ภาควิชาครุศาสตร์วิศวกรรม

คณะครุศาสตร์อุตสาหกรรม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองปริญญานิพนธ์

ชื่อหัวข้อ ตัวถอดรหัสสัญญาณเสียง MP3 ด้วย DSK TMS320C31

MP3 Audio Decoder by DSK TMS320C31

ชื่อนักศึกษา

1. นายเฉลิมพงษ์	ขันเงิน	รหัสประจำตัว	44035319
2. นายชัยพัฒนพงษ์	ภูสัตย์คำ	รหัสประจำตัว	44035341
3. นายมนตรี	ศรีสง่า	รหัสประจำตัว	44035342

หลักสูตร ครุศาสตร์อุตสาหกรรมบัณฑิต **สาขาวิชา** อิเล็กทรอนิกส์และคอมพิวเตอร์

อาจารย์ที่ปรึกษา อาจารย์อำพล ทองระอา

อาจารย์ที่ปรึกษาร่วม อาจารย์กิติพงค์ มะโน

คณะกรรมการสอบปริญญานิพนธ์		ลายมือชื่อ
1. อาจารย์อำพล	ทองระอา	.....
2. อาจารย์กิติพงค์	มะโน	.....
3. อาจารย์พงษ์เกียรติ	เชษฐพิทักษ์สกุล	.....
4. อาจารย์สุชิน	อาจหาญ	.....
5. ดร.สุรสิทธิ์	ราตรี	.....

วัน/เดือน/ปีที่สอบ วันเสาร์ที่ 9 พฤศจิกายน พ.ศ. 2545 เวลา 09.00 น.

สถานที่สอบ ห้อง ค.301 คณะครุศาสตร์อุตสาหกรรม สจล.

ภาควิชารับรองแล้ว

ลงนาม.....

(ผศ.วิสุทธิ์ อธิพรธรรม)

หัวหน้าภาควิชาครุศาสตร์วิศวกรรม

วันที่ 16 เดือน พ.ย. พ.ศ. 2545

<BT4502052>

ตัวถอดรหัสสัญญาณเสียง MP3 ด้วย DSK TMS320C31

ปริญญานิพนธ์

ตัวถอดรหัสสัญญาณเสียง MP3 ด้วย DSK TMS320C31

MP3 AUDIO DECODER BY DSK TMS320C31

นายเฉลิมพงษ์ ขันเงิน
นายชัยพัฒนพงษ์ ภูสัตย์คำ
นายมนตรี ศรีสง่า

ปริญญานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรครุศาสตร์อุตสาหกรรมบัณฑิต
สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์
ภาควิชาครุศาสตร์วิศวกรรม คณะครุศาสตร์อุตสาหกรรม
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2545

ปริญญานิพนธ์

เรื่อง ตัวถอดรหัสสัญญาณเสียง MP3 ด้วย DSK TMS320C31

MP3 Audio Decoder by DSK TMS320C31

วัตถุประสงค์

1) เพื่อศึกษาเรียนรู้หลักการทำงานของตัวถอดรหัสสัญญาณเสียง MP3 ด้วย DSK TMS320C31
2) เพื่อออกแบบวงจรตัวถอดรหัสสัญญาณเสียง MP3 ด้วย DSK TMS320C31
3) เพื่อสร้างตัวถอดรหัสสัญญาณเสียง MP3 ด้วย DSK TMS320C31
4) เพื่อทดสอบตัวถอดรหัสสัญญาณเสียง MP3 ด้วย DSK TMS320C31
5) เพื่อนำตัวถอดรหัสสัญญาณเสียง MP3 ด้วย DSK TMS320C31 ไปใช้งานได้จริง

ประโยชน์ที่คาดว่าจะได้รับ

1) มีความรู้ความเข้าใจหลักการทำงานและอัลกอลิทึมของตัวถอดรหัสสัญญาณเสียง MP3 ด้วย DSK TMS320C31
2) ได้วงจรต้นแบบตัวถอดรหัสสัญญาณเสียง MP3 ด้วย DSK TMS320C31
3) ได้ชุดต้นแบบตัวถอดรหัสสัญญาณเสียง MP3 ด้วย DSK TMS320C31
4) ได้ผลการทดลองตัวถอดรหัสสัญญาณเสียง MP3 ด้วย DSK TMS320C31
5) นำตัวถอดรหัสสัญญาณเสียง MP3 ด้วย DSK TMS320C31 ไปใช้งานจริง

ชื่อหัวข้อ	ตัวถอดรหัสสัญญาณเสียง MP3 ด้วย DSK TMS320C31	
นักศึกษา	นายเฉลิมพงษ์	ขันเงิน
	นายชัยพัฒนพงษ์	ภูสัตย์คำ
	นายมนตรี	ศรีสง่า
อาจารย์ที่ปรึกษา	อาจารย์อำพล	ทองระอา
อาจารย์ที่ปรึกษาร่วม	อาจารย์กิติพงค์	มะโน
หลักสูตร	ครุศาสตร์อุตสาหกรรมบัณฑิต	
สาขาวิชา	อิเล็กทรอนิกส์และคอมพิวเตอร์	
ปีการศึกษา	2545	

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้นำเสนอตัวถอดรหัสสัญญาณเสียง MP3 ด้วย DSK TMS320C31 โดยบอร์ด DSK ใช้ชิปเบอร์ TMS320C31 เป็นตัวประมวลผลการถอดรหัส MP3 ซึ่งรับข้อมูลไฟล์ MP3 จากพอร์ตขนานของเครื่องไมโครคอมพิวเตอร์ โดยจากการทดลองสามารถถอดรหัสสัญญาณเสียงทำการถอดรหัสในส่วนหัวของข้อมูล และข้อมูลข้างเคียงได้ แต่ยังมีปัญหาในส่วนสเกลแฟกเตอร์ทำให้ไม่สามารถถอดรหัสสัญญาณเสียง MP3 ได้

Thesis Title	MP3 Audio Decoder by DSK TMS320C31	
Students	Mr. Chaloempong	Khan - ngen
	Mr. Chaiphattanaphong	Phusatchum
	Mr. Montree	Srisa - nga
Advisor	Mr. Amphon	Thongra - ar
Co – Advisor	Mr. Kitipong	Mano
Education Level	Bachelor of Science in Industrial Education	
Program in	Electronics and Computer	
Academic Year	2002	

ABSTRACT

This thesis present with MP3 Audio Decoder by DSK TMS320C31. The DSK TMS320C31 board is used central processing to decoder MP3 algorithm. The input of MP3 decoder receive from parallel of microcomputer. In conclusion the MP3 Audio Decoder by DSK TMS320C31 can be decoder MP3 algorithm head data and side information. But can not decoder MP3 have problem part scale factor.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้ลุล่วงไปด้วยดี เนื่องมาจากความร่วมมือของสมาชิกภายในกลุ่มทุกท่าน ขอขอบคุณอาจารย์อำพล ทองระอา อาจารย์กิติพงค์ มะโน และคณาจารย์ภาควิชาครุศาสตร์วิศวกรรมทุกท่านที่ให้ความอนุเคราะห์เครื่องมือ และอุปกรณ์ รวมทั้งยังให้คำแนะนำแนวความคิด ความรู้ต่างๆ แนวทางการแก้ไขปัญหา ในการจัดทำปริญญานิพนธ์ ขอขอบคุณห้องสมุดคณะครุศาสตร์อุตสาหกรรม ห้องสมุดคณะวิศวกรรมศาสตร์ สำนักหอสมุดกลาง ที่ช่วยอำนวยความสะดวกเอื้อเฟื้อสถานที่ในการค้นคว้าข้อมูล สุดท้ายที่ควรระลึกถึงอย่างยิ่ง บิดา และมารดาที่เป็นผู้ให้ความสนับสนุนด้านการศึกษา และเป็นผู้ให้กำลังใจด้วยดีตลอดมา ตั้งแต่อดีตจนถึงปัจจุบัน

สารบัญ

สารบัญ (ต่อ)

สารบัญ (ต่อ)

สารบัญ (ต่อ)

สารบัญตาราง

สารบัญตาราง (ต่อ)

สารบัญรูป

สารบัญรูป (ต่อ)

สารบัญรูป (ต่อ)

สารบัญรูป (ต่อ)

สารบัญรูป (ต่อ)

บทที่ 1
บทนำ

1.1 ความสำคัญของปริญญานิพนธ์

ในปัจจุบันการเข้ารหัสสัญญาณเสียงเป็นมาตรฐาน MP3 เป็นที่นิยมกันมาก เนื่องจากสามารถบีบอัดข้อมูลได้มาก ทำให้ประหยัดเนื้อที่ในการจัดเก็บข้อมูล ดังนั้นจึงได้ทำการศึกษาศึกษาอัลกอริทึมของการเข้ารหัสและถอดรหัสสัญญาณเสียงมาตรฐาน MP3 เพื่อเขียนโปรแกรมถอดรหัสสัญญาณเสียงมาตรฐาน MP3 เพื่อเป็นแนวทางในการพัฒนาระบบถอดรหัสสัญญาณเสียงมาตรฐาน MP3 โดยโครงงานนี้จะใช้บอร์ด DSK TMS320C31 เพื่อถอดรหัสสัญญาณเสียงมาตรฐาน MP3

1.2 ขีดความสามารถของโครงงาน

โครงงานนี้มีขีดความสามารถดังนี้

1) มีหน่วยความจำ (Memory) ไม่น้อยกว่า 128 Kword (32 Bit)

2) มีส่วนแสดงผลโดยใช้ไมโครคอลโทรลเลอร์ (Microcontroller)

3) สามารถรับข้อมูล MP3 จากเครื่องคอมพิวเตอร์ผ่านทางพอร์ตขนานได้

4) สามารถถอดรหัสสัญญาณเสียงมาตรฐาน MP3 ได้

5) สามารถเลือกไฟล์ (File) ข้อมูล MP3 ในการถอดรหัสได้

1.3 เนื้อหาโดยสังเขป

เนื้อหาภายในปริญญานิพนธ์ฉบับนี้แบ่งออกเป็นบทต่างๆ เพื่อสะดวกต่อการศึกษาและทำความเข้าใจ ในแต่ละบทประกอบด้วยเนื้อหาต่อไปนี้

บทที่ 1 ความสำคัญของปริญญานิพนธ์ ขีดความสามารถของโครงงาน เนื้อหาโดยสังเขปของตัวถอดรหัสสัญญาณเสียง MP3 ด้วย DSK TMS320C31

บทที่ 2 ทฤษฎีและหลักการ ประกอบด้วยเนื้อหาในทางทฤษฎีที่เกี่ยวข้อง ซึ่งทำให้ผู้อ่านได้มีความรู้ความเข้าใจที่เป็นพื้นฐานของตัวประมวลสัญญาณเชิงเลข (TMS320C31) อันจะเป็นประโยชน์ต่อการทำความเข้าใจเกี่ยวกับโปรแกรมที่ใช้งานจริงต่อไป

บทที่ 3 การออกแบบ การสร้าง และการทำงาน กล่าวถึงการสร้าง การออกแบบฮาร์ดแวร์และซอฟต์แวร์ รวมทั้งหลักการทำงานในส่วนต่างๆ ซึ่งทำให้ผู้อ่านมีความเข้าใจการทำงานโดยรวมของโครงงานนี้

บทที่ 4 การทดลอง และผลการทดลอง กล่าวถึงขั้นตอนการทดลอง การทดสอบประสิทธิภาพในการทำงานของฮาร์ดแวร์และซอฟต์แวร์ของโครงงานนี้ เพื่อตรวจสอบว่าโครงงานนี้มีความสามารถทำงานได้ตรงตามวัตถุประสงค์หรือไม่

บทที่ 5 บทสรุป ปัญหา แนวทางแก้ไข และพัฒนา เป็นการสรุปผลการทำงานและได้เสนอแนะแนวทางในการแก้ไข และแนวทางในการพัฒนาให้มีประสิทธิภาพสูงขึ้น และการใช้งานได้อย่างกว้างขวางมากขึ้นต่อไป

ภาคผนวก ก เครื่องต้นแบบ

ภาคผนวก ข วงจรและแผ่นวงจรพิมพ์

ภาคผนวก ค แผนผังการทำงานและโปรแกรม

ภาคผนวก ง รายการอุปกรณ์

ภาคผนวก จ รายละเอียด และคุณสมบัติของอุปกรณ์

บทที่ 2
ทฤษฎีและหลักการ

2.1 การบีบอัดข้อมูลเสียงแบบ MPEG (MPEG Audio Compression)

การบีบอัดข้อมูลเสียงแบบ MPEG (MPEG Audio Compression) เป็นวิธีการบีบอัดข้อมูลเสียงแบบดิจิตอล (Digital Compression) ที่มีความเหมือนจริงสูง (High Fidelity : HiFi) การบีบอัดข้อมูลเสียงแบบ MPEG สำเร็จลงในปี 1993 โดยคณะกรรมการสากลแห่งการเชี่ยวชาญการบีบอัดเสียงอย่างเหมือนจริง ซึ่งรู้จักกันดีในชื่อว่า Motion Picture Expert Group (MPEG) และมาตรฐาน ISO/IEC 1172-3

2.1.1 หลักการพื้นฐานและการใช้งาน

หลักการบีบอัดข้อมูลแบบ MPEG จะใช้ประโยชน์จากขีดจำกัดในการได้ยินเสียงของมนุษย์โดยไม่จัดเก็บข้อมูลเสียงที่มนุษย์ฟังไม่ได้ยิน เพื่อให้ข้อมูลมีขนาดเล็กลง ซึ่งใช้หลักการว่า ถ้าเราได้ยินเสียง 2 เสียงที่มีความถี่ใกล้เคียงกัน โดยให้เสียงที่ 2 ดังกว่า จะได้ยินเสียงที่ 2 เพียงเสียงเดียว ดังนั้นในการจัดเก็บข้อมูลสามารถตัดข้อมูลเสียงที่ค่อยกว่าออกไปได้บางส่วน ซึ่งหลักการนี้จะถูกอธิบายในหัวข้อไซโคอะคูสติก (Psychoacoustic) อีกครั้ง การบีบอัดข้อมูลแบบ MPEG สามารถเลือกวิธีการบีบอัดได้หลายๆ แบบ ดังนี้

1) อัตราการสุ่มข้อมูล (Sampling Rate) 32, 44.1 หรือ 48 กิโลเฮิรตซ์

2) รองรับระบบเสียงได้ทั้ง 1 และ 2 ช่องสัญญาณเสียง ซึ่งมีอยู่ 4 แบบ ดังนี้

2.1) แบบโมโนเดี่ยว (Monophonic Mode)

2.2) แบบโมโนคู่ (Dual Monophonic Mode)

2.3) แบบสเตอริโอ (Stereo Mode)

2.4) แบบจอยท์สเตอริโอ (Joint Stereo Mode)

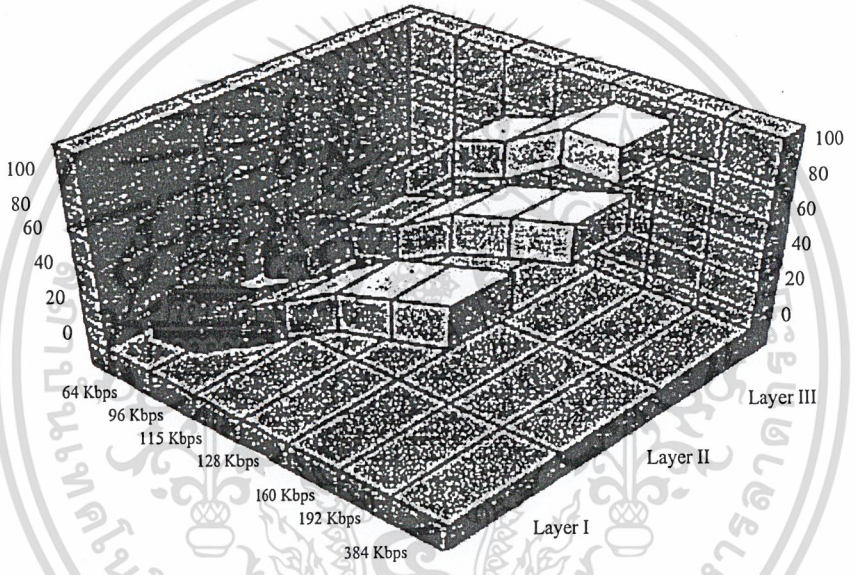
3) สามารถเลือกค่าอัตราการส่งข้อมูล (Bit Rate) ได้ตั้งแต่ 32 ถึง 244 Kbit/sec ต่อหนึ่งช่องสัญญาณเสียง ขึ้นอยู่กับอัตราการสุ่มตัวอย่าง (Sampling Rate)

รูปแบบการเข้ารหัสไฟล์ MPEG มี 3 เลเยอร์ (Layer) คือ เลเยอร์ 1 เลเยอร์ 2 และ เลเยอร์ 3 แต่ละเลเยอร์มีลักษณะต่างกันดังนี้

1) เลเยอร์ 1 มีความซับซ้อนน้อยที่สุด ต้องใช้อัตราการส่งข้อมูลสูงถึง 384 กิโลบิตต่อวินาที (Kbps) เพื่อที่จะให้ได้เสียงคุณภาพเสียงสูง

2) เลเยอร์ 2 มีความซับซ้อนมากขึ้น คุณภาพเสียงสูงกว่า เลเยอร์ 1 ใช้อัตราการส่งข้อมูลที่ 160 กิโลบิตต่อวินาทีจะได้เสียงคุณภาพสูง และที่ 192 กิโลบิตต่อวินาทีจะได้คุณภาพเสียงที่ไม่แตกต่างจากเสียงต้นแบบ

3) เลเยอร์ 3 มีความซับซ้อนมากที่สุดแต่สามารถให้คุณภาพเสียงที่ดีที่สุด ใช้อัตราการส่งข้อมูลที่ 160 กิโลบิตต่อวินาทีให้คุณภาพเสียงที่ไม่แตกต่างจากเสียงต้นแบบ สามารถแสดงความสัมพันธ์ของอัตราการส่งข้อมูลเสียง(กิโลบิตต่อวินาที)สำหรับการเข้ารหัสเลเยอร์ต่างๆ กับคุณภาพเสียงต้นแบบได้ดังรูปที่ 2.1



รูปที่ 2.1 กราฟเปรียบเทียบอัตราส่วนการบีบอัดเสียงของการเข้ารหัสเลเยอร์ต่างๆ เมื่อคุณภาพเสียงต้นแบบจากคอมแพ็คดิสก์ (CD)

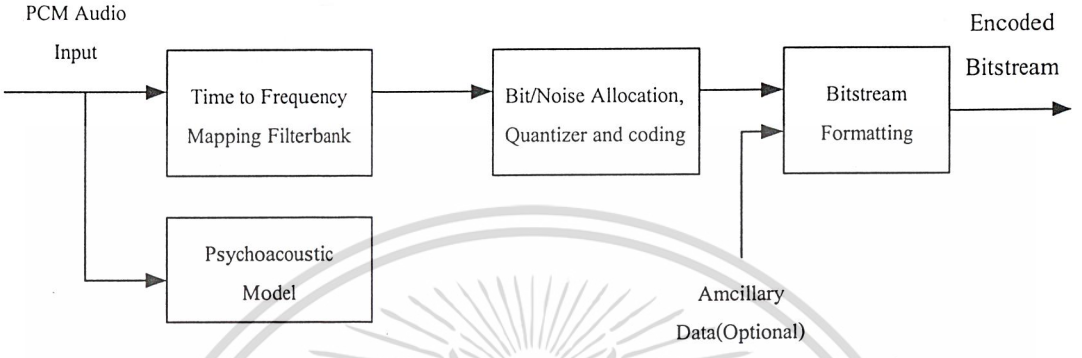
2.1.2 การเข้ารหัสแบบ MPEG โดยทั่วไป

การเข้ารหัส MPEG โดยทั่วไป จะตัดข้อมูลเสียงที่จัดเก็บบางส่วนออกแต่สามารถคงรายละเอียดของเสียงที่ได้ยินไว้เท่าเดิม ดังรูปที่ 2.2 แสดงแผนผังการทำงาน (Block Diagram) ของการเข้ารหัสและถอดรหัสแบบ MPEG กระบวนการเข้ารหัสมีขั้นตอนดังต่อไปนี้

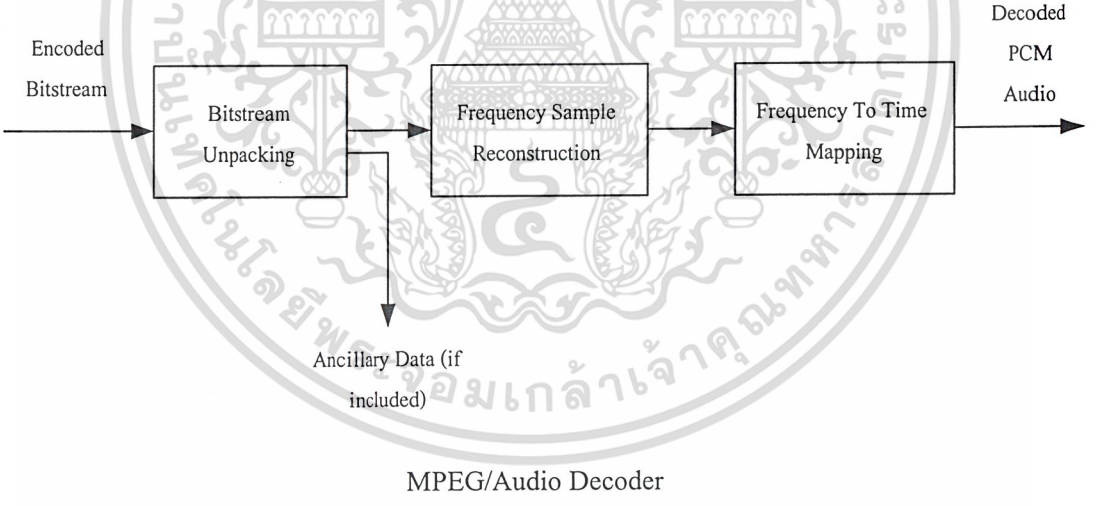
1) เริ่มจากข้อมูลเสียงจะถูกส่งไปยังโพลีเฟสฟลเตอร์แบงค์ (The Polyphase Filter Bank) เพื่อแบ่งข้อมูลเสียงออกเป็นหลายๆ ช่วงความถี่ (Subband of Frequency)

2) ของสัญญาณต่อการกำหนดค่ามาสก์กิงเทรดโฮล (Masking Threshold : SMR) ของแต่ละช่วงความถี่

3) ข้อมูลจากโพลีเฟส ฟิลเตอร์แบงค์ ถูกส่งไปส่วนแยกบิต (Bit/Noise Allocation) และปรับข้อมูล (Quantizing)



รูปที่ 2.2 แผนผังการทำงานของการเข้ารหัสแบบ MPEG



รูปที่ 2.3 แผนผังการทำงานของการถอดรหัสแบบ MPEG

กระบวนการถอดรหัสบิตสตรีม (Bitstream) จะนำข้อมูลมาผ่านกระบวนการปรับข้อมูลย้อนกลับและทำการรวมข้อมูลแต่ละช่วงความถี่กลับเป็นข้อมูลเสียง

2.1.3 โพลีเฟส ฟิลเตอร์แบงค์ (The Polyphase Filter Bank)

โพลีเฟส ฟิลเตอร์แบงค์ (The Polyphase Filter Bank) เป็นหลักการทั่วไปของการเข้ารหัสแบบ MPEG โพลีเฟส ฟิลเตอร์แบงค์ จะแบ่งสัญญาณเสียงออกเป็นช่วงความถี่ที่มีความกว้าง (Bandwidth) ออกเป็น 32 ช่วงความถี่ย่อย (Subband) ที่เท่ากัน

1) คุณสมบัติของโพลีเฟส ฟิลเตอร์แบงค์

1.1) ความกว้างของช่วงความถี่แต่ละช่วงเท่ากัน

1.2) ฟิลเตอร์แบงค์ และการแปลงกลับเป็นการแปลงกลับแบบมีการสูญเสีย (Lossy)

1.3) ย่านความถี่แต่ละย่านจะมีการเหลื่อมซึ่งกันและกัน

2) ขั้นตอนการทำโพลีเฟส ฟิลเตอร์แบงค์

ขั้นตอนการทำโพลีเฟส ฟิลเตอร์แบงค์ เป็นกระบวนการในการแยกสัญญาณที่ถูกสุ่มตัวอย่าง (Sampling) ด้วยความถี่สุ่มตัวอย่าง (Fs) ออกเป็น 32 ย่านความถี่ย่อย โดยแต่ละช่วงความถี่ย่อยเหล่านี้จะมีช่วงความถี่เท่ากัน Fs/32 ขั้นตอนการวิเคราะห์มีดังนี้

2.1) ข้อมูลเสียงเข้า 32 ข้อมูลสุ่มตัวอย่าง (Sampling)

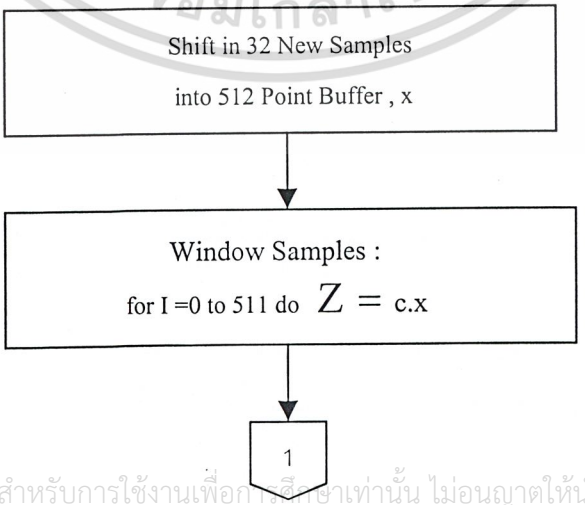
2.2) สร้างข้อมูลบัฟเฟอร์ (Buffer) ไว้สำหรับข้อมูลสุ่มตัวอย่าง 512 ข้อมูล โดยจะเลื่อนข้อมูลเข้า 32 ข้อมูล เข้ามาในบัฟเฟอร์ ในตำแหน่งที่ 0-31

2.3) คูณค่าข้อมูลอินพุต 512 ค่า กับสัมประสิทธิ์ (C) 512 ค่า

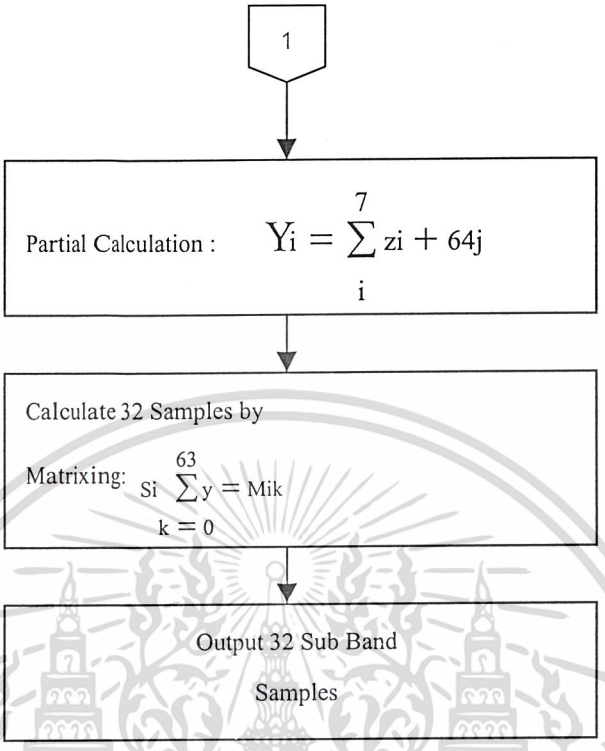
2.4) คำนวณค่า Y ตามสมการที่ให้มา (64 ค่า)

2.5) คำนวณค่า Subband Sample : S 32 ค่าออกมา

อธิบายขั้นตอนการทำโพลีเฟส ฟิลเตอร์แบงค์ เป็นผังงานของโปรแกรมได้ดังรูปที่ 2.4 แสดงผังงานของโปรแกรมอธิบายขั้นตอนการทำโพลีเฟส ฟิลเตอร์แบงค์



รูปที่ 2.4 ผังงานของโปรแกรมอธิบายขั้นตอนการทำโพลีเฟส ฟิลเตอร์แบงค์



รูปที่ 2.4 (ต่อ) ผังงานของโปรแกรมอธิบายขั้นตอนการทำโพลีเฟส ฟิลเตอร์แบงค์

$$St[i] = \sum_{k=0}^{63} \sum_{j=0}^{7} M[i][k] \times \left(C[k + 64] \times x[k + 64j] \right) \tag{2.1}$$

โดย

i คือ ดรรชนีด้านความถี่ตั้งแต่ 0-31

St[i] คือ สัญญาณเอาต์พุตที่ถูกกรองความถี่แล้ว สำหรับย่านความถี่ i ที่เวลา t โดยที่ t เป็นเลขจำนวนเต็ม

C[n] คือ สัมประสิทธิ์ของวินโดว์ ลำดับที่ n ถูกนิยามไว้ในมาตรฐาน

X[n] คือ สัญญาณเสียงที่ถูกอ่านจากอินพุตบัฟเฟอร์ที่เก็บอินพุตไว้ 512 สัญญาณความถี่

สมการที่ 2.1 เป็นการคำนวณที่สามารถลดจำนวนครั้งการคูณได้มากที่สุด เนื่องจากพจน์ที่อยู่ในวงเล็บไม่เป็นฟังก์ชั่นของ i และ *M[i][k]* ไม่ขึ้นกับค่า j ดังนั้นการคำนวณทุกๆ 32 สัญญาณสุ่ม

$$M[i][k] = \cos \frac{\left(2 \times i + 1\right) \times \left(n - 16\right) \times \pi}{64} \tag{2.2}$$

ตัวอย่างเอาต์พุตจะคูณเพียง 512+32*64 = 2560 ครั้ง และการบวก 64*7+32*63 = 2464 ครั้ง หรือประมาณ 80 ครั้งต่อสัญญาณเอาต์พุต 1 สัญญาณ

ข้อสังเกตจากการทำฟิลเตอร์แบงค์ทุกๆ 32 สัญญาณอินพุตเราจะได้สัญญาณเอาต์พุต 32 สัญญาณ ด้วยเหตุนี้ทั้ง 32 ย่าน เมื่ออินพุตเข้า 32 สัญญาณ

จากสมการที่ 2.1 สามารถเขียนเป็นสมการพื้นฐานในรูปแบบผลบวกประสาน (Convolution) ได้คือ

$$S_t[i] = \sum_{n=0}^{511} x[t-n] \times Hi[n] \tag{2.3}$$

$$H_t[n] = h[n] \times cos\left[\frac{\left(2 \times i + 1\right) \times \left(n - 16\right) \times \pi}{64}\right] \tag{2.4}$$

โดย

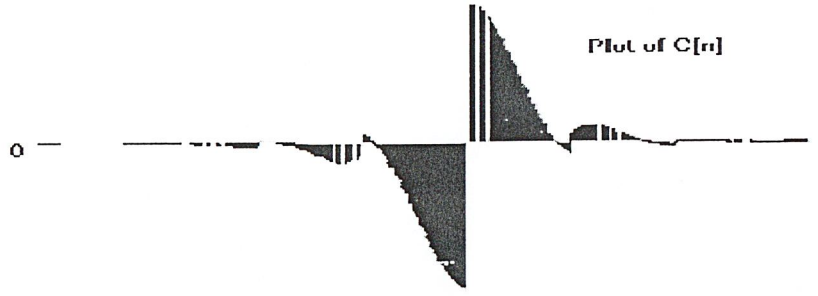
$x[t]$ คือ สัญญาณเสียงสุ่มความถี่

$h[n]$ $= -C[n]$ เมื่อ n/64 เป็นเลขคี่

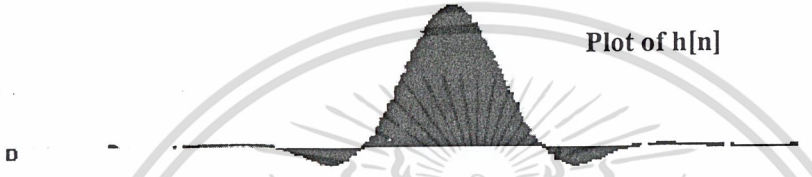
$= C[n]$ กรณีอื่น

ในรูปแบบนี้แต่ละย่านความถี่ย่อยจะมีผลตอบสนองความถี่ของตนเองคือ $H_i[n]$ แม้ว่ารูปแบบนี้จะง่ายต่อการวิเคราะห์แต่จะมีจำนวนครั้งของการคูณและบวกในการคำนวณมากกว่าสมการที่ 2.1 คือจะคูณเลข 512*32 = 16384 ครั้งและการบวก 512*32 = 16384 ครั้ง เพื่อที่จะคำนวณสัญญาณเอาต์พุต 32 สัญญาณ

ค่าสัมประสิทธิ์ $h[n]$ เป็นผลตอบสนองตัวกรองความถี่ต่ำสำหรับโพลีเฟส ฟิลเตอร์แบงค์ รูปที่ 2.5 แสดงสัมประสิทธิ์ $C[n]$ และ $h[n]$ สมการสำหรับ $H[n]$ แสดงให้เห็นว่า ค่า $H[n]$ ได้มาจาก $h[n]$ คูณกับเทอมโคไซน์ (Cosine) เพื่อที่จะเลื่อนเฟสของผลตอบสนองตัวกรองความถี่ต่ำ $h[n]$ ให้เหมาะสมกับย่านความถี่ ดังนั้น จึงเรียกการกรองความถี่แบบนี้ว่า “โพลีเฟส” (Polyphase) การกรองความถี่แบบนี้จะมีความถี่กลางอยู่ที่ค่าเลขคี่คูณด้วย $\frac{\pi}{64T}$ ซึ่งค่า T นี้เป็นคาบของการสุ่มตัวอย่าง (Sampling period)



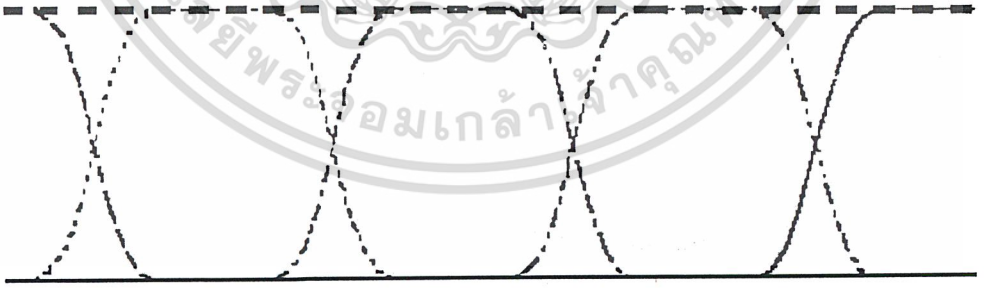
ก) สัมประสิทธิ์ C[n]



ข) สัมประสิทธิ์ h[n]

รูปที่ 2.5 การเปรียบเทียบสัมประสิทธิ์ C[n] และ h[n]

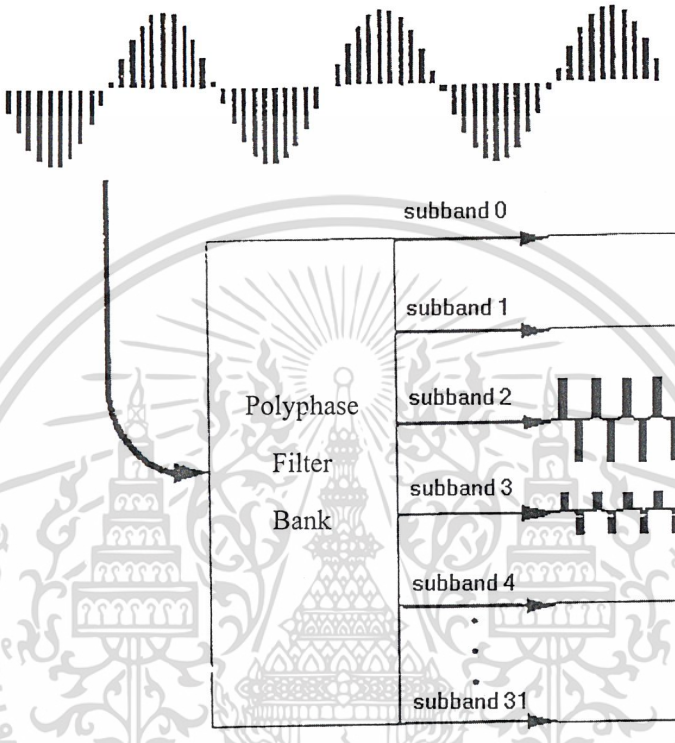
เนื่องจากความถี่ต่ำไม่มีความคมในความถี่คัทออฟ (แถบนำกว้าง) ดังนั้น เมื่อแบ่งช่วงความถี่ที่ใช้งานทั้งหมดเป็น 32 ช่วง จะเกิดการเหลื่อมซึ่งกันและกัน ดังแสดงตัวอย่างในรูปที่ 2.6



รูปที่ 2.6 การเหลื่อมซึ่งกันและกันระหว่างย่านความถี่ย่อยที่ติดกัน

ซึ่งผลการเหลื่อมกันของย่านความถี่ที่ติดกัน จะทำให้ประสิทธิภาพของการบีบอัดข้อมูลลดลงเนื่องจากพลังงานของสัญญาณที่มีความถี่ใกล้ๆ กับขอบของย่านความถี่หนึ่งๆ จะปรากฏเป็นเอาต์พุตของโพลีเฟส ฟิลเตอร์แบงค์ 2 เอาต์พุตที่อยู่ติดกัน ดังรูปที่ 2.7 แสดงตัวอย่างของสัญญาณ

รูปไซน์หนึ่งความถี่ เมื่อผ่านกระบวนการโพลีเฟส ฟิลเตอร์แบงค์ แล้วปรากฏเอาต์พุตออกมา 2 ย่านความถี่



รูปที่ 2.7 การเกิดเอาต์พุตของโพลีเฟส ฟิลเตอร์แบงค์ เมื่อสัญญาณเข้าเป็นสัญญาณหนึ่งความถี่

3) ไซโคอะคูสติก โมเดล (Psychoacoustic Model)

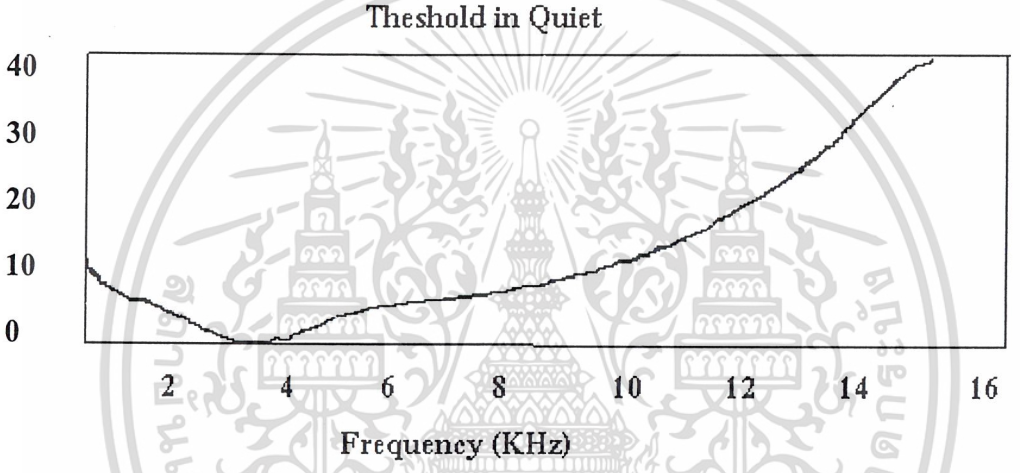
การเข้ารหัสข้อมูลเสียงแบบ MPEG ใช้หลักการตัดเสียงบางส่วนที่ฟังไม่ได้ยินออก เนื่องจากความได้เปรียบในการรับฟังเสียงของมนุษย์ที่ไม่สามารถได้ยินเสียงรบกวน (Noise) ภายใต้เงื่อนไขการปิดกั้นการได้ยิน (Auditory Masking) การปิดกั้นการได้ยินเป็นคุณสมบัติอย่างหนึ่งในการรับฟังเสียงของมนุษย์ เกิดขึ้นเมื่อเรารับฟังเสียงจากแหล่งกำเนิดสองแหล่งจะไม่สามารถได้ยินเสียงจากแหล่งกำเนิดที่ค่อยกว่าถ้าเครื่องกำเนิดทั้งสองมีความถี่ใกล้เคียงกัน ความสามารถในการได้ยินของมนุษย์

3.1) ช่วงความถี่ที่สามารถรับฟังประมาณ 20-20,000 เฮิรตซ์

3.2) ไดนามิคเรนจ์ (Dynamic Range) คือ เสียงเบาสุดถึงดังสุดที่สามารถรับฟังได้ประมาณ 96 เดซิเบล

3.3) ความถี่เสียงพูดปกติ 500-2000 เฮิรตซ์

3.4) ความไวในการรับฟังเสียงของมนุษย์ จากการทดลองให้คนหนึ่งคนเข้าไปนั่งในห้องที่เงียบสนิทแล้วค่อยๆ เพิ่มความดังเสียง จนกระทั่งคนๆ นั้นเริ่มได้ยิน (Threshold Level) วัดระดับความดังเสียง แปรความถี่แล้วนำค่ามาวาดกราฟ แสดงได้ดังรูปที่ 2.8 จะพบว่ามนุษย์สามารถรับฟังเสียงช่วงความถี่ 2-4 กิโลเฮิรตซ์ได้ไวที่สุด

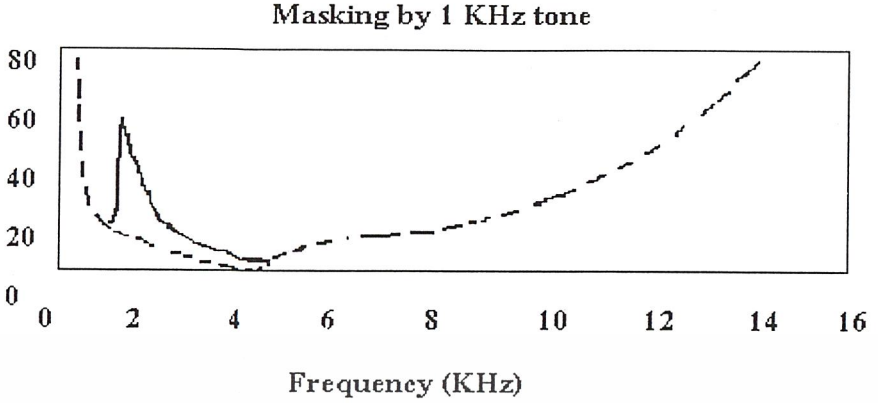


รูปที่ 2.8 ระดับความดังเสียงที่เริ่มได้ยินที่ความถี่ต่างๆ

3.5) การปิดกั้นเสียงเมื่อสัญญาณเสียงหลายๆ สัญญาณที่มีความถี่ใกล้เคียงกันเข้ามามนุษย์สามารถได้ยินเสียงที่มีความดัง (แรง) กว่าเท่านั้น หรือกล่าวได้ว่าสัญญาณที่แรงกว่าจะปิดกั้นสัญญาณที่อ่อนกว่า และเรียกสัญญาณที่แรงกว่านั้นว่า "ตัวปิดกั้น" (Masker)

จากการทดลองเรื่องการปิดกั้นเสียง โดยให้คนฟังเสียงในห้องที่มีเสียงที่มีความถี่ 1 กิโลเฮิรตซ์ ความดัง 60 เดซิเบล แล้วเล่นเสียงจากแหล่งกำเนิดอีกแหล่ง วัดระดับความดังเสียงที่ได้ยินจากแหล่งกำเนิดที่สอง แปรความถี่ วาดกราฟ ได้ดังรูปที่ 2.9 ซึ่งแสดงให้เห็นว่ามีความถี่ใกล้กับความถี่ 1 กิโลเฮิรตซ์ ระดับความดังเสียงของแหล่งกำเนิดที่ 2 ต้องมีค่าสูงเพื่อให้รับฟังเสียงได้

สามารถสรุปจากข้อมูลทั้งหมดได้ว่า มนุษย์สามารถรับฟังเสียงเป็นช่วงความถี่และรับพลังงานเสียงได้ไม่เท่ากันในแต่ละย่านความถี่ซึ่งเราเรียกว่าย่านความถี่วิกฤต (Critical Band) และในแต่ละย่านความถี่มนุษย์จะแยกสัญญาณจากแหล่งกำเนิดต่างๆ ออกจากกันได้ยาก ตารางที่ 2.1 แสดงความกว้างของย่านความถี่วิกฤติย่านต่างๆ



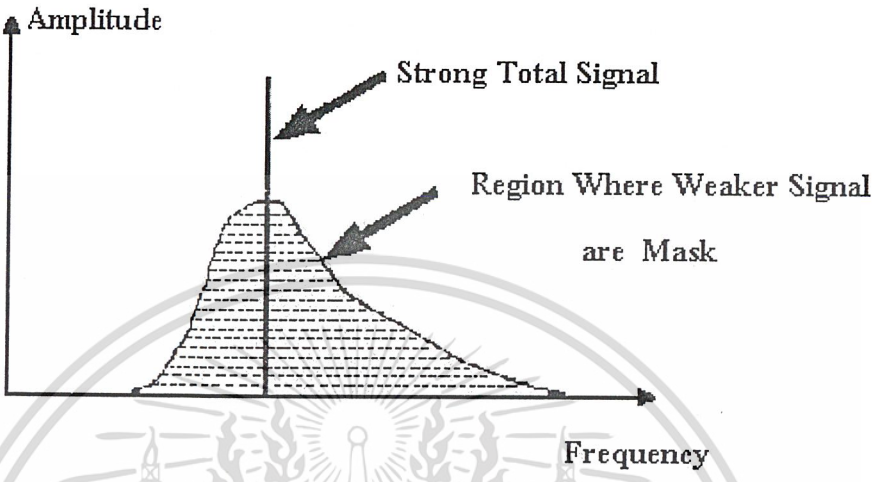
รูปที่ 2.9 ผลของการปิดกั้น

ตารางที่ 2.1 ความกว้างของความถี่วิกฤตย่านต่างๆ

จำนวนแถบ Band Number	ความถี่ Frequency (Hz)	จำนวนแถบ Band Number	ความถี่ Frequency (Hz)
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 (Noise Masking) ภายในย่านความถี่หนึ่งๆ ได้ ดังรูปที่ 2.10 หลักการนี้จะแบ่งสัญญาณเสียงออกเป็นย่านความถี่ย่อย

