

การควบคุมเครื่องใช้ไฟฟ้าด้วยเสียงพูดผ่านตู้สายโทรศัพท์

TELEPHONE CONTROL ELECTRICAL APPLIANCE BY VOICE



โดย

นายชำนาญ วงศ์ตระกูลชัย

นางสาววรรณรีย์ วงศ์ไตรรัตน์

นายวิเชษฐ์ อ่อนรำพรรณ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2543

เลขหน้า.....
เลขทะเบียน..... 42149
วัน, เดือน, ปี 14 พ.ค. 2545

b.....
i.....

b11204977

การควบคุมเครื่องใช้ไฟฟ้าด้วยเสียงพูดผ่านตู้สายโทรศัพท์

TELEPHONE CONTROL ELECTRICAL APPLIANCE BY VOICE

โดย

นายชำนาญ วงศ์ตระกูลชัย 41013010

นางสาววรรณรีย์ วงศ์ไตรรัตน์ 41013028

นายวิเชษฐ์ อ่อนรำพรรณ 41013030

อาจารย์ที่ปรึกษา

รศ.ดร. สุวิพล ลิทธิชีวภาค

ผศ. เกรียงไกร วงศ์โรจนกรณ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2543

การควบคุมเครื่องใช้ไฟฟ้าด้วยเสียงพูดผ่านตู้สายโทรศัพท์

Telephone Control Electrical Appliance by Voice

โดย	นายชำนาญ วงศ์ตระกูลชัย	41013010
	นางสาววรรณรีย์ วงศ์ไครรัตน์	41013028
	นายวิเชษฐ์ อ่อนราพรรณ	41013030

อาจารย์ที่ปรึกษา รศ.ดร. สุวิพล สิริชีวะภาค

ผศ. เกรียงไกร วงศ์โรจนภรณ์

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้เป็นการศึกษาเกี่ยวกับการนำเสียงของมนุษย์มาทำการควบคุมการเปิด-ปิด อุปกรณ์เครื่องใช้ไฟฟ้าผ่านตู้สายโทรศัพท์ ซึ่งจะเริ่มจากโมเด็มทำหน้าที่รับสัญญาณเสียงจากโทรศัพท์ แล้วนำสัญญาณเสียงที่ได้เข้าสู่การ์ดเสียง(sound card)เพื่อนำมาทำการวิเคราะห์ โดยการเขียนโปรแกรมควบคุมการ์ดเสียงของเครื่องคอมพิวเตอร์เพื่อให้สามารถรับคำสั่งทางเสียง การนำสัญญาณที่ได้มาทำการวิเคราะห์จะต้องทำการสร้างโมเดลเสียง (ใช้ Dynamic Time Warping : DTW) จากไฟล์ต้นแบบให้สามารถบันทึกเสียงที่ได้แบ่งออกเป็นคำ ๆ และจะนำเสียงที่ได้ไปเปรียบเทียบกับ โมเดลเสียงจากไฟล์ต้นแบบเพื่อทำการวิเคราะห์ว่าตรงกับโมเดลใด หลังจากนั้นจะทำการส่งสัญญาณข้อมูลที่ได้เข้าสู่วงจรที่ควบคุมเครื่องใช้ไฟฟ้า

ABSTRACT

This project concerns about the study of taking human's voice to control on-off the electronics through the telephone line. Beginning with modem that receives sound from the telephone , then sending our voice via sound card for analysis. This project will create the application program to control the sound card of the computer for ability to receive the command by voice. This is the result of sound analysis when makes sound model (using Dynamic Time Warping : DTW) from the original file. We can record the received sound with the model sound to find out the matching model. After that information sound is sent to the circuit that controls electrical appliance.

ปริญญานิพนธ์ปีการศึกษา 2543

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การควบคุมเครื่องใช้ไฟฟ้าด้วยเสียงพูดผ่านคู่สายโทรศัพท์

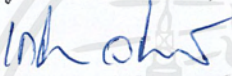
Telephone Control Electrical Appliance by Voice

ผู้จัดทำ

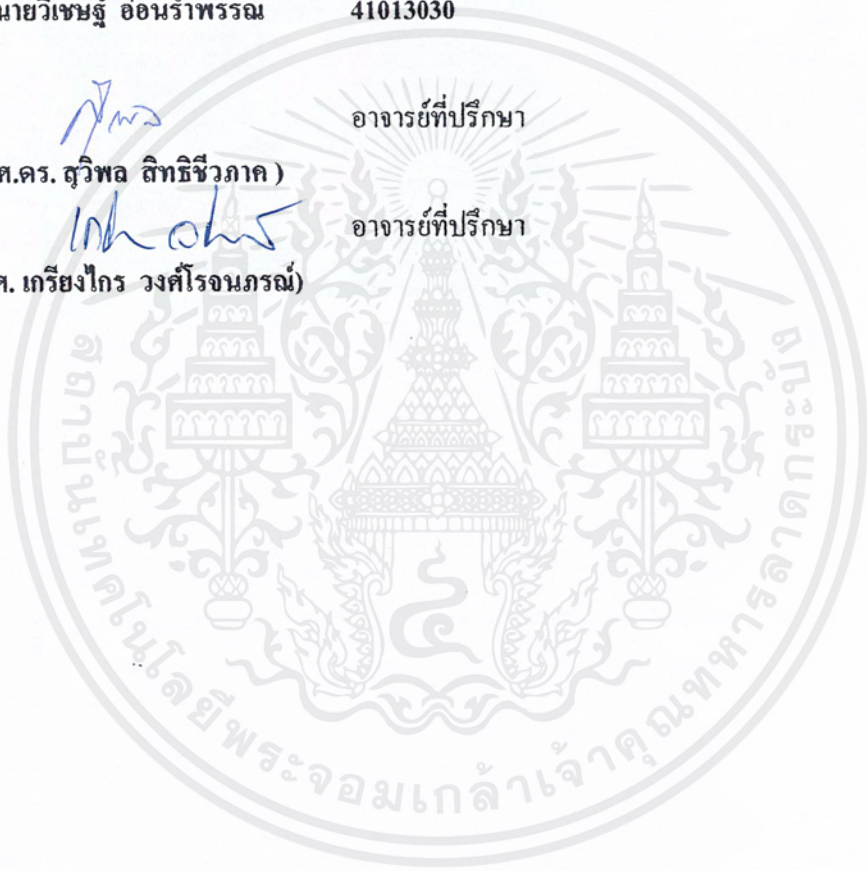
1. นายชำนาญ วงศ์ตระกูลชัย 41013010
2. นางสาววรรณรีย์ วงศ์ไตรรัตน์ 41013028
3. นายวิเชษฐ์ อ่อนรำพรรณ 41013030


(รศ.ดร. สุวิพล ลิทธิชีวภาค)

อาจารย์ที่ปรึกษา


(ผศ. เกียรติเกรียงไกร วงศ์โรจนภรณ์)

อาจารย์ที่ปรึกษา



บทที่ 1 บทนำ	1
1.1 วัตถุประสงค์ของปริญยานิพนธ์	1
1.2 ขอบเขตของปริญยานิพนธ์	2
1.3 ขั้นตอนในการศึกษาและดำเนินงาน	2
บทที่ 2 ทฤษฎีและหลักการ	4
2.1 ความรู้เบื้องต้นเกี่ยวกับโทรศัพท์	4
2.1.1 ส่วนประกอบของโทรศัพท์	5
2.1.2 สัญญาณ (Signalling)	5
2.2 ความรู้เบื้องต้นเกี่ยวกับ โมเด็ม	7
2.2.1 ชุดคำสั่ง AT	7
2.2.2 การใช้งานชุดคำสั่ง AT	7
2.2.3 ประเภทของชุดคำสั่ง AT	7
2.2.4 รูปแบบของชุดคำสั่ง AT	8
2.2.5 การเชื่อมต่อกับ RS-232-C	8
2.2.6 ชุดคำสั่ง AT ที่สำคัญ	8
2.3 หลักการวิเคราะห์เสียง	14
2.3.1 วิเคราะห์เสียง (Front-end Processing)	14
2.3.1.1 การคำนวณหาค่าพิชช์(pitch)	15
2.3.1.2 การประมาณเชิงเส้น(Linear Predictive Code : LPC)	19
2.3.1.3 การควอนไทซ์(Quantization)	29
2.3.2 ปรับรูปแบบเสียง (Pattern Matching)	37
2.3.3 กฎการตัดสินใจ (Dicision rule)	41
2.4 หลักการตัดเสียงให้เป็นคำเดี่ยว	42
2.4.1 การประมวลเสียงพูดโดยขึ้นกับเวลา	42
2.4.2 ค่าของกำลังสองของสัญญาณเสียงในช่วงเวลาสั้น ๆ	43
2.4.3 การแยกเสียงพูดออกจากเสียงเงียบ(Silence)	45
2.5 โครงสร้างและสถาปัตยกรรมของไมโครคอนโทรลเลอร์ MCS-51	46
2.5.1 คุณสมบัติทางเทคนิคของไมโครคอนโทรลเลอร์ตระกูล MCS-51	
อนุกรม AT89xx	46
2.5.2 การจัดขาของไมโครคอนโทรลเลอร์ MCS-51	48
2.5.3 โครงสร้างและการทำงานของพอร์ต	51
2.5.4 การใช้งานเป็นพอร์ตอินพุต	54

สารบัญ(ต่อ)

	หน้า
2.5.5 การใช้งานเป็นพอร์ทเอ้าท์พุท	54
2.5.6 การอ่านค่าลอจิกจากพอร์ท	54
2.5.7 จังหวะการทำงานของไมโครคอนโทรลเลอร์ MCS-51	55
2.6 มาตรฐานพอร์ทอนุกรมแบบ RS-232	56
บทที่ 3 การคำนวณและการสร้าง	58
3.1 ส่วนของการตอบรับอัตโนมัติ	59
3.1.1 การตอบรับอัตโนมัติโดยใช้โมเด็ม	59
3.2 ส่วนของโปรแกรมติดต่อกับผู้ใช้งาน	60
3.2.1 การตั้งค่าเริ่มต้นของโปรแกรมและการทำงานสั่งเครื่องใช้ไฟฟ้า ในโหมดแมนนวล	60
3.2.2 โหมดแสดงรูปและการอ่านข้อมูลไฟล์ .wav	62
3.2.3 โหมดการหาค่าพลังงานเสียง	62
3.2.4 โหมดการตัดคำ	63
3.2.5 โหมดการหาค่าสัมประสิทธิ์การประมาณเชิงเส้น	63
3.2.6 โหมดการหาค่าสัมประสิทธิ์เซปสตรีม	64
3.2.7 โหมดการตัดสินใจ (Pattern Matching)	64
3.3 ส่วนเชื่อมต่อกับคอมพิวเตอร์และควบคุมการเปิดปิดเครื่องใช้ไฟฟ้า	65
3.3.1 ส่วนของการต่อพอร์ทและไมโครคอนโทรลเลอร์	65
3.3.2 ส่วนของวงจรขั้วรีเลย์	66
บทที่ 4 การทดลองและผลการทดลอง	70
4.1 การทดลองในส่วนของการตอบรับ	70
4.2 ผลการทดลองในส่วนของการเชื่อมต่อบอร์ดควบคุมกับคอมพิวเตอร์	70
4.3 ผลการทดลองในส่วนของโปรแกรม	71
4.3.1 กราฟแสดงผลการคำนวณที่ได้จากโปรแกรม	71
4.3.2 ตารางแสดงผลการทดลอง	93
บทที่ 5 บทวิจารณ์และสรุป	95
5.1 สรุปผลการทดลอง	95
5.1.1 ส่วนของภาคฮาร์ดแวร์	95
5.1.2 ส่วนของภาคซอฟต์แวร์	95
5.2 ข้อเสนอแนะ และปัญหาที่พบ	96
5.3 แนวทางการพัฒนา	96

ภาคผนวก

กิตติกรรมประกาศ

หนังสืออ้างอิง



สารบัญรูปภาพ

รูปที่	หน้า
2.1 บล็อกไดอะแกรมของเครื่องรับโทรศัพท์	4
2.2 แสดง Basic isolate – word recognition system	14
2.3 ตัวอย่างคาบเวลาพิชช์ของเสียง	15
2.4 เฟรมเริ่มต้น(T1) และเฟรมสุดท้าย (T2) ของพลังงานเสียงพูดที่ใช้ในการคำนวณ	16
2.5 ออร์โตคอร์เรชันของเสียงพูด	18
2.6 ตัวอย่างเสียงพูดของคำว่า “จิม”	18
2.7 แสดงสเปกตรัมของสัญญาณเสียงเทียบกับสเปกตรัมที่ได้จากการประมาณเชิงเส้นที่มีอันดับ (LPC order) เท่ากับ p ต่าง ๆ (ก)p=1 (ข)p=8 (ค)p=12 (ง)p=16 (จ)p=20	20
2.8 แสดงขั้นตอนการเตรียมสัญญาณในการวิเคราะห์	21
2.9 แสดงขนาดสเปกตรัมของฟังก์ชันถ่ายโอนของการพรีเอมฟาสีส	22
2.10 แสดงการแบ่งช่วงสัญญาณในการวิเคราะห์	22
2.11 แสดงวินโดว์สี่เหลี่ยม (ก) ในโดเมนเวลา (ข) ในโดเมนความถี่	23
2.12 แสดงวินโดว์แฮมมิง (ก) ในโดเมนเวลา (ข) ในโดเมนความถี่	24
2.13 บล็อกไดอะแกรมแสดงแบบจำลองการสร้างสัญญาณเสียงพูดอย่างง่าย	24
2.14 ตัวอย่างข้อมูลที่จะทำการควอนไทซ์	30
2.15 แสดงการจับกลุ่มและดัชนีของตารางรหัส (code book) ที่ได้	30
2.16 แสดงบล็อกไดอะแกรมของการควอนไทซ์แบบเวกเตอร์	31
2.17 แสดงเวกเตอร์ตัวควอนไทซ์ (vector quantizer) ที่ใช้ในระบบการจดจำเสียงพูด	35
2.18 แผ่นแบบของคำพูดที่ประกอบด้วย N(N tenth order) ของชุดเวกเตอร์	38
2.19 ค่าเวลามาตรฐานเชิงเส้นของแผ่นแบบสัญญาณอ้างอิงกับแผ่นแบบของสัญญาณอินพุต	38
2.20 ค่าเวลามาตรฐานที่ไม่เป็นเชิงเส้นของสัญญาณอ้างอิงกับสัญญาณอินพุต และสามารถอ้างอิง กับเฟรมทั้งหมดที่ขยายออกไปได้ในช่วง 2-5 เฟรม	39
2.21 ตัวอย่างของวิธีการไดนามิกที่ค่า ij ต่าง ๆ ที่เกี่ยวเนื่องกับระยะทาง d และค่าความแตกต่าง D	40
2.22 แสดงขั้นตอนการตัดสินใจ	41
2.23 แสดงบล็อกไดอะแกรม (a) ขนาดกำลังสองในช่วงเวลาสั้น ๆ (b) ขนาดในช่วงเวลาสั้น ๆ	44
2.24 โครงสร้างพื้นฐานของไมโครคอนโทรลเลอร์ MCS-51 ในอนุกรม AT89Cxx	47
2.25 โครงสร้างพื้นฐานของอนุกรม AT89Sxx	47
2.26 แสดงการจัดขามาตรฐานของไมโครคอนโทรลเลอร์ 89S53xx	49
2.27 แสดงรายละเอียดโครงสร้างหลักของไมโครคอนโทรลเลอร์แบบแฟลช	49
2.28 แสดงวงจรภายในของพอร์ททุกพอร์ท	52
2.29 แสดงวงจรพูลอัพภายในพอร์ทของไมโครคอนโทรลเลอร์แบบแฟลช	55
2.30 แสดงไขเคล็ดการทำงานของไมโครคอนโทรลเลอร์แบบแฟลช	55

สารบัญรูปภาพ(ต่อ)

รูปที่	หน้า
2.31 รูปร่างและตำแหน่งต่าง ๆ ของคอนเนกเตอร์สำหรับพอร์ท RS-232	57
3.1 แสดงบล็อกไดอะแกรมการทำงานของเครื่องใช้ไฟฟ้าผ่านทางโทรศัพท์โดยการใช้เสียง	58
3.2 แสดงรูปโปรแกรม TCEAV(Telephone Electricla Appliance by Voice)	60
3.3 แสดงการกำหนดค่าเริ่มต้นให้แก่โปรแกรม TCEAV	61
3.4 แสดงการทำงานโหมดแมนนวลของโปรแกรม TCEAV	61
3.5 แสดงการทำงานของโหมด Edit Wave	62
3.6 แสดงการทำงานในโหมด Energy	62
3.7 แสดงการทำงานในโหมด End-of-Speech	63
3.8 แสดงการทำงานในโหมด LPC	64
3.9 แสดงการทำงานในโหมด CEP	64
3.10 แสดงการทำงานในโหมดการตัดสินใจ	65
3.11 Flochart แสดงลำดับการทำงานโดยรวม	67
3.12 วงจรส่วนควบคุมเครื่องใช้ไฟฟ้า	68
3.13 เครื่องควบคุมเครื่องใช้ไฟฟ้าด้วยเสียงพูดผ่านคู่สายโทรศัพท์	69
4.1 แสดงสัญญาณกระดิ่งที่อินพุทของโมเด็ม	70
4.2 รูปแบบสัญญาณเสียงของเสียงพูดคำว่า “หนึ่ง”	72
4.3 รูปแบบพลังงานเสียงของเสียงพูดคำว่า “หนึ่ง”	72
4.4 รูปแบบสัมประสิทธิ์การประมาณเชิงเส้นของเสียงพูดคำว่า “หนึ่ง”	73
4.5 รูปแบบสัมประสิทธิ์เซปสตรัมของเสียงพูดคำว่า “หนึ่ง”	73
4.6 รูปแบบสัญญาณเสียงของเสียงพูดคำว่า “สอง”	74
4.7 รูปแบบพลังงานเสียงของเสียงพูดคำว่า “สอง”	74
4.8 รูปแบบสัมประสิทธิ์การประมาณเชิงเส้นของเสียงพูดคำว่า “สอง”	75
4.9 รูปแบบสัมประสิทธิ์เซปสตรัมของเสียงพูดคำว่า “สอง”	75
4.10 รูปแบบสัญญาณเสียงของเสียงพูดคำว่า “สาม”	76
4.11 รูปแบบพลังงานเสียงของเสียงพูดคำว่า “สาม”	76
4.12 รูปแบบสัมประสิทธิ์การประมาณเชิงเส้นของเสียงพูดคำว่า “สาม”	77
4.13 รูปแบบสัมประสิทธิ์เซปสตรัมของเสียงพูดคำว่า “สาม”	77
4.14 รูปแบบสัญญาณเสียงของเสียงพูดคำว่า “สี่”	78
4.15 รูปแบบพลังงานเสียงของเสียงพูดคำว่า “สี่”	78
4.16 รูปแบบสัมประสิทธิ์การประมาณเชิงเส้นของเสียงพูดคำว่า “สี่”	79
4.17 รูปแบบสัมประสิทธิ์เซปสตรัมของเสียงพูดคำว่า “สี่”	79
4.18 รูปแบบสัญญาณเสียงของเสียงพูดคำว่า “ห้า”	80

สารบัญรูปลักษณ์(ต่อ)

รูปที่	หน้า
4.19 รูปแบบพลังงานเสียงของเสียงพูดคำว่า “ห้า”	80
4.20 รูปแบบสัมประสิทธิ์การประมาณเชิงเส้นของเสียงพูดคำว่า “ห้า”	81
4.21 รูปแบบสัมประสิทธิ์เซปสตรัมของเสียงพูดคำว่า “ห้า”	81
4.22 รูปแบบสัญญาณเสียงของเสียงพูดคำว่า “หก”	82
4.23 รูปแบบพลังงานเสียงของเสียงพูดคำว่า “หก”	82
4.24 รูปแบบสัมประสิทธิ์การประมาณเชิงเส้นของเสียงพูดคำว่า “หก”	83
4.25 รูปแบบสัมประสิทธิ์เซปสตรัมของเสียงพูดคำว่า “หก”	83
4.26 รูปแบบสัญญาณเสียงของเสียงพูดคำว่า “เจ็ด”	84
4.27 รูปแบบพลังงานเสียงของเสียงพูดคำว่า “เจ็ด”	84
4.28 รูปแบบสัมประสิทธิ์การประมาณเชิงเส้นของเสียงพูดคำว่า “เจ็ด”	85
4.29 รูปแบบสัมประสิทธิ์เซปสตรัมของเสียงพูดคำว่า “เจ็ด”	85
4.30 รูปแบบสัญญาณเสียงของเสียงพูดคำว่า “แปด”	86
4.31 รูปแบบพลังงานเสียงของเสียงพูดคำว่า “แปด”	86
4.32 รูปแบบสัมประสิทธิ์การประมาณเชิงเส้นของเสียงพูดคำว่า “แปด”	87
4.33 รูปแบบสัมประสิทธิ์เซปสตรัมของเสียงพูดคำว่า “แปด”	87
4.34 รูปแบบสัญญาณเสียงของเสียงพูดคำว่า “ปิด”	88
4.35 รูปแบบพลังงานเสียงของเสียงพูดคำว่า “ปิด”	88
4.36 รูปแบบสัมประสิทธิ์การประมาณเชิงเส้นของเสียงพูดคำว่า “ปิด”	89
4.37 รูปแบบสัมประสิทธิ์เซปสตรัมของเสียงพูดคำว่า “ปิด”	89
4.38 รูปแบบสัญญาณเสียงของเสียงพูดคำว่า “เปิด”	90
4.39 รูปแบบพลังงานเสียงของเสียงพูดคำว่า “เปิด”	90
4.40 รูปแบบสัมประสิทธิ์การประมาณเชิงเส้นของเสียงพูดคำว่า “เปิด”	91
4.41 รูปแบบสัมประสิทธิ์เซปสตรัมของเสียงพูดคำว่า “เปิด”	91
4.42 แบบของสัญญาณเสียงพูดคำว่า “หนึ่ง” ในแต่ละครั้ง	92

สารบัญตาราง

	หน้า
ตารางที่ 2.1 พารามิเตอร์ที่ใช้ในการหมุนหมายเลขโทรศัพท์	9
ตารางที่ 2.2 แสดงรายละเอียดของไมโครคอนโทรลเลอร์ตระกูล MCS-51 ที่ Atmel ผลิตขึ้น	48
ตารางที่ 2.3 แสดงหน้าที่พิเศษของพอร์ต 1 ในไมโครคอนโทรลเลอร์แบบแฟลช	51
ตารางที่ 2.4 ชื่อและชนิดของสายสัญญาณของคอนเน็กเตอร์ DB-9 สำหรับพอร์ต RS-232	57
ตารางที่ 4.1 แสดงผลการทดลองส่วนบอร์ทควบคุม	71
ตารางที่ 4.2 แสดงผลการทดลองจากการใช้แบบเสียงอ้างอิง 3 เสียงเพื่อการเปรียบเทียบ	93
ตารางที่ 4.3 แสดงผลการทดลองจากการใช้แบบเสียงอ้างอิง 4 เสียงเพื่อการเปรียบเทียบ	93
ตารางที่ 4.4 แสดงผลการทดลองจากการใช้แบบเสียงอ้างอิง 5 เสียงเพื่อการเปรียบเทียบ	94



บทที่ 1

บทนำ

เนื่องจากความเจริญก้าวหน้าทางเทคโนโลยีในปัจจุบัน ทำให้การติดต่อสื่อสารเป็นไปได้อย่างรวดเร็วและประหยัด ทำให้มีอุปกรณ์อำนวยความสะดวกต่าง ๆ ถูกสร้างขึ้นมามากมายและมีราคาถูก ดังนั้นเราจึงสามารถนำเทคโนโลยีใหม่ ๆ เข้ามาช่วยอำนวยความสะดวกในชีวิตประจำวันของเราได้อย่างมากเช่นกัน

ในอดีตมีการวิเคราะห์ว่าการสอนให้คอมพิวเตอร์สามารถรับรู้ และจำเสียงพูดของมนุษย์ได้นั้นไม่สามารถที่จะทำได้ เนื่องจากการพูดของมนุษย์มีความซับซ้อนและมีความแตกต่างกันในแต่ละบุคคล อีกทั้งความเร็วในการพูด รวมถึงเสียงสูงต่ำของแต่ละคนนั้นก็มีความแตกต่างกันไปด้วย แต่ก็มีงานวิจัยเพื่อให้คอมพิวเตอร์สามารถรับรู้และจำเสียงพูดเรื่อยมา จนในปัจจุบันการรับรู้และจำเสียงพูดสามารถจะใช้งานได้ดี และมีการนำไปใช้กับอุปกรณ์ต่าง ๆ เช่น เอทีเอ็ม(Automatic Teller Machine : ATM) และเครื่องตอบรับโทรศัพท์อัตโนมัติ ถึงแม้คอมพิวเตอร์จะไม่สามารถรับรู้และจำเสียงได้เท่ากับมนุษย์ แต่เราสามารถสอนให้คอมพิวเตอร์รับรู้และจำเสียงได้ในขีดจำกัดระดับหนึ่ง โดยมีการกำหนดขอบเขตของการรับรู้และจำ เช่น จำกัดคำที่สามารถรับรู้ได้

ปริญญาพนธ์นี้ได้มีการศึกษาและทดลองนำเอาส่วนของการรับรู้และจำเสียงที่เป็นทฤษฎีมาใช้งานจริง โดยทำในรูปการใช้เสียงสั่งงานเครื่องใช้ไฟฟ้าผ่านทางตู้สายโทรศัพท์ ซึ่งจะได้อธิบายต่อไป

1.1 วัตถุประสงค์ของปริญญาพนธ์ มีดังนี้

- 1.) ศึกษาเกี่ยวกับรายละเอียดและสัญญาณต่าง ๆ ของตู้สายโทรศัพท์ โมเด็ม และทำการนำเอาสัญญาณจากตู้สายโทรศัพท์ผ่านเข้าเครื่องพีซี(PC) โดยการต่อผ่านทางโมเด็ม
- 2.) แยกคำพูดของมนุษย์ออกเป็นคำเดี่ยว ๆ ได้
- 3.) ศึกษาการวิเคราะห์เสียงพูดแบบขึ้นกับผู้พูด
- 4.) นำงานวิจัยเกี่ยวกับการรับรู้และจำเสียงภาษาไทยที่ได้มีผู้พัฒนาไว้ในระดับหนึ่งแล้วมาทำการศึกษา และพัฒนาต่อ โดยจะเน้นพิจารณาและพัฒนาในด้านการนำไปใช้งานจริง
- 5.) เพื่อพัฒนาโปรแกรมส่วนต่าง ๆ ที่ทำให้คอมพิวเตอร์สามารถรับรู้และจำเสียงพูด และนำโปรแกรมมาใช้งานได้
- 6.) ทำการปรับปรุงวิธีการทางทฤษฎีให้สามารถนำมาใช้งานในทางปฏิบัติได้ เช่น หาวิธีที่จะช่วยเพิ่มความเร็วในการวิเคราะห์เสียง หรือปรับปรุงขั้นตอนบางอย่างเพื่อลดความผิดพลาดในการวิเคราะห์ เป็นต้น
- 7.) เพื่อเป็นพื้นฐานในการประยุกต์ใช้งานเพิ่มเติมต่อไป
- 8.) ศึกษาการทำงานของ MCS-51
- 9.) ศึกษาและออกแบบวงจรขั้วรีเลย์
- 10.) ศึกษาการทำงานของโมเด็ม การเชื่อมต่อกับ RS-232 และการ์ดเสียง(sound card)

1.2 ของเขตของปริณิธาน

เพื่อให้สามารถใช้เสียงสั่งงานควบคุมอุปกรณ์ไฟฟ้าโดยผ่านทางตู้สายโทรศัพท์ ซึ่งมีขั้นตอนพอสังเขปดังนี้

- 1.) เปิดเครื่องที่ซีไว้แล้วเปิดโปรแกรมสั่งงานควบคุมอุปกรณ์ไฟฟ้าอัตโนมัติ
- 2.) เมื่ออยู่นอกบ้านก็สามารถที่จะโทรศัพท์เข้ามาที่บ้านโดยที่จะมีโมเด็มคอยทำการตอบรับอัตโนมัติ
- 3.) รับเสียงพูดผ่านทางไมโครโฟน(Microphone)ของเครื่องโทรศัพท์ โดยที่นำสัญญาณเสียงต่อเข้าโมเด็ม แล้วนำมาประมวลผลและส่งผลที่ได้ไปสั่งงานอุปกรณ์ไฟฟ้านั้น
- 4.) เสียงที่ใช้สั่งงานอุปกรณ์ไฟฟ้านั้นจะขึ้นกับผู้พูด เสียงของคำที่มีความหมายเดียวกันจากผู้พูดต่างบุคคลก็จะได้ผลจากการรับรู้ไม่เหมือนกัน โดยขั้นตอนจะมีการเก็บตัวอย่างเสียงของคำนั้น ๆ ที่มากและหลากหลายพอไว้ก่อน เพื่อผลการรับรู้ที่ถูกต้อง
- 5.) เป็นการวิเคราะห์เสียงพูดแบบต่อเนื่อง ช่วงวิเคราะห์สัญญาณโดยที่ความยาวของคำสั่งนั้น เราได้ทำการออกแบบไว้ คือคำสั่งในการประมวลผลจะรวดเร็วขึ้น ตัวอย่างคำสั่งที่ได้ออกแบบไว้ได้แก่ “หนึ่ง” ก็หมายความว่าสั่งให้อุปกรณ์ตัวที่ 1, “เปิด” หมายความว่า ให้ทำการเปิดอุปกรณ์ไฟฟ้า โดยที่ในโครงการนี้ได้มีการออกแบบสวิตช์(switch)ได้ 8 ตัว คือจะมีสวิตช์เบอร์ 1-8

ในการศึกษาเรื่องการสั่งงานคอมพิวเตอร์ด้วยเสียง ส่วนของขั้นตอนต่าง ๆ ที่ใช้ในการวิเคราะห์เสียงและการควบคุมอุปกรณ์ต่าง ๆ นั้นจะต้องมีความรู้พื้นฐานพอสมควรในเรื่องต่าง ๆ ดังต่อไปนี้

- ลักษณะต่าง ๆ ของเสียงพูดของมนุษย์ เช่น ความถี่ , ความดัง , ความแตกต่างของเสียงชายและหญิง , เสียงที่เปล่งออกมาทางปาก หรือออกมาทางจมูก , เสียงรบกวนที่เกิดขึ้น และกราฟต่าง ๆ ของเสียง เป็นต้น
- พื้นความรู้ทางด้านคณิตศาสตร์ที่สำคัญ เช่น ฟังก์ชันการถ่ายโอน , การประมาณเชิงเส้น , ผลรวมกำลังสองของความคลาดเคลื่อน , เวกเตอร์หลายมิติ และเมตริกซ์ เป็นต้น
- พื้นความรู้ทางสถิติ เช่น ความน่าจะเป็น ความน่าจะเป็นแบบมีเงื่อนไข
- ลักษณะการทำงานของโมเด็มว่าทำการติดต่อกับสัญญาณที่ส่งผ่านมาทางสายโทรศัพท์อย่างไร และสัญญาณเสียงที่ได้มีการจัดเก็บและนำมาใช้งานได้อย่างไร
- หลักการเขียนโปรแกรมภาษา C หรือ Visual C

1.3 ขั้นตอนในการศึกษาและดำเนินงาน มีดังนี้

- 1.) ศึกษารูปแบบการวิเคราะห์เสียงพูดแบบขึ้นกับผู้พูด และ ไม่ต่อเนื่อง
- 2.) ศึกษาขั้นตอนการประมาณเชิงเส้นของสัญญาณเสียงพูด โดยวิธีออโตคอร์รีเลชัน (Autocorrelation analysis) เพื่อนำมาใช้ในการวิเคราะห์เสียงพูดของมนุษย์ สาเหตุที่ใช้การประมาณเชิงเส้นเพราะสามารถนำมาประยุกต์ใช้กับการสื่อสารทางเสียงได้หลายด้าน และเป็นวิธีการบีบอัดข้อมูลได้เป็นอย่างดีโดยมีการสูญเสียข้อมูลน้อยมาก

- 3.) วิเคราะห์การหาค่าพารามิเตอร์ LPC และเซปตรัม (Ceptum) ของเสียง
- 4.) สร้างแบบจำลองโดยวิธีไดนามิกไทม์แวกปีง (Dynamic Time Warping : DTW) เพื่อใช้ในการรับรู้และจำเสียงพูด
- 5.) เขียนโปรแกรมตามรูปแบบการวิเคราะห์เสียงที่ได้ศึกษามา

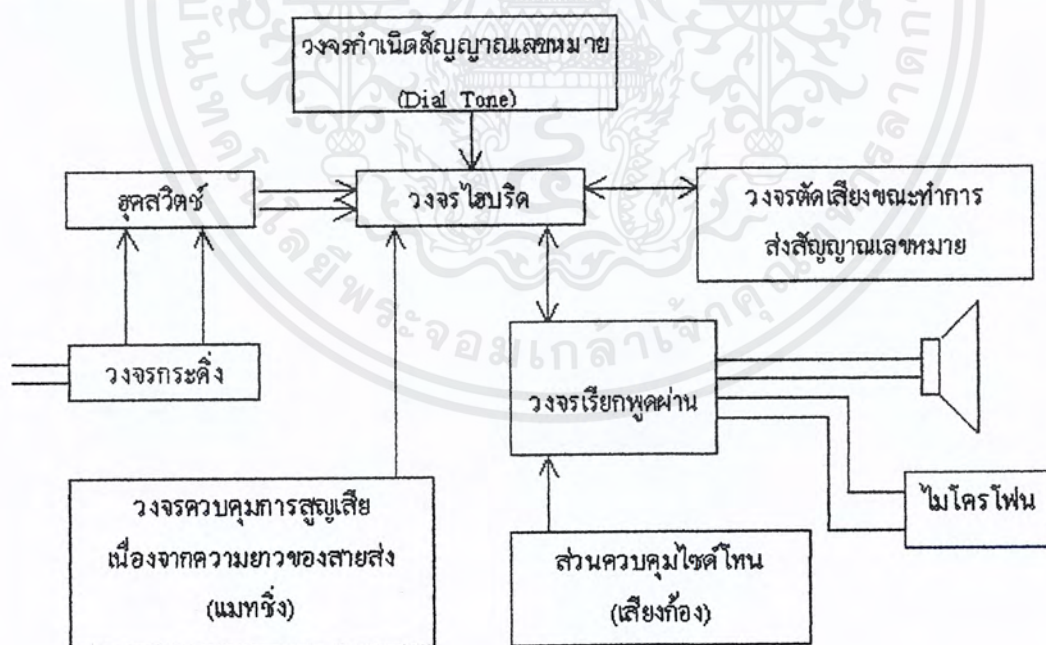


บทที่ 2

ทฤษฎีและหลักการ

2.1 ความรู้เบื้องต้นเกี่ยวกับโทรศัพท์

เครื่องโทรศัพท์ เป็นอุปกรณ์ปลายทางซึ่งทำหน้าที่รับและส่งสัญญาณเสียงพูดระหว่างผู้เช่า โดยการทำงานคือ ทางด้านส่งจะแปลงพลังงานเสียงให้เป็นพลังงานไฟฟ้าแล้วส่งไปตามคู่สายโทรศัพท์ ทางด้านรับก็จะได้รับสัญญาณไฟฟ้า แล้วก็จะทำการเปลี่ยนพลังงานไฟฟ้านั้นให้เป็นพลังงานเสียง โดยจะทำการรับส่งอย่างนี้ไปเรื่อย ๆ โดยที่ในช่วงปกติ(สายว่าง)จะมีแหล่งจ่ายไฟร่วม โดยทางชุมสายโทรศัพท์จะจ่ายไฟกระแสตรง(DC)ขนาด 48 โวลต์ ให้แต่ละคู่ของผู้ใช้ โดยคู่สายโทรศัพท์นั้นจะมีตัวนำ 2 เส้น ที่มีชื่อว่า ทิป(Tip) และ ริง(Ring) โดยที่ทิปจะต่อลงกราวด์ และริงจะต่อกับไฟ -48 โวลต์ เมื่อมีการเรียกหมายเลขนี้เข้ามา ทางชุมสายจะจ่ายสัญญาณกระดิ่งไปยังผู้ถูกเรียก คือจากสภาวะปกติจะมีไฟตกคร่อมคู่สาย 48 โวลต์ จะมีสัญญาณกระดิ่งซึ่งเป็นแรงดันกระแสสลับประมาณ 100 โวลต์พีค เป็นเวลา 2 วินาที และหยุด 4 วินาที เป็นจังหวะให้กระดิ่งโทรศัพท์ทำงาน เมื่อผู้ใช้ปลายทาง(ผู้ถูกเรียก) ยกหูโทรศัพท์ขึ้น สวิตช์ภายในโทรศัพท์จะทำการต่อคู่สายเข้ากับวงจรภายในที่มีความต้านทานกระแสตรงต่ำ จะเกิดการครบวงจรและทำให้แรงดันไฟ 48 โวลต์ตกลงเหลือ 5 ถึง 10 โวลต์ ทั้งนี้ขึ้นอยู่กับวงจรภายในของเครื่องโทรศัพท์นั้น ๆ การที่แรงดันไฟฟ้าตกลงนั้นจะเป็นการทำให้ชุมสายได้รู้ว่า ผู้ใช้ปลายทางได้ยกหูแล้ว ดังนั้นจึงทำการสวิตช์เชื่อมต่อคู่สายทั้งสองเข้าด้วยกัน ทำให้สามารถติดต่อกันได้



รูปที่ 2.1 บล็อกไดอะแกรมของเครื่องรับโทรศัพท์

2.1.1 ส่วนประกอบของโทรศัพท์

ในเครื่องโทรศัพท์ส่วนใหญ่จะมีส่วนประกอบที่สำคัญดังต่อไปนี้

- 1.) วงจรกำเนิดสัญญาณเลขหมาย จะทำหน้าที่ในการสร้างเลขหมายโทรศัพท์ ซึ่งอาจจะเป็นสัญญาณพัลส์ หรือสัญญาณความถี่คู่ผสม (DTMF) แล้วแต่โทรศัพท์ที่ใช้งาน
- 2.) วงจรตัดเสียง จะทำการตัดเสียงต่าง ๆ มิให้เข้าไปในชุมสายเพื่อป้องกันการผิดพลาด ถ้าบังเอิญมีสัญญาณเรียกที่มีความถี่ใกล้เคียงกับสัญญาณความถี่ของเลขหมายจะทำให้การส่งสัญญาณหมายเลขเกิดความผิดพลาดได้
- 3.) ไคอัลโทน(Dial Tone)ทำการกำเนิดสัญญาณหมายเลขโทรศัพท์ เพื่อส่งไปยังชุมสายโทรศัพท์
- 4.) วงจรส่งเสียงพูด(Transmitter) เป็นส่วนที่รับสัญญาณจากผู้พูดผ่านทางไปยังผู้ใช้ปลายทาง โดยจะใช้ไมโครโฟนแปลงสัญญาณเสียงให้เป็นสัญญาณไฟฟ้า
- 5.) วงจรรับเสียงพูด(Receiver) เป็นตัวแปลงสัญญาณไฟฟ้าจากคั่นทางให้กลายเป็นสัญญาณเสียง โดยจะใช้ลำโพงหรือไดอะเฟรม
- 6.) วงจรกระดิ่ง(Ringer) ทำหน้าที่ตรวจจับสัญญาณจากชุมสายโทรศัพท์ เมื่อมีสัญญาณกระดิ่งเข้ามาก็จะทำการแปลงสัญญาณจากชุมสายให้กลายเป็นเสียงกระดิ่ง
- 7.) สุกสวิทช์(Hook Switct) ทำหน้าที่ตัดต่อเครื่องโทรศัพท์เข้ากับคู่สาย มี 2 สถานะ คือ สถานะการยกหู (Off Hook) และสถานะการวางหู (On Hook)
- 8.) วงจรไฮบริดจ์(Hybrid) เนื่องจากวงจรในเครื่องโทรศัพท์จะใช้ระบบ 4 สาย ก็คือใช้สาย 2 สายสำหรับส่งสัญญาณเสียงพูด และอีก 2 สายสำหรับนำเสียงพูดจากอีกด้านหนึ่งมายังหูผู้ฟัง แต่ในการเชื่อมต่อระหว่างเครื่องโทรศัพท์กับชุมสายโทรศัพท์จะเป็นระบบ 2 สาย จะติดต่อกันโดยใช้สายสัญญาณเพียง 2 เส้นเท่านั้นเอง จึงต้องใช้วงจรไฮบริดจ์ทำการเชื่อมต่อระบบ 2 สาย และ 4 สายให้สามารถติดต่อกันได้

2.1.2 สัญญาณ(Signalling)

สัญญาณ คือ ข่าวดำเนินการที่ติดต่อกันระหว่างเครื่องโทรศัพท์กับชุมสาย หรือข่าวดำเนินการที่ติดต่อกันระหว่างชุมสายกับชุมสาย

หน้าที่ทั่ว ๆ ไปของสัญญาณที่ใช้กับโทรศัพท์ในปัจจุบันมีอยู่ 4 หน้าที่ คือ

- 1.) การเตรียมพร้อม(Alerting)
- 2.) การส่งที่อยู่ของข่าวสาร(Transmitting Address Information)
- 3.) การตรวจตรา(Supervising)
- 4.) การส่งสัญญาณข่าวสาร(Transmitting Information Signalling)

สัญญาณระหว่างผู้เข้ากับชุมสาย (Subscriber Signalling)

- 1.) สัญญาณที่ส่งจากผู้เข้ากับชุมสาย

- ยกหู(Off Hook) คือ สถานะผู้เข้ายกหูโทรศัพท์จะมีสภาพลูปปิด (Closed Loop-Low Impedance)

- วางหู(On Hook) คือ สภาพผู้เข้าวางหูหรือสภาพว่าง สายจะมีสภาพลูปเปิด (Open Loop-High Impedance)
- สัญญาณเลขหมาย(Dialing) คือ สภาพที่ผู้เข้าหมุนหมายเลขเข้าเครื่องเป็นแบบหมุน(Rotary Dial) สัญญาณจะเป็นพัลส์(Pulsing) ค่าความต้านทาน(Impedance) จะสูง-ต่ำสลับกันไปตามที่หมุนเลขหมาย ถ้าเครื่องเป็นแบบกดปุ่ม(Touch - Tone) สัญญาณออกจะเป็นความถี่ DTMF ส่งออกไปชุมสาย

2.) สัญญาณที่ส่งมาจากชุมสาย

- สัญญาณพร้อมหมุน หรือพร้อมกดหมายเลข(Dialing Tone) คือ สัญญาณที่บอกถึงสภาพการว่างของอุปกรณ์ชุมสาย และชุมสายพร้อมจะรับรหัส(Code)ที่ทำการหมุนเข้ามา สัญญาณนี้จะเป็นสัญญาณต่อเนื่องความถี่ 425 เฮิรตซ์ มอดูเลตด้วยความถี่ 50 เฮิรตซ์ ผู้เข้าจะได้ยินเมื่อทำการยกหูโทรศัพท์
- สัญญาณไม่ว่าง(Busy Tone) คือ สัญญาณที่บอกให้ทราบว่าอุปกรณ์ชุมสายไม่ว่าง แต่ถ้ายกหูแล้วได้สัญญาณนี้แสดงว่า อุปกรณ์ในชุมสายไม่ว่างและถ้าได้ยินเสียงนี้หลังจากหมุนเลขหมายไปแล้วแสดงว่าผู้เข้าฝ่ายถูกเรียกไม่ว่าง ในกรณีเรียกต่างชุมสาย ลักษณะสัญญาณที่ส่งจะเป็นสัญญาณที่ขาดตอนเป็นช่วง ๆ ส่ง 0.3 วินาที หยุด 0.5 วินาที ความถี่ของสัญญาณ 425 เฮิรตซ์ รูปคลื่นไซน์(Sine wave)
- สัญญาณเรียกกลับ(Ringing Back Tone) เป็นสัญญาณที่ผู้เรียกได้ยินหลังจากหมุนเลขหมายครบแล้ว เพื่อบอกให้ทราบว่า การต่อกระทำสำเร็จ ในขณะนี้ชุมสายจะส่งสัญญาณเรียก(Ringing Signal) ไปยังผู้ถูกเรียก ความถี่ของสัญญาณ 425 เฮิรตซ์ รูปคลื่นไซน์(Sine wave) โดยจะส่ง 1 วินาที หยุด 4 วินาที
- สัญญาณกริ่ง(Ringing Signal) เป็นสัญญาณต่อเนื่องความถี่ของสัญญาณ 25 Hz ค่าแรงดัน 70-90 Vrms โดยส่งไปยังผู้เข้าฝ่ายถูกเรียก ส่ง 1 วินาที หยุด 4 วินาที
- สัญญาณโทนอื่น ๆ เช่น No Tone (Number Unobtainable Tone) บอกให้ทราบว่าเลขหมายที่หมุนมาไม่มีการใช้งานอยู่ เป็นต้น

สัญญาณระหว่างชุมสายกับชุมสาย (Inter Exchange Signalling) มีอยู่ 5 ประเภท คือ

- 1.) สัญญาณจับวงจร (Seizure) เป็นสัญญาณให้ชุมสายปลายทางทราบว่าคู่สายขณะนี้ถูกใช้งานอยู่ ชุมสายปลายทางจะทำการจัดเตรียมอุปกรณ์ที่รับเลขหมายของผู้ถูกเรียกที่จะส่งมา
- 2.) สัญญาณที่อยู่ข่าวสาร(Address information) เป็นสัญญาณที่บอกเลขหมาย หรือประเภทของผู้เข้า
- 3.) สัญญาณตอบรับ(Answer Signal) สัญญาณนี้ถูกส่งเมื่อผู้เข้าฝ่ายถูกเรียกยกหูรับ หน้าหลักของสัญญาณก็คือ
 1. เริ่มต้นคิดเงิน
 2. ส่งสัญญาณคิดเงิน
 3. ตัดวงจรการจับเวลาการใช้อุปกรณ์

4.) สัญญาขยกเล็กการต่อตรง (Clear-Forward) จะถูกส่งเมื่อฝ่ายผู้เรียกวางหู ผลของสัญญานี้จะทำให้วงจรทางด้านปลายทางขยกเล็กการต่อวงจรต่าง ๆ

5.) สัญญาขยกเล็กการต่อกลับ (Clear-Back) จะถูกส่งเมื่อผู้เข้าฝ่ายถูกเรียกวางหู ผลของสัญญานี้จะทำให้ชุมสายต้นทางเริ่มดำเนินการจับเวลา เมื่อเวลาผ่านไป 90-120 วินาที ชุมสายต้นทางจะขยกเล็กการติดต่อมาพร้อมกับส่งสัญญาขยกเล็กการต่อตรง ออกไปเพื่อให้ชุมสายปลายทางขยกเล็กเช่นกัน

2.2 ความรู้เบื้องต้นเกี่ยวกับโมเด็ม

เราสามารถเชื่อมต่อพีซี เพื่อให้บริการอื่น ๆ ได้ด้วยโมเด็ม ซึ่งการเชื่อมต่อระหว่างโครงข่ายโทรศัพท์กับเครื่องพีซีของปรินูณานิพนธ์นี้จะใช้โมเด็มในการเชื่อมต่อด้วย

โมเด็มทุกเครื่องจะใช้วิธีการในการเคลื่อนย้ายข้อมูลผ่านออกไปตามสายโทรศัพท์เหมือนกัน โดยจะเปลี่ยนข้อมูลดิจิทัลของเครื่องพีซีไปตามสัญญาณเสียงก่อนที่จะส่งออกไปตามสายโทรศัพท์ จากนั้นเสียงก็จะเดินทางผ่านเครือข่ายระบบโทรศัพท์ไปยังปลายทาง

2.2.1 ชุดคำสั่ง AT

การใช้งานชุดคำสั่ง AT สำหรับโมเด็มทั่ว ๆ ไปที่ใช้มาตรฐานชุดคำสั่งนี้ โดยจะรวมทั้ง การตอบสนองคำสั่งของโมเด็ม (Result Code) การกำหนดค่ารีจิสเตอร์ s ต่าง ๆ (S-register) และการทำงานของรีจิสเตอร์เหล่านี้ ซึ่งโมเด็มส่วนใหญ่จะถูกควบคุมการทำงานโดยชุดคำสั่งที่ส่งออกมาทางพอร์ตสื่อสารอนุกรม

2.2.2 การใช้งานชุดคำสั่ง AT

ขั้นตอนการเชื่อมต่อระหว่างโมเด็มต้นทาง และปลายทางนั้นจำเป็นจะต้องมีพื้นฐานที่เหมือนกันบางประการ เช่น ความเร็ว รูปแบบของข้อมูล และโปรโตคอลในการรับ-ส่งข้อมูล โดยการเชื่อมต่อโมเด็มในแต่ละครั้งอย่างน้อยที่สุด ผู้ใช้โมเด็มต้นทางจำเป็นต้องทราบว่า จะให้โมเด็มโทรศัพท์ออกด้วยหมายเลขใด ควรจะตั้งเวลาในการรอเสียงตอบจากโมเด็มปลายทางนานเท่าไร ระบบโทรศัพท์นั้นใช้ระบบโทน(Tone) หรือพัลส์(Pulse) และในขณะที่เดียวกันผู้ใช้โมเด็มปลายทางก็จะต้องกำหนดให้โมเด็มรับทราบด้วยว่าจะให้โมเด็มรับสายเข้าหลังจากที่มีเสียงกริ่ง(Ringing)กี่ครั้ง และจะต้องรอสัญญาณได้คอบนานเท่าไรเป็นต้น ซึ่งการดำเนินงานทั้งหมดนี้จะสามารถควบคุมได้โดยใช้ชุดคำสั่ง AT

2.2.3 ประเภทของชุดคำสั่ง AT

โดยพื้นฐานแล้ว เราสามารถจะแบ่งชุดคำสั่ง AT ออกเป็น 2 ประเภทใหญ่ ได้ดังนี้

- ประเภทแรกคือ ชุดคำสั่งที่ใช้ในการปฏิบัติงานอย่างเช่น ATD (คำสั่งให้หมุนหมายเลขโทรศัพท์) หรือ ATH (คำสั่งวางสายโทรศัพท์) เป็นต้น
- ประเภทที่ 2 คือชุดคำสั่งที่ใช้กำหนดค่า หรือเปลี่ยนค่าต่าง ๆ อย่างเช่น ATS7=90 เป็นการกำหนดค่าให้รีจิสเตอร์ S7 มีค่า 90 ซึ่งก็คือให้โมเด็มรอการตอบรับจากโมเด็มปลายทางเป็น

เวลา 90 วินาที หลังจากนั้นหากยังไม่มีสัญญาณใด ๆ คอบกลับมา โมเด็มก็จะวางสายทันที ตามปกติแล้วโปรแกรมสื่อสารมักจะตั้งค่าของรีจิสเตอร์ S ต่าง ๆ ที่จำเป็นเอาไว้ตั้งแต่ตอนที่ผู้ใช้เรียกโปรแกรมนั้น ๆ ขึ้นมา ซึ่งผู้ใช้จะสามารถควบคุมและเปลี่ยนแปลงค่าของรีจิสเตอร์ S เหล่านี้ได้ภายหลัง

2.2.4 รูปแบบของชุดคำสั่ง AT

การใช้ชุดคำสั่ง AT จำเป็นต้องขึ้นด้วยอักษร AT เสมอ และจะต้องจบลงด้วยการกดปุ่ม Enter (หรือ Carriage return) ยกเว้นเฉพาะคำสั่ง A/ (ไม่ต้องกดปุ่ม Enter) ซึ่งหมายถึงให้โมเด็มกลับไปทำคำสั่งล่าสุดซ้ำอีกครั้งหนึ่ง การที่คำสั่งต่าง ๆ ต้องขึ้นต้นด้วยอักษร AT ก็เพราะต้องการให้โมเด็มรับรู้ถึงความเร็วและรูปแบบของอักษรคำสั่งที่ถูกส่งออกมาจากพอร์ตสื่อสารอนุกรมของพีซี

2.2.5 การเชื่อมต่อกับ RS-232-C

RS-232-C จะประกอบไปด้วยวงจรรับ(Receive Data : RD) และวงจรส่ง (Transmit Data : TD) แยกอิสระไม่ขึ้นต่อกัน ซึ่งแต่ละวงจรก็จะมีลักษณะการเคลื่อนย้ายข้อมูลแบบทางเดียว คือยอมให้ข้อมูลเข้าหรือออกเท่านั้น ดังนั้นจึงทำให้การเชื่อมต่อระหว่างพีซี และ โมเด็มมีลักษณะแบบ 2 ทางคือ ทั้งรับข้อมูลเข้า และส่งข้อมูลออกไปพร้อม ๆ กันได้ นอกจากนี้โมเด็มก็ยังมียังวงจรต่าง ๆ ที่ใช้สำหรับเชื่อมต่อกับพีซี และเป็นวงจรที่สำคัญและมีผลต่อการทำงานของพีซี หรือมีผลต่อโปรแกรมสื่อสาร รวมทั้งจะได้กล่าวถึงการควบคุมวงจรดังกล่าวด้วย

2.2.6 ชุดคำสั่ง AT ที่สำคัญ

คำสั่ง AT ที่สำคัญและมักจะถูกใช้งานอยู่เสมอ ซึ่งเป็นคำสั่งทั่ว ๆ ไปที่สามารถใช้งานได้กับโมเด็ม

— ATA (รับโทรศัพท์)

คำสั่งนี้จะทำให้โมเด็มสามารถรับสายโทรเข้าได้ แต่มักจะใช้ในกรณีชั่วคราวที่จำเป็นเท่านั้น โดยปกติแล้วผู้ใช้จะสามารถกำหนดให้โมเด็มรับสายทุกครั้งที่มีเสียงกริ่ง(Ringing) ได้โดยตั้งค่ารีจิสเตอร์ S0 ให้มีค่าเป็น 1 ส่วนในกรณีที่ไม่ต้องการให้โมเด็มรับสายโทรเข้าก็ให้ตั้งค่ารีจิสเตอร์ S0 เป็น 0 แต่ในบางครั้งผู้ใช้อาจต้องการให้โมเด็มสามารถรับสายได้ทันทีได้ ทั้ง ๆ ที่ตั้งค่ารีจิสเตอร์ S0 ไว้เป็น 0 โดยการใช้คำสั่ง ATA ซึ่งการทำงานของโมเด็มหลังจากที่ได้รับคำสั่งนี้ก็คือ โมเด็มจะยกหูโทรศัพท์ขึ้น(Off-Hook) และจะส่งสัญญาณพาหะ (Carrier) ไปยังโมเด็มปลายทาง หลังจากนั้นโมเด็มก็จะรอเป็นเวลา X วินาทีตามที่ได้อ้างไว้ในรีจิสเตอร์ S7 ถ้าไม่มีสัญญาณพาหะตอบกลับมายุในระยะเวลาดังกล่าว โมเด็มก็จะวางหูโทรศัพท์และจะกลับมาอยู่ในสภาวะคำสั่ง (Off-line) อีกครั้ง

- ATD (รับโทรศัพท์)

คำสั่ง ATD จะทำให้โมเด็มยกหูโทรศัพท์ขึ้น(Off-Hook) และหมุนหมายเลขโทรศัพท์ออกด้วย หมายเลขที่คุณกำหนดเอาไว้ในพารามิเตอร์ของคำสั่งนี้ โดยมีรูปแบบดังตัวอย่าง ATD3197707 ซึ่งคำสั่ง ATD นี้จะต้องการพารามิเตอร์บางอย่างทุกครั้งที่ใช้คำสั่งนี้เสมอ อย่างเช่น “T”(Tone), “P”(Pulse), “W”(Wait), หรือ “,” ดังแสดงไว้ในตาราง 2.1

ตารางที่ 2.1 พารามิเตอร์ที่ใช้ในการหมุนหมายเลข โทรศัพท์

พารามิเตอร์	ความหมาย
!	วางหูโทรศัพท์ชั่วคราว
,	หยุดรอเป็นเวลา 2 วินาทีขณะหมุนหมายเลข
;	หลังจากหมุนหมายเลขแล้วจะส่งไปที่สภาวะคำสั่ง
P	หมุนหมายเลขด้วยระบบพัลส์
R	เรียกโหมครีเวิร์ส
S=x	เก็บหมายเลขโทรศัพท์ไว้ที่พื้นที่ x
T	หมุนหมายเลขด้วยระบบโทน
W	รอฟังเสียงสัญญาณ”สายว่าง”

พารามิเตอร์ “P” หมายถึงให้โมเด็มหมุนหมายเลขโทรศัพท์ด้วยระบบพัลส์ ซึ่งเป็นระบบที่ใช้กันในระบบโทรศัพท์แบบหมุน ส่วนพารามิเตอร์ “T” หมายถึงให้โมเด็มหมุนหมายเลขโทรศัพท์ออกด้วยระบบโทน ซึ่งเป็นระบบที่ใช้กันในโทรศัพท์แบบกดปุ่มทั่วไป (บางครั้งอาจจะเรียกโทน ว่า DTMF : Dual Tone Multi-frequency) ดังตัวอย่าง ATDP3197707 (PULSE) ATDT3197707 (TONE) โดยปกติแล้ว ถ้าใช้คำสั่งเพียง ATD และตามด้วยหมายเลข ก็หมายความว่าโมเด็มจะทำการหมุนหมายเลขโทรศัพท์ด้วยวิธีล่าสุดที่เคยหมุนหมายเลขโทรศัพท์มา ยกตัวอย่างเช่น ถ้าโมเด็มได้รับคำสั่ง ATDT3197707 มาก่อน หลังจากนั้นโมเด็มได้รับคำสั่ง ATD3197707 นั่นก็หมายความว่าโมเด็มจะทำการหมุนหมายเลขด้วยระบบโทน ในทางตรงกันข้ามถ้าโมเด็มได้รับคำสั่ง ATDP3197707 มาก่อน จากนั้นก็ได้รับคำสั่ง ATD3197707 โมเด็มก็จะหมุนหมายเลขด้วยระบบพัลส์ แต่อย่างไรก็ตามก่อนที่จะสั่งให้โมเด็มหมุนหมายเลขโทรศัพท์ออก ก็ควรจะทราบเสียก่อนว่าสายโทรศัพท์ที่ต่ออยู่กับโมเด็มนั้นใช้ระบบพัลส์ หรือ โทน

พารามิเตอร์ “W” และ “,” มักจะใช้ในกรณีที่โมเด็มต่ออยู่กับชุมสายโทรศัพท์ภายใน (Private Branch Exchange : PBX) โดยที่ “W” จะมีความหมายว่าให้โมเด็มรอฟังเสียงไดอัล โทน (เสียงที่บอกว่าสายว่าง) ครั้งที่สองก่อนที่จะหมุนหมายเลขโทรศัพท์ออกดังตัวอย่าง ATDT 9W3197707 หมายถึงให้โมเด็มหมุนหมายเลขโทรศัพท์ออกด้วยระบบโทน โดยเริ่มหมุนหมายเลข 9 (เพื่อจะตัดสายออก) จากนั้นก็ให้รอฟังเสียงไดอัล โทนอีกครั้งหนึ่งและจึงหมุนหมายเลข 3197707ซึ่งเสียงไดอัล โทนที่โมเด็มได้ยินครั้งแรกก่อนที่จะหมุนหมายเลข 9

นั่นจะเป็นเสียงที่ส่งออกมาจากชุมสายโทรศัพท์ภายใน และเสียงไดอัลโทนครั้งที่สองหลังจากที่หมุนหมายเลข 9 นั่นจะเป็นเสียงส่งออกมาจากชุมสายโทรศัพท์ขององค์การโทรศัพท์แห่งประเทศไทย ถ้าหากโมเด็มไม่ได้ยินเสียงครั้งที่ 2 ในเวลา X วินาทีตามที่ได้กำหนดไว้ในรีจิสเตอร์ S6 โมเด็มก็จะส่งผลเป็นโค้ด(Result Code) ให้แก่พีซี ว่า “NO DIALTONE” ส่วนพารามิเตอร์ “,” (เครื่องหมายคอมม่า) หมายถึงให้โมเด็มหยุดรอเป็นเวลา X วินาทีก่อนที่จะหมุนหมายเลขโทรศัพท์ต่อไป ซึ่งเวลาค้างกล่าวจะถูกกำหนดด้วยค่าในรีจิสเตอร์ S8 โดยจะมีค่าดีฟอลต์เป็น 2 วินาที ในบางครั้งผู้ใช้อาจสามารถใช้ “,” แทน “W” ได้ดังตัวอย่าง ATDT9,3197707 หมายความว่าให้โมเด็มหมุนหมายเลข 9 และหยุดรอ 2 วินาที จากนั้นก็หมุนหมายเลข 3197707

พารามิเตอร์ “R”(Reverse Mode) หมายถึงให้โมเด็มเปลี่ยนไปเป็นโหมดตอบรับ (Answer mode) หลังจากที่ได้รับหมายเลขโทรศัพท์เรียบร้อยแล้ว ซึ่งโดยปกติแล้วโมเด็มตัวต้นทางที่เป็นผู้หมุนหมายเลขโทรศัพท์ออก จะเป็นโหมดกำหนด(Originate mode)และปลายทางจะเป็นโหมดตอบรับ (สำหรับโหมดดังกล่าวทั้งคู่ หมายถึงย่านความถี่ที่โมเด็มใช้สื่อสารกัน) พารามิเตอร์ “R” นี้มักจะใช้อยู่หลังหมายเลขโทรศัพท์ดังตัวอย่าง ATAT3197707r

พารามิเตอร์ “!” หมายถึงให้โมเด็มวางหูโทรศัพท์ชั่วคราวเป็นเวลา 500-600 ไมโครวินาที และจากนั้นก็ยกหูโทรศัพท์ขึ้นมาอีกครั้ง ลักษณะการวางหูและยกหูโทรศัพท์เช่นนี้เรียกว่าแฟลช(Flash) ใช้กันมากในระบบชุมสายโทรศัพท์ภายใน ซึ่งเครื่องโทรศัพท์รุ่นเก่าก็จะต้องใช้มือกดและปล่อยในขณะที่เครื่องโทรศัพท์รุ่นใหม่จะมีปุ่มแฟลช สำหรับชุมสายโทรศัพท์ภายในบางชนิดมักจะใช้ลักษณะการแฟลชเพื่อควบคุมและสั่งงานพิเศษเกี่ยวกับการโทรศัพท์เข้าและออก

พารามิเตอร์ S=x หมายถึงให้โมเด็มหมุนหมายเลขโทรศัพท์ที่เก็บเอาไว้ใน NVRAM ตำแหน่ง x ดังตัวอย่าง ATDS=2 ก็คือให้โมเด็มหมุนหมายเลขโทรศัพท์ที่เก็บเอาไว้ใน NVRAM ตำแหน่งที่ 2

- ATE เปิดและปิด

เอคโค่(Echo)เป็นคำสั่งที่ใช้สำหรับควบคุมการแสดงตัวอักษรที่ผู้ใช้งานพิมพ์ออกมาทางจอภาพ ถ้าเอคโค่ เป็นปิด(OFF) ก็หมายความว่าเมื่อกดปุ่มใด ๆ แล้ว ผู้ใช้จะไม่สามารถเห็นตัวอักษรบนหน้าจอในขณะที่กำลังพิมพ์เข้าไป ดังตัวอย่างเช่น ถ้าหากผู้ใช้พิมพ์คำว่า ATZ และมีสถานะของเอคโค่เป็นปิด ผู้ใช้จะไม่สามารถมองเห็นคำว่า ATZ บนหน้าจอภาพ ส่วนลักษณะการใช้คำสั่งจะเป็นดังนี้

ATE0 หรือ ATE : ECHO OFF

ATE1 : ECHO ON (Default)

- ATH ยกหูและวางหูโทรศัพท์

คำสั่งนี้ทำให้โมเด็มวางหูหรือยกหูโทรศัพท์ โดยมีรูปแบบดังนี้

ATH0 หรือ ATH : ให้โมเด็มวางหูโทรศัพท์

ATH1 : ให้โมเด็มยกหูโทรศัพท์

คำสั่ง ATH0 หรือ ATH มักจะใช้หลังจากที่ส่ง Escape Sequence (+++) ให้แก่โมเด็ม เพื่อให้โมเด็มกลับเข้าสู่สถานะคำสั่ง ซึ่งโปรแกรมสื่อสารทั่ว ๆ ไปมักจะใช้วิธีนี้ เพื่อทำการหยุดการสื่อสารหรือตัดสาย (Dis-connect)

- **ATL การควบคุมความดังของเสียง**

คำสั่งนี้ทำให้สามารถควบคุมความดังของเสียงจากลำโพงของโมเด็มได้ แต่โมเด็มบางรุ่นที่เป็นแบบแพ็คเกจหรือเป็นกล่องเล็ก ๆ ใช้คีย์อยู่ภายนอกที่ซึ่งบางประเภทอาจจะไม่มีลำโพง ดังนั้นจึงไม่สนับสนุนคำสั่งนี้ รูปแบบการใช้จะเป็นดังนี้

ATL0 หรือ ATL : หมายถึงให้มีเสียงเบาที่สุด

ATL1 : หมายถึงให้มีเสียงดังปานกลาง

ATL2 : หมายถึงให้มีเสียงดังที่สุด

แต่ถ้าต้องการให้ปิดเสียงของลำโพงจะต้องเปลี่ยนไปใช้ชุดคำสั่ง ATMn ดังหัวข้อต่อไป

- **ATM ควบคุมการเกิดเสียง**

ลักษณะของคำสั่ง จะมีรูปแบบดังนี้

ATM0 หรือ ATM: หมายถึง ให้โมเด็มปิดเสียงทั้งหมด

ATM1 : หมายถึง ให้โมเด็มส่งเสียงได้ในขณะที่หมุนหมายเลขโทรศัพท์และปิดเสียงเมื่อคีย์โมเด็มกับปลายทางเรียบร้อยแล้ว

ATM2 : หมายถึง อนุญาตให้โมเด็มส่งเสียงออกจากได้ลำโพงตลอดเวลา

ATM3 : หมายถึง ให้โมเด็มปิดเสียง ในขณะที่หมุนหมายเลขโทรศัพท์ และหลังจากนั้นก็ทำการเปิดเสียงจนกระทั่งได้รับสัญญาณพาหะโมเด็มปลายทางจึงจะปิดเสียงอีกครั้งหนึ่ง

- **ATO กลับสู่สถานะออนไลน์**

การใช้คำสั่งนี้ในสถานะคำสั่ง (Command state) จะทำให้โมเด็มกลับเข้าไปสู่สถานะออนไลน์อีกครั้งหนึ่ง มักจะใช้ในกรณีที่โมเด็มกำลังอยู่ในสถานะออนไลน์ และผู้ใช้กดรหัส Escape Sequence (+++) ออกมาสู่สถานะคำสั่งเพื่อออกคำสั่ง AT ต่าง ๆ แก่โมเด็มแล้วต้องการกลับเข้าไปสู่สถานะออนไลน์อีกครั้งหนึ่ง

- **ATQ ควบคุม Result code**

คำสั่งนี้ใช้ควบคุมการตอบสนองคำสั่งจากโมเด็มสู่พีซี โดยปกติแล้วเมื่อโมเด็มได้รับคำสั่งใด ๆ ก็จะส่งชุดอักขระกลับออกมาอย่างเช่น "OK" "CONNECT" หรือ "ERROR" เป็นต้น รูปแบบการใช้คำสั่ง มีดังต่อไปนี้

ATQ0 หรือ ATQ : อนุญาตให้โมเด็มตอบสนองคำสั่งออกมาได้

ATQ1 : ไม่อนุญาตให้โมเด็มตอบสนองคำสั่งออกมา

ATQ2 : ให้โมเด็มตอบสนองคำสั่งออกมา ถ้าเป็นโมเด็มตัวต้นทาง
และไม่ให้ตอบสนองคำสั่งออกมาถ้าเป็นโมเด็มปลายทาง

- **ATS คำสั่งตั้งค่าให้แก่วีธีเตอร์ S**

ชุดคำสั่ง ATS จะช่วยให้สามารถขูดและเปลี่ยนแปลงค่าที่อยู่ในรีจิสเตอร์ S ต่าง ๆ ได้
สำหรับการตั้งค่ารีจิสเตอร์ S มีรูปแบบดังนี้ $ATSx=y$ โดยที่ x เป็นหมายเลขรีจิสเตอร์ S และ
y หมายถึงค่าที่ต้องการบรรจุลงไปในรีจิสเตอร์ S ตัวนั้น ๆ ดังตัวอย่าง ถ้าหากต้องการ
กำหนดให้รีจิสเตอร์ S ตัวที่ 11 มีค่าเท่ากับ 50 ก็ให้ใช้คำสั่งดังนี้ $ATS11=50$ และสำหรับการ
ขูดค่าในรีจิสเตอร์ S ก็จะมีรูปแบบดังนี้ $ATSx?$ หรืออาจจะใช้คำสั่ง AT&V เพื่อขูดค่าใน
รีจิสเตอร์ S ทั้งหมดรวมทั้งข้อมูลการเซตทั้งหมดที่เก็บเอาไว้ใน NVRAM สำหรับรีจิสเตอร์
S ที่สำคัญต่าง ๆ จะมีดังต่อไปนี้

