โปรแกรมวิเคราะห์และแต่งเสียง

Sound wave Edit & Analyzer

โดย

นายอธิป	ศรีกระจิบ	48011042
นายอินทรีย์	เถี่ยนมิตรภาพ	48011126
นายเอกพจน์	พรโสภิณ	48011143

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2551

ตรวจแล้ว

โปรแกรมวิเคราะห์และแต่งเสียง

Sound wave Edit & Analyzer

โดย

นายอธิป ศรีกระจิบ 48011042

นายอินทรีย์ เถี่ยนมิตรภาพ 48011126

นายเอกพจน์ พรโสภิณ 48011143

อาจารย์ที่ปรึกษา

รศ.ดร. สุวิพล สิทธิชีวิภาค

รศ. เกรียงไกร วงศ์โรจนภรณ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2551

ปริญญานิพนธ์ปีการศึกษา 2551

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง **โปรแกรมวิเคราะห์และแต่งเสียง**

Sound wave Edit & Analyzer

ผู้จัดทำ

1. นายอธิป ศรีกระจิบ 48011042
2. นายอินทรีย์ เถี่ยนมิตรภาพ 48011126
3. นายเอกพจน์ พรโสภิณ 48011143

..... อาจารย์ที่ปรึกษา

(รศ.ดร. สุวิพล สิทธิชีวภาค)

..... อาจารย์ที่ปรึกษา

(รศ. เกรียงไกร วงศ์โรจนภรณ์)

โครงงานเลขที่ 512403

โปรแกรมวิเคราะห์และแต่งเสียง

Sound wave Edit & Analyzer

โดย	นาย อธิป	ศรีกระจิบ	48011042
	นาย อินทรีย์	เถี่ยนมิตรภาพ	48011126
	นาย เอกพจน์	พรโสภิณ	48011143

อาจารย์ที่ปรึกษา รศ. เกรียงไกร วงศ์โรจนภรณ์

รศ. ดร. สุวิพล สิทธิชีวภาค

บทคัดย่อ

โครงงานนี้เป็นการออกแบบและสร้างโปรแกรม Sound wave Editor & Analyzer ซึ่งเป็นโปรแกรมที่ใช้ในการวิเคราะห์ ตัดต่อ และแต่งไฟล์เสียงนามสกุล .wav โปรแกรมนี้สามารถที่จะตัดต่อข้อความเสียงที่ช่วงใดๆของไฟล์ก็ได้ สามารถเพิ่มลดขนาดของแอมพลิจูดเสียง และสามารถกรองความถี่ได้ โปรแกรมนี้แสดงผลเป็นภาพกราฟเสียงในแกนเวลา และแกนความถี่ บนจอคอมพิวเตอร์ได้

Abstract

This project is about design and creates a program for sound wave editor analyzer. This program can edit and analyze a sound wave in .wav file format. This program can cut, copy, paste inside wave file and also adjust gain of the wave file data. This program can filter some frequency define by user and can display wave file in graphic on time domain and frequency domain by computer display.

สารบัญ

สารบัญรูป

หน้า

สารบัญตาราง

หน้า

บทที่ 1
บทนำ

3.1 ความเป็นมาและความสำคัญของปัญหา

สัญญาณเสียงเป็นสัญญาณพื้นฐานในการติดต่อสื่อสารระหว่างมนุษย์มาตั้งแต่สมัยโบราณโดยเสียงมีวิวัฒนาการมาเรื่อยๆ จนทำให้เกิดรูปแบบเสียงในลักษณะต่างๆ มากมาย ไม่ว่าจะเป็นเสียงของการสนทนากัน เสียงดนตรี เสียงเครื่องจักรที่กำลังทำงาน ต่างก็มีรูปแบบและองค์ประกอบทางสัญญาณที่แตกต่างกันไป เช่น มีเฟสที่ต่างกัน ความถี่ที่ไม่เท่ากัน สิ่งเหล่านี้ล้วนส่งผลให้เสียงมีคุณสมบัติที่แตกต่างกัน เราจึงควรที่จะศึกษารูปแบบพื้นฐานของสัญญาณเสียงและนำไปประยุกต์ใช้ให้เกิดประโยชน์

เนื่องจากสัญญาณรบกวน (noise) เป็นสิ่งที่ไม่พึงปรารถนาให้เกิดขึ้นกับสัญญาณที่ใช้ในการประมวลผลทำให้สัญญาณที่เกิดขึ้นมีความผิดเพี้ยน เช่น มีความถี่ (frequency) , เฟส (phase) และ แอมพลิจูด (amplitude) ของสัญญาณที่แตกต่างไปจากเดิมส่งผลทำให้สัญญาณที่ได้มีความคมชัดน้อยลงไป วิธีการที่ใช้ในการกำจัดสัญญาณรบกวนนี้ จำเป็นที่จะต้องใช้หลักการของ digital signal processing เราจึงแปลงสัญญาณที่เป็น อนาล็อกให้เป็น สัญญาณดิจิทัลในรูปแบบของไฟล์ชนิดต่างๆ เพื่อที่จะให้คอมพิวเตอร์สามารถรับรู้ และประมวลผลได้

1.2 วัตถุประสงค์ของโครงงาน

โครงงานนี้จัดทำขึ้นโดยการเขียนโปรแกรมบนเครื่องคอมพิวเตอร์ โดยนำเอาสัญญาณเสียงที่ถูกเก็บไว้เป็น .wav format แบบ PCM (Uncompressed) เพื่อนำมา plot เป็นรูปของสัญญาณเสียงได้ เพื่อให้ผู้ใช้ได้เห็นลักษณะของสัญญาณ และทำการแก้ไขดัดแปลงเสียงได้สะดวกยิ่งขึ้น เพื่อทำให้เสียงที่ถูกบันทึกในไฟล์นั้นๆมีคุณภาพดียิ่งขึ้น หรือมีคุณสมบัติเป็นไปตามที่ผู้ใช้ต้องการ และสามารถนำไปประยุกต์ใช้ให้เกิดประโยชน์ในขั้นต่อไปได้ ซึ่งโปรแกรมนี้จะออกแบบโดยการใช้ภาษา c++ โดยใช้โปรแกรม Microsoft visual studio 2008 ในการพัฒนาโปรแกรม

1.3 ขอบเขตของโครงงาน

โปรแกรมนี้สามารถแสดงรูปของสัญญาณเสียงได้โดยผ่านทางหน้าจอคอมพิวเตอร์ และสามารถใช้งานฟังก์ชั่นเหล่านี้ได้

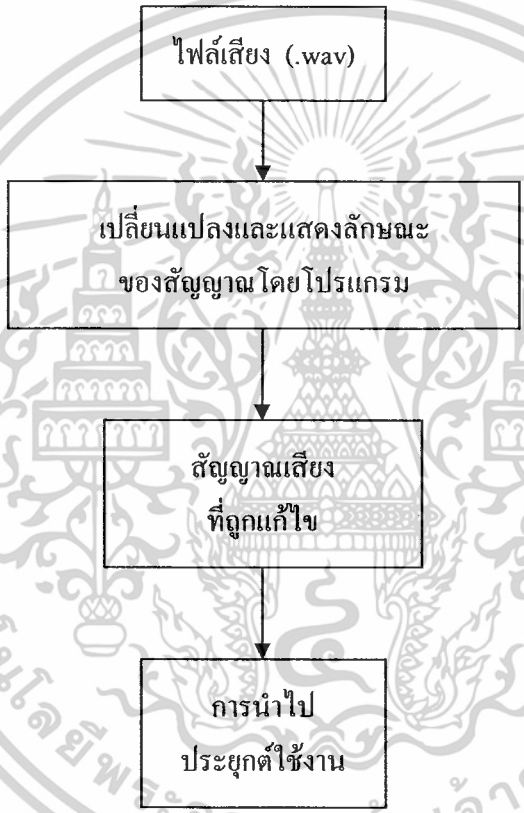
- Copy
- Cut
- Paste
- Play
- Pause
- Stop
- Change Gain
- Silence
- Change Play Rate

1.4 ประโยชน์ที่คาดว่าจะได้รับ

1. ผู้ใช้สามารถมองเห็นรูปลักษณะของสัญญาณเสียงระหว่างที่ทำการแก้ไข
2. ผู้ใช้สามารถทำการแก้ไขดัดแปลงเสียงได้โดยง่าย
3. สามารถทำให้เสียงมีคุณภาพที่ดีขึ้น
4. สามารถลด noise บางส่วนลงได้

1.5 โครงสร้างของโครงงาน

แผนผังแสดงโครงสร้างของโครงงานโดยรวมจะเป็นดังนี้



รูปที่ 1.1 แสดงแผนผังโดยรวมของโครงงาน

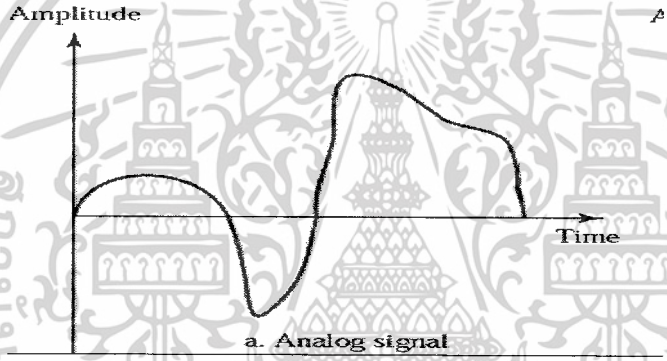
บทที่ 2
ทฤษฎีหรือหลักการ

2.1 ชนิดของสัญญาณ

ในการส่งข้อมูลเพื่อทำการติดต่อสื่อสารระหว่างเครื่องคอมพิวเตอร์ หรืออุปกรณ์นั้น จะต้องแปลงข้อมูลที่ต้องการให้อยู่ในรูปแบบ “สัญญาณ (Signal)” เพื่อให้สามารถส่งไปในตัวกลางที่เป็นช่องทางการสื่อสารได้ โดยสามารถแบ่งสัญญาณที่ใช้ในการสื่อสารได้เป็น 2 ชนิดคือ สัญญาณอนาล็อกและสัญญาณดิจิตอล

สัญญาณอนาล็อก

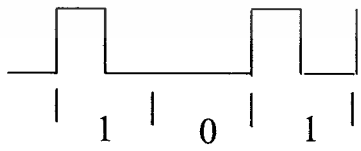
สัญญาณอนาล็อก คือ สัญญาณที่อยู่ในรูปแบบของคลื่น ที่มีความต่อเนื่องกัน มีการเปลี่ยนแปลงระดับของสัญญาณขึ้น – ลงตามขนาดของสัญญาณ และมีความถี่ ที่เรียกว่า Hertz (Hz) ตัวอย่างของสัญญาณอนาล็อก เช่น เสียงพูด (Voice) กระแสไฟฟ้าสลับ เป็นต้น



รูปที่ 2.1 แสดงลักษณะของสัญญาณอนาล็อก

สัญญาณดิจิตอล

สัญญาณดิจิตอล หรือเรียกว่า “สัญญาณพัลซ์ (Pulse Signal)” คือ สัญญาณที่มีระดับของสัญญาณเพียง 2 ระดับ คือ สูงและต่ำ การเปลี่ยนระดับสัญญาณจะไม่มีความต่อเนื่องกัน โดยปกติแล้วระดับสูงจะแทนด้วยตัวเลข 1 และระดับต่ำจะแทนด้วยตัวเลข 0



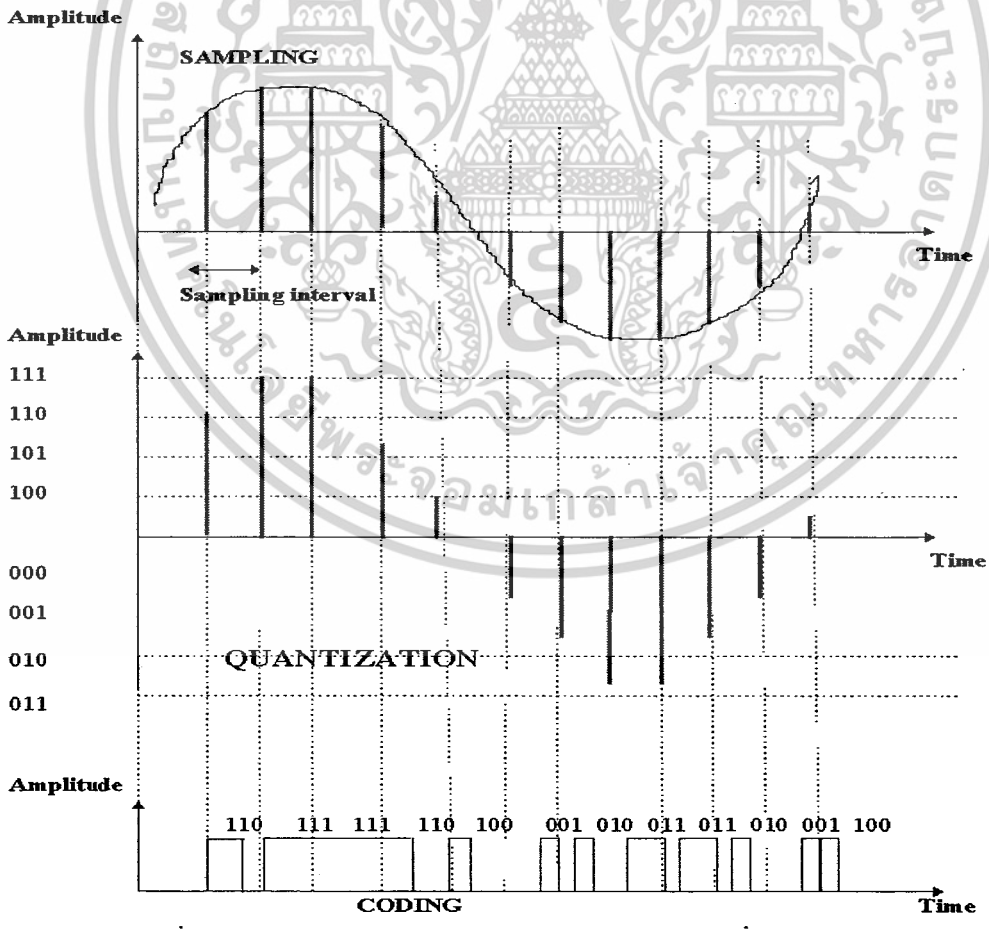
รูปที่ 2.2 แสดงลักษณะของสัญญาณอนาล็อก

2.2 ระบบพัลซ์โค้ดมอดูเลชั่น (PCM)

ในปีค.ศ. 1937 ระบบ PCM ได้ถูกคิดค้นโดย A.H. Reeves ซึ่งเป็นสิ่งที่ดีมากของระบบของสายส่ง ปัญหาเกี่ยวกับระบบความผิดเพี้ยนของสัญญาณและสัญญาณรบกวนก็สามารถแก้ไขได้ และพยายามที่จะแก้ไขปัญหาอื่น ๆ ซึ่งทำได้ยากในระบบอนาล็อก แม้จะมีปัญหาสำคัญซึ่งก็คือระบบ PCM ไม่สามารถที่จะส่งได้ตลอดเวลา จึงทำให้ต้องใช้อุปกรณ์ที่มีความเร็วสูง เพื่อจะทำให้สัญญาณที่ใกล้เคียงกับสัญญาณเดิมเป็นต้นว่า จากการที่มีการพัฒนาอุปกรณ์ที่มีความเร็วสูง เป็นต้นว่า ทรานซิสเตอร์ที่ถูกสร้างขึ้นเมื่อปี 1945 จากนั้นก็มีการพัฒนาอย่างต่อเนื่อง การทดลองใช้ชุมสายโทรศัพท์ดิจิตอง (ESSEY) ได้ถูกสร้างโดย Bell Telephone Laboratory ในปี 1959 ชุมสายนี้ได้นำเอาวิธีการของ PCM เข้ามาใช้ในส่วนของเสียงพูดผ่านแต่ก็ไม่ได้นำมาทดลองใช้กับโครงข่ายโทรศัพท์เดิม เพราะต้องใช้อุปกรณ์บางส่วนซึ่งในขณะนั้นยังมีราคาแพงทำให้ต้นทุนการผลิตมีราคาแพงกว่าระบบอนาล็อกที่มีอยู่เดิม

พื้นฐานของ PCM

PCM เป็นวิธีการเปลี่ยนศักดาไฟฟ้าของสัญญาณอนาล็อก ให้เป็นรหัสไบนารี่ ในช่วงเวลาที่กำหนด โดยรหัสไบนารี่นี้จะถูกเปลี่ยนแปลงตามศักดาไฟฟ้าที่แต่ละค่า ซึ่งรบกวนการสร้างสัญญาณ PCM สามารถทำเป็น 3 ขั้นตอนดังนี้

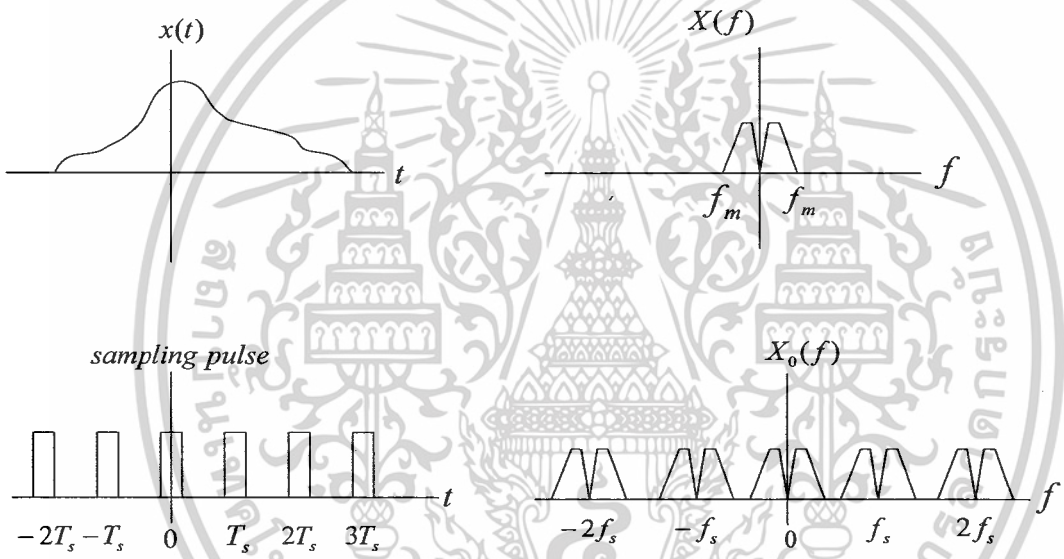


รูปที่ 2.3 การแปลงสัญญาณ Analog เป็นสัญญาณ Digital ที่ใช้ในระบบ PCM

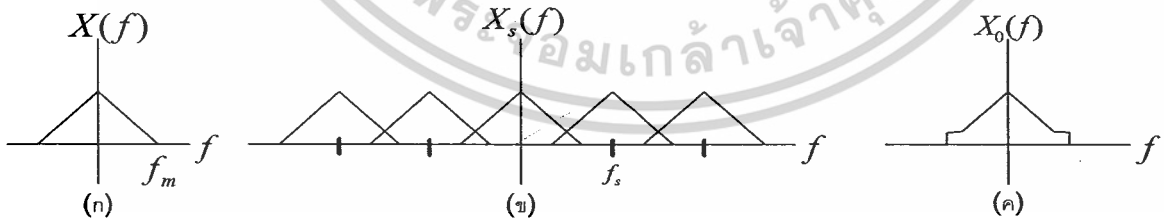
- การสุ่มตัวอย่าง (Sampling)
- การแบ่งย่าน Amplitude ออกเป็นระดับต่าง ๆ (Quantizing)
- การเข้ารหัส (Coding)

1) การสุ่มตัวอย่าง (Sampling)

หมายถึงการเลือกเอาค่า Amplitude ที่จุดใด ๆ ของสัญญาณ Analog ที่มีช่วงเวลาที่เท่ากัน ตัวอย่างที่สุ่มมาได้ก็คือ Pulse Train หรือเรียกว่า PAM Sample จำนวนของสุ่ม ตัวอย่างต่อวินาทีคือ Sampling Rate จาก Sampling Theorem ที่กล่าวได้ว่า ถ้าได้ทำการสุ่มตัวอย่าง (Sampling) สัญญาณ Analog ด้วยช่วงเวลาที่สม่ำเสมอในอัตราอย่างน้อยเป็น 2 เท่าของความถี่สูงสุดของสัญญาณนั้น ๆ แล้ว ตัวอย่างที่สุ่มมาได้จะบรรจุข่าวสารของสัญญาณเดิมครบถ้วน ในระบบ PCM สัญญาณโทรศัพท์จะถูกสุ่มตัวอย่างด้วย Sampling Rate 8000 ครั้งต่อวินาที หรือถูกสุ่มตัวอย่างทุก ๆ 125 ไมโครวินาที ซึ่งเรียกว่า Sampling Interval



รูปที่ 2.4 การ sampling สัญญาณและสเปกตรัม

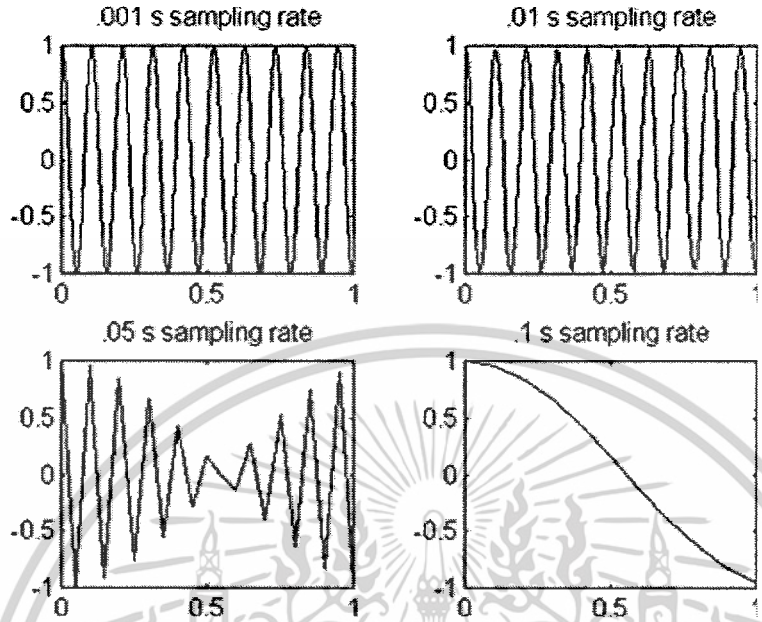


รูปที่ 2.5 สเปกตรัมของสัญญาณ X(f) ที่ เกิด aliasing (ก) X(f) (ข) หลังจาก sampling (ค) หลังผ่านฟิลเตอร์

Sampling Rate และ Bit Depth ของเสียง

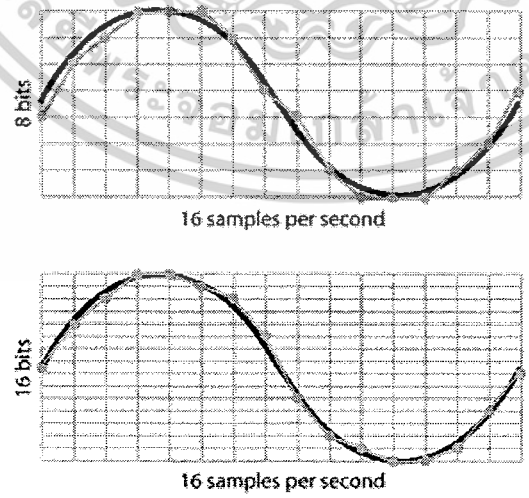
Sampling Rate เป็นตัวกำหนดความละเอียดของคลื่นความถี่เสียง หรือจุดที่บอกว่าเสียง ณ ตำแหน่งนั้นๆอยู่ที่ความถี่เท่าใด เช่นไฟล์เสียงที่มี Sampling Rate 22 กิโลเฮิรตซ์ หมายความว่า ในหนึ่งคลื่นเสียงประกอบไปด้วยจุดที่บอกความถี่ประมาณ 2 หมื่นจุดและในทำนองเดียวกัน ไฟล์ที่มี Sampling Rate 44

กิโลเฮิรตซ์ หมายความว่า มีจุดบอกตำแหน่งความถี่จำนวน 4 หมื่นกว่าจุดในหนึ่งคลื่นเสียงนั่นเองซึ่งยิ่งมี Sampling Rate มากเท่าไหร่ ก็หมายถึงคุณภาพเสียงก็จะมีมากเพิ่มขึ้นเท่านั้น



รูปที่ 2.6 แสดงสัญญาณที่ถูกแซมปลิ้ง

Bit Depth คือ ระยะห่างของจุดเสียงสูงสุดกับจุดเสียงต่ำสุด (เสียงแหลม - เสียงทุ้ม) ของคลื่นเสียง หรือ เรียกว่า Dynamic ซึ่งเป็นช่วงความกว้างของคลื่นเสียงที่มีผลต่อโทนเสียงทุ้มและเสียงแหลมของเพลง ดังนั้นการที่มี Bit Depth มากๆก็หมายความว่า ในคลื่นเสียงแต่ละลูกจะมีความต่างของเสียงสูงสุดและต่ำสุดมาก แต่ถ้ามี Bit Depth น้อยแล้ว ณ จุดที่เสียงสูงหรือต่ำกว่าค่าเฉลี่ยก็จะถูกเพิ่มหรือลดความถี่เสียงลงเพื่อให้อยู่ในค่าเฉลี่ยที่กำหนดไว้ ผลที่ได้คือเสียงออกมาจะไม่เป็นธรรมชาติ



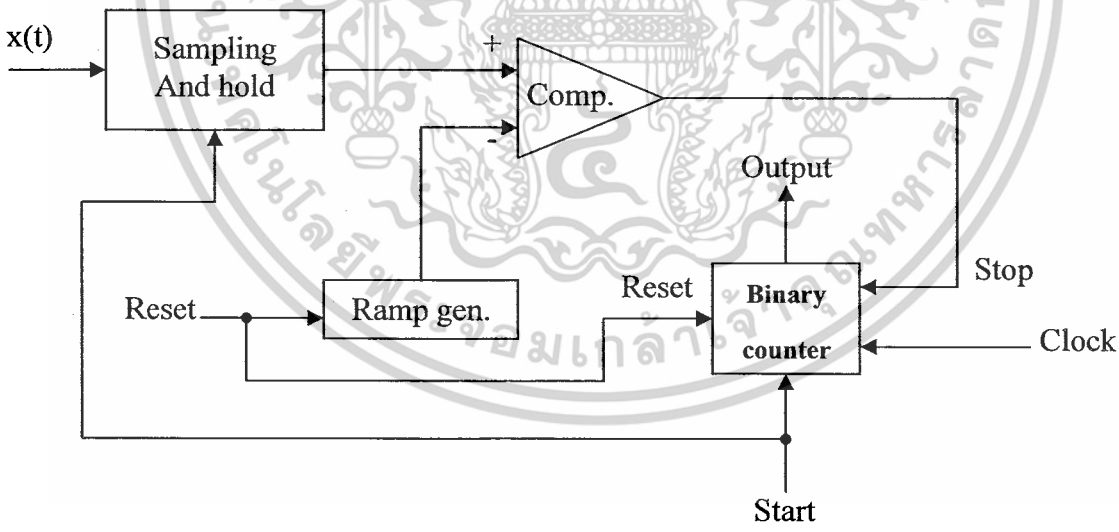
รูปที่ 2.7 แสดงจำนวนแซมเปิ้ลต่อวินาทีของสัญญาณ

2) การควอนไทซ์

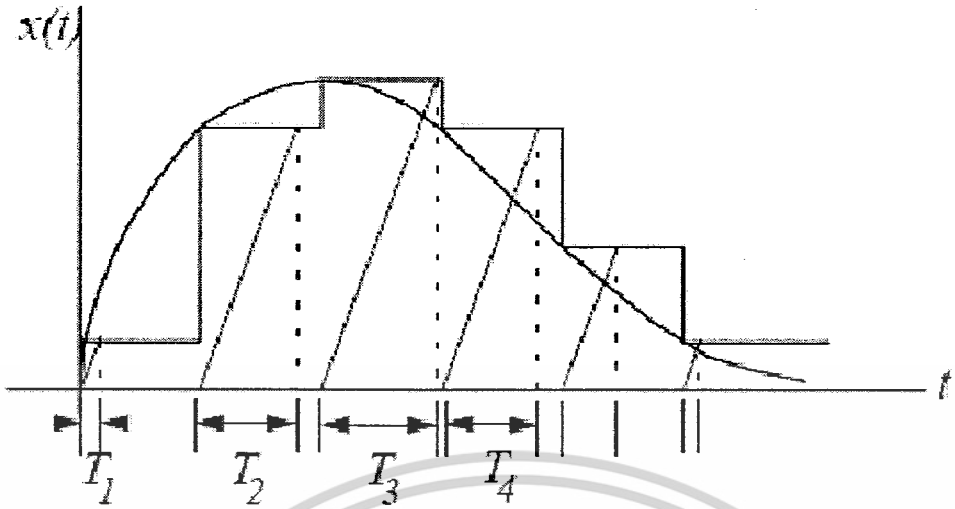
ซึ่งจะเป็นการปรับโวลท์เตจของสัญญาณ PAM ที่ได้จากการแซมปลิ้ง เพื่อทำให้เป็นค่าคงที่ค่าหนึ่งซึ่งก็คือสเต็ปของการควอนไทซ์ ซึ่งจากสเต็ปที่ได้ก็จะใช้ในการเปลี่ยนแปลงรหัสไบนารี่ ความผิดพลาดระหว่างสเต็ปของการควอนไทซ์ กับโวลท์เตจของสัญญาณเดิมที่เดิดขึ้นจะขึ้นอยู่กับค่าของสเต็ปว่าจะมากหรือน้อยแค่ไหน ในการ quantize สัญญาณนี้ โดยทั่วไปมี 5 วิธี คือ

1. Counter quantizer
2. Serial quantizer
3. Parallel quantizer
4. Quantization noise
5. Non - uniform quantization

Counter quantizer รูปที่ 2.8 แสดง counter quantizer สัญญาณอนาลอกจะถูกวงจร sampling and hold สุ่มสัญญาณและค้างไว้ซึ่งก็ พร้อมกับวงจรนับและวงจรกำเนิดสัญญาณ ramp เริ่มทำงาน วงจรนับจะนับเพิ่มค่าไปเรื่อยๆ จนกว่าสัญญาณ ramp จะเท่ากับสัญญาณอนาลอกที่สุ่มและค้างไว้ วงจรนับจะหยุดและค่าที่ได้ก็จะเป็นค่าดิจิตอลแทนค่าอนาลอกที่สุ่มได้ ถ้าค่าสัญญาณที่สุ่มได้มีค่าสูงวงจร ramp ต้องใช้เวลานานในการสร้างสัญญาณจนมีค่าเท่ากับสัญญาณที่สุ่มได้ วงจรนับจึงมีเวลาในการนับนานทำให้ ได้ค่าดิจิตอลที่แทนสัญญาณอนาลอกมีค่าสูงด้วย

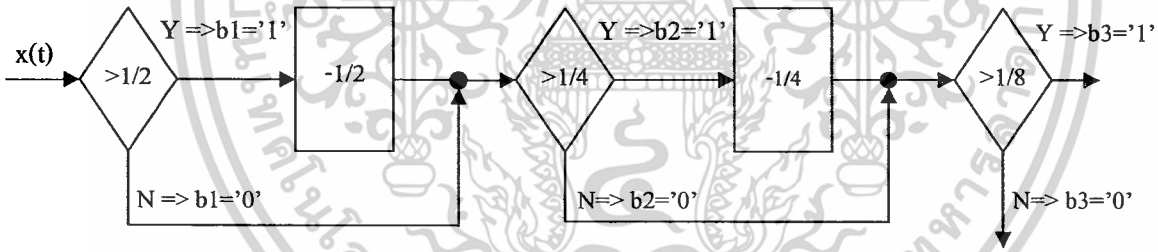


รูปที่ 2.8 Counter quantizer



รูปที่ 2.9 สัญญาณจาก Counter quantizer

Serial quantizer สัญญาณที่ sampling ได้ จะถูกตรวจสอบว่าสูงกว่าครึ่งหนึ่งของสัญญาณสูงสุดหรือไม่ โดยการเปรียบเทียบ ถ้าสูงกว่าครึ่งให้ผลลัพธ์บิตแรกเป็น 1 ถ้าต่ำกว่าครึ่งจะให้ผลลัพธ์บิตแรกนี้เป็น 0 แล้วจึงพิจารณาแบ่งครึ่งของส่วนที่เกินครึ่งหรือส่วนที่เหลือไม่ถึงครึ่งแรกนั้นคือเปรียบเทียบกับหนึ่งในสี่ของสัญญาณสูงสุดทำการเปรียบเทียบครึ่งของครึ่งของสัญญาณไปเรื่อยๆ จนครบจำนวนบิตผลลัพธ์ที่ต้องการ ตัวอย่างการใช้ serial quantizer คือ IC ADC0804 เป็นต้น รูปที่ 2.10 แสดง serial quantizer

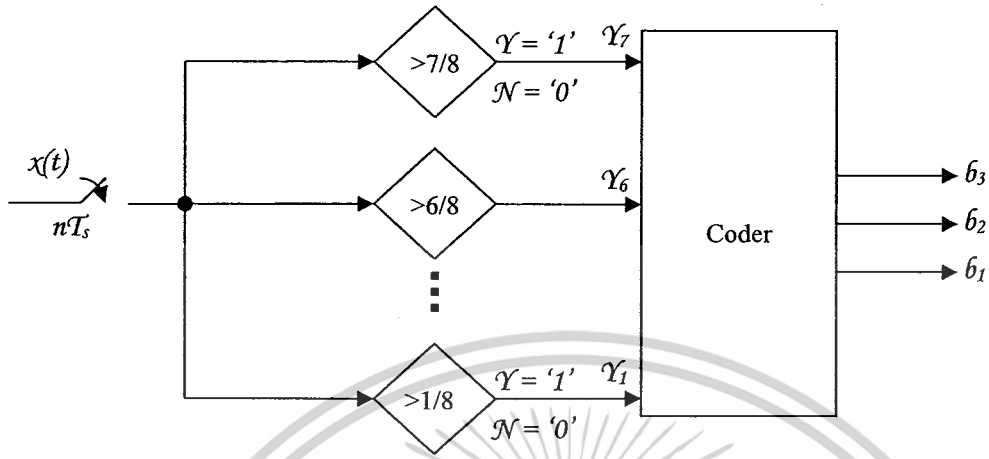


รูปที่ 2.10 Serial quantizer

เป็นที่น่าสังเกตในการทำ serial quantize นั้นผลลัพธ์ที่ได้จะไม่ออกมาทันทีที่ป้อนสัญญาณเข้า เนื่องจากต้องรอการเปรียบเทียบเป็นขั้นๆ ไปจึงต้องมีเวลาของการ quantize ซึ่งจะมากน้อยขึ้นอยู่กับจำนวนบิตที่จะใช้ในการแทนสัญญาณนั้น

