การรับส่งข้อมูลเสียงผ่านระบบอินเตอร์เน็ตระหว่างเครื่องคอมพิวเตอร์ด้วยมาตรฐาน G.7 series

PC -to- PC Voice over IP with G.7 series

โดย

นางสาวธนิตา วิเศษโชติกุล 44010206

นายนิกันติ์ วชิรมน 44010258

นางสาวนิดา ชาติวัฒนศิริ 44010259

อาจารย์ที่ปรึกษา

รศ.ดร. กอบชัย เดชหาญ

ดร.พิพัฒน์ พรหมมี

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2547

ปริญญานิพนธ์ปีการศึกษา 2547

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การรับส่งข้อมูลเสียงผ่านระบบอินเตอร์เน็ตระหว่างเครื่องคอมพิวเตอร์ด้วยมาตรฐาน G.7 series

PC -to- PC Voice over IP with G.7 series

ผู้จัดทำ

1. นางสาวธนิตา วิเศษโชติกุล 44010206
2. นายนิกันติ์ วชิรมน 44010258
3. นางสาวนิดา ชาติวัฒนศิริ 44010259

อาจารย์ที่ปรึกษา

(รศ.ดร. กอบชัย เดชหาญ)

อาจารย์ที่ปรึกษา

(ดร.พิพัฒน์ พรหมมี)

การรับส่งข้อมูลเสียงผ่านระบบอินเตอร์เน็ตระหว่างเครื่องคอมพิวเตอร์ ด้วยมาตรฐาน G.7 series

PC -to- PC Voice over IP with G.7 series

โดย นางสาวธนิตา วิเศษโชติกุล 44010206

นายนิกันต์ วชิรมน 44010258

นางสาวนิดา ชาติวัฒนศิริ 44010259

อาจารย์ที่ปรึกษา

รศ.ดร. กอบชัย เดชาหาญ

ดร.พิพัฒน์ พรหมมี

บทคัดย่อ

ปริญญานิพนธ์นี้นำเสนอการส่งและรับข้อมูลเสียงผ่านเครือข่ายอินเตอร์เน็ต (Voice over IP) โดยอาศัยการแปลงข้อมูลเสียงจากอนาล็อกเป็นดิจิตอล แล้วใช้อัลกอริทึมในการบีบอัดข้อมูลในแบบต่างๆ เพื่อเปรียบเทียบประสิทธิภาพในการใช้งาน ให้มีประสิทธิภาพมากที่สุดเท่าที่จะทำได้ จากนั้นจะทำการส่งข้อมูลเสียงที่ได้ทำการบีบอัดแล้ว ผ่านทางอินเตอร์เน็ตโปรโตคอลระหว่างเครื่องคอมพิวเตอร์กับเครื่องคอมพิวเตอร์ โดยข้อมูลเสียงที่รับส่งนี้เป็นการติดต่อแบบเวลาจริง

Abstract

This project concern about voice over INTERNET Protocol (VoIP) system base on PC - to – PC . The various compression techniques are used for compressing data in order to utilize a minimum bandwidth. The internet has carried out for real-time communicating between PC - to - PC VoIP system.

สารบัญ

สารบัญ

สารบัญตาราง

สารบัญรูปภาพ

สารบัญรูปภาพ (ต่อ)

รูป หน้า

สารบัญรูปภาพ (ต่อ)

บทที่ 1
บทนำ

การติดต่อสื่อสารเป็นสิ่งที่มีความสำคัญอย่างยิ่งในชีวิตคนเรา ตั้งแต่อดีตจนถึงปัจจุบันการสื่อสารที่ใช้มากที่สุดคือ การพูดคุยกันหรือการสื่อสารทางเสียงนั่นเอง

เมื่ออินเตอร์เน็ตมีบทบาทกับชีวิตประจำวันมากขึ้นและใช้งานกันอย่างกว้างขวาง โดยเฉพาะอย่างยิ่งความจำเป็นที่จะต้องใช้ข้อมูลร่วมกันระหว่างสำนักงาน ความต้องการประยุกต์แบบใหม่ ๆ บนอินเทอร์เน็ตจึงได้รับการพัฒนาเพื่อรองรับการสื่อสารรูปแบบต่าง ๆ เช่น การใช้โทรศัพท์บนเครือข่าย การติดต่อด้วยเสียงระบบวิดีโอคอนเฟอเรนซ์ การกระจายสัญญาณเสียงหรือภาพบนเครือข่าย และสิ่งหนึ่งที่มีการพัฒนาการ คือ ระบบการสื่อสารด้วยเสียงผ่านเครือข่าย IP จนสามารถใช้งานได้ดีขึ้นเพื่อได้รับประโยชน์มากที่สุดและมีความสะดวกมากขึ้น

การสื่อสารด้วยเสียงผ่านเครือข่ายอินเตอร์เน็ต เป็นระบบที่นำเอาสัญญาณข้อมูลเสียงที่เป็นสัญญาณอนาล็อก (Analog Signal) ที่นำมาแปลงเป็นสัญญาณดิจิตอล (Digital signal) แล้วบรรจุลงในไอพีแพ็คเกจ (IP Packet) เพื่อส่งผ่านไปทางเครือข่ายอินเตอร์เน็ต โดยใช้โปรโตคอลทีซีพี/ไอพี (Transmission Control Protocol / Internet Protocol: TCP/IP)

จากที่กล่าวมาข้างต้นโครงงานนี้จะทำการศึกษาการส่งข้อมูลเสียงระหว่างเครื่องคอมพิวเตอร์ 2 เครื่อง โดยผ่านเครือข่ายอินเตอร์เน็ตด้วยโปรแกรมวิชวลซีพลัสพลัส (Visual C++) ที่ทำหน้าที่ส่งและรับข้อมูลเสียงด้วยเวลาจริง (Real Time Protocol: RTP) ที่ทำให้ผู้ใช้สามารถสนทนากันได้แบบเวลาจริง เช่นเดียวกับการสนทนากันทางโทรศัพท์ รวมทั้งศึกษาข้อมูลด้านการติดต่อกับระบบเสียง วินซ็อก และการบีบอัดเสียงด้วยอัลกอลิทึมตามมาตรฐาน ITU-T (G.7 Series) ซึ่งทั้งหมดนี้เป็นความรู้ที่จำเป็นในการเขียนโปรแกรมติดต่อระหว่างเครื่องคอมพิวเตอร์ในเครือข่ายอินเตอร์เน็ต

ประโยชน์ของโครงงานนี้คือ ค่าใช้จ่ายในการสื่อสารจะถูกกว่าเมื่อเทียบกับการใช้โทรศัพท์ทางไกลตามปกติ

บทที่ 2
ทฤษฎีและหลักการ

2.1 ทฤษฎีระบบเครือข่าย

2.1.1 ระบบเครือข่าย (Network System)

การพัฒนาโปรแกรมประยุกต์บนระบบเครือข่ายจำเป็นต้องมีความรู้พื้นฐานทางด้านระบบเครือข่ายพอสมควร ในหัวข้อนี้จะอธิบายหลักการของระบบเครือข่ายที่จำเป็นสำหรับการทำความเข้าใจในการพัฒนาโปรแกรมประยุกต์บนระบบเครือข่ายที่จะกล่าวดังต่อไปนี้

2.1.1.1 ความหมายของระบบเครือข่ายคอมพิวเตอร์และอินเตอร์เน็ตเวิร์คกิ้ง(Internetworking)

ระบบเครือข่ายคอมพิวเตอร์ (Networking) คือระบบเชื่อมต่อระหว่างระบบปลายทาง (End System) ซึ่งระบบปลายทางเป็นระบบที่อิสระจากกัน (Autonomous) ระบบปลายทางสามารถเป็นไปได้ตั้งแต่ไมโครคอมพิวเตอร์ (Microcomputer) ไปจนกระทั่งซูเปอร์คอมพิวเตอร์ (Super computer) ขนาดใหญ่ เพื่อจุดมุ่งหมายในการแลกเปลี่ยนข้อมูลและการแบ่งข้อมูลในการแบ่งปันทรัพยากรของระบบ เช่น ไฟล์ข้อมูล เครื่องพิมพ์ (Printer) โมเด็ม (Modem) ตลอดจนการให้บริการฐานข้อมูลร่วม (Sharing Database)

อินเตอร์เน็ตเวิร์คกิ้งหรือ อินเตอร์เน็ต (Internet) คือ การเชื่อมต่อระบบเครือข่าย 2 เครือข่ายขึ้นไป ดังนั้น คอมพิวเตอร์บนระบบเครือข่ายหนึ่งก็สามารถติดต่อกับคอมพิวเตอร์บนระบบเครือข่ายอื่นๆได้

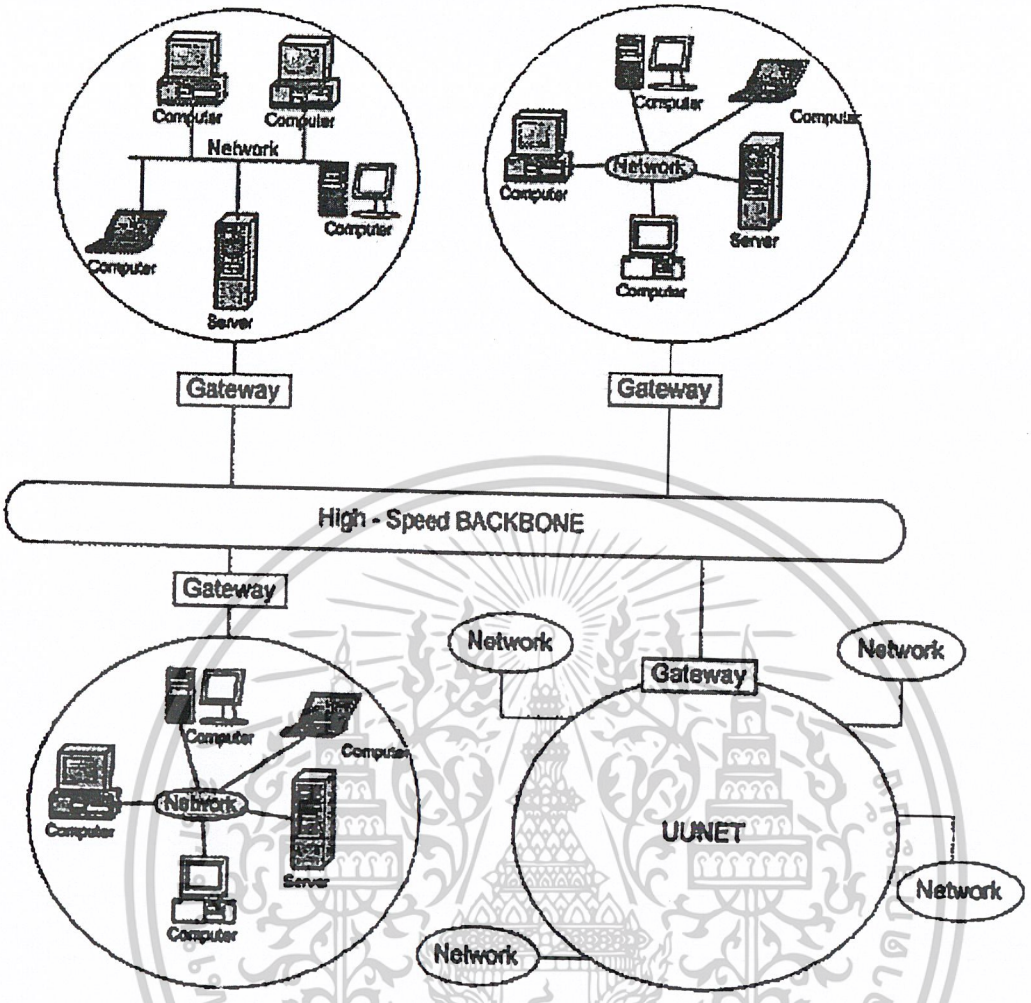
2.1.1.2 สถาปัตยกรรมของอินเตอร์เน็ต

คำกล่าวถึงอินเตอร์เน็ตว่าเป็นระบบเครือข่ายที่ใหญ่ที่สุดในโลก เครือข่ายที่มีการต่อเสมือนกับใยแมงมุมครอบคลุมโลก อินเตอร์เน็ตเชื่อมโยงโลกเข้าด้วยกันอย่างไร้มิติ จากคำกล่าวเหล่านี้ทำให้เราควรรู้ว่าอินเตอร์เน็ตจัดการส่วนประกอบต่างๆ อย่างไร จึงทำให้แต่ละส่วนสามารถรับส่งข้อมูลและทำงานสัมพันธ์กันได้อย่างดี เราจึงจำเป็นต้องศึกษาสถาปัตยกรรมของอินเตอร์เน็ตดังต่อไปนี้

อินเตอร์เน็ตประกอบด้วยสายสื่อสารความเร็วสูงโดยใช้สายโทรศัพท์ตามมาตรฐาน TS เป็นแบคโบน (Back bone) สามารถวิ่งด้วยความเร็ว 44.736 เมกะบิตต่อวินาที

เครือข่ายที่ต้องการเชื่อมโยงโดยตรงกับอินเตอร์เน็ตจะต่อกับแบคโบน ด้วยอุปกรณ์ที่เรียกว่า "เกตเวย์" (Gateway) ซึ่งเป็นเครื่องคอมพิวเตอร์ที่ทำหน้าที่ควบคุมการผ่านเข้า-ออกของข่าวสารระหว่างเครือข่ายกับแบคโบน เกตเวย์ทุกตัวสามารถกำหนดการติดต่อกับเกตเวย์ตัวอื่น หรือเครือข่ายอื่นได้โดยใช้ไอพีแอดเดรส (IP Address) ของเครือข่ายอ้างอิงถึงกัน เช่น อินเตอร์เน็ตประเทศไทยเป็นเกตเวย์ของเครือข่ายในกลุ่มติดต่อกับเกตเวย์ของ UUNET ที่รัฐเวอร์จิเนียร์ สหรัฐอเมริกา

ข่าวสารจากเครือข่ายถูกส่งออกไปผ่านเกตเวย์เข้าสู่อินเตอร์เน็ต โดยที่เกตเวย์เป็นตัวเลือกทิศทางการเดินทางเพื่อไปยังปลายทางที่ต้องการ แต่ตามเส้นทางอาจจะต้องผ่านเกตเวย์อีกหลายตัว เพื่อรับช่วงส่งข่าวสารจนถึงที่หมาย ถึงแม้ว่าจะต้องเดินทางระยะไกลก็ตาม แต่ด้วยสายสื่อสารความเร็วสูงทำให้การส่งข่าวสารทำได้อย่างรวดเร็ว



รูปที่ 2.1 แสดงเครือข่ายอินเตอร์เน็ต

2.1.1.3 ข้อกำหนดรูปแบบของเกตเวย์ (Gateway Protocols)

เกตเวย์ต้องมีข้อมูลของเกตเวย์ตัวอื่นและรู้จักเครือข่ายปลายทาง เพื่อนำมาใช้ในการกำหนดเส้นทางที่ข่าวสารสามารถเดินทางไปถึงได้เร็วที่สุด เกตเวย์จะมีการแลกเปลี่ยนข้อมูลระหว่างกันเกี่ยวกับเส้นทางรายละเอียด สถานะเครือข่ายและคุณสมบัติเครือข่ายย่อยที่ติดต่อเข้าสู่เครือข่ายใหญ่ตามลำดับขั้นจึงต้องมีการกำหนดรูปแบบพิเศษสำหรับเกตเวย์ขึ้นมา

ข้อกำหนดรูปแบบเกตเวย์แบ่งออกตามการใช้งานได้ดังนี้

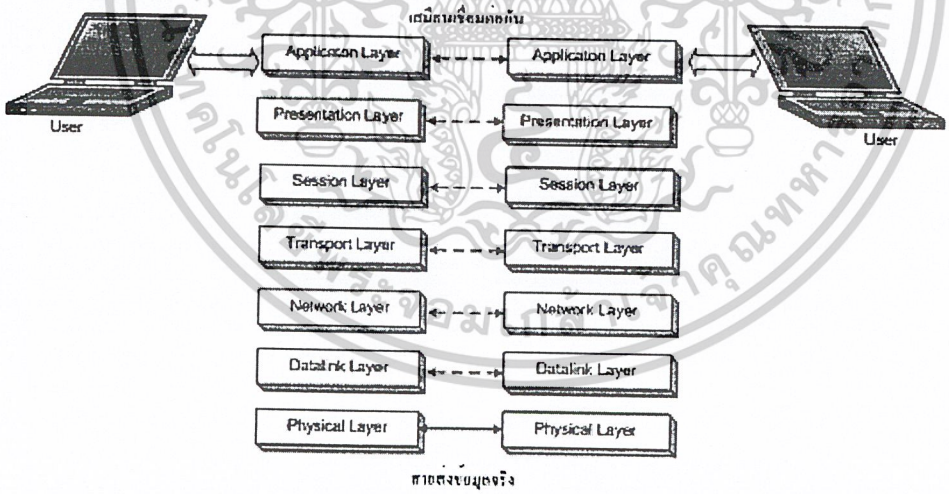
1. IGP (Interior Gateway Protocol) ถูกนำมาใช้กับเกตเวย์ที่อยู่ในเครือข่ายลูกติดต่อกับเครือข่ายลูกที่อยู่ภายในเครือข่ายแม่เดียวกัน หรือ ติดต่อกันระหว่างเครือข่ายแม่กับเครือข่ายลูก การเชื่อมต่อเกตเวย์ประเภทนี้มักจะไม่ค่อยมีการเปลี่ยนแปลง จึงเรียกว่าระบบอิสระจากกัน หรือ self complete
2. EGP (External Gateway Protocol) ในเครือข่ายใหญ่ๆการติดต่อกับเครือข่ายจะมีการเปลี่ยนแปลงอยู่เสมอตามการเปลี่ยนแปลงของเครือข่ายย่อยที่เชื่อมโยงอยู่เป็นจำนวนมาก จึงมีข้อกำหนดรูปแบบที่ใช้กับการสื่อสารระหว่างเกตเวย์ด้วยเครือข่าย EGP

3. GGP (Gateway to Gateway Protocol) การเดินทางของข่าวสารระยะทางไกลบนแบคโบนอาจจะต้องผ่านเกตเวย์หลายตัวกว่าจะถึงปลายทาง GGP เป็นข้อกำหนดรูปแบบของการสื่อสารระหว่างเกตเวย์บนแบคโบนเพื่อทำให้การจราจรบนแบคโบนไม่ติดขัดข่าวสารเคลื่อนที่ไปได้รวดเร็ว

2.1.2 โมเดลโอเอสไอ (OSI Model)

OSI Model เกิดขึ้นเพื่อลดปัญหาในความยุ่งยากสับสนในการติดต่อสื่อสารข้อมูล โครงสร้างของการสื่อสารข้อมูลภายในอุปกรณ์คอมพิวเตอร์ซึ่งส่วนใหญ่จะถูกแบ่งออกเป็นชั้นๆโดยแต่ละชั้นมีอิสระไม่ขึ้นต่อกันทำให้การแปลงบริการชั้นใดๆไม่ก่อให้เกิดปัญหากับบริการชั้นอื่น การเพิ่มเติมการบริการใหม่ทำได้โดยไม่ต้องเปลี่ยนแปลงโปรแกรมระบบเดิมและสิ่งที่สำคัญคือ การทำระดับชั้นนั้นทำให้ตัวโปรแกรมมีขนาดเล็กสามารถระบุส่วนที่ต้องการปรับปรุงได้แน่นอน ทำให้การพัฒนาประสิทธิภาพของระบบทำได้ง่ายขึ้น

OSI Model จะแบ่งออกเป็น 7 ชั้นย่อยๆ ซึ่งคอมพิวเตอร์ทั้งสองระบบจะมีขั้นตอนทั้ง 7 ชั้นนี้เหมือนกัน โดยแต่ละชั้นหรือแต่ละเลเยอร์ จะเสมือนเชื่อมต่อเพื่อส่งข้อมูลที่อยู่กับชั้นเดียวกันในคอมพิวเตอร์อีกด้านหนึ่ง แต่ในการเชื่อมต่อจริงๆแล้วจะมีเพียงเลเยอร์ที่ 1 ซึ่งเป็นชั้นล่างสุดเท่านั้นที่มีการรับส่งข้อมูลเกิดขึ้นผ่านสายส่งข้อมูลระหว่างคอมพิวเตอร์ทั้งสองเครื่อง สำหรับในเลเยอร์อื่นๆจะไม่ได้เชื่อมต่อกันจริงๆ เพียงแต่ทำงานเสมือนกับว่ามีการติดต่อรับส่งข้อมูลกับชั้นเดียวกันของคอมพิวเตอร์อีกด้านหนึ่งเท่านั้น OSI Model แต่ละชั้นมีชื่อเรียกและการทำงานดังต่อไปนี้



รูปที่ 2.2 แสดงการรับส่งข้อมูลของ OSI Model

Layer 7 Application Layer

เป็น Layer ที่อยู่บนสุดของรูปแบบ OSI ซึ่งเป็นขั้นที่ติดต่อระหว่างผู้ใช้โดยตรง ซึ่งได้แก่คอมพิวเตอร์ส่วนบุคคล (PC) แอพพลิเคชั่นใน Layer นี้สามารถนำเข้าหรือออกจากระบบเครือข่ายได้โดยไม่ต้องสนใจว่าจะ

มีขั้นตอนการทำงานอย่างไร เพราะจะมีชั้นพรีเซนเตชั่นเป็นผู้รับผิดชอบอยู่แล้ว ในรูปแบบ OSI แอปพลิเคชั่นจะทำการติดต่อกับชั้นพรีเซนเตชั่นโดยตรง

Layer 6 Presentation Layer

เป็น Layer ที่คอยรวบรวมข้อความและแปลงรหัสหรือแปลความหมายของคำสั่งที่ได้รับมาจาก Layer ที่ 7 ให้เป็นคำสั่งระดับปฏิบัติการ ส่งให้ Layer ที่ 5 ต่อไปอีกด้วย

Layer 5 Session Layer

ทำหน้าที่ควบคุมจังหวะ ในการรับส่งข้อมูลของคอมพิวเตอร์ทั้งสองด้าน ที่รับส่งแลกเปลี่ยนข้อมูลให้มีความสอดคล้องกัน (Synchronization) และกำหนดวิธีที่ใช้รับส่งข้อมูล เช่น อาจจะเป็นในลักษณะสลับกันส่ง (Half Duplex) หรือรับส่งข้อมูลพร้อมกันทั้งสองด้าน (Full Duplex)

Layer 4 Transport Layer

ทำหน้าที่เชื่อมต่อการรับส่งข้อมูลระดับสูงของ Layer 5 บางครั้งเรียกว่าขั้นเครื่องต่อเครื่อง (Host to Host) และจาก Layer 4 – Layer 7 นี้เรียกรวมกันว่า เลเยอร์เอนทูเอน (Layer End-to-End) ใน Layer ชั้นทรานสปอร์ต (Transport) นี้จะเป็นการสื่อสารระหว่างต้นทางและปลายทาง จริงๆแล้วสามารถแบ่งบริการเป็น 2 กรณี คือ

กรณีที่หนึ่ง ส่งข้อมูลระดับ Layer 5 โดยนำมาใส่ข้อมูลของปลายทาง เช่นที่อยู่แล้วส่งออกไปโดยแบ่งช่วงข่าวสารยาวๆให้เป็นหน่วยที่เล็กลง เรียกว่า แพ็จเกจ (Packet) หรือเฟรม (Frame)

กรณีที่สอง เป็นบริการที่ทำการถอดรหัสข้อมูล ที่รับเข้ามาพิจารณาตัดสินว่ามี ความสัมพันธ์เกี่ยวข้องกับข้อมูล ที่ได้ส่งออกไปหรือไม่

เลเยอร์ชั้นทรานสปอร์ต จะทำหน้าที่ตรวจสอบว่าข้อมูลที่ส่งมาจากชั้นเลเยอร์ที่ 5 นั้นไปถึงปลายทางจริงหรือไม่ ดังนั้นการกำหนดตำแหน่งของข้อมูลจึงสำคัญในชั้นนี้

Layer 3 Network Layer

เป็นชั้นที่ออกแบบหรือกำหนดเส้นทางการเดินทางของข้อมูลที่ส่ง-รับในการส่งผ่านข้อมูลระหว่างต้นทางและปลายทางซึ่งเป็นที่แน่นอนว่าในการสื่อสารข้อมูลผ่านเครือข่ายการสื่อสารจะต้องมีเส้นทางการรับส่ง ข้อมูลมากกว่า 1 เส้นทางดังนั้นในชั้นนี้จะมีหน้าที่เลือกเส้นทางที่ใช้เวลาในการสื่อสารน้อยที่สุดและระยะทางสั้นที่สุดด้วย

Layer 2 Data Link Layer

เป็นเสมือนผู้ควบคุมการรวมบิตเข้าเป็นตัวอักษรและจัดข้อมูลให้เป็น แพ็กเกจรวมทั้งตรวจทั้งตรวจสอบความผิดพลาดในการส่งข้อมูล ถ้าผู้รับได้รับข้อมูลแล้วถูกต้องก็จะส่งสัญญาณยืนยันกลับมาว่าได้รับข้อมูลแล้วเรียกว่าสัญญาณ ACK (Acknowledge) ให้กับผู้ส่งแต่ถ้าผู้ส่งไม่ได้รับสัญญาณ ACK หรือรับสัญญาณ NAK (Negative Acknowledge) กลับมาผู้ส่งอาจจะทำการส่งข้อมูลไปใหม่อีกหน้าที่ของชั้นนี้ คือป้องกันไม่ให้เครื่องส่งทำการส่งข้อมูลเร็วจนเกินขีดความสามารถของเครื่องผู้รับข้อมูลจะรับได้

Layer 1 Physical Layer

เป็นระดับชั้นการติดต่อระหว่างอุปกรณ์กับสื่อกลางของเครือข่ายจริงๆการส่งข้อมูลระดับชั้นนี้ข้อมูลมีลักษณะเป็นบิต สื่อกลางอาจเป็นสายไฟเบอร์ออปติค ไมโครเวฟ หรือ ดาวเทียม ตามระบบที่ใช้

โดยมาตรฐานสำหรับเลเยอร์ชั้นนี้จะกำหนดว่าแต่ละคอนเนคเตอร์ เช่น RS-232 มีกี่พิน แต่ละทำหน้าที่อะไรบ้าง ใช้สัญญาณไฟกี่โวลต์ เทคนิคการมัลติเพล็กซ์แบบต่างๆก็จะถูกกำหนดอยู่ในชั้นเลเยอร์นี้

ในการสื่อสารกันระหว่างระบบหรือระหว่างคอมพิวเตอร์กับคอมพิวเตอร์โปรโตคอลแบบหนึ่งก็จะถูกใช้สำหรับการสื่อสารระหว่างชั้น N คอมพิวเตอร์เครื่องหนึ่งกับชั้น N เดียวกันของคอมพิวเตอร์อีกเครื่องหนึ่งซึ่งในระหว่างการสื่อสารข้อมูลกันจริงๆนั้น โปรโตคอลชั้น N ของอุปกรณ์ทั้งสองเครื่องจะสื่อสารกันผ่านชั้นล่างซึ่งเป็นกายภายหรือวัตถุ (สื่อกลางการสื่อสาร) แต่โดยแนวความคิดของการสื่อสารข้อมูลแล้วถือว่าชั้นที่ N ของเครื่องหนึ่งกำลังติดต่อโดยตรงกับชั้น N ของอีกเครื่องหนึ่งเรียกการสื่อสารแบบนี้ว่าการสื่อสาร แบบเสมือนจริง (Virtual Communication)

โปรโตคอลของในเลเยอร์แต่ละชั้นจะแตกต่างกันออกไปแต่อย่างไรก็ตามการที่เครื่องคอมพิวเตอร์หลายๆเครื่องจะติดต่อสื่อสารกันได้แต่ละเลเยอร์ของแต่ละเครื่องจะต้องใช้โปรโตคอลแบบเดียวกันหรือถ้าใช้โปรโตคอลต่างชนิดกันก็ต้องมีอุปกรณ์ หรือซอฟต์แวร์ที่สามารถแปลงโปรโตคอลที่ต่างกันนั้นให้มีรูปแบบอย่างเดียวกัน เพื่อเชื่อมโยงให้คอมพิวเตอร์ทั้ง 2 เครื่อง สามารถติดต่อกันได้ในหนึ่งชั้นของเลเยอร์ไม่ได้ มีการกำหนดว่าจะต้องมีเพียงหนึ่งโปรโตคอลเท่านั้นที่อยู่ระดับเลเยอร์เดียวกันและในทางตรงข้าม ชุดของโปรโตคอลใดๆอาจจะมีมากกว่าหนึ่งเลเยอร์ประกอบกันเป็นข้อกำหนดของระบบเครือข่าย เรียกว่า ชุดโปรโตคอล (Protocol Suite) เป็นชุดโปรโตคอล TCP/IP เป็นต้น ประโยชน์ในการแบ่งเลเยอร์ คือ กำหนดการติดต่อระหว่างเลเยอร์ทำโดยไม่ต้องคำนึงถึงการเปลี่ยนแปลงในเลเยอร์ใดๆ ที่ติดกัน

2.1.3 ทีซีพี (Transmission Control Protocol/Internet Protocol: TCP)

อินเตอร์เน็ตนับได้ว่าเป็นเครือข่ายที่เปิดโอกาสให้เครือข่ายคอมพิวเตอร์อื่นเชื่อมโยงใช้งานหรือเป็นศูนย์กลางเชื่อมโยงเครือข่ายคอมพิวเตอร์อื่นๆ แต่หากที่เกิดขึ้นในการเชื่อมโยงเครือข่ายคอมพิวเตอร์เข้าด้วยกันก็คือ แต่ละเครือข่ายใช้คอมพิวเตอร์ต่างๆ ชนิด ต่างยี่ห้อ และระบบปฏิบัติการที่ต่างกัน มาตรฐาน ทีซีพี จึงถูกใช้เป็นกุญแจสำคัญในการแก้ปัญหาเหล่านี้ โดยกลายเปิดระบบเปิดที่สมบูรณ์แบบ มีการเชื่อมโยงคอมพิวเตอร์ได้ตั้งแต่เครื่องพีซีจนถึงเมนเฟรม และไม่จำกัดระบบปฏิบัติการที่ใช้ทีซีพี จึงเป็นมาตรฐานที่ทั่วโลกยอมรับ มีอุปกรณ์และซอฟต์แวร์ผลิตออกมาสนับสนุนทีซีพี/ไอพี (TCP/IP) มากมาย ดังนั้นจึงนับได้ว่า ทีซีพี/ไอพี เป็นหัวใจของอินเตอร์เน็ตเลยทีเดียว

2.1.3.1 ทีซีพี/ไอพี คืออะไร ?

ทีซีพี/ไอพี เป็นข้อกำหนดเกี่ยวกับรูปแบบการเชื่อมโยงในเครือข่าย (Network Protocol) จัดทำเพื่อใช้เป็นกฎเกณฑ์ให้เครื่องคอมพิวเตอร์ใช้งานร่วมกันในลักษณะของระบบเปิด (Open System) คือไม่ว่าจะเป็นคอมพิวเตอร์ชนิดใดหรือระบบใดก็ตาม จะสามารถติดต่อสื่อสารและแลกเปลี่ยนข้อมูลกันใช้ได้

ทีซีพี/ไอพี เป็นการกำหนดรูปแบบการสื่อสารระหว่างซอฟต์แวร์ การจัดโอนย้ายข้อมล การแสดงสถานะของเครื่องคอมพิวเตอร์ที่อยู่ในเครือข่าย ตลอดจนกฎระเบียบต่างๆที่กำหนดให้ทำเมื่อเกิดความผิดพลาด หรือต้องทำเพื่อป้องกันเพื่อไม่ให้เกิดความผิดพลาด

ทีซีพี/ไอพี เกิดจากการนำข้อกำหนดของรูปแบบต่างๆกันมาใช้ร่วมกันทีซีพีและไอพี (IP) ต่างก็เป็นรูปแบบหนึ่งของชุดข้อกำหนดนี้ (แต่เรียกชุดข้อกำหนดรูปแบบนี้ว่า ทีซีพี/ไอพี) ถูกออกแบบมาเพื่อใช้รับส่ง

หรือโอนย้ายข้อมลระหว่างคอมพิวเตอร์ที่อยู่บนระบบเครือข่ายเดียวกัน หรือต่างเครือข่ายกันก็ได้และมีการจัดเตรียมข้อมูลสถานะของเครือข่ายขึ้นได้ภายในข้อกำหนดรูปแบบเอง ในการสร้างซอฟต์แวร์ของระบบเครือข่ายจะใช้ทีซีพี/ไอพี เป็นส่วนสนับสนุนได้ทั้งระบบเครือข่ายเฉพาะบริเวณ (Local Area Network) และเครือข่ายบริเวณกว้าง (Wide Area Network) ไม่ได้ใช้งานเฉพาะ กับอินเตอร์เน็ตเท่านั้น

2.1.3.2 ส่วนประกอบของ ทีซีพี/ไอพี

จากที่กล่าวมาแล้วข้างต้น ทีซีพี/ไอพี ประกอบไปด้วยชุดข้อกำหนดรูปแบบต่างๆซึ่งแบ่งเป็นกลุ่มได้ดังนี้

2.1.3.2.1 กลุ่มข้อกำหนดรูปแบบการขนส่ง (Transport Protocol) ทำหน้าที่ควบคุมการเคลื่อนย้ายข้อมูล ระหว่างคอมพิวเตอร์สองเครื่อง แบ่งย่อยออกได้เป็นสองชนิด คือ

ทีซีพี เป็นการบริการแบบ คอนเน็คชั่นเบสเซอร์วิส (Connection Based Service) ซึ่งคอมพิวเตอร์ด้านรับและส่งต้องส่งต่อถึงกันอยู่ตลอดเวลาในระหว่างการสื่อสาร ถ้าเปรียบเทียบก็คล้ายกับระบบโทรศัพท์ที่ต้องติดต่อกันให้ได้ก่อนจะพูดคุยกันได้

ยูดีพี (User Datagram Protocol: UDP) เป็นการให้บริการแบบ คอนเน็คชั่นเซอร์วิส (Connection Service) คอมพิวเตอร์ด้านส่งไม่จำเป็นต้องติดต่อกับด้านรับก่อน เพียงรู้ที่อยู่ของด้านผู้รับแล้วใส่ที่อยู่นั้นไปกับข้อมูลที่ส่งออก ข้อมูลจะเดินทางตามเส้นทางต่างๆเพื่อไปถึงปลายทางตามที่อยู่ คล้ายกับการส่งจดหมายที่ไปรษณีย์จะส่งให้ตามที่อยู่ที่จ่าหน้าซองจดหมายโดยผู้ส่งและผู้รับไม่ต้องติดต่อถึงกัน

2.1.3.2.2 กลุ่มข้อกำหนดเกี่ยวกับรูปแบบเส้นทาง (Routing Protocol) ทำหน้าที่พิจารณาเส้นทางที่ดีที่สุดที่ใช้ส่งข้อมูลและถ้ามีข้อมูลเป็นจำนวนมากหรือมีขนาดใหญ่ กลุ่มข้อมูลรูปแบบนี้ก็จะทำการแบ่งย่อยข้อมูลให้มีขนาดเหมาะสมแล้วส่งออกไป เมื่อถึงผู้รับปลายทาง กลุ่มข้อมูลนี้จะทำหน้าที่ตรงข้าม คือ รวบรวมข้อมูลย่อยให้ถูกต้องก่อนการแสดงผล กลุ่มข้อกำหนดรูปแบบกลุ่มนี้ประกอบด้วย

IP (Internet Protocol) เป็นการกำหนดรูปแบบการส่งข้อมูล

ICMP (Internet Control Message Protocol) เป็นข้อกำหนดรูปแบบของข้อมูลข่าวสารเกี่ยวกับสถานะของ IP เช่นข่าวสารความผิดพลาดและผลกระทบต่อเส้นทางเมื่อมีการเปลี่ยนแปลงฮาร์ดแวร์ในเครือข่าย

RIP (Routing Information Protocol) ข้อกำหนดรูปแบบหนึ่งที่ใช้สำหรับทำการพิจารณาวิธีการเลือกเส้นทางเพื่อให้ได้เส้นทางที่ดีที่สุดเหมาะสมกับข้อมูลมากที่สุด

OSPF (Open Shortest Path First) ข้อกำหนดรูปแบบอีกประเภทหนึ่งที่ใช้ตัดสินใจเลือกเส้นทางโดยพิจารณาจากเส้นทางที่สั้นที่สุดก่อน

2.1.3.2.3 กลุ่มข้อกำหนดรูปแบบเกี่ยวกับที่อยู่เครือข่าย (Network Address) ทำหน้าที่พิจารณาที่อยู่ของเครือข่ายและเครื่องคอมพิวเตอร์ ไม่ว่าจะเป็นลักษณะตัวเลขหรือชื่อก็ตาม เพื่อความถูกต้องของข้อมูลที่จะไปยังผู้รับปลายทาง โดยที่ไม่ว่าเครือข่ายจะใหญ่โตสักเพียงใดหรือมีเครื่องคอมพิวเตอร์จำนวนมากก็ตาม ที่อยู่ต้องไม่ซ้ำกัน กลุ่มข้อกำหนดรูปแบบมีดังนี้

ARP (Address Resolution Protocol) ข้อกำหนดรูปแบบที่พิจารณาตัวเลขที่อยู่เพื่อไม่ให้เกิดที่อยู่ซ้ำกัน

DNS (Domain Name System) ข้อกำหนดรูปแบบที่พิจารณาตัวเลขที่อยู่เมื่อรู้ชื่อของโครงข่ายหรือเครื่องคอมพิวเตอร์ เพราะการใช้งานจริงนั้นใช้เพียงที่อยู่ที่เป็นตัวเลข แต่ระบบชื่อจัดทำขึ้นเพื่อให้สะดวกต่อการใช้งานของผู้ใช้

RARP (Reverse Address Resolution Protocol) ข้อกำหนดรูปแบบที่พิจารณาตัวเลขที่อยู่เดียวกับ ARP แต่จะทำงานตรงข้ามกับ ARP

2.1.3.2.4 กลุ่มข้อกำหนดรูปแบบเกี่ยวกับเส้นทางการสื่อสารระหว่างเครือข่าย (Gateway Protocol) และสนับสนุนข้อมูลสถานะเพื่อนำไปใช้เลือกเส้นทางที่เหมาะสม ข้อกำหนดรูปแบบเหล่านี้ประกอบไปด้วย

EGP (Exterior Gateway Protocol) ข้อกำหนดรูปแบบนี้จะหาการถ่ายโอนข้อมูลเส้นทางกันระหว่าง Gateway กับเครือข่ายภายนอกเพื่อทำการสื่อสาร

GGP (Gateway to Gateway Protocol) เป็นข้อกำหนดรูปแบบที่ทำงานถ่ายโอนข้อมูลเส้นทางกันระหว่าง Gateway กับ Gateway

IGP (Interior Gateway Protocol) ข้อกำหนดในแบบที่ถ่ายโอนข้อมูลเส้นทางกันภายในเครือข่ายเดียวกัน

2.1.3.2.5 กลุ่มข้อกำหนดรูปแบบเกี่ยวกับการบริการผู้ใช้ (User Services) ผู้ใช้สามารถใช้ข้อกำหนดรูปแบบได้โดยตรง ข้อกำหนดรูปแบบนี้ประกอบไปด้วย

BOOTP (BOOT Protocol) เมื่อผู้ใช้เปิดเครื่องคอมพิวเตอร์บนเครือข่ายให้เริ่มทำงาน ข้อกำหนดรูปแบบนี้จะอ่านโปรแกรมควบคุมการทำงานจากคอมพิวเตอร์ให้บริการ (Service Computer) มาให้

FTP (File Transfer Protocol) ข้อกำหนดรูปแบบที่ให้บริการถ่ายโอนไฟล์ข้อมูลระหว่างคอมพิวเตอร์ ซึ่งอาจจะอยู่บนเครือข่ายเดียวกันหรือต่างเครือข่ายกันก็ได้

TELNET เป็นข้อกำหนดรูปแบบที่ทำให้บริการเกี่ยวกับการควบคุมการติดต่อระยะทางไกล

2.1.3.2.6 กลุ่มข้อกำหนดรูปแบบอื่นที่นอกเหนือจากกลุ่มที่จัดไว้และบริการที่สำคัญๆจัดทำไว้บนเครือข่ายที่สนใจดังนี้

NFS (Network File System) เป็นข้อกำหนดรูปแบบที่ทำให้ผู้ใช้คอมพิวเตอร์เครื่องหนึ่งสามารถเข้าใช้งานไฟล์ข้อมูลและดูไฟล์ข้อมูลซึ่งอยู่ในคอมพิวเตอร์เครื่องอื่นได้

NIS (Network Information System) เป็นข้อกำหนดรูปแบบที่ให้บริการกับ User Accounts ข้ามเครือข่าย เช่น Login และ Password

RPC (Remote Procedure Call) ข้อกำหนดรูปแบบที่อำนวยความสะดวกให้กับโปรแกรมประยุกต์ที่ใช้งานกับการควบคุมระยะทางไกล

SMTP (Simple Procedure Call) ข้อกำหนดรูปแบบที่ให้บริการถ่ายโอนจดหมายอิเล็กทรอนิกส์ (Electronic Mail) ระหว่างคอมพิวเตอร์

SNMP (Simple Network Management Protocol) ข้อกำหนดรูปแบบที่ให้บริการข่าวสารต่างๆที่แสดงสถานะของเครือข่ายและอุปกรณ์ที่ต่ออยู่บนเครือข่าย

2.1.3.3 โครงสร้างของชุดโปรโตคอล ทีซีพี/ไอพี

OSI Model	TCP/IP (Internet)
Application	Application
Presentation	
Session	
Transport	Transport
Network	Internet
Data Link	Network Interface
Physical	Physical

ตารางที่ 2.1 โครงสร้างของชุดโปรโตคอล ทีซีพี/ไอพีเปรียบเทียบกับแบบจำลองโอเอสไอ

Application Layer ในชั้นนี้ประกอบด้วยโปรแกรมประยุกต์ที่ใช้ในเครือข่ายเช่นโปรแกรมส่งถ่ายข้อมูล (File-Transfer Program) และอาจกล่าวได้ว่าโปรโตคอลทีซีพี/ไอพี ก็คือโปรโตคอลในชั้นแอปพลิเคชั่นรวมกับชั้นพรีเซนเตชั่นของ OSI โมเดลนั่นเอง

Transport Layer ในชั้นนี้ประกอบด้วยชั้นที่ให้การส่งข้อมูลจากจุดปลายเปรียบเทียบได้กับชั้นทรานสปอร์ตเลเยอร์นั่นเอง โดยโปรโตคอลทีซีพี/ไอพีมีซ็อกเก็ต (Socket) เป็นปลายทาง (End-Point) ในการสื่อสาร ซึ่งซ็อกเก็ตนี้ประกอบไปด้วย หมายเลขของคอมพิวเตอร์และหมายเลขพอร์ต (Port) ของเครื่องต้องการส่งข้อมูลไปถึง ในชั้นนี้มีการรับรองให้ถึงที่หมายและลำดับของข้อมูลที่ส่งโดยปราศจากข้อมูลซ้ำซ้อนโดยในชั้นนี้มีโปรโตคอลหลัก 2 ตัว คือทีซีพีและยูดีพี

Internet Layer ในชั้นนี้มีการกำหนดดาต้าแกรม (Datagram) และทำการหาเส้นทางการส่งการทำงานในชั้นนี้จะเป็นแบบ Connectionless เนื่องจากไม่มีการเชื่อมต่อระหว่างต้นทางกับปลายทางก่อนดาต้าแกรมแต่ละตัวสามารถเลือกเส้นทางไปโดยอิสระและไม่มีการรับประกันความถูกต้องข้อมูลหรือลำดับการส่ง

Network Interface Physical Layer ทำหน้าที่ควบคุมตัวกลางที่ใช้สื่อสารข้อมูลและรูปแบบการเชื่อมต่อในทางกายภาพ ชั้นนี้จะแบ่งข้อมูลออกเป็นส่วนๆรียกว่า เฟรม หรือแพ็คเกจและส่งข้อมูลที่ได้ไปยังปลายทางที่เชื่อมต่ออยู่บนเครือข่ายเดียวกัน

2.1.3.4 ข้อแตกต่างระหว่างชุดโปรโตคอลทีซีพี/ไอพี

2.1.3.4.1 ลำดับการติดต่อสื่อสารของชั้นเลเยอร์ ในรูปแบบโอเอสไอ นั้นจะกำหนดลำดับชั้นการสื่อสารที่เป็นลำดับขั้นตอนการติดต่อที่แน่นอน โดยเฉพาะการอินเตอร์เฟสระหว่างชั้นเลเยอร์ซึ่งทำให้รูปแบบโอเอสไอ สามารถเป็นระบบเปิดสำหรับเครือข่ายคอมพิวเตอร์ทั่วไป เพราะไม่ว่าจะเป็นการเปลี่ยนแปลงโปรโตคอลในเลเยอร์ชั้นใดก็ตามจะไม่มีผลกระทบต่อการสื่อสารเลเยอร์ชั้นถัดไป ในขณะที่ชุดโปรโตคอล

ทีซีพี/ไอพี จะไม่มีการกำหนดรูปแบบการติดต่อที่ตายตัวเพื่อให้ผู้ออกแบบเครือข่ายมีอิสระสามารถเปลี่ยนแปลงโครงสร้างของเครือข่ายได้ง่ายขึ้น

2.1.3.4.2 การติดต่อสื่อสารระหว่างเครือข่ายหรือการติดต่อผ่านอินเตอร์เน็ต คือ การติดต่อสื่อสารข้อมูลระหว่างระบบคอมพิวเตอร์ 2 ระบบที่ไม่สามารถติดต่อสื่อสารกันได้โดยผ่านทางเครือข่ายการสื่อสารข้อมูลเพียงเครือข่ายเดียวได้ต้องอาศัยเครือข่ายตั้งแต่ 2 เครือข่ายขึ้นไปในการติดต่อสื่อสารและเครือข่ายเหล่านี้อาจจะมีลักษณะของเครือข่ายที่ต่างกันก็ได้

ความแตกต่างในเรื่องของอินเตอร์เน็ตระหว่างชุดโปรโตคอลทีซีพี/ไอพี กับรูปแบบโอเอสไอ ก็คือในชุดโปรโตคอลทีซีพี/ไอพีจะใช้โปรโตคอลสำหรับอินเตอร์เน็ตที่เรียกว่า โปรโตคอลไอพี (Internet Protocol: IP) ซึ่งในรูปแบบโอเอสไอ จะเรียกว่าโปรโตคอลสำหรับการอินเตอร์เน็ตว่า โปรโตคอลเน็ตเวิร์ค

2.1.3.4.3 การบริการการเชื่อมต่อการสื่อสาร (Connection Service) ในชุดโปรโตคอลทีซีพี/ไอพี นั้นจะมีการบริการการเชื่อมต่อการสื่อสารระหว่างต้นทางและปลายทาง 2 แบบ คือ การบริการแบบ Connectionless และ Connection-Oriented เท่านั้น

2.1.3.4.4 โปรโตคอลควบคุมการจัดการสื่อสาร ในชุดโปรโตคอลทีซีพี/ไอพี จะใช้โปรโตคอลทีซีพีเป็นโปรโตคอลสำหรับควบคุมการสื่อสาร กำหนดตำแหน่งต้นทางและปลายทาง และอื่นๆกับข้อมูลซึ่งในรูปแบบโอเอสไอนั้นจะแบ่งแยกการควบคุมการสื่อสารออกจากกันโดยใช้โปรโตคอลเซสชั่นและโปรโตคอลทรานสปอร์ตตามลำดับ

2.1.3.5 ลักษณะของการติดต่อ แบ่งออกเป็น 2 ชนิดคือ

2.1.3.5.1 Connection-Oriented คือการติดต่อที่ต้องมีการเชื่อมโพรเซส (Process) ที่จะมีการส่งหรือรับข้อมูลซึ่งใช้คำว่าวงจรเสมือน (Virtual Circuit) เพราะว่าจะทำงานเสมือนมีวงจรต่ออยู่ระหว่างโพรเซส ถึงแม้ว่าข้อมูลนี้อาจจะจ่ายโครงข่ายเพ็กเกจสวิทช์ชิ่ง (Packet-Switching Network) บริการชนิดนี้ส่วนมากจะใช้ในกรณีที่มีการส่งข่าวสารต้องการมากกว่าหนึ่งข่าวสาร ดังนั้นสามารถแบ่งขั้นตอนการทำงานออกเป็น

- ขั้นการสร้างการติดต่อ (Connection Establishment)
- ขั้นการส่งผ่านข้อมูล (Data Transfer)
- ขั้นยกเลิกการติดต่อ (Connection Termination)

2.1.3.5.2 Connectionless หรือดาต้าแกรม คือจะไม่มีขั้นการสร้างการติดต่อ และขั้นการยกเลิกการติดต่อ แต่จะมีขั้นการส่งผ่านข้อมูลเพียงอย่างเดียว โดยข้อมูลซึ่งเรียกว่าดาต้าแกรมจะถูกส่งจากระบบหนึ่งไปสู่ระบบหนึ่งอย่างเป็นอิสระโดยไม่ขึ้นอยู่กับดาต้าแกรมอื่น

2.1.3.6 ความสัมพันธ์ของทีซีพีและยูดีพีกับอินเตอร์เน็ต

ตามสถาปัตยกรรมของชุดข้อกำหนดรูปแบบทีซีพี/ไอพี ระดับชั้นทรานสปอร์ต จะมีบริการส่งข้อมูลอยู่ 2 ประเภท คือทีซีพีและยูดีพี

ความแตกต่างของทีซีพีกับยูดีพีอยู่ที่วิธีการสื่อสารระหว่างเครื่องคอมพิวเตอร์ 2 เครื่อง ทีซีพีต้องสร้างการเชื่อมต่อให้ได้ก่อนแล้วจึงส่งข้อมูลและยกเลิกเส้นทางที่ต่อกันเมื่อจบการส่งข้อมูล ส่วนยูดีพีจะไม่มีการสร้างการติดต่อ แต่จะใช้ไอพีแอดเดรสของคอมพิวเตอร์ปลายทางใส่รวมกับข้อมูลแล้วส่งออกไป ข้อมูลจะเดินตามไอพีแอดเดรสด้วยเส้นทางที่ไม่แน่นอน ขึ้นอยู่กับความเหมาะสมจนกว่าจะถึงคอมพิวเตอร์ปลายทาง

ทีซีพีเป็นวิธีการสื่อสารที่มีความเชื่อมั่นได้มากกว่ายูดีพีทั้งการส่งและการตอบรับข่าวสาร วิธีของยูดีพีจะไม่มีการรับประกันว่าข่าวสารจะไปถึงผู้รับ แต่ยูดีพีมีวิธีการที่จะตรวจสอบโดยให้เครื่องคอมพิวเตอร์ปลายทางส่งข่าวสารการตอบรับกลับมาเมื่อได้รับข่าวสารที่ส่งไปให้ ทางด้านผู้ส่งจะตั้งเวลาว่านานเท่าใดควรมีการตอบรับกลับมา ถ้าเกินเวลา (Time-out) แล้วยังไม่มีการตอบรับกลับมาก็สันนิษฐานได้ว่าข่าวสารสูญหายยูดีพีก็จะทำการส่งข่าวสารเดิมให้ใหม่ เครือข่ายขนาดใหญ่ เช่น อินเตอร์เน็ตซึ่งมีคอมพิวเตอร์บนเครือข่ายเป็นจำนวนมาก การใช้ทีซีพีที่ต้องสร้างการติดต่อและยกเลิกอยู่บ่อยๆจึงเป็นไปไม่ได้ในทางปฏิบัติดังนั้นอินเตอร์เน็ต จึงใช้การส่งข่าวสารด้วยยูดีพีแทน

ลักษณะความแตกต่างของโปรโตคอล	โปรโตคอลทีซีพี	โปรโตคอลยูดีพี
1.ความถูกต้องของการรับส่งข้อมูล	มีการตรวจสอบข้อมูลทั้งสองฝั่ง	ไม่มีการตรวจสอบข้อมูลในการรับส่งข้อมูล
2. ความเร็วในการรับส่งข้อมูล	ช้ากว่าเนื่องจากต้องเสียเวลาในการตรวจสอบข้อมูล	เร็วกว่าเนื่องจากส่งข้อมูลอย่างเดียวไม่มีการตรวจสอบ
3. ประหยัดทรัพยากร	ต้องอาศัยทรัพยากรมากกว่าเนื่องจากมีข้อมูลในส่วนต่างๆ มาก ไม่ว่าจะเป็นส่วน Header ส่วน Checksum และ ส่วน Acknowledge	อาศัยทรัพยากรน้อยกว่าเนื่องจากมีส่วน Header ที่น้อยกว่าและไม่ต้องมีส่วนการตรวจสอบ

ตารางที่ 2.2 แสดงความแตกต่างของโปรโตคอลทีซีพีและยูดีพี

การบริการที่ดำเนินการด้วยชุดข้อกำหนดรูปแบบทีซีพี/ไอพี ต้องออกแบบการให้บริการเป็นทีซีพีหรือยูดีพีอย่างใดอย่างหนึ่งเท่านั้น การบริการอย่างเดียวจะให้บริการทั้งสองประเภทไม่ได้ เช่น การ Login จากระยะไกล (Telnet) และการโอนย้ายไฟล์ข้อมูล FTP ใช้การให้บริการแบบทีซีพี ส่วน TFTP (Trivial File Transfer Protocol) ใช้การให้บริการแบบยูดีพี

2.1.3.7 พอร์ตและซ็อกเก็ต (TCP Port and Socket)

การใช้บริการของทีซีพีทำโดยผ่านทางพอร์ต โดยมีการกำหนดหมายเลขให้กับการบริการต่างๆ การบริการที่ต่างกันจะมีหมายเลขพอร์ตไม่เหมือนกัน

พอร์ตถูกกำหนดตามชนิดของการบริการ คือ ต้องกำหนดหมายเลขให้กับการบริการต่างๆ การบริการที่ต่างกันจะมีหมายเลขพอร์ตไม่เหมือนกัน เช่น ต้องการสื่อสารกันด้วย Telnet เครื่องคอมพิวเตอร์ที่ต้องการจะส่งความต้องการออกทางพอร์ตหมายเลข 23 ที่จัดไว้สำหรับบริการ Telnet ไปยังเครื่องคอมพิวเตอร์ที่ต้องการติดต่อด้วย ถ้าคอมพิวเตอร์ปลายทางยอมรับการติดต่อ การสื่อสาร Telnet ก็จะเริ่มขึ้น ในช่วงเวลานี้ถ้ามีความต้องการการสื่อสารด้วย Telnet จากเครื่องคอมพิวเตอร์อื่นเข้ามา จะไม่สามารถทำการติดต่อได้เนื่องจากพอร์ต

หมายเลข 23 Telnet จากเครื่องคอมพิวเตอร์อื่นเข้ามา จะไม่สามารถทำการติดต่อได้เนื่องจากพอร์ตหมายเลข 23 สำหรับบริการ Telnet ถูกใช้งานอยู่ทีซีพีพอร์ตเป็นพอร์ตทางโลจิกคอล (Logical) ไม่ใช่พอร์ตที่อยู่ด้านหลังของเครื่องคอมพิวเตอร์ที่ใช้คอนเน็คเตอร์ (Connector) บนแบคโบนเพื่อให้สามารถสื่อสารบริการหลายประการได้พร้อมๆกัน

ซ็อกเก็ตเป็นหมายเลขกำหนดวงจรซึ่ง ทางเข้า-ออก ของระดับชั้นทีซีพีของเครื่องคอมพิวเตอร์บนเครือข่าย หมายเลขซ็อกเก็ตประกอบด้วยเลขไอพีแอดเดรส กับหมายเลขพอร์ตรวมกัน หมายเลขซ็อกเก็ตที่ใช้กับการสื่อสารระหว่างเครื่องคอมพิวเตอร์ที่อยู่ต่างเครือข่ายได้เนื่องจาก ไอพีแอดเดรสของเครือข่ายจะไม่ซ้ำกัน ถึงแม้ว่าหมายเลขพอร์ตจะซ้ำกันได้

เครื่องคอมพิวเตอร์บนเครือข่ายจะสร้างตารางพอร์ตขึ้นมาเก็บรายการของพอร์ตที่มีอยู่เพื่อใช้ตรวจสอบว่าจะใช้บริการพอร์ตที่ต้องการได้หรือไม่และพอร์ตมีสถานะเป็นอย่างไร การที่พอร์ตที่อยู่ในรายการตารางพอร์ตของแต่ละเครื่อง ร้องขอการสื่อสารซึ่งกันและกัน เรียกว่า 'Binding' ของพอร์ต และวิธีที่ทำให้พอร์ตที่ถูกใช้สามารถใช้สื่อสารได้อีกจากการขอใช้บริการเดียวกันเรียกว่า 'Multiplexing'

TCP พอร์ต	บริการ
20	FTP Data
21	FTP Control
23	Telnet
25	Simple Mail Transfer Protocol
53	Domain Name Server
69	Trivial File Transfer Protocol
79	Finger

ตารางที่ 2.3 หมายเลขพอร์ตที่ให้บริการต่างๆ

2.1.3.8 ข้อกำหนดรูปแบบ IP (Internet Protocol)

ไอพีเป็นข้อกำหนดรูปแบบหนึ่งในชุดข้อกำหนดรูปแบบทีซีพี/ไอพี ลักษณะการให้บริการเป็นแบบคอลเนคชั่นเลส เช่นเดียวกับยูดีพี มีหน้าที่หลักเกี่ยวกับการจัดการดาต้าแกรมระหว่างเครื่องคอมพิวเตอร์ เช่น ที่อยู่ปลายทาง วิธีหาเส้นทางที่ดีที่สุดที่จะไปถึงปลายทาง และจัดเตรียมวิธีการแก้ปัญหาที่อาจจะเกิดขึ้นในกรณีต่างๆ เป็นต้น นอกจากนี้ยังมีหน้าที่แบ่งย่อยดาต้าแกรมที่มีขนาดใหญ่เกินกว่าที่ไอพีเมจเสจ (IP Message) กำหนดไว้ (ประมาณ 64 กิโลไบต์) และรวบรวมดาต้าแกรมย่อยๆกลับมาเป็นเมจเสจ (Message) ให้เหมือนเดิมที่เครื่องปลายทาง การแบ่งย่อยและรวบรวมดาต้าแกรม มีอยู่ 4 วิธีคือ

1. Segmentation วิธีแบ่งดาต้าแกรมขนาดใหญ่ให้เป็นดาต้าแกรมย่อย
2. Reasemble วิธีรวบรวมดาต้าแกรมย่อยๆ ให้กลับอยู่ในสภาพเดิม
3. Concatenation วิธีรวมดาต้าแกรมให้เป็นบล็อก
4. Separation วิธีแบ่งบล็อกให้เป็นดาต้าแกรม

