โปรแกรมลดขนาดไฟล์ .WAV โดยใช้วิธีเดลต้ามอดูเลชั่น

(COMPRESS .WAV FILE PROGRAM

BY

DELTA MODULATION ALGORITHM)

โดย

นายสมศักดิ์ พึ่งธรรมเกิดผล

นายอาณัตต์ อ่วมเจริญ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาตรบัณฑิต

สาขาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2537

ปริญญานิพนธ์ปีการศึกษา 2537

ภาควิชา วิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง โปรแกรมลดขนาดไฟล์ .wav โดยใช้วิธีเดลต้ามอดูเลชั่น

ผู้จัดทำ

1.นายสมศักดิ์ พึ่งธรรมเกิดผล 34107398

2.นายอาณัตต์ อ่วมเจริญ 34109510

.....

(อาจารย์ยุทธพงษ์ รังสรรค์เสรี)

อาจารย์ที่ปรึกษา

สารบัญ

บทที่ 1
บทนำ

เนื่องจากความก้าวหน้าทางเทคโลยี การส่งสัญญาณแบบดิจิตอลเป็นที่นิยมมากขึ้นเนื่องจากสัญญาณแบบดิจิตอลปราศจากสิ่งรบกวนจากที่อื่นโดยเฉพาะนอยส์ (noise) และการแทรกซ้อนที่จะเข้ามาปะปน หรือกล่าวอีกนัยคือ สัญญาณดิจิตอลจะถูกสร้างขึ้นมาใหม่ได้โดยมีการผิดเพี้ยนน้อยมาก นอกจากนี้การส่งแบบนี้มีประสิทธิภาพสูงและเหมาะสมกับการบริการในรูปแบบต่างๆ ทั้งเสียง, ข้อมูลและภาพ แม้ว่าสัญญาณดิจิตอลก็มีข้อเสียบางข้อ เช่น มีแบนด์วิดธ์ (bandwidth) กว้างมีน้อยส์แฝงอยู่ในตัว และอุปกรณ์สร้างสัญญาณใหม่มีราคาแพง แต่มีแนวโน้มที่อุปกรณ์ต่างๆ จะมีราคาลดลงเรื่อยๆ

การส่งสัญญาณแบบดิจิตอลที่เป็นที่นิยมวิธีหนึ่งคือ Pulse Code Modulation (PCM)ซึ่งส่งเป็นรหัส 8 บิท แต่ยังมีการเข้ารหัสอีกแบบหนึ่งคือ Delta Modulation (DM) ที่ใช้การเข้ารหัสเพียง 1 บิทต่อหนึ่งตัวอย่างซึ่งเป็นการประหยัดจำนวนบิทในการส่งสัญญาณเป็นอย่างมาก แต่วิธีนี้ก็ยังพบปัญหาหลายอย่างที่จะแสดงให้เห็นในบทต่อๆ ไป

จากเหตุผลข้างต้นจึงมีการพัฒนาการเข้ารหัสแบบใหม่โดยการปรับปรุง DM ซึ่งก็คือ ระบบ Adaptive Delta Modulation (ADM) ที่ใช้การเข้ารหัส 1 บิทต่อหนึ่งตัวอย่างเช่นเดียวกับ DM ระบบนี้สามารถทำได้หลายรูปแบบ แต่ในการศึกษาครั้งนี้จะพิจารณาเฉพาะ Hybrid Companding DM (HCDM) เท่านั้น โดยจะจำลองการทำงานของการมอดูเลชั่นด้วยซอฟแวร์ (software) ทั้งวิธีการมอดูเลชั่นแบบลิเนียร์และวิธีการมอดูเลชั่นแบบปรับค่า (ซึ่งใช้วิธีการของ HCDM) โดยการปฏิบัติกับแฟ้มข้อมูล (file) ที่มีนามสกุลเป็น .WAV ซึ่งได้จาก Sound Blaster เพื่อเป็นแนวทางในการศึกษา HCDM ต่อไป

โปรแกรมลดขนาดไฟล์ .wav โดยใช้วิธีเดลต้ามอดูเลชั่น

(Compress .wav File Program by Delta Modulation Algorithm)

โดย นายสมศักดิ์ พึ่งธรรมเกิดผล
นายอาณัตต์ อ่วมเจริญ

อาจารย์ที่ปรึกษา อาจารย์ยุทธพงษ์ รังสรรค์เสรี

บทคัดย่อ

การใช้คอมพิวเตอร์ในปัจจุบันนี้ ส่วนใหญ่มักจะเน้นไปในรูปของมัลติมีเดียร์ ซึ่งที่ใช้กันมากก็เป็นการใช้ในด้านเสียงโดยมีโปรแกรมสนับสนุนการบันทึกเสียงลงสู่แผ่นเก็บข้อมูล เช่นโปรแกรม Sound Blaster ซึ่งไฟล์ที่บรรจุเสียงนี้ โดยมากจะมีขนาดใหญ่มากทำให้การเคลื่อนย้ายไฟล์ระหว่างเครื่องคอมพิวเตอร์กระทำได้ลำบาก ในโครงงานนี้จึงได้จัดทำโปรแกรมลดขนาดไฟล์ .WAV ที่ได้จาก Sound Blaster โดยใช้เทคนิคของเดลต้ามอดูเลชั่นทั้งแบบลิเนียร์และแบบปรับค่าได้ เพื่อลดขนาดไฟล์นี้ให้มีขนาดเล็กลงทำให้การเคลื่อนย้ายไฟล์กระทำได้สะดวกยิ่งขึ้น ทั้งยังได้นำมาเปรียบเทียบให้เห็นข้อแตกต่างระหว่างทั้งสองวิธีนี้ด้วยว่าวิธีการใดเหมาะสำหรับการใช้งานมากกว่ากัน

ABSTRACT

Nowaday computer usage must concentrates to the multimedia usage that focus to sound utility by program used to record sound to disks. The example of this program is Sound Blaster. The sound files must have very big size thus it's difficult to transfer this files between the computers. This thesis is the file .WAV compression. .WAV file is producted by Sound Blaster program. The technique of program uses linear delta modulation and adaptive delta modulation algorithm to compress file. The thesis will show the difference between two algorithms and tell you that which one is better for your usage.

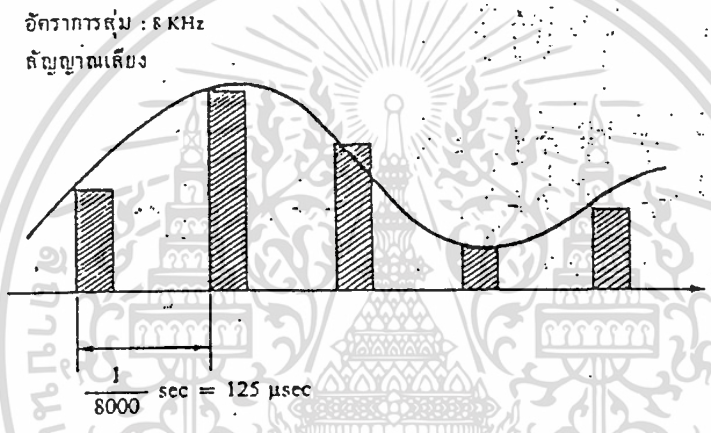
บทที่ 2

พัลส์โค๊ดมอดูเลชั่น

(Pulse Code Modulation)

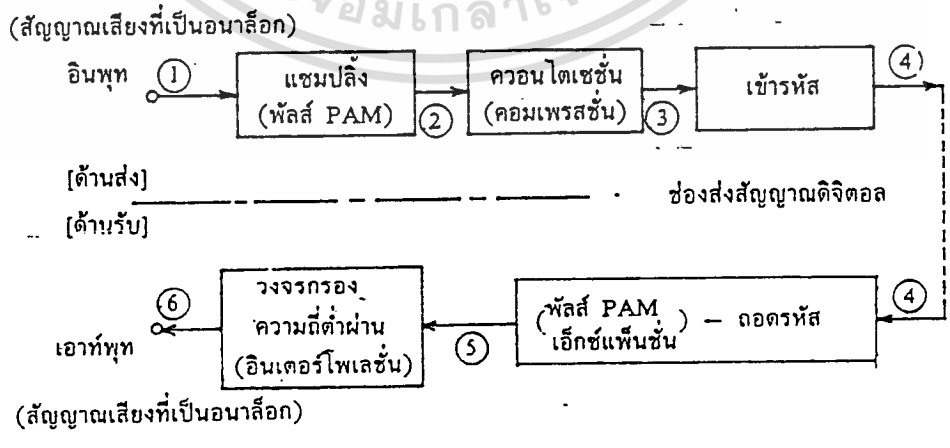
2.1 กล่าวนำ

ในปัจจุบันระบบ PCM ถูกนำไปใช้อย่างกว้างขวางโดยเฉพาะสำหรับสัญญาณเสียงและในระยะหลังๆ นี้การพัฒนาระบบนี้เพื่อนำไปใช้กับสัญญาณภาพก็มีบทบาทขึ้นด้วย ดังนั้นในบทนี้จะกล่าวถึงขั้นตอนของกระบวนการ PCM อย่างคร่าวๆ



รูปที่ 2.1 ความถี่ของสัญญาณสุ่มค่า 8 kHz สำหรับสัญญาณเสียง

2.2 กระบวนการเข้ารหัสและถอดรหัสของ PCM

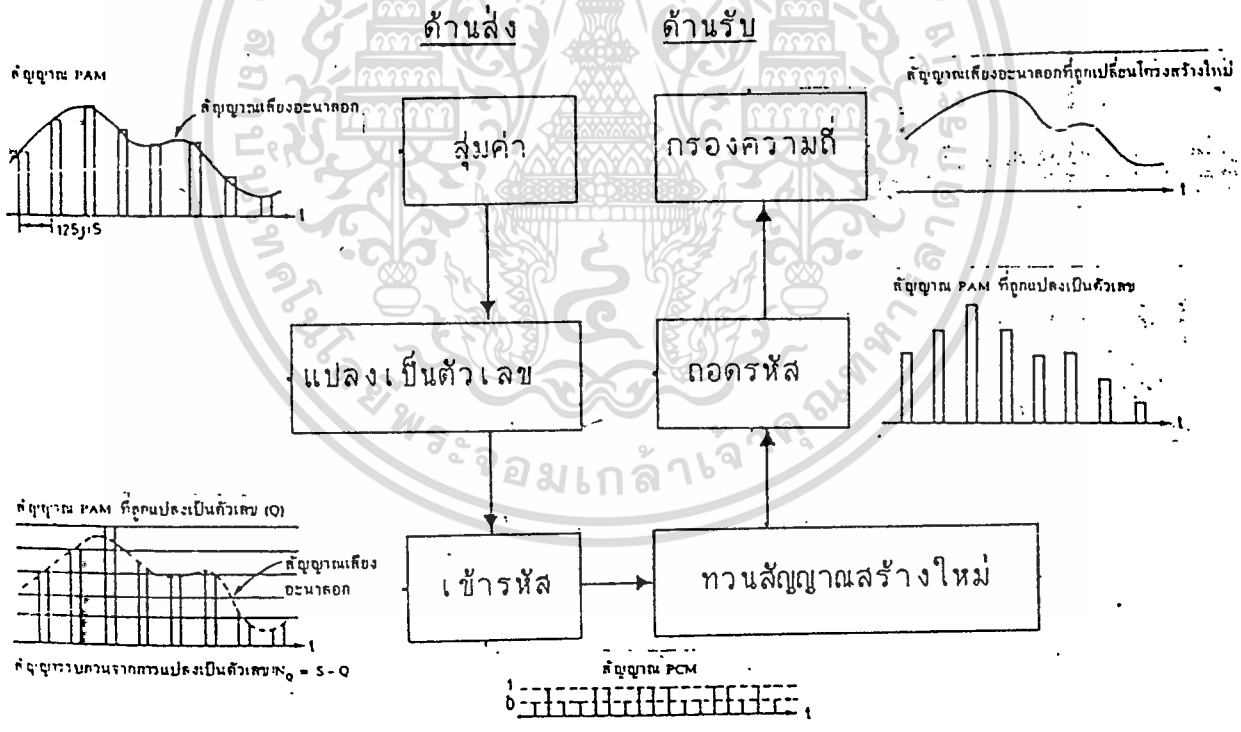


รูปที่ 2.2 กระบวนการเข้ารหัสและถอดรหัสของระบบ PCM

รูปที่ 2.2 แสดงขั้นตอนการประมวลสัญญาณเพื่อให้ได้รับ PCM อย่างกว้างๆ ก็คือการเข้ารหัส (coding) และการนำสัญญาณไปแปลงกลับซึ่งเรียกว่าการถอดรหัส (decoding) เพื่อให้ได้สัญญาณเดิม สำหรับรูป 2.3 แสดงลักษณะคลื่นในโดเมนเวลาและสเปคตรัมในโดเมนความถี่ของแต่ละขั้นตอนในกระบวนการนี้ ซึ่งหลักการของแต่ละขั้นตอนจะกล่าวให้ละเอียดดังต่อไปนี้

2.3 การแซมปลิ้ง (Sampling)

แซมปลิ้งคือการทำสัญญาณซึ่งมีค่าต่อเนื่องให้เป็นแบบดิสครีทในช่วงเวลาที่เท่าๆ กันโดยใช้ทฤษฎีการแซมปลิ้งที่ว่า ถ้าเก็บแซมปลิ้งด้วยอัตรา 2 เท่าหรือมากกว่าความถี่สูงสุดของสัญญาณอนาล็อกแล้ว จะสามารถทำให้สัญญาณเดิมกลับคืนมาได้อย่างถูกต้อง สัญญาณที่ผ่านการแซมปลิ้งแล้วจะได้สัญญาณ PAM (Pulse Amplitude Modulation)

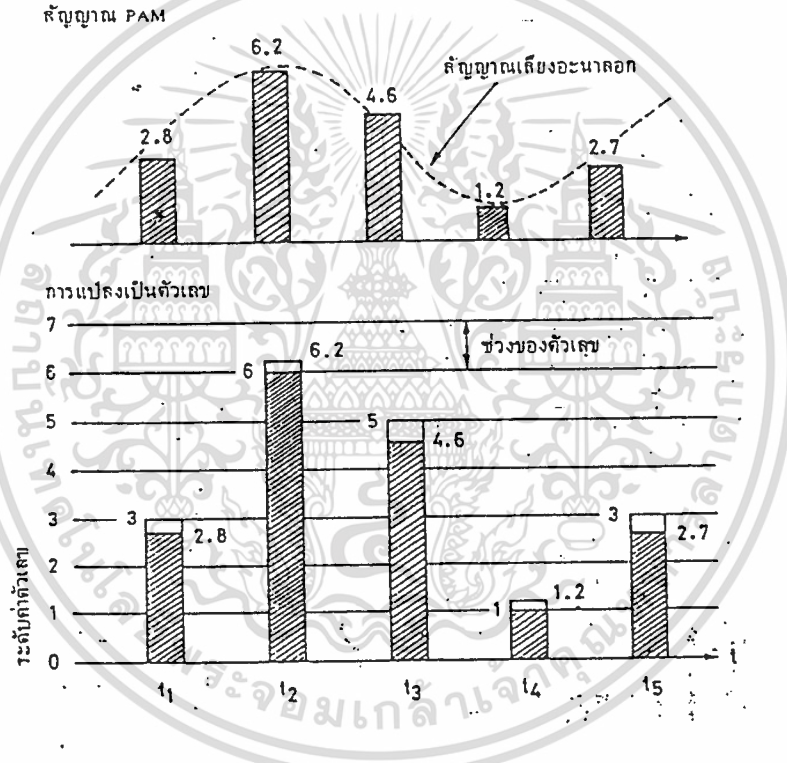


รูปที่ 2.3 การแปลงสัญญาณอนาล็อกไปเป็นสัญญาณดิจิตอล และ
สัญญาณดิจิตอลเป็นสัญญาณอนาล็อก

2.4 การจัดระดับ (Quantizing)

ขบวนพัลส์ PAM ที่ผ่านการแซมปลิ้งแล้ว ยังถือว่าเป็นชนิดอนาล็อกอยู่ คือมันจะมีขนาดเปลี่ยนแปลงอย่างต่อเนื่องไปกับเวลาที่เป็นช่วงๆ การจัดระดับคือ กระบวนการเปลี่ยนขนาดของ PAM เหล่านั้นให้เป็นค่าตัวเลขแบบดิสครีทตามที่แสดงไว้ในรูป 2.4

จากรูปที่ 2.4 แอมปลิจูดของแซมเปิ้ลทุกตัวของ PAM จะถูกจัดให้เป็นระดับ ซึ่งเรียกว่าระดับควอนไตซ์ (Quantizing level) โดยมีระยะห่างระหว่างระดับข้างเคียง เรียกว่าควอนไตซิ่งอินเทอร์วัล (Quantizing interval) หรือควอนไตซิ่งสเต็ปนั้นเท่ากัน กรณีนี้เรียกว่าการจัดระดับแบบยูนิฟอร์มหรือแบบลิเนียร์ (uniform quantizing)

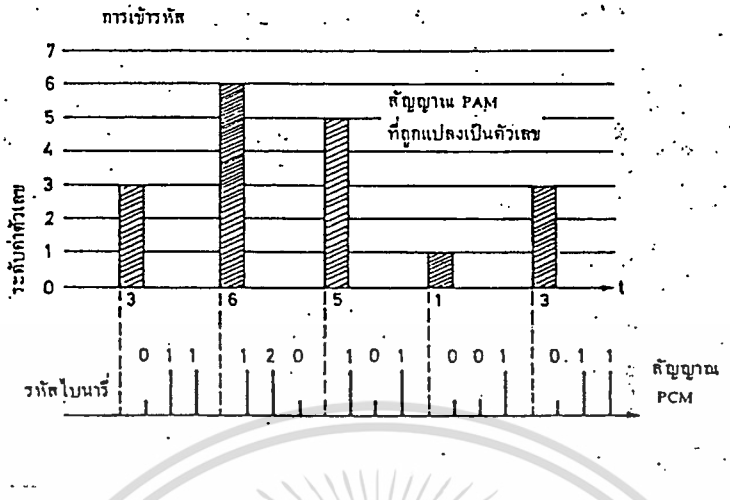


รูปที่ 2.4 การแปลงสัญญาณ PAM ให้เป็นตัวเลข(การจัดระดับ)

ขนาดของแซมเปิ้ลทุกตัวจะแสดงด้วยค่าระดับควอนไตซ์ที่ใกล้เคียงที่สุด จะเห็นได้ว่าสัญญาณ PAM ที่ถูกจัดระดับแล้วนี้จะเป็นเพียงค่าโดยประมาณของสัญญาณอนาล็อกเท่านั้น ดังนั้นส่วนเกินและส่วนขาดจากการจัดระดับจึงเป็นค่าผิดพลาดระหว่างสัญญาณเดิมและค่าที่ได้จัดระดับซึ่งค่าผิดพลาดนี้เรียกว่า ควอนไตซิ่งน้อยส์(quantizing noise) หรือการพร่าเพี้ยนจากการควอนไตซ์ (quantizing distortion)

ในการทำให้ควอนไตซิ่งน้อยส์นี้ลดลงในเบื้องต้นคือการลดควอนไตซิ่งอินเทอร์วัลให้แคบลงอย่างพอเพียงก็สามารถจะลดควอนไตซิ่งน้อยส์ได้ในระดับหนึ่ง เช่นถ้าลดควอนไตซิ่งอินเทอร์วัลลงครึ่งหนึ่ง ปริมาณของควอนไตซิ่งน้อยส์จะลดลงเป็น 1/4 และลดควอนไตซิ่งอินเทอร์วัลให้เหลือครึ่งหนึ่งนั้นจะสุดท

คล้องกับการเพิ่มจำนวนบิทอีก 1 บิท นั่นคือเพาเวอร์ของควอนไตซิ่งน้อยส์จะลดลง 6 dB ทุกๆ การเพิ่ม 1 บิท



รูปที่ 2.5 การนำสัญญาณ PAM มาเข้ารหัส

2.5 การเข้ารหัส (Coding)

หลังจากขบวนพัลส์ PAM ได้ผ่านขั้นตอนการจัดระดับมาแล้ว จะต้องเปลี่ยนขนาดเหล่านั้นให้เป็นรหัสไบนารี่ (binary code) กรณีที่เป็นสัญญาณเสียงสำหรับการส่งทางโทรศัพท์ จะถูกเปลี่ยนเป็นรหัส 8 บิท ซึ่งสามารถแสดงค่าแอมปลิจูดได้ 2^8 ระบบการเข้ารหัสจะมีหลายๆ แบบ แต่ส่วนมากจะใช้กัน 3 แบบดังแสดงไว้ในตาราง 2.1 ซึ่งแสดงไว้เพียง 3 บิทเท่านั้น

ตาราง 2.1 รหัสไบนารีแบบต่างๆ

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

บทที่ 3
เดลต้ามอดูเลชั่น
(Delta Modulation)

3.1 หลักการทำงานของเดลต้ามอดูเลชั่น (DM)

ในรูปแบบปกติ สัญญาณที่ได้จาก DM จะให้สัญญาณเป็นรูปขั้นบันได (stair case) ซึ่งประมาณได้จากสัญญาณอินพุท (input signal) หรือสัญญาณเบสแบนด์ (base band) ดังแสดงในรูปที่ 3.1.1 ความแตกต่างระหว่างอินพุทและสัญญาณที่ได้จากค่าประมาณ (approximate) จะถูกเปลี่ยนเป็นสัญญาณ 2 ระดับคือ +d และ -d เท่านั้น หลักการทำการประมาณค่าสัญญาณนั้น ถ้าสัญญาณอินพุทมีค่าสูงกว่าสัญญาณที่ได้จากการประมาณค่าที่ตำแหน่งของการแซมปลิ้ง (sampling) ครั้งก่อนก็จะทำการเพิ่มสัญญาณค่าประมาณขึ้นอีก+d ในทางกลับกัน ถ้าสัญญาณค่าประมาณมีค่ามากกว่าสัญญาณอินพุทมันก็จะถูกลดลง -d เช่นกัน ถ้าสัญญาณเปลี่ยนแปลงไม่เร็วเกินไป เราจะพบว่าขั้นบันไดการประมาณ (stair case approximate) จะยังคงสูงหรือต่ำกว่าสัญญาณอินพุทไม่เกิน +d หรือ -d

ในสัญญาณอินพุทเป็น m(t) และขั้นบันไดการประมาณเป็น $m_a(t)$ แล้ว หลักการเบื้องต้นของ DM สามารถเขียนเป็นสมการได้ดังนี้

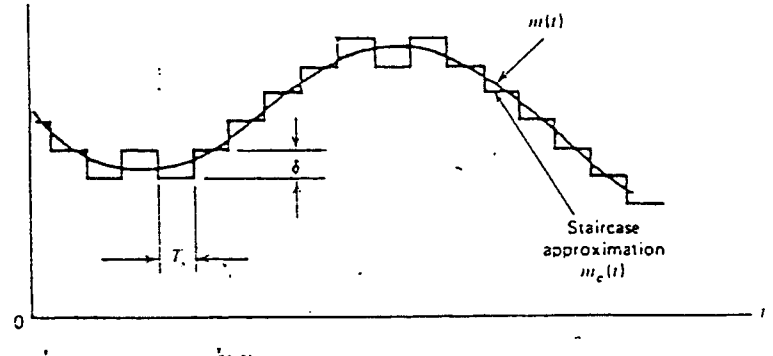
$$b_n = \text{sgn}\,[m(n.Ts) - m_a(n.Ts - Ts)] \qquad ...(3.1)$$

$$m_a(n.Ts) = m(n.Ts - Ts) + d.b_n \qquad ...(3.2)$$

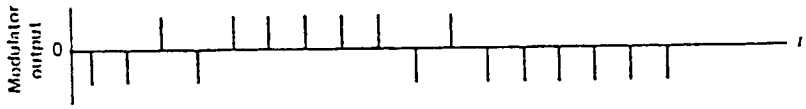
โดย T_s เป็นเวลาของการแซมปลิ้ง และ b_n เป็นเครื่องหมายที่ได้

สำหรับแต่ละตัวอย่าง (sample) การส่งจะส่งเป็นข้อมูล 1 bit คือ b_n และ อัตราของการส่งผ่านข้อมูลจะเท่ากับอัตราการแซมปลิ้ง 1/Ts ดังแสดงในรูปที่ 3.1.2

ความถี่ที่ใช้ในการแซมปลิ้งจะต้องมากกว่าหรือเท่ากับ 2 เท่าของความถี่สูงสุดของความถี่ที่เราต้องการแซมปลิ้งนั้น

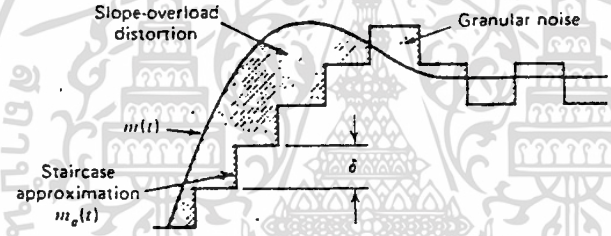


รูปที่ 3.1.1 สัญญาณที่ได้จาก DM



รูปที่ 3.1.2 การส่งข้อมูลจะส่งเป็น 1 บิทต่อ 1 ตัวอย่าง(sample)

การแซมปลิ้งของระบบ PCM ซึ่งใช้เข้ารหัสสัญญาณเสียงความถี่ประมาณ 0-4 kHzจึงต้องใช้ความถี่ในการแซมปลิ้ง 8 kHzขึ้นไป แต่ในทางปฏิบัติจะใช้ความถี่สูงกว่านี้มากเพื่อลดความผิดพลาด ซึ่งในระบบ DM จะเกิดความผิดพลาดหรือคลาดเคลื่อนที่เรียกว่า ความผิดพลาดควอนไตซิ่ง (Quantizing error) ซึ่งมี 2 อย่างคือความผิดพลาดที่เกิดจากสัญญาณมีความชันในการเปลี่ยนแปลงค่ามากเกินไปหรือสโลปโอเวอร์โหลดดิสทอร์ชั่น (slope overload distortion) และความผิดพลาดที่เรียกว่าแกนูล่าน้อยส์ (granular noise) ดังแสดงในรูปที่ 3.2

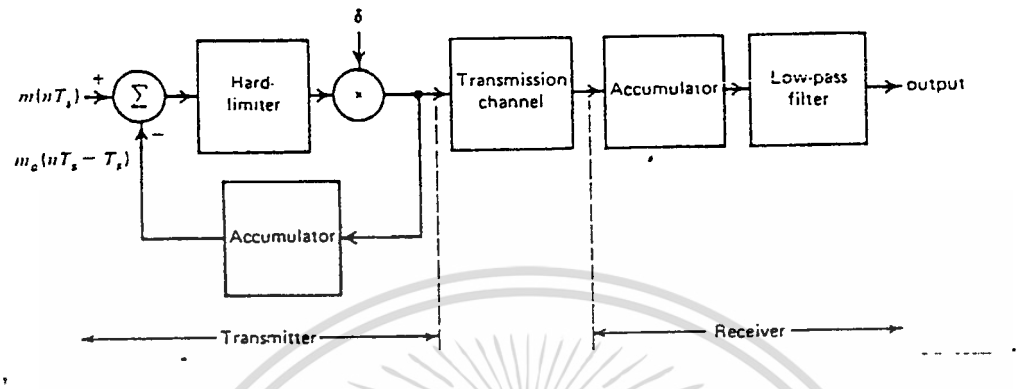


รูปที่ 3.2 แสดงการเกิด quantizing error ใน DM

สโลปโอเวอร์โหลดดิสทอร์ชั่นเกิดขึ้นเมื่อขนาดของสเต็ป (step) d เล็กเกินไปที่จะทำให้ขั้นบันไดการประมาณ $m_a(t)$ ตามสัญญาณอินพุท m(t) ได้ทัน และแกนูล่าน้อยส์จะเกิดขึ้นเมื่อขนาดของสเต็ป d ใหญ่เกินไป สัญญาณอินพุทซึ่งเป็นผลมาจากสัญญาณ $m_a(t)$ สเต็ปขึ้นและลงระหว่างส่วนที่ค่อนข้างราบเรียบของอินพุท แกนูล่าน้อยส์ในระบบ DM นี้เทียบเท่าควอนไตซิ่งน้อยส์ในระบบ PCM ดังนั้นสำหรับสัญญาณที่มีค่าความชัน (slope) อันหนึ่ง ถ้าขนาดของสเต็ปเล็กก็จะทำให้เกิดสโลปโอเวอร์โหลดดิสทอร์ชั่น ในขณะที่ขนาดของสเต็ปใหญ่จะทำให้เกิดแกนูล่าน้อยส์

สำหรับหลักการของ DM แบบธรรมดานั้น สามารถแสดงในบล็อกไดอะแกรมในรูป 3.3 ซึ่งประกอบด้วย summer, hard limiter และ accumulator หรืออินทีเกรเตอร์ (integrator) นั่นเอง โดยตัว summer จะเปรียบเทียบความแตกต่างระหว่างสัญญาณอินพุท m(n.Ts) และเอ้าท์พุทm_a(n.Ts - Ts) จาก accumulator ค่าความแตกต่างนี้จะผ่านเข้าไปใน hard limiter ซึ่งจะให้เอ้าท์พุทออกมาเป็น +1 หรือ -1ขึ้นอยู่กับค่าความแตกต่างแล้วเอ้าท์พุทจาก hard limiter จะถูกคูณด้วย d และจะถูกส่งเข้าไปสู่ accumulator ซึ่งทำหน้าที่สร้างสัญญาณค่าประมาณดังสมการ

$$m_a(n.Ts-Ts) = \sum_{i=1}^{n-1} db_i \qquad ...(3.3)$$



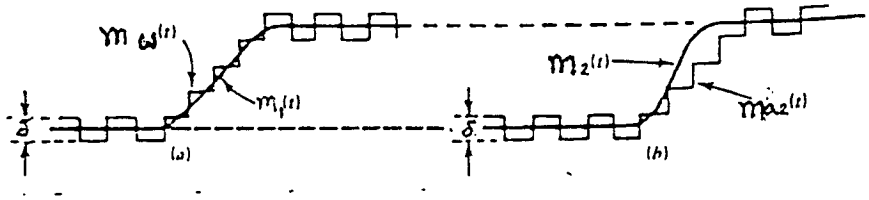
รูปที่ 3.3 block diagram ของระบบ DM

ดังนั้นทุกชั่วขณะของการแซมปลิ้ง ตัว accumulator จะเพิ่มสัญญาณค่าประมาณด้วยขนาดสเต็ป (d) ตามทิศทางของ b_i ถ้าสัญญาณแซมปลิ้ง m(n.Ts) มากกว่าสัญญาณค่าประมาณ m_a(n.Tss) ค่า +d ก็จะถูกเพิ่มให้กับสัญญาณค่าประมาณ ถ้าสัญญาณตัวอย่าง m(n.Ts) มีค่าน้อยกว่าค่าประมาณ m_a(n.Ts-Ts) ค่า -d จะถูกเพิ่ม (ลดลง d)ให้กับ accumulator โดยวิธีการนี้ accumulator จะทำการสร้างสัญญาณใหม่ตามสัญญาณอินพุทโดยใช้ 1 สเต็ปในแต่ละครั้งของการแซมปลิ้งพบว่าพัลส์ที่ได้ (+d และเหล่านี้จะประกอบเป็นสัญญาณดิจิตอล (digital) เพื่อส่งไปทางด้านรับ ในทางด้านรับก็จะสร้างขั้นบันได $m_a(t)$ ออกมา โดยนำพัลส์บวกและลบที่ได้รับผ่าน accumulator ซึ่งมีการทำงานเช่นเดียวกับทางด้านส่ง ส่วนควอนไตซิ่งน้อยส์ที่มีอยู่ใน $m_a(t)$ จะถูกกำจัดออกไปได้โดยผ่านวงจรกรองความถี่ต่ำผ่าน (low pass filter) ซึ่งมีแบนด์วิธ (bandwidth) เท่ากับแบนด์วิธของสัญญาณเบสแบนด์

3.2 Slope Overloading

ปัญหาใหญ่ในระบบ DM คือการเกิดสโลปโอเวอร์โหลด เมื่อสัญญาณอินพุท m(t)เปลี่ยนแปลงสัญญาณค่าประมาณ $m_a(t)$ จะสเต็ปตามสัญญาณอินพุทไปตลอดตราบเท่าที่สัญญาณตัวอย่างยังมากหรือน้อยกว่าสัญญาณ $m_a(t)$ ซึ่งสเต็ปตามสัญญาณ m(t) ไม่ทันนั้น ก็จะเกิดเป็นความผิดพลาดที่เรียกว่าโอเวอร์โหลด (overload) นี้เกิดขึ้นเนื่องจากความชันของสัญญาณ m(t) เราเรียกว่าสโลปโอเวอร์โหลด (slope overload) ดังแสดงในรูปที่ 3.4

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



รูปที่ 3.4 การเกิด slope overload

ดังนั้นความชันสูงสุดของสัญญาณคือ

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

การเปลี่ยนแปลงมากที่สุดในช่วงระหว่างแซมเปิ้ลของสัญญาณอินพุทคือ $A.2\pi f_m.Ts$ เพื่อหลีกเลี่ยงการเกิดสโลปโอเวอร์โหลด การเปลี่ยนแปลงของแซมเปิ้ลนี้ต้องน้อยกว่า d นั่นคือ

$$2\pi f_m.Ts.A < d$$

หรือขนาด (amplitude) ที่ซึ่งสโลปโอเวอร์โหลด จะเกิดคือ

$$A = (d.f_s)/(2\pi f_m) \qquad ...(3.4)$$

ซึ่ง $f_s = 1/Ts$ เป็นอัตราการแซมปลิ้งของระบบ DM ได้มีการตรวจสอบโดยการทดลองพบว่า DM จะส่งสัญญาณเสียงโดยปราศจากการเกิดสโลปโอเวอร์โหลดนั้น ขนาดของสัญญาณจะต้องไม่เกินกว่าขนาดมากที่สุดของสัญญาณในสมการที่ (3.4) ซึ่งใช้กับ f_m 800 Hz ปัญหาของการเกิดสโลปโอเวอร์โหลดในระบบ DM สามารถแก้ไขได้โดยการใช้วงจร กรองสัญญาณ หรือโดยสัญญาณเพื่อจำกัดอัตราสูงสุดของการเปลี่ยนแปลงการเพิ่มขนาดของสเต็ป หรือเพิ่มอัตราการแซมปลิ้งจะทำให้ต้องใช้แบนด์วิธมากขึ้น

ทางที่ดีที่สุดที่จะหลีกเลี่ยงสโลปโอเวอร์โหลด โดยการตรวจสอบเงื่อนไขการเกิดโอเวอร์โหลด และทำให้สเต็ปมีขนาดมากขึ้น เพื่อตรวจสอบได้ว่าเกิดโอเวอร์โหลด ซึ่งเป็นแบบที่ใช้ขนาดของสเต็ปไม่คงที่นี้เรียกว่า "Adaptive Delta Modulation(ADM)"

3.3 สัญญาณรบกวน (NOISE) ในระบบ DM

เอ้าท์พุทของด้านดีโคดเดอร์ (decoder) จะแตกต่างจากทางด้านอินพุท เนื่องมาจากควอนไตซิ่งน้อยส์ $n_q(t)$ และสัญญาณรบกวนที่เกิดขึ้นในขณะทำการส่ง $n_o(t)$ ดังนั้น

$$m_1(t) = m_0(t) + n_q(t) + n_o(t) \qquad ...(3.5)$$

ซึ่ง $m_1(t)$ เป็นเอ้าท์พุทของดีมอดูเลเตอร์ที่ผ่านวงจรกรองสัญญาณความถี่ต่ำผ่าน (LPF) แล้ว

$m_0(t)$ เป็นสัญญาณเอ้าท์พุท สมมุติเท่ากับ $m(t)$

$n_o(t)$ และ $n_q(t)$ เป็นสัญญาณรบกวนที่เอ้าท์พุทเมื่อผ่านวงจรกรองความถี่แล้ว

คุณภาพของสัญญาณทั้งหมดในระบบ DM ถูกวัดในเทอมของ signal-to-noise ratio (S/N) ซึ่งอัตราส่วนนี้หาได้จาก

$$(S/N)_o = (E[\{m_o(t)\}^2]) / (E[\{n_q(t)\}^2] + E[\{n_o(t)\}^2])$$

พลังงานเฉลี่ยที่มีสัญญาณรบกวนอยู่สามารถคำนวณได้ดังต่อไปนี้

3.3.1 ควอนไตซิ่งน้อยส์ในระบบ DM

เราให้ $m(t) = m_a(t) + e_q(t)$ ซึ่ง $|e_q(t)| = | m(t) - m_a(t)| < d$ ในขณะที่เกิดสโลปโอเวอร์โหลดควอนไตซิ่งน้อยส์ $n_q(t)$ ในสมการ (3.5)เป็นผลของฟิลเตอร์แก่ $e_q(t)$ ถ้าเราสมมุติ uniform สำหรับ $e_q(t)$ ดังนั้น

$$E[\{e_q(t)^2\}] = \int (1/2d).e^2 \, de$$
$$= d^2/3$$

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

$$G_{eq}(f) = \begin{cases} d^2/6f_s & , f < f_s \\ 0 & , \text{กรณีอื่นๆ} \end{cases}$$

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

$$E[\{n_q(t)\}^2] = \int G_{eq}(f)\, df$$
$$= (d^2/3).(f_m/f_s) \qquad ...(3.6)$$

ในการที่จะคำนวณค่า singnal to quantizing noise power ratio เราต้องคำนวณหา signal power $E[m_o^2(t)]$ เพื่อความง่ายในการคำนวณ เราจะคิดกรณีที่แย่ที่สุดสำหรับ DM ซึ่งกำลังงานของสัญญาณทั้งหมดมารวมกันที่ขอบบนของเบสแบนด์ ดังนั้นเราให้