Parallel quantizer สัญญาณที่สุ่มตัวอย่างได้ จะถูกนำไปเปรียบเทียบกับ comparator มีจำนวนเท่ากับจำนวนระดับที่ ต้องการแยกโดย comparator แต่ละตัวจะถูกกำหนดระดับสำหรับการเปรียบเทียบไว้ด้วยระดับต่าง ๆ ขึ้นอยู่กับความละเอียดที่ต้องการแล้วจึงนำเอาผลที่ ได้จาก comparator ไปเข้าวงจรเข้ารหัสซึ่งเป็นวงจรตรรกสัญญาณขาออกของวงจรเข้ารหัสจะเป็นสัญญาณดิจิตอลที่มี จำนวนบิตเท่ากับจำนวนบิตที่ สามารถแทนระดับที่ แยกไว้ ด้วย comparator ได้ รูปที่ 2.11 แสดง parallel quantization จากการที่ สัญญาณถูกเปรียบเทียบทุกระดับพร้อมกันทันที และผ่านวงจรเข้ารหัสที่ เป็นวงจรตรรกจึงทำให้ได้ผลลัพธ์ออกมาเกือบจะ

ทันทีที่ให้สัญญาณเข้าบางครั้งจึงถูกเรียก ว่า flash ADC เป็นวงจรแปลงสัญญาณอนาลอกเป็นสัญญาณดิจิตอล ที่ เร็วที่ สุดแต่จะต้องใช้ comparator จำนวนมาก



รูปที่ 2.11 Parallel quantizer

Quantization noise จากการแปลงสัญญาณด้วยวิธีการต่าง ๆ ดังที่กล่าวไปแล้วนั้นความสำคัญของการแปลงสัญญาณอยู่ที่ การได้สัญญาณดิจิตอลที่แทนสัญญาณอนาลอกได้ใกล้เคียงมากที่สุด และใช้เวลาในการแปลงน้อยที่สุดการที่จะแทนสัญญาณ ให้ใกล้เคียงสัญญาณมากที่สุดต้องแบ่งจำนวนระดับที่จะแทนสัญญาณนั้นมากที่สุดนั่นคือจำนวนบิตที่ใช้ในการแทนสัญญาณต้องมากที่สุดเท่าที่จะเป็นไปได้ ในจำนวนบิตที่จำกัดค่าหนึ่งจะได้จำนวนระดับที่จำกัดเช่นกัน ตัวอย่างเช่น จำนวนบิตเท่ากับ 3 จะมี จำนวนระดับเท่ากับ 8 ระดับในการ quantize สัญญาณอนาลอกขนาดเท่าใดก็ตามจะถูกจัดเข้าเป็นระดับแค่ 8 ระดับนี้ เท่านั้นในขั้นตอนการแปลงสัญญาณกลับเป็นสัญญาณอนาลอกอาจทำให้เกิดความแตกต่าง ระหว่างสัญญาณอนาลอกเดิมกับสัญญาณอนาลอกที่ แปลงกลับจากสัญญาณดิจิตอล เนื่องจากสัญญาณอนาลอกมีค่าที่เป็นไปได้ ทุกค่า ในขณะที่ สัญญาณอนาลอกที่ แปลงจากสัญญาณดิจิตอลดังกล่าวจะมีค่าเพียง 8 ค่าดังกล่าวค่าแตกต่างระหว่างสัญญาณเดิมกับสัญญาณที่ quantize ได้ แต่ละค่าจะไม่ เกิน $\pm \Delta x/2$ เมื่อ Δx คือความกว้างของขั้นหรือระดับของการ quantize error

$$error \leq \pm \frac{\Delta x}{2}$$

ให้ p(en) เป็นพรอบอะบิลิตี้ของ ค่า error(en) นี้มีค่าคงที่ตลอดช่วง - $\Delta x/2$ถึง + $\Delta x/2$ หาค่า mean square error; mse ได้เป็น

$$mse = \int_{-\infty}^{\infty} p(e_n) e_n^2 de_n$$

$$= \frac{1}{\Delta x} \int_{-\Delta x/2}^{\Delta x/2} e_n^2 de_n = \frac{\Delta x^2}{12}$$

จากสมการค่ายกกำลังสองเฉลี่ยของ error(en) ที่จะเป็นไปได้อาจเรียกว่าเป็นสัญญาณรบกวนเมื่อนำมาพิจารณากับตัวสัญญาณ ได้เป็นค่าสัญญาณต่อสัญญาณรบกวน (signal to quantization noise ratio; SNR)

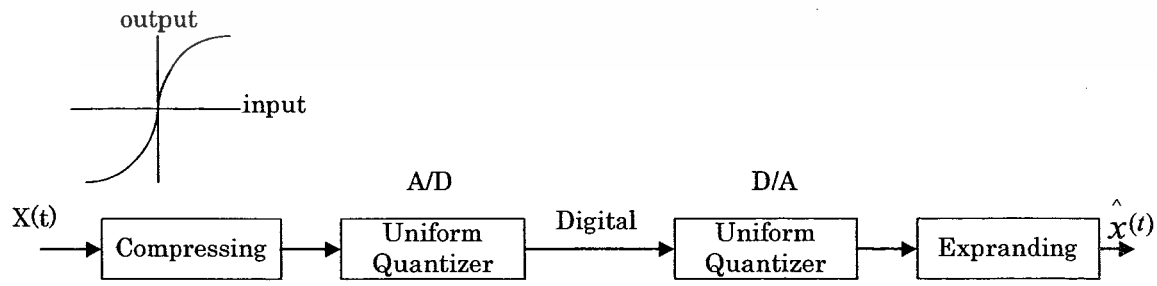
$$SNR = \frac{\overline{x^2}}{e_n^2} = 12\frac{\overline{x^2}}{\Delta x^2}$$

เมื่อคิดเทียบกับอินพุต

$$\overline{x^2} = \frac{x_{\max}^2}{3} \qquad ,\Delta x = \frac{2x_{\max}}{2^N}$$

$$SNR = 2^{2N} \qquad ,\mathrm{N} = \text{จำนวนบิต}$$

Non - uniform quantization จาก error(en) ที่ได้ในหัวข้อก่อนมีค่าเป็นไปได้ $-\Delta x/2$ถึง $+\Delta x/2$ ทุกช่วงของการแบ่งระดับเท่ากันพิจารณาที่ สัญญาณมีค่าต่ำๆ ค่า error คงเท่าเดิมค่า SNR จะแย่ลงยกตัวอย่างสัญญาณ 8 โวล์ต แบ่งระดับเป็น 8 ระดับ ระดับละ 1 โวล์ตเท่ากันทุกช่วงค่า error โวล์ต $+\Delta x/2 = 0.5$ เมื่อสัญญาณ 5.5 โวล์ตเข้ามาตัวสัญญาณขาออกที่ได้ จากการแปลงสัญญาณกลับมีค่าเท่ากับ 5 โวล์ต (ไม่มี ระดับ 5.5 โวลต์) ส่วนที่แตกต่างจากสัญญาณเดิมเรียก ว่าเป็นสัญญาณรบกวนเท่ากับ 0.5 โวลต์ ได้ SNR =5/0.5 แต่เมื่อสัญญาณเป็น 1.5 โวล์ต จะได้ SNR =1/0.5 จะเห็นว่า SNR จะแตกต่าง กันมากและมี ผลที่ สัญญาณต่ำๆ นั่นคือผลของการ quantize ที่ ใช้ระดับเท่ากันทุกระดับที่ เรียก ว่า uniform quantization จึงมี การพัฒนาวิธี การ quantize โดยพิจารณาช่ วงสัญญาณต่ำๆ ให้มีการแบ่งระดับละเอียดขึ้นและช่วงสัญญาณสูงแบ่งระดับหยาบขึ้น เรียกว่า non - uniform quantization การทำ non - uniform quantization นั้นทำได้ โดยการแปลงสัญญาณที่เข้ามาให้มีค่าเปลี่ยนไปในลักษณะ nonlinearคือที่ สัญญาณต่ำๆ ให้มีการขยายสัญญาณออก ในขณะที่สัญญาณที่สูง อยู่แล้วก็ขยายน้อยเรียก ขั้นตอนนี้ ว่าการ compressingสัญญาณจากนั้นจึงผ่านเข้า uniform quantizer เพื่อเปลี่ยนเป็น สัญญาณดิจิตอล ในทางกลับกันเมื่อแปลงกลับเป็นสัญญาณอนาลอกแล้วต้องแปลงสัญญาณโดยการขยายที่ ตรงข้ามกับตอนแรก เรียก ขั้นตอนนี้ ว่าการ expanding รวมเรียก ขบวนการเพื่อทำ non - uniform quantization แบบนี้ ว่า companding ดังรูปที่ 2.12 แสดงการทำ companding



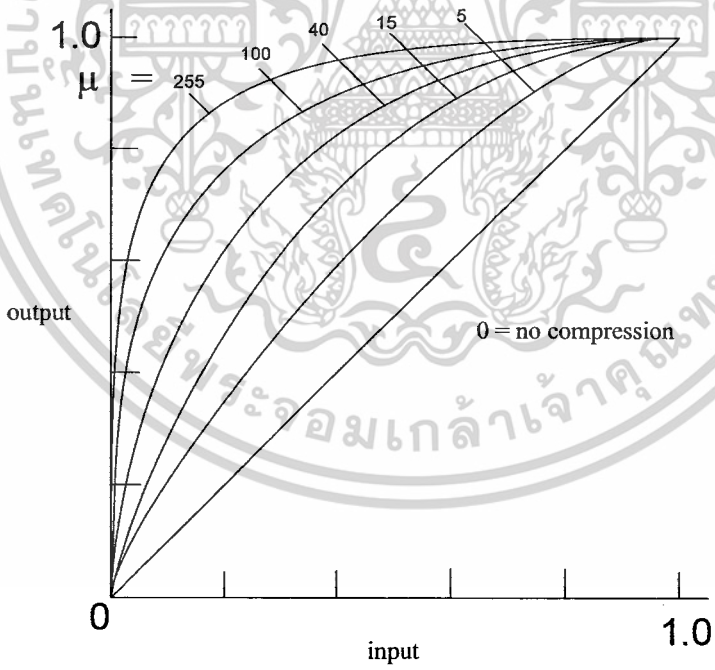
รูปที่ 2.12 Companding

ความสัมพันธ์ของสัญญาณขาเข้าและสัญญาณขาออกของการ compress สัญญาณอธิบายได้โดยเส้นโค้งความสัมพันธ์ซึ่งมี มาตรฐานอยู่สองมาตรฐานคือ A-law มาตรฐานยุโรปและ μ- law มาตรฐานของอเมริกาและญี่ปุ่นมีความสัมพันธ์ ดังสมการ

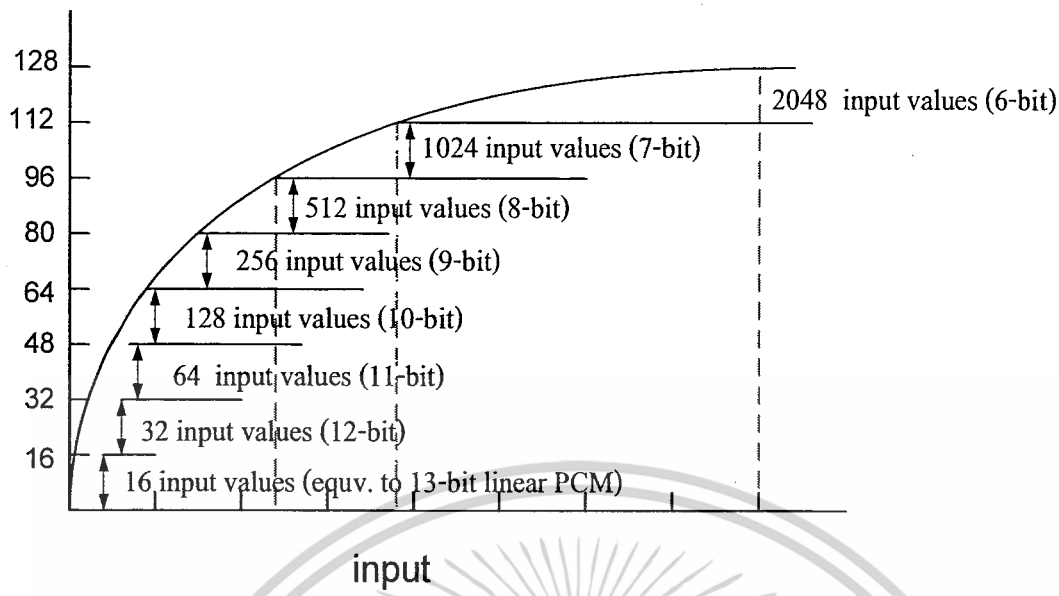
$$F_{\mu}(x) = \operatorname{sgn}(x)\frac{\ln(1+\mu|x|)}{\ln(1+\mu)}$$

$$F_{A}(x) = \begin{cases} \operatorname{sgn}(x)\left(\dfrac{A|x|}{1+\ln A}\right) & ;0 \le |x| < \dfrac{1}{A} \\ \operatorname{sgn}(x)\left(\dfrac{1+\ln|Ax|}{1+\ln A}\right) & ;\dfrac{1}{A} \le |x| < 1 \end{cases}$$

มาตรฐานที่ใช้สำหรับ PCM มาตรฐานจำนวนบิตเท่ากับ 8 บิต โดยการแบ่งช่วงสัญญาณเข้าออกเป็น 8 ช่วง ในแต่ละด้านบวก ลบของสัญญาณ ในแต่ละช่วงจะประมาณด้วย uniform quantization 8 บิตที่ เข้ารหัสนี้ จะแบ่งออกเป็นบิตแรก ใช้สำหรับแยกสัญญาณด้านบวกกับด้านลบ 3 บิต ต่อมาเป็นการเลือกช่วงบวก 8 ช่วง ลบ 8 ช่วง อีก 4 บิตเป็นการเลือกจุด 16 จุดในแต่ละช่วง



รูปที่ 2.13 แสดงกราฟของความสัมพันธ์ ของ μ- law ที่ μ ค่าต่าง ๆ



รูปที่ 2.14 การประมาณ non - uniform quantization ด้วย uniform quantization

การประมาณ non - uniform quantization ด้วย uniform quantization เป็นช่วงๆ จากการทำ companding ของ non - uniform quantization เป็นผลให้ error ที่ เกิดจากการ quantize ลดลง จากค่าเฉลี่ยของ error ในสมการ

$$mse = \frac{\Delta x^2}{12} \int_x \frac{p(x)}{[F'(x)]^2} dx$$

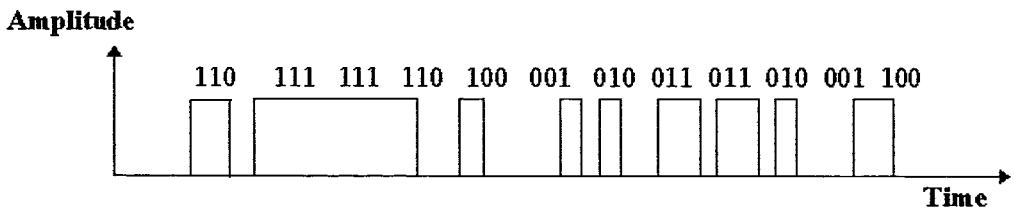
โดยที่ p(x) เป็น พรอบอะบิลิตี้เดนซิตี้ ฟังก์ชันของสัญญาณ x และ F'(x) เป็น ดิฟเฟอเรนชิเอตของ compression function จะเห็นว่าถ้าส่วนอินทีกรัลมีค่าน้อยกว่าหนึ่ง จะได้ mse ของ non - uniform quantization น้อยกว่า mse กรณี uniform quantization

3) การเข้ารหัส

จะเป็นขวบการที่ใช้แปลงระดับของควอนไทซ์เซชั่นแต่ละสเต็ปให้เป็นไบนารี่ ซึ่งเราสามารถที่จะสร้างความสัมพันธ์โดยเขียนสูตรได้ดังนี้

จำนวนสเต็ปของควอนไทซ์เซชั่น = $2^{(\text{จำนวนของบิตที่เข้ารหัส})}$

จากสมการนี้เราจะเห็นได้ว่าจำนวนของบิตที่เข้ารหัสที่เพิ่มขึ้นจะเป็นการแก้ไขอัตราส่วนของควอนไทซ์ซึ่งน้อยส์ให้ดีขึ้น



รูปที่ 2.15 การเข้ารหัส Coding

2.3 วงจรกรองความถี่

วงจรกรองความถี่ นับเป็นวงจรอิเล็กทรอนิกส์ที่มีความสำคัญมาก ไม่ว่าทางด้านอิเล็กทรอนิกส์ , การสื่อสาร, การควบคุม และทางด้านเครื่องมือแพทย์ ในทุกวันนี้วงจรกรองความถี่แบ่งออกเป็นสองรูปแบบคือ วงจรกรองความถี่แบบแอนาลอก (analog filter) กับวงจรกรองความถี่แบบดิจิตอล (digital filter)

2.3.1 วงจรกรองความถี่แบบแอนาลอก(Analog filter)

วงจรกรองความถี่แบบแอนาลอก(Analog filter) ประกอบขึ้นด้วย R (resistor) L (inductors) และ C (capacitors) โดยเอาคุณสมบัติประจำตัวของอุปกรณ์แต่ละชนิด คือ L จะยอมให้ความถี่ต่ำผ่านได้ง่าย ความถี่สูงผ่านยาก C ความถี่ต่ำผ่านยาก ความถี่สูงผ่านง่าย ส่วน R จะมีต้านทานทุกความถี่ให้มีระดับสัญญาณลดลง วงจรกรองความถี่ จะมีทั้งแบบ passive และ active วงจรแบบ active นั้นจะมีวงจรขยายสัญญาณอยู่ภายในมักจะใช้กันที่ความถี่ต่ำ ๆ เช่น ในวงจรเครื่องขยายเสียง ชนิดของวงจรกรองความถี่มี 4 แบบคือ

- วงจรกรองความถี่ต่ำผ่าน (low-pass filter; LPF)
- วงจรกรองความถี่สูงผ่าน (high-pass filter; HPF)
- วงจรกรองแถบความถี่ผ่าน (Band-pass filter; BPF)
- วงจรกรองแถบความถี่หยุดผ่าน (Band-stop filter; BSF)

วงจรกรองความถี่ต่ำผ่าน (low - pass filter; LPF)

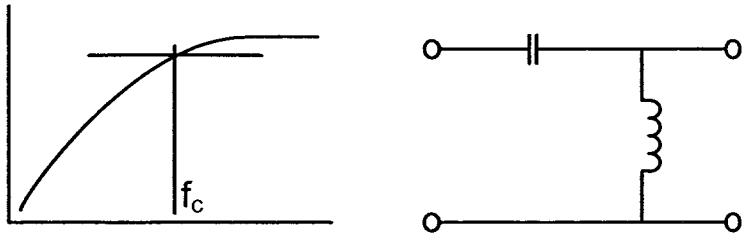
บางครั้งอาจจะเรียกว่าวงจร high-cut filter สำหรับ ความถี่วิทยุ และ treble cut filter สำหรับวงจรขยายเสียง

รูปที่ 2.16 Low-pass filter; LPF

วงจร low pass filter มีลักษณะการต่อคือ ใช้ L อนุกรมกับวงจร และ C ขนานกับวงจร คุณสมบัติของวงจรก็คือ เมื่อเราป้อนความถี่ ต่ำเข้าวงจร L จะมีค่า XL ต่ำ C จะมีค่า XC สูง ทำให้ความถี่ ต่ำผ่าน L ได้สะดวก ระดับสัญญาณ Output จึงผ่านได้มาก แต่เมื่อความถี่สูงกว่าจุดที่กำหนด ค่า XL จะมากขึ้น ค่า XC จะลดลง ทำให้ความถี่ ผ่านขดลวดได้ลดลง บางส่วนที่ผ่านไปได้ก็จะถูก C ดึงลงกราวด์ ระดับสัญญาณ Output จึงผ่านได้น้อยมาก

วงจรกรองความถี่สูงผ่าน (high - pass filter; HPF)

บางครั้งอาจจะเรียกว่าวงจร Low-cut filter สำหรับ ความถี่วิทยุ และ bass-cut filet สำหรับวงจรขยายเสียง



รูปที่ 2.17 High-pass filter; HPF

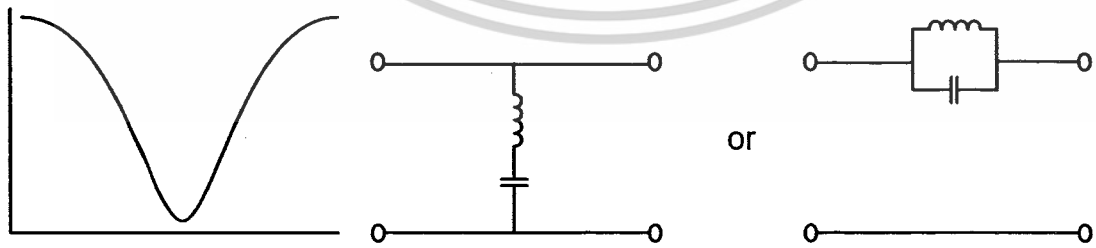
วงจรนี้จะยอมให้ความถี่ที่สูงกว่ากำหนดผ่านไปได้ ส่วนความถี่ที่ต่ำกว่าจะโดนจับลง กราวด์ จากรูปจะเห็นว่า C ต่ออนุกรมกับวงจร ส่วน L ต่อขนานกับวงจร เมื่อป้อนความถี่ต่ำกว่าเข้ามา C จะมีค่า XC สูง ทำให้สัญญาณผ่านไปได้น้อย ส่วน L จะมีค่า XL น้อย ทำให้สัญญาณที่ผ่านมาจาก C ลงกราวด์ได้หมด แต่เมื่อความถี่สูงขึ้น C จะมีค่า XC ลดลง สัญญาณจะผ่านได้มากขึ้น ส่วน L จะมีค่า XL มากขึ้น สัญญาณก็จะลงกราวด์น้อยลง สัญญาณที่ออกไปยัง Output ก็มากขึ้น จนถึงระดับความแรงของสัญญาณประมาณ 70.7 % ของความแรงสูงสุด ระดับนี้เองที่เราเรียกว่า ช่วงความถี่ Cut off เมื่อความถี่สูงกว่า ความถี่ นี้ C จะยอมให้สัญญาณผ่านได้สะดวกและค่า XL จะต้านสัญญาณไม่ให้ลงกราวด์ ความถี่จึงผ่านไปที่จุด Output ได้ทั้งหมด

วงจรกรองแถบความถี่ผ่าน (Band-pass filter; BPF)

รูปที่ 2.18 Band-pass filter; BPF

วงจรกรองความถี่แบบ Band pass filter จะยอมให้ความถี่เฉพาะช่วงที่กำหนดให้ผ่านได้ควาถี่นอกจากนี้จะโดนตัดลงกราวด์ เราสามารถสร้างวงจร band pass filter โดยการใช้วงจร Resonance

วงจรกรองแถบความถี่หยุดผ่าน (Band-stop filter; BSF)



รูปที่ 2.19 Band-stop filter; BSF

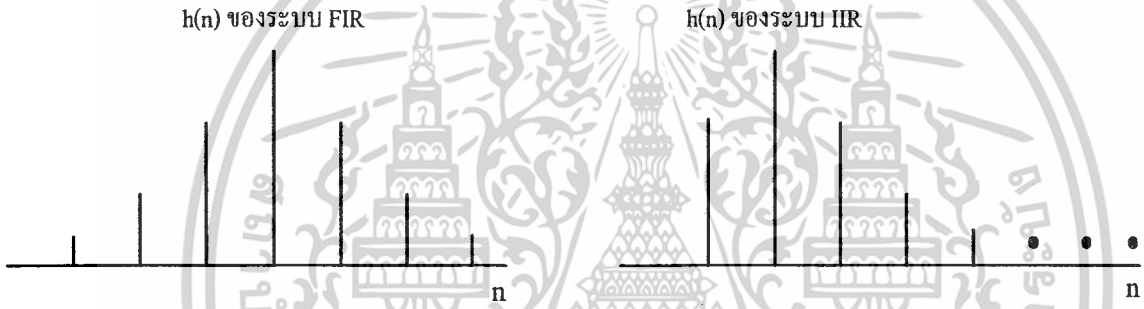
วงจรกรองความถี่แบบ Band stop filter จะยอมให้ความถี่อื่น ๆ ผ่านไปได้สะดวก แต่สำหรับความถี่ Resonance (ความถี่ที่จะกำจัด) จะโดนดึงลงกราวด์

วงจรกรองความถี่แบบแอนาลอก ซึ่งประกอบไปด้วย ตัวความต้านทาน, ตัวเก็บประจุ, ตัวเหนี่ยวนำ และอุปกรณ์กึ่งตัวนำเช่นออปแอมป์ ข้อดีคือออกแบบได้ง่าย ราคาถูก แต่มีข้อเสียที่วงจรขาดเสถียรภาพ (stability) ความถี่ที่ต้องการมีความคลาดเคลื่อนสูง

2.3.2 วงจรกรองความถี่แบบดิจิตอล(Digital filter)

วงจรกรองความถี่แบบดิจิตอล(Digital filter) คือ การประมวลผลสัญญาณในโดเมนเวลาเพื่อดัดแปลงผลตอบสนองทางความถี่ในทางขนาด และ/หรือ เฟส

การแบ่งประเภทของวงจรกรองความถี่แบบดิจิตอล จะแบ่งตามผลตอบสนองอิมพัลส์ของระบบตามรูปที่ 2.20 คือผลตอบสนองอิมพัลส์จำนวนจำกัด (Finite Impulse Response; FIR) กับผลตอบสนองอิมพัลส์จำนวนไม่จำกัด (Infinite Impulse Response; IIR) แต่ในที่นี้ขอยกตัวอย่างการออกแบบวงจรกรองความถี่แบบดิจิตอลที่มีผลตอบสนองอิมพัลส์จำนวนจำกัด หรือ FIR เพราะมีลักษณะเด่นคือ มีผลตอบสนองทางเฟสแบบเชิงเส้น (linear phase) เหมาะสำหรับการพัฒนาเครื่องมือวัดทางด้านการแพทย์



รูปที่ 2.20 ผลตอบสนองต่อสัญญาณอิมพัลส์ของวงจรกรอง FIR และ IIR

2.4 การแปลงฟูริเยร์เต็มหน่วย (DFT)

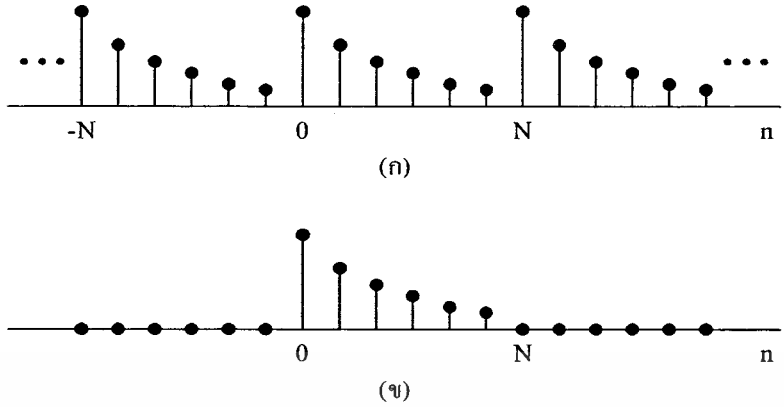
ผลการแปลงฟูริเยร์เต็มหน่วยของสัญญาณ x(n) พิจารณาได้จากอนุกรมฟูริเยร์เต็มหน่วย (discrete Fourier series หรือ DFS) ของสัญญาณเป็นคาบ $\tilde{x}(n)$ ที่มีคาบเท่ากับ N แสดงได้ดังสมการ (1)

$$\tilde{x}(n) = \frac{1}{N}\sum_{k=0}^{N-1} \tilde{X}(k)\, e^{j(\frac{2\pi}{N})kn} \qquad (1)$$

โดยที่ $\tilde{X}(k)$ คือค่าสัมประสิทธ์ของอนุกรมฟูริเยร์ ซึ่งคำนวณได้จาก $\tilde{x}(n)$ ตามความสัมพันธ์ดังสมการ (2)

$$\tilde{X}(k) = \sum_{n=0}^{N-1} \tilde{x}(n)\, e^{-j(\frac{2\pi}{N})kn} \qquad (2)$$

เมื่อพิจารณา x(n) ให้เป็นหนึ่งคาบของ $\tilde{x}(n)$ แสดงตัวอย่างดังรูปที่ 2.21



รูปที่ 2.21 ตัวอย่างของ (ก) สัญญาณ $\tilde{x}(n)$ และ (ข) สัญญาณ x(n)

ค่าสัมประสิทธิ์ฟูริเยร์ของ x(n) จึงมีค่าเท่ากับหนึ่งคาบของ $\tilde{X}(k)$ ซึ่งเรียกว่า ผลการแปลงฟูริเยร์เต็มหน่วย หรือ DFT (Oppenheim and Schafer, 1989) แสดงดังสมการ (3)

$$X(k)=\sum_{n=0}^{N-1} x(n)\, e^{-j(\frac{2\pi}{N})kn}, \quad 0\le k\le N-1 \tag{3}$$

สเปกตรัมขนาดของ X(k) จะแสดงองค์ประกอบทางความถี่ของสัญญาณ x(n) ที่ให้พลังงานสูงสุด หรือเรียกว่า ฟอร์แมนท์ (formant) ในกรณีสัญญาณเสียงดนตรี (สราวุฒิ สุจิตจร, 2545) นำอัลกอริทึมการแปลงฟูริเยร์อย่างรวดเร็ว (fast Fourier transform หรือ FFT) มาใช้ในการคำนวณ DFT โดยกำหนดให้ N เป็นค่ากำลังของสอง และ $W_N = e^{-j(2\pi/N)}$ จะได้ X(k) มีค่าตามสมการ (4)

$$X(k)=\sum_{n=0}^{N-1} x(n)\, W_N^{kn}, \quad 0\le k\le N-1 \tag{4}$$

อัลกอริทึม FFT จะทำการแบ่ง x(n) ออกเป็นสองลำดับเท่าๆกัน คือลำดับของเลขคู่ (แทน n = 2r) และลำดับของเลขคี่ (แทน n = 2r + 1) โดยที่ $0 \le r \le (N/2) - 1$ ดังสมการ (5)

$$X(k)=\sum_{r=0}^{(N/2)-1} x(2r)\, W_N^{2rk}+\sum_{r=0}^{(N/2)-1} x(2r+1)\, W_N^{(2r+1)k} \tag{5}$$

ให้ $W_N^2 = W_{N/2}$ จะได้ DFT จำนวน N จุด เป็นผลรวมของ DFT จำนวน N / 2 จุด สองลำดับ แสดงดังสมการ (6)

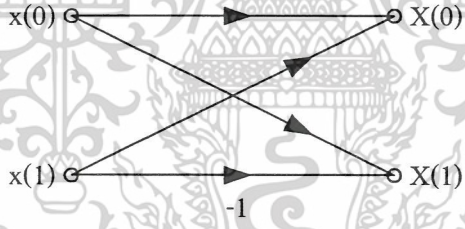
$$X(k) = \sum_{r=0}^{(N/2)-1} x(2r)\ W_{N/2}^{rk} + W_N^k \sum_{r=0}^{(N/2)-1} x(2r+1)\ W_{N/2}^{rk} \qquad (6)$$

ซึ่งแต่ละลำดับสามารถแยกออกเป็นผลรวมของ DFT จำนวน N / 4 จุด สองลำดับ จนกระทั่งแยกได้เป็น DFT จำนวน 2 จุด N / 2 ลำดับ กระบวนการจึงจะสิ้นสุด การคำนวณ DFT จำนวน 2 จุด แสดงได้ดังสมการ (7) และเนื่องจาก $W_1^{0k} = 1$ และ $W_2^k = (-1)^k$ จะได้ค่าของ X(k) แสดงดังสมการ (8) และ (9) แทนด้วยกราฟที่เรียกว่า บัตเตอร์ฟลาย (butterfly) ได้ดังรูปที่ 2.22 และการคำนวณ DFT จำนวน 4 จุด แสดงดังรูปที่ 2.23 (Oppenheim and Schafer, 1989)

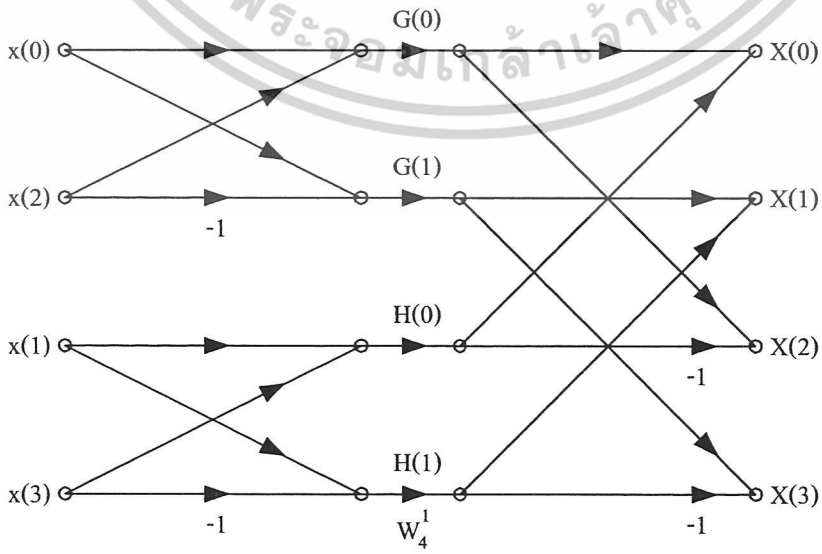
$$X(k) = x(0)\ W_1^{0k} + W_2^k [\, x(1)\ W_1^{0k}\,] \quad , \ k = 0, 1 \qquad (7)$$

$$X(0) = x(0) + x(1) \qquad (8)$$

$$X(1) = x(0) - x(1) \qquad (9)$$



รูปที่ 2.22 กราฟของการคำนวณ DFT จำนวน 2 จุด (บัตเตอร์ฟลาย)

