โทรศัพท์ผ่านอินเตอร์เน็ต

INTERNET PHONE

โดย

นายมนัสนันท์ น้อมภักดี

นางสาวเมธินี มานะทวีวัฒน์

นายรุ่งธวัช สุวรรณปรีชา

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2547

ภาควิชา
วิศวกรรมโทรคมนาคม

โทรศัพท์ผ่านอินเตอร์เน็ต

INTERNET PHONE

โดย

นายมนัสนันท์ น้อมภักดี 44010371

นางสาวเมธินี มานะทวีวัฒน์ 44010382

นายรุ่งธวัช สุวรรณปรีชา 44010400

อาจารย์ที่ปรึกษา

รศ.นิภา ลีลารุจิ

ผศ.ดร. สุทธิชัย นพนาคีพงษ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2547

ปริญญานิพนธ์ปีการศึกษา 2547

ภาควิชาโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง **โทรศัพท์ผ่านอินเตอร์เน็ต**

INTERNET PHONE

ผู้จัดทำ

1. นายมนัสนันท์ น้อมภักดี 44010371

2. นางสาวเมธินี มานะทวีวัฒน์ 44010382

3. นายรุ่งธวัช สุวรรณปรีชา 44010400

อาจารย์ที่ปรึกษา

(ผศ.ดร. สุทธิชัย นพนาคีพงษ์)

อาจารย์ที่ปรึกษา

(รศ. นิภา ลีลารุจิ)

โทรศัพท์ผ่านอินเตอร์เน็ต

INTERNET PHONE

โดย นายมนัสนันท์ น้อมภักดี 44010371
นางสาวเมธินี มานะทวีวัฒน์ 44010382
นายรุ่งธวัช สุวรรณปรีชา 44010400

อาจารย์ที่ปรึกษา รศ.นิภา ลีลารุจิ
ผศ.ดร.สุทธิชัย นพนาคีพงษ์

บทคัดย่อ

ในปัจจุบันการสื่อสารได้เข้ามามีส่วนเกี่ยวข้องต่อวิถีชีวิตของมนุษย์เป็นอย่างมากและได้มีการพัฒนาอย่างต่อเนื่องเพื่อความสะดวกและรวดเร็วของผู้ใช้งาน การสื่อสารที่นับได้ว่ามีความสะดวกและรวดเร็วอย่างหนึ่งก็คือการติดต่อสื่อสารผ่านทางโทรศัพท์ แต่การสื่อสารผ่านทางโทรศัพท์ที่มีระยะทางไกลจะมีค่าบริการค่อนข้างสูงอินเตอร์เน็ตจึงได้เข้ามามีบทบาทเนื่องจากมีค่าใช้จ่ายที่ต่ำกว่า แต่ก็ยังให้ความสะดวกแก่ผู้ใช้งานได้ไม่เท่ากับโทรศัพท์

โครงงานนี้จึงได้ทำการศึกษาถึงอุปกรณ์ที่ช่วยลดค่าใช้จ่ายทางโทรศัพท์ โดยทำการสื่อสารผ่านทางอินเตอร์เน็ตแทน หลักการทำงานของโครงงานนี้คือจะทำการแปลงสัญญาณเสียงของมนุษย์ซึ่งเป็นสัญญาณอนาล็อกให้เป็นสัญญาณดิจิตอล โดยมีคอมพิวเตอร์เป็นสื่อกลางในการส่งผ่านข้อมูลไปยังเครื่องคอมพิวเตอร์อื่นๆ ในเครือข่าย โดยมีไอพี แอดเดรส(IP address: Internet Protocol address) เป็นตัวระบุเป้าหมายที่ต้องการติดต่อสื่อสารด้วย และทำการแปลงสัญญาณดิจิตอลกลับมาเป็นสัญญาณอนาล็อกในทางรับ ทำให้สามารถลดค่าใช้จ่ายในการติดต่อสื่อสารในระยะทางไกลๆได้ และมีความสะดวกในการใช้งานใกล้เคียงกับโทรศัพท์ทั่วๆไป

Abstract

Nowadays, communication is necessary for human life. Telephone is a pattern of all, which is in common society. But international communication still has a problem of too expensive fee that is main obstruct for communication. So we choose voice communication via Internet instead of old telephone. As it makes ours reduce the fee that become the main key issue. In this scheme we will study about the method to send the data packet that is converted from analog voice via the Internet network, which is use IP address to determine the address of the user who is communicating with.

สารบัญ

สารบัญภาพ

สารบัญตาราง

บทที่ 1

บทนำ

1.1 ความสำคัญและที่มา

ในปัจจุบันเครือข่ายอินเตอร์เน็ตได้มีจำนวนผู้ใช้เพิ่มขึ้นอย่างรวดเร็ว เนื่องจากว่าผู้ใช้สามารถค้นหาข้อมูลที่ต้องการจากแหล่งข้อมูลขนาดใหญ่และสามารถติดต่อสื่อสารกับบุคคลจากที่ต่างๆได้อย่างสะดวก นอกจากนี้การพัฒนาการให้บริการในรูปแบบใหม่เพื่อรองรับความต้องการของผู้ใช้ได้เพิ่มขึ้นอย่างรวดเร็ว เนื่องมาจากความยืดหยุ่นของเครือข่ายอินเตอร์เน็ตทำให้สามารถพัฒนาการให้บริการได้ง่ายกว่าเครือข่ายอื่นๆ การให้บริการในรูปแบบหนึ่งที่กำลังได้รับความสนใจจากผู้ให้บริการรายต่างๆ เป็นอย่างมากคือ การใช้งานโทรศัพท์บนเครือข่ายอินเตอร์เน็ตซึ่งโดยใช้อินเตอร์เน็ตโปรโตคอล(Internet Protocol) โดยเรียกว่า วอยส์โอเวอร์ไอพี (Voice over IP :VoIP) ซึ่งจะรวมไปถึงการส่งวีดีโอและข้อมูลด้วย (หรือเรียกว่า การสื่อสารแบบพหุสื่อ (multimedia communication)) การให้บริการแบบนี้จะทำให้ผู้ใช้สามารถใช้โทรศัพท์บนเครือข่ายอินเตอร์เน็ตแทนเครือข่ายชนิดอื่นที่ใช้อยู่ ซึ่งการใช้งานในลักษณะนี้จะเป็นผลดีต่อผู้ใช้คือ จะช่วยประหยัดค่าโทรศัพท์ทางไกล สามารถใช้งานโทรศัพท์พร้อมกับการใช้งาน อย่างอื่นบนอินเตอร์เน็ต มีการเชื่อมต่อในการใช้งานที่ยืดหยุ่นกว่าโทรศัพท์ทั่วไปรวมทั้งยังสามารถปรับแต่งการใช้งานโทรศัพท์ได้หลากหลายกว่า เมื่อพิจารณาในแง่ของผู้ให้บริการหรือผู้ดูแลเครือข่าย การพัฒนาปรับปรุงเครือข่ายหรือการให้บริการสามารถทำได้ง่ายกว่า รวมทั้งการดูแลเครือข่ายทำได้สะดวกกว่าเพราะว่าไม่จำเป็นต้องแยกเครือข่ายสำหรับ ข้อมูล, เสียงและวิดีโอ ระบบวอยส์โอเวอร์ไอพีจะทำให้ข้อมูล, เสียงและวีดีโอสามารถส่งรวมกันไปในเครือข่ายอินเตอร์เน็ตเพียงเครือข่ายเดียวจากข้อดีเหล่านี้ต่างๆ ทำให้วอยส์โอเวอร์ไอพี เป็นที่สนใจอย่างมาก และอาจจะเป็นไปได้ว่าวอยส์โอเวอร์ไอพี จะเข้ามาแทนที่ระบบโทรศัพท์ในปัจจุบันในอนาคตอันใกล้นี้ อย่างไรก็ตามความต้องการระดับพื้นฐานของวอยส์โอเวอร์ไอพี คือคุณภาพของการให้บริการ (QOS) ต้องเทียบเท่าคุณภาพในระบบโทรศัพท์หรือดีกว่า ซึ่งเป็นปัญหาสำคัญสำหรับการใช้งานวอยส์โอเวอร์ไอพีในทางปฏิบัติ เนื่องจากว่าเครือข่ายอินเตอร์เน็ตเป็นเครือข่ายที่ไม่รับประกันในเรื่องคุณภาพของการให้บริการ ปัญหานี้จึงเป็นอุปสรรคสำคัญในการนำวอยส์โอเวอร์ไอพี มาใช้แทนระบบโทรศัพท์ที่มีอยู่ เครือข่ายอินเตอร์เน็ตได้ให้การบริการในการส่งข้อมูลทั่วไปซึ่งไม่ต้องการคุณสมบัติแบบเวลาจริง (Real time) แต่สำหรับในกรณีของวอยส์โอเวอร์ไอพี หรือการสื่อสารแบบพหุสื่อต้องการคุณสมบัติแบบเวลาจริง ดังนั้นจึงต้องมีการพัฒนาโปรโตคอลที่สามารถให้คุณสมบัติดังกล่าวในการส่งข้อมูล และที่สำคัญวอยส์โอเวอร์ไอพี ต้องการฟังก์ชันในการสร้างหรือสิ้นสุดการเรียก และหาตำแหน่งที่อยู่ของผู้ใช้ที่ถูกเรียก รวมทั้งฟังก์ชันที่ช่วยให้สามารถให้บริการได้เช่นเดียวกับในระบบโทรศัพท์ซึ่งในชุดโปรโตคอล ทีซีพี/ไอพี (Transport Control Protocol/Internet Protocol:TCP/IP) นั้นไม่มีโปรโตคอลที่มีฟังก์ชันดังกล่าว ดังนั้นจึงต้องมีการพัฒนาโปรโตคอลขึ้นใหม่เพื่อรองรับวอยส์โอเวอร์ไอพีในปัจจุบันโปรโตคอลสำหรับวอยส์โอเวอร์ไอพี มีอยู่ 2 มาตรฐานคือ H.323และ SIP(Session Initial Protocol)โปรโตคอล H.323 เป็นโปรโตคอลที่พัฒนาโดย ITU-T(International Telecommunications Union- Telecommunications section) ส่วน SIP ถูกพัฒนาโดย IETF (Internet Engineering Task Force) โปรโตคอลทั้งสองมีหน้าที่หลักในการสร้างจุดสิ้น

สุดและการเปลี่ยนแปลงรายละเอียดของการเรียกระหว่างผู้ใช้ วอยส์โอเวอร์ไอพี รวมทั้งยังสามารถให้ฟังก์ชันเพิ่มเติมอื่นๆ โปรโตคอลทั้งสองเป็นโปรโตคอลสำหรับวอยส์โอเวอร์ไอพีซึ่งใช้บริการชุดโปรโตคอลทีซีพี/ไอพี ในชั้นต่ำกว่า และสามารถใช้งานร่วมกับโปรโตคอลอื่น เพื่อให้เกิดการบริการที่มีคุณภาพมากขึ้น

1.2 วัตถุประสงค์

1. เพื่อให้ได้ต้นแบบที่สามารถทำการเชื่อมต่อกับเครื่องโทรศัพท์
2. ศึกษาการทำงานของโปรโตคอล H.323 และสามารถนำมาประยุกต์ใช้งานได้
3. เพื่อให้ได้ระบบจำลองการใช้งานโทรศัพท์
4. เพื่อเป็นแนวทางในการนำระบบ ไปประยุกต์ใช้งานต่อไป

1.3 ขอบเขตของโครงงาน

การพัฒนาโครงงานนี้มีการสร้างระบบขึ้นมา เพื่อจำลองการทำงานระบบโทรศัพท์ขึ้นมา ครอบคลุมงานต่างๆ แบ่งออกเป็น 2 ส่วนดังนี้

1. การเชื่อมต่อกับเครื่องโทรศัพท์และจำลองสัญญาณต่างๆ ของโทรศัพท์
2. คอมพิวเตอร์ที่เป็นเทอร์มินัล (Terminal) ทำหน้าที่เป็นตัวกลางในการนำข้อมูลที่ได้ทำการแปลงมาทำการจัดให้อยู่ในรูปแบบของโปรโตคอล ในที่นี้จะอ้างอิงกับโปรโตคอล H.323 เพื่อทำการติดต่อภายในระบบที่มีการจำลองขึ้นมา

1.4 วิธีการดำเนินงาน

โครงงานนี้เริ่มต้นด้วยการศึกษาทฤษฎีพื้นฐานต่างๆ ซึ่งมีเรื่องหลักๆที่ต้องศึกษาอยู่ 4 เรื่องคือ

1. ระบบโทรศัพท์พื้นฐานทั่วไป
2. รูปแบบการใช้งานของวอยส์โอเวอร์ไอพี
3. รูปแบบพื้นฐานของแพ็คเกจเสียง
4. โปรโตคอล H.323

จากนั้นจะเป็นขั้นตอนของการพัฒนาทั้งในส่วนของฮาร์ดแวร์และโปรแกรมในส่วนของเทอร์มินัล

บทที่ 2

ทฤษฎีที่เกี่ยวข้อง

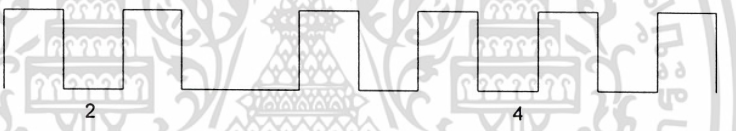
2.1 ระบบโทรศัพท์พื้นฐาน

ปัจจุบันการสื่อสารได้เข้ามามีบทบาทเป็นอย่างมากในชีวิตประจำวัน มีการติดต่อสื่อสารกันตลอดเวลาก็ว่าได้และเทคโนโลยีที่สำคัญที่ใช้ในการติดต่อสื่อสารกันก็คือโทรศัพท์และระบบโทรศัพท์จัดว่าเป็นระบบสื่อสารที่ใกล้ตัวเรามาก โทรศัพท์ที่ใช้กันทั่วๆไปมีอยู่ 2 แบบคือ แบบกดปุ่มและระบบหมุน หน้าที่ของทั้ง 2 ระบบนี้จะเหมือนๆกัน จะต่างกันตรงที่แบบกดปุ่มจะส่งสัญญาณโดยใช้คลื่นความถี่ผสมกันเรียกว่าดีทีเอ็มเอฟ ส่วนแบบหมุนจะส่งสัญญาณเป็นจำนวนพัลส์ (Pulse) ตามหมายเลขที่หมุน

2.1.1 ระบบสัญญาณโทรศัพท์

ระบบโทรศัพท์ในปัจจุบันแบ่งออกเป็น 2 ชนิดใหญ่คือ

1. ระบบพัลส์ ระบบนี้จะผลิตพัลส์จำนวนตามหมายเลขเว้นเป็นช่วงๆ ดังแสดงตัวอย่างได้ดังรูปที่ 2.1



รูปที่ 2.1 แสดงสัญญาณพัลส์เมื่อกด2และ4

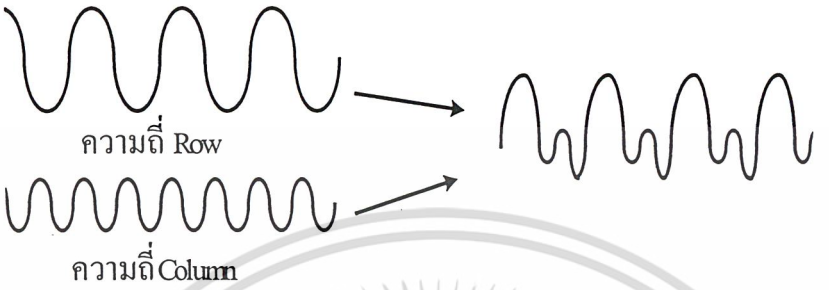
การผลิตพัลส์นี้อาจทำโดยการตัดต่อด้วยสวิตช์ทางกล เช่นในระบบจานหมุน(Dial) หรือใช้การผลิตความถี่ด้วยออสซิเลเตอร์แบบระบบกดปุ่มกด (Touch Pulse) ก็ได้

2. ระบบดีทีเอ็มเอฟ (Dual Tone Multi Frequency dialing) โทรศัพท์ชนิดกดปุ่มจะมีสัญญาณดีทีเอ็มเอฟซึ่งเป็นสัญญาณของความถี่ เราสามารถนำสัญญาณเหล่านี้ในการควบคุมจะประกอบไปด้วยโทนเสียง 2 ความถี่ผสมกัน โดยถ้าเราพิจารณาปุ่มกดหรือคีย์บอร์ด ประกอบกับตารางแบ่งกลุ่มความถี่ตามแถวและหลัก

	1209 Hz	1366 Hz	1477 Hz	1633 Hz
579 Hz	1	2	3	A
770 Hz	4	5	6	B
852 Hz	7	8	9	C
941 Hz	*	0	#	D

รูปที่ 2.2 แสดงการแบ่งกลุ่มความถี่ตามแถวและหลัก

หากกดหมายเลข 2 ซึ่งตรงกับ แถวที่1 หลักที่2 วงจรภายในเครื่องจะทำการเข้ารหัสข้อมูลที่ได้แล้วส่งให้วงจรผลิตสัญญาณผสมระหว่างความถี่ 679 Hz กับ 1336 Hz ออกไป หรือในกรณีหมายเลข 9 ก็จะประกอบด้วยความถี่ในแถวที่ 3 คือ 825 Hz กับความถี่ในหลักที่ 3 คือ 1447 Hz ผสมออกไปยังคู่สายโทรศัพท์ซึ่งถ้าจับสัญญาณที่ส่งออกไปจะได้ดังรูปที่ 2.3



รูปที่ 2.3 แสดงสัญญาณผสมคลื่นความถี่

2.1.2 สัญญาณพื้นฐานของโทรศัพท์

สัญญาณพื้นฐานของโทรศัพท์ที่สำคัญ มีอยู่ 3 สัญญาณด้วยกันคือ

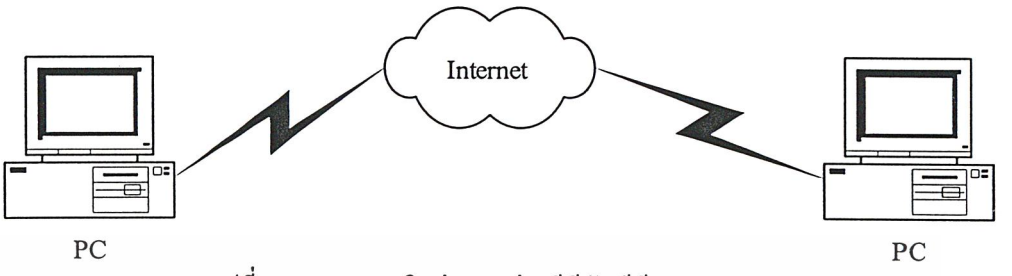
1. สัญญาณให้หมุน (Dial Tone) เป็นสัญญาณต่อเนื่องความถี่ประมาณ 400 Hz ใช้เพื่อแสดงให้ผู้เรียกทราบว่า "พร้อมรับหมายเลขโทรศัพท์ได้"
2. สัญญาณสายไม่ว่าง (Busy Tone) เป็นสัญญาณ 400 Hz ดังเป็นช่วงเวลาประมาณ 0.5 วินาที เงียบ 0.5 วินาที ใช้เพื่อเตือนให้ผู้เรียกทราบว่า "สายไม่ว่าง" ผู้เรียกควรวางหูสักพักแล้วจึงทำการติดต่อใหม่
3. สัญญาณเรียกกลับ (Ring Back Tone) เป็นสัญญาณ 400 Hz ส่ง 1 วินาทีเงียบประมาณ 3 วินาที เป็นสัญญาณที่ชุมสายแจ้งให้ผู้รับทราบว่า "มีคนโทรเข้ามา" ให้ไปรับสายได้ สัญญาณกริ่งเรียกนี้ในระบบเก่ามักใช้ความถี่ประมาณ 25 Hz โดยจะส่งเป็นเสียงกริ่งให้ได้ยินประมาณ 15 ครั้ง

2.2 วอยส์โอเวอร์ไอพี (Voice over IP)

บริการโทรศัพท์ผ่านระบบอินเตอร์เน็ต ที่รู้จักในชื่อที่แตกต่างกันออกไป ไม่ว่าจะเป็น VoIP, Voice over IP, Voice over Internet Protocol, Net Phone, Web Phoneหรือ Internet Telephone ซึ่งล้วนแล้วแต่หมายถึงเทคโนโลยีซึ่งหลอมรวมการสื่อสารสัญญาณเสียง กับการสื่อสารด้วยสัญญาณข้อมูลเข้าไว้ด้วยกัน ทั้งนี้ก็เพื่อให้การส่งผ่านสัญญาณทั้งสองไปบนระบบเครือข่ายด้วยโปรโตคอลที่ใช้กันอย่างแพร่หลายในปัจจุบันอย่างไอพีได้ หรือจะให้เข้าใจให้ง่ายขึ้นก็คือ การบริการที่ใช้เครื่องคอมพิวเตอร์ ในการสื่อสารพูดคุยแลกเปลี่ยนกัน แทนการใช้เครื่องโทรศัพท์แบบเดิม ผ่านทางระบบเครือข่ายอินเตอร์เน็ตนั่นเอง

2.2.1 ประเภทของโทรศัพท์ผ่านอินเตอร์เน็ต

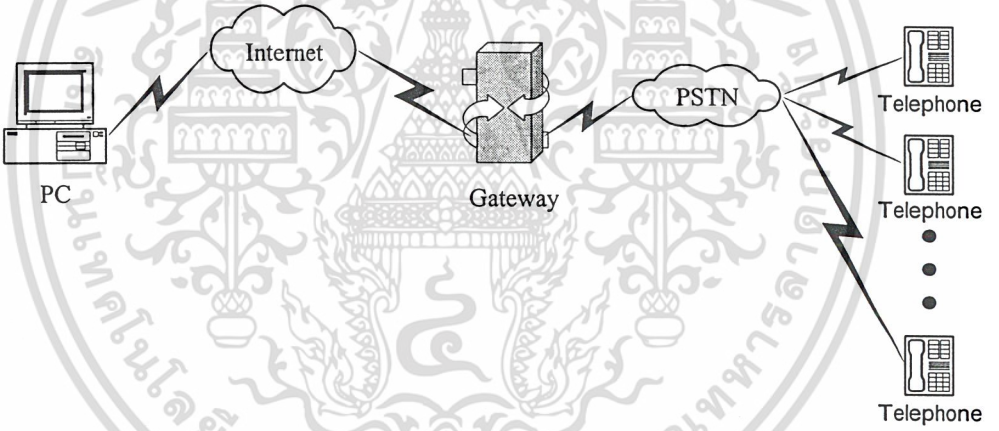
1. จากเครื่องคอมพิวเตอร์เครื่องหนึ่งสู่เครื่องคอมพิวเตอร์อีกเครื่องหนึ่ง (PC-to-PC)



รูปที่ 2.4 แสดงการติดต่อระหว่างพีซีกับพีซี

วิธีการนี้จำเป็นต้องอาศัยเครื่องคอมพิวเตอร์ทั้งต้นทางและปลายทาง พร้อมทั้งติดตั้งโปรแกรมเดียวกัน หรือติดตั้งโปรแกรมที่สามารถใช้งานร่วมกันได้ ซึ่งรูปแบบนี้เป็นวิธีการสื่อสารที่ไม่ต้องเสียค่าบริการโทรศัพท์แต่อย่างใดเลย

2. จากเครื่องคอมพิวเตอร์สู่เครื่องโทรศัพท์ (PC-to-Phone)



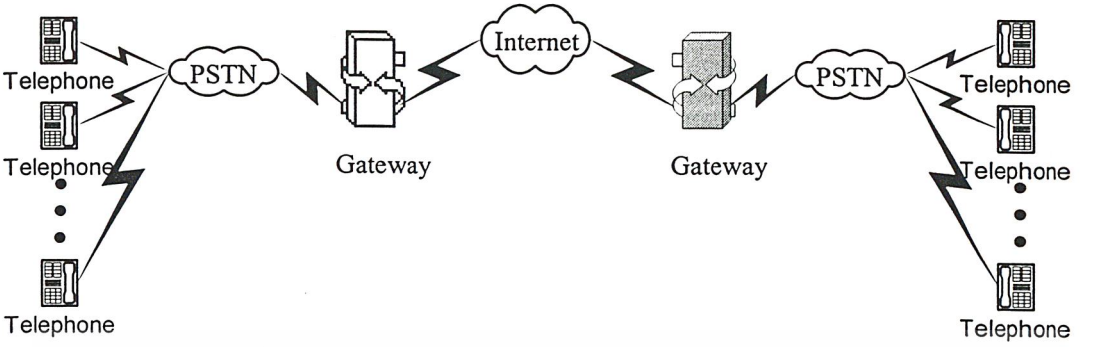
รูปที่ 2.5 แสดงการติดต่อระหว่างพีซีกับเครื่องโทรศัพท์

เป็นรูปแบบที่ใช้ได้กับผู้ใช้ต้นทางที่มีเครื่องคอมพิวเตอร์และโปรแกรมโทรศัพท์ โดยผู้รับปลายทางนั้นใช้เครื่องโทรศัพท์ธรรมดา แต่วิธีนี้ต้องอาศัยผู้ให้บริการในการเชื่อมต่อระบบอินเตอร์เน็ตเข้ากับระบบเครือข่ายโทรศัพท์ท้องถิ่น (Internet Telephone Service Provider หรือ ITSP) โดยผู้ใช้บริการต้องเสียค่าบริการตามเวลาที่ใช้งานจริง

3. จากเครื่องโทรศัพท์สู่เครื่องคอมพิวเตอร์ (Phone-to PC)

วิธีการนี้ใช้หลักการเช่นเดียวกันกับ PC-to-Phone แต่ต้นทางจะเป็นเครื่องโทรศัพท์ธรรมดา ขณะที่ปลายทางนั้นเป็นเครื่องคอมพิวเตอร์และโปรแกรมโทรศัพท์แทน ซึ่งผู้ใช้งานต้องเสียค่าบริการตามที่ใช้งานจริงเช่นเดียวกัน

4. จากเครื่องโทรศัพท์สู่เครื่องโทรศัพท์ (Phone-to-Phone)



รูปที่ 2.6 แสดงการติดต่อระหว่างเครื่องโทรศัพท์กับเครื่องโทรศัพท์

เป็นวิธีที่ทั้งต้นทางและปลายทางจะต้องอาศัยการบริการจาก ITSP ซึ่งทำให้ค่าบริการสูงกว่าวิธีอื่นๆ แต่เป็นวิธีที่ง่ายและเป็นที่คุ้นเคยในการใช้งานมากที่สุดด้วย

2.2.2 ขั้นตอนการทำงานของวอยส์โอเวอร์ไอพี

1. เมื่อผู้พูดโทรศัพท์จากเครื่องโทรศัพท์ธรรมดาหรือพูดผ่านไมโครโฟนที่ถูกต่อเข้ากับการ์ดเสียงของเครื่องคอมพิวเตอร์ คลื่นสัญญาณเสียงแบบอนาลอก ก็จะได้รับการแปลงเป็นสัญญาณดิจิตอล จากนั้นจะถูกบีบอัดด้วยตัวถอดรหัสผ่านอุปกรณ์ชุมสายส่วนตัว (PrivateBoxExchange:PBX) หรือวอยส์โอเวอร์ไอพีเกตเวย์

2. เมื่อผ่านวอยส์โอเวอร์ไอพีเกตเวย์แล้วก็จะถูกส่งต่อไปยังเกตคีปเปอร์เพื่อค้นหาเครื่องปลายทางที่จะรับการติดต่อ เช่นหมายเลขไอพี, หมายเลขโทรศัพท์ เป็นต้น แล้วแปลงเป็นแพ็คเกจข้อมูล ส่งออกไปบนระบบเครือข่ายอินเตอร์เน็ตนั่นเอง

3. ข้อมูลจะผ่านมาที่วอยส์โอเวอร์ไอพีเกตเวย์ปลายทางแล้วจึงทำการย้อนกระบวนการทั้งหมดเพื่อส่งให้กับผู้รับปลายทางต่อไป

2.2.3 องค์ประกอบของวอยส์โอเวอร์ไอพี

1. โปรแกรมที่รันบนเครื่องคอมพิวเตอร์ของเครื่องลูกข่ายหรือเครื่องโทรศัพท์ผ่านอินเตอร์เน็ต อาจจะเป็นเครื่องคอมพิวเตอร์ที่ได้รับการติดตั้งโปรแกรมสื่อสารไอพีหรืออุปกรณ์ที่ได้รับการออกแบบขึ้นมาสำหรับใช้งานโทรศัพท์ผ่านระบบอินเตอร์เน็ตโดยเฉพาะ

2. วอยส์โอเวอร์ไอพีเกตเวย์ซึ่งเป็นเครื่องเซิร์ฟเวอร์ที่ใช้งานสำหรับให้บริการโทรศัพท์ผ่านระบบอินเตอร์เน็ตเป็นตัวกลางในการเชื่อมต่อกับเครื่องชุมสายโทรศัพท์สาธารณะ (Public Switched Telephone Network: PSTN) และระบบเครือข่ายอินเตอร์เน็ตอย่างเครือข่ายไอพีซึ่งการจะใช้งานระบบโทรศัพท์ไอพีต้องอาศัยอุปกรณ์นี้เป็นตัว กลาง

3. เกตคีปเปอร์ซึ่งเป็นเครื่องเซิร์ฟเวอร์ที่เชื่อมต่อเข้ากับระบบอินเตอร์เน็ต เป็นตัวกลางที่ใช้บริหารจัดการและควบคุมการให้บริการของวอยส์โอเวอร์ไอพีเกตเวย์กับเครื่องคอมพิวเตอร์ที่ติดตั้งซอฟท์แวร์สำหรับใช้งานวอยส์โอเวอร์ไอพีหรือเครื่องโทรศัพท์แบบไอพี

2.2.4 คุณสมบัติของอุปกรณ์วอยส์โอเวอร์ไอพี

อุปกรณ์ที่จำเป็นจะต้องมีเมื่อใช้งานเครื่องคอมพิวเตอร์สำหรับบริการโทรศัพท์ผ่านอินเตอร์เน็ต นอกจากจะเป็นเครื่องคอมพิวเตอร์ที่มีอยู่แล้ว ก็ประกอบไปด้วยการ์ดเสียง, ลำโพง, ไมโครโฟน, หูฟังและซอฟท์แวร์สำหรับใช้งาน แต่ถ้าเป็นอุปกรณ์ที่ได้รับการออกแบบมาเฉพาะงานนั้น ก็มี 3 อุปกรณ์หลักๆ คือ