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

ดังนั้น

$$m_o(t) = A\cos 2f_m t$$

และ

$$E[m_o^2(t)] = A^2/2 \qquad ....(3.7.1)$$

และเพื่อหลีกเลี่ยงการเกิดสโลปโอเวอร์โหลดจากสมการ (3.4) เรามี

$$A = (d/2\pi).(f_s/f_m) \qquad ....(3.7.2)$$

รวมสมการ (3.6) กับ (3.7) เราจะได้ singnal to quantizing noise ower ratio คือ

$$\{E[m_o^2(t)]\}/\{E[n_q^2(t)]\} = (3/8\pi^2).(f_s/f_m)^3 \qquad ...(3.8)$$

3.4 การเปรียบเทียบระหว่าง DM กับ PCM

เราสามารถเปรียบเทียบประสิทธิภาพของ DM กับ PCM ได้ในเทอมของคุณภาพสัญญาณและความยุ่งยากในวงจรเพื่อให้การเปรียบเทียบได้กระทำภายใต้เงื่อนไขอันเดียวกัน เราสมมุติว่าระบบทั้งสองใช้แบนด์วิธในการส่งเท่ากัน ถ้าเราให้ f_s เป็นอัตราการแซมปลิ้งของ N-bit PCM และ f_m เป็นอัตราการแซมปลิ้งของ DM ดังนั้นบิทเรตสำหรับระบบทั้งสองคือ Nf_s และ f_s ตามลำดับ ถ้าสเปคตรัมของสัญญาณขยายไปถึง f_m Hz ดังนั้น $_s = 2f_m$ และแบนด์วิธที่เทียบเท่ากันที่ต้องการสำหรับ DM คือ

$$f_s = 2Nf_m$$

3.4.1 เปรียบเทียบ signal to noise ratio

ถ้าช่อง S/N มีค่าสูงดังนั้นประสิทธิภาพของ PCM และ DM ถูกจำกัดโดยควอนไตซิ่งน้อยส์ กำลังงานS/N สำหรับ PCM คือ

$$(S_o/N_q)_{PCM} = Q^2 = 2^{2N} \quad ; N > 2 \qquad ...(3.9)$$

ซึ่ง $Q = 2^N$ เป็นจำนวนระดับที่ทำการควอนไตซ์

สำหรับ DM กำลังงาน S/N ถูกกำหนดโดยสมการ(3.8) จะได้

$$(S_o/N_q)_{DM} = (3/8\pi^2).(f_s/f_m) \approx 0.3N^3 \qquad ...(3.10)$$

จากสมการทั้งสองจะเห็นว่าถ้าแบนด์วิธเท่ากันแล้ว ประสิทธิภาพของ DM จะด้อยกว่า PCM เสมอ เช่นถ้าใช้ช่องแบนด์วิธสำหรับ 8-bit PCM จะได้

$$(S_o/N_q)_{PCM} = 48 \text{ dB}$$

$$(S_o/N_q)_{DM} = 22 \text{ dB}$$

ประสิทธิภาพของ DM สามารถปรับปรุงให้ดีขึ้นโดยใช้ระบบ ADM ซึ่งทำให้ประสิทธิภาพระหว่างPCM และ ADM แตกต่างกันบ้างเพียงเล็กน้อยเท่านั้นสำหรับบิทเรต 64kb/s

3.4.2 เปรียบเทียบแบนด์วิธที่ต้องการ

เนื่องจาก PCM และ DM พิจารณาเฉพาะสำหรับการส่งเสียงพูด ในระบบ PCM พบว่าการส่งเสียงพูดจะได้คุณภาพดีเมื่ออัตราการแซมปลิ้ง f_s 8000 Hz, N=8 ซึ่งจะได้บิทเรต 64 kb/s เปรียบเทียบกับ DM แล้วจะต้องใช้บิทเรตถึง 100 kb/s จะทำให้ได้คุณภาพเท่ากัน

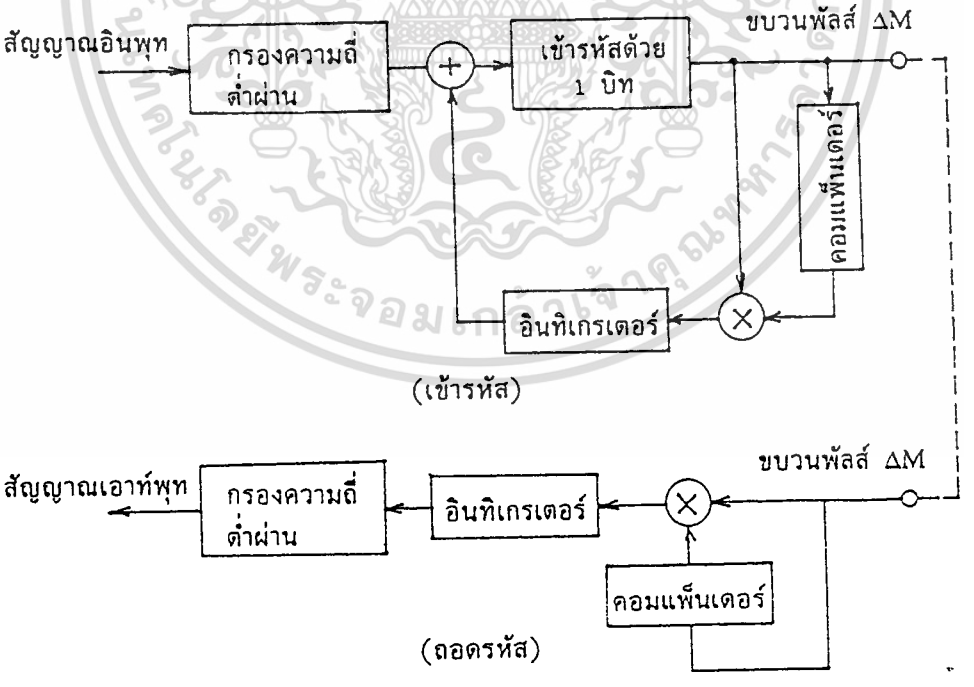
บทที่ 4
อแดปทีฟเดลต้ามอดูเลชั่น
Adaptive Delta Modulation

4.1 ลักษณะโดยทั่วไปของ ADM

สโลปโอเวอร์โหลดจะเกิดขึ้นเมื่อความชันของสัญญาณสูงมากๆ ปัญหานี้สามารถแก้ไขได้โดยการปรับขนาดของสเต็ป โดยจริงๆ แล้วขนาดของสเต็ปควรจะเล็กเมื่อการเปลี่ยนแปลงของสัญญาณเกิดอย่างช้าๆ และเพิ่มขนาดของสเต็ปเมื่อการเปลี่ยนแปลงของสัญญาณเกิดขึ้นอย่างรวดเร็ว เพื่อที่จะหลีกเลี่ยงสโลปโอเวอร์โหลดเมื่อสัญญาณมีการเปลี่ยนแปลงมาก

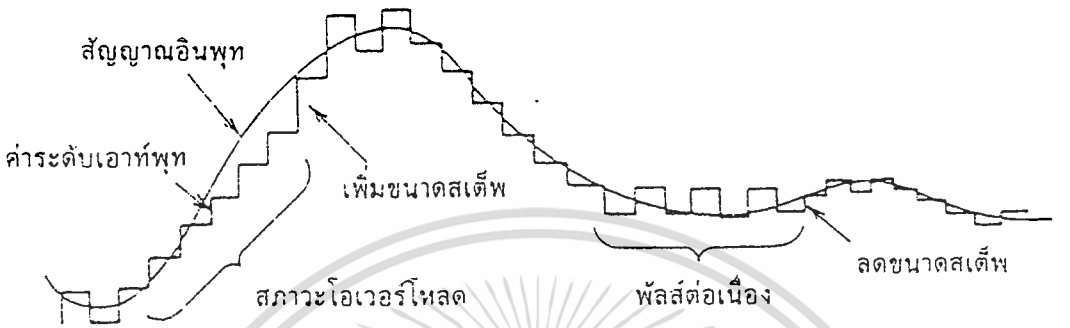
ในขณะที่ความถี่ในการแซมปลิ้งสูงขึ้นผลต่างระหว่างค่าแซมเปิ้ลข้างเคียงจะน้อยลงระบบการเข้ารหัสแบบ ADM จะพิจารณาจากจุดนี้คือใช้ความถี่ในการแซมปลิ้งให้สูงขึ้นและเข้ารหัสของผลต่างของสัญญาณเพื่อส่งออกไปด้วย 1 บิท

โครงสร้างของการเข้ารหัส/ถอดรหัสแบบ ADM ประกอบด้วยคอมแพเรเตอร์อินทิเกรเตอร์ (integrator) D/A คอนเวอร์เตอร์ และวงจรลอจิกที่จำเป็นบางวงจรสามารถสร้างได้ง่าย



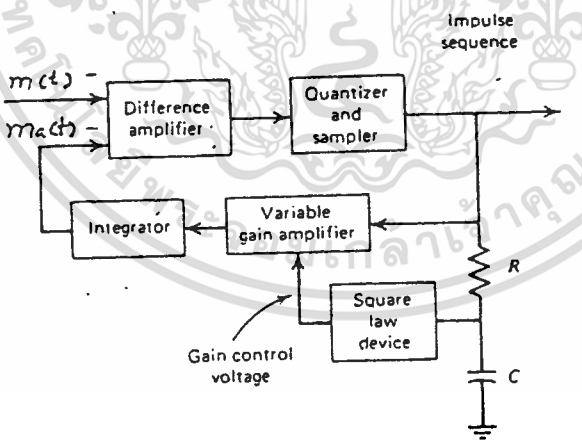
รูปที่ 4.1 โครงสร้างของระบบ ADM

กรณีที่เข้ารหัสโดยใช้เพียง 1 บิท จะกำหนดขนาดของระดับขั้น (step) จากขบวนพัลส์ที่เข้ามา ก่อนกรณีที่พัลส์ซึ่งเข้ามีขั้วเหมือนกันอย่างต่อเนื่องจะกำหนดว่าเกิดโอเวอร์โหลดในขณะนี้จะเพิ่มขนาดของระดับขั้นให้กว้างขึ้น ในทางตรงกันข้ามถ้าพัลส์มีขั้วสลับกันเกิดขั้นอย่างต่อเนื่องในการลดสัญญาณรบกวนที่เกิดจากการจัดระดับ (quantizing oise) จะต้องลดขนาดของระดับขั้นให้แคบลง



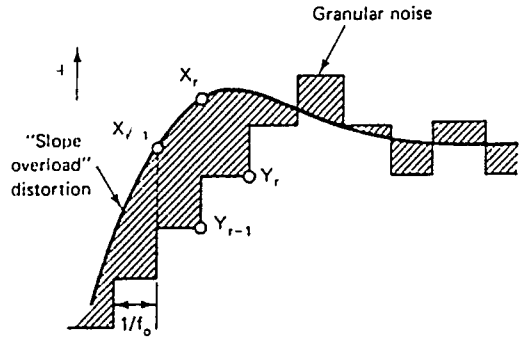
รูปที่ 4.2 การเปลี่ยนสเต็ปในการจัดระดับของ ADM

บล็อกไดอะแกรมแบบหนึ่งของระบบ ADM ที่สามารถปรับขนาดของสเต็ปตามลักษณะของสัญญาณแสดงในรูปที่ 4.1 ขนาดของสเต็ปจะเปลี่ยนแปลงได้ โดยการควบคุมเกน (gain) ของอินทิเกรเตอร์ ซึ่งเกนจะมีค่าต่ำเมื่อแรงดัน (voltage) เป็น 0 และเกนจะเพิ่มขึ้นเมื่อแรงดันควบคุมเพิ่มมากขึ้น

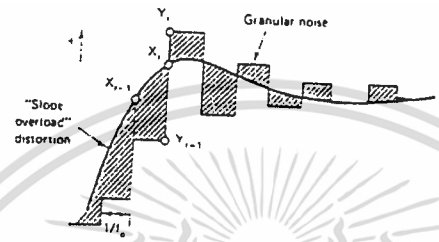


รูปที่ 4.3 block diagram แบบหนึ่งของระบบ ADM

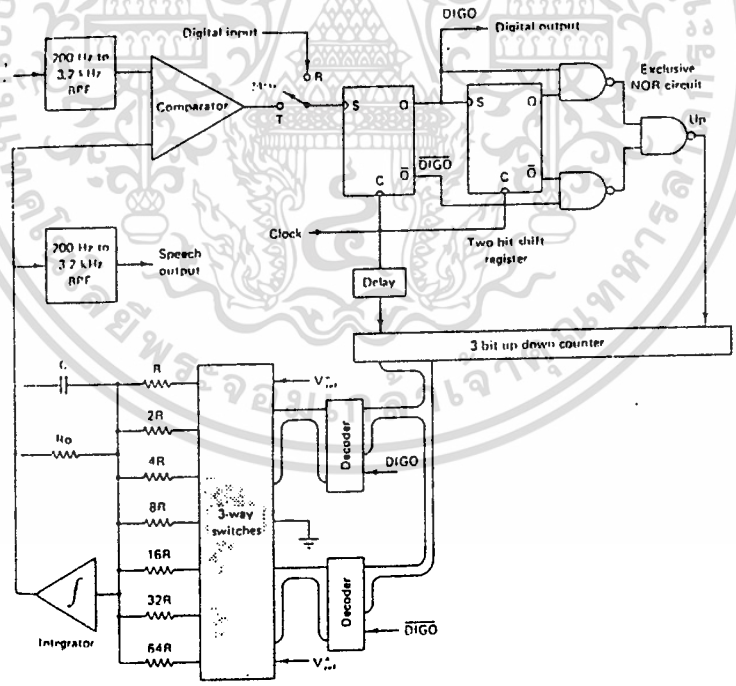
วงจรควบคุมเกนประกอบด้วย R, C และอุปกรณ์ square law เมื่อสัญญาณอินพุทคงที่หรือเปลี่ยนแปลงช้าๆ DM จะตามได้ทัน และเอ้าท์พุทของมอดูเลเตอร์จะเป็นพัลส์ บวกและลบสลับกันตลอดเวลา พัลส์เหล่านี้เมื่ออินทิเกรตโดย R C แล้วจะได้เอ้าท์พุทออกมาเฉลี่ยเป็น 0 V.gain control จะต่ำ ดังนั้นสเต็ปของ accumulator จะต่ำด้วย



รูปที่ 4.4.1 ความผิดพลาดในการจัดระดับของ linear DM (LDM)



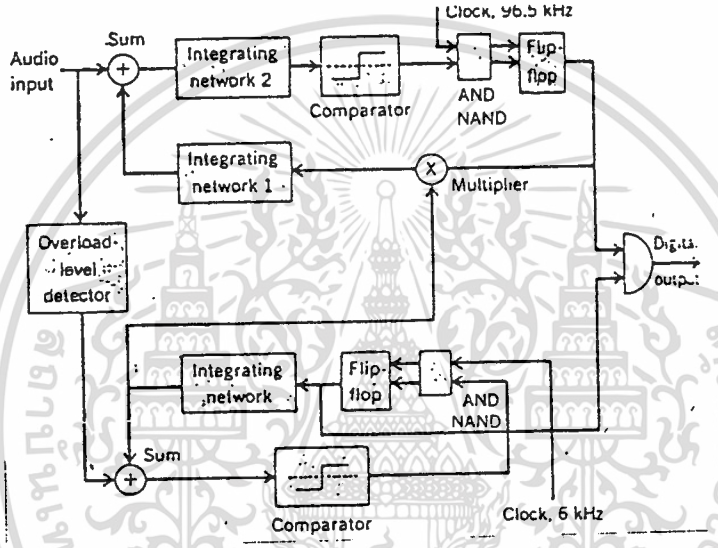
รูปที่ 4.4.2 ความผิดพลาดในการจัดระดับของ ADM



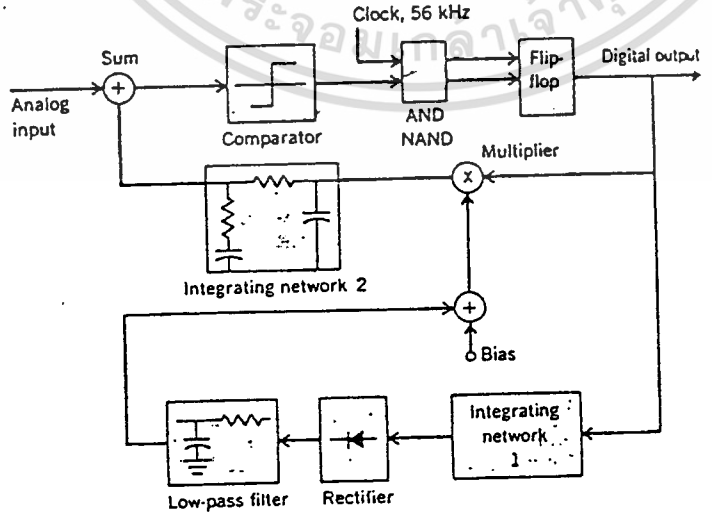
รูปที่ 4.5 ฮาร์ดแวร์ใน I.C. ที่ใช้เทคนิค ADM

ในกรณีที่จะเกิดสโลปโอเวอร์โหลด เอ้าท์พุทของมอดูเลเตอร์จะเป็นพัลส์บวกหรือลบตลอด ตอนนี้อินทิเกรเตอร์จะอินทิเกรตให้เอ้าท์พุทโวลเตจออกมา จะไปเพิ่มเกนของวงจรขยาย(amplifier) ทำให้มีขนาดของสเต็ปเพิ่มขึ้น ทำให้ไม่เกิดสโลปโอเวอร์โหลดหรือเกิดเพียงเล็กน้อย

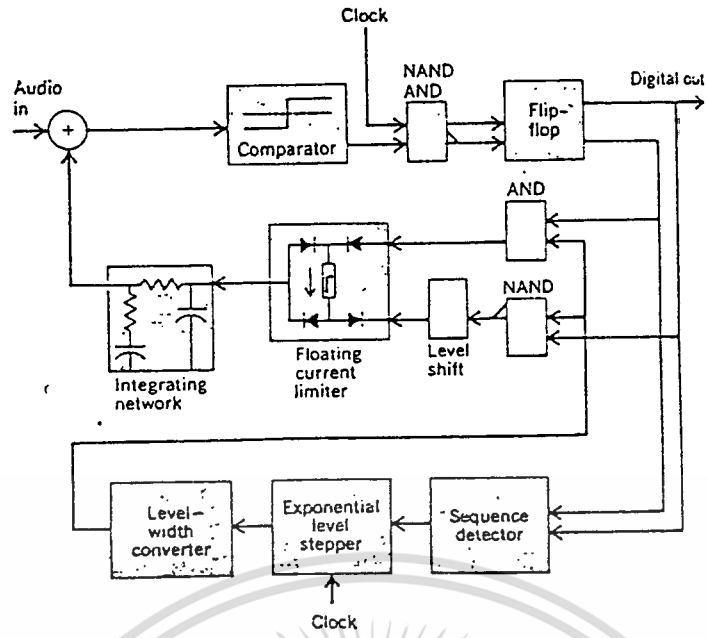
สำหรับดีมอดูเลเตอร์ในระบบ ADM จะมีวงจร adaptive gain เช่นเดียวกับทางด้านเอ็นโค็ดเดอร์ (encoder) ทำให้เอ้าท์พุทมีลักษณะเช่นเดียวกับทางด้านเอ็นโค็ดเดอร์ทุกประการ สำหรับบล็อกไดอะแกรมแบบอื่นๆ ของระบบ ADM ก็มีดังแสดงในรูปที่ 4.6, 4.7 และ 4.8



รูปที่ 4.6 Bell Telelphone Lab's companded delta coder



รูปที่ 4.7 Nippon Electric's companded delta coder



รูปที่ 4.8 IBM's companded delta coder

การเข้ารหัสแบบ ADM นี้สามารถทำได้หลายวิธี เช่น

- Continuous Variable Slope Delta modulation (CVSD)
- Constant Factor Delta modulation (CFDM)
- High Information Delta Modulation (HIDM)
- Hybrid Companding Delta modulation (HCDM)

ในที่นี้จะอธิบายเฉพาะ HCDM เท่านั้น

4.2 Hybrid Companding Delta Modulation (HCDM)

HCDM เป็นการแก้ไข Adaptive Delta Modulation (ADM) ระบบนี้ดีกว่าระบบ ADM แบบเดิม คือระบบ Continuous Variable Slope DM (CVSD) หรือ Constant Factor DM (CFDM) โดยเฉพาะอย่างยิ่งการเข้ารหัสเสียงพูด

4.2.1 ลักษณะโดยทั่วไปของ HCDM

ระบบการเข้ารหัสดิจิตอลสำหรับส่งเสียงพูดต่างๆ นั้น Delta Modulation (DM) เป็นวิธีการหนึ่งที่มีประสิทธิภาพมาก DM เป็นระบบการจัดระดับ (Quantization) ที่คาดการณ์ได้อย่างง่าย หรือกล่าวอีกอย่างก็คือ DM ก็คือ DPCM (Differantial PCM) บิทนั่นเอง แต่ระบบ DM นี้ง่ายและถูกกว่า PCM และ DPCM

ทั้งที่ DM มีข้อได้เปรียบหลายประการ แต่เหตุผลหนึ่งซึ่งทำให้ DM มิได้รับการยอมรับอย่างกว้างขวางในอดีตก็คือ พิสัยทางพลศาสตร์(dynamic)แคบ จากความยุ่งยากในการจัดระดับซึ่งทำให้ขนาดของระดับ (step size) คงที่ ทำให้เกิด noise จากการจัดระดับโดยธรรมชาติ 2 ชนิด คือ

- granular noise เกิดจากการจำกัด step size ของตัวเข้ารหัส
- slope-overload noise เกิดเมื่อความชันของสัญญาณมากกว่าที่ DM จะตามทัน

ปัญหาพิสัยทางพลศาสตร์แคบสามารถบรรเทาลงได้ด้วยการทำให้ระดับขั้นของการจัดระดับที่เหมาะสมซึ่งก็คือ ขนาดของระดับเปลี่ยนแปลงตามขนาดและความชันของสัญญาณอินพุท (input) มีการวิจัยโดยการใช้การคอมแพนดิ้ง (companding) ทางพยางค์ (syllabic)หรือ การคอมแพนดิ้งชั่วขณะ (instantaneous) ในการเปลี่ยนแปลงขนาดของระดับขั้นในการจัดระดับ(quantization)

ขั้นตอนในการเปลี่ยนแปลงให้เหมาะสมที่ใช้ในการคอมแพนดิ้งทางพยางค์จะปรับเปลี่ยนขนาดของขั้น (step) ในการจัดระดับค่อนข้างช้าที่อัตราของพยางค์ (syllabic rate) ตัวอย่างคือ CVSD

การคอมแพนดิ้งชั่วขณะนั้น ขนาดของขั้นในการจัดระดับจะเปลี่ยนแปลงอย่างทันที ณ ช่วงเวลาการสุ่มตัวอย่าง (sampling) ซึ่งขึ้นอยู่กับการเปลี่ยนแปลงของสัญญาณอินพุทตัวอย่างคือ CFDM และ High Information DM (HIDM)

ทั้งที่ระบบ ADM ที่ใช้การคอมแพนดิ้งทางพยางค์ หรือ แบบชั่วขณะ มีคุณสมบัติดีกว่าระบบ DM เชิงเส้นที่มิได้มีการดัดแปลง (nonadaptive linear delta modulation-LDM) แต่วิธีนี้ยังมีข้อบกพร่องบางอย่าง เช่นADM ไม่สามารถตามสัญญาญอินพุทที่มีการเปลี่ยนแปลงอย่างทันทีทันใดเช่นเดียวกับรูปคลื่นของเสียงพูด เนื่องจากขนาดของระดับขึ้นอยู่กับสัญญาณอินพุทหรือความชันเฉลี่ยต่อช่วงเวลาหนึ่ง

ในทางตรงกันข้าม ถ้าใช้การคอมแพนดิ้งชั่วขณะสำหรับการเปลี่ยนแปลงขนาดของการจัดระดับและมีการจำกัดขนาดพื้นฐานของแต่ละขั้น ระบบ ADM จะทำงานได้ดีสำหรับเสียงพูดเพียง 1 ส่วน (segment) แต่จะเกิด noise ที่ยอมรับไม่ได้จากการจัดระดับในส่วนอื่นๆ ของเสียงพูด เนื่องจากโดยทั่วไปจะมีความแตกต่างอย่างมากในพิสัยทางพลศาสตร์ของสัญญาณเสียง และขนาดของระดับขั้นของ ADM ไม่สามารถหาขีดจำกัดที่ได้ผลดีที่สุดสำหรับสัญญาณต่างๆ และการเปลี่ยนแปลงขนาดของแต่ละขั้นอย่างรวดเร็ว ทำให้เกิดลักษณะที่ไม่ดีในส่วนของเสียงพูดในภาวะคงที่

ในโครงงานนี้จะแสดงระบบ ADM ที่ได้รับการปรับปรุงแล้ว ซึ่งมีการใช้ทั้งวิธีการคอมแพนดิ้งแบบพยางค์และแบบชั่วขณะ ระบบนี้ก็คือ HCDM (Hybrid Compandingelta Modulation) พบว่าคุณลักษณะของHCDM มีข้อดีกว่าทั้งระบบ CVSD และ CFDM โดยไม่คำนึงถึงสภาวะของช่องสัญญาณ

4.2.2.ขั้นตอนของ HCDM และ การหาจุดที่ดีที่สุดของระบบ (system optimization)

4.2.2.1 ขั้นตอนของ HCDM

ตัวอย่างอินพุท (input sanple - s_n)จะถูกเปรียบเทียบกับค่าโดยประมาณ (estimated value) ของตัวมันซึ่งได้จากการเพิ่มหรือลดค่าโดยประมาณก่อนหน้าที่เวลาการสุ่มตัวอย่าง (sampling time) แต่ละช่วงด้วยขนาดของระดับขั้น 1 ระดับ โดยขึ้นอยู่กับเครื่องหมายของความแตกต่างระหว่างสัญญาณ และค่าโดยประมาณข่าวสารทางเครื่องหมาย (sign information) ขนาด 1 บิทต่อการสุ่มตัวอย่างแต่ละครั้งจะถูกส่งผ่านช่องสัญญาณไบนารี่ไปยังเครื่องรับ ความสามารถพิเศษของระบบ HCDM คือการใช้งานคอมแพนเดอร์ (companders) แบบพยางค์และแบบชั่วขณะพร้อมกัน ส่วนแรก ใช้เพื่อปรับปรุงขนาดของระดับขั้นพื้นฐานของอุปกรณ์จัดระดับ(quantizer) ขึ้นอยู่กับการเปลี่ยนแปลงของสัญญาณอินพุท และส่วนหลังใช้เพื่อการเปลี่ยนแปลงขนาดของระดับขั้นทุกๆการสุ่มตัวอย่าง

ขนาดของระดับขั้นพื้นฐาน A หาได้จากค่ารากที่ 2 ของกำลังสองเฉลี่ยของlope energy E ของสัญญาณเสียงที่ถูกคาดหมายโดยการผ่าน low-pass filter จะถูกเปลี่ยนแปลงครั้งหนึ่งทุกๆ M ตัวอย่าง ซึ่งคือ

$$A = \alpha E \qquad ...(4.1)$$

เมื่อ α คือ scale factor

$$E = \left[\sum_{i=1}^{M-1} (s_i - s_{i-1})^2 / M-1 \right]^{1/2} \qquad ...(4.2)$$

เมื่อ s_i คือ ตัวอย่างเสียงที่ถูกประมาณหรือคาดการณ์เป็นลำดับที่ i

ด้วยขนาดของระดับขั้นพื้นฐาน A การคอมแพนดิ้งชั่วขณะถูกกระทำโดยกฎทางตรรก(logic rule) ขึ้นอยู่กับเอ้าท์พุท 3 บิทที่ตามกันมาดังแสดงในตารางที่ 4.1 output bit (b_n) ถูกสร้างในตัวเข้ารหัส(coder) ของHCDM

$$b_n = \mathrm{sgn}(s_n - d_n) \qquad ...(4.3)$$

โดย

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

$$d_n = A\gamma_n$$

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

และ

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

เมื่อ β คือ ส่วนกลับของค่าคงตัวเวลารั่วไหลของวงรอบการประมาณค่า (leakage time constant of the prediction loop)

d_n คือ ขนาดของระดับขั้น (step size) ของการสุ่มตัวอย่างชั่วขณะลำดับที่ n

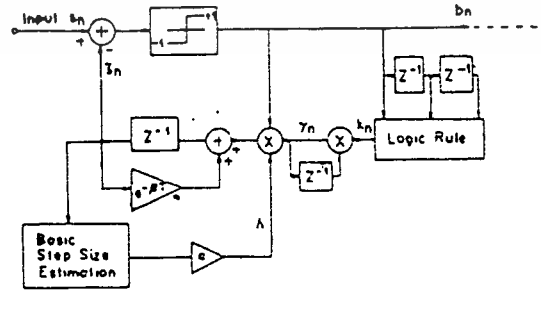
k_n คือ ตัวประกอบการคูณ ขึ้นอยู่กับค่าปัจจุบันและอีก 2 บิทก่อนหน้า

d_n (HCDM step size) ในทางปฏิบัติขึ้นอยู่กับค่า E (signal slope energy)ซึ่งเปลี่ยนแปลงอย่างช้าๆ ที่อัตราพยางค์ และตัวประกอบของระดับขั้น $ธ_n$เปลี่ยนแปลงอย่างทันทีทันใดทุกๆ การสุ่มตัวอย่าง

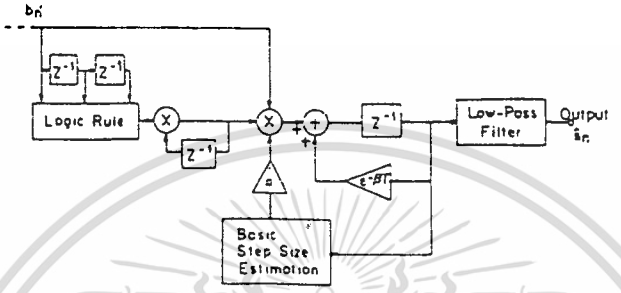
ตารางที่ 4.1 HCDM Companding Logic

b	b	b	Multiplication Factor (k)
+	+	+	1.50
-	-	-	1.50
-	-	+	1.00
+	+	-	1.00
-	+	+	0.66
+	-	-	0.66
-	+	-	0.66
+	-	+	0.66

จากสมการที่ให้มาข้างต้น สามารถนำมาเขียนเป็นบล็อคไดอะแกรมของตัวเข้ารหัส (encoder) และตัวถอดรหัส(decoder) ได้ดังนี้



รูปที่ 4.9 บล็อคไดอะแกรมของตัวเข้ารหัส (encoder) ของ HCDM



รูปที่ 4.10 บล็อคไดอะแกรมของตัวถอดรหัส (decoder) ของ HCDM

สามารถประมาณค่าการเปลี่ยนแปลงความชันของสัญญาณอินพุทด้วยแบบการไปข้างหน้าโดยตรง ในกรณีนี้ตัวเข้ารหัสจะซับซ้อนเพราะข่าวสารทางความชันจะถูกส่งกับสัญญาณเอ้าท์พุทแบบไบนารี่ของ HCDM หลังจากการเข้ารหัสและการมัลติเพล็กซ์ การได้เปรียบในการประมาณค่าแบบไปข้างหน้าจะน้อยลงเมื่อเปรียบเทียบกับแบบป้อนกลับ (feedback mode) แต่กระนั้น codec ของ DM เป็นแบบพื้นฐานซึ่งตามความชันของอินพุทมากกว่าขนาดของมัน การใช้งานของข่าวสารทางความชันจะถูกใช้งานมากกว่าทั้งๆ ที่ขั้นตอนของ HCDM จะถูกอธิบายในรูปแบบดิจิตอลแต่ก็สามารถกระทำในรูปแบบอนาลอก (analog) ได้ด้วย

4.2.3 การหาขีดจำกัดที่ดีที่สุดของ parameter และการพิจารณาคุณสมบัติของระบบ (Parameter Optimization and System Performance)

มาพิจารณาถึงค่าของตัวแปรที่สำคัญของ HCDM ด้วยการจำลองระบบโดยใช้คอมพิวเตอร์ เพื่อให้ได้คุณสมบัติของระบบที่ดีที่สุด เราใช้ค่าของ signal-to-quantization noise ratio (SQNR) เพื่อทำการวัดคุณสมบัติของระบบ

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

เมื่อ s_i คือ สัญญาณอินพุท

$\hat{s}_i$ คือ สัญญาณเสียงที่ถูกถอดรหัส(decode)

N คือ จำนวนสัญญาณเสียงอินพุททั้งหมดที่ถูกสุ่มตัวอย่าง

การหา SQNR พบว่าค่าของ scale factor (α) ของ long-term basic step size (A) มีบทบาทสำคัญในการควบคุม quantization noise

หมายเหตุ ในระบบนี้ว่าขนาดของระดับขั้นเบื้องต้น (basic step size) ถูกกำหนดผ่านวงรอบ (loop) ปิด ถ้า α ถูกกำหนดไว้ต่ำมากเอ้าท์พุทของ local decoder จะไม่สามารถติดตามสัญญาณอินพุทได้ดี ในกรณีทั่วไป step sizeขนาดเล็กนี้ทำให้เกิด slope-overload noise มาก ในทางตรงกันข้าม ถ้าตั้ง α ให้สูงมาก ทำให้ขนาดของระดับขั้นเบื้องต้นมีขนาดใหญ่ ทำให้เอ้าท์พุทของตัวถอดรหัสมีค่ามากเมื่อเปรียบเทียบกับสัญญาณอินพุท สิ่งที่ตามมาก็คือเอ้าท์พุทของตัวถอดรหัสจะไม่เสถียรภาพ

บทที่ 5
รูปแบบไฟล์ .WAV

5.1 SOUND BLASTER

Sound Blaster ซึ่งเปิดตัวในปี 1989 เป็นคู่แข่งขันตัวแรกของ AdLib card ซึ่งเป็น sound card ที่ได้รับความนิยมขณะนั้นแต่ Sound Blaster card มี 3 ลักษณะที่ sound card อื่นๆ ในขณะนั้นไม่มี นั่นคือ

1. SB card นี้มีลักษณะเป็น fully compatible กับ Game Blaster กล่าวคือ SB card สามารถใช้กับ Game Blaster software ทั้งหมดที่มีอยู่

2. SB card มีลักษณะที่ compatible อย่างสมบูรณ์กับ AdLib card

3. ลักษณะที่สำคัญที่สุดของ card นี้คือ integrated sampling capacity ด้วยช่องสัญญาณเสียง (audio)แบบดิจิตอล 1 ช่องสัญญาณ สำหรับการบันทึกเสียง และเล่นเสียงธรรมชาติ ดนตรี หรือเสียงพูด

5.1.1 ความสามารถพิเศษของ SB software

-Intelligent organ

SB. มีความสามารถหลากหลายและใช้ได้กับ MIDI keyboard หรือ แป้นพิมพ์ บนไมโครคอมพิวเตอร์ (PC) สามารถบันทึก (record) เก็บ (save) และเล่นกลับ (play back) เสียงดนตรี สามารถทำให้สร้างเสียงดนตรีจากเครื่องคอมพิวเตอร์ ทำหน้าที่เสมือนเป็น bass, โน้ตเสียงเรียงตามลำดับ (arpeggio) และเสียงดนตรีสังเคราะห์ (artificial melody)

-Talking Parrot

โปรแกรมสำหรับ SB card สามารถเลียนเสียงคนได้ โดยการพูดผ่านไมโครโฟน และสามารถนำไปใช้ประกอบกับสิ่งอื่นๆ ได้ เช่น นำไปประกอบกับตู้เย็นทำให้ตอนเปิดมีเสียงดนตรีหรือเสียงคนล้อเลียน

-Voice Editor

สามารถบันทึก, เก็บ และเล่นตัวอย่างเสียงจาก sample files ที่ถูกแยกเป็นส่วนเล็กๆ เพื่อที่จะทำงาน(edit) แยกจากกันได้ โดยอาจทำการปรับความดัง-ค่อย เติมเสียงก้อง (echo) หรือ fade in/out สัญญาณ การตัด, เคลื่อนย้าย และคัดลอกส่วนของเสียงทำได้ไม่ยาก

-SBTalker

SBTalker เป็น speech synthesizer ที่ประสพผลสำเร็จอย่างมาก จากการสามารถอ่านตัวหนังสือภาษาอังกฤษได้ด้วยการใช้ Multi Media Player (MMPlay) สามารถรวม impressive graphics, เสียงดนตรี

และเสียงในรูปของข้อมูลดิจิตอล เพื่อนำเสนอสิ่งต่างๆ ได้ และนำมาประกอบเป็นสื่อเพื่อความบันเทิงต่างๆ เช่นเกมส์คอมพิวเตอร์

Sound Blaster software มีการพัฒนาให้สามารถใช้กับ Windows ได้ซึ่งมีไอคอน (icon) ต่างๆดังนี้ WaveStudio, EnsembleRemote, EnsembleWave, Sound'LE , Creative Mixer, Creative Mosaic

เราจะสนใจเฉพาะไอคอนที่ใช้งานในโครงงานนี้คือ WaveStudio ซึ่งมีความสามารถหลายๆ อย่าง เช่นการเล่น (PLAY), หยุด (STOP), หยุดชั่วขณะ (PAUSE),บันทึก (RECORD) และแก้ไข (EDIT) wave file โดยจะแสดงให้เห็นลักษณะของคลื่นเสียงบนจอภาพด้วย

5.1.2 รูปแบบของการบันทึกคลื่นเสียง

- เลือก Mono สำหรับ 1 ช่องสัญญาณเสียงซึ่งเหมาะกับการบันทึกเพื่อแสดงเสียงทั่วไป และ Stereo สำหรับ 2 ช่องสัญญาณเสียง ซึ่งเหมาะกับการบันทึกเสียงดนตรี

- เลือก 11025 Hz สำหรับเสียง (voice), 22050 Hz สำหรับเทปบันทึกเสียงและ 44100 Hz สำหรับการบันทึกลงบนแผ่น CD

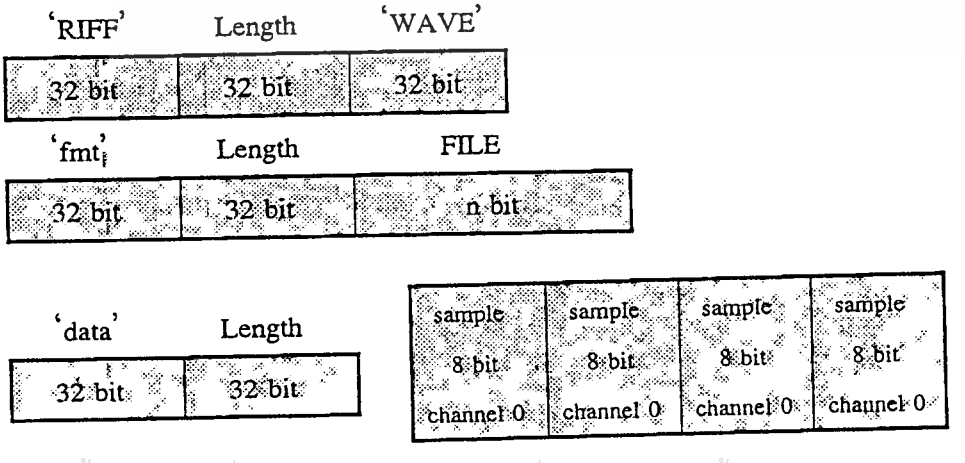
- เลือก 8 bits สำหรับเทปบันทึกเสียง และ 16 bits สำหรับการบันทึกลงบนแผ่น CD

5.2 รูปแบบของไฟล์ .WAV (WAVE file format)

รูปแบบของไฟล์ .WAV มีทั้งหมด 4 ประเภท คือ

1. ไฟล์ 8-bit mono

ตัวอย่างแบบ 8-bit mono นี้ ไบท์ (bytes) ทั้งหมดถูกเก็บโดยเรียงตามลำดับโดยช่องสัญญาณที่ 0 (channel 0) ถูกนำมาใช้สำหรับตัวอย่างแบบ mono

