ตัวถอดรหัสเสียงเอ็มเป็ก

MPEG AUDIO DECODER

โดย

นาย พิพัฒพงศ์ เกื้อศิริกุล รหัส38014345

นาย สุวสิต วิทยวิจักษณ์ รหัส38014597

อาจารย์ที่ปรึกษา

อาจารย์ ชินภัทร นันทุจิวากรชัย

ปริญญานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2541

ปริญญานิพนธ์ ปีการศึกษา 2541

ภาควิชา อิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ตัวถอดรหัสเสียงเอ็มเป็ก

ผู้จัดทำ

1. นาย พิพัฒพงศ์ เกื้อศิริกุล
2. นาย สุวสิต วิทยวิจักษณ์

.....อาจารย์ที่ปรึกษา

(อาจารย์ ชินภัทร นันทจิวากรชัย)

ตัวถอดรหัสเสียงเอ็มเป็ก

MPEG AUDIO DECODER

นาย พิพัฒพงศ์ เกื้อศิริกุล รหัส 38014345

นาย สุวสิต วิทยวิจักษณ์ รหัส 38014597

โครงงานได้รับการตรวจสอบแล้ว พร้อมที่จะทำการสอบได้

.....

(อาจารย์ ชินภัทร นันทจิวากรชัย)

อาจารย์ที่ปรึกษา

ตัวถอดรหัสข้อมูลเอ็มเป็กเลเยอร์ 3

นาย พิพัฒพงศ์ เกื้อศิริกุล
นาย สุวสิต วิทยวิจักษณ์
อ.ชินภัทร นันทจิวากรชัย อาจารย์ที่ปรึกษา
ปีการศึกษา 2541

บทคัดย่อ

ปริญญานิพนธ์เรื่องตัวถอดรหัสเอ็มเป็กเลเยอร์ 3 นี้ ประกอบด้วยเนื้อหาหลักสามส่วน ส่วนแรกเป็นหลักการทั่วไปของการเข้ารหัสข้อมูลเอ็มเป็กเลเยอร์ 3 ส่วนที่สองเป็นการออกแบบโปรแกรมถอดรหัสข้อมูลเอ็มเป็กเลเยอร์ 3 ด้วยภาษาซีบนเครื่องคอมพิวเตอร์ ส่วนสุดท้ายคือการทดลองใช้ดีเอสพีประมวลผลการถอดรหัสแทนหน่วยประมวลผลกลางของเครื่องคอมพิวเตอร์ แล้วนำผลการทดลองไปประเมินผลเพื่อประยุกต์ใช้ ในการสร้างตัวถอดรหัสข้อมูลเอ็มเป็กเลเยอร์ 3 โดยใช้ดีเอสพีโดยตรงในอนาคต

MPEG LAYER DECODER

Pipatpong Kuesirikul

Suwasit Witayawijug

Chinnapat Nuntawakornchai Advisor

1998

ABTRACT

This thesis consists of three main parts . The first part cercerns to the theory of MPEG1-layer3 audio compression. Second is designing of programme in C language for decoding MPEG1 layer3 file on PC. Finally, The DSP is use to process instead of the PC and summarize the effect and attempt to apply it for creating the stand alone MPEG layer3 decoder.

สารบัญ

หน้าที่

หน้าที่

สารบัญภาพ

หน้าที่

หน้าที่

สารบัญตาราง

บทที่ 1
บทนำ

1.1 กล่าวนำ

ในปัจจุบันเทคโนโลยีทางด้านคอมพิวเตอร์ไม่ว่าจะเป็นฮาร์ดแวร์(Hardware) หรือซอร์ฟแวร์(Software) ได้ถูกพัฒนาขึ้นอย่างต่อเนื่องและรวดเร็ว แต่จากพัฒนานี้ทำให้ข้อมูลต่างๆ มีขนาดใหญ่ขึ้นตามไปด้วย การบีบอัดหรือการเข้ารหัสข้อมูล(Compress or Encode)จึงมีความจำเป็นมากขึ้น การบีบอัดข้อมูลก็ได้ถูกพัฒนาขึ้นเช่นกันซึ่งมีรูปแบบและหลักการต่างกันไปตามการใช้งาน การเข้ารหัสข้อมูลเสียงแบบเอ็มเป็คเป็นเทคโนโลยีการบีบอัดข้อมูลชนิดหนึ่งที่มีประสิทธิภาพสูงมาก ซึ่งสามารถนำไปประยุกต์ใช้ในงานด้านการสื่อสาร การเก็บรักษาข้อมูล สื่อกลางต่างๆ ได้มากมาย

ผู้จัดทำปริญญานิพนธ์เล็งเห็นถึงคุณประโยชน์ของการเข้ารหัสข้อมูลชนิดนี้ จึงจัดทำปริญญานิพนธ์ เรื่องตัวถอดรหัสเอ็มเป็กเลเยอร์3(Mpeg Layer-3) ขึ้นเพื่อศึกษาและรวบรวมข้อมูลเกี่ยวกับการเข้ารหัสและถอดรหัสข้อมูลแบบเอ็มเป็ค และวิเคราะห์ความเป็นไปได้ในการใช้ชิบดีเอสพี (DSP)ประมวลผล โดยไม่ต้องอาศัยการติดต่อกับอุปกรณ์ภายนอกของเครื่องคอมพิวเตอร์ ซึ่งชิบดีเอสพีนี้เป็นอุปกรณ์ที่มีประสิทธิภาพและประโยชน์สูงมากในอนาคต

1.2 วัตถุประสงค์ของปริญญานิพนธ์

1. ศึกษาหลักการการเข้ารหัสข้อมูลแบบเอ็มเป็กเลเยอร์ 3
2. เขียนโปรแกรมถอดรหัสข้อมูลเอ็มเป็กเลเยอร์ 3 ด้วยภาษาซี บนเครื่องคอมพิวเตอร์
3. ใช้ดีเอสพี (DSP) ชุด "TMS320C50 Starter Kit" ร่วมประมวลผลกับเครื่องคอมพิวเตอร์
4. ประเมินผลและหาความเป็นได้ที่จะใช้ ชิปดีเอสพี สร้างเครื่องเล่นเพลงจากไฟล์เอ็มเป็กเลเยอร์ 3 โดยไม่อาศัยเครื่องคอมพิวเตอร์

1.3 ขอบเขตของวิทยานิพนธ์

ทดลองให้บอร์ดดีเอสพีร่วมประมวลผลในการถอดรหัสข้อมูลเอ็มเป็กเลเยอร์3กับ เครื่องคอมพิวเตอร์ โดยเครื่องคอมพิวเตอร์จะส่งผ่านข้อมูลต่อให้กับดีเอสพี และประเมินความเป็นไปได้ ที่จะใช้ ชิบดีเอสพี สร้างเครื่องเล่นเพลงจากไฟล์เอ็มเป็กเลเยอร์ 3 โดยไม่อาศัยเครื่องคอมพิวเตอร์

1.4 แผนการดำเนินงาน

อธิบายแผนการดำเนินงานเป็นขั้นตอนดังต่อไปนี้

1. หาข้อมูลจากแหล่งต่างๆ เช่น อินเตอร์เน็ต (Internet) , สถาบันการศึกษา , ห้องสมุด และหนังสือต่างๆ เป็นต้น
2. ศึกษาการทำงานของการเข้ารหัส (Encode) และการถอดรหัส (Decode) ของเอ็มเป็กออดิโอ (Mpeg Audio)
3. ศึกษาโปรแกรม (Source Code) ในการถอดรหัสเอ็มเป็กเพื่อความเข้าใจมากขึ้น
4. ปรับปรุงและแก้ไข โปรแกรม เพื่อให้เหมาะกันการใช้งาน โดยในภาคเรียนที่ 1 จะใช้ภาษาซี สร้างแอพพรีเคชัน(Application)ใช้งานบนดอส(Dos)บนเครื่องคอมพิวเตอร์(PC)
5. ศึกษาการใช้งานบอร์ด "DSP" ชุด Starter Kit รุ่น TMS320C5x และการเชื่อมต่อกับคอมพิวเตอร์ (PC)
6. แปลงโปรแกรมภาษาซีเป็นภาษาแอสแซมบลี(Assamble)ของ DSP
7. วิเคราะห์และออกแบบให้ DSP สามารถช่วยถอดรหัสเอ็มเป็กเลเยอร์3 ร่วมกับเครื่องคอมพิวเตอร์
8. เขียนโปรแกรมเพื่อใช้ในการส่งต่อข้อมูลจากคอมพิวเตอร์สู่ DSP เพื่อถอดรหัสต่อ
9. สรุปและประเมินผลหาความเป็นได้ในการสร้างเครื่องเล่นเพลงจากไฟล์เอ็มเป็กเลเยอร์ 3

1.5 ประโยชน์และผลที่ได้รับจากการดำเนินงาน

1. ทราบถึงหลักการเข้ารหัสข้อมูลเสียงแบบเอ็มเป็กเลเยอร์ 3
2. ทราบถึงหลักการและเทคนิคการเขียนโปรแกรมภาษาซี
3. ทราบถึงการเชื่อมต่อระหว่างเครื่องคอมพิวเตอร์และอุปกรณ์ภายนอกต่างๆ
4. สามารถประยุกต์ใช้งานชิบดีเอสพีได้
5. สามารถนำผลการดำเนินการ ไปประยุกต์ใช้กับการสร้างเครื่องเล่นเพลงเอ็มเป็คเลเยอร์ 3 ได้ในอนาคต

บทที่ 2
การบีบอัดข้อมูล
(Data Compression)

2.1 ทำไมจึงต้องบีบอัดข้อมูล?

ในสมัยอดีตนั้นหน่วยความจำมีราคาแพงมาก ทำให้ต้องประหยัดเนื้อที่หน่วยความจำเป็นอย่างมาก จึงเกิดกระบวนการที่ทำให้ข้อมูลมีขนาดเล็กลง กระบวนการนั้นเรียกว่า **"การบีบอัด"** (Compression) การบีบอัดจะเป็นการลดอัตราการส่งข้อมูล(Bit Rate) และลดขนาดของข้อมูลลงแต่สามารถจะขยายกลับเป็นขนาดเดิม และมีรายละเอียดของข้อมูลครบถ้วนเหมือนตอนก่อนที่จะบีบอัด

ในปัจจุบันหน่วยความจำมีราคาถูกลงมาก แต่การบีบอัดกลับมีความสำคัญมากขึ้นกว่าในสมัยอดีต เพราะขนาดข้อมูลก็มีขนาดใหญ่ขึ้นกว่าแต่ก่อนมากเช่นเดียวกัน การที่จะเคลื่อนย้ายข้อมูลจะทำได้ลำบากขึ้นเพราะต้องใช้หน่วยความจำขนาดใหญ่ตามไปด้วย และในงานบางประเภทหากไม่ใช้การบีบอัดเข้าช่วยก็จะไม่สามารถใช้งานได้

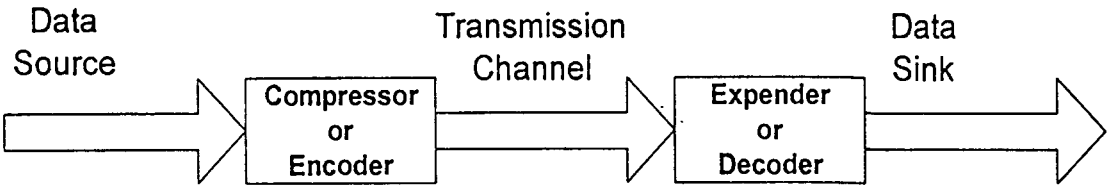
จะเห็นได้ว่า "การบีบอัด" (Compression) เป็นกระบวนการที่มีความสำคัญมาก ดังนั้นจึงสรุปได้ว่าเราใช้"การบีบอัด"เพื่อ

- ลดพื้นที่ในการเก็บข้อมูล ช่วยให้ประหยัดพื้นที่ในการเก็บข้อมูล
- ช่วยให้บางกระบวนการที่เคยทำไม่ได้ให้ทำได้

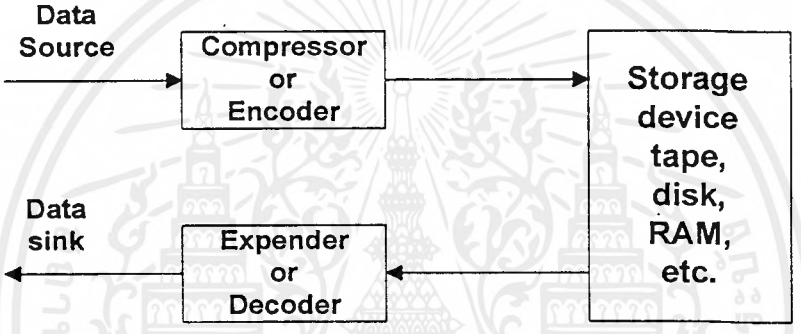
2.2 การประยุกต์การบีบอัด

การบีบอัดแสดงไว้ดังรูปที่ 2.1 แสดงการไหลของข้อมูลจากอินพุทไปที่เอาท์พุท จากรูปที่ 2.1(a) ปริมาณของข้อมูลถูกลดลงโดยตัวบีบอัด(Compressor)หรือโดยตัวเข้ารหัส(Encoder) จากนั้นข้อมูลที่ถูกบีบอัดแล้วจะผ่านไปตามช่องส่ง(Transmission Channel) และข้อมูลที่ถูกบีบอัดจะถูกขยายออกเป็นข้อมูลเดิมก่อนถูกเข้ารหัสโดยตัวขยาย(Expender)หรือตัวถอดรหัส (Decoder) แต่กรณีนี้มีข้อเสียคือ ข้อมูลจะถูกบีบอัดและขยายเลย ถ้าต้องการใช้งานอีกก็ต้องบีบอัดและทำการขยายใหม่ ดังนั้นจึงใช้หน่วยความจำ เช่น เทป(Tape) , ดิสก์(Disk) , แรม(RAM)เป็นต้น เก็บข้อมูลที่ถูกเข้ารหัสแล้วจากตัวเข้ารหัส ถ้าต้องการนำไปใช้งานก็จึงนำข้อมูลที่ถูกเข้ารหัสนั้นมาถอดรหัสแล้วจึงนำไปใช้งาน โดยที่ข้อมูลที่ถูกเข้ารหัสแล้วก็ยังคงถูกเก็บอยู่ในหน่วยความจำ ดังนั้นถ้า

ต้องการใช้งานอีกก็สามารถนำข้อมูลที่ถูกเข้ารหัสแล้วนี้ไปถอดรหัสได้เลย โดยไม่ต้องทำการเข้ารหัสอีกดังรูปที่ 2.1(b) แสดงตำแหน่งของหน่วยความจำ



(a)



(b)

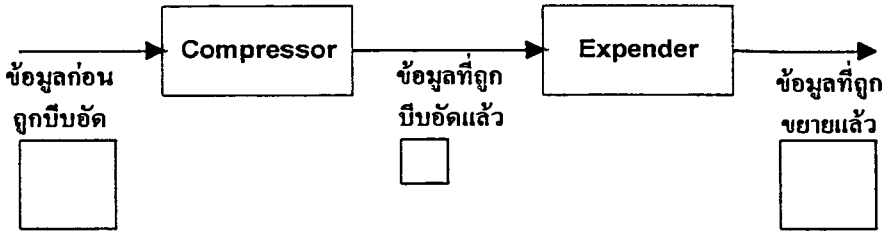
รูปที่ 2.1 แสดงการไหลของข้อมูลจากอินพุทไปที่เอาท์พุท

2.3 รูปแบบในการบีบอัด

การบีบอัดนั้นจะมีอยู่ด้วยกัน 2 ประเภท คือ

- แบบไม่มีการสูญเสีย(Lossless)
- แบบมีการสูญเสีย(Lossy)

2.3.1 แบบไม่มีการสูญเสีย(Lossless)



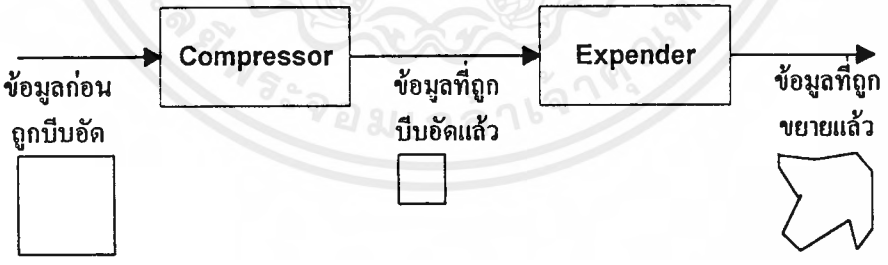
รูปที่ 2.2 แสดงข้อมูลก่อนการบีบอัดและข้อมูลหลังการขยาย

เป็นการบีบอัดที่ข้อมูลก่อนถูกบีบอัดกับข้อมูลเดียวกันที่ถูกขยายออกมา จะมีลักษณะเหมือนกันทุกประการ ไม่มีส่วนขาดส่วนเกิน ก็คือในกระบวนการบีบอัดจะไม่มีการตัดข้อมูลออกนั่นเอง ซึ่งแสดงได้ดังรูปที่ 2.2 แสดงข้อมูลก่อนการบีบอัดและข้อมูลหลังการขยาย

2.3.2 แบบมีการสูญเสีย(Lossy)

การบีบอัดแบบนี้ ข้อมูลจะถูกตัดออกบางส่วนในระหว่างการบีบอัด ซึ่งมีผลทำให้ข้อมูลหลังการขยายจะผิดเพี้ยนไปจากเดิม ดังรูปที่ 2.3 แสดงการบีบอัดแบบมีการสูญเสีย(Lossy) วิธีการนี้จะสามารถบีบอัดข้อมูลได้มากกว่าแบบแรก (แบบ Lossless) การบีบอัดแบบนี้มักจะใช้กับข้อมูลเสียงและข้อมูลภาพ

- สำหรับข้อมูลเสียงเราจะสนใจเพียงเสียงที่ได้ยินเท่านั้น โดยไม่สนใจว่าข้อมูลจะถูกตัดออกไปหรือไม่ ก็คือข้อมูลเสียงจะถูกตัดเสียงที่มนุษย์ไม่ได้ยินเมื่อเทียบกับเสียงอื่นออกและบีบอัดข้อมูล เมื่อทำการขยายออกมาข้อมูลที่ได้จะผิดเพี้ยนจากเดิม แต่คุณภาพเสียงเหมือนเดิม(แยกไม่ออกว่าเสียงไหนเป็นเสียงจริง เสียงไหนเป็นเสียงที่ถูกตัดบางส่วนออกแล้ว)
- สำหรับข้อมูลภาพเราจะสนใจเพียงภาพที่มองเห็นเท่านั้น โดยไม่สนใจว่าข้อมูลจะถูกตัดออกไปหรือไม่ ก็คือข้อมูลภาพจะตัดสีที่เหลื่อมกันเล็กน้อยจนมนุษย์ไม่สามารถมองเห็นได้ ออกจากข้อมูลเดิมและบีบอัดข้อมูล เมื่อขยายออกมาก็จะภาพคุณภาพไม่ต่างจากภาพจริงเลย(มนุษย์แยกไม่ออก)



รูปที่ 2.3 แสดงการบีบอัดแบบมีการสูญเสีย(Lossy)

บทที่ 3
การบีบอัดข้อมูลเสียงแบบเอ็มเป็ก
(MPEG Audio Compression)

3.1 บทนำ

การบีบอัดข้อมูลเสียงแบบเอ็มเป็ก (Mpeg Audio Compression) เป็นวิธีการบีบอัดข้อมูลเสียงแบบดิจิตอล (Digital Compression) ที่มีความเหมือนจริงสูง (High Fidelity:HiFi) การบีบอัดข้อมูลแบบเอ็มเป็ก สำเร็จลงในปี 1993 โดยคณะกรรมการสากลแห่งการเชี่ยวชาญการบีบอัดเสียงเหมือนจริง ซึ่งรู้จักกันดีในชื่อว่า Motion Picture Expert Group (MPEG)และได้รับมาตรฐาน ISO/IEC 11172-3

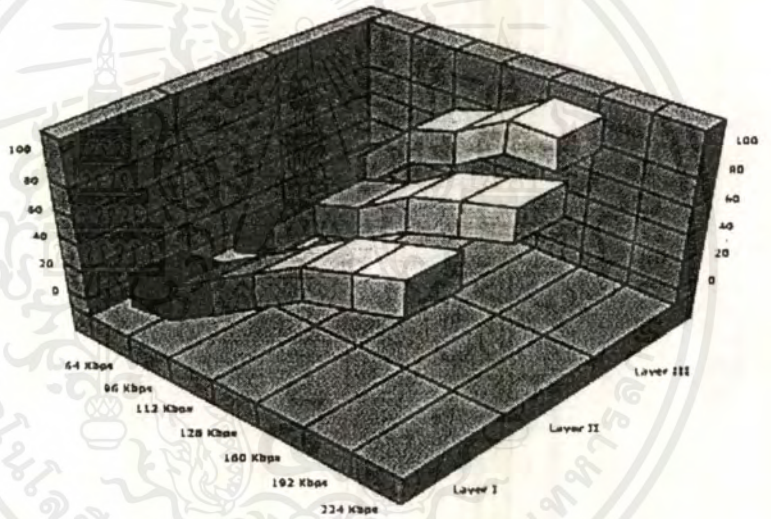
3.2 หลักการพื้นฐานละการใช้งาน

หลักการบีบอัดข้อมูลแบบเอ็มเป็ก จะใช้ประโยชน์จากขีดจำกัดในการได้ยินของมนุษย์โดยไม่จัดเก็บข้อมูลเสียงที่ที่มนุษย์ฟังไม่ได้ยิน เพื่อให้ข้อมูลมีขนาดเล็กลง ซึ่งใช้หลักการว่า ถ้าเราได้ยินเสียง 2 เสียงที่มีความถี่ใกล้เคียงกัน โดยให้เสียงที่ 2 ดังกว่าเสียงที่ 1 มาก เมื่อเราฟัง 2 เสียงนี้พร้อมกันจะได้ยินเสียงที่ 2 (ดังกว่า) เพียงเสียงเดียว ดังนั้นในการจัดเก็บข้อมูลสามารถตัดข้อมูลเสียงที่ค่อยกว่าออกไปได้บางส่วนซึ่งหลักการนี้จะถูกอธิบายในหัวข้อ"ไซโครอะคูสติก" (Phychoacoustic) อีกครั้ง การบีบอัดข้อมูลแบบเอ็มเป็กสามารถเลือกวิธีการบีบอัดเสียงได้หลายๆ แบบ ได้ดังนี้

- อัตราการสุ่มข้อมูล (Sampling rate) 32 , 44.1 หรือ 48 กิโลเฮิร์ต (kHz)
- รองรับระบบเสียงได้ทั้ง 1 และ 2 ช่องเสียง. ซึ่งมีอยู่ 4 แบบดังนี้
    1. แบบ โมโนเดี่ยว (Monophonic mode)
    2. แบบ โมโนคู่ (Dual Monophonic mode)
    3. แบบ สเตอริโอ (Stereo mode)
    4. แบบ จอย- สเตอริโอ (Joint Stereo mode)
- สามารถเลือกค่าอัตราการส่งข้อมูล (bit reate) ได้ตั้งแต่ 32 ถึง 224 Kbit/sec ต่อ หนึ่งช่องเสียง ขึ้นอยู่กับอัตราการสุ่มตัวอย่าง (Sampling rate)
- รูปแบบของการเข้ารหัสไฟล์เอ็มเป็ต มี 3 เลเยอร์ (Layer) คือ เลเยอร์ 1 เลเยอร์ 2 และเลเยอร์ 3 แต่ละเลเยอร์มีลักษณะต่างกันดังนี้

- เลเยอร์ 1 มีความซับซ้อนน้อยที่สุด ต้องใช้อัตราการส่งข้อมูลสูงถึง 384 กิโลบิตต่อวินาที (Kbps) เพื่อที่จะให้ได้เสียงคุณภาพสูง
- เลเยอร์ 2 มีความซับซ้อนมากขึ้น คุณภาพเสียงสูงกว่าเลเยอร์ 1 ใช้อัตราการส่งข้อมูลที่ 160 กิโลบิตต่อวินาทีจะได้เสียงคุณภาพสูง และที่ 192 กิโลบิตต่อวินาทีจะได้คุณภาพเสียงไม่แตกต่างจากเสียงต้นแบบ
- เลเยอร์ 3 มีความซับซ้อนที่สุดแต่สามารถให้คุณภาพเสียงที่ดีที่สุด ใช้อัตราการส่งข้อมูลที่ 160 กิโลบิตต่อวินาที ให้คุณภาพเสียงไม่แตกต่างเสียงต้นแบบ

สามารถแสดงความสัมพันธ์ของอัตราการส่งข้อมูลเสียง(กิโลบิตต่อวินาที) สำหรับการเข้ารหัสเลเยอร์ต่างๆ กับคุณภาพเสียงเมื่อเทียบกับเสียงต้นแบบได้ดังรูปที่ 3.1



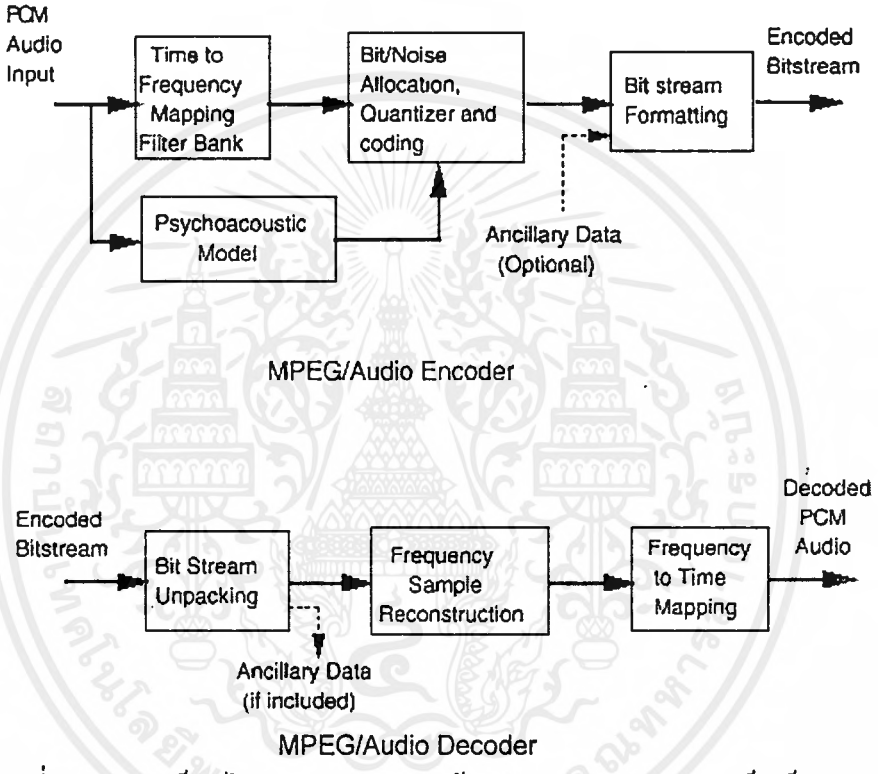
รูปที่ 3.1 กราฟเปรียบเทียบอัตราส่วนการบีบอัดเสียงของการเข้ารหัสเลเยอร์ต่างๆ เมื่อเทียบกับคุณภาพเสียงต้นแบบจากคอมแพ็คดิสก์(CD)

3.3 การเข้ารหัสแบบเอ็มเป็กโดยทั่วไป

การเข้ารหัสแบบเอ็มเป็กโดยทั่วไป จะตัดข้อมูลเสียงที่จัดเก็บบางส่วนออกแต่สามารถคงรายละเอียดของเสียงที่ได้ยินไว้เท่าเดิม ดังรูปที่ 3.2 แสดงบล็อกไดอะแกรม (Block Diagram) ของการเข้ารหัสและถอดรหัสแบบเอ็มเป็ก กระบวนการเข้ารหัสมีขั้นตอนดังนี้

1. เริ่มจากข้อมูลเสียงจะถูกส่งไปยังโพลีเฟส ฟิลเตอร์แบงค์ (The Polyphase Filter bank) เพื่อแบ่งข้อมูลเสียงออกเป็นหลายๆช่วงความถี่ (Subband of Frequency)

2. ขณะเดียวกัน ข้อมูลเสียงก็จะผ่านไปยัง ไซโครอะคูสติก โมเดล(Psychoacoustic Model) ด้วยเพื่อหา สัดส่วนของพลังงานของสัญญาณต่อการกำหนดค่ามาสก์กิงเทรดโฮล (Masking threshold : SMR) ของแต่ละช่วงความถี่
3. ข้อมูลจากโพลีเฟส ฟิลเตอร์แบงค์ ถูกส่งไปยังส่วนแบ่งแยกบิต (Bit/Noise Allocation) และปรับข้อมูล (Quantizing)



รูปที่ 3.2แสดงบล็อคไดอะแกรมของการเข้ารหัสและถอดรหัสแบบเอ็มเป็ก

กระบวนการถอดรหัสบิตสตรีม (Bitstream) จะนำข้อมูลมาผ่านกระบวนการปรับข้อมูลย้อนกลับ และทำการรวมข้อมูลแต่ละช่วงความถี่กลับเป็นข้อมูลเสียง

3.3.1 โพลีเฟส ฟิลเตอร์แบงค์ (The Polyphase Filter bank)

โพลีเฟส ฟิลเตอร์แบงค์ (The Polyphase Filter bank) เป็นหลักการทั่วไปในการเข้ารหัสแบบเอ็มเป็ก โพลีเฟส ฟิลเตอร์แบงค์ จะแบ่งสัญญาณเสียงออกเป็นช่วงความถี่ที่มีความกว้าง (bandwidth) ออกเป็น 32 ช่วงความถี่ย่อย (Subband) ที่เท่ากัน

คุณสมบัติของโพลีเฟส ฟิลเตอร์แบงค์

1. ความกว้างของช่วงความถี่ในแต่ละช่วงเท่ากัน
2. ฟิลเตอร์แบงค์ และการแปลงกลับ เป็นการแปลงแบบมีการสูญเสีย(lossy)

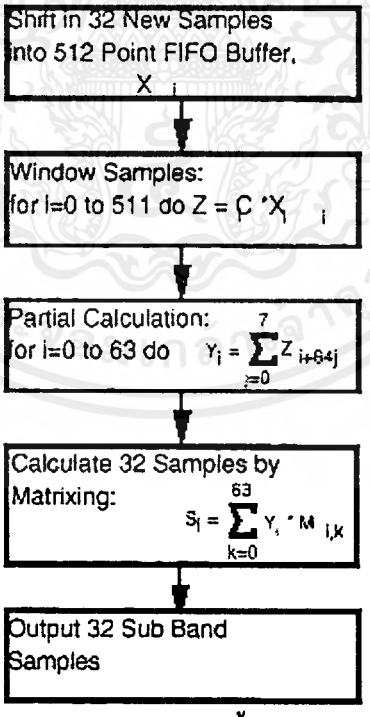
3. ย่านความถี่แต่ละย่านจะมีการเหลื่อมซึ่งกันและกัน

ขั้นตอนการทำโพลีเฟส ฟิลเตอร์แบงค์

โพลีเฟส ฟิลเตอร์แบงค์ เป็นกระบวนการในการแยกสัญญาณที่ถูกสุ่มตัวอย่าง(Sampling) ด้วยความถี่สุ่มตัวอย่าง (F_s) ออกเป็น 32 ย่านความถี่ย่อย โดยแต่ละช่วงของย่านความถี่ย่อยเหล่านี้จะมีความถี่เท่ากันคือ $F_s/32$ ขั้นตอนการวิเคราะห์ มีดังนี้

1. ข้อมูลเสียงเข้า 32 ข้อมูลสุ่มตัวอย่าง(Sample)
2. สร้างข้อมูลบัฟเฟอร์ (Buffer) ไว้สำหรับข้อมูลสุ่มตัวอย่าง 512 ข้อมูล โดยจะเลื่อนข้อมูลเข้า 32 ข้อมูล เข้ามาในบัฟเฟอร์ ในตำแหน่งที่ 0-31
3. คูณค่าข้อมูลอินพุท 512 ค่า กับ สัมประสิทธิ์ (C) 512 ค่า
4. คำนวณค่า Y ตามสมการที่ให้มา (64 ค่า)
5. คำนวณค่าสับแบนด์แซมเปิล(Subband Sample:S) 32 ค่าออกมา

อธิบายขั้นตอนการทำโพลีเฟส ฟิลเตอร์แบงค์ เป็น โฟว์ชารจ์ (Flowchart) ได้ดังรูปที่ 3.3 แสดงโฟลว์ชารจ์ (Flowchart) อธิบายขั้นตอนการทำโพลีเฟส ฟิลเตอร์แบงค์



รูปที่ 3.3 แสดงโฟลว์ชารจ์ (Flowchart) อธิบายขั้นตอนการทำโพลีเฟส ฟิลเตอร์แบงค์

$$S_t[i] = \sum_{k=0}^{63} \sum_{j=0}^{7} M[i][k] \times (C[k+64j] \times x[k+64j]) \quad \text{.....สมการที่ 3.1}$$

โดย

i คือ ดรรชนีย่านความถี่ ตั้งแต่ 0-31

s_t คือ สัญญาณเอาท์พุทที่ถูกกรองความถี่แล้ว สำหรับย่านความถี่ I ที่เวลา t โดย t เป็นเลขจำนวนเต็ม

C[n] คือ สัมประสิทธิ์ของวินโดว์ ลำดับที่ n ถูกนิยามไว้ในมาตรฐาน

X[n] คือ สัญญาณเสียงอินพุทที่ถูกอ่านจากอินพุทบัฟเฟอร์ที่เก็บอินพุทไว้ 512 สัญญาณความถี่

$$M[i][k] = \cos\frac{(2\times i+1)\times(k-16)\times\pi}{64}$$

สมการที่ 3.1 เป็นการคำนวณที่สามารถลดจำนวนครั้งของการคูณได้มากที่สุด เนื่องจากพจน์ที่อยู่ในวงเล็บไม่เป็นฟังก์ชันของ I และ M[I][k] ไม่ขึ้นกับ j ดังนั้นการคำนวณทุกๆ 32 สัญญาณสุ่มตัวอย่างเอาท์พุท จะคูณเลขเพียง 512 + 32×64 = 2560 ครั้ง และการบวก 64×7+32×63 = 2464 ครั้ง หรือประมาณ 80 ครั้งต่อสัญญาณเอาท์พุท 1 สัญญาณ

ข้อสังเกตจากการทำฟิลเตอร์แบงค์ ทุกๆ 32 สัญญาณอินพุทเราจะได้สัญญาณเอาท์พุท 32 สัญญาณ ด้วยเหตุนี้ทั้ง ในแต่ละย่านความถี่ย่อย(Subband) จึงถูกแบ่งให้มีเพียงเอาท์พุทเดียว ทั้ง 32 ย่าน เมื่ออินพุทเข้ามา 32 สัญญาณ

จากสมการที่3.1สามารถเขียนเป็นสมการพื้นฐานในรูปแบบผลบวกประสาน(Convolution) ได้คือ

$$s_t[i] = \sum_{n=0}^{511} x[t-n]\times H_i[n] \qquad \text{.....สมการที่ 3.2}$$

$$H_i[n] = h[n]\times\cos\left[\frac{(2\times i+1)\times(n-16)\times\pi}{64}\right] \qquad \text{.....สมการที่ 3.3}$$

โดย

x[τ] คือ สัญญาณเสียงสุ่มความถี่

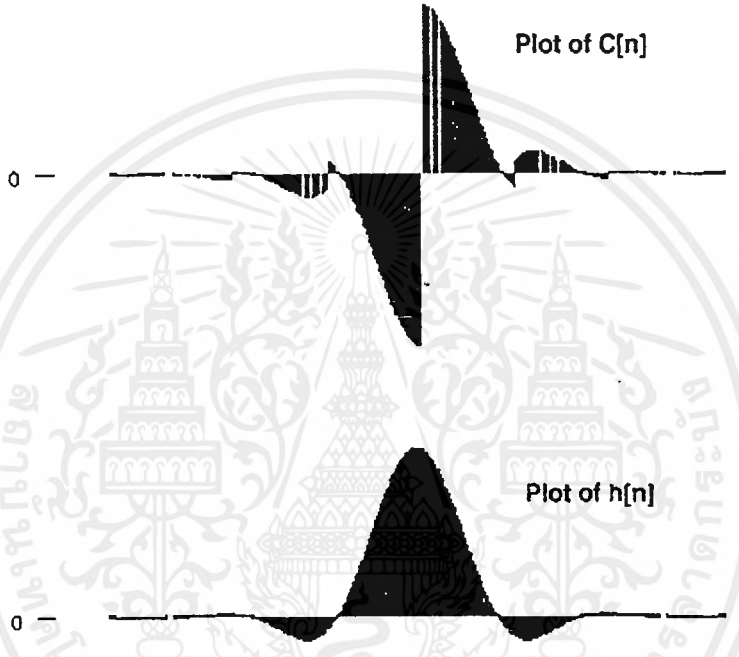
h[n] = -C[n] เมื่อ n/64 เป็นเลขคี่

= C[n] กรณีอื่น

ในรูปแบบนี้แต่ละย่านความถี่ย่อยจะมีผลตอบสนองความถี่ของตนเองคือ $H_i[n]$ แม้ว่ารูปแบบนี้จะง่ายต่อการวิเคราะห์แต่จะมีจำนวนครั้งของการคูณและบวกในการคำนวณมากกว่าสมการที่ 3.1 คือจะคูณเลข 512×32 = 16384 ครั้ง และการบวก 512×32 = 16384 ครั้ง เพื่อที่คำนวณสัญญาณเอาท์พุท 32 สัญญาณ

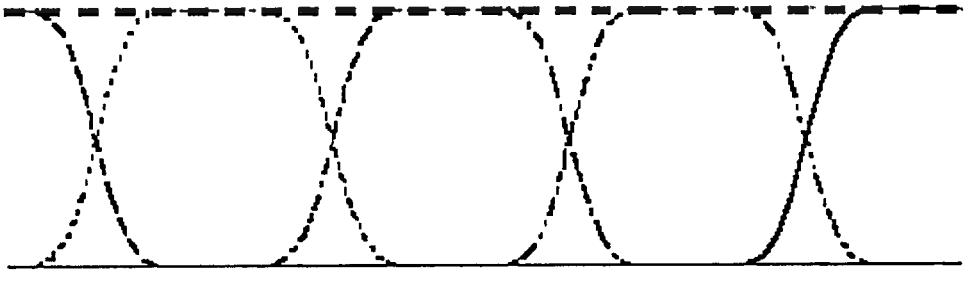
ค่าสัมประสิทธ์ h[n] เป็นผลตอบสนองตัวกรองความถี่ต่ำสำหรับโพลีเฟส ฟิลเตอร์แบงค์ รูปที่ 3.4 แสดงสัมประสิทธ์ C[n] และ h[n]

สมการสำหรับ H[n] แสดงให้เห็นว่า ค่า H[n] ได้มาจาก h[n] คูณกับเทอมโคไซน์(Cosine) เพื่อที่จะเลื่อนเฟสของผลตอบสนองตัวกรองความถี่ต่ำ (h[n]) ให้เหมาะสมกับย่านความถี่ ดังนั้นจึงเรียกการกรองความถี่แบบนี้ว่า "โพลีเฟส " (Polyphase) การกรองความถี่แบบนี้จะมีความถี่กลางอยู่ที่ค่าเลขคี่คูณด้วย $\pi/(64T)$ ซึ่ง T เป็นคาบของการสุ่มตัวอย่าง (Sampling period)



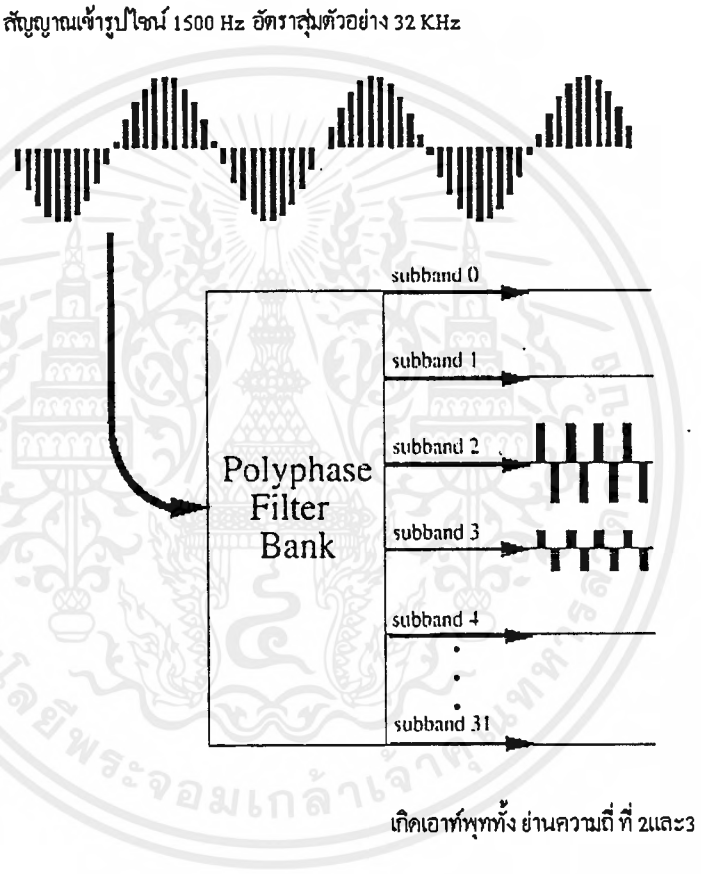
รูปที่ 3.4 แสดงการเปรียบเทียบสัมประสิทธ์ C[n] และ h[n]

เนื่องจากผลตอบสนองตัวกรองความถี่ต่ำ ไม่มีความคมในการัทออฟ (แถบนำกว้าง) ดังนั้น เมื่อแบ่งช่วงความถี่ที่ใช้งานทั้งหมดเป็น 32 ช่วง จะเกิดการเหลื่อมซึ่งกันและกัน ดังแสดงตัวอย่างในรูปที่ 3.5



รูปที่ 3.5 แสดงการเหลื่อมซึ่งกันและกันระหว่างย่านความถี่ย่อยที่ติดกัน

ซึ่งผลของการเหลื่อมกันของกันของย่านความถี่ที่ติดกัน จะทำให้ประสิทธิภาพของการบีบอัดข้อมูลลดลงเนื่องจากพลังงานของสัญญาณที่มีความถี่ใกล้ๆกันขอบของย่านความถี่หนึ่งๆ จะปรากฏเป็นเอาท์พุทของโพลีเฟส ฟิลเตอร์แบงค์ สองเอาท์พุทที่อยู่ติดกัน ดังรูปที่ 3.6 แสดงตัวอย่างของสัญญาณรูปไซน์หนึ่งความถี่ เมื่อผ่านกระบวนการโพลีเฟส ฟิลเตอร์แบงค์ แล้วปรากฏเอาท์พุทออกมาสองย่านความถี่



รูปที่ 3.6 แสดงการเกิดเอาท์พุทของโพลีเฟส ฟิลเตอร์แบงค์เมื่อสัญญาณเข้าเป็นสัญญาณไซน์หนึ่งความถี่

3.3.2 ไซโครอะคูสติก โมเดล (Psychoacoustics Model)

การเข้ารหัสข้อมูลเสียงแบบเอ็มเป็ก ใช้หลักการตัดเสียงบางส่วนที่ฟังไม่ได้ยินออก เนื่องจากความได้เปรียบในการรับฟังเสียงของมนุษย์ที่ไม่สามารถได้ยินเสียงรบกวน(noise)ภายใต้เงื่อนไขการปิดกั้นการได้ยิน (Auditory masking) การปิดกั้นการได้ยินเป็นคุณสมบัติอย่างหนึ่งในการรับฟังเสียงของมนุษย์ เกิดขึ้นเมื่อเรารับฟังเสียงที่จากแหล่งกำเนิดสองแหล่ง จะไม่สามารถได้ยินเสียงจากแหล่งกำเนิดที่ค่อยกว่า ถ้าแหล่งกำเนิดทั้งสองมีความถี่ใกล้เคียงกัน

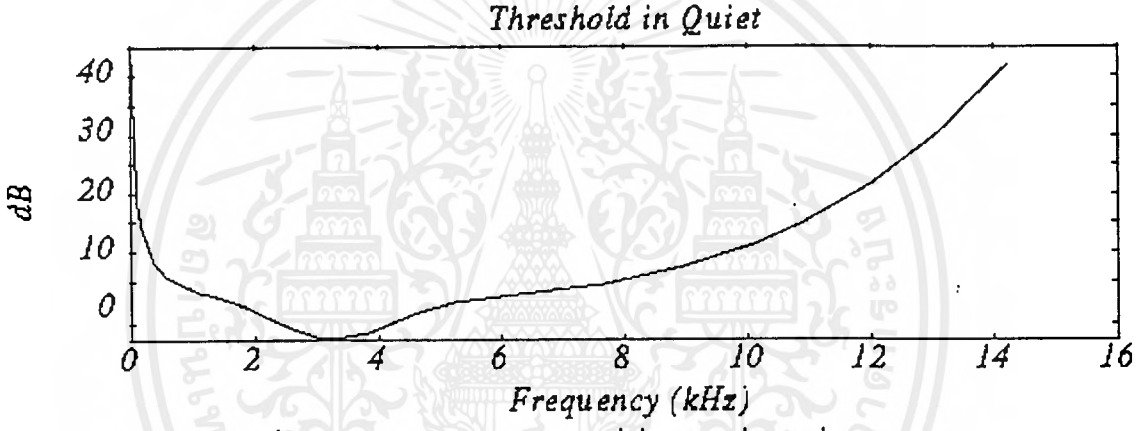
ความสามารถใน การได้ยินเสียงของมนุษย์

1 ช่วงความถี่ที่สามารถรับฟัง ประมาณ 20-20000 เฮิร์ต

2 ไดนามิคเรนจ์(dynamic range) คือเสียงค่อยสุดถึงดังสุด ที่สามารถรับฟังได้ ประมาณ 96 เดซิเบล

3. ความถี่เสียงพูดปกติ 500-2000 เฮิร์ต

4. ความไวในการรับฟังเสียงของมนุษย์ จากการทดลองให้คน 1 คนเข้าไปนั่งในห้องที่เงียบสนิท แล้วเพิ่มความค่อยๆดังเสียง จนกระทั่งคนๆนั้นเริ่มได้ยิน (Threshold level) วัดระดับความดังเสียง แปรความถี่แล้วนำค่ามาวาดกราฟ แสดงได้ดังรูปที่ 3.7 พบว่ามนุษย์สามารถรับเสียงช่วงความถี่ 2-4 กิโลเฮิร์ตได้ไวที่สุด