ประมาณเท่าย่านความถี่วิกฤตแล้วจึงปรับข้อมูลในแต่ละย่านตามความสามารถในการได้ยินในแต่ละย่านความถี่



รูปที่ 2.10 การปิดกั้นเสียงที่ไม่ได้ยินในย่านความถี่ย่อยหนึ่งๆ ของการเข้ารหัส MPEG

ไซโคอะคูสติก โมเดล (Psychoacoustic Model) ทำหน้าที่วิเคราะห์หาจุด (ความถี่) ที่ต้องปิดกั้นในย่านความถี่หนึ่ง ตัวเข้ารหัสจะใช้ข้อมูลนี้ในการตัดสินใจว่าควรแทนสัญญาณอินพุตที่เข้ามาด้วยจำนวนบิตมากน้อยเพียงใด

4) การแบ่งแยกบิต (Bit or Noise Allocation)

การแบ่งแยกบิตทำหน้าที่หาจำนวนของบิตเพื่อที่จะแยกในแต่ละย่านความถี่ย่อยขึ้นอยู่กับข้อมูลที่ได้มาจากไซโคอะคูสติกโมเดล

การแบ่งแยกบิตของการเข้ารหัส MPEG เลเยอร์ 3 มีรายละเอียดดังนี้

4.1) ปรับข้อมูลให้เป็นค่า (Quantize)

4.2) นับจำนวนบิตของการเข้ารหัสแบบฮัฟแมน (Huffman Coding)

4.3) คำนวณผลของเสียงที่ตัดออกจริงๆ (Actual Calculates the Result Noise)

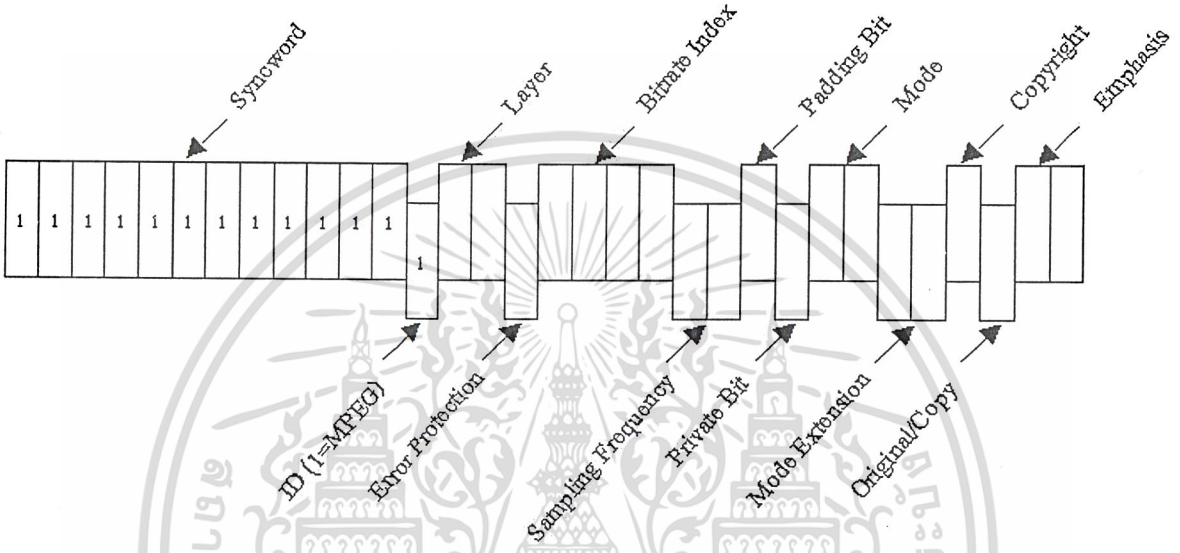
4.4) ทำขั้นตอนที่ 1-3 ซ้ำเพื่อปรับค่าให้อยู่ในระดับที่ยอมรับได้ (Iterative Loop)

5) การเข้ารูปแบบ (Formatting)

เป็นขั้นตอนการจัดรูปแบบข้อมูลให้ตรงตามมาตรฐาน ซึ่งมาตรฐานการจัดเรียงข้อมูลแบบ MPEGของเลเยอร์ 3 แสดงได้ดังรูปที่ 2.11

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

รูปที่ 2.11 รูปแบบข้อมูล MPEG เลเยอร์ 3



รูปที่ 2.12 บิตต่างๆ ในส่วนหัวข้อมูล

5.1) ส่วนหัวข้อมูล (Header) เป็นข้อมูลขนาด 32 บิต แสดงลักษณะทั่วๆ ไปของไฟล์นั้นๆ ข้อมูลในส่วนนี้ประกอบด้วยบิตต่างๆ ดังรูปที่ 2.11 ซึ่งแต่ละบิตจะมีรายละเอียดดังนี้

รายละเอียดของส่วนหัว

Syncword ส่วนรหัสของลำดับข้อมูล MPEG เท่ากับ 1111 1111 1111

ID บิตแสดงถึงมาตรฐานการเข้ารหัส ดังตารางที่ 2.2

ตารางที่ 2.2 บิตมาตรฐานของการเข้ารหัส

ID	ความหมาย
0	ไม่ใช่มาตรฐาน ISO
1	เข้ารหัสตามมาตรฐาน ISO 1172-3

Layer แสดงเลเยอร์ที่กำลังถอดรหัสอยู่ดังตารางที่ 2.3

ตารางที่ 2.3 ความหมายของรหัสข้อมูลในเลเยอร์

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

Protect Bit บอกให้ทราบว่า ข้อมูลเสริมเกี่ยวกับการตรวจสอบความผิดพลาดได้บันทึกมาด้วยหรือไม่ ดังตารางที่ 2.4

ตารางที่ 2.4 ข้อมูลเสริมเกี่ยวกับการตรวจสอบความผิดพลาด

Protect Bit	ความหมาย
0	มีข้อมูลเสริม
1	ไม่มีข้อมูลเสริม

Bitrate Index บอกอัตราการบีบอัดข้อมูล (Bitrate) ที่ใช้ ดังตารางที่ 2.3

ตารางที่ 2.5 อัตราการบีบอัดข้อมูลในแต่ละเลเยอร์

ดัชนีอัตราบิต (Bitrate Index)	อัตราบิต (Bitrate)		
	Layer I	Layer II	Layer III
0000	Free format	Free format	Freeformat
0001	32 kbit/s	32 kbit/s	32 kbit/s
0010	64 kbit/s	48 kbit/s	40 kbit/s
0011	96 kbit/s	56 kbit/s	48 kbit/s
0100	128 kbit/s	64 kbit/s	56 kbit/s

ตารางที่ 2.5 (ต่อ) อัตราการบีบอัดข้อมูลในแต่ละเลเยอร์

ดัชนีอัตราบิต (Bitrate Index)	อัตราบิต (Bitrate)		
	Layer I	Layer II	Layer III
0101	160 kbit/s	80 kbit/s	64 kbit/s
0110	192 kbit/s	96 kbit/s	80 kbit/s
0111	224 kbit/s	112 kbit/s	96 kbit/s
1000	256 kbit/s	128 kbit/s	112 kbit/s
1001	288 kbit/s	160 kbit/s	128 kbit/s
1010	320 kbit/s	192 kbit/s	160 kbit/s
1011	352 kbit/s	224 kbit/s	192 kbit/s
1100	384 kbit/s	256 kbit/s	224 kbit/s
1101	416 kbit/s	320 kbit/s	256 kbit/s
1110	448 kbit/s	384 kbit/s	320 kbit/s

Sampling Frequency บอกความถี่ในการสุ่มตัวอย่าง (Sampling) ดังตารางที่ 2.4

ตารางที่ 2.6 ความถี่ในการสุ่มตัวอย่าง

ความถี่ในการสุ่มตัวอย่าง (Sampling Frequency)	ความถี่ในการสุ่มตัวอย่าง
00	44.1 kHz.
01	48 kHz.
10	32 kHz.
11	สงวน

Padding Bit เท่ากับ "1" แสดงว่าเฟรมนั้นๆ บรรจุสล็อต (Slot) ในเลเยอร์3=1ไบต์เพิ่มเติมมา ถ้าเท่ากับ "0" แสดงว่าไม่ได้บรรจุ การบรรจุสล็อตเพื่อปรับอัตราส่วนระหว่างอัตราส่วนการบีบอัดกับความถี่ในการสุ่มตัวอย่าง Sampling Frequency ให้ลงตัว โดย Padding Bit จำเป็นสำหรับความถี่ในการสุ่มตัวอย่าง 44.1 kHz. เท่านั้น

Private ในอนาคตจะไม่ใช้ในการเข้ารหัสตามมาตรฐาน ISO/IEC 11172-3

Copyright "1" มีลิขสิทธิ์, "0" ไม่มีลิขสิทธิ์

Original/Copy "0" ข้อมูลนั้นถูก copy มา, "1" ข้อมูลนั้นเป็นต้นฉบับ

Mode บอกเกี่ยวกับรูปแบบของข้อมูลว่าเป็นหนึ่งชองเสียง (Single Channel) , สองช่องเสียง (Double Channel) , สเตอริโอ (Stereo) , จอยท์-สเตอริโอ (Joint Stereo) ดังตารางที่ 2.3 โดยในเลเยอร์ 3 จอยท์-สเตอริโอ คือ intensity และ/หรือ ms-stereo

ตารางที่ 2.7 ความหมายของรหัสข้อมูลใน Mode

โหมด (Mode)	ความหมาย
00	สเตอริโอ
01	จอยท์-สเตอริโอ
10	สองช่องเสียง
11	หนึ่งช่องเสียง

Mode_extention ใช้บอกชนิดของจอยท์-สเตอริโอ (Joint Stereo Mode) ว่ามี Ms-stereo, Intensity-stereo หรือไม่อย่างไร ดังตารางที่ 2.4

ตารางที่ 2.8 ความหมายของรหัสข้อมูลใน Mode_extention

โหมดการขยาย (Mode_extention)	ความเข้ม (Intensity)	เอ็มเอส สเตอริโอ (Ms-stereo)
00	Off	Off
01	On	Off
10	Off	On
11	On	On

Emphasis แสดงถึงชนิดเอ็มฟาไซส์ (Emphasis) ที่ใช้งาน แสดงในตารางที่ 2.5

ตารางที่ 2.9 ชนิดเอ็มฟาไซส์

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

5.2) ส่วนตรวจสอบความผิดพลาด เป็นข้อมูลขนาด 16 บิต ตรวจสอบพาริตี้ของข้อมูลที่ถูกเข้ารหัส

5.3) ส่วนข้อมูลข้างเคียงเป็นข้อมูลขนาด 17 หรือ 32 ไบต์ (17 ไบต์ สำหรับการเข้ารหัสแบบโมโนเดี่ยว, 32 ไบต์ สำหรับแบบอื่นๆ) ข้อมูลในส่วนนี้จะเก็บองค์ประกอบต่างๆ ที่ใช้ในการถอดรหัสโดยตรง

54) ข้อมูลหลักจะไม่ถูกจำกัดความยาวข้อมูล ขึ้นอยู่กับความถี่สุ่มตัวอย่างและอัตราการส่งข้อมูล ดังสมการ

$$N = 144000 \times \frac{bitrate}{sampling\_frequency} \tag{2.5}$$

เมื่อ N คือ ความยาวข้อมูลระหว่าง Syncword ที่อยู่ติดกัน

bitrate คือ อัตราการส่งข้อมูล

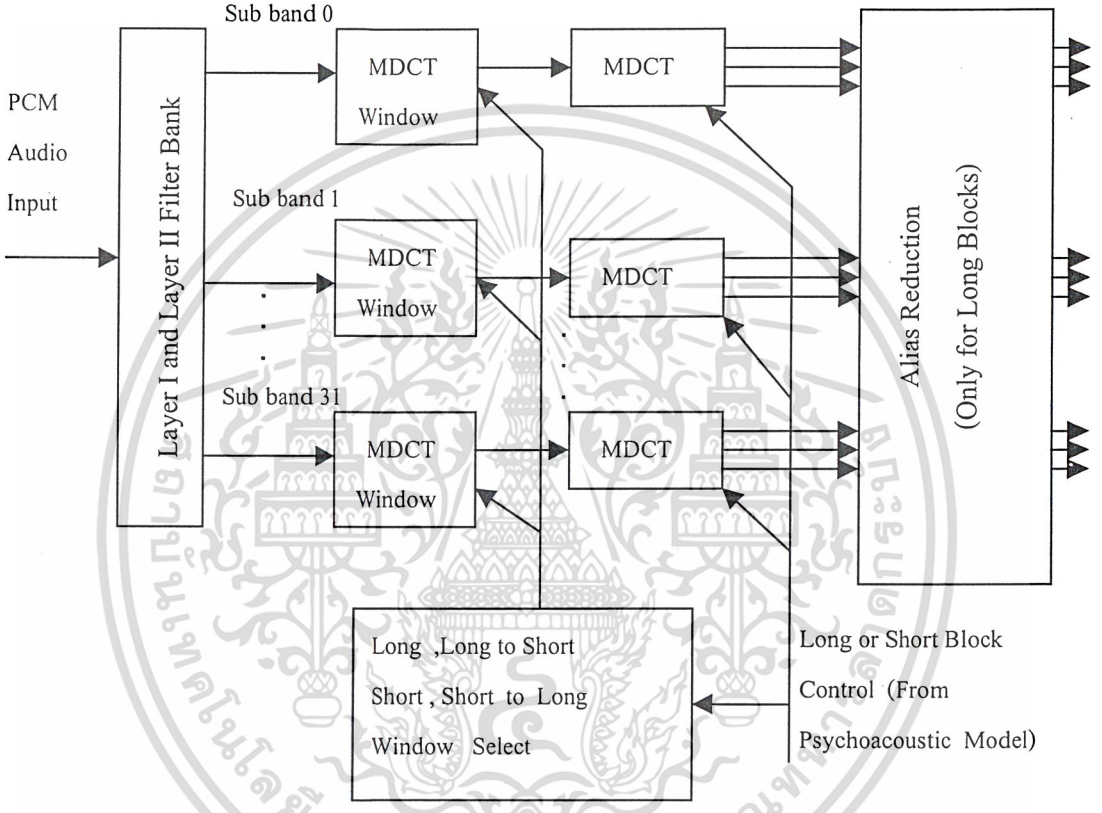
sampling_frequency คือ ความถี่สุ่มตัวอย่าง

2.1.4 การเข้ารหัสข้อมูลเอ็มเป็กเลเยอร์ 3

การเข้ารหัสข้อมูลเอ็มเป็กเลเยอร์ 3 มีหลักการเหมือนการเข้ารหัสแบบเอ็มเป็กโดยทั่วไปดังกล่าวมาแล้วในหัวข้อ 2.1.3 มีเพียงบางเทคนิคที่เพิ่มเข้ามาเพื่อให้การบีบอัดข้อมูลมีประสิทธิภาพสูงขึ้น ดังนี้

1) การใช้การแปลงแบบ MDCT (Modified Discrete Cosine Transform)

การใช้การแปลงแบบ MDCT (Modified Discrete Cosine Transform) เพื่อเพิ่มความละเอียดของสเปกตรัม หลังจากที่ผ่านขั้นตอนการทำโพลีเฟส ฟิลเตอร์แบงค์ แสดงแผนผังการทำ MDCT ดังรูปที่ 2.13



รูปที่ 2.13 แผนผังการทำ MDCT

มีรูปแบบการทำ MDCT อยู่ 2 ลักษณะ โดยแบ่งตามความยาวของบล็อก คือ

1.1) บล็อกยาวมี 18 สัญญาณสุ่มตัวอย่างแต่มีการเหลื่อมของย่านความถี่อยู่ร้อยละ 50 ดังนั้น บล็อกจึงมีความยาว 36 สัญญาณสุ่มตัวอย่าง

1.2) บล็อกสั้นมี 6 สัญญาณสุ่มตัวอย่าง แต่มีการเหลื่อมของย่านความถี่อยู่ที่ร้อยละ 50 ดังนั้น บล็อกสั้นจึงมีความยาว 12 สัญญาณสุ่มตัวอย่าง

ความแตกต่างของบล็อกยาวและบล็อกสั้นก็คือ บล็อกยาวให้ความละเอียดในด้านความถี่มาก ส่วนบล็อกสั้นให้รายละเอียดด้านเวลาสำหรับสัญญาณทรานเซียนส์ (Transient) สังเกตว่า

ความยาวของบล็อกสั้นเท่ากับหนึ่งส่วนสามของบล็อกยาว ดังนั้น ถ้าใช้บล็อกสั้นต้องใช้สามบล็อกเพื่อให้ความยาวข้อมูลเท่ากับเมื่อใช้บล็อกยาว

ในการเข้ารหัสเฟรม (หนึ่งเฟรมคือ 1152 สัญญาณสุ่มตัวอย่างอินพุต) หนึ่งๆ สามารถใช้บล็อกชนิดเดียวกันหรือบล็อกผสม (Mixed Block) ก็ได้ ถ้าเข้ารหัสแบบผสม 2 ย่านความถี่ต่ำจะต้องใช้บล็อกยาว อีก 30 ย่านความถี่ที่เหลือเป็นบล็อกสั้น ซึ่งถ้าการเข้ารหัสแบบบล็อกผสมจะให้ความละเอียดด้านความถี่สูงที่ความถี่ต่ำและความละเอียดด้านเวลาสูงที่ความถี่สูง

2) การลดผลจากการเหลื่อมของย่านความถี่ติดกัน (Alias Reduction)

3) การปรับค่าแบบไม่เป็นรูปแบบ (Non-Uniform Quantization)

4) ย่านสเกลแฟคเตอร์ (Scalefactor Band) ใช้สเกลแฟคเตอร์ต่างกันเมื่อต่างย่านความถี่

5) การเข้ารหัสข้อมูลแบบเอ็นทรอปี้ MPEG เลเยอร์ 3 ใช้การเข้ารหัสข้อมูลแบบฮัฟแมน (Huffman) มาช่วยในการบีบอัดข้อมูล

2.2 การเข้ารหัสและถอดรหัสฮัฟแมน (Huffman Coding and Decoding)

2.2.1 การเข้ารหัสฮัฟแมน (Huffman Coding)

การเข้ารหัสแบบฮัฟแมนจะใช้ทฤษฎีความน่าจะเป็นในการเข้ารหัส ความน่าจะเป็นที่กล่าวถึงนี้คือความซ้ำซ้อนของข้อมูลที่จะเกิดขึ้นในข้อมูลชุดใหญ่ๆ ชุดหนึ่งแทนที่เราจะแทนรหัสข้อมูลที่มีความซ้ำซ้อนกันมากๆ แทนที่จะเก็บด้วยข้อมูลบิตปกติ เราจะทำการเข้ารหัสฮัฟแมนด้วยข้อมูลบิตน้อยลงหรือเพิ่มขึ้นตามความน่าจะเป็นของข้อมูลที่จะเกิดขึ้นว่ามีสูงขนาดใด ถ้าความน่าจะเป็นเกิดขึ้นมากก็แทนด้วยข้อมูลบิตน้อยๆ และถ้าความน่าจะเป็นเกิดขึ้นน้อยก็แทนด้วยรหัสที่มีจำนวนบิตมากขึ้นตามหลักการของฮัฟแมนซึ่งจะกล่าวถึงโดยละเอียดต่อไป

ตัวอย่างการเข้ารหัสฮัฟแมน มีข้อมูลอยู่ชุดหนึ่งมีความน่าจะเป็นที่เกิดขึ้นตามตาราง โดยแทนข้อมูลดังกล่าวด้วยรหัสฮัฟแมน

ตารางที่ 2.10 ข้อมูลที่ต้องการเข้ารหัสฮัฟแมน

สัญลักษณ์ (Symbols)	ความน่าจะเป็น (Probability)
S0	40%
S1	24%
S2	10%
S3	9%
S4	7%
S5	5%
S6	3%
S7	2%
รวม (Total)	100%

1) หลักการเข้ารหัสฮัฟแมน

การเข้ารหัสฮัฟแมน เราจะสามารถกระทำได้ดังนี้คือ

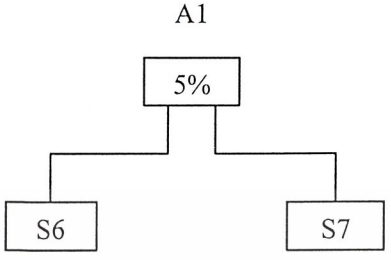
1.1) เริ่มจับคู่จากข้อมูลที่มีความน่าจะเป็นน้อยที่สุดไปยังมากที่สุด

1.2) เอาข้อมูลที่มีความน่าจะเป็นน้อยไว้ทางขวามือ

1.3) เมื่อทำการจับคู่ข้อมูลแล้วจะได้ความน่าจะเป็นของข้อมูลเพิ่มขึ้น ให้ทำการเลื่อนความน่าจะเป็นเพิ่มขึ้นไปเพื่อจับคู่กับข้อมูลตัวที่มีความน่าจะเป็นเรียงลำดับกัน

1.4) ทำการจับคู่ชุดข้อมูลจนกว่าจะมีความน่าจะเป็น 100%

2) การเขียน Binary tree เพื่อเข้ารหัสแบบฮัฟแมน



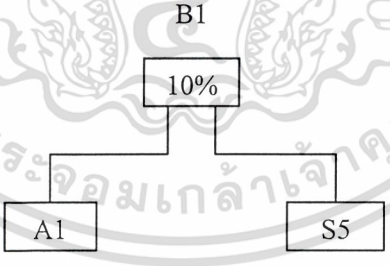
รูปที่ 2.14 การจับคู่ข้อมูลที่มีความน่าจะเป็นระหว่าง S6 และ S7

A1 เป็นรหัสเพื่อให้การจับคู่ดูง่ายในการที่จะไปจับคู่กับตัวถัดไป

5% เป็นความน่าจะเป็นใหม่ที่เกิดขึ้นจากการรวมความน่าจะเป็นของ S6 และ S7 คือ 3% และ 2%

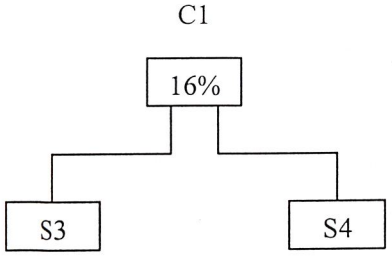
S7 อยู่ทางขวามือเพราะความน่าจะเป็นต่ำกว่า S6

เมื่อทำการจับคู่ระหว่าง S6 และ S7 จะได้ความน่าจะเป็นเพิ่มขึ้นแล้วนำไปจับกับ S5



รูปที่ 2.15 การจับคู่ข้อมูลที่มีความน่าจะเป็นระหว่าง A1 และ S5

B1 เกิดจากการรวมของ S5 และ A1 ซึ่งทั้งสองรวมกันได้ความน่าจะเป็น 10% จะได้ความน่าจะเป็นไปอยู่ต้นๆ จากนั้นทำการจับคู่ S3, S4 ซึ่งมีความน่าจะเป็น 9% และ 7%



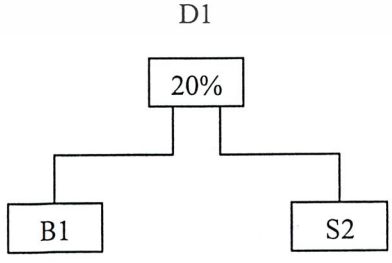
รูปที่ 2.16 การจับคู่ข้อมูลที่มีความน่าจะเป็นระหว่าง S3 และ S4

จะได้ตารางใหม่ดังนี้

ตารางที่ 2.11 ผลการจับคู่ความน่าจะเป็น

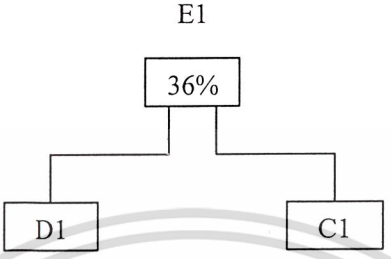
สัญลักษณ์ (Symbols)	ความน่าจะเป็น (Probability)
S0	40%
S1	24%
C1	16%
B1	10%
S2	10%

ทำการจับคู่ลำดับความน่าจะเป็นระหว่าง S2 และ B1 โดยเรา S2 ไว้ทางขวามือเพราะความน่าจะเป็นน้อยกว่า



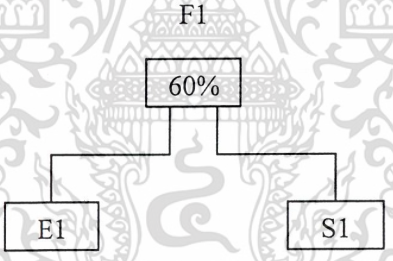
รูปที่ 2.17 การจับคู่ข้อมูลที่มีความน่าจะเป็นระหว่าง B1 และ S2

จากนั้นทำการจับคู่ตามลำดับความน่าจะเป็นระหว่าง C1 และ D1 โดยเอา C1 ไว้ทางขวามือเพราะความน่าจะเป็นน้อยกว่า



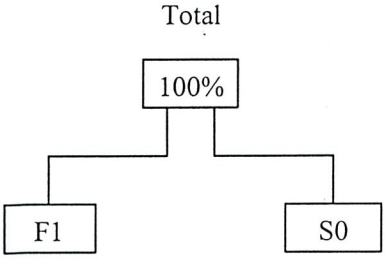
รูปที่ 2.18 การจับคู่ข้อมูลที่มีความน่าจะเป็นระหว่าง D1 และ C1

จากนั้นทำการจับคู่ตามลำดับความน่าจะเป็นระหว่าง S1 และ E1 โดยเอา S1 ไว้ทางขวามือเพราะความน่าจะเป็นน้อยกว่า



รูปที่ 2.19 การจับคู่ข้อมูลที่มีความน่าจะเป็นระหว่าง E1 และ S1

และลำดับสุดท้ายในการจับคู่คือ S0 และ F1 โดยเอา S0 ไว้ทางขวามือเพราะความน่าจะเป็นน้อยกว่า เป็นการจับคู่อันดับสุดท้ายเพราะทั้งคู่รวมความน่าจะเป็นได้ 100%



รูปที่ 2.20 การจับคู่ข้อมูลที่มีความน่าจะเป็นระหว่าง F1 และ S0

จากนั้นนำการจับคู่อันดับสุดท้ายมาขยายย่อยเพื่อทำการแทนรหัส

F1 คือ S1,E1

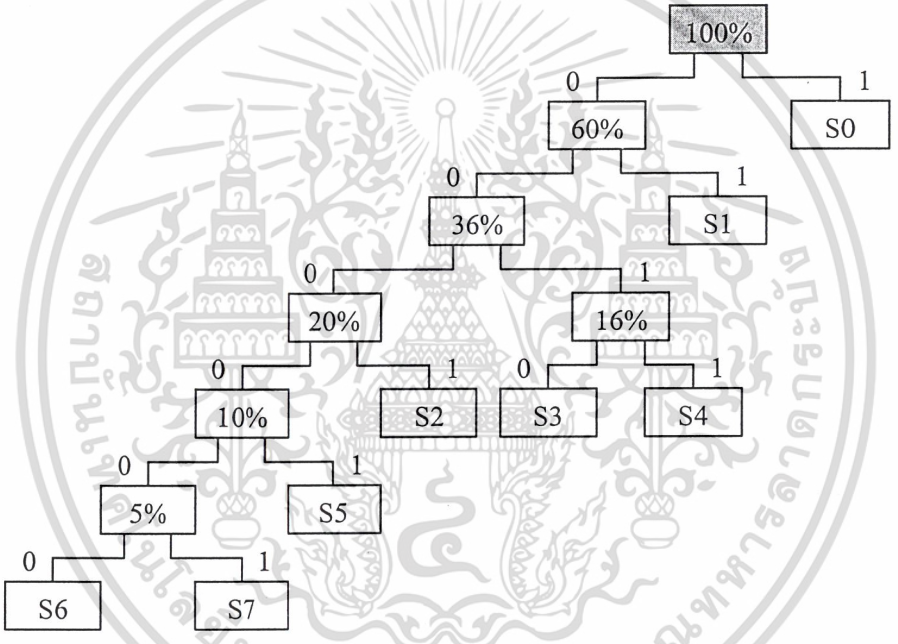
E1 คือ C1,D1

C1 คือ S4,S3

D1 คือ S2,B1

B1 คือ S5,A1

A1 คือ S7,S6



รูปที่ 2.21 ผลการนำการจับคู่ครั้งสุดท้ายมาขยายย่อย

3) การแทนรหัสฮัฟแมน

จากฮัฟแมนทรี (Huffman tree) สามารถแทนรหัสฮัฟแมนได้ดังนี้คือ

ตารางที่ 2.12 การแทนข้อมูลด้วยรหัสฮัฟแมน

สัญลักษณ์ (Symbols)	ความน่าจะเป็น (Probability)	รหัสฮัฟแมน (Huffman Codes)
S0	40%	1
S1	24%	01
S2	10%	0001
S3	9%	0010
S4	7%	0011
S5	5%	00001
S6	3%	000000
S7	2%	000001
Total	100%	

2.2.2 การถอดรหัสฮัฟแมน (Hufaman Decode)

สำหรับการถอดรหัสฮัฟแมนที่เก็บไว้ในหน่วยความจำ ซึ่งรหัสฮัฟแมนจะเรียงกันอยู่เป็นอนุกรม การถอดรหัสต้องดึงข้อมูลออกมาแล้วเช็คข้อมูลเป็นระดับบิตเพื่อนำไปชี้ว่ารหัสดังกล่าวเป็นรหัสของข้อมูลใด

ตัวอย่าง จงถอดรหัสฮัฟแมนต่อไปนี้ 000000110000110001

วิธีทำ

000000 เป็นรหัสของ S6

จะเหลือข้อมูล 110000110001

1 เป็นรหัสของ S0

จะเหลือข้อมูล 10000110001

1 เป็นรหัสของ S0

จะเหลือข้อมูล 0000110001

00001 เป็นรหัสของ S5

จะเหลือข้อมูล 10001

1 เป็นรหัสของ S0

จะเหลือข้อมูล 0001

0001 เป็นรหัสของ S2

ดังนั้น ข้อมูลที่อนุกรมกันมาคือ S6, S0, S0, S5, S0, S2

2.3 แผงวงจร DSK (DSP Starter Kit)

ใช้แผงวงจร DSK (DSP Starter Kit) TMS320C31 เป็นไมโครโปรเซสเซอร์ที่ต้องใช้กับชุดคำสั่งพิเศษและสถาปัตยกรรมเหมาะกับการประมวลสัญญาณเชิงเลข โดยมีคุณสมบัติดังนี้

2.3.1 คุณสมบัติแผงวงจร DSK

1) TMS320C31 เป็นตัวประมวลผลแบบแบบ Floating – Point จะประกอบไปด้วย AIC ย่อมาจาก Analog Interface Circuit Chip ซึ่งมีไว้สำหรับการทำโปรแกรม ADC (ตัวเปลี่ยนสัญญาณดิจิตอลเป็นสัญญาณแอนะล็อก) กับ DAC (ตัวเปลี่ยนสัญญาณแอนะล็อกเป็นสัญญาณดิจิตอล) และอินพุต และเอาต์พุต ฟิลเตอร์, สัญญาณทั้งหมดบนชิฟ

2) แผงวงจร DSK จะต่อพอร์ตขนาน (เครื่องพิมพ์) เพื่อติดต่อกับคอมพิวเตอร์

3)สามารถใช้งานร่วมกับออสซิลโลสโคป, เครื่องกำเนิดสัญญาณ, ลำโพง และ Signal/Spectrum Analyzer ได้

4) สามารถโปรแกรมด้วยภาษาแอสเซมบลี้และภาษาซีได้

5) ใช้กับแหล่งจ่ายไฟตรง 7.5 – 12 V

2.3.2 สถาปัตยกรรมและหน่วยความจำ TMS320C31

TMS320C31 มีหน่วยความจำของเวิร์ด (Words) เท่ากับ 2 K (32 Bit) จากหน่วยความจำภายในและมี 16 ล้านเวิร์ดสำหรับตำแหน่งหน่วยความจำภายนอก ซึ่งจะประกอบไปด้วย ส่วนของโปรแกรม, ส่วนของข้อมูลและ Input/Output

TMS320C31 เป็นสมาชิกในตระกูล C3x เป็น Floating – Point (จุดทศนิยม) มีขนาด 32 บิต โดยมีหน่วยความจำสามารถแยกได้ดังนี้ ส่วนโปรแกรม Boot – Loader (ส่วนของการเริ่มต้นโปรแกรม), หน่วยความจำภายใน 2K(RAM) ในสถาปัตยกรรมสามารถแบ่งได้ดังนี้

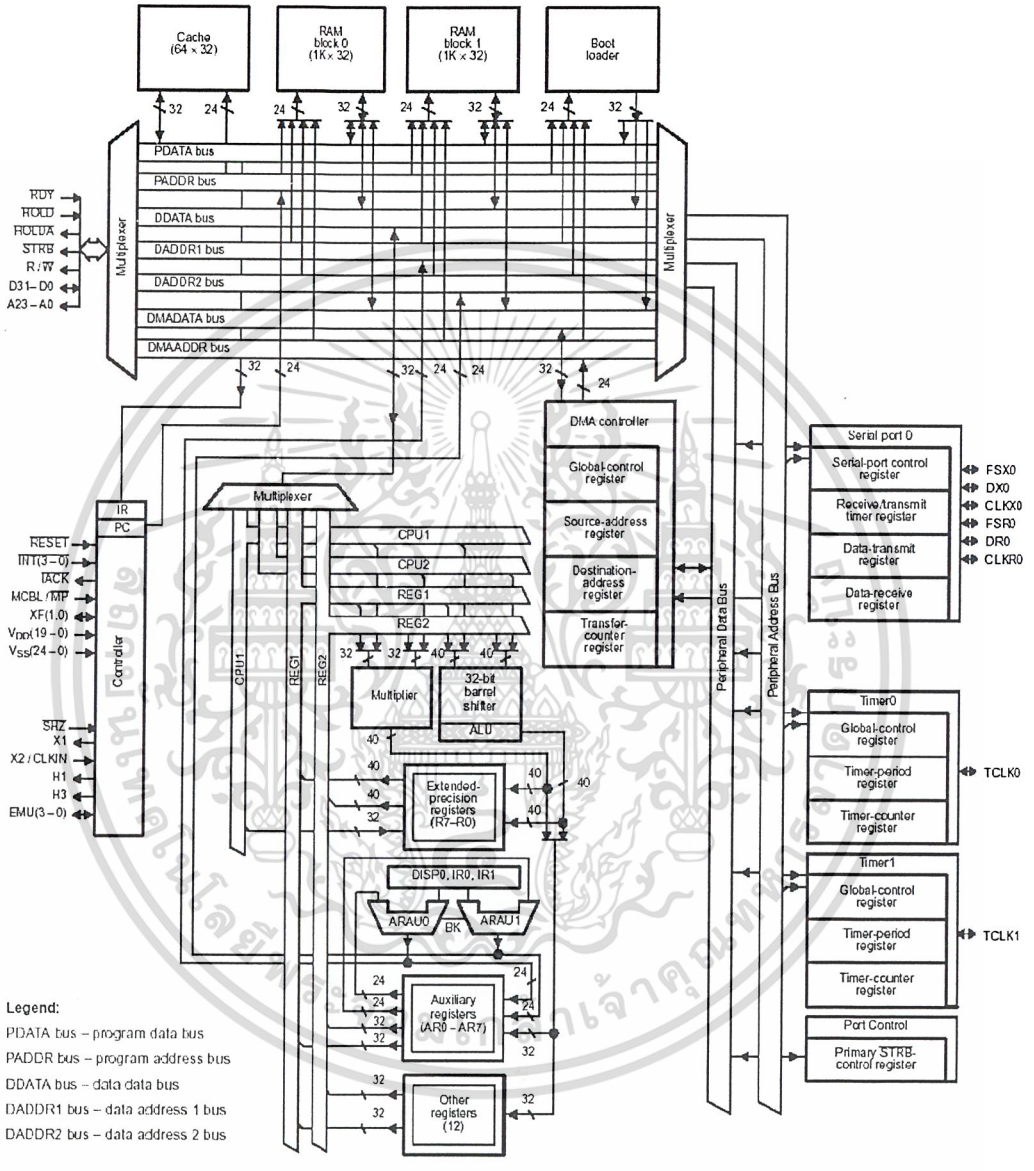
1) Arithmetic Logic Unit (ALU)

2) 32 – Bit Barrel Shifter

3) Internal Buses (CPU1/CPU2 and REG1/REG2)

4) Auxiliary Register Arithmetic Units (ARAUs)

5) CPU Register File



รูปที่ 2.22 สถาปัตยกรรม TMS320C31

จำนวนเต็ม (Floating – Point) มีขนาด 24 บิต และค่าทศนิยม (Floating – Point) มีขนาด 32 บิต สำหรับ Floating – Point และ Fixed – Point จะมีความเร็ว 33 ns ต่อคำสั่งหนึ่ง Cycle

การคูณ ALU จะคำนวณใน Cycle เดียว เมื่อไรมีการกระทำ Multiplier โดย Floating – Point Multiplication, Input มีขนาด 32 บิตแต่ผลลัพธ์มีขนาด 40 บิต

2.3.3 รีจิสเตอร์

1) R0 – R7 มีทั้งหมด 8 ตัว เป็นรีจิสเตอร์ขนาด 40 บิต สามารถเก็บผลลัพธ์ที่เป็นจำนวนเต็ม 32 บิต และสำหรับตัวเลขทศนิยมเก็บ 40 บิต

2) AR0 – AR7 เป็นรีจิสเตอร์ช่วย

3) IR0 และ IR1 ใช้สำหรับชี้ตำแหน่งหน่วยความจำ

4) ST ใช้บอกสถานะของ CPU

5) SP เป็น Stack Pointer

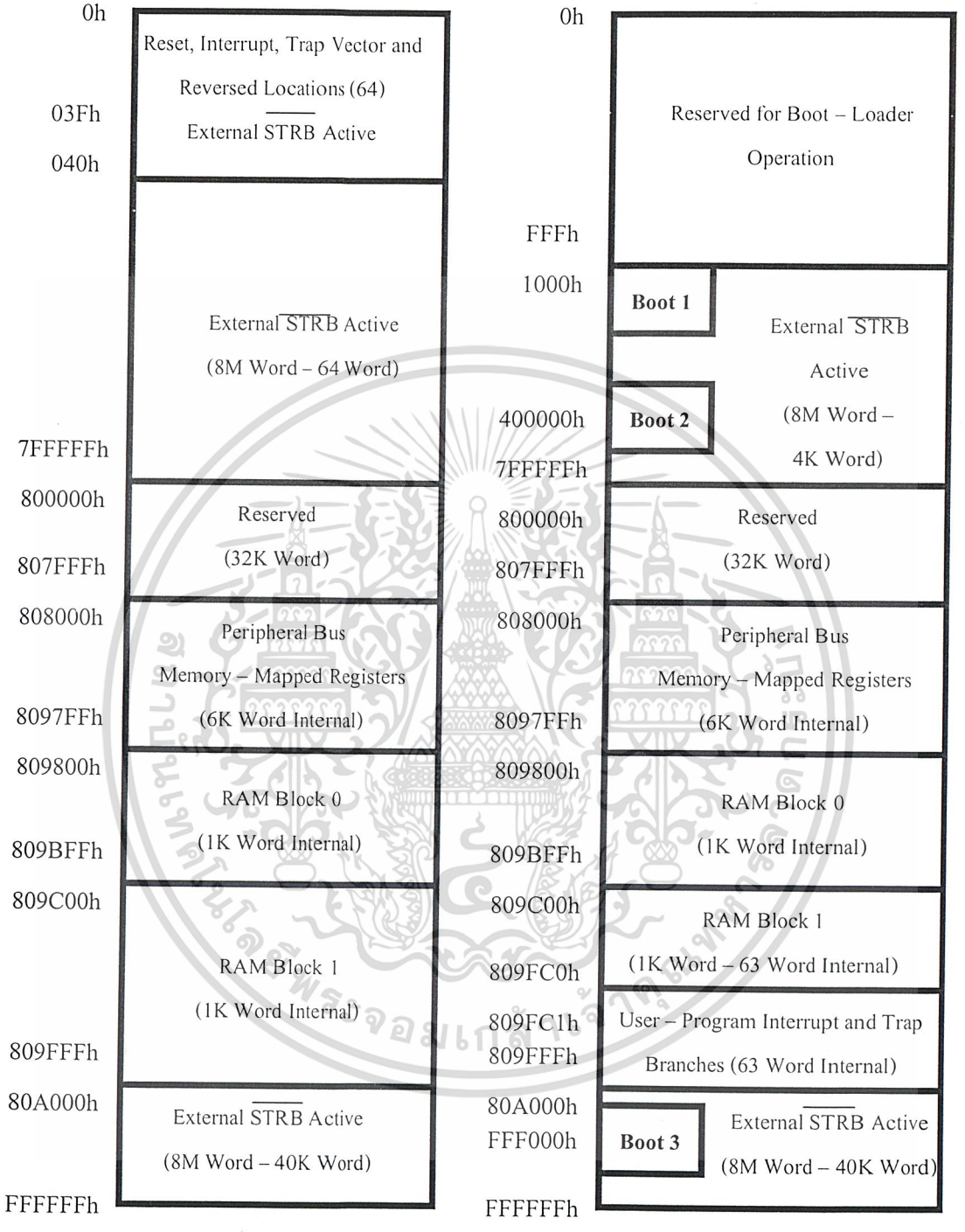
6) BK (Specify the Book)

7) IE, IF, IOF คือ อินเตอร์รัปท์ Enable, อินเตอร์รัปท์ Flag, I/O Flag (IOF)

8) RC (Repeat Count) ใช้เกี่ยวกับ Time

9) RS และ RE จะประกอบไปด้วยตำแหน่งเริ่มต้นกับตำแหน่งสุดท้าย

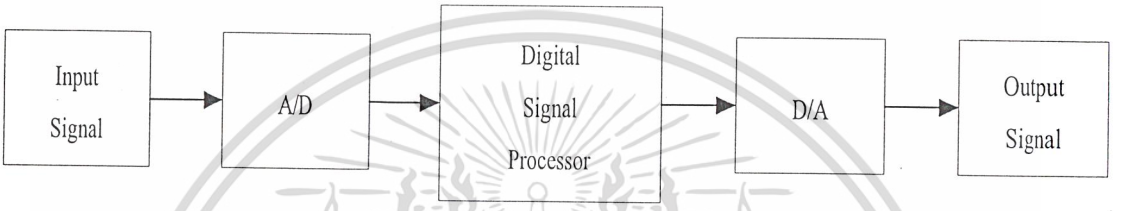
10) PC (Program Counter)



รูปที่ 2.23 ตำแหน่งหน่วยความจำ TMS320C31

2.3.4 การติดต่ออินพุตและเอาต์พุตของ DSK ขั้นพื้นฐาน

การติดต่ออินพุตและเอาต์พุตของ DSK ขั้นพื้นฐาน ดังแสดงในรูปที่ 2.24 ซึ่งประกอบไปด้วยแอนะล็อกอินพุตและแอนะล็อกเอาต์พุต ในส่วนอินพุตคือ Antialiasing Filter ซึ่งจะมี A/D คือวงจรแปลงแอนะล็อกเป็นดิจิตอลแล้วจึงจะนำสัญญาณที่ได้ไปประมวลผลโดยวงจรประมวลผลสัญญาณเชิงเลข ซึ่งระดับสูงสุดของสัญญาณอินพุตจะถูกจำกัดด้วยตัว A/D ดังแสดงในรูปที่ 2.24



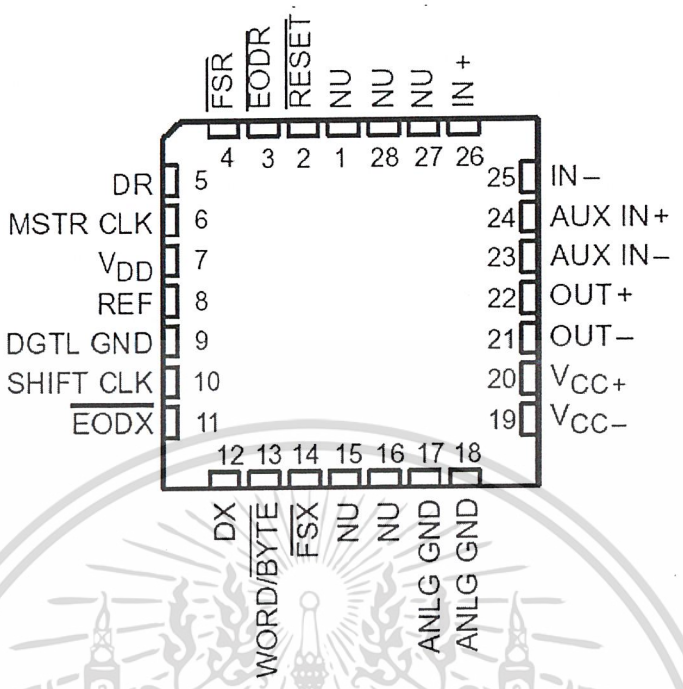
รูปที่ 2.24 อินพุตและเอาต์พุตของระบบ

หลังจากสัญญาณผ่านการประมวลแล้ว ผลลัพธ์ที่ต้องการจะผ่าน D/A คือวงจรแปลงดิจิตอลเป็นแอนะล็อก