1. โทรศัพท์ไอพี (IP Telephony)

นอกจากจะมีหน้าตาที่ดูคล้ายกับโทรศัพท์โดยทั่วไป ที่ใช้งานกันอยู่นั้น ยังมีองค์ประกอบที่แตกต่างกันอยู่ก็คือ จะมีพอร์ตแบบ RJ-45 สำหรับเชื่อมต่อเข้ากับสวิตช์ในระบบเครือข่ายไอพีและพอร์ตที่เชื่อมต่อเข้ากับเครื่องชุมสายส่วนตัวได้ ทำให้เราสามารถใช้งานได้ทั้งการโทรศัพท์แบบปกติทั่วไปหรือจะเลือกใช้งานในแบบโทรศัพท์ไอพีก็สามารถทำได้เช่นเดียวกัน

2.โทรศัพท์ไอพีแบบไร้สาย

มีรูปแบบเช่นเดียวกับโทรศัพท์มือถือที่เราคุ้นเคยกัน โดยสามารถติดต่อสื่อสารในแบบไร้สายมาตรฐานไวไฟ (Wi-Fi) ซึ่งต้องเชื่อมต่อผ่านแอคเซสพอยท์เข้าสู่อินเตอร์เน็ตด้วยแล้ว ก็ยิ่งทำให้การใช้งานนั้นมีความยืดหยุ่นยิ่งขึ้นไปอีกด้วย

3. วอยส์โอเวอร์ไอพีเกตเวย์

เป็นอุปกรณ์ในรูปแบบเราท์เตอร์ ที่มีคุณสมบัติเช่นเดียวกับเราท์เตอร์ที่ใช้งานกันอยู่ แต่มีคุณสมบัติที่ถูกเพิ่มเติม ให้สามารถรองรับโปรโตคอลการสื่อสารของวอยส์โอเวอร์ไอพี นั่นก็คือ โปรโตคอล H.323, SIP หรือ MGCP เป็นต้น ซึ่งวอยส์โอเวอร์ไอพีเกตเวย์ เป็นอุปกรณ์ที่ใช้เชื่อมต่อกับระบบอินเตอร์เน็ต เพื่อให้สามารถโทรศัพท์ผ่านเครือข่ายได้ โดยถ้าต้องการใช้งานเป็นจุดเชื่อมต่อเครื่องโทรศัพท์ธรรมดา หรือเครื่องโทรสารให้ใช้งานร่วมกับวอยส์โอเวอร์ไอพีได้ ก็ต้องเลือกรุ่นที่มีพอร์ตแบบ FXS ซึ่งเป็นพอร์ตที่ใช้เชื่อมเข้ากับเครื่องโทรศัพท์ หรือถ้าต้องการเชื่อมต่อเข้ากับกล่องอุปกรณ์เครื่องชุมสายส่วนตัวหรือเครื่องชุมสายโทรศัพท์สาธารณะ ก็เลือกรุ่นที่มีพอร์ตแบบ FXO นอกจากนั้นก็ยังมีรุ่นที่มีทั้งสองพอร์ตให้เลือกใช้ก็ยิ่งให้ความคุ้มค่าได้มากมายทีเดียว

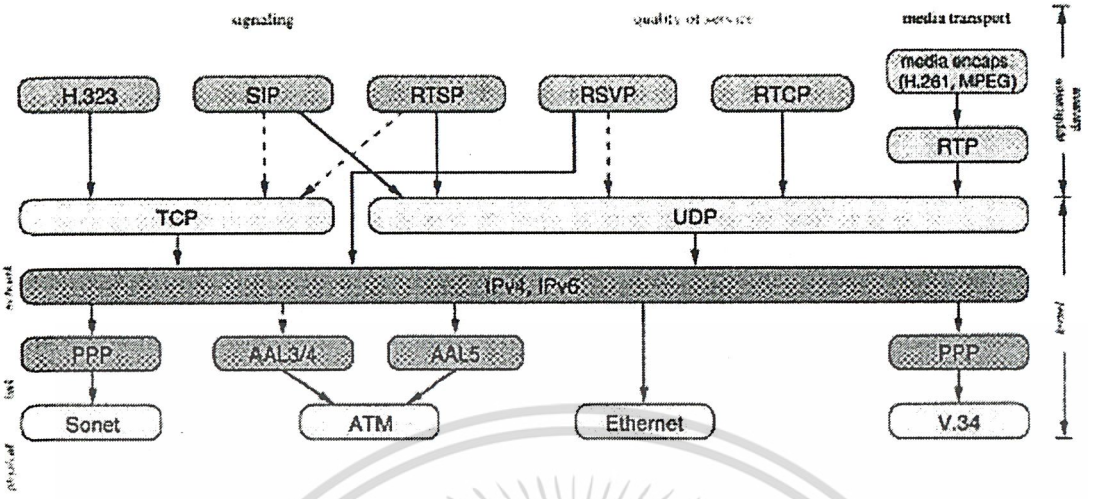
2.2.5 ตารางแสดงคุณสมบัติสำคัญของ VoIP

มาตรฐานที่สนับสนุน	**มาตรฐาน VoIP:** โปรโตคอล ITU H.323 V3 VoIP หรือโปรโตคอล SIP **มาตรฐาน IP:** IEEE802.3, 10BASE-T, IEEE802.3u, 100BASE-TX
พอร์ต	**IP Telephony:** พอร์ต RJ-45 สำหรับเชื่อมต่อเข้ากับเครือข่ายไอพี, พอร์ต RJ-11 สำหรับเชื่อมต่อเข้ากับระบบโทรศัพท์ **VoIP Gateway:** พอร์ต FXS สำหรับเชื่อมต่อโทรศัพท์, โทรสาร/พอร์ต FXO สำหรับเชื่อมต่อกับอุปกรณ์ PBX หรือ PSTN, พอร์ต RJ-45 สำหรับเชื่อมต่อเข้ากับเครือข่ายไอพี
การเข้ารหัสเสียง	มาตรฐาน G.711A/g law, G723.1, G729A
คุณสมบัติด้านเสียง	ลดสัญญาณรบกวนได้, สามารถป้องกันเสียงสะท้อนได้, ให้สัญญาณเสียงที่คมชัด

ตารางที่ 2.1 คุณสมบัติที่สำคัญของวอยส์โอเวอร์ไอพี

2.2.6 โปรโตคอลสแตกของวอยส์โอเวอร์ไอพี

โปรโตคอลสแตกสำหรับวอยส์โอเวอร์ไอพีจะเป็นดังรูปที่ 2.7 จะเห็นว่าทั้งโปรโตคอล H.323 และ SIPเป็นโปรโตคอลในชั้นแอพพลิเคชัน (application layer) และใช้บริการของโปรโตคอลในชั้นที่ต่ำกว่า SIP สามารถใช้ได้ทั้งยูดีพีและทีซีพีส่วน H.323 ใช้ทีซีพีเท่านั้นแต่เนื่องจากว่าฟังก์ชันของ H.323 และ SIP มีขอบเขตจำกัด ดังนั้นจึงได้นำโปรโตคอลอื่นมาช่วยในการทำงานซึ่งได้แก่ RTSP (Real-time Streaming Protocol) RSVP (Resource Reservation Protocol) และ RTP/RTCP ซึ่งทำให้วอยส์โอเวอร์ไอพีสามารถให้บริการได้อย่างมีประสิทธิภาพมากขึ้น โปรโตคอลดังกล่าว เป็นโปรโตคอลในชั้นแอพพลิเคชันซึ่งทำงานอยู่บนชุดโปรโตคอล ทีซีพี/ไอพีโปรโตคอลเหล่านี้ไม่ได้เป็นโปรโตคอลเฉพาะสำหรับวอยส์โอเวอร์ไอพีแต่สำหรับโปรโตคอล H.323 และ SIP ซึ่งเป็นโปรโตคอลหลักสำหรับวอยส์โอเวอร์ไอพีจะกล่าวถึงรายละเอียดในหัวข้อถัดไป



รูปที่ 2.7 โปรโตคอลสแตกของวอยส์โอเวอร์ไอพี

2.3 พื้นฐานแพ็กเกจเสียง

2.3.1 กระบวนการในการแปลงสัญญาณเสียงเป็นสัญญาณดิจิตอล

เสียงที่มนุษย์สามารถได้ยินมีความถี่เสียงอยู่ที่ประมาณ 20 – 20kHz โครงข่ายโทรศัพท์แบบอนาล็อกความถี่เสียงที่สามารถทำการป้อนเข้าสู่ระบบจะอยู่ในช่วง 300 - 3400 Hz แต่ในการทำโครงการนี้เราต้องนำเอาเสียงที่ได้เปลี่ยนมาเป็นข้อมูลแทนเพื่อที่จะทำการส่งผ่านเป็นดิจิตอลได้ การจะแปลงเสียงจากอนาล็อกเป็นดิจิตอลจะมีอยู่ 3 ขั้นตอนซึ่งได้แก่

แซมเปิล(Sample)

ในขั้นตอนนี้จะอาศัยหลักการของคณิตศาสตร์เข้ามาช่วย กระบวนการในการสุ่มตัวอย่างจะใช้วิธีที่เรียกว่าพัลส์แอมพลิจูดมอดูเลชั่น (Pulse Amplitude Modulation: PAM)

พัลส์โคดมอดูเลเตอร์นั้นจะเป็นการนำเอาแอมพลิจูดที่มีค่าคงที่ของพัลส์จากการสุ่มตัวอย่างที่มอดูเลตเป็นโมเดลของสัญญาณอนาล็อก ซึ่งในการกำหนดหลักของไนควิส (Nyquist) ได้กล่าวไว้ว่าต้องกำหนดให้มีอัตราการสุ่มตัวอย่างเป็นสองเท่าขององค์ประกอบทางความถี่สูงสุดของสัญญาณอนาล็อก สาเหตุที่เราใช้หลักการของไนควิสในการกำหนดก็เพราะว่าข้อมูลที่ทำการรับนั้นมีจำนวนมากที่ต้องทำการสร้างสัญญาณอินพุทขึ้นมาใหม่ ถ้ามีการผิดพลาดเกิดขึ้น สัญญาณที่ถูกสุ่มตัวอย่างที่ผิดพลาดนั้นก็สามารถตัดทิ้งไปได้ สำหรับความถี่เสียงนี้จะมีย่านความถี่อยู่ที่ 4 kHz อัตราการสุ่มจะถูกกำหนดไว้ที่ 8 kHz มีขนาดใหญ่กว่าที่ความต้องการกำหนดไว้ อย่างไรก็ตามความถี่เสียงที่ 4 kHz นี้มีประโยชน์หลายอย่าง หนึ่งในข้อได้เปรียบนี้คือ ความจริง 4 มาจากยกกำลัง 2 ของ 2 และตัวประมวลที่เป็นดิจิตอลส่วนมากนั้นจะใช้เลขฐาน 2 เป็นพื้นฐานอยู่แล้ว

ควอนไทซ์ (Quantize)

วิธีการควอนไทเซชันนี้เป็นการนำเอาสัญญาณพีเอเอ็มมาทำให้เป็นสัญญาณพีซีเอ็มในรูปของ 0 หรือ 1 ซึ่งการควอนไทซชันนี้เป็นการนำสัญญาณพีเอเอ็มที่ผ่านตัวแปลงสัญญาณอนาล็อกไป

เป็นดิจิตอล (Analog to Digital Converter) มาทำการเข้ารหัสเพื่อให้ได้รูปแบบที่ต้องการแต่ละตัวอย่างเราจะทำการ กำหนดด้วย 0 หรือ 1 ถ้าพัลส์แอมพลิจูดมอดูเลเตอร์นั้นทำการผลิต 8000th ตัวอย่างต่อของวินาที ดังนั้นบิตที่ใช้แทนพีเอเอ็มแซมเปิลต้องทำการส่งทุก 125 μs หรือ 1/8000 ของวินาทีจะหมายความว่าแต่ละพีเอเอ็มแซมเปิลจะถูกแทนด้วย 16 บิต ดังนั้นการเชื่อมต่อทางดิจิตอลอาจจะมีการรันที่ 125 kb/s (8000x16) จึงต้องมีการจัดการกำจัดบางส่วนออกไป

จะเห็นได้ชัดว่าตั้งแต่ 64 kb/s พีซีเอ็มทำการรันที่ 64 kb/s และแต่ละพีเอเอ็มแซมเปิลนั้นแทนด้วยจำนวน 8 บิตซึ่งเป็นจำนวนที่เหมาะสมเพราะว่า 8 นั้นมาจาก กำลังของ 2 ด้วย ซึ่งจาก 1 ไบต์เท่ากับ 8 บิต ดังนั้นเราจึงใช้ดิจิตอลซิกแนลโพรเซสซิ่ง (DSP) ในการดิจิไทซ์เสียงที่มีขนาดโพรเซสเซอร์ 8 บิตซึ่งในการทำงานส่วนต่างๆจะอยู่ในรูปของ 8 บิตด้วย

เข้ารหัส (Code)

การเข้ารหัสนี้เป็นการเข้ารหัสแบบไลน์โคด (Line Code) หรือว่า 0s หรือ 1sที่เหมาะสมสำหรับการส่งไปในระยะทางไกลๆ ในการทำการส่งสัญญาณพีเอเอ็มแซมเปิลในรูปของพีซีเอ็มนั้นจะสามารถทำการส่งได้ในระยะใกล้ๆเท่านั้นเพราะว่าในการส่งระยะทางไกลๆจะทำให้ความเสี่ยงในการผิดพลาดมีมากขึ้นจึงต้องมีการเข้ารหัสเพื่อแก้ไขข้อผิดพลาดในส่วนนี้

ไลน์โคดที่ใช้ในการส่งพีซีเอ็มได้แก่ Bipolar alternate mark inversion AMI (bipolar AMI) ได้ถูกนำมาประยุกต์ใช้งานโทรศัพท์ผ่านอินเตอร์เน็ตในระบบที-แคเรียร์ (T-Carrier) ซึ่งเป็นระดับเริ่มแรกของ T-carrier multiplex hierarchy หรือ T1 ปัจจุบันยังมีการใช้ระบบนี้อยู่ T1 นี้จะไม่ใช่ AMI แต่จะมีการเข้ารหัสพื้นฐานมาจาก AMI เรียกว่า binary 8 zero substitution (B8ZS) ซึ่งวิธีนี้เป็นวิธีแก้ปัญหาของวิธี AMI ไลน์โคดแบบดั้งเดิม

2.3.2 เซอร์กิตสวิตช์และแพ็กเกจสวิตช์

ระบบโทรคมนาคมในปัจจุบันมีการ Switching โดยแบ่งการทำงานได้เป็น

1. เซอร์กิตสวิตช์ (Circuit Switch) ใช้สำหรับการสื่อสารทางเสียง
2. แพ็คเกจสวิตช์ (Packet Switch) ใช้สำหรับการสื่อสารข้อมูล
3. เซลรีเลย์ (Cell Relay) ใช้สำหรับการสื่อสารทั้งทางเสียงและข้อมูล

เซอร์กิตสวิตช์ (Circuit Switch)

หลักการของเซอร์กิตสวิตช์ จะมีการจองวงจร (Dedicated Circuit) ก่อน หลังจากนั้นจึงจะสามารถสนทนาได้ เมื่อสนทนาเสร็จแล้วก็จะปล่อยวงจร (Released Circuit) เพื่อให้ผู้ใช้คนอื่นสามารถใช้สนทนาได้ต่อไป

ลักษณะเด่นทางด้านเน็ตเวิร์ค	Circuit Switching	Packet Switching
อัตราการส่งข้อมูล	คงที่และช้า (น้อยกว่า 64 kb/s)	มีความหลากหลาย (ถึงGb/s)
อัตราการเปลี่ยนแปลงของบิตเรท	ไม่มี	สูง (100/100:1)
การส่งข้อมูลซ้ำ(Retransmission)	ไม่สามารถทำได้	สามารถทำได้เร็วมากเพียงพอ
การหน่วงเวลา(Delay)	ไม่มีการหน่วงของเวลา	มีการหน่วงเวลาเกิดขึ้น
การเชื่อมต่อ	Static Route	Dynamic Route

ตารางที่ 2.2 แสดงการเปรียบเทียบการใช้เนตเวิร์คของเซอร์กิตสวิตช์และแพ็กเกจสวิตช์

ในกรณีการเชื่อมต่อโทรศัพท์จากต้นทางไปยังปลายทาง ถ้าเป็นกรณีทางไกล ก็จะทำการตรวจสอบฐานข้อมูล แล้วจองวงจรจาก Local Exchange ไปยัง TANDEM และวงจรจาก TANDEM ไปยังชุมสายปลายทาง ซึ่งการเชื่อมต่อลักษณะนี้จะไม่คุ้มค่ากับแบนด์วิดธ์ที่เสียไป จึงได้เกิดวิธีการมัลติเพล็กซ์ (Multiplex) โดยแบ่งช่องสัญญาณเป็นหลายๆช่อง (Channel) เพื่อให้สามารถใช้งานได้พร้อมๆกันหลายๆคู่สนทนา แต่ก็ยังไม่คุ้มค่ามากนัก เพราะยังมีบางช่วงเวลาของการสนทนาที่ว่างอยู่ (Silence Period) เพราะการสนทนาจริงนั้น ไม่มีใครที่สามารถพูดได้ตลอดเวลา จึงมีแนวความคิดว่าจะต้องนำช่วงเวลานี้มาใช้ประโยชน์ ซึ่งจะทำให้สามารถใช้งานได้เต็มแบนด์วิดธ์

เซอร์กิตสวิตช์นั้นจะเป็นลักษณะของคอนเนคชันโอเรียนเต็ด (Connection Oriented) คือเป็นการสร้างวงจรตลอดเส้นทางในลักษณะ Logical connection ดังนั้นเซอร์กิตสวิตช์จึงเหมาะกับการส่งข้อมูลที่มีลักษณะเป็นเวลาจริงหรือข้อมูลที่เป็นอินเตอร์แอ็คทีฟ เช่น เสียง เป็นต้น

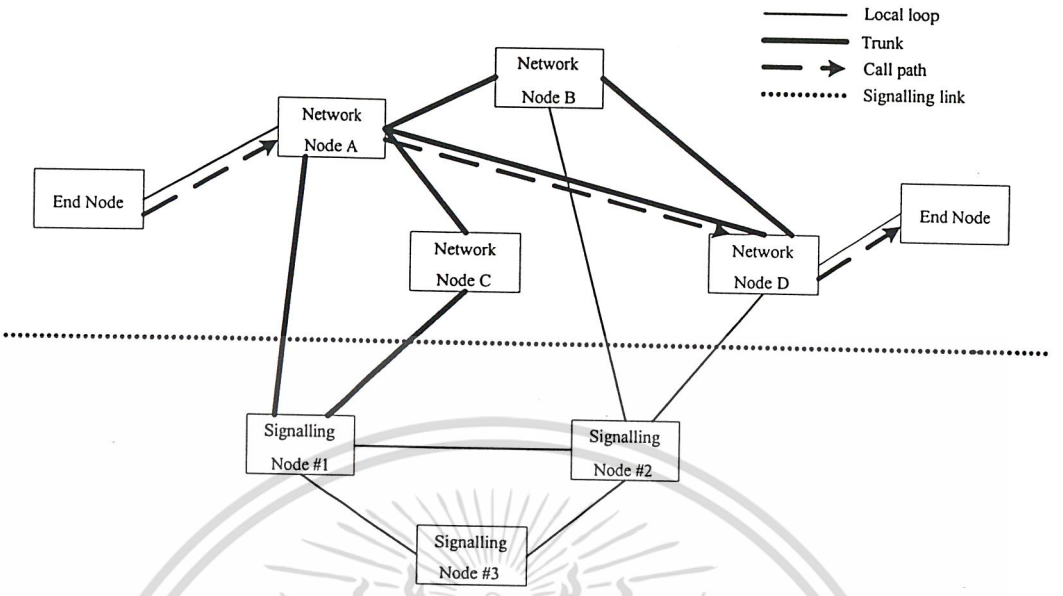
ในโครงข่ายโทรศัพท์ทั่วไปเช่น โครงข่ายโทรศัพท์พื้นฐาน (Public Switching Telephone Network: PSTN) ก็ใช้หลักการสวิตช์แบบเซอร์กิตสวิตช์นี้เช่นกัน แม้ว่าในส่วนของซิกแนลลิงจะเป็นแพ็คเกจสวิตช์ก็ตาม

เซอร์กิตสวิตช์เน็ตเวิร์คจะมีลักษณะร่วม 3 ประการคือ

1. ซิกแนลลิง โปรโตคอลจะใช้ในการเซตอัพการเชื่อมต่อ
1. ทุกบิตจะถูกส่งผ่านเส้นทางเดียวกันหมด
2. พาธหรือเส้นทางนั้นจะสามารถรองรับทุกย่านความถี่และทุกเวลา

ความสัมพันธ์ระหว่าง 3 คุณลักษณะนี้แสดงดังรูปที่ 2.8

จากรูป ภาพโหนดปลาย(End node) ของเน็ตเวิร์คเช่นโทรศัพท์และเส้นที่เป็นแสงที่แสดงการทำงานจะแทนโลคอลลูปเน็ตเวิร์ค โหนดนี้จะเป็นตัวเลือกของโทรศัพท์และเส้นที่หนาคือทรังค์ (Trunk) ที่มีความหนาแน่นของช่องสัญญาณของเสียง เส้นที่มีลูกศรและเป็นเส้นประ จะแทนเส้นทางที่เลือกเมื่อมีการเรียก การต่อเชื่อมระหว่างสวิตช์ของเสียงและซิกแนลลิงโหนดจะกระทำกันเองโดยที่ไม่ต้องมีอะไรมาช่วย ซึ่งแสดงได้จากเส้นประเมื่อมีการเรียกจากซิกแนลลิงเน็ตเวิร์คซึ่งจะมีการแยกทางกายภาพจากเน็ตเวิร์คของเสียง ซึ่งโหนดปลายจะไม่มีการต่อตรงกับซิกแนลลิงเน็ตเวิร์ค มีเพียงสวิตช์เท่านั้น



รูปที่ 2.8 แสดงคุณลักษณะของเซอร์กิตสวิตช์

ดังนั้น เสียงที่เรียกจะเซ็ตอัพโดยซิกแนลลิงโปรโตคอล มันจะทำการเชื่อมต่อเส้นทางสำหรับการเรียกเพื่อเข้าสู่เน็ตเวิร์ค ในขั้นตอนนี้จะมีการเลือกเส้นทางสำหรับการเรียก ตัวอย่างเช่นเมื่อไม่สามารถเข้าสู่เน็ตเวิร์คโหนด C ได้ ซิกแนลลิงเน็ตเวิร์คนี้สามารถที่จะหาเส้นทางที่เรียกผ่านเน็ตเวิร์คโหนด B สู่เน็ตเวิร์คโหนด D ได้ แต่ซิกแนลลิงเนตเวิร์คจะทำการเลือกเส้นทางจากโหนด A สู่โหนด B เพราะว่าในการเลือกเส้นทางนั้นซิกแนลลิงเนตเวิร์คจะเลือกเส้นทางที่สั้นที่สุด แต่ในกรณีที่เส้นทางจากโหนด A สู่โหนด D ได้ถูกใช้อยู่ซึ่งจะถูกทำการจองเส้นทางไว้ตลอดตามคุณสมบัติของเซอร์กิตสวิตช์ ดังนั้นจึงทำการเลือกเส้นทางที่ผ่านโหนด B เป็นทางเลือกที่สอง

ซิกแนลลิงโหนดจะเป็นส่วนสำคัญของการทำการเรียก เช่นจากรูปซิกแนลลิงโหนดที่ 1 จะต่ออยู่กับโหนด A และ C ส่วนซิกแนลลิงโหนดที่ 2 จะต่ออยู่กับโหนด B กับ D ดังนั้นการควบคุมโหนดต่างๆ จะสามารถทำได้โดยการใช้ซิกแนลลิงโหนดทั้งสองบนอินเตอร์เน็ตเราท์เตอร์จะมีหน้าที่ในการแลกเปลี่ยนข้อมูลต่างๆ โดยการใช้เราท์เตอร์โปรโตคอล ซึ่งจะมีการทำงานเหมือนกับซิกแนลลิงของวอยส์เน็ตเวิร์ค กระนั้นการเรียกที่ทำการเซตอัพ หรือมีการรับสายจากผู้ที่เราต้องการโทรไปหาจะใช้เส้นทางเดิมในการส่งเสียงทุกบิต ซึ่งในแต่ละทิศทางจะใช้อัตราการส่งข้อมูลที่ 64 kb/s แต่ที่อัตราการส่งข้อมูลนี้จะไม่สามารถเป็นไปได้สำหรับซิกแนลลิงเนตเวิร์ค ซึ่งซิกแนลลิงแมสเสจไม่สามารถใช้ขนาดอัตราการส่งข้อมูลเดียวกันได้

ดังนั้นคุณลักษณะของซิกแนลลิงเนตเวิร์คที่ใช้ในการเซตอัพวอยส์คอลมีดังต่อไปนี้

1. ซิกแนลลิงเนตเวิร์คจะเป็นการเชื่อมต่อกันแบบคอนเน็คชั่นเลส (Connection less) คือไม่จำเป็นต้องใช้เส้นทางเดิมในการติดต่อเสมอ
2. แมสเสจไม่จำเป็นต้องส่งผ่านเส้นทางเดิมเสมอ แต่สามารถที่จะทำการค้นหาเส้นทางใหม่ได้
3. เส้นทางนั้นไม่สามารถทำการกำหนดแบนด์วิดธ์ตายตัวได้

แพ็คเกจสวิตช์ (Packet Switch)

เมื่อมีการส่งไฟล์ระหว่างคอมพิวเตอร์สองเครื่องนั้น เส้นทางในการใช้ส่งแบบจุดต่อจุดได้ถูกนำมาใช้ วิธีนี้จะทำให้สิ้นเปลืองในการใช้แบนด์วิดธ์มาก โครงข่ายแบบแพ็คเกจได้ถูกนำมาใช้ และสามารถตอบสนองความต้องการของข้อมูลได้เจาะจงมากขึ้นซึ่งเป็นสิ่งที่จำเป็นได้แก่ การจัดการอัตราการส่งข้อมูล อัตราการเปลี่ยนแปลงของการส่งข้อมูล ความไม่แตกต่างกันของการหน่วงเวลา และที่สำคัญคือการจัดการกับความผิดพลาดที่ได้เกิดขึ้น

ในแพ็คเกจสวิตช์นั้นจะไม่มีการจองวงจรเฉพาะแต่ละวงจรในแพ็คเกจสวิตช์ จะมีการส่งหลายๆรายการมาในเวลาเดียวกัน โดยหลักการของแพ็คเกจสวิตช์จะทำการใส่เฮดเดอร์และข้อมูลข่าวสารที่จำเป็นในการจัดเส้นทางไปยังปลายทางมาในแพ็คเกจและโครงข่ายก็จะพิจารณาข้อมูลข่าวสารนั้นๆเองว่าจะส่งไปยังเส้นทางใด ซึ่งทำให้การลงทุนต่ำลงในแง่ของวงจรที่ใช้งาน

การส่งแบบแพ็คเกจสวิตช์นั้นข้อมูลจะเดินทางได้หลายเส้นทาง ในบางกรณีข้อมูลที่ส่งก่อนอาจจะได้รับหลังจากข้อมูลที่ส่งทีหลัง ดังนั้นปลายทางจะต้องทำการจัดเรียงข้อมูลใหม่ (Reassemble) เพื่อให้ข้อมูลที่ได้มีค่าถูกต้อง

ในแพ็คเกจสวิตช์ จะมีความเชื่อมั่นของการจัดเส้นทางสูงกว่าเซอร์กิตสวิตช์ กล่าวคือ ในการส่งข้อมูลของเซอร์กิตสวิตช์ เมื่อเกิดปัญหาในด้านการส่ง (Transmission) ของตัวกลาง เส้นทางสื่อสารนั้นจะขาดและต้องเชื่อมต่อใหม่ ส่วนในแพ็คเกจสวิตช์นั้นจะสามารถหาเส้นทางใหม่เพื่อส่งข้อมูลไปยังปลายทางด้วยคุณสมบัติของตัวมันเอง แพ็คเกจสวิตช์นั้นจะเป็นลักษณะของคอนเนคชั่นเลส (Connection Less) คือเป็นการสร้าง เส้นทางที่ไม่ตายตัวที่มีการส่งกระจายแพ็คเกจไปในโครงข่ายโดยไม่มีเส้นทางที่ตายตัวดังนั้นแพ็คเกจสวิตช์จึงเหมาะกับการสื่อสารข้อมูลที่มีลักษณะไม่เป็นอินเตอร์แอคทีฟ

2.4 โปรโตคอล H.323

H.323 เป็นมาตรฐานหรือโปรโตคอลสำหรับการสื่อสารแบบพหุสื่อ (Multimedia Communication) แบบเวลาจริงบนเครือข่ายไอพี โปรโตคอล H.323 ได้ให้รายละเอียดสำหรับขั้นตอนในการสร้างการเรียก (Call Setup) เอนทิตี้ภายในเครือข่าย H.323 และการทำงานร่วมกันระหว่างเอนทิตี้ภายใน

เครือข่าย H.323 ถูกพัฒนาโดย ITU-T โดยเป็นส่วนหนึ่งของมาตรฐาน H.32x ที่เป็นมาตรฐานสำหรับการประชุมแบบพหุสื่อ (Multimedia Conference) บนเครือข่ายต่างๆเช่นH.320 สำหรับเครือข่าย ISDN (Integrated Service Digital Networks) H.324 สำหรับเครือข่ายโทรศัพท์สาธารณะ H.323 จะครอบคลุมโปรโตคอลอื่นไว้ คือ

- H.225.0 สำหรับ call signaling และการจัดรูปแบบแพ็กเกจมีเดีย (Media Packet Format)
- H.245 สำหรับการแลกเปลี่ยนข้อมูลความสามารถเกี่ยวกับมีเดีย (Media Capability Exchange) และการควบคุมช่องสัญญาณตัวกลาง (Media Channel Control)
- H.450.x เป็นขั้นตอนสำหรับสร้างการบริการเพิ่มเติม (Supplementary Service)
- H.235 เป็นมาตรฐานเกี่ยวกับความปลอดภัย