2.1.3.9 ข้อกำหนดรูปแบบ ICMP (Internet Control Message Protocol)

ปัญหาหนึ่งที่เกิดขึ้นได้ง่ายกับดาต้าแกรม คือ ดาต้าแกรมเดินทางผิดเส้นทาง ทำให้สูญหายหรือข่าวสารบางส่วนเสียหาย เครื่องคอมพิวเตอร์ด้านส่งต้องรู้เงื่อนไขที่จะทำให้เกิดความผิดพลาดต่างๆบนเครือข่ายเพื่อให้ดาต้าแกรมถูกส่งไปอย่างถูกต้อง

ICMP จะรายงานความผิดพลาดในระบบให้กับไอพีพิเศษ ICMP มี Header เหมือนกับข่าวสารไอพีอื่นๆ โดยจะถูกส่งรวมไปกับข่าวสารอื่นสู่เครื่องปลายทาง ซึ่งเครื่องปลายทางจะส่งรายงานความผิดพลาดกลับมาให้เครื่องด้านส่งได้รู้ว่าเกิดอะไรขึ้นบ้างในการเดินทาง ซึ่งข้อมูลในรายงานจะถูกนำมาใช้แก้ปัญหาต่อไป

2.1.4 ไอพีแอดเดรส (IP Address)

เครื่องคอมพิวเตอร์ที่ต่อเชื่อมกับอินเตอร์เน็ตหรือเครือข่ายที่ใช้ทีซีพี/ไอพี มีอยู่เป็นจำนวนมาก เครื่องคอมพิวเตอร์แต่ละเครื่องต้องสามารถระบุ หรืออ้างอิงได้โดยไม่เกิดความซ้ำซ้อนกับเครื่องคอมพิวเตอร์อื่น มิฉะนั้นแล้วข่าวสารที่เครือข่ายรับมาจะไม่สามารถส่งไปให้กับเครื่องคอมพิวเตอร์ที่ต้องการข่าวสารนั้นได้ จึงต้องมีการจัดระบบที่ดี เครือข่ายอินเตอร์เน็ตหรือเครือข่ายที่ใช้ทีซีพี/ไอพี ได้ออกแบบการจัดการระบบตรงส่วนนี้ไว้แล้ว เครือข่ายอินเตอร์เน็ตใช้รหัสหมายเลขมากำหนดให้แต่ละเครือข่ายและเครื่องคอมพิวเตอร์ภายในเครือข่ายที่เชื่อมโยงเรียกรหัสหมายเลขนี้ว่า อินเตอร์เน็ตแอดเดรส (Internet Address)

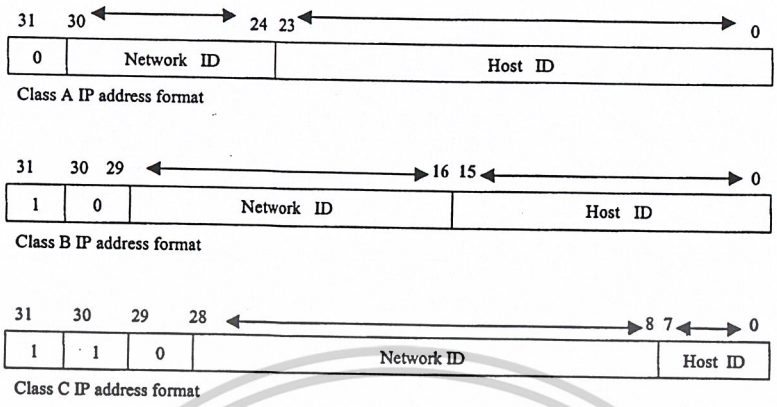
ไอพีแอดเดรส ประกอบด้วยเลขฐานสองจำนวน 32 บิต แบ่งออกเป็น 4 ส่วน แต่ละส่วนมี 8 บิต เมื่อดูเฉพาะแต่ละส่วนเป็นเลขฐานสิบจะได้เลขจำนวน 256 ค่าไม่ซ้ำกัน (0-255) ไอพีแอดเดรสจะนำเอาหมายเลขทั้ง 4 ส่วนมารวมกัน โดยแยกแต่ส่วนด้วยจุด ดังนั้นหมายเลขทั้งหมดที่เป็นไปได้ โดยไม่ซ้ำกันคือ 256^4 หรือ 4,294,967,296 จำนวน มีค่าหมายเลขจาก 000.000.000.000 จนถึง 255.255.255.255 หมายเลขเหล่านี้เองที่อินเตอร์เน็ตใช้กำหนดให้กับเครือข่ายและเครื่องคอมพิวเตอร์เพื่อใช้อ้างอิงถึง

ไอพีแอดเดรสบางหมายเลขสงวนไว้ใช้ ด้วยจุดหมายกรณีพิเศษ ทำให้ไอพีแอดเดรสที่ใช้งานทั่วไปลดลงจากจำนวนที่เป็นไปได้ ความหมายของไอพีแอดเดรสจะแบ่งได้เป็น 2 กลุ่มดังนี้

- กลุ่มที่ใช้เป็นรหัสประจำเครือข่าย
- กลุ่มที่ใช้เป็นรหัสประจำเครื่องคอมพิวเตอร์ที่อยู่ภายในเครือข่าย (Host Computers) ไอพีแอดเดรสในกลุ่มรหัสประจำเครื่องคอมพิวเตอร์สามารถซ้ำกันได้ แต่กลุ่มรหัสประจำเครือข่ายจะซ้ำกันไม่ได้ ดังนั้นรหัสเครื่องที่ซ้ำกันจึงไม่มีผลต่อการอ้างอิง

นอกจากนี้เพื่อความเหมาะสมในการกำหนดไอพีแอดเดรสให้กับผู้ขอ ทางผู้บริหารเครือข่ายอินเตอร์เน็ตแบ่งคลาสของผู้ขอไอพีแอดเดรสตามขนาดของเครือข่ายเพื่อให้ทรัพยากรส่วนนี้ถูกใช้อย่างคุ้มค่า

ที่สุด องค์กรขนาดใหญ่ก็จะจัดให้อยู่ในคลาสที่สามารถกำหนดไอพีแอดเดรสให้กับเครื่องคอมพิวเตอร์ในเครือข่ายได้มาก การแบ่งคลาสจะแบ่งได้ดังนี้



รูปที่ 2.3 แสดงคลาสของไอพีแอดเดรส

การกำหนดหมายเลขของเครือข่ายและเครื่องคอมพิวเตอร์นอกจากจะแบ่งไอพีแอดเดรส เป็นคลาสทั้ง 3 ประเภทแล้ว ยังมีข้อกำหนดปลีกย่อยอีกหลายประการที่ผู้วางระบบต้องรู้ไว้ เพื่อที่จะกำหนดแอดเดรสใช้งานได้อย่างถูกต้องไม่ผิดพลาด ได้แก่ ไอพีแอดเดรส

ที่เป็น '0' ทุกบิต จะไม่ใช้งานทั่วไป แต่นำไปใช้กับอุปกรณ์หาเส้นทาง (Router) เพื่อกำหนด 'Default Route'

1. ไอพีแอดเดรสในแต่ละส่วนคือ ส่วนของหมายเลขเครือข่ายและส่วนของหมายเลขประจำเครื่องคอมพิวเตอร์จะเป็น '0' หรือ '1' ทุกบิตไม่ได้

- ถ้าส่วนของหมายเลขประจำเครื่องคอมพิวเตอร์เป็น '0' ทุกบิต หมายถึง หมายเลขเครือข่ายนั้นใช้งานร่วมกับอุปกรณ์หาเส้นทางหรือเพื่อบอกให้เครื่องคอมพิวเตอร์รู้ว่าตนเองอยู่ในเครือข่ายใดแอดเดรส
- ส่วนของหมายเลขประจำเครื่องคอมพิวเตอร์เป็น '1' ทุกบิต หมายถึง หมายเลขเครือข่ายนั้นใช้สำหรับการกระจายข่าวภายในเครือข่าย (Broadcast Address)
- ถ้าไอพีแอดเดรสทั้งสองส่วนเป็น '1' ทุกบิต หมายถึง แอดเดรสที่ใช้กระจายข่าวหรืออีกนัยหนึ่งเครื่องคอมพิวเตอร์ที่ส่งแอดเดรสนี้เข้าสู่ระบบไม่ทราบว่าตนเองอยู่ในเครือข่ายใด

2. ไอพีแอดเดรสของคลาส A หมายเลข 127.0.0.0 จะสงวนไว้ใช้งานเฉพาะอย่าง เช่น IPC (Inter-Process Communication)

จากรูปที่ 2.3 จะแสดงประเภทของไอพีแอดเดรสและข้อกำหนดปลีกย่อยต่างๆ เราสามารถทราบได้ว่าเครือข่ายขององค์กรถูกจัดอยู่ในคลาสใด โดยดูจากค่าหมายเลขของ 8 บิตแรกซ้ายมือสุดดังนี้

คลาส A ไอพีแอดเดรสอยู่ในช่วง 1 ถึง 126

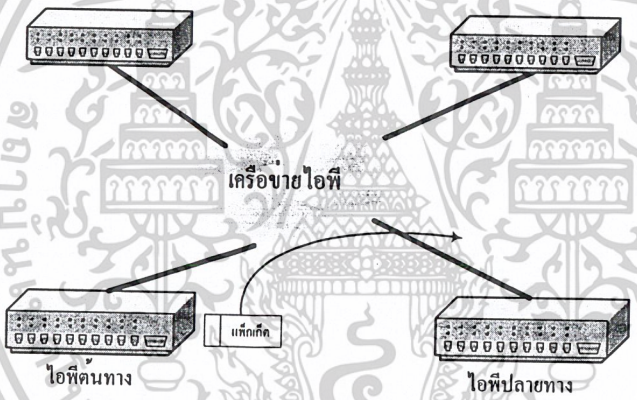
คลาส B ไอพีแอดเดรสอยู่ในช่วง 128 ถึง 191

คลาส C ไอพีแอดเดรสอยู่ในช่วง 192 ถึง 233

2.2 วีโอไอพี (Voice over IP)

เมื่ออินเตอร์เน็ตมีการใช้งานกว้างขวางขึ้น ความต้องการประยุกต์แบบใหม่ๆบนอินเตอร์เน็ตจึงได้พัฒนาขึ้น โดยเฉพาะอย่างยิ่งการใช้อินเตอร์เน็ต เป็นโครงสร้างพื้นฐานเพื่อรองรับการสื่อสารรูปแบบต่างๆ เช่น การใช้โทรศัพท์บนเครือข่าย การติดต่อด้วยเสียง ระบบวีดีโอคอนเฟอเรนซ์ การกระจายสัญญาเสียงและภาพบนเครือข่าย และสิ่งหนึ่งที่มีการพัฒนาการประยุกต์จนสามารถใช้งานได้ดี คือ ระบบการสื่อสารด้วยเสียงผ่านเครือข่ายไอพี หรือวีโอไอพี (VoIP)

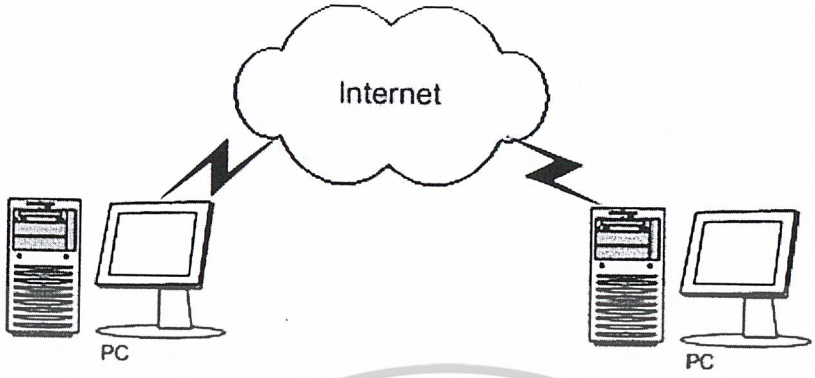
ไอพีหรืออินเตอร์เน็ตโปรโตคอล เป็นโปรโตคอลที่อยู่ในชั้นเน็ตเวิ์คซึ่งพัฒนามาจากรากฐานระบบการสื่อสารแบบแพ็คเกจ โดยระบบมีการกำหนดแอดเดรสที่เรียกว่าไอพีแอดเดรส การส่งข่าวสารจากไอพีแอดเดรส (IP Address) หนึ่งไปยังอีกไอพีแอดเดรสหนึ่งนั้นใช้หลักการบรรจุข้อมูลใส่แพ็คเกจ แล้วส่งไปในเครือข่าย ระบบการจัดส่งแพ็คเกจกระทำด้วยอุปกรณ์สื่อสารจำพวกเราน์เตอร์ (router) มีหลักการส่งแบบไปรษณีย์สมัยเก่า บางทีเราจึงเรียกการส่งแบบนี้ว่า คาต้าแกรม การสื่อสารแบบไอพีแพ็คเกจเข้าไปในเครือข่ายโดยไม่มีการประกันว่าแพ็คเกจนั้นจะถึงปลายทางเมื่อไร



รูปที่ 2.4 แสดงการส่งแพ็คเกจผ่านเครือข่าย

วีโอไอพีเป็นเทคโนโลยีสำหรับการส่งข้อมูลเสียงผ่านระบบอินเตอร์เน็ต ซึ่งต่างจากระบบโทรศัพท์แบบเก่าที่ส่งสัญญาณเสียงผ่านชุมสาย วิธีนี้ทำโดยนำข้อมลเสียงมาทำการแปลงสัญญาณดิจิตอลแล้วรวมเป็นแพ็คเกจ แล้วจึงส่งแพ็คเกจที่ได้ผ่านอินเตอร์เน็ต ไปโดยโปรโตคอลที่ได้ทำการกำหนดไว้ โดยรูปแบบของการใช้งานวีโอไอพีทั่วไปสามารถแบ่งออกได้เป็น 3 ลักษณะ คือ

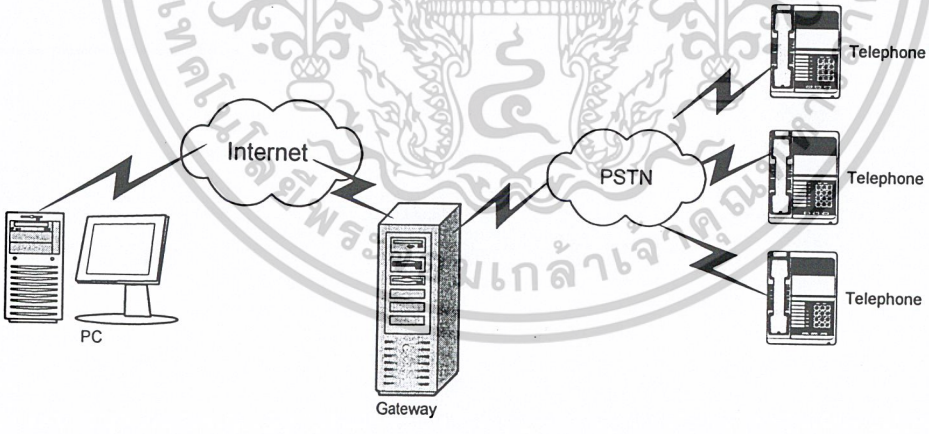
2.2.1 ระหว่างพีซีกับพีซี (PC-To-PC Connection)



รูปที่ 2.5 แสดงลักษณะการติดต่อระหว่างพีซีกับพีซี

ลักษณะการทำงานเป็นการใช้คอมพิวเตอร์ 2 เครื่องในการเชื่อมต่อ โดยใช้ผ่านทางซอฟต์แวร์ หรือใช้งานผ่านทางเว็บในการส่งข้อมูลระหว่างกัน ข้อมูลที่เป็นเสียงที่เป็นอนาล็อกจะถูกแปลงให้เป็นข้อมูลดิจิตอลผ่านทางซาวนด์การ์ดแล้วจึงนำข้อมูลที่ได้มาทำการบีบอัด แล้วส่งผ่านโปรโตคอลทีซีพี/ไอพี เพื่อส่งผ่านทางอินเตอร์เน็ตไปยังเครื่องคอมพิวเตอร์ปลายทางที่ต้องการจะคุยด้วย

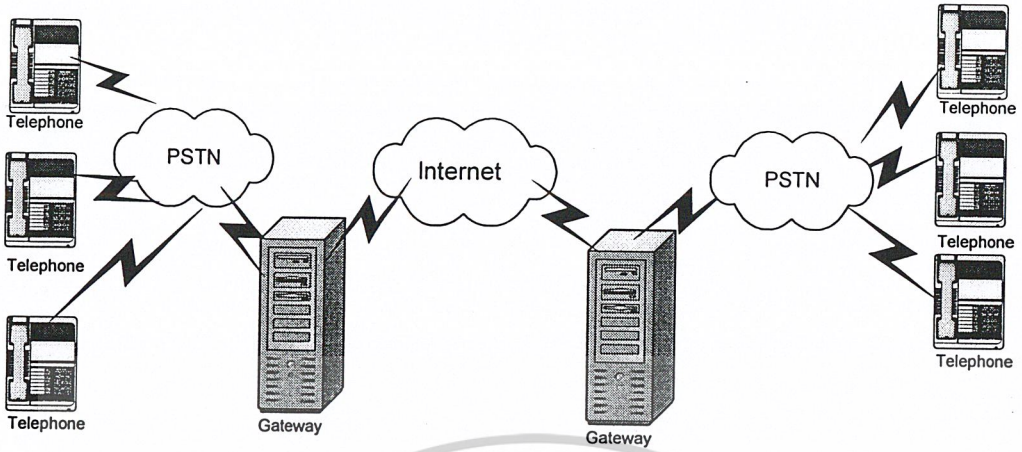
2.2.2 ระหว่างพีซีกับเครื่องโทรศัพท์ (PC-To-Phone Connection)



รูปที่ 2.6 แสดงลักษณะการติดต่อระหว่างพีซีกับเครื่องโทรศัพท์

ลักษณะการทำงานเป็นการนำเครื่องคอมพิวเตอร์ ทำการติดต่อไปยังระบบโทรศัพท์ที่ใช้กันอยู่ตามบ้าน (Public Telephone Network: PSTN) ในส่วนของคอมพิวเตอร์จะใช้งานผ่านทางซอฟต์แวร์หรือผ่านทางเว็บที่มีให้บริการ มีลักษณะการทำงานคล้ายกับแบบพีซีกับพีซี แต่จะมีข้อแตกต่างตรงที่ปลายทางที่จะทำการติดต่อไปยังระบบโทรศัพท์พื้นฐาน ซึ่งต้องเพิ่มขึ้นมาคือ เกตเวย์ สำหรับแปลงข้อมูลทางอินเตอร์เน็ตไปเป็นสัญญาณในระบบโทรศัพท์ไปยังเครื่องปลายทาง

2.2.3 ระหว่างโทรศัพท์กับโทรศัพท์ (Phone-To-Phone Connection)



รูปที่ 2.7 แสดงลักษณะการติดต่อระหว่างโทรศัพท์กับโทรศัพท์

ลักษณะการทำงานเป็นระบบที่มีการนำเอาโทรศัพท์ระบบเดิมมาใช้ แต่ได้มีการเปลี่ยนแปลงเพิ่มเติม โดยการเพิ่มเกตเวย์ขึ้นที่ชุมสายโทรศัพท์ เพื่อทำการแปลงสัญญาณข้อมูลในระบบโทรศัพท์ไปเป็นข้อมูล เพื่อส่งผ่านทางอินเตอร์เน็ต และที่ปลายทางก็จะมีเกตเวย์ ทำการแปลงข้อมูลที่ส่งผ่านอินเตอร์เน็ต แปลงกลับไปเป็นสัญญาณโทรศัพท์อีกครั้ง

2.2.4 ประโยชน์ของวีโอไอพี

- ประหยัดค่าใช้จ่ายในการติดต่อในระยะทางไกล เช่น การติดต่อระหว่างประเทศ

- มีการเรียกใช้แบนด์วิดท์ น้อยกว่าเพราะว่าลดความจำเป็นต้องเช่าคู่สายโทรศัพท์เพิ่มจากเดิมเพื่อใช้ในการโทรศัพท์อีกต่อไป เมื่อนำระบบโทรศัพท์ผ่านอินเตอร์เน็ต มาใช้ทำให้เราสามารถใช้งานบนระบบเครือข่ายที่มีอยู่แล้วในบริษัทได้

- มีการให้บริการที่ดีกว่าและมากกว่าการใช้งานแบบเก่าเพราะบริการต่างๆ ที่มีอยู่ในตู้สาขาโทรศัพท์ (PBX) ที่ใช้ตามบริษัทนั้น ถ้านำระบบโทรศัพท์ผ่านอินเตอร์เน็ตมาใช้ก็ยังคงให้ บริการต่างๆนั้นได้ปกติและยังสามารถประยุกต์ใช้งานตามความสามารถของระบบเครือข่ายและระบบอินเตอร์เน็ต ที่ต่ออยู่ได้เต็มที่ เช่น บริษัทหนึ่งมีเว็บ และแสดงเบอร์โทรศัพท์ติดต่อไว้ ซึ่งขณะนั้นลูกค้ากำลังดูเว็บเพ็จแลสนใจในตัวสินค้า ซึ่งลูกค้าสามารถติดต่อแต่ไม่ต้องตัดอินเตอร์เน็ตออกเพื่อจะโทรติดต่อ เพื่อคุยติดต่อกับพนักงานโดยตรงผ่านทางอินเตอร์เน็ตเลย ซึ่งมีผลต่อความอยากสั่งซื้อสินค้า

- เป็นการใช้ประโยชน์จากไอพีได้คุ้มค่าและง่ายเนื่องจากมีการใช้ไอพีเป็นพื้นฐานอยู่ทั่วไปอยู่แล้วไม่ว่าจะเป็นระบบเน็ตเวิร์คหรือระบบอินเตอร์เน็ต ในเมื่อมีอยู่ทั่วไปอยู่แล้ว เราก็ควรจะใช้ประโยชน์จากระบบไอพีที่มีอยู่ให้คุ้มค่า

2.3 การติดต่ออินเตอร์เน็ตของวิชวลซีพลัสพลัสโดยใช้ MS Winsock Control

MS Winsock Control เป็นคอนโทรลที่จะทำการใช้งานทีซีพีและยูดีพีได้ง่ายขึ้น ซึ่งคอนโทรลตัวนี้สามารถที่จะใช้ได้ทั้ง MS Visual Basic, MS Visual C++ ซึ่งทำให้เราสามารถที่จะสร้างการติดต่อได้โดยไม่จำเป็นต้องทราบรายละเอียดเกี่ยวกับทีซีพีหรือวิธีที่จะทำการเรียกใช้ Window API ระดับล่างเลยเราเพียงแค่กำหนด Properties ต่างๆ แล้วเรียกใช้วิธีการเหล่านี้ เราก็จะสามารถสร้างการติดต่อสื่อสารกับเครื่องอื่นๆ แล้วแลกเปลี่ยนข้อมูลข่าวสารซึ่งกันและกัน

2.3.1 คุณสมบัติของ Winsock Control (Properties)

Property นี้จะใช้สำหรับการกำหนดคุณสมบัติต่างๆให้กับวินซ็อก เช่น เลขหมายเครื่องปลายทาง เลขหมายพอร์ต ที่เราต้องการจะติดต่อหรือจะเป็นการเรียกดูคุณสมบัติของวินซ็อกในขณะนั้น เช่นดูว่า ขณะนั้นวินซ็อกมีสถานะเป็นอย่างไร อ่านค่าหมายเลขที่อยู่ IP หรือค่าพอร์ตของเราเอง เป็นต้น

ชื่อ Property	Return Value	อ่านได้อย่างเดียว	คำอธิบาย
BytesRecived	Long	ใช่	เป็นการคืนค่ากลับจำนวนของไบต์ที่รอ ในการรับจากบัฟเฟอร์ ซึ่งใช้ GetData method ในการรับข้อมูล
LocalHostName	String	ใช่	เป็นการคืนค่า ชื่อที่อยู่ของเรา เพื่อให้เรารู้ว่าเป็นอะไร
LocalIP	String	ใช่	คืนค่าของ Local host ที่อยู่ IP ในรูปแบบของ String เช่น 198.164.0.94 เป็นต้น
LocalPort	Long	ไม่ใช่	กำหนดพอร์ตที่รับข้อมูลของเครื่องเรา เช่น 1000 ถ้ามีข้อมูลส่งมาที่ IP เราตรงดับ LocalPort ในที่นี้ คือ 1000 เราก็จะได้รับข้อมูลนี้จากวินซ็อกตัวที่มี LocalPort อยู่ที่ 1000
Protocol	Long	ไม่ใช่	เป็นการตั้งค่าให้กับวินซ็อกว่าจะใช้โปรโตคอลอะไรซึ่งมีให้เลือก คือ sckTCPProtocol และ sckUDPProtocol (TCP และUDP) ซึ่งการคืนค่าคือ 0 และ 1 ตามลำดับ
RemoteHost	String	ไม่ใช่	คืนค่าหรือตั้งเครื่องปลายทางที่ติดต่อ ซึ่งสามารถที่จะตั้งให้เป็นแบบ String หรือ เลขหมายที่อยู่ IP ก็ได้แบบ String เช่น www.kmitl.ac.th และ แบบเลขหมายที่อยู่ก็เช่น 161.246.10.21
RemoteHostIP	String	ใช่	คืนค่าที่อยู่ IP ของเครื่องปลายทาง สำหรับ TCP นี้จะได้ค่าเมื่อการติดตั้งเสร็จแล้วเท่านั้น ส่วน UDP จะคืนค่านี้ได้เมื่อเกิดเหตุการณ์ DataArrival, ซึ่งตอนนั้นก็จะได้รับค่าที่อยู่ IP ของเครื่องที่ส่งข้อมูลมาด้วย
RemotePort	Long	ไม่ใช่	จะคืนค่าหรือติดตั้งค่าพอร์ต เพื่อที่เราจะส่งข้อมูลไป

SocketHandle	Long	ใช่	คืนค่าที่ตรงกับที่การใช้ SocketHandle
State	Integer	ใช่	คืนค่าสถานะของตัวควบคุม ซึ่งจะมีเป็นแต่ละชนิดไป

ตารางที่ 2.4 คุณสมบัติวินซ็อกคอลโทรล

2.3.2 คุณลักษณะสถานะ (State Property)

คุณลักษณะสถานะจะส่งค่าสถานะของวินซ็อกเป็นค่าๆออกมาว่าขณะนั้นวินซ็อกตัวนั้นอยู่ในสถานะใด อาจจะเอาสถานะตัวนี้ไปเป็นเงื่อนไขในการเขียนโปรแกรมเพื่อป้องกันการเกิดความผิดพลาดที่เกี่ยวกับข้อบังคับของวินซ็อกได้ หรือจะใช้เพื่อดูว่าขณะนี้กำลังทำอะไรอยู่ เพื่อที่เราจะได้เขียนโปรแกรมได้ว่า ถ้าเป็นอย่างนี้จะทำอะไรได้บ้าง ซึ่งโดยที่รายละเอียดทั้งจะระบุตามตาราง

ค่า Constant	ค่า	ความหมาย
SckClosed	0	ปิด เป็นค่าปกติ
SckOpen	1	เปิด ซ็อกเก็ต
SckListening	2	การรอรับรู้สำหรับการเชื่อมต่อ
SckConnectionPending	3	การร้องขอการเชื่อมต่อมาถึงแล้วแต่ยังไม่เสร็จ
SckResolvingHost	4	Host ทำตามความต้องการ
SckHostResolved	5	ความต้องการของ Host เป็นจริง
SckConnecting	6	การร้องขอการติดต่อเริ่มทำ แต่ยังไม่เสร็จ
SckConnected	7	การเชื่อมต่อสำเร็จ
SckClosing	8	เริ่มต้นการปิด
SckError	9	มีการผิดพลาดเกิดขึ้น

ตารางที่ 2.5 คุณลักษณะสถานะ

2.3.3 Methods ของ Winsock Control

Methods นี้จะเป็นวิธีการที่จะทำให้วินซ็อกทำตามความต้องการของเราในการเขียนโปรแกรมการกำหนดก็จะมีลักษณะที่ขึ้นต้นด้วยชื่อของวินซ็อกแล้วตามด้วยวิธีที่จะให้วินซ็อกทำ เช่น ต้องการให้วินซ็อกที่ชื่อ winsock1 สร้างการเชื่อมต่อ ก็จะใช้คำสั่งดังนี้ Winsock1.connect

การที่จะใช้ Method นี้ส่วนใหญ่จะต้องมีกำหนด คุณสมบัติ ให้กับวินซ็อกก่อนจึงต้องระวังด้วยว่าเราได้กำหนดให้มันหรือยัง ตัวอย่างก็เช่นการที่เราจะใช้ Winsock1.connect ได้นั้นต้องมีการกำหนดหมายเลข IP และเลขหมาย Port ปลายทางให้กับ วินซ็อกก่อน เป็นต้น ซึ่งรายละเอียดต่างๆของ Method ก็อยู่ที่ตารางข้างล่าง

Method	พารามิเตอร์	Return Value	คำอธิบาย
Accept	RequestID	Void	สำหรับการติดต่อแบบ TCP เท่านั้นใช้ Method นี้ ในการยอมรับการติดต่อเมื่อเกิดเหตุการณ์ ConnectionRequest
Bind	LocalPort LocalIP	Void	ทำการเชื่อมต่อซ็อกเก็ตให้กับ LocalPort และ LocalIP ใช้ Bind ถ้ามี Network หลายตัว Bind จะต้องถูกเรียกก่อน Listen
Close	None	Void	ปิดการติดต่อ หรือ ปิดการรอรับการติดต่อ
Connect	RemoteHost RemotePort	Void	สร้างการเชื่อมต่อของ TCP ไปยังหมายเลข RemotePort ที่อยู่บน RemoteHost
GetData	Data Type MaxLen	Void	รับเอาข้อมูลที่ค้างอยู่เข้ามา มี Type และ Maxlen เป็นพารามิเตอร์ควบคุม พารามิเตอร์ Type จะใช้กำหนดชนิดของข้อมูลสำหรับอ่าน พารามิเตอร์ MaxLen จะใช้กับจำนวนไบต์ หรือ ตัวอักษรที่รับเข้ามา GetData จะข้ามพารามิเตอร์ Maxlen สำหรับชนิดอื่นๆที่มากกว่า ไบต์อะเรย์ และstring.
Listen	None	Void	สร้างซ็อกเก็ตและวางมันไว้ในโหมดการรอรับการติดต่อ Listen จะใช้สำหรับการติดต่อแบบ TCP เท่านั้น
PeekData	Data Type MaxLen	Void	คุณสมบัติเหมือนกับ GetData ยกเว้นแต่ว่าข้อมูลไม่ได้ถูกย้ายจากบัฟเฟอร์ของระบบ
SendData	Data	Void	ส่งข้อมูลไปยังเครื่องปลายทาง ถ้า UNICODE string ผ่าน มันก็จะถูกแปลงเป็น ANSI string ก่อน จำไว้ว่าต้องใช้ ไบต์อะเรย์สำหรับข้อมูลไบนารีเสมอ

ตารางที่ 2.6 Methods ของวินซ็อกคอนโทรล

2.3.4 Events ของวินซ็อก

Event คือเหตุการณ์ที่เกิดขึ้นเมื่อมีลักษณะของซ็อกเก็ตจากภายในและภายนอกเปลี่ยนไปจากเดิม เช่น การเข้ามาของข้อมูล การทำการติดต่อสำเร็จ

เราจะใช้ประโยชน์ของ Events ในการที่เรากำหนดให้กับโปรแกรมว่าเราต้องการให้โปรแกรมทำอะไรเมื่อเกิดเหตุการณ์นี้ขึ้น เช่นสมมุติว่า เมื่อมีข้อมูลเข้ามาถึงซ็อกเก็ต (ในที่นี้คือ Data Arrival) ของเราให้เรารับข้อมูลนี้ไว้ (โดยใช้ GetData Method) ซึ่งแต่ละ Event ก็มีดังตารางข้างล่างนี้

Event	Arguments	คำอธิบาย
Close	ไม่มี	เกิดขึ้นเมื่อขาดการติดต่อกันกับเครื่องปลายทางที่เราติดต่อด้วย
Connect	ไม่มี	เกิดขึ้นเหลังจาก Connect method ติดต่อได้สำเร็จแล้ว
ConnectionRequest	RequestID	เกิดขึ้นเมื่อเครื่องปลายทางต้องการจะติดต่อด้วย
DataArrival	BytesTotal	เกิดขึ้นเมื่อมีข้อมูลใหม่เข้ามา
Error	Number Description Scode Source HelpFile HelpContext CancelDisplay	เมื่อวินซ็อกมีข้อผิดพลาด Event นี้ก็จะถูกสร้างขึ้น
SendComplete	ไม่มี	เกิดขึ้นเมื่อคำสั่งการส่งทำได้สำเร็จเรียบร้อยแล้ว
SendProgress	BytesSent bytesRemaining	เกิดขึ้นขณะที่ข้อมูลทำการส่ง

ตารางที่ 2.7 Event ของวินซ็อก

2.4 เครือข่ายระบบสื่อสารแพ็คเกจเสียง

ในปัจจุบันเครือข่ายโทรศัพท์ได้รับพัฒนาและใช้งานอย่างต่อเนื่องทำให้เครือข่ายโทรศัพท์ในปัจจุบันเป็นเครือข่ายที่เชื่อมโยงกันทั่วโลก ทำให้สามารถติดต่อกันอย่างกว้างขวาง แต่เนื่องจากเครือข่ายโทรศัพท์ยังมีต้นทุนสูงในเชิงค่าใช้จ่าย ทำให้ต้องหาวิธีลดต้นทุน เพิ่มประสิทธิภาพ โดยเฉพาะวิธีทางเทคโนโลยีสมัยใหม่หลายได้เข้ามามีบทบาทที่ทำให้การใช้งานกว้างขวางและยังลดต้นทุนค่าใช้ค่าจ่ายให้ถูกลงอีกด้วย

แนวคิดที่จะทำให้ระบบสื่อสารถูกลงคือ การจัดระบบสื่อสารเป็นรูปแพ็คเกจ เพื่อว่าข้อมูลที่รับส่งกันเป็นกลุ่มก้อนทำให้เกิดการจัดการที่จะบริหารแถบกว้างให้แบ่งกันใช้เพื่อได้ประสิทธิภาพสูงสุด เช่นเครือข่ายอินเตอร์เน็ต

2.4.1 เหตุผลในการใช้แพ็คเกจเสียง

แรงขับดันที่ทำให้เทคโนโลยีแพ็คเกจเสียงเป็นจริงขึ้นมา ทั้งนี้เพราะเทคโนโลยีอื่นเข้ามามีบทบาทและเกี่ยวข้องอย่างมาก ได้แก่

- ฮาร์ดดิสก์ มีความจุมากขึ้น ขณะที่ราคาลดลง
- ความเร็วของซีพียูที่เพิ่มขึ้น
- การใช้หน่อยความจำในเครื่องคอมพิวเตอร์มากขึ้น
- ความเร็วของตัวซีดีรอมเพิ่มมากขึ้น

สำหรับในระบบการสื่อสารแล้ว การเปลี่ยนสัญญาณอนาล็อกเป็นดิจิตอล เพื่อให้การประมวลผลข้อมูลและสื่อสารง่าย ซึ่งปัจจุบันมี DSP ใช้ประมวลสัญญาณความเร็วสูง เช่น โมเด็ม

2.5 การบีบอัดเสียง(Voice Compression)

2.5.1.ประเภทของการเข้ารหัสเสียงพูด (Classification of speech Coder)

แบ่งได้เป็น 2 พวกใหญ่ๆ

1. การเข้ารหัสสียงตามรูปคลื่น (Waveform Coding)

2. การเข้ารหัสเสียงตามลักษณะของเสียงพูดหรือโวโคดเดอร์ (VoCoder) เป็นการเข้ารหัสที่เน้นในเรื่องคุณภาพในการรับฟังเสียงพูดโดยไม่จำเป็นต้องได้สัญญาณที่เหมือนเดิมทุกประการโวโคดเดอร์สามารถทำงานที่อัตราข้อมูลต่ำมากได้โดยให้คุณภาพของเสียงที่ระดับเสียงสังเคราะห์ โดยที่ในอัตราข้อมูลที่สูงขึ้นก็จะให้คุณภาพของเสียงที่ดีขึ้น

ต่อมามีรูปแบบการเข้ารหัสเสียงแบบต่างๆ เกิดขึ้นมากมาย ซึ่งคงไม่สามารถกล่าวถึงในที่นี้ได้หมด ที่จะพูดต่อไปนี้จะเป็นเรื่องโวโคดเดอร์ ซึ่งใช้หลักการกำเนิดเสียงที่เลียนแบบการกำเนิดเสียงพูดของมนุษย์ โดยเน้นที่เรื่องโวโคดเดอร์แบบการทำนายพันธะเชิงเส้น (Linear Predictive Vocoder หรือ LPC) เนื่องจากเป็นเทคนิคที่มีการพัฒนาอย่างกว้างที่สุดในช่วง 2 ทศวรรษหลังนี้

มาตรฐานการสร้างรหัสของข้อมูลเสียงจึงมีหลายมาตรฐาน โดย ITU (International Telephony Union) เป็นผู้กำหนดโดยจัดอยู่ในข้อแนะนำที่เรียกว่า G-series ซึ่งประกอบด้วยมาตรฐานสำคัญคือ

- G.711 เป็นมาตรฐานที่ใช้สัญญาณเข้ารหัสแบบพีซีเอ็มมีขนาด 64000 บิตต่อวินาที ใช้ในกิจการโทรศัพท์ รวมถึงการสวิตช์ในพีบีเอ็กซ์ (PBX) ทั่วไป

- G.726 เป็นมาตรฐานการบีบอัดเสียงด้วยวิธีการเอดีพีซีเอ็ม (ADPCM) ทำให้ได้ขนาดสัญญาณลดลงเป็น 40, 32, 24 และ16 กิโลบิตต่อวินาที เป็นที่นิยมใช้ในการสร้างแพ็คเกจเสียงที่ใช้ในสวิตชิงแบบพีบีเอ็กซ์

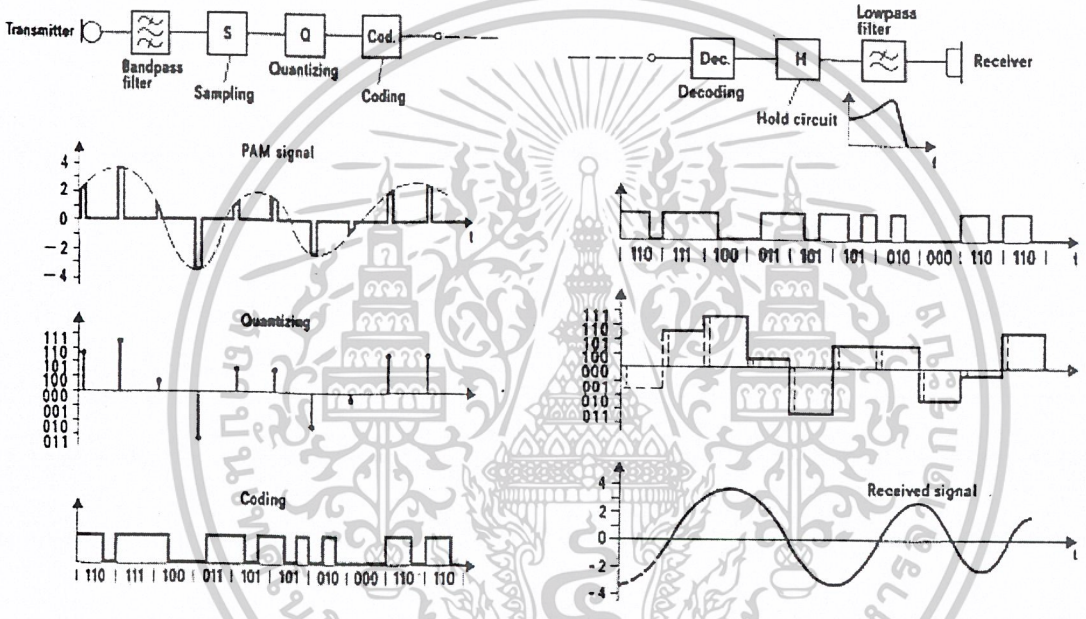
- G.728 เป็นมาตรฐานที่ใช้หลักการของ LPC ที่ใช้ชื่อว่า GELP Code Excited Linear Prediction สามารถบีบอัดสัญญาณให้เหลือเพียง 16 กิโลบิตต่อวินาที G.728 เป็นมาตรฐานที่ใช้ในการแปลงสัญญาณเสียงเพื่อส่งกันระหว่างพีบีเอ็กซ์

- G.729 ใช้ CELP เช่นกัน แต่บีบอัดข้อมูลเสียงให้เหลือเพียง 8 กิโลบิตต่อวินาที

- G.723.1 เป็นมาตรฐานที่สร้างสายสัญญาณเสียงที่ใช้ในเรื่องมัลติมีเดีย ซึ่งใช้กันในระบบการรับส่งข้อมูลที่ความเร็วไม่สูงมาก สามารถบีบอัดข้อมูลให้เหลือเพียง 5.3 และ 6.3 กิโลบิตต่อวินาที โดยคุณภาพของเสียงยังดี

2.5.2 วิธีการเข้ารหัสของสัญญาณเสียงแบบ G.711

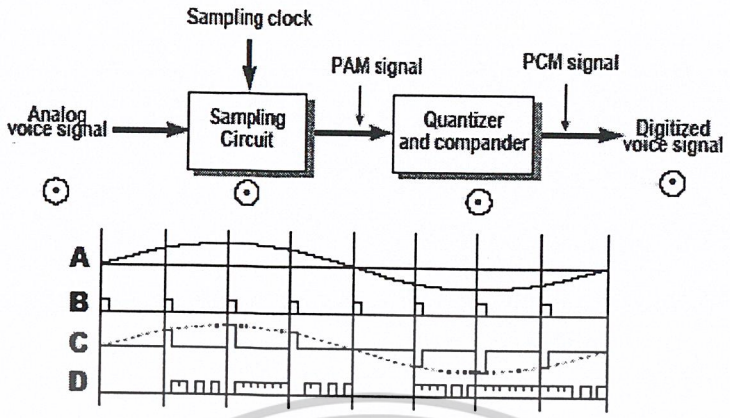
วิธีการเข้ารหัสสัญญาณเสียงในระบบสื่อสารแบบดิจิตอลที่ได้รับการพัฒนาและนำมาใช้อย่างกว้างขวางก็คือ พัลส์โค้ดมอดูเลชั่น ซึ่งระบบนี้เป็นวิธีการหนึ่ง ในการเปลี่ยนสัญญาณเสียงจากอนาลอกให้เป็นดิจิตอล โดยจะประกอบด้วยขั้นตอนต่างๆ ดังนี้



รูปที่ 2.8 กระบวนการเข้ารหัสและถอดรหัสระบบพีซีเอ็ม

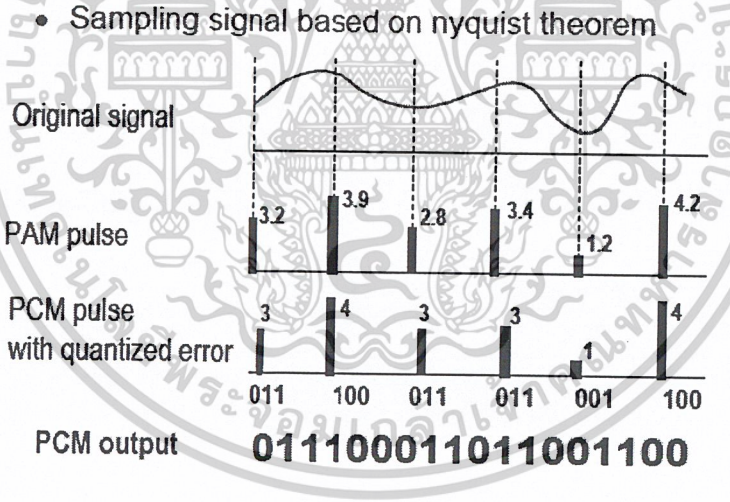
จากรูปที่ 2.8 แสดงขั้นตอนการประมวลสัญญาณเพื่อให้ได้สัญญาณพีซีเอ็ม กล่าวคือ ระบบพีซีเอ็มเป็นการจัดการกับสัญญาณพัลส์แอมปลิจูดมอดูเลชั่น (Pulse Amplitude Modulation: PAM) โดยการนำสัญญาณพีเอเอ็ม ไปทำการเข้ารหัส (coding) เป็นสัญญาณดิจิตอล แล้วจึงนำสัญญาณดิจิตอลที่ได้นั้นส่งผ่านระบบต่อไปและทางด้านรับจะทำการถอดรหัส (decoding) เป็นสัญญาณพีเอเอ็ม แล้วนำสัญญาณพีเอเอ็ม นั้นไปดีมอดูเลตเพื่อให้ได้รับสัญญาณเดิมกลับคืนมา

2.5.2.1 การแซมปลิ้ง (Sampling)



รูปที่ 2.9 วิธีการแซมปลิ้งสัญญาณเสียง

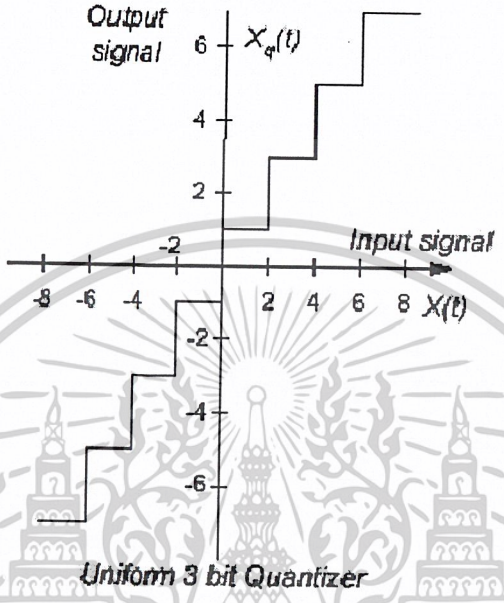
แซมปลิ้ง คือ การทำสัญญาณซึ่งมีค่าต่อเนื่องให้เป็นแบบดิสครีทในช่วงเวลาที่เท่าๆกัน จากทฤษฎีการแซมปลิ้ง



รูปที่ 2.10 กระบวนการพีซีเอ็ม

2.5.2.2 การจัดระดับ (Quantization)

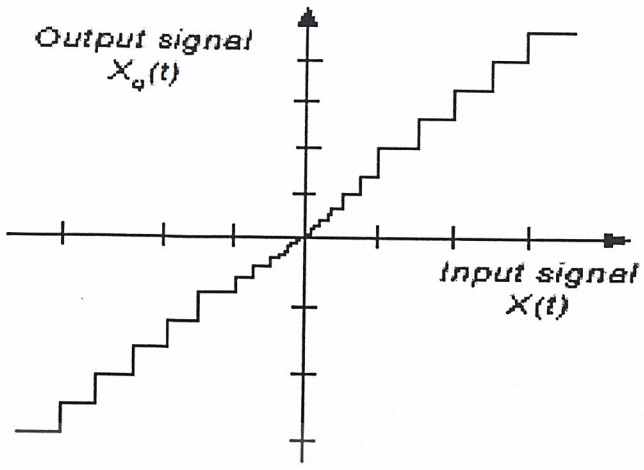
ขบวนพัลส์พีเอเอ็ม ที่ผ่านการแซมปลิ้งแล้ว ยังถือว่าเป็นชนิดอนาล็อกอยู่ คือจะมีแอมปลิจูดที่เปลี่ยนแปลงอย่างต่อเนื่องไปกับเวลาที่เป็นช่วงๆ การจัดระดับคือ กระบวนการที่เปลี่ยนแปลงแอมปลิจูดของพัลส์พีเอเอ็มเหล่านั้นให้เป็นค่าตัวเลขแบบดิสครีต ตามที่แสดงไว้ในรูป



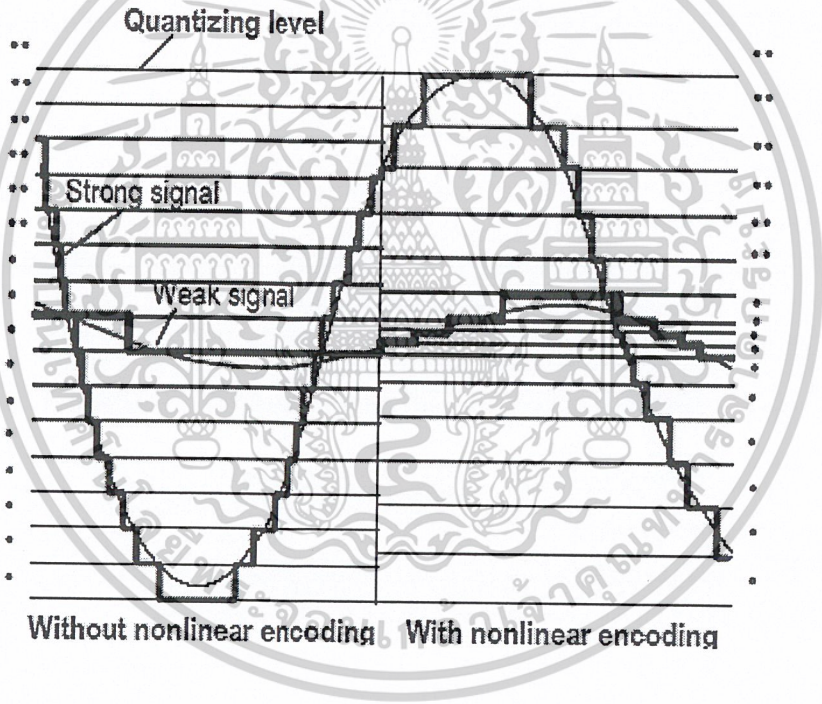
รูปที่ 2.11 การจัดระดับแบบยูนิฟอร์ม ควอนไทซ์เซอร์ (uniform quantizer)

จากรูปที่ 2.11 แอมปลิจูดของแซมเปิ้ลทุกตัวของพีเอเอ็ม จะถูกจัดให้เป็นระดับซึ่งเรียกว่า ระดับการควอนไตซ์ (quantizing level) โดยมีระยะห่างระหว่างระดับข้างเคียง เรียกว่า ควอนไทซ์ซิ่งอินเทอร์วัล (quantizing interval) เท่ากัน กรณีนี้เรียกว่าเป็นการจัดแบบยูนิฟอร์ม ขนาดของแซมเปิ้ลทุกตัวจะแสดงด้วยค่าระดับการควอนไตซ์ที่ใกล้เคียงที่สุด เช่น ขนาดของแซมเปิ้ลที่ $t = t_1$ คือ 3.2 จะจัดให้เป็นระดับ 3 หรือค่าแซมเปิ้ลที่ $t = t_2$ มีขนาด 6.2 จะจัดให้เป็น 6 เป็นต้น จะเห็นได้ว่าสัญญาณพีเอเอ็ม ที่ถูกจัดระดับแล้วนี้จะเป็นเพียงค่าโดยประมาณของสัญญาณอนาล็อกเท่านั้น ดังนั้นส่วนเกินและส่วนขาดจากการจัดระดับจึงเป็นค่าผิดพลาดระหว่างสัญญาณเดิมและค่าที่ได้จัดระดับ ซึ่งค่าผิดพลาดนี้เรียกว่า ควอนไตซ์ซิ่งนอยส์ (quantizing noise) หรือค่าผิดเพี้ยนจากการควอนไตซ์ (quantizing distortion)

จากหลักการที่กล่าวมา ในทางปฏิบัติจะไม่สามารถหลีกเลี่ยงควอนไตซ์ซิ่งน้อยส์ได้ แต่เพื่อรักษาคุณภาพเสียงในการสนทนาให้ดี จึงจำเป็นต้องทำการควอนไตซ์ซิ่งอินเทอร์วัลให้แคบลง ก็สามารถลดควอนไตซ์ซิ่งนอยส์ได้ในระดับหนึ่ง



รูปที่ 2.12 การจัดระดับแบบนอนยูนิฟอร์ม ควอนไทซ์เซอร์ (nonuniform quantizer)



รูปที่ 2.13 แสดงการควอนไตซ์ แบบการเข้ารหัสแบบไม่เชิงเส้น (Non-Linear Encoding) และการเข้ารหัสแบบเชิงเส้น (Linear Encoding)

2.5.2.3 สัญญาณรบกวนควอนไทซ์

สมรรถนะของตัวควอนไทซ์แสดงได้ด้วยค่า อัตราส่วนสัญญาณต่อสัญญาณรบกวนควอนไทซ์ (S/N_q) ถ้านิยามความผิดพลาดควอนไทซ์ให้เป็น

$$e = F(x) - x \tag{2.1}$$

แล้วค่าผิดพลาดเฉลี่ยกำลังสอง (Mean square error) จะเป็น

$$\overline{e^2} = \int_{-\infty}^{\infty} [F(x) - x]^2 \, p(x)dx \qquad (2.2)$$

โดย $p(x)$ จะเป็นความหนาแน่นของความน่าจะเป็น (Probability density function : PDF) ของสัญญาณอินพุท x

สำหรับการควอนไทซ์เชิงเส้นโดยทั่วไปแล้ว $\overline{e^2}$ จะหาได้จากผลรวมของการอินทริเกรตภายในช่วงควอนไทซ์สเตป Δv ดังนี้

$$\overline{e^2} = \sum_{j=1}^{N} \int_{j-1}^{x_j} (y_j - x)^2 \, p(x)dx \qquad (2.3)$$

โดย $F(x) = y_j$ เมื่อ x อยู่ในช่วง R_j

เนื่องจากในแต่ละช่วงควอนไทซ์สเตป y_j มีค่าคงที่ตลอด และ $p(x)$ มีค่าเดียวในแต่ละ ช่วงควอนไทซ์สเตป ดังนั้นจะแทน $p(x)$ ด้วย $p(y_j)$

$$\begin{aligned} \overline{e^2} &= \sum_{j=1}^{N} \frac{(x_j - x_{j-1})^2}{12} (x_j - x_{j-1}) p(y_j) \\ &= \frac{1}{12} \sum_{j=1}^{N} (\Delta v_j)^2 [p(y_j)\Delta v_j] \end{aligned} \qquad (2.4)$$

เทอม $p(y_j)\Delta v_j$ คือ ความน่าจะเป็นที่สัญญาณอินพุท x จะมีค่าอยู่ในช่วง x_{j-1} ถึง x_j ดังนั้น เมื่อหาผลรวมตลอดช่วง (x_0, x_n) ความน่าจะเป็นมีค่าเท่ากับ 1 และเนื่องจากเป็นการควอนไทซ์เชิงเส้น Δv จะมีค่าคงที่ตลอดช่วง (x_0, x_n) ดังนั้นสมการ จะเป็น

$$\begin{aligned} \overline{e^2} &= \frac{(\Delta v)^2}{12} \sum_{j=1}^{N} p(y_j)\Delta v \\ &= \frac{(\Delta v)^2}{12} \end{aligned} \qquad (2.5)$$

เพื่อที่จะสามารถคำนวณหาค่าสัญญาณต่อสัญญาณรบกวนควอนไทซ์ S/N_q ดังนั้นเราจึงต้องกำหนดคุณลักษณะของสัญญาณอินพุทโดย ถ้ากำหนดให้สัญญาณอินพุทเป็นสัญญาณซายน์ มีขนาดเป็น (-V,V) ดังนั้นกำลังงานเฉลี่ยของสัญญาณจะเป็น

$$S = \frac{V^2}{2} \qquad (2.6)$$

และเมื่อเราพิจารณาจากสมการ ทราบว่าขนาดของสัญญาณอินพุท (-V,V) ของตัวควอนไตซ์มีค่าเป็น $2^n \Delta v$ ดังนั้นสมการ จะเป็น

$$\overline{e^2} = \frac{2}{3} \cdot \frac{V^2/2}{2^{2n}} \qquad (2.7)$$

จากสมการที่ 2.7 จะได้ค่าอัตราส่วนสัญญาณ ต่อสัญญาณรบกวนควอนไทซ์ S/N_q ของตัวควอนไตซ์เชิงเส้น

$$\frac{S}{N_q} = \frac{S}{e^2} = \frac{V^2/2}{(2/3)[(V^2/2)/2^{2n}]} \qquad (2.8)$$

$$\frac{S}{N_q} = \left[\frac{3}{2}\right] \cdot 2^{2n}$$

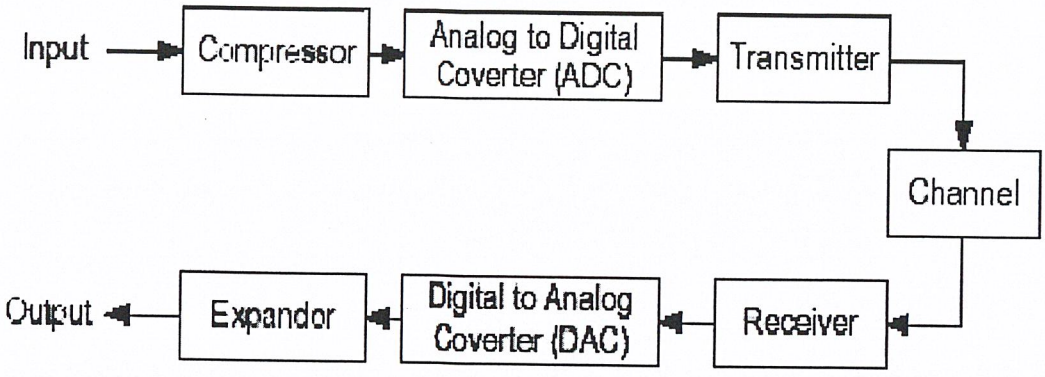
หรือในหน่วยเดซิเบลเป็น

$$\left[\frac{S}{N_q}\right]_{dB} = 6n + 1.8\ dB \qquad (2.9)$$

จากสมการนี้ทำให้ทราบว่า การเพิ่มควอนไทซ์บิตขึ้น 1 บิต จะทำให้ค่า S/N_q ดีขึ้น 6 dB