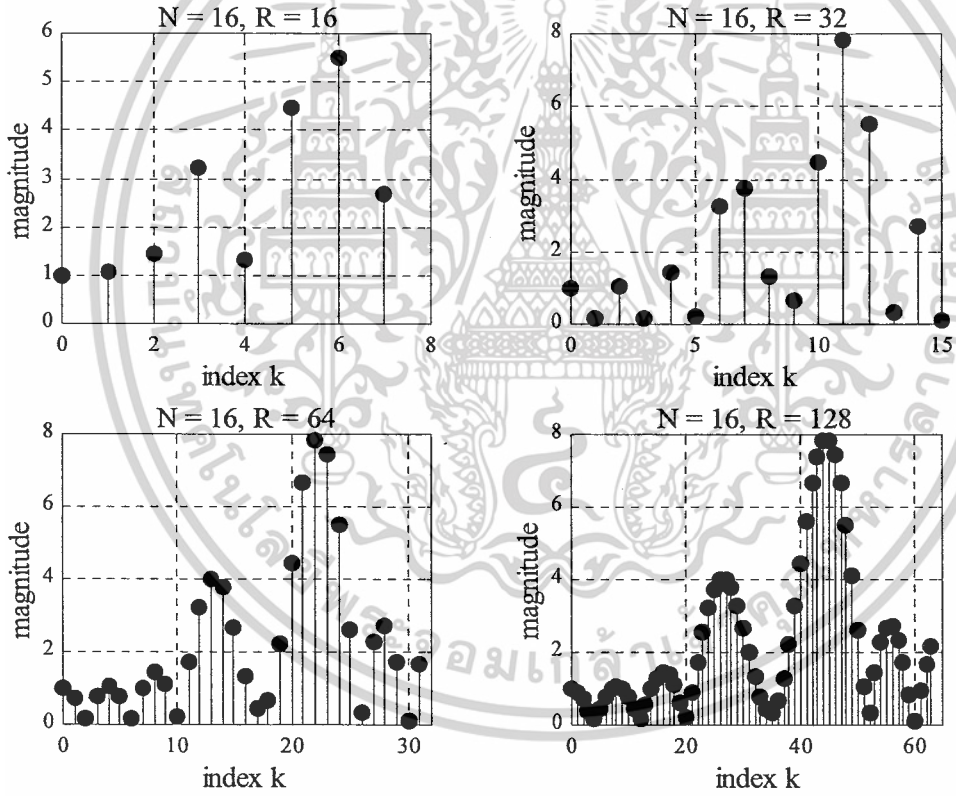


รูปที่ 2.23 กราฟของการคำนวณ DFT จำนวน 4 จุด

สำหรับการวิเคราะห์สัญญาณด้วย DFT การเพิ่มความถูกต้องของตำแหน่งเส้นสเปกตรัม ทำได้โดยเพิ่มขนาดของการคำนวณ DFT ให้เป็น R จุด ซึ่งจำนวนจุดที่เพิ่มขึ้นจะเป็นค่าศูนย์ ตัวอย่างต่อไปนี้แสดงการคำนวณ DFT ของสัญญาณไซนูซอยด์ y(n) มีค่าตามสมการ (10) โดยกำหนดให้ N = 16, f1 = 0.22 Hz, f2 = 0.34 Hz และ R มีค่าเปลี่ยนแปลงจาก 16 จุด ถึง 128 จุด

$$y(n) = (\tfrac{1}{2}) \sin(2\pi f_1 n) + \sin(2\pi f_2 n) \quad , \quad 0 \leq n \leq N-1 \qquad (10)$$

จากรูปที่ 2.30 จะสังเกตได้ว่า ที่ R เท่ากับ 64 จุด ขนาดสูงสุดปรากฏที่ตำแหน่ง k เท่ากับ 13 และ 22 ซึ่งตรงกับความถี่ 0.2031 Hz และ 0.3438 Hz ตามลำดับ และที่ R เท่ากับ 128 จุด ขนาดสูงสุดปรากฏที่ตำแหน่ง k เท่ากับ 27 และ 44 ซึ่งตรงกับความถี่ 0.2109 Hz และ 0.3438 Hz ตามลำดับ นั่นคือเมื่อ R เพิ่มมากขึ้น สเปกตรัมของ y(n) ปรากฏขนาดสูงสุดที่สองความถี่อย่างชัดเจน และมีค่าใกล้เคียงกับความถี่จริงของสัญญาณ (Mitra, 2001)



รูปที่ 2.24 ผลการวิเคราะห์สัญญาณ y(n) ด้วย DFT

2.5 แปลงฟูริเยร์แบบเร็ว Fast Fourier Transform (FFT)

FFT เป็นชื่อเรียกโดยรวมๆของ อัลกอริธึมใดๆ ที่มีการแปลง DFT อย่างเร็ว วิธี"แบ่งแยกแล้วปกครอง (Divide and conquer)" ก็เป็นหนึ่งวิธีที่จะลดจำนวนการคูณเลขเชิงซ้อนลง ใช้ การแบ่งทางเวลา (Decimation in time) กับ N สัญญาณโดเมนเวลา โดยที่ N เป็นเลขกำลังของ 2 หรือเรียกว่า Radix-2 ดังนั้นชื่อเต็มเรียกว่า Radix-2 DIT-FFT

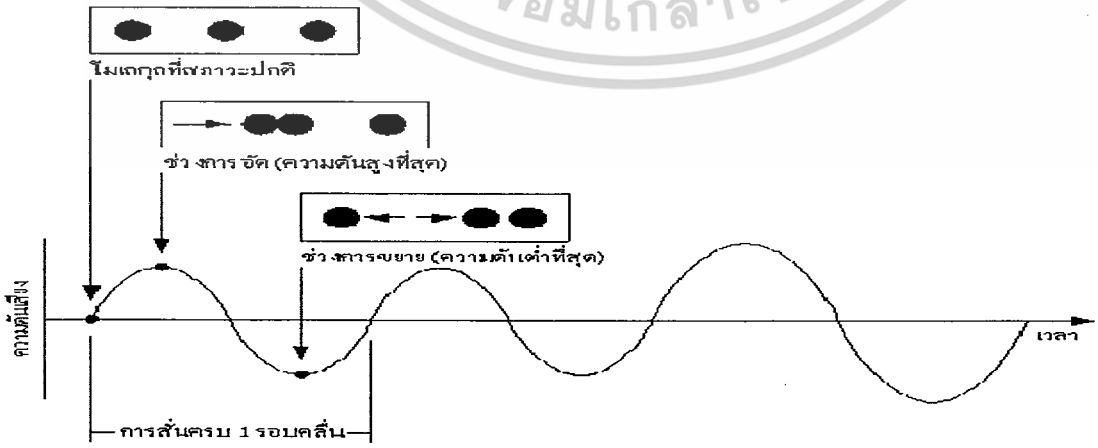
FFT ก็คือ DFT แต่เป็นการสลับตำแหน่งข้อมูลและเทคนิกการรวมสัญญาณ เพื่อย่อยให้จำนวนการแปลงลดรูปลง วิธีการนี้ เรียกว่า Decimation in Time (DIT) และเรียก การแปลงฟูริเยร์แบบเร็วนี้ว่า DIT-FFT การแปลงฟูริเยร์แบบเร็ว (FFT) แบบจะทำให้เหลือการคูณเลขเชิงซ้อนเหลือเพียง Nlog2N ครั้ง จาก N2 ครั้ง เมื่อใช้ DFT หรืออาจจะลดการคูณเลขเชิงซ้อนลงได้อีกเป็น (N/2) log2N หากใช้การปรับปรุงบัตเตอร์ฟลาย

2.6 เสียง

เสียง คือ คลื่นเสียงจะเปลี่ยนไปตามขนาด (Amplitude) และความถี่ (Frequency) ของการสั่นสะเทือนตามระยะเวลา

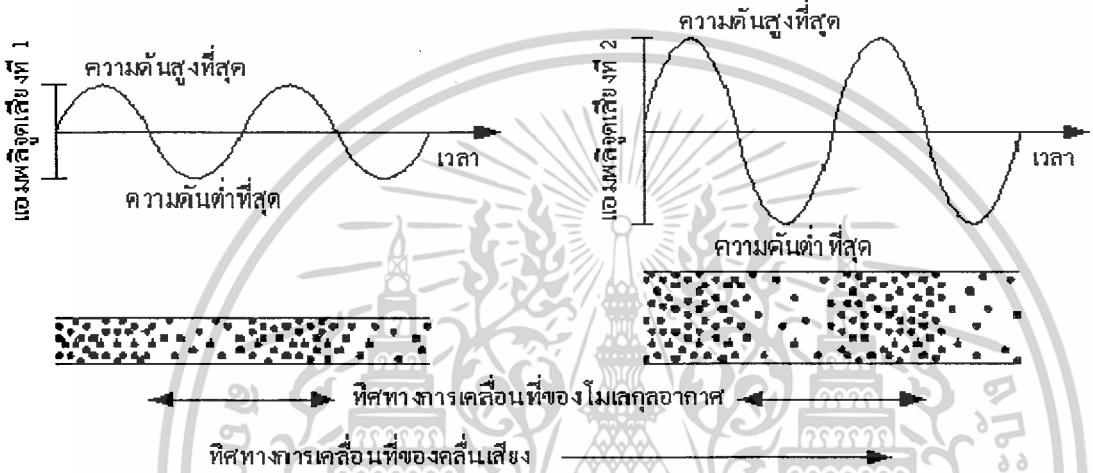
การเกิดเสียง

"เสียง เริ่มเกิดขึ้นเมื่อวัตถุหรือแหล่งกำเนิดเสียงมีการสั่นสะเทือน ส่งผลต่อการเคลื่อนที่ของโมเลกุลของอากาศที่อยู่โดยรอบ กล่าวคือโมเลกุลของอากาศเหล่านี้จะเคลื่อนที่จากตำแหน่งเดิม ไปชนกับโมเลกุลที่อยู่ถัดไป ก่อให้เกิดการถ่ายโอนโมเมนตัม จากโมเลกุลที่มีการเคลื่อนที่ให้กับโมเลกุลที่อยู่ในสภาวะปกติ จากนั้นโมเลกุลที่ชนกันนี้จะแยกออกจากกัน โดยโมเลกุลที่เคลื่อนที่มาจะถูกดึงกลับไปยังตำแหน่งเดิมด้วยแรงปฏิกิริยา และโมเลกุลที่ได้รับการถ่ายโอนพลังงานจะเคลื่อนที่ไปชนกับโมเลกุลที่อยู่ถัดไป ปรากฏการณ์นี้จะเกิดขึ้นสลับกันไปมาได้เมื่อสื่อกลาง (ในที่นี้คืออากาศ) มีคุณสมบัติของความยืดหยุ่น (Alten, 1999) การเคลื่อนที่ของโมเลกุลอากาศนี้จึงเกิดเป็นคลื่นเสียง แสดงการเคลื่อนที่ของโมเลกุลอากาศ เทียบกับลักษณะของคลื่น เมื่อแหล่งกำเนิดเสียงมีการสั่นสะเทือน อาจสังเกตได้จากภาพว่า ขณะที่แหล่งกำเนิดเสียงไม่มีการสั่นสะเทือน หรือโมเลกุลของอากาศอยู่ในสภาวะปกติ ความดันเสียง (sound pressure) ในขณะนี้จะคงที่ที่ค่าหนึ่ง เมื่อโมเลกุลของอากาศมีการชนกัน ความดันอากาศจะมีค่าเพิ่มมากขึ้นจากปกติ ส่งผลให้ความดันเสียง ณ ช่วงเวลานี้เพิ่มมากขึ้นด้วย เสมือนเป็นช่วงการอัด (compression) เกิดเป็นยอดคลื่นที่มีความดันเสียงสูงที่สุดในคลื่นเสียง และเมื่อโมเลกุลของอากาศแยกออกจากกัน ความดันอากาศจะมีค่าลดลงจากปกติ ส่งผลให้ความดันเสียง ณ ช่วงเวลานี้ลดลงด้วย เสมือนเป็นช่วงการขยาย (rarefaction) เกิดเป็นจุดที่มีความดันเสียงต่ำที่สุดในคลื่นเสียง (Alten, 1999)



รูปที่ 2.25 การเคลื่อนที่ของโมเลกุลของอากาศเทียบกับลักษณะของคลื่น

ดังนั้น การสั่นสะเทือนของแหล่งกำเนิดเสียงจากช่วงการอัดถึงช่วงการขยาย จึงเปรียบได้กับการเคลื่อนที่ครบหนึ่งรอบ (cycle) ของคลื่น ซึ่งจำนวนรอบในเวลาหนึ่งวินาทีนี้หมายถึงความถี่ (frequency) ของคลื่นเสียง มีหน่วยเป็นเฮิรตซ์ (Hz) หรือรอบต่อวินาที (cps) และนอกจากนี้จำนวนโมเลกุลของอากาศที่เคลื่อนที่ทั้งในช่วงการอัดและช่วงการขยายของโมเลกุลของอากาศ ยังหมายถึงแอมพลิจูด (amplitude) ของคลื่นเสียงด้วย เช่นถ้าจำนวนโมเลกุลของคลื่นเสียงที่หนึ่ง น้อยกว่าของคลื่นเสียงที่สอง ยอดคลื่นของเสียงที่หนึ่ง ย่อมจะต่ำกว่ายอดคลื่นของเสียงที่สอง ซึ่งทำให้แอมพลิจูดของเสียงที่หนึ่ง ต่ำกว่าแอมพลิจูดของเสียงที่สอง เป็นต้น รูปที่ 2.26 แสดงจำนวนโมเลกุลของอากาศกับแอมพลิจูดของคลื่นเสียง



รูปที่ 2.26 จำนวนโมเลกุลของอากาศกับแอมพลิจูดของคลื่นเสียง

เมื่อพิจารณาจากภาพดังกล่าวจะเห็นได้ว่าเสียงเป็นคลื่นตามยาว (longitudinal wave)เนื่องจากโมเลกุลของอากาศเคลื่อนที่ในทิศทางเดียวกันกับทิศทางการเคลื่อนที่ของคลื่นเสียง คลื่นเสียงจะเคลื่อนที่ออกห่างจากแหล่งกำเนิดเสียงมีลักษณะคล้ายกับคลื่นที่เกิดขึ้นเมื่อโยก้อนหินลงในน้ำ รูปคลื่นที่ใช้แทนคลื่นเสียงจากรูปที่ 2.25 และ รูปที่ 2.26 นั้นเป็นรูปคลื่นไซน์ ประกอบด้วยพลังงานที่มีเพียงความถี่เดียว จึงเรียกว่าเสียงบริสุทธิ์ (pure tones) ซึ่งในความเป็นจริงแล้ว เสียงที่มนุษย์ได้ยินมิใช่ในรูปของเสียงบริสุทธิ์ แต่อยู่ในรูปของคลื่นความถี่ต่างๆผสมกัน เพราะเสียงที่มนุษย์สามารถได้ยิน เกิดจากแหล่งกำเนิดเสียงที่มีการสั่นในหลายลักษณะหรือมีหลายความถี่รวมกัน ทำให้รูปคลื่นที่เกิดขึ้นมีความซับซ้อนมากกว่าคลื่นรูปไซน์ (Rumsey and McCormick, 1994)"

การวัดระดับของคลื่นเสียง มีหน่วยวัด 2 หน่วย คือ

- เดซิเบล (Decibel) เป็นหน่วยวัดความดังของเสียง
- เฮิรตซ์ (Hertz: Hz) เป็นหน่วยวัดความถี่ของเสียง โดยสามารถแสดงระดับความดังและชนิดของเสียงได้

ความดังของเสียง (เดซิเบล)	ชนิดของเสียง
0	เสียงที่แผ่วเบาที่สุดที่หูมนุษย์ได้ยิน
30	เสียงกระซิบ หรือเสียงในห้องสมุดที่เงียบสงัด
60	เสียงพูดคุยตามปกติ เสียงจักรเย็บผ้า หรือเสียงเครื่องพิมพ์ดีด
85	เสียงตะโกนข้ามเขา หรือพื้นที่โล่งกว้าง เพื่อให้ได้ยินเสียงสะท้อนของตนเองกลับมา
90	เสียงเครื่องตัดหญ้า เสียงเครื่องจักรในโรงงาน หรือเสียงรถบรรทุก (ไม่ควรได้ยินเกินวันละ 8 ชม.)
100	เสียงเลื่อยไฟฟ้า หรือเครื่องเจาะที่ใช้ลม : Pneumatic Drill (ไม่ควรได้ยินเกินวันละ 2 ชม.)
115	เสียงระเบิดหิน เสียงในร็อคคอนเสิร์ต หรือเสียงแตรรถยนต์ (ไม่ควรได้ยินเกินวันละ 15 นาที)
140	เสียงยิงปืน เสียงเครื่องบินเจ็ท ซึ่งเป็นเสียงที่ทำให้ปวดหู และอาจทำให้หูเสื่อมได้ แม้จะได้ยินเพียงครั้งเดียวก็ตาม ดังนั้นผู้ที่จำเป็นต้องอยู่กับเสียงในระดับนี้ จะต้องสวมอุปกรณ์ป้องกันหูเสมอ

ตารางที่ 2.1 แสดงระดับความดังและชนิดของเสียง

2.6.1 ประเภทของเสียง

แบ่งออกเป็น 2 ชนิด

- เสียงแบบ MIDI
- เสียงแบบดิจิตอล

MIDI คือ ข้อมูลแสดงลักษณะเสียงแทนเครื่องดนตรีชนิดต่างๆ ซึ่งเป็นมาตรฐานในการสื่อสารด้านเสียงที่ได้รับการพัฒนามาตั้งแต่ปี ค.ศ.1980 สำหรับใช้กับเครื่องดนตรีอิเล็กทรอนิกส์และคอมพิวเตอร์ เช่น สร้างเสียงตามตัวโน้ต เสมือนการเล่นของเครื่องเล่นดนตรีนั้นๆ โดยในมุมมองของนักดนตรี MIDI จะหมายถึง โน้ตเพลงที่มีรูปแบบเป็นสัญลักษณ์หรือตัวเลข ที่จะบอกให้รู้ว่าต้องเล่นโน้ตตัวใดในเวลานานเท่าไร เพื่อให้เกิดเป็นเสียงดนตรี

ดนตรีแบบ MIDI จะไม่เหมือนเสียงจากเครื่องดนตรีจริงๆ ดังนั้นเครื่องมือในการเล่นเพลงแบบ MIDI จะมีผลต่อคุณภาพของเสียงที่ได้ ดังนั้นจึงมีความจำเป็นในการสร้างและปรับแต่งเสียง MIDI ให้มีความไพเราะมากยิ่งขึ้น

ข้อดีของ MIDI ไฟล์ข้อมูลมีขนาดเล็ก การสร้างข้อมูล MIDI ไม่จำเป็นต้องใช้เครื่องดนตรีจริงๆ ใช้หน่วยความจำน้อย ทำให้ประหยัดพื้นที่บนฮาร์ดดิสก์ เหมาะสำหรับใช้งานบนระบบเครือข่าย และง่ายต่อการแก้ไขและปรับปรุง

ข้อเสียของ MIDI แสดงผลเฉพาะดนตรีบรรเลงและเสียงที่เกิดจากโน้ตดนตรีเท่านั้น และอุปกรณ์อิเล็กทรอนิกส์ที่ใช้เสียงมีราคาค่อนข้างสูง

มาตรฐาน MIDI (General Midi Standard) ยังเป็นนโยบายของไมโครซอฟต์ในการกำหนดรูปแบบของการสร้างข้อมูลเสียงแบบ MIDI เพื่อให้เป็นมาตรฐานสำหรับอุปกรณ์เล่นเกม (Play Back)

เสียงแบบดิจิตอล คือ สัญญาณเสียงที่ส่งมาจากไมโครโฟน เครื่องสังเคราะห์เสียง เครื่องเล่นเทป หรือจากแหล่งกำเนิดเสียงต่างๆ ทั้งจากธรรมชาติ และที่สร้างขึ้น แล้วนำข้อมูลที่ได้แปลงเป็นสัญญาณดิจิตอล ซึ่งข้อมูลดิจิตอลจะถูกสุ่มให้อยู่ในรูปแบบของบิต และไบต์ โดยเรียกอัตราการสุ่มข้อมูลที่ได้มา เรียกว่า "Sampling Rate" และจำนวนของข้อมูลที่ได้เรียกว่า "Sampling Size" ซึ่งจะเป็นตัวกำหนดคุณภาพของเสียงที่ได้จากการเล่นเสียงแบบดิจิตอล

เสียงแบบดิจิตอลจะมีขนาดของข้อมูลใหญ่ ทำให้ต้องใช้หน่วยความจำและทรัพยากรบนหน่วยประมวลผลกลางมากกว่า MIDI แต่จะแสดงผลเสียงได้หลากหลาย และเป็นธรรมชาติกว่า MIDI มาก

เสียงแบบดิจิตอลที่พบบ่อย จะอยู่ช่วงความถี่ 44.1 KHz, 22.05 KHz และ 11.025 KHz ซึ่งมี Sampling Size เป็น 8 บิต และ 16 บิต โดยที่ Sampling Rate และ Sampling Size ที่สูงกว่าจะให้คุณภาพของเสียงที่ดีกว่า และจะต้องมีเนื้อที่บนฮาร์ดดิสก์สำหรับรองรับอย่างเหมาะสม

Sampling Rate (KHz)	Sampling Size (bit)	Stereo หรือ Mono	จำนวน Byte ที่ใช้ 1 วินาที
44.1	16	Stereo	8.5 Mb
44.1	16	Mono	5.25 Mb
44.1	8	Stereo	5.25 Mb
44.1	8	Mono	2.6 Mb
22.05	16	Stereo	5.25 Mb
22.05	16	Mono	2.5 Mb
22.05	8	Stereo	2.6 Mb
22.05	8	Mono	1.3 Mb
11.025	8	Stereo	1.3 Mb
11.025	8	Mono	0.65 Mb

ตารางที่ 2.2 แสดงคุณสมบัติที่เหมาะสมของสัญญาณเสียงแบบดิจิตอลใน 1 วินาที

2.6.2 การบันทึกเสียง

เป็นการนำเสียงมาจัดเก็บลงในหน่วยความจำเพื่อนำไปใช้งาน เสียงที่ทำงานผ่านคอมพิวเตอร์เป็นสัญญาณดิจิตอลมี 2 รูปแบบ คือ

Synthesize Sound เป็นเสียงที่เกิดจากตัววิเคราะห์เสียง ที่เรียกว่า MIDI โดยเมื่อตัวโน้ตทำงาน คำสั่ง MIDI จะถูกส่งไปยัง Synthesize Chip เพื่อทำการแยกเสียงว่าเป็นเสียงดนตรีชนิดใด ไฟล์ MIDI ที่ได้ก็จะมีขนาดเล็กเนื่องจากเก็บคำสั่งในรูปแบบง่ายๆ

Sound Data เป็นเสียงที่ได้จากการแปลงสัญญาณอนาลอกเป็นสัญญาณดิจิตอล โดยจะมีการบันทึกตัวอย่างคลื่น (Sample) ให้อยู่ที่ใดที่หนึ่งในช่วงของเสียงนั้นๆ และการบันทึกตัวอย่างคลื่นจะเรียงกันเป็นจำนวนมาก เพื่อให้มีคุณภาพที่ดี ซึ่งจะทำให้ขนาดของไฟล์ใหญ่ตามไปด้วย

การบันทึกเสียงแบบสเตริโอ (Stereo Recording) จะให้คุณภาพของเสียงที่ฟังแล้วสมจริงมากขึ้น และเมื่อเปรียบเทียบการบันทึกเสียงแบบโมโน (Mono Recording) กับการบันทึกเสียงแบบสเตริโอ โดยใช้ระยะเวลาในการบันทึกเท่ากัน แฟ้มข้อมูลเสียงแบบสเตริโอที่บันทึกได้จะใช้พื้นที่มากกว่าแฟ้มข้อมูลเสียงแบบโมโน ซึ่งการคำนวณขนาดของแฟ้มข้อมูลที่ได้จากการบันทึกทั้งสองแบบมีดังนี้

การบันทึกเสียงแบบโมโน : Sampling Rate X ระยะเวลาในการบันทึก X (Sampling Size/8) X 1

การบันทึกเสียงแบบสเตริโอ : Sampling Rate X ระยะเวลาในการบันทึก X (Sampling Size/8) X 2

2.7 สัญญาณรบกวน (noise)

Noise คือ สัญญาณที่ไม่พึงประสงค์ที่เข้ามารวมกับสัญญาณเอาท์พุตทำให้ค่าที่ได้เปลี่ยนแปลงไป โดยทั่วไปสัญญาณรบกวนมีหลายชนิดซึ่งเกิดจากหลายสาเหตุแต่ในวงจรชักตัวอย่างและคงค่าที่อาศัยเทคนิคแบบเกตลอยพบสัญญาณรบกวน 2 ชนิด คือ สัญญาณรบกวนที่เกิดจากอุณหภูมิ (Thermal noise) เกิดขึ้นมาเนื่องจากการเคลื่อนที่แบบสุ่มตามอุณหภูมิของอิเล็กตรอนที่วิ่งผ่านตัวนำที่มีความต้านทานภายในวงจรโดยสเปคตรัมกำลัง (Power Spectrum) ของสัญญาณรบกวนนี้จะมีลักษณะที่ เรียบ หรือกล่าวได้ว่าทุกๆ ฮาโมนิก (harmonic) ของสัญญาณรบกวนจะมีค่าพลังงานเท่ากันอย่างต่อเนื่องตลอด ย่านสเปคตรัม บางครั้งจะเรียกสัญญาณรบกวนประเภทนี้ว่า สัญญาณรบกวนขาว (white noise) สัญญาณรบกวนอีกชนิดหนึ่งคือ สัญญาณรบกวนฟลิกเกอร์ (Flicker noise) หรือ สัญญาณรบกวน 1 / f (1/f Noise) จะให้ระดับสัญญาณรบกวนเป็นเส้นกราฟที่มีความชัน 1/f ซึ่งจะมีค่าลดลงเรื่อยๆ เมื่อความถี่เพิ่มมากขึ้นระดับของสัญญาณรบกวนนี้ขึ้นอยู่กับคุณภาพในกระบวนการผลิต

2.7.1 ประเภทของสัญญาณรบกวน

นอยส์ (noise) คือ เสียงรบกวน, สัญญาณรบกวน สัญญาณไฟฟ้าที่ไม่เป็นที่ต้องการที่เกิดขึ้นในช่องทางการสื่อสารของการสื่อสารข้อมูล ซึ่งไม่ใช่สัญญาณที่มีข้อมูลที่เราต้องการ ปกติแล้วช่องทางการสื่อสารทั้งหมดจะมีสัญญาณรบกวนอยู่ด้วยเสมอ แต่ถ้ามีมากเกินไปก็จะทำให้ข้อมูลที่ส่งมาสูญหายไปได้ การส่งข้อมูลทางสายโทรศัพท์มักจะมีสัญญาณรบกวนอยู่ด้วย จึงจำเป็นต้องใช้โปรแกรมการสื่อสารที่สามารถตรวจสอบความผิดพลาดเพื่อให้แน่ใจว่าข้อมูลที่ได้รับนั้นไม่มีส่วนที่เสียไป noise เป็นสัญญาณที่เข้ามาแทรกแซงหรือรบกวน (interfere) นอยส์ที่รับเข้ามาได้ แบ่งเป็น

1. นอยส์บรรยากาศ (atmosphericnoise) เกิดขึ้นจากความแปรปรวนของบรรยากาศที่ห่อหุ้มโลก เช่น ฟ้าแลบ ฟ้าผ่า ก่อให้เกิดคลื่นวิทยุแผ่กระจายออกไปรอบโลก นอยส์บรรยากาศเกิดขึ้นอยู่ตลอดเวลา แม้จะไม่มีพายุฝนฟ้าคะนองก็ตาม
2. เกิดจากดวงอาทิตย์และดวงดาวนับล้าน ๆ ดวงในจักรวาล ดวงอาทิตย์เป็นวัตถุที่มีขนาดมหึมาและมีความร้อนสูงถึง 6,000 องศาเซลเซียสที่ผิวดวงอาทิตย์ ฉะนั้น ดวงอาทิตย์จะแผ่พลังงานออกมามีสเปกตรัมความถี่กว้างมาก พลังงานนี้ปรากฏออกเป็นนอยส์คงที่ อย่างไรก็ตามที่ผิวดวงอาทิตย์ยังมีความแปรปรวนอื่น ๆ อีก เช่น จุดบนดวงอาทิตย์ (sun spot) การลุกโชติช่วง (solar flare) ซึ่งก่อให้เกิดนอยส์เพิ่มขึ้นอีก นอกจากนี้ดวงอาทิตย์บางดวงที่ไกลออกไปจากระบบสุริยจักรวาลก็มีคุณสมบัติเหมือนดวงอาทิตย์ คือ มีความร้อนสูงและสามารถกำเนิดนอยส์มายังโลกได้
3. นอยส์ที่เกิดขึ้นจากสิ่งประดิษฐ์ที่มนุษย์สร้างขึ้น (man-made noise) ได้แก่ นอยส์จากมอเตอร์ไฟฟ้า เช่น พัดลม ที่เป่าผม เครื่องดูดฝุ่น นอกจากนี้ก็ยังมีนอยส์ จากระบบจุดระเบิดของรถยนต์ การรั่วของสายไฟแรงสูง หลอดไฟฟลูออเรสเซนต์ ฯลฯ
4. นอยส์ภายในตัวอุปกรณ์ในเครื่องรับ (internal noise) แยกเป็น 2 ประเภท คือนอยส์อุณหภูมิ (thermal noise) และช็อตนอยส์(shot noise) นอยส์อุณหภูมิเกิดจากการเคลื่อนที่ของอิเล็กตรอนในตัวอุปกรณ์ บางครั้งเรียกว่า จอห์นสันนอยส์ (Johnson noise) ส่วนช็อตนอยส์เกิดขึ้นในอุปกรณ์แอกทีฟ (active device) ทุกชนิด เนื่องจากการรวมตัวของอิเล็กตรอนกับโฮล (hole) เช่น ในทรานซิสเตอร์ ซึ่งไม่ขึ้นอยู่กับอุณหภูมิสัญญาณรบกวน(Noise)

นอยส์เหล่ามีสาเหตุหลักๆมาจาก

1. เสียงที่อยู่รอบตัวผู้ส่งสารในขณะที่เราส่งสาร
2. เสียงที่เกิดจากขึ้นระหว่างการส่งสาร
3. เสียงที่อยู่รอบตัวผู้รับสารในขณะรับสาร
4. สภาพภูมิอากาศอาจจะฝนตก พายุเข้า อากาศร้อนจัด อากาศหนาวจัด
5. สภาพภูมิศาสตร์อาจจะเป็นที่ราบ ภูเขา
6. ระยะทางในการส่งสาร
7. เสียงระหว่างการเข้ารหัสสัญญาณ

2.7.2 ลักษณะของ Noise

noise เป็นสิ่งไม่ต้องการของพลังงานไฟฟ้า และแม่เหล็กไฟฟ้า ซึ่งจะลดคุณภาพของสัญญาณและข้อมูล noise ปรากฏในระบบดิจิตอล และอนาล็อก และสามารถมีผลกับไฟล์และการสื่อสารของไฟล์ทั้งหมด รวมถึงข้อความ โปรแกรม ภาพ และเสียงเป็นต้น

ในวงจรสายแข็ง เช่น โทรศัพท์อินเตอร์เน็ตแบบใช้สาย external noise เป็นผลจากเครื่องใช้ในตำแหน่งใกล้เคียง, จากทรานส์ฟอร์มเมอร์, จากบรรยากาศ โดยปกติ noise ชนิดนี้มีผลเพียงเล็กน้อยหรือไม่มี อย่างไรก็ตามในระยะพายุฟ้าคะนอง หรือในตำแหน่ง ที่มีอุปกรณ์ไฟฟ้าจำนวนมากกำลังใช้งาน external noise มีผลกับการสื่อสาร ในการติดต่อทางอินเตอร์เน็ต external noise จะทำให้อัตราการส่งข้อมูลช้าลง เพราะระบบต้องปรับความเร็วให้ตรงกับเงื่อนไขบนสาย การสนทนาทางโทรศัพท์ noiseจะเป็นเสียงรบกวน noise เป็นปัญหา

สำคัญในระบบไร้สายมากกว่าระบบมีสาย โดยทั่วไป จุดเริ่มต้นของ noise จากภายนอก ที่แปลงตามสัดส่วนความถี่ และสัดส่วนโดยตรงกับความยาวคลื่น ที่ความถี่ต่ำ เช่น 300 kHz noise จากบรรยากาศและไฟฟ้า จะมากกว่าที่ความถี่สูง เช่น 300 MHz ส่วน noise ที่เกิดภายในตัวรับไร้สาย เรียกว่า internal noise จะขึ้นต่อความถี่น้อย วิศวกรมีความเกี่ยวข้องกับ internal noise ที่ความถี่สูงมากกว่าที่ความถี่ต่ำ เพราะถ้า external noise ลดลง internal noise จะมีความสำคัญมากขึ้น วิศวกรการสื่อสาร มีการพัฒนาในทางที่ดีกว่าในส่วนที่เกี่ยวข้องกับ noise วิธีการดั้งเดิมในการลดสัญญาณ bandwidth เพื่อการขยายระยะมากขึ้น การลดสเปคตรัม ของสัญญาณที่ใช้งาน จะลด noise ที่ผ่านไปยังวงจรลับ อย่างไรก็ตาม การลด bandwidth จำกัดความเร็วสูงสุดของข้อมูลในการส่ง การพัฒนาแบบแผนในทางผลกระทบของ noise เรียกว่า digital signal processing