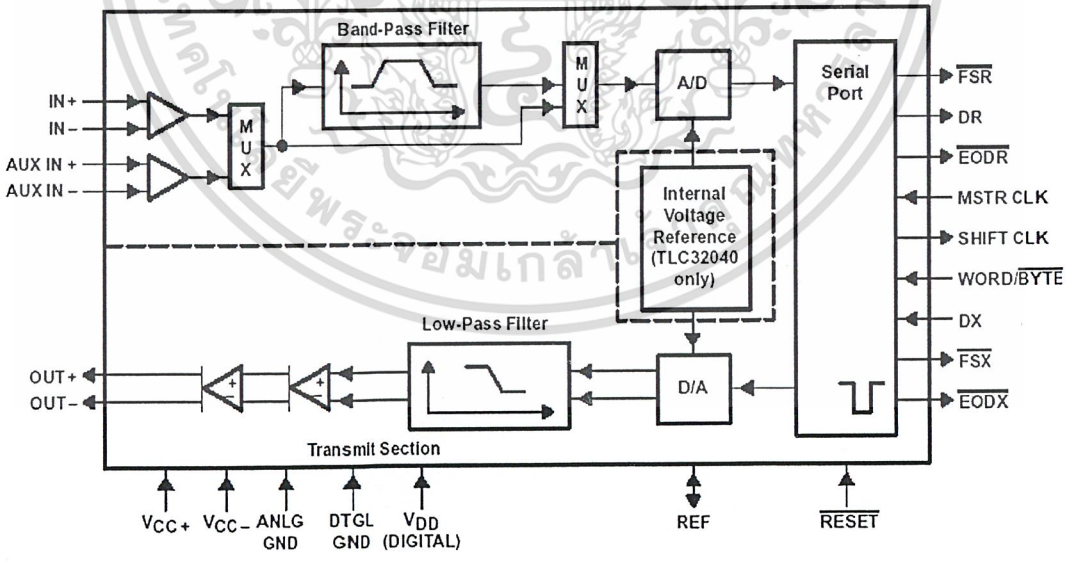
2.3.5 AIC (Analog Interface Circuit)

มีคุณสมบัติดังนี้

1) มีขนาด 14 บิต Dynamic Range ADC และ DAC
2) ADC และ DAC มีอัตราการสุ่ม 19,200 Samples ต่อวินาที
3) มี Switched – Capacitor Antialiasing เป็น Input Filter และ Output Filter
4) เป็นชุด Serial Port สำหรับต่อโดยตรง เพื่อนำไปประมวลผลสัญญาณเชิงเลข
5) มี Synchronous และ Asynchronous ADC และ DAC Convert
6) เป็นชุด Port ส่วนติดต่อ SN74299
7) เป็นเทคโนโลยี CMOS ซึ่งแสดงในรูป 2.25



รูปที่ 2.25 ตัวถังชิพ AIC (Analog Interface Circuit)



รูปที่ 2.26 การทำงานของ AIC ชิพ

ตารางที่ 2.13 ขาต่างๆ ของ AIC ชิพ

ชื่อ	หน้าที่
ANLG GND	อยู่ที่ขา 17, 18 เป็น Analog Ground อยู่ภายใน Analog Circuits จะไม่เชื่อมต่อกับ DGTL GND
AUX IN+	อยู่ที่ขา 24 เป็น Noninverting Auxiliary Analog Input สามารถเปลี่ยนค่าในการกรองความถี่แบบ Band Pass และ A/D
AUN IN-	อยู่ที่ขา 23 เป็น Inverting Auxiliary Analog Input (อยู่ข้างบน AUX IN+)
DGTL GND	อยู่ที่ขา 9 Digital Ground อยู่ ภายใน Logic Circuits จะไม่เชื่อมต่อ ANLG GND
DR	อยู่ที่ขา 5 เป็นเอาต์พุต DR ถูกใช้เพื่องส่งข้อมูลจาก ADC ไปยัง AIC
DX	อยู่ที่ขา 12 เป็นอินพุต DX ถูกใช้เพื่อรับข้อมูลจาก DAC Input และหน่วยควบคุมไปยัง AIC ทาง port
EODR	อยู่ที่ขา 3 เป็นเอาต์พุต เพื่อแสดงการจบของข้อมูลและเป็นชุด Port ของ Timing Diagrams ระหว่างโหมดคำ
EODR	ไบต์ต่ำหลังไบต์แรกเป็น Transmitted จาก AIC สู่ TMS320C31 เป็นชุด Port และ Kept เป็นลอจิก LOW กระทำไบต์ที่สองได้เป็น Transmitted TMS320C31 เพื่อบอกความแตกต่างระหว่างไบต์สอง ซึ่งก็คือวินาที
EODR	จะปรากฏ หลัง Secondary ของการสื่อสาร
EODX	อยู่ที่ขา 11 เป็นเอาต์พุต เป็นการแสดงการจบของข้อมูลที่ส่งไปยังชุด Port การ Timing Diagram ระหว่าง โหมดคำที่ Timer, EODX คือไบต์ต่ำ
SR4	เป็นเอาต์พุต เป็นเฟรม sync ที่ใช้รับข้อมูลในชุด Transmission โหมด ซึ่งถูกอ้างถึงในคำ FSR คือ Held เป็นลอจิก LOW ระหว่างบิต เมื่อเกิดการ Transmission เมื่อไร FSR เป็นไบต์ต่ำ TMS320C31 เมื่อนั้นเป็นชุด Port เริ่มต้นการรับบิตจาก AIC
FFSX	อยู่ที่ขา 14 เป็นเอาต์พุตเฟรม sync ส่งไป เมื่อไร FSX เป็นไบต์ต่ำ เมื่อนั้นชุด Port เริ่มต้นการส่งกลับไปสู่ AIC
DX	โหมดคำสั่งชุด Transmission ซึ่งถูกอ้างถึงในคำ FSX คือ Held ในบิตต่ำระหว่างบิต Transmission

ตารางที่ 2.13 (ต่อ) ขาต่างๆ ของชิพ AIC

IN+	อยู่ที่ขา 26 เป็นอินพุต Non Inverting Input เป็นตัวขยาย
IN-	อยู่ที่ขา 25 เป็นอินพุต Inverting Input เป็นตัวขยาย
MS TR CLK	อยู่ที่ขา 6 เป็นอินพุต Master Clock เมื่อถูกใช้ Key Logic Signals ที่ AIC Switched – Capacitor, A/D และ D/A ค่าเวลาที่ตั้งไว้ภายใน Diagram จะให้เอาต์พุตเป็นสัญญาณ โดยความถี่ของสัญญาณขึ้นอยู่กับการโปรแกรม
OUT+	อยู่ที่ขา 22 เป็นเอาต์พุต Non – Inverting ส่งออกสัญญาณเป็น Analog ส่งออก Power Amplifier
OUT-	อยู่ที่ขา 21 เป็นเอาต์พุต Inverting ส่งออกสัญญาณเป็น Analog ส่งออก Power Amplifier
REF	อยู่ที่ขา 8 เป็นแรงดันอ้างอิงสำหรับ TLC32040 สำหรับ TLC32040 และ TLC32041 แรงดันภายนอก
V+	อยู่ที่ขา 20 เป็น Positive Analog จ่ายแรงดัน 5 V ถึง +-5%
V-	อยู่ที่ขา 19 เป็น Negative Analog จ่ายแรงดัน 5 V ถึง +-5%

2.3.6 การติดต่อระหว่าง PC (Personal Computer) กับ TMS320C31

การติดต่อระหว่างคอมพิวเตอร์กับ TMS320C31 จะเริ่มต้นจากการ Load Kernel จากคอมพิวเตอร์ผ่านพอร์ตขนาน ก่อนที่ Kernel จะ Load TMS320C31 ต้องการการรีเซ็ตจากคอมพิวเตอร์ผ่านพอร์ตขนาน โดยในการ Boot (การเริ่มต้น) จะโหลดจากตำแหน่งเริ่มต้น 0xFFF000

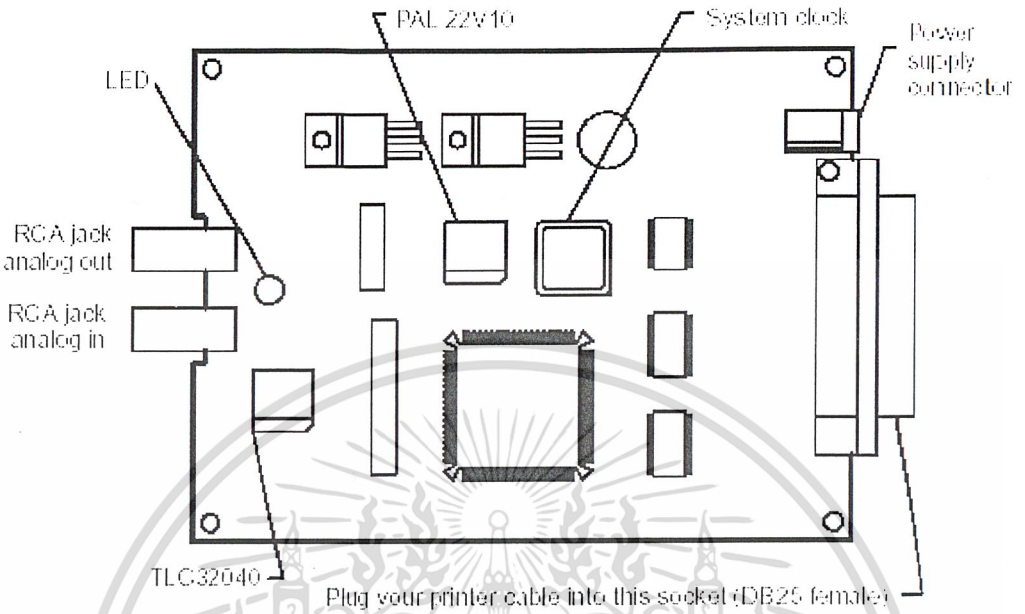
การเชื่อมต่อระหว่างเครื่องคอมพิวเตอร์กับแผงวงจร DSK ต้องใช้การเชื่อมต่อพอร์ตขนาน (เครื่องพิมพ์)

ขั้นตอนที่ 1

1) ปิดสวิตซ์เครื่องคอมพิวเตอร์

2) เชื่อมต่อ Parallel Port (เครื่องคอมพิวเตอร์) สู่ Parallel Communication Port (แผงวงจร DSK)

3) ต่อแหล่งจ่ายไฟตรง 7 – 12 Vdc หรือ 6 – 9 Vac



รูปที่ 2.27 การเชื่อมต่อ Parallel Port เข้ากับแผงวงจร DSK

4) เสียบแหล่งจ่ายเข้าที่รูเสียบ

5) เปิดเครื่องคอมพิวเตอร์

6) LED เกิดแสงสว่างทั้งสีแดงและสีเขียว

ขั้นตอนที่ 2 : การติดตั้ง DSK ซอฟแวร์

1) สร้างไดเร็กทอรี่ ชื่อ Dsktools เอกสารจะบรรจุอยู่ใน DSK Assembler และซอฟแวร์ Debugger

2) ทำการ Copy โปรแกรม

ขั้นตอนที่ 3 : แก้ไขแฟ้ม Config.sys

เมื่อไรที่มีการใช้ Debugger ต้องเกิดการแก้ไข Config.sys และเพิ่ม File

ขั้นตอนที่ 4 : การแก้ไข PATH Statement

ทำเพื่อให้แน่ใจ Debugger และ Assembler เป็นการอ้างจาก Directory ในคอมพิวเตอร์ แก้ไข PATH statement เพื่อค้นหาไดเร็กทอรี่ของ Dsktools เช่น

PATH=C:\dsktools; pathname2; pathname3

การติดตั้ง DSK Environment บน Dos Command (ตัวอย่าง autoexec.bat File)

PATH statement →

```
DATE
TIME
ECHO OFF
PATH=c:\dos;c:\dsktools
CLS
```

รูปที่ 2.28 การติดตั้ง DSK Environment บน Dos Command

ถ้าแก้ไข autoexec.bat

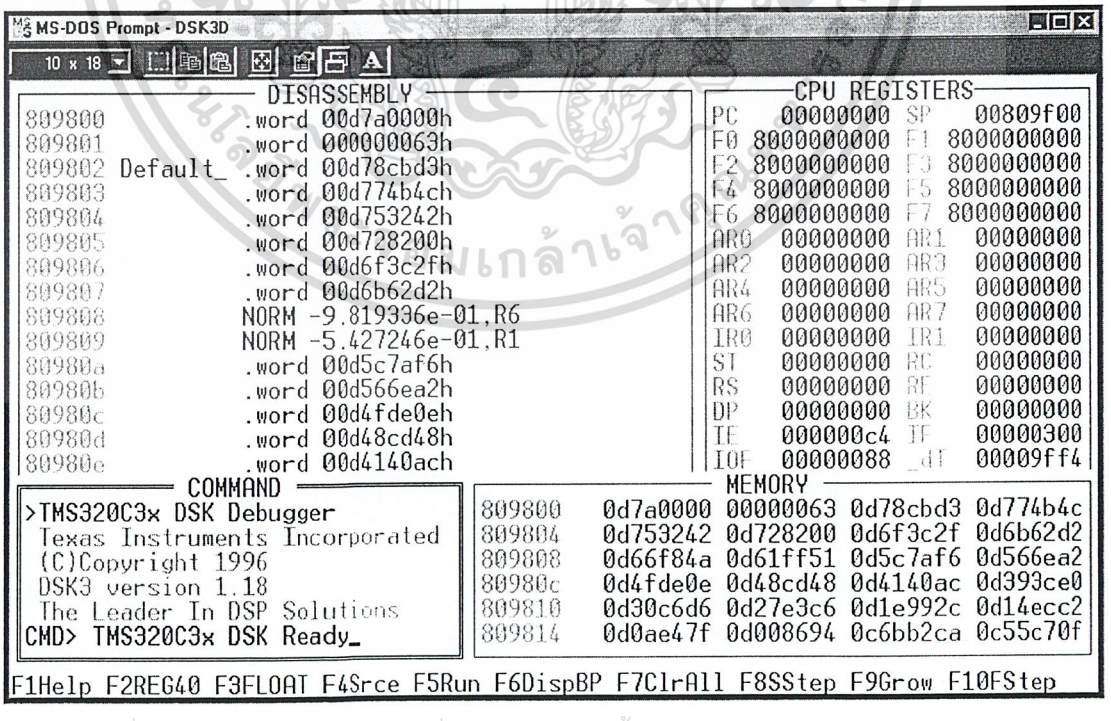
ขั้นตอนที่ 5 : ทดสอบการติดตั้ง

เพื่อให้แน่ใจในการติดตั้ง DSK บอร์ด, Assembler และ Debugger ใส่คำสั่งต่อไปนี้ที่ระบบพร้อมเริ่มต้น DSK

debugger:

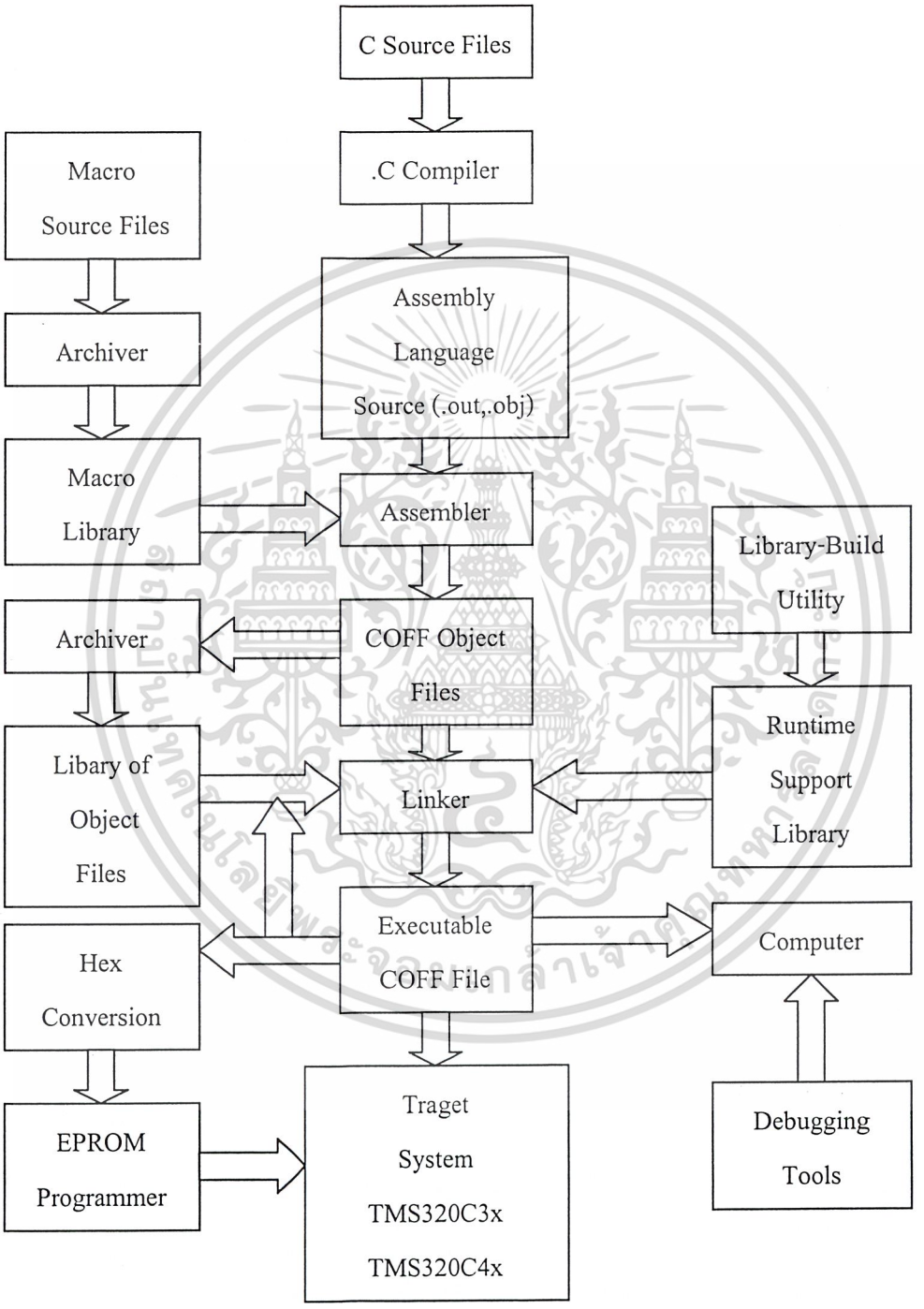
dsk3d

หลังจากใส่คำสั่ง dsk3d จะเห็นหน้าจอดังรูปที่ 2.29



รูปที่ 2.29 พื้นฐานหน้าจอ Debugger

2.3.7 เครื่องมือสำหรับการพัฒนา



รูปที่ 2.30 ส่วนประกอบ TMS320C3x/C4x เกี่ยวกับเครื่องมือสำหรับการพัฒนา

ส่วนประกอบ TMS320C3x/C4x เกี่ยวกับเครื่องมือสำหรับการพัฒนา ที่แสดงในรูปที่ 2.30

1) การคอมไพล์ .C เพื่อเป็นรหัสข้อมูลซึ่งจะใช้กับ TMS320C3x หรือ TMS320C4x หรือภาษาแอสเซมบลี (Assembly) ซึ่งชุด C Compiler จะประกอบด้วยโปรแกรม Shell (cl30), Optimizer (opt30) และ Interlist (clist)

2) โปรแกรม Shell เป็นโปรแกรมที่ช่วยในการตรวจสอบอัตโนมัติโดย Assemble และ Link Source Modules

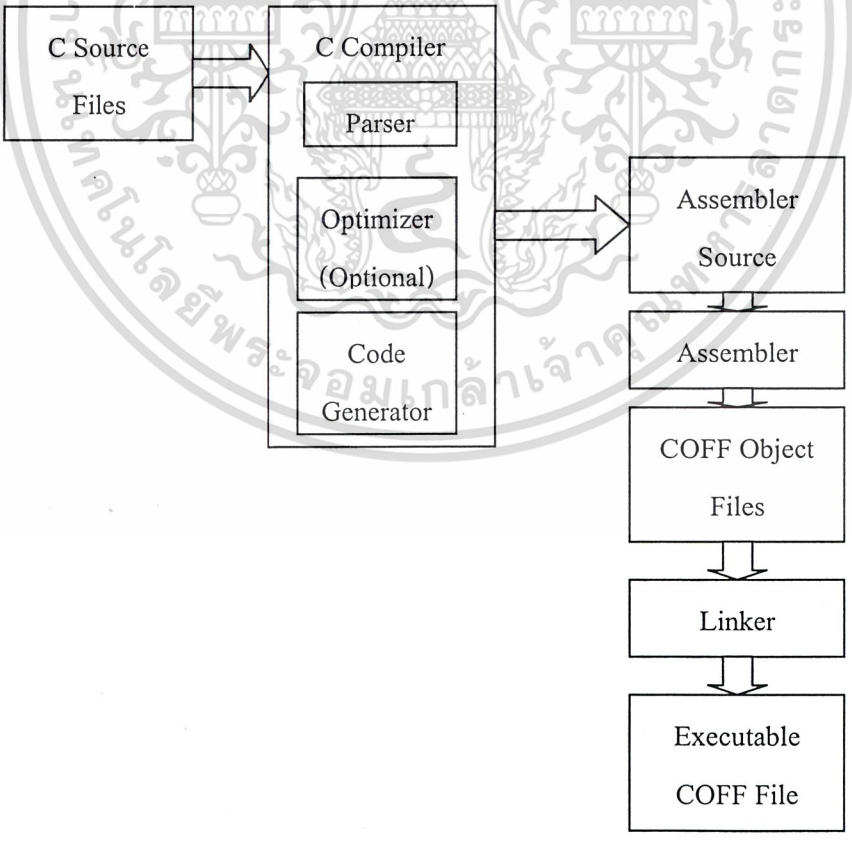
3) Optimizer ทำการแก้ไขปรับปรุงโปรแกรมภาษา C

4) เครื่องมือ Interlist ในภาษาซี จะแปลงข้อมูลประโยคไปเป็นภาษา Assembly

5) Assembler จะทำการแปล Assembly เป็นข้อมูลเข้าไปในเครื่อง

2.3.8 การคอมไพล์ภาษาซี

cl30 ในโปรแกรม Shell คือเครื่องมือใช้ Compile, Assemble และข้อกำหนดเพิ่มเติมการเชื่อมต่อ Compiler ประกอบด้วย Parser, Optimizer และ Generator Code, Assembler จะได้แฟ้ม COFF



รูปที่ 2.31 การใช้ C Compiler

Linker สามารถ Invoked (อ้าง) เป็นส่วนของกระบวนการ Larger หรือสามารถ Compiler และ Assembler Invoking ของ C Compiler เพื่อใช้ในการคอมไพล์ (Compiler)

Cl30 [- options] [filenames] [- z [link_options] [opject files]]

C130 คือ คำสั่งในการ Invokes ของ Compiler

Options คือ ข้อกำหนดที่มีผลต่อวิธี Compiler ของ กระบวนการ Input Files

- z คือ ข้อกำหนดใช้ Linker เป็น หน่วยควบคุมข้อกำหนดทางเชื่อมต่อ เช่น

> C130 symtab

เมื่อ Enter แล้วจะเกิดการ Compiler ดังข้อความด้านล่าง

```
[symtab]
TMS320C3x/4x ANSI C Compiler Version x.xx
Copyright (c) 1987 – 1997, Texas Instruments Incorporated
”symtab.c”:= => main
”symtab.c”:= => lookup
TMS320C3x/4x ANSI C Codegen Version x.xx
Copyright (c) 1987 – 1997, Texas Instruments Incorporated
”symtab.c”:= => main
”symtab.c”:= => lookup
TMS320C3x/4x COFF Assembler Version x.xx
Copyright (c) 1987 – 1997, Texas Instruments Incorporated
PASS 1
PASS 2
No Errors, No Warnings
```

รูปที่ 2.32 ผลที่เกิดจากการใช้ C Compiler

2.4 การส่งข้อมูลผ่านพอร์ตขนาน (Interfacing the Standard Parallel Port)

พอร์ตขนานในเครื่องคอมพิวเตอร์โดยทั่วไปประกอบด้วยขาสัญญาณอินพุต 9 บิต และเอาต์พุต 12 บิตซึ่งเป็นสายสัญญาณควบคุม (Control Line) 4 เส้น สายสัญญาณสถานะ (Status Line) 5 เส้น และสายสัญญาณข้อมูล (Data Line) 8 เส้น ต่อเชื่อมกับคอมพิวเตอร์ด้วยฟีเมลคอนเนคเตอร์ (Female Connector) ขนาด 25 ขา

โหมดการใช้งานการส่งข้อมมูลติดต่อกับคอมพิวเตอร์ทางพอร์ตขนานตามมาตราฐาน 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)

ในการ Centronics Mode, Nibble Mode และ Byte Mode อาจจัดรวมได้เป็นกลุ่มมาตราฐานพอร์ตขนาน (SPP) ส่วนโหมด EPP และ ECP จำเป็นต้องใช้ร่วมกับฮาร์ดแวร์เพิ่มเติมที่มีความเร็วสูง

การใช้งานในโครงการนี้เลือกใช้ Standard Parallel Port จึงขอกล่าวเน้นเฉพาะเนื้อหาในส่วนที่เกี่ยวข้องนี้เท่านั้น

โดยปกติ Centronics Mode จะส่งข้อมูลด้วยความเร็ว 50 กิโลไบท์ต่อวินาที และอาจสามารถปรับให้มีความเร็วเพิ่มขึ้นไปถึง 150 กิโลไบท์ต่อวินาทีได้ ในการรับข้อมูลจะเปลี่ยนโหมดการทำงานเข้าสู่ นิบเบิล (Nibble Mode) หรือ ไบท์โหมด (Byte Mode) โดยนิบเบิลโหมดจะรับข้อมูลขนาด 4 บิต จากอุปกรณ์ภายนอกเข้ามาเพียงทิศทางเดียว และไบท์โหมดจะทำงานใน 2 ทิศทางกับข้อมูล 8 บิต

คอนเนคเตอร์ที่ใช้เชื่อมต่อโดยทั่วไปเป็น D – Type 25 Pin Connector ซึ่งใช้กับพอร์ตขนานโดยทั่วไป หรือ Centronics 34 Pin Connector ซึ่งใช้ต่อกับเครื่องพิมพ์ (Printer) ประกอบด้วยขาสัญญาณดังต่อไปนี้

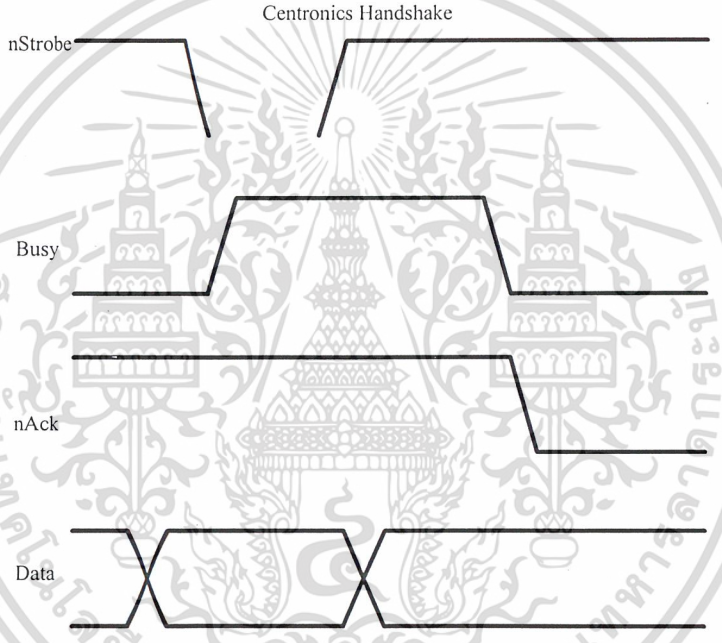
ตารางที่ 2.14 ขาสัญญาณและการใช้งานในพอร์ตส่งข้อมูลแบบขนาน

หมายเลขขาสัญญาณ (D-Type 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/ Paper End	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)

2.4.1 การควบคุมการส่ง – รับ ข้อมูลด้วยซอฟแวร์ (Centronics Handshake)

ข้อมูลที่ส่งเข้ามาจะอยู่ที่ขาสัญญาณที่ 2 – 9 ดังรูปที่ 2.33 และทำการตรวจสอบเครื่องพิมพ์ว่างและอยู่ในสภาวะที่พร้อมจะทำงานหรือไม่ ถ้าเครื่องพิมพ์ว่างสัญญาณ Busy จะมีสถานะต่ำ โปรแกรมจะส่งสัญญาณ Strobe สถานะต่ำออกไปนาน 1 μS เครื่องพิมพ์จะอ่านข้อมูลเข้าไปแสดงสภาวะการอ่านข้อมูลนี้ด้วยขาสัญญาณ Busy ที่มีสถานะสูง เมื่อเครื่องพิมพ์รับข้อมูลทั้งหมดเรียบร้อยแล้วจะส่งสัญญาณ Ack สถานะต่ำออกมานาน 5 μS



รูปที่ 2.33 สัญญาณควบคุมการส่งข้อมูล

ในส่วนการเขียนโปรแกรมจริง จึงเพียงแต่สั่งเขียนข้อมูลออกไปยังเครื่องพิมพ์เท่านั้นและฮาร์ดแวร์จะทำการตรวจสอบสภาวะการทำงานต่างๆ เอง ยกเว้นแต่สัญญาณ Ack ที่จะไม่ได้รับการตรวจสอบ

2.4.2 พอร์ตแอดเดรส (Port Address)

พอร์ตขนานถูกใช้งานโดยกำหนดแอดเดรสไว้ 3 แบบ ดังตารางที่ 2.15

ตารางที่ 2.15 แอดเดรสพอร์ตของพอร์ตขนาน

แอดเดรสพอร์ต	คำอธิบาย
3BCh – 3BFh	ใช้ติดต่อกับวีดีโอการ์ด (Video Card) ซึ่งถูกควบคุมด้วยโปรแกรม BIOS และไม่สามารถใช้งานในโหมด ECP ได้
378h – 37BFh	แอดเดรสที่ใช้งานใน LPT1
278h – 27BFh	แอดเดรสที่ใช้งานใน LPT2

2.4.3 โปรแกรมควบคุมรีจิสเตอร์พอร์ตขนาน (Software Register – Standard Parallel Port : SPP)

1) พอร์ตส่งข้อมูล (Data Port)

พอร์ตส่งข้อมูล (Data Port) หรือเรียกอีกอย่างหนึ่งว่ารีจิสเตอร์พอร์ต (Register Port) ซึ่งใช้ในการส่งข้อมูลออกจาก PC สู่อุปกรณ์ภายนอก ถ้าพอร์ตขนานเป็นแบบ 2 ทาง (Bi – Directional) ก็จะสามารถรับข้อมูลเข้าได้ด้วยพอร์ตนี้ ขา 2 – 9 ดังตารางที่ 2.16

ตารางที่ 2.16 พอร์ตส่งข้อมูล (Data Port)

ออฟเซ็ท (Offset)	ชื่อ	การทำงาน	หมายเลขบิต	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)

2) พอร์ตสถานะ (Status Port)

พอร์ตสถานะเป็นพอร์ตทิศทางเดียว ทำการอ่านข้อมูลเพียงอย่างเดียวเท่านั้น การเขียนข้อมูลด้วยพอร์ตนี้ไม่สามารถใช้งานได้ พอร์ตสถานะประกอบด้วย 5 สัญญาณ ดังตารางที่ 2.17

ตารางที่ 2.17 พอร์ตสถานะ (Status Port)

ออฟเซ็ท (Offset)	ชื่อ	การทำงาน	หมายเลขบิต	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

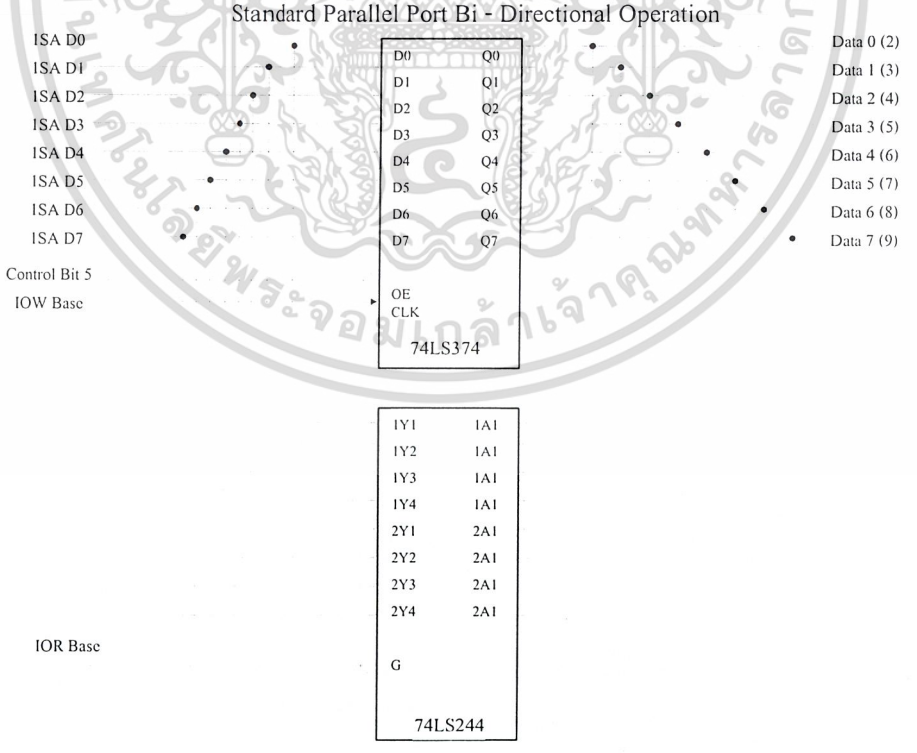
3) พอร์ตควบคุม (Control Port)

พอร์ตควบคุมเป็นพอร์ตทิศทางเดียว ทำการเขียนข้อมูลเพียงอย่างเดียวเท่านั้น พอร์ตควบคุมประกอบด้วย 8 ขาสัญญาณ ดังตารางที่ 2.18

ตารางที่ 2.18 พอร์ตควบคุม (Control Port)

ออฟเซ็ท (Offset)	ชื่อ	การทำงาน	หมายเลขบิต	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 Vai Ack Line
			Bit 3	Select Printer
			Bit 2	Initialized Printer (Reset)
			Bit 1	Auto Line Feed
			Bit 0	Strobe

2.4.4 การต่อรีจิสเตอร์พอร์ตขนาน (Bi – Directional Ports)



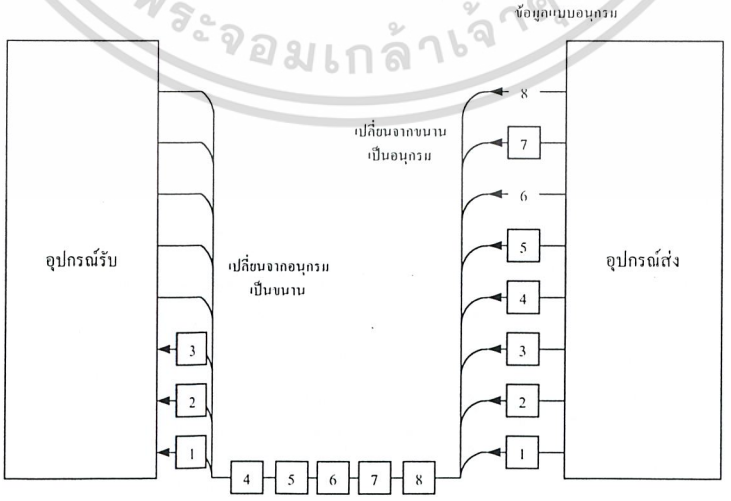
รูปที่ 2.34 การต่อรีจิสเตอร์พอร์ตขนาน

จากรูปที่ 2.34 แสดงการต่อรีจิสเตอร์ข้อมูลซึ่งทำหน้าที่เป็นบัฟเฟอร์ของพอร์ตขนาน กรณีพอร์ตขนานส่งข้อมูลทิศทางเดียวจะสร้างจากไอซี 74LS374 ที่ขาสัญญาณ Output Enable จะมีสถานะต่ำตลอดเวลา ทำให้พอร์ตข้อมูลส่งข้อมูลออกเพียงด้านเดียว การใช้งานให้เป็นพอร์ต 2 ทิศทาง (Bi – directional Port) จะจ่ายกระแสเกิน (Overdrive) ให้กับขา OE นี้ ขา OE ตรงกับพอร์ตควบคุมบิต 5 ดังนั้น ในทางปฏิบัติจึงควบคุมให้ทำงานส่งข้อมูล 2 ทิศทางผ่านบิตนี้ โดยการเขียน 1 ไปยังบิตควบคุม จะทำให้ขาสัญญาณ 2 – 9 มีสถานะความต้านทานสูง (Hi – Impedance) พอร์ตนี้จะใช้รับข้อมูลเข้ามายังพีซีได้ และยกเลิกการส่งข้อมูล 2 ทิศทางโดยการเขียนค่า 0 มาที่บิต 5 ของพอร์ตควบคุม

ในกรณีที่ฮาร์ดแวร์ไม่สามารถใช้งานเป็นพอร์ตส่งข้อมูลแบบ 2 ทิศทางได้ อาจส่งข้อมูลอินพุตเข้าไปยังพีซีได้โดยผ่านทางพอร์ตควบคุมและพอร์ตสถานะ ซึ่งทำงานในทิศทางอินพุตได้อยู่แล้วแทน

2.5 การถ่ายโอนข้อมูลแบบอนุกรม

ในการถ่ายโอนข้อมูลแบบนุกรม ข้อมูลถูกส่งออกมาทีละบิต ระหว่างจุดส่งและจุดรับจะเห็นว่าการส่งข้อมูลแบบนี้จะช้ากว่าแบบขนานที่กล่าวมาแล้วแน่นอน แล้วทำไมจะต้องมีการส่งแบบนี้ละ คำตอบก็คือตัวกลางการสื่อสารต้องการเพียงช่องเดียวหรือสายเพียงคู่เดียว ค่าใช้จ่ายในการสื่อสารจะต้องถูกว่าแบบขนานอย่างแน่นอน สำหรับการส่งระยะทางไกลๆ โดยเฉพาะเมื่อเรามีระบบสื่อสารทางโทรศัพท์ไว้ใช้งานอยู่แล้ว ย่อมจะเป็นการประหยัดกว่าที่จะทำการติดต่อสื่อสารทีละ 8 ช่อง เพื่อการถ่ายโอนข้อมูล แบบขนานอย่างแน่นอน



รูปที่2.35 การส่งข้อมูลแบบอนุกรม

จากรูปที่ 2.35 แสดงให้เห็นการส่งข้อมูลแบบอนุกรมจากจุดส่งจะถูกเปลี่ยนให้เป็นอนุกรมเสียก่อน แล้วค่อยทยอยส่งออกทีละบิตไปยังจุดรับ ณ.ที่จุดรับจะต้องมีกลไกในการเปลี่ยนข้อมูลที่ส่งมาทีละบิตให้เป็นสัญญาณแบบขนานซึ่งลงตัวพอดีนั่นคือ บิต 1 ลงที่บัสข้อมูลเส้นที่ 1 พอดี การที่จะทำการแปลงสัญญาณอนุกรมทีละบิตให้ลงพอดีนั้นจำเป็นต้องมีกลไกที่เหมาะสม เพื่อป้องกันข้อผิดพลาดในการรับ กลไกที่ว่านี้แบ่งเป็น 2 แบบ คือ

1) การสื่อสารแบบซิงโครนัส

2) การสื่อสารแบบอะซิงโครนัส

2.5.1 รูปแบบของการสื่อสารแบบอนุกรม

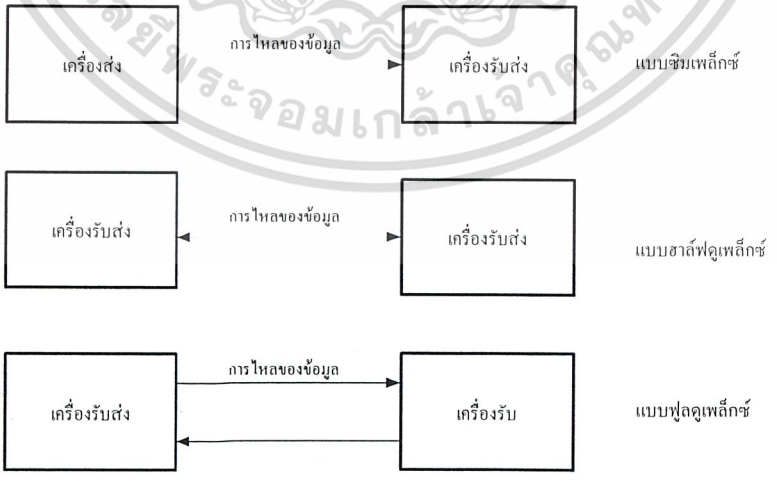
การติดต่อแบบอนุกรมอาจจะแบ่งตามรูปลักษณะได้ 3 แบบ ตามรูปที่ 2.36

1) แบบซิมเพล็กซ์ (Simplex) ข้อมูลส่งได้ทางเดียวเท่านั้น บางครั้งเรียกว่า การส่งทิศทางเดียว (Undirection Data Bus)

2) แบบฮาล์ฟดูเพล็กซ์ (Half Duplex) ข้อมูลสามารถส่งได้ทั้งสองสถานี แต่จะต้องผลัดกันส่งผลัดกันรับ จะส่งและรับพร้อมกันไม่ได้

3) แบบฟูลดูเพล็กซ์ (Full Duplex) ทั้งสองสถานีสามารถรับและส่งได้ในเวลาเดียวกัน

การส่งแบบฟูลดูเพล็กซ์และฮาล์ฟดูเพล็กซ์ ไม่ขึ้นอยู่กับจำนวนของสายการติดต่อบางครั้งคำว่า ทูไวร์ (Two Wire) หรือสองเส้น และ โฟร์ไวร์ (Four Wire) หรือ 4 เส้น ใช้ในการบรรยายถึงลักษณะการสื่อสารข้อมูลซึ่งอาจจะทำให้เข้าใจ และฮาล์ฟดูเพล็กซ์ สายโทรศัพท์ทั่วไปเป็นแบบ 2 เส้น ส่วนในสายที่เป็นแบบเช่า (Lease Line) นั้นส่วนมากจะเป็น 4 เส้น



รูปที่ 2.36 รูปแบบของการติดต่อสื่อสารข้อมูลแบบอนุกรม

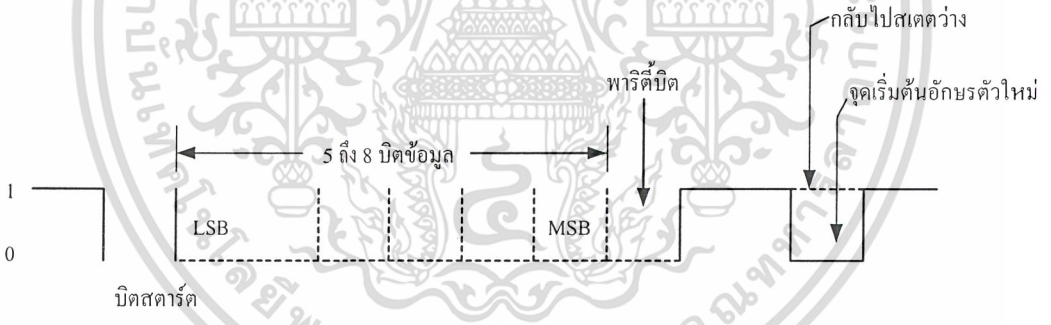
2.5.2 ความเร็วในการถ่ายโอนข้อมูลแบบอนุกรม

ความเร็วในการถ่ายโอนข้อมูลแบบอนุกรม หน่วยวัดเป็นบิตต่อวินาที (bps) เป็นหน่วยที่บรรยายถึงการเปลี่ยนแปลงของสัญญาณใน 1 วินาที เรียกว่าบอดเรต (baud rate) หรืออัตราบอด หลายคนยังเข้าใจสับสนระหว่าอัตราบอดและอัตราบิต (bit rate) การเปลี่ยนแปลงของสัญญาณ 1 ครั้ง อาจจะแสดงถึงการส่งข้อมูลแบบอนุกรมมากว่า 1 บิต ถ้าเขียนในรูปของสมการคณิตศาสตร์เราก็จะได้

$$\textit{อัตราบิต (bit rate)} = \textit{อัตราบอด (baud rate)} \times \textit{(บิตใน 1 บอด)} \qquad (2.6)$$

2.5.3 การสื่อสารแบบอะซิงโครนัส

การส่งแบบอะซิงโครนัสนี้ พัฒนามาจากการส่งโทรพิมพ์ในสมัยก่อน ลักษณะของสัญญาณแสดงไว้ในรูปที่ 2.37 เพื่อกลไกในการรับส่งอย่างถูกต้อง สัญญาณอะซิงโครนัสจะประกอบด้วยบิตเริ่มต้นหรือบิตสตาร์ท (Start Bit) และบิตสิ้นสุดหรือบิตสต๊อป (Stop Bit)



