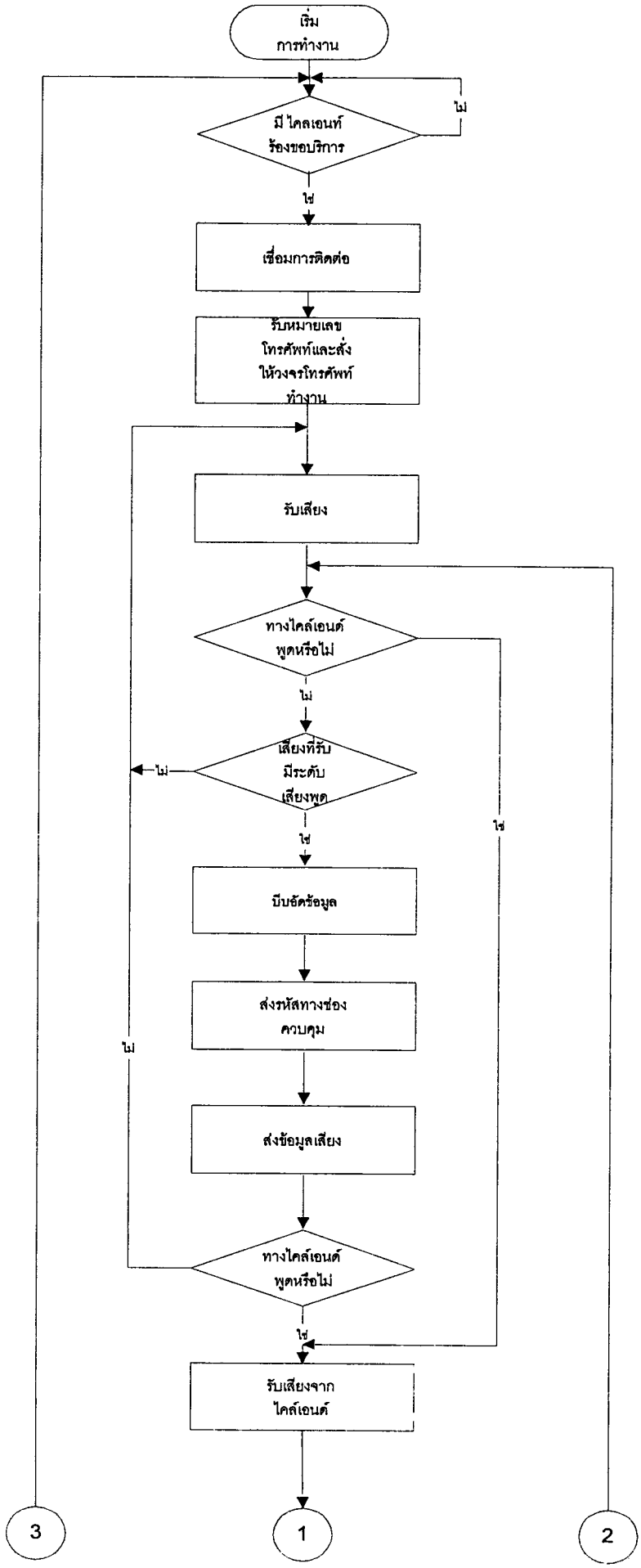
```
        ORG 0000H

        MOV PCON,#00H    ;INITIAL SERIAL PORT
        MOV SCON,#52H
        MOV TMOD,#20H
        MOV TH1,#0FDH
        SETB TR1
        CLR RI
        CLR P2.4         ;LOW MUTE
        CLR P2.5         ;LOW TRIG
        CLR P2.6         ;CLEAR RELAY
        CLR P2.7         ;CLEAR TONE ENABLE

MAIN:   CALL RBYTE       ;CALL WAIT  DATA FROM SERIAL
        CALL SBYTE       ;CALL SEND DATA  BACK TO COMPUTER

        SJMP MAIN

RBYTE:  JNB RI,$         ;WAIT DATA PROCEDURE
        CLR RI
        MOV A,SBUF       :SHIFT DATA INTO A
        CALL DIAL        ;DIAL NUMBER IN A
        RET
SBYTE:  JNB TI,$         ;SEND DATA PROCEDURE
        CLR TI
        MOV SBUF,A       ;SHIFT DATA FROM A
        RET

DIAL:   CJNE A,#'1',$+6  ;IF 1 DIAL
        CALL DIAL1
        CJNE A,#'2',$+6  ;IF 2 DIAL
        CALL DIAL2
        CJNE A,#'3',$+6  ;IF 3 DIAL
        CALL DIAL3
        CJNE A,#'4',$+6  ;IF 4 DIAL
        CALL DIAL4
        CJNE A,#'5',$+6  ;IF 5 DIAL
        CALL DIAL5
        CJNE A,#'6',$+6  ;IF 6 DIAL
        CALL DIAL6
        CJNE A,#'7',$+6  ;IF 7 DIAL
        CALL DIAL7
        CJNE A,#'8',$+6  ;IF 8 DIAL
```

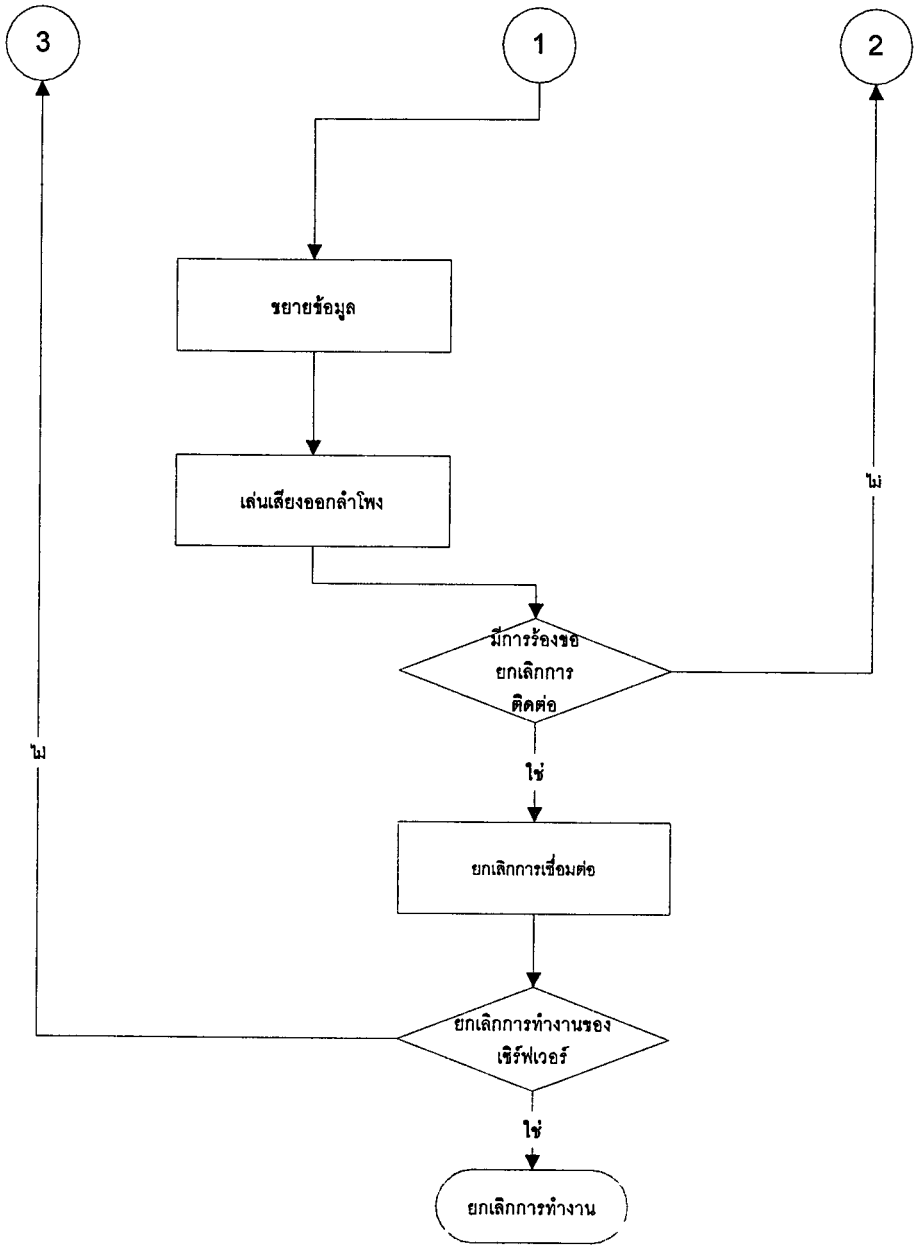
```
        CALL DIAL8
        CJNE A,#'9',$+6   ;IF 9 DIAL
        CALL DIAL9
        CJNE A,#'0',$+6   ;IF 0 DIAL
        CALL DIAL0

        CJNE A,#'C',$+6   ;IF C TURN ON
        CALL RELAYON

        CJNE A,#'D',$+6   ;IF D TURN OFF
        CALL RELAYOFF

      RET     ;RETURN  TO MAIN

    RELAYON:SETB P2.6  ;SET RELAY ON
          CLR P2.5
          SETB P2.5
          RET

    RELAYOFF:CLR P2.6   ;SET RELAY OFF
           CLR P2.5
           SETB P2.5
           RET

  TONEHOLDING:MOV R7,#05H     ;TONE HOLDING PERIOD LOOP
       D3:MOV R6,#0FFH
       D2:MOV R5,#0FFH
       D1:DJNZ R5,D1
          DJNZ R6,D2
          DJNZ R7,D3
          RET

     DELAY:MOV R2,#0FFH  ;DELAY NEXT NUMBER
     DELAY2:MOV R3,#0FFH
     DELAY1:DJNZ R3,DELAY1
          DJNZ R2,DELAY2
          RET


    DIAL0:  MOV P1,#0AH
          SETB P2.7       ;SET TONE ENABLE
          CALL TONEHOLDING
          CLR P2.7        ;CLEAR TONE ENABLE
          CALL DELAY
          RET
```

```
DIAL1:  MOV P1,#01H
        SETB P2.7       ;SET TONE ENABLE
        CALL TONEHOLDING
        CLR P2.7        ;CLEAR TONE ENABLE
        CALL DELAY
        RET

DIAL2:  MOV P1,#02H
        SETB P2.7       ;SET TONE ENABLE
        CALL TONEHOLDING
        CLR P2.7        ;CLEAR TONE ENABLE
        CALL DELAY
        RET

DIAL3:  MOV P1,#03H
        SETB P2.7       ;SET TONE ENABLE
        CALL TONEHOLDING
        CLR P2.7        ;CLEAR TONE ENABLE
        CALL DELAY
        RET

DIAL4:  MOV P1,#04H
        SETB P2.7       ;SET TONE ENABLE
        CALL TONEHOLDING
        CLR P2.7        ;CLEAR TONE ENABLE
        CALL DELAY
        RET

DIAL5:  MOV P1,#05H
        SETB P2.7       ;SET TONE ENABLE
        CALL TONEHOLDING
        CLR P2.7        ;CLEAR TONE ENABLE
        CALL DELAY
        RET

DIAL6:  MOV P1,#06H
        SETB P2.7       ;SET TONE ENABLE
        CALL TONEHOLDING
        CLR P2.7        ;CLEAR TONE ENABLE
        CALL DELAY
        RET

DIAL7:  MOV P1,#07H
        SETB P2.7       ;SET TONE ENABLE
```

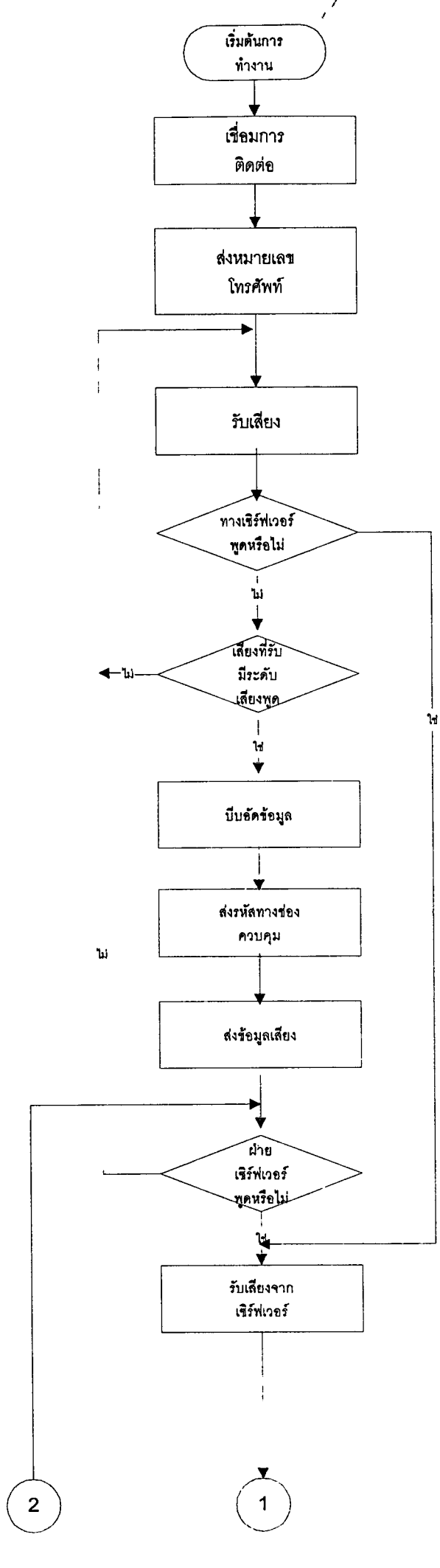
```
        CALL TONEHOLDING
        CLR P2.7        ;CLEAR TONE ENABLE
        CALL DELAY
        RET

DIAL8:  MOV P1,#08H
        SETB P2.7       ;SET TONE ENABLE
        CALL TONEHOLDING
        CLR P2.7        ;CLEAR TONE ENABLE
        CALL DELAY
        RET

DIAL9:  MOV P1,#09H
        SETB P2.7       ;SET TONE ENABLE
        CALL TONEHOLDING
        CLR P2.7        ;CLEAR TONE ENABLE
        CALL DELAY
        RET


    END
```

โฟล์ชาร์ตการทำงานของเซิร์ฟเวอร์




3
1
2
รยายข้อมูล
เล่นเสียงออกลำโพง
มีการร้องขอ
ยกเลิกการ
ติดต่อ
ไม่
ไม่
ใช่
ยกเลิกการเชื่อมต่อ
ยกเลิกการทำงานรอง
เซิร์ฟเวอร์
ใช่
ยกเลิกการทำงาน