2.7.3 ความผิดเพี้ยนของข้อมูล

ข้อมูลผิดเพี้ยน(error) หมายถึง ข้อมูลที่ผู้รับได้รับไม่เหมือนกับที่ผู้ส่งส่งให้ โดยปกติแล้วในระหว่างการรับ-ส่งข้อมูล หรือระหว่างการถ่ายทอดข้อมูลนั้น ข้อมูลมักจะถูกทำให้ผิดเพี้ยนไปจากเดิมเนื่องจากการรบกวนจากสิ่งต่างๆ ภายนอกระบบเครือข่ายซึ่งสามารถหลีกเลี่ยงได้แต่ไม่สามารถแก้ไขได้ และอีกส่วนหนึ่งเกิดจากปัญหาภายในระบบเองซึ่งสามารถหลีกเลี่ยงและแก้ไขได้

สาเหตุที่ทำให้ข้อมูลผิดเพี้ยน

1. สัญญาณอิมพัลส์ เป็นการรบกวนที่เกิดจากกระแสไฟฟ้าแรงดันสูงที่อยู่ภายนอกระบบเครือข่าย เช่นกระแสไฟฟ้าที่เกิดจากฟ้าผ่าสายสื่อสารที่เป็นประเภททองแดงที่อยู่ใกล้จะทำปฏิกิริยากับกระแสไฟฟ้าที่รุนแรงนั้นทำให้กระแสไฟภายในสายสื่อสายเปลี่ยนแปลงไปจากเดิม

2. สัญญาณกัสเสี้ยนเป็นสัญญาณรบกวนที่เกิดขึ้นจากการเปลี่ยนแปลงคุณสมบัติของลวดทองแดงเนื่องจากความร้อนที่เพิ่มขึ้นในระหว่างการใช้งาน จึงเรียกอีกอย่างว่าสัญญาณความร้อน สัญญาณประเภทนี้จะเกิดขึ้นรุนแรงตามระดับอุณหภูมิที่สูงขึ้น ผู้รับอาจแปลความหมายของสัญญาณไม่ได้หรืออาจจะแปลไม่รู้เรื่อง

3. สัญญาณอ่อนกำลัง การส่งสัญญาณออกไปทางสื่อไม่ว่าจะทำมาจากอะไรก็ตาม สัญญาณนั้นจะค่อยๆอ่อนกำลังลงเองตามระยะทางที่เพิ่มขึ้น วิธีการแก้ไขปัญหานี้สามารถทำได้โดยใช้อุปกรณ์ที่เรียกว่า "แอมปลิไฟเออร์" ใช้ในระบบสัญญาณแบบอนาลอก หรือใช้"รีพีทเตอร์" ในระบบสัญญาณแบบดิจดตอล อุปกรณ์นี้จะช่วยเพิ่มกำลังสัญญาณให้มีความแรงตามปกติ

4. ครอสทอล์ก การวางสายสื่อสาร(โดยปกติจะหมายถึงลวดทองแดง) หลายเส้นไว้ด้วยกัน จะทำให้สัญญาณจากสายสื่อสารต่างๆ เกิดการรบกวนซึ่งกันและกัน โดย ปกติสายแต่ละชนิดจะมีฉนวนหุ้มอยู่ซึ่งจะช่วยป้องกันสัญญาณรบกวนจากภายนอกและ ป้องกันไม่ให้สัญญาณภายในภายกระจายออกไปภายนอกด้วยเช่นกัน

5. การผิดเพี้ยนของสัญญาณเนื่องจากดีเลย์ ด้วยคุณสมบัติทางธรรมชาติของคลื่นสัญญาณที่ส่งออกผ่านทางสายสื่อสาร สัญญาณจะใช้ความถี่คลื่นที่ต่างกันแม้จะถูกส่งออกจากผู้ส่งพร้อมกันก็ตาม แต่จะเดินทางมาถึงผู้รับไม่พร้อมกัน เรียกเหตุการณ์เช่นนี้ว่า การผิดเพี้ยนของสัญญาณเนื่องจากดีเลย์

6. ปัญหาของสายสื่อสาร มักเกิดขึ้นอยู่เสมอในการสื่อสารข้อมูล คือ ปัญหาที่สายสื่อสารเอง โดยสายอาจจะชำรุดหรือขาดออกจากกัน ในกรณีนี้การสื่อสารจะหยุดชะงักลง ไม่สามารถใช้งานได้จนกว่าสายจะได้รับการซ่อมแซม

2.8 Audio file format

Audio file format เป็นฟอร์แมตที่ใช้ในการบรรจุ audio data บนระบบคอมพิวเตอร์ ซึ่งแบ่งเป็นหลายลักษณะดังนี้

1. Uncompressed audio format เช่น WAV, AIFF and AU
2. Compressed audio format แบ่งเป็น

- Lossless audio formats เช่น FLAC, Monkey's Audio (filename extension APE), WavPack (filename extension WV), Shorten, Tom's lossless Audio Kompressor (TAK), TTA, ATRAC Advanced Lossless, Apple Lossless and lossless Windows Media Audio (WMA).
- lossy compression เช่น MP3, Vorbis, Musepack, ATRAC, lossy Windows Media Audio (WMA) and AAC.

3. Free and open file formats
4. Open file formats
5. Proprietary formats

Uncompressed audio format

ตัวอย่างของ Uncompressed audio format ที่สำคัญคือ PCM (pulse code modulation) ที่ใช้เก็บฟอร์แมตของ .wav บน window, aiff บน macos .wav เป็นฟอร์แมตที่เหมาะสมในการจัดเก็บองค์ประกอบของ sampling rates หรือ bit rates Lossless compression จะมีข้อดีที่สามารถจัดเก็บไฟล์ได้อย่างมีประสิทธิภาพมากกว่าการเก็บไฟล์ในฟอร์แมตของ .wav อย่างไรก็ตามเพื่อที่จะเข้ารหัสไฟล์ฟอร์แมตโดยเปรียบเทียบระหว่าง Lossless compression กับ Uncompressed จะพบว่า แบบ Lossless compression ก็ยังมีความสำคัญมากกว่า .wav format .wav format จะมีพื้นฐานมาจาก ไฟล์ฟอร์แมตของ RIFF ซึ่งคล้ายคลึงกับ IFF File Format อีกทอดหนึ่ง

Lossless audio formats

Lossless audio formats ที่ใช้ทั่วไปเช่น FLAC, WavPack, Monkey's Audio, ALAC/Apple Lossless จะมีอัตราส่วนของการบีบอัดประมาณ 2:1

AUDIO FILE FORMATS

ในเวลาที่สัญญาณเสียง audio ถูกบันทึกผ่านเข้าไปในคอมพิวเตอร์ ข้อมูลของเสียงจะถูกเก็บไว้ในรูปแบบของ format ที่แตกต่างกันออกไป ที่น่าสนใจไปกว่านั้นก็คือ ข้อมูลเหล่านี้สามารถถูกเปลี่ยนจาก format เดิมไปเป็น format อื่นได้อีกด้วย โดย format เหล่านี้มีมากมายให้เลือกใช้ได้ แต่โดยส่วนใหญ่แล้วมักจะเก็บไว้ใน format ประเภทดังต่อไปนี้

AIFF (Audio Interchange File Format) โดยส่วนมากแล้ว จะถูกใช้โดย MIDI/Audio โปรแกรมบันทึกเสียงของ Mac โดย AIFF สามารถเป็นได้ทั้ง stereo และ mono พร้อมทั้งยังรับรอง resolutions จาก 8-bit/22kHz (lo-fi) ถึง 24-bit/96kHz(hi-fi) หรือสูงกว่า

MP3 เป็น format ประเภทที่ให้เสียงคุณภาพเสียงเยี่ยมในขนาด file ที่เล็กมาก แค่เพียง 1MB ต่อ1 นาทีของดนตรีเท่านั้น MP3 ใช้วิธีบีบข้อมูลให้เล็กลงโดยตัดข้อมูลเสียงที่หูคนเราได้ยินได้ยากออกไป จนเหลือเป็น

เพียงแค่ audio clip ที่เล็กว่าการเก็บข้อมูลเสียงที่ถูกเก็บด้วย format อื่นอย่างเช่น WAV หรือ AIFFข้อเสียของมันก็คือ คุณภาพเสียง MP3 จะไม่ดีเท่า Full band-width อย่างเช่าน hi-fi audio

QuickTime QuickTime ไม่ใช่ file format มันเป็นเพียง media player/record program ที่ถูกสร้างขึ้นโดย Apple แต่ ถึงอย่างไรก็ตาม โปรแกรมดนตรีบางชนิดสามารถ load และ save audio ให้เป็น QuickTime files ได้โดยที่ audio files เหล่านั้นจะมี .mov, .aif หรือ .WAV เพราะโปรแกรมดนตรีส่วนใหญ่สามารถเล่น QuickTime audio ได้ไม่ว่าจะมี file extension อะไรต่อท้าย

RealAudio Files ของ RealAudio นั้นลงท้ายด้วย .ra หรือ .rm โดย format นี้เป็นส่วนประกอบของ RealSystem G2 ที่ใช้สำหรับ multimedia ที่รวมถึง players, encoders, และ severs สำหรับการใช้สร้าง audio, video และ animation โปรแกรมดนตรีส่วนมากจะไม่ใช้ format นี้สำหรับการบันทึกมีแค่บางโปรแกรมเท่านั้นที่จะทำให้สามารถเก็บ audio ในรูปแบบของ RealAudio format

REX เป็น format ของ Propellerhead Recycle ซึ่งเป็น program ที่สามารถแบ่ง sampled loops ให้เป็นส่วนประกอบย่อย หรือแบ่งออกเป็นส่วนๆย่อยได้อย่างอัตโนมัติ (kick,snare,hi-hats หรืออื่นๆ)โดยส่วนที่แบ่งหรือแยกออกนี้สามารถ load ไปที่ sampler หรือ sequencer และใช้ triggered โดย MIDI files (อย่างเช่น" Recycle") และสามารถเพิ่มหรือลด speed ได้โดยที่ไม่ทำให้ pitch เปลี่ยน REX files จะลงท้ายด้วย .rx2(recycle 2.0 หรือสูงกว่า)และ .rcy หรือ .rex (สำหรับversionก่อนๆ)

Sound Designer II (SD II) เป็น stereo editing sofewareที่ผลิตโดย Digidesign ที่โด่งดัง และยังถูกใช้ใน Digi's multitrack recording program จนถึงทุกวันนี้เช่นเดียวกับ WAV หรือ AIFF Sound Designer II สามารถใช้กับ bit depth and sample rates ในระดับต่างกันได้ นอกจากนี้ pro audio programs ยังสามารถเปลี่ยน SDI files ไปเป็น WAV และ AIFF ได้

WAV ถูกสร้างขึ้นจากการรวมตัวกันของ Microsoft กับ IBM WAV format สามารถใช้ได้กับ bit depths และ sample rate ในระดับต่างกัน ในขณะที่ AIFF เป็นที่นิยมในหมู่ผู้ใช้ PC ด้วย ในเร็วๆนี้ Acidized WAV files ได้รับความนิยมเพิ่มขึ้นอีก นี่คือชนิดของ WAV files ที่รวมข้อมูลของ pitch กับ tempo เข้าไว้ด้วยกัน Acidized WAV สามารถถูกอ่านได้โดย Sonic Foundry Acid และโปรแกรมอื่นๆที่สามารถให้ samples ที่จัด pitch and tempo ได้โดยอัตโนมัติ

AUDIO DRIVER FORMATS ถึงแม้ audio driver จะเป็นเพียงจุดเล็กๆ แต่มันก็ช่วยให้โปรแกรมดนตรีเห็นและสามารถหาต้นตอของ audio ที่ผ่านเข้าและออกจาก hard ware audio interface ที่ต่อกับคอมพิวเตอร์ คุณสามารถที่จะติดตั้ง driver มากกว่าหนึ่งตัวเข้าไปในคอมพิวเตอร์ โดยในบางครั้ง driver บางยี่ห้ออาจจะ conflict หรือไม่ยอมทำงานกับส่วนประของ software บางชนิด และอาจจะทำให้มีปัญหากับตัวคอมพิวเตอร์หรือไม่สามารถทำงานกับ audio interface นั้นได้

ASIO (Mac,PC) Audio Steaming Input and Output (ASIO)(MAC,PC) คือ muti-channel audio driver ที่เป็นที่นิยมที่สุด ASIO drivers สามารถทำให้ muti-channel ของ inputs กับ outputs ทำงานพร้อมกันได้โดยส่วนมากแล้ว driver ตัวนี้จะมีความยืดหยุ่นน้อยกับ software synthesizers

Core Audio (MAC) เป็นการต่อของ Applications และ I/O devices มีความยืดหยุ่นต่ำแค่ 1 ms และมี multichannel support บริษัท Apple หวังว่า Core Audio จะเป็น standard audio driver สำหรับ MAC audiosoftware ทุกรุ่น

DAE (MAC,PC) ไม่เหมือนกับ audio drivers ชนิดอื่นๆ DAE ใช้ได้กับ Digidesign audio interface เท่านั้น แต่เป็นทั้ง multi-channel driver และ audio engine application โดย DAE สามารถใช้ควบคู่ไปกับ pro Tool,Logic และ Digital Performer และ DAEยังใช้ได้กับ RTAS และ/หรือ TDM plug-ins

Direct I/O (MAC,PC) เหมือนกับ DAE เช่นกัน Direct I/O ใช้ติดต่อกับ Digidsign interface เท่านั้น แต่เป็นแค่ multi-channel driver จึงไม่สามรถใช้ RTAS หรือ TDM effects ร่วมกับ Direct I/O

GSIF (PC) Driver ตัวนี้ถูกออกแบบให้มีความยืดหยุ่นในเรื่องของ latency ได้ดี ในเวลาใช้ samples จาก hard disk ผ่าน GigiStudio software sampler ของ TASCAM

MAS (MAC) เป็น driver โดยเฉพาะจาก Mark of the Unicorn โดย MAS เป็น Hard disk recording engine ที่ใช้กับ audio จาก Digital perfor ให้ resolution ได้ถึง 24-bit/96kHz และใช้ mutiple input และ output ได้พร้อมๆกัน

Wave PC เทียบเท่าได้กับ Sound Manager ของคอมพิวเตอร์ระบบ Mac เราสามารถใช้ Wave กับ audio interface ได้หลายแบบ (อย่างพวก soundcard ของ Sondblaster ชนิดต่างๆ)ในการบันทึกหรือเล่น mono/stereo audio แต่อาการเหลื่อมมักเป็นปัญหาที่พบได้บ่อยใน Wave และ SoundManager Drives

WDM (PC) เป็น multi-channel driver เช่นเดียวกับ ASIO ที่ใช้ได้ดีกับนักดนตรีที่นิยมใช้คอมพิวเตอร์ในระบบ PC สามารถใช้กับ Dxi software โดยไม่มีปัญหาเรื่องความเหลื่อม และใช้เล่น live กับ DirectX audio effects ไม่เฉพาะเวลา playback เท่านั้น ยังสามารถ monitor และบันทึก effects ไปได้พร้อมๆกันด้วย

WMA เป็นรูปแบบไฟล์เสียงที่พัฒนาโดยไมโครซอฟต์ มีความคล้าย MP3 แต่จะมีขนาดเล็กกว่า ณ. คุณภาพเสียงเดียวกันกับ MP3 อย่างไรก็ตามไฟล์ WMA นี้ยังไม่เป็นที่รู้จักแพร่หลายเท่า MP3

2.9 ไฟล์ Wave

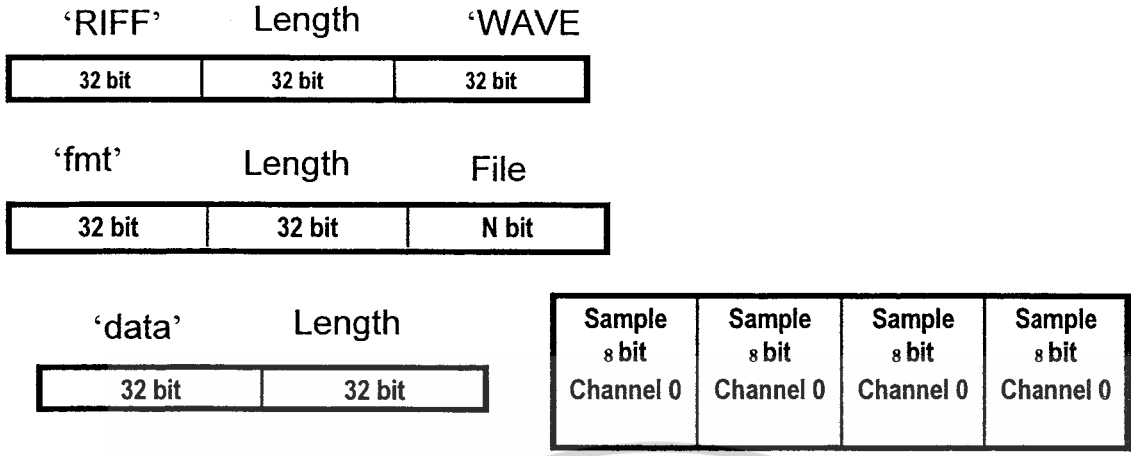
ไฟล์เสียง wave เป็นไฟล์เสียงที่เราคุ้นเคยกันมากที่สุด ไฟล์ประเภทนี้มีนามสกุล .wav จัดเป็นไฟล์เสียงมารฐานที่ใช้กับ Windows คุณสมบัติที่สำคัญคือครอบคลุมความถี่เสียงได้ทั้งหมด ทำให้คุณภาพเสียงดีมาก และยังให้เสียงในรูปแบบสเตอริโอได้อีกด้วย ข้อเสียคือไฟล์ .wav มีขนาดใหญ่ทำให้สิ้นเปลืองพื้นที่ในการเก็บข้อมูลมาก

2.9.1 รูปแบบของไฟล์ .wav (wav file format)

รูปแบบของไฟล์.WAV มีทั้งหมด 4 ประเภทคือ

ไฟล์ 8-bit mono

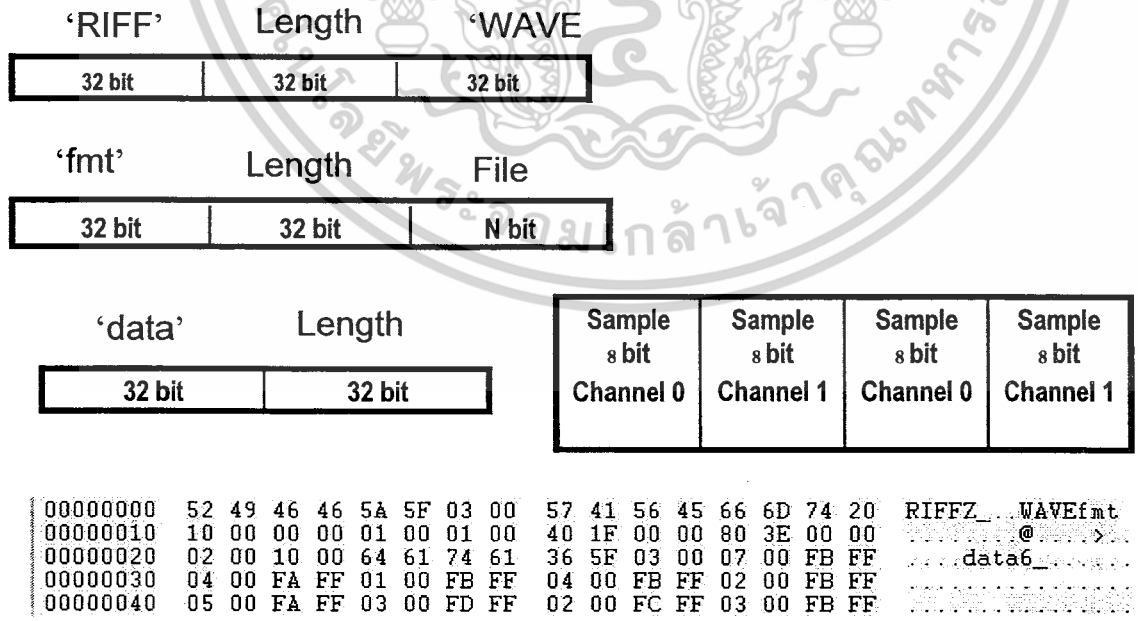
ตัวอย่างแบบ 8-bit mono นี้ ไบท์ (bytes) ทั้งหมดจะถูกเก็บโดยเรียงตามลำดับโดยช่องที่ 0 (channel 0) ถูกนำมาใช้สำหรับตัวอย่างแบบ mono ตามตัวอย่างรูปที่ 2.27



รูปที่ 2.27 แสดงโครงสร้างของไฟล์ 8-bit mono และตัวอย่างไฟล์

ไฟล์ 8-bit stereo

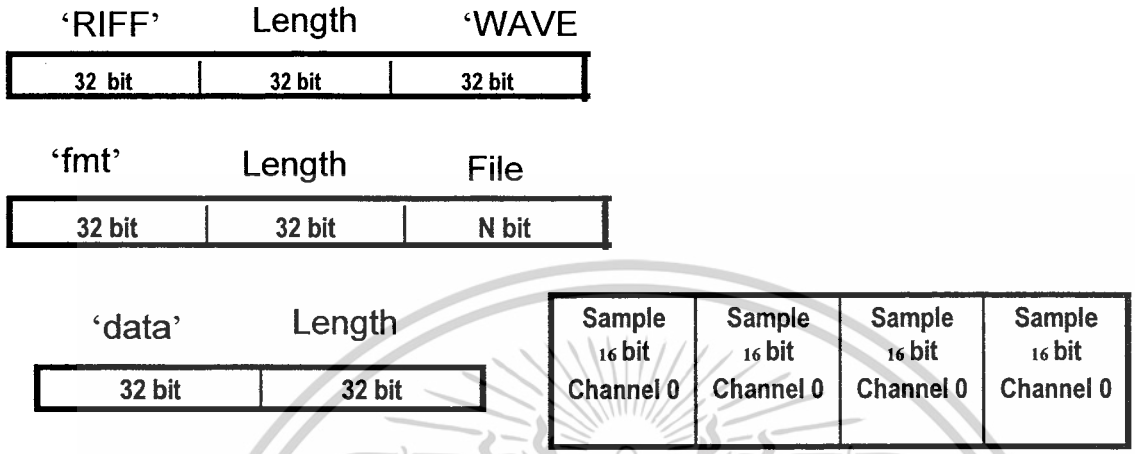
ตัวอย่างแบบ 8-bit stereo นี้ ช่องสัญญาณที่ 0 (channel 0) คือช่องสัญญาณทางด้านซ้าย และช่องสัญญาณที่ 1 (channel 1) คือช่องสัญญาณทางด้านขวา ข้อมูลจะถูกเก็บไว้โดยการเก็บข้อมูลของช่องสัญญาณ 0 และ ช่องสัญญาณ 1สลับกันไปเรื่อยๆ ตามตัวอย่างรูปที่ 2.28



รูปที่ 2.28 แสดงโครงสร้างของไฟล์ 8-bit stereo และตัวอย่างไฟล์

ไฟล์ 16-bit mono

การแสดงตัวอย่างแบบ 16-bit mono ในหน่วยความจำนี้จะใช้ไบท์ในการเก็บข้อมูลจำนวน 2 ไบท์ (16 bit) ในแต่ละตัวอย่าง (sample) เรียงลำดับของไบท์ข้อมูลเหมือนกันกับแบบ 8-bit mono ตามตัวอย่างรูปที่ 2.29

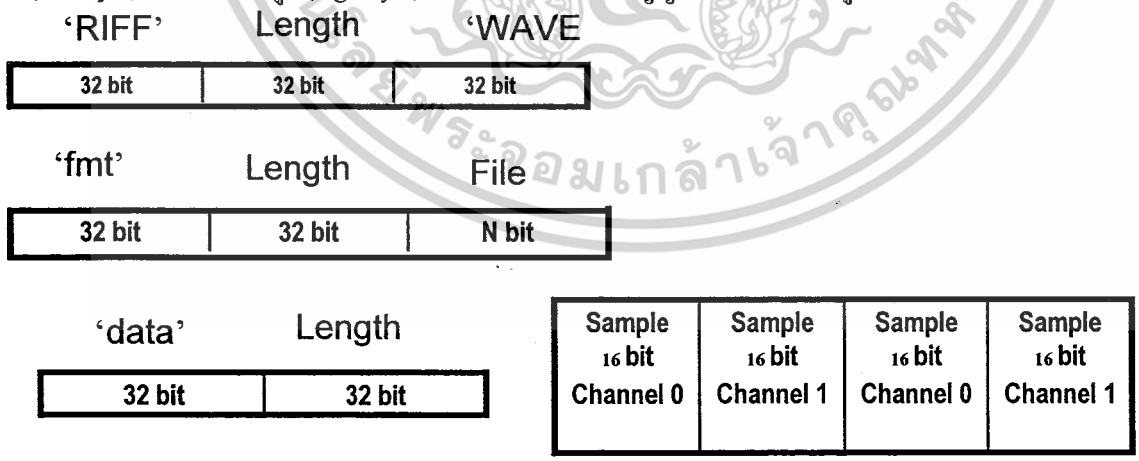



```
00000000  52 49 46 46 F0 52 02 00  57 41 56 45 66 6D 74 20  RIFF.R..WAVEfmt
00000010  10 00 00 00 01 00 01 00  11 2B 00 00 11 2B 00 00  .........+...+..
00000020  01 00 08 00 64 61 74 61  CB 52 02 00 80 80 80 80  ....data.R......
00000030  80 80 80 80 81 80 80 80  80 80 80 80 80 80 80 80  ................
00000040  80 80 80 80 80 80 80 80  80 80 81 80 81 80 81 80  ................
00000050  81 80 81 80 81 80 81 80  81 80 81 80 81 80 81 80  ................
```

รูปที่ 2.29 แสดงโครงสร้างของไฟล์ 16-bit mono และตัวอย่างไฟล์

ไฟล์ 16-bit stereo

ต้องการจำนวนหน่วยความจำจำนวนมากสำหรับแต่ละตัวอย่าง (sample) ซึ่งต้องการถึง 4 ไบท์ 2 ไบท์สำหรับช่องสัญญาณทางซ้าย และอีก 2 ไบท์สำหรับช่องสัญญาณทางขวา) ลำดับของการเรียงข้อมูล คือ 1 ไบท์ต่ำ(low byte) และ 1 ไบท์สูง (high byte) สำหรับแต่ละช่องสัญญาณตามตัวอย่างรูปที่ 2.30




```
00000000  52 49 46 46 BA A5 04 00  57 41 56 45 66 6D 74 20  RIFF....WAVEfmt
00000010  10 00 00 00 01 00 01 00  11 2B 00 00 22 56 00 00  .........+.."V..
00000020  02 00 10 00 64 61 74 61  96 A5 04 00 00 00 FE FF  ....data........
00000030  00 00 FE FF FF FF FD FF  FF FF FE FF 01 00 FD FF  ................
00000040  FF FF FE FF FF FF 00 00  00 00 FC FF 00 00 FE FF  ................
00000050  00 00 FC FF 00 00 FD FF  00 00 FD FF 00 00 FD FF  ................
```

รูปที่ 2.30 แสดงโครงสร้างของไฟล์ 16-bit stereo และตัวอย่างไฟล์

ในตัวอย่างแบบ 8-bit ค่าของแต่ละไบท์อยู่ระหว่าง 0 และ 255 ดังนั้นค่า 128 แสดงค่ากลางของรูปคลื่น แบบดิจิตอล(median of the digitized waveform) สำหรับตัวอย่างแบบ16-bit พิสัยของค่าอยู่ ระหว่าง -32768 ถึง 32767 ในกรณีนี้ค่า 0 เป็นค่ากลาง (median)

2.9.2 Wave File Format

เนื่องจากข้อมูลเสียงมีลักษณะที่ต่อเนื่อง แต่คอมพิวเตอร์นั้นจะทำงานกับข้อมูลที่ไม่ต่อเนื่อง เราจึงต้องมีการแปลงข้อมูลเสียงออกมาก่อนหรือที่เรียกว่า การแซมปลิ้ง (Sampling) ซึ่งก็เหมือนกับการดูว่า ณ เวลา t ระยะแอมปลิจูด A ที่อนุภาคขยับจากค่าปกติเป็นเท่าไหร่ ถ้าเราแซมปลิ้งด้วยความถี่ที่มากๆ เมื่อเอาข้อมูลมาต่อกัน สัญญาณก็น่าจะคล้ายกับของเดิม ความถี่ในการแซมปลิ้งนี้เราจะเรียกว่า Sampling Rate

เนื่องจากระบบคอมพิวเตอร์รู้จักแค่ค่า 0 และ 1 ดังนั้นถ้าเราใช้แค่บิตเดียวแทนก็คงได้เสียงหึ่งๆ ออกมาแค่นั้น ก็เลยจำต้องกำหนดว่าเราจะใช้ข้อมูลกี่บิตในการแทนข้อมูล 1 ชุด ซึ่งปกติก็จะใช้เป็นจำนวนที่มีค่าเป็นยกกำลังของสองเช่น 8, 16, 32, ... เป็นต้น และเนื่องจากเรามีสองหูซ้ายขวา ทำให้เราต้องบอกว่าจะสร้างให้หูข้างซ้ายเป็นเสียงยังไง และเสียงสำหรับหูข้างขวาเป็นยังไง ดังนั้นเราจึงสามารถสร้างเสียงเปิดสำหรับหูข้างซ้ายมาฟังพร้อมๆกับเสียงแตรวงในหูขวาก็ได้ ซึ่งเราเรียกระบบนี้ว่า สเตอริโอ แต่ถ้าจะสร้างออกมาให้เหมือนกันทั้งสองหู หรือ โมโน ก็ได้เช่นกัน โดยเรามักจะเรียกจำนวนหูนี้ว่า Channels

สำหรับไฟล์ Wav นั้นจริงๆแล้วมันคือก้อนข้อมูลอะไรบางอย่าง ที่ได้ใส่เฮดเดอร์เพื่อบอกว่าหน้าตาของก้อนข้อมูลนั้นเป็นยังไง ไฟล์ Wav นี้ใช้กันมากในวินโดวส์(window) สำหรับในแมค(MAC)จะใช้ AIFF ซึ่งหน้าตาคล้ายกันเพราะเป็นลูกพี่ลูกน้องสืบเชื้อสายมาจากไฟล์ประเภท RIFF เหมือนกันแทน โดยปกติก้อนข้อมูลนี้มักจะใช้การเข้ารหัสแบบ PCM (Pulse Code Modulation) เพื่อแสดงแอมปลิจูด A ของคลื่นที่เวลา t แต่ในความเป็นจริงเราสามารถใช้ Codec อื่นๆในการเก็บข้อมูลก็ได้เช่นกัน

การที่เราจะเล่นไฟล์เสียงได้ เราก็ต้องรู้ก่อนว่าไฟล์เสียงนั้นเก็บข้อมูลยังไง ซึ่งกุญแจสำคัญก็อยู่ที่เฮดเดอร์(Header)นี่แหละ เฮดเดอร์ที่ใช้ในไฟล์ Wav นั้นโดยทั่วไปประกอบด้วยข้อมูลขนาด 44 ไบต์พอดีไม่มีขาดไม่มีเกิน ข้อมูลจะถูกเก็บในรูปแบบ little endian โดยจะมีโครงสร้างดังนี้

Header 12 ไบต์แรกเรียกว่า "RIFF Chunk"

ไบต์ที่	ขนาด (ไบต์)	รายละเอียด
0-3	4	ข้อความ "RIFF"
4-7	4	ขนาดของข้อมูลที่จะตามมาด้านหลัง
8-11	4	ข้อความ "Wave"

ตารางที่ 2.3 แสดง RIFF Chunk

Header 24 ไบต์ถัดมาเรียกว่า "Format Chunk"

ไบต์ที่	ขนาด (ไบต์)	รายละเอียด
12-15	4	ข้อความ "fmt " (มีช่องว่างต่อท้าย)
16-19	4	ขนาดของ FORMAT Chunk ปกติแล้วถ้าใช้ PCM ก็จะมีค่าเป็น 16 เสมอ
20-21	2	ถ้าเป็น Uncompressed data จะมีค่าเป็น 1 แต่ ถ้าเป็นค่าอื่นแสดงว่าข้อมูลถูกบีบอัดอยู่
22-23	2	จำนวน Channel มีค่า 1 หรือ 2
24-27	4	กำหนด Sampling Rate ปกติแล้วถ้าเป็น CD Quality จะมีค่าเป็น 44100
28-31	4	Bytes Rate - คำนวณจาก Sampling Rate * จำนวน Channels * BitsPerSample / 8
32-33	2	Block Alignment - คำนวณจาก จำนวน Channels * BitsPerSample / 8
34-35	2	Bits per Sample - จำนวนบิตต่อแซมเปิล มีค่าเป็น 8, 16,24, ...