รูปที่ 2.37 ฟอร์แมตการสื่อสารแบบอะซิงโครนัส

ขณะที่สถานะของการส่งแบบว่า (Idle) คือยังไม่มีสัญญาณส่งออกมาจะมีแรงดันหรือกระแส (สัญญาณ) ตลอดเวลา เพื่อความแน่ใจว่าฝ่ายรับยังติดต่ออยู่กับฝ่ายส่ง เมื่อเริ่มจะส่งข้อมูลสัญญาณอะซิงโครนัสจะเป็น “0” หนึ่งช่วงของสัญญาณนาฬิกาบิตนี้เรียกว่าสตาร์ทบิต ตามหลังของสาตร์ทบิตก็จะเป็นข้อมูลสำหรับ 1 ตัวอักษร ซึ่งอาจจะมีขนาดตั้งแต่ 5 บิตจนถึง 8 บิต โดยบิตที่มีค่าน้อยที่สุด (LSB) จะถูกส่งออกมาก่อนไล่ไปจนถึงบิตที่มีค่ามากที่สุด (MSB) การเข้ารหัสอักขระนี้ ส่วนมากจะนิยมใช้รหัส ASCII แรกเริ่มทีเดียวในงานของโทรพิมพ์ เขาใช้รหัส (Baudit) ซึ่งใช้ 5 บิตในการส่งแทนตัวอักขระ 1ตัว ตามหลังข้อมูลก็จะเป็นพาริตี้บิตอาจจะเป็นแบบคู่ (Even)

หรือแบบคี่ (Odd) หมายความว่า ถ้าหากเป็นพาริตี้คู่ จำนวนบิตที่เป็น 1 ช่วงของบิตข้อมูลกับบิตพาริตี้รวมแล้วจะต้องเป็นจำวนคู่ ผู้ส่งจะต้องทำหน้าที่ตรวจสอบข้อมูลแล้วใส่พาริตี้เอง ฝ่ายรับเมื่อได้รับแล้วก็ต้องตรวจสอบดูว่าเป็นจริงตามที่สถานการณ์ที่ตั้งเอาไว้หรือไม่ หากมีความผิดพลาดเกิดขึ้นก็หมายความว่า สัญญาณที่รับนั้นผิดพลาดไปจากสถานีส่งส่งออกมา ทั้งนี้ทั้งนั้นจะต้องเป็นจำนวนคี่เท่านั้น คือผิดไป 1 บิต 3 บิต หรือ 5 บิต พร้อมกันจึงจะตรวจสอบได้ว่าผิด มองเห็นง่ายๆ ว่าถ้าผิดเป็นจำนวนคู่ ผลรวมของจำนวนหนึ่งก็ยังเป็นคู่อยู่ดี ทั้งนี้ทั้งนั้นไม่ได้หมายความว่าพาริตี้คู่ (Even Parity) แต่แทนที่จะตรวจสอบดูว่าสัญญาณที่รับเข้ามามีจำนวนคู่ ก็ตรวจสอบดูว่ามีจำนวนคี่หรือเปล่าอย่างไรก็ตามโอกาสที่จะผิดพลาด 2 บิต พร้อมกันมีน้อยมาก

ย้อนกลับมาดูสัญญาณอะซิงโครนัสใหม่ หลังจากบิตพาริตี้แล้ว ก็จะมีสต๊อปบิตหนึ่งความกว้างของสต๊อฟบิต อาจจะเป็น 1,1.5, หรือ 2 พัลส์ ของสัญญาณนาฬิกา แล้วแต่ผู้รับส่งจะตกลงกันเอง การเริ่มใช้พอร์ตอนุกรม (ทางออกอนุกรม) จึงจำเป็นจะต้องตั้งค่าต่างๆ สำหรับเป็นการส่งแบบอนุกรมอันได้แก่

1) ความเร็วในการส่ง

2) ความยาวรหัส 1 อักขระ

3) บิตตรวจสอบ

4) จำนวนบิตสต๊อป

ในการส่งโทรพิมพ์หรือโทรเลขเมื่อก่อนนี้ใช้ความเร็วแค่ 70 บอด และ 110 บอด สำหรับคอมพิวเตอร์ความเร็วในการส่งมีให้เลือกตั้งแต่ 110,200,300,1200,2400,4800,9600 บอด และสูงไปกว่านั้น เนื่องจากมี IC หลายเบอร์ทำหน้ามี่รับส่งแบบอะซิงโครนัสให้ใช้การส่งแบบอนุกรม จึงสะดวกสบายสำหรับคนออกแบบพอร์ตอนุกรม

จะเห็นว่ากลไกลในการซิงโครนัสของการสื่อสารอะซิงโครนัส มีลักษณะเป็นไปทีละตัวอักขระจำนวนพัลส์ของสัญญาณที่ส่งออกมา ยังมีส่วนใช้ในการควบคุมการส่งอยู่ อันได้แก่ บิตสตาร์ท บิตสต๊อป และบิตพาริตี้ ทำให้ความเร็วในการส่งอักขระต่อวินาทีน้อยลงไป การส่งสัญญาณด้วยความเร็ว 300 บอด สำหรับการเข้ารหัส 7 บิต ไม่ได้หมายความว่าส่งได้ 300 หารด้วย 7 อักขระต่อวินาที

2.5.4 การสื่อสารข้อมูลแบบอะซิงโครนัส ที่มีการแมตซ์ความเร็ว

เราได้กล่าวถึงการถ่ายโอนข้อมูลจากแบบอนุกรมจากอุปกรณ์เครื่องหนึ่งไปยังอีกเครื่องหนึ่งซึ่งอาจจะเป็นคอมพิวเตอร์หรืออุปกรณ์สื่อสารชนิดอื่น โดยสมมติฐานว่าความเร็วในการเปลี่ยนสัญญาณจากขนานเป็นอนุกรมได้เร็วพอ และฝ่ายรับเปลี่ยนจากอนุกรมเป็นขนาน แล้วนำไปแสดงบนจอพิมพ์ออกที่เครื่องพิมพ์หรือเก็บไว้ในดิสก์ทันทีได้ทันเวลาด้วยความเร็วในแต่ละการ

ทำงานเท่ากันทั้งฝ่ายรับและฝ่ายส่ง ไม่มีการหน่วงเวลาหรือ การอินเทอร์รัพต์ระหว่างกลางอย่างไรก็ตาม การสมมติฐานนี้ย่อมไม่เป็นความจริง ฝ่ายส่งทำหน้าที่ส่งอย่างเดียวแต่ฝ่ายรับอาจจะต้องทำหน้าที่หลายอย่างให้ทันฝ่ายส่ง (แน่นอนย่อมขึ้นกับความเร็วในการส่ง) ก็จำเป็นจะต้องมีกลไกในการควบคุมการถ่ายโอน เทคนิคในการควบคุมความเร็วในการส่งมีอยู่หลายรูปแบบ ซึ่งอาจจะแบ่งออกได้เป็นสองลักษณะ คือข้อมูลบอกการออน – ออฟการทำงาน (On – of Data Flow Toggle) และหาที่เก็บชั่วคราวหรือสร้างบัฟเฟอร์ (Temperary Data Storage Mechanism)

2.5.5 การควบคุมการส่งเมื่อความเร็วในการทำงานของฝ่ายรับและฝ่ายส่งไม่เท่ากัน

เนื่องจากการใช้ภาษาในระดับสูงเขียนเป็นโปรแกรมสำหรับการควบคุมการทำงานของการถ่ายโอนข้อมูลแบบอนุกรม อาจจะใช้เวลามากกว่าที่จะรับข้อมูลเข้ามาได้ทันกับสถานีส่งข้อมูลมา จำเป็นต้องมีวิธีการควบคุมไม่ให้เกิดการสูญหายของข้อมูลที่ส่ง ส่งมาวิธีการดังกล่าวมีอยู่หลายวิธี

2.5.6 การมีบัฟเฟอร์ในการสื่อสารข้อมูล

บัฟเฟอร์สำหรับการสื่อสารก็คือหน่วยความจำในคอมพิวเตอร์ ซึ่งแยกออกมาจากหน่วยความจำหลักสำหรับเก็บข้อมูลในการติดต่อชั่วคราว บัฟเฟอร์สำหรับการสื่อสารนี้ส่วนมากใช้สำหรับฝ่ายรับเท่านั้น เนื่องจากฝ่ายรับจำเป็นจะต้องตามฝ่ายส่งให้ทันถ้าหากฝ่ายรับใช้ภาษาแอสแซมบลี้ควบคุมมีความเร็วพอ อาจไม่จำเป็นต้องใช้บัฟเฟอร์สำหรับการสื่อสารเนื่องจากภาษาแอสแซมบลี้ความเร็วสูง

ข้อมูลที่จัดส่งให้คอมพิวเตอร์ที่เป็นฝ่ายรับ ส่วนมากจะอ่านมาจากแฟ้มที่บันทึกไว้ในดิสก์ หากพิจารณาระหว่างการส่งข้อมูลออก ข้อมูลที่อ่านมาจากดิสก์ก็จะมีลักษณะเป็นกลุ่มที่ได้รับการนำมาสู่บัฟเฟอร์การอ่านแต่ละกลุ่มดำเนินไปจนกระทั่งบัฟเฟอร์เต็ม การอ่านจะหยุดลงจนกระทั่งบัฟเฟอร์จะถูกส่งออกไปหมดในลักษณะของเข้าก่อนออกก่อน ข้อมูลก็จะถูกอ่านออกมาใส่ในบัฟเฟอร์ส่งอีกครั้ง โดยปกติบัฟเฟอร์ส่งจะมีขนาด 255 ตัวอักษรหรือประมาณ 3 บรรทัดของ 80 อักษร

บัฟเฟอร์รับของฝ่ายรับมีผลกระทบต่อการรับ – ส่งข้อมูลมากกว่าบัฟเฟอร์ส่ง บัฟเฟอร์รับทำหน้าที่เช่นเดียวกับบัฟเฟอร์ส่ง แต่ทิศทางการไหลของข้อมูลอยู่ในทางตรงกันข้าม ฝ่ายรับข้อมูลมาเก็บไว้ในบัฟเฟอร์รับ จนกว่าโปรแกรมควบคุมการสื่อสารจะนำข้อมูลออกไปจากบัฟเฟอร์รับเพื่อไปแสดงหรือพิมพ์ หรือเก็บไว้ในแฟ้มก็แล้วแต่ ขอเพิ่มเติมอีกนิดว่าในระบบควบคุมการทำงานของไมโครคอมพิวเตอร์อย่างเช่น IBM PC มีกลไกบัฟเฟอร์รับส่งนี้ไว้รออยู่โปรแกรมในระดับสูง จึงเพียงแต่ทำหน้าที่ดึงเอาข้อมูลจากบัฟเฟอร์นี้ไปใช้เราจะเห็นได้ชัดจากความจำเป็นในการใช้บัฟเฟอร์เมื่อความเร็วในการส่งสูงเกินกว่า 600 บอด ภาษาในระดับสูงเช่น ภาษาเบสิกไม่สามารถที่จะรับข้อมูลจากพอร์ตอนุกรมได้ทันแน่ ๆ ระบบควบคุมการทำงาน จึงถูกออกแบบมาเพื่อการสื่อสารข้อมูลโดยการใช้อินเตอร์รัพต์เข้าช่วย เมื่อมีข้อมูลมาที่พอร์ตอนุกรมเมื่อไร ระบบ

ควบคุมอินเตอร์รัพต์การทำงานเพื่อดึงข้อมูลไปใส่ในบัฟเฟอร์รับทันที เพื่อไม่ให้ข้อมูลที่รับหายไปก่อน เมื่อมีตัวใหม่ส่งมาทับที่พอร์ตอนุกรมหน้าที่ของโปรแกรมควบคุมการรับส่งก็คือการอ่านข้อมูลจากบัฟเฟอร์รับไปใช้เมื่อถูกอ่านจากบัฟเฟอร์รับไปแล้วตัวอ่านที่ออกไปก็จะหายไปจากบัฟเฟอร์ ลองนึกภาพดูจะเห็นว่าฝ่ายหนึ่งคือระบบควบคุมการทำงาน (OS) รับข้อมูลจากพอร์ตอนุกรมใส่บัฟเฟอร์อีกฝ่ายหนึ่งคือโปรแกรมควบคุมการรับส่งดึงข้อมูลออกจากบัฟเฟอร์เปรียบเสมือนคนหนึ่งตักน้ำใส่ตุ่มอีกคนหนึ่งตักออกจากตุ่ม ถ้าฝ่ายที่ตักออกมีความเร็วมากกว่าตุ่มก็จะมีโอกาสแห้ง ในทางตรงกันข้ามถ้าฝ่ายตักออกช้ากว่าฝ่ายตักเข้าโอกาสที่จะล้นตุ่มก็ย่อมจะมี ในทางสื่อสารเรียกว่าบัฟเฟอร์รับโอเวอร์โฟล์ว (Receive Buffer Overflow) การไหลล้นดังกล่าวทำให้ข้อมูลที่ได้รับหายไป

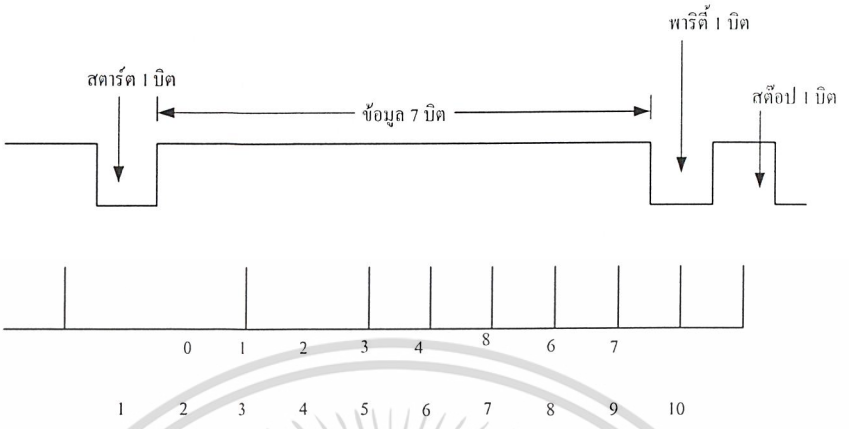
โปรแกรมที่เขียนด้วยภาษาแอสเซมบลี้สามารถทำงานได้เร็ว และอาจไม่จำเป็นต้องใช้บัฟเฟอร์สำหรับการรับส่งเลยก็ได้ แต่ถ้าหากเขียนด้วยภาษาเบสิกจำเป็นจะต้องมีบัฟเฟอร์อย่างน้อย 1024 ไบต์ สำหรับการสื่อสารที่ความเร็วไม่เกิน 600 บอด

ในเครื่อง IBM PC เราสามารถกำหนดบัฟเฟอร์สำหรับการสื่อสารได้เมื่อเราเรียกใช้ BASIC หรือ BASICA เช่น A> BASIC/C:4096

เป็นการกำหนดบัฟเฟอร์การสื่อสารด้วยขนาด 4 กิโลไบต์ ถ้าหากเราไม่กำหนดค่าบัฟเฟอร์สื่อสารเอาไว้ DOS จะตั้งค่าให้เท่ากับ 256 ไบต์ โดยการใส่ /C: ทำให้เราสามารถบัฟเฟอร์ได้จาก 0 ถึง 32,767 ไบต์ อย่างไรก็ตามภาษาเบสิกใช้หน่วยความจำได้ 64 กิโลไบต์ หากเรากำหนดค่าบัฟเฟอร์สื่อสารเอาไว้มากย่อมจะมีหน่วยความจำเหลือสำหรับใช้งานจริงๆ ในขณะที่โหลดโปรแกรมภาษาเบสิกเข้ามาขนาดของโปรแกรม ย่อมจะเข้าไปอาศัยที่ในหน่วยความจำที่เหลือทำให้หน่วยความจำที่จะใช้งานสำหรับเก็บตัวแปรต่างๆ น้อยลงไปอีก เราสามารถใช้คำสั่งโดยตรงสำหรับการตรวจดูขนาดของหน่วยความจำที่ว่างโดยการพิมพ์คำสั่ง PRINT FRE(O)

2.5.7 การควบคุมโดยใช้ XON/XOFF

ถึงแม้ว่าเราจะมีบัฟเฟอร์สำหรับการสื่อสารแล้วก็ตาม ในบางครั้ง การถ่ายโอนข้อมูลด้วยความเร็วสูงและด้วยขนาดของแฟ้มที่จะทำการถ่ายโอนมีขนาดใหญ่กว่าบัฟเฟอร์สื่อสาร โอกาสที่ข้อมูลจะหายไปมีอยู่มาก ลองคำนวณดูง่ายๆ สมมุติว่าใช้ความเร็วในการถ่ายโอนข้อมูล 9600 บิตต่อวินาที สต๊อปบิตเป็น 1 บิต ข้อมูล 7 บิต และพาริตี้เป็นคู่ ใน 1 ตัวอักษรจะต้องใช้ 11 บิต



รูปที่ 2.38 รูปแบบของข้อมูลหนึ่งตัวอักษร

เพราะฉะนั้นฝ่ายรับจะต้องอ่านข้อมูลจากพอร์ตอนุกรมทุก 11/9600 ได้ผลประมาณ (ความเร็วนาฬิกาของ IBM PC = 4.77 MHz หรือประมาณ 0.2 ไมโครวินาทีต่อหนึ่งพัลส์ 1000/0.2=5000) ในเมื่อบัฟเฟอร์ไม่เพียงพอเราก็จำเป็นจะต้องควบคุมการรับส่ง โดยการบอกให้ฝ่ายส่งหยุดส่งชั่วคราว (XOFF) จนกว่าฝ่ายรับจะจัดการเอาข้อมูลออกจากบัฟเฟอร์สื่อสารหมดเสียก่อน จึงบอกให้ฝ่ายส่งจัดการส่งต่อไป (XON) ในรหัส ASCII XON มีค่าเท่ากับ 17 XOFF มีค่าเท่ากับ 19 เป็นหน้าที่ของนักเขียนโปรแกรมที่จะต้องจัดการส่ง XOFF ออกไปให้ฝ่ายส่งได้รู้ก่อนที่บัฟเฟอร์สื่อสารจะเต็มเสียก่อน

คอมพิวเตอร์เมนเฟรมส่วนมากจะมีระบบ XON/XOFF ให้สำหรับการเชื่อมต่อทางด้านความเร็ว (Speed Matching) แต่โปรแกรมสื่อสารที่มีขายสำหรับ IBM PC ไม่มี XON/XOFF ทุกตัวโดยมากโปรแกรมที่เขียนโดยภาษาแอสเซมบลี้จะมี XON/XOFF ให้แต่โปรแกรมที่เขียนโดยภาษาเบสิกจะมีเพียงบางโปรแกรมเท่านั้นโปรแกรมที่เขียนโดยภาษาแอสเซมบลี้มีความเร็วพอที่จะคอยตรวจสอบดู XOFF ที่ส่งมาจากฝ่ายรับแต่ถ้าเป็นภาษาเบสิกความเร็วอาจจะไม่เพียงพอต่อการตรวจสอบ XOFF ที่ฝ่ายส่ง

2.5.8 ฮาร์ดแวร์ที่เกี่ยวข้องกับการสื่อสาร

1) พอร์ต RS232C

โดยปกติไมโครคอมพิวเตอร์จะมีพอร์ตที่เป็นแบบอนุกรม เรียกชื่อกันว่า RS232C อยู่ในตัวเองอยู่แล้ว หลายเครื่องไม่มีมากับเครื่อง อย่างเช่น IBM PC จำเป็นจะต้องมีการ์ดที่เรียกว่าอะซิงโคนัสอะแดปเตอร์ (Asynchronrus Communication Adapter) มาเสียบใส่

พอร์ต RS232C ทำหน้าที่รับและส่งข้อมูลในแบบอนุกรมเรียกว่า Universal Asynchronous Adapter เหตุที่มีชื่อเรียกว่า RS232C ก็เนื่องมาจากสมาคมผู้ผลิตอุปกรณ์อิเล็กทรอนิกส์ของอเมริกา หรือ EIA ได้กำหนดมาตราฐานของอุปกรณ์การสื่อสารแบบอนุกรมเอาไว้ภายใต้ชื่อว่า RS232C ความจริงมาตราฐานการส่งข้อมูลแบบอนุกรมมีหลายมาตราฐาน แต่ที่นิยมใช้กันมากที่สุดสำหรับไมโครคอมพิวเตอร์ก็คือ RS232C

2) หน้าที่สำคัญของการสื่อสารแบบอะซิงโครนัส

2.1 รับสัญญาณ

2.1.1) เปลี่ยนสัญญาณเข้ามาแบบอนุกรมให้เป็นแบบขนาน

2.1.2) ตรวจสอบความผิดพลาดของสัญญาณที่รับ

2.1.3) ตัดสต็อปบิตและพาริตี้บิตออก

2.1.4) ส่งสัญญาณให้ซีพียูรู้ว่ารับสัญญาณได้แล้ว

2.2 ส่งสัญญาณ

2.2.1) เปลี่ยนสัญญาณแบบขนานจากซีพียูต่อและทยอยส่งออกเป็นแบบอนุกรม

2.2.2) เพิ่มสต็อปบิตและพาริตี้บิตออก

2.2.3) เพิ่มสัญญาณควบคุมโมเด็มที่เชื่อมต่อ (ถ้ามี)

บทที่ 3
การออกแบบ การสร้างและการทำงาน

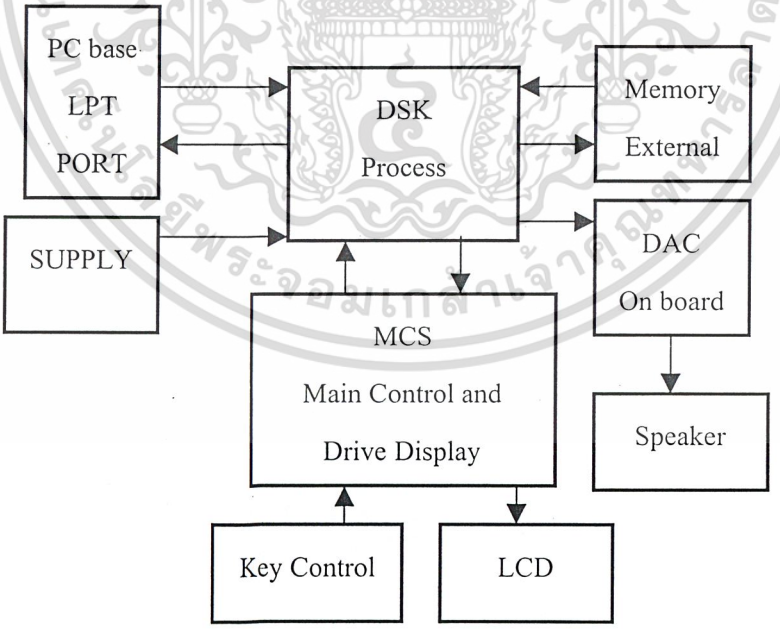
3.1 กล่าวนำ

ในบทนี้จะกล่าวถึงการถอดรหัสข้อมูล "MPEG-1 เลเยอร์-3" (MPEG-1 Layer-3 Decoder) การทำงานคล้ายการเข้ารหัส ซึ่งจะประกอบด้วย 2 ส่วนคือ ฮาร์ดแวร์ และซอฟแวร์ ซึ่งจะรับข้อมูลผ่านทางพอร์ตขนานของเครื่องคอมพิวเตอร์ เพื่อจะทำการถอดรหัสของข้อมูลตามต้องการ

3.2 ส่วนของฮาร์ดแวร์

3.2.1 โครงสร้างของตัวถอดรหัสสัญญาณ MP3

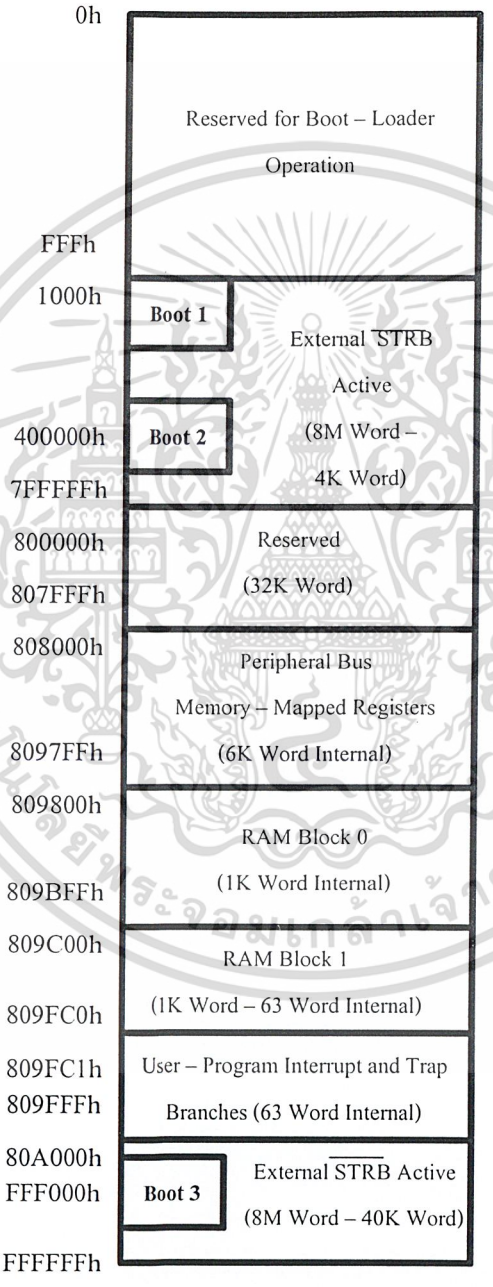
การทำงานของตัวถอดรหัสสัญญาณเสียง MP3 โดยรับข้อมูล MP3 จากเครื่องคอมพิวเตอร์ผ่านทางพอร์ตขนาน ข้อมูลที่จะถูกเก็บไว้ในหน่วยความจำภายนอกของตัวถอดรหัสสัญญาณเสียง MP3 โดยผ่านการประมวลผลด้วยบอร์ด DSK TMS320C31 โครงสร้างของตัวถอดรหัสสัญญาณเสียง MP3 แสดงดังรูปที่ 3.1



รูปที่ 3.1 แผนผังการทำงานตัวถอดรหัสสัญญาณเสียง MP3 ด้วย DSK TMS320C31

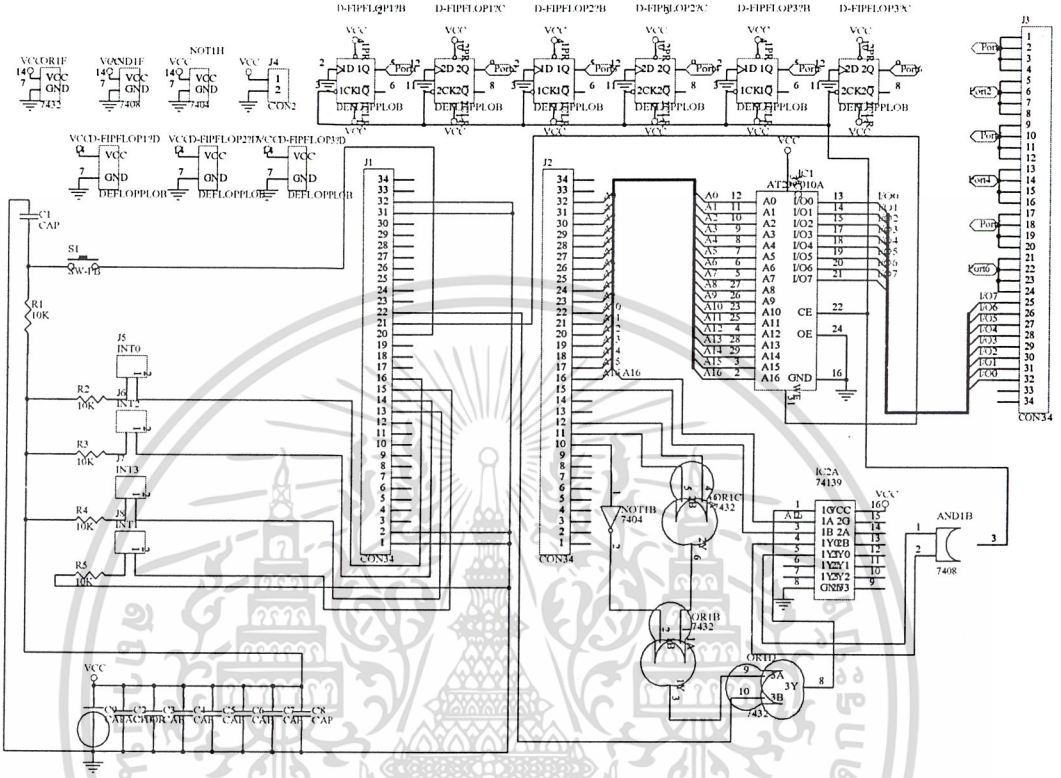
3.2.2 ส่วน Boot Loader

ส่วน Boot Loader จะมีการทำงาน 2 โหมดคือ โหมดไมโครโพรเซสเซอร์ และโหมดไมโครคอมพิวเตอร์ ในโครงงานนี้จะเลือกการทำงานในโหมดไมโครคอมพิวเตอร์โดยไม่ต้องติดต่อกับเครื่องไมโครคอมพิวเตอร์ ซึ่งโหมดของการทำงานมีตำแหน่งหน่วยความจำ ดังรูปที่ 3.2



Microcomputer / Boot Loader Mode

รูปที่ 3.2 ตำแหน่งหน่วยความจำของ Boot Loader



รูปที่ 3.3 วงจร Boot Loader

3.2.3 ส่วนแสดงผล LCD 16 ตัวอักษร 1 บรรทัด

ในการติดต่อกับโมดูล LCD จะต้องมีการหน่วงเวลาหลังจากที่ทำการส่งรหัสคำสั่งหรือข้อมูล เนื่องจากต้องรอคอนโทรลเลอร์ภายใน LCD โมดูล แปลความหมายรหัสคำสั่งและทำงานตามคำสั่งให้เรียบร้อยก่อนจากนั้นจะรับข้อมูลหรือดำเนินการต่อไป

ดังนั้นในการใช้งานโมดูล LCD การเขียนโปรแกรมต้องเขียนโปรแกรมหน่วงเวลารอให้โมดูล LCD พร้อมทำงานด้วย โดยเมื่อเริ่มจ่ายไฟให้แก่โมดูล LCD ต้องรอประมาณ 10 มิลลิวินาที เพื่อให้โมดูล LCD ทำการเตรียมความพร้อม (Initial) หลังจากนั้นจะทำการกำหนดลอจิกให้แก่ขา RS ของโมดูล LCD แล้วต้องหน่วงเวลาอีกประมาณ 2 มิลลิวินาที เพื่อให้คอลโทรลเลอร์ในโมดูล LCD แปลความหมายของลอจิกที่ขา RS ว่า ข้อมูลต่อไปที่จะได้รับนั้นเป็นรหัสคำสั่งหรือเป็นข้อมูลที่ต้องการแสดงผล จากนั้นจะเป็นการส่งข้อมูลมารอที่บัสข้อมูล D0 – D7 (กรณีทำงานในโหมด 8 บิต) ขั้นตอนต่อไปจะทำการส่งสัญญาณนาฬิกาไปที่ขา E เพื่อสั่งงานโมดูล LCD ให้รับข้อมูลจากบัสข้อมูลเข้าไป โดยนาฬิกาที่ป้อนเข้าที่ขา E ของโมดูล LCD ต้องเป็นสัญญาณนาฬิกาขอบขาขึ้น จากนั้นทำการหน่วงเวลา 2 มิลลิวินาที

ทั้งหมดที่กล่าวมาเป็นขั้นตอนและจังหวะในการทำงาน 1 รอบของโมดูล LCD จะเห็นว่ามีโปรแกรมย่อยอยู่ 3 โปรแกรมคือ โปรแกรมเตรียมความพร้อม LCD, โปรแกรมหน่วงเวลา และโปรแกรมย่อยการส่งนาฬิกาเพื่อสั่งงานโมดูล LCD โปรแกรมการทำงานและโฟลวชาร์ตแสดงไว้ในภาคผนวก ค

1) อธิบายโปรแกรมหลักส่วนแสดงผล LCD 16 ตัวอักษร 1 บรรทัด

เริ่มต้นการเตรียมความพร้อมโมดูล LCD ด้วยโปรแกรมย่อย INIT_LCD จากนั้นจะทำการเรียกข้อความที่ต้องการแสดงออกมา โดยต้องแยกออกเป็น 2 ส่วน ส่วนละ 8 ตัวอักษร ข้อความที่นำมาแสดงในตอนแรกคือ โมดูลแสดงผล LCD (LCD Module Show)

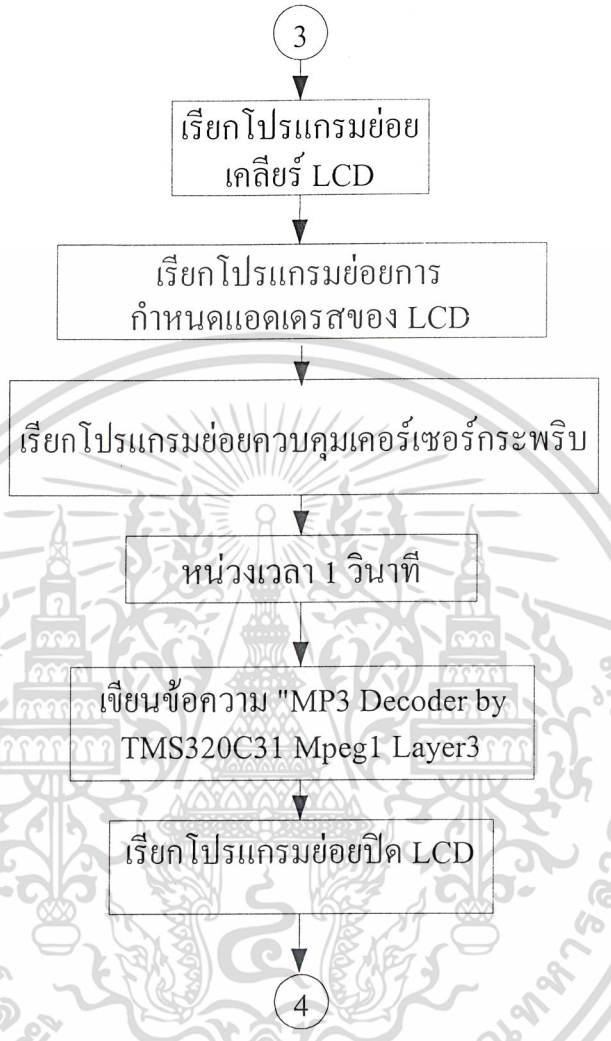
จากนั้นจะทำการเปลี่ยนข้อความ โดยการเลื่อนข้อมูลโดยโปรแกรมย่อย LOOP_LCD_SHF ข้อความที่แสดงใหม่คือ “MP3 Decoder by TMS320C31 MPEG1 Layer3” จากนั้นจะทำการเปลี่ยนข้อความครั้งละ 1 ตัวอักษร โดยใช้โปรแกรมย่อย WRCHAR_LCD ในขณะที่ทำการเขียนต้องแสดงเคอร์เซอร์และกำหนดให้เคอร์เซอร์กระพริบด้วย โดยการใช้โปรแกรมย่อย LCD_BLINK ในการแสดงข้อความจะปรากฏตัวอักษรขึ้นครั้งละ 1 ตัวอักษร และเคอร์เซอร์จะเลื่อนไปทางขวาด้วยโปรแกรม LOOP_LCD_R_SHF ข้อความที่แสดงคือ “MP3 Decoder by TMS320C31 MPEG1 Layer3”

2) อธิบายโปรแกรมย่อยส่วนแสดงผล LCD 16 ตัวอักษร 1 บรรทัด

การทำงานของโปรแกรมทำการเรียกโปรแกรมย่อยในส่วนของการแสดงผล LCD 16 ตัวอักษร 1 บรรทัด ซึ่งมีขั้นตอนการทำงานดังแสดงในแผนผังการทำงานตามรูปที่ 3.4

รูปที่ 3.4 แผนผังการทำงานของโปรแกรมส่วนแสดงผล LCD 16 ตัวอักษร 1 บรรทัด

รูปที่ 3.5 แผนผังการทำงานของโปรแกรมส่วนแสดงผล LCD 16 ตัวอักษร 1 บรรทัด



รูปที่ 3.6 แผนผังการทำงานของโปรแกรมย่อยส่วนแสดงผล LCD 16 ตัวอักษร 1 บรรทัด

รูปที่ 3.7 แผนผังการทำงานของโปรแกรมย่อยส่วนแสดงผล LCD 16 ตัวอักษร 1 บรรทัด

จากแผนผังการทำงานของโปรแกรมส่วนแสดงผล LCD 16 ตัวอักษร 1 บรรทัด ข้างต้น สามารถอธิบายการทำงานในแต่ละส่วนดังนี้

1) INIT_LCD เป็นโปรแกรมสำหรับเตรียม LCD ให้พร้อมในการทำงาน โดยจะเรียกเพียงครั้งแรกครั้งเดียว หลังจากเริ่มจ่ายไฟให้กับตัว LCD โดยจะตั้งค่าการเชื่อมต่อเป็นแบบ 8 บิต เคลียร์ข้อความใน DDRAM ตั้งค่าเพิ่มแอดเดรส DDRAM โดยอัตโนมัติ และเคลียร์ค่าแอดเดรส DDRAM ด้วย

2) LCD_CLR เป็นโปรแกรมย่อยสำหรับเคลียร์ข้อความใน DDRAM ทั้งหมด

3) LCD_HOME เป็นโปรแกรมย่อยสำหรับเคลียร์ค่าแอดเดรส DDRAM เป็น 00H

4) LCD_OFF เป็นโปรแกรมย่อยสำหรับดับการแสดงหน้าจอแสดงผลบน LCD โดยที่ข้อยังคงอยู่

5) LCD_CLK เป็นโปรแกรมย่อยสำหรับการป้อนสัญญาณในการส่งข้อมูลไปยัง LCD

6) LCD_ON เป็นโปรแกรมย่อยสำหรับแสดงข้อมูลหน้าจอแสดงผลบน LCD

7) LCD_BLINK เป็นโปรแกรมย่อยสำหรับแสดงการกระพริบในตำแหน่งที่เคอร์เซอร์อยู่

8) LCD_LSHF เป็นโปรแกรมย่อยสำหรับเลื่อนค่า DDRAM ให้เลื่อนไปทางซ้ายมือ 1 ช่อง

9) LCD_RSHF เป็นโปรแกรมย่อยสำหรับเลื่อนค่า DDRAM ให้เลื่อนไปทางขวามือ 1 ช่อง

10) SET_ADDR_LCD เป็นโปรแกรมย่อยสำหรับตั้งค่าแอดเดรส DDRAM (ตำแหน่งของเคอร์เซอร์) โดยรับค่าตัวแปรจาก LCD_ADDR

11) WRCHAR_LCD เป็นโปรแกรมย่อยสำหรับเขียนค่าตัวแปร LCD_DATA ไปยัง LCD ในตำแหน่งที่เคอร์เซอร์อยู่

12) WRLINE_LCD เป็นโปรแกรมย่อยสำหรับเขียนข้อความ 16 ตัวอักษร จากค่าที่เก็บอยู่ในโปรแกรม โดยจะใช้ร่วมกับรีจิสเตอร์ DPTR ในการอ้างถึงตำแหน่งข้อความ ที่ต้องการนำมาแสดงบน LCD

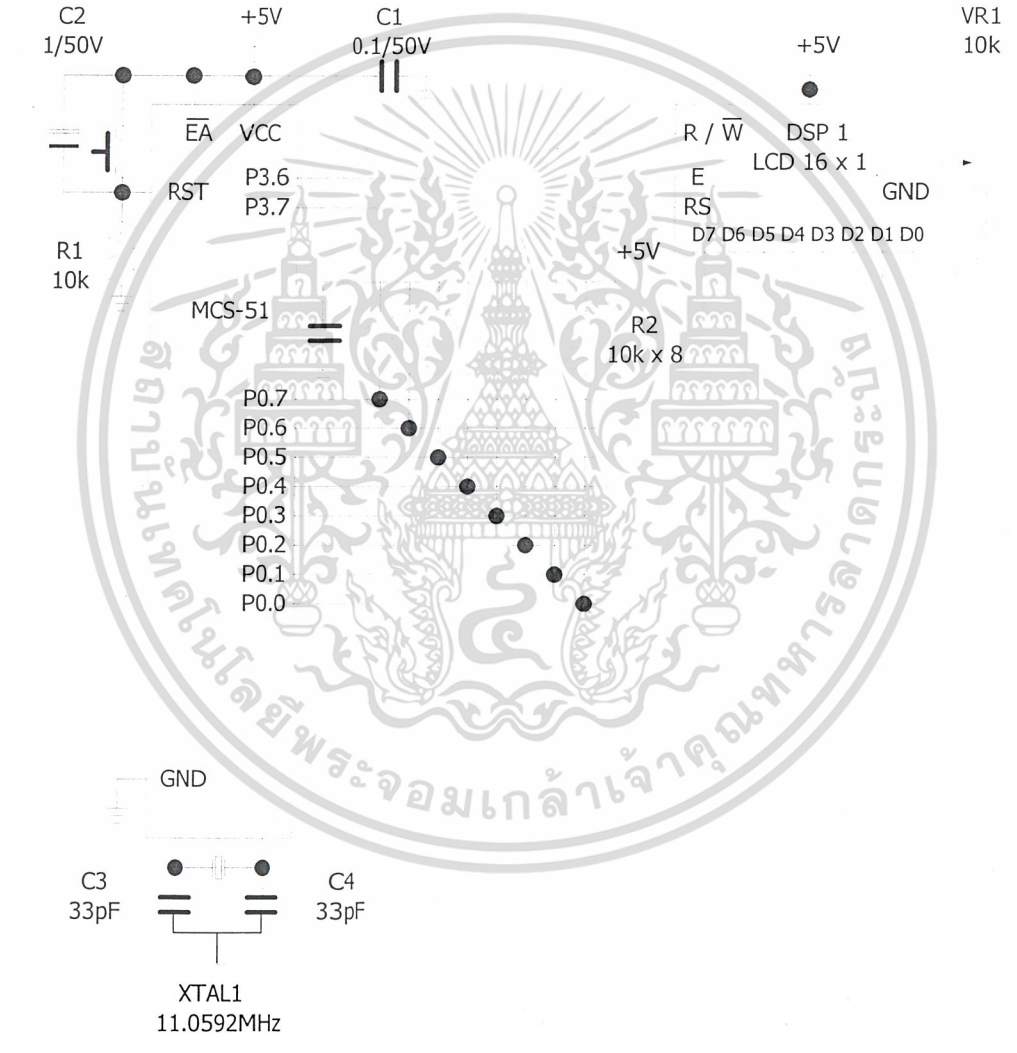
13) WR3CHAR_LCD เป็นโปรแกรมย่อยสำหรับเขียนข้อความ 3 ตัว จากค่าที่เก็บอยู่ในโปรแกรม โดยจะใช้ร่วมกับรีจิสเตอร์ DPTR ในการอ้างถึงตำแหน่งข้อความในตัวแรก และจะใช้รีจิสเตอร์ LCD_ADDR ในการอ้างถึงตำแหน่งข้อความย่อย ที่ต้องการนำมาแสดงบน LCD

14) LOAD_KEY_CHAR เป็นโปรแกรมย่อยสำหรับอ่านสัญลักษณ์รูปกุญแจ จากค่าที่เก็บอยู่ในโปรแกรมโดยจะใช้ร่วมกับรีจิสเตอร์ DPTR ในการอ้างถึงตำแหน่งข้อมูล มาเก็บไว้ในรีจิสเตอร์ CG_ADDR แอดเดรส DDRAM 00H ใช้ร่วมกับโปรแกรมย่อย CG_WR_LOOP

15) CG_WR_LOOP เป็นโปรแกรมย่อยสำหรับอ่านข้อมูลสัญลักษณ์ที่สร้างขึ้นเอง ที่ถูกเก็บอยู่ในโปรแกรมโดยใช้ค่าเริ่มต้นที่อ้างจากรีจิสเตอร์ DPTR เพื่อเก็บข้อมูลที่อ่านได้ในตำแหน่งแอดเดรส DDRAM ที่รีจิสเตอร์ CG_ADDR กำหนดให้

3) วงจรส่วนแสดงผล LCD 16 ตัวอักษร 1 บรรทัด

วงจรส่วนแสดงผล LCD 16 ตัวอักษร 1 บรรทัด ใช้ไมโครคอนโทรลเลอร์ในการควบคุมการแสดงผล ดังแสดงในรูปที่ 3.8 ซึ่งมีโปรแกรมควบคุมการทำงานในภาคผนวก ค



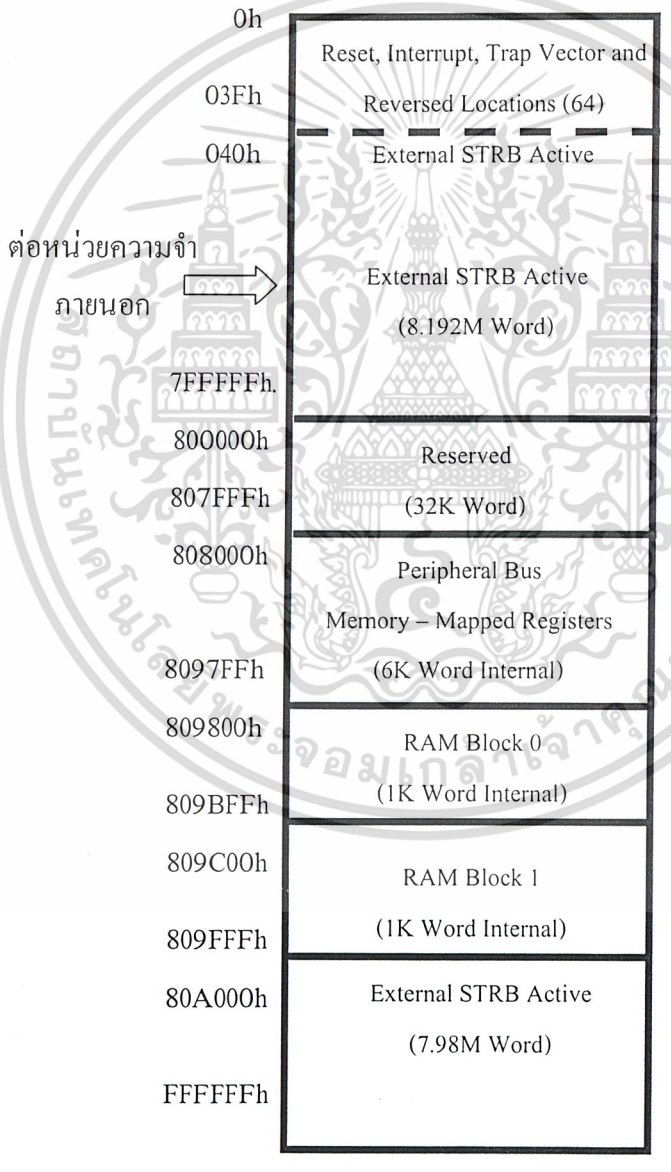
รูปที่ 3.8 วงจรส่วนแสดงผล LCD 16 ตัวอักษร 1 บรรทัด