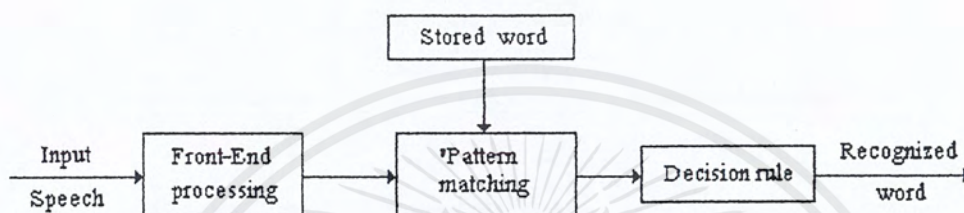
- S0 : ใช้เก็บค่าจำนวนครั้งของเสียงกระดิ่งโทรศัพท์ ก่อนที่โมเด็มจะรับสาย
โทรศัพท์ สามารถตั้งค่าให้สูงได้ถึง 255 (หมายถึง ให้เสียงกระดิ่งดัง
255 ครั้งแล้ว โมเด็มจึงจะรับโทรศัพท์) หรือถ้าตั้งเป็น 0 ก็มีความ
หมายว่าไม่ให้โมเด็มรับสายโทรเข้า
- S1 : ใช้เก็บค่าจำนวนครั้งของเสียงกระดิ่ง เป็นรีจิสเตอร์แบบอ่านอย่างเดียว
ไม่สามารถจะเปลี่ยนแปลงค่าได้โดยการใช้คำสั่ง AT ใด ๆ
- S2 : รีจิสเตอร์นี้ใช้เก็บค่ารหัส ASCII (ในลักษณะเลขฐาน 10) ที่ต้องการใช้
เป็นรหัส Escape Sequence โดยปกติแล้วจะมีดีฟอลต์เป็น 43 ซึ่งตรง
กับตัวอักษร “+”
- S3 : รีจิสเตอร์ตัวนี้ใช้เก็บค่ารหัส ASCII (ในลักษณะเลขฐาน 10) ที่ต้องการ
ใช้เป็นรหัส CR (Carriage Return) โดยปกติแล้วจะมีดีฟอลต์เป็น 13 ซึ่ง
ก็จะเหมือนกับการกดปุ่ม Enter
- S4 : รีจิสเตอร์ตัวนี้ใช้เก็บค่ารหัส ASCII (ในลักษณะเลขฐาน 10) ที่ต้องการ
ใช้เป็นรหัส Line Feed โดยปกติแล้วจะมีดีฟอลต์เป็น 10
- S5 : รีจิสเตอร์ตัวนี้ใช้เก็บค่ารหัส ASCII (ในลักษณะเลขฐาน 10) ที่ต้องการใช้
เป็นรหัส Back-space โดยปกติแล้วจะมีดีฟอลต์เป็น 8 ซึ่งก็เหมือนกับการ
กดปุ่ม Backspace
- S6 : ใช้เก็บค่าของเวลาในการหยุดรอ ก่อนที่จะหมุนหมายเลขโทรศัพท์ออก
มีหน่วยเป็นวินาที ซึ่งค่าในรีจิสเตอร์ S6 นี้จะใช้งานก็ต่อเมื่อโมเด็มถูก
สั่งให้ไม่สนใจฟังไคอัลโทน หรือคำสั่ง ATXn หลังจากนั้นเมื่อ
ต้องการให้โมเด็มหมุนหมายเลขโทรศัพท์ออก โมเด็มก็จะไม่สนใจว่า
มีเสียงไคอัลโทนหรือไม่และจะรอเป็นเวลา S6 จึงหมุนหมายเลข
โทรศัพท์ออก ค่าในรีจิสเตอร์ S6 นี้สามารถมีได้ตั้งแต่ 1 ไปจนถึง 255
วินาที โดยมีดีฟอลต์เป็น 50 วินาที

- S7 : ใช้เก็บค่าของเวลาในการรอฟังสัญญาณพาหะของโมเด็มปลายทาง ซึ่งมีหน่วยเป็นวินาที หลังจากทีโมเด็มหมุนหมายเลขโทรศัพท์ออก หรือรอรับสายโทรศัพท์เข้าโมเด็มจะรอฟังสัญญาณเป็นเวลา S7 วินาที ซึ่งในกรณีที่ต้องการโทรศัพท์ระหว่างประเทศ ก็อาจจะต้องตั้งค่า S7 ให้มากกว่า 50 วินาที แล้วแต่ความเหมาะสม
- S8 : ใช้เก็บค่าของเวลาในการรอ มีหน่วยเป็นวินาที หลังจากทีโมเด็มได้รับพารามิเตอร์ “,” ในคำสั่ง ATD โมเด็มก็จะหยุดรอเป็นเวลา S8 วินาที ก่อนที่จะทำคำสั่ง หรือพารามิเตอร์ตัวต่อไป S8 นี้จะสามารถมีค่าได้ตั้งแต่ 2 ไปจนถึง 255 วินาที แต่โดยปกติแล้วจะมีดีฟอลต์เป็น 2 วินาที
- S9 : ใช้เก็บค่าของช่วงเวลาที่โมเด็มรอรับทราบสัญญาณพาหะ ของโมเด็มที่ปลายทาง มีหน่วยเป็น 1/10 วินาที โดยโมเด็มจะรอสัญญาณพาหะจากปลายทางภายในช่วงเวลาดังกล่าว หลังจากนั้นโมเด็มก็จะส่งสัญญาณ CD (Carrier Detect) ที่ขา 8 ของขั้วต่อ RS-232-C ไปยังพีซี โดยผู้ใช้งานสามารถกำหนดค่าใน S9 ได้ตั้งแต่ 1 ถึง 255
- S10 : มีหน่วยเป็น 1/10 วินาที มีค่าได้ตั้งแต่ 1 จนถึง 255 มักจะใช้อยู่ 2 กรณี ซึ่งกรณีแรกจะใช้เก็บช่วงเวลาที่โมเด็มจะทำการรอสัญญาณพาหะที่ขาดหายไปในขณะที่กำลังออนไลน์ ซึ่งอาจจะเกิดจากสัญญาณรบกวน และถ้าสัญญาณพาหะยังคงขาดตอนไปในช่วงเวลาดังกล่าว โมเด็มก็จะวางสายทันที ส่วนกรณีที่สองก็คือ จะใช้เก็บค่าของช่วงเวลาในการรอสัญญาณพาหะก่อนที่จะวางสายแล้วจะให้โมเด็มตรวจสอบให้แน่ใจว่าการสื่อสารนั้นถูกตัดขาดอย่างแน่นอนหรือไม่ โดยปกติแล้วจะมีค่าดีฟอลต์เป็น 14 ซึ่งก็เพื่อให้โมเด็มหยุดรอสัญญาณพาหะเป็นเวลา 1.4 วินาทีหลังจากการสื่อสารถูกตัดขาดไป และในทางกลับกันถ้าต้องการจะให้โมเด็มค้างสัญญาณ CD ไว้ตลอดเวลา (เพื่อเป็นการ “หลอก” เครื่องพีซี) ก็สามารถตั้งค่าไว้เป็น 255 ได้เช่นกัน
- S11 : ใช้เก็บค่าของช่วงเวลาในการหมุนโทรศัพท์แบบโทน ซึ่งแต่ละหมายเลขมีหน่วยเป็นมิลลิวินาที ในขณะที่โมเด็มกำลังหมุนหมายเลขโทรศัพท์ออก ก็จะเว้นช่วงเวลาไว้ประมาณ S11 มิลลิวินาทีระหว่างหมายเลขโทรศัพท์แต่ละตัว ซึ่งค่าของ S11 นี้จะมีผลต่อการหมุนแบบ Pulse (ATDP) ผู้ใช้งานสามารถกำหนดค่าใน S11 ได้ตั้งแต่ 50 ไปจนถึง 255 ได้ (0.05-0.255 วินาที) โดยมีดีฟอลต์เป็น 95 (0.095 วินาที)
- S12 : ใช้เก็บค่าของช่วงเวลาที่โมเด็มหยุดรอ Escape Sequence มีหน่วยเป็น 1/50 วินาที ถ้าหากผู้ใช้งานต้องการออกจากสถานะออนไลน์ กลับเข้าสู่สถานะคำสั่ง(Command state) จะต้องหยุดรอเป็นเวลา S12 วินาที แล้ว

จึงสามารถส่ง Escape Sequence ให้แก่โมเด็มได้ เป็นข้อมูลธรรมดาผ่านสายโทรศัพท์ออกไป รีจิสเตอร์ S12 สามารถมีค่าได้ตั้งแต่ 0 ไปจนถึง 255 (0-5.1 วินาที) โดยมีค่าดีฟอลต์เป็น 50 (1 วินาที)

2.3 หลักการวิเคราะห์เสียง

จากการศึกษารูปแบบการรับรู้และจำเสียงพูดแบบคำเดี่ยว (Isolated word recognition) มีรูปแบบและหลักการพื้นฐานดังรูปที่ 2.2



รูปที่ 2.2 แสดง Basic isolate-word recognition system

สัญญาณที่เข้าโมเด็มเป็นคลื่นเสียง ส่วนผลลัพธ์เป็นเสียงวิเคราะห์ที่ได้จากสัญญาณเข้า โมเด็มนี้มีการใช้กันอย่างแพร่หลาย แบ่งเป็นส่วนต่าง ๆ ดังนี้

- วิเคราะห์เสียง(Front-End processing) เป็นขั้นตอนการวิเคราะห์เสียงที่ได้รับมาให้อยู่ในรูปแบบที่สามารถนำไปวิเคราะห์ได้
- ปรับรูปแบบเสียง(Pattern matching) เป็นขั้นตอนการสร้างโมเดลของเสียง
- กฎการตัดสินใจ(Decision rule) เป็นขั้นตอนการตัดสินใจในการเลือกโมเดลเสียงที่ใกล้เคียงกับเสียงทดสอบมากที่สุด

2.3.1 วิเคราะห์เสียง(Front-End processing)

ขั้นตอนการวิเคราะห์เสียง เป็นขั้นตอนที่ทำการแปลงข้อมูลที่มีอยู่มากมายเป็นส่วนเล็ก ๆ ที่สามารถแสดงคุณสมบัติของคลื่นเสียงนั้น ๆ โดยผ่านขั้นตอน ดังนี้

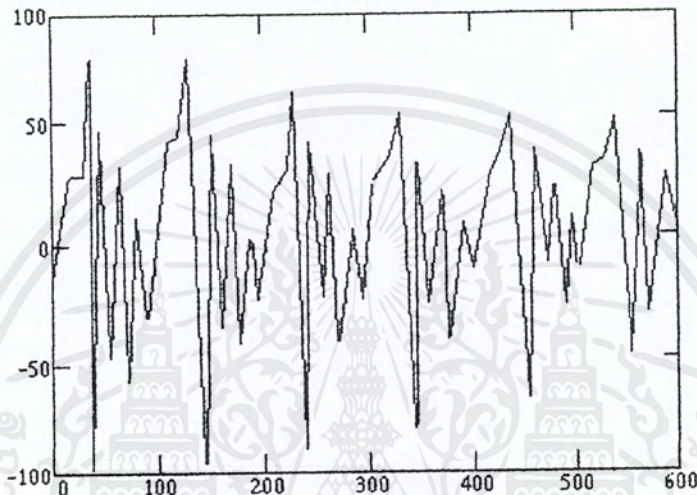
2.3.1.1 การคำนวณหาค่าพิตช์(Pitch)

2.3.1.2 การประมาณเชิงเส้น (Linear predictive coding : LPC)

2.3.1.3 การควอนไทซ์ (Quantization)

2.3.1.1 การคำนวณหาพิตช์

คุณลักษณะของสัญญาณเสียงพูดเป็นสัญญาณที่มีคุณลักษณะไม่คงที่ (Nonstationary Signal) มีการเปลี่ยนแปลงอย่างช้า ๆ แต่อย่างไรก็ตามเมื่อพิจารณาเฉพาะส่วนย่อยของสัญญาณเสียงพูด นั่นคือในช่วงเวลาไม่เกิน 30 มิลลิวินาทีแล้ว สามารถอนุมานได้ว่า เสียงพูดเป็นสัญญาณที่มีคุณลักษณะคงที่ ดังนั้นสัญญาณเสียงพูดจึงถูกแบ่งเป็นส่วนย่อย ๆ เรียกว่าเฟรม (frame) จากนั้นจึงทำการประมวลผลสัญญาณแต่ละเฟรม คุณลักษณะที่สำคัญอย่างหนึ่งของเสียงพูดคือพิตช์(Pitch)ซึ่งเป็นความถี่พื้นฐานที่เกิดจากการสั่นในหลอดเสียง ดังตัวอย่างในรูป 2.3



รูปที่ 2.3 ตัวอย่างคาบเวลาพิตช์ของเสียงพูด

พิตช์เป็นตัวกำหนดระดับความสูงต่ำของเสียง เนื่องจากว่าภาษาไทยเป็นภาษาที่มีเสียงวรรณยุกต์ (tonal language) คือมีระดับโทนเสียงสามัญ เอก โท ตรี และจัตวา ซึ่งแต่ละโทนเสียงก็จะให้ความหมายที่แตกต่างกันออกไป ดังนั้นพิตช์จึงเป็นสิ่งที่สำคัญในการแยกแยะคำในภาษาไทย

การหาจุดเริ่มต้นและจุดสุดท้ายของเสียง

ในระบบการรู้และจำเสียงแบบคำโดด ความถูกต้องในการหาจุดเริ่มต้นและจุดสุดท้ายของเสียงมีความสำคัญอยู่ 2 ประการคือ ความคลาดเคลื่อนในการรู้และจำ และปริมาณข้อมูลที่ใช้ในการคำนวณ ถ้ากำหนดให้สัญญาณเสียง $s(t)$ ถูกแปลงเป็นเชิงเลข(digitize)เป็นลำดับของข้อมูลเสียง $x(m)$ โดยที่

$$\{ x[m] \} = \{ x[1], x[2], x[3], \dots, x[m], \dots, x[K] \} \quad (2.1)$$

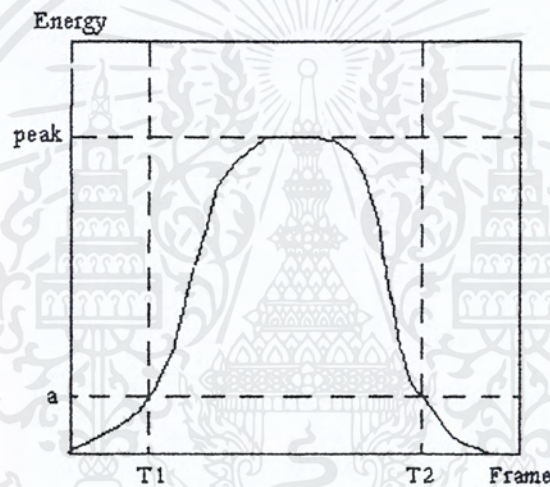
จะสามารถหาค่าพลังงานของสัญญาณได้คือ

$$E_1(0) = \sum_{m=0}^{N-1} x^2(m) \quad (2.2)$$

และเพื่อลดการคำนวณสามารถใช้ขนาดของสัญญาณแทนค่ากำลังสอง ดังสมการ

$$M(0) = \sum_{m=0}^{N-1} |x[m]| \quad (2.3)$$

โดยที่ $M_1(0)$, N , และ 1 คือ ค่าพลังงาน, ขนาดของเฟรม (frame size), และลำดับที่ของเฟรม หลังจากนั้น จะกำหนดระดับของพลังงานขีดเริ่มเปลี่ยน(threshold) a เพื่อใช้ในการหาจุดเริ่มต้นและจุดสุดท้ายของ สัญญาณเสียง ดังแสดงในรูปที่ 2.4



รูปที่ 2.4 เฟรมเริ่มต้น (T1) และเฟรมสุดท้าย(T2)ของพลังงานของเสียงพูดที่ใช้ในการคำนวณ

โดยกำหนดให้ a มีค่า 10% ของค่าของเฟรมข้อมูลที่มีพลังงานมากที่สุด โดยค่า 10% นี้ได้มาจากการทดลองที่ให้ผลในการตัดข้อมูลที่ไม่ต้องการออกไปได้ดีที่สุด เขียนได้ดังสมการ

$$a = 0.1 M_{\max} \quad (2.4)$$

ซึ่งก็คือการตัดเฟรมที่มีพลังงานต่ำกว่า 10% ของเฟรมที่มีพลังงานมากที่สุด แล้วคำนวณหาค่า $T1$ และ $T2$ ใช้เป็นเฟรมเริ่มต้นและเฟรมสุดท้ายของข้อมูลที่ใช้ในการคำนวณ

การคำนวณหาออร์โตคอร์รีเรชัน

เสียงวรรณยุกต์แต่ละเสียงเกิดขึ้นเกิดจากการเปลี่ยนแปลงของพิตช์ซึ่งการคำนวณหา นั้นทำได้หลายวิธี วิธีหนึ่งซึ่งให้ผลดีคือใช้ออร์โตคอร์รีเรชันหรืออัตโนมัติสหสัมพันธ์(Autocorrelation)ดังสมการ

$$\phi(k) = \sum_{m=-\infty}^{\infty} x(m)x(m+k) \quad (2.5)$$

โดยที่ $X(m)$ เป็นสัญญาณอินพุต ถ้า $x(m)$ เป็นสัญญาณที่มีคาบเวลาคงที่เท่ากับ P จะได้ว่า

$$\phi(k) = \phi(k+P) \quad (2.6)$$

นอกจากนี้ ออร์โตคอร์รีเรชันยังมีคุณสมบัติที่สำคัญคือ

$$\phi(k) = \phi(-k) \quad (2.7)$$

และมีค่าสูงสุดเมื่อ $k=0$ นั่นคือ

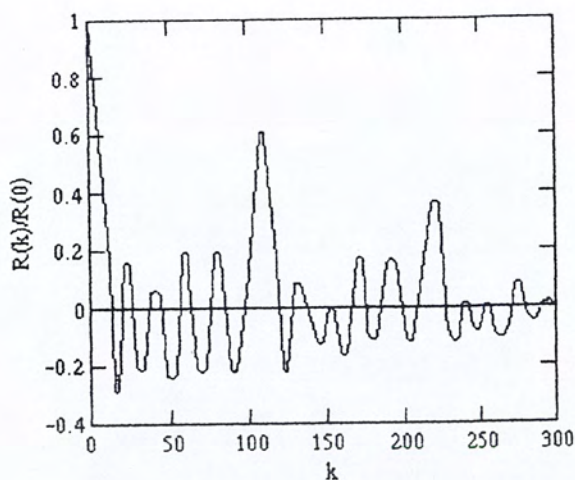
$$|\phi(k)| \leq |\phi(k+P)| \quad (2.8)$$

สำหรับสัญญาณที่มีคาบเวลา P ออร์โตคอร์รีเรชันจะมีค่าสูงสุดที่แซมเปิล $0, P, 2P, \dots$ ดังนั้นจึงสามารถคำนวณหาคาบเวลาของสัญญาณได้โดยการค้นหาตำแหน่งแรกที่ทำให้ออร์โตคอร์รีเรชันมีค่าสูงสุด ในการประมวลสัญญาณเสียงพูดนั้นใช้ออร์โตคอร์รีเรชันสำหรับช่วงเวลานั้น ๆ โดยให้คำนิยามว่า

$$R_n = \sum_{m=-\infty}^{\infty} [x(m)w(n-m)][x(m+k)w(n-m-k)] \quad (2.9)$$

โดยที่ $w(n)$ เป็นฟังก์ชันหน้าต่าง ซึ่งอาจจะเป็นแบบสามเหลี่ยม สี่เหลี่ยม หรือแบบแฮมมิง ทำการคำนวณค่าของ $R_n(k)$ ในช่วงของ k จากน้อยที่สุดเท่าที่เป็นไปได้คือช่วงคาบเวลาของพิคซ์ของเสียงผู้หญิงซึ่งมีค่าเท่ากับ 3 มิลลิวินาที จนถึงค่า k สูงสุดเท่าที่เป็นไปได้คือช่วงคาบเวลาพิคซ์ของเสียงผู้ชายซึ่งมีค่าเท่ากับ 20 มิลลิวินาที ถ้าใช้อัตราการซิกตัวอย่าง (Sampling rate) 10 kHz ก็จะใช้ k อยู่ในช่วง 30 ถึง 200 ถ้าเลือกให้ $w(n)$ มีความยาว N จะต้องมีค่ามากกว่า k มาก ๆ ($N \gg k$) ก็จะได้นิยามของออร์โตคอร์รีเรชันสำหรับช่วงเวลานั้น ๆ ดังนี้

$$R_n = \sum_{m=0}^{N-1-k} [x(m)w(n-m)][x(m+k)w(n-m-k)] \quad (2.10)$$

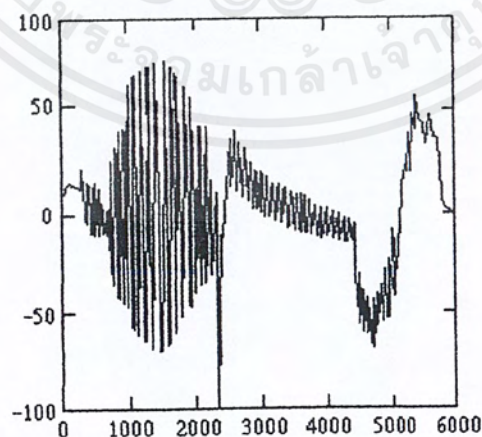


รูปที่ 2.5 ออโต้คอร์เรชันของสัญญาณเสียงพูด

รูปที่ 2.5 แสดงให้เห็นถึงออโต้คอร์เรชันแบบช่วงเวลาสั้น ๆ ของสัญญาณเสียงพูด จะเห็นว่ามีค่าสูงสุดเมื่อ $k = 0$ และมีค่าสูงสุดอีกครั้งเมื่อ $k = 110$ ดังนั้นคาบเวลาของพิชจึงเป็น 110 แซมเปิ้ล หรือเท่ากับ 11 มิลลิวินาที เมื่ออัตราการซิกตัวอย่าง คือ 10 kHz นั่นคือความถี่ของพิชเท่ากับ 91 Hz

การปรับแต่งค่าพิช

เนื่องจากสัญญาณเสียงพูดแต่ละคำพูดจะมีความแตกต่างกันของค่าพิชและความต่อเนื่องของพิช โดยการเปลี่ยนแปลงของพิชจะขึ้นอยู่กับเสียงวรรณยุกต์ ส่วนความต่อเนื่องของพิชจะขึ้นอยู่กับองค์ประกอบของเสียงพยัญชนะเป็นหลัก ดังตัวอย่างในรูปที่ 2.6 ซึ่งจะเห็นได้ว่าข้อมูลบางช่วงจะไม่มีคาบพิชอยู่



รูปที่ 2.6 ตัวอย่างเสียงพูดของคำว่า “จิม”

การคำนวณค่าออร์โตคอร์เรชันจากทุกเฟรมของสัญญาณเสียงพูดเพื่อใช้ในการหาค่าพิตช์จะทำให้ได้ข้อมูลพิตช์ที่มีความผิดพลาดปะปนอยู่อันเนื่องมาจากเสียงพยัญชนะดังกล่าว จึงทำให้ต้องมีการตัดข้อมูลพิตช์ที่ปะปนนี้ออก โดยใช้วิธีเทียบกับค่าเฉลี่ยพิตช์รวมของเสียงทั้งหมดที่ใช้เป็นต้นแบบในการฝึกจำ และค่ามัธยฐานของเสียงนั้นเฉลี่ยกัน ดังนี้

$$P_i = (\text{Mean_of_Pitch}_{\text{all words}} + \text{Median_of_Pitch}) / 2$$

$$k = 1$$

For j = 1 to length_of_Pitch

$$\text{If } |\text{Pitch}(j) - P_i| \leq 0.3 P_i$$

$$\text{NewPitch}(k) = \text{Pitch}(j)$$

$$k = k + 1$$

Endif

Next j

โดย P_i เป็นค่าเฉลี่ยรวมกับค่ามัธยฐานของเสียงที่ i ซึ่งพิตช์ที่มีขนาดมากกว่า 30% ของค่าเฉลี่ย P_i จะถูกตัดออก (ค่า 30% นี้ได้มาจากการทดลองที่ทำให้ค่าพิตช์มีความต่อเนื่องมากที่สุด) ทำให้ค่าพิตช์ใหม่ที่ได้นั้นมีความต่อเนื่องดีขึ้น การตัดค่าพิตช์ออกในลักษณะนี้จะให้ค่าพิตช์ที่ได้จากคำพูดที่มีเสียงพยัญชนะผสมอยู่มีความเหมาะสมที่จะใช้ในการฝึกจำ ซึ่งจะทำให้แบบจำลองของเสียงวรรณยุกต์ที่ได้มีความสามารถในการจดจำเสียงวรรณยุกต์ที่มีเสียงพยัญชนะต่าง ๆ ผสมอยู่

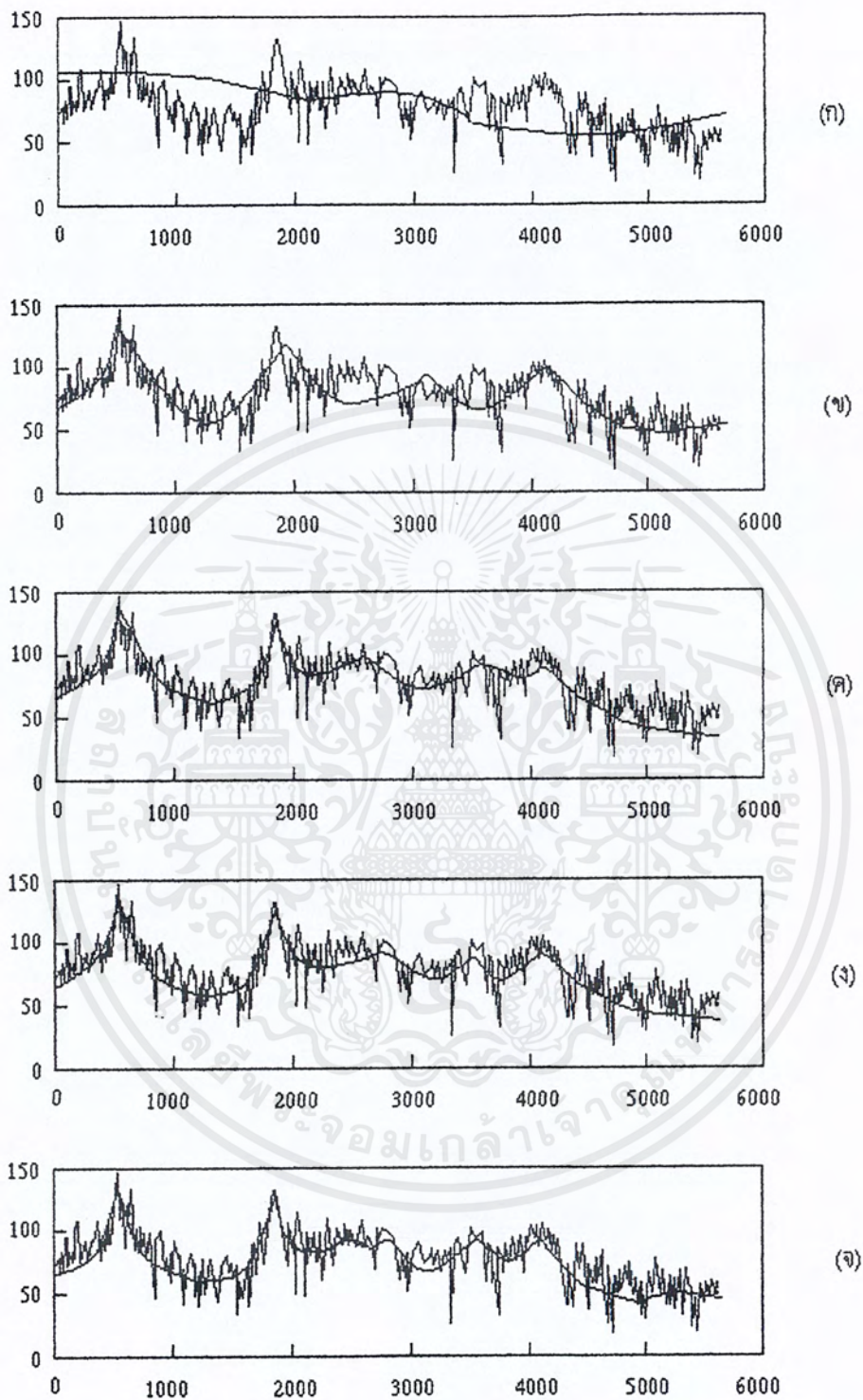
2.3.1.2 การประมาณเชิงเส้น

การประมาณเชิงเส้น (LPC : Linear Predictive Coding) เป็นการประมาณค่าของสัญญาณจากผลรวมเชิงเส้นของสัญญาณก่อนหน้า ซึ่งผลที่ได้จะเสมือนกับเป็นการประมาณค่าสเปกตรัมของสัญญาณนั่นเองดังในรูปที่ 2.7 ส่วนวิธีในการหาค่าสัมประสิทธิ์ LPC(α) และอัตราขยาย (G) ซึ่งต้องมีการเตรียมข้อมูลก่อนการวิเคราะห์เพื่อเตรียมสัญญาณเข้าเป็นดังรูปที่ 2.8

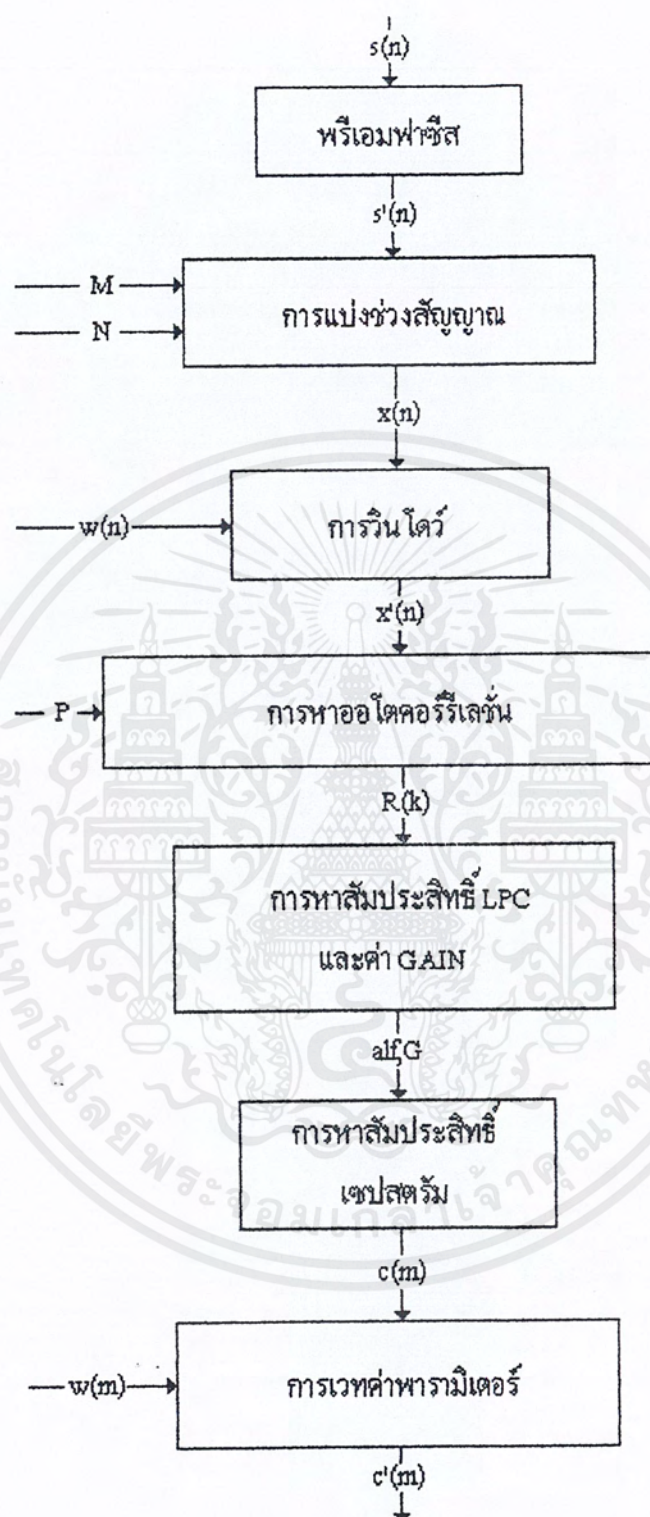
การพรีเอมฟาซิส

เนื่องจากสัญญาณเสียงพูดของมนุษย์ จะมีองค์ประกอบส่วนใหญ่อยู่บริเวณความถี่ต่ำ เมื่อเทียบกับแถบความถี่ที่ปฏิบัติงาน (bandwidth) ไม่เกิน 5 kHz ดังนั้น เพื่อให้อัตราส่วนสัญญาณเสียงต่อสัญญาณรบกวน (signal to noise ratio : SNR) มีค่าค่อนข้างคงที่ ตลอดช่วงความถี่ที่ปฏิบัติงานนี้ จึงต้องมีการพรีเอมฟาซิสหรือการเน้นล่วงหน้า (pre-emphasis) โดยเน้นความถี่สูงให้มีขนาดสูงขึ้น นั่นคือการพรีเอมฟาซิสก็คือการกรองสัญญาณด้วยวงจรกรองความถี่สูงผ่าน (high pass filter) ซึ่งมักนิยมใช้วงจรกรองอันดับหนึ่ง มีฟังก์ชันถ่ายโอนเป็น

$$H(z) = 1 - a \cdot z^{-1}, \quad 0.9 \leq a \leq 1.0$$



รูปที่ 2.7 แสดงสเปกตรัมของสัญญาณเสียงเทียบกับสเปกตรัมที่ได้จากการประมาณเชิงเส้นที่มี
อันดับ (LPC order) เท่ากับ p ต่าง ๆ (ก) $p=4$ (ข) $p=8$ (ค) $p=12$ (ง) $p=16$ (จ) $p=20$

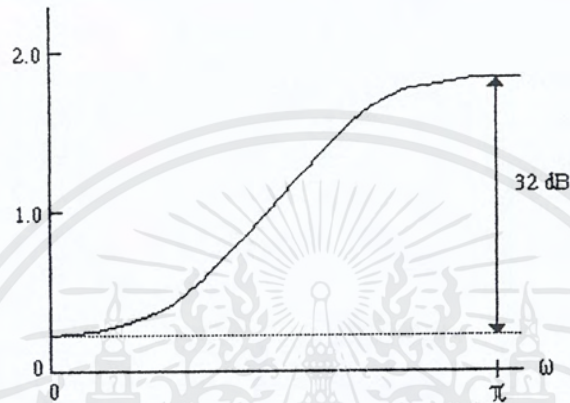


รูปที่ 2.8 แสดงขั้นตอนการเตรียมสัญญาณในการวิเคราะห์

กรณีนี้เอาค่าพหุคูณการพรีเอมฟาสซิส $s'(n)$ มีความสัมพันธ์กับอินพุต $s(n)$ ดังสมการผลต่างสืบเนื่อง

$$s'(n) = s(n) - a \cdot s(n-1)$$

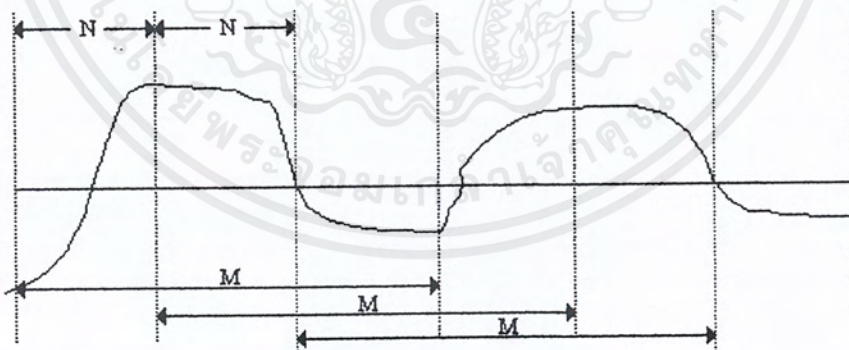
ยิ่งค่า a ใกล้ 1 เท่าใด ความถี่สูงก็จะถูกขยายมากขึ้นเท่านั้น ค่า a ที่นิยมสำหรับใช้ในการหาพารามิเตอร์ของ LPC คือ 0.95 เมื่อนำฟังก์ชันถ่ายโอนมาพรีอตรกราฟของขนาดเทียบกับความถี่จะได้ดังรูป 2.9



รูปที่ 2.9 แสดงขนาดสเปกตรัมของฟังก์ชันถ่ายโอนของการพรีเอมฟาสซิส

การแบ่งช่วงสัญญาณ

สัญญาณที่ผ่านการพรีเอมฟาสซิสแล้ว $s'(n)$ จะถูกตัดแบ่งออกเป็นช่วง ๆ หรือ เฟรม ช่วงละ M ตัวอย่างสัญญาณ การวิเคราะห์จะวิเคราะห์ทีละช่วงของแต่ละ M ตัวอย่างสัญญาณ ดังรูปที่ 2.10



รูปที่ 2.10 แสดงการแบ่งช่วงสัญญาณที่ใช้ในการวิเคราะห์

โดยช่วงในการวิเคราะห์แต่ละช่วงจะถูกเลื่อนไปเป็นระยะทาง N ช่วงสัญญาณ จะเห็นได้ว่า ถ้าค่า N โตกว่าค่า M ในการเลื่อนของช่วงในการวิเคราะห์จะทำให้บางสัญญาณไม่ถูกใช้ในการวิเคราะห์ ก็จะเป็นการสูญเสียส่วนหนึ่งทำให้ผลที่ได้ไม่ถูกต้องเท่าที่ควร ถ้าค่า N เล็กกว่าค่า M ก็จะทำให้ตัวอย่าง

สัญญาณทุกตัวถูกนำมาวิเคราะห์ ยิ่งค่า N เล็กเท่าใด ความแม่นยำในการวิเคราะห์ก็จะยิ่งสูงขึ้นเท่านั้น แต่ก็จะทำให้การคำนวณช้าลง

หลักการกำหนดขนาดของเฟรม

1. เฟรมจะต้องสั้นพอที่คุณสมบัติของเสียงที่สนใจจะวิเคราะห์นั้นยังไม่เปลี่ยนแปลงในเฟรม
2. เฟรมจะต้องยาวพอที่จำนวนตัวอย่างสัญญาณในเฟรมสามารถนำมาคำนวณหาคุณสมบัติที่ต้องการได้
3. เฟรมที่ติดกันไม่ควรจะสั้นจนกระโดดข้ามข้อมูลบางส่วนไป แต่ควรเลื่อนเฟรมให้น้อยกว่าขนาดของเฟรม

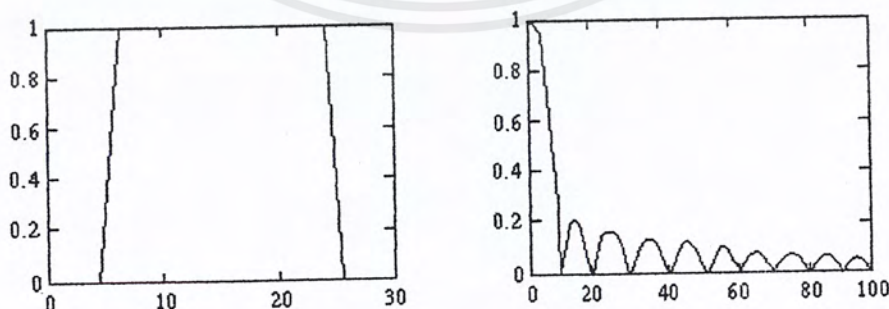
การวินโดว์

ถ้าพิจารณาช่วงสัญญาณ M ตัวอย่างสัญญาณของช่วงใด ๆ ที่ตัดมาวิเคราะห์ จะเห็นว่าที่ขอบของเฟรมที่ตัดมานี้มีความไม่ต่อเนื่องของสัญญาณ ถ้ามองในโดเมนความถี่สูง ก็จะมีความถี่สูงเกิดขึ้น ดังนั้นเพื่อที่จะลดองค์ประกอบทางความถี่สูงเหล่านี้ จะคูณด้วยฟังก์ชันวินโดว์เพื่อลดความไม่ต่อเนื่องของสัญญาณที่ขอบ และไม่ทำให้สเปกตรัมของสัญญาณในช่วงความถี่ต่ำเปลี่ยนไปมากนัก ในที่นี่จะใช้ฟังก์ชันวินโดว์แฮมมิง (Hamming window function) ซึ่งนิยามโดยสมการดังนี้คือ

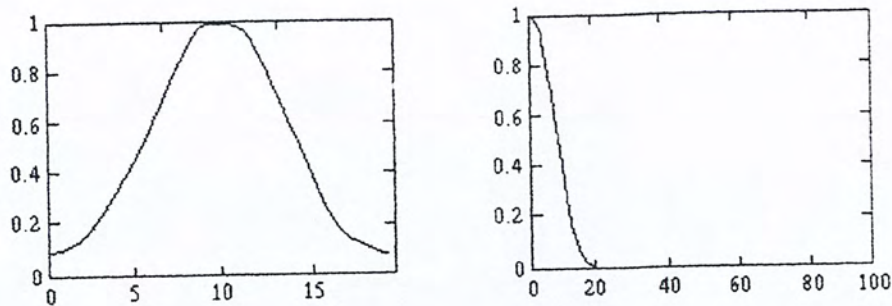
$$w(n) = 0.54 - 0.46 \cos\left(\frac{2\pi n}{N-1}\right) \quad \text{เมื่อ } n = 0, 1, 2, \dots, N-1 \quad (2.11)$$

รูปที่ 2.11 และ 2.12 แสดงองค์ประกอบทางเวลา และทางความถี่ของฟังก์ชันวินโดว์แบบสี่เหลี่ยมและแฮมมิง ตามลำดับ จะเห็นว่าสเปกตรัมของวินโดว์แฮมมิงมีความพลั่ว (ripple) น้อยกว่าของวินโดว์สี่เหลี่ยม ดังนั้น จากรูปที่ 2.7 จะได้ว่า

$$x'(n) = x(n) \cdot w(n) \quad (2.12)$$



รูปที่ 2.11 แสดงวินโดว์สี่เหลี่ยม (ก) ในโดเมนเวลา (ข) ในโดเมนความถี่



รูปที่ 2.12 แสดงวินโดว์แบบแสมมิ่ง (ก) ในโดเมนเวลา (ข) ในโดเมนความถี่

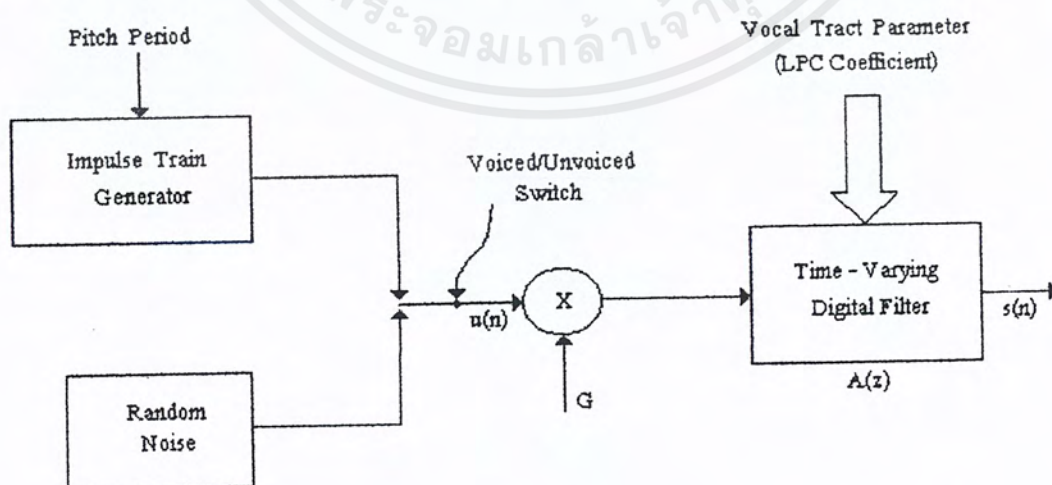
การหาค่าสัมประสิทธิ์ด้วยวิธีออร์ทอดอรัลเรชัน

จากหลักการพื้นฐานของการประมาณเชิงเส้นคือ การประมาณค่าสัญญาณจากผลรวมเชิงเส้นของสัญญาณก่อนหน้านี้ สมมติว่าสัญญาณเดิมเป็น $s(n)$ การประมาณค่าสัญญาณเป็น $s'(n)$ ดังนั้นสามารถอธิบายการประมาณเชิงเส้นได้ด้วยสมการต่อไปนี้

$$s'(n) = \sum_{k=1}^p \alpha_k s(n-k) \quad (2.13)$$

เมื่อ α_k เป็นค่าคงที่ เรียกวิธีการนี้ว่าการประมาณเชิงเส้นอันดับ p โดยมีเงื่อนไขว่าค่า α_k ที่ใช้ในการประมาณจะต้องทำให้ ผลรวมของกำลังสองของความคลาดเคลื่อน $\{s(n) - s'(n)\}^2$ มีค่าน้อยที่สุด นั่นคือ $\sum e^2(n) = \sum \{s(n) - s'(n)\}^2$ มีค่าต่ำที่สุด ซึ่งจะใช้การประมาณเชิงเส้นวิธีออร์ทอดอรัลเรชัน (Autocorrelation Method) หรือวิธีออสถสัมพัทธ์

จากหลักการพื้นฐานของการประมาณเชิงเส้น และแบบจำลองระบบสร้างสัญญาณเสียง สามารถเขียนบล็อกไดอะแกรมการทำการประมาณเชิงเส้นมาสร้างสัญญาณเสียงพูดได้ดังรูปที่ 2.13



รูปที่ 2.13 บล็อกไดอะแกรมแสดงแบบจำลองการสร้างสัญญาณเสียงพูดอย่างง่าย

จากรูปที่ 2.13 สามารถเขียนสมการได้เป็น

$$s(n) = G * u(n) + \sum_{k=1}^p \alpha_k s(n-k) \quad (2.14)$$

การประมาณเชิงเส้นโดยใช้สัมประสิทธิ์ $\{\alpha_k\}$ คือ

$$s'(n) = \sum_{k=1}^p \alpha_k s(n-k) \quad (2.15)$$

ดังนั้น ความคลาดเคลื่อน คือ

$$\begin{aligned} e(n) &= s(n) - s'(n) \\ &= s(n) - \sum_{k=1}^p \alpha_k s(n-k) \end{aligned} \quad (2.16)$$

ฟังก์ชันถ่ายโอนระหว่าง $e(n)$ และ $s(n)$ คือ

$$\begin{aligned} A(z) &= E(z) / S(z) \\ &= 1 - \sum_{k=1}^p \alpha_k z^{-k} \end{aligned} \quad (2.17)$$

จากสมการ (2.14) - (2.16) จะเห็นได้ว่า ถ้า $\{\alpha_k\} = \{a_k\}$ แล้ว

$$e(n) = G * u(n) \quad (2.18)$$

ดังนั้น ค่าผลรวมของกำลังสองของความคลาดเคลื่อน

$$\begin{aligned} E_n &= \sum_m e_n^2(m) \\ E_n &= \sum_m [s(m) - s'(m)]^2 \end{aligned} \quad (2.19)$$

โดยที่ n คือช่วงที่ n ของสัญญาณที่ใช้คำนวณ เพราะฉะนั้นเพื่อให้ได้ค่า E_n ค่าที่สุดจะต้องมีเงื่อนไขว่า

$$\frac{\partial E_n}{\partial \alpha_i} = 0 \quad \text{เมื่อ } i = 1, 2, 3, \dots, p$$

จากสมการ (2.19)

$$\frac{\partial E_n}{\partial \alpha_i} = -2s_n(m-i) \sum_m [s_n(m) \sum_{k=1}^p \alpha_k s_n(m-k)] \quad \text{เมื่อ } i = 1, 2, 3, \dots, p$$

$$= 2 \left[\sum_m s(m)s(m-i) - \sum_{k=1}^p \sum_m \alpha_k s(m-k)s(m-i) \right]$$

$$\frac{\partial E_n}{\partial \alpha_i} = 0 \quad \text{ก็ต่อเมื่อ}$$

$$\sum_{k=1}^p \alpha_k \sum_m s_n(m-k)s_n(m-i) = \sum_m s_n(m)s_n(m-i) \quad \text{เมื่อ } i = 1, 2, 3, \dots, p \quad (2.20)$$

ถ้ากำหนดให้ $\phi_n(i, j) = \sum_m s_n(m-k)s_n(m-i)$ เพราะฉะนั้น

$$\sum_{k=1}^p \alpha_k \phi_n(i, j) = \phi_n(i, 0) \quad (2.21)$$

โดยสมการ (2.19)-(2.20) จะได้ว่า

$$E_n = \sum_m s_n^2(m) - \sum_{k=1}^p \alpha_k \sum_m s_n(m)s_n(m-k)$$

และจาก $\phi_n(i, j) = \sum_m s_n(m-k)s_n(m-i)$

$$E_n = \phi_n(0, 0) - \sum_{k=1}^p \alpha_k \phi_n(0, k) \quad (2.22)$$

สมมติว่าใน 1 เฟรมของสัญญาณที่ศึกษามีจำนวนมี N ตัวอย่าง คือ $s_n(0), s_n(1), s_n(2), \dots, s_n(N-1)$ ในที่นี้ให้ $s_n(m)=0$ เมื่อ $m < 0$ หรือ $m > N-1$ เพราะฉะนั้น

$$\begin{aligned} \phi_n(i, j) &= \sum_m s_n(m-k)s_n(m-i) \\ &= \sum_{m=0}^{N-1-(i-k)} s_n(m)s_n(m+i-k) \quad 0 \leq k \leq p, 1 \leq i \leq p \end{aligned}$$

ให้
$$R_n(k) = \sum_{m=0}^{N-1-k} s_n(m)s_n(m+k) \quad \text{เมื่อ } k = 0, 1, 2, \dots, p \quad (2.23)$$

ดังนั้น จากสมการ (2.12) และสมการ(2.23) จะได้ว่า

$$R_n(k) = \sum_{m=0}^{N-1-k} x'(m)x'(m+k) \quad (2.24)$$

จากสมการที่ (2.21) จะได้ว่า

$$\sum_{k=1}^p \alpha_k R_n(i-k) = R_n(i) \quad \text{เมื่อ } i = 1, 2, 3, \dots, p \quad (2.25)$$

จากสมการ (2.25) เขียนให้อยู่ในรูปเมตริกซ์ได้เป็น

$$\begin{bmatrix} R_n(0) & R_n(1) & \dots & R_n(p-1) \\ R_n(1) & R_n(0) & \dots & R_n(p-2) \\ \vdots & \vdots & \ddots & \vdots \\ R_n(p-1) & R_n(p-2) & \dots & R_n(0) \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_p \end{bmatrix} = \begin{bmatrix} R_n(1) \\ R_n(2) \\ \vdots \\ R_n(p) \end{bmatrix} \quad (2.26)$$

หรือ
$$R_n \cdot \alpha = r_n \quad (2.27)$$

$$\text{เมื่อ } R_n = \begin{bmatrix} R_n(0) & R_n(1) & \dots & R_n(p-1) \\ R_n(1) & R_n(0) & \dots & R_n(p-2) \\ \vdots & \vdots & \ddots & \vdots \\ R_n(p-1) & R_n(p-2) & \dots & R_n(0) \end{bmatrix}, \alpha = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_p \end{bmatrix} \text{ และ } r_n = \begin{bmatrix} R_n(1) \\ R_n(2) \\ \vdots \\ R_n(p) \end{bmatrix}$$

การหาสัมประสิทธิ์ α และอัตราขยาย G

เมื่อได้ค่า $R_n(0), R_n(1), R_n(2), \dots, R_n(p)$ แล้วก็สามารถหา α ได้จากสมการ(2.27) นั่นคือ

$$\alpha = R_n^{-1} \cdot r_n \quad (2.28)$$

และจากสมการ (2.18) จะได้ว่า

$$e(n) = G \cdot u(n) \\ \therefore E_n = \sum_{m=0}^{N-1} e^2(m) = G^2 \sum_{m=0}^{N-1} u^2(m)$$

จากสมการ (2.22) จะได้ว่า

$$\begin{aligned} E_n &= \phi_n(0,0) - \sum_{k=1}^p \alpha_k \phi_n(0,k) \\ &= R_n(0) - \sum_{k=1}^p \alpha_k R_n(k) \end{aligned} \quad (2.29)$$

และจากสมการ (2.29) สามารถหาค่า G โดยตรงจาก

$$G^2 = \frac{R_n(0) - \sum_{k=1}^p \alpha_k R_n(k)}{\sum_{m=0}^{N-1} u^2(m)} \quad (2.30)$$

การเปลี่ยนค่าพารามิเตอร์ LPC เป็นสัมประสิทธิ์เซปสตรัม (Cepstrum)

ในการรู้และจำเสียงพูดนั้น สัมประสิทธิ์เซปสตรัมนี้เป็นพารามิเตอร์ที่มีลักษณะน่าเชื่อถือได้ดีกว่าสัมประสิทธิ์ LPC ทั้งยังมีความสัมพันธ์ใกล้ชิดกับการรู้และจำเสียงพูดตามความรู้สึกของมนุษย์โดยแท้จริง สัมประสิทธิ์เซปสตรัมสามารถหาได้โดยตรงจากสัมประสิทธิ์ LPC ดังนี้

$$C_0 = \ln G \quad (2.31)$$

$$C_m = a_m + \sum_{k=1}^{m-1} \left(\frac{k}{m} \right) C_k a_{m-k}, \quad 1 \leq m \leq p \quad (2.32)$$

$$C_m = \sum_{k=1}^{m-1} \left(\frac{k}{m} \right) C_k a_{m-k}, \quad m \geq p \quad (2.33)$$

$$Q \approx \frac{3}{2} p \quad (2.34)$$

การถ่วงน้ำหนักค่าพารามิเตอร์ (Parameter Weighting)

เนื่องจากสัมประสิทธิ์เซปสตรัมที่ได้นั้นช่วงลำดับต้น ๆ และลำดับท้าย ๆ ของเฟรมที่นำมาวิเคราะห์จะเกิดความคลาดเคลื่อนมากกว่าบริเวณส่วนอื่น เพราะฉะนั้นจึงทำการถ่วงน้ำหนักเพื่อลดความคลาดเคลื่อนดังกล่าวนี้ด้วยฟังก์ชันเวทดัง ดังนี้คือ

$$W_m = \left[1 + \frac{Q}{2} \sin \left(\frac{\pi m}{Q} \right) \right], \quad 1 \leq m \leq Q \quad (2.35)$$

จะได้ $C' = C_m * W_m \quad (2.36)$

2.3.1.3 การควอนไทซ์(Quantization)

การจดจำเสียงพูด เป็นลักษณะหนึ่งของการจดจำรูปแบบ (Pattern Recognition) คือ จะเป็นการเปรียบเทียบระหว่างแบบทดสอบ (Test Pattern) กับแบบอ้างอิง (Reference Pattern) ซึ่งเป็นรูปแบบที่ทราบและเก็บไว้ล่วงหน้า

ขั้นตอนในการจดจำ แบ่งเป็น 2 ขั้นตอน ดังนี้

ก.ขั้นเรียนรู้(Learning) จะเป็นการสร้างกลุ่มของแบบอ้างอิงในการจดจำเสียงพูด ในขั้นนี้จะทำการวิเคราะห์เสียงพูดก่อน แล้วเก็บลักษณะของเสียงในรูปของพารามิเตอร์ และใช้การควอนไทซ์แบบเวกเตอร์ในการสร้างแบบอ้างอิง เพื่อใช้เปรียบเทียบในขั้นตอนต่อไป

ข.ขั้นจดจำ(Recognition) จะเป็นการทดสอบการจดจำ ระหว่างแบบอ้างอิงกับแบบทดสอบ โดยจะทำการเปรียบเทียบพารามิเตอร์ของแบบทดสอบกับแบบอ้างอิงทั้งหมด แบบอ้างอิงที่เลือก คือ แบบอ้างอิงที่มีพารามิเตอร์ใกล้เคียงกับแบบทดสอบมากที่สุด

การควอนไทซ์แบบเวกเตอร์ (Vector Quantization)

ในทฤษฎีของการสื่อสารนั้น แหล่งเข้ารหัสจะให้เทคนิคการเปลี่ยนค่าสัญญาณเอาต์พุตไปเป็นรหัสแบบไบนารี ซึ่งเป้าหมายของเครื่องเข้ารหัสคือ การเป็นไปได้ที่จะมีการผิดเพี้ยนของข้อมูลน้อยที่สุดด้วยอัตราความเร็ว (Bit rate) ที่กำหนด

การควอนไทซ์แบบเวกเตอร์เป็นวิธีการลดมิติ (Dimension) หรือจำนวนของข้อมูลลง เวกเตอร์อินพุตหรือเซตเทรนนิ่ง หรือพารามิเตอร์ที่ได้จากขั้นตอน LPC จะถูกเลือกมากลุ่มหนึ่งซึ่งใช้เป็นตัวแทนของข้อมูลจำนวนหนึ่งหรือเรียกว่าการหาไดคบุค(Codebook)อินพุตที่เข้ามาจะถูกทำการเปรียบเทียบกับ ไดคบุคที่มีอยู่โดยจะพิจารณาว่าอินพุตที่เข้ามาห่างจากไดคบุคใดน้อยที่สุด ซึ่งอินพุตดังกล่าวจะถูกแทนด้วยเวกเตอร์ไคค(index) นั้น อินพุตทุกตัวที่เป็นสมาชิกของเวกเตอร์ไคคใด ๆ จะถูกนำมาหาจุดศูนย์กลางร่วมใหม่ และนำจุดศูนย์กลางที่ได้นี้ไปทำการหาความคลาดเคลื่อนกับสมาชิกทุกตัว ถ้าค่าความคลาดเคลื่อนที่ได้มีค่ามากกว่าค่าที่กำหนดไว้ค่าหนึ่งหรือค่าที่ยอมรับได้ ก็จะนำจุดศูนย์กลางใหม่นั้นไปเป็นไดคบุคแทน และจะทำการจัดกลุ่มอินพุตเข้ากับ ไดคบุคใหม่ที่ได้และหาความคลาดเคลื่อนอีกครั้งทำอย่างนี้ซ้ำ ๆ จนกระทั่งค่าความคลาดเคลื่อนมีค่าน้อยถึงค่าที่ยอมรับได้ ก็จะถือว่าไดคบุคที่ดีที่สุดที่จะเป็นตัวแทนของอินพุตทั้งหมด จะสังเกตได้ว่าทุกครั้งที่มีการหาไดคบุคใหม่นั้น ค่าความคลาดเคลื่อนที่ได้จะมีค่าลดลงทุกครั้งด้วย

ถ้าเปรียบเทียบอัตราข้อมูลของการแทนเวกเตอร์สำหรับเสียงที่ยังไม่ได้เข้ารหัสจะเห็นว่า การวิเคราะห์เสียงนั้นต้องการการลดลงของอัตราข้อมูลพิจารณาดังนี้ เสียงพูดที่ซิกด้วยอย่าง (sampling) ด้วยความถี่ 10 kHz และมีขนาด 16 บิต จะต้องการอัตราข้อมูล 160,000 bps ในการจัดเก็บ ซึ่งเป็นรูปแบบที่ยังไม่ได้มีการบีบอัดข้อมูล สำหรับการวิเคราะห์เสียงพูดพิจารณาเวกเตอร์ที่มี 10 มิติ ใช้ 100 สเปกตรัมต่อวินาที ถ้าใช้ 16 บิตแทนแต่ละเวกเตอร์ อัตราที่ใช้ในการเก็บ คือ $100 \times 10 \times 16$ bps หรือ 16,000bps สามารถลดลงไปได้ถึง 10 เท่าของสัญญาณที่ยังไม่ได้บีบอัด ยิ่งไปกว่านั้นวิธีการควอนไทซ์แบบเวกเตอร์ยังสามารถลดอัตราข้อมูลลงไปได้อีก โดยการสร้างตารางรหัส เพื่อใช้ในการชี้ว่าเวกเตอร์

ตารางรหัสโหนดที่เหมือนกับเวกเตอร์เสียงอินพุตที่เข้ามามากที่สุด สมมติตารางรหัสแทนด้วยตัวเลขจำนวน 10 บิต และมีอัตราข้อมูล 100 สเปกตรัมต่อวินาทีจะได้ว่าอัตราข้อมูล 1,000bps ซึ่งลดอัตราข้อมูลลงไปถึง 16 เท่า

เวกเตอร์สเปกตรัมที่จะนำไปทำการควอนไทซ์ เป็นเวกเตอร์ที่ได้จากการประมาณเชิงเส้นและผ่านการปรับปรุงให้ดีขึ้นโดยการทำ เชปสตรัมและการเวทค่าพารามิเตอร์ ซึ่งจะแสดงโดย $V, l = 1, 2, \dots, L$ ซึ่งแต่ละเวกเตอร์เป็นเวกเตอร์ที่มี p มิติ (p คือ order หลังจากผ่านการเวทค่าเรียบร้อยแล้ว) เช่นในรูป ตัวอย่างที่ 2.14 และ 2.15 คือ $p=2$ (order=2) ของ LPC Model

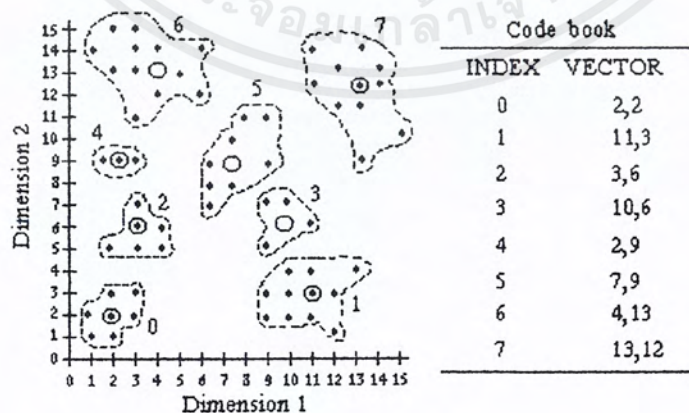
ข้อดีของวิธีการการควอนไทซ์แบบเวกเตอร์ คือ

- ลดความจุในการเก็บข้อมูลสเปกตรัมที่ได้จากการวิเคราะห์หลัง
- ลดการคำนวณในการเปรียบเทียบความเหมือนของเวกเตอร์สเปกตรัมที่ได้จากการวิเคราะห์หลัง ซึ่งเป็นการคำนวณหลักที่ใช้ในการจำเสียงพูด

Training data

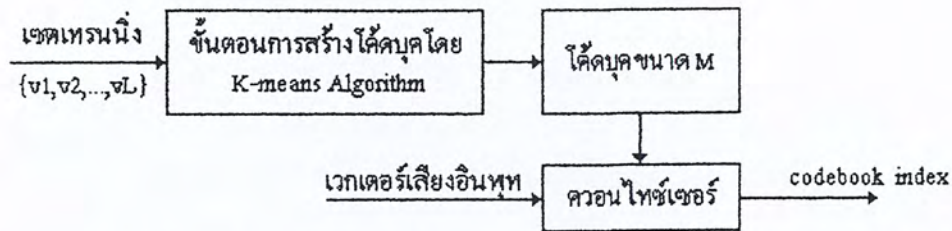
1,1	3,5	6,7	9,9	12,1
1,2	3,6	6,8	9,11	12,3
1,9	3,7	6,9	10,2	12,11
1,14	3,9	6,12	10,3	12,13
2,1	3,11	6,14	10,4	13,4
2,2	3,13	7,8	10,7	13,9
2,3	3,14	7,10	11,2	13,11
2,5	3,14	8,11	11,3	13,12
2,9	4,5	9,2	11,4	13,14
2,13	4,6	9,3	11,6	14,12
2,15	4,12	9,5	11,12	14,13
3,2	4,14	9,7	11,14	15,10
3,3	5,13			

รูปที่ 2.14 ตัวอย่างข้อมูลที่จะทำการควอนไทซ์



รูปที่ 2.15 แสดงการจับกลุ่มและดัชนีของตารางรหัส (code book) ที่ได้

ส่วนประกอบของการควอนไทซ์แบบเวกเตอร์



รูปที่ 2.16 แสดงบล็อกไดอะแกรมของการควอนไทซ์แบบเวกเตอร์

1. การทำงานของการควอนไทซ์แบบเวกเตอร์

1.1 เซตของเวกเตอร์สเปกตรัมที่ได้จากการวิเคราะห์ $v_1, v_2, v_3, \dots, v_L$ ซึ่งเป็นเซตขนาดใหญ่ จะถูกนำมาใช้ในการสร้างเซตของตารางรหัส เพื่อใช้เป็นตัวแทนเวกเตอร์สเปกตรัมของเสียงที่สังเกตได้ ขนาดตารางรหัสของการควอนไทซ์แบบเวกเตอร์คือ $M = 2^B$ เวกเตอร์ (เรียกว่า B bit ตารางรหัส) ซึ่งต้องการให้ $L \gg M$ เพื่อที่จะหาเซตของ M ตารางรหัสที่ดีที่สุด ในทางปฏิบัตินั้นเพื่อให้ตารางรหัสของการควอนไทซ์แบบเวกเตอร์สามารถทำงานได้ดีนั้น ขนาด L อย่างน้อยที่สุดที่สามารถใช้ได้คือ $10M$

1.2 การวัดความเหมือน หรือระยะทางระหว่างเวกเตอร์สเปกตรัมที่ได้จากการวิเคราะห์ 2 เวกเตอร์ ดังนั้นสามารถจัดกลุ่มเซตของเวกเตอร์เทรนนิ่ง เพื่อสร้างตารางรหัส แสดงระยะทางสเปกตรัม $d(v_i, v_j)$ ระหว่าง 2 เวกเตอร์ v_i และ v_j คือ d_{ij}

1.3 ฟังก์ชันย่อยในการคำนวณหาจุดศูนย์กลาง เพื่อแบ่งแยกเซตของ เทรนนิ่ง จำนวน L เวกเตอร์ ไปเป็น M กลุ่ม และ M ตารางรหัส ที่ได้คือจุดศูนย์กลางของแต่ละกลุ่ม

1.4 ฟังก์ชันย่อยในการแบ่งแยกสำหรับเวกเตอร์สเปกตรัมที่ได้จากการวิเคราะห์ ซึ่งจะเลือกตารางรหัสที่ใกล้เคียงที่สุดให้กับเวกเตอร์อินพุต และผลลัพธ์ที่ได้คือ วรรณิตารายรหัสซึ่งเป็นตัวแทนของสเปกตรัม มักจะใช้กฎ Nearest Neighbour ในการพิจารณา ส่วนนี้เป็นหัวใจหลักของตัวควอนไทซ์ ซึ่งอินพุต คือ เวกเตอร์สเปกตรัมของเสียง และเอาต์พุตคือวรรณิตารายรหัส ที่มีความเหมือนกับเวกเตอร์เสียงอินพุตมากที่สุด

2. เซตเทรนนิ่ง (Training) ของการควอนไทซ์แบบเวกเตอร์

เซตเทรนนิ่งของการควอนไทซ์แบบเวกเตอร์ซึ่งเป็นเสียงพูดที่จะนำมาใช้ในการสร้างตารางรหัส ซึ่งเซตเทรนนิ่งจะต้องมีขนาดใหญ่มาก ๆ ถึงจะสามารถสร้างตารางรหัสที่ดีได้ เซตเทรนนิ่งจะแปรตามสถานะต่าง ๆ ดังนี้

- ผู้พูด รวมทั้งช่วงของอายุ เพศ ความเร็ว-ช้าในการพูด ระดับเสียง ฯลฯ
- สถานะแวดล้อมขณะพูด เช่น ในห้องเงียบ หรือในรถ
- ระบบการแปลงส่ง รวมทั้ง ไมโครโฟน หูโทรศัพท์
- หน่วยของเสียงรวมถึง คำศัพท์ต่าง ๆ และข้อความที่สนทนา

หลักการของการควอนไทซ์แบบเวกเตอร์

การควอนไทซ์แบบเวกเตอร์ เป็นวิธีการควอนไทซ์คลื่นเสียง ในการควอนไทซ์แบบเวกเตอร์มีวิธีการควอนไทซ์ดังนี้ คือ จะมีเวกเตอร์รหัส หรือตารางรหัสเก็บไว้ อินพุตที่เข้ามาจะถูกทำการเปรียบเทียบกับตารางรหัสที่มีอยู่ โดยพิจารณาว่าอินพุตที่เข้ามานั้นห่างจากตารางรหัสใดน้อยที่สุด อินพุตดังกล่าวจะถูกแทนด้วยเวกเตอร์รหัสนั้น