ตารางที่ 2.4 แสดง FORMAT Chunk

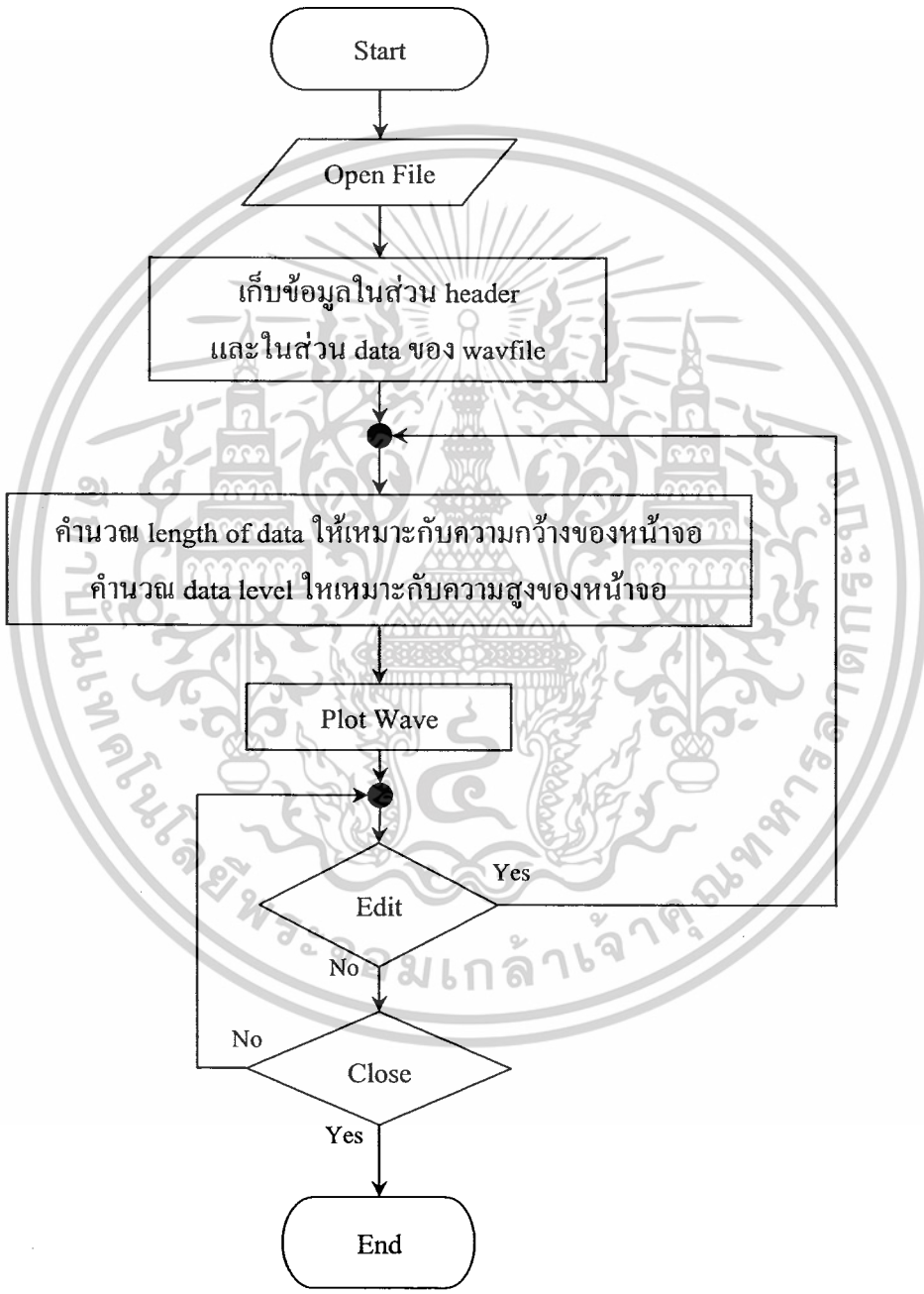
ส่วนสุดท้ายเรียกว่า "Data Chunk"

ไบต์ที่	ขนาด (ไบต์)	รายละเอียด
36-40	4	ข้อความ "data"
41-44	4	ขนาดของข้อมูลที่เหลือใน Data Chunk
45-จบ	n	ข้อมูลของไฟล์ Wav

ตารางที่ 2.5 แสดง Data Chunk

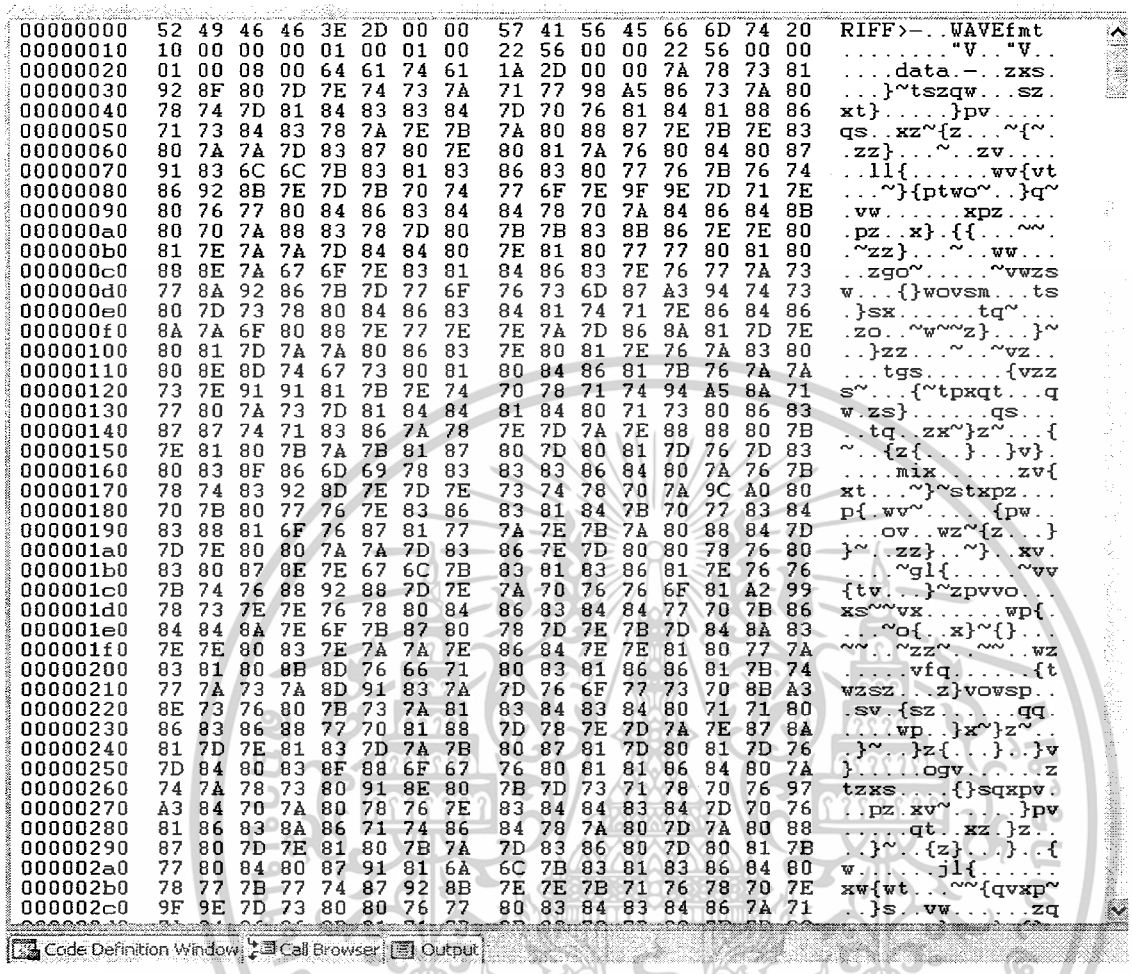
บทที่ 3
การออกแบบ และการสร้าง

3.1 โครงสร้างโดยรวมของโปรแกรม



รูปที่ 3.1 แผนภาพแสดงลำดับการทำงานโดยรวมของโปรแกรม

3.2 การคำนวณข้อมูลในส่วน header ของ wav file



รูปที่ 3.2 แสดงตัวอย่าง wav file format

ตัวอย่างการคำนวณ

- Channel Number

  จาก แถวที่ 00000010 10 00 00 00 **01 00** **01 00** 22 56 00 00 22 56 00 00

  0x0001 => 1 channel (Mono)

- Sampling Rate

  จาก แถวที่ 00000010 10 00 00 00 01 00 01 00 **22 56 00 00** 22 56 00 00

  0x00005622 = 22050 Hz (samples/sec

- Bytes Per Second

  จาก แถวที่ 00000010 10 00 00 00 01 00 01 00 22 56 00 00 **22 56 00 00**

  0x00005622 = 22050 bytes/sec

- Bytes Per Sample

จาก แถวที่ 00000020 **01 00** 08 00 64 61 74 61 1A 2D 00 00 7A 78 73 81

0x0001 = 1 byte/sample (8 bit Mono)

- Bits Per Sample

จาก แถวที่ 00000020 01 00 **08 00** 64 61 74 61 1A 2D 00 00 7A 78 73 81

0x0008 = 8 bits/sample

- Length Of Data

จาก แถวที่ 00000020 01 00 08 00 64 61 74 61 **1A 2D 00 00** 7A 78 73 81

0x00002D1A = 11546 Data

3.3 การ คำนวณสเกล

สเกลแอมปลิจูด

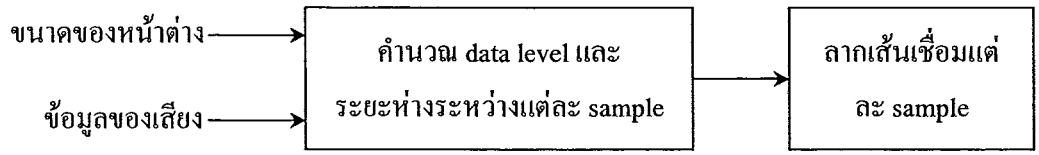
- **Mono** จะคิดเป็นเปอร์เซ็นต์ โดยคำนวณจาก ค่า bits/sample หรือ quantized level เช่น ถ้า wav file เป็นแบบ 8 bits/sample จะได้ quantized level ทั้งหมด 256 ระดับ โดยระดับ 0 จะกำหนดให้เป็น -100% และ ระดับสูงสุด จะกำหนดให้เป็น 100% โดยแบ่งสเกลละเอียดช่องละ 12.5% และให้มีความกว้างเต็มจอ ในแนวตั้ง
- **Stereo** จะทำการแบ่งครึ่งหน้าจอ โดยครึ่งบนจะเป็น channel ซ้าย และครึ่งล่างเป็น channel ขวา โดยกำหนดให้เป็น 100% โดยแบ่งสเกลละเอียดช่อง 25% ทั้งครึ่งบน และครึ่งล่าง

สเกลเวลา

จะคิดเป็นวินาที โดยคำนวณหาเวลาที่ใช้ในการเล่นเสียงทั้งหมด จากการนำ length of data มาหารด้วย sampling rate จากนั้นคำนวณหาความกว้างของหน้าให้เหมาะสมกับแต่ละวินาที

3.4 การ Plot Wave Form

ฟังก์ชันนี้จะทำงานตลอดเมื่อหน้าต่างของโปรแกรมแสดงอยู่บนหน้าจอ โดยฟังก์ชันนี้จะทำการคำนวณค่าขนาดของหน้าต่าง ณ ปัจจุบัน และรับค่าข้อมูลของเสียงในขณะนั้น เช่น sampling rate, channel, bits/sample และ data ณ ขณะนั้น มาทำการคำนวณ data level และ ระยะห่างระหว่างแต่ละ sample ในแนวแกนเวลา แล้วทำการลากเส้นเชื่อมต่อระหว่างแต่ละ sample ไปเรื่อยๆจนจบไฟล์

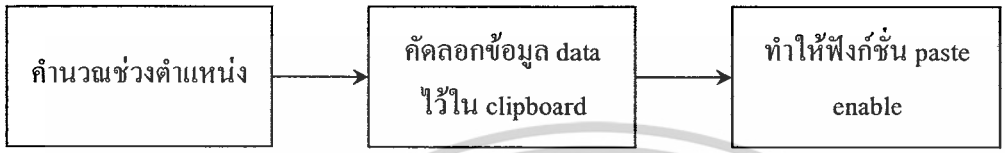


รูปที่ 3.3 แสดงลำดับการทำงานของการ plot wave form

3.5 ฟังก์ชั่นต่างๆของโปรแกรม

ฟังก์ชั่น Copy

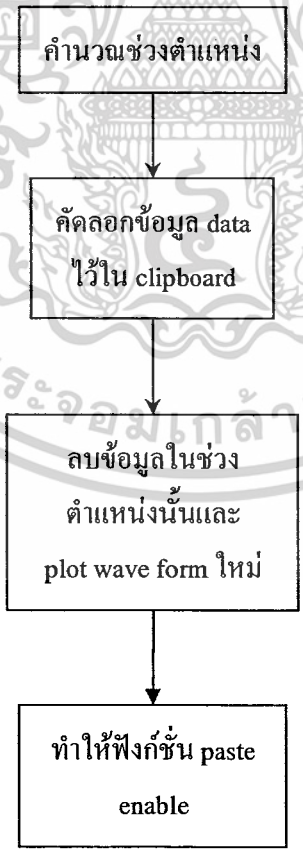
จะรับการส่งตำแหน่งเริ่มต้นและตำแหน่งสิ้นสุดที่ได้ทำการโฟกัสไว้จากการลากแถบโฟกัส ไปยังไฟล์ แล้วทำการคัดลอกข้อมูล data ในช่วงตำแหน่งนั้นเก็บไว้ใน clipboard แล้วทำให้ฟังก์ชั่น paste นั้นพร้อมที่จะทำงาน



รูปที่ 3.4 แสดงลำดับการทำงานของฟังก์ชั่น copy

ฟังก์ชั่น Cut

จะรับตำแหน่งเริ่มต้นและตำแหน่งสิ้นสุดที่ได้ทำการโฟกัสไว้จากลากแถบโฟกัสไปยังไฟล์ แล้วทำการคัดลอกข้อมูล data ในช่วงนั้นเก็บไว้ ใน clipboard แล้วลบข้อมูล data ในช่วงตำแหน่งนั้นพร้อมกับทำการคำนวณ และ plot wave form แล้วทำให้ฟังก์ชั่น paste นั้นพร้อมที่จะทำงาน

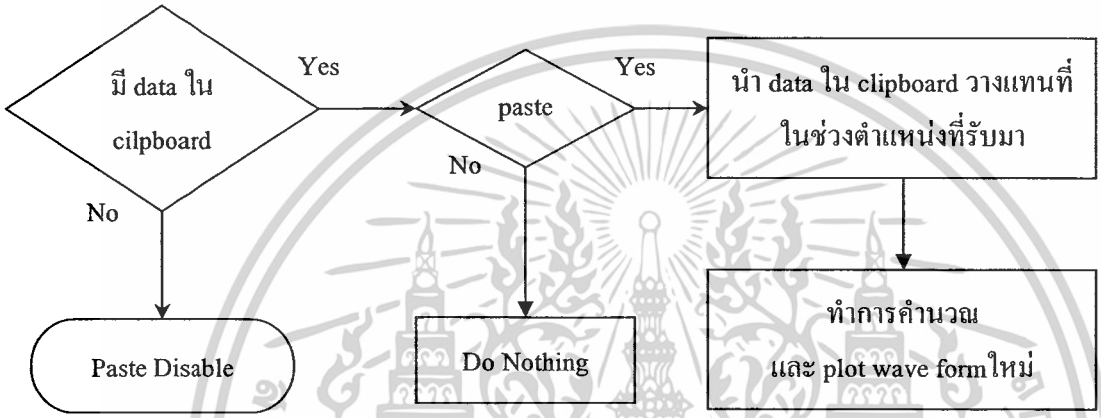


รูปที่ 3.5 แสดงลำดับการทำงานของฟังก์ชั่น cut

ฟังก์ชั่น Paste

จะทำการตรวจสถานะว่ามีข้อมูล data ส่วนที่คัดลอกอยู่ใน clipboard หรือเปล่า

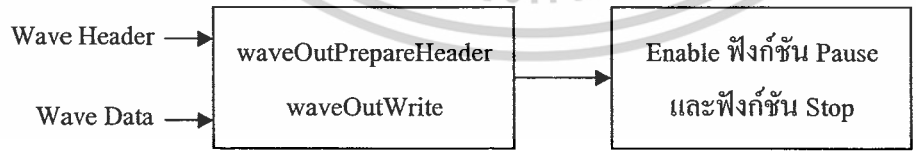
- ถ้ามี และมีการ paste ก็จะนำข้อมูล data ที่อยู่ใน clipboard วางแทนที่ในช่วงตำแหน่งที่รับมาจาก cursor แล้วส่ง data ที่ได้แก้ไขแล้วไปยังฟังก์ชัน plot wave form ให้มีการคำนวณ และ plot ใหม่
- ถ้าไม่มี ก็ยังไม่สามารถใช้ฟังก์ชั่นนี้ได้



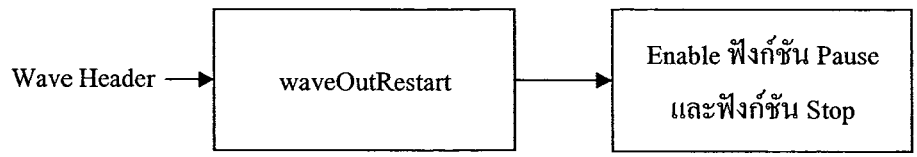
รูปที่ 3.6 แสดงลำดับการทำงานของฟังก์ชั่น paste

ฟังก์ชั่น Play

จะทำการคำนวณข้อมูลต่างๆในส่วนของ header และรับค่าข้อมูล data ของไฟล์ในปัจจุบัน และส่งข้อมูลเหล่านี้แปลงไปเป็นเสียงออกทางลำโพง โดยผ่านฟังก์ชั่น waveOutPrepareHeader และ waveOutWrite ใน code โปรแกรม แต่หากมีการ pause ไว้ ก็ใช้ฟังก์ชัน waveOutRestart เพื่อเล่นเสียงต่อ ณ ตำแหน่งที่ pause ไว้ แล้วทำให้ฟังก์ชัน Pause และ Stop พร้อมที่จะทำงาน



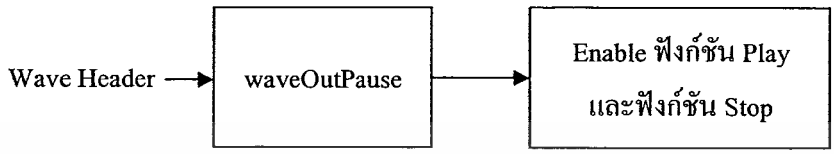
รูปที่ 3.7 แสดงลำดับการทำงานของฟังก์ชั่น Play



รูปที่ 3.8 แสดงลำดับการทำงานของฟังก์ชั่น play เมื่อมีการ pause

ฟังก์ชัน Pause

จะทำหยุดการเล่นเสียงไว้ชั่วคราว ณ ตำแหน่งที่กำลังเล่นอยู่โดยใช้ฟังก์ชัน waveOutPause โดยฟังก์ชันนี้จะรับค่าข้อมูลในส่วนของ header เข้าไป แล้วทำให้ฟังก์ชัน Play และ Stop พร้อมที่จะทำงาน



รูปที่ 3.9 แสดงลำดับการทำงานของฟังก์ชั่น pause

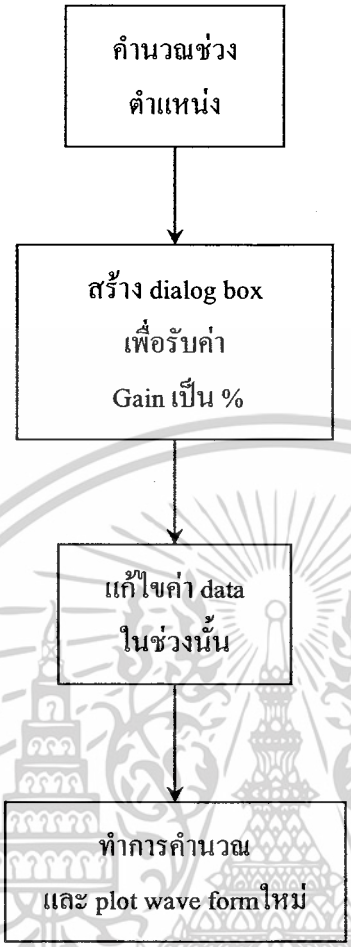
ฟังก์ชัน Stop

จะทำหยุดการเล่นเสียง และ reset กลับไปยังตำแหน่งเริ่มต้น โดยใช้ฟังก์ชัน waveOutReset และ waveOutUnprepareHeader โดยจะผ่านข้อมูลในส่วนของ header และ data ของ wave file เข้าไป แล้วทำให้ฟังก์ชัน Play พร้อมที่จะทำงาน แต่ disable ฟังก์ชัน Pause และ Stop

รูปที่ 3.10 แสดงลำดับการทำงานของฟังก์ชั่น stop

ฟังก์ชั่น Change Gain

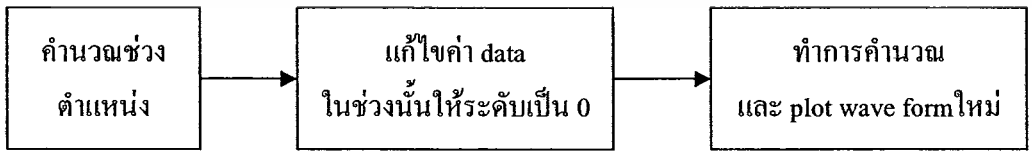
จะรับตำแหน่งเริ่มต้นและตำแหน่งสิ้นสุดที่ได้ทำการโฟกัสไว้จากการลากแถบโฟกัสไปยังไฟล์ แล้วสร้าง dialog box ขึ้นมาเพื่อให้ผู้ใช้สามารถกำหนดอัตราการขยายได้ในหน่วยของเปอร์เซ็นต์ตัวอย่างเช่นใส่ 200% ก็จะเพิ่มแอมปลิจูดเป็น 2 เท่าจากเดิม และถ้าใส่ 50% ก็ลดแอมปลิจูดลงครึ่งหนึ่ง เมื่อกำหนดอัตราขยายแล้ว ก็จะมีการแก้ไขค่าข้อมูล data ในส่วนนั้น หลังจากนั้นก็จะไปทำงานยังส่วนของการ plot wave form ให้มีการคำนวณ และ plot ใหม่



รูปที่ 3.11 แสดงลำดับการทำงานของฟังก์ชั่น change gain

ฟังก์ชัน Silence

จะรับตำแหน่งเริ่มต้นและตำแหน่งสิ้นสุดที่ได้ทำการ โฟกัสไว้จากการลากแถบโฟกัสไปยังไฟล์แล้ว data เสียงในช่วงนั้นเงียบไป หลังจากนั้นก็จะไปทำงานยังส่วนของการ plot wave form ให้มีการคำนวณและ plot ใหม่



รูปที่ 3.12 แสดงลำดับการทำงานของฟังก์ชั่น silence

ฟังก์ชัน Change Play Rate

ฟังก์ชันนี้จะรับค่า sampling rate จากผู้ใช้ผ่านทาง dialog box เพื่อทำให้เล่นเสียงเร็วขึ้นหรือช้าลง

บทที่ 4

ผลการทดลอง และวิเคราะห์ผลการทดลอง

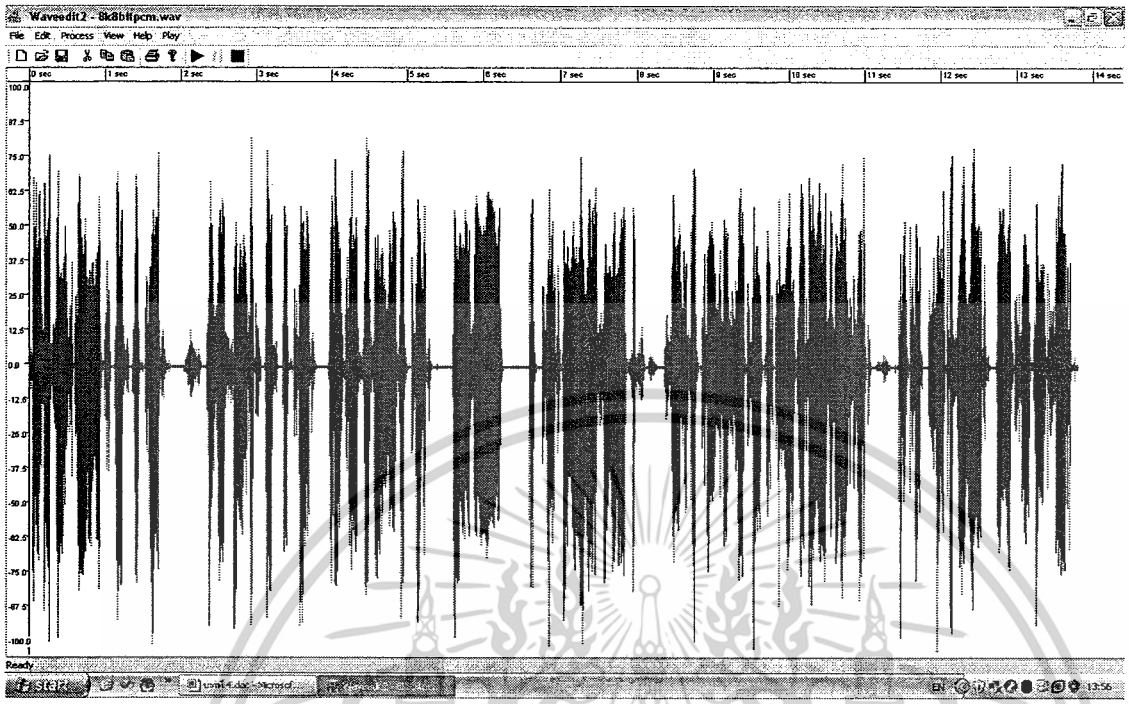
4.1 เปิดโปรแกรม Wave Reader

รูปที่ 4.1 แสดงหน้าต่างหลักของโปรแกรม

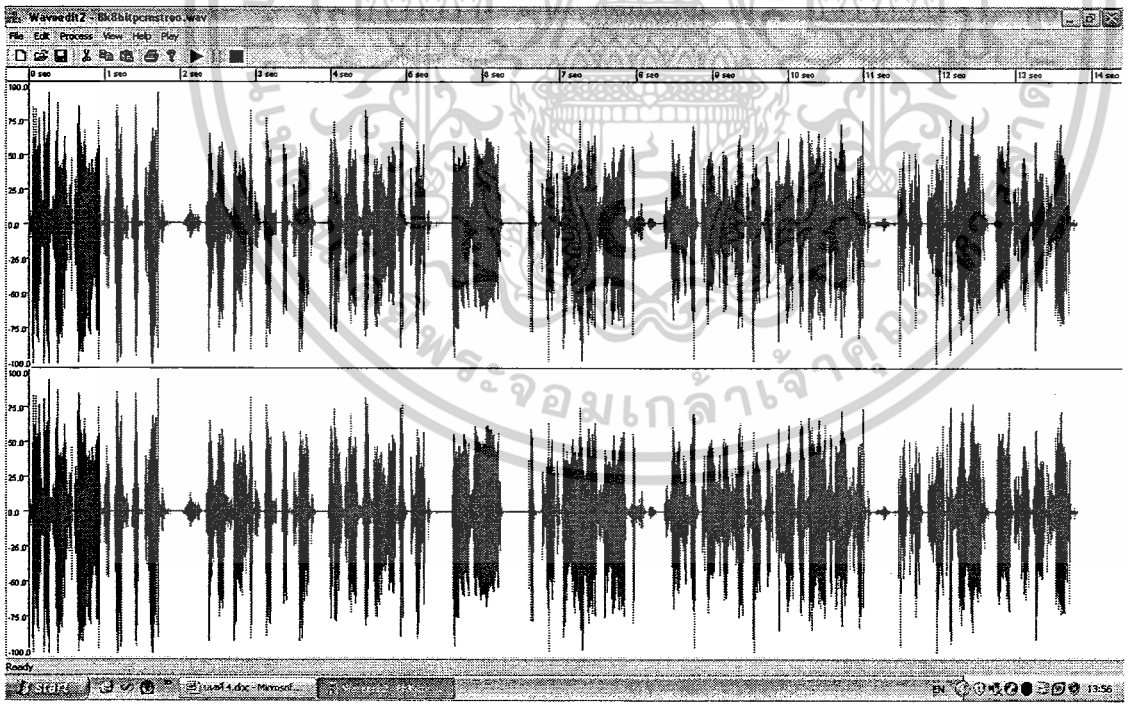
เมื่อกดปุ่ม open โปรแกรมจะสร้าง dialog box ขึ้นเพื่อเลือกไฟล์เสียงใน directory ที่ต้องการ

รูปที่ 4.2 แสดง dialog box ในการ open file

เมื่อเลือกไฟล์เสียงแล้วโปรแกรมจะทำการ plot wave form พร้อมทั้งเขียนสเกล ดังรูป



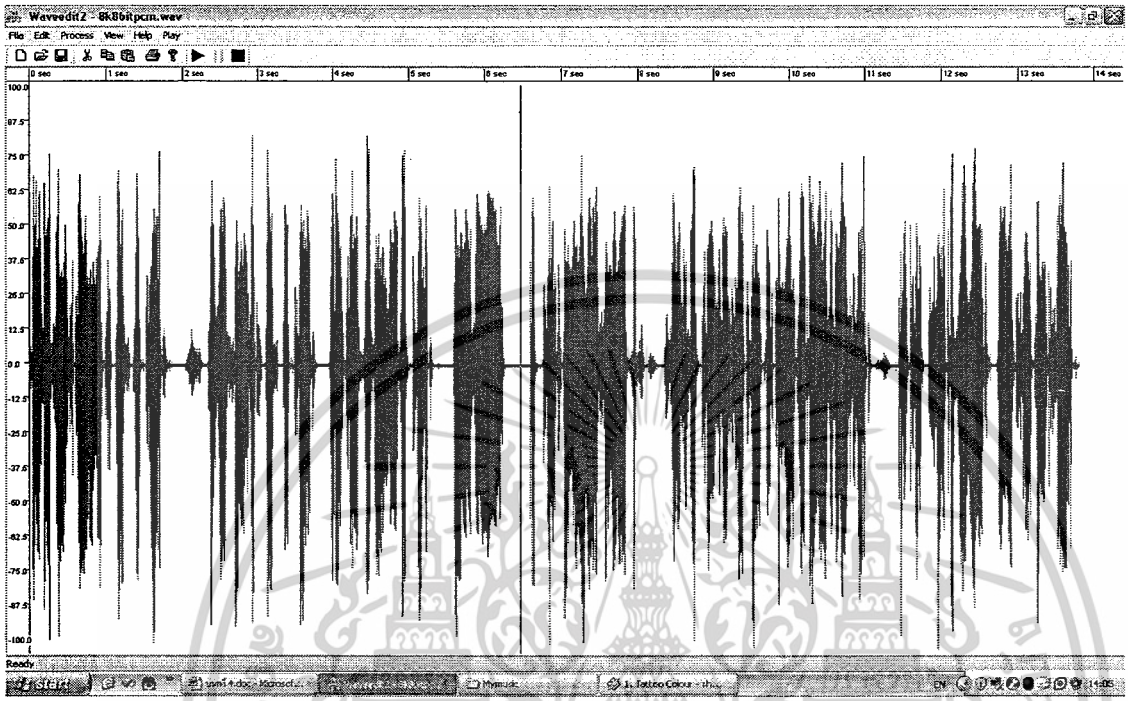
รูปที่ 4.3 แสดงการ plot wave form (Mono)



รูปที่ 4.4 แสดงการ plot wave form (Stereo)

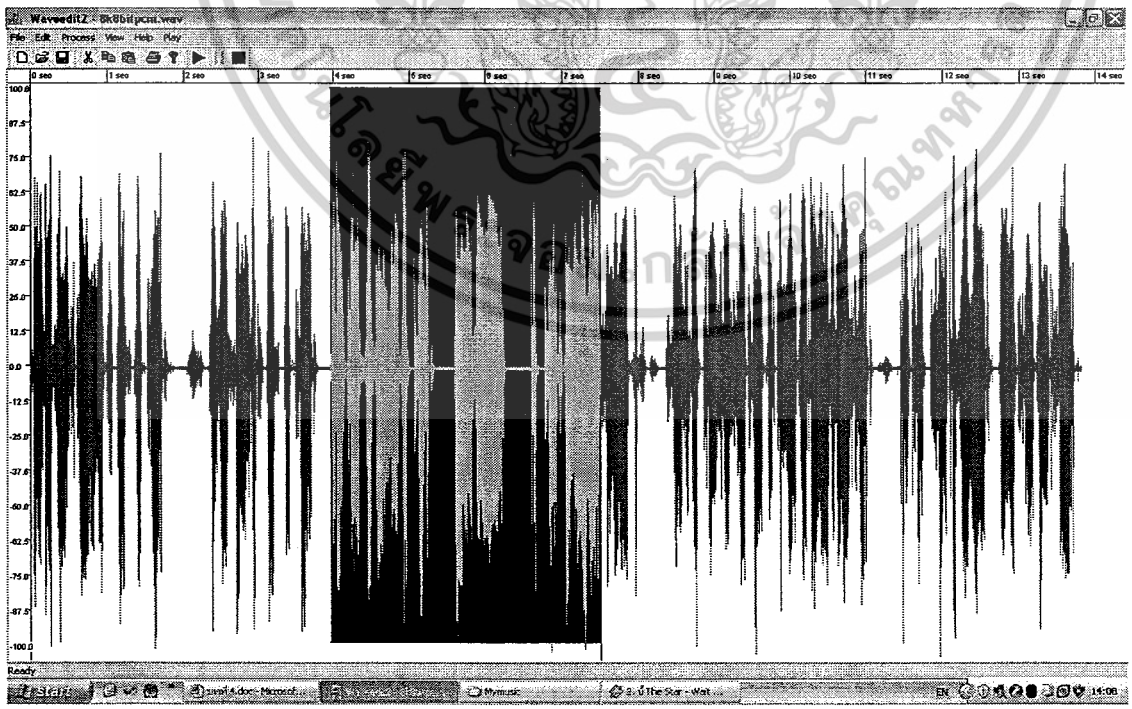
4.2 การใช้ cursor เลือกตำแหน่ง

เมื่อทำการคลิ๊กเม้าท์ซ้ายบนหน้าจอ จะมี cursor กระพริบ ณ ตำแหน่งนั้น และจะมีการเก็บค่าตำแหน่งนั้นไว้ในตัวแปร