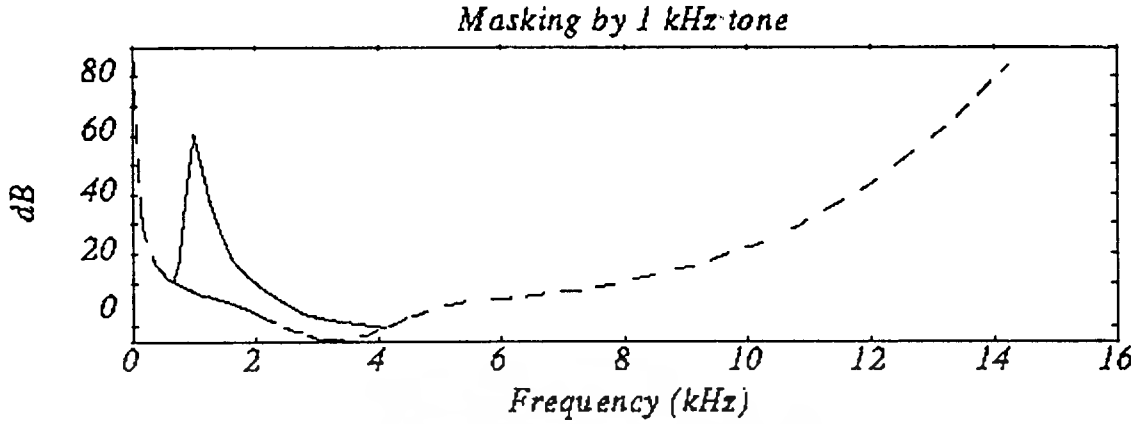


รูปที่ 3.7 แสดงระดับความดังเสียงที่เริ่มได้ยินที่ความถี่ต่างๆ

5. การปิดกั้นเสียงเมื่อมีสัญญาณเสียงหลายๆสัญญาณที่มีความถี่ใกล้เคียงกันเข้ามา มนุษย์สามารถได้ยินสัญญาณที่มีความดัง(แรง)กว่าเท่านั้นหรือกล่าวได้ว่าสัญญาณที่แรงกว่าปิดกั้นสัญญาณ ที่อ่อนกว่า และเรียกสัญญาณที่แรงกว่านั้นว่า ตัวปิดกั้น(Masker)

จากการทดลองเรื่องการปิดกั้นเสียง โดยใช้ให้คนฟังเสียง ในห้องที่มีเสียงที่ความถี่ 1 กิโลเฮิรต์ ความดัง 60 เดซิเบล แล้วเล่นเสียงจากแหล่งกำเนิดอีกแหล่ง วัดระดับความดังเสียงที่ได้ยินจากแหล่งกำเนิดที่สอง แปรความถี่ วาดกราฟ ได้ดังรูปที่ 3.8 ซึ่งแสดง ให้เห็นว่าที่ความถี่ใกล้กับความถี่ 1 กิโลเฮิร์ต ระดับความดังเสียงของแหล่งกำเนิดที่สองต้องมีค่าสูง เพื่อให้รับฟังเสียงได้

สามารถสรุปจากข้อมูลทั้งหมดได้ว่า มนุษย์สามารถรับฟังเสียงเป็นช่วงความถี่ และรับพลังงานเสียงได้ไม่เท่ากันในแต่ละย่านความถี่ ซึ่งเราเรียกว่าย่านความถี่วิกฤติ(critical band) และในแต่ละย่านความถี่มนุษย์จะแยกสัญญาณจากแหล่งกำเนิดต่างๆ ออกจากกันได้ยาก ตารางที่ 3.1 แสดงความกว้างของย่านความถี่วิกฤติย่านต่างๆ

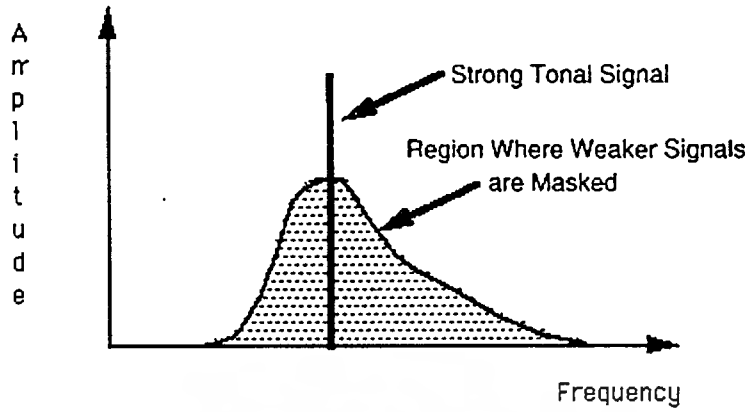


รูปที่ 3.8 แสดงผลของการปิดกั้นเสียง

ตารางที่ 3.1 แสดงความกว้างของย่านความถี่วิกฤติย่านต่างๆ

Band Number	Frequency (Hz)[1]	Band Number	Frequency (Hz)[1]
0	50	14	1,970
1	95	15	2,340
2	140	16	2,720
3	235	17	3,280
4	330	18	3,840
5	420	19	4,690
6	560	20	5,440
7	660	21	6,375
8	800	22	7,690
9	940	23	9,375
10	1,125	24	11,625
11	1,265	25	15,375
12	1,500	26	20,250
13	1,735		

เนื่องด้วยเหตุผลที่กล่าวมาทั้งหมดข้างต้น สามารถปิดกั้นเสียงที่ไม่ได้ยิน(NoiseMasking)ภายในย่านความถี่หนึ่งๆได้ดังรูปที่ 3.9 หลักการนี้การเข้ารหัสแบบเอ็มเป็กยึดถือเป็นสำคัญ โดยแบ่งสัญญาณเสียงออกเป็นย่าน ความถี่ย่อยประมาณเท่ากับย่านความถี่วิกฤติแล้วจึงปรับข้อมูล (quantize) ในแต่ละย่านความถี่ตามความสามารถในการได้ยินในแต่ละย่านความถี่



รูปที่ 3.9 แสดงการปิดกั้นเสียงที่ไม่ได้ยินในย่านความถี่ย่อยหนึ่งๆของการเข้ารหัสเอ็มเป็ก

ไซโครอะครูสติก โมเดล(Phychoacoustic Model)

ไซโครอะครูสติก โมเดล(Phychoacoustic Model) ทำหน้าที่วิเคราะห์หาจุด(ความถี่)ที่ต้องปิดกั้นในย่านความถี่หนึ่ง ตัวเข้ารหัสจะใช้ข้อมูลนี้ในการในการตัดสินใจว่า ควรแทนสัญญาณอินพุทที่เข้ามาด้วยจำนวนบิตมากน้อยเพียงใด

3.3.3 การแบ่งแยกบิต (Bit or Noise allocation)

การแบ่งแยกบิตทำหน้าที่หาจำนวนของบิตเพื่อที่จะแยกในแต่ละย่านความถี่ย่อยขึ้นอยู่กับข้อมูลที่ได้มาจากไซโครอะครูสติก โมเดล

การแบ่งแยกบิตของการเข้ารหัสเอ็มเป็กเลเยอร์ 3 มีรายละเอียดดังนี้

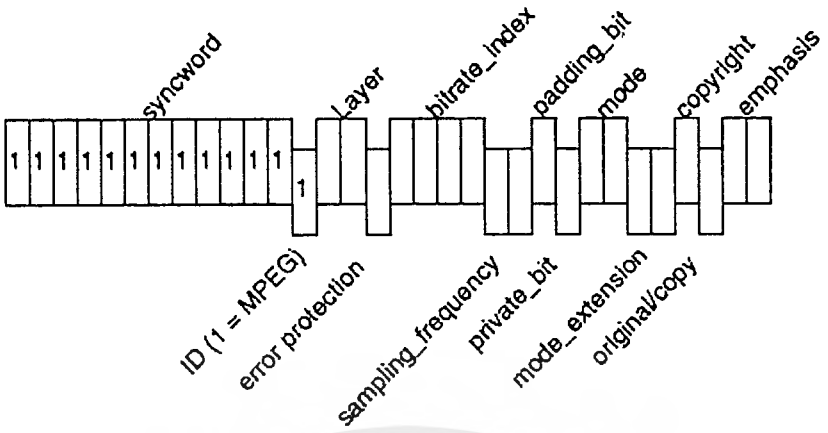
1. ปรับข้อมูลให้เป็นค่า (quantize)
2. นับจำนวนบิตของการเข้ารหัสแบบฮัฟแมน (Huffman Coding)
3. คำนวณผลของเสียงที่ตัดออกจริงๆ (Actual calculates the resulting noise)
4. ทำขั้นตอนที่ 1-3 ซ้ำ เพื่อปรับค่าให้อยู่ในระดับที่ยอมรับได้ (Iterative Loop)

3.3.4 การเข้ารูปแบบ (Formatting)

เป็นขั้นตอนการจัดรูปแบบข้อมูลให้ตรงตามมาตรฐาน ซึ่งมาตรฐานการจัดเรียงข้อมูลแบบเอ็มเป็ก ของเลเยอร์ 3 แสดงได้ดังรูปที่ 3.10

หัวข้อมูล (Header)	ตรวจสอบความผิดพลาด (CRC)	ข้อมูลข้างเคียง (Side Information)	ข้อมูลหลัก (Main Data)

รูปที่ 3.10 แสดงรูปแบบข้อมูลเอ็มเป็กเลเยอร์ 3



รูปที่ 3.11 แสดงบิตต่างๆในส่วนหัวข้อมูล

1. ส่วนหัวข้อมูล (Header) เป็นข้อมูลขนาด 32 บิตแสดงลักษณะทั่วไปของไฟล์นั้นๆข้อมูลในส่วนนี้ประกอบด้วยบิตต่างๆดังรูปที่ 3.11
2. ส่วนตรวจสอบความผิดพลาด เป็นข้อมูลขนาด 16 บิต ตวรจสอบพาริตี้ ของข้อมูลที่ถูกเข้ารหัส
3. ส่วนข้อมูลข้างเคียง เป็นข้อมูลขนาด 17 หรือ 32 ไบต์ (17 ไบต์ สำหรับการเข้ารหัสแบบโมโนเดี่ยว (Monophonic mode) ,32 ไบต์สำหรับแบบอื่นๆ) ข้อมูลในส่วนนี้เก็บองค์ประกอบต่างๆที่ใช้ในการถอดรหัสโดยตรง
4. ข้อมูลหลัก จะไม่ถูกจำกัดความยาวข้อมูล ขึ้นอยู่กับความถี่สุ่มตัวอย่าง และอัตราการส่งข้อมูล ดังสมการ

$$N = 144 \times \frac{\text{bitrate}}{\text{sampling\_frequency}} \qquad \text{.....สมการที่ 3.4}$$

เมื่อ N คือ ความยาวข้อมูลระหว่างสอง syncword ที่อยู่ติดกัน

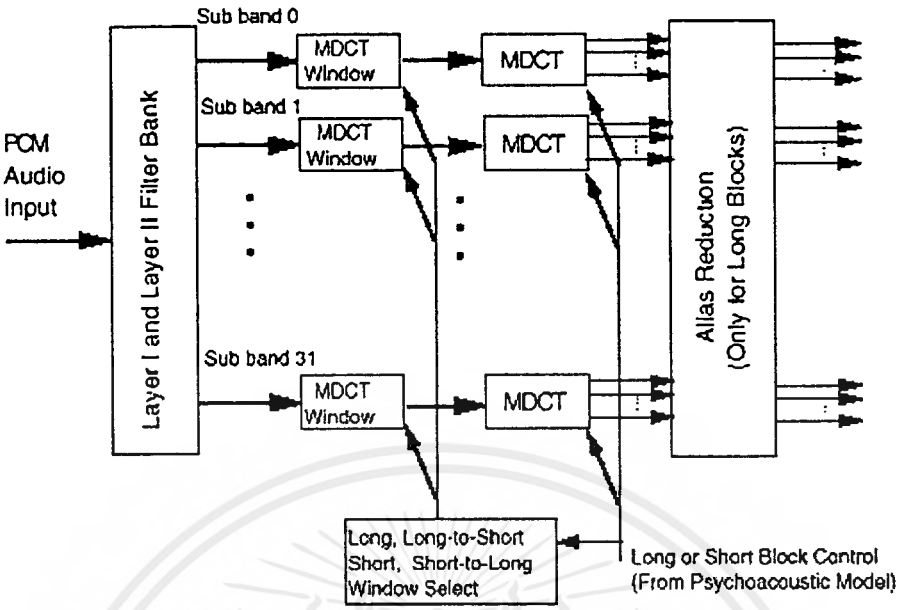
bitrate คือ อัตราการส่งข้อมูล

sampling_frequency คือ ความถี่สุ่มตัวอย่าง

3.4 การเข้ารหัสข้อมูลแบบเอ็มเป็กเลเยอร์ 3

การเข้ารหัสข้อมูลแบบเอ็มเป็กเลเยอร์ 3 มีหลักการเหมือนการเข้ารหัสแบบเอ็มเป็กโดยทั่วไปดังที่กล่าวมาในหัวข้อ 3.3 มีเพียงบางเทคนิคที่เพิ่มเติมเข้ามาเพื่อให้การบีบข้อมูลมีประสิทธิภาพสูงขึ้นดังนี้

1. ใช้การแปลงแบบ MDCT (Modified Discrete Cosine Tranfrom) เพื่อเพิ่มความละเอียดของสเปคตรัม หลังจากที่ผ่านขั้นตอนการทำ โพลีเฟส ฟิลเตอร์แบงค์ แสดงบล็อคไดอะแกรมของการทำ MDCT ดังรูปที่ 3.12



รูปที่ 3.12 แสดงบล็อคไดอะแกรมของการทำ MDCT

มีรูปแบบการทำ MDCT อยู่สองลักษณะ โดยแบ่งตามความยาวของบล็อคคือ

- บล็อคยาว มี 18 สัญญาณสุ่มตัวอย่าง แต่มีการเหลื่อมของย่านความถี่อยู่ ร้อยละ 50 ดังนั้น บล็อคยาว จึงมีความยาว 36 สัญญาณสุ่มตัวอย่าง
- บล็อคสั้น มี 6 สัญญาณสุ่มตัวอย่าง แต่มีการเหลื่อมของย่านความถี่อยู่ ร้อยละ 50 ดังนั้น บล็อคสั้นจึงมีความยาว 12 สัญญาณสุ่มตัวอย่าง

ความแตกต่างของบล็อคยาวกับบล็อคสั้นคือ บล็อคยาวให้ความละเอียดในด้านความถี่มากส่วน บล็อคสั้นให้ความละเอียดด้านเวลาสำหรับสัญญาณทรานเซียนส์ (Transient) สังเกตว่าความยาวของบล็อคสั้นเท่ากับหนึ่งในสามของบล็อคยาว ดังนั้นถ้าใช้บล็อคสั้นต้องใช้สามบล็อคเพื่อให้ความยาวข้อมูลเท่ากับเมื่อใช้บล็อคยาว

ในการเข้ารหัสเฟรม (หนึ่งเฟรมคือ 1152 สัญญาณสุ่มตัวอย่างอินพุท) หนึ่งๆสามารถใช้บล็อคชนิดเดียวกัน หรือบล็อคผสม(Mix block) ก็ได้ ถ้าเข้ารหัสแบบผสม 2 ย่านความถี่ต่ำสุดจะต้องใช้บล็อคยาว อีก 30 ย่านความถี่ที่เหลือเป็นบล็อคสั้น ซึ่งการเข้ารหัสแบบบล็อคผสมจะให้ความละเอียดด้านความถี่สูงที่ความถี่ต่ำ และความละเอียดด้านเวลาสูงที่ความถี่สูง

2. การลดผลจากการเหลื่อมของย่านความถี่ที่ติดกัน(Alias reduction)
3. การปรับค่าแบบไม่เป็นรูปแบบ (non-uniform quantization)
4. ย่านสเกลแฟคเตอร์ (scalefactor band) ใช้สเกลแฟคเตอร์ต่างกันเมื่อต่างย่านความถี่
5. การเข้ารหัสข้อมูลแบบเอ็นทรอปี้ เอ็มเป็กเลเยอร์ 3 ใช้การเข้ารหัสข้อมูลแบบฮัฟแมน (Huffman) มาช่วยในการบีบอัดข้อมูล

บทที่4

โครงสร้างของ TMS320C50

"TMS320C50" เป็นชิบโปรเซสเซอร์ตระกูล "TMS320C5x" ซึ่ง TMS320C5x นี้ก็จัดอยู่ในตระกูล "TMS320" อีกทีหนึ่ง TMS320 ใช้ประมวลผลทางด้านสัญญาณเชิงเลข (Digital Signal Processing :DSP) ตระกูล 'C5x ('C5x = TMS320C5x)นี้ใช้ประมวลผลทางด้านสัญญาณเชิงเลขแบบจำนวนเต็ม (Fix-Point Digital Signal Processing) ซึ่งเป็นการนำเอาสถาปัตยกรรมของ 'C25 มาปรับปรุงให้ดีขึ้น ทั้งในด้านความเร็วและความสะดวกอื่นๆ

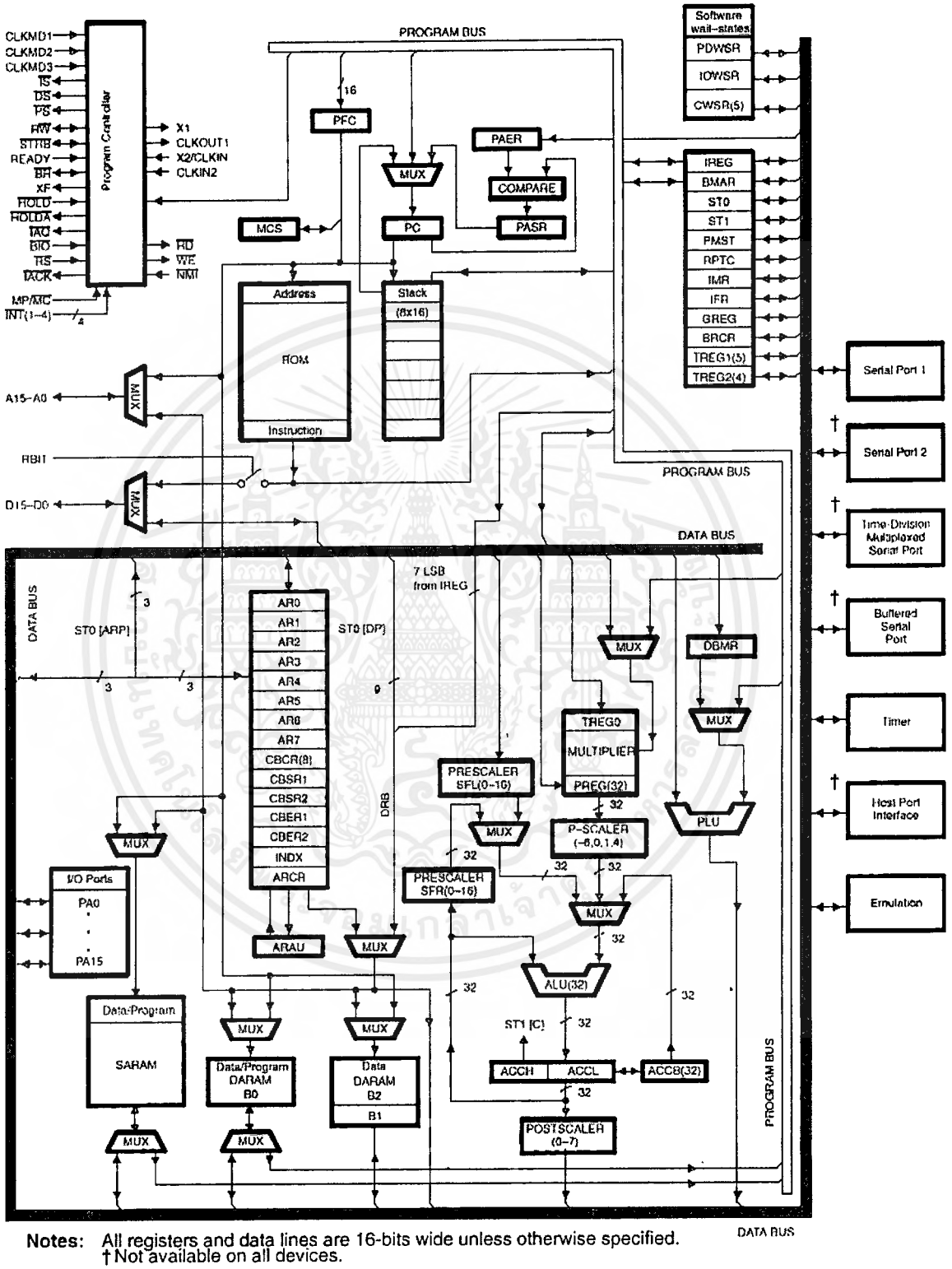
4.1 องค์ประกอบภายในและความสามารถของ TMS320C50

4.1.1 องค์ประกอบภายในของ TMS320C50

1. มีแรมบนชิบ (On-Chip RAM) 10K เวิร์ด
2. มีแรมสำหรับโปรแกรม/ข้อมูล
3. มีรอมขนาด 2K x 16 บิต สำหรับการบูต
4. มี ALU (Arithmatic Logic Unit) ACC (Accumulator) และ ACCB (Accumulator buffer) ขนาด 32 บิต
5. มี PLU (Parallel Logic Unit) ขนาด 16 บิต
6. มีคำสั่งการคูณ 16 บิต ที่ทำงานใน 1 ไซเคิล
7. มีรีจิสเตอร์ถึง 8 ตัว ในการคำนวณและเก็บค่า
8. มีพอร์ตอนุกรมที่มีการรับส่งแบบฟูลดูเพล็กซ์ (Full Duplex)
9. มี TDM (Time-Division Multiple) ของพอร์ตอนุกรม
10. มีพอร์ต I/O ได้ถึง 64 K และมี 16ตำแหน่ง สำหรับการเข้าถึงหน่วยความจำ

4.1.2 ความสามารถของ TMS320C50

1. ทำงานแบบไปป์ไลน์ (Pipeline)
2. ทำงานได้รวดเร็ว ถึง 30-50 ns ต่อหนึ่งคำสั่ง
3. ใช้งานแทน 'C1x และ 'C2x ได้
4. สามารถต่อหน่วยความจำข้างนอกได้ถึง 224K x 16 บิต ซึ่งประกอบด้วย หน่วยความจำเก็บข้อมูล 64K สำหรับ I/O และอื่นๆอีก 64K
5. สามารถกำหนดสัญญาณนาฬิกา โดยใช้ไทม์เมอร์และ เคาท์เตอร์ (Counter)



Notes: All registers and data lines are 16-bits wide unless otherwise specified.
† Not available on all devices.

รูปที่4.1 บล็อกไดอะแกรมของ TMS320C50

สถาปัตยกรรมของ TMS320C50 สร้างขึ้นเพื่อความเร็วในการทำงานด้าน DSP และเพื่อให้การทำงานของบัสไม่ขึ้นต่อกัน จึงแยกโปรแกรมบัส (Program Bus)และดาตาบัส (Data Bus)ออกจากกัน โดยโปรแกรมบัสจะเป็นทางเข้าของรหัสคำสั่ง (Instruction Code) และโอแปรนด์ของคำสั่ง ส่วนดาตาบัสจะเชื่อมต่อโดยตรงระหว่างหน่วยความจำที่ใช้เก็บข้อมูลกับวงจรการทำงานประมวลผล เช่น ALU AR0-AR7 ซึ่งโครงสร้างการคำนวณทางคณิตศาสตร์นี้ยึดหลักให้ทำงานด้วยประสิทธิภาพ เช่น การเลื่อนบิต (Shift) การคูณ และคำสั่งทางลอจิก

ตาราง 4.1 ตำแหน่งขาและหน้าที่การทำงานของ TMS320C50

***หมายเหตุ* สถานะ (I) = Input , (O) = Output , (Z) = High-Impedance , (S) = Supply**

สัญญาณ	ขา	สถานะ	การทำงาน
Address and Data bus			
$A0 - A9$ $A10 - A15$	55-64 72-77	I/O/Z	เป็นบัสแบบขนาน (Parallel Address Bus)ใช้สำหรับชี้ตำแหน่งของหน่วยความจำข้อมูล และหน่วยความจำโปรแกรม หรือ I/O ภายนอก เมื่ออยู่ใน โฮล์ดโหมด (Hold Mode) จะเป็นอิมพีแดนซ์สูง (High Impedance) สัญญาณเหล่านี้ใช้เป็นอินพุต เมื่อ $\overline{HOLDA}$ และ $\overline{BR}$ ถูกขับ (Drive) ให้มีสถานะต่ำ (Low)
$D15 - D8$ $D7 - D0$	6-13 23-30	I/O/Z	เป็นบัสข้อมูลแบบขนาน (Parallel Data Bus)ใช้ส่งข้อมูลระหว่างซีพียูหลัก (Core CPU) กับหน่วยความจำข้อมูล/โปรแกรมภายนอก หรืออุปกรณ์ I/O เมื่อไม่มีเอาท์พุทสัญญาณเหล่านี้จะเป็นอิมพีแดนซ์สูง หรือเมื่อขา $\overline{RS}$ หรือ $\overline{HOLD}$ อยู่ในสถานะต่ำ(active low) และขา OFF เป็นสถานะต่ำ นอกจากนี้ยังใช้สำหรับ DMA ภายนอกของแรม (Single Access RAM)
Memory Control Signals			
$\overline{DS}$ $\overline{PS}$ $\overline{IS}$	89 91 90	O/Z	เลือกหน่วยความจำข้อมูล/โปรแกรม และ I/O ปกติมีสถานะสูง แต่เมื่อเป็นสถานะต่ำจะเป็นการติดต่อภายนอก เมื่อขา $\overline{OFF}$ อยู่ในสถานะต่ำ จะมีอิมพีแดนซ์สูง

สัญญาณ	ขา	สถานะ	การทำงาน
READY	128	I	สัญญาณข้อมูลพร้อม (Data Ready Input) ใช้แสดงเมื่ออุปกรณ์ภายนอกส่งข้อมูลเรียบร้อยแล้ว และเมื่อยังทำงานไม่เสร็จ (*READY*=0) จะต้องมีการรอ 1 cycle และเช็คขา *READY* อีกครั้ง ในสาวะปกติขา *READY* จะทำงาน เมื่อมีสัญญาณ $\overline{BR}$ เข้ามา
$R/\overline{W}$	92	I/O/Z	สัญญาณอ่าน/เขียน (Read/Write Signal) เป็นสัญญาณควบคุมการอ่านเขียนข้อมูล เมื่ออยู่ในโฮล์ดโหมด จะมีอิมพีแดนซ์สูง ถูกใช้ใน DMA ของแรมภายนอกเมื่อ $\overline{HOLDA}$ และ $\overline{IAQ}$ อยู่ในสถานะต่ำ ใช้แสดงทิศทางของบัสข้อมูล สำหรับ DMA อ่าน (Read อยู่ในสถานะสูง)และ เขียน (Write อยู่ในสถานะต่ำ)
$\overline{STRB}$	93	I/O/Z	สถานะสูง จะเป็นสถานะต่ำ เมื่อค่าของบัสภายนอกเป็นอิมพีแดนซ์สูง ใน โฮลด์โหมด เมื่อ $\overline{HOLDA}$ และ $\overline{IAQ}$ แอคทีฟ สัญญาณนี้จะใช้เลือกการเข้าถึงหน่วยความจำ
$\overline{RD}$	82	O/Z	สัญญาณเลือกอ่าน (read select)ขานี้จะทำงานเมื่อมีการอ่าน จะต่อโดยตรงกับ $\overline{OE}$ ของอุปกรณ์ภายนอก สัญญาณนี้จะใช้อ่านค่าหน่วยความจำโปรแกรม/ข้อมูลและ I/O ภายนอก ทั้งหมดเป็นอิมพีแดนซ์สูงเมื่ออยู่ในโฮลด์โหมด
$\overline{WE}$	83	O/Z	สัญญาณเขียน (Write Enable) ขานี้จะใช้ในการเขียนค่าหน่วยความจำโปรแกรม/ข้อมูล และ I/O ภายนอก เมื่ออยู่ในโฮลด์โหมดจะมีอิมพีแดนซ์สูง

สัญญาณ	ขา	สถานะ	การทำงาน
Mulltiprocessing Signals			
$\overline{HOLD}$	129	I	สัญญาณโฮลด์อินพุท (Hold Input)เป็นสัญญาณที่ใช้เพื่อแสดงว่ากำลังมีการติดต่อกับบัสตำแหน่ง บัสข้อมูล บัสควบคุม เมื่อถูกตอบรับ(Acknowledge)โดย 'C5x จะอยู่ในสถานะอิมพีแดนซ์สูง
$\overline{HOLDA}$	108	O/Z	สัญญาณตอบรับสัญญาณโฮล์ด (Hold Acknowledge signal) ใช้แสดงว่าวงจรอยู่ในสถานะโฮล์ด (Hold State) บัสตำแหน่ง,บัสข้อมูล,บัสควบคุมอยู่ในสภาวะอิมพีแดนซ์สูง
$\overline{BR}$	94	I/O/Z	สัญญาณการขอใช้บัส (Bus Request Signal) แสดงเมื่อมีการติดต่อกับหน่วยความจำข้อมูล สัญญาณจากขานี้ใช้กับหน่วยความจำที่ว่างได้ 32 k word เมื่อขา $\overline{HOLDA}$ อยู่ในสถานะต่ำสัญญาณจากขานี้ใช้กับ DMA ของแรมภายนอก $\overline{BR}$ จะมีสถานะต่ำเมื่อ มีการติดต่อกับแรมายนอก
$\overline{IAQ}$	1	O/Z	สัญญาณรับคำสั่ง(Instruction Acquisition Signal) จะแสดงสถานะต่ำเมื่อมีคำสั่งอยู่ในแอดเดรสบัส ใช้กับ DMA ของแรมภายนอกเมื่อ $\overline{HOLDA}$ อยู่ในสถานะต่ำ
$\overline{BIO}$	130	I	สัญญาณควบคุมการบรานซ์ (Brance Control Input) ถ้าเป็นสถานะต่ำจะเป็นให้ทำคำสั่งที่เป็นเงื่อนไข สัญญาณนี้จะทำงานเมื่อมีการ Fetch ที่เป็นเงื่อนไข
XF	109	O/Z	สัญญาณติดต่อภายนอก (External Flag Output) ถูกเซ็ตให้เป็นสถานะสูงหรือสถานะต่ำโดยคำสั่งพิเศษหรือโดยโหลดค่าใน stare register ($ST1$) เมื่อมีการรีเซ็ตขานี้จะมีสถานะสูง

สัญญาณ	ขา	สถานะ	การทำงาน
$\overline{IACK}$	112	O/Z	สัญญาณตอบรับอินเตอร์รัพต์ (Interrupt Acknowledge Signal) แสดงค่าเมื่อมีการรับค่าอินเตอร์รัพต์ และค่าโปรแกรมเคาท์เตอร์
Initialization , Interrupt , and Reset Operations			
$\overline{INT4}$ $\overline{INT3}$ $\overline{INT2}$ $\overline{INT1}$	41 40 39 38	I	สัญญาณอินเตอร์รัพต์จากผู้ใช้ภายนอก (External User Interrupt Input) กำหนดโดยรีจิสเตอร์ควบคุมอินเตอร์รัพต์ (Interrupt Mask Register) และบิตอินเตอร์รัพต์โหมด (Interrupt Mode Bit) สามารถรีเซ็ตผ่าน รีจิสเตอร์บอกอินเตอร์รัพต์ (Interrupt Flag Register)
$\overline{NMI}$	42	I	สัญญาณนอน-มาร์กเอเบิลอินเตอร์รัพต์ (Non Maskable Interrupt) เป็นอินเตอร์รัพต์ภายนอกไม่สามารถควบคุมโดย *INTM* หรือ *MR* เมื่อ $\overline{NMI}$ ทำงานจะมีการอินเตอร์รัพต์
$MP/\overline{MC}$	5		ขาเลือกโหมดไมโครโปรเซสเซอร์/ไมโครคอมพิวเตอร์(Microprocessor/Microcomputer Mode Select Pin) ถ้าเป็นสถานะต่ำ (Microcomputer Mode) จะทำให้โปรแกรมรอมภายในถูกส่งไปยังหน่วยความจำโปรแกรม (Program Memory Space)
Oscillator/Timer Signals CLKIN ½			
*CLKOUT*1	110	O/Z	สัญญาณนาฬิกาส่งออก (master clock output signal หรือ CLKIN2 frequency) มีค่าไซเคิลเท่ากับอัตราแมชชีนไซเคิล (machine-cycle) ของซีพียู
*CLKMD*1 *CLKMD*2	71 103	I	*CLKMD*1 *CLKMD*2 0 0 สัญญาณนาฬิกาภายนอกเป็นสัญญาณนาฬิกาเข้าจากขา X2/CLKIN ทำให้ออสซิเลเตอร์ (oscillator) ภายใน และ PLL disable 0 1 สำหรับตรวจสอบ

สัญญาณ	ขา	สถานะ	การทำงาน
(ต่อ) *CLKMD*1 *CLKMD*2	 71 103	 I	*CLKMD*1 *CLKMD*2 1 0 เป็นสัญญาณนาฬิกาอินพุต (input clock) สำหรับ CLKIN2 ทำให้ออสซิเลเตอร์ภายในหยุดทำงาน และ PLL ทำงานแทน 1 1 เป็นสัญญาณนาฬิกาอินพุตสำหรับขา *X2/CLKIN*1 ทำให้ออสซิเลเตอร์ภายในทำงาน และ PLL ภายในไม่ทำงาน
*X2/CLKIN*1	96	I	ขาอินพุตสำหรับออสซิเลเตอร์ภายใน ถ้าออสซิเลเตอร์ภายในไม่ถูกใช้งาน สัญญาณนาฬิกาจะเป็นอินพุทของอุปกรณ์บนขานี้ แมชชีนไซเคิลภายในเป็นครึ่งหนึ่งของอัตราของสัญญาณนาฬิกา (Clock Rate)
*X*1	97	O	เป็นขาเอาท์พุทของออสซิเลเตอร์ภายใน สำหรับคริสตรอล ถ้าไม่ใช้ออสซิเลเตอร์ภายในจะไม่มีการต่อกับขานี้
*CLKIN*2	95	I	เป็นอินพุตสำหรับสัญญาณนาฬิกาสำหรับขับอัตราแมชชีน(Machine Rate)
TOUT	122	O	เอาท์พุตไทม์เมอร์ (Timer Output) ขานี้ให้สัญญาณพัลส์เมื่อไทม์เมอร์ภายใน (On-Chip Timer) นับถึง 0 ความกว้างพัลส์เท่ากับ *CLKOUT*1 cycle
Serial Port Signals			
CLKR *TCLKR*	46 126	I I	เป็นขาที่รับสัญญาณนาฬิกาจากข้างนอกเพื่อกำหนดให้การรับข้อมูล (*DR/TDR*) เข้าไปเก็บไว้ที่ RSR (Serial Port Receive Shift Regirter) แต่ถ้าขานี้ไม่ใช้สามารถที่จะใช้เป็นขาอินพุต ของ *IN*0 ของ *SPC/TSPC* รีจิสเตอร์ได้

สัญญาณ	ขา	สถานะ	การทำงาน
CLKX *TCLKX*	124 123	I/O/Z I/O/Z	เป็นขาแสดงสัญญาณนาฬิกาจากภายนอกเพื่อกำหนดให้ *DR/TDR* ส่งข้อมูลไปที่ *DX/TDX* *CLKX* จะเป็นอินพุตถ้า *MCM* บิตใน Serial Port Control Register มีค่าเป็นศูนย์ และอาจจะขับความถี่เป็น CLKOUT1/4 เมื่อ *MCM* มีค่าเป็น 1 ถ้าขานี้ไม่ใช้สามารถที่จะทำเป็นอินพุตของบิต *IN*1 ของ *SPC/TSPC* รีจิสเตอร์
DR *TDR*	43 44	I I	เป็นขาเพื่อรับสัญญาณของข้อมูล ซึ่งเมื่อรับมาแล้วจะเก็บไว้ที่ RSR (Serial Port Receive Shift Regirter)
DX *TDX*	106 107	O/Z	เป็นขาเพื่อส่งสัญญาณข้อมูล ซึ่งข้อมูลจะส่งจาก XSR (Serial Port Transmit Shift Register)
FSR *TFSR/TADD*	45 125	I I/O/Z	แสดงสัญญาณการพร้อมของเฟรม(Frame synchronization) สำหรับรับสัญญาณอินพุต *TFSR* จะเป็นได้ทั้งอินพุต/เอาท์พุท เมื่อพอร์ตอนุกรมอยู่ในโหมด *TDM*
FSX *TFSX/TFRM*	104 105	I/O/Z I/O/Z	แสดงสัญญาณการพร้อมของเฟรม (Frame Synchronization) สำหรับการส่งสัญญาณขานี้จะเลือกได้โดยทางซอฟต์แวร์ จะเป็นเอาท์พุทเมื่อ *TXM* ถูกเซ็ตให้เป็น 1
Test Signals			
TCK	34	I	เปลี่ยน TAP (Test Access Port) ซึ่งเป็นอินพุตสัญญาณนาฬิกา จะควบคุม TAP รีจิสเตอร์คำสั่ง (Instruction Register) หรือเลือกการทดลองรีจิสเตอร์ข้อมูล (Data Register) ที่ขอบขาขึ้นของ *TCK* ในการเปลี่ยนสัญญาณเอาท์พุทจะปรากฏที่ขอบขาลงของ *TCK*
TDI	67	I	เป็นสัญญาณนาฬิกาเพื่อเลือกรีจิสเตอร์ในขอบขาขึ้นของ *TCK*

สัญญาณ	ขา	สถานะ	การทำงาน
TDO	100	O/Z	เป็นการทดสอบเอาท์พุทข้อมูลในขอบขาลงของ *TCK*
TMS	31	I	เป็นการเลือกโหมดทดสอบ JTAG และเป็นสัญญาณนาฬิกาอินพุตที่ทดสอบพอร์ต ตรวจสอบการเข้าถึง (Test Access Port) TAP จะทำงานที่ขอบขาขึ้นของ *TCK*
$\overline{TRST}$	2	I	ทดสอบการรีเซ็ตจะเป็นสถานะสูง
EMU1/$\overline{OFF}$	118 119	I/O/Z I/O/Z	Emulator pin1/disable all output จะทำงานที่สถานะเอาท์พุท

4.2 การจัดหน่วยความจำของ TMS320C50

ในการเลือกหน่วยความจำของ TMS320C50 นั้นขึ้นอยู่กับการเซ็ตรีจิสเตอร์ โดยสำหรับ program memory นั้น จะมีการเลือกโหมดของคอมพิวเตอร์ว่าจะใช้โหมดไมโครโปรเซสเซอร์หรือไมโครคอมพิวเตอร์ นอกจากนั้นขึ้นอยู่กับค่าใน บิต CNF และ บิต RAM ว่ามีค่าเป็น 0 หรือ 1 สำหรับใน data memory จะขึ้นอยู่กับการเซ็ตค่าในบิต CNF และบิต OVLY สำหรับรายละเอียดของแต่ละส่วนและการเซ็ตค่าในแต่ละบิตเป็นดังนี้คือ

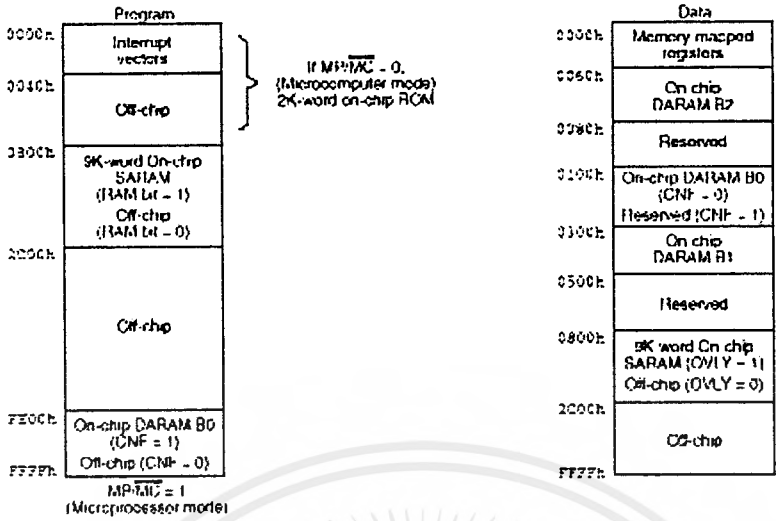
4.2.1 หน่วยความจำข้อมูล (Data memory)

สำหรับ TMS320C50 นี้จะมีการแยกหน่วยความจำข้อมูล (data memory) กับหน่วยความจำโปรแกรม(program memory) ออกจากกันซึ่งจะทำให้ความเร็วดีขึ้น สำหรับหน่วยความจำสามารถขยายได้ถึง 64K x 16 บิต ซึ่งสำหรับ 'C50 จะมีแรมบนชิพถึง 9K ซึ่งแรมจะเป็นแบบ SRAM (single-access RAM) และ DRAM (dual-access RAM) ขนาด 1056 เวิร์ด ซึ่งมีข้อดี คือ

- ทำงานได้รวดเร็วเนื่องจากทำงานโดยวิธีไปป์ไลน์
- ลดค่าใช้จ่ายในเรื่องการต่อหน่วยความจำภายนอก
- มีการปฏิบัติงานได้เร็วไม่มีการรอ
- ช่วยประหยัดไฟ ซึ่งถ้าให้การต่อหน่วยความจำภายนอกจะเปลืองกว่า

ในการกำหนดหน่วยความจำข้อมูล จะต้องมีการเซ็ตค่าของบิตของรีจิสเตอร์ควบคุมหน่วยความจำ(Control state Register) ดังตารางที่ 4.2

สำหรับการติดต่อกับรีจิสเตอร์ต่างๆจะทำบนเพจ0 ซึ่งจะเป็นการใช้แบบส่งแบบรีจิสเตอร์ (Register Memory Map)



รูปที่ 4.2 แสดงถึงการจัดหน่วยความจำข้อมูลใน TMS320C50

ตารางที่ 4.2 การเซ็ตบิตรีจิสเตอร์ควบคุมหน่วยความจำเพื่อใช้หน่วยความจำข้อมูล

CNF	OVLY	DRAMB0	DRAM B1	DRAM B2	SRAM	Off-Chip
0	0	100h-2FFh	300h-4FFh	60h-7Fh	-	800hFFFFh
0	1	100h-2FFh	300h-4FFh	60h-7Fh	800h-BFFh	C00h-FFFh
1	0	-	300h-4FFh	60h-7Fh	-	800h-FFFh
1	1	-	300h-4FFh	60h-7Fh	800h-BFFh	C00h-FFFh

4.2.2 หน่วยความจำโปรแกรม (Program memory)

ในการใช้งานหน่วยความจำโปรแกรม สามารถขยายได้ถึง 64 K ซึ่ง ' C50 มีรอม SRAM และ DRAM ซึ่งมีความเร็วสูง โดยไม่มีสภาวะการรอ (wait state)

ในการทำงานของซีพียู 'C50 สามารถใช้ร่วมกับรอมภายในขนาด 4K ซึ่งสามารถโปรแกรมจากโรงงาน มีความเร็วในการทำงานเต็มที่ ในการเลือกใช้จะต้องกำหนดที่ขาของ $MP/\overline{MC}$ หากขาดังกล่าวเป็นสถานะสูง ตำแหน่ง 4K แรกจะเป็นหน่วยความจำภายนอกชิพ แต่ถ้าเป็นสถานะต่ำ ตำแหน่ง 4K เวิร์ดแรกจะเป็นรอมภายในชิพ การกำหนดสถานะของค่าต่างๆจะกำหนดดังนี้

ตารางที่ 4.3 ตำแหน่งแอดเดรสข้อมูลเพจ 0

Address			
Dec	Hex	Name	Description
0–3	0–3	—	Reserved
4	4	IMR	Interrupt mask register
5	5	GREG	Global memory allocation register
6	6	IFR	Interrupt flag register
7	7	PMST	Processor mode status register
8	8	RPTC	Repeat counter register
9	9	BRCR	Block repeat counter register
10	A	PASR	Block repeat program address start register
11	B	PAER	Block repeat program address end register
12	C	TREG0	Temporary register 0 (used for multiplicand)
13	D	TREG1	Temporary register 1 (used for dynamic shift count)
14	E	TREG2	Temporary register 2 (used as bit pointer in dynamic bit test)
15	F	DBMR	Dynamic bit manipulation register
16	10	AR0	Auxiliary register 0
17	11	AR1	Auxiliary register 1
18	12	AR2	Auxiliary register 2
19	13	AR3	Auxiliary register 3
20	14	AR4	Auxiliary register 4
21	15	AR5	Auxiliary register 5
22	16	AR6	Auxiliary register 6
23	17	AR7	Auxiliary register 7
24	18	INDX	Index register
25	19	ARCR	Auxiliary register compare register
26	1A	CBSR1	Circular buffer 1 start register
27	1B	CBER1	Circular buffer 1 end register
28	1C	CBSR2	Circular buffer 2 start register
29	1D	CBER2	Circular buffer 2 end register
30	1E	CBCR	Circular buffer control register
31	1F	BMAR	Block move address register
32–35	20–23	—	Memory-mapped serial port registers†
36–42	24–2A	—	Memory-mapped peripheral registers†
43–47	2B–2F	—	Reserved for test/emulation
48–55	30–37	—	Memory-mapped serial port registers†
56–79	38–4F	—	Reserved
80–95	50–5F	—	Memory-mapped I/O ports†
96–127	60–7F	—	Scratch-pad RAM (DARAM block B2)

ตาราง 4.4 การกำหนดค่าสำหรับใช้หน่วยความจำโปรแกรม

CNF	*RAM*	*MP / $\overline{MC}$*	*ROM*	*SRAM*	*DRAM B0*	*Off – Chip*
0	0	0	0000-07FF			0800-FFFF
0	0	1				0000-FFFF
0	1	0	0000-07FF	0800-2BFF		2C00-FFFF
0	1	1		0800-2BFF		0000-07FF
						2C00-FFFF
1	0	0	0000-07FF		FE00-FFFF	0800-FDFF
1	0	1			FE00-FFFF	0000-FDFF
1	1	0	0000-07FF	0800-2BFF	FE00-FFFF	2C00-FDFF
1	1	1		0800-2BFF	FE00-FFFF	0000-07FF
						2400-FDFF

4.3 โหมดการอ้างถึงหน่วยจำ (Memory Addressing Mode)