ตารางรหัสที่ใช้จะเหมาะสมเพียงใด ขึ้นอยู่กับความคลาดเคลื่อนรวมทั้งหมดของคั่นแบบที่ใช้หา เมื่อคั่นแบบที่พิจารณามีจำนวนจำกัด ตารางรหัสที่มีความคลาดเคลื่อนน้อยที่สุด จะได้จากการทำซ้ำจนกว่าจะได้ค่าที่เหมาะสม ด้วยเหตุนี้จึงขึ้นกับค่าเรนดอมที่สุ่มขึ้นมาจำนวนเท่ากับขนาดของตารางรหัสเพื่อใช้เป็นค่าเริ่มต้น ในวิธีการนี้ข้อมูลคั่นแบบที่ต้องการหาถูกแยกไปอยู่ในแต่ละกลุ่มของค่าเริ่มต้น จะทำการหาค่าเฉลี่ยของแต่ละกลุ่มเพื่อเป็นค่ากลางของกลุ่ม ทำวิธีดังกล่าวซ้ำจนกว่าความคลาดเคลื่อนรวมจะต่ำกว่าค่า ๆ หนึ่ง หรือการลดลงของความคลาดเคลื่อนรวมน้อยกว่าค่า ๆ หนึ่ง โดยค่ากลางดังกล่าวของกลุ่มจะถูกเก็บเป็นเวกเตอร์รหัส แต่แน่นอนว่าค่าความคลาดเคลื่อนรวมจะลดลงทุกครั้งที่มีการคำนวณซ้ำใหม่ จึงขึ้นกับค่าที่กำหนดว่าต้องการให้ความคลาดเคลื่อนรวมน้อยเพียงใด

การคำนวณการควอนไทซ์แบบเวกเตอร์

ถ้าสมมติ $x = [x_1, x_2, \dots, x_p]$ มี p มิติ จะทำการควอนไทซ์ของเวกเตอร์ x กับตารางรหัสสามารถเขียนว่า y เป็นควอนไทซ์ของค่า x

$$y = q(x) \quad (2.37)$$

โดย $q(\cdot)$ เป็นโอเปอเรเตอร์ของควอนไทซ์ y ถูกเรียกว่าเอาต์พุตของเวกเตอร์ของค่า x โดย y เป็นค่าใดค่าหนึ่งใน $Y = \{y_i, 1 \leq i \leq M\}$ โดย $y_i = [y_{i1}, y_{i2}, \dots, y_{ip}]$ Y เป็นเซตของตารางรหัส M เป็นขนาดของตารางรหัส และ $\{y_i\}$ เป็นเซตของเวกเตอร์รหัส y_i อาจเรียกว่าเป็นรหัสอ้างอิง และ M อาจเรียกว่าจำนวนระดับขั้นจะทำการแบ่งเวกเตอร์ x ไปใน $M \{C_i, 1 \leq i \leq M\}$ เมื่อ x อยู่ในเซต C_i

$$q(x) = y_i \text{ ถ้า } x \in C_i$$

ถ้า x ถูกควอนไทซ์ได้ค่า y ค่าความคลาดเคลื่อนจากการควอนไทซ์สามารถแสดงได้โดยระยะห่างของทั้งสอง $d(x, y)$ โดยความคลาดเคลื่อนรวมคือ

$$D = \lim_{M \rightarrow \infty} \frac{1}{M} \sum_{n=1}^M d[x(n), y(n)] \quad (2.38)$$

การคำนวณหาตารางรหัสมีหลายวิธี แต่ที่จะนำมาใช้คือ Lloyd Algorithm K-means Algorithm)

การวัดความคลาดเคลื่อน

การวัดความคลาดเคลื่อนเป็นส่วนจำเป็นและเป็นประโยชน์ต่อการออกแบบ ตารางรหัสซึ่งวัดโดยการวัดระยะทาง ซึ่งสมการพีชคณิตที่ใช้ในการหาระยะทางในปัจจุบันมีวิธีการหลายแบบ แต่ที่จะนำมาใช้ในปริภูมยานุพันธ์นี้ คือ square error distortion

square error distortion : ถ้าสัญญาณที่เข้ามามี P มิติ สามารถหาระยะห่างระหว่างสัญญาณเข้ากับเวกเตอร์รหัส

$$d(v_1, v_2) = \|v_1 - v_2\|^2 = \sum_{i=0}^{k-1} (x_i - y_i)^2 \quad (2.39)$$

ซึ่งเป็นวิธีการคำนวณที่ง่าย และรวดเร็วในการวัดความคลาดเคลื่อน จึงนิยมใช้วิธีนี้ แต่ก็ยังไม่เหมาะสมเพียงพอ ควรใช้ Itakura distance มากกว่า แต่เป็นวิธีที่มีความซับซ้อนมาก และยุ่งยากในการคำนวณ

การจัดกลุ่มเพื่อสร้างแบบอ้างอิงโดยใช้ K-means Algorithm

การสร้างกลุ่มของแบบอ้างอิงเป็นขั้นตอนในส่วนของการเรียนรู้ โดยเริ่มต้นจากการนำข้อมูลสัญญาณเสียงที่ได้จากส่วนของกาวิเคราะห์ มาทำการสร้างเป็นกลุ่มของแบบอ้างอิง ในระบบการจดจำเสียงพูดแบบต่างบุคคล จะใช้แบบอ้างอิงของคำหนึ่ง ๆ จากผู้พูดจำนวนมาก เพื่อที่จะได้ครอบคลุมถึงความแปรปรวนต่าง ๆ ที่จะเกิดขึ้นระหว่างผู้พูดแต่ละคน แต่อย่างไรก็ตาม ผลที่เกิดขึ้นตามมาเมื่อจำเป็นจะต้องมีข้อมูลจำนวนมากในกลุ่มของแบบอ้างอิง คือ

1. เวลาที่ใช้ในการตอบสนอง ในการจดจำเสียงพูด ขั้นตอนการเปรียบเทียบแบบทดสอบกับแบบอ้างอิงทั้งหมดที่มีเป็นขั้นตอนที่สำคัญในการจดจำ เมื่อแบบอ้างอิงมีจำนวนมาก การเปรียบเทียบจึงจำเป็นต้องอาศัยเวลาเพิ่มขึ้นด้วย
2. เนื้อที่ในหน่วยความจำสำรองที่ใช้ในการเก็บแบบอ้างอิง เมื่อมีแบบอ้างอิงจำนวนมากที่จำเป็นต้องใช้ในการเปรียบเทียบ เนื้อที่ในหน่วยความจำสำรองก็จำเป็นต้องมีเพิ่มขึ้นด้วย
3. ความถูกต้องในการจดจำ เนื่องจากจำนวนแบบอ้างอิงในแต่ละคำไม่ได้เป็นสัดส่วนโดยตรงกับความถูกต้องในการจดจำ กล่าวคือเมื่อเพิ่มจำนวนแบบอ้างอิงในแต่ละคำไปจนถึงระดับหนึ่ง ความถูกต้องในการจดจำจะเริ่มคงที่ และในบางครั้งอาจจะมีค่าลดลงด้วย

จากเหตุผลดังกล่าว จึงทำให้มีความพยายามที่จะจัดกลุ่มของแบบอ้างอิงในแต่ละคำใหม่ เพื่อให้ได้แบบอ้างอิงที่พหุเหมาะสม และสามารถให้เป็นตัวแทนของแบบอ้างอิงที่มีอยู่ทั้งหมดได้ อัลกอริทึมที่ใช้ในการแบ่งกลุ่มมีอยู่เป็นจำนวนมาก สำหรับอัลกอริทึมที่จะกล่าวถึง คือ K-means Algorithm ตามขั้นตอนโดยสรุป ดังนี้

- การหาจุดศูนย์กลางของกลุ่ม (Center Cluster)

ในขั้นตอนนี้จะทำการกำหนดจำนวนกลุ่มของแบบอ้างอิงให้มีค่าเท่ากับ M และเริ่มต้นจากการสุ่ม (random) ค่าจำนวนเท่ากับขนาดของตารางรหัส เพื่อเป็นค่าจุดศูนย์กลางของแต่ละกลุ่มก่อน จากนั้นจะแบ่งเวกเตอร์ต่าง ๆ เข้าไปตามกลุ่มต่าง ๆ โดยพิจารณาจากระยะทาง หรือความแตกต่างกันน้อยที่สุดกับกลุ่มไหน ก็จะให้อยู่กลุ่มนั้น จากนั้นก็จะทำการเฉลี่ยค่าของเวกเตอร์ที่อยู่ในกลุ่มเพื่อให้เป็นจุดศูนย์กลางใหม่

- การจัดกลุ่มใหม่ (Reclassification)

แบบอ้างอิงที่เหลืออยู่ในแต่ละคำ จะถูกจัดให้เข้าไปในแต่ละกลุ่ม โดยมีเงื่อนไขคือ ระยะทางของแบบอ้างอิงนี้กับแบบอ้างอิงในกลุ่มจะมีค่าเฉลี่ยน้อยที่สุด หลังจากที่แบบอ้างอิงทุกแบบถูกจัดเข้ากลุ่มเรียบร้อยแล้ว จะทำการหาจุดศูนย์กลางของแต่ละกลุ่มใหม่ ขั้นตอนนี้จะทำซ้ำจนกว่าจุดศูนย์กลางของแต่ละกลุ่มจะไม่มีเปลี่ยนแปลง อย่างไรก็ตาม จากขั้นตอนดังกล่าวจะเห็นว่า การกำหนดจำนวนกลุ่มที่เหมาะสมจะทำได้ลำบาก ดังนั้นจึงได้มีการปรับปรุงเทคนิคนี้เป็นการกำหนดค่าระยะทางสูงสุดที่จะทำให้เกิดกลุ่มใหม่ขึ้นมา คือเมื่อแบบอ้างอิงใดที่ถูกเลือกให้เข้ากลุ่มด้วยระยะทางที่น้อยที่สุดแล้ว แต่ระยะทางนี้ยังคงมีค่ามากกว่าระดับที่กำหนด แบบอ้างอิงนี้จะถูกนำไปสร้างเป็นจุดศูนย์กลางของกลุ่มใหม่ทันที

การคำนวณจุดศูนย์กลางสำหรับการออกแบบตารางรหัสการควอนไทซ์แบบเวกเตอร์
การสร้างตารางรหัส โดยใช้ K-means สามารถสรุปสั้น ๆ ได้ดังนี้

1. ความเพี้ยนน้อยสุด สำหรับแต่ละเวกเตอร์ อินพุต x_i เมื่อเปรียบเทียบกับเวกเตอร์ y_i ซึ่งเป็นตารางรหัส C ที่เข้าไป กลุ่มของเวกเตอร์เทรนนิ่งจะเป็นไปตาม Code words

2. การคำนวณหาจุดศูนย์กลาง แต่ละกลุ่มของเวกเตอร์จะคำนวณหาจุดศูนย์กลางใหม่ซึ่งมีความผิดพลาดน้อยที่สุด สำหรับสมาชิกกลุ่มที่ถูกแทนโดยเวกเตอร์จุดศูนย์กลาง

ทำ 2 ขั้นตอนซ้ำจนกว่าจะเกิดการลู่เข้า (convergent) ขั้นแรกนั้นเป็นการทำโดยตรง คือนำเข้าไปเพื่อหาความผิดพลาด ส่วนขั้นที่ 2 นั้นเป็นการหาจุดศูนย์กลาง ซึ่งเป็นขั้นตอนที่สำคัญ

พิจารณา $\{x_i\}_{i=1}^L$ และค่าความผิดพลาด $d(x,y)$ โดยทั่วไป สมมติว่าเวกเตอร์เหล่านี้ถูกกำหนดไว้ในกลุ่มป้ายเดียวกัน (หรือ Code word) จุดศูนย์กลางของ $\{x_i\}_{i=1}^L$ ถูกจำกัดความเป็นเหมือนเวกเตอร์ $\bar{Y}$ ซึ่งเป็นค่าความเพี้ยนเฉลี่ยน้อยที่สุด

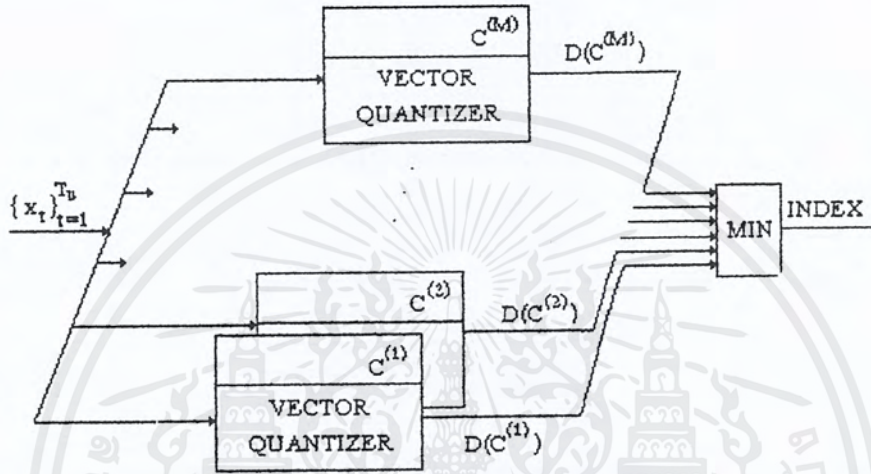
$$\bar{Y} = \Delta \arg \min \frac{1}{L} \sum_{i=1}^L d(x_i, y_i) \quad (2.40)$$

เมื่อ x_i และ y เป็นเวกเตอร์ โดยที่ $x_i = (x_{i1}, x_{i2}, x_{i3}, \dots, x_{ik})$ และ $y = (y_1, y_2, \dots, y_k)$ วัดในพื้นที่ k มิติกับระยะทางตามแบบยูคลิด จุดศูนย์กลางคือค่าเฉลี่ยของเซตของเทรนนิ่ง

$$\bar{Y} = \frac{1}{L} \sum_{i=1}^L x_i \quad (2.41)$$

$\bar{Y}$ เป็นจุดศูนย์กลางซึ่งเป็นเวกเตอร์ที่อยู่ตรงกลางของ $\{x_i\}_{i=1}^L$ ซึ่งแต่ละมิติจะไม่ขึ้น
แก่กัน หมายความว่า แต่ละ y_k เป็นค่ากลางของ $\{x_i\}_{i=1}^L$

การควอนไทซ์แบบเวกเตอร์สำหรับการจดจำเสียงพูด



รูปที่ 2.17 แสดงเวกเตอร์ควอนไทซ์ (vector quantizer) ที่ใช้ในระบบการจดจำเสียงพูด

การที่จะสร้างตารางรหัสจำเป็นต้องใช้เวกเตอร์ทดสอบเป็นจำนวนมาก ซึ่งจะจัดเวกเตอร์ออกเป็นกลุ่ม ๆ และแต่ละกลุ่ม i แทนจุดศูนย์กลางด้วย y_i ซึ่งหาจุดศูนย์กลางได้จากการเฉลี่ยค่าเวกเตอร์ทั้งหมดในกลุ่ม ซึ่ง y_i จะเป็นเวกเตอร์รหัสที่เก็บไว้ หรือเป็นตารางรหัสเก็บไว้สำหรับการควอนไทซ์เวกเตอร์เสียงต่อไป เวกเตอร์เสียงอินพุตของตัวควอนไทซ์ $= x_i$ จะถูกเปรียบเทียบระยะทางกับ $d(x_i, y_i)$ ค่า y_m ที่เลือกคือระยะทาง d น้อยที่สุด ซึ่ง y_m จะแทน x_i ที่เข้ามา จากนั้นจะส่งครรชน m ไปยังเครื่องรับ จากเมตริกซ์ระยะทาง ได้ว่า ถ้า x ใกล้ y_1 มากกว่า y_2 สรุปได้ว่า x เสียงเหมือน y_1 มากกว่า y_2

การควอนไทซ์แบบเวกเตอร์ที่ใช้เพื่อการออกแบบการจดจำเสียงพูดนั้น มีจำนวนตัวควอนไทซ์ M ตัว นั้นหมายถึง มี M ระดับของเสียงเพื่อการจดจำ แต่ละระดับเสียงสามารถพิจารณาจากเซตของข้อมูล เทรนนิ่ง $\{x_i^{(i)}\}$ เมื่อ $i = 1, 2, 3, \dots, M$ ซึ่ง M เป็นครรชนระดับ แต่ละเซตของ เทรนนิ่ง ในแต่ละระดับจะเก็บเสียงที่อยู่ในระดับเดียวกัน เมื่อมีเสียงที่ไม่รู้ $\{x_i\}_{i=1}^{T_v}$ เข้ามา จะเป็นอินพุตเข้าไปยังทุก ๆ ตัวควอนไทซ์ ค่าครรชนระดับ (index) ที่เลือกจะเป็นระดับที่มีความผิดพลาดน้อยที่สุด $D(c^{(i)})$ เมื่อ $i = 1, 2, \dots, M$ ซึ่งความผิดพลาดนี้ได้อาจการวัดระยะทางกลุ่มที่ i ในจำนวน j กลุ่ม ซึ่งจะทำให้การวนซ้ำ k ครั้งแสดงโดย $\{w_j^{(k)}\}$ ซึ่ง $i = 1, 2, \dots, j$ และ $k = 0, 1, 2, \dots, k_{\max}$ ($k_{\max}$ คือ จำนวนการนับวนซ้ำที่มากที่สุด) $j = 1$ (กลุ่มเดียว) จนถึง $j_{\max}$ (จำนวนกลุ่มที่มากที่สุด) โดยมีรายละเอียดดังนี้

1. เริ่มโดย $j = 1, i = 1$ คำนวณเมตริกซ์ระยะทาง $D = \delta(x_i, x_m), 1 \leq i, m \leq N$

2. คำนวณจุดศูนย์กลางกลุ่ม คือ $c(\Omega)$ ของ Ω จุดศูนย์กลางกลุ่มของข้อมูล เทานั้น อาจถูกคำนวณได้โดยแบบจุดศูนย์กลางที่สุด หรือศูนย์กลางเฉลี่ย

$$3. \text{ ให้ } (w_i^1)^{(0)} = \Omega, c[(w_i^1)^{(0)}] = c(\Omega)$$

4. แบ่งแยกแต่ละ pattern x_i ของ Ω ซึ่งเป็นของกลุ่ม 1 กลุ่ม $(\Omega_i^j)^{(k)}, i = 1, 2, \dots, j$ โดยเลือกกลุ่มซึ่งมีระยะทางน้อยที่สุดจาก pattern x_i ไปยังจุดศูนย์กลางของกลุ่ม $c[(w_i^j)^{(k)}]$ เช่น

$$x_i \in (w_j^i)^{(k)} \text{ iff } \delta(x_i, c[(w_j^i)^{(k)}]) \leq \delta(x_i, c[(w_j^i)^{(k)}]) \text{ สำหรับทุก } i \neq j$$

5. สำหรับแต่ละกลุ่มที่ได้ $(w_i^j)^{(k)}, i = 1, 2, \dots, j$ คำนวณหาจุดศูนย์กลางกลุ่ม $c[(w_i^j)^{(k)}]$ อีกครั้ง โดยใช้จุดศูนย์กลางที่สุด หรือจุดศูนย์กลางเฉลี่ย

6. ตรวจสอบการบรรจบกัน โดยดูจากมีบางกลุ่มถูกเปลี่ยนไปจากการทำซ้ำครั้งก่อนหรือไม่ (เช่น $(w_i^j)^{(k)} = (w_i^j)^{(k-1)}, i = 1, 2, \dots, j$) ถ้ามีบางกลุ่มเปลี่ยนไป ตัวนับ k จะเพิ่มค่าขึ้น และตรวจสอบกับค่าที่มากที่สุดที่กำหนด ถ้าค่าการวนซ้ำยังไม่ถึงค่ามากที่สุด K-means จะทำการวนซ้ำอีก (ขั้นตอนที่ 4-5 จะถูกทำซ้ำ)

7. คำนวณหา $\bar{D}_i$ ซึ่งเป็นระยะทางภายในกลุ่มเฉลี่ย ทุก ๆ กลุ่ม $| (w_i^j)^{(k)} |$ เป็นจำนวน pattern ของ Ω ที่รวมอยู่ในกลุ่ม $(w_i^j)^{(k)}$ ที่จุดสิ้นสุดของการทำซ้ำของ K-means

8. เมื่อการบรรจบกันได้รับ (ทุกกลุ่มไม่เปลี่ยนไปจากการทำซ้ำในครั้งก่อน หรือได้รับค่าสูงสุดของการทำซ้ำแล้ว) กลุ่ม $(w_i^j)^{(k)}$ ถูกเก็บไว้เป็น j กลุ่มที่ดีที่สุด j จะเพิ่มขึ้นและตรวจสอบขนาดกลุ่ม ถ้าถึง $j_{\max}$ แล้ว กระบวนการจะสิ้นสุด และศูนย์กลางกลุ่มสุดท้ายจะถูกสร้างขึ้นสำหรับทุกกลุ่ม ($j = 1, 2, \dots, j_{\max}, i = 1, 2, \dots, j$)

9. ถ้าขั้นตอนอื่นของ K-means ถูกทำงานนั้น อันดับแรกต้องหา (λ_{j-1}^i) ซึ่งมีระยะทางภายในกลุ่มมากที่สุด

$$\bar{D}_i' \geq \bar{D}_i \quad i \neq i'$$

กลุ่มที่มีขนาดใหญ่ที่สุดนี้ ถูกแบ่งแยกไปเป็น 2 กลุ่ม โดยการหารรชน l_1, l_2 ของทั้งสองแบบใน (λ_{j-1}^i) ดังนั้นระยะทางระหว่างแบบเหล่านี้จะมากที่สุด

$$\delta(x_{l_1}, x_{l_2}) \geq \delta(x_{l_3}, x_{l_4}) \quad \text{สำหรับทุก } l_3, l_4 \in (\lambda_{j-1}^i)$$

ศูนย์กลางกลุ่มเริ่มต้นใหม่ สำหรับกลุ่มที่ i ถูกตั้งเป็นแบบ x_{l_1} และศูนย์กลางเริ่มต้นใหม่สำหรับกลุ่มที่ j (เช่นอันใหม่ที่ถูกรวมเข้าไปที่การทำซ้ำนี้) เป็นแบบ x_{l_2} ทุก ๆ ศูนย์กลางกลุ่มเริ่มต้นอื่น ๆ $c[(w_i^j)^{(k)}]$ ถูกเลือกเป็นศูนย์กลางกลุ่ม $c[(w_i^j)^{(k)}]$ ที่จุดสิ้นสุดของการวนซ้ำของ K-means ในครั้งก่อนนั้น ตัวนับการวนซ้ำจะถูกรีเซ็ตไปเป็น $k = 1$

10. ทำการวนซ้ำอีก ขั้นตอนที่ 4-5

Template ชุดท้ายถูกสร้างโดยการใช้จุดศูนย์กลางของการคำนวณกลุ่ม หรือโดยการหาค่าเฉลี่ยทุก pattern ในกลุ่ม ลักษณะเด่นของ K-means Algorithm คือมันจะคำนวณการแบ่งกลุ่มซึ่งครอบคลุม 100 % สำหรับทุกขนาดการแบ่งกลุ่มจาก 1 กลุ่มต่อ 1 คำ จนถึง j กลุ่มต่อ 1 คำ

2.3.2 ปรับรูปแบบเสียง(Pattern Matching)

วิธีในการสร้างและหารูปแบบของคำพูดที่เหมือนมืออยู่สองแนวทางคือ

- Dynamic time warping (DTW)
- Hidden Markov Models (HMM)

ในที่นี้จะเลือกใช้วิธีการไดนามิกไทม์วาร์ปเนื่องจากเหมาะกับการวิเคราะห์คำพูดทั้งแบบต่อเนื่องและไม่ต่อเนื่องได้ดี

ในการรับรู้และจำเสียงพูดที่มีรูปแบบโดยเฉพาะและมีความต่อเนื่องของสัญญาณนั้น จะใช้วิธีในการสังเกตเฟรมของข้อมูล (เฟรมของสัญญาณที่แสดงออกมา หรือเวกเตอร์ของสัญญาณนั่นเอง) ซึ่งอยู่ในรูปของ $X = (X_1, X_2, X_3, \dots, X_n, \dots, X_N)$ กับลำดับที่สองที่เกี่ยวข้องกับ X คือ $Q = (q_1, q_2, q_3, \dots, q_n, \dots, q_N)$ โดย Q จะแสดงถึงลักษณะของภาษาที่คล้ายคลึงกัน ลำดับของ Q ที่ถูกเลือกจะมีค่าความผิดพลาดน้อยที่สุดซึ่งลำดับ Q นี้จะกำหนดลำดับของเฟรมที่ใช้สังเกตเพื่อนำไปใช้ในการอ้างอิง ในคำอื่น ๆ ก็จะเปรียบเทียบกับลำดับ X^{in} เป็นลำดับ X_k^{ref} สำหรับค่า $1 \leq k \leq K$ เมื่อ K คือตัวเลขที่ใช้ในการอ้างอิงลำดับ ในการแยกความแตกต่างเหล่านี้จะใช้วิธีการคำนวณทางเมตริกซ์ และวิธีการหาค่าเวลามาตรฐาน

การแบ่งแยกคำในการจำ

สามารถอธิบายการพิจารณาด้วยตัวอย่างง่าย ๆ ของแบบคำพูดมาตรฐานในการแยกคำจากระบบ สมมติเสียงพูดที่ได้รับนั้นมาจากโทรศัพท์ที่มีความถี่ในการสุ่มสัญญาณ 8 กิโลเฮิร์ตซ์ และมีส่วนของสัญญาณที่ไม่เรียบอยู่ 3.4 เฮิร์ตซ์ สัญญาณเสียงพูดนี้จะถูกเน้นโดยการนำไปผ่านตัวกรองแบบซิงเกิลซีโร่ (single-zero) ที่มีลักษณะความชันสัมพันธ์กับความเรียบของสัญญาณเสียง ทุก ๆ 10 มิลลิวินาทีจะถูกตัดแปลงไปเป็นข้อมูลที่เป็น 25 มิลลิวินาทีของแฮมมิงวินโดว์

สำหรับสัญญาณ 10 มิลลิวินาทีก็จะนำมาทำการแบ่งเป็น 10 ช่วงสัญญาณ เช่นสัญญาณตัวเลขที่มีนั้นมีความเรียบของสัญญาณมีความยาวเฉลี่ย 300 มิลลิวินาที เราก็สามารถประมาณเป็น 30 ช่วงสัญญาณเวกเตอร์ ความถูกต้องของสัญญาณเวกเตอร์ X จะเป็นแผ่นแบบสัญญาณเสียงอย่างในรูป 2.18 ระหว่างชุดของสัญญาณก็สามารถคำนวณและเก็บค่าแบบอ้างอิง X_k^{ref} เมื่อทำการทดสอบก็จะนำมาเปรียบเทียบกับรูปแบบใหม่ของแบบสัญญาณเวกเตอร์ที่นำเข้ามา X^{in} กับทุกค่าอ้างอิงที่มีอยู่ เมื่อได้ค่าความแตกต่างที่น้อยที่สุดก็สามารถอ้างได้ว่าคำนั้นตรงกับคำว่าอะไรในแบบอ้างอิง

สำหรับการคำนวณความแตกต่างระหว่าง 2 ลำดับสัญญาณที่มีรูปแบบเวกเตอร์เฉพาะตัว ก็สามารถใช้วิธีของการหาค่าเวลามาตรฐานเปรียบเทียบ 2 ลำดับสัญญาณที่มีความยาวต่างกัน

C ₁						
C ₂						
C ₃						
C ₄						
C ₅						
C ₆						
C ₇						
C ₈						
C ₉						
C ₁₀						
	x ₁	x ₂	x ₃		x _N	

รูปที่ 2.18 แผ่นแบบของคำพูดที่ประกอบด้วย N(N tenth-order) ของชุดของเวกเตอร์

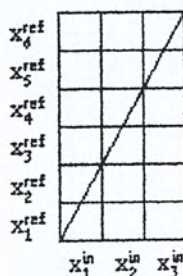
ค่าเวลาที่เป็นเชิงเส้น(Linear Time Warp)

การคาดคะเนค่าการเปรียบเทียบระหว่าง 2 ลำดับสัญญาณที่มีค่าดัชนีของการถ่วงน้ำหนักเวกเตอร์จะบอกถึงความแตกต่างทางเมตริกซ์ระหว่าง 2 เวกเตอร์นั้น ถ้าแบบของเสียงมีความยาวเท่ากันก็สามารถหาความแตกต่างทั้งหมดระหว่าง 2 สัญญาณโดยการรวมค่าความแตกต่างที่แต่ละจุดที่แตกต่างกันระหว่างเฟรมที่สอดคล้องกันของแผ่นแบบเสียงตั้งแต่เริ่มต้นเปรียบเทียบ

$$D(X_k^{ref}, X_i^{in}) = \sum_{l=1}^N d(x_{kl}^{ref}, x_{il}^{in}) \quad 2.42$$

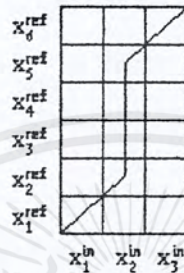
โดยที่ d คือ จุดต่างๆที่มีความแตกต่างกันระหว่างเวกเตอร์ 2 ชุด , D คือ ค่าความแตกต่างรวมระหว่าง 2 แผ่นแบบเสียง , และ N คือ ความยาวของคำพูด (โดยสมมติให้ทั้งคู่มีความยาวเท่ากัน)

โดยทั่วไป คำพูดที่ใช้อ้างอิงกับคำพูดที่เข้ามาจะมีจำนวนเวกเตอร์แตกต่างกัน ดังนั้นจึงไม่สามารถเลือกเปรียบเทียบความแตกต่างกับเฟรมใดเฟรมหนึ่งโดยเฉพาะได้ เช่น มีคำพูดคำว่า “two” 2 ครั้ง ค่าความแตกต่างรวมอาจมีค่าน้อยมากถึงแม้ว่าคำหนึ่งจะมีความยาว 300 มิลลิวินาที และอีกคำหนึ่งมีความยาว 600 มิลลิวินาทีก็ตาม การเปรียบเทียบความคล้ายคลึงกันของสองคำนี้คือการหาค่าเวลามาตรฐานเชิงเส้น เวกเตอร์ของเสียงอ้างอิงหรือเวกเตอร์ของสัญญาณอินพุตเวกเตอร์จะถูกยอมให้มีลำดับที่มีความยาวเหมือนกัน โดยอาจทำการทวนซ้ำสัญญาณเสียง 300 มิลลิวินาที หรือตัดบางส่วนของสัญญาณเสียง 600 มิลลิวินาทีออกไป ดังเช่นตัวอย่างในรูป 2.19 (แสดงเพียงแค่ 10 % เท่านั้น)



รูปที่ 2.19 ค่าเวลามาตรฐานเชิงเส้นของแผ่นแบบสัญญาณอ้างอิงกับแผ่นแบบสัญญาณอินพุต

แต่โดยทั่วไปสัญญาณเสียงพูดมักจะมี ความแตกต่างกัน จังหวะหยุดของคำพูดเพียงเล็กน้อยก็จะมี ความยาว เสียงสระควบ หรือเสียงผสมก็จะมี ความแตกต่างกันมากในการพูดคำเดียวกันในแต่ละครั้ง เช่นการพูดคำว่า “two” เสียงพยัญชนะโดยทั่วไปจะมีความยาวคล้ายกันมากในการพูดที่ 300 มิลลิวินาที และ 600 มิลลิวินาที แต่ในทางตรงกันข้ามเสียงสระมักจะประกอบไปด้วยค่าเวลามาตรฐาน เหตุนี้จึงเกิดความบิดเบือนของสัญญาณ ในช่วงที่หยุดได้น้อยกว่าสัญญาณเสียงสระ โดยทั่วไปวิธีที่ดีที่สุดที่จะนำมาใช้ในการหาค่าความคล้ายคลึงกันสามารถใช้วิธีการที่ไม่เป็นเชิงเส้น หรือวิธีการไดนามิกไทม์วาร์ป ดังแสดงในรูป 2.20



รูปที่ 2.20 ค่าเวลามาตรฐานที่ไม่เป็นเชิงเส้นของแบบสัญญาณอ้างอิงกับสัญญาณอินพุต และสามารถอ้างอิงกับเฟรมทั้งหมดที่ขยายออกไปได้ในช่วง 2-5 เฟรม

ไดนามิกไทม์วาร์ป (Dynamic Time Warp)

ลักษณะโดยทั่วไป ในกรณีที่ผ่านมา ค่าเวลาเชิงเส้นของสัญญาณเสียงอะคูสติคจะไม่มี การชดเชยค่าความแตกต่างที่เหมาะสมในการขยายส่วนประกอบจากสัญญาณเสียงที่มีความเร็วหรือช้า แตกต่างกัน แต่ก็สามารถปรับให้ดีขึ้นได้โดยวิธีทางไดนามิก ซึ่งจะทำการหาค่าโค้งงอที่ดีที่สุดที่อาจเป็นไปได้ในแกนเวลาสำหรับหนึ่งหรือสองลำดับในการเริ่มต้นเปรียบเทียบ

วิธีการแบบไดนามิกก็คือการรวบรวมวิธีการที่เป็นไปได้ทั้งหมดในการแก้ปัญหาที่ทำให้ได้ค่าความผิดพลาดรวมน้อยที่สุด ซึ่งค่าความผิดพลาดรวมจะมีการปรับค่าในแต่ละระดับชั้น โดยในแต่ละระดับนั้นจะมีข้อมูลเกี่ยวกับค่าความเป็นไปได้ที่เพียงพอเพื่อนำมาใช้ในการจำ เมื่อได้ค่าความผิดพลาดรวมที่ดีที่สุด ก็สามารถกำหนดค่านั้นเป็นค่าในการนำไปคัดเลือกค่าอื่น ๆ เพื่อหาค่าที่ตรงกัน ปัญหาของวิธีการนี้ก็คือต้องปรับค่าความแตกต่าง N ค่าที่มีจำนวนค่าต่างกันให้มีขนาด M ค่าเป็น (จำนวนมาตรฐาน) เป็นค่ามากที่สุด ซึ่งก็จะทำให้วิธีการไดนามิกมีประสิทธิภาพในการคำนวณ เช่น กำหนด $M = 10$ ก็คือเมื่อวิธีการแก้ปัญหาที่ $M = 9$ ที่สามารถพิจารณาได้กับปัญหาใหม่บวกเข้าด้วยกัน ในการบวกกันนี้ แต่ละจุดของสัญญาณเสียงก็จะมีค่าที่ได้จากการแก้ปัญหาที่ได้รับเก็บไว้ ค่าของ M แต่ละจุดของสัญญาณเสียงจะถูกนำมาบวกกันแล้วเก็บไว้ ซึ่งจะได้อัตราที่ดีที่สุดเมื่อ $M = 10$

การดัดแปลงแบบของเสียงในการจดจำจะมีวิธีการที่ทำให้ได้ค่าถูกต้องง่าย ๆ โดยการ ใช้เมตริกซ์ D ที่ตรงกันในแต่ละแถวของเฟรมสำหรับอ้างอิง และคอลัมน์ที่ตรงกันของสัญญาณเสียง อินพุต แต่ละอีลิเมนต์ในเมตริกซ์ D สามารถคำนวณหาได้จาก

$$D(i, j) = d(i, j) + \min_{p(i, j)} \{d[p(i, j)] + T[(i, j), p(i, j)]\} \quad 2.43$$

เมื่อ d คือค่าความแตกต่างแต่ละจุดระหว่างเฟรม i ของเสียงอินพุต กับเฟรม j ของเสียงอ้างอิง และ $p(i, j)$ คือค่าความเป็นไปได้ที่ตั้งไว้ก่อนที่จุด i, j ของคำนั้นเพื่อหาความเป็นไปได้ในแต่ละจุดระหว่าง 2 สัญญาณเสียง, ฟังก์ชัน $T(\cdot)$ คือ เทอมที่เกี่ยวข้องกันในการส่งผ่าน แต่ละอิลิเมนต์ของเมตริกซ์ คือค่ารวมของความผิดพลาดที่ดีที่สุดในการคลัมน์สุดท้าย (เฟรมสุดท้ายของสัญญาณเสียงอินพุต) จะสามารถเปรียบเทียบความคล้ายคลึงกันมากที่สุดระหว่างเสียงอ้างอิงกับเสียงอินพุต

ลำดับขั้นตอนในการคำนวณ มีลำดับดังนี้

1. คำนวณหาระยะทางในแต่ละจุดของอิลิเมนต์ในคอลัมน์ที่ 1 ที่ผลิตเขียนไป (ค่าความแตกต่างระหว่างเฟรม 1 ของเสียงอินพุตกับเสียงอ้างอิง) ทำให้ได้ค่าความแตกต่างจากการคำนวณสำหรับแต่ละอิลิเมนต์ของเมตริกซ์

		$D(i-1, j)$	$D(i, j)$	
		$d(i-1, j)$	$d(i, j)$	
	j	$D(i-1, j-1)$	$D(i, j-1)$	
		$d(i-1, j-1)$	$d(i, j-1)$	
				i

รูปที่ 2.21 ตัวอย่างของวิธีการไดนามิก ที่ค่า i, j ต่าง ๆ ที่เกี่ยวข้องกับระยะทาง d และค่าความแตกต่าง D

2. ในเฟรมที่ 2 ของเสียงอินพุต เริ่มต้นที่แถวสุดท้าย คำนวณค่าความแตกต่างและบวกเข้าด้วยกันกับค่าความแตกต่างที่มีอยู่ จะได้ค่าในแต่ละอิลิเมนต์ในคอลัมน์ของสัญญาณเสียงอ้างอิง

3. คำนวณเช่นนี้ในคอลัมน์อื่น

4. เลือกค่าความแตกต่างที่ดีที่สุดที่คอลัมน์สุดท้ายของเสียงอ้างอิง และเลือกค่านั้นสำหรับใช้เป็นค่าเปรียบเทียบในการอ้างอิงครั้งต่อไป

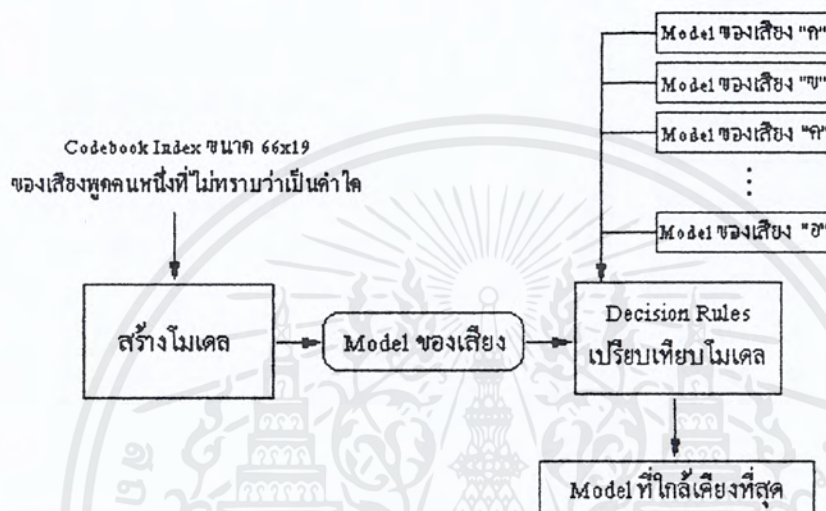
5. เลือกเสียงอ้างอิงที่ใกล้เคียงที่สุดเพื่อทราบว่าเป็นคำพูดของเสียงนั้น

ลำดับขั้นตอนดังกล่าวดัดแปลงมาเป็นวิธีการไดนามิกเพื่อประมาณค่าไทม์แวก แล้วจึงเรียกว่าไดนามิกไทม์แวก หรือ DTW ในลำดับขั้นตอนดังกล่าวจะทำการเปรียบเทียบเสียงอินพุตกับเสียงอ้างอิง ซึ่งในการเปรียบเทียบนั้นจะทำเพียงครั้งเดียวไม่พอ มักจะทำงานกระทั่งได้ค่าอ้างอิงในการเปรียบเทียบที่น้อยที่สุดของแต่ละคำเพื่อนำมาใช้ในการตัดสินใจ

2.3.3 กฎการตัดสินใจ (Decision Rule)

วิทเทอร์บี อัลกอริทึมใช้ในการหาระยะทางที่สั้นที่สุดจากระยะทางที่เป็นไปได้ของโมเดลที่มีอยู่กับโมเดลของเสียงที่เข้ามา โดยเลือกโมเดลที่มีความเป็นไปได้ในการเกิดเหตุการณ์ ถ้าโมเดลของคำใดมีค่าความน่าจะเป็นสูงกว่าโมเดลของคำอื่น ๆ จะแสดงว่าคำที่ไม่ทราบว่าเป็นเสียงใดก็คือคำนั้นนั่นเอง

ขั้นการเปรียบเทียบโมเดลเป็นขั้นนำเสียงอินพุตที่ไม่ทราบว่าเป็นคำใด เข้ามาผ่านกระบวนการสร้างโมเดลเฉพาะเสียงนั้น และนำโมเดลที่ได้นี้ไปเปรียบเทียบกับ โมเดลเสียงของทุกเสียงที่ได้เก็บไว้แล้ว



รูปที่ 2.22 แสดงขั้นตอนการตัดสินใจ

การเปรียบเทียบโมเดลนั้นจะใช้วิธีวิทเทอร์บี อัลกอริทึมซึ่งจะเลือกเส้นทางที่สั้นที่สุดของโมเดลเสียงที่ไม่ทราบกับ โมเดลที่เก็บไว้ทุกตัว ผลของการเลือกจะทำให้ทราบว่าเป็นเสียงใด

ความสั้นยาวของเส้นทางที่กล่าวถึงก็คือความน่าจะเป็นของการเปลี่ยนสแตตนั้นเอง ค่าปรากฏที่เกิดจากโมเดลของเสียงที่ไม่ทราบค่าจะเป็นตัวกำหนดสแตตที่เกิดขึ้นในแต่ละช่วงเวลา

การตัดสินใจว่าเสียงนั้นตรงกับ โมเดลใด ได้มาจากการเปรียบเทียบค่าที่น้อยที่สุดของ “ความยาวรวมของเส้นทางที่ได้จากสแตตทั้งหมด” ที่เปรียบเทียบกับแต่ละโมเดล

ผลที่ได้จากการเปรียบเทียบระยะทางระหว่างแบบทดสอบกับแบบอ้างอิงแต่ละแบบ ระยะที่ได้ทั้งหมดจะนำมาผ่านขั้นตอนการตัดสินใจ เพื่อหาผลลัพธ์ของการจดจำ กฎการตัดสินใจที่ใช้ในการจดจำเสียงพูด มีดังนี้ คือ

1. กฎ Nearest-Neighbor (NN)

ผลลัพธ์ของการจดจำจากกฎการตัดสินใจนี้ ได้แก่ แบบอ้างอิงที่มีระยะทางจากแบบทดสอบน้อยที่สุด กฎนี้เหมาะสมสำหรับระบบการจดจำเสียงพูดแบบบุคคลเดียว

2. กฎ K Nearest-Neighbor (KNN)

เป็นกฎที่ดัดแปลงมาจากกฎ NN อีกทีหนึ่ง เพื่อให้เหมาะสมกับการจดจำเสียงพูดแบบต่างบุคคล ซึ่งมีแบบอ้างอิงในแต่ละคำมากกว่า 1 แบบ กฎนี้แบ่งออกเป็น 2 แบบ คือ

2.1 แบบที่ 1 จะทำการหาแบบอ้างอิงจำนวน K แบบ ที่มีระยะทางห่างจากแบบทดสอบน้อยที่สุด ผลลัพธ์ของการจดจำจะได้จาก แบบอ้างอิงที่มีจำนวนมากที่สุดในจำนวน K แบบ ที่ได้รับการเลือก

2.2 แบบที่ 2 จะทำการหาแบบอ้างอิงจำนวน K แบบ ในแต่ละคำ โดยที่ทั้ง K แบบมีระยะทางห่างจากแบบทดสอบน้อยที่สุด ผลลัพธ์ของการเรียนรู้จะได้จาก แบบอ้างอิงที่ค่าเฉลี่ยของระยะทางในจำนวน K แบบ ที่ได้รับเลือกมาน้อยที่สุด วิธีนี้จะให้ผลลัพธ์ที่ถูกต้องมากขึ้น โดยกำหนดค่า K เป็น 2 หรือ 3 เมื่อเทียบกับกฎ NN อย่างไรก็ตามวิธีนี้จะทำให้เวลาที่ใช้ในการจดจำเพิ่มขึ้นด้วย

ในระบบเสียงพูดทั่วไป จะมีการกำหนดค่าระยะทางมากที่สุดที่ยอมรับได้ (Reject Value) เพื่อเพิ่มความถูกต้องของการจดจำในกรณีที่ผลลัพธ์ที่ได้จากการตัดสินใจมีค่าใกล้เคียงกันมาก วิธีการกำหนดค่านี้ต้องใช้ความระมัดระวังเป็นอย่างมาก เนื่องจากถ้ากำหนดค่านี้สูงเกินไป หรือไม่ได้กำหนด โอกาสที่จะทำให้ผลลัพธ์ของการจดจำผิดพลาดไปจะมีมาก แต่ถ้ากำหนดค่านี้ต่ำเกินไป จะทำให้ผลลัพธ์ของการจดจำสำหรับแบบอ้างอิงที่ถูกต้องเกิดขึ้นได้ยาก ดังนั้นโดยทั่วไปค่าระยะทางนี้มักจะได้มาจากการทดลอง เพื่อหาค่าที่เหมาะสมสำหรับกลุ่มของแบบอ้างอิงหนึ่ง ๆ เท่านั้น

2.4 หลักการตัดสินใจให้เป็นคำเดียว

หลักการตัดสินใจให้เป็นคำเดียวเพื่อใช้ในการวิเคราะห์เสียง จากหลักการของการวิเคราะห์เสียง จำเป็นต้องเป็นคำเดียวเท่านั้นจึงจะทำให้การจำเสียงได้ดี ดังนั้นเราจึงจำเป็นต้องแยกเสียงที่เข้ามาให้เป็นคำเดียวก่อน จากหลักการประมวลผลในเชิงเวลานั้นคือ การวิเคราะห์สัญญาณเสียงพูดโดยตรงจากรูปคลื่นสัญญาณ (wave form) ซึ่งแตกต่างจากการประมวลผลเชิงความถี่ (Frequency Domain) ตัวอย่างของการประมวลผลของสัญญาณเสียงพูดเชิงเวลา เช่น การหาค่าอัตราการตัดศูนย์เฉลี่ย (Zero-crossing Rate), การหาค่าพลังงาน (Energy), และค่าออโต้คอรัลเลชัน (Autocorrelation) โดยการประมวลผลหาค่าเหล่านี้เป็นไปโดยงานมีขั้นตอนที่ไม่ซับซ้อน แต่ให้องค์ประกอบสำคัญของสัญญาณเสียง แต่จากการศึกษาในการคำนวณหาคุณลักษณะของเสียงที่ใช้ในการตัดคำแล้ว เราจะเห็นได้ว่าวิธีการใช้การหาค่าพลังงานของเสียง นั่นคือการหาขนาดกำลังสองของสัญญาณเฉลี่ยในช่วงเวลาสั้น ๆ เป็นวิธีการที่สามารถทำการตัดคำได้ดี และยังใช้เวลาในการคำนวณน้อยเมื่อเปรียบเทียบกับวิธีการหาอื่น ๆ นั่นจึงทำให้การหาค่าพลังงานสามารถที่จะนำมาใช้งานได้จริง โดยจะมีหลักการที่จะใช้ศึกษาดังนี้

2.4.1 การประมวลผลเสียงพูดโดยขึ้นกับเวลา

สัญญาณเสียงพูดจะมีอัตราการสุ่มตัวอย่างของสัญญาณเป็น 8000 ค่าใน 1 วินาที จะเห็นได้ชัดว่าคุณสมบัติของสัญญาณเสียงพูดจะมีลักษณะที่เปลี่ยนแปลงไปตามเวลา ตัวอย่างเช่น การเปลี่ยนแปลงสัญญาณเสียงพูดระหว่างเสียงที่เป็นเสียงก้องหรือเสียงโหม่ง (voice) และเสียงไม่ก้องหรือเสียงอโหม่ง

(unvoice) โดยที่เสียงจะมีการเปลี่ยนแปลงที่เห็นได้ชัดจากขนาด (magnitude) ของสัญญาณ และยังมีการเปลี่ยนแปลงของความถี่มูลฐานของเสียง (fundamental frequency) ภายในช่วงที่เป็นโหมะด้วย โดยการที่แสดงรูปแบบสัญญาณเป็นรูปคลื่นนี้สามารถทำให้ง่ายต่อการสังเกตถึงองค์ประกอบและคุณลักษณะของเสียงนั้น เช่น ความเข้มของเสียง (Intensity) , ชนิดของเสียง (เสียงก้องหรือเสียงไม่ก้อง) หรือ excitation mode , ความถี่มูลฐาน (pitch หรือ fundamental frequency) และอาจรวมถึงสัมประสิทธิ์ในอวัยวะกำเนิดเสียง เช่น ความถี่กำพอน (resonant หรือ formant frequency)

สมมติฐานที่ใช้ในการประมวลผลสัญญาณเสียงส่วนมากก็คือการที่ถือเอาว่าคุณลักษณะของเสียงนั้นมีการเปลี่ยนแปลงช้ามากเมื่อเทียบกับเวลา และสมมติฐานนี้ทำให้เกิดการประมวลผลที่เรียกว่าการประมวลผลแบบช่วงเวลาสั้น ๆ (short time) โดยที่ส่วนเล็ก ๆ แต่ละส่วนของเสียงพูดนั้นที่ถูกแยกออกมาและจะถูกประมวลผลโดยแต่ละส่วนของเสียงพูดนี้เรียกว่าการวิเคราะห์เฟรม (analysis frames) ซึ่งเฟรมแต่ละเฟรมจะมีส่วนที่ซ้อนทับกัน และผลลัพธ์ที่ได้ของแต่ละส่วนนั้นอาจเป็นตัวเลขตัวเดียว หรืออาจเป็นกลุ่มของตัวเลขก็ได้ ดังนั้นกระบวนการนี้จะทำให้เกิดลำดับของค่าใหม่ในเชิงของเวลา ซึ่งแสดงถึงลักษณะของสัญญาณเสียง

กระบวนการวิเคราะห์เสียงแบบช่วงเวลาสั้น ๆ นั้นเกือบทั้งหมดสามารถแสดงในรูปของสมการทางคณิตศาสตร์ที่มีรูปแบบคือ

$$Q_n = \sum_{m=-\infty}^{\infty} T[X(m)]W(n-m)$$

สมการนี้สัญญาณเสียงจะถูกกรองเอาเฉพาะย่านความถี่ที่ต้องการ และจะถูกกระทำโดยฟังก์ชัน $T[]$ ซึ่งอาจเป็นเชิงเส้นหรือไม่ขึ้นอยู่กับสัมประสิทธิ์ที่จะทำการคำนวณ และหลังจากนั้นลำดับที่ได้จะถูกคูณด้วยลำดับวินโดว์ ในตำแหน่งที่สัมพันธ์กับเวลาของตัวอย่างสัญญาณที่ n และผลคูณก็จะถูกรวมกันโดยทั่วไปแล้วผลลัพธ์จะเกิดจากลำดับที่จำกัด

2.4.2 ค่าขนาดกำลังสองของสัญญาณเฉลี่ยในช่วงเวลาสั้น ๆ

เราสามารถสังเกตได้ว่าขนาดของสัญญาณ (Amplitude) นั้นจะมีการเปลี่ยนแปลงไปตามเวลา โดยเฉพาะขนาดของสัญญาณของเสียงไม่ก้อง มักจะมีค่าต่ำกว่าขนาดของสัญญาณของเสียงก้องมาก ซึ่งการใช้ค่ากำลังสองของสัญญาณเฉลี่ยในช่วงสั้น ๆ สามารถแสดงให้เห็นถึงความแตกต่างนี้ได้อย่างชัดเจนได้มากกว่าการใช้ขนาด โดยมีรูปแบบสมการของการใช้ในการคำนวณ คือ

$$E_n = \sum_{m=-\infty}^{\infty} [X(m)W(n-m)]^2$$

หรือสามารถเขียนได้เป็น $E_n = \sum_{m=-\infty}^{\infty} X^2(m)h(n-m)$

โดยที่ $h(n) = W^2(m)$

สมการนี้สามารถแสดงเป็นบล็อกไดอะแกรมได้ดังรูปที่ 2.28 สัญญาณ $X^2(n)$ จะถูกกรองโดยตัวกรองแบบเชิงเส้น $h(n)$ ก็คือวินโดว์ (window) ซึ่งวินโดว์ที่แตกต่างกันจะมีผลกับค่าพลังงานผลลัพธ์ที่ได้ โดยถ้า $h(n)$ ให้ค่าขนาดของสัญญาณคงที่ในช่วงเวลาที่ยาวนานแล้ว ค่า E จะมีการเปลี่ยนแปลงน้อยมาก เนื่องจากเราต้องการให้ค่ากำลังสองในช่วงเวลาสั้น ๆ แสดงให้เห็นถึงการเปลี่ยนแปลงของขนาดเสียงในสัญญาณเสียง ดังนั้นวินโดว์ที่มีช่วงเวลาด้านเกินไปก็จะทำให้ไม่ได้ค่าความเรียบพอเพียง โดยตัวฟังก์ชันวินโดว์มีรูปแบบดังนี้

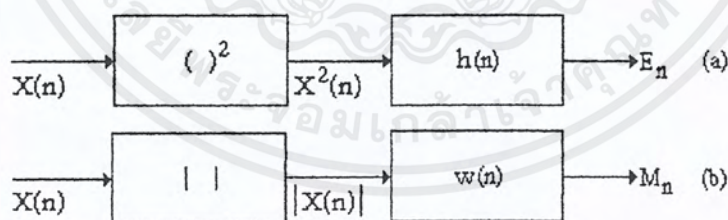
วินโดว์แบบสี่เหลี่ยมผืนผ้า $h(n) = 1$; เมื่อ $0 \leq n \leq N-1$
 $h(n) = 0$; อื่น ๆ

วินโดว์แบบแฮมมิง $h(n) = 0.54 - 0.46 \cos\left(\frac{2n\pi}{(N-1)}\right)$; เมื่อ $0 \leq n \leq N-1$
 $h(n) = 0$; อื่น ๆ

จากสมการจะเห็นได้ว่า วินโดว์แบบสี่เหลี่ยมผืนผ้า จะมีการถ่วงน้ำหนัก (weight) ค่าเท่ากันตลอดทุกสัญญาณ (Sample) ในช่วง $(n-N+1)$ ถึง n ส่วนผลตอบสนองทางความถี่ของวินโดว์แบบสี่เหลี่ยมผืนผ้าจะได้ตามสมการดังนี้

$$H(e^{j\Omega T}) = \frac{\sin(\Omega NT / 2)}{\sin(\Omega T / 2)} e^{-j\Omega T (N-1) / 2}$$

ส่วนแฮมมิงวินโดว์นั้นจะใช้ประโยชน์ได้ดีสำหรับการวิเคราะห์เชิงความถี่ เพราะให้ค่าลดทอนภายนอกย่านมากกว่า



รูปที่ 2.23 แสดงบล็อกไดอะแกรม (a) ขนาดกำลังสองในช่วงเวลาสั้น ๆ (b) ขนาดในช่วงเวลาสั้น ๆ

จากที่กล่าวมาแล้วว่าถ้าค่า N ในวินโดว์มีค่าน้อยเกินไปก็จะมีการเปลี่ยนแปลงของค่า E_n มากเกินไป แต่ถ้า N มากเกินไปก็จะทำให้สูญเสียคุณสมบัติของเสียงไป แต่เป็นการยากที่จะกำหนดค่า N เพียงค่าเดียวให้เหมาะสมกับเสียงทั้งหมด เนื่องจากคาบเวลาของเสียงที่เรียกว่า พิตช์ จะแตกต่างกันเช่น เด็กหรือผู้หญิงอาจมีค่าเพียง 20 ตัวอย่าง ที่อัตราสุ่มตัวอย่าง 10 เฮิร์ตซ์ แต่ผู้ชายอาจจะสูงถึง 250 ตัวอย่าง แต่ค่า N ที่ใช้กันนั้นอยู่ในช่วงประมาณ 100-200 ตัวอย่างที่อัตราสุ่ม 10 เฮิร์ตซ์ (10-20 มิลลิวินาที)

สิ่งสำคัญที่ได้จากค่า E_n นั่นคือ การที่เราสามารถแยกแยะส่วนของเสียงก้อง และเสียงไม่ก้องออกจากกันได้ โดยส่วนของเสียงที่ไม่ก้องนั้นจะมีค่า E_n ที่ต่ำกว่าเสียงก้องอย่างเห็นได้ชัด ซึ่งจะทำให้สามารถทำการกำหนดจุดได้อย่างคร่าว ๆ ได้ว่าตำแหน่งไหนเป็นจุดที่เสียงได้ทำการเปลี่ยนจากเสียงก้องหรือในทางกลับกันสัญญาณที่มีค่าอัตราส่วนสัญญาณเสียงต่อสัญญาณรบกวนสูง ๆ นั้น E_n จะทำให้เราสามารถแยกเสียงพูดออกจากเสียงเงียบ (silence) ได้ด้วย

2.4.3 การแยกเสียงพูดออกจากเสียงเงียบ(silence)

ในระบบการประมวลผลสัญญาณเสียงพูดนั้น สิ่งสำคัญสิ่งหนึ่งคือการแยกแยะและการหาขอบเขตทั้งจุดเริ่มต้น และจุดสิ้นสุดของเสียงออกจากสัญญาณรบกวน(Background Noise) โดยเฉพาะในกระบวนการรู้และจำเสียงพูดชนิดคำโดด(Isolated Word) จำเป็นอย่างยิ่งที่จะต้องกำหนดจุดเริ่มต้น และจุดสิ้นสุดของคำให้ได้อย่างมีประสิทธิภาพ

ลักษณะของเสียงที่ยากต่อการกำหนดขอบเขต มีดังนี้

- ลักษณะของเสียงที่พ่นลมออกมา ณ จุดเริ่มต้นหรือสิ้นสุด(Weak fricative)
- ลักษณะของเสียงที่เป็นการระเบิดออกมา(Weak plosive bursts)
- ลักษณะของเสียงที่เป็นเสียงนาสิก(Nasals)

ในการที่จะกำหนดจุดเริ่มต้นและสิ้นสุดของเสียงเหล่านี้ สามารถใช้กระบวนการของการหาขนาดกำลังสองและอัตราการตัดศูนย์เฉลี่ยร่วมกันได้ ตัวอย่างในระบบรู้และจำเสียงพูดแบบคำโดดการบันทึกเสียงจะบันทึกส่วนที่เป็นเสียงพูดและส่วนที่ไม่ได้พูดไว้ด้วย แล้วใช้ระบบหาจุดเริ่มต้นและสิ้นสุดของคำเพื่อที่จะทำให้ผ่านเข้ากระบวนการรู้และจำเสียงโดยส่งเฉพาะส่วนที่เป็นเสียงพูดเข้าไปประมวลผล

การทำงานของระบบนี้จะใช้การหาค่าอัตราการตัดศูนย์เฉลี่ยที่มีขนาดเฟรมเท่ากับ 10 มิลลิวินาที และการหาค่าพลังงาน และอัตราการตัดศูนย์เฉลี่ยร่วมกัน โดยใช้วินโดวขนาด 10 มิลลิวินาที ซึ่งทั้ง 2 กระบวนการจะทำตลอดช่วงของสัญญาณด้วยอัตรา 100 ครั้งต่อวินาที โดยการสมมติให้ช่วงเวลา 100 มิลลิวินาทีแรกไม่มีเสียงพูด หมายความว่าค่าอัตราการตัดศูนย์ และค่าพลังงานในช่วงนี้จะเป็นค่าของสัญญาณรบกวน และทำการคำนวณค่าพลังงาน และอัตราการตัดศูนย์เฉลี่ยที่ใช้เป็นเกณฑ์ในการตัดสินใจลำดับของพลังงาน

ค่าระดับที่ใช้ส่วนมากจะให้ค่าโดยประมาณซึ่งในความเป็นจริงแล้วจุดเริ่มต้น และจุดสิ้นสุดนั้นจะอยู่ภายนอกจากค่าระดับนี้ จึงมีการทำซ้ำโดยดูจากจุดที่กำหนดโดยค่าระดับในครั้งแรกและลดลงจนถึงค่าระดับอีกค่าหนึ่งจึงถือได้ว่าเป็นจุดเริ่มต้นและจุดสิ้นสุดของคำ ขอบเขตที่ได้จากการใช้ค่าระดับแรกนั้นเป็นที่น่าเชื่อถือว่าภายในขอบเขตนี้จะไม่มีการจุดเริ่มต้นหรือจุดสิ้นสุดของคำอยู่

กระบวนการถัดไปคือการย้อนกลับจากจุดเริ่มต้นที่ได้จากค่าระดับแรกแล้ว ทำการเปรียบเทียบค่าอัตราการตัดศูนย์เฉลี่ยกับค่าระดับการตัดศูนย์เฉลี่ยของมันซึ่งได้จากสัญญาณรบกวน โดยถ้าอัตราการตัดศูนย์เฉลี่ยมีค่ามากกว่าค่าระดับเกิน 3 ครั้งแล้ว จะถือว่าจุดเริ่มต้นเป็นจุดที่มีค่าอัตราการตัดศูนย์เฉลี่ยเกินค่าระดับเป็นครั้งแรก

2.5 โครงสร้างและสถาปัตยกรรมของไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลช

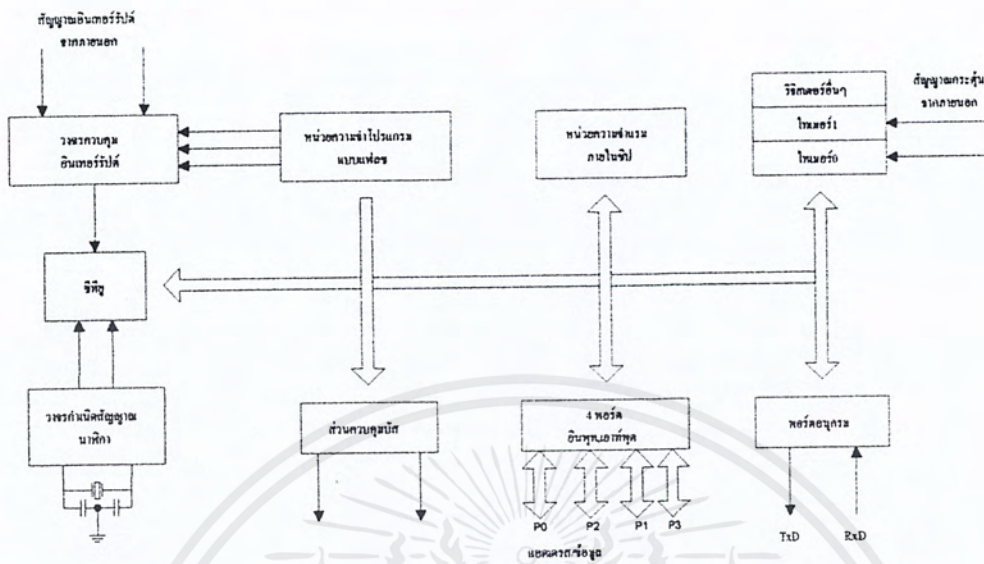
ไมโครคอนโทรลเลอร์ที่ใช้ในโครงงานนี้จะอ้างอิงถึงไมโครคอนโทรลเลอร์ตระกูล MCS-51 ซึ่งมีหน่วยความจำแบบแฟลช (flash memory) ของ Atmel Corporation ซึ่งเบอร์ที่ใช้ในโครงงานคือ AT89S53 เหตุผลที่ใช้ไมโครคอนโทรลเลอร์เบอร์นี้ในโครงงานก็เนื่องจาก

- 1.) หน่วยความจำโปรแกรมภายในเป็นแบบแฟลช ทำให้สามารถลบและเขียนใหม่ได้ประมาณหนึ่งพันครั้ง จึงสามารถใช้งานในรูปแบบของไมโครคอนโทรลเลอร์ชิปเดี่ยว โดยไม่จำเป็นต้องต่อหน่วยความจำภายนอก จึงส่งผลให้สามารถใช้งานอินพุตและเอาต์พุตพอร์ท ได้อย่างเต็มประสิทธิภาพ
- 2.) ต้นทุนและเวลาในการพัฒนาระบบไมโครคอนโทรลเลอร์ลดลงมาก เพราะไม่ต้องใช้เครื่องมือพัฒนาจำพวกอิมูเลเตอร์และเครื่องโปรแกรมอีพรอม
- 3.) บริษัทผู้ผลิตไมโครคอนโทรลเลอร์ตระกูลนี้ผลิตออกมาหลายเบอร์ และมีความสามารถแตกต่างกันไป ทำให้มีทางเลือกในการใช้งานสูง
- 4.) สามารถป้องกันการคัดลอกข้อมูลของหน่วยความจำโปรแกรมได้เป็นอย่างดี เพราะการใช้หน่วยความจำภายในไมโครคอนโทรลเลอร์
- 5.) ไมโครคอนโทรลเลอร์บางเบอร์ที่ผลิตโดย Atmel (เช่น AT89S53 ที่ใช้ในโครงงานนี้) สามารถทำการ โปรแกรมข้อมูลในหน่วยความจำโปรแกรมได้โดยไม่ต้องถอดตัวไมโครคอนโทรลเลอร์ออกมาทำการ โปรแกรมใหม่หรือที่เรียกว่า การ โปรแกรมในวงจร หรือในระบบ(In-system programming) โดยใช้ลักษณะการติดต่อแบบ SPI (Serial Peripheral Interface) ทำให้การพัฒนาหรือการซ่อมบำรุง ตลอดจนการปรับปรุงข้อมูลในหน่วยความจำโปรแกรมทำได้สะดวก ภายใต้งบประมาณที่ไม่แพง
- 6.) ชุดคำสั่งและสถาปัตยกรรมพื้นฐานเหมือนกับไมโครคอนโทรลเลอร์ MCS-51 ของผู้ผลิตอื่น ไม่ว่าจะเป็นอินเทล ซิเมนส์ หรือ คัลลิส ดังนั้นจึงสามารถนำความรู้พื้นฐานเดิม มาพัฒนาโครงงานได้สะดวกและรวดเร็ว

2.5.1 คุณสมบัติทางเทคนิคของไมโครคอนโทรลเลอร์ตระกูล MCS-51 อนุกรม AT89xx

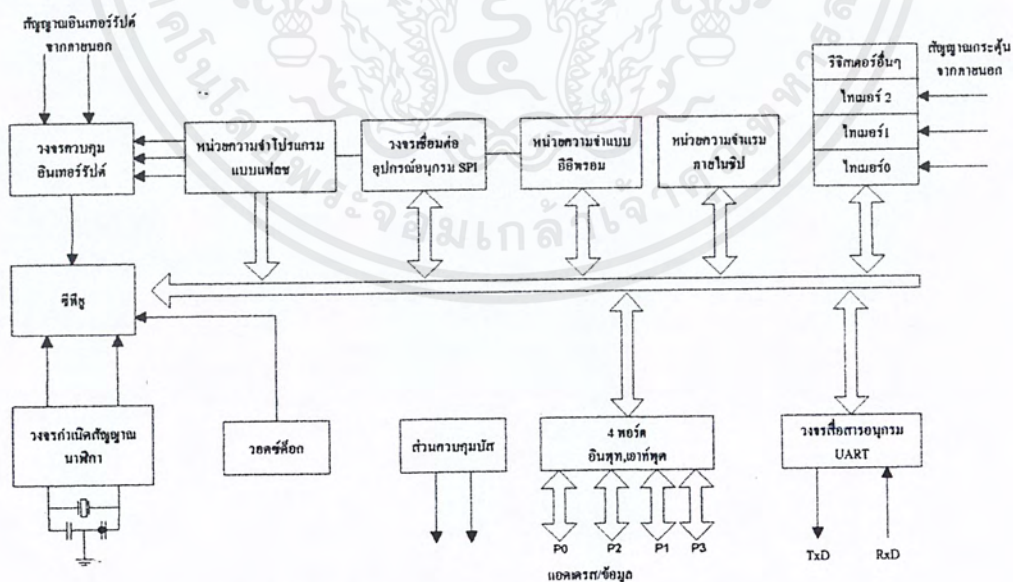
- 1.) เป็นไมโครคอนโทรลเลอร์ที่ใช้ซีพียูขนาด 8 บิต
- 2.) ภายในมีหน่วยความจำโปรแกรมเป็นแบบแฟลชสามารถลบและเขียนใหม่ได้พันครั้ง
- 3.) หน่วยความจำข้อมูลพื้นฐานเป็นหน่วยความจำแบบแรม ในบางเบอร์จะมีหน่วยความจำแบบอีพรอมเพิ่มเติม
- 4.) พอร์ทเป็นแบบสองทิศทางสามารถใช้งานเป็นได้ทั้งอินพุตและเอาต์พุต
- 5.) มีวงจรสื่อสารอนุกรมแบบดูเพล็กซ์เต็ม (full duplex)
- 6.) ไทมเมอร์/คาน์เตอร์ ขนาด 16 บิต อย่างน้อย 2 ตัว
- 7.) สามารถรองรับแหล่งกำเนิดการอินเทอร์รัปต์ได้ 6 ประเภท
- 8.) สามารถขยายหน่วยความจำภายนอกเพิ่มเติมได้สูงสุด 64 กิโลไบต์
- 9.) มีวงจรกำเนิดสัญญาณพิกายู่ภายในชิป
- 10.) มีวงจรสื่อสารอนุกรมแบบ SPI สำหรับในอนุกรม AT89Sxx