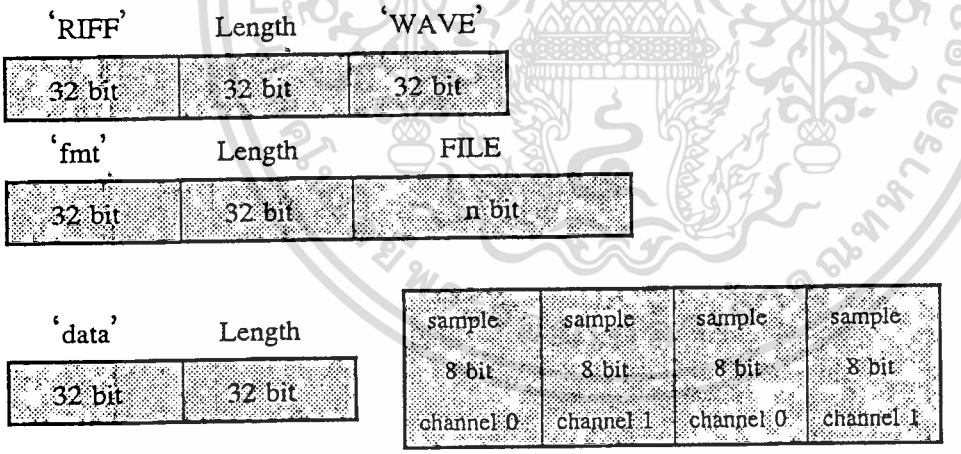


```
Cluster 1,305, Sector 21,304
00000000: 52 49 46 46 58 81 00 00  - 57 41 56 45 66 6D 74 20   RIFFX...WAVEfmt
00000010: 10 00 00 00 01 00 01 00  - 22 56 00 00 22 56 00 00   ........"V.."V..
00000020: 01 00 08 00 64 61 74 61  - 33 81 00 00 80 80 80 80   ....data3.......
00000030: 80 80 80 80 80 80 80 80  - 80 80 80 80 80 80 80 80   ................
```

รูปที่ 5.1แสดงโครงสร้างของไฟล์ 8-bit mono และตัวอย่างไฟล์

2. ไฟล์ 8-bit stereo

ตัวอย่างแบบ stereo นี้ ช่องสัญญาณที่ 0 คือช่องสัญญาณทางด้านซ้าย และช่องสัญญาณที่ 1 คือช่องสัญญาณทางด้านขวา ข้อมูลจะเก็บอยู่ตามรูปแบบนี้คือ 8 bits สำหรับช่องสัญญาณที่ 0 , 8 bits สำหรับช่องสัญญาณที่ 1,8 bits สำหรับช่องสัญญาณที่ 0,8 bits สำหรับช่องสัญญาณที่ 1,... ซึ่งหมายความว่าตัวอย่างหนึ่งประกอบด้วย 2 ไบท์ ไบท์แรกสำหรับช่องสัญญาณทางซ้าย และอีกไบท์สำหรับช่องสัญญาณทางขวา

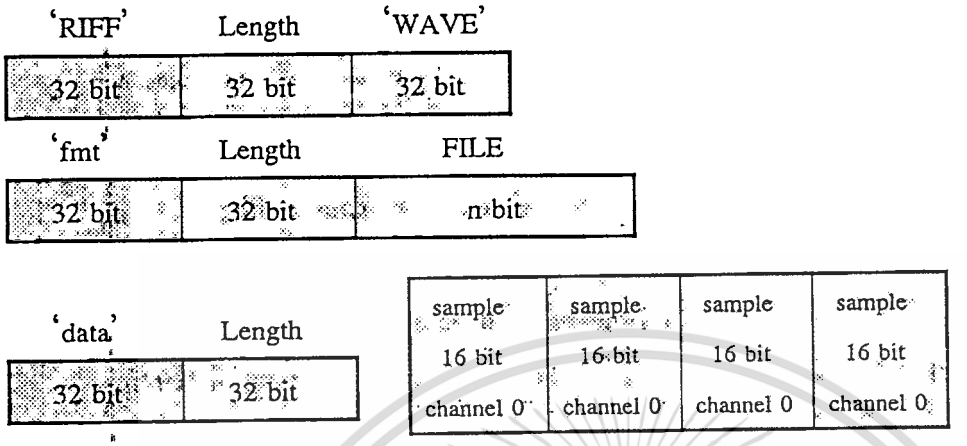



```
Cluster 1,331, Sector 21,720
00000000: 52 49 46 46 68 AC 00 00  - 57 41 56 45 66 6D 74 20   RIFFh...WAVEfmt
00000010: 10 00 00 00 01 00 02 00  - 22 56 00 00 44 AC 00 00   ........"V..D...
00000020: 02 00 08 00 64 61 74 61  - 44 AC 00 00 80 80 80 80   ....dataD.......
00000030: 80 80 80 80 80 80 80 80  - 80 80 80 80 80 80 80 80   ................
```

รูปที่ 5.2 แสดงโครงสร้างของไฟล์ 8-bit stereo และตัวอย่างไฟล์

3. ไฟล์ 16-bit mono

การแสดงตัวอย่างแบบ 16-bit mono ในหน่วยความจำนี้ 2 ไบท์ถูกใช้สำหรับการบันทึกแต่ละตัวอย่างการเรียงลำดับของไบท์ข้อมูลเหมือนกับแบบ 8-bit mono

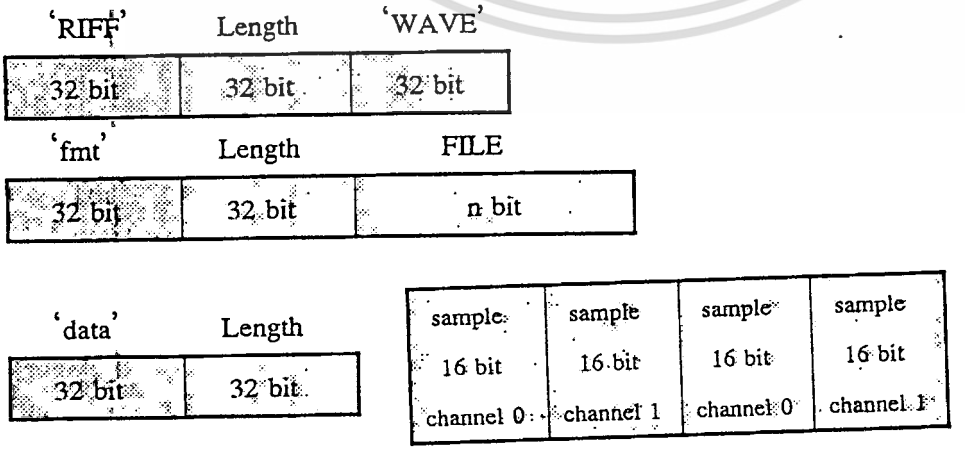



```
Cluster 1,444, Sector 23,528
00000000: 52 49 46 46 68 AC 00 00  - 57 41 56 45 66 6D 74 20   RIFFh...WAVEfmt
00000010: 10 00 00 00 01 00 01 00  - 22 56 00 00 44 AC 00 00   ........"V..D...
00000020: 02 00 10 00 64 61 74 61  - 44 AC 00 00 A1 00 A2 00   ....dataD.......
00000030: 9C 00 9A 00 9D 00 9C 00  - 9B 00 9D 00 A0 00 A3 00   ................
```

รูปที่ 5.3 แสดงโครงสร้างของไฟล์ 16-bit mono และตัวอย่างไฟล์

4. ไฟล์ 16-bit stereo

ตัวอย่างแบบ 16-bit stereo ต้องการจำนวนหน่วยความจำจำนวนมากสำหรับแต่ละตัวอย่าง ซึ่งต้องการถึง 4 ไบท์ (2 ไบท์สำหรับช่องสัญญาณทางซ้าย และ อีก 2 ไบท์สำหรับช่องสัญญาณทางขวา) ลำดับของการเรียงไบท์ข้อมูล คือ 1 ไบท์ต่ำ (low byte) และ 1 ไบท์สูง (high byte) สำหรับแต่ละช่องสัญญาณ



บทที่ 6
การทำงานของโปรแกรม

จากบทที่ผ่านๆ มา เราได้อธิบายถึงหลักการต่างๆ ของวิธีการมอดูเลชั่นแบบเดลต้ามอดูเลชั่นในบทนี้เราจะได้อธิบายถึงโครงงานชิ้นนี้

โครงงานชิ้นนี้เป็นการจำลองการมอดูเลชั่นและการดีมอดูเลชั่นแบบเดลต้ามอดูเลชั่นซึ่งมีทั้งแบบลิเนียร์ (linear delta modulation) และแบบปรับค่าได้ (adaptive delta modulation) โดยการเขียนโปรแกรมจากภาษาซี มีการรับข้อมูลจากไฟล์ .wav ซึ่งได้จากซาวด์บลาสเตอร์ นำมาผ่านกระบวนการมอดูเลชั่นแล้วให้เอ้าท์พุทไฟล์ออกมา ซึ่งไฟล์เอ้าท์พุทนี้จะมีขนาดของไฟล์ลดลงจากไฟล์ต้นฉบับประมาณ 8 หรือ 16 เท่า (ขึ้นกับว่าไฟล์ต้นฉบับมีการเก็บข้อมูลแบบ 8 บิทต่อ 1 แซมปลิ้ง หรือเก็บข้อมูลแบบ 16 บิทต่อ 1 แซมปลิ้ง) ซึ่งประโยชน์ที่ได้ก็คือสามารถจะประหยัดเนื้อที่ของฮาร์ดดิสต์ที่ใช้เก็บข้อมูลเป็นอันมาก และเมื่อต้องการได้ไฟล์เดิมกลับคืนมาก็ใช้โปรแกรมดีมอดูเลชั่นเป็นตัวแปลงไฟล์ข้อมูลกลับคืน ซึ่งไฟล์ที่ได้กลับคืนมานี้จะมีความผิดเพี้ยนไปจากไฟล์ต้นฉบับอยู่มากหรือน้อยนั้น ขึ้นอยู่กับค่าของสเต็ปไซด์ และ วิธีการมอดูเลชั่นที่เราเลือกใช้

โปรแกรมในโครงงานนี้ ประกอบด้วยไฟล์ต่าง ๆ ดังนี้

1.ไฟล์ DELTA.C	เป็นโปรแกรมหลักเพื่อเลือกการทำงานอื่นๆ ต่อไป
2.ไฟล์ STEPSIZE.C	เป็นโปรแกรมที่ใช้หาค่าสเต็ปไซด์เพื่อให้ได้ค่า SQNR สูงสุด
3.ไฟล์ LN_DM.C	เป็นโปรแกรมการมอดูเลชั่นแบบลิเนียร์
4.ไฟล์ LN_DDM.C	เป็นโปรแกรมการดีมอดูเลชั่นแบบลิเนียร์
5.ไฟล์ ADM_DM.C	เป็นโปรแกรมการมอดูเลชั่นแบบปรับค่าได้
6.ไฟล์ ADM_DDM.C	เป็นโปรแกรมการดีมอดูเลชั่นแบบปรับค่าได้
7.ไฟล์ EGAVGA.BGI	เป็นไฟล์ที่ใช้เพื่อการแสดงผลแบบกราฟฟิค

ต่อไปนี้จะได้อธิบายถึงการทำงานของโปรแกรมโครงงานนี้

6.1 การทำงานของโปรแกรม DELTA.C

เริ่มต้นโปรแกรมจะแสดงเมนูให้เลือกรายการต่างๆ ที่ต้องการ แล้วจะรอรับค่าที่ผู้ใช้เลือกเมื่อผู้ใช้เลือกรายการที่ต้องการแล้ว โปรแกรมนี้ก็จะทำตามโดยจะมีการเรียกโปรแกรมย่อยอื่นอีกทีหนึ่ง และเมื่อทำรายการนั้นเสร็จเรียบร้อยแล้วก็จะมีการคืนการทำงานกลับมาที่โปรแกรมนี้อีกครั้งเพื่อรอรับการทำงานใหม่หรือเพื่อออกจากโปรแกรม ผังการทำงานของโปรแกรม DELTA.C นี้แสดงดังรูปที่ 6.1

6.2 การทำงานของโปรแกรมคำนวณค่า stepsize ที่ดีที่สุด

1. เริ่มจากการรับชื่อของไฟล์ที่ต้องการคำนวณ ทำการเปิดไฟล์นั้น

2. รับการป้อนข้อมูลชนิดของการคอมแพนดิ้ง (companding) ที่เลือกใช้ในโปรแกรมนี้ มี 2 เทคนิค คือ

- LINEAR DELTA MODULATION (LDM)
- ADAPTIVE DELTA MODULATION (ADM)

3. ตรวจสอบชนิดของไฟล์ ว่าเป็นไฟล์ชนิดไหน ซึ่งชนิดของไฟล์มีดังนี้

- MONO 8 bit
- MONO 16 bit
- STEREO 8 bit
- STEREO 16 bit

4. รับจำนวน sample ที่ต้องการคำนวน

5. คำนวนค่า root-mean-square (rms) slope energy (E) จากสูตร

$$E = \left[\sum_{i=1}^{M-1} (s_i - s_{i-1})^2 / M-1 \right]^{1/2} \quad \ldots(6.1)$$

เมื่อ s_i คือ input sample ลำดับที่ i

M คือ จำนวน sample

จากสูตรอาจกล่าวได้ว่า E คือ ค่าเฉลี่ยของความแตกต่างของสัญญาณทั้งหมดนั่นเอง

6. คำนวนหาค่า Basic step-size (A) จาก

$$A = \propto E \quad \ldots(6.2)$$

เมื่อ $\propto$ คือ Scale factor

ในโปรแกรมจะทำการวนลูปเพื่อเปลี่ยนค่า$\propto$ไปเรื่อยๆ เพื่อหาค่า step size ที่ดีที่สุด ซึ่งมีค่า SQNR สูงที่สุด

7. ทำการประมวลผลตามชนิดของการคอมแพนดิ้งที่เลือกใช้ และชนิดของไฟล์นั้นๆ

เทคนิคการคอมแพนดิ้งที่ใช้แบ่งเป็น 2 ประเภทใหญ่ๆ คือ

(1) Linear Delta Modulation(LDM)

(2) Adaptive Delta Modulation(ADM)

LDM นั้น ค่า step-size ที่ใช้ในการคำนวนจะคงที่ตลอด ไม่มีการเปลี่ยนแปลง ส่วน ADM นั้น ค่า step-size จะมีการเปลี่ยนแปลงไปตามแนวโน้มของข้อมูลที่สังเกตจากบิทก่อนหน้า

ตารางที่ 6.1 แสดงการเปลี่ยนค่า K โดยดูแนวโน้มของข้อมูลตัวหน้า

b	b	b	Multiplication Factor (k)
+	+	+	1.50
-	-	-	1.50
-	-	+	1.00
+	+	-	1.00
-	+	+	0.66
+	-	-	0.66
-	+	-	0.66
+	-	+	0.66

เมื่อ kคือ multiplication factor ที่ใช้คูณกับ step-size ก่อนหน้า

8. กำหนดค่าสำหรับเปรียบเทียบเริ่มต้น (approx) ให้เท่ากับค่ากลาง (median) ของข้อมูล

9. การวนลูปเพื่อคำนวนค่า SQNR เมื่อมีการเปลี่ยนแปลงค่า scale factor จากสูตร

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

เมื่อ s_i คือ สัญญาณอินพุท

$\hat{s}_i$ คือ สัญญาณเสียงที่ถูกถอดรหัส(decode)

N คือ จำนวนสัญญาณเสียงอินพุททั้งหมดที่ถูกสุ่มตัวอย่าง

10. แสดงผลค่า SQNR เมื่อ scale factor มีการเปลี่ยนแปลง เลือกค่า step-size ที่ได้ SQNR สูงที่สุด นำไปใช้คำนวณในโปรแกรมที่ทำการคอมแพนดิ้งข้อมูล แล้วทำการบันทึกลงไฟล์

ผังการทำงานของโปรแกรม STEPSIZE.C นี้แสดงดังรูปที่ 6.2

6.3 การทำงานของโปรแกรม LN_DM.C

เริ่มต้นโปรแกรมจะรอรับชื่อไฟล์ที่ต้องการทำการมอดูเลชั่นแบบลิเนียร์ และค่าของสเต็ปไซด์ จากนั้นโปรแกรมจะทำการเปิดไฟล์ตามชื่อที่ได้รับมาแล้วทำการตรวจสอบดูว่าไฟล์นั้นเป็นไฟล์แบบใด เมื่อรู้

ชนิดของไฟล์แล้วก็จะทำการเรียกฟังก์ชั่นย่อยสำหรับการมอดูเลชั่นไฟล์แบบนั้นๆ มาทำการมอดูเลชั่นแบบลิเนียร์จนกระทั่งจบไฟล์ ก็จะจบการทำงานของโปรแกรมนี้ และมีการส่งการทำงานไปยังโปรแกรม DELTA.C อีกทีหนึ่ง

6.3.1 การทำงานของฟังก์ชั่นย่อยในการมอดูเลชั่นแบบลิเนียร์ของไฟล์ 8 บิทโมโน (ฟังก์ชั่น LDM_8MONO)

เนื่องจากการทำงานของฟังก์ชั่นย่อยเพื่อการมอดูเลชั่นไฟล์แบบต่างๆ นั้น มีการทำงานที่คล้ายคลึงกันมาก ดังนั้นจึงขออธิบายการทำงานของฟังก์ชั่นย่อยการมอดูเลชั่นไฟล์แบบ 8 บิทโมโนเพียงแบบเดียวเท่านั้น

การทำงานของฟังก์ชั่นนี้เริ่มต้นด้วยการกำหนดตัวแปรต่างๆ รวมทั้งค่าของสเต็ปไซด์จากนั้นจะทำการอ่านไฟล์อินพุท 44 ไบต์แรก (ทั้ง 44 ไบต์นี้จะเป็นข้อมูลสำคัญที่บอกให้รู้ว่าเป็นไฟล์แบบใดซึ่งสำคัญมาก) แล้วเก็บข้อมูลลงไฟล์เอ้าท์พุท จากนั้นจะอ่านค่าไบต์ต่อมามาทำการเปรียบเทียบกับค่าประมาณ (ค่าประมาณเริ่มต้นเป็น 128) ผลจากการเปรียบเทียบจะมี 3 แบบคือ

1.ค่าข้อมูลมากกว่าค่าประมาณ ก็จะทำการเขียนข้อมูลลงบิทเป็น 1 แล้วบวกค่าประมาณด้วยค่าของสเต็ปไซด์ (ตามค่าที่รับมาก่อนเริ่มการมอดูเลชั่น)

2.ค่าข้อมูลน้อยกว่าค่าประมาณ ก็จะทำการเขียนข้อมูลลงบิทเป็น 0 แล้วลบค่าประมาณด้วยค่าของสเต็ปไซด์

3.ค่าข้อมูลเท่ากับค่าประมาณ ก็จะทำการเขียนข้อมูลลงบิทเป็น 1 และ 0 สลับกันไปโดยมีการตรวจสอบตัวแปร equal_value ถ้าequal_value เป็น 1 ก็จะเขียนบิทเป็น 1 แล้วบวกค่าประมาณด้วยค่าของสเต็ปไซด์ ถ้าเป็น 2 ก็จะเขียนข้อมูลลงบิทเป็น 0 แล้วลบค่าประมาณด้วยค่าของสเต็ปไซด์ จากนั้นจะนำตัวแปร equal_value มาลบออกด้วย 2 เพื่อใช้ในการตรวจสอบครั้งใหม่ต่อไป

จากนั้นโปรแกรมก็จะทำการแสดงผลการเปรียบเทียบออกทางจอภาพ แล้วทำการเช็คดูว่าเขียนข้อมูลบิทครบ 8 บิทแล้วหรือไม่ ถ้ายังก็ทำการอ่านค่าไบต์ถัดมามาทำการเปรียบเทียบต่อ ถ้าครบ 8 บิทแล้วก็ทำการเขียนข้อมูลลงไฟล์เอ้าท์พุท 1 ไบต์ จากนั้นจะทำการตรวจสอบว่าอ่านข้อมูลมาจนจบไฟล์หรือยัง ถ้ายังก็อ่านค่าไบต์ถัดมามาเปรียบเทียบต่อ ถ้าจบไฟล์แล้วก็จะจบการทำงานของฟังก์ชั่นย่อยนี้

สำหรับส่วนแตกต่างของฟังก์ชั่น LDM_8MONO กับฟังก์ชั่นย่อยอื่นคือ กรณีของ ฟังก์ชั่นสำหรับไฟล์แบบสเตอริโอจะมีการแยกตัวแปรออกเป็น 2 ชุดสำหรับช่องซ้ายมือและช่องขวามือและเวลาเขียนข้อมูลบิทนั้นจะเขียนข้อมูลสลับกันทีละบิทสำหรับช่องซ้ายและช่องขวา (คือบิทที่ 8,6,4 และ 2 สำหรับช่องซ้าย และบิท 7,5,3 และ 1 สำหรับช่องขวา) ส่วนการมอดูเลชั่นก็ใช้วิธีการเดียวกัน ส่วนกรณีของฟังก์ชั่นสำหรับไฟล์แบบ 16 บิทนั้นก็ทำงานเหมือนกับการทำงานกับไฟล์ 8 บิท เพียงแต่การอ่านข้อมูลจะทำการอ่านมาทีละ 2 ไบต์เท่านั้นเอง และค่าประมาณเริ่มต้นมีค่าเป็น 0

สำหรับผังการทำงานของโปรแกรม LN_DM.C และฟังก์ชั่นย่อย LDM_8MONO นั้นแสดงดังในรูป 6.3 และ 6.4 ตามลำดับ

6.4 การทำงานของโปรแกรม LN_DDM.C

การทำงานของโปรแกรมนี้เริ่มจากโปรแกรมจะรับชื่อไฟล์และค่าของสเต็ปไซด์เข้ามาจากนั้นโปรแกรมจะทำการเปิดไฟล์ตามชื่อที่รับเข้ามาแล้วทำการตรวจสอบรูปแบบของไฟล์นั้นว่าเป็นแบบใดแล้วจึงทำการเรียกใช้ฟังก์ชั่นย่อยในการดีมอดูเลชั่นแบบลิเนียร์สำหรับไฟล์แบบนั้น เมื่อสิ้นสุดการทำงานของฟังก์ชั่นย่อยแล้วก็จะส่งการทำงานกลับไปที่โปรแกรม DELTA.C อีกครั้งหนึ่ง

6.4.1 การทำงานของฟังก์ชั่นย่อยในการดีมอดูเลชั่นแบบลิเนียร์ของไฟล์ 8 บิทโมโน(ฟังก์ชั่น LDDM_8MONO)

เริ่มต้นฟังก์ชั่นย่อยนี้จะทำการกำหนดค่าตัวแปรต่างๆ และค่าสเต็ปไซด์ตามที่ได้รับมาในตอนเริ่มโปรแกรม จากนั้นจะอ่านข้อมูลจากไฟล์อินพุท 44 ไบต์แรกมาแล้วเก็บข้อมูลนี้ลงไฟล์เอ้าท์พุท จากนั้นจะทำการอ่านไบต์ถัดมาแล้วทำการตรวจสอบข้อมูลทีละบิทจาก MSB ไปยังLSB ถ้าข้อมูลของบิทเป็น 1 ก็จะนำค่าของสเต็ปไซด์ไปบวกกับค่าประมาณ จากนั้นก็เก็บค่าประมาณลงไฟล์เอ้าท์พุท แต่ถ้าข้อมูลของบิทเป็น 0 ก็จะนำค่าของสเต็ปไซด์ไปลบออกจากค่าประมาณแล้วเก็บค่าประมาณลงไฟล์เอ้าท์พุท จากนั้นจะดูว่าตรวจสอบข้อมูลครบ 8บิทแล้วหรือยังถ้ายังก็จะทำการตรวจสอบบิทต่อไป แต่ถ้าครบแล้วก็ตรวจสอบต่อไปว่าจบไฟล์อินพุทแล้วหรือยังถ้ายังก็จะทำการอ่านข้อมูลไบต์ถัดมาทำการตรวจสอบต่อ แต่ถ้าจบไฟล์แล้วก็จะจบการทำงานของฟังก์ชั่นย่อยนี้

สำหรับกรณีของฟังก์ชั่นย่อยที่ใช้กับไฟล์แบบสเตอริโอ นั้นก็ใช้วิธีการทำงานเหมือนกันเพียงแต่แยกตัวแปรออกเป็น 2 ชุดสำหรับช่องซ้ายและช่องขวา และข้อมูลที่อ่านออกมาทีละบิทนั้นจะเป็นบิทสำหรับช่องซ้ายและบิทสำหรับช่องขวาสลับกัน (ข้อมูล 1 ไบต์จะเป็นของช่องซ้าย 4 บิทคือบิท 8,6,4และ 2 และของช่องขวา 4 บิทคือบิท 7,5,3 และ 1) ส่วนฟังก์ชั่นย่อยสำหรับไฟล์แบบ 16 บิทนั้นก็ทำงานเช่นเดียวกับการทำงานกับไฟล์ 8 บิท เพียงแต่การเขียนข้อมูลลงไฟล์เอ้าท์พุทนั้นจะทำการเก็บทีละ 2 ไบต์เท่านั้นเอง

ผังการทำงานของโปรแกรม LN_DDM.C และฟังก์ชั่นย่อย LDDM_8MONO นั้นแสดงดังในรูป 6.5 และ 6.6 ตามลำดับ

6.5 การทำงานของโปรแกรม ADT_DM.C

เริ่มต้นโปรแกรมจะรอรับชื่อไฟล์ที่ต้องการทำการมอดูเลชั่นแบบปรับค่า และค่าของสเต็ปไซด์ จากนั้นโปรแกรมจะทำการเปิดไฟล์ตามชื่อที่ได้รับมาแล้วทำการตรวจสอบดูว่าไฟล์นั้นเป็นไฟล์แบบใด เมื่อรู้ชนิดของไฟล์แล้วก็จะทำการเรียกฟังก์ชั่นย่อยสำหรับการมอดูเลชั่นไฟล์แบบนั้นๆ มาทำการมอดูเลชั่นแบบปรับค่าจนกระทั่งจบไฟล์ ก็จะจบการทำงานของโปรแกรมนี้ และมีการส่งการทำงานไปยังโปรแกรม DELTA.C อีกทีหนึ่ง

6.5.1 การทำงานของฟังก์ชั่นย่อยในการมอดูเลชั่นแบบปรับค่าของไฟล์ 8 บิทโมโน(ฟังก์ชั่น ADM_8MONO)

การทำงานของฟังก์ชั่นนี้เริ่มต้นด้วยการกำหนดตัวแปรต่างๆ รวมทั้งค่าของสเต็ปไซด์เริ่มต้น จากนั้นจะทำการอ่านไฟล์อินพุท 44 ไบต์แรกแล้วเก็บข้อมูลลงไฟล์เอ้าท์พุท จากนั้นจะอ่านค่าไบต์ต่อมามาทำการเปรียบเทียบกับค่าประมาณ (ค่าประมาณเริ่มต้นเป็น 128) ผลจากการเปรียบเทียบจะมี 3 แบบคือ

1.ค่าข้อมูลมากกว่าค่าประมาณ ก็จะทำการเขียนข้อมูลลงบิทเป็น 1 จากนั้นจะเรียกใช้ฟังก์ชั่นเพื่อการปรับค่าของสเต็ปไซด์ซึ่งค่าของสเต็ปไซด์จะมีการเปลี่ยนแปลงตามแนวโน้มของค่าที่ได้จากการเปรียบเทียบก่อนหน้านั้น 2 ครั้ง และการเปรียบครั้งปัจจุบันแล้วเทียบกับตารางการเปลี่ยนค่าสเต็ปไซด์ จากนั้นจะบวกค่าประมาณด้วยค่าของสเต็ปไซด์

2.ค่าข้อมูลน้อยกว่าค่าประมาณ ก็จะทำการเขียนข้อมูลลงบิทเป็น 0 เรียกใช้ฟังก์ชั่นเพื่อปรับค่าสเต็ปไซด์ แล้วลบค่าประมาณด้วยค่าของสเต็ปไซด์

3.ค่าข้อมูลเท่ากับค่าประมาณ ก็จะทำการเขียนข้อมูลลงบิทเป็น 1 และ 0 สลับกันไปโดยมีการตรวจสอบตัวแปร equal_value ถ้า equal_value เป็น 1 ก็จะเขียนบิทเป็น 1 เรียกใช้ฟังก์ชั่นเพื่อปรับค่าสเต็ปไซด์แล้วบวกค่าประมาณด้วยค่าของสเต็ปไซด์ ถ้าเป็น 2 ก็จะเขียนข้อมูลลงบิทเป็น 0 เรียกใช้ฟังก์ชั่นเพื่อปรับค่าสเต็ปไซด์แล้วลบค่าประมาณด้วยค่าของสเต็ปไซด์ จากนั้นจะนำตัวแปรequal_value มาลบออกด้วย 2 เพื่อใช้ในการตรวจสอบครั้งใหม่ต่อไป

จากนั้นโปรแกรมก็จะทำการแสดงผลการเปรียบเทียบออกทางจอภาพ แล้วทำการเช็คดูว่าเขียนข้อมูลบิทครบ 8 บิทแล้วหรือไม่ ถ้ายังก็ทำการอ่านค่าไบต์ถัดมามาทำการเปรียบเทียบต่อ ถ้าครบ 8 บิทแล้วก็ทำการเขียนข้อมูลลงไฟล์เอ้าท์พุท 1 ไบต์ จากนั้นจะทำการตรวจสอบว่าอ่านข้อมูลมาจนจบไฟล์หรือยัง ถ้ายังก็อ่านค่าไบต์ถัดมามาเปรียบเทียบต่อ ถ้าจบไฟล์แล้วก็จะจบการทำงานของฟังก์ชั่นย่อยนี้

สำหรับส่วนแตกต่างของฟังก์ชั่น ADM_8MONO กับฟังก์ชั่นย่อยอื่นคือ กรณีของฟังก์ชั่นสำหรับไฟล์แบบสเตอริโอจะมีการแยกตัวแปรออกเป็น 2 ชุดสำหรับช่องซ้ายมือและช่องขวามือและเวลาเขียน

ข้อมูลบิทนั้นจะเขียนข้อมูลสลับกันทีละบิทสำหรับช่องซ้ายและช่องขวา (คือบิทที่ 8,6,4 และ 2 สำหรับช่องซ้าย และบิท 7,5,3 และ 1 สำหรับช่องขวา) ส่วนการมอดูเลชั่นก็ใช้วิธีการเดียวกัน ส่วนกรณีของฟังก์ชั่นสำหรับไฟล์แบบ 16 บิทนั้นก็ทำงานเหมือนกับการทำงานกับไฟล์ 8 บิท เพียงแต่การอ่านข้อมูลจะทำการอ่านมาทีละ 2 ไบต์เท่านั้นเอง และค่าประมาณเริ่มต้นมีค่าเป็น 0

สำหรับผังการทำงานของโปรแกรม ADT_DM.C และฟังก์ชั่นย่อย ADM_8MONO นั้นแสดงดังในรูป 6.7 และ 6.8 ตามลำดับ

6.6 การทำงานของโปรแกรม ADT_DDM.C

การทำงานของโปรแกรมนี้เริ่มจากโปรแกรมจะรับชื่อไฟล์และค่าของสเต็ปไซด์เข้ามาจากนั้นโปรแกรมจะทำการเปิดไฟล์ตามชื่อที่รับเข้ามาแล้วทำการตรวจสอบรูปแบบของไฟล์นั้นว่าเป็นแบบใดแล้วจึงทำการเรียกใช้ฟังก์ชั่นย่อยในการดีมอดูเลชั่นแบบปรับค่าสำหรับไฟล์แบบนั้น เมื่อสิ้นสุดการทำงานของฟังก์ชั่นย่อยแล้วก็จะส่งการทำงานกลับไปที่โปรแกรม DELTA.C อีกครั้งหนึ่ง

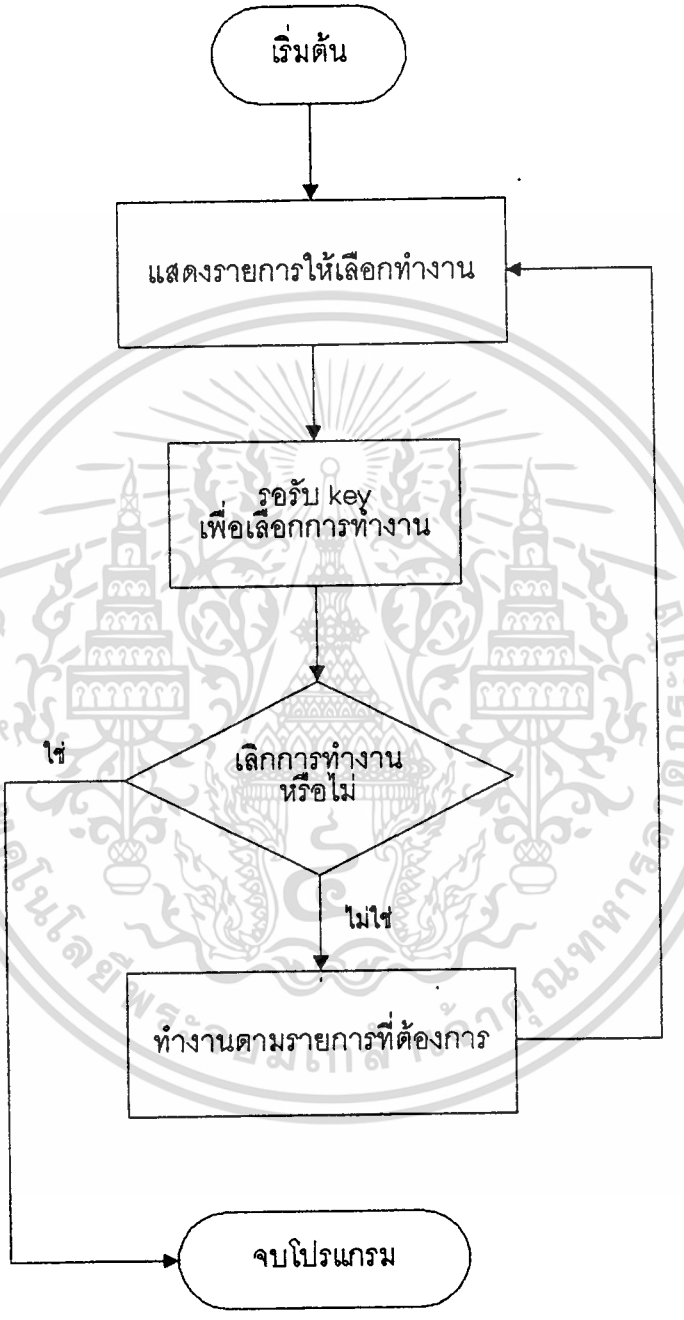
6.6.1 การทำงานของฟังก์ชั่นย่อยในการดีมอดูเลชั่นแบบปรับค่าของไฟล์ 8 บิทโมโน(ฟังก์ชั่น ADDM_8MONO)

เริ่มต้นฟังก์ชั่นย่อยนี้จะทำการกำหนดค่าตัวแปรต่างๆ และค่าสเต็ปไซด์ตามที่ได้รับมาในตอนเริ่มโปรแกรม จากนั้นจะอ่านข้อมูลจากไฟล์อินพุท 44 ไบต์แรกมาแล้วเก็บข้อมูลนี้ลงไฟล์เอ้าท์พุท จากนั้นจะทำการอ่านไบต์ถัดมาแล้วทำการตรวจสอบข้อมูลทีละบิทจาก MSB ไปยังLSB ถ้าข้อมูลของบิทเป็น 1 ก็จะเรียกใช้ฟังก์ชั่นเพื่อปรับค่าสเต็ปแล้วนำค่าของสเต็ปไซด์ไปบวกกับค่าประมาณ จากนั้นก็เก็บค่าประมาณลงไฟล์เอ้าท์พุท แต่ถ้าข้อมูลของบิทเป็น 0 ก็จะเรียกใช้ฟังก์ชั่นเพื่อปรับค่าสเต็ปแล้วนำค่าของสเต็ปไซด์ไปลบออกจากค่าประมาณแล้วเก็บค่าประมาณลงไฟล์เอ้าท์พุท จากนั้นจะดูว่าตรวจสอบข้อมูลครบ 8 บิทแล้วหรือยัง ถ้ายังก็จะทำการตรวจสอบบิทต่อไป แต่ถ้าครบแล้วก็ตรวจสอบต่อไปว่าจบไฟล์อินพุทแล้วหรือยัง ถ้ายังก็จะทำการอ่านข้อมูลไบต์ถัดมาทำการตรวจสอบต่อ แต่ถ้าจบไฟล์แล้วก็จะจบการทำงานของฟังก์ชั่นย่อยนี้

สำหรับกรณีของฟังก์ชั่นย่อยที่ใช้กับไฟล์แบบสเตอริโอ นั้นก็ใช้วิธีการทำงานเหมือนกันเพียงแต่แยกตัวแปรออกเป็น 2 ชุดสำหรับช่องซ้ายและช่องขวา และข้อมูลที่อ่านออกมาทีละบิทนั้นจะเป็นบิทสำหรับช่องซ้ายและบิทสำหรับช่องขวาสลับกัน (ข้อมูล 1 ไบต์จะเป็นของช่องซ้าย 4 บิทคือบิท 8,6,4และ 2 และของช่องขวา 4 บิทคือบิท 7,5,3 และ 1) ส่วนฟังก์ชั่นย่อยสำหรับไฟล์แบบ 16 บิทนั้นก็ทำงานเช่น