'C50 สามารถอ้างแอดเดรสได้ 64K word สำหรับหน่วยความจำโปรแกรม (program memory) และ 96K word สำหรับหน่วยความจำข้อมูล สำหรับวิธีในการอ้างแอดเดรส ทำได้ด้วยกัน 8 ทาง

1. การเข้าถึงข้อมูลโดยตรง (direct address bus, DRB) ในการอ้างถึงรีจิสเตอร์แบบนี้จะต้องมีรีจิสเตอร์สำหรับชี้เพจข้อมูล โดยใช้ DP ในการชี้ DP (data page pointer) สามารถอ้างได้ 512 เพจ โดยแต่ละเพจมี 128 เวิร์ด

   ตัวอย่าง ADD 01h เป็นคำสั่งในการนำข้อมูลที่ชี้ที่ตำแหน่ง 01h โดยมี DP บอกว่าชี้ที่เพจใด นำมาบวกกับ ACC

2. โดยการเข้าถึงข้อมูลในหน่วยความจำ (Memory map) จะเหมือนกับแบบแรกแต่จะใช้เฉพาะเพจ 0 เท่านั้น
3. โดยใช้ ออกซิลลารี รีจิสเตอร์ (Auxiliary register) โดยการอ้างแอดเดรสที่บรรจุใน ARO-AR7 เช่น ADD* ซึ่ง ARP จะเป็นตัวบอกว่าเป็น ARx ตัวที่เท่าใด
4. โดยการใช้รีจิสเตอร์คำสั่ง (Instruction Register) ซึ่งเป็นการเอาแอดเดรสโดยตรงมาจากคำสั่งเลย
5. โดยการใช้ PC (program counter) จะเหมือนการใช้รีจิสเตอร์คำสั่ง แต่สามารถอ้างแบบยาว (Long Immediate (0000-FFFF))

6. โดยเข้าถึงรีจิสเตอร์โดยตรง (Register Access) เป็นการใช้รีจิสเตอร์ที่ทำงานโดยเฉพาะ เช่น TREGO, TREG1, TREG2, ARCR, DBMR
7. โดยใช้คำสั่งตัวที่ 2 เป็นตัวชี้แอดเดรส เช่น คำสั่ง BLDD # 02345h, 012h หมายถึงเอาข้อมูลที่ชี้ที่แอดเดรส 02345h ไปเก็บที่แอดเดรส 012h
8. โดยอ้างถึงหน่วยความจำเป็นบล็อก (Block Memory Address Register) ซึ่ง BMAR จะเป็นตัวชี้ข้อมูล ตวอย่างเช่น BLDD BMAR, 012h

ตารางที่ 4.5 ตำแหน่งและลำดับความสำคัญของอินเตอร์รัพต์เวคเตอร์

NAME	LOCATION		PRIORITY	FUNCTION
	DEC	HEX		
$\overline{RS}$	0	0	1(highest)	สัญญาณการรีเซ็ตภายนอก
$\overline{INT1}$	2	2	3	อินเตอร์รัพต์ภายนอก 1
$\overline{INT2}$	4	4	4	อินเตอร์รัพต์ภายนอก 2
$\overline{INT3}$	6	6	5	อินเตอร์รัพต์ภายนอก 3
TINT	8	8	6	สัญญาณอินเตอร์รัพต์เวลาภายใน
RINT	10	A	7	สัญญาณอินเตอร์รัพต์การรับข้อมูลของพอร์ตอนุกรม
XINT	12	C	8	สัญญาณอินเตอร์รัพต์การส่งข้อมูลของพอร์ตอนุกรม
TRNT	14	E	9	สัญญาณอินเตอร์รัพต์การรับข้อมูลของ TDM พอร์ต
TXNT	16	10	10	สัญญาณอินเตอร์รัพต์การส่งข้อมูลของ TDM พอร์ต
$\overline{INT4}$	18	12	11	อินเตอร์รัพต์ภายนอก 4
	20-23	14-17	N/A	Reserved
	26-33	1A-21	N/A	Reserved
TRAP	34	22	N/A	Software Trap Instruction
$\overline{NMI}$	36	24	2	นอนมาร์กเอเบิลอินเตอร์รัพต์
	38-39	26-27	N/A	Reserved
	40-63	28-3F	N/A	Software Interrupts

4.4 อุปกรณ์เสริม (Peripherals)

TMS320C50 มีการอินเตอร์เฟสกับภายนอกได้ถึง 8 วิธี ซึ่งประกอบด้วย

1. อินเตอร์รัพต์ (Interrupt)
2. พอร์ตอนุกรม (Serial Port)
3. พอร์ตอนุกรมแบบแบ่งเวลา (TDM Serial Port)
4. ไทม์เมอร์ (Timer)
5. รีจิสเตอร์ใช้โปรแกรมสภาวะการรอ (Software-Programable Wait State)
6. พอร์ตติดต่อภายนอก (I/O port)
7. รีจิสเตอร์หารสัญญาณนาฬิกา (Divide-By-One Clock)
8. XF และ BIO

บทที่5

โครงสร้างของ TMS320C50 DSP Starter Kit

TMS320C5x นั้นเป็นโปรเซสเซอร์ที่ได้รับความนิยมอย่างแพร่ในงานด้านวิศวกรรม เพราะเป็นโปรเซสเซอร์ที่สามารถทำงานแบบเวลาจริง(Real Time) และเป็นการประมวลผลเชิงเลข (Digital Signal Processing) ซึ่งในปัจจุบันมีความสำคัญมากขึ้น

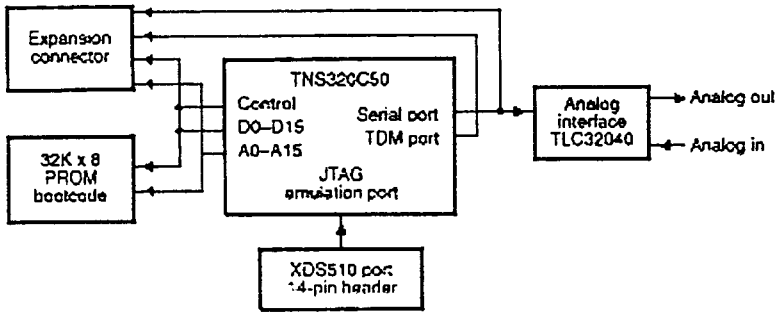
ต่อจากนี้จะกล่าวถึงโครงสร้างของ TMS320C50 DSP Starter Kit ซึ่งเป็นDSP สำเร็จรูป สามารถใช้งานได้เลย มีรายละเอียดดังต่อไปนี้

5.1 ลักษณะทั่วไปของบอร์ด

- ใช้โปรเซสเซอร์เบอร์ TMS320C50 ซึ่งจะมีลักษณะเป็น Fixed-Point DSP
- ใช้เวลาประมาณ 50ns ต่อหนึ่งคำสั่ง (Instruction Cycle Time)
- 32K byte PROM (Programable Read Only Memory)
- ใช้มาตรฐาน RCA คอนเนคเตอร์ สำหรับสัญญาณอนาลอกอินพุทและเอาท์พุท เพื่อให้สามารถต่อเข้ากับออสซิโลสโคป และ ฟังก์ชั่นเจนเนอเรเตอร์ ได้โดยตรง
- ในการติดต่อกับการควบคุมมีคอนเนคเตอร์ XDS510 ติดต่อกับคอมพิวเตอร์ทางพอร์ต อนุกรมสามารถใช้ I/O บัสได้เพื่อสำหรับการออกแบบภายนอก

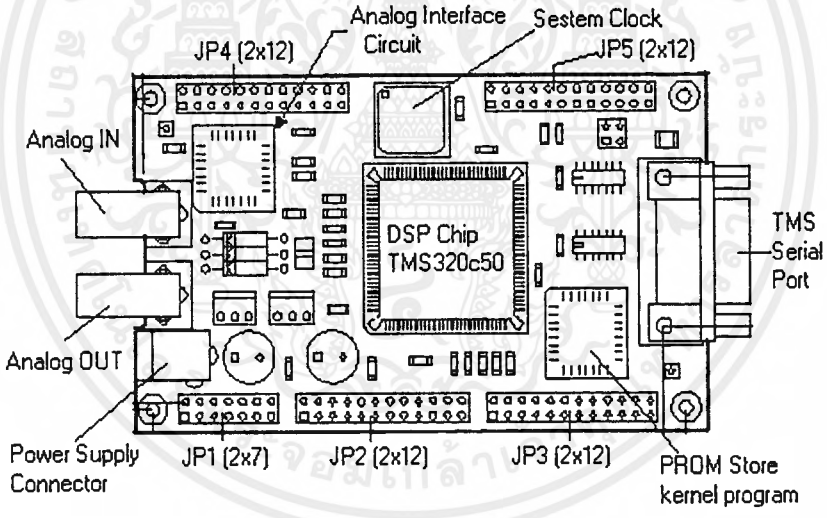
5.2 ส่วนประกอบที่สำคัญของบอร์ด

1. TMS320C50
2. ส่วนของ Analog Interface TLC32040
3. PROM ขนาด 32Kx8bit
4. RCA คอนเนคเตอร์ , Power Supply คอนเนคเตอร์
5. Serial to Pararell Shift Register ซึ่งจะต่อแบบ Pararell กับ TMS320C50
6. System Clock 40 Mhz
7. TDM Serial Port ซึ่งจะต่อกับ RS232 Cable(DB9 female)
8. LM7805 LM7905 เป็นไอซีเรกกูเรเตอร์
9. 14 pin XDS 510 สำหรับต่อกับฮาร์ดแวร์ภายนอก



รูปที่5.1 แสดงบล็อกไดอะแกรมของ TMS320C5x DSK

รูปที่5.1แสดงบล็อกไดอะแกรมของบอร์ดซึ่งประกอบด้วย โฮสอินเตอร์เฟส, อนาลอกอินเตอร์เฟส และ อิมูเลชั่นพอร์ททำให้สามารถติดต่อกับพีซีได้โดยผ่านทาง RS232 นอกจากนี้ยังมี PROM ขนาด 32 Kbyte ที่ใช้เก็บคอร์แนลโปรแกรมไว้สำหรับบูต ส่วนของอนาลอกอินเตอร์เฟสใช้TLC32040 ซึ่งเป็นวงจรอินเตอร์เฟสสัญญาณอนาลอกที่มีคอนเนคเตอร์ 2 ตัว สำหรับอินพุทและเอาท์พุท



รูปที่5.2 แสดงส่วนประกอบของบอร์ดTMS 320C5X DSP Starter Kit

5.2.1 วงจรส่วน CPU TMS320C50

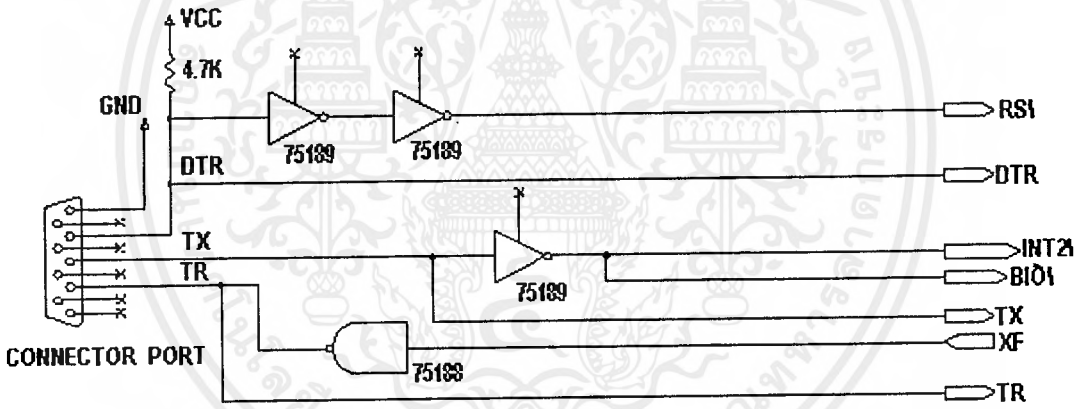
ในที่นี้จะทำงานอยู่ในโหมดไมโครคอมพิวเตอร์ โดยกำหนดให้ $MP/\overline{MC}$ อยู่ในสถานะต่ำซึ่งจะทำให้โปรแกรมภายใน ROM ถูกส่งไปยังหน่วยความจำโปรแกรม(Program Memory Space) การติดต่อกับ RS232 จะต่อที่ขา $\overline{BIO}$ และ XF ซึ่ง $\overline{BIO}$ จะเป็นสัญญาณควบคุมการบรานซ์ (Brance Control Input) ส่วนสัญญาณ XF เป็นสัญญาณติดต่อภายนอก(External Flag Output) สัญญาณนาฬิกาของระบบจะใช้คริสตอลขนาด 40 MHz ต่อเข้าที่ $CLKIN$ ในส่วนของ $CLKOUT1$ จะต่อไปยัง TLC32040

สำหรับการติดต่อกับ PROM จะใช้บัสแอดเดรส $A0-A14$ และบัสข้อมูล $D0-D7$ ในการอ่านใช้ขา $\overline{RD}$ และ OE โดยการเลือกชิปที่ CE และ BR ในส่วนของแรมที่ใช้จะใช้แรมภายใน TMS320C50

การติดต่อกับ TLC32040 จะติดต่อผ่านทาง Serial Port โดยจะรับข้อมูลที่ผ่านการแปลงจากสัญญาณอนาลอกเป็นสัญญาณดิจิตอลที่ TLC32040 โดยรับข้อมูลทางขา DR เมื่อข้อมูลผ่านการประมวลผลแล้วจะถูกส่งกลับไปยัง AIC ทางขา DX

5.2.2 วงจรส่วนการติดต่อกับ RS232

ข้อมูลจาก พีซี จะติดต่อกับตัวบอร์ดผ่านทางพอร์ตอนุกรมโดยข้อมูลจะเข้าที่ขา TX ผ่าน U7A75189 เพื่อเปลี่ยนสัญญาณ RS232 มาเป็นสัญญาณ TTL แล้วเข้า TMS320C50 ที่ $\overline{INT2}$ และ $\overline{BIO}$ ส่วนในการส่งข้อมูลจาก TMS ไปยัง PC จะผ่านที่ขา XF โดยมีไอซีเบอร์ 75188 เปลี่ยนระดับสัญญาณ TTL เป็นสัญญาณ RS232 อีกที ซึ่งการติดต่อกับ RS232 จะแสดงได้ดังรูปที่ 5.3



รูปที่ 5.3 แสดงการติดต่อกับRS232

5.2.3 วงจรส่วน TLC32040

เมื่อสัญญาณอินพุทเข้ามาทางขา IN+ IN- หรือ AUX IN+ AUX IN- ของTLC32040 ซึ่งใน TLC จะมี A/D เพื่อแปลงสัญญาณอนาลอกเป็นสัญญาณดิจิตอล สัญญาณที่ได้จะถูกส่งไปยัง TMS320C50 เพื่อทำการประมวลผล สัญญาณที่ได้จาก TMS320C50 จะถูกส่งกลับมายัง AIC ทางขา DX เมื่อ AIC รับสัญญาณโดยผ่านทางพอร์ตอนุกรม D/A จะทำการแปลงสัญญาณกลับมาเป็นสัญญาณอนาลอกสัญญาณที่ได้ผ่าน Low Pass Filter เพื่อกำจัดสัญญาณความถี่สูงออกและส่งออกที่ขา OUT+ OUT-

5.2.4 วงจรส่วนไฟเลี้ยงและหัวต่อขยายบอร์ด

บอร์ดทำงานโดยใช้ไฟขนาด ±5V โดยมีการต่อหม้อแปลงขนาด $9V_{DC}$ เข้าทาง Power Supply Connector แรงดันนี้จะผ่านส่วนเรกกูเรเตอร์ให้แรงดันเอาท์พุทคงที่ โดยใช้ ไอซี LM7805 สำหรับไฟซีกบวก และ LM7905 สำหรับไฟซีกลบ ส่วนการขยายบอร์ด หรือส่วนที่สามารถติดต่อกับภายนอกจะมีอยู่ด้วยกัน 5 ส่วนคือ JP1-JP5 ซึ่งเป็นการต่อกับ TMS320C50 และ TLC32040ทั้งหมด

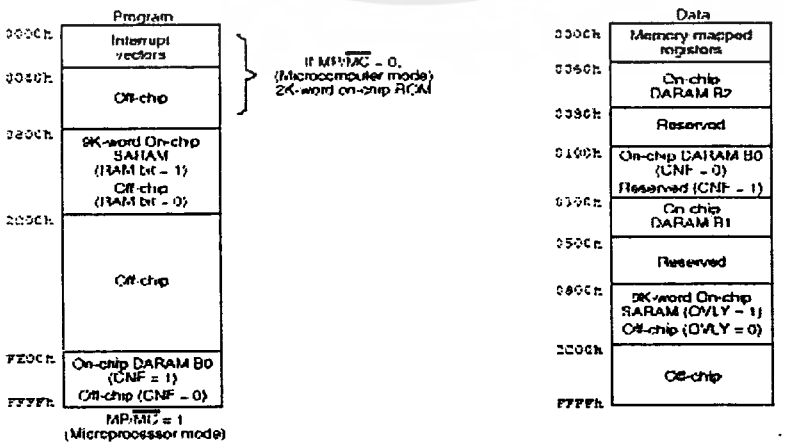
5.3 การจัดหน่วยความจำใน 'C50 DSK

ในการแบ่งหน่วยความจำใน 'C50 จะแบ่งเป็น 4 ส่วนคือ

- Program Memory จะเป็นตัวที่จะเก็บคำสั่งที่จะปฏิบัติการ(Executed)
- Local Data Memory ใช้เก็บข้อมูล(Data) ที่ใช้ในคำสั่ง
- Global Data Memory
- Input / Output Space มีขนาด 64K ใช้ในการติดต่อ(Interface)กับหน่วยความจำภายนอก

จากรูปสามารถกำหนดหน่วยความจำที่ต้องการใช้งานได้ ในส่วนหน่วยความจำข้อมูลจะกำหนดที่ บิต CNF และ OVLY ซึ่งอยู่ใน Status Register โดยในส่วนนี้จะแบ่งเป็น

- DRAM B0 (100h-2FFh)
- DRAM B1(300h-4FFh)
- DRAM B2(60h-7Fh)
- SRAM(800h-BFFh)
- OFF-CHIP



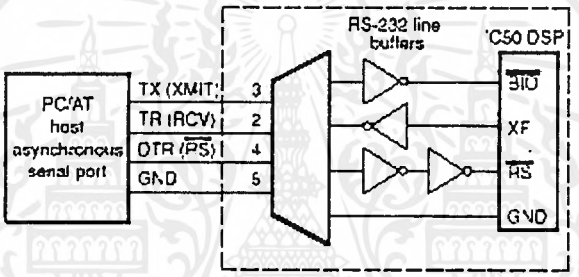
รูปที่ 5.4 หน่วยความจำภายใน 'C50

สำหรับหน่วยความจำโปรแกรม สามารถขยายได้ถึง 64K การเลือกหน่วยความจำที่จะใช้งานจะกำหนดที่ บิต CNF RAM MP/MC หน่วยความจำส่วนนี้แบ่งเป็น

- ROM (0000-1FFFh)
- SRAM (2000h-23FFh)
- DRAM B0 (FE00h-FFFFh)
- Off-Chip

5.4 การต่อใช้งาน DSK

การต่อใช้งาน 'C50 กับ PC จะใช้ขา XF กับ BIO ซึ่งต่อผ่านพอร์ตอนุกรม RS232 ดังแสดงได้ดังรูป

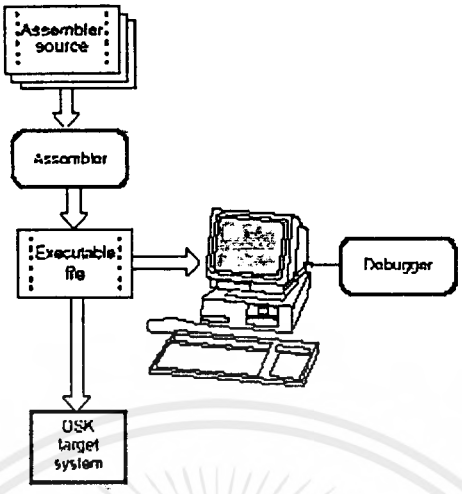


รูปที่5.5 การต่อระหว่าง DSK กับ PC โดยผ่านทาง RS232

นอกจากนี้ DSK ยังมีแอสแซมเบอร์ และดีบักเกอร์ ของมันเอง ซึ่งจะมีประโยชน์มากเนื่องจากภาษาแอสแซมบลีที่ใช้ใน DSK นั้นจะสนับสนุนการทำงานด้านการประมวลผลสัญญาณ (Signal Processing) และดีบักเกอร์ก็มีความสามารถที่จะทำงานในแบบทีละขั้นและแบบเบรคพอยท์เพื่อช่วยในการแก้ไขโปรแกรม

5.5 การสร้างโปรแกรมเพื่อใช้กับ DSK

1. ทำการสร้างไฟล์หลัก (source file) สำหรับโปรแกรม
2. .แปลงไฟล์หลักโดยใช้ DSK แอสแซมเบอร์
3. หากต้องการตรวจแก้ไขโปรแกรมโดยใช้ดีบักเกอร์ก็ทำได้โดยใช้โปรแกรม dsk5d.exe ซึ่งให้มากับบอร์ดแล้ว



รูปที่5.6 แสดงขั้นตอนการสร้างโปรแกรม

บทที่ 6

การส่งข้อมูลผ่านพอร์ตขนาน

(Interfacing the Standard Parallel Port)

พอร์ตขนานในเครื่องคอมพิวเตอร์โดยทั่วไปประกอบด้วยขาสัญญาณอินพุท 9 บิท และเอาท์พุท 12 บิทซึ่งเป็นสายสัญญาณควบคุม (Control Line) 4 เส้น สายสัญญาณสถานะ (Status Line) 5 เส้น และสายสัญญาณข้อมูล (Data Line) 8 เส้น ต่อเชื่อมกับเครื่องคอมพิวเตอร์ด้วยฟีเมลคอนเนคเตอร์ (Female Connector) ขนาด 25 ขา

โหมดการใช้งานการส่งข้อมูลติดต่อกับคอมพิวเตอร์ทางพอร์ตขนานตามมาตรฐาน IEEE1284 ประกอบด้วยการใช้งานใน 5 โหมด คือ

1. Compatibility Mode or Centronics Mode
2. Nibble Mode
3. Byte Mode
4. EPP Mode (Enhance Parallel Port)
5. ECP Mode (Enhance Capabilities Port)

ใน Centronics Mode ,Nibble Mode และ Byte Mode อาจจัดรวมได้เป็นกลุ่ม Standard Parallel Port (SPP) ส่วนในอีก 2 โหมดจำเป็นต้องใช้ร่วมกับฮาร์ดแวร์เพิ่มเติมที่มีความเร็วสูง

การใช้งานในโครงงานนี้เลือกใช้ Standard Parallel Port จึงขอกล่าวเน้นเฉพาะเนื้อหาในส่วนที่เกี่ยวข้องนี้เท่านั้น

โดยปกติ Centronics Modes จะส่งข้อมูลด้วยความเร็ว 50 กิโลไบท์ต่อวินาที และอาจสามารถปรับให้มีความเร็วเพิ่มขึ้นไปถึง 150 กิโลไบท์ต่อวินาทีได้ ในการรับข้อมูลจะเปลี่ยนโหมดการทำงานเข้าสู่ นิบเบิล(Nibble Mode)หรือไบท์โหมด (Byte Mode) โดยนิบเบิลโหมดจะรับข้อมูลขนาด 4 บิทจากอุปกรณ์ภายนอกเข้ามาเพียงทิศทางเดียว และไบท์โหมดจะทำงานใน 2 ทิศทางกับข้อมูล 8 บิท

คอนเนคเตอร์ที่ใช้เชื่อมต่อโดยทั่วไปอาจเป็น D-Type 25 pin connector ซึ่งใช้กับพอร์ตขนานโดยทั่วไป หรือ Centronics 34 pin connector ซึ่งใช้ต่อกับเครื่องพิมพ์ (Printer) ประกอบด้วยขาสัญญาณดังต่อไปนี้

ตารางที่ 6.1 แสดงขาสัญญาณและการใช้งานในพอร์ตส่งข้อมูลแบบขนาน

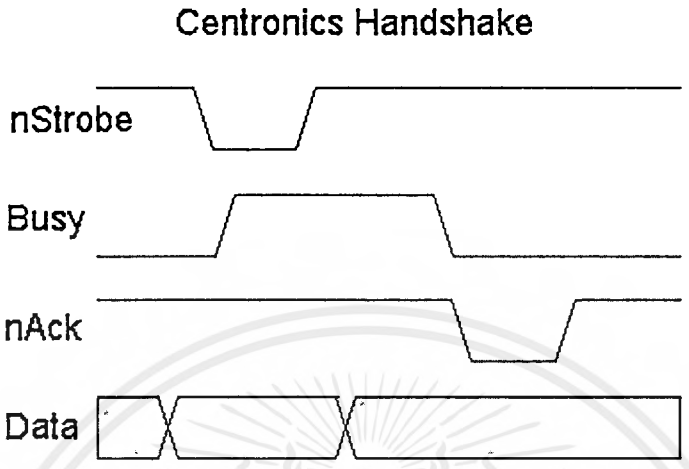
หมายเลขขาสัญญาณ (D-Tybe 25 pin)	หมายเลขขาสัญญาณ (Centronic)	สัญญาณ SPP	ทิศทางข้อมูล (In/Out)	ชนิด Register
1	1	*Strobe	In/Out	Control
2	2	Data 0	Out	Data
3	3	Data 1	Out	Data
4	4	Data 2	Out	Data
5	5	Data 3	Out	Data
6	6	Data 4	Out	Data
7	7	Data 5	Out	Data
8	8	Data 6	Out	Data
9	9	Data 7	Out	Data
10	10	*Ack	In	Status
11	11	Busy	In	Status
12	12	Paper-out / PeperEnd	In	Status
13	13	Select	In	Status
14	14	*Auto-Linefeed	In/Out	Control
15	32	*Error/Fault	In	Status
16	31	*Initialize	In/Out	Control
17	36	*Select-Printer/*Select-in	In/Out	Control
18-25	19-30	Ground	Gnd	

เครื่องหมาย * หมายถึงสัญญาณทำงานที่สถานะต่ำ (Active Low)

6.1 การควบคุมการส่ง-รับ ข้อมูลด้วยซอฟแวร์(Centronics Handshake)

ข้อมูลที่จะส่งจะเข้ามารออยู่ที่ขาสัญญาณที่ 2 –9 ดังรูป และทำการตรวจสอบว่าเครื่องพิมพ์ว่างและอยู่ในสภาวะที่พร้อมจะใช้งานหรือไม่ หากเครื่องพิมพ์ว่างสัญญาณ Busy จะมีสถานะต่ำ โปรแกรมจึงจะส่งสัญญาณ Strobe สถานะต่ำออกไปนาน 1 µS.เครื่องพิมพ์จะอ่านข้อมูลเข้าไป

แสดงสภาวะการอ่านข้อมูลนี้ด้วยขาสัญญาณ Busy ที่ มีสถานะสูง จนเมื่อเครื่องพิมพ์รับข้อมูลทั้งหมดเรียบร้อยแล้วจะส่งสัญญาณ Ack สถานะต่ำออกมานาน 5 µS



รูปที่ 6.1 สัญญาณควบคุมการส่งข้อมูล

ในส่วนของการเขียนโปรแกรมจริง จึงเพียงแต่สั่งเขียนข้อมูลออกไปยังเครื่องพิมพ์เท่านั้น และฮาร์ดแวร์จะทำการตรวจสอบสภาวะการทำงานต่างๆเอง ยกเว้นแต่สัญญาณ Ack ที่จะไม่ได้รับการตรวจสอบ

6.2 พอร์ตแอดเดรส (Port Address)

พอร์ตขนานถูกใช้งานโดยกำหนดแอดเดรสได้ 3 แบบดังตารางที่ 6.2

ตารางที่ 6.2 แสดงแอดเดรสพอร์ตของพอร์ตขนาน

แอดเดรสพอร์ต	คำอธิบาย
3BCh-3BFh	ใช้ติดต่อกับวิดีโอการ์ด(Vidio Card) ซึ่งถูกควบคุมด้วยโปรแกรม BIOS และไม่สามารถใช้งานในโหมด ECP ได้
378h-37Fh	แอดเดรสที่ใช้งานใน LPT1
278h-27Fh	แอดเดรสที่ใช้งานใน LPT2

6.3 Software Register-Standard Parallel Port (SPP)

6.3.1 ดาตาพอร์ต(Data Port)

ดาตาพอร์ต(Data Port) หรือเรียกอีกชื่อหนึ่งว่ารีจิสเตอร์พอร์ต (Register Port) ใช้ในการส่งข้อมูลออกจาก PC สู่อุปกรณ์ภายนอก ถ้าพอร์ตขนานเป็นแบบ 2 ทาง(Bi-Directional) ก็จะสามารถรับข้อมูลเข้าได้ด้วย พอร์ตนี้ประกอบด้วย ขา 2-9 ดังตารางที่ 6.3

ตารางที่ 6.3 ดาตาพอร์ต (Data Port)

ออฟเซ็ท(Offset)	ชื่อ	การทำงาน	หมายเลขบิท	คุณสมบัติ
Base+0	พอร์ตส่งข้อมูล(Data Port)	เขียนเพียงทิศทางเดียว หรือทำงาน 2 ทิศทางกรณี Bi-Directional	Bit 7	Data 7 (Pin 9)
			Bit 6	Data 6 (Pin 8)
			Bit 5	Data 5 (Pin 7)
			Bit 4	Data 4 (Pin 6)
			Bit 3	Data 3 (Pin 5)
			Bit 2	Data 2 (Pin 4)
			Bit 1	Data 1 (Pin 3)
			Bit 0	Data 0 (Pin 2)

6.3.2 พอร์ตสถานะ (Status Port)

พอร์ตสถานะเป็นพอร์ตทิศทางเดียว ทำการอ่านข้อมูลเพียงอย่างเดียวเท่านั้น การเขียนข้อมูลด้วยพอร์ตนี้ไม่สามารถใช้งานได้ พอร์ตสถานะประกอบด้วย 5 ขาสัญญาณ ดังตารางที่ 6.4

ตารางที่ 6.4 พอร์ตสถานะ (Status Port)

ออฟเซ็ท(Offset)	ชื่อ	การทำงาน	หมายเลขบิท	คุณสมบัติ
Base+1	พอร์ตสถานะ (Status Port)	อ่านอย่างเดียว (Read Only)	Bit 7	Busy
			Bit 6	Ack
			Bit 5	Paper Out
			Bit 4	Select In
			Bit 3	Error
			Bit 2	IRQ(Not)
			Bit 1	Reserved
			Bit 0	Reserved

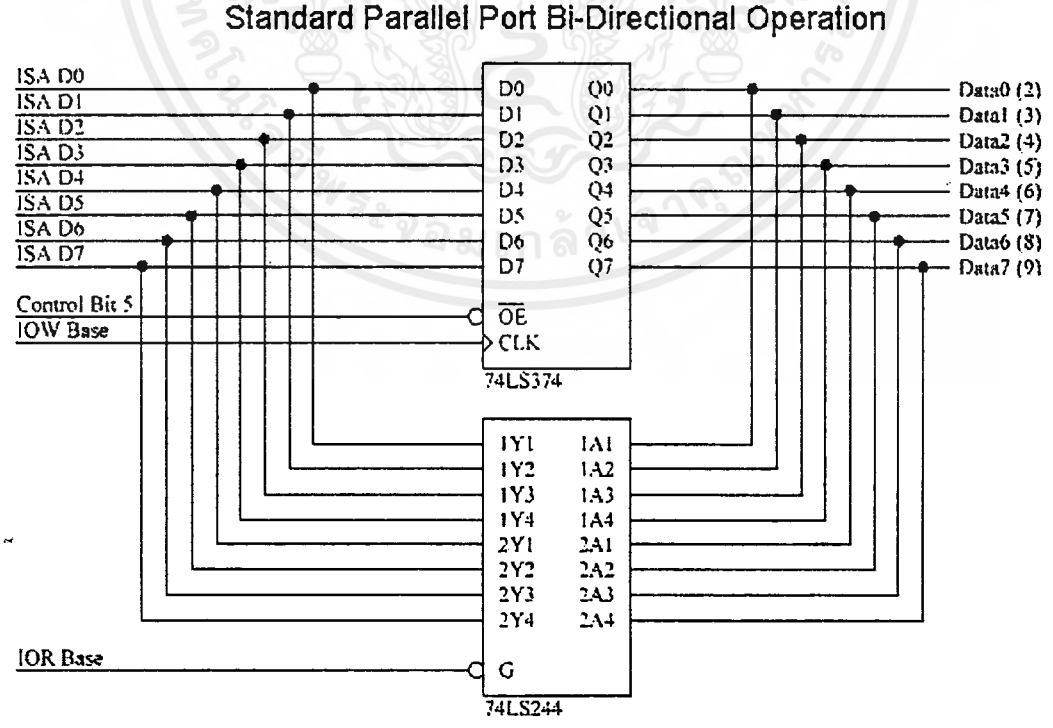
6.3.3 พอร์ตควบคุม (Control Port)

พอร์ตควบคุมเป็นพอร์ตทิศทางเดียว ทำการเขียนข้อมูลเพียงอย่างเดียวเท่านั้น พอร์ตสถานะประกอบด้วย 8 ขาสัญญาณ ดังตารางที่ 6.5

ตารางที่ 6.5 พอร์ตควบคุม (Control Port)

ออฟเซ็ท(Offset)	ชื่อ	การทำงาน	หมายเลขบิท	คุณสมบัติ
Base+2	พอร์ตควบคุม (Control Port)	อ่าน/เขียน (Read/Write)	Bit 7	Unused
			Bit 6	Unused
			Bit 5	Enable bi-directional Port
			Bit 4	Enable IRQ Via Ack Line
			Bit 3	Select Printer
			Bit 2	Initialized Printer (Reset)
			Bit 1	Auto Line Feed
			Bit 0	Strobe

6.4 Bi-directional Ports



รูปที่ 6.2 แสดงการต่อรีจิสเตอร์พอร์ตขนาน

จากรูปที่ 6.2 แสดงการต่อรีจิสเตอร์ข้อมูลซึ่งทำหน้าที่เป็นบัฟเฟอร์ของพอร์ตขนาน กรณีพอร์ตขนานส่งข้อมูลทิศทางเดียวจะสร้างจากไอซี 74LS374 ที่ขาสัญญาณ Output Enable จะมีสถานะต่ำตลอดเวลา ทำให้พอร์ตข้อมูลส่งข้อมูลออกเพียงด้านเดียว การใช้งานให้เป็นพอร์ต 2 ทิศทาง(Bi- Directional Port) จะจ่ายกระแสเกิน (Overdrive) ให้กับขาOE นี้ ขา OE ตรงกับพอร์ตควบคุมบิท 5 ดังนั้น ในทางปฏิบัติจึงควบคุมให้ทำงานส่งข้อมูล 2 ทิศทางผ่านบิทนี้ โดยการเขียน 1 ไปยังบิทควบคุม จะทำให้ขาสัญญาณบิท 2-9 มีสถานะความต้านทานสูง(Hi-Impedance) พอร์ตจะใช้รับข้อมูลเข้ามายังพีซีได้ และยกเลิกการส่งข้อมูล 2 ทิศทางโดยการเขียนค่า 0 มาที่บิท 5 ของพอร์ตควบคุม

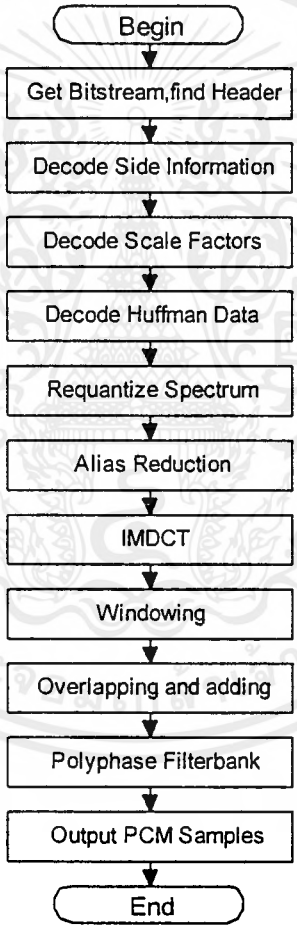
ในกรณีที่ฮาร์ดแวร์ไม่สามารถใช้งานเป็นพอร์ตส่งข้อมูลแบบ 2 ทิศทางได้ อาจส่งข้อมูลอินพุทเข้าไปยังพีซีได้โดยผ่านทางพอร์ตควบคุมและพอร์ตสถานะ ซึ่งทำงานในทิศทางอินพุทได้อยู่แล้วแทน

บทที่ 7

การถอดรหัส ข้อมูล "เอ็มเป็ก-1 เลเยอร์-3"

(MPEG-1 Layer-3 Decoder)

ในบทนี้จะกล่าวถึงการถอดรหัสข้อมูล"เอ็มเป็ก-1 เลเยอร์-3" (MPEG-1 Layer-3 Decoder) การทำงานก็จะคล้ายการเข้ารหัส แต่จะทำตรงข้ามกัน จะมีขั้นตอนตามรูปที่ 7.1 แสดงโฟลว์ชาร์ต (Flow Chart)ของการถอดรหัส ไฟล์ "เอ็มเป็ก-1 เลเยอร์-3"

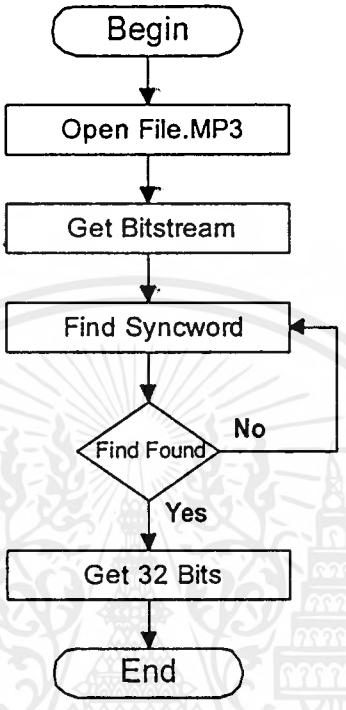


รูปที่ 7.1 แสดงโฟลว์ชาร์ต(Flow Chart)ของการถอดรหัส ไฟล์ "เอ็มเป็ก-1 เลเยอร์-3"

7.1 การดึงบิตข้อมูลและค้นหาส่วนหัวของเฟรม(Get Bitstream and Find Header)

กระบวนการนี้เริ่มต้นด้วย การเปิดไฟล์เอ็มเป็กเลเยอร์3 (MPEG-1 Layer-3 files) ดึงข้อมูลจากสายการไหลของบิต(Bitstream) จากนั้นทำการค้นหาส่วนหัวของเฟรมแรก(Header of First

Frame) โดยค้นหา"ซิงค์เวิร์ด"(syncword) ซึ่งเป็นบิต "1" จำนวน 12 ตัวติดกัน เมื่อพบจึงทำการดึงข้อมูลจากบิตสตรีมมา 32 บิต โดยเริ่มจากบิตแรกของซิงค์เวิร์ด ส่วนรายละเอียดของแต่ละบิตได้กล่าวไปแล้วในบทที่ 4 แล้ว รูปที่ 7.2 แสดงโฟลว์ชาร์ตการทำงานในส่วนนี้



รูปที่ 7.2 แสดงโฟลว์ชาร์ตการทำงานในส่วนนี้

7.2 การถอดรหัสข้อมูลข้างเคียง(Decode Side Information)

ข้อมูลข้างเคียง(Side Information)จะเป็นส่วนต่อจากเฮดเดอร์(ถ้าไม่มี CRC Check Error) ซึ่งมีความสำคัญมากเหมือนกัน เพราะเป็นตัวบอกถึงค่าต่างๆที่จำเป็นในการถอดรหัสเช่น ตารางฮัฟแมน(Huffman Table) เป็นต้น ไฟล์.MP3 ที่เป็นการเข้ารหัสโดยใช้มาตรฐาน ISO/IEC11172-3 และอยู่ในโหมด สเตอริโอ(Stereo mode) ข้อมูลข้างเคียงจะมีขนาด 32 ไบท์ ในส่วนนี้จะดึงข้อมูลจากไฟล์ MP3 ต่อจากเฮดเดอร์อีก 32 ไบท์ จากนั้นก็จะนำข้อมูลจากเฮดเดอร์และข้อมูลข้างเคียงไปถอดรหัสโดยนำไปเทียบกับตารางที่สร้างไว้แล้ว และดึงค่าในตารางมาเก็บไว้เพื่อนำไปใช้ในส่วนต่อไป

ต่อจากส่วนนี้ก็จะเริ่มเข้าสู่ข้อมูลเสียงหลัก(main audio data)ของเฟรม โดยจะมีขนาดเท่ากับขนาดของแต่ละเฟรม(N)ลบด้วยเฮดเดอร์(Header(Bit))และลบด้วยข้อมูลข้างเคียงดังสมการที่ (7.1)

$$Main\_Data(bit) = N - Header(bit) - Side\_Information(bit) \quad \text{.....สมการ 7.1}$$

สำหรับเลเยอร์-3 จะมีขนาดของแต่ละเฟรม (N) ดังสมการที่ 7.2

$$N = 144000 \times \frac{Bitrate}{Sampling\_frequecy} \qquad \text{.....สมการ 7.2}$$

ข้อมูลหลักของเฟรมนั้นๆจะถูกดึงมาเก็บไว้ในบัฟเฟอร์ (Buffer) เพื่อนำไปใช้ในการถอดรหัสเอ็มเป็ก

7.3 การถอดรหัสสเกลแฟกเตอร์(Decode ScaleFactors)

สเกลแฟกเตอร์อยู่ในรูปของบิตที่เรียงต่อๆกันในส่วนของข้อมูลหลัก โดยขึ้นกับค่า slen1 กับ slen2 (2 ค่านี้ถูกกำหนดโดยค่าของ scalefac_compress[gr] ตามตารางที่ ก.7 ในภาคผนวก ก.) จำนวนบิตทั้งหมดของสเกลแฟกเตอร์เรียกว่า "part2_length" ค่าของ part2_length จะขึ้นกับค่าของ Block_type และ mixed_block_flag

- สำหรับ block_type = 0,1,3 (long block)

  part2_length = 11*slen1 + 10 * slen2

- สำหรับ block_type = 2 (short block) และ mix_block_flag = 0

  part2_length = 18*slen1 + 18 * slen2

- สำหรับ block_type = 2 (short + long block) และ mix_block_flag = 1

  part2_length = 17*slen1 + 18 * slen2

กำหนดให้ "scalefac_l" แทนสเกลแฟกเตอร์ของบล็อกยาว(long block)

"scalefac_s" แทนสเกลแฟกเตอร์ของบล็อกสั้น(short block)

ค่าของ สเกลแฟกเตอร์ สามารถจำแนกได้ดังนี้

1. ค่าของ block_type = 0,1,3 มีแต่บล็อกยาวโดย
   - ค่า scalefac_l 11 ตัวแรก มีจำนวนบิตเท่ากับค่าของ slen1
   - ค่า scalefac_l 10 ตัวหลัง มีจำนวนบิตเท่ากับค่าของ slen2
2. ค่าของ block_type = 2 และ mix_block_flag = 0 มีแต่บล็อกสั้นโดย
   - ค่า scalefac_s 18 ตัวแรก มีจำนวนบิตเท่ากับค่าของ slen1
   - ค่า scalefac_s 18 ตัวหลัง มีจำนวนบิตเท่ากับค่าของ slen2
3. ค่าของ block_type = 2 และ mix_block_flag = 1 มีทั้งบล็อกยาวและสั้นโดยที่
   - ค่า scalefac_l 8 ตัวแรก มีจำนวนบิตเท่ากับค่าของ slen1
   - ค่า scalefac_s 9 ตัวต่อมา มีจำนวนบิตเท่ากับค่าของ slen1
   - ค่า scalefac_s 18 ตัวหลัง มีจำนวนบิตเท่ากับค่าของ slen2

7.4 การถอดรหัสข้อมูลฮัฟแมน (Decode Huffman Data)

การถอดรหัสข้อมูลฮัฟแมน (Huffman Data) ใน 1 เฟรม แบ่งเป็น 4 ภาค (region) ซึ่งแต่ละภาคไม่จำเป็นที่จะต้องใช้ตารางฮัฟแมนเดียวกัน ก่อนอื่นต้องกำหนดตารางฮัฟแมนให้แต่ละภาคก่อน โดยค่าของ 3 ภาคแรกถูกเก็บใน table_select และ ของภาคสุดท้ายถูกเก็บใน count1table_select

ต่อไปเป็นการกำหนดว่าในแต่ละภาคใน 3 ภาคแรกจะมีจุดสิ้นสุดที่ตัวที่เท่าใดโดยจะมีขั้นตอนตามรูปที่ 7.3 แสดงโฟลว์ชาร์ตการกำหนดค่าของจุดสิ้นสุดของ 3 ภาคแรก

กำหนดให้ r[0] แทนจุดสิ้นสุดของ ภาคที่ 1

r[1] แทนจุดสิ้นสุดของ ภาคที่ 2

r[2] แทนจุดสิ้นสุดของ ภาคที่ 3

ค่า big value จะเป็นค่าของ 2 เท่าของ big_values ที่อยู่ใน Side Information

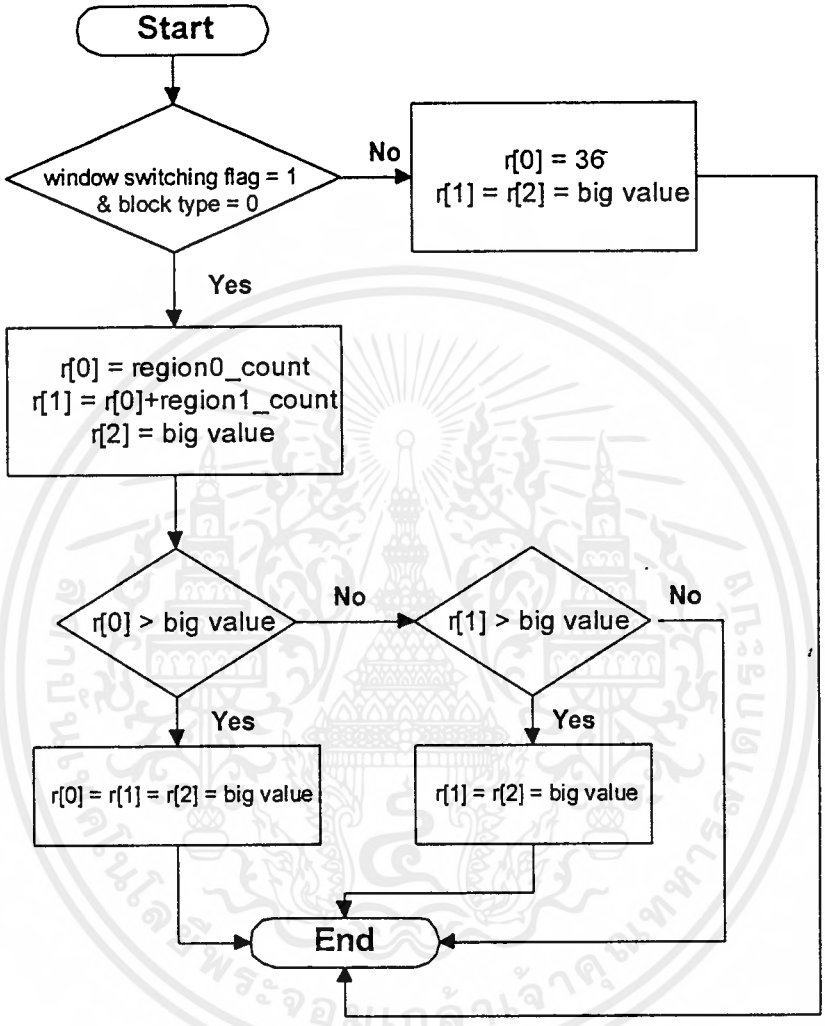
หลังจากกำหนดค่าต่างๆแล้ว ต่อไปจะเป็นการเริ่มการถอดรหัสข้อมูลฮัฟแมนมีขั้นตอนดังนี้