11.) มีวอตซ์ด็อกไทเมอร์ในตัว สำหรับในอนุกรม AT89Sxx



รูปที่ 2.24 โครงสร้างพื้นฐานของไมโครคอนโทรลเลอร์ MCS-51 ในอนุกรม AT89Cxx

จากรูปที่ 2.24 จะเห็นได้ว่า โครงสร้างของ AT89Cxx จะเหมือนกับไมโครคอนโทรลเลอร์ตระกูล MCS-51 พื้นฐาน หากแตกต่างกันเฉพาะหน่วยความจำโปรแกรมแบบแฟลชที่เพิ่มเติมขึ้นมา หากเป็นไมโครคอนโทรลเลอร์ ในอนุกรม 89Cxx หน่วยความจำโปรแกรมภายในจะเป็นแบบอีพรอม และบางเบอร์สามารถโปรแกรมได้ครั้งเดียว



รูปที่ 2.25 โครงสร้างพื้นฐานของอนุกรม AT89Sxx

จากรูปที่ 2.25 จะเห็นได้ว่า มีส่วนประกอบที่เพิ่มเติมแตกต่างจาก AT89Cxx อยู่หลายส่วน อาทิ วงจรเชื่อมต่ออนุกรมแบบ SPI ซึ่งในไมโครคอนโทรลเลอร์อนุกรมนี้ ใช้ในการเขียนข้อมูลลงในหน่วยความจำโปรแกรม โดยไม่ต้องถอดตัวชิปออกไปจากระบบหรือเรียกว่า การโปรแกรมในวงจรไทมเมอร์/เคาน์เตอร์ขนาด 16 บิต ที่เพิ่มเติมเข้ามา อีกหนึ่งตัวเป็นไทมเมอร์ 2 และวงจรวอตช์ด็อกที่ใช้ในการตรวจสอบความผิดพลาดของชิป

ตารางที่ 2.2 แสดงรายละเอียดของไมโครคอนโทรลเลอร์ ตระกูล MCS-51 ที่ Atmel ผลิตขึ้น

เบอร์ของไมโครคอนโทรลเลอร์	หน่วยความจำโปรแกรม	หน่วยความจำข้อมูล	จำนวนไทมเมอร์/เคาน์เตอร์ 16 บิต
AT89C1051	แบบแฟลช 1 กิโลไบต์	แรม 64 ไบต์	1
AT89C2051	แบบแฟลช 2 กิโลไบต์	แรม 128 ไบต์	2
AT89C51	แบบแฟลช 4 กิโลไบต์	แรม 128 ไบต์	2
AT89C52	แบบแฟลช 8 กิโลไบต์	แรม 256 ไบต์	3
AT89C55	แบบแฟลช 20 กิโลไบต์	แรม 256 ไบต์	3
AT89S8252	แบบแฟลช 8 กิโลไบต์	แรม 256 ไบต์ อีพรอม 2 กิโลไบต์	3
**AT89S53	แบบแฟลช 12 กิโลไบต์	แรม 256 ไบต์	3

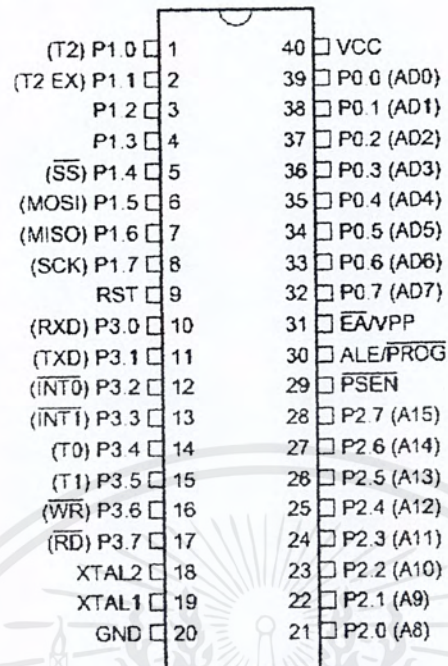
หมายเหตุ ** เป็นเบอร์ที่ใช้ในการทำโครงการ

ในตารางที่ 2.2 แสดงรายละเอียดบางส่วนของไมโครคอนโทรลเลอร์ ตระกูล MCS-51 แต่ละเบอร์ที่ Atmel ผลิตขึ้น และมีใช้งานอยู่ในปัจจุบัน

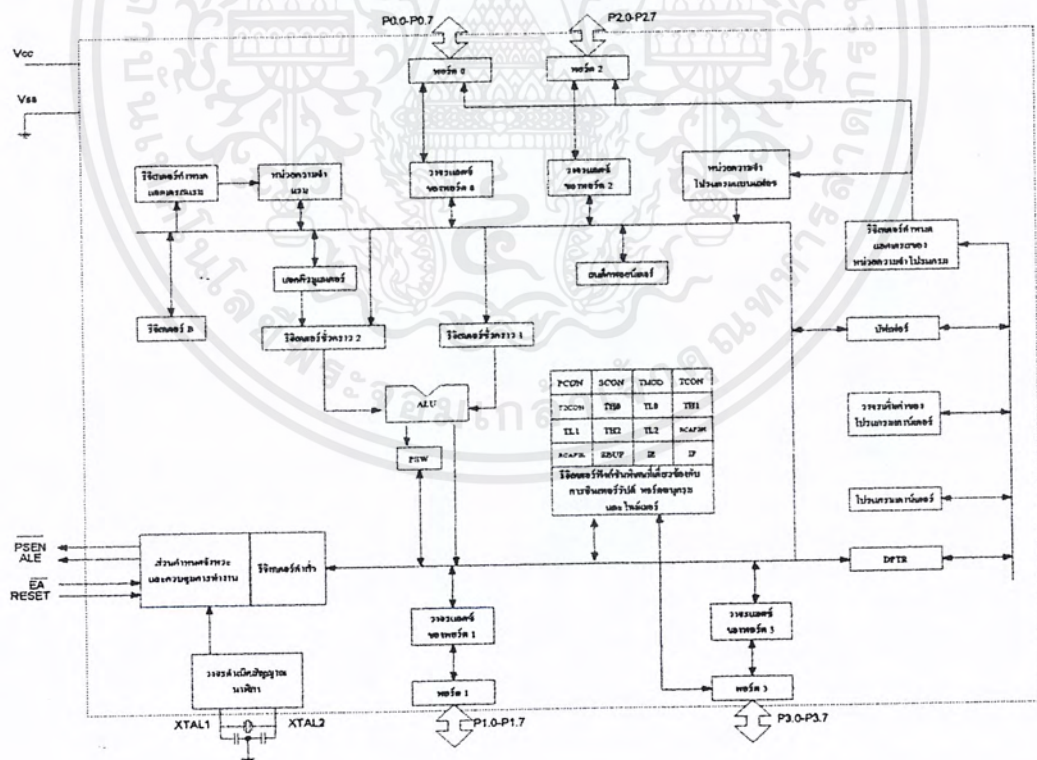
2.5.2 การจัดขาของไมโครคอนโทรลเลอร์ MCS-51

ไมโครคอนโทรลเลอร์ MCS-51 ทุกเบอร์จะมีสถาปัตยกรรมและขาใช้งานพื้นฐานเหมือนกัน ดังแสดงในรูปที่ 2.26 และ 2.27 โดยมีรายละเอียดขั้นต้น ดังนี้

รูปที่ 2.26 แสดงการจัดขามาตรฐานของไมโครคอนโทรลเลอร์ เบอร์ 89S53



รูปที่ 2.27 แสดงรายละเอียดโครงสร้างหลักของไมโครคอนโทรลเลอร์แบบแฟลช



ขา Vcc ใช้สำหรับต่อไปเลี้ยง +5 Volt

ขา Gnd เป็นขาราวด์ สำหรับต่อกับกราวด์ของระบบ

ขาพอร์ท 0 (P0.0-P0.7) มี 8 ขา แต่ละขาสามารถกำหนดให้เป็นได้ทั้งอินพุตและเอาต์พุตสำหรับใช้งานทั่วไป ถ้าหากต้องการกำหนดให้ขาพอร์ท 0 ขาใดขาหนึ่งเป็นอินพุตสามารถทำได้โดยการเขียนข้อมูล “1” ไปยังแต่ละบิตของพอร์ทที่ต้องการติดต่อด้วย ส่งผลให้ขาพอร์ทนั้นมีสถานะปล่อยลอย (float) จึงมีอินพุตอิมพีแดนซ์สูง สามารถใช้งานเป็นขาพอร์ทอินพุตได้ นอกจากนั้นขาพอร์ทนี้ยังถูกใช้งานในการติดต่อกับขาแอดเดรสไบต์ต่ำของหน่วยความจำภายนอก (A0 – A7) และขาข้อมูล (D0 – D7) โดยใช้กระบวนการมัลติเพล็กซ์เข้าช่วยเพื่อสลับการทำงานให้เป็นได้ทั้งขาติดต่อกับแอดเดรสและขาติดต่อกับข้อมูล

ขาพอร์ท 1 (P1.0 – P1.7) มี 8 ขา แต่ละขาสามารถกำหนดให้เป็นได้ทั้งอินพุตและเอาต์พุตสำหรับใช้งานทั่วไป ถ้าหากต้องการกำหนดให้ขาพอร์ท 1 ขาใดขาหนึ่งเป็นอินพุต สามารถทำได้โดยการเขียนข้อมูล “1” ไปยังแต่ละบิตของพอร์ทที่ต้องการติดต่อกับ นอกจากนั้นในอนุกรม AT89Sxx จะใช้ขา P1.0 เป็นขาอินพุตสำหรับนับค่าของไทเมอร์ 2 และ P1.1 เป็นขาอินพุตทริกเกอร์ของไทเมอร์ 2 ในขณะที่ขา P1.4 ถึง P1.7 เป็นขาสำหรับเชื่อมต่อแบบ SPI เพื่อทำการโปรแกรมข้อมูลในระบบ

ขาพอร์ท 2 (P2.0 – P2.7) มี 8 ขา แต่ละขาสามารถกำหนดให้เป็นได้ทั้งอินพุตและเอาต์พุตสำหรับใช้งานทั่วไป ถ้าหากต้องการกำหนดให้ขาพอร์ท 2 ขาใดขาหนึ่งเป็นอินพุต สามารถทำได้โดยการเขียนข้อมูล “1” ไปยังแต่ละบิตของพอร์ทที่ต้องการติดต่อกับ ส่งผลให้ขาพอร์ทนี้มีสถานะปล่อยลอย จึงมีอินพุตอิมพีแดนซ์สูง สามารถใช้งานเป็นขาพอร์ทอินพุตได้ นอกจากนั้นขาพอร์ทนี้ยังถูกใช้งานในการติดต่อกับขาแอดเดรสไบต์สูงของหน่วยความจำภายนอก (A8 – A15)

ขาพอร์ท 3 (P3.0 – P3.7) มี 8 ขา แต่ละขาสามารถกำหนดให้เป็นได้ทั้งอินพุตและเอาต์พุตสำหรับใช้งานทั่วไป ถ้าหากต้องการกำหนดให้ขาพอร์ท 3 ขาใดขาหนึ่งเป็นอินพุต ก็สามารถทำได้โดยการเขียนข้อมูล “1” ไปยังแต่ละบิตของพอร์ทที่ต้องการติดต่อกับ ส่งผลให้ขาพอร์ทนี้มีสถานะปล่อยลอย จึงมีอินพุตอิมพีแดนซ์สูง สามารถใช้งานเป็นขาพอร์ทอินพุตได้ นอกจากนั้นขาพอร์ท 3 ยังเป็นขาที่มีหน้าที่การใช้งานพิเศษ ดังมีรายละเอียดดังนี้

P3.0 ใช้เป็นขาอินพุต สำหรับรับข้อมูลจากการสื่อสารแบบอนุกรม หรือขา RxD

P3.1 ใช้เป็นขาอินพุต สำหรับส่งข้อมูลจากการสื่อสารแบบอนุกรม หรือขา TxD

P3.2 ใช้เป็นขาอินพุต สำหรับรับสัญญาณอินเทอร์รัปต์จากภายนอกช่องที่ 0 หรือขา $\overline{\text{INT0}}$

P3.3 ใช้เป็นขาอินพุต สำหรับรับสัญญาณอินเทอร์รัปต์จากภายนอกช่องที่ 1 หรือขา $\overline{\text{INT1}}$

P3.4 ใช้เป็นขาอินพุต สำหรับรับสัญญาณไทเมอร์จากภายนอกช่องที่ 0 หรือขา T0

P3.5 ใช้เป็นขาอินพุต สำหรับรับสัญญาณไทเมอร์จากภายนอกช่องที่ 1 หรือขา T1

P3.6 ใช้เป็นขาสัญญาณ $\overline{\text{WR}}$ ในกรณีที่ใช้เชื่อมต่อกับหน่วยความจำภายนอก

P3.7 ใช้เป็นขาสัญญาณ $\overline{\text{RD}}$ ในกรณีที่ใช้เชื่อมต่อกับหน่วยความจำภายนอก

ขา reset ใช้ในการรีเซ็ตการทำงานของไมโครคอนโทรลเลอร์ โดยในการป้อนสัญญาณเพื่อรีเซ็ต สถานะที่ขานี้ต้องอยู่ในระดับรีเซ็ตอย่างน้อย 2 แมกซีนไซเคิล โดยที่วงจรกำเนิดสัญญาณนาฬิกายังคงทำงานต่อเนื่องไปอย่างปกติ

ขา ALE/PROG (Address Latch Enable/ Program Pulse Input) เป็นขาที่ใช้ในการควบคุมการแลตช์ของขาพอร์ท 0 เมื่อมีการใช้งานหน่วยความจำภายนอก นอกจากนั้นขานี้ยังใช้เป็นขาสำหรับรับ

พัลส์ของการโปรแกรมสำหรับโปรแกรมข้อมูลลงในไมโครคอนโทรลเลอร์ MCS-51 ในรุ่นที่มีหน่วยความจำโปรแกรมเป็นแบบอีพรอม

ขา EA/Vpp (External Access Enable/ Programming Voltage Input) ใช้สำหรับเลือกการติดต่อหน่วยความจำโปรแกรมจากภายนอกหรือภายในตัวไมโครคอนโทรลเลอร์ ถ้าหากขานี้เป็น “0” เป็นการเลือกให้ไมโครคอนโทรลเลอร์ติดต่อกับหน่วยความจำโปรแกรมภายนอก แต่ถ้าหากขานี้เป็น “1” เป็นการเลือกให้ไมโครคอนโทรลเลอร์ติดต่อกับหน่วยความจำภายในตัวไมโครคอนโทรลเลอร์ นอกจากนี้ที่ขานี้ยังใช้เป็นขาอินพุตสำหรับแรงดันไฟสูงสำหรับการโปรแกรมหน่วยความจำภายในไมโครคอนโทรลเลอร์สำหรับไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลชต้องการแรงดันสำหรับการโปรแกรม +12 โวลต์

ขา XTAL1 และ XTAL2 เป็นขาสำหรับต่อคริสตัล เพื่อการสร้างสัญญาณนาฬิกาในการกำหนดจังหวะการทำงานของไมโครคอนโทรลเลอร์

2.5.3 โครงสร้างและการทำงานของพอร์ท

ไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลชมีพอร์ทให้ใช้งานทั้งสิ้น 4 พอร์ทคือ พอร์ท 0, 1, 2 และ 3 แต่ละพอร์ทมีขนาด 8 บิต เป็นพอร์ท แบบ 2 ทิศทาง กล่าวคือ สามารถเป็นได้ทั้งอินพุตสำหรับรับสัญญาณข้อมูลเข้าและเอาต์พุตสำหรับสัญญาณข้อมูลออก ทุกพอร์ทของไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลชมีวงจรแลคซ์และวงจรขับ คลอคจนบัฟเฟอร์อินพุต ดังในรูปที่ 2.27

ตารางที่ 2.3 แสดงหน้าที่พิเศษของพอร์ท 1 ในไมโครคอนโทรลเลอร์แบบแฟรช

ขา	เบอร์ของ	หน้าที่พิเศษ
P1.0	AT89C52/AT89Sxx	ขา T2 เป็นขาอินพุตนับค่าของไทเมอร์/เคาน์เตอร์ 2 และเป็นขาเอาต์พุตของการกำเนิดสัญญาณนาฬิกาโดยไทเมอร์ 2 (Clock out)
P1.1	AT89C52/AT89Sxx	ขา T2EX เป็นขาอินพุตทริกเกอร์สำหรับการแคปเจอร์ / รีโพลด์และควบคุมทิศทางของสัญญาณ
P1.4	AT89Sxx	ขา $\overline{SS}$ (Slave Select) เป็นขาเลือกการติดต่อในกรณีที่ไมโครคอนโทรลเลอร์เป็นอุปกรณ์สเลฟ ในระบบการติดต่อแบบ SPI
P1.5	AT89Sxx	ขา MOIS (Master Data Output , Slave Data Output) ใช้ในการติดต่อกับพอร์ท SPI
P1.6	AT89Sxx	ขา MISO (Master Data Input , Slave Data Output) ใช้ในการติดต่อกับพอร์ท SPI
P1.7	AT89Sxx	ขา SCK (Master Clock Output) เป็นขาสัญญาณนาฬิกาของการติดต่อกับพอร์ท SPI

ที่พอร์ท 0 และพอร์ท 2 จะใช้งานเป็นพอร์ทอินพุตและเอาต์พุตสำหรับงานทั่วไป และใช้ในการติดต่อกับหน่วยความจำภายนอก สำหรับพอร์ท 3 และพอร์ท 1 บางงานนอกจากจะใช้เป็นขาอินพุตหรือเอาต์พุตพอร์ทตามปกติแล้วยังสามารถใช้งานในหน้าที่พิเศษได้อีก ขึ้นอยู่กับว่าเป็นไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลชเบอร์ใด ดังตารางที่ 2.3

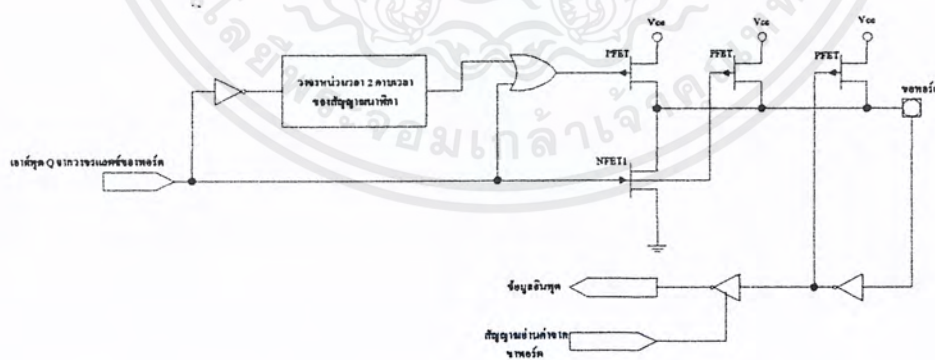
ในรูปที่ 2.28 แสดงวงจรภายในของแต่ละพอร์ทของไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลช โดยในรูปที่ 2.28(ก) เป็นวงจรของพอร์ท 0 วงจรแลคช์ของแต่ละบิตในแต่ละพอร์ทก็คือวงจรดีฟลิปฟล็อปนั่นเอง การอ่านค่าสถานะของพอร์ทแต่ละสถานะของวงจรแลคช์สามารถกระทำได้อย่างอิสระต่อกัน ด้วยสัญญาณที่แยกจากกัน นั่นคือสัญญาณอ่านข้อมูลจากขาพอร์ทและสัญญาณอ่านข้อมูลจากวงจรแลคช์ ส่วนการเขียนข้อมูลมายังพอร์ท ต้องส่งสัญญาณมายังขา CLK ของดีฟลิปฟล็อปในขณะที่ข้อมูลจะส่งผ่านมาจากขาบั๊สข้อมูลภายในเข้าสู่ขา D ของดีฟลิปฟล็อป

ที่พอร์ทนี้มีวงจรมัลติเพลกซ์สำหรับกำหนดลักษณะการทำงานของพอร์ทว่าต้องการใช้งานเป็นขาอินพุตเอาต์พุตปกติหรือใช้ในการติดต่อกับหน่วยความจำภายนอกไมโครคอนโทรลเลอร์

เนื่องจากที่ขาพอร์ท 0 ไม่มีวงจรพูลอัพภายในหากมีการนำพอร์ท 0 ไปใช้งานเป็นพอร์ทอินพุตจะต้องต่อตัวต้านทานพูลอัพภายนอกเข้าที่ขาพอร์ท 0 ทุกขาด้วย

ในรูปที่ 2.28(ข) เป็นวงจรของพอร์ท 1 ซึ่งมีลักษณะโดยทั่วไปคล้ายกับพอร์ท 0 หากแต่ไม่มีวงจรมัลติเพลกซ์เนื่องจากพอร์ทนี้จะไม่ใช้ในการติดต่อกับหน่วยความจำภายนอก แต่จะมีวงจรพูลอัพภายในที่แต่ละบิตของพอร์ทนี้แทน สำหรับรายละเอียดของวงจรพูลอัพแสดงในรูปที่ 2.29

ในรูปที่ 2.28(ค) เป็นวงจรของพอร์ท 2 จะคล้ายกับพอร์ท 0 มาก ต่างกันเพียงมีวงจรพูลอัพเพิ่มเติมเข้ามา ส่วนในรูป 2.28(ง) เป็นวงจรภายในของพอร์ท 3 จะเห็นได้ว่าคล้ายกับพอร์ท 1 มีการเพิ่มเติมวงจรบัฟเฟอร์ และวงจรอินพุตเอาต์พุตเมื่อทำงานในฟังก์ชันพิเศษเข้ามา เนื่องจากพอร์ท 3 สามารถนำไปใช้งานในหน้าที่พิเศษได้ทุกขา



รูปที่ 2.29 แสดงวงจรพูลอัพภายในพอร์ทของไมโครคอนโทรลเลอร์แบบแฟลช

2.5.4 การใช้งานเป็นพอร์ทอินพุต

ในการกำหนดให้เป็นพอร์ท อินพุตต้องเริ่มต้นด้วยการเขียนข้อมูล “1” มาที่แต่ละบิตของพอร์ท ที่ต้องการใช้งานเป็นอินพุต เพื่อหยุดการทำงานของเฟดที่ใช้ในการขับสัญญาณเอาต์พุตของบิตนั้น ๆ ทำให้ขาสัญญาณของพอร์ทเชื่อมต่อกับวงจรพูลอัพภายในโดยตรง ส่งผลให้ขาพอร์ทนั้นมีลอจิกเป็น 1 สามารถรับสัญญาณลอจิก 0 จากอุปกรณ์ภายนอกได้ง่าย สัญญาณข้อมูลจากอุปกรณ์ภายนอกจะถูกส่งเข้ามาแล้วเก็บไว้ในวงจรบัฟเฟอร์ภายในพอร์ท แล้วรอให้ชิพผู้อ่านค่าเข้าไป เมื่อเป็นเช่นนี้ อุปกรณ์ภายนอกที่เชื่อมต่อกับพอร์ทอินพุตของไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลชควรกำหนดให้ทำงานในสถานะลอจิก 0 จะดีและสะดวกที่สุด (ในปัจจุบันอุปกรณ์อินพุตที่เชื่อมต่อกับไมโครคอนโทรลเลอร์แทบทั้งหมดทำงานที่ลอจิก 0 อยู่แล้ว)

2.5.5 การใช้งานเป็นพอร์ทเอาต์พุต

โดยปกติแล้วขาพอร์ทจะกำหนดให้มีลักษณะเป็นเอาต์พุตอยู่แล้ว ดังนั้นจึงสามารถส่งข้อมูลออกไปได้อย่างง่ายดายกล่าวคือ เมื่อต้องการส่งข้อมูล 0 ออกไปทางเอาต์พุตก็ให้เขียนข้อมูล 0 ไปยังวงจรแลตช์ ซึ่งก็จะส่งผลต่อไปขับเฟดทำให้เฟดทำงาน ที่ขาพอร์ท ที่กำหนดให้ทำงานก็จะเกิดลอจิก 0 ขึ้น ในทางตรงข้ามหากต้องการส่งข้อมูล 1 ออกไป ก็ให้เขียนข้อมูล 1 ไปยังวงจรแลตช์ วงจรขับก็จะหยุดทำงาน ทำให้ที่ขาพอร์ทเชื่อมต่อกับวงจรพูลอัพภายในเกิดเป็นลอจิก 1 ที่ขาพอร์ทนั้น ซึ่งจะคล้ายกับการกำหนดให้เป็นขาอินพุตมาก เพียงแต่แตกต่างกันที่กระบวนการในการเคลื่อนย้ายข้อมูล โดยถ้าเป็นอินพุตจะมีสัญญาณมาอ่านข้อมูลที่บัฟเฟอร์ แต่ถ้าเป็นเอาต์พุตจะไม่มีการอ่านข้อมูลที่บัฟเฟอร์แต่อย่างใด เว้นแต่ในกรณีที่ต้องการตรวจสอบข้อมูลที่ส่งออกมาทางเอาต์พุต

เมื่อใช้งานพอร์ทของไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลชเป็นพอร์ทเอาต์พุต แต่ละขา (หรือแต่ละบิต) ของแต่ละพอร์ทมีความสามารถในการจ่ายกระแสหรือที่เรียกว่า กระแสซอร์ส(source current) สำหรับพอร์ท 0 จ่ายกระแสได้ 26 มิลลิแอมป์ ส่วน พอร์ท 1,2,3 จะจ่ายกระแสได้ 15 มิลลิแอมป์ และ ในกรณีที่ใช้งานทุกพอร์ท เอาต์พุตสามารถจ่ายกระแสรวมกันได้สูงสุด 71 มิลลิแอมป์ ดังนั้นในการใช้งานเป็นพอร์ทเอาต์พุตเพื่อไม่ให้เกิดปัญหาเกี่ยวกับความสามารถในการจ่ายกระแสจึงควรต่อวงจรบัฟเฟอร์ทางเอาต์พุตเพื่อช่วยในการขับกระแสอีกทางหนึ่ง

2.5.6 การอ่านค่าลอจิกจากพอร์ท

ในไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลชสามารถอ่านค่าลอจิกจากพอร์ทได้ 2 ลักษณะคือ อ่านค่าจากพอร์ทโดยตรง และอ่านจากวงจรแลตช์ของแต่ละพอร์ท

ในกรณีที่พอร์ทต่อกับขาเบสของทรานซิสเตอร์ชนิด NPN และขาอิมิตเตอร์ต่อลงกราวด์ หากมีการส่งข้อมูล 1 ไปยังทรานซิสเตอร์ จะทำให้ทรานซิสเตอร์ทำงานสถานะลอจิกที่ขาพอร์ทจะเป็น 0 เนื่องจากเมื่อทรานซิสเตอร์ทำงาน จะเสมือนว่าขาพอร์ทนั้นถูกต่อลงกราวด์ทำให้หากอ่านค่าลอจิกที่ขาพอร์ท จะได้ผลตรงกันข้ามกับที่ส่งออกมา แต่ถ้าหากทำงานอ่านค่าลอจิกที่วงจรแลตช์จะได้ค่าที่ตรงกับค่าที่

เมื่อเริ่มจ่ายไฟให้กับไมโครคอนโทรลเลอร์แบบเฟลช จะเกิดการรีเซ็ตในลักษณะที่เรียกว่า Power-on Reset ทำให้ซีพียูไปเริ่มต้นทำงานที่แอดเดรส 0000H ของหน่วยความจำโปรแกรม จังหวะการทำงานของซีพียูเป็นไปตามรูปแบบ โดยได้รับการกำหนดมาจากการรอบการทำงาน(machine cycle) ในรูปที่ 2.30 เป็นแผนภาพเวลาแสดงจังหวะการทำงานของไมโครคอนโทรลเลอร์ โดยใน 1 รอบการทำงาน จะแบ่งย่อยออกเป็น 6 สถานะ กำหนดชื่อเป็น S1-S6 ในแต่ละสถานะมีคาบเวลาเท่ากับ 2 คาบเวลาของสัญญาณนาฬิกา ถ้าสัญญาณนาฬิกามีความถี่ 12 เมกะเฮิร์ตซ์ จะมีคาบเวลาเท่ากับ 1 ไมโครวินาที คาบเวลาทั้งสองภายในหนึ่งสถานะ จะเรียกว่า เฟส 1 (phase 1) เฟส 2 (phase 2)

ในรูปที่ 2.30 (ก) และ (ข) จะเป็นการเอ็ชชีควิด์คำสั่งที่ใช้เวลา 1 ไชเคิล เริ่มต้นที่สถานะ 1 จะเป็นการอ่านคำสั่งดำเนินการ (Op code) อันเป็นกระบวนการแลตซ์ค่าของรหัสดำเนินการ ส่งไปให้รีจิสเตอร์คำสั่ง (Instruction Register:IR) การเฟตซ์ครั้งที่สองจะเกิดที่สถานะ 4 ภายในเมซซีนไชเคิลเดียวกัน ในกรณีที่คำสั่งไบต์เดียวการเฟตซ์ครั้งที่ 2 ภายในเมซซีนไชเคิลเดียวกันจะถูกตัดทิ้งไป ในคำสั่งที่มีการใช้เวลา 1 ไชเคิลจะสิ้นสุดการทำงานลงในสถานะ 6 ของเมซซีนไชเคิลเดียวกัน

ในกรณีที่คำสั่งใช้เวลา 2 ไชเคิล การทำงานของคำสั่งนั้นจะสิ้นสุดลงในสถานะ 6 ของเมซซีนไชเคิลที่สอง ดังนั้นในรูปที่ 2.30(ค) สำหรับในการกระทำคำสั่ง MOVX นี้ เนื่องจากซีพียูจะไปทำการติดต่อกับหน่วยความจำภายนอกดังแสดงในรูปที่ 2.30(ง) จะเห็นได้ว่า เวลาในการเอ็ชชีควิด์จะไม่ได้ขึ้นอยู่กับว่าทำการติดต่อกับหน่วยความจำภายในหรือภายนอก

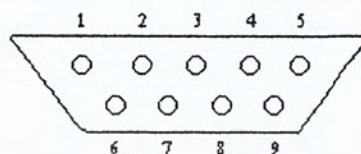
2.6 มาตรฐานพอร์ทอนุกรมแบบ RS-232

ในขณะที่การรับส่งข้อมูลแบบอนุกรมเป็นการรับส่งข้อมูลครั้งละ 1บิต แต่ก็สามารถรับส่งข้อมูลได้คราวละหลาย ๆ บิตได้ หากแต่จะต้องมีการตกลงกันระหว่างตัวส่งและตัวรับว่า จะรับส่งข้อมูลคราวละกี่บิต ตัวรับจะต้องรอข้อมูลมาให้ครบทุกบิตเสียก่อนจึงทำการประมวลผล ส่งผลให้การสื่อสารข้อมูลอนุกรมอาจมีความเร็วต่ำกว่าแบบขนาน ในด้านจำนวนสายสัญญาณการรับส่งข้อมูลแบบอนุกรมจะให้จำนวนสายที่น้อยกว่ามาก อย่างน้อยที่สุดใช้เพียง 2-3 เส้นเท่านั้น แต่อัตราเร็วในการรับส่งข้อมูลอาจต่ำกว่าแบบขนาน อย่างไรก็ตามการรับส่งข้อมูลแบบอนุกรมสามารถใช้สายสัญญาณที่มีความยาวมากกว่าแบบขนานทำให้ระยะทางในการสื่อสารข้อมูลแบบอนุกรมสามารถทำได้มากกว่า

มาตรฐานการเชื่อมต่อแบบอนุกรม RS -232 เป็นมาตรฐานอุตสาหกรรมที่ออกแบบมาเพื่อใช้ในการส่งข้อมูลอนุกรมแบบอะซิงโครนัส 2 ทิศทาง โดยมาตรฐาน RS -232 ในอดีตนั้นถูกออกแบบมาเพื่อการส่งผ่านข้อมูลจากคอมพิวเตอร์ไปยังโมเด็มเพียงอย่างเดียว เพื่อที่จะนำข้อมูลจากโมเด็มนี้สื่อสารผ่านสายโทรศัพท์ไปยังคอมพิวเตอร์อีกชุดซึ่งอยู่ห่างไกลกัน โดยคณะกรรมการที่เรียกว่า สมาคมอุตสาหกรรมอิเล็กทรอนิกส์ (Electronic Industries Association : EIA) ได้วางมาตรฐานที่มีชื่อเรียกกันว่า EIA RS-232 มาตรฐานนี้ในช่วงแรกจะใช้คอนเน็กเตอร์เป็นแบบ DB-25 โดยกำหนดความยาวสูงสุดของสายสัญญาณไว้ที่ 50 ฟุต มีระดับสัญญาณตั้งแต่ -3 ถึง -12 V แสดงว่ามีข้อมูล(Mark) และ +3 ถึง +12 V แสดงว่าเป็นช่องว่าง (Space)

คอนเน็กเตอร์สำหรับพอร์ท RS-232

มาตรฐานการเชื่อมต่อแบบ RS-232 จะใช้คอนเน็กเตอร์แบบ DB-25 ตัวผู้หรือ ตัวผู้ซึ่งคอนเน็กเตอร์แบบ DB-25 จะมีขาต่อใช้งานเพียง 9 เส้นเช่นเดียวกับคอนเน็กเตอร์แบบ DB-9 เนื่องจากขาอื่น ๆ ที่เคยใช้งานในอดีต ปัจจุบันมีการใช้งานไม่มากนัก จึงถูกยกเลิกไป โดยแสดงรูปร่างและตำแหน่งขาต่าง ๆ ดังรูป



รูปที่ 2.31 รูปร่าง และตำแหน่งต่าง ๆ ของคอนเน็กเตอร์สำหรับพอร์ท RS-232

ตารางที่ 2.4 ชื่อและชนิดของสายสัญญาณของคอนเน็กเตอร์ DB-9 สำหรับพอร์ท RS-232

คอนเน็กเตอร์ DB-9	ชื่อของสายสัญญาณ	ชนิดของสายสัญญาณ
1	Data Carrier Detect : DCD	อินพุต
2	Received Data : RxD	อินพุต
3	Transmitted Data : TxD	เอาต์พุต
4	Data Terminal Ready : DTR	เอาต์พุต
5	Signal Ground : GND	-
6	Data Set Ready : DSR	อินพุต
7	Request To Send : RTS	เอาต์พุต
8	Clear To Send : CTS	อินพุต
9	Ring Indicator : RI	อินพุต

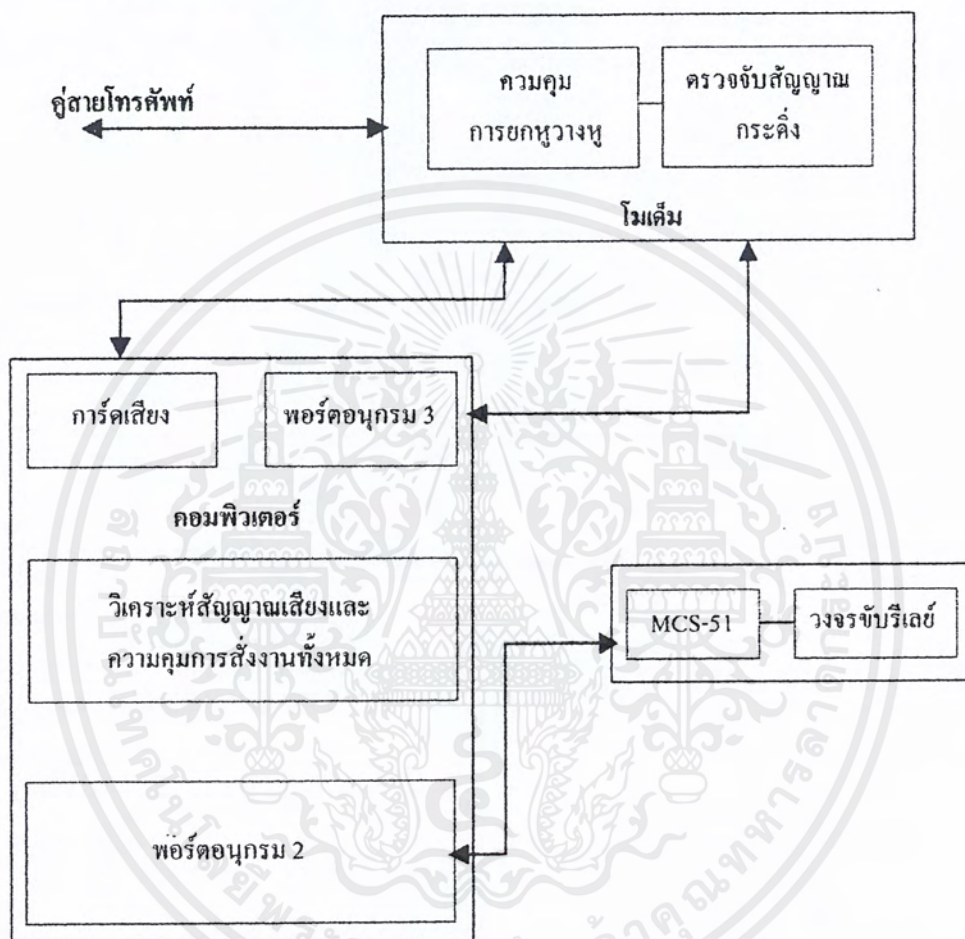
บทที่ 3

การคำนวณและการสร้าง

โครงการนี้สามารถแบ่งการทำงานออกเป็นส่วนหลักๆ ได้ 3 ส่วน ได้แก่

3.1 ส่วนของการตอบรับอัตโนมัติ

3.1.1 การตอบรับอัตโนมัติโดยใช้โมเด็ม



รูปที่ 3.1 แสดงบล็อกไดอะแกรมการทำงานของเครื่องใช้ไฟฟ้าผ่านทางโทรศัพท์โดยการใช้เสียง

3.2 ส่วนของโปรแกรมติดต่อกับผู้ใช้งาน สามารถแบ่งเป็นส่วนย่อยได้คือ

3.2.1 การตั้งค่าเริ่มต้นของโปรแกรม และการทำงานสั่งเครื่องใช้ไฟฟ้าในโหมดแมน

นวล(Manual)

3.2.2 โหมดแสดงรูปและการอ่านข้อมูลไฟล์ .wav

3.2.3 โหมดการหาค่าพลังงานเสียง

3.2.4 โหมดการตัดคำ

3.2.5 โหมดการหาค่าสัมประสิทธิ์การประมาณเชิงเส้น

3.2.6 โหมดการหาค่าสัมประสิทธิ์เซปสตรัม

3.2.7 โหมดการคัสดินใจ (Pattern Matching)

3.3 ส่วนเชื่อมตอกับคอมพิวเตอร์และความคุมการเปิดปิดเครื่องใช้ไฟฟ้า

3.3.1 ส่วนของการต่อพอร์ทและไมโครคอนโทรลเลอร์

3.3.2 ส่วนของวงจรขั้วรีเลย์

3.1 ส่วนของการตอบรับอัตโนมัติ

3.1.1 การตอบรับอัตโนมัติโดยการใช้โมเด็ม

เมื่อมีสัญญาณกระดิ่งจากชุมสายโทรศัพท์ โมเด็มทำการส่งข้อมูลผ่านพอร์ทอนุกรม RS-232 โดยส่งผ่านทางขา RI (ขาที่ 22 ของ DB-25) เพื่อแจ้งให้ทราบว่ามีสัญญาณกระดิ่งเข้ามา จากนั้นก็ทำการโปรแกรมเพื่อสั่งให้โมเด็มทำการรับสาย ซึ่งจะใช้คำสั่ง AT Command ซึ่งมีรูปแบบดังนี้

Hn Hang Up
H0 วางหู
H1 ยกหู

ขั้นตอนในการทำการโปรแกรม

1. ทำการกำหนด Handle ให้กับ Communication Resource ด้วย Function Createfile

```
hComm = CreateFile("COM5",GENERIC_READ|GENERIC_WRITE
,0,0,OPEN_EXISTING,FILE_FLAG_OVERLAPPED,0);
if (hComm == INVALID_HANDLE_VALUE)
    // Error Opening Port ; Abort
```

2. กำหนดค่าพารามิเตอร์ต่างๆ ในการติดต่อกับ Communication Resource

```
dcB.BaudRate=9600;           //   กำหนดค่า BaudRate
dcB.ByteSize=8;              //   กำหนดขนาดของข้อมูล
dcB.Parity=NOPARITY;        //   กำหนด พาริตีบิต
dcB.StopBits=ONESTOPBIT;    //   กำหนด StopBit
```

3. ทำการเขียนหรืออ่านจาก Communication Resource ด้วย Functon ReadFile() หรือ WriteFile()

```
WriteFile(hFile,lpBuffer,nNumberOfBytesToWrite,lpNumberOfBytesWritten,lpOverlapped);
ReadFile(hFile,lpBuffer,nNumberOfBytesToRead,lpNumberOfBytesRead,lpOverlapped);
```

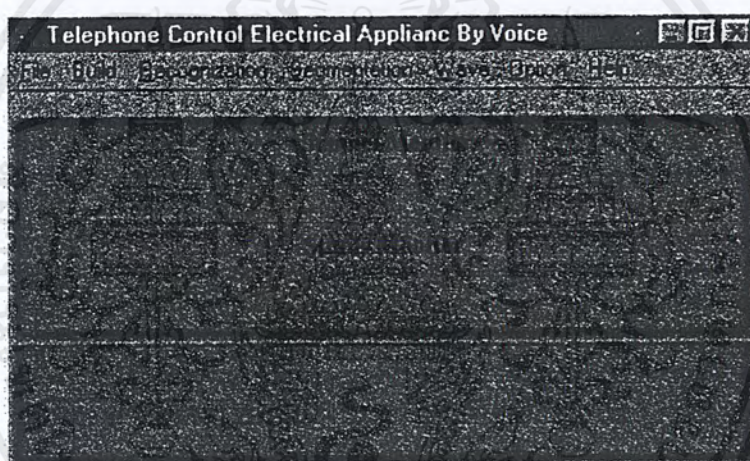
4. แสดงผลและตอบสนองเหตุการณ์ที่เกิดขึ้นในการติดต่อกับ Communication Resource เช่น ตรวจสอบสัญญาณกระดิ่งได้ แล้วทำการตอบสนองโดยการยกหู

```
// Create an Event object For use in WaitCommEvent
o.hEvent = CreateEvent( NULL, FALSE, FALSE, NULL);
assert(o.hEvent);
if( WaitCommEvent(hComm, &dwEvtMask, &o))
{
    if(dwEvtMask & EV_RING)
    {
        .....// ทำการตอบสนองสัญญาณกระดิ่ง
    }
}
}
```

3.2 ส่วนของโปรแกรมการติดต่อกับผู้ใช้งาน

3.2.1 การตั้งค่าเริ่มต้นของโปรแกรมและการทำงานสั่งเครื่องใช้ไฟฟ้าในโหมดแมนนวล

โปรแกรมการทำงานของจะแบ่งเป็น 2 โหมดหลักๆ คือ โหมดอัตโนมัติและโหมดแมนนวลซึ่งจะรูปแบบดังรูปที่ 3.2



รูปที่ 3.2 แสดงรูปโปรแกรม TCEAV (Telephone Control Electrical Appliance by Voice)

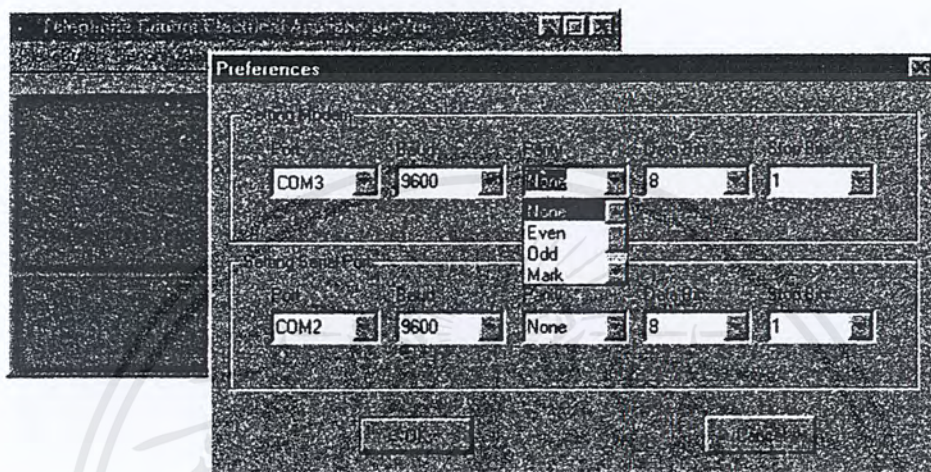
โหมดอัตโนมัติ ผู้ใช้งานจะต้องโทรศัพท์เข้ามาเพื่อทำการสั่งงานเปิดปิดเครื่องใช้ไฟฟ้าด้วยเสียง โหมดแมนนวล ผู้ใช้เป็นผู้ทำการควบคุมการเปิดปิดเครื่องใช้ไฟฟ้าโดยตรง

ก่อนการทำงานของโปรแกรมจะต้องทำการตั้งค่าเริ่มต้นให้กับโปรแกรม เป็นการกำหนดค่าพารามิเตอร์ที่ใช้ทำการติดต่อสื่อสารระหว่างคอมพิวเตอร์กับโมเด็มและบอร์ดควบคุม ซึ่งจะมีขั้นตอนและรายละเอียดและขั้นตอนดังนี้

1. เริ่มจากการการคลิกเมาท์ไป Option เมนู แล้ว เลือก Preference จะปรากฏไดอะล็อกใหม่
2. ภายในไดอะล็อกจะแยกเป็นได้ 2 ส่วนได้แก่ การกำหนดค่าเริ่มต้นกับพอร์ทอนุกรมให้กับโมเด็ม และการกำหนดค่าเริ่มต้นให้กับพอร์ทอนุกรมที่ใช้ติดต่อกับบอร์ดควบคุม ซึ่งจะมีรายละเอียดเหมือนกัน ประกอบด้วยค่าต่างๆดังนี้ ดังรูปที่ 3.3

2.1 หมายเลขพอร์ทอนุกรม

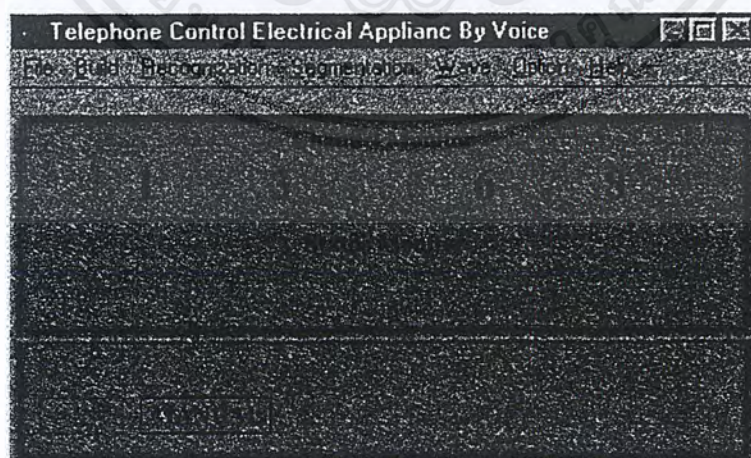
- 2.2 BaudRate
- 2.3 รูปแบบของพาริตีบิต
- 2.4 จำนวนบิตข้อมูล
- 2.5 ขนาดของ StopBit
3. ให้ทำการกำหนดค่าเริ่มต้นต่างๆให้เหมาะสม จากนั้นทำการกดปุ่ม OK ถ้าต้องการใช้ค่าที่กำหนด ถ้าต้องการยกเลิกให้กดปุ่ม Cancel



รูปที่ 3.3 แสดงการกำหนดค่าเริ่มต้นให้กับโปรแกรม TCEAV

การทำงานของโหมดแมนนวล

1. เริ่มจากการคลิกเมาส์ไปที่อักษร MANUAL โปรแกรมจะเข้าทำงานในโหมดแมนนวล
2. ทำการเลือกช่องที่ต้องการทำการเปิดหรือปิด ถ้าตัวเลขประจำช่องเป็นตัวเข้มแสดงว่าเครื่องใช้ไฟฟ้านั้นเปิด ถ้าเป็นตัวเลขจางแสดงว่าเครื่องใช้ไฟฟ้านั้นปิด ดังรูปที่ 3.4



รูปที่ 3.4 แสดงการทำงานของโหมดแมนนวลของโปรแกรม TCEAV

3.2.2 โหมดแสดงรูปและทำการอ่านข้อมูลไฟล์ .wav

โปรแกรมจะแสดงรูปสัญญาณเสียงจากไฟล์เสียง และทำการบันทึกเป็นไฟล์ .dat เพื่อนำไปทำการคำนวณผลต่อไป

วิธีการใช้งาน

1. เลือกรายการจากเมนูบาร์ไปที่ Wave -> Edit Wave
2. คลิกปุ่ม Open ทำการเลือกไฟล์ที่ต้องการ
3. จากนั้นโปรแกรมจะแสดงลักษณะของสัญญาณที่ได้จากไฟล์นั้น
4. ทำการจัดเก็บไฟล์เป็นนามสกุล .dat



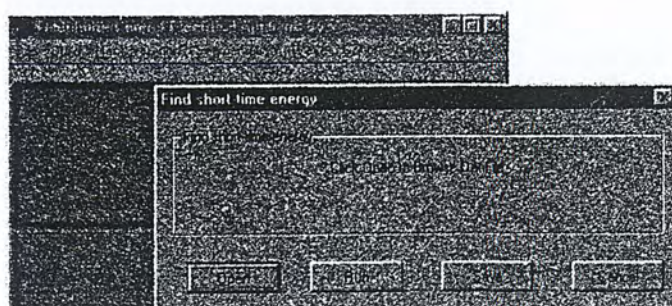
รูปที่ 3.5 แสดงการทำงานของโหมด Edit Wave

3.2.3 โหมดการหาค่าพลังงานของเสียง

โปรแกรมจะทำการคำนวณหาค่าพลังงานของเสียงที่ได้จากไฟล์ .dat

วิธีการใช้งาน

1. เลือกรายการจากเมนูบาร์ไปที่ Segmentation -> Energy
2. คลิกที่ปุ่ม Open ทำการเลือกไฟล์ .dat ที่ต้องการ
3. คลิกที่ปุ่ม Run จากนั้นโปรแกรมจะคำนวณหาค่าพลังงานของเสียงที่ได้จากไฟล์นั้น
5. ทำการคลิกที่ปุ่ม Save เพื่อจัดเก็บไฟล์เป็นนามสกุล .eng



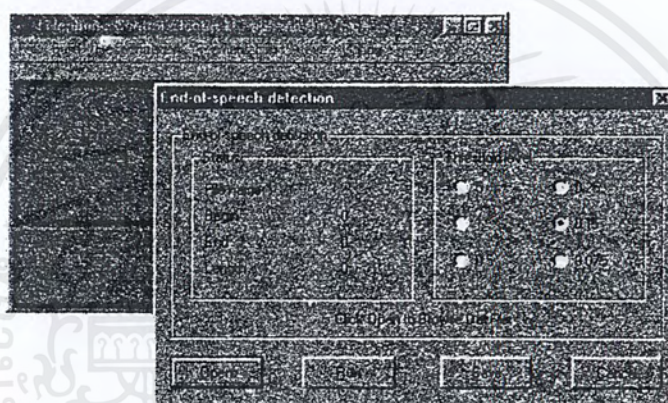
รูปที่ 3.6 แสดงการทำงานในโหมด Energy

3.2.4 โหมดการตัดคำ

โปรแกรมจะทำแยกเสียงพูดออกจากเสียงเงียบโดยทำการคำนวณหาจากค่าพลังงานที่ได้จากไฟล์ .eng

วิธีการใช้งาน

1. เลือกรายการจากเมนูบาร์ไปที่ Segmentation -> End-of-Speech
2. คลิกที่ปุ่ม Open ทำการเลือกไฟล์ .eng ที่ต้องการ
3. ทำการเลือกค่าของ Threshold level
4. คลิกที่ปุ่ม Run จากนั้นโปรแกรมจะคำนวณหาจุดเริ่มต้นและจุดสิ้นสุดของคำ และแสดงค่าความยาวของคำที่คำนวณได้
5. ทำการคลิกที่ปุ่ม Save เพื่อจัดเก็บไฟล์เป็นนามสกุล .end



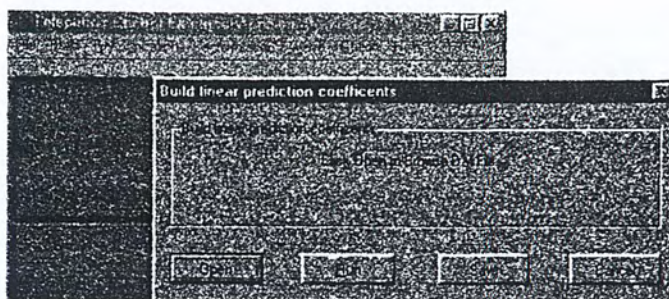
รูปที่ 3.7 แสดงการทำงานในโหมด End-of-Speech

3.2.5 โหมดการหาค่าสัมประสิทธิ์การประมาณเชิงเส้น

โปรแกรมจะทำการคำนวณหาค่าสัมประสิทธิ์การประมาณเชิงเส้นของเสียงที่ได้จากไฟล์ .end

วิธีการใช้งาน

1. เลือกรายการจากเมนูบาร์ไปที่ Build -> LPC
2. คลิกที่ปุ่ม Open ทำการเลือกไฟล์ .end ที่ต้องการ
3. คลิกที่ปุ่ม Run จากนั้นโปรแกรมจะคำนวณหาค่าสัมประสิทธิ์การประมาณเชิงเส้นของเสียงที่ได้จากไฟล์นั้น
4. ทำการคลิกที่ปุ่ม Save เพื่อจัดเก็บไฟล์เป็นนามสกุล .lpc



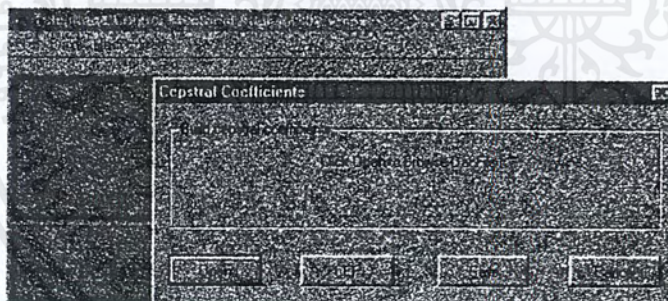
รูปที่ 3.8 แสดงการทำงานในโหมด LPC

3.2.6 โหมดการหาค่าสัมประสิทธิ์เซปสตรัม

โปรแกรมจะทำการคำนวณหาค่าสัมประสิทธิ์เซปสตรัมของเสียงที่ได้จากไฟล์ .end

วิธีการใช้งาน

1. เลือกรายการจากเมนูบาร์ไปที่ Build -> CEP
2. คลิกที่ปุ่ม Open ทำการเลือกไฟล์ .end ที่ต้องการ
3. คลิกที่ปุ่ม Run จากนั้นโปรแกรมจะคำนวณหาค่าสัมประสิทธิ์เซปสตรัมของเสียงที่ได้จากไฟล์นั้น
4. ทำการคลิกที่ปุ่ม Save เพื่อจัดเก็บไฟล์เป็นนามสกุล .cep



รูปที่ 3.9 แสดงการทำงานในโหมด CEP

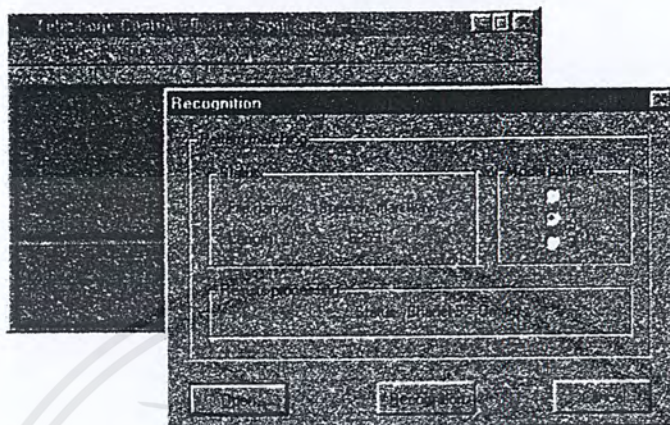
3.2.7 โหมดการตัดสินใจ (Pattern Matching)

โปรแกรมจะทำการตัดสินใจเลือกคำที่ได้จากการคำนวณค่าสัมประสิทธิ์เซปสตรัมโดยใช้วิธีการของ DTW ในการพิจารณาเลือกกว่าคำที่รับเข้ามานั้นตรงกับคำใดในสัญญาณเสียงอ้างอิงที่เก็บไว้

วิธีการใช้งาน

1. เลือกรายการจากเมนูบาร์ไปที่ Recognition -> Manual
2. คลิกที่ปุ่ม Open ทำการเลือกไฟล์ .wav ที่ต้องการ
3. ทำการเลือกจำนวนแพทเทิร์น

4. คลิที่ปุ่ม Run จากนั้นโปรแกรมจะคำนวณหาค่าพารามิเตอร์ต่าง ๆ พร้อมทั้งทำการตัดสินใจในการเลือกว่าเป็นคำใด ถ้าเสียงนั้นมีพารามิเตอร์ตรงใกล้เคียงกับค่าพารามิเตอร์ของเสียงอ้างอิงที่เก็บไว้ ก็จะแสดงผลของคำส่งคำนั้น แต่ถ้าไม่ตรงตามพารามิเตอร์เสียงอ้างอิง ก็จะมีการสั่งให้ป้อนสัญญาณเสียงใหม่เข้าไป



รูปที่ 3.10 แสดงการทำงานใน โหมดการตัดสินใจ

3.3 ส่วนเชื่อมต่อกับคอมพิวเตอร์และความคุมการเปิดปิดเครื่องใช้ไฟฟ้า

ซึ่งสามารถแบ่งออกได้อีก 2 ส่วนดังนี้

3.3.1 ส่วนของการต่อพอร์ทและไมโครคอนโทรลเลอร์

3.3.2 ส่วนของวงจรขับรีเลย์

3.3.1 ส่วนของการต่อพอร์ทและไมโครคอนโทรลเลอร์

- ส่วนของการต่อพอร์ท

พอร์ทอนุกรมนอกจากจะมีขาสำหรับการรับและการส่งข้อมูลปกติแล้วยังมีขาที่ออกแบบไว้สำหรับควบคุมการไหลของข้อมูลอีกหลายตำแหน่งด้วยกัน โดยแบ่งเป็น 2 ประเภท คือ ขาที่ทำหน้าที่เป็นเอาต์พุต และขาที่ทำหน้าที่เป็นอินพุตของพอร์ทอนุกรม

ขาที่ทำหน้าที่เป็นเอาต์พุต ได้แก่ ขา DTR , RTS และ TxD โดยรีจิสเตอร์ที่ทำหน้าที่ควบคุมขาเหล่านี้คือรีจิสเตอร์ควบคุมโมเด็ม (MRC) โดยมีแอดเดรสอยู่ถัดจากรีจิสเตอร์หลักของพอร์ทอนุกรม 4 ตำแหน่ง รีจิสเตอร์ควบคุมการทำงานของขา DTR จะอยู่ที่ตำแหน่งบิต 0 ส่วนขา RTS จะอยู่ที่ตำแหน่งบิต 1 ส่วนขา TxD เป็นขาปกติในการส่งข้อมูล ดังนั้นจึงมีแอดเดรสอยู่ที่แอดเดรสของรีจิสเตอร์หลักระดับแรงดันที่ใช้งานสำหรับพอร์ทอนุกรม RS-232

มาตรฐานการสื่อสารข้อมูลอนุกรม ได้ระบุช่วงระดับแรงดันสำหรับการทำงานของพอร์ทอนุกรมไว้ว่า ที่ลอจิก “0” จะมีระดับสัญญาณ +3 ถึง +15 V ส่วนลอจิก “1” จะมีระดับสัญญาณ -3 ถึง -15 V ระดับสัญญาณนี้ทำให้ไม่สามารถที่จะนำขาเอาต์พุตใด ๆ ต่อเข้ากับลอจิกเกตเพื่อใช้งานได้โดย

ตรง จะต้องผ่านวงจรเพื่อเปลี่ยนระดับแรงดันเสียก่อน โดยปกติจะใช้ไอซีจำพวก RS-232 transceiver ที่นิยมมากคือ MAX232 หรือ ICL232 ไอซีในกลุ่มนี้จะทำหน้าที่แปลงระดับแรงดันของ RS-232 ให้อยู่ในระดับที่ที่แอล โดยลอจิก “0” ซึ่งเดิมมีระดับสัญญาณ +3 ถึง +15 จะถูกแปลงเป็น 0 V ส่วนลอจิก “1” ซึ่งมีระดับ -3 ถึง -15 V จะแปลงเป็น +5 V ทั้งนี้เพื่อให้สามารถเชื่อมต่อกับอุปกรณ์ดิจิทัลอื่นที่ใช้ระดับแรงดันที่ที่แอลได้

- พอร์ตสื่อสารอนุกรมของไมโครคอนโทรลเลอร์

พอร์ตสื่อสารอนุกรมมีโครงสร้างการทำงานในแบบที่เรียกว่าฟูลดูเพล็กซ์ (Full Duplex) ในการรับและส่งข้อมูลอนุกรมได้ในเวลาเดียวกัน โดยทางด้านส่งใช้ขา TxD (พอร์ต 3.1) ทางด้านรับใช้ขา RxD (พอร์ต 3.0) SBUF ใช้เป็นบัฟเฟอร์สำหรับรับและส่งข้อมูลอนุกรม

พอร์ตสื่อสารอนุกรมสามารถโปรแกรมการทำงานได้หลายโหมดด้วยกันโดยเลือกที่บิต SM1 และ SM0 ซึ่งอยู่ในรีจิสเตอร์ควบคุม SCON

ในการออกแบบจะใช้การรับข้อมูลอนุกรมโหมด 1 โดยมีความเร็ว 9600 bps ซึ่งตัวไมโครคอนโทรลเลอร์จะทำการรับข้อมูลที่ส่งมาและทำการเปรียบเทียบ โดยใช้คำสั่ง CJNE(Compare and Jump if Not Equal) คำสั่งนี้จะทำให้ซีพียูเกิดการกระโดดไปยังแอดเดรสปลายทางเมื่อข้อมูลที่นำมาเปรียบเทียบไม่เท่ากัน

รูปแบบของคำสั่งเป็นดังนี้

CJNE A,#data,rel

การทำงานของคำสั่ง : ซีพียูกระโดดไปยังแอดเดรสปลายทางตามค่าสัมพัทธ์(rel) เมื่อค่าของรีจิสเตอร์ A ไม่เท่ากับค่าของ data

จำนวนไบต์ : 3

จำนวนไชนเคิล : 2

ตัวอย่าง : CH_1 : CJNE A,#41H,OFF_CH1 ;จะเกิดการกระโดดไปยังตำแหน่ง OFF_CH1

;ค่าของรีจิสเตอร์ A ไม่เท่ากับข้อมูล #01H

SETB P0.0

;ถ้าค่าของรีจิสเตอร์ A เท่ากับข้อมูล #01H

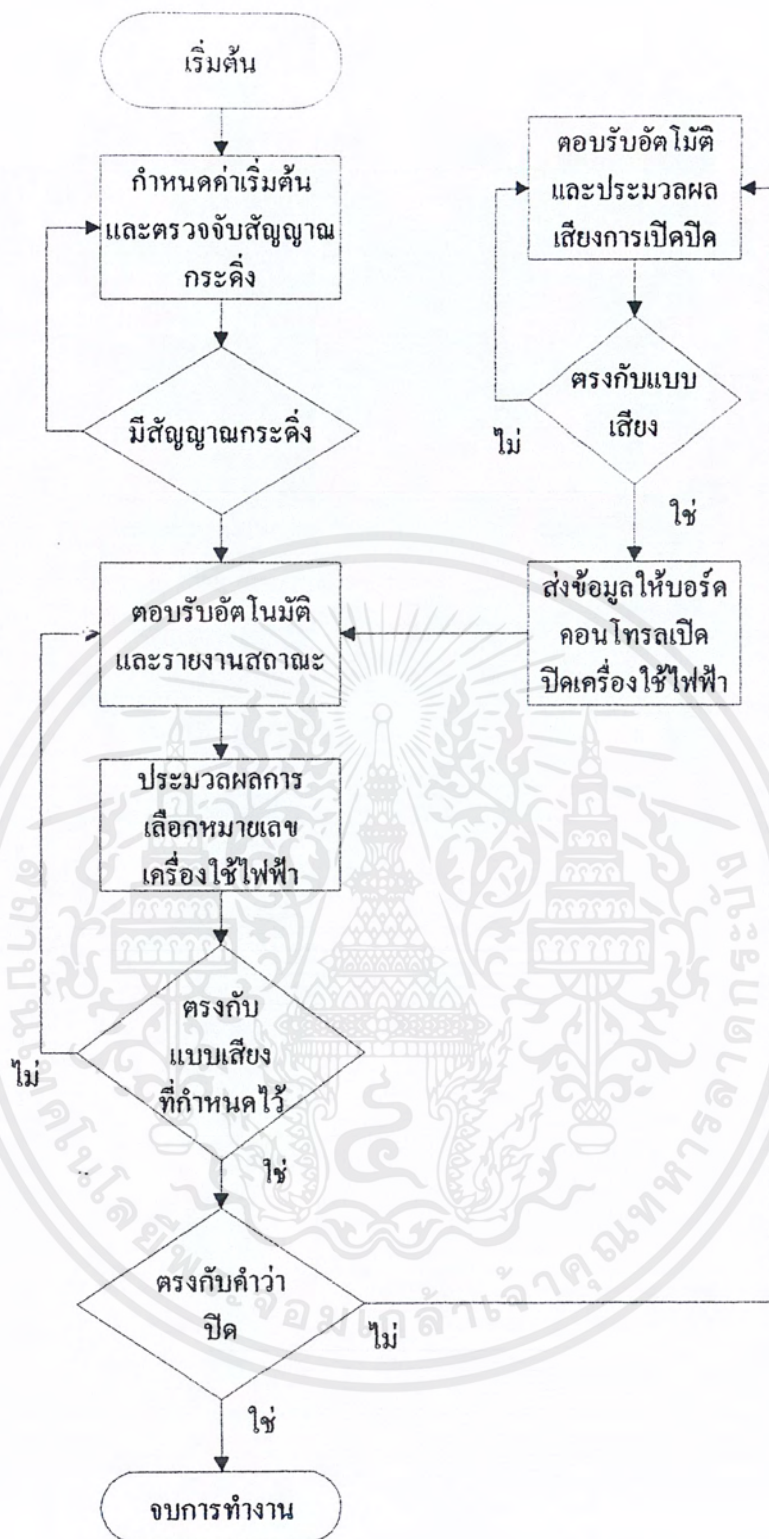
;จะมีการ SET ที่พอร์ต 0 บิต 0

RET

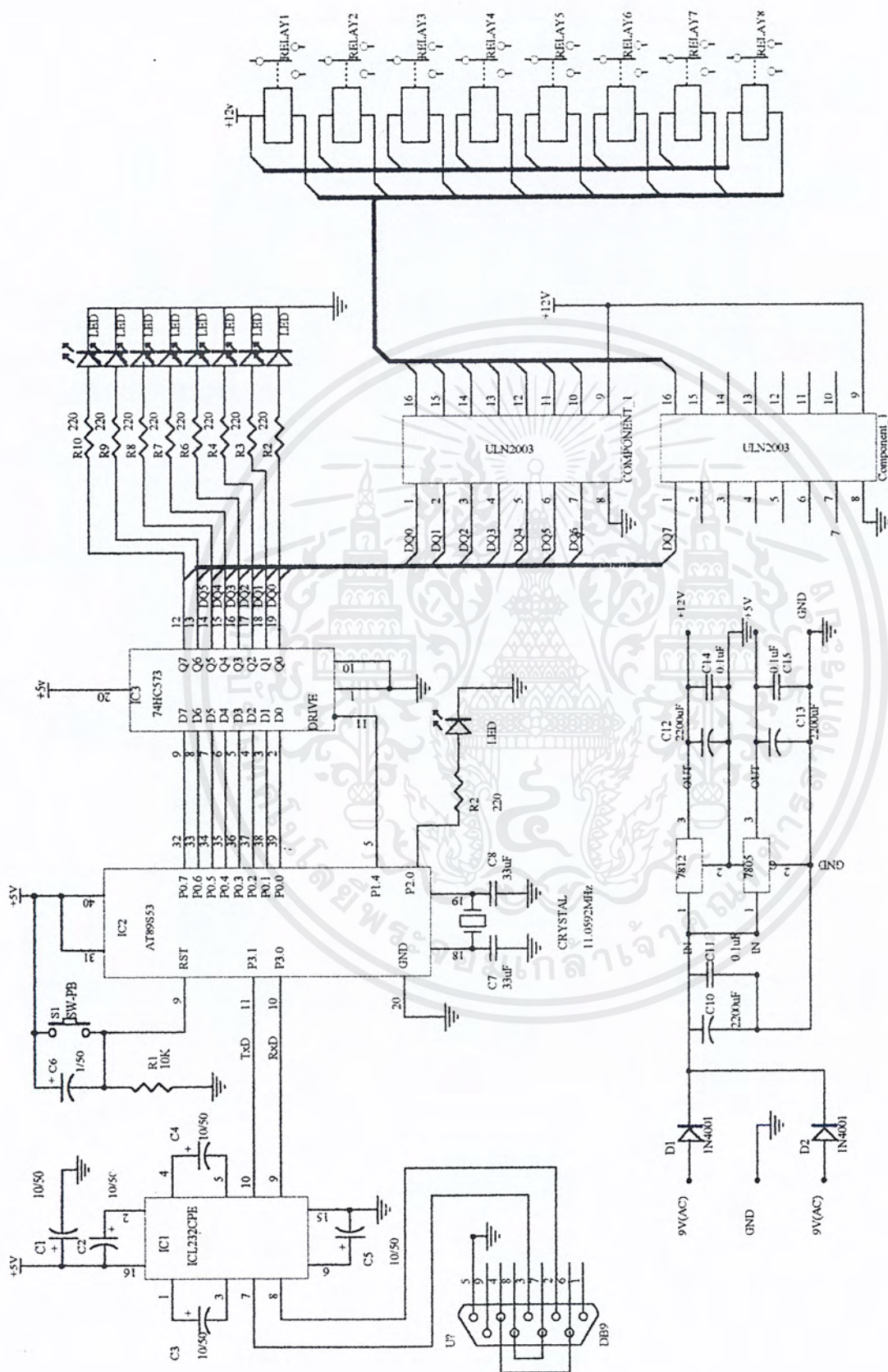
OFF_CH1 : CJNE A,#51H,CH_2
CRL P0.0
RET

3.3.2 ภาควงจรขั้วรีเลย์ (Relay)