โฟล์ชาร์ตการทำงานของไคล์เอนต์



```
unit Server2;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Sockets, Winsock, MMioWrap, MMsystem, WaveWrap, Waveplay, inifiles,
  ExtCtrls, ComCtrls, ComDrv32;

type
  TForm1 = class(TForm)
    Edit1: TEdit;
    Button1: TButton;
    Button2: TButton;
    Sockets1: TSockets;
    Sockets2: TSockets;
    Memo1: TMemo;
    Timer1: TTimer;
    Sockets3: TSockets;
    CommPortDriver: TCommPortDriver;
    ComPortRG: TRadioGroup;
    BaudRateRG: TRadioGroup;
    DataBitsRG: TRadioGroup;
    ParityRG: TRadioGroup;
    HandshakingRG: TRadioGroup;
    TxMemo: TMemo;
    RxMemo: TMemo;
    Button3: TButton;
    Button4: TButton;
    StatusBar1: TStatusBar;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Edit2: TEdit;
    N: TLabel;
    Label4: TLabel;
    Button15: TButton;
    Timer2: TTimer;
    Button5: TButton;
    procedure Button1Click(Sender: TObject);
    procedure GetStream;
    procedure PlayStream;
    function TimedOut:boolean;
    procedure Move(filename : string);
    function Sockets2Error:boolean;
```

```
    procedure closeSocket2;
    procedure FormCreate(Sender: TObject);
    procedure CheckT;
    procedure Timer1Timer(Sender: TObject);
    procedure sendstream;
    procedure WavRecord;
    procedure SendSockets3;
    procedure WaitControl;
    procedure WaitDial;
    procedure Button2Click(Sender: TObject);
    procedure Main;
    procedure DetectVoice;
    procedure Sockets3SessionClosed(Sender: TObject; Socket: Integer);
    procedure Sockets3DataAvailable(Sender: TObject; Socket: Integer);
    procedure TxMemoKeyPress(Sender: TObject; var Key: Char);
    procedure CommPortDriverReceiveData(Sender: TObject; DataPtr: Pointer; DataSize: Integer);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure BaudRateRGClick(Sender: TObject);
    procedure DataBitsRGClick(Sender: TObject);
    procedure ParityRGClick(Sender: TObject);
    procedure HandshakingRGClick(Sender: TObject);
    procedure ComPortRGClick(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure Sockets3SessionAvailable(Sender: TObject; Socket: Integer);
    procedure Sockets3SessionConnected(Sender: TObject; Socket: Integer);
    procedure Timer2Timer(Sender: TObject);
    procedure Plus(i:integer;var bit:byte);
    procedure compress;
    procedure Decompress;
    procedure Button15Click(Sender: TObject);
    procedure Dial(s:string);
    procedure Button5Click(Sender: TObject);

    private
    { Private declarations }
    procedure ApplyCommSettings;
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  T,abort,RealTel : Boolean;
  WaveForm: tAudio_waveform;
```

```
  TelNum : String;
  ReturnString: array[0..255] of Char;
  FirstData,Aborted : boolean;
  OutOfTimer2,VoiceDetected,cancel: boolean;
  OpenedFile,i,ErrorReturn,FirstTempLength,TelNumCount : Integer;
  rms,slope,scale_factor:real ;
implementation

{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
begin
  Edit1.Text := '161.246.18.136';
  Abort := False;
  Sockets1.NonBlocking := false;   {Receive}
  Sockets1.Timeout := 30;
  Sockets1.IpAddr := Edit1.Text;
  Sockets1.Port := '2541';
  Sockets1.SListen;
  Memo1.Lines.Add('..Waiting Connection.. ');
  Sockets1.SAccept;
  Memo1.Lines.Clear;
  Memo1.Lines.Add('.. Voice Channel Connected .. ');
  WaitControl;
end;

procedure TForm1.WaitControl;
begin
  Sockets2.NonBlocking := false;   {Send}
  Sockets2.Timeout := 30;
  Sockets2.IpAddr := Edit1.Text;
  Sockets2.Port := '2542';
  Sockets2.SListen;
  Memo1.Lines.Add('..Waiting Control Channel.. ');
  Sockets2.SAccept;
  Memo1.Lines.Clear;
  Memo1.Lines.Add('.. Control Channel Connected .. ');
  Sockets3.IPAddr := Sockets3.GetLocalIPAddr;
  Sockets3.Port := '2543';
  Sockets3.SListen;
end;
procedure TForm1.SendSockets3;
var  buff : array[0..255] of char;
begin
   Sockets3.Text := 'T';
```

```
    Memo1.Lines.Add('... Send T ... ');
  end;

  procedure TForm1.WaitDial;
  var prev,count,idx2 : integer;
     temp,buf,num : string;
  begin
   Sockets3.NonBlocking := false;
   Sockets3.Timeout := 30;
   Sockets3.IpAddr := Edit1.Text;
   Sockets3.Port := '2543';
   Sockets3.SListen;
   Memo1.Lines.Add('..Waiting Telephone Number.. ');
   Sockets3.SAccept;
    prev:=1;
    count :=0;
    idx2 := 0;
    buf := '';
    buf := Sockets3.peek;
    Memo1.lines.Add(buf);
    idx2 := Pos( ';' , buf);
    if idx2 > 0 then
     begin
       temp := buf;
       count := count+1;
       Num := Copy(temp,prev,1);
       Memo1.Lines.Add('Connect to Telephone Number : '+ Num);
    end;
   Memo1.Lines.Add('.. Got Telephone Number  : ' +  Num);
   WavRecord;
  end;

  procedure TForm1.Main;
  begin
   GetStream;
  end;
  procedure TForm1.Move(filename : string);
  var
   pPrev8bitSample,
   p8bitSample:  ^Byte;
   pPrev16bitSample,
   p16bitSample: ^SmallInt;
   i,s,num_detect:          integer;
   szNewBuffer:  Cardinal;
   NewData:     PChar;
```

```
  GainFactor,
  Period:   integer;
  numbyte,fsize:integer;
  temp : array[0..255] of char;
  temp_file : integer;
begin
  {process the waveform data to add an echo effect}
  with WaveForm do begin

    if Format.wFormatTag = WAVE_FORMAT_PCM then begin
     if Format.wBitsPerSample = 8 then begin
       {working with an 8-bit sample}
       {Re-allocate memory for the new size of the wave data}
       StrpCopy(temp,filename);
       temp_file := _lopen(temp,0);
       fsize := _llseek(temp_file,0,2);
       szNewBuffer := fsize;
       NewData := strAlloc(szNewBuffer);
       _llseek(temp_file,0,0);
       _lread(Temp_file,@NewData[0],fsize);
       StrDispose(Data);
       Data := NewData;
       _lclose(temp_file);
       {initialize the new part of the buffer to 128}
       Header.lpData := Data;
       Header.dwBufferLength := szNewBuffer + 1;
     end;
    end {if PCM}
   else
     MessageDlg('Non-PCM format audio', mtError, [mbOK], 0);
   end; {with}
end;

procedure TForm1.CheckT;
var  Buf: string;
begin
  buf := Sockets3.Text;
  if buf = 'T' then
  Begin
  Memo1.Lines.Add('**** Found T at Check T****');
  T := True;
  GetStream;
  end
  else
  WavRecord;
```

```
end;
procedure TForm1.PlayStream;
begin
  WaveForm := tAudio_waveform.Create;
  with WaveForm do
   try
    Read(pchar('format.wav'));
    {Display Information about the wav file}
    except
     on E : eMMIO_error do
       MessageDlg(
        'MMIO Error ' + IntToStr(E.MMIO_result) + ': ' + E.Message,
        mtError,
        [mbOK], 0
        );
     on E : eWave_error do
       MessageDlg(
        'Wave Error ' + IntToStr(E.Wave_result) + ': ' + E.Message,
        mtError,
        [mbOK], 0
        );
    end; {except}
  MOVE('temp.dat');

  try
   WaveForm.Play;
   except
    on E : eMMIO_error do
      MessageDlg(
       'MMIO Error ' + IntToStr(E.MMIO_result) + ': ' + E.Message,
       mtError,
       [mbOK], 0
       );

    on E : eWave_error do
      MessageDlg(
       'Wave Error ' + IntToStr(E.Wave_result) + ': ' + E.Message,
       mtError,
       [mbOK], 0
       );
   end; {except}


end;
function TForm1.Sockets2Error:Boolean;
```

```
 begin
  Result := False;
  if Timedout or (ErrorReturn <> 0) then
    begin
    Result := True;
    ErrorReturn := 0;
    MessageDlg( 'Disconnected Data Stream', mtConfirmation,[mbOk],0);
    end;
 end;
 procedure TForm1.CloseSocket2;
 begin
  if Sockets1.SocketNumber <> INVALID_SOCKET then
     begin
     Sockets2.SClose;
     end;
 end;

 function Tform1.TimedOut;
 begin
  if Aborted then
  begin
    Aborted := false;
    Result := True;
  end
  else
    Result := False;
 end;

 procedure TForm1.FormCreate(Sender: TObject);
 begin
  i := 0;
  T := False;
  Abort := false;
  FirstData := true;
  Timer1.Enabled := false;
  Timer2.Enabled := false;
  OutOfTimer2 := false;
  Edit1.Text  := '161.246.18.136';
  Edit2.Text  := '';
  TxMemo.Lines.Clear;
  RxMemo.Lines.Clear;
  Memo1.Lines.clear;
 end;
 procedure TForm1.Compress;
 var Ftest,Fcom:File of byte;
```

```
    bit:byte;
    CBuff,buffer2:Array[1..10000]of byte;
    Buff,SlopeBuff:Array[1..80000] of byte;
    bufferbit:Array[1..8]of byte;
    size,num,a,eq_check,compressbit:integer;
    i,count,t,up,dwn,sum:integer;
    sum1,sum2:longint;
    step,sgn,code,sqnr1,sqnr2:real;
    alfa:real ;
    s,s2:string;
begin

 memo1.lines.clear;
 Assignfile(Ftest,'record.wav');
  {$I+}
      reset(Ftest);{open file for read only may be I
                  can write on it}
  {$I-}

  Assignfile(fcom,'compress.cmp');
  {$I-}
   rewrite(fcom);
  {$I+}
   if IOResult <>0 then
     begin
        memo1.lines.add('...Step4 Write to file ERROR...');
        Exit;
     end;
  rms:=3;
  seek(Ftest,55);
  seek(Fcom,0);
  Blockread(Ftest,buffer2[1],1);
  Blockwrite(Fcom,buffer2[1],1);
  sgn:=buffer2[1];up:=0;Dwn:=0;
  size:=filesize(Ftest)-56;
  eq_check:=0; sum1:=0;sum2:=0;

  while not Eof(Ftest) do
   begin
   if size>80000 then num:=80000
   else num:=size;
   Blockread(Ftest,Buff,num,a);
   count:=0;
   repeat{count}
   i:=0;
```

```
repeat{i=8}
inc(i);
code:=Buff[count+i]-sgn;
if code<0 then
 begin bit:=$00;
     inc(dwn);
     if up>=2      then step:=rms
     else if dwn>=3 then step:=rms*1.5
     else step:= rms*0.66;
    sgn:=sgn-step;up:=0;
    if sgn<0 then sgn:=0;
    end{code<0}
else  if code>0 then
begin Plus(i,bit)  ;
    inc(up);
    if dwn>=2     then step:=rms
    else if up>=3 then step:=rms*1.5
    else step:=rms*0.66;
    sgn:=sgn+step;dwn:=0;
    if sgn>255 then sgn:=255;
    end;{code>0}
if code = 0 then
begin
    eq_check := eq_check +1;
  ;
    if eq_check = 1 then
      begin
      Plus(i,Bit);
      inc(up);
      if dwn>=2     then step:=rms
      else if up>=3 then step:=rms*1.5
      else step:=rms*0.66;
      sgn:=sgn+step;dwn:=0;
      if sgn>255 then sgn:=255;
      end
      else
      if eq_check = 2 then
      begin
      bit:=$00;
      inc(dwn);
      if up>=2      then step:=rms
      else if dwn>=3 then step:=rms*1.5
      else step:= rms*0.66;
      sgn:=sgn-step;up:=0;
      if sgn<0 then sgn:=0;
```

```
                eq_check:=0;
                end;
            end;{code=0}
          sum1:=sum1+sqr(Buff[count+i]);
          sum2:=sum2+sqr(Buff[count+i]-round(sgn));
          BufferBit[i]:=bit;
          until i=8;
        t:=0;
        repeat{T=7}
        inc(T);
        bufferbit[T+1]:=bufferbit[T] or bufferbit[T+1];
        until T = 7;
        count:=count+8;
        compressbit:=count div 8;
        CBuff[compressbit]:=bufferbit[8];
        until count >=num;
        Blockwrite(Fcom,CBuff,compressbit,a);
        size:=size-80000;
        end;{Eof}

        memo1.lines.add('Compressing complete');
        closefile(Ftest);
        closefile(Fcom);
    end;{compress}

    procedure TForm1.Plus(i:integer;var bit:byte);
     begin
     case i of
     1: bit:= $80;
     2: bit:= $40;
     3: bit:= $20;
     4: bit:= $10;
     5: bit:= $08;
     6: bit:= $04;
     7: bit:= $02;
     8: bit:= $01;
     end;
     end;{Plus}



    procedure TForm1.SendStream;
    var buf,Filetosend :string;
       input_file,idx,Numbytes : integer;
       szPCFile : array[0..255] of char;
```

```
    szBuff : array[0..255] of char;
    ByteSend,filesize : longint;
begin
    Memo1.Lines.Add('Send Voice ');
    compress;
    Timer2.Enabled := true;
    FileToSend := 'compress.cmp';
    StrPCopy(szPcFile,FileToSend);
    input_file := _lopen(szPCfile,0);
    OpenedFile := input_file;
    if input_file = -1 then
     begin
      Application.MessageBox('Could not Open Local File','Open Error',MB_ICONEXCLAMATION);
      exit;
     end;
   FileSize := _llseek(input_file,0,2);
   _llseek(input_file,0,0);
   NumBytes :=0;
   ByteSend :=0;
   NumBytes := _lread(input_file,@szbuff[0],255);
   while numbytes > 0 do
    begin
     Sockets2.SSend(Sockets2.SocketNumber,szBuff,NumBytes);
     ByteSend := ByteSend + NumBytes;
     NumBytes := _lread(input_file,@szBuff[0],255);
    if Sockets2Error then
    begin
     _lclose(input_file);
     CloseSocket2;
     Exit;
    end;
   end; {End Send Loop}
    _lclose(input_file);
    if Numbytes =-1 then
    begin
     Memo1.Lines.Add('File Error, File Transfer may be not complete NumByte =-1');
     Memo1.Lines.Add('Total Bytes transfered : '+IntToStr(ByteSend));
    end;
    Timer2.Enabled := false;
    if not T then begin WavRecord; end
    else Getstream;
end;
procedure TForm1.GetStream;
var
 Buff : string;
```

```
tmpinFile,szPCfile,szPCfile2 : array [0..255] of char;
tempbuff : array[0..65535] of char;
szbuff : array[0..8191] of char;
tempinputfile,number,len,output_file,output_file2 : integer;
ByteWritten : Longint;
 bit:byte;

begin
  ByteWritten := 0;
  repeat
   strpCopy(szPCFile,'firsttemp.dat');
   output_file := _lcreat(szPCFile,0);
   len := 8191;
   len := Sockets2.SReceive(Sockets2.SocketNumber,szbuff,len);
   Memo1.Lines.Add(IntTostr(len));
   FirstTempLength := len;
   if Sockets2Error then
    begin
     CloseSocket2;
     Exit;
   end;
   if cancel then
    begin
     Exit;
   end;
   strpCopy(szPCFile2,'temp.dat');
   output_file2 := _lcreat(szPCFile2,0);
   FillChar(TEmpBuff,65535,#00);
    if _lwrite(output_file2,TempBuff,65535) =- 1 then
      begin
        Memo1.Lines.Add('Error in write file ');
        Aborted := true;
        exit;
      end;
   _lclose(output_file2);
   if len > 0 then
   begin
    if _lwrite(output_file,szbuff,len) =- 1 then
      begin
        Memo1.Lines.Add('Error in write file ');
        Aborted := true;
        exit;
      end;
    _lclose(output_file);
    Decompress;
```

```
      end;
    {old part}
      ByteWritten := ByteWritten + len;
    until ByteWritten >= 1000;
    _lclose(output_file);
    output_file := 0;
    T := false;
    if not T then begin WavRecord; end
    else
    GetStream;
  end;
  procedure TForm1.Decompress;
  var   Fcom,Fdecom:File of byte;
        bit:byte;
        buffer2:Array[1..8192]of byte;
        DBuff:Array[1..65536] of real;
        Xbuff:Array[1..65536] of byte;
        buff:Array[1..65536]of byte;
        count,size,code,num,a:integer;
        i,j,counter,t,z,dwn,up:integer;
        rms,step,sgn:real;
  {Move }
       pPrev8bitSample,
   p8bitSample:  ^Byte;
   pPrev16bitSample,
   p16bitSample: ^SmallInt;
   szNewBuffer:  Cardinal;
   NewData:      PChar;
   GainFactor,
   Period:   integer;
   numbyte,fsize:integer;
   temp : array[0..255] of char;
   temp_file : integer;

   begin
   rms:=5;
   Assignfile(Fcom,'firsttemp.dat');
    {$I+}
   reset(Fcom);{open file for read only may be I
                  can write on it}
    {$I-}
   Assignfile(fdecom,'temp.dat');
    {$I-}
   reset(fdecom);
   {$I+}
```

```
if IOResult <>0 then
    begin
      memo1.lines.add('...Step4 Write to file ERROR...');
      Exit;
    end;
 seek(Fcom,0);
 seek(Fdecom,0);
 buffer2[1] := 128;
 Fillchar(DBuff,65536,#00);
 Blockwrite(Fdecom,buffer2[1],1);
 size:=(filesize(Fcom)-1)*8;
 Memo1.lines.add('...Size '+ IntTostr(size));
 t:=0;z:=0;
 i:=round(rms);
 rms:=3;
 while not Eof(Fcom) do
  begin
   if size>8192 then num:=8192
    else num:=8192;
    Blockread(Fcom,Buff,num,a);
    j:=0;z:=0;
  repeat{j=num}
    i:=0;inc(z) ;
   repeat{i=8}
    inc(i);inc(t);
    Plus(i,bit);
   if t=1 then sgn:=buffer2[1]
   else sgn:=DBuff[j+i-1];
    code:=Buff[z] and bit ;
      if code<=0 then
      begin inc(dwn);
       if up>=2 then step:=rms
       else if dwn>=3 then step:=rms*1.5
       else step:=rms*0.66;
       up:=0;
       DBuff[j+i]:=sgn-step;
       if DBuff[j+i]<0 then DBuff[j+i]:=0;
       end {code<0}
    else
       begin  inc(up);
        if dwn>=2 then step:=rms
         else if up>=3 then step:=rms*1.5
         else step:=rms*0.66;
         dwn:=0;
         DBuff[j+i]:=sgn+step;
```

```
        if DBuff[j+i]>255 then DBuff[j+i]:=255;
        end;{else}
      Xbuff[j+i]:=Round(DBuff[j+i]);
      inc(t);
      until i=8;
      j:=j+8;
      until j>=num*8;
      size:=size-10000  ;
      Blockwrite(Fdecom,xbuff,j,a);

      end;{Eof}
      closefile(Fcom);
      closefile(Fdecom);
      {Newpart}
    WaveForm := tAudio_waveform.Create;
    with WaveForm do
     try
      Read(pchar('format.wav'));
      {Display Information about the wav file}
      except
       on E : eMMIO_error do
          MessageDlg(
           'MMIO Error ' + IntToStr(E.MMIO_result) + ': ' + E.Message,
           mtError,
           [mbOK], 0
           );
       on E : eWave_error do
          MessageDlg(
           'Wave Error ' + IntToStr(E.Wave_result) + ': ' + E.Message,
           mtError,
           [mbOK], 0
           );
      end; {except}
    with WaveForm do begin
     if Format.wFormatTag = WAVE_FORMAT_PCM then begin
      if Format.wBitsPerSample = 8 then begin
       {working with an 8-bit sample}
       {Re-allocate memory for the new size of the wave data}
       StrpCopy(temp,'temp.dat');
       temp_file := _lopen(temp,0);
       fsize := _llseek(temp_file,0,2);
       szNewBuffer := FirstTempLength*8;
       NewData := strAlloc((szNewBuffer));
       _llseek(temp_file,0,0);
       _lread(Temp_file,@NewData[0],(szNewBuffer));
```

```
        StrDispose(data);
        Data := NewData;
        _lclose(temp_file);
        Header.lpData := Data;
        Header.dwBufferLength := szNewBuffer + 1;
       end;
      end {if PCM}
     else
      MessageDlg('Non-PCM format audio', mtError, [mbOK], 0);
     end; {with}


    try
     WaveForm.Play;
     except
      on E : eMMIO_error do
        MessageDlg(
         'MMIO Error ' + IntToStr(E.MMIO_result) + ': ' + E.Message,
         mtError,
         [mbOK], 0
         );

      on E : eWave_error do
        MessageDlg(
         'Wave Error ' + IntToStr(E.Wave_result) + ': ' + E.Message,
         mtError,
         [mbOK], 0
         );
    end; {except}

      {process the waveform data to add an echo effect}


     end;{Decoder}

procedure TForm1.WavRecord;
begin
 Timer1.Enabled := true;
 mciSendString(
  'Open New type WaveAudio alias wave', //Start a new waveaudio file in memory
  ReturnString,
  256,
  0
  );
```

```
  mciSendString(
    'set wave bitpersample 8',          //Set the bits per sample to 8
    ReturnString,
    256,
    0
    );