1. ถอดรหัสในภาคแรก และใช้ตารางฮัฟแมนและค่า linbits ของภาคแรก
2. นำหัวของขบวนบิต มาเปรียบเทียบกับค่า hcod ในตารางฮัฟแมน
3. ถ้าตรงกับค่า hcod ตัวไหนก็จะนำค่า x และ y ของ hcod ตัวนั้นมา
4. ถ้าค่า x = 15 ต้องดึงข้อมูลจากขบวนของบิตซึ่งมีจำนวนบิตเท่ากับค่าของ linbit มาบวกเข้ากับ x
5. ถ้า x ไม่เท่ากับ 0 ก็ต้องดึงข้อมูลมาอีก 1 บิตเพื่อกำหนดเครื่องหมาย โดยที่

   ถ้าบิตที่ดึงมามีค่าเป็น 0 x จะมีค่าบวก

   ถ้าบิตที่ดึงมามีค่าเป็น 1 x จะมีค่าลบ
6. สำหรับค่า y ทำเช่นเดียวกับค่า x
7. เก็บค่า x และ y ลงใน “is”
8. เมื่อจำนวนตัวรวมของ x และ y เท่ากับค่า r[0] แล้ว ก็จะเริ่มถอดรหัสฮัฟแมนในภาคที่ 2 และภาคที่ 3 ต่อไป โดยการถอดรหัสจะเช่นเดียวกับภาคแรก แต่จะใช้ตารางฮัฟแมนและค่า linbits ของภาคที่ 2 และภาคที่ 3 ตามลำดับ
9. ในภาคที่ 4 จะใช้ค่าจาก count1table_select โดยค่านี้จะใช้ชี้ตารางฮัฟแมน(Huffman Table) A และ B เท่านั้น (2 ตารางนี้จะให้ค่า 4 ค่าคือ v,w,x,y จากค่า hcod 1 ค่า) ส่วนการถอดรหัสจะเหมือนกับภาคที่ 1 ต่างกันเพียงแต่จะได้ค่าจาก 4 ค่า จุดสิ้นสุดของภาคที่ 4 จะถูกกำหนดโดยค่า part2_3_length ใน side information ลบด้วยค่า

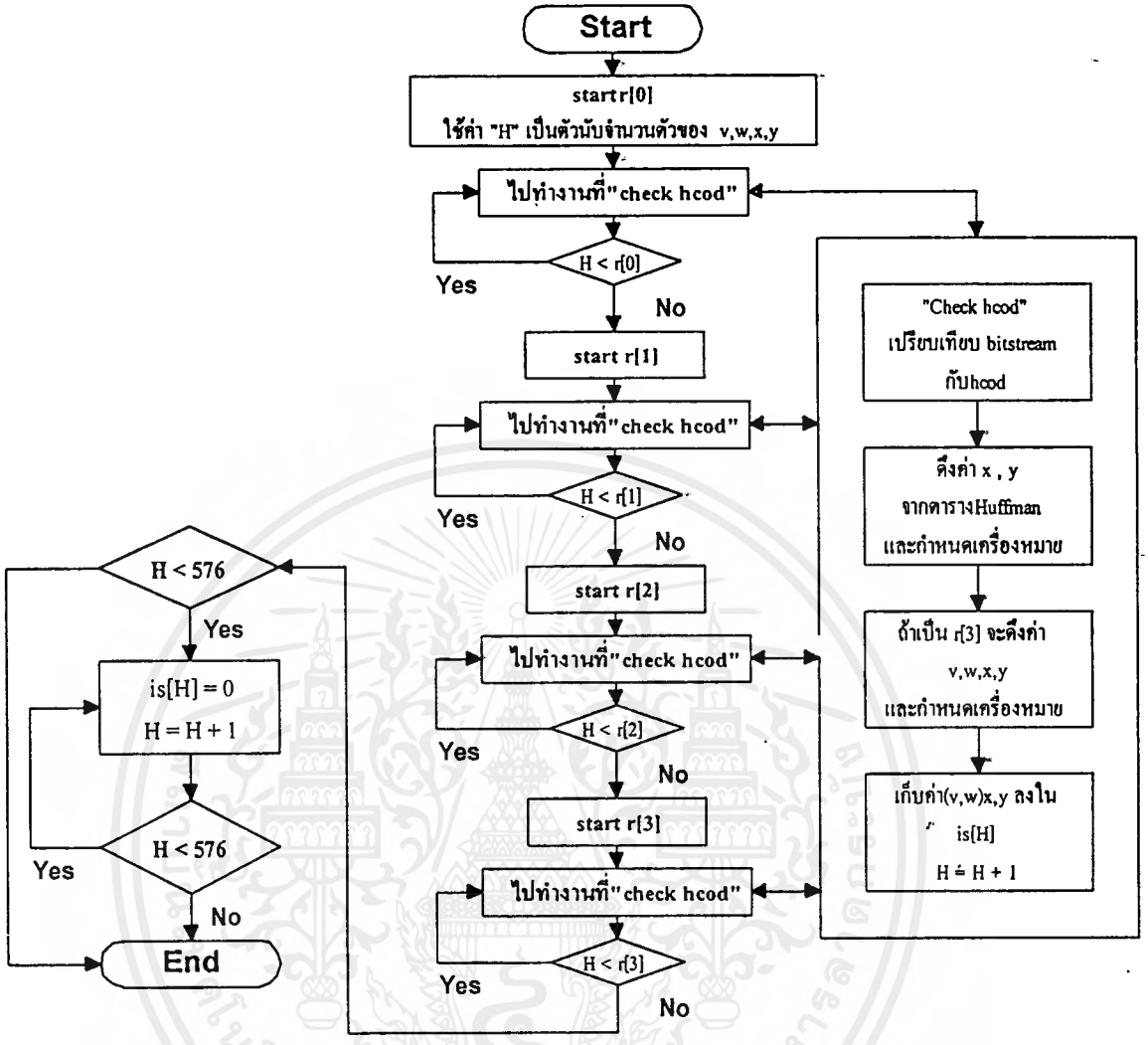
part2_length (จากหัวข้อที่ 7.3) แต่ถ้าค่าของ part2_3_length – part2_length มีค่ามากกว่า 576 จุดสิ้นสุดของภาคที่ 4 จะอยู่ที่ 576



รูปที่ 7.3 แสดงโฟลว์ชาร์ตการกำหนดค่าของจุดสิ้นสุดของ 3 ภาคแรก

10. ในกรณีที่ค่าของ part2_3_length – part2_length มีค่าน้อยกว่า 576 ค่าที่เหลือจากตำแหน่งที่ค่าของ part2_3_length – part2_length ชี้อยู่ จนถึง 567 จะให้มีค่าเป็น "0"

ขั้นตอนต่างๆ เขียนเป็นโฟลว์ชาร์ตได้ดังรูปที่ 7.4 แสดงโฟลว์ชาร์ตของการถอดรหัสฮัฟแมน



รูปที่ 7.4 แสดงโฟลว์ชาร์ตของการถอดรหัสฮัฟแมน

7.5 การรีควอนไตซ์ (Requantize Spectrum)

เป็นการนำเอาค่าจาก "is" มาทำการรีควอนไตซ์จะได้ผลลัพท์เป็นสัญญาณที่ถูกสุ่มตัวอย่าง (Sampling Signal)ในแกนความถี่(Frequency Domain)จะกำหนดให้สัญญาณนี้เป็น "xr"

ในการรีควอนไตซ์จะใช้สูตรทางคณิตศาสตร์ตามสมการที่ 7.3 ซึ่งใช้กับบล็อกสั้น

$$xr_i = sign(is_i) \times |is_i|^{4/3} \times 2^{\frac{1}{4}(global\_gain[gr]-210-(8\times subblock\_gain[window][gr]))} \times 2^{-(scalefac\_multipier\times(scalefac\_s[gr][ch][sfg][window]))} \quad \text{.....สมการที่ 7.3}$$

สมการที่ 7.4 เป็นสมการสำหรับบล็อกยาว

$$xr_i = sign(is_i) \times |is_i|^{4/3} \times 2^{\frac{1}{4}(global\_gain[gr]-210)} \times 2^{-(scalefac\_multipier \times (scalefac\_l[gr][ch][sfg] + preflag[gr] + pretab[sfb]))} \quad \text{.....สมการที่ 7.4}$$

ซึ่งจะเห็นได้ว่าค่าของ xr_i จะขึ้นกับค่าของ

- is_i เป็นค่าเอาท์พุทจากฮัฟแมน
- global_gain[gr] เป็นค่าใน side information
- scalefac_multiplier = scalefac_scale + 1 (ค่า scalefac_scale อยู่ใน side information)
- preflag[gr] เป็นค่าใน side information
- pretab[sfb] ถูกกำหนดโดย scalefactor band แสดงดังตารางที่ 7.1

ตารางที่ 7.1 แสดงค่า pretab[cb] ที่ scalefactor band ต่างๆ

Scalefactor band	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Pretab[cb]	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	2	2	3	3	3	2

- subblock_rain[window][gr] เป็นค่าใน side information
- ค่า scalefac_l คือค่าสเกลแฟกเตอร์ของบล็อกยาวที่ได้จากการถอดรหัสสเกลแฟกเตอร์ในหัวข้อที่ 7.3
- ค่า scalefac_s คือค่าสเกลแฟกเตอร์ของบล็อกสั้นที่ได้จากการถอดรหัสสเกลแฟกเตอร์ในหัวข้อที่ 7.3

∴ ในหัวข้อนี้จึงมีเพียงการแทนค่าตามสูตรก็จบกระบวนการ แต่ต้องอยู่ในเงื่อนไขดังนี้

ค่าของ scalefac_l และ scalefac_s ใช้ค่าเดียวกันในแต่ละ subband

การใช้สูตร

1. ถ้า window_switching_flag = 1 และ block_type[gr][ch]==2 และ
   1.1 mixed_block_flag = 0 xr_i ทุกตัวจะเป็นบล็อกสั้นหมด
   1.2 mixed_block_flag = 1 xr_i 36ตัวแรกจะเป็นบล็อกยาว ที่เหลือจะเป็นบล็อกสั้นหมด
2. จาก 1 ถ้าไม่ใช่ xr_i ทุกตัวจะเป็นบล็อกยาวหมด

7.6 การลดค่าปลอม (Alias Reduction)

ในกรณีที่ค่า xr_i เป็นบล็อกยาว(block_type ไม่เท่ากับ 2) จะเกิดการเหลื่อมกันของสเปกตรัม จนทำให้เกิดค่าปลอมของ xr_i ขึ้น 1 ตัว (เกิด alias) จึงต้องทำการลดค่าปลอม(Alias Reduction) ลง โดยใช้โปรแกรมดังนี้

```
for (sb = 1 ; sb<32 ; sb++)
        for (i = 0 ; i < 8 ; i++){
                xar[18*sb-1-i] = xr[18*sb-1-i]Cs[i] – xr[18*sb+i]Ca[i]
                xar[18*sb+i] = xr[18*sb+i]Cs[i] + xr[18*sb-1-i]Ca[i]
        }
```

โดยค่า Ca[i] และ Cs[i] หาได้จากสูตรดังนี้

$$Cs\,[i] = \frac{1}{\sqrt{1 + C_i^2}} \qquad \textit{สมการที่ 7.5}$$

$$Ca\,[i] = \frac{C_i}{\sqrt{1 + C_i^2}} \qquad \textit{สมการที่ 7.6}$$

ซึ่งค่า Ci เป็นค่าสัมประสิทธ์ของการลดค่าปลอม(Coefficients for alias reduction) มีค่าดังตารางที่7.2

ตารางที่ 7.2 แสดงค่าสัมประสิทธ์ของการลดค่าปลอม(Coefficients for alias reduction)

(i)	C_I
0	-0.6000
1	-0.5350
2	-0.3300
3	-0.1850
4	-0.0950
5	-0.0410
6	-0.0142
7	-0.0037

7.7 IMDCT (Inverse Modiflied Discrete Cosine Transform)

เป็นการแปลงสัญญาณในแกนความถี่(Frequency Domain)เป็นสัญญาณเชิงเวลา(Time Domain)โดยใช้สูตรดังสมการที่ 7.7 แสดงสูตรการแปลง IMDCT

$$x_i = \sum_{k=0}^{\frac{n}{2}-1} X_k \cos\left(\frac{\pi}{2n}(2i + 1 + \frac{n}{2})(2k + 1)\right) \qquad \text{; for i = 0 to n – 1} \qquad \textit{สมการที่ 7.7}$$

โดยที่ x_i เป็นสัญญาณในแกนเวลา(Time Domain)

X_k เป็นสัญญาณในแกนความถี่(Frequency Domain)

n เป็นจำนวนของตัวอย่างวินโดว์(บล็อกสั้นใช้ n = 12 , บล็อกยาว n = 36)

7.8 การทำวินโดว์(Windowing)

ขึ้นกับค่า Block_type ดังนี้

7.8.1 Block_type = 0 (normal window)

ใช้สูตรดังสมการที่ 7.8

$$z_i = x_i \sin(\frac{\pi}{36}(i+\frac{1}{2})) \quad ; \text{for i = 0 to 35} \qquad \textit{สมการที่ 7.8}$$

7.8.2 Block_type = 1 (start block)

ใช้สูตรดังสมการที่ 7.9

$$z_i = x_i \sin(\frac{\pi}{36}(i+\frac{1}{2})) \quad \text{for i =0 to 17}$$

$$z_i = x_i \quad \text{for i =18 to 23}$$

$$z_i = x_i \sin(\frac{\pi}{12}(i-18+\frac{1}{2})) \quad \text{for i =24 to 29}$$

$$z_i = 0 \quad \text{for i =30 to 35} \qquad \textit{สมการที่ 7.9}$$

7.8.3 Block_type = 3 (stop block)

ใช้สูตรดังสมการที่ 7.10

$$z_i = 0 \quad ;\text{for i = 0 to 5}$$

$$z_i = x_i \sin(\frac{\pi}{12}(i-6+\frac{1}{2})) \quad ;\text{for i = 6 to 11}$$

$$z_i = x_i \quad ;\text{for i = 12 to 17}$$

$$z_i = x_i \sin(\frac{\pi}{36}(i+\frac{1}{2})) \quad ;\text{for i = 18 to 35} \qquad \textit{สมการที่ 7.10}$$

7.8.4 Block_type = 2 (short block)

มี 3 window แยกกัน window ละ 12 ตัว โดยที่

$$y_i^{(j)} = x_i^{(j)} \sin(\frac{\pi}{12}(i+\frac{1}{2})) \quad ;\text{for i = 0 to 11 , j = 0 to 2} \qquad \textit{สมการที่ 7.11}$$

โดยที่ i เป็นลำดับสัญญาณ

j เป็นลำดับวินโดว์

ใช้สูตรดังสมการที่ 7.12

$$z_i = 0 \quad \text{;for i = 0 to 5}$$

$$z_i = y_{i-6}^{(1)} \quad \text{;for i = 6 to 11}$$

$$z_i = y_{i-6}^{(1)} + y_{i-12}^{(2)} \quad \text{;for i = 12 to 17}$$

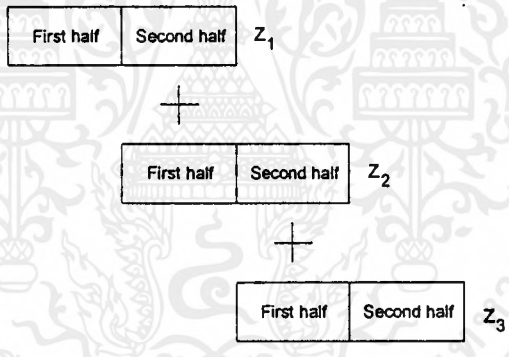
$$z_i = y_{i-12}^{(2)} + y_{i-18}^{(3)} \quad \text{;for i = 18 to 23}$$

$$z_i = y_{i-18}^{(3)} \quad \text{;for i = 24 to 29}$$

$$z_i = 0 \quad \text{;for i = 30 to 35} \qquad \textit{สมการที่ 7.12}$$

7.9 การทำโอเวอร์แลบ(Overlapping)

ครึ่งแรกของบล็อกของค่า 36 ค่า จะทับกับครึ่งหลังของบล็อกก่อนหน้านี้(Previous block) ดังรูปที่ 7.5



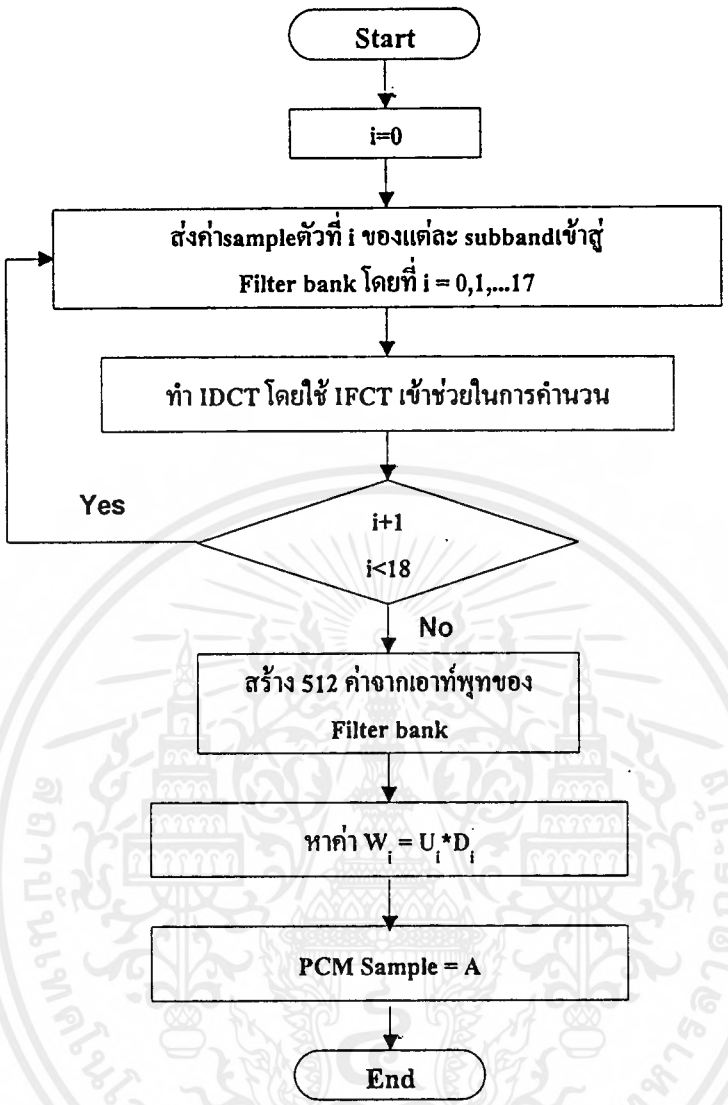
รูปที่ 7.5 แสดงการซ้อนทับ(Overlapping)ของสัญญาณ

จะได้ว่า $\text{result}_i = z_i + s_i$; for i = 0 to 17

$s_i = z_{i+18}$; for i = 0 to 17 .....*สมการที่ 7.13*

7.10 การรวมสัญญาณจากแต่ละช่องตัวกรอง(Synthesis Filter Banks)

หัวข้อนี้เป็นขั้นตอนสุดท้ายของกระบวนการถอดรหัส ได้ผลลัพท์เป็นสัญญาณแบบพีซีเอ็ม(PCM Sample) สามารถอธิบายกระบวนการโดยโฟลว์ชาร์ตดังรูปที่ 7.6 แสดงโฟลว์ชาร์ตแสดงกระบวนการรวมสัญญาณจากแต่ละย่านความถี่ย่อย



รูปที่ 7.6 แสดงโฟลว์ชาร์ตแสดงกระบวนการรวมสัญญาณจากแต่ละย่านความถี่ย่อย

***หมายเหตุ** $A = S_j = \sum_{i=0}^{15} W_{j+32i}$

7.10.1 การส่งค่าอินพุทเข้าไปยังโพลีเฟส ฟิลเตอร์แบงค์ (Polyphase Filter banks)

ค่าที่ส่งให้กับโพลีเฟส ฟิลเตอร์แบงค์ คือค่าของข้อมูลสุ่มตัวอย่างตัวที่ i ของทุกๆ ย่านความถี่ย่อย โดยโพลีเฟส ฟิลเตอร์แบงค์จะรับค่าทีละ 1 ค่าจากแต่ละย่านความถี่ย่อย(ที่ 32 ย่านความถี่ย่อย) ก็คือโพลีเฟส ฟิลเตอร์แบงค์รับค่าและประมวลผลทีละ 32 ค่า

ในแต่ละย่านความถี่ย่อยมีข้อมูลสุ่มตัวอย่างอยู่ 18 ค่า เพราะฉะนั้น โพลีเฟส ฟิลเตอร์แบงค์จะประมวลผล 18 ครั้งต่อแซลแนลต่อเฟรม

7.10.2 IDCT (Inverse Discrete Cosine Transform)

นำ 32 ค่าอินพุทเข้ากระบวนการทำ IDCT โดยนำ32 ค่าอินพุทแทนลงในสมการที่7.14

$$x_i = \sum_{k=0}^{31} X_k \varepsilon_k \cos\left(\frac{\pi k(2i+1)}{32}\right) \qquad \textit{สมการที่ 7.14}$$

โดยที่ x_i เอาท์พุท

X_k อินพุทจากย่านความถี่ย่อย

$\varepsilon_k = \frac{1}{\sqrt{2}}$ สำหรับ k = 0

$\varepsilon_k = 1$ สำหรับ k = 1,2,….,31

7.10.3 การสร้างค่า 512 ค่าจากค่า x_i

นำค่าเอาท์พุท x_i จากหัวข้อ 7.10.2 มาสร้างค่า 512 และกำหนดให้ 512 ค่านี้แทนด้วย U_i โดยที่ i =0,1,2,….,511 ดังโปรแกรมด้านล่าง

```
for i =0 to 7 do
    for j = 0 to 31 do
        U[i*64+j] = x[i*128+j]
        U[i*64+32+j] = x[i*128+96+j]
```

7.10.4 การคูณค่าสัมประสิทธ์ของการสังเคราะห์วินโดว์(Coefficients D_i of the synthesis window)

นำ U_i จากหัวข้อ 5.10.3 มาคูณด้วยค่า D_i ซึ่งค่า D_i นี้ถูกกำหนดอยู่ในมาตรฐานของเอ็มเป็ก ดังสมการที่ 7.15 จะได้ค่า W_i ออกมา

$$W_i = U_i D_i \qquad \textit{สมการที่ 7.15}$$

7.10.5 การคำนวณค่าของข้อมูลสุ่มตัวอย่างPCM

การคำนวณค่าของข้อมูลสุ่มตัวอย่างPCM 32 ค่านี้ สามารถหาได้จากสมการที่ 7.16

$$S_j = \sum_{i=0}^{15} W_{j+32i} \qquad \text{สมการที่ 7.16}$$

ค่า Si ที่ได้ เป็นค่าของการสุ่มตัวอย่างแบบPCM (PCM Sample) เป็นข้อมูลเสียงแบบเชิงเลข (Digital) จากนั้นค่า 32 ค่านี้จะถูกส่งไปยังอุปกรณ์เสียง(Audio Device) ของตัวถอดรหัส ค่าของการสุ่มตัวอย่างแบบPCM จะถูกส่งเข้าสู่อุปกรณ์อย่างต่อเนื่อง และเกิดเสียงต่อเนื่องขึ้นที่อุปกรณ์เสียง

บทที่ 8

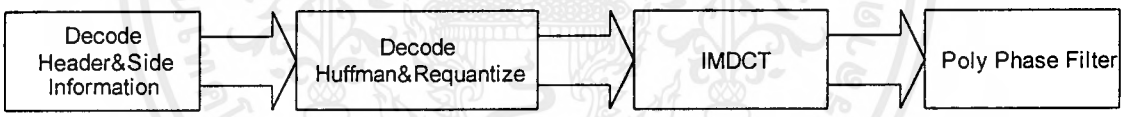
การถอดรหัสเอ็มเป็กโดย PC กับ DSP

(Decoding MPEG Audio By PC&DSP)

ในบทนี้จะได้กล่าวถึง การนำทฤษฎีต่างๆตามที่ได้กล่าวมาประยุกต์ใช้งาน โดยในการถอดรหัสข้อมูลเสียงแบบเอ็มเป็ก (MPEG Audio) นี้ จะแบ่งการทำงานหลักออกเป็น 2 ส่วน คือ ส่วนแรกจะทำงานบนคอมพิวเตอร์ส่วนบุคคล (Personal Computer : PC) และส่วนที่ 2 จะทำงานบน DSP ชุด Starter Kit ใช้พอร์ตขนาน (Parallel Port) ในการเชื่อมต่อและส่งข้อมูลจาก PC ไปยัง DSP เพื่อทำการประมวลผลต่อจนเสร็จ ซึ่งรายละเอียดสามารถอ่านได้ในหัวข้อต่อไปดังนี้

8.1 โปรแกรมภาษาซีบนคอมพิวเตอร์ส่วนบุคคล

ในการถอดรหัสเสียงแบบเอ็มเป็กนั้น จะใช้โปรแกรมภาษาซี (C Language Program) ในการสร้างโปรแกรมในการถอดรหัส โปรแกรมนี้จะใช้งานบนดอส (Dos) ซึ่งมีขั้นตอนต่างๆ ดังรูปที่ 8.1 แสดงถึงบล็อกไดอะแกรมของโปรแกรมถอดรหัสข้อมูลเสียงแบบเอ็มเป็ก



รูปที่ 8.1 แสดงถึงบล็อกไดอะแกรมของโปรแกรมถอดรหัสข้อมูลเสียงแบบเอ็มเป็ก

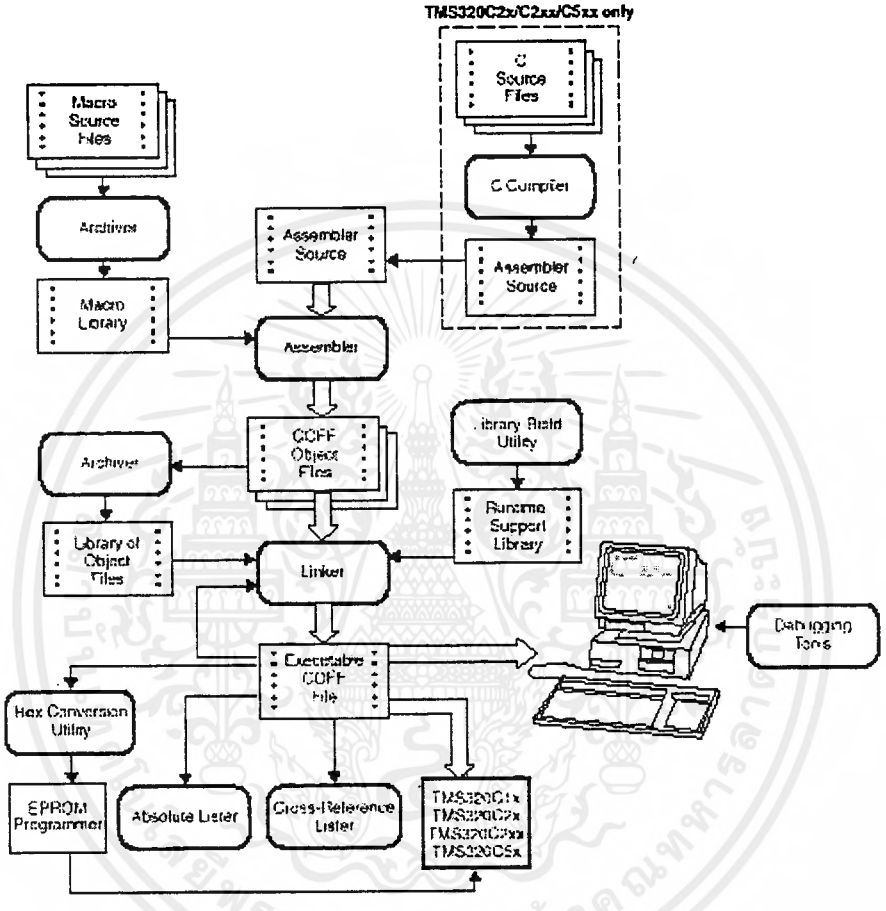
จากรูปแสดงขั้นตอนการทำงานของโปรแกรมซึ่งขั้นตอนต่างๆ ดังกล่าวได้กล่าวไปแล้วในบทที่ 7 และโปรแกรมภาษาซีได้แสดงไว้ในภาคผนวก

โปรแกรมภาษาซีจะทำงานสิ้นสุดที่ขั้นตอนการทำ IMDCT (Inverse Modifies Discrete Cosine Transform) จากนั้นก็จะส่งไปให้ DSP ประมวลผลต่อไป

8.2 การจัดค่าเริ่มต้นของ ดีเอสพี (DSP)

ส่วนของโปรแกรมที่จะนำไปประมวลผลใน ดีเอสพี คือส่วนของโพลีเฟส ฟิลเตอร์ แบงค์ (Polyphase Filter Bank) เป็นต้นไป การเขียนโปรแกรมในส่วนนี้จึงจำเป็นต้องโหลดข้อมูล(Data)ที่ผ่านกระบวนการ ส่วนโปรแกรม(Program)หน่วยความจำโปรแกรม/ข้อมูล (Program/Data memory) ให้ถูกต้อง

ในที่นี้ใช้การลิงค์ (Link) แบบ COFF เนื่องจากสามารถกำหนดการใช้เนื้อที่หน่วยความจำได้ง่ายกว่าการใช้การเขียนโปรแกรมแบบ DSK Format ซึ่งการเขียนโปรแกรมแบบ COFF จะนำออปเจ็คไฟล์ (Object file) มารวมกันให้ได้ไฟล์รูปแบบ COFF เพียงไฟล์เดียว



รูปที่ 8.2 แสดงการสร้างไฟล์แบบ COFF

ขั้นตอนการสร้างไฟล์ COFF

1. สร้างไฟล์ภาษา C
2. ใช้ TMS320C2x/C2xx/C5x คอมไพล์เลอร์ (Compiler) แปลงไฟล์ภาษาซีเป็น ภาษาแอสเซมบลี ใช้ แอสเซมเบอร์ (Asembler) แปลงไฟล์ภาษาแอสเซมบลี เป็นไฟล์แบบ COFF ออปเจ็คไฟล์
3. ลิงค์ ออปเจ็คไฟล์ ทั้งหมดเข้าด้วยกัน โดยใช้ ลิงค์เกอร์ (Linker) โดยขั้นตอนนี้จะต้องจัดการหน่วยความจำด้วย

เมื่อได้ ไฟล์แบบ COFF ที่ผ่านการลิงค์แล้ว สามารถนำไฟล์นี้ไป Run โปรแกรมโดยอาจใช้ดีบัคเกอร์ (Debugger) บน PC เข้าช่วย หรืออาจจะนำไปโหลดลง EPROM โดยใช้ EPROM Programmer

8.2.1การเขียนโปรแกรมเพื่อเริ่มต้นไฟล์แอสซิมบี้

การจัดเก็บอินเตอร์รัปเวคเตอร์ (Interrupt vector) ในที่นี้ใช้นอน-มาสก์เอเบิ้ล อินเตอร์รัป 3 ตัวคือ INT1 ,INT2, RINT,XINT จากหัวข้อ4.3 และจากการตั้งค่าของรีจิสเตอร์ PMST เป็น 0834h จะได้ค่าของตำแหน่งที่เป็นอินเตอร์รัปเวคเตอร์ บิตบนคือ 80x เมื่อค่า x เป็นแอดแคสของอินเตอร์รัปเวคเตอร์ ตามตารางที่ 4.5

ดังนั้นจะได้ตำแหน่งของเวคเตอร์ต่างๆคือ

INT1	ตำแหน่งในหน่วยความจำ	0802h	ถูกใช้โดย	PC
INT2	ตำแหน่งในหน่วยความจำ	0804h	ถูกใช้โดย	ดีบักเกอร์
RINT	ตำแหน่งในหน่วยความจำ	080ah	ถูกใช้โดย	AIC
XINT	ตำแหน่งในหน่วยความจำ	080ch	ถูกใช้โดย	AIC

8.2.2 การจัดค่าเริ่มต้นให้กับ AIC (Analog Interface Circuit)

การทำโครงงานครั้งนี้ ได้ใช้ชุดทดลอง (TMS320C5X Starter Kit) ในการทดลอง เพื่อความสะดวกในการตรวจสอบและปรับปรุงโปรแกรมภาษาแอสเซมบลี โครงสร้าง ของ TMS320C5X Starter Kit ประกอบด้วยส่วนประกอบสำคัญ ส่วนต่างๆ เช่น ชิป TMS320C50, ส่วนอนาลอกอินเตอร์เฟส , PROM ขนาด 32Kx8bit รายละเอียดของได้กล่าวไว้แล้วโดยละเอียดในในบทที่ 5 และ เมื่อต้องการใช้งานส่วนของอนาลอกอินเตอร์เฟส ต้องมีการตั้งค่าเริ่มต้นให้กับส่วนอนาลอกอินเตอร์เฟสเสียก่อน

กระบวนการจัดค่าเริ่มต้นให้กับ AIC ประกอบด้วย

1. การจัดค่าเริ่มต้นให้กับ ดีเอสพี ในส่วนสัญญาณนาฬิกา (On chip timer)
2. การจัดค่าเริ่มต้นให้กับ TIMER และพอร์ตอนุกรม (Serial port)
3. การจัดค่าเริ่มต้นให้กับอนาลอกอินเตอร์เฟส

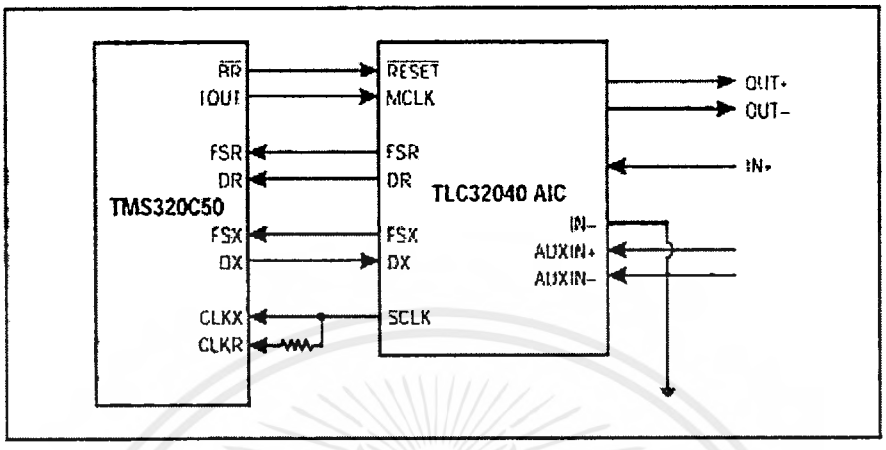
การจัดค่าเริ่มต้นให้กับดีเอสในส่วนสัญญาณนาฬิกา (On chip timer)

จากรูปที่ 8.3 สัญญาณนาฬิกาภายใน TMS320C50 (TOUT) ถูกใช้สร้างสัญญาณนาฬิกาหลักของอนาลอกอินเตอร์เฟส (MCLK) โดยสามารถคำนวณหาค่า TOUT ได้จากสมการที่ 8.1

$$Tout = \frac{Internal\_clock}{(TDDR+1)\times(PRD+1)} \quad \text{.....สมการที่ 8.1}$$

โดย Internal_clock = 20 Mhz

ได้ MCLK 10 Mhz เมื่อ PRD = 1



รูปที่ 8.3 สัญญาณนาฬิกาภายใน TMS320C50

การจัดค่าเริ่มต้นให้กับ TIMER และพอร์ตอนุกรม (Serial port)

การตั้งค่าเริ่มต้นให้กับ TIMER และพอร์ตอนุกรม (Serial port) สามารถทำได้โดยการเปลี่ยนแปลงค่าของ SPC รีจิสเตอร์ ปกติจะตั้งโหมดการเคลื่อนที่อยู่ที่ ซิงโครนัสโหมด (Synchronous mode) ซึ่งต้องให้ค่า SPC เป็น

```
SPLK   #008h,SPC          ; FSM=1 ,XRST และ RRST = 00
SPLK   #0c8h,SPC          ; FSM=1 ,XRST และ RRST = 11
```

การจัดค่าเริ่มต้นให้กับอนาลอกอินเตอร์เฟส

เมื่อวงจรอนาลอกอินเตอร์เฟส.ถูกได้รับสัญญาณนาฬิกาจาก MCLK และมีการจัดค่าเริ่มต้นให้กับ TIMER และพอร์ตอนุกรม (Serial port) เรียบร้อยแล้ว ขั้นตอนต่อไปคือการกำหนดค่าอัตราการสุ่มข้อมูล (Sampling frequency) และค่าความถี่คัทออฟ (Cutoff frequency) ตามต้องการ (ค่าความถี่คัทออฟสูงสุดคือ 19.2 กิโลเฮิรต์)

ค่าความถี่คัทออฟ (ใช้สัญลักษณ์ F_c) สามารถหาได้จากสมการที่ 8.2

$$f_{cLP} = \frac{f_{MCLK}}{2 \times TA} \times \frac{3.6}{288} \qquad \text{.....สมการที่ 8.2}$$

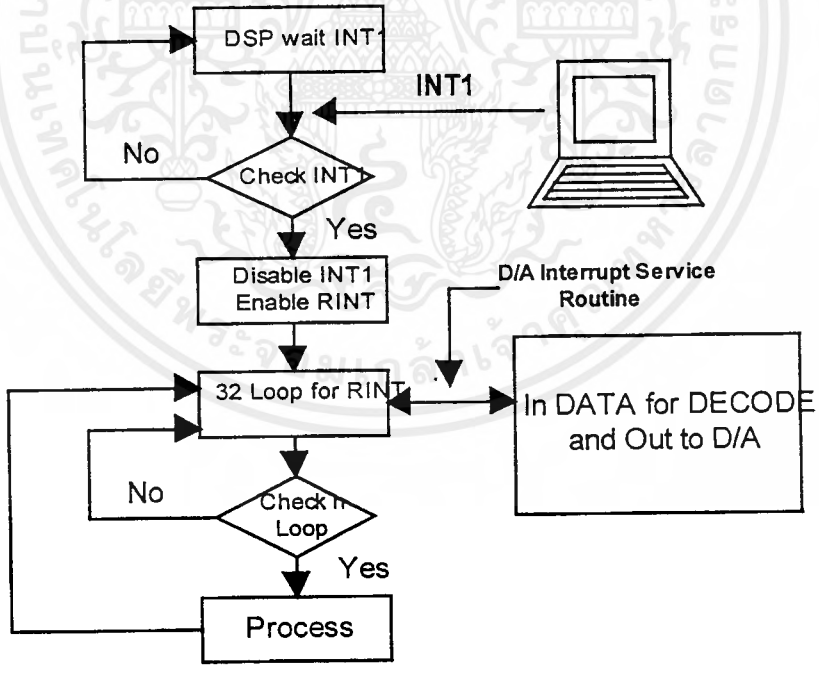
หรือ $f_{cLP} = \frac{62.5kHz}{TA}$ เมื่อ F_{MCLK} เท่ากับ 10MHz .....สมการที่ 8.3

ค่าความถี่คัทออฟ (ใช้สัญลักษณ์ F_s) สามารถหาได้จากสมการที่ 8.4

$$TA \times TB = \frac{f_{MCLK}}{2f_s}$$ .....สมการที่ 8.4

8.3 การทำงานร่วมกันระหว่างเครื่องพีซี (PC) และ ดีเอสพี

การใช้ดีเอสพีเพื่อประมวลผลข้อมูลแบบเอ็มเป็กโดยตรง จะเกิดข้อจำกัดในเรื่องหน่วยความจำของดีเอสพี เนื่องจากดีแอสพีมีหน่วยความจำเพียงแค่ 64 คำ (word) ซึ่ง 1 คำ = 2 ไบต์ (byte) แต่ส่วนโปรแกรมถอดรหัสที่เขียนด้วยภาษาแอสเซมบลีทั้งหมดมีความยาวมาก จึงเกิดข้อจำกัดที่ไม่สามารถประมวลผลโปรแกรมถอดรหัสไฟล์เอ็มเป็กโดยใช้ดีเอสพีเพียงอย่างเดียวได้ ในการทดลองจะให้ดีเอสพีประมวลผลโปรแกรมในส่วนของการทำโพลีเฟส ฟิลเตอร์ แบงค์ (Poly phase filter bank) เป็นต้นไป ดังนั้นระหว่างการประมวลโปรแกรม จำเป็นต้องส่งข้อมูลจากเครื่องคอมพิวเตอร์ ที่ผ่านการประมวลผลในส่วนของโอเวอร์แลบ (Overlapping) แล้วเข้าไปยังดีเอสพี และให้ดีเอสพีทำการประมวลผล แล้วจึงส่งข้อมูลเสียงออกมายังส่วนวงจรอนาลอกอินเตอร์เฟส เพื่อทำการแปลงเป็นสัญญาณเสียงส่งออกทางช่องจ่ายสัญญาณอนาลอก



รูปที่ 8.4 แสดงการทำงานร่วมกันระหว่างเครื่องพีซีและดีเอสพี

สามารถอธิบายการทำงานของระบบได้ดังนี้

ดีเอสพี เริ่มประมวลผล หยุดรอรับสัญญาณอินเตอร์รัป INT1

เมื่อสัญญาณ INT1 ส่งมาจากพีซี ดีเอสพี จะกระโดด ยังแอดเดสที่เวคเตอร์อินเตอร์รัป ชี้อยู่เพื่อทำให้เกิดสถานะไม่รับอินเตอร์รัป INT1 รับแต่อินเตอร์รัป RINT ของ วงจรอนาลอกอินเตอร์เฟส

เกิดการอินเตอร์รัปจากวงจรอนาลอกอินเตอร์เฟส 32 สัญญาณ เพื่อรับข้อมูลอินพุทจากพอร์ตขนานของเครื่องพีซี 64 ค่า โดยรับข้อมูลทีละ 1 ข้อมูล กระบวนการรับค่าดังนี้

1) รับค่าจากพอร์ตข้อมูล ขนาด 8 บิต เพื่อยกเลิกสถานะ BUSY ที่พอร์ตขนาน
2) จัดเรียงข้อมูล (ข้อมูลเริ่มเข้าเป็นข้อมูลลำดับที่ 0,1,2,..)
   - ถ้าเป็นข้อมูลลำดับคู่ จะต้องเก็บในหน่วยความจำ 8 ไบต์บน โดยนำข้อมูลที่ได้รับครั้งแรกเลื่อนไปข้างหน้า 8 บิต แล้วเก็บลงในหน่วยความจำตำแหน่งเดิม
   - ถ้าเป็นข้อมูลลำดับคู่ จะต้องเก็บในหน่วยความจำ 8 ไบต์บน โดยนำข้อมูลที่ได้รับครั้งแรกเลื่อนไปข้างหน้า 8 บิต แล้วเก็บลงในหน่วยความจำตำแหน่งเดิม
   - ส่งข้อมูลบางส่วนที่ประมวลผลเสร็จแล้วออกไปทาง วงจรอนาลอกอินเตอร์เฟส

พีซีส่งสัญญาณข้อมูลออกที่พอร์ตขนาน ทำให้เกิดสถานะ BUSY

พีซีส่งสัญญาณอินเตอร์รัป INT1 ออกที่พอร์ตขนาน และรอสถานะโดยการอ่านข้อมูลจากขา BUSY ของพอร์ตขนาน

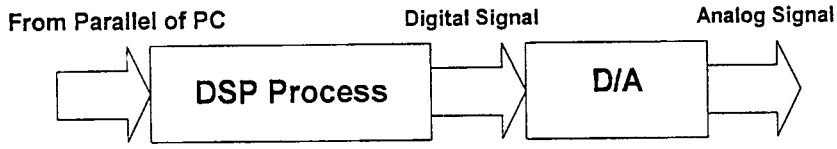
พีซี เริ่มส่งข้อมูลลำดับถัดไป จนครบ 64 ข้อมูล

ดีเอสพีรับข้อมูลครบ 32 ข้อมูล (จัดข้อมูล 64 เป็น 32) จะเริ่มประมวลผลโปรแกรมถอดรหัสข้อมูลส่วนโพลีเฟส ฟิลเตอร์ แบงค์

กลับไปรับค่าจากพีซีใหม่อีกครั้ง พร้อมส่งค่าออกไปยังอนาลอกอินเตอร์เฟส

8.4 โปรแกรมภาษาแอสเซมบลีสำหรับดีเอสพี

เป็นการนำโปรแกรมภาษาซีในส่วนของการทำโพลีเฟส (Polyphase) มาแปลงเป็นภาษาแอสเซมบลี (แสดงโปรแกรมไว้ในภาคผนวก) ในภาคนี้จะรับค่าอินพุทจาก PC ผ่านมาทางพอร์ตขนาน และประมวลผลโดย DSP จากนั้นจะส่งผ่าน D/A (Digital Signal to Analog Signal) ท้ายที่สุดจะได้สัญญาณอนาล็อก (Analog Signal) ซึ่งเป็นสัญญาณเสียงที่ถูกถอดรหัสเรียบร้อยแล้ว