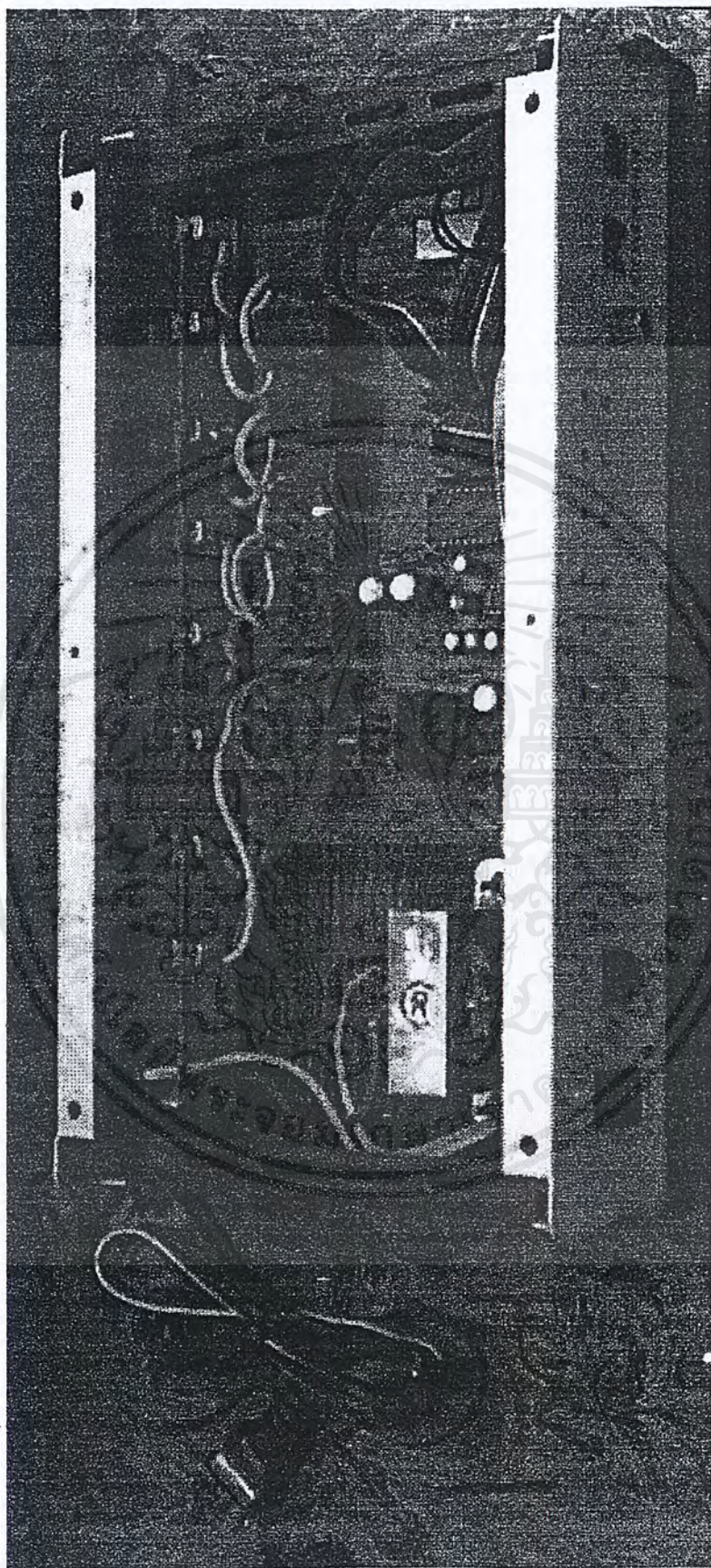
วงจรขั้วรีเลย์ในโครงงานใช้ทั้งหมด 8 ช่องสัญญาณ โดยใช้ไอซี ULN2003 การควบคุมรีเลย์ให้ทำการเปิด-ปิด โดยหัวใจสำคัญของวงจรขั้วรีเลย์นี้คือ ULN2003 ซึ่งเป็นอินเวอร์เตอร์ไคเวอร์ 7 ตัว ขาอินพุตทั้งหมดจะต่อกับเอาต์พุตของไมโครคอนโทรลเลอร์เพื่อทำการควบคุมรีเลย์ทั้งหมด 8 ช่องสัญญาณ



Flow chart แสดงผลการทำงานโดยรวม



รูปที่ 3.12 วงจรส่วนควบคุมอุปกรณ์เครื่องใช้ไฟฟ้า



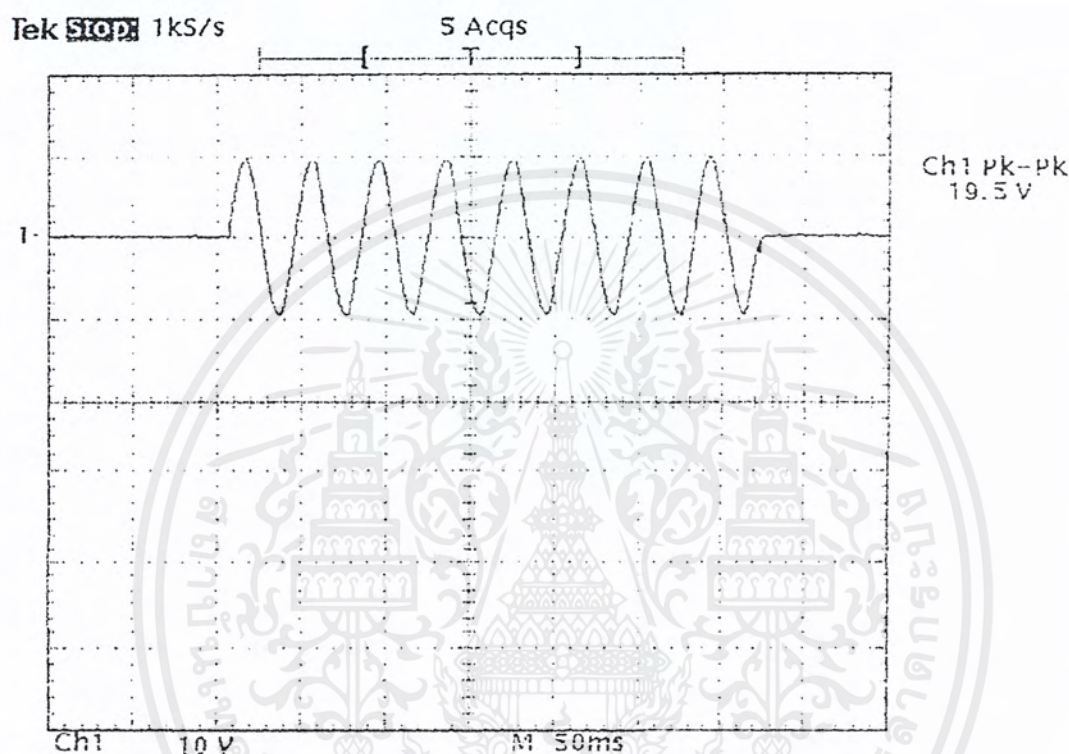
รูปที่ 3.13 เครื่องควบคุมเครื่องใช้ไฟฟ้าด้วยเสียงพูดผ่านตู้สายโทรศัพท์

บทที่ 4

การทดลองและผลการทดลอง

4.1 การทดลองในส่วนของการตอบรับโทรศัพท์

ในส่วนของการตอบรับอัตโนมัติ จะใช้โมเด็มในการตอบรับอัตโนมัติซึ่งสัญญาณกระดิ่งจะผ่านตามสายเข้าสู่โมเด็ม ซึ่งลักษณะของสัญญาณที่วัดได้จะแสดงในรูปที่ 4.1



รูปที่ 4.1 แสดงสัญญาณกระดิ่งที่อินพุตของโมเด็ม

จากรูปที่ 4.1 สัญญาณที่วัดได้จะมีลักษณะเป็นสัญญาณรูปไซน์ มีขนาดแอมพลิจูดประมาณ 195 โวลต์พีคทูพีค (จากรูป ตั้ง Probe ไว้ที่ $\times 10$) มีความถี่ประมาณ 500 เฮิรตซ์ ซึ่งค่าที่วัดได้ใกล้เคียงกับค่ามาตรฐานที่กำหนดไว้ คือ แรงดันประมาณ 100-200 โวลต์พีคทูพีค ความถี่ 425 เฮิรตซ์ จากนั้นโมเด็มจะทำส่งข้อมูลไปยังคอมพิวเตอร์ และทำการการยกหูโทรศัพท์เมื่อได้รับสัญญาณกระดิ่งครบตามจำนวนที่กำหนดไว้ในส่วนของโปรแกรม

4.2 ผลการทดลองในส่วนของการเชื่อมต่อบอร์ดควบคุมกับคอมพิวเตอร์

เมื่อทำการเชื่อมต่อบอร์ดควบคุมกับคอมพิวเตอร์ แล้วทำการสั่งการเปิดปิดเครื่องใช้ไฟฟ้าผ่านโปรแกรม TCEAV โดยทดลองในโหมดแมนนวลได้ผลการทดลองเป็นไปตามตาราง

ตารางที่ 4.1 แสดงผลการทดลองส่วนบอร์ดควบคุม

คำสั่ง	ข้อมูลที่ส่ง	สถานะเครื่องใช้ไฟฟ้า
กลุ่ม 1	41H	ช่องสัญญาณที่ 1 ทำการเปิด
	51H	ช่องสัญญาณที่ 1 ทำการปิด
กลุ่ม 2	42H	ช่องสัญญาณที่ 2 ทำการเปิด
	52H	ช่องสัญญาณที่ 2 ทำการปิด
กลุ่ม 3	43H	ช่องสัญญาณที่ 3 ทำการเปิด
	53H	ช่องสัญญาณที่ 3 ทำการปิด
กลุ่ม 4	44H	ช่องสัญญาณที่ 4 ทำการเปิด
	54H	ช่องสัญญาณที่ 4 ทำการปิด
กลุ่ม 5	45H	ช่องสัญญาณที่ 5 ทำการเปิด
	55H	ช่องสัญญาณที่ 5 ทำการปิด
กลุ่ม 6	46H	ช่องสัญญาณที่ 6 ทำการเปิด
	56H	ช่องสัญญาณที่ 6 ทำการปิด
กลุ่ม 7	47H	ช่องสัญญาณที่ 7 ทำการเปิด
	57H	ช่องสัญญาณที่ 7 ทำการปิด
กลุ่ม 8	48H	ช่องสัญญาณที่ 8 ทำการเปิด
	58H	ช่องสัญญาณที่ 8 ทำการปิด

4.3 ผลการทดลองในส่วนของโปรแกรม

เมื่อมีสัญญาณกระดิ่งเข้ามาโปรแกรมสั่งให้โมเด็มทำการรับหู ก็จะมีเสียงตอบรับเข้าสู่ขั้นตอนการรอร์รับคำสั่ง โปรแกรมทำการสั่งงานให้การคัดเสียงบันทึกเสียงในรูปแบบของไฟล์ .wav (wave file) ลักษณะรูปแบบของการบันทึกเป็นรูปแบบของการเข้ารหัสพีซีเอ็ม(pulse code modulation) จากนั้นหาพลังงานของเสียงที่บันทึกได้เพื่อทำแยกเสียงพูดออกจากเสียงเงียบ เมื่อได้เสียงที่ต้องการแล้วทำการคำนวณหาค่าสัมประสิทธิ์การประมาณเชิงเส้นและสัมประสิทธิ์เซปสตรัม จากนั้นก็จะนำมาทำการเปรียบเทียบกับโมเดลเสียงที่บันทึกไว้แล้วว่าเป็นเสียงใด

4.3.1 กราฟแสดงผลการคำนวณที่ได้จากโปรแกรม

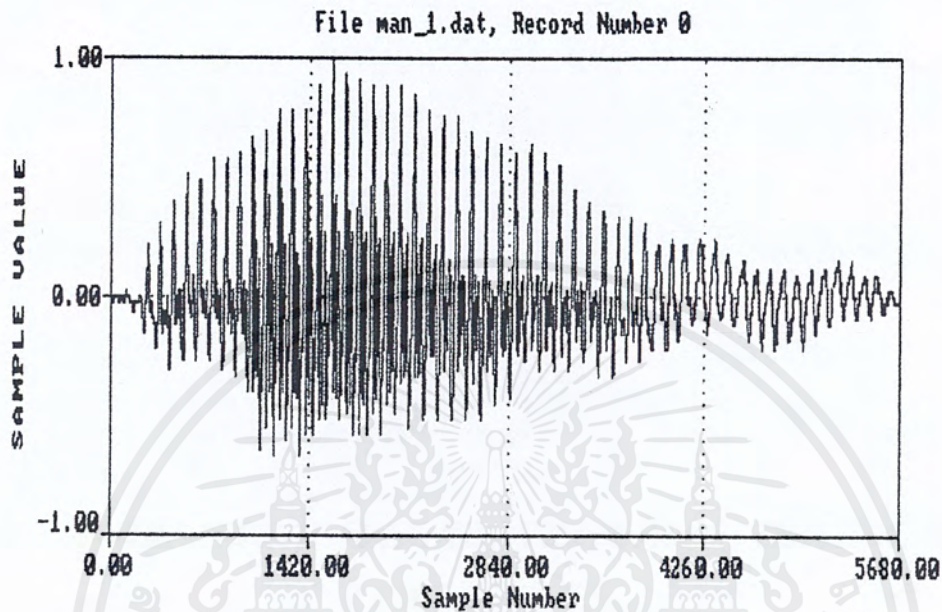
กราฟที่แสดงผลจากทำงานของ โปรแกรมมีอยู่ 4 รูปแบบคือ

1. กราฟแสดงรูปสัญญาณเสียง(ไฟล์ .dat) ซึ่งเป็นกราฟที่แสดงรูปสัญญาณเสียงที่บันทึกได้ โดยแกน x แสดงแอมพลิจูดของสัญญาณเสียง แกน y แสดงลำดับของสัญญาณซึ่งใช้ความถี่ในการชักตัวอย่าง 11,025 Hz ความละเอียด 8 บิต

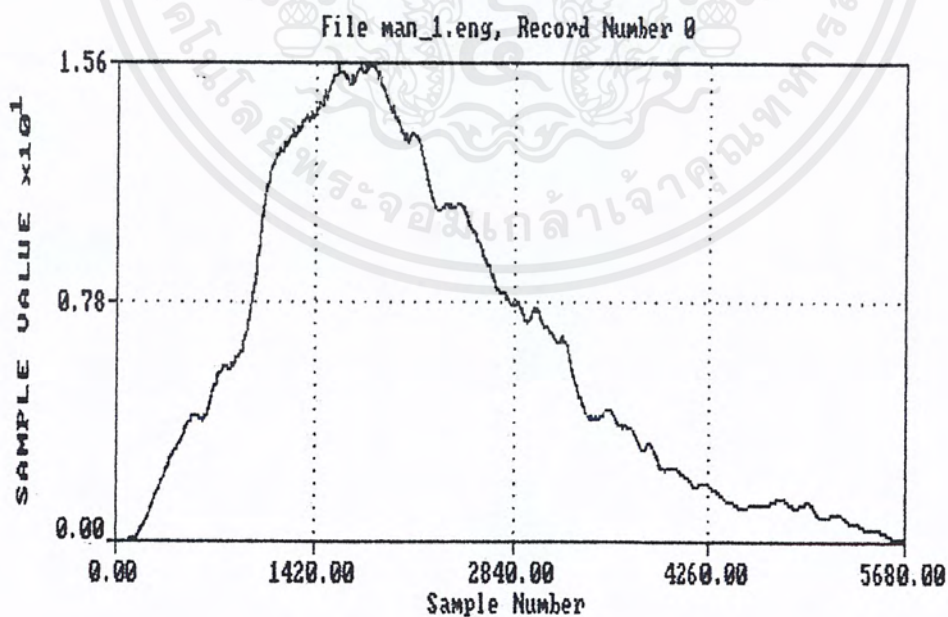
2. กราฟแสดงพลังงานของเสียง(ไฟล์ .eng) เป็นกราฟแสดงพลังของเสียงที่คำนวณได้ โดยแกน x แสดงระดับของพลังงาน แกน y แสดงลำดับของสัญญาณ

3. กราฟแสดงสัมประสิทธิ์การประมาณเชิงเส้น เป็นการแสดงค่าสัมประสิทธิ์เชิงเส้นที่คำนวณได้ โดยแกน x แสดงค่าของสัมประสิทธิ์ แกน y แสดงลำดับของสัมประสิทธิ์

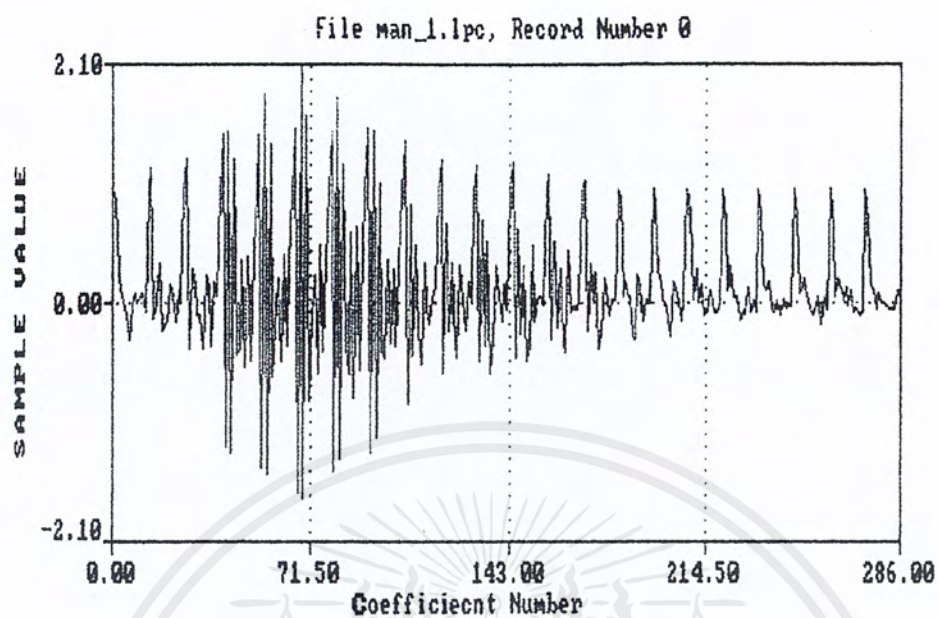
4. กราฟแสดงสัมประสิทธิ์สัมประสิทธิ์เซปสตรัม เป็นการแสดงค่าของสัมประสิทธิ์เซปสตรัม ที่คำนวณได้ โดยแกน x แสดงค่าของสัมประสิทธิ์ แกน y แสดงลำดับของสัมประสิทธิ์



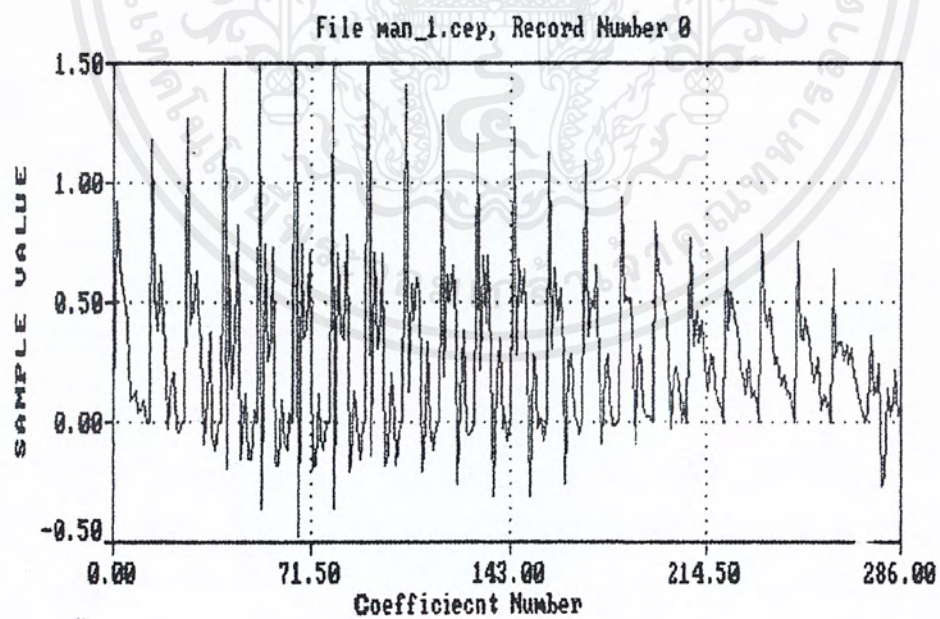
รูปที่ 4.2 รูปแบบสัญญาณเสียงของเสียงพูดคำว่า “หนึ่ง”



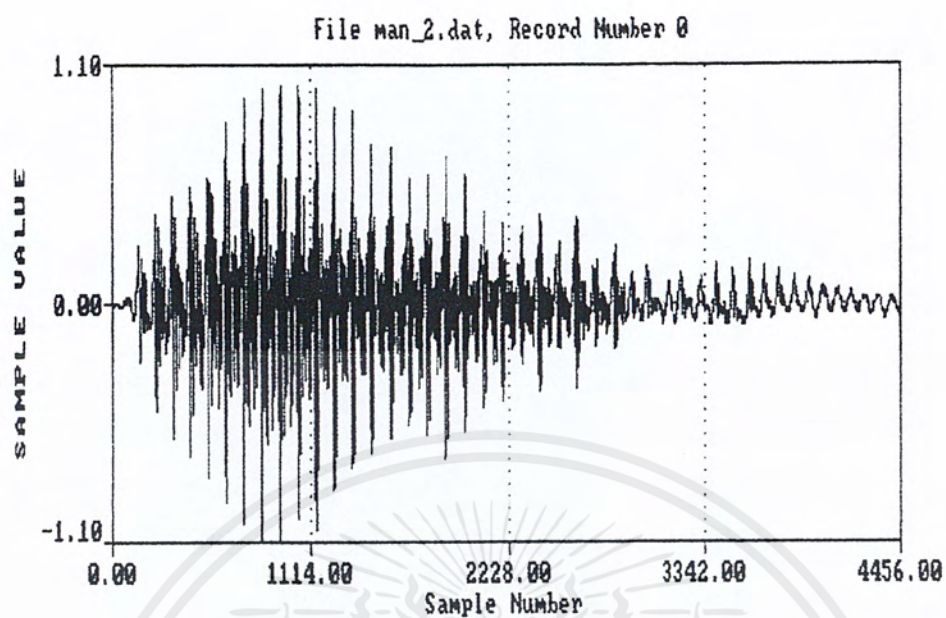
รูปที่ 4.3 รูปแบบพลังงานเสียงของเสียงพูดคำว่า “หนึ่ง”



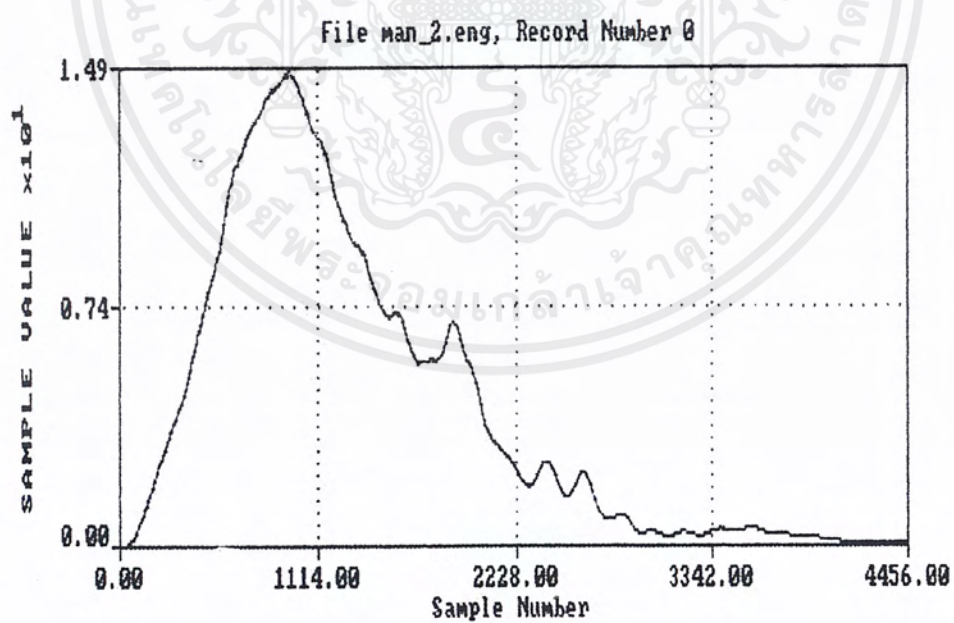
รูปที่ 4.4 รูปแบบสัมประสิทธิ์การประมาณเชิงเส้นของเสียงพูดคำว่า “หนึ่ง”



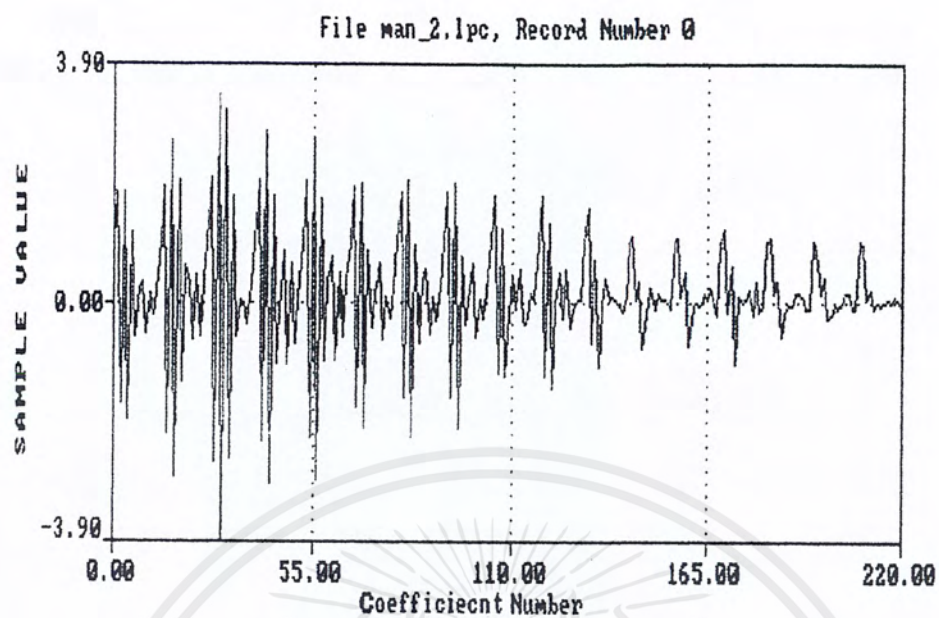
รูปที่ 4.5 รูปแบบสัมประสิทธิ์เซปสตรัมของเสียงพูดคำว่า “หนึ่ง”



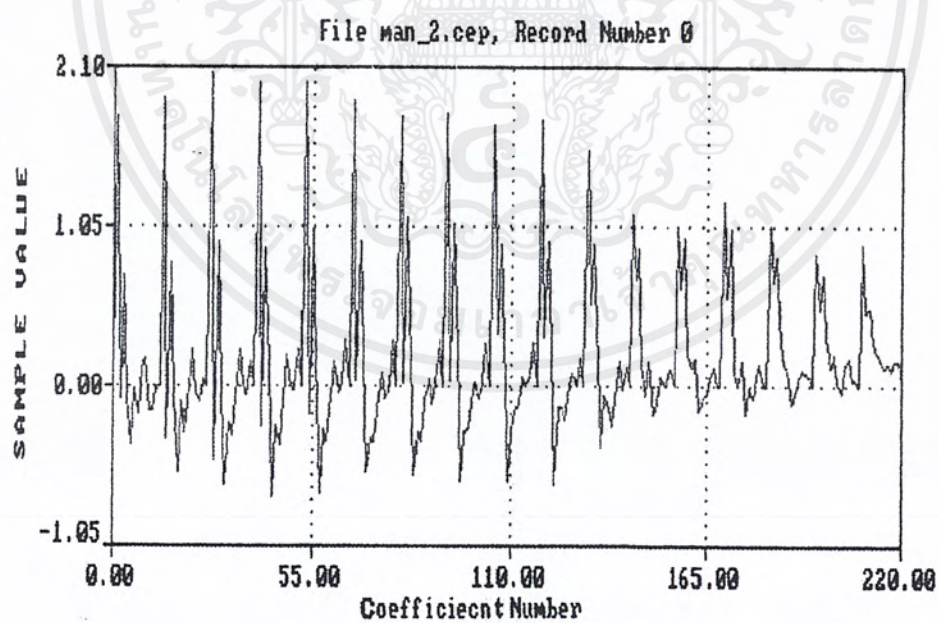
รูปที่ 4.6 รูปแบบสัญญาณเสียงของเสียงพูดคำว่า “สอง”



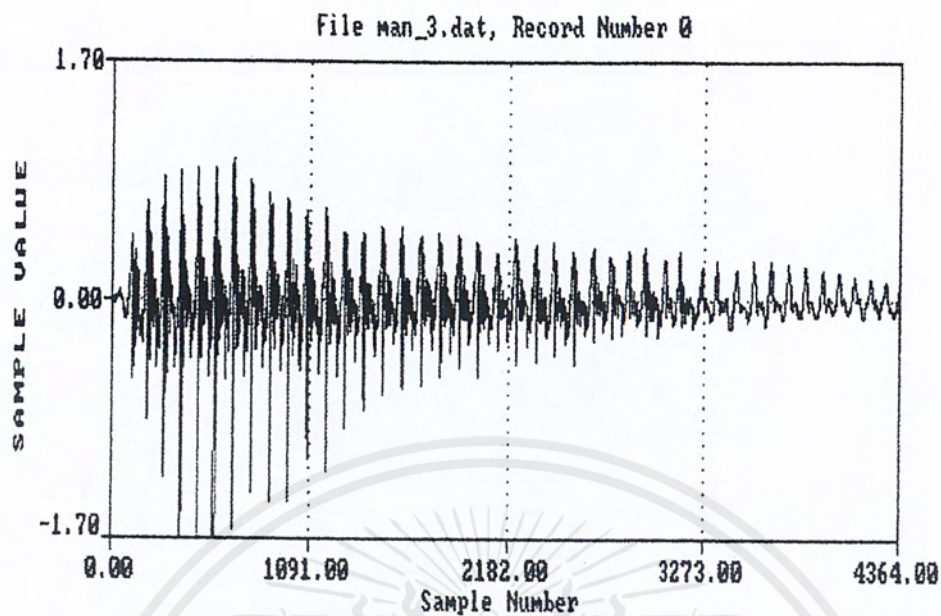
รูปที่ 4.7 รูปแบบพลังงานเสียงของเสียงพูดคำว่า “สอง”



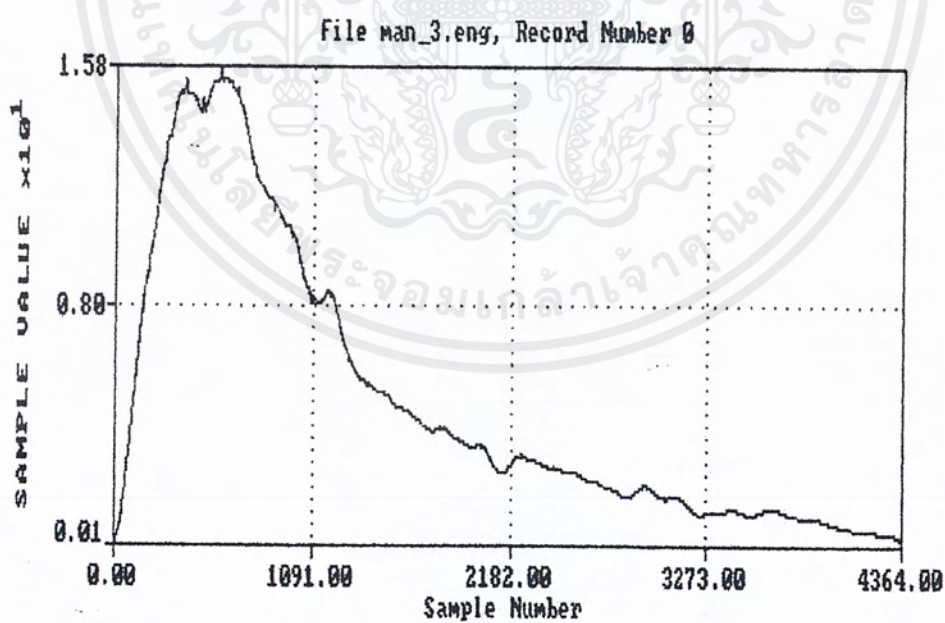
รูปที่ 4.8 รูปแบบสัมประสิทธิ์การประมาณเชิงเส้นของเสียงพูดคำว่า “สอง”



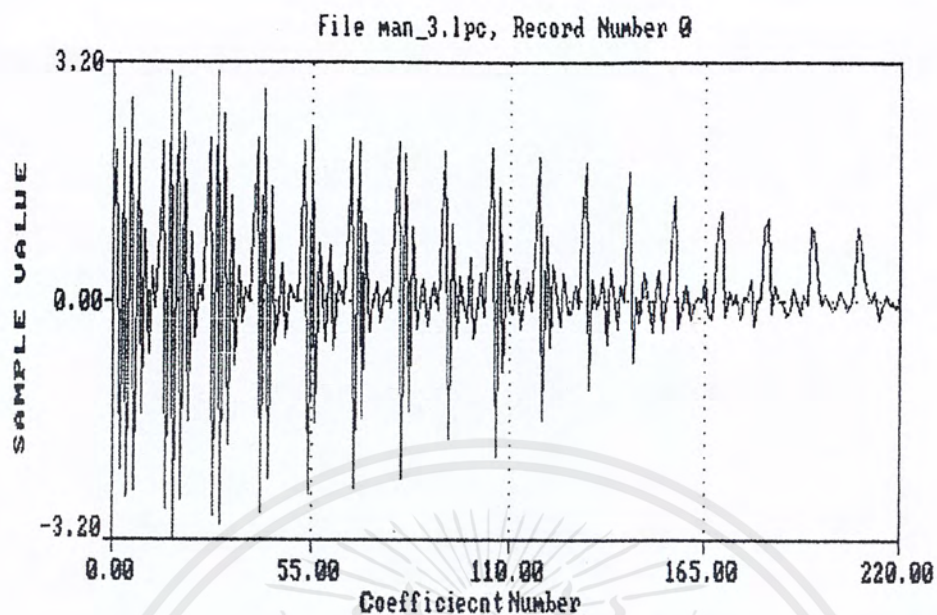
รูปที่ 4.9 รูปแบบสัมประสิทธิ์เซปสตรัมของเสียงพูดคำว่า “สอง”



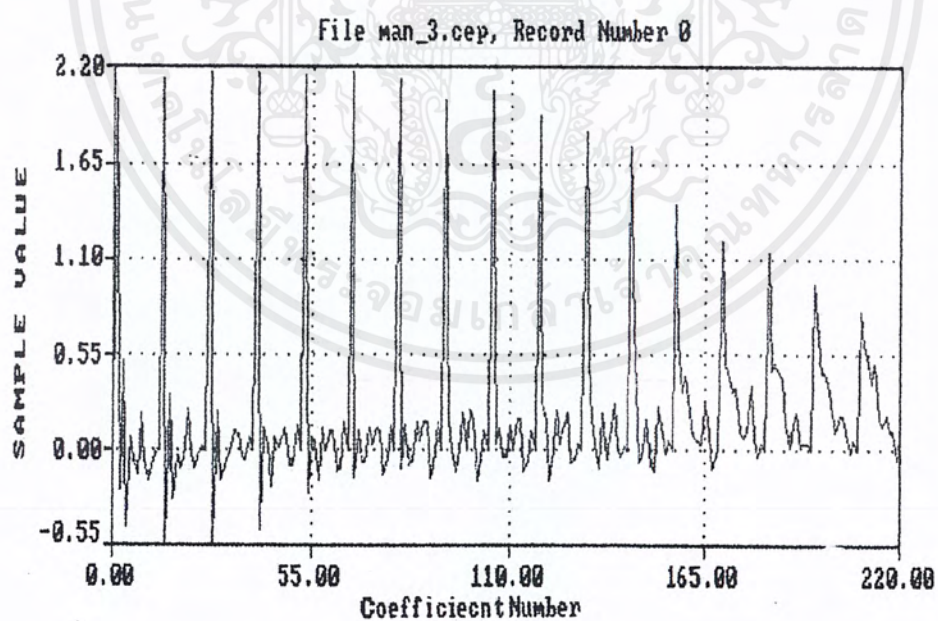
รูปที่ 4.10 รูปแบบสัญญาณเสียงของเสียงพูดคำว่า “สาม”



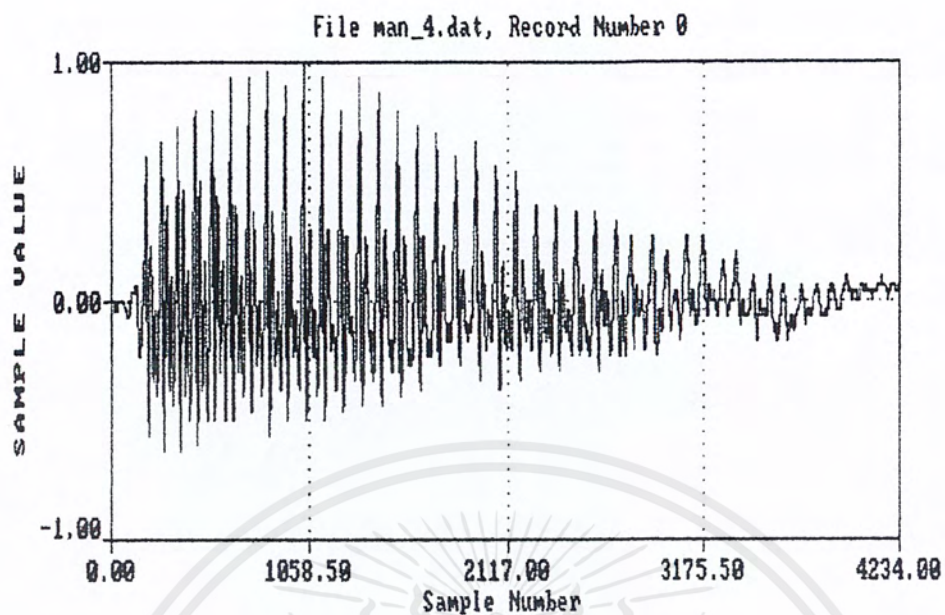
รูปที่ 4.11 รูปแบบพลังงานเสียงของเสียงพูดคำว่า “สาม”



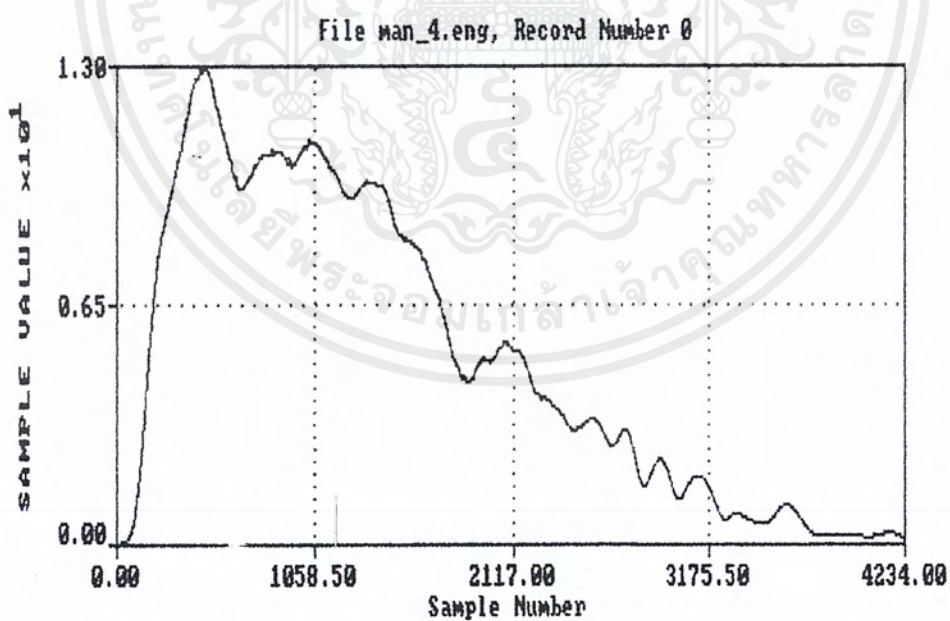
รูปที่ 4.12 รูปแบบสัมประสิทธิ์การประมาณเชิงเส้นของเสียงพูดคำว่า “สาม”



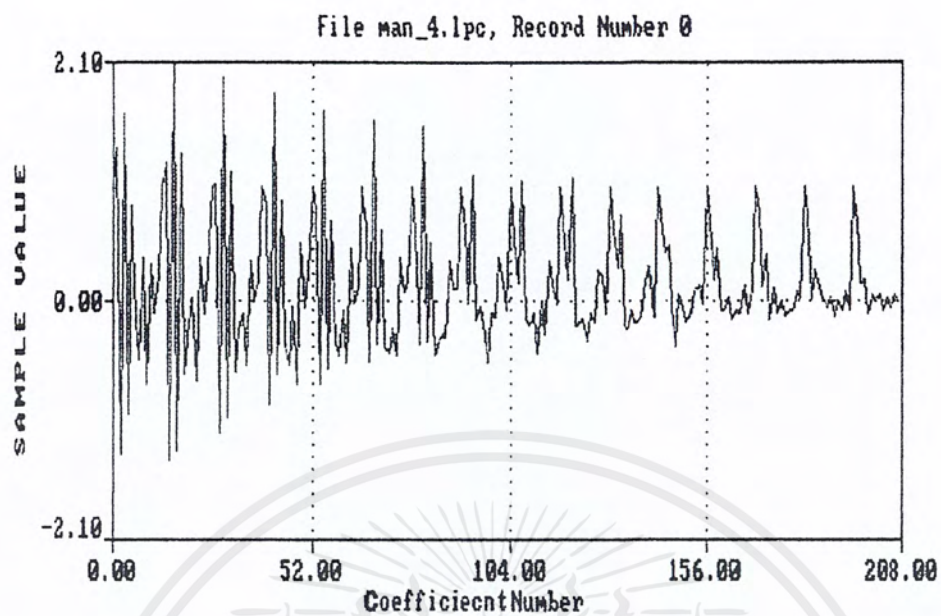
รูปที่ 4.13 รูปแบบสัมประสิทธิ์เซปสตรัมของเสียงพูดคำว่า “สาม”



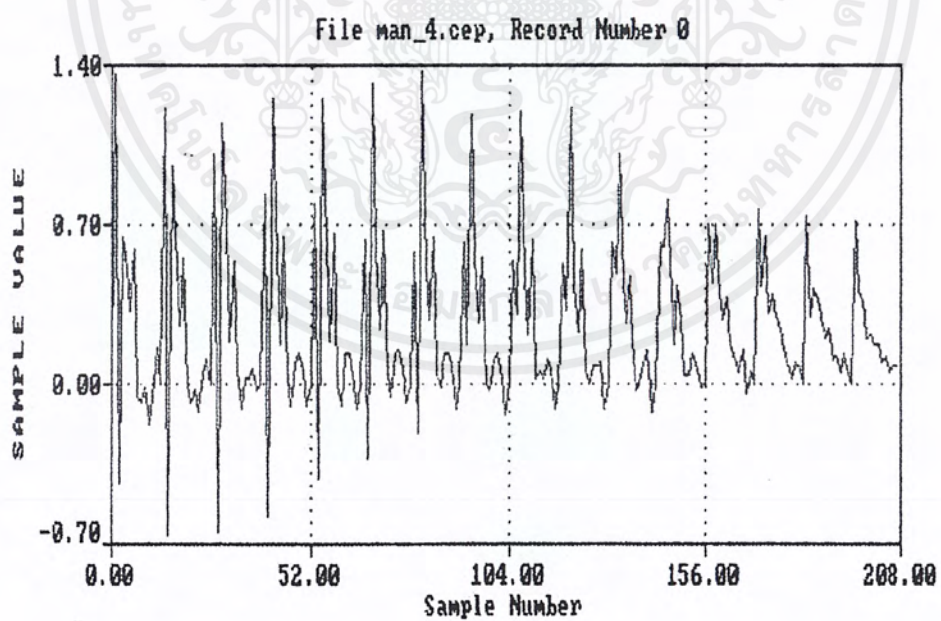
รูปที่ 4.14 รูปแบบสัญญาณเสียงของเสียงพูดคำว่า “สี่”



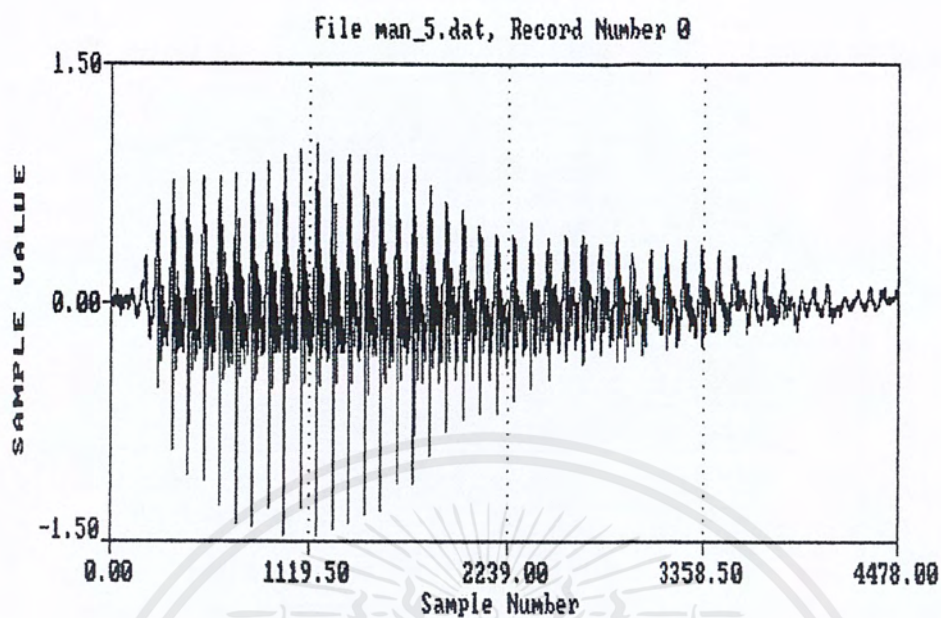
รูปที่ 4.15 รูปแบบพลังงานเสียงของเสียงพูดคำว่า “สี่”



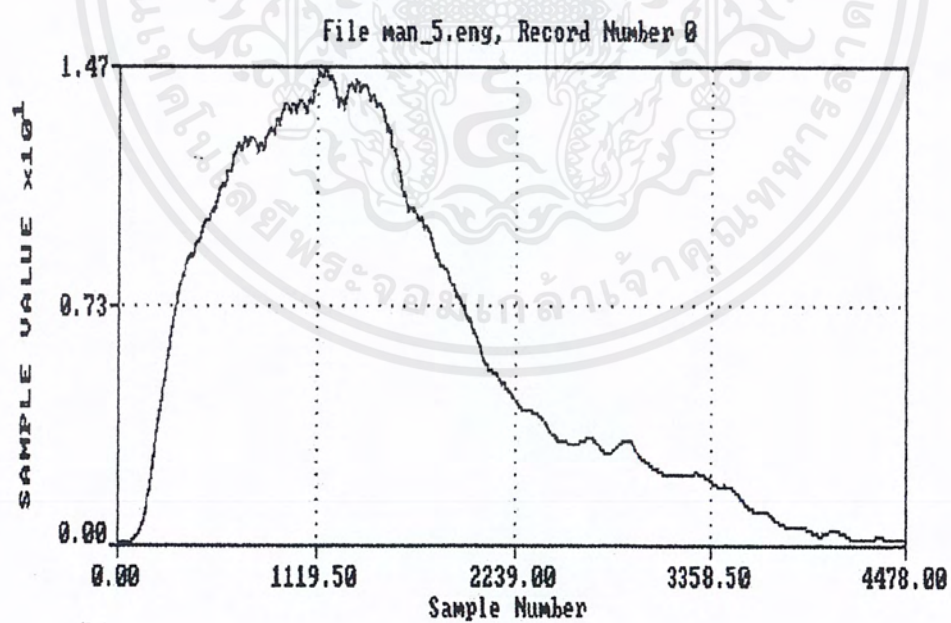
รูปที่ 4.16 รูปแบบสัมประสิทธิ์การประมาณเชิงเส้นของเสียงพูดคำว่า “สี่”



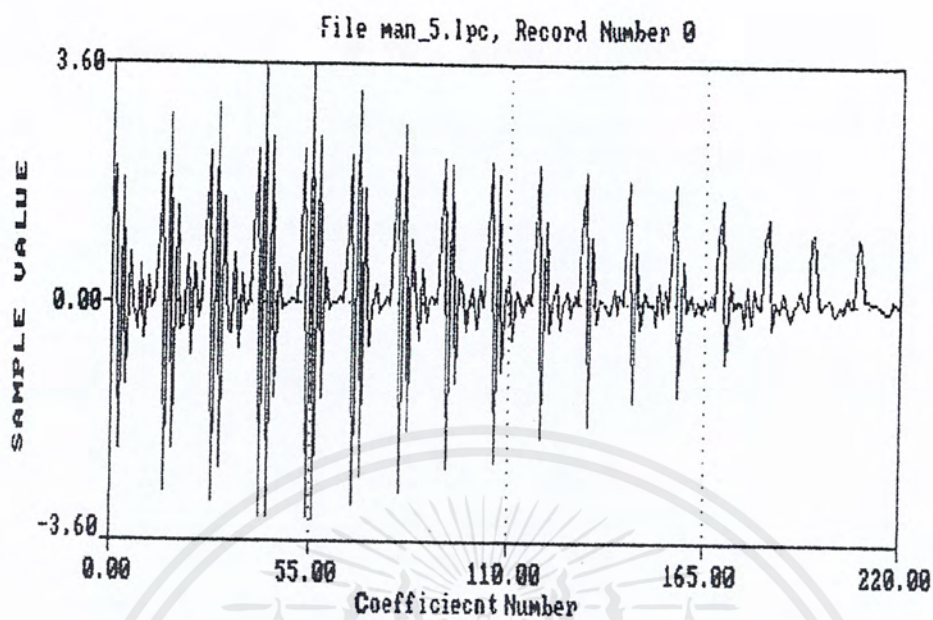
รูปที่ 4.17 รูปแบบสัมประสิทธิ์เซปสตรัมของเสียงพูดคำว่า “สี่”



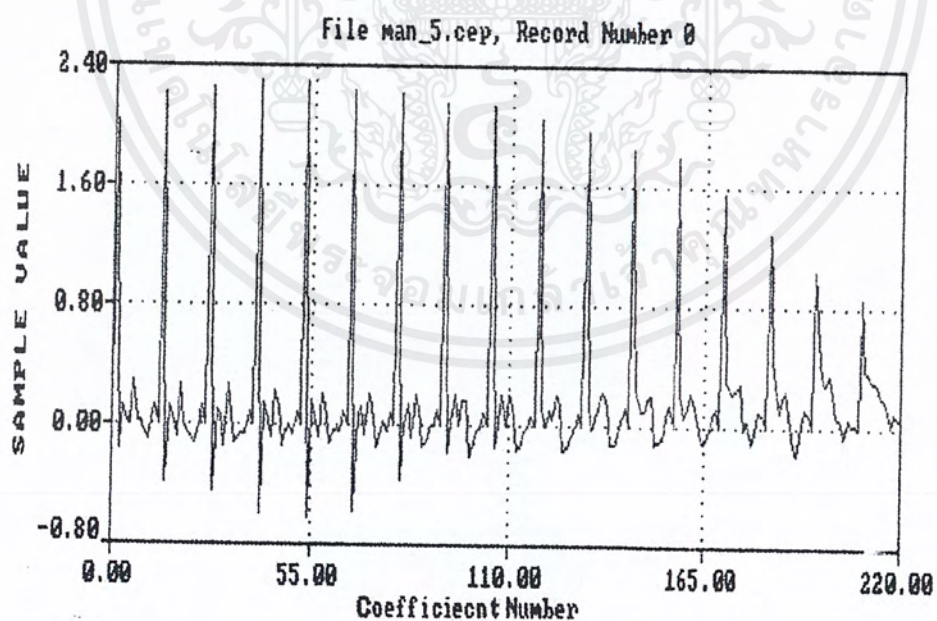
รูปที่ 4.18 รูปแบบสัญญาณเสียงของเสียงพูดคำว่า "ห้า"



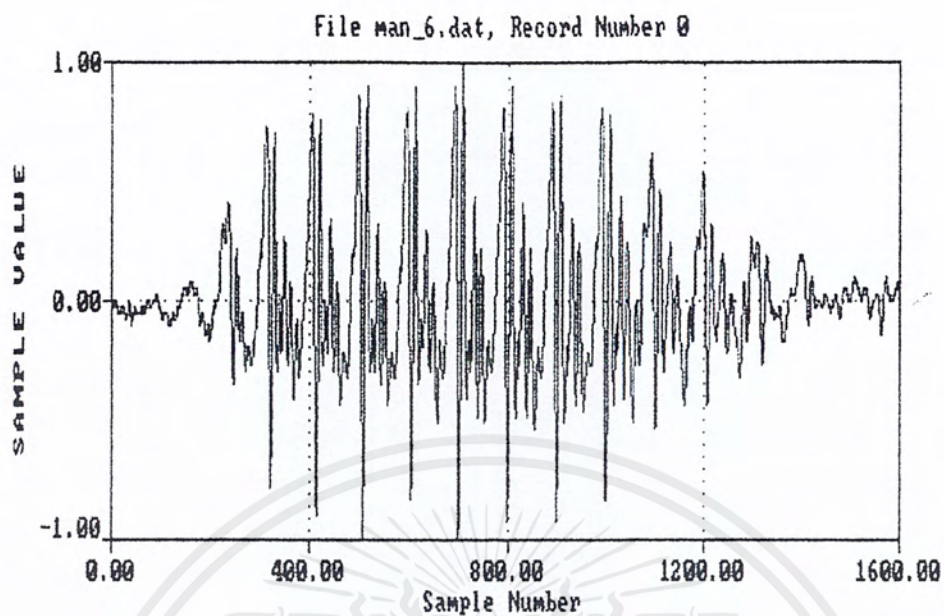
รูปที่ 4.19 รูปแบบพลังงานเสียงของเสียงพูดคำว่า "ห้า"



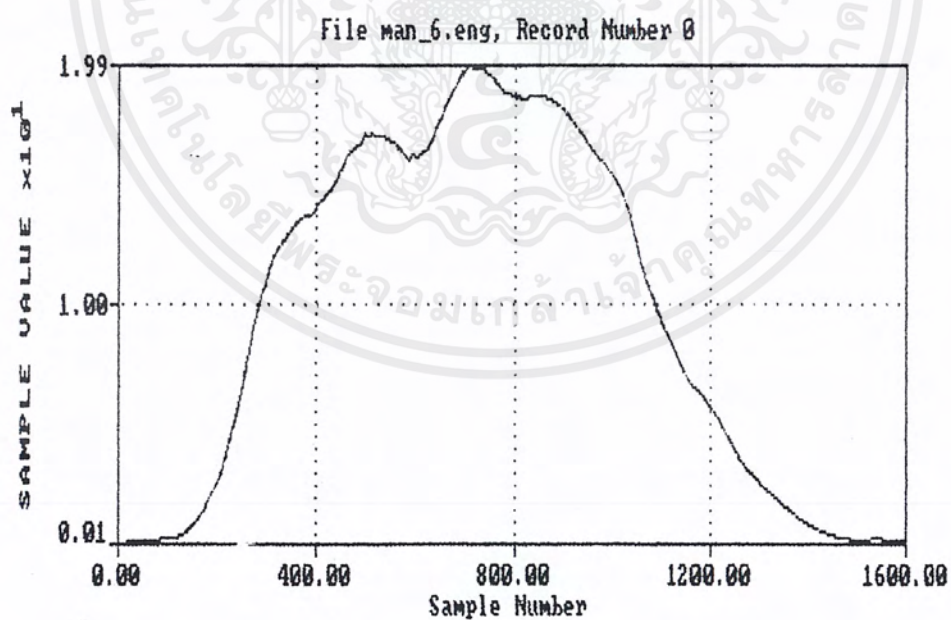
รูปที่ 4.20 รูปแบบสัมประสิทธิ์การประมาณเชิงเส้นของเสียงพูดคำว่า “ห่า”



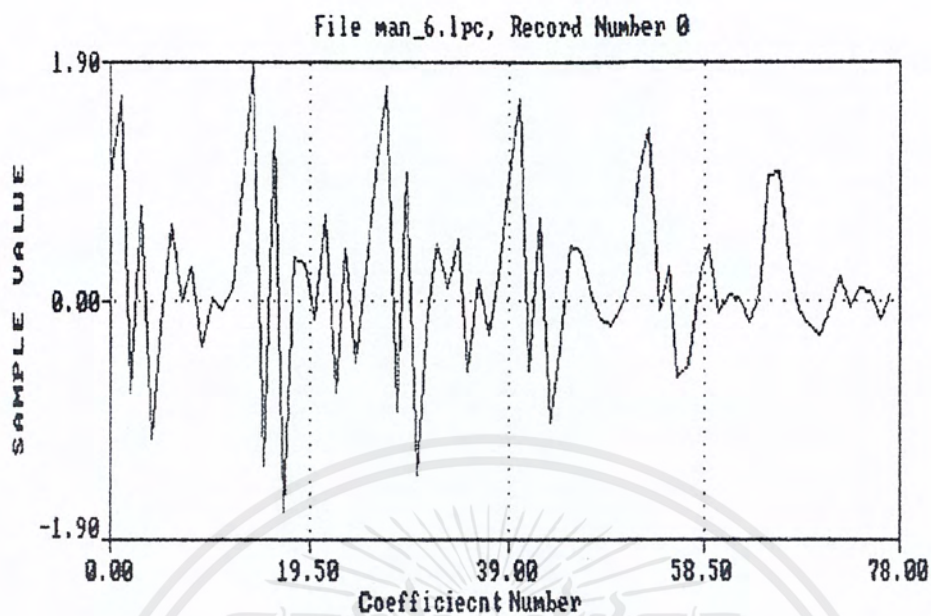
รูปที่ 4.21 รูปแบบสัมประสิทธิ์เซปสตรัมของเสียงพูดคำว่า “ห่า”



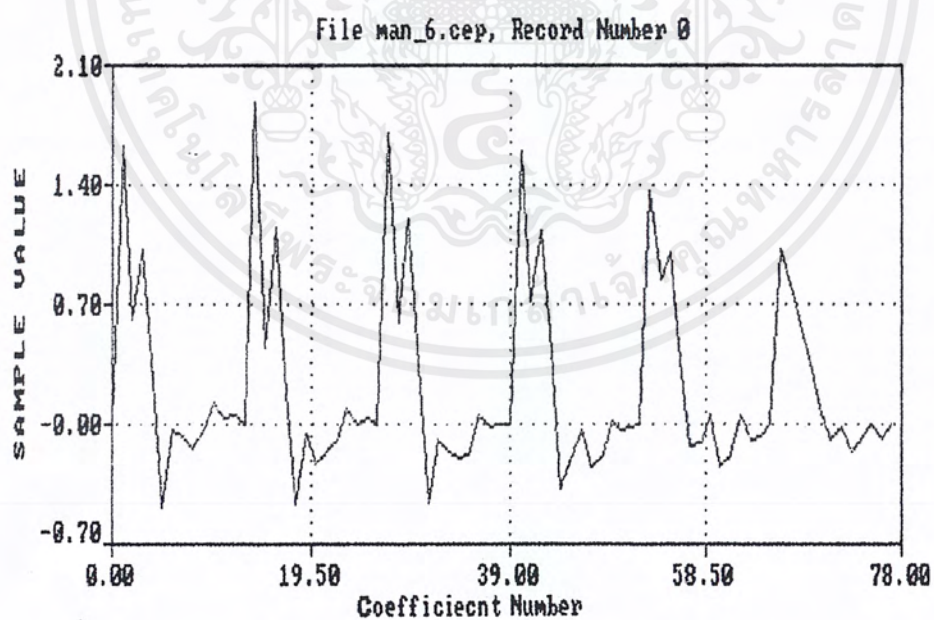
รูปที่ 4.22 รูปแบบสัญญาณเสียงของเสียงพูดคำว่า “หก”



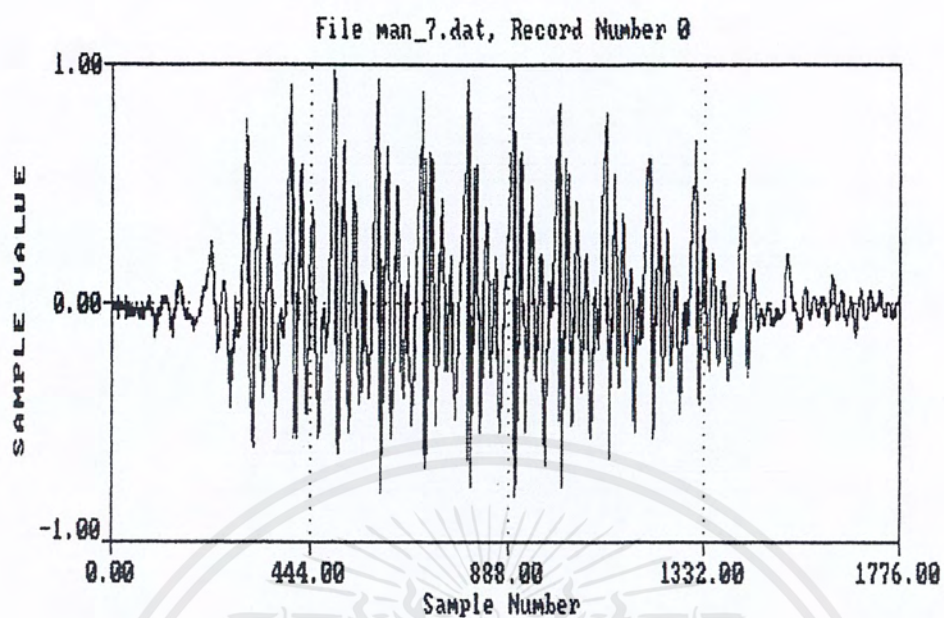
รูปที่ 4.23 รูปแบบพลังงานเสียงของเสียงพูดคำว่า “หก”



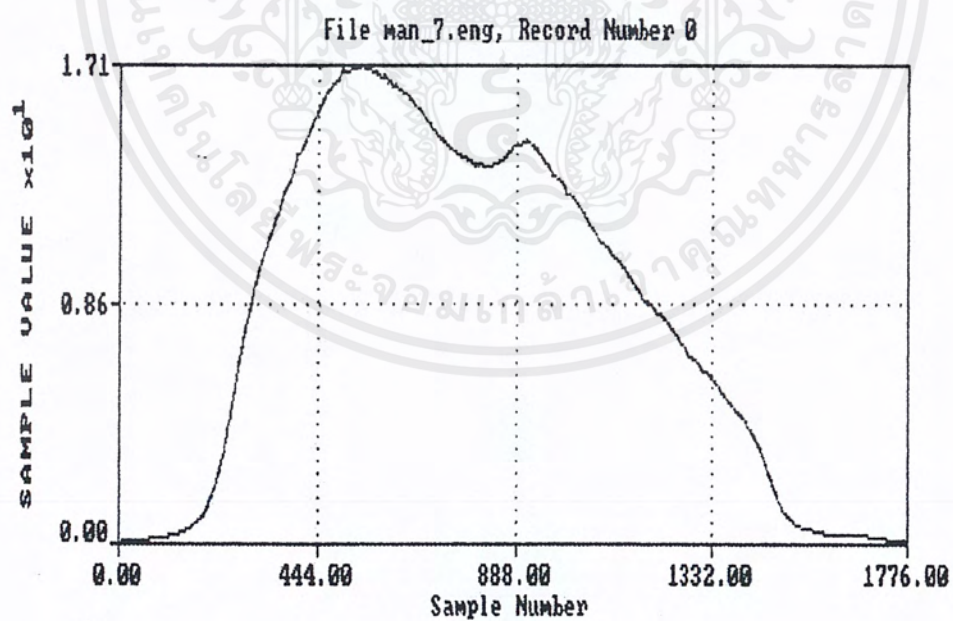
รูปที่ 4.24 รูปแบบสัมประสิทธิ์การประมาณเชิงเส้นของเสียงพูดคำว่า “hak”



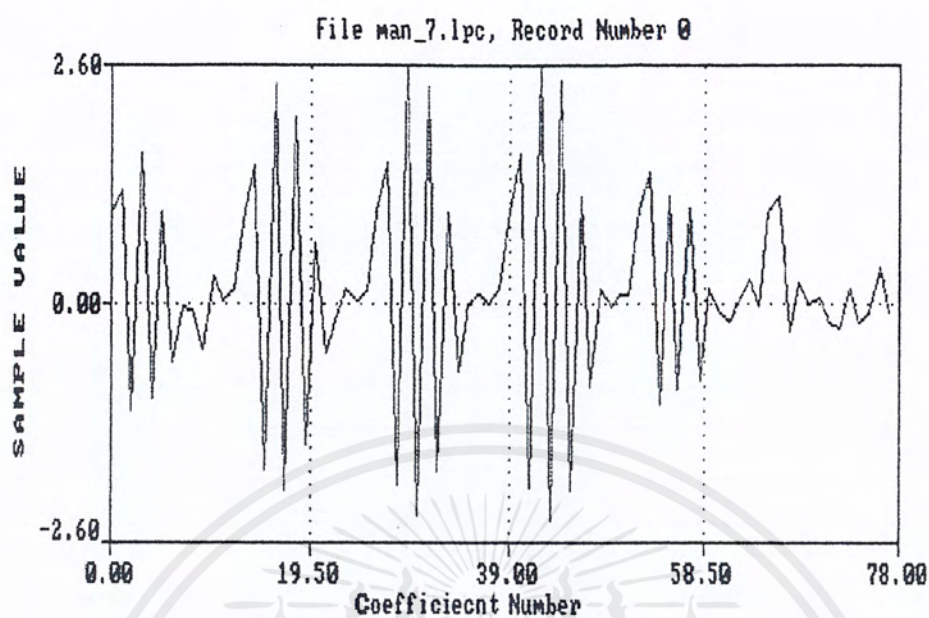
รูปที่ 4.25 รูปแบบสัมประสิทธิ์เซปสตรัมของเสียงพูดคำว่า “hak”