  mciSendString(
    'set wave samplespersec 11025',     //Set the sample rate to 11025
    ReturnString,
    256,
    0
    );

  mciSendString(
    'set wave channels 1',              //Set the number of channels to 2
    ReturnString,
    256,
    0
    );

  mciSendString(
    'record wave',                      //Start recording
    ReturnString,
    256,
    0
    );
  Memo1.Lines.Add('Recording..');
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Timer1.Enabled := false;
  Memo1.Lines.Add('Stop...');
    mciSendString(                      //Stop recording
    'stop wave',
    ReturnString,
    256,0
    );

  mciSendString(
    'save wave record.wav',             //Save the file
    ReturnString,
    256,0
    );
```

```
  mciSendString(                    //Close the file
    'close wave',
    ReturnString,
    256,0
    );
  DetectVoice;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
    abort := true;
end;
procedure TForm1.DetectVoice;
var
 pPrev8bitSample,
 p8bitSample:  ^Byte;
 pPrev16bitSample,
 p16bitSample: ^SmallInt;
 i,s,num_detect:          integer;
 szNewBuffer:  Cardinal;
 NewData:     PChar;
 GainFactor,
 Period:   integer;
 numbyte,fsize:integer;
 temp : array[0..255] of char;
 temp_file : integer;
 buf: string;
begin
      VoiceDetected := false;
      StrpCopy(temp,'record.wav');
      temp_file := _lopen(temp,0);
      fsize := _llseek(temp_file,0,2);
      szNewBuffer := fsize;
      NewData := strAlloc(szNewBuffer);
      _llseek(temp_file,0,0);
      _lread(Temp_file,@NewData[0],fsize);
      p8Bitsample := addr (NewData^);
      num_detect := 0;
      for i:= 1 to 15 do
       begin
        inc(p8bitsample,20*i);
        s := p8bitsample^;
        if abs(s-128) > 5 then
          num_detect := num_detect+1;
```

```
            end;
        if num_detect > 2 then VoiceDetected := true;
        _lclose(temp_file);
        if VoiceDetected then
         begin
          Memo1.Lines.Add('Detected your voice');
         if not T then
          begin
           SendSockets3;
           SendStream;
          end
          else
           GetStream;
        end
        else
         if not T then begin WavRecord; end
         else
         GetStream;
end;

procedure TForm1.Sockets3SessionClosed(Sender: TObject; Socket: Integer);
begin
        Sockets3.SClose;
end;

procedure TForm1.Sockets3DataAvailable(Sender: TObject; Socket: Integer);
var idx2: Integer;
   Temp,buf : string;
begin
 Memo1.Lines.Add('Data recevied from socket(' + IntToStr(Socket) + ')');
 buf := Sockets3.Text;
 Temp := buf;
 idx2 := Pos( '>' , Temp);
 if idx2 > 0 then
  begin
    TelNum := Copy(temp,idx2+1,7);
    Edit2.Text := TelNum;
 end;
 if buf = 'T' then
 begin
 Memo1.Lines.Add('**** Found T ****');
 T := true;
 Timer1.Interval := 2000;
 end;
 Memo1.Lines.Add('-->' + buf + '<--');
```

```
end;
procedure TForm1.TxMemoKeyPress(Sender: TObject; var Key: Char);
var s: string;
begin
 if CommPortDriver.Connected then
  // Commit data only when RETURN key is pressed
  case Key of
   #13: if TxMemo.Lines.Count>0 then
        begin
         s := TxMemo.Lines[TxMemo.Lines.Count-1];
         CommPortDriver.SendData( pchar(s), length(s) );
         CommPortDriver.SendData( @Key, 1 );
        end
        else CommPortDriver.SendData( @Key, 1 );

  end;
end;
procedure TForm1.CommPortDriverReceiveData(Sender: TObject;
 DataPtr: Pointer; DataSize: Integer);
var p: pchar;
  s: string;
begin
 // Get current line
 if RxMemo.Lines.Count <> 0 then
  s := RxMemo.Lines[RxMemo.Lines.Count-1]
 else
  s := '';
 // Parse incoming text
 p := DataPtr;
 while DataSize > 0 do
 begin
  case p^ of
   #10:; // LF
   #13: // CR - cursor to next line
    begin
     if RxMemo.Lines.Count <> 0 then
      RxMemo.Lines[RxMemo.Lines.Count-1] := s
     else
      RxMemo.Lines.Add( s );
     RxMemo.Lines.Add( '' );
     s := '';
    end;
   #8: // Backspace - delete last char
    delete( s, length(s), 1 );
   else // Any other char - add it to the current line
```

```
      s := s + p^;
    end;
    dec( DataSize );
    inc( p );
  end;
  // If current line isn't empty
  if (s<>'') then
    if RxMemo.Lines.Count <> 0 then
      // Update current line
      RxMemo.Lines[RxMemo.Lines.Count-1] := s
    else
      // New line - add it
      RxMemo.Lines.Add( s );
  RxMemo.Update;
  if (TelNumCount < 8) and RealTel then
  begin
  TelNumCount := TelNumCount+1;
   begin
    case TelNum [TelNumCount] of
     '0': dial('0');
     '1': dial('1');
     '2': dial('2');
     '3': dial('3');
     '4': dial('4');
     '5': dial('5');
     '6': dial('6');
     '7': dial('7');
     '8': dial('8');
     '9': dial('9');
    end;
   end;
 end;
end;
procedure TForm1.Button3Click(Sender: TObject);
var s: string;
begin
 // Apply settings
  ApplyCommSettings;
  // Connect
  if CommPortDriver.Connect then
  begin
    TxMemo.SetFocus;
    StatusBar1.SimpleText:='Connected to Serial Port';
    s := TelNum;
    CommPortDriver.SendData( pchar(s), length(s) );
```

```
    end
    else // Error !
     begin
      MessageBeep( 0 );
     end;
  end;

  procedure TForm1.Button4Click(Sender: TObject);
  begin
   CommPortDriver.Disconnect;
  end;

  procedure TForm1.BaudRateRGClick(Sender: TObject);
  begin
    CommPortDriver.ComPortSpeed := TComPortBaudRate(BaudRateRG.ItemIndex+1);
  end;

  procedure TForm1.DataBitsRGClick(Sender: TObject);
  begin
    CommPortDriver.ComPortDataBits := TComPortDataBits(DataBitsRG.ItemIndex);
  end;

  procedure TForm1.ParityRGClick(Sender: TObject);
  begin
    CommPortDriver.ComPortParity := TComPortParity(ParityRG.ItemIndex);
  end;

  procedure TForm1.HandshakingRGClick(Sender: TObject);
  begin
    case HandshakingRG.ItemIndex of
     0: // none
      begin
       CommPortDriver.ComPortHwHandshaking := hhNone;
       CommPortDriver.ComPortSwHandshaking := shNone;
      end;
     1: // RTS/CTS
      begin
       CommPortDriver.ComPortHwHandshaking := hhRTSCTS;
       CommPortDriver.ComPortSwHandshaking := shNone;
      end;
     2: // XON/XOFF
      begin
       CommPortDriver.ComPortHwHandshaking := hhNone;
       CommPortDriver.ComPortSwHandshaking := shXONXOFF;
      end;
```

```
      3: // RTS/CTS + XON/XOFF
        begin
          CommPortDriver.ComPortHwHandshaking := hhRTSCTS;
          CommPortDriver.ComPortSwHandshaking := shXONXOFF;
        end;
    end;
  end;

  procedure TForm1.ComPortRGClick(Sender: TObject);
  begin
   // Apply com settings
    //ApplyCommSettings;
  end;
  procedure TForm1.ApplyCommSettings;
  var wasConnected: boolean;
  begin
    wasConnected := CommPortDriver.Connected;
    // This change needs CommPortDriver not connected
    if wasConnected then
      Button4Click( nil );

    CommPortDriver.ComPort := TComPortNumber(ord(ComPortRG.ItemIndex));
    BaudRateRGClick( nil );
    DataBitsRGClick( nil );
    ParityRGClick( nil );
    HandshakingRGClick( nil );

    // Reconnect
    if wasConnected then
      Button3Click( nil );
  end;
  procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
  begin
      Sockets1.Sclose;
      Sockets2.Sclose;
      Sockets3.Sclose;
  end;


  procedure TForm1.Sockets3SessionAvailable(Sender: TObject;
    Socket: Integer);
  var   new_client: TSocket;
  begin
    new_client := Sockets3.SAccept;
    Memo1.Lines.Add('Client session accepted, socket(' + IntToStr(new_client) + ')' +
```

```
    ' Peer IP='+Sockets3.GetPeerIPAddr(new_client) +
    ' Peer Port='+Sockets3.GetPeerPort(new_client));
   WavRecord;


end;


procedure TForm1.Sockets3SessionConnected(Sender: TObject;
  Socket: Integer);
begin
  Memo1.Lines.Add('Connected to IP='+Sockets3.GetPeerIPAddr(sockets3.SocketNumber)+
    ' on his port #'+Sockets3.GetPeerPort(Sockets3.SocketNumber));
end;



procedure TForm1.Timer2Timer(Sender: TObject);
begin
  Timer2.Enabled := false;
  _lclose(OpenedFile);
  WavRecord;
end;



procedure TForm1.Button15Click(Sender: TObject);
var s:String;
    i :integer;
begin
    RealTel := false;
    TelNumCount := 1;
    TelNum := Edit2.Text;
    Dial('0');
    TxMemo.Lines.Add('Dial 0');
    for i := 1 to 1850 do
     begin
      TxMemo.Lines.Clear;
      TxMemo.Lines.Add(IntToStr(i));
     end;
    RealTel := true;
    Dial(TelNum[TelNumCount]);
```

```
end;
procedure TForm1.Dial(s:string);
begin
   TxMemo.Lines.Add(s);
   CommPortDriver.SendData( pchar(s), length(s) );
end;


procedure TForm1.Button5Click(Sender: TObject);
var s : string;
begin
   s:='C';
   CommPortDriver.SendData( pchar(s), length(s) );
   Memo1.Lines.Add('C');
end;


end.
```

```
unit Client2;

interface

uses
 Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
 StdCtrls, Sockets, Winsock, MMioWrap, MMsystem, WaveWrap, Waveplay, inifiles,
 ExtCtrls, ComCtrls ;

type
 TForm1 = class(TForm)
  Edit1: TEdit;
  Button1: TButton;
  Button2: TButton;
  Sockets1: TSockets;
  Sockets2: TSockets;
  Memo1: TMemo;
  Timer1: TTimer;
  Sockets3: TSockets;
  Edit2: TEdit;
  StatusBar1: TStatusBar;
  Label1: TLabel;
  Label2: TLabel;
  Button4: TButton;
  Timer2: TTimer;
  procedure Button1Click(Sender: TObject);
  procedure GetStream;
  procedure PlayStream;
  function TimedOut:boolean;
  procedure Move(filename : string);
  function Sockets2Error:boolean;
  procedure closeSocket2;
  procedure FormCreate(Sender: TObject);
  procedure Timer1Timer(Sender: TObject);
  procedure sendstream;
  procedure WavRecord;
  procedure Button3Click(Sender: TObject);
  procedure Main;
  procedure CheckT;
  procedure Button2Click(Sender: TObject);
  procedure DetectVoice;
  procedure Compress;
  procedure SendSockets3;
  procedure Sockets3SessionClosed(Sender: TObject; Socket: Integer);
  procedure Button4Click(Sender: TObject);
```

```
    procedure Sockets3SessionAvailable(Sender: TObject; Socket: Integer);
    procedure Sockets3DataAvailable(Sender: TObject; Socket: Integer);
    procedure Sockets3SessionConnected(Sender: TObject; Socket: Integer);

    procedure Timer2Timer(Sender: TObject);
    procedure Plus(i:integer;var bit:byte);
    procedure Decompress;
  private
  { Private declarations }
  public
   { Public declarations }
  end;

var
  Form1: TForm1;
  OutOfTimer2,abort,VoiceDetected : Boolean;
  WaveForm: tAudio_waveform;
  ReturnString: array[0..255] of Char;
  Aborted : boolean;
  cancel: boolean;
  OpenedFile,i,ErrorReturn,FirstTempLength  : Integer;
  rms,slope,scale_factor:real ;
  T : boolean;
implementation

{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
begin
  Abort := False;
  Timer1.Enabled := false;
  Sockets1.NonBlocking := false;  {Send}
  Sockets1.Timeout := 30;
  Sockets1.IpAddr := Edit1.Text;
  Sockets1.Port := '2541';
  Sockets1.SConnect;
  Memo1.Lines.Add('.. Voice Channel Connected .. ');
  Sockets2.NonBlocking := false;  {Send}
  Sockets2.Timeout := 30;
  Sockets2.IpAddr := Edit1.Text;
  Sockets2.Port := '2542';
  Sockets2.Sconnect;
  Memo1.Lines.Add('.. Control Channel Connected .. ');
end;
```

```
procedure TForm1.SendSockets3;
var  buff : array[0..255] of char;
begin
   Sockets3.Text := 'T';
   Memo1.Lines.Add('... Send T ... ');
end;



procedure TForm1.Main;
begin
 WavRecord;
end;
procedure TForm1.Move(filename : string);
var
 pPrev8bitSample,
 p8bitSample:  ^Byte;
 pPrev16bitSample,
 p16bitSample: ^SmallInt;
 i,num_detect:          integer;
 szNewBuffer:  Cardinal;
 NewData:     PChar;
 GainFactor,
 Period,s:   integer;
 numbyte,fsize:integer;
 temp : array[0..255] of char;
 temp_file : integer;
begin
 {process the waveform data to add an echo effect}
 with WaveForm do begin

  if Format.wFormatTag = WAVE_FORMAT_PCM then begin
   if Format.wBitsPerSample = 8 then begin
    {working with an 8-bit sample}
    {Re-allocate memory for the new size of the wave data}
    StrpCopy(temp,filename);
    temp_file := _lopen(temp,0);
    fsize := _llseek(temp_file,0,2);
    szNewBuffer := fsize;
    NewData := strAlloc(szNewBuffer);
    _llseek(temp_file,0,0);
    _lread(Temp_file,@NewData[0],fsize);
    StrDispose(Data);

    Data := NewData;
```

```
      _lclose(temp_file);
      {initialize the new part of the buffer to 128}
      Header.lpData := Data;
      Header.dwBufferLength := szNewBuffer + 1;
     end;
    end {if PCM}
   else
    MessageDlg('Non-PCM format audio', mtError, [mbOK], 0);
   end; {with}
end;

procedure TForm1.CheckT;
var  Buf: string;
begin
 Memo1.Lines.Add(Buf);
 buf := Sockets3.Text;
 if buf = 'T' then
 begin
 Memo1.Lines.Add('**** Found T at Check T****');
 T := True;
 GetStream;
 end
 else
 WavRecord;
end;

procedure TForm1.PlayStream;
begin
  WaveForm := tAudio_waveform.Create;
   with WaveForm do
    try
     Read(pchar('format.wav'));
     {Display Information about the wav file}
     except
      on E : eMMIO_error do
         MessageDlg(
          'MMIO Error ' + IntToStr(E.MMIO_result) + ': ' + E.Message,
          mtError,
          [mbOK], 0
          );
      on E : eWave_error do
         MessageDlg(
          'Wave Error ' + IntToStr(E.Wave_result) + ': ' + E.Message,
          mtError,
          [mbOK], 0
```

```
            );
        end; {except}
    MOVE('temp.dat');

    try
     WaveForm.Play;
     except
      on E : eMMIO_error do
         MessageDlg(
          'MMIO Error ' + IntToStr(E.MMIO_result) + ': ' + E.Message,
          mtError,
          [mbOK], 0
          );

      on E : eWave_error do
         MessageDlg(
          'Wave Error ' + IntToStr(E.Wave_result) + ': ' + E.Message,
          mtError,
          [mbOK], 0
          );
     end; {except}


end;
function TForm1.Sockets2Error:Boolean;
begin
 Result := False;
 if Timedout or (ErrorReturn <> 0) then
   begin
   Result := True;
   ErrorReturn := 0;
   MessageDlg( 'Disconnected Data Stream', mtConfirmation,[mbOk],0);
   end;
end;
procedure TForm1.CloseSocket2;
begin
  if Sockets1.SocketNumber <> INVALID_SOCKET then
     begin
     Sockets2.SClose;
     end;
end;

function Tform1.TimedOut;
begin
  if Aborted then
```

```
 begin
   Aborted := false;
   Result := True;
 end
 else
   Result := False;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
 i:=0;
 Abort := false;
 Edit1.Text  := '161.246.18.136';
 Edit2.Text :='';
 Memo1.Lines.clear;

 Timer1.Enabled := false;
 Timer2.Enabled := false;
 OutOfTimer2 := false;
end;
Procedure TForm1.Compress;
var Ftest,Fcom:File of byte;
    bit:byte;
    CBuff,buffer2:Array[1..10000]of byte;
    Buff,SlopeBuff:Array[1..80000] of byte;
    bufferbit:Array[1..8]of byte;
    size,num,a,eq_check,compressbit:integer;
    i,count,t,up,dwn,sum:integer;
    sum1,sum2:longint;
    step,sgn,code,sqnr1,sqnr2:real;
    alfa:real ;
    s,s2:string;
 begin

  memo1.lines.clear;
  Assignfile(Ftest,'record.wav');
   {$I+}
      reset(Ftest);{open file for read only may be I
                  can write on it}
   {$I-}

   Assignfile(fcom,'compress.cmp');
   {$I-}
    rewrite(fcom);
   {$I+}
```

```
 if IOResult <>0 then
   begin
      memo1.lines.add('...Step4 Write to file ERROR...');
      Exit;
   end;
rms:=3;
seek(Ftest,55);
seek(Fcom,0);
Blockread(Ftest,buffer2[1],1);
Blockwrite(Fcom,buffer2[1],1);
sgn:=buffer2[1];up:=0;Dwn:=0;
size:=filesize(Ftest)-56;
eq_check:=0; sum1:=0;sum2:=0;

while not Eof(Ftest) do
 begin
 if size>80000 then num:=80000
 else num:=size;
 Blockread(Ftest,Buff,num,a);
 count:=0;
 repeat{count}
 i:=0;
 repeat{i=8}
 inc(i);
 code:=Buff[count+i]-sgn;
 if code<0 then
  begin bit:=$00;
      inc(dwn);
      if up>=2      then step:=rms
      else if dwn>=3 then step:=rms*1.5
      else step:= rms*0.66;
     sgn:=sgn-step;up:=0;
     if sgn<0 then sgn:=0;
     end{code<0}
 else  if code>0 then
 begin Plus(i,bit)  ;
     inc(up);
     if dwn>=2     then step:=rms
     else if up>=3 then step:=rms*1.5
     else step:=rms*0.66;
     sgn:=sgn+step;dwn:=0;
     if sgn>255 then sgn:=255;
     end;{code>0}
 if code = 0 then
 begin
```

```
        eq_check := eq_check +1;
      ;
        if eq_check = 1 then
          begin
          Plus(i,Bit);
          inc(up);
          if dwn>=2    then step:=rms
          else if up>=3 then step:=rms*1.5
          else step:=rms*0.66;
          sgn:=sgn+step;dwn:=0;
          if sgn>255 then sgn:=255;
          end
          else
          if eq_check = 2 then
          begin
          bit:=$00;
          inc(dwn);
          if up>=2     then step:=rms
          else if dwn>=3 then step:=rms*1.5
          else step:= rms*0.66;
          sgn:=sgn-step;up:=0;
          if sgn<0 then sgn:=0;
          eq_check:=0;
          end;
    end;{code=0}
   sum1:=sum1+sqr(Buff[count+i]);
   sum2:=sum2+sqr(Buff[count+i]-round(sgn));
   BufferBit[i]:=bit;
   until i=8;
  t:=0;
  repeat{T=7}
  inc(T);
  bufferbit[T+1]:=bufferbit[T] or bufferbit[T+1];
  until T = 7;
  count:=count+8;
  compressbit:=count div 8;
  CBuff[compressbit]:=bufferbit[8];
  until count >=num;
  Blockwrite(Fcom,CBuff,compressbit,a);
  size:=size-80000;
  end;{Eof}

  memo1.lines.add('Compressing complete');
  closefile(Ftest);
  closefile(Fcom);
```

```
end;{compress}

procedure TForm1.Plus(i:integer;var bit:byte);
 begin
 case i of
 1: bit:= $80;
 2: bit:= $40;
 3: bit:= $20;
 4: bit:= $10;
 5: bit:= $08;
 6: bit:= $04;
 7: bit:= $02;
 8: bit:= $01;
 end;
 end;{Plus}

procedure TForm1.GetStream;
var
 Buff : string;
 tmpinFile,szPCfile,szPCfile2 : array [0..255] of char;
 tempbuff : array[0..65535] of char;
 szbuff : array[0..8191] of char;
 tempinputfile,number,len,output_file,output_file2 : integer;
 ByteWritten : Longint;
  bit:byte;

begin
   ByteWritten := 0;
   repeat
    strpCopy(szPCFile,'firsttemp.dat');
    output_file := _lcreat(szPCFile,0);
    len := 8191;
    len := Sockets2.SReceive(Sockets2.SocketNumber,szbuff,len);
    Memo1.Lines.Add(IntTostr(len));
    FirstTempLength := len;
    if Sockets2Error then
     begin
      CloseSocket2;
      Exit;
    end;
    if cancel then
     begin
      Exit;
    end;
    strpCopy(szPCFile2,'temp.dat');
```

```
      output_file2 := _lcreat(szPCFile2,0);
      FillChar(TEmpBuff,65535,#00);
      if _lwrite(output_file2,TempBuff,65535) =- 1 then
        begin
          Memo1.Lines.Add('Error in write file ');
          Aborted := true;