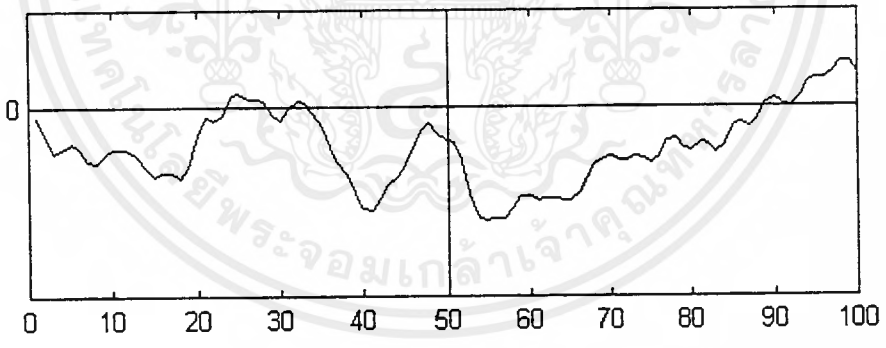
รูปที่ 8.5 แสดงถึงบล็อกไดอะแกรมการประมวลผลของ DSP

8.5 การทดลองและผลการทดลอง

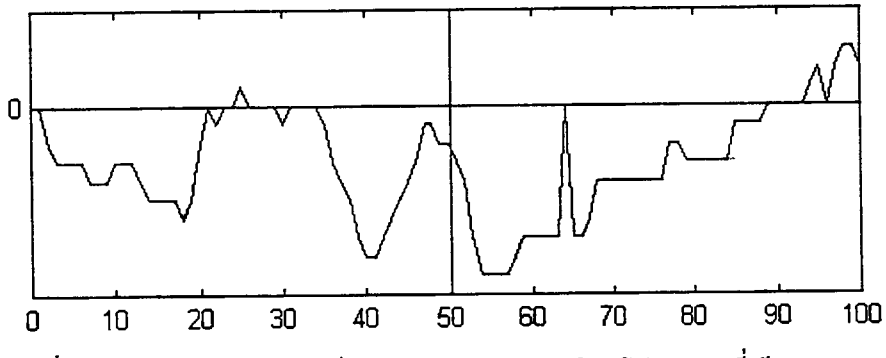
ในขั้นตอนนี้แบ่งการทดลองออกเป็น 2 แบบ คือ แบบที่ทำการถอดรหัสโดยโปรแกรมภาษาซีบนเครื่อง PC เท่านั้น และ แบบที่ 2 คือ แบบที่ใช้โปรแกรมภาษาซีบนเครื่อง PC ทำงานร่วมกับโปรแกรมภาษาแอสเซมบลีบน DSP

8.5.1 การถอดรหัสโดยใช้โปรแกรมภาษาซีบนเครื่องคอมพิวเตอร์

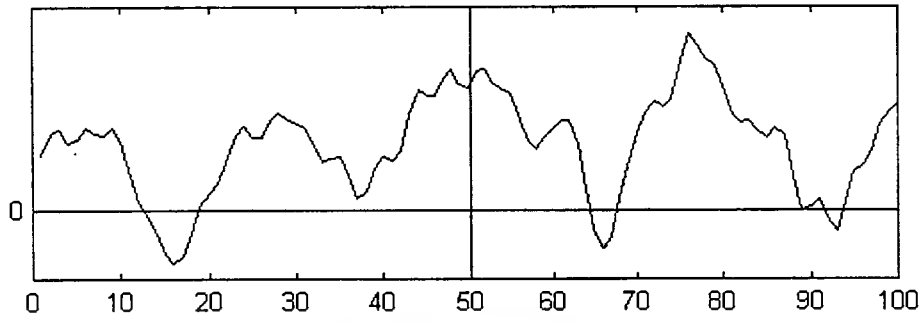
ผลการทดลองจะเป็นการเปรียบเทียบรูปสัญญาณเอาท์พุท จากการถอดรหัสโดยโปรแกรมถอดรหัสทั่วไป (เช่น Winamp Jetaudio) กับรูปสัญญาณเอาท์พุท จากการถอดรหัสโดยโปรแกรมที่เขียนเอง



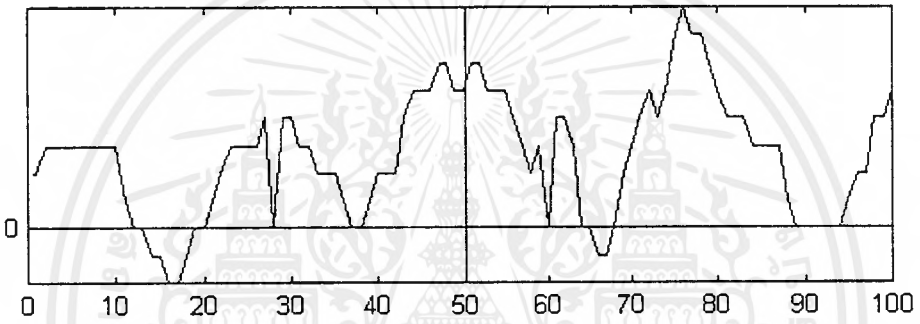
รูปที่ 8.6 (ก) รูปสัญญาณเอาท์พุทจากการถอดรหัสโดยโปรแกรมถอดรหัสทั่วไป (1)



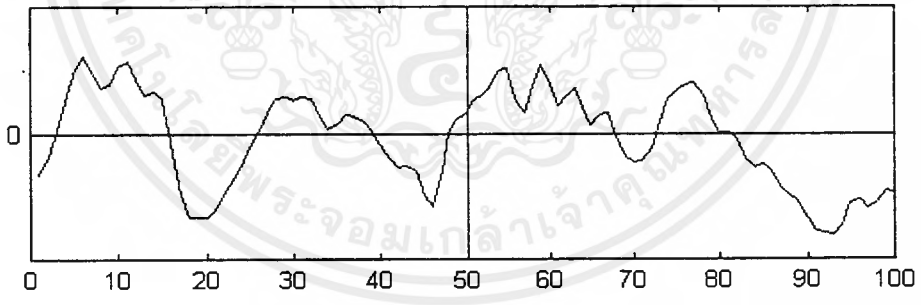
รูปที่ 8.6 (ข) รูปสัญญาณเอาท์พุทจากการถอดรหัสโดยโปรแกรมที่เขียนเอง (1)



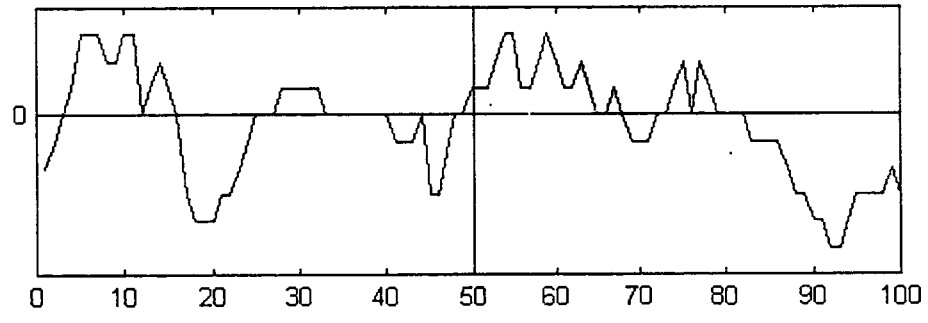
รูปที่ 8.7 (ก) รูปสัญญาณเอาท์พุทจากการถอดรหัสโดยโปรแกรมถอดรหัสทั่วไป (2)



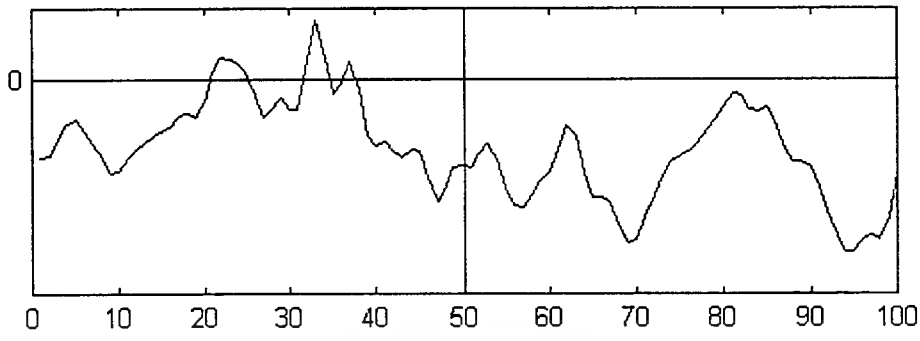
รูปที่ 8.7 (ข) รูปสัญญาณเอาท์พุทจากการถอดรหัสโดยโปรแกรมที่เขียนเอง (2)



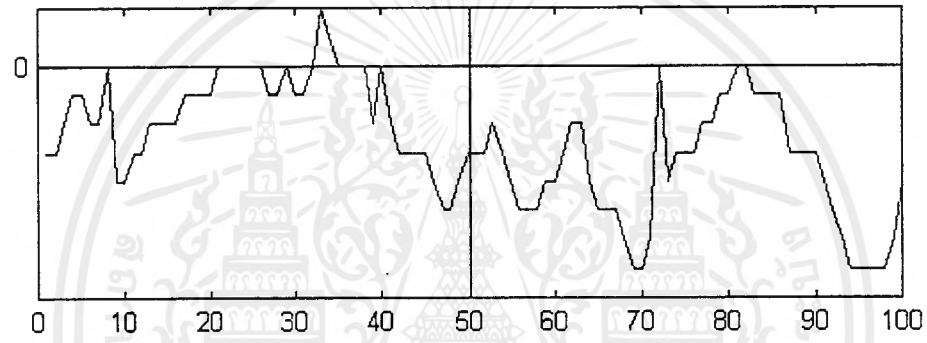
รูปที่ 8.8 (ก) รูปสัญญาณเอาท์พุทจากการถอดรหัสโดยโปรแกรมถอดรหัสทั่วไป (3)



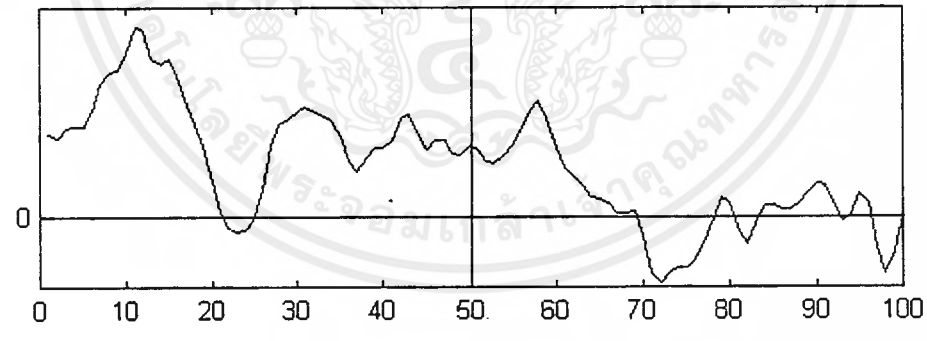
รูปที่ 8.8 (ข) รูปสัญญาณเอาท์พุทจากการถอดรหัสโดยโปรแกรมที่เขียนเอง (3)



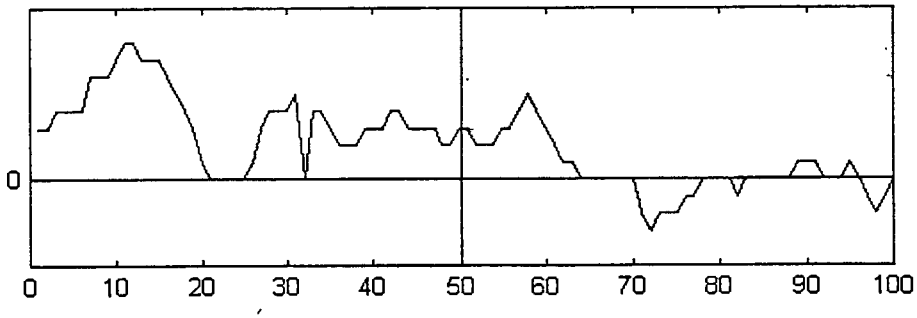
รูปที่ 8.9 (ก) รูปสัญญาณเอาท์พุทจากการถอดรหัสโดยโปรแกรมถอดรหัสทั่วไป (4)



รูปที่ 8.9 (ข) รูปสัญญาณเอาท์พุทจากการถอดรหัสโดยโปรแกรมที่เขียนเอง (4)



รูปที่ 8.10 (ก) รูปสัญญาณเอาท์พุทจากการถอดรหัสโดยโปรแกรมถอดรหัสทั่วไป (5)

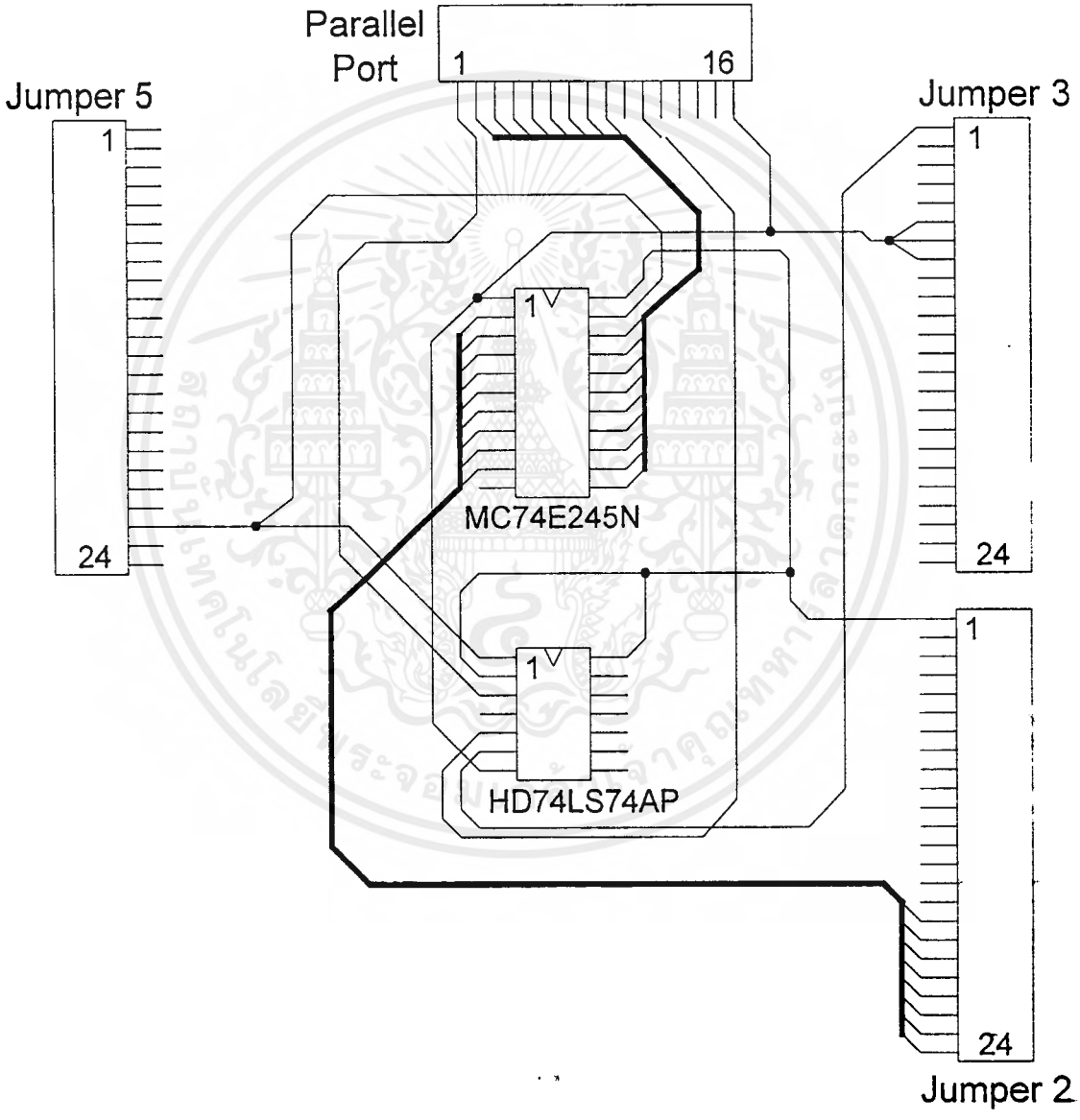


รูปที่ 8.10 (ข) รูปสัญญาณเอาท์พุทจากการถอดรหัสโดยโปรแกรมที่เขียนเอง (5)

จากรูปที่ 8.6 – 8.10 (ก) และ (ข) แสดงให้เห็นถึงความแตกต่างของสัญญาณทั้งสองอย่าง เห็นได้ชัดว่า โปรแกรมที่เขียนให้ความละเอียดที่ต่ำกว่า จึงทำให้เสียงที่ออกมาไม่คมชัดเท่า

8.5.2 การถอดรหัสโดยใช้โปรแกรมภาษาซีบนเครื่องคอมพิวเตอร์ร่วมกับดีเอสพี

การเชื่อมต่อดีเอสพีกับคอมพิวเตอร์จะเชื่อมผ่านพอร์ตขนาน จึงต้องต่อวงจรเพิ่มดังรูปที่ 8.11



รูปที่ 8.11 วงจรเชื่อมต่อดีเอสพีกับคอมพิวเตอร์ผ่านพอร์ตขนาน

การทดลองในส่วนนี้ค่าเอาท์พุทจะอยู่ในที่ที่ไม่แน่นอนในเมมโมรี ดังนั้นจึงใช้วิธีส่งค่าเข้าไปทีละชุดและสังเกตุค่าเอาท์พุทเทียบกับเอาท์พุทจากโปรแกรมภาษาซี ผลที่ได้คือทุกๆ 16 ค่าเอาท์พุทจะมีค่าผิด 1 ค่า ซึ่งก็คือมีเปอร์เซนต์ความถูกต้องเท่ากับ 93.75%

บทที่ 9
สรุปปัญหาและผลการดำเนินงาน

9.1 ความก้าวหน้าของการดำเนินงาน

สามารถรายงานความก้าวหน้าตามแผนการดำเนินที่กล่าวไว้ในบทที่ 1 ได้ดังนี้

ลำดับงาน	การดำเนินงาน	สถานะงาน
1	หาข้อมูลจากแหล่งต่างๆ	✓
2	ศึกษาการทำงานของเอ็มเป็กออดิโอ	✓
3	ศึกษาโปรแกรมในการถอดรหัสเอ็มเป็ก	✓
4	ปรับปรุงและแก้ไข โปรแกรม	✓
5	ศึกษาการใช้งานบอร์ด"DSP" ,การเชื่อมต่อกับคอมพิวเตอร์	✓
6	แปลงโปรแกรมภาษาซีเป็นภาษาแอสแซมบลีของ DSP	✓
7	ประมวลผลโปรแกรมถอดรหัสร่วมกันระหว่างเครื่องพีซี และดีเอสพี	✓
8	สรุปการดำเนินงานและวิเคราะห์ปัญหา	✓

9.2 ปัญหาจากการดำเนินงาน

การดำเนินงานที่ผ่านมาประสบปัญหาต่างๆทั้งในการดำเนินงานในส่วนของการเขียนโปรแกรมถอดรหัสไฟล์เอ็มเป็กเลเยอร์ 3 และในส่วนของการใช้ชิบดีเอสพีร่วมประมวลผลร่วมกับเครื่องพีซีดังต่อไปนี้

9.2.1 ปัญหาในการดำเนินงานในส่วนการเขียนโปรแกรมถอดรหัสเอ็มเป็ก

1. ขาดแหล่งข้อมูลที่เพียงพอในการดำเนินงานในช่วงแรก เนื่องจากไม่มีแหล่งข้อมูลเบื้องต้นที่ละเอียดเพียงพอ ทำให้ต้องใช้เวลาช่วงแรกเป็นเวลานานในการหาข้อมูล
2. ขาดความรู้ความเข้าใจในหลักการเขียนโปรแกรมภาษาซีที่ดีพอ เนื่องจากผู้ดำเนินการเพียงเริ่มฝึกเขียนโปรแกรมภาษาซี ทำให้ไม่สามารถใช้เทคนิคที่ดีมากนักในการเขียนโปรแกรม มีผลทำให้โปรแกรมที่เขียนขึ้นมามีประสิทธิภาพไม่ดีเท่าที่ควร
3. ขาดความรู้ ความเข้าใจในหลักการทำงานของการเข้ารหัสข้อมูลเอ็มเป็คเลเยอร์3 ที่เพียงพอ เนื่องจากต้องศึกษาหลักการทั้งหมดตั้งแต่เริ่มแรก ดังนั้นจึงประสบปัญหาเรื่อง

ความรู้ที่ไม่พอเพียงที่จะทำให้เข้าใจหลักการทำงานทั้งหมดของ การถอดรหัสข้อมูลเอ็มเป็ก

4. เนื่องจากดีเอสพีที่ใช้เป็นแบบฟิกส์พอย์ท (Fix Point DSP)คือคำนวนได้แต่จำนวนเต็ม แต่ข้อมูลที่ต้องการให้ดีเอสพีคำนวนเป็นแบบจำนวนจริง (float) จึงต้องทำการแปลงรูปแบบข้อมูลที่เป็นจำนวนจริงบนคอมพิวเตอร์ให้เป็นจำนวนเต็มเสียก่อน แล้วจึงส่งให้กับดีเอสพี จึงทำให้ข้อมูลที่ได้เปลี่ยนแปลงไปจากเดิม

9.2.2 ปัญหาในส่วนของการใช้ชิบดีเอสพีร่วมประมวลผลร่วมกับเครื่องพีซี

1. ขาดความรู้ความเข้าใจในหลักการทำงานและสถาปัตยกรรมของชิบดีเอสพีโดยถ่องแท้ ทำให้เกิดความล้าช้าในดำเนินงานมาก
2. จากผลของปัญหาในข้อที่ 1 ทำให้การใช้งานชิบดีเอสพีต้องใช้ชุดทดลองเท่านั้นจึงเกิดปัญหาตามมาในเรื่องข้อจำกัดของการใช้ชุดทดลองซึ่งอธิบายโดยสังเขปได้ดังนี้
   - การอ้างถึงหน่วยความจำไม่เป็นไปอย่างอิสระ เนื่องจากชุดทดลองจะใช้หน่วยความจำบางส่วนในการจัดค่าเริ่มต้นเพื่อประมวลผลโปรแกรม
   - ชุดทดลองจำเป็นต้องติดต่อกับเครื่องพีซีในขณะที่ดีบักโปรแกรม ดังนั้นจะเกิดปัญหาในการประมวลผลแบบเวลาจริง (Real time) เนื่องจากข้อมูลที่รับเข้าทางพอร์ตขนานจะขาดไปเป็นช่วงๆ ทำให้ข้อมูลที่ถอดรหัสเกิดความผิดพลาด
3. การติดต่อทางพอร์ตขนานถูกจำกัดด้วยเวลาการส่งที่ไม่สามารถส่งได้ด้วยความเร็วสูงมากนักทำให้ไม่สามารถการประมวลผลข้อมูลที่ถูกส่งมากจากพีซีที่มีอัตราการส่งข้อมูลสูงๆได้

9.3 สรุปผลการดำเนินงานทั้งหมด

เนื่องด้วยการดำเนินงานที่ผ่านมาในภาคการศึกษาที่ 1และภาคการศึกษาที่ 2 ประสบปัญหาต่างๆ ดังที่กล่าวมา ทำให้การดำเนินค่อนข้างจะติดขัด แต่ผู้ดำเนินการได้พยายามที่ให้การดำเนินการลุล่วงไปได้มากเท่าที่เป็นไปได้ ในที่นีขอสรุปผลการดำเนินงานในแต่ละขั้นตอนการดำเนินเพื่อให้เกิดความเข้าใจในลำดับการดำเนินงานตลอดถึงแนวทางการแก้ไขปัญหา เพื่อการพัฒนาในการเพิ่มประสิทธิภาพของการสร้างตัวถอดรหัสไฟล์เอ็มเป็กเลเยอร์ 3 ในอนาคตต่อไป

1. หาข้อมูลจากแหล่งต่างๆและศึกษาการทำงานของการเข้ารหัสไฟล์แบบเอ็มเป็กเลเยอร์ 3ผู้จัดทำได้ค้นคว้าหาข้อมูลเกี่ยวกับรูปแบบ,การเข้ารหัส,การถอดรหัส มาพอสมควร โดยแหล่งข้อมูลส่วนใหญ่จะอยู่ในอินเตอร์เน็ท และมาตรฐานอุตสาหกรรมของข้อมูล

แบบเอ็มเป็ก แต่เนื่องจากข้อมูลที่ได้มาอยู่ในรูปของหลักการและวิธีการมากกว่าเป็นรูปแแบบของทฤษฎี ส่งผลให้การทำความเข้าใจไม่สามารถทำได้อย่างถ่องแท้ และการแก้ไขปัญหาที่เกิดขึ้นเป็นไปอย่างยากลำบาก

2. การศึกษาการเขียนโปรแกรมแบบเอ็มเป็ก และแก้ไขโปรแกรมให้สามารถประมวลผลได้บนเครื่องพีซี ในส่วนนี้ผู้ดำเนินการได้ปรับปรุงและแก้ไขโปรแกรมถอดรหัสข้อมูลจนกระทั่งได้ข้อมูลเสียงที่ได้มีลักษณะ คล้ายข้อมูลเสียงที่ถูกถอดรหัสจากโปรแกรมถอดรหัสที่ได้รับความกันอยู่ในปัจจุบัน (โปรแกรม Winamp) แต่ก็ยังมีความผิดพลาดของข้อมูลอยู่ ซึ่งยังไม่สามารถสรุปหาสาเหตุของความผิดพลาดที่เกิดขึ้นอย่างแน่นอนได้
3. ศึกษาการทำงานและทำการส่งข้อมูลเชื่อมต่อระหว่างเครื่องพีซีและดีเอสพี ในส่วนนี้ได้ใช้เวลาการศึกษาและทดลองส่งข้อมูลจากเครื่องพีซี จนกระทั่งแน่ใจว่าข้อมูลที่ดีเอสพีรับได้ไม่มีความผิดพลาดเลย (เปอร์เซ็นความผิดพลาดเป็นศูนย์) เมื่อทดลองเฉพาะการรับข้อมูลจากเครื่องพีซี (ยังไม่ประมวลผล และส่งข้อมูลออก)
4. แปลงโปรแกรมภาษาซีเป็นภาษาแอสเซมบลีของ DSP ในส่วนนี้ดำเนินการเขียนโปรแกรมภาษาแอสเซมบลี ในส่วนของการถอดรหัสตั้งแต่ขั้นตอน โพลีเฟส ฟิลเตอร์ แบงค์ (Poly phase filter bank) เป็นต้นไป เนื่องจากข้อจำกัดด้านหน่วยความจำของชิบดีเอสพี หลายประการคือ
   - หน่วยความจำของดีเอสพีมีไม่เพียงพอ และยังถูกจำกัดการใช้หน่วยความจำโดยดีบักเกอร์ (Dcbuger) อีกด้วย
   - ต้องความเป็นเวลาจริง จึงไม่สามารถประมวลผลโปรแกรมที่มีความยาวสูงๆ ได้
   - ชิบดีเอสพีที่ใช้เป็นการอ้างและเข้าถึงข้อมูลแบบเลขจำนวนเต็ม ดังนั้นการประมวลผลโปรแกรม ที่มีความยาวและยุ่งยากจะทำให้ความผิดพลาดของข้อมูลเอาท์พุทมีเพิ่มมากขึ้น
   - ด้วยเหตุผลที่กล่าวมาทั้งสามประการนี้เป็นสาเหตุที่ตัดสินใจให้ดีเอสพีประมวลผลส่วนของโพลีเฟส ฟิลเตอร์ แบงค์ เพียงขั้นตอนเดียว
   - การเขียนโปรแกรม โพลีเฟส ฟิลเตอร์ แบงค์ เมื่อตรวจสอบกับข้อมูลที่ประมวลผลได้บนเครื่องพีซี พบว่ามีความผิดพลาดเพียง 1ข้อมูล ทุกๆข้อมูลอินพุท 32 ค่า (เปอร์เซนต์การผิดพลาด 3.125 เปอร์เซ็นต์)
5. ประมวลผลโปรแกรมถอดรหัสร่วมกันระหว่างเครื่องพีซีและดีเอสพี การทดลองในขั้นตอนนี้ได้ผลที่ไม่ค่อยดีเท่าที่ควร เนื่องจากปัญหาเรื่องความเร็วในการส่งสัญญาณ

และการถูกรบกวนการส่งข้อมูลโดยสัญญาณอินเตอร์รัปจากภายนอก (ยังสรุปหาสาเหตุ) การถูกรบกวนไม่ได้แน่นอนในขณะนี้ ทำให้ข้อมูลที่เข้ามาเกิดความผิดพลาด ส่งผลให้ข้อมูลเอาท์พุทเกิดความผิดพลาดเช่นเดียวกัน ดังนั้น งานการดำเนินงานในขั้นสุดท้ายอยู่ที่การใช้ดีเอสพีร่วมประมวลผลร่วมกับพีซี แต่ก็ยังไม่ได้ผลการทดลองที่มีประสิทธิภาพมากนัก

ภาคผนวก

ภาคผนวก ก

รูปแบบข้อมูลเสียงเอ็มเป็ก

(MPEG LAYER 3 FORMAT)

รูปแบบแฟ้มข้อมูลเอ็มเป็ก 3 (mpeg layer3 file format) อธิบายโดยใช้ไวยากรณ์(Syntex) ของภาษาซีเพื่อให้เห็นภาพพจน์ โดยใช้ตัวอักษรเข้มแสดงชื่อของข้อมูลในแต่ละส่วน เลขด้านท้ายของแต่ละบรรทัดแสดงความยาวบิตของข้อมูลนั้น ส่วนคำอธิบายละเอียดจะถูกยกไปอธิบายในหัวข้อ ก.2

ก.1 ไวยากรณ์ของข้อมูลเอ็มเป็ก (Codec audio bitstream syntax)

ก.1.1 ลำดับข้อมูลเสียง (Audio sequence)

```
Audio sequence()
{
while (nextbits()==syncword){
frame();
}
}
```

ก.1.2 เฟรมข้อมูลเสียง (Audio frame)

```
Frame()
{
    header();
    error_check();
    audio_data();
    ancillary_data();
}
```

ก.1.3 ส่วนหัวข้อมูล (Header)

```
Header(){
    synword                 12
    ID                      1
    layer                   2
    protection_bit          1
    bitrate_index           4
    sampling_frequency      2
    padding_bit             1
    private_bit             1
    mode                    2
    mode_extention          2
```

```
        copyright                                                   1
        original/copy                                               1
        emphasis                                                    2}
```

ก.1.4 ส่วนตรวจสอบความผิดพลาด (Error check)

```
Error_check()
{
        if(protection_bit==0)
                crc_check                                           16
}
```

ก.1.5 ส่วนข้อมูลเสียง(Audio data)

```
Audio_data()
{
        main_data_begin                                             9
        if(mode==single_channel)
                private_bits                                        5
        else
                private_bits                                        3
        for(ch=0;ch<nch;ch++)
                for(scfsi_band=0;scfi_band<4;scfi_band++)
                        scfsi[ch][scfsi_band]                       1
        for(gr=0;gr<2;gr++)
                for(ch=0;ch<nch;ch++){
                        part2_3_length[gr] [ch]                     2
                        big_values[gr] [ch]                         9
                        global_gain[gr] [ch]                        8
                        scalefac_compress[gr] [ch]                  4
                        window_switching_flag[gr] [ch]              1
                        if(window_switching_flag[gr] [ch]){
                                block_type[gr] [ch]                 2
                                mixed_block_flag[gr] [ch]           1
                                for (region=0;region<2;region++)
                                        table_select[gr] [ch] [region]  5
                                for (window=0;window<3;window++)
                                        subblock_gain[gr] [ch] [window] 3}
                        else {
                                for(region=0;region<3;region++)
                                        table_select [gr] [ch] [region] 5
                                        region0_count [gr] [ch]     4
                                        region1_count [gr] [ch]     3
                        }
                        preflag [gr] [ch]                           1
                        scalefac_scale [gr] [ch]                    1
                        count1table_select [gr] [ch]                1
                        }
                main_data()
}
```

ก.1.6 ส่วนข้อมูลหลัก (main data)

```
maindata()
{
for(gr=;gr<2;gr++) {
        for (ch=0;ch<nch;ch++){
```

```
        if ((window_switching_flag[gr][ch] ==1)&&(block_type[gr][ch]==2)){
                if (mixed_block_flag [gr][ch]){
                        for (sfb =0; sfb<8 ;sfb++)
                                scalefac_l[gr][ch][sfb]                    0..4
        for (sfb=3;sfb<12;sfb++)
                for (window = 0;windows<3;windows++)
                        scalefac_s[gr][ch][sfb][window]                    0..4}
        else {
        for(sfb=0;sfb<12;sfb++)
                for(window=0;window<3;window++)
                        scalefac_s[gr][ch][sfb][window]                    0..4
                }}
else{
        if((scfi[ch][0]==0 ||(gr==0))
                for(sfb=0;sfb<6;sfb++)
                        scalefac_l[gr][ch][sfb]                            0..4
        if((scfi[ch][1]==0 || (gr==0))
                for(sfb=6;sfb<11;sfb++)
                        scalefac_l[gr][ch][sfb]                            0..4
        if((scfi[ch][2]==0 ||(gr==0))
                for(sfb=11;sfb<16;sfb++)
                        scalefac_l[gr][ch][sfb]                            0..3
        if((scfi[ch][3]==0 || (gr==0))
                for(sfb=16;sfb<21;sfb++)
                        scalefac_l[gr][ch][sfb]                            0..3
                }
Huffmancodebit()
        }
}
for(b=0;b<no_of_ancillarybits;b++)
                ancillary_bit                                              1
}
```

ก.1.7 ส่วนการเข้ารหัสแบบฮัฟแมน

```
Huffmancodebits() {
        for( i=0; i < big_value *2 ; i+=2) {
                 hcod[|x|][|y|]                                            0..19
        if (|x|==15 &&linbits>0)
                linbitsx                                                   1..13
        if (x!=0)
                signx                                                      1
        if (|y|==15 &&linbits>0)
                linbitsy                                                   1..13
        if (y!=0)
                signy                                                      1
        is[i] = x
                is[i+l] = y
        }
        for(;i<big_values*2+countl*4;i+=4){
                hcod[|v|][|w|][|x|][|y|]                                   1..6
        if(v!=0)
                signv                                                      1
        if(w!=0)
                signw                                                      1
        if(x!=0)
                signx                                                      1
```

```
        if(y!=0)
                signy                                                   1
                is[l] = v
                is[l+1] = w
                is[l+2] = x
                is[l+3] = y
}
        for(;l<576;l++)
                is[l] = 0
}
```

ก.1.8 ข้อมูลช่วย(Ancillary data)

```
ancillary_data(){
        if ((layer==1)||(layer==2))
                                        for(b=0;b<no_of_ancillary_bits;b++)
                        ancillary_bit                                   1
}
```

ก.2 ความหมายของคำในไวยากรณ์ของข้อมูลเอ็มเป็กเลเยอร์ 3

ก.2.1 ลำดับสัญญาณเสียงทั่วๆไป

frame ส่วนของ ลำดับข้อมูลที่สามารถถอดรหัสได้ บรรจุข้อมูล 1152 สัญญาณสุ่มความถี่

ก.2.2 เฟรมของสัญญาณ (Audio Frame)

header ส่วนของลำดับข้อมูล(bitstream)ที่บรรจุข้อมูลเกี่ยวกับ การให้ความสัมพันธ์ (Synchronization) และข้อมูลทั่วไป

error_check ส่วนของลำดับข้อมูลที่บรรจุข้อมูลเกี่ยวกับการหาข้อผิดพลาด

audio_data ส่วนของลำดับข้อมูลที่บรรจุข้อมูลเกี่ยวกับข้อมูลเสียงที่สุ่มตัวอย่าง

ancillary_data ส่วนของลำดับข้อมูลที่บรรจุข้อมูลเพิ่มเติม

ก.2.3 ส่วนหัวของข้อมูล (Header)

32 บิทแรกของลำดับข้อมูล บรรจุข้อมูลที่บอกถึงลักษณะทั่วๆไปทั้งหมดของการเข้ารหัส

syncword ส่วนส่วนหัวของลำดับข้อมูลเอ็มเป็ก เท่ากับ “1111 1111 1111”

ID บิทแสดงถึงมาตรฐานการเข้ารหัส “1” หมายถึงเข้ารหัสตามมาตรฐาน ISO 11172-3 “0” ไม่ใช่มาตรฐาน ISO

layer แสดงเลเยอร์ ที่กำลังถอดรหัสอยู่ ดังตารางที่ ก.1

ตารางที่ ก.1 ความหมายของรหัสข้อมูลในเลเยอร์

เลเยอร์	ความหมาย
"11"	เลเยอร์ 1
"10"	เลเยอร์ 2
"01"	เลเยอร์ 3
"00"	ไม่ใช้

protect_bit บอกให้ทราบว่า ข้อมูลเสริมเกี่ยวกับการตรวจสอบความผิดพลาดได้บันทึกมาด้วยหรือไม่โดย

"1" ไม่มีข้อมูลเสริม

"0" มีข้อมูลเสริม

bitrate_index บอกอัตราการบีบอัดข้อมูล (bitrate) ที่ใช้

sapling_frequncy บอกความถี่ในการสุ่มตัวอย่าง (sampling)

padding_bit เท่ากับ 1 แสดงว่าเฟรม ของข้อมูลนั้นๆ บรรจุสล็อด(slot) ใน layer 3 = 1 byte เพิ่มเติมมา ถ้าเท่ากับ "0" แสดงว่าไม่ได้บรรจุ การบรรจุสล็อด เพื่อปรับอัตราส่วนระหว่างอัตราส่วนการบีบอัด กับความถี่ในการสุ่มตัวอย่างsampling frequncy ให้ลงตัว ในกรณี ความถี่สุ่มตัวอย่าง 44.1 กิโลเฮิร์ต ไม่มี Padding คือเท่ากับ 0 เสมอ

private ไม่ใช้ในการเข้ารหัสตามมาตรฐาน ISO/IEC 11172-3

copy right "1" ป้องกันการ copy "0" ไม่ป้องกันการ copy

original/copy "0" ข้อมูลนั้นถูก copy มา, "1" ข้อมูลนั้น เป็นต้นฉบับ

mode บอกเกี่ยวกับรูปแแบบของข้อมูล ว่าเป็น หนึ่งช่องเสียง (single channel) , สองช่องเสียง(double channel) , สเตอริโอ(stereo) , จอย-สเตอริโอ (joint to stereo) , ดังตารางที่ ก.2

ตารางที่ ก.2 ความหมายของรหัสข้อมูลใน Mode

Mode	ความหมาย
"00"	สเตอริโอ
"01"	จอย-สเตอริโอ
"10"	สองช่องเสียง
"11"	หนึ่งช่องเสียง

mode_extention ใช้บอกชนิดของจอย-สเตอริโอ(joint to stereo mode) ว่า มี ms –stereo ,intensity-stereo หรือไม่อย่างไร ดังตารางที่ ก.3

ตารางที่ ก.3 ความหมายของรหัสข้อมูลใน Mode_extention

Mode_extention	Intensity	Ms_sterio
"00"	Off	Off
"01"	On	Off
"10"	Off	On
"11"	On	On

emphasis แสดงถึงชนิดของเอ็มพาไซส์ (emphasis)ที่ใช้งาน แสดงในตารางที่ ก.4

ตารางที่ ก.4 ความหมายของรหัสข้อมูลใน Emphasis

Emphasis	ความหมาย
"00"	ไม่มีใช้
"01"	50/15 ไมโครวินาที
"10"	สงวน
"11"	CCITT J.17

ก.2.4 ส่วนตรวจสอบความผิดพลาด

Crc check ข้อมูล 16 บิต เพื่อตรวจสอบ parity ของข้อมูล ว่าถูกต้องหรือไม่

ก.2.5 ส่วนข้อมูลเสียง

Main_data_begin แสดงตำแหน่งแรกของข้อมูลหลักของกรอบนั้นๆ โดยเป็นค่าออฟเซ็ท (offset byte) เป็นค่าตำแหน่งที่ห่างออกมาจากไบต์แรกของซิงค์เวิร์ด (syncword)โดยไม่รวมจำนวนไบต์ ของส่วนหัว(Header) และ ไซด์ อินฟรอมเมชั่น (side information)

Private_bits จะไม่ใช้ในมาตรฐาน ISO/IEC จำนวนของไปรเวต์บิท(private bit) ขึ้นอยู่กับจำนวนช่องเสียง,จำนวนของไปรเวต์บิท ถูกใช้พิจารณาเพื่อเทียบกับจำนวนของ bit ทั้งหมด ในไซด์ อินฟรอมเมชั่น (side information)

scfsi[ch][scfsi_band] ในเข้ารหัสแบบเอ็มเป็กเลเยอร์ 3 scfsi (scalefactor selection information) ให้ข้อมูลเกี่ยวกับค่า สเกลเฟคเตอร์ (scale factor) ของแต่ละย่านความถี่ (subband) และแต่ละช่องเสียง (ch) ค่า scfsi_band ถูกใช้เพื่อเลือกกลุ่มของสเกลเฟคเตอร์ การใช้งานของสเกลเฟคเตอร์ ในแต่ละแกรนนูล (granule) จะถูกควบคุมโดย scfsi ตารางที่ ก.5 แสดงความหมายของ Scfsi[scfsi_band]

ตารางที่ ก.5 ความหมายของรหัสข้อมูลใน Emphasis

Scfsi[scfsi_band]	
"0"	เลือกใช้แต่ละ granule แยกกัน
"1"	ทั้ง 2 เลือกใช้ scale factor เดียวกัน

scfsi_band ควบคุมการใช้ scfsi สำหรับกลุ่มของสเกลเฟคเตอร์ แสดงความหมายในตาราง ก.6

ตารางที่ ก.6 ความหมายของรหัสข้อมูลใน Scfsi_band

Scfsi_band	ย่านความถี่ที่ถูกใช้งาน
0	0..5
1	6..10
2	11..15
3	16..20

part2_3_length[gr][ch] บอกจำนวนของ bit ของข้อมูลในส่วนที่เข้ารหัสแบบฮัฟแมน (Huffman code) และสเกลเฟคเตอร์เพื่อใช้หาตำแหน่ง เริ่มต้นของ **main information** สำหรับแกรนนูล (granule) ถัดไป

big _values[gr][ch] ค่าที่ถูกเข้ารหัสโดยใช้ รหัสฮัฟแมน (Huffman code) โดยอ้างต่าจากตารางฮัฟแมน (Huffman code table)

global_gain[gr][ch] เป็นตัวแปรที่ใช้ในขั้นตอนการรีควอนไตซ์ จะกล่าวถึงอีกครั้งในเรื่องการรีควอนไตซ์ในบทที่ 5

scalefac_compress[gr][ch] เลือกจำนวนของบิทที่ถูกใช้สำหรับ ส่งค่า สเกลเฟคเตอร์(scalefactor) ดังตารางที่ ก.7

ตารางที่ ก.7 ความหมายของรหัสข้อมูลใน scalefac_compress[gr][ch].

scalefac_compress[gr][ch	Slen1	Slen
0	0	0
1	0	1
2	0	2

scalefac_compress[gr][ch	Slen1	Slen
3	0	3
4	3	0
5	1	1
6	1	2
7	1	3
8	2	1
9	2	2
10	2	3
11	3	1
12	3	2
13	3	3
14	4	2
15	4	3

block_type 0,1,3

slen1 : เป็นความยาวของ scale factor ย่าน 0 ถึง 10

slen2 : เป็นความยาวของ scale factor ย่าน 11 ถึง 20

block_type 2 และ mixed_block_flag เป็น 0

slen1 : เป็นความยาวของ scale factor ย่าน 0 ถึง 5

slen2 : เป็นความยาวของ scale factor ย่าน 6 ถึง 11

block_type 2 และ mixed_block_flag เป็น 1

slen1 : เป็นความยาวของ scale factor ย่าน 0 ถึง 7(ย่าน long window)
เป็นความยาวของ scale factor ย่าน 3 ถึง 5(ย่าน short window)

slen2 : เป็นความยาวของ scale factor ย่าน 6 ถึง 11

window_switching_flag[gr][ch] แสดงว่าข้อมูลไม่ได้ใช้วินโดว์(window) ปกติ(type=0)

ในกรณีที่ window_switching_flag = 1 จะมีผลดังนี้

region0_count = 7 กรณี **block_type** ==1 หรือ **block_type** ==3
หรือ **block_type** ==2 และ mix_block_flag

region0_count = 8 **block_type** ==2 และ not mix_block_flag

region_count = 36 ค่าทั้งหมดของ **big_value** อยู่ใน **region 1**

ถ้า window_switching_flag = 0แล้วค่า **block_type = 0**

block_type[gr][ch] ชี้ ชนิดของ window ที่ใช้ของ granule นั้นแสดงความหมายใน ตารางที่ ก.8

block_type และ mix_clock_flag ให้ข้อมูลเกี่ยวกับค่าที่อยู่ในบล็อค เกี่ยวกับความยาว, และการนับสำหรับการแปลง ถ้า window_switching_flag == 1 mix_block_flag จะเป็นตัวชี้ว่า lower frequency poly phase ย่านใดถูกถอดรหัส โดยใช้ type ปกติ

ในกรณีบล็อคยาว (block_type ไม่ใช่ 2 , หรือ ย่านต่ำที่ block type เป็น 2 เมื่อmix block flag = 1) IMDCT จะให้เอาท์พุท 36 ค่าทุกๆอินพุท 18 ค่า เอาท์พุทจะขึ้นอยู่กับ block_type

ในกรณีบล็อคสั้น (ย่านที่เหนือกว่าบล็อกยาว ของ block_type 2 เมื่อ mix block flag = 1 หรือทุกย่านความถี่ของ block_type 2 เมื่อ mix blck flag = 0) IMDCT ให้เอาท์พุท 12 ค่า

mix_block_flag[gr][ch] เป็นตัวกำหนดการแปลงที่ความถี่ต่ำว่าใช้ block_type แบบใด

- กรณี **mix_block_flag** เป็น 0 : ทุก block ถูกแปลงค่าโดยชี้จาก block_type[gr][ch]
- กรณี **mix_block_flag** เป็น 1: ย่านความถี่ต่ำสุด 2 ย่านถูกกำหนดให้แปลงด้วย window ปกติ 30 ย่านความถี่ที่เหลือ ถูกแปลงค่าโดยชี้จาก block_type[gr][ch]

table_select[gr][ch][region] ใช้เลือกตารางฮัฟแมน (Huffman) จาก 32 ตาราง

subblock_gain[gr][ch][window] บอกค่าอัตราการขยายที่เพิ่มขึ้น/ลดลงจาก grobal_gain ของแต่ละ บล็อคย่อย(subblock)