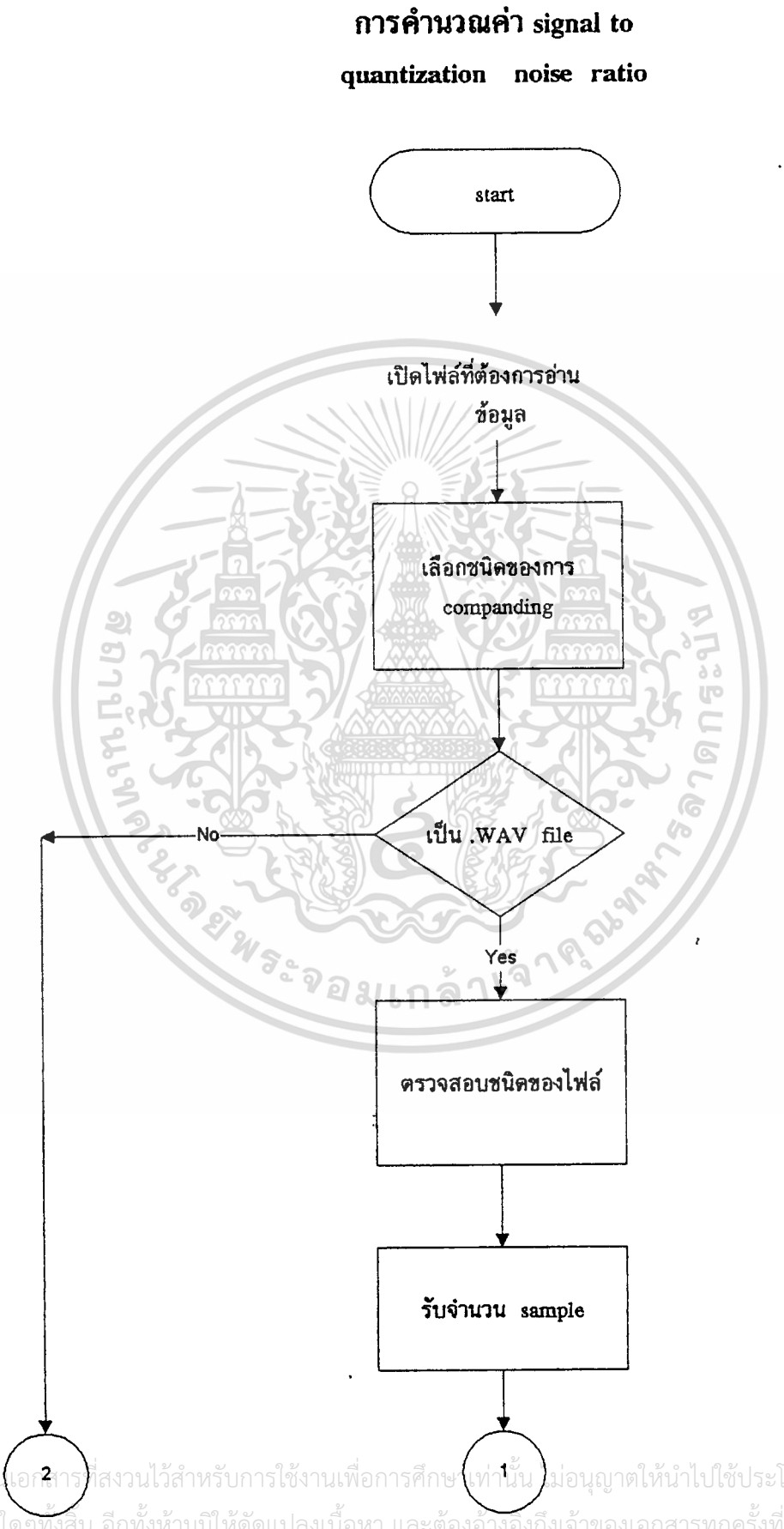
เดียวกับการทำงานกับไฟล์ 8 บิท เพียงแต่การเขียนข้อมูลลงไฟล์เอ้าท์พุทนั้นจะทำการเก็บทีละ 2 ไบต์เท่านั้นเอง

ผังการทำงานของโปรแกรม ADT_DDM.C และฟังก์ชั่นย่อย ADDM_8MONO นั้นแสดงดังในรูป 6.9 และ 6.10 ตามลำดับ

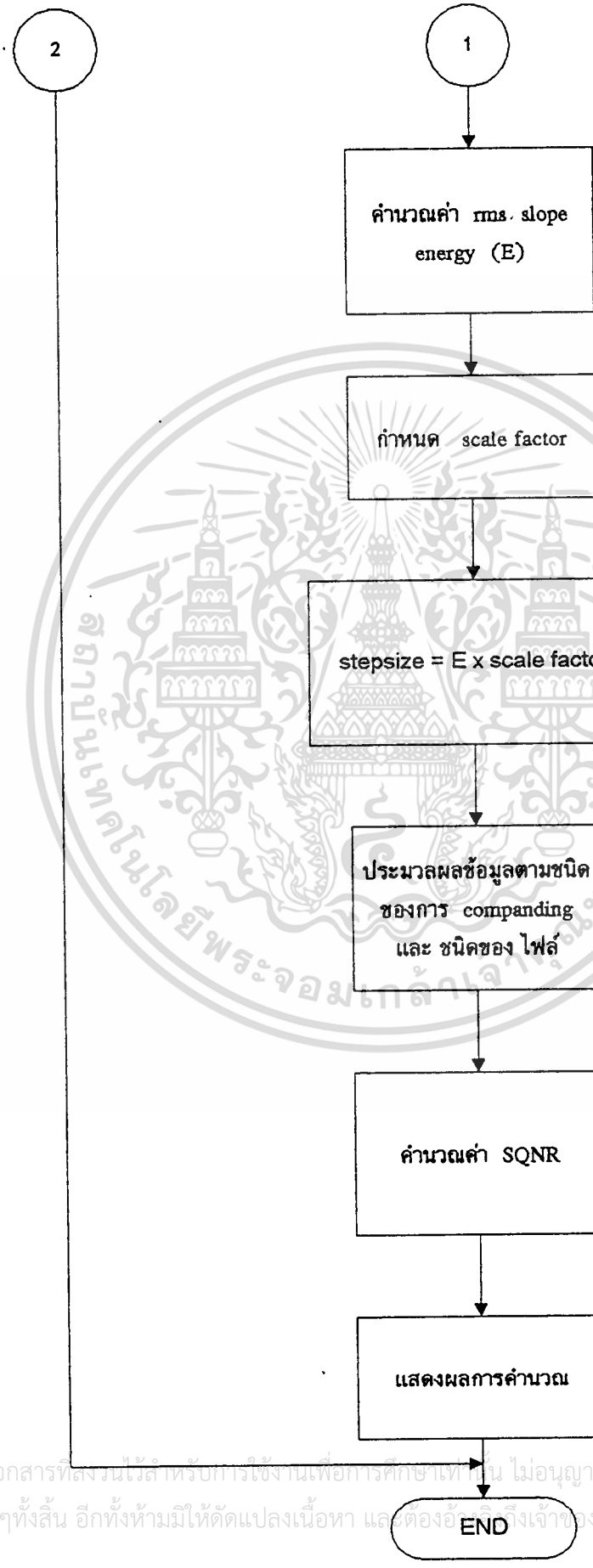


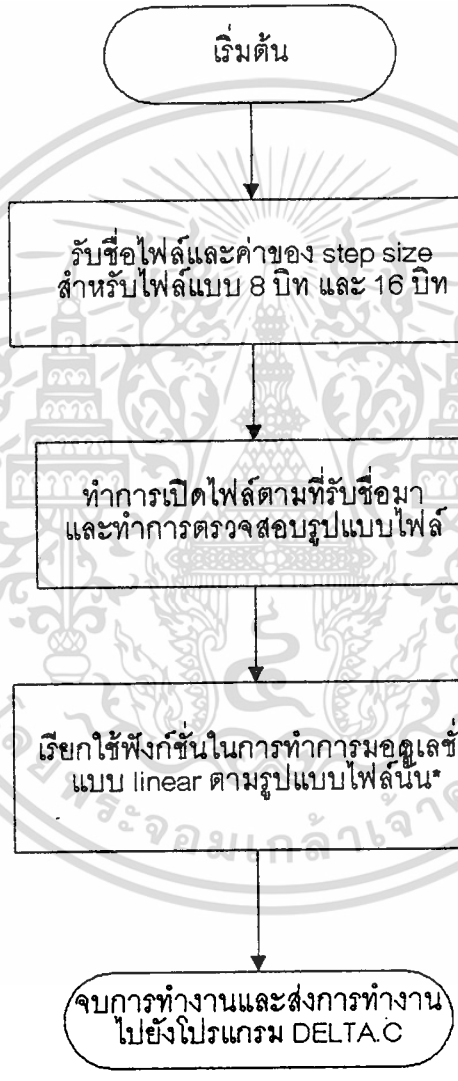
รูปที่ 6.1 แสดงผังการทำงานของโปรแกรม DELTA.C

รูปที่ 6.2 แสดงผังการทำงานของโปรแกรม STEPSIZE.C



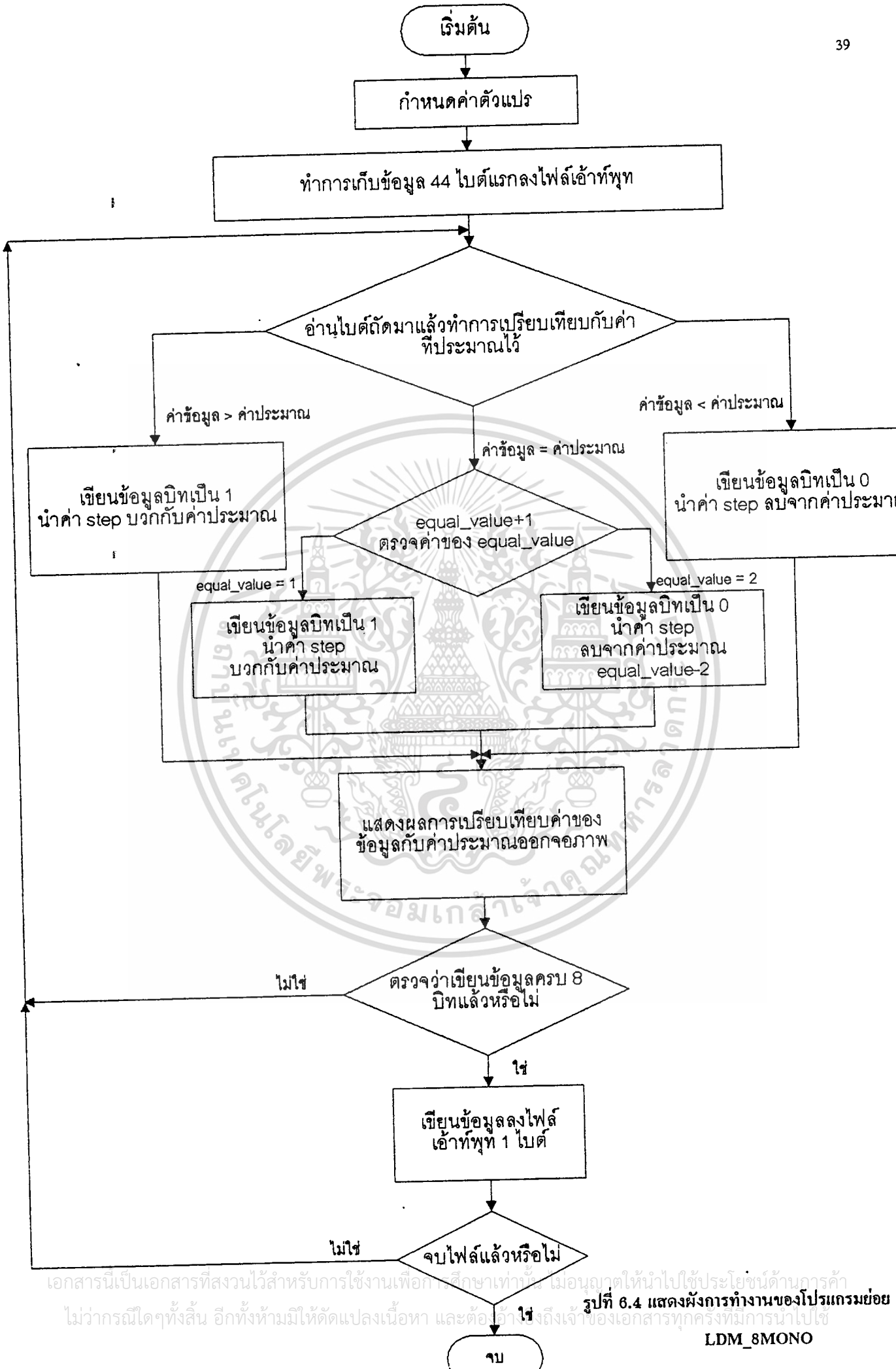
(ต่อ)



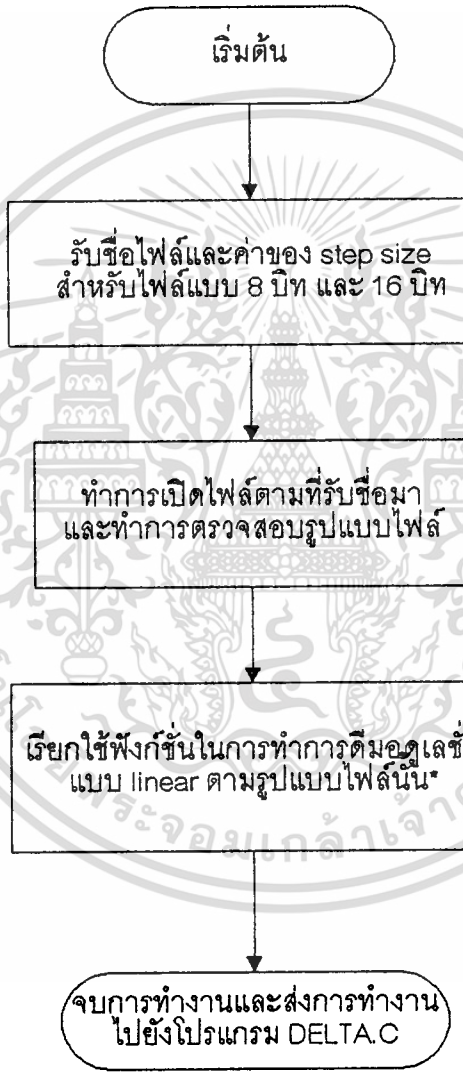


รูปที่ 6.3 แสดงผังการทำงานของโปรแกรม LN_DM.C

<u>หมายเหตุ</u> เนื่องจากการมอดูเลชั่นแบบลิเนียร์ของไฟล์รูปแบบต่างๆ นั้นมีการทำงานที่คล้ายคลึงกัน จึงขอแสดงผังการทำงานสำหรับไฟล์ 8 บิทโมโนเท่านั้นคือฟังก์ชั่นย่อย LDM_8MONO

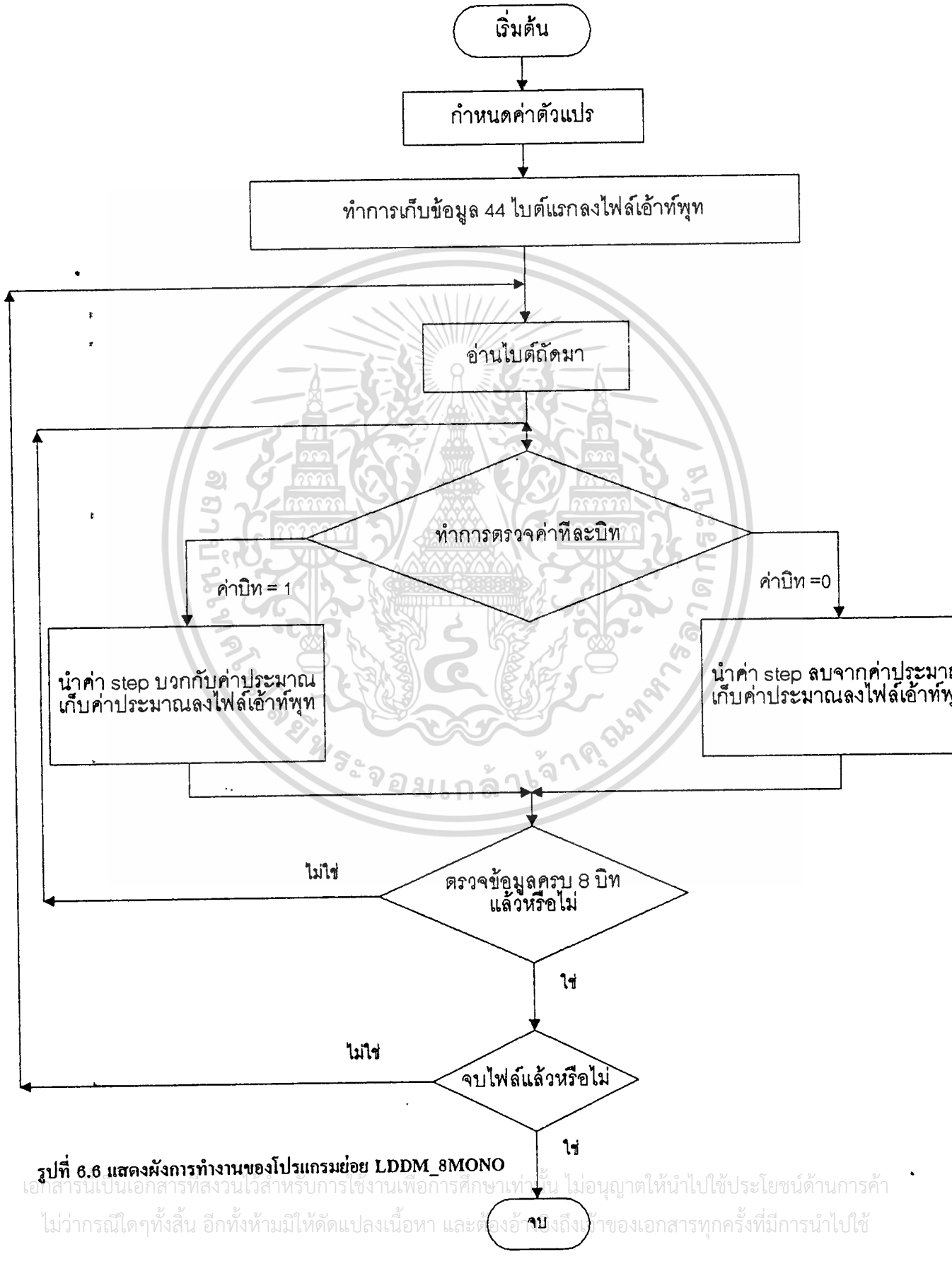


รูปที่ 6.4 แสดงผังการทำงานของโปรแกรมย่อย LDM_8MONO

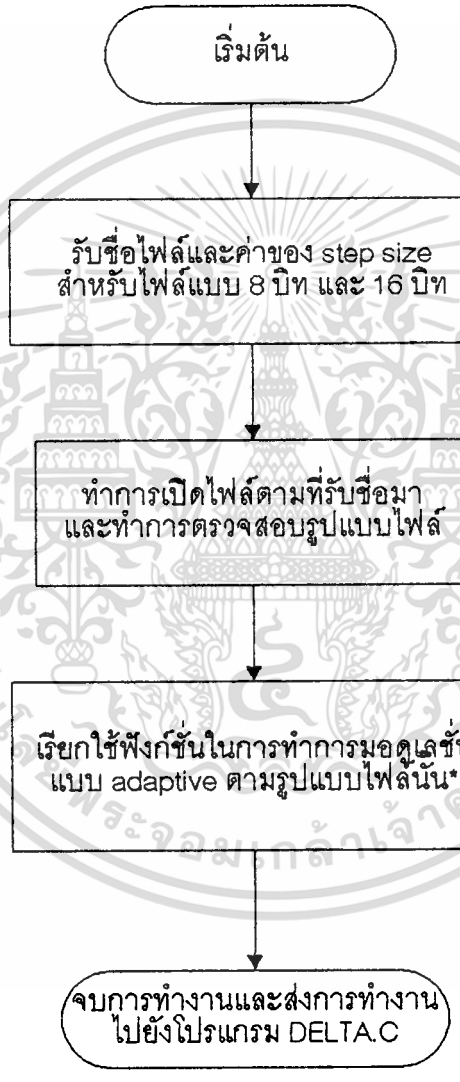


รูปที่ 6.5 แสดงผังการทำงานของโปรแกรม LN_DDM.C

<u>หมายเหตุ</u> เนื่องจากการดีมอดูเลชั่นแบบลิเนียร์ของไฟล์รูปแบบต่างๆ นั้นมีการทำงานที่คล้ายคลึงกันจึงขอแสดงผังการทำงานสำหรับไฟล์ 8 บิทโมโนเท่านั้นคือฟังก์ชั่นย่อย LDDM_8MONO

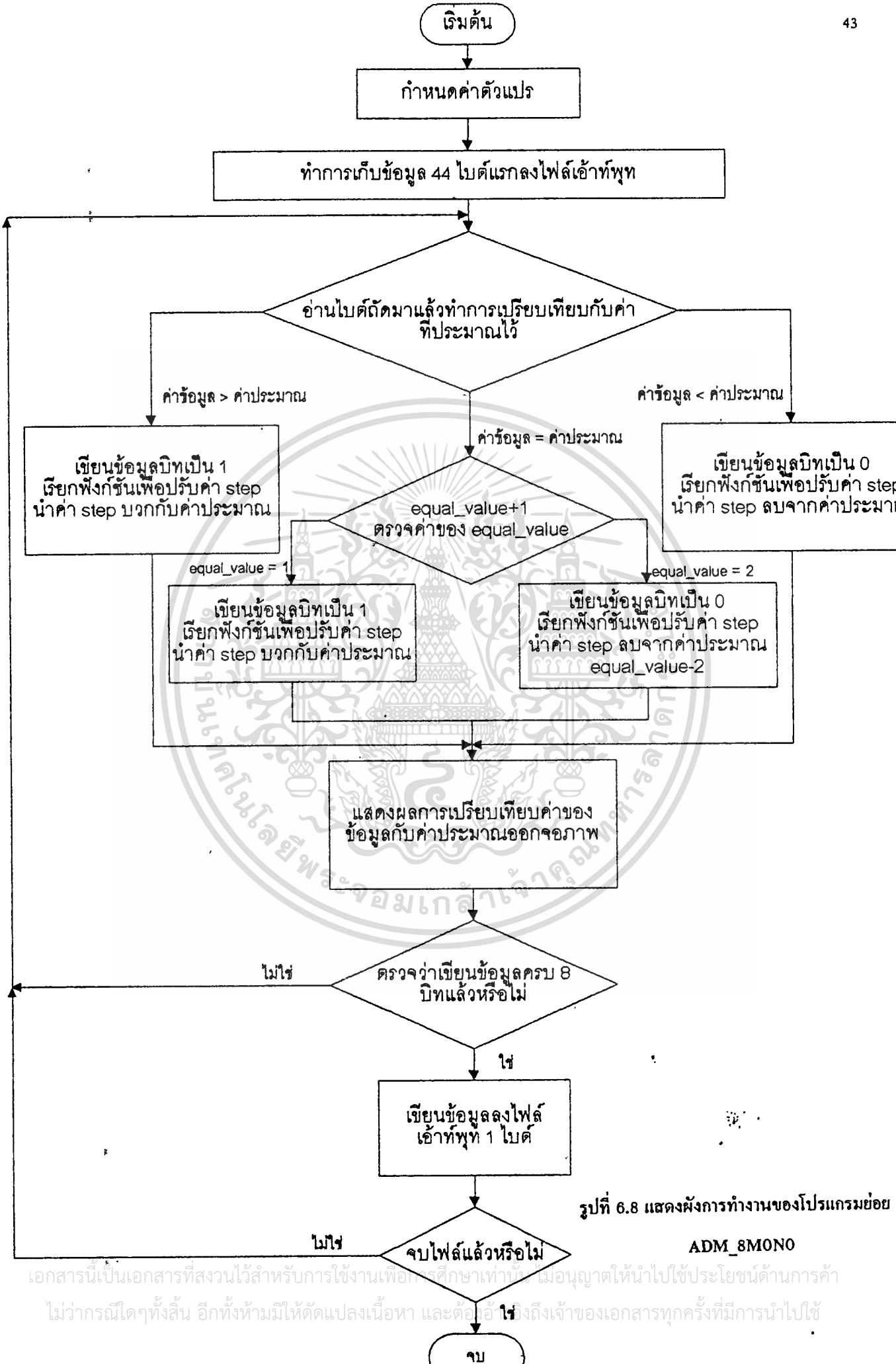


รูปที่ 6.6 แสดงผังการทำงานของโปรแกรมย่อย LDDM_8MONO

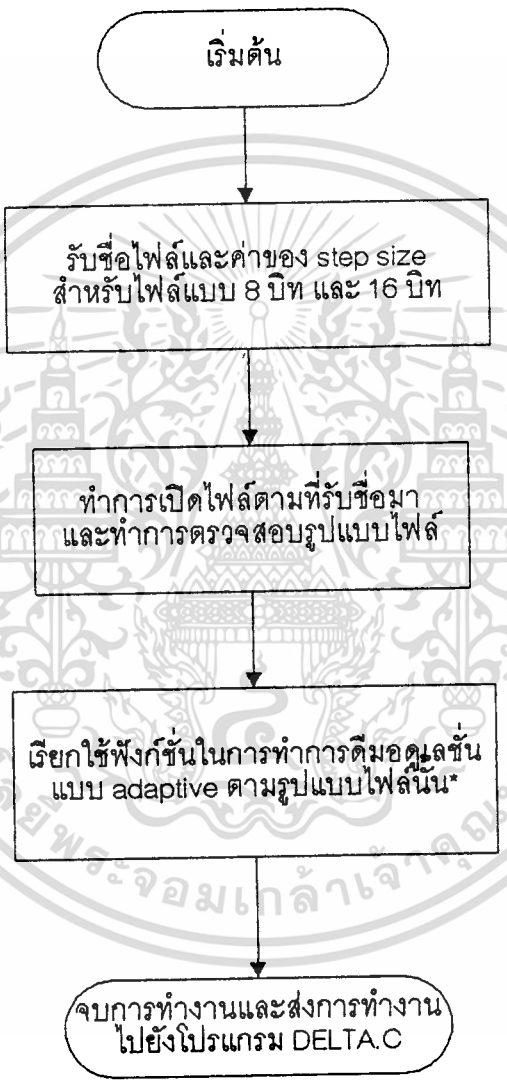


รูปที่ 6.7 แสดงผังการทำงานของโปรแกรม ADT_DM.C

<u>หมายเหตุ</u> เนื่องจากการมอดูเลชั่นแบบปรับค่าได้ของไฟล์รูปแบบต่างๆ นั้นมีการทำงานที่คล้ายคลึงกัน จึงขอแสดงผังการทำงานสำหรับไฟล์ 8 บิทโมโนเท่านั้นคือฟังก์ชั่นย่อย ADM_8MONO

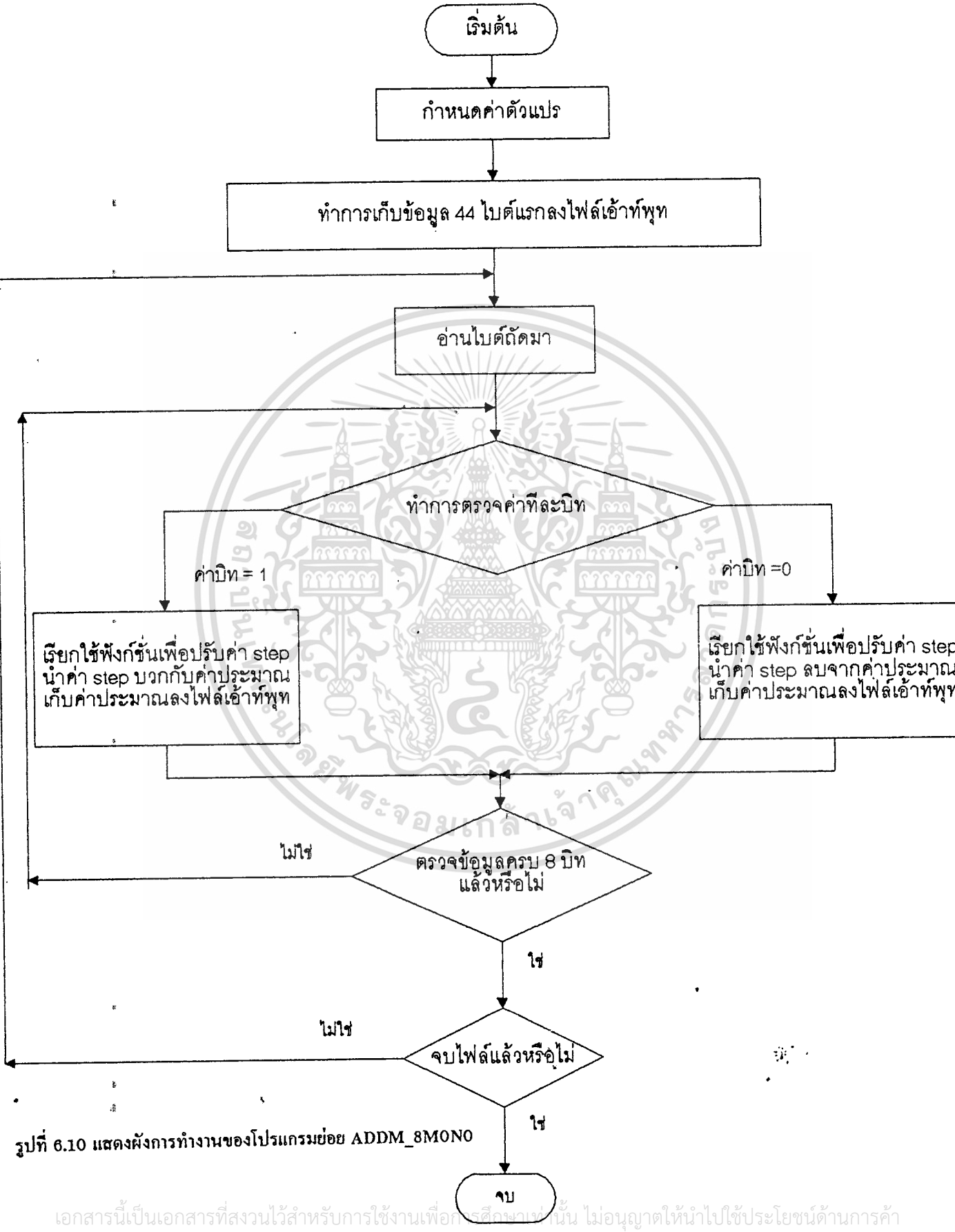


รูปที่ 6.8 แสดงผังการทำงานของโปรแกรมย่อย ADM_8M0N0



รูปที่ 6.9 แสดงผังการทำงานของโปรแกรม ADT_DDM.C

<u>หมายเหตุ</u> เนื่องจากการดีมอดูเลชั่นแบบปรับค่าได้ของไฟล์รูปแบบต่างๆ นั้นมีการทำงานที่คล้ายคลึงกัน จึงขอแสดงผังการทำงานสำหรับไฟล์ 8 บิทโมโนเท่านั้นคือฟังก์ชั่นย่อย ADDM_8MONO



รูปที่ 6.10 แสดงผังการทำงานของโปรแกรมย่อย ADDM_8M0N0

บทที่ 7

ผลการทดลอง

จากการใช้โปรแกรมหาค่าของสเต็ปไซด์โดยทำการทดลองกับไฟล์ .WAV ทั้ง 4 ชนิด คือ

1. MONO 8 bit
   - MONO811.WAV
   - RECORD.WAV
   - RECORD7.WAV
   - RECORD11.WAV
   - RECORD2.WAV
2. STEREO 8 bit
   - STE811.WAV
   - TONLOVE3.WAV
3. MONO 16 bit
   - MONO1611.WAV
4. STEREO 16 bit
   - STE1611.WAV
   - S_16_44.WAV

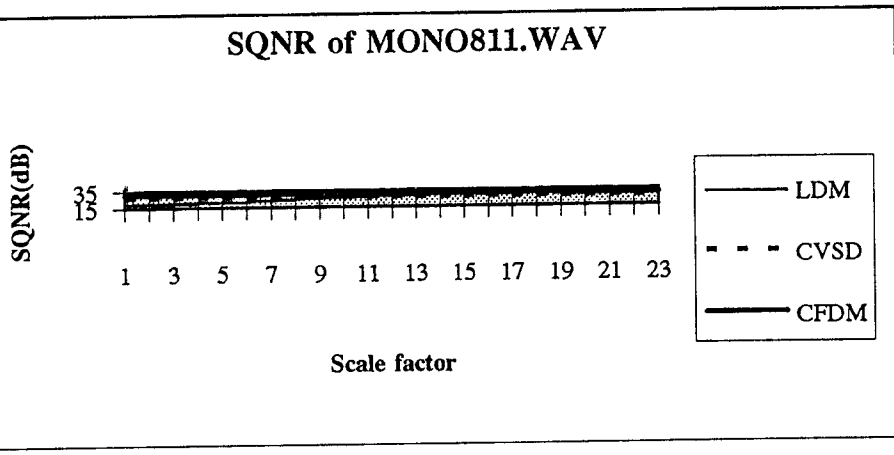
โดยแต่ละไฟล์บันทึกจากเสียงลักษณะต่างๆ มีรายละเอียดดังนี้

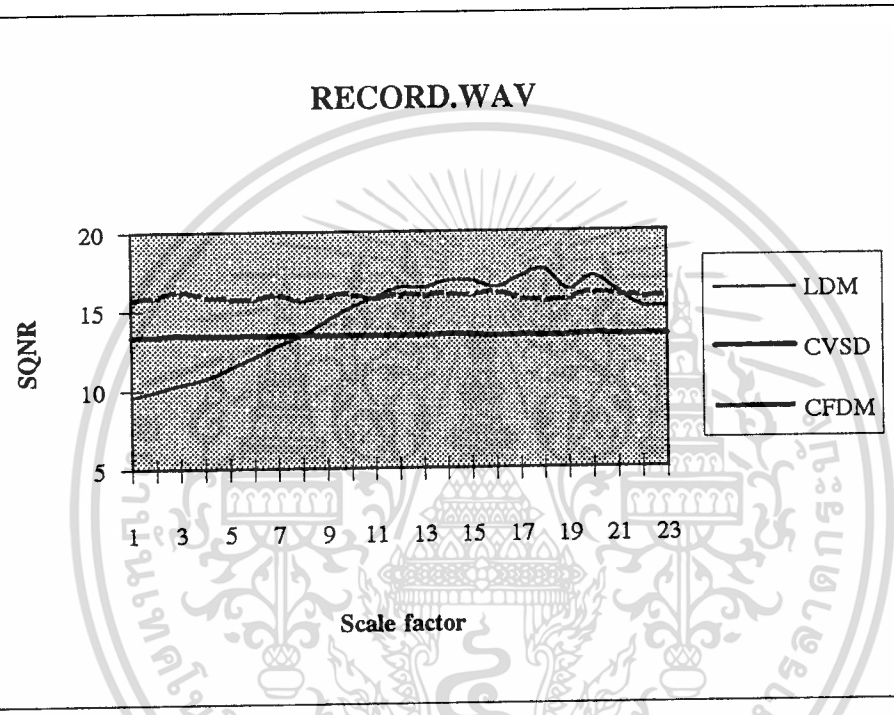
- จากวิทยุ ทั้งภาษาไทยและภาษาอังกฤษ ทั้งเสียงดังและเสียงเบา เช่น RECORD2 เป็นภาษาอังกฤษเสียงดัง, RECORD7 เป็นภาษาอังกฤษเสียงเบา RECORD เป็นเสียงภาษาไทย
- จากเสียงคนทั้งผู้ชายและผู้หญิง เช่น RECORD11 เป็นเสียงผู้ชาย
- แหล่งอื่นๆ เช่น S_16_44 เป็นเสียงจากเครื่องยนต์รถ

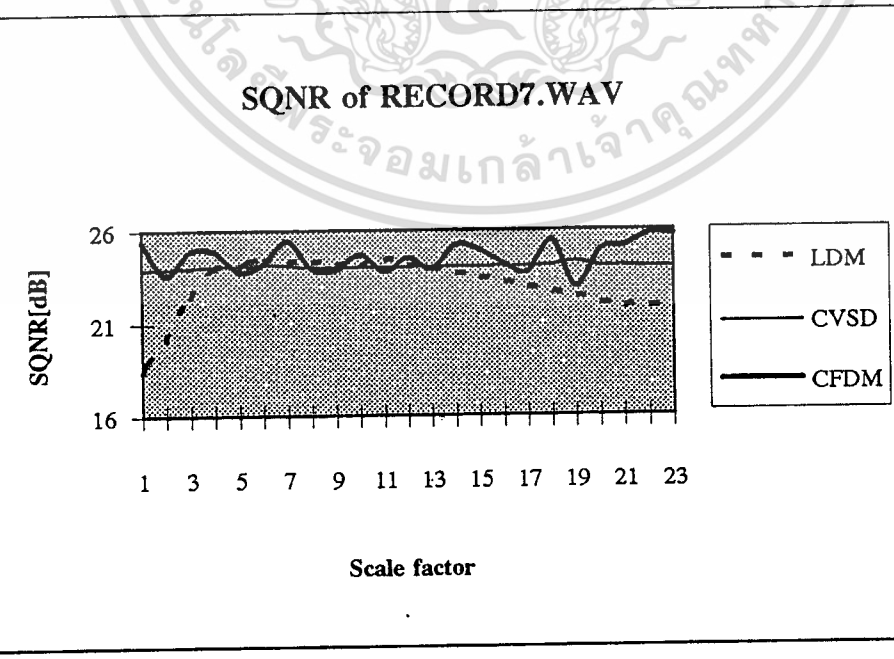
สามารถแสดงผลเป็นตารางและกราฟแสดงค่า SQNR ของไฟล์ชนิดต่างๆ ดังนี้