          exit;
        end;
      _lclose(output_file2);
      if len > 0 then
      begin
       if _lwrite(output_file,szbuff,len) =- 1 then
        begin
          Memo1.Lines.Add('Error in write file ');
          Aborted := true;
          exit;
        end;
       _lclose(output_file);
       Decompress;
      end;
{old part}
    ByteWritten := ByteWritten + len;
  until ByteWritten >= 1000;
  _lclose(output_file);
  output_file := 0;
  T := false;
  if not T then begin WavRecord; end
  else
  GetStream;
end;
procedure TForm1.Decompress;
var   Fcom,Fdecom:File of byte;
      bit:byte;
      buffer2:Array[1..8192]of byte;
      DBuff:Array[1..65536] of real;
      Xbuff:Array[1..65536] of byte;
      buff:Array[1..65536]of byte;
      count,size,code,num,a:integer;
      i,j,counter,t,z,dwn,up:integer;
      rms,step,sgn:real;
 {Move }
        pPrev8bitSample,
  p8bitSample:  ^Byte;
  pPrev16bitSample,
  p16bitSample: ^SmallInt;
```

```
szNewBuffer: Cardinal;
NewData:     PChar;
GainFactor,
Period:   integer;
numbyte,fsize:integer;
temp : array[0..255] of char;
temp_file : integer;

begin
rms:=5;
Assignfile(Fcom,'firsttemp.dat');
{$I+}
reset(Fcom);{open file for read only may be I
                can write on it}
{$I-}
Assignfile(fdecom,'temp.dat');
{$I-}
reset(fdecom);
{$I+}
if IOResult <>0 then
    begin
       memo1.lines.add('...Step4 Write to file ERROR...');
       Exit;
    end;
seek(Fcom,0);
seek(Fdecom,0);
buffer2[1] := 128;
Fillchar(DBuff,65536,#00);
Blockwrite(Fdecom,buffer2[1],1);
size:=(filesize(Fcom)-1)*8;
Memo1.lines.add('...Size '+ IntTostr(size));
t:=0;z:=0;
i:=round(rms);
rms:=3;
while not Eof(Fcom) do
 begin
  if size>8192 then num:=8192
   else num:=8192;
   Blockread(Fcom,Buff,num,a);
   j:=0;z:=0;
  repeat{j=num}
   i:=0;inc(z) ;
  repeat{i=8}
   inc(i);inc(t);
   Plus(i,bit);
```

```
    if t=1 then sgn:=buffer2[1]
    else sgn:=DBuff[j+i-1];
     code:=Buff[z] and bit ;
       if code<=0 then
       begin inc(dwn);
        if up>=2 then step:=rms
        else if dwn>=3 then step:=rms*1.5
        else step:=rms*0.66;
        up:=0;
        DBuff[j+i]:=sgn-step;
        if DBuff[j+i]<0 then DBuff[j+i]:=0;
        end {code<0}
     else
        begin  inc(up);
        if dwn>=2 then step:=rms
         else if up>=3 then step:=rms*1.5
         else step:=rms*0.66;
         dwn:=0;
         DBuff[j+i]:=sgn+step;
         if DBuff[j+i]>255 then DBuff[j+i]:=255;
         end;{else}
    Xbuff[j+i]:=Round(DBuff[j+i]);
    inc(t);
    until i=8;
    j:=j+8;
    until j>=num*8;
    size:=size-10000  ;
    Blockwrite(Fdecom,xbuff,j,a);

    end;{Eof}
    closefile(Fcom);
    closefile(Fdecom);
    {Newpart}
  WaveForm := tAudio_waveform.Create;
  with WaveForm do
   try
    Read(pchar('format.wav'));
    {Display Information about the wav file}
    except
     on E : eMMIO_error do
        MessageDlg(
         'MMIO Error ' + IntToStr(E.MMIO_result) + ': ' + E.Message,
         mtError,
         [mbOK], 0
         );
```

```
        on E : eWave_error do
          MessageDlg(
           'Wave Error ' + IntToStr(E.Wave_result) + ': ' + E.Message,
           mtError,
           [mbOK], 0
           );
      end; {except}
    with WaveForm do begin
     if Format.wFormatTag = WAVE_FORMAT_PCM then begin
      if Format.wBitsPerSample = 8 then begin
       {working with an 8-bit sample}
       {Re-allocate memory for the new size of the wave data}
       StrpCopy(temp,'temp.dat');
       temp_file := _lopen(temp,0);
       fsize := _llseek(temp_file,0,2);
       szNewBuffer := FirstTempLength*8;
       NewData := strAlloc((szNewBuffer));
       _llseek(temp_file,0,0);
       _lread(Temp_file,@NewData[0],(szNewBuffer));
       StrDispose(data);
       Data := NewData;
       _lclose(temp_file);
       Header.lpData := Data;
       Header.dwBufferLength := szNewBuffer + 1;
      end;
     end {if PCM}
    else
      MessageDlg('Non-PCM format audio', mtError, [mbOK], 0);
    end; {with}


   try
    WaveForm.Play;
   except
     on E : eMMIO_error do
       MessageDlg(
        'MMIO Error ' + IntToStr(E.MMIO_result) + ': ' + E.Message,
        mtError,
        [mbOK], 0
        );

     on E : eWave_error do
       MessageDlg(
        'Wave Error ' + IntToStr(E.Wave_result) + ': ' + E.Message,
        mtError,
```

```
        [mbOK], 0
        );
    end; {except}

      {process the waveform data to add an echo effect}


    end;{Decoder}



procedure TForm1.SendStream;
var buf,Filetosend :string;
   input_file,idx,Numbytes : integer;
   szPCFile : array[0..255] of char;
   szBuff : array[0..255] of char;
   ByteSend,filesize : longint;
begin
    Memo1.Lines.Add('Send Voice ');
    compress;
    Timer2.Enabled := true;
    FileToSend := 'compress.cmp';
    StrPCopy(szPcFile,FileToSend);
    input_file := _lopen(szPCfile,0);
    OpenedFile := input_file;
    if input_file = -1 then
     begin
      Application.MessageBox('Could not Open Local File','Open Error',MB_ICONEXCLAMATION);
      exit;
     end;
  FileSize := _llseek(input_file,0,2);
  _llseek(input_file,0,0);
  NumBytes :=0;
  ByteSend :=0;
  NumBytes := _lread(input_file,@szbuff[0],255);
  while numbytes > 0 do
   begin
    Sockets2.SSend(Sockets2.SocketNumber,szBuff,NumBytes);
    ByteSend := ByteSend + NumBytes;
    NumBytes := _lread(input_file,@szBuff[0],255);
   if Sockets2Error then
   begin
    _lclose(input_file);
    CloseSocket2;
```

```
      Exit;
     end;
    end; {End Send Loop}
    _lclose(input_file);
    if Numbytes =-1 then
    begin
     Memo1.Lines.Add('File Error, File Transfer may be not complete NumByte =-1');
     Memo1.Lines.Add('Total Bytes transfered : '+IntToStr(ByteSend));
    end;
    Timer2.Enabled := false;
    if not T then begin WavRecord; end
    else Getstream;
end;

procedure TForm1.WavRecord;
begin
 Timer1.Enabled := true;
 mciSendString(
  'Open New type WaveAudio alias wave'. //Start a new waveaudio file in memory
  ReturnString,
  256,
  0
  );

 mciSendString(
  'set wave bitpersample 8',          //Set the bits per sample to 8
  ReturnString,
  256,
  0
  );

 mciSendString(
  'set wave samplespersec 11025',     //Set the sample rate to 11025
  ReturnString,
  256,
  0
  );

 mciSendString(
  'set wave channels 1',              //Set the number of channels to 2
  ReturnString,
  256,
  0
  );
```

```
  mciSendString(
   'record wave',                    //Start recording
   ReturnString,
   256,
   0
   );
  Memo1.Lines.Add('..Recording..');
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Timer1.Enabled := false;
  Memo1.Lines.Add('...Stop...');
   mciSendString(                    //Stop recording
  'stop wave',
  ReturnString,
  256,0
  );

 mciSendString(
  'save wave record.wav',          //Save the file
  ReturnString,
  256,0
  );

 mciSendString(                     //Close the file
  'close wave',
  ReturnString,
  256,0
  );
  DetectVoice;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
 Sockets3.NonBlocking := false;  {Send}
 Sockets3.Timeout := 30;
 Sockets3.IpAddr := Edit1.Text;
 Sockets3.Port := '2543';
 Sockets3.Sconnect;
 Sockets3.Text := Edit2.Text;
 StatusBar1.SimpleText := 'Connected to Server';
 WavRecord;
end;
```

```
procedure TForm1.Button2Click(Sender: TObject);
begin
   abort := true;
end;
procedure TForm1.DetectVoice;
var
 pPrev8bitSample,
 p8bitSample:  ^Byte;
 pPrev16bitSample,
 p16bitSample: ^SmallInt;
 i,s,num_detect:          integer;
 szNewBuffer:  Cardinal;
 NewData:     PChar;
 GainFactor,
 Period:   integer;
 numbyte,fsize:integer;
 temp : array[0..255] of char;
 temp_file : integer;
 buf : string;
begin
     VoiceDetected := false;
     StrpCopy(temp,'record.wav');
     temp_file := _lopen(temp,0):
     fsize := _llseek(temp_file,0,2);
     szNewBuffer := fsize;
     NewData := strAlloc(szNewBuffer);
     _llseek(temp_file,0,0);
     _lread(Temp_file,@NewData[0],fsize);
     p8Bitsample := addr (NewData^);
     num_detect := 0;
     for i:= 1 to 15 do
      begin
       inc(p8bitsample,20*i);
       s := p8bitsample^;
       if abs(s-128) > 5 then
       num_detect := num_detect+1;
         end;
     if num_detect > 2 then VoiceDetected := true;
     _lclose(temp_file);
     if VoiceDetected then
      begin
        Memo1.Lines.Add('Detected your voice');
        if not T then
        begin
         SendSockets3;
```

```
          SendStream;
          end
          else
          GetStream;
        end
        else
          if not T then begin WavRecord; end
          else
          GetStream;
end;

procedure TForm1.Sockets3SessionClosed(Sender: TObject; Socket: Integer);
begin
 Sockets3.SClose;
end;

procedure TForm1.Button4Click(Sender: TObject);
var
 szMsg: array[0..255] of char;
begin
 Sockets3.IPAddr := Edit1.Text;
 Sockets3.Port := '2543';
 Sockets3.SConnect;
 Sockets3.Text := '>'+Edit2.Text;
 WavRecord;
end;



procedure TForm1.Sockets3SessionAvailable(Sender: TObject;
 Socket: Integer);

var   new_client: TSocket;
begin
 new_client := Sockets3.SAccept;
 Memo1.Lines.Add('Client session accepted, socket(' + IntToStr(new_client) + ')' +
   ' Peer IP='+Sockets3.GetPeerIPAddr(new_client) +
   ' Peer Port='+Sockets3.GetPeerPort(new_client));
end;

procedure TForm1.Sockets3DataAvailable(Sender: TObject; Socket: Integer);
var
 Buf: string;
begin
```

```
  Memo1.Lines.Add('Data recevied from socket(' + IntToStr(Socket) + ')');
  buf := Sockets3.Text;
  if buf = 'T' then
  begin
  Memo1.Lines.Add('**** Found T ****');
  T := true;
  Timer1.Interval := 2000;
  end;
  Memo1.Lines.Add('-->' + buf + '<--');
end;

procedure TForm1.Sockets3SessionConnected(Sender: TObject;
  Socket: Integer);
begin
  Memo1.Lines.Add('Connected to IP='+Sockets3.GetPeerIPAddr(sockets3.SocketNumber)+
   ' on his port #'+Sockets3.GetPeerPort(Sockets3.SocketNumber));
end;


procedure TForm1.Timer2Timer(Sender: TObject);
begin
 Timer2.Enabled := false;
 _lclose(OpenedFile);
 WavRecord;
end;

end.
```

```
unit MMIOWrap;

interface

  uses
    SysUtils, Windows, MMSystem;

  type
    FourChars = array[0..3] of char;

    eMMIO_error = class(Exception)
      MMIO_result : integer;
      constructor Create(
       const Err : integer;
       const Msg : string
       );
      end;

    tMMIO_file = class
      Handle : HMMIO;

      constructor Create;

      function Open(
       const Filename : string
       ) : HMMIO;

      function Close : MMRESULT;

      function Read(
       const Pointer_to_buffer : pchar;
       const Size_of_buffer    : UINT
       ) : longint;

      function Ascend(
       var Chunk_info : TMMCKInfo
       ) : MMRESULT;

      function Descend(
       var   Chunk_info,
           Parent_chunk_info : TMMCKInfo;
       const Flags           : UINT
       ) : MMRESULT;

      function First_descend(
```

```
    var   Chunk_info : TMMCKInfo;
    const Flags      : UINT
    ) : MMRESULT;

  end;

implementation

 constructor eMMIO_error.Create(
  const Err : integer;
  const Msg : string
  );
   begin
     inherited Create(Msg);
     MMIO_result := Err;
     end;

 constructor tMMIO_file.Create;
   begin
     inherited Create;
     Handle := 0;
     end;

 function tMMIO_file.Open(
  const Filename : string
  ) : HMMIO;
   begin
     result := mmioOpen(pchar(Filename),nil,MMIO_READ);
     Handle := result;
     if result = 0 then
       raise eMMIO_error.Create(result,
        'Error opening tMMIO_file with file "' + Filename + '"'
        );
     end;

 function tMMIO_file.Close : MMRESULT;
   begin
     result := mmioClose(Handle,0);
     if result <> 0 then
       raise eMMIO_error.Create(result,
        'Error closing tMMIO_file'
        );
     end;

 function tMMIO_file.Read(
```

```
  const Pointer_to_buffer : pchar;
  const Size_of_buffer    : UINT
  ) : longint;
   begin
     result := mmioRead(Handle,Pointer_to_buffer,Size_of_buffer);
     if result = 0 then
       raise eMMIO_error.Create(result,
        'Error reading tMMIO_file'
        );
     end;

function tMMIO_file.Ascend(
  var Chunk_info : TMMCKInfo
  ) : MMRESULT;
   begin
     result := mmioAscend(Handle,@Chunk_info,0);
     if result <> 0 then
       raise eMMIO_error.Create(result,
        'Error ascending in tMMIO_file'
        );
     end;

function tMMIO_file.Descend(
  var   Chunk_info,
     Parent_chunk_Info : TMMCKInfo;
  const Flags          : UINT
  ) : MMRESULT;
   begin
     result := mmioDescend(Handle,@Chunk_info,@Parent_chunk_info,Flags);
     if result <> 0 then
       raise eMMIO_error.Create(result,
        'Error descending in tMMIO_file'
        );
     end;

function tMMIO_file.First_descend(
  var   Chunk_info : TMMCKInfo;
  const Flags     : UINT
  ) : MMRESULT;
   begin
     result := mmioDescend(Handle,@Chunk_info,nil,Flags);
     if result <> 0 then
       raise eMMIO_error.Create(result,
        'Error doing first descend in tMMIO_file'
        );
```

```
        end;

{initialization}
    end.
```

```
unit WaveWrap;

interface

  uses
    SysUtils, Windows, MMSystem;

  const
    NUM_WAVE_FORMATS = 12;
    NUM_WAVE_FUNCTIONS = 7;

  type
    eWave_error = class(Exception)
      Wave_result : integer;
      constructor Create(
       const Err : integer;
       const Msg : string
       );
      end;

    tWave_device = class
      Handle : HWaveOut;
      Caps   : TWaveOutCaps;

      constructor Create;

      function GetOutCaps(
       const DeviceID: integer
       ) : MMRESULT;

      function Open(
       var   Format : TWaveFormatEx;
       const Flags  : DWORD
       ) : MMRESULT;

      function Close : MMRESULT;

      function Prepare_header(
       var   Header : TWaveHdr
       ) : MMRESULT;

      function Unprepare_header(
       var   Header : TWaveHdr
       ) : MMRESULT;
```

```
        function Write(
          var   Header : TWaveHdr
          ): MMRESULT;

        end;

    function WaveFormatString(
     const FormatConstant: UINT
     ): string;

    function WaveFunctionString(
     const FunctionConstant: UINT
     ): string;

implementation

    constructor eWave_error.Create(
     const Err : integer;
     const Msg : string
     );
      begin
        inherited Create(Msg);
        Wave_result := Err;
        end;

    constructor tWave_device.Create;
      begin

        inherited Create;
        Handle := 0;
        end;

    function tWave_device.GetOutCaps(
     const DeviceID: integer
     ) : MMRESULT;
      begin
      result := waveOutGetDevCaps(
              DeviceID,
              @Caps,
              SizeOf(Caps)
              );
      if result <> MMSYSERR_NOERROR then
        raise eWave_error.Create(result,
          'Unable to get device capabilities'
          );
```

```
  end;

function tWave_device.Open(
 var   Format : TWaveFormatEx;
 const Flags  : DWORD
 ) : MMRESULT;
  begin
    result := waveOutOpen(
              @Handle,
              WAVE_MAPPER,  { Let Windows choose device. }
              @Format,
              DWORD(nil),   { No callback. }
              0,            { No instance. }
              Flags
              );
    if result <> 0 then
      raise eWave_error.Create(result,
       'Error opening output wave device'
       );
    end;

function tWave_device.Close : MMRESULT;
  begin
    result := waveOutClose(Handle);
    if result <> 0 then
      raise eWave_error.Create(result,
       'Error closing output wave device'
       );
    end;

function tWave_device.Prepare_header(
 var   Header : TWaveHdr
 ) : MMRESULT;
  begin
    result := waveOutPrepareHeader(Handle,
              @Header,
              sizeof(Header)
              );
    if result <> 0 then
      raise eWave_error.Create(result,
       'Error preparing header for output wave device'
       );
    end;

function tWave_device.Unprepare_header(
```

```
  var   Header : TWaveHdr
  ) : MMRESULT;
   begin
      result := waveOutUnprepareHeader(Handle,
                @Header,
                sizeof(Header)
                );
      if result <> 0 then
         raise eWave_error.Create(result,
          'Error unpreparing header for output wave device'
          );
      end;

function tWave_device.Write(
  var   Header : TWaveHdr
  ): MMRESULT;
   begin
      result := waveOutWrite(Handle,
                @Header,
                sizeof(Header)
                );
      if result <> 0 then
         raise eWave_error.Create(result,
          'Error writing to output wave device'
          );
      end;

function WaveFormatString(
  const FormatConstant: UINT
  ): string;

  begin
    case FormatConstant of
       WAVE_FORMAT_1M08:
         result := '11.025 kHz. Mono. 8-bit';
       WAVE_FORMAT_1S08:
         result := '11.025 kHz. Stereo 8bit';
       WAVE_FORMAT_1M16:
         result := '11.025 kHz. Mono. 16-bit';
       WAVE_FORMAT_1S16:
         result := '11.025 kHz. Stereo 16-bit';
       WAVE_FORMAT_2M08:
         result := '22.05 kHz. Mono. 8-bit';
       WAVE_FORMAT_2S08:
         result := '22.05 kHz. Stereo 8-bit';
```

```
        WAVE_FORMAT_2M16:
          result := '22.05 kHz. Mono. 16-bit';
        WAVE_FORMAT_2S16:
          result := '22.05 kHz. Stereo 16-bit';
        WAVE_FORMAT_4M08:
          result := '44.1 kHz. Mono. 8-bit';
        WAVE_FORMAT_4S08:
          result := '44.1 kHz. Stereo 8-bit';
        WAVE_FORMAT_4M16:
          result := '44.1 kHz. Mono. 16-bit';
        WAVE_FORMAT_4S16:
          result := '44.1 kHz. Stereo 16-bit';
        else
          result := 'Unknown Format';
        end;
    end;

function WaveFunctionString(
  const FunctionConstant: UINT
  ): string;

  begin
    case FunctionConstant of
      WAVECAPS_PITCH:
        result := 'Pitch Control';
      WAVECAPS_PLAYBACKRATE:
        result := 'Playback Rate Control';
      WAVECAPS_VOLUME:
        result := 'Volume Control';
      WAVECAPS_LRVOLUME:
        result := 'Separate L/R Volume Control';
      WAVECAPS_SYNC:
        result := 'Syncronization';
      WAVECAPS_SAMPLEACCURATE:
        result := 'Sample Accurate';
      WAVECAPS_DIRECTSOUND:
        result := 'DirectSound!!!!';

      else
        result := 'Invalid Function';

      end;
    end;

{initialization}
```