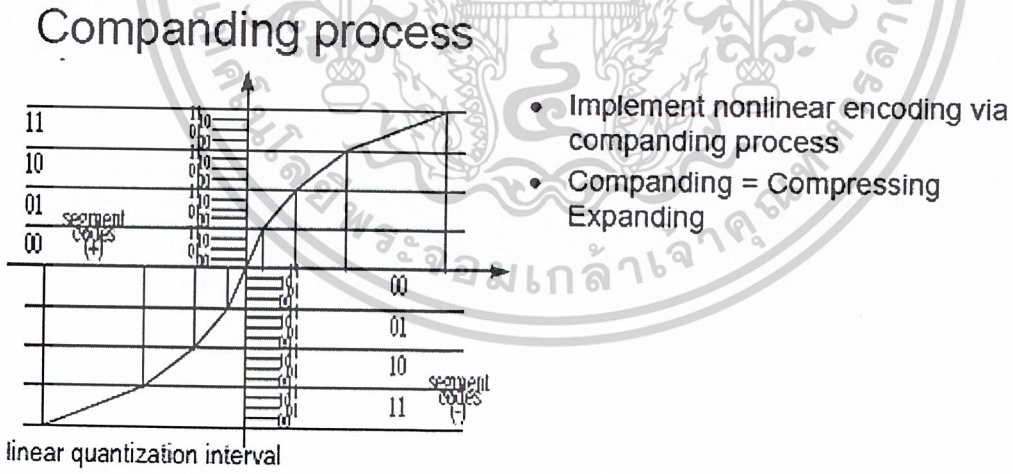
2.5.2.4 คอมแพนดิ้ง (companding)

ตามที่ได้กล่าวมาแล้วว่าเราไม่สามารถหลีกเลี่ยงควอนไตซ์ซิ่งนอยส์ได้ แต่เราสามารถทำให้มันลดลงได้โดยการลดควอนไตซ์ซิ่งอินเทอร์วัล หรือเพิ่มจำนวนระดับนั่นเอง แต่เมื่อจำนวนระดับเพิ่มขึ้นแล้ว จำนวนบิทที่ใช้ก็เพิ่มขึ้นด้วย จึงจำเป็นต้องใช้ความเร็ว ในการส่งสัญญาณดิจิตอลสูงขึ้น ตามปกติ ควอนไตซ์ซิ่งนอยส์จะเกิดขึ้นอย่างสม่ำเสมอในทุกอินเทอร์วัล โดยไม่เกี่ยวข้องกับแอมปลิจูดของสัญญาณเดิม ในกรณีที่สัญญาณมีระดับสูง คุณภาพของการเข้ารหัสของสัญญาณเสียงจะดีกว่ากรณีของสัญญาณซึ่งมีระดับต่ำ ดังนั้นจึงจำเป็นต้องพิจารณาวอนไตซ์ซิ่งนอยส์ในบริเวณที่สัญญาณมีระดับต่ำ การจัดระดับแบบนอนยูนิฟอร์ม คือ บริเวณที่สัญญาณมีแอมปลิจูดต่ำจะใช้ควอนไตซ์ซิ่งอินเทอร์วัลแคบๆ และในทางตรงกันข้ามบริเวณที่สัญญาณมีแอมปลิจูดจะใช้ควอนไตซ์ซิ่งอินเทอร์วัลกว้างๆ ซึ่งการทำให้เป็นแบบนอนยูนิฟอร์ม นั้นจะใช้หลักการคอมแพนดิ้งเข้าช่วย



รูปที่ 2.14 การแทรกขั้นตอนของการอัดสัญญาณและการยืดสัญญาณลงในระบบพีซีเอ็ม

การทำคอมแพนดิ้งเมื่อนำมาใช้ในระบบพีซีเอ็ม จะทำให้ประสิทธิภาพในการใช้จำนวนบิตในการเข้ารหัสสูงขึ้น กรณีที่จัดระดับแบบยูนิฟอร์มนั้นจะใช้ประมาณ 2000 ระดับ จึงจะสามารถรักษาคุณภาพเสียงให้ดีในการเข้ารหัสจะต้องใช้ถึง 11 บิตต่อแซมเปิ้ล 1 ตัว แต่ถ้าใช้แบบนอนยูนิฟอร์มแล้วจะใช้เพียง 7 บิต ซึ่งมีระดับเพียง 128 ระดับเท่านั้น ก็เพียงพอที่จะทำให้อัตราส่วนของสัญญาณต่อควอนไตซ์ซิ่งนอยส์ใกล้เคียงกับการจัดระดับแบบยูนิฟอร์ม ซีซีไอทีที (CCITT) กำหนดว่าให้ใช้ 8 บิตต่อแซมเปิ้ล 1 ตัวและ ระดับการควอนไตซ์เท่ากับ 256 ระดับก็จะเป็นการรับรองว่าเสียงพูดจะมีคุณภาพที่ดี



รูปที่ 2.15 กระบวนการคอมแพนดิ้ง

2.5.2.4.1 สมการการบีบสัญญาณของคอมเพรสเซอร์ (Compressor) ลักษณะเชิงล็อกกาลิทึม

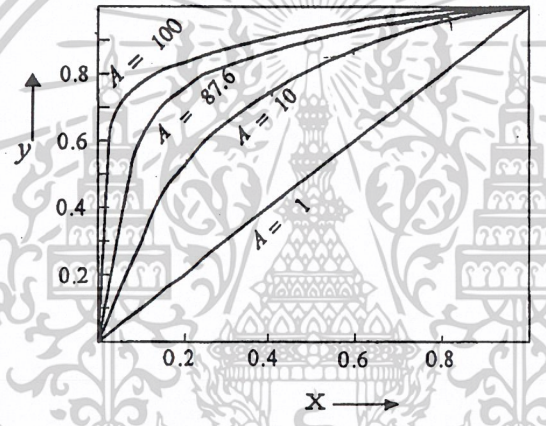
จากการวิเคราะห์สัญญาณ จะพบว่าถ้าต้องการให้ค่าเอสเอ็นอาร์ของเอาท์พุตของควอนไทเซอร์ ไม่ขึ้นกับค่ากำลังเฉลี่ยของสัญญาณอินพุตก็จะต้องทำค่าความสัมพันธ์ระหว่างเอาท์พุต และอินพุต ของวงจรบีบ

สัญญาณให้อยู่ในลักษณะเชิงลอกการิทึม รูปสมการของ y ที่เป็นเอาท์พุต ที่นิยมทำกันมีอยู่ 2 ฟังก์ชัน หรือ 2 กฎ ตามข้อแนะนำของซีซีไอทีที คือ

1. กฎเอ (A-Law) เป็นกฎหรือฟังก์ชันที่นิยมใช้กันในยุโรป และสายการสื่อสารระหว่างประเทศส่วนใหญ่ ยกเว้นอเมริกาเหนือ และญี่ปุ่น มีรูปความสัมพันธ์ดังนี้ คือ

$$Y = \frac{Ax}{1+\log A};\left(\frac{1}{A} > x > 0\right)$$
$$= 1+\frac{\log Ax}{1+\log A};(1 > x > \frac{1}{A}) \qquad (2.10)$$

โดยในที่นี้ A เป็นค่าพารามิเตอร์ที่ใช้สำหรับปรับลักษณะการบีบสัญญาณ เมื่อเลือกค่า A ต่างๆกัน ลักษณะความสัมพันธ์ของอินพุต และเอาท์พุตของวงจรบีบสัญญาณจะต่างกันไปดังแสดงในรูป

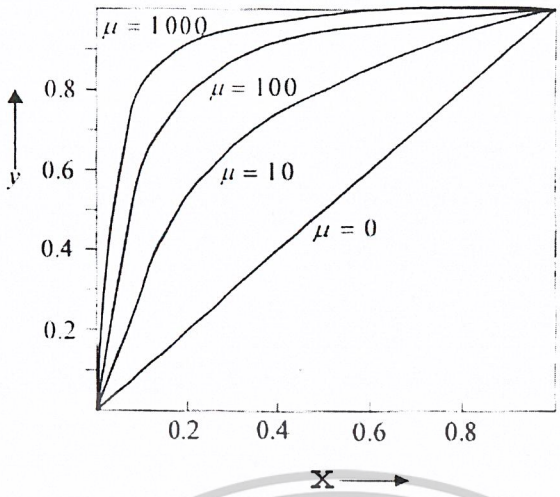


รูปที่ 2.16 ความสัมพันธ์ของอินพุต และเอาท์พุตของวงจรบีบสัญญาณตามกฎเอ

2. กฎมิว (μ -Law) นิยมใช้กันในอเมริกาเหนือ และญี่ปุ่น มีความสัมพันธ์ดังนี้ คือ

$$Y = \frac{\log(\mu x+1)}{\log(\mu+1)} \qquad (2.11)$$

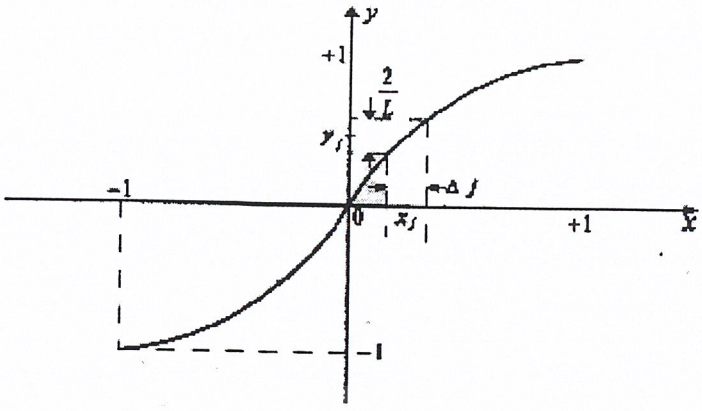
ในที่นี้ μ เป็นค่าพารามิเตอร์ที่ใช้สำหรับปรับลักษณะการบีบสัญญาณ เมื่อเลือกค่า μ ต่างๆกัน ลักษณะความสัมพันธ์ของอินพุต และเอาท์พุตของวงจรบีบสัญญาณจะต่างกันไปดังแสดงในรูปที่ 2.17



รูปที่ 2.17 ความสัมพันธ์ของอินพุต และเอาท์พุตของวงจรบีบสัญญาณตามกฎมิว

เป็นที่น่าสังเกตว่า คุณสมบัติตามกฎทั้งสองตามรูป จะมีคุณสมบัติบีบสัญญาณของอินพุตที่มีขนาดใหญ่มากกว่าการบีบสัญญาณอินพุตที่มีขนาดเล็ก เพราะในระบบการทำควอนไทซ์แบบไม่เชิงเส้นแบบนี้มีการใช้วงจรบีบสัญญาณก่อนทำการควอนไทซ์แบบเชิงเส้น ดังนั้นในทางเครื่องรับจึงมีความจำเป็นต้องใช้ วงจรถ่างสัญญาณ หรือ เอกซ์แพนเดอร์ ที่มีคุณสมบัติกลับกันกับคุณสมบัติของวงจรบีบสัญญาณ เพื่อที่จะทำให้สัญญาณที่ถูกบีบมานั้นถูกถ่างกลับคืนสู่สภาพเหมือนเดิม ระบบวงจรที่ใช้วงจรบีบและถ่างสัญญาณนี้มีชื่อเรียกว่าระบบที่ใช้ วงจรบีบ-ถ่างสัญญาณ หรือคอมแพนเดอร์

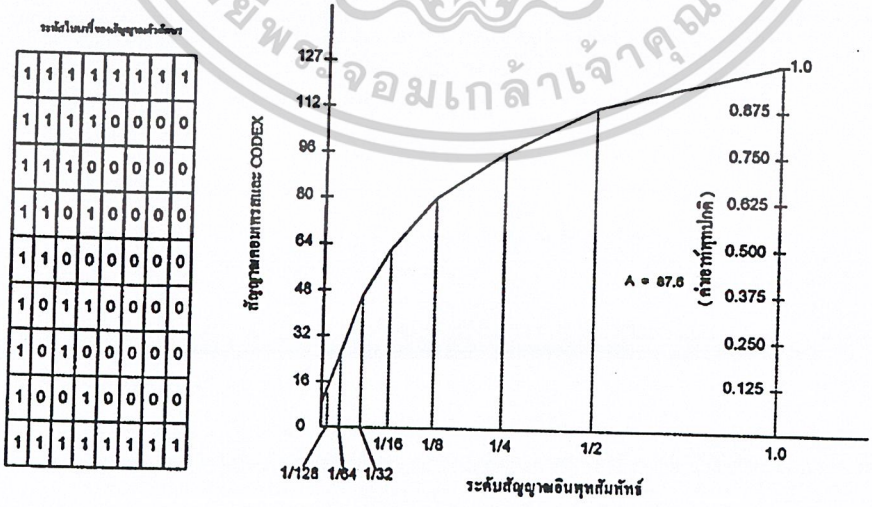
เพื่อวัตถุประสงค์ที่จะทำให้เกิดการปรับเปลี่ยนค่าเอสเอ็นอาร์ของสัญญาณที่ออกจากระบบให้มีค่าประมาณเท่ากัน โดยถึงแม้ว่าสัญญาณที่เข้ามานั้นจะมีค่าระดับกำลัง หรือขนดที่ต่างกันก็ตาม เพื่อให้ได้คุณสมบัติดังกล่าวนั้น ในทางปฏิบัตินิยมการใช้ วงจรบีบสัญญาณ มาทำการกดระดับของสัญญาณลงเมื่อสัญญาณมีระดับสูง การต่อวงจรบีบสัญญาณที่ข้างหน้าวงจรควอนไทเซอร์แบบสามัญ จะทำให้สามารถหลีกเลี่ยงการสร้างวงจรควอนไทเซอร์ที่มีระยะระหว่างระดับไม่สม่ำเสมอได้โดยตรง การใช้วงจรบีบสัญญาณมาต่อเช่นนี้จะให้ผลลัพธ์เหมือนกันกับการใช้วงจรควอนไทเซอร์ที่มีช่วงห่างของการจัดระดับสัญญาณที่ห่างไม่เท่ากัน แต่จะลดความยุ่งยากในการจัดแบ่งระดับสัญญาณที่มีช่วงห่างไม่เท่ากันลงได้ และเพื่อให้ได้สัญญาณเอาท์พุตที่ถูกต้องทางด้านเครื่องรับนั้นจำเป็นที่จะต้องใช้ วงจรถ่างสัญญาณ หรือวงจร เอกซ์แพนเดอร์ ซึ่งมีคุณสมบัติตรงข้ามกับวงจรคอมเพรสเซอร์ที่ใช้ทางเครื่องส่งมาต่อไว้หลังวงจรถอดรหัสที่อยู่ทางเครื่องรับด้วย เทคนิคการใช้คอมเพรสเซอร์ และเอกซ์แพนเดอร์รวมกัน นี้มีชื่อว่า เทคนิคการใช้ คอมแพนเดอร์



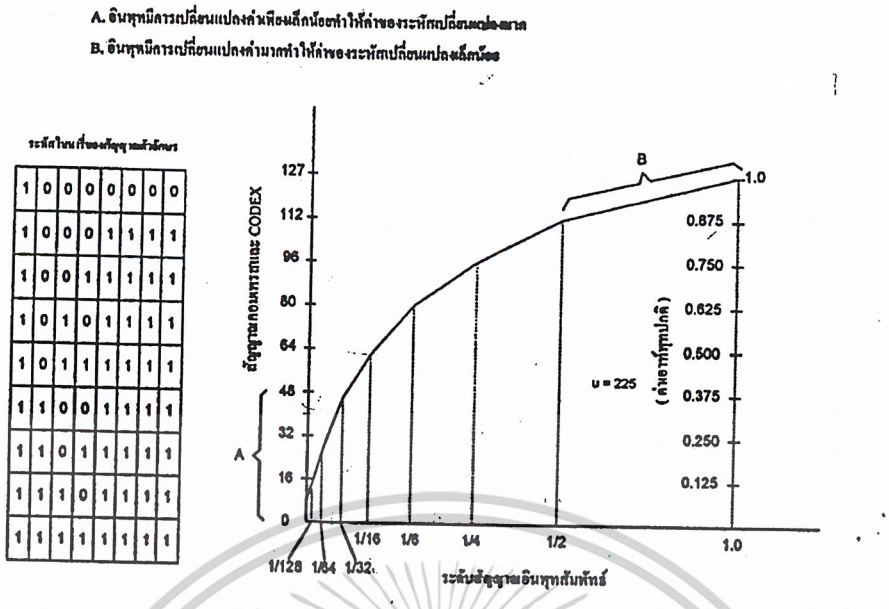
รูปที่ 2.18 ความสัมพันธ์ของอินพุต และเอาท์พุตของวงจรคอมเพรสเซอร์

2.5.2.5 การเข้ารหัส (Coding)

ในการจัดแบ่งเซกเมนต์ในการณ์ที่ใช้ u = 255 จะแบ่งเป็น 15 เซกเมนต์ สำหรับกรณีที่ใช้ A = 87.6 จะแบ่งเป็น 13 เซกเมนต์ โดยเซกเมนต์ที่ 1 จะเป็นเส้นตรงผ่านจุดเริ่มต้นไปทั้งทางบวกและทางลบสำหรับคุณลักษณะของ A-Law นั้นแสดงไว้ดังรูป ซึ่งแสดงคุณลักษณะเฉพาะด้านบวกเท่านั้น ดังนั้นจะต้องใช้สัญญาณไบนารี่ 4 บิต มาแสดงตำแหน่งของเซกเมนต์นั้น เมื่อทำการแบ่งเป็นเซกเมนต์แล้วการควอนไตซ์สัญญาณภายในแต่ละเซกเมนต์ก็จะสามารถใช้ควอนไตซ์เป็นเชิงเส้นได้ เนื่องจากระบบพีซีเอ็ม ใช้การเข้ารหัสด้วยสัญญาณไบนารี่ 8 บิต 4 บิตบนต้องใช้แสดงตำแหน่งของเซกเมนต์จึงเหลือเพียง 4 บิตล่างไว้แสดงระดับภายในแต่ละเซกเมนต์ นั่นคือในแต่ละเซกเมนต์ก็จะแบ่งออกเป็น 16 ขั้นได้ และเมื่อใช้ควอนไตซ์เซอร์แบบเป็นเชิงเส้นแต่ละขั้นมีค่าเท่ากัน การแสดงตำแหน่งของเซกเมนต์นั้นจะใช้บิตบนสุดแสดงว่าเซกเมนต์นั้นอยู่ในซีกบวกหรือซีกลบ และใช้ 3 บิตถัดไปแสดงลำดับของเซกเมนต์ในซีกนั้น



รูปที่ 2.19 การประมาณค่าเส้นโค้งการคอมแพนดิ้งแบบลอการิทึม A = 87.6 ด้วยส่วนของเส้นตรงเป็นช่วงๆ โดยแสดงเฉพาะค่าบวก



รูปที่ 2.20 การประมาณค่าเส้นโค้งการคอมแพนดิ้งแบบลอการิทึม $u = 255$ ด้วยส่วนของเส้นตรงเป็นช่วงๆ โดยแสดงเฉพาะค่าบวก

2.5.3 วิธีการเข้ารหัสของสัญญาณเสียงแบบ G.726

การเข้ารหัสเสียงแบบอแด็พทีฟดิฟเฟอร์เรนเชียลพัลส์โค้ดมอดูเลชั่น(Adaptive Differential Pulse Code Modulation: ADPCM) เป็นการเข้ารหัสแบบดิฟเฟอร์เรนเชียลพัลส์โค้ดมอดูเลชั่น (Differential pulse code modulation: DPCM)แต่

สัญญาณประเภทเสียงที่ถูกสุ่มตัวอย่าง (Sampling) แล้วจะมีความสัมพันธ์กันอย่างมากและจากข้อเท็จจริงนี้เองทำให้เราสามารถทำนายหรือคาดคะเน (Predict) แต่ละตัวอย่าง (Sample) โดยใช้ข้อมูลพื้นฐานจากตัวอย่างที่ผ่านมา แล้วเพียงแต่เข้าโค้ดที่แตกต่างระหว่างค่าที่ได้ทำนายไว้กับค่าตัวอย่าง แล้วส่งไปซึ่งเรียกวิธีการนี้ว่า "Differential Encoding" ซึ่งจะช่วยลดปริมาณของข้อมูลที่จำเป็นต้องใช้

อัลกอริทึมเบื้องต้น (The Basic Algorithm)

ในการทำนายอย่างง่ายเราจะทำนายว่า ตัวอย่างถัดไปจะเท่ากับตัวอย่างปัจจุบัน

$p_n = X_{n-1}$ เมื่อ p_n คือ ค่าที่ที่ทำนายตัวอย่างที่ n

X_{n-1} คือตัวอย่างที่ n-1

แม้ว่าในการเข้าโค้ดของผลต่างจะใช้จำนวนบิตน้อยกว่าเดิม (PCM) แต่เรากล่าวมาได้ว่ามันสามารถครอบคลุมได้ในเรื่องของการถอดรหัสที่ยอมรับได้ของ ลำดับสัญญาณจากค่าแตกต่างที่ถูกควอนไตซ์ (Quantize) แล้ว

ถ้าเราสนใจกับการบีบอัดข้อมูลแบบไม่มีการสูญเสียข้อมูล (Lossless Compression) เราจะพบว่าถ้าทำการเข้าโค้ดความแตกต่างระหว่างตัวอย่าง เราจะสามารถคลอบคลุมลำดับเดิมได้อย่างไม่สูญเสียตัวอย่างเช่น ลำดับต่อไปนี้

6.2 9.7 13.2 5.9 8 7.4 4.2 1.8

เราจะได้ลำดับของผลต่างระหว่างตัวอย่างดังนี้

6.2 3.5 3.5 -7.3 2.1 -0.6 -3.2 -2.4

จากนั้นลองมาดูว่าจะเกิดอะไรขึ้น ถ้าผลต่างถูกเข้าโค้ดแบบสูญเสียข้อมูล (Lossy) สมมุติว่าเรามีตัวควอนไตซ์เจ็ดระดับ ด้วยค่าเอาท์พุท -6,-4,-2, 0,2,4,6 จะได้ลำดับที่ถูกควอนไตซ์ เป็น

6 4 4 -6 2 0 -4 -2

เมื่อเราทำตามกระบวนการเดิมสำหรับสร้างสัญญาณคืน เราจะได้ลำดับดังนี้

6 10 14 8 10 10 6 4

ค่าผิดพลาดของสัญญาณที่สร้างคืนได้เมื่อเทียบกับสัญญาณเดิม มีค่าดังนี้

0.2 -0.3 -0.8 -2.1 -2 -2.6 -1.8 -2.2

สังเกตว่า ค่าผิดพลาดจะมีค่าน้อยสำหรับลำดับต้นๆ (0.2, 0.3) แต่จะเพิ่มขึ้นเมื่อลำดับเพิ่มมากขึ้น (2.6, 1.8, 2.2) ในการพิสูจน์นี้สามารถพิสูจน์ได้ด้วยสมการ พิจารณาลำดับตัวอย่าง $\{X_n\}$ ลำดับผลต่าง $\{\hat{d}_n\}$ ซึ่งเกิดจาก $X_n - X_{n-1}$ ซึ่งลำดับผลต่างนี้ถูกควอนไทซ์ได้ $\{\hat{d}_n\}$

$$\{\hat{d}_n\} = Q[d_n] = d_n + q_n \tag{2.12}$$

โดย q_n คือ ค่าความผิดพลาดที่เกิดจากการควอนไตซ์ (Quantization Error) และที่เครื่องรับ $\{\hat{X}_n\}$ ถูกสะสมโดยการบวกค่า $\hat{d}$ กับค่าสะสมก่อนหน้า $\hat{X}_{n-1}$

$$\hat{X}_n = \hat{X}_{n-1} + \hat{d}_n \tag{2.13}$$

สมมุติว่าสัญญาณตัวอย่างเริ่มต้นด้วย X_0 จะได้

$$d_1 = X_1 - X_0 \tag{2.14}$$

$$\hat{d}_1 = Q[d_1] = d_1 + q_1 \tag{2.15}$$

$$X_1 = X_0 + \hat{d}_1 = X_0 + d_1 + q_1 = X_1 + q_1 \tag{2.16}$$

$$d_2 = X_2 - X_1 \tag{2.17}$$

$$\hat{d}_2 = Q[d_2] = d_2 + q_2 \tag{2.18}$$

$$\hat{X}_2 = \hat{X}_1 + \hat{d}_2 = X_1 + q_1 + d_2 + q_2 \quad (2.19)$$

$$= X_2 + q_1 + q_2 \quad (2.20)$$

เมื่อกระบวนการนี้มีต่อเรื่อยๆ ไปจนถึงลำดับที่ n จะได้ว่า

$$\hat{X}_n = X_n + \sum_{k=1}^{n} q_k$$

ซึ่ง $\sum_{k=1}^{n} q_k$ คือค่าของ Error (2.21)

จะพบว่าเมื่อลำดับมากขึ้นค่าความผิดพลาดจะมากขึ้น ก็เนื่องจาก $\sum_{k=1}^{n} q_k$

จะมีค่าสะสมมากขึ้นเรื่อยๆ ปัญหานี้จะแก้ไขโดยการแทน $d_n = X_n - \hat{X}_{n-1}$ ด้วย $\hat{X}_0 = X_0$ จะได้

$$d_1 = X_1 - X_0 \quad (2.22)$$

$$\hat{d}_1 = Q[d_1] = d_1 + q_1 \quad (2.23)$$

$$\hat{X}_1 = X_0 + \hat{d} = X_0 + d_1 + q_1 = X_1 + q_1 \quad (2.24)$$

$$d_2 = X_2 - \hat{X}_1 \quad (2.25)$$

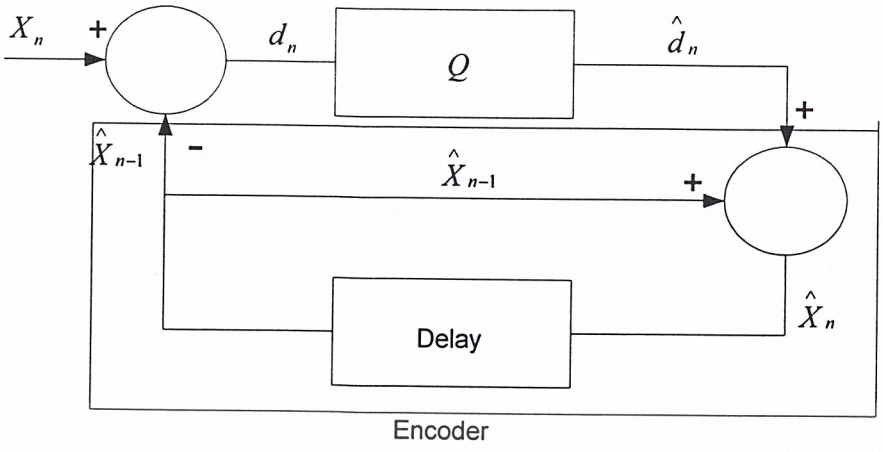
$$\hat{d}_2 = Q[d_2] = d_2 + q_2 \quad (2.26)$$

$$\hat{X}_2 = \hat{X}_1 + \hat{d}_2 = \hat{X}_1 + d_2 + q_2 \quad (2.27)$$

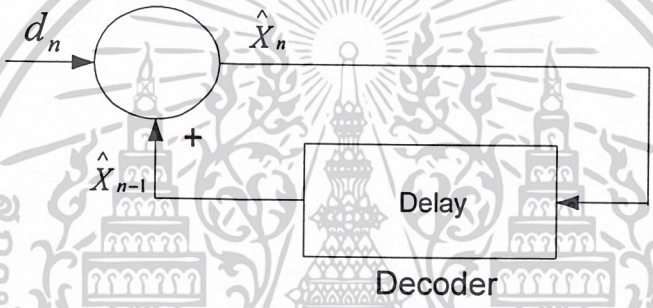
$$= X_2 + q_2 \quad (2.28)$$

เราจะพบว่า $\hat{X}_n = X_n + q_n$ (2.29)

ซึ่งค่าผิดพลาดจากการควอนไตซ์จะไม่เพิ่มขึ้นเมื่อลำดับเพิ่มขึ้น ซึ่งเทคนิคที่กล่าวมาสามารถเขียนเป็นระบบการเข้ารหัส และถอดรหัสได้ดังนี้



รูปที่ 2.21 แสดงระบบการเข้ารหัสแบบผลต่าง



รูปที่ 2.22 แสดงระบบการถอดรหัสแบบผลต่าง

ในการใช้งานพีซีเอ็ม มีอัตราการส่งมาตรฐานคือ 64 Kbit/s ซึ่งต้องการช่องสัญญาณในการส่งที่มีแบนด์วิทธ์กว้างในการนำไปใช้งาน เช่นงานเกี่ยวกับความปลอดภัยของสัญญาณเสียงในการส่งในสายส่งบนช่องสัญญาณวิทยุซึ่งมีความจุต่ำ การประยุกต์ใช้งานแบบนี้เป็นที่ต้องการสำหรับการเข้ารหัสสัญญาณเสียงที่มีอัตราการส่งต่ำ จากการศึกษาได้พบว่าการเข้ารหัสสัญญาณแบบดีพีซีเอ็ม เป็นวิธีการบีบอัดสัญญาณวิธีหนึ่งที่ถูกนำมาใช้ในการลดจำนวนบิทของการเข้ารหัสสัญญาณ แต่ความต้องการในการลดจำนวนบิตในการเข้ารหัสนั้นยังคงมีอยู่เรื่อยๆ จึงได้เกิดระบบการเข้ารหัสสัญญาณเสียงอีกแบบหนึ่งขึ้นมาคือ การเข้ารหัสเสียงแบบอแด็พทีฟดิฟเฟอร์เรนเชียลพัลส์โค้ดมอดูเลชั่นดูเลชั่น

หลักการของการเข้ารหัสเสียงแบบอแด็พทีฟดิฟเฟอร์เรนเชียลพัลส์โค้ดมอดูเลชั่น หรือ เอดีพีซีเอ็ม จะคล้ายคลึงกับระบบดีพีซีเอ็ม แต่มีการเพิ่มคุณสมบัติในการปรับขั้นของการควอนไตซ์ และวิธีการเปรียบเทียบสัญญาณเข้าไปในระบบดีพีซีเอ็ม ซึ่งทำให้ลดจำนวนบิทในการเข้ารหัสได้มากขึ้น มาตรฐานของระบบเอดีพีซีเอ็ม จะอธิบายถึงการคอมเพรซซิ่งและเอ็กแพนดิ้ง ค่าความยาวของแซมเปิ้ลจาก 8 บิต จะลดลงเหลือ 3, 4 หรือ 5 บิท โดยมีอัตราการบีบอัดเป็น 2.67, 2 และ 1.6 ตามลำดับ และมีอัตราการส่งสัญญาณเท่ากับ 24,32 และ 40 Kbit/s ตามลำดับ เมื่อมีอัตราการแซมปลิ้งเท่ากับ 8 KHz โดยใช้ควอนไตซ์เซอร์เป็นแบบอแด็พทีฟควอนไตซ์เซอร์ (adaptive quantizer) ซึ่งค่าระดับการควอนไตซ์จะเปลี่ยนตามค่าระดับของสัญญาณแบบนอนยูนิ

ฟอร์มแล้ว ค่าสเต็ปไซต์ของระดับการควอนไตซ์ยังปรับตามค่าระดับสัญญาณผลต่างที่เข้ามา คือ เมื่อระดับของสัญญาณผลต่างมีค่าสูง ค่าสเต็ปไซต์ก็จะปรับเป็นค่าสูง แต่ถ้าระดับของผลต่างมีค่าต่ำค่าสเต็ปไซต์ก็จะปรับเป็นค่าต่ำ เพื่อให้จำนวนบิทที่ใช้ในการเข้ารหัสสัญญาณสามารถครอบคลุมค่าระดับของผลต่างของสัญญาณได้ทั้งหมด ส่วนการเปรียบเทียบสัญญาณจะใช้อแด็พทีฟพรีดิกเตอร์ (adaptive predictor) เป็นตัวหาค่าประมาณของสัญญาณอินพุทเพื่อนำมาใช้เปรียบเทียบกับสัญญาณอินพุทที่เข้ามาตัวถัดไป

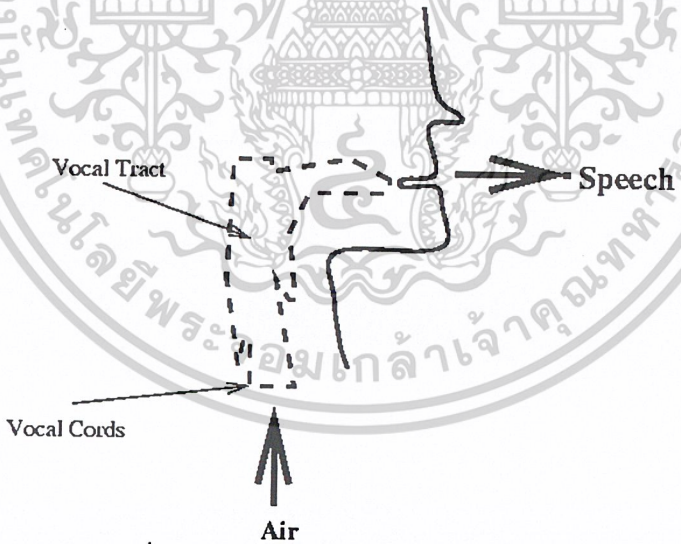
2.5.4 วิธีการเข้ารหัสของสัญญาณเสียงแบบ G.729

G.729 จัดอยู่ใน Speech coders ในประเภท CELP (Code Excited Linear Prediction) ซึ่ง Model ชนิดนี้มีพื้นฐานมาจากระบบกำเนิดเสียงของมนุษย์ (LPC Modeling)

G.729 ซึ่งเป็นการควอนไตซ์แบบพารามิเตอร์ สามารถบีบอัดข้อมูลได้มากถึง 8 เท่า อัลกอริธึมที่ใช้ในมาตรฐานนี้คือ CS-ACELP ซึ่งจะนำเสียงที่ถูกสุ่มด้วยความถี่ 8000 Hz มาทีละ 10 ms (เท่ากับสัญญาณ80จุด) แล้วทำการแปลงเป็นพารามิเตอร์ขนาด 80 บิต พารามิเตอร์ที่คำนวณได้นี้ คือ พารามิเตอร์ของส่วนประกอบที่ใช้ในการสร้างเสียง โดยมองว่าเสียงสร้างขึ้นจากสัญญาณกระตุ้น (ลมในช่องท้อง คือลักษณะของเสียงกระตุ้นในรูป) ที่ส่งผ่านวงจรกรอง (ช่องคอ,ทางเดินอากาศ,ช่องปากและลิ้น คือ ส่วน filter ทั้งหลายในรูป)

2.5.4.1 การสร้างแบบจำลอง LPC (LPC Modeling)

2.5.4.1.1 รูปแบบทางกายภาพ

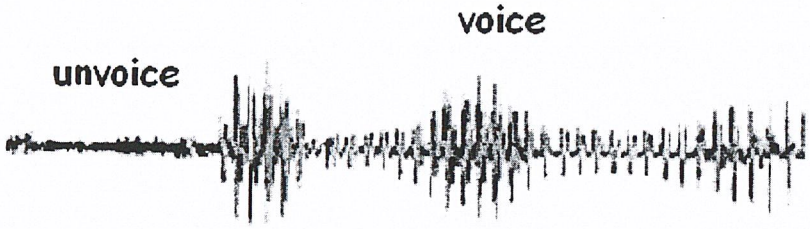


รูปที่ 2.23 แสดงกลไกการเกิดเสียงของมนุษย์

เวลาที่คุณพูด:

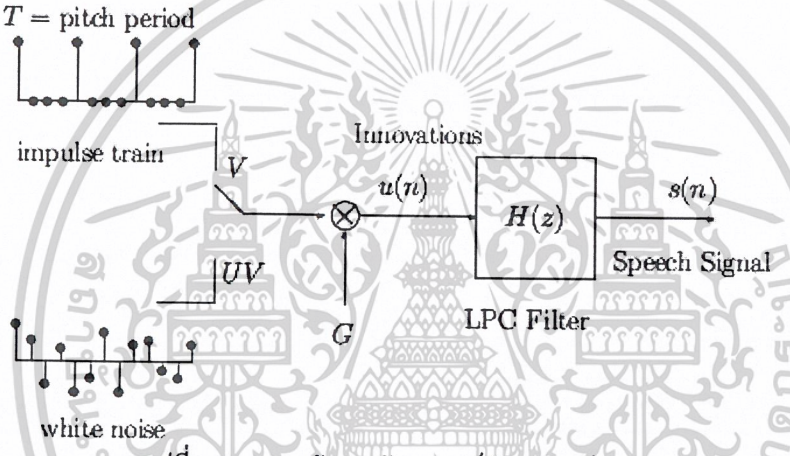
1. อากาศจะถูกผลักออกมาจากปอดผ่านหลอดลมและออกมาทางปากกลายมาเป็นเสียงพูด.
2. สำหรับเสียงที่ถูกพูดออกมา จะมีจังหวะการเปิดและปิด โดยคาบของการสั่นเรียกว่า Pitch ของเสียง ในผู้หญิงและเด็กๆอายุน้อยมักจะมี Pitch สูงกว่าของเพศชาย
3. สำหรับเสียงที่เกิดจากการเสียดแทรก (unvoiced) เส้นเสียงของคุณไม่สั่นแต่ยังคงเปิดให้อากาศผ่าน
4. รูปร่างของหลอดลมสามารถกำหนดเสียง เมื่อรูปร่างหลอดลมเปลี่ยนรูปร่างทำให้เกิดเสียงแตกต่าง

5. รูปร่างของหลอดลมเปลี่ยนอย่างค่อนข้างช้าๆ (โดยทั่วไปอยู่ในระดับ 10 มิลลิวินาที ถึง100 มิลลิวินาที)
6. จำนวนรวมของอากาศที่มาจากปอดเป็นตัวกำหนดความดังของเสียง



รูปที่ 2.24 แสดงลักษณ์ของเสียงพูด

2.5.4.1.2 รูปแบบทางคณิตศาสตร์



รูปที่ 2.25 แสดงผังการสังเคราะห์สัญญาณเสียง

โมเดลข้างต้นถูกเรียกว่า "LPC Model" สัญญาณเสียงที่สังเคราะห์ขึ้นได้จากอินพุท 2 ส่วนนำมาร่วมกันผ่านวงจรกรองซึ่ง อินพุททั้ง 2 นี้แทนคุณลักษณะของเสียงโดย Voiced sound มีลักษณะเป็นคาบในขณะที่ Unvoiced sound นั้นมีลักษณะของสัญญาณรบกวนขาว

2.5.4.1.3 เปรียบเทียบความสัมพันธ์ระหว่างรูปแบบทางกายภาพและรูปแบบทางคณิตศาสตร์

$$\text{Vocal Tract} \Longleftrightarrow H(z) \text{ (LPC Filter)}$$

$$\text{Air} \Longleftrightarrow u(n) \text{ (Innovations)}$$

$$\text{Vocal Cord Vibration} \Longleftrightarrow V \text{ (voiced)}$$

$$\text{Vocal Cord Vibration Period} \Longleftrightarrow T \text{ (pitch period)}$$

$$\text{Fricatives and Plosives} \Longleftrightarrow UV \text{ (unvoiced)}$$

$$\text{Air Volume} \Longleftrightarrow G \text{ (gain)}$$

หลังจากทำการบีบอัดข้อมูลแล้วก็ก็จะส่งออกไปทางช่องสื่อสาร เมื่อข้อมูลถูกส่งไปถึงปลายทางก็จะทำการขยายหรือแปลงข้อมูลจากค่าพารามิเตอร์เหล่านี้กลับมาเป็นเสียงพูดใหม่

2.5.5 คุณภาพกับการบีบอัด

การบีบอัดย่อมทำให้คุณภาพตกลง แต่คุณภาพยังอยู่ที่ระดับความพอใจ และยิ่งบีบอัดได้มากเท่าไร โดยที่ยังอยู่ในระดับความพอใจได้ก็นับว่าเป็นเรื่องดี ในการสื่อสารสัญญาณเสียงในย่านความถี่นี้ โดยจะใช้การวัดคุณภาพความพอใจใช้มาตรฐานที่ชื่อว่า MOS-Mean Opinion Score หรือให้เป็นคะแนน โดยให้คะแนนจาก 0 ถึง 5 ถ้า 0 หมายถึง คุณภาพแย่ และ 5 คือคุณภาพดี

เมื่อนำเทคนิคต่างๆมาหาค่า MOS ผลปรากฏว่าพีซีเอ็ม ได้คะแนน MOS = 4.4 ขณะที่ G.726 เอดีพีซีเอ็ม ได้คะแนน 4.2 และ G.728 ได้คะแนน 4.2 เท่ากัน G.729 ก็ยังได้คะแนน 4.2 แสดงให้เห็นว่าเทคนิคการบีบอัดที่ดีก็ยังทำให้ระดับคุณภาพอยู่ในเกณฑ์ดีได้แต่ต้องเสียเวลาในการประมวลผล

ถึงแม้ว่าจะได้คุณภาพที่มีความพอใจสูง แต่ก็ต้องเสียเวลาหน่วงหรือที่เรียกว่าเวลาหน่วง (Delay Time) ทั้งนี้เพราะการบีบอัดข้อมูลต้องใช้หลักการคำนวณทางคณิตศาสตร์ช่วย ทำให้ต้องเสียเวลาการเข้ารหัสและการถอดรหัสทำให้มีเวลาหน่วงสูงกว่า

เวลาหน่วงนี้เป็นผลให้สัญญาณเสียงที่รับส่งจะล่าช้าไป และอาจสร้างปัญหาในเรื่องการสะท้อนเสียงกลับ ซึ่งต้องมีวิธีในการแก้ปัญหานี้

เทคนิคการบีบอัดเสียง	คะแนน MOS	การหน่วง
PCM (G.711)	4.4	0.75
32K ADPCM (G.726)	4.2	1
16K LD-CELP (G.728)	4.2	3-5
8K CS-ACELP (G.729)	4.2	10
8K CS-ACELP (G.729a)	4.2	10
6.3K MPMLQ (G.723.1)	3.98	30
5.3K ACELP(G.723.1)	3.5	30

ตารางที่ 2.8 แสดงการเปรียบเทียบเทคนิควิธีการต่างๆ กับคุณภาพและเวลาหน่วง

จากตารางที่ 2.8 จะเห็นได้ชัดว่าเวลาหน่วงจะมากขึ้นเมื่อต้องการบีบอัดสัญญาณให้ได้มากขึ้น การหน่วงเวลาเป็นปัญหาใหญ่ปัญหาหนึ่งของระบบโทรคมนาคมในปัจจุบัน อุปกรณ์ในการสวิตช์แพ็คเกจเสียงต้องเสียเวลาเพิ่มมากขึ้น และเมื่อรวมๆกันแล้วจะมีมากขึ้น ซึ่งอาจถึงระดับที่เป็นปัญหาได้แต่หากดูจากตารางการเข้ารหัสแล้วพบว่าเสียเวลาเพิ่มอีกเพียงเล็กน้อย ดังนั้นสภาพการเสียเวลาหน่วงในการรับส่งและประมวลจึงน่าจะอยู่ในระดับการยอมรับได้ อีกทั้งการประมวลดิจิตอลมีแนวโน้มการพัฒนาให้ดียิ่งขึ้นต่อไปอีก ปัญหาในเรื่องเวลาหน่วงจึงไม่ใช่ปัญหาใหญ่ทีเดียว

เมื่อสามารถเข้ารหัสสัญญาณเสียงที่มีประสิทธิภาพได้ สัญญาณเสียงจึงมีลักษณะเป็นข้อมูลดิจิตอลที่มีขนาดไม่มากนัก เพียงประมาณ 8 กิโลบิตต่อวินาที ซึ่งนั่นหมายถึงสามารถส่งไปยังเครือข่ายคอมพิวเตอร์หรือเครือข่ายสื่อสารอื่นๆได้ รูปแบบการผสมสัญญาณเสียงไปในเครือข่ายคอมพิวเตอร์และเครือข่ายอื่นที่ดีและเหมาะสมจึงต้องใช้การรับส่งเป็นแพ็คเกจ

บทที่ 3
ออกแบบและการสร้าง

3.1 โครงสร้างโดยรวมของโปรแกรม

ในการออกแบบโปรแกรมสนทนาด้วยเสียง เพื่อให้ผู้ใช้งานคอมพิวเตอร์ที่มีการติดตั้งการ์ดเสียงและต่อคอมพิวเตอร์เข้ากับระบบเครือข่าย สามารถที่จะสื่อสารด้วยเสียงโดยผ่านทางระบบเน็ตเวิร์คที่มีอยู่ ซึ่งจะสามารถใช้งานระบบเน็ตเวิร์คที่มีอยู่ให้มีประสิทธิภาพนั้น ในการสื่อสารผ่านระบบเน็ตเวิร์คจำเป็นต้องมีการใช้โปรโตคอลซึ่งเปรียบเทียบเสมือนกับภาษาที่ใช้ในการพูดคุยระหว่างเครื่องคอมพิวเตอร์ ซึ่งต่ออยู่ในระบบเน็ตเวิร์คโดยมีโปรโตคอลที่ใช้อยู่หลายแบบ เช่น IPX/SPX,TCP/IP,UDP เป็นต้น และเพื่อให้โปรแกรมสนทนาด้วยเสียง สามารถที่จะใช้ได้อย่างกว้างขวาง เราจึงทำการออกแบบโปรแกรมโดยเลือกใช้โปรโตคอลทีซีพี/ไอพีและยูดีพี

การออกแบบตัวโปรแกรมจะแบ่งออกเป็น 2 ส่วนสำคัญ คือ

1. ส่วนของการรับและส่งข้อมูลผ่านระบบเน็ตเวิร์ค
2. ส่วนของการจัดการเกี่ยวกับการบีบอัดของเสียง

ซึ่งในการออกแบบโปรแกรมเราจะใช้หลักการของ Object ซึ่งหลักการของ Object จะทำให้เราสามารถที่จะแยกโปรแกรมแต่ละส่วนให้เป็นอิสระจากกัน โดยที่แต่ละ Object ก็ยังคงมีความสัมพันธ์กัน

หลังจากทำการออกแบบหลักการทำงานของโปรแกรมแล้ว จะมาดูความสามารถของโปรแกรมว่าทำอะไรได้บ้าง ซึ่งได้วางความสามารถไว้ดังนี้

1.สามารถทำการส่งเสียงพูดจากคอมพิวเตอร์เครื่องหนึ่งที่เรียกว่าผู้ส่งไปยังคอมพิวเตอร์อีกเครื่องหนึ่งที่เรียกว่าผู้รับได้

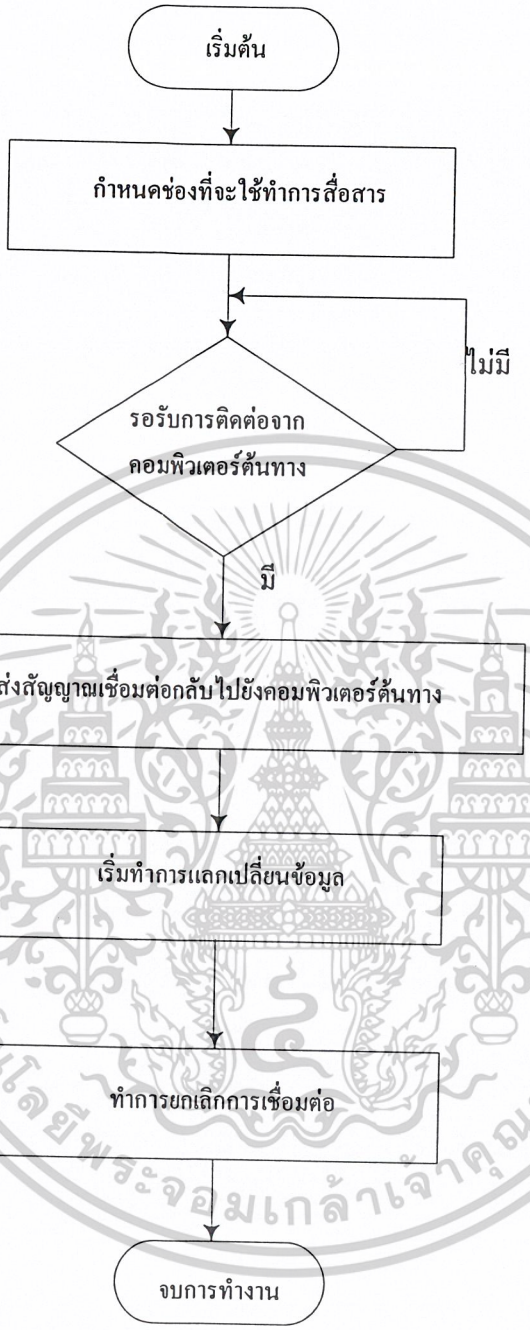
2. สามารถรับเสียงที่ส่งมาจากเครื่องคอมพิวเตอร์ที่เรียกว่าผู้ส่งแสดงออกลำโพงของเครื่องตนได้

3. สามารถที่จะยกเลิกการรับและส่งเสียงได้

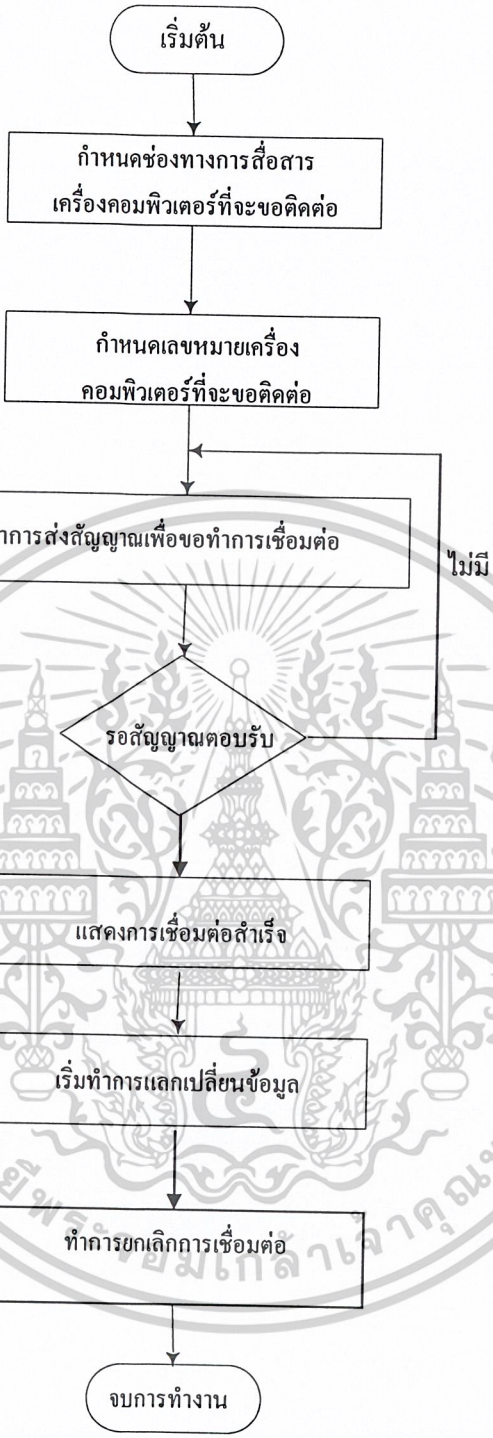
3.2 การออกแบบโปรแกรมในส่วนของการรับและส่งข้อมูลผ่านระบบเน็ตเวิร์ค (Network Connection)

ในการจัดการส่วนของการรับและส่งข้อมูลผ่านระบบเน็ตเวิร์คเราได้เลือกใช้โปรโตคอลทีซีพี/ไอพีมาใช้เป็นโปรโตคอลที่ใช้ในการสื่อสาร โดยในส่วนทำการเชื่อมต่อจะใช้โปรโตคอลทีซีพี แต่ในส่วนการสนทนาจะใช้โปรโตคอลยูดีพี ดังนั้นจะกล่าวถึงโปรโตคอลทีซีพีก่อน

เนื่องมาจากการติดต่อในโปรโตคอลทีซีพี/ไอพีจะเป็นแบบ Connection-Oriented คือมีการสร้างการติดต่อกันเป็นเซสชั่นทั้ง 2 ด้าน แล้วจึงจะสามารถส่งข้อมูลไปได้พร้อมกันซึ่งด้วยกลไกของโปรโตคอลทีซีพีจะมีการเพิ่มส่วนในการตรวจสอบข้อมูลไม่ให้ผิดพลาดไปจากเดิม โดยการส่งสัญญาณสอบทานข้อมูล (Acknowledgement) โดยจะทำการส่งข้อมูลให้ใหม่อีกครั้ง ถ้าปลายทางไม่ได้รับหรือเกิดความผิดพลาดขึ้น ซึ่งทำให้โปรโตคอลทีซีพี/ไอพี มีความน่าเชื่อถือในการส่งผ่านข้อมูลมาก และในส่วนของการออกแบบนี้เราจะทำการแบ่งส่วนการทำงานออกเป็นส่วนของคอมพิวเตอร์ 2 เครื่อง โดยจะแสดงในส่วนของคอมพิวเตอร์ปลายทางก่อน และแสดงในส่วนของคอมพิวเตอร์ต้นทาง

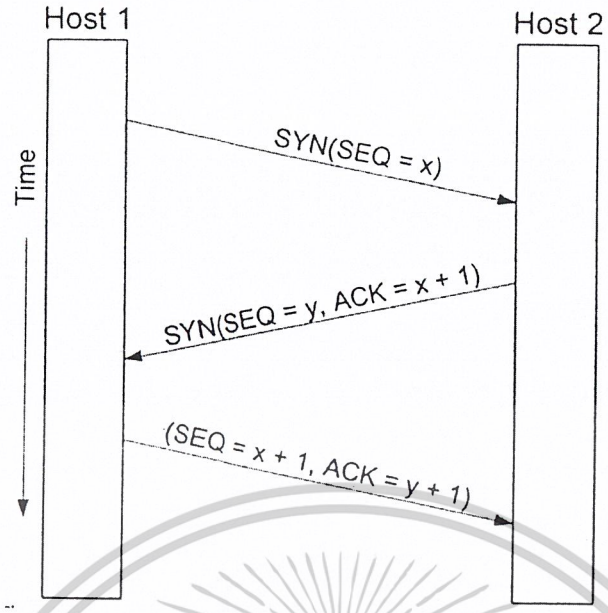


รูปที่ 3.1 แสดงผังการทำงานของโปรแกรมในส่วนของคอมพิวเตอร์ปลายทาง



รูปที่ 3.2 แสดงผังการทำงานของโปรแกรมในส่วนของคอมพิวเตอร์ต้นทาง

จากผังการทำงานของโปรแกรมในคอมพิวเตอร์ทั้ง 2 เครื่องนั้น เมื่อทำการดูจากผังการทำงานทั้งสองอาจจะยังเห็นภาพของระบบซึ่งเป็นการเชื่อมต่อผ่านระบบทีซีพีไม่ชัดเจนดีนัก จึงได้แสดงสภาวะการเชื่อมต่อผ่านทีซีพี ตามรูปที่ 3.3 เป็นดังนี้



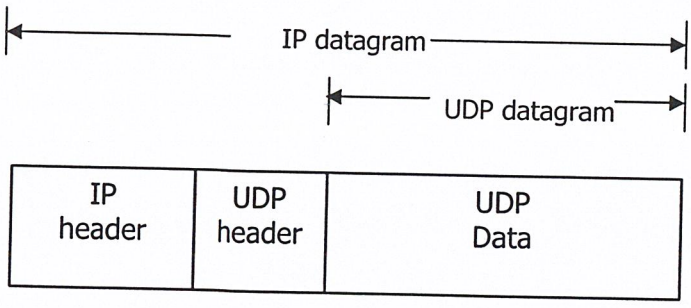
รูปที่ 3.3 แสดงลำดับขั้นตอนการส่งทีซีพีในการสร้างการเชื่อมต่อระหว่างคอมพิวเตอร์ 2 เครื่อง

การสร้างการเชื่อมต่อในระบบทีซีพีใช้กระบวนการจับมือร่วมสามขั้นตอน (three-way handshake) โดยทางคอมพิวเตอร์ปลายทาง จะเรียกใช้บริการ Listen เพื่อรอคอยสัญญาณ Connection จากผู้ส่งซึ่งโดยคอมพิวเตอร์ต้นทาง จะทำการส่งข้อมูลมาให้ โดยข้อมูลหลักที่ส่งมาด้วยคือ หมายเลขไอพีและพอร์ตที่ใช้ ขนาดสูงสุดของเซกเมนต์ และจบลงด้วยการเรียกใช้บริการ Accept เพื่อยืนยันการเชื่อมต่อกลับไป

เมื่อเซกเมนต์ Connect (SYN = "1" และ ACK = "0") เดินทางมาถึง เอ็นติตี้ ทีซีพีที่คอมพิวเตอร์ปลายทาง จะค้นหาโพรเซสตามหมายเลขพอร์ตที่กำหนดในเขตข้อมูล พอร์ตปลายทาง (Destination port) ซึ่งถ้าหาไม่พบก็จะตอบปฏิเสธด้วยเซกเมนต์ที่มี RST = "1" กลับไปยังคอมพิวเตอร์ต้นทาง

เซกเมนต์ Connect ของคอมพิวเตอร์ต้นทาง จะถูกส่งต่อไปยังโพรเซสตามพอร์ตที่ระบุ ซึ่งอาจจะตอบรับหรือตอบปฏิเสธก็ได้ ถ้าโพรเซสนั้นต้องการสื่อสารด้วยก็จะส่งเซกเมนต์ตอบรับกลับไป

เมื่อการเชื่อมต่อทำสำเร็จแล้ว เราจะใช้โปรโตคอลยูดีพีเพื่อทำการสนทนา ซึ่งยูดีพีเป็นโปรโตคอลพื้นฐานที่อาศัยไอพีเป็นพาหะในการส่งข้อมูล โดยตัวยูดีพีนั้นจัดอยู่ในชั้นทรานสปอร์ต เลเยอร์ ลักษณะของโปรโตคอลจะจัดการครั้งละ 1 ชุดของข้อมูลที่เรียกว่า ยูดีพี ดาต้าแกรม โดยข้อมูลแต่ละดาต้าแกรมจะไม่มีความสัมพันธ์กัน (ในความรับรู้ของ ยูดีพี) เพราะความสัมพันธ์ระหว่างดาต้าแกรมจะถูกจัดการโดยโปรโตคอลอื่นในแอพพลิเคชั่นเลเยอร์แทน