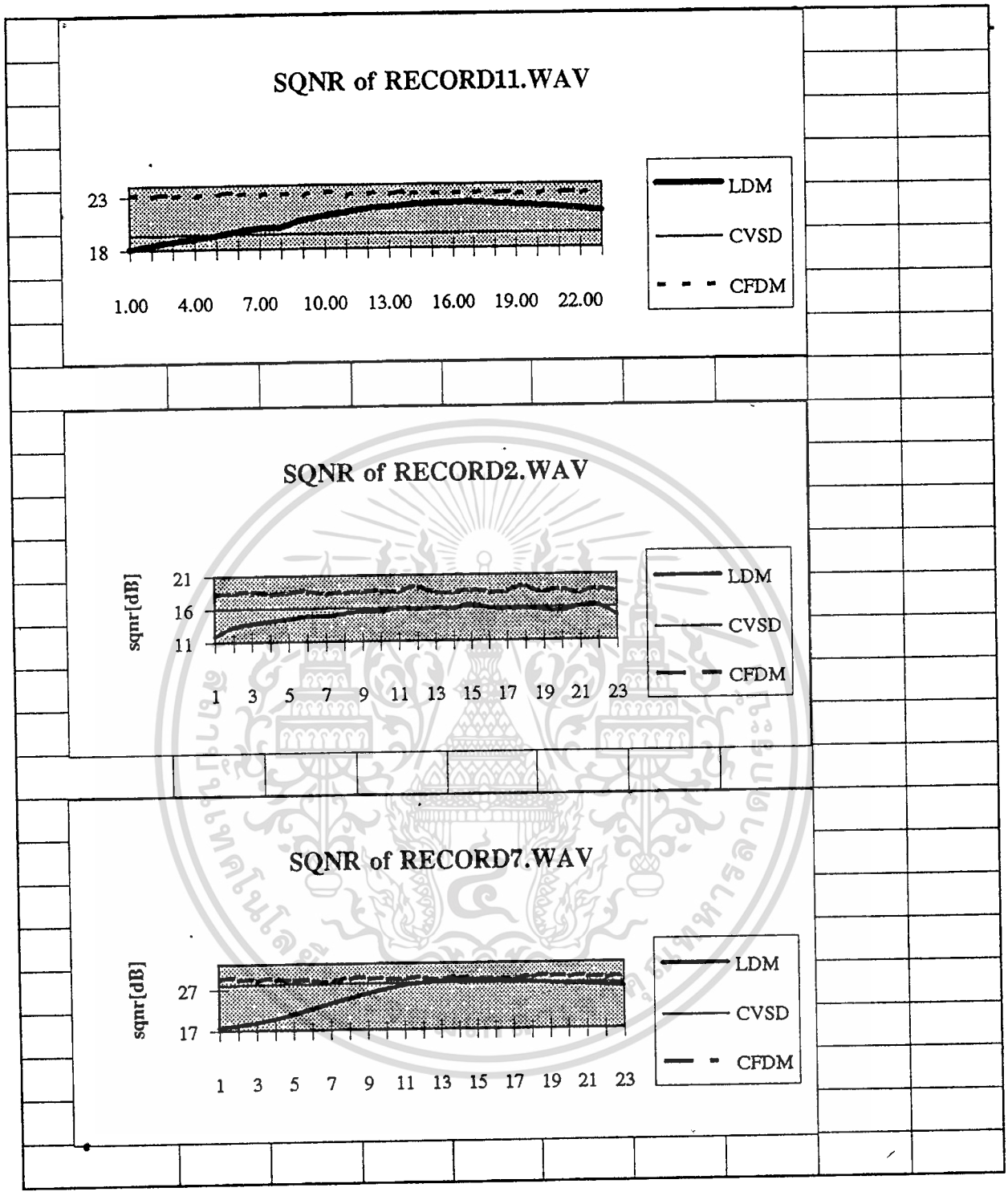
	การคำนวณค่า SQNR ของไฟล์ชนิด MONO 8 bit								
ชื่อไฟล์		MONO811.WAV			RECORD.WAV			RECORD7.WAV	
จำนวน sample	30000	30000	30000	30000	30000	30000	5000	5000	5000
rms.Slope energy	3.1416	3.1416	3.1416	6.11576	6.11576	6.11576	7.100014	7.100014	7.100014
Scale factor	LDM	CVSD	CFDM	LDM	CVSD	CFDM	LDM	CVSD	CFDM
0.1	19.381	26.52945	30.01249	9.591158	13.35684	15.6542	18.26527	23.85394	25.3814
0.2	19.847	26.54359	30.06939	9.960081	13.35628	15.7192	19.99867	**23.9412**	23.58897
0.3	20.32882	**26.5863**	30.05919	10.33361	13.45901	16.06144	22.31993	23.94861	24.90899
0.4	20.81708	26.5129	30.1337	10.73633	13.37223	15.80866	23.77824	24.05656	24.8501
0.5	21.4782	26.54965	30.015	11.35121	13.40438	15.67342	24.06648	24.06159	23.76329
0.6	22.2611	26.53463	30.13373	12.0503	13.41394	15.54495	24.30876	24.13908	24.19031
0.7	23.2414	26.55265	30.13394	12.79028	13.39828	15.8553	24.20041	24.12059	25.45385
0.8	24.3445	26.55938	30.13323	13.45206	13.47345	15.51771	24.26789	24.0064	23.93197
0.9	25.3937	26.55913	30.13367	14.34727	13.3778	15.83846	24.13036	24.00306	23.84875
1.0	26.2052	26.54382	30.0349	15.22441	13.38092	15.88984	23.96486	24.01183	24.69503
1.1	26.989	26.58518	30.1354	15.81196	13.37361	15.56806	**24.3282**	24.00642	23.7676
1.2	27.836	26.4787	30.13313	16.48766	13.40822	15.87355	24.13962	24.01481	24.52343
1.3	28.436	26.55781	30.06645	16.44133	13.40348	15.82228	23.92552	24.03538	23.91141
1.4	28.829	26.54237	30.09591	16.88676	13.46049	15.92184	23.69198	24.02396	25.19174
1.5	29.1592	26.55815	30.0978	16.86872	13.44138	15.76761	23.45117	24.02291	24.94688
1.6	29.00438	26.49398	**30.2172**	16.45153	13.36849	16.03734	23.19501	24.05551	24.1832
1.7	**29.2860**	26.54569	30.02488	17.11033	13.41018	15.61331	22.92607	24.0173	23.73957
1.8	28.9705	26.56804	30.13294	**17.5649**	13.33662	15.46651	22.65516	24.09986	25.46027
1.9	28.832	26.58105	30.12262	16.28406	13.40661	15.51648	22.3678	24.30995	22.93098
2.0	28.4507	26.54342	30.13671	17.13486	**13.4795**	**15.9304**	22.08412	24.03723	24.93191
2.1	28.336	26.56284	30.09565	16.16763	13.42355	15.82626	21.78967	24.07229	25.14399
2.2	28.0682	26.53629	30.12722	15.16106	13.37553	15.64447	21.82673	24.00343	**25.8028**
2.3	27.8059	26.53735	30.0582	15.16106	13.37468	15.66154	21.69811	24.00169	25.74461
best condition									
Scale factor	1.7	0.3	1.6	1.8	2	2	1.1	0.6	2.2
Stepsize	5.339841	0.942325	5.02656	11.00804	12.23115	12.23152	7.81006	4.26009	15.62003

	การคำนวณค่า SQNR ของไฟล์ชนิด MONO 8 bit(ต่อ)								
ชื่อไฟล์		RECORD11.WAV			RECORD2.WAV			RECORD7.WAV	
จำนวน sample	32500	32500	32500	32600	32600	32600	32500	32500	32500
rms.Slope energy	6.686632	6.686632	6.686632	3.952374	3.952374	3.952374	3.66989	3.66989	3.66989
Scale factor	LDM	CVSD	CFDM	LDM	CVSD	CFDM	LDM	CVSD	CFDM
0.1	18.12125	19.36617	23.09299	11.92644	16.02819	18.19452	17.74833	27.99412	29.28827
0.2	18.41079	19.34916	23.02231	13.30283	16.0138	18.10651	18.20955	28.03984	29.20299
0.3	18.73034	19.374	23.02231	13.78013	16.02371	18.38652	18.90409	28.13001	28.85306
0.4	19.01305	19.38031	22.96934	14.18406	16.02345	18.05547	19.80899	28.11799	28.97463
0.5	19.28739	19.37624	23.13095	14.4624	16.04258	18.18448	20.9156	28.15277	28.60553
0.6	19.67298	19.38128	23.12585	14.86123	16.02133	18.47517	22.14991	28.10169	28.66209
0.7	19.96627	19.37275	22.999	14.88994	16.0222	18.04105	23.4417	28.20137	28.12639
0.8	20.03836	**19.3839**	23.10996	15.22519	16.03838	18.27695	24.68288	28.16419	29.14484
0.9	20.71517	19.37648	22.96925	15.57307	16.04733	18.17023	25.81397	28.18716	29.20572
1.0	21.14335	19.38248	**23.2606**	15.50846	16.01159	18.38505	26.78871	**28.2205**	28.96575
1.1	21.4022	19.37529	22.92711	15.94333	16.03126	18.16092	27.80188	28.15561	28.65367
1.2	21.78167	19.38313	23.10988	15.86235	16.01649	18.94494	28.38063	28.16329	29.03558
1.3	21.92344	19.37482	23.04315	16.05182	16.03659	18.18785	28.60616	28.18082	28.47401
1.4	22.09224	19.37697	23.09919	15.89915	16.0444	17.90208	**29.0072**	28.20622	29.17245
1.5	22.20196	19.37851	23.10152	**16.4994**	16.00972	18.36851	28.94217	28.13787	28.58382
1.6	22.26891	19.37632	23.10746	15.97889	16.02413	18.17563	28.76105	28.16256	28.68613
1.7	**22.3071**	19.37595	23.08972	15.72292	16.01554	17.98966	28.56708	28.13188	28.65252
1.8	22.16859	19.37443	23.10971	15.87514	16.01421	**18.9648**	28.37426	28.18994	29.17274
1.9	22.06375	19.37193	23.0011	15.61106	16.00923	17.9869	28.09614	28.16758	**29.4962**
2.0	21.94927	19.3715	23.00582	15.3874	16.03121	18.49305	27.84654	28.15707	29.27101
2.1	21.79422	19.37829	23.09896	16.16086	16.00853	17.83343	27.75606	28.17311	29.26225
2.2	21.63596	19.37343	22.99024	16.2385	**16.0528**	18.44744	27.45341	28.1761	28.78815
2.3	21.43304	19.36836	23.02465	14.70224	16.0118	18.11253	27.17031	28.14362	29.15167
best condition									
Scale factor	1.7	0.8	1.0	1.5	2.2	1.8	1.4	1.0	1.9
Stepsize	11.36728	5.349306	6.686632	5.928561	8.69522	7.114272	5.137846	3.66989	6.972791



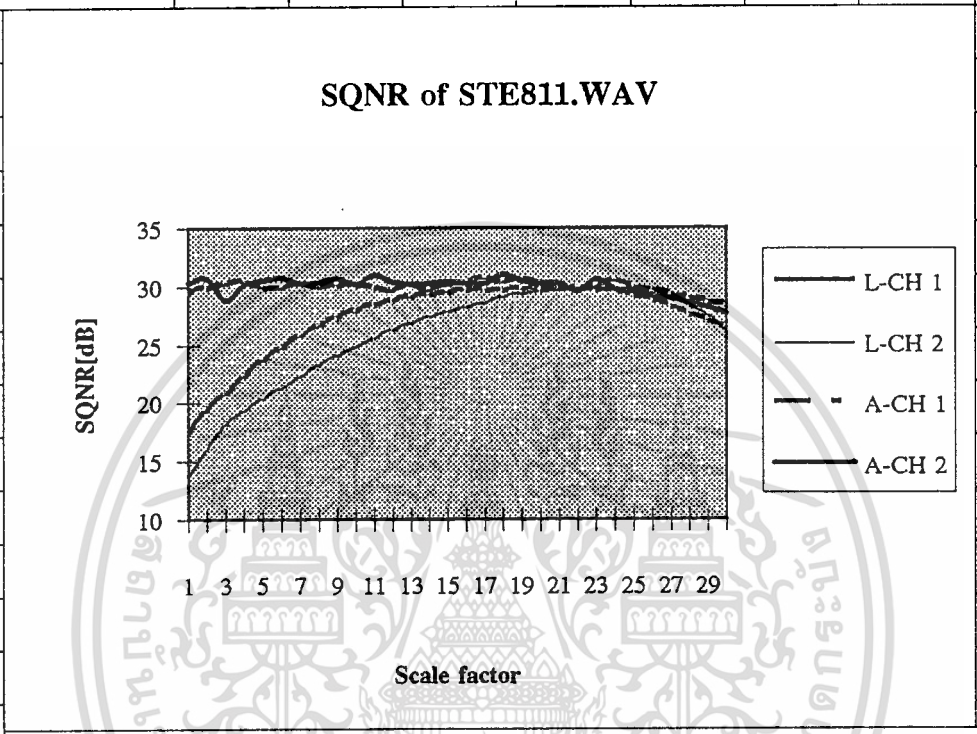


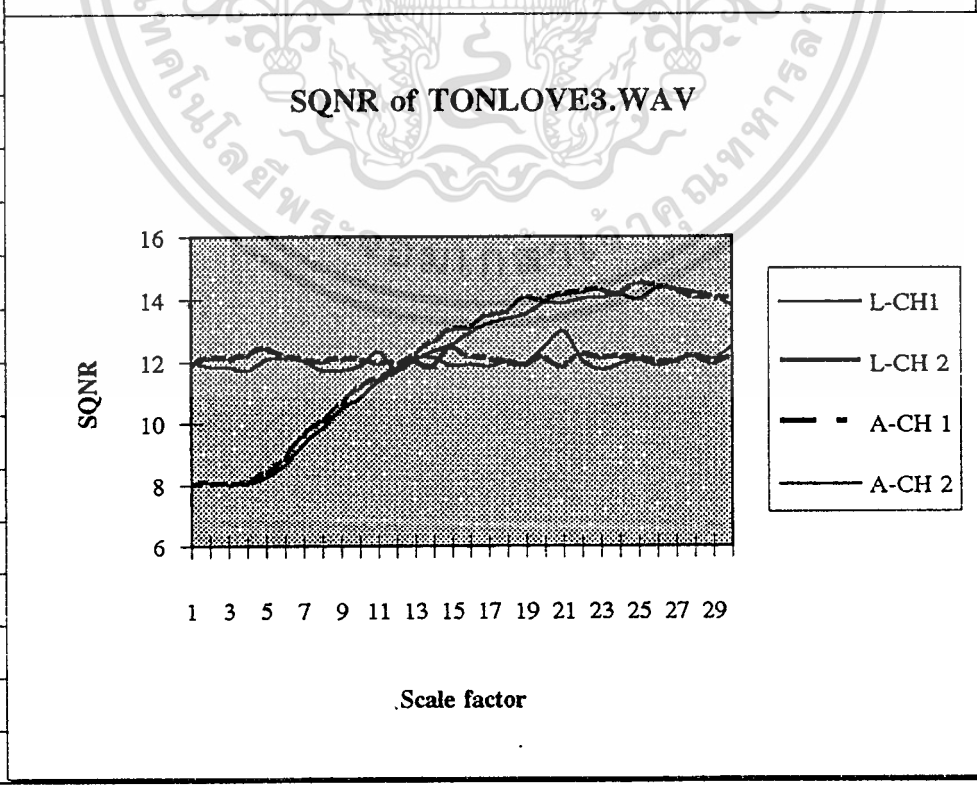




		การคำนวณค่า SQNR ของไฟล์ชนิด STEREO 8 bit						
ชื่อไฟล์		STE811.WAV				TONLOVE3.WAV		
เทคนิค	LDM		ADM		LDM		ADM	
จำนวน sample	30,000	30,000	30,000	30,000	30,000	30,000	30,000	30,000
rms. Slope energy	2.01344	1.3699	2.01344	1.3699	8.415174	8.836354	8.415174	8.836354
Scale factor	L-CH 1	L-CH 2	A-CH 1	A-CH 2	L-CH1	L-CH 2	A-CH 1	A-CH 2
0.1	17.1522	13.69909	29.43003	30.39577	8.041754	8.010239	11.88234	11.99627
0.2	19.19173	15.91316	29.86796	30.72354	8.07824	8.033504	12.04755	11.84804
0.3	20.68259	18.03487	30.10096	28.83618	8.031001	7.985066	12.04754	11.82848
0.4	22.11501	19.25334	30.32441	30.2262	8.036337	8.040166	12.13617	11.71707
0.5	23.49407	20.27852	29.83554	30.5174	8.281428	8.362299	12.35434	12.07809
0.6	24.58627	21.25618	29.716	30.73271	8.67837	8.772059	12.13616	12.10775
0.7	25.65404	22.23335	30.0914	30.27128	9.349163	9.541657	12.0317	12.02425
0.8	26.54205	23.23227	29.76626	30.35529	9.85715	10.01803	11.9265	11.72728
0.9	27.26346	24.01937	29.96904	30.71911	10.47701	10.56235	12.05376	11.71705
1	27.84594	24.80518	29.96282	30.22776	10.83935	11.07906	12.03802	11.85871
1.1	28.34901	25.51273	29.80424	30.97554	11.37271	11.41048	11.8845	12.31558
1.2	28.73352	26.23151	29.41715	30.37846	11.67847	11.77468	11.92657	11.72725
1.3	29.07203	26.81694	29.96773	30.2084	12.07936	12.14605	11.94903	12.09069
1.4	29.2256	27.32198	29.51559	30.24648	12.34182	12.58525	11.7665	12.14416
1.5	29.3722	27.79605	29.9696	30.42858	12.56285	12.95522	12.3802	11.85867
1.6	29.58623	28.14553	30.16578	30.28023	12.95143	13.0115	12.0883	11.93233
1.7	29.59495	28.49587	**30.6178**	30.28803	13.24038	13.41028	12.06241	11.85023
1.8	29.57679	29.0753	30.19109	**31.0486**	13.40396	13.5264	11.92651	12.03995
1.9	29.65664	29.31055	30.49309	30.5061	13.53209	13.99372	11.8247	11.96716
2	**29.6834**	29.50228	30.12946	30.25687	13.88452	13.89691	12.07201	**12.5164**
2.1	29.50346	29.6136	29.67134	30.24641	13.87626	14.09003	11.76642	12.97995
2.2	29.62212	29.69529	29.71473	29.59511	14.03655	14.15471	12.18773	12.05918
2.3	29.57341	29.84533	29.72541	30.61238	14.05429	14.23848	12.05672	11.74593
2.4	29.49112	**29.5563**	30.3608	29.80936	14.15687	14.07473	12.08819	11.93219
2.5	29.10766	29.84533	29.89655	29.54876	13.99468	**14.4465**	12.06423	12.0697
2.6	28.76543	29.69529	29.48765	29.23464	**14.3675**	14.36857	11.94213	11.86488
2.7	28.23458	29.49112	29.21345	28.96435	14.33372	14.2533	11.92639	12.03981
2.8	27.54376	28.22158	28.96546	28.54362	14.2259	13.98851	**12.1105**	12.18961

2.9	27.00876	27.50376	28.63215	28.15488	14.0969	13.97203	11.87645	12.09829
3	26.43279	26.15655	28.34257	27.65424	13.77978	14.00251	12.07186	12.51624
best condition								
scale factor	2	2.4	1.7	1.8	2.6	2.5	2.8	2
basic stepsize	4.02628	3.28778	3.42285	2.465836	21.8794	22.09089	23.5624	17.67271





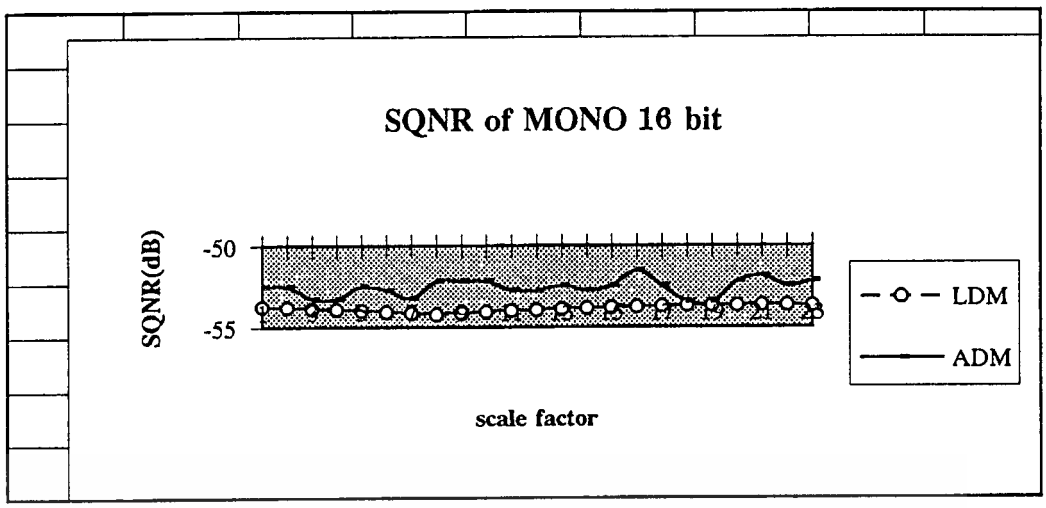
ผลการทำงานของโปรแกรมประมวลผลไฟล์ชนิด M0N0 16 bit

ชื่อไฟล์ : MONO1611.WAV

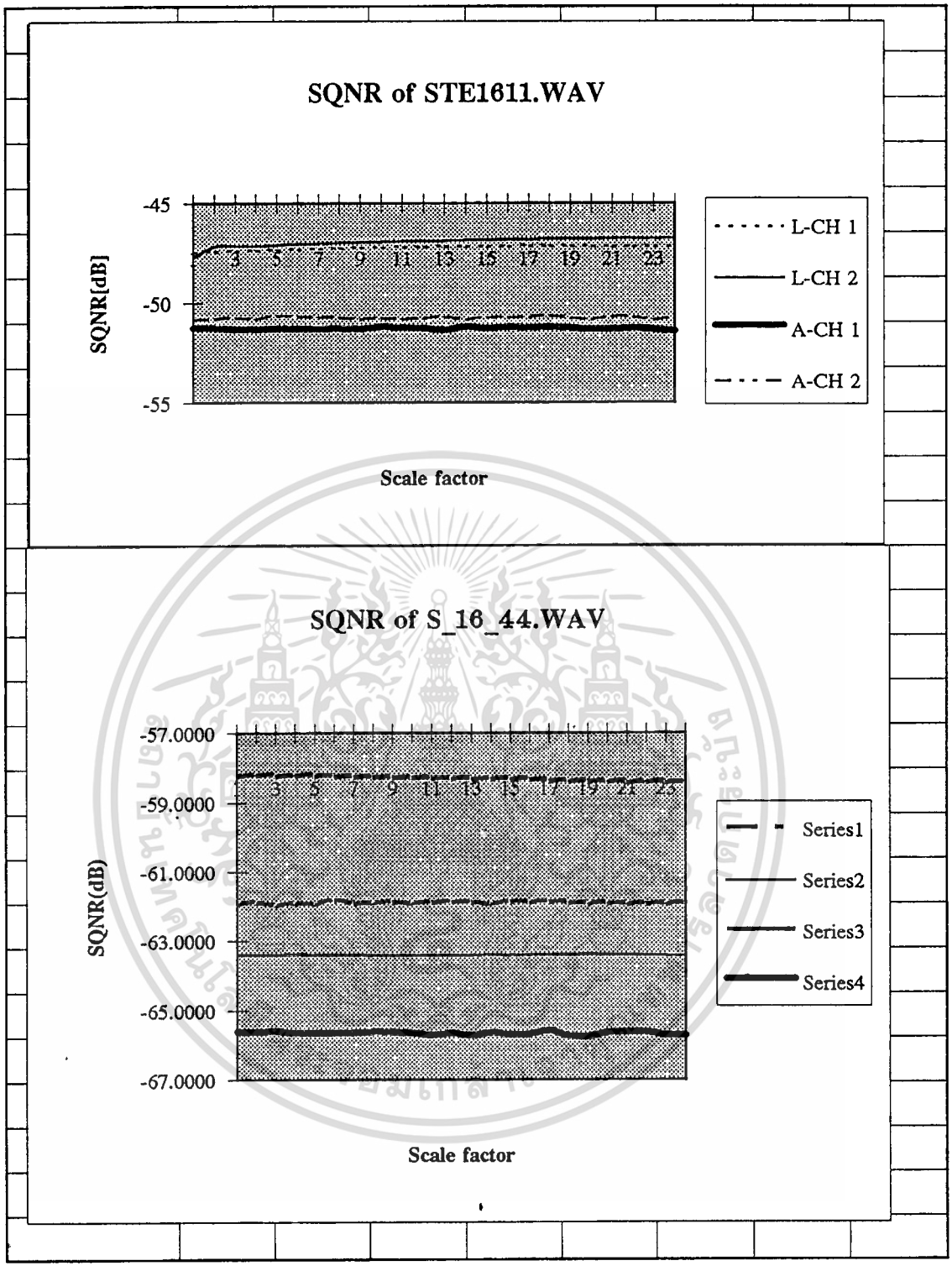
Slope Energy rms. : 2.090449

จำนวน sample : 30,000

Scale factor	SQNR(dB) LDM	ADM
656	-53.7478	-52.489
657	-53.802	-52.488
658	-53.8611	-53.2353
659	-53.9365	-53.2334
660	-54.0049	-52.4662
661	-54.0832	-52.7333
662	-54.1654	-53.2163
663	-54.2409	-52.172
664	-54.1625	-52.1682
665	-54.0882	-52.1669
666	-54.0181	-52.7288
667	-53.9525	-52.7234
668	-53.8973	-52.4577
669	-53.8414	-52.7239
670	-53.7964	-52.4546
671	-53.7627	-51.5245
672	-53.7288	-52.454
673	-53.6947	-53.4329
674	-53.6782	-53.4319
675	-53.6678	-52.1512
676	-53.6573	-51,8442
677	-53.6528	-52.4493
678	-53.6664	-52.1461
best condition		
Scale factor	677	671
basic step size	1394.33	1056.56



		การคำนวณค่า SQNR ของไฟล์ชนิด STEREO 16 bit						
ชื่อไฟล์		STE1611.WAV			S_16_44.WAV			
เทคนิค	LDM		ADM		LDM		ADM	
จำนวน sample	20,000	20,000	20,000	20,000	20,000	20,000	20,000	20,000
rms. Slope energy	36.1969	35.5373	36.1969	35.5373	38.7023	23.9215	38.7023	23.9215
Scale factor	L-CH 1	L-CH 2	A-CH 1	A-CH 2	L-CH 1	L-CH 2	A-CH 1	A-CH 2
300	-47.4752	-47.7286	-51.2587	-50.8443	**-58.2411**	**-63.3857**	-61.9537	-65.5954
301	-47.4432	-47.1849	-51.2566	-50.8281	-58.2442	-63.3917	-61.9163	-65.6121
302	-47.4138	-47.1487	-51.3244	-50.759	-58.2518	-63.3907	-62.0004	-65.5812
303	-47.3807	-47.1442	-51.3237	-50.8307	-58.2596	-63.3877	-61.9494	-65.6471
304	-47.3546	-47.0815	-51.2784	**-50.6783**	-58.2631	-63.3933	-61.9819	-65.6534
305	-47.3301	-47.0491	-51.2964	-50.7066	-58.2714	-63.3964	**-61.8552**	-65.6477
306	-47.3069	-47.0187	-51.2947	-50.7243	-58.28	-63.3995	-61.9452	-65.6263
307	-47.2868	-46.9896	-51.2685	-50.7579	-58.2897	-63.4062	-61.9604	-65.6027
308	-47.2675	-46.9635	-51.3161	-50.847	-58.2996	-63.413	-61.9101	-65.6193
309	-47.2504	-46.9378	-51.1936	-50.7844	-58.3098	-63.4199	-61.9752	-65.6545
310	-47.2357	-46.9147	-51.2417	-50.8384	-58.3211	-63.417	-61.9051	-65.7122
311	-47.2219	-46.8927	-51.2793	-50.7782	-58.3326	-63.4147	-61.8939	-65.6453
312	-47.2112	-46.8727	-51.3569	-50.7305	-58.3444	-63.4103	-61.9158	-65.734
313	-47.2021	-46.8545	**-51.1859**	-50.8664	-58.3573	-63.4012	-61.9832	-65.64
314	-47.1953	-46.8343	-51.2737	-50.7563	-58.3704	-63.4053	-61.9076	-65.6959
315	-47.1903	-46.8236	-51.2107	-50.7702	-58.3738	-63.4012	-61.9239	-65.6801
316	-47.187	-46.8111	-51.243	-50.7615	-58.3909	-63.4014	-61.8954	**-65.5732**
317	**-47.1861**	-46.8008	-51.2074	-50.6822	-58.4055	-63.3944	-61.9193	-65.7276
318	-47.1874	-46.7911	-51.2574	-50.7954	-58.4202	-63.3867	-61.9581	-65.7452
319	-47.1912	-46.7842	-51.3444	-50.8793	-58.4253	-63.3938	-61.9399	-65.6322
320	-47.1955	-46.7794	-51.2927	-50.7329	-58.4402	-63.4008	-61.9739	-65.6216
321	-47.2033	-46.7763	-51.2511	-50.7251	-58.446	-63.4092	-61.955	-65.6129
322	-47.2126	-46.7749	-51.3424	-50.8651	-58.4431	-63.4192	-61.9701	-65.7052
323	-47.2232	**-46.7707**	-51.4001	-50.7926	-58.4795	-63.4246	-61.9435	-65.7123
best condition								
scale factor	317	323	313	304	300	300	305	316
basic stepsize	8976.85	9130.5	11365.35	10803.37	8901.59	1161.2	9510.7	7682.4

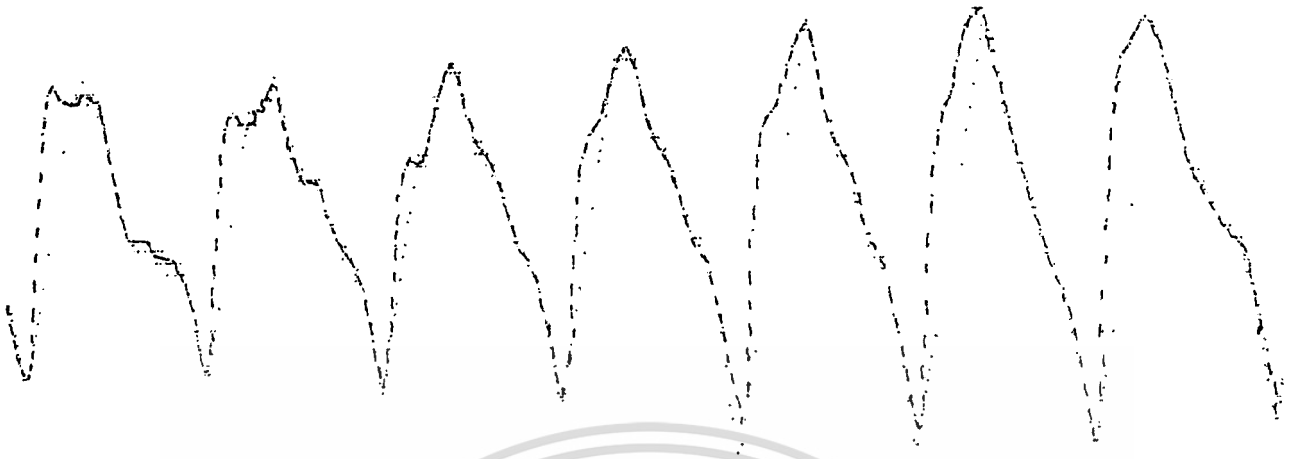


และเมื่อทำการมอดูเลชั่นทั้งวิธีลิเนียร์และอแดปทีฟเดลต้ามอดูเลชั่น กับไฟล์ชนิดต่างๆแล้วทำการเปรียบเทียบกับโปรแกรมลดขนาดไฟล์ชื่อ PKZIP แล้วผลที่ได้ดังแสดงในตาราง 7.1

ตารางที่ 7.1 แสดงขนาดของไฟล์ที่ผ่านการลดขนาดจากการใช้โปรแกรมต่างๆ

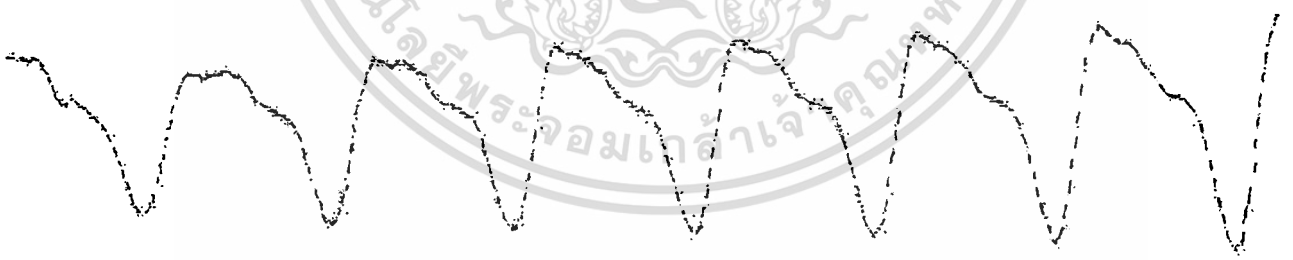
ชื่อไฟล์ (.WAV)	ชนิดของไฟล์	ขนาดไฟล์ (byte)	ขนาดหลังการมอดูเลชั่นแบบลิเนียร์	ขนาดหลังการมอดูเลชั่นแบบปรับค่า	ขนาดหลังใช้โปรแกรม PKZIP
MONO811	MONO 8 BIT	33116	4179	4179	10660
1	MONO 8 BIT	1025370	128210	128210	377235
STE811	STERIO 8 BIT	55164	6935	6935	12486
2	STERIO 8 BIT	1367144	170932	170932	541255
MONO1611	MONO 16 BIT	55164	3490	3490	45150
3	MONO 16 BIT	1830194	114429	114429	1505858
STE1611	STERIO 16 BIT	110284	6935	6935	81334
4	STERIO 16 BIT	3572144	223301	223301	2897972

จะเห็นได้ว่าโดยการใช้วิธีการมอดูเลชั่นทั้ง 2 แบบนั้น จะเป็นการลดขนาดของไฟล์ได้มากกว่าการลดโดยใช้โปรแกรม PKZIP และมีอัตราการลดขนาดไฟล์ค่อนข้างคงที่ โดยถ้าเป็น ไฟล์แบบ 8 บิท จะลดขนาดไฟล์ได้ประมาณ 8 เท่าและถ้าเป็นไฟล์ 16 บิทจะลดขนาดไฟล์ได้ประมาณ 16 เท่า สำหรับเวลาในการทำงานนั้น โปรแกรม PKZIP จะทำงานเร็วกว่าโปรแกรมการมอดูเลชั่นแบบเดลต้ามอดูเลชั่น และเมื่อทำการเล่นไฟล์นี้กลับคืนโดยใช้โปรแกรม sound blaster แล้วปรากฏว่าไฟล์ที่ได้จากโปรแกรมมอดูเลชั่นจะมีเสียงซ่าเกิดขึ้นในขณะที่เล่นและมีความดังของเสียงลดลง ซึ่งเสียงซ่าและความดังที่ลดลงนี้จะมีมากในการใช้โปรแกรมมอดูเลชั่นแบบลิเนียร์ และจะลดน้อยลงถ้าใช้โปรแกรมมอดูเลชั่นแบบปรับค่าได้ แต่หากใช้โปรแกรม PKZIP เป็นตัวจัดการแล้วจะไม่มีเสียงซ่าเกิดขึ้นเลย รูปแสดงหน้าจอขณะทำการมอดูเลชั่นทั้งสองแบบแสดงดังรูปที่ 7.1 และ 7.2



_._._._._ สัญญาณข้อมูล
..... สัญญาณค่าประมาณ

รูปที่ 7.1 แสดงหน้าจอขณะทำการมอดูเลชั่นแบบลิเนียร์



_._._._._ สัญญาณข้อมูล
..... สัญญาณค่าประมาณ

รูปที่ 7.2 แสดงหน้าจอขณะทำการมอดูเลชั่นแบบปรับค่า

บทที่ 8
สรุปผลการทดลอง และแนวทางการพัฒนาโครงงาน

จากการทดลองกับไฟล์ 8 bit ทั้งแบบ MONO และ STEREO ได้ค่า SQNR ที่สูงเป็นที่น่าพอใจ แสดงถึงข้อมูลที่ถูกดีโค้ดกลับมีความใกล้เคียงกับสัญญาณต้นฉบับ แสดงว่าสามารถใช้การคอมแพนดิ้งแบบ LDM และ ADM กับไฟล์ 8 bit ได้และในไฟล์เดียวกัน การประมวลผลโดยADM มีค่า SQNR สูงกว่า LDM เนื่องจาก

- LDM : แม้ว่า step-size บางค่าจะให้ค่า SQNR ค่อนข้างสูง แต่พบว่า ถ้าค่า step-size ดังกล่าวเปลี่ยนแปลงไปเพียงเล็กน้อย ค่า SQNR จะลดลงมากจึงไม่เหมาะสมที่จะใช้งานนัก

- ADM ค่า SQNR ค่อนข้างคงที่ไม่แตกต่างกันมาก เมื่อมีการเปลี่ยนแปลงค่า step-size ค่าstep-size บางค่าก็มีค่า SQNR สูงกว่า LDM แต่ก็มีบางค่าที่ให้ step-size ต่ำกว่า LDM แต่ส่วนใหญ่จะให้ค่าสูงกว่า LDM

ในการคอมแพนดิ้งข้อมูล 16 bit ทั้งไฟล์ประเภท MONO และ STEREO ไม่ว่าจะใช้เทคนิค LDM หรือ ADM จะได้ค่า SQNR ต่ำมาก แสดงว่าสัญญาณที่ดีโค้ด (decoded) กลับมามีความแตกต่างจากสัญญาณต้นฉบับมากและมีคุณภาพต่ำ ดังนั้นกระบวนการคอมแพนดิ้งดังกล่าวจึงไม่เหมาะสมที่จะใช้การสุ่มตัวอย่าง (sampling) แบบ 16 bit ต่อ 1 sample เนื่องจากข้อมูลในไฟล์ประเภทนี้มีค่าอยู่ระหว่าง -32768 ถึง +32767 โดยมี 0 เป็นค่ากลาง ซึ่งพบว่าช่วงห่าง (range) ของข้อมูลมีช่วงที่กว้างมาก ทำให้การคอมแพนดิ้งทั้งแบบ LDM และ ADM ตามสัญญาณได้ไม่ทันและเกิดเป็นสโลปโอเวอร์โหลดขึ้น

จากการทดสอบหลักการของการมอดูเลชั่นโดยการเขียนโปรแกรมจำลองการทำงานโดยนำมาประยุกต์ใช้กับไฟล์ .WAV ที่ได้จาก sound blaster นั้นจะเห็นได้ว่าหลักการดังกล่าวเป็นความจริงกล่าวคือ เป็นการลดขนาดไฟล์ได้ตามจำนวนบิทของสัญญาณ sample คือถ้าสัญญาณที่เก็บใช้ข้อมูล 8 บิทต่อ 1 sample โปรแกรมดังกล่าวก็สามารถลดขนาดไฟล์ได้ 8 เท่าเหมือนกับหลักการที่กล่าวไว้ในบทต้นๆ (คือ ลดให้ 1 sample เหลือเพียง 1 บิท) เช่นเดียวกับการทำกับไฟล์ 16 บิทก็ลดขนาดไฟล์ได้ 16 เท่าด้วย ซึ่งการลดขนาดไฟล์ได้ขนาดนี้จะมากกว่าการลดขนาดไฟล์ด้วยโปรแกรมจำพวก PKZIP แต่อย่างไรก็ตามไฟล์ที่ได้กลับคืนมาจากการดีมอดูเลชั่นจะไม่เหมือนไฟล์ต้นฉบับทุกประการ ทำให้การเล่นกับคืนมาเสียงที่ได้จะมีความเพี้ยนอยู่ด้วยคือเกิดเป็นเสียงซ่าๆ ขึ้นขณะที่เล่น ซึ่งเสียงซ่านี้ก็คือควอนไตซิ่งน้อยส์ที่เกิด

ขึ้นตามทฤษฎีนั่นเอง และจะเกิดจากสโลปโอเวอร์โหลดน้อยส์ด้วยสำหรับการมอดูเลชั่นแบบลิเนียร์ ทั้งยังเป็นการพิสูจน์ให้เห็นจริงว่าอัตราส่วนของสัญญาณต่อควอนไตซิ่งน้อยส์ (SQNR) ของการมอดูเลชั่นแบบปรับค่าได้จะมีค่าดีกว่าแบบลิเนียร์ แต่ปัญหานี้จะไม่เกิดขึ้นเลยสำหรับโปรแกรม PKZIP ทั้งความเร็วของโปรแกรม PKZIP ก็มากกว่าโปรแกรมมอดูเลชั่นนี้ด้วย ซึ่งจุดนี้จะเป็นจุดที่ผู้ที่สนใจการลดขนาดไฟล์สามารถนำไปพัฒนาเป็นโปรแกรมใหม่ได้ต่อไปโดยการเพิ่มความเร็วของการทำงานของโปรแกรมรวมทั้งการลดน้อยส์ที่เกิดขึ้นโดยใช้เทคนิคการมอดูเลชั่นแบบใหม่ๆ ได้