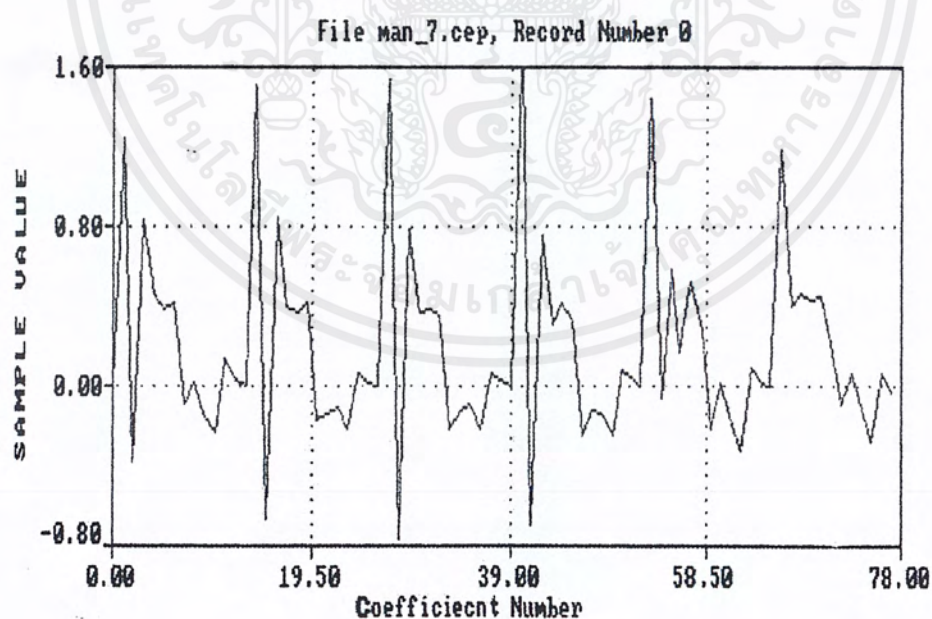
รูปที่ 4.26 รูปแบบสัญญาณเสียงของเสียงพูดคำว่า “เจ็ด”



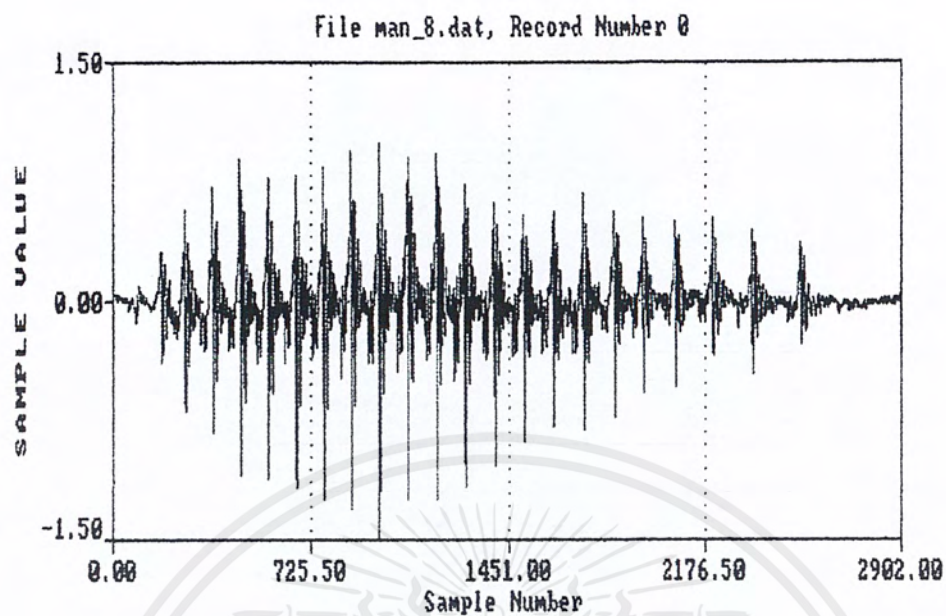
รูปที่ 4.27 รูปแบบพลังงานเสียงของเสียงพูดคำว่า “เจ็ด”



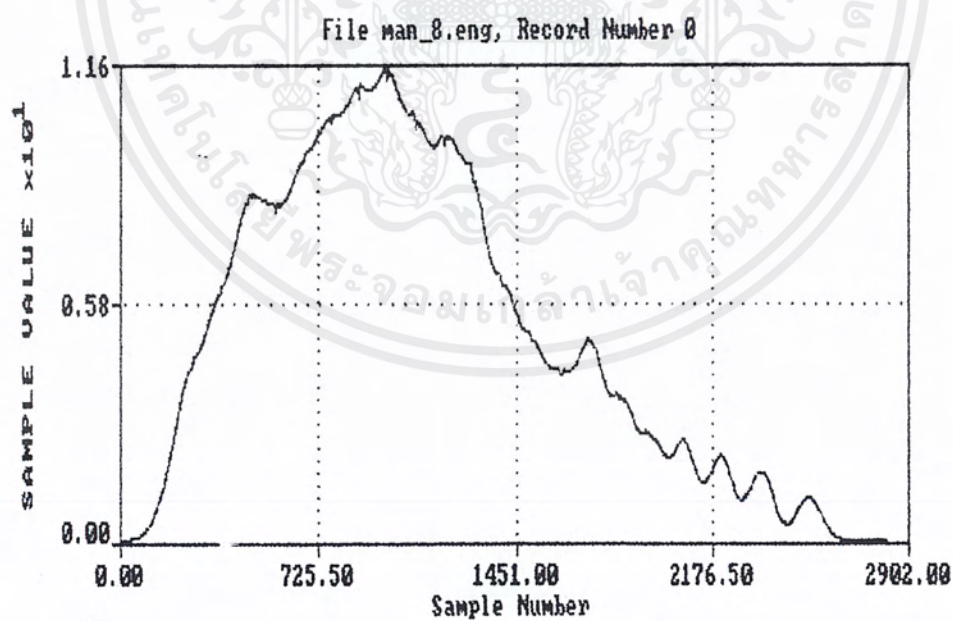
รูปที่ 4.28 รูปแบบสัมประสิทธิ์การประมาณเชิงเส้นของเสียงพูดคำว่า “เจ็ด”



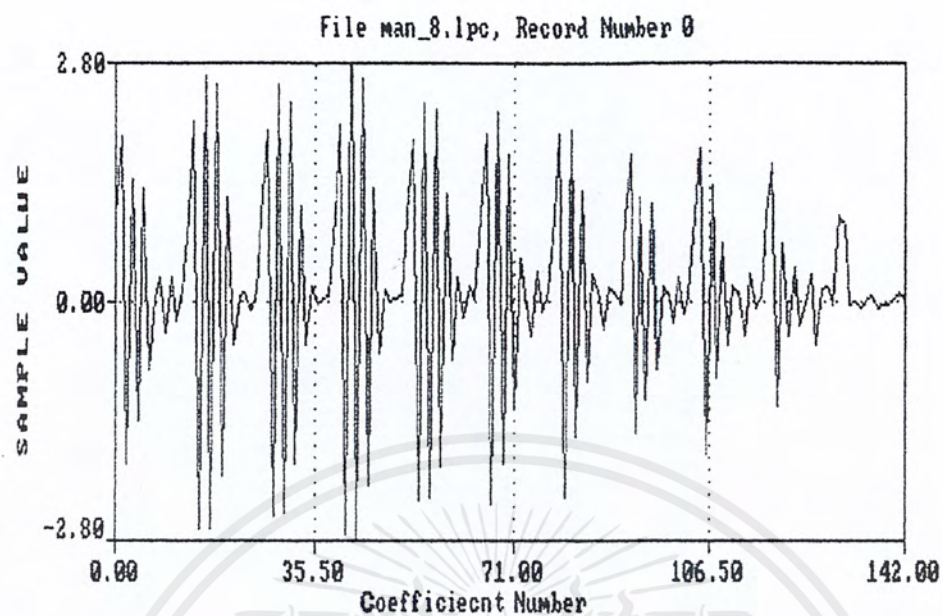
รูปที่ 4.29 รูปแบบสัมประสิทธิ์เซปสตรีมของเสียงพูดคำว่า “เจ็ด”



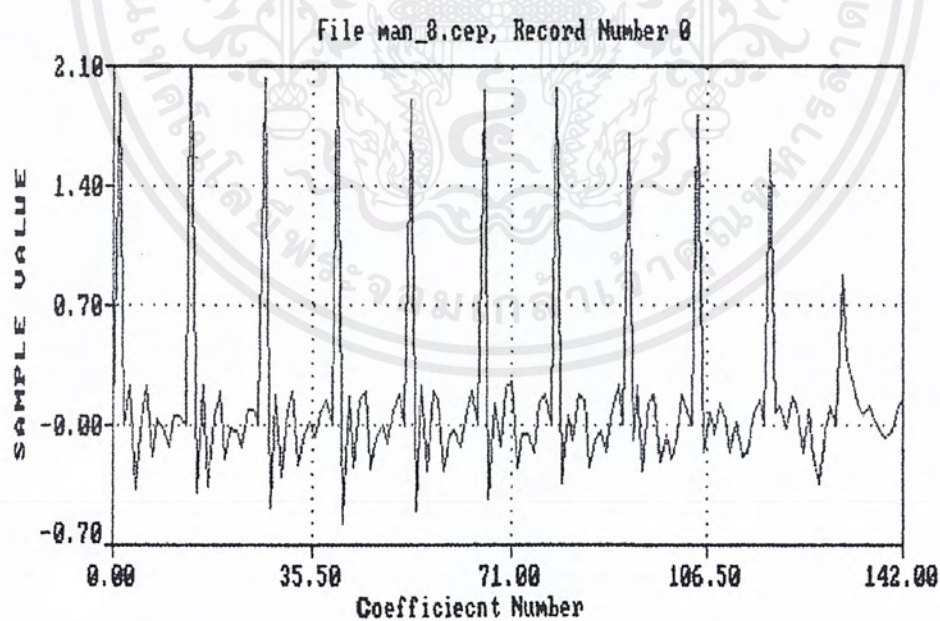
รูปที่ 4.30 รูปแบบสัญญาณเสียงของเสียงพูดคำว่า “แปด”



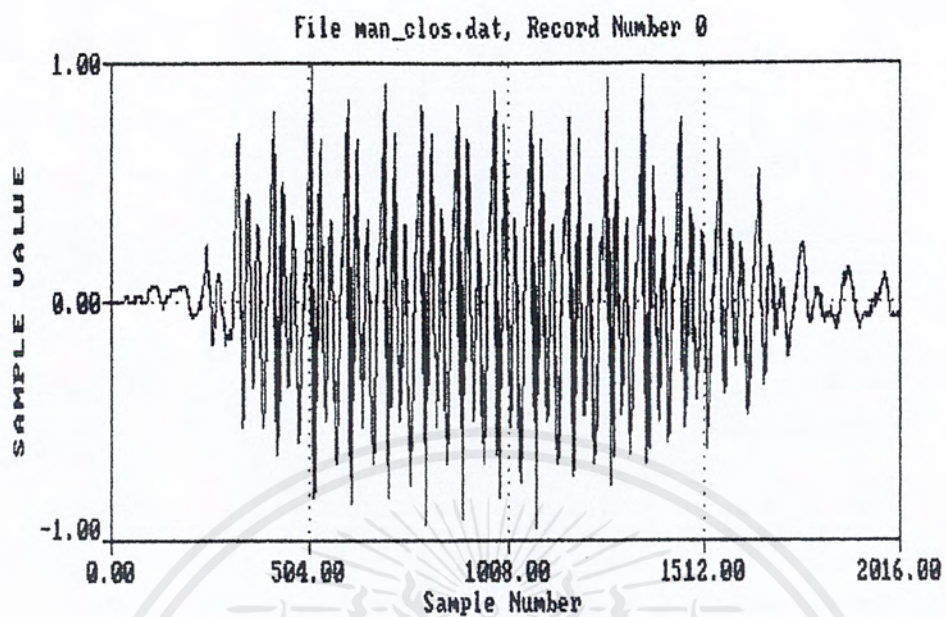
รูปที่ 4.31 รูปแบบพลังงานเสียงของเสียงพูดคำว่า “แปด”



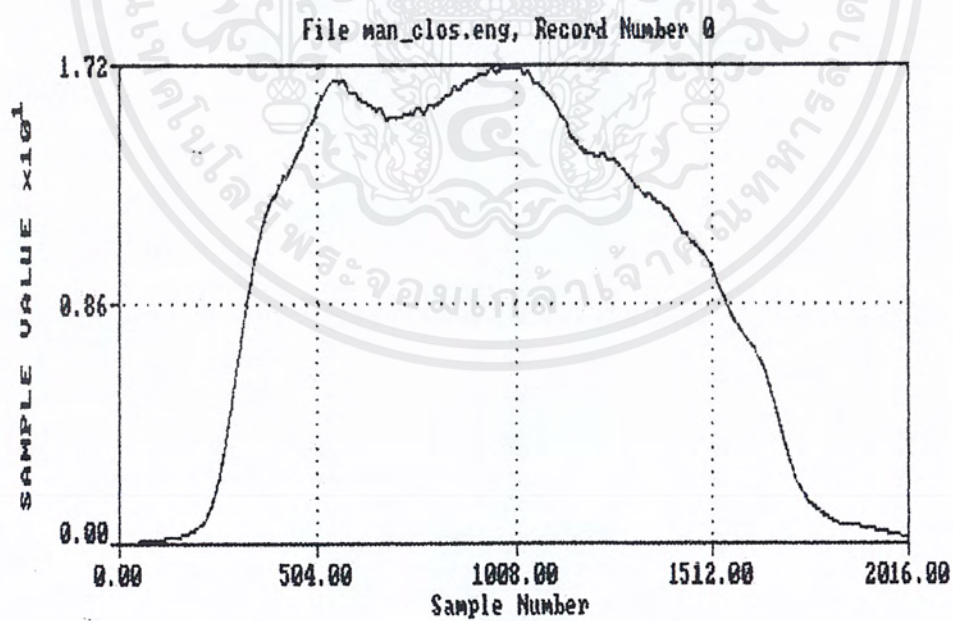
รูปที่ 4.32 รูปแบบสัมประสิทธิ์การประมาณเชิงเส้นของเสียงพูดคำว่า “แปด”



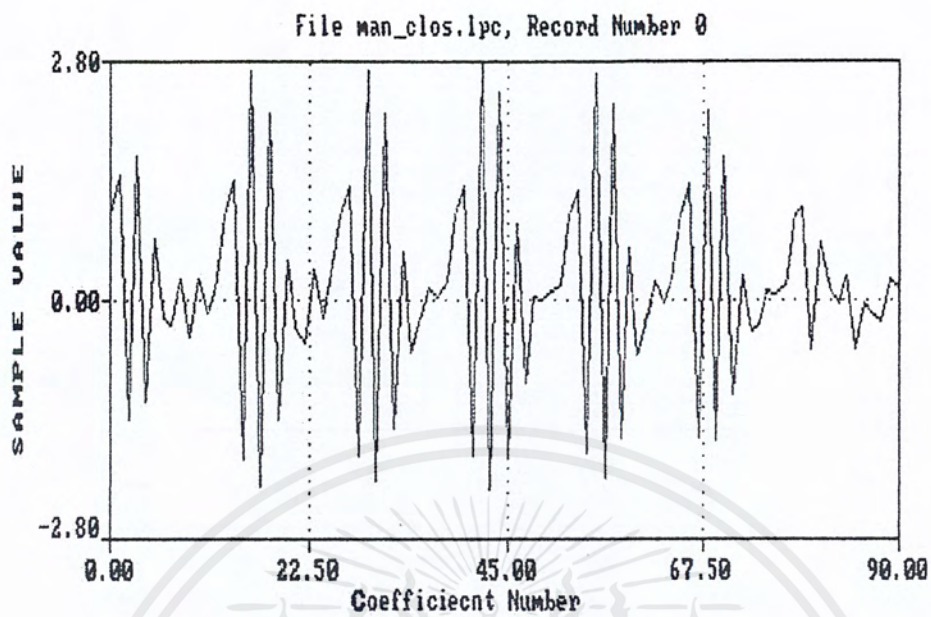
รูปที่ 4.33 รูปแบบสัมประสิทธิ์เซปสตรัมของเสียงพูดคำว่า “แปด”



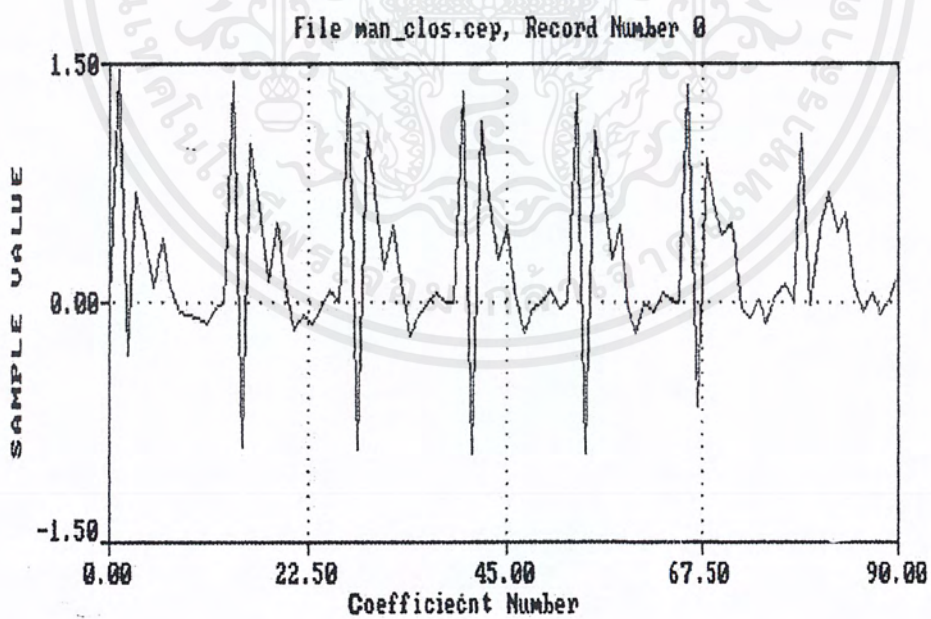
รูปที่ 4.34 รูปแบบสัญญาณเสียงของเสียงพูดคำว่า "ปิด"



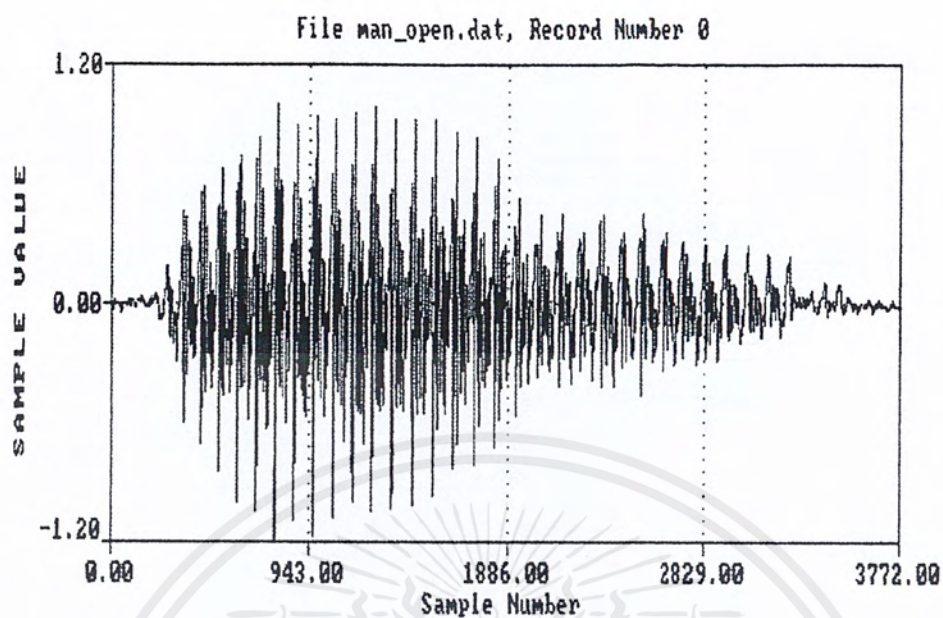
รูปที่ 4.35 รูปแบบพลังงานเสียงของเสียงพูดคำว่า "ปิด"



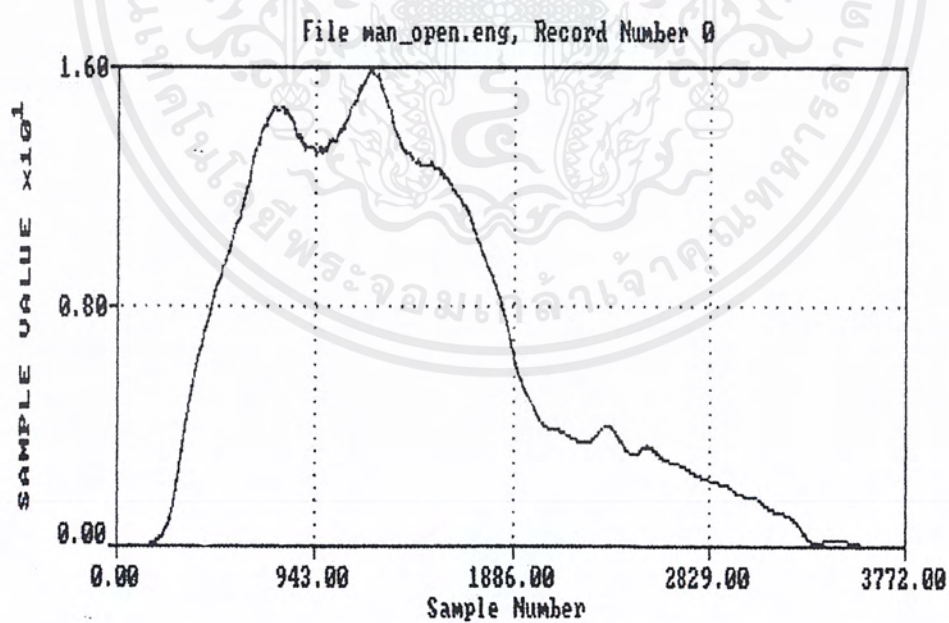
รูปที่ 4.36 รูปแบบสัมประสิทธิ์การประมาณเชิงเส้นของเสียงพูดคำว่า “ปิด”



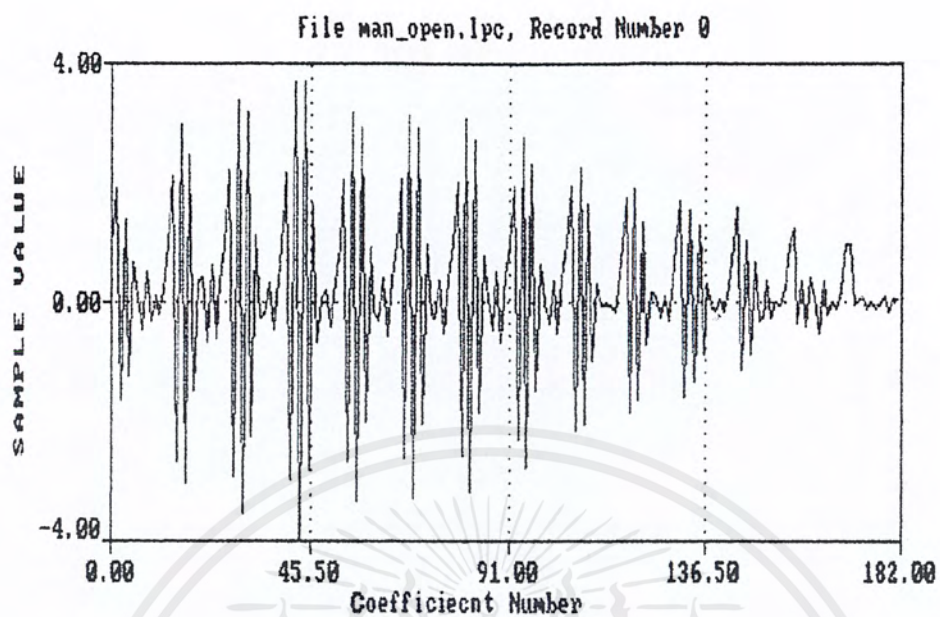
รูปที่ 4.37 รูปแบบสัมประสิทธิ์เซปสตรัมของเสียงพูดคำว่า “ปิด”



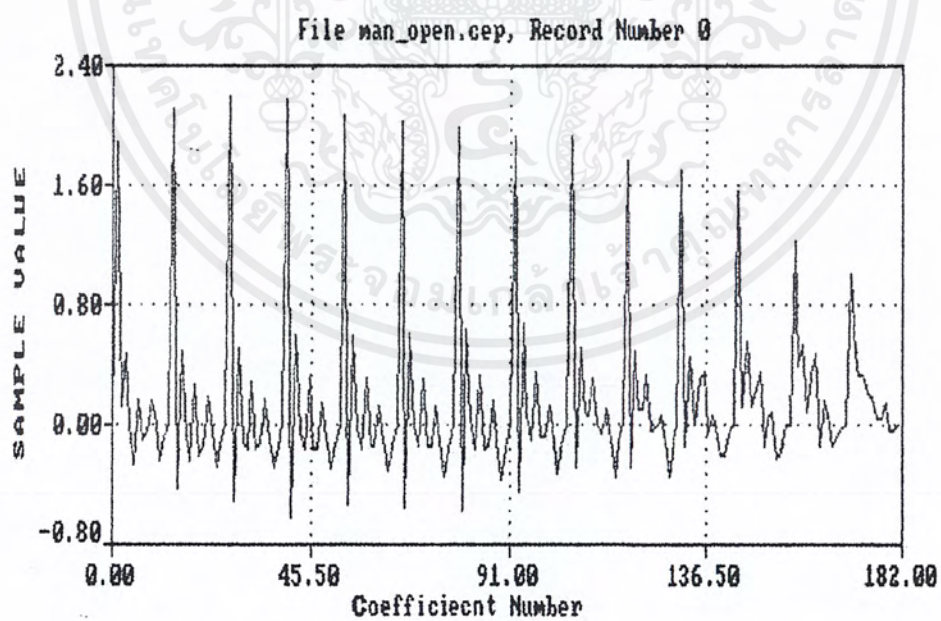
รูปที่ 4.38 รูปแบบสัญญาณเสียงของเสียงพูดคำว่า “เปิด”



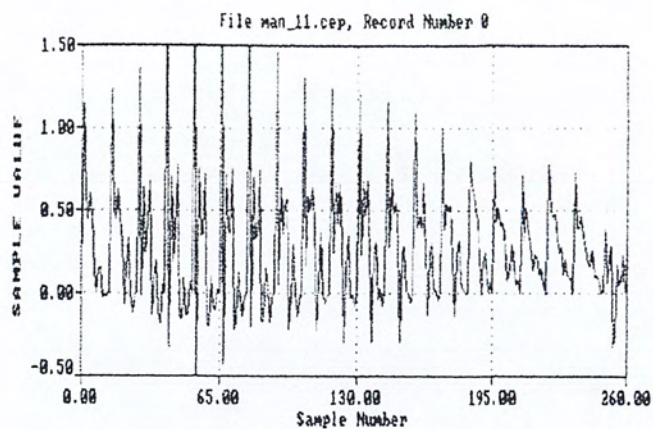
รูปที่ 4.39 รูปแบบพลังงานเสียงของเสียงพูดคำว่า “เปิด”



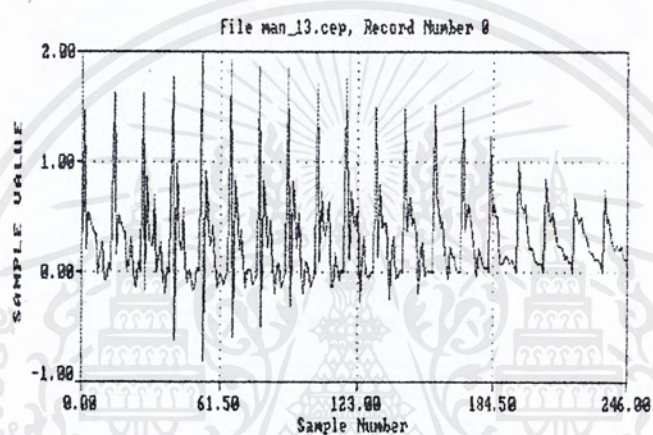
รูปที่ 4.40 รูปแบบสัมประสิทธิ์การประมาณเชิงเส้นของเสียงพูดคำว่า “เปิด”



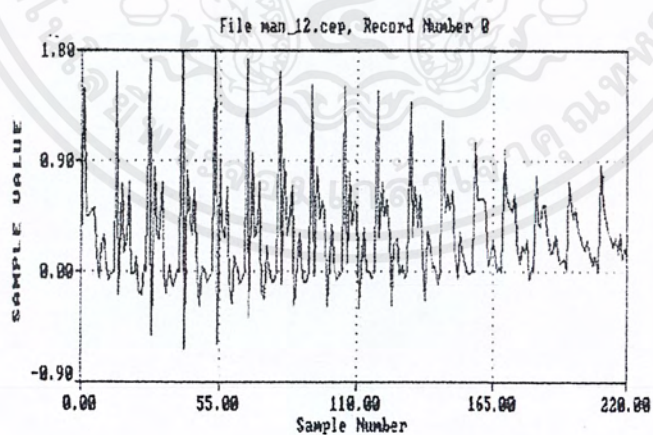
รูปที่ 4.41 รูปแบบสัมประสิทธิ์เซปสตรัมของเสียงพูดคำว่า “เปิด”



(ก)



(ข)



(ค)

รูปที่ 4.42 แบบของสัญญาณเสียงพูดคำว่า “หนึ่ง” ในแต่ละครั้ง

4.3.2 ตารางแสดงผลการทดลอง

ตารางที่ 4.2 ผลการทดลองจากการใช้แบบเสียงอ้างอิง 3 เสียงเพื่อเปรียบเทียบ

เสียงพูด	ครั้งที่										เปอร์เซ็นต์ ความถูกต้อง
	1	2	3	4	5	6	7	8	9	10	
“เปิด”	ถูก	ผิด	ถูก	ผิด	ถูก	ถูก	ผิด	ถูก	ถูก	ผิด	60%
“ปิด”	ถูก	ผิด	ผิด	ถูก	ผิด	ผิด	ถูก	ผิด	ถูก	ผิด	40%
“หนึ่ง”	ผิด	ผิด	ถูก	ผิด	ถูก	ผิด	ถูก	ถูก	ผิด	ถูก	50%
“สอง”	ถูก	ถูก	ผิด	ถูก	ถูก	ผิด	ผิด	ถูก	ถูก	ผิด	60%
“สาม”	ถูก	ผิด	ถูก	ผิด	ผิด	ถูก	ผิด	ถูก	ถูก	ผิด	50%
“สี่”	ผิด	ผิด	ถูก	ถูก	ถูก	ถูก	ผิด	ถูก	ผิด	ถูก	60%
“ห้า”	ผิด	ถูก	ถูก	ผิด	ผิด	ผิด	ถูก	ผิด	ถูก	ถูก	50%
“หก”	ถูก	ถูก	ผิด	ถูก	ถูก	ผิด	ถูก	ผิด	ผิด	ถูก	60%
“เจ็ด”	ถูก	ผิด	ถูก	ถูก	ผิด	ผิด	ถูก	ถูก	ถูก	ผิด	60%
“แปด”	ถูก	ผิด	ถูก	ผิด	ถูก	ผิด	ผิด	ถูก	ถูก	ผิด	50%

ตารางที่ 4.3 ผลการทดลองจากการใช้แบบเสียงอ้างอิง 4 เสียงเพื่อเปรียบเทียบ

เสียงพูด	ครั้งที่										เปอร์เซ็นต์ ความถูกต้อง
	1	2	3	4	5	6	7	8	9	10	
“เปิด”	ถูก	ผิด	ถูก	ถูก	ถูก	ผิด	ถูก	ถูก	ผิด	ถูก	70%
“ปิด”	ถูก	ถูก	ผิด	ถูก	ผิด	ผิด	ถูก	ผิด	ถูก	ผิด	50%
“หนึ่ง”	ถูก	ถูก	ผิด	ถูก	ผิด	ผิด	ถูก	ผิด	ถูก	ถูก	60%
“สอง”	ผิด	ถูก	ผิด	ถูก	ถูก	ผิด	ถูก	ผิด	ถูก	ถูก	60%
“สาม”	ผิด	ถูก	ถูก	ถูก	ผิด	ถูก	ผิด	ผิด	ถูก	ถูก	60%
“สี่”	ถูก	ถูก	ถูก	ถูก	ผิด	ถูก	ผิด	ถูก	ผิด	ถูก	70%
“ห้า”	ถูก	ถูก	ผิด	ผิด	ถูก	ถูก	ถูก	ผิด	ผิด	ถูก	60%
“หก”	ถูก	ถูก	ผิด	ถูก	ถูก	ผิด	ถูก	ถูก	ถูก	ผิด	70%
“เจ็ด”	ถูก	ถูก	ผิด	ถูก	ผิด	ถูก	ผิด	ถูก	ถูก	ผิด	60%
“แปด”	ถูก	ถูก	ถูก	ผิด	ผิด	ถูก	ผิด	ผิด	ถูก	ถูก	60%

ตารางที่ 4.4 ผลการทดลองจากการใช้แบบเสียงอ้างอิง 5 เสียงเพื่อเปรียบเทียบ

เสียงพูด	ครั้งที่										เปอร์เซ็นต์ ความถูกต้อง
	1	2	3	4	5	6	7	8	9	10	
“เปิด”	ถูก	ถูก	ถูก	ผิด	ถูก	ถูก	ผิด	ถูก	ถูก	ผิด	70%
“ปิด”	ถูก	ถูก	ผิด	ถูก	ผิด	ผิด	ถูก	ผิด	ถูก	ผิด	50%
“หนึ่ง”	ผิด	ถูก	ถูก	ถูก	ถูก	ผิด	ถูก	ถูก	ผิด	ถูก	70%
“สอง”	ถูก	ถูก	ผิด	ถูก	ถูก	ถูก	ถูก	ถูก	ถูก	ผิด	80%
“สาม”	ถูก	ผิด	ถูก	ถูก	ผิด	ถูก	ถูก	ถูก	ถูก	ถูก	80%
“สี่”	ผิด	ถูก	ถูก	ถูก	ถูก	ถูก	ผิด	ถูก	ผิด	ถูก	70%
“ห้า”	ผิด	ถูก	ถูก	ผิด	ผิด	ถูก	ถูก	ผิด	ถูก	ถูก	60%
“หก”	ถูก	ถูก	ถูก	ถูก	ถูก	ผิด	ถูก	ผิด	ถูก	ถูก	80%
“เจ็ด”	ถูก	ถูก	ถูก	ถูก	ผิด	ถูก	ถูก	ถูก	ถูก	ผิด	80%
“แปด”	ถูก	ถูก	ถูก	ผิด	ถูก	ถูก	ถูก	ถูก	ถูก	ผิด	80%

บทที่ 5

บทวิจารณ์และสรุป

ในการวิเคราะห์นี้จะทำการวิเคราะห์ผลการทดลองในแต่ละภาคเป็นการเบื้องต้น โดยจะแสดงปัญหาและแนวทางการแก้ปัญหาของวงจรและโปรแกรมที่ออกแบบไว้ รวมถึงแนวโน้มการนำไปใช้งานจริง โดยทำการเชื่อมต่อเข้ากับส่วนต่าง ๆ ว่าผลที่ได้เป็นอย่างไร โดยในการวิเคราะห์นี้จะแบ่งเป็น 2 ภาคใหญ่ๆ ดังนี้คือ

1. ภาคซอฟต์แวร์
2. ภาคฮาร์ดแวร์

5.1 สรุปผลการทดลอง

5.1.1 ส่วนของภาคฮาร์ดแวร์

จากการทดลองในส่วนของการควบคุมการเปิด-ปิดเครื่องใช้ไฟฟ้า สามารถใช้ได้เป็นอย่างดี ถูกต้องตามทฤษฎีที่ได้กำหนดไว้ โดยเครื่องควบคุมนี้สามารถทำการปิด หรือเปิดเครื่องใช้ไฟฟ้าได้ตามที่โปรแกรมทำการสั่งการมาอย่างถูกต้อง

5.1.2 ส่วนของภาคซอฟต์แวร์

ภาคซอฟต์แวร์ได้ทำการเขียนโปรแกรมโดยใช้ภาษา Visual C++ ซึ่งเป็นการเขียนแอปพลิเคชันบนวินโดวส์ การทำงานดังกล่าวได้ผลตามที่คาดหมายไว้ โดยซอฟต์แวร์ได้จัดทำเมนูในการรับคำสั่งจากผู้ใช้งานที่ต้องการเปิดหรือปิดอุปกรณ์ไฟฟ้า ในโหมดแมนนวล และโหมดการทำงานอัตโนมัติ

จากตารางผลการทดลอง เมื่อทำการทดลองซ้ำ ๆ กันหลาย ๆ ครั้ง จะได้ค่าความถูกต้องโดยเฉลี่ยประมาณ 72 เปอร์เซ็นต์ ซึ่งได้มาจากการใช้โมเดลเสียงถึง 5 โมเดลเพื่อใช้ในการเปรียบเทียบ จะเห็นได้ว่าหากโมเดลที่ใช้ในการเปรียบเทียบมากขึ้น ค่าความถูกต้องก็จะเพิ่มขึ้นตามไปด้วย แต่หากใช้โมเดลในการเปรียบเทียบมากเกินไป ก็จะมีผลทำให้การประมวลผลของโปรแกรมเกิดการล่าช้าขึ้นได้ และเพื่อความถูกต้องมากที่สุด ในการสั่งงานก็ควรพูดให้คล้ายกับโมเดลเสียงที่บันทึกเก็บไว้ให้มากที่สุด ก็จะมีผลทำให้ค่าความถูกต้องเพิ่มมากขึ้น

ในส่วนของการหาค่าพลังงาน การเลือกค่าความยาวของการวินโดวส์ เมื่อค่าความยาวเพิ่มขึ้น ก็จะทำให้ได้รูปพลังงานที่ราบเรียบ(Smooth)มากขึ้น

ในส่วนของการแยกสัญญาณเสียงพูดออกจากเสียงเงียบ การเลือกค่าระดับพลังงานที่เป็นเสียงเงียบเพื่อแยกคำที่ต้องการออกมา จะต้องทำการเลือกโดยหาค่าพลังงานของเสียงเงียบ(Noise)ให้เหมาะสมก็จะทำให้ความถูกต้องเพิ่มมากขึ้น เมื่อได้ช่วงของเสียงที่ต้องการแล้ว รอยต่อระหว่างคำพูดที่ตัดออกมาจะต้องนำไปเวทค่าพารามิเตอร์ ซึ่งจะช่วยลดข้อผิดพลาดที่เกิดจากรอยต่อของเฟรมซึ่งทำให้ค่าสัมประสิทธิ์ถูกต้องมากยิ่งขึ้น

ในส่วนของการหาค่าสัมประสิทธิ์เชิงปหสตรัม และค่าสัมประสิทธิ์การประมาณเชิงเส้น จะใช้จำนวนเท่ากันคืออย่างละ 12 ตัว ซึ่งถ้าเพิ่มค่าสัมประสิทธิ์มากขึ้น การประมาณก็จะมีค่าความถูกต้องมากขึ้น แต่ก็ใช้เวลาในการคำนวณมากขึ้นตามไปด้วย

การเขียนโปรแกรมด้วย Visual C++ ทำให้มีความสะดวกในการแก้ไขโปรแกรม โดยที่โปรแกรมสามารถที่จะแยกออกเป็นส่วน ๆ ได้ และหน้าจอการใช้งานจะมีความสวยงาม การจัดรูปแบบการทำงานที่ดี และขนาดของโปรแกรมกับไฟล์ที่ใช้ในการสร้างจะมีขนาดเล็กกว่าโปรแกรมอื่น จึงทำให้ใช้พื้นที่ในฮาร์ดดิสก์(Hard disk)น้อย อีกทั้งการคำนวณสามารถทำได้เร็วอีกด้วย

5.2 ข้อแนะนำ และปัญหาที่พบ

เสียงที่บันทึกจากคู่สายโทรศัพท์ จะมีสัญญาณรบกวนซึ่งถ้ามากเกินไปจะทำให้การทำงานผิดพลาดมากขึ้น ดังนั้นจึงควรทำการลดสัญญาณรบกวนต่าง ๆ ก่อนเพื่อให้ได้เสียงที่สมบูรณ์ที่สุด ระดับความดังของเสียงที่ต้องการบันทึกไม่ควรมีความแรงเกินไปจนทำให้เกิดการขลิบสัญญาณ ซึ่งจะทำให้สัญญาณเสียงที่ได้ไม่สามารถนำมาทำการบันทึกได้ คุณภาพของการ์ดเสียงที่ดีก็จะช่วยในการบันทึกเสียงที่มีความถูกต้องสมบูรณ์ และช่วยลดเสียงรบกวนในสัญญาณที่บันทึกได้

ในส่วนการโปรแกรม ควรใช้ความรอบคอบและความเป็นระเบียบในการโปรแกรม เพื่อลดความผิดพลาด และเมื่อพบปัญหาก็จะสามารถแก้ปัญหาได้ง่ายขึ้น ในการวิเคราะห์บางส่วนเราสามารถใช้โปรแกรมอื่นๆ มาช่วยในการวิเคราะห์ได้ เช่น โปรแกรม Matlab จะทำให้การโปรแกรมง่ายขึ้น และจากการทดลองที่ผ่านมา พบว่าปัญหาที่สำคัญอย่างหนึ่งก็คือ กรณีที่เสียงที่ต้องนำไปวิเคราะห์ในบางคำมีการออกเสียงที่คล้ายคลึงกันมาก เช่น เสียงพูดของคำว่า “สอง” และ “สาม” ในการวิเคราะห์คำพูดอาจเกิดการตัดสินใจที่ผิดพลาดขึ้นได้ ดังนั้นขั้นตอนในการทำโมเดลเสียงที่มีความคล้ายคลึงกันจึงต้องพยายามออกเสียงให้มีลักษณะที่แตกต่างกันให้มากที่สุด แต่ก็ต้องไม่ทำให้คำพูดนั้นผิดเพี้ยนไปด้วย วิธีการนี้จะทำให้โปรแกรมทำการวิเคราะห์ได้ถูกต้องมากยิ่งขึ้น และการออกเสียงนี้ก็ควรควบคุมให้มีเสียงรบกวน เช่น เสียงของลมหายใจ หรือเสียงจากภายนอกอื่นๆ ที่ไม่เกี่ยวข้องออกไปให้น้อยที่สุด

5.3 แนวทางการพัฒนา

ในการที่จะพัฒนาต่อไปในอนาคตก็ควรที่จะพัฒนาการทำงานให้สามารถสั่งงานโดยไม่ขึ้นกับผู้พูด และสามารถใช้คำพูดสั่งงานแบบต่อเนื่องได้ หรือนำไปใช้ในระบบรักษาความปลอดภัย เช่น การตรวจสอบว่าเป็นบุคคลใด โดยใช้เสียงที่มีอยู่เป็นตัวอ้างอิง และการพัฒนาอัลกอริทึมให้ใช้เวลาในการประมวลผลให้น้อยที่สุดเพื่อที่จะสามารถนำไปใช้กับไมโครคอนโทรลเลอร์ได้ก็จะทำให้สามารถประยุกต์ใช้งานได้หลากหลายมากยิ่งขึ้น อีกทั้งถ้าหากโปรแกรมสามารถทำการวิเคราะห์เสียงพูดที่มีคำพูดหลาย ๆ คำติดต่อกันได้อย่างถูกต้อง ก็จะทำให้การนำไปใช้งานสามารถทำได้กว้างมากยิ่งขึ้น แต่ทั้งนี้การวิเคราะห์ก็ควรขึ้นอยู่กับความถูกต้องเป็นสำคัญ



ขอรหัสโค้ดส่วนไมโครคอนโทรลเลอร์

```

;-----
; PROJECT REMOTE TELEPHONE VOICE COMMANDER
;   MPU AT89S53
;-----
;DEFINEPORT
; PO EQU 80H
;-----
;MAIN PROGRAM
;-----
INITIAL:  ORG 0000H
          MOV P0,#00000000B          ;INITIAL PORT 0
          MOV P2,#00000000B
          ACALL START
MAIN:     LCALL INIT_RS232
          LCALL CONT
          ACALL MAIN
;-----
START:    MOV R0,#03H                ;START DELAY TIME
LOOP1:    SETB P2.0
          LCALL DELAY_500mS
          CLR P2.0
          LCALL DELAY_500mS
          DJNZ R0,LOOP1
          SETB P2.0
          AJMP MAIN
;-----
;INITIAL RECEIVE DATA FROM RxD SEND CONTROL CH.
;-----
INIT_RS232: MOV PCON,#00000000B
            MOV SCON,#01010000B          ;REN,MODE1
            MOV TMOD,#00100000B
            MOV TH1,#0FDH                ; 9,600 BIT/SEC
            SETB TR1
RECEIVE:   JNB RI,RECEIVE
            MOV A,SBUF
            CLR RI
            RET
;-----
;CONTROL CHANNEL ON-OFF EQU.
;-----
CONT:      MOV A,SBUF
            CLR RI
CH_1:      CJNE A,#41H,OFF_CH1
            SETB P0.0
            RET
OFF_CH1:   CJNE A,#51H,CH_2
            CLR P0.0
            RET
;-----
CH_2:      CJNE A,#42H,OFF_CH2
            SETB P0.1
            RET
OFF_CH2:   CJNE A,#52H,CH_3
            CLR P0.1
            RET
;-----
CH_3:      CJNE A,#43H,OFF_CH3
            SETB P0.2
            RET
OFF_CH3:   CJNE A,#53H,CH_4
            CLR P0.2
            RET
;-----
CH_4:      CJNE A,#44H,OFF_CH4
            SETB P0.3
            RET
OFF_CH4:   CJNE A,#54H,CH_5

```



```

CLR P0.3
RET
;-----
CH_5:    CJNE A,#45H,OFF_CH5
         SETB P0.4
         RET
OFF_CH5: CJNE A,#55H,CH_6
         CLR P0.4
         RET
;-----
CH_6:    CJNE A,#46H,OFF_CH6
         SETB P0.5
         RET
OFF_CH6: CJNE A,#56H,CH_7
         CLR P0.5
         RET
;-----
CH_7:    CJNE A,#47H,OFF_CH7
         SETB P0.6
         RET
OFF_CH7: CJNE A,#57H,CH_8
         CLR P0.6
         RET
;-----
CH_8:    CJNE A,#48H,OFF_CH8
         SETB P0.7
         RET
OFF_CH8: CJNE A,#58H,AA
         CLR P0.7
         RET
;-----
AA:      CJNE A,#49H,BB
         SETB P2.1
         RET
BB:      CJNE A,#59H,DD ;
         CLR P2.1
         RET
;-----
DD:      AJMP MAIN
;-----
;DELAY 10ms,1S
;-----
DELAY_500mS: MOV 5,#50
DELAY_10mS:  MOV 7,#010
DELAY1:     MOV 6,#0E6H
DELAY2:     NOP
            NOP
            DJNZ R6,DELAY2
            DJNZ R7,DELAY1
            DJNZ R5,DELAY_10mS
            RET
            END

```

Speech.cpp

```
/*-----
Speech.C -- Speech Recognition
(c) Wichet On-rampan, 2000
-----*/

#include "StdAfx.h"
#include "windows.h"
#include "resource.h"
#include "Speech.h"
#include "Tty.h"
#include "mmsystem.h"
#include "assert.h"
#include "winbase.h"
#include "Math.h"
#include "stdlib.h"
#include "process.h"
#include "time.h"
DWORD BaudTable[] =
{
    CBR_110, CBR_300, CBR_600, CBR_1200, CBR_2400,
    CBR_4800, CBR_9600, CBR_14400, CBR_19200, CBR_38400,
    CBR_56000, CBR_128000, CBR_256000
};

unsigned char ParityTable[] =
{
    NOPARITY, EVENPARITY, ODDPARITY, MARKPARITY, SPACEPARITY
};

unsigned char StopBitsTable[] =
{
    ONESTOPBIT, ONE5STOPBITS, TWOSTOPBITS
};

unsigned char DataBitsTable[] =
{
    5, 6, 7, 8
};

char ComNumber[4][5] = {"COM1", "COM2", "COM3", "COM4"};

struct Status
{
    DWORD StatusParity;
    DWORD StatusBits;
    DWORD StatusData;
    DWORD StatusBaud;
    DWORD StatusCom;
}Modem, Serial;

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
BOOL CALLBACK PreferencesDlgProc (HWND, UINT, WPARAM, LPARAM);
BOOL CALLBACK LPCDlgProc (HWND hDlg, UINT iMsg, WPARAM wParam, LPARAM lParam);
BOOL NEAR SettingsDlgInit( HWND ) ;
VOID NEAR FillComboBox( HINSTANCE hInstance, HWND hCtrlWnd, int nIDString,
    DWORD NEAR *npTable, WORD wTableLen,
    DWORD dwCurrentSetting );

BOOL NEAR OpenConnect(HWND hwnd);
DWORD RecordWaveFile( HWND hwnd, DWORD dwMilliseconds, char* pstrName );
BOOL WriteABuffer(char *Com, char *lpBuf, DWORD dwToWrite, HWND hwnd);
BOOL PopFileOpenDlg (HWND hwnd, PSTR pstrFileName, PSTR pstrTitleName);
BOOL Initial();
BOOL CurrentPre(HWND hDlg);
BOOL InitialSerial(HANDLE hCom, DCB dcb);
BOOL InitialModem(HANDLE hCom, DCB dcb);
void DetectRing(PVOID pvoid);
BOOL Capture(HWND hwnd, int x, int y);
BOOL CALLBACK WaveDlgProc (HWND hDlg, UINT iMsg, WPARAM wParam, LPARAM lParam);
BOOL PaintWave(HWND hDlg, HDC hdcWave);
BOOL OpenWave(HWND hwnd, HANDLE waveDataBlock);
```



```

void PopFileInitialize (HWND hwnd ,char szFilter[]);
BOOL PopFileSaveWave(HWND hDlg,PSTR pstrFileName,PSTR pstrTitleName);
void WriteRecord(char *ptr,DSP_FILE *dsp_info);
DSP_FILE *OpenWrite(char *file_name,int type,unsigned short int records,
                    unsigned short int rec_len);
DSP_FILE *OpenRead(char *file_name);
void ReadRecord( char *ptr,DSP_FILE *dsp_info);
BOOL SaveWaveDat(PSTR pstrFileName);
void fir_filter_array(float *in,float *out,int in_len,FILTER *fir);
BOOL CALLBACK EnergyDlgProc(HWND hDlg,UINT iMsg,WPARAM wParam,LPARAM lParam);
BOOL OpenDat(PSTR pstrFileName);
BOOL SaveDat(PSTR pstrFileName,float *data,long lDataSize);
BOOL CALLBACK ZeroDlgProc(HWND hDlg,UINT iMsg,WPARAM wParam,LPARAM lParam);
float Sgn(float Datain);
void Swav2acf(float *wav, int nwav, float *acf, int nacf);
float Sacf2lpc(float *acf, int nacf, float *lpc, float *ref);
float *SmkHammwindow(int nitem);
void SzeroMeanHammWindow(float *wave, int nitem, float *window);
void Slpc2cep(float *lpc, int nlpc, float *cep, int ncep);
BOOL CALLBACK RecogDlgProc(HWND hDlg,UINT iMsg,WPARAM wParam,LPARAM lParam);
void **Smatrix(size_t nitem0, size_t nitem1, size_t size);
float Sdtw(float **unkTemp, int nunkTemp, float **refTemp, int nrefTemp,
           int tempSize);
float Sl2norm(float *vec0, float* vec1, int nitem);
void SHammwindow(float *wave, int nitem, float *window);
BOOL CALLBACK EndpointDlgProc(HWND hDlg,UINT iMsg,WPARAM wParam,LPARAM lParam);
float Sl1kediag(float* vec, float* mean, float* var, int nitem);
BOOL CALLBACK CEPDlgProc(HWND hDlg,UINT iMsg,WPARAM wParam,LPARAM lParam);
BOOL CALLBACK EndOfSpeechDlgProc(HWND hDlg,UINT iMsg,WPARAM wParam,LPARAM
lParam);
END *EndOfSpeech(float *Wave,unsigned int Length,float Ratio,unsigned int
Time_sample);
DTW_DATA *Dtw(unsigned int nfile,char* path[],int nChannel,char *file_name);
BOOL PaintWndProc();
BOOL Train(HWND hwnd);
BOOL ReportStatus();
DWORD RecordWave( HWND hwnd,DWORD dwMilliSeconds,char* lpstrName );
BOOL Send_command_1(HWND hwnd,DTW_DATA *dtw_info);
BOOL Energy(float *Data_in,float *Data_out,int lDataSize);
DTW_DATA* Recognition_block1(HWND hwnd);
DTW_DATA *Recognition_block_2( HWND hwnd);
BOOL Send_command_1_2(HWND hwnd,DTW_DATA *dtw_info);
BOOL Write_on_command(HWND hwnd);
BOOL Write_off_command(HWND hwnd);
BOOL Reset_fSelect();

char szAppName[] = "IDR_MENU" ,szTemp[100];
static HINSTANCE ghInst;
static HWND hDlg;
static OPENFILENAME ofn ;
static char szFileName[_MAX_PATH],szTitleName[_MAX_PATH+_MAX_EXT];
static HBITMAP hBitmapMain,hBitmapLine,hBitmapManual,
                hBitmapOn,hBitmap1,hBitmap2,hBitmap3,hBitmap4
                ,hBitmap5,hBitmap6,hBitmap7,hBitmap8,hBitmapOnHook
                ,hBitmapOffHook,hBitmapModem,hBitmapRing;

static HDC
static HDC
PAINTSTRUCT ps;
HBRUSH hBrushBackground;
static BOOL fStateManual,fStateOn,fState1,fState2,fState3,fState4
                ,fState5,fState6,fState7,fState8,fStateOnHook,fStateOffHook
                ,fStateRing,fStateModem,fStateWaveData,
                fSelect_channel1,fSelect_channel2,fSelect_channel3,
                fSelect_channel4,fSelect_channel5,fSelect_channel6,
                fSelect_channel7,fSelect_channel8;

static char * cOffHook={"ath1\r"};
static char * cOnHook={"ath0\r"};
static char * cOn1={"A"};
static char * cOff1={"Q"};
static char On[9][2]={"A","B","C","D","E","F","G","H","I"};
static char Off[9][2]={"Q","R","S","T","U","V","W","X","Y"};

//-----Wave Data-----//

```



```

static long lDataSize ;
static LPBYTE pWave;
static HANDLE waveDataBlock;
static float *out_temp,*DataIn,*DataOut,*DataPower,*DataZero,
              *Htemp,temp01,temp00,temp02,temp03;

static BOOL fRING,iCheckComm,iCheckThread=0,fStatusConnect=0
            ,fStatusTraining=0,fCheckDisconnect=0,iCheck_record=0
            ,iCheck_error=0,iCheck_dtw=0,bCheck_block_1=0,bCheck_block_2=0
            ,bCheck_connect_loop=0,iCheck_block_error;

//-----
static HMENU      hMenu ;
static int iCheck_pattern=0,iCount_block=0;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    PSTR szCmdLine, int iCmdShow)
{
    HWND      hwnd ;
    MSG        msg ;
    WNDCLASSEX wndclass ;
    ghInst=hInstance;

    wndclass.cbSize          = sizeof (wndclass) ;
    wndclass.style           = CS_HREDRAW | CS_VREDRAW ;
    wndclass.lpfnWndProc     = WndProc ;
    wndclass.cbClsExtra      = 0 ;
    wndclass.cbWndExtra      = 0 ;
    wndclass.hInstance       = hInstance ;
    wndclass.hIcon           = LoadIcon (hInstance,MAKEINTRESOURCE(IDI_ICON1)) ;
    wndclass.hCursor         = LoadCursor (NULL, IDC_ARROW) ;
    wndclass.hbrBackground   = (HBRUSH) GetStockObject (LTGRAY_BRUSH) ;
    ;//;hBrushBackground;//////hBrush; //
    wndclass.lpszMenuName    = szAppName ;
    wndclass.lpszClassName   = szAppName ;
    wndclass.hIconSm         = LoadIcon (hInstance,MAKEINTRESOURCE(IDI_ICON3)) ;

    RegisterClassEx (&wndclass) ;

    hwnd = CreateWindow (szAppName, "Telephone Control Electrical Appliance By
Voice",
                        WS_CAPTION|WS_SYSMENU|WS_MINIMIZEBOX,
                        CW_USEDEFAULT, CW_USEDEFAULT,
                        406,243,
                        NULL, NULL, hInstance, NULL) ;

    ShowWindow (hwnd, iCmdShow) ;
    UpdateWindow (hwnd) ;

    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
    }
    return msg.wParam ;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
    static HWND hEdit;
    static PARAMS params ;
    static HANDLE hEvent ;
    static int cxBlock,cyBlock;
    static HINSTANCE hInstance;
    RECT rect;
    DTW_DATA *dtw_info_1,*dtw_info_12;

```



```

static int x,y,iOpenResultWav;
END *End_wave;
static float *Wnd_data_out,*Wnd_bin1,*Wnd_bin2;
int j;

switch (iMsg)
{
case WM_CREATE:

    Initial();

    params.hwnd=hwnd;
    params.hEvent = hEvent ;
    params.bContinue=FALSE;
    iCheck_pattern=1;
    dtw_info_1 = (DTW_DATA *) malloc(sizeof(DTW_DATA));
    dtw_info_12 = (DTW_DATA *) malloc(sizeof(DTW_DATA));
    End_wave = (END*) malloc(sizeof(END));
    return 0;

case WM_RING:

    rect.left=68;
    rect.top = 99;
    rect.right=128;
    rect.bottom=119;
    InvalidateRect(hwnd,&rect,FALSE);
    return 0;

case WM_PAINT:

    hdc = BeginPaint(hwnd,&sps);
    hMemDC=CreateCompatibleDC(hdc);
    PaintWndProc();
    DeleteDC(hMemDC);
    EndPaint(hwnd,&sps);
    return 0;

case WM_SIZE:

    cxBlock = LOWORD (lParam);
    cyBlock = HIWORD (lParam);
    return 0;

case WM_MOVE:
    return 0;

case WM_LBUTTONDOWN:

    x=LOWORD (lParam);
    y=HIWORD (lParam);
    Capture(hwnd,x,y);

    return 0;

case WM_CHAR:

    return 0;

case WM_END_RING:

    if(fStatusConnect)
    {
        WriteABuffer("COM1",On[8],1,hwnd);
        WriteABuffer("COM3",cOnHook,5,hwnd);
        PlaySound(Wave_connect[0],NULL,SND_FILENAME);
    while(!bCheck_connect_loop)
    {
        while(!bCheck_block_1)
        {
            dtw_info_1=Recognition_block1(hwnd);

            /*char Wnd_char[40];

```

```

sprintf(Wnd_char, "%d %2.2e %d %2.2e", dtw_info_1->Chanel_1, dtw_info_1->distance_1, dtw_info_1->Chanel_2, dtw_info_1->distance_2);
MessageBox(hwnd, Wnd_char, szAppName, MB_ICONINFORMATION|MB_OK);
*/
if (!Send_command_1(hwnd, dtw_info_1))
{
    WriteABuffer("COM3", cOnHook, 5, hwnd);
    PlaySound(Wave_connect[3], NULL, SND_FILENAME);
    iCheck_error=1;
    iCount_block++;
    if (iCount_block>=3)
    {
        bCheck_block_1=1;
        iCount_block=0;
        iCheck_block_error=1;
        bCheck_connect_loop=1;
    }
}
else
    bCheck_block_1=1;
}
iCount_block=0;
while (!bCheck_block_2 && !iCheck_block_error)
{
    dtw_info_12=Recognition_block_2(hwnd);
    /* char Wnd_char[40];
    sprintf(Wnd_char, "%d %2.2e %d %2.2e", dtw_info_12->Chanel_1, dtw_info_12->distance_1, dtw_info_12->Chanel_2, dtw_info_12->distance_2);
    MessageBox(hwnd, Wnd_char, szAppName, MB_ICONINFORMATION|MB_OK);
    */
    if (!Send_command_1_2(hwnd, dtw_info_12))
    {
        WriteABuffer("COM3", cOnHook, 5, hwnd);
        PlaySound(Wave_connect[5], NULL, SND_FILENAME);
        iCount_block++;
        if (iCount_block>=3)
        {
            bCheck_block_2=1;
            iCount_block=0;
            bCheck_connect_loop=1;
        }
    }
    else
        bCheck_block_2=1;

    Reset_fSelect();
    InvalidateRect(hwnd, NULL, FALSE);
    SendMessage(hwnd, WM_PAINT, 0, 0);
}

bCheck_block_2=0;
bCheck_block_1=0;
iCheck_block_error=0;

}

bCheck_connect_loop=1;
sndPlaySound(Wave_connect[6], SND_SYNC|SND_FILENAME);
WriteABuffer("COM3", cOnHook, 5, hwnd);
WriteABuffer("COM1", Off[8], 1, hwnd);
}

bCheck_connect_loop=0;
if (fStatusTraining)
{
    WriteABuffer("COM1", On[8], 5, hwnd);
    WriteABuffer("COM3", cOnHook, 5, hwnd);
    PlaySound(Wave_name[0], NULL, SND_FILENAME);

    Train(hwnd);
    PlaySound(Wave_name[2], NULL, SND_FILENAME);
}

```



```

        WriteABuffer("COM1",Off[8],5,hwnd);
        fStatusTraining=0;
    }

    return 0;

case WM_COMMAND :

    hMenu = GetMenu (hwnd) ;

    switch (LOWORD (wParam))
    {
        case ID_FILE_CONNECT :
            if(iCheckComm==0)
            {
                _beginthread(DetectRing,0,&params);
            }
            fStateOn=1,fStateManual=0;
            InvalidateRect(hwnd,NULL,FALSE);
            EnableMenuItem (hMenu, ID_FILE_CONNECT,MF_GRAYED) ;
            EnableMenuItem (hMenu, ID_FILE_TRAINING,MF_GRAYED) ;
            EnableMenuItem (hMenu, ID_FILE_DISCONNECT,MF_ENABLED) ;
            fStatusConnect=1;
            fStatusTraining=0;
            iCheckComm=1;

            //

            return 0;

        case ID_FILE_DISCONNECT :
            iCheckComm=0;
            params.bContinue=FALSE;
            fStateOn=0;
            InvalidateRect(hwnd,NULL,FALSE);
            EnableMenuItem (hMenu, ID_FILE_DISCONNECT, MF_GRAYED) ;
            EnableMenuItem (hMenu, ID_FILE_CONNECT, MF_ENABLED) ;
            EnableMenuItem (hMenu, ID_FILE_TRAINING, MF_ENABLED);
            fStatusConnect=0;fStatusTraining=0;
            return 0;

        case ID_FILE_TRAINING:
            EnableMenuItem (hMenu,ID_FILE_TRAINING , MF_GRAYED) ;
            EnableMenuItem (hMenu, ID_FILE_CONNECT, MF_GRAYED ) ;
            EnableMenuItem (hMenu, ID_FILE_DISCONNECT, MF_ENABLED);
            fStatusTraining=1;
            fStatusConnect=0;
            if(iCheckComm==0)
            {
                _beginthread(DetectRing,0,&params);
            }
            iCheckComm=1;
            return 0;

        case ID_FILE_EXIT :
            WriteABuffer("COM1",Off[8],5,hwnd);
            WriteABuffer("COM3",cOnHook,5,hwnd);
            _endthread();
            SendMessage (hwnd, WM_CLOSE, 0, 0L) ;

            return 0 ;

        case ID_OPTION_PREFERENCES:
            //MessageBox(hwnd,"MODEM",szAppName
            ,MB_ICONINFORMATION|MB_OK);
            DialogBox(hInstance,MAKEINTRESOURCE(IDD_PREFER),hwnd
            ,PreferencesDlgProc);
            return 0;

        case ID_HELP_ABOUT:
            OpenConnect(hwnd);
            return 0;

        case ID_BUILD_CEP:
            DialogBox(hInstance,MAKEINTRESOURCE(IDD_CEP),

```

```

    hwnd,CEPDlgProc);
        return 0;

    case ID_BUILD_LPC:
        DialogBox(hInstance,MAKEINTRESOURCE(IDD_LPC12),hwnd,LPCDlgProc);
        return 0;
    case ID_WAVE_EDITWAVE:
        DialogBox(hInstance,MAKEINTRESOURCE(IDD_WAVE),hwnd
        ,WaveDlgProc);
        return 0;

    case ID_SEGMENTATION_ENERGY:
        DialogBox(hInstance,MAKEINTRESOURCE(IDD_ENERGY),hwnd
        ,EnergyDlgProc);
        return 0;

    case ID_SEGMENTATION_ZEROCROSSING:
        DialogBox(hInstance,MAKEINTRESOURCE(IDD_ZERO),hwnd,
        ZeroDlgProc);
        return 0;

    case ID_RECOGNIZATION_MANUAL:
        DialogBox(hInstance,MAKEINTRESOURCE(IDD_RECOG),hwnd
        ,RecogDlgProc);
        return 0;

    case ID_ENDPOINT:
        DialogBox(hInstance,MAKEINTRESOURCE(IDD_ENDPOINT),hwnd
        ,EndpointDlgProc);
        return 0;

    case ID_END:
        DialogBox(hInstance,MAKEINTRESOURCE(IDD_END),hwnd
        ,EndOfSpeechDlgProc);
        return 0;
    }
    break;

    case WM_DESTROY :
        PostQuitMessage (0) ;
        return 0 ;
    }
}
return DefWindowProc (hwnd, iMsg, wParam, lParam) ;
}

```

```

BOOL Send_command_1_2(HWND hwnd ,DTW_DATA *dtw_info)
{
    int iResult=0;

    if(dtw_info->distance_1<40.0)
        iResult=dtw_info->Chanel_1;

    switch (iResult)
    {
    case 0:
        // MessageBox(hwnd,"Not
Recognition",szAppName,MB_ICONINFORMATION|MB_OK);
        return FALSE;
        break;

    case 9:
        Write_on_command(hwnd);
        InvalidateRect(hwnd,NULL,FALSE);
        // SendMessage(hwnd,WM_PAINT,0,0);
        break;

    case 10:
        Write_off_command(hwnd);

```



```

// MessageBox(hwnd,"off",szAppName,MB_ICONINFORMATION|MB_OK);
break;
default :
    return FALSE;
    InvalidateRect(hwnd,NULL,FALSE);
    // SendMessage(hwnd,WM_PAINT,0,0);
}

return TRUE;
}

BOOL Write_off_command(HWND hwnd)
{
    if(fSelect_channel1==1)
    {
        if(fState1==1)
        {
            fState1=0;
            WriteABuffer(ComNumber[Serial.StatusCom],Off[0],1,hwnd);
        }
    }

    if(fSelect_channel2==1)
    {
        if(fState2==1)
        {
            fState2=0;
            WriteABuffer(ComNumber[Serial.StatusCom],Off[1],1,hwnd);
        }
    }

    if(fSelect_channel3==1)
    {
        if(fState3==1)
        {
            fState3=0;
            WriteABuffer(ComNumber[Serial.StatusCom],Off[2],1,hwnd);
        }
    }

    if(fSelect_channel4==1)
    {
        if(fState4==1)
        {
            fState4=0;
            WriteABuffer(ComNumber[Serial.StatusCom],Off[3],1,hwnd);
        }
    }

    if(fSelect_channel5==1)
    {
        if(fState5==1)
        {
            fState5=0;
            WriteABuffer(ComNumber[Serial.StatusCom],Off[4],1,hwnd);
        }
    }

    if(fSelect_channel6==1)
    {
        if(fState6==1)
        {
            fState6=0;
            WriteABuffer(ComNumber[Serial.StatusCom],Off[5],1,hwnd);
        }
    }

    if(fSelect_channel7==1)
    {
        if(fState7==1)
        {
            fState7=0;
            WriteABuffer(ComNumber[Serial.StatusCom],Off[6],1,hwnd);
        }
    }

    if(fSelect_channel8==1)
    {

```

```

        if(fState8==1)
        {
            fState8=0;
            WriteABuffer(ComNumber[Serial.StatusCom],Off[7],1,hwnd);
        }
    }

    return TRUE;
}

BOOL Write_on_command(HWND hwnd)
{
    if(fSelect_channel1==1)
    {
        if(fState1==0)
        {
            fState1=1;
            WriteABuffer(ComNumber[Serial.StatusCom],On[0],1,hwnd);
        }
    }
    if(fSelect_channel2==1)
    {
        if(fState2==0)
        {
            fState2=1;
            WriteABuffer(ComNumber[Serial.StatusCom],On[1],1,hwnd);
        }
    }
    if(fSelect_channel3==1)
    {
        if(fState3==0)
        {
            fState3=1;
            WriteABuffer(ComNumber[Serial.StatusCom],On[2],1,hwnd);
        }
    }
    if(fSelect_channel4==1)
    {
        if(fState4==0)
        {
            fState4=1;
            WriteABuffer(ComNumber[Serial.StatusCom],On[3],1,hwnd);
        }
    }
    if(fSelect_channel5==1)
    {
        if(fState5==0)
        {
            fState5=1;
            WriteABuffer(ComNumber[Serial.StatusCom],On[4],1,hwnd);
        }
    }
    if(fSelect_channel6==1)
    {
        if(fState6==0)
        {
            fState6=1;
            WriteABuffer(ComNumber[Serial.StatusCom],On[5],1,hwnd);
        }
    }
    if(fSelect_channel7==1)
    {
        if(fState7==0)
        {
            fState7=1;
            WriteABuffer(ComNumber[Serial.StatusCom],On[6],1,hwnd);
        }
    }
    if(fSelect_channel8==1)
    {
        if(fState8==0)
        {
            fState8=1;
            WriteABuffer(ComNumber[Serial.StatusCom],On[7],1,hwnd);
        }
    }
}