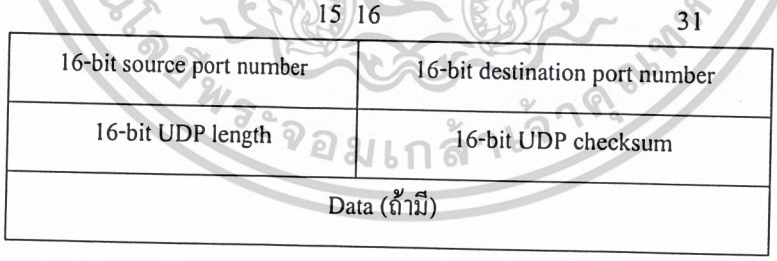
รูปที่ 3.4 ยูดีพี เอนเคปซูเลท

ยูดีพี ดาต้าแกรมจะถูกเอนเคปซูเลทลงในไอพีดาต้าแกรมดังแสดงในรูปที่ 3.4 โดยเมื่อเอนเคปซูเลทแล้ว 20 ไบต์แรกจะเป็นของไอพีเฮดเดอร์ (IP Header) และในไบต์ที่ 9 ของไอพีเฮดเดอร์จะต้องมีค่า 17 ด้วย คุณสมบัติสำคัญของยูดีพีคือจัดรูปแบบข้อมูลอย่างง่ายให้อยู่ในรูปของยูดีพี ดาต้าแกรม และรับส่งข้อมูลชุดนี้ให้ถึงปลายทางเท่านั้น ไม่มีกลไกในการตรวจสอบยืนยันการรับส่งข้อมูลในตัวของยูดีพีเอง ดังนั้น แอพพลิเคชั่นที่ใช้ยูดีพีจะต้องระลึกเสมอว่ายูดีพีเป็นโปรโตคอลที่ไม่มีเสถียรภาพและไม่รับประกันการรับส่งข้อมูล แต่อย่างไรก็ตาม แอพพลิเคชั่นบางประเภทที่ไม่ต้องการความถูกต้องข้อมูลมากนักอาจจะเลือกใช้โปรโตคอลนี้เพื่อประสิทธิภาพในการสื่อสารข้อมูลเร็วกว่าได้

ยูดีพีเฮดเดอร์ จากรูปที่ 3.5 จะเห็นว่ายูดีพีเฮดเดอร์ได้มีการกำหนดฟิลด์ไว้อย่างง่ายๆ มีขนาดทั้งหมดของเฮดเดอร์เพียง 8 ไบต์เท่านั้น โดยแต่ละฟิลด์มีความหมายดังนี้

ไบต์ 0-1 Source Port Number หมายเลขพอร์ตต้นทาง ที่ส่งข้อมูลดาต้าแกรมนี้

ไบต์ 2-3 Destination Port Number หมายเลขพอร์ตปลายทาง ที่จะเป็นผู้รับข้อมูลดาต้าแกรมไปใช้งาน

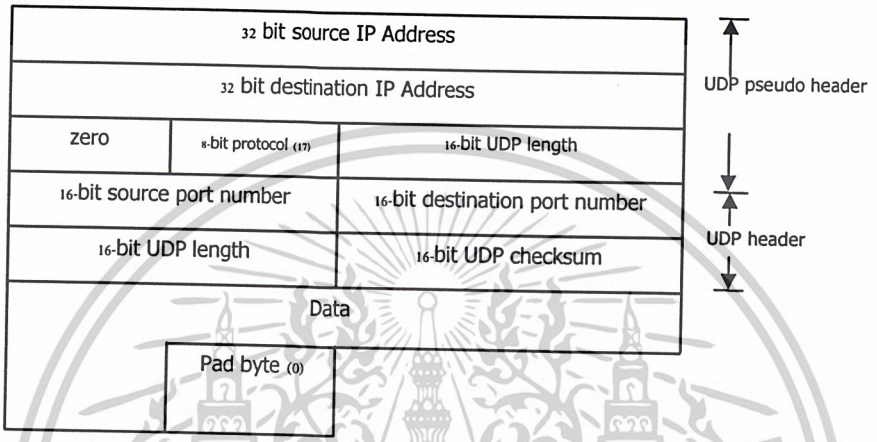


รูปที่ 3.5 ยูดีพีเฮดเดอร์

ไบต์ 4-5 UDP Length เป็นฟิลด์ที่ระบุความยาวของยูดีพี ดาต้าแกรม คือ ยูดีพีเฮดเดอร์รวมกับยูดีพีดาต้า ขนาดต่ำสุดของฟิลด์คือ 8 ไบต์ หมายถึง ดาต้าแกรมนี้มีเฉพาะ ยูดีพีเฮดเดอร์ ซึ่งเท่ากับ 8 ไบต์และไม่มียูดีพีดาต้า

ไบต์ 6-7 UDP Checksum ทำหน้าที่ตรวจสอบความถูกต้องของยูดีพีดาต้าแกรมทั้งหมด ถึงแม้ว่ายูดีพีจะถูกเอนเคปซูเลทอยู่ในไอพีดาต้าแกรมและมีไอพีเช็คซัม (IP Checksum) คุมอยู่แล้วก็ตาม แต่ไอพีเช็คซัมนั้น

จะทำหน้าที่ตรวจสอบความถูกต้องของไอพีเฮดเดอร์เท่านั้น มิได้ครอบคลุมทั้งไอพี ดาต้าแกรมแต่อย่างใด นั่นหมายถึงหากมีความผิดพลาดในส่วนข้อมูลของไอพี ดาต้าแกรม ไอพีก็จะไม่สามารถทราบได้ ซึ่งต่างจากยูดีพีเช็คซัม ซึ่งทำหน้าที่ตรวจสอบความถูกต้องทั้งยูดีพีดาต้าแกรมและเป็นกลไกของยูดีพีเองแยกต่างหากจากไอพี ดังนั้นการตรวจสอบความถูกต้องด้วยเช็คซัม เมื่อใช้ยูดีพีก็จะมี 2 ขั้นตอน คือการตรวจสอบความถูกต้องในระดับ ไอพีแอดเดรส โดยใช้เช็คซัมของไอพี หลังจากนั้นจึงค่อยทำการตรวจสอบความถูกต้องของยูดีพีโดยใช้ยูดีพีเช็คซัมอีกทีหนึ่ง



รูปที่ 3.6 ฟิลด์ที่ใช้ในการคำนวณหายูดีพีเช็คซัม

ยูดีพี เช็คซัม เป็นกลไกในการหาค่าเช็คซัมเพื่อใช้ตรวจสอบความถูกต้องของยูดีพี จะคล้ายกับการหาค่าเช็คซัมของไอพีคือค่าเช็คซัมที่ได้จะเป็นค่าผลรวมของข้อมูลขนาด 16 บิตทั้งหมด และแปลงเป็นวันคอมพลิเมนท์ (one's compliment) แต่จะมีจุดแตกต่างจากการหาค่าเช็คซัมของไอพีอยู่ 2 ประการคือ

1. ขนาดของยูดีพี ดาต้าแกรมจะไม่คงที่เนื่องจากขนาดของส่วนที่เป็นดาต้าอาจเปลี่ยนแปลงได้ตามขนาดของข้อมูลจริง

2. ถึงแม้ว่าจริงๆแล้วยูดีพีเฮดเดอร์จะมีขนาด 8 ไบต์ แต่ในการหาค่าเช็คซัมนั้นจะนำบางค่าในไอพีเฮดเดอร์มารวมเป็นส่วนหนึ่งของยูดีพีเฮดเดอร์ (เรียกว่า UDP Pseudo Header) จากนั้นจึงหาค่าเช็คซัมทั้งหมดอีกทีหนึ่ง

ฟิลด์ที่ยูดีพีนำมาจากไอพีเฮดเดอร์ดังแสดงในรูปที่ 3.6 ได้แก่ ไอพีแอดเดรสต้นทาง ไอพีแอดเดรสปลายทาง ซีโร่ (Zero) โปรโตคอล ความยาวยูดีพี ทั้งนี้เพื่อให้ค่ายูดีพีเช็คซัมได้ทำการตรวจสอบซ้ำในส่วนที่สำคัญสำหรับยูดีพีด้วยว่าการรับส่งถูกต้องทั้งต้นทางและปลายทาง

กลไกในการตรวจสอบโดยเช็คซัมของยูดีพีนี้เพื่อเป็นการป้องกันข้อมูลที่อาจจะถูกแก้ไขหรือมีความผิดพลาดระหว่างการส่ง และหากเกิดเหตุการณ์ดังกล่าวปลายทางจะได้ทราบว่ามีข้อผิดพลาด แต่เป็นการตรวจสอบเพียงฝ่ายเดียวเท่านั้น โดยในข้อกำหนดของยูดีพี หากพบว่ามีความผิดพลาดของเช็คซัม (Checksum Error) ก็ให้ผู้รับปลายทางทำการทิ้งข้อมูลนั้นเสียแต่ไม่มีการแจ้งกลับไปยังผู้ส่งแต่อย่างใด จุดนี้เองที่ทำให้ยูดีพีเป็นโปรโตคอลที่ไม่มีเสถียรภาพและเชื่อถือไม่ค่อยได้ การรับและส่งข้อมูลแต่ละครั้งหากมีข้อผิดพลาดใน

ระดับไอพี เช่น ส่งไม่ถึง หมดเวลา ผู้ส่งจะได้รับข้อความแสดงความผิดพลาดจากระดับไอพี เป็น ICMP Error Message แต่เมื่อข้อมูลส่งถึงปลายทางถูกต้องแต่เกิดข้อผิดพลาดในส่วนของยูดีพีเอง กลับไม่มีการยืนยันหรือแจ้งให้ทราบแต่อย่างใด

ขนาดของยูดีพี ดาต้าแกรมในทางทฤษฎีแล้วยูดีพี ดาต้าแกรม จะมีขนาดได้สูงสุดเท่ากับ 65535 ไบต์ โดยแบ่งเป็นไอพี เฮดเดอร์ เสีย 20 ไบต์ และเป็นยูดีพี เฮดเดอร์ อีก 8 ไบต์ คงเหลือส่วนที่เป็นข้อมูลเท่ากับ 65507 ไบต์ อย่างไรก็ตามการนำยูดีพีไปใช้งานส่วนใหญ่จะใช้ขนาดของดาต้าแกรมที่ต่ำกว่านี้ โดยทั่วไปขีดจำกัดของยูดีพี ดาต้าแกรมจะมีอยู่ 2 ประการคือ

แอพพลิเคชั่น : ในการรับส่งแต่ละครั้ง แอพพลิเคชั่นจะต้องทำการจองพื้นที่หน่วยความจำเพื่อทำการรับ-ส่งข้อมูลให้เหมาะสมกับสภาพของโฮสต์และคุณสมบัติของแอพพลิเคชั่นเองด้วย เช่น หากกำหนดขนาดใหญ่เกินไปก็จะทำให้เปลืองหน่วยความจำ หรือการรับส่งข้อมูลจำนวนหนึ่งเท่านั้น มิได้ต้องการเต็มทั้ง 64K ดังนั้นโดยทั่วไปขนาดของยูดีพี ดาต้าแกรมจะถูกตั้งค่าไว้เท่ากับ 8192 ไบต์

ขนาดของไอพี ดาต้าแกรม : โดยส่วนใหญ่การนำทีซีพี/ไอพี ไปใช้งานกับระบบปฏิบัติการต่างๆจะมีการกำหนดขนาดของไอพี ดาต้าแกรมให้ต่ำกว่า 64 K จึงทำให้ยูดีพี ดาต้าแกรม ซึ่งต้องอาศัยอยู่ในไอพี ดาต้าแกรมอีกชั้นหนึ่งถูกจำกัดขนาดไปด้วยโดยปริยาย

3.3 การออกแบบโปรแกรมในส่วนของการจัดการเรื่องการบีบอัดเสียง

3.3.1 การเข้ารหัสเสียงด้วยมาตรฐาน G.711

ในส่วนนี้เราจะทำการบีบอัดและขยายข้อมูล (Compression/Decompression) โดยใช้มาตรฐาน ITU-T ด้วย G.711 ที่ชื่อว่า PULSE CODE MODULATION (PCM) OF VOICE FREQUENCIES โดยเราจะแสดงคุณลักษณะการบีบอัดด้วยอัลกอลิทึมตามมาตรฐาน ITU-T

จากกฎการคอมเพนดิ้ง ซึ่งถ้าสร้างตัวควอนไทซ์อย่างตรงไปตรงมาจะต้องใช้จำนวนควอนไตซ์สเตปจำนวนมาก คือใช้จำนวนบิตมากประกอบกับเหตุผลที่ว่าการที่จะลดต้นทุนในการสร้างตัวควอนไตซ์บบไม่สม่ำเสมอนั้น จะต้องพยายามลดจำนวนควอนไตซ์สเตปไซท์ให้มีจำนวนน้อยที่สุด ดังนั้นเพื่อให้ได้ข้อดีของกระบวนการบีบอัดสัญญาณและราคาต้นทุนต่ำ จึงเกิดวิธีการที่เรียกว่า การคอมแพนดิ้งแบบแบ่งเป็นส่วนๆ (Segmented Companding) วิธีการนี้ ใช้หลักการประมาณเส้นโค้งของการบีบอัด ด้วยส่วนของเส้นตรงหลายเส้นที่ต่อกันในแต่ละส่วนของเส้นตรงทำการควอนไตซ์แบบเชิงเส้นและจะเรียกส่วนเส้นตรงแต่ละส่วนว่า "เซกเมนต์" (segment)

3.3.1.1 การบีบสัญญาณของคอมเพรสเซอร์ (Compressor) ลักษณะเชิงล็อกกาลิทึม

3.3.1.1.1 μ เป็นค่าพารามิเตอร์ที่ใช้สำหรับปรับลักษณะการบีบสัญญาณ

เมื่อปรับค่าพารามิเตอร์ μ

$\mu = 0$	Value at segment end points	Decision value
8	8159	128
7	4063	64
6	295	32
5	991	16
4	479	8
3	223	4
2	95	2
1	31	0

ตารางที่ 3.1 แสดงค่าที่ได้จากการคำนวณแบบ μ , μ =0

$\mu = 100$	Value at segment end points	Decision value
8	8159	127
7	4063	108
6	295	89
5	991	71
4	479	54
3	223	39
2	95	26
1	31	9

ตารางที่ 3.2 แสดงค่าที่ได้จากการคำนวณแบบ μ , μ =100

การคอมแพนดิ้งแบบ μ , μ =255 จะประมาณเส้นโค้งด้วยส่วนของเส้นตรง 15 เซกเมนต์ แบ่งเป็น 8 เซกเมนต์ สำหรับช่วงบวก และอีก 8 เซกเมนต์สำหรับช่วงลบ แต่เนื่องจากเซกเมนต์แรกของทั้งช่วงบวกและช่วงลบอยู่ในแนวเดียวกันซึ่งผ่านจุดกำเนิด จึงถือเป็นเซกเมนต์เดียวกัน รวมแล้วจึงเป็น 15 เซกเมนต์ ซึ่งแสดงเฉพาะช่วงบวกของสัญญาณ

การจัดเซกเมนต์จัดได้โดยให้เซกเมนต์ใดๆที่ติดกันรับช่วงสัญญาณอินพุทต่างกัน 2 เท่า คือเซกเมนต์ที่ 8 จะรับช่วงสัญญาณอินพุทเป็น 2 เท่าของเซกเมนต์ที่ 7 และเซกเมนต์ที่ 7 จะรับช่วงเป็น 2 เท่าของเซกเมนต์ที่ 6 เป็นเช่นนี้ไปเรื่อยๆ ทำให้ความชันของเซกเมนต์ที่ติดกันต่างกันอยู่ 2 เท่าด้วย รหัสที่ได้จากการควอนไทซ์มีขนาด 8 บิต มีรายละเอียดคือ

B1 (MSB) บิตเครื่องหมาย (sign bit) เป็นบิตที่ใช้แทนช่วงของสัญญาณอินพุท โดย 0 แทนช่วงลบ และ 1 แทนช่วงบวก

B2 – B4 ค่าเซกเมนต์ เป็นกลุ่มบิตที่ใช้ระบุหมายเลขเซกเมนต์ ซึ่งมี 8 เซกเมนต์

B5 – B8 ค่าควอนไตซ์สเตป เป็นกลุ่มบิตที่ใช้ระบุควอนไตซ์สเตปในแต่ละเซกเมนต์ซึ่งมี 16 ควอนไตซ์สเตป

หมายเลขเซกเมนต์	จำนวนสเตปต่อเซกเมนต์	ขนาดสเตป	แอมปลิจูดอินพุท (x_n to x_{n+1})	เอาท์พุทของการเข้ารหัส			
				ค่ารหัสปกติ	รหัสเซกเมนต์	รหัสควอนไทซ์	เอาท์พุทจากการถอดรหัส y_n
1	32	1	0-1	0	111	1111	0
		2	1-3	1		1110	2
			...			...	...
			29-31	15		0000	30
2	16	4	31-35	16	110	1111	33
			...	...		...	...
			91-95	31		0000	93
3	16	8	95-103	32	101	1111	99
			...	...		...	...
			215-223	47		0000	219
4	16	16	223-239	48	100	1111	231
			...	...		...	...
			463-473	63		0000	471
5	16	32	479	64	011	1111	495
			...	...		...	...
			959-991	79		0000	975
6	16	64	991-1055	80	010	1111	1023
			...	...		...	...
			1951-2015	95		0000	1983
7	16	128	2015-2143	96	001	1111	2079
			...	...		...	...
			3935-4063	111		0000	3999
8	16	256	4063-4319	112	000	1111	4191
			...	...		...	...
			7903-8159	127		0000	8031

*ค่าเต็มสเกลของช่วงแอมปลิจูดอินพุทนอมอลไลซ์ไว้ที่ 4096 และบิตเครื่องหมาย (B1) = 0 สำหรับค่าอินพุท ลบ1 อินพุทค่าบวก

*เอาท์พุทจากการถอดรหัสหาได้จาก $y_0=x_0=0$ และ $yn = (x_n + x_{n-1})/2$,$n=1,2,...,127$

ตารางที่ 3.3 แสดงการเข้ารหัส พีซีเอ็ม สำหรับเซกเมนต์คอมแพนดิ้งแบบ μ , μ =255

จากตารางที่ 3.3 จะเห็นว่ารหัสที่ไม่ได้อยู่ในรูปของรหัสเลขฐานสองปกติ ตัวอย่าง เช่น ในช่วงแอมพลิจูดอินพุท 0-1 แทนที่รหัสจะเป็น 000-0000 (B2 – B8) ตามค่ารหัสปกติแต่กลับเป็น 111-1111 เนื่องจากเป็นการเข้ารหัสรูปคลื่นแบบไบโพล่าจะมีเสถียรภาพของการซิงโครไนซ์ที่ดีกับบิตข้อมูลที่มีความหนาแน่นของระดับลอจิก 1มากๆ แต่เป็นที่ทราบแล้วว่าค่าของความน่าจะเป็นสำหรับเสียงพูดอยู่ที่ระดับสัญญาณต่ำๆ จึงไม่เหมาะในการใช้ส่งข้อมูล ดังนั้นจึงใช้วิธีการคอมพลีเมนท์บิต B2 - B8 ของรหัสในรูปปกติ ซึ่งจะทำให้ค่าความหนาแน่นของระดับลอจิก 1 มีค่าสูง และสำหรับรหัสที่มีค่าปกติเป็น 0111-1111 (-127) ซึ่งอยู่ในช่วงลบของสัญญาณจะเห็นว่าเมื่อเปลี่ยนแปลงแล้วรหัสที่ได้จะเป็น 0 ทั้งหมด ซึ่งในทางปฏิบัติจะไม่ใช้รหัสนี้แต่จะแทนด้วย 0000-0010 (-125) ซึ่งจะทำให้แน่ใจได้ว่าจะไม่มีโอกาสเกิดลอจิก 0 ติดๆ กันมากกว่า 13 บิต

3.3.1.1.2 A เป็นค่าพารามิเตอร์ที่ใช้สำหรับปรับลักษณะการบีบสัญญาณ

จากรูปที่ 2.16 จะเห็นได้ว่า เมื่อปรับค่าพารามิเตอร์ A

A=1	Value at segment end points	Decision value
7	4096	128
6	2408	64
5	1204	32
4	512	16
3	256	8
2	128	4
1	64	2
	0	0

ตารางที่ 3.4 แสดงค่าที่ได้จากการคำนวณแบบ A, A=1

A=10	Value at segment end points	Decision value
7	4096	128
6	2408	100
5	1204	74
4	512	47
3	256	24
2	128	12
1	64	6
	0	0

ตารางที่ 3.5 แสดงค่าที่ได้จากการคำนวณแบบ A, A=10

A=100	Value at segment end points	Decision value
7	4096	128
6	2408	112
5	1204	96
4	512	80
3	256	65
2	128	49
1	64 0	36 0

ตารางที่ 3.6 แสดงค่าที่ได้จากการคำนวณแบบ A, A = 100

ส่วนการคอมแพนดิ้งแบบ A,A = 87.6 จะประมาณเส้นโค้งด้วยส่วนของเส้นตรง 13 เซกเมนต์ โดย 2 เซกเมนต์แรกของทั้งช่วงบวกและช่วงลบ อยู่ในแนวของเส้นตรงเดียวกันจึงถือเป็นเซกเมนต์เดียวกัน การจัดเซกเมนต์จะจัดให้เซกเมนต์ที่ติดกันรับช่วงอินพุตต่างกัน 2 เท่า ซึ่งคล้ายกับแบบ μ ยกเว้น 2 เซกเมนต์แรกจะมีขนาดเท่ากัน สองเซกเมนต์นี้จึงอยู่ในแนวเส้นตรงเดียวกันจึงทำให้เมื่อนับตามแนวเส้นตรงแล้วจะมีเพียง 7 เซกเมนต์ (เดิมถ้านับตามบิต B2 – B4 จะมี 8 เซกเมนต์) จากรูปความชันของเซกเมนต์แรกสามารถคำนวณได้คือ

$$Slope_1 = \frac{32/128}{64/4096} = 16 \qquad (3.1)$$

จากสมการ (3.1) จะได้ค่าความชัน

$$Slope_1 = \frac{d}{dV_i}[V_c] = \frac{A}{1+InA} \qquad (3.2)$$

แทนค่า A=87.6

$$\left.\frac{A}{1+InA}\right|_{A=87.6} = 16 \qquad (3.3)$$

จะเห็นได้ว่าค่าที่ได้สอดคล้องกับทฤษฎี แสดงให้เห็นว่าเซกเมนต์แรกมีความเป็นเชิงเส้นอย่างแท้จริง

B1 (MSB) บิตเครื่องหมาย (sign bit) เป็นบิตที่ใช้แทนช่วงของสัญญาณอินพุท โดย 0 แทนช่วงลบ และ 1 แทนช่วงบวก

B2 – B4 ค่าเซกเมนต์ เป็นกลุ่มบิตที่ใช้ระบุหมายเลขเซกเมนต์ ซึ่งมี 7 เซกเมนต์ (รหัส 000 และ 001 อยู่ในเซกเมนต์เดียวกัน)

B5 – B8 ค่าควอนไตซ์สเตป เป็นกลุ่มบิตที่ใช้ระบุควอนไตซ์สเตปในแต่ละเซกเมนต์ ซึ่งมี 16 ควอนไตซ์สเตป

หมายเลขเซกเมนต์	จำนวนสเตปต่อเซกเมนต์	ขนาดสเตป	แอมปลิจูดอินพุต (x_n to x_{n+1})	เอาท์พุทของการเข้ารหัส			
				ค่ารหัสปกติ	รหัสเซกเมนต์	รหัสควอนไทซ์	เอาท์พุทจากการถอดรหัส y_n
1	32	2	0-2	0	101	0101	1
			...			...	...
			30-32	15		1010	31
			32-34	16	100	0101	33
			...			...	...
			62-64	31		1010	63
2	16	4	64-68	32	111	0101	66
			...	...		...	...
			124-128	47		1010	126
3	16	8	128-136	48	110	0101	132
			...	...		...	...
			248-256	63		1010	252
4	16	16	256-272	64	001	0101	264
			...	...		...	...
			496-512	79		1010	502
5	16	32	512-544	80	000	0101	528
			...	...		...	...
			992-1024	95		1010	1008
6	16	64	1024-1088	96	011	0101	1056
			...	...		...	...
			1984-2048	111		1010	2016
7	16	128	2048-2176	112	010	0101	2112
			...	...		...	...
			3968-4096	127		1010	4032

*ค่าเต็มสเกลของช่วงแอมปลิจูดอินพุทนอมอลไลซ์ไว้ที่ 4096 และบิตเครื่องหมาย (B1)=0 สำหรับค่าอินพุท ลบ1 อินพุทค่าบวก

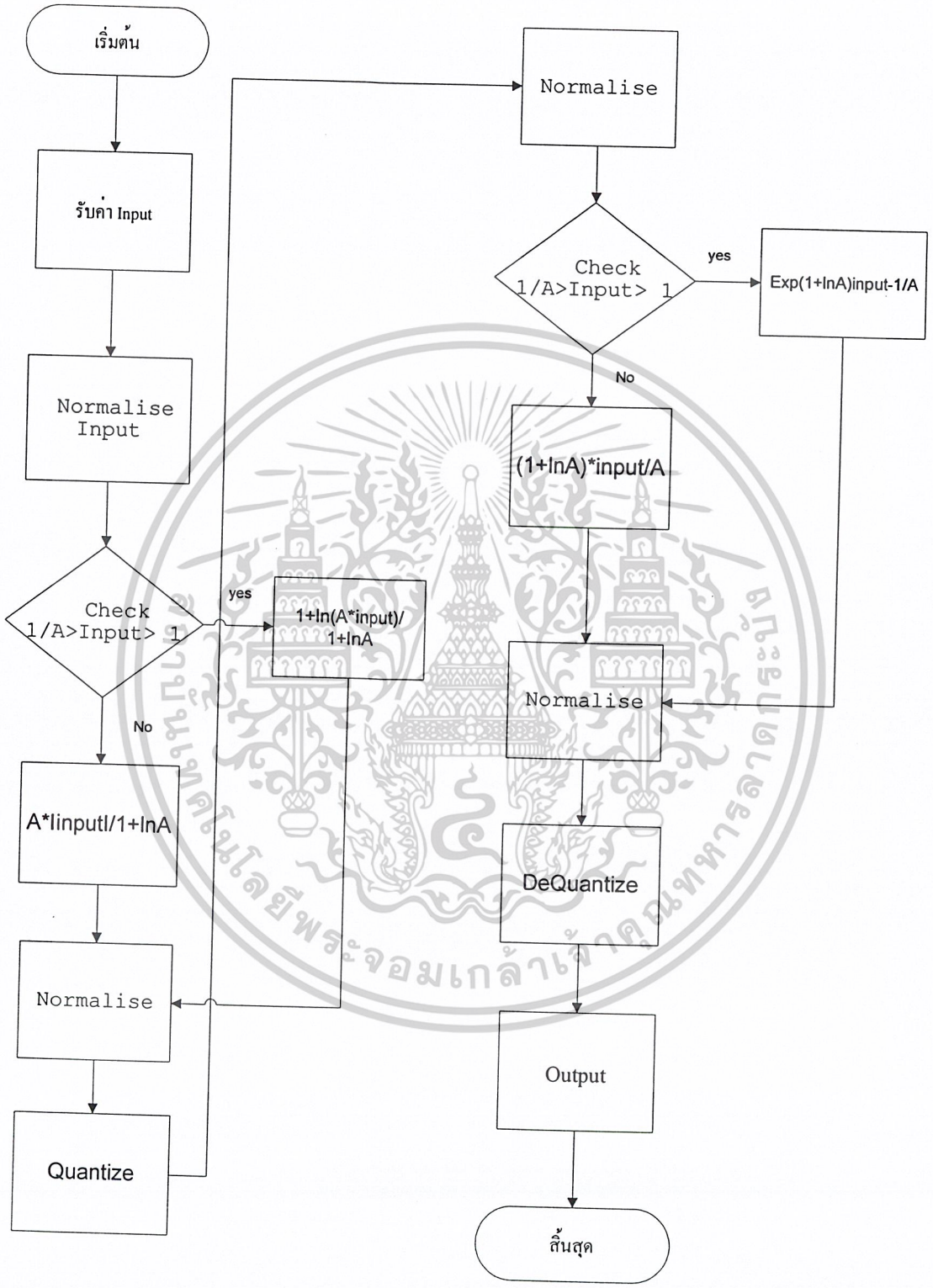
*เอาท์พุทจากการถอดรหัสหาได้จาก $yn = (x_n + x_{n-1})/2$,$n=1,2,....,127$

ตารางที่ 3.7 แสดงการเข้ารหัส พีซีเอ็ม สำหรับเซกเมนต์คอมแพนดิ้งแบบ A, A=87.6

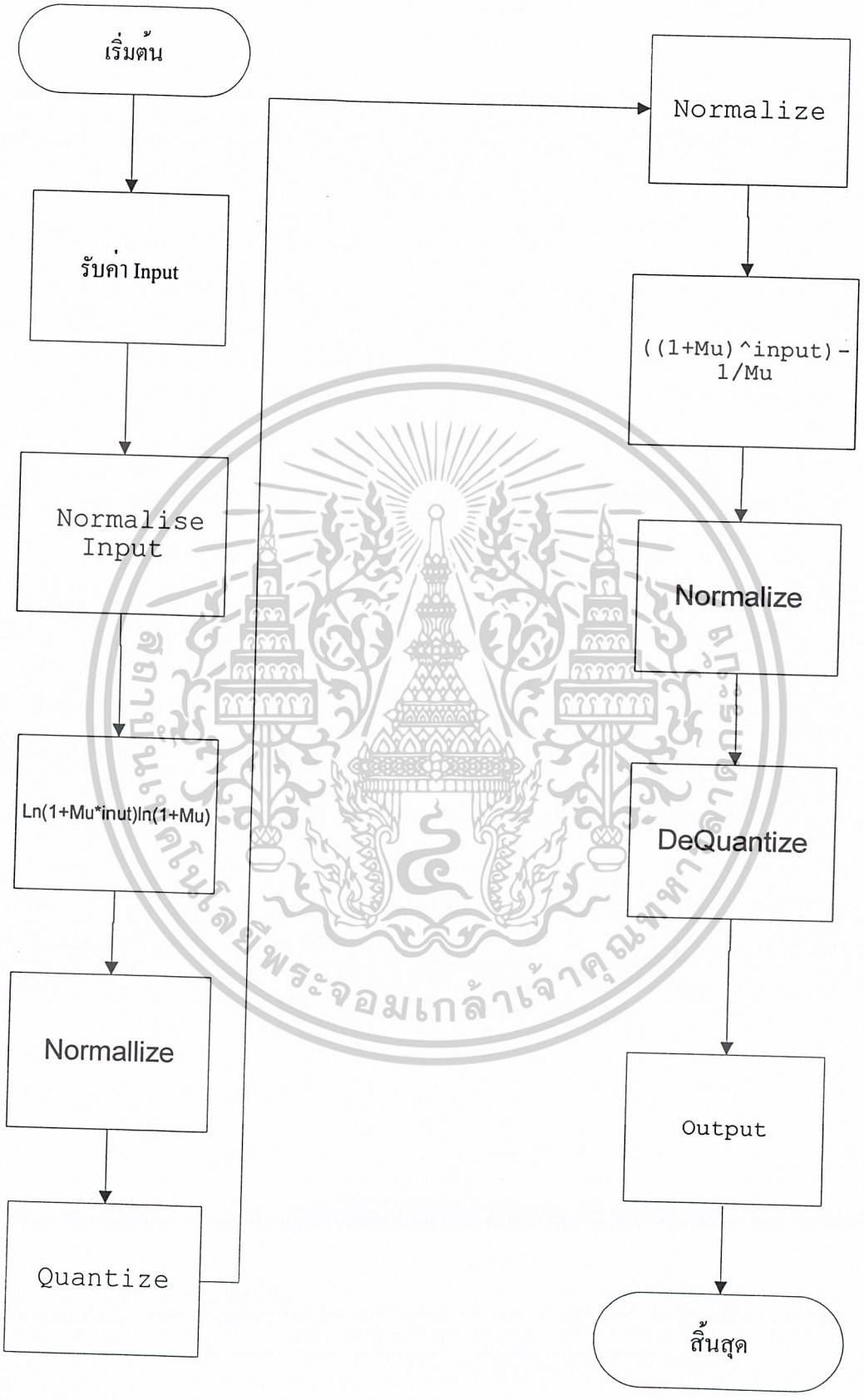
รายละเอียดของการเข้ารหัสแสดงดังตารางที่ 3.7 จากตารางจะเห็นว่าค่ารหัส B2 – B8 ไม่ได้อยู่ในรูปรหัสฐานสองปกติ ซึ่งเป็นเหตุผลเดียวกับการคอมแพนดิ้งแบบ u คือ เพื่อเสถียรภาพของการซิงโครไนซ์ โดย B2- B8 ได้จากการคอมพลีเมนต์บิตลำดับคู่ (B2, B4, B6, B8) ของรหัสปกติ เช่น รหัสปกติเป็น 0000-0001 เมื่อแปลงแล้วจะเป็น 0101-0100

3.3.1.1.3 การจัดการเรื่องคอมเพรสชันเสียง

ใช้โปรแกรมแมทเลปทำอัลกอริทึมในการทำการบีบอัดและขยายข้อมูล ด้วยวิธีพีซีเอ็มแบบเชิงเส้นและไม่เชิงสัน โดยการกำหนดปรับค่าพารามิเตอร์ A และ μ โดยจะแสดงผังการทำงานดังรูป

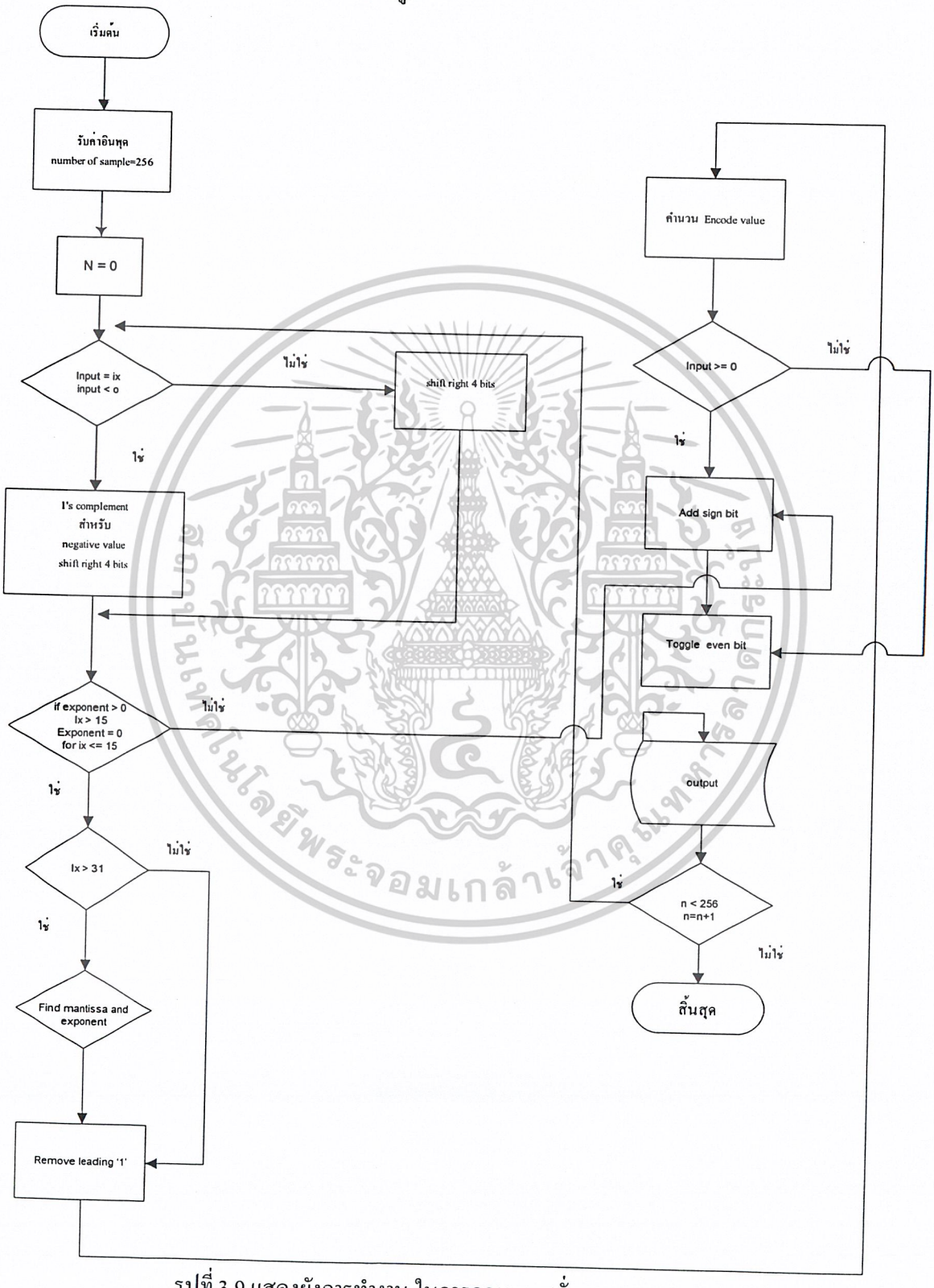


รูปที่ 3.7 แสดงผังการทำงานของการบีบอัดและขยายข้อมูลแบบ A-Law

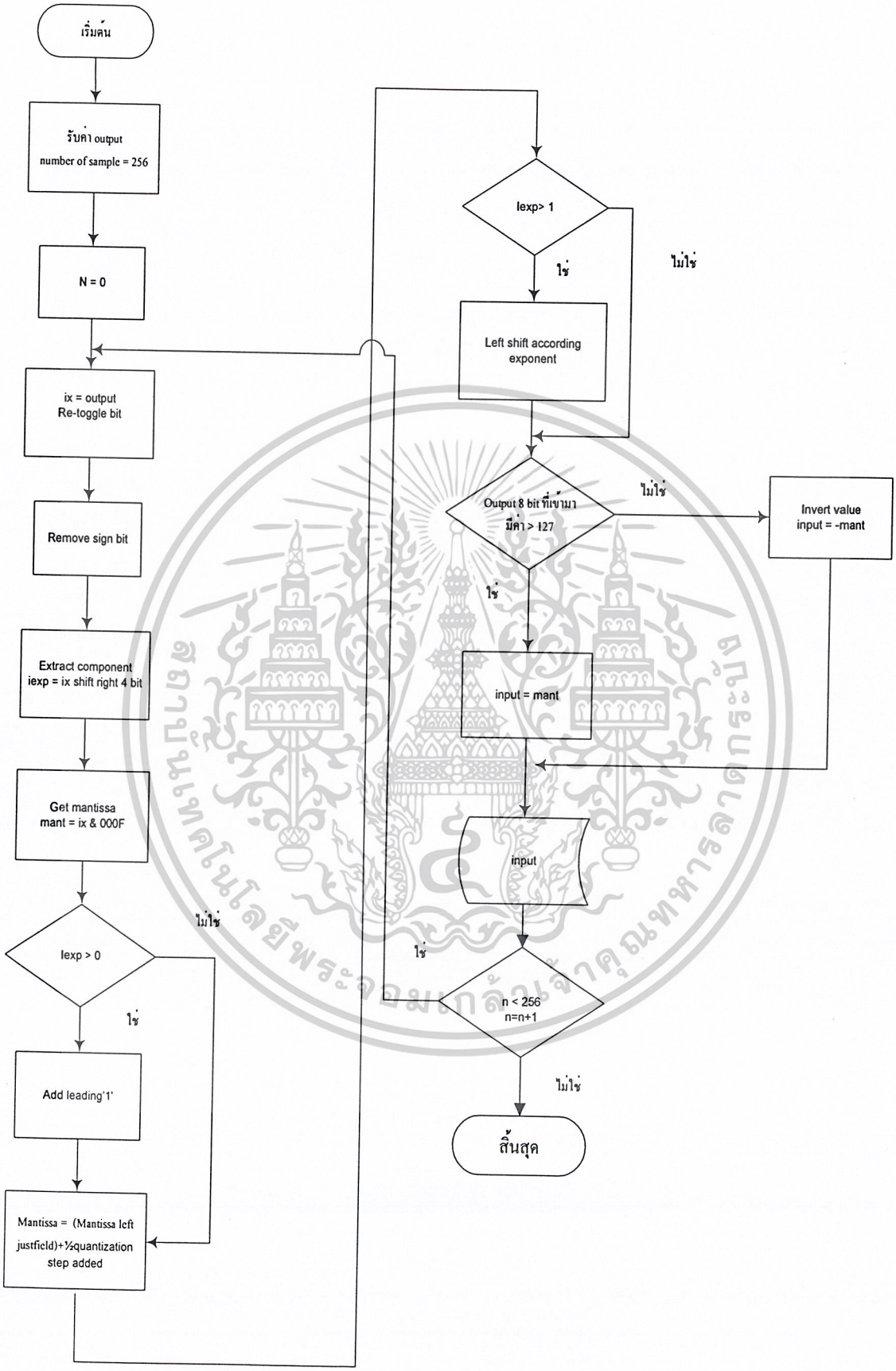


รูปที่ 3.8 แสดงผังการทำงานของการบีบอัดและขยายข้อมูลแบบ μ-Law

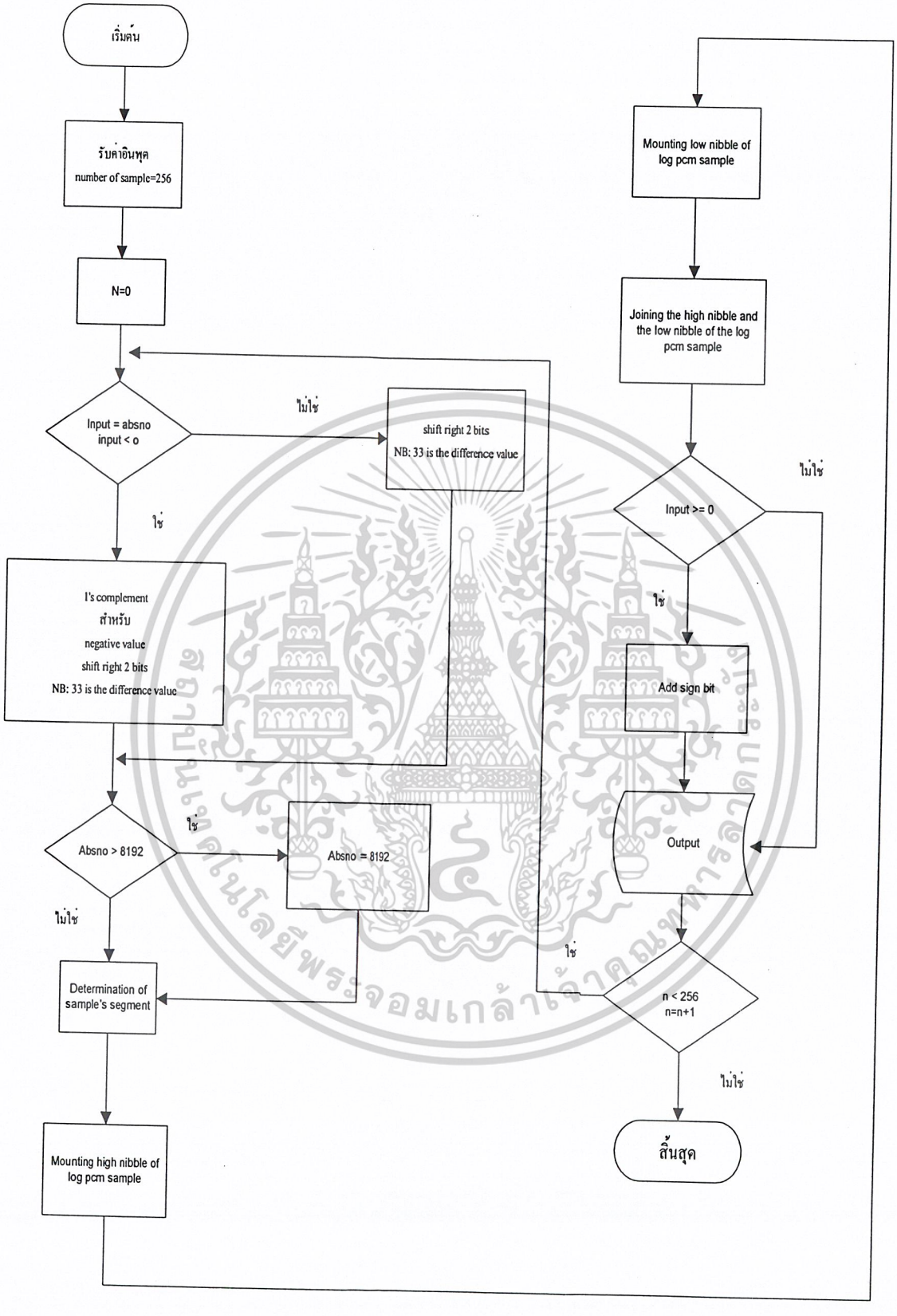
และจะใช้โปรแกรมภาษาซีทำอัลกอริทึมในการทำการบีบอัดและขยายข้อมูลเพื่อทำการส่งผ่านโครงข่ายต่อไป ตามมาตรฐาน ITU-T ด้วย G.711 ที่ชื่อว่า PULSE CODE MODULATION (PCM) OF VOICE FREQUENCIES โดยจะแสดงผังการทำงานดังรูป



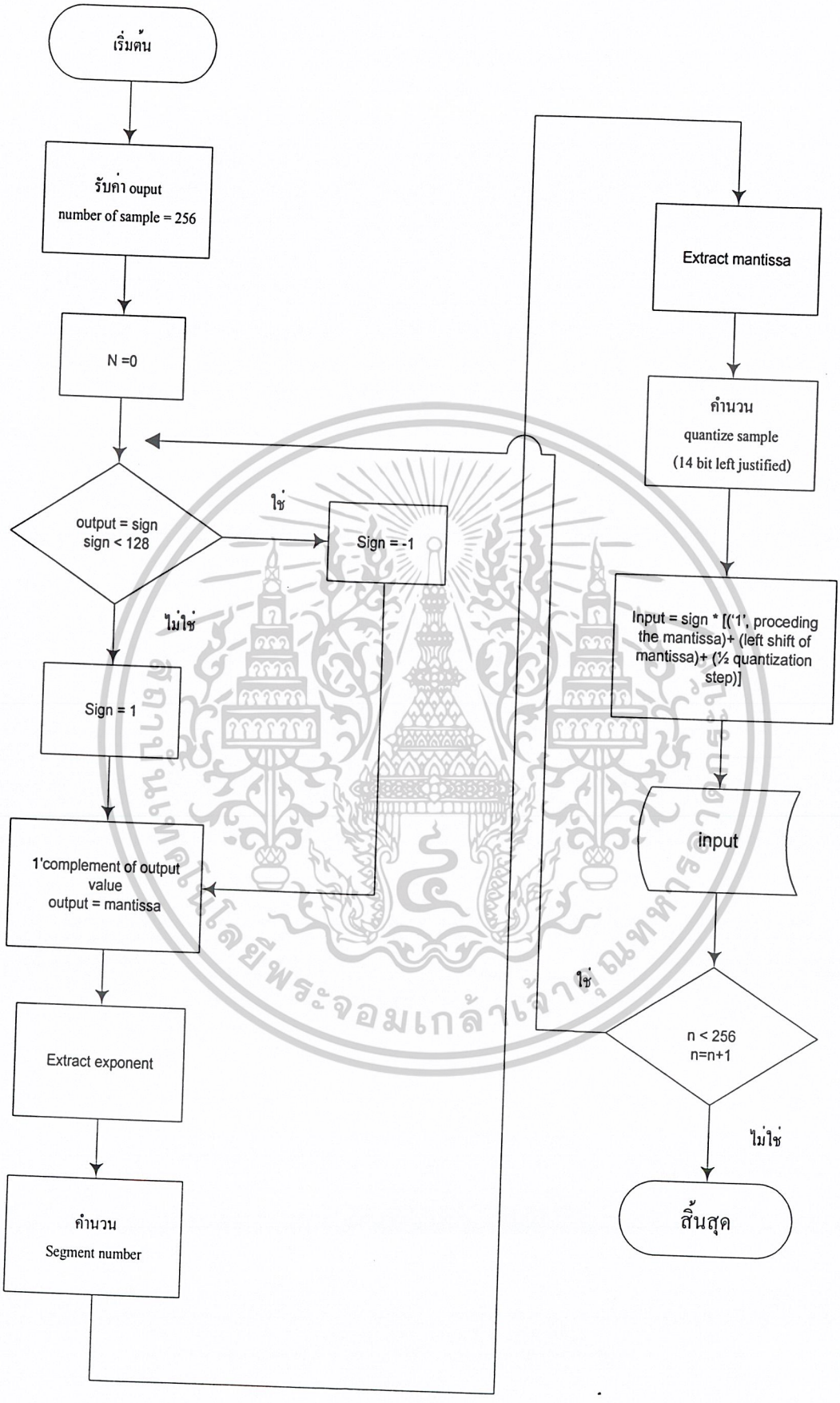
รูปที่ 3.9 แสดงผังการทำงาน ในการคอมเพรสชั่น แบบ A –Law



รูปที่ 3.10 แสดงผังการทำงาน ในการเอ็กซแพนด์ดิ้ง แบบ A –Law



รูปที่ 3.11 แสดงผังการทำงาน ในการคอมเพรสชั่น แบบ μ –Law

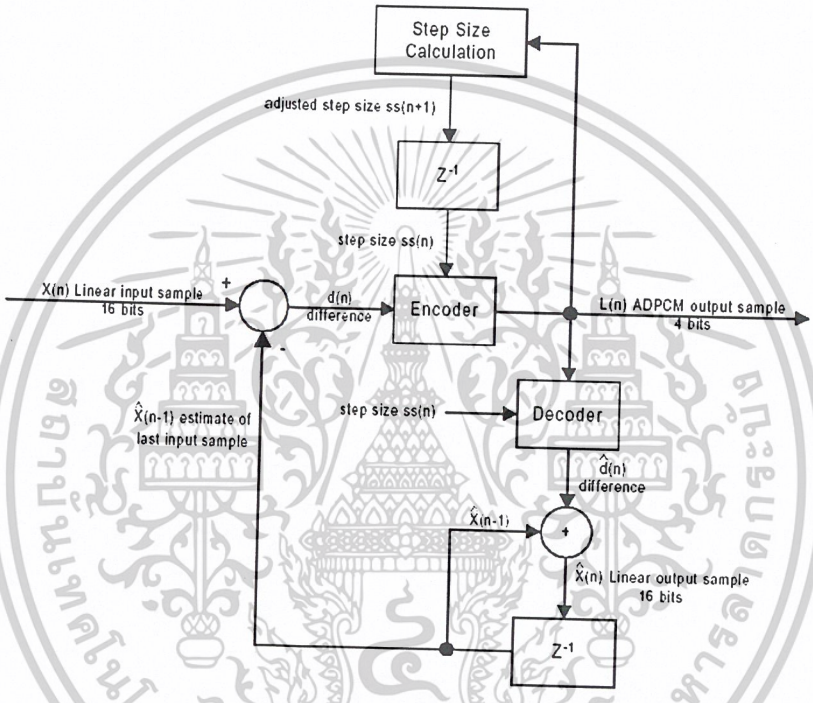


รูปที่ 3.12 แสดงผังการทำงาน ในการเอ็กซแพนดิ้ง แบบ μ –Law

3.3.2 การเข้ารหัสเสียงด้วยมาตรฐาน G.726

3.3.2.1 การเข้ารหัสเอดีพีซีเอ็ม

การเข้ารหัสเสียงด้วนมาตรฐาน G.726 เป็นการเข้ารหัสเสียงด้วยวิธีเอดีพีซีเอ็ม จากรูปที่ 3.13 เป็นบล็อกไดอะแกรมของการเข้ารหัสด้วยวิธีเอดีพีซีเอ็ม ซึ่งสามารถแบ่งเป็นขั้นตอนการทำงานได้ดังนี้ คือ ทำการแปลงสัญญาณเสียงอินพุท 64 Kbit/s ให้เป็นสัญญาณแบบยูนิฟอร์มพีซีเอ็ม ซึ่งได้ค่าแอมพลิจูดของสัญญาณแล้วใช้เป็นสัญญาณอินพุทในการเข้ารหัสสัญญาณ จากนั้นหาค่าผลต่างของสัญญาณ (difference signal) ซึ่งได้จากการลบระหว่างค่าคาดคะเนสัญญาณอินพุทปัจจุบันกับค่าสัญญาณอินพุทปัจจุบัน



รูปที่ 3.13 แสดงบล็อกไดอะแกรมการเข้ารหัสเอดีพีซีเอ็ม

เมื่อกำหนดให้

$d(n) =$ สัญญาณผลต่าง (Difference signal)

$Xp(n) =$ สัญญาณอินพุท (Input signal)

$Xp(n) =$ ค่าคาดคะเนสัญญาณอินพุทปัจจุบัน (Signal estimate)

จะได้ว่า

$$d(n) = X(n) - Xp(n) \tag{3.3}$$

ค่าสัญญาณผลต่างที่ได้จะถูกนำไปทำการจัดระดับการควอนไตซ์ตามค่าสเต็ปไซต์ปัจจุบัน [ss(n)] ซึ่งแทนลอจิกการเข้ารหัส และผลที่ได้คือ รหัสเอดีพีซีเอ็ม ซึ่งเป็นเอาท์พุท เพื่อนำไปทำการถอดรหัสให้ได้สัญญาณเสียงที่ใกล้เคียงกับสัญญาณเสียงเดิมกลับมา

ค่ารหัสเอดีพีซีเอ็มที่ได้จะถูกนำกลับไปใช้ในการคำนวณค่าสเต็ปไซต์ต่อไป : $ss(n+1)$ และอีกส่วนจะถูกป้อนเข้าไปยังอินเวอร์สอแด็พทีฟ ควอนไตซ์เซอร์ (Inverse adaptive quantizer) เพื่อสร้างสัญญาณผลต่างที่ถูก ควอนไตซ์ที่ใกล้เคียงกับสัญญาณผลต่างตัวเดิมกลับคืนมา

เมื่อกำหนดให้

$v(n) =$ สัญญาณผลต่างที่ถูกควอนไตซ์ (Quantized difference signal)

$q(n) =$ ค่าความผิดพลาดของการควอนไตซ์ (Quantization error)

จะได้ว่า

$$v(n) = d(n) + q(n) \tag{3.4}$$

และเนื่องจากการจัดระดับเป็นแบบอแด็พทีฟ การหาค่าสัญญาณผลต่างที่ถูกควอนไตซ์จึงเป็นแบบอแด็พทีฟ จากนั้นสัญญาณผลต่างที่ถูกควอนไตซ์จะถูกรวมเข้ากับค่าคาดคะเนสัญญาณอินพุทตัวก่อน(Previous signal estimate)

เพื่อสร้างสัญญาณกลับคืนมา(Reconstructed signal) ป้อนให้กับอแด็พทีฟพรีดิกเตอร์ (adaptive predictor) ร่วมกับสัญญาณผลต่างที่ถูกควอนไตซ์ เพื่อทำนายค่าคาดคะเนสัญญาณอินพุทตัวถัดไป

เมื่อกำหนดให้

$r(n)$ คือ สัญญาณที่สร้างกลับคืนมา (Reconstructed signal)

$X_p(n-1)$ คือ ค่าคาดคะเนสัญญาณอินพุทตัวก่อน (Previous signal estimate)

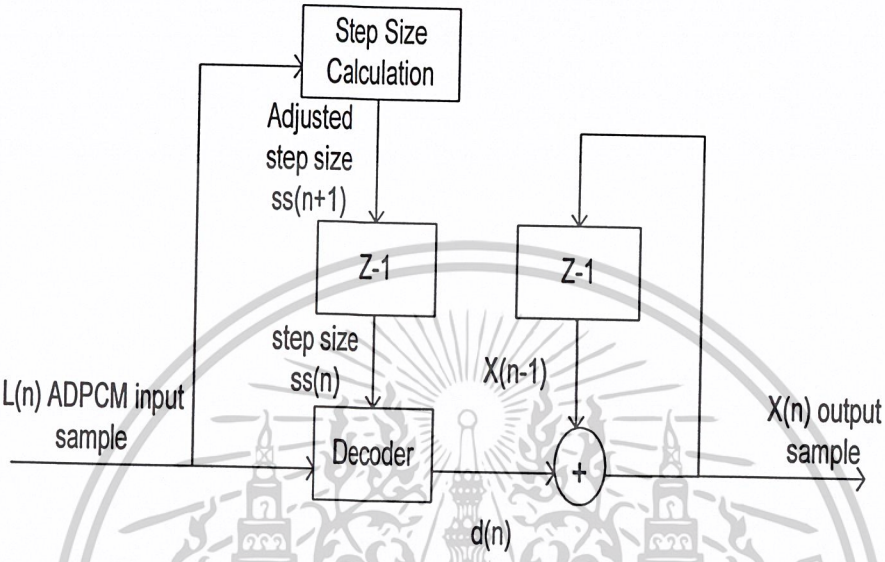
จะได้ว่า

$$r(n) = Xp(n-1) + v(n) \tag{3.5}$$

ค่าคาดคะเนสัญญาณอินพุทที่ได้จะนำไปเปรียบเทียบกับสัญญาณอินพุทปัจจุบันที่เข้ามาใหม่ เพื่อหาค่าสัญญาณผลต่างตัวถัดไป แล้วนำไปสร้างเป็นรหัสเอดีพีซีเอ็มตัวถัดไป

3.3.2.2 การถอดรหัสเอดีพีซีเอ็ม

ในรูปที่ 3.14 จะแสดงในส่วนของบล็อกไดอะแกรมของกระบวนการถอดรหัสเอดีพีซีเอ็ม โดยโครงสร้างของกระบวนการถอดรหัสสัญญาณจะเหมือนกับส่วนป้อนกลับของสัญญาณที่กระบวนการเข้ารหัสสัญญาณ ซึ่งจะรับค่ารหัสเอดีพีซีเอ็ม และค่าสเตปไซต์ โดยเป็นการคำนวณผลของค่าความแตกต่างซ้ำไปอีกครั้ง และจะทำการสะสมค่าที่ประมาณได้ให้อยู่ในค่าของสัญญาณอินพุทที่กระบวนการเข้ารหัส(x)