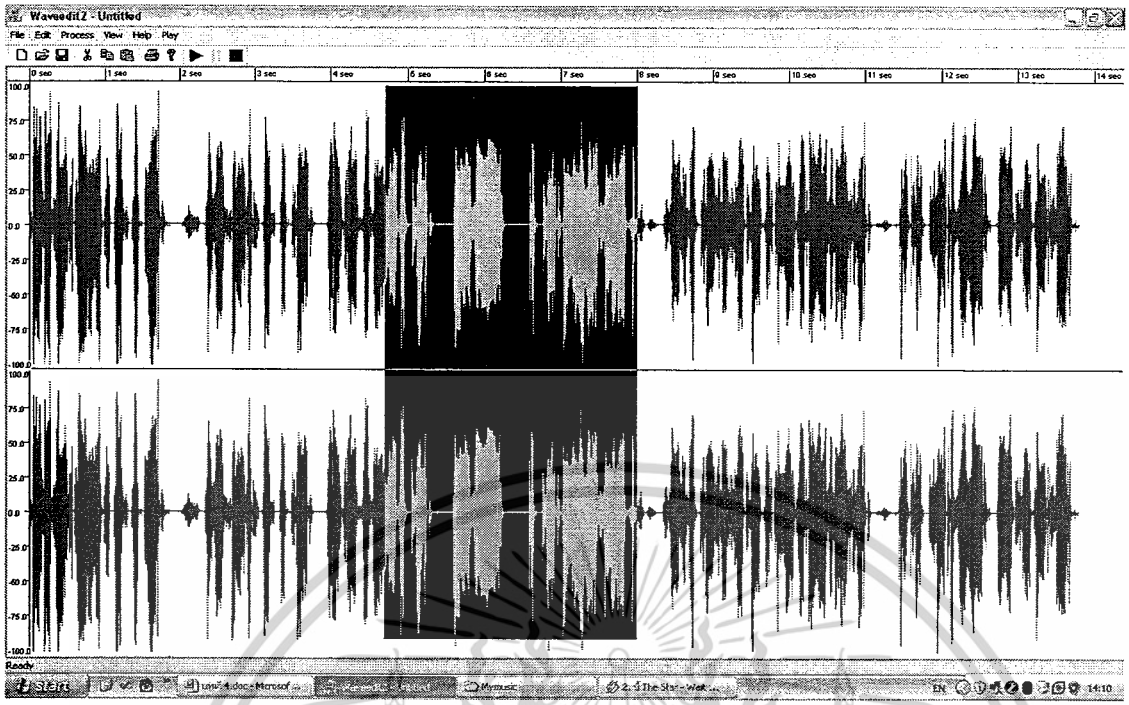


รูปที่ 4.5 แสดงการเลือกตำแหน่ง

เมื่อทำคลิ๊กเม้าท์ซ้ายค้างแล้วลากจะได้แถบดำ focus ช่วงตำแหน่งที่ต้องการ ดังรูป



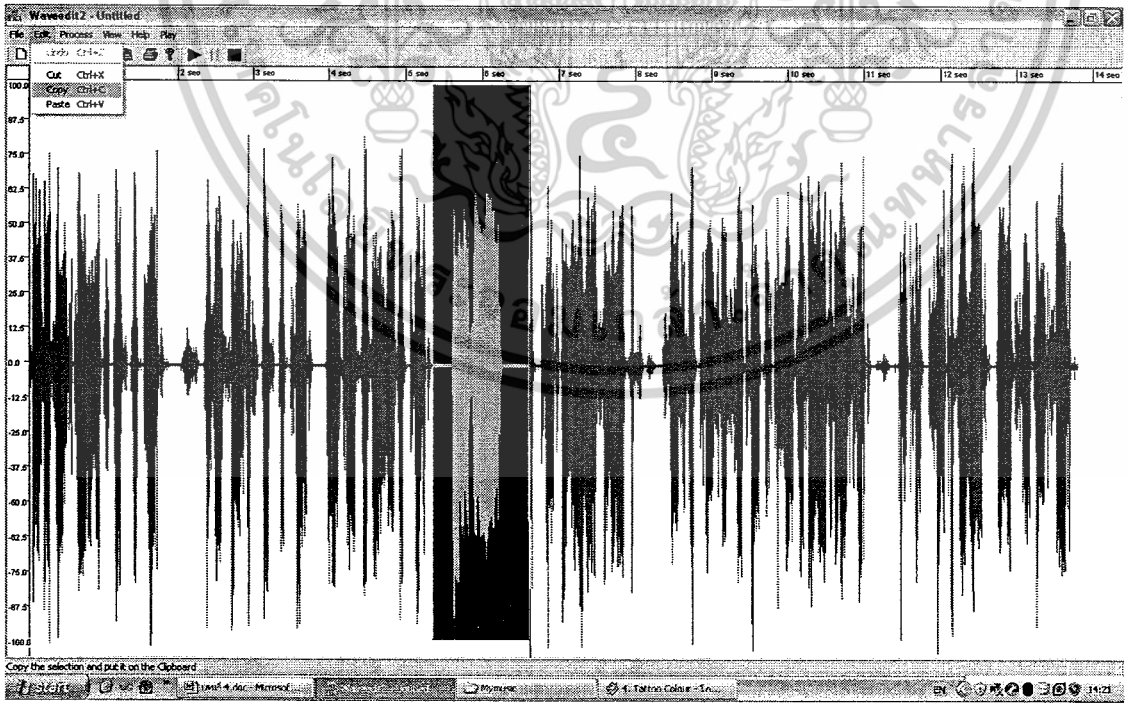
รูปที่ 4.6 แสดงแถบ focus (Mono)



รูปที่ 4.7 แสดงแถบ focus (Stereo)

4.3 การใช้ฟังก์ชัน copy

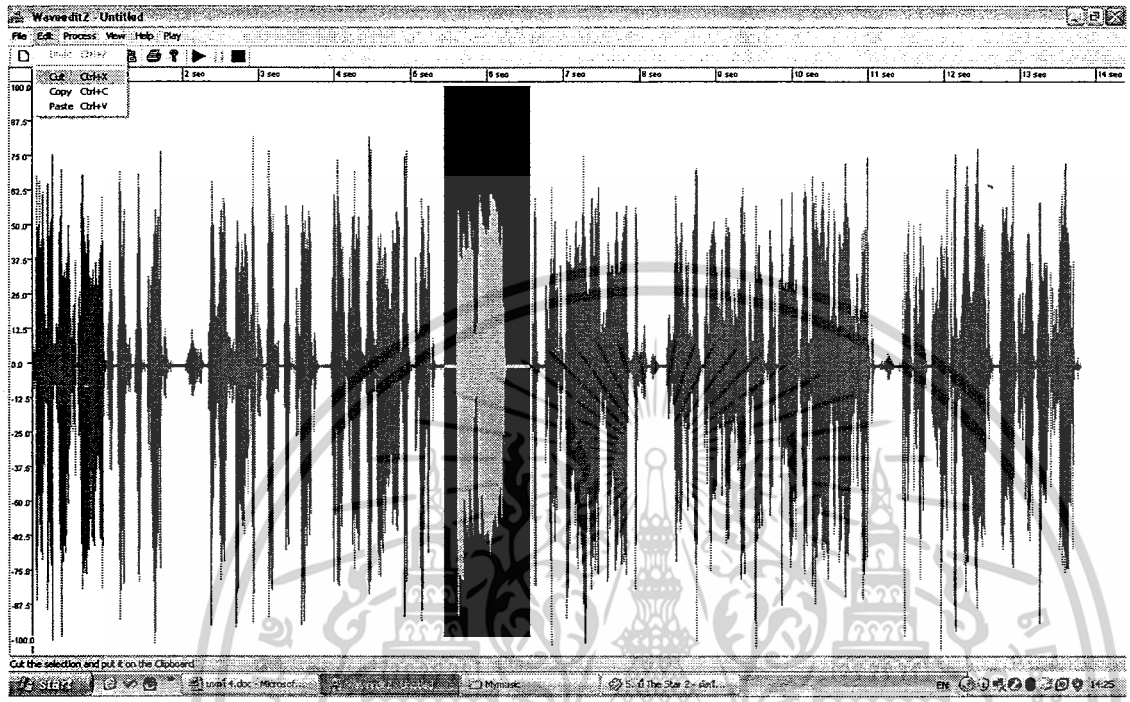
หลังได้จากได้ทำการ focus ช่วงเสียงที่ต้องการได้แล้ว ก็ทำการ copy โดยข้อมูลเสียงในช่วงนั้นก็จะถูกคัดลอกไว้ใน clipboard ในที่นี้จะแสดงเฉพาะไฟล์แบบ Mono



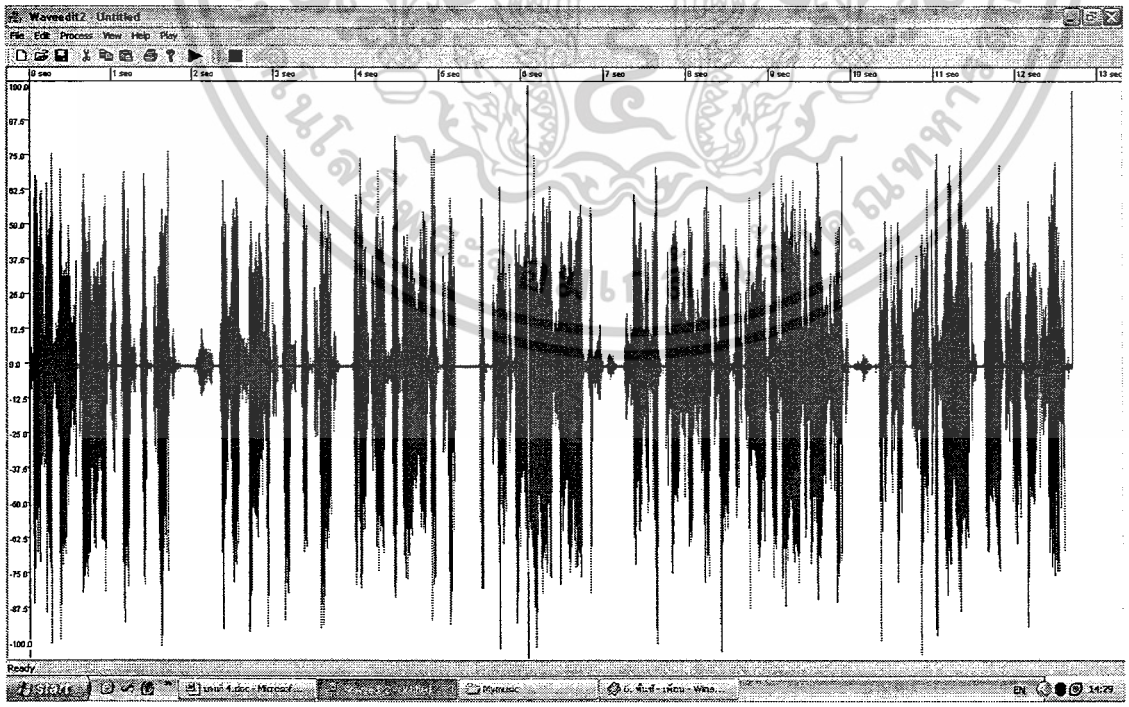
รูปที่ 4.8 แสดงการใช้ฟังก์ชัน copy

4.4 การใช้ฟังก์ชัน cut

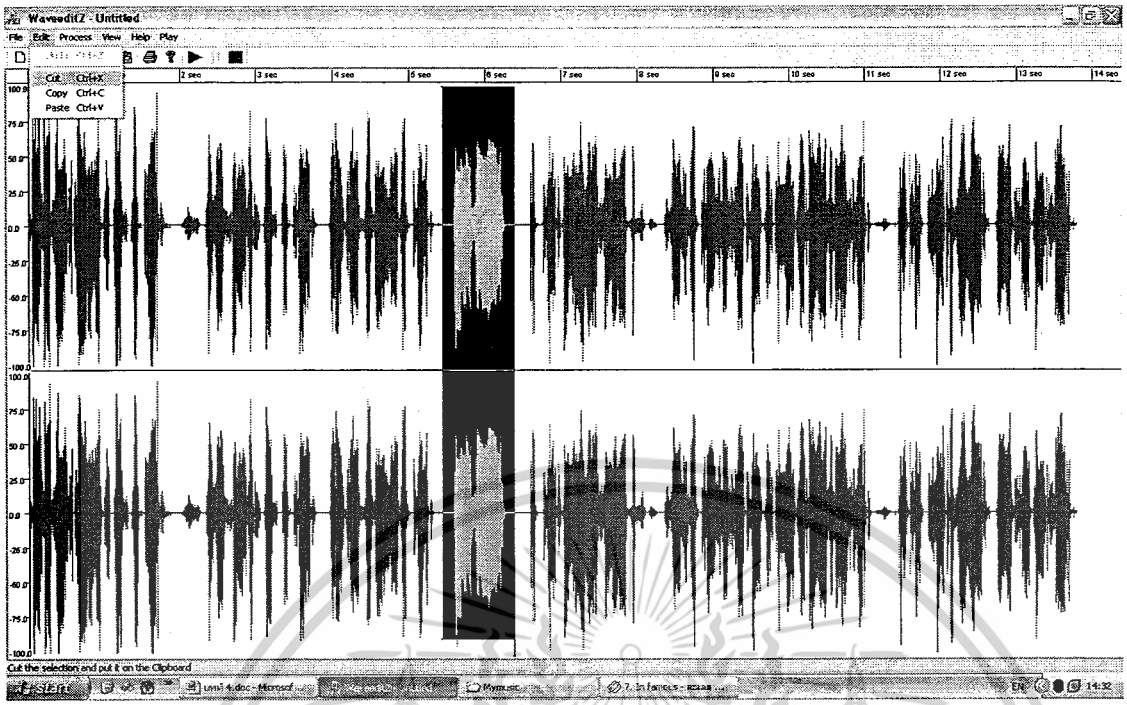
หลังได้จากได้ทำการ focus ช่วงเสียงที่ต้องการได้แล้ว ก็ทำการ cut โดยข้อมูลเสียงในช่วงนั้นก็จะถูกคัดลอกไว้ใน clipboard จากนั้นข้อมูลในช่วงนั้นก็ถูกลบไป



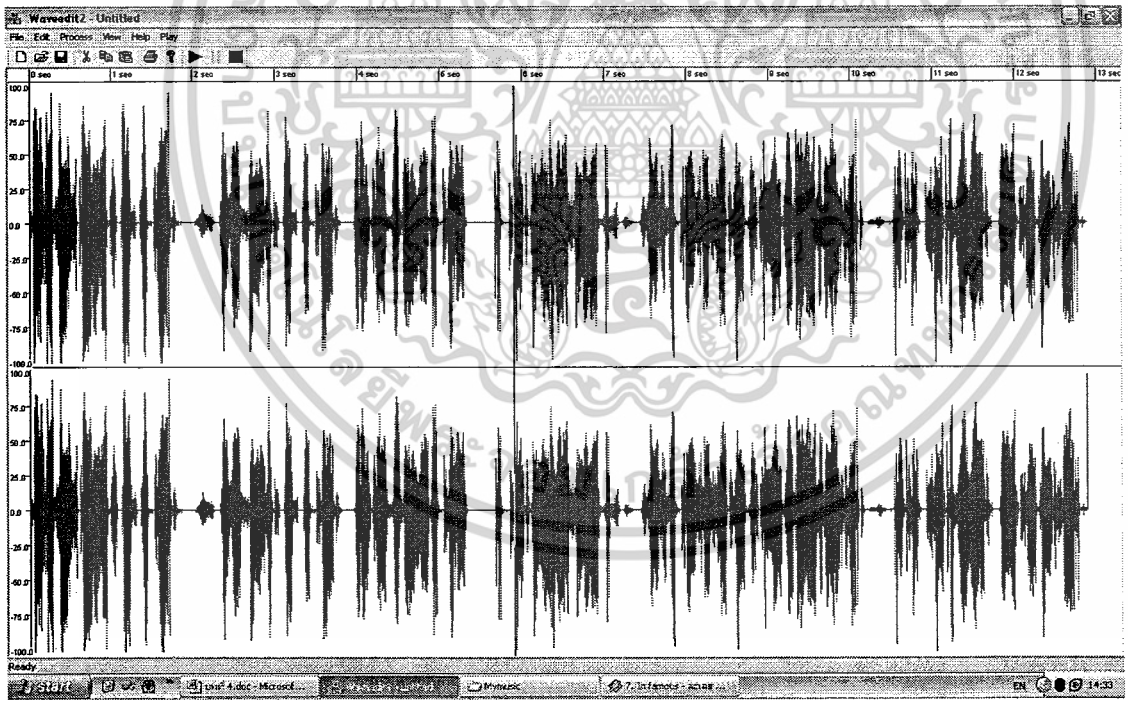
รูปที่ 4.9 แสดงการใช้ฟังก์ชัน cut (Mono)



รูปที่ 4.10 แสดงหลังจากใช้ฟังก์ชัน cut (Mono)



รูปที่ 4.11 แสดงการใช้ฟังก์ชัน cut (Stereo)

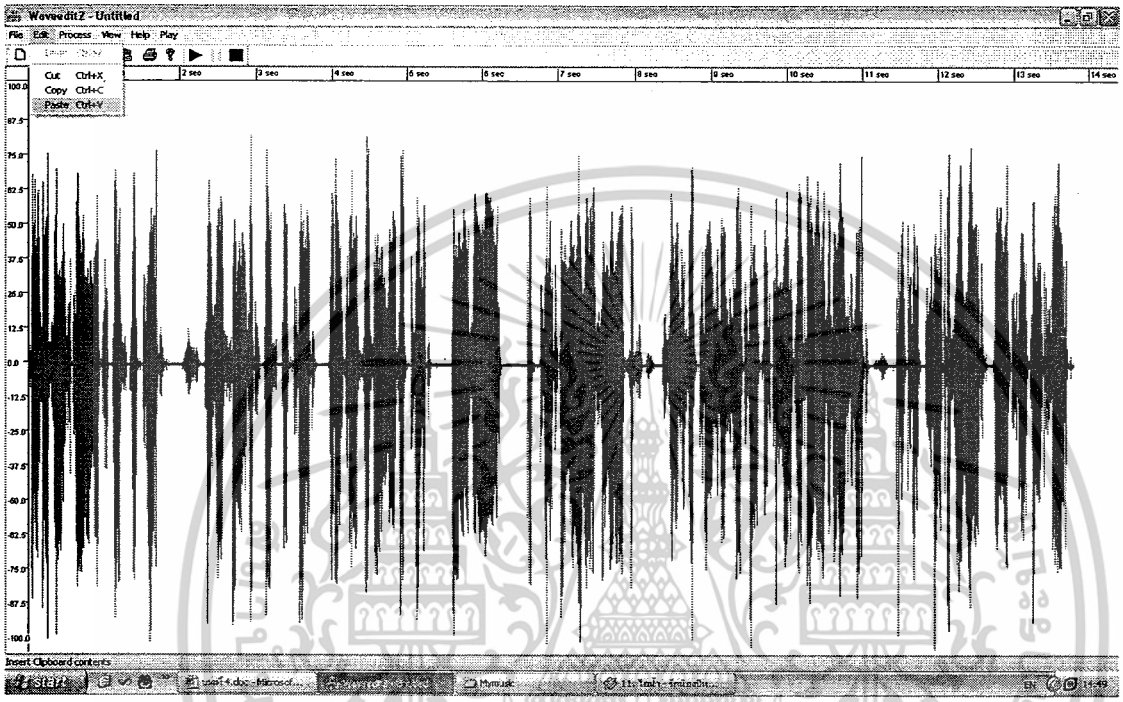


รูปที่ 4.12 แสดงหลังจากใช้ฟังก์ชัน cut (Stereo)

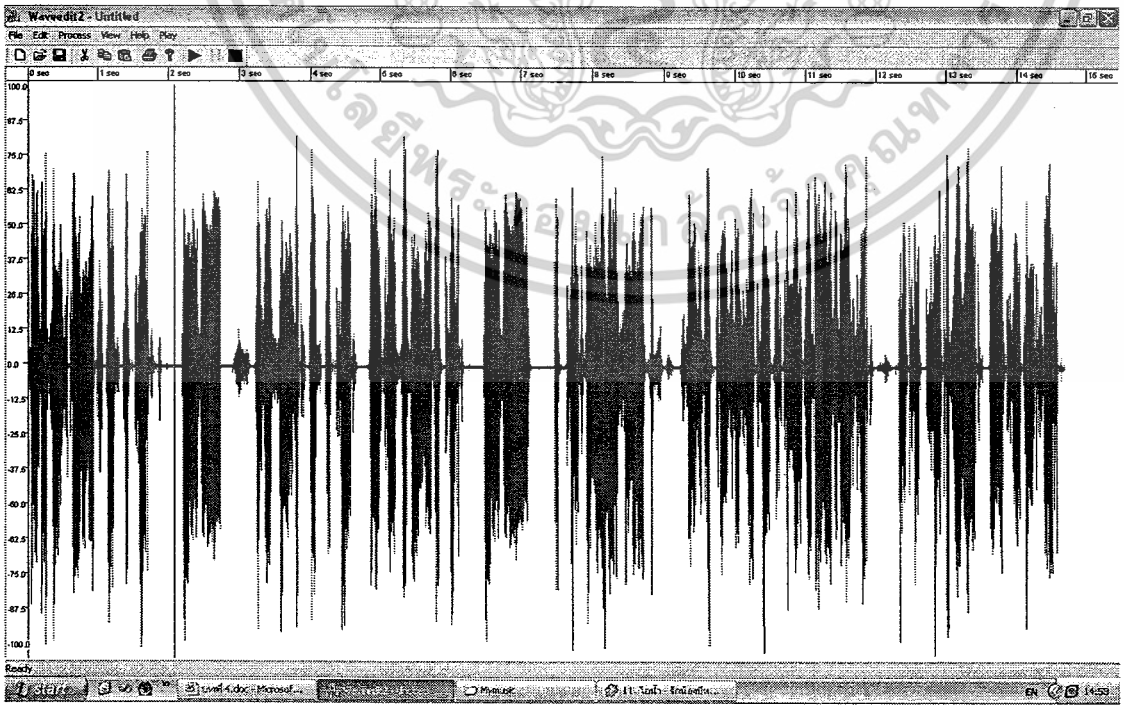
4.5 การใช้ฟังก์ชัน paste

หลังจากได้ทำการ copy หรือ cut ก็จะมีข้อมูลส่วนที่คัดลอกไว้อยู่ใน clipboard การ paste ก็แบ่งออกเป็น 2 กรณี คือ

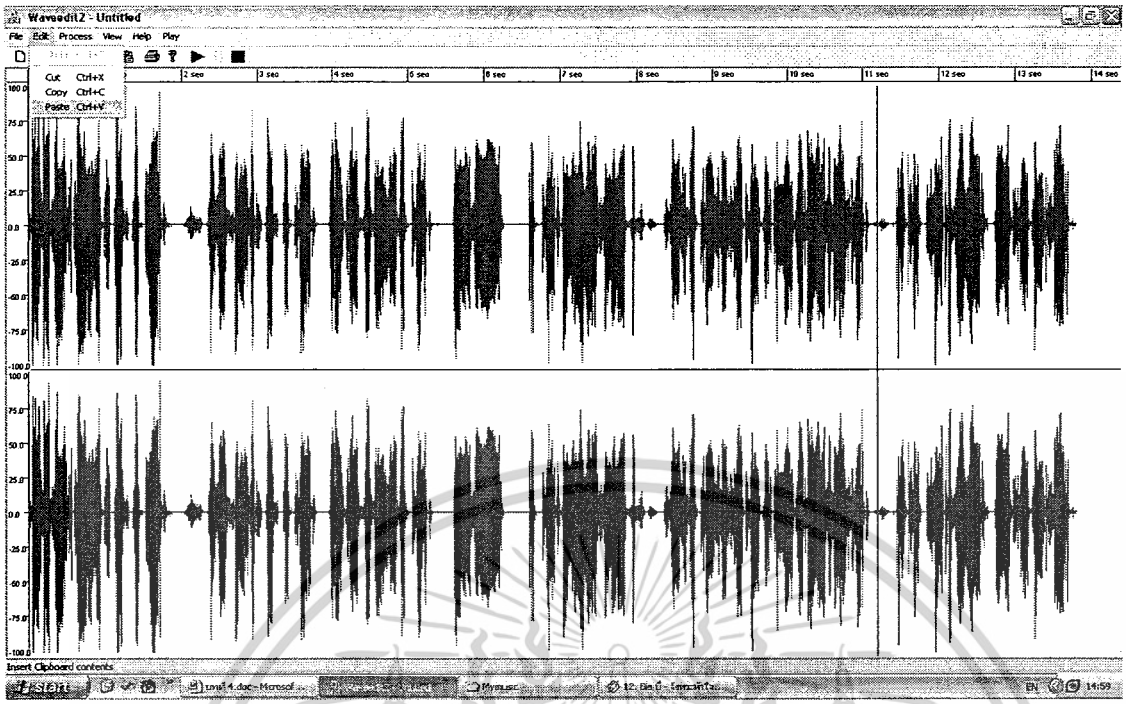
1.) เมื่อ cursor ระบุตำแหน่งเดียว เมื่อการทำการ paste ข้อมูลที่อยู่ใน clipboard ก็วางแทรกลงไปในตำแหน่งนั้น



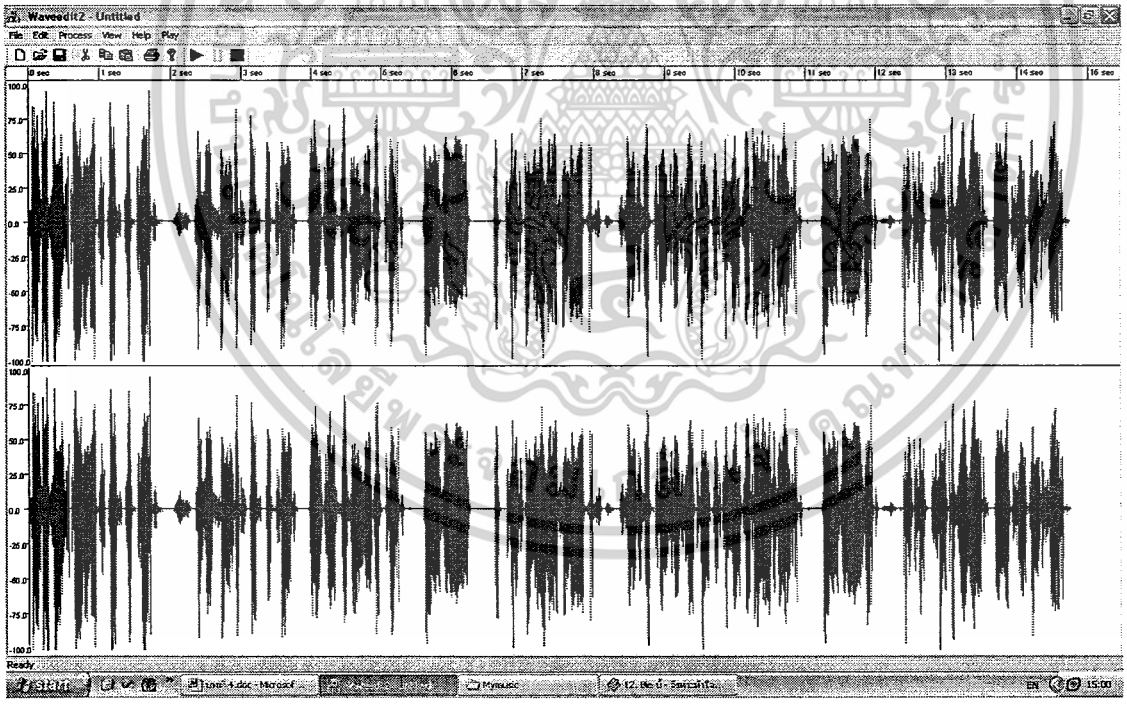
รูปที่ 4.13 แสดงการใช้ฟังก์ชัน paste กรณีแรก (Mono)



รูปที่ 4.14 แสดงหลังจากใช้ฟังก์ชัน paste กรณีแรก (Mono)

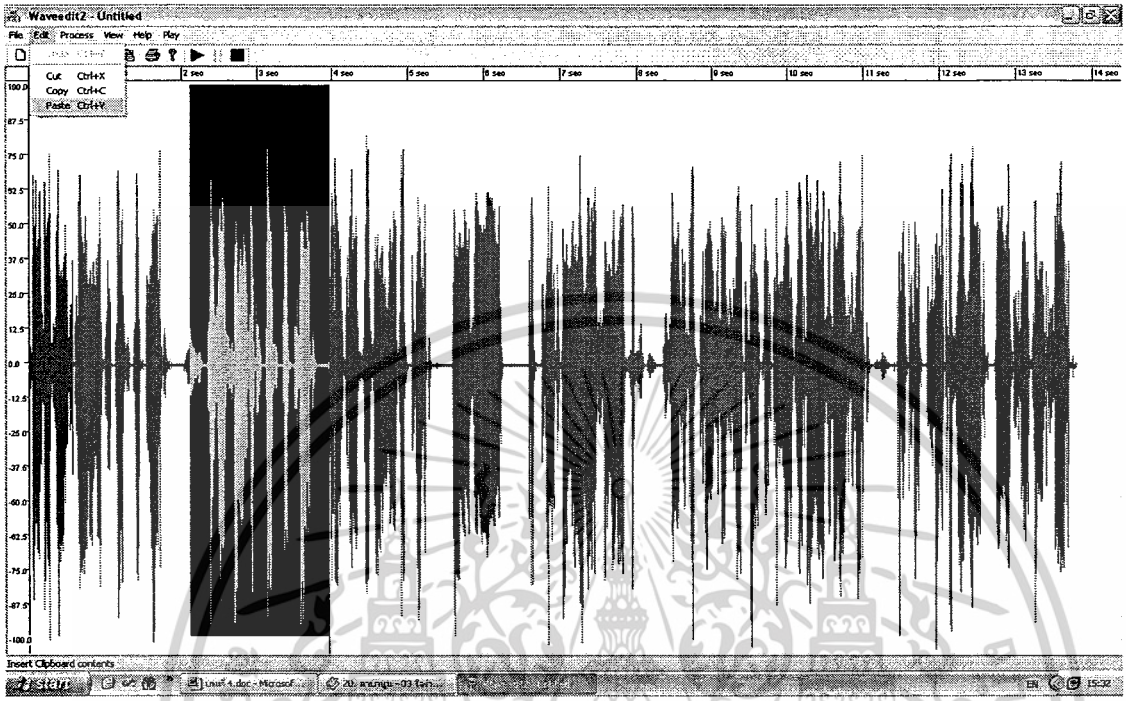


รูปที่ 4.15 แสดงการใช้ฟังก์ชัน paste กรณีแรก (Stereo)

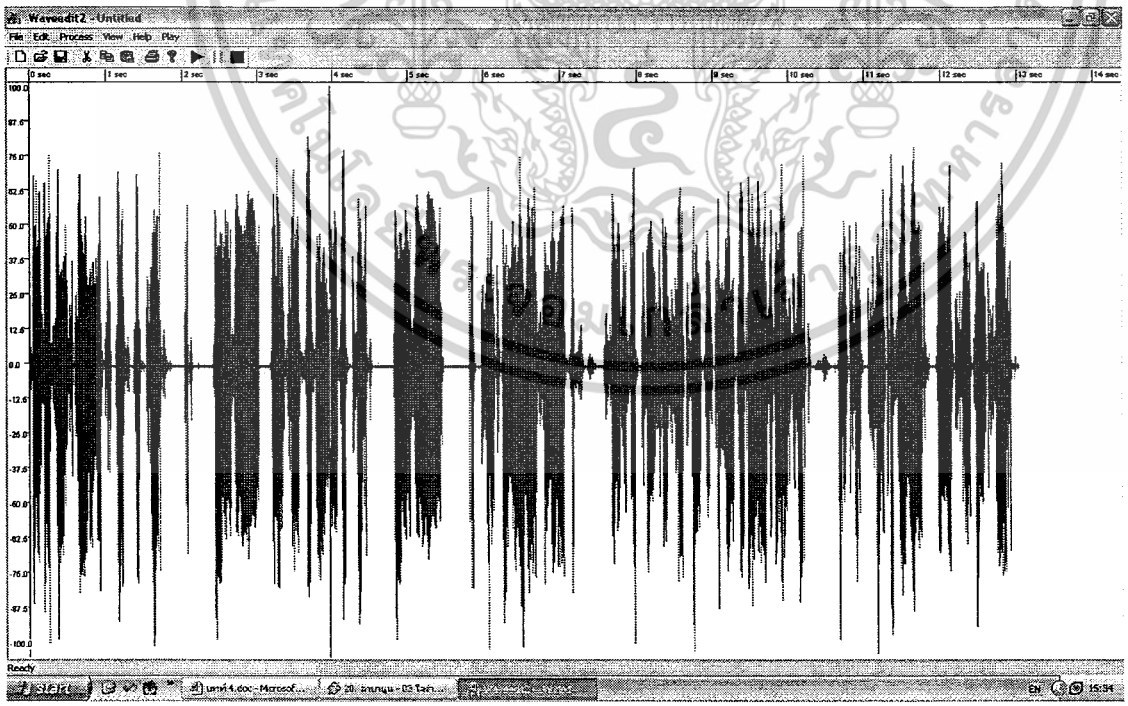


รูปที่ 4.16 แสดงหลังจากใช้ฟังก์ชัน paste กรณีแรก (Stereo)