ภาคผนวก

หน้า

ภาคผนวก ก. แสดงโปรแกรม DELTA.C

```
#include<stdio.h>
#include<conio.h>
#include<process.h>

void main(void)
{
char key;

      DISPLAY();
       key=getche();
      printf("\n");
      do
      {
       switch(key)
       {
          case '1' :  spawnl(P_WAIT,"stepsize.exe",NULL);
                      break;
          case '2' :  spawnl(P_WAIT,"ln_dm.exe",NULL);
                      break;
          case '3' :  spawnl(P_WAIT,"ln_ddm.exe",NULL);
                      break;
          case '4' :  spawnl(P_WAIT,"adt_dm.exe",NULL);
                      break;
          case '5' :  spawnl(P_WAIT,"adt_ddm.exe",NULL);
                      break;
          case '6' :  exit(0);
       }
      DISPLAY();
      key=getch();
       }while(key!=3);
}

DISPLAY()
{
    clrscr();
      printf("\n \n \n");
      printf("This program is a DELTA MOD/DEMODULATION program \n \n");
      printf("     << This program use Linear and Adaptive Delta ");
      printf("Modulation algorithms >> \n \n");
      printf("Do you want to \n");
      printf("    1.FIND STEP SIZE FOR .WAV FILE \n");
      printf("    2.LINEAR DELTA MODULATION (compress file) \n");
      printf("    3.LINEAR DELTA DEMODULATION (expand file) \n");
      printf("    4.ADAPTIVE DELTA MODULATION (compress file) \n");
      printf("    5.ADAPTIVE DELTA DEMODULATION (expand file) \n");
      printf("    6.exit to DOS \n");
      printf("your choice is: ");
      printf("\n \n \n \n \n \n \n ");
      printf("NOTE: It will compress your .WAV file about 8 times for \n");
      printf("       8 bits record and about 16 times for 16 bits record. \
n");
}

□
```

ภาคผนวก ข. แสดงโปรแกรม LN_DM.C

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<dos.h>
#include<process.h>
#include<math.h>

struct byte{
  unsigned int LSB  : 1;
  unsigned int bit2 : 1;
  unsigned int bit3 : 1;
  unsigned int bit4 : 1;
  unsigned int bit5 : 1;
  unsigned int bit6 : 1;
  unsigned int bit7 : 1;
  unsigned int MSB  : 1;
     } BYTE;
int bit_acc,byte,byte_count,DM_format,int1,
    estimate,L_estimate,R_estimate,
    samp_value,L_samp_value,R_samp_value,
    maxx,maxy,midx,midy;
int step_8,step_16;
int gdriver=DETECT,gmode;
int equal_value=0,L_equal_value=0,R_equal_value=0;
int x=5,L_x=5,R_x,y1,y2,L_y1,L_y2,R_y1,R_y2;
FILE *fp1,*fp2;
char char1;

main()
{
char input[40],output[40];

   clrscr();

      printf("You select LINEAR MODULATION process \n");
      printf("    you should give the file name in DOS structure \n");
      printf("     e.g. c:\\wavedata\\mono8.wav \n \n");
      printf("  please enter your input file name ---> ");
       scanf("%s",input);
      printf("  enter your output file name       ---> ");
       scanf("%s",output);
      printf("\n         And please enter the step size value");
      printf("\n              suggest   3 for  8 bits");
      printf("\n                        and 150 for 16 bits \n");
      printf("\n and remember this value for uses DEMODULATION process \n");
      printf("\n   enter the step size for  8 bits ---> ");
       scanf("%d",&step_8);
      printf("                step size for 16 bits ---> ");
       scanf("%d",&step_16);
      printf("\n");

      fp1 = fopen(input,"rb");
      fp2 = fopen(output,"wb");

     chkwave_file();
     chkbit_record();
     chkfile_format();
     chksampling_rate();

      printf("\n \n \n \n \n \n \n \n \n \n");
      printf("              Press any key to continue. \n");
      getch();

  clrscr();

  initgraph(&gdriver,&gmode,"f:\\lang\\tc");
       maxx=getmaxx();
       maxy=getmaxy();
       midx=maxx/2;
       midy=maxy/2;

   LINEAR_DM();

      fclose(fp1);
      fclose(fp2);
```

```
    getch();
      closegraph();
}

chkwave_file()
{
  fseek(fp1,8,0);
  char1=fgetc(fp1);
  if(char1=='W')
     { char1=fgetc(fp1);
       if(char1=='A')
       { char1=fgetc(fp1);
         if(char1=='V')
            { char1=fgetc(fp1);
              if(char1=='E')
                  printf("WAVE FILE \n");
              else
                  printf("This file is not a WAVE file! \n");
            }
         else
            printf("This file is not a WAVE file! \n");
       }
       else
       printf("This file is not a WAVE file! \n");
     }
  else
       printf("This file is not a WAVE file! \n");
}

chkbit_record()
{
  fseek(fp1,34,0);
  char1=fgetc(fp1);
  int1=char1;
  switch(int1)
     {
       case  8   : printf("  data is stored in 8 bit PCM \n");
                   DM_format+=0;
                   break;
       case  16  : printf("  data is stored in 16 bit PCM \n");
                   DM_format+=1;
                   break;
     }
}

chkfile_format()
{
  fseek(fp1,22,0);
  char1=fgetc(fp1);
  int1=char1;
  switch(int1)
     {
       case  1   : printf("  This file save in MONO system \n");
                   DM_format+=0;
                   break;
       case  2   : printf("  This file save in STEREO system \n");
                   DM_format+=2;
                   break;
     }
}

chksampling_rate()
{
   fseek(fp1,24,0);
   char1=fgetc(fp1);
   int1=char1;
   switch(int1)
     {
      case  17  : printf("  sampling rate = 11,025 Hz \n");
                  break;
      case  34  : printf("  sampling rate = 22,050 Hz \n");
                  break;
      case  68  : printf("  sampling rate = 44,100 Hz \n");
                  break;
      }
}

LINEAR_DM()
{
  switch(DM_format)
```

```
    {
      case  0  : LDM_8mono();
                 break;
      case  1  : LDM_16mono();
                 break;
      case  2  : LDM_8stereo();
                 break;
      case  3  : LDM_16stereo();
                 break;
    }
}

LDM_8mono()
{
  fseek(fp1,0,0);
  for(byte=1;byte<=44;byte++)
   {
      char1=fgetc(fp1);
      fputc(char1,fp2);
   }
  estimate=128;
do
{
  bit_acc+=1;                    /* account bit in the byte */
     fscanf(fp1,"%c",&samp_value);
  if(samp_value>estimate)        /* next data is greater than old data */
    {
       write_byte(1);
       estimate=estimate+step_8;
    }
  else
    {  if(samp_value<estimate) /* next data is less than old data */
       {
          write_byte(0);
          estimate=estimate-step_8;
       }
       else
       {
       equal_value+=1;   /* next value is equal old value */
       if(equal_value==1)
       {
         write_byte(1);
         estimate=estimate+step_8;
       }
       else
       {
          write_byte(0);
          estimate=estimate-step_8;
          equal_value = equal_value-2;
       }
       }
    }

if(bit_acc==8 || feof(fp1))
{
  fprintf(fp2,"%c",BYTE);
  bit_acc = 0;
}

   y1=(256-samp_value)*(maxy/256.0);
   y2=(256-estimate)*(maxy/256.0);
     putpixel(x,y1,LIGHTMAGENTA);
     putpixel(x,y2,LIGHTGREEN);
   x+=1;
   if(x==maxx-4)
   {
     cleardevice();
     x=5;
   }
}while(!feof(fp1));
}

LDM_8stereo()
{
  L_x=5;  R_x=midx+5;
  fseek(fp1,0,0);
  for(byte=1;byte<=44;byte++)
     {
       char1=fgetc(fp1);
       fputc(char1,fp2);
```

```
        }
      L_estimate=128; R_estimate=128;
      byte_count=0;
      bit_acc=0;
    do
    {
      bit_acc+=1;
      byte_count+=1;
      if(byte_count==1)
      {
          fscanf(fp1,"%c",&L_samp_value);
          if(L_samp_value>L_estimate)
          {
            write_byte(1);
            L_estimate=L_estimate+step_8;
          }
          else
          {
          if(L_samp_value<L_estimate)
          {
            write_byte(0);
            L_estimate=L_estimate-step_8;
          }
          else
          {
             L_equal_value+=1;
             if(L_equal_value==1)
             {
               write_byte(1);
               L_estimate=L_estimate+step_8;
             }
             else
             {
               write_byte(0);
               L_estimate=L_estimate-step_8;
               L_equal_value = L_equal_value-2;
             }
          }
          }
          L_y1=(256-L_samp_value)*(maxy/256.0);
          L_y2=(256-L_estimate)*(maxy/256.0);
          putpixel(L_x,L_y1,LIGHTMAGENTA);
          putpixel(L_x,L_y2,LIGHTGREEN);
          L_x+=1;
          if(L_x==midx-4)
          {
            L_x=5;
          }
    }
    else
    {
          fscanf(fp1,"%c",&R_samp_value);
          if(R_samp_value>R_estimate)
          {
            write_byte(1);
            R_estimate=R_estimate+step_8;
          }
          else
          {
            if(R_samp_value<R_estimate)
            {
              write_byte(0);
              R_estimate=R_estimate-step_8;
            }
            else
            {
                R_equal_value+=1;
                if(R_equal_value==1)
                {
                  write_byte(1);
                  R_estimate=R_estimate+step_8;
                }
                else
                {
                  write_byte(0);
                  R_estimate=R_estimate-step_8;
                  R_equal_value = R_equal_value-2;
                }
            }
          }
```

```
      byte_count-=2;

      R_y1=(256-R_samp_value)*(maxy/256.0);
      R_y2=(256-R_estimate)*(maxy/256.0);
      putpixel(R_x,R_y1,LIGHTMAGENTA);
      putpixel(R_x,R_y2,LIGHTGREEN);
      R_x+=1;
      if(R_x==maxx-5)
      {
        cleardevice();
        R_x=midx+5;
      }
  }

if(bit_acc==8 || feof(fp1))
{
     fprintf(fp2,"%c",BYTE);
     bit_acc = 0;
}

}while(!feof(fp1));
}

LDM_16mono()
{
  fseek(fp1,0,0);
  for(byte=1;byte<=44;byte++)
     {
       char1=fgetc(fp1);
       fputc(char1,fp2);
     }
  estimate=0;
do
{
  bit_acc+=1;
  fread(&samp_value,2,1,fp1);
  if(samp_value>estimate)
  {
     write_byte(1);
     estimate=estimate+step_16;
  }
  else
  {
     if(samp_value<estimate)
     {
       write_byte(0);
       estimate=estimate-step_16;
     }
     else
     {
       equal_value+=1;
       if(equal_value==1)
       {
         write_byte(1);
         estimate=estimate+step_16;
       }
       else
       {
         write_byte(0);
         estimate=estimate-step_16;
         equal_value = equal_value-2;
       }
      }
   }

if(bit_acc==8 || feof(fp1))
{
    fprintf(fp2,"%c",BYTE);
    bit_acc = 0;
}

   y1=(32768-samp_value)*(maxy/65536.0);
   y2=(32768-estimate)*(maxy/65536.0);
      putpixel(x,y1,LIGHTMAGENTA);
      putpixel(x,y2,LIGHTGREEN);
   x+=1;
   if(x==maxx-4)
   {
     cleardevice();
     x=5;
```

```
    }
}while(!feof(fp1));
}

LDM_16stereo()
{
  L_x=5;  R_x=midx+5;
  fseek(fp1,0,0);
  for(byte=1;byte<=44;byte++)
   {
      char1=fgetc(fp1);
      fputc(char1,fp2);
   }
  L_estimate=0; R_estimate=0;
  byte_count=0;
do
{
  bit_acc+=1;
  byte_count+=1;
  if(byte_count==1)
  {
      fread(&L_samp_value,2,1,fp1);
      if(L_samp_value>L_estimate)
      {
        write_byte(1);
        L_estimate=L_estimate+step_16;
      }
      else
      {
        if(L_samp_value<L_estimate)
        {
            write_byte(0);
            L_estimate=L_estimate-step_16;
        }
        else
        {
            L_equal_value+=1;
            if(L_equal_value==1)
            {
              write_byte(1);
              L_estimate=L_estimate+step_16;
            }
            else
            {
              write_byte(0);
              L_estimate=L_estimate-step_16;
              L_equal_value = L_equal_value-2;
            }
        }
      }
      L_y1=(32768-L_samp_value)*(maxy/65536.0);
      L_y2=(32768-L_estimate)*(maxy/65536.0);
        putpixel(L_x,L_y1,LIGHTMAGENTA);
        putpixel(L_x,L_y2,LIGHTGREEN);
      L_x+=1;
      if(L_x==midx-4)
      {
         L_x=5;
      }
   }
   else
   {
      fread(&R_samp_value,2,1,fp1);
      if(R_samp_value>R_estimate)
      {
          write_byte(1);
          R_estimate=R_estimate+step_16;
      }
      else
      {
          if(R_samp_value<R_estimate)
          {
            write_byte(0);
            R_estimate=R_estimate-step_16;
          }
          else
          {
              R_equal_value+=1;
              if(R_equal_value==1)
                {
```

```
                write_byte(1);
                R_estimate=R_estimate+step_16;
              }
              else
              {
                write_byte(0);
                R_estimate=R_estimate-step_16;
                R_equal_value = R_equal_value-2;
              }
            }     .
        }
        byte_count-=2;
        R_y1=(32768-R_samp_value)*(maxy/65536.0);
        R_y2=(32768-R_estimate)*(maxy/65536.0);
           putpixel(R_x,R_y1,LIGHTMAGENTA);
           putpixel(R_x,R_y2,LIGHTGREEN);
        R_x+=1;
        if(R_x==maxx-5)
        {
          cleardevice();
          R_x=midx+5;
        }
   }

   if(bit_acc==8 || feof(fp1))
   {
       fprintf(fp2,"%c",BYTE);
       bit_acc = 0;
   }

}while(!feof(fp1));
}

write_byte(int bit_value)
{
switch(bit_acc) {
 case 1 : BYTE.MSB  = bit_value; break;
 case 2 : BYTE.bit7 = bit_value; break;
 case 3 : BYTE.bit6 = bit_value; break;
 case 4 : BYTE.bit5 = bit_value; break;
 case 5 : BYTE.bit4 = bit_value; break;
 case 6 : BYTE.bit3 = bit_value; break;
 case 7 : BYTE.bit2 = bit_value; break;
 case 8 : BYTE.LSB  = bit_value; break;
          }
}

□
```

ภาคผนวก ค. แสดงโปรแกรม LN_DDM.C

```
#include<stdio.h>
#include<conio.h>
#include<dos.h>
#include<process.h>

int byte,DM_format,int1,
    estimate,L_estimate,R_estimate,
    samp_value,L_samp_value,R_samp_value,
    maxx,maxy,midx,midy,step_8,step_16;
FILE *fp1,*fp2;
char char1;

main()
{
char input[40],output[40];

   clrscr();

      printf("You select LINEAR DEMODULATION process \n");
      printf("   you should give the file name in DOS structure \n");
      printf("     e.g. c:\\wavedata\\mono8.wav \n \n ");
      printf("  please enter your input file name ---> ");
      scanf("%s",input);
      printf("  enter your output file name       ---> ");
      scanf("%s",output);
      printf("\n        And please enter the step size value");
      printf("\n             suggest   3 for  8 bits");
      printf("\n                    and 100 for 16 bits");
      printf("\n and remember this value for uses DEMODULATION process \n");
      printf("\n   enter the step size for  8 bits ---> ");
       scanf("%d",&step_8);
      printf("                step size for 16 bits ---> ");
       scanf("%d",&step_16);
      printf("\n");

   fp1 = fopen(input,"rb");
   fp2 = fopen(output,"wb");

      chkwave_file();
      chkbit_record();
      chkfile_format();
      chksampling_rate();

      printf("\n \n \n \n \n \n \n \n \n \n");
      printf("                Press any key to continue. \n");
      getch();

    clrscr();

     LINEAR_DDM();

    fclose(fp1);
    fclose(fp2);
}

LINEAR_DDM()
{
  switch(DM_format)
   {
     case  0  : LDDM_8mono();
                break;
     case  1  : LDDM_16mono();
                break;
     case  2  : LDDM_8stereo();
                break;
     case  3  : LDDM_16stereo();
                break;
   }
}

LDDM_8mono()
{
 fseek(fp1,0,0);
 for(byte=1;byte<=44;byte++)
    {
      char1=fgetc(fp1);
```

```
        fputc(char1,fp2);
      }
    estimate=128;
  do
  {
    fread(&samp_value,1,1,fp1);
    if(samp_value & 0x80)
    {
        estimate=estimate+step_8;
        fprintf(fp2,"%c",estimate);
    }
    else
    {
        estimate=estimate-step_8;
        fprintf(fp2,"%c",estimate);
    }
    if(samp_value & 0x40)
    {
        estimate=estimate+step_8;
        fprintf(fp2,"%c",estimate);
    }
    else
    {
        estimate=estimate-step_8;
        fprintf(fp2,"%c",estimate);
    }
    if(samp_value & 0x20)
    {
        estimate=estimate+step_8;
        fprintf(fp2,"%c",estimate);
    }
    else
    {
        estimate=estimate-step_8;
        fprintf(fp2,"%c",estimate);
    }
    if(samp_value & 0x10)
    {
        estimate=estimate+step_8;
        fprintf(fp2,"%c",estimate);
    }
    else
    {
        estimate=estimate-step_8;
        fprintf(fp2,"%c",estimate);
    }
    if(samp_value & 0x08)
    {
        estimate=estimate+step_8;
        fprintf(fp2,"%c",estimate);
    }
    else
    {
        estimate=estimate-step_8;
        fprintf(fp2,"%c",estimate);
    }
    if(samp_value & 0x04)
    {
         estimate+=step_8;
         fprintf(fp2,"%c",estimate);
    }
    else
    {
          estimate-=step_8;
          fprintf(fp2,"%c",estimate);
    }
    if(samp_value & 0x02)
    {
         estimate+=step_8;
         fprintf(fp2,"%c",estimate);
    }
    else
    {
          estimate-=step_8;
          fprintf(fp2,"%c",estimate);
    }
    if(samp_value & 0x01)
    {
         estimate+=step_8;
         fprintf(fp2,"%c",estimate);
```

```
  }
  else
  {
        estimate-=step_8;
        fprintf(fp2,"%c",estimate);
   }

}while(!feof(fp1));
}

LDDM_8stereo()
{
  fseek(fp1,0,0);
  for(byte=1;byte<=44;byte++)
     {
       char1=fgetc(fp1);
       fputc(char1,fp2);
     }
  L_estimate=128; R_estimate=128;
do
{
  fread(&samp_value,1,1,fp1);
  if(samp_value & 0x80)
  {
       L_estimate+=step_8;
       fprintf(fp2,"%c",L_estimate);
  }
  else
  {
        L_estimate-=step_8;
        fprintf(fp2,"%c",L_estimate);
  }
  if(samp_value & 0x40)
  {
       R_estimate+=step_8;
       fprintf(fp2,"%c",R_estimate);
  }
  else
  {
        R_estimate-=step_8;
        fprintf(fp2,"%c",R_estimate);
  }
  if(samp_value & 0x20)
  {
       L_estimate+=step_8;
       fprintf(fp2,"%c",L_estimate);
  }
  else
  {
        L_estimate-=step_8;
        fprintf(fp2,"%c",L_estimate);
  }
  if(samp_value & 0x10)
  {
       R_estimate+=step_8;
       fprintf(fp2,"%c",R_estimate);
  }
  else
  {
        R_estimate-=step_8;
        fprintf(fp2,"%c",R_estimate);
  }
  if(samp_value & 0x08)
  {
       L_estimate+=step_8;
       fprintf(fp2,"%c",L_estimate);
  }
  else
  {
        L_estimate-=step_8;
        fprintf(fp2,"%c",L_estimate);
  }
  if(samp_value & 0x04)
  {
       R_estimate+=step_8;
       fprintf(fp2,"%c",R_estimate);
  }
  else
  {
        R_estimate-=step_8;
```

```
        fprintf(fp2,"%c",R_estimate);
    }
    if(samp_value & 0x02)
    {
        L_estimate+=step_8;
        fprintf(fp2,"%c",L_estimate);
    }
    else
    {
        L_estimate-=step_8;
        fprintf(fp2,"%c",L_estimate);
    }
    if(samp_value & 0x01)
    {
        R_estimate+=step_8;
        fprintf(fp2,"%c",R_estimate);
    }
    else
    {
        R_estimate-=step_8;
        fprintf(fp2,"%c",R_estimate);
    }

}while(!feof(fp1));
}

LDDM_16mono()
{
  fseek(fp1,0,0);
  for(byte=1;byte<=44;byte++)
     {
       char1=fgetc(fp1);
       fputc(char1,fp2);
     }
  estimate=0;
do
{
  fread(&samp_value,1,1,fp1);
  if(samp_value & 0x80)
  {
      estimate=estimate+step_16;
      fwrite(&estimate,2,1,fp2);
  }
  else
  {
      estimate=estimate-step_16;
      fwrite(&estimate,2,1,fp2);
  }
  if(samp_value & 0x40)
  {
      estimate=estimate+step_16;
      fwrite(&estimate,2,1,fp2);
  }
  else
  {
      estimate=estimate-step_16;
      fwrite(&estimate,2,1,fp2);
  }
  if(samp_value & 0x20)
  {
      estimate=estimate+step_16;
      fwrite(&estimate,2,1,fp2);
  }
  else
  {
      estimate=estimate-step_16;
      fwrite(&estimate,2,1,fp2);
  }
  if(samp_value & 0x10)
  {
      estimate=estimate+step_16;
      fwrite(&estimate,2,1,fp2);
  }
  else
  {
      estimate=estimate-step_16;
      fwrite(&estimate,2,1,fp2);
  }
  if(samp_value & 0x08)
  {
```

```
        estimate=estimate+step_16;
        fwrite(&estimate,2,1,fp2);
    }
    else
    {
        estimate=estimate-step_16;
        fwrite(&estimate,2,1,fp2);
    }
    if(samp_value & 0x04)
    {
        estimate=estimate+step_16;
        fwrite(&estimate,2,1,fp2);
    }
    else
    {
        estimate=estimate-step_16;
        fwrite(&estimate,2,1,fp2);
    }
    if(samp_value & 0x02)
    {
        estimate=estimate+step_16;
        fwrite(&estimate,2,1,fp2);
    }
    else
    {
        estimate=estimate-step_16;
        fwrite(&estimate,2,1,fp2);
    }
    if(samp_value & 0x01)
    {
        estimate=estimate+step_16;
        fwrite(&estimate,2,1,fp2);
    }
    else
    {
        estimate=estimate-step_16;
        fwrite(&estimate,2,1,fp2);
     }

}while(!feof(fp1));
}

LDDM_16stereo()
{
   fseek(fp1,0,0);
   for(byte=1;byte<=44;byte++)
    {
        char1=fgetc(fp1);
        fputc(char1,fp2);
    }
   L_estimate=0; R_estimate=0;
do
{
    fread(&samp_value,1,1,fp1);
    if(samp_value & 0x80)
    {
       L_estimate=L_estimate+step_16;
       fwrite(&L_estimate,2,1,fp2);
    }
    else
    {
       L_estimate=L_estimate-step_16;
       fwrite(&L_estimate,2,1,fp2);
    }
    if(samp_value & 0x40)
    {
       R_estimate=R_estimate+step_16;
       fwrite(&R_estimate,2,1,fp2);
    }
    else
    {
       R_estimate=R_estimate-step_16;
       fwrite(&R_estimate,2,1,fp2);
    }
    if(samp_value & 0x20)
    {
       L_estimate=L_estimate+step_16;
       fwrite(&L_estimate,2,1,fp2);
    }
    else
```

```
    {
       L_estimate=L_estimate-step_16;
       fwrite(&L_estimate,2,1,fp2);
    }
    if(samp_value & 0x10)
    {
       R_estimate=R_estimate+step_16;
       fwrite(&R_estimate,2,1,fp2);
    }
    else
    {
       R_estimate=R_estimate-step_16;
       fwrite(&R_estimate,2,1,fp2);
    }
    if(samp_value & 0x08)
    {
       L_estimate=L_estimate+step_16;
       fwrite(&L_estimate,2,1,fp2);
    }
    else
    {
       L_estimate=L_estimate-step_16;
       fwrite(&L_estimate,2,1,fp2);
    }
    if(samp_value & 0x04)
    {
       R_estimate=R_estimate+step_16;
       fwrite(&R_estimate,2,1,fp2);
    }
    else
    {
       R_estimate=R_estimate-step_16;
       fwrite(&R_estimate,2,1,fp2);
    }
    if(samp_value & 0x02)
    {
       L_estimate=L_estimate+step_16;
       fwrite(&L_estimate,2,1,fp2);
    }
    else
    {
       L_estimate=L_estimate-step_16;
       fwrite(&L_estimate,2,1,fp2);
    }
    if(samp_value & 0x01)
    {
       R_estimate=R_estimate+step_16;
       fwrite(&R_estimate,2,1,fp2);
    }
    else
    {
       R_estimate=R_estimate-step_16;
       fwrite(&R_estimate,2,1,fp2);
    }

}while(!feof(fp1));
}
```

□

ภาคผนวก ง. แสดงโปรแกรม ADT_DM.C

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<dos.h>
#include<process.h>
#include<math.h>

struct byte{
  unsigned int LSB  : 1;
  unsigned int bit2 : 1;
  unsigned int bit3 : 1;
  unsigned int bit4 : 1;
  unsigned int bit5 : 1;
  unsigned int bit6 : 1;
  unsigned int bit7 : 1;
  unsigned int MSB  : 1;
     } BYTE;
int bit_acc,byte,byte_count,DM_format,int1,
    estimate,L_estimate,R_estimate,
    samp_value,L_samp_value,R_samp_value,
    maxx,maxy,midx,midy;
int gdriver=DETECT,gmode;
int TABLE,TABLE_L,TABLE_R;
int equal_value=0,L_equal_value=0,R_equal_value=0;
int STEP,STEP_L,STEP_R,step_8,step_16;
float step,step_L,step_R;
int x=5,L_x=5,R_x,y1,y2,L_y1,L_y2,R_y1,R_y2;
FILE *fp1,*fp2;
char char1;

main()
{
char input[40],output[40];

   clrscr();

      printf("You select ADAPTIVE MODULATION process \n");
      printf("   you should give the file name in DOS structure \n");
      printf("      e.g. c:\\wavedata\\mono8.wav \n \n ");
      printf("  please enter your input file name ---> ");
       scanf("%s",input);
      printf("   and enter your output file name   ---> ");
       scanf("%s",output);
      printf("\n         And please enter the step size value");
      printf("\n              suggest   3 for  8 bits");
      printf("\n                        and 150 for 16 bits");
      printf("\n and remember this value for uses DEMODULATION process \n");
      printf("\n   enter the step size for  8 bits ---> ");
       scanf("%d",&step_8);
      printf("              step size for 16 bits ---> ");
       scanf("%d",&step_16);
      printf("\n");

      fp1 = fopen(input,"rb");
      fp2 = fopen(output,"wb");

     chkwave_file();
     chkbit_record();
     chkfile_format();
     chksampling_rate();

      printf("\n \n \n \n \n \n \n \n \n \n \n \n \n");
      printf("                    Press any key to continue.");
      getch();

  clrscr();

  initgraph(&gdriver,&gmode,"f:\\lang\\tc");
        maxx=getmaxx();
        maxy=getmaxy();
        midx=maxx/2;
        midy=maxy/2;

   ADAP_DM();

      fclose(fp1);
```

```
      fclose(fp2);

    getch();
      closegraph();
}

ADAP_DM()
{
  switch(DM_format)
   {
     case  0  : ADM_8mono();
                break;
     case  1  : ADM_16mono();
                break;
     case  2  : ADM_8stereo();
                break;
     case  3  : ADM_16stereo();
                break;
   }
}

ADM_8mono()
{
  STEP=step=step_8; TABLE=2;
  fseek(fp1,0,0);
  for(byte=1;byte<=44;byte++)
   {
      char1=fgetc(fp1);
      fputc(char1,fp2);
   }
  estimate=128;
do
{
    bit_acc+=1;                 /* account bit in the byte */
     fscanf(fp1,"%c",&samp_value);
  if(samp_value>estimate)       /* next data is greater than old data */
    {
     write_byte(1);
     ADAP(1);
     estimate=estimate+STEP;
    }
  else
    {  if(samp_value<estimate) /* next data is less than old data */
       {
         write_byte(0);
         ADAP(0);
         estimate=estimate-STEP;
       }
       else
       {
       equal_value+=1;    /* next value is equal old value */
       if(equal_value==1)
       {
         write_byte(1);
         ADAP(1);
         estimate=estimate+STEP;
       }
       else
       {
          write_byte(0);
          ADAP(0);
          estimate=estimate-STEP;
          equal_value = equal_value-2;
       }
       }
   }

if(bit_acc==8 || feof(fp1))
{
  fprintf(fp2,"%c",BYTE);
  bit_acc = 0;
}

   y1=(256-samp_value)*(maxy/256.0);
   y2=(256-estimate)*(maxy/256.0);
     putpixel(x,y1,LIGHTMAGENTA);
     putpixel(x,y2,LIGHTGREEN);
   x+=1;
   if(x==maxx-4)
   {
```

```
        cleardevice();
        x=5;
      }

}while(!feof(fp1));
}

ADM_8stereo()
{
  STEP_L=STEP_R=step_L=step_R=step_8;  TABLE_L=2; TABLE_R=2;
  L_x=5;
  R_x=midx+5;
  fseek(fp1,0,0);
  for(byte=1;byte<=44;byte++)
     {
       char1=fgetc(fp1);
       fputc(char1,fp2);
     }
  L_estimate=128; R_estimate=128;
  byte_count=0;
  bit_acc=0;
do
{
  bit_acc+=1;
  byte_count+=1;
  if(byte_count==1)
  {
      fscanf(fp1,"%c",&L_samp_value);
      if(L_samp_value>L_estimate)
      {
        write_byte(1);
        ADAP_L(1);
        L_estimate=L_estimate+STEP_L;
      }
      else
      {
      if(L_samp_value<L_estimate)
      {
        write_byte(0);
        ADAP_L(0);
        L_estimate=L_estimate-STEP_L;
      }
      else
      {
         L_equal_value+=1;
         if(L_equal_value==1)
         {
           write_byte(1);
           ADAP_L(1);
           L_estimate=L_estimate+STEP_L;
         }
         else
         {
           write_byte(0);
           ADAP_L(0);
           L_estimate=L_estimate-STEP_L;
           L_equal_value = L_equal_value-2;
         }
      }
     }
       L_y1=(256-L_samp_value)*(maxy/256.0);
       L_y2=(256-L_estimate)*(maxy/256.0);
      putpixel(L_x,L_y1,LIGHTMAGENTA);
      putpixel(L_x,L_y2,LIGHTGREEN);
       L_x+=1;
       if(L_x==midx-4)
       {
         L_x=5;
       }
  }
  else
  {
      fscanf(fp1,"%c",&R_samp_value);
      if(R_samp_value>R_estimate)
      {
        write_byte(1);
        ADAP_R(1);
        R_estimate=R_estimate+STEP_R;
      }
      else
```

```
        {
          if(R_samp_value<R_estimate)
          {
            write_byte(0);
            ADAP_R(0);
            R_estimate=R_estimate-STEP_R;
          }
          else
          {
              R_equal_value+=1;
              if(R_equal_value==1)
              {
               write_byte(1);
               ADAP_R(1);
               R_estimate=R_estimate+STEP_R;
              }
              else
              {
               write_byte(0);
               ADAP_R(0);
               R_estimate=R_estimate-STEP_R;
               R_equal_value = R_equal_value-2;
              }
          }
        }
        byte_count-=2;

        R_y1=(256-R_samp_value)*(maxy/256.0);
        R_y2=(256-R_estimate)*(maxy/256.0);
        putpixel(R_x,R_y1,LIGHTMAGENTA);
        putpixel(R_x,R_y2,LIGHTGREEN);
        R_x+=1;
        if(R_x==maxx-5)
        {
         cleardevice();
         R_x=midx+5;
        }
    }

  if(bit_acc==8 || feof(fp1))
  {
       fprintf(fp2,"%c",BYTE);
       bit_acc = 0;
  }

  }while(!feof(fp1));
  }

  ADM_16mono()
  {
    STEP=step=step_16;  TABLE=2;
    fseek(fp1,0,0);
    for(byte=1;byte<=44;byte++)
       {
         char1=fgetc(fp1);
         fputc(char1,fp2);
       }
    estimate=0;
  do
  {
    bit_acc+=1;
    fread(&samp_value,2,1,fp1);
    if(samp_value>estimate)
    {
        write_byte(1);
        ADAP(1);
        estimate=estimate+STEP;
    }
    else
    {
        if(samp_value<estimate)
        {
          write_byte(0);
          ADAP(0);
          estimate=estimate-STEP;
        }
        else
        {
          equal_value+=1;
          if(equal_value==1)
```

```
        {
          write_byte(1);
          ADAP(1);
          estimate=estimate+STEP;
        }
        else
        {
          write_byte(0);
          ADAP(0);
          estimate=estimate-STEP;
          equal_value = equal_value-2;
        }
       }
   }

if(bit_acc==8 || feof(fp1))
{
    fprintf(fp2,"%c",BYTE);
    bit_acc = 0;
}

   y1=(32768-samp_value)*(maxy/65536.0);
   y2=(32768-estimate)*(maxy/65536.0);
      putpixel(x,y1,LIGHTMAGENTA);
      putpixel(x,y2,LIGHTGREEN);
   x+=1;
   if(x==maxx-4)
   {
     cleardevice();
     x=5;
   }
}while(!feof(fp1));
}

ADM_16stereo()
{
  STEP_L=STEP_R=step_L=step_R=step_16;  TABLE_L=2; TABLE_R=2;
  L_x=5;
  R_x=midx+5;
  fseek(fp1,0,0);
  for(byte=1;byte<=44;byte++)
   {
      char1=fgetc(fp1);
      fputc(char1,fp2);
   }
  L_estimate=0; R_estimate=0;
  byte_count=0;
do
{
  bit_acc+=1;
  byte_count+=1;
  if(byte_count==1)
  {
      fread(&L_samp_value,2,1,fp1);
      if(L_samp_value>L_estimate)
      {
       write_byte(1);
       ADAP_L(1);
       L_estimate=L_estimate+STEP_L;
      }
      else
      {
       if(L_samp_value<L_estimate)
       {
           write_byte(0);
           ADAP_L(0);
           L_estimate=L_estimate-STEP_L;
       }
       else
       {
           L_equal_value+=1;
           if(L_equal_value==1)
           {
            write_byte(1);
            ADAP_L(1);
            L_estimate=L_estimate+STEP_L;
           }
           else
           {
            write_byte(0);
```

```
                ADAP_L(0);
                L_estimate=L_estimate-STEP_L;
                L_equal_value = L_equal_value-2;
                }
          }
        }
        L_y1=(32768-L_samp_value)*(maxy/65536.0);
        L_y2=(32768-L_estimate)*(maxy/65536.0);
          putpixel(L_x,L_y1,LIGHTMAGENTA);
          putpixel(L_x,L_y2,LIGHTGREEN);
        L_x+=1;
        if(L_x==midx-4)
        {
           L_x=5;
        }
     }
     else
     {
        fread(&R_samp_value,2,1,fp1);
        if(R_samp_value>R_estimate)
        {
            write_byte(1);
            ADAP_R(1);
            R_estimate=R_estimate+STEP_R;
        }
        else
        {
            if(R_samp_value<R_estimate)
            {
             write_byte(0);
             ADAP_R(0);
             R_estimate=R_estimate-STEP_R;
            }
            else
            {
              R_equal_value+=1;
              if(R_equal_value==1)
              {
                write_byte(1);
                ADAP_R(1);
                R_estimate=R_estimate+STEP_R;
              }
              else
              {
                write_byte(0);
                ADAP_R(0);
                R_estimate=R_estimate-STEP_R;
                R_equal_value = R_equal_value-2;
              }
            }
        }
         byte_count-=2;
         R_y1=(32768-R_samp_value)*(maxy/65536.0);
         R_y2=(32768-R_estimate)*(maxy/65536.0);
            putpixel(R_x,R_y1,LIGHTMAGENTA);
            putpixel(R_x,R_y2,LIGHTGREEN);
         R_x+=1;
         if(R_x==maxx-5)
         {
           cleardevice();
           R_x=midx+5;
         }
   }

   if(bit_acc==8 || feof(fp1))
   {
       fprintf(fp2,"%c",BYTE);
       bit_acc = 0;
   }

}while(!feof(fp1));
}

write_byte(int bit_value)
{

switch(bit_acc) {
 case 1 : BYTE.MSB  = bit_value; break;
 case 2 : BYTE.bit7 = bit_value; break;
 case 3 : BYTE.bit6 = bit_value; break;
```

```
 case 4 : BYTE.bit5 = bit_value; break;
 case 5 : BYTE.bit4 = bit_value; break;
 case 6 : BYTE.bit3 = bit_value; break;
 case 7 : BYTE.bit2 = bit_value; break;
 case 8 : BYTE.LSB  = bit_value; break;
             }
}

ADAP(int history)
{
  int temp;
   TABLE=TABLE << 1;
  if(history==1)
      TABLE=TABLE | 0x01;
   temp=TABLE & 0x07;
  switch(temp) {
    case 0 : step=step*1.50;
           outstep();
           break;
    case 1 : step=step*0.66;
           outstep();
           break;
    case 2 : step=step*0.66;
           outstep();
           break;
    case 3 : step=step*1.00;
           outstep();
           break;
    case 4 : step=step*1.00;
           outstep();
           break;
    case 5 : step=step*0.66;
           outstep();
           break;
    case 6 : step=step*0.66;
           outstep();
           break;
    case 7 : step=step*1.50;
           outstep();
           break;
          }
}

ADAP_L(int history)
{
  int temp;
   TABLE_L=TABLE_L << 1;
  if(history==1)
     TABLE_L=TABLE_L | 0x1;
   temp=TABLE_L & 0x7;
  switch(temp) {
    case 0 : step_L=step_L*1.50;
           outstep_L();
           break;
    case 1 : step_L=step_L*0.66;
             outstep_L();
           break;
    case 2 : step_L=step_L*0.66;
             outstep_L();
           break;
    case 3 : step_L=step_L*1.00;
             outstep_L();
           break;
    case 4 : step_L=step_L*1.00;
             outstep_L();
           break;
    case 5 : step_L=step_L*0.66;
             outstep_L();
           break;
    case 6 : step_L=step_L*0.66;
             outstep_L();
           break;
    case 7 : step_L=step_L*1.50;
             outstep_L();
           break;
          }
}

ADAP_R(int history)
{
```

```
    int temp;
    TABLE_R=TABLE_R << 1;
    if(history==1)
       TABLE_R=TABLE_R | 0x1;
      temp=TABLE_R & 0x7;
    switch(temp) {
      case 0 : step_R=step_R*1.50;
             outstep_R();
             break;
      case 1 : step_R=step_R*0.66;
               outstep_R();
             break;
      case 2 : step_R=step_R*0.66;
               outstep_R();
             break;
      case 3 : step_R=step_R*1.00;
               outstep_R();
             break;
      case 4 : step_R=step_R*1.00;
               outstep_R();
             break;
      case 5 : step_R=step_R*0.66;
               outstep_R();
             break;
      case 6 : step_R=step_R*0.66;
               outstep_R();
             break;
      case 7 : step_R=step_R*1.50;
               outstep_R();
             break;
           }
}

outstep()
{
  float x;
  int y;
    y=step/1.0;
    x=step-y;
   STEP=y;
   if(x>=0.5)
     STEP+=1;
   if(STEP==0)
    {
      STEP=1;
      step=1;
    }
}

outstep_L()
{
  float x;
  int y;
    y=step_L/1.0;
    x=step_L-y;
   STEP_L=y;
   if(x>=0.5)
     STEP_L+=1;
   if(STEP_L==0)
    {
      STEP_L=1;
      step_L=1;
    }
}

outstep_R()
{
  float x;
  int y;
    y=step_R/1.0;
    x=step_R-y;
   STEP_R=y;
   if(x>=0.5)
     STEP_R+=1;
   if(STEP_R==0)
    {
      STEP_R=1;
      step_R=1;
    }
}
```