3.2.4 การต่อหน่วยความจำภายนอก (RAM)

ในการต่อหน่วยความจำ จำเป็นต้องใช้ขาสัญญาณของ TMS320C31 ดังนี้

1) บัสข้อมูล (Data Bus) ขนาด 32 บิต
2) บัสตำแหน่ง (Address Bus) ขนาด 24 บิต
3) ขาสัญญาณควบคุม เช่น STRB, R/W

สำหรับการต่อหน่วยความจำภายนอก จะต้องเลือกตำแหน่งหน่วยความจำที่ว่างไว้ ดังรูปที่ 3.9 ในโครงงานเลือกตำแหน่งที่ 100000H โดยขนาดหน่วยความจำ 128 K Word

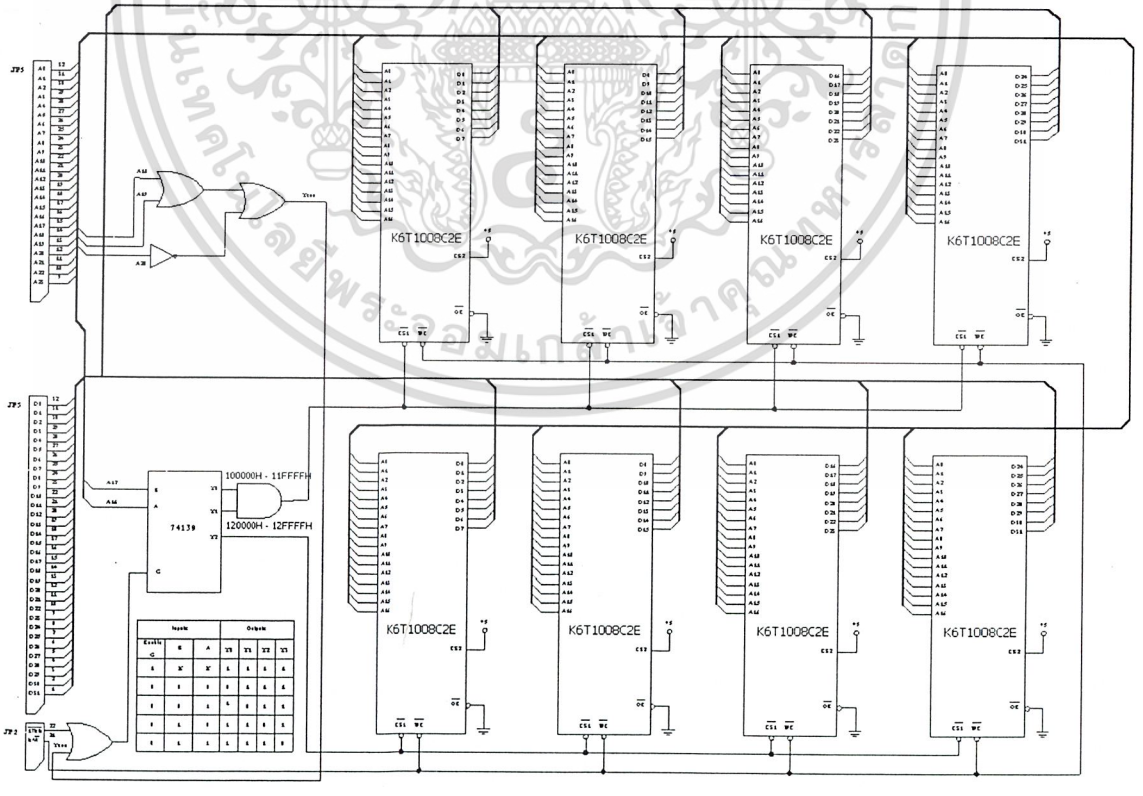


รูปที่ 3.9 ตำแหน่งที่ต่อหน่วยความจำ

สำหรับหน่วยความจำภายนอก ใช้ SRAM เบอร์ K6T1008C2E มีขนาด 128 Kbyte ใช้ 8 ตัว (ได้ขนาดหน่วยความจำภายนอก 256 K Word) โดย TMS320C31 จัดสรรชุดในการต่อหน่วยความจำภายนอก 4 ชุด ชุดละ 32 ขา คือ JP2-3 และ JP5-6 ซึ่งจะอยู่ที่ขอบของบอร์ด ซึ่งจะประกอบไปด้วย ตำแหน่ง (Address), ข้อมูล (Data), V+, GND, R/W, INTO-3 และ STRB ดังนี้

JP2 ขาที่ 1,2,31,32	เป็นขา GND
JP2 ขาที่ 21	เป็นขา R/W
JP2 ขาที่ 22	เป็นขา STRB
JP3 ขาที่ 9-32	เป็นขา Address A8 – A23
JP5 ขาที่ 1-32	เป็นขา Data D0 – D31
JP6 ขาที่ 1,2,31,32	เป็นขา GND
JP6 ขาที่ 23,24	เป็นขา V+, V-

ใช้ SRAM เบอร์ K6T1008C2E ซึ่งมีขนาด 128 Kbyte ใช้ 8 ตัว โดยให้ขา Data 8 บิตนั่นก็คือตัวแรกต่อ D0 – D7, ตัวที่ 2 ต่อ D8 – D15, ตัวที่ 3 ต่อ D16 – D23, ตัวที่ 4 ต่อ D24 – D31 ใช้ IC74139 Decode ที่ตำแหน่ง 100000H ลักษณะการประกอบวงจรดังรูปที่ 3.10



รูปที่ 3.10 การต่อวงจรหน่วยความจำภายนอก

3.2.5 การควบคุมการทำงานบอร์ด DSK

การติดต่อสื่อสารระหว่างอินพุตและเอาต์พุตของ DSK จะมี AIC (Analog Interface Circuit) เป็นตัวติดต่อระหว่างอินพุตและเอาต์พุตของบอร์ด DSK ซึ่งได้อธิบายไปแล้วในบทที่ 2 ในการติดต่อกับ AIC ต้องมีการตั้งค่าเริ่มต้นให้รีจิสเตอร์ที่เกี่ยวข้องและต้องรู้ตำแหน่งรีจิสเตอร์ต่างๆ ดังตารางที่ 3.1

ตารางที่ 3.1 ตำแหน่งรีจิสเตอร์ต่างๆ

รีจิสเตอร์ (Register)	ตำแหน่ง (Address)	คำสั่ง (Command)
Timer 0 period	0x808028	Load 0x1
Timer 0 global control	0x808020	Load 0x3c1
I/O flag	IOF	Load 0x2
SPO transmit port control	0x808042	Load 0x131
SPO receive port control	0x808043	Load 0x131
SPO global control	0x808040	Load 0x0E970300
SPO data control	0x808048	Load 0x0
I/O	IOF	Load 0x6
Interrupt flag	IF	Load 0x0
Interrupt enable	IF	OR 0x10
Status register	ST	OR 0x2000

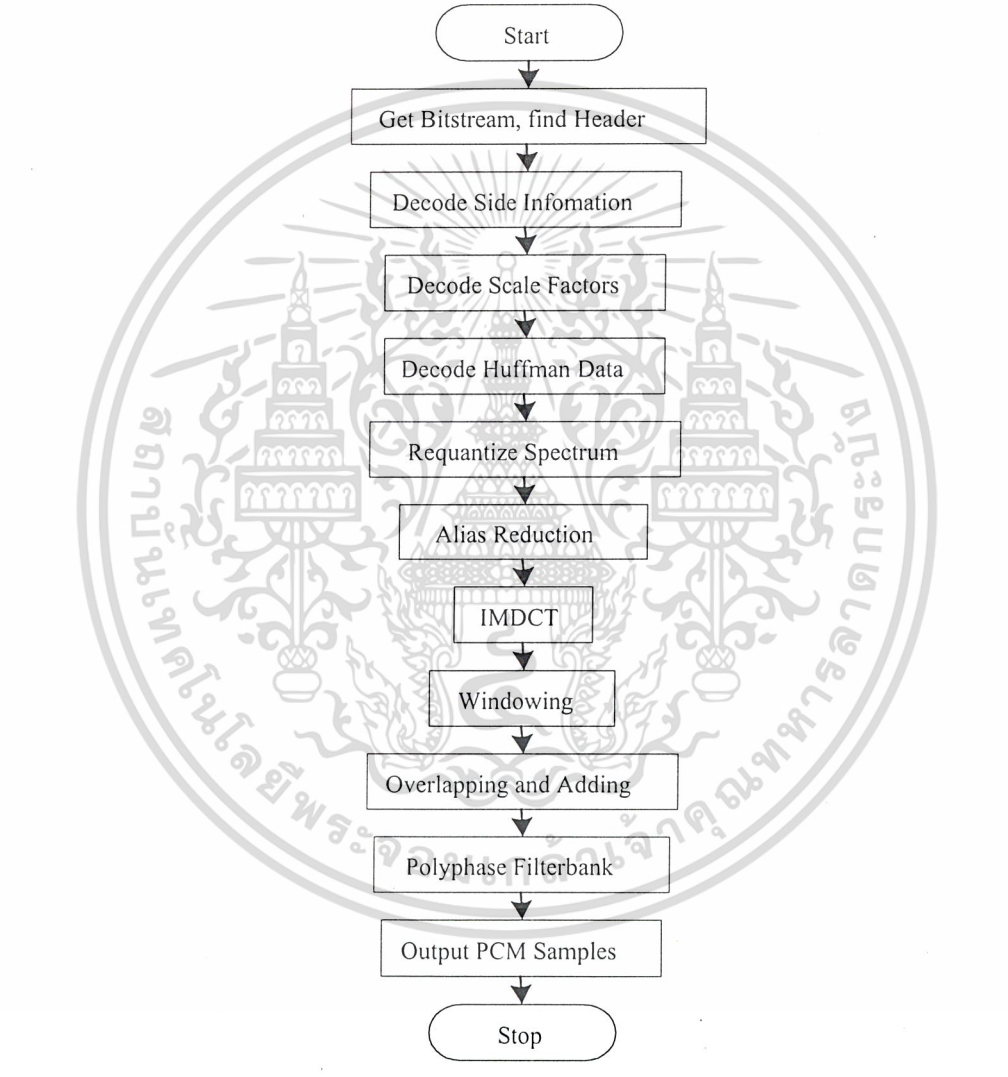
ทำการโปรแกรม AICC0MC.C โดยโปรแกรมนี้จะเป็นการติดต่อสื่อสารระหว่าง Input และ Output สามารถสรุปขั้นตอนได้ดังนี้

1) Set Timer Period ที่ตำแหน่ง 0x808028
2) Set รีจิสเตอร์ Timer Control ที่ตำแหน่ง 0x808020
3) Set IOF (Input / Output Flag) ให้เป็น Low (ต่ำ) เพื่อไปรีเซ็ต AIC
4) Set Xmit Port Control ที่ตำแหน่ง 0x808042 (SPO Transmit Port Control)
5) Set Receive Port Control (พอร์ตรับ) ที่ตำแหน่ง 0x808043
6) Set Serial Port Global Register ที่ตำแหน่ง 0x808040

3.3 ส่วนของซอฟแวร์

3.3.1 ถอดรหัสไฟล์ MP3

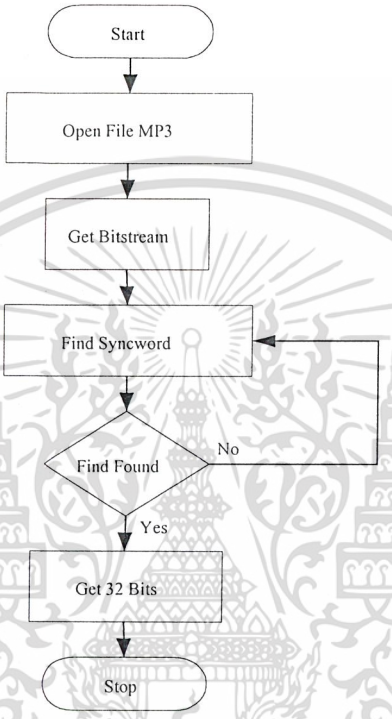
การถอดรหัสไฟล์ MP3 จะมีขั้นตอนตามรูปที่ 3.4 ผังงานของโปรแกรมการถอดรหัสไฟล์ "MPEG-1 เลเยอร์-3" และแผนผังการทำงานของการถอดรหัสแสดงดังรูปที่ 3.11



รูปที่ 3.11 ผังงานของตัวถอดรหัสไฟล์ "MPEG-1 เลเยอร์-3"

3.3.2 การดึงบิตข้อมูลและค้นหาส่วนหัวของเฟรม (Get Bitstream and Find Header)

ในการถอดรหัสสัญญาณเสียงมาตรฐาน MP3 จะต้องเริ่มต้นด้วยการดึงบิตข้อมูลและค้นหาส่วนหัวของเฟรมก่อน ซึ่งสามารถแสดงวิธีการได้ดังรูปที่ 3.12



รูปที่ 3.12 ผังงานของโปรแกรมการดึงข้อมูลและค้นหาส่วนหัว

จากรูปที่ 3.12 สามารถอธิบายได้ดังนี้

1) เปิดไฟล์ MP3 ที่ต้องการทำการถอดรหัส

2) ดึงข้อมูลจากสายการไหลของบิต (Bitstream)

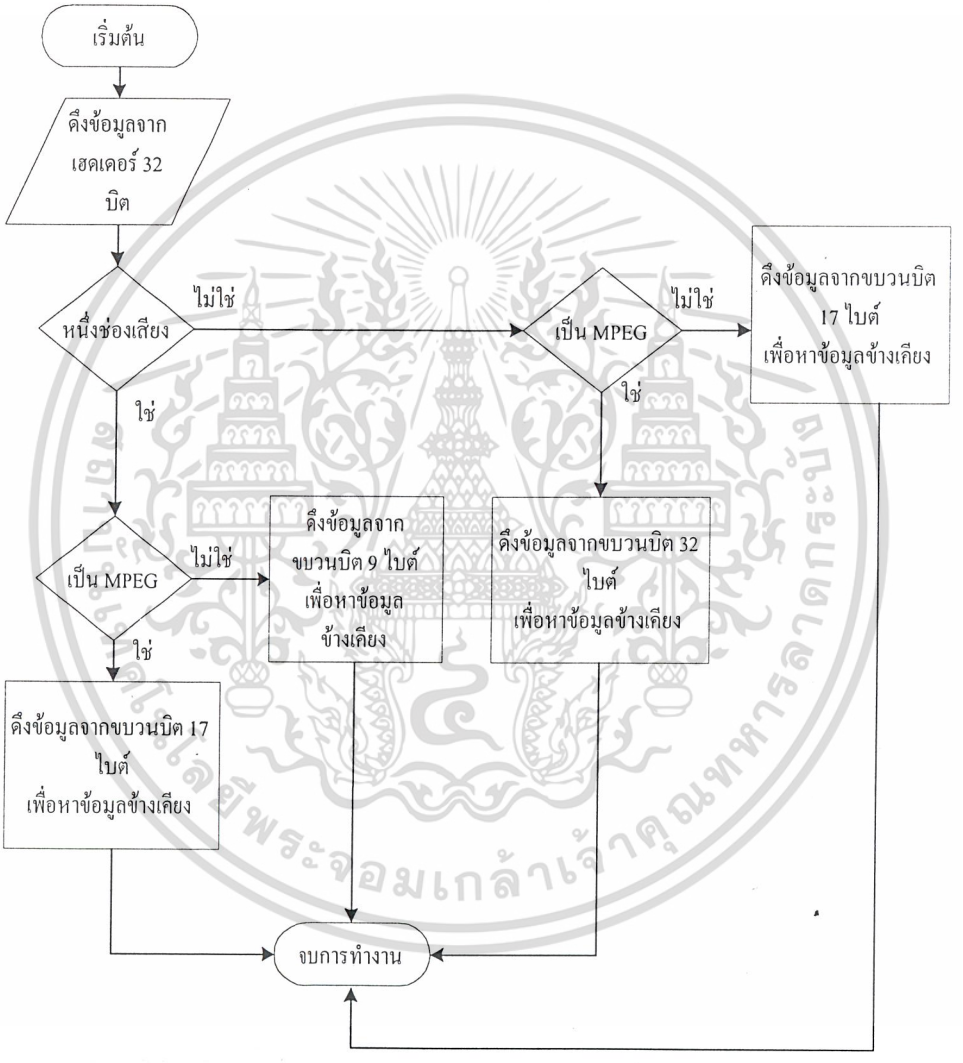
3) ค้นหาส่วนหัวของเฟรมแรก (Header of First Frame) โดยค้นหา "ซิงค์เวิร์ด (Syncword)" ซึ่งเป็นบิต "1" จำนวน 12 ตัวติดกัน

4) ถ้าหาซิงค์เวิร์ดไม่พบจะทำการหาต่อไปเรื่อยๆ เมื่อพบจะไปทำงานในขั้นตอนต่อไป

5) ดึงข้อมูลจากบิตสตรีมมา 32 บิต โดยเริ่มจากบิตแรกของซิงค์เวิร์ด เพื่อทำมาถอดรหัสหาคุณสมบัติของไฟล์นั้นๆ ซึ่งรายละเอียดของแต่ละบิตของส่วนหัวข้อมูลได้กล่าวไว้แล้วในบทที่ 2

3.3.3 การถอดรหัสข้อมูลข้างเคียง (Decode Side Information)

ข้อมูลข้างเคียง (Side Information) จะเป็นส่วนต่อจากเฮดเดอร์ (ถ้าไม่มี CRC Check Error) ซึ่งมีความสำคัญมากเหมือนกัน เพราะเป็นตัวบอกถึงค่าต่างๆ ที่จำเป็นในการถอดรหัส เช่น ตารางฮัฟฟ์แมน (Huffman Table) เป็นต้น มีกระบวนการถอดรหัสดังรูปที่ 3.13



รูปที่ 3.13 ผังงานของโปรแกรมการถอดรหัสข้อมูลข้างเคียง

จากรูปที่ 3.13 การทำงานของการถอดรหัสข้อมูลข้างเคียงเริ่มจากการดึงข้อมูลของส่วนหัว ข้อมูลที่ถอดรหัสแล้วมาเปรียบเทียบหาระบบเสียงของเพลงถ้าเป็นแบบสเตอริโอจะดึงข้อมูลต่อจาก

ส่วนหัวข้อมูลจำนวน 32 ไบต์ แต่ถ้าเป็นแบบโมโนจะดึงข้อมูลจำนวน 17 ไบต์ เพื่อนำมาถอดรหัสหาข้อมูลของการเข้ารหัสต่างๆ เพื่อนำไปใช้ในการถอดรหัสข้อมูลในส่วนต่อไป โดยข้อมูลที่ได้จากการถอดรหัสข้อมูลข้างเคียงมีดังนี้

1) main_data_begin คือจุดเริ่มต้นของข้อมูลหลัก

2) private_bits ในอนาคตจะไม่ถูกใช้ในมาตรฐาน ISO

3) scfsi ใช้เลือกค่าสเกลแฟกเตอร์ที่ใช้ในการถอดรหัสในแต่ละย่านความถี่ และแต่ละช่องสัญญาณเสียง

4) big_values บอกจำนวนของคู่ข้อมูลที่ถูกเข้ารหัสฮัฟฟ์แมน ซึ่งค่านี้จะใช้ในการคำนวณหาจุดสิ้นสุดของภาคที่ 1-3

5) global_gain เป็นตัวแปรที่ใช้ในขั้นตอนการรีควอนไตซ์

6) scalefac_compress บอกจำนวนของบิตที่ถูกใช้สำหรับส่งค่าสเกลแฟกเตอร์ (Scale Factor) ดังตารางที่ 3.2

ตารางที่ 3.2 ความหมายของค่า scalefac_compress

Scalefac_compress	slen1	slen2
0	0	0
1	0	1
2	0	2
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

ตารางที่ 3.2 (ต่อ) ความหมายของค่า scalefac_compress

Scalefac_compress	slen1	slen2
13	3	3
14	4	2
15	4	3

7) window_switching_flag ถ้าเป็น 0 แสดงว่าวินโดว์ไม่ถูกใช้ ถ้าเป็น 1 การทำงานจะขึ้นกับ block_type

8) block_type บอกชนิดของวินโดว์ที่ใช้ ดังตารางที่ 3.3

ตารางที่ 3.3 ความหมายของรหัสข้อมูลใน block_type

block_type	ความหมาย
0	ไม่ใช้
1	เริ่มต้นบล็อก
2	วินโดว์บล็อกสั้นสามบล็อก
3	ท้ายบล็อก

9) mixed_block_flag ถ้าเป็น 1 แสดงว่าเป็นบล็อกผสม ถ้าเป็น 0 แสดงว่าไม่เป็นบล็อกผสม

10) table_select ใช้เลือกตารางฮัฟฟ์แมนของการถอดรหัสภาคที่ 1-3 อยู่ในตารางที่ ค.3

11) subblock_gain บอกค่าอัตราการขยายที่เพิ่มขึ้น/ลดลงจาก global_gain ของแต่ละบล็อกย่อย (Subblock)

12) region0_count ถูกใช้ชี้ขอบเขตของภาคที่ 1

13) region1_count ถูกใช้ชี้ขอบเขตของภาคที่ 2

14) preflag เป็นค่าเสริมเพื่อขยายค่าที่ความถี่สูง ถ้าค่า preflag ถูกตั้ง จะนำค่าในตาราง preflag ไปคูณกับค่าของสเกลแฟกเตอร์อีกครั้ง ในกรณี block_type==2 preflag ไม่ถูกใช้ ค่า preflag

15) scalefac_scale เป็นสเกลแฟกเตอร์ที่ถูกจัดระดับ (Quantize) ด้วยสเกลอัลกอการิทึม โดยคูณค่าของแต่ละลำดับด้วย 2 หรือ /2 ขึ้นอยู่กับ scalefac_scale ดังตารางที่ 3.4

ตารางที่ 3.4 ความหมายชองรหัสข้อมูลใน scalefac_scale[gr]

scalefac_scale[gr]	Scalefac_multipier
0	0.5
1	1

16) count1table_select ใช้เลือกตารางฮัฟฟ์แมนของการถอดรหัสในภาคที่ 4

17) part2_3_length จุดสิ้นสุดของการถอดรหัสฮัฟฟ์แมนภาคที่ 4

ต่อจากนี้จะเริ่มเข้าสู่ข้อมูลเสียงหลัก (Main Audio Data) ของเฟรมโดยจะมีขนาดเท่ากับขนาดของแต่ละเฟรม (N) ลบด้วยเฮดเดอร์ (Header (Bit)) และลบด้วยข้อมูลข้างเคียงดังสมการที่ 3.1

$$Main\_Data(bit) = N - Header(bit) - Side\_Infomation(bit) \tag{3.1}$$

สำหรับเลเยอร์-3 จะมีขนาดของแต่ละเฟรม (N) ดังสมการที่ 3.2

$$N = 144000 \times \frac{Bitrate}{Sampling\_Frequency} \tag{3.2}$$

ข้อมูลหลักของเฟรมนั้นๆ จะถูกดึงมาเก็บไว้ในบัฟเฟอร์ (Buffer) เพื่อนำไปใช้ในการถอดรหัส MPEG-1 เลยอร์-3

3.3.4 การถอดรหัสสเกลแฟกเตอร์ (Decode ScaleFactors)

สเกลแฟกเตอร์อยู่ในรูปของบิตที่เรียงต่อๆ กันในส่วนของข้อมูลหลัก โดยขึ้นอยู่กับค่า slen1 กับ slen2 (2 ค่านี้ถูกกำหนดโดยค่าของ scalefac_compress[gr] ตามตารางที่ 3.2) จำนวนบิตทั้งหมดของสเกลแฟคเตอร์เรียกว่า "part2_length" ค่าของ part2_length จะขึ้นกับค่าของ Block_type และ mixed_block_flag ซึ่งค่าของ scalefac_compress, Block_type และ mixed_block_flag ได้จากการถอดรหัสข้อมูลข้างเคียง

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

part2_length = 11*slen1 + 10*slen2

สำหรับ block_type = 2 (short block) และ mixed_block_flag = 0

part2_length = 18*slen1 + 18*slen2

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

part2_length = 17*slen1 + 18*slen2

กำหนดให้ "scalefac_l" แทนสเกลแฟคเตอร์ของบล็อกยาว (long block)

"scalefac_s" แทนสเกลแฟคเตอร์ของบล็อกสั้น (short block)

ค่าของสกเลแฟคเตอร์สามาถจำแนกได้ดังนี้

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 mixed_block_flag = 0 มีแต่บล็อกสั้นโดย

ค่า scalefac_s 18 ตัวแรก มีจำนวนบิตเท่ากับค่าของ slen1

ค่า scalefac_s 18 ตัวหลัง มีจำนวนบิตเท่ากับค่าของ slen2

3) ค่าของ block_type = 2 และ mixed_block_flag = 1 มีทั้งบล็อกยาวและสั้นโดย

ค่า scalefac_l 8 ตัวแรก มีจำนวนบิตเท่ากับค่าของ slen1

ค่า scalefac_s 9 ตัวต่อมา มีจำนวนบิตเท่ากับค่าของ slen1

ค่า scalefac_s 18 ตัวหลัง มีจำนวนบิตเท่ากับค่าของ slen2

3.3.5 การถอดรหัสข้อมูลฮัฟฟ์แมน (Decode Huffman Data)

การถอดรหัสข้อมูลฮัฟฟ์แมน (Huffman Data) ใน 1 เฟรม แบ่งเป็น 4 ภาค (Region) ซึ่งแต่ละภาคไม่จำเป็นต้องใช้ตารางฮัฟฟ์แมนเดียวกัน ก่อนอื่นต้องกำหนดตารางฮัฟฟ์แมนให้แต่ละภาคก่อน โดยค่าของ 3 ภาคแรกถูกเก็บใน table_select และ ของภาคสุดท้ายถูกเก็บใน count1table_select โดยตารางฮัฟฟ์แมนที่ใช้ในการถอดรหัสจะอยู่ที่ภาคผนวก ค

ต่อไปเป็นการกำหนดว่าในแต่ละภาคใน 3 ภาคแรกจะมีจุดสิ้นสุดที่ตัวที่เท่าใด โดยจะมีขั้นตอนตามรูปที่ 3.11 แผนผังการทำงานของโปรแกรมการกำหนดค่าของจุดสิ้นสุดของ 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

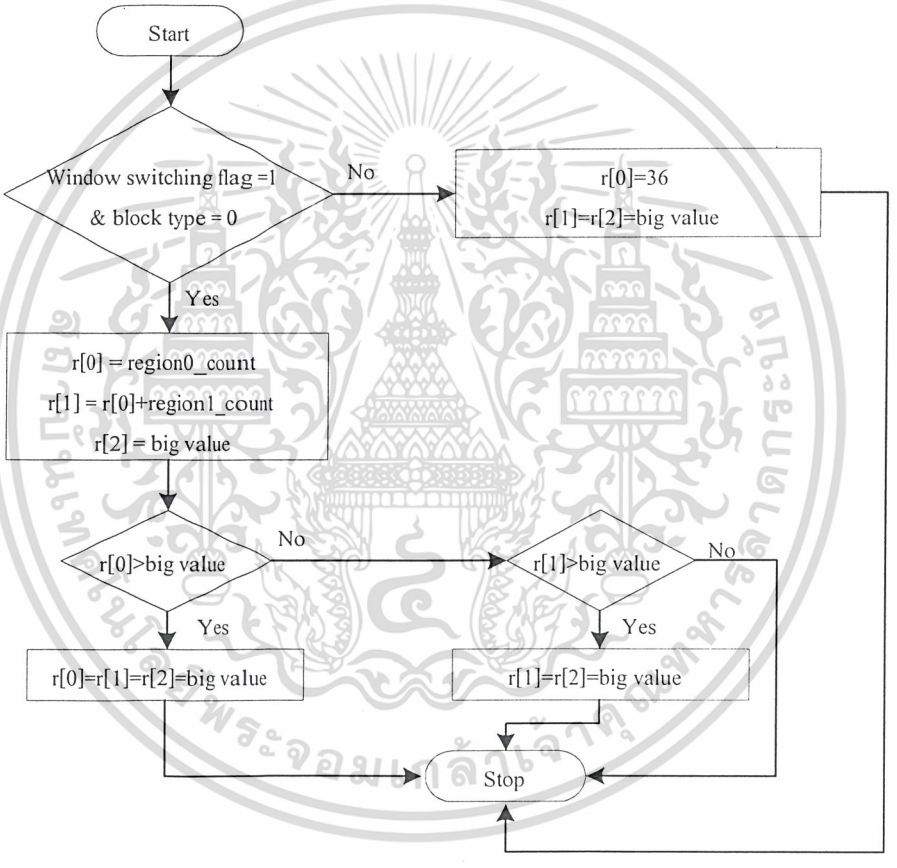
หลังจากกำหนดค่าต่างๆ แล้ว ต่อไปจะเป็นการเริ่มถอดรหัสข้อมูลฮัฟฟ์แมนมีขั้นตอนดังนี้

1) ถอดรหัสในภาคแรกก่อน โดยใช้ตารางฮัฟฟ์แมนและค่า linbits ของภาคแรก
2) นำหัวของขบวนบิต มาเปรียบเทียบกับค่า hcod ในตารางฮัฟฟ์แมน
3) ถ้าตรงกับค่า hcod ตัวไหนก็จะนำค่า x และ y ของ hcod ตัวนั้นมา
4) ถ้าค่า x = 15 ต้องดึงข้อมูลจากขบวนบิตซึ่งมีจำนวนบิตเท่ากับค่าของ linbits มาบวกเข้ากับ x
5) ถ้า x ไม่เท่ากับ 0 ต้องดึงข้อมูลมาอีก 1 บิตเพื่อกำหนดเครื่องหมาย โดยที่

   ถ้าบิตที่ดึงมามีค่าเป็น 0 x จะมีค่าเป็นบวก

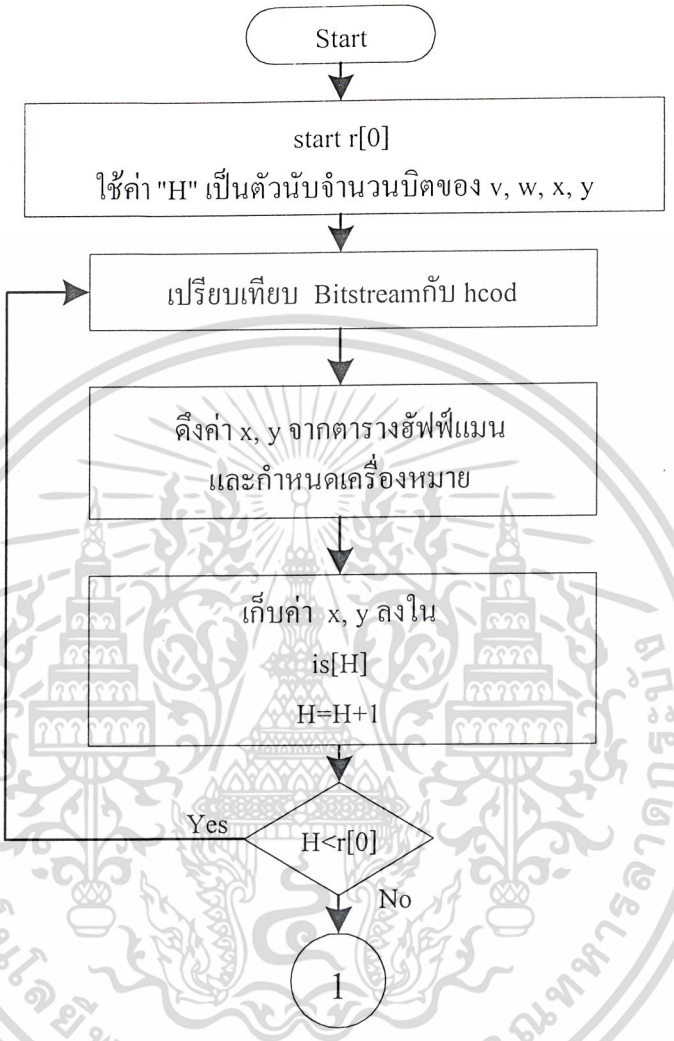
   ถ้าบิตที่ดึงมามีค่าเป็น 1 x จะมีค่าเป็นลบ
6) สำหรับค่า y ทำเช่นเดียวกับค่า x
7) เก็บค่า x และ y ลงใน “is”
8) เมื่อจำนวนตัวรวมของ x และ y เท่ากับ r[0] แล้วก็จะเริ่มถอดรหัสฮัฟฟ์แมนในภาคที่ 2 และ 3 ต่อไป โดยการถอดรหัสจะเป็นเช่นเดียวกับภาคแรก แต่จะใช้ตารางฮัฟฟ์แมนและค่า linbits ของภาคที่ 2 และ 3 ตามลำดับ

9) ในภาคที่ 4 จะใช้ค่าจาก count1table_select โดยค่านี้จะใช้ชี้ตารางฮัฟฟ์แมน (Huffman Table) A และ B เท่านั้น (2 ตารางนี้จะให้ค่า 4 ค่าคือ v, w, x, y จากค่า hcod 1 ค่า) ส่วนการถอดรหัสจะเหมือนกับภาคที่ 1 ต่างกันเพียงแต่จะได้ค่าจาก 4 ค่า จุดสิ้นสุดของ ภาคที่ 4 จะถูกกำหนดโดยค่า part2_3_length ใน Side Information ลบด้วยค่า part2_length (จากหัวข้อที่ 3.3) แต่ถ้าค่าของ part2_3_length – part2_length มีค่ามากกว่า 576 จุดสิ้นสุดของภาคที่ 4 จะอยู่ที่ 576

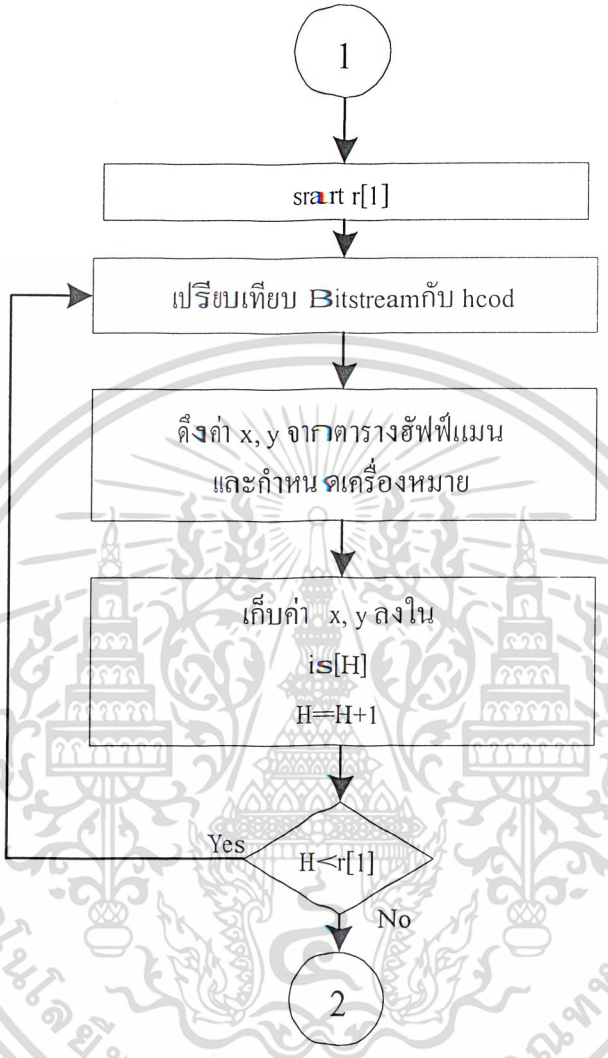


รูปที่ 3.14 ผังงานของโปรแกรมการกำหนดค่าของจุดสิ้นสุดของ 3 ภาคแรก

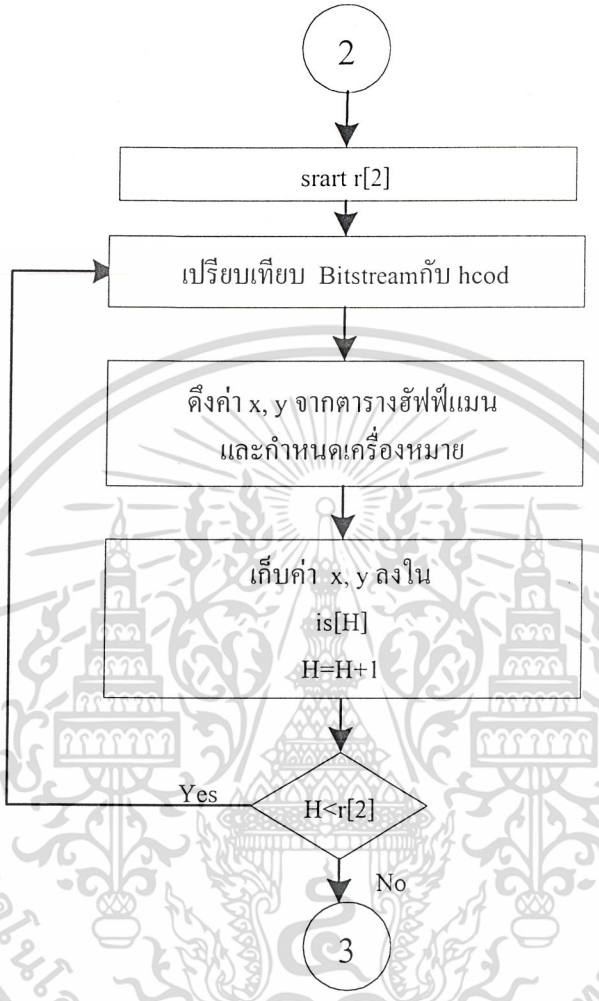
10) ในกรณีที่ค่าของ part2_3_length – part2_length มีค่าน้อยกว่า 576 ค่าที่เหลือจากตำแหน่งที่ค่าของ part2_3_length – part2_length ชี้อยู่จนถึง 576 จะให้มีค่าเป็น "0" ขั้นตอนต่างๆ เขียนเป็นแผนผังการทำงานของโปรแกรมได้ดังรูปที่ 3.15



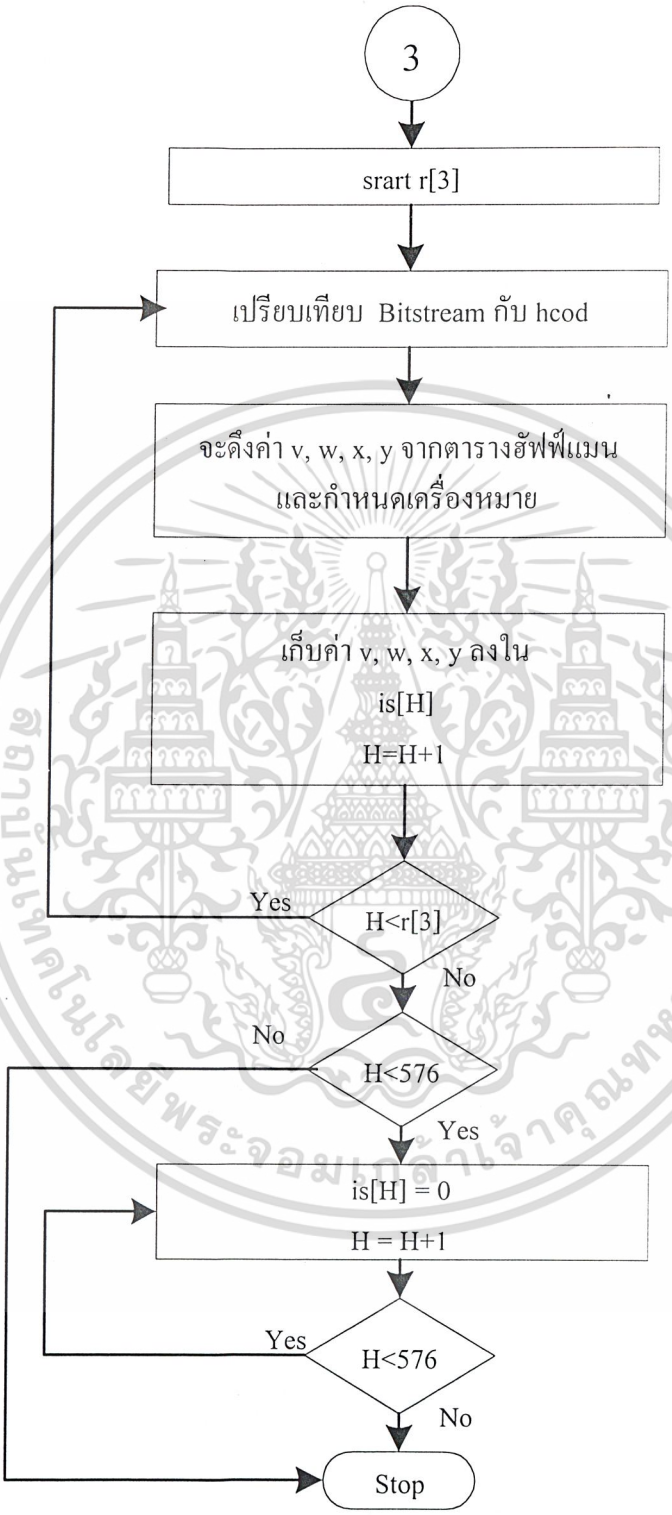
รูปที่ 3.15 ผังงานของโปรแกรมการถอดรหัสฮัฟฟ์แมน



รูปที่ 3.16 ผังงานของโปรแกรมการถอดรหัสฮัฟฟ์แมน



รูปที่ 3.17 ผังงานของโปรแกรมการถอดรหัสฮัฟฟ์แมน



รูปที่ 3.18 ผังงานของโปรแกรมการถอดรหัสฮัฟฟ์แมน

3.3.6 การรีควอนไตซ์ (Requantiza Spectrum)

เป็นการนำเอาค่าจาก "is" ซึ่งได้จากการถอดรหัสฮัฟฟ์แมนมาทำการรีควอนไตซ์จะได้ผลลัพธ์เป็นสัญญาณที่ถูกสุ่มตัวอย่าง (Sampling Signal) ในแกนความถี่ (Frequency Domain) จะกำหนดให้สัญญาณนี้เป็น "xr_i"

ในการรีควอนไตซ์จะใช้สูตรทางคณิตศาสตร์ตามสมการ (3.3) ซึ่งใช้กับบล็อกสั้น

$$xr_i = is_i^{4/3} * 2^{0.25*(global\_gain[gr]-64-8*subblock\_gain[window][gr])} * 2^{0.25*\left[-2*(1+scalefac\_scale[gr])*scalefac[cb][window][gr]-2*preflag[gr]*(1+scalefac\_scale[gr])*pretab[cb]\right]} \quad (3.3)$$

ซึ่งจะเห็นได้ว่าค่าของ xr_i จะขึ้นกับค่าของ

1) is_i เป็นค่าเอาต์พุตจากฮัฟฟ์แมน

2) global_gain[gr] เป็นค่าใน Side Information

3) scalefac_multipier = scalefac_scale +1 (ค่า scalefac_scale อยู่ใน Side Information)

4) preflag[sfg] เป็นค่าใน Side Information

5) pretab[sfb] ถูกกำหนดโดย Scalefactor Band ดังตารางที่ 3.2

ตารางที่ 3.5 ค่า 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

6) subblock_gain[window][gr] เป็นค่าใน Side Information

7) ค่า scalefac_l คือค่าสกเลแฟกเตอร์ของบล็อกยาวที่ได้จากการถอดรหัสสเกลแฟกเตอร์

8) ค่า scalefac_s คือค่าสกเลแฟกเตอร์ของบล็อกสั้นที่ได้จากการถอดรหัสสเกลแฟกเตอร์

เพราะฉะนั้น ในหัวข้อนี้จะมีเพียงการแทนค่าตามสูตรก็จบกระบวนการ 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 36 ตัวแรกจะเป็นบล็อกยาวที่เหลือจะเป็นบล็อกสั้นหมด

2) จาก 1 ถ้าไม่ใช่ xr_i ทุกตัวจะเป็นบล็อกยาวหมด

3.3.7 การลดค่าปลอม (Alias Reduction)

ในกรณีที่ค่า xr_i ซึ่งได้จากการรีควอนไตซ์เป็นบล็อกยาว (block_type ไม่เท่ากับ 2) จะเกิดการเหลื่อมกันของสเปกตรัมจนทำให้เกิดค่าปลอมของ xr_i ขึ้น 1 ตัว (เกิด Alias) จึงต้องทำการลดค่าปลอม (Alias Reduction) ลง

การคำนวณของการลดค่าปลอมในการถอดรหัสมีการคำนวณที่เหมือนกับการเข้ารหัสโดยใช้หน่วยคำนวณผีเสื้อซึ่งมีรูปแบบทั่วไป ส่วนนิยามของการลดค่าปลอมโดยหน่วยคำนวณผีเสื้อของการถอดรหัสแสดงดังรูปที่ 3.20

รูปที่ 3.19 นิยามของการลดค่าปลอมโดยหน่วยคำนวณผีเสื้อของการถอดรหัส

โดยการลดค่าปลอมสามารถหาค่าของ Cs_i และ Ca_i ได้จากสมการ (2.5) และ (2.6) ตามลำดับ โดยค่า C_i ซึ่งเป็นค่าสัมประสิทธิ์ในการคำนวณ Cs_i และ Ca_i จะถูกกำหนดไว้ตามมาตรฐานดังตารางที่ 2.8