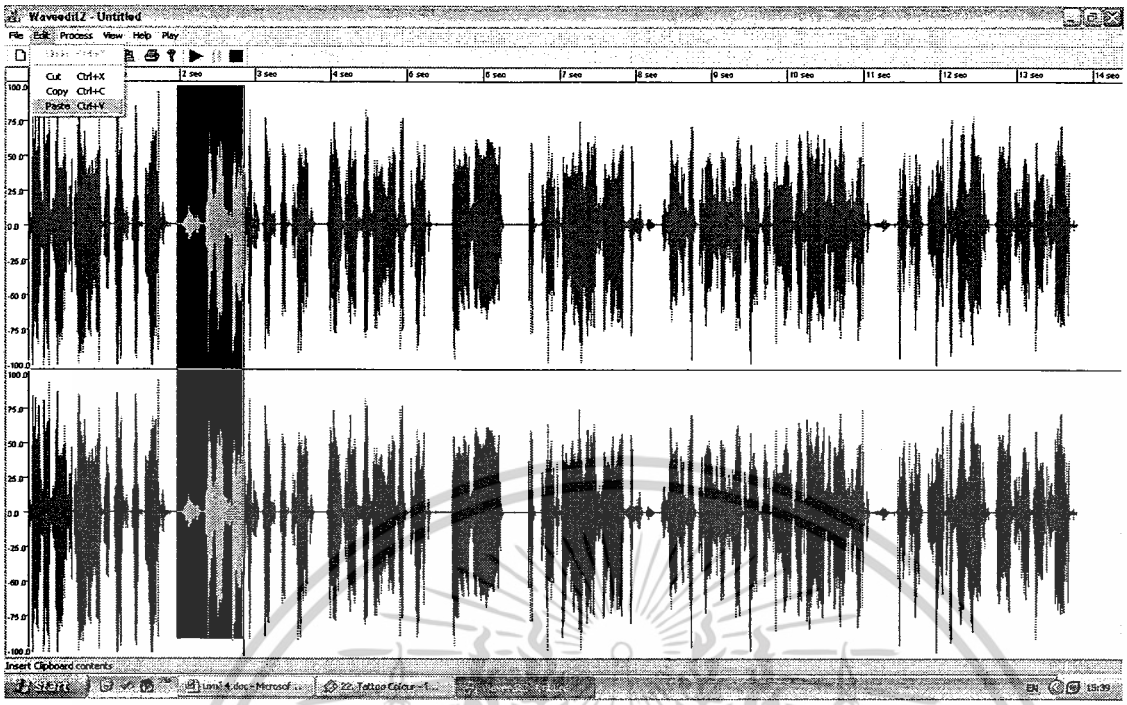
2.) เมื่อ cursor focus เป็นแถบ เมื่อการทำการ paste ข้อมูลที่อยู่ใน clipboard ก็วางแทนที่ข้อมูลในช่วงตำแหน่งนั้น



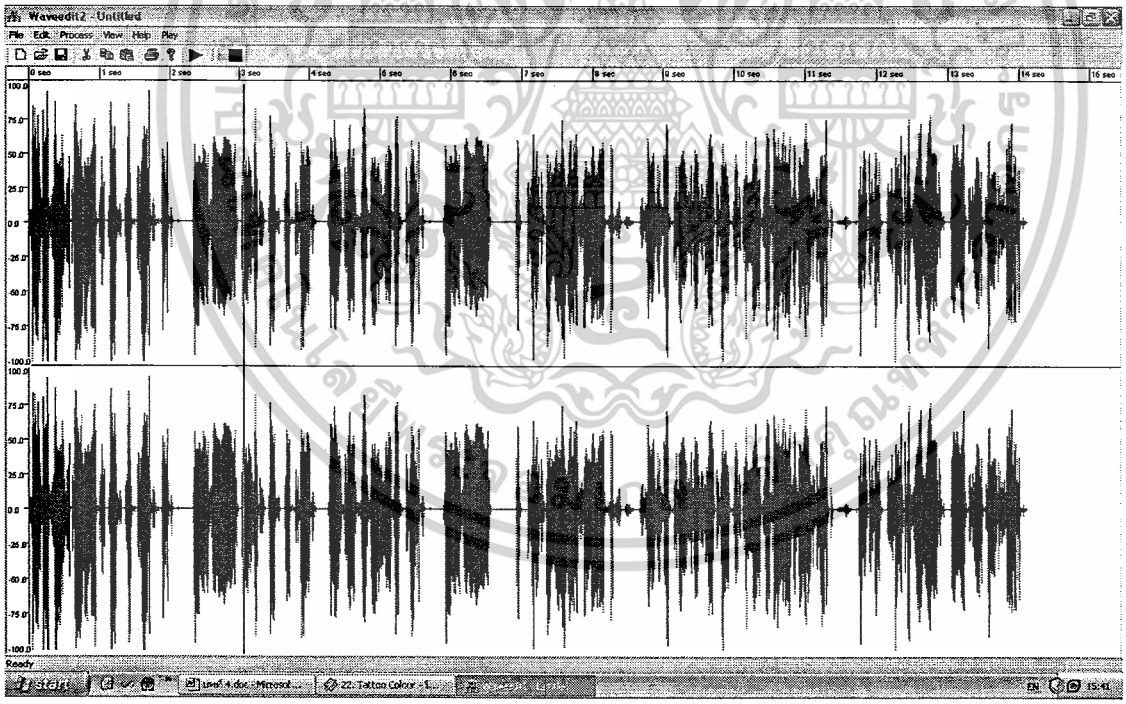
รูปที่ 4.17 แสดงการใช้ฟังก์ชัน paste กรณีสอง (Mono)



รูปที่ 4.18 แสดงหลังจากใช้ฟังก์ชัน paste กรณีสอง (Mono)



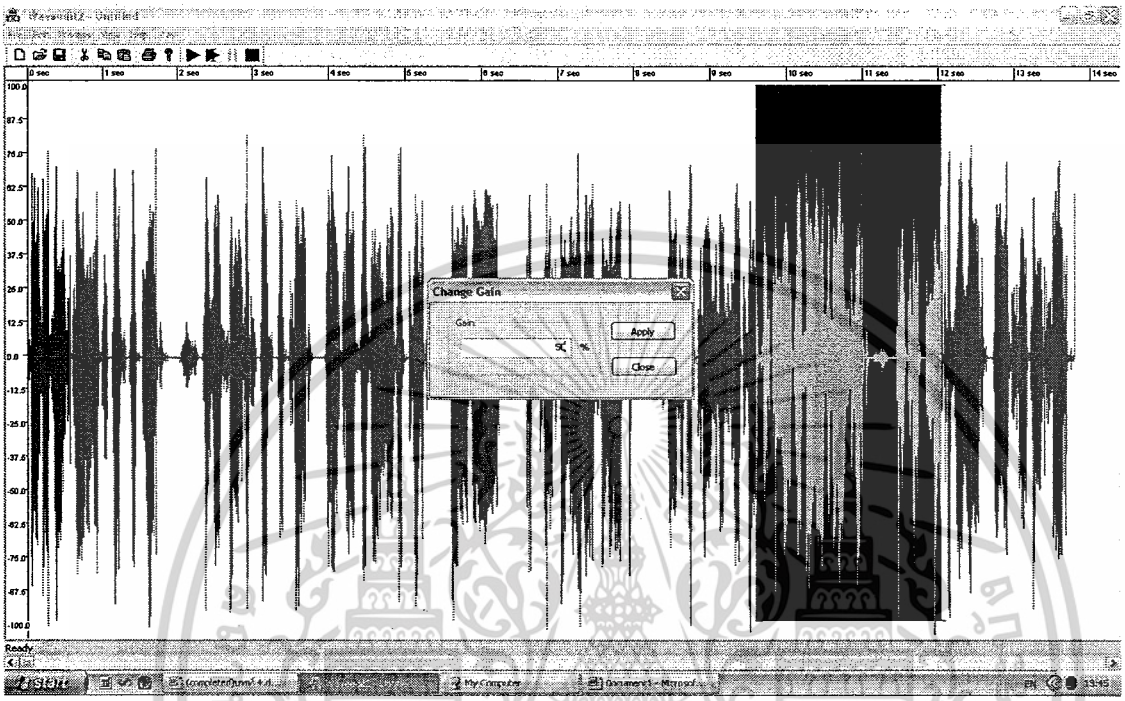
รูปที่ 4.19 แสดงการใช้ฟังก์ชัน paste กรณีสอง (Stereo)



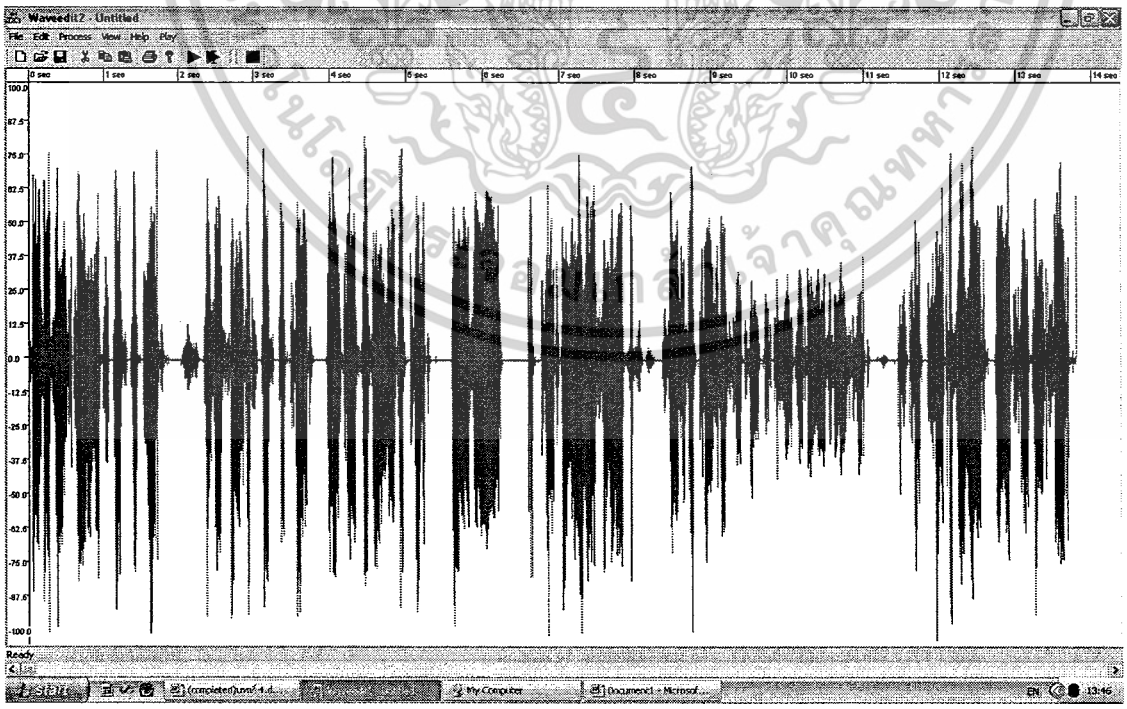
รูปที่ 4.20 แสดงหลังจากใช้ฟังก์ชัน paste กรณีสอง (Stereo)

4.6 การใช้ฟังก์ชัน change gain

เมื่อ focus ช่วงตำแหน่งที่ต้องการแล้ว ก็เลือกฟังก์ชัน change gain ซึ่งอยู่ใน process menu จากนั้น โปรแกรมก็จะสร้าง dialog box ขึ้นมาเพื่อให้ผู้ใช้ได้ป้อนอัตราการขยายเป็น % เข้าไป เมื่อกดตกลง ระดับเสียงในช่วงนั้นก็จะถูกเปลี่ยนไปตามอัตราที่ผู้ใช้กำหนด (ในที่นี้ป้อน 50% เพื่อให้ระดับเสียงลดลงครึ่งหนึ่ง)



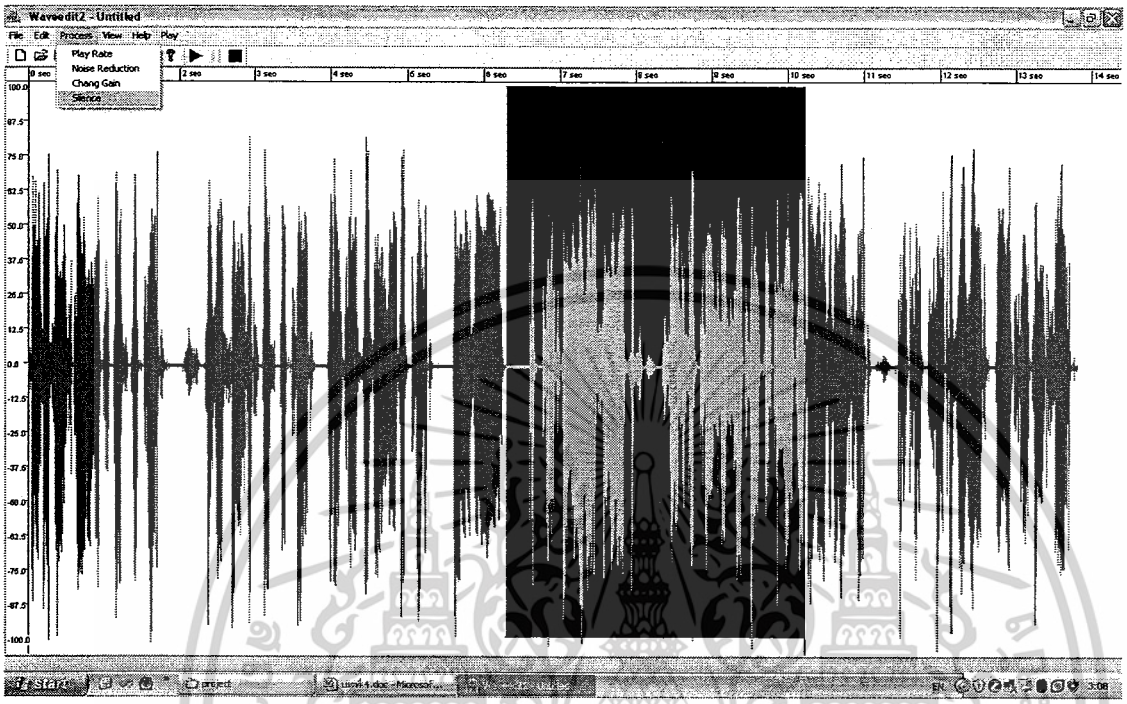
รูปที่ 4.21 แสดงการกำหนดอัตราขยายในฟังก์ชัน change gain



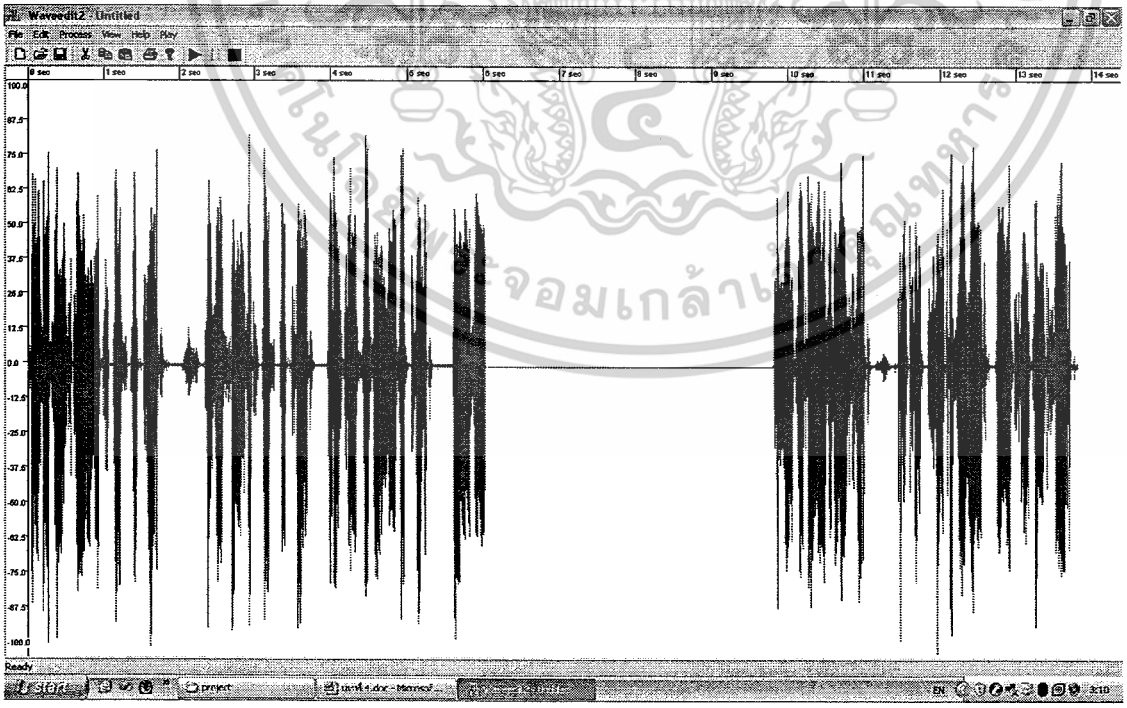
รูปที่ 4.22 แสดงระดับของสัญญาณเสียงตามอัตราการขยาย

4.7 การใช้ฟังก์ชัน silence

เมื่อ focus ช่วงตำแหน่งที่ต้องการแล้ว ก็เลือกฟังก์ชัน silence ซึ่งอยู่ใน process menu ฟังก์ชันนี้จะทำให้ระดับเสียงในช่วงนั้นลดลงไปถึงระดับต่ำสุด หรือไม่มีเสียง



รูปที่ 4.23 แสดงการใช้ฟังก์ชัน silence



รูปที่ 4.24 แสดงหลังจากใช้ฟังก์ชัน silence

บทที่ 5
สรุปผลและวิจารณ์การทดลอง

5.1 บทวิจารณ์และบทสรุป

จากโครงงานนี้ทำให้ทราบถึงหลักการในการแปลงสัญญาณเสียงซึ่งเป็นสัญญาณอนาล็อค ให้มาเป็นสัญญาณเสียงแบบดิจิตอล ด้วยอัตราการ sampling และระดับในการ quantization ซึ่งจะมีผลกับความละเอียดความชัดเจนของเสียงที่ได้ยิน แล้วทำการเก็บไว้ในรูปแบบของไฟล์ซึ่งสะดวกในการแก้ไขปรับปรุง อีกทั้งยังมีความต้านทานต่อความผิดพลาดภายในไฟล์เนื่องมาจากการส่งจากจุดหนึ่งไปยังอีกจุดหนึ่งหรือหลายๆจุด และสามารถนำไฟล์เสียงนี้ไปใช้กับอุปกรณ์อิเล็กรอนิกส์ต่างๆได้หลากหลาย

5.2 ปัญหาและอุปสรรค

โปรแกรมนี้ยังไม่สมบูรณ์ ยังไม่สามารถอ่านข้อมูลไฟล์เสียง format อื่นๆได้ และยังไม่สามารถใช้งานในอีกหลายๆฟังก์ชันได้ เช่น ฟังก์ชัน zoom, scroll, fft อีกทั้งฟังก์ชันที่มีอยู่ก็ยังทำงานได้ไม่สมบูรณ์ เนื่องจาก code โปรแกรม และฟังก์ชันต่างๆที่ใช้ มีความยากและซับซ้อนมาก ทำให้ต้องเสียเวลาในการศึกษา และการสูญเสียข้อมูล Data ไปในระหว่างการคำนวณของโปรแกรมที่มีค่า จำนวนจริง กับ จำนวนเต็ม จึงทำให้ค่าที่ได้จากการคำนวณมีความผิดเพี้ยนไป จึงทำให้การ plot และตำแหน่งของข้อมูลมีการคลาดเคลื่อนไปด้วย ทำให้ไม่สามารถแสดงองค์ประกอบของสัญญาณเสียงได้อย่างสมบูรณ์

กิตติกรรมประกาศ

โครงงานนี้สำเร็จลุล่วงไปได้ด้วยความช่วยเหลืออย่างดียิ่งของ รศ. เกรียงไกร วงศ์โรจนภรณ์ อาจารย์ที่ปรึกษาโครงงาน และ รศ.ดร สุวิพล สิทธิชีวภาค อาจารย์ที่ปรึกษาร่วม ที่ได้ให้คำแนะนำและข้อคิดเห็นต่างๆ ในการทำงานมาโดยตลอด ขอขอบคุณ นายเอกณัฐ ชูเที่ยงตรง และอาจารย์ทุกๆท่านที่ได้ให้ความอนุเคราะห์ในการใช้เครื่องมือ และให้คำแนะนำในการใช้งานเป็นอย่างดี

ผู้จัดทำจึงขอขอบพระคุณเป็นอย่างสูงมา ณ โอกาสนี้

ผู้จัดทำ

บรรณานุกรม

[1] ยุทธนา ลีลาศวัฒนกุล, เริ่มต้นการเขียนโปรแกรมด้วยภาษา C++, หจก.ไทยเจริญการพิมพ์: กรุงเทพฯ, 2547

[2] ยุทธนา ลีลาศวัฒนกุล, คู่มือการเขียนโปรแกรมและการใช้งาน Visual C++ .NET ฉบับสมบูรณ์, อินโฟเพรส: นนทบุรี, 2546

[3] นิรุธ อำนวยศิลป์, Visual C++ and MFC Programming, บริษัท ด่านสุทธาการพิมพ์ จำกัด: กรุงเทพฯ

ภาคผนวก

-------- Waveedit2Doc.h --------

```
#pragma once
#include "wave.h"
typedef struct
{
      char ID[4];
   BYTE LengthOfPackage[4];
   char FileFormat[4];
}RIFFChunk;

typedef struct
{
   char fmt[4];
   BYTE LengthOfFormatChunk[4];
   BYTE CompressionCode[2];
   BYTE ChannelNumber[2];
   BYTE SampleRate[4];
   BYTE BytesPerSecond[4];
   BYTE BytesPerSample[2];
   BYTE BitsPerSample[2];
}FormatChunk;

typedef struct
{
   char BeginData[4];
   BYTE LengthOfData[4];
}DataChunk;

class CWaveedit2Doc : public CDocument
{
protected:
   CWaveedit2Doc();
   DECLARE_DYNCREATE(CWaveedit2Doc)
public:
   virtual BOOL OnNewDocument();
   virtual void Serialize(CArchive& ar);
public:
   virtual ~CWaveedit2Doc();

#ifdef _DEBUG
   virtual void AssertValid() const;
   virtual void Dump(CDumpContext& dc) const;
#endif

protected:
   DECLARE_MESSAGE_MAP()
public:
   LPBYTE data;
   bool readed;
   int iLOD;
   float BPSec, LOD;
   short ChannelNumber, BytesPerSample;
   DWORD BytesPerSecond;
protected:
   CString m_strOpenFile;
   CWave m_Wave;
   CFile* wavfile1, * wavfile2;
public:
   afx_msg void OnPlayPlay();
```

```
    afx_msg void OnPlayPause();
    afx_msg void OnPlayStop();
    afx_msg void OnPlayRate();
public:
    void NoiseReduce(int index1, int index2);
    void Chage_Gain(float x1, float x2, float width, bool uu);
    void Copy(float x1, float x2, float width);
    void Cut(float x1, float x2, float width);
    LPBYTE cdata, ndata;
    int xx1;
    void Paste(float x1, float x2, float width);
    RIFFChunk riff1, riff2;
    FormatChunk fchunk1, fchunk2;
    DataChunk dchunk1, dchunk2;
    void OpenWavFile(void);
    void SaveWavFile(void);
    void TransferData(void);
    LPBYTE wavdata1;
    DWORD SamplingRate;
    SHORT BitsPerSample;
    DWORD ASam;
    DWORD BSam;
    LPBYTE tdata[11];
    void CopyTemp(void);
    int tiLOD[11],tSamplingRate[11],k;
    float tLOD[11];
    void Undot(void);
    void PlatSel(float x1, float x2, float width);
};
```