รวมทั้งยังได้อ้างอิงถึงมาตรฐานในการเข้ารหัสสำหรับสัญญาณเสียง เช่น G.711, G723.1, G.729 และสัญญาณวีดีโอเช่น H.261และH.263

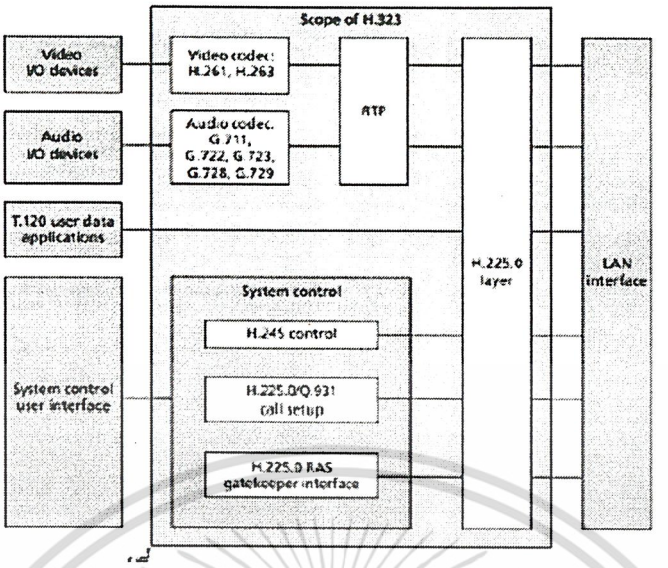
2.4.1 สถาปัตยกรรมของ H.323 (H.323 Architecture)

โปรโตคอลH.323ครอบคลุมและอ้างอิงถึงโปรโตคอลอื่นๆ เช่น H.225.0/Q.931,H.245และ H.225.0 /RAS เพื่อช่วยในการทำงานของโปรโตคอล H.323 โดยสถาปัตยกรรมของโปรโตคอล H.323 สำหรับ endpoint (หรือ terminal) จะเป็นดังรูปที่ 2.9 ขอบเขตของ H.323 ดังแสดงในรูปที่ 2.9 จะจำกัดอยู่ที่มาตรฐานในการบีบอัดข้อมูล (compression) รูปแบบแพ็กเกจมีเดีย (media packet format) การส่งสัญญาณ (signalling) และ การควบคุมการรับ -ส่งข้อมูล (flow control) ซึ่งจะมีรายละเอียดดังนี้

- การเข้ารหัส/ถอดรหัสสัญญาณวีดีโอ (video codec) ทำหน้าที่ในการเข้ารหัสสัญญาณวีดีโอสำหรับการส่ง และถอดรหัสสัญญาณวีดีโอที่ได้รับซึ่งจะถูกนำไปแสดงผลต่อไป มาตรฐานการเข้ารหัส/ถอดรหัสที่ endpoint จำเป็นต้องเข้า/ถอดรหัสได้คือ H.261 ที่ระดับความละเอียด QCIF (Quarter Common Intermediate Format) ส่วน H.263 ซึ่งให้คุณภาพที่ดีกว่าเป็นตัวเลือกที่อาจจะรองรับหรือไม่ก็ได้รายละเอียดเกี่ยวกับการเข้า/ถอดรหัสสัญญาณวีดีโอจะตกลงกันระหว่างendpointในช่วงของการแลกเปลี่ยน ความสามารถ (Capability Exchange) โดยการใช้โปรโตคอล H.245 มาตรฐานการเข้ารหัส/ถอดรหัสที่จะใช้จะต้องรองรับ โดยทุกๆ endpoint ที่เข้าร่วมในการสื่อสาร
- การเข้ารหัสสัญญาณเสียง (Audio Codec) ทำหน้าที่ในการเข้ารหัสเสียงจากไมโครโฟนหรือแหล่งกำเนิดอื่นสำหรับการส่งและถอดรหัสสัญญาณที่ถูกเข้ารหัสที่รับได้ซึ่งจะถูกส่งต่อไปยังลำโพงมาตรฐานที่ endpoint จำเป็นต้องรองรับ คือ G.711 สำหรับ G.722, G.728, G.729, MPEG-1และ G.723.1 เป็นตัวเลือกที่อาจจะรองรับหรือไม่ก็ได้ มาตรฐานการเข้ารหัส/ถอดรหัสที่ใช้จะต้องรองรับโดยทุก endpoint ซึ่งจะทำการตกลงกันโดยใช้โปรโตคอล H.245

Standard	Algorithm	Bit Rate (kbps)	Voice Quality
G711	PCM	64	ดีมาก
G723.1	MPE/ACELP	5.3, 6.3	5.3 พอใช้, 6.3 ดี
G722	Sub-band ADPCM	48, 56, 64	ดี
G728	LD-CELP	16	ดี
G729	CS-CELP	8	ดี

ตารางที่ 2.3 แสดงการบีบอัดข้อมูลด้วยวิธีต่างๆ



รูปที่ 2.9 สถาปัตยกรรม H.323

- ช่องสัญญาณส่งข้อมูล (Data Channel) มาตรฐานที่ใช้ คือ T.120 สำหรับการสร้างการประชุมข้อมูล (Data Conferencing) รายละเอียดหรือพารามิเตอร์อาจจะตกลงกันโดยใช้โปรโตคอล H.245
- RTP (Real Time Transport Protocol) ทั้งสัญญาณเสียงและวีดีโอจะถูกส่งโดยบรรจุในแพ็กเกจอาร์ทีพีซึ่งเป็นโปรโตคอลที่ใช้สำหรับการส่งข้อมูลแบบเวลาจริงบนเครือข่ายไอพี เพราะทีซีพีนั้นมีปัญหาเรื่องเสียโอเวอร์มากในการแจ้งความผิดพลาดทุกครั้งที่ส่งข้อมูลผิดพลาดและยูดีพีก็เป็นการส่งข้อมูลแบบคอนเน็คชั่นเลสซึ่งไม่สามารถนำมาใช้กับเรื่องดังกล่าวได้ อาร์ทีพีไม่ได้มีการจัดการเรื่องการจองทรัพยากรของระบบและไม่รับรองเรื่องคุณภาพการบริการ (Quality of Service) ในเรื่องของเวลาจริง โดยอาร์ทีพีจะทำงานร่วมกับอาร์ทีซีพีซึ่งทำหน้าที่ในการควบคุมการส่งข้อมูลโดยอาร์ทีพีดังที่ได้กล่าวมาแล้ว

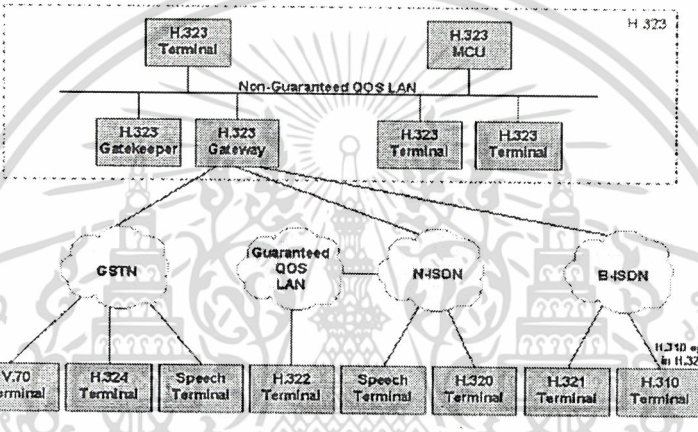
System Control Unit เป็นส่วนที่ทำหน้าที่เกี่ยวกับ การส่งสัญญาณและควบคุมการไหลของข้อมูล ประกอบด้วยส่วนต่างๆ ดังนี้

- H.245 เป็นโปรโตคอลควบคุมตัวกลางทำหน้าที่ในการแลกเปลี่ยนความสามารถ ตกลงรายละเอียดของช่องสัญญาณ (Channel Negotiation) เปลี่ยนโหมดของตัวกลาง (Switching of Media Mode) และการสร้างช่องสัญญาณทางตรรกะสำหรับการส่งเสียงหรือวิดีโอโปรโตคอลนี้จะใชทีซีพีในการส่งแมสเสจโดยใช้ช่องสัญญาณในการส่งของตัวเอง
- H.225.0/Q.931 เป็นโปรโตคอล H.225.0 ในส่วนที่ทำหน้าที่สร้างการเชื่อมต่อ (Connection Establishment) ซึ่งถูกดัดแปลงมาจากโปรโตคอล Q.931 โดยโปรโตคอลนี้จะใชทีซีพีในการส่งข้อความผ่านช่องสัญญาณของตัวเอง
- H.225.0 RAS เป็นโปรโตคอล H.225.0 ที่ทำหน้าที่ในการควบคุมการยอมรับ (Admission Control) การลงทะเบียน (Registration) และการรายงานสถานะ โปรโตคอลนี้จะใช้ระหว่าง endpoint และเกตคีปเปอร์เพื่อใช้สำหรับการควบคุมดูแลโดยเกตคีปเปอร์โดยแมสเสจในโปรโตคอลจะใช้ยูดีพี

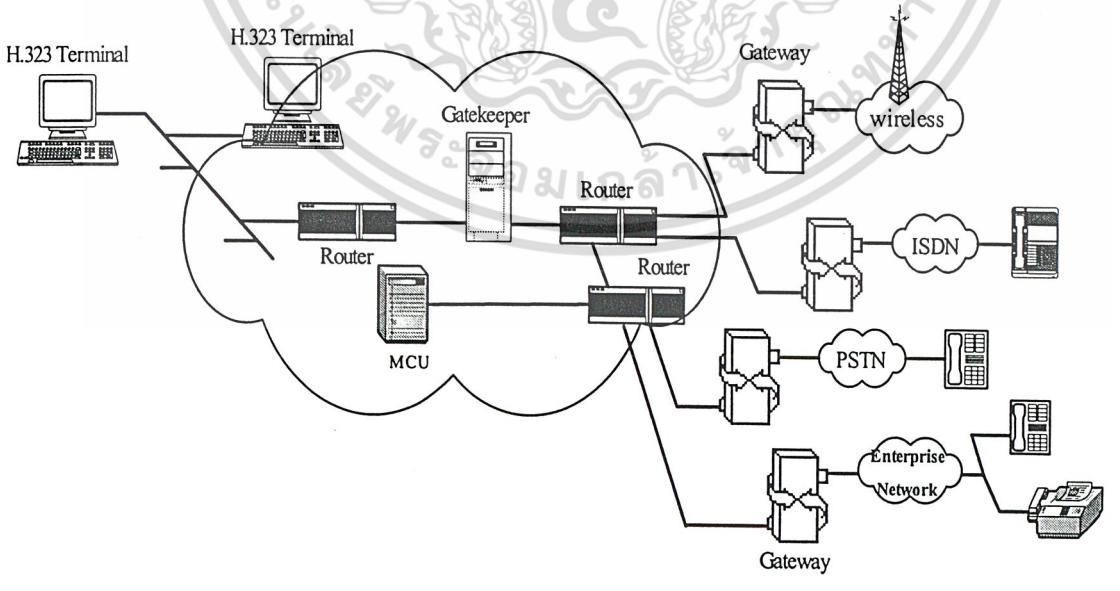
- H.225.0 layer โปรโตคอล H.225.0 เป็นโปรโตคอลสำหรับ Call-Signaling ซึ่งมีหน้าที่ดังที่กล่าวมาข้างต้น นอกจากนั้นยังทำหน้าที่ในการแปลงตัวกลาง (วิดีโอ, เสียง และข้อมูล) และข้อมูลสำหรับการ ควบคุมที่จะถูกส่งให้อยู่ในรูปแพ็กเกจที่เหมาะสมเพื่อส่งต่อให้กับ network interface และทำหน้าที่รับข้อมูลทั้งหมดจาก network interface เพื่อส่งให้ส่วนอื่นต่อไป

2.4.2 เอนทิตี้และฟังก์ชันของH.323 (H.323entities&functions)

ในมาตรฐาน H.323 ได้อธิบายถึงเอนทิตี้ที่เป็นองค์ประกอบเครือข่าย H.323ซึ่งได้แก่ เทอร์มินัล, เอ็มซียู, เกตเวย์และเกตคีปเปอร์ การเชื่อมต่อระหว่างเอนทิตี้ภายในเครือข่าย H.323 กับเครือข่ายอื่นดังแสดงในรูปที่ 2.10 รายละเอียดของแต่ละเอนทิตี้มีดังนี้



รูปที่ 2.10 เครือข่าย H.323 เอนทิตี้และฟังก์ชัน



รูปที่ 2.11 โครงสร้างโปรโตคอล H.323

a. H.323 เทอร์มินัล (H.323 terminal)

เทอร์มินัลเป็นendpointของเครือข่ายซึ่งอาจจะเป็นคอมพิวเตอร์หรือชุดอุปกรณ์ที่สามารถใช้งานโปรโตคอล H.323 ได้ เทอร์มินัลต้องสนับสนุนการสื่อสารโดยใช้เสียง ส่วนสัญญาณวีดีโอและข้อมูลเป็นตัวเลือกซึ่งฟังก์ชันหลักของเทอร์มินัลมีดังนี้

- ทำหน้าที่ในการติดต่อกับผู้ใช้โดยรับคำสั่งและแสดงผลให้กับผู้ใช้
- จัดการในการส่ง Call Signaling ให้กับ Voice Gateway
- ส่งหมายเลขโทรศัพท์(มาตรฐานE.164)และหมายเลขไอพีของผู้ใช้ให้กับเกตคีปเปอร์ซึ่งเป็นโปรโตคอลหมายเลขที่ใช้อ้างอิงถึงในการเชื่อมต่อ ในการส่งหมายเลขดังกล่าวจะบรรจุอยู่ในแมสเสจ ARQ ของคอล H.225.0/ RAS ซึ่งอาจจะมีหมายเลข alias address ส่งไปพร้อมกัน
- ทำการแปลงแพ็กเกจที่ได้รับจากเครือข่ายโดยผ่านกระบวนการของโปรโตคอลในชั้นต่างๆตามลำดับ(IP->UDP->RTP)เป็นเฟรมเสียงแล้วทำการถอดรหัสG.xxxให้อยู่ในรูปของพีซีเอ็มสตรีมเพื่อทำการส่งให้กับซาวน์การ์ดเพื่อแสดงผล ต่อไป
- ทำการเข้ารหัสG.xxxให้กับพีซีเอ็มสตรีมจากซาวน์การ์ดและทำการรวมเป็นแพ็กเกจ แล้วแปลงแพ็กเกจเป็นแพ็กเกจที่ส่งในเครือข่ายโดยผ่านกระบวนการของโปรโตคอลในชั้นต่างๆแล้วจึงทำการส่งผ่านเครือข่ายในรูปของแพ็กเกจ

b. H.323 เกทเวย์ (H.323 Gateway)

เกทเวย์เป็นเอนทิตี้ที่ทำหน้าที่ในการเชื่อมต่อระหว่างเครือข่าย H.323 กับเครือข่ายอื่นซึ่งอาจจะไม่จำเป็น ต้องมีในกรณีที่ไม่มีการเชื่อมต่อกับเครือข่ายชนิดอื่นๆ การเชื่อมต่อเครือข่าย H.323 กับเครือข่ายอื่นโดยใช้เกตเวย์จะมีลักษณะดังรูปที่ 2.10 เกตเวย์ทำหน้าที่เสมือนเป็น endpoint ของเครือข่ายหนึ่งในการเชื่อมต่อระหว่างเครือข่าย โดยจะทำหน้าที่ในการเชื่อมต่อระหว่างเครือข่ายดังนี้

- สร้างการเชื่อมต่อกับพีเอสทีเอ็นเทอร์มินัลในระบบอนาลอก
- สร้างการเชื่อมต่อกับเทอร์มินัลที่รองรับมาตรฐานH.320 บนเครือข่ายเซอร์กิตสวิตช์ที่เป็นไอเอสดีเอ็น
- สร้างการเชื่อมต่อเทอร์มินัลที่รองรับมาตรฐาน H.324 บน เครือข่ายพีเอสทีเอ็น

เนื่องจากเกตเวย์สามารถให้การเชื่อมต่อระหว่าง H.323 กับเครือข่ายอื่น ดังนั้นฟังก์ชันของเกตเวย์ จึงเป็น ฟังก์ชันในการแปลงข้อมูลระหว่าง 2 เครือข่ายคือ

- รับและประมวลผลการเรียกที่มาจากเทอร์มินัลในเครือข่ายอื่นไปยังเทอร์มินัล H.323 เกตเวย์จะต้องทำการแปลง การส่งสัญญาณควบคุมต่างๆ จากเครือข่ายอื่นมาเป็นของ H.323 เช่น จากขั้นตอนในการสร้างการสื่อสารจากH.242เป็นH.245รวมทั้งทำหน้าที่สร้างและสิ้นสุดการเรียกหรืออาจจะมองได้ว่าเกตเวย์จะทำหน้าที่แทนเทอร์มินัลในเครือข่ายอื่นโดยเสมือนกับเป็นเทอร์มินัลในเครือข่าย H.323
- รับและประมวลผลการเรียกจากเทอร์มินัล H.323 ไปยังเครือข่ายอื่นเกตเวย์จะต้องทำการแปลงการส่งสัญญาณควบคุมต่างๆ ตามH.323 ให้เป็นมาตรฐานในเครือข่ายอื่น เช่น แปลงจากโปรโตคอล H.245 เป็น H.242 รวมทั้งสร้างและสิ้นสุดการเรียก หรืออาจจะมองว่าเกตเวย์จะทำหน้าที่แทนเทอร์มินัลในเครือข่าย H.323 เสมือนกับเป็นเทอร์มินัลในเครือข่ายอื่น

- ทำหน้าที่ในการดูแลกระบวนการการเรียกและการส่งสัญญาณ (Signalin)ว่ามีความผิดพลาดเกิดขึ้นหรือไม่ ซึ่งการทำงานจะอยู่ภายใต้การควบคุมของเกตคีปเปอร์
- ทำการเข้ารหัส G.xxx ให้กับสัญญาณพีซีเอ็ม จากเครือข่ายอื่นแล้วทำการรวมสัญญาณที่ถูกเข้ารหัสให้เป็นแพ็กเกจเพื่อส่งไปในเครือข่ายไอพีโดยผ่านกระบวนการแปลงของโปรโตคอลในชั้นต่างๆเพื่อให้อยู่ในรูปแพ็กเกจที่สามารถส่งไปในเครือข่าย
- ทำการแปลงแพ็กเกจจากเครือข่ายไอพีกลับแพ็กเกจเสียง แล้วทำการถอดรหัส G.xxx และแปลงสัญญาณพีซีเอ็ม เพื่อส่งให้กับเครือข่าย ภายนอก

สำหรับฟังก์ชันเพิ่มเติมของเกตเวย์ไม่ได้มีการกำหนดไว้แน่นอนขึ้นอยู่กับผู้ออกแบบ เช่น จำนวนของเทอร์มินัลที่รองรับจำนวนการเชื่อมต่อกับเครือข่ายเซอร์กิตสวิตช์และจำนวนการประชุมที่สนับสนุนเป็นต้น เมื่อเกตเวย์มีการพัฒนามากขึ้นจะทำให้เครือข่าย H.323 สามารถเชื่อมต่อกันกับเครือข่ายชนิดอื่นๆ ได้มากขึ้น

c. H.323 Gatekeeper

เกตคีปเปอร์ทำหน้าที่ในการดูแลและให้บริการกับเอนทิตี้อื่นภายในโซนโดยโซนจะประกอบไปด้วยเกตคีปเปอร์ 1 ตัวและเอนทิตี้อื่นๆ ทั้งหมดที่ลงทะเบียนกับเกตคีปเปอร์ถึงแม้ว่าเกตคีปเปอร์เป็นเอนทิตี้ที่ไม่จำเป็นต้องมีในเครือข่าย H.323 แต่เกตคีปเปอร์ก็เป็นเอนทิตี้ที่สำคัญมาก ด้วยเหตุผลต่างๆ ดังนี้

- เครือข่ายขนาดใหญ่สามารถแบ่งเป็นหลายโซนซึ่งแต่ละโซนจะอยู่ภายในการดูแลของเกตคีปเปอร์เพื่อความสะดวกในการดูแลรักษาเครือข่าย
- เกตคีปเปอร์สามารถให้ความปลอดภัยในการเข้าถึงเครือข่ายได้โดยการให้บริการรับรองความถูกต้องสำหรับแต่ละการเรียก หรือแต่ละเอนทิตี้
- เกตคีปเปอร์เป็นศูนย์กลางในการให้บริการรับรองความถูกต้อง การให้สิทธิ และ การยอมรับของโซน
- เกตคีปเปอร์สามารถจัดการควบคุมแบนด์วิดธ์ bandwidth management) เช่นการจำกัดจำนวนของการเชื่อมต่อ เพื่อเป็นการรักษาแบนด์วิดธ์สำหรับการใช้งานอย่างอื่น เช่น อีเมล์ การโอนย้ายไฟล์ เกตคีปเปอร์ไม่สามารถเป็น endpoint ของการเชื่อมต่อได้ เมื่อมีเกตคีปเปอร์ทุกเอนทิตี้จะต้องทำการลงทะเบียนกับเกตคีปเปอร์ดังนั้นเกตคีปเปอร์จึงทำหน้าที่เป็นศูนย์กลางของการเรียกทั้งหมดภายในโซนและอาจจะให้บริการเพิ่มเติมกับโซนได้ สำหรับฟังก์ชันหลักที่จำเป็นของเกตคีปเปอร์ตามมาตรฐาน H.323 มี 15 ฟังก์ชันดังนี้

1. การแปลงแอดเดรส (Address translation) เกตคีปเปอร์จะทำหน้าที่ในการแปลง alias addressให้เป็น transport address เอนทิตี้จะทำการส่ง alias พร้อมกับการลงทะเบียนโดยใช้แมสเสจ RRQ ซึ่งอาจจะสามารถปรับเปลี่ยนในภายหลังได้
2. การควบคุมการยอมรับ (Admission control) เมื่อเอนทิตี้ภายในโซนต้องการสร้างการเรียกจะต้องทำการขออนุญาตไปยังเกตคีปเปอร์โดยใช้แมสเสจ ARQ เกตคีปเปอร์ อาจจะอนุญาตหรือไม่ก็ได้โดยจะทำการตรวจสอบจากเงื่อนไขต่างๆ เช่น แบนด์วิดธ์ แหล่งกำเนิดการเรียก และ การรับรองความถูกต้องเป็นต้น

3. การควบคุมแบนด์วิดธ์เกตคีปเปอร์สามารถรองรับการควบคุมแบนด์วิดธ์ได้ โดยเอนทิตี้จะทำการร้องขอแบนด์วิดธ์ที่ต้องการโดยใช้แมสเสจBRQและเกตคีปเปอร์จะทำการตรวจสอบค่าแบนด์วิดธ์ที่ร้องขอกับเงื่อนไขที่กำหนดไว้สำหรับการจัดการแบนด์วิดธ์แล้ว จึงจะอนุญาตหรือไม่อนุญาตด้วยการส่งแมสเสจ BCF หรือ BRJ ตามลำดับ
4. การจัดการโซน (Zone management and Directory service) จากรูปที่ 2.10 โซนจะประกอบด้วยเทอร์มินัล เกตเวย์และเอ็มซียูทั้งหมดที่ลงทะเบียนกับเกตคีปเปอร์1 ตัว เกตคีปเปอร์ทำหน้าที่ในการดูแลและจัดการให้กับทุกเอนทิตี้ที่อยู่ในภายในโซนโดยการใช้ฟังก์ชันข้างต้นและการให้บริการอื่นๆรวมทั้งการให้บริการ Directory Service ของโซน
5. การควบคุมการส่งสัญญาณ (Call Control Signaling) เกตคีปเปอร์อาจจะช่วยในการประมวลผลแมสเสจ Q.931 ที่ส่งระหว่างเทอร์มินัลได้ในระหว่างการสร้างการเรียก
6. การตรวจสอบการเรียก (Call Authorization) เกตคีปเปอร์อาจจะปฏิเสธการสร้างการเรียกจากเทอร์มินัลด้วยเหตุผลบางอย่าง เช่นจำกัดการเข้าถึงจากเทอร์มินัลหรือเกทเวย์บางตัวจำกัดการเข้าถึงในบางช่วงเวลาเป็นต้น ซึ่งเงื่อนไขในการตรวจสอบจะอยู่นอกเหนือขอบเขตของ H.323
7. การจัดการแบนด์วิดธ์ เกตคีปเปอร์จะปฏิเสธการเรียกจากเทอร์มินัลในกรณีที่มีแบนด์วิดธ์ไม่เพียงพอ รวมถึงในกรณีที่มีการร้องขอการเพิ่มแบนด์วิดธ์ สำหรับเงื่อนไขจะอยู่นอกเหนือขอบเขตของ H.323
8. การจัดการการเรียก (Call Management) เกตคีปเปอร์อาจจะทำการเก็บรักษารายการการเรียกที่เกิดขึ้นเพื่อใช้ในการระบุเทอร์มินัลที่ถูกเรียกว่าว่างหรือไม่หรือเพื่อให้ข้อมูลกับฟังก์ชันในการจัดการแบนด์วิดธ์
9. การตรวจสอบผู้ใช้ (Authenticating Users) สามารถจำกัดการเข้าถึงของผู้ใช้ได้ตามเงื่อนไขที่กำหนดไว้
10. การจัดการบริการ(Managing Services) เกตคีปเปอร์จะทำหน้าที่ในการจัดการให้บริการต่างๆแก่ผู้ใช้
11. การจัดการฐานข้อมูลของสมาชิก (Managing Subscriber Databases) เกตคีปเปอร์ทำหน้าที่ดูแลและจัดการเกี่ยวกับฐานข้อมูลของสมาชิกที่ได้ลงทะเบียนไว้กับเกตคีปเปอร์
12. การหาตำแหน่งของสมาชิก (Locating Subscribers) เกตคีปเปอร์ทำหน้าที่ในการหาตำแหน่งของสมาชิกได้โดยการค้นหาจากข้อมูลของสมาชิก ที่สมาชิกได้จากการลงทะเบียน
13. การรวบรวมข้อมูลสำหรับการเก็บค่าบริการ (Collecting Charging Information) เกตคีปเปอร์จะทำการเก็บข้อมูลที่จำเป็นสำหรับการคิดค่าบริการของการเรียกโดยที่การเรียกต้องถูกจัดเส้นทางผ่านเกตคีปเปอร์
14. การควบคุมเกตเวย์ (Managing Gateway) เกตคีปเปอร์จะควบคุมการทำงานของเกตเวย์ เช่นควบคุมการสร้างการเรียกของเกตเวย์ระหว่างเครือข่าย
15. การช่วยในการสร้างการเรียก (Assisting in Call Setup) เมื่อการเรียกถูกจัดเส้นทางผ่านเกตคีปเปอร์จะช่วยในการสร้างการเรียก เช่นอาจจะทำการจัดเส้นทางให้กับการเรียกไปยังเกตเวย์ที่

เหมาะสม การติดต่อสื่อสารระหว่างเอนทิตี้กับเกตคีปเปอร์จะใช้โปรโตคอล H.225.0 /RAS ส่วนแมสเสจ Call Signaling (H.225.0/ Q.931) และ Media Control (H.245) อาจจะผ่านเกตคีปเปอร์หรือไม่ก็ได้ขึ้นอยู่กับการลงทะเบียนของเอนทิตี้และเงื่อนไขของเกตคีปเปอร์สำหรับเกตคีปเปอร์อาจจะถูกรวมอยู่ในเกตเวย์และเอ็มซียูได้ โดยที่ต้องแยกทางตรรกะ (logical) จาก endpoint

d. Multipoint Control Unit (MCU)

เอ็มซียูทำหน้าที่ในการสนับสนุนการประชุมแบบหลายจุด (Multipoint Conference) ระหว่างเทอร์มินัล 3 เทอร์มินัลขึ้นไปเอ็มซียูเป็นเอนทิตี้ที่จะมีหรือไม่ก็ได้ เอ็มซียูประกอบด้วย Multipoint Controller (MC) และ Multipoint Processor (MP) ในการประชุมจะต้องมีเอ็มซีส่วนเอ็มพีอาจจะมีหรือไม่มีก็ได้หรืออาจจะมีมากกว่าหนึ่งก็ได้เอ็มซีเป็นส่วนที่ทำหน้าที่ในจัดการเกี่ยวกับการส่งสัญญาณในการควบคุมตัวกลางให้กับแต่ละเทอร์มินัล โดยที่ทุกเทอร์มินัลต้องมีช่องสัญญาณ H.245 เชื่อมต่อกับเอ็มซีแบบจุดถึงจุด (point-to-point) ส่วนเอ็มพีจะทำหน้าที่ในการจัดการกับมีเดียสตรีมโดยทำหน้าที่ในการผสม (mixing) สวิตซ์ (switching) และประมวลผลตัวกลางที่ใช้การประชุมภายใต้การควบคุมของเอ็มซี

2.4.3 การประชุมแบบหลายจุด (Multipoint Conference)

การประชุมแบบหลายจุดคือการสื่อสารที่มีผู้เข้าร่วมมากกว่า 2 ซึ่งจำเป็นต้องมีเอ็มซีอยู่เป็นอย่างน้อย สำหรับแบบจำลองที่ใช้มี 3 แบบ