```



```

    }

    return TRUE;
}
BOOL Send_command_1(HWND hwnd ,DTW_DATA *dtw_info)
{
    int iResult=0;

    if(dtw_info->distance_1<40.0)
        iResult=dtw_info->Chanel_1;

    switch (iResult)
    {
    case 0:
        // MessageBox(hwnd,"Not
Recognition",szAppName,MB_ICONINFORMATION|MB_OK);
        return FALSE;
        break;

    case 1:
        fSelect_channel1=1;
        break;

    case 2:
        fSelect_channel2=1;
        break;

    case 3:
        fSelect_channel3=1;
        break;

    case 4:
        fSelect_channel4=1;
        break;

    case 5:
        fSelect_channel5=1;
        break;

    case 6:
        fSelect_channel6=1;
        break;

    case 7:
        fSelect_channel7=1;
        break;

    case 8:
        fSelect_channel8=1;
        break;

    default :
        return FALSE;

    }

    return TRUE;
}

BOOL Reset_fSelect()
{
    fSelect_channel1=0,fSelect_channel2=0,fSelect_channel3=0
    ,fSelect_channel4=0,fSelect_channel5=0,fSelect_channel6=0
    ,fSelect_channel7=0,fSelect_channel8=0;
    return TRUE;
}

DTW_DATA* Recognition_block1(HWND hwnd)
{
    END *End_wave;
    float *Wnd_data_out,*Wnd_bin1,*Wnd_bin2;

```

```

int i;
iCheck_dtw=0;

DTW_DATA *dtw_info;

dtw_info = (DTW_DATA*) malloc(sizeof(DTW_DATA));
End_wave = (END*) malloc(sizeof(END));

WriteABuffer("COM1",On[8],1,hwnd);
WriteABuffer("COM3",cOnHook,5,hwnd);

ReportStatus();

// PlaySound("end.wav",NULL,SND_FILENAME);
// PlaySound(Wave_connect[2],NULL,SND_FILENAME);
// WriteABuffer("COM3",cOffHook,5,hwnd);
// RecordWave(hwnd,1400,"Data_in.wav");
// ofn.lpstrFile="Data_in.wav";
// OpenWave(hwnd,waveDataBlock);
// SaveWaveDat("Data_in.dat");
// OpenDat("Data_in.dat");
// DataOut=(float *)calloc(1DataSize,sizeof(float));
// Energy(out_temp,DataOut,(int)1DataSize);
// SaveDat("Data_in.eng",DataOut,1DataSize);
// OpenDat("Data_in.eng");
// End_wave=EndOfSpeech(DataOut,1DataSize,(float)
0.03,1);
if(!End_wave->fStatus_start||!End_wave->fStatus_end)
{
/* char Wnd_char[40];
wsprintf(Wnd_char,"%d",End_wave->Start);

MessageBox(hwnd,Wnd_char,szAppName,MB_ICONINFORMATION|MB_OK);
*/
Wnd_data_out = (float *) calloc((End_wave->
Length)+256, sizeof(float*));
Wnd_bin1=Wnd_data_out+128;
Wnd_bin2=out_temp+End_wave->Start;
for(i=0;i<(int)End_wave->Length;i++)
*Wnd_bin1++=*Wnd_bin2++;

SaveDat("Data_in.end",Wnd_data_out,(End_wave->
Length)+256);

free(Wnd_data_out);
free(End_wave);

dtw_info=Dtw(1+(iCheck_pattern*10)+30,Mod_5,1,"Data_in.end");

/* char Wnd_char[40];
sprintf(Wnd_char,"%d %2.2e %d %2.2e ",dtw_info->
Chanel_1
,dtw_info->distance_1,dtw_info->
Chanel_2,dtw_info->distance_2);

MessageBox(hwnd,Wnd_char,szAppName,MB_ICONINFORMATION|MB_OK);
*/

}

return (dtw_info);
}

DTW_DATA *Recognition_block_2( HWND hwnd)
{
DTW_DATA *dtw_info_2;
END *End_wave_2;
float *Wnd_data_out_2,*Wnd_bin1,*Wnd_bin2;
int i;
iCheck_dtw=1;
dtw_info_2 = (DTW_DATA*) malloc(sizeof(DTW_DATA));
End_wave_2 = (END*) malloc(sizeof(END));

```



```

WriteABuffer("COM3",cOnHook,5,hwnd);

/*      if(!iCheck_error)
        {
            iCheck_error=0;
            PlaySound(Wave_connect[0],NULL,SND_FILENAME);
            ReportStatus();
        }*/
PlaySound(Wave_connect[4],NULL,SND_FILENAME);
WriteABuffer("COM3",cOffHook,5,hwnd);
RecordWave(hwnd,1400,"Data_i2.wav");
ofn.lpstrFile="Data_i2.wav";
OpenWave(hwnd,waveDataBlock);
SaveWaveDat("Data_i2.dat");
OpenDat("Data_i2.dat");
DataOut=(float *)calloc(1DataSize,sizeof(float));
Energy(out_temp,DataOut,(int)1DataSize);
SaveDat("Data_i2.eng",DataOut,1DataSize);
//      OpenDat("Data_in.eng");
End_wave_2=EndOfSpeech(DataOut,1DataSize,(float)
0.03,1);
if(!End_wave_2->fStatus_start||!End_wave_2->fStatus_end)
{
    /*      char Wnd_char[40];
        wsprintf(Wnd_char,"%d",End_wave_2->Start);

        MessageBox(hwnd,Wnd_char,szAppName,MB_ICONINFORMATION|MB_OK);
        */
        Wnd_data_out_2 = (float *) calloc((End_wave_2->
Length)+256, sizeof(float*));
        Wnd_bin1=Wnd_data_out_2+128;
        Wnd_bin2=out_temp+End_wave_2->Start;
        for(i=0;i<(int)End_wave_2->Length;i++)
            *Wnd_bin1++=*Wnd_bin2++;

        SaveDat("Data_i2.end",Wnd_data_out_2,(End_wave_2->
Length)+256);

        free(Wnd_data_out_2);
        free(End_wave_2);

        dtw_info_2=Dtw(1+(iCheck_pattern*10),Mod_open_close,1,"Data_i2.end");

        /*      char Wnd_char[40];
            sprintf(Wnd_char,"%d %2.2e %d %2.2e ",dtw_info->
Chanel_1
            ,dtw_info->distance_1,dtw_info->
Chanel_2,dtw_info->distance_2);

            MessageBox(hwnd,Wnd_char,szAppName,MB_ICONINFORMATION|MB_OK);
            */
        }

        return (dtw_info_2);
    }

    }

    BOOL Train(HWND hwnd)
    {
        int i,j=1;

        for(i=0;i<(10*iCheck_pattern);i++)
        {
            WriteABuffer("COM3",cOnHook,5,hwnd);
            do
            {
                //WriteABuffer("COM3",cOnHook,5,hwnd);
                sndPlaySound(Wave_name[1],SND_SYNC |SND_FILENAME);
                sndPlaySound(Wave_channel[i],SND_SYNC
|SND_FILENAME);

                //sndPlaySound("end.wav",SND_SYNC |SND_FILENAME);
                WriteABuffer("COM3",cOffHook,5,hwnd);
            }
            while(1);
        }
    }
}

```

```

        }

while(RecordWaveFile(hwnd,1400,Model_name[i])&&!iCheck_record);
    if(iCheck_record)
    {
        iCheck_record=0;
        return TRUE;
    }
}

return TRUE;
}

BOOL CALLBACK PreferencesDlgProc(HWND hDlg,UINT iMsg,WPARAM wParam,LPARAM lParam)
{
    int Pre_temp;

    switch (iMsg)
    {
    case WM_INITDIALOG:
        SettingsDlgInit( hDlg);
        SendDlgItemMessage(hDlg,IDC_PRE_1,EM_SETCHECK,1,0L);

        return TRUE;

    case WM_COMMAND:
        switch(LOWORD (wParam))
        {
        case IDC_PRE_1:
        case IDC_PRE_2:
        case IDC_PRE_3:
            Pre_temp=wParam;
            CheckRadioButton(hDlg,IDC_PRE_1,IDC_PRE_3,wParam);
            iCheck_pattern=3-(IDC_PRE_3-Pre_temp);

            return TRUE;
        case IDOK:
            CurrentPre(hDlg);
            EndDialog(hDlg,0);
            return TRUE;

        case IDCANCEL:
            EndDialog(hDlg,0);
            return TRUE;

        }

    break;

    case WM_PAINT:
        break;

    }
    return FALSE;
}

BOOL CALLBACK LPCDlgProc(HWND hDlg,UINT iMsg,WPARAM wParam,LPARAM lParam)
{

```



```

char szBuffer[75];
char szFilterSaveEng[] = "Data Files (*.LPC)\0*.lpc\0" \
    "All Files (*.*)\0*.*\0\0" ;
char szFilterOpenDat[] = "Data Files (*.DAT)\0*.dat\0" \
    "All Files (*.*)\0*.*\0\0" ;
////////////////////////////////////

float *wwave, *acf, *lpc, *ref, *temp, *cep, *templ, *temp2;
temp=out_temp;
int framestep = 256;
int windowsize = 512;
int lporder = 12;
static int nsample, frame, nframe, i, check, iCheckOpen, iCheckLpc;

HDC hdcLPC;;

switch (iMsg)
{
case WM_INITDIALOG:

    return TRUE;

case WM_COMMAND:
    switch (LOWORD (wParam))
    {
    case IDC_LPC_SAVE:

        if (iCheckOpen==1&&iCheckLpc==1)
        {
            PopFileInitialize (hDlg , szFilterSaveEng);
            if (0!=PopFileSaveWave (hDlg,szFileName,szTitleName))
            {
                SaveDat (szFileName,DataOut,(lporder+1)*nframe);
                check=3;
                }MessageBeep(0);
            }
            else
            {
                if (iCheckOpen==0)check=0;
                if (iCheckLpc==0&&iCheckOpen==1)check=2;
            }
            InvalidateRect (hDlg,NULL,TRUE);
            return TRUE;

        case IDC_LPC_OPEN:

            PopFileInitialize (hDlg , szFilterOpenDat);
            if (0!=PopFileOpenDlg (hDlg,szFileName,szTitleName))
            {
                OpenDat (szFileName);
                iCheckOpen=1;
                check=2;
            }

            InvalidateRect (hDlg,NULL,TRUE);
            return TRUE;

        case IDC_LPC_LPC:

            ///////////////////////////////////
            if (iCheckOpen==1)
            {check=1;
                nsample=(int)lDataSize;
                wwave = (float *) calloc(windowsize, sizeof(*wwave));
                acf = (float *) calloc(lporder + 1, sizeof(*acf));
                lpc = (float *) calloc(lporder + 1, sizeof(*lpc));
                ref = (float *) calloc(lporder + 1, sizeof(*ref));
                cep = (float *) calloc(lporder + 1, sizeof(*cep));

                nframe = (nsample / framestep);
            }
        }
    }
}

```

```

DataOut = (float *) calloc((lporder + 1)*nframe+256, sizeof
(*ref));
templ=DataOut;
for(frame = 0; frame < nframe; frame++)
{
    float pred_res;
    SzeroMeanHammWindow(temp + (framestep*frame),
window size, wwave);
    Swav2acf(wwave, window size, acf, lporder + 1);
    pred_res = Sacf2lpc(acf, lporder, lpc, ref);
    cep[0] = (float)log(acf[0]);
    Slpc2cep(lpc, lporder, cep, lporder);
    temp2=lpc;
    for(i=0;i<lporder+1;i++)
    {
        *temp1++=*temp2++;
    }
}

free(acf);
free(lpc);
free(ref);
free(cep);
iCheckLpc=1;
}
else
    check=0;
    InvalidateRect(hDlg,NULL,TRUE);

    return TRUE;

case IDC_LPC_CANCEL:
    EndDialog(hDlg,0);
    return TRUE;
}
break;
case WM_PAINT:
    hdcLPC = BeginPaint(hDlg,&ps);
    SetBkMode(hdcLPC,TRANSPARENT);
    SelectObject(hdcLPC,GetStockObject(DEFAULT_GUI_FONT));
    SetTextAlign(hdcLPC,TA_BASELINE);

    if(check==0)
        TextOut(hdcLPC,120, 50,szBuffer,wsprintf(szBuffer," Click Open
to Browse Dat File "));
    if(check==1)
        TextOut(hdcLPC,120, 50,szBuffer,wsprintf(szBuffer,"..Click Save
Button To Save Lpc File.."));
    if(check==2)
    {
        TextOut(hdcLPC,120, 50,szBuffer,wsprintf(szBuffer,"..Please Press
LPC Button To Process.."));
        if(iCheckOpen==1)
            TextOut(hdcLPC,120,70,szBuffer,wsprintf(szBuffer,"
%s",szTitleName));
    }
    if(check==3)
    {
        TextOut(hdcLPC,120,70,szBuffer,wsprintf(szBuffer,"
%s",szTitleName));
        TextOut(hdcLPC,120, 50,szBuffer,wsprintf(szBuffer,"
..Process Completed.. "));
    }

    EndPaint(hDlg,&ps);

    break;

```



```

    }
    return FALSE;
}

```

```

BOOL CALLBACK CEPDlgProc(HWND hDlg,UINT iMsg,WPARAM wParam,LPARAM lParam)

```

```

{
    char szBuffer[75];
    char szFilterSaveEng[] = "Data Files (*.CEP)\0*.cep\0" \
        "All Files (*.*)\0*.*\0\0" ;
    char szFilterOpenDat[] = "Data Files (*.DAT)\0*.dat\0" \
        "All Files (*.*)\0*.*\0\0" ;
    ///////////////////////////////////

    float *wwave, *acf, *lpc, *ref,*temp,*cep,*templ,*temp2;
    temp=out_temp;
    int framestep = 256;
    int windowsize = 512;
    int lporder = 12;
    static int nsample, frame, nframe,i,check,iCheckOpen,iCheckCep;
    HDC hdcCEP;

    switch (iMsg)
    {
    case WM_INITDIALOG:
        return TRUE;

    case WM_COMMAND:
        switch(LOWORD (wParam))
        {
        case IDC_CEP_SAVE:
            if(iCheckOpen==1&&iCheckCep==1)
            {
                PopFileInitialize (hDlg , szFilterSaveEng);
                if(0!=PopFileSaveWave(hDlg,szFileName,szTitleName))
                {
                    SaveDat(szFileName,DataOut,(lporder+1)*nframe);
                    check=3;
                    }MessageBeep(0);
                }
            else
            {
                if(iCheckOpen==0)check=0;
                if(iCheckCep==0&&iCheckOpen==1)check=2;
            }

            InvalidateRect(hDlg,NULL,TRUE);
            return TRUE;

        case IDC_CEP_OPEN:
            PopFileInitialize (hDlg , szFilterOpenDat);
            if(0!=PopFileOpenDlg(hDlg,szFileName,szTitleName))
            {
                OpenDat(szFileName);
                iCheckOpen=1;
                check=2;
            }

            InvalidateRect(hDlg,NULL,TRUE);
            return TRUE;

        case IDC_CEP_CEP:
            ///////////////////////////////////
            if(iCheckOpen==1)
            {
                check=1;
                nsample=(int)lDataSize;
            }
        }
    }
}

```

```

wwave = (float *) calloc(windowSize, sizeof(*wwave));
acf = (float *) calloc(lporder + 1, sizeof(*acf));
lpc = (float *) calloc(lporder + 1, sizeof(*lpc));
ref = (float *) calloc(lporder + 1, sizeof(*ref));
cep = (float *) calloc(lporder + 1, sizeof(*cep));

nframe = (nsample / framestep);
DataOut = (float *) calloc((lporder + 1)*nframe+256, sizeof
(*ref));

temp1=DataOut;
for(frame = 0; frame < nframe; frame++)
{
    float pred_res;
    SzeroMeanHammWindow(temp + (framestep*frame),
windowSize, wwave);
    Swav2acf(wwave, windowSize, acf, lporder + 1);
    pred_res = Sacf2lpc(acf, lporder, lpc, ref);
    //cep[0] = (float)log(acf[0]);
    Slpc2cep(lpc, lporder, cep, lporder);
    temp2=cep;
    for(i=0;i<lporder+1;i++)
    {
        *temp1++=*temp2++;
    }
}

free(acf);
free(lpc);
free(ref);
free(cep);
iCheckCep=1;
}
else
    check=0;

InvalidateRect(hDlg,NULL,TRUE);

return TRUE;

case IDC_CEP_CANCEL:
    EndDialog(hDlg,0);
    return TRUE;
}
break;
case WM_PAINT:
    hdcCEP = BeginPaint(hDlg,&ps);
    SetBkMode(hdcCEP,TRANSPARENT);
    SelectObject(hdcCEP,GetStockObject(DEFAULT_GUI_FONT));
    SetTextAlign(hdcCEP,TA_BASELINE);

    if(check==0)
        TextOut(hdcCEP,120, 50,szBuffer,wsprintf(szBuffer,"      Click Open
to Browse Dat File  "));
    if(check==1)
        TextOut(hdcCEP,120, 50,szBuffer,wsprintf(szBuffer,"..Click Save
Button To Save Cep File.."));
    if(check==2)
    {
        TextOut(hdcCEP,120, 50,szBuffer,wsprintf(szBuffer,"..Please Press
CEP Button To Process.."));
        if(iCheckOpen==1)
            TextOut(hdcCEP,120, 70,szBuffer,wsprintf(szBuffer,"
%s
",szTitleName));
    }
    if(check==3)
    {
        TextOut(hdcCEP,120, 50,szBuffer,wsprintf(szBuffer,"
%s
",szTitleName));
        TextOut(hdcCEP,120, 70,szBuffer,wsprintf(szBuffer,"
..Process Completed..
"));
        EndPaint(hDlg,&ps);
        break;
    }
}

```



```

    }
    return FALSE;
}
END *EndOfSpeech(float *Wave,unsigned int Length,float Ratio,unsigned int
Time_sample)
{
    END *end_info;
    unsigned int Sample_start[10],Sample_end[10],Sample_max=0
        ,i=0,Number_start_limit=0,Number_end_limit=0;
    BOOL fStatus_power;
    float *Wave_temp,Data_temp,Data_max,Limit_power ;
    end_info= (END *) malloc(sizeof(END));
    const Sample_time=700;
    Wave_temp=Wave;
    Data_temp=*Wave_temp;
    for(i=1;i<Length;i++)
    {
        Wave_temp++;
        if(Data_temp<=*Wave_temp)
        {
            Data_temp=*Wave_temp;
            Sample_max=i;
        }
    }

    //////////--Find start point--//////////

    Data_max=Data_temp;
    Limit_power=Data_max*Ratio;
    fStatus_power=FALSE;
    Wave_temp=Wave+Sample_max;
    Data_temp=*Wave_temp;

    for(i=1;i<Sample_max;i++)
    {
        Data_temp=*Wave_temp--;
        if(Data_temp<=Limit_power&&!fStatus_power)
        {
            Number_start_limit++;
            if(Number_start_limit<11)
            {
                Sample_start[Number_start_limit-1]=Sample_max-i;
                fStatus_power=TRUE;
            }
        }
        else
        {
            if(Data_temp>Limit_power&&fStatus_power)
                fStatus_power=FALSE;
        }
    }

    if(Number_start_limit==0)
        end_info->fStatus_start=TRUE;
    else
        end_info->fStatus_start=FALSE;

    //////////--Find end point--//////////

    fStatus_power=FALSE;
    Wave_temp=Wave+Sample_max;
    Data_temp=*Wave_temp;

    for(i=1;i<(Length-Sample_max);i++)
    {
        Data_temp=*Wave_temp++;
        if(Data_temp<=Limit_power&&!fStatus_power)
        {
            Number_end_limit++;
            if(Number_end_limit<11)
            {
                Sample_end[Number_end_limit-1]=Sample_max+i;
                fStatus_power=TRUE;
            }
        }
    }

```

```

    }
    else
    {
        if (Data_temp>Limit_power&&fStatus_power)
            fStatus_power=FALSE;
    }
}

if (Number_end_limit==0)
    end_info->fStatus_end=TRUE;
else
    end_info->fStatus_end=FALSE;

end_info->Length=Sample_end[0]-Sample_start[0];
end_info->Start=Sample_start[0];
end_info->End=Sample_end[0];
    ///---Check 700 Error---///
    if (Sample_max<Sample_time)
        end_info->fStatus_700_front=TRUE;
    if ((Length-Sample_max)<700)
        end_info->fStatus_700_rear=TRUE;
if (end_info->fStatus_start||end_info->fStatus_end
    ||end_info->Length<700)
{
    end_info->fStatus_complete=TRUE;
    end_info->Length=0;
    end_info->Start=0;
    end_info->End=0;
    return (end_info);
}

if ((Number_start_limit==1)&&(Number_end_limit==2))
{
    end_info->Length=Sample_end[1]-Sample_start[0];
    end_info->Start=Sample_start[0];
    end_info->End=Sample_end[1];
}

if ((Number_start_limit==2)&&(Number_end_limit==2))
{
    end_info->Length=Sample_end[1]-Sample_start[1];
    end_info->Start=Sample_start[1];
    end_info->End=Sample_end[1];
}

if ((Number_start_limit==2)&&(Number_end_limit==1))
{
    end_info->Length=Sample_end[0]-Sample_start[1];
    end_info->Start=Sample_start[1];
    end_info->End=Sample_end[0];
}

return(end_info);
}

BOOL CALLBACK EndOfSpeechDlgProc(HWND hDlg,UINT iMsg,WPARAM wParam,LPARAM lParam)
{
    static double Level[6]={0.0025,0.005,0.01,0.025,0.05,0.075};
    char szBuffer[75];
    static char szChar[30]="test";
    char szFilterSaveEng[] = "Data Files (*.END)\0*.end\0" \
        "All Files (*.*)\0*.*\0\0" ;
    char szFilterOpenDat[] = "Data Files (*.ENG)\0*.Eng\0" \
        "All Files (*.*)\0*.*\0\0" ;
    ///////////////////////////////////

    static float *bin1,*bin2;
    static float *Wave_temp1,*Wave_temp2;
    int framestep = 256;
    int windowsize = 512;
    int lporder = 12;
    static int nsample, frame, nframe,i=0,check=0,iCheckOpen=0,iCheckEnd=0;
    static short nLevel;

    HDC hdcEND;
    static END *End_speech;

```



```

switch (iMsg)
{
case WM_INITDIALOG:

    End_speech=(END *) malloc(sizeof(END));
    End_speech->Start=0;
    End_speech->Length=0;
    End_speech->End=0;
    nLevel=IDC_END_5;
    SendDlgItemMessage(hDlg,IDC_END_5,EM_SETCHECK,1,0L);
    wsprintf(szTitleName,"");

    return TRUE;

case WM_COMMAND:
    switch(LOWORD (wParam))
    {
    case IDC_END_SAVE:

        if(iCheckOpen==1&&iCheckEnd==1)
        {
            PopFileInitialize (hDlg , szFilterSaveEng);
            if(0!=PopFileSaveWave(hDlg,szFileName,szTitleName))
            {
                SaveDat(szFileName,DataOut,(End_speech->
Length)+256);

                check=3;
                }MessageBeep(0);
            }
        }
        else
        {
            if(iCheckOpen==0)check=0;
            if(iCheckEnd==0&&iCheckOpen==1)check=2;
        }
        InvalidateRect(hDlg,NULL,TRUE);
        return TRUE;

    case IDC_END_OPEN:

        PopFileInitialize (hDlg , szFilterOpenDat);
        if(0!=PopFileOpenDlg(hDlg,szFileName,szTitleName))
        {
            OpenDat(szFileName);
            iCheckOpen=1;
            check=2;
            Wave_temp1=out_temp;
        }

        InvalidateRect(hDlg,NULL,TRUE);
        return TRUE;

    case IDC_END_1:
    case IDC_END_2:
    case IDC_END_3:
    case IDC_END_4:
    case IDC_END_5:
    case IDC_END_6:
        nLevel=wParam;
        CheckRadioButton(hDlg,IDC_END_1,IDC_END_6,wParam);
        return TRUE;

    case IDC_END_END:

        if(iCheckOpen==1)
        {

            check=1;
            nsample=(int)lDataSize;

```

```

Level[ IDC_END_6-nLevel],1);
End_speech=EndOfSpeech(Wave_temp1,nsample,(float)
DataOut = (float *) calloc((End_speech->Length)+256,
sizeof(float*));
bin1=DataOut+128;

wsprintf(szChar,szFileName);
memcpy(szChar+23,"DAT",3);
OpenDat(szChar);
Wave_temp2=out_temp;
bin2=Wave_temp2+End_speech->Start;
for(i=0;i<(int)End_speech->Length;i++)
    *bin1++=*bin2++;

/*for(i=0;i<128;i++)
{
    Bin_data=(float)(log(1.0+((9.0/128.0)*i)));
    *bin1+=(float) (Bin_data);/*(*bin2+200));
}
*/

iCheckEnd=1;
}
else
    check=0;

InvalidateRect(hDlg,NULL,TRUE);
return TRUE;

case IDC_END_CANCEL:
    free(End_speech);
    free(DataOut);
    EndDialog(hDlg,0);
    return TRUE;
}
break;
case WM_PAINT:
    hdcEND = BeginPaint(hDlg,&ps);
    SetBkMode(hdcEND,TRANSPARENT);

    SelectObject(hdcEND,GetStockObject(DEFAULT_GUI_FONT));
    SetTextAlign(hdcEND,TA_CENTER);

    if(check==0)
        TextOut(hdcEND,213, 160,szBuffer,wsprintf(szBuffer,"Click Open to
Browse Dat File"));

    if(check==2)
    {
        TextOut(hdcEND,213, 160,szBuffer,wsprintf(szBuffer,"..Please Press
Run Button To Process.."));
        if(iCheckOpen==1)
            TextOut(hdcEND,150,
56,szBuffer,wsprintf(szBuffer,"%s",szTitleName));

        if(check==3)
        {
            TextOut(hdcEND,213, 160,szBuffer,wsprintf(szBuffer,"..Process
Completed.."));and..Click Open to Browse New Dat File.."));
            TextOut(hdcEND,150, 56,szBuffer,wsprintf(szBuffer,"%s",szTitleName));
            TextOut(hdcEND,150, 77,szBuffer,wsprintf(szBuffer,"%d",End_speech->
Start));
            TextOut(hdcEND,150, 98,szBuffer,wsprintf(szBuffer,"%d",End_speech->
End));
            TextOut(hdcEND,150, 119,szBuffer,wsprintf(szBuffer,"%d",End_speech->
Length));
            // TextOut(hdcEND,200, 56,szBuffer,wsprintf(szBuffer,"%s",szChar));

```



```

        EndPaint(hDlg,&ps);

        break;

    }
    return FALSE;
}

DTW_DATA *Dtw(unsigned int nfile,char* path[],int nChannel,char *file_name)
{
    DTW_DATA *Dtw_data;
    Dtw_data= (DTW_DATA *) malloc(sizeof(DTW_DATA));
    int nref,*nframe,i;
    float *window,*acf,*lpc,*ref,*cep,cMin;
    float ***templatel,Rec_result0,Rec_result1;
    dist_type *globdist;
    int check=0,iResult0,iResult1,iCheck_channel,iCheck_Error_1000=0;
    double
iError[10],dtw_temp,dtw_data;//iError1,iError2,iError3,iError4,iError5,iError6,iE
rror7,iError8,iError9,iError10,dtw_temp;

    nref = nfile - 1;
    window = (float*) calloc(WINDOWSIZE, sizeof(*window));
    acf = (float*) calloc(NCEPS + 1, sizeof(*acf));
    lpc = (float*) calloc(NCEPS + 1, sizeof(*lpc));
    ref = (float*) calloc(NCEPS + 1, sizeof(*ref));
    cep = (float*) calloc(NCEPS + 1, sizeof(*cep));
    templatel = (float***) calloc(nfile, sizeof(*templatel));
    nframe = (int *) calloc(nfile, sizeof(*nframe));
    globdist = (struct dist_type_struct *) calloc(nref, sizeof
(*globdist));

    for(i = 0; i < (int)nfile; i++)
    {
        float *sample;
        int nsample,j;
        if(i==(int)nfile-1)
            OpenDat(file_name);
        else
            OpenDat(path[i]);
        sample=out_temp;
        nsample=(int)lDataSize;

        if(lDataSize>1000)
        {
            nframe[i] = (nsample / FRAMESIZE)-1;

            templatel[i] = (float**) Smatrix(nframe[i], NCEPS,
sizeof(**templatel));

            for(j = 0; j < nframe[i]; j++)
            {
                int k;

                SHammwindow(sample + j * FRAMESIZE, WINDOWSIZE, window);
                Swav2acf(window, WINDOWSIZE, acf, NCEPS + 1);
                Sacf2lpc(acf, NCEPS, lpc, ref);
                Slpc2cep(lpc, NCEPS, cep, NCEPS);
                for(k = 0; k < NCEPS; k++)
                    templatel[i][j][k] = cep[k + 1];
            }
        }
    }

    // SaveDat("temp.dat",templatel[3][0],(NCEPS));//*nframe[0]);

```

```

// Match the i'th reference against the unknown using DTW
if(1DataSize>1000)
    iCheck_Error_1000=1;
else
    iCheck_Error_1000=0;

if(1DataSize>1000)
{
    for(i = 0; i < nref; i++)
    {
        globdist[i].distance = Sdtw(templatel[nref], nframe[nref],
        templatel[i],
            nframe[i], NCEPS);
        globdist[i].index = i;
    }

    if(!iCheck_dtw)
    {
        iError[0]=globdist[0].distance+globdist[1].distance
            +globdist[2].distance+globdist[3].distance
            +globdist[4].distance;
        iError[1]=globdist[5].distance+globdist[6].distance
            +globdist[7].distance+globdist[8].distance
            +globdist[9].distance;
        iError[2]=globdist[10].distance+globdist[11].distance+globdist[12].distance+globdist[13].distance
            +globdist[14].distance;
        iError[3]=globdist[15].distance+globdist[16].distance+globdist[17].distance+globdist[18].distance+globdist[19].distance;
        iError[4]=globdist[20].distance+globdist[21].distance+globdist[22].distance+globdist[23].distance+globdist[24].distance;
        iError[5]=globdist[25].distance+globdist[26].distance+globdist[27].distance+globdist[28].distance
            +globdist[29].distance;
        iError[6]=globdist[30].distance+globdist[31].distance+globdist[32].distance+globdist[33].distance+globdist[34].distance;
        iError[7]=globdist[31].distance+globdist[32].distance
            +globdist[33].distance+globdist[34].distance
            +globdist[35].distance;
        dtw_temp=iError[0];
        iCheck_channel=1;
        if(iError[0]>iError[1]||iError[0]>iError[2]||
            iError[0]>iError[3]||iError[0]>iError[4]
            ||iError[0]>iError[5]||iError[0]>iError[6]
            ||iError[0]>iError[7])
        {
            dtw_temp=iError[1];
            iCheck_channel=2;
            if(iError[1]>iError[2]||iError[1]>iError[3]
                ||iError[1]>iError[4]||iError[1]>iError[5]
                ||iError[1]>iError[6]||iError[1]>iError[7])
            {
                dtw_temp=iError[2];
                iCheck_channel=3;
                if(iError[2]>iError[3]||iError[2]>iError[4]
                    ||iError[2]>iError[5]||iError[2]>iError[6]
                    ||iError[2]>iError[7])
                {
                    dtw_temp=iError[3];
                    iCheck_channel=4;
                    if(iError[3]>iError[4]||iError[3]>iError[5]
                        ||iError[3]>iError[6]||iError[3]>iError[7])
                    {
                        dtw_temp=iError[4];
                        iCheck_channel=5;
                        if(iError[4]>iError[5]||iError[4]>iError[7]
                            ||iError[4]>
iError[6])
                        {
                            dtw_temp=iError[5];
                            iCheck_channel=6;

```



```

        if(iError[5]>iError[6]||iError[5]>iError[7])
        {
            dtw_temp=iError[6];
            iCheck_channel=7;

        if(iError[6]>iError[7])
        {
            dtw_temp=iError[7];
            iCheck_channel=8;
        }
        }}}}

    Dtw_data->Chanel_1=iCheck_channel;//iResult0+1;//(1+(10*
(nChannel-1)));
    Dtw_data->Chanel_2=iResult1+1;//(1+(10*(nChannel-1)));
    Dtw_data->distance_1=dtw_temp;//Rec_result0;
    Dtw_data->distance_2=(float)Rec_result1;
}
else
{
    iError[0]=globdist[0].distance+globdist[1].distance
    +globdist[2].distance+globdist[3].distance
    +globdist[4].distance;
    iError[1]=globdist[5].distance+globdist[6].distance
    +globdist[7].distance+globdist[8].distance
    +globdist[9].distance;

    dtw_temp=iError[0];
    iCheck_channel=9;

    if(iError[0]>iError[1])
    {
        dtw_temp=iError[1];
        iCheck_channel=10;
    }

    Dtw_data->Chanel_1=iCheck_channel;
    Dtw_data->distance_1=dtw_temp;
}

}

if(!iCheck_Error_1000==1)
{
    Dtw_data->Chanel_1=0;//iResult0+1;//(1+(10*(nChannel-1)));
    Dtw_data->distance_1=0.0;
}

    free(acf);
    free(lpc);
    free(templ1);
    free(globdist);
    free(nframe);
    free(cep);
    free(ref);

return (Dtw_data);
}

BOOL CALLBACK RecogDlgProc(HWND hDlg,UINT iMsg,WPARAM wParam,LPARAM lParam)
{

```

```

    char szFilterSaveEng[] = "Data Files (*.REC)\J*.rec\0" \
        "All Files (*.*)\0*.*\0\0" ;
    char szFilterOpenDat[] = "Data Files (*.END)\0*.end\0" \
        "All Files (*.*)\0*.*\0\0" ;
    static char *ipath[40]={"1.DAT","2.DAT","3.DAT","4.DAT","5.DAT"};
    static char szBuffer[40],buffer[50];
    static int i, nfile, nref, *nframe,a;
    static float *window, *acf, *lpc, *ref, *cep,*temp1,*temp2,cMin;
    static float ***templ1,Rec_result0,Rec_result1;
    static dist_type *globdist;
    static int check=0,iResult0,iResult1;
    static short nRef=IDC_REC_1;

```

```

static int nModel[3]={3,2,1};
HDC hdcRecog;
static DTW_DATA *dtw;
static unsigned int Length=0;
char *Status = {"Recognition"};

static double source = -3.1415e5,Rec_data,Rec_temp1,Rec_temp2;
//static float ;

switch (iMsg)
{
case WM_INITDIALOG:

hdcRecog=GetDC(hDlg);
wsprintf(szTitleName,"");
SendDlgItemMessage(hDlg,IDC_REC_1,BM_SETCHECK,1,0L);
Length=0;
return TRUE;

case WM_COMMAND:

switch(LOWORD (wParam))
{
case IDC_REC_1:
case IDC_REC_2:
case IDC_REC_3:
nRef=wParam;
CheckRadioButton(hDlg,IDC_REC_1,IDC_REC_3,wParam);
return TRUE;

/*case IDC_SAVE:
PopFileInitialize (hDlg , szFilterSaveEng);
if(0!=PopFileSaveWave(hDlg,szFileName,szTitleName))
{
SaveDat(szFileName,DataOut,1DataSize);
}
MessageBeep(0);
return TRUE;
*/

case IDC_RECOG_OPEN:

PopFileInitialize (hDlg , szFilterOpenDat);
if(0!=PopFileOpenDlg(hDlg,szFileName,szTitleName))
{
// wsprintf(path[(10*(nModel[IDC_REC_3-nRef]))],szFileName);//
OpenDat(szFileName);
OpenDat(szFileName);
Length=(unsigned int) 1DataSize;
check=1;

}

InvalidateRect(hDlg,NULL,TRUE);
return TRUE;

case IDC_RECOG_RECOGNITION:

if(check==1)
{

dtw =(DTW_DATA *) malloc(sizeof(DTW_DATA));
i=nModel[IDC_REC_3-nRef]*10+1;
dtw=Dtw(i+30,Mod_5,nModel[IDC_REC_3-
nRef],szFileName);

Rec_data=dtw->distance_1;
temp01=(float)dtw->Chanel_1;
/*
Rec_temp1=(float)dtw->Chanel_2;
Rec_temp2=dtw->distance_2;*/
free(dtw);

}

InvalidateRect(hDlg,NULL,TRUE);

```



```

        return TRUE;

    case IDC_RECOG_CANCEL:
        EndDialog(hDlg,0);
        return TRUE;
    }
    break;

case WM_PAINT:

    hdcRecog = BeginPaint(hDlg,&ps);
    SetBkMode(hdcRecog,TRANSPARENT);
    SelectObject(hdcRecog,GetStockObject(DEFAULT_GUI_FONT));
    SetTextAlign(hdcRecog,TA_CENTER);

    TextOut(hdcRecog,150,
61,szBuffer,wsprintf(szBuffer,"%s",szTitleName));
    TextOut(hdcRecog,150, 84,szBuffer,wsprintf(szBuffer,"%d",(int)
Length));
    // TextOut(hdcRecog,200, 142,szBuffer,wsprintf(szBuffer," %s
",Status));
    TextOut(hdcRecog,190,142,szBuffer,sprintf(szBuffer,"Distant value
is %0.3g Chanel %d ",Rec_data,(int)temp01));
//TextOut(hdcRecog,190,142,szBuffer,sprintf(szBuffer," %e %d %e %d",temp00,(int)
temp01,Rec_temp1,(int)Rec_temp2));

    /* if(check==0)
    {
        TextOut(hdcRecog,70,10,szBuffer,wsprintf(szBuffer,"File
1.DAT = %d",check));
    }
    */
    if(check==1)
    {
        TextOut(hdcRecog,70,10,buffer,sprintf(buffer,"file 1.dat = %e",temp00));
        TextOut(hdcRecog,70,40,buffer,sprintf(buffer,"file 2.dat = %e",temp01));
        TextOut(hdcRecog,70,70,buffer,sprintf(buffer,"file 3.dat = %e",temp02));
        TextOut(hdcRecog,70,100,buffer,sprintf(buffer,"file 4.dat = %e",temp03));
    }*/

    EndPaint(hDlg,&ps);

    break;

}
return FALSE;
}

float Sllikeddiag(float* vec, float* mean, float* var, int nitem) {
    int i;
    float nsuml = 0.0;

    for(i = 0; i < nitem; i++) {
        float dev = (vec[i] - mean[i]);
        nsuml += dev * dev / var[i] + (float)log((double)(2.0 * M_PI * var[i]));
    }
    return((float)(-0.5 * nsuml));
}

BOOL CALLBACK EndpointDlgProc(HWND hDlg,UINT iMsg,WPARAM wParam,LPARAM lParam)
{

    char szFilterOpenDat[] = "Data Files (*.DAT)\0*.dat\0" \
        "All Files (*.*)\0*.*\0\0" ;

    static int headersize = 1024;
    static int framestep = 256;
    static int windowsize = 2*256;
    static int ncepstra = 12;
    static int minsilence = 4;
    static float vqff = 1.0;
    static short *wave;
    static float *window;

```

```

static float **cepCache, *silMean, *sigMean, *silVar,
*sigVar;
static double *silSum, *sigSum, *silSumSq, *sigSumSq;
static int nsample, frame, nframe;
static int check=0, i, ninit, nquit, nacooustic,
lastninit, lastnquit;
static BOOL Check=FALSE;
static HDC hdcEndpoint;
static char szBuffer[40], buffer[50];

switch (iMsg)
{
case WM_INITDIALOG:

    hdcEndpoint=GetDC(hDlg);

    return TRUE;

case WM_COMMAND:

    switch(LOWORD (wParam))
    {

case IDOK:

PopFileInitialize (hDlg, szFilterOpenDat);
if(0!=PopFileOpenDlg(hDlg, szFileName, szTitleName))
{
    OpenDat(szFileName);
}

    Check=TRUE;
    return TRUE;

case IDC_ENDPOINT:

if(Check)

{
// #ifdef DELTAS
    nacooustic = 2 * ncepstra;
// #else
// nacooustic = ncepstra;
// #endif
    nsample=lDataSize;
    nframe = nsample / framestep;
    window = (float*)calloc(windowSize, sizeof(*window));
    cepCache = (float**) Smatrix(nframe, nacooustic, sizeof
(**cepCache));

// compute all the cepstra
for(frame = 0; frame < nframe; frame++) {
    SzeroMeanHammWindow(out_temp + frame * framestep,
windowSize, window);
// #ifdef CEPSTRA
// Swav2cep(window, windowSize, cepCache[frame], ncepstra);
// #else
// Swav2acf(window, windowSize, cepCache[frame], ncepstra);
for(i = 1; i < ncepstra; i++)
    cepCache[frame][i] /= cepCache[frame][0];
    cepCache[frame][0] = (float)pow(cepCache[frame][0],
0.333333);
// #endif
    }

// #ifdef DELTAS
// add the deltas

```



```

    for(i = 0; i < ncepstra; i++)
        cepCache[0][i + ncepstra] = 2 * (cepCache[1][i] - cepCache
[0][i]);
    for(frame = 1; frame < nframe - 1; frame++)
        for(i = 0; i < ncepstra; i++)
            cepCache[frame][i + ncepstra] =
                cepCache[frame + 1][i] - cepCache[frame - 1][i];
    for(i = 0; i < ncepstra; i++)
        cepCache[nframe - 1][i + ncepstra] =
            2 * (cepCache[nframe - 1][i] - cepCache[nframe - 2][i]);

//
#endif

//scale every component to have unity variance
for(i = 0; i < nacoustic; i++) {
    float sumx = 0.0;
    float sumxx = 0.0;
    float mean, var;

    for(frame = 0; frame < nframe; frame++) {
        float tmp = cepCache[frame][i];
        sumx += tmp;
        sumxx += tmp * tmp;
    }
    mean = sumx / nframe;
    var = sumxx / nframe - mean * mean;

    for(frame = 0; frame < nframe; frame++)
        cepCache[frame][i] /= var;
}

temp00=(float)nacoustic ;

// grab space for silence and signal sums and means
(*silSum));
(*sigSum));
(*silMean));
(*sigMean));
(*silSumSq));
(*sigSumSq));
(*silVar));
(*sigVar));

silSum = (double *)calloc((unsigned int )nacoustic, sizeof
sigSum = (double *)calloc((unsigned int )nacoustic, sizeof
silMean = (float *)calloc((unsigned int )nacoustic, sizeof
sigMean = (float *)calloc((unsigned int )nacoustic, sizeof
silSumSq = (double *)calloc((unsigned int )nacoustic, sizeof
sigSumSq = (double *)calloc((unsigned int )nacoustic, sizeof
silVar = (float *)calloc((unsigned int )nacoustic, sizeof
sigVar = (float *)calloc((unsigned int )nacoustic, sizeof

// assume that the first and the last frames are silence
ninit = minsilence;
nquit = minsilence;

// compute the sum of the silence vectors
for(i = 0; i < nacoustic; i++) {
    silSum[i] = 0.0;
    silSumSq[i] = 0.0;

    for(frame = 0; frame < ninit; frame++) {
        silSum[i] += cepCache[frame][i];
        silSumSq[i] += cepCache[frame][i] * cepCache[frame][i];
    }
    for(frame = nframe - nquit; frame < nframe; frame++) {
        silSum[i] += cepCache[frame][i];
        silSumSq[i] += cepCache[frame][i] * cepCache[frame][i];
    }
}

// compute the sum of the signal vectors
for(i = 0; i < nacoustic; i++) {
    sigSum[i] = 0.0;
    sigSumSq[i] = 0.0;

```

```

        for(frame = ninit; frame < nframe - nquit; frame++) {
            sigSum[i] += cepCache[frame][i];
            sigSumSq[i] += cepCache[frame][i] * cepCache[frame][i];
        }
    }

    lastninit = 0;
    lastnquit = 0;

    // iterate to convergence
    while(ninit != lastninit || nquit != lastnquit)
    {
        int maxsig;

        // store the last boundaries so we can see if they change
        lastninit = ninit;
        lastnquit = nquit;

        // compute means and variances of silence and signal
        for(i = 0; i < nacoustic; i++)
        {
            int nsil = ninit + nquit;
            int nsig = nframe - nsil;

            silMean[i] = (float)(silSum[i] / nsil);
            silVar[i] = (float)(silSumSq[i] / nsil - silMean[i] *
silMean[i]);

            sigMean[i] = (float)(sigSum[i] / nsig);
            sigVar[i] = (float)(sigSumSq[i] / nsig - sigMean[i] *
sigMean[i]);

            // CHEAT!
            silVar[i] = (float)((1.0 - vqff) * silVar[i] + vqff *
sigVar[i]);
        }

        // see if the start point can be increased
        if(Sllikeddiag(cepCache[ninit], silMean, silVar, nacoustic)
>
            Sllikeddiag(cepCache[ninit], sigMean, sigVar, nacoustic))
        {
            for(i = 0; i < 24; i++)
            {
                silSum[i] += (double) cepCache[ninit][i];
                sigSum[i] -= (double) cepCache[ninit][i];
                silSumSq[i] += (double) (cepCache[ninit][i] *
cepCache[ninit][i]);
                sigSumSq[i] -= (double) (cepCache[ninit][i] *
cepCache[ninit][i]);
            }
            ninit++;
        }

        // see if the end point can be decreased
        maxsig = nframe - nquit - 1;

        /*if(Sllikeddiag(cepCache[maxsig], silMean, silVar,
nacoustic) >
            Sllikeddiag(cepCache[maxsig], sigMean, sigVar,
nacoustic))
        {
            for(i = 0; i < nacoustic; i++)
            {
                silSum[i] += cepCache[maxsig][i];
                sigSum[i] -= cepCache[maxsig][i];
                silSumSq[i] += cepCache[maxsig][i] * cepCache
[maxsig][i];
                sigSumSq[i] -= cepCache[maxsig][i] * cepCache
[maxsig][i];
            }
            nquit++;
        }*/
    }

```



```

        MessageBeep(0);

    }

    /*//printf("%d\t%d\n", ninit * framestep, (nframe - nquit) *
framestep);
    */

}

check=1;

InvalidateRect(hDlg,NULL,TRUE);
return TRUE;

case IDCANCEL:

    EndDialog(hDlg,0);

    return TRUE;
}

break;

case WM_PAINT:

    hdcEndpoint = BeginPaint(hDlg,&ps);
    SetBkMode(hdcEndpoint,TRANSPARENT);
    if(check==0)
    {
        TextOut(hdcEndpoint,70,10,szBuffer,wsprintf(szBuffer,"File 1.DAT =
%d",check));
    }
    if(check==1)
    {
        TextOut(hdcEndpoint,70,10,buffer,sprintf(buffer,"Start Point =
%e",temp00));
        TextOut(hdcEndpoint,70,40,buffer,sprintf(buffer,"End Point =
%e",temp01));
        TextOut(hdcEndpoint,70,70,buffer,sprintf(buffer,"Number of Length =
%e",temp02));
        // TextOut(hdcEndpoint,70,100,buffer,sprintf(buffer,"file 4.dat =
%e",temp03));
    }

    EndPaint(hDlg,&ps);

    break;

}

return FALSE;
}

BOOL Energy(float *Data_in,float *Data_out,int lDataSize)
{
    FILTER *Hamming;
    float *hptr,*Data_temp,*Data_power,*h;
    int n,i;
    double arg;

    n=201;
    h=(float *)calloc(n,sizeof(float));
    hptr=h;
    arg=2*M_PI/(n-1);
    for(i=0;i<n;i++)
        *hptr++=(float)(0.54-0.46*cos(arg*i));
    Hamming = (FILTER *) malloc(sizeof(FILTER));
    Hamming->length=n;
    Hamming->coef=h;

```

```

Data_power=(float *)calloc(1DataSize,sizeof(float));
Data_temp=Data_power;
for(i=0;i<1DataSize;i++)
{
    *Data_temp++=(float)pow(*Data_in++,2.0);
}
fir_filter_array(Data_power,Data_out,1DataSize,Hamming);
free(h);
free(Hamming);

return TRUE;
}

BOOL CALLBACK EnergyDlgProc(HWND hDlg,UINT iMsg,WPARAM wParam,LPARAM lParam)
{
    char szBuffer[75];
    char szFilterSaveEng[] = "Data Files (*.ENG)\0*.eng\0" \
        "All Files (*.*)\0*.*\0\0" ;
    char szFilterOpenDat[] = "Data Files (*.DAT)\0*.dat\0" \
        "All Files (*.*)\0*.*\0\0" ;
    static HWND hwndPath;
    float *h,*hptr,*DataTemp;//,acc;
    double arg,i,x;
    static unsigned int n,check,iCheckOpen,iCheckEng;
    HDC hdcENG;
    FILTER *Hamming;//,*Average;

    switch (iMsg)
    {
        case WM_INITDIALOG:;

            return TRUE;

        case WM_COMMAND:
            switch(LOWORD (wParam))
            {
                case IDC_SAVE:

                    if(iCheckOpen==1&&iCheckEng==1)
                    {
                        PopFileInitialize (hDlg , szFilterSaveEng);
                        if(0!=PopFileSaveWave(hDlg,szFileName,szTitleName))
                        {
                            SaveDat(szFileName,DataOut,1DataSize);
                            check=3;
                        }
                        MessageBeep(0);
                    }
                    else
                    {
                        if(iCheckOpen==0)check=0;
                        if(iCheckEng==0&&iCheckOpen==1)check=2;
                    }

                    InvalidateRect(hDlg,NULL,TRUE);
                    return TRUE;

                case IDC_OK:

                    PopFileInitialize (hDlg , szFilterOpenDat);
                    if(0!=PopFileOpenDlg(hDlg,szFileName,szTitleName))
                    {
                        OpenDat(szFileName);
                        iCheckOpen=1;
                        check=2;
                    }

            }
    }
}

```



```

        return TRUE;

    case IDC_ENERGY:

        if(iCheckOpen==1)
        {
            DataOut=(float *)calloc(1DataSize,sizeof(float));
            Energy(out_temp,DataOut,(int)1DataSize);

            check=1; /*
            // HAMMING WINDOWS
            DataIn=out_temp;
            n=201;
            h=(float *)calloc(n,sizeof(float));
            hptr=h;

            arg=2*M_PI/(n-1);
            for(i=0;i<n;i++)
                *hptr++=(float)(0.54-0.46*cos(arg*i));

            Hamming = (FILTER *) malloc(sizeof(FILTER));

            Hamming->length=n;
            Hamming->coef=h;

            DataOut=(float *)calloc(1DataSize,sizeof(float));
            x=2.0;
            DataPower=(float *)calloc(1DataSize,sizeof(float));
            DataTemp=DataPower;

            for(i=0;i<1DataSize;i++)
            {
                *DataTemp++=(float)pow(*DataIn++,2.0);
            }

            fir_filter_array(DataPower,DataOut,1DataSize,Hamming);
            free(h);
            free(Hamming); /*

            iCheckEng=1;
        }
        else
            check=0;

        InvalidateRect(hDlg,NULL,TRUE);

        return TRUE;

    case IDCANCEL:
        EndDialog(hDlg,0);
        return TRUE;
    }
    break;
case WM_PAINT:

    hdcENG = BeginPaint(hDlg,&ps);
    SetBkMode(hdcENG,TRANSPARENT);
    SelectObject(hdcENG,GetStockObject(DEFAULT_GUI_FONT));
    SetTextAlign(hdcENG,TA_BASELINE);

    if(check==0)
        TextOut(hdcENG,120, 50,szBuffer,wsprintf(szBuffer,"      Click
Open to Browse Dat File      "));
    if(check==1)
        TextOut(hdcENG,120, 50,szBuffer,wsprintf(szBuffer," ..Click Save
Button To Save Eng File..  "));
    if(check==2)
    {
        TextOut(hdcENG,120, 50,szBuffer,wsprintf(szBuffer,"..Please Press
Energy Button To Process.."));
        if(iCheckOpen==1)

```

```

        TextOut(hdcENG,120,70,szBuffer,wsprintf(szBuffer,"
        ",szTitleName));
    }
    if(check==3)
    {
        TextOut(hdcENG,120,70,szBuffer,wsprintf(szBuffer,"
        ",szTitleName));
        TextOut(hdcENG,120, 50,szBuffer,wsprintf(szBuffer,"
        ..Process Completed..      ")); //and..Click Open to Browse New Dat File..");
    }

    EndPaint(hDlg,&ps);

    break;

}
return FALSE;
}

```

BOOL CALLBACK ZeroDlgProc(HWND hDlg,UINT iMsg,WPARAM wParam,LPARAM lParam)

```

{
    char szFilterSaveEng[] = "Data Files (*.ZER)\0*.zer\0" \
        "All Files (*.*)\0*.*\0\0" ;
    char szFilterOpenDat[] = "Data Files (*.DAT)\0*.dat\0" \
        "All Files (*.*)\0*.*\0\0" ;
    static HWND hwndPath;
    float *h,*hptr,*DataTemp,acc,*DataBin;
    double i;//arg,i;
    unsigned int n;

    FILTER *Average;//*Hamming;

    switch (iMsg)
    {
    case WM_INITDIALOG:
        return TRUE;

    case WM_COMMAND:
        switch(LOWORD (wParam))
        {
        case IDC_ZERO_SAVE:
            PopFileInitialize (hDlg , szFilterSaveEng);
            if(0!=PopFileSaveWave(hDlg,szFileName,szTitleName))
            {
                SaveDat(szFileName,DataOut,lDataSize);
            }
            MessageBeep(0);

            return TRUE;

        case ID_ZERO_OPEN:
            PopFileInitialize (hDlg , szFilterOpenDat);
            if(0!=PopFileOpenDlg(hDlg,szFileName,szTitleName))
            {
                OpenDat(szFileName);
            }

            return TRUE;

        case IDC_ZERO:
            if(out_temp!=0)
            {
                // AVERAGE WINDOWS
                DataIn=out_temp;
                DataBin=out_temp;
                n=150;
                h=(float *)calloc(n,sizeof(float));
                hptr=h;

                /*
                arg=2*M_PI/(n-1);
                for(i=0;i<n;i++)
                    *hptr++=0.54-0.46*cos(arg*i);
                */
            }
        }
    }
}

```



```

        Hamming = (FILTER *) malloc(sizeof(FILTER));

        Hamming->length=n;
        Hamming->coef=h;*/

        acc=1/(2*100);
        for(i=0;i<n;i++)
            *hptr++=(float)0.005;

        Average = (FILTER *) malloc(sizeof(FILTER));
        Average->length=n;
        Average->coef=h;

        DataOut=(float *)calloc(1DataSize,sizeof(float));

        DataZero=(float *)calloc(1DataSize,sizeof(float));
        DataTemp=DataZero;

        acc=0.0;
        acc=(float)abs((int)(Sgn(*DataIn)-1));
        DataIn++;
        *DataTemp=acc;

        for(i=0;i<1DataSize-1;i++)
        {
            acc=(float) abs((int)((Sgn(*DataIn++)-Sgn
(*DataBin++))));
            *DataTemp+=acc;
        }
        fir_filter_array(DataZero,DataOut,1DataSize,Average);
        free(h);
        free(Average);
        /*
        for(i=0;i<100;i++)
        {
            if(*DataOut!=0)
                MessageBeep(0);

            DataOut++;
        }*/
        //
        free(DataTemp);

        }
        else MessageBeep(0);
        return TRUE;

        case ID_ZERO_CANCEL:
            EndDialog(hDlg,0);
            return TRUE;
        }

        break;
        case WM_PAINT:
            break;

        }
        return FALSE;
    }
    float Sgn(float Datain)
    {
        if(Datain>=0)
            return 1;
        else
            return -1;
    }
    BOOL OpenWave(HWND hwnd,HANDLE waveDataBlock)
    {

```

```

char                achInFile[200];

HMMIO                hmmioIn = NULL;
MMCKINFO             ckInRIFF;
MMCKINFO             ckIn;
PCMWaveFormat        pcmWaveFormat; // contents of 'fmt' chunks

//-----Read IO Wave File-----//
wsprintf(achInFile, ofn.lpstrFile);
hmmioIn = mmioOpen(achInFile, NULL, MMIO_ALLOCBUF | MMIO_READ);
if (hmmioIn == NULL)

MessageBox(hwnd, "ERROR1", szAppName, MB_ICONINFORMATION|MB_OK);

//---Descend the input file into the 'RIFF' chunk---//
if (mmioDescend(hmmioIn, &ckInRIFF, NULL, 0) != 0)
    MessageBox(hwnd, "ERROR1", szAppName, MB_ICONINFORMATION|MB_OK);

//---Make sure the input file is a WAVE file.---//
if ((ckInRIFF.ckid != FOURCC_RIFF) ||
    (ckInRIFF.fccType != mmioFOURCC('W', 'A', 'V', 'E')))
    MessageBox(hwnd, "ERROR1", szAppName, MB_ICONINFORMATION|MB_OK);

//----- Search the input file for for the 'fmt' chunk.---//
ckIn.ckid = mmioFOURCC('f', 'm', 't', ' ');
if (mmioDescend(hmmioIn, &ckIn, &ckInRIFF, MMIO_FINDCHUNK) != 0)

MessageBox(hwnd, "ERROR1", szAppName, MB_ICONINFORMATION|MB_OK);

//-----'fmt' chunk too small---//
if (ckIn.cksize < (long) sizeof(pcmWaveFormat))

MessageBox(hwnd, "ERROR1", szAppName, MB_ICONINFORMATION|MB_OK);

/* Read the 'fmt' chunk into <pcmWaveFormat>*/
if (mmioRead(hmmioIn, (HPSTR) &pcmWaveFormat,
    (long) sizeof(pcmWaveFormat)) != (long) sizeof(pcmWaveFormat))

MessageBox(hwnd, "ERROR1", szAppName, MB_ICONINFORMATION|MB_OK);

/* Ascend the input file out of the 'fmt' chunk.*/
if (mmioAscend(hmmioIn, &ckIn, 0) != 0)

MessageBox(hwnd, "ERROR1", szAppName, MB_ICONINFORMATION|MB_OK);

/* Make sure the input file is an 8-bit mono PCM WAVE file.*/
if ((pcmWaveFormat.wf.wFormatTag != WAVE_FORMAT_PCM) ||
    (pcmWaveFormat.wf.nChannels != 1) ||
    (pcmWaveFormat.wBitsPerSample != 8))

MessageBox(hwnd, "ERROR1", szAppName, MB_ICONINFORMATION|MB_OK);

/* Search the input file for for the 'data' chunk.*/
ckIn.ckid = mmioFOURCC('d', 'a', 't', 'a');
if (mmioDescend(hmmioIn, &ckIn, &ckInRIFF, MMIO_FINDCHUNK) != 0)

MessageBox(hwnd, "ERROR1", szAppName, MB_ICONINFORMATION|MB_OK);

lDataSize = ckIn.cksize;

waveDataBlock = ::GlobalAlloc(GMEM_MOVEABLE, lDataSize);
if (waveDataBlock == NULL)
{
    MessageBox(hwnd, "Could not allocate
Memory", szAppName, MB_ICONINFORMATION|MB_OK);
}

pWave = (LPBYTE)::GlobalLock(waveDataBlock);
if (mmioRead(hmmioIn, (LPSTR)pWave, lDataSize) != lDataSize)
{
    MessageBox(hwnd, "Data Read
Failed", szAppName, MB_ICONINFORMATION|MB_OK);
    mmioClose(hmmioIn, 0);
    ::GlobalFree(waveDataBlock);
}

```



```

    }
    // wsprintf(szBuffer,"%d",lDataSize);
    // MessageBox(hwnd,"",szBuffer,MB_ICONINFORMATION|MB_OK);

    /*
        int xCount,yCount=0;
        LPBYTE pTemp;
        pTemp=pWave+800;

        int ySample[600];

        for(xCount=55;xCount<555;xCount++)
        {
            ySample[yCount]**pTemp;
            SetPixel(hdc,xCount,(ySample
[yCount]+200),RGB(0,0,255));

            yCount++;
            pTemp=pTemp++;
        }

    */

    return TRUE;
}

BOOL CALLBACK WaveDlgProc(HWND hDlg,UINT iMsg,WPARAM wParam,LPARAM lParam)
{
    HDC hdcWave;

    switch (iMsg)
    {
        case WM_INITDIALOG:

            return TRUE;

        case WM_COMMAND:
            switch(LOWORD (wParam))
            {
                case IDC_WAVEOPEN:

                    PopFileInitialize (hDlg , szFilter);
                    if(0!=PopFileOpenDlg(hDlg,szFileName,szTitleName))
                    {
                        fStateWaveData=0;
                        OpenWave(hDlg,waveDataBlock);
                        fStateWaveData=1;
                        InvalidateRect(hDlg,NULL,TRUE);
                    }

                    MessageBeep(0);
                    return TRUE;

                case IDOK:
                    PopFileInitialize (hDlg , szFilterSaveData);
                    PopFileSaveWave(hDlg,szFileName,szTitleName);
                    SaveWaveDat(szFileName);

            /*
                int i;
                long data_out[10000];
                DSP_FILE *out;
                LPBYTE TempWave;
                TempWave=pWave;

                out=OpenWrite("output.dat",SIGNED_LONG,1,lDataSize);
                for(i=0;i<lDataSize;i++)
                {
                    data_out[i] **TempWave;

```

```

        TempWave=TempWave++;
    }

    WriteRecord((char *)data_out,out);

    /*return TRUE;
        case IDCANCEL:
            EndDialog(hDlg,0);
            return TRUE;
        }
        break;

    case WM_PAINT:

        hdcWave = BeginPaint(hDlg,&ps);
        PaintWave(hDlg,hdcWave);
        EndPaint(hDlg,&ps);

        break;
    }
    return FALSE;
}

BOOL NEAR SettingsDlgInit( HWND hDlg)
{
    char        szBuffer[ MAXLEN_TEMPSTR ], szTemp[ MAXLEN_TEMPSTR ];
    WORD        wCount;

    LoadString( GETHINST( hDlg ), 1, szTemp, sizeof( szTemp ) );
    for (wCount = 0; wCount < 4; wCount++)
    {
        wsprintf( szBuffer, "%s%d", (LPSTR) szTemp, wCount + 1 );
        SendDlgItemMessage( hDlg, IDC_PORTM, CB_ADDSTRING, 0,
            (LPARAM) (LPSTR) szBuffer );
    }

    SendDlgItemMessage( hDlg, IDC_PORTM, CB_SETCURSEL,
        Modem.StatusCom, 0L );

    //Fill BaudRate to ComboBox
    for(wCount=2;wCount<15;wCount++)
    {
        LoadString( GETHINST( hDlg ), wCount, szTemp, sizeof( szTemp ) );
        SendDlgItemMessage( hDlg, IDC_BAUDM, CB_ADDSTRING, 0,
            (LPARAM) (LPSTR) szTemp );
    }
    SendDlgItemMessage( hDlg, IDC_BAUDM, CB_SETCURSEL,
        Modem.StatusBaud, 0L );

    //Fill Parity to ComboBox
    for(wCount=15;wCount<20;wCount++)
    {
        LoadString( GETHINST( hDlg ), wCount, szTemp, sizeof( szTemp ) );
        SendDlgItemMessage( hDlg, IDC_PARITYM, CB_ADDSTRING, 0,
            (LPARAM) (LPSTR) szTemp );
    }
    SendDlgItemMessage( hDlg, IDC_PARITYM, CB_SETCURSEL,
        Modem.StatusParity, 0L );

    for(wCount=20;wCount<23;wCount++)

```



```

{
LoadString( GETHINST( hDlg ), wCount, szTemp, sizeof( szTemp ) );
SendDlgItemMessage( hDlg, IDC_STOPBITM, CB_ADDSTRING, 0,
(LPSTR) szTemp );
}
SendDlgItemMessage( hDlg, IDC_STOPBITM, CB_SETCURSEL,
Modem.StatusBits, 0L );

for(wCount=23;wCount<27;wCount++)
{
LoadString( GETHINST( hDlg ), wCount, szTemp, sizeof( szTemp ) );
SendDlgItemMessage( hDlg, IDC_DATABITM, CB_ADDSTRING, 0,
(LPSTR) szTemp );
}
SendDlgItemMessage( hDlg, IDC_DATABITM, CB_SETCURSEL,
Modem.StatusData, 0L );

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

LoadString( GETHINST( hDlg ), 1, szTemp, sizeof( szTemp ) );
for (wCount = 0; wCount < 4; wCount++)
{
wsprintf( szBuffer, "%s%d", (LPSTR) szTemp, wCount + 1 );
SendDlgItemMessage( hDlg, IDC_PORTS, CB_ADDSTRING, 0,
(LPSTR) szBuffer );
}

SendDlgItemMessage( hDlg, IDC_PORTS, CB_SETCURSEL,
Serial.StatusCom, 0L );

//Fill BaudRate to ComboBox
for(wCount=2;wCount<15;wCount++)
{
LoadString( GETHINST( hDlg ), wCount, szTemp, sizeof( szTemp ) );
SendDlgItemMessage( hDlg, IDC_BAUDS, CB_ADDSTRING, 0,
(LPSTR) szTemp );
}
SendDlgItemMessage( hDlg, IDC_BAUDS, CB_SETCURSEL,
Serial.StatusBaud, 0L );

//Fill Parity to ComboBox
for(wCount=15;wCount<20;wCount++)
{
LoadString( GETHINST( hDlg ), wCount, szTemp, sizeof( szTemp ) );
SendDlgItemMessage( hDlg, IDC_PARITYS, CB_ADDSTRING, 0,
(LPSTR) szTemp );
}
SendDlgItemMessage( hDlg, IDC_PARITYS, CB_SETCURSEL,
Serial.StatusParity, 0L );

for(wCount=20;wCount<23;wCount++)
{
LoadString( GETHINST( hDlg ), wCount, szTemp, sizeof( szTemp ) );
SendDlgItemMessage( hDlg, IDC_STOPBITS, CB_ADDSTRING, 0,
(LPSTR) szTemp );
}
SendDlgItemMessage( hDlg, IDC_STOPBITS, CB_SETCURSEL,
Serial.StatusBits, 0L );

for(wCount=23;wCount<27;wCount++)
{
LoadString( GETHINST( hDlg ), wCount, szTemp, sizeof( szTemp ) );
SendDlgItemMessage( hDlg, IDC_DATABITS, CB_ADDSTRING, 0,
(LPSTR) szTemp );
}
SendDlgItemMessage( hDlg, IDC_DATABITS, CB_SETCURSEL,
Serial.StatusData, 0L );

return ( TRUE );
}

```

```

}

BOOL NEAR OpenConnect(HWND hwnd)
{
    // char lpBuf[]={'a','t','h','l','\n','\0'};

    OVERLAPPED osWrite = {0};

    BOOL fSuccess;
    DCB dcb;
    HANDLE hComm;
    hComm=CreateFile(ComNumber[Modem.StatusCom],GENERIC_READ|GENERIC_WRITE
        ,0,NULL,OPEN_EXISTING,0,NULL);

    if (hComm==INVALID_HANDLE_VALUE)
        MessageBox(hwnd,"MODEM",szAppName,MB_ICONINFORMATION|MB_OK);

    fSuccess=GetCommState(hComm,&dcb);
    if(!fSuccess)
    {
        return FALSE;
    }

    dcb.BaudRate=9600;
    dcb.ByteSize=8;
    dcb.Parity=NOPARITY;
    dcb.StopBits=ONESTOPBIT;

    fSuccess=SetCommState(hComm,&dcb);
    if(!fSuccess) return FALSE;

    TransmitCommChar(hComm,'a');
    TransmitCommChar(hComm,'t');
    TransmitCommChar(hComm,'h');
    TransmitCommChar(hComm,'l');
    TransmitCommChar(hComm,'\r');
    // TransmitCommChar(hComm,'\0');
    //MessageBox(hwnd,"COMPLETE",szAppName,MB_ICONINFORMATION|MB_OK);
    //}

    return TRUE;
}

BOOL WriteABuffer(char * Com,char * lpBuf,DWORD dwToWrite,HWND hwnd)
{
    OVERLAPPED osWrite = {0};
    DWORD dwWritten;
    BOOL fRes,fSuccess;
    HANDLE hComm;
    DCB dcb;

    hComm=CreateFile(Com,GENERIC_READ|GENERIC_WRITE
        ,0,NULL,OPEN_EXISTING,FILE_ATTRIBUTE_NORMAL|FILE_FLAG_OVERLAPPED,NULL);

    if (hComm==INVALID_HANDLE_VALUE) //return FALSE;
        MessageBox(hwnd,"MODEM",szAppName,MB_ICONINFORMATION|MB_OK);

    fSuccess=GetCommState(hComm,&dcb);
    dcb.BaudRate=9600;
    dcb.ByteSize=8;
    dcb.fParity=NOPARITY;
    dcb.StopBits=ONESTOPBIT;
    fSuccess=SetCommState(hComm,&dcb);

    osWrite.hEvent=CreateEvent(NULL,TRUE,FALSE,NULL);
    if(osWrite.hEvent==NULL)
    {
        MessageBox(hwnd,"Error Create Overlapped
Even",szAppName,MB_ICONINFORMATION|MB_OK);
        return FALSE;
    }
}