-------- Waveedit2Doc.cpp --------

```
#include "stdafx.h"
#include "Waveedit2.h"
#include "Waveedit2Doc.h"
#include "math.h"
#include "afxcmn.h"
#include "afxwin.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

IMPLEMENT_DYNCREATE(CWaveedit2Doc, CDocument)

BEGIN_MESSAGE_MAP(CWaveedit2Doc, CDocument)
    ON_COMMAND(ID_PLAY_PLAY, &CWaveedit2Doc::OnPlayPlay)
    ON_COMMAND(ID_PLAY_PAUSE, &CWaveedit2Doc::OnPlayPause)
    ON_COMMAND(ID_PLAY_STOP, &CWaveedit2Doc::OnPlayStop)
    ON_COMMAND(ID_PROCESS_PLAYS, &CWaveedit2Doc::OnPlayRate)
END_MESSAGE_MAP()

CWaveedit2Doc::CWaveedit2Doc()
: data(0)
, readed(false)
, BPSec(0)
, LOD(0)
, ChannelNumber(0)
, BytesPerSample(0)
, BytesPerSecond(0)
```

```
, m_strOpenFile(_T(""))
, iLOD(0)
, cdata(NULL)
, xx1(0)
, ndata(NULL)
, wavfile1(NULL)
, wavfile2(NULL)
, wavdata1(NULL)
, SamplingRate(0)
, ASam(0)
, BSam(0)
, k(-1)

{
   // TODO: add one-time construction code here

}

CWaveedit2Doc::~CWaveedit2Doc()
{
}

BOOL CWaveedit2Doc::OnNewDocument()
{
   if (!CDocument::OnNewDocument())
      return FALSE;

   // TODO: add reinitialization code here
   // (SDI documents will reuse this document)

   return TRUE;
}

void CWaveedit2Doc::Serialize(CArchive& ar)
{
   if (ar.IsStoring())
   {
      // TODO: add storing code here
      //m_Wave.Save((LPTSTR)((LPCTSTR)ar.m_strFileName));
      wavfile2 = ar.GetFile();
      TransferData();
      SaveWavFile();

   }
   else
   {
      // TODO: add loading code here
      wavfile1 = ar.GetFile();
      OpenWavFile();
   }
}

#ifdef _DEBUG
void CWaveedit2Doc::AssertValid() const
{
   CDocument::AssertValid();
}

void CWaveedit2Doc::Dump(CDumpContext& dc) const
{
   CDocument::Dump(dc);
```

```
}
#endif //_DEBUG

void CWaveedit2Doc::OpenWavFile(void)
{
   wavfile1->Read(riff1.ID,4);
   wavfile1->Read(riff1.LengthOfPackage,4);
   wavfile1->Read(riff1.FileFormat,4);
   wavfile1->Read(fchunk1.fmt,4);
   wavfile1->Read(fchunk1.LengthOfFormatChunk,4);
   wavfile1->Read(fchunk1.CompressionCode,2);
   wavfile1->Read(fchunk1.ChannelNumber,2);
   wavfile1->Read(fchunk1.SampleRate,4);
   wavfile1->Read(fchunk1.BytesPerSecond,4);
   wavfile1->Read(fchunk1.BytesPerSample,2);
   wavfile1->Read(fchunk1.BitsPerSample,2);
   wavfile1->Read(dchunk1.BeginData,4);
   wavfile1->Read(dchunk1.LengthOfData,4);
   iLOD =
(dchunk1.LengthOfData[3]*256*256*256)+(dchunk1.LengthOfData[2]*256*256)+
(dchunk1.LengthOfData[1]*256)+(dchunk1.LengthOfData[0]);
   LOD = (float)iLOD;
   if((dchunk1.BeginData[0] != 'd')&&(dchunk1.BeginData[1] !=
'a')&&(dchunk1.BeginData[2] != 't')&&(dchunk1.BeginData[3] != 'a'))
   {
      wavfile1->Seek(iLOD,CFile::current);
      wavfile1->Read(dchunk1.BeginData,4);
      wavfile1->Read(dchunk1.LengthOfData,4);
      iLOD =
(dchunk1.LengthOfData[3]*256*256*256)+(dchunk1.LengthOfData[2]*256*256)+
(dchunk1.LengthOfData[1]*256)+(dchunk1.LengthOfData[0]);
      LOD = (float)iLOD;
   }
   wavdata1 = (LPBYTE)malloc(iLOD*sizeof(BYTE));
   wavfile1->Read(wavdata1,iLOD);
   ChannelNumber = fchunk1.ChannelNumber[0];
   SamplingRate =
(fchunk1.SampleRate[3]*256*256*256)+(fchunk1.SampleRate[2]*256*256)+(fch
unk1.SampleRate[1]*256)+(fchunk1.SampleRate[0]);
   BytesPerSecond =
(fchunk1.BytesPerSecond[3]*256*256*256)+(fchunk1.BytesPerSecond[2]*256*2
56)+(fchunk1.BytesPerSecond[1]*256)+(fchunk1.BytesPerSecond[0]);
   BPSec = (float)BytesPerSecond;
   BitsPerSample = fchunk1.BitsPerSample[0];
   BytesPerSample = fchunk1.BytesPerSample[0];
   m_Wave.m_lpData = wavdata1;
   m_Wave.m_dwSize = (DWORD)iLOD;
   m_Wave.Open(ChannelNumber, SamplingRate, BitsPerSample);
   CopyTemp();
   readed = true;
}

void CWaveedit2Doc::TransferData(void)
{
   riff2 = riff1;
   ASam = SamplingRate/(256*256*256);
   BSam = SamplingRate - (ASam*256*256*256);
   fchunk1.SampleRate[3] = (BYTE)ASam;
   ASam = BSam/(256*256);
   BSam = BSam - (ASam*256*256);
   fchunk1.SampleRate[2] = (BYTE)ASam;
```

```
    ASam = BSam/256;
    BSam = BSam - (ASam*256);
    fchunk1.SampleRate[1] = (BYTE)ASam;
    fchunk1.SampleRate[0] = (BYTE)BSam;
    fchunk2 = fchunk1;
    ASam = iLOD/(256*256*256);
    BSam = iLOD-(ASam*256*256*256);
    dchunk1.LengthOfData[3]= (BYTE)ASam;
    ASam = BSam/(256*256);
    BSam = BSam - (ASam*256*256);
    dchunk1.LengthOfData[2] = (BYTE)ASam;
    ASam = BSam/256;
    BSam = BSam - (ASam*256);
    dchunk1.LengthOfData[1] = (BYTE)ASam;
    dchunk1.LengthOfData[0] = (BYTE)ASam;
    dchunk2 = dchunk1;
}

void CWaveedit2Doc::SaveWavFile(void)
{
    wavfile2->Write(riff2.ID,4);
    wavfile2->Write(riff2.LengthOfPackage,4);
    wavfile2->Write(riff2.FileFormat,4);
    wavfile2->Write(fchunk2.fmt,4);
    wavfile2->Write(fchunk2.LengthOfFormatChunk,4);
    wavfile2->Write(fchunk2.CompressionCode,2);
    wavfile2->Write(fchunk2.ChannelNumber,2);
    wavfile2->Write(fchunk2.SampleRate,4);
    wavfile2->Write(fchunk2.BytesPerSecond,4);
    wavfile2->Write(fchunk2.BytesPerSample,2);
    wavfile2->Write(fchunk2.BitsPerSample,2);
    wavfile2->Write(dchunk2.BeginData,4);
    wavfile2->Write(dchunk2.LengthOfData,4);
    wavfile2->Write(wavdata1,iLOD);
    wavfile2->Flush();
}

void CWaveedit2Doc::OnPlayPlay()
{
    // TODO: Add your command handler code here
    m_Wave.Play(wavdata1, (DWORD)LOD);
}

void CWaveedit2Doc::PlatSel(float x1, float x2, float width)
{
    int index1 = 0;
    int index2 = 0;
    LPBYTE lpdata;
    index1 = (int)(((x1-25.00)/(width-25.00))*LOD);
    index2 = (int)(((x2-25.00)/(width-25.00))*LOD);
    if(index2 == 0)
    {
        if((ChannelNumber == 1)&&(BytesPerSample == 2))
        {
            if(index1%2 == 1)
                index1 = index1-1;
            lpdata = (LPBYTE)malloc((iLOD-index1)*sizeof(BYTE));
            for(int i=0 ; i<=(iLOD-index1-1) ; i++)
            {
                lpdata[i] = wavdata1[index1+i];
            }
```

```
            m_Wave.Play(lpdata, (DWORD)(iLOD - index1));
        }
        lpdata = (LPBYTE)malloc((iLOD-index1+1)*sizeof(BYTE));
        for(int i=0 ; i<=(iLOD - index1) ; i++)
        {
            lpdata[i] = wavdata1[index1+i];
        }
        m_Wave.Play(lpdata, (DWORD)(iLOD - index1+1));
    }
    else
    {
        if((ChannelNumber == 1)&&(BytesPerSample == 2))
        {
            if(index1%2 == 1)
                index1 = index1-1;
            if(index2%2 == 1)
                index2 = index2+1;
            lpdata = (LPBYTE)malloc((index2-index1)*sizeof(BYTE));
            for(int i=0 ; i<=(index2-index1-1) ; i++)
            {
                lpdata[i] = wavdata1[index1+i];
            }
            m_Wave.Play(lpdata, (DWORD)(index2-index1));
        }
        lpdata = (LPBYTE)malloc((index2-index1+1)*sizeof(BYTE));
        for(int i=0 ; i<=(index2 - index1) ; i++)
        {
            lpdata[i] = wavdata1[index1+i];
        }
        m_Wave.Play(lpdata, (DWORD)(index2-index1+1));
    }
}

void CWaveedit2Doc::OnPlayPause()
{
    // TODO: Add your command handler code here
    m_Wave.Pause();
}

void CWaveedit2Doc::OnPlayStop()
{
    // TODO: Add your command handler code here
    m_Wave.Stop();
}

class CPlayRateDlg : public CDialog
{
public:
    CPlayRateDlg();
    enum {IDD = IDD_PLAY_RATE};

protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
    virtual BOOL OnInitDialog();

protected:
    DECLARE_MESSAGE_MAP()
public:
    DWORD m_speedrate;
    afx_msg void OnBnClickedOk();
```

```
    DWORD m_currentspeed;
};

CPlayRateDlg::CPlayRateDlg() : CDialog(CPlayRateDlg::IDD)
, m_speedrate(0)
, m_currentspeed(0)
{
}

void CPlayRateDlg::DoDataExchange(CDataExchange *pDX)
{
    CDialog::DoDataExchange(pDX);
    DDX_Text(pDX, IDC_EDIT, m_speedrate);
    DDX_Text(pDX, IDC_EDIT1, m_currentspeed);
}

BEGIN_MESSAGE_MAP(CPlayRateDlg, CDialog)
    ON_BN_CLICKED(IDOK, &CPlayRateDlg::OnBnClickedOk)
END_MESSAGE_MAP()

BOOL CPlayRateDlg::OnInitDialog()
{
    CDialog::OnInitDialog();
    m_currentspeed = m_speedrate;
    UpdateData(false);
    return TRUE;
}
void CPlayRateDlg::OnBnClickedOk()
{
    // TODO: Add your control notification handler code here
    UpdateData(true);
    OnOK();
}

void CWaveedit2Doc::OnPlayRate()
{
    // TODO: Add your command handler code here
    if(!readed) return;
    CPlayRateDlg PlayRateDlg;
    PlayRateDlg.m_speedrate = SamplingRate;
    PlayRateDlg.DoModal();
    if(SamplingRate != PlayRateDlg.m_speedrate)
    {
        SamplingRate = PlayRateDlg.m_speedrate;
        m_Wave.Open(ChannelNumber, SamplingRate, BitsPerSample);
    }
    CopyTemp();
}

void CWaveedit2Doc::NoiseReduce(int index1, int index2)
{
    float Abvalue = 0;
    float AbSum = 0;
    float DAv = 0;
    for(int i = index1 ; i < index2 ; i++)
    {
        Abvalue = abs(128.00 - wavdata1[i]);
        AbSum = AbSum + Abvalue;
    }
    DAv = 128.00 + (AbSum/2000.00);
    if(BytesPerSample == 1)
```

```
    {
        if(DAv <= 140.80) // 10%
        {
            for(int i = index1 ; i < index2 ; i++)
            {
                wavdata1[i] = 128;
            }
        }
    }
    else if(BytesPerSample == 2)
    {
        if(DAv <= 36044.80) // 10%
        {
            for(int i = index1 ; i < index2 ; i++)
            {
                wavdata1[i] = (BYTE)32768;
            }
        }
    }
    CopyTemp();
}

class CChangeGainDlg : public CDialog
{
public:
    CChangeGainDlg();
    enum {IDD = IDD_CHANGE_GAIN};

protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
    virtual BOOL OnInitDialog();

protected:
    DECLARE_MESSAGE_MAP()
public:
    afx_msg void OnBnClickedApply();
    afx_msg void OnBnClickedClose();
    float fGain;
};

CChangeGainDlg::CChangeGainDlg() : CDialog(CChangeGainDlg::IDD)
, fGain(100)
{

}

void CChangeGainDlg::DoDataExchange(CDataExchange *pDX)
{
    CDialog::DoDataExchange(pDX);
    DDX_Text(pDX, IDC_EDIT1, fGain);
}

BEGIN_MESSAGE_MAP(CChangeGainDlg, CDialog)
    ON_BN_CLICKED(IDAPPLY, &CChangeGainDlg::OnBnClickedApply)
    ON_BN_CLICKED(IDCLOSE, &CChangeGainDlg::OnBnClickedClose)
END_MESSAGE_MAP()

BOOL CChangeGainDlg::OnInitDialog()
{
    CDialog::OnInitDialog();
```

```
   UpdateData(false);
   return TRUE;
}

void CChangeGainDlg::OnBnClickedApply()
{
   // TODO: Add your control notification handler code here
   UpdateData(true);
   OnOK();
}

void CChangeGainDlg::OnBnClickedClose()
{
   // TODO: Add your control notification handler code here
   OnCancel();
}

void CWaveedit2Doc::Chage_Gain(float x1, float x2, float width, bool uu)
{
   int index1 = 0;
   int index2 = 0;
   int idata = 0;
   float fdata = 0.00;
   index1 = (int)(((x1-25.00)/(width-25.00))*LOD);
   index2 = (int)(((x2-25.00)/(width-25.00))*LOD);
   if(ndata != NULL)
      ndata = NULL;
   ndata = (LPBYTE)malloc(iLOD*sizeof(BYTE));
   if(uu == true)
   {
      CChangeGainDlg ChangeGainDlg;
      ChangeGainDlg.DoModal();
      if(ChannelNumber == 1)
      {
         if (BytesPerSample == 1)
         {
            for (int j=0 ; j<index1 ; j++)
            {
               ndata[j] = wavdata1[j];
            }
            for(int i = index1 ; i<index2 ; i++)
            {
               if(wavdata1[i] > 127)
               {
                  fdata = (float)wavdata1[i];
                  fdata = ((wavdata1[i]-
128)*(ChangeGainDlg.fGain/100))+128;
                  idata = (int)fdata;
                  if(fdata - (float)idata >= 0.5)
                     ndata[i] = (BYTE)(idata + 1);
                  else
                     ndata[i] = (BYTE)idata;
               }
               else if(wavdata1[i] <= 127)
               {
                  fdata = (float)wavdata1[i];
                  fdata = (128 -((128-
wavdata1[i])*(ChangeGainDlg.fGain/100)));
                  idata = (int)fdata;
                  if(fdata - (float)idata >= 0.5)
                     ndata[i] = (BYTE)(idata + 1);
```

```
                    else
                        ndata[i] = (BYTE)idata;
                }
            }
            for(int j=0 ; j<((int)iLOD-index2+1) ; j++)
            {
                ndata[index2+j] = wavdata1[index2+j];
            }
        }
        else if(BytesPerSample == 2)
        {
            for (int j=0 ; j<index1 ; j++)
            {
                ndata[j] = wavdata1[j];
            }
            for(int i = index1 ; i<index2 ; i++)
            {
                if(i%2 == 0)
                {
                    if((wavdata1[i]+wavdata1[i+1]*256) > 32767)
                    {
                        fdata = (float)(wavdata1[i]+(wavdata1[i+1]*256));
                        fdata = (65535-((65535-
fdata)*(ChangeGainDlg.fGain/100)));
                        idata = (int)(fdata/256.00);
                        ndata[i+1] = (BYTE)idata;
                        idata = (int)(fdata - (idata*256));
                        ndata[i] = (BYTE)idata;
                    }
                    else
                    {
                        fdata = (float)(wavdata1[i]+(wavdata1[i+1]*256));
                        fdata = (fdata)*(ChangeGainDlg.fGain/100);
                        idata = (int)(fdata/256.00);
                        ndata[i+1] = (BYTE)idata;
                        idata = (int)(fdata - (idata*256));
                        ndata[i] = (BYTE)idata;
                    }
                }
            }
            for(int j=0 ; j<((int)iLOD-index2+1) ; j++)
            {
                ndata[index2+j] = wavdata1[index2+j];
            }
        }
    }
    else
    {
        if (BytesPerSample == 2)
        {
            for (int j=0 ; j<index1 ; j++)
            {
                ndata[j] = wavdata1[j];
            }
            for(int i = index1 ; i<index2 ; i++)
            {
                if(wavdata1[i] > 127)
                {
                    fdata = (float)wavdata1[i];
                    fdata = ((wavdata1[i]-
128)*(ChangeGainDlg.fGain/100))+128;
```

```
                    idata = (int)fdata;
                    if(fdata - (float)idata >= 0.5)
                       ndata[i] = (BYTE)(idata + 1);
                    else
                       ndata[i] = (BYTE)idata;
                 }
                 else if(wavdata1[i] <= 127)
                 {
                    fdata = (float)wavdata1[i];
                    fdata = (128 -((128-
wavdata1[i])*(ChangeGainDlg.fGain/100)));
                    idata = (int)fdata;
                    if(fdata - (float)idata >= 0.5)
                       ndata[i] = (BYTE)(idata + 1);
                    else
                       ndata[i] = (BYTE)idata;
                 }
              }
              for(int j=0 ; j<((int)iLOD-index2+1) ; j++)
              {
                 ndata[index2+j] = wavdata1[index2+j];
              }
           }
        }
     }
     else
     {
        if(ChannelNumber == 1)
        {
           if(BytesPerSample == 1)
           {
              for (int j=0 ; j<index1 ; j++)
              {
                 ndata[j] = wavdata1[j];
              }
              for(int i = index1 ; i<index2 ; i++)
              {
                 ndata[i] = 128;
              }
              for(int j=0 ; j<((int)iLOD-index2+1) ; j++)
              {
                 ndata[index2+j] = wavdata1[index2+j];
              }
           }
           else
           {
              for (int j=0 ; j<index1 ; j++)
              {
                 ndata[j] = wavdata1[j];
              }
              for(int i = index1 ; i<index2 ; i++)
              {
                 if(i%2 == 0)
                 {
                    ndata[i] = 0;
                    ndata[i+1] = 0;
                 }
              }
              for(int j=0 ; j<((int)iLOD-index2+1) ; j++)
              {
                 ndata[index2+j] = wavdata1[index2+j];
```

```
            }
         }
      }
      else
      {
         if (BytesPerSample == 2)
         {
            for (int j=0 ; j<index1 ; j++)
            {
               ndata[j] = wavdata1[j];
            }
            for(int i = index1 ; i<index2 ; i++)
            {
               ndata[i] = 128;
            }
            for(int j=0 ; j<((int)iLOD-index2+1) ; j++)
            {
               ndata[index2+j] = wavdata1[index2+j];
            }
         }
      }
   }
   wavdata1 = NULL;
   m_Wave.m_lpData = NULL;
   wavdata1 = ndata;
   m_Wave.m_lpData = wavdata1;
   CopyTemp();
}

void CWaveedit2Doc::Copy(float x1, float x2, float width)
{
   int index1 = 0;
   int index2 = 0;
   index1 = (int)(((x1-25.00)/(width-25.00))*LOD);
   index2 = (int)(((x2-25.00)/(width-25.00))*LOD);
   if (cdata != NULL)
   {
      cdata = (LPBYTE)malloc((index2-index1+1)*sizeof(BYTE));
      for (int i=0 ; i < (index2-index1+1) ; i++)
      {
         cdata[i]  = wavdata1[index1+i];
      }
   }
   else
   {
      cdata = (LPBYTE)malloc((index2-index1+1)*sizeof(BYTE));
      for (int i=0 ; i < (index2-index1+1) ; i++)
      {
         cdata[i]  = wavdata1[index1+i];
      }
   }
   xx1=index2-index1+1;
}

void CWaveedit2Doc::Cut(float x1, float x2, float width)
{
   int index1 = 0;
   int index2 = 0;
   index1 = (int)(((x1-25.00)/(width-25.00))*LOD);
   index2 = (int)(((x2-25.00)/(width-25.00))*LOD);
   if (cdata != NULL)
```

```
    {
        cdata = NULL;
        cdata = (LPBYTE)malloc((index2-index1+1)*sizeof(BYTE));
        for (int i=0 ; i<(index2-index1+1) ; i++)
        {
            cdata[i]  = wavdata1[index1+i];
        }
        if(ndata != NULL)
            ndata = NULL;
        ndata = (LPBYTE)malloc((iLOD-(index2-index1+1))*sizeof(BYTE));
        for (int i=0 ; i<index1 ; i++)
        {
            ndata[i] = wavdata1[i];
        }
        for (int i=0 ; i < ((int)iLOD-index2) ; i++)
        {
            ndata[index1+i] = wavdata1[index2+1+i];
        }
    }
    else
    {
        cdata = (LPBYTE)malloc((index2-index1+1)*sizeof(BYTE));
        for (int i=0 ; i < (index2-index1+1) ; i++)
        {
            cdata[i]  = wavdata1[index1+i];
        }
        if(ndata != NULL)
            ndata = NULL;
        ndata = (LPBYTE)malloc((iLOD-(index2-index1+1))*sizeof(BYTE));
        for (int i=0 ; i<index1 ; i++)
        {
            ndata[i] = wavdata1[i];
        }
        for (int i=0 ; i < ((int)iLOD-index2) ; i++)
        {
            ndata[index1+i] = wavdata1[index2+1+i];
        }
    }
    wavdata1 = NULL;
    m_Wave.m_lpData = NULL;
    wavdata1 = ndata;
    m_Wave.m_lpData = wavdata1;
    iLOD = iLOD - (index2-index1+1);
    LOD = (float)iLOD;
    xx1= index2-index1+1;
    CopyTemp();
}

void CWaveedit2Doc::Paste(float x1, float x2, float width)
{
    int index1 = 0;
    int index2 = 0;
    index1 = (int)(((x1-25.00)/(width-25.00))*LOD);
    index2 = (int)(((x2-25.00)/(width-25.00))*LOD);
    if(index2 == 0)
    {
        if(ndata != NULL)
        {
            ndata = NULL;//free(ndata);
            ndata = (LPBYTE)malloc((iLOD+xx1)*sizeof(BYTE));
        }
```

```
      else
      {
         ndata = (LPBYTE)malloc((iLOD+xx1)*sizeof(BYTE));
      }
      for(int i=0 ; i< index1 ;i++)
      {
         ndata[i] = wavdata1[i] ;
      }
      for (int i =0 ; i< xx1 ;i++)
      {
         ndata[index1+i] = cdata[i];
      }
      for (int i =0 ;i< iLOD-index1; i++)
      {
         ndata[index1+xx1+i] = wavdata1[index1+i];
      }
      wavdata1 = NULL;//free(data);
      wavdata1 = (LPBYTE)malloc((iLOD+xx1)*sizeof(BYTE));
      iLOD = iLOD+xx1;
   }
   else
   {
      if(ndata != NULL)
      {
         ndata = NULL;
         ndata = (LPBYTE)malloc((iLOD+xx1-(index2-
index1+1))*sizeof(BYTE));
      }
      else
      {
         ndata = (LPBYTE)malloc((iLOD+xx1-(index2-
index1+1))*sizeof(BYTE));
      }
      for (int i=0; i<index1 ;i++)
      {
         ndata[i] = wavdata1[i];
      }
      for (int i=0 ; i < xx1 ;i++)
      {
         ndata[index1+i] = cdata[i];
      }
      for (int i =0 ; i < iLOD-index2-1 ;i++)
      {
         ndata[index1+xx1+i] = wavdata1[index2+i];
      }
      wavdata1 = NULL;//free(data);
      wavdata1 = (LPBYTE)malloc((iLOD+xx1-(index2-
index1+1))*sizeof(BYTE));
      iLOD = iLOD+xx1-(index2-index1+1);
   }
   wavdata1 = ndata;
   m_Wave.m_lpData = NULL;
   m_Wave.m_lpData = wavdata1;
   LOD = (float)iLOD;
   m_Wave.m_dwSize = (DWORD)LOD;
   CopyTemp();
}
void CWaveedit2Doc::CopyTemp(void)
{
   if(k >= 9)
   {
```

```
        k = 9;
        for(int i=0 ; i<=8 ; i++)
        {
           tiLOD[i] = tiLOD[i+1];
           tLOD[i] = tLOD[i+1];
           tSamplingRate[i] = tSamplingRate[i+1];
           tdata[i] = NULL;
           tdata[i] = tdata[i+1];
        }
        tiLOD[k] = iLOD;
        tLOD[k] =LOD;
        tSamplingRate[k] = SamplingRate;
        if(tdata[k] != NULL)
        {
           tdata[k] = NULL;
        }
        tdata[k] =(LPBYTE)malloc(tiLOD[k]*sizeof(BYTE));
        tdata[k] = wavdata1;
    }
    k = k + 1;
    tiLOD[k] = iLOD;
    tLOD[k] = LOD;
    tSamplingRate[k] = SamplingRate;
    if(tdata[k] != NULL)
    {
       tdata[k] = NULL;
    }
    tdata[k] =(LPBYTE)malloc(tiLOD[k]*sizeof(BYTE));
    tdata[k] = wavdata1;
}

void CWaveedit2Doc::Undot(void)
{
    if(k <= 0 ) return;
    k = k - 1;
    iLOD = tiLOD[k];
    LOD = tLOD[k];
    SamplingRate = tSamplingRate[k];
    wavdata1 = tdata[k];
    m_Wave.m_lpData = wavdata1;

}
```

-------- Waveedit2View.h --------

```
#pragma once
#include "atltypes.h"


class CWaveedit2View : public CScrollView
{
protected: // create from serialization only
    CWaveedit2View();
    DECLARE_DYNCREATE(CWaveedit2View)
public:
    CWaveedit2Doc* GetDocument() const;
public:
    virtual void OnDraw(CDC* pDC);  // overridden to draw this view
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
protected:
    virtual void OnInitialUpdate(); // called first time after construct
```

```
    virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
    virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
    virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
public:
    virtual ~CWaveedit2View();

#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:
    DECLARE_MESSAGE_MAP()
private:
    float x,y,cx,cy,m,n,ii,t,amp;
    int tt,period;
    bool m_MouseSel,isComm;
public:
    afx_msg void OnEditNoisereduction();
    afx_msg void OnProcessChanggain();
    afx_msg void OnProcessSilence();
    afx_msg void OnEditCut();
    afx_msg void OnEditCopy();
    afx_msg void OnEditPaste();
    afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
    afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
    afx_msg void OnMouseMove(UINT nFlags, CPoint point);
    int PosX, PosX1, PosX2;
    int m_selpoint;
    int CaretHeight;
    int OldPosX;
    CPoint old;
    void Draw(CDC* pDC);
    afx_msg void OnHScroll(UINT nSBCode, UINT nPos, CScrollBar*
pScrollBar);
    bool IsFocus;
    afx_msg void OnEditUndo();
    afx_msg void OnPlayPlaysel();
};

#ifndef _DEBUG  // debug version in Waveedit2View.cpp
inline CWaveedit2Doc* CWaveedit2View::GetDocument() const
    { return reinterpret_cast<CWaveedit2Doc*>(m_pDocument); }
#endif
```

-------- Waveedit2View.cpp --------

```
// Waveedit2View.cpp : implementation of the CWaveedit2View class
//

#include "stdafx.h"
#include "Waveedit2.h"

#include "Waveedit2Doc.h"
#include "Waveedit2View.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif
```

```
// CWaveedit2View

IMPLEMENT_DYNCREATE(CWaveedit2View, CScrollView)

BEGIN_MESSAGE_MAP(CWaveedit2View, CScrollView)
   // Standard printing commands
   ON_COMMAND(ID_FILE_PRINT, &CScrollView::OnFilePrint)
   ON_COMMAND(ID_FILE_PRINT_DIRECT, &CScrollView::OnFilePrint)
   ON_COMMAND(ID_FILE_PRINT_PREVIEW, &CScrollView::OnFilePrintPreview)
   ON_COMMAND(ID_PROCESS_NOISEREDUCTION,
&CWaveedit2View::OnEditNoisereduction)
   ON_WM_LBUTTONDOWN()
   ON_WM_LBUTTONUP()
   ON_WM_MOUSEMOVE()
   //ON_WM_SETFOCUS()
   //ON_WM_KILLFOCUS()
   ON_COMMAND(ID_PROCESS_CHANGGAIN, &CWaveedit2View::OnProcessChanggain)
   ON_COMMAND(ID_EDIT_CUT, &CWaveedit2View::OnEditCut)
   ON_COMMAND(ID_EDIT_COPY, &CWaveedit2View::OnEditCopy)
   ON_COMMAND(ID_EDIT_PASTE, &CWaveedit2View::OnEditPaste)
   ON_COMMAND(ID_PROCESS_SILENCE, &CWaveedit2View::OnProcessSilence)
   ON_WM_HSCROLL()
   ON_COMMAND(ID_EDIT_UNDO, &CWaveedit2View::OnEditUndo)
   ON_COMMAND(ID_PLAY_PLAYSEL, &CWaveedit2View::OnPlayPlaysel)
END_MESSAGE_MAP()

// CWaveedit2View construction/destruction

CWaveedit2View::CWaveedit2View()
: x(0)
, y(0)
, cx(0)
, cy(0)
, m(0)
, n(0)
, ii(0)
, t(0)
, amp(0)
, tt(0)
, period(0)
, m_MouseSel(false)
, isComm(false)
, PosX(25)
, m_selpoint(0)
, CaretHeight(0)
, old(0)
, OldPosX(25)
, PosX1(0)
, PosX2(0)
, IsFocus(false)
{
   // TODO: add construction code here
   old.x = 25;

}

CWaveedit2View::~CWaveedit2View()
{
}

BOOL CWaveedit2View::PreCreateWindow(CREATESTRUCT& cs)
```

```
{
   // TODO: Modify the Window class or styles here by modifying
   //  the CREATESTRUCT cs

   return CScrollView::PreCreateWindow(cs);
}

// CWaveedit2View drawing

void CWaveedit2View::OnDraw(CDC* pDC)
{
   Draw(pDC);
}

void CWaveedit2View::Draw(CDC* pDC)
{
   CWaveedit2Doc* pDoc = GetDocument();
   ASSERT_VALID(pDoc);
   if (!pDoc)
      return;

   // TODO: add draw code for native data here
   RECT r;
   GetClientRect (&r);
   CaretHeight = r.bottom - 20;
   CFont font, *oldfont;
   CPen pen, *oldpen;
   CString stramp, strperiod;

   if (pDoc->readed)
   {
      t = (pDoc->LOD)/(pDoc->BPSec);
      if((pDoc->iLOD) % ((int)pDoc->BPSec) == 0)
         tt = (int)t;
      else
         tt = (int)(t+1.00);
      cx = (r.right-50)/tt;
      pDC->MoveTo(1,15);
      pDC->LineTo(r.right,15);
      pDC->MoveTo(25,1);
      pDC->LineTo(25,r.bottom);

   font.CreateFontW(12,4,0,0,FW_NORMAL,FALSE,FALSE,0,ANSI_CHARSET,OUT_DE
FAULT_PRECIS,CLIP_DEFAULT_PRECIS,DEFAULT_QUALITY,DEFAULT_PITCH|FF_SWISS,
_T("AngsanaNEW"));
      oldfont = pDC->SelectObject(&font);
      ii = 1;
      period = 0;
      pDC->TextOutW(28,1,_T("0 sec"));
      for (int i = 0 ; i<tt ; i++)
      {
         pDC->MoveTo((25+(cx*ii)),1);
         pDC->LineTo((25+(cx*ii)),15);
         period = period + 1;
         strperiod.Format(_T("%d sec"),period);
         pDC->TextOutW((28+(cx*ii)),1,strperiod);
         ii++;
      }
      if(pDoc->ChannelNumber == 1)
      {
         ii = 0;
```

```
            amp =112.50;
            for (int i = 0 ;i < 17 ; i++)
            {
               pDC->MoveTo(20,(20+(ii*((r.bottom-25)/16))));
               pDC->LineTo(25,(20+(ii*((r.bottom-25)/16))));
               amp = amp - 12.50;
               stramp.Format(_T("%.1f"),amp);
               pDC->TextOutW(2,(17+(ii*((r.bottom-25)/16))),stramp);
               ii++;
            }
         }
         else if(pDoc->ChannelNumber == 2)
         {
            pDC->MoveTo(25,(15+((r.bottom-15)/2)));
            pDC->LineTo(r.right,(15+((r.bottom-15)/2)));
            ii = 0;
            amp = 125.00;
            for(int i = 0 ; i < 9 ; i++)
            {
               pDC->MoveTo(20,(20+(ii*((((r.bottom-15)/2)-10)/8))));
               pDC->LineTo(25,(20+(ii*((((r.bottom-15)/2)-10)/8))));
               amp = amp -25.0;
               stramp.Format(_T("%.1f"),amp);
               pDC->TextOutW(2,(17+(ii*((((r.bottom-15)/2)-
10)/8))),stramp);
               ii++;
            }
            ii = 0;
            amp = 125.0;
            for (int i = 0 ; i < 9 ; i++)
            {
               pDC->MoveTo(20,((20+((r.bottom-15)/2))+(ii*((((r.bottom-
15)/2)-10)/8))));
               pDC->LineTo(25,((20+((r.bottom-15)/2))+(ii*((((r.bottom-
15)/2)-10)/8))));
               amp = amp - 25.0;
               stramp.Format(_T("%.1f"),amp);
               pDC->TextOutW(2,((17+((r.bottom-15)/2))+(ii*((((r.bottom-
15)/2)-10)/8))),stramp);
               ii++;
            }
         }
         pDC->SelectObject(oldfont);
         if(pDoc->ChannelNumber == 1)
         {
            ii = 2;
            pen.CreatePen(PS_SOLID,0,RGB(0,90,180));
            oldpen = pDC->SelectObject(&pen);
            if (pDoc->BytesPerSample == 1)
            {
               for (int i = 0; i < (pDoc->LOD-1) ; i++)
               {
                  if (i == 0)
                  {
                     n = pDoc->wavdata1[i];
                     m = n/255;
                     y = (r.bottom-5)-(m*(r.bottom-25));
                     pDC->MoveTo(25,(int)y);
                  }
                  n = pDoc->wavdata1[i+1];
                  m = n/255;
```

```
                x = 25 +((ii/((pDoc->LOD-2)))*(cx*t));
                y = (r.bottom-5)-(m*(r.bottom-25));
                pDC->LineTo((int)x,(int)y);
                pDC->MoveTo((int)x,(int)y);
                ii++;
             }
          }
          else if (pDoc->BytesPerSample == 2)
          {
             for (int i = 0 ; i < (pDoc->LOD) ; i++)
             {
                if (i%2 == 0)
                {
                   if (i == 0)
                   {
                      n = (pDoc->wavdata1[i]) + (pDoc-
>wavdata1[i+1]*256);
                      if(n<32768)
                      {
                         m = n/32767;
                         y = (15+((r.bottom-15)/2)) - (m*((r.bottom-
25)/2));
                         pDC->MoveTo(25,(int)y);
                      }
                      else
                      {
                         n = 65535 - n;
                         m = n/32767;
                         y = (15+(r.bottom-15)/2) + (m*((r.bottom-
25)/2));
                         pDC->MoveTo(25,(int)y);
                      }
                   }
                   n = (pDoc->wavdata1[i+2]) + (pDoc->wavdata1[i+3]*256);
                   if(n<32767)
                   {
                      m = n/32767;
                      x = 25 +(ii/((pDoc->LOD)/2))*(cx*t);
                      y = (15+((r.bottom-15)/2)) - (m*((r.bottom-25)/2));
                      pDC->LineTo((int)x,(int)y);
                      pDC->MoveTo((int)x,(int)y);
                      ii++;
                   }
                   else
                   {
                      n = 65535 - n;
                      m = n/32767;
                      x = 25 +(ii/((pDoc->LOD)/2))*(cx*t);
                      y = (15+(r.bottom-15)/2) + (m*((r.bottom-25)/2));
                      pDC->LineTo((int)x,(int)y);
                      pDC->MoveTo((int)x,(int)y);
                      ii++;
                   }
                }
             }
          }
       }
       else if (pDoc->ChannelNumber == 2) // plot Stereo
       {
          pen.CreatePen(PS_SOLID,0,RGB(0,90,180));
          oldpen = pDC->SelectObject(&pen);
```

```
         if(pDoc->BytesPerSample == 2)
         {
            ii = 2;
            for (int i = 0 ; i < ((pDoc->LOD)-2) ; i++)
            {
               if(i%2 == 0)
               {
                  if(i == 0)
                  {
                     n = pDoc->wavdata1[i];
                     m = n/255;
                     y = (10+((r.bottom-15)/2))-(m*(((r.bottom-15)/2)-
10));
                     pDC->MoveTo(25,(int)y);
                  }
                  n = pDoc->wavdata1[i+2];
                  m = n/255;
                  x = 25+(ii/((pDoc->LOD)/2))*(cx*t);
                  y = (10+((r.bottom-15)/2))-(m*(((r.bottom-15)/2)-10));
                  pDC->LineTo((int)x,(int)y);
                  pDC->MoveTo((int)x,(int)y);
                  ii++;
               }
            }
            ii = 2;
            for (int i = 0 ; i < ((pDoc->LOD)-1) ; i++)
            {
               if(i%2 == 1)
               {
                  if(i == 1)
                  {
                     n = pDoc->wavdata1[i];
                     m = n/255;
                     y = (r.bottom-5)-(m*(((r.bottom-15)/2)-10));
                     pDC->MoveTo(25,(int)y);
                  }
                  n = pDoc->wavdata1[i+2];
                  m = n/255;
                  x = 25+(ii/((pDoc->LOD)/2))*(cx*t);
                  y = (r.bottom-5)-(m*(((r.bottom-15)/2)-10));
                  pDC->LineTo((int)x,(int)y);
                  pDC->MoveTo((int)x,(int)y);
                  ii++;
               }
            }
         }
      }
      pDC->SelectObject(oldpen);
      m_MouseSel = true;
      isComm = true;
   }
}

void CWaveedit2View::OnInitialUpdate()
{
   CScrollView::OnInitialUpdate();

   CSize sizeTotal;
   // TODO: calculate the total size of this view
   RECT r;
   GetClientRect(&r);
```

```
    sizeTotal.cx = r.right;
    sizeTotal.cy = r.bottom;
    SetScrollSizes(MM_TEXT, sizeTotal);
}


// CWaveedit2View printing

BOOL CWaveedit2View::OnPreparePrinting(CPrintInfo* pInfo)
{
    // default preparation
    return DoPreparePrinting(pInfo);
}

void CWaveedit2View::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo*
/*pInfo*/)
{
    // TODO: add extra initialization before printing
}

void CWaveedit2View::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
    // TODO: add cleanup after printing
}


// CWaveedit2View diagnostics

#ifdef _DEBUG
void CWaveedit2View::AssertValid() const
{
    CScrollView::AssertValid();
}

void CWaveedit2View::Dump(CDumpContext& dc) const
{
    CScrollView::Dump(dc);
}

CWaveedit2Doc* CWaveedit2View::GetDocument() const // non-debug version
is inline
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CWaveedit2Doc)));
    return (CWaveedit2Doc*)m_pDocument;
}
#endif //_DEBUG


// CWaveedit2View message handlers

void CWaveedit2View::OnEditNoisereduction()
{
    // TODO: Add your command handler code here
    CWaveedit2Doc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    int z = 0;
    int zz = 0;
    int zzz = 0;
    int n = 0;
    int nn = 2000;
    z = pDoc->iLOD/2000;
```

```
    zz = z + 1;
    zzz = pDoc->iLOD - (2000*z);
    for (int i = 0 ; i < zz+1 ; i++)
    {
        if(i==zz)
            pDoc->NoiseReduce(n, pDoc->iLOD);
        else
        {
            pDoc->NoiseReduce(n, nn);
            n = n + 2000;
            nn = nn + 2000;
        }
    }
    InvalidateRect(NULL);
}
void CWaveedit2View::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: Add your message handler code here and/or call default
    if(!m_MouseSel) return;
    CClientDC clientDC(this);
    OnPrepareDC(&clientDC);
    clientDC.DPtoLP(&point);
    DestroyCaret();
    SetCapture();
    if(IsFocus)
    {
        InvalidateRect(NULL);
        IsFocus = false;
    }
    if(point.x>24)
    {
        PosX2 = 25;
        PosX = point.x;
        PosX1 = PosX;
        HWND hwnd = GetSafeHwnd();
        ::CreateCaret(hwnd,NULL,0,CaretHeight);
        ::SetCaretPos(PosX,20);
    }

    CScrollView::OnLButtonDown(nFlags, point);
}

void CWaveedit2View::OnLButtonUp(UINT nFlags, CPoint point)
{
    // TODO: Add your message handler code here and/or call default
    if(!m_MouseSel) return;
    if(GetCapture() == this)
        ReleaseCapture();
    ShowCaret();
    CScrollView::OnLButtonUp(nFlags, point);
}

void CWaveedit2View::OnMouseMove(UINT nFlags, CPoint point)
{
    // TODO: Add your message handler code here and/or call default
    if(!m_MouseSel) return;
    CClientDC clientDC(this);
    OnPrepareDC(&clientDC);
    clientDC.DPtoLP(&point);
    if((nFlags & MK_LBUTTON) && (this == GetCapture()))
    {
```

```
        CBrush brush, *oldbrush;
        brush.CreateSolidBrush(COLORREF(0));
        oldbrush = clientDC.SelectObject(&brush);
        clientDC.SetROP2(R2_NOT);
        old = GetCaretPos();
        if(old.x<24)
           OldPosX = 25;
        else
           OldPosX = old.x;
        if(point.x!=PosX)
        {
           //HideCaret();
           if(point.x>24)
           {
              clientDC.Rectangle(OldPosX,20,point.x,CaretHeight);
              ::SetCaretPos(point.x,20);
              if(PosX<point.x)
              {
                 PosX1 = PosX;
                 PosX2 = point.x;
              }
              else
              {
                 PosX1 = point.x;
                 PosX2 = PosX;
              }
           }
           else
           {
              clientDC.Rectangle(OldPosX,20,25,CaretHeight);
              ::SetCaretPos(25,20);
              PosX1 = 25;
              PosX2 = OldPosX;
           }
           IsFocus = true;
           //ShowCaret();
        }
        else
        {
           clientDC.Rectangle(PosX,20,OldPosX,CaretHeight);
        }

        clientDC.SelectObject(oldbrush);
     }
     CScrollView::OnMouseMove(nFlags, point);
}

void CWaveedit2View::OnProcessChanggain()
{
     // TODO: Add your command handler code here
     CWaveedit2Doc* pDoc = GetDocument();
     ASSERT_VALID(pDoc);
     RECT rect;
     GetClientRect(&rect);
     pDoc->Chage_Gain((float)PosX1,(float)PosX2,(float)rect.right,true);
     InvalidateRect(NULL);
}

void CWaveedit2View::OnProcessSilence()
{
     // TODO: Add your command handler code here
}
```

```
    CWaveedit2Doc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    RECT rect;
    GetClientRect(&rect);
    pDoc->Chage_Gain((float)PosX1,(float)PosX2,(float)rect.right,false);
    InvalidateRect(NULL);
}

void CWaveedit2View::OnEditCut()
{
    // TODO: Add your command handler code here
    if(PosX2 <= PosX1) return;
    CWaveedit2Doc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    RECT rect;
    GetClientRect(&rect);
    pDoc->Cut((float)PosX1,(float)PosX2,(float)rect.right);
    InvalidateRect(NULL);
    PosX1 = 25;
    PosX2 = 25;
}

void CWaveedit2View::OnEditCopy()
{
    // TODO: Add your command handler code here
    if(PosX2 <= PosX1) return;
    CWaveedit2Doc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    RECT rect;
    GetClientRect(&rect);
    pDoc->Copy((float)PosX1,(float)PosX2,(float)rect.right);
}

void CWaveedit2View::OnEditPaste()
{
    // TODO: Add your command handler code here
    CWaveedit2Doc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    RECT rect;
    GetClientRect(&rect);
    pDoc->Paste((float)PosX1,(float)PosX2,(float)rect.right);
    InvalidateRect(NULL);
}
void CWaveedit2View::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar*
pScrollBar)
{
    // TODO: Add your message handler code here and/or call default

    CScrollView::OnHScroll(nSBCode, nPos, pScrollBar);
}

void CWaveedit2View::OnEditUndo()
{
    // TODO: Add your command handler code here
    CWaveedit2Doc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    pDoc->Undot();
    InvalidateRect(NULL);
}

void CWaveedit2View::OnPlayPlaysel()
```

```
{
    // TODO: Add your command handler code here
    CWaveedit2Doc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    RECT rect;
    GetClientRect(&rect);
    pDoc->PlatSel((float)PosX1,(float)PosX2,(float)rect.right);
}
```