end.

```
unit WavePlay;

interface

  uses  Windows, Messages, SysUtils,Stdctrls,Classes,MMSystem, MMIOWrap, WaveWrap ;

  type
    tAudio_waveform = class
      Header : TWaveHdr;
      Data   : pchar;
      Format : TWaveFormatEx;

      destructor Destroy; override;

      function Read(
       const Filename : string
       ) : boolean;
      procedure Mix;
      procedure Play;
      end;
  var
    temp : array[0..255] of char;
    len : Integer;
    buff : array[0..8191] of char;
    temp_file : integer;
    f:file;
    NumByte,fsize:longint;
implementation
  uses
   inifiles,server2, Forms, Dialogs;

destructor tAudio_waveform.Destroy;
    begin
      strDispose(Data);
       { ...which was set to nil automatically by
         Delphi, since it is a pointer field of
         a class structure. }
      end;

function tAudio_waveform.Read(
   const Filename : string
   ) : boolean;
    var
      Parent_chunk_info,
      Chunk_info       : TMMCKInfo;
```

```
begin
  result := false;
  strDispose(Data);
  Data := nil;
  with tMMIO_file.Create do
    try
      Open(filename);

      { Find the "WAVE" audio Parent Chunk. }
      FourChars(Parent_chunk_info.fccType) := 'WAVE';
      First_descend(Parent_chunk_info,MMIO_FINDRIFF);

      { Find the "fmt " format chunk. }
      FourChars(Chunk_info.ckid) := 'fmt ';
      Descend(Chunk_info,Parent_chunk_info,MMIO_FINDCHUNK);

      { Read PCM wave format record. }
      Read(@Format,Chunk_info.ckSize);
       { Here, we could instantiate a tWave_device
         and do a WAVE_FORMAT_QUERY to see if we are
         able to handle the wave format, and whether
         we should go any fuurther. }

      { Go back up to the WAVE level, and then find
        the "data" chunk. }
      Ascend(Chunk_info);
      FourChars(Chunk_info.ckid) := 'data';
      Descend(Chunk_info,Parent_chunk_info,MMIO_FINDCHUNK);

      { Allocate the wave data buffer, and read it in. }
      Data := strAlloc(Chunk_info.ckSize);
      Read(Data,Chunk_info.ckSize);


      { Fill in the Wave_header }
      with Header do begin
        lpData        := Data;
        dwBufferLength := Chunk_info.ckSize;
        dwFlags       := 0;
        dwLoops       := 0;
        end;
     result:= true;
    finally
     Free;
    end;
```

```
    end;

procedure tAudio_waveform.Mix;
  var
    Parent_chunk_info,
    Chunk_info      : TMMCKInfo;
  begin
  with tMMIO_file.Create do
    try
      Open('format.wav');

      { Find the "WAVE" audio Parent Chunk. }
      FourChars(Parent_chunk_info.fccType) := 'WAVE';
      First_descend(Parent_chunk_info,MMIO_FINDRIFF);

      { Find the "fmt " format chunk. }
      FourChars(Chunk_info.ckid) := 'fmt ';
      Descend(Chunk_info,Parent_chunk_info,MMIO_FINDCHUNK);

      { Read PCM wave format record. }
      Read(@Format,Chunk_info.ckSize);
       { Here, we could instantiate a tWave_device
         and do a WAVE_FORMAT_QUERY to see if we are
         able to handle the wave format, and whether
         we should go any fuurther. }
      { Allocate the wave data buffer, and read it in. }
      strPcopy(Temp,'Buffer.wav');
      temp_file := _lopen(Temp,0);
      fsize := _llseek(Temp_file,0,2);   (* get file's size to send *)
      Data:= stralloc(fsize);
      Form1.Memo1.Lines.Add('File size ' +IntTostr(fsize));
      _llseek(temp_file,0,0);
      while Numbyte > 0 do
      begin
      NumByte :=_lread(temp_file,@Data[0],8192);
      end;
      with Header do begin
        lpData        := Data;
        dwBufferLength := fsize;
        dwFlags       := 0;
        dwLoops       := 0;
        end;

      { Fill in the Wave_header }
      _lclose(Temp_file);
```

```
            finally
              Free;
            end;

        end;


procedure tAudio_waveform.Play;
    begin
      with tWave_device.Create do
        try
          Open(Format,WAVE_FORMAT_QUERY);
           { Die here if we can't handle the format. }
          Open(Format,0);
          Prepare_header(Header);
          Write(Header);
          while (Header.dwFlags and WHDR_DONE) = 0 do
            Application.ProcessMessages;
          Unprepare_header(Header);
          Close;
        finally
          Free;
        end;
      end;

{initialization}
  end.
```

```
unit Waveshow;
{ This is our treaming data transfer project wave file viewer
 to compare the result of decompressed sound and original sound }
interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, ExtCtrls, StdCtrls;

type
  TForm1 = class(TForm)
            ScrollBar1: TScrollBar;
            OpenDialog1: TOpenDialog;
    Button1: TButton;
            cmdExit: TButton;
            Panel1: TPanel;
            PaintBox1: TPaintBox;
    OpenDialog2: TOpenDialog;
    Button2: TButton;
    Label1: TLabel;
    Label2: TLabel;
            procedure Button1Click(Sender: TObject);
            procedure FormCreate(Sender: TObject);
            procedure cmdExitClick(Sender: TObject);
            procedure PaintBox1Paint(Sender: TObject);
            procedure ScrollBar1Change(Sender: TObject);
            procedure FormDestroy(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
            { Private declarations }
            File1IsOpen,File2IsOpen : Boolean;
            TheFileName1,TheFileName2 : String[79];
            WaveFile1,WaveFile2 : File;
            WaveBytes1,WaveBytes2 : LongInt;
            CurrPos1,CurrPos2 : LongInt;
            BytesPerSBPosition : Integer;
  public
            { Public declarations }
  end;

var
  Form1: TForm1;

implementation
```

```
{$R *.DFM}

const
        WaveHdrSize = 44;

var
  ReadBuffer1 : Array[0..1023] of Byte;
  ReadBuffer2 : Array[0..1023] of Byte;

procedure TForm1.FormCreate(Sender: TObject);
begin
        File1IsOpen := False;
    File2IsOpen := False;
        OpenDialog1.DefaultExt := 'wav';
        OpenDialog1.FileName := '*.wav';
        OpenDialog1.Filter := 'Wave files (*.wav)|*.wav';
    OpenDialog2.DefaultExt := 'wav';
        OpenDialog2.FileName := '*.wav';
        OpenDialog2.Filter := 'Wave files (*.wav)|*.wav';

end;

procedure TForm1.Button1Click(Sender: TObject);
begin
        if OpenDialog1.Execute then { Select wave File to plot with green line}
        begin
      If File1IsOpen then CloseFile (WaveFile1); { If already opened ,close and open  new file }
                try
                        TheFileName1 := OpenDialog1.Filename;
                        AssignFile (WaveFile1, TheFileName1);
            {I-}
                        Reset (WaveFile1, 1);
            {I+}
                        File1IsOpen := True;
            Label1.caption := TheFileName1;
                        WaveBytes1 := FileSize (WaveFile1) - WaveHdrSize;
                        BytesPerSBPosition := 1;
                        while (WaveBytes1 div BytesPerSBPosition) > 32767 do
                                inc(BytesPerSBPosition);
                        if WaveBytes1 div BytesPerSBPosition > PaintBox1.Width then
                        begin
                                ScrollBar1.Visible := True;
                                ScrollBar1.Max := WaveBytes1 div BytesPerSBPosition
                                                                - PaintBox1.Width;
                        end
```

```
                    else
                            ScrollBar1.Visible := False;
                    if ScrollBar1.Position <> 0 then
                            ScrollBar1.Position := 0
                    else
                            PaintBox1.Refresh;

            except
                    on E: EInOutError do
                            MessageDlg ('Unable to open ' + TheFileName1 + #10 + #13
                                                    + 'Error was ' + E.Message,
mtError, [mbOK], 0);
            end;
    end;
end;

procedure TForm1.PaintBox1Paint(Sender: TObject);
var
    X1,X2, NumRead1,NumRead2 : Integer;
    CurrPos2,CurrPos1 : LongInt;
begin
    if File1IsOpen then
    begin
            CurrPos1 := (ScrollBar1.Position * BytesPerSBPosition) + WaveHdrSize; { Matching cursor
position with position in wavefile}
            Seek (WaveFile1, CurrPos1);
            with PaintBox1.Canvas do
            begin
    FillChar (ReadBuffer1, SizeOf(ReadBuffer1), $80);
                    BlockRead (WaveFile1, ReadBuffer1, ClipRect.Right, NumRead1);
                    Pen.Color := clLime;   { Select color of graph ( light green)}
                    MoveTo (0, ReadBuffer1[0] shr 1); { Loop to write the line}
                    for X1 := 1 to NumRead1 - 1 do
                            LineTo (x1, ReadBuffer1[X1] shr 1); {Line between sample value}
            end;
    end;
    if File2IsOpen then
        begin
            CurrPos2 := (ScrollBar1.Position * BytesPerSBPosition) + WaveHdrSize; { Matching cursor
position with position in wavefile}
            Seek (WaveFile2, CurrPos2);
            with PaintBox1.Canvas do
            begin
    FillChar (ReadBuffer2, SizeOf(ReadBuffer2), $80);
                    BlockRead (WaveFile2, ReadBuffer2, ClipRect.Right, NumRead2);
```

```
                    Pen.Color := clRed; { Select color of graph ( red )}
                    MoveTo (0, ReadBuffer2[0] shr 1);
                    for X2 := 1 to NumRead2 - 1 do { Loop to write the line}
                        LineTo (x2, ReadBuffer2[X2] shr 1); {Line between sample value}
                end;
            end;


end;

procedure TForm1.ScrollBar1Change(Sender: TObject);
begin
        PaintBox1.Refresh;
end;

procedure TForm1.cmdExitClick(Sender: TObject);
begin
  Application.Terminate;
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
    if File1IsOpen then CloseFile (WaveFile1);
        if File2IsOpen then CloseFile (WaveFile2);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
        if OpenDialog2.Execute then { Select wave File to plot with green line}
        begin
            If File2IsOpen then CloseFile (WaveFile2); { If already opened ,close and open  new file }
                try
                    TheFileName2 := OpenDialog2.Filename;
                    AssignFile (WaveFile2, TheFileName2);
            {I-}
                    Reset (WaveFile2, 1);
            {I+}
                    File2IsOpen := True;
            Label2.caption := TheFileName2;
                    WaveBytes2 := FileSize (WaveFile2) - WaveHdrSize;
                    BytesPerSBPosition := 1;
```

```
                while (WaveBytes2 div BytesPerSBPosition) > 32767 do
                        inc(BytesPerSBPosition);
                if WaveBytes2 div BytesPerSBPosition > PaintBox1.Width then
                begin
                        ScrollBar1.Visible := True;
                        ScrollBar1.Max := WaveBytes2 div BytesPerSBPosition
                                                                - PaintBox1.Width;
                end
                else
                        ScrollBar1.Visible := False;
                if ScrollBar1.Position <> 0 then
                        ScrollBar1.Position := 0
                else
                        PaintBox1.Refresh;

        except
                on E: EInOutError do
                        MessageDlg ('Unable to open ' + TheFileName2 + #10 + #13
                                                + 'Error was ' + E.Message,
mtError, [mbOK], 0);
        end;
    end;
end;

end.
```

```
unit ComDrv32;

interface

uses
  Windows, Messages, SysUtils, Classes, Forms;

type
  // COM Port Baud Rates
  TComPortBaudRate = ( br110, br300, br600, br1200, br2400, br4800,
                       br9600, br14400, br19200, br38400, br56000,
                       br57600, br115200{v1.02: removed ->, br128000, br256000} );
  // COM Port Numbers
  TComPortNumber = ( pnCOM1, pnCOM2, pnCOM3, pnCOM4 );
  // COM Port Data bits
  TComPortDataBits = ( db5BITS, db6BITS, db7BITS, db8BITS );
  // COM Port Stop bits
  TComPortStopBits = ( sb1BITS, sb1HALFBITS, sb2BITS );
  // COM Port Parity
  TComPortParity = ( ptNONE, ptODD, ptEVEN, ptMARK, ptSPACE );
  // COM Port Hardware Handshaking
  TComPortHwHandshaking = ( hhNONE, hhRTSCTS );
  // COM Port Software Handshaing
  TComPortSwHandshaking = ( shNONE, shXONXOFF );

  TComPortReceiveDataEvent = procedure( Sender: TObject; DataPtr: pointer; DataSize: integer ) of object;

  TCommPortDriver = class(TComponent)
  protected
    FComPortHandle          : THANDLE; // COM Port Device Handle

    FComPort                : TComPortNumber; // COM Port to use (1..4)
    FComPortBaudRate        : TComPortBaudRate; // COM Port speed (brXXXX)
    FComPortDataBits        : TComPortDataBits; // Data bits size (5..8)
    FComPortStopBits        : TComPortStopBits; // How many stop bits to use (1,1.5,2)
    FComPortParity          : TComPortParity; // Type of parity to use (none,odd,even,mark,space)
    FComPortHwHandshaking   : TComPortHwHandshaking; // Type of hw handshaking to use
    FComPortSwHandshaking   : TComPortSwHandshaking; // Type of sw handshaking to use
    FComPortInBufSize       : word; // Size of the input buffer
    FComPortOutBufSize      : word; // Size of the output buffer
    FComPortReceiveData     : TComPortReceiveDataEvent; // Event to raise on data reception
    FComPortPollingDelay    : word; // ms of delay between COM port pollings
    FEnableDTROnOpen        : boolean; { enable/disable DTR line on connect }
    FOutputTimeout          : word; { output timeout - milliseconds }
    FNotifyWnd              : HWND; // This is used for the timer
```

```
    FTempInBuffer           : pointer;

    procedure SetComHandle( Value: THANDLE );
    procedure SetComPort( Value: TComPortNumber );
    procedure SetComPortBaudRate( Value: TComPortBaudRate );
    procedure SetComPortDataBits( Value: TComPortDataBits );
    procedure SetComPortStopBits( Value: TComPortStopBits );
    procedure SetComPortParity( Value: TComPortParity );
    procedure SetComPortHwHandshaking( Value: TComPortHwHandshaking );
    procedure SetComPortSwHandshaking( Value: TComPortSwHandshaking );
    procedure SetComPortInBufSize( Value: word );
    procedure SetComPortOutBufSize( Value: word );
    procedure SetComPortPollingDelay( Value: word );

    procedure ApplyCOMSettings;

    procedure TimerWndProc( var msg: TMessage );
  public
    constructor Create( AOwner: TComponent ); override;
    destructor Destroy; override;

    function Connect: boolean;
    procedure Disconnect;
    function Connected: boolean;
    { v1.02: flushes the rx/tx buffers }
    procedure FlushBuffers( inBuf, outBuf: boolean );
    { v1.02: returns the output buffer free space or 65535 if
          not connected }
    function OutFreeSpace: word;

    { Send data }
    { v1.02: changed result time from 'boolean' to 'integer'. See the docs
          for more info }
    function SendData( DataPtr: pointer; DataSize: integer ): integer;
    // Send a pascal string (NULL terminated if $H+ (default))
    function SendString( s: string ): boolean;
    // v1.02: send a C-style strings (NULL terminated)
    function SendZString( s: pchar ): boolean;
    // v1.02: set DTR line high (onOff=TRUE) or low (onOff=FALSE).
    //      You must not use HW handshaking.
    procedure ToggleDTR( onOff: boolean );
    // v1.02: set RTS line high (onOff=TRUE) or low (onOff=FALSE).
    //      You must not use HW handshaking.
    procedure ToggleRTS( onOff: boolean );
```

```
    // v1.02: make the Handle to the com port public (for TAPI...)
    property ComHandle: THANDLE read FComPortHandle write SetComHandle;
  published
    // Which COM Port to use
    property ComPort: TComPortNumber read FComPort write SetComPort default pnCOM2;
    // COM Port speed (bauds)
    property ComPortSpeed: TComPortBaudRate read FComPortBaudRate write SetComPortBaudRate default
br9600;
    // Data bits to used (5..8, for the 8250 the use of 5 data bits with 2 stop bits is an invalid combination,
    // as is 6, 7, or 8 data bits with 1.5 stop bits)
    property ComPortDataBits: TComPortDataBits read FComPortDataBits write SetComPortDataBits default
db8BITS;
    // Stop bits to use (1, 1.5, 2)
    property ComPortStopBits: TComPortStopBits read FComPortStopBits write SetComPortStopBits default
sb1BITS;
    // Parity Type to use (none,odd,even,mark,space)
    property ComPortParity: TComPortParity read FComPortParity write SetComPortParity default ptNONE;
    // Hardware Handshaking Type to use:
    // cdNONE        no handshaking
    // cdCTSRTS      both cdCTS and cdRTS apply (** this is the more common method**)
    property ComPortHwHandshaking: TComPortHwHandshaking
          read FComPortHwHandshaking write SetComPortHwHandshaking default hhNONE;
    // Software Handshaking Type to use:
    // cdNONE        no handshaking
    // cdXONXOFF     XON/XOFF handshaking
    property ComPortSwHandshaking: TComPortSwHandshaking
          read FComPortSwHandshaking write SetComPortSwHandshaking default shNONE;
    // Input Buffer size
    property ComPortInBufSize: word read FComPortInBufSize write SetComPortInBufSize default 2048;
    // Output Buffer size
    property ComPortOutBufSize: word read FComPortOutBufSize write SetComPortOutBufSize default 2048;
    // ms of delay between COM port pollings
    property ComPortPollingDelay: word read FComPortPollingDelay write SetComPortPollingDelay default 50;
    // v1.02: Set to TRUE to enable DTR line on connect and to leave it on until disconnect.
    //       Set to FALSE to disable DTR line on connect.
    property EnableDTROnOpen: boolean read FEnableDTROnOpen write FEnableDTROnOpen default true;
    // v1.02: Output timeout (milliseconds)
    property OutputTimeout: word read FOutputTimeOut write FOutputTimeout default 4000;
    // Event to raise when there is data available (input buffer has data)
    property OnReceiveData: TComPortReceiveDataEvent read FComPortReceiveData write
FComPortReceiveData;
  end;

procedure Register;
```

```
implementation

constructor TCommPortDriver.Create( AOwner: TComponent );
begin
 inherited Create( AOwner );
 // Initialize to default values
 FComPortHandle          := 0;     // Not connected
 FComPort                := pnCOM2; // COM 2
 FComPortBaudRate        := br9600; // 9600 bauds
 FComPortDataBits        := db8BITS; // 8 data bits
 FComPortStopBits        := sb1BITS; // 1 stop bit
 FComPortParity          := ptNONE; // no parity
 FComPortHwHandshaking   := hhNONE; // no hardware handshaking
 FComPortSwHandshaking   := shNONE; // no software handshaking
 FComPortInBufSize       := 2048;   // input buffer of 2048 bytes
 FComPortOutBufSize      := 2048;   // output buffer of 2048 bytes
 FComPortReceiveData     := nil;    // no data handler
 FComPortPollingDelay    := 50;     // poll COM port every 50ms
 FOutputTimeout          := 4000;   // output timeout - 4000ms
 FEnableDTROnOpen        := true;   // DTR high on connect
 // Temporary buffer for received data
 GetMem( FTempInBuffer, FComPortInBufSize );
 // Allocate a window handle to catch timer's notification messages
 if not (csDesigning in ComponentState) then
  FNotifyWnd := AllocateHWnd( TimerWndProc );
end;

destructor TCommPortDriver.Destroy;
begin
 // Be sure to release the COM device
 Disconnect;
 // Free the temporary buffer
 FreeMem( FTempInBuffer, FComPortInBufSize );
 // Destroy the timer's window
 DeallocateHWnd( FNotifyWnd );
 inherited Destroy;
end;

// v1.02: The COM port handle made public and writeable.
// This lets you connect to external opened com port.
// Setting ComPortHandle to 0 acts as Disconnect.
procedure TCommPortDriver.SetComHandle( Value: THANDLE );
begin
 // If same COM port then do nothing
 if FComPortHandle = Value then
```

```
    exit;
  { If value is $FFFFFFFF then stop controlling the COM port
    without closing in }
  if Value = $FFFFFFFF then
  begin
    if Connected then
      { Stop the timer }
      if Connected then
        KillTimer( FNotifyWnd, 1 );
    { No more connected }
    FComPortHandle := 0;
  end
  else
  begin
    { Disconnect }
    Disconnect;
    { If Value is = 0 then exit now }
    { (ComPortHandle := 0 acts as Disconnect) }
    if Value = 0  then
      exit;

    { Set COM port handle }
    FComPortHandle := Value;

    { Start the timer (used for polling) }
    SetTimer( FNotifyWnd, 1, FComPortPollingDelay, nil );
  end;
end;

procedure TCommPortDriver.SetComPort( Value: TComPortNumber );
begin
  // Be sure we are not using any COM port
  if Connected then
    exit;
  // Change COM port
  FComPort := Value;
end;

procedure TCommPortDriver.SetComPortBaudRate( Value: TComPortBaudRate );
begin
  // Set new COM speed
  FComPortBaudRate := Value;
  // Apply changes
  if Connected then
    ApplyCOMSettings;
```

```
end;

procedure TCommPortDriver.SetComPortDataBits( Value: TComPortDataBits );
begin
  // Set new data bits
  FComPortDataBits := Value;
  // Apply changes
  if Connected then
    ApplyCOMSettings;
end;

procedure TCommPortDriver.SetComPortStopBits( Value: TComPortStopBits );
begin
  // Set new stop bits
  FComPortStopBits := Value;
  // Apply changes
  if Connected then
    ApplyCOMSettings;
end;

procedure TCommPortDriver.SetComPortParity( Value: TComPortParity );
begin
  // Set new parity
  FComPortParity := Value;
  // Apply changes
  if Connected then
    ApplyCOMSettings;
end;

procedure TCommPortDriver.SetComPortHwHandshaking( Value: TComPortHwHandshaking );
begin
  // Set new hardware handshaking
  FComPortHwHandshaking := Value;
  // Apply changes
  if Connected then
    ApplyCOMSettings;
end;

procedure TCommPortDriver.SetComPortSwHandshaking( Value: TComPortSwHandshaking );
begin
  // Set new software handshaking
  FComPortSwHandshaking := Value;
  // Apply changes
  if Connected then
    ApplyCOMSettings;
```

```
end;

procedure TCommPortDriver.SetComPortInBufSize( Value: word );
begin
 { Do nothing if connected }
 if Connected then
  exit;
 // Free the temporary input buffer
 FreeMem( FTempInBuffer, FComPortInBufSize );
 // Set new input buffer size
 FComPortInBufSize := Value;
 // Allocate the temporary input buffer
 GetMem( FTempInBuffer, FComPortInBufSize );
end;

procedure TCommPortDriver.SetComPortOutBufSize( Value: word );
begin
 { Do nothing if connected }
 if not Connected then
  exit;
 // Set new output buffer size
 FComPortOutBufSize := Value;
end;

procedure TCommPortDriver.SetComPortPollingDelay( Value: word );
begin
 // If new delay is not equal to previous value...
 if Value <> FComPortPollingDelay then
 begin
  // Stop the timer
  if Connected then
   KillTimer( FNotifyWnd, 1 );
  // Store new delay value
  FComPortPollingDelay := Value;
  // Restart the timer
  if Connected then
   SetTimer( FNotifyWnd, 1, FComPortPollingDelay, nil );
 end;
end;

const
 Win32BaudRates: array[br110..br115200] of DWORD =
  ( CBR_110, CBR_300, CBR_600, CBR_1200, CBR_2400, CBR_4800, CBR_9600,
   CBR_14400, CBR_19200, CBR_38400, CBR_56000, CBR_57600, CBR_115200{v1.02 removed: CRB_128000,
CBR_256000} );
```

```
const
  dcb_Binary            = $00000001;
  dcb_ParityCheck       = $00000002;
  dcb_OutxCtsFlow       = $00000004;
  dcb_OutxDsrFlow       = $00000008;
  dcb_DtrControlMask    = $00000030;
    dcb_DtrControlDisable   = $00000000;
    dcb_DtrControlEnable    = $00000010;
    dcb_DtrControlHandshake = $00000020;
  dcb_DsrSensivity      = $00000040;
  dcb_TXContinueOnXoff  = $00000080;
  dcb_OutX              = $00000100;
  dcb_InX               = $00000200;
  dcb_ErrorChar         = $00000400;
  dcb_NullStrip         = $00000800;
  dcb_RtsControlMask    = $00003000;
    dcb_RtsControlDisable   = $00000000;
    dcb_RtsControlEnable    = $00001000;
    dcb_RtsControlHandshake = $00002000;
    dcb_RtsControlToggle    = $00003000;
  dcb_AbortOnError      = $00004000;
  dcb_Reserveds         = $FFFF8000;

// Apply COM settings.
procedure TCommPortDriver.ApplyCOMSettings;
var dcb: TDCB;
begin
  // Do nothing if not connected
  if not Connected then
    exit;

  // Clear all
  fillchar( dcb, sizeof(dcb), 0 );
  // Setup dcb (Device Control Block) fields
  dcb.DCBLength := sizeof(dcb); // dcb structure size
  dcb.BaudRate := Win32BaudRates[ FComPortBaudRate ]; // baud rate to use
  // Set fBinary: Win32 does not support non binary mode transfers
  // (also disable EOF check)
  dcb.Flags := dcb_Binary;
  if EnableDTROnOpen then
    { Enabled the DTR line when the device is opened and leaves it on }
    dcb.Flags := dcb.Flags or dcb_DtrControlEnable;

  case FComPortHwHandshaking of // Type of hw handshaking to use
```

```
    hhNONE:; // No hardware handshaking
    hhRTSCTS: // RTS/CTS (request-to-send/clear-to-send) hardware handshaking
      dcb.Flags := dcb.Flags or dcb_OutxCtsFlow or dcb_RtsControlHandshake;
   end;
   case FComPortSwHandshaking of // Type of sw handshaking to use
    shNONE:; // No software handshaking
    shXONXOFF: // XON/XOFF handshaking
      dcb.Flags := dcb.Flags or dcb_OutX or dcb_InX;
   end;
   dcb.XONLim := FComPortInBufSize div 4; // Specifies the minimum number of bytes allowed
                                          // in the input buffer before the XON character is sent
                                          // (or CTS is set)
   dcb.XOFFLim := 1; // Specifies the maximum number of bytes allowed in the input buffer
                     // before the XOFF character is sent. The maximum number of bytes
                     // allowed is calculated by subtracting this value from the size,
                     // in bytes, of the input buffer
   dcb.ByteSize := 5 + ord(FComPortDataBits); // how many data bits to use
   dcb.Parity := ord(FComPortParity); // type of parity to use
   dcb.StopBits := ord(FComPortStopbits); // how many stop bits to use
   dcb.XONChar := #17; // XON ASCII char
   dcb.XOFFChar := #19; // XOFF ASCII char
   SetCommState( FComPortHandle, dcb );
   { Flush buffers }
   FlushBuffers( true, true );
   // Setup buffers size
   SetupComm( FComPortHandle, FComPortInBufSize, FComPortOutBufSize );
end;

function TCommPortDriver.Connect: boolean;
var comName: array[0..4] of char;
    tms: TCOMMTIMEOUTS;
begin
  // Do nothing if already connected
  Result := Connected;
  if Result then
    exit;
  // Open the COM port
  StrPCopy( comName, 'COM' );
  comName[3] := chr( ord('1') + ord(FComPort) );
  comName[4] := #0;
  FComPortHandle := CreateFile(
                        comName,
                        GENERIC_READ or GENERIC_WRITE,
                        0, // Not shared