เมื่อหาค่าของ Cs_i และ Ca_i ได้เรียบร้อยก็จะนำค่าทั้งสองค่านี้ไปคำนวณตามวิธีการของหน่วยคำนวณผีเสื้อต่อไปเพื่อให้ได้ค่าของ xr_i ที่แท้จริง

ตารางที่ 3.6 ค่าสัมประสิทธิ์ของการลดค่าปลอม

(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

3.3.8 IMDCT (Inverse Modified Discrete Cosine Transform)

เป็นการแปลงสัญญาณในแกนความถี่ (Frequency Domain) เป็นสัญญาณเชิงเวลา (Time Domain) โดยใช้สูตรดังสมการที่

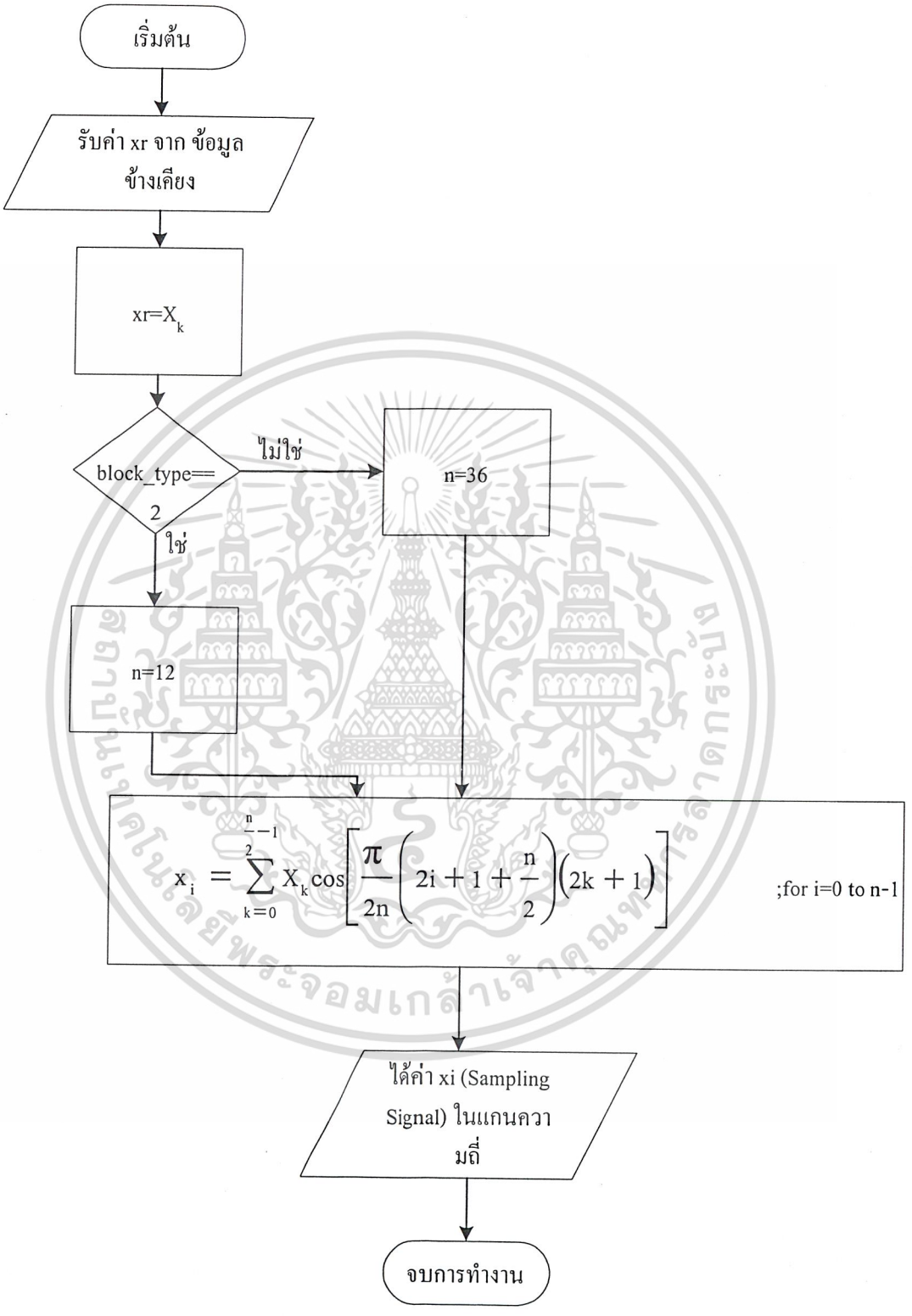
$$x_i = \sum_{k=0}^{\frac{n}{2}-1} X_k \cos\left(\frac{\pi}{2n}\left(2i+1+\frac{n}{2}\right)\left(2k+1\right)\right) \quad ;for\ i=0\ to\ n-1 \tag{3.7}$$

โดยที่ x_i เป็นสัญญาณในแกนเวลา (Time Domain)

X_k เป็นสัญญาณในแกนความถี่ (Frequency Domain)

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

วิธีการแปลงสัญญาณในแกนความถี่เป็นสัญญาณเชิงเวลาแสดงได้ดังรูปที่ 3.21



รูปที่ 3.20 ผังงานของโปรแกรมการทำ IMDCT

3.3.9 การทำวินโดว์ (Windowing)

การทำวินโดว์จะใช้ค่าของ x_i ที่ได้จากการทำ IMDCT โดยจะได้ผลลัพธ์เป็น z_i โดยการทำวินโดว์ขึ้นกับค่า block_type ดังนี้

1) block_type = 0 (normal window)

ใช้สูตรดังสมการที่ 3.8

$$z_i = x_i \sin\left(\frac{\pi}{36}\left(i+\frac{1}{2}\right)\right) \quad ;for\ i=0\ to\ 35 \tag{3.8}$$

2) block_type = 1 (start block)

ใช้สูตรดังสมการที่ 3.9

$$\begin{aligned} z_i &= x_i \sin\left(\frac{\pi}{36}\left(i+\frac{1}{2}\right)\right) && ;for\ i=0\ to\ 17 \\ z_i &= x_i && ;for\ i=19\ to\ 23 \\ z_i &= x_i \sin\left(\frac{\pi}{12}\left(i-18+\frac{1}{2}\right)\right) && ;for\ i=24\ to\ 29 \\ z_i &= 0 && ;for\ i=30\ to\ 35 \end{aligned} \tag{3.9}$$

3) block_type = 0 (stop block)

ใช้สูตรดังสมการที่ 3.10

$$\begin{aligned} z_i &= 0 && ;for\ i=1\ to\ 5 \\ z_i &= x_i \sin\left(\frac{\pi}{12}\left(i+18+\frac{1}{2}\right)\right) && ;for\ i=6\ to\ 11 \\ z_i &= x_i && ;for\ i=21\ to\ 17 \\ z_i &= x_i \sin\left(\frac{\pi}{36}\left(i+\frac{1}{2}\right)\right) && ;for\ i=18\ to\ 35 \end{aligned} \tag{3.10}$$

4) block_type = 0 (short block)

มี 3 window แยกกัน window ละ 12 ตัว โดยที่

$$y_i^{(j)} = x_i^{(j)} sin\left(\frac{\pi}{12}\left(i+\frac{1}{2}\right)\right) \qquad ;for\ i=0\ to\ 11\ ,\ j=0to\ 2 \qquad (3.11)$$

โดยที่ i เป็นลำดับสัญญาณ

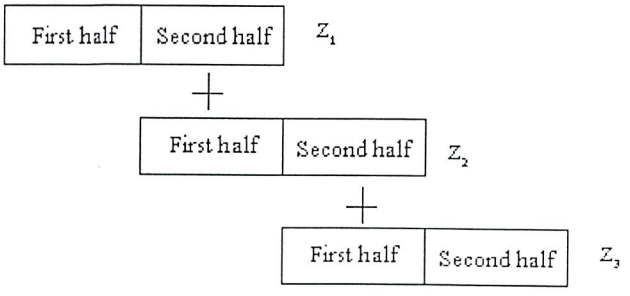
j เป็นลำดับวินโดว์

ใช้สูตรดังสมการที่ 3.12

$$\begin{aligned}
z_i &= 0 && ;for\ i=0\ to\ 5\\
z_i &= y_{i-6}^{(1)} && ;for\ i=6\ to\ 11\\
z_i &= y_{i-6}^{(1)} + y_{i-12}^{(2)} && ;for\ i=12\ to\ 17\\
z_i &= y_{i-12}^{(2)} + y_{i-18}^{(3)} && ;for\ i=17\ to\ 23\\
z_i &= y_{i-18}^{(3)} && ;for\ i=24\ to\ 29\\
z_i &= 0 && ;for\ i=30\ to\ 35
\end{aligned} \qquad (3.12)$$

3.3.10 การทำโอเวอร์แลป (Overlapping)

ครึ่งแรกของบล็อกของค่า 36 ค่า จะทับกับครึ่งหลังของบล็อกก่อนหน้าที่ (Previous Block) ดังรูปที่ 3.22



รูปที่ 3.21 การซ้อนทับ (Overlapping) ของสัญญาณ

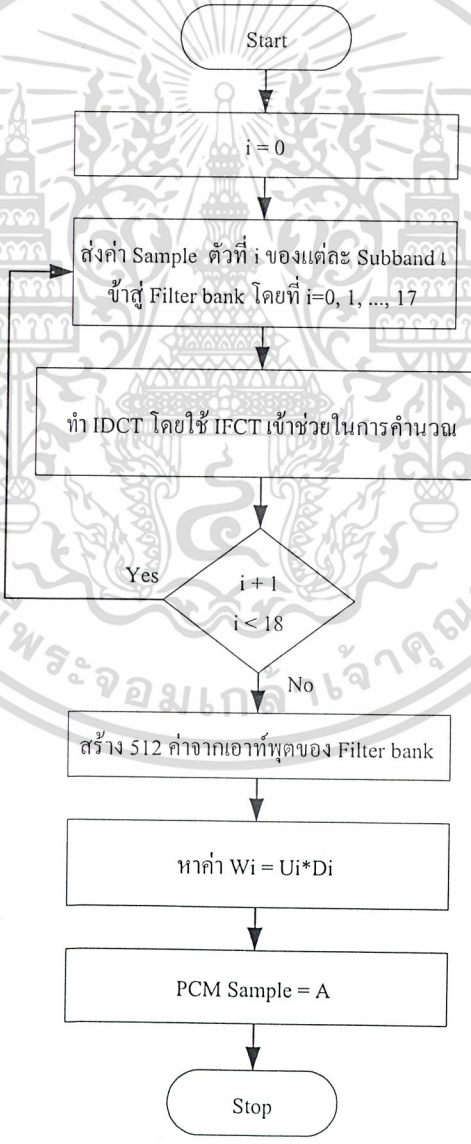
จะได้ว่า

$$resulti = z_i + s_i \qquad ;for\ i=0\ to\ 17 \tag{3.13}$$

$$s_i = z_{i+18} \qquad ;for\ i=0\ to\ 17 \tag{3.14}$$

3.3.11 การรวมสัญญาณจากแต่ละช่วงตัวกรอง (Synthesis Filter Banks)

หัวข้อนี้เป็นขั้นตอนสุดท้ายของกระบวนการถอดรหัส ได้ผลลัพธ์เป็นสัญญาณแบบพีซีเอ็ม (PCM Sample) สามารถอธิบายกระบวนการโดยแผนผังการทำงานของโปรแกรมได้ดังรูปที่ 3.23



รูปที่ 3.22 ผังงานของโปรแกรมการรวมสัญญาณจากแต่ละย่านความถี่ย่อย

หมายเหตุ

$$A = S_i = \sum_{i=0}^{15} W_{j+32i} \tag{3.15}$$

1) การส่งค่าอินพุตเข้าไปยังโพลีเฟส ฟิลเตอร์แบงค์ (Polyphase Filterbanks)

ค่าที่ส่งให้กับโพลีเฟส ฟิลเตอร์แบงค์ คือค่าของข้อมูลสุ่มตัวอย่างที่ i ของทุกๆ ย่านความถี่ย่อย โดยโพลีเฟส ฟิลเตอร์แบงค์ จะรับค่าทีละ 1 ค่าจากแต่ละย่านความถี่ย่อยที่ (ที่ 32 ย่านความถี่ย่อย) ก็คือโพลีเฟส ฟิลเตอร์แบงค์รับค่าและประมวลทีละ 32 ค่า

ในแต่ละย่านความถี่ย่อยมีข้อมูลสุ่มตัวอย่างอยู่ 18 ค่า เพราะฉะนั้น โพลีเฟส ฟิลเตอร์แบงค์จะประมวลผล 18 ครั้งต่อแชนแนลต่อเฟรม

2) IDCT (Inverse Discrete Cosine Transform)

นำ 32 ค่าอินพุตเข้ากระบวนการทำ IDCT โดยนำ 32 ค่าอินพุตแทนลงในสมการที่ 3.16

$$x_i = \sum_{k=0}^{31} X_k \varepsilon_k \cos\left(\frac{\pi k(2i+1)}{32}\right) \tag{3.16}$$

โดยที่ x_i เป็นเอาต์พุต

X_k เป็นอินพุตจากย่านความถี่ย่อย

$$\varepsilon_k = \frac{1}{\sqrt{2}}$$

$$\varepsilon_k = 1$$

3) การสร้างค่า 512 ค่าจาก x_i

นำค่าเอาต์พุต x_i จากหัวข้อ 3.14.2 มาสร้างค่า 512 ค่าและกำหนดให้ 512 ค่านี้แทนด้วย U_i โดยที่ i = 0, 1, 2,..., 511 ดังโปรแกรมด้านล่างนี้

```
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]
```

4) การคูณค่าสัมประสิทธิ์ของการสังเคราะห์วินโดว์

นำค่า U_i จากหัวข้อ 3.10.3 มาคูณด้วย D_i ซึ่งค่า D_i นี้ถูกกำหนดอยู่ในมาตรฐานของ MPEG ดังสมการที่ 3.17 จะได้ค่า W_i ออกมา

$$W_i = U_i D_i \tag{3.17}$$

5) การคำนวณค่าของข้อมูลสุ่มตัวอย่าง PCM

การคำนวณค่าของข้อมูลสุ่มตัวอย่าง PCM 32 ค่านี้ สามารถหาได้จากสมการที่ 3.18

$$S_j = \sum_{i=0}^{15} W_{j+32i} \tag{3.18}$$

ค่า Si ที่ได้ เป็นค่าของการสุ่มตัวอย่างแบบ PCM (PCM Sample) เป็นข้อมูลแบบเชิงเลข (Digital) จากนั้นค่า 32 ค่านี้จะถูกส่งไปยังอุปกรณ์เสียง (Audio Device) ของตัวถอดรหัส ค่าของการสุ่มตัวอย่างแบบ PCM จะถูกส่งเข้าสู่อุปกรณ์อย่างต่อเนื่อง และเกิดเสียงต่อเนื่องขึ้นที่อุปกรณ์เสียง

3.3.12 ไวยากรณ์ของข้อมูล MP3

รูปแบบข้อมูล MP3 (MPEG Layer3 File Format) อธิบายโดยใช้ไวยากรณ์ (Syntax) ของภาษาซีเพื่อให้เห็นภาพพจน์ โดยใช้ตัวอักษรเข้มแสดงชื่อของข้อมูลในแต่ละส่วน เลขด้านท้ายของแต่ละบรรทัดแสดงความยาวบิตของข้อมูลนั้น

1) ส่วนข้อมูลเสียง (Audio Data)

```
audio_data()
{
        main_data_begin                                    8       uimsbf
        if (mode==single_channel)
                private_bits                               1       bslbf
        else
                private_bits                               2       bslbf
        for (ch=0; ch<nch; ch++) {
                part2_3_length[ch]                         12      uimsbf
                big_values[ch]                             9       uimsbf
                global_gain[ch]                            8       uimsbf
                scalefac_compress[ch]                      9       bslbf
                window_switching_flag[ch]                  1       bslbf
                if (window_switching_flag[ch]) {
                        block_type[ch]                     2       bslbf
                        mixed_block_flag[ch]               1       uimsbf
                        for (region=0; region<2; region++)
                                table_select[ch][region]   5       bslbf
                        for (window=0; window<3; window++)
                                subblock_gain[ch][window]  3       uimsbf
                }
                else {
                        for (region=0; region<3; region++)
                                table_select[ch][region]   5       bslbf
```


รูปที่ 3.23 ไวยกรณ์ของ MP3 ในส่วนข้อมูลเสียง

```
                    region0_count[ch]                    4       bslbf
                    region1_count[ch]                    3       bslbf
                }
                scalefac_scale[ch]                       1       bslbf
                count1table_select[ch]                   1       bslbf
        }
        main_data ()
}
```

รูปที่ 3.23 (ต่อ) ไวยกรณ์ของ MP3 ในส่วนข้อมูลเสียง

2) ส่วนข้อมูลหลัก (Main Data)

```
Main_data()
{
        for (ch=0; ch<nch; ch++) {
                if ((window_switching_flag[ch]==1) &&

                        block_type[ch]==2)) {
                        if (mixed_block_flag[ch]) {
                                for (sfb=0; sfb<8; sfb++)
                                        scalefac_l[ch][sfb]          0..4    uimsbf
                                for (sfb=3; sfb<12; sfb++)
                                        for (window=0; window<3;
window++)
                                                Scalefac_s[ch][sfb]  0..5    uimsbf
[window]        }
```


รูปที่ 3.24 ไวยกรณ์ของ MP3 ในส่วนข้อมูลหลัก

```
                    else {
                        for (sfb=0; sfb<12; sfb++)
                            for (window=0; window<3;
window++)
                                scalefac_s[ch][sfb]      0..5      uimsbf
[window]
                    }
                }
                else {
                    for (sfb=0;sfb<21;sfb++)
                        scalefac_l[ch][sfb]              0..5      uimsbf
                }
                Huffmancodebits()
            }
            for (b=0; b<no_of_ancillary_bits; b++)
                Ancillary_bit                            1         bslbf
}
```

รูปที่ 3.24 (ต่อ) ไวยกรณ์ของ MP3 ในส่วนข้อมูลหลัก

3) ส่วนการเข้ารหัสแบบฮัฟแมน

```
Huffmancodebits() {
    For (i=0; i<big_values*2; i+=2) {
        hcod[|x|][|y|]                                   0..19     bslbf
        if (|x|==15 && linbits>0)
            linbitsx                                     1..13     uimsbf
        if (x != 0)
```

รูปที่ 3.25 ไวยกรณ์ของ MP3 การเข้ารหัสฮัฟแมน

```
                    signx                                   1       bslbf
                if (|y|==15 && linbits>0)
                    linbitsy                                1..13   uimsbf
                if (y != 0)
                    signy                                   1       bslbf
                is[l] = x
                is[l+1] = y
        }
        for (; l<big_values*2+count1*4; l+=4) {
                hcod[|v|][|w|][|x|][|y|]                    1..6    bslbf
                if (v!=0)
                    signv                                   1       bslbf
                if (w!=0)
                    signw                                   1       bslbf
                if (x!=0)
                    signx                                   1       bslbf
                if (y!=0)
                    signy                                   1       bslbf
                is[l] = v
                is[l+1] = w
                is[l+2] = x
                is[l+3] = y
        }
        for (; l<576; l++)
                is[l] = 0
}
```

รูปที่ 3.25 (ต่อ) ไวยกรณ์ของ MP3 การเข้ารหัสฮัฟแมน

4) ความหมายของคำในไวยากรณ์ของข้อมูล MP3

4.1) ลำดับข้อมูลเสียงทั่วไป

Frame ส่วนของลำดับข้อมูลที่สามารถถอดรหัสได้ บรรจุข้อมูล 1152 สัญญาณสุ่มความถี่

4.2) เฟรมของสัญญาณเสียง (Audio Frame)

Header ส่วนของลำดับข้อมูล (Bitstream) ที่บรรจุข้อมูลเกี่ยวกับ การให้ความสัมพันธ์ (Synchronization) และข้อมูลทั่วไป

Error_check ส่วนของลำดับข้อมูลที่บรรจุข้อมูลเกี่ยวกับการหาข้อผิดพลาด

Audio_data ส่วนของลำดับข้อมูลที่บรรจุข้อมูลเกี่ยวกับข้อมูลเสียงที่สุ่มตัวอย่าง

Ancillary_data ส่วนของลำดับข้อมูลที่บรรจุข้อมูลเพิ่ม

4.3) ส่วนหัวของข้อมูล (Header)

32 บิตแรกของข้อมล บรรจุข้อมูลที่บอกถึงลักษณะทั่วๆ ไปทั้งหมดของการเข้ารหัส

Syncword ส่วนหัวของลำดับข้อมูล MPEG เท่ากับ “1111 1111 1111”

ID บิตแสดงถึงมาตรฐานการเข้ารหัส “1” หมายถึงการเข้ารหัสมาตราฐาน ISO 11172-3 “0” ไม่ใช่มาตราฐาน ISO

Layer แสดงเลเยอร์ที่กำลังถอดรหัสอยู่ ดังตารางที่ 3.7

ตารางที่ 3.7 ความหมายของรหัสข้อมูลในเลเยอร์

เลเยอร์ (Layer)	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

Sampling_frequency บอกความถี่ในการสุ่มตัวอย่าง (Sampling)

Padding_bit เท่ากับ "1" แสดงว่าเฟรมของข้อมูลนั้น บรรจุสล็อต (Slot) ใน Layer3 = 1 Byte เพิ่มเติมมา ถ้าเท่ากับ "0" แสดงว่าไม่ได้บรรจุ การบรรจุสล็อต เพื่อปรับอัตราส่วนระหว่างอัตราส่วนการบีบอัดกับความถี่ในการสุมตัวอย่างให้ลงตัว ในกรณีการสุ่มตัวอย่าง 44.1 กิโลเฮิร์ต ไม่มี Padding คือเท่ากับ "0" เสมอ

Private ไม่ใช่ในการเข้ารหัสมาตราฐาน ISO/IEC 11172-3

Copy Right "1" ป้องกันการ Copy

"0" ไม่ป้องกันการ Copy

Original/copy "0" ข้อมูลนั้นถูก Copy มา

"1" ข้อมูลนั้นเป็นต้นฉบับ

Mode บอกเกี่ยวกับรูปแบบของข้อมูลว่าเป็นหนึ่งช่องเสียง (Single Channel), สองช่องเสียง (Double Channel), สเตอริโอ (Stereo), จอย – สเตอริโอ (Joint to Stereo) ดังตารางที่ 3.8

ตารางที่ 3.8 ความหมายของรหัสข้อมูลใน Mode

โหมด (Mode)	ความหมาย
"00"	สเตอริโอ
"01"	จอย – สเตอริโอ
"10"	สองช่องเสียง
"11"	หนึ่งช่องเสียง

Mode_extention ใช้บอกชนิดของจอย – สเตอริโอ (Joint to Stereo Mode) ว่ามี MS – Stereo, Intensity – Stereo หรือไม่อย่างไร ดังแสดงในตารางที่ 3.9

ตารางที่ 3.9 ความหมายของรหัสข้อมูลใน Mode_extention

Mode_extention	Intensity	MS- Stereo
“00”	Off	Off
“01”	On	Off
“10”	Off	On
“11”	On	On

Emphasis แสดงถึงชนิดของเอ็มพาไซส์ (Emphasis) ที่ใช้งาน แสดงในตารางที่ 3.10

ตารางที่ 3.10 ความหมายของรหัสข้อมูลใน Emphasis

Emphasis	ความหมาย
“00”	ไม่มีใช้
“01”	50/15 ไมโครวินาที
“10”	สงวน
“11”	CCITT .17

บทที่ 4

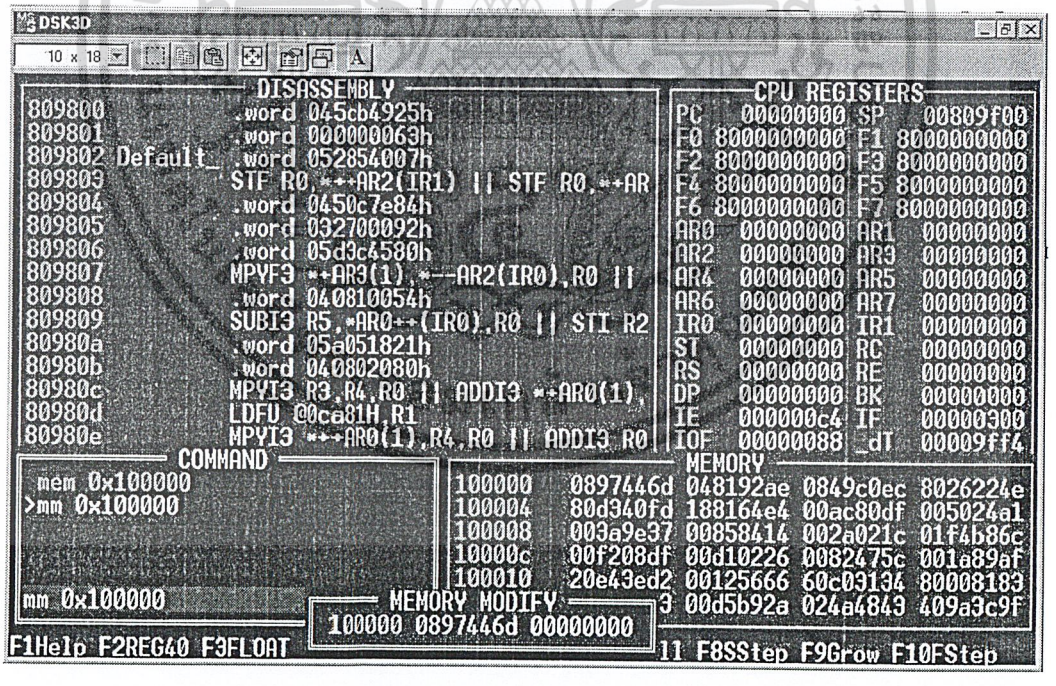
การทดลอง และผลการทดลอง

โครงงานเรื่องตัวถอดรหัสสัญญาณเสียง MP3 ด้วย DSK TMS320C31 จากบทที่ 3 ได้ทำการออกแบบการทำงานของระบบ โดยโครงงานนี้สามารถแบ่งการทดลองของระบบออกเป็น 2 ส่วนใหญ่ คือ การทดลองทางด้านฮาร์ดแวร์และการทดลองทางด้านซอฟต์แวร์

4.1 การทดลองส่วนของฮาร์ดแวร์

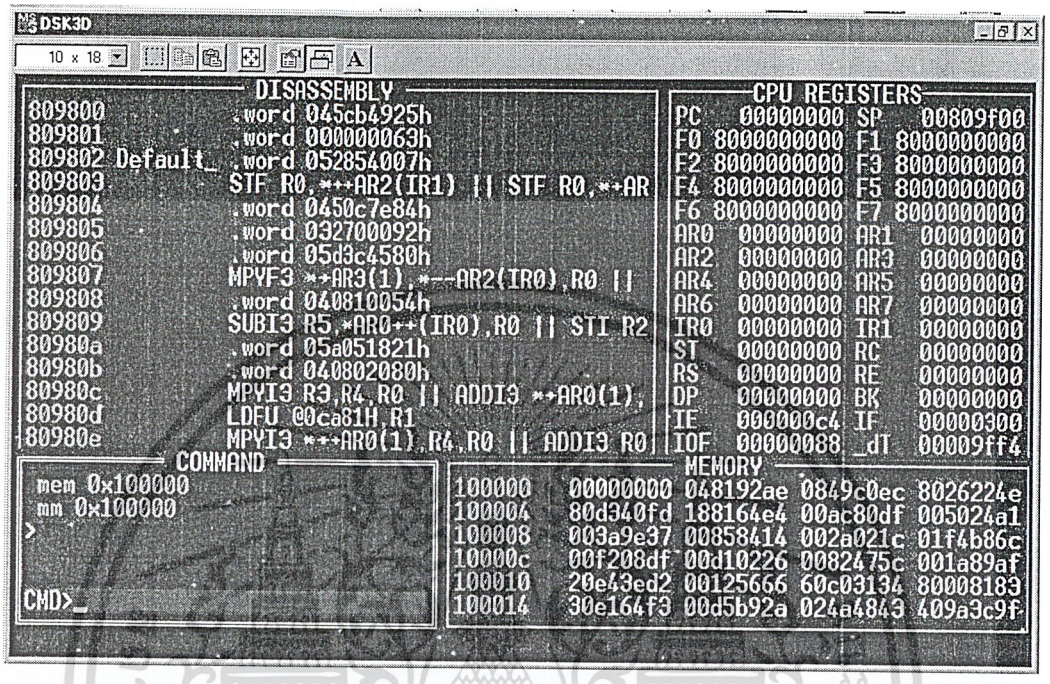
4.1.1 ทดสอบหน่วยความจำภายนอก (RAM) โดยใช้โปรแกรม DSK3D

การทดสอบหน่วยความจำภายนอก (RAM) โดยใช้โปรแกรม DSK3D เพื่อตรวจสอบการอ่านหรือเขียนข้อมูลในหน่วยความจำทำการเขียนข้อมูลค่า 00000000H ที่ตำแหน่งหน่วยความจำ 100000H โดยใช้การทดสอบการเขียนข้อมูลลงในหน่วยความจำ (RAM) แสดงในรูปที่ 4.1



รูปที่ 4.1 การทดสอบหน่วยความจำภายนอกค่า 00000000 ที่ตำแหน่ง 0x100000

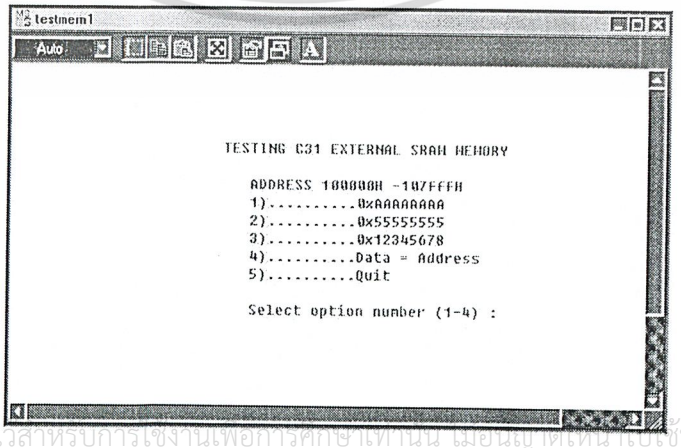
จากผลการทดลองแสดงในรูปที่ 4.2 ที่ตำแหน่งหน่วยความจำ 100000H มีการเขียนข้อมูล 00000000H ลงไปในตำแหน่งหน่วยความจำได้จริง



รูปที่ 4.2 ผลการทดสอบหน่วยความจำภายนอกค่า 00000000 ที่ตำแหน่ง 0x100000

4.1.2 ทดสอบหน่วยความจำภายนอก (RAM) โดยการใช้โปรแกรม Testmem1

การทดสอบหน่วยความจำภายนอก (RAM) โดยการใช้โปรแกรม Testmem1 ซึ่งแตกต่างจากดังวิธีการข้างต้นคือสามารถเขียนค่าลงในตำแหน่งหน่วยความจำได้มากกว่าและสามารถเลือกค่าที่ต้องการเขียนได้ ในการทดสอบหน่วยความจำ (RAM) โดยให้ตำแหน่งที่ 100000H – 107FFFH เป็น 0xAAAAAAAA โดยโปรแกรมจะอยู่ในภาคผนวก ค และผลการทดลองตามรูปที่ 4.3

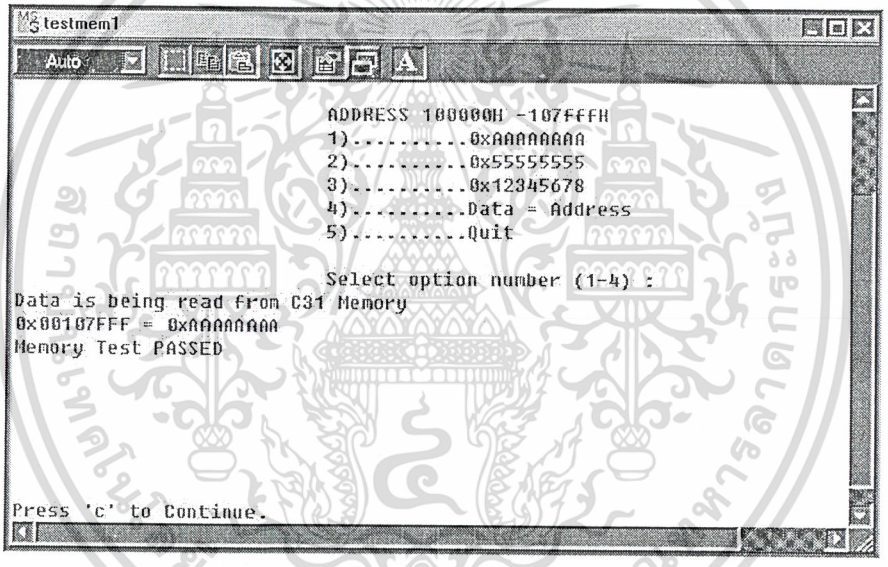


รูปที่ 4.3 หน้าจอของโปรแกรมทดสอบหน่วยความจำภายนอก (RAM)

จากรูปที่ 4.3 เป็นการแสดงการเขียนข้อมูลค่า AAAAAAAH ลงในตำแหน่งหน่วยความจำภายนอก ซึ่งจะมีการเลือกค่าที่เขียนลงในหน่วยความจำได้โดยการกดปุ่มดังนี้

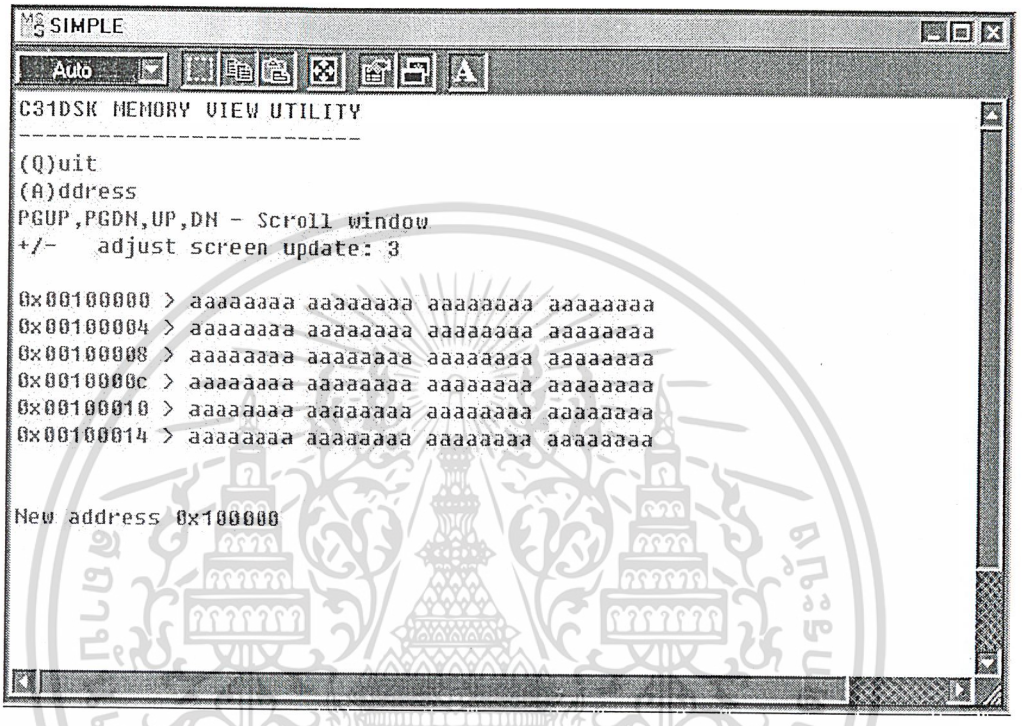
1) เลือก 1 จะเขียนค่า AAAAAAAH ลงในตำแหน่งหน่วยความจำภายนอก
2) เลือก 2 จะเขียนค่า 55555555H ลงในตำแหน่งหน่วยความจำภายนอก
3) เลือก 3 จะเขียนค่า 12345678H ลงในตำแหน่งหน่วยความจำภายนอก
4) เลือก 4 จะเป็นการเปรียบเทียบค่าระหว่างตำแหน่งข้อมูลและแอดเดรส
5) เลือก 5 ออกจากโปรแกรม

จากการทดลองโปรแกรมได้เขียนข้อมูลลงไปในตำแหน่งหน่วยความจำได้จริง ดังแสดงในรูปที่ 4.4



รูปที่ 4.4 การเขียนข้อมูลค่า AAAAAAAAH ลงตำแหน่งหน่วยความจำภายนอก (RAM)

จากผลการทดลองเพื่อแสดงถึงการเขียนข้อมูลลงในตำแหน่งหน่วยความจำภายนอกจริงได้ทำการเปรียบเทียบข้อมูลที่เขียนกับตำแหน่งหน่วยความจำ โดยพิมพ์ค่าตำแหน่งหน่วยความจำที่ต้องการตรวจสอบ คือ ตำแหน่ง 0x100000 ดังแสดงในรูปที่ 4.5



รูปที่ 4.5 ผลการเขียนข้อมูลค่า AAAAAAAA ลงตำแหน่งหน่วยความจำภายนอก (RAM)

4.2 การทดลองด้านซอฟต์แวร์

4.2.1 การสร้างไฟล์ DSKLIB.H ให้เป็นไฟล์ .IDE

การใช้โปรแกรมย่อยในไลบราลีของ DSK สำหรับการติดต่อระหว่าง PC กับ TMS320C31 เช่น Getmem ที่ใช้สำหรับอ่านข้อมูลจาก TMS320C31 ทางพอร์ตขนานไปเขียนข้อมูลที่ PC ถูกประกาศโดย DSK.H โดยการใช้คุณสมบัติของ TARGET.CPP, DRIVER.CPP, OBJECT.CPP ที่สนับสนุนฟังก์ชั่นการติดต่อสำหรับการเขียน/อ่านบล็อกของ C31

สำหรับฟังก์ชั่นที่ใช้สำหรับการติดต่อระหว่าง PC กับ TMS320C31 บนตัวบอร์ด DSK ด้วยฟังก์ชั่น Putmem และ Getmem เราสามารถที่จะใช้การรวมเครื่องมือในการพัฒนาระหว่าง TMS320 floating-point DSP assembly language tools กับ Borland's C/C++ complier ทำการสร้างไฟล์ DSKLIB.H และ DSKLIB.LIB โดยจะต้องทำการสร้างไฟล์ทั้งสองก่อนเป็นอันดับแรก โดยมีขั้นตอนการสร้างไฟล์ทั้งสองดังนี้ดังนี้

1) เปิดไฟล์ DSKLIB.H แสดงในรูปที่ 4.6

2) เลือก File > New > Project แสดงในรูปที่ 4.7

3) พิมพ์ไฟล์ที่ต้องการลงใน Project Path and Name แสดงในรูปที่ 4.8

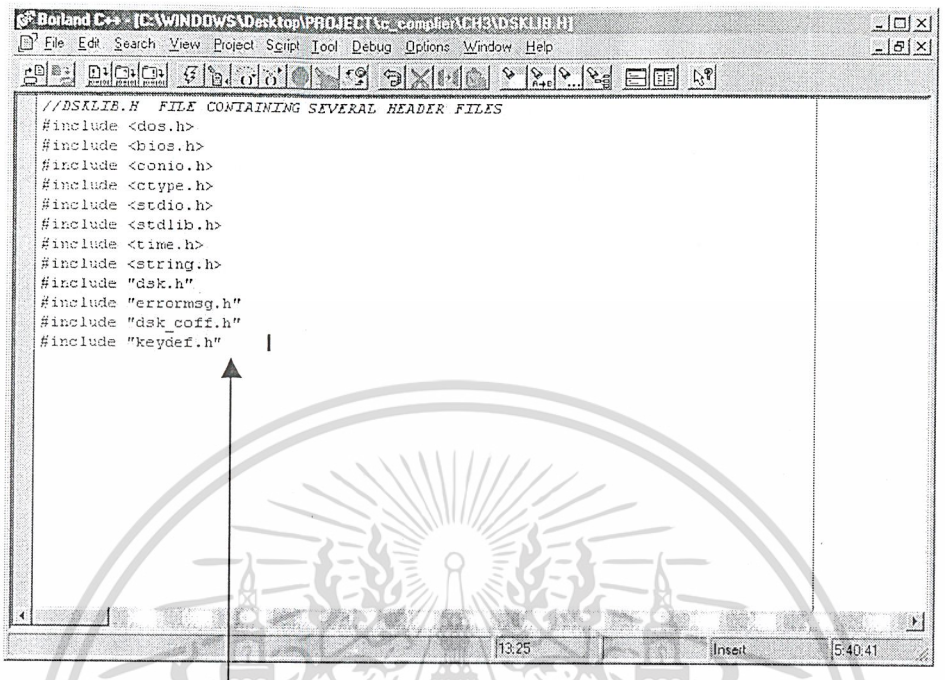
4) เลือก Static Library [for.exe] [.lib] เลือก DOS (Standard) และ Large ในส่วนของ Platform และ Target Model

5) คลิกปุ่ม OK

6) คลิกขวา เลือก Add node แสดงในรูปที่ 4.9

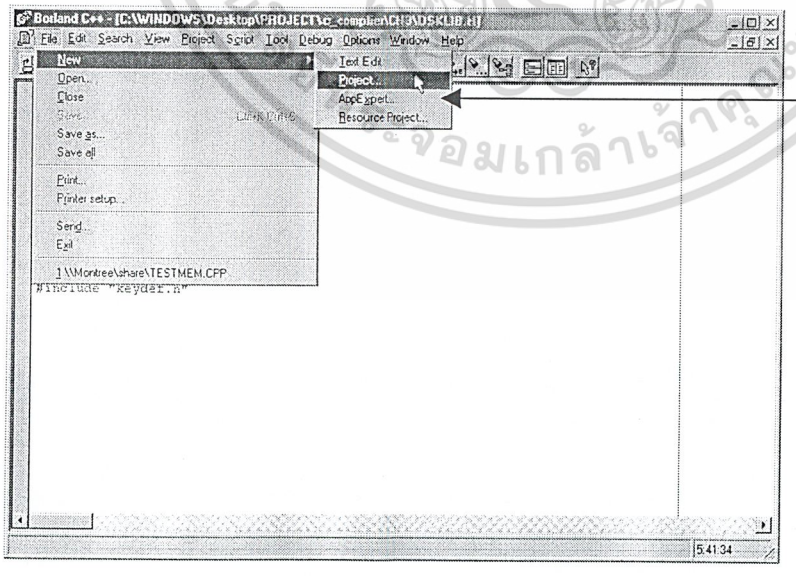
7) เลือกไฟล์ driver.cpp, tmsfloat.cpp, textwin.cpp, target.cpp, symbol.cpp, rand386.cpp, object.cpp, errormsg.cpp, dsk_coff.cpp. คลิกปุ่ม Open แสดงในรูปที่ 4.10