```



```

    if(!WriteFile(hComm,lpBuf,dwToWrite,&dwWritten,&osWrite))
    {
        if(GetLastError()!=ERROR_IO_PENDING)
        {
            fRes=FALSE;
        }
        else
        {
            if(!GetOverlappedResult(hComm,&osWrite,&dwWritten,TRUE))
            {
                fRes=FALSE;
            }
            else fRes=TRUE;
        }
    }
    CloseHandle(osWrite.hEvent);
    CloseHandle(hComm);
    return fRes;
}

void DetectRing (PVOID pvoid)
{
    PPARAMS pparams;
    DWORD dwCount,dwCommEvent,dwWrite;
    HANDLE hCommPort;
    COMMTIMEOUTS ctmoCommPort;
    DCB dcbCommPort;
    OVERLAPPED ov={0},osWrite={0};
    BOOL fR,fI,fN,fG;

    char c;
    int i=0;
    pparams= (PPARAMS) pvoid;
    // clock_t wait;

    hCommPort = CreateFile(ComNumber[Modem.StatusCom], GENERIC_READ |
    GENERIC_WRITE, 0,
        NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL |
        FILE_FLAG_OVERLAPPED, 0);

    ctmoCommPort.ReadIntervalTimeout = MAXDWORD;
    ctmoCommPort.ReadTotalTimeoutMultiplier = MAXDWORD;
    ctmoCommPort.ReadTotalTimeoutConstant = MAXDWORD;
    ctmoCommPort.WriteTotalTimeoutMultiplier = 0;
    ctmoCommPort.WriteTotalTimeoutConstant = 0;
    SetCommTimeouts(hCommPort, &ctmoCommPort);
    dcbCommPort.DCBlength = sizeof(DCB);
    GetCommState(hCommPort, &dcbCommPort);
    dcbCommPort.BaudRate=9600;
    dcbCommPort.ByteSize=8;
    dcbCommPort.StopBits=ONESTOPBIT;
    dcbCommPort.Parity=NOPARITY;
    SetCommState(hCommPort, &dcbCommPort);
    SetCommMask(hCommPort, EV_RXCHAR);
    ov.Offset = 0;
    ov.OffsetHigh = 0;
    ov.hEvent = CreateEvent(NULL,FALSE, FALSE, NULL);//TRUE, FALSE, NULL);
    osWrite.hEvent= CreateEvent(NULL,FALSE, FALSE, NULL);

    if(!pparams->bContinue&&!iCheckThread)
    {
        while (TRUE)
        {
            //pparams->bContinue=TRUE;
            iCheckThread=1;

            if(WaitCommEvent(hCommPort,&dwCommEvent,NULL))
            {
                while (ReadFile(hCommPort, &c, 1, &dwCount, &ov))
                {
                    if (dwCount == 1)
                    {
                        if(c=='R')fR=1;
                        if(c=='I')fI=1;
                        if(c=='N')fN=1;
                    }
                }
            }
        }
    }
}

```

```

        if(c=='G')fG=1;
    }
    if(fr&fI&fN&fG)
    {
        clock_t goal;
        fr=0;fI=0;fN=0;fG=0;
        int iCount;
        PlaySound("C:\\WAV\\ring.WAV",NULL,SND_FILENAME
        |SND_ASYNC);

        for(iCount=0;iCount<48;iCount++)
        {
            goal=(((clock_t)1 * CLOCKS_PER_SEC )/16+clock());
            fStateRing^=1;
            SendMessage(pparams->hwnd,WM_RING,0,0);
            while( goal > clock() );
        }

        WriteFile(hCommPort,cOffHook,5,&dwWrite
        ,&osWrite);
        CloseHandle(ov.hEvent);
        CloseHandle(hCommPort);
        iCheckThread=0;
        SendMessage(pparams->hwnd,WM_END_RING,0,0);
        goto LOOP;
    }
    else
    {
        MessageBox(pparams->hwnd,"
WaitComEvent12",szAppName,MB_ICONINFORMATION|MB_OK);
        break;
    }
}

LOOP:
    MessageBeep(0);
    CloseHandle(hCommPort);
    CloseHandle(ov.hEvent);
}

DSP_FILE *OpenRead(char *file_name)
    /* file name string */
{
    DSP_FILE *dsp_info;
    int status;

    /* allocate the DSP data file structure */
    dsp_info = (DSP_FILE *) malloc(sizeof(DSP_FILE));

    /* open file for binary read and update */
    dsp_info->fp = fopen(file_name,"r+b");

    /* copy and allocate file name string for the DSP_FILE structure */
    dsp_info->name = (char *)malloc(strlen(file_name) + 1);
    strcpy(dsp_info->name,file_name);

    /* read in header from file */
    status = fread((char *)dsp_info,sizeof(HEADER),1,dsp_info->fp);

    /* return pointer to DSP_FILE structure */
    return(dsp_info);
}

DSP_FILE *OpenWrite(char *file_name,int type,unsigned short int records,
    unsigned short int rec_len)
{

```



```

    DSP_FILE *dsp_info;
    int status;

/* allocate the DSP data file structure */
    dsp_info = (DSP_FILE *) malloc(sizeof(DSP_FILE));
    if(!dsp_info) {
        // MessageBox(hwnd, "ERROR1", szAppName, MB_ICONINFORMATION|MB_OK);
        exit(1);
    }

/* set the basics */
    dsp_info->type = (unsigned char)type;
    dsp_info->records = records;
    dsp_info->rec_len = rec_len;

/* set element size from data type */
    switch(type) {
        case 0:
        case 4:
            dsp_info->element_size = sizeof(char);
            break;
        case 1:
        case 5:
            dsp_info->element_size = sizeof(short int);
            break;
        case 2:
        case 6:
            dsp_info->element_size = sizeof(long int);
            break;
        case 3:
            dsp_info->element_size = sizeof(float);
            break;
        case 7:
            dsp_info->element_size = sizeof(double);
            break;
        default:
            // MessageBox(hwnd, "ERROR1", szAppName, MB_ICONINFORMATION|MB_OK);
            exit(1);
    }

/* open file for binary write */
    dsp_info->fp = fopen(file_name, "wb");
    if(!dsp_info->fp) {
        // MessageBox(hwnd, "ERROR1", szAppName, MB_ICONINFORMATION|MB_OK);
        return(NULL);
    }

/* copy and allocate file name string for the DSP_FILE structure */
    dsp_info->name = (char *)malloc(strlen(file_name) + 1);
    if(!dsp_info->name) {
        // MessageBox(hwnd, "ERROR1", szAppName, MB_ICONINFORMATION|MB_OK);
        exit(1);
    }
    strcpy(dsp_info->name, file_name);

/* write header to file */
    status = fwrite((char *)dsp_info, sizeof(HEADER), 1, dsp_info->fp);
    if(status != 1) {
        // MessageBox(hwnd, "ERROR1", szAppName, MB_ICONINFORMATION|MB_OK);
        exit(1);
    }

/* return pointer to DSP_FILE structure */
    return(dsp_info);
}

void ReadRecord( char *ptr, DSP_FILE *dsp_info)
{
    int status;
    status = fread(ptr, dsp_info->element_size,
                    dsp_info->rec_len, dsp_info->fp);
    if(status != dsp_info->rec_len) {
        exit(1);
    }
}

```

```

    }
}

void WriteRecord(char *ptr,DSP_FILE *dsp_info)
{
    int status;
    if(!dsp_info) {
        // printf("\nError in DSP_FILE structure passed to write_record\n");
        exit(1);
    }
    status = fwrite(ptr,dsp_info->element_size,
                    dsp_info->rec_len,dsp_info->fp);
    if(status != dsp_info->rec_len) {
        // printf("\nError write_record, file %s\n",dsp_info->name);
        exit(1);
    }
}

BOOL SaveDat(PSTR pstrFileName,float *data,long lDataSize)
{
    DSP_FILE *out;

    out=OpenWrite(pstrFileName,FLOAT,1,(unsigned short)lDataSize);
    WriteRecord((char *)data,out);
    fclose(out->fp);
    // free(out_temp);
    return TRUE;
}

BOOL OpenDat(PSTR pstrFileName)
{
    DSP_FILE *in;
    in=OpenRead(pstrFileName);
    lDataSize=in->rec_len;
    out_temp =(float *)calloc(in->rec_len,sizeof(float));

    ReadRecord((char *)out_temp,in);
    fclose(in->fp);
    return TRUE;
}

BOOL SaveWaveDat(PSTR pstrFileName)
{
    DSP_FILE *out;
    long l;
    float temp;
    float *out_prt;
    LPBYTE pDat;
    pDat=pWave;
    out_temp=(float *)calloc(lDataSize,sizeof(float));
    out_prt=out_temp;

    for(l=0;l<lDataSize;l++)
    {
        *out_prt++ = ((float)(*pDat++)-128);
    }

    out_prt=out_temp;
    temp=0;

    for(l=0;l<lDataSize;l++)
    {
        if(temp<*out_prt)
            temp=*out_prt;
        else
            out_prt++;
    }
    out_prt=out_temp;
    for(l=0;l<lDataSize;l++)
    {
        *out_prt=(*out_prt/temp);
        out_prt++;
    }
}

```



```

        out=OpenWrite(pstrFileName,FLOAT,1,(unsigned short)lDataSize);
        WriteRecord((char *)out_temp,out);
        fclose(out->fp);
        free(out_temp);
        return TRUE;
    }

void PopFileInitialize (HWND hwnd ,char szFilter[])
{
    ofn.lStructSize      = sizeof (OPENFILENAME) ;
    ofn.hwndOwner        = hwnd ;
    ofn.hInstance        = NULL ;
    ofn.lpstrFilter       = szFilter ;
    ofn.lpstrCustomFilter = NULL ;
    ofn.nMaxCustFilter    = 0 ;
    ofn.nFilterIndex      = 0 ;
    ofn.lpstrFile         = NULL ;           // Set in Open and Close functions
    ofn.nMaxFile          = _MAX_PATH ;
    ofn.lpstrFileTitle    = NULL ;           // Set in Open and Close functions
    ofn.nMaxFileTitle     = _MAX_FNAME + _MAX_EXT ;
    ofn.lpstrInitialDir   = NULL ;
    ofn.lpstrTitle        = NULL ;
    ofn.Flags              = 0 ;             // Set in Open and Close functions
    ofn.nFileOffset       = 0 ;
    ofn.nFileExtension    = 0 ;
    ofn.lpstrDefExt       = "txt" ;
    ofn.lCustData          = 0L ;
    ofn.lpfnHook           = NULL ;
    ofn.lpTemplateName    = NULL ;
}

BOOL PopFileOpenDlg (HWND hwnd, PSTR pstrFileName, PSTR pstrTitleName)
{
    ofn.hwndOwner        = hwnd ;
    ofn.lpstrFile         = pstrFileName ;
    ofn.lpstrFileTitle    = pstrTitleName ;
    ofn.Flags              = OFN_HIDEREADONLY | OFN_CREATEPROMPT ;

    return GetOpenFileName (&ofn) ;
}

BOOL PopFileSaveWave (HWND hwnd,PSTR pstrFileName,PSTR pstrTitleName)
{
    ofn.hwndOwner        = hwnd ;
    ofn.lpstrFile         = pstrFileName ;
    ofn.lpstrFileTitle    = pstrTitleName ;
    ofn.Flags              = OFN_OVERWRITEPROMPT;
    return GetSaveFileName (&ofn);
}

DWORD RecordWaveFile( HWND hwnd,DWORD dwMilliseconds,char* lpstrName )
{
    UINT wDeviceID;
    DWORD dwReturn,Check_message;
    MCI_OPEN_PARMS mciOpenParms;
    MCI_RECORD_PARMS mciRecordParms;
    MCI_SAVE_PARMS mciSaveParms;
    MCI_PLAY_PARMS mciPlayParms;
    char szFilterSaveEng[]="Data Files (*.WAV)\0*.wav\0" \
        "All Files (*.*)\0*.*\0\0" ;
    PopFileInitialize (hwnd , szFilterSaveEng);
    ofn.lpstrInitialDir="C:\\Recognition\\";

    mciOpenParms.lpstrDeviceType="waveaudio";

```

```

mciOpenParms.lpstrElementName="";
if(dwReturn=mciSendCommand(0,MCI_OPEN,MCI_OPEN_ELEMENT|MCI_OPEN_TYPE,
    (DWORD)(LPVOID) &mciOpenParms))
{
    MessageBeep(0);
    return dwReturn;
}
wDeviceID=mciOpenParms.wDeviceID;
mciRecordParms.dwTo=dwMilliseconds;

if(dwReturn = mciSendCommand(wDeviceID,MCI_RECORD,MCI_TO|MCI_WAIT,
    (DWORD)(LPVOID)&mciRecordParms))
{
    mciSendCommand(wDeviceID,MCI_CLOSE,0,NULL);
    return dwReturn;
}

mciPlayParms.dwFrom=0;

if(WriteABuffer("COM3",cOnHook,5,hwnd))
{
    if(mciSendCommand(wDeviceID,MCI_PLAY,MCI_FROM|MCI_WAIT
        ,(DWORD)(LPVOID) &mciPlayParms))
    {
        mciSendCommand(wDeviceID,MCI_CLOSE,0,NULL);
        return dwReturn;
    }
}

Wave"
    Check_message=MessageBox(hwnd,"Do you want Save this File","Record
        ,MB_ICONINFORMATION|MB_YESNOCANCEL);

    switch(Check_message)
    {
case IDCANCEL:
    {
        mciSendCommand(wDeviceID,MCI_CLOSE,0,NULL);
        iCheck_record=1;
        return TRUE;
    }
        break;
case IDYES:
        break;
case IDNO:
    {
        mciSendCommand(wDeviceID,MCI_CLOSE,0,NULL);
        return TRUE;
    }
        break;
    }

    wsprintf(szFileName,lpstrName);
    if(0==PopFileSaveWave(hwnd,szFileName,szTitleName))
    {
        mciSendCommand(wDeviceID,MCI_CLOSE,0,NULL);
        return TRUE;
    }
    InvalidateRect(hwnd,NULL,FALSE);
    SendMessage(hwnd,WM_PAINT,0,0);
    if(MessageBox(hwnd,"Do you want Save this File ","",
        MB_YESNO)==IDNO)
    {
        mciSendCommand(wDeviceID,MCI_CLOSE,0,NULL);
        return dwReturn;
    }

mciSaveParms.lpfilename=szFileName;
if(dwReturn=mciSendCommand(wDeviceID,MCI_SAVE,MCI_SAVE_FILE|
    MCI_WAIT,(DWORD)(LPVOID) &mciSaveParms))
{
    mciSendCommand(wDeviceID,MCI_CLOSE,0,NULL);
}

```



```

        return dwReturn;

    }
    mciSendCommand(wDeviceID,MCI_CLOSE,0,NULL);
    return 0;
}

DWORD RecordWave( HWND hwnd,DWORD dwMilliseconds,char* lpstrName )
{
    UINT wDeviceID;
    DWORD dwReturn;
    MCI_OPEN_PARMS mciOpenParms;
    MCI_RECORD_PARMS mciRecordParms;
    MCI_SAVE_PARMS mciSaveParms;
    MCI_PLAY_PARMS mciPlayParms;
    mciOpenParms.lpstrDeviceType="waveaudio";
    mciOpenParms.lpstrElementName="";
    if(dwReturn=mciSendCommand(0,MCI_OPEN,MCI_OPEN_ELEMENT|MCI_OPEN_TYPE,
        (DWORD)(LPVOID) &mciOpenParms))
    {
        MessageBeep(0);
        return dwReturn;
    }
    wDeviceID=mciOpenParms.wDeviceID;
    mciRecordParms.dwTo=dwMilliseconds;
    if(dwReturn = mciSendCommand(wDeviceID,MCI_RECORD,MCI_TO|MCI_WAIT,
        (DWORD)(LPVOID) &mciRecordParms))
    {
        mciSendCommand(wDeviceID,MCI_CLOSE,0,NULL);
        return dwReturn;
    }
    mciPlayParms.dwFrom=0;
/*    if(mciSendCommand(wDeviceID,MCI_PLAY,MCI_FROM|MCI_WAIT
        ,(DWORD)(LPVOID) &mciPlayParms))
    {
        mciSendCommand(wDeviceID,MCI_CLOSE,0,NULL);
        return dwReturn;
    }
*/
    if(MessageBox(hwnd,"Do you want Save this File","Record Wave"
        ,MB_ICONINFORMATION|MB_YESNO)==IDNO)
    {
        mciSendCommand(wDeviceID,MCI_CLOSE,0,NULL);
        return TRUE;
    }
    wsprintf(szFileName,lpstrName);
    if(0==PopFileSaveWave(hwnd,szFileName,szTitleName))
    {
        mciSendCommand(wDeviceID,MCI_CLOSE,0,NULL);
        return TRUE;
    }
    // if(MessageBox(hwnd,"Do you want Save this File ","",
    //     MB_YESNO)==IDNO)
    // {
    //     mciSendCommand(wDeviceID,MCI_CLOSE,0,NULL);
    //     return dwReturn;
    // }
    /*
    mciSaveParms.lpfilename=lpstrName;
    if(dwReturn=mciSendCommand(wDeviceID,MCI_SAVE,MCI_SAVE_FILE|
        MCI_WAIT,(DWORD)(LPVOID) &mciSaveParms))
    {
        mciSendCommand(wDeviceID,MCI_CLOSE,0,NULL);
        return dwReturn;
    }
    mciSendCommand(wDeviceID,MCI_CLOSE,0,NULL);
    return 0;
}

```

```

BOOL PaintWave(HWND hDlg,HDC hdcWave)
{
    hdcWave=GetDC(hDlg);
    SetBkMode(hdcWave,TRANSPARENT);
    SetTextAlign(hdcWave,TA_CENTER);
    double dx,xd[PX],yd[PY],pi = 3.1415926535;
    int i,data_pts;
    double x, ySample[1000];
    for(x=0,i=0;(x<=1000)&&(i<1000);i++,x++)
    {
        ySample[i]=50*cos(2*pi*0.01*x);
    }

    int
xSample[]={128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128};
    char szBuffer[40];
    int yCount=100,xCount=55,
        xxCount=105,yyCount=50;
    int DataTemp;

    dx=(500)/PX;

    //-----Store Sample in Arrays-----//
    for (x=0,i=0;(x<=500)&&(i<PX);x+=dx,i++)
    {
        xd[i]=x;
        yd[i]=50*cos(2*pi*0.01*x);
    }
    data_pts=i;

    Rectangle(hdcWave,55,50,555,350);

    //-----Draw the axes-----//

    MoveToEx(hdcWave,55,200,NULL);
    LineTo(hdcWave,555,200);

    //-----Draw X axis-----//
    for(i=0;i<PX;i+=DECX/10)
    {
        if(i%DECX==0)
        {
            MoveToEx(hdcWave,i+GL,GB-5,NULL);
            LineTo(hdcWave,i+GL,GB+4);
        }
        else
        {
            MoveToEx(hdcWave,i+GL,GB-3,NULL);
            LineTo(hdcWave,i+GL,GB+2);
        }
    }

    //-----Draw Y axis-----//
    for(i=0;i<PY;i+=DECY/10)
    {
        if(i%DECY==0)
        {
            MoveToEx(hdcWave,GL-4,GT+i,NULL);
            LineTo(hdcWave,GL+5,GT+i);
        }
        else
        {
            MoveToEx(hdcWave,GL-2,GT+i,NULL);
            LineTo(hdcWave,GL+3,GT+i);
        }
    }

    //-----Number On Y axis-----//
    TextOut(hdcWave,60,10,szBuffer,wsprintf(szBuffer,"Sample Value"));
    TextOut(hdcWave,35,43,szBuffer,wsprintf(szBuffer,"127"));
}

```



```

TextOut(hdcWave,35,190,szBuffer,wsprintf(szBuffer,"0"));
TextOut(hdcWave,35,340,szBuffer,wsprintf(szBuffer,"-128"));

//----Number On X axis-----//
TextOut(hdcWave,53,355,szBuffer,wsprintf(szBuffer,"0"));
TextOut(hdcWave,300,380,szBuffer,wsprintf(szBuffer,"Sample Number"));

if(fStateWaveData)
{
    DataTemp=lDataSize/500;

    TextOut(hdcWave,305,355,szBuffer,wsprintf(szBuffer,"%d",lDataSize/2));

    TextOut(hdcWave,550,355,szBuffer,wsprintf(szBuffer,"%d",lDataSize));

    int ySample[200];
    LPBYTE TempWave;
    TempWave=pWave;

    for(xCount=55;xCount<555;xCount++)
    {
        for(yCount=0;yCount<DataTemp;yCount++)
        {
            ySample[yCount]=*TempWave;
            SetPixel(hdcWave,xCount,(ySample
[yCount]+72),RGB(0,0,255));
            //
            yCount++;
            TempWave=TempWave++;
        }
    }

}

/*
int i,yn,xn;
for(i=0;i<data_pts;i++)
{
    yn=(int)((yd[i]-
*/
do{
do{
    SetPixel(hdc,xCount,yCount,RGB(0,0,0));
    xCount=xCount+10;
}while(xCount<555);
xCount=55;
yCount=yCount+50;
}while(yCount<350);

do{
do{
    SetPixel(hdc,xxCount,yyCount,RGB(0,0,0));
    yyCount=yyCount+10;
}while(yyCount<350);
yyCount=50;
xxCount=xxCount+50;
}while(xxCount<555);
yCount=100;

do{
for(xCount=551;xCount<555;xCount++)
{
    SetPixel(hdc,xCount,yCount,RGB(0,0,0));
}

for(xCount=55;xCount<59;xCount++)
{
    SetPixel(hdc,xCount,yCount,RGB(0,0,0));
}
yCount=yCount+50;
}
while(yCount<350);
xCount=55;

do{
for(yCount=346;yCount<350;yCount++)
{
    SetPixel(hdc,xCount,yCount,RGB(0,0,0));
}
}
}

```

```

    }
    for(yCount=50;yCount<54;yCount++)
    {
        SetPixel(hdc,xCount,yCount,RGB(0,0,0));
    }
    xCount=xCount+50;
}
while(xCount<555);
*/
//TextOut(hdc,250,370,szBuffer, wsprintf
(szBuffer,ofn.lpstrFile));

/*
yCount=0;
for(xCount=55;xCount<555;xCount++)
{
    SetPixel(hdc,xCount,ySample[yCount]+200,RGB(0,0,255));
    yCount++;
}
*/

ReleaseDC(hDlg,hdcWave);
return 0;
}

BOOL PaintWndProc()
{
    {
        SelectObject(hMemDC,hBitmapLine);

        BitBlt(hdc,0,0,406,1,hMemDC,0,0,SRCCOPY);
        SelectObject(hMemDC,hBitmapMain);
        BitBlt(hdc,0,1,400,200,hMemDC,0,0,SRCCOPY);
    }

    //          Display Modem Button          ///
    if(fStateModem)
    {
        SelectObject(hMemDC,hBitmapModem);
        BitBlt(hdc,16,99,47,20,hMemDC,0,0,SRCCOPY);
    }
    //          Display Ring Button          ///
    if(fStateRing)
    {
        SelectObject(hMemDC,hBitmapRing);
        BitBlt(hdc,68,99,60,20,hMemDC,0,0,SRCCOPY);
    }
    //          Display Off-Hook Button          ///
    if(fStateOffHook)
    {
        SelectObject(hMemDC,hBitmapOffHook);
        BitBlt(hdc,68,99,60,20,hMemDC,0,0,SRCCOPY);
    }
    //          Display On-Hook Button          ///
    if(fStateOnHook)
    {
        SelectObject(hMemDC,hBitmapOnHook);
        BitBlt(hdc,68,98,57,20,hMemDC,0,0,SRCCOPY);
    }
    //          Display Eight Button          ///
    if(fState8)
    {
        SelectObject(hMemDC,hBitmap8);
        BitBlt(hdc,315,43,17,20,hMemDC,0,0,SRCCOPY);
    }
    //          Display Seven Button          ///

```



```

        if(fState7)
        {
            SelectObject(hMemDC,hBitmap7);
            BitBlt(hdc,279,43,17,20,hMemDC,0,0,SRCCOPY);
        }
//          Display Six   Button          ///
if(fState6)
{
    SelectObject(hMemDC,hBitmap6);
    BitBlt(hdc,243,43,17,20,hMemDC,0,0,SRCCOPY);
}

//          Display Five   Button          ///

if(fState5)
{
    SelectObject(hMemDC,hBitmap5);
    BitBlt(hdc,205,43,17,20,hMemDC,0,0,SRCCOPY);
}
//          Display Four   Button          ///

if(fState4)
{
    SelectObject(hMemDC,hBitmap4);
    BitBlt(hdc,172,43,17,20,hMemDC,0,0,SRCCOPY);
}

//          Display Three   Button          ///
if(fState3)
{
    SelectObject(hMemDC,hBitmap3);
    BitBlt(hdc,134,43,17,20,hMemDC,0,0,SRCCOPY);
}

//          Display Two     Button          ///
if(fState2)
{
    SelectObject(hMemDC,hBitmap2);
    BitBlt(hdc,99,43,17,20,hMemDC,0,0,SRCCOPY);
}
//          Display One     Button          ///

if(fState1)
{
    SelectObject(hMemDC,hBitmap1);
    BitBlt(hdc,63,43,16,20,hMemDC,0,0,SRCCOPY);
}

//          Display Manual Button          ///

if(fStateManual)
{
    SelectObject(hMemDC,hBitmapManual);
    BitBlt(hdc,69,165,80,20,hMemDC,0,0,SRCCOPY);
}

//          Display On button          //

if(fStateOn)
{
    SelectObject(hMemDC,hBitmapOn);
    BitBlt(hdc,15,165,45,20,hMemDC,0,0,SRCCOPY);
}
return TRUE;
}

```

```

BOOL ReportStatus()
{
    if(fState1|fState2|fState3|fState4|fState5|fState6)//|fState7|fState8)
    {
        PlaySound(Wave_connect[1],NULL,SND_FILENAME);
        if(fState1)
            PlaySound(Wave_channel[0],NULL,SND_FILENAME);
        if(fState2)
            PlaySound(Wave_channel[1],NULL,SND_FILENAME);
        if(fState3)
            PlaySound(Wave_channel[2],NULL,SND_FILENAME);
        if(fState4)
            PlaySound(Wave_channel[3],NULL,SND_FILENAME);
        if(fState5)
            PlaySound(Wave_channel[4],NULL,SND_FILENAME);
        if(fState6)
            PlaySound(Wave_channel[5],NULL,SND_FILENAME);
        if(fState7)
            PlaySound(Wave_channel[6],NULL,SND_FILENAME);
        if(fState8)
            PlaySound(Wave_channel[7],NULL,SND_FILENAME);
    }

    return TRUE;
}

BOOL Capture(HWND hwnd,int x,int y)
{
    OVERLAPPED osWrite={0};
    osWrite.hEvent= CreateEvent(NULL,FALSE, FALSE, NULL);
    char * file={"3.wav"};

//          Capture Eight Button          //
    if((x>=316)&&(x<=330)&&(y>=40)&&(y<=60))
    {
        if((fStateOn==1)|| (fStateManual==0))
        fState8=fState8;
        else fState8 ^=1;
        if(fState8==1)
        /*
        {
            fStatusTraining=1;
            SendMessage(hwnd,WM_END_RING,0,0);
        }*/
        {
            if(WriteABuffer(ComNumber[Serial.StatusCom],On[7],1,hwnd)==0)MessageBox
(hwnd,"ERROR",szAppName,MB_ICONINFORMATION|MB_OK);
        }
        else
        {
            if(WriteABuffer(ComNumber[Serial.StatusCom],Off[7],1,hwnd)==0)MessageBox
(hwnd,"ERROR",szAppName,MB_ICONINFORMATION|MB_OK);
        }
        InvalidateRect(hwnd,NULL,FALSE);
        MessageBeep(0);
    }

//          Capture Seven Button          //
    if((x>=280)&&(x<=296)&&(y>=40)&&(y<=60))
    {
        if((fStateOn==1)|| (fStateManual==0))
        fState7=fState7;
        else fState7 ^=1;
        if(fState7==1)
        /*
        {
            fStatusConnect=1;
            SendMessage(hwnd,WM_END_RING,0,0);
        }*/
    }
}

```



```

{
    if (WriteABuffer(ComNumber[Serial.StatusCom], On[6], 1, hwnd) == 0) MessageBox
(hwnd, "ERROR", szAppName, MB_ICONINFORMATION|MB_OK);
    }
    else
    {
        if (WriteABuffer(ComNumber[Serial.StatusCom], Off[6], 1, hwnd) == 0) MessageBox
(hwnd, "ERROR", szAppName, MB_ICONINFORMATION|MB_OK);
        }
        InvalidateRect(hwnd, NULL, FALSE);
        MessageBeep(0);
    }

//          Capture Six Button          //

if ((x>=245)&&(x<=260)&&(y>=40)&&(y<=60))
{
    if ((fStateOn==1) || (fStateManual==0))
fState6=fState6;
    else fState6 ^=1;
    if (fState6==1)
    {
        if (WriteABuffer(ComNumber[Serial.StatusCom], On[5], 1, hwnd) == 0) MessageBox
(hwnd, "ERROR", szAppName, MB_ICONINFORMATION|MB_OK);
        }
        else
        {
            if (WriteABuffer(ComNumber[Serial.StatusCom], Off[5], 1, hwnd) == 0) MessageBox
(hwnd, "ERROR", szAppName, MB_ICONINFORMATION|MB_OK);
            }
            InvalidateRect(hwnd, NULL, FALSE);
            MessageBeep(0);
        }

//          Capture Five Button          //

if ((x>=208)&&(x<=222)&&(y>=40)&&(y<=60))
{
    if ((fStateOn==1) || (fStateManual==0))
fState5=fState5;
    else fState5 ^=1;
    if (fState5==1)
    {
        if (WriteABuffer(ComNumber[Serial.StatusCom], On[4], 2, hwnd) == 0) MessageBox
(hwnd, "ERROR", szAppName, MB_ICONINFORMATION|MB_OK);
        }
        else
        {
            if (WriteABuffer(ComNumber[Serial.StatusCom], Off[4], 2, hwnd) == 0) MessageBox
(hwnd, "ERROR", szAppName, MB_ICONINFORMATION|MB_OK);
            }
            InvalidateRect(hwnd, NULL, FALSE);
            MessageBeep(0);
        }

//          Capture Four Button          //

if ((x>=171)&&(x<=188)&&(y>=40)&&(y<=60))
{
    if ((fStateOn==1) || (fStateManual==0))
fState4=fState4;
    else fState4 ^=1;
    if (fState4==1)
    {
        if (WriteABuffer(ComNumber[Serial.StatusCom], On[3], 1, hwnd) == 0) MessageBox
(hwnd, "ERROR", szAppName, MB_ICONINFORMATION|MB_OK);
        }
        else
        {

```

```

        if (WriteABuffer(ComNumber[Serial.StatusCom], Off[3], 1, hwnd) == 0) MessageBox
(hwnd, "ERROR", szAppName, MB_ICONINFORMATION|MB_OK);
    }
    InvalidateRect(hwnd, NULL, FALSE);

    MessageBeep(0);
}

//          Capture Three Button          //
if ((x >= 135) && (x <= 150) && (y >= 40) && (y <= 60))
{
    fState3 = fState3;
    if ((fStateOn == 1) || (fStateManual == 0))
    {
        else fState3 ^= 1;
        if (fState3 == 1)
        {
            if (WriteABuffer(ComNumber[Serial.StatusCom], On[2], 1, hwnd) == 0) MessageBox
(hwnd, "ERROR", szAppName, MB_ICONINFORMATION|MB_OK);
        }
        else
        {
            if (WriteABuffer(ComNumber[Serial.StatusCom], Off[2], 1, hwnd) == 0) MessageBox
(hwnd, "ERROR", szAppName, MB_ICONINFORMATION|MB_OK);
        }
        InvalidateRect(hwnd, NULL, FALSE);
    }
    //          if (RecordWaveFile(5000, file) == 0) { MessageBox
(hwnd, "MODEM", szAppName, MB_ICONINFORMATION|MB_OK); }
    //          MessageBeep(0);
}

//          Capture Two Button          //
if ((x >= 100) && (x <= 115) && (y >= 40) && (y <= 60))
{
    fState2 = fState2;
    if ((fStateOn == 1) || (fStateManual == 0))
    {
        else fState2 ^= 1;
        if (fState2 == 1)
        {
            if (WriteABuffer(ComNumber[Serial.StatusCom], On[1], 1, hwnd) == 0) MessageBox
(hwnd, "ERROR", szAppName, MB_ICONINFORMATION|MB_OK);
        }
        else
        {
            if (WriteABuffer(ComNumber[Serial.StatusCom], Off[1], 1, hwnd) == 0) MessageBox
(hwnd, "ERROR", szAppName, MB_ICONINFORMATION|MB_OK);
        }
        InvalidateRect(hwnd, NULL, FALSE);
    }
    /*
    PlaySound("C:\\WAV\\HELLOWIN.WAV", NULL, SND_FILENAME|SND_ASYNC);
    //
    PlaySound("C:\\WAV\\PUI_1.WAV", NULL, SND_FILENAME|SND_ASYNC);
    InvalidateRect(hwnd, NULL, FALSE);
    //          PlaySound("SoundName", hInst,
SND_RESOURCE | SND_ASYNC);

    /*          // MessageBeep(0);
    }
    //          Capture One Button          //
    if ((x >= 65) && (x <= 80) && (y >= 42) && (y <= 60))
    {

```



```

fState1=fState1;
        if((fStateOn==1)|| (fStateManual==0))
        else fState1 ^=1;
        if(fState1==1)
        {
            if(WriteABuffer(ComNumber[Serial.StatusCom],On[0],1,hwnd)==0)MessageBox
(hwnd,"ERROR",szAppName,MB_ICONINFORMATION|MB_OK);
        }
        else
        {
            if(WriteABuffer(ComNumber[Serial.StatusCom],Off[0],1,hwnd)==0)MessageBox
(hwnd,"ERROR",szAppName,MB_ICONINFORMATION|MB_OK);
        }
        /* if(fState1==1)
        {
            if(WriteABuffer(ComNumber[Modem.StatusCom],cOffHook,5,hwnd)==0)MessageBox
(hwnd,"ERROR",szAppName,MB_ICONINFORMATION|MB_OK);
        }
        else
        {
            if(WriteABuffer(ComNumber[Modem.StatusCom],cOnHook,5,hwnd)==0)MessageBox
(hwnd,"ERROR",szAppName,MB_ICONINFORMATION|MB_OK);
        }
        */
        InvalidateRect(hwnd,NULL,FALSE);
        MessageBeep(0);
    }
    // Capture Manual Button///
    if((x>=69)&&(x<=183)&&(y>=164)&&(y<=183))
    {
        if(fStateOn==1) fStateManual=0;
        else fStateManual ^=1;
        InvalidateRect(hwnd,NULL,FALSE);
        MessageBeep(0);
    }
    // Capture On Button //
    if((x>=15)&&(x<=59)&&(y>=164)&&(y<=183))
    {
        if(fStateManual==1) fStateManual=0;
        InvalidateRect(hwnd,NULL,FALSE);
        fStateOn ^=1;
        if(fStateOn==1)
        {
            EnableMenuItem(hMenu,
ID_FILE_CONNECT,MF_GRAYED) ;
            EnableMenuItem(hMenu,
ID_FILE_DISCONNECT,MF_ENABLED) ;
        }
        else
        {
            EnableMenuItem(hMenu, ID_FILE_DISCONNECT,
MF_GRAYED) ;
            EnableMenuItem(hMenu, ID_FILE_CONNECT, MF_ENABLED) ;
        }
        MessageBeep(0);
    }
    return 0;
}
BOOL Initial()
{
    //-----Initial Modem-----//
    Modem.StatusBaud=6;
    Modem.StatusBits=0;

```

```

Modem.StatusCom=2;
Modem.StatusData=3;
Modem.StatusParity=0;

//-----Initial Sreial-----//
Serial.StatusBaud=6;
Serial.StatusBits=0;
Serial.StatusCom=0;
Serial.StatusData=3;
Serial.StatusParity=0;

//-----Initial Bitmap-----//
hBitmapManual=LoadBitmap(ghInst,MAKEINTRESOURCE
(IDB_MANUAL));
hBitmapMain = LoadBitmap(ghInst,MAKEINTRESOURCE(IDB_MAIN));
hBitmapLine = LoadBitmap(ghInst,MAKEINTRESOURCE(IDB_LINE));
hBitmapOn = LoadBitmap(ghInst,MAKEINTRESOURCE(IDB_ON));
hBitmap1 = LoadBitmap(ghInst,MAKEINTRESOURCE(IDB_1));
hBitmap2 = LoadBitmap(ghInst,MAKEINTRESOURCE(IDB_2));
hBitmap3 = LoadBitmap(ghInst,MAKEINTRESOURCE(IDB_3));
hBitmap4 = LoadBitmap(ghInst,MAKEINTRESOURCE(IDB_4));
hBitmap5 = LoadBitmap(ghInst,MAKEINTRESOURCE(IDB_5));
hBitmap6 = LoadBitmap(ghInst,MAKEINTRESOURCE(IDB_6));

hBitmap7 = LoadBitmap(ghInst,MAKEINTRESOURCE(IDB_7));
hBitmap8 = LoadBitmap(ghInst,MAKEINTRESOURCE(IDB_8));
hBitmapOnHook = LoadBitmap(ghInst,MAKEINTRESOURCE
(IDB_ONHOOK));
hBitmapOffHook = LoadBitmap(ghInst,MAKEINTRESOURCE
(IDB_OFFHOOK));
hBitmapRing = LoadBitmap(ghInst,MAKEINTRESOURCE(IDB_RING));
hBitmapModem = LoadBitmap(ghInst,MAKEINTRESOURCE
(IDB_MODEM));

return TRUE;
}

BOOL CurrentPre(HWND hDlg)
{
    ///-----Current Modem Port -----///
    Modem.StatusCom = SendDlgItemMessage( hDlg, IDC_PORTM,
        CB_GETCURSEL,
        0, 0L );
    Modem.StatusBaud = SendDlgItemMessage( hDlg, IDC_BAUDM,
        CB_GETCURSEL,
        0, 0L );
    Modem.StatusBits = SendDlgItemMessage( hDlg, IDC_STOPBITM,
        CB_GETCURSEL,
        0, 0L );
    Modem.StatusData = SendDlgItemMessage( hDlg, IDC_DATABITM,
        CB_GETCURSEL,
        0, 0L );
    Modem.StatusParity = SendDlgItemMessage( hDlg, IDC_PARITYM,
        CB_GETCURSEL,
        0, 0L );

    ///-----Current Status Serial Port ----///
    Serial.StatusCom = SendDlgItemMessage( hDlg, IDC_PORTS,
        CB_GETCURSEL,
        0, 0L );
    Serial.StatusBaud = SendDlgItemMessage( hDlg, IDC_BAUDS,
        CB_GETCURSEL,
        0, 0L );
    Serial.StatusBits = SendDlgItemMessage( hDlg, IDC_STOPBITS,
        CB_GETCURSEL,
        0, 0L );
    Serial.StatusData = SendDlgItemMessage( hDlg, IDC_DATABITS,
        CB_GETCURSEL,
        0, 0L );
    Serial.StatusParity = SendDlgItemMessage( hDlg, IDC_PARITYS,
        CB_GETCURSEL,
        0, 0L );

return TRUE;
}

```



```

}

BOOL InitialSerial(HANDLE hCom ,DCB dcb)
{
//    DCB dcb;
//    HANDLE hCom;
    FillMemory(&dcb,sizeof(dcb),0);
    if(!GetCommState(hCom,&dcb))
        return FALSE;

    dcb.BaudRate=BaudTable[Serial.StatusBaud];
    dcb.Parity=ParityTable[Serial.StatusParity];
    dcb.StopBits=StopBitsTable[Serial.StatusBits];
    dcb.ByteSize=DataBitsTable[Serial.StatusData];

    if(!SetCommState(hCom,&dcb))
        return FALSE;
    return TRUE;
}

BOOL InitialModem(HANDLE hCom ,DCB dcb)
{
//    DCB dcb;
//    HANDLE hCom;
    FillMemory(&dcb,sizeof(dcb),0);
    if(!GetCommState(hCom,&dcb))
        return FALSE;

    dcb.BaudRate=BaudTable[Modem.StatusBaud];
    dcb.Parity=ParityTable[Modem.StatusParity];
    dcb.StopBits=StopBitsTable[Modem.StatusBits];
    dcb.ByteSize=DataBitsTable[Modem.StatusBits];

    if(!SetCommState(hCom,&dcb))
        return FALSE;
    return TRUE;
}

void Swav2acf(float *wav, int nwav, float *acf, int nacf) {
    int i, j;

    for(i = 0; i < nacf; i++) {
        double sum = 0.0;
        for(j = 0; j < nwav - i; j++)
            sum += wav[j] * wav[j + i];
        acf[i] = (float) sum;
    }
}

float Sacf2lpc(float *acf, int nacf, float *lpc, float *ref) {
    static float *tmp;
    static int ntmp = 0;
    double e, ci;
    int i, j;

    if(ntmp != nacf) {
        if(ntmp != 0) free(tmp);
        tmp = (float *)calloc(nacf, sizeof(*tmp));
        ntmp = nacf;
    }

    /* find lpc coefficients */
    e = acf[0];
    lpc[0] = 1.0;
    for(i = 1; i <= nacf; i++) {
        ci = 0.0;
        for(j = 1; j < i; j++) ci += lpc[j] * acf[i-j];
        ci = ((acf[i] - (float)ci) / e);
        ref[i] = (float)ci;
        lpc[i] = (float)ci;
        for(j = 1; j < i; j++) tmp[j] = lpc[j] - (float)(ci * lpc[i-j]);
        for(j = 1; j < i; j++) lpc[j] = tmp[j];
        e = (1 - ci * ci) * e;
    }
    return((float)e);
}

```

```

}

float *SmkHammwindow(int nitem) {
    float *window, k;
    int i;

    k = (float)(2.0 * M_PI / (nitem - 1));

    window = (float *)calloc(nitem, sizeof(*window));

    for(i = 0; i < nitem; i++)
        window[i] = (float)(0.54 - 0.46 * cos(k * i));
    return(window);
}

float Sl2norm(float *vec0, float* vec1, int nitem) {
    float sum = 0.0;
    int i;

    for(i = 0; i < nitem; i++) {
        float diff = vec0[i] - vec1[i];
        sum += diff * diff;
    }

    return(sum);
}

float Sdtw(float **unkTemp, int nunkTemp, float **refTemp, int nrefTemp,
           int tempSize) {
    int i, j;
    float **localDist, **globalDist;
    float bestDist;

    localDist = (float**) Smatrix(nunkTemp, nrefTemp, sizeof(**localDist));
    globalDist = (float**) Smatrix(nunkTemp, nrefTemp, sizeof(**globalDist));

    /* compute and store all the local distances */
    for(i = 0; i < nunkTemp; i++)
        for(j = 0; j < nrefTemp; j++)
            localDist[i][j] = Sl2norm(unkTemp[i], refTemp[j], tempSize);

    /* for the first frame the only possible match is at (0, 0) */
    globalDist[0][0] = localDist[0][0];
    for(j = 1; j < nrefTemp; j++)
        globalDist[0][j] = (float)REALLY_VERY_BIG;

    /* in the second frame the only valid state is (1, 1) */
    globalDist[1][0] = (float)REALLY_VERY_BIG;
    globalDist[1][1] = globalDist[0][0] + localDist[1][1];
    for(j = 2; j < nrefTemp; j++)
        globalDist[1][j] = (float)REALLY_VERY_BIG;

    /* and do for the general case of the rest of the frames */
    for(i = 2; i < nunkTemp; i++) {
        /* */
        globalDist[i][0] = (float)REALLY_VERY_BIG;
        globalDist[i][1] = globalDist[i-1][0] + localDist[i][1];
        float topPath, midPath, botPath;
        for(j = 2; j < nrefTemp; j++) {

            if(globalDist[i-2][j-1] < 0.0)
                MessageBeep(0);
            /* Spanic("globalDist[i-2][j-1] < 0.0\n: %d\t%d\t%f\n", i, j, globalDist[i-2][j-1]); */

            topPath = globalDist[i-2][j-1] + localDist[i-1][j] + localDist[i][j];
            midPath = globalDist[i-1][j-1] + localDist[i][j];
            botPath = globalDist[i-1][j-2] + localDist[i][j-1] + localDist[i][j];

            /* find and store the smallest gloabal distance */
            if(topPath < midPath && topPath < botPath)
                globalDist[i][j] = topPath;
            else if (midPath < botPath)

```



```

        globalDist[i][j] = midPath;
    else
        globalDist[i][j] = botPath;
    }

}

bestDist = globalDist[nunkTemp - 1][nrefTemp - 1];

free(localDist);
free(globalDist);

return(bestDist);
}

void Slpc2cep(float *lpc, int nlpc, float *cep, int ncep) {
    int n;

    for(n = 1; n <= nlpc && n <= ncep; n++) {
        float sum = 0.0;
        int k;

        for(k = 1; k < n; k++) sum += k * cep[k] * lpc[n - k];
        cep[n] = lpc[n] + sum / n;
    }

    /* be wary of these interpolated values */
    for(n = nlpc + 1; n <= ncep; n++) {
        float sum = 0.0;
        int k;

        for(k = n - (nlpc - 1); k < n; k++) sum += k * cep[k] * lpc[n - k];
        cep[n] = sum / n;
    }
}

void SHammwindow(float *wave, int nitem, float *window) {
    static int nwindowf = 0;
    static float *windowf = NULL;
    int i;

    if(nwindowf != nitem) {
        if(nwindowf != 0) free(windowf);
        windowf = SmkHammwindow(nitem);
        nwindowf = nitem;
    }

    for(i = 0; i < nitem; i++)
        window[i] = windowf[i] * wave[i];
}

void SzeroMeanHammWindow(float *wave, int nitem, float *window) {
    static int nwindowf = 0;
    static float *windowf = NULL;
    int i, sum;
    float mean;

    if(nwindowf != nitem) {
        if(nwindowf != 0) free(windowf);
        windowf = SmkHammwindow(nitem);
        nwindowf = nitem;
    }

    sum = 0;
    for(i = 0; i < nitem; i++)
        sum += (int)wave[i];
    mean = (float) sum / nitem;

    for(i = 0; i < nitem; i++)
        window[i] = windowf[i] * (wave[i] - mean);
}

void **Smatrix(size_t nitem0, size_t nitem1, size_t size) {
    size_t nbyte;
    void **array0;

```

```

char *array1;
int i;

nbyte = nitem0 * sizeof(char*) + nitem0 * nitem1 * size;
if(nbyte == 0) nbyte = 1;
array0 = (void**)malloc(nbyte);
/*if(array0 == NULL)
    Sperror("Smatrix(%d, %d, %d)", nitem0, nitem1, size);
*/

array1 = (char*) (array0 + nitem0);
for(i = 0; i < (int)nitem0; i++)
    array0[i] = array1 + i * nitem1 * size;
return(array0);
}

void fir_filter_array(float *in,float *out,int in_len,FILTER *fir)
{
    int i,j,coef_len2,acc_length;
    float acc;
    float *in_ptr,*data_ptr,*coef_start,*coef_ptr,*in_end;

    /* set up for coefficients */
    coef_start = fir->coef;
    coef_len2 = (fir->length + 1)/2;

    /* set up input data pointers */
    in_end = in + in_len - 1;
    in_ptr = in + coef_len2 - 1;

    /* initial value of accumulation length for startup */
    acc_length = coef_len2;

    for(i = 0 ; i < in_len ; i++) {
        /* set up pointers for accumulation */
        data_ptr = in_ptr;
        coef_ptr = coef_start;

        /* do accumulation and write result */
        acc = (*coef_ptr++) * (*data_ptr--);
        for(j = 1 ; j < acc_length ; j++)
            acc += (*coef_ptr++) * (*data_ptr--);
        *out++ = acc;

        /* check for end case */
        if(in_ptr == in_end) {
            acc_length--; /* one shorter each time */
            coef_start++; /* next coefficient each time */
        }

        /* if not at end, then check for startup, add to input pointer */
        else {
            if(acc_length < (int)fir->length) acc_length++;
            in_ptr++;
        }
    }
}

```

Resource.h

```

//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by Speech.rc
//

```


#define IDS_COMPREFFIX	1
#define IDS_BAUD110	2
#define IDS_BAUD300	3
#define IDS_BAUD600	4
#define IDS_BAUD1200	5
#define IDS_BAUD2400	6
#define IDS_BAUD4800	7
#define IDS_BAUD9600	8
#define IDS_BAUD14000	9
#define IDS_BAUD19200	10
#define IDS_BAUD38400	11
#define IDS_BAUD56000	12
#define IDS_BAUD128000	13
#define IDS_BAUD256000	14
#define IDS_PARITYNONE	15
#define IDS_PARITYEVEN	16
#define IDS_PRITYODD	17
#define IDS_PARITYMARK	18
#define IDS_PARITYSPACE	19
#define IDS_ONESTOPBIT	20
#define IDS_SONESTOPBIT	21
#define IDS_TWOSTOPBIT	22
#define IDS_DATABIT5	23
#define IDS_DATABIT6	24
#define IDS_DATABIT7	25
#define IDS_DATABIT8	26
#define IDD_DIALOG_MODEM	104
#define IDD_MODEM	104
#define IDB_BITMAP1	112
#define IDD_PREFER	133
#define IDI_ICON1	134
#define IDI_ICON	134
#define IDB_LINE	137
#define IDB_MANUAL	140
#define IDB_ON	142
#define IDB_MAIN	144
#define IDB_1	145
#define IDB_2	146
#define IDB_3	147
#define IDB_4	148
#define IDB_5	149
#define IDB_6	150
#define IDB_8	152
#define IDB_MODEM	153
#define IDB_OFFHOOK	154
#define IDB_ONHOOK	155
#define IDB_RING	156
#define IDB_7	157
#define IDD_LPC	158
#define IDD_LP	158
#define IDI_ICON2	159
#define IDI_ICON3	160
#define IDD_WAVE	161
#define IDD_ENERGY	162
#define IDD_ZERO	163
#define IDD_LPC12	164
#define IDD_RECOG	165
#define IDD_ENDPOINT	166
#define IDD_CEP	167
#define IDD_END	168
#define IDC_RADIOCOM1	1000
#define IDC_RADIO_COM2	1001
#define IDC_RADIO_COM3	1002
#define IDC_RADIO_COM4	1003
#define IDC_RADIO_BAUD1	1004
#define IDC_RADIO_BAUD2	1005
#define IDC_RADIO_BAUD3	1006
#define IDC_RADIO_BAUD4	1007
#define IDC_RADIO_PARITY1	1008
#define IDC_RADIO_PARITY2	1009
#define IDC_RADIO_PARITY3	1010
#define IDC_RADIO_PARITY4	1011
#define IDC_RADIO_BIT1	1012
#define IDC_RADIO_BIT2	1013
#define IDC_RADIO_BIT3	1014
#define IDC_RADIO_BIT4	1015

#define IDC_RADIO_STOP1	1016
#define IDC_RADIO_STOP2	1017
#define IDC_RADIO_STOP3	1018
#define IDC_PORT1	1019
#define IDC_PORTM	1019
#define IDC_PORT2	1020
#define IDC_PORT3	1021
#define IDC_PORT4	1022
#define IDC_MODEM_PORT	1023
#define IDC_COMBO2	1024
#define IDC_BAUDM	1024
#define IDC_COMBO3	1025
#define IDC_PARITYM	1025
#define IDC_COMBO4	1026
#define IDC_DATABITM	1026
#define IDC_COMBO5	1027
#define IDC_STOPBITM	1027
#define IDC_COMBO6	1028
#define IDC_PORTS	1028
#define IDC_COMBO7	1029
#define IDC_BAUDS	1029
#define IDC_COMBO8	1030
#define IDC_PARITYS	1030
#define IDC_COMBO9	1031
#define IDC_DATABITS	1031
#define IDC_COMBO10	1032
#define IDC_STOPBITS	1032
#define IDC_LPC	1034
#define IDC_CHANGEDIR	1037
#define IDC_EINLPC	1038
#define IDC_EDIT4	1039
#define IDC_WAVEOPEN	1041
#define IDC_SAVE	1042
#define IDC_ENERGY	1043
#define IDC_ZERO	1044
#define IDC_ZERO_SAVE	1045
#define ID_ZERO_OPEN	1046
#define ID_ZERO_CANCEL	1047
#define IDC_LPC_LPC	1048
#define IDC_LPC_SAVE	1049
#define IDC_LPC_OPEN	1050
#define IDC_LPC_CANCEL	1051
#define IDC_RECOG_RECOGNITION	1052
#define IDC_RECOG_OPEN	1054
#define IDC_RECOG_CANCEL	1055
#define IDC_ENDPOINT	1056
#define IDC_CEP_CEP	1057
#define IDC_CEP_SAVE	1058
#define IDC_CEP_CANCEL	1059
#define IDC_CEP_OPEN	1060
#define IDC_END_OPEN	1061
#define IDC_END_END	1062
#define IDC_END_SAVE	1063
#define IDC_END_CANCEL	1064
#define IDC_RADIO1	1071
#define IDC_RADIO2	1072
#define IDC_RADIO3	1073
#define IDC_RADIO4	1074
#define IDC_RADIO5	1075
#define IDC_RADIO6	1076
#define IDC_END_1	1084
#define IDC_END_2	1085
#define IDC_END_3	1086
#define IDC_END_4	1087
#define IDC_END_5	1088
#define IDC_END_6	1089
#define IDC_REC_1	1092
#define IDC_REC_2	1093
#define IDC_REC_3	1094
#define IDC_PRE_1	1095
#define IDC_PRE_2	1096
#define IDC_PRE_3	1097
#define ID_FILE_CONNECT	40001
#define ID_FILE_DISCONNECT	40002
#define ID_FILE_EXIT	40003
#define ID_BUILD_LPC	40004


```

#define ID_BUILD_COMBINE 40005
#define ID_BUILD_VQLN 40006
#define ID_BUILD_VQCMF 40007
#define ID_BUILD_HMM 40008
#define ID_RECOGNIZATION_MANUAL 40009
#define ID_RECOGNIZATION_SEGRECOG 40010
#define ID_SEGMENTATION_ENERGY 40011
#define ID_SEGMENTATION_SEGMENTING 40012
#define ID_ENDPOINT 40012
#define ID_WAVE_EDITWAVE 40013
#define ID_PREFERENCE_MODEM 40014
#define ID_PREFERENCE_SERIALPORT 40015
#define ID_HELP_ABOUT 40016
#define ID_OPTION_PREFERENCES 40017
#define ID_SEGMENTATION_ZEROCROSSING 40018
#define ID_FILE_TRAINING 40019
#define ID_BUILD_CEP 40020
#define ID_END 40021
#define IDC_STATIC -1

```

```

// Next default values for new objects
//

```

```

#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NO_MFC 1
#define _APS_NEXT_RESOURCE_VALUE 170
#define _APS_NEXT_COMMAND_VALUE 40023
#define _APS_NEXT_CONTROL_VALUE 1098
#define _APS_NEXT_SYMED_VALUE 101
#endif
#endif

```

Speech.h

```

/*-----
MENUDEMO.H header file
-----*/

```

```

#include "windows.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "Math.h"
#include "process.h"

```

```

#define MAXLEN_TEMPSTR 81

```

```

#define MAXROWS 25

```

```

#define MAXCOLS 80

```

```

// #define _MAX_PATH 256

```

```

#define _MAX_EXT 256

```

```

#define _MAX_FNAME 256

```

```

//-----//

```

```

#define GL 55
#define GR 555
#define GT 50
#define GB 350
#define DECX 50
#define DECY 50
#define PX (GR-GL)
#define PY (GB-GT)
#define GRIDX 10
#define GRIDY 10

```

```

#define UNSIGNED_CHAR 0

```

```

#define UNSIGNED_INT 1

```

```

#define UNSIGNED_LONG 2

```

```

#define FLOAT 3

```

```

#define SIGNED_CHAR 4

```

```

#define SIGNED_INT 5

```

```

#define SIGNED_LONG 6

```

```

#define DOUBLE 7

```

```

#define M_PI

```

3.14159265358979323846

```

#define WM_RING (WM_USER + 0)
#define WM_END_RING (WM_USER + 1)

# define ALMOST_LOG_ZERO      (1e-30)
# define REALLY_VERY_BIG      (1e30)

# define WINDOWSIZE  512
# define FRAMESIZE   256
# define NCEPS       12

static char szFilter[] = "Wave Files (*.WAV)\0*.wav\0" \
                        "All Files (*.*)\0*.*\0" ;

static char szFilterSaveData[] = "Data Files (*.DAT)\0*.dat\0" \
                                "All Files (*.*)\0*.*\0" ;

typedef struct {
    unsigned char type;           /* data type 0-7 as defined below */
    unsigned char element_size;  /* size of each element */
    unsigned short int records;  /* number of records */
    unsigned short int rec_len;  /* number of elements in each record */
    char *name;                  /* pointer to file name */
    FILE *fp;                    /* pointer to FILE structure */
} DSP_FILE;

/* FILE HEADER STRUCTURE FOR DSP DATA FILES */
typedef struct {
    unsigned char type;           /* data type 0-7 as defined below */
    unsigned char element_size;  /* size of each element */
    unsigned short int records;  /* number of records */
    unsigned short int rec_len;  /* number of elements in each record */
} HEADER;

typedef struct {
    unsigned int length;          /* size of filter */
    float *history;               /* pointer to history in filter */
    float *coef;                  /* pointer to coefficients of filter */
} FILTER;

/* defines for data type used in DSP data file header and structure */
typedef struct dist_type_struct {
    float distance;
    int index;
} dist_type;

typedef struct
{
    HWND    hwnd ;
    HANDLE  hEvent ;
    BOOL    bContinue ;
}
PARAMS,*PPARAMS ;

typedef struct
{
    unsigned int Start;
    unsigned int Length;
    unsigned int End;
    BOOL fStatus_start;
    BOOL fStatus_end;
    BOOL fStatus_700_front;
    BOOL fStatus_700_rear;
    BOOL fStatus_complete;
}END;

typedef struct
{
    unsigned int Chanel_1;
    unsigned int Chanel_2;
    unsigned int Chanel_3;
    double distance_1;
    double distance_2;
    double distance_3;
}

```



```

}DTW_DATA;

static char *Model_name[30]=
{
    "Mod_1_1.wav", "Mod_1_2.wav", "Mod_1_3.wav", "Mod_1_4.wav", "Mod_1_5.wav",
    "Mod_1_6.wav", "Mod_1_7.wav", "Mod_1_8.wav", "Mod_1_0.wav", "Mod_1_C.wav",
    "Mod_2_1.wav", "Mod_2_2.wav", "Mod_2_3.wav", "Mod_2_4.wav", "Mod_2_5.wav",
    "Mod_2_6.wav", "Mod_2_7.wav", "Mod_2_8.wav", "Mod_2_0.wav", "Mod_2_C.wav",
    "Mod_3_1.wav", "Mod_3_2.wav", "Mod_3_3.wav", "Mod_3_4.wav", "Mod_3_5.wav",
    "Mod_3_6.wav", "Mod_3_7.wav", "Mod_3_8.wav", "Mod_3_0.wav", "Mod_3_C.wav"
};

static char *Wave_name[30]=
{
    "wave_train_1.wav", "wave_train_2.wav", "wave_train_3.wav", "wave_train_4.wa
v",
    "wave_train_5.wav", "wave_train_6.wav", "wave_train_7.wav", "wave_train_8.wa
v",
    "wave_train_9.wav", "wave_train_10.wav", "wave_train_11.wav", "wave_train_12
.wav",
};

static char *Wave_connect[30]=
{
    "wave_connect_1.wav", "wave_connect_2.wav", "wave_connect_3.wav",
    "wave_connect_4.wav", "wave_connect_5.wav", "wave_connect_6.wav",
    "wave_connect_7.wav", "wave_connect_8.wav", "wave_connect_9.wav",
    "wave_connect_10.wav", "wave_connect_11.wav", "wave_connect_12.wav",
};

static char *Wave_channel[30]=
{
    "wave_channel_1.wav", "wave_channel_2.wav", "wave_channel_3.wav",
    "wave_channel_4.wav", "wave_channel_5.wav", "wave_channel_6.wav",
    "wave_channel_7.wav", "wave_channel_8.wav", "wave_channel_9.wav",
    "wave_channel_10.wav", "wave_channel_1.wav", "wave_channel_2.wav",
    "wave_channel_3.wav", "wave_channel_4.wav", "wave_channel_5.wav",
    "wave_channel_6.wav", "wave_channel_7.wav", "wave_channel_8.wav",
    "wave_channel_9.wav", "wave_channel_10.wav", "wave_channel_1.wav",
    "wave_channel_2.wav", "wave_channel_3.wav", "wave_channel_4.wav",
    "wave_channel_5.wav", "wave_channel_6.wav", "wave_channel_7.wav",
    "wave_channel_8.wav", "wave_channel_9.wav", "wave_channel_10.wav"
};

static char *path[31]=
{
    "Mod_1_1.end", "Mod_1_2.end", "Mod_1_3.end", "Mod_1_4.end",
    "Mod_1_5.end", "Mod_1_6.end", "Mod_1_7.end", "Mod_1_8.end",
    "Mod_1_0.end", "Mod_1_C.end",
    "Mod_2_1.end", "Mod_2_2.end", "Mod_2_3.end", "Mod_2_4.end",
    "Mod_2_5.end", "Mod_2_6.end", "Mod_2_7.end", "Mod_2_8.end",
    "Mod_2_0.end", "Mod_2_C.end",
    "Mod_3_1.end", "Mod_3_2.end", "Mod_3_3.end", "Mod_3_4.end",
    "Mod_3_5.end", "Mod_3_6.end", "Mod_3_7.end", "Mod_3_8.end",
    "Mod_3_0.end", "Mod_3_C.end"
};

static char *Mod_open_close[10]=
{
    "Mod_0_1.end", "Mod_0_2.end", "Mod_0_3.end", "Mod_0_4.end",
    "Mod_0_5.end",
    "Mod_C_1.end", "Mod_C_2.end", "Mod_C_3.end", "Mod_C_4.end",
    "Mod_C_5.end",
};

static char *Mod_5[50]=
{
    "Mod_1_1.end", "Mod_1_2.end", "Mod_1_3.end", "Mod_1_4.end",

```



```

"Mod_1_5.end",
"Mod_2_1.end", "Mod_2_2.end", "Mod_2_3.end", "Mod_2_4.end",
"Mod_2_5.end",
"Mod_3_1.end", "Mod_3_2.end", "Mod_3_3.end", "Mod_3_4.end",
"Mod_3_5.end",
"Mod_4_1.end", "Mod_4_2.end", "Mod_4_3.end", "Mod_4_4.end",
"Mod_4_5.end",
"Mod_5_1.end", "Mod_5_2.end", "Mod_5_3.end", "Mod_5_4.end",
"Mod_5_5.end",
"Mod_6_1.end", "Mod_6_2.end", "Mod_6_3.end", "Mod_6_4.end",
"Mod_6_5.end",
"Mod_7_1.end", "Mod_7_2.end", "Mod_7_3.end", "Mod_7_4.end",
"Mod_7_5.end",
"Mod_8_1.end", "Mod_8_2.end", "Mod_8_3.end", "Mod_8_4.end",
"Mod_8_5.end",
"Mod_0_1.end", "Mod_3_2.end", "Mod_3_3.end", "Mod_3_4.end",
"Mod_0_5.end",
"Mod_C_1.end", "Mod_1_2.end", "Mod_1_3.end", "Mod_1_4.end",
"Mod_C_5.end",

```

```
};
```

```
typedef struct
```

```
{
```

```

    int Select_channel;
    BOOL Status_channel_1;
    BOOL Status_channel_2;
    BOOL Status_channel_3;
    BOOL Status_channel_4;
    BOOL Status_channel_5;
    BOOL Status_channel_6;
    BOOL Status_channel_7;
    BOOL Status_channel_8;

```

```
}DTW_RESULT;
```



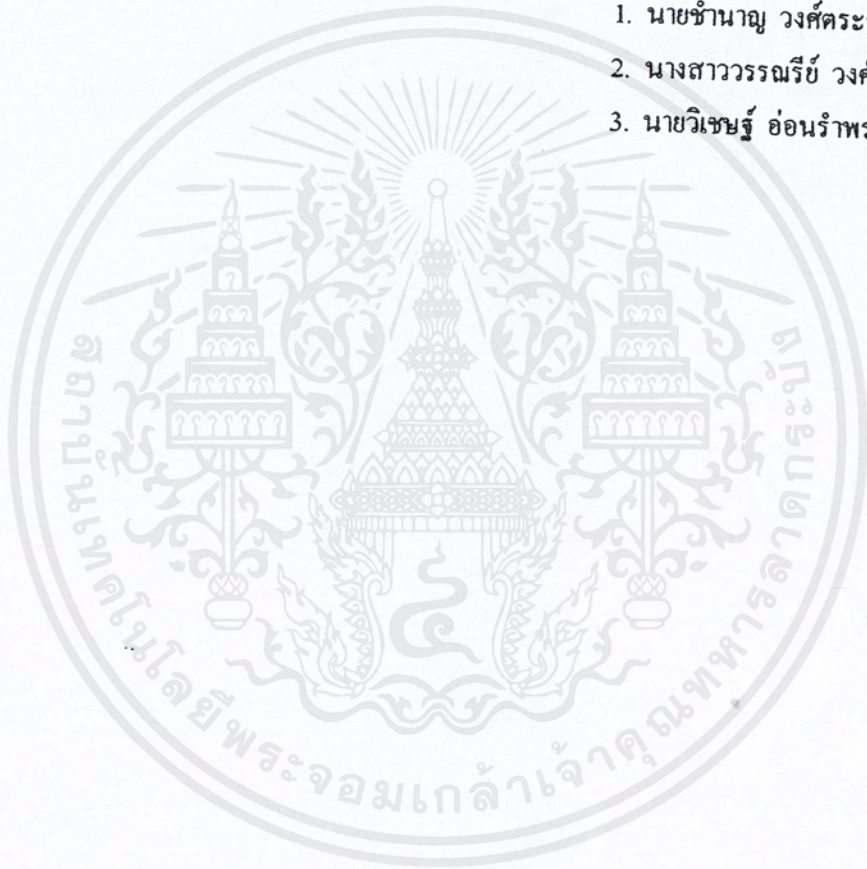
กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้เสร็จสมบูรณ์ได้ด้วยความอนุเคราะห์จากผู้มีพระคุณทุกท่าน โดยเฉพาะอาจารย์สุวิมล สิริชีวะภาค และอาจารย์เกรียงไกร วงศ์โรจนภรณ์ ซึ่งคอยให้คำแนะนำ และคำปรึกษาที่ดีมาโดยตลอด และกำลังใจจากเพื่อน ๆ 3T/1 ทุกคนที่มีให้จนไม่สามารถเอ่ยนามในที่นี้ได้ครบทุกคน

จึงขอแสดงความขอบคุณมา ณ ที่นี้ด้วย

คณะผู้จัดทำ

1. นายชำนาญ วงศ์ตระกูลชัย
2. นางสาววรรณรีย์ วงศ์ไครรัตน์
3. นายวิเชษฐ์ อ่อนรำพรรณ



หนังสืออ้างอิง

- [1] กิตติกรณ์ ตัญยาบุทท์ และวันประชา เขาวลิตวงศ์, “การส่งงานอุปกรณ์ไฟฟ้าผ่านคู่สายโทรศัพท์โดยใช้เสียง”, ปรินญานินพนธ์ สาขาวิศวกรรมโทรคมนาคม คณะวิศวกรรมศาสตร์, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, 2541
- [2] ทศเวท วีระวัฒน์, “การรู้จำเสียงคำไทยเฉพาะบุคคล”, วิทยานิพนธ์ สาขาวิศวกรรมไฟฟ้า บัณฑิตวิทยาลัย สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, 2541
- [3] สุรชัย เพิ่มสินทวี, “นำทางสู่การใช้งาน การสื่อสารข้อมูลด้วยระบบ MODEM”, ซีเอ็ดยุคเข่นจำกัด(มหาชน), 2537
- [4] ชัยวัฒน์ อัมพรจิตรวิไล, “เรียนรู้และปฏิบัติการไมโครคอนโทรลเลอร์ MCS-51”, กรุงเทพมหานคร, อินโนเวทีฟเอ็กเพอริเมนต์, 2542
- [5] Claudio Beechetti and Lucio Prina Ricotti, ” Speec Recognition theory and C++ Implementation ”, John Wiley & Sons Ltd., New York , USA ,1999