                        nil, // No security attributes
```

```
                    OPEN_EXISTING,
                    FILE_ATTRIBUTE_NORMAL,
                    0 // No template
                   );
  Result := Connected;
  if not Result then
    exit;
  // Apply settings
  ApplyCOMSettings;
  // Setup timeouts: we disable timeouts because we are polling the com port!
  tms.ReadIntervalTimeout := 1; // Specifies the maximum time, in milliseconds,
                    // allowed to elapse between the arrival of two
                    // characters on the communications line
  tms.ReadTotalTimeoutMultiplier := 0; // Specifies the multiplier, in milliseconds,
                      // used to calculate the total time-out period
                      // for read operations.
  tms.ReadTotalTimeoutConstant := 1; // Specifies the constant, in milliseconds,
                     // used to calculate the total time-out period
                     // for read operations.
  tms.WriteTotalTimeoutMultiplier := 0; // Specifies the multiplier, in milliseconds,
                      // used to calculate the total time-out period
                      // for write operations.
  tms.WriteTotalTimeoutConstant := 0; // Specifies the constant, in milliseconds,
                     // used to calculate the total time-out period
                     // for write operations.
  SetCommTimeOuts( FComPortHandle, tms );
  // Start the timer (used for polling)
  SetTimer( FNotifyWnd, 1, FComPortPollingDelay, nil );
end;

procedure TCommPortDriver.Disconnect;
begin
  if Connected then
  begin
    // Stop the timer (used for polling)
    KillTimer( FNotifyWnd, 1 );
    // Release the COM port
    CloseHandle( FComPortHandle );
    // No more connected
    FComPortHandle := 0;
  end;
end;

function TCommPortDriver.Connected: boolean;
begin
```

```
  Result := FComPortHandle > 0;
end;

// v1.02: flish rx/rx buffers
procedure TCommPortDriver.FlushBuffers( inBuf, outBuf: boolean );
var dwAction: DWORD;
begin
  if not Connected then
    exit;
  // Flush the incoming data buffer
  dwAction := 0;
  if outBuf then
    dwAction := dwAction or PURGE_TXABORT or PURGE_TXCLEAR;
  if inBuf then
    dwAction := dwAction or PURGE_RXABORT or PURGE_RXCLEAR;
  PurgeComm( FComPortHandle, dwAction );
end;

// v1.02: returns the output buffer free space or 65535 if
//       not connected }
function TCommPortDriver.OutFreeSpace: word;
var stat: TCOMSTAT;
    errs: DWORD;
begin
  if not Connected then
    Result := 65535
  else
  begin
    ClearCommError( FComPortHandle, errs, @stat );
    Result := FComPortOutBufSize - stat.cbOutQue;
  end;
end;

// Send data
{function TCommPortDriver.SendData( DataPtr: pointer; DataSize: integer ): boolean;
var nsent: DWORD;
begin
  Result := WriteFile( FComPortHandle, DataPtr^, DataSize, nsent, nil );
  Result := Result and (nsent=DataSize);
end;}

{ Send data (breaks the data in small packets if it doesn't fit in the output
  buffer) }
function TCommPortDriver.SendData( DataPtr: pointer; DataSize: integer ): integer;
var nToSend, nsent: integer;
```

```
    t1: longint;
  begin
   { 0 bytes sent }
   Result := 0;
   { Do nothing if not connected }
   if not Connected then
    exit;
   { Current time }
   t1 := GetTickCount;
   { Loop until all data sent or timeout occurred }
   while DataSize > 0 do
   begin
    { Get output buffer free space }
    nToSend := OutFreeSpace;
    { If output buffer has some free space... }
    if nToSend > 0 then
    begin
     { Don't send more bytes than we actually have to send }
     if nToSend > DataSize then
      nToSend := DataSize;
     { Send }
     WriteFile( FComPortHandle, DataPtr^, DataSize, nsent, nil );
     { Update number of bytes sent }
     Result := Result + abs(nsent);
     { Decrease the count of bytes to send }
     DataSize := DataSize - abs(nsent);
     { Get current time }
     t1 := GetTickCount;
     { Continue. This skips the time check below (don't stop
       trasmitting if the FOutputTimeout is set too low) }
     continue;
    end;
    { Buffer is full. If we are waiting too long then
      invert the number of bytes sent and exit }
    if (GetTickCount-t1) > FOutputTimeout then
    begin
     Result := -Result;
     exit;
    end;
   end;
  end;

  // Send a pascal string (NULL terminated if $H+ (default))
  function TCommPortDriver.SendString( s: string ): boolean;
  var len: integer;
```

```
begin
  len := length( s );
  {$IFOPT H+}
  // New syle pascal string (NULL terminated)
  Result := SendData( pchar(s), len ) = len;
  {$ELSE}
  // Old style pascal string (s[0] = length)
  Result := SendData( pchar(@s[1]), len ) = len;
  {$ENDIF}
end;

// v1.02: send a C-style strings (NULL terminated)
function TCommPortDriver.SendZString( s: pchar ): boolean;
var len: integer;
begin
  len := strlen( s );
  Result := SendData( s, len ) = len;
end;

// v1.02: set DTR line high (onOff=TRUE) or low (onOff=FALSE).
//       You must not use HW handshaking.
procedure TCommPortDriver.ToggleDTR( onOff: boolean );
const funcs: array[boolean] of integer = (CLRDTR,SETDTR);
begin
  if Connected then
    EscapeCommFunction( FComPortHandle, funcs[onOff] );
end;

// v1.02: set RTS line high (onOff=TRUE) or low (onOff=FALSE).
//       You must not use HW handshaking.
procedure TCommPortDriver.ToggleRTS( onOff: boolean );
const funcs: array[boolean] of integer = (CLRRTS,SETRTS);
begin
  if Connected then
    EscapeCommFunction( FComPortHandle, funcs[onOff] );
end;

// COM port polling proc
procedure TCommPortDriver.TimerWndProc( var msg: TMessage );
var nRead: dword;
begin
  if (msg.Msg = WM_TIMER) and Connected then
  begin
    nRead := 0;
    if ReadFile( FComPortHandle, FTempInBuffer^, FComPortInBufSize, nRead, nil ) then
```

```
    if (nRead <> 0) and Assigned(FComPortReceiveData) then
      FComPortReceiveData( Self, FTempInBuffer, nRead );
  end;
end;

procedure Register;
begin
  { Register this component and show it in the 'System' tab
    of the component palette }
  RegisterComponents('System', [TCommPortDriver]);
end;

end.
```

```
unit Sockets;
interface

uses
 SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
 Forms, Dialogs, WinSock;
const
 { User Windows Messages }
 WM_ASYNCSELECT = WM_USER + 0;
type
 TDataAvailable = procedure (Sender: TObject; Socket: TSocket) of object;
 TDataNeeded = procedure (Sender: TObject; Socket: TSocket) of object;
 TSessionClosed = procedure (Sender: TObject; Socket: TSocket) of object;
 TSessionAvailable = procedure (Sender: TObject; Socket: TSocket) of object;
 TSessionConnected = procedure (Sender: TObject; Socket: TSocket) of object;
 TErrorOccurred = procedure (Sender: TObject; Socket: Integer; Error: integer; Msg: string) of object;

 TSockets = class(TWinControl)
 private
  Pse: PServEnt;
  Phe: PHostEnt;
  Ppe: PProtoEnt;
  sin: TSockAddrIn;
  initdata: TWSAData;
  FAuthorized: Boolean;
  FPort: String;
  FIPAddr: String;
  FSocket: TSocket;
  FMSocket: TSocket;
  FMode: longint;
  FTimeout: integer;
  FMaximumReceiveLength: integer;
  FDataAvailable: TDataAvailable;
  FDataNeeded : TDataNeeded;
  FSessionClosed: TSessionClosed;
  FSessionAvailable: TSessionAvailable;
  FSessionConnected: TSessionConnected;
  FErrorOccurred: TErrorOccurred;
  procedure SetText(Text: string);
  function GetText : string;
  procedure SetTextOOB(Text: string);
  function GetTextOOB : string;
  function PeekData : string;
  function SocketErrorDesc(error: integer) : string;
  procedure SocketError(Socket: TSocket; sockfunc: string; error: integer);
```

```
    procedure TWMPaint(var msg:TWMPaint); message WM_PAINT;
    procedure SetTimeout;
    procedure ResetTimeout;
    function GetLocalHostName: string;
  protected
    procedure WMASyncSelect(var msg: TMessage); message WM_ASYNCSELECT;
    procedure WMTimer(var msg: TMessage); message WM_TIMER;
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
    { I'd like to call these methods Connect, Close, Listen, etc but
      they would conflict with the WSock32.DLL function names ! }
    procedure SConnect;
    procedure SClose;
    procedure SListen;
    procedure SCancelListen;
    function SAccept: TSocket;
    function SReceive(Socket: TSocket; szBuff: PChar; var rlen: integer): integer;
    function SSend(Socket: TSocket; szBuff: PChar; var slen: integer): integer;
    function GetIPAddr(aSocket: TSocket): string;
    function GetLocalIPAddr: string;
    function GetPort(aSocket: TSocket): string;
    function GetPeerIPAddr(aSocket: TSocket): string;
    function GetPeerPort(aSocket: TSocket): string;
    function GetBlocking: Boolean;
    procedure SetBlocking(flag: Boolean);
    property Text: string read GetText write SetText;
    property Authorized: Boolean read FAuthorized write FAuthorized;
    property Peek: string read PeekData;
    property OOB: string read GetTextOOB write SetTextOOB;
    property SocketNumber: TSocket read FSocket write FSocket;
    property MasterSocket: TSocket read FMSocket write FMSocket;
    property HostName: string read GetLocalHostName;
  published
    property MaximumReceiveLength: integer read FMaximumReceiveLength write FMaximumReceiveLength;
    property IPAddr: string read FIPAddr write FIPAddr;
    property Port: string read FPort write FPort;
    property NonBlocking: Boolean read GetBlocking write SetBlocking default True;
    property Timeout: integer read FTimeout write FTimeout default 30;
    property OnDataAvailable: TDataAvailable read FDataAvailable
      write FDataAvailable;
    property OnDataNeeded: TDataNeeded read FDataNeeded
      write FDataNeeded;
    property OnSessionClosed: TSessionClosed read FSessionClosed
      write FSessionClosed;
```

```
    property OnSessionAvailable: TSessionAvailable read FSessionAvailable
      write FSessionAvailable;
    property OnSessionConnected: TSessionConnected read FSessionConnected
      write FSessionConnected;
    property OnErrorOccurred: TErrorOccurred read FErrorOccurred
      write FErrorOccurred;
  end;

procedure Register;

implementation


procedure Register;
begin
  RegisterComponents('Samples', [TSockets]);
end;


constructor TSockets.Create(AOwner: TComponent);
var
  iStatus: integer;
begin
  inherited Create(AOwner);
  FAuthorized := False;
  FMode := 1;
  FTimeout := 30;
  FMaximumReceiveLength := 8192;
  FSocket := INVALID_SOCKET;
  FMSocket := INVALID_SOCKET;
  iStatus := WSAStartup($101,initdata);
  if iStatus <> 0 then
    SocketError(0,'Constructor (WSAStartup)',WSAGetLastError);
  invalidate;
end;

destructor TSockets.Destroy;
var
  iStatus: integer;
begin
  iStatus := WSACleanup;
  if iStatus < 0 then
    SocketError(INVALID_SOCKET,'Destructor (WSACleanup)',WSAGetLastError);
  inherited Destroy;
end;
```

```
procedure TSockets.TWMPaint(var msg: TWMPaint);
var
 icon: HIcon;
 dc: HDC;
begin
 if csDesigning in ComponentState then
 begin
  icon := LoadIcon(HInstance,MAKEINTRESOURCE('TSOCKETS'));
  dc := GetDC(Handle);
  Width := 32;
  Height := 32;
  DrawIcon(dc,0,0,icon);
  ReleaseDC(Handle,dc);
  FreeResource(icon);
 end;
 ValidateRect(Handle,nil);
end;

function TSockets.GetBlocking: Boolean;
begin
 if FMode = 1 then
  Result := True
 else
  Result := False;
end;

procedure TSockets.SetBlocking(flag: Boolean);
begin
 if flag then
  FMode := 1
 else
  FMode := 0;
end;

procedure TSockets.SetText(Text: string);
var
 BytesSent: integer;
 pBuff: PChar;
begin
 pBuff := StrAlloc(Length(Text)+1);
 StrPCopy(pBuff,Text);
 if FMode = 0 then
  SetTimeout;
 BytesSent := send(FSocket,pBuff^,Length(Text),0);
```

```
  if FMode = 0 then
    ResetTimeout;
  if BytesSent < 0 then
    SocketError(FSocket,'SetText (Send)',WSAGetLastError);
  StrDispose(pBuff);
end;

function TSockets.GetText: string;
var
  len: integer;
  pBuff: PChar;
begin
  if FSocket <> INVALID_SOCKET then
  begin
    pBuff := StrAlloc(FMaximumReceiveLength);
    if FMode = 0 then
      SetTimeout;
    len := recv(FSocket,pBuff^,FMaximumReceiveLength,0);
    if FMode = 0 then
      ResetTimeout;
    if len < 0 then
      SocketError(FSocket,'GetText (Recv)',WSAGetLastError);
    pBuff[len] := chr(0);
    Result := pBuff;
    StrDispose(pBuff);
  end
  else Result := '';
end;

procedure TSockets.SetTextOOB(Text: string);
var
  BytesLeft, BytesSent: integer;
  pBuff: PChar;
begin
  pBuff := StrAlloc(Length(Text)+1);
  StrPCopy(pBuff,Text);
  if FMode = 0 then
    SetTimeout;
  BytesSent := send(FSocket,pBuff^,Length(Text),MSG_OOB);
  if FMode = 0 then
    ResetTimeout;
  if BytesSent < 0 then
    SocketError(FSocket,'SetText (Send)',WSAGetLastError);
  StrDispose(pBuff);
end;
```

```
function TSockets.GetTextOOB: string;
var
 len: integer;
 pBuff: PChar;
begin
 if FSocket <> INVALID_SOCKET then
 begin
  pBuff := StrAlloc(FMaximumReceiveLength);
  if FMode = 0 then
   SetTimeout;
  len := recv(FSocket,pBuff^,FMaximumReceiveLength,MSG_OOB);
  if FMode = 0 then
   ResetTimeout;
  if len < 0 then
   SocketError(FSocket,'GetText (Recv)',WSAGetLastError);
  Result := pBuff;
  StrDispose(pBuff);
 end
 else Result := '';
end;


function TSockets.PeekData: string;
var
 len: integer;
 pBuff: PChar;
begin
 if FSocket <> INVALID_SOCKET then
 begin
  pBuff := StrAlloc(FMaximumReceiveLength);
  if FMode = 0 then
   SetTimeout;
  len := recv(FSocket,pBuff^,FMaximumReceiveLength,MSG_PEEK);
  if FMode = 0 then
   ResetTimeout;
  if len < 0 then
   SocketError(FSocket,'PeekData (Peek)',WSAGetLastError);
  Result := pBuff;
  StrDispose(pBuff);
 end
 else Result := '';
end;

function TSockets.GetPort(aSocket: TSocket): string;
```

```
var
  addr: TSockAddrIn;
  addrlen: integer;
begin
  addrlen := sizeof(addr);
  getsockname(aSocket,addr,addrlen);
  Result := IntToStr(ntohs(addr.sin_port));
end;

function TSockets.GetIPAddr(aSocket: TSocket): string;
var
  addr: TSockAddrIn;
  addrlen: integer;
  szIPAddr: PChar;
begin
  addrlen := sizeof(addr);
  getsockname(aSocket,addr,addrlen);
  szIPAddr := inet_ntoa(addr.sin_addr);
  Result := StrPas(szIPAddr);
end;

function TSockets.GetLocalIPAddr: string;
var
  addr: TSockAddrIn;
  Phe: PHostEnt;
  szHostName: array[0..128] of char;
begin
  GetHostName(szHostName,128);
  Phe := GetHostByName(szHostName);
  if Phe = nil then
    Result := '0.0.0.0'
  else
  begin
    addr.sin_addr.S_addr := longint(plongint(Phe^.h_addr_list^)^);
    Result := inet_ntoa(addr.sin_addr);
  end;
end;

function TSockets.GetLocalHostName: string;
var
  szHostName: array[0..128] of char;
begin
  GetHostName(szHostName,128);
  Result := szHostName;
end;
```

```
function TSockets.GetPeerPort(aSocket: TSocket): string;
var
  addr: TSockAddrIn;
  addrlen: integer;
begin
  addrlen := sizeof(addr);
  getpeername(aSocket,addr,addrlen);
  Result := IntToStr(ntohs(addr.sin_port));
end;

function TSockets.GetPeerIPAddr(aSocket: TSocket): string;
var
  addr: TSockAddrIn;
  addrlen: integer;
  szIPAddr: PChar;
begin
  addrlen := sizeof(addr);
  getpeername(aSocket,addr,addrlen);
  szIPAddr := inet_ntoa(addr.sin_addr);
  Result := StrPas(szIPAddr);
end;


function TSockets.SReceive(Socket: TSocket; szBuff: PChar; var rlen: integer) : integer;
begin
  if Socket <> INVALID_SOCKET then
  begin
    if FMode = 0 then
      SetTimeout;
    Result := recv(Socket,szBuff^,rlen,0);
    if FMode = 0 then
      ResetTimeout;
    if rlen < 0 then
      SocketError(FSocket,'SReceive',WSAGetLastError);
  end
  else Result := -1;
end;

function TSockets.SSend(Socket: TSocket; szBuff: PChar; var slen: integer): integer;
begin
  if Socket <> INVALID_SOCKET then
  begin
    if FMode = 0 then
```

```
      SetTimeout;
    slen := send(Socket,szBuff^,slen,0);
    if FMode = 0 then
      ResetTimeout;
    if slen < 0 then
      SocketError(FSocket,'SSend',WSAGetLastError);
    Result := slen;
  end;
end;

procedure TSockets.WMASyncSelect(var msg: TMessage);
var
  err: integer;
  errfn: string;
begin
  err := WSAGetSelectError(msg.LParam);
  if err > WSABASEERR then
  begin
    case WSAGetSelectEvent(msg.lParam) of
      FD_READ: errfn := 'FD_READ';
      FD_WRITE: errfn := 'FD_WRITE';
      FD_CLOSE: errfn := 'FD_CLOSE';
      FD_ACCEPT: errfn := 'FD_ACCEPT';
      FD_CONNECT: errfn := 'FD_CONNECT';
    end;
    SocketError(msg.wParam,errfn,err);
  end
  else
  case WSAGetSelectEvent(msg.lParam) of
    FD_READ:
    begin
      if Assigned(FDataAvailable) then
        FDataAvailable(Self,msg.wParam);
    end;
    FD_WRITE:
    begin
      if Assigned(FDataNeeded) then
        FDataNeeded(Self,msg.wParam);
    end;
    FD_CLOSE:
    begin
      if Assigned(FSessionClosed) then
        FSessionClosed(Self,msg.wParam);
    end;
    FD_ACCEPT:
```

```
    begin
     if Assigned(FSessionAvailable) then
      FSessionAvailable(Self,msg.wParam);
    end;
    FD_CONNECT:
    begin
     if Assigned(FSessionConnected) then
      FSessionConnected(Self,msg.wParam);
    end;
   end;
  end;

  procedure TSockets.WMTimer(var msg: TMessage);
  var
   szErrMsg: array[0..255] of char;
  begin
   KillTimer(Handle,10);
   if WSAIsBlocking then
   begin
    WSACancelBlockingCall;
    if Assigned(FErrorOccurred) then
     FErrorOccurred(Self,FSocket,WSAETIMEDOUT,'Blocking call timed out')
    else
     begin
      StrPCopy(szErrMsg,'Error ' + IntToStr(WSAETIMEDOUT) + #13#10 +
       'Blocking call timed out');
      Application.MessageBox(szErrMsg, 'WINSOCK CALL CANCELED', mb_OKCancel +
       mb_DefButton1);
     end;
   end;
  end;


  procedure TSockets.SConnect;
  var
   iStatus: integer;
   szTcp: PChar;
   szPort: array[0..31] of char;
   szData: array[0..256] of char;
   bind_sin: TSockAddrIn;
   alport: TSocket;
  begin
   if FPort = '' then
   begin
    Application.MessageBox('No Port Specified', 'WINSOCK ERROR', mb_OKCancel +
```

```
    mb_DefButton1);
   exit;
  end;
  if FIPAddr = '' then
  begin
   Application.MessageBox('No IP Address Specified', 'WINSOCK ERROR', mb_OKCancel +
    mb_DefButton1);
   exit;
  end;
  sin.sin_family := AF_INET;
  StrPCopy(szPort,FPort);
  szTcp := 'tcp';
  Pse := getservbyname(szPort,szTcp);
  if Pse = nil then
    sin.sin_port := htons(StrToInt(StrPas(szPort)))
  else sin.sin_port := Pse^.s_port;
  StrPCopy(szData,FIPAddr);
  sin.sin_addr.s_addr := inet_addr(szData);
  if sin.sin_addr.s_addr = INADDR_NONE then
   begin
    Phe := gethostbyname(szData);
    if Phe = nil then
     begin
      StrPCopy(szData,'Cannot convert host address');
      Application.MessageBox(szData, 'WINSOCK ERROR', mb_OKCancel +
        mb_DefButton1);
      exit;
     end;
    sin.sin_addr.S_addr := longint(plongint(Phe^.h_addr_list^)^);
   end;
  Ppe := getprotobyname(szTcp);
  FSocket := socket(PF_INET,SOCK_STREAM,Ppe^.p_proto);
  if FSocket < 0 then
   SocketError(INVALID_SOCKET,'SConnect (socket)',WSAGetLastError);
  if FAuthorized = True then
  begin
   alport := IPPORT_RESERVED;
   bind_sin.sin_family := AF_INET;
   bind_sin.sin_addr.s_addr := 0;
   repeat
    bind_sin.sin_port := htons(alport);
    if bind(FSocket,bind_sin,sizeof(bind_sin)) = 0 then
     break;
    if WSAGetLastError <> WSAEADDRINUSE then
     SocketError(FSocket,'SConnect bind()',WSAGetLastError);
```

```
      dec(alport);
     until(alport <= (IPPORT_RESERVED div 2));
    end;
    if FMode = 1 then
    begin
     iStatus := WSAASyncSelect(FSocket,Handle,WM_ASYNCSELECT,
      FD_READ or FD_CLOSE or FD_CONNECT or FD_WRITE);
     if iStatus <> 0 then
      SocketError(FSocket,'WSAAsyncSelect',WSAGetLastError);
    end
    else
     iStatus := ioctlsocket(FSocket,FIONBIO,FMode);
    if FMode = 0 then
     SetTimeout;
    iStatus := connect(FSocket,sin,sizeof(sin));
    if FMode = 0 then
     ResetTimeout;
    if iStatus <> 0 then
     begin
     iStatus := WSAGetLastError;
     if iStatus <> WSAEWOULDBLOCK then
      SocketError(FSocket,'SConnect',WSAGetLastError);
     end;
   end;

   procedure TSockets.SListen;
   var
    iStatus: integer;
    szTcp: PChar;
    szPort: array[0..31] of char;
    szData: array[0..256] of char;
   begin
    if FPort = '' then
    begin
     Application.MessageBox('No Port Specified', 'WINSOCK ERROR', mb_OKCancel +
      mb_DefButton1);
     exit;
    end;
    sin.sin_family := AF_INET;
    sin.sin_addr.s_addr := INADDR_ANY;
    szTcp := 'tcp';
    StrPCopy(szPort,FPort);
    Pse := getservbyname(szPort,szTcp);
    if Pse = nil then
     sin.sin_port := htons(StrToInt(StrPas(szPort)))
```

```
  else sin.sin_port := Pse^.s_port;
  Ppe := getprotobyname(szTcp);
  FMSocket := socket(PF_INET,SOCK_STREAM,Ppe^.p_proto);
  if FMSocket < 0 then
    SocketError(INVALID_SOCKET,'socket',WSAGetLastError);
  iStatus := bind(FMSocket, sin, sizeof(sin));
  if iStatus <> 0 then
    SocketError(FMSocket,'Bind',WSAGetLastError);
  iStatus := listen(FMSocket,5);
  if iStatus <> 0 then
    SocketError(FMSocket,'Listen',WSAGetLastError);
  if FMode = 1 then
  begin
    iStatus := WSAASyncSelect(FMSocket,Handle,WM_ASYNCSELECT,
      FD_READ or FD_WRITE or FD_ACCEPT or FD_CLOSE);
    if iStatus <> 0 then
      SocketError(FMSocket,'WSAASyncSelect',WSAGetLastError);
  end
  else ioctlsocket(FMSocket,FIONBIO,FMode);
end;

procedure TSockets.SCancelListen;
var
  iStatus: integer;
begin
  if FMode = 1 then
    WSAASyncSelect(FMSocket,Handle,WM_ASYNCSELECT,0);
  shutdown(FMSocket,2);
  iStatus := closesocket(FMSocket);
  if iStatus <> 0 then
    SocketError(FMSocket,'CancelListen (closesocket)',WSAGetLastError);
  FMSocket := 0;
end;


function TSockets.SAccept: TSocket;
var
  iStatus: integer;
  len: integer;
begin
  len := sizeof(sin);
  if FMode = 0 then
    SetTimeout;
  FSocket := accept(FMSocket,sin,len);
  if FMode = 0 then
```

```
  begin
    ResetTimeout;
    ioctlsocket(FSocket,FIONBIO,FMode);
  end;
  if FMSocket < 0 then
    SocketError(FSocket,'Accept',WSAGetLastError);
  Result := FSocket;
end;

procedure TSockets.SClose;
var
  iStatus: integer;
  lin: TLinger;
  linx: array[0..3] of char absolute lin;
begin
  if FMode = 1 then
    WSAASyncSelect(FSocket,Handle,WM_ASYNCSELECT,0);
  if WSAIsBlocking then
    WSACancelBlockingCall;
  shutdown(FSocket,2);
  lin.l_onoff := 1;
  lin.l_linger := 0;
  setsockopt(FSocket,SOL_SOCKET,SO_LINGER,linx,sizeof(lin));
  iStatus := closesocket(FSocket);
  if iStatus <> 0 then
    SocketError(FSocket,'Disconnect (closesocket)',WSAGetLastError);
  FSocket := INVALID_SOCKET;
end;


procedure TSockets.SocketError(Socket: TSocket; sockfunc: string; error: Integer);
var
  szLine: array[0..255]  of char;
  line, ErrMsg: string;
begin
  ErrMsg := SocketErrorDesc(error);
  line := 'Error '+ IntToStr(error) + ' in function ' + sockfunc +
  #13#10 + ErrMsg;
  if Assigned(FErrorOccurred) then
    FErrorOccurred(Self,Socket,error,ErrMsg)
  else
    begin
      StrPCopy(szLine,line);
      Application.MessageBox(szLine, 'WINSOCK ERROR', mb_OKCancel +
        mb_DefButton1);
```

```
    halt;
  end;
end;

function TSockets.SocketErrorDesc(error: integer) : string;
begin
 case error of
  WSAEINTR:
   SocketErrorDesc := 'Interrupted system call';
  WSAEBADF:
   SocketErrorDesc := 'Bad file number';
  WSAEACCES:
   SocketErrorDesc := 'Permission denied';
  WSAEFAULT:
   SocketErrorDesc := 'Bad address';
  WSAEINVAL:
   SocketErrorDesc := 'Invalid argument';
  WSAEMFILE:
   SocketErrorDesc := 'Too many open files';
  WSAEWOULDBLOCK:
   SocketErrorDesc := 'Operation would block';
  WSAEINPROGRESS:
   SocketErrorDesc := 'Operation now in progress';
  WSAEALREADY:
   SocketErrorDesc := 'Operation already in progress';
  WSAENOTSOCK:
   SocketErrorDesc := 'Socket operation on non-socket';
  WSAEDESTADDRREQ:
   SocketErrorDesc := 'Destination address required';
  WSAEMSGSIZE:
   SocketErrorDesc := 'Message too long';
  WSAEPROTOTYPE:
   SocketErrorDesc := 'Protocol wrong type for socket';
  WSAENOPROTOOPT:
   SocketErrorDesc := 'Protocol not available';
  WSAEPROTONOSUPPORT:
   SocketErrorDesc := 'Protocol not supported';
  WSAESOCKTNOSUPPORT:
   SocketErrorDesc := 'Socket type not supported';
  WSAEOPNOTSUPP:
   SocketErrorDesc := 'Operation not supported on socket';
  WSAEPFNOSUPPORT:
   SocketErrorDesc := 'Protocol family not supported';
  WSAEAFNOSUPPORT:
   SocketErrorDesc := 'Address family not supported by protocol family';
```

```
WSAEADDRINUSE:
  SocketErrorDesc := 'Address already in use';
WSAEADDRNOTAVAIL:
  SocketErrorDesc := 'Can''t assign requested address';
WSAENETDOWN:
  SocketErrorDesc := 'Network is down';
WSAENETUNREACH:
  SocketErrorDesc := 'Network is unreachable';
WSAENETRESET:
  SocketErrorDesc := 'Network dropped connection on reset';
WSAECONNABORTED:
  SocketErrorDesc := 'Software caused connection abort';
WSAECONNRESET:
  SocketErrorDesc := 'Connection reset by peer';
WSAENOBUFS:
  SocketErrorDesc := 'No buffer space available';
WSAEISCONN:
  SocketErrorDesc := 'Socket is already connected';
WSAENOTCONN:
  SocketErrorDesc := 'Socket is not connected';
WSAESHUTDOWN:
  SocketErrorDesc := 'Can''t send after socket shutdown';
WSAETOOMANYREFS:
  SocketErrorDesc := 'Too many references: can''t splice';
WSAETIMEDOUT:
  SocketErrorDesc := 'Connection timed out';
WSAECONNREFUSED:
  SocketErrorDesc := 'Connection refused';
WSAELOOP:
  SocketErrorDesc := 'Too many levels of symbolic links';
WSAENAMETOOLONG:
  SocketErrorDesc := 'File name too long';
WSAEHOSTDOWN:
  SocketErrorDesc := 'Host is down';
WSAEHOSTUNREACH:
  SocketErrorDesc := 'No route to host';
WSAENOTEMPTY:
  SocketErrorDesc := 'Directory not empty';
WSAEPROCLIM:
  SocketErrorDesc := 'Too many processes';
WSAEUSERS:
  SocketErrorDesc := 'Too many users';
WSAEDQUOT:
  SocketErrorDesc := 'Disc quota exceeded';
WSAESTALE:
```

```
      SocketErrorDesc := 'Stale NFS file handle';
    WSAEREMOTE:
      SocketErrorDesc := 'Too many levels of remote in path';
    WSASYSNOTREADY:
      SocketErrorDesc := 'Network sub-system is unusable';
    WSAVERNOTSUPPORTED:
      SocketErrorDesc := 'WinSock DLL cannot support this application';
    WSANOTINITIALISED:
      SocketErrorDesc := 'WinSock not initialized';
    WSAHOST_NOT_FOUND:
      SocketErrorDesc := 'Host not found';
    WSATRY_AGAIN:
      SocketErrorDesc := 'Non-authoritative host not found';
    WSANO_RECOVERY:
      SocketErrorDesc := 'Non-recoverable error';
    WSANO_DATA:
      SocketErrorDesc := 'No Data';
    else SocketErrorDesc := 'Not a WinSock error';
  end;
end;

procedure TSockets.SetTimeout;
begin
  if FTimeout > 0 then
    SetTimer(Handle,10,FTimeout*1000,nil);
end;

procedure TSockets.ResetTimeout;
begin
  if FTimeout > 0 then
    KillTimer(Handle,10);
end;

end.
```