8) เลือก Project > Build all แสดงในรูปที่ 4.12



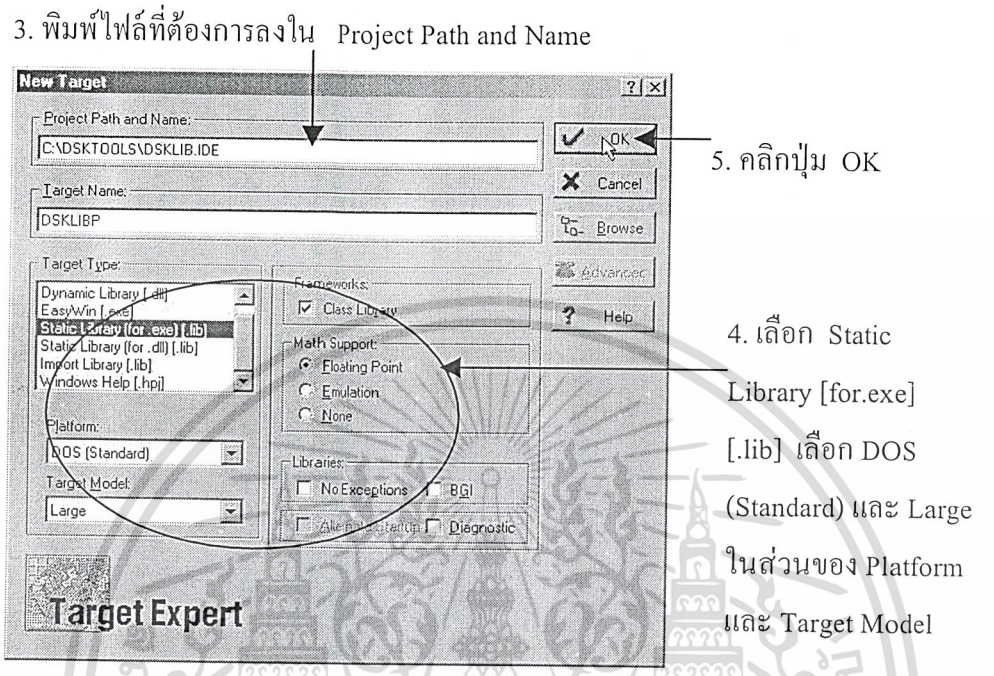
1. เปิดไฟล์ DSKLIB.H

รูปที่ 4.6 การเปิดไฟล์ DSKLIB.H

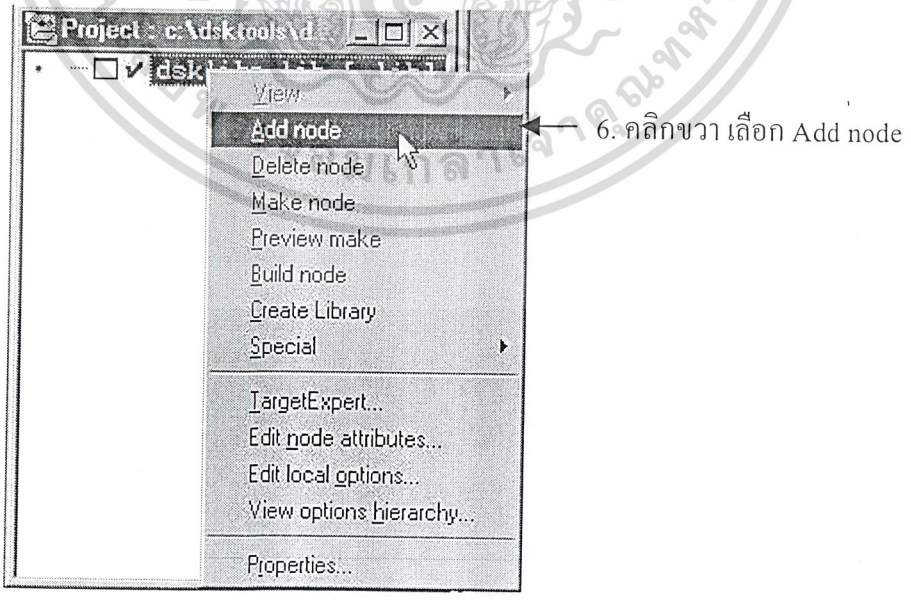


2. เลือก File > New > Project

รูปที่ 4.7 การสร้าง New Target

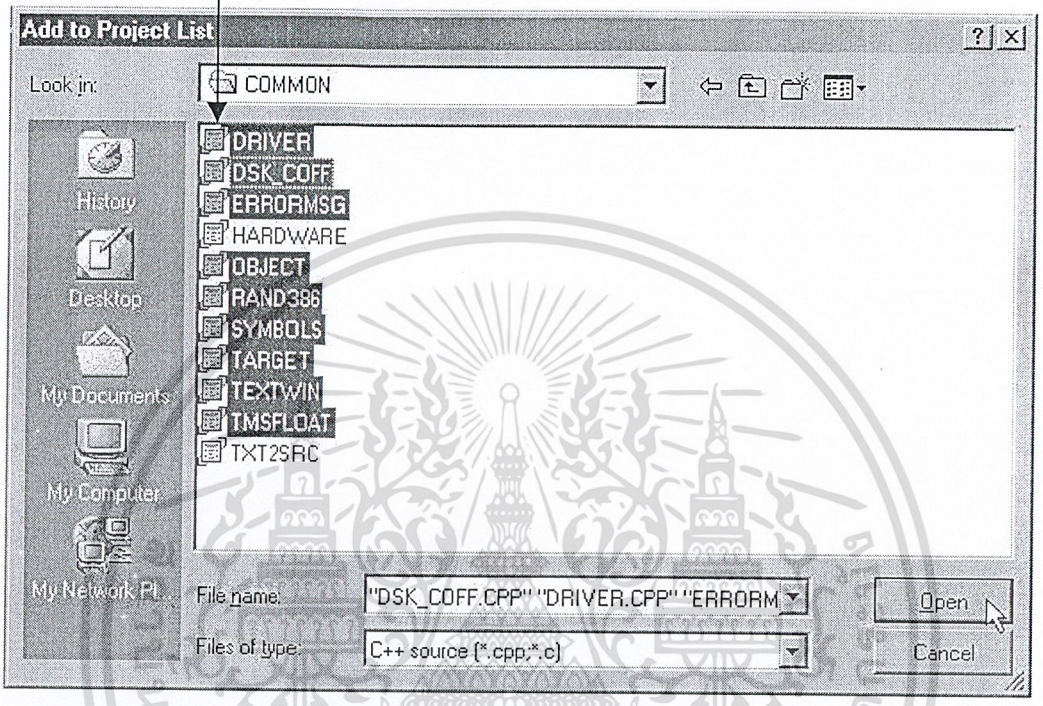


รูปที่ 4.8 การ Set ค่าตำแหน่งต่างๆ ของ New Target

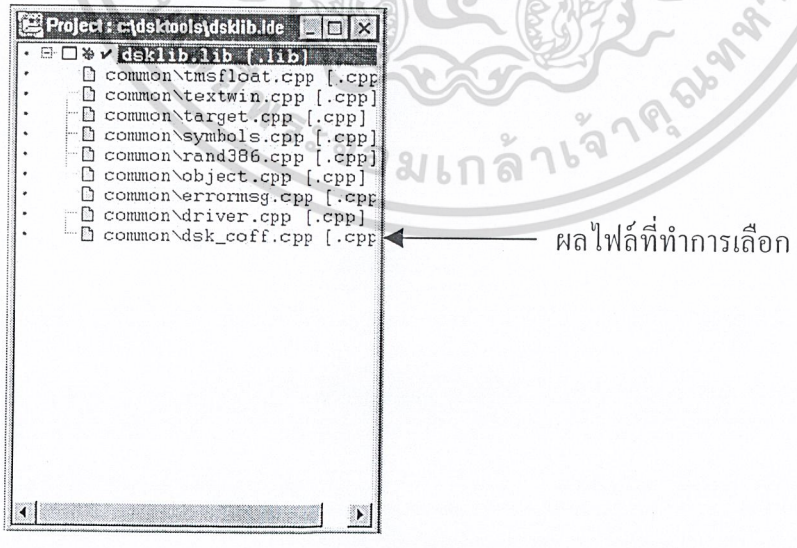


รูปที่ 4.9 การเปิด Add to Project List

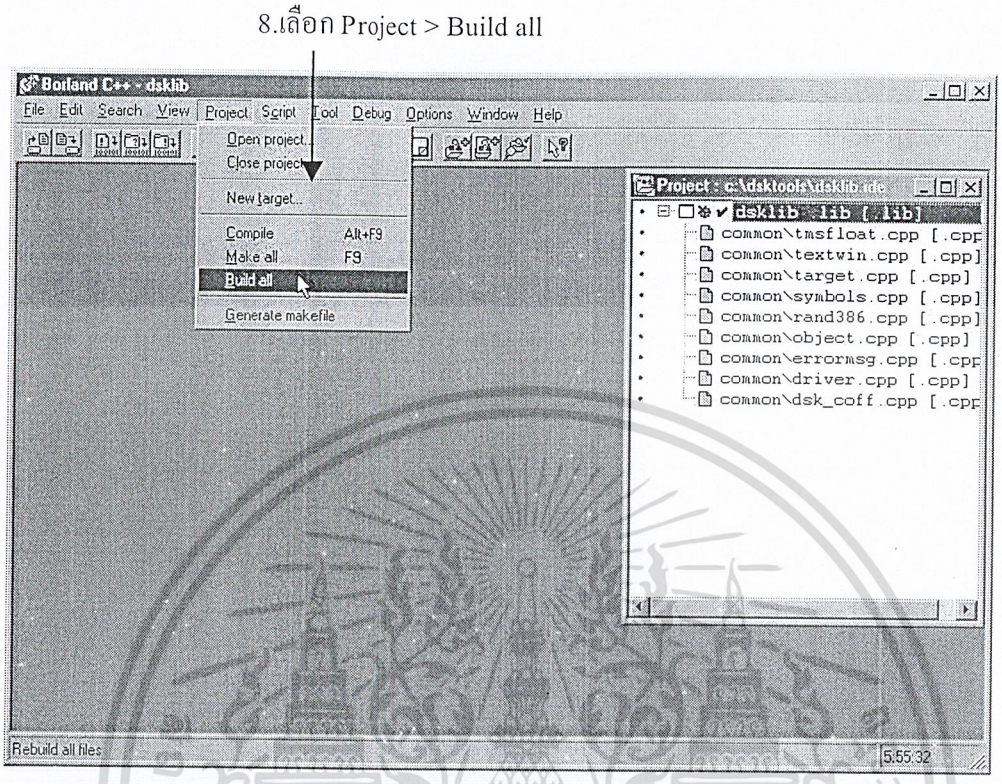
7. เลือกไฟล์ driver.cpp, tmsfloat.cpp, textwin.cpp, target.cpp, symbol.cpp, rand386.cpp, object.cpp, errormsg.cpp, dsk_coff.cpp. คลิกปุ่ม Open



รูปที่ 4.10 การเลือก Add to Project List



รูปที่ 4.11 ผลการ Add to Project List



รูปที่ 4.12 การสร้างไฟล์ทั้งหมดให้เป็นนามสกุล .IDE

4.2.2 การทดลองโปรแกรมถอดรหัสส่วนหัวสัญญาณเสียง MP3

การทดลองโปรแกรมถอดรหัสสัญญาณเสียง MP3 จะต้องทราบโครงสร้างของไฟล์ MP3 ซึ่งข้อมูลจะถูกจัดให้อยู่ในรูปของเฟรมข้อมูล ดังที่กล่าวในบทที่ 3 มีส่วนประกอบทั้งหมด 4 ส่วน คือ

1) หัวข้อมูล (Header) เป็นข้อมูลขนาด 32 บิต แสดงลักษณะทั่วไปของไฟล์นั้นๆ

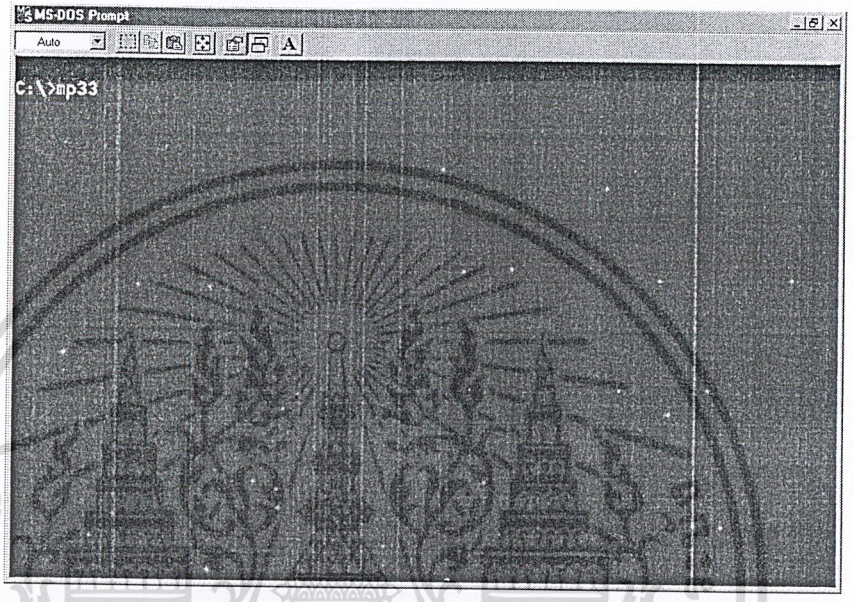
2) ส่วนตรวจสอบความผิดพลาด (CRC) เป็นข้อมูลขนาด 16 บิต ใช้ตรวจสอบความถูกต้องข้อมูลภายในเฟรม

3) ข้อมูลข้างเคียง (Side Information) มีขนาด 17 หรือ 32 ไบต์ (17 ไบต์สำหรับระบบโมโน 32 ไบต์ สำหรับระบบอื่นๆ) เป็นส่วนที่เก็บองค์ประกอบที่ใช้ในการถอดรหัส

4) ข้อมูลหลัก (Main Data) มีความยาวขึ้นอยู่กับอัตราการส่งข้อมูล (Baud rate) และอัตราการสุ่มข้อมูลในการแปลงกลับเป็นสัญญาณแอนาล็อก (Sampling Frequency)

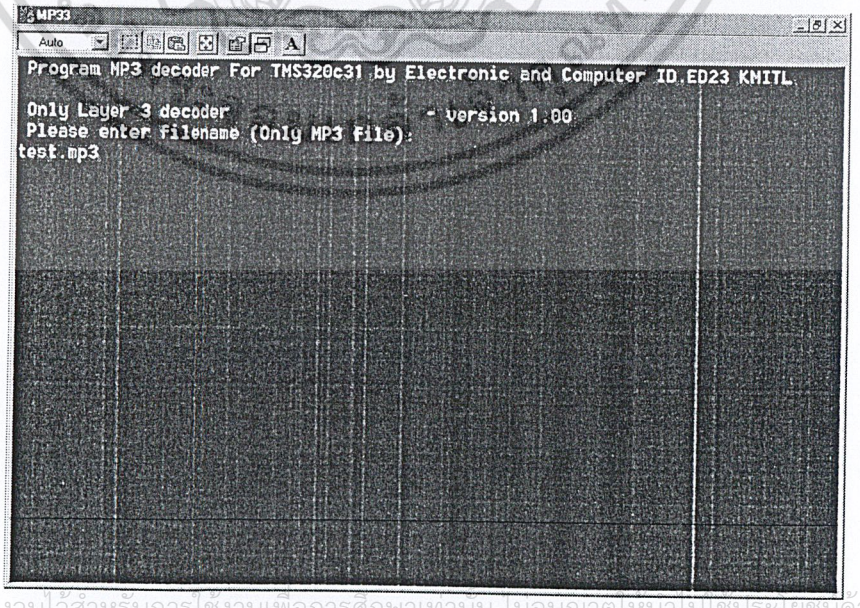
ในการทดลองทำการทดลองโดยใช้โปรแกรมถอดรหัสไฟล์ MP3 ซึ่งในส่วนของโปรแกรมได้แสดงไว้ในภาคผนวก ค

1) เปิดโปรแกรม mp33.exe เพื่อแสดงการถอดรหัสในส่วนหัวข้อข้อมูลเสียง MP3 ดังแสดงในรูปที่ 4.13



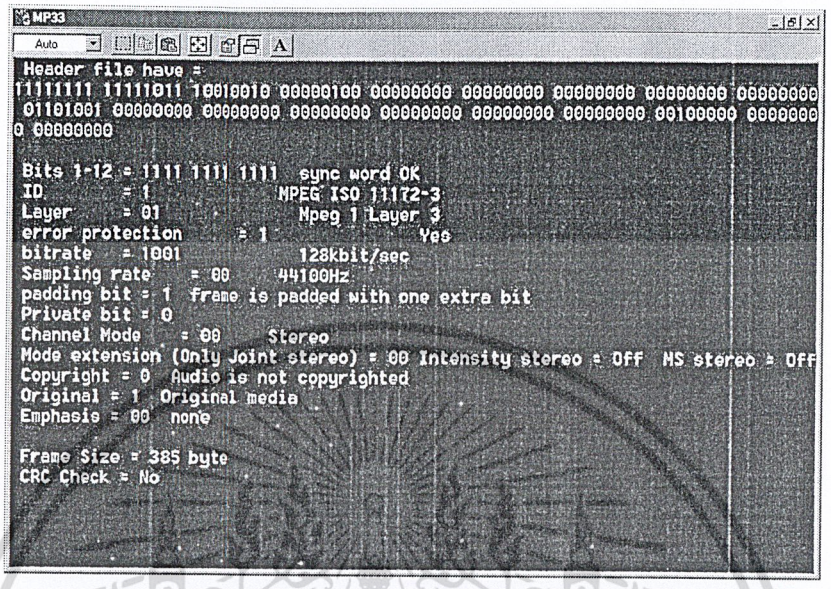
รูปที่ 4.13 การทดสอบถอดรหัสไฟล์ MP3

2) เลือกไฟล์ test.mp3 เพื่อทำการถอดรหัสสัญญาณ ดังแสดงในรูปที่ 4.14



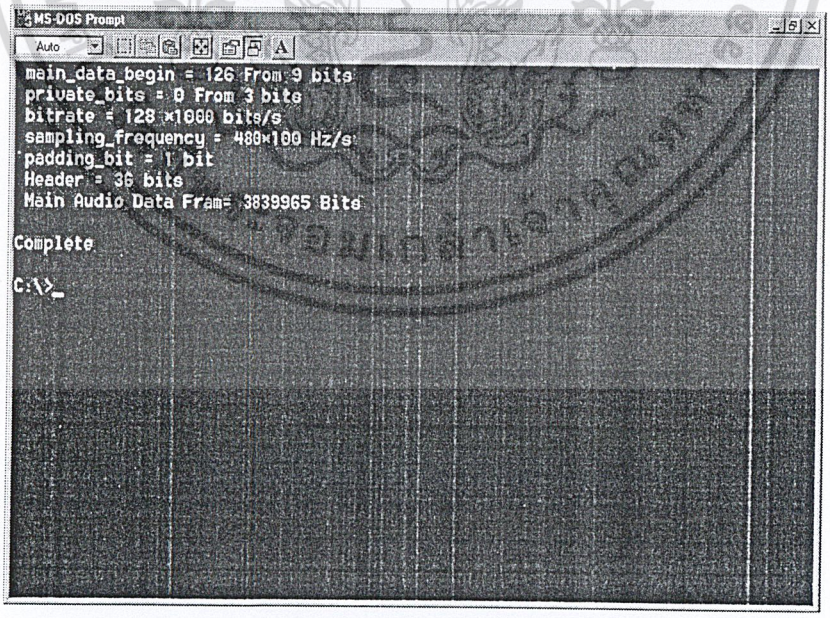
รูปที่ 4.14 การป้อนไฟล์ test.mp3 เพื่อทำการถอดรหัส

3) ผลการถอดรหัสในส่วนหัวของ test.mp3 สัญญาณเสียง ดังแสดงในรูปที่ 4.15



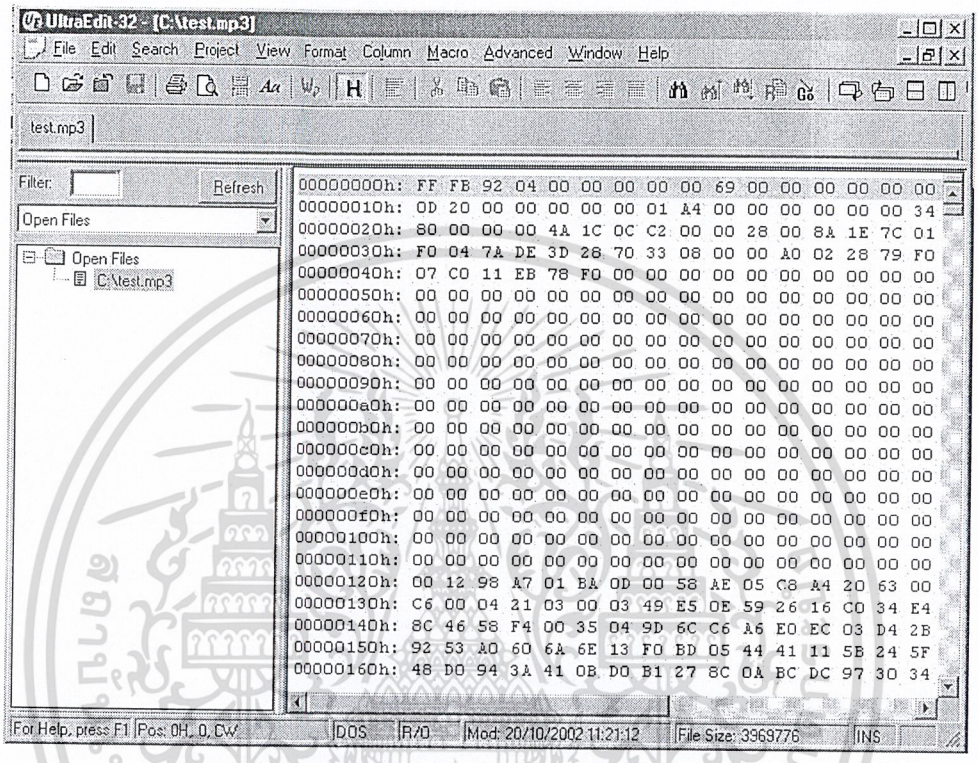
รูปที่ 4.15 ผลการถอดรหัสของไฟล์ส่วนหัวของ test.mp3

4) ผลการถอดรหัสข้อมูลข้างเคียง (Side Information) ของ Test.mp3 ดังแสดงในรูปที่ 4.16



รูปที่ 4.16 ผลการแสดงข้อมูลข้างเคียง (Side Information) ของ Test.mp3

5) เมื่อทำการถอดรหัสในส่วนของโปรแกรมที่เขียนขึ้นมา เพื่อแสดงการทำงานของโปรแกรมว่าได้ทำการถอดรหัสได้จริง โดยใช้โปรแกรม UltraEdit-32 มาใช้ในการเปรียบเทียบดังที่แสดงในรูปที่ 4.17

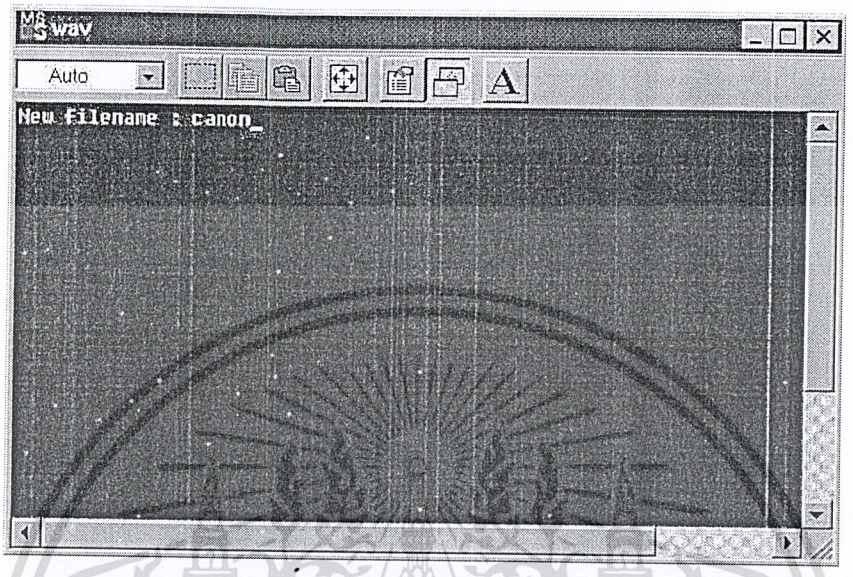


รูปที่ 4.17 ผลการถอดไฟล์ Test.mp3 เป็น Hex Code

4.2.3 การทดลองโปรแกรมถอดรหัสสัญญาณเสียง

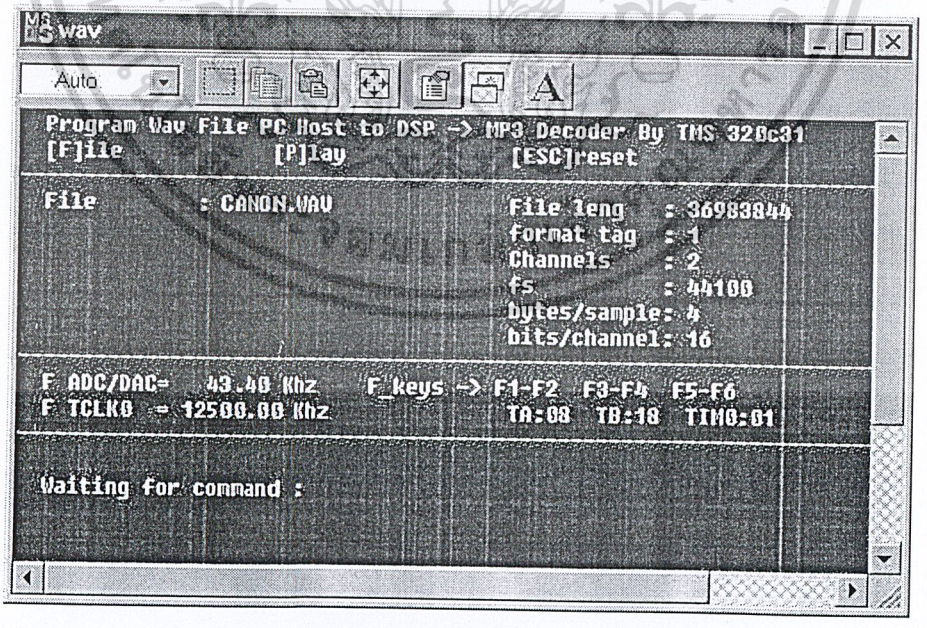
ในส่วนของโปรแกรมถอดรหัสสัญญาณเสียงนี้ทำการถอดรหัสสัญญาณเสียงนามสกุลเวฟ (Wave) เพื่อแสดงการติดต่อบอร์ด DSK TMS320C31 กับเครื่องไมโครคอมพิวเตอร์ผ่านทางพอร์ตขนาน โดยเปิดโปรแกรม Wave.exe ในส่วนของโปรแกรมได้แสดงไว้ในภาคผนวก ค ซึ่งมีขั้นตอนการทดสอบโปรแกรมดังนี้

1) เปิดโปรแกรม Wave.exe และเลือกไฟล์ Cannon.wave เพื่อที่จะทำการถอดรหัส ดังที่แสดงในรูปที่ 4.18



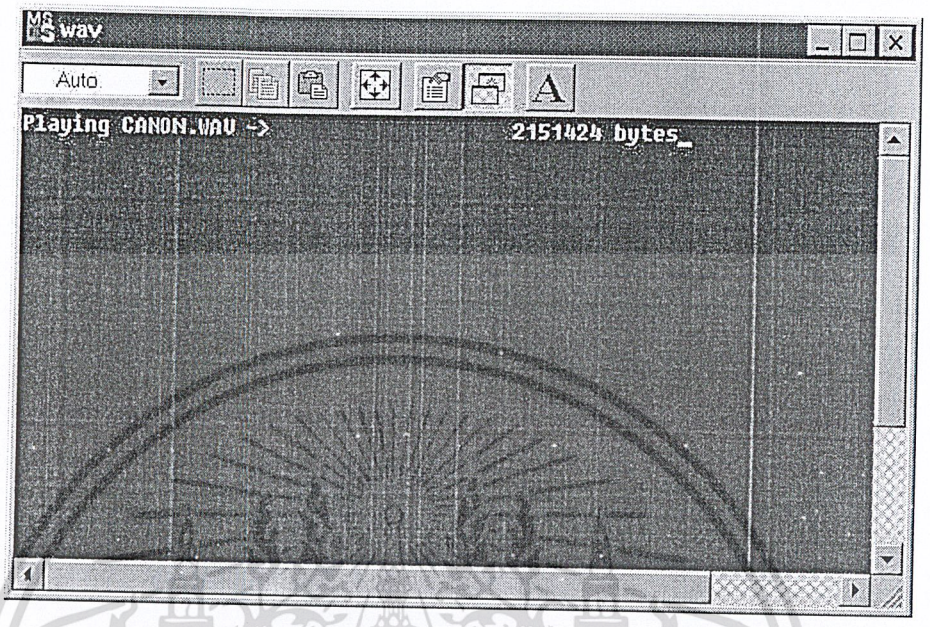
รูปที่ 4.18 การป้อนไฟล์ Cannon.wave เพื่อที่จะทำการรันโปรแกรม

2) เลือกความถี่ที่ใช้ในการเล่นโดยกดปุ่ม F1 และ F2 แสดงในรูปที่ 4.19



รูปที่ 4.19 เลือกความถี่ที่จะทำการรันโปรแกรมโดยกดปุ่ม F1 และ F2

3) โปรแกรมจะทำการถอดไฟล์ Cannon.wave ดังแสดงในรูปที่ 4.20



รูปที่ 4.20 ผลการทำงานของโปรแกรม

ในส่วนการใช้งานโปรแกรมสามารถควบคุมโดยคีย์บอร์ด โดยแต่ละปุ่มจะมีการใช้งานดังนี้

1) กดปุ่ม F (File) เพื่อทำการป้อนไฟล์ใหม่ที่ต้องการถอดรหัส

2) กดปุ่ม P (Play) เพื่อให้โปรแกรมทำงานต่อไป

3) กดปุ่ม ESC (Reset) เพื่อทำการรีเซ็ตโปรแกรม

4) กดปุ่ม F1, F2 เพื่อทำการปรับความถี่ที่ต้องการ

จากผลการทดลอง เมื่อทำการปรับค่าความถี่โดยกดปุ่ม F1, F2 เมื่อค่าที่ปรับมากกว่า 43.40 KHz ในการแสดงผลเพลงที่ทำการทดลองจะมีความเร็วมาก ในทางกลับกับเมื่อค่าที่ปรับน้อยกว่า 43.40 KHz ในการแสดงผลเพลงที่ทำการทดลองช้ามาก เมื่อปรับความถี่ 43.40 KHz ผลการทดลองจะมีประสิทธิภาพและใกล้เคียงกับเพลงที่รับเข้ามาได้จริง

บทที่ 5
บทสรุป ปัญหา แนวทางการแก้ไข และพัฒนา

5.1 บทสรุป

ในการออกแบบสร้างวงจรตัวถอดรหัสสัญญาณเสียงมาตรฐาน MP3 โดยใช้ตัวประมวลผล DSP TMS320C31 โดยศึกษาหลักการทำงานของอัลกอลิทึมของตัวถอดรหัสสัญญาณเสียง MP3 โพลีเฟส ฟิลเตอร์แบงค์ การเข้ารหัสและถอดรหัสฮัฟแมน

โพลีเฟส ฟิลเตอร์แบงค์ (The Polyphase Filter Bank) เป็นหลักการทั่วไปของการเข้ารหัสแบบเอ็มเป็ก โพลีเฟส ฟิลเตอร์แบงค์ จะแบ่งสัญญาณเสียงออกเป็นช่วงความถี่ที่มีความกว้าง (Bandwidth) ออกเป็น 32 ช่วงความถี่ย่อย (Subband) ที่เท่ากัน

การเข้ารหัสแบบฮัฟแมนจะใช้ทฤษฎีความน่าจะเป็นในการเข้ารหัส ความน่าจะเป็นที่กล่าวถึงนี้คือความซ้ำซ้อนของข้อมูลที่จะเกิดขึ้นในข้อมูลชุดใหญ่ๆ ชุดหนึ่งแทนที่เราจะแทนรหัสข้อมูลที่มีความซ้ำซ้อนกันมากๆ แทนที่จะเก็บด้วยข้อมูลบิตปกติ เราจะทำการเข้ารหัสฮัฟแมนด้วยข้อมูลบิตน้อยลงหรือเพิ่มขึ้นตามความน่าจะเป็นของข้อมูลที่จะเกิดขึ้นว่ามีสูงขนาดใด ถ้าความน่าจะเป็นเกิดขึ้นมากก็แทนด้วยข้อมูลบิตน้อยๆ และถ้าความน่าจะเป็นเกิดขึ้นน้อยก็แทนด้วยรหัสที่มีจำนวนบิตมากขึ้นตามหลักการของฮัฟแมน

การถอดรหัสฮัฟแมนที่เก็บไว้ในหน่วยความจำ ซึ่งรหัสฮัฟแมนจะเรียงกันอยู่เป็นอนุกรม การถอดรหัสต้องดึงข้อมูลออกมาแล้วเช็คข้อมูลเป็นระดับบิตเพื่อนำไปชี้ว่ารหัสดังกล่าวเป็นรหัสของข้อมูลใด

ส่วนโปรแกรมถอดรหัสสัญญาณเสียง MP3 จะใช้โปรแกรมภาษาซี (C Language Program) การสร้างโปรแกรมในการถอดรหัสจะทำงานบนดอส (DOS) โปรแกรมภาษาซีจะทำงานสิ้นสุดที่ขั้นตอนการทำ IMDCT (Inverse Modifiers Discrete Cosine Transform) จากนั้นจะส่งไปให้ DSP ประมวลผล

ส่วนโปรแกรมที่จะใช้ในการประมวลผลใน DSP คือ ส่วนของโพลีเฟส ฟิลเตอร์แบงค์ (Polyphase Filter Bank) เป็นต้นไป การเขียนโปรแกรมในส่วนนี้จึงจำเป็นต้องโหลดข้อมูล (Data) ที่ผ่านกระบวนการส่วนโปรแกรมหน่วยความจำโปรแกรม/ข้อมูลให้ถูกต้อง

ส่วนของ DSP TMS320C31 เป็นตัวประมวลผลสัญญาณที่มีความเร็วสูง และการทำงานจะเป็นแบบ Real Time และยังสามารถพัฒนาโปรแกรมบนเครื่องคอมพิวเตอร์ แล้วส่งไปยังบอร์ด

DSP จึงสามารถแก้ไขและพัฒนาโปรแกรมได้อย่างสะดวกและใช้เวลาไม่มาก ซึ่งในการทำงานของ DSP TMS320C31 เหมาะกับการประมวลผลสัญญาณเชิงเลข

5.2 ปัญหาและแนวทางแก้ไข

1) การถอดรหัสสัญญาณเสียง MP3 มีปัญหาที่การถอดสเกลแฟกเตอร์

แนวทางแก้ไข ศึกษาและทำความเข้าใจอัลกอลิทึมการถอดรหัสในส่วนของสเกลแฟกเตอร์ให้มากขึ้น และทำการหาข้อมูลเพิ่มเติมมากขึ้น

2) การออกแบบแหล่งจ่ายไฟที่จ่ายให้บอร์ดไม่เพียงพอ ในชุดวงจรขยายหน่วยความจำ, Buffer, Boot Loader เนื่องจากวงจรเหล่านี้ต้องการกระแสมาก

แนวทางแก้ไข ออกแบบแหล่งจ่ายไฟให้เพียงพอกับบอร์ด

3) การออกแบบวงจรขยายหน่วยความจำภายนอกสายที่ทำการต่อกันจะต้องมีความถูกต้องแม่นยำ ถ้าช๊อตกันจะทำให้การทดสอบหน่วยความจำมี Error เกิดขึ้น

แนวทางแก้ไข ออกแบบและทำลายวงจรพิมพ์แทนการต่อโดยใช้สาย

4) ในการทดสอบไฟล์ Wave อาจจะมีเสียงรบกวนเกิดขึ้น ทำให้เสียงไม่มีความ ชัดเจน

แนวทางแก้ไข ถอดแหล่งจ่ายไฟที่จ่ายให้กับวงจรขยายหน่วยความจำ, Buffer, Boot Loader ออกเพราะไม่ต้องใช้ เนื่องจากไฟล์ Wave สามารถทำการประมวลผลสัญญาณผ่านบอร์ดได้เลย หรือทำการกดปุ่ม F1,F2 เลือกความถี่ที่ 43.40KHz และระหว่างที่ทำการทดลองนั้นไม่ควร กดคีย์ใดๆ

5) การกำหนดค่าตัวแปรในการเขียนโปรแกรมมีขนาดหน่วยความจำไม่เพียงพอกับการเก็บค่าในการถอดรหัสไฟล์ MP3

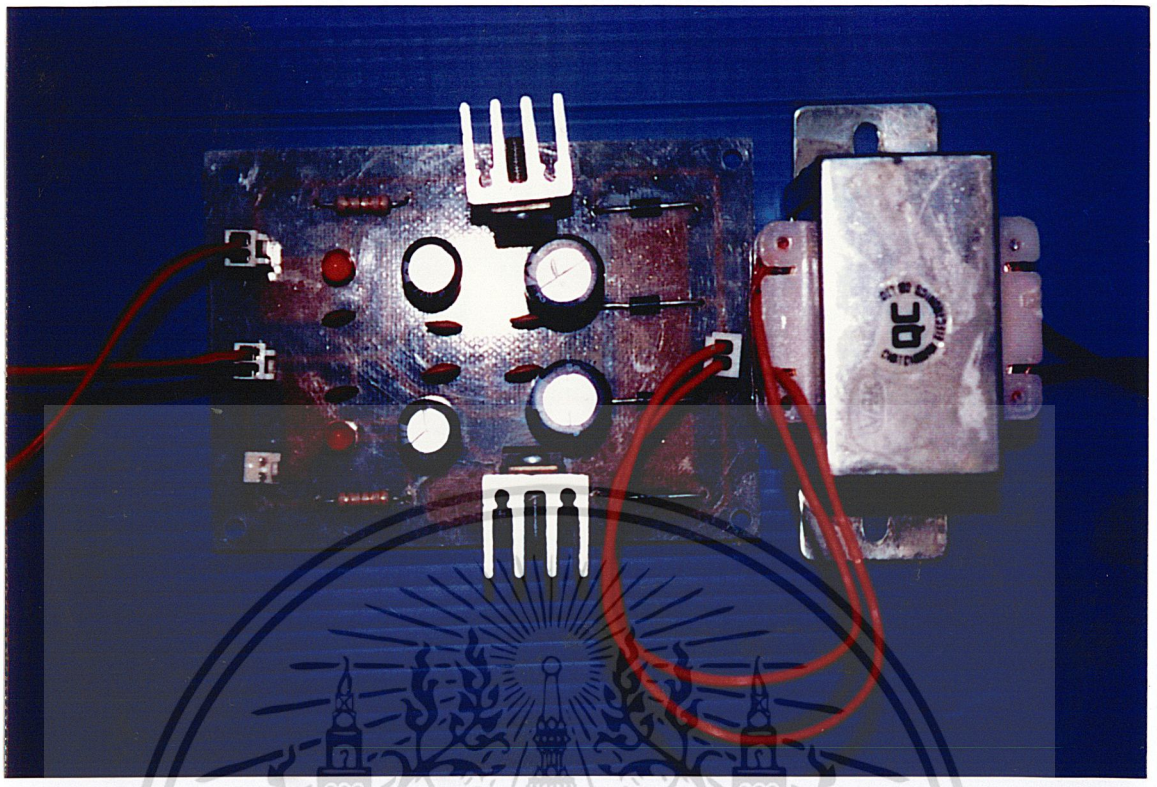
แนวทางแก้ไข แก้ไขโดยใช้ตัวแปรจากชนิด Unsigned Int เป็น Long Int

5.3 แนวทางในการพัฒนา

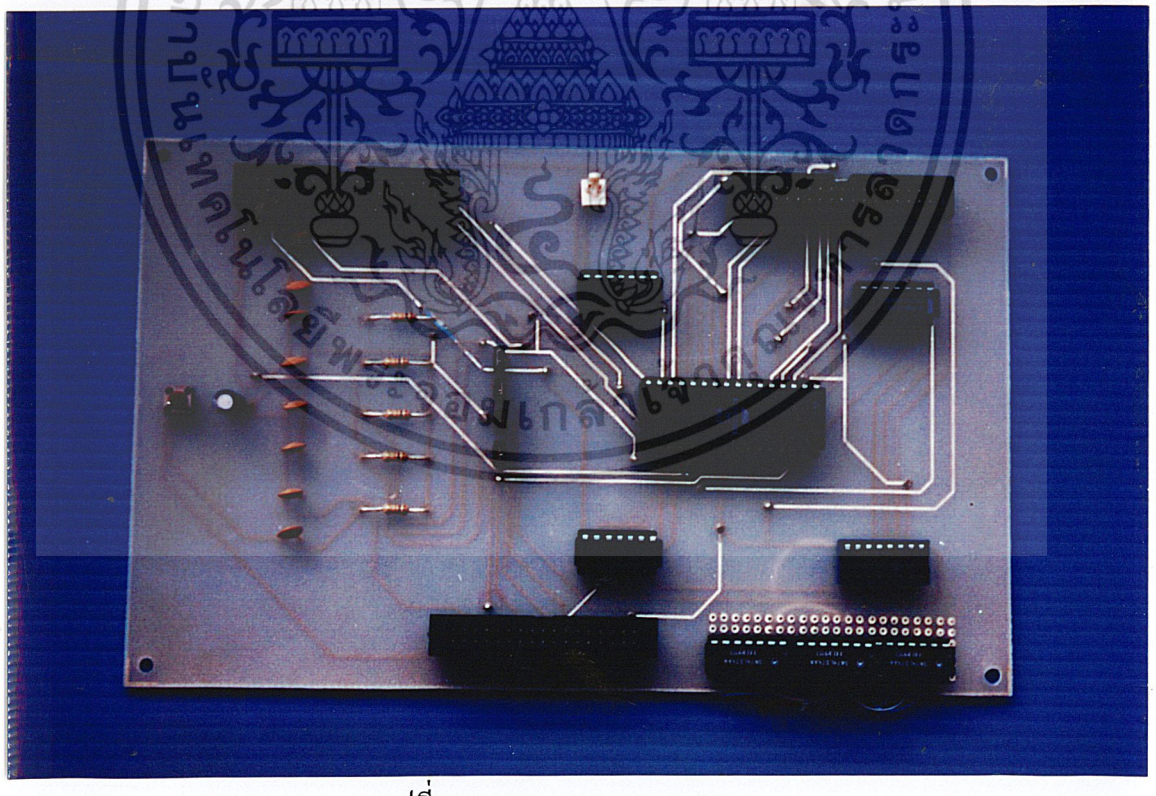
1) วงจรตัวถอดรหัสสัญญาณเสียง MP3 สามารถเก็บข้อมูล MP3 ได้มากกว่า 128 Kbyte สามารถต่อหน่วยความจำภายนอก (RAM) เพิ่มเติมเพื่อเก็บข้อมูลได้มากขึ้น

2) นำไปประยุกต์ใช้งานในการถอดรหัส MPEG1 - layer1 และ layer2 เพิ่มเติมซอฟต์แวร์เพื่อให้สามารถถอดรหัส MPEG1 - layer1 และ layer2 ได้ เพราะมีหลักการใกล้เคียงกัน