รูปที่ 3.14 แสดงบล็อกไดอะแกรมการถอดรหัส เอดีพีซีเอ็ม

เมื่อกำหนดให้

$\hat{X}$ = สัญญาณที่ได้จากการถอดรหัสของเอดีพีซีเอ็ม

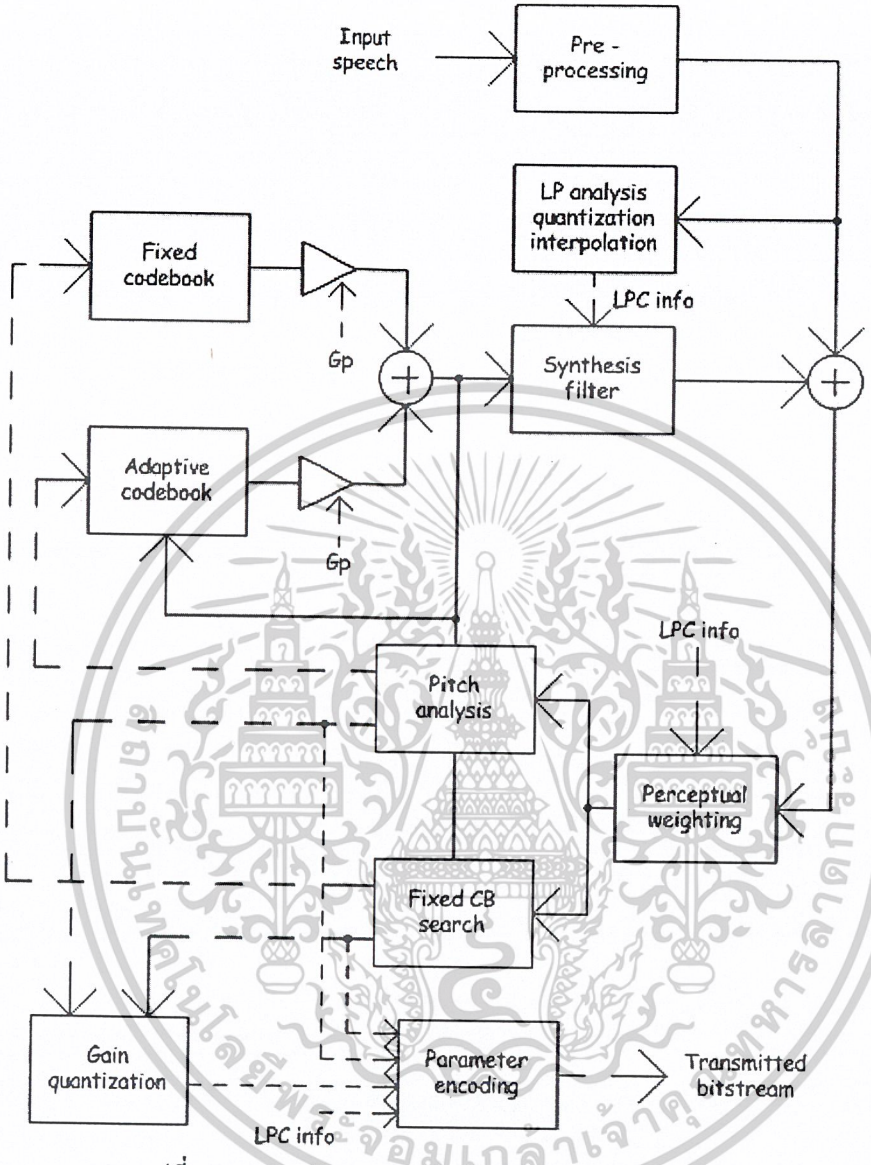
จะได้ว่า

$$\hat{X}(n) = X(n-1) + d(n) \tag{3.6}$$

3.3.3 การเข้ารหัสเสียงด้วยมาตรฐาน G.729

3.3.3 การเข้ารหัสเสียงด้วยมาตรฐาน G.729

3.3.3.1 การเข้ารหัส CS-ACELP



รูปที่ 3.15 แสดงผังการทำงานของ CS-ACELP Encoder

ดังรูปที่ 3.15 สัญญาณอินพุทจะถูกกรองผ่านวงจรกรองความถี่สูงผ่านและแบ่งผ่านขั้นตอนการเตรียมสัญญาณก่อนเริ่มกระบวนการเข้ารหัสสัญญาณ เป็นการจัดเตรียมสัญญาณอินพุทเพื่อที่ผ่านของไปวิเคราะห์ LP จะทำการวิเคราะห์ทุกๆ 10 ms (1 frame) เพื่อที่จะประมวลหาค่าสัมประสิทธิ์ LP (LPC) และนำค่าสัมประสิทธิ์เหล่านี้ไปเปลี่ยนเป็น Line Spectrum Pairs (LSP) และถูกควอนไตซ์โดยวิธี two-stage Vector Quantization (VQ) ได้ LSP จำนวน 18 บิต ส่วนสัญญาณกระตุ้นหาโดยการเลือกสัญญาณที่นำมาแทนด้วยวิธีการ analysis by synthesis หา error ระหว่างสัญญาณต้นฉบับและสัญญาณที่สร้างขึ้น ให้น้อยที่สุด โดย error signal นี้จะถูกกรองผ่าน perceptual weighting filter ซึ่งค่าสัมประสิทธิ์ของวงจรกรองนั้นได้มาจากการ unquantized LP filter ซึ่งเป็นการปรับปรุงสัญญาณให้ตอบสนองกับความถี่ได้ดีขึ้น

พารามิเตอร์ของสัญญาณกระตุ้น (Fixed and Adaptive-codebook parameters) จะถูกหาทุกๆ 5ms (40 samples) quantized และ unquantized LP filter coefficient จะถูกใช้เฟรมย่อยที่สอง การหา Pitch นั้นจะถูกคำนวณหาทุก 10ms โดยคร่าวๆ (Open Loop) และจะทำการคำนวณซ้ำในแต่ละเฟรมย่อย (Close loop) โดยใช้พื้นฐานของความคล้ายคลึงกัน (Correlation) เป็นคาบของสัญญาณเสียงที่ได้รับ หลังจากนั้น Impulse response $h(n)$ ของ weighted synthesis filter จะถูกคำนวณ และสัญญาณเป้าหมายที่เราต้องการคือ $x(n)$ ซึ่งคืออินพุทของระบบ ถูกคำนวณโดยใช้ LP residual $r(n)$ ลบออกด้วยความผิดพลาดระหว่าง LP residual และ excitation signal ซึ่งเปรียบเทียบได้กับการลบออกด้วย the zero-input response แล้วนำไปผ่าน weighted synthesis filter ต่อไปเป็นขั้นตอนของ Close-loop pitch analysis ซึ่งพยายาม maximize correlation ระหว่าง $x(n)$ กับ $r(n) * h(n)$ เพื่อหาค่าที่ละเอียดของ Pitch นั้นถูก encode เป็น 8 บิต สำหรับเฟรมย่อยแรก และ 5 บิต สำหรับเฟรมย่อยที่ 2 ซึ่ง Pitch นั้นถูก encode สัมพัทธ์กัน รวมถึง gain ของ Adaptive code-book ด้วยและ $x(n)$ ที่ถูกกำหนดให้ลบออกด้วย adaptive code-book gain*filter adaptive code-book vector (filtered adaptive-codebook contribution) และจะได้สัญญาณเป้าหมายใหม่ คือ $x'(n)$ ซึ่งจะถูกใช้ในกระบวนการหาพารามิเตอร์ของ fixed codebook ซึ่งจะได้ vector 17 บิต สำหรับ fixed-codebook Gain ของ adaptive และ fixed codebook เป็น vector ที่ถูกควอนไตซ์ด้วย 7 บิต (โดยใช้ MA prediction กับ fixed codebook gain) ท้ายที่สุด filter memory จะถูกกำหนดโดยใช้ excitation signal ที่หาได้

Parameter	Codeword	Subframe 1	Subframe 2	Total per frame
Line spectrum pairs	*L0, L1, L2, L3*			18
Adaptive-codebook delay	*P1, P2*	8	5	13
Pitch-delay parity	*P0*	1		1
Fixed-codebook index	*C1, C2*	13	13	26
Fixed-codebook sign	*S1, S2*	4	4	8
Codebook gains (stage 1)	*GA1, GA2*	3	3	6
Codebook gains (stage 2)	*GB1, GB2*	4	4	8
Total				80

ตารางที่ 3.8 แสดงการจัดสรรบิตของ CS-ACELP ใน 1 เฟรม

3.3.3.1.1 วิเคราะห์สัญญาณเสียงเบื้องต้น (Pre-processing)

ในการวิเคราะห์สัญญาณเสียง เราจำต้องมีการเตรียมข้อมูลการวิเคราะห์ก่อน โดยสมมุติว่าสัญญาณอินพุทเป็น 16 บิต PCM signal ซึ่งขั้นตอนในการเตรียมสัญญาณเข้ามี 2 กระบวนการ คือ

1. Signal scaling ใช้วิธีนำอินพุทมาหารด้วย 2 ลดโอกาสการเกิด Overflow ในกระบวนการทำ fixed point

2. High-Pass filtering เพื่อป้องกันสัญญาณที่มีความถี่ต่ำที่ไม่ต้องการ ตามสมการ

$$H_{h1}(z) = \frac{0.46363718 - 0.92724705z^{-1} + 0.46363718z^{-2}}{1 - 1.9059465z^{-1} + 0.9114024z^{-2}} \quad (3.7)$$

3.3.3.1.2 Linear prediction analysis and quantization

Short term analysis และ synthesis filter อาศัยพื้นฐานของ Linear prediction (LP) filter order ที่ 10 โดยจะถูกใช้วิธี correlation จัดรูป 1 ครั้ง ทุกๆ 1 เฟรมคำพูด ด้วย asymmetric window 30 ms ทุก 80 sample (10 ms) autocorrelation Coefficient ของ window speed จะถูกคำนวณและเปลี่ยนไปเป็น LP Coefficient โดยใช้ Levinson algorithm หลังจากนั้น LP coefficient จะถูกเปลี่ยนไปเป็น LSP domain สำหรับการแทรกบิตและการทำ Quantization ในการแทรกบิตที่ได้จาก Quantized และ unquantized filter จะถูกแปลงกลับไปเป็น LP filter coefficient อีกครั้ง เพื่อทำการสร้าง Synthesis and weighting filters สำหรับแต่ละเฟรมย่อย

3.3.2.1.3 Perceptual weighting

Perceptual weighting filters นั้นมีพื้นฐานมาจาก Unquantized LP filter coefficient และสามารถนำไปหาผลของการตอบสนองเชิงความถี่ของ filter $W(z)$ สำหรับการปรับที่ค่าของ γ_1 และ γ_2 เหมาะสมจะทำให้การทำ weighting เกิดประสิทธิภาพมาก ซึ่งจะต้องทำให้ γ_1 และ γ_2 นั้นเป็น spectral function แบบเดียวกับสัญญาณอินพุทซึ่งการทำนี้จะทำเพียงครั้งเดียวใน 10 ms และมีการประมาณในเฟรมย่อยแรกของแต่ละเฟรมจะทำให้การปรับสัญญาณเรียบขึ้น สำหรับ Spectral function นั้นจะสามารถหาได้จาก 2^{nd} order linear prediction filter ซึ่งได้มาจากผลพลอยได้ของ Lervinson – Durbin recursion โดย reflection coefficient k_i ถูกเปลี่ยนไปเป็น Log Area Ratio (LAR) coefficient o_i

$$o_i = \log\frac{(1.0+k_i)}{(1.0-k_i)} \qquad i = 1,2 \tag{3.8}$$

3.3.3.1.4 Open-loop pitch analysis

เพื่อที่จะลดความซับซ้อนในการหาสำหรับ Adaptive codebook delay (Pitch) ดังนั้นจุดในการหาจะถูกกำหนดให้อยู่รอบ ๆ Pitch คร่าว ๆ จากการทำ Open loop pitch analysis ซึ่งการทำขั้นตอนนี้จะทำเพียงครั้งเดียวต่อเฟรม ซึ่งจะต้องใช้หลักการของความเหมือนกัน (Correlation) ของสัญญาณเป็นคาบ ๆ

3.3.3.1.5 Computation of the impulse response

Impulse response $h(n)$ ของการ weight synthesis filter $W(z)/\hat{A}(z)$ จำเป็นต้องใช้ในการหา adaptive codebook และ fix codebook Impulse response $h(n)$ ถูกคำนวณไว้สำหรับแต่ละเฟรมย่อย โดยการกรองสัญญาณซึ่งประกอบไปด้วยค่าสัมประสิทธิ์ของ filter $A(z/\gamma_1)$ และทำการกรองต่อโดยใช้ $1/\hat{A}(z)$ และ $1/A(z/\gamma_2)$

3.3.3.1.6 Computation of the target signal

Target signal $x(n)$ ใช้สำหรับหา Adaptive codebook สามารถคำนวณได้จากนำ LP residual signal $(r(n))$ ลบออกด้วย error ของสัญญาณเสียงจริงกับสัญญาณเสียงที่สังเคราะห์ขึ้น และกรอง LP residual signal $r(n)$ ผ่านการรวมกันระหว่าง Synthesis filter $1/\hat{A}(z)$ และ weighting filter $A(z/\gamma_1)/A(z/\gamma_2)$

Weighting filter ($A(z/\gamma_1)/A(z/\gamma_2)$ จะได้สัญญาณ Past excitation signal $(y(n))$ $x(n)$ และ $y(n)$ ถูก shift ออกไปด้วย delay k แล้วหาความเหมือนกันเป็นคาบ เพื่อหา Pitch ที่ละเอียดขึ้น

3.3.3.1.7 Adaptive codebook search

Adaptive codebook parameter ประกอบด้วย Pitch และ gain โดย adaptive codebook นั้นถูกสร้างขึ้นโดยใช้ pitch filter excitation โดยการหา Adaptive codebook จะกระทำทุก 5 ms หรือทุก ๆ เฟรมย่อย ในเฟรมย่อยแรกนั้น Pitch T_1 ถ้าประมาณแล้วมีค่าต่ำกว่า 85 จะต้องถูกหาให้ละเอียดขึ้น โดยการประมาณค่าเศษส่วนซึ่งมีความละเอียด 1/3([19 1/3, 84 2/3]) และกรณีที่มากกว่า 85 จะประมาณเป็นจำนวนเต็มเท่านั้น ([85, 143]) สำหรับ sub frame ที่ 2 delay T_2 จะถูกใช้ร่วมกับการประมาณเศษส่วนด้วยความละเอียด 1/3 และอยู่ในช่วงของ ([int(T_1) – 5 2/3, int(T_1) + 4 2/3])

สำหรับ Pitch ของแต่เฟรมย่อยแรกใช้ closed loop analysis เพื่อลด weight MSE พบว่าจะมีค่าอยู่รอบ Open loop pitch ซึ่งจะไม่เกิน 6 samples ดังนั้นจะช่วยลดการหา Pitch ลงไปได้โดยกำหนดช่วงของการหาได้โดยอยู่ระหว่าง $t_{\min}$ และ $t_{\max}$

3.3.3.1.8 Fixed codebook – Structure and search

Fixed codebook มีพื้นฐานมาจาก algebraic codebook โดยใช้ ISPP ซึ่ง codebook แต่ละ codebook vector จะประกอบด้วยพัลส์ 4 พัลส์ที่ไม่ใช่ zero pulse ซึ่งแต่ละพัลส์จะมีทั้ง amplitude ที่เป็น +1 หรือ -1

Pulse	Sign	Positions
$i0$	$s0$: ± 1	$m0$:0, 5, 10, 15, 20, 25, 30, 35
$i1$	$s1$: ± 1	$m1$:1, 6, 11, 16, 21, 26, 31, 36
$i2$	$s2$: ± 1	$m2$:2, 7, 12, 17, 22, 27, 32, 37
$i3$	$s3$: ± 1	$m3$:3, 8, 13, 18, 23, 28, 33, 38 4, 9, 14, 19, 24, 29, 34, 39

ตารางที่ 3.9 แสดงโครงสร้างของ Fixed codebook

3.3.3.1.9 Quantization of the gains

Adaptive-codebook gain (pitch gain) และ the fixed-codebook gain จะทำการควอนไตซ์แบบเวกเตอร์ 7 บิต การค้นหา gain codebook ถูกทำให้ mean-squared weighted ผิดพลาดน้อยที่สุด ด้วยคำพูดเดิมและคำพูดที่นำมาสร้างใหม่

3.3.3.1.10 การอัพเดทหน่วยความจำ (Memory update)

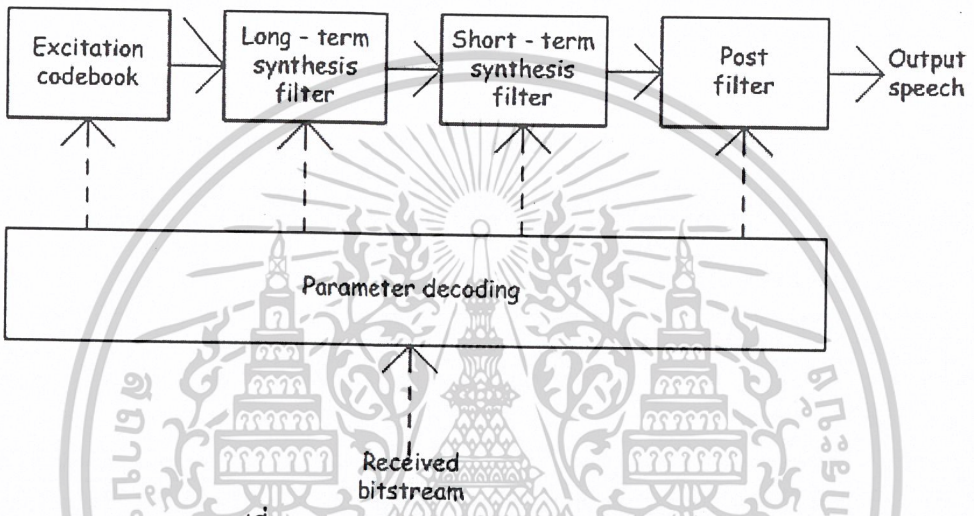
หน้าที่สุดท้ายของเฟรมย่อยที่ i ก่อนที่จะทำเฟรมย่อยต่อไป คือ การอัพเดทหน่วยความจำของ synthesis filter $\widetilde{A}_i(z)$ และ formant perceptual weighting filter $W_i(z)$ และ harmonic noise shaping filter $P_i(z)$

ก่อนที่จะ Encode ในเฟรมย่อยต่อไปจะต้องทำการอัพเดทหน่วยความจำได้แก่ค่า synthesis filter, formant perceptual weighting filter และ harmonic noise shaping filter สำหรับใช้ในเฟรมย่อยถัดไปที่จะใช้ในการคำนวณ zero input response

Parameters	No. of Bits
LSP	18
Pitch Prediction Filter	14
Codebook Indices	34
Gains	14
Total	80

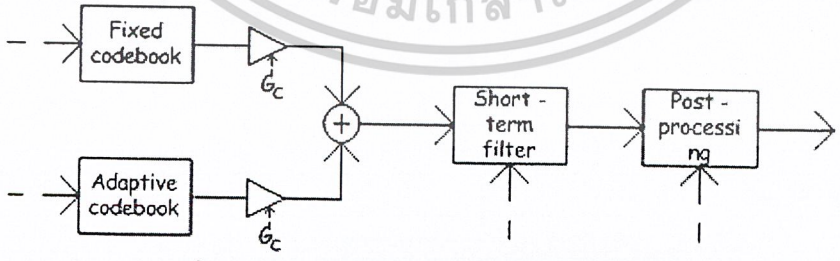
ตารางที่ 3.10 แสดงการจัดสรรบิตของ CS-ACELP ใน 1 เฟรม

3.3.3.2 การถอดรหัสเสียงด้วยวิธี CS-ACELP



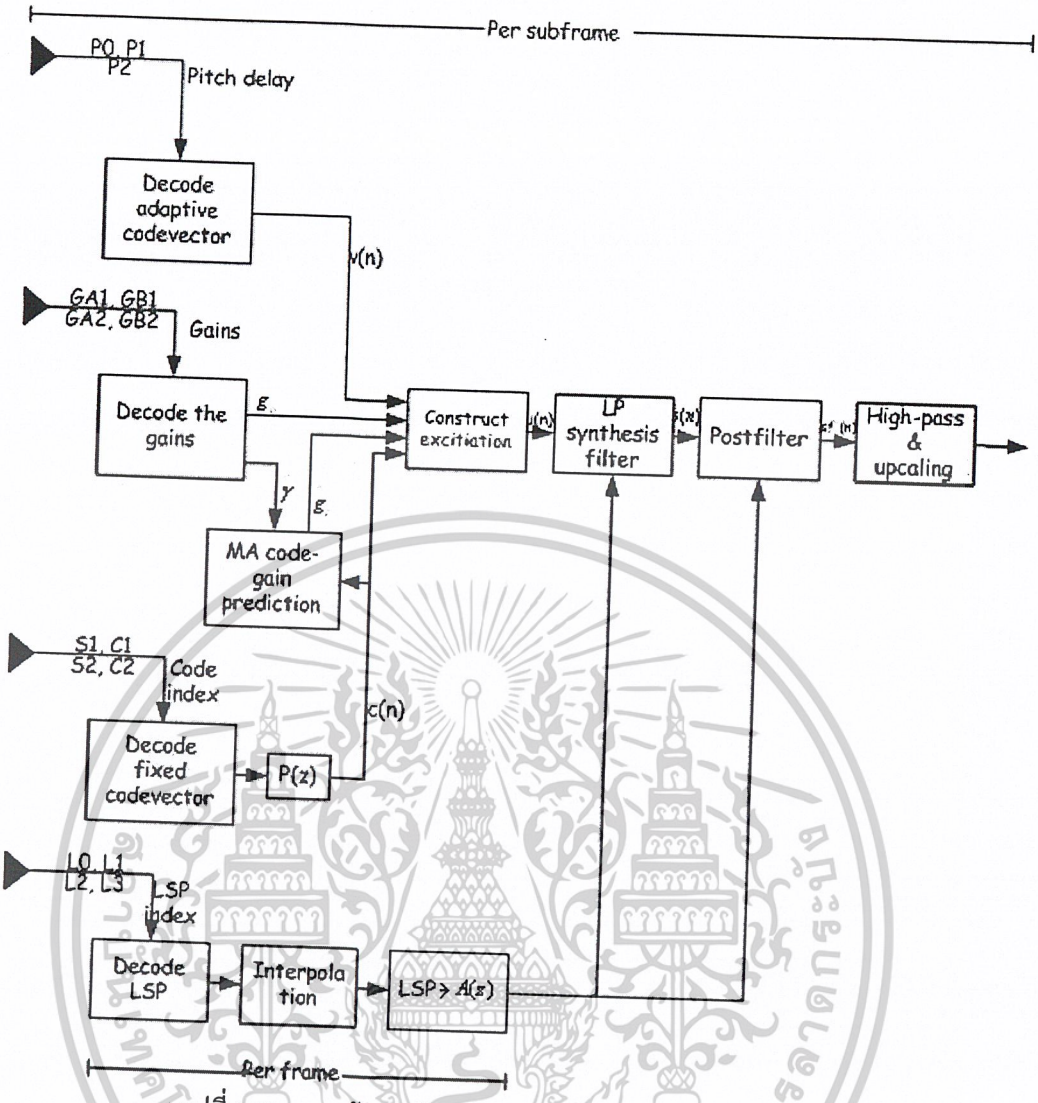
รูปที่ 3.16 แสดงผังของหลักการถอดรหัสสัญญาณ

หลักการของการถอดรหัสสัญญาณนั้น คือ แยกออกมาจากกระแสข้อมูล (ประกอบด้วยพารามิเตอร์ของ LP coefficients, adaptive-codebook vector, fixed-codebook vector และ gains) ซึ่งนำพารามิเตอร์มาทำงานตามผังการทำงาน



รูปที่ 3.17 แสดงผังการทำงานของ CS-ACELP Decoder

1. สัญญาณกระตุ้นจะถูกสร้างขึ้นโดยบวกขนาด adaptive และ fixed codebook vector โดยขึ้นอยู่กับ gain
2. สัญญาณเสียงจะถูกสร้างขึ้นใหม่โดยการกรองสัญญาณกระตุ้นผ่าน LP synthesis filter
3. เสียงที่ถูกสร้างขึ้นมาใหม่นั้นจะส่งผ่านไปยัง Post-processing stage ซึ่งประกอบด้วย post filter ซึ่งมีพื้นฐานมาจาก short-term และ long-term filter และตามด้วยการทำ high pass filter และ scale ขนาดของสัญญาณ



รูปที่ 3.18 แสดงผังการทำงานของ CS-ACELP Decoder

3.3.3.2.1 Post-processing

Post-processing ประกอบด้วย 3 ฟังก์ชัน คือ adaptive postfiltering, high-pass filtering และsignal upscaling โดย adaptive postfiltering จะประกอบด้วย ฟิลเตอร์ 3 ตัว ต่อกัน มี long-term postfilter, short-term postfilter และ tilt compensation filter ตามด้วยขั้นตอน adaptive gain control ซึ่ง postfilter coefficients จะทำการ อัพเดตทุกๆ 5 ms/subframe

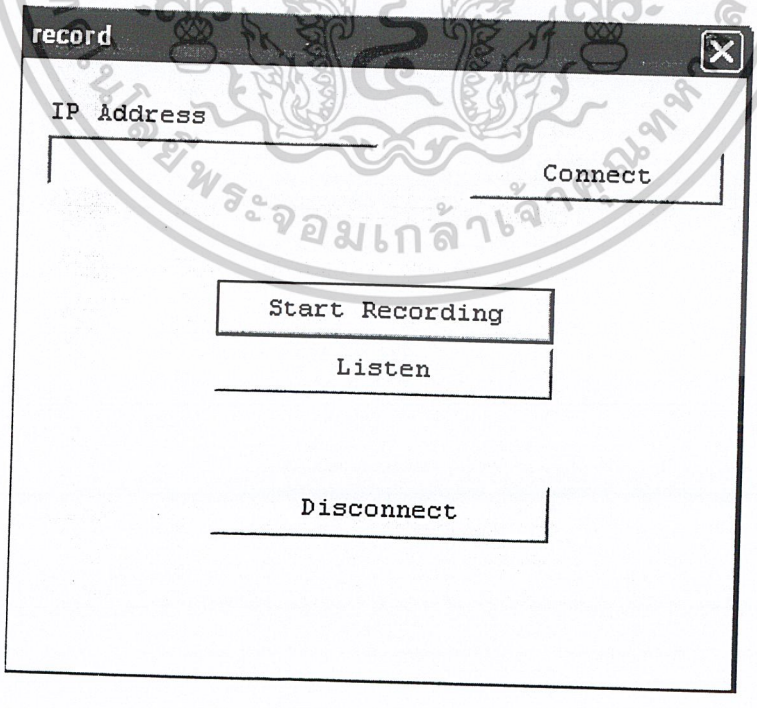
กระบวนการของ postfilter คือ สร้างคำพูด speech $\hat{s}(n)$ ใหม่ หรือ inverse filter เพื่อผลิต residual $\hat{r}(n)$ สัญญาณนี้จะนำมาคำนวณ delay T และ gain g_t ของlong-term postfilter สัญญาณ $\hat{r}(n)$ จะกรองผ่าน long-term filter และ สุดท้าย สัญญาณเอาท์พุตของ synthesis filter จะผ่าน tilt compensation filter เพื่อสร้างคำพูดspeech $sf(n)$ และ adaptive gain control จะปรับปรุง $sf(n)$ ให้สอดคล้องกับ $s(n)$ ผลที่ได้จะผ่าน HP filter และ scaled เพื่อให้ได้สัญญาณเอาท์พุตของการถอดรหัส

Symbol	Description	Bits
L0	Switched MA predictor of LSP quantizer	1
L1	First stage vector of quantizer	7
L2	Second stage lower vector of LSP quantizer	5
L3	Second stage higher vector of LSP quantizer	5
P1	Pitch delay first subframe	8
P0	Parity bit for pitch delay	1
C1	Fixed codebook first subframe	13
S1	Signs of fixed-codebook pulses 1st subframe	4
GA1	Gain codebook (stage 1) 1st subframe	3
GB1	Gain codebook (stage 2) 1st subframe	4
P2	Pitch delay second subframe	5
C2	Fixed codebook 2nd subframe	13
S2	Signs of fixed-codebook pulses 2nd subframe	4
GA2	Gain codebook (stage 1) 2nd subframe	3
GB2	Gain codebook (stage 2) 2nd subframe	4

ตารางที่ 3.11แสดงการส่งค่าพารามิเตอร์ในกระแสข้อมูล

3.4 การออกแบบส่วนติดต่อกับผู้ใช้ (User - Interface) และการใช้งานโปรแกรม

การรับส่งข้อมูลเสียงผ่านเครือข่ายอินเตอร์เน็ต คอมพิวเตอร์ทั้ง 2 เครื่องจะมีส่วนติดต่อกับผู้ใช้ที่เหมือนกัน ดังรูปที่ 3.19



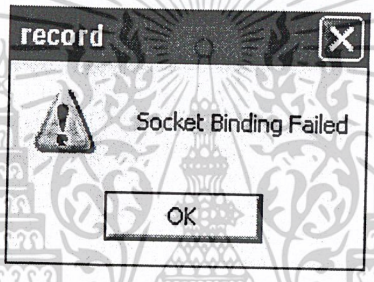
รูปที่ 3.19 แสดงส่วนติดต่อกับผู้ใช้

การใช้งานโปรแกรม

ผู้ใช้จะต้องใส่หมายเลขไอพีแอดเดรสปลายทางที่ต้องการติดต่อก่อน โดยการใส่หมายเลขไอพีแอดเดรสปลายทางนั้นมีรูปแบบมาตรฐานดังนี้ xxx.xxx.xxx.xxx กล่าวคือ เลขฐานสิบ 0-255 ซึ่งก็คือ 8 บิต ในเลขฐานสอง ซึ่งแต่ละ 8 บิต จะขั้นด้วยจุด (.) ซึ่งรวมกันได้ทั้งหมด 32 บิต ตามมาตรฐานที่กำหนด

สำหรับรายละเอียดของปุ่มต่างๆมีดังนี้

ปุ่ม Connect จะเป็นการเริ่มต้นการติดต่อระหว่างเครื่องคอมพิวเตอร์ โดยจะนำเอาหมายเลขไอพีแอดเดรสที่ผู้ใช้ใส่ในช่องไปเริ่มทำการติดต่อโดยเมื่อทำสำเร็จผู้ใช้สามารถใช้งานได้ แต่ถ้าหมายเลขไอพีแอดเดรส นั้นไม่มีหรือไม่ได้เปิดโปรแกรมรออยู่จะแสดงหน้าจอผิดพลาดในการติดต่อ แสดงดังรูปที่ 3.20 เมื่อการทำการติดต่อสำเร็จ โดยการทำงานจะทำเวลาที่ต้องการส่งเสียงไปยังอีกฝ่าย ซึ่งมันจะเริ่มอัดเสียงแล้วส่งออกไป

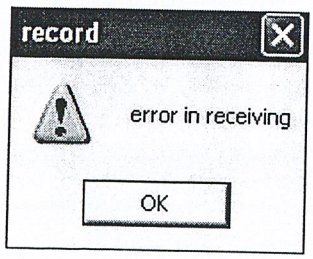


รูปที่ 3.20 แสดงความผิดพลาดในการติดต่อ

ปุ่ม Start Recording จะเป็นการเริ่มต้นการสนทนาโดยผู้ใช้จะกดปุ่มนี้เมื่อการติดต่อทำสำเร็จแล้ว

ปุ่ม Listen ผู้ใช้จะกดปุ่มนี้เมื่อการติดต่อทำสำเร็จแล้ว โดยปุ่มนี้มีไว้สำหรับการฟังเสียงที่ส่งมาจากเครื่องคอมพิวเตอร์อีกเครื่องหนึ่ง

ปุ่ม Disconnect ใช้สำหรับยกเลิกการติดต่อ หรือเมื่อมีหน้าจอแสดงความผิดพลาดในการรับสัญญาณเสียงจากคอมพิวเตอร์อีกเครื่องหนึ่ง แสดงดังรูปที่ 3.21 โดยจะต้องกดปุ่ม OK ที่หน้าต่างของโปรแกรมที่แสดงก่อน จากนั้นจะกดปุ่ม Disconnect เพื่อยกเลิกการติดต่อ (การยกเลิกการติดต่อสามารถกดปุ่มปิดหน้าต่างของโปรแกรมแทนได้) แล้วทำการติดต่อใหม่



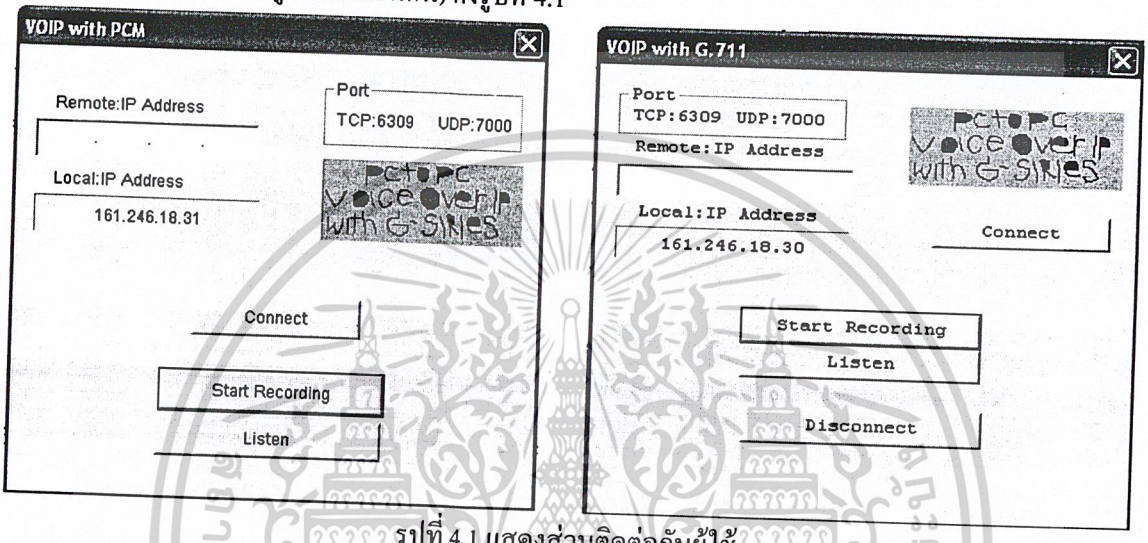
รูปที่ 3.21 แสดงความผิดพลาดในการรับสัญญาณเสียงจากคอมพิวเตอร์อีกเครื่องหนึ่ง

บทที่ 4
การทดลองและผลการทดลอง

4.1 การทดลองในส่วนการติดต่อระหว่างเครื่องคอมพิวเตอร์

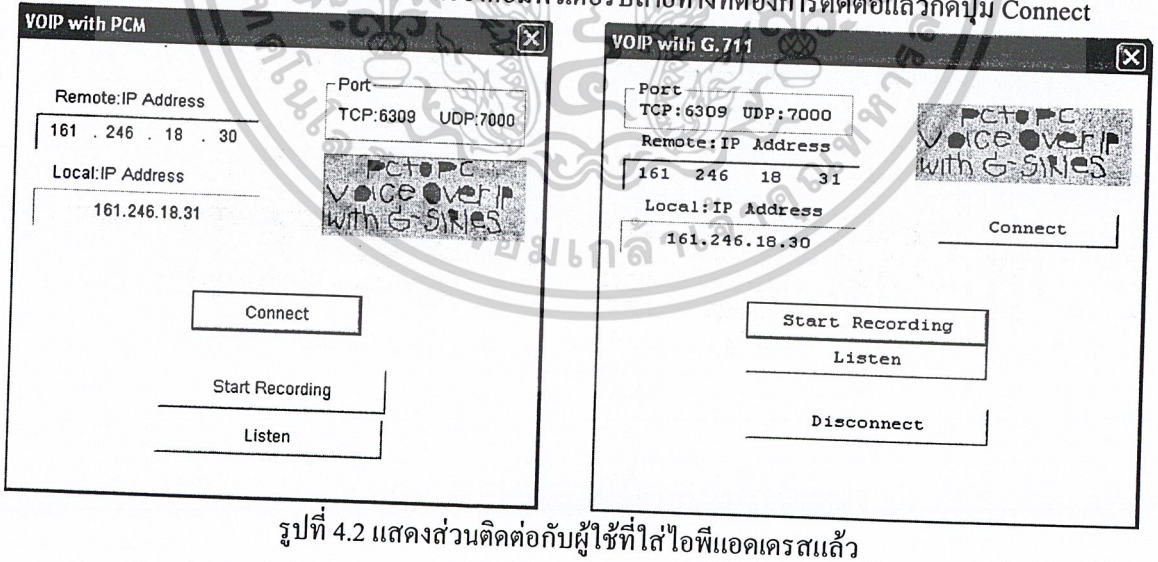
4.1.1 วิธีการทดลอง

1. เปิดโปรแกรมสำหรับทำการติดต่อในเครื่องคอมพิวเตอร์ทั้ง 2 เครื่องที่ต้องการติดต่อกัน (ทั้ง 2 เครื่องจะมีส่วนติดต่อกับผู้ใช้ที่เหมือนกัน) ดังรูปที่ 4.1



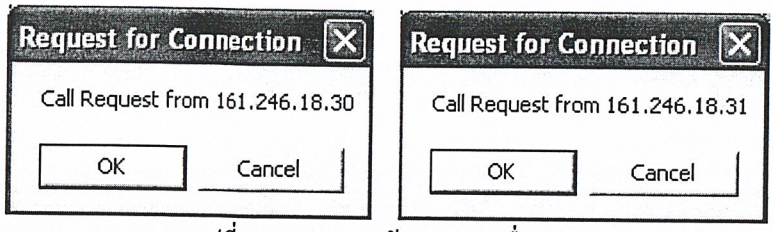
รูปที่ 4.1 แสดงส่วนติดต่อกับผู้ใช้

2. ทำการใส่ไอพีแอดเดรสของเครื่องคอมพิวเตอร์ปลายทางที่ต้องการติดต่อแล้วกดปุ่ม Connect



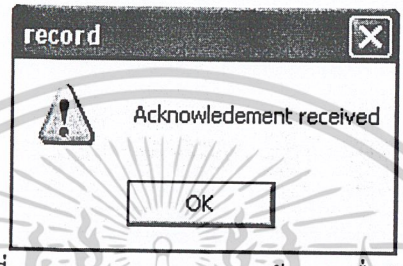
รูปที่ 4.2 แสดงส่วนติดต่อกับผู้ใช้ที่ใส่ไอพีแอดเดรสแล้ว

1. ทางด้านปลายทางส่วนติดต่อกับผู้ใช้จะปรากฏหน้าต่างแสดงการร้องขอการเชื่อมต่อตามรูปที่ 4.3 จากนั้นทำการกดปุ่ม OK เมื่อต้องการติดต่อ ถ้าไม่ต้องการติดต่อกดปุ่ม Cancel



รูปที่ 4.3 แสดงการร้องขอการเชื่อมต่อ

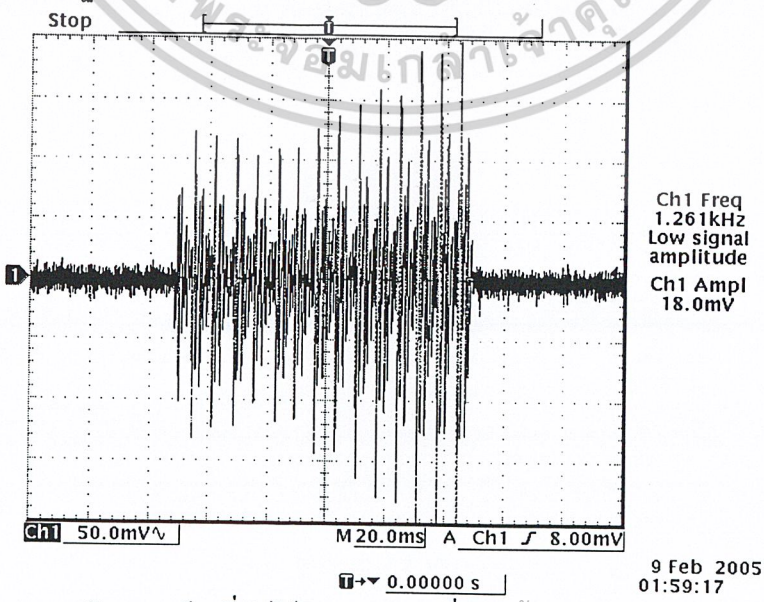
2. ทางด้านต้นทางจะปรากฏหน้าต่างแสดงการยอมรับการร้องขอเพื่อติดต่อ เมื่อผู้ใช้ทำการกดปุ่ม OK ก็จะเสร็จสิ้นในขั้นตอนการเชื่อมต่อ



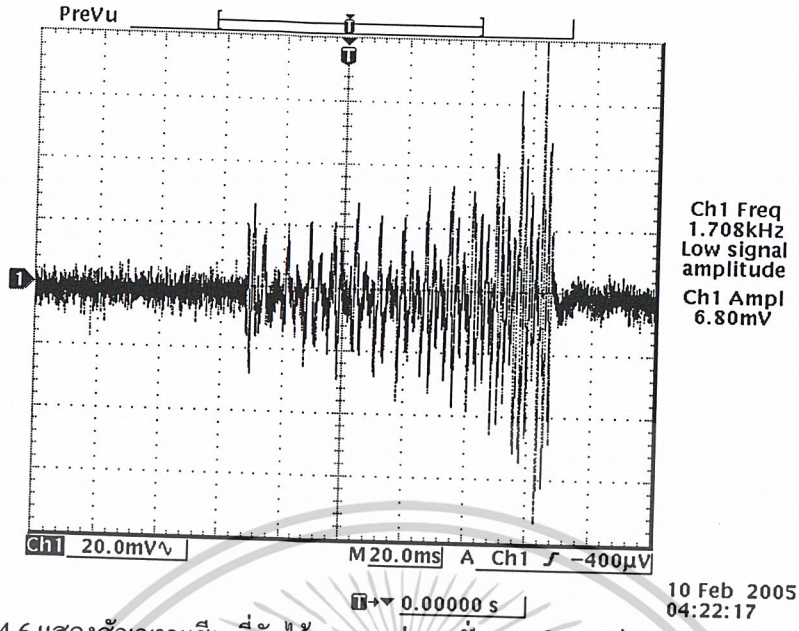
รูปที่ 4.4 แสดงการยอมรับการร้องขอเพื่อติดต่อ

3. จากนั้นเครื่องคอมพิวเตอร์ทั้ง 2 เครื่องที่สามารถติดต่อกันได้แล้ว ผู้ใช้จะต้องทำการกดปุ่ม Start Recording เพื่อให้โปรแกรมสามารถรับเสียงจากผู้ใช้ และกดปุ่ม Listen เพื่อให้สามารถฟังเสียงที่ส่งมาได้จากอีกด้านได้
4. ทำการทดลองสนทนาโต้ตอบกัน
5. เมื่อต้องการยกเลิกการติดต่อกดปุ่ม Disconnect หรือ กดปุ่มปิดหน้าต่างของโปรแกรม

4.1.2 ผลการทดลองในส่วนการติดต่อระหว่างเครื่องคอมพิวเตอร์โดยสัญญาณเสียงที่ส่งเป็นการเข้ารหัสแบบพีซีเอ็มโดยไม่ได้อิงมาตรฐาน

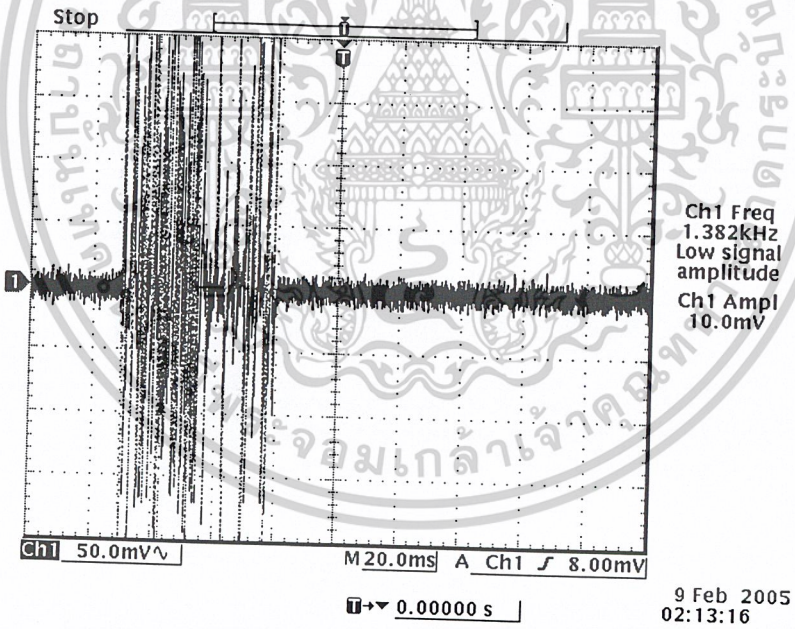


รูปที่ 4.5 แสดงสัญญาณเสียงที่วัดได้จากการส่งจากฝั่งคอมพิวเตอร์ต้นทางมายังปลายทาง

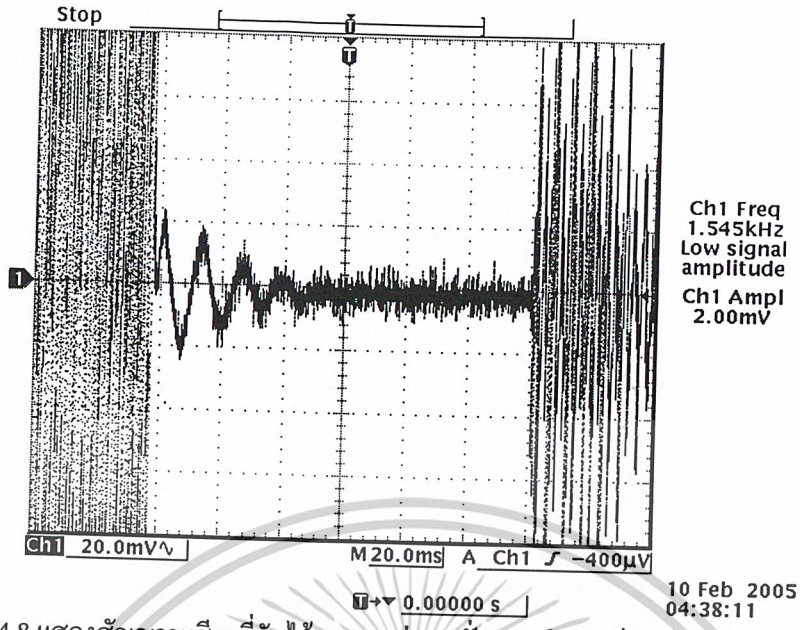


รูปที่ 4.6 แสดงสัญญาณเสียงที่วัดได้จากการส่งจากฝั่งคอมพิวเตอร์ปลายทางมายังต้นทาง

4.1.3 ผลการทดลองในส่วนการติดต่อระหว่างเครื่องคอมพิวเตอร์โดยสัญญาณเสียงที่ส่งเป็นการเข้ารหัสแบบพีซีเอ็มโดยอิงมาตรฐาน G.711 แบบ A-Law

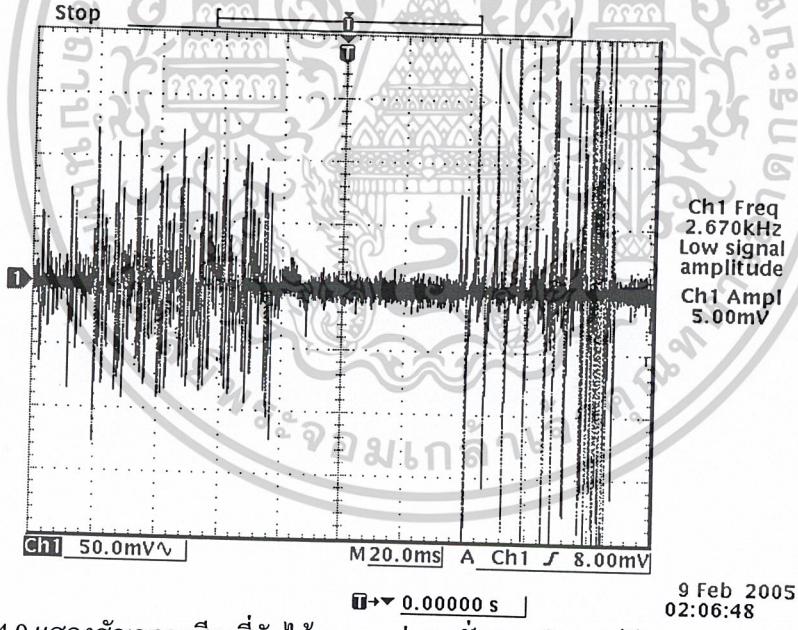


รูปที่ 4.7 แสดงสัญญาณเสียงที่วัดได้จากการส่งจากฝั่งคอมพิวเตอร์ต้นทางมายังปลายทาง

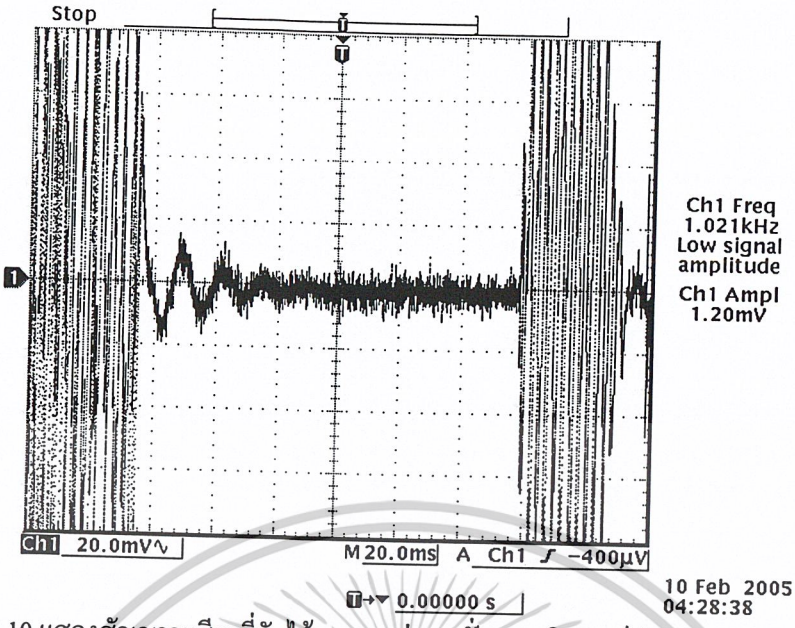


รูปที่ 4.8 แสดงสัญญาณเสียงที่วัดได้จากการส่งจากฝั่งคอมพิวเตอร์ปลายทางมายังต้นทาง

4.1.4 ผลการทดลองในส่วนการติดต่อระหว่างเครื่องคอมพิวเตอร์โดยสัญญาณเสียงที่ส่งเป็นการเข้ารหัสแบบพีซีเอ็มโดยอิงมาตรฐาน G.711 แบบ μ-Law

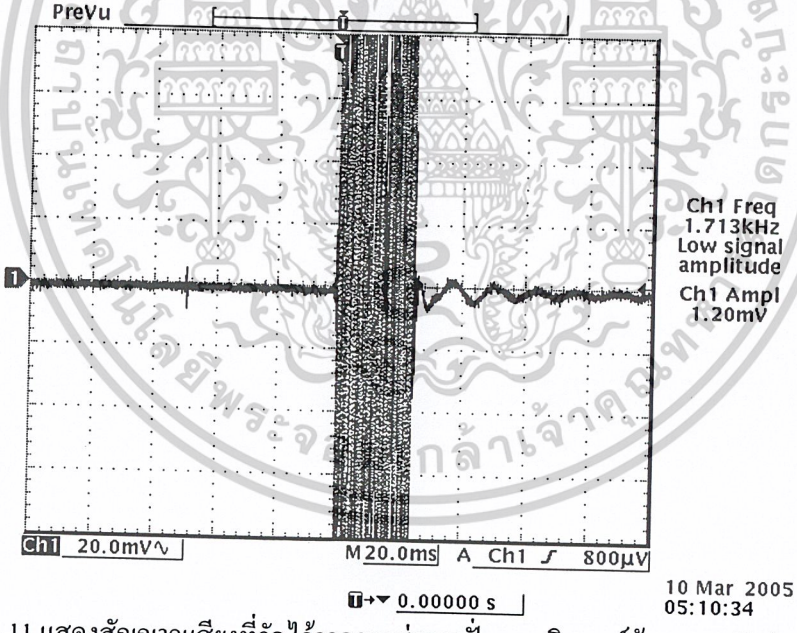


รูปที่ 4.9 แสดงสัญญาณเสียงที่วัดได้จากการส่งจากฝั่งคอมพิวเตอร์ต้นทางมายังปลายทาง

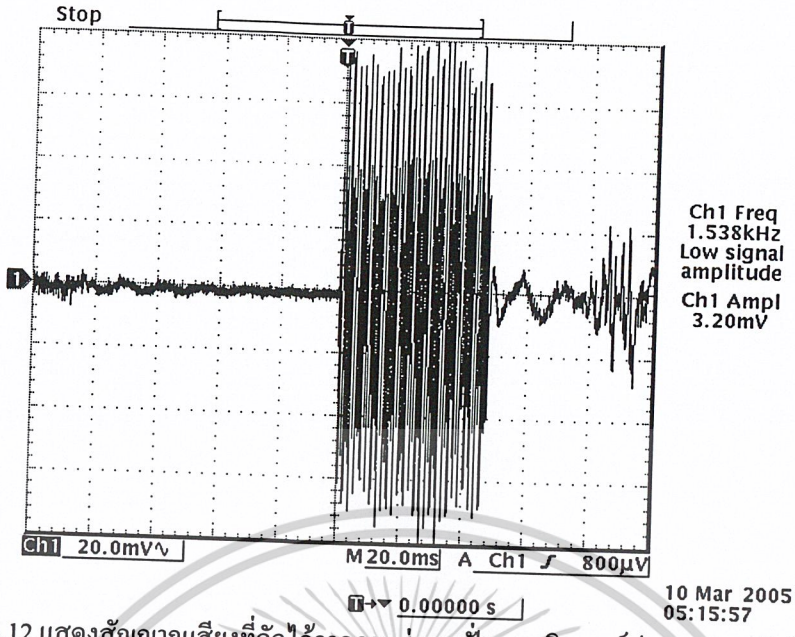


รูปที่ 4.10 แสดงสัญญาณเสียงที่วัดได้จากการส่งจากฝั่งคอมพิวเตอร์ปลายทางมายังต้นทาง

4.1.5 ผลการทดลองในส่วนการติดต่อระหว่างเครื่องคอมพิวเตอร์โดยสัญญาณเสียงที่ส่งเป็นการเข้ารหัสด้วยมาตรฐาน G.726



รูปที่ 4.11 แสดงสัญญาณเสียงที่วัดได้จากการส่งจากฝั่งคอมพิวเตอร์ต้นทางมายังปลายทาง



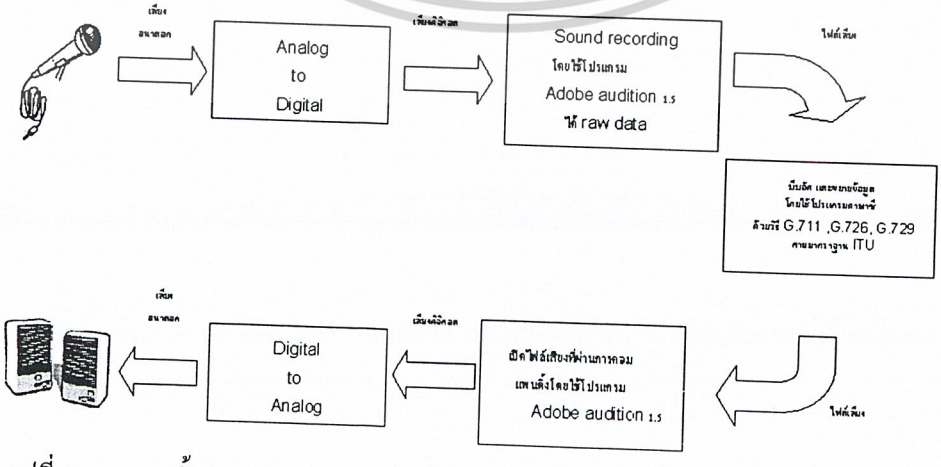
รูปที่ 4.12 แสดงสัญญาณเสียงที่วัดได้จากการส่งจากฝั่งคอมพิวเตอร์ปลายทางมายังต้นทาง

จากผลการทดลอง เกิดจากการส่งสัญญาณเสียงพูดคำว่า “สวัสดี” โดยช่วงของสัญญาณที่มีแอมปลิจูดมากกว่า คือ ช่วงที่มีสัญญาณเสียงพูด ในช่วงของสัญญาณที่มีแอมปลิจูดน้อยกว่า คือ ช่วงของการเงียบ

จากการทดลองพบว่าสามารถทำการเชื่อมต่อ และสนทนาได้ แต่มีสัญญาณรบกวนขณะทำการทดลองมาก สังเกตจากรูปในส่วนของสัญญาณที่เป็นช่วงเงียบจะพบว่า แอมปลิจูดไม่เท่ากับ 0 นั่นคือมีสัญญาณรบกวน และมีการหน่วงของสัญญาณที่พอรับได้ รวมทั้งได้ยินเสียงตนเองสะท้อนกลับมา แต่เป็นเสียงที่เบาและมาช้ากว่าปกติ

4.2 การทดลองและผลการทดลองในส่วนบีบอัดสัญญาณเสียง

แสดงขั้นตอนการทำงานในส่วนของการรับเสียงเพื่อเข้ารหัสและถอดรหัสเสียง ผลการทดลองจากโปรแกรมภาษาซี โดยจะแสดงการทำงานของการจัดการเกี่ยวกับเสียง ได้ ดังนี้

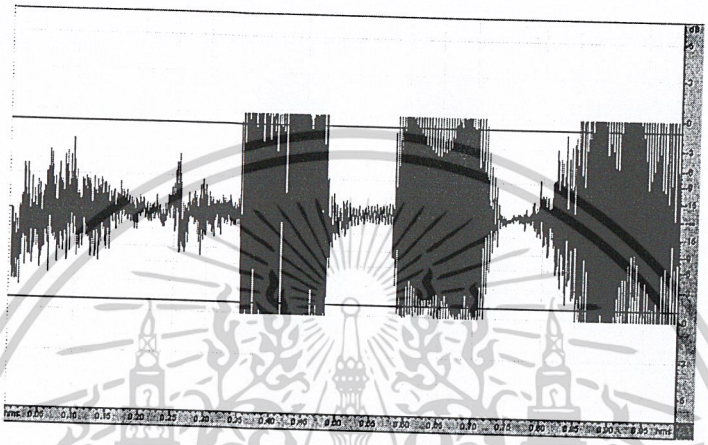


รูปที่ 4.13 แสดงขั้นตอนการทำงานในส่วนของการรับเสียงเพื่อเข้ารหัสและถอดรหัสเสียง

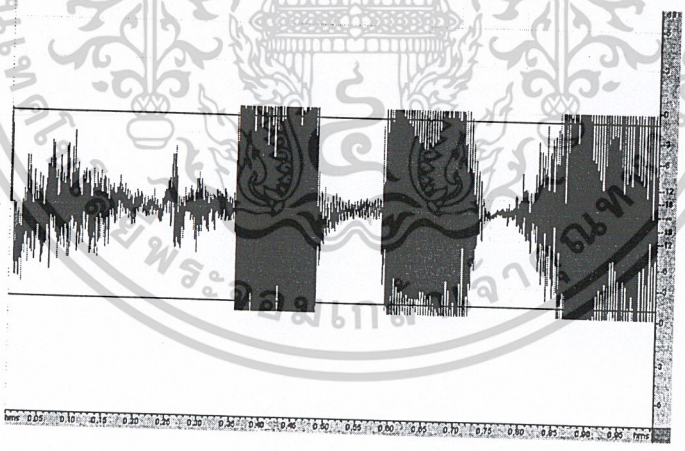
4.2.1 การเข้ารหัสด้วยมาตรฐาน G.711

ได้มีการส่งข้อมูลเสียงออกไปโดยอาศัยหลักการพัลส์โค้ดมอดูเลชั่น ตามมาตรฐาน ITU G.711 แบบ A-Law และ μ -Law ซึ่งได้ทำการทดลองการส่งข้อมูลเสียงออกไป ได้นำการทดลองกับข้อมูลที่เป็นไฟล์เสียงที่ได้จากการแซมปลิ้ง 8000 แซมเปิ้ลต่อวินาที และควอนไตซ์ที่ 16 บิต