ภาคผนวก จ. แสดงโปรแกรม ADT_DDM.C

```
#include<stdio.h>
#include<conio.h>
#include<dos.h>
#include<process.h>
#include<math.h>

int byte,DM_format,int1,maxx,maxy,midx,midy,
    estimate,L_estimate,R_estimate,
    samp_value,L_samp_value,R_samp_value;
int STEP,STEP_L,STEP_R,step_8,step_16;
float step,step_L,step_R;
int TABLE,TABLE_L,TABLE_R;
FILE *fp1,*fp2;
char char1;

main()
{
char input[40],output[40];

   clrscr();

      printf("You select ADAPTIVE DEMODULATION process \n");
      printf("   you should give the file name in DOS structure \n");
      printf("      e.g. c:\\wavedata\\mono8.wav \n \n ");
      printf("  please enter your input file name ---> ");
      scanf("%s",input);
      printf("   and enter your output file name   ---> ");
      scanf("%s",output);
      printf("\n        And please enter the step size value");
      printf("\n        <use the value in MODULATION process> \n");
      printf("\n   enter the step size for  8 bits ---> ");
       scanf("%d",&step_8);
      printf("                    step size for 16 bits ---> ");
       scanf("%d",&step_16);
      printf("\n");

   fp1 = fopen(input,"rb");
   fp2 = fopen(output,"wb");

      chkwave_file();
      chkbit_record();
      chkfile_format();
      chksampling_rate();

      printf("\n \n \n \n \n \n \n");
      printf("                    Press any key to continue. \n");
      getch();

    clrscr();

     ADAP_DDM();

    fclose(fp1);
    fclose(fp2);
}

ADAP_DDM()
{
  switch(DM_format)
   {
     case  0  : ADDM_8mono();
                break;
     case  1  : ADDM_16mono();
                break;
     case  2  : ADDM_8stereo();
                break;
     case  3  : ADDM_16stereo();
                break;
   }
}

ADDM_8mono()
{
 TABLE=2; STEP=5;  step=step_8;
 fseek(fp1,0,0);
 for(byte=1;byte<=44;byte++)
```

```
    {
      char1=fgetc(fp1);
      fputc(char1,fp2);
    }
  estimate=128;
do
{
  fread(&samp_value,1,1,fp1);
  if(samp_value & 0x80)
  {
      ADAP(1);
      estimate+=STEP;
      fprintf(fp2,"%c",estimate);
  }
  else
  {
      ADAP(0);
      estimate-=STEP;
      fprintf(fp2,"%c",estimate);
  }
  if(samp_value & 0x40)
  {
      ADAP(1);
      estimate+=STEP;
      fprintf(fp2,"%c",estimate);
  }
  else
  {
      ADAP(0);
      estimate-=STEP;
      fprintf(fp2,"%c",estimate);
  }
  if(samp_value & 0x20)
  {
      ADAP(1);
      estimate+=STEP;
      fprintf(fp2,"%c",estimate);
  }
  else
  {
      ADAP(0);
      estimate-=STEP;
      fprintf(fp2,"%c",estimate);
  }
  if(samp_value & 0x10)
  {
      ADAP(1);
      estimate+=STEP;
      fprintf(fp2,"%c",estimate);
  }
  else
  {
      ADAP(0);
      estimate-=STEP;
      fprintf(fp2,"%c",estimate);
  }
  if(samp_value & 0x08)
  {
      ADAP(1);
      estimate+=STEP;
      fprintf(fp2,"%c",estimate);
  }
  else
  {
      ADAP(0);
      estimate-=STEP;
      fprintf(fp2,"%c",estimate);
  }
  if(samp_value & 0x04)
  {
      ADAP(1);
      estimate+=STEP;
      fprintf(fp2,"%c",estimate);
  }
  else
  {
      ADAP(0);
      estimate-=STEP;
      fprintf(fp2,"%c",estimate);
  }
```

```
  if(samp_value & 0x02)
  {
      ADAP(1);
      estimate+=STEP;
      fprintf(fp2,"%c",estimate);
  }
  else
  {
      ADAP(0);
      estimate-=STEP;
      fprintf(fp2,"%c",estimate);
  }
  if(samp_value & 0x01)
  {
      ADAP(1);
      estimate+=STEP;
      fprintf(fp2,"%c",estimate);
  }
  else
  {
      ADAP(0);
      estimate-=STEP;
      fprintf(fp2,"%c",estimate);
   }

}while(!feof(fp1));
}

ADDM_8stereo()
{
  TABLE_L=2;  TABLE_R=2;  STEP_L=STEP_R=step_L=step_R=step_8;
  fseek(fp1,0,0);
  for(byte=1;byte<=44;byte++)
     {
       char1=fgetc(fp1);
       fputc(char1,fp2);
     }
  L_estimate=128; R_estimate=128;
do
{
  fread(&samp_value,1,1,fp1);
  if(samp_value & 0x80)
  {
       ADAP_L(1);
       L_estimate+=STEP_L;
       fprintf(fp2,"%c",L_estimate);
  }
  else
  {
       ADAP_L(0);
       L_estimate-=STEP_L;
       fprintf(fp2,"%c",L_estimate);
  }
  if(samp_value & 0x40)
  {
       ADAP_R(1);
       R_estimate+=STEP_R;
       fprintf(fp2,"%c",R_estimate);
  }
  else
  {
       ADAP_R(0);
       R_estimate-=STEP_R;
       fprintf(fp2,"%c",R_estimate);
  }
  if(samp_value & 0x20)
  {
       ADAP_L(1);
       L_estimate+=STEP_L;
       fprintf(fp2,"%c",L_estimate);
  }
  else
  {
       ADAP_L(0);
       L_estimate-=STEP_L;
       fprintf(fp2,"%c",L_estimate);
  }
  if(samp_value & 0x10)
  {
       ADAP_R(1);
```

```
        R_estimate+=STEP_R;
        fprintf(fp2,"%c",R_estimate);
    }
    else
    {
        ADAP_R(0);
        R_estimate-=STEP_R;
        fprintf(fp2,"%c",R_estimate);
    }
    if(samp_value & 0x08)
    {
        ADAP_L(1);
        L_estimate+=STEP_L;
        fprintf(fp2,"%c",L_estimate);
    }
    else
    {
        ADAP_L(0);
        L_estimate-=STEP_L;
        fprintf(fp2,"%c",L_estimate);
    }
    if(samp_value & 0x04)
    {
        ADAP_R(1);
        R_estimate+=STEP_R;
        fprintf(fp2,"%c",R_estimate);
    }
    else
    {
        ADAP_R(0);
        R_estimate-=STEP_R;
        fprintf(fp2,"%c",R_estimate);
    }
    if(samp_value & 0x02)
    {
        ADAP_L(1);
        L_estimate+=STEP_L;
        fprintf(fp2,"%c",L_estimate);
    }
    else
    {
        ADAP_L(0);
        L_estimate-=STEP_L;
        fprintf(fp2,"%c",L_estimate);
    }
    if(samp_value & 0x01)
    {
        ADAP_R(1);
        R_estimate+=STEP_R;
        fprintf(fp2,"%c",R_estimate);
    }
    else
    {
        ADAP_R(0);
        R_estimate-=STEP_R;
        fprintf(fp2,"%c",R_estimate);
    }

}while(!feof(fp1));
}

ADDM_16mono()
{
  TABLE=2;  STEP=150;  step=step_16;
  fseek(fp1,0,0);
  for(byte=1;byte<=44;byte++)
     {
       char1=fgetc(fp1);
       fputc(char1,fp2);
     }
  estimate=0;
do
{
  fread(&samp_value,1,1,fp1);
  if(samp_value & 0x80)
  {
        ADAP(1);
      estimate+=STEP;
      fwrite(&estimate,2,1,fp2);
  }
```

```
else
{
    ADAP(0);
    estimate-=STEP;
    fwrite(&estimate,2,1,fp2);
}
if(samp_value & 0x40)
{
      ADAP(1);
    estimate+=STEP;
    fwrite(&estimate,2,1,fp2);
}
else
{
    ADAP(0);
    estimate-=STEP;
    fwrite(&estimate,2,1,fp2);
}
if(samp_value & 0x20)
{
      ADAP(1);
    estimate+=STEP;
    fwrite(&estimate,2,1,fp2);
}
else
{
    ADAP(0);
    estimate-=STEP;
    fwrite(&estimate,2,1,fp2);
}
if(samp_value & 0x10)
{
      ADAP(1);
    estimate+=STEP;
    fwrite(&estimate,2,1,fp2);
}
else
{
    ADAP(0);
    estimate-=STEP;
    fwrite(&estimate,2,1,fp2);
}
if(samp_value & 0x08)
{
      ADAP(1);
    estimate+=STEP;
    fwrite(&estimate,2,1,fp2);
}
else
{
    ADAP(0);
    estimate-=STEP;
    fwrite(&estimate,2,1,fp2);
}
if(samp_value & 0x04)
{
      ADAP(1);
    estimate+=STEP;
    fwrite(&estimate,2,1,fp2);
}
else
{
    ADAP(0);
    estimate-=STEP;
    fwrite(&estimate,2,1,fp2);
}
if(samp_value & 0x02)
{
      ADAP(1);
    estimate+=STEP;
    fwrite(&estimate,2,1,fp2);
}
else
{
    ADAP(0);
    estimate-=STEP;
    fwrite(&estimate,2,1,fp2);
}
if(samp_value & 0x01)
{
```

```
            ADAP(1);
          estimate+=STEP;
          fwrite(&estimate,2,1,fp2);
      }
      else
      {
          ADAP(0);
          estimate-=STEP;
          fwrite(&estimate,2,1,fp2);
       }

    }while(!feof(fp1));
    }

    ADDM_16stereo()
    {
      TABLE_L=2;  TABLE_R=2;  STEP_L=STEP_R=step_L=step_R=step_16;
      fseek(fp1,0,0);
      for(byte=1;byte<=44;byte++)
       {
          char1=fgetc(fp1);
          fputc(char1,fp2);
       }
      L_estimate=0; R_estimate=0;
    do
    {
       fread(&samp_value,1,1,fp1);
       if(samp_value & 0x80)
       {
            ADAP_L(1);
          L_estimate+=STEP_L;
          fwrite(&L_estimate,2,1,fp2);
       }
       else
       {
          ADAP_L(0);
          L_estimate-=STEP_L;
          fwrite(&L_estimate,2,1,fp2);
       }
       if(samp_value & 0x40)
       {
          ADAP_R(1);
          R_estimate+=STEP_R;
          fwrite(&R_estimate,2,1,fp2);
       }
       else
       {
          ADAP_R(0);
          R_estimate-=STEP_R;
          fwrite(&R_estimate,2,1,fp2);
       }
       if(samp_value & 0x20)
       {
            ADAP_L(1);
          L_estimate+=STEP_L;
          fwrite(&L_estimate,2,1,fp2);
       }
       else
       {
          ADAP_L(0);
          L_estimate-=STEP_L;
          fwrite(&L_estimate,2,1,fp2);
       }
       if(samp_value & 0x10)
       {
          ADAP_R(1);
          R_estimate+=STEP_R;
          fwrite(&R_estimate,2,1,fp2);
       }
       else
       {
          ADAP_R(0);
          R_estimate-=STEP_R;
          fwrite(&R_estimate,2,1,fp2);
       }
       if(samp_value & 0x08)
       {
            ADAP_L(1);
          L_estimate+=STEP_L;
          fwrite(&L_estimate,2,1,fp2);
```

```
    }
    else
    {
       ADAP_L(0);
       L_estimate-=STEP_L;
       fwrite(&L_estimate,2,1,fp2);
    }
    if(samp_value & 0x04)
    {
       ADAP_R(1);
       R_estimate+=STEP_R;
       fwrite(&R_estimate,2,1,fp2);
    }
    else
    {
       ADAP_R(0);
       R_estimate-=STEP_R;
       fwrite(&R_estimate,2,1,fp2);
    }
    if(samp_value & 0x02)
    {
        ADAP_L(1);
       L_estimate+=STEP_L;
       fwrite(&L_estimate,2,1,fp2);
    }
    else
    {
       ADAP_L(0);
       L_estimate-=STEP_L;
       fwrite(&L_estimate,2,1,fp2);
    }
    if(samp_value & 0x01)
    {
       ADAP_R(1);
       R_estimate+=STEP_R;
       fwrite(&R_estimate,2,1,fp2);
    }
    else
    {
       ADAP_R(0);
       R_estimate-=STEP_R;
       fwrite(&R_estimate,2,1,fp2);
    }

}while(!feof(fp1));
}

ADAP(int history)
{
  int temp;
   TABLE=TABLE << 1;
  if(history==1)
      TABLE=TABLE | 0x01;
   temp=TABLE & 0x07;
  switch(temp) {
    case 0 : step=step*1.50;
           outstep();
           break;
    case 1 : step=step*0.66;
           outstep();
           break;
    case 2 : step=step*0.66;
           outstep();
           break;
    case 3 : step=step*1.00;
           outstep();
           break;
    case 4 : step=step*1.00;
           outstep();
           break;
    case 5 : step=step*0.66;
           outstep();
           break;
    case 6 : step=step*0.66;
           outstep();
           break;
    case 7 : step=step*1.50;
           outstep();
           break;
          }
```

```
}

ADAP_L(int history)
{
  int temp;
   TABLE_L=TABLE_L << 1;
  if(history==1)
     TABLE_L=TABLE_L | 0x1;
   temp=TABLE_L & 0x7;
  switch(temp) {
    case 0 : step_L=step_L*1.50;
           outstep_L();
           break;
    case 1 : step_L=step_L*0.66;
             outstep_L();
           break;
    case 2 : step_L=step_L*0.66;
             outstep_L();
           break;
    case 3 : step_L=step_L*1.00;
             outstep_L();
           break;
    case 4 : step_L=step_L*1.00;
             outstep_L();
           break;
    case 5 : step_L=step_L*0.66;
             outstep_L();
           break;
    case 6 : step_L=step_L*0.66;
             outstep_L();
           break;
    case 7 : step_L=step_L*1.50;
             outstep_L();
           break;
          }
}

ADAP_R(int history)
{
  int temp;
  TABLE_R=TABLE_R << 1;
  if(history==1)
     TABLE_R=TABLE_R | 0x1;
    temp=TABLE_R & 0x7;
  switch(temp) {
    case 0 : step_R=step_R*1.50;
           outstep_R();
           break;
    case 1 : step_R=step_R*0.66;
             outstep_R();
           break;
    case 2 : step_R=step_R*0.66;
             outstep_R();
           break;
    case 3 : step_R=step_R*1.00;
             outstep_R();
           break;
    case 4 : step_R=step_R*1.00;
             outstep_R();
           break;
    case 5 : step_R=step_R*0.66;
             outstep_R();
           break;
    case 6 : step_R=step_R*0.66;
             outstep_R();
           break;
    case 7 : step_R=step_R*1.50;
             outstep_R();
           break;
          }
}

outstep()
{
  float x;
  int y;
    y=step/1.0;
    x=step-y;
   STEP=y;
   if(x>=0.5)
```

```
      STEP+=1;
    if(STEP==0)
     {
       STEP=1;
       step=1;
     }
}

outstep_L()
{
  float x;
  int y;
    y=step_L/1.0;
    x=step_L-y;
   STEP_L=y;
   if(x>=0.5)
     STEP_L+=1;
   if(STEP_L==0)
     {
       STEP_L=1;
       step_L=1;
     }
}

outstep_R()
{
  float x;
  int y;
    y=step_R/1.0;
    x=step_R-y;
   STEP_R=y;
   if(x>=0.5)
     STEP_R+=1;
   if(STEP_R==0)
     {
       STEP_R=1;
       step_R=1;
     }
}

□
```