ตารางที่ ก.8 ความหมายของรหัสข้อมูลใน Block_type[gr]

Block_type[gr]	
0	ไม่ใช้
1	เริ่มต้นบล็อค
2	วินโดว์บล็อคสั้นสามบล็อก
3	ท้ายบล็อค

region0_count[gr][ch] ส่วนของสเป็คตรัม(Spectrum) ที่ถูกแบ่งเพื่อใช้เพิ่มความสามารถของตัวเข้ารหัสให้มีความถูกต้องยิ่งขึ้น โดยแต่ละส่วนที่ถูกแบ่งจะถูกเรียกว่า region 0,1,2 แต่ละ region จะใช้ ตารางถอดรหัสฮัฟแมน (Huffman Table) ต่างกันขึ้นอยู่กับ maximun quntized value และ siganl statistic ค่าของ region0_count และ region1_count ถูกใช้ชี้ขอบเขตของ region นั้นๆ

region1_count[gr][ch] นับจำนวนของสเกลแฟคเตอร์ใน region1 ลบออก 1

preflag[gr][ch] เป็นค่าเสริมเพื่อขยายค่าที่ความถี่สูง ถ้าค่า preflag ถูกตั้ง จะนำค่าในตาราง preflag ไปคูณกับค่าของ scale factor อีกครั้ง ในกรณี block type เท่ากับ 2 preflag ไม่ถูกใช้

scalefac_scale[gr][ch] เป็นสเกลแฟคเตอร์ (scale factor) ที่ถูกปรับค่า (quantize) ด้วยสเกล ล็อกการิทึม โดยคูณค่าของแต่ละลำดับด้วย 2 หรือ $\sqrt{2}$ ขึ้นอยู่กับ scalefac_scale ดังแสดงความหมายในตารางที่ ก.9

ตารางที่ ก.9 ความหมายของรหัสข้อมูลใน Scalfac_scale[gr]

Scalfac_scale[gr]	Scalefac_multipier
0	0,5
1	1

count1table_select[gr][ch] ใช้เลือกค่าจากตารางฮัฟแมน (Huffman) B7.A หรือ B7.B เมื่อค่าที่ปรับระดับ(quantize) ใน region คูณด้วย 4 แล้วไม่เกิน 1 ดังแสดงความหมายของตารางที่ ก.10

ตารางที่ ก.10 ความหมายของรหัสข้อมูลใน Count1table_select[gr][ch]

Count1table_select[gr][ch]	
0	ตาราง B7.A
1	ตาราง B7.B

scalefac_l[gr][ch], scalefac_s[gr][ch][sfb][window], is_pos[sfb] ถูกใช้ในการปรับระดับค่าคืน(requantization)

huffmancodedbits() ข้อมูลที่ถูกเข้ารหัสฮัฟแมน(Huffman)

รูปแบบของ Huffmancodecbit() แสดงวิธีการเข้ารหัส ข้อมูลลงใน big_value โดยข้อมูลที่ได้จะเป็นคู่ (x,y) ให้ค่าแต่ละค่ามีจำนวนบิท น้อยกว่า 15 บิต การเข้ารหัสจะถูกเลือกจากตาราง Huffman 0 ถึง 31 ถ้าข้อมูลที่เข้ามามีขนาดเกิน 15 บิต จะแยกเข้ารหัสต่างหาก ถ้าในคู่ของข้อมูลที่เข้ารหัสไม่เป็น 0 จะปรากฏเครื่องหมาย ให้ค่า + หรือ –

ในตาราง huffman จะบรรจุ องค์ประกอบ 3 ส่วนคือ

- hcod [x] [y] รหัสข้อมูล huffman
- hlen [x] [y] ความยาวข้อมูล huffman
- linbits ความยาวของ linbitsX หรือ linbitsY เมื่อถูกเข้ารหัส

รูปแบบของส่วนประกอบ ของ huffmancodebits ประกอบด้วยกลุ่มของข้อมูลประกอบดังนี้

- **sig nv** เครื่องหมายของค่า v (0 เป็นบวก 1 เป็นลบ)
- **sig nw** เครื่องหมายของค่า w (0 เป็นบวก 1 เป็นลบ)
- **sig nx** เครื่องหมายของค่า x (0 เป็นบวก 1 เป็นลบ)
- **sig ny** เครื่องหมายของค่า y (0 เป็นบวก 1 เป็นลบ)
- **litbitsX** ใช้เมื่อเข้ารหัสถ้าขนาดของ x เกิน 15
- **litbitY** ใช้เมื่อเข้ารหัสถ้าขนาดของ x เกิน 15
- **is [l]** คือ ค่าที่ถูก quatized สำหรับ frequency line ที่1

ก.2.6. Ancillary data

ancillary_bit ผู้ใช้สามารถนิยามเอง

ภาคผนวก ข

โปรแกรมถอดรหัสเอ็มเป็ก(ภาษาซี)

```
#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#include <mem.h>

#include <math.h>

#include "tab_mono4.h"

#include "var_mono6.h"

#include "fun_mono4.h"

#define GETHDR_ERR  0x1
#define GETHDR_NS   0x2
#define GETHDR_FL1  0x4
#define GETHDR_FL2  0x8
#define GETHDR_FF   0x10
#define GETHDR_SYN  0x20
#define GETHDR_EOF  0x30
#define BUFFER_SIZE  4096
#define BUFFER_AUX  2048
#define TRUE                    1
#define FALSE                   0
#define MAJOR                   0
#define MINOR                   7
#define PATCH                   6
#define MAX(a,b)                ((a) > (b) ? (a) : (b))
#define MAX3(a,b,c)     ((a) > (b) ? MAX(a, c) : MAX(b, c))
#define MIN(a,b)                ((a) < (b) ? (a) : (b))
#define N_CUE                   16
#define NC_O                    4
#define PI12    0.261799387f
#define PI36    0.087266462f
#define i_sq2                   0.707106781188
#define IS_ILLEGAL      0xfeed
unsigned char  buffer[BUFFER_SIZE+BUFFER_AUX],_buffer[80];
unsigned char  filename[80]="d:\\hleang\\test_mp3\\amp_mono.mp3",savefile[80]="d:\\hleang\\test_pcm\\amp_mono6.pcm";
FILE *in_file;
FILE *out_file;
int scalefac_l[2][2][22];
int scalefac_s[2][2][13][3];
int append,_bptr;
int is[578];
int non_zero; /* this is 2*bigvalues+4*count1, i guess...*/
float s[32][18];
float res[32][18];
float win[4][36];
int data;
int *t_l,*t_s;
static float t_43[30];
float xr[32][18];
short sample_mono[32];
struct SIDE_INFO{
        int main_data_begin;
        int scfsi[2][4];
        int part2_3_length[2][2];
        int big_values[2][2];
        int global_gain[2][2];
        int scalefac_compress[2][2];
        int window_switching_flag[2][2];
```

```
        int block_type[2][2];
        int mixed_block_flag[2][2];
        int table_select[2][2][3];
        int subblock_gain[2][2][3];
        int region0_count[2][2];
        int region1_count[2][2];
        int preflag[2][2];
        int scalefac_scale[2][2];
        int count1table_select[2][2];
};
struct AUDIO_HEADER {
        int id;
        int layer;
        int protection_bit;
        int bitrate_index;
        int sampling_frequency;
        int padding_bit;
        int private_bit;
        int mode;
        int mode_extension;
        int copyright;
        int original;
        int emphasis;
};

int t_sampling_frequency[2][3] = {
{ 22050 , 24000 , 16000},
{ 44100 , 48000 , 32000}
};
short t_bitrate[2][3][15] = {{
{0,32,48,56,64,80,96,112,128,144,160,176,192,224,256},
{0,8,16,24,32,40,48,56,64,80,96,112,128,144,160},
{0,8,16,24,32,40,48,56,64,80,96,112,128,144,160}
},{
{0,32,64,96,128,160,192,224,256,288,320,352,384,416,448},
{0,32,48,56,64,80,96,112,128,160,192,224,256,320,384},
{0,32,40,48,56,64,80,96,112,128,160,192,224,256,320}
}};
int t_b8_l[2][3][22]={{ /* table B.8b ISO/IEC 11172-3 long block*/
{5,11,17,23,29,35,43,53,65,79,95,115,139,167,199,237,283,335,395,463,521,575},
{5,11,17,23,29,35,43,53,65,79,95,113,135,161,193,231,277,331,393,463,539,575},
{5,11,17,23,29,35,43,53,65,79,95,115,139,167,199,237,283,335,395,463,521,575}
},{
{3,7,11,15,19,23,29,35,43,51,61,73,89,109,133,161,195,237,287,341,417,575}, // case 44.1
{3,7,11,15,19,23,29,35,41,49,59,71,87,105,127,155,189,229,275,329,383,575}, // case 48
{3,7,11,15,19,23,29,35,43,53,65,81,101,125,155,193,239,295,363,447,549,575} //case 32
}};
int t_b8_s[2][3][13]={{ /* table B.8b ISO/IEC 11172-3 short block*/
 // don't have id
{3,7,11,17,23,31,41,55,73,99,131,173,191},
{3,7,11,17,25,35,47,61,79,103,135,179,191},
{3,7,11,17,25,35,47,61,79,103,133,173,191}
},{
// have id
{3,7,11,15,21,29,39,51,65,83,105,135,191}, // case 44.1
{3,7,11,15,21,27,37,49,63,79,99,125,191},  // case 48
{3,7,11,15,21,29,41,57,77,103,137,179,191} // case 32
}};

static char t_slen1[16]={0,0,0,0,3,1,1,1,2,2,2,3,3,3,4,4}; // slen = lenght of scale factor
static char t_slen2[16]={0,1,2,3,0,1,2,3,1,2,3,1,2,3,2,3};
/*----------------------------Huffman Table----------------------------*/
int t_linbits[32]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,2,3,4,6,8,10,13,4,5,6,7,8,9,11,13};

unsigned int h0[1]={0x0};
unsigned int h1[4]={0x311, 0x20000301, 0x40000210, 0x80000100};
unsigned int h2[9]={0x622, 0x4000602, 0x8000512, 0x10000521, 0x18000520, 0x20000311, 0x40000301, 0x60000310,
 0x80000100};
unsigned int h3[9]={ 0x622, 0x4000602, 0x8000512, 0x10000521, 0x18000520, 0x20000310, 0x40000211, 0x80000201,
 0xc0000200};
unsigned int h5[16]={0x833, 0x1000823, 0x2000732, 0x4000631, 0x8000713, 0xa000703, 0xc000730, 0xe000722,
 0x10000612, 0x14000621, 0x18000602, 0x1c000620, 0x20000311, 0x40000301, 0x60000310, 0x80000100};
```

```
unsigned int h6[16]={0x733, 0x2000703, 0x4000623, 0x8000632, 0xc000630, 0x10000513, 0x18000531, 0x20000522,
 0x28000502, 0x30000412, 0x40000421, 0x50000420, 0x60000301, 0x80000211, 0xc0000310, 0xe0000300};
unsigned int h7[36]={ 0xa55, 0x400a45, 0x800a54, 0xc00a53, 0x1000935, 0x1800944, 0x2000925, 0x2800952,
 0x3000815, 0x4000851, 0x5000905, 0x5800934, 0x6000850, 0x7000943, 0x7800933, 0x8000824,
 0x9000842, 0xa000714, 0xc000741, 0xe000740, 0x10000804, 0x11000823, 0x12000832, 0x13000803,
 0x14000713, 0x16000731, 0x18000730, 0x1a000722, 0x1c000612, 0x20000521, 0x28000602, 0x2c000620,
 0x30000411, 0x40000301, 0x60000310, 0x80000100};
unsigned int h8[36]={0xb55, 0x200b54, 0x400a45, 0x800953, 0x1000a35, 0x1400a44, 0x1800925, 0x2000952,
 0x2800905, 0x3000815, 0x4000851, 0x5000934, 0x5800943, 0x6000950, 0x6800933, 0x7000824,
 0x8000842, 0x9000814, 0xa000741, 0xc000804, 0xd000840, 0xe000823, 0xf000832, 0x10000813,
 0x11000831, 0x12000803, 0x13000830, 0x14000622, 0x18000602, 0x1c000620, 0x20000412, 0x30000421,
 0x40000211, 0x80000301, 0xa0000310, 0xc0000200};
unsigned int h9[36]={ 0x955, 0x800945, 0x1000835, 0x2000853, 0x3000954, 0x3800905, 0x4000844, 0x5000825,
 0x6000852, 0x7000815, 0x8000751, 0xa000734, 0xc000743, 0xe000850, 0xf000804, 0x10000724,
 0x12000742, 0x14000733, 0x16000740, 0x18000614, 0x1c000641, 0x20000623, 0x24000632, 0x28000513,
 0x30000531, 0x38000603, 0x3c000630, 0x40000522, 0x48000502, 0x50000412, 0x60000421, 0x70000420,
 0x80000311, 0xa0000301, 0xc0000310, 0xe0000300};
unsigned int h10[64]={ 0xb77, 0x200b67, 0x400b76, 0x600b57, 0x800b75, 0xa00b66, 0xc00a47, 0x1000a74,
 0x1400a56, 0x1800a65, 0x1c00a37, 0x2000a73, 0x2400a46, 0x2800b55, 0x2a00b54, 0x2c00a63,
 0x3000927, 0x3800972, 0x4000a64, 0x4400a07, 0x4800970, 0x5000962, 0x5800a45, 0x5c00a35,
 0x6000906, 0x6800a53, 0x6c00a44, 0x7000817, 0x8000871, 0x9000936, 0x9800926, 0xa000a25,
 0xa400a52, 0xa800915, 0xb000951, 0xb800a34, 0xbc00a43, 0xc000816, 0xd000861, 0xe000860,
 0xf000905, 0xf800950, 0x10000924, 0x10800942, 0x11000933, 0x11800904, 0x12000814, 0x13000841,
 0x14000840, 0x15000823, 0x16000832, 0x17000803, 0x18000713, 0x1a000731, 0x1c000730, 0x1e000722,
 0x20000612, 0x24000621, 0x28000602, 0x2c000620, 0x30000411, 0x40000301, 0x60000310, 0x80000100};
unsigned int h11[64]={ 0xa77, 0x400a67, 0x800a76, 0xc00a75, 0x1000a66, 0x1400a47, 0x1800a74, 0x1c00b57,
 0x1e00b55, 0x2000a56, 0x2400a65, 0x2800937, 0x3000973, 0x3800946, 0x4000a45, 0x4400a54,
 0x4800a35, 0x4c00a53, 0x5000827, 0x6000872, 0x7000964, 0x7800907, 0x8000771, 0xa000817,
 0xb000870, 0xc000836, 0xd000863, 0xe000860, 0xf000944, 0xf800925, 0x10000952, 0x10800905,
 0x11000815, 0x12000762, 0x14000826, 0x15000806, 0x16000716, 0x18000761, 0x1a000851, 0x1b000834,
 0x1c000850, 0x1d000943, 0x1d800933, 0x1e000824, 0x1f000842, 0x20000814, 0x21000841, 0x22000804,
 0x23000840, 0x24000723, 0x26000732, 0x28000613, 0x2c000631, 0x30000703, 0x32000730, 0x34000622,
 0x38000521, 0x40000412, 0x50000502, 0x58000520, 0x60000311, 0x80000301, 0xa0000310, 0xc0000200};
unsigned int h12[64]={ 0xa77, 0x400a67, 0x800976, 0x1000957, 0x1800975, 0x2000966, 0x2800947, 0x3000974,
 0x3800965, 0x4000856, 0x5000837, 0x6000973, 0x6800955, 0x7000827, 0x8000872, 0x9000846,
 0xa000864, 0xb000817, 0xc000871, 0xd000907, 0xd800970, 0xe000836, 0xf000863, 0x10000845,
 0x11000854, 0x12000844, 0x13000906, 0x13800905, 0x14000726, 0x16000762, 0x18000761, 0x1a000816,
 0x1b000860, 0x1c000835, 0x1d000853, 0x1e000825, 0x1f000852, 0x20000715, 0x22000751, 0x24000734,
 0x26000743, 0x28000850, 0x29000804, 0x2a000724, 0x2c000742, 0x2e000714, 0x30000633, 0x34000641,
 0x38000623, 0x3c000632, 0x40000740, 0x42000703, 0x44000630, 0x48000513, 0x50000531, 0x58000522,
 0x60000412, 0x70000421, 0x80000502, 0x88000520, 0x90000400, 0xa0000311, 0xc0000301, 0xe0000310};
unsigned int h13[256]={
 0x13fe, 0x33fc, 0x52fd, 0x91ed, 0x110ff, 0x210ef, 0x310df, 0x410ee,
 0x510cf, 0x610de, 0x710bf, 0x810fb, 0x910ce, 0xa10dc, 0xb11af, 0xb91e9,
 0xc0fec, 0xe0fdd, 0x1010fa, 0x1110cd, 0x120fbe, 0x140feb, 0x160f9f, 0x180ff9,
 0x1a0fea, 0x1c0fbd, 0x1e0fdb, 0x200f8f, 0x220ff8, 0x240fcc, 0x2610ae, 0x27109e,
 0x280f8e, 0x2a107f, 0x2b107e, 0x2c0ef7, 0x300eda, 0x340fad, 0x360fbc, 0x380fcb,
 0x3a0ff6, 0x3c0e6f, 0x400ee8, 0x440e5f, 0x480e9d, 0x4c0ed9, 0x500ef5, 0x540ee7,
 0x580eac, 0x5c0ebb, 0x600e4f, 0x640ef4, 0x680fca, 0x6a0fe6, 0x6c0ef3, 0x700d3f,
 0x780e8d, 0x7c0ed8, 0x800d2f, 0x880df2, 0x900e6e, 0x940e9c, 0x980d0f, 0xa00ec9,
 0xa40e5e, 0xa80dab, 0xb00e7d, 0xb40ed7, 0xb80d4e, 0xc00ec8, 0xc40ed6, 0xc80d3e,
 0xd00db9, 0xd80e9b, 0xdc0eaa, 0xe00c1f, 0xf00cf1, 0x1000cf0, 0x1100dba, 0x1180de5,
 0x1200de4, 0x1280d8c, 0x1300d6d, 0x1380de3, 0x1400ce2, 0x1500d2e, 0x1580d0e, 0x1600c1e,
 0x1700ce1, 0x1800de0, 0x1880d5d, 0x1900dd5, 0x1980d7c, 0x1a00dc7, 0x1a80d4d, 0x1b00d8b,
 0x1b80db8, 0x1c00dd4, 0x1c80d9a, 0x1d00da9, 0x1d80d6c, 0x1e00cc6, 0x1f00c3d, 0x2000dd3,
 0x2080d7b, 0x2100c2d, 0x2200cd2, 0x2300c1d, 0x2400cb7, 0x2500d5c, 0x2580dc5, 0x2600d99,
 0x2680d7a, 0x2700cc3, 0x2800da7, 0x2880d97, 0x2900c4b, 0x2a00bd1, 0x2c00c0d, 0x2d00cd0,
 0x2e00c8a, 0x2f00ca8, 0x3000c4c, 0x3100cc4, 0x3200c6b, 0x3300cb6, 0x3400b3c, 0x3600b2c,
 0x3800bc2, 0x3a00b5b, 0x3c00cb5, 0x3d00c89, 0x3e00b1c, 0x4000bc1, 0x4200c98, 0x4300c0c,
 0x4400bc0, 0x4600cb4, 0x4700c6a, 0x4800ca6, 0x4900c79, 0x4a00b3b, 0x4c00bb3, 0x4e00c88,
 0x4f00c5a, 0x5000b2b, 0x5200ca5, 0x5300c69, 0x5400ba4, 0x5600c78, 0x5700c87, 0x5800b94,
 0x5a00c77, 0x5b00c76, 0x5c00ab2, 0x6000a1b, 0x6400ab1, 0x6800b0b, 0x6a00bb0, 0x6c00b96,
 0x6e00b4a, 0x7000b3a, 0x7200ba3, 0x7400b59, 0x7600b95, 0x7800a2a, 0x7c00aa2, 0x8000a1a,
 0x8400aa1, 0x8800b0a, 0x8a00b68, 0x8c00aa0, 0x9000b86, 0x9200b49, 0x9400a93, 0x9800b39,
 0x9a00b58, 0x9c00b85, 0x9e00b67, 0xa000a29, 0xa400a92, 0xa800b57, 0xaa00b75, 0xac00a38,
 0xb000a83, 0xb400b66, 0xb600b47, 0xb800b74, 0xba00b56, 0xbc00b65, 0xbe00b73, 0xc000919,
 0xc800991, 0xd000a09, 0xd400a90, 0xd800a48, 0xdc00a84, 0xe000a72, 0xe400b46, 0xe600b64,
 0xe800928, 0xf000982, 0xf800918, 0x10000a37, 0x10400a27, 0x10800917, 0x11000971, 0x11800a55,
 0x11c00a07, 0x12000a70, 0x12400a36, 0x12800a63, 0x12c00a45, 0x13000a54, 0x13400a26, 0x13800a62,
 0x13c00a35, 0x14000881, 0x15000908, 0x15800980, 0x16000916, 0x16800961, 0x17000906, 0x17800960,
 0x18000a53, 0x18400a44, 0x18800925, 0x19000952, 0x19800905, 0x1a000815, 0x1b000851, 0x1c000934,
 0x1c800943, 0x1d000950, 0x1d800924, 0x1e000942, 0x1e800933, 0x1f000814, 0x20000741, 0x22000804,
```

0x23000840, 0x24000823, 0x25000832, 0x26000713, 0x28000731, 0x2a000703, 0x2c000730, 0x2e000722,
0x30000612, 0x34000621, 0x38000602, 0x3c000620, 0x40000411, 0x50000401, 0x60000310, 0x80000100};
unsigned int h15[256]={ 0xdff, 0x80def, 0x100dfe, 0x180ddf, 0x200cee, 0x300dfd, 0x380dcf, 0x400dfc,
0x480dde, 0x500ded, 0x580dbf, 0x600cfb, 0x700dce, 0x780dec, 0x800cdd, 0x900caf,
0xa00cfa, 0xb00cbe, 0xc00ceb, 0xd00ccd, 0xe00cdc, 0xf00c9f, 0x1000cf9, 0x1100cea,
0x1200cbd, 0x1300cdb, 0x1400c8f, 0x1500cf8, 0x1600ccc, 0x1700c9e, 0x1800ce9, 0x1900c7f,
0x1a00cf7, 0x1b00cad, 0x1c00cda, 0x1d00cbc, 0x1e00c6f, 0x1f00dae, 0x1f80d0f, 0x2000bcb,
0x2200bf6, 0x2400c8e, 0x2500ce8, 0x2600c5f, 0x2700c9d, 0x2800bf5, 0x2a00b7e, 0x2c00be7,
0x2e00bac, 0x3000bca, 0x3200bbb, 0x3400cd9, 0x3500c8d, 0x3600b4f, 0x3800bf4, 0x3a00b3f,
0x3c00bf3, 0x3e00bd8, 0x4000be6, 0x4200b2f, 0x4400bf2, 0x4600c6e, 0x4700cf0, 0x4800b1f,
0x4a00bf1, 0x4c00b9c, 0x4e00bc9, 0x5000b5e, 0x5200bab, 0x5400bba, 0x5600be5, 0x5800b7d,
0x5a00bd7, 0x5c00b4e, 0x5e00be4, 0x6000b8c, 0x6200bc8, 0x6400b3e, 0x6600b6d, 0x6800bd6,
0x6a00be3, 0x6c00b9b, 0x6e00bb9, 0x7000b2e, 0x7200baa, 0x7400be2, 0x7600b1e, 0x7800be1,
0x7a00c0e, 0x7b00ce0, 0x7c00b5d, 0x7e00bd5, 0x8000b7c, 0x8200bc7, 0x8400b4d, 0x8600b8b,
0x8800ad4, 0x8c00bb8, 0x8e00b9a, 0x9000ba9, 0x9200b6c, 0x9400bc6, 0x9600b3d, 0x9800ad3,
0x9c00ad2, 0xa000b2d, 0xa200b0d, 0xa400a1d, 0xa800a7b, 0xac00ab7, 0xb000ad1, 0xb400b5c,
0xb600bd0, 0xb800ac5, 0xbc00a8a, 0xc000aa8, 0xc400a4c, 0xc800ac4, 0xcc00a6b, 0xd000ab6,
0xd400b99, 0xd600b0c, 0xd800a3c, 0xdc00ac3, 0xe000a7a, 0xe400aa7, 0xe800aa6, 0xec00bc0,
0xee00b0b, 0xf0009c2, 0xf800a2c, 0xfc00a5b, 0x10000ab5, 0x10400a1c, 0x10800a89, 0x10c00a98,
0x11000ac1, 0x11400a4b, 0x11800ab4, 0x11c00a6a, 0x12000a3b, 0x12400a79, 0x128009b3, 0x13000a97,
0x13400a88, 0x13800a2b, 0x13c00a5a, 0x140009b2, 0x14800aa5, 0x14c00a1b, 0x150009b1, 0x15800ab0,
0x15c00a69, 0x16000a96, 0x16400a4a, 0x16800aa4, 0x16c00a78, 0x17000a87, 0x17400a3a, 0x178009a3,
0x18000959, 0x18800995, 0x1900092a, 0x198009a2, 0x1a00091a, 0x1a8009a1, 0x1b000a0a, 0x1b400aa0,
0x1b800968, 0x1c000986, 0x1c800949, 0x1d000994, 0x1d800939, 0x1e000993, 0x1e800a77, 0x1ec00a09,
0x1f000958, 0x1f800985, 0x20000929, 0x20800967, 0x21000976, 0x21800992, 0x22000891, 0x23000919,
0x23800990, 0x24000948, 0x24800984, 0x25000957, 0x25800975, 0x26000938, 0x26800983, 0x27000966,
0x27800947, 0x28000828, 0x29000882, 0x2a000818, 0x2b000881, 0x2c000974, 0x2c800908, 0x2d000980,
0x2d800956, 0x2e000965, 0x2e800937, 0x2f000973, 0x2f800946, 0x30000827, 0x31000872, 0x32000864,
0x33000817, 0x34000855, 0x35000871, 0x36000907, 0x36800970, 0x37000836, 0x38000863, 0x39000845,
0x3a000854, 0x3b000826, 0x3c000862, 0x3d000816, 0x3e000906, 0x3e800960, 0x3f000835, 0x40000761,
0x42000853, 0x43000844, 0x44000725, 0x46000752, 0x48000715, 0x4a000751, 0x4c000805, 0x4d000850,
0x4e000734, 0x50000743, 0x52000724, 0x54000742, 0x56000733, 0x58000641, 0x5c000714, 0x5e000704,
0x60000623, 0x64000632, 0x68000740, 0x6a000703, 0x6c000613, 0x70000631, 0x74000630, 0x78000522,
0x80000512, 0x88000521, 0x90000502, 0x98000520, 0xa0000311, 0xc0000401, 0xd0000410, 0xe0000300};
unsigned int h16[256]={ 0xbef, 0x200bfe, 0x400bdf, 0x600bfd, 0x800bcf, 0xa00bfc, 0xc00bbf, 0xe00bfb,
0x1000aaf, 0x1400bfa, 0x1600b9f, 0x1800bf9, 0x1a00bf8, 0x1c00a8f, 0x2000a7f, 0x2400af7,
0x2800a6f, 0x2c00af6, 0x30008ff, 0x4000a5f, 0x4400af5, 0x480094f, 0x50009f4, 0x58009f3,
0x60009f0, 0x6800a3f, 0x6c010ce, 0x6c111ec, 0x6c191dd, 0x6c20fde, 0x6c40fe9, 0x6c610ea,
0x6c710d9, 0x6c80eee, 0x6cc0fed, 0x6ce0feb, 0x6d00ebe, 0x6d40ecd, 0x6d80fdc, 0x6da0fdb,
0x6dc0eae, 0x6e00ecc, 0x6e40fad, 0x6e60fda, 0x6e80f7e, 0x6ea0fac, 0x6ec0eca, 0x6f00fc9,
0x6f20f7d, 0x6f40e5e, 0x6f80dbd, 0x70008f2, 0x800092f, 0x880090f, 0x900081f, 0xa0008f1,
0xb000d9e, 0xb080ebc, 0xb0c0ecb, 0xb100e8e, 0xb140ee8, 0xb180e9d, 0xb1c0ee7, 0xb200ebb,
0xb240e8d, 0xb280ed8, 0xb2c0e6e, 0xb300de6, 0xb380d9c, 0xb400eab, 0xb440eba, 0xb480ee5,
0xb4c0ed7, 0xb500d4e, 0xb580ee4, 0xb5c0e8c, 0xb600dc8, 0xb680d3e, 0xb700d6d, 0xb780ed6,
0xb7c0e9b, 0xb800eb9, 0xb840eaa, 0xb880de1, 0xb900dd4, 0xb980eb8, 0xb9c0ea9, 0xba00d7b,
0xba80eb7, 0xbac0ed0, 0xbb00ce3, 0xbc00d0e, 0xbc80de0, 0xbd00d5d, 0xbd80dd5, 0xbe00d7c,
0xbe80dc7, 0xbf00d4d, 0xbf80d8b, 0xc000d9a, 0xc080d6c, 0xc100dc6, 0xc180d3d, 0xc200d5c,
0xc280dc5, 0xc300c0d, 0xc400d8a, 0xc480da8, 0xc500d99, 0xc580d4c, 0xc600db6, 0xc680d7a,
0xc700c3c, 0xc800d5b, 0xc880d89, 0xc900c1c, 0xca00cc0, 0xcb00d98, 0xcbb80d79, 0xcc00be2,
0xce00c2e, 0xcf00c1e, 0xd000cd3, 0xd100c2d, 0xd200cd2, 0xd300cd1, 0xd400c3b, 0xd500d97,
0xd580d88, 0xd600b1d, 0xd800cc4, 0xd900c6b, 0xda00cc3, 0xdb00ca7, 0xdc00b2c, 0xde00cc2,
0xdf00cb5, 0xe000cc1, 0xe100c0c, 0xe200c4b, 0xe300cb4, 0xe400c6a, 0xe500ca6, 0xe600bb3,
0xe800c5a, 0xe900ca5, 0xea00b2b, 0xec00bb2, 0xee00b1b, 0xf000bb1, 0xf200c0b, 0xf300cb0,
0xf400c69, 0xf500c96, 0xf600c4a, 0xf700ca4, 0xf800c78, 0xf900c87, 0xfa00ba3, 0xfc00c3a,
0xfd00c59, 0xfe00b2a, 0x10000c95, 0x10100c68, 0x10200ba1, 0x10400c86, 0x10500c77, 0x10600b94,
0x10800c49, 0x10900c57, 0x10a00b67, 0x10c00aa2, 0x11000a1a, 0x11400b0a, 0x11600ba0, 0x11800b39,
0x11a00b93, 0x11c00b58, 0x11e00b85, 0x12000a29, 0x12400a92, 0x12800b76, 0x12a00b09, 0x12c00a19,
0x13000a91, 0x13400b90, 0x13600b48, 0x13800b84, 0x13a00b75, 0x13c00b38, 0x13e00b83, 0x14000b66,
0x14200b28, 0x14400a82, 0x14800b47, 0x14a00b74, 0x14c00a18, 0x15000a81, 0x15400a80, 0x15800b08,
0x15a00b56, 0x15c00a37, 0x16000a73, 0x16400b65, 0x16600b46, 0x16800a27, 0x16c00a72, 0x17000b64,
0x17200b55, 0x17400a07, 0x17800917, 0x18000971, 0x18800a70, 0x18c00a36, 0x19000a63, 0x19400a45,
0x19800a54, 0x19c00a26, 0x1a000962, 0x1a800916, 0x1b000961, 0x1b800a06, 0x1bc00a60, 0x1c000953,
0x1c800a35, 0x1cc00a44, 0x1d000925, 0x1d800952, 0x1e000851, 0x1f000915, 0x1f800905, 0x20000934,
0x20800943, 0x21000950, 0x21800924, 0x22000942, 0x22800933, 0x23000814, 0x24000841, 0x25000904,
0x25800940, 0x26000823, 0x27000832, 0x28000713, 0x2a000731, 0x2c000803, 0x2d000830, 0x2e000722,
0x30000612, 0x34000621, 0x38000602, 0x3c000620, 0x40000411, 0x50000401, 0x60000310, 0x80000100};
unsigned int h24[256]={ 0x8ef, 0x10008fe, 0x20008df, 0x30008fd, 0x40008cf, 0x50008fc, 0x60008bf, 0x70008fb,
0x80007fa, 0xa0008af, 0xb00089f, 0xc0007f9, 0xe0007f8, 0x1000088f, 0x1100087f, 0x120007f7,
0x1400076f, 0x160007f6, 0x1800075f, 0x1a0007f5, 0x1c00074f, 0x1e0007f4, 0x2000073f, 0x220007f3,
0x2400072f, 0x260007f2, 0x280007f1, 0x2a00081f, 0x2b0008f0, 0x2c00090f, 0x2c800bee, 0x2ca00bde,
0x2cc00bed, 0x2ce00bce, 0x2d000bec, 0x2d200bdd, 0x2d400bbe, 0x2d600beb, 0x2d800bcd, 0x2da00bdc,
0x2dc00bae, 0x2de00bea, 0x2e000bbd, 0x2e200bdb, 0x2e400bcc, 0x2e600b9e, 0x2e800be9, 0x2ea00bad,

```
0x2ec00bda, 0x2ee00bbc, 0x2f000bcb, 0x2f200b8e, 0x2f400be8, 0x2f600b9d, 0x2f800bd9, 0x2fa00b7e,
0x2fc00be7, 0x2fe00bac, 0x300004ff, 0x40000bca, 0x40200bbb, 0x40400b8d, 0x40600bd8, 0x40800c0e,
0x40900ce0, 0x40a00b0d, 0x40c00ae6, 0x41000b6e, 0x41200b9c, 0x41400ac9, 0x41800a5e, 0x41c00aba,
0x42000ae5, 0x42400bab, 0x42600b7d, 0x42800ad7, 0x42c00ae4, 0x43000a8c, 0x43400ac8, 0x43800b4e,
0x43a00b2e, 0x43c00a3e, 0x44000a6d, 0x44400ad6, 0x44800ae3, 0x44c00a9b, 0x45000ab9, 0x45400aaa,
0x45800ae2, 0x45c00a1e, 0x46000ae1, 0x46400a5d, 0x46800ad5, 0x46c00a7c, 0x47000ac7, 0x47400a4d,
0x47800a8b, 0x47c00ab8, 0x48000ad4, 0x48400a9a, 0x48800aa9, 0x48c00a6c, 0x49000ac6, 0x49400a3d,
0x49800ad3, 0x49c00a2d, 0x4a000ad2, 0x4a400a1d, 0x4a800a7b, 0x4ac00ab7, 0x4b000ad1, 0x4b400a5c,
0x4b800ac5, 0x4bc00a8a, 0x4c000aa8, 0x4c400a99, 0x4c800a4c, 0x4cc00ac4, 0x4d000a6b, 0x4d400ab6,
0x4d800bd0, 0x4da00b0c, 0x4dc00a3c, 0x4e000ac3, 0x4e400a7a, 0x4e800aa7, 0x4ec00a2c, 0x4f000ac2,
0x4f400a5b, 0x4f800ab5, 0x4fc00a1c, 0x50000a89, 0x50400a98, 0x50800ac1, 0x50c00a4b, 0x51000bc0,
0x51200b0b, 0x51400a3b, 0x51800bb0, 0x51a00b0a, 0x51c00a1a, 0x520009b4, 0x52800a6a, 0x52c00aa6,
0x53000a79, 0x53400a97, 0x53800ba0, 0x53a00b09, 0x53c00a90, 0x540009b3, 0x54800988, 0x55000a2b,
0x55400a5a, 0x558009b2, 0x56000aa5, 0x56400a1b, 0x56800ab1, 0x56c00a69, 0x57000996, 0x578009a4,
0x58000a4a, 0x58400a78, 0x58800987, 0x5900093a, 0x598009a3, 0x5a000959, 0x5a800995, 0x5b00092a,
 0x5b8009a2, 0x5c0009a1, 0x5c800968, 0x5d000986, 0x5d800977, 0x5e000949, 0x5e800994, 0x5f000939,
0x5f800993, 0x60000958, 0x60800985, 0x61000929, 0x61800967, 0x62000976, 0x62800992, 0x63000919,
0x63800991, 0x64000948, 0x64800984, 0x65000957, 0x65800975, 0x66000938, 0x66800983, 0x67000966,
0x67800928, 0x68000982, 0x68800918, 0x69000947, 0x69800974, 0x6a000981, 0x6a800a08, 0x6ac00a80,
0x6b000956, 0x6b800965, 0x6c000917, 0x6c800a07, 0x6cc00a70, 0x6d000873, 0x6e000937, 0x6e800927,
0x6f000872, 0x70000846, 0x71000864, 0x72000855, 0x73000871, 0x74000836, 0x75000863, 0x76000845,
0x77000854, 0x78000826, 0x79000862, 0x7a000816, 0x7b000861, 0x7c000906, 0x7c800960, 0x7d000835,
0x7e000853, 0x7f000844, 0x80000825, 0x81000852, 0x82000815, 0x83000905, 0x83800950, 0x84000751,
 0x86000834, 0x87000843, 0x88000724, 0x8a000742, 0x8c000733, 0x8e000714, 0x90000741, 0x92000804,
 0x93000840, 0x94000723, 0x96000732, 0x98000613, 0x9c000631, 0xa0000703, 0xa2000730, 0xa4000622,
 0xa8000512, 0xb0000521, 0xb8000602, 0xbc000620, 0xc0000411, 0xd0000401, 0xe0000410, 0xf0000400};
unsigned int hA[16]={ 0x6b0, 0x40006f0, 0x80006d0, 0xc0006e0, 0x10000670, 0x14000650, 0x18000590, 0x20000560,
 0x28000530, 0x300005a0, 0x380005c0, 0x40000420, 0x50000410, 0x60000440, 0x70000480, 0x80000100};
unsigned int hB[16]={  0x4f0, 0x100004e0, 0x200004d0, 0x300004c0, 0x400004b0, 0x500004a0, 0x60000490, 0x70000480,
 0x80000470, 0x90000460, 0xa0000450, 0xb0000440, 0xc0000430, 0xd0000420, 0xe0000410, 0xf0000400};

/* now the cues, remember to change these tables if you change N_CUE
*/
unsigned char h_cue[34][N_CUE]={                    //N_CUE=16
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,1,1,2,2,2,2,3,3,3,3,3,3,3,3},
{0,3,5,5,6,6,7,7,8,8,8,8,8,8,8,8},
{0,3,5,5,6,6,6,6,7,7,7,7,7,8,8,8},
{0,8,12,12,13,13,14,14,15,15,15,15,15,15,15,15},
{0,8,12,12,13,13,14,14,15,15,15,15,15,15,15,15},
{0,5,7,9,10,11,12,12,13,13,13,13,14,14,15,15},
{0,20,29,32,33,33,34,34,35,35,35,35,35,35,35,35},
{0,23,30,31,32,32,32,32,33,33,34,34,35,35,35,35},
{0,15,21,24,27,29,30,31,32,32,33,33,34,34,35,35},
{0,42,56,60,61,61,62,62,63,63,63,63,63,63,63,63},
{0,30,45,53,57,58,60,60,61,61,62,62,63,63,63,63},
{0,23,37,46,50,54,56,57,58,60,61,61,62,62,63,63},
{0,203,238,248,252,253,254,254,255,255,255,255,255,255,255,255},
{0,132,178,205,223,233,240,245,248,250,252,252,253,254,255,255},
{0,132,178,205,223,233,240,245,248,250,252,252,253,254,255,255},
{0,162,231,248,252,253,254,254,255,255,255,255,255,255,255,255},
{0,162,231,248,252,253,254,254,255,255,255,255,255,255,255,255},
{0,162,231,248,252,253,254,254,255,255,255,255,255,255,255,255},
{0,162,231,248,252,253,254,254,255,255,255,255,255,255,255,255},
{0,162,231,248,252,253,254,254,255,255,255,255,255,255,255,255},
{0,162,231,248,252,253,254,254,255,255,255,255,255,255,255,255},
{0,162,231,248,252,253,254,254,255,255,255,255,255,255,255,255},
{0,13,22,58,59,131,177,209,226,238,245,249,252,253,254,255},
{0,13,22,58,59,131,177,209,226,238,245,249,252,253,254,255},
{0,13,22,58,59,131,177,209,226,238,245,249,252,253,254,255},
{0,13,22,58,59,131,177,209,226,238,245,249,252,253,254,255},
{0,13,22,58,59,131,177,209,226,238,245,249,252,253,254,255},
{0,13,22,58,59,131,177,209,226,238,245,249,252,253,254,255},
{0,13,22,58,59,131,177,209,226,238,245,249,252,253,254,255},
{0,13,22,58,59,131,177,209,226,238,245,249,252,253,254,255},
{0,4,7,9,11,12,13,14,15,15,15,15,15,15,15,15},
{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15}
};

/*-------------Choose Table-------------*/
unsigned int *tables[34]={h0,h1,h2,h3,h5,h5,h6,h7,h8,h9,h10,h11,h12,h13,h15,h15,
```

```
        h16,h16,h16,h16,h16,h16,h16,h16,h24,h24,h24,h24,h24,h24,h24,h24,hA,hB};
static const int t_pretab[22]={0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,2,2,3,3,3,2,0};
static const float Cs[8]={0.857492925712, 0.881741997318, 0.949628649103,
        0.983314592492, 0.995517816065, 0.999160558175,
        0.999899195243, 0.999993155067};
static const float Ca[8]={-0.5144957554270, -0.4717319685650, -0.3133774542040,
        -0.1819131996110, -0.0945741925262, -0.0409655828852,
        -0.0141985685725,  -0.00369997467375};
static const float tab[4]={1,1.189207115,1.414213562,1.6817928301};
static const float tabi[4]={1,0.840896415,0.707106781,0.594603557};
static const short t_reorder[2][3][576]={{
{ 0,  1,  2,  3,  6,  7,  8,  9, 12, 13, 14, 15,  4,  5, 18, 19, 10, 11, 24, 25,
 16, 17, 30, 31, 20, 21, 22, 23, 26, 27, 28, 29, 32, 33, 34, 35, 36, 37, 38, 39,
 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59,
 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 90, 91,
 78, 79, 80, 81, 82, 83, 96, 97, 84, 85, 86, 87, 88, 89,102,103, 92, 93, 94, 95,
108,109,110,111,112,113, 98, 99,100,101,114,115,116,117,118,119,104,105,106,107,
120,121,122,123,124,125,126,127,128,129,130,131,144,145,146,147,148,149,162,163,
132,133,134,135,136,137,150,151,152,153,154,155,168,169,138,139,140,141,142,143,
156,157,158,159,160,161,174,175,164,165,166,167,180,181,182,183,184,185,198,199,
200,201,202,203,216,217,170,171,172,173,186,187,188,189,190,191,204,205,206,207,
208,209,222,223,176,177,178,179,192,193,194,195,196,197,210,211,212,213,214,215,
228,229,218,219,220,221,234,235,236,237,238,239,252,253,254,255,256,257,270,271,
272,273,274,275,288,289,290,291,224,225,226,227,240,241,242,243,244,245,258,259,
260,261,262,263,276,277,278,279,280,281,294,295,296,297,230,231,232,233,246,247,
248,249,250,251,264,265,266,267,268,269,282,283,284,285,286,287,300,301,302,303,
292,293,306,307,308,309,310,311,324,325,326,327,328,329,342,343,344,345,346,347,
360,361,362,363,364,365,378,379,380,381,382,383,298,299,312,313,314,315,316,317,
330,331,332,333,334,335,348,349,350,351,352,353,366,367,368,369,370,371,384,385,
386,387,388,389,304,305,318,319,320,321,322,323,336,337,338,339,340,341,354,355,
356,357,358,359,372,373,374,375,376,377,390,391,392,393,394,395,396,397,398,399,
400,401,414,415,416,417,418,419,432,433,434,435,436,437,450,451,452,453,454,455,
468,469,470,471,472,473,486,487,488,489,490,491,504,505,506,507,508,509,402,403,
404,405,406,407,420,421,422,423,424,425,438,439,440,441,442,443,456,457,458,459,
460,461,474,475,476,477,478,479,492,493,494,495,496,497,510,511,512,513,514,515,
408,409,410,411,412,413,426,427,428,429,430,431,444,445,446,447,448,449,462,463,
464,465,466,467,480,481,482,483,484,485,498,499,500,501,502,503,516,517,518,519,
520,521,522,523,524,525,526,527,540,541,542,543,544,545,558,559,560,561,562,563,
528,529,530,531,532,533,546,547,548,549,550,551,564,565,566,567,568,569,534,535,
536,537,538,539,552,553,554,555,556,557,570,571,572,573,574,575},

{ 0,  1,  2,  3,  6,  7,  8,  9, 12, 13, 14, 15,  4,  5, 18, 19, 10, 11, 24, 25,
 16, 17, 30, 31, 20, 21, 22, 23, 26, 27, 28, 29, 32, 33, 34, 35, 36, 37, 38, 39,
 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59,
 72, 73, 60, 61, 62, 63, 64, 65, 78, 79, 66, 67, 68, 69, 70, 71, 84, 85, 74, 75,
 76, 77, 90, 91, 92, 93, 94, 95, 80, 81, 82, 83, 96, 97, 98, 99,100,101, 86, 87,
 88, 89,102,103,104,105,106,107,108,109,110,111,112,113,126,127,128,129,130,131,
114,115,116,117,118,119,132,133,134,135,136,137,120,121,122,123,124,125,138,139,
140,141,142,143,144,145,146,147,148,149,162,163,164,165,166,167,180,181,150,151,
152,153,154,155,168,169,170,171,172,173,186,187,156,157,158,159,160,161,174,175,
176,177,178,179,192,193,182,183,184,185,198,199,200,201,202,203,216,217,218,219,
220,221,234,235,188,189,190,191,204,205,206,207,208,209,222,223,224,225,226,227,
240,241,194,195,196,197,210,211,212,213,214,215,228,229,230,231,232,233,246,247,
236,237,238,239,252,253,254,255,256,257,270,271,272,273,274,275,288,289,290,291,
292,293,306,307,242,243,244,245,258,259,260,261,262,263,276,277,278,279,280,281,
294,295,296,297,298,299,312,313,248,249,250,251,264,265,266,267,268,269,282,283,
284,285,286,287,300,301,302,303,304,305,318,319,308,309,310,311,324,325,326,327,
328,329,342,343,344,345,346,347,360,361,362,363,364,365,378,379,380,381,382,383,
396,397,398,399,314,315,316,317,330,331,332,333,334,335,348,349,350,351,352,353,
366,367,368,369,370,371,384,385,386,387,388,389,402,403,404,405,320,321,322,323,
336,337,338,339,340,341,354,355,356,357,358,359,372,373,374,375,376,377,390,391,
392,393,394,395,408,409,410,411,400,401,414,415,416,417,418,419,432,433,434,435,
436,437,450,451,452,453,454,455,468,469,470,471,472,473,486,487,488,489,490,491,
504,505,506,507,508,509,522,523,524,525,526,527,406,407,420,421,422,423,424,425,
438,439,440,441,442,443,456,457,458,459,460,461,474,475,476,477,478,479,492,493,
494,495,496,497,510,511,512,513,514,515,528,529,530,531,532,533,412,413,426,427,
428,429,430,431,444,445,446,447,448,449,462,463,464,465,466,467,480,481,482,483,
484,485,498,499,500,501,502,503,516,517,518,519,520,521,534,535,536,537,538,539,
540,541,542,543,544,545,558,559,560,561,562,563,546,547,548,549,550,551,564,565,
566,567,568,569,552,553,554,555,556,557,570,571,572,573,574,575},
{ 0,  1,  2,  3,  6,  7,  8,  9, 12, 13, 14, 15,  4,  5, 18, 19, 10, 11, 24, 25,
```