- Centralize Model ในแบบจำลองนี้จำเป็นต้องมีเอ็มซียูอยู่ทุกเทอร์มินัลที่เข้าร่วมในการประชุมต้องมีช่องสัญญาณ H.245 เชื่อมต่อแบบจุดถึงจุด (point-to-point) กับเอ็มซียูซึ่งเอ็มซีจะหน้าที่ควบคุมการประชุมโดยใช้ฟังก์ชันของ H.245 ส่วนเอ็มพีจะทำหน้าที่รับมีเดียสตรีมจากทุกเทอร์มินัลทำการรวมสัญญาณเสียง เลือกสัญญาณวีดีโอที่ตรงกัน และประมวลผล แล้วทำการส่งกลับไปให้กลับเทอร์มินัลอื่นๆ ทุกเทอร์มินัล
- Decentralized Model ในแบบจำลองนี้ เทอร์มินัลจะมัลติคาสท์สัญญาณเสียงและวีดีโอให้กับเทอร์มินัลอื่นๆ โดยไม่ผ่านเอ็มซียูแต่การควบคุมยังคงถูกควบคุมโดยเอ็มซีผ่านทางช่องสัญญาณ H.245 ที่เชื่อมต่อกับเทอร์มินัลแบบจุดถึงจุด (point-to-point) เทอร์มินัลที่ได้รับสัญญาณจะทำหน้าที่ในการประมวลผลสัญญาณเองโดยอาจจะใช้ฟังก์ชันเอ็มพีของแต่ละเทอร์มินัลช่วยทำหน้าที่ในการประมวลผลมีเดียสตรีม
- Hybrid Model แมสเสจ H.245 รวมทั้งสัญญาณเสียงหรือวีดีโอจะถูกส่งและประมวลผลผ่านเอ็มซียูโดยใช้การเชื่อมต่อแบบจุดถึงจุด (point-to-point) ส่วนสัญญาณที่เหลือจะถูกส่งโดยเทอร์มินัลแบบมัลติคาสท์ให้กับเทอร์มินัลอื่นๆ

2.5 ทีซีพี/ไอพี (TCP/IP: Transmission Control Protocol/Internet Protocol)

อินเตอร์เน็ตนับได้ว่าเป็นเครือข่ายบนเครือข่ายที่เปิด โอกาสให้เครือข่ายคอมพิวเตอร์อื่นเชื่อมโยงเข้ามาใช้งานหรือเป็นศูนย์กลางกลางเชื่อมโยงเครือข่ายคอมพิวเตอร์อื่นๆ แต่หากที่เกิดขึ้นในการเชื่อมโยงเครือข่ายคอมพิวเตอร์เข้าด้วยกันก็คือแต่ละเครือข่ายใช้คอมพิวเตอร์ต่างชนิด ต่างยี่ห้อและระบบปฏิบัติการที่ต่างกัน

มาตรฐานของทีซีพี จึงถูกใช้เป็นกุญแจสำคัญในการแก้ปัญหาเหล่านี้ โดยกลายเป็นระบบเปิดที่สมบูรณ์แบบที่มีการเชื่อมโยงคอมพิวเตอร์ได้ตั้งแต่เครื่องพีซีจนถึงเครื่องเมนเฟรม และไม่จำกัดระบบปฏิบัติการที่ใช้ทีซีพีจึงเป็นมาตรฐานที่ทั่วโลกยอมรับและ มีอุปกรณ์และซอฟต์แวร์ผลิตออกมาสนับสนุน ทีซีพี/ไอพี มากมาย

ทีซีพี/ไอพี เป็นข้อกำหนดเกี่ยวกับรูปแบบการเชื่อมโยงในเครือข่าย (Network Protocol) จัดทำเพื่อใช้เป็นกฎเกณฑ์ให้เครื่องคอมพิวเตอร์ใช้งานร่วมกัน ในลักษณะของระบบเปิด (Open System) คือไม่ว่าจะเป็นคอมพิวเตอร์ใดหรือระบบใดก็ตาม จะสามารถติดต่อสื่อสารและแลกเปลี่ยนข้อมูลกันได้

ทีซีพี/ไอพีเป็นการกำหนดรูปแบบการสื่อสารระหว่างซอฟต์แวร์การจัดโอนย้ายข้อมูลการเสดงสถานะของเครื่องคอมพิวเตอร์ที่อยู่บนเครือข่าย ตลอดจนกฎระเบียบต่างๆ ที่กำหนดให้ทำเมื่อเกิดความผิดพลาด หรือต้องทำเพื่อป้องกันเพื่อไม่ให้เกิดความผิดพลาด

ทีซีพี/ไอพี เกิดจากการนำข้อกำหนดของรูปแบบต่างๆ กันมาใช้ร่วมกัน ทีซีพีและไอพี ต่างก็เป็นรูปแบบหนึ่งของชุดข้อกำหนดนี้ (แต่เรียกชุดข้อกำหนดรูปแบบนี้ว่า ทีซีพี/ไอพี) ถูกออกแบบมาเพื่อใช้รับส่งหรือโอนย้ายข้อมูลระหว่างคอมพิวเตอร์ที่อยู่บนระบบเครือข่ายเดียวกันหรือต่างเครือข่ายกันก็ได้และมีการจัดเตรียมข้อมูลสถานะของเครือข่ายขึ้นได้ภายในข้อกำหนดรูปแบบเอง ในการสร้างซอฟต์แวร์ของระบบ เครือข่ายจะใช้ ทีซีพี/ไอพี เป็นส่วนสนับสนุนได้ทั้งระบบเครือข่ายเฉพาะบริเวณ (Local Area Network) และเครือข่ายบริเวณกว้าง (Wide Area Network) ไม่ได้ใช้งานเฉพาะกับอินเทอร์เน็ตเท่านั้น

2.5.1 ส่วนประกอบของ ทีซีพี/ไอพี

จากที่กล่าวมาแล้วว่าทีซีพี/ไอพีประกอบไปด้วยชุดข้อกำหนดรูปแบบต่างซึ่งแบ่งเป็นกลุ่มได้ดังนี้

- **กลุ่มข้อกำหนดรูปแบบการขนส่ง (Transport Protocols)**

  ทำหน้าที่ควบคุมการเคลื่อนย้ายข้อมูลระหว่างคอมพิวเตอร์สองเครื่องแบ่งย่อยออกได้เป็นสองชนิดคือ

  1. **ทีซีพี (Transmission Control Protocol)** เป็นการบริการแบบ คอนเน็คชันเบสเซอร์วิสซึ่งคอมพิวเตอร์ด้านผู้รับและส่งต้องส่งต่อถึงกันอยู่ตลอดเวลาในระหว่างการสื่อสารถ้าเปรียบเทียบก็คล้ายกับระบบโทรศัพท์ที่ต้องติดต่อกันให้ได้ ก่อนจะพูดคุยกันได้
  2. **ยูดีพี (User Datagram Protocol)** เป็นการให้บริการแบบคอนเน็คชันเซอร์วิส คอมพิวเตอร์ด้านผู้ส่งไม่จำเป็นต้องติดต่อกับด้านผู้รับก่อน เพียงรู้ที่อยู่ข้องด้านผู้รับแล้วใส่ที่อยู่นั้น ไปกับข้อมูลที่ส่งออก ข้อมูลจะเดินทางตามเส้นทางต่างๆ เพื่อไปถึงปลายทางตามที่อยู่ คล้ายกับการส่งจดหมายที่ไปรษณีย์จะส่งให้ตามที่อยู่ที่จ่าหน้าซองจดหมายโดยผู้ส่ง และผู้รับไม่ต้องติดต่อถึงกัน

- **กลุ่มข้อกำหนดเกี่ยวกับรูปแบบเส้นทาง (Routing Protocol)**

  ทำหน้าที่พิจารณาเส้นทางที่ดีที่สุดที่ใช้ส่งข้อมูลให้มีขนาดเหมาะสม แล้วส่งออกไป เมื่อถึงผู้รับปลายทาง กลุ่มข้อมูลนี้จะทำหน้าที่ตรงข้าม คือ รวบรวมข้อมูลย่อยให้ถูกต้องก่อนการแสดงผล กลุ่มข้อกำหนดรูปแบบกลุ่มนี้ประกอบด้วย

  1. **IP (Internet Protocol)** เป็นการกำหนดรูปแบบการส่งข้อมูล

2. **ICMP (Internet Control Message Protocol)** เป็นข้อกำหนดรูปแบบของข้อมูลข่าวสารเกี่ยวกับสถานะของIPเช่นข่าวสารความผิดพลาดและผลกระทบต่อเส้นทาง เมื่อมีการเปลี่ยนแปลงฮาร์ดแวร์ ในเครือข่าย
3. **RIP (Routing Information Protocol)** ข้อกำหนดรูปแบบหนึ่งที่ใช้สำหรับทำการพิจารณาวิธีการเลือกเส้นทางเพื่อให้ได้เส้นทางที่ดีที่สุดเหมาะสมกับข้อมูลมากที่สุด
4. **OSPF (Open Shortest Path First)** ข้อกำหนดรูปแบบอีกประเภทหนึ่งที่ใช้ตัดสินเลือกเส้นทางโดยพิจารณาจากทางที่สั้นที่สุดก่อน

- **กลุ่มข้อกำหนดรูปแบบเกี่ยวกับที่อยู่เครือข่าย (Network Address)**

ทำหน้าที่พิจารณาที่อยู่ของเครือข่ายและเครื่องคอมพิวเตอร์ ไม่ว่าจะเป็นลักษณะตัวเลขหรือชื่อก็ตาม เพื่อความถูกต้องของข้อมูลที่จะไปยังผู้รับปลายทางโดยไม่ว่าเครือข่ายจะใหญ่โตสักเพียงใดหรือมีเครื่องคอมพิวเตอร์จำนวนมากก็ตาม ที่อยู่ต้องไม่ซ้ำกัน กลุ่มข้อกำหนดรูปแบบนี้มีดังนี้

1. **ARP (Address Resolution Protocol)** ข้อกำหนดรูปแบบที่พิจารณาตัวเลขที่อยู่เพื่อไม่ให้เกิดที่อยู่ซ้ำกัน
2. **DNS (Domain Name System)** ข้อกำหนดรูปแบบที่พิจารณาตัวเลขที่อยู่เมื่อรู้ชื่อของเครือข่ายหรือเครื่องคอมพิวเตอร์ เพราะการใช้งานจริงนั้นใช้เพียงที่อยู่ที่เป็นตัวเลข แต่ระบบชื่อจัดทำขึ้นเพื่อให้สะดวกต่อการใช้งานของผู้ใช้
3. **RARP (Reverse Address Resolution Protocol)** ข้อกำหนดรูปแบบที่พิจารณาตัวเลขที่อยู่เดียวกับเออาร์พี แต่จะทำตรงข้ามกับเออาร์พี

- **กลุ่มข้อกำหนดรูปแบบเกี่ยวกับเส้นทางการสื่อสารระหว่างเครือข่าย (Gateway Protocols) และสนับสนุนข้อมูลสถานะเพื่อนำไปใช้เลือกเส้นทางที่เหมาะสม**

ข้อกำหนดรูปแบบเหล่านี้ประกอบด้วย

1. **EGP (Exterior Gateway Protocol)** ข้อกำหนดรูปแบบนี้จะหาการถ่ายโอนข้อมูลเส้นทางกันระหว่างเกตเวย์ กับเครือข่ายภายนอกเพื่อทำการสื่อสาร
2. **GGP (Gateway to Gateway Protocol)** เป็นข้อกำหนดรูปแบบที่ทำงานถ่ายโอนข้อมูลเส้นทางกันระหว่างเกตเวย์ กับ เกตเวย์
3. **IGP (Interior Gateway Protocol)** ข้อกำหนดในแบบที่ถ่ายโอนข้อมูลเส้นทางกันภายในเครือข่ายเดียวกัน

- **กลุ่มข้อกำหนดรูปแบบเกี่ยวกับการบริการผู้ใช้ (User Services)**

ผู้ใช้สามารถใช้ข้อกำหนดรูปแบบได้โดยตรง ข้อกำหนดรูปแบบนี้ประกอบด้วย

1. **BOOTP (BOOT Protocol)** ข้อกำหนดรูปแบบนี้จะอ่านโปรแกรมควบคุมการทำงานจากเซิร์ฟเวอร์มาให้
2. **FTP (File Transfer Protocol)** ข้อกำหนดรูปแบบที่ให้บริการถ่ายโอนไฟล์ข้อมูลระหว่างคอมพิวเตอร์ ซึ่งอาจจะอยู่บนเครือข่ายเดียวกันหรือต่างเครือข่ายกันก็ได้
3. **TELNET** เป็นข้อกำหนดรูปแบบที่ทำให้บริการเกี่ยวกับการควบคุมการติดต่อระยะทางไกล

- **กลุ่มข้อกำหนดรูปแบบอื่นนอกเหนือจากกลุ่มที่จัดไว้และบริการที่สำคัญๆจัดทำไว้ให้บนเครือข่ายที่สนใจ มีดังนี้**

    1. **NFS (Network File System)** เป็นข้อกำหนดรูปแบบที่ทำให้ผู้ใช้คอมพิวเตอร์เครื่องหนึ่งสามารถเข้าใช้งานไฟล์ข้อมูลและดูไฟล์ข้อมูลซึ่งอยู่ในคอมพิวเตอร์เครื่องอื่นได้
    2. **NIS (Network Information System)** เป็นข้อกำหนดรูปแบบที่ให้บริการกับ User Accounts ข้ามเครือข่าย เช่น Logins และ Password
    3. **RPC (Remote Procedure Call)** ข้อกำหนดรูปแบบที่อำนวยความสะดวกให้กับโปรแกรมประยุกต์ที่ใช้งานกับการควบคุมระยะทางไกล
    4. **SMTP (Simple Procedure Call)** ข้อกำหนดรูปแบบที่ให้บริการถ่ายโอนจดหมายอิเล็กทรอนิกส์ (Electronic Mail) ระหว่างคอมพิวเตอร์
    5. **SNMP (Simple Network Management Protocol)** ข้อกำหนดรูปแบบที่ให้บริการข่าวสารต่างๆ ที่แสดงสถานะของเครือข่ายและอุปกรณ์ที่ต่ออยู่บนเครือข่าย

2.5.2 โครงสร้างของชุดโปรโตคอล ทีซีพี/ไอพี

1. **Application Layer** ในชั้นนี้ประกอบด้วยโปรแกรมประยุกต์ที่ใช้ในเครือข่ายเช่นโปรแกรมส่งถ่ายข้อมูล (File-Transfer Program) และอาจกล่าวได้ว่าโปรโตคอล ทีซีพี/ไอพี ก็คือโปรโตคอลในชั้นแอพพลิเคชันรวมกับชั้นพรีเซนเตชั่นของโอเอสไอโมเดลนั่นเอง
2. **Transport Layer** ในชั้นนี้เป็นชั้นที่ให้การส่งข้อมูลจากจุดปลายเปรียบเทียบได้กับชั้นเซสชันรวมกับทรานสปอร์ตเลเยอร์นั่นเอง โดยโปรโตคอลทีซีพี/ไอพีมีซ็อกเก็ต (Socket) เป็นจุดปลายในการสื่อสาร ซึ่งซ็อกเก็ตนี้ประกอบไปด้วยหมายเลขของคอมพิวเตอร์และหมายเลขพอร์ตของเครื่องที่ต้องการส่งข้อมูลไปถึงในชั้นนี้มีการรับรองให้ถึงที่หมายและลำดับข้อมูลที่ส่งโดยปราศจากข้อมูลซ้ำซ้อน โดยในชั้นนี้มีโปรโตคอลหลัก 2 ตัวคือทีซีพีและยูดีพี
3. **Internet Layer** ในชั้นนี้มีการกำหนดดาต้าแกรม (Datagram) และทำการหาเส้นทางการส่งการทำงานในชั้นนี้จะเป็นแบบคอนเนคชั่นเลสเนื่องจากไม่มีการเชื่อมต่อระหว่างต้นทางกับปลายทางก่อนดาต้าแกรมแต่ละตัวจะสามารถเลือกเส้นทางไปโดยอิสระและไม่มีการรับประกันความถูกต้องข้อมูลหรือลำดับการส่ง
4. **Network Interface Physical Layer** ทำหน้าที่ควบคุมตัวกลางที่ใช้สื่อสารข้อมูลและรูปแบบการเชื่อมต่อในทางกายภาพ ชั้นนี้จะแบ่งข้อมูลออกเป็นส่วนๆ เรียกว่าเฟรมหรือแพ็คเกจและส่งข้อมูลที่ได้ไปยังปลายทางที่เชื่อมต่อกันอยู่บนเครือข่ายเดียวกัน

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

ตารางที่ 2.4 โครงสร้างของชุดโปรโตคอล ทีซีพี/ไอพี เปรียบเทียบกับ แบบจำลอง โอเอสไอ

2.5.3 ข้อแตกต่างระหว่างชุดโปรโตคอล ทีซีพี/ไอพีและ โอเอสไอ

1. **ลำดับการติดต่อสื่อสารของชั้นเลเยอร์** ในรูปแบบโอเอสไอนั้นจะกำหนดลำดับชั้นการสื่อสารที่เป็นลำดับขั้นตอนการติดต่อที่แน่นอน โดยเฉพาะการอินเตอร์เฟสระหว่างชั้นเลเยอร์ ซึ่งทำให้รูปแบบโอเอสไอสามารถเป็นระบบเปิดสำหรับระบบเครือข่ายคอมพิวเตอร์ทั่วไป เพราะไม่ว่าจะมีการเปลี่ยนแปลงโปรโตคอลในเลเยอร์ชั้นใดก็ตามจะไม่มีผลกระทบต่อการสื่อสารเลเยอร์ชั้นถัดไป
2. **การติดต่อสื่อสารระหว่างเครือข่ายหรือการติดต่อผ่านอินเทอร์เน็ต** คือการติดต่อสื่อสารข้อมูลระหว่างระบบคอมพิวเตอร์ 2 ระบบที่ไม่สามารถติดต่อสื่อสารกันได้โดยผ่านทางเครือข่ายการสื่อสารข้อมูลเพียงเครือข่ายเดียวได้ ต้องอาศัยเครือข่ายตั้งแต่ 2 เครือข่ายขึ้นไปในการติดต่อสื่อสารกันและเครือข่ายเหล่านี้อาจจะมีลักษณะของเครือข่ายที่ต่างกันได้ ความแตกต่างในเรื่องของอินเทอร์เน็ตระหว่างชุดโปรโตคอล ทีซีพี/ไอพีกับรูปแบบโอเอสไอก็คือในชุดโปรโตคอลทีซีพี/ไอพีจะใช้ไอพีซึ่งในรูปแบบโอเอสไอจะเรียกว่าไอพีว่าโปรโตคอลเน็ตเวิร์ค
3. **การบริการการเชื่อมต่อการสื่อสาร (Connection Service)** ในชุดโปรโตคอลทีซีพี/ไอพีนั้น จะมีการบริการการเชื่อมต่อการสื่อสารระหว่างต้นทางและปลายทาง 2 แบบ คือบริการแบบคอนเนคชันเลสและแบบคอนเนคชันโอเรียนเต็ดส่วนในรูปแบบโอเอสไอจะให้ความสำคัญเฉพาะกับการบริการแบบคอนเนคชันโอเรียนเต็ดเท่านั้น
4. **โปรโตคอลควบคุมการจัดการสื่อสาร** ในชุดโปรโตคอลทีซีพี/ไอพีจะใช้โปรโตคอลทีซีพีเป็นโปรโตคอลสำหรับควบคุมการสื่อสาร กำหนดตำแหน่งต้นทางและปลายทาง และอื่นๆ กับข้อมูลซึ่งในรูปแบบโอเอสไอนั้นจะแบ่งแยกการควบคุมการสื่อสารออกจากกันโดยใช้โปรโตคอลเซสชั่นและโปรโตคอลทรานสปอร์ตตามลำดับ

2.5.4 ลักษณะของการติดต่อ

แบ่งออกเป็น 2 ชนิดคือ

1. **Connection-Oriented** คือการติดต่อที่ต้องมีการเชื่อมกระบวนการที่จะมีการส่งหรือรับข้อมูลซึ่งใช้คำว่าวงจรเสมือน (Virtual Circuit) เพราะว่าจะทำงานเสมือนมีวงจรต่ออยู่ระหว่างกระบวนการถึงแม้ว่าข้อมูลนี้อาจจะจ่าย Packet-Switching Network บริการชนิดนี้ส่วนมากจะใช้ในกรณีที่มีข่าวสารต้องการมากกว่าหนึ่งข่าวสาร ดังนั้นสามารถแบ่งขั้นตอนการทำงานออกเป็น
    - ขั้นการสร้างการติดต่อ (Connection Establishment)
    - ขั้นการส่งผ่านข้อมูล (Data Transfer)
    - ขั้นยกเลิกการติดต่อ (Connection Termination)
2. **Connectionless** หรือดาต้าแกรม คือจะไม่มีขั้นการสร้างการติดต่อ และขั้นการยกเลิกการติดต่อ แต่จะมีขั้นการส่งผ่านข้อมูลเพียงอย่างเดียว โดยข้อมูลซึ่งเรียกว่าดาต้าแกรมจะถูกส่งจากระบบหนึ่งไปสู่ระบบหนึ่งอย่างเป็นอิสระโดยไม่ขึ้นอยู่กับดาต้าแกรมอื่น

2.5.5 ความสัมพันธ์ของทีซีพี และยูดีพีกับอินเทอร์เน็ต

ตามสถาปัตยกรรมของชุดข้อกำหนดรูปแบบทีซีพี/ไอพีระดับชั้นทรานสปอร์ตจะมีบริการส่งข้อมูลอยู่ 2 ประเภท คือทีซีพีและยูดีพี

ความแตกต่างของทีซีพีกับยูดีพีอยู่ที่วิธีการสื่อสารระหว่างเครื่องคอมพิวเตอร์สองเครื่อง ทีซีพีต้องสร้างการติดต่อให้ได้ก่อนแล้วจึงส่งข้อมูลและยกเลิกเส้นทางที่ต่อกันเมื่อจบการส่งข้อมูล ส่วนยูดีพีจะไม่มีการสร้างการติดต่อ แต่จะใช้ไอพีแอดเดรสของคอมพิวเตอร์ปลายทางใส่รวมกับข้อมูลแล้วส่งออกไป ข้อมูลจะเดินตามไอพีแอดเดรสด้วยเส้นทางที่ไม่แน่นอน ขึ้นอยู่กับความเหมาะสมจนกว่าจะถึงคอมพิวเตอร์ปลายทาง

ทีซีพีเป็นวิธีการสื่อสารที่มีความเชื่อมั่นได้มากกว่ายูดีพีทั้งการส่งและการตอบรับข่าวสารวิธีของยูดีพีจะไม่มีการรับประกันว่าข่าวสารจะไปถึงผู้รับแต่ยูดีพีมีวิธีการที่จะตรวจสอบโดยให้เครื่องคอมพิวเตอร์ปลายทางส่งข่าวสารการตอบรับกลับมาเมื่อได้รับข่าวสารที่ส่งไปให้ ทางด้านผู้ส่งจะตั้งเวลาว่านานเท่าใดควรมีการตอบกลับรับกลับมา ถ้าเกินเวลาแล้วยังไม่มีการตอบรับกลับมาก็สันนิษฐานได้ว่าข่าวสารสูญหายยูดีพีก็จะทำการส่งข่าวสารเดิมให้ใหม่

เครือข่ายขนาดใหญ่เช่นอินเทอร์เน็ตซึ่งมีเครื่องคอมพิวเตอร์บนเครือข่ายเป็นจำนวนมากการใช้ทีซีพีที่ต้องสร้างการติดต่อและยกเลิกอยู่บ่อยๆ จึงเป็นไปไม่ได้ในทางปฏิบัติดังนั้นอินเทอร์เน็ต จึงใช้การส่งข่าวสารด้วยยูดีพีแทน

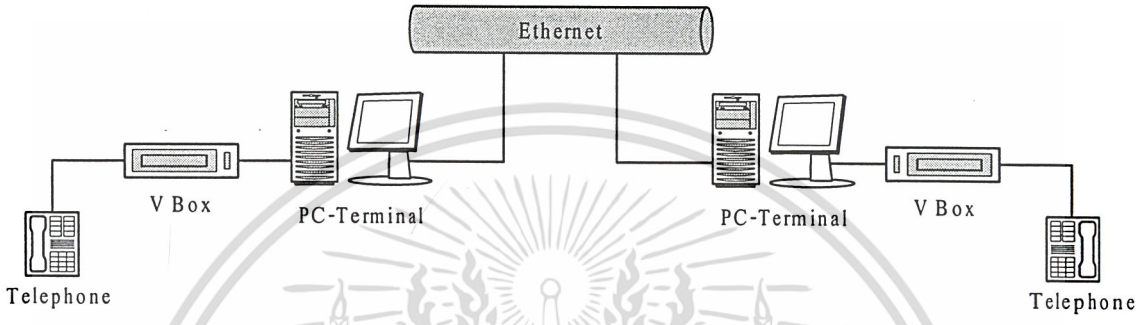
การบริการที่ดำเนินการด้วยชุดข้อกำหนดรูปแบบทีซีพี/ไอพีต้องออกแบบการให้บริการเป็นทีซีพีหรือยูดีพีอย่างใดอย่างหนึ่งเท่านั้น การบริการอย่างเดียวจะให้บริการทั้งสองประเภทไม่ได้ เช่นการเข้าระบบจากระยะไกล และการโอนย้ายไฟล์ข้อมูลเอฟทีพีใช้การให้บริการแบบทีซีพีส่วนทีเอฟทีพี (Trivial File Transfer Protocol) ใช้การให้บริการแบบยูดีพี

บทที่ 3

การคำนวณ การสร้างและการออกแบบ

3.1 โครงสร้างรวมของระบบ

ในการออกแบบระบบเพื่อจำลองการทำงานระบบโทรศัพท์แสดงได้ดังรูป



รูปที่ 3.1 โครงสร้างของระบบ

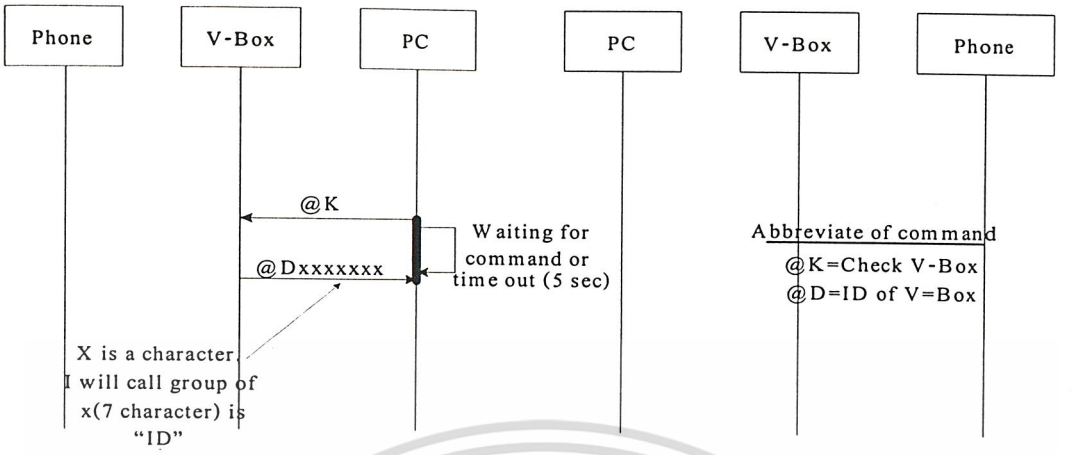
จากรูปประกอบไปด้วย

1. วี-บ็อกซ์ (V-box)
2. พีซี-เทอร์มินัล (PC-Terminal)

การทำงาน

ในส่วนของวี-บ็อกซ์จะเป็นส่วนสำหรับการเชื่อมต่อกับเครื่องโทรศัพท์ จะเป็นตัวกลางในการแปลงเสียงให้เป็นข้อมูลดิจิตอล แล้วทำการส่งมายังเครื่องพีซี-เทอร์มินัลหรือรับข้อมูลจากพีซี-เทอร์มินัลที่อยู่ในรูปสัญญาณดิจิตอล มาแปลงให้อยู่ในรูปของสัญญาณเสียง นอกจากนี้ยังต้องมีหน้าที่สร้างสัญญาณที่เกี่ยวข้องกับสัญญาณโทรศัพท์ ได้แก่ สัญญาณให้หมุน (Dial Tone) สัญญาณไม่ว่าง (Busy Tone) และสัญญาณเรียกกลับ (Ring back Tone) เป็นต้น

สำหรับพีซี-เทอร์มินัลจะนำข้อมูลที่ต้องการส่งมาทำการแปลงให้อยู่ในรูปแพ็กเกจเพื่อส่งออกไป หรือแปลงจากแพ็กเกจที่รับเข้ามาให้เป็นข้อมูล วี-บ็อกซ์และใช้ในการเชื่อมต่อระหว่างพีซี-เทอร์มินัลด้วยกัน



รูปที่ 3.2 แสดงในส่วนเริ่มต้นการติดต่อ

จากรูป จะแสดงการติดต่อในช่วงต้นจะมีการตรวจสอบสถานะของวี-บ็อกซ์ด้วย การตรวจสอบจะทำการส่ง @K จากนั้นวี-บ็อกซ์จะส่ง @D พร้อมกับหมายเลขเครื่อง (ID) หลังจากนั้นจะรอรับคำสั่งต่อไป คำสั่งจะมีอยู่ 2 อย่างด้วยกันคือ