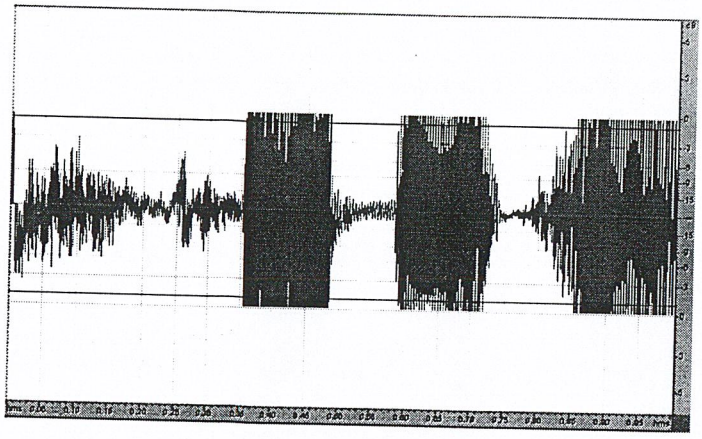
โดยทำการทดลองด้วยการส่งสัญญาณเสียงพูดออกไปผ่านทางไมโครโฟน และวัดสัญญาณ ข้อมูลดิบทั้งด้านส่ง และทางด้านรับมาเป็นกราฟ โดยใช้โปรแกรม Adobe Audition1.5 ก็จะได้สัญญาณ ซึ่งมีผลการทดลองดังนี้



รูปที่ 4.14 แสดงสัญญาณอินพุทที่เป็นสัญญาณเสียงพูด คำว่า “สวัสดีค่ะ” โดยแสดงในแกนเวลา



รูปที่ 4.15 แสดงสัญญาณเอาท์พุท ที่ผ่านการเข้ารหัสและถอดรหัส ตามมาตรฐาน ITU ด้วย G.711 แบบ A-Law ที่เป็นสัญญาณเสียงพูด คำว่า “สวัสดีค่ะ” โดยแสดงในแกนเวลา



รูปที่ 4.16 แสดงสัญญาณเอาท์พุท ที่ผ่านการเข้ารหัสและถอดรหัส ตามมาตรฐาน ITU ด้วย G.711 แบบ μ-Law ที่เป็นสัญญาณเสียงพูด คำว่า "สวัสดีค่ะ" โดยแสดงในแกนเวลา

ผลการทดลองที่ออกมาจากกราฟจะเห็นว่า เมื่อผ่านการเข้าและถอดรหัสแบบ A-Law จะได้สัญญาณเสียงพูดที่มีความใกล้เคียงกับสัญญาณเสียงพูดที่เป็นอินพุทมากกว่า การเข้าและถอดรหัสแบบ μ-Law แต่จากการฟังสัญญาณเสียงที่เป็นเอาท์พุททั้งสองแบบแล้วสามารถฟังได้ชัดเจนและรู้เรื่องเมื่อเทียบจากการฟังจากสัญญาณเสียงที่เป็นอินพุทแล้ว โดยนำมาแสดงเป็นค่าที่ละเอียด

วิธีการเข้ารหัส	จำนวนบิตที่ผิดพลาด	จำนวนบิตที่เข้ารหัส	จำนวนไบต์ที่เข้ารหัส	เปอร์เซ็นต์ความผิดพลาด
G.711 แบบ A-Law	30895	128464	16058	24.05 %
G.711 แบบ μ-Law	33276	128464	16058	25.9 %
G.726	45394	128464	16058	35.336 %
G.729	63471	128464	16058	49.4 %

ตารางที่ 4.1 แสดงผลการหาความผิดพลาดของบิตข้อมูลในส่วนการบีบอัด

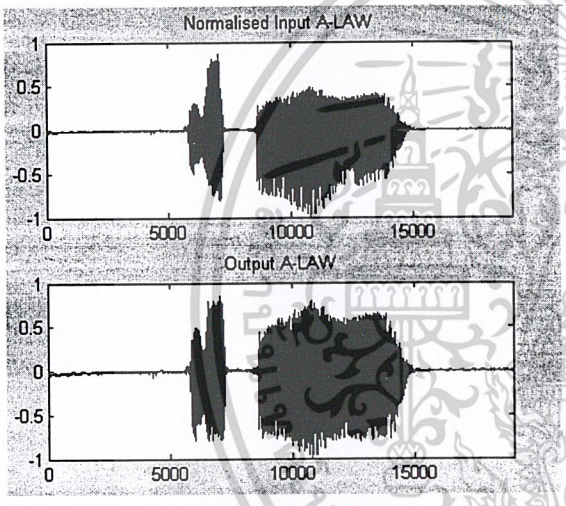
จากตารางจะเห็นว่าเปอร์เซ็นต์ความผิดพลาดของการบีบอัดด้วย G.711 แบบ A-Law และแบบ μ-Law จะมีค่าใกล้เคียงกัน เพราะเป็นการบีบอัดที่ใช้มาตรฐานเดียวกัน ส่วนการบีบอัดด้วยมาตรฐาน G.726 จะมีความผิดพลาดมากกว่า G.711 เพราะมีการลดขนาดของข้อมูลลง 4 เท่า แต่ G.7111 ลดลงเพียง 2 เท่า ส่วนการบีบอัดด้วย G.729 จะมีความผิดพลาดมากที่สุดเพราะใช้อัลกอลิทึมในการบีบอัดต่างจากมาตรฐาน G.711 และ G.726

จากนั้นจะนำโปรแกรมแมทเลปมาทำการซิมมูเลชั่นเพื่อดูสัญญาณ โดยการนำไฟล์เสียงมาผ่านโปรแกรมแมทแลบด้วยสมการของวิธีเข้ารหัสแบบ μ-Law และ A-Law และผ่านการควอนไทซ์และนอมอลไลซ์ค่าออกมา ผลที่ได้จะนำมาคำนวณค่า Signal to noise Ratio ผลการทดลองจะแสดงเป็นค่าอินพุตและเอาท์พุตผ่านการเข้ารหัสและถอดรหัสที่นอมอลไลซ์แล้ว

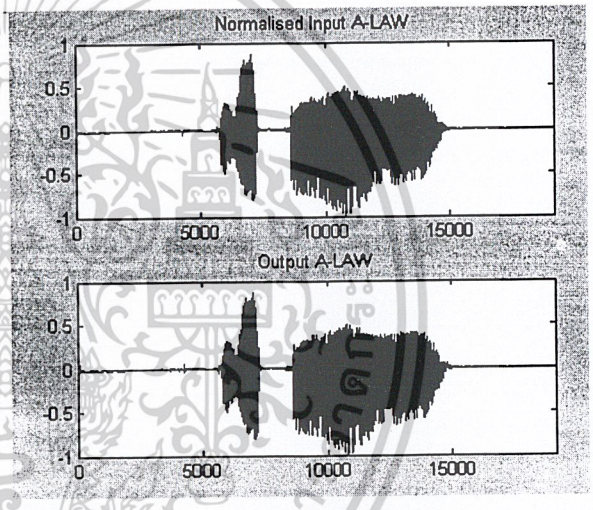
เมื่อปรับค่าพารามิเตอร์ A โดยทดลองด้วยไฟล์เสียงที่พูดคำว่า “สวัสดี”

A	SNR(dB)
2	6.165
10	11.27
87.6	25.69
100	27.01

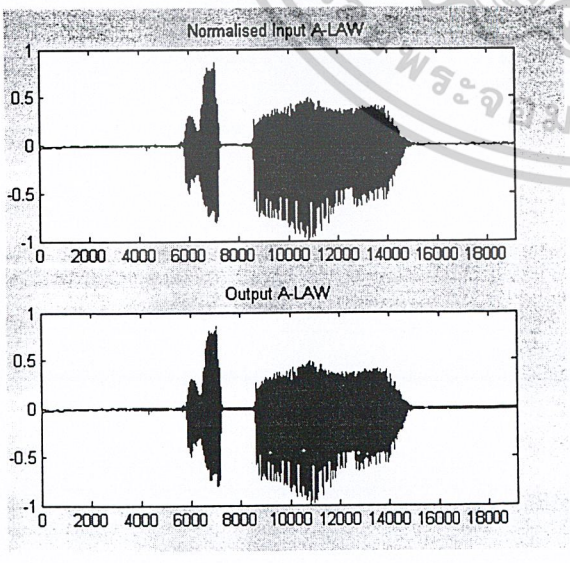
ตารางที่ 4.2 แสดงการปรับค่าพารามิเตอร์ A และค่าเอสเอ็นอาร์



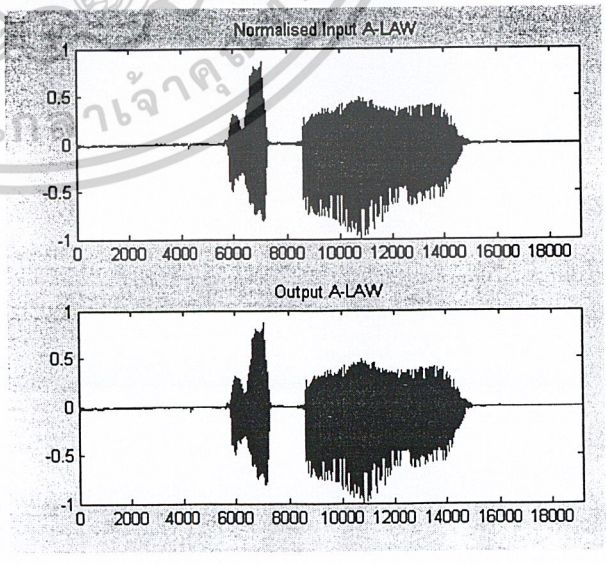
A=2



A=10



A= 87.6



A=100

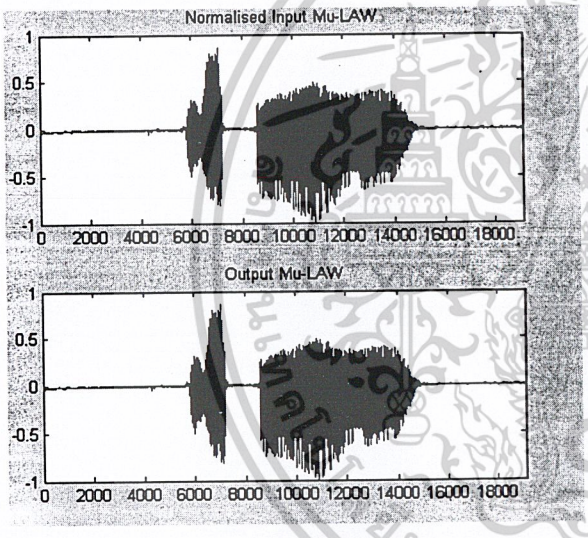
รูปที่ 4.17 แสดงผลการซิมมูเลชั่นด้วยโปรแกรมแมทเลป

จากค่าเอสเอ็นอาร์จะเห็นว่า เมื่อปรับค่าพารามิเตอร์ A ให้มีค่าน้อยๆ จนเท่ากับ 1 จะมีค่าเหมือนการจัดระดับการควอนไตซ์แบบเชิงเส้น และเมื่อปรับค่า A ให้มีค่าเพิ่มขึ้นจะมีค่าเหมือนการจัดระดับการควอนไตซ์แบบไม่เชิงเส้น ซึ่งคุณภาพของเสียงที่ดีกว่าจะมีค่าเอสเอ็นอาร์ที่ดีกว่า เพราะเป็นการลดควอนไตซ์น็อยส์

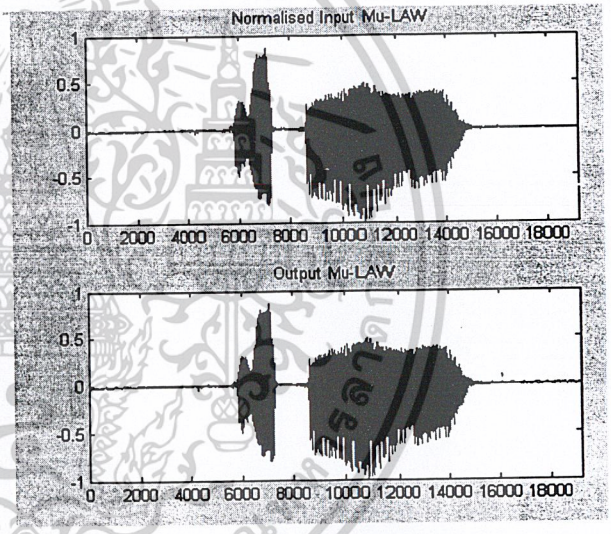
เมื่อปรับค่าพารามิเตอร์ μ โดยทดลองด้วยไฟล์เสียงที่พูดคำว่า “สวัสดี”

μ	SNR(dB)
1	43.95
10	47.71
100	45.50
255	43.91

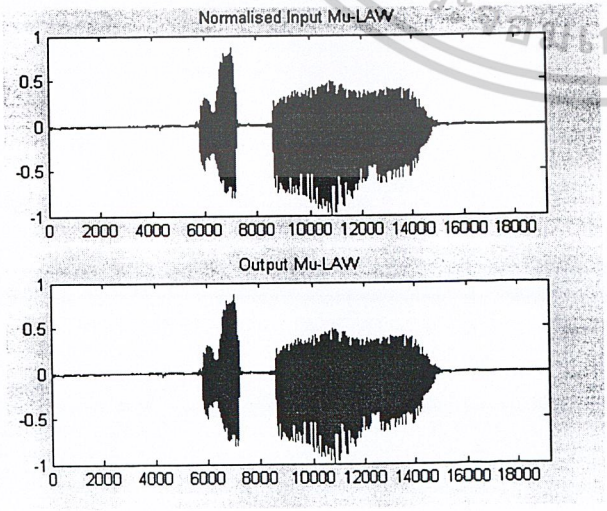
ตารางที่ 4.3 แสดงการปรับค่าพารามิเตอร์ μ และค่าเอสเอ็นอาร์



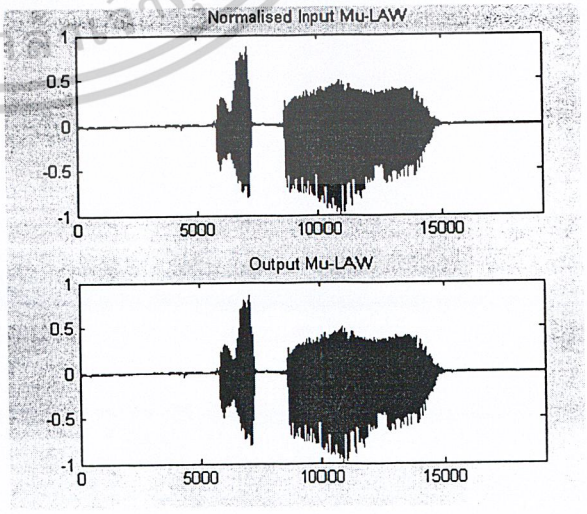
μ =1



μ =10



μ = 100



μ = 255

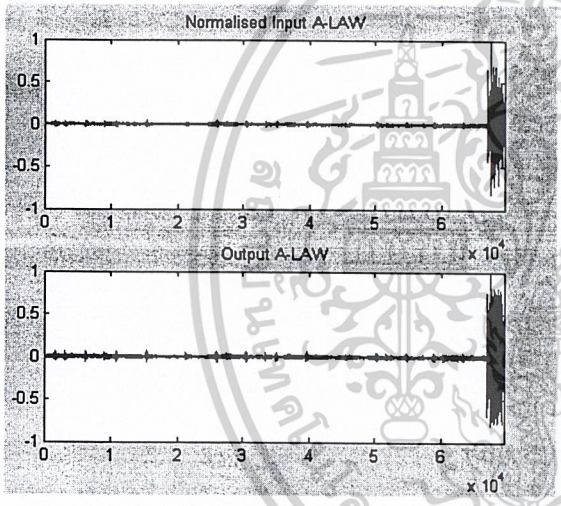
รูปที่ 4.18 แสดงผลการซิมมูเลชั่นด้วยโปรแกรมแมทแลป

จากค่าเอสเอ็นอาร์จะเห็นว่า เมื่อปรับค่าพารามิเตอร์ μ ให้มีค่าน้อยๆ จนเท่ากับ 1 จะมีค่าเหมือนการจัดระดับการควอนไตซ์แบบเชิงเส้น และเมื่อปรับค่า μ ให้มีค่าเพิ่มขึ้น จะมีค่าเหมือนการจัดระดับการควอนไตซ์แบบไม่เชิงเส้น ซึ่งคุณภาพของเสียงที่ดีกว่าจะมีค่าเอสเอ็นอาร์ที่ดีกว่า เพราะเป็นการลดควอนไตซ์น็อยส์

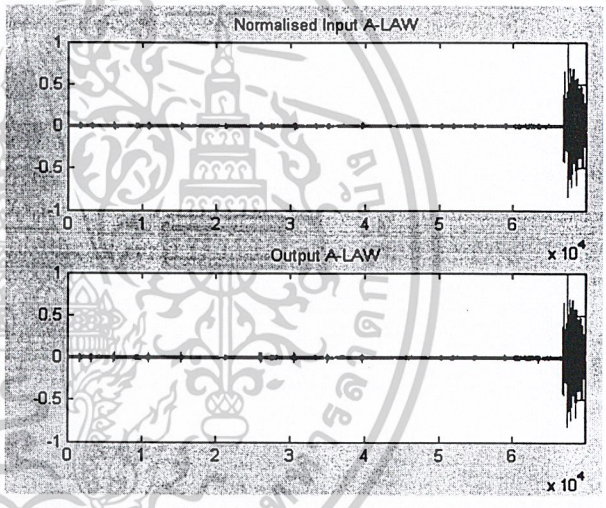
เมื่อทำการทดลองโดยใช้ไฟล์เสียง “MP3” และปรับค่าพารามิเตอร์ A

A	SNR(dB)
2	5.097
10	12.37
87.6	14.26
100	15.62

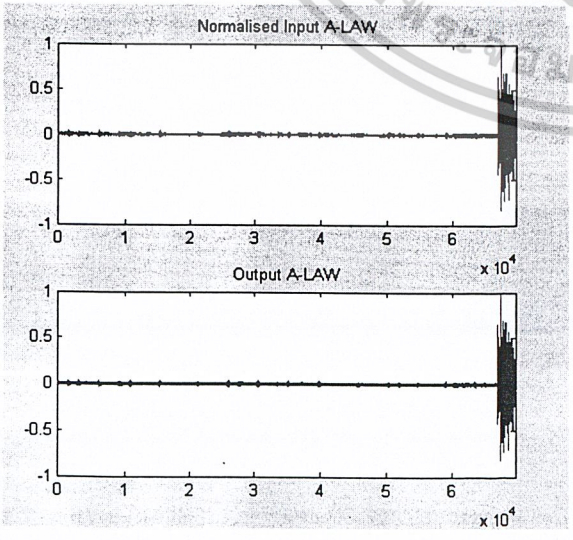
ตารางที่ 4.4 แสดงการปรับค่าพารามิเตอร์ A และค่าเอสเอ็นอาร์



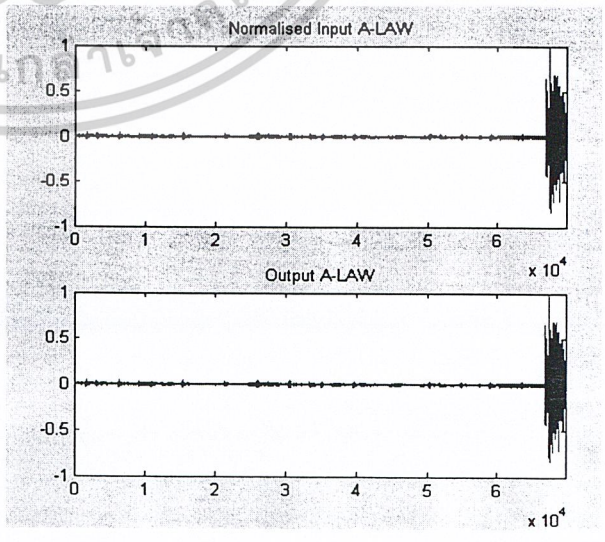
A=2



A=10



A= 87.6



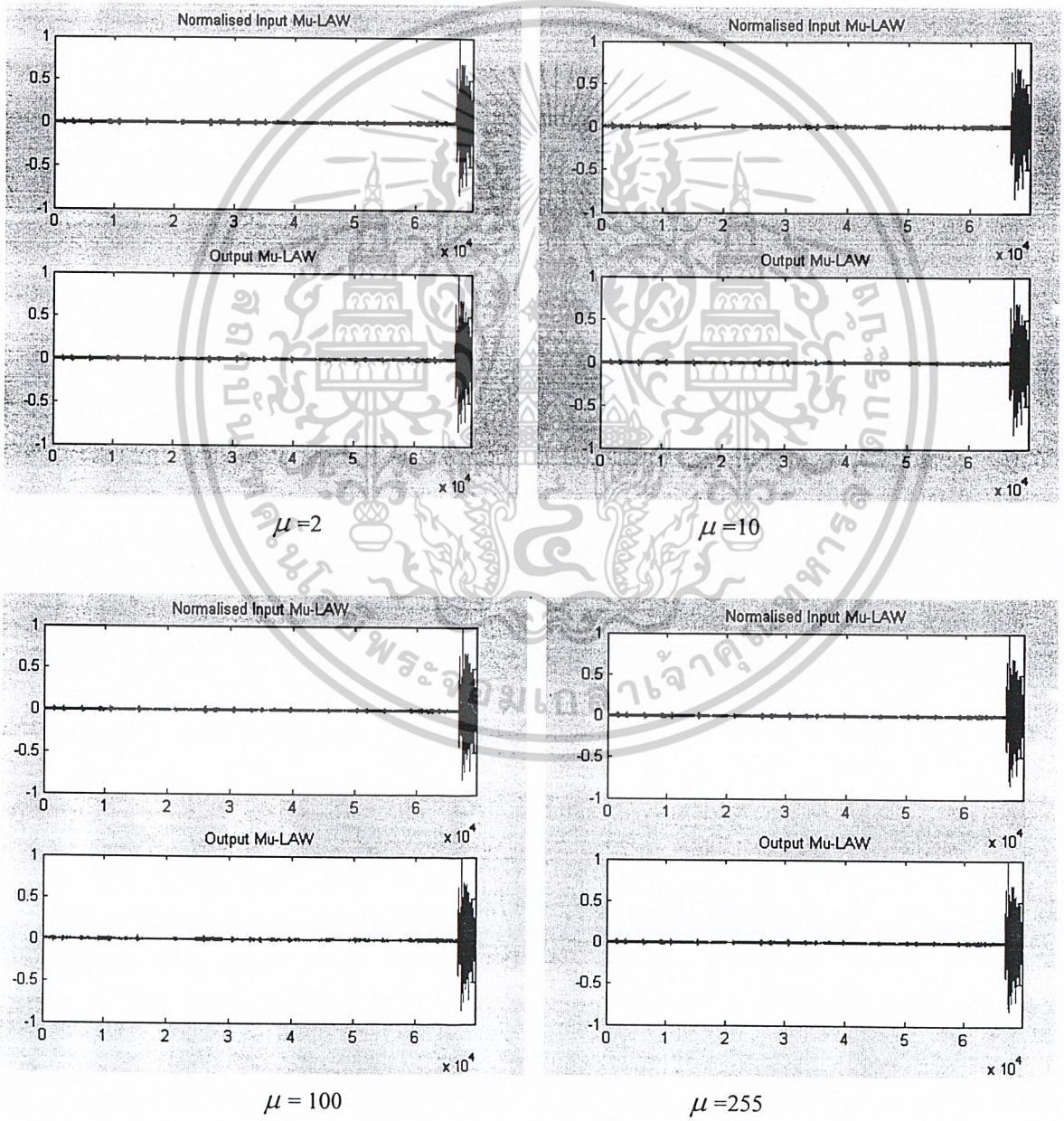
A=100

รูปที่ 4.19 แสดงผลการชิมมูเลชั่นด้วยโปรแกรมแมทเลป

เมื่อทำการทดลองโดยใช้ไฟล์เสียง “MP3” และปรับค่าพารามิเตอร์ μ

μ	SNR(dB)
2	36.95
10	42.71
100	4.93
255	43.81

ตารางที่ 4.5 แสดงการปรับค่าพารามิเตอร์ μ และค่าเอสเอ็นอาร์



μ =2 μ =10

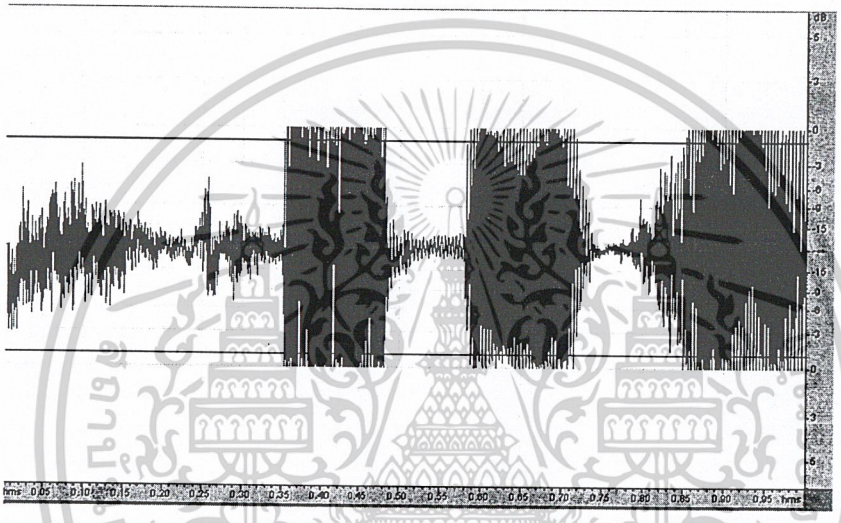
μ = 100 μ =255

รูปที่ 4.20 แสดงผลการซิมมูเลชั่นด้วยโปรแกรมแมทเลป

4.2.2 การเข้ารหัสด้วยมาตรฐาน G.726

ทำการส่งข้อมูลเสียงออกไปโดยอาศัยหลักการอแด็พทีฟดิฟเฟอร์เรนเชียลพัลส์โค้ดมอดูเลชั่น ตามมาตรฐาน ITU G.726 ซึ่งได้ทำการทดลองการส่งข้อมูลเสียงออกไป ได้นำการทดลองกับข้อมูลที่เป็นไฟล เสียงที่ได้จากการแซมปลิ้ง 8000 แซมเปิ้ลต่อวินาที และควอนไตซ์ที่ 16 บิต

โดยทำการทดลองด้วยการส่งสัญญาณเสียงพูดออกไปผ่านทางไมโครโฟน และวัดสัญญาณ ข้อมูลดิบทั้งด้านส่ง และทางด้านรับมาเป็นกราฟ โดยใช้ โปรแกรม Adobe Audition1.5 ซึ่งจะใช้สัญญาณอินพุทที่เป็นสัญญาณเสียงพูด คำว่า "สวัสดีค่ะ" โดยแสดงในแกนเวลาสัญญาณเดียวกับมาตรฐานอื่นด้วยก็จะได้สัญญาณเอาท์พุตซึ่งมีผลการทดลองดังนี้

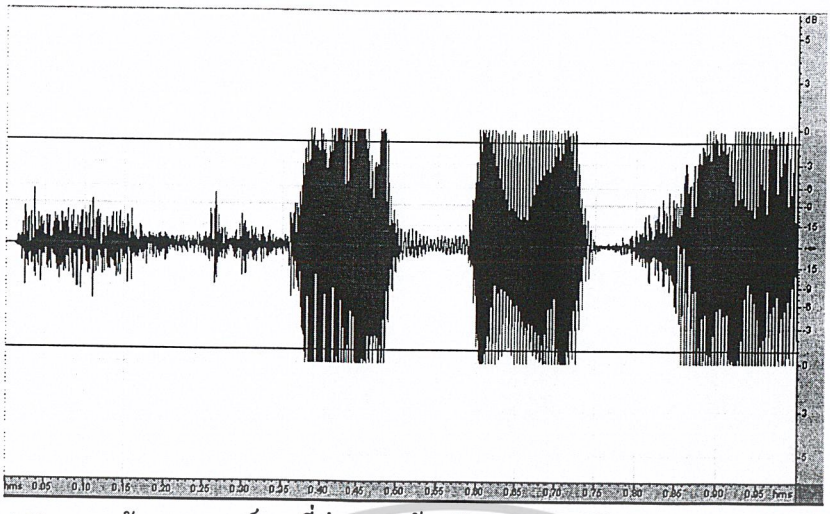


รูปที่ 4.21 แสดงสัญญาณเอาท์พุท ที่ผ่านการเข้ารหัสและถอดรหัส ตามมาตรฐาน ITU ด้วย G.726 ที่เป็นสัญญาณเสียงพูด คำว่า "สวัสดีค่ะ" โดยแสดงในแกนเวลา

4.2.3 การเข้ารหัสด้วยมาตรฐาน G.729

ทำการส่งข้อมูลเสียงออกไปโดยอาศัยหลักการ Conjugate-Structure Algebraic-Code–Excited Linear–Prediction (CS-ACELP) เป็นมาตรฐานของการเข้ารหัสเสียง G.729 ตามมาตรฐาน ITU ซึ่งได้ทำการทดลองการส่งข้อมูลเสียงออกไปได้นำการทดลองกับข้อมูลที่เป็นไฟล์เสียงที่ได้จากการแซมปลิ้ง 8000 แซมเปิ้ลต่อวินาที และควอนไตซ์ที่ 16 บิต

โดยทำการทดลองด้วยการส่งสัญญาณเสียงพูดออกไปผ่านทางไมโครโฟน และวัดสัญญาณ ข้อมูลดิบทั้งด้านส่ง และทางด้านรับมาเป็นกราฟ โดยใช้ โปรแกรม Adobe Audition1.5 ซึ่งจะใช้สัญญาณอินพุทที่เป็นสัญญาณเสียงพูด คำว่า "สวัสดีค่ะ" โดยแสดงในแกนเวลาสัญญาณเดียวกับมาตรฐานอื่นด้วยก็จะได้สัญญาณเอาท์พุตซึ่งมีผลการทดลองดังนี้



รูปที่ 4.22 แสดงสัญญาณเอาท์พุท ที่ผ่านการเข้ารหัสและถอดรหัส ตามมาตรฐาน ITU ด้วย G.729 ที่เป็นสัญญาณเสียงพูด คำว่า "สวัสดีค่ะ" โดยแสดงในแกนเวลา

4.2.4 การทดลองส่วนของการบีบอัดเสียงโดยดูที่ขนาดไฟล์

มาตรฐาน ITU-T		ขนาดไฟล์เสียงอินพุต(KB)	ขนาดไฟล์เสียงที่ผ่านการเข้ารหัส (KB)	ขนาดไฟล์เสียงเอาท์พุต (KB)
G.711	A-Law	15.6	7.84	15.6
	μ -Law	15.6	7.84	15.6
G.726		15.6	3.96	15.6
G.729		15.6	16	15.6

ตารางที่ 4.6 แสดงการเปรียบเทียบขนาดไฟล์ที่ผ่านการบีบอัดด้วยมาตรฐานต่างๆ

จากตาราง จะเห็นว่าการทดลองด้วยวิธีตามมาตรฐานต่างกันจะให้ขนาดไฟล์ที่ผ่านการเข้ารหัสแล้วต่างๆกัน ซึ่งเป็นไปตามแต่ละมาตรฐานที่กำหนดไว้ ซึ่งจากการทดลองจะเห็นว่าสามารถทำการทดลองได้ผลตามที่มาตรฐานกำหนดไว้ และเมื่อนำสัญญาณเสียงที่ผ่านการเข้ารหัสและถอดรหัสและนำมาฟัง เสียงที่ได้มีความแตกต่างจากสัญญาณเสียงอินพุทเพียงเล็กน้อย ซึ่งฟังได้ชัดเจนรู้เรื่อง

4.3 การทดลองในส่วนคุณภาพของระบบโครงข่ายคอมพิวเตอร์

เราจะทำการทดสอบเพื่อหาค่าเวลาหน่วง (Delay Time) ของโครงงานนี้ โดยใช้โปรแกรมดักเพ็กเก็จที่ถูกส่งจากคอมพิวเตอร์เครื่องหนึ่งไปยังอีกเครื่องหนึ่ง และทำการวัดหาค่าเวลาหน่วง ในการทดลองโครงงานนี้ใช้โปรแกรม ethereal ในการดักเพ็กเก็จ)

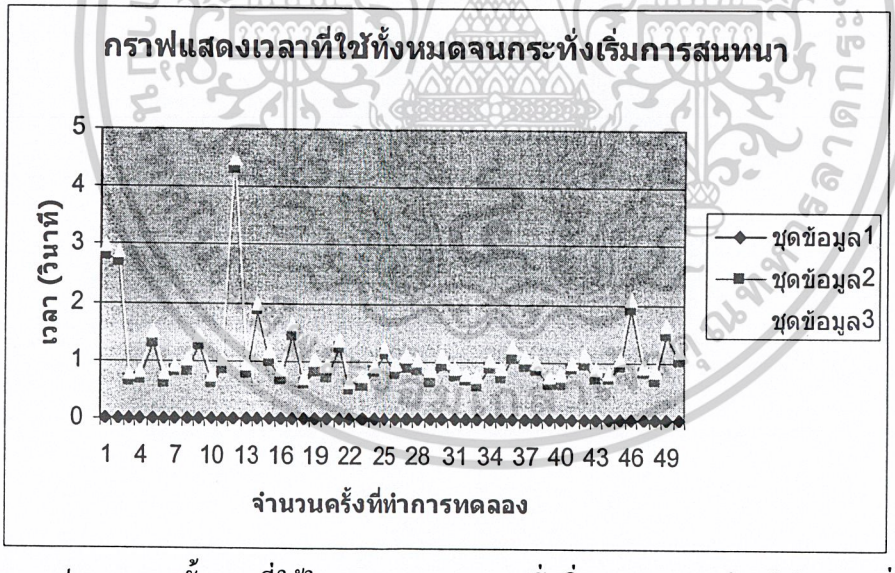
4.3.1 ผลการทดลองในส่วนการวัดคุณภาพของโครงข่ายคอมพิวเตอร์

4.3.1.1 ผลการทดลองในส่วนการวัดเวลาที่ใช้ทั้งหมดจนกระทั่งเริ่มการสนทนา ที่วัดโดยการเรียกจากเครื่องคอมพิวเตอร์ต้นทาง

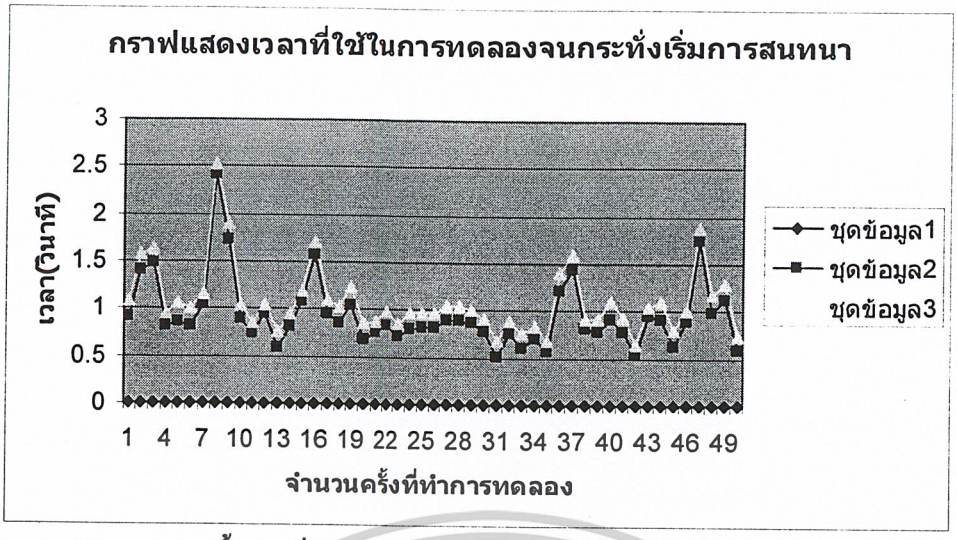
จากกราฟแสดงเวลาทั้งหมดที่ใช้ในการทดลองจนกระทั่งเริ่มการสนทนา ชุดข้อมูล 1 คือเวลาที่เกิดหลังจากการกดปุ่ม Connect คือการวัดเวลาจากการที่คอมพิวเตอร์ต้นทางร้องขอการเชื่อมต่อกับเครื่องปลายทางโดยใช้โปรโตคอลทีซีพี (เป็นเวลาที่เกิดขึ้นโดยการจัดการของโครงข่าย)

ชุดข้อมูล 2 คือ เวลาที่เกิดหลังจากการกดปุ่ม OK ที่หน้าต่างแสดงการร้องขอการเชื่อมต่อที่ชื่อว่า Request for connect บนเครื่องคอมพิวเตอร์ที่ถูกเรียก (เป็นเวลาที่เกิดขึ้นโดยการจัดการของผู้ทำการทดลอง)

ชุดข้อมูล 3 คือ เวลาที่เกิดจากการที่เครื่องคอมพิวเตอร์ที่เรียกมีการตอบรับไปยังเครื่องที่ถูกเรียก ว่าการเชื่อมต่อสมบูรณ์พร้อมที่จะส่งสัญญาณเสียงแล้ว (เป็นเวลาที่เกิดขึ้นโดยการจัดการของโครงข่าย)

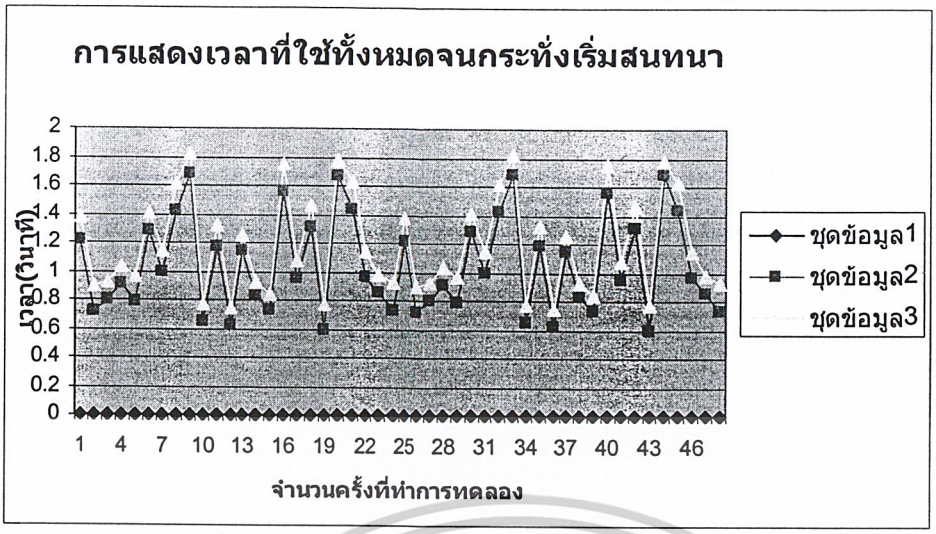


รูปที่ 4.23 กราฟแสดงเวลาทั้งหมดที่ใช้ในการทดลองจนกระทั่งเริ่มการสนทนา โดยโปรแกรมที่ใช้ติดต่อเข้ารหัสเสียงแบบพีซีเอ็มโดยไม่ได้อิงมาตรฐาน



รูปที่ 4.24 กราฟแสดงเวลาทั้งหมดที่ใช้ในการทดลองจนกระทั่งเริ่มการสนทนา โดยโปรแกรมที่ใช้ติดต่อเข้ารหัสเสียงตามมาตรฐาน ITU ด้วย G.711 แบบ A-Law

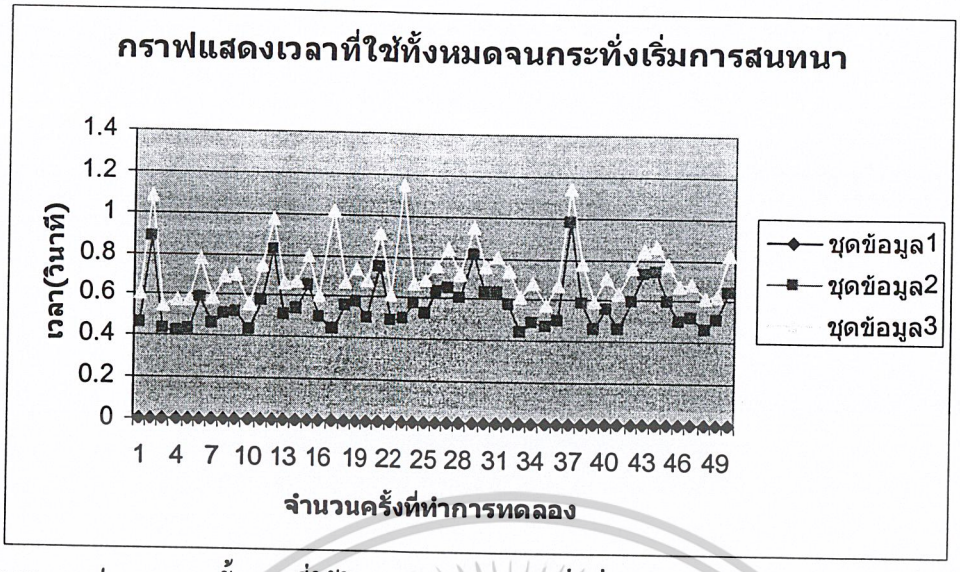
รูปที่ 4.25 กราฟแสดงเวลาทั้งหมดที่ใช้ในการทดลองจนกระทั่งเริ่มการสนทนา โดยโปรแกรมที่ใช้ติดต่อเข้ารหัสเสียงตามมาตรฐาน ITU ด้วย G.711 แบบ μ-Law



รูปที่ 4.26 กราฟแสดงเวลาทั้งหมดที่ใช้ในการทดลองจนกระทั่งเริ่มการสนทนา โดยโปรแกรมที่ใช้ติดต่อเข้ารหัสเสียงตามมาตรฐาน ITU ด้วย G.726

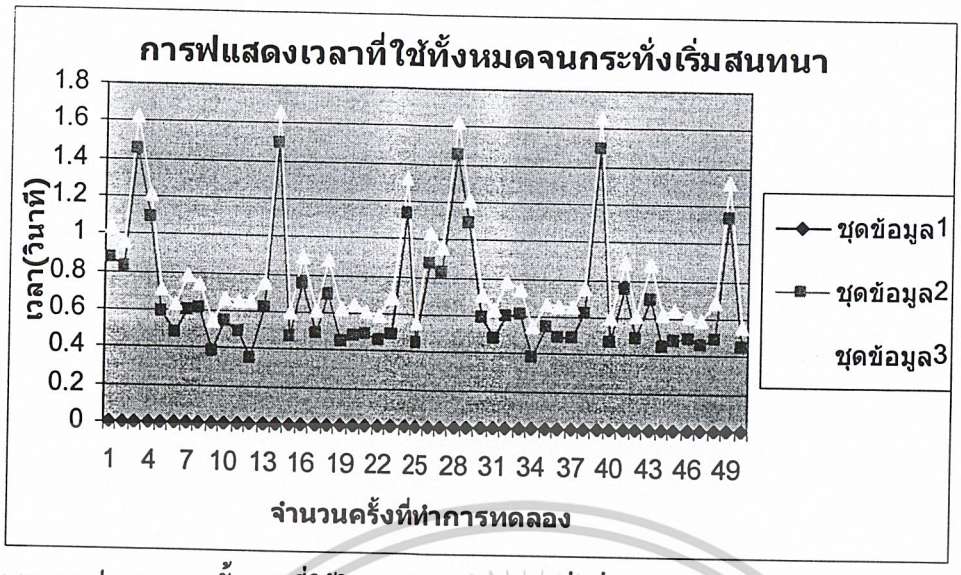
4.3.1.2 ผลการทดลองในส่วนการวัดเวลาที่ใช้ทั้งหมดจนกระทั่งเริ่มการสนทนา ที่วัดโดยการเรียกจากเครื่องคอมพิวเตอร์ปลายทาง

รูปที่ 4.27 กราฟแสดงเวลาทั้งหมดที่ใช้ในการทดลองจนกระทั่งเริ่มการสนทนา โดยโปรแกรมที่ใช้ติดต่อเข้ารหัสเสียงแบบพีซีเอ็มโดยไม่ได้อิงมาตรฐาน



รูปที่ 4.28 กราฟแสดงเวลาทั้งหมดที่ใช้ในการทดลองจนกระทั่งเริ่มการสนทนา โดยโปรแกรมที่ใช้ติดต่อเข้ารหัสเสียงตามมาตรฐาน ITU ด้วย G.711 แบบ A-Law

รูปที่ 4.29 กราฟแสดงเวลาทั้งหมดที่ใช้ในการทดลองจนกระทั่งเริ่มการสนทนา โดยโปรแกรมที่ใช้ติดต่อเข้ารหัสเสียงตามมาตรฐาน ITU ด้วย G.711 แบบ μ-Law

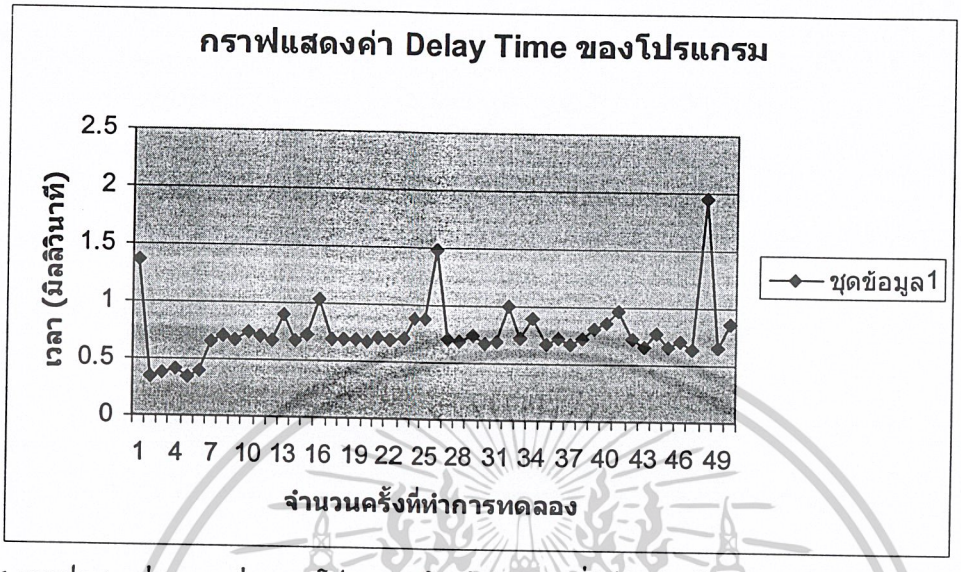


รูปที่ 4.30 กราฟแสดงเวลาทั้งหมดที่ใช้ในการทดลองจนกระทั่งเริ่มการสนทนา โดยโปรแกรมที่ใช้ติดต่อเข้ารหัสเสียงตามมาตรฐาน ITU ด้วย G.726

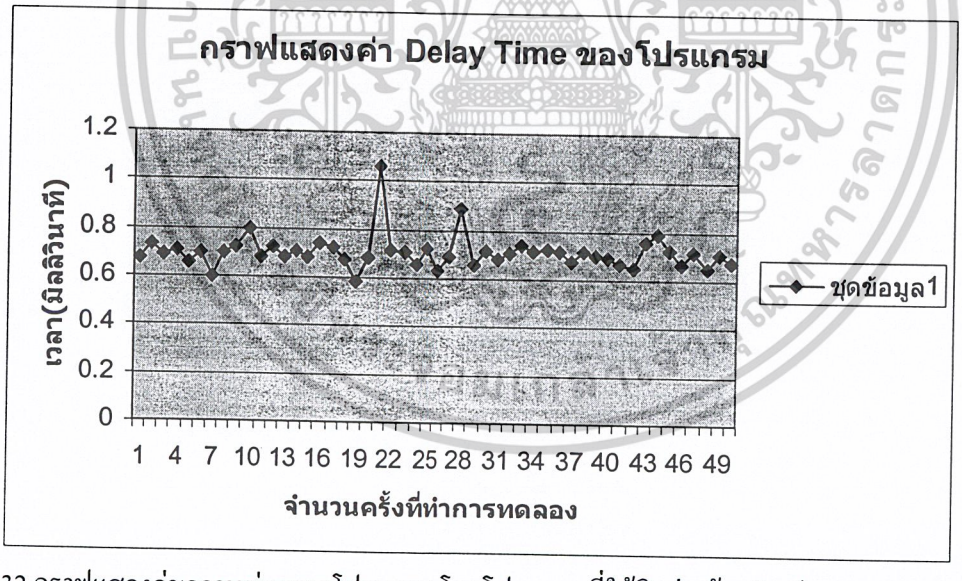
จากผลการทดลองในส่วนนี้จะพบว่าเวลาในชุดข้อมูล 1 คือ เวลาที่เกิดจากการหน่วงของโครงข่าย ส่วนชุดข้อมูล 2 คือ เวลาหน่วงที่เกิดจากการกดปุ่ม OK (เวลาที่แสดงว่ายอมรับการร้องขอการเชื่อมต่อ) จะเห็นว่าเวลาในส่วนนี้เป็นส่วนที่ขึ้นอยู่กับผู้ใช้งาน ไม่เกี่ยวข้องกับโครงข่าย ส่วนข้อมูล 3 เป็นเวลาหน่วงที่เกิดจากโครงข่ายจึงใช้เวลาน้อยกว่าช่วงเวลาของชุดข้อมูล 2 เทียบกับชุดแรก

จากการเข้ารหัสข้อมูลทั้ง 3 แบบ พบว่าไม่ได้ส่งผลกับคุณภาพของโครงข่ายหรือไม่ได้ทำให้การร้องขอการเชื่อมต่อดีขึ้นหรือช้าลง แต่ส่วนที่ทำให้เกิดการหน่วงของสัญญาณเสียงพูดหรือส่วนที่ทำให้คุณภาพของเสียงที่ทำการส่งเปลี่ยนแปลงคือการเข้ารหัสที่เปลี่ยนไป

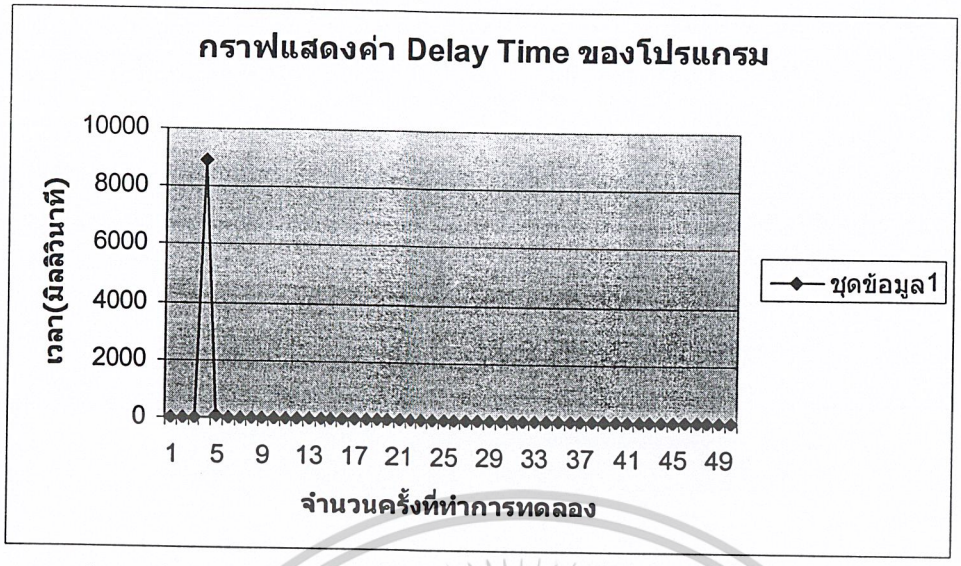
4.3.1.3 ผลการทดลองในส่วนการแสดงค่าเวลาหน่วงของโปรแกรม ที่วัดโดยการเรียกจากเครื่องคอมพิวเตอร์ต้นทาง



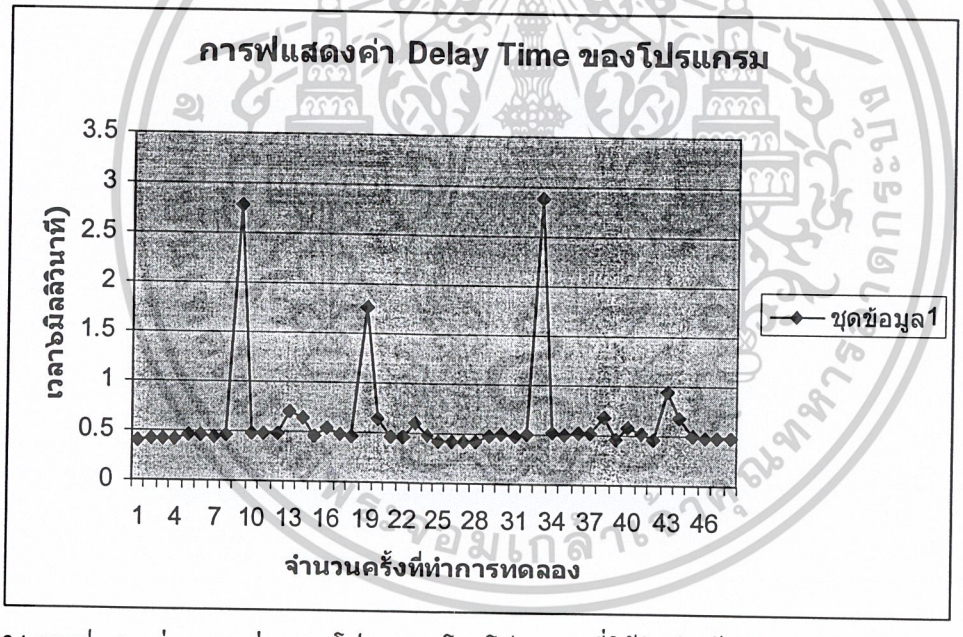
รูปที่ 4.31 กราฟแสดงค่าเวลาหน่วงของโปรแกรม โดยโปรแกรมที่ใช้ติดต่อเข้ารหัสเสียงแบบพีซีเอ็มโดยไม่ได้อิงมาตรฐาน



รูปที่ 4.32 กราฟแสดงค่าเวลาหน่วงของโปรแกรม โดยโปรแกรมที่ใช้ติดต่อเข้ารหัสเสียงตามมาตรฐาน ITU ด้วย G.711 แบบ A-Law

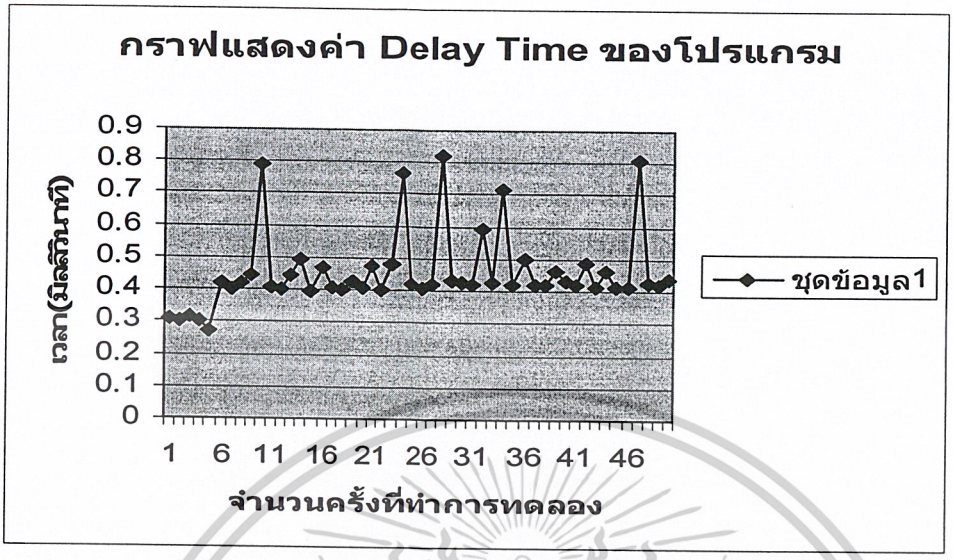


รูปที่ 4.33 กราฟแสดงค่าเวลาหน่วงของโปรแกรม โดยโปรแกรมที่ใช้ติดต่อเข้ารหัสเสียงตามมาตรฐาน ITU ด้วย G.711 แบบ μ-Law

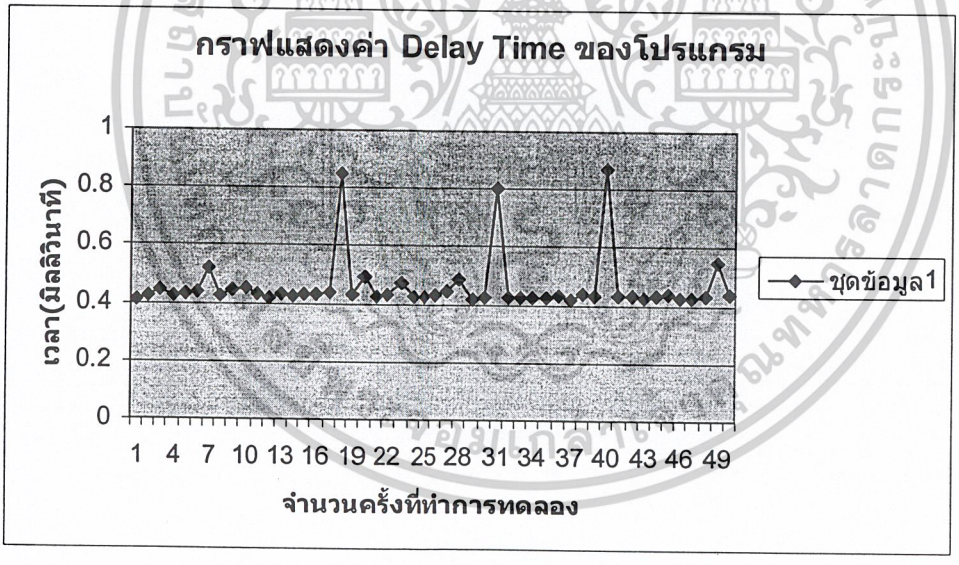


รูปที่ 4.34 กราฟแสดงค่าเวลาหน่วงของโปรแกรม โดยโปรแกรมที่ใช้ติดต่อเข้ารหัสเสียงตามมาตรฐาน ITU ด้วย G.726