National Semiconductor

October 1991

TP5088 DTMF Generator for Binary Data

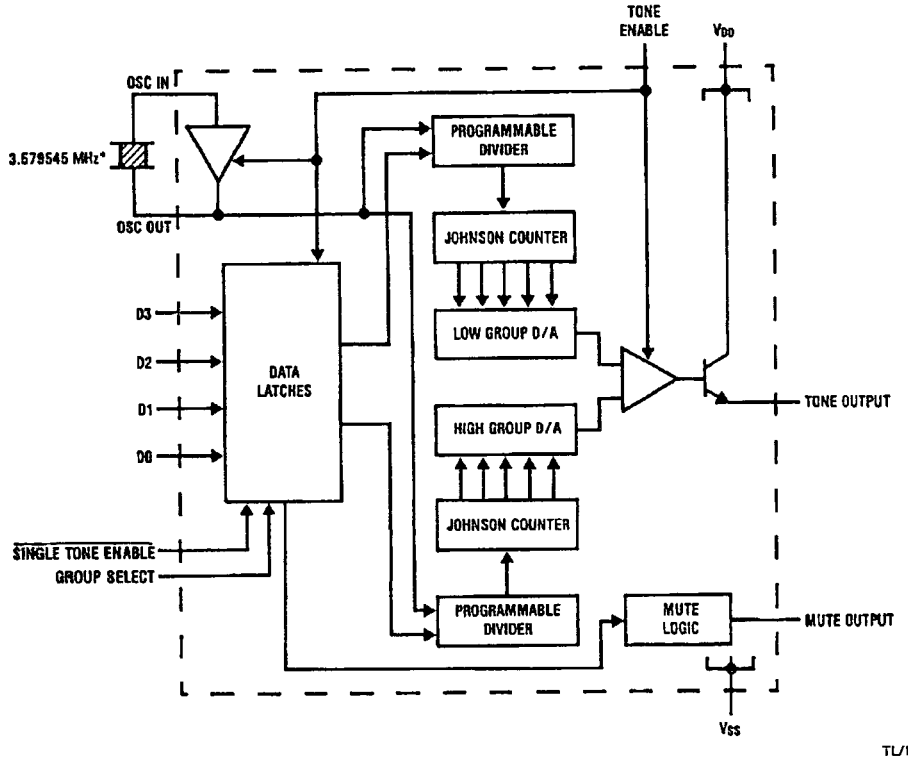
General Description

This CMOS device provides low cost tone-dialing capability in microprocessor-controlled telephone applications. 4-bit binary data is decoded directly, without the need for conversion to simulated keyboard inputs required by standard DTMF generators. With the TONE ENABLE input low, the oscillator is inhibited and the device is in a low power idle mode. On the low-to-high transition of TONE ENABLE, data is latched into the device and the selected tone pair from the standard DTMF frequencies is generated. An open-drain N-channel transistor provides a MUTE output during tone generation.

Features

- Direct microprocessor interface
- Binary data inputs with latches
- Generates 16 standard tone pairs
- On-chip 3.579545 MHz crystal-controlled oscillator
- Better than 0.64% frequency accuracy
- High group pre-emphasis
- Low harmonic distortion
- MUTE output interfaces to speech network
- Low power idle mode
- 3.5V–8V operation

Block Diagram



TL/H/5004–1

*Crystal Specification: Parallel Resonant 3.579545 MHz, $R_S \leq 150\Omega$, L = 100 mH, C_0 = 5 pF, C_1 = 0.02 pF.

RRD-B30M115/Printed in U. S. A.

Absolute Maximum Ratings

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Supply Voltage ($V_{DD}-V_{SS}$)	12V
$\overline{MUTE}$ Voltage	12V
Maximum Voltage at Any Other Pin	V_{DD} + 0.3V to V_{SS} − 0.3V
Operating Temperature, T_A	−30°C to +70°C
Storage Temperature	−55°C to +150°C
Maximum Power Dissipation	500 mW

Electrical Characteristics

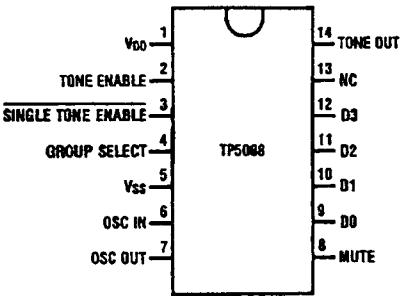
Unless otherwise noted, limits printed in **BOLD** characters are guaranteed for V_{DD} = 3.5V to 8V, T_A = 0°C to +70°C by correlation with 100% electrical testing at T_A = 25°C. All other limits are assured by correlation with other production tests and/or product design and characterization.

Parameter	Conditions	Min	Typ	Max	Units
Minimum Supply Voltage, V_{DD} (min)	Generating Tones	**3.5**			V
Minimum Supply Voltage for Data Input, TONE ENABLE and MUTE Logic Functions		2			V
Operating Current Idle Generating Tones	 R_L = ∞, D0–D3 Open V_{DD} = 3.5V, Mute Open		 55 1.5	 **350** **2.5**	 μA mA
Input Pull-Up Resistance D0–D3 TONE ENABLE			 100 50		 kΩ kΩ
Input Low Level TONE ENABLE, D0–D3				**0.2 V_{DD}**	V
Input High Level TONE ENABLE, D0–D3		**0.8 V_{DD}**			V
$\overline{MUTE}$ OUT Sink Current (TONE ENABLE LOW)	V_{DD} = 3.5V V_o = 0.5V	**0.4**			mA
$\overline{MUTE}$ OUT Leakage Current (TONE ENABLE HIGH)	V_{DD} = 3.5V V_o = V_{DD}		1		μA
Output Amplitudes Low Group High Group	R_L = 240 Ω V_{DD} = 3.5V T_A = 25°C	 **130** **180**	 170 230	 **220** **310**	 mVrms mVrms
Mean Output DC Offset	V_{DD} = 3.5V V_{DD} = 8V		1.2 3.6		V V
High Group Pre-Emphasis		**2.2**	2.7	**3.2**	dB
Dual Tone/Total Harmonic Distortion Ratio	1 MHz Bandwidth, V_{DD} = 5V R_L = 240Ω	**−20**			dB
Start-Up Time (to 90% Amplitude), t_{OSC}			4		ms
Data Set-Up Time, t_S (*Figure 2*)	V_{DD} = 5V	100			ns
Data Hold Time, t_H	V_{DD} = 5V	280			ns
Data Duration t_W	V_{DD} = 5V	600			ns

Note 1: R_L is the external load resistor connected from TONE OUT to V_{SS}.

Connection Diagram

Dual-In-Line Package



TL/H/5004–2

Top View

Order Number TP5088WM or TP5088N
See NS Package M14B or N14A

Functional Description

With the TONE ENABLE pin pulled low, the device is in a low power idle mode, with the oscillator inhibited and the output transistor turned off. Data on inputs D0–D3 is ignored until a rising transition on TONE ENABLE. Data meeting the timing specifications is latched in, the oscillator and output stage are enabled, and tone generation begins. The decoded data sets the high group and low group programmable counters to the appropriate divide ratios. These counters sequence two ratioed-capacitor D/A converters through a series of 28 equal duration steps per sine wave cycle. On-chip regulators ensure good stability of tone amplitudes with variations in supply voltage and temperature. The two tones are summed by a mixer amplifier, with preemphasis applied to the high group tone. The output is an NPN emitter-follower requiring the addition of an external load resistor to V_{SS}.

Table I shows the accuracies of the tone output frequencies and Table II is the Functional Truth Table.

TABLE I. Output Frequency Accuracy

Tone Group	Standard DTMF (Hz)	Tone Output Frequency	% Deviation from Standard
Low	697	694.8	−0.32
Group	770	770.1	+0.02
f_L	852	852.4	+0.03
	941	940.0	−0.11
High	1209	1206.0	−0.24
Group	1336	1331.7	−0.32
f_H	1477	1486.5	+0.64
	1633	1639.0	+0.37

Pin Descriptions

V_{DD} (Pin 1): This is the positive supply to the device, referenced to V_{SS}. The collector of the TONE OUT transistor is also connected to this pin.

V_{SS} (Pin 5): This is the negative voltage supply. All voltages are referenced to this pin.

OSC IN, OSC OUT (Pins 6 and 7): All tone generation timing is derived from the on-chip oscillator circuit. A low-cost 3.579545 MHz A-cut crystal (NTSC TV color-burst) is needed between pins 6 and 7. Load capacitors and a feedback resistor are included on-chip for good start-up and stability. The oscillator is stopped when the TONE ENABLE input is pulled to logic low.

TONE ENABLE Input (Pin 2): This input has an internal pull-up resistor. When TONE ENABLE is pulled to logic low, the oscillator is inhibited and the tone generators and output transistor are turned off. A low to high transition on TONE ENABLE latches in data from D0–D3. The oscillator starts, and tone generation continues until TONE ENABLE is pulled low again.

MUTE (Pin 8): This output is an open-drain N-channel device that sinks current to V_{SS} when TONE ENABLE is low and no tones are being generated. The device turns off when TONE ENABLE is high.

D0, D1, D2, D3 (Pins 9, 10, 11, 12): These are the inputs for binary-coded data, which is latched in on the rising edge of TONE ENABLE. Data must meet the timing specifications of *Figure 2*. At all other times these inputs are ignored and may be multiplexed with other system functions.

TONE OUT (Pin 14): This output is the open emitter of an NPN transistor, the collector of which is connected internally to V_{DD}. When an external load resistor is connected from TONE OUT to V_{SS}, the output voltage on this pin is the sum of the high and low group tones superimposed on a DC offset. When not generating tones, this output transistor is turned off to minimize the device idle current.

$\overline{\text{SINGLE TONE ENABLE}}$ (Pin 3): This input has an internal pull-up resistor. When pulled to V_{SS}, the device is in single tone mode and only a single tone will be generated at pin 14 (for testing purposes). For normal operation, leave this pin open-circuit or pull to V_{DD}.

GROUP SELECT (Pin 4): This pin is used to select the high group or low group frequency when the device is in single tone mode. It has an internal pull-up resistor. Leaving this pin open-circuit or pulling it to V_{DD} will generate the high group, while pulling to V_{SS} will generate the low group frequency at the TONE OUT pin.

TABLE II. Functional Truth Table

Keyboard Equivalent	Data Inputs				TONE ENABLE	TONES OUT		MUTE
	D3	D2	D1	D0		f_L (Hz)	f_H(Hz)	
X	X	X	X	X	0	0V	0V	0V
1	0	0	0	1	⎍	697	1209	O/C
2	0	0	1	0	⎍	697	1336	O/C
3	0	0	1	1	⎍	697	1477	O/C
4	0	1	0	0	⎍	770	1209	O/C
5	0	1	0	1	⎍	770	1336	O/C
6	0	1	1	0	⎍	770	1477	O/C
7	0	1	1	1	⎍	852	1209	O/C
8	1	0	0	0	⎍	852	1336	O/C
9	1	0	0	1	⎍	852	1477	O/C
0	1	0	1	0	⎍	941	1336	O/C
*	1	0	1	1	⎍	941	1209	O/C
#	1	1	0	0	⎍	941	1477	O/C
A	1	1	0	1	⎍	697	1633	O/C
B	1	1	1	0	⎍	770	1633	O/C
C	1	1	1	1	⎍	852	1633	O/C
D	0	0	0	0	⎍	941	1633	O/C

Timing Diagram

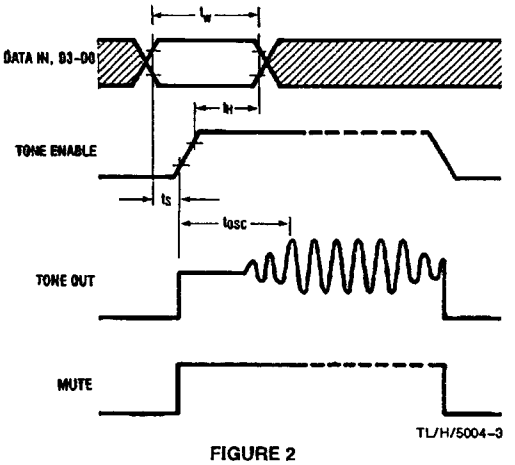
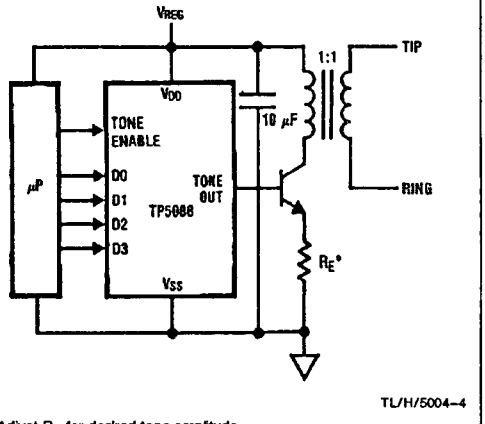


FIGURE 2

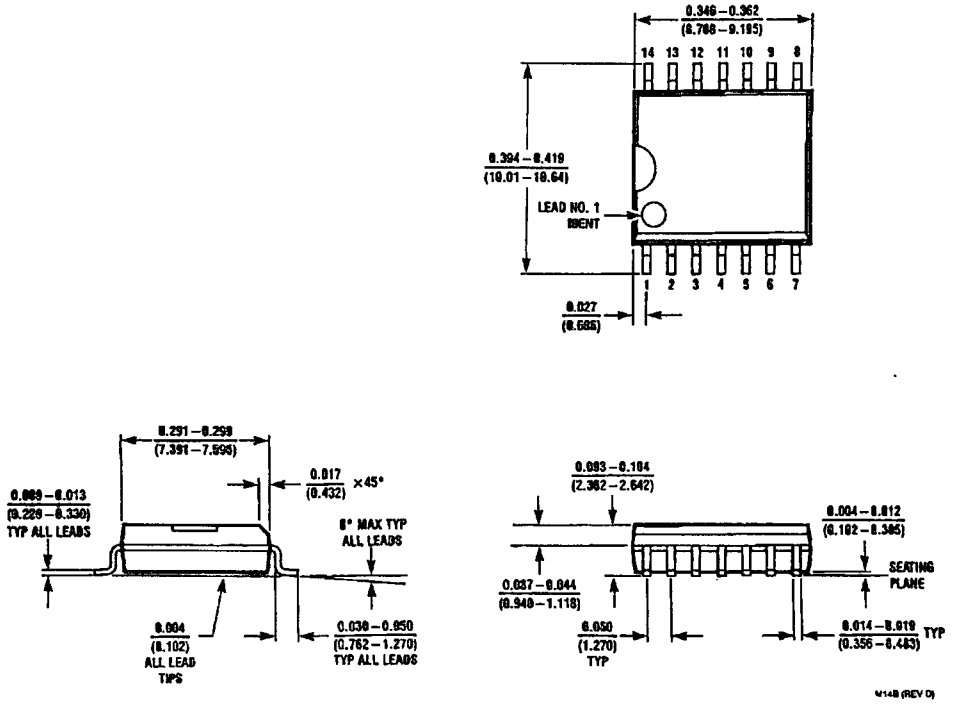
Typical Application



*Adjust R_E for desired tone amplitude.

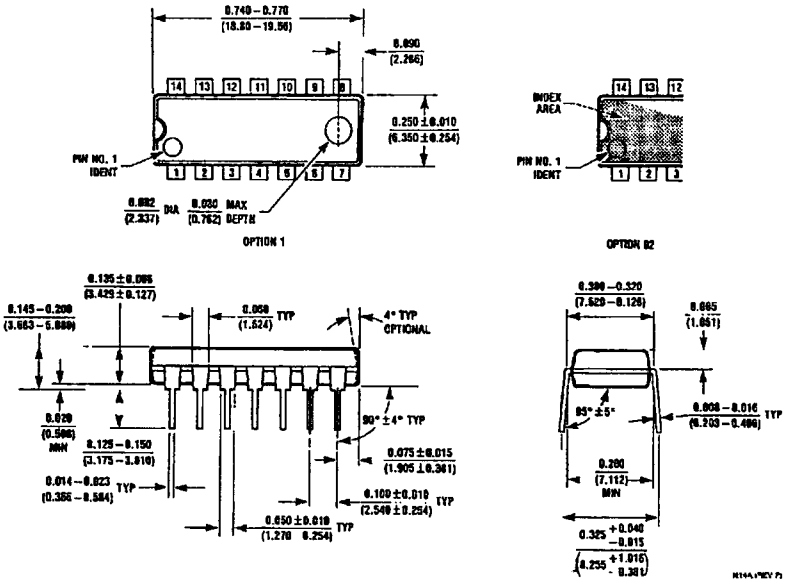
FIGURE 3

Physical Dimensions inches (millimeters)



Order Number TP5088WM
NS Package Number M14B

Physical Dimensions inches (millimeters) (Continued)



Molded Dual-In-Line (N)
Order Number TP5088N
NS Package Number N14A

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

National Semiconductor Corporation
1111 West Bardin Road
Arlington, TX 76017
Tel: 1(800) 272-9959
Fax: 1(800) 737-7018

National Semiconductor Europe
Fax: (+49) 0-180-530 85 86
Email: cnjwge@tevm2.nsc.com
Deutsch Tel: (+49) 0-180-530 85 85
English Tel: (+49) 0-180-532 78 32
Français Tel: (+49) 0-180-532 93 58
Italiano Tel: (+49) 0-180-534 16 80

National Semiconductor Hong Kong Ltd.
13th Floor, Straight Block,
Ocean Centre, 5 Canton Rd.
Tsimshatsui, Kowloon
Hong Kong
Tel: (852) 2737-1600
Fax: (852) 2736-9960

National Semiconductor Japan Ltd.
Tel: 81-043-299-2309
Fax: 81-043-299-2408

HARRIS Semiconductor

ICL232

ugust 1997

+5V Powered, Dual RS-232 Transmitter/Receiver

Features

- Meets All RS-232C and V.28 Specifications
- Requires Only Single +5V Power Supply
- Onboard Voltage Doubler/Inverter
- Low Power Consumption
- 2 Drivers
  - ±9V Output Swing for +5V Input
  - 300Ω Power-off Source Impedance
  - Output Current Limiting
  - TTL/CMOS Compatible
  - 30V/μs Maximum Slew Rate
- 2 Receivers
  - ±30V Input Voltage Range
  - 3kΩ to 7kΩ Input Impedance
  - 0.5V Hysteresis to Improve Noise Rejection
- All Critical Parameters are Guaranteed Over the Entire Commercial, Industrial and Military Temperature Ranges

Applications

- Any System Requiring RS-232 Communications Port
  - Computer - Portable and Mainframe
  - Peripheral - Printers and Terminals
  - Portable Instrumentation
  - Modems
- Dataloggers

Description

The ICL232 is a dual RS-232 transmitter/receiver interface circuit that meets all EIA RS-232C and V.28 specifications. It requires a single +5V power supply, and features two onboard charge pump voltage converters which generate +10V and -10V supplies from the 5V supply.

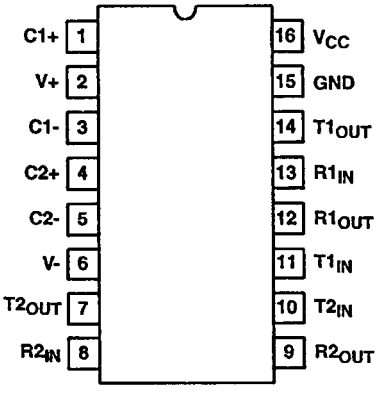
The drivers feature true TTL/CMOS input compatibility, slew-rate-limited output, and 300Ω power-off source impedance. The receivers can handle up to +30V, and have a 3kΩ to 7kΩ input impedance. The receivers also have hysteresis to improve noise rejection.

Ordering Information

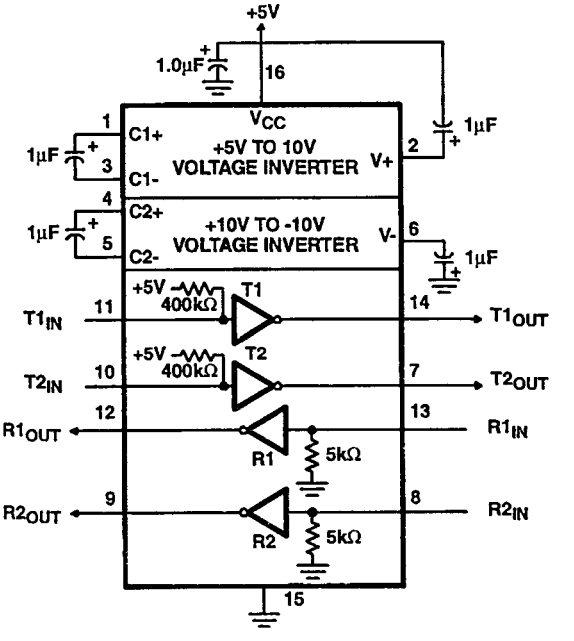
PART NUMBER	TEMP. RANGE (°C)	PACKAGE	PKG. NO.
ICL232CPE	0 to 70	16 Ld PDIP	E16.3
ICL232CBE	0 to 70	16 Ld SOIC	M16.3
ICL232IPE	-40 to 85	16 Ld PDIP	E16.3
ICL232IJE	-40 to 85	16 Ld CERDIP	F16.3
ICL232IBE	-40 to 85	16 Ld SOIC	M16.3
ICL232MJE	-55 to 125	16 Ld CERDIP	F16.3

Pinout

ICL232 (PDIP, CERDIP, SOIC)
TOP VIEW



Functional Diagram



CAUTION: These devices are sensitive to electrostatic discharge. Users should follow proper IC Handling Procedures.