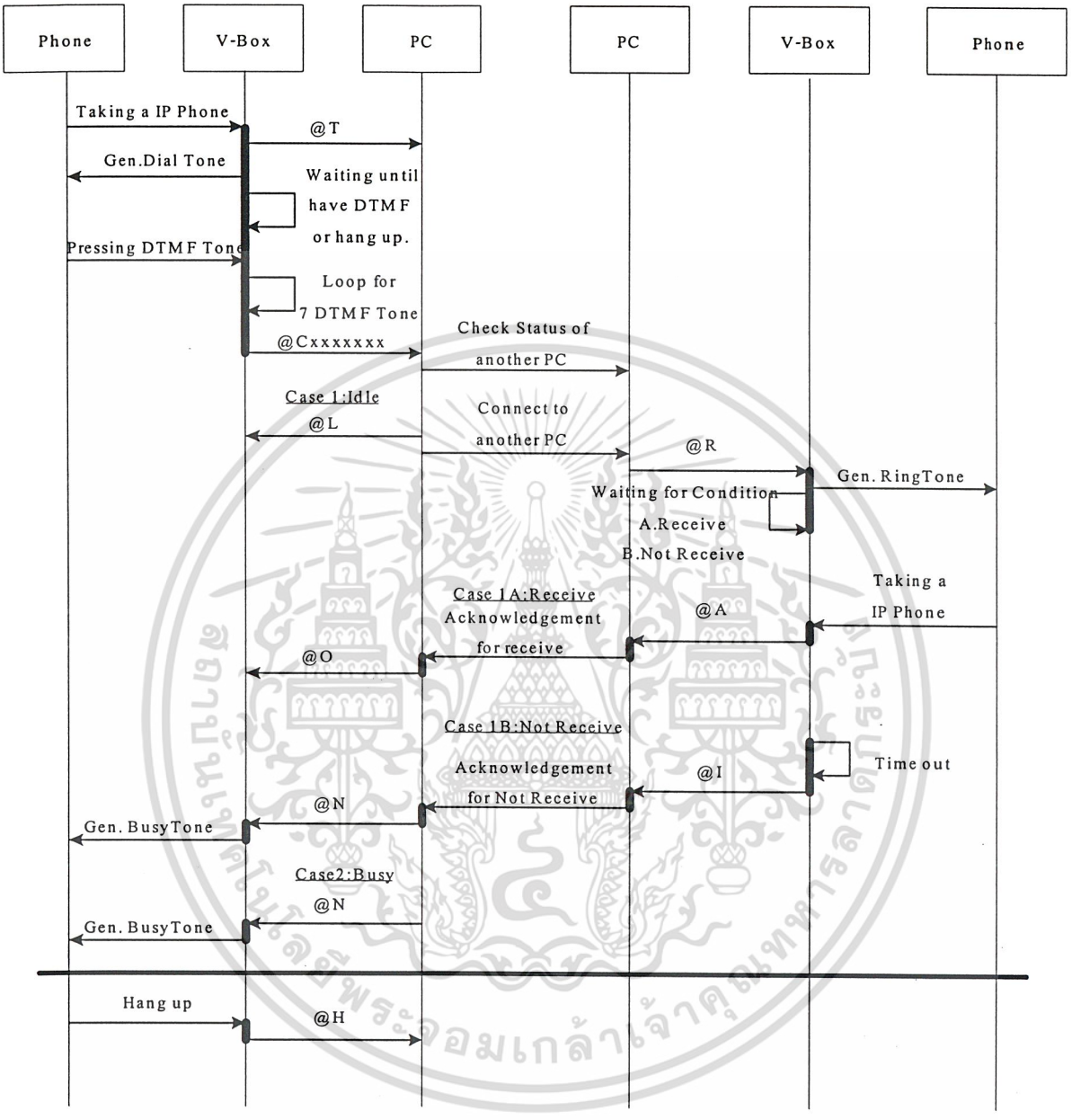
1. การโทรออก
2. การรับสายเรียกเข้า

การโทรออกเมื่อทำการยกหูโทรศัพท์ขึ้นมาจะเกิดการอินเตอร์รับขึ้น วี-บ็อกซ์ก็จะส่ง @T ไปยังพีซี-เทอร์มินัลและส่งสัญญาณให้หมุนไปยังเครื่องรับโทรศัพท์ จากนั้นวี-บ็อกซ์จะรอรับการกดหมายเลขที่ทำการติดต่อซึ่งจะทราบเลขหมายโดยการติดต่อจากสัญญาณดีทีเอ็มเอฟ (DTMF) ที่เกิดจากการกดปุ่มเลขหมายบนเครื่องโทรศัพท์ เมื่อกดหมายเลขครบแล้ววี-บ็อกซ์จะส่ง @C แล้วตามด้วยหมายเลขที่ต้องการเรียกไปยังพีซี-เทอร์มินัลเพื่อติดต่อกับพีซีเทอร์มินัลอีกเครื่องหนึ่ง

มี 2 สภาวะที่เกิดขึ้นได้คือ

1. เครื่องที่ต้องการติดต่อว่างสามารถทำการติดต่อได้ พีซี-เทอร์มินัลจะทำการส่ง @L ให้เครื่องวี-บ็อกซ์ วี-บ็อกซ์ก็จะทำการส่งสัญญาณเรียกกลับไปยังเครื่องโทรศัพท์แล้วทำการติดต่อไปยังเครื่องที่ถูกเรียกตามเลขหมายไอพีแล้วจะรอว่าเครื่องปลายทางจะรับสายหรือไม่ ถ้ารับก็จะทำการติดต่อส่งข้อมูลถึงกัน แต่ถ้าไม่รับโดยตรวจสอบจากเวลาที่เรียกจะมีการส่ง @N กลับมา ทางวี-บ็อกซ์ก็จะทำการส่งสัญญาณไม่ว่างไปยังเครื่องโทรศัพท์ ในกรณีที่มีการวางหูเครื่องโทรศัพท์เกิดขึ้นทางวี-บ็อกซ์ก็จะส่ง @H แก่พีซี-เทอร์มินัลเพื่อแจ้งการหยุดใช้บริการ

2. เครื่องที่ต้องการติดต่อไม่ว่าง พีซี-เทอร์มินัลจะทำการส่ง @N ให้เครื่องวี-บ็อกซ์ วี-บ็อกซ์จะทำการส่งสัญญาณไม่ว่างให้กับเครื่องโทรศัพท์

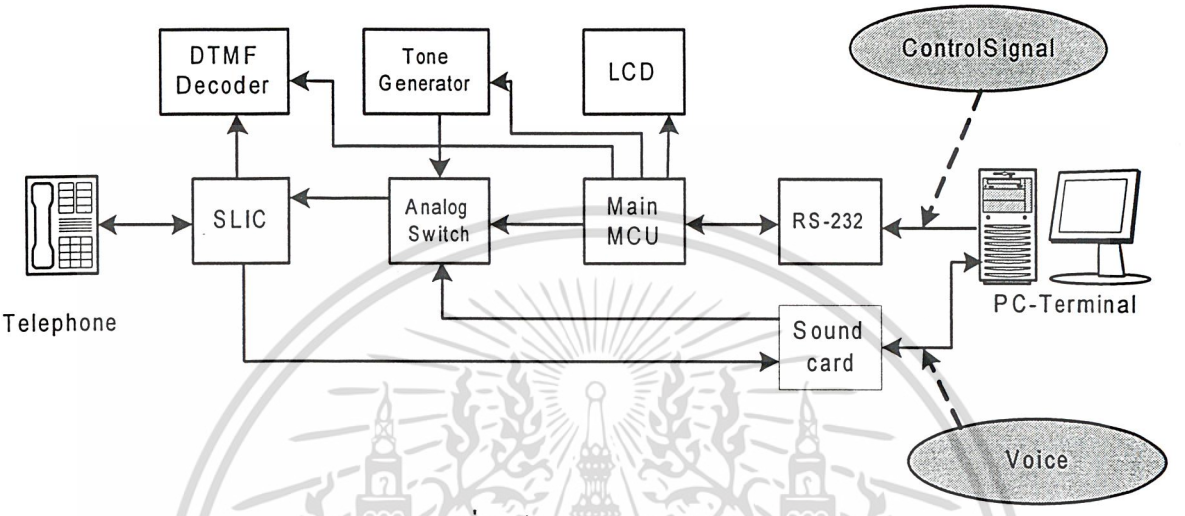


รูปที่ 3.3 แสดงการติดต่อระหว่าง V-Box กับ PC Terminal

3.2 การออกแบบในส่วนของวี-บ็อกซ์

3.2.1 การออกแบบในส่วนฮาร์ดแวร์

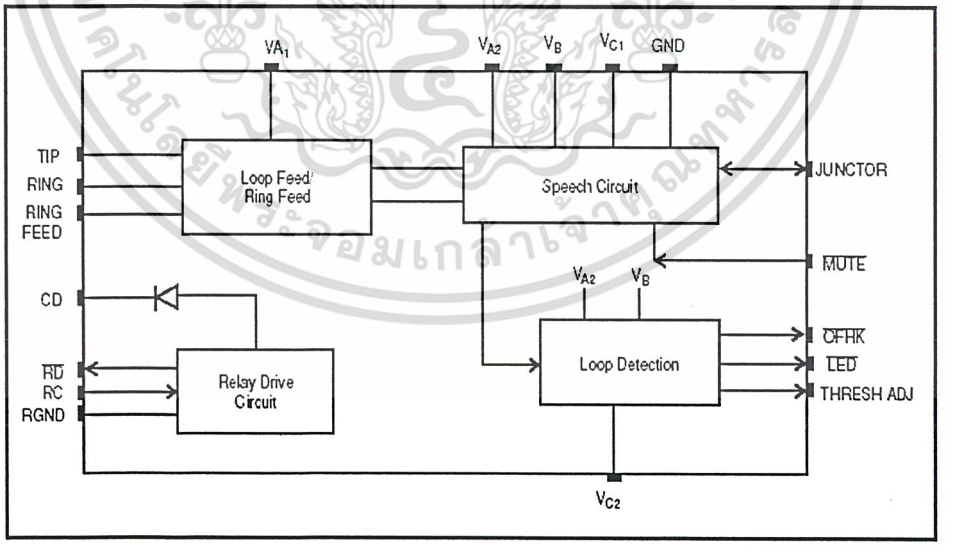
ในส่วนของฮาร์ดแวร์มีไดอะแกรมดังรูป



รูปที่ 3.4 ไดอะแกรมของ V-Box

สามารถอธิบายได้ดังนี้

1. SLIC (Subscriber Line Interface Circuit) ทำหน้าที่เชื่อมต่อกับโทรศัพท์ ใช้ไอซีMH88500 ของบริษัทไมเทลส่วนประกอบภายในของไอซีแสดงได้ดังรูป 3.5



รูปที่ 3.5 โครงสร้างภายใน SLIC เบอร์ MH88500

ภายในประกอบด้วยส่วนต่างๆ สามารถอธิบายการทำงานของแต่ละส่วน มีรายละเอียดได้ดังนี้

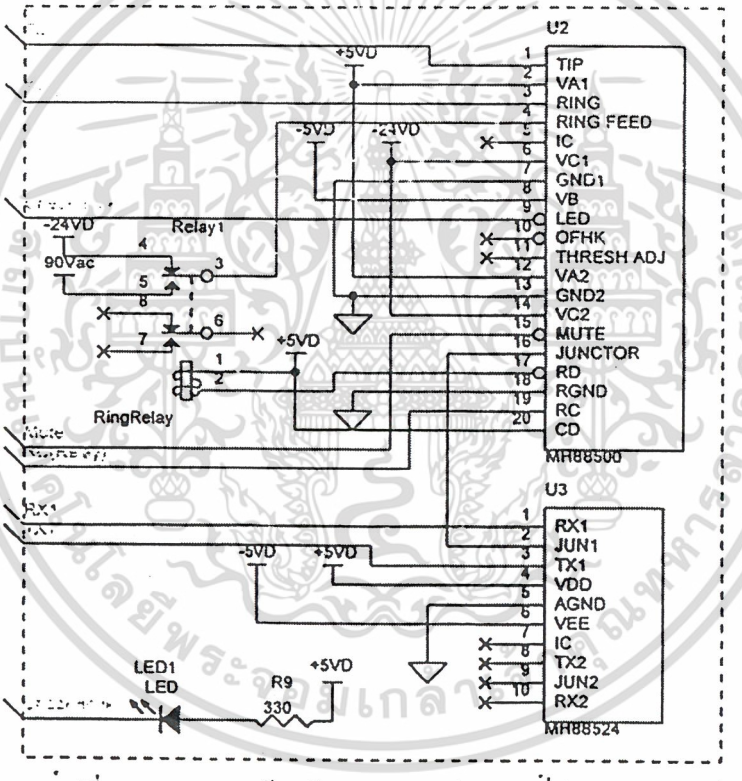
1.1 สปีชเซอร์กิต (Speech circuit) เป็นวงจรที่ใช้ในการแปลงสัญญาณที่เข้ามาทางขา Tipและ Ring ซึ่งเป็นขาสัญญาณที่มีลักษณะเป็น 2 ทิศทางให้มีเอาท์พุตเพียงขา Junctor เพียงขาเดียว

แต่ยังเป็นแบบ 2 ทิศทางและยังมีขา Mute สำหรับตัดสัญญาณเสียงที่ออกไปยังขา Tipและ Ring

1.2 ลูปดีเทคชั่น (Loop detection) เป็นวงจรที่กำหนดค่าความต้านทานระหว่างขา TipและRing เพื่อใช้ในเงื่อนไขการตรวจสอบการยกหูโทรศัพท์

1.3 ไลน์ฟีด / ริงฟีดเซอร์กิต (Line feed / Ring feed circuit) เป็นวงจรที่กำหนดค่ากระแสลูปและสามารถที่จะนำสัญญาณเรียกเข้าที่มีค่าแรงดันขนาด 70-120 Vacป้อนเข้าไปยังขา TipและRing ค่าความต้านทานระหว่างขา TipและRing มีค่า 600โอห์ม

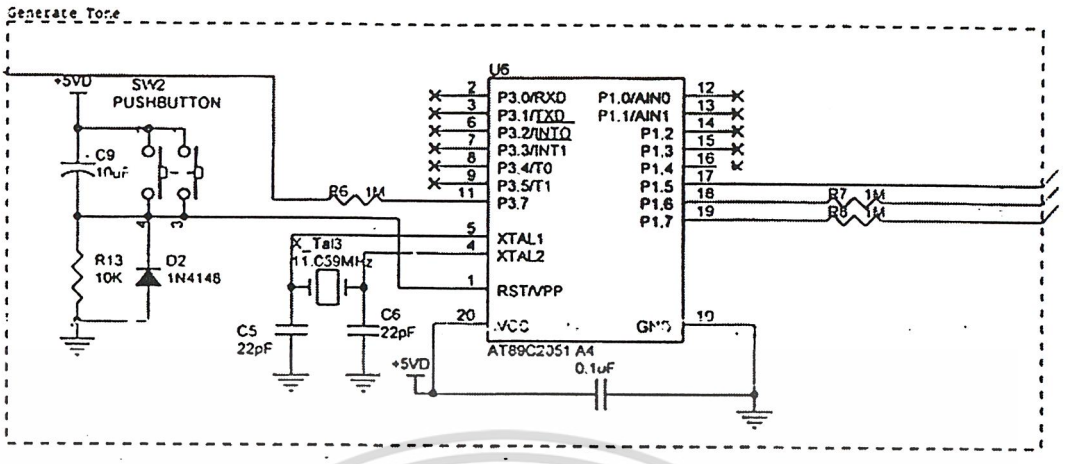
1.4 รีเลย์ไดรฟ์เซอร์กิต (Relay drive circuit) เป็นวงจรสำหรับตัดต่อรีเลย์เพื่อป้อนสัญญาณเรียกเข้า เข้าขา RING FEED โดยมีไดโอดเพื่อป้องกันสัญญาณรบกวนในระหว่างการตัดต่อรีเลย์ที่เกิดจากขดลวดรีเลย์



รูปที่ 3.6 แสดงวงจรป้องกันการรบกวนในขณะที่มีการตัดต่อรีเลย์

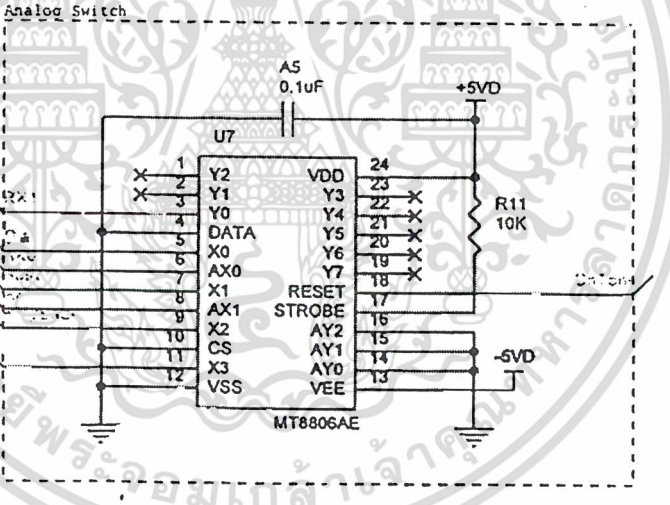
2. โทนเจนเนอเรเตอร์ (Tone Generator) ใช้สร้างสัญญาณโทนของระบบโทรศัพท์ ซึ่งนำไมโครคอนโทรลเลอร์ เบอร์ AT89C2051 ของบริษัทแอทเมลที่มีขนาดเล็กเพียงพอต่อการนำมาใช้ สัญญาณที่ทำการสร้างได้แก่

- ไดอัลโทน (Dial tone)
- สัญญาณไม่ว่าง (Busy tone)
- สัญญาณเรียกกลับ (Ringback tone)



รูปที่ 3.7 แสดงวงจรที่ทำการสร้างสัญญาณโทน

3. อนาล็อกสวิตช์ (Analog switch) ทำหน้าที่เลือกว่าจะนำสัญญาณใดระหว่างสัญญาณโทนกับสัญญาณที่ออกจากพีซีเอ็มฟิลเตอร์ออกไปยัง SLIC ได้ใช้ไอซีเบอร์ MT8806 เป็นไอซีอนาล็อกสวิตซ์แบบอาเรย์ขนาด 8 x 4 สามารถเลือกการตัดต่อว่าจะให้เอาท์พุตออกที่ขาใดก็ได้ ด้วยการเลือกตำแหน่งแถวและคอลัมน์

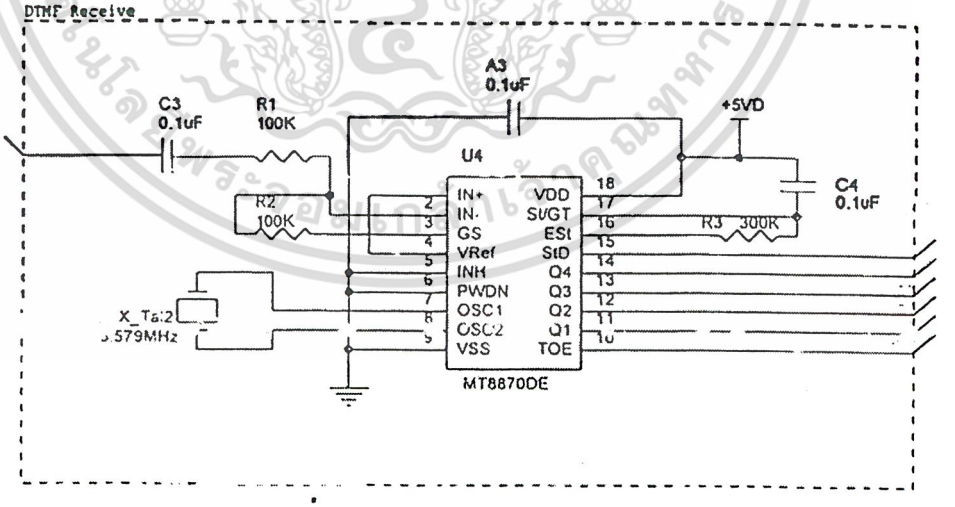


รูปที่ 3.8 แสดงวงจรในส่วนของ Analog switch

4. ดีทีเอ็มเอฟรีซีฟเวอร์ (DTMF receiver) จะเป็นส่วนในการถอดรหัสสัญญาณดีทีเอ็มเอฟให้เป็นสัญญาณดิจิตอลขนาด4บิต เลือกใช้ไอซีเบอร์ MT8870 ซึ่งเป็นที่นิยมใช้กันอย่างแพร่หลาย ภายใน MT8870 ประกอบด้วยส่วนสำคัญ 5 ส่วนคือ

- **ภาคกรองสัญญาณความถี่** ในส่วนนี้จะแยกสัญญาณดีทีเอ็มเอฟที่เข้ามาเป็น2 กลุ่มความถี่ คือ ช่วงความถี่สูงและช่วงความถี่ต่ำโดยใช้วงจรกรองแถบความถี่อันดับ 6 ชนิดสวิตช์คาปาซิเตอร์ (six-order switched capacitor) ซึ่งความถี่ที่แยกได้มี 2 ช่วงคือ ช่วงความถี่สูงและช่วงความถี่ต่ำ

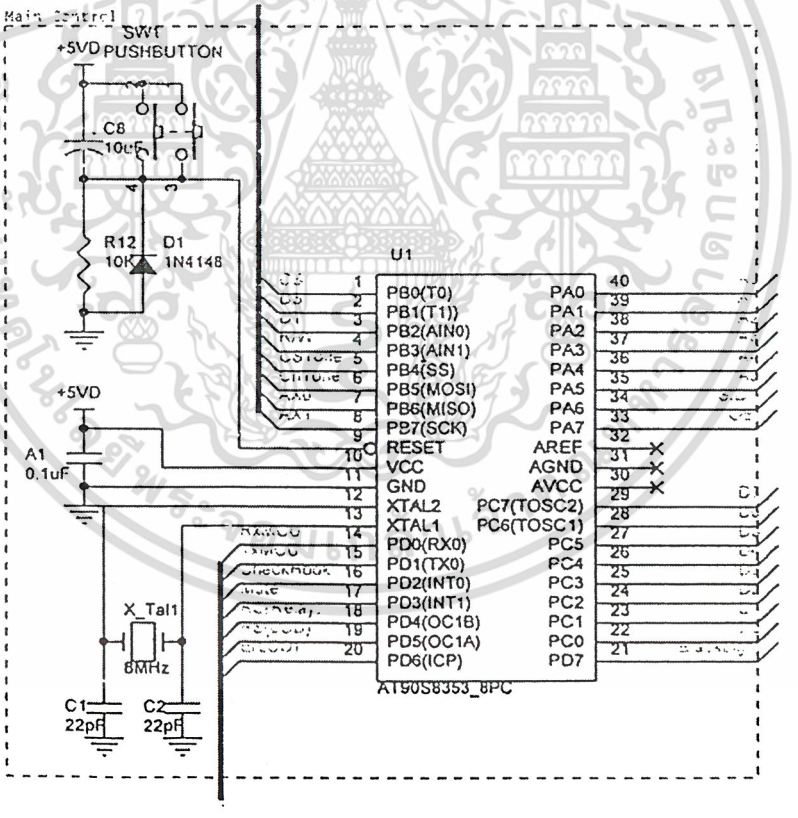
- **ภาคถอดรหัส** ความถี่ดีทีเอ็มเอฟที่ถูกกรองเรียบร้อยแล้วจะผ่านเข้าวงจรถอดรหัสความถี่ออกเป็นตัวเลข โดยใช้เทคนิคการนับแบบดิจิตอลและมีการตรวจสอบความถี่ที่เข้ามาว่าเป็นความถี่มาตรฐานดีทีเอ็มเอฟหรือไม่ เพื่อป้องกันความถี่อื่นเข้ามาผสม
- **ภาคตรวจสอบสัญญาณ** ก่อนที่จะมีการถอดรหัสความถี่ออกไปที่เอาท์พุทจะมีการตรวจสอบช่วงความถี่ที่เข้ามาว่ามีระยะเวลาตามที่กำหนดหรือไม่ โดยสังเกตจากระยะเวลาการกดปุ่มโทรศัพท์ ซึ่งต้องกดปุ่มให้มีความถี่ออกมาเป็นช่วงเวลาพอสมควรมิฉะนั้นวงจรส่วนนี้จะไม่รับ โดยถือว่าสัญญาณนั้นไม่ถูกต้อง ส่วนระยะเวลายาวเท่าใดสามารถตั้งได้โดยใช้ ตัวต้านทาน (R) และตัวเก็บประจุ (C) ต่อภายนอก สัญญาณที่ขา ESt จะเป็น “High” นานใกล้เคียงกับระยะเวลาที่มีความถี่ DTMF เข้ามา เมื่อขา ESt เป็น “High”ทำให้ตัวเก็บประจุ Vc จะคายประจุทำให้แรงดัน Vc สูงขึ้นจนถึงค่าเทรสโฮลด์ วงจรถอดรหัสจึงจะถอดรหัสออกมาเป็นตัวเลขขนาด 4 บิตสำหรับคำว่าการ์ดไทม์นั้น หมายถึง ช่วงคาบเวลาความถี่ที่เข้ามา ซึ่งจะต้องนานเท่ากับหรือมากกว่าช่วงเวลาที่เราตั้งไว้ จึงจะได้รับการยอมรับว่าสัญญาณความถี่นั้นถูกหรือพูดได้ว่าเวลาที่เราตั้งไว้โดย RC การ์ดไทม์นั่นเอง เมื่อสัญญาณความถี่เข้ามานานเท่ากับหรือมากกว่าเวลาที่ตั้งไว้จะสามารถแปลงเป็นตัวเลขได้ ถ้าสัญญาณความถี่ที่เข้ามาสั้นกว่าก็จะไม่มีการถอดรหัสเป็นตัวเลขออกไป
- **ภาคขยายสัญญาณความแตกต่างของ** MT8870 วงจรส่วนอินพุตของ MT8870 เป็นภาคขยายออปแอมป์ที่เพิ่มอัตราขยายโดยต่อวงจรภายนอกเพิ่มเข้า แสดงการต่อวงจรภายนอกกับอินพุทซึ่งสามารถคำนวณอัตราความแตกต่างของอินพุทและอิมพีแดนซ์ได้
- **ภาคขยายกำเนิดความถี่** ต่อกับคริสตอลเพื่อผลิตความถี่สำหรับใช้สร้างสัญญาณดีทีเอ็มเอฟ



รูปที่ 3.9 แสดงวงจรของ DTMF

5. เมนเอ็มซียู (Main MCU) มีหน้าที่ควบคุมการทำงานทั้งหมดของวี-บ็อกซ์ได้นำไมโครคอนโทรลเลอร์ AT90S8535 (AVR) มาใช้ มีคุณสมบัติดังนี้
   - เป็นไมโครคอนโทรลเลอร์ที่สถาปัตยกรรมแบบริสค์ (RISC)
   - มีชุดคำสั่งทั้งหมด 118 คำสั่ง

- แฟลชเมมโมรี่ขนาด 8 กิโลไบต์ (In-System Programmable Flash)
- อีพรอมขนาด 512 ไบต์
- แรมขนาด 512 ไบต์
- ADC ขนาด 10 บิต 8 ช่อง
- มีรีจิสเตอร์ 32 x 8
- อินพุท/เอาท์พุท 32
- สูงสุดได้ 8 เอ็มไอพีเอส
- ไทม์เมอร์/เคานท์เตอร์ 8 บิต 2 ช่อง
- ไทม์เมอร์/เคานท์เตอร์ 16 บิต 1 ช่อง
- 3 พีดับบลิวเอ็ม (PWM)
- วอชด็อกไทม์เมอร์ (Watch dog Timer)
- อินเตอร์รัปท์ภายในและภายนอก(External Interrupt and Internal Interrupt)
- Programmable serial UART



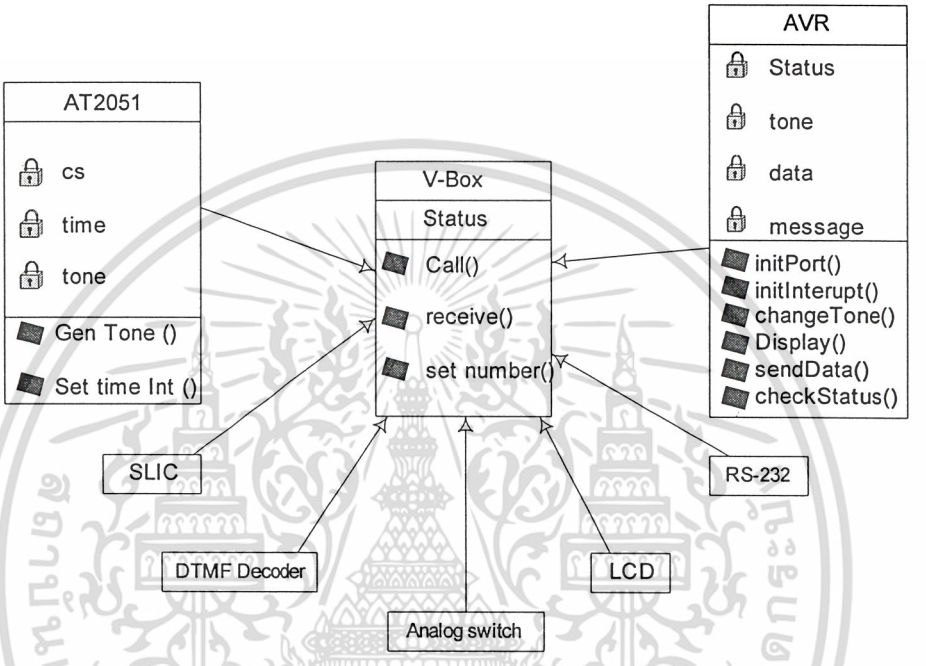
รูปที่ 3.10 แสดงวงจรในส่วนของโครงสรร้างหลักของวงจร

6. แอลซีดี มีหน้าที่แสดงข้อความ ในโครงงานนี้ใช้ขนาด 16 ตัวอักษร 1 บรรทัดมาใช้ในการแสดงผลต่างๆ
7. RS232 เป็นตัวในการติดต่อสื่อสาร ตามมาตรฐาน RS232 ได้นำไอซีเบอร์ DS275 ซึ่งมีข้อดีคือมีขนาดเล็ก ไม่ต้องมีการนำตัวเก็บประจุมาต่อเพิ่ม

3.2.2 การออกแบบซอฟท์แวร์ของตัววี-บ็อกซ์

3.2.2.1 แผนภาพคลาสของวี-บ็อกซ์

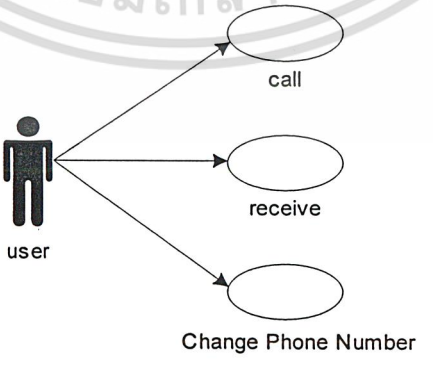
จากการวิเคราะห์ความต้องการเบื้องต้นของวี-บ็อกซ์และการทำงานหลักๆที่วี-บ็อกซ์ต้องการจะตอบสนองให้กับผู้ใช้งานได้ จึงได้ทำการออกแบบโดยใช้แผนภาพคลาสได้ดังรูปที่ 3.11



รูปที่ 3.11 แสดงแผนภาพคลาสของวี-บ็อกซ์

3.2.2.2 Use Case Diagram

จากการวิเคราะห์การทำงานเบื้องต้นสามารถกำหนดการทำงานของวี-บ็อกซ์แสดงโดยใช้แผนภาพ Use Case ได้ดังรูปที่ 3.12