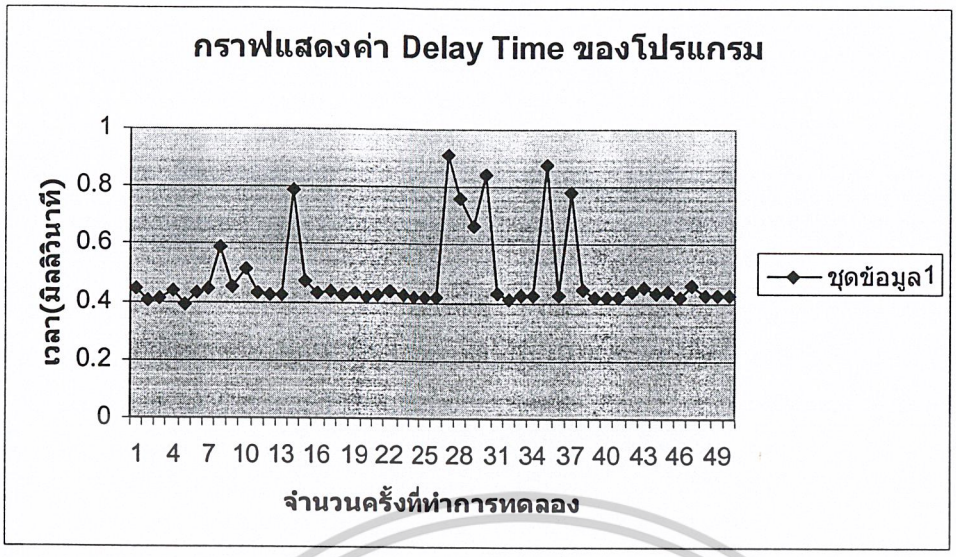
4.3.1.4 ผลการทดลองในส่วนการแสดงค่าเวลาหน่วงของโปรแกรม ที่วัดโดยการเรียกจากเครื่องคอมพิวเตอร์ปลายทาง



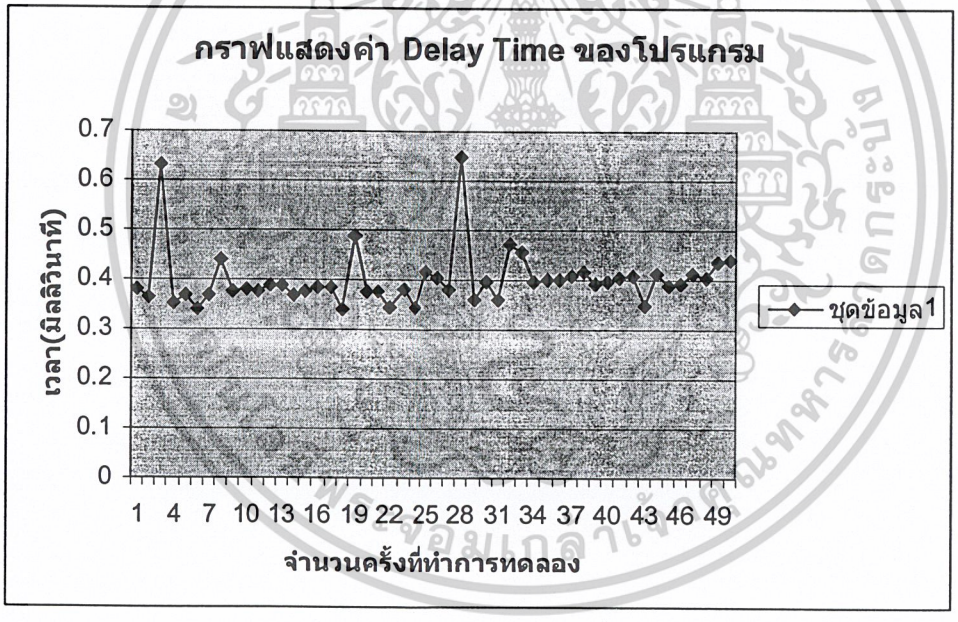
รูปที่ 4.35 กราฟแสดงค่าเวลาหน่วงของโปรแกรม โดยโปรแกรมที่ใช้ติดต่อเข้ารหัสเสียงแบบพีซีเอ็มโดยไม่ได้อิงมาตรฐาน



รูปที่ 4.36 กราฟแสดงค่าเวลาหน่วงของโปรแกรม โดยโปรแกรมที่ใช้ติดต่อเข้ารหัสเสียงตามมาตรฐาน ITU ด้วย G.711 แบบ A-Law



รูปที่ 4.37 กราฟแสดงค่าเวลาหน่วงของโปรแกรม โดยโปรแกรมที่ใช้ติดต่อเข้ารหัสเสียงตามมาตรฐาน ITU ด้วย G.711 แบบ μ-Law



รูปที่ 4.38 กราฟแสดงค่าเวลาหน่วงของโปรแกรม โดยโปรแกรมที่ใช้ติดต่อเข้ารหัสเสียงตามมาตรฐาน ITU ด้วย G 726

จากกราฟแสดงค่าเวลาหน่วงของโปรแกรม ชุดข้อมูล 1 คือ เวลาที่เกิดจากการหน่วงของโครงข่ายโดยโปรโตคอลทีซีพี โดยเราจะพิจารณาสัญญาณบิต SYN (Synchronous) ซึ่งใช้ในการเริ่มต้นขอติดต่อกับปลายทาง และบิต ACK (Acknowledgements) ซึ่งเป็นบิตที่แสดงหมายเลขลำดับการตอบรับ จากกราฟจะนำค่าเวลาที่ได้จากการทดลองทั้ง 50 ครั้งมาทำการหาค่าเฉลี่ย จะได้ค่าดังตาราง

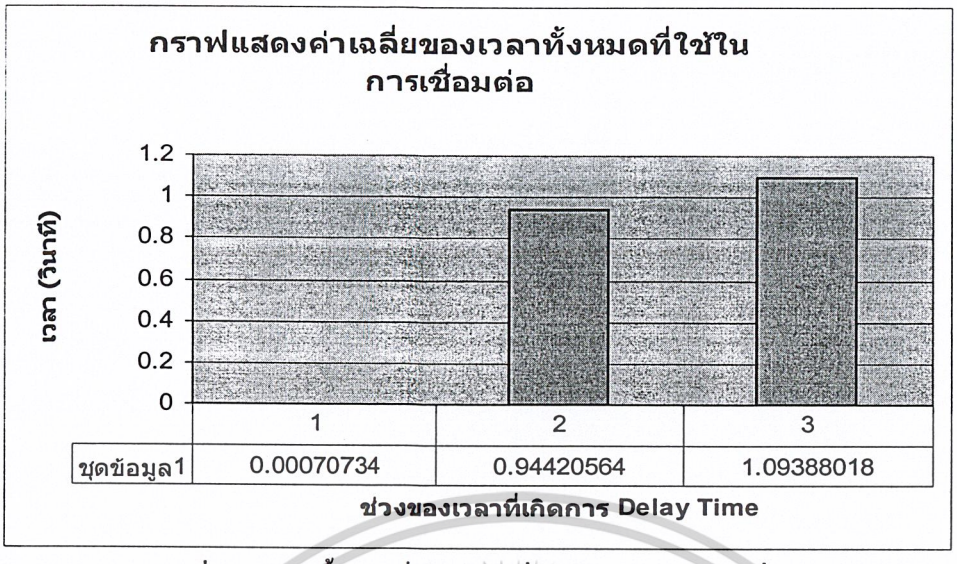
รูปแบบการเข้ารหัส	การเรียกจากเครื่องต้นทาง (มิลลิวินาที)	การเรียกจากเครื่องปลายทาง (มิลลิวินาที)
พีซีเอ็มที่ไม่ได้อิงมาตรฐาน	0.828	0.456
G.711 แบบ A-Law	0.707	0.538
G.711 แบบ μ-Law	179.215	0.486
G.726	0.631	0.401

ตารางที่ 4.7 แสดงค่าเฉลี่ยของค่าเวลาหน่วงของโปรแกรมที่ได้จากการทดลอง 50 ครั้ง

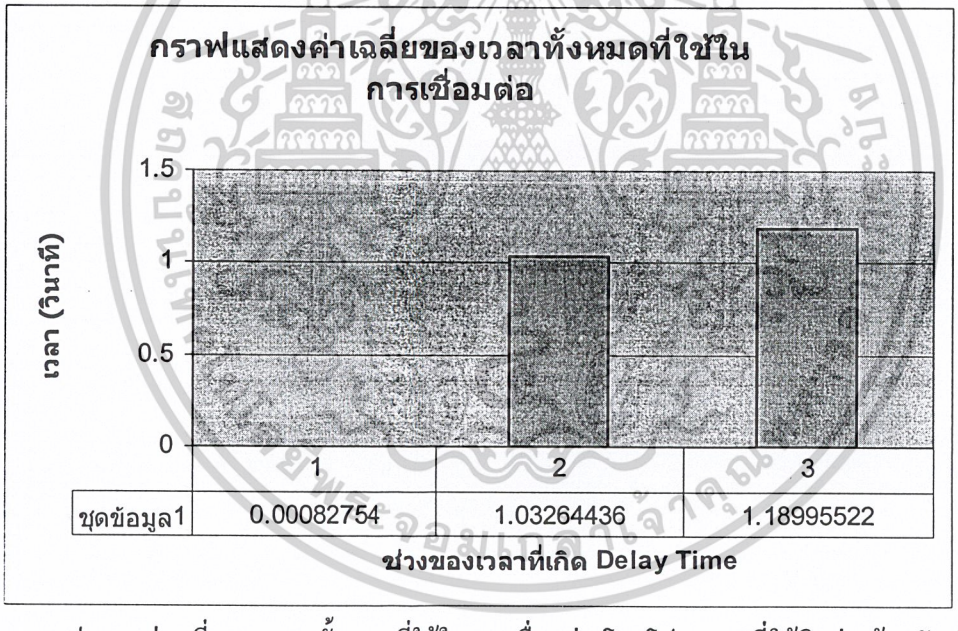
จากตารางจะเห็นว่าเวลาเฉลี่ยของการเรียกจากเครื่องต้นทางมากกว่าการเรียกจากเครื่องปลายทางและค่าประมาณของเวลาจากทั้งตาราง คือ 22.90775 มิลลิวินาที ซึ่งน้อยกว่าค่าที่ถือว่าเกิดการหน่วงของโครงข่าย คือ 150 มิลลิวินาที และจากการสังเกตพบว่าการเข้ารหัสด้วยมาตรฐาน G.711 แบบ μ-Law ที่มีค่าการหน่วงมากกว่าค่าที่ตั้งไว้

4.3.1.5 ผลการทดลองในส่วนการแสดงค่าเฉลี่ยของเวลาที่ใช้ทั้งหมดในการเชื่อมต่อ ที่วัดโดยการเรียกจากเครื่องคอมพิวเตอร์ต้นทาง

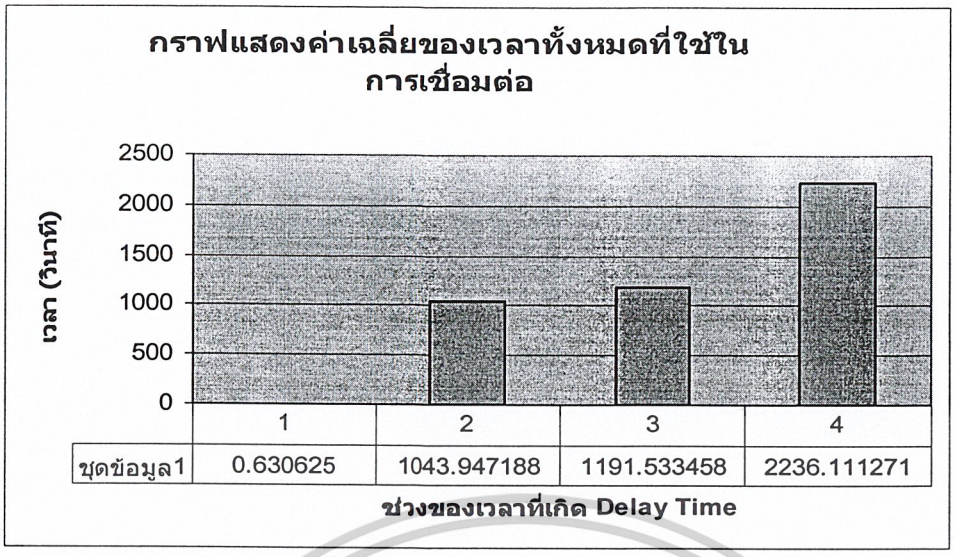
รูปที่ 4.39 กราฟแสดงค่าเฉลี่ยของเวลาทั้งหมดที่ใช้ในการเชื่อมต่อ โดยโปรแกรมที่ใช้ติดต่อเข้ารหัสเสียงแบบพีซีเอ็มโดยไม่ได้อิงมาตรฐาน



รูปที่ 4.40 กราฟแสดงค่าเฉลี่ยของเวลาทั้งหมดที่ใช้ในการเชื่อมต่อ โดยโปรแกรมที่ใช้ติดต่อเข้ารหัสเสียงตามมาตรฐาน ITU ด้วย G.711 แบบ A-Law



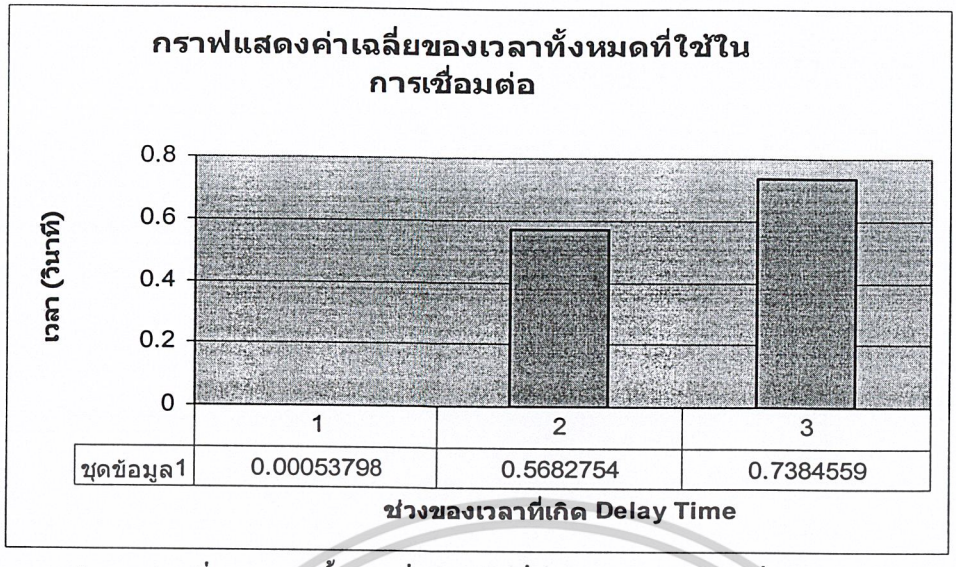
รูปที่ 4.41 กราฟแสดงค่าเฉลี่ยของเวลาทั้งหมดที่ใช้ในการเชื่อมต่อ โดยโปรแกรมที่ใช้ติดต่อเข้ารหัสเสียงตามมาตรฐาน ITU ด้วย G.711 แบบ μ -Law



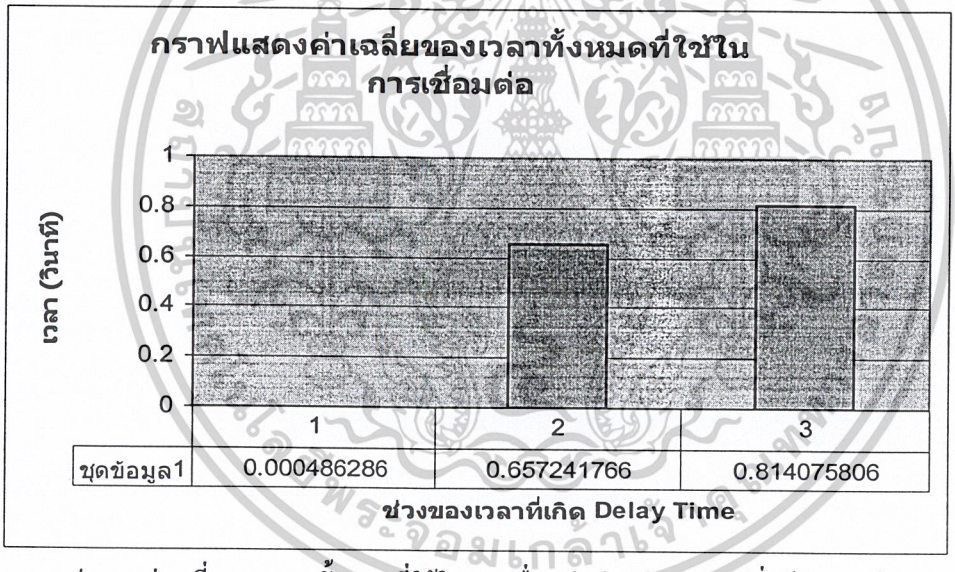
รูปที่ 4.42 กราฟแสดงค่าเฉลี่ยของเวลาทั้งหมดที่ใช้ในการเชื่อมต่อ โดยโปรแกรมที่ใช้ติดต่อเข้ารหัสเสียงตามมาตรฐาน ITU ด้วย G.726

4.3.1.6 ผลการทดลองในส่วนการแสดงค่าเฉลี่ยของเวลาที่ใช้ทั้งหมดในการทดลอง ที่วัดโดยการเรียกจากเครื่องคอมพิวเตอร์ปลายทาง

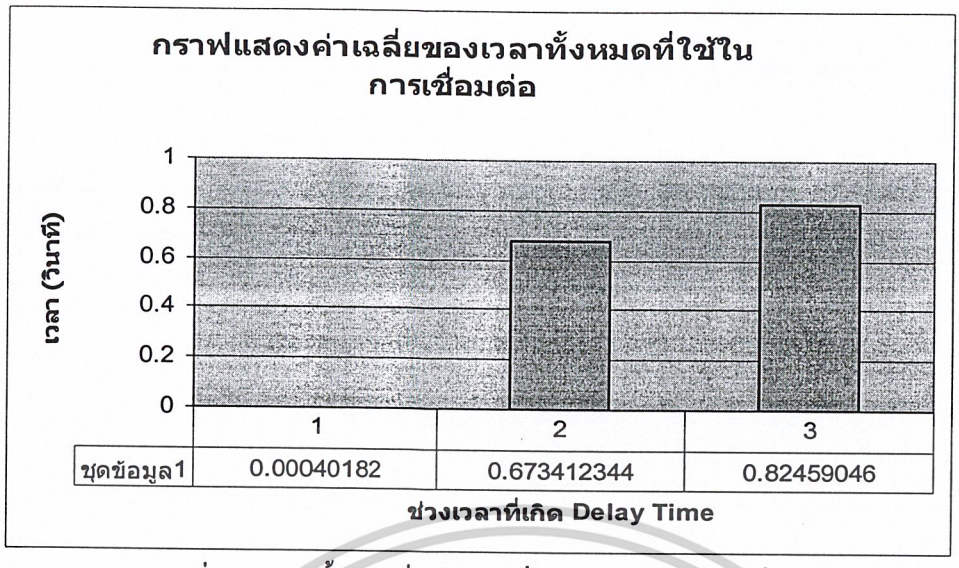
รูปที่ 4.43 กราฟแสดงค่าเฉลี่ยของเวลาทั้งหมดที่ใช้ในการเชื่อมต่อ โดยโปรแกรมที่ใช้ติดต่อเข้ารหัสเสียงแบบพีซีเอ็มโดยไม่ได้อิงมาตรฐาน



รูปที่ 4.44 กราฟแสดงค่าเฉลี่ยของเวลาทั้งหมดที่ใช้ในการเชื่อมต่อ โดยโปรแกรมที่ใช้ติดต่อเข้ารหัสเสียงตามมาตรฐาน ITU ด้วย G.711 แบบ A-Law



รูปที่ 4.45 กราฟแสดงค่าเฉลี่ยของเวลาทั้งหมดที่ใช้ในการเชื่อมต่อ โดยโปรแกรมที่ใช้ติดต่อเข้ารหัสเสียงตามมาตรฐาน ITU ด้วย G.711 แบบ μ-Law



รูปที่ 4.46 กราฟแสดงค่าเฉลี่ยของเวลาทั้งหมดที่ใช้ในการเชื่อมต่อ โดยโปรแกรมที่ใช้ติดต่อเข้ารหัสเสียงตามมาตรฐาน ITU ด้วย G.726

จากกราฟค่าเฉลี่ยของเวลาทั้งหมดที่ใช้ในการเชื่อมต่อ ชุดข้อมูล 1 คือ เวลาที่เกิดการทดลอง 50 ครั้ง แล้วนำมาหาค่าเฉลี่ย จากผลการทดลองสามารถนำข้อมูลมาแสดงดังตารางที่ 4.8

รูปแบบการเข้ารหัส	เครื่องคอมพิวเตอร์ที่ทำการเรียก	เวลาเฉลี่ยที่เกิดจากการหน่วงของโครงข่าย (วินาที)	เวลาหน่วงเฉลี่ยที่เกิดจากการกดปุ่มยอมรับการร้องขอการเชื่อมต่อ (วินาที)	ค่าเวลาทั้งหมดที่ใช้ในการเชื่อมต่อ (วินาที)
พีซีเอ็มที่ไม่ได้อิงมาตรฐาน	เครื่องต้นทาง	0.000828	1.032644	1.189955
	เครื่องปลายทาง	0.000456	0.761789	0.915254
G.711 แบบ A-Law	เครื่องต้นทาง	0.000707	0.944206	1.09388
	เครื่องปลายทาง	0.000538	0.568275	0.738456
G.711 แบบ μ-Law	เครื่องต้นทาง	0.179215	1.747209	1.899662
	เครื่องปลายทาง	0.000486	0.657242	0.814076
G.726	เครื่องต้นทาง	0.000631	1.043947	1.191533
	เครื่องปลายทาง	0.000402	0.673412	0.82459

ตารางที่ 4.8 แสดงค่าเฉลี่ยของเวลาทั้งหมดที่ใช้ในการเชื่อมต่อ

จากตาราง 4.7 ค่าที่ได้จากการทดลอง จะเห็นว่าการเรียกจากเครื่องต้นทาง เวลาเฉลี่ยในการหน่วงของโครงข่ายจะมากกว่าการเรียกจากเครื่องปลายทาง ซึ่งตรงกับการทำคอนเนคชันของโปรโตคอลทีซีพี

ส่วนค่าเวลาทั้งหมดที่ใช้ในการเชื่อมต่อ จะเห็นว่าการเรียกจากเครื่องต้นทางใช้เวลามากกว่า แต่จากการสังเกตจะพบว่าการเข้ารหัสสัญญาณเสียงในแบบต่างๆ เวลาทั้งหมดที่ใช้ในการเชื่อมต่อจะประมาณ 1 วินาที

4.4 การทดลองในส่วนการวัดความผิดพลาดของข้อมูลในส่วนโครงข่าย

ทำการทดลองโดยการส่งแพ็คเกจไปในโครงข่าย (Ping) โดยการทดลองใช้แพ็คเกจขนาด 800 ไบต์ และ 1000 ไบต์ โดยทำการทดลองเป็นเวลา 24 ชั่วโมง ได้ผลดังแสดงในตาราง

รายละเอียดที่เก็บค่าได้		ขนาดแพ็คเกจ 800 ไบต์	ขนาดแพ็คเกจ 1000 ไบต์
Ping statistics for		161.246.18.31	161.246.18.30
Packets: Sent		85736	85736
Packets: Received		85731	85730
Lost		5 (0% loss)	6 (0% loss)
Approximate round trip times in milli-seconds:	Minimum	1 ms	1 ms
	Maximum	188 ms	213 ms
	Average	1 ms	2 ms

ตารางที่ 4.9 แสดงการหาความผิดพลาดของข้อมูลในส่วนโครงข่าย

บทที่ 5
สรุปผลและวิจารณ์

จากการพัฒนาโปรแกรมสื่อสารด้วยเสียงพูดบนระบบเครือข่ายนี้ เราจะกล่าวแยกออกเป็น 2 หัวข้อหลัก ดังนี้

- ส่วนของการติดต่อสื่อสารภายในเครือข่าย
- ส่วนของแนวทางการพัฒนา

5.1 ส่วนของการติดต่อสื่อสารผ่านเครือข่าย

ในส่วนการติดต่อสื่อสารผ่านเครือข่าย เราได้ทำการศึกษาโปรโตคอลรูปแบบต่างๆ ที่คิดว่าเหมาะสมกับโครงงานนี้ โดยใช้โปรโตคอลทีซีพีในส่วนทำการติดต่อ เมื่อสามารถทำการติดต่อสำเร็จแล้ว จะใช้โปรโตคอลยูดีพีในส่วนที่ทำการสนทนา

จากการทดลองรับส่งข้อมูลเสียงบนเครือข่าย โดยในตอนแรกกำหนดอัตราการส่งอยู่ที่ 64 กิโลบิตต่อวินาที อัตราการแซมปลิ้งที่ความถี่ 8000 เฮิร์ต แบบพัลส์โค้ดมอดูเลชั่น ทำให้ได้ข้อสรุปว่าสามารถรับส่งข้อมูลเสียงได้จริง

จากการทดลองส่วนบีบอัดสัญญาณเสียงด้วยวิธีต่างๆ สามารถทำการเข้ารหัสข้อมูลแบบนี้ได้ ทางด้านรับก็สามารถรับไฟล์จากทางด้านส่งแล้วทำการถอดรหัสได้ ซึ่งไฟล์ที่ได้รับใกล้เคียงจากต้นฉบับทางด้านส่งมาก แม้อาจจะผิดพลาดอยู่บ้างแต่ก็อยู่ในระดับที่ยอมรับได้

จากนั้นนำเสียงที่ผ่านการเข้ารหัสด้วยวิธีต่างๆ มาทำการส่งเข้าไปในโครงข่ายอินเตอร์เน็ต ซึ่งประสิทธิภาพของการรับส่งข้อมูลเสียงที่ได้จากการทดลองใช้งานในระบบเครือข่าย พบว่าสัญญาณเสียงที่ได้นับว่าพอจะรับฟังได้เป็นที่น่าพอใจ ถึงแม้ว่าจะมีการหน่วงของสัญญาณเสียงบ้าง ซึ่งถ้าพิจารณาแล้วก็ต้องยอมรับว่าคงจะเป็นเช่นนั้นเพราะเหตุผลต่างๆ เช่น

1. ทราฟฟิกของข้อมูลภายในเครือข่ายมีค่อนข้างสูง ทำให้เกิดการหน่วงของสัญญาณ ตามสภาวะของทราฟฟิกขณะนั้นด้วย
2. พื้นฐานการทำงานของโปรโตคอลทีซีพี/ไอพี ที่ต้องรับแพ็กเกจของข้อมูลทุกๆ แพ็กเกจ
3. ช่วงเวลาในการบันทึกและบีบข้อมูลก่อนส่งออกไปของตัวโปรแกรมเอง

5.2 ส่วนของแนวทางการพัฒนา

ส่วนของแนวทางการพัฒนาโปรแกรมให้มีความสมบูรณ์หรือให้สามารถใช้ได้หลากหลายมากขึ้นอาจทำได้โดย

1. พัฒนาในส่วนของการเข้ารหัสและถอดรหัสแบบพีซีเอ็มให้ใช้งานได้กับไฟล์ในรูปแบบต่างๆ มากขึ้น โดยอาจพัฒนาให้สามารถเข้ารหัสไฟล์ที่เป็นสเตอริโอได้ หรืออาจจะสามารถเข้ารหัสไฟล์ที่มีอัตราการแซมปลิ้ง ต่างๆ นอกจาก 8 kHz

2. พัฒนาในส่วนของอัตราส่วนของการบีบอัดข้อมูล ให้สามารถบีบอัดข้อมูลได้มากขึ้น โดยอาจจะพัฒนาอัตราส่วนของการบีบอัดให้เป็น 4:1 , 8:1

3. อาจจะเปลี่ยนวิธีในการเข้ารหัสและถอดรหัสข้อมูลให้มีประสิทธิภาพมากขึ้น โดยอาจใช้วิธี CELP ซึ่งสามารถบีบอัดข้อมูลได้เล็กกว่าวิธีพีซีเอ็ม

VoIP CODECs	อัตราการส่งข้อมูลที่บีบอัดแล้ว (Kbps)	ความยุ่งยากซับซ้อน	คุณภาพ	ความล่าช้าในการทำเป็นดิจิตอล
G.711 PCM	64	N/A	ดีมาก	แทบจะไม่มี
G.726 ADPCM	40/32/24	ต่ำ (8 MIPS)	ดี (40K) ถึง แย่ (16K)	ต่ำมาก
G.729 CS-ACELP	8	สูง (30 MIPS)	ดี	ต่ำ
G.729A CS-ACELP	8	ปานกลาง	พอใช้	ต่ำ
G.723 MP-MLQ	6.4/5.3	ปานกลาง-สูง(20 MIPS)	ดี (6.4K) พอใช้(5.3K)	สูง
G.723.1 MP-MLQ	6.4/5.3	----------	----------	----------
G.728 LD-CELP	16	สูงมาก (40 MIPS)	ดี	ต่ำ

ตารางที่ 5.1 แสดงการเปรียบเทียบการเข้ารหัสด้วยวิธีต่างๆ โดยมีพีซีเอ็มเป็นตัวเปรียบเทียบ

ภาคผนวก

หนังสืออ้างอิง

1. ยุทธนา ลีลาศวัฒนกุล , คู่มือการเขียนโปรแกรมและการใช้งาน Visual C++ 6.0 ฉบับโปรแกรมเมอร์ , Info press, 2000

2. นิรุธ อำนวยศิลป์ คู่มือการเขียนโปรแกรม Visual C++ Version 6.0 , สำนักพิมพ์ SUCCESS MEDIA , 1989

3. วิวัฒน์ กิรานนท์ , "วิศวกรรมสื่อสาร" , คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง,2544

4. สุวัฒน์ ปุณณชัยยะ , ต้น ตัณฑ์สุทธิวงศ์ , สุพจน์ ปุณณชัยยะ , เปิดโลก TCP/IP และโปรโตคอลของอินเตอร์เน็ต , สำนักพิมพ์ โปรวิชั่น จำกัด , 2002 , หน้า 11-59

5. ธัชพล อัศวเสรีนนท์ , นันทพงศ์ แหยมแสง , ปริญญานิพนธ์การรับส่งข้อมูลเสียงผ่านอินเตอร์เน็ต , 2546

6. ITU-T Recommendation G.711 was published in Fascicle III.4 of the Blue Book

7. ITU-T Recommendation G.726 was prepared by Study Group XV and was approved under the Resolution No. 2

8. ITU-T Recommendation G.729 was prepared by ITU-T Study Group 15 (1993-1996) and was approved under the WTSC Resolution No.1

9. http://www.data-compression.com/speech.html

10. http://www.vocal.com/data_sheets/codecs_voip_g728.html

11. http://www.talkswitch.com/voip/speed_table.php

```
#if !defined(AFX_PLAYMMSOUND_H__F3383123_E3DD_11D3_8685_806A49C10000__INCLUDED_)
#define AFX_PLAYMMSOUND_H__F3383123_E3DD_11D3_8685_806A49C10000__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// PlayMMSound.h : header file
//
#define WM_PLAYMMSOUND_PLAYFILE WM_USER+301
#define WM_PLAYMMSOUND_CLOSEFILE WM_USER+302
#define WM_PLAYMMSOUND_PLAYSOUNDPTR WM_USER+303
#define WM_PLAYMMSOUND_ENDTHREAD WM_USER+304
#define WM_PLAYMMSOUND_SOCKET_INFO WM_USER+305


/////////////////////////////////////////////////////////////////////////////
// CPlayMMSound thread

class CPlayMMSound : public CWinThread
{
    DECLARE_DYNCREATE(CPlayMMSound)
public:
    BYTE* ReadSoundFile(int*dwBytesRead);
    CPlayMMSound(unsigned int Recvsock);           // protected constructor used by dynamic cre
ation
    CPlayMMSound();
protected:
    BOOL OpenSoundFile(CString csFileName);
    LRESULT OnEndThread(WPARAM wParam, LPARAM lParam);
    LRESULT OnSocket_info(WPARAM wParam, LPARAM lParam);


// Attributes
public:
    CEvent* m_pStopEvent;
    CPlaySound* m_pPlaySound;
    BOOL m_bContinuePlaying;

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CPlayMMSound)
    public:
    virtual BOOL InitInstance();
    virtual int ExitInstance();
    //}}AFX_VIRTUAL
    static UINT PlaySound( LPVOID pParam );


// Implementation
protected:
    virtual ~CPlayMMSound();

    // Generated message map functions
    //{{AFX_MSG(CPlayMMSound)
        // NOTE - the ClassWizard will add and remove member functions here.
    //}}AFX_MSG


    afx_msg LRESULT PlaySoundFile(WPARAM wParam, LPARAM lParam);
    afx_msg LRESULT CloseSoundFile(WPARAM wParam, LPARAM lParam);
    afx_msg LRESULT OnPlaySoundPtr(WPARAM wParam, LPARAM lParam);
    afx_msg void OnSocket_Info(WPARAM wParam, LPARAM lParam);
    DECLARE_MESSAGE_MAP()


    char m_FileName[MAX_PATH];
    MMCKINFO m_MMCkInfoParent;
    MMCKINFO m_MMCkInfoChild;
    HMMIO m_hmmio;
    WAVEFORMATEX m_PCMWaveFmtRecord;
    BYTE m_WaveFormatExtras[MAX_PATH]; // for non PCM waveformatex there are extra bytes at the
end
    BYTE * m_SoundBuffer;
    CWinThread* m_pSoundThread;
    int m_BytesToRead;
```

```
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previo
us line.

#endif // !defined(AFX_PLAYMMSOUND_H__F3383123_E3DD_11D3_8685_806A49C10000__INCLUDED_)
```

```
// PlayMMSound.cpp : implementation file
//

#include "stdafx.h"
#include<winsock2.h>
#include "record.h"
#include "PlaySound.h"
#include "PlayMMSound.h"
#include "g711.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
#define SEPH 21
/////////////////////////////////////////////////////////////////////////////
/*#define MCASTADDR      "234.7.7.7"
#define MCASTPORT      27000
#define BUFSIZE        800
#define DEFAULT_COUNT  500*/


//BOOL bloopback = FALSE;
//int multiport = MCASTPORT;

DWORD   dwinterface ,dwmulticastgroup,dwcount;
char *recvbuf;


WSADATA             wsd;
SOCKET              sockm,recvsock;

DWORD               i=0;

struct              sockaddr_in local,remote,from,server;
struct              hostent *host = NULL;

//char              buffer[100];

int                 len = sizeof(struct sockaddr_in),optval,ret,t;
int                 localport = 7000;
/////////////////////////////////////////////////////////////////////////////
#define RECEIVESAMPLES 1504
char temp[SEPH][RECEIVESAMPLES];   //Buffer Queue
int To_Play,To_Fill;     // Counters

short decompressdata[RECEIVESAMPLES];

/////////////////////////////////////////////////////////////////////////////
// CPlayMMSound

 UINT  Fill_Buffer_Queue( LPVOID pParam );
void Receive_Audio_Data();

// Decompress Data
void DecompressedSoundData(char *input,short* output,DWORD dwSamples);
//int Streaming();

/////////////////////////////////////////////////////////////////////////////
// CPlayMMSound
 UINT  PlaySound( LPVOID pParam );

IMPLEMENT_DYNCREATE(CPlayMMSound, CWinThread)

CPlayMMSound::CPlayMMSound(unsigned int Recvsock)
{
//   recvsock = Recvsock;
     //int  k = Streaming();
     m_pPlaySound = NULL;
     strcpy(m_FileName,"");
     ZeroMemory(&m_MMCkInfoParent,sizeof(MMCKINFO));
     ZeroMemory(&m_MMCkInfoChild,sizeof(MMCKINFO));
     m_SoundBuffer = NULL;
     m_pSoundThread = NULL;
     m_pStopEvent = new CEvent(FALSE,TRUE);
}
CPlayMMSound::~CPlayMMSound()
{
```

```
    //recvsock = Recvsock;
    //int  k = Streaming();
    To_Play = 10;
    To_Fill = 1;
    m_pPlaySound = NULL;
    //AfxBeginThread(Fill_Buffer_Queue,NULL,THREAD_PRIORITY_NORMAL);
    strcpy(m_FileName,"");
    ZeroMemory(&m_MMCkInfoParent,sizeof(MMCKINFO));
    ZeroMemory(&m_MMCkInfoChild,sizeof(MMCKINFO));
    m_SoundBuffer = NULL;
    m_pSoundThread = NULL;
    m_pStopEvent = new CEvent(FALSE,TRUE);
}
CPlayMMSound::~CPlayMMSound()
{
    if(m_SoundBuffer)
        delete m_SoundBuffer;
}

BOOL CPlayMMSound::InitInstance()
{
    // TODO:  perform and per-thread initialization here
    return TRUE;
}

int CPlayMMSound::ExitInstance()
{
    // TODO:  perform any per-thread cleanup here
    return CWinThread::ExitInstance();
}

BEGIN_MESSAGE_MAP(CPlayMMSound, CWinThread)
    //{{AFX_MSG_MAP(CPlayMMSound)
        // NOTE - the ClassWizard will add and remove mapping macros here.
    //}}AFX_MSG_MAP
    ON_THREAD_MESSAGE(WM_PLAYMMSOUND_PLAYFILE,PlaySoundFile)
    ON_THREAD_MESSAGE(WM_PLAYMMSOUND_CLOSEFILE,CloseSoundFile)
    ON_THREAD_MESSAGE(WM_PLAYMMSOUND_PLAYSOUNDPTR,OnPlaySoundPtr)
    ON_THREAD_MESSAGE(WM_PLAYMMSOUND_ENDTHREAD,OnEndThread)
    ON_THREAD_MESSAGE(WM_PLAYMMSOUND_SOCKET_INFO,OnSocket_Info)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CPlayMMSound message handlers

LRESULT  CPlayMMSound::PlaySoundFile(WPARAM wParam,LPARAM lParam)
{
    char *pFileName = (char*) lParam;
    strcpy(m_FileName,"in.wav");//pFileName);
    if(!OpenSoundFile(m_FileName))
        return FALSE;
    return TRUE;
}
BOOL CPlayMMSound::OpenSoundFile(CString csFileName)
{
    // code taken from Visual C++ Multimedia -- Aitken and Jarol p 122

    CloseSoundFile(0,0);
    m_hmmio = mmioOpen((LPSTR)(LPCTSTR)csFileName,NULL,MMIO_READ);
    if(!m_hmmio)
    {
        AfxMessageBox("unable to open Sound MM File");
        return FALSE;
    }
    m_MMCkInfoParent.fccType = mmioFOURCC('W','A','V','E');
    int errorCode = mmioDescend(m_hmmio, &m_MMCkInfoParent,NULL,MMIO_FINDRIFF);
    if(errorCode)
    {
        AfxMessageBox("Error descending into file");
        mmioClose(m_hmmio,0);
        m_hmmio = NULL;
        return FALSE;
    }
    m_MMCkInfoChild.ckid = mmioFOURCC('f','m','t',' ');
    errorCode = mmioDescend(m_hmmio,&m_MMCkInfoChild,&m_MMCkInfoParent,MMIO_FINDCHUNK);
    if(errorCode)
    {
        AfxMessageBox("Error descending in file");
```

```
        mmioClose(m_hmmio,0);
        m_hmmio = NULL;
        return FALSE;
    }
    DWORD bytesRead = mmioRead(m_hmmio,(LPSTR)&m_PCMWaveFmtRecord,m_MMCkInfoChild.cksize);
    if(bytesRead < 0)
    {
        AfxMessageBox("Error reading PCM wave format record");
        mmioClose(m_hmmio,0);
        return FALSE;
    }

    // open output sound file
    errorCode = mmioAscend(m_hmmio,&m_MMCkInfoChild,0);
    if(errorCode)
    {
        AfxMessageBox("Error ascending in File");
        mmioClose(m_hmmio,0);
        m_hmmio = NULL;
        return FALSE;
    }
    m_MMCkInfoChild.ckid = mmioFOURCC('d','a','t','a');
    errorCode = mmioDescend(m_hmmio,&m_MMCkInfoChild,
        &m_MMCkInfoParent,MMIO_FINDCHUNK);
    if(errorCode)
    {
        AfxMessageBox("error reading data chunk");
        mmioClose(m_hmmio,0);
        m_hmmio = NULL;
        return FALSE;
    }
    m_BytesToRead = m_PCMWaveFmtRecord.nChannels*
        (m_PCMWaveFmtRecord.wBitsPerSample/sizeof(BYTE))
        *m_PCMWaveFmtRecord.nBlockAlign*100;
    if(m_BytesToRead > m_MMCkInfoChild.cksize)
        m_BytesToRead = m_MMCkInfoChild.cksize;

    m_bContinuePlaying = TRUE;
    m_pStopEvent->ResetEvent();
    m_pSoundThread = AfxBeginThread(PlaySound,this);

    return TRUE;

}
BYTE* CPlayMMSound::ReadSoundFile(int*dwBytesRead)
{
    if(m_BytesToRead <= 0)
        return NULL;
    if(!m_hmmio)
        return NULL;
    TRACE("ReadSoundFile\n");
    BYTE * pSoundBuffer = new BYTE[m_BytesToRead];
    if(!pSoundBuffer)
        return NULL;
    DWORD dwRetc = mmioRead(m_hmmio,(LPSTR)pSoundBuffer, m_BytesToRead);
    if(dwRetc == -1)
    {
        AfxMessageBox("Error reading WAVE file\n");
        if(pSoundBuffer)
            delete pSoundBuffer;
        return NULL;
    }
    else if(dwRetc == 0)
    {
        CloseSoundFile(0,0);
        m_bContinuePlaying = FALSE;
    }
    if(dwBytesRead)
        *dwBytesRead = dwRetc;
    char debugBuffer[MAX_PATH];
    sprintf(debugBuffer,"read %d bytes\n",dwRetc);
    TRACE(debugBuffer);
    return pSoundBuffer;
}
LRESULT CPlayMMSound::CloseSoundFile(WPARAM wParam,LPARAM lParam)
{
    if(m_pStopEvent)
        m_pStopEvent->SetEvent();
```

```
    TRACE("STOP EVENT SET\n");
    m_bContinuePlaying = FALSE;
    if(m_hmmio)
    {
        mmioClose(m_hmmio,0);
        m_hmmio = NULL;
    }
    return TRUE;
}
LRESULT  CPlayMMSound::OnPlaySoundPtr(WPARAM wParam,LPARAM lParam)
{
    CPlaySound*pPlaySound = (CPlaySound*)lParam;
    m_pPlaySound = pPlaySound;
    return TRUE;
}
LRESULT CPlayMMSound::OnEndThread(WPARAM wParam, LPARAM lParam)
{
    CloseSoundFile(0,0);
    ::PostQuitMessage(0);
    return TRUE;
}
void CPlayMMSound::OnSocket_Info(WPARAM wParam, LPARAM lParam)
{
    recvsock =lParam;// unsigned int(lParam);
    To_Play = 10;
    To_Fill = 1;
    /*char value[30];
    itoa(recvsock,value,10);
    strcat(value," Socket Number In -- PlayMM Sound  File thread");
    AfxMessageBox(value);*/
//  AfxMessageBox("Socket received");
    AfxBeginThread(Fill_Buffer_Queue,NULL,THREAD_PRIORITY_NORMAL);
    /*CloseSoundFile(0,0);
    ::PostQuitMessage(0);*/
    //return TRUE;
}

UINT CPlayMMSound::PlaySound( LPVOID pParam )
{
    CPlayMMSound* pPlayMMSound = (CPlayMMSound*) pParam;
    if(pPlayMMSound &&
            pPlayMMSound->m_pPlaySound)
    {
        pPlayMMSound->m_pPlaySound->PostThreadMessage(WM_PLAYSOUND_STARTPLAYING,GetCurrentThrea
dId(),(LPARAM)0);
    }

    if(!pPlayMMSound)
        return FALSE;
    if(pPlayMMSound && !pPlayMMSound->m_pPlaySound)
        return FALSE;
    HANDLE hHandle[2];
    hHandle[1] = (HANDLE) pPlayMMSound->m_pPlaySound->m_pSemaphore->m_hObject;
    hHandle[0] = (HANDLE) pPlayMMSound->m_pStopEvent->m_hObject;
    int dwBytesRead = 0;
    BYTE* pSoundBuffer = pPlayMMSound->ReadSoundFile(&dwBytesRead);
    char value[30];
    itoa(dwBytesRead,value,10);
      //AfxMessageBox(value);
    while(pPlayMMSound->m_bContinuePlaying)
    {
        DWORD dwRetc = WaitForMultipleObjects(2,hHandle,FALSE,INFINITE);
        if(dwRetc == WAIT_FAILED)
        {
            DWORD dwRetc = ::GetLastError();
            char errorBuffer[MAX_PATH];
            sprintf(errorBuffer,"WaitForMultipleObjects failed: %d\n",dwRetc);
            TRACE(errorBuffer);

        }
        dwRetc -= WAIT_OBJECT_0;
        if(dwRetc == 0) // stop event;
        {
            TRACE("STOP Event seen\n");
            break;
        }
        if(!pPlayMMSound->m_bContinuePlaying)
            break;
```

```
        WAVEHDR* pWaveHdr = new WAVEHDR;
        ZeroMemory(pWaveHdr,sizeof(WAVEHDR));
        pWaveHdr->lpData = (char*)pSoundBuffer;
        pWaveHdr->dwBufferLength = dwBytesRead;
        if(pPlayMMSound &&
            pPlayMMSound->m_pPlaySound)
        {
            pPlayMMSound->m_pPlaySound->PostThreadMessage(WM_PLAYSOUND_PLAYBLOCK,GetCurrentThre
adId(),(LPARAM)pWaveHdr);
        }
        //pSoundBuffer = pPlayMMSound->ReadSoundFile(&dwBytesRead);
/*************************DECOMPRESSED_SOUND*********************************/
    void* input =temp[To_Play] ;
    DecompressedSoundData((char*)input,decompressdata,RECEIVESAMPLES);
    if ( To_Play > 1)memset(temp[To_Play-1],0,sizeof(char)*RECEIVESAMPLES);
    if(To_Play==1)memset(temp[SEPH-1],0,RECEIVESAMPLES*sizeof(char));

/*****************************************************************/
            pSoundBuffer =  (BYTE*)decompressdata;//recvbuf;
        To_Play = To_Play + 1  ;
        if(To_Play==SEPH) To_Play = 1;
    //  itoa(dwBytesRead,value,10);
     // AfxMessageBox(value);

    }
    if(pPlayMMSound &&
        pPlayMMSound->m_pPlaySound)
    {

    //  pPlayMMSound->m_pPlaySound->PostThreadMessage(WM_PLAYSOUND_STOPPLAYING,GetCurrentThread
Id(),(LPARAM)0);
    }


    return TRUE;
}



UINT  Fill_Buffer_Queue( LPVOID pParam )
{
    //, (struct sockaddr *)&from ,&len )
//AfxMessageBox("Ready to receive");

    while(1)
    {
if(( ret = recv(recvsock,temp[To_Fill],RECEIVESAMPLES, 0))== SOCKET_ERROR)
{   AfxMessageBox("error in receiving");
exit(0);}
//AfxMessageBox(temp[To_Fill]);
  To_Fill = To_Fill + 1;
  if(To_Fill==SEPH) To_Fill = 1;
   }

 return 0;
}


// Decompress Data
void DecompressedSoundData(char *input,short* output,DWORD dwSamples)
{


   if (COMPRESS_TYPE == "alaw"){
   alaw_expandV2(dwSamples,output,input);
   } else {
   ulaw_expandV2(dwSamples,input,output);
   }

}
```

```
#if !defined(AFX_PLAYSOUND_H__5260C01C_03B2_11D2_A421_FC4B2C882A60__INCLUDED_)
#define AFX_PLAYSOUND_H__5260C01C_03B2_11D2_A421_FC4B2C882A60__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// PlaySound.h : header file

#include <afxmt.h>


//

#ifndef __COMPLEX__
#define __COMPLEX__
typedef struct
{
    float real;
    float imag;
}
COMPLEXNUMBER;
#endif



/////////////////////////////////////////////////////////////////////////////
// CPlaySound thread
#define WM_PLAYSOUND_STARTPLAYING WM_USER+600
#define WM_PLAYSOUND_STOPPLAYING WM_USER+601
#define WM_PLAYSOUND_PLAYBLOCK WM_USER+602
#define WM_PLAYSOUND_ENDTHREAD  WM_USER+603


class CRecordDlg;

class CPlaySound : public CWinThread
{
    DECLARE_DYNCREATE(CPlaySound)
public:
    CPlaySound(int iHertz=8000);           // protected constructor used by dynamic creation


// Attributes
public:

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CPlaySound)
    public:
    virtual BOOL InitInstance();
    virtual int ExitInstance();
    //}}AFX_VIRTUAL

    virtual void ProcessSoundData(void *sound, DWORD dwSamples);

// Implementation
protected:
    virtual ~CPlaySound();

    // Generated message map functions
    //{{AFX_MSG(CPlaySound)
        // NOTE - the ClassWizard will add and remove member functions here.
    //}}AFX_MSG

    afx_msg LRESULT OnStartPlaying(WPARAM wParam, LPARAM lParam);
    afx_msg LRESULT OnStopPlaying(WPARAM wParam, LPARAM lParam);
    afx_msg LRESULT OnEndPlaySoundData(WPARAM wParam, LPARAM lParam);
    afx_msg LRESULT OnWriteSoundData(WPARAM wParam, LPARAM lParam);
    afx_msg LRESULT OnEndThread(WPARAM wParam, LPARAM lParam);
    DECLARE_MESSAGE_MAP()
```

```
protected:
    int m_nOutputBuffers;
    int m_nMaxOutputBuffers;
    WAVEFORMATEX m_WaveFormatEx;
    BOOL m_bPlay;
    HWAVEOUT m_hPlay;
public:
    CSemaphore* m_pSemaphore;

protected:

LPWAVEHDR CreateWaveHeader();

};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previo
us line.

#endif // !defined(AFX_PLAYSOUND_H__5260C01C_03B2_11D2_A421_FC4B2C882A60__INCLUDED_)
```

```
// PlaySound.cpp : implementation file
//

#ifndef __COMPLEX__
#define __COMPLEX__
typedef struct
{
    float real;
    float imag;
}
COMPLEXNUMBER;
#endif

#include "stdafx.h"
#include "winbase.h"
#include <mmsystem.h>
#include <math.h>
#include "g711.h"


#include "record.h"
#include "PlaySound.h"

#include "RecordDlg.h"



#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CPlaySound

IMPLEMENT_DYNCREATE(CPlaySound, CWinThread)

#define BLOCKSAHEAD 3

CPlaySound::CPlaySound(int iHertz)
{
    m_nOutputBuffers = 0;
    m_nMaxOutputBuffers = 0;
    memset(&m_WaveFormatEx,0x00,sizeof(m_WaveFormatEx));
    m_WaveFormatEx.wFormatTag = WAVE_FORMAT_PCM;
    m_WaveFormatEx.nChannels = 1;
    m_WaveFormatEx.wBitsPerSample = 16;//16;
    m_WaveFormatEx.cbSize = 0;
    m_WaveFormatEx.nSamplesPerSec = 8000;//iHertz; // 20050;
    m_WaveFormatEx.nAvgBytesPerSec = m_WaveFormatEx.nSamplesPerSec
        *(m_WaveFormatEx.wBitsPerSample/8);
    m_WaveFormatEx.nBlockAlign =
        (m_WaveFormatEx.wBitsPerSample/8)*
        m_WaveFormatEx.nChannels;
    m_pSemaphore = new CSemaphore(BLOCKSAHEAD,BLOCKSAHEAD);

    m_bPlay = FALSE;

}

CPlaySound::~CPlaySound()
{
}

BOOL CPlaySound::InitInstance()
{
    // TODO:  perform and per-thread initialization here
    return TRUE;
}

int CPlaySound::ExitInstance()
{
    // TODO:  perform any per-thread cleanup here
    return CWinThread::ExitInstance();
}
```

```
BEGIN_MESSAGE_MAP(CPlaySound, CWinThread)
    //{{AFX_MSG_MAP(CPlaySound)
        // NOTE - the ClassWizard will add and remove mapping macros here.
    //}}AFX_MSG_MAP
    ON_THREAD_MESSAGE(WM_PLAYSOUND_STARTPLAYING, OnStartPlaying)
    ON_THREAD_MESSAGE(WM_PLAYSOUND_STOPPLAYING, OnStopPlaying)
    ON_THREAD_MESSAGE(WM_PLAYSOUND_PLAYBLOCK, OnWriteSoundData)
    ON_THREAD_MESSAGE(MM_WOM_DONE, OnEndPlaySoundData)
    ON_THREAD_MESSAGE(WM_PLAYSOUND_ENDTHREAD,OnEndThread)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CPlaySound message handlers
LRESULT CPlaySound::OnStartPlaying(WPARAM wParam, LPARAM lParam)
{
    DWORD mmReturn = 0;

    if(m_bPlay)
        return FALSE;

    if(!m_bPlay)
    {
        // open wavein device
        MMRESULT mmReturn = ::waveOutOpen( &m_hPlay, WAVE_MAPPER,
            &m_WaveFormatEx, ::GetCurrentThreadId(), 0, CALLBACK_THREAD);
        if(mmReturn )
        {
            char errorbuffer[MAX_PATH];
            char errorbuffer1[MAX_PATH];
            waveOutGetErrorText( mmReturn,
                                 errorbuffer,
                                 MAX_PATH);
            sprintf(errorbuffer1,"WAVEOUT:%x:%s",mmReturn,errorbuffer);
            AfxMessageBox(errorbuffer1);

        }



        if(!mmReturn)
        {
            m_bPlay = TRUE;
        }

    }
    return TRUE;
}
LRESULT CPlaySound::OnStopPlaying(WPARAM wParam, LPARAM lParam)
{
    MMRESULT mmReturn = 0;
    if(!m_bPlay)
        return FALSE;

    if(m_bPlay)
    {
        mmReturn = ::waveOutReset(m_hPlay);

        if(!mmReturn)
            m_bPlay = FALSE;
        Sleep(500);
        if(!mmReturn)
            mmReturn = ::waveOutClose(m_hPlay);
        return mmReturn;
    }
    return TRUE;
}
LRESULT CPlaySound::OnEndPlaySoundData(WPARAM wParam, LPARAM lParam)
{
    LPWAVEHDR lpHdr = (LPWAVEHDR) lParam;
    if(lpHdr)
    {
        ::waveOutUnprepareHeader(m_hPlay, lpHdr, sizeof(WAVEHDR));

        if(lpHdr->lpData)
            delete (BYTE*) lpHdr->lpData;
        delete lpHdr;
        m_pSemaphore->Unlock();
    }
```

```
    return ERROR_SUCCESS;
}
LRESULT CPlaySound::OnWriteSoundData(WPARAM wParam, LPARAM lParam)
{
    LPWAVEHDR lpHdr = (LPWAVEHDR) lParam;
    MMRESULT mmResult = 0;


    if(lpHdr)
    {
        char debugbuffer[256];
        sprintf(debugbuffer, "SOUND BUFFER written: %d, %d\n",lpHdr->dwBufferLength,m_nOutputBu
ffers);
        TRACE(debugbuffer);
        if(m_bPlay)
        {
            void* lpByte = (short int*) lpHdr->lpData;
            DWORD dwSamples = lpHdr->dwBufferLength/sizeof(short int);

//      ProcessSoundData(lpByte, dwSamples);

            mmResult = ::waveOutPrepareHeader(m_hPlay, lpHdr, sizeof(WAVEHDR));
            if(mmResult)
                TRACE("error from waveoutpreparehead\n");

            mmResult = ::waveOutWrite(m_hPlay, lpHdr, sizeof(WAVEHDR));
            if(mmResult)
                TRACE("error from waveoutwrite\n");
            m_nOutputBuffers++;
        }
    }


    return ERROR_SUCCESS;
}

LPWAVEHDR CPlaySound::CreateWaveHeader()
{
    LPWAVEHDR lpHdr = new WAVEHDR;
    ZeroMemory(lpHdr, sizeof(WAVEHDR));
    BYTE* lpByte = new BYTE[(m_WaveFormatEx.nBlockAlign*SOUNDSAMPLES)];

    lpHdr->lpData = (char *) lpByte;
    lpHdr->dwBufferLength = (m_WaveFormatEx.nBlockAlign*SOUNDSAMPLES);
    return lpHdr;

}

// Decompress Data
void CPlaySound::ProcessSoundData(void *input,DWORD dwSamples)
{

    short *output = new short[dwSamples];



    if (COMPRESS_TYPE == "alaw"){
    alaw_expandV2(dwSamples,output,(char*)input);
    } else {
    ulaw_expandV2(dwSamples,(char*)input,output);
    }

    memcpy((short*)input,output,dwSamples*sizeof(short));

    delete(output);


}
LRESULT CPlaySound::OnEndThread(WPARAM wParam, LPARAM lParam)
{
    if(m_bPlay)
        OnStopPlaying(0,0);
    ::PostQuitMessage(0);
    return TRUE;
}
```

```
// record.h : main header file for the RECORD application
//

#if !defined(AFX_RECORD_H__5260C00F_03B2_11D2_A421_FC4B2C882A60__INCLUDED_)
#define AFX_RECORD_H__5260C00F_03B2_11D2_A421_FC4B2C882A60__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"       // main symbols
#include "RecordSound.h"
#include "PlaySound.h"
#include "PlayMMSound.h"



/////////////////////////////////////////////////////////////////////////////
// CRecordApp:
// See record.cpp for the implementation of this class
//

class CRecordApp : public CWinApp
{
public:
    CRecordApp();

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CRecordApp)
    public:
    virtual BOOL InitInstance();
    //}}AFX_VIRTUAL
    virtual int ExitInstance();

// Implementation

    //{{AFX_MSG(CRecordApp)
        // NOTE - the ClassWizard will add and remove member functions here.
        //    DO NOT EDIT what you see in these blocks of generated code !
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
    protected:
        BOOL m_bRecording;
        CRecordSound* m_pRecordSound;
        CPlaySound* m_pPlaySound;
        CWriteSoundFile* m_pWriteSound;
        CPlayMMSound* m_pPlayMMSound;

    protected:
        BOOL InitRecording();
        BOOL InitPlaying();
        BOOL InitWriting();
        BOOL InitPlayMMSound();
        BOOL Streaming();

};


/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previo
us line.

#endif // !defined(AFX_RECORD_H__5260C00F_03B2_11D2_A421_FC4B2C882A60__INCLUDED_)
```

```
// record.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include<winsock2.h>
#include <mmsystem.h>
#include "record.h"
#include "recordDlg.h"
#include "PlaySound.h"
#include "RecordSound.h"
#include "WriteSoundFile.h"
#include "PlayMMSound.h"
//#include <afxsock.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
/////////////////////////////////////////////////////////////////////////////
#define MCASTADDR        "234.7.7.7"
#define MCASTPORT        27000
#define BUFSIZE          800
#define DEFAULT_COUNT    500


BOOL bloopback = FALSE;
int multiport = MCASTPORT;

DWORD   dwinterface1 ,dwmulticastgroup1,dwcount1;
char *recvbuf1;


WSADATA             wsd1;
SOCKET              sockm1,recvsock1;

DWORD               i1=0;

struct              sockaddr_in local1,remote1,from1,server1;
struct              hostent *host1 = NULL;

char                buffer1[100];

int                 len1 = sizeof(struct sockaddr_in),optval1,ret1,t1;
int                 localport1 = 2000;
/////////////////////////////////////////////////////////////////////////////
// CRecordApp
int Streaming();
BEGIN_MESSAGE_MAP(CRecordApp, CWinApp)
    //{{AFX_MSG_MAP(CRecordApp)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        //    DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG
    ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CRecordApp construction

CRecordApp::CRecordApp()
{
    // TODO: add construction code here,
    // Place all significant initialization in InitInstance

}

/////////////////////////////////////////////////////////////////////////////
// The one and only CRecordApp object

CRecordApp theApp;

/////////////////////////////////////////////////////////////////////////////
// CRecordApp initialization
int CRecordApp::ExitInstance()
{
    if(m_pPlayMMSound)
    {
        m_pPlayMMSound->PostThreadMessage(WM_PLAYMMSOUND_ENDTHREAD,0,0);
```

```
        ::WaitForSingleObject(m_pPlayMMSound->m_hThread, 5000);
        m_pPlayMMSound = NULL;

    }
    if(m_pPlaySound)
    {
        m_pPlaySound->PostThreadMessage(WM_PLAYSOUND_ENDTHREAD,0,0);
        ::WaitForSingleObject(m_pPlaySound->m_hThread, 5000);
        m_pPlaySound = NULL;

    }
    if(m_pRecordSound)
    {
        m_pRecordSound->PostThreadMessage(WM_RECORDSOUND_ENDTHREAD,0,0);
        ::WaitForSingleObject(m_pRecordSound->m_hThread, 5000);
        m_pRecordSound = NULL;
    }
    if(m_pWriteSound)
    {
        m_pWriteSound->PostThreadMessage(WM_WRITESOUNDFILE_ENDTHREAD,0,0);
        ::WaitForSingleObject(m_pWriteSound->m_hThread, 5000);
        m_pWriteSound = NULL;

    }


    return 0;
}
BOOL CRecordApp::InitInstance()
{
/*  if (!AfxSocketInit())
    {
        //AfxMessageBox(IDP_SOCKETS_INIT_FAILED);
        return FALSE;
    }*/
    AfxEnableControlContainer();

    // Standard initialization
    // If you are not using these features and wish to reduce the size
    //  of your final executable, you should remove from the following
    //  the specific initialization routines you do not need.

#ifdef _AFXDLL
    Enable3dControls();         // Call this when using MFC in a shared DLL
#else
    Enable3dControlsStatic();   // Call this when linking to MFC statically
#endif
//int k = Streaming();
    InitRecording();
    InitPlaying();
    InitWriting();
    InitPlayMMSound();

    if(m_pRecordSound)
    {
        m_pRecordSound->PostThreadMessage(WM_RECORDSOUND_SOUNDPLAYER,(WPARAM)0, (LPARAM)
            m_pPlaySound);
        m_pRecordSound->PostThreadMessage(WM_RECORDSOUND_WRITERTHREAD ,(WPARAM)0, (LPARAM)
            m_pWriteSound);

    }



    CRecordDlg dlg;
    m_pMainWnd = &dlg;

    dlg.m_RecordThread = m_pRecordSound;
    dlg.m_PlayThread = m_pPlaySound;
    dlg.m_PlayMMSound = m_pPlayMMSound;
    //dlg.m

    int nResponse = dlg.DoModal();
    if (nResponse == IDOK)
    {
        // TODO: Place code here to handle when the dialog is
        //  dismissed with OK
    }
```

```
    else if (nResponse == IDCANCEL)
    {
        // TODO: Place code here to handle when the dialog is
        //  dismissed with Cancel
    }

    // Since the dialog has been closed, return FALSE so that we exit the
    //  application, rather than start the application's message pump.
    return FALSE;
}
BOOL CRecordApp::InitRecording()
{
    m_pRecordSound = new CRecordSound();
    m_pRecordSound->CreateThread();
    return TRUE;
}
BOOL CRecordApp::InitPlayMMSound()
{

    m_pPlayMMSound = new CPlayMMSound(0);
    m_pPlayMMSound->CreateThread();
    return TRUE;
}

BOOL CRecordApp::InitWriting()
{
    m_pWriteSound = new CWriteSoundFile(0);
    m_pWriteSound->CreateThread();
    return TRUE;
}
BOOL CRecordApp::InitPlaying()
{
    m_pPlaySound = new CPlaySound();
    m_pPlaySound->CreateThread();
    return TRUE;
}

BOOL CRecordApp::Streaming()
{


if(WSAStartup(MAKEWORD(2,2), &wsd1) !=0)
{
AfxMessageBox("ERROR");
    printf("WSAStartup() failed\n");
    return -1;
}



/************** Here was the problem last time *********************************/
/*************** Specify the local interface to listen on *********************/

local1.sin_family = AF_INET;
local1.sin_port  =  htons(multiport);
//local.sin_addr.s_addr = inet_addr("202.141.151.73");
local1.sin_addr.s_addr = htonl(INADDR_ANY);



/******************** Multicast Group address *********************************/

dwmulticastgroup1 = inet_addr(MCASTADDR);
remotel.sin_family = AF_INET;
remotel.sin_port       = htons(multiport);
remotel.sin_addr.s_addr = dwmulticastgroup1;



/************** Create a UDP Connectionless socket  ************************/

if((recvsock1 = WSASocket(AF_INET,SOCK_DGRAM,0,NULL,0,WSA_FLAG_MULTIPOINT_C_LEAF|WSA_FLAG_MULTI
POINT_D_LEAF|WSA_FLAG_OVERLAPPED)) == INVALID_SOCKET)
{
    printf("Socket failed with :%d\n", WSAGetLastError());
    AfxMessageBox("ERROR");
```

```
    WSACleanup();
    return -1;
}

if(recvsock1>0)

//printf("Socket Created Successfully.....\n");



/******************** Bind the socket to local interface ********************/

if(bind(recvsock1,(struct sockaddr *)&local1,sizeof(local1)) == SOCKET_ERROR)
{
    printf("bind() failed with:%d\n", WSAGetLastError());
    AfxMessageBox("ERROR");
    closesocket(recvsock1);
    WSACleanup();
    return -1;
}



/************ Join the Multicast group using WSAJoinLeaf() ***************************/

if((sockm1 = WSAJoinLeaf(recvsock1, (SOCKADDR *)&remote1, sizeof(remote1), NULL, NULL, NULL, NU
LL, JL_BOTH)) == INVALID_SOCKET)
{

    printf("WSAJoinLeaf() failed: %d\n", WSAGetLastError());
    AfxMessageBox("ERROR");
    closesocket(recvsock1);
    WSACleanup();
    return -1;
}
    return -1;
}
```

```
#if !defined(AFX_RECORDSOUND_H__5260C01B_03B2_11D2_A421_FC4B2C882A60__INCLUDED_)
#define AFX_RECORDSOUND_H__5260C01B_03B2_11D2_A421_FC4B2C882A60__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// RecordSound.h : header file
//

#define WM_RECORDSOUND_STARTRECORDING WM_USER+500
#define WM_RECORDSOUND_STOPRECORDING WM_USER+501
#define WM_RECORDSOUND_SOUNDPLAYER WM_USER+502
#define WM_RECORDSOUND_ENDTHREAD WM_USER+503
#define WM_RECORDSOUND_WRITERTHREAD WM_USER+504
#define WM_RECORDSOUND_SOCKET_INFO WM_USER+505


/////////////////////////////////////////////////////////////////////////////
// CRecordSound thread

#include <mmsystem.h>

#include "PlaySound.h"
#include "WriteSoundFile.h"

#define SOUNDSAMPLES 1700

class CRecordSound : public CWinThread
{
    DECLARE_DYNCREATE(CRecordSound)
public:
    CRecordSound(int iHertz=8000);           // protected constructor used by dynamic creation
    LPWAVEHDR CreateWaveHeader();
    virtual void ProcessSoundData(void* sound, DWORD dwSamples);
// Attributes
public:

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CRecordSound)
    public:
    virtual BOOL InitInstance();
    virtual int ExitInstance();
    //}}AFX_VIRTUAL

// Implementation
protected:
    virtual ~CRecordSound();

    // Generated message map functions
    //{{AFX_MSG(CRecordSound)
        // NOTE - the ClassWizard will add and remove member functions here.
    //}}AFX_MSG
    afx_msg LRESULT OnStartRecording(WPARAM wParam, LPARAM lParam);
    afx_msg LRESULT OnStopRecording(WPARAM wParam, LPARAM lParam);
    afx_msg LRESULT OnSoundData(WPARAM wParam, LPARAM lParam);
    afx_msg LRESULT OnPtrSoundPlayer(WPARAM wParam, LPARAM lParam);
    afx_msg LRESULT OnPtrSoundWriter(WPARAM wParam, LPARAM lParam);
    afx_msg LRESULT OnEndThread(WPARAM wParam, LPARAM lParam);
    afx_msg void OnSocket_Info(WPARAM wParam, LPARAM lParam);

    DECLARE_MESSAGE_MAP()
protected:
    HWAVEIN m_hRecord;
    int m_nInputBuffers;
    int m_nMaxInputBuffers;
    WAVEFORMATEX m_WaveFormatEx;
    BOOL m_bRecording;
    CPlaySound* m_Player;
    CWriteSoundFile* m_Writer;
    CWinThread *m_pSoundThread;
};
/////////////////////////////////////////////////////////////////////////////
```

```
// RecordSound.cpp : implementation file
//

#include "stdafx.h"
#include <mmsystem.h>
#include "record.h"
#include "RecordSound.h"
#include "PlaySound.h"
#include "WriteSoundFile.h"
#include "g711.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CRecordSound

IMPLEMENT_DYNCREATE(CRecordSound, CWinThread)


#define MAXINPUTBUFFERS 25

CRecordSound::CRecordSound(int iHertz)
{
    m_nInputBuffers = 0;
    m_nMaxInputBuffers = MAXINPUTBUFFERS;
    memset(&m_WaveFormatEx,0x00,sizeof(m_WaveFormatEx));
    m_WaveFormatEx.wFormatTag = WAVE_FORMAT_PCM;
    m_WaveFormatEx.nChannels = 1;
    m_WaveFormatEx.wBitsPerSample = 16;//16;
    m_WaveFormatEx.cbSize = 0;
    m_WaveFormatEx.nSamplesPerSec = 8000;//iHertz;//20050;
    m_WaveFormatEx.nAvgBytesPerSec = m_WaveFormatEx.nSamplesPerSec
        *(m_WaveFormatEx.wBitsPerSample/8);
    m_WaveFormatEx.nBlockAlign =
        (m_WaveFormatEx.wBitsPerSample/8)*
        m_WaveFormatEx.nChannels;

    m_bRecording = FALSE;
    m_Player = NULL;
    m_Writer = NULL;
}



CRecordSound::~CRecordSound()
{
}

BOOL CRecordSound::InitInstance()
{
    // TODO:  perform and per-thread initialization here
    return TRUE;
}

int CRecordSound::ExitInstance()
{
    // TODO:  perform any per-thread cleanup here
    return CWinThread::ExitInstance();
}

BEGIN_MESSAGE_MAP(CRecordSound, CWinThread)
    //{{AFX_MSG_MAP(CRecordSound)
        // NOTE - the ClassWizard will add and remove mapping macros here.
    //}}AFX_MSG_MAP
    ON_THREAD_MESSAGE(WM_RECORDSOUND_STARTRECORDING, OnStartRecording)
    ON_THREAD_MESSAGE(WM_RECORDSOUND_STOPRECORDING, OnStopRecording)
    ON_THREAD_MESSAGE(MM_WIM_DATA, OnSoundData)
    ON_THREAD_MESSAGE(WM_RECORDSOUND_SOUNDPLAYER, OnPtrSoundPlayer)
    ON_THREAD_MESSAGE(WM_RECORDSOUND_ENDTHREAD, OnEndThread)
    ON_THREAD_MESSAGE(WM_RECORDSOUND_WRITERTHREAD,OnPtrSoundWriter)
    ON_THREAD_MESSAGE(WM_RECORDSOUND_SOCKET_INFO,OnSocket_Info)
```

```
END_MESSAGE_MAP()


//////////////////////////////////////////////////////////////////////////
// CRecordSound message handlers
LRESULT CRecordSound::OnPtrSoundPlayer(WPARAM wParam, LPARAM lParam)
{
    m_Player = (CPlaySound*) lParam;

    return ERROR_SUCCESS;
}
LRESULT CRecordSound::OnStartRecording(WPARAM wParam, LPARAM lParam)
{
    DWORD mmReturn = 0;

    if(m_bRecording)
        return FALSE;

    if(!m_bRecording)
    {
        // open wavein device
        MMRESULT mmReturn = ::waveInOpen( &m_hRecord, WAVE_MAPPER,
            &m_WaveFormatEx, ::GetCurrentThreadId(), 0, CALLBACK_THREAD);
        if(mmReturn )
        {
            char errorbuffer[MAX_PATH];
            char errorbuffer1[MAX_PATH];
            waveInGetErrorText( mmReturn,
                                errorbuffer,
                                MAX_PATH);
            sprintf(errorbuffer1,"WAVEIN:%x:%s",mmReturn,errorbuffer);

            AfxMessageBox(errorbuffer1);

        }


        if(!mmReturn)
        {
            for(int i=0; i < m_nMaxInputBuffers; i++)
            {
                LPWAVEHDR lpHdr = CreateWaveHeader();
                mmReturn = ::waveInPrepareHeader(m_hRecord,lpHdr, sizeof(WAVEHDR));
                mmReturn = ::waveInAddBuffer(m_hRecord, lpHdr, sizeof(WAVEHDR));
                m_nInputBuffers++;
            }
            mmReturn = ::waveInStart(m_hRecord);
            if(mmReturn )
        {
            char errorbuffer[MAX_PATH];
            char errorbuffer1[MAX_PATH];
            waveInGetErrorText( mmReturn,
                                errorbuffer,
                                MAX_PATH);
            sprintf(errorbuffer1,"WAVEIN:%x:%s",mmReturn,errorbuffer);

            AfxMessageBox(errorbuffer1);

        }

            if(!mmReturn)
            {
                m_bRecording = TRUE;
                if(m_Player)
                    m_Player->PostThreadMessage(WM_PLAYSOUND_STARTPLAYING, 0, 0);

                if(m_Writer)
                {
                    PWRITESOUNDFILE pwsf= (PWRITESOUNDFILE) new WRITESOUNDFILE;
                    ZeroMemory(pwsf,sizeof(WRITESOUNDFILE));
                    char *p = pwsf->lpszFileName;
                    strcpy(p,"out.wav");
                    memcpy(&pwsf->waveFormatEx,&m_WaveFormatEx,sizeof(m_WaveFormatEx));
                    m_Writer->PostThreadMessage(
                    WM_WRITESOUNDFILE_FILENAME,0,(LPARAM)pwsf);
                }
```

```
                return ERROR_SUCCESS;
            }
        }


    }
    return TRUE;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
///////////////
void CRecordSound::OnSocket_Info(WPARAM wParam, LPARAM lParam)
{
/*  char value[30];
    itoa((unsigned int)lParam,value,10);
    strcat(value," Socket Number In -- Record Sound thread");
    AfxMessageBox(value);*/
    m_Writer->PostThreadMessage(WM_WRITESOUNDFILE_SOCKET_INFO,0,lParam);
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
///
LRESULT CRecordSound::OnStopRecording(WPARAM wParam, LPARAM lParam)
{
    MMRESULT mmReturn = 0;
    if(!m_bRecording)
        return FALSE;

    if(m_bRecording)
    {
        mmReturn = ::waveInStop(m_hRecord);
        if(!mmReturn)
            mmReturn = ::waveInReset(m_hRecord);
        if(!mmReturn)
            m_bRecording = FALSE;
        Sleep(500);
        if(!mmReturn)
            mmReturn = ::waveInClose(m_hRecord);
        if(m_Player)
            m_Player->PostThreadMessage(WM_PLAYSOUND_STOPPLAYING,0,0);

        if(m_Writer)
            m_Writer->PostThreadMessage(WM_WRITESOUNDFILE_CLOSEFILE,0,0);

        return mmReturn;
    }
    return TRUE;
}
LRESULT CRecordSound::OnSoundData(WPARAM wParam, LPARAM lParam)
{
    LPWAVEHDR lpHdr = (LPWAVEHDR) lParam;

    if(lpHdr)
    {
        void * lpInt = (short int*) lpHdr->lpData;
        DWORD cbRecorded = lpHdr->dwBytesRecorded;
        ::waveInUnprepareHeader(m_hRecord, lpHdr, sizeof(WAVEHDR));

// compress to Byte before send
    //  ProcessSoundData(lpInt, cbRecorded/sizeof(short int));


        if(m_Writer)

            WAVEHDR* pWriteHdr = new WAVEHDR;
            if(!pWriteHdr)
                return FALSE;
            memcpy(pWriteHdr,lpHdr,sizeof(WAVEHDR));
            BYTE * pSound = new BYTE[lpHdr->dwBufferLength];
            if(!pSound)
            {
                delete pWriteHdr;
                return FALSE;
            }
            memcpy(pSound,lpHdr->lpData,lpHdr->dwBufferLength);
            pWriteHdr->lpData = (char*)pSound;
            m_Writer->PostThreadMessage(WM_WRITESOUNDFILE_WRITEBLOCK,GetCurrentThreadId(),(LPAR
M) pWriteHdr);
        }
```

```
        if(m_Player)
        {

        //  m_Player->PostThreadMessage(WM_PLAYSOUND_PLAYBLOCK, GetCurrentThreadId(),(LPARAM) l
pHdr);
        ;
        }
        else
        {
            delete lpInt;
            delete lpHdr;
        }
        char debugbuffer[256];
        sprintf(debugbuffer, "SOUND BUFFER returned: %d\n",cbRecorded);
        TRACE(debugbuffer);
        if(m_bRecording)
        {
            LPWAVEHDR lpHdr = CreateWaveHeader();
            ::waveInPrepareHeader(m_hRecord,lpHdr, sizeof(WAVEHDR));
            ::waveInAddBuffer(m_hRecord, lpHdr, sizeof(WAVEHDR));
            m_nInputBuffers++;
        }
    }


    return ERROR_SUCCESS;
}

LPWAVEHDR CRecordSound::CreateWaveHeader()
{
    LPWAVEHDR lpHdr = new WAVEHDR;
    ZeroMemory(lpHdr, sizeof(WAVEHDR));
    BYTE* lpByte = new BYTE[(m_WaveFormatEx.nBlockAlign*SOUNDSAMPLES)];

    lpHdr->lpData = (char *) lpByte;
    lpHdr->dwBufferLength = (m_WaveFormatEx.nBlockAlign*SOUNDSAMPLES);
    return lpHdr;

}
// Compression
void CRecordSound::ProcessSoundData(void* sound , DWORD dwSamples)
{

    char* output = new char[dwSamples];

    if (COMPRESS_TYPE == "alaw"){
    alaw_compressV2(dwSamples,(short*)sound,output);
    } else {
    ulaw_compressV2(dwSamples,(short*)sound,output);
    }


    memcpy((char*)sound,output,dwSamples*sizeof(char));

    delete(output);
}

LRESULT CRecordSound::OnEndThread(WPARAM wParam, LPARAM lParam)
{
    if(m_bRecording)
    {
        OnStopRecording(0, 0);
    }
    ::PostQuitMessage(0);
    return TRUE;
}
LRESULT CRecordSound::OnPtrSoundWriter(WPARAM wParam, LPARAM lParam)
{
    m_Writer = (CWriteSoundFile*) lParam;
    return TRUE;
}
```

```
// recordDlg.h : header file
//

#if !defined(AFX_RECORDDLG_H__5260C011_03B2_11D2_A421_FC4B2C882A60__INCLUDED_)
#define AFX_RECORDDLG_H__5260C011_03B2_11D2_A421_FC4B2C882A60__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
#include "RecordSound.h"
#include "PlaySound.h"
const WM_New_Connection = WM_USER + 500;
/////////////////////////////////////////////////////////////////////////////
// CRecordDlg dialog

class CRecordDlg : public CDialog
{
// Construction
public:
    CRecordDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
    //{{AFX_DATA(CRecordDlg)
    enum { IDD = IDD_RECORD_DIALOG };
    CIPAddressCtrl  m_IP_Address;
    CButton m_Recording;
    //}}AFX_DATA
    CButton m_Train;

    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CRecordDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:
    HICON m_hIcon;

    // Generated message map functions
    //{{AFX_MSG(CRecordDlg)
    virtual BOOL OnInitDialog();
    afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
    afx_msg void OnPaint();
    afx_msg HCURSOR OnQueryDragIcon();
    afx_msg void OnButton1();
    afx_msg void OnButton2();
    afx_msg void On_Connect();
    afx_msg void On_Send_Voice();
    afx_msg void OnFieldchangedIpaddress4(NMHDR* pNMHDR, LRESULT* pResult);
    //}}AFX_MSG
    afx_msg void OnTrain();
    afx_msg void On_New_Connection(WPARAM wParam, LPARAM lParam);
    DECLARE_MESSAGE_MAP()
    BOOL m_bRecording;
public:
    CRecordSound* m_RecordThread;
    CPlaySound* m_PlayThread;
    CPlayMMSound* m_PlayMMSound;
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previo
us line.

#endif // !defined(AFX_RECORDDLG_H__5260C011_03B2_11D2_A421_FC4B2C882A60__INCLUDED_)
```