16, 17, 30, 31, 20, 21, 22, 23, 26, 27, 28, 29, 32, 33, 34, 35, 36, 37, 38, 39,
40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59,
72, 73, 60, 61, 62, 63, 64, 65, 78, 79, 66, 67, 68, 69, 70, 71, 84, 85, 74, 75,
76, 77, 90, 91, 92, 93, 94, 95, 80, 81, 82, 83, 96, 97, 98, 99, 100, 101, 86, 87,
88, 89, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 126, 127, 128, 129, 130, 131,
114, 115, 116, 117, 118, 119, 132, 133, 134, 135, 136, 137, 120, 121, 122, 123, 124, 125, 138, 139,
140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 162, 163, 164, 165, 166, 167, 180, 181, 150, 151,
152, 153, 154, 155, 168, 169, 170, 171, 172, 173, 186, 187, 156, 157, 158, 159, 160, 161, 174, 175,
176, 177, 178, 179, 192, 193, 182, 183, 184, 185, 198, 199, 200, 201, 202, 203, 216, 217, 218, 219,
220, 221, 234, 235, 188, 189, 190, 191, 204, 205, 206, 207, 208, 209, 222, 223, 224, 225, 226, 227,
240, 241, 194, 195, 196, 197, 210, 211, 212, 213, 214, 215, 228, 229, 230, 231, 232, 233, 246, 247,
236, 237, 238, 239, 252, 253, 254, 255, 256, 257, 270, 271, 272, 273, 274, 275, 288, 289, 290, 291,
292, 293, 306, 307, 242, 243, 244, 245, 258, 259, 260, 261, 262, 263, 276, 277, 278, 279, 280, 281,
294, 295, 296, 297, 298, 299, 312, 313, 248, 249, 250, 251, 264, 265, 266, 267, 268, 269, 282, 283,
284, 285, 286, 287, 300, 301, 302, 303, 304, 305, 318, 319, 308, 309, 310, 311, 324, 325, 326, 327,
328, 329, 342, 343, 344, 345, 346, 347, 360, 361, 362, 363, 364, 365, 378, 379, 380, 381, 382, 383,
396, 397, 314, 315, 316, 317, 330, 331, 332, 333, 334, 335, 348, 349, 350, 351, 352, 353, 366, 367,
368, 369, 370, 371, 384, 385, 386, 387, 388, 389, 402, 403, 320, 321, 322, 323, 336, 337, 338, 339,
340, 341, 354, 355, 356, 357, 358, 359, 372, 373, 374, 375, 376, 377, 390, 391, 392, 393, 394, 395,
408, 409, 398, 399, 400, 401, 414, 415, 416, 417, 418, 419, 432, 433, 434, 435, 436, 437, 450, 451,
452, 453, 454, 455, 468, 469, 470, 471, 472, 473, 486, 487, 488, 489, 490, 491, 504, 505, 506, 507,
508, 509, 404, 405, 406, 407, 420, 421, 422, 423, 424, 425, 438, 439, 440, 441, 442, 443, 456, 457,
458, 459, 460, 461, 474, 475, 476, 477, 478, 479, 492, 493, 494, 495, 496, 497, 510, 511, 512, 513,
514, 515, 410, 411, 412, 413, 426, 427, 428, 429, 430, 431, 444, 445, 446, 447, 448, 449, 462, 463,
464, 465, 466, 467, 480, 481, 482, 483, 484, 485, 498, 499, 500, 501, 502, 503, 516, 517, 518, 519,
520, 521, 522, 523, 524, 525, 526, 527, 540, 541, 542, 543, 544, 545, 558, 559, 560, 561, 562, 563,
528, 529, 530, 531, 532, 533, 546, 547, 548, 549, 550, 551, 564, 565, 566, 567, 568, 569, 534, 535,
536, 537, 538, 539, 552, 553, 554, 555, 556, 557, 570, 571, 572, 573, 574, 575}
},
{
{ 0, 1, 2, 3, 6, 7, 8, 9, 12, 13, 14, 15, 4, 5, 18, 19, 10, 11, 24, 25,
16, 17, 30, 31, 20, 21, 22, 23, 26, 27, 28, 29, 32, 33, 34, 35, 36, 37, 38, 39,
42, 43, 44, 45, 48, 49, 50, 51, 40, 41, 54, 55, 56, 57, 46, 47, 60, 61, 62, 63,
52, 53, 66, 67, 68, 69, 58, 59, 72, 73, 74, 75, 76, 77, 64, 65, 78, 79, 80, 81,
82, 83, 70, 71, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 108, 109, 110, 111,
96, 97, 98, 99, 100, 101, 114, 115, 116, 117, 102, 103, 104, 105, 106, 107, 120, 121, 122, 123,
112, 113, 126, 127, 128, 129, 130, 131, 144, 145, 146, 147, 118, 119, 132, 133, 134, 135, 136, 137,
150, 151, 152, 153, 124, 125, 138, 139, 140, 141, 142, 143, 156, 157, 158, 159, 148, 149, 162, 163,
164, 165, 166, 167, 180, 181, 182, 183, 184, 185, 154, 155, 168, 169, 170, 171, 172, 173, 186, 187,
188, 189, 190, 191, 160, 161, 174, 175, 176, 177, 178, 179, 192, 193, 194, 195, 196, 197, 198, 199,
200, 201, 202, 203, 216, 217, 218, 219, 220, 221, 234, 235, 236, 237, 238, 239, 204, 205, 206, 207,
208, 209, 222, 223, 224, 225, 226, 227, 240, 241, 242, 243, 244, 245, 210, 211, 212, 213, 214, 215,
228, 229, 230, 231, 232, 233, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 270, 271,
272, 273, 274, 275, 288, 289, 290, 291, 292, 293, 306, 307, 308, 309, 258, 259, 260, 261, 262, 263,
276, 277, 278, 279, 280, 281, 294, 295, 296, 297, 298, 299, 312, 313, 314, 315, 264, 265, 266, 267,
268, 269, 282, 283, 284, 285, 286, 287, 300, 301, 302, 303, 304, 305, 318, 319, 320, 321, 310, 311,
324, 325, 326, 327, 328, 329, 342, 343, 344, 345, 346, 347, 360, 361, 362, 363, 364, 365, 378, 379,
380, 381, 382, 383, 396, 397, 398, 399, 316, 317, 330, 331, 332, 333, 334, 335, 348, 349, 350, 351,
352, 353, 366, 367, 368, 369, 370, 371, 384, 385, 386, 387, 388, 389, 402, 403, 404, 405, 322, 323,
336, 337, 338, 339, 340, 341, 354, 355, 356, 357, 358, 359, 372, 373, 374, 375, 376, 377, 390, 391,
392, 393, 394, 395, 408, 409, 410, 411, 400, 401, 414, 415, 416, 417, 418, 419, 432, 433, 434, 435,
436, 437, 450, 451, 452, 453, 454, 455, 468, 469, 470, 471, 472, 473, 486, 487, 488, 489, 490, 491,
504, 505, 506, 507, 508, 509, 522, 523, 524, 525, 526, 527, 540, 541, 542, 543, 544, 545, 558, 559,
560, 561, 562, 563, 406, 407, 420, 421, 422, 423, 424, 425, 438, 439, 440, 441, 442, 443, 456, 457,
458, 459, 460, 461, 474, 475, 476, 477, 478, 479, 492, 493, 494, 495, 496, 497, 510, 511, 512, 513,
514, 515, 528, 529, 530, 531, 532, 533, 546, 547, 548, 549, 550, 551, 564, 565, 566, 567, 568, 569,
412, 413, 426, 427, 428, 429, 430, 431, 444, 445, 446, 447, 448, 449, 462, 463, 464, 465, 466, 467,
480, 481, 482, 483, 484, 485, 498, 499, 500, 501, 502, 503, 516, 517, 518, 519, 520, 521, 534, 535,
536, 537, 538, 539, 552, 553, 554, 555, 556, 557, 570, 571, 572, 573, 574, 575},

{ 0, 1, 2, 3, 6, 7, 8, 9, 12, 13, 14, 15, 4, 5, 18, 19, 10, 11, 24, 25,
16, 17, 30, 31, 20, 21, 22, 23, 26, 27, 28, 29, 32, 33, 34, 35, 36, 37, 38, 39,
42, 43, 44, 45, 48, 49, 50, 51, 40, 41, 54, 55, 56, 57, 46, 47, 60, 61, 62, 63,
52, 53, 66, 67, 68, 69, 58, 59, 72, 73, 74, 75, 64, 65, 78, 79, 80, 81, 70, 71,
84, 85, 86, 87, 76, 77, 90, 91, 92, 93, 94, 95, 108, 109, 82, 83, 96, 97, 98, 99,
100, 101, 114, 115, 88, 89, 102, 103, 104, 105, 106, 107, 120, 121, 110, 111, 112, 113, 126, 127,
128, 129, 130, 131, 144, 145, 116, 117, 118, 119, 132, 133, 134, 135, 136, 137, 150, 151, 122, 123,
124, 125, 138, 139, 140, 141, 142, 143, 156, 157, 146, 147, 148, 149, 162, 163, 164, 165, 166, 167,
180, 181, 182, 183, 152, 153, 154, 155, 168, 169, 170, 171, 172, 173, 186, 187, 188, 189, 158, 159,
160, 161, 174, 175, 176, 177, 178, 179, 192, 193, 194, 195, 184, 185, 198, 199, 200, 201, 202, 203,
216, 217, 218, 219, 220, 221, 234, 235, 190, 191, 204, 205, 206, 207, 208, 209, 222, 223, 224, 225,
226, 227, 240, 241, 196, 197, 210, 211, 212, 213, 214, 215, 228, 229, 230, 231, 232, 233, 246, 247,

```
236, 237, 238, 239, 252, 253, 254, 255, 256, 257, 270, 271, 272, 273, 274, 275, 288, 289, 290, 291,
242, 243, 244, 245, 258, 259, 260, 261, 262, 263, 276, 277, 278, 279, 280, 281, 294, 295, 296, 297,
248, 249, 250, 251, 264, 265, 266, 267, 268, 269, 282, 283, 284, 285, 286, 287, 300, 301, 302, 303,
292, 293, 306, 307, 308, 309, 310, 311, 324, 325, 326, 327, 328, 329, 342, 343, 344, 345, 346, 347,
360, 361, 362, 363, 364, 365, 298, 299, 312, 313, 314, 315, 316, 317, 330, 331, 332, 333, 334, 335,
348, 349, 350, 351, 352, 353, 366, 367, 368, 369, 370, 371, 304, 305, 318, 319, 320, 321, 322, 323,
336, 337, 338, 339, 340, 341, 354, 355, 356, 357, 358, 359, 372, 373, 374, 375, 376, 377, 378, 379,
380, 381, 382, 383, 396, 397, 398, 399, 400, 401, 414, 415, 416, 417, 418, 419, 432, 433, 434, 435,
436, 437, 450, 451, 452, 453, 454, 455, 468, 469, 470, 471, 472, 473, 486, 487, 488, 489, 490, 491,
504, 505, 506, 507, 508, 509, 522, 523, 524, 525, 526, 527, 540, 541, 542, 543, 544, 545, 558, 559,
560, 561, 562, 563, 384, 385, 386, 387, 388, 389, 402, 403, 404, 405, 406, 407, 420, 421, 422, 423,
424, 425, 438, 439, 440, 441, 442, 443, 456, 457, 458, 459, 460, 461, 474, 475, 476, 477, 478, 479,
492, 493, 494, 495, 496, 497, 510, 511, 512, 513, 514, 515, 528, 529, 530, 531, 532, 533, 546, 547,
548, 549, 550, 551, 564, 565, 566, 567, 568, 569, 390, 391, 392, 393, 394, 395, 408, 409, 410, 411,
412, 413, 426, 427, 428, 429, 430, 431, 444, 445, 446, 447, 448, 449, 462, 463, 464, 465, 466, 467,
480, 481, 482, 483, 484, 485, 498, 499, 500, 501, 502, 503, 516, 517, 518, 519, 520, 521, 534, 535,
536, 537, 538, 539, 552, 553, 554, 555, 556, 557, 570, 571, 572, 573, 574, 575},

{ 0,  1,  2,  3,  6,  7,  8,  9, 12, 13, 14, 15,  4,  5, 18, 19, 10, 11, 24, 25,
 16, 17, 30, 31, 20, 21, 22, 23, 26, 27, 28, 29, 32, 33, 34, 35, 36, 37, 38, 39,
 42, 43, 44, 45, 48, 49, 50, 51, 40, 41, 54, 55, 56, 57, 46, 47, 60, 61, 62, 63,
 52, 53, 66, 67, 68, 69, 58, 59, 72, 73, 74, 75, 76, 77, 64, 65, 78, 79, 80, 81,
 82, 83, 70, 71, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95,108,109,110,111,
112,113, 96, 97, 98, 99,100,101,114,115,116,117,118,119,102,103,104,105,106,107,
120,121,122,123,124,125,126,127,128,129,130,131,144,145,146,147,148,149,162,163,
164,165,132,133,134,135,136,137,150,151,152,153,154,155,168,169,170,171,138,139,
140,141,142,143,156,157,158,159,160,161,174,175,176,177,166,167,180,181,182,183,
184,185,198,199,200,201,202,203,216,217,218,219,220,221,172,173,186,187,188,189,
190,191,204,205,206,207,208,209,222,223,224,225,226,227,178,179,192,193,194,195,
196,197,210,211,212,213,214,215,228,229,230,231,232,233,234,235,236,237,238,239,
252,253,254,255,256,257,270,271,272,273,274,275,288,289,290,291,292,293,306,307,
240,241,242,243,244,245,258,259,260,261,262,263,276,277,278,279,280,281,294,295,
296,297,298,299,312,313,246,247,248,249,250,251,264,265,266,267,268,269,282,283,
284,285,286,287,300,301,302,303,304,305,318,319,308,309,310,311,324,325,326,327,
328,329,342,343,344,345,346,347,360,361,362,363,364,365,378,379,380,381,382,383,
396,397,398,399,400,401,314,315,316,317,330,331,332,333,334,335,348,349,350,351,
352,353,366,367,368,369,370,371,384,385,386,387,388,389,402,403,404,405,406,407,
320,321,322,323,336,337,338,339,340,341,354,355,356,357,358,359,372,373,374,375,
376,377,390,391,392,393,394,395,408,409,410,411,412,413,414,415,416,417,418,419,
432,433,434,435,436,437,450,451,452,453,454,455,468,469,470,471,472,473,486,487,
488,489,490,491,504,505,506,507,508,509,522,523,524,525,526,527,420,421,422,423,
424,425,438,439,440,441,442,443,456,457,458,459,460,461,474,475,476,477,478,479,
492,493,494,495,496,497,510,511,512,513,514,515,528,529,530,531,532,533,426,427,
428,429,430,431,444,445,446,447,448,449,462,463,464,465,466,467,480,481,482,483,
484,485,498,499,500,501,502,503,516,517,518,519,520,521,534,535,536,537,538,539,
540,541,542,543,544,545,558,559,560,561,562,563,546,547,548,549,550,551,564,565,
566,567,568,569,552,553,554,555,556,557,570,571,572,573,574,575}
};
float t_dewindow[17][32] =              {{
   0.000000000 , -7.2497806675 ,  53.248373787 , -114.746495164 , 509.2344536755 , -1288.21068509 , 1643.449839497 , -
9371.963978941 ,
 18758.92751087 , 9371.963978941 , 1643.449839497 ,  1288.21068509 , 509.2344536755 ,  114.746495164 ,  53.248373787 ,
7.2497806675 ,
   0.000000000 , -7.2497806675 ,  53.248373787 , -114.746495164 , 509.2344536755 , -1288.21068509 , 1643.449839497 , -
9371.963978941 ,
 18758.92751087 , 9371.963978941 , 1643.449839497 ,  1288.21068509 , 509.2344536755 ,  114.746495164 ,  53.248373787 ,
7.2497806675 ,
},{
 -0.2499958265 ,  -7.749755937 ,  54.498336536 ,  -129.746048152 ,  499.984739163 , -1379.207904407 , 1489.704536054 , -
9833.699897305 ,
  18747.4278503 , 8909.728085307 , 1783.445569084 ,  1196.963469947 ,  515.734263247 ,  100.246933829 ,  51.998411038 ,
6.4998095715 ,
 -0.2499958265 ,  -7.749755937 ,  54.498336536 ,  -129.746048152 ,  499.984739163 , -1379.207904407 , 1489.704536054 , -
9833.699897305 ,
  18747.4278503 , 8909.728085307 , 1783.445569084 ,  1196.963469947 ,  515.734263247 ,  100.246933829 ,  51.998411038 ,
6.4998095715 ,
},{
 -0.2499958265 ,  -8.749739243 ,  55.4983034585 , -145.245560026 ,  487.985103326 , -1469.705148454 , 1321.95966293 , -
10293.68584488 ,
  18713.42889302 , 8447.492191674 , 1909.941704258 , 1106.216246457 , 519.9841267635 ,  86.7473558 ,  50.4984524625 ,
5.9998179185 ,
```