File Number 3020.5

Absolute Maximum Ratings

V_{CC} to Ground ..... (GND -0.3V) < V_{CC} < 6V
V+ to Ground ..... (V_{CC} -0.3V) < V+ < 12V
V- to Ground ..... -12V < V- < (GND +0.3V)
Input Voltages
  $T1_{IN}$, $T2_{IN}$ ..... (V- -0.3V) < V_{IN} < (V+ +0.3V)
  $R1_{IN}$, $R2_{IN}$ ..... ±30V
Output Voltages
  $T1_{OUT}$, $T2_{OUT}$ ..... (V- -0.3V) < V_{TXOUT} < (V+ +0.3V)
  $R1_{OUT}$, $R2_{OUT}$ ..... (GND -0.3V) < V_{RXOUT} < (V_{CC} +0.3V)
Short Circuit Duration
  $T1_{OUT}$, $T2_{OUT}$ ..... Continuous
  $R1_{OUT}$, $R2_{OUT}$ ..... Continuous

Operating Conditions

Temperature Ranges
  ICL232C ..... 0°C to 70°C
  ICL232I ..... -40°C to 85°C
  ICL232M ..... -55°C to 125°C

Thermal Information

Thermal Resistance (Typical, Note 1)	θ_{JA} (°C/W)	θ_{JC} (°C/W)
CERDIP Package	80	18
PDIP Package	100	N/A
SOIC Package	100	N/A

Maximum Junction Temperature
  Plastic Packages ..... 150°C
  Ceramic Package ..... 175°C
Maximum Storage Temperature Range ..... -65°C to 150°C
Maximum Lead Temperature (Soldering 10s) ..... 300°C

CAUTION: Stresses above those listed in "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress only rating and operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied.

NOTE:

1. θ_{JA} is measured with the component mounted on an evaluation PC board in free air.

Electrical Specifications

Test Conditions: V_{CC} = +5V ±10%, T_A = Operating Temperature Range. Test Circuit as in Figure 8 Unless Otherwise Specified

PARAMETER	TEST CONDITIONS	MIN	TYP	MAX	UNITS
Transmitter Output Voltage Swing, T_{OUT}	$T1_{OUT}$ and $T2_{OUT}$ Loaded with 3kΩ to Ground	±5	±9	±10	V
Power Supply Current, I_{CC}	Outputs Unloaded, T_A = 25°C	-	5	10	mA
T_{IN}, Input Logic Low, V_{IL}		-	-	0.8	V
T_{IN}, Input Logic High, V_{IH}		2.0	-	-	V
Logic Pullup Current, I_P	$T1_{IN}$, $T2_{IN}$ = 0V	-	15	200	µA
RS-232 Input Voltage Range, V_{IN}		-30	-	+30	V
Receiver Input Impedance, R_{IN}	V_{IN} = ±3V	3.0	5.0	7.0	kΩ
Receiver Input Low Threshold, V_{IN} (H-L)	V_{CC} = 5V, T_A = 25°C	0.8	1.2	-	V
Receiver Input High Threshold, V_{IN} (L-H)	V_{CC} = 5V, T_A = 25°C	-	1.7	2.4	V
Receiver Input Hysteresis, V_{HYST}		0.2	0.5	1.0	V
TTL/CMOS Receiver Output Voltage Low, V_{OL}	I_{OUT} = 3.2mA	-	0.1	0.4	V
TTL/CMOS Receiver Output Voltage High, V_{OH}	I_{OUT} = -1.0mA	3.5	4.6	-	V
Propagation Delay, t_{PD}	RS-232 to TTL	-	0.5	-	µs
Instantaneous Slew Rate, SR	C_L = 10pF, R_L = 3kΩ, T_A = 25°C (Notes 2, 3)	-	-	30	V/µs
Transition Region Slew Rate, SR_T	R_L = 3kΩ, C_L = 2500pF Measured from +3V to -3V or -3V to +3V	-	3	-	V/µs
Output Resistance, R_{OUT}	V_{CC} = V+ = V- = 0V, V_{OUT} = ±2V	300	-	-	Ω
RS-232 Output Short Circuit Current, I_{SC}	$T1_{OUT}$ or $T2_{OUT}$ Shorted to GND	-	±10	-	mA

NOTES:

2. Guaranteed by design.
3. See Figure 4 for definition.

Test Circuits

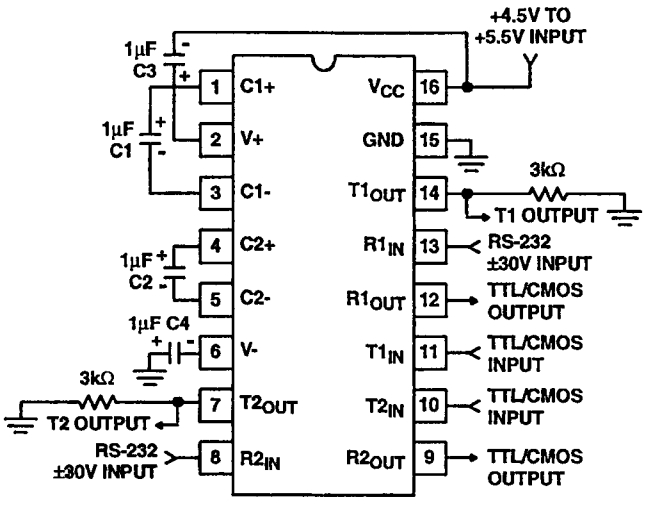


FIGURE 1. GENERAL TEST CIRCUIT

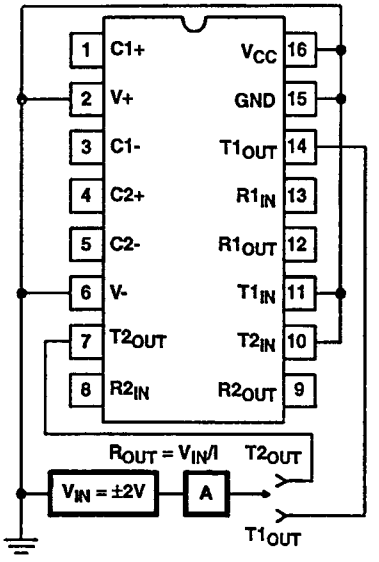


FIGURE 2. POWER-OFF SOURCE RESISTANCE CONFIGURATION

Typical Performance Curves

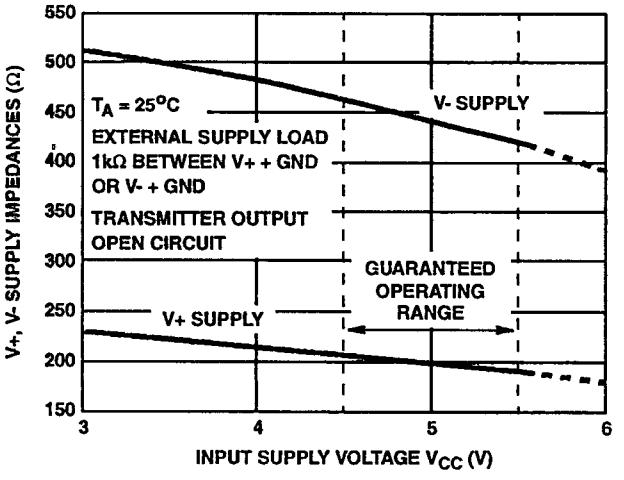


FIGURE 3. V+, V- OUTPUT IMPEDANCES vs V_{CC}

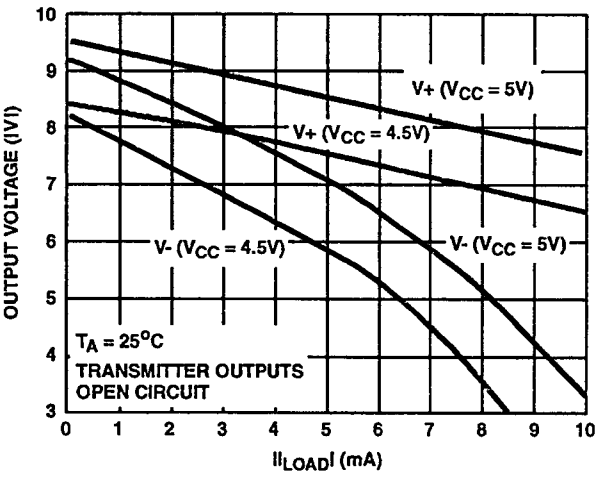


FIGURE 4. V+, V- OUTPUT VOLTAGES vs LOAD CURRENT

Pin Descriptions

PDIP, CERDIP	SOIC	PIN NAME	DESCRIPTION
1	1	C1+	External capacitor "+" for internal voltage doubler.
2	2	V+	Internally generated +10V (typical) supply.
3	3	C1-	External capacitor "-" for internal voltage doubler.
4	4	C2+	External capacitor "+" internal voltage inverter.
5	5	C2-	External capacitor "-" internal voltage inverter.
6	6	V-	Internally generated -10V (typical) supply.
7	7	$T2_{OUT}$	RS-232 Transmitter 2 output ±10V (typical).
8	8	$R2_{IN}$	RS-232 Receiver 2 input, with internal 5K pulldown resistor to GND.
9	9	R2out	Receiver 2 TTL/CMOS output.
10	10	$T2_{IN}$	Transmitter 2 TTL/CMOS input, with internal 400K pullup resistor to V_{CC}.
11	11	$T1_{IN}$	Transmitter 1 TTL/CMOS input, with internal 400K pullup resistor to V_{CC}.

Pin Descriptions (Continued)

PDIP, CERDIP	SOIC	PIN NAME	DESCRIPTION
12	12	$R1_{OUT}$	Receiver 1 TTL/CMOS output.
13	13	$R1_{IN}$	RS-232 Receiver 1 input, with internal 5K pulldown resistor to GND.
14	14	$T1_{OUT}$	RS-232 Transmitter 1 output ±10V (typical).
15	15	GND	Supply Ground.
16	16	V_{CC}	Positive Power Supply +5V ±10%

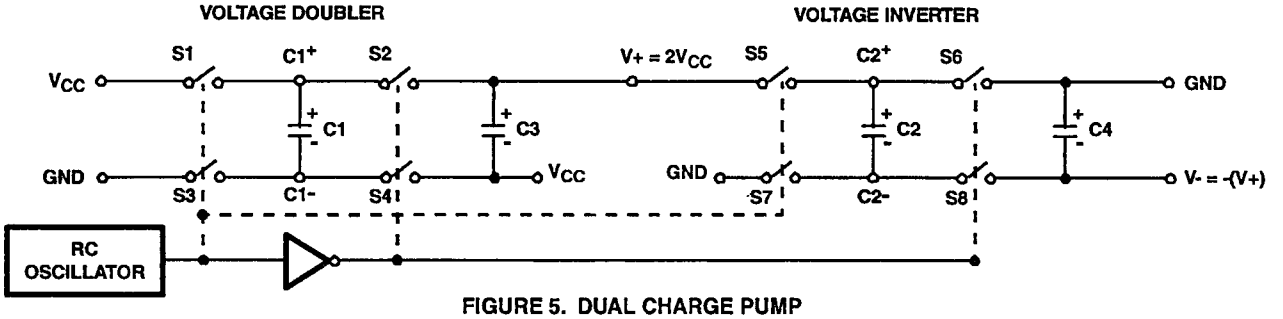


FIGURE 5. DUAL CHARGE PUMP

Detailed Description

The ICL232 is a dual RS-232 transmitter/receiver powered by a single +5V power supply which meets all EIA RS232C specifications and features low power consumption. The functional diagram illustrates the major elements of the ICL232. The circuit is divided into three sections: a voltage doubler/inverter, dual transmitters, and dual receivers Voltage Converter.

An equivalent circuit of the dual charge pump is illustrated in Figure 5.

The voltage quadrupler contains two charge pumps which use two phases of an internally generated clock to generate +10V and -10V. The nominal clock frequency is 16kHz. During phase one of the clock, capacitor C1 is charged to V_{CC}. During phase two, the voltage on C1 is added to V_{CC}, producing a signal across C2 equal to twice V_{CC}. At the same time, C3 is also charged to $2V_{CC}$, and then during phase one, it is inverted with respect to ground to produce a signal across C4 equal to $-2V_{CC}$. The voltage converter accepts input voltages up to 5.5V. The output impedance of the doubler (V+) is approximately 200Ω, and the output impedance of the inverter (V-) is approximately 450Ω. Typical graphs are presented which show the voltage converters output vs input voltage and output voltages vs load characteristics. The test circuit (Figure 3) uses 1µF capacitors for C1-C4, however, the value is not critical. Increasing the values of C1 and C2 will lower the output impedance of the voltage doubler and inverter, and increasing the values of the reservoir capacitors, C3 and C4, lowers the ripple on the V+ and V- supplies.

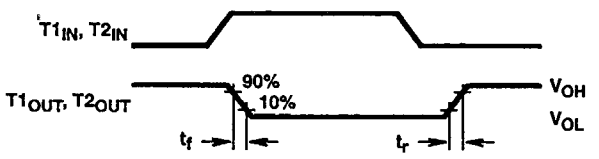


FIGURE 6. SLEW RATE DEFINITION

Transmitters

The transmitters are TTL/CMOS compatible inverters which translate the inputs to RS-232 outputs. The input logic threshold is about 26% of V_{CC}, or 1.3V for V_{CC} = 5V. A logic 1 at the input results in a voltage of between -5V and V- at the output, and a logic 0 results in a voltage between +5V and (V+ - 0.6V). Each transmitter input has an internal 400kΩ pullup resistor so any unused input can be left unconnected and its output remains in its low state. The output voltage swing meets the RS-232C specification of ±5V minimum with the worst case conditions of: both transmitters driving 3kΩ minimum load impedance, V_{CC} = 4.5V, and maximum allowable operating temperature. The transmitters have an internally limited output slew rate which is less than 30V/µs. The outputs are short circuit protected and can be shorted to ground indefinitely. The powered down output impedance is a minimum of 300Ω with ±2V applied to the outputs and V_{CC} = 0V.

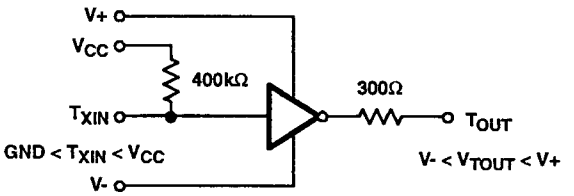


FIGURE 7. TRANSMITTER

Receivers

The receiver inputs accept up to ±30V while presenting the required 3kΩ to 7kΩ input impedance even it the power is off (V_{CC} = 0V). The receivers have a typical input threshold of 1.3V which is within the ±3V limits, known as the transition region, of the RS-232 specification. The receiver output is 0V to V_{CC}. The output will be low whenever the input is greater than 2.4V and high whenever the input is floating or driven between +0.8V and -30V. The receivers feature 0.5V hysteresis to improve noise rejection.

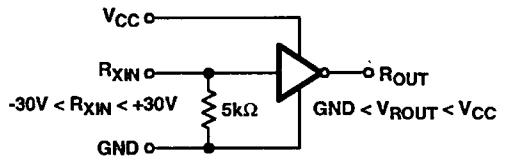
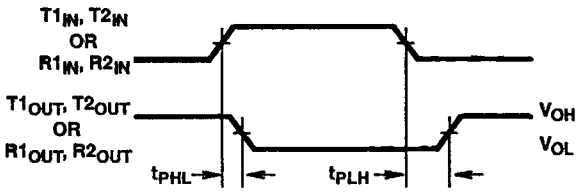


FIGURE 8. RECEIVER



$$\text{Average Propagation Delay} = \frac{t_{PHL} + t_{PLH}}{2}$$

FIGURE 9. PROPAGATION DELAY DEFINITION

Applications

The ICL232 may be used for all RS-232 data terminal and communication links. It is particularly useful in applications where ±12V power supplies are not available for conventional RS-232 interface circuits. The applications presented represent typical interface configurations.

A simple duplex RS-232 port with CTS/RTS handshaking is illustrated in Figure 10. Fixed output signals such as DTR (data terminal ready) and DSRS (data signaling rate select) is generated by driving them through a 5kΩ resistor connected to V+.

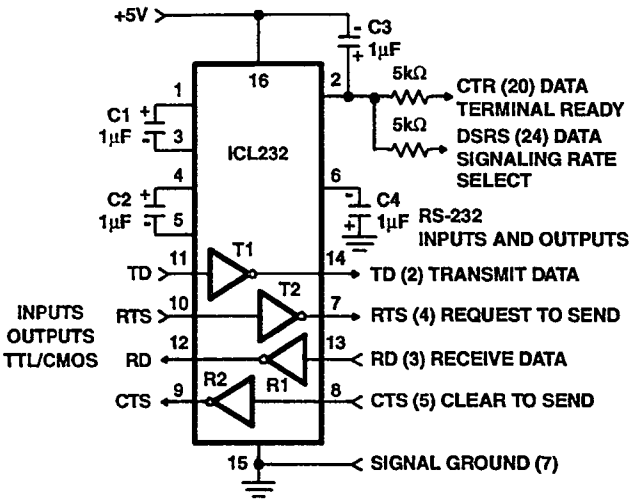


FIGURE 10. SIMPLE DUPLEX RS-232 PORT WITH CTS/RTS HANDSHAKING

In applications requiring four RS-232 inputs and outputs (Figure 11), note that each circuit requires two charge pump capacitors (C1 and C2) but can share common reservoir capacitors (C3 and C4). The benefit of sharing common reservoir capacitors is the elimination of two capacitors and the reduction of the charge pump source impedance which effectively increases the output swing of the transmitters.

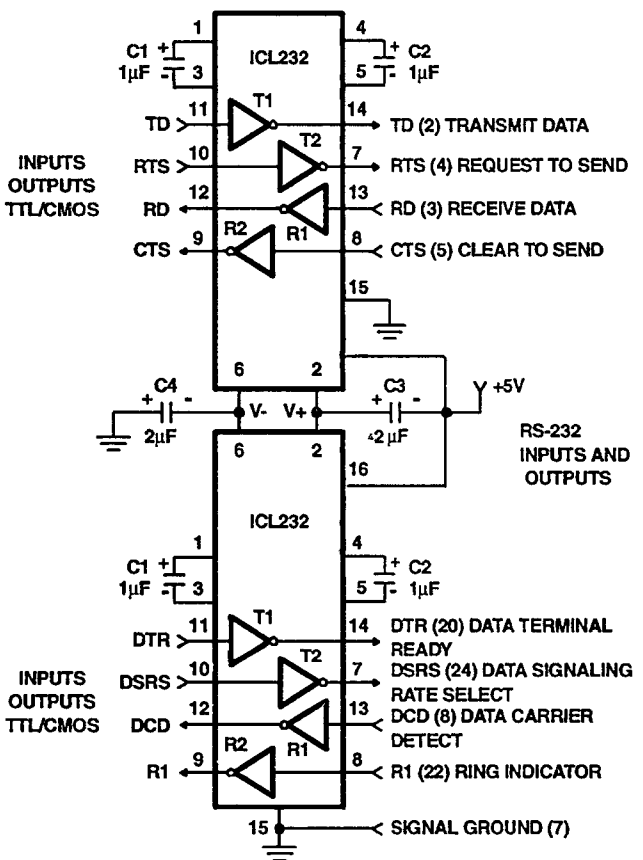


FIGURE 11. COMBINING TWO ICL232s FOR 4 PAIRS OF RS-232 INPUTS AND OUTPUTS

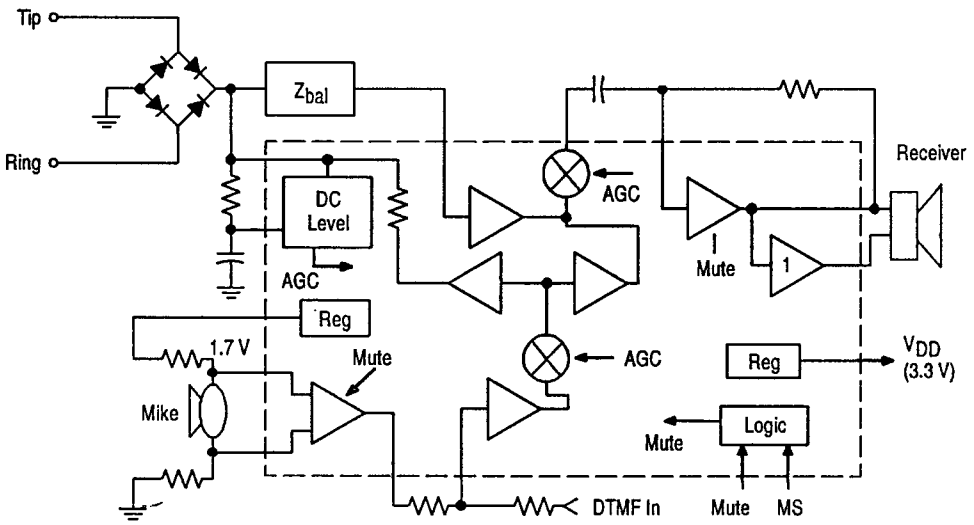
Speech Networks

Telephone Speech Network with Dialer Interface

MC34114P, DW

T_A = –20° to +70°C, Case 707, 751D

- Operation Down to 1.2 V
- Adjustable Transmit, Receive, and Sidetone Gains by External Resistors
- Differential Microphone Amplifier Input Minimizes RFI
- Transmit, Receive, and Sidetone Equalization on both Voice and DTMF Signals
- Regulated 1.7 V Output for Biasing Microphone
- Regulated 3.3 V Output for Powering External Dialer
- Microphone and Receive Amplifiers Muted During Dialing
- Differential Receive Amplifier Output Eliminates Coupling Capacitor
- Operates with Receiver Impedances of 150 Ω and Higher



หนังสืออ้างอิง

1. Todd Bill,Vince Kellen,Ray Novak," **Delphi Multimedia** ",New York Mt&t Book
2. Tim Kiemzle," **A Program's Guide to sound** ",Addison-Wesley developers press
3. Wallace Tendon," **Delphi2's Developer solutions** ",Wait Group Press
4. Jeff Duteman,Jim Micheal,Don Taylor," **Delphi proramming explorer** ",Scott Sdale
5. John L. Fike and George E. Iriend " **Understanding Telephone Electronics**" ,Texas Instrument Information Publishing Center ,1984
6. วิสันต์ อาชาเดโชพล " **ระบบโทรศัพท์ดิจิตอล** ",หจก.สำนักพิมพ์ ฟิสิกส์เซ็นเตอร์ ,2536
7. น.อ. ธวัชชัย เลื่อนฉวี " **เทคโนโลยีโทรศัพท์** " ,สำนักพิมพ์ บุ๊คสโตร์,พ.ค. 2533