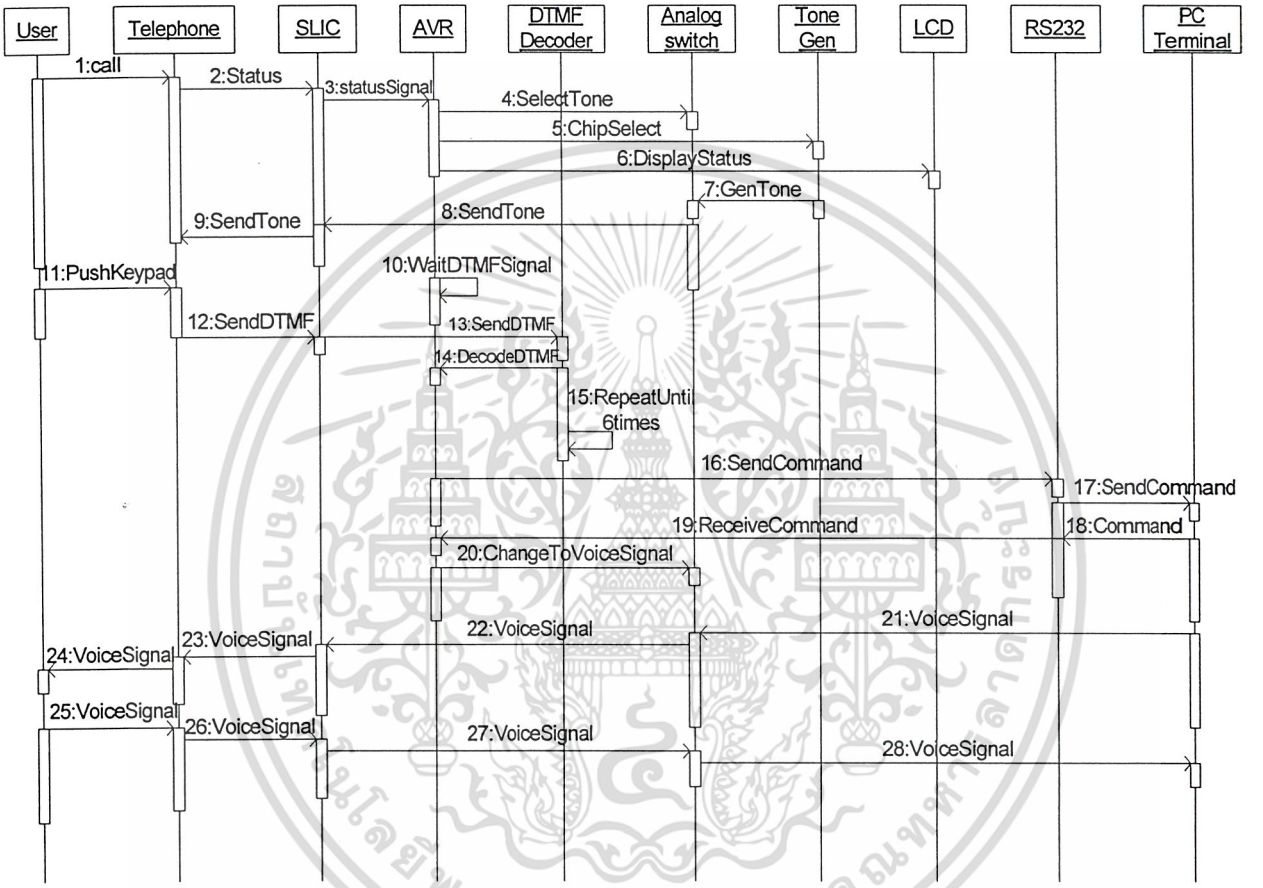


รูปที่ 3.12 แสดง Use Case ของวี-บ็อกซ์

3.2.2.3 ตัวอย่าง Scenarios ที่สำคัญในการทำงานของวี-บ็อกซ์

การโทรออกจากวี-บ็อกซ์เป็น Scenarios แสดงการทำงานเมื่อมีการโทรออกไปยังวี-บ็อกซ์อีกเครื่องหนึ่งเพื่อทำการติดต่อสื่อสารถึงกัน แสดงโดยใช้แผนภาพ Sequence ได้ดังรูปที่ 3.13

การรับโทรศัพท์เมื่อมีการเรียกเข้ามาจากวี-บ็อกซ์เป็น Scenarios แสดงการทำงานเมื่อมีการเรียกเข้ามาจากวี-บ็อกซ์อีกเครื่องหนึ่งเพื่อทำการติดต่อสื่อสารถึงกัน แสดงโดยใช้แผนภาพ Sequence ได้ดังรูปที่ 3.13



รูปที่ 3.13 แสดง Scenarios ในการโทรออกของ วี-บ็อกซ์

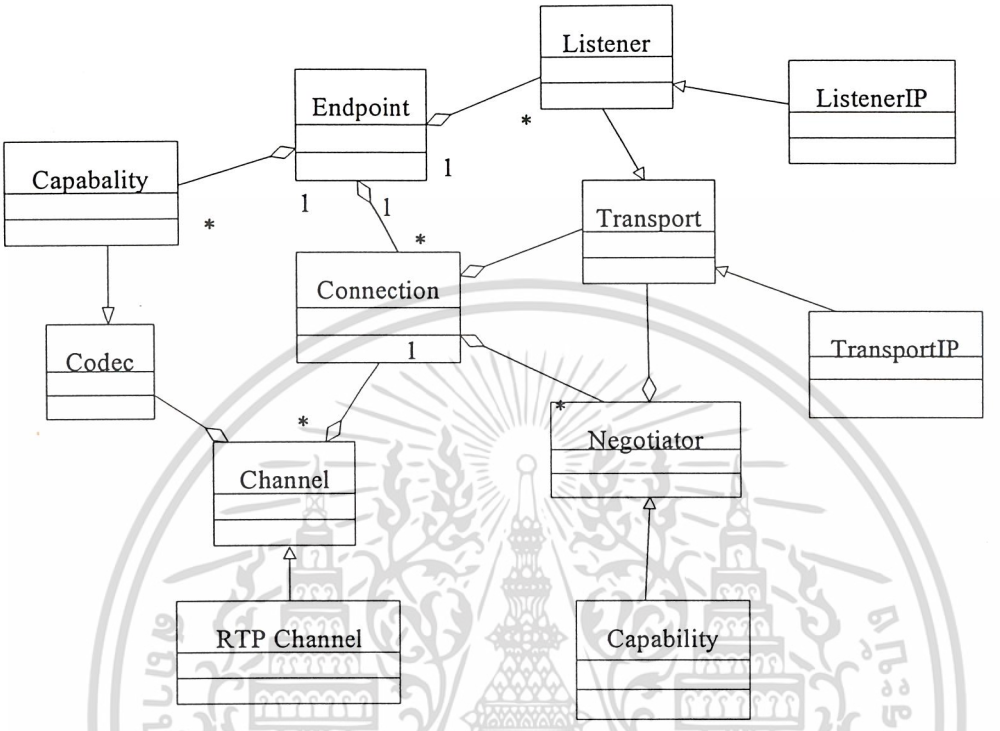
3.3 การออกแบบในส่วนของพีซี-เทอร์มินัล

ในส่วนของพีซี-เทอร์มินัลนี้ เราใช้ลีนุกส์เป็นระบบปฏิบัติการหลักของระบบ เหตุที่นำลีนุกส์มาใช้นี้เพราะลีนุกส์เป็นระบบปฏิบัติการที่แจกฟรีและเปิดเผยซอร์สโคด ทำให้ไม่ผิดลิขสิทธิ์และง่ายแก่การหามาใช้งาน

ซอฟท์แวร์ที่เรานำมาทำเป็นพีซี-เทอร์มินัลนี้เราใช้ซอร์สโคดของ H.323 ซึ่งได้รับการพัฒนามาจากผู้ที่มีความรู้ในเรื่องของมาตรฐานนี้เป็นอย่างดี ดังนั้นเรานำโปรแกรมนี้มาใช้ ในการนำมาใช้นี้เราจะใช้โปรแกรมทั้งหมด 3 ส่วน ซึ่งได้แก่

1. ส่วน PWLib เป็นการนำไลบรารี่ที่พอร์เทเบิลบนวินโดว์สมาใช้บนลีนุกส์

2. ส่วน Openh323 เป็นส่วนที่รวบรวมคลาสโครงสร้างโปรโตคอลของ H.323 ทั้งหมด
3. ส่วน ohphone เป็นแอพพลิเคชั่นที่นำมาตราฐาน H.323 มาใช้งาน



รูปที่ 3.14 แสดงโครงสร้างคลาสไลบรารี่ของ H.323

จากการดัดแปลงนำเอาซอร์สโคดมาใช้งานนั้นได้ทำการดัดแปลงในส่วนของ ohphone ซึ่งเป็นแอพพลิเคชันที่เรานำมาเป็นโปรแกรมหลักบนพีซี แต่ตัว ohphone ตัวเดียวไม่สามารถทำงานได้เอง แต่ยังต้องการโปรแกรมในส่วนที่ 1 และ 2 ตามที่ได้กล่าวมาข้างต้น

ในส่วนของโปรแกรม ohphone เราได้เพิ่มในส่วนของการติดต่อกับวี-บ็อกซ์

ส่วนที่ทำการติดต่อกับวี-บ็อกซ์เราจะใช้การติดต่อผ่านทางพอร์ตอนุกรม โดยจะมีการกำหนดคำสั่งต่างๆ ที่ทำการติดต่อกันได้แก่

@T = วี-บ็อกซ์ส่งไปยังพีซี เมื่อมีการยกหูโทรศัพท์

@C = วี-บ็อกซ์ส่งไปยังพีซี เมื่อวี บ็อกซ์ได้รับสัญญาณดีทีเอ็มเอฟ

@R = วี-บ็อกซ์รับจากพีซี เพื่อให้รู้ว่ามีการโทรศัพท์เข้ามา

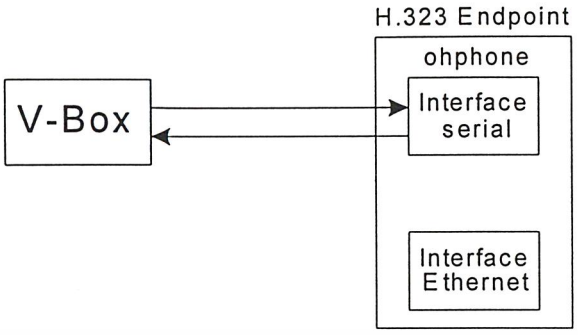
@A = วี-บ็อกซ์ส่งไปยังพีซี เมื่อวี-บ็อกซ์ฝ่ายที่ได้รับคำสั่ง @R ยกหูรับโทรศัพท์

@I = วี-บ็อกซ์ส่งไปยังพีซี เมื่อวี-บ็อกซ์ฝ่ายที่ได้รับคำสั่ง @R ไม่รับโทรศัพท์ตามเวลาที่กำหนด

@O = วี-บ็อกซ์จะได้รับจากพีซี เมื่ออีกฝ่ายได้ยกหูรับโทรศัพท์

@N = วี-บ็อกซ์จะได้รับจากพีซี เมื่ออีกฝ่ายไม่ได้รับหูโทรศัพท์

@H = วี-บ็อกซ์จะส่งไปยังพีซี เมื่อได้เกิดการวางหูโทรศัพท์



รูปที่ 3.15แสดงลักษณะการเชื่อมต่อระหว่างพีซี-เทอร์มินัลกับวี-บ็อกซ์

ลำดับการทำงานและคำสั่งที่รับ-ส่งระหว่างวี-บ็อกซ์กับพีซีสามารถดูได้จากรูป 3.16 ในการเขียนโปรแกรมที่ทำหน้าที่รับ-ส่งคำสั่งระหว่างพีซี-เทอร์มินัลและวี-บ็อกซ์นั้นเราจะใช้หลักการแบบ Canonical Input Processing ในการรอรับข้อมูลจะไม่วนลูปเพื่อรอรับข้อมูลทีละตัว แต่จะทำการอ่านค่ามาเก็บไว้ก่อน และจะทำการส่งข้อมูลมาเมื่อได้รับอักขระแอสกีโคดไลน์ฟีดหรือว่าเครื่องหมายที่ทำการจบสายอักขระ แต่ในการส่งข้อมูลออกทางซีเรียลพอร์ตนั้นจะส่งออกปกติ โดยในระบบปฏิบัติการลีนุกส์นั้นจะมีการกำหนดค่าในการส่งถ่ายข้อมูลผ่านซีเรียลพอร์ตดังนี้

```
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<termios.h>
#include<stdio.h>

/* baudrate settings are defined in <asm/termbits.h>, which is
included by
<termios.h> */
#define BAUDRATE B9600

/* change this definition for the correct port */
#define MODEMDEVICE "/dev/ttyS0"
#define_POSIX_SQURCE 1 /* POSIX compliant source */
#define FALSE 0
#define TRUE 1

main(){
  int fd,res;
  struct termios oldtio,newtio:
  char buf[2];

/*
  Open modem device for reading and writing and not as controlling
tty
  Because we don't want to get killed if linenoise sends CTRL-C.
*/

fd = open(MODEMDEVICE, O_RDWR | O_NOCTTY | O_NONBLOCK);
if (fd<0) {perrer(MODEMDEVICE); exit(-1);}

tcgetattr(fd,&oldtio);/* save current serial port settings */
bzero(&newtio, sizeof(newtio)): /* clear struct for new port
settings */
```

```
      /*
        BAUDRATE: Set bps rate. You could also use cfsetispeed and
cfsetospeed.
        CRTSCTS : output hardware flow control (only used if the cable
has
            All necessary lines. See sect. 7 of Serial-HOWTO)
        CS8     : 8n1 (8bit,no parity,1 stopbit)
        CLOCAL      : local connection, no modem control
        CREAD     : enable receiving characters
      */
        newtio.c_cflag = BAUDRATE | CRTSCTS | CS8 | CLOCAL | CREAD;

      /*
        IGNPAR  : ignore bytes with parity errors
        ICRNL     : map CR to NL (otherwise a CR input on the other
computer
                    Will not terminate input)
        Otherwise make device raw (no other input processing)
      */
        newtio.c_iflag = IGNPAR | ICRNL;

      /*
        Raw output.
      */
        newtio.c_oflag = 0;


      /*
        ICANON  : enable canonicai input
        Disable all echo functionality, and don,t send signals to
calling program
      */
        newtio.c_lflag = ICANON;

      /*
        now clean the modem line and activate the settings for the port
      */

        tcflush(fd, TCIFLUSH);
        tcsetattr(fd, TCSANOW,&ncwtio);

      }
```

เมื่อเราทำการกำหนดค่าเริ่มต้นให้กับซีเรียลพอร์ตแล้ว เราก็สามารถนำไปใช้งานได้โดยการรับค่าจะสามารถรับได้ดังนี้

Res = read(fd,buf,(ค่าจำนวนที่ต้องการรับ));

ค่าที่รับได้จะอยู่ใน buf ส่วนในการส่งค่าออกทางซีเรียลพอร์ตนั้น จะสามารถทำได้ดังนี้

Res = write(fd,buf,(ค่าจำนวนที่ต้องการส่งออก));

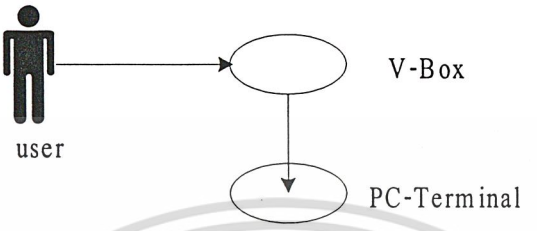
เราจะใส่ค่าที่ต้องการส่งออกไว้ใน buf

ที่ผ่านมาได้กล่าวถึงการใช้ ลีนุกส์ในการติดต่อกับฮาร์ดแวร์ โดยลีนุกส์จะมีการจัดการเรื่องติดต่อกับฮาร์ดแวร์ที่ดี ทำให้สามารถทำการติดต่อได้โดยง่าย

3.3.1 การออกแบบซอฟท์แวร์ของพีซี-เทอร์มินัล

3.3.1.1 Use Case Diagram

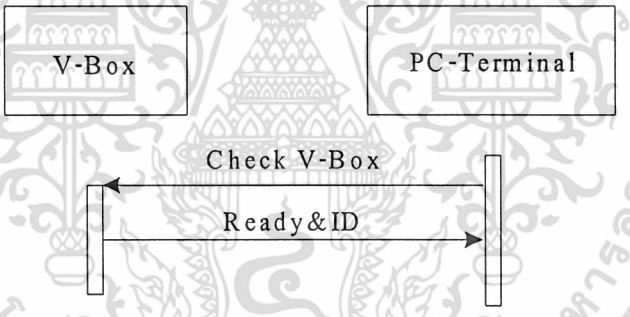
ในการออกแบบในส่วนของ Use Case Diagram นี้เราสามารถแสดงได้ดังนี้



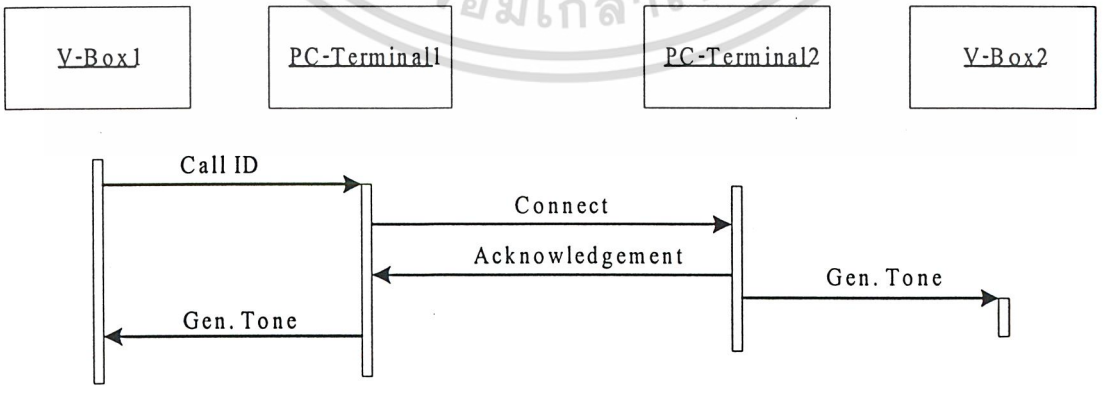
รูปที่ 3.16 แสดง Use Case Diagram ของพีซี-เทอร์มินัล

3.3.1.2 Sequence Diagram

ลำดับการทำงานของโปรแกรมนี้ เราจะสามารถแบ่งได้เป็นสองส่วนคือ ในส่วนของการเริ่มต้นและส่วนของการเรียกและรับโทรศัพท์ ซึ่งสามารถแสดงได้ดังรูปที่ 3.17 และ 3.18



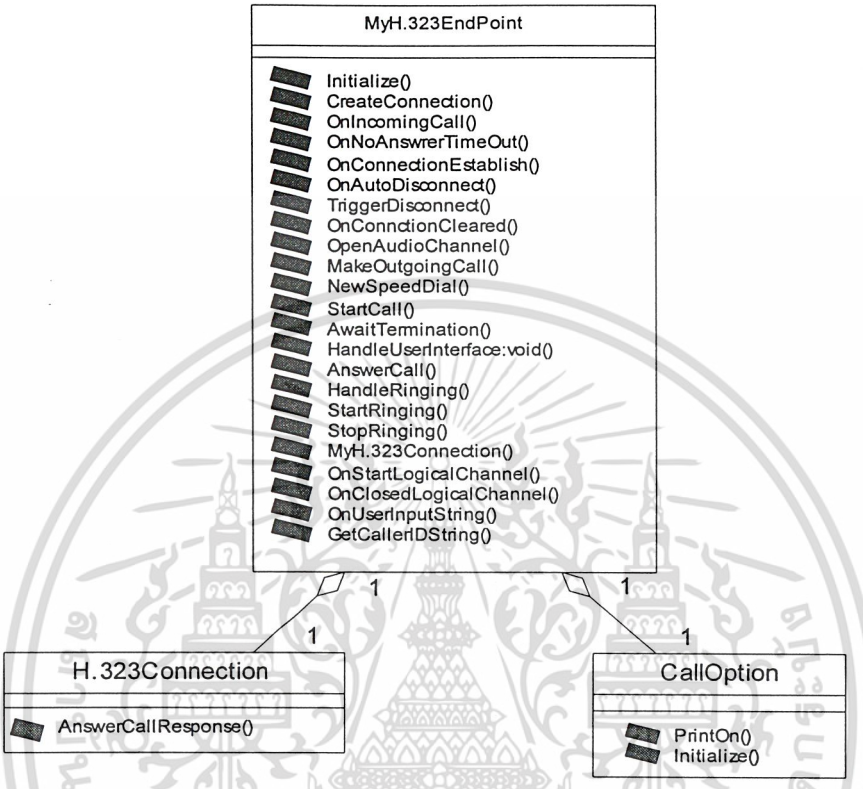
รูปที่ 3.17 แสดงลำดับการทำงานของโปรแกรมในช่วงเริ่มต้น



รูปที่ 3.18 แสดงลำดับการทำงานของการเรียกและการรับโทรศัพท์

3.3.1.3 Class Diagram

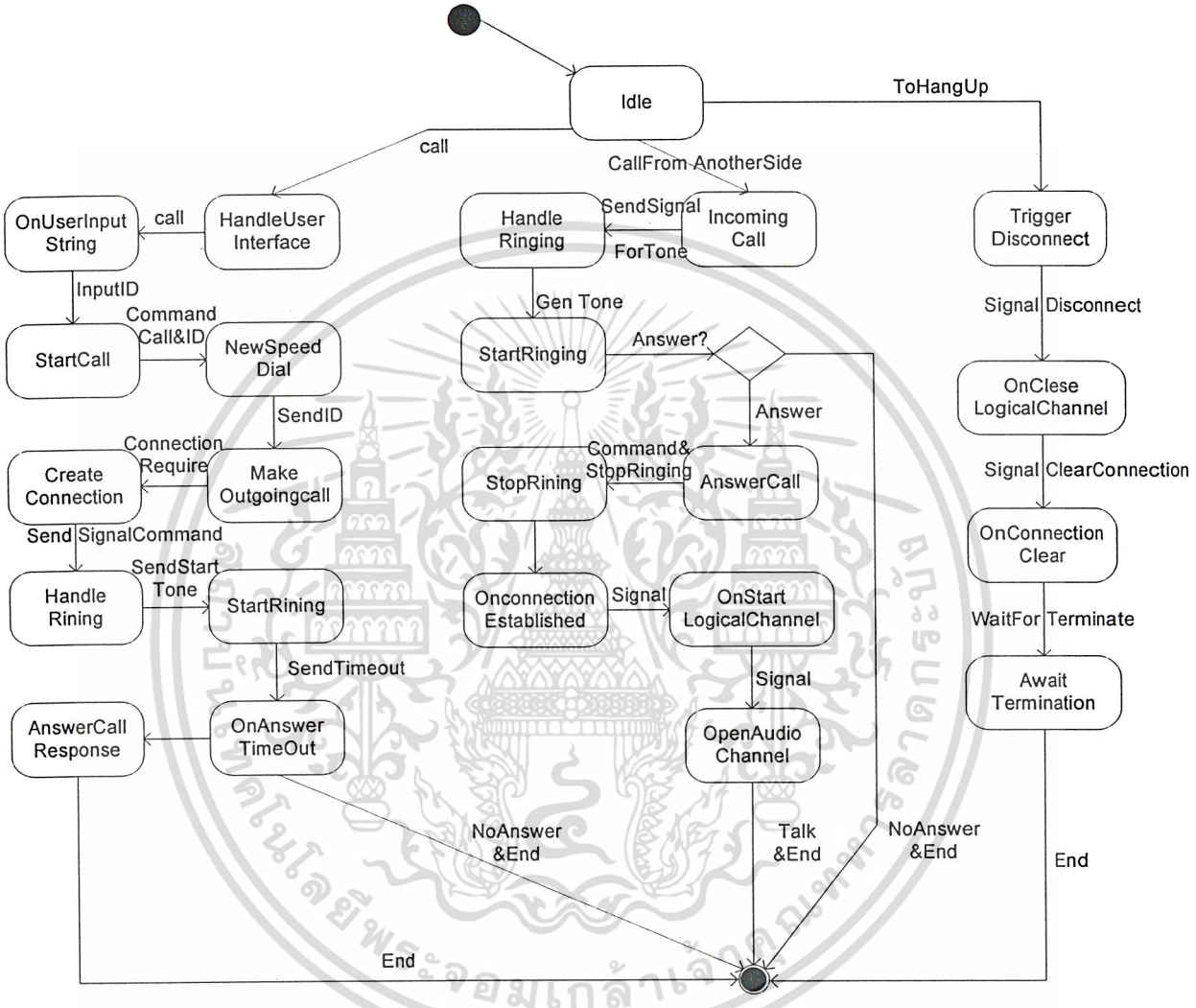
รูปที่ 3.19 แสดงคลาส ไดอะแกรมของพีซี-เทอร์มินัลที่ใช้ในการสร้างซอฟท์แวร์



รูปที่ 3.19 แสดงคลาสไดอะแกรมของพีซี-เทอร์มินัล

3.3.1.4 State Diagram

State Diagram นี้จะแสดงการทำงานทั้งหมดของระบบซึ่งได้แก่ การรับ, การเรียกหรือว่าการวางโทรศัพท์ซึ่งสามารถแสดงได้ดังรูปที่ 3.20



รูปที่ 3.20 แสดงสเตทไดอะแกรมของพีซี-เทอร์มินัล

บทที่ 4

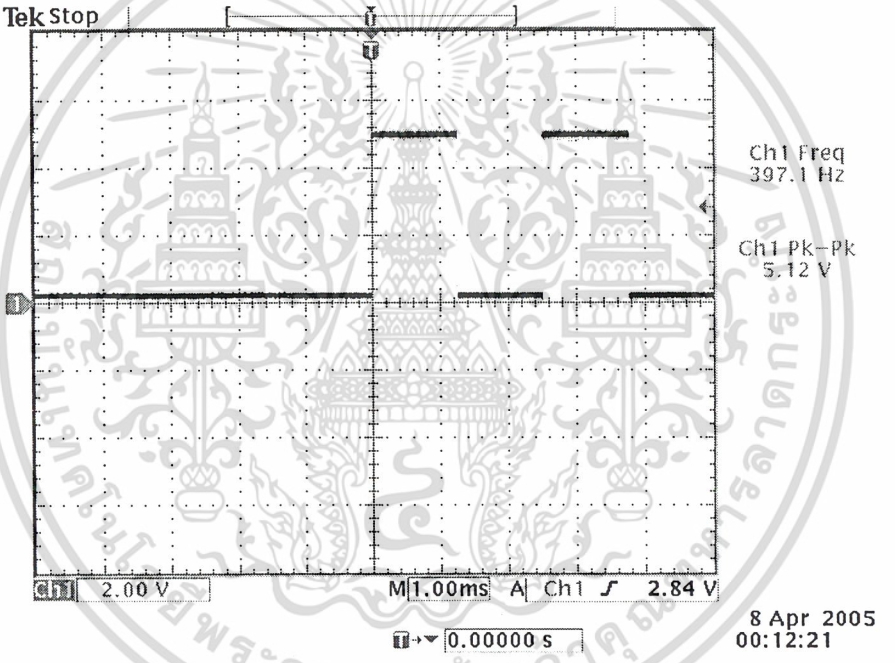
ผลการทดลอง

4.1 ผลการทดลองในส่วนวี-บ็อกซ์

4.1.1 วงจรกำเนิดสัญญาณในระบบโทรศัพท์ (Tone Generator)

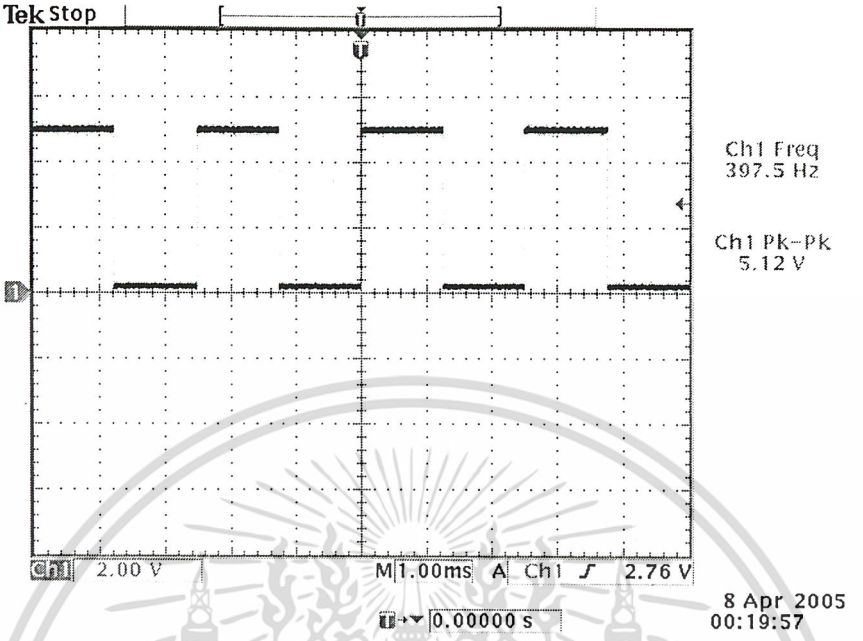
ไอซีเบอร์ AT89C2051 เป็นไมโครคอนโทรลเลอร์ซึ่งต้องทำการโปรแกรมก่อนจึงสามารถทำงานได้ โดยเมื่อโปรแกรมแล้ว ป้อนค่าอินเทอร์รัพที่ขา 17 แล้ววัดค่าสัญญาณต่างๆ ได้ดังนี้

1. สัญญาณเรียกกลับ (Ring Back Tone) โดยวัดที่ขา 18 ได้ผลเป็นสัญญาณความถี่400Hz ดังรูป