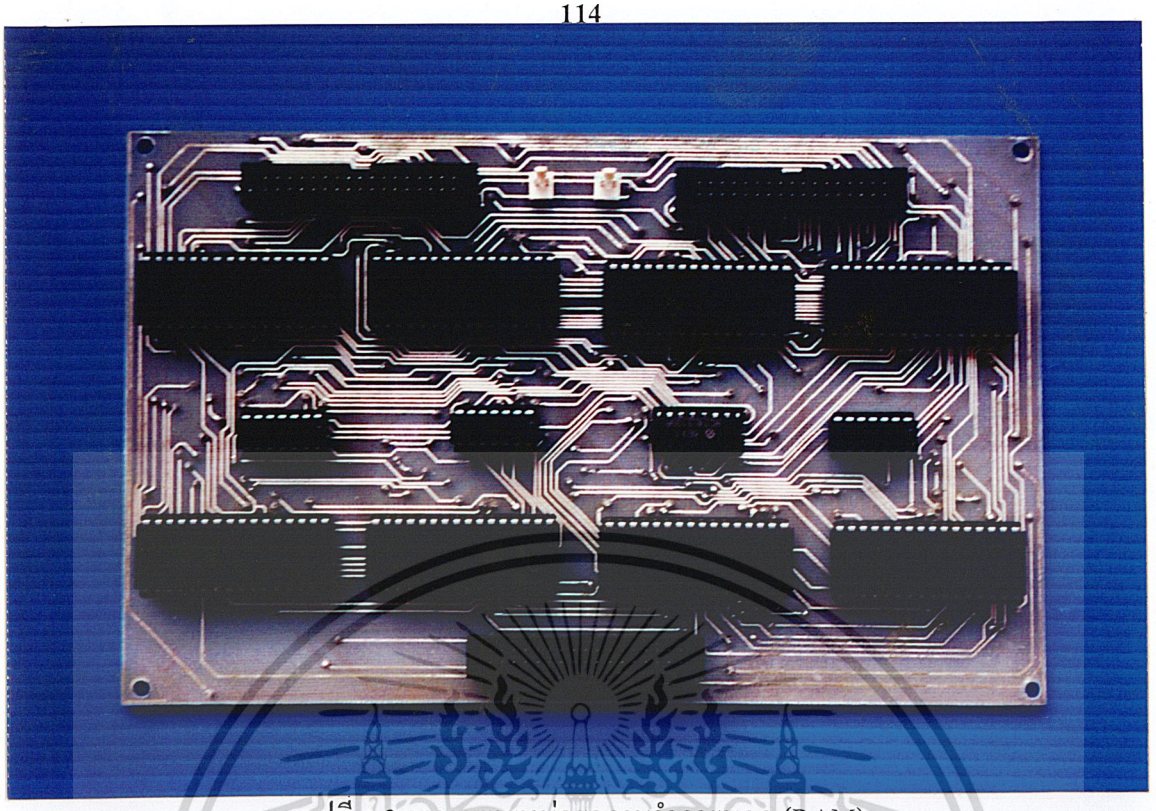
ภาคผนวก ก
เครื่องต้นแบบ



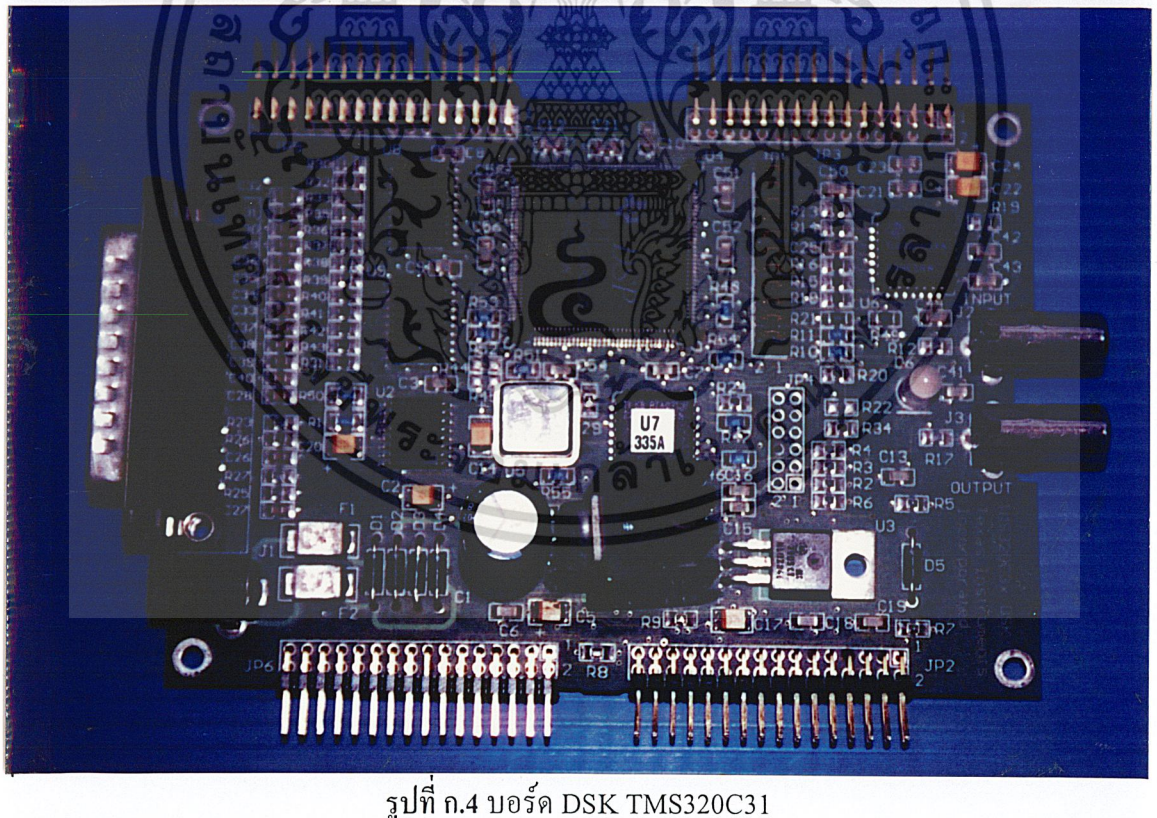
รูปที่ ก.1 แผงวงจรแหล่งจ่ายไฟ



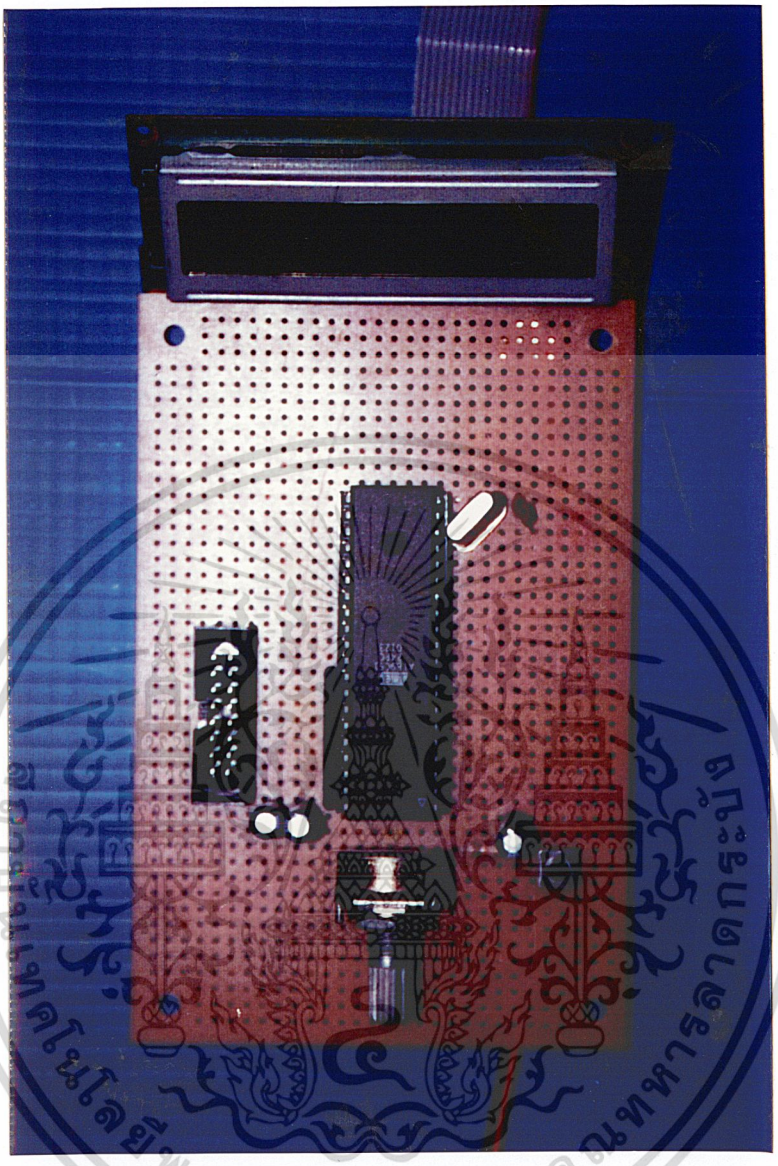
รูปที่ ก.2 แผงวงจร Boot Loader



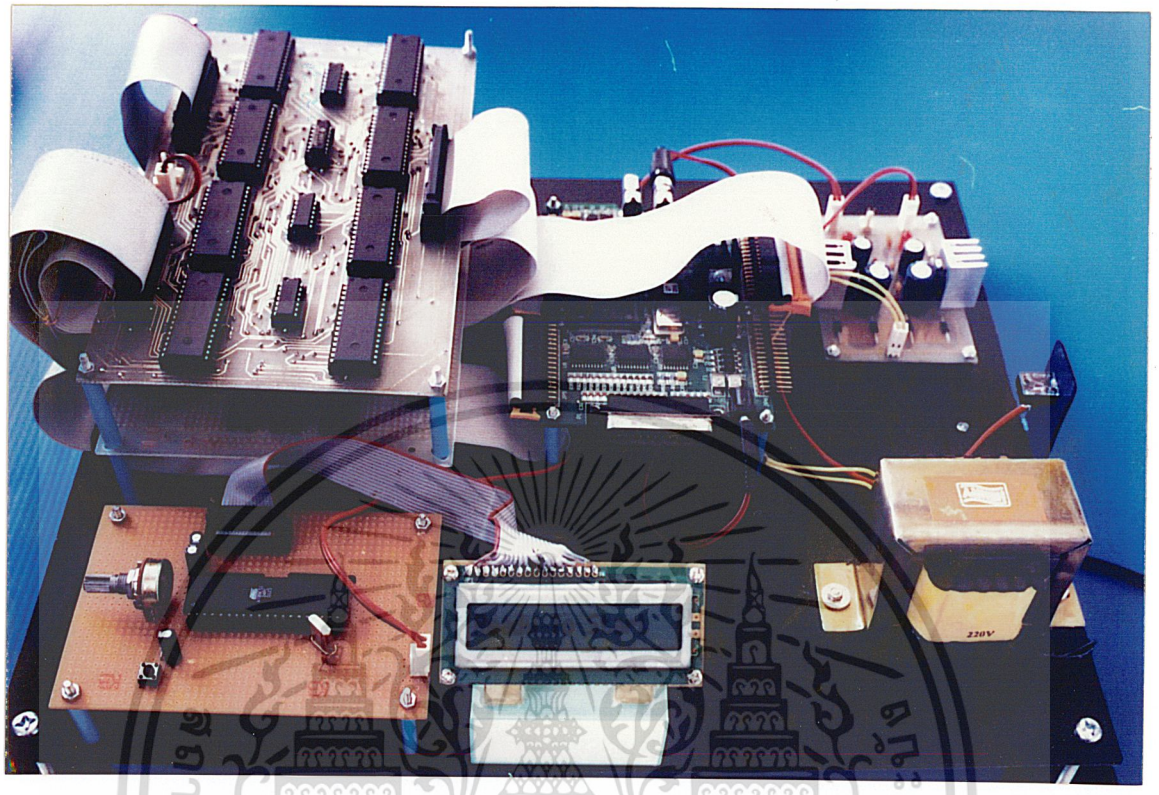
รูปที่ ก.3 วงจรขยายหน่วยความจำภายนอก (RAM)



รูปที่ ก.4 บอร์ด DSK TMS320C31

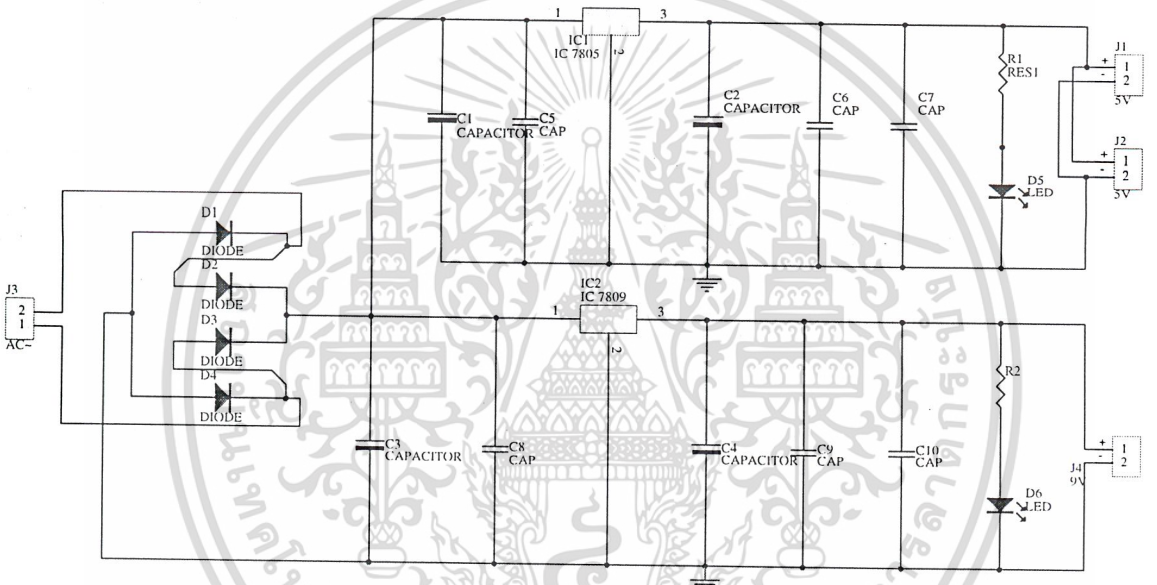


รูปที่ ก.5 แผงวงจรแสดงผลด้วย LCD

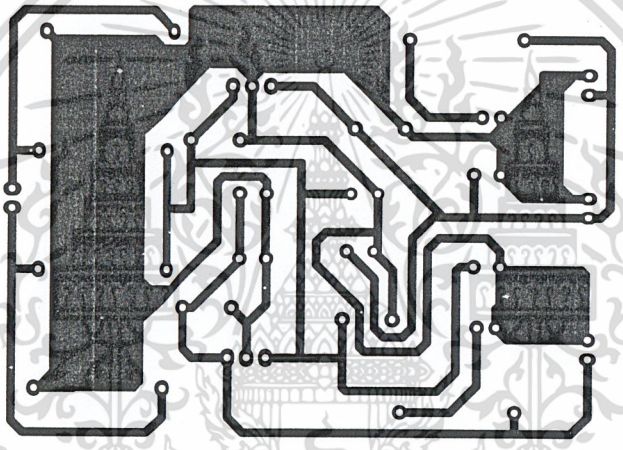


รูปที่ ก.6 ตัวถอดรหัสสัญญาณเสียง MP3

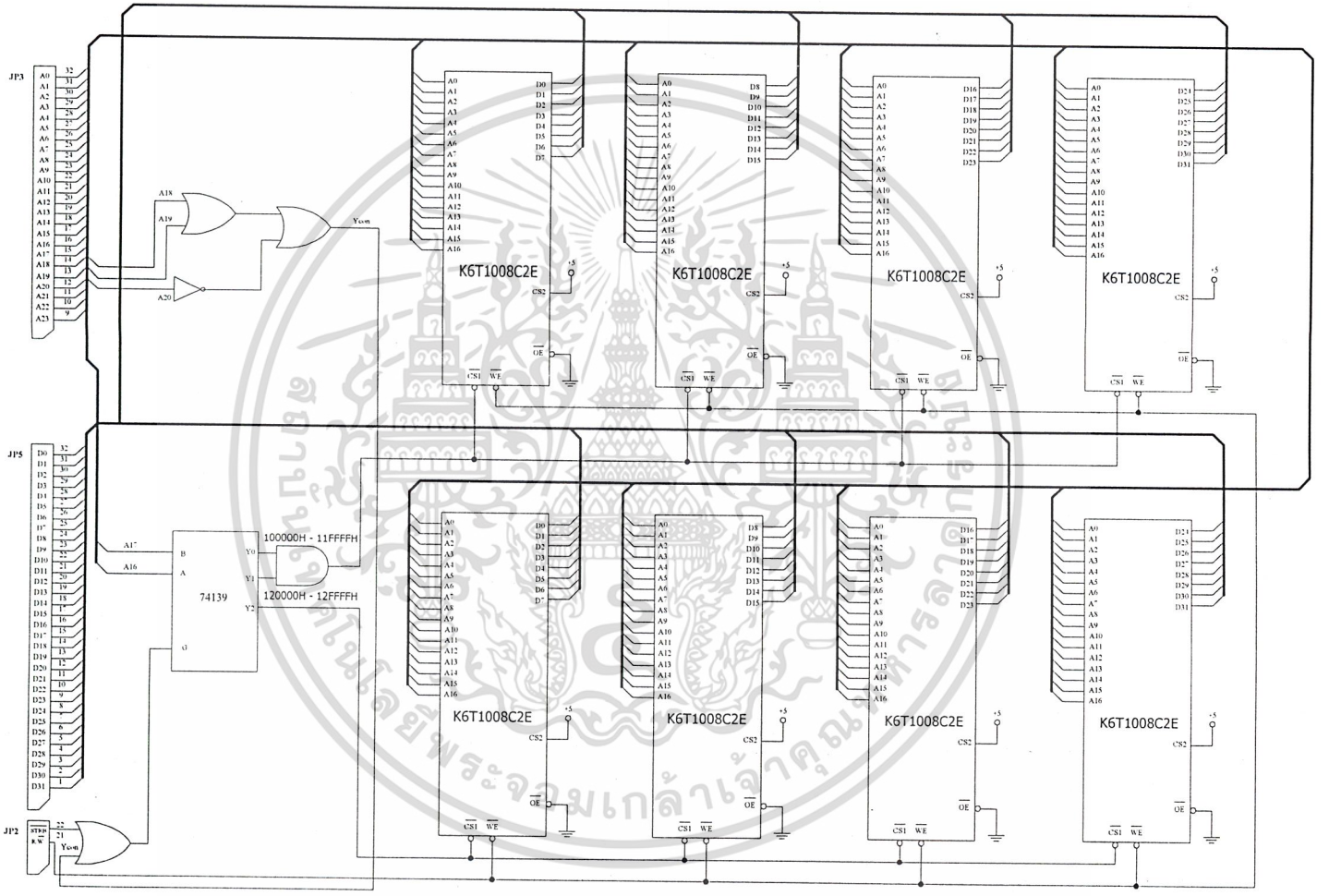
ภาคผนวก ข
วงจรและแผ่นวงจรพิมพ์



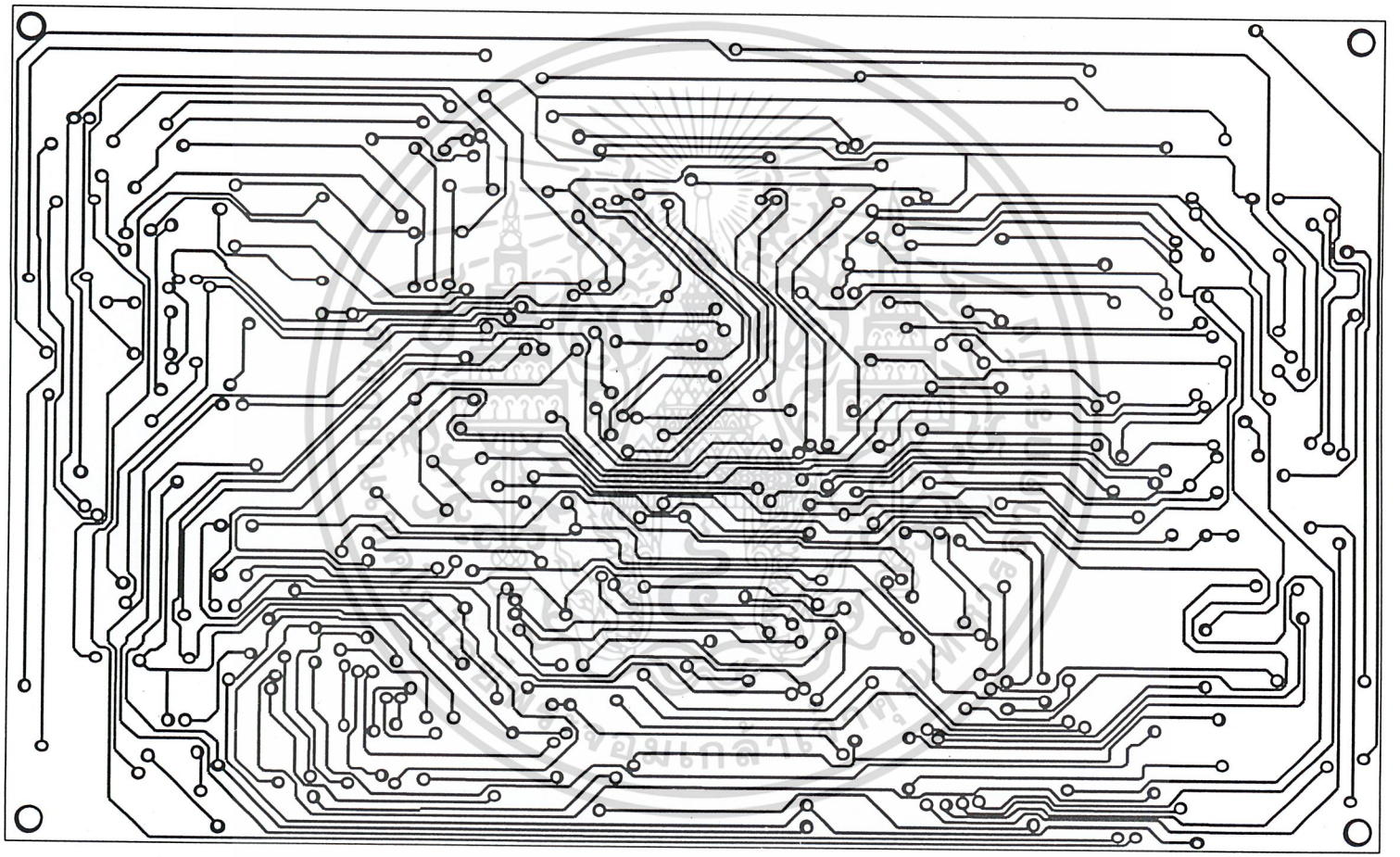
รูปที่ ข.1 วงจรแหล่งจ่ายไฟ



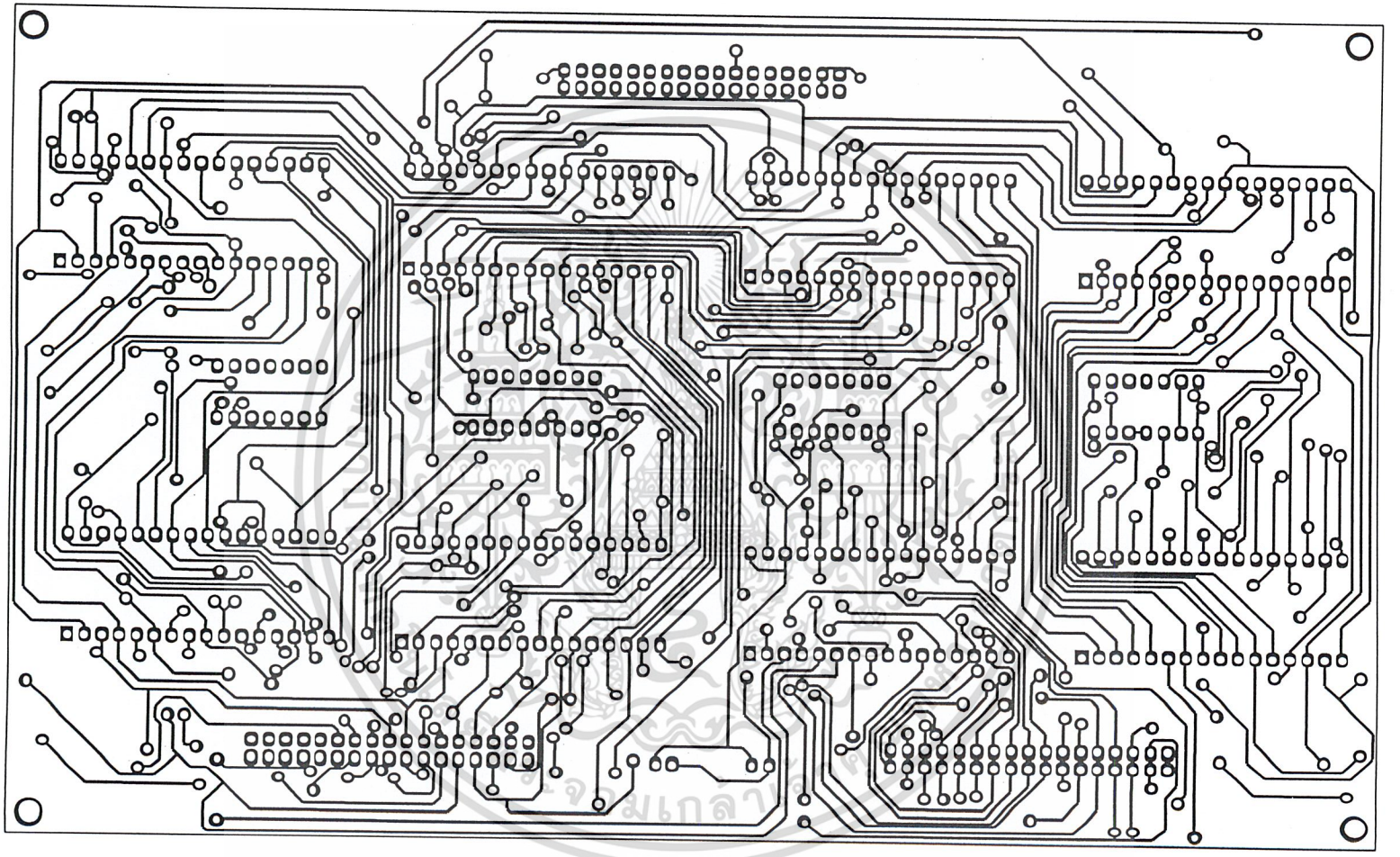
รูปที่ ข.2 แผ่นวงจรพิมพ์แหล่งจ่ายไฟ



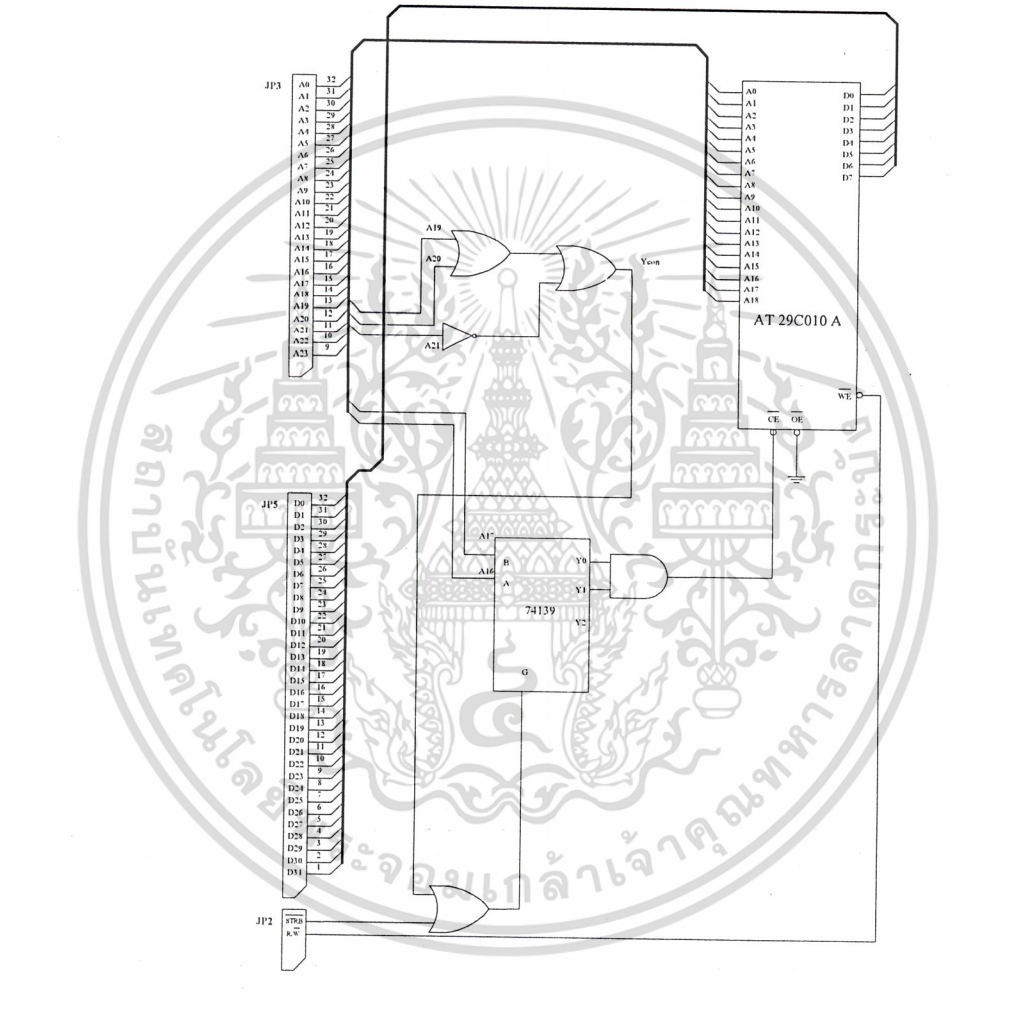
รูปที่ ข.3 วงจรหน่วยความจำภายนอก



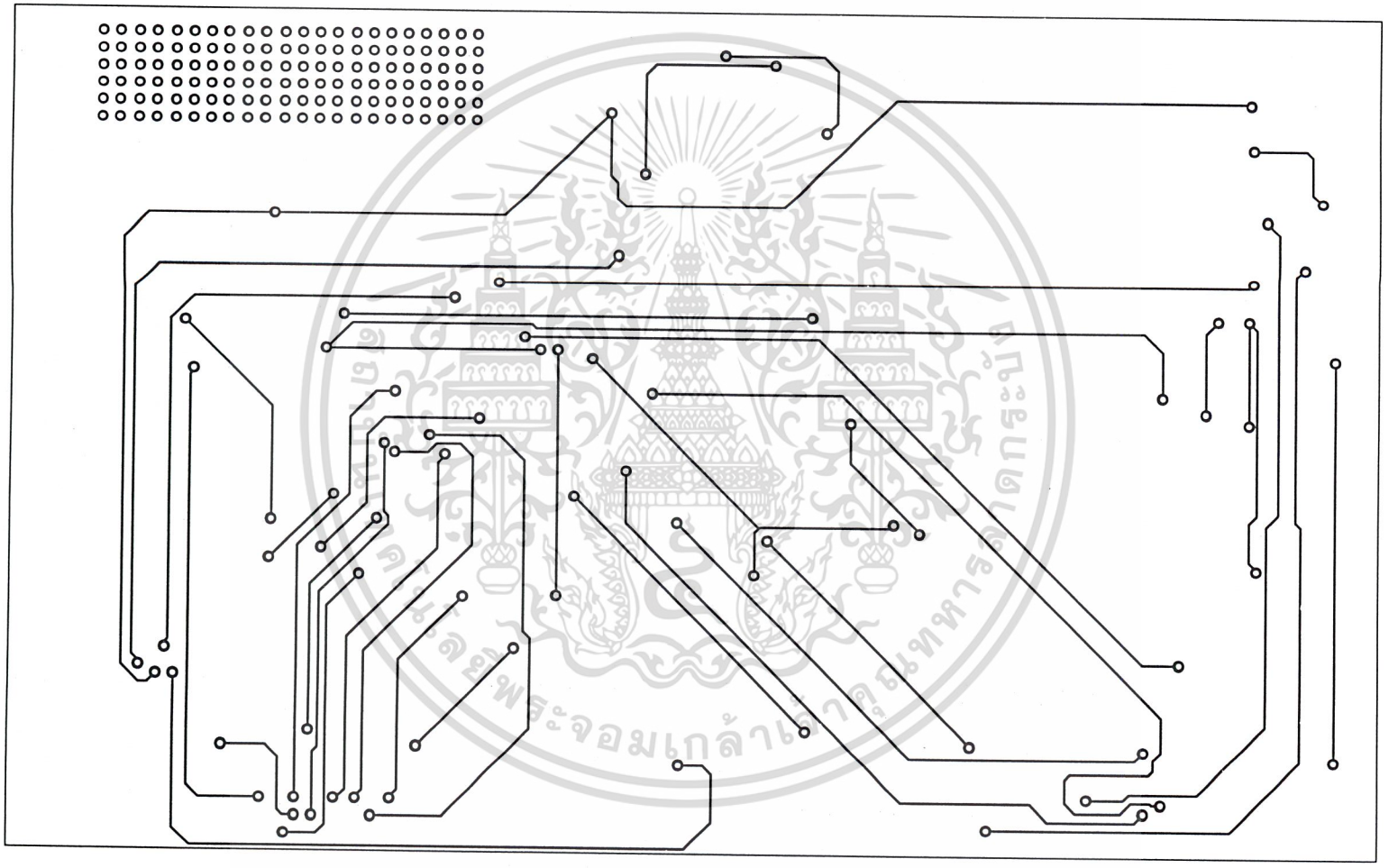
รูปที่ ข.4 แผงวงจรพิมพ์ของหน่วยความจำภายนอก



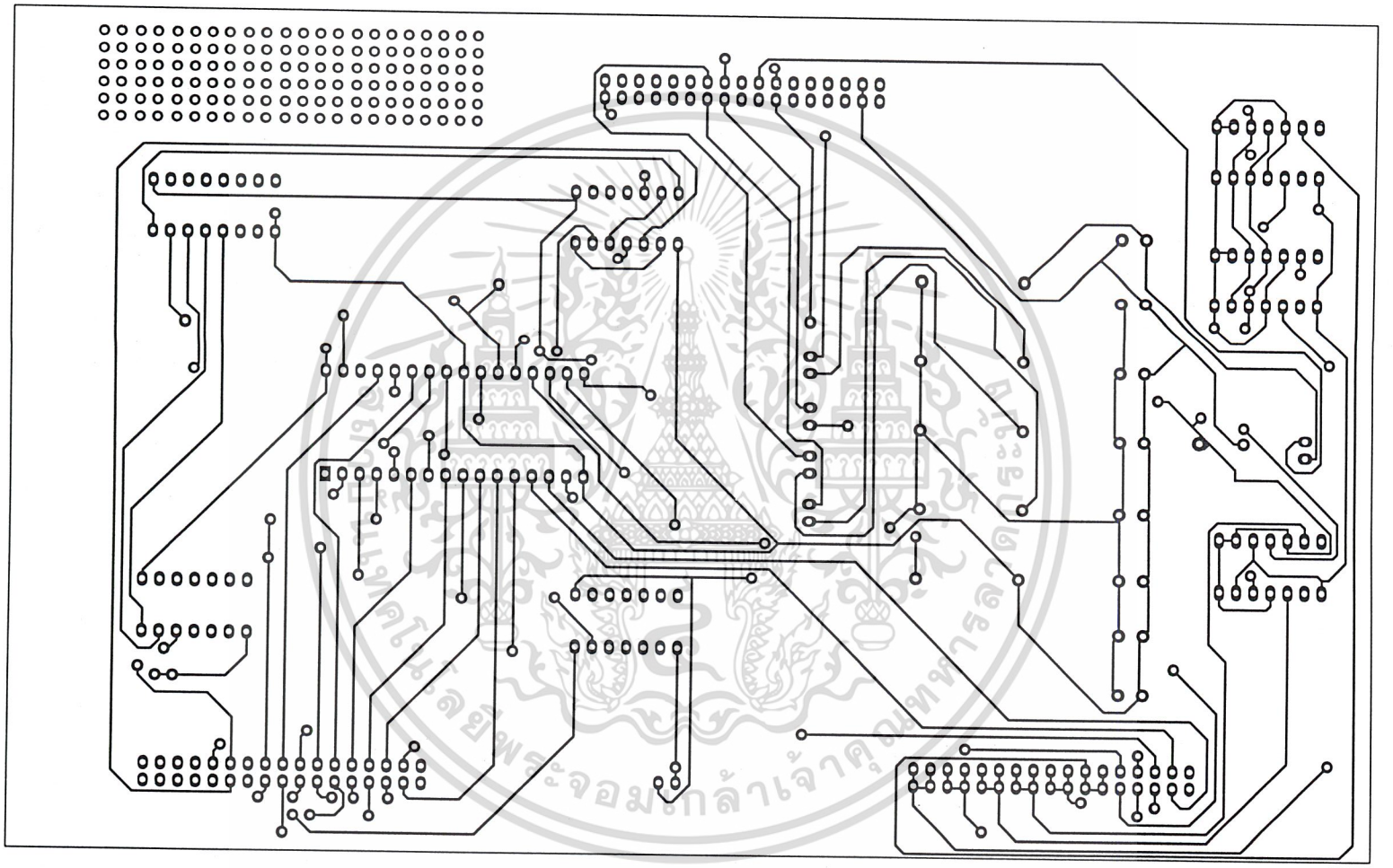
รูปที่ ข.5 แผ่นวงจรพิมพ์ของหน่วยความจำภายนอก (ด้านล่าง)



รูปที่ ข.6 วงจรบูตโหลดเดอร์

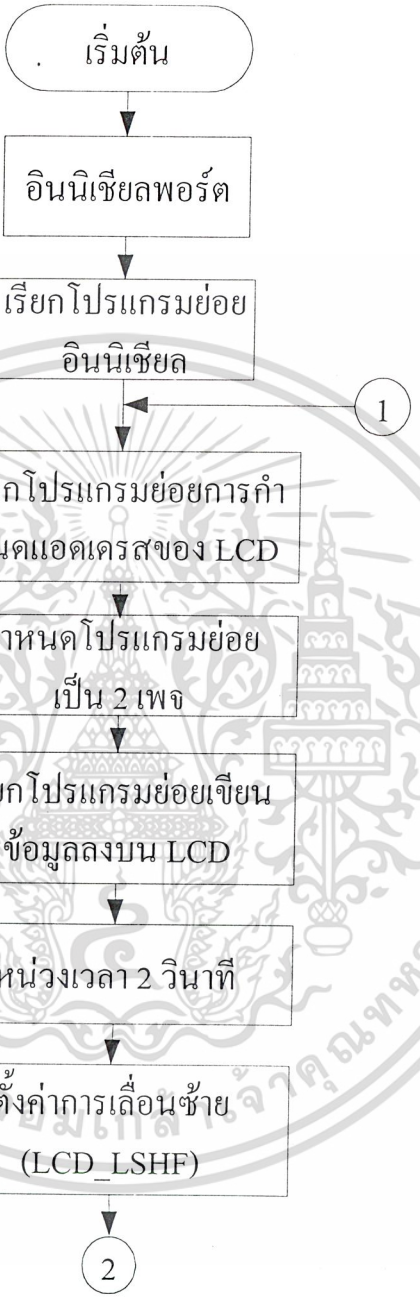


รูปที่ ข.7 แผ่นวงจรพิมพ์วงจรบูตโหลด



รูปที่ ข.8 แผ่นวงจรพิมพ์วงจรบูตโหลดเดอร์ (ด้านล่าง)

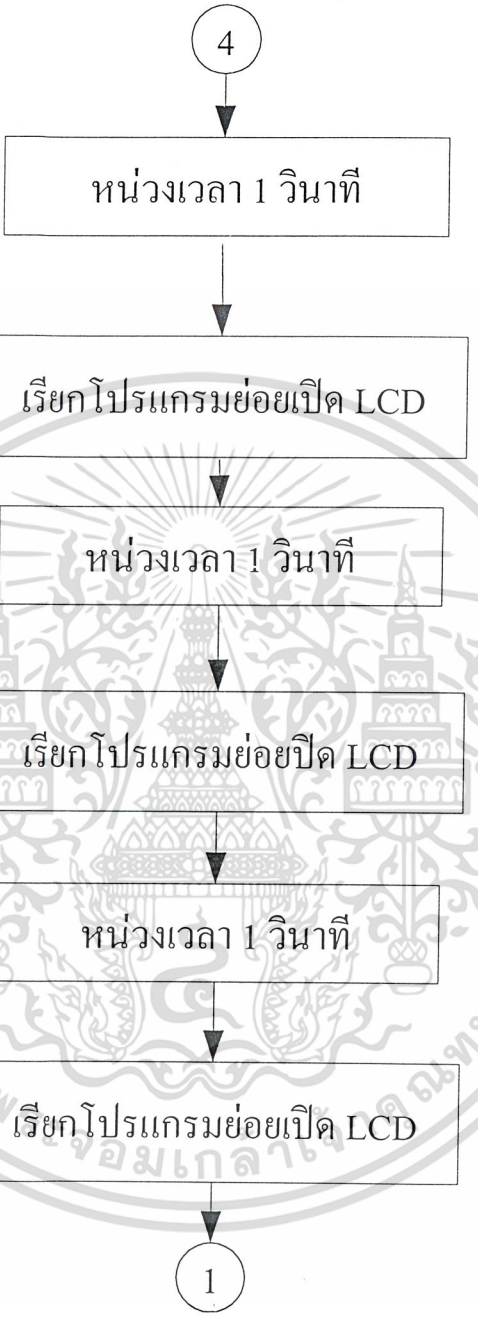
ภาคผนวก ค
แผนผังการทำงาน และโปรแกรม



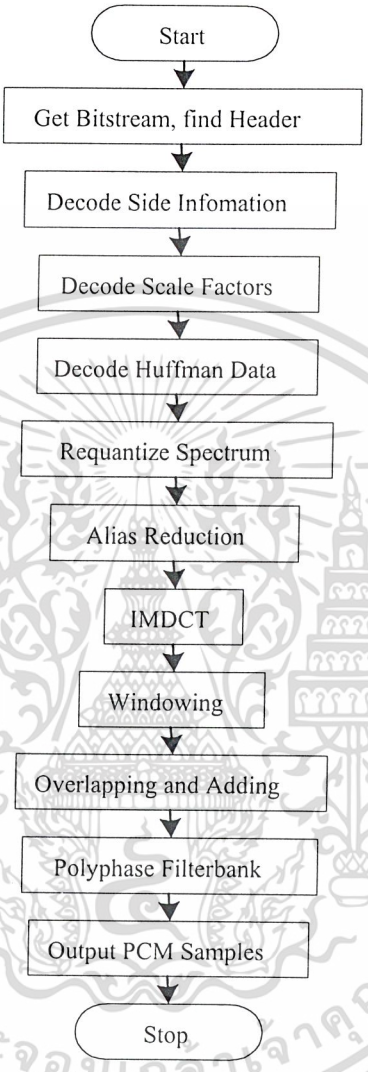
รูปที่ ค.1 แผนผังการทำงานของโปรแกรมส่วนแสดงผล LCD 16 ตัวอักษร 1 บรรทัด

รูปที่ ค.2 แผนผังการทำงานของโปรแกรมส่วนแสดงผล LCD 16 ตัวอักษร 1 บรรทัด

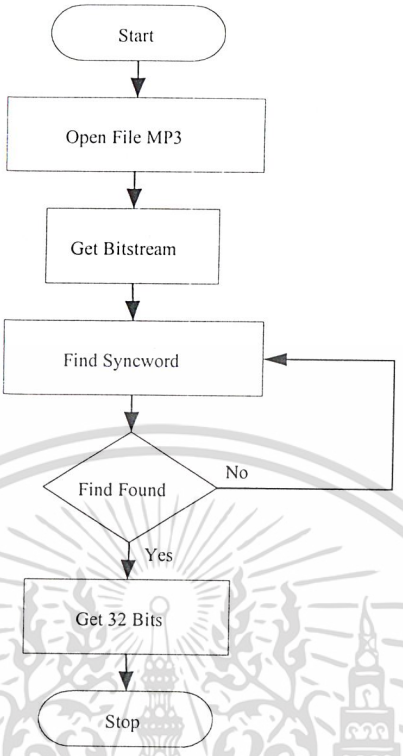
รูปที่ ค.3 แผนผังการทำงานของโปรแกรมย่อยส่วนแสดงผล LCD 16 ตัวอักษร 1 บรรทัด



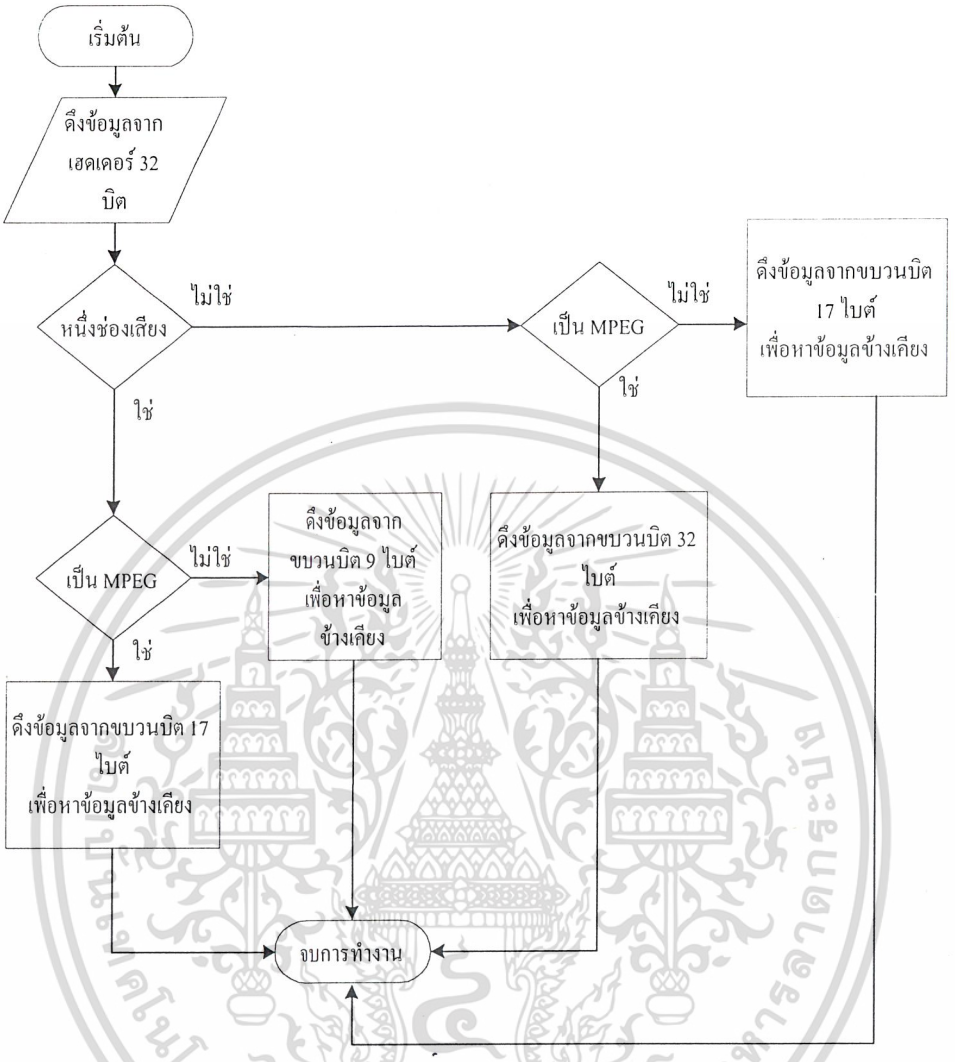
รูปที่ ค.4 แผนผังการทำงานของโปรแกรมย่อยส่วนแสดงผล LCD 16 ตัวอักษร 1 บรรทัด



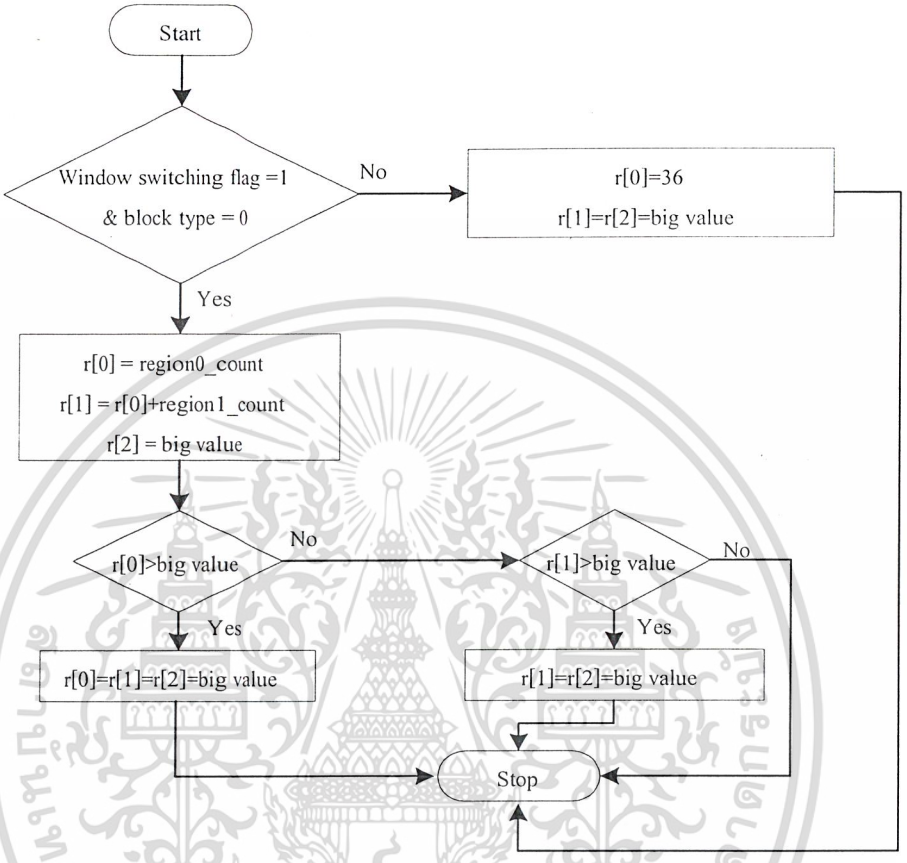
รูปที่ ค.5 ผังงานของตัวถอดรหัสไฟล์ "เอ็มเป็ก-1 เลเยอร์-3"