-0.2499958265 , -8.749739243 , 55.4983034585 , -145.245560026 , 487.985103326 , -1469.705148454 , 1321.95966293 , -10293.68584488 ,
18713.42889302 , 8447.492191674 , 1909.941704258 , 1106.216246457 , 519.9841267635 , 86.7473558 , 50.4984524625 , 5.9998179185 ,
},{
-0.2499958265 , -9.499710339 , 56.248290938 , -161.2450799365 , 473.235562548 , -1559.202409195 , 1140.21520374 , -10751.17188335 ,
18656.93062264 , 7986.506260789 , 2022.938261404 , 1015.71900241 , 521.7340811655 , 73.497757214 , 48.9985102705 , 5.2498468225 ,
-0.2499958265 , -9.499710339 , 56.248290938 , -161.2450799365 , 473.235562548 , -1559.202409195 , 1140.21520374 , -10751.17188335 ,
18656.93062264 , 7986.506260789 , 2022.938261404 , 1015.71900241 , 521.7340811655 , 73.497757214 , 48.9985102705 , 5.2498468225 ,
},{
-0.2499958265 , -10.249681435 , 56.7482662075 , -177.7445751165 , 455.486104619 , -1647.199727744 , 943.9711832145 , -11204.90804995 ,
18577.68304333 , 7527.770259576 , 2122.93521579 , 926.2217252855 , 521.2340895125 , 60.9981461075 , 47.498551695 , 4.7498551695 ,
-0.2499958265 , -10.249681435 , 56.7482662075 , -177.7445751165 , 455.486104619 , -1647.199727744 , 943.9711832145 , -11204.90804995 ,
18577.68304333 , 7527.770259576 , 2122.93521579 , 926.2217252855 , 521.2340895125 , 60.9981461075 , 47.498551695 , 4.7498551695 ,
},{
-0.2499958265 , -11.249664741 , 56.998262034 , -194.7440619495 , 434.736729539 , -1733.697087717 , 733.727609391 , -11653.894345 ,
18476.4361262 , 7072.034175514 , 2209.932551033 , 837.724431467 , 518.7341640145 , 49.2484897135 , 45.748597293 , 4.2498635165 ,
-0.2499958265 , -11.249664741 , 56.998262034 , -194.7440619495 , 434.736729539 , -1733.697087717 , 733.727609391 , -11653.894345 ,
18476.4361262 , 7072.034175514 , 2209.932551033 , 837.724431467 , 518.7341640145 , 49.2484897135 , 45.748597293 , 4.2498635165 ,
},{
-0.2499958265 , -12.2496316635 , 56.998262034 , -211.9935282255 , 410.9874536915 , -1817.694522193 , 509.2344536755 , -12097.13081795 ,
18353.18988762 , 6620.297959142 , 2284.680270996 , 750.977075667 , 514.2343046715 , 38.2488371825 , 43.9986592745 , 3.9998840735 ,
-0.2499958265 , -12.2496316635 , 56.998262034 , -211.9935282255 , 410.9874536915 , -1817.694522193 , 509.2344536755 , -12097.13081795 ,
18353.18988762 , 6620.297959142 , 2284.680270996 , 750.977075667 , 514.2343046715 , 38.2488371825 , 43.9986592745 , 3.9998840735 ,
},{
-0.499991653 , -13.249598586 , 56.7482662075 , -229.7429861545 , 383.7382854235 , -1899.192047554 , 270.491748835 , -12533.86748133 ,
18208.19430703 , 6173.31159794 , 2347.178375679 , 665.7296784425 , 507.9844909265 , 27.749159921 , 42.2487048725 , 3.4998924205 ,
-0.499991653 , -13.249598586 , 56.7482662075 , -229.7429861545 , 383.7382854235 , -1899.192047554 , 270.491748835 , -12533.86748133 ,
18208.19430703 , 6173.31159794 , 2347.178375679 , 665.7296784425 , 507.9844909265 , 27.749159921 , 42.2487048725 , 3.4998924205 ,
},{
-0.499991653 , -14.499561335 , 55.9982951115 , -247.74243991 , 353.489216388 , -1977.439643554 , 17.4994621025 , -12962.85438878 ,
18041.69939666 , 5732.075058829 , 2397.926823968 , 582.4822314465 , 500.2347349895 , 17.9994537555 , 40.2487710275 , 3.249896594 ,
-0.499991653 , -14.499561335 , 55.9982951115 , -247.74243991 , 353.489216388 , -1977.439643554 , 17.4994621025 , -12962.85438878 ,
18041.69939666 , 5732.075058829 , 2397.926823968 , 582.4822314465 , 500.2347349895 , 17.9994537555 , 40.2487710275 , 3.249896594 ,
},{
-0.499991653 , -15.749524084 , 55.248307632 , -265.991889492 , 319.990234375 , -2052.187363516 , -249.4923779285 , -13383.09157337 ,
17854.4551112 , 5297.088333464 , 2437.425607516 , 501.4846977385 , 490.485028824 , 8.999718686 , 38.498833009 , 2.7499213245 ,
-0.499991653 , -15.749524084 , 55.248307632 , -265.991889492 , 319.990234375 , -2052.187363516 , -249.4923779285 , -13383.09157337 ,
17854.4551112 , 5297.088333464 , 2437.425607516 , 501.4846977385 , 490.485028824 , 8.999718686 , 38.498833009 , 2.7499213245 ,
},{
-0.499991653 , -16.999486833 , 53.74836544 , -284.2413226905 , 282.741364115 , -2122.685219963 , -530.483804025 , -13794.07902706 ,
17646.96144232 , 4869.351388766 , 2465.67475909 , 422.9870895285 , 479.735355736 , 0.499991653 , 36.748878607 , 2.499925498 ,

```
 -0.499991653 , -16.999486833 ,  53.74836544 , -284.2413226905 ,  282.741364115 , -2122.685219963 , -530.483804025 , -
13794.07902706 ,
 17646.96144232 , 4869.351388766 , 2465.67475909 , 422.9870895285 ,  479.735355736 ,  0.499991653 ,  36.748878607 ,
2.499925498 ,
},{
 -0.749971096 , -18.249449582 ,  51.998411038 , -302.4907722725 ,  242.4925930875 , -2188.683200683 , -824.974824534 , -
14194.06681184 ,
 17419.21839001 , 4449.614212215 , 2483.674196462 ,  346.9894068165 ,  467.485740456 ,  -7.2497806675 ,  34.748944762 ,
2.2499296715 ,
 -0.749971096 , -18.249449582 ,  51.998411038 , -302.4907722725 ,  242.4925930875 , -2188.683200683 , -824.974824534 , -
14194.06681184 ,
 17419.21839001 , 4449.614212215 , 2483.674196462 ,  346.9894068165 ,  467.485740456 ,  -7.2497806675 ,  34.748944762 ,
2.2499296715 ,
},{
 -0.749971096 , -19.749391774 ,  49.998477193 , -320.740205471 ,  198.4939501965 , -2249.431350964 , -1133.215418899 , -
14582.80494826 ,
 17172.47591702 , 4038.62674214 , 2491.423968782 ,  273.741645429 ,  454.23614187 ,  -14.2495655085 ,  32.99899036 ,
1.999933845 ,
 -0.749971096 , -19.749391774 ,  49.998477193 , -320.740205471 ,  198.4939501965 , -2249.431350964 , -1133.215418899 , -
14582.80494826 ,
 17172.47591702 , 4038.62674214 , 2491.423968782 ,  273.741645429 ,  454.23614187 ,  -14.2495655085 ,  32.99899036 ,
1.999933845 ,
},{
 -0.9999669225 , -21.2493503495 ,  47.2485558685 , -338.989655053 ,  151.2453779445 , -2304.679658596 , -1454.455616023 , -
14959.04347357 ,
 16906.73403974 , 3636.889002961 , 2489.67401438 ,  203.493784809 ,  439.736580535 ,  -20.7493586965 ,  31.2490523415 ,
1.749954402 ,
 -0.9999669225 , -21.2493503495 ,  47.2485558685 , -338.989655053 ,  151.2453779445 , -2304.679658596 , -1454.455616023 , -
14959.04347357 ,
 16906.73403974 , 3636.889002961 , 2489.67401438 ,  203.493784809 ,  439.736580535 ,  -20.7493586965 ,  31.2490523415 ,
1.749954402 ,
},{
 -0.9999669225 ,  -22.749308925 ,  44.248655101 , -356.9891088085 ,  100.4969296555 , -2353.928164693 , -1788.44542008 , -
15321.78240447 ,
 16622.99269232 , 3244.900969947 , 2478.924341292 ,  136.24584134 ,  424.487048104 ,  -26.499197172 ,  29.249102113 ,
1.749954402 ,
 -0.9999669225 ,  -22.749308925 ,  44.248655101 , -356.9891088085 ,  100.4969296555 , -2353.928164693 , -1788.44542008 , -
15321.78240447 ,
 16622.99269232 , 3244.900969947 , 2478.924341292 ,  136.24584134 ,  424.487048104 ,  -26.499197172 ,  29.249102113 ,
1.749954402 ,
},{
 -1.249962749 , -24.2492675005 ,  40.7487626805 , -374.488570911 ,  46.248588946 , -2396.176869566 , -2134.934851627 , -
15670.52176569 ,
 16322.00187862 , 2863.662610022 , 2459.424945345 ,  71.9977986385 ,  408.4875281935 ,  -31.749027611 ,  27.749159921 ,
1.4999585755 ,
 -1.249962749 , -24.2492675005 ,  40.7487626805 , -374.488570911 ,  46.248588946 , -2396.176869566 , -2134.934851627 , -
15670.52176569 ,
 16322.00187862 , 2863.662610022 , 2459.424945345 ,  71.9977986385 ,  408.4875281935 ,  -31.749027611 ,  27.749159921 ,
1.4999585755 ,
},{
 -1.249962749 , -25.999205519 ,  36.4988827805 , -391.738037187 ,  -11.249664741 , -2431.675785424 , -2493.673898454 , -
16004.26157392 ,
 16004.26157392 , 2493.673898454 , 2431.675785424 ,  11.249664741 ,  391.738037187 ,  -36.4988827805 ,  25.999205519 ,
1.249962749 ,
 -1.249962749 , -25.999205519 ,  36.4988827805 , -391.738037187 ,  -11.249664741 , -2431.675785424 , -2493.673898454 , -
16004.26157392 ,
 16004.26157392 , 2493.673898454 , 2431.675785424 ,  11.249664741 ,  391.738037187 ,  -36.4988827805 ,  25.999205519 ,
1.249962749 ,
}};
void open(void);
void save(void);
void decodeMPEG(void);
int decode_scalefactors(struct SIDE_INFO *info,int gr);
void fillbfr(int advance);
int _fillbfr(unsigned int size);
unsigned int getbits(int n);
unsigned int _getbits(int n);
int get_syncword(void);
int get_input(unsigned char* bp, unsigned int size);
void get_header(struct AUDIO_HEADER *header);
void getinfo(struct AUDIO_HEADER *header,struct SIDE_INFO *info);
static  unsigned int viewbits(int n);
```

```
static  void sackbits(int n);
static  int huffman_decode(int tbl,int *x,int *y);
static  int _qsign(int x,int *q);
int decode_huffman_data(struct SIDE_INFO *info,int gr,int ssize);
void imdct_init(void);
void imdct(int win_type,int sb);
void premultiply(void);
int layer3_frame(struct AUDIO_HEADER *header);
void requantize_mono(int gr,struct SIDE_INFO *info,struct AUDIO_HEADER *header);
static float fras2(int is,float a);
static float fras_l(int sfb,int global_gain,int scalefac_scale,int scalefac,int preflag);
static float fras_s(int global_gain,int subblock_gain,int scalefac_scale,int scalefac);
void calculate_t43(void);
void alias_reduction(void);
void poly1(int i);
short put_sample(int out);
/*-------------------------main-------------------------------------*/
void main(void)
{
        open();
        save();
        if (in_file != NULL) {

                append=data=0;

                memset(s,0,sizeof s);

                memset(res,0,sizeof res);

                calculate_t43();        /* is pow 4/3 */

                imdct_init();
                decodeMPEG();           /* find sync and read header */

                fclose(in_file);
                fclose(out_file);       /* close file when end */
        }
        printf("Complete\n");
        getch();
}
/*-------------------------open-------------------------------------*/
void open(void)
{
printf("Please enter file name and path \"DECODE\" : E:\\Hleang\\Test1\\3.mp3 \n");
if ((in_file =fopen(filename,"rb"))==NULL)
        printf("NO file %s\n",filename);
else
        printf("File %s loaded\n",filename);
}
/*-------------------------save-------------------------------------*/
void save(void)
{
printf("Please enter file name and path to \"SAVE\"(*.pcm): E:\\Hleang\\Test1\\35.pcm \n");
out_file = fopen(savefile,"wb");
}
/*-------------------------calculate_t43-------------------------------------*/
void calculate_t43(void)      /* |is| pow 4/3 (4/3 = 1.3333333333333333) */
{
int i;
for (i=0;i<30;i++){
        t_43[i]=(float)pow((float)i,1.33333333333f);
        }
}
/*-------------------------imdct_init-------------------------------------*/
void imdct_init(void)
{
int i;
for(i=0;i<36;i++)
        win[0][i] = (float) sin(PI36 *(i+0.5));
for(i=0;i<18;i++)
        win[1][i] = (float) sin(PI36 *(i+0.5));
for(i=18;i<24;i++)
```

```
                win[1][i] = 1.0f;
for(i=24;i<30;i++)
                win[1][i] = (float) sin(PI12 *(i+0.5-18));
for(i=30;i<36;i++)
                win[1][i] = 0.0f;
for(i=0;i<6;i++) /* 3 */
                win[3][i] = 0.0f;
for(i=6;i<12;i++)
                win[3][i] = (float) sin(PI12 * (i+ 0.5 - 6.0));
for(i=12;i<18;i++)
                win[3][i] = 1.0f;
for(i=18;i<36;i++)
                win[3][i] = (float) sin(PI36 * (i + 0.5));
}
/*-------------------------decodeMPEG-------------------------------*/
void decodeMPEG(void)
{
  int sync,count;
  count=0;
  while (1) {
                sync=get_syncword();
  printf("Syncword = %d , counter = %d \n",sync,count);
  if (sync != 4095) break;
  count++;
  get_header(&header);
                getinfo(&header,&info);
                layer3_frame(&header); /******  Layer 3 *******/
  }
                printf("Please any key to exit :");
}
/*-------------------------get_input-------------------------------*/
int get_input(unsigned char* bp, unsigned int size)
{
if ( fread( bp , 1, size, in_file) != size)
          if (feof(in_file)) return GETHDR_EOF;
          else return GETHDR_ERR;
return 0;
}
/*-------------------------fillbfr-------------------------------*/
void fillbfr(int advance)
{
int overflow;
get_input(&buffer[append], advance); /* buffer pointers: append counts in bytes, data in bits*/
                if ( append + advance >= BUFFER_SIZE ){
                          overflow = append + advance - BUFFER_SIZE;
                          memcpy (buffer,&buffer[BUFFER_SIZE], overflow);
                          if (overflow < 4) memcpy(&buffer[BUFFER_SIZE],buffer,4);
                                    append = overflow;
                          }
                else {
                          if (append==0) memcpy(&buffer[BUFFER_SIZE],buffer,4);
                          append+=advance;
                          }
}
/*-------------------------_fillbfr-------------------------------*/
int _fillbfr(unsigned int size)
{
                _bptr=0;
                return get_input(_buffer, size);
}
/*-------------------------getbits-------------------------------*/
unsigned int getbits(int n)
{
if (n) {
      unsigned int pos,ret_value;
            pos = data >> 3;
            ret_value = (   (buffer[pos] << 24) |
                                    (buffer[pos+1] << 16) |
                                    (buffer[pos+2] << 8) |
                                    (buffer[pos+3]));
            ret_value <<= data & 7;
            ret_value >>= 32 - n;
```

```
        data += n;
        data &= (8*BUFFER_SIZE)-1;
        return ret_value;
} else     return 0;
}
/*-------------------------_getbits-------------------------------------*/
unsigned int _getbits(int n)
{
unsigned int pos,ret_value;
    pos = _bptr >> 3;
    ret_value =         (_buffer[pos] << 24 )|
                        (_buffer[pos+1] << 16 )|
                        (_buffer[pos+2] << 8) |
                        (_buffer[pos+3]);
    ret_value <<= _bptr & 7;
    ret_value >>= 32 - n;
    _bptr += n;
    return ret_value;
}
/*-----------------------------------------------------------------------*/
/*-----------------------------Get Header Section------------------------*/
/*-----------------------------------------------------------------------*/
/*-----------------------------get_syncword------------------------------*/
int get_syncword(void)
{
int a;
        _fillbfr(4);   /* read file size 4 byte in buffer */
        a = _getbits(12);
        return a;
}
/*-----------------------------get_header--------------------------------*/
void get_header(struct AUDIO_HEADER *header)
{
    header->id=_getbits(1);
    header->layer=_getbits(2);
    header->protection_bit=_getbits(1);
    header->bitrate_index=_getbits(4);
    header->sampling_frequency=_getbits(2);
    header->padding_bit=_getbits(1);
    header->private_bit=_getbits(1);
    header->mode=_getbits(2);
    header->mode_extension=_getbits(2);
    if (!header->mode) header->mode_extension=0;
    header->copyright=_getbits(1);
    header->original=_getbits(1);
    header->emphasis=_getbits(2);
}
/*-----------------------------------------------------------------------*/
/*-----------------------Decode Side Information Section-----------------*/
/*-----------------------------------------------------------------------*/
/*-----------------------------getinfo-----------------------------------*/
void getinfo(struct AUDIO_HEADER *header,struct SIDE_INFO *info)
{
int gr,scfsi_band,region,window;
if (header->mode==3) {/* single channel*/
        if (header->id) {
                _fillbfr(17); /* Set bptr =0 and fill data from stream to buffer 17 byte */
                info->main_data_begin=_getbits(9);  /* get main data  9 bits */
                _getbits(5);                                    /* get private 5 bits */
        }else {
                _fillbfr(9);
                info->main_data_begin=_getbits(8);  /* In case not ISO can't use private */
                _getbits(1);
        }}  else {              /* dual mode */
    if (header->id){
        _fillbfr(32);
        info->main_data_begin=_getbits(9);
        _getbits(3);            /* get private 3 bit in dual mode */
      }else{
        _fillbfr(17);
        info->main_data_begin=_getbits(8);
        _getbits(2);
```

```
        }}
    if (header->id)
        for (scfsi_band=0;scfsi_band<4;scfsi_band++)
            info->scfsi[0][scfsi_band]=_getbits(1);
        for (gr=0;gr<(header->id ? 2:1);gr++){
            info->part2_3_length[gr][0]=_getbits(12);
            info->big_values[gr][0]=_getbits(9);
            info->global_gain[gr][0]=_getbits(8);
            if (header->id)     info->scalefac_compress[gr][0]=_getbits(4);
            else                info->scalefac_compress[gr][0]=_getbits(9);
            info->window_switching_flag[gr][0]=_getbits(1);
            if (info->window_switching_flag[gr][0]) {
                info->block_type[gr][0]=_getbits(2);
                info->mixed_block_flag[gr][0]=_getbits(1);
            for (region=0;region<2;region++)
                info->table_select[gr][0][region]=_getbits(5);
                info->table_select[gr][0][2]=0;
            for (window=0;window<3;window++)
                info->subblock_gain[gr][0][window]=_getbits(3);
        } else {
            for (region=0;region<3;region++)
                info->table_select[gr][0][region]=_getbits(5);
                info->region0_count[gr][0]=_getbits(4);
                info->region1_count[gr][0]=_getbits(3);
                info->block_type[gr][0]=0;
        }
        if (header->id) info->preflag[gr][0]=_getbits(1);
            info->scalefac_scale[gr][0]=_getbits(1);
            info->count1table_select[gr][0]=_getbits(1);
    }
return;
}
/*-------------------------------------------------------------*/
/*----------------------Decode ScaleFactor Section-------------*/
/*-------------------------------------------------------------*/
/*----------------------decode_scalefactors--------------------*/
int decode_scalefactors(struct SIDE_INFO *info,int gr)
{
int sfb,window;
int slen1,slen2;
int i1,i2,i=0;
        slen1=t_slen1[info->scalefac_compress[gr][0]];
        slen2=t_slen2[info->scalefac_compress[gr][0]];
        i1=3*slen1;
        i2=3*slen2;
        if (info->window_switching_flag[gr][0] && info->block_type[gr][0]==2) {
            if (info->mixed_block_flag[gr][0]) {
                for (sfb=0;sfb<8;sfb++) {
                    scalefac_[gr][0][sfb]=getbits(slen1);
                    i+=slen1;
                }
            for (sfb=3;sfb<6;sfb++) {
                for (window=0;window<3;window++)
                    scalefac_s[gr][0][sfb][window]=getbits(slen1);
                    i+=i1;
                }
            for (;sfb<12;sfb++) {
                for (window=0;window<3;window++)
                    scalefac_s[gr][0][sfb][window]=getbits(slen2);
                    i+=i2;
                }}
        else { /* !mixed_block_flag */
            for (sfb=0;sfb<6;sfb++) {
                for (window=0;window<3;window++)
                    scalefac_s[gr][0][sfb][window]=getbits(slen1);
                    i+=i1;
                }
                for (;sfb<12;sfb++) {
                    for (window=0;window<3;window++)
                        scalefac_s[gr][0][sfb][window]=getbits(slen2);
                        i+=i2;
                }}
```

```
                        for (window=0;window<3;window++)
                                scalefac_s[gr][0][12][window]=0;
                }else { /* block_type!=2 */
                        if ( !info->scfsi[0][0] || !gr )
                                for (sfb=0;sfb<6;sfb++) {
                                        scalefac_l[gr][0][sfb]=getbits(slen1);
                                        i+=slen1;
                                }
                        else for (sfb=0;sfb<6;sfb++) {
                                scalefac_l[1][0][sfb]=scalefac_l[0][0][sfb];
                                }
                        if ( !info->scfsi[0][1] || !gr )
                                for (sfb=6;sfb<11;sfb++) {
                                        scalefac_l[gr][0][sfb]=getbits(slen1);
                                        i+=slen1;
                                }
                        else for (sfb=6;sfb<11;sfb++) {
                                scalefac_l[1][0][sfb]=scalefac_l[0][0][sfb];
                                }
                        if ( !info->scfsi[0][2] || !gr )
                                for (sfb=11;sfb<16;sfb++) {
                                        scalefac_l[gr][0][sfb]=getbits(slen2);
                                        i+=slen2;
                                }
                        else for (sfb=11;sfb<16;sfb++) {
                                scalefac_l[1][0][sfb]=scalefac_l[0][0][sfb];
                        }
                        if ( !info->scfsi[0][3] || !gr )
                                for (sfb=16;sfb<21;sfb++) {
                                        scalefac_l[gr][0][sfb]=getbits(slen2);
                                        i+=slen2;
                                        }
                        else for (sfb=16;sfb<21;sfb++) {
                                scalefac_l[1][0][sfb]=scalefac_l[0][0][sfb];
                        }
          scalefac_l[gr][0][21]=0;
                  )
        return i;
        }
        /*-------------------------------------------------------------------*/
        /*-----------------------Decode Huffman Section----------------------*/
        /*-------------------------------------------------------------------*/
        /*-------------------------viewbits----------------------------------*/
        static  unsigned int viewbits(int n)
        {
        unsigned int pos,ret_value;

             pos = data >> 3;
             ret_value = (buffer[pos] << 24 )|
                     (buffer[pos+1] << 16 )|
                     (buffer[pos+2] << 8 )|
                     (buffer[pos+3]);
             ret_value <<= data & 7;
             ret_value >>= 32 - n;

             return ret_value;
        }
        /*-------------------------Sackbits----------------------------------*/
        static  void sackbits(int n)
        {
             data += n;
             data &= 8*BUFFER_SIZE-1;       /* 0111 1111 1111 1111 =7F FF */
        }

        /*-------------------------Huffman Decode----------------------------*/
        static int huffman_decode(int tbl,int *x,int *y)
        {
        unsigned int chunk;
        register unsigned int *h_tab;
        register unsigned int lag;
        register unsigned int half_lag;
        int len;                                /* hlen in ISO11172-3 huffman table */
```

```
		h_tab=tables[tbl];		/* get table huffman */
		chunk=viewbits(19);		/* view huffman code 19 bit *not get* */
		h_tab += h_cue[tbl][chunk >> (19-NC_O)];  /* NC_O=4 , shift to desire position from chunk */
		len=(*h_tab>>8)&0x1f;
		if ((*h_tab>>(32-len)) != (chunk>>(19-len))) {   /* chunk>>(19-hlen) = hcod in ISO */
			if ((chunk >> (19-NC_O)) < (N_CUE-1)){
			 lag=(h_cue[tbl][(chunk >> (19-NC_O))+1] - h_cue[tbl][chunk >> (19-NC_O)]);
	}
			else {
				exit (-1);
			}
			chunk <<= 32-19;   /* =13 */
			chunk |= 0x1ff;
			half_lag = lag >> 1;
			h_tab += half_lag;
			lag -= half_lag;
			while (lag > 1) {
			    half_lag = lag >> 1;
			    if (*h_tab < chunk)
				h_tab += half_lag;
			    else
				h_tab -= half_lag;
		  lag -= half_lag;
			}
			len=(*h_tab>>8)&0x1f;
	if ((*h_tab>>(32-len)) != (chunk>>(32-len))) {
			    if (*h_tab > chunk)
				h_tab--;
			    else
				h_tab++;
			    len=(*h_tab>>8)&0x1f;
			}
		}
		sackbits(len);
		*x=(*h_tab>>4)&0xf;
		*y=*h_tab&0xf;
		return len;
}
/*-------------------------_Qsign-------------------------------*/
static  int _qsign(int x,int *q)
{
int ret_value=0,i;
		for (i=3;i>=0;i--)
			if ((x>>i) & 1) {
				if (getbits(1)) *q++=-1;
				else *q++=1;
				ret_value++;
				}
			else *q++=0;
		return ret_value;
}
/*-------------------------Decode Huffman Data-------------------------*/
int decode_huffman_data(struct SIDE_INFO *info,int gr,int ssize)
{
int l,i,cnt,x=0,y=0;
int q[4],r[3],linbits[3],tr[4]={0,0,0,0};
int big_value = info->big_values[gr][0] << 1;
		for (l=0;l<3;l++) {
			tr[l]=info->table_select[gr][0][l];		/*5bit , select table huffman 0 - 31 */
			linbits[l]=t_linbits[info->table_select[gr][0][l]];  /* select linbit */
			}
  tr[3]=32+info->count1table_select[gr][0];		/*1bit , select 32=table A , 33=table B */
		if (!info->window_switching_flag[gr][0] && info->block_type[gr][0]==0) {
			r[0]=t_l[info->region0_count[gr][0]] + 1;  /* t_l[a] will shift form t_l[0] A*4byte(int) */
			if (r[0] > big_value)
				r[0]=r[1]=big_value;
			else {
				r[1]=t_l[ info->region0_count[gr][0] + info->region1_count[gr][0] + 1 ] + 1;
				if (r[1] > big_value)
					r[1]=big_value;
			}
			r[2]=big_value;
```

```
        } else {
                if (info->block_type[gr][0]==2 && info->mixed_block_flag[gr][0]==0)
                        r[0]=3*(t_s[2]+1);          /* =36 */
                else
                        r[0]=t_l[7]+1;              /* =36 */
                if (r[0] > big_value)
                        r[0]=big_value;
                r[1]=r[2]=big_value;
        }
        l=0; cnt=0;
        for (i=0;i<3;i++) {
                for (;l<r[i];l+=2) {
                        int j = linbits[i];
                cnt+=huffman_decode(tr[i],&x,&y);     /* cnt + len = cnt */
        if (x==15 && j>0) {
                                        x+=getbits(j);
                                        cnt+=j;
                                                }
                                        if (x) {
                                        if (getbits(1)) x=-x;   /* get sign of x */
                                        cnt++;
                                                }
                                        if (y==15 && j>0) {
                                        y+=getbits(j);
                                        cnt+=j;
                                                }
                                        if (y) {
                                        if (getbits(1)) y=-y;
                                        cnt++;
                                                }
                                is[l]=x;    /* store x in IS */
                                is[l+1]=y;  /* store y in IS */
                        }
        }
        while ((cnt < info->part2_3_length[gr][0]-ssize) && (l<576)) {
                cnt+=huffman_decode(tr[3],&x,&y);
                cnt+=_qsign(x,q);
                for (i=0;i<4;i++) is[l+i]=q[i]; /* ziher je ziher, is[578] , store v,w x,y in IS */
                l+=4;
                }
        if (cnt != info->part2_3_length[gr][0] - ssize ) {
                        data-=cnt-(info->part2_3_length[gr][0] - ssize);
                        data&= 8*BUFFER_SIZE - 1;
                        }
        if (l<576) non_zero=l;
        else non_zero=576;
        for (;l<576;l++) is[l]=0;
        return 1;
}
/*------------------------------------------------------------*/
/*-----------------------Requantize Section-------------------*/
/*------------------------------------------------------------*/
/*-----------------------requantize_mono----------------------*/
void requantize_mono(int gr,struct SIDE_INFO *info,struct AUDIO_HEADER *header)
{
int l,i,sfb;
float a;
int global_gain=info->global_gain[gr][0];
int scalefac_scale=info->scalefac_scale[gr][0];
int sfreq=header->sampling_frequency;
        if (info->window_switching_flag[gr][0] && info->block_type[gr][0]==2)
                if (info->mixed_block_flag[gr][0]) {    /* long & short block */
                int window,window_len,preflag=0; /* pretab is all zero in this low frequency area */
                int scalefac=scalefac_l[gr][0][0];
                l=0;sfb=0;
                                                a=fras_l(sfb,global_gain,scalefac_scale,scalefac,preflag);
            while (l<36) {              /* 36 loop */
                                xr[0][l]=fras2(is[l],a);
                    if (l==t_l[sfb]) {
        scalefac=scalefac_l[gr][0][++sfb];
        a=fras_l(sfb,global_gain,scalefac_scale,scalefac,preflag);
            }
```

```
                i++;
                }
                        sfb=3;
                        window_len=t_s[sfb]-t_s[sfb-1];
                        while (i<non_zero) {
                        for (window=0;window<3;window++) {
                        int scalefac=scalefac_s[gr][0][sfb][window];
                        int subblock_gain=info->subblock_gain[gr][0][window];
                        a=fras_s(global_gain,subblock_gain,scalefac_scale,scalefac);
                                for (i=0;i<window_len;i++) {
                                xr[0][t_reorder[header->id][sfreq][i]]=fras2(is[i],a);
                        i++;
                        }}
                        sfb++;
                        window_len=t_s[sfb]-t_s[sfb-1];
                }
                while (i<576) {
                xr[0][t_reorder[header->id][sfreq][i++]]=0;
}}
        else { /* short block */
                int window,window_len;
                sfb=0; i=0;
                window_len=t_s[0]+1;
                while (i<non_zero) {
                for (window=0;window<3;window++) {
                int scalefac=scalefac_s[gr][0][sfb][window];
                int subblock_gain=info->subblock_gain[gr][0][window];
                float a=fras_s(global_gain,subblock_gain,scalefac_scale,scalefac);
                for (i=0;i<window_len;i++) {
                        xr[0][t_reorder[header->id][sfreq][i]]=fras2(is[i],a);
                i++;
                }}
                sfb++;
                window_len=t_s[sfb]-t_s[sfb-1];
                }
                while (i<576)        {
                xr[0][t_reorder[header->id][sfreq][i++]]=0;
                }}
                else {      /* long blocks */
                        int preflag=info->preflag[gr][0];
                        int scalefac=scalefac_l[gr][0][0];
                        sfb=0; i=0;
                        a=fras_l(sfb,global_gain,scalefac_scale,scalefac,preflag);
                        while (i<non_zero) {  /* form huffman.c  nonzero[ch] = i = number of is not zero */
                        xr[0][i]=fras2(is[i],a);
                if (i==t_l[sfb]) {    /* when finish each band scale factor will be changed */
                scalefac=scalefac_l[gr][0][++sfb];  /* sfb up to point end scale fartor band */
                 a=fras_l(sfb,global_gain,scalefac_scale,scalefac,preflag);
                }
                i++;
                }
                while (i<576)    {
         xr[0][i++]=0;
}}}
/*-------------------------fras2-------------------------*/
static float fras2(int is,float a)  /* multiplie scale factor with is */
{
      if (is > 0)
                return t_43[is]*a;
      else
                return (-1)*t_43[-is]*a;
}
/*-------------------------fras_l-------------------------*/
static float fras_l(int sfb,int global_gain,int scalefac_scale,int scalefac,int preflag)
{               /* find scale factor what use to multiplie with IS */
register int a,scale;
scale=scalefac_scale+1;
a= global_gain -210 -(scalefac << scale);   /* scalefac * (scale*2) */
if (preflag)
                a-=(t_pretab[sfb] << scale);
if (a < -127)
                return 0;
```

```
if (a>=0)
        return tab[a&3]*(1 << (a>>2));
else
        return tabi[(-a)&3]/(2 << ((-a) >> 2));   //hleang
}
/*-------------------------fras_s-------------------------*/
static float fras_s(int global_gain,int subblock_gain,int scalefac_scale,int scalefac)
{
int a;
a=global_gain - 210 - (subblock_gain << 3);
        if (scalefac_scale)
                a-= (scalefac << 2);
        else
                a-= (scalefac << 1);
        if (a < -127)
                return 0;
    if (a>=0)
        return tab[a&3]*(1 << (a>>2));
    else
        return tabi[(-a)&3]/(2 << ((-a) >> 2));
}
/*-------------------------------------------------------------*/
/*-------------------Alias Reduction Section-------------------*/
/*-------------------------------------------------------------*/
/*-------------------alias_reduction-------------------*/
void alias_reduction()
{
unsigned int sb;
for (sb=1;sb<32;sb++) {
        float *x = xr[sb];
        register float a, b;
        a = x[0];
        b = x[-1];
        x[-1] = b * Cs[0] - a * Ca[0];
        x[0]  = a * Cs[0] + b * Ca[0];
        a = x[1];
        b = x[-2];
        x[-2] = b * Cs[1] - a * Ca[1];
        x[1]  = a * Cs[1] + b * Ca[1];
        a = x[2];
        b = x[-3];
        x[-3] = b * Cs[2] - a * Ca[2];
        x[2]  = a * Cs[2] + b * Ca[2];
        a = x[3];
        b = x[-4];
        x[-4] = b * Cs[3] - a * Ca[3];
        x[3]  = a * Cs[3] + b * Ca[3];
        a = x[4];
        b = x[-5];
        x[-5] = b * Cs[4] - a * Ca[4];
        x[4]  = a * Cs[4] + b * Ca[4];
        a = x[5];
        b = x[-6];
        x[-6] = b * Cs[5] - a * Ca[5];
        x[5]  = a * Cs[5] + b * Ca[5];
        a = x[6];
        b = x[-7];
        x[-7] = b * Cs[6] - a * Ca[6];
        x[6]  = a * Cs[6] + b * Ca[6];
        a = x[7];
        b = x[-8];
        x[-8] = b * Cs[7] - a * Ca[7];
        x[7]  = a * Cs[7] + b * Ca[7];
        }       /*      printf("alias_reduction  Complete\n"); */
}
/*-------------------------------------------------------------*/
/*-------------------IMDCT and Windowing Section-------------------*/
/*-------------------------------------------------------------*/
/*-------------------IMDCT-------------------*/
void imdct(int win_type,int sb)
{
float tmp0, tmp1, tmp2, tmp3, tmp4, tmp5, tmp6, tmp7, tmp8, tmp9, tmp10, tmp11;
```

```
register float  save;
float  pp1, pp2;
float  *win_bt;
int    i, p, ss;
float out[36];

  if(win_type == 2){    /*  win_type = 2  */
                        for(p=0;p<36;p+=9) {
                                out[p]   = out[p+1] = out[p+2] = out[p+3] =
                                out[p+4] = out[p+5] = out[p+6] = out[p+7] =
                                out[p+8] = 0.0f;
                                }
                        for(ss=0;ss<18;ss+=6) {
/* Begin 12 point , IDCT Input aliasing for 12 pt IDCT */
                        xr[sb][5+ss]+=xr[sb][4+ss];xr[sb][4+ss]+=xr[sb][3+ss];xr[sb][3+ss]+=xr[sb][2+ss];
                        xr[sb][2+ss]+=xr[sb][1+ss];xr[sb][1+ss]+=xr[sb][0+ss];
/* Input aliasing on odd indices (for 6 point IDCT) */
                        xr[sb][5+ss] += xr[sb][3+ss];  xr[sb][3+ss]  += xr[sb][1+ss];
/* 3 point IDCT on even indices */
                        pp2 = xr[sb][4+ss] * 0.500000000f;
                        pp1 = xr[sb][2+ss] * 0.866025403f;    /* (sqrt 3)/2 */
                        save = xr[sb][0+ss] + pp2;
                        tmp1 = xr[sb][0+ss] - xr[sb][4+ss];       /* 0-4-3 */
                        tmp0 = save + pp1;        /* 0+4+3+2+1 */
                        tmp2 = save - pp1;        /* 0+4+3-2-1 */
/* End 3 point IDCT on even indices */
/* 3 point IDCT on odd indices (for 6 point IDCT) */
                        pp2 = xr[sb][5+ss] * 0.500000000f;
                        pp1 = xr[sb][3+ss] * 0.866025403f;
                        save = xr[sb][1+ss] + pp2;
                        tmp4 = xr[sb][1+ss] - xr[sb][5+ss];
                        tmp5 = save + pp1;
                        tmp3 = save - pp1;
/* End 3 point IDCT on odd indices */
/* Twiddle factors on odd indices (for 6 point IDCT) */
                        tmp3 *= 1.931851653f;
                        tmp4 *= 0.707106781f;
                        tmp5 *= 0.517638090f;
/* Output butterflies on 2 3 point IDCT's (for 6 point IDCT) */
                        save = tmp0;
                        tmp0 += tmp5;
                        tmp5 = save - tmp5;
                        save = tmp1;
                        tmp1 += tmp4;
                        tmp4 = save - tmp4;
                        save = tmp2;
                        tmp2 += tmp3;
                        tmp3 = save - tmp3;
/* End 6 point IDCT */
/* Twiddle factors on indices (for 12 point IDCT) */
                        tmp0 *= 0.504314480f;
                        tmp1 *= 0.541196100f;
                        tmp2 *= 0.630236207f;
                        tmp3 *= 0.821339815f;
                        tmp4 *= 1.306562965f;
                        tmp5 *= 3.830648788f;
/* End 12 point IDCT */
/* Shift to 12 point modified IDCT, multiply by window type 2 */
                        tmp8  = tmp0 * -0.793353340f;
                        tmp9  = tmp0 * -0.608761429f;
                        tmp7  = tmp1 * -0.923879532f;
                        tmp10 = tmp1 * -0.382683432f;
                        tmp6  = tmp2 * -0.991444861f;
                        tmp11 = tmp2 * -0.130526192f;
                        tmp0  = tmp3;
                        tmp1  = tmp4 *  0.382683432f;
                        tmp2  = tmp5 *  0.608761429f;
                        tmp3  = tmp5 * -0.793353340f;
                        tmp4  = tmp4 * -0.923879532f;
                        tmp5  = tmp0 * -0.991444861f;
                        tmp0 *= 0.130526192f;
                        out[ss + 6] += tmp0;
```

```
                        out[ss + 7]  += tmp1;
                        out[ss + 8]  += tmp2;
                        out[ss + 9]  += tmp3;
                        out[ss + 10] += tmp4;
                        out[ss + 11] += tmp5;
                        out[ss + 12] += tmp6;
                        out[ss + 13] += tmp7;
                        out[ss + 14] += tmp8;
                        out[ss + 15] += tmp9;
                        out[ss + 16] += tmp10;
                        out[ss + 17] += tmp11;
}if (sb&1) {
                        for (i=0;i<18;i+=2)
                                res[sb][i]=out[i] + s[sb][i];
                        for (i=1;i<18;i+=2)
                                res[sb][i]=-out[i] - s[sb][i];
        }       else
                        for (i=0;i<18;i++)
                                res[sb][i]=out[i] + s[sb][i];
                        for (i=18;i<36;i++)
                                s[sb][i-18]=out[i];
}else {                 /*  win_type != 2  */
/* 36 point IDCT */
float tmp[18];
/*  input aliasing for 36 point IDCT */
    xr[sb][17]+=xr[sb][16]; xr[sb][16]+=xr[sb][15]; xr[sb][15]+=xr[sb][14]; xr[sb][14]+=xr[sb][13];
    xr[sb][13]+=xr[sb][12]; xr[sb][12]+=xr[sb][11]; xr[sb][11]+=xr[sb][10]; xr[sb][10]+=xr[sb][9];
    xr[sb][9] +=xr[sb][8];  xr[sb][8] +=xr[sb][7];  xr[sb][7] +=xr[sb][6];  xr[sb][6] +=xr[sb][5];
    xr[sb][5] +=xr[sb][4];  xr[sb][4] +=xr[sb][3];  xr[sb][3] +=xr[sb][2];  xr[sb][2] +=xr[sb][1];
    xr[sb][1] +=xr[sb][0];
/* 18 point IDCT for odd indices */
/* input aliasing for 18 point IDCT */
    xr[sb][17]+=xr[sb][15]; xr[sb][15]+=xr[sb][13]; xr[sb][13]+=xr[sb][11]; xr[sb][11]+=xr[sb][9];
    xr[sb][9] +=xr[sb][7];  xr[sb][7] +=xr[sb][5];  xr[sb][5] +=xr[sb][3];  xr[sb][3] +=xr[sb][1];
{
float tmp0,tmp1,tmp2,tmp3,tmp4,tmp0_,tmp1_,tmp2_,tmp3_;
float tmp0o,tmp1o,tmp2o,tmp3o,tmp4o,tmp0_o,tmp1_o,tmp2_o,tmp3_o;
/*9 point IDCT on even indices */
  {
/* 5 points on odd indices (not realy an IDCT) */
float i0 = xr[sb][0]+xr[sb][0];
float i0p12 = i0 + xr[sb][12];
            tmp0 = i0p12 + xr[sb][4]*1.8793852415718f  + xr[sb][8]*1.5320888862238f  + xr[sb][16]*0.347296355333386f;
            tmp1 = i0    + xr[sb][4]                   - xr[sb][8] - xr[sb][12] - xr[sb][12] - xr[sb][16];
            tmp2 = i0p12 - xr[sb][4]*0.347296355333386f - xr[sb][8]*1.8793852415718f  + xr[sb][16]*1.532088886238f;
            tmp3 = i0p12 - xr[sb][4]*1.532088886238f  + xr[sb][8]*0.347296355333386f - xr[sb][16]*1.8793852415718f;
            tmp4 = xr[sb][0] - xr[sb][4]              + xr[sb][8] - xr[sb][12]       + xr[sb][16];
}{
float i6_ = xr[sb][6]*1.732050808f;
            tmp0_ = xr[sb][2]*1.9696155060244f  + i6_ + xr[sb][10]*1.2855752193731f  + xr[sb][14]*0.684040286651 34f;
            tmp1_ = (xr[sb][2]                  - xr[sb][10]                 - xr[sb][14])*1.732050808f;
            tmp2_ = xr[sb][2]*1.2855752193731f  - i6_ - xr[sb][10]*0.684040286651 34f + xr[sb][14]*1.9696155060244f;
            tmp3_ = xr[sb][2]*0.684040286651 34f - i6_ + xr[sb][10]*1.9696155060244f  - xr[sb][14]*1.2855752193731f;
}
/* 9 point IDCT on odd indices */
{
/* 5 points on odd indices (not realy an IDCT) */
float i0 = xr[sb][0+1]+xr[sb][0+1];
float i0p12 = i0 + xr[sb][12+1];
tmp0o = i0p12  + xr[sb][4+1]*1.8793852415718f  + xr[sb][8+1]*1.5320888862238f  + xr[sb][16+1]*0.347296355333386f;
tmp1o = i0     + xr[sb][4+1]                   - xr[sb][8+1] - xr[sb][12+1] - xr[sb][12+1] - xr[sb][16+1];
tmp2o = i0p12  - xr[sb][4+1]*0.347296355333386f - xr[sb][8+1]*1.8793852415718f  + xr[sb][16+1]*1.532088886238f;
tmp3o = i0p12  - xr[sb][4+1]*1.532088886238f  + xr[sb][8+1]*0.347296355333386f - xr[sb][16+1]*1.8793852415718f;
tmp4o = (xr[sb][0+1] - xr[sb][4+1]              + xr[sb][8+1] - xr[sb][12+1]       + xr[sb][16+1])*0.707106781f; /* Twiddled */
}{
/* 4 points on even indices */
float i6_ = xr[sb][6+1]*1.732050808f;                 /*  Sqrt[3]  */
  tmp0_o = xr[sb][2+1]*1.9696155060244f  + i6_ + xr[sb][10+1]*1.2855752193731f  + xr[sb][14+1]*0.684040286651 34f;
  tmp1_o = (xr[sb][2+1]                  - xr[sb][10+1]                 - xr[sb][14+1])*1.732050808f;
  tmp2_o = xr[sb][2+1]*1.2855752193731f  - i6_ - xr[sb][10+1]*0.684040286651 34f + xr[sb][14+1]*1.9696155060244f;
  tmp3_o = xr[sb][2+1]*0.684040286651 34f - i6_ + xr[sb][10+1]*1.9696155060244f  - xr[sb][14+1]*1.2855752193731f;
  }
```

```
/* Twiddle factors on odd indices and Butterflies on 9 point IDCT's and twiddle factors for 36 point IDCT */
  {
float e, o;
e = tmp0 + tmp0_; o = (tmp0o + tmp0_o)*0.501909918f; tmp[0] = (e + o)*(-0.500476342f*.5f);
tmp[17] = (e - o)*(-11.46279281f*.5f);
e = tmp1 + tmp1_; o = (tmp1o + tmp1_o)*0.517638090f; tmp[1] = (e + o)*(-0.504314480f*.5f);
tmp[16] = (e - o)*(-3.830648788f*.5f);
e = tmp2 + tmp2_; o = (tmp2o + tmp2_o)*0.551688959f; tmp[2] = (e + o)*(-0.512139757f*.5f);
tmp[15] = (e - o)*(-2.310113158f*.5f);
e = tmp3 + tmp3_; o = (tmp3o + tmp3_o)*0.610387294f; tmp[3] = (e + o)*(-0.524264562f*.5f);
tmp[14] = (e - o)*(-1.662754762f*.5f);
tmp[4] = (tmp4 + tmp4o)*(-0.541196100f); tmp[13] = (tmp4 - tmp4o)*(-1.306562965f);
e = tmp3 - tmp3_; o = (tmp3o - tmp3_o)*0.871723397f; tmp[5] = (e + o)*(-0.563690973f*.5f);
tmp[12] = (e - o)*(-1.082840285f*.5f);
e = tmp2 - tmp2_; o = (tmp2o - tmp2_o)*1.183100792f; tmp[6] = (e + o)*(-0.592844523f*.5f);
tmp[11] = (e - o)*(-0.930579498f*.5f);
e = tmp1 - tmp1_; o = (tmp1o - tmp1_o)*1.931851653f; tmp[7] = (e + o)*(-0.630236207f*.5f);
tmp[10] = (e - o)*(-0.821339815f*.5f);
e = tmp0 - tmp0_; o = (tmp0o - tmp0_o)*5.736856623f; tmp[8] = (e + o)*(-0.678170852f*.5f);
tmp[9] =  (e - o)*(-0.740093616f*.5f);
}}
/* shift to modified IDCT */
win_bt = win[win_type];
        if (sb&1) {                         //Hleang
                res[sb][0] =  -tmp[9]  * win_bt[0]  + s[sb][0] ;
                res[sb][1] =-(-tmp[10] * win_bt[1]  + s[sb][1] );
                res[sb][2] =  -tmp[11] * win_bt[2]  + s[sb][2] ;
                res[sb][3] =-(-tmp[12] * win_bt[3]  + s[sb][3] );
                res[sb][4] =  -tmp[13] * win_bt[4]  + s[sb][4] ;
                res[sb][5] =-(-tmp[14] * win_bt[5]  + s[sb][5] );
                res[sb][6] =  -tmp[15] * win_bt[6]  + s[sb][6] ;
                res[sb][7] =-(-tmp[16] * win_bt[7]  + s[sb][7] );
                res[sb][8] =  -tmp[17] * win_bt[8]  + s[sb][8] ;
                res[sb][9] = -(tmp[17] * win_bt[9]  + s[sb][9] );
                res[sb][10]=  tmp[16] * win_bt[10]  + s[sb][10] ;
                res[sb][11]=-(tmp[15] * win_bt[11]  + s[sb][11] );
                res[sb][12]=  tmp[14] * win_bt[12]  + s[sb][12] ;
                res[sb][13]=-(tmp[13] * win_bt[13]  + s[sb][13] );
                res[sb][14]=  tmp[12] * win_bt[14]  + s[sb][14] ;
                res[sb][15]=-(tmp[11] * win_bt[15]  + s[sb][15] );
                res[sb][16]=  tmp[10] * win_bt[16]  + s[sb][16] ;
                res[sb][17]=-(tmp[9]  * win_bt[17]  + s[sb][17] );
                }
 else {
                res[sb][0] = -tmp[9]  * win_bt[0]  + s[sb][0] ;
                res[sb][1] = -tmp[10] * win_bt[1]  + s[sb][1] ;
                res[sb][2] = -tmp[11] * win_bt[2]  + s[sb][2] ;
                res[sb][3] = -tmp[12] * win_bt[3]  + s[sb][3] ;
                res[sb][4] = -tmp[13] * win_bt[4]  + s[sb][4] ;
                res[sb][5] = -tmp[14] * win_bt[5]  + s[sb][5] ;
                res[sb][6] = -tmp[15] * win_bt[6]  + s[sb][6] ;
                res[sb][7] = -tmp[16] * win_bt[7]  + s[sb][7] ;
                res[sb][8] = -tmp[17] * win_bt[8]  + s[sb][8] ;
                res[sb][9] = tmp[17] * win_bt[9]  + s[sb][9] ;
                res[sb][10]= tmp[16] * win_bt[10]  + s[sb][10] ;
                res[sb][11]= tmp[15] * win_bt[11]  + s[sb][11] ;
                res[sb][12]= tmp[14] * win_bt[12]  + s[sb][12] ;
                res[sb][13]= tmp[13] * win_bt[13]  + s[sb][13] ;
                res[sb][14]= tmp[12] * win_bt[14]  + s[sb][14] ;
                res[sb][15]= tmp[11] * win_bt[15]  + s[sb][15] ;
                res[sb][16]= tmp[10] * win_bt[16]  + s[sb][16] ;
                res[sb][17]= tmp[9]  * win_bt[17]  + s[sb][17] ;
                }
        s[sb][0]= tmp[8]  * win_bt[18];
        s[sb][1]= tmp[7]  * win_bt[19];
```

```
                s[sb][2]= tmp[6]  * win_bt[20];
                s[sb][3]= tmp[5]  * win_bt[21];
                s[sb][4]= tmp[4]  * win_bt[22];
                s[sb][5]= tmp[3]  * win_bt[23];
                s[sb][6]= tmp[2]  * win_bt[24];
                s[sb][7]= tmp[1]  * win_bt[25];
                s[sb][8]= tmp[0]  * win_bt[26];
                s[sb][9]= tmp[0]  * win_bt[27];
                s[sb][10]= tmp[1]  * win_bt[28];
                s[sb][11]= tmp[2]  * win_bt[29];
                s[sb][12]= tmp[3]  * win_bt[30];
                s[sb][13]= tmp[4]  * win_bt[31];
                s[sb][14]= tmp[5]  * win_bt[32];
                s[sb][15]= tmp[6]  * win_bt[33];
                s[sb][16]= tmp[7]  * win_bt[34];
                s[sb][17]= tmp[8]  * win_bt[35];
 }}
/*-------------------------------------------------------------------*/
/*-------------------Synthesis Filter Bank Section-------------------*/
/*-------------------------------------------------------------------*/
/*-------------------------Poly Phase--------------------------------*/
void poly1(int f)
{
static float u[2][2][17][16];           /* no v[][], it's redundant */
static int start = 0;
static int div = 0;
float (*u_p)[16];
                {
                float d16,d17,d18,d19,d20,d21,d22,d23,d24,d25,d26,d27,d28,d29,d30,d31;
                float d0,d1,d2,d3,d4,d5,d6,d7,d8,d9,d10,d11,d12,d13,d14,d15;
/* step 1: initial reordering and 1st (16 wide) butterflies          */
                d0 = res[ 0][f]; d16=(d0  - res[31][f]) *  1.997590912; d0 += res[31][f];
                d1 = res[ 1][f]; d17=(d1  - res[30][f]) *  1.978353019; d1 += res[30][f];
                d3 = res[ 2][f]; d19=(d3  - res[29][f]) *  1.940062506; d3 += res[29][f];
                d2 = res[ 3][f]; d18=(d2  - res[28][f]) *  1.883088130; d2 += res[28][f];
                d6 = res[ 4][f]; d22=(d6  - res[27][f]) *  1.807978586; d6 += res[27][f];
                d7 = res[ 5][f]; d23=(d7  - res[26][f]) *  1.715457220; d7 += res[26][f];
                d5 = res[ 6][f]; d21=(d5  - res[25][f]) *  1.606415063; d5 += res[25][f];
                d4 = res[ 7][f]; d20=(d4  - res[24][f]) *  1.481902251; d4 += res[24][f];
                d12= res[ 8][f]; d28=(d12 - res[23][f]) *  1.343117910; d12+= res[23][f];
                d13= res[ 9][f]; d29=(d13 - res[22][f]) *  1.191398609; d13+= res[22][f];
                d15= res[10][f]; d31=(d15 - res[21][f]) *  1.028205488; d15+= res[21][f];
                d14= res[11][f]; d30=(d14 - res[20][f]) *  0.855110187; d14+= res[20][f];
                d10= res[12][f]; d26=(d10 - res[19][f]) *  0.673779707; d10+= res[19][f];
                d11= res[13][f]; d27=(d11 - res[18][f]) *  0.485960360; d11+= res[18][f];
                d9 = res[14][f]; d25=(d9  - res[17][f]) *  0.293460949; d9 += res[17][f];
                d8 = res[15][f]; d24=(d8  - res[16][f]) *  0.098135349; d8 += res[16][f];
                {
                float c0,c1,c2,c3,c4,c5,c6,c7,c8,c9,c10,c11,c12,c13,c14,c15;
/* a test to see what can be done with memory separation first we process indexes 0-15 */
                c0 = d0 + d8 ; c8 = ( d0 - d8 ) *  1.990369453;
                c1 = d1 + d9 ; c9 = ( d1 - d9 ) *  1.913880671;
                c2 = d2 + d10; c10= ( d2 - d10) *  1.546020907;
                c3 = d3 + d11; c11= ( d3 - d11) *  1.763842529;
                c4 = d4 + d12; c12= ( d4 - d12) *  0.196034281;
                c5 = d5 + d13; c13= ( d5 - d13) *  0.580569355;
                c6 = d6 + d14; c14= ( d6 - d14) *  1.268786568;
                c7 = d7 + d15; c15= ( d7 - d15) *  0.942793474;
/* step 3: 4-wide butterflies                */
                d0 = c0 + c4 ; d4 = ( c0 - c4 ) *  1.961570560;
                d1 = c1 + c5 ; d5 = ( c1 - c5 ) *  1.662939225;
                d2 = c2 + c6 ; d6 = ( c2 - c6 ) *  0.390180644;
                d3 = c3 + c7 ; d7 = ( c3 - c7 ) *  1.111140466;
                d8 = c8 + c12; d12= ( c8 - c12) *  1.961570560;
                d9 = c9 + c13; d13= ( c9 - c13) *  1.662939225;
                d10= c10+ c14; d14= (c10 - c14) *  0.390180644;
                d11= c11+ c15; d15= (c11 - c15) *  1.111140466;
/* step 4: 2-wide butterflies                */
                {
                float rb8  = 1.847759065;
                float rb24 = 0.765366865;
/**/ c0 = d0 + d2 ; c2 = ( d0 - d2 ) *  rb8;
```

```
                        c1 = d1 + d3 ; c3 = ( d1 - d3 ) * rb24;
/**/            c4 = d4 + d6 ; c6 = ( d4 - d6 ) *  rb8;
                        c5 = d5 + d7 ; c7 = ( d5 - d7 ) * rb24;
/**/            c8 = d8 + d10; c10= ( d8 - d10) *  rb8;
                        c9 = d9 + d11; c11= ( d9 - d11) * rb24;
/**/            c12= d12+ d14; c14= (d12 - d14) *  rb8;
                        c13= d13+ d15; c15= (d13 - d15) * rb24;
                }
/* step 5: 1-wide butterflies           */
                {
                float rb16 = 1.414213562;
/* this is a little 'hacked up'         */
                d0 = (-c0 -c1) * 2; d1 = ( c0 - c1 ) * rb16;
                d2 = c2 + c3; d3 = ( c2 - c3 ) * rb16;
                d3 -= d2;
                d4 = c4 +c5; d5 = ( c4 - c5 ) * rb16;
                d5 += d4;
                d7 = -d5;
                d7 += ( c6 - c7 ) * rb16; d6 = +c6 +c7;
                d8 = c8 + c9 ; d9 = ( c8 - c9 ) * rb16;
                d11= +d8 +d9;
                d11 +=(c10 - c11) * rb16; d10= c10+ c11;
                d12 = c12+ c13; d13 = (c12 - c13) * rb16;
                d13 += -d8-d9+d12;
                d14 = c14+ c15; d15 = (c14 - c15) * rb16;
                d15-=d11;
                d14 += -d8 -d10;
                }
/* step 6: final resolving & reordering * the other 32 are stored for use with the next granule */
    u_p = (float (*)[16]) &u[0][div][0][start];
/*16*/  u_p[ 0][0] =+d1 ;
    u_p[ 2][0] = +d9 -d14;
/*20*/  u_p[ 4][0] = +d5 -d6;
    u_p[ 6][0] = -d10 +d13;
/*24*/  u_p[ 8][0] =d3;
    u_p[10][0] = -d8 -d9 +d11 -d13;
/*28*/  u_p[12][0] = +d7;
    u_p[14][0] = +d15;
    u_p[16][0] = 0;
/* the other 32 are stored for use with the next granule  */
    u_p = (float (*)[16]) &u[0][!div][0][start];
/*0*/   u_p[16][0] = d0;
    u_p[14][0] = -(+d8 );
/*4*/   u_p[12][0] = -(+d4 );
    u_p[10][0] = -(-d8 +d12 );
/*8*/   u_p[ 8][0] = -(+d2 );
    u_p[ 6][0] = -(+d8 +d10 -d12 );
/*12*/  u_p[ 4][0] = -(-d4 +d6 );
    u_p[ 2][0] = -d14;
    u_p[ 0][0] = -d1;
}
            float c0,c1,c2,c3,c4,c5,c6,c7,c8,c9,c10,c11,c12,c13,c14,c15;
/* memory separation, second part */
/* 2 */
    c0=d16 + d24; c8= (d16 - d24)  *  1.990369453;
    c1=d17 + d25; c9= (d17 - d25)  *  1.913880671;
    c2=d18 + d26; c10= (d18 - d26) *  1.546020907;
    c3=d19 + d27; c11= (d19 - d27) *  1.763842529;
    c4=d20 + d28; c12= (d20 - d28) *  0.196034281;
    c5=d21 + d29; c13= (d21 - d29) *  0.580569355;
    c6=d22 + d30; c14= (d22 - d30) *  1.268786568;
    c7=d23 + d31; c15= (d23 - d31) *  0.942793474;
/* 3 */
    d16= c0+ c4; d20= (c0 - c4) *  1.961570560;
    d17= c1+ c5; d21= (c1 - c5) *  1.662939225;
    d18= c2+ c6; d22= (c2 - c6) *  0.390180644;
    d19= c3+ c7; d23= (c3 - c7) *  1.111140466;
    d24= c8+ c12; d28= (c8 - c12)  *  1.961570560;
    d25= c9+ c13; d29= (c9 - c13)  *  1.662939225;
    d26= c10+ c14; d30= (c10 - c14) *  0.390180644;
    d27= c11+ c15; d31= (c11 - c15) *  1.111140466;
/* 4 */
```

```
        {
        float rb8  = 1.847759065;
        float rb24 = 0.765366865;
/**/    c0= d16+d18; c2= (d16 - d18) *  rb8;
        c1= d17+d19; c3= (d17 - d19) * rb24;
/**/    c4= d20+d22; c6= (d20 - d22) *  rb8;
        c5= d21+d23; c7= (d21 - d23) * rb24;
/**/    c8= d24+d26; c10= (d24 - d26) *  rb8;
        c9= d25+d27; c11= (d25 - d27) * rb24;
/**/    c12= d28+d30; c14= (d28 - d30) *  rb8;
        c13= d29+d31; c15= (d29 - d31) * rb24;
        }
/* 5 */
        {
        float rb16 = 1.414213562;
        d16= c0+c1; d17= (c0 - c1) * rb16;
        d18= c2+c3; d19= (c2 - c3) * rb16;
        d20= c4+c5; d21= (c4 - c5) * rb16;
        d20+=d16; d21+=d17;
        d22= c6+c7; d23= (c6 - c7) * rb16;
        d22+=d16; d22+=d18;
        d23+=d16; d23+=d17; d23+=d19;
        d24= c8+c9; d25= (c8 - c9) * rb16;
        d26= c10+c11; d27= (c10 - c11) * rb16;
        d26+=d24;
        d27+=d24; d27+=d25;
        d28= c12+c13; d29= (c12 - c13) * rb16;
        d28-=d20; d29+=d28; d29-=d21;
        d30= c14+c15; d31= (c14 - c15) * rb16;
        d30-=d22;
        d31-=d23;
                }
/* step 6: final resolving & reordering * the other 32 are stored for use with the next granule */
                u_p = (float (*)[16]) &u[0][!div][0][start];
                u_p[ 1][0] = -(+d30 );
                u_p[ 3][0] = -(+d22 -d26 );
                u_p[ 5][0] = -(-d18 -d20 +d26 );
                u_p[ 7][0] = -(+d18 -d28 );
                u_p[ 9][0] = -(+d28 );
                u_p[11][0] = -(+d20 -d24 );
                u_p[13][0] = -(-d16 +d24 );
                u_p[15][0] = -(+d16 );
/* the other 32 are stored for use with the next granule          */
                u_p = (float (*)[16]) &u[0][div][0][start];
                u_p[15][0] = +d31;
                u_p[13][0] = +d23 -d27;
                u_p[11][0] = -d19 -d20 -d21 +d27;
                u_p[ 9][0] = +d19 -d29;
                u_p[ 7][0] = -d18 +d29;
                u_p[ 5][0] = +d18 +d20 +d21 -d25 -d26;
                u_p[ 3][0] = -d17 -d22 +d25 +d26;
                u_p[ 1][0] = +d17 -d30;
                }
                }
/* we're doing dewindowing and calculating final samples now */
                {
/* char *samples = (&sample_buffer[f>>(2-nch)][nch==2?0:(f&1?16:0)][ch]); */
                 int out, j;
    const float *dewindow = t_dewindow[0] + 16 - start;   /* ask my advisor? */
/* These optimisations are tuned specifically for architectures without autoincrement and -decrement. */
                 {
                  float (*u_ptr)[16] = u[0][div];
                  for (j = 0; j < 16; ++j) {
                   float outf1, outf2, outf3, outf4;
                   outf1  = u_ptr[0][ 0] * dewindow[0x0];
                   outf2  = u_ptr[0][ 1] * dewindow[0x1];
                   outf3  = u_ptr[0][ 2] * dewindow[0x2];
                   outf4  = u_ptr[0][ 3] * dewindow[0x3];
                   outf1 += u_ptr[0][ 4] * dewindow[0x4];
                   outf2 += u_ptr[0][ 5] * dewindow[0x5];
                   outf3 += u_ptr[0][ 6] * dewindow[0x6];
                   outf4 += u_ptr[0][ 7] * dewindow[0x7];
```

```
        outf1 += u_ptr[0][ 8] * dewindow[0x8];
        outf2 += u_ptr[0][ 9] * dewindow[0x9];
        outf3 += u_ptr[0][10] * dewindow[0xa];
        outf4 += u_ptr[0][11] * dewindow[0xb];
        outf1 += u_ptr[0][12] * dewindow[0xc];
        outf2 += u_ptr[0][13] * dewindow[0xd];
        outf3 += u_ptr[0][14] * dewindow[0xe];
        outf4 += u_ptr[0][15] * dewindow[0xf];
        out = (outf1 + outf2 + outf3 + outf4)/1024;
        dewindow += 32;
        u_ptr++;
                sample_mono[j] = put_sample(out);                }
      if (div & 0x1) {
        {
                float outf2, outf4;
                outf2  = u_ptr[0][ 0] * dewindow[0x0];
                outf4  = u_ptr[0][ 2] * dewindow[0x2];
                outf2 += u_ptr[0][ 4] * dewindow[0x4];
                outf4 += u_ptr[0][ 6] * dewindow[0x6];
                outf2 += u_ptr[0][ 8] * dewindow[0x8];
                outf4 += u_ptr[0][10] * dewindow[0xa];
                outf2 += u_ptr[0][12] * dewindow[0xc];
                outf4 += u_ptr[0][14] * dewindow[0xe];
                out = (outf2 + outf4)/1024;
                sample_mono[j] = put_sample(out);
        }
        dewindow -= 48;
        dewindow += start;
        dewindow += start;
        for (j=17; j < 31; ++j) {
                float outf1, outf2, outf3, outf4;
                --u_ptr;
                outf1  = u_ptr[0][ 0] * dewindow[0xf];
                outf2  = u_ptr[0][ 1] * dewindow[0xe];
                outf3  = u_ptr[0][ 2] * dewindow[0xd];
                outf4  = u_ptr[0][ 3] * dewindow[0xc];
                outf1 += u_ptr[0][ 4] * dewindow[0xb];
                outf2 += u_ptr[0][ 5] * dewindow[0xa];
                outf3 += u_ptr[0][ 6] * dewindow[0x9];
                outf4 += u_ptr[0][ 7] * dewindow[0x8];
                outf1 += u_ptr[0][ 8] * dewindow[0x7];
                outf2 += u_ptr[0][ 9] * dewindow[0x6];
                outf3 += u_ptr[0][10] * dewindow[0x5];
                outf4 += u_ptr[0][11] * dewindow[0x4];
                outf1 += u_ptr[0][12] * dewindow[0x3];
                outf2 += u_ptr[0][13] * dewindow[0x2];
                outf3 += u_ptr[0][14] * dewindow[0x1];
                outf4 += u_ptr[0][15] * dewindow[0x0];
                out = (-outf1 + outf2 - outf3 + outf4)/1024;
                dewindow -= 32;
                sample_mono[j] = put_sample(out);
                    }
      } else {
        {
                float outf2, outf4;
                outf2  = u_ptr[0][ 1] * dewindow[0x1];
                outf4  = u_ptr[0][ 3] * dewindow[0x3];
                outf2 += u_ptr[0][ 5] * dewindow[0x5];
                outf4 += u_ptr[0][ 7] * dewindow[0x7];
                outf2 += u_ptr[0][ 9] * dewindow[0x9];
                outf4 += u_ptr[0][11] * dewindow[0xb];
                outf2 += u_ptr[0][13] * dewindow[0xd];
                outf4 += u_ptr[0][15] * dewindow[0xf];
                out = (outf2 + outf4)/1024;
                sample_mono[j] = put_sample(out);
        }
        dewindow -= 48;
        dewindow += start;
        dewindow += start;
        for (j=17; j < 31; ++j) {
                float outf1, outf2, outf3, outf4;
                --u_ptr;
```

```
                        outf1  = u_ptr[0][ 0] * dewindow[0xf];
                        outf2  = u_ptr[0][ 1] * dewindow[0xe];
                        outf3  = u_ptr[0][ 2] * dewindow[0xd];
                        outf4  = u_ptr[0][ 3] * dewindow[0xc];
                        outf1 += u_ptr[0][ 4] * dewindow[0xb];
                        outf2 += u_ptr[0][ 5] * dewindow[0xa];
                        outf3 += u_ptr[0][ 6] * dewindow[0x9];
                        outf4 += u_ptr[0][ 7] * dewindow[0x8];
                        outf1 += u_ptr[0][ 8] * dewindow[0x7];
                        outf2 += u_ptr[0][ 9] * dewindow[0x6];
                        outf3 += u_ptr[0][10] * dewindow[0x5];
                        outf4 += u_ptr[0][11] * dewindow[0x4];
                        outf1 += u_ptr[0][12] * dewindow[0x3];
                        outf2 += u_ptr[0][13] * dewindow[0x2];
                        outf3 += u_ptr[0][14] * dewindow[0x1];
                        outf4 += u_ptr[0][15] * dewindow[0x0];
                        out = (outf1 - outf2 + outf3 - outf4)/1024;
                        dewindow -= 32;
                        sample_mono[j] = put_sample(out);
}}}}
                --start;
                start &= 0xf;
                div = div ? 0 : 1;
  fwrite(&sample_mono[0],64,1,out_file);
}
/*-------------------------put_sample-------------------------------*/
short put_sample(int out)
{
  if (out>32767)
                        return 32767;
                else
                        if (out<-32768)
                                return -32768;
                        else
                                return out;
}
[]
/*------------------------------------------------------------------*/
/*-------------------------Layer3 Decode Section--------------------*/
/*------------------------------------------------------------------*/
/*-------------------------layer3_frame-----------------------------*/
int layer3_frame(struct AUDIO_HEADER *header)
{
int gr,sb,i;
int mean_frame_size,bitrate,fs,hsize,ssize;
/* we need these later, hsize is the size of header+side_info */
                if (header->id)
                        if (header->mode==3) {
                                hsize=21;
                                }
                        else {
                                hsize=36;
                                }
                else
                        if (header->mode==3) {
                                hsize=13;
                                }
                        else {
                                hsize=21;
                                }
/* crc increases hsize by 2 */
                if (header->protection_bit==0)
                hsize+=2;
  bitrate=t_bitrate[header->id][3-header->layer][header->bitrate_index];
                fs=t_sampling_frequency[header->id][header->sampling_frequency];
                if (header->id)
                mean_frame_size=144000*bitrate/fs;
                else
                mean_frame_size=72000*bitrate/fs;
                data = 8 * ((append - info.main_data_begin) & (BUFFER_SIZE-1));
  fillbfr(mean_frame_size + header->padding_bit - hsize);/* read into the buffer all bytes up to the start of next heade */
/* these two should go away */
```

```
        t_l=&t_b8_l[header->id][header->sampling_frequency][0];    /* keep index end of scale factor band 0-20 */
        t_s=&t_b8_s[header->id][header->sampling_frequency][0];
/* decode the scalefactors and huffman data  */
        for (gr=0;gr < ((header->id) ? 2 : 1);gr++){  /* if id  bound =2 else bound 1 */
                                int win_type; /* same as in the standard, long=0, start=1 ,.... */
                                int window_switching_flag = info.window_switching_flag[gr][0];
                                int block_type = info.block_type[gr][0];
                                int mixed_block_flag = info.mixed_block_flag[gr][0];
                                ssize=decode_scalefactors(&info,gr);
/* decode_huffman_data  */
                                decode_huffman_data(&info,gr,ssize);
/* requantized mono  */
        requantize_mono(gr,&info,header);
/* antialiasing butterflies  */
        if (!(window_switching_flag && block_type==2))  /* long block */
                alias_reduction();
    if (window_switching_flag && block_type==2 &&  mixed_block_flag)  /* long & short block */
        win_type=0;
    else
        if (!window_switching_flag)   /* long block */
                win_type=0;
      else                  /* long or short block */
        win_type=block_type;
/* Imdct  */
        for (sb=0;sb<2;sb++)
        imdct(win_type,sb);
    if (window_switching_flag && block_type==2 && mixed_block_flag)
        win_type=2;             /* long & short block */
        for (sb=2;sb<32;sb++)
                imdct(win_type,sb);
/* Overlapping and adding  */
for (;sb<32;sb++)
        for (i=0;i<18;i++) {
                res[sb][i]=s[sb][i];
                s[sb][i]=0.0f;
                }
/* Synthesis filterbank  */
        for (i=0;i<18;i++)
        poly1(i); /* end 1 ch */
        }                       /* end of "for loop" of "gr" */
        return 0;     /*"0" status is ok */
}
/*-------------------------------------------------------------*/
```

กิตติกรรมประกาศ

ขอขอบคุณ

- อาจารย์ ชินภัทร นันทจิวากรชัยที่ให้คำปรึกษาแนะนำ
- ท่านคณาจารย์ภาคอิเล็กทรอนิกส์ทุกท่านที่ให้คำปรึกษา
- หนังสือทุกๆเล่มที่มีส่วนช่วยในการเขียนรายงาน
- บุคคลทุกท่านที่มีส่วนช่วยให้ปริญญานิพนธ์สำเร็จลงด้วยดี

.....

(นาย พิพัฒพงศ์ เกื้อศิริกุล)

.....

(นาย สุวสิต วิทยวิจักษณ์)

ผู้จัดทำ

วันที่ ...../...../.....

หนังสืออ้างอิง

1. ธันวา ศรีประโมง, "การเขียนโปรแกรมภาษาซีสำหรับวิศวกรรม",พิมพ์ครั้งที่สี่,มหาวิทยาลัย-มหานคร.
2. Davis Pan, "A tutorial on MPEG/Audio Compression", Motorolas Inc. , 1996.
3. Davis Pan, "Digital AudioCompression" , Motorolas Inc.,1996.
4. John Watkinson, "Compression In Video&Audio", Butterworth-Heinemann LTD. ,103-126p.,1995.
5. ISO/IEC International Standard IS 11172-3, "Information Technology-Coding of Moving Pictures and Assocoated Audio for Digital Storage Media at up about 1.5 Mbits/s Part3:Audio",1993.
6. Dr. Ze-Nian Li, "Audio Compression Overview",Simon Freaser University.
7. ศ.ดร. วัลลภ สุรกำพลธร ," การประมวลผลสัญญาณเชิงเลข ",สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง,ตุลาคม พ.ศ.2533,หน้า47-73.
8. Andreas Antoniou, " Digital Filter", Mcgraw-Hill ,second edition, chapter3.
9. Emanuel C.Ifeachor(University of Plymouth)& Barrie W. Jerris(Sheffield Hallam Univcrsity),"Digital Signal Proccssing",AddisonWcslcyPublishingCompany,Chaptcr2, page47-73.
10. Taxas Instrument, "TMS320C5X User's Guide" , Houston, Texas, Januar 1993.
11. Texas Instrument, " TMS320C5X DSP Starter Kit User'Guide", Dallas, Texas, 1994.
12. Interfacing the standard Pararell Port , http://www.senet.com.au/~cpeacock.
13. นายณัฐวุฒิ ระวีวงศ์อโณทัย, นางสาว ธารทิพย์ ชุติกุลรังษี, นาย ธีระ จิระรัตนโพธิ์ชัย "วิทยานิพนธ์ ดิจิตอลซิกแนลโปรเซสซิ่ง", คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, หน้า 37-60