รูปที่ 4.1 สัญญาณเรียกกลับ

2. สัญญาณไม่ว่าง (Busy Tone) โดยวัดที่ขา 17 ได้ผลดังรูป



รูปที่ 4.2 สัญญาณไม่ว่าง

4.1.2 วงจรอนาล็อกสวิตช์

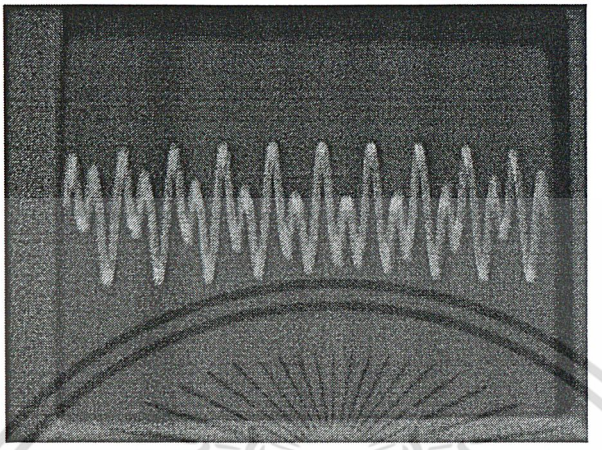
ไอซีอนาลอกสวิตช์เป็นอาเรย์สวิตช์ดังนั้นต้องทำการป้อนอินพุทเป็นโลจิก “0” แทนค่าด้วยแรงดันไฟตรง 0 V และ “1” แทนค่าด้วยแรงดันไฟตรง 5 V เพื่อเลือกสัญญาณต่างๆ ส่งให้กับวงจร SLIC ที่ขา Y0ได้ผลดังตารางต่อไปนี้

AX0 (ขา 6)	AX1 (ขา 8)	X0 (สัญญาณอินพุท)
“0”	“0”	Dial Tone
“0”	“1”	Ringing, Ring Back Tone
“1”	“0”	Busy Tone

ตารางที่ 4.1 แสดงการเลือกสัญญาณส่งไปยัง SLIC

4.1.3 วงจรดีทีเอ็มเอฟ

ผลการทดลองวัดสัญญาณ DTMF เมื่อกดแป้นโทรศัพท์หมายเลข 2



Volt/div = 0.1mV Time/div = 1mSec

รูปที่ 4.3 สัญญาณ DTMF เมื่อกดแป้นโทรศัพท์หมายเลข 2

สัญญาณ DTMF เมื่อออกจากไอซี MT8870 วัดที่ขา 11-14 จะได้เป็นระดับสัญญาณคงที่โดยแรงดันไฟตรง 1.8V แทนโลจิก 1 แรงดันไฟตรง 0V แทนโลจิก 0 สรุปได้ดังตารางต่อไปนี้

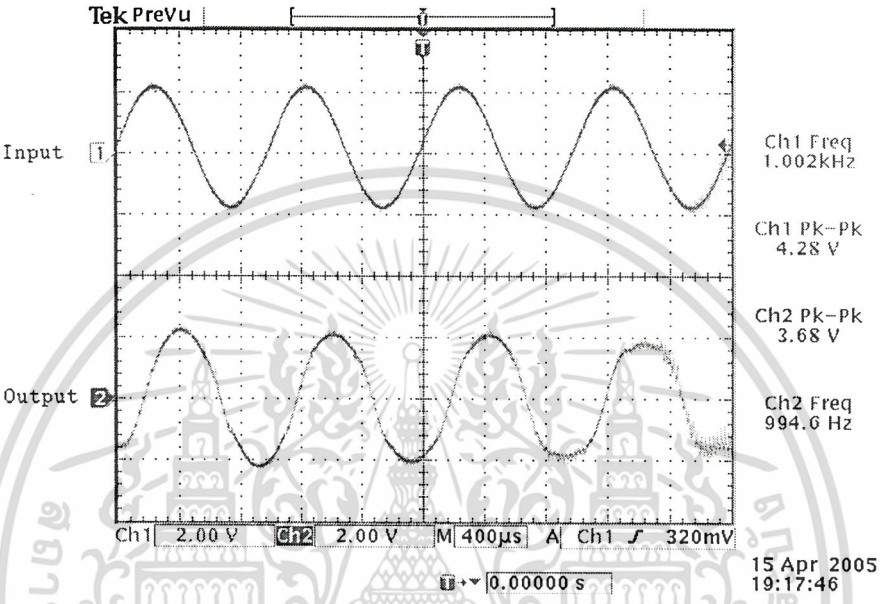
NUMBER	Q4 (14)	Q3 (13)	Q2 (12)	Q1 (11)
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
0	1	0	1	0
*	1	0	1	1
#	1	1	0	0

รูปที่ 4.4 ตารางแสดงค่าการถอดรหัสสัญญาณ DTMF

4.1.4 ผลการทดลองโดยรวมของวี-บ็อกซ์

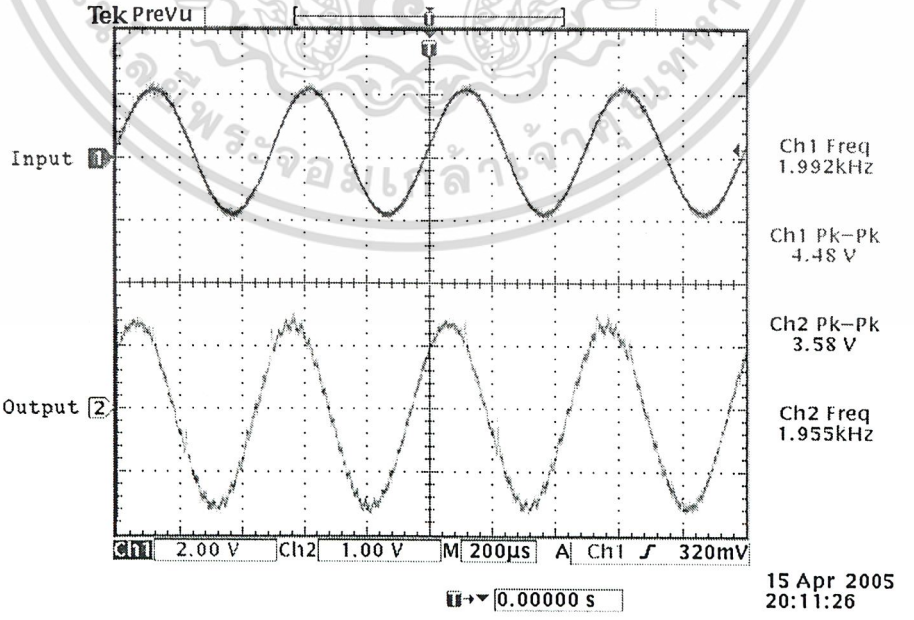
ต่อวงจรทั้งหมดเข้าด้วยกัน ทำการป้อนสัญญาณ Sine ในช่วงที่อยู่ในย่านความถี่เสียงพูดของมนุษย์ ได้ผลดังนี้

1. ป้อนสัญญาณ Sine 1kHz 1Vpp เข้าที่ขา 3 ของไอซี MH88524 แล้ววัดเอาท์พุททางขา 1 ของไอซี MH88524 ได้ผลดังรูปต่อไปนี้



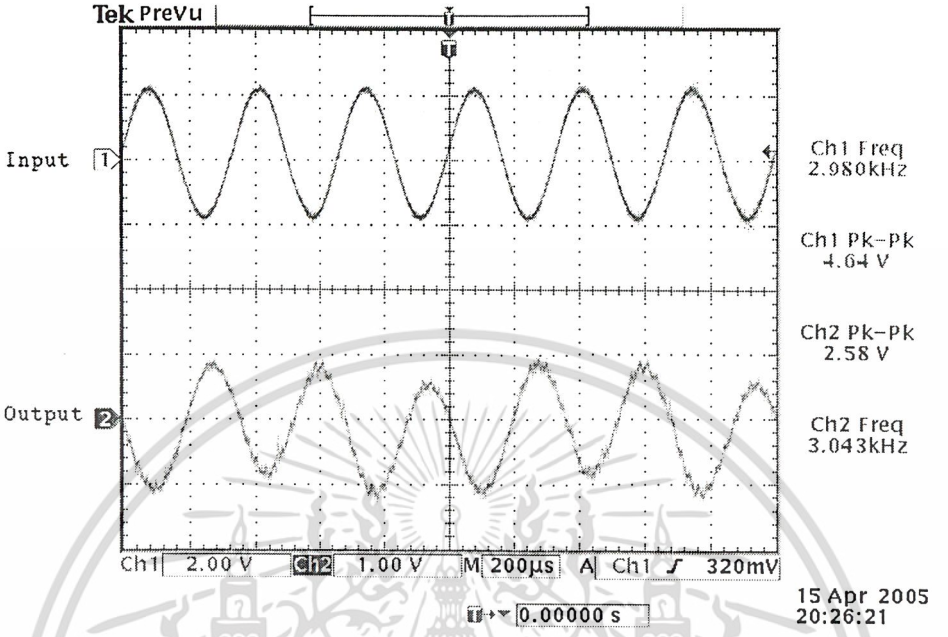
รูปที่ 4.5 ผลการทดลองเมื่อป้อนสัญญาณ Sine ความถี่ 1 kHz 1Vpp

2. ป้อนสัญญาณ Sine 2 kHz 1Vpp เข้าที่ขา 3 ของไอซี MH88524 แล้ววัดเอาท์พุททางขา 1 ของไอซี MH88524 ได้ผลดังรูปต่อไปนี้



รูปที่ 4.6 ผลการทดลองเมื่อป้อนสัญญาณ Sine ความถี่ 2 kHz 1Vpp

3. ป้อนสัญญาณ Sine 3 kHz 1Vpp เข้าที่ขา 3 ของไอซี MH88524 แล้ววัดเอาท์พุททางขา 1 ของไอซี MH88524 ได้ผลดังรูปต่อไปนี้

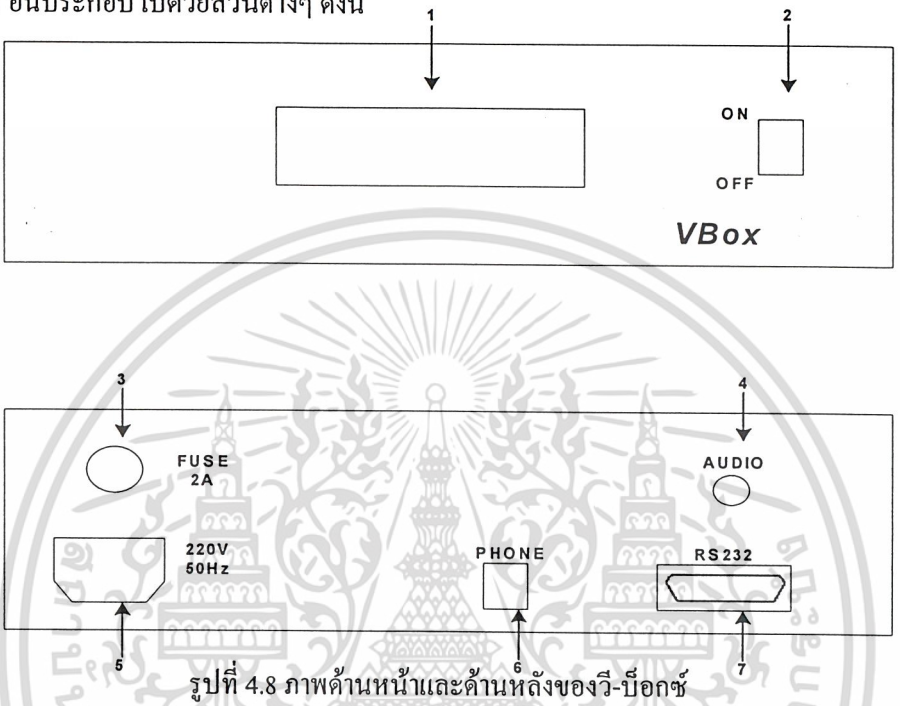


รูปที่ 4.7 ผลการทดลองเมื่อป้อนสัญญาณ Sine ความถี่ 3 kHz 1Vpp

4.2 ผลการทดลองในส่วนของ พีซี-เทอร์มินัล

4.2.1 การติดตั้งวี-บ็อกซ์ก่อนทำการทดลอง

ส่วนประกอบของฮาร์ดแวร์ของวี-บ็อกซ์ประกอบไปด้วย ส่วนควบคุมการใช้งานและอินเทอร์เฟส ต่างๆ ดังรูป อันประกอบไปด้วยส่วนต่างๆ ดังนี้



รูปที่ 4.8 ภาพด้านหน้าและด้านหลังของวี-บ็อกซ์

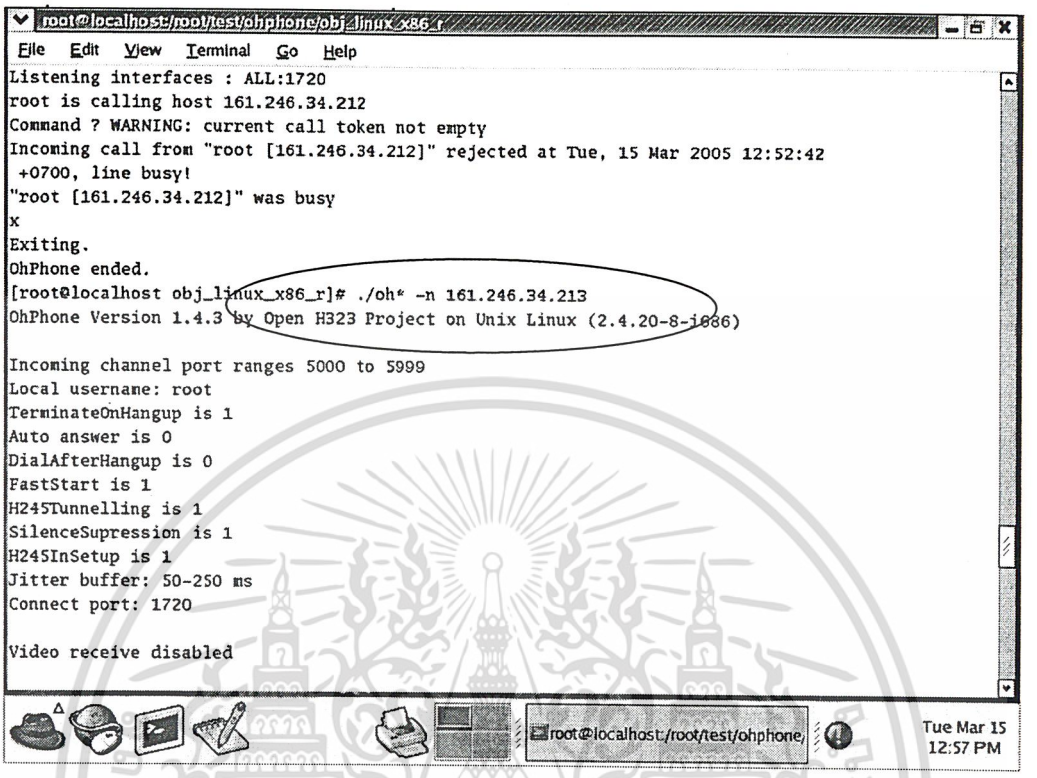
อธิบายส่วนประกอบต่างๆได้ดังรูปที่ 4.9

หมายเลข	ลักษณะ
1	แอลซีดีแสดงผลของคำสั่ง
2	สวิตช์เปิดปิด
3	ฟิวส์สำหรับตัวเครื่องที่ต่อเข้ากับแหล่งจ่ายไฟหลัก
4	ช่องเสียบสัญญาณเสียงเข้ากับการ์ดเสียงของพีซี
5	ช่องเสียบไฟฟ้า 220 Vac เพื่อจ่ายไฟสำหรับตัวเครื่อง
6	ช่องเสียบสำหรับต่อเข้ากับเครื่องโทรศัพท์
7	ช่องเสียบสำหรับต่อเข้าพอร์ตคอม 1หรือคอม 2 ของพีซี

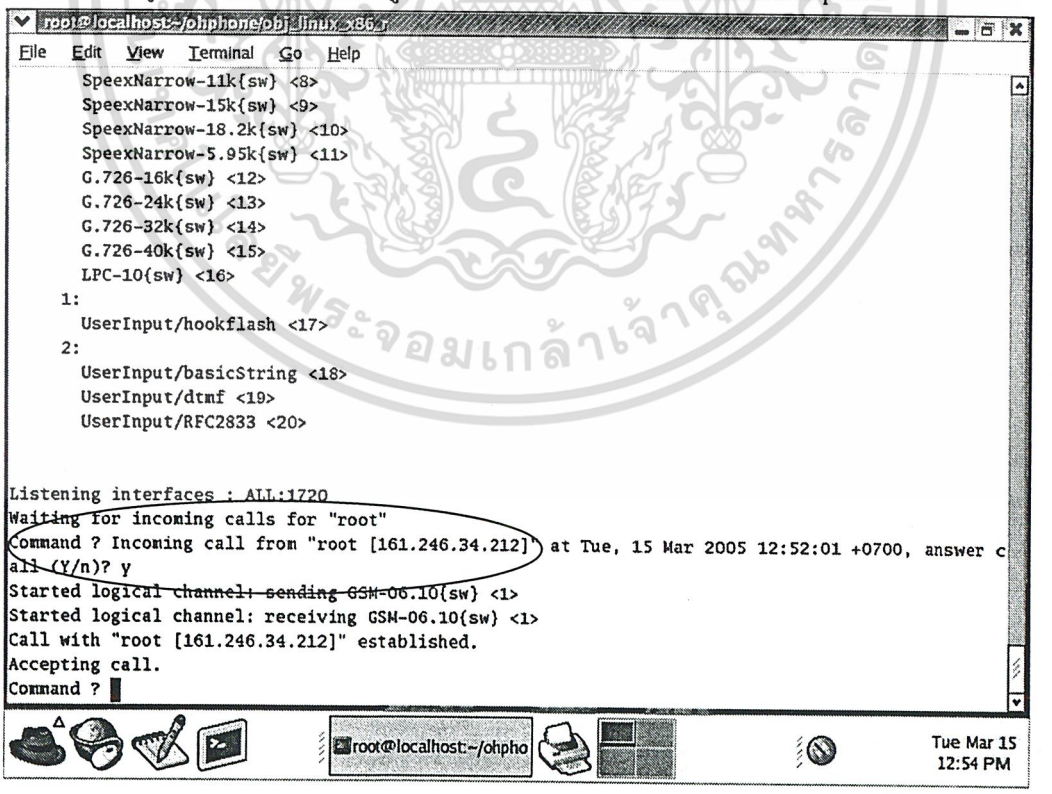
รูปที่ 4.9 ส่วนต่างๆของวี-บ็อกซ์

การต่อใช้งานทำได้โดยการต่อยสายโทรศัพท์เข้ากับวี-บ็อกซ์ที่ช่องเสียบ โทรศัพท์ ต่อสาย 220 Vac เข้ากับตัวเครื่องและต่อสายสัญญาณเสียงเข้ากับการ์ดเสียงต่อผ่านทางช่อง ลำโพงและไมค์ให้ถูกต้อง จากนั้น เปิดสวิตช์เครื่องแล้วทำการเรียกใช้โปรแกรมที่พีซี-เทอร์มินัลเพื่อเริ่มต้นการใช้งาน

4.2.2 ผลการทดลอง



รูปที่ 4.10 แสดงหน้าจอผู้เรียกเมื่อทำการเชื่อมต่อผ่านโปรแกรม ohphone



รูปที่ 4.11 แสดงหน้าจอผู้ถูกเรียกเมื่อทำการเชื่อมต่อผ่านโปรแกรม ohphone

บทที่ 5

สรุปและวิจารณ์

5.1 ขั้นตอนการดำเนินงาน

1. ได้ทำการสร้างฮาร์ดแวร์ (V-Box) ขึ้นมา เพื่อใช้ทำการเชื่อมต่อระหว่างโทรศัพท์และพีซี-เทอร์มินัล โดยฮาร์ดแวร์ที่สร้างขึ้นนี้ ใช้รับสัญญาณควบคุมจากตัวพีซี-เทอร์มินัลและรับสัญญาณเสียงจากโทรศัพท์ไปยังซาว์นการ์ดและจากซาว์นการ์ดมายังเครื่องโทรศัพท์ นอกจากนี้ยังใช้ถอดรหัสสัญญาณดีทีเอ็มเอฟจากการกดหมายเลขโทรศัพท์ และแสดงผลการทำงานต่างๆที่หน้าจอแอลซีดีอีกด้วย ทำให้สะดวกในการใช้งาน

2. ในส่วนของพีซี-เทอร์มินัล ได้ใช้ระบบปฏิบัติการลีนุกส์เป็นหลัก ซึ่งมีการใช้โปรแกรมทั้งหมด 3 ส่วน ดังนี้

2.1 PWLib เป็นฟังก์ชันที่เก็บไลบารี่ของโปรแกรมOhphone

2.2 Openh323 ใช้เก็บคลาสและโครงสร้างของโปรโตคอลH.323

2.3 Ohphone เพื่อใช้ในการส่งผ่านสัญญาณเสียงผ่านระบบอินเตอร์เน็ต ซึ่งในส่วนนี้ได้ทำการเขียนโปรแกรมเพิ่มเข้าไปให้สามารถควบคุมการติดต่อกับฮาร์ดแวร์ผ่านซีเรียลพอร์ตได้

5.2 ผลการทดลอง

จากการทดลองพบว่าเมื่อทำการต่อหมายเลขไปยังเครื่องโทรศัพท์ปลายทาง สามารถทำการเชื่อมต่อช่องสัญญาณเสียงและส่งสัญญาณเสียงไปยังอีกฝั่งได้ในระดับที่น่าพอใจ แต่มีปัญหาเกิดขึ้นเกี่ยวกับไซด์โทน(Side Tone) ของโทรศัพท์ คือเมื่อเราพูดไปนั้นจะได้ยินเสียงตัวเองกับมาคล้ายเสียงสะท้อน แต่เสียงสะท้อนนั้นจะเบามากและสัญญาณเสียงที่ส่งผ่านจะมีการหน่วงเวลาประมาณ1วินาที คือเมื่อเราทำการพูดไปยังอีกฝั่งหนึ่งจะได้ยินเสียงในประมาณ1วินาทีต่อมา

5.3 แนวทางการพัฒนาต่อ

สามารถนำโครงงานนี้ไปพัฒนาเป็นโครงข่ายขนาดใหญ่ขึ้น โดยอาจจะทำการเพิ่มส่วนของเซิร์ฟเวอร์เพื่อทำการเก็บข้อมูลของผู้ใช้แต่ละคนและอาจทำให้สามารถรองรับระบบวีดีโอคอนเฟอร์เรนซ์(Video Conference) ได้