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by Record.rc
//
#define IDM_ABOUTBOX                    0x0010
#define IDD_ABOUTBOX                    100
#define IDS_ABOUTBOX                    101
#define IDD_RECORD_DIALOG               102
#define IDR_MAINFRAME                   128
#define IDD_TRAIN                       129
#define IDB_BITMAP1                     130
#define IDC_BUTTON1                     1000
#define IDC_STATIC_SILENCE              1001
#define IDC_STATIC_ZEROCROSSINGS        1002
#define IDC_TRAIN                       1003
#define IDC_UP                          1004
#define IDC_DOWN                        1005
#define IDC_BUTTON2                     1005
#define IDC_RIGHT                       1006
#define IDC_BUTTON3                     1006
#define IDC_LEFT                        1007
#define IDC_BUTTON5                     1007
#define IDC_CLICK                       1008
#define IDC_IPADDRESS1                  1008
#define IDC_STOPTRAINING                1009
#define IDC_BUTTON4                     1009
#define IDC_IPADDRESS4                  1012

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        131
#define _APS_NEXT_COMMAND_VALUE         32771
#define _APS_NEXT_CONTROL_VALUE         1013
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

```
// stdafx.h : include file for standard system include files,
//  or project specific include files that are used frequently, but
//      are changed infrequently
//

#if !defined(AFX_STDAFX_H__5260C013_03B2_11D2_A421_FC4B2C882A60__INCLUDED_)
#define AFX_STDAFX_H__5260C013_03B2_11D2_A421_FC4B2C882A60__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#define VC_EXTRALEAN        // Exclude rarely-used stuff from Windows headers

#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions
#include <afxdisp.h>        // MFC OLE automation classes
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>         // MFC support for Windows Common Controls
#endif // _AFX_NO_AFXCMN_SUPPORT
//#include <afxsock.h>  //For Socket Extensions

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previo
us line.

#endif // !defined(AFX_STDAFX_H__5260C013_03B2_11D2_A421_FC4B2C882A60__INCLUDED_)
```

```
#if !defined(AFX_WRITESOUNDFILE_H__A0EDF320_057F_11D2_A421_E60ECC112860__INCLUDED_)
#define AFX_WRITESOUNDFILE_H__A0EDF320_057F_11D2_A421_E60ECC112860__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// WriteSoundFile.h : header file
//
#include <mmsystem.h>

#define WM_WRITESOUNDFILE_FILENAME WM_USER+700
#define WM_WRITESOUNDFILE_WRITEBLOCK WM_USER+701
#define WM_WRITESOUNDFILE_CLOSEFILE WM_USER+702
#define WM_WRITESOUNDFILE_ENDTHREAD  WM_USER+703
#define WM_WRITESOUNDFILE_SOCKET_INFO  WM_USER+704
typedef struct writesoundfile_tag {
    char lpszFileName[MAX_PATH];
    WAVEFORMATEX waveFormatEx;
    TCHAR buffer[100];
} WRITESOUNDFILE, *PWRITESOUNDFILE;

/////////////////////////////////////////////////////////////////////////////
// CWriteSoundFile thread

class CWriteSoundFile : public CWinThread
{
    DECLARE_DYNCREATE(CWriteSoundFile)
public:
    CWriteSoundFile();
    CWriteSoundFile(unsigned int Recvsock);
protected:
                // protected constructor used by dynamic creation

// Attributes
public:

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CWriteSoundFile)
    public:
    virtual BOOL InitInstance();
    virtual int ExitInstance();
    //}}AFX_VIRTUAL

// Implementation
protected:
    virtual ~CWriteSoundFile();
    BOOL IsSpeech();

    // Generated message map functions
    //{{AFX_MSG(CWriteSoundFile)
        // NOTE - the ClassWizard will add and remove member functions here.
    //}}AFX_MSG
    afx_msg LRESULT CreateWaveFile(WPARAM wParam, LPARAM lParam);
    afx_msg LRESULT WriteToSoundFile(WPARAM wParam, LPARAM lParam);
    afx_msg LRESULT CloseSoundFile(WPARAM wParam, LPARAM lParam);
    afx_msg LRESULT OnEndThread(WPARAM wParam, LPARAM lParam);
    afx_msg void OnSocket_Info(WPARAM wParam, LPARAM lParam);

    DECLARE_MESSAGE_MAP()

    HMMIO m_hFile;
    MMCKINFO m_MMCKInfoData;
    MMCKINFO m_MMCKInfoParent;
    MMCKINFO m_MMCKInfoChild;
protected:

};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previo
us line.
```

```
// WriteSoundFile.cpp : implementation file
//

#include "stdafx.h"
#include<winsock2.h>
#include <mmsystem.h>
#include "record.h"
#include "WriteSoundFile.h"
#include<stdlib.h>
#include "g711.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
#define SENDDEFINE 1504

/*#define MCASTADDR      "234.7.7.7"
#define MCASTPORT      27000
#define BUFSIZE        800
#define DEFAULT_COUNT  500*/


//BOOL bloopback = FALSE;
//int multiport = MCASTPORT;

DWORD   dwinterface2 ,dwmulticastgroup2,dwcount2;


WSADATA             wsd2;
SOCKET              sockm2,recvsock2;

DWORD               i2=0;

struct              sockaddr_in local2,remote2,from2,server2;
struct              hostent *host2 = NULL;

char                buffer2[100];
char                compressedData[SENDDEFINE];
int                 len2 = sizeof(struct sockaddr_in),optval2,ret2,t2;
int                 localport2 = 7000;


//////////////////////////////////////////////////////////////////////////

void Send_Audio_Data(char *recvbuf);
void CompressSoundData(short *input,char *output,DWORD dwSamples);
//int Streaming();
//////////////////////////////////////////////////////////////////////////
// CWriteSoundFile

IMPLEMENT_DYNCREATE(CWriteSoundFile, CWinThread)

CWriteSoundFile::CWriteSoundFile()
{
//  Streaming();
}
CWriteSoundFile::CWriteSoundFile(unsigned int Recvsock)
{
//  recvsock2 = Recvsock;
//  Streaming();
}
CWriteSoundFile::~CWriteSoundFile()
{
}


BOOL CWriteSoundFile::InitInstance()
{
    // TODO:  perform and per-thread initialization here
    return TRUE;
}

int CWriteSoundFile::ExitInstance()
{
    // TODO:  perform any per-thread cleanup here
    return CWinThread::ExitInstance();
}
```

```
BEGIN_MESSAGE_MAP(CWriteSoundFile, CWinThread)
    //{{AFX_MSG_MAP(CWriteSoundFile)
        // NOTE - the ClassWizard will add and remove mapping macros here.
    //}}AFX_MSG_MAP
    ON_THREAD_MESSAGE(WM_WRITESOUNDFILE_FILENAME, CreateWaveFile)
    ON_THREAD_MESSAGE(WM_WRITESOUNDFILE_WRITEBLOCK, WriteToSoundFile)
    ON_THREAD_MESSAGE(WM_WRITESOUNDFILE_CLOSEFILE, CloseSoundFile)
    ON_THREAD_MESSAGE(WM_RECORDSOUND_ENDTHREAD, OnEndThread)
    ON_THREAD_MESSAGE(WM_WRITESOUNDFILE_SOCKET_INFO, OnSocket_Info)

END_MESSAGE_MAP()

LRESULT CWriteSoundFile::CreateWaveFile(WPARAM wParam, LPARAM lParam)
{
    PWRITESOUNDFILE pWriteSoundFile = (PWRITESOUNDFILE) lParam;
    int cbWaveFormatEx = sizeof(WAVEFORMATEX) + pWriteSoundFile->waveFormatEx.cbSize;

    m_hFile = ::mmioOpen(pWriteSoundFile->lpszFileName,NULL, MMIO_CREATE|MMIO_WRITE|MMIO_EXCLUS
IVE | MMIO_ALLOCBUF);
    if(!m_hFile)
        return TRUE;

    ZeroMemory(&m_MMCKInfoParent, sizeof(MMCKINFO));
    m_MMCKInfoParent.fccType = mmioFOURCC('W','A','V','E');

    MMRESULT mmResult =  ::mmioCreateChunk( m_hFile,&m_MMCKInfoParent,
                         MMIO_CREATERIFF);

    ZeroMemory(&m_MMCKInfoChild, sizeof(MMCKINFO));
    m_MMCKInfoChild.ckid = mmioFOURCC('f','m','t',' ');
    m_MMCKInfoChild.cksize = cbWaveFormatEx;
    mmResult = ::mmioCreateChunk(m_hFile, &m_MMCKInfoChild, 0);
    mmResult = ::mmioWrite(m_hFile, (char*)&pWriteSoundFile->waveFormatEx, cbWaveFormatEx);
    mmResult = ::mmioAscend(m_hFile, &m_MMCKInfoChild, 0);
    m_MMCKInfoChild.ckid = mmioFOURCC('d', 'a', 't', 'a');
    mmResult = ::mmioCreateChunk(m_hFile, &m_MMCKInfoChild, 0);

    return ERROR_SUCCESS;

}
void CWriteSoundFile::OnSocket_Info(WPARAM wParam, LPARAM lParam)
{

        recvsock2 = unsigned int(lParam);
        /*char value[30];
    itoa(recvsock2,value,10);
    strcat(value," Socket Number In -- Write Sound  File thread");
    AfxMessageBox(value);*/
    /*CloseSoundFile(0,0);
    ::PostQuitMessage(0);*/

}
LRESULT CWriteSoundFile::WriteToSoundFile(WPARAM wParam, LPARAM lParam)
{
    LPWAVEHDR lpHdr = (LPWAVEHDR) lParam;
    int cbLength = lpHdr->dwBufferLength;
//  char len[40];
//  itoa(cbLength,len,10);
    //HWND h1 = GetSafeHwnd
//  if(cbLength==1000)
//AfxMessageBox(len);
    //else AfxMessageBox("NO");
//  char *soundbuffer = (char*) lpHdr->lpData;

    short *soundbuffer = (short*) lpHdr->lpData;
/************************* Process sound **********************************/

    //  REAL 1000 Samples
    DWORD dwSamples = lpHdr->dwBufferLength/sizeof(short); //m_WaveFormatEx.nBlockAlign);

    CompressSoundData(soundbuffer, compressedData , SENDDEFINE);


/*****************************************************************/
    /////char sendbuff[800];

    /////strncpy(sendbuff,soundbuffer,800);
//      char len[40];
```

```
//  itoa(sizeof(sendbuff),len,10);
    //HWND h1 = GetSafeHwnd
//  if(cbLength==1000)
//AfxMessageBox(sendbuff);

//  AfxMessageBox(sendbuff);
        //Send_Audio_Data(sendbuff);
        Send_Audio_Data(compressedData);
/*  if(lpHdr)
    {
        char *soundbuffer = (char*) lpHdr->lpData;
        if(m_hFile && soundbuffer)
            ::mmioWrite(m_hFile, soundbuffer, cbLength);
        if(soundbuffer)
            delete (BYTE*) soundbuffer;
        if(lpHdr)
            delete lpHdr;
    }*/

    return ERROR_SUCCESS;
}
LRESULT CWriteSoundFile::CloseSoundFile(WPARAM wParam, LPARAM lParam)
{

    if(m_hFile)
    {
        ::mmioAscend(m_hFile, &m_MMCKInfoChild, 0);
        ::mmioAscend(m_hFile, &m_MMCKInfoParent, 0);
        ::mmioClose(m_hFile, 0);
        m_hFile = NULL;
    }

    return ERROR_SUCCESS;
}
LRESULT CWriteSoundFile::OnEndThread(WPARAM wParam, LPARAM lParam)
{
    CloseSoundFile(0,0);
    ::PostQuitMessage(0);
    return TRUE;
}


void Send_Audio_Data(char *recvbuf)
{
//  char recvbuf[800];
    //strcpy(recvbuf1,recvbuf);
    //char temp;
    char* temp1;
    temp1= recvbuf;

    int k=sizeof(recvbuf);
    //AfxMessageBox(temp1);

    //strcpy(temp,temp1);

    //(struct sockaddr *)&from2 , 0)
        if((ret2 = send(recvsock2, recvbuf, SENDDEFINE, 0 ))== SOCKET_ERROR)
        {
        /*  char value[30];
        //  itoa(WSAGetLastError(),value,10);
            itoa(recvsock2,value,10);

            strcat(value," Send failed with: ");
            AfxMessageBox(value);*/

              char value[30];
   itoa(recvsock2,value,10);
   strcat(value," Socket Number In -- Write Sound  File thread");
   AfxMessageBox(value);


          // closesocket(sockm2);
       //  closesocket(recvsock2);
            WSACleanup();
            //return -1;
        }

    /*char value[30];
```

```
    itoa(sizeof(recvbuf),value,10);
    AfxMessageBox(value);*/

}


// Compress Data
void CompressSoundData(short *input,char *output,DWORD dwSamples)
{

    if (COMPRESS_TYPE == "alaw"){
    alaw_compressV2(dwSamples,(short*)input,output);
    } else {
    ulaw_compressV2(dwSamples,(short*)input,output);
    }


}
```

```
#ifndef G711_defined
#define G711_defined 301
/*
/* Smart function prototypes: for [ag]cc, VaxC, and [tb]cc
#if !defined(ARGS)
#if (defined(__STDC__) || defined(VMS) || defined(__DECC)  || defined(MSDOS) || defined(__MSDOS
__)) || defined (__CYGWIN__) || defined (_MSC_VER)
#define ARGS(s) s
#else
#define ARGS(s) ()
#endif
#endif

*/
/* Function prototypes */
void  alaw_compress(long lseg, short *linbuf, short *logbuf);
void  alaw_expand(long lseg, short *logbuf, short *linbuf);
void  ulaw_compress(long lseg, short *linbuf, short *logbuf);
void  ulaw_expand(long lseg, short *logbuf, short *linbuf);
void  alaw_compressV2(long lseg, short *linbuf, char *logbuf);
void  alaw_expandV2(long lseg, short *logbuf, char *linbuf);
void  ulaw_compressV2(long lseg, short *linbuf, char *logbuf);
void  ulaw_expandV2(long lseg, char *logbuf, short *linbuf);
/* Definitions for better user interface (?!) */
#define IS_LIN 1
#define IS_LOG 0
#define COMPRESS_TYPE "ulaw"
#endif
/* ......................... End of G711.H ......................... */
```

```
#include "stdafx.h"
#include "g711.h"
#include <stdlib.h>
#include <string.h>
#include <iostream.h>
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif


void            alaw_compress(
  long            lseg,
  short          *linbuf,
  short          *logbuf)
{
  short           ix, iexp;
  long            n;

  for (n = 0; n < lseg; n++)
  {
    ix = linbuf[n] < 0       /* 0 <= ix < 2048 */
      ? (~linbuf[n]) >> 4    /* 1's complement for negative values */
      : (linbuf[n]) >> 4;

    /* Do more, if exponent > 0 */
    if (ix > 15)           /* exponent=0 for ix <= 15 */
    {
      iexp = 1;            /* first step: */
      while (ix > 16 + 15)  /* find mantissa and exponent */
      {
    ix >>= 1;
    iexp++;
      }
      ix -= 16;            /* second step: remove leading '1' */

      ix += iexp << 4;     /* now compute encoded value */
    }
    if (linbuf[n] >= 0)
      ix |= (0x0080);      /* add sign bit */

    logbuf[n] = ix ^ (0x0055);  /* toggle even bits */
  }
}
/* ................... End of alaw_compress() ..................... */


void            alaw_expand(
  long            lseg,
  short          *linbuf,
  short          *logbuf)
{
  short           ix, mant, iexp;
  long            n;

  for (n = 0; n < lseg; n++)
  {
    ix = logbuf[n] ^ (0x0055);  /* re-toggle toggled bits */

    ix &= (0x007F);      /* remove sign bit */
    iexp = ix >> 4;      /* extract exponent */
    mant = ix & (0x000F);    /* now get mantissa */
    if (iexp > 0)
      mant = mant + 16;      /* add leading '1', if exponent > 0 */

    mant = (mant << 4) + (0x0008);  /* now mantissa left justified and */
    /* 1/2 quantization step added */
    if (iexp > 1)        /* now left shift according exponent */
      mant = mant << (iexp - 1);

    linbuf[n] = logbuf[n] > 127 /* invert, if negative sample */
      ? mant
      : -mant;
  }
}
/* ..... End of alaw_expand() ..... */
```

```
void            ulaw_compress(
  long            lseg,
  short          *linbuf,
  short          *logbuf)
{
  long            n;            /* samples's count */
  short           i;        /* aux.var. */
  short           absno;    /* absolute value of linear (input) sample */
  short           segno;    /* segment (Table 2/G711, column 1) */
  short           low_nibble;   /* low  nibble of log companded sample */
  short           high_nibble;  /* high nibble of log companded sample */

  for (n = 0; n < lseg; n++)
  {
    /* ---------------------------------------------------------------------- */
    /* Change from 14 bit left justified to 14 bit right justified */
    /* Compute absolute value; adjust for easy processing */
    /* ---------------------------------------------------------------------- */
    absno = linbuf[n] < 0   /* compute 1's complement in case of  */
      ? ((~linbuf[n]) >> 2) + 33/* negative samples */
      : ((linbuf[n]) >> 2) + 33;/* NB: 33 is the difference value */
    /* between the thresholds for */
    /* A-law and u-law. */
    if (absno > (0x1FFF))   /* limitation to "absno" < 8192 */
      absno = (0x1FFF);

    /* Determination of sample's segment */
    i = absno >> 6;
    segno = 1;
    while (i != 0)
    {
      segno++;
      i >>= 1;
    }

    /* Mounting the high-nibble of the log-PCM sample */
    high_nibble = (0x0008) - segno;

    /* Mounting the low-nibble of the log PCM sample */
    low_nibble = (absno >> segno)   /* right shift of mantissa and */
      & (0x000F);       /* masking away leading '1' */
    low_nibble = (0x000F) - low_nibble;

    /* Joining the high-nibble and the low-nibble of the log PCM sample */
    logbuf[n] = (high_nibble << 4) | low_nibble;

    /* Add sign bit */
    if (linbuf[n] >= 0)
      logbuf[n] = logbuf[n] | (0x0080);
  }
}
/* ..................... End of ulaw_compress() ..................... */


void            ulaw_expand( long lseg,
  short          *logbuf,
  short          *linbuf)
{
  long            n;        /* aux.var. */
  short           segment;  /* segment (Table 2/G711, column 1) */
  short           mantissa; /* low  nibble of log companded sample */
  short           exponent; /* high nibble of log companded sample */
  short           sign;     /* sign of output sample */
  short           step;

  for (n = 0; n < lseg; n++)
  {
    sign = logbuf[n] < (0x0080) /* sign-bit = 1 for positiv values */
      ? -1
      : 1;
    mantissa = ~logbuf[n];  /* 1's complement of input value */
    exponent = (mantissa >> 4) & (0x0007);  /* extract exponent */
    segment = exponent + 1; /* compute segment number */
    mantissa = mantissa & (0x000F); /* extract mantissa */

    /* Compute Quantized Sample (14 bit left justified!) */
```

```
      step = (4) << segment;  /* position of the LSB */
      /* = 1 quantization step) */
      linbuf[n] = sign *       /* sign */
        (((0x0080) << exponent)   /* '1', preceding the mantissa */
         + step * mantissa    /* left shift of mantissa */
         + step / 2       /* 1/2 quantization step */
         - 4 * 33
        );
    }
  }

/********************************** Editttt ********************************/

void            alaw_compressV2(
  long            lseg,
  short          *linbuf,
  char           *logbuf)
{
  short           ix, iexp;
  long            n;

  for (n = 0; n < lseg; n++)
  {
    ix = linbuf[n] < 0       /* 0 <= ix < 2048 */
      ? (~linbuf[n]) >> 4    /* 1's complement for negative values */
      : (linbuf[n]) >> 4;

    /* Do more, if exponent > 0 */
    if (ix > 15)             /* exponent=0 for ix <= 15 */
    {
      iexp = 1;              /* first step: */
      while (ix > 16 + 15)   /* find mantissa and exponent */
      {
    ix >>= 1;
    iexp++;
      }
      ix -= 16;              /* second step: remove leading '1' */

      ix += iexp << 4;       /* now compute encoded value */
    }
    if (linbuf[n] >= 0)
      ix |= (0x0080);        /* add sign bit */

    logbuf[n] = (char)(ix ^ (0x0055));  /* toggle even bits */
  }
}
/* ................... End of alaw_compress() ..................... */


void            alaw_expandV2(
  long            lseg,
  short          *linbuf,
  char           *logbuf)
{
  short           ix, mant, iexp;
  long            n;

  for (n = 0; n < lseg; n++)
  {
    ix = (short)(logbuf[n] ^ (0x55));   /* re-toggle toggled bits */

    ix &= (0x007F);          /* remove sign bit */
    iexp = ix >> 4;          /* extract exponent */
    mant = ix & (0x000F);    /* now get mantissa */
    if (iexp > 0)
      mant = mant + 16;      /* add leading '1', if exponent > 0 */

    mant = (mant << 4) + (0x0008);  /* now mantissa left justified and */
    /* 1/2 quantization step added */
    if (iexp > 1)            /* now left shift according exponent */
      mant = mant << (iexp - 1);

    linbuf[n] = logbuf[n]    /* invert, if negative sample */
      ? mant
      : -mant;
  }
}
/* ..... End of alaw_expand() ..... */
```

```
void            ulaw_compressV2(
  long            lseg,
  short          *linbuf,
  char           *logbuf)
{
  long             n;             /* samples's count */
  short            i;         /* aux.var. */
  short            absno;     /* absolute value of linear (input) sample */
  short            segno;     /* segment (Table 2/G711, column 1) */
  short            low_nibble;    /* low  nibble of log companded sample */
  short            high_nibble;   /* high nibble of log companded sample */

  for (n = 0; n < lseg; n++)
  {
    /* ---------------------------------------------------------------------- */
    /* Change from 14 bit left justified to 14 bit right justified */
    /* Compute absolute value; adjust for easy processing */
    /* ---------------------------------------------------------------------- */
    absno = linbuf[n] < 0   /* compute 1's complement in case of  */
      ? ((~linbuf[n]) >> 2) + 33/* negative samples */
      : ((linbuf[n]) >> 2) + 33;/* NB: 33 is the difference value */
    /* between the thresholds for */
    /* A-law and u-law. */
    if (absno > (0x1FFF))    /* limitation to "absno" < 8192 */
      absno = (0x1FFF);

    /* Determination of sample's segment */
    i = absno >> 6;
    segno = 1;
    while (i != 0)
    {
      segno++;
      i >>= 1;
    }

    /* Mounting the high-nibble of the log-PCM sample */
    high_nibble = (0x0008) - segno;

    /* Mounting the low-nibble of the log PCM sample */
    low_nibble = (absno >> segno)   /* right shift of mantissa and */
      & (0x000F);       /* masking away leading '1' */
    low_nibble = (0x000F) - low_nibble;

    /* Joining the high-nibble and the low-nibble of the log PCM sample */
    logbuf[n] = (char)(high_nibble << 4) | low_nibble;

    /* Add sign bit */
    if (linbuf[n] >= 0)
      logbuf[n] = logbuf[n] | (0x80);
  }
}
/* ..................... End of ulaw_compress() ..................... */


void            ulaw_expandV2( long lseg,
  char           *logbuf1,
  short          *linbuf)
{
  long             n;         /* aux.var. */
  short            segment;   /* segment (Table 2/G711, column 1) */
  short            mantissa;  /* low  nibble of log companded sample */
  short            exponent;  /* high nibble of log companded sample */
  short            sign;      /* sign of output sample */
  short            step;
  short            logbuf;
  for (n = 0; n < lseg; n++)
  {
    logbuf = logbuf1[n];
    logbuf = logbuf &(0x00FF) ;
    sign = logbuf < (0x0080)     /* sign-bit = 1 for positiv values */
      ? -1
      : 1;
    mantissa = ~(logbuf);    /* 1's complement of input value */
    mantissa = mantissa&(0x00FF) ;
    exponent = (mantissa >> 4) & (0x0007);  /* extract exponent */
    segment = exponent + 1;  /* compute segment number */
```

```
    mantissa = mantissa & (0x000F); /* extract mantissa */

    /* Compute Quantized Sample (14 bit left justified!) */
    step = (4) << segment;  /* position of the LSB */
    /* = 1 quantization step) */
    linbuf[n] = sign *      /* sign */
      (((0x0080) << exponent)   /* '1', preceding the mantissa */
       + step * mantissa    /* left shift of mantissa */
       + step / 2       /* 1/2 quantization step */
       - 4 * 33
      );

  }
 }
 /*
 void main(){

 short a[10] = {1000,-30000,40000,4,100,6,-700,8000,-19000,10};
 short b[10];
 short c[10];

 char e[10];
 short cc[10];

 alaw_compress(10,a,b);
 alaw_expand(10,c,b);

 alaw_compressV2(10,a,e);
 alaw_expandV2(10,cc,e);

 ulaw_compress(10,a,b);
 ulaw_expand(10,b,c);

 ulaw_compressV2(10,a,e);
 ulaw_expandV2(10,e,cc);
 cout << "sdfsdf" ;

 }*/
```

```
#include <stdlib.h>
#include <iostream.h>
#include <stdio.h>
#include "struct.h"
#include "ADPCM.h"
void main()
{
    char FileName[25];

    printf("ADPCM Encoder\n\n");

    //Read Wave File
    printf("Enter wave file name : ");
    gets(FileName);
    ReadWave(FileName);

    //Encode
    ADPCMCoder(PCM,ADPCM,DataSize/2);

    //Write ADPCM File
    printf("Enter ADPCM file name : ");
    gets(FileName);
    WriteADPCM(FileName);
}
```

```
#include <stdio.h>
#include <iostream.h>

#define msPerFrame 20
#define SamplingRate 8000
#define SamplesPerOneFrame SamplingRate*msPerFrame/1000

FILE *ReadPointer,*WritePointer;

char RIFF[4];           //"RIFF"
long FileSize;          //total length of file - 8
char WAVE[4];           //designates as "WAVE" file

//****fmt ****
char fmt[4];                //type of chunk "fmt "
long WaveFormatSize;        //length in bytes
char WaveFormatInfo[40];    //data format
//************

//****fact****
char fact[4];               //type of chunk "fact"
long factSize;              //length in bytes
char factInfo[40];          //number of samples
//************

//****data****
char data[4];                   //"data"
long DataSize;                  //Size of PCM
short PCM[160000];
char ADPCM[80000];
//************

//for encode
void ReadWave(char FileName[25]);
void WriteADPCM(char FileName[25]);

//for decode
void ReadADPCM(char FileName[25]);
void WriteWave(char FileName[25]);

void ReadWave(char FileName[25])
{
    if((ReadPointer=fopen(FileName,"rb"))!=NULL)
    {
        fread(&RIFF,sizeof(char),4,ReadPointer);
        fread(&FileSize,sizeof(long),1,ReadPointer);
        fread(&WAVE,sizeof(char),4,ReadPointer);

        fread(&fmt,sizeof(char),4,ReadPointer);
        fread(&WaveFormatSize,sizeof(long),1,ReadPointer);
        fread(&WaveFormatInfo,sizeof(char),WaveFormatSize,ReadPointer);

        fread(&fact,sizeof(char),4,ReadPointer);
        fread(&factSize,sizeof(long),1,ReadPointer);
        fread(&factInfo,sizeof(char),factSize,ReadPointer);

        fread(&data,sizeof(char),4,ReadPointer);
        fread(&DataSize,sizeof(long),1,ReadPointer);
        fread(&PCM,sizeof(short),DataSize/2,ReadPointer);

        fclose(ReadPointer);
    }
    else printf("Error opening wave file to read.\n");
}

void WriteADPCM(char FileName[25])
{
    if((WritePointer=fopen(FileName,"wb"))!=NULL)
    {
        fwrite(&RIFF,sizeof(char),4,ReadPointer);
        fwrite(&FileSize,sizeof(long),1,ReadPointer);
        fwrite(&WAVE,sizeof(char),4,ReadPointer);

        fwrite(&fmt,sizeof(char),4,ReadPointer);
        fwrite(&WaveFormatSize,sizeof(long),1,ReadPointer);
        fwrite(&WaveFormatInfo,sizeof(char),WaveFormatSize,ReadPointer);

        fwrite(&fact,sizeof(char),4,ReadPointer);
```

```
        fwrite(&factSize,sizeof(long),1,ReadPointer);
        fwrite(&factInfo,sizeof(char),factSize,ReadPointer);

        fwrite(&data,sizeof(char),4,ReadPointer);
        fwrite(&DataSize,sizeof(long),1,ReadPointer);
        fwrite(&ADPCM,sizeof(char),DataSize/4,ReadPointer);

        fclose(WritePointer);
    }
    else printf("Error opening ADPCM file to write.\n");
}

void ReadADPCM(char FileName[25])
{
    if((ReadPointer=fopen(FileName,"rb"))!=NULL)
    {
        fread(&RIFF,sizeof(char),4,ReadPointer);
        fread(&FileSize,sizeof(long),1,ReadPointer);
        fread(&WAVE,sizeof(char),4,ReadPointer);

        fread(&fmt,sizeof(char),4,ReadPointer);
        fread(&WaveFormatSize,sizeof(long),1,ReadPointer);
        fread(&WaveFormatInfo,sizeof(char),WaveFormatSize,ReadPointer);

        fread(&fact,sizeof(char),4,ReadPointer);
        fread(&factSize,sizeof(long),1,ReadPointer);
        fread(&factInfo,sizeof(char),factSize,ReadPointer);

        fread(&data,sizeof(char),4,ReadPointer);
        fread(&DataSize,sizeof(long),1,ReadPointer);
        fread(&ADPCM,sizeof(char),DataSize/4,ReadPointer);

        fclose(ReadPointer);
    }
    else printf("Error opening ADPCM file to read.\n");
}

void WriteWave(char FileName[25])
{
    if((WritePointer=fopen(FileName,"wb"))!=NULL)
    {
        fwrite(&RIFF,sizeof(char),4,ReadPointer);
        fwrite(&FileSize,sizeof(long),1,ReadPointer);
        fwrite(&WAVE,sizeof(char),4,ReadPointer);

        fwrite(&fmt,sizeof(char),4,ReadPointer);
        fwrite(&WaveFormatSize,sizeof(long),1,ReadPointer);
        fwrite(&WaveFormatInfo,sizeof(char),WaveFormatSize,ReadPointer);

        fwrite(&fact,sizeof(char),4,ReadPointer);
        fwrite(&factSize,sizeof(long),1,ReadPointer);
        fwrite(&factInfo,sizeof(char),factSize,ReadPointer);

        fwrite(&data,sizeof(char),4,ReadPointer);
        fwrite(&DataSize,sizeof(long),1,ReadPointer);
        fwrite(&PCM,sizeof(short),DataSize/2,ReadPointer);

        fclose(WritePointer);
    }
    else printf("Error opening wave file to write.\n");
}
```

```
#include <stdlib.h>
#include <iostream.h>
#include <stdio.h>
#include "ADPCM.h"
#include "struct.h"

void main()
{
    char FileName[15];

    printf("ADPCM Decoder\n\n");

    //Read ADPCM File
    printf("Enter ADPCM file name : ");
    gets(FileName);
    ReadADPCM(FileName);

    //Decode
    ADPCMDecoder(ADPCM,PCM,DataSize/2);

    //Write Wave File
    printf("Enter wave file name : ");
    gets(FileName);
    WriteWave(FileName);
}
```

```
#include <stdio.h>
#include <iostream.h>

#define msPerFrame 20
#define SamplingRate 8000
#define SamplesPerOneFrame SamplingRate*msPerFrame/1000

FILE *ReadPointer,*WritePointer;

char RIFF[4];           //"RIFF"
long FileSize;          //total length of file - 8
char WAVE[4];           //designates as "WAVE" file

//****fmt ****
char fmt[4];                //type of chunk "fmt "
long WaveFormatSize;        //length in bytes
char WaveFormatInfo[40];    //data format
//*************

//****fact****
char fact[4];               //type of chunk "fact"
long factSize;              //length in bytes
char factInfo[40];          //number of samples
//*************

//****data****
char data[4];                   //"data"
long DataSize;                  //Size of PCM
short PCM[160000];
char ADPCM[80000];
//*************

//for encode
void ReadWave(char FileName[25]);
void WriteADPCM(char FileName[25]);

//for decode
void ReadADPCM(char FileName[25]);
void WriteWave(char FileName[25]);

void ReadWave(char FileName[25])
{
    if((ReadPointer=fopen(FileName,"rb"))!=NULL)
    {
        fread(&RIFF,sizeof(char),4,ReadPointer);
        fread(&FileSize,sizeof(long),1,ReadPointer);
        fread(&WAVE,sizeof(char),4,ReadPointer);

        fread(&fmt,sizeof(char),4,ReadPointer);
        fread(&WaveFormatSize,sizeof(long),1,ReadPointer);
        fread(&WaveFormatInfo,sizeof(char),WaveFormatSize,ReadPointer);

        fread(&fact,sizeof(char),4,ReadPointer);
        fread(&factSize,sizeof(long),1,ReadPointer);
        fread(&factInfo,sizeof(char),factSize,ReadPointer);

        fread(&data,sizeof(char),4,ReadPointer);
        fread(&DataSize,sizeof(long),1,ReadPointer);
        fread(&PCM,sizeof(short),DataSize/2,ReadPointer);

        fclose(ReadPointer);
    }
    else printf("Error opening wave file to read.\n");
}

void WriteADPCM(char FileName[25])
{
    if((WritePointer=fopen(FileName,"wb"))!=NULL)
    {
        fwrite(&RIFF,sizeof(char),4,ReadPointer);
        fwrite(&FileSize,sizeof(long),1,ReadPointer);
        fwrite(&WAVE,sizeof(char),4,ReadPointer);

        fwrite(&fmt,sizeof(char),4,ReadPointer);
        fwrite(&WaveFormatSize,sizeof(long),1,ReadPointer);
        fwrite(&WaveFormatInfo,sizeof(char),WaveFormatSize,ReadPointer);
        fwrite(&fact,sizeof(char),4,ReadPointer);
```

```
    fwrite(&factSize,sizeof(long),1,ReadPointer);
    fwrite(&factInfo,sizeof(char),factSize,ReadPointer);

    fwrite(&data,sizeof(char),4,ReadPointer);
    fwrite(&DataSize,sizeof(long),1,ReadPointer);
    fwrite(&ADPCM,sizeof(char),DataSize/4,ReadPointer);

    fclose(WritePointer);
}
else printf("Error opening ADPCM file to write.\n");


d ReadADPCM(char FileName[25])

if((ReadPointer=fopen(FileName,"rb"))!=NULL)
{
    fread(&RIFF,sizeof(char),4,ReadPointer);
    fread(&FileSize,sizeof(long),1,ReadPointer);
    fread(&WAVE,sizeof(char),4,ReadPointer);

    fread(&fmt,sizeof(char),4,ReadPointer);
    fread(&WaveFormatSize,sizeof(long),1,ReadPointer);
    fread(&WaveFormatInfo,sizeof(char),WaveFormatSize,ReadPointer);

    fread(&fact,sizeof(char),4,ReadPointer);
    fread(&factSize,sizeof(long),1,ReadPointer);
    fread(&factInfo,sizeof(char),factSize,ReadPointer);

    fread(&data,sizeof(char),4,ReadPointer);
    fread(&DataSize,sizeof(long),1,ReadPointer);
    fread(&ADPCM,sizeof(char),DataSize/4,ReadPointer);

    fclose(ReadPointer);
}
else printf("Error opening ADPCM file to read.\n");


d WriteWave(char FileName[25])

if((WritePointer=fopen(FileName,"wb"))!=NULL)
{
    fwrite(&RIFF,sizeof(char),4,ReadPointer);
    fwrite(&FileSize,sizeof(long),1,ReadPointer);
    fwrite(&WAVE,sizeof(char),4,ReadPointer);

    fwrite(&fmt,sizeof(char),4,ReadPointer);
    fwrite(&WaveFormatSize,sizeof(long),1,ReadPointer);
    fwrite(&WaveFormatInfo,sizeof(char),WaveFormatSize,ReadPointer);

    fwrite(&fact,sizeof(char),4,ReadPointer);
    fwrite(&factSize,sizeof(long),1,ReadPointer);
    fwrite(&factInfo,sizeof(char),factSize,ReadPointer);

    fwrite(&data,sizeof(char),4,ReadPointer);
    fwrite(&DataSize,sizeof(long),1,ReadPointer);
    fwrite(&PCM,sizeof(short),DataSize/2,ReadPointer);

    fclose(WritePointer);
}
else printf("Error opening wave file to write.\n");
```