ภาคผนวก ฉ. แสดงโปรแกรม STEPSIZE.C

```
#include "stdio.h"
#include "stdlib.h"
#include "conio.h"
#include "math.h"
double    sqrt();
double    fstep,fstep2;
void     filetype(),strcpy();

main()
{
 char  ch_m1,ch_r1,ch_b1,filename[40];
 int   choice,mode,bit,xsamp;

 FILE  *fp;

 printf("\nPROGRAM FOR CALCULATE SQNR FOR LDM & ADM");
 printf("\nEnter file name(CAPITAL CHARACTER):");
 scanf("%s",filename);

        if ((fp = fopen(filename,"rb")) == NULL )
          {
           printf("error in opening file\n");
           getche();
           exit(1);
          }

        clrscr();

fseek(fp,22,0);
ch_m1 = fgetc(fp);
mode  = (int)ch_m1;
fseek(fp,24,0);
ch_r1 = fgetc(fp);
fseek(fp,34,0);
ch_b1 = fgetc(fp);
bit   = (int)ch_b1;

printf("\n%s",filename);
filetype(ch_m1,ch_r1,ch_b1);

  printf("\n  This is a list of COMPANDING  METHOD\n");
  printf(" - DELTA  MODULATION(DM)          \n");
  printf("     [1]   : LDM                  \n");
  printf(" - ADAPTIVE DELTA MODULATION(ADM)\n");
  printf("     [2]   : CFDM                 \n");
  printf("     [3]   : CVSD(mono 8 bit only)\n");

do
  {
  printf("\nENTER  CHOICE  FOR  COMPANDING  METHOD :");
  scanf("%d",&choice);
  if ( choice < 1 || choice > 3 )
      printf("please try again !\n");
  }
while ( choice < 1 || choice > 3 );

printf("\nPROGRAM TO CALCULATE STEPSIZE AND SQNR\n");
printf("Number of samples : ");
scanf("%d",&xsamp);

switch(choice)
   {
     case 1 :  printf("delta mod\n");
               dm(mode,bit,filename,xsamp);
               break;
     case 2 :  printf("cfdm\n");
               cfdm2(mode,bit,filename,xsamp);
               break;
     case 3 :  printf("cvsd\n");
               cvsd(mode,bit,filename,xsamp);
               break;
   }

     if ( mode == 1 )
          printf("\nTHE BEST STEPSIZE IS : %f",fstep);
     else
```

```
            {
             printf("\nCH1:BEST STEPSIZE IS : %f",fstep);
             printf("\nCH2:BEST STEPSIZE IS : %f",fstep2);
            }
 printf("\007");
 getche();
 fclose(fp);
}

/* DELTA MODULATION */
dm(int xmode,int xbit,char f_name[40],int msamp)
{
  char f_name2[40];
  int  sample;

     sample = msamp;
     strcpy(f_name2,f_name);
   if ( xmode == 1 && xbit == 8 )
      {
       printf("DM mono 8 bit\n");
       printf("%s",f_name2);
       dm8mono(f_name2,sample);
      }
   else
      {
       if ( xmode == 1 && xbit == 16 )
          {
           printf("DM mono 16 bit\n");
           printf("%s",f_name2);
           dm16mono(f_name2,sample);
          }
        else
           {
            if ( xmode == 2 && xbit == 8 )
               {
                printf("DM stereo 8 bit\n");
                printf("%s",f_name2);
                dm8ster(f_name2,sample);
               }
              else
                 {
                  if ( xmode == 2 && xbit == 16 )
                     {
                      printf("DM stereo 16 bit\n");
                      printf("%s",f_name2);
                      dm16ster(f_name2,sample);
                     }
                    else
                      printf("ERROR can't be classified");
                 }
           }
      }
 }

/* CFDM 2   */
cfdm2(int xmode,int xbit,char f_name[40],int nsamp)
{
char  f_name2[40];
int   sample;

sample = nsamp;
strcpy(f_name2,f_name);
if ( xmode == 1 && xbit == 8 )
   {
       printf("CFDM2 mono 8 bit\n");
       printf("%s",f_name2);
       cf28mono(f_name2,sample);
   }
   else
      {
       if ( xmode == 1 && xbit == 16 )
          {
           printf("CFDM2 mono 16 bit\n");
           printf("%s",f_name2);
           cf216mono(f_name2,sample);
          }
         else
            {
             if ( xmode == 2 && xbit == 8 )
                {
```

```
                printf("CFDM2 stereo 8 bit\n");
                printf("%s",f_name2);
                cf28ster(f_name2,sample);
               }
            else
               {
                if ( xmode == 2 && xbit == 16 )
                   {
                   printf("CFDM2 stereo 16 bit\n");
                   printf("%s",f_name2);
                   cf216ster(f_name2,sample);
                   }
                else
                   printf("ERROR can't be classified");
               }
            }
        }
    }

     /*    CVSD      */
    cvsd(int xmode,int xbit,char f_name[40],int nsamp)
    {
    char f_name2[40];
    int  sample;

       sample = nsamp;
       strcpy(f_name2,f_name);

         if ( xmode == 1 && xbit == 8 )
       {
          printf("CVSD mono 8 bit\n");
          printf("%s",f_name2);
          cvsmono(f_name2,sample);
       }
       else
          printf("ERROR can't be classified");
      }

    /* TEST FILE TYPE */
    void filetype(ch_m,ch_r,ch_b)
    char  ch_m,ch_r,ch_b;
    {
        int   fmode,frate,fbit;

      fmode = (int)ch_m;

      switch(fmode)
         {
           case 1 : printf("\nThis is MONO mode file\n");
                    break;
           case 2 : printf("\nThis is STEREO mode file\n");
                    break;
         }

      frate = (int)ch_r;

      switch(frate)
         {
           case 17 : printf("Sampling rate = 11,025 Hz\n");
                     break;
           case 34 : printf("Sampling rate = 22,050 Hz\n");
                     break;
           case 68 : printf("Sampling rate = 44,100 Hz\n");
         }

      fbit = (int)ch_b;

      switch(fbit)
         {
           case 8  : printf("This is  8 bit PCM file\n");
                     break;
           case 16 : printf("This is 16 bit PCM file\n");
                     break;
         }
        printf("\nEND OF FILE TYPE TESTING PROGRAM(press any key)\n");
        getche();
}

  /* DM 8 BIT MONO */
 dm8mono(char f_name3[40],int dsamp)
```

```
{
        char  ch,ch_adv,cp;
        int   i,j,n,m,m_adv,diff,sample,cp_int,samp;
        float sq_diff,sum_sq,approx,s_diff,sq_s_dif,sq_diff2,sum_sig,
              sum_dif,sf_max;
       double total,stepsize,b_step,sqnr,sqdb,sqdb_max;

        FILE  *fp;

          if ((fp = fopen(f_name3,"r+")) == NULL )
           {
             printf("error in function\n");
             exit(1);
           }

         samp = dsamp;

          fseek(fp,44,0);
          ch = getc(fp);
          m  = conv8(ch);

             sum_sq = 0;
             sample = 0;

             for ( n = 43 ; n <= samp + 42 ; n++ )
                   {
                        ch_adv = getc(fp);
                        m_adv  = conv8(ch_adv);

                        diff   = m_adv - m;
                        sq_diff= diff*diff;
                        sum_sq = sum_sq + sq_diff;
                    sample   = sample + 1;
                        m      = m_adv;
                   }
                        total  = sum_sq / sample;
                        b_step  = sqrt(total);

printf("\nTotal sample = %i Basic stepsize = %f",sample,b_step);

  sqdb_max = 0;
  sf_max   = 0;

 for ( j = 1 ; j <= 23 ; j++ )
   {
                   stepsize  = 0.1*j*sqrt(total);

         rewind(fp);
         for ( i=0 ; i<44 ; i++ )
                {
                  ch = getc(fp);
                }

         approx = 0x80;
             cp = getc(fp);
         cp_int = conv8(cp);

                 sum_sig = 0;
                 sum_dif = 0;
         for ( i = 44 ; i < samp + 42 ; i++ )
           {
             if ( cp_int > approx )
                   {
                        approx = approx + stepsize;
                        s_diff = cp_int - approx;
                    sq_s_dif = s_diff * s_diff;
                    sq_diff2 = cp_int * cp_int;
                     sum_sig = sum_sig + sq_diff2;
                     sum_dif = sum_dif + sq_s_dif;
                   }
             else
                   {
                        approx = approx - stepsize;
                        s_diff = approx - cp_int;
                    sq_s_dif = s_diff * s_diff;
                    sq_diff2 = cp_int * cp_int;
                     sum_sig = sum_sig + sq_diff2;
                     sum_dif = sum_dif + sq_s_dif;
                   }
```

```
                    cp = getc(fp);
                 cp_int = conv8(cp);

               }
              sqnr = sum_sig / sum_dif;
              sqdb = 10*log10(sqnr);
printf("\nScale factor =%2.2f SQNR =%f or %f dB.",j*0.1,sqnr,sqdb);

        if  ( sqdb > sqdb_max )
         {
         sqdb_max = sqdb;
         sf_max   = 0.1*j;
         }

          }
 printf("\nTHE BEST STEP SIZE IS : %f SCALE FACTOR = %f",
                                        sf_max*b_step,sf_max);

        fstep = sf_max*b_step;
            fclose(fp);
          }

/*  DM 16 BIT MONO    */
dm16mono(char f_name4[40],int dsamp)
{
     char  ch1,ch2,ch_adv1,ch_adv2,cp1,cp2;
     int   i,j,n,m,m_adv,diff,sample,cp_int,samp;
     float sq_diff,sum_sq,approx,s_diff,sq_s_dif,sq_diff2,sum_sig,
           sum_dif,sf_max;
     double total,stepsize,b_step,sqnr,sqdb,sqdb_max;

     FILE  *fp;

          if ((fp = fopen(f_name4,"r+")) == NULL )
          {
          printf("error 16 bit mono\n");
          getche();
          exit(1);
          }

         samp = dsamp;

          fseek(fp,44,0);

                ch1 = fgetc(fp);
                ch2 = fgetc(fp);
                m   = conv16(ch2,ch1);

            sum_sq = 0;
            sample = 0;

            for ( n = 43 ; n <= samp+42 ; n++ )
                 {
                      ch_adv1 = fgetc(fp);
                      ch_adv2 = fgetc(fp);
                      m_adv   = conv16(ch_adv2,ch_adv1);

                      diff   = m_adv - m;
                      sq_diff= diff*diff;
                      sum_sq = sum_sq + sq_diff;
                  sample   = sample + 1;
                      m    = m_adv;
                 }
                     total = sum_sq / sample;
                  b_step  = sqrt(total);
printf("\nTotal sample = %i  Basic stepsize = %f",sample,b_step);

        sqdb_max = -100;
        sf_max   = 0;

        for ( j = 655 ; j <= 678 ; j++ )
         {
              stepsize = j * b_step;

              rewind(fp);
              fseek(fp,44,0);
              approx = 0;

                  cp1 = fgetc(fp);
                  cp2 = fgetc(fp);
```

```
            cp_int = conv16(cp2,cp1);

                  sum_sig = 0;
                  sum_dif = 0;

            for ( i = 44 ; i < samp + 42 ; i++ )
              {
                if ( cp_int > approx )
                      {
                           approx = approx + stepsize;
                           s_diff = cp_int - approx;
                      sq_s_dif = s_diff * s_diff;
                      sq_diff2 = cp_int * cp_int;
                       sum_sig = sum_sig + sq_diff2;
                       sum_dif = sum_dif + sq_s_dif;
                      }
                else
                      {
                           approx = approx - stepsize;
                           s_diff = approx - cp_int;
                      sq_s_dif = s_diff * s_diff;
                      sq_diff2 = cp_int * cp_int;
                       sum_sig = sum_sig + sq_diff2;
                       sum_dif = sum_dif + sq_s_dif;
                      }

                 cp1 = fgetc(fp);
                cp2 = fgetc(fp);
             cp_int = conv16(cp2,cp1);


              }
             sqnr = sum_sig / sum_dif;
             sqdb = 10*log10(sqnr);
printf("\nScale factor =%4.1i SQNR =%2.8f or %f dB.",j,sqnr,sqdb);
          if ( sqdb > sqdb_max )
            {
             sqdb_max = sqdb;
              sf_max = j;
            }
   }
printf("\nTHE BEST CONDITION WHEN SCALE FACTOR = %f STEPSIZE IS = %f",
                                          sf_max,sf_max*b_step);
            fclose(fp);
           fstep = sf_max * b_step;
           }

/*  DM  8 BIT STEREO  */
dm8ster(char f_name5[40],int dsamp)
{
     char  ch,ch_adv,cp;
     int   i,y,j,n,m,m_adv,diff,sample,cp_int,samp;
     float sq_diff,sum_sq,approx,s_diff,sq_s_dif,sq_diff2,sum_sig,
           sum_dif,sf_max1,sf_max2;
     double total,stepsiz1,stepsiz2,stepsize,sqnr1,sqnr2,sqdb1,sqdb2,
            sqdb_mx1,sqdb_mx2;

     FILE  *fp;

            if ((fp = fopen(f_name5,"r+")) == NULL )
            {
            printf("error 8 bit stereo\n");
            getche();
            exit(1);
            }

        samp = dsamp;

 /*    CHANNEL  1   */

     fseek(fp,44,0);
            ch = getc(fp);
            m  = conv8(ch);

               sum_sq = 0;
               sample = 0;

                    y = 1;
               for ( n = 43 ; n <= samp + 42 ; n++ )
                    {
```

```
                        fseek(fp,44+(2*y),0);
                          ch_adv = getc(fp);
                          m_adv  = conv8(ch_adv);

                          diff   = m_adv - m;
                          sq_diff= diff*diff;
                          sum_sq = sum_sq + sq_diff;
                      sample   = sample + 1;
                          m    = m_adv;
                          y    = y + 1;
                     }
                          total  = sum_sq / sample;
                    stepsiz1  = sqrt(total);
printf("\nCH1:Total sample = %i Basic stepsize = %f\n",sample,stepsiz1);

          sqdb_mx1 = 0;
          sf_max1  = 0;

     for ( j = 1 ; j <= 30 ; j++ )
       {
                    stepsize  = 0.1*j*stepsiz1;

          rewind(fp);
          for ( i=0 ; i<44 ; i++ )
                  {
                    ch = getc(fp);
                  }

          approx = 0x80;
              cp = getc(fp);
          cp_int = conv8(cp);

                  sum_sig = 0;
                  sum_dif = 0;
                        y  = 1;

          for ( i = 44 ; i < samp + 42 ; i++ )
            {
              if ( cp_int > approx )
                  {
                      approx = approx + stepsize;
                      s_diff = cp_int - approx;
                    sq_s_dif = s_diff * s_diff;
                    sq_diff2 = cp_int * cp_int;
                     sum_sig = sum_sig + sq_diff2;
                     sum_dif = sum_dif + sq_s_dif;
                  }
               else
                  {
                      approx = approx - stepsize;
                      s_diff = approx - cp_int;
                    sq_s_dif = s_diff * s_diff;
                    sq_diff2 = cp_int * cp_int;
                     sum_sig = sum_sig + sq_diff2;
                     sum_dif = sum_dif + sq_s_dif;
                  }

                fseek(fp,44+(2*y),0);
                   cp = getc(fp);
                cp_int = conv8(cp);
                    y = y + 1;
            }
           sqnr1 = sum_sig / sum_dif;
           sqdb1 = 10*log10(sqnr1);
printf("\nCH1: Scale factor =%2.2f SQNR =%f or %f dB.",j*0.1,sqnr1,sqdb1);
       if ( sqdb1 > sqdb_mx1 )
          {
           sqdb_mx1 = sqdb1;
           sf_max1  = 0.1 * j;
          }
    }
   printf("\nTHE BEST CONDITION WHEN SCALE FACTOR = %f STEPSIZE IS = %f",
                                          sf_max1,sf_max1*stepsiz1);
          fstep = sf_max1 * stepsiz1;
          printf("\007");
          getche();

  /*  CHANNEL  2  */

          fseek(fp,45,0);
```

```
            ch = getc(fp);
            m  = conv8(ch);

               sum_sq = 0;
               sample = 0;

                    y = 1;
               for ( n = 43 ; n <= samp + 42 ; n++ )
                     {
                        fseek(fp,45+(2*y),0);
                        ch_adv = getc(fp);
                        m_adv  = conv8(ch_adv);

                        diff   = m_adv - m;
                        sq_diff= diff*diff;
                        sum_sq = sum_sq + sq_diff;
                     sample   = sample + 1;
                        m    = m_adv;
                        y    = y + 1;
                     }
                        total  = sum_sq / sample;
                      stepsiz2  = sqrt(total);
printf("\nCH2:Total sample = %i Basic stepsize = %f",sample,stepsiz2);

         sqdb_mx2 = 0;
         sf_max2  = 0;

      for ( j = 1 ; j <= 30 ; j++ )
        {
                     stepsize  = 0.1*j*stepsiz2;

            rewind(fp);
            for ( i=0 ; i<45 ; i++ )
                  {
                    ch = getc(fp);
                  }

            approx = 0x80;
                cp = getc(fp);
            cp_int = conv8(cp);

                  sum_sig = 0;
                  sum_dif = 0;
                        y  = 1;

            for ( i = 44 ; i < samp + 42 ; i++ )
              {
               if ( cp_int > approx )
                     {
                          approx = approx + stepsize;
                          s_diff = cp_int - approx;
                      sq_s_dif = s_diff * s_diff;
                      sq_diff2 = cp_int * cp_int;
                       sum_sig = sum_sig + sq_diff2;
                       sum_dif = sum_dif + sq_s_dif;
                     }
                else
                     {
                          approx = approx - stepsize;
                          s_diff = approx - cp_int;
                      sq_s_dif = s_diff * s_diff;
                      sq_diff2 = cp_int * cp_int;
                       sum_sig = sum_sig + sq_diff2;
                       sum_dif = sum_dif + sq_s_dif;
                     }

                  fseek(fp,45+(2*y),0);
                      cp = getc(fp);
                  cp_int = conv8(cp);
                       y = y + 1;
              }
             sqnr2 = sum_sig / sum_dif;
             sqdb2 = 10*log10(sqnr2);
printf("\nCH2: Scale factor =%2.2f SQNR =%f or %f dB.",j*0.1,sqnr2,sqdb2);

          if ( sqdb2 > sqdb_mx2 )
           {
            sqdb_mx2 = sqdb2;
            sf_max2  = 0.1 * j;
           }
```

```
    }
    printf("\nTHE BEST CONDITION WHEN SCALE FACTOR = %f STEPSIZE IS = %f",
                                              sf_max2,sf_max2*stepsiz2);
            fstep2 = sf_max2 * stepsiz2;
            fclose(fp);
           }

 /*  DM 16 BIT STEREO  */
dm16ster(char f_name6[40],int dsamp)
{
    char  ch1,ch2,ch_adv1,ch_adv2,cp1,cp2;
    int   i,j,y,n,m,m_adv,diff,sample,cp_int,samp;
    float sq_diff,sum_sq,approx,s_diff,sq_s_dif,sq_diff2,sum_sig,
          sum_dif,sf_max1,sf_max2;
    double total,stepsiz1,stepsiz2,stepsize,sqnr1,sqnr2,sqdb1,sqdb2,
           sqdb_mx1,sqdb_mx2;

    FILE  *fp;

           if ((fp = fopen(f_name6,"r+")) == NULL )
           {
            printf("error 16 bit stereo\n");
            getche();
            exit(1);
           }

       samp = dsamp;

     /* CHANNEL 1 :CALCULATE BASIC STEPSIZE  */

           fseek(fp,44,0);

                  ch1 = fgetc(fp);
                  ch2 = fgetc(fp);
                  m   = conv16(ch2,ch1);

             sum_sq = 0;
             sample = 0;

                 y = 1;
             for ( n = 43 ; n <= samp+42 ; n++ )
                  {
                    fseek(fp,44+(4*y),0);
                      ch_adv1 = fgetc(fp);
                      ch_adv2 = fgetc(fp);
                      m_adv   = conv16(ch_adv2,ch_adv1);

                      diff    = m_adv - m;
                      sq_diff= diff*diff;
                      sum_sq = sum_sq + sq_diff;
                   sample   = sample + 1;
                      m    = m_adv;
                      y    = y + 1;
                  }
                     total = sum_sq / sample;
                  stepsiz1  = sqrt(total);
            printf("\nCH1: sample = %i total = %f  ",sample,total);
            printf("stepsize = %f\n",stepsiz1);

    /* CHANNEL 1 : CALCULATE SQNR */

          sqdb_mx1 = -100;
          sf_max1  = 0;

       for ( j = 230 ; j <= 254 ; j++ )
        {
          stepsize = j * stepsiz1;

          rewind(fp);
          fseek(fp,44,0);

          approx = 0;

              cp1 = fgetc(fp);
              cp2 = fgetc(fp);
           cp_int = conv16(cp2,cp1);

                 sum_sig = 0;
                 sum_dif = 0;
                       y = 1;
```

```
            for ( i = 44 ; i < samp + 42 ; i++ )
              {
                if ( cp_int > approx )
                      {
                           approx = approx + stepsize;
                           s_diff = cp_int - approx;
                         sq_s_dif = s_diff * s_diff;
                         sq_diff2 = cp_int * cp_int;
                          sum_sig = sum_sig + sq_diff2;
                          sum_dif = sum_dif + sq_s_dif;
                         }
                else
                      {
                           approx = approx - stepsize;
                           s_diff = approx - cp_int;
                         sq_s_dif = s_diff * s_diff;
                         sq_diff2 = cp_int * cp_int;
                          sum_sig = sum_sig + sq_diff2;
                          sum_dif = sum_dif + sq_s_dif;
                         }

               fseek(fp,44+(4*y),0);
                   cp1 = fgetc(fp);
                  cp2 = fgetc(fp);
               cp_int = conv16(cp2,cp1);
                    y = y + 1;

               }

              sqnr1 = sum_sig / sum_dif;
              sqdb1 = 10*log10(sqnr1);
   printf("\nCH1:Scale factor =%i SQNR =%2.8f or %f dB.",j,sqnr1,sqdb1);

          if ( sqdb1 > sqdb_mx1 )
           {
            sqdb_mx1 = sqdb1;
            sf_max1  = j;
           }
     }
printf("\nTHE BEST CONDITION WHEN SCALE FACTOR = %f STEPSIZE IS = %f",
                                          sf_max1, sf_max1*stepsiz1);

              fstep = sf_max1 * stepsiz1;
              printf("\007");
              getche();

   /*   CHANNEL 2 : CALCULATE STEPSIZE & SQNR   */

              fseek(fp,46,0);
                      ch1 = fgetc(fp);
                      ch2 = fgetc(fp);
                      m   = conv16(ch2,ch1);

                sum_sq = 0;
                sample = 0;

                     y = 1;
                for ( n = 43 ; n <= samp+42 ; n++ )
                      {
                        fseek(fp,46+(4*y),0);
                            ch_adv1 = fgetc(fp);
                            ch_adv2 = fgetc(fp);
                            m_adv   = conv16(ch_adv2,ch_adv1);

                            diff    = m_adv - m;
                            sq_diff= diff*diff;
                            sum_sq = sum_sq + sq_diff;
                         sample   = sample + 1;
                              m   = m_adv;
                              y   = y + 1;
                       }
                           total = sum_sq / sample;
                       stepsiz2  = sqrt(total);
                 printf("\nCH2: sample = %i total = %f  ",sample,total);
                 printf("stepsize = %f\n",stepsiz2);

     /* CHANNEL 2 : CALCULATE  SQNR */

          sqdb_mx2 = -100;
```

```
        sf_max2  = 0;

      for ( j = 230 ; j <= 254 ; j++ )
       {
            stepsize = j * stepsiz2;

            rewind(fp);
            fseek(fp,46,0);

            approx = 0;

                cp1 = fgetc(fp);
                cp2 = fgetc(fp);
             cp_int = conv16(cp2,cp1);

                   sum_sig = 0;
                   sum_dif = 0;
                         y = 1;

            for ( i = 44 ; i < samp + 42 ; i++ )
              {
                if ( cp_int > approx )
                      {
                           approx = approx + stepsize;
                           s_diff = cp_int - approx;
                        sq_s_dif = s_diff * s_diff;
                        sq_diff2 = cp_int * cp_int;
                         sum_sig = sum_sig + sq_diff2;
                         sum_dif = sum_dif + sq_s_dif;
                        }
                else
                        {
                           approx = approx - stepsize;
                           s_diff = approx - cp_int;
                        sq_s_dif = s_diff * s_diff;
                        sq_diff2 = cp_int * cp_int;
                         sum_sig = sum_sig + sq_diff2;
                         sum_dif = sum_dif + sq_s_dif;
                        }

               fseek(fp,46+(4*y),0);
                 cp1 = fgetc(fp);
                cp2 = fgetc(fp);
              cp_int = conv16(cp2,cp1);
                   y = y + 1;

               }

               sqnr2 = sum_sig / sum_dif;
               sqdb2 = 10*log10(sqnr2);
 printf("\nCH2:Scale factor =%i SQNR =%2.8f or %f dB.",j,sqnr2,sqdb2);

          if ( sqdb2 > sqdb_mx2 )
           {
             sqdb_mx2 = sqdb2;
             sf_max2  = j;
           }
     }
     printf("\nTHE BEST CONDITION WHEN SCALE FACTOR = %f STEPSIZE IS = %f",
                                              sf_max2,sf_max2*stepsiz2);

               fstep2 = sf_max2 * stepsiz2;
               fclose(fp);
            }

/*  CFDM2  8 BIT MONO  */
cf28mono(char f_name3[40],int csamp)
{
     char  ch,ch_adv,cp;
     int   i,j,n,samp,m,m_adv,diff,sample,cp_int,b1,b2,b3;
     float sq_diff,sum_sq,approx,s_diff,sq_s_dif,sq_diff2,sum_sig,
           sum_dif,sf_max;
     double total,b_step,stepsize,sqnr,sqdb,sqdb_max;

     FILE  *fp;

            if ((fp = fopen(f_name3,"r+")) == NULL )
            {
             printf("error cfdm 8 bit mono\n");
             getche();
```

```
            exit(1);
           }

      samp = csamp;

          fseek(fp,44,0);

                  ch = getc(fp);
                  m  = conv8(ch);
             sum_sq = 0;
             sample = 0;

             for ( n = 43 ; n <= samp + 42 ; n++ )
                   {
                        ch_adv = getc(fp);
                        m_adv  = conv8(ch_adv);

                        diff   = m_adv - m;
                        sq_diff= diff*diff;
                        sum_sq = sum_sq + sq_diff;
                    sample   = sample + 1;
                        m      = m_adv;
                   }
                        total  = sum_sq / sample;
                        b_step = sqrt(total);
printf("\nTotal sample = %i Basic stepsize = %f",sample,b_step);

      sqdb_max = 0;
      sf_max   = 0;

 for ( j = 1 ; j <= 24 ; j++ )
   {
                  stepsize  = 0.1*j*b_step;

          rewind(fp);
          fseek(fp,44,0);

          approx = 0x80;
              cp = getc(fp);
          cp_int = conv8(cp);
             b1  = 0;

                sum_sig = 0;
                sum_dif = 0;

           if ( cp_int > approx )
                {
                     approx = approx + stepsize;
                     s_diff = cp_int - approx;
                  sq_s_dif = s_diff * s_diff;
                  sq_diff2 = cp_int * cp_int;
                   sum_sig = sum_sig + sq_diff2;
                   sum_dif = sum_dif + sq_s_dif;
                      b2  = 1;
                }
             else
                {
                     approx = approx - stepsize;
                     s_diff = approx - cp_int;
                  sq_s_dif = s_diff * s_diff;
                  sq_diff2 = cp_int * cp_int;
                   sum_sig = sum_sig + sq_diff2;
                   sum_dif = sum_dif + sq_s_dif;
                      b2  = 0;
                }

                   cp = getc(fp);
               cp_int = conv8(cp);

          for ( i = 45 ; i < samp + 42 ; i++ )
            {

             if ( cp_int > approx )
             {
                    b3 = 1;
                if  ( b1 == 1 && b2 == 1 && b3 == 1 )
                      stepsize = 1.5 * stepsize;
                else
                  {
                     if ( b1 == 0 && b2 == 1 && b3 == 1 )
```

```
                        stepsize = 1.0 * stepsize;
                    else
                        stepsize = 0.66 * stepsize;
                }

                    approx = approx + stepsize;
                    s_diff = cp_int - approx;
                sq_s_dif = s_diff * s_diff;
                sq_diff2 = cp_int * cp_int;
                 sum_sig = sum_sig + sq_diff2;
                 sum_dif = sum_dif + sq_s_dif;
                    b1  = b2;
                    b2  = b3;
            }
            else
                {
                  b3 = 0;
               if  ( b1 == 0 && b2 == 0 && b3 == 0 )
                    stepsize = 1.5 * stepsize;
               else
                  {
                    if ( b1 == 1 && b2 == 0 && b3 == 0 )
                      stepsize = 1.0 * stepsize;
                    else
                      stepsize = 0.66 * stepsize;
                  }

                    approx = approx - stepsize;
                    s_diff = approx - cp_int;
                sq_s_dif = s_diff * s_diff;
                sq_diff2 = cp_int * cp_int;
                 sum_sig = sum_sig + sq_diff2;
                 sum_dif = sum_dif + sq_s_dif;
                    b1  = b2;
                    b2  = b3;
                }

                cp = getc(fp);
             cp_int = conv8(cp);

           }
          sqnr = sum_sig / sum_dif;
          sqdb = 10*log10(sqnr);
    printf("\nScale factor =%2.2f SQNR =%f or %f dB.",j*0.1,sqnr,sqdb);

        if ( sqdb > sqdb_max )
         {
          sqdb_max = sqdb;
          sf_max  = 0.1 * j;
         }
  }
  printf("\nTHE BEST CONDITION WHEN SCALE FACTOR = %f STEPSIZE IS = %f",
                                        sf_max,sf_max*b_step);
         fstep = sf_max * b_step;
          fclose(fp);
         }

/*  CFDM2  16 BIT MONO  */
cf216mono(char f_name4[40],int csamp)
{
    char  ch1,ch2,ch_adv1,ch_adv2,cp1,cp2;
    int   i,j,n,samp,m,m_adv,diff,sample,cp_int,b1,b2,b3;
    float sq_diff,sum_sq,approx,s_diff,sq_s_dif,sq_diff2,sum_sig,
          sum_dif,sf_max;
    double total,b_step,stepsize,sqnr,sqdb,sqdb_max;

    FILE  *fp;

          if ((fp = fopen(f_name4,"r+")) == NULL )
          {
          printf("error cfdm2 16 bit mono\n");
          getche();
          exit(1);
          }

       samp = csamp;

          fseek(fp,44,0);

                    ch1 = fgetc(fp);
```

```
                ch2 = fgetc(fp);
                m   = conv16(ch2,ch1);

          sum_sq = 0;
          sample = 0;

          for ( n = 43 ; n <= samp+42 ; n++ )
                {
                     ch_adv1 = fgetc(fp);
                     ch_adv2 = fgetc(fp);
                     m_adv   = conv16(ch_adv2,ch_adv1);

                     diff   = m_adv - m;
                     sq_diff= diff*diff;
                     sum_sq = sum_sq + sq_diff;
                  sample    = sample + 1;
                     m   = m_adv;
                }
                    total = sum_sq / sample;
                  b_step  = sqrt(total);
        printf("\n   sample = %i total = %f ",sample,total);
        printf(" stepsize = %f\n",b_step);

      sqdb_max = -100;
      sf_max   = 0;

for ( j = 495 ; j <= 520 ; j++ )
 {
      stepsize = j * b_step;

      rewind(fp);
      fseek(fp,44,0);

      approx = 0;

        cp1 = fgetc(fp);
        cp2 = fgetc(fp);
     cp_int = conv16(cp2,cp1);

            sum_sig = 0;
            sum_dif = 0;

                b1 = 0;
        if ( cp_int > approx )
                {
                    approx = approx + stepsize;
                    s_diff = cp_int - approx;
                sq_s_dif = s_diff * s_diff;
                sq_diff2 = cp_int * cp_int;
                 sum_sig = sum_sig + sq_diff2;
                 sum_dif = sum_dif + sq_s_dif;
                    b2   = 1;
                }
          else
                {
                    approx = approx - stepsize;
                    s_diff = approx - cp_int;
                sq_s_dif = s_diff * s_diff;
                sq_diff2 = cp_int * cp_int;
                 sum_sig = sum_sig + sq_diff2;
                 sum_dif = sum_dif + sq_s_dif;
                    b2   = 0;
                }

                cp1 = getc(fp);
                cp2 = getc(fp);
             cp_int = conv16(cp2,cp1);

      for ( i = 45 ; i < samp + 42 ; i++ )
        {

          if ( cp_int > approx )
          {
                b3 = 1;
            if  ( b1 == 1 && b2 == 1 && b3 == 1 )
                  stepsize = 1.5 * stepsize;
            else
              {
                 if ( b1 == 0 && b2 == 1 && b3 == 1 )
                       stepsize = 1.0 * stepsize;
```

```
                        else
                            stepsize = 0.66 * stepsize;
                    }

                        approx = approx + stepsize;
                        s_diff = cp_int - approx;
                    sq_s_dif = s_diff * s_diff;
                    sq_diff2 = cp_int * cp_int;
                     sum_sig = sum_sig + sq_diff2;
                     sum_dif = sum_dif + sq_s_dif;
                        b1   = b2;
                        b2   = b3;
                }
                else
                    {
                      b3 = 0;
                if  ( b1 == 0 && b2 == 0 && b3 == 0 )
                      stepsize = 1.5 * stepsize;
                else
                    {
                      if ( b1 == 1 && b2 == 0 && b3 == 0 )
                        stepsize = 1.0 * stepsize;
                      else
                        stepsize = 0.66 * stepsize;
                    }

                        approx = approx - stepsize;
                        s_diff = approx - cp_int;
                    sq_s_dif = s_diff * s_diff;
                    sq_diff2 = cp_int * cp_int;
                     sum_sig = sum_sig + sq_diff2;
                     sum_dif = sum_dif + sq_s_dif;
                        b1   = b2;
                        b2   = b3;
                    }

                  cp1 = fgetc(fp);
                 cp2 = fgetc(fp);
               cp_int = conv16(cp2,cp1);

                }
               sqnr = sum_sig / sum_dif;
               sqdb = 10*log10(sqnr);
        printf("\nScale factor =%i SQNR =%2.8f or %f dB.",j,sqnr,sqdb);
           if ( sqdb > sqdb_max )
            {
             sqdb_max = sqdb;
             sf_max   =  j;
            }
      }
  printf("\nTHE BEST CONDITION WHEN SCALE FACTOR = %4.1f STEPSIZE IS = %f",
                                          sf_max,sf_max*b_step);

              fstep = sf_max * b_step;
              fclose(fp);
          }

    /*  CFDM2   8 BIT STEREO  */
     cf28ster(char f_name5[40],int csamp)
     {
        char  ch,ch_adv,cp;
        int   i,y,j,n,b1,b2,b3,samp,m,m_adv,diff,sample,cp_int;
        float sq_diff,sum_sq,approx,s_diff,sq_s_dif,sq_diff2,sum_sig,
              sum_dif,sf_max1,sf_max2;
        double total,stepsiz1,stepsiz2,stepsize,sqnr1,sqnr2,sqdb1,sqdb2,
               sqdb_mx1,sqdb_mx2;

        FILE  *fp;

              if ((fp = fopen(f_name5,"r+")) == NULL )
              {
               printf("error\n");
               getche();
               exit(1);
              }

           samp = csamp;

      /*    CHANNEL  1    */
```

```
            fseek(fp,44,0);
            ch = getc(fp);
            m  = conv8(ch);

               sum_sq = 0;
               sample = 0;

                    y = 1;
               for ( n = 43 ; n <= samp + 42 ; n++ )
                     {
                        fseek(fp,44+(2*y),0);
                          ch_adv = getc(fp);
                          m_adv  = conv8(ch_adv);

                          diff   = m_adv - m;
                          sq_diff= diff*diff;
                          sum_sq = sum_sq + sq_diff;
                      sample   = sample + 1;
                          m    = m_adv;
                          y    = y + 1;
                     }
                          total  = sum_sq / sample;
                        stepsiz1  = sqrt(total);
printf("\nCH1:Total sample = %i Basic stepsize = %f\n",sample,stepsiz1);

        sqdb_mx1 = 0;
        sf_max1  = 0;

      for ( j = 1 ; j <= 30 ; j++ )
        {
                     stepsize  = 0.1*j*stepsiz1;

            rewind(fp);
            for ( i=0 ; i<44 ; i++ )
                  {
                    ch = getc(fp);
                  }

           approx = 0x80;
               cp = getc(fp);
           cp_int = conv8(cp);

                  sum_sig = 0;
                  sum_dif = 0;
                        y  = 1;

                    b1 = 0;
                if ( cp_int > approx )
                     {
                         approx = approx + stepsize;
                         s_diff = cp_int - approx;
                      sq_s_dif = s_diff * s_diff;
                      sq_diff2 = cp_int * cp_int;
                       sum_sig = sum_sig + sq_diff2;
                       sum_dif = sum_dif + sq_s_dif;
                          b2   = 1;
                     }
                else
                     {
                         approx = approx - stepsize;
                         s_diff = approx - cp_int;
                      sq_s_dif = s_diff * s_diff;
                      sq_diff2 = cp_int * cp_int;
                       sum_sig = sum_sig + sq_diff2;
                       sum_dif = sum_dif + sq_s_dif;
                          b2   = 0;
                     }

                     cp = getc(fp);
                 cp_int = conv8(cp);

            for ( i = 45 ; i < samp + 42 ; i++ )
              {

                if ( cp_int > approx )
                {
                        b3 = 1;
                    if  ( b1 == 1 && b2 == 1 && b3 == 1 )
                          stepsize = 1.5 * stepsize;
                    else
```

```
                    {
                       if ( b1 == 0 && b2 == 1 && b3 == 1 )
                            stepsize = 1.0 * stepsize;
                       else
                            stepsize = 0.66 * stepsize;
                     }

                          approx = approx + stepsize;
                          s_diff = cp_int - approx;
                       sq_s_dif = s_diff * s_diff;
                       sq_diff2 = cp_int * cp_int;
                        sum_sig = sum_sig + sq_diff2;
                        sum_dif = sum_dif + sq_s_dif;
                           b1  = b2;
                           b2  = b3;
                 }
                 else
                       {
                         b3 = 0;
                     if  ( b1 == 0 && b2 == 0 && b3 == 0 )
                           stepsize = 1.5 * stepsize;
                     else
                        {
                          if ( b1 == 1 && b2 == 0 && b3 == 0 )
                             stepsize = 1.0 * stepsize;
                          else
                             stepsize = 0.66 * stepsize;
                        }

                             approx = approx - stepsize;
                             s_diff = approx - cp_int;
                          sq_s_dif = s_diff * s_diff;
                          sq_diff2 = cp_int * cp_int;
                           sum_sig = sum_sig + sq_diff2;
                           sum_dif = sum_dif + sq_s_dif;
                              b1  = b2;
                              b2  = b3;
                        }

                     fseek(fp,44+(2*y),0);
                        cp = getc(fp);
                     cp_int = conv8(cp);
                         y = y + 1;
               }
              sqnr1 = sum_sig / sum_dif;
              sqdb1 = 10*log10(sqnr1);
printf("\nCH1: Scale factor =%2.2f SQNR =%f or %f dB.",j*0.1,sqnr1,sqdb1);
       if ( sqdb1 > sqdb_mx1 )
          {
           sqdb_mx1 = sqdb1;
           sf_max1  = 0.1*j;
          }
   }
   printf("\nTHE BEST CONDITION WHEN SCALE FACTOR = %f STEPSIZE IS = %f",
                                          sf_max1,sf_max1*stepsiz1);

            fstep = sf_max1 * stepsiz1;
            printf("\007");
            getche();

 /*  CHANNEL  2   */

            fseek(fp,45,0);
            ch = getc(fp);
            m  = conv8(ch);

               sum_sq = 0;
               sample = 0;

                    y = 1;
               for ( n = 43 ; n <= samp + 42 ; n++ )
                     {
                        fseek(fp,45+(2*y),0);
                          ch_adv = getc(fp);
                          m_adv  = conv8(ch_adv);

                          diff   = m_adv - m;
                          sq_diff= diff*diff;
                          sum_sq = sum_sq + sq_diff;
                     sample   = sample + 1;
```

```
                        m    = m_adv;
                        y    = y + 1;
                    }
                        total  = sum_sq / sample;
                    stepsiz2  = sqrt(total);
printf("\nCH2:Total sample = %i Basic stepsize = %f",sample,stepsiz2);

        sqdb_mx2 = 0;
        sf_max2  = 0;

    for ( j = 1 ; j <= 30 ; j++ )
      {
            stepsize  = 0.1*j*stepsiz2;

        rewind(fp);
        fseek(fp,45,0);

        approx = 0x80;
            cp = getc(fp);
        cp_int = conv8(cp);

            sum_sig = 0;
            sum_dif = 0;
                y  = 1;

                b1 = 0;
        if ( cp_int > approx )
            {
                approx = approx + stepsize;
                s_diff = cp_int - approx;
              sq_s_dif = s_diff * s_diff;
              sq_diff2 = cp_int * cp_int;
               sum_sig = sum_sig + sq_diff2;
               sum_dif = sum_dif + sq_s_dif;
                  b2  = 1;
            }
        else
            {
                approx = approx - stepsize;
                s_diff = approx - cp_int;
              sq_s_dif = s_diff * s_diff;
              sq_diff2 = cp_int * cp_int;
               sum_sig = sum_sig + sq_diff2;
               sum_dif = sum_dif + sq_s_dif;
                  b2  = 0;
            }

            cp = getc(fp);
        cp_int = conv8(cp);

        for ( i = 45 ; i < samp + 42 ; i++ )
          {

            if ( cp_int > approx )
            {
                   b3 = 1;
                if  ( b1 == 1 && b2 == 1 && b3 == 1 )
                     stepsize = 1.5 * stepsize;
                else
                  {
                    if ( b1 == 0 && b2 == 1 && b3 == 1 )
                         stepsize = 1.0 * stepsize;
                    else
                         stepsize = 0.66 * stepsize;
                  }

                      approx = approx + stepsize;
                      s_diff = cp_int - approx;
                    sq_s_dif = s_diff * s_diff;
                    sq_diff2 = cp_int * cp_int;
                     sum_sig = sum_sig + sq_diff2;
                     sum_dif = sum_dif + sq_s_dif;
                        b1  = b2;
                        b2  = b3;
            }
            else
                {
                  b3 = 0;
                if  ( b1 == 0 && b2 == 0 && b3 == 0 )
                     stepsize = 1.5 * stepsize;
```

```
                    else
                       {
                         if ( b1 == 1 && b2 == 0 && b3 == 0 )
                            stepsize = 1.0 * stepsize;
                         else
                            stepsize = 0.66 * stepsize;
                       }

                            approx = approx - stepsize;
                            s_diff = approx - cp_int;
                         sq_s_dif = s_diff * s_diff;
                         sq_diff2 = cp_int * cp_int;
                          sum_sig = sum_sig + sq_diff2;
                          sum_dif = sum_dif + sq_s_dif;
                             b1   = b2;
                             b2   = b3;
                         }

                    fseek(fp,45+(2*y),0);
                        cp = getc(fp);
                    cp_int = conv8(cp);
                         y = y + 1;
               }
              sqnr2 = sum_sig / sum_dif;
              sqdb2 = 10*log10(sqnr2);
printf("\nCH2: Scale factor =%2.2f SQNR =%f or %f dB.",j*0.1,sqnr2,sqdb2);
       if ( sqdb2 > sqdb_mx2 )
          {
           sqdb_mx2 = sqdb2;
           sf_max2  = 0.1*j;
          }
   }
   printf("\nTHE BEST CONDITION WHEN SCALE FACTOR = %f STEPSIZE IS = %f",
                                          sf_max2,sf_max2*stepsiz2);

              fstep2 = sf_max2 * stepsiz2;
              fclose(fp);
   }

/*  CFDM2  16 BIT STEREO  */
cf216ster(char f_name6[40],int csamp)
{
     char  ch1,ch2,ch_adv1,ch_adv2,cp1,cp2;
     int   i,j,y,n,samp,m,m_adv,diff,sample,cp_int,b1,b2,b3;
     float sq_diff,sum_sq,approx,s_diff,sq_s_dif,sq_diff2,sum_sig,
           sum_dif,sf_max1,sf_max2;
     double total,stepsiz1,stepsiz2,stepsize,sqnr1,sqnr2,sqdb1,sqdb2,
            sqdb_mx1,sqdb_mx2;

     FILE  *fp;

            if ((fp = fopen(f_name6,"r+")) == NULL )
            {
             printf("error\n");
             getche();
             exit(1);
            }

        samp = csamp;

/* CHANNEL 1 :CALCULATE BASIC STEPSIZE  */

            fseek(fp,44,0);

                    ch1 = fgetc(fp);
                    ch2 = fgetc(fp);
                    m   = conv16(ch2,ch1);

               sum_sq = 0;
               sample = 0;

                    y = 1;
               for ( n = 43 ; n <= samp+42 ; n++ )
                     {
                       fseek(fp,44+(4*y),0);
                           ch_adv1 = fgetc(fp);
                           ch_adv2 = fgetc(fp);
                           m_adv   = conv16(ch_adv2,ch_adv1);

                           diff    = m_adv - m;
```

```
                        sq_diff= diff*diff;
                        sum_sq = sum_sq + sq_diff;
                    sample   = sample + 1;
                        m    = m_adv;
                        y    = y + 1;
                    }
                       total = sum_sq / sample;
                   stepsiz1  = sqrt(total);
printf("\nCH1:  ample = %i stepsize = %f \n ",sample,stepsiz1);

    /* CHANNEL 1 : CALCULATE SQNR */

        sqdb_mx1 = -100;
        sf_max1  = 0;

      for ( j = 230 ; j <= 253 ; j++ )
       {
        stepsize = j * stepsiz1;

        rewind(fp);
        fseek(fp,44,0);

        approx = 0;

            cp1 = fgetc(fp);
            cp2 = fgetc(fp);
         cp_int = conv16(cp2,cp1);

               sum_sig = 0;
               sum_dif = 0;
                     y = 1;

                  b1 = 0;
            if ( cp_int > approx )
               {
                  approx = approx + stepsize;
                  s_diff = cp_int - approx;
               sq_s_dif = s_diff * s_diff;
               sq_diff2 = cp_int * cp_int;
                sum_sig = sum_sig + sq_diff2;
                sum_dif = sum_dif + sq_s_dif;
                  b2  = 1;
               }
            else
               {
                  approx = approx - stepsize;
                  s_diff = approx - cp_int;
               sq_s_dif = s_diff * s_diff;
               sq_diff2 = cp_int * cp_int;
                sum_sig = sum_sig + sq_diff2;
                sum_dif = sum_dif + sq_s_dif;
                  b2  = 0;
               }

              cp1 = getc(fp);
              cp2 = getc(fp);
            cp_int = conv16(cp2,cp1);

        for ( i = 45 ; i < samp + 42 ; i++ )
          {

            if ( cp_int > approx )
            {
                 b3 = 1;
              if  ( b1 == 1 && b2 == 1 && b3 == 1 )
                    stepsize = 1.5 * stepsize;
              else
                {
                  if ( b1 == 0 && b2 == 1 && b3 == 1 )
                       stepsize = 1.0 * stepsize;
                  else
                       stepsize = 0.66 * stepsize;
                }

                     approx = approx + stepsize;
                     s_diff = cp_int - approx;
                  sq_s_dif = s_diff * s_diff;
                  sq_diff2 = cp_int * cp_int;
                   sum_sig = sum_sig + sq_diff2;
                   sum_dif = sum_dif + sq_s_dif;
```

```
                        b1  = b2;
                        b2  = b3;
                }
                else
                      {
                        b3 = 0;
                    if  ( b1 == 0 && b2 == 0 && b3 == 0 )
                          stepsize = 1.5 * stepsize;
                    else
                       {
                         if ( b1 == 1 && b2 == 0 && b3 == 0 )
                            stepsize = 1.0 * stepsize;
                         else
                            stepsize = 0.66 * stepsize;
                       }

                            approx = approx - stepsize;
                            s_diff = approx - cp_int;
                        sq_s_dif = s_diff * s_diff;
                        sq_diff2 = cp_int * cp_int;
                         sum_sig = sum_sig + sq_diff2;
                         sum_dif = sum_dif + sq_s_dif;
                            b1  = b2;
                            b2  = b3;
                       }

              fseek(fp,44+(4*y),0);
                  cp1 = fgetc(fp);
                 cp2 = fgetc(fp);
              cp_int = conv16(cp2,cp1);
                   y = y + 1;

               }

              sqnr1 = sum_sig / sum_dif;
              sqdb1 = 10*log10(sqnr1);
printf("\nCH1:Scale factor =%i SQNR =%2.8f or %f dB.",j,sqnr1,sqdb1);

      if ( sqdb1 > sqdb_mx1 )
         {
          sqdb_mx1 = sqdb1;
          sf_max1  = j;
         }
   }
   printf("\nTHE BEST CONDITION WHEN SCALE FACTOR = %f STEPSIZE IS = %f",
                                          sf_max1,sf_max1*stepsiz1);
              fstep = sf_max1 * stepsiz1;
              printf("\007");
              getche();

  /* CHANNEL 2 :CALCULATE BASIC STEPSIZE  */

              fseek(fp,46,0);
                        ch1 = fgetc(fp);
                        ch2 = fgetc(fp);
                        m   = conv16(ch2,ch1);

                 sum_sq = 0;
                 sample = 0;

                      y = 1;
                 for ( n = 43 ; n <= samp+42 ; n++ )
                       {
                         fseek(fp,46+(4*y),0);
                            ch_adv1 = fgetc(fp);
                            ch_adv2 = fgetc(fp);
                            m_adv   = conv16(ch_adv2,ch_adv1);

                            diff    = m_adv - m;
                            sq_diff= diff*diff;
                            sum_sq = sum_sq + sq_diff;
                          sample   = sample + 1;
                            m    = m_adv;
                            y    = y + 1;
                       }
                           total = sum_sq / sample;
                        stepsiz2  = sqrt(total);
printf("\nCH2: sample = %i stepsize = %f \n ",sample,stepsiz2);

   /* CHANNEL 2 : CALCULATE  SQNR */
```

```
    sqdb_mx2 = -100;
    sf_max2  = 0;

for ( j = 230 ; j <= 253 ; j++ )
 {
      stepsize = j * stepsiz2;

      rewind(fp);
      fseek(fp,46,0);

      approx = 0;

          cp1 = fgetc(fp);
          cp2 = fgetc(fp);
       cp_int = conv16(cp2,cp1);

              sum_sig = 0;
              sum_dif = 0;
                    y = 1;

                   b1 = 0;
          if ( cp_int > approx )
                {
                     approx = approx + stepsize;
                     s_diff = cp_int - approx;
                  sq_s_dif = s_diff * s_diff;
                  sq_diff2 = cp_int * cp_int;
                   sum_sig = sum_sig + sq_diff2;
                   sum_dif = sum_dif + sq_s_dif;
                       b2  = 1;
                }
          else
                {
                     approx = approx - stepsize;
                     s_diff = approx - cp_int;
                  sq_s_dif = s_diff * s_diff;
                  sq_diff2 = cp_int * cp_int;
                   sum_sig = sum_sig + sq_diff2;
                   sum_dif = sum_dif + sq_s_dif;
                       b2  = 0;
                }

            cp1 = getc(fp);
            cp2 = getc(fp);
          cp_int = conv16(cp2,cp1);

      for ( i = 45 ; i < samp + 42 ; i++ )
        {

          if ( cp_int > approx )
          {
                b3 = 1;
             if  ( b1 == 1 && b2 == 1 && b3 == 1 )
                   stepsize = 1.5 * stepsize;
             else
               {
                  if ( b1 == 0 && b2 == 1 && b3 == 1 )
                       stepsize = 1.0 * stepsize;
                  else
                       stepsize = 0.66 * stepsize;
                }

                     approx = approx + stepsize;
                     s_diff = cp_int - approx;
                  sq_s_dif = s_diff * s_diff;
                  sq_diff2 = cp_int * cp_int;
                   sum_sig = sum_sig + sq_diff2;
                   sum_dif = sum_dif + sq_s_dif;
                       b1  = b2;
                       b2  = b3;
          }
          else
                {
                  b3 = 0;
             if  ( b1 == 0 && b2 == 0 && b3 == 0 )
                   stepsize = 1.5 * stepsize;
             else
                {
                  if ( b1 == 1 && b2 == 0 && b3 == 0 )
```

```
                        stepsize = 1.0 * stepsize;
                      else
                        stepsize = 0.66 * stepsize;
                    }

                        approx = approx - stepsize;
                        s_diff = approx - cp_int;
                    sq_s_dif = s_diff * s_diff;
                    sq_diff2 = cp_int * cp_int;
                     sum_sig = sum_sig + sq_diff2;
                     sum_dif = sum_dif + sq_s_dif;
                        b1   = b2;
                        b2   = b3;
                    }

            fseek(fp,46+(4*y),0);
                cp1 = fgetc(fp);
               cp2 = fgetc(fp);
            cp_int = conv16(cp2,cp1);
                 y = y + 1;

             }

            sqnr2 = sum_sig / sum_dif;
            sqdb2 = 10*log10(sqnr2);
 printf("\nCH2:Scale factor =%i SQNR =%2.8f or %f dB.",j,sqnr2,sqdb2);
      if ( sqdb2 > sqdb_mx2 )
         {
          sqdb_mx2 = sqdb2;
          sf_max2  = j;
         }
   }
   printf("\nTHE BEST CONDITION WHEN SCALE FACTOR = %f STEPSIZE IS = %f",
                                        sf_max2,sf_max2*stepsiz2);
           fstep2 = sf_max2 * stepsiz2;
           fclose(fp);
         }

/*  CVSD 8 BIT MONO  */
cvsmono(char f_name3[40],int csamp)
#define   STEP_FAC 6.5
{
     char  ch,ch_adv,cp;
     int   i,j,n,samp,m,m_adv,b1,b2,b3,diff,sample,cp_int;
     float sq_diff,sum_sq,approx,s_diff,sq_s_dif,sq_diff2,sum_sig,
           sum_dif,sf_max;
     double total,b_step,stepsize,sqnr,sqdb,sqdb_max;

     FILE  *fp;

           if ((fp = fopen(f_name3,"r+")) == NULL )
           {
            printf("error in cvsd mono 8 bit\n");
            exit(1);
           }
             fseek(fp,44,0);

                   ch = getc(fp);
                   m  = conv8(ch);

            sum_sq = 0;
            sample = 0;

       samp = csamp;

            for ( n = 43 ; n <= samp + 42 ; n++ )
                  {
                       ch_adv = getc(fp);
                       m_adv  = conv8(ch_adv);

                       diff   = m_adv - m;
                       sq_diff= diff*diff;
                       sum_sq = sum_sq + sq_diff;
                     sample   = sample + 1;
                       m      = m_adv;
                  }
                       total  = sum_sq / sample;
                       b_step = sqrt(total);

    printf("\nTotal sample = %i Basic stepsize = %f",sample,b_step);
```

```
      sqdb_max = 0;
      sf_max   = 0;

       for ( j = 1 ; j <= 25 ; j++ )
         {
                       stepsize  = 0.1*j*b_step;

             rewind(fp);
             fseek(fp,44,0);

             approx = 0x80;
                 cp = getc(fp);
             cp_int = conv8(cp);

                    sum_sig = 0;
                    sum_dif = 0;
                         b1 = -1;
                         b2 =  1;

   for ( i = 44 ; i < samp + 42 ; i++ )
     {
      if ( cp_int > approx )
       {
           b3  = 1;
       if ( b1 == b2 && b2 == b3 )
          stepsize = (0.9937 * stepsize) + STEP_FAC*(1-0.9937);
       else
          stepsize = 0.9937 * stepsize;

                     approx = approx + stepsize;
                     s_diff = cp_int - approx;
                   sq_s_dif = s_diff * s_diff;
                   sq_diff2 = cp_int * cp_int;
                    sum_sig = sum_sig + sq_diff2;
                    sum_dif = sum_dif + sq_s_dif;
                         b1 = b2;
                         b2 = b3;
                       }
   else
     {
           b3  = -1;
       if ( b1 == b2 && b2 == b3 )
          stepsize = (0.9937 * stepsize) + STEP_FAC*(1-0.9937);
       else
          stepsize =  0.9937 * stepsize;

                     approx = approx - stepsize;
                     s_diff = approx - cp_int;
                   sq_s_dif = s_diff * s_diff;
                   sq_diff2 = cp_int * cp_int;
                    sum_sig = sum_sig + sq_diff2;
                    sum_dif = sum_dif + sq_s_dif;
                         b1 = b2;
                         b2 = b3;
      }

                        cp = getc(fp);
                    cp_int = conv8(cp);

   }
               sqnr = sum_sig / sum_dif;
               sqdb = 10*log10(sqnr);
    printf("\nScale factor =%2.2f SQNR =%f or %f dB.",j*0.1,sqnr,sqdb);
              if ( sqdb > sqdb_max )
              {
               sqdb_max = sqdb;
               sf_max   = 0.1*j;
              }
             }
   printf("\nTHE BEST CONDITION WHEN SCALE FACTOR = %f STEPSIZE = %f",
                                              sf_max,sf_max*b_step);
              fstep = sf_max*b_step;
              fclose(fp);
             }

          /*  CONVERT  PROGRAM  FOR  8 BIT  */
           int   conv8(ch8)
           char  ch8;
           {
```

```
int  mc;

     mc = (int)ch8;
   if  ( mc < 0 )
       mc = mc - 0xFF00;
     return(mc);
}

/*  CONVERT  PROGRAM FOR 16 BITS  */
  int  conv16(chr1,chr2)
  char chr1,chr2;
  {
    int  m1,m2,m_con;

      m1  =  (int)chr1;
    if ( m1 < 0 )
       m1  = m1 - 0xFF00;

      m2  =  (int)chr2;
    if ( m2 < 0 )
       m2  = m2 - 0xFF00;

      m_con = (m1 * 256) + m2;
      m_con = m_con + 0x8000;
   return(m_con);
}
```

บรรณานุกรม

1. HERBERT TAUB AND DONALD L.SCHILLING, *"PRINTCIPLES OF COMMU-NICATION SYSTEMS"*, SECOND EDITION, MCGRAW-HILL BOOK COMPANY, 1986
2. LEON W.COUCH, *"DIGITAL AND ANALOG COMMUNICATION SYSTEMS"*, SECOND EDITION, MACMILLAN PUBLISHING COMPANY, NEW YORK. 1987
3. N.S.JAYANT-PETER NOLL. *"DIGITAL CODING OF WAVEFORMS :PRINCIPLE AND APPLICATIONS TO SPEECH AND VIDEO"*, PRENTICE HALL INC., 1984
4. CHONG KWAN UN.,HWANG SOO LEE, JOO SEON SONG. "HYBRID COMPANDING DELTA MODULATION", *"IEEE TRANSACTION ON COMMUNICATION"*, VOL COM-19. No. 9,SEPTENBER 1981
5. H.S.LEE, C.K.UN, "QUANTIZATION NOISE IN ADAPTIVE DELTA MODULATION SYSTEM", *"IEEE TRANSACTION ON COMMUNICATION"*, VOL. COM-28, pp.1794-1802, OCTOBER 1980
6. ขจร โรจนเมธินทร์, *"Visual Basic:สนุกกับเวฟไฟล์"*, ไมโครคอมพิวเตอร์ ฉบับที่ 108, หน้า 255-260
7. สุทธิศักดิ์ พงศ์ธนาพาณิช, *"มัลติมีเดีย MCI :ไฟล์ .wav"*, ไมโครคอมพิวเตอร์ ฉบับที่ 108, หน้า 293-298
8. บัณฑิต โรจน์อารยานนท์, *"หลักการไฟฟ้าสื่อสาร"*, พิมพ์ครั้งที่ 2 , สำนักพิมพ์จุฬาลงกรณ์มหาวิทยาลัย, 2534
9. ถวิล กิ่งทอง, *"เทคโนโลยีการส่งสัญญาณดิจิตอล"*, กรุงเทพฯ 2535, สำนักพิมพ์สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

กิตติกรรมประกาศ

ผู้จัดทำโครงงานต้องขอขอบพระคุณอาจารย์ยุทธพงษ์ รังสรรค์เสรี ซึ่งเป็นอาจารย์ที่ปรึกษาโครงงานเป็นอย่างสูงที่ได้ให้หัวข้อโครงงานทั้งยังให้คำแนะนำต่างๆ ที่เป็นประโยชน์ต่อการทำโครงงานนี้ ทั้งยังต้องขอบพระคุณ รศ.ดร.กอบชัย เดชหาญ ที่ได้ให้คำปรึกษาแก่ผู้จัดทำในขณะที่ อาจารย์ยุทธพงษ์ ไม่อยู่ด้วย ซึ่งอาจารย์ทั้งสองท่านนี้มีส่วนอย่างมากในการทำให้โครงงานนี้สำเร็จลงได้ด้วยดี