ภาคผนวก

```
#include <ptlib.h>

#include "main.h"
#include "gsmcodec.h"
#include "mscodecs.h"
#include "lpc10codec.h"
#include "h323pdu.h"

/* header for canonical */

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>

/* End header for canonical */


#include "version.h"

PCREATE_PROCESS(VBox);

#define     DEFAULT_TIMEOUT   60000
#define     LAST_CALL_COUNT   16

class RingThread : public PThread
{
  PCLASSINFO(RingThread, PThread);

  public:
    RingThread(MyH323EndPoint & ep)
      : PThread(1000, NoAutoDeleteThread),
        endpoint(ep)
      { Resume(); }

    void Main()
      { endpoint.HandleRinging(); }

  protected:
    MyH323EndPoint & endpoint;
};

#define new PNEW


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

VBox::VBox()
  : PProcess("Open H323 Project", "VBox",
             MAJOR_VERSION, MINOR_VERSION, BUILD_TYPE, BUILD_NUMBER)
{
}


VBox::~VBox()
{
}


void VBox::Main()
```

```
{
  //PArgList & args = GetArguments();
  PConfigArgs args(GetArguments());

  args.Parse(
             "h-help."
             "j-jitter:"             "-no-jitter."
             "l-listen."
             "n-no-gatekeeper."
           , FALSE);

  int verbose = 255;

  if (verbose >= 3)
    cout << GetName()
         << " Version " << GetVersion(TRUE)
         << " by " << GetManufacturer()
         << " on " << GetOSClass() << ' ' << GetOSName()
         << " (" << GetOSVersion() << '-' << GetOSHardware() << ")\n\n";

  if (args.HasOption('h') || (!args.HasOption('l') && args.GetCount() ==
0)) {
    cout << "Usage : " << GetName() << " [options] -l\n"
            "      : " << GetName() << " [options] [-p host]
hostname/alias\n"
          "\n   where:  hostname/alias = Remote host/alias to call\n"

             "\nOptions:\n"
             "  -h --help                : Display this help message.\n"
             "  -l --listen              : Only listen for incoming
calls\n"
             "  -n --no-gatekeeper       : Disable gatekeeper
discovery.\n"
          << endl;
    return;
  }


  BOOL hasMenu = !args.HasOption("disable-menu");

  args.Save("save");

  MyH323EndPoint * endpoint = new MyH323EndPoint;
  if (endpoint->Initialise(args, verbose, hasMenu)) {
      if (args.HasOption('l'))
        if (verbose >= 2)
          cout << "Waiting for incoming calls for \"" << endpoint-
>GetLocalUserName() << "\"\n";
      endpoint->AwaitTermination();
  }

  delete endpoint;

  if (verbose >= 3)
    cout << GetName() << " ended." << endl;
}


//////////////////////////////////////////////////////////////////
```

```
BOOL MyH323EndPoint::Initialise(PConfigArgs & args, int _verbose, BOOL
_hasMenu)
{
  PINDEX i;

  verbose = _verbose;
  hasMenu = _hasMenu;
  uiState = uiDialtone;

  // get ports
  WORD port       = H323ListenerTCP::DefaultSignalPort;
  WORD listenPort = port;
  if (!args.GetOptionString("port").IsEmpty())
    port = (WORD)args.GetOptionString("port").AsInteger();


  defaultCallOptions.jitter      = GetMaxAudioDelayJitter();
  defaultCallOptions.connectPort = port;
  defaultCallOptions.connectRing = 0;

  if (!defaultCallOptions.Initialise(args))
    return FALSE;

  currentCallOptions = defaultCallOptions;

  //////////

  terminateOnHangup    = !args.HasOption('l');
  alwaysForwardParty   = args.GetOptionString('F');
  busyForwardParty     = args.GetOptionString('B');
  noAnswerForwardParty = args.GetOptionString('N');
  noAnswerTime         = args.GetOptionString("answer-timeout",
"30").AsUnsigned();
  dialAfterHangup      = args.HasOption("dial-after-hangup");

  noAnswerTimer.SetNotifier(PCREATE_NOTIFIER(OnNoAnswerTimeout));


  if (verbose >= 3) {
    cout << "Local username: " << GetLocalUserName() << "\n"
         << "TerminateOnHangup is " << terminateOnHangup << "\n"
         << "DialAfterHangup is " << dialAfterHangup << "\n"
         << defaultCallOptions
         << endl;

  }

  if (args.HasOption("ringdelay"))
    ringDelay = args.GetOptionString("ringdelay").AsInteger();
  else
    ringDelay = 5;

  // The order in which capabilities are added to the capability table
  // determines which one is selected by default.

  int g711Frames = 30;
  int gsmFrames = 4;

  if (verbose >= 3) {
    cout <<"G.711 frame size: " << g711Frames << endl;
    cout <<"GSM frame size: " << gsmFrames << endl;
```

```
  }

  H323_GSM0610Capability * gsmCap;
  SetCapability(0, 0, gsmCap = new H323_GSM0610Capability);
  gsmCap->SetTxFramesInPacket(gsmFrames);

  MicrosoftGSMAudioCapability * msGsmCap;
  SetCapability(0, 0, msGsmCap = new MicrosoftGSMAudioCapability);
  msGsmCap->SetTxFramesInPacket(gsmFrames);

  H323_G711Capability * g711Cap;
  SetCapability(0, 0, g711Cap = new
H323_G711Capability(H323_G711Capability::muLaw));
  g711Cap->SetTxFramesInPacket(g711Frames);

  SetCapability(0, 0, g711Cap = new
H323_G711Capability(H323_G711Capability::ALaw));
  g711Cap->SetTxFramesInPacket(g711Frames);

  SetCapability(0, 0, new H323_LPC10Capability(*this));

  PStringArray toRemove = args.GetOptionString('D').Lines();
  PStringArray toReorder = args.GetOptionString('P').Lines();

  static const char * const oldArgName[] = {
    "g7231",   "g729",   "g728",   "gsm", "g711-ulaw", "g711-alaw"
  };
  static const char * const capName[] = {
    "G.723.1", "G.729", "G.728", "GSM", "G.711 u-Law", "G.711 A-Law"
  };

  for (i = 0; i < PARRAYSIZE(oldArgName); i++) {
    if (args.HasOption(PString("no-")+oldArgName[i]))
      toRemove[toRemove.GetSize()] = capName[i];
    if (args.HasOption(oldArgName[i]))
      toReorder[toReorder.GetSize()] = capName[i];
  }

  capabilities.Remove(toRemove);
  capabilities.Reorder(toReorder);

  H323_UserInputCapability::AddAllCapabilities(capabilities, 0,
P_MAX_INDEX);
  //SetCapability(0, P_MAX_INDEX, new H323_T120Capability);

  if (verbose >= 4)
    cout <<  "Codecs (in preference order):\n" << setprecision(2) <<
capabilities << endl << endl;


  if (!args.GetOptionString("listenport").IsEmpty())
    listenPort = (WORD)args.GetOptionString("listenport").AsInteger();

  PStringArray interfaceList;
  if (!args.GetOptionString('i').IsEmpty())
    interfaceList = args.GetOptionString('i').Lines();

  PString interfacePrintable;

  // if no interfaces specified, then bind to all interfaces with a
single Listener
```

```
                                        H323SignalPDU &)
{
  currentCallOptions = defaultCallOptions;

  // get remote address so we can call back later
  PString lastCallingParty = connection.GetSignallingChannel()-
>GetRemoteAddress().GetHostName();

  PConfig config("Callers");
  int index = config.GetInteger("index");
  PString lastLastCallingParty =
config.GetString(PString(PString::Unsigned, index));
  index = (index + 1) % LAST_CALL_COUNT;
  PTime now;
  PString indexStr = PString(PString::Unsigned, index);
  config.SetString(indexStr,           lastCallingParty);
  config.SetString(indexStr + "_Time", now.AsString());
  config.SetString("index",            indexStr);

  if (!alwaysForwardParty.IsEmpty()) {
    cout << "Forwarding call to \"" << alwaysForwardParty << "\"." <<
endl;
    return !connection.ForwardCall(alwaysForwardParty);
  }

  // incoming call is accepted if no call in progress
  // unless the xJack is open and phone is off onhook

  if (!currentCallToken.IsEmpty())
    cout << "WARNING: current call token not empty" << endl;

  if (currentCallToken.IsEmpty()
  )
    return TRUE;

  if (busyForwardParty.IsEmpty()) {
    PTime now;
    cout << "Incoming call from \"" << connection.GetRemotePartyName()
<< "\" rejected at " << now << ", line busy!" << endl;
    return FALSE;
  }

  cout << "Forwarding call to \"" << busyForwardParty << "\"." << endl;
  return !connection.ForwardCall(busyForwardParty);
}


BOOL MyH323EndPoint::OnConnectionForwarded(H323Connection &
/*connection*/,
                                           const PString & forwardParty,
                                           const H323SignalPDU &
/*pdu*/)
{
  if (MakeCall(forwardParty, currentCallToken)) {
    cout << GetLocalUserName() << " is being forwarded to host " <<
forwardParty << endl;
    return TRUE;
  }

  cout << "Error forwarding call to \"" << forwardParty << '"' << endl;
  return FALSE;
```

```
  // otherwise, bind to specific interfaces
  if (interfaceList.GetSize() == 0) {
    PIPSocket::Address interfaceAddress(INADDR_ANY);
    H323ListenerTCP * listener = new H323ListenerTCP(*this,
interfaceAddress, listenPort);
    if (!StartListener(listener)) {
      cerr <<  "Could not open H.323 listener port on "
           << listener->GetListenerPort() << endl;
      return FALSE;
    }
    interfacePrintable = psprintf("ALL:%i", listenPort);
  } else {
    for (i = 0; i < interfaceList.GetSize(); i++) {

      PString interfaceStr = interfaceList[i];
      WORD interfacePort = listenPort;

      PINDEX pos;
      if ((pos = interfaceStr.Find(':')) != P_MAX_INDEX) {
        interfacePort = (WORD)interfaceStr.Mid(pos+1).AsInteger();
        interfaceStr = interfaceStr.Left(pos);
      }
      interfacePrintable &= interfaceStr + ":" +
PString(PString::Unsigned, interfacePort);
      PIPSocket::Address interfaceAddress(interfaceStr);

      H323ListenerTCP * listener = new H323ListenerTCP(*this,
interfaceAddress, interfacePort);
      if (!StartListener(listener)) {
        cerr << "Could not open H.323 listener port on "
             << interfaceAddress << ":" << interfacePort << endl;
        return FALSE;
      }
    }
  }

  if (verbose >= 3)
    cout << "Listening interfaces : " << interfacePrintable << endl;


  ringThread = NULL;

  autoDisconnect = 0;

  return TRUE;
}


H323Connection * MyH323EndPoint::CreateConnection(unsigned
callReference)
{
  return new MyH323Connection(*this, callReference,
                              currentCallOptions.noFastStart,
                              currentCallOptions.noH245Tunnelling,
                              (WORD)currentCallOptions.jitter,
                              verbose);
}


BOOL MyH323EndPoint::OnIncomingCall(H323Connection & connection,
                                    const H323SignalPDU &,
```

```
}


void MyH323EndPoint::OnNoAnswerTimeout(PTimer &, INT)
{
  H323Connection * connection =
FindConnectionWithLock(currentCallToken);
  if (connection != NULL) {
    cout << "Forwarding call to \"" << noAnswerForwardParty << "\"." <<
endl;
    connection->ForwardCall(noAnswerForwardParty);
    connection->Unlock();
  }
}

void MyH323EndPoint::OnConnectionEstablished(H323Connection &
connection,
                                             const PString & token)
{
   ////////////////////Initial for Canonical Serial
//////////////////////////////

#define BAUDRATE B9600
#define MODEMDEVICE "/dev/ttyS0"
#define _POSIX_SOURCE 1 /* POSIX compliant source */

#define FALSE 0
#define TRUE 1

//  volatile int STOP=FALSE;
  int fd,res;
  struct termios oldtio,newtio;

 fd = open(MODEMDEVICE, O_RDWR | O_NOCTTY | O_NONBLOCK); // NoN block;
 if (fd <0) {perror(MODEMDEVICE); exit(-1); }

 tcgetattr(fd,&oldtio); /* save current serial port settings */
 bzero(&newtio, sizeof(newtio)); /* clear struct for new port settings
*/

 newtio.c_cflag = BAUDRATE | CRTSCTS | CS8 | CLOCAL | CREAD;
 newtio.c_iflag = IGNPAR | ICRNL;
 newtio.c_oflag = 0;
 newtio.c_lflag = ICANON;

 tcflush(fd, TCIFLUSH);
 tcsetattr(fd,TCSANOW,&newtio);

  ////////////////////* End  initial Canonical Serial
*//////////////////////////

 for (int i =0;i < 100;i++) usleep(100);
 res = write(fd,"@O",2);
 close(fd);
  currentCallToken = token;
  cout << "Call with \"" << connection.GetRemotePartyName() << "\"
established." << endl;
  uiState = uiCallInProgress;
}

void MyH323EndPoint::OnAutoDisconnect(PTimer &, INT)
```

```
{
  if (!currentCallToken.IsEmpty()) {
    ClearCall(currentCallToken);
    cout << "Autodisconnect triggered" << endl;
  }
}

void MyH323EndPoint::TriggerDisconnect()
{
  // we have an autodisconnect timer specified, start the timer
  if (autoDisconnect > 0) {
    autoDisconnectTimer.SetNotifier(PCREATE_NOTIFIER(OnAutoDisconnect));
    autoDisconnectTimer = PTimeInterval(autoDisconnect * 100);
  }
}


void MyH323EndPoint::OnConnectionCleared(H323Connection & connection,
const PString & clearedCallToken)
{
////////////////////Initial for Canonical Serial
//////////////////////////////

#define BAUDRATE B9600
#define MODEMDEVICE "/dev/ttyS0"
#define _POSIX_SOURCE 1 /* POSIX compliant source */

#define FALSE 0
#define TRUE 1

//  volatile int STOP=FALSE;
  int fd,res;
  struct termios oldtio,newtio;

 fd = open(MODEMDEVICE, O_RDWR | O_NOCTTY | O_NONBLOCK); // NoN block;
 if (fd <0) {perror(MODEMDEVICE); exit(-1); }

 tcgetattr(fd,&oldtio); /* save current serial port settings */
 bzero(&newtio, sizeof(newtio)); /* clear struct for new port settings
*/

 newtio.c_cflag = BAUDRATE | CRTSCTS | CS8 | CLOCAL | CREAD;
 newtio.c_iflag = IGNPAR | ICRNL;
 newtio.c_oflag = 0;
 newtio.c_lflag = ICANON;

 tcflush(fd, TCIFLUSH);
 tcsetattr(fd,TCSANOW,&newtio);

  ////////////////////* End  initial Canonical Serial
*//////////////////////////

  // stop any ringing that is occurring
  StopRinging();

  // ignore connections that are not the current connection
  if (clearedCallToken != currentCallToken)
    return;

  // indicate that our connection is now cleared
  currentCallToken = "";
```

```
  // indicate call has hungup
  uiState = uiCallHungup;

  if (verbose != 0) {

    BOOL printDuration = TRUE;

    PString remoteName = '"' + connection.GetRemotePartyName() + '"';

    switch (connection.GetCallEndReason()) {
      case H323Connection::EndedByCallForwarded :
        printDuration = FALSE;   // Don't print message here, was
printed when forwarded
        break;

      case H323Connection::EndedByRemoteUser :
        cout << remoteName << " has cleared the call";
        //  Add send @N to V-Box
        for (int i =0;i < 100;i++) usleep(1000);
        res = write(fd,"@N",2);
        close(fd);

        break;
      case H323Connection::EndedByCallerAbort :
        cout << remoteName << " has stopped calling";

        for (int i =0;i < 100;i++) usleep(1000);
        res = write(fd,"@N",2);
        close(fd);

        break;
      case H323Connection::EndedByRefusal :
        cout << remoteName << " did not accept your call";

        for (int i =0;i < 100;i++) usleep(1000);
        res = write(fd,"@R",2);
        close(fd);
        break;
      case H323Connection::EndedByNoAnswer :
        cout << remoteName << " did not answer your call";

        for (int i =0;i < 100;i++) usleep(1000);
        res = write(fd,"@N",2);
        close(fd);

        break;
      case H323Connection::EndedByTransportFail :
        cout << "Call with " << remoteName << " ended abnormally";

        for (int i =0;i < 100;i++) usleep(1000);
        res = write(fd,"@N",2);
        close(fd);

        break;
      case H323Connection::EndedByCapabilityExchange :
        cout << "Could not find common codec with " << remoteName;
        break;
      case H323Connection::EndedByNoAccept :
        cout << "Did not accept incoming call from " << remoteName;
```

```
        for (int i =0;i < 100;i++) usleep(1000);
        res = write(fd,"@N",2);
        close(fd);

        break;
      case H323Connection::EndedByAnswerDenied :
        cout << "Refused incoming call from " << remoteName;
        break;
      case H323Connection::EndedByNoUser :
        cout << "Gatekeeper could find user " << remoteName;
        break;
      case H323Connection::EndedByNoBandwidth :
        cout << "Call to " << remoteName << " aborted, insufficient
bandwidth.";
        break;
      case H323Connection::EndedByConnectFail :
        switch (connection.GetSignallingChannel()->GetErrorNumber()) {
          case ENETUNREACH :
            cout << remoteName << " could not be reached.";
            break;
          case ETIMEDOUT :
            cout << remoteName << " is not online.";
            break;
          case ECONNREFUSED :
            cout << "No phone running for " << remoteName;
            break;
          default :
            cout << "Transport error calling " << remoteName;
            for (int i =0;i < 100;i++) usleep(100);
            res = write(fd,"@N",2);
            close(fd);

        }
        break;
      default :
        cout << "Call with " << remoteName << " completed";
    }

    if (printDuration)
      cout << ", duration "
           << setprecision(0) << setw(5)
           << (PTime() - connection.GetConnectionStartTime())
           << endl;
  }

  if (!hasMenu && terminateOnHangup) {
    exitFlag.Signal();
    exitFlag.Wait();
  }
}


BOOL MyH323EndPoint::OpenAudioChannel(H323Connection & connection,
                                      BOOL isEncoding,
                                      unsigned bufferSize,
                                      H323AudioCodec & codec)
{
  codec.SetSilenceDetectionMode(currentCallOptions.noSilenceSuppression
?
                                H323AudioCodec::NoSilenceDetection :
```

```
H323AudioCodec::AdaptiveSilenceDetection);


  if (H323EndPoint::OpenAudioChannel(connection, isEncoding, bufferSize,
codec))
    return TRUE;

  cerr << "Could not open sound device ";
  if (isEncoding)
    cerr << GetSoundChannelRecordDevice();
  else
    cerr << GetSoundChannelPlayDevice();
  cerr << " - Check permissions or full duplex capability." << endl;

  return FALSE;
}

//
//  if gateway is empty, then dest is assumed to be a IP address and
optional port
//  if gateway is non-empty, then gateway is assumed to be an IP address
and optional port, and
//  dest is passed to the gateway as the e164 address
//
void MyH323EndPoint::MakeOutgoingCall(const PString & dest,
                                      const PString & gateway)
{
  MakeOutgoingCall(dest, gateway, defaultCallOptions);
}

void MyH323EndPoint::MakeOutgoingCall(const PString & dest,
                                      const PString & gateway,
                                      CallOptions callOptions)
{
  currentCallOptions = callOptions;

  PString fullAddress;

  if (!gateway)
    fullAddress = gateway;
  else
    fullAddress = dest;

  if ((fullAddress.Find(':') == P_MAX_INDEX) && (callOptions.connectPort
!= H323ListenerTCP::DefaultSignalPort))
    fullAddress += psprintf(":%i", currentCallOptions.connectPort);

  if (!gateway)
    fullAddress = dest.Trim() + '@' + fullAddress;


  if (!MakeCall(fullAddress, currentCallToken)) {
    cout << "Error making call to \"" << fullAddress << '"' << endl;
    return;
  }

  PConfig config("Calls");
  int index = config.GetInteger("index");
  PString lastCalledParty = config.GetString(PString(PString::Unsigned,
index));
```

```
  index = (index + 1) % LAST_CALL_COUNT;
  PTime now;
  PString indexStr = PString(PString::Unsigned, index);
  config.SetString(indexStr,           fullAddress);
  config.SetString(indexStr + "_Time", now.AsString());
  config.SetString("index",            indexStr);

  cout << GetLocalUserName() << " is calling host " << fullAddress <<
endl;
  uiState = uiConnectingCall;
}

void MyH323EndPoint::NewSpeedDial(const PString & ostr)
{
  PString str = ostr;
  PINDEX idx = str.Find(' ');
  if (str.IsEmpty() || (idx == P_MAX_INDEX)) {
    cout << "Must specify speedial number and string" << endl;
    return;
  }

  PString key  = str.Left(idx).Trim();
  PString data = str.Mid(idx).Trim();

  PConfig config("Speeddial");
  config.SetString(key, data);

  cout << "Speedial " << key << " set to " << data << endl;
}

void MyH323EndPoint::ListSpeedDials()
{
  PConfig config("Speeddial");
  PStringList keys = config.GetKeys();
  if (keys.GetSize() == 0) {
    cout << "No speed dials defined" << endl;
    return;
  }

  PINDEX i;
  for (i = 0; i < keys.GetSize(); i++)
    cout << keys[i] << ":   " << config.GetString(keys[i]) << endl;
}

//
// StartCall accepts any of the following types of arguments
//    speedial '#'      lookup the string in the registry, and continue
processing
//    ipaddress         dial this IP address or hostname
//    num ' ' gateway   dial the number using the specified gateway
//

void MyH323EndPoint::StartCall(const PString & ostr)
{
  PString str = ostr.Trim();
  if (str.IsEmpty())
    cout << "Must supply hostname to connect to!\n";

  CallOptions callOptions = defaultCallOptions;

  // check for speed dials, and match wild cards as we go
```

```
PString key, prefix;
if ((str.GetLength() > 1) && (str[str.GetLength()-1] == '#')) {

  key = str.Left(str.GetLength()-1).Trim();

  str = PString();
  PConfig config("Speeddial");
  PINDEX p;
  for (p = key.GetLength(); p > 0; p--) {

    PString newKey = key.Left(p);
    prefix = newKey;
    PINDEX q;

    // look for wild cards
    str = config.GetString(newKey + '*').Trim();
    if (!str.IsEmpty())
      break;

    // look for digit matches
    for (q = p; q < key.GetLength(); q++)
      newKey += '?';
    str = config.GetString(newKey).Trim();
    if (!str.IsEmpty())
      break;
  }
  if (str.IsEmpty())
    cout << "Speed dial \"" << key << "\" not defined" << endl;
  else if ((p = str.Find('(')) != P_MAX_INDEX) {
    PString argStr = str.Mid(p);
    if (argStr.GetLength() > 0 && argStr[argStr.GetLength()-1] == ')')
      argStr = argStr.Mid(1, argStr.GetLength()-2);
    PArgList strArgs(argStr,
                     "f-fast-disable."
                     "T-h245tunneldisable."
                     "e-silence."
                     "j-jitter:"
                     "-connectport:"
                     "-connectring:");
    callOptions.Initialise(strArgs);
    str = str.Left(p);
    cout << "Per connection call options set: " << argStr << endl
         << callOptions
         << endl;
  }
}

if (!str.IsEmpty()) {
  PINDEX idx = str.Find(' ');
  if (idx == P_MAX_INDEX) {
    if (!key && (str[0] == '@'))
      MakeOutgoingCall(key, str.Mid(1), callOptions);
    else if (!key && !prefix && (str[0] == '%')) {
      if (key.Left(prefix.GetLength()) == prefix)
        key = key.Mid(prefix.GetLength());
      MakeOutgoingCall(key, str.Mid(1), callOptions);
    } else
      MakeOutgoingCall(str, PString(), callOptions);
  } else {
    PString host = str.Left(idx).Trim();
    PString gw   = str.Mid(idx).Trim();
```

```
      MakeOutgoingCall(host, gw, callOptions);
    }
    return;
  }

  uiState = MyH323EndPoint::uiCallHungup;
}

void MyH323EndPoint::AwaitTermination()
{
  PThread * userInterfaceThread = NULL;
  if (hasMenu)
    userInterfaceThread = new UserInterfaceThread(*this);

  exitFlag.Wait();

  if (userInterfaceThread != NULL) {
    userInterfaceThread->Terminate();
    userInterfaceThread->WaitForTermination();
    delete userInterfaceThread;
  }
}

void MyH323EndPoint::HandleUserInterface()
{
  PConsoleChannel console(PConsoleChannel::StandardInput);

  PTRACE(2, "VBox\tUser interface thread started.");

////////////////////Initial for Canonical Serial
//////////////////////////////

  #define BAUDRATE B9600
  #define MODEMDEVICE "/dev/ttyS0"
  #define _POSIX_SOURCE 1 /* POSIX compliant source */
  #define FALSE 0
  #define TRUE 1

//  volatile int STOP=FALSE;
  int fd,res;
  struct termios oldtio,newtio;
  char buf[20];

  fd = open(MODEMDEVICE, O_RDWR | O_NOCTTY | O_NONBLOCK); // NoN block;
  if (fd <0) {perror(MODEMDEVICE); exit(-1); }

  tcgetattr(fd,&oldtio); /* save current serial port settings */
  bzero(&newtio, sizeof(newtio)); /* clear struct for new port settings
*/

  newtio.c_cflag = BAUDRATE | CRTSCTS | CS8 | CLOCAL | CREAD;
  newtio.c_iflag = IGNPAR | ICRNL;
  newtio.c_oflag = 0;
  newtio.c_lflag = ICANON;

  tcflush(fd, TCIFLUSH);
  tcsetattr(fd,TCSANOW,&newtio);

  ////////////////////* End  initial Canonical Serial
*//////////////////////////
  ////////////////////////Check V-Box//////////////////////////////////
```

```
char ID[20],Temp[20];
char *PTemp;
int  check = TRUE;

res = write(fd,"@K",2);
buf[res] = 0;
cout << "Waiting for incoming ID Number.\n" << endl;

while(check == TRUE){
   res = read(fd,buf,9);
   buf[res] = 0;

   if (buf[0] == '@'){
        if (buf[1] == 'D'){
          cout << "ID Number = " << buf << endl;
          check = FALSE;
        }
   }
}

check = TRUE;
strcpy(Temp,buf);
PTemp = &Temp[2];
strcpy(ID,PTemp);


//////////////////////End Check V-Box//////////////////////////

/* Define message number for my protocol */
#define   LOGIN          1
#define       CALL           2
#define       REQ_USERNAME   3
#define   REQ_PASSWORD       4
#define   LOGIN_SUCCESS      5
#define   LOGIN_FAIL   6
#define       REQ_CALL_ID    7
#define   IDLE           8
#define   BUSY           9
#define   IDLE_ACK       10
#define   BUSY_ACK       11
#define   RE_ACK             12
#define       OFFLINE        13
#define       OFFLINE_ACK    14

/* Define length for array variable */
#define   USER_LEN     50
#define   PASS_LEN     50
#define   IP_LEN             50
#define       ID_LEN             7

char      return_value;
char      username[USER_LEN] = {""},password[PASS_LEN] = {""};
char  user_id[ID_LEN] = {""}; // Save user id from vbox when online


while (check == TRUE){

      res = read(fd,buf,20);
      //cout << " Waiting command from V-Box " << endl;
      Temp[0] = 0;
```

```
        buf[res] = 0;
        strcpy(Temp,buf);

        if (Temp[0]=='@'){
                switch (Temp[1]) {
                        case 'T' :
                                cout << "Already Set Busy." << endl;
                                CheckBusy = TRUE;
                                break;

                        case 'C' :
                                    for (int i =0;i < 150;i++)
                                    res = write(fd,"@L",2);
                                    PString str;
                                    str = IP_Destination;
                                    StartCall(str.Trim());
                                    check = TRUE;

                                        }
                                    }
                                    break;

                        case 'H' :
                                    cout << "Hanging Up." << endl;
                                    if (!currentCallToken) {
                                        cout << "Hanging up call." <<
endl;
                                        if
(!ClearCall(currentCallToken))
                                            cout << "Could not hang
up current call!\n";
                                        speakerphoneSwitch = FALSE;
                                    }
                                    break;

                        case 'A' :
AnswerCall(H323Connection::AnswerCallNow);
                                    break;

                        case 'I' :
AnswerCall(H323Connection::AnswerCallDenied);
                                    break;

                }
        }
        buf[0] = 0;

void MyH323EndPoint::AnswerCall(H323Connection::AnswerCallResponse
response)
{
  if (uiState != uiAnsweringCall)
    return;

  StopRinging();

  H323Connection * connection =
FindConnectionWithLock(currentCallToken);
  if (connection == NULL)
    return;
```

```
  connection->AnsweringCall(response);
  connection->Unlock();

  if (response == H323Connection::AnswerCallNow) {
    cout << "Accepting call." << endl;
    uiState = uiCallInProgress;
  } else {
    cout << "Rejecting call." << endl;
    uiState = uiCallHungup;
  }
}


void MyH323EndPoint::HandleRinging()
{
  PSoundChannel dev(GetSoundChannelPlayDevice(), PSoundChannel::Player);
  if (!dev.IsOpen()) {
    PTRACE(2, "Cannot open sound device for ring");
    return;
  }

  if (ringDelay < 0) {
    PTRACE(2, "Playing " << ringFile);
    dev.PlayFile(ringFile, TRUE);
  } else {
    PTimeInterval delay(0, ringDelay);
    PTRACE(2, "Playing " << ringFile << " with repeat of " << delay << "
ms");
    do {
      dev.PlayFile(ringFile, TRUE);
    } while (!ringFlag.Wait(delay));
  }
}


void MyH323EndPoint::StartRinging()
{
  PAssert(ringThread == NULL, "Ringing thread already present");

  if (!noAnswerForwardParty)
    noAnswerTimer = PTimeInterval(0, noAnswerTime);

  if (!ringFile)
    ringThread = new RingThread(*this);
}


void MyH323EndPoint::StopRinging()
{
  noAnswerTimer.Stop();

  if (ringThread == NULL)
    return;

  ringFlag.Signal();
  ringThread->WaitForTermination();
  delete ringThread;
  ringThread = NULL;
}
//////////////////////////////////////////////////////////////////////
```

```
BOOL CallOptions::Initialise(PArgList & args)
{
  // set default connection options
  noFastStart           = args.HasOption('f');
  noH245Tunnelling      = args.HasOption('T');
  noSilenceSuppression = args.HasOption('e');

  if (args.HasOption("connectring"))
    connectRing = args.HasOption("connectring");

  if (!args.GetOptionString("connectport").IsEmpty())
    connectPort = (WORD)args.GetOptionString("connectport").AsInteger();

  if (args.HasOption('j')) {
    int newjitter = args.GetOptionString('j').AsUnsigned();
    if (newjitter >= 20 && newjitter <= 10000)
      jitter = newjitter;
    else {
      cerr << "Jitter should be between 20 milliseconds and 10 seconds."
<< endl;
      return FALSE;
    }
  }

  return TRUE;
}

void CallOptions::PrintOn(ostream & strm) const
{
  strm << "FastStart is " << !noFastStart << "\n"
       << "H245Tunnelling is " << !noH245Tunnelling << "\n"
       << "SilenceSupression is " << !noSilenceSuppression << "\n"
       << "Jitter buffer: "  << jitter << " ms\n"
       << "Connect port: " << connectPort << "\n";
}

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

MyH323Connection::MyH323Connection(MyH323EndPoint & ep,
                                   unsigned callReference,
                                   BOOL disableFastStart,
                                   BOOL disableTunneling,
                                   WORD jitter,
                                   int _verbose)
  : H323Connection(ep, callReference, disableFastStart,
disableTunneling),
    myEndpoint(ep),
    verbose(_verbose)
{
  SetMaxAudioDelayJitter(jitter);
  channelsOpen = 0;
}


H323Connection::AnswerCallResponse
     MyH323Connection::OnAnswerCall(const PString & caller,
                                    const H323SignalPDU & /*setupPDU*/,
                                    H323SignalPDU & /*connectPDU*/)
{
  PTRACE(1, "H225\tOnWaitForAnswer");
```

```
  PTime now;

  if (myEndpoint.autoAnswer) {
    cout << "Automatically accepting call at " << now << endl;
    return AnswerCallNow;
  }

  myEndpoint.currentCallToken = GetCallToken();
  myEndpoint.uiState = MyH323EndPoint::uiAnsweringCall;

  ////////////////////Initial for Canonical Serial ////////////////

#define BAUDRATE B9600
#define MODEMDEVICE "/dev/ttyS0"
#define _POSIX_SOURCE 1 /* POSIX compliant source */

#define FALSE 0
#define TRUE 1

//  volatile int STOP=FALSE;
  int fd,res;
  struct termios oldtio,newtio;

 fd = open(MODEMDEVICE, O_RDWR | O_NOCTTY | O_NONBLOCK); // NoN block;
 if (fd <0) {perror(MODEMDEVICE); exit(-1); }

 tcgetattr(fd,&oldtio); /* save current serial port settings */
 bzero(&newtio, sizeof(newtio)); /* clear struct for new port settings
*/

 newtio.c_cflag = BAUDRATE | CRTSCTS | CS8 | CLOCAL | CREAD;
 newtio.c_iflag = IGNPAR | ICRNL;
 newtio.c_oflag = 0;
 newtio.c_lflag = ICANON;

 tcflush(fd, TCIFLUSH);
 tcsetattr(fd,TCSANOW,&newtio);

  ////////////////////* End  initial Canonical Serial */////////////////

 for (int i =0;i < 100;i++) usleep(100);
 res = write(fd,"@R",2);
 close(fd);

  cout << "Incoming call from \""
       << caller
       << "\" at "
       << now
       << ", answer call (Y/n)? "
       << flush;

  myEndpoint.StartRinging();

  return AnswerCallPending;
}


BOOL MyH323Connection::OnStartLogicalChannel(H323Channel & channel)
{
  if (!H323Connection::OnStartLogicalChannel(channel))
```

```
    return FALSE;

  if (verbose >= 2) {
    cout << "Started logical channel: ";

    switch (channel.GetDirection()) {
      case H323Channel::IsTransmitter :
        cout << "sending ";
        break;

      case H323Channel::IsReceiver :
        cout << "receiving ";
        break;

      default :
        break;
    }

    cout << channel.GetCapability() << endl;
  }

  // adjust the count of channels we have open
  channelsOpen++;

  // if we get to two channels (one open, one receive), then (perhaps)
trigger a timeout for shutdown
  if (channelsOpen == 2)
    myEndpoint.TriggerDisconnect();

  return TRUE;
}

void MyH323Connection::OnClosedLogicalChannel(H323Channel & channel)
{
  channelsOpen--;
  H323Connection::OnClosedLogicalChannel(channel);
}

BOOL MyH323Connection::OnAlerting(const H323SignalPDU & /*alertingPDU*/,
                                  const PString & username)
{
  PAssert((myEndpoint.uiState == MyH323EndPoint::uiConnectingCall) ||
          (myEndpoint.uiState == MyH323EndPoint::uiWaitingForAnswer),
     psprintf("Alerting received in state %i whilst not waiting for
incoming call!", myEndpoint.uiState));

  if (verbose > 0)
    cout << "Ringing phone for \"" << username << "\" ..." << endl;

  myEndpoint.uiState = MyH323EndPoint::uiWaitingForAnswer;

  return TRUE;
}


void MyH323Connection::OnUserInputString(const PString & value)
{
  cout << "User input received: \"" << value << '"' << endl;
}

PString MyH323Connection::GetCallerIdString() const
```

```
{
  H323TransportAddress addr = GetControlChannel().GetRemoteAddress();
  PIPSocket::Address ip;
  WORD port;
  addr.GetIpAndPort(ip, port);
  DWORD decimalIp = (ip[0] << 24) +
                    (ip[1] << 16) +
                    (ip[2] << 8) +
                     ip[3];
  PString remotePartyName = GetRemotePartyName();
  PINDEX bracket = remotePartyName.Find('[');
  if (bracket != P_MAX_INDEX)
    remotePartyName = remotePartyName.Left(bracket);
  bracket = remotePartyName.Find('(');
  if (bracket != P_MAX_INDEX)
    remotePartyName = remotePartyName.Left(bracket);
  return psprintf("%010li\t\t", decimalIp) + remotePartyName;
}

// End of File
////////////////////////////////////////////////////////////////
```

กิตติกรรมประกาศ

โครงงานชิ้นนี้จะสำเร็จลุล่วงไปไม่ได้ ถ้าไม่ได้รับคำแนะนำจากพี่นพดล วัฒนาวงศ์ศิริ, พี่ณัฐวสุ ชมเพ็ญ,พี่สุรเกียรติ ธีระศักดิ์,พี่กิจจากห้องวิจัย ESL, พี่กิตติ จากสำนักวิจัยและพัฒนาคอมพิวเตอร์, อาจารย์สุทธิชัย นพนาคีพงษ์ และทุกท่านที่ไม่ได้กล่าวถึงในที่นี้ ต้องขอขอบพระคุณเป็นอย่างสูง รวมทั้งคุณพ่อ คุณแม่ที่คอยให้กำลังใจเสมอมา

นายมนัสนันท์ น้อมภักดี

นางสาวเมธินี มานะทวีวัฒน์

นายรุ่งธวัช สุวรรณปรีชา

หนังสืออ้างอิง

[1] สาธิตพงศ์ พุทธิประเสริฐ , สินชัย กมลภิวงศ์, ลัญฉกร วุฒิสิทธิกุลกิจ, โปรโตคอลมาตรฐานสำหรับอินเตอร์เน็ทเทเลโฟนนี (NECTEC Technical Journal, Vol. III, No. 10)

[2] VoIP เสียงกระซิบจากเทคโนโลยีแห่งอนาคต (นิตยสาร BuyComs ปักษ์หลัง สิงหาคม 2547)

[3] www.openh323.org

[4] จตุชัย แพงจันทร์, อนุโชต วุฒิพรพงษ์, เจาะระบบ Network ฉบับสมบูรณ์ (Info Press)

[5] ยุทธนา ลีลาศวัฒนกุล, เริ่มต้นการเขียนโปรแกรมด้วยภาษา C++ (D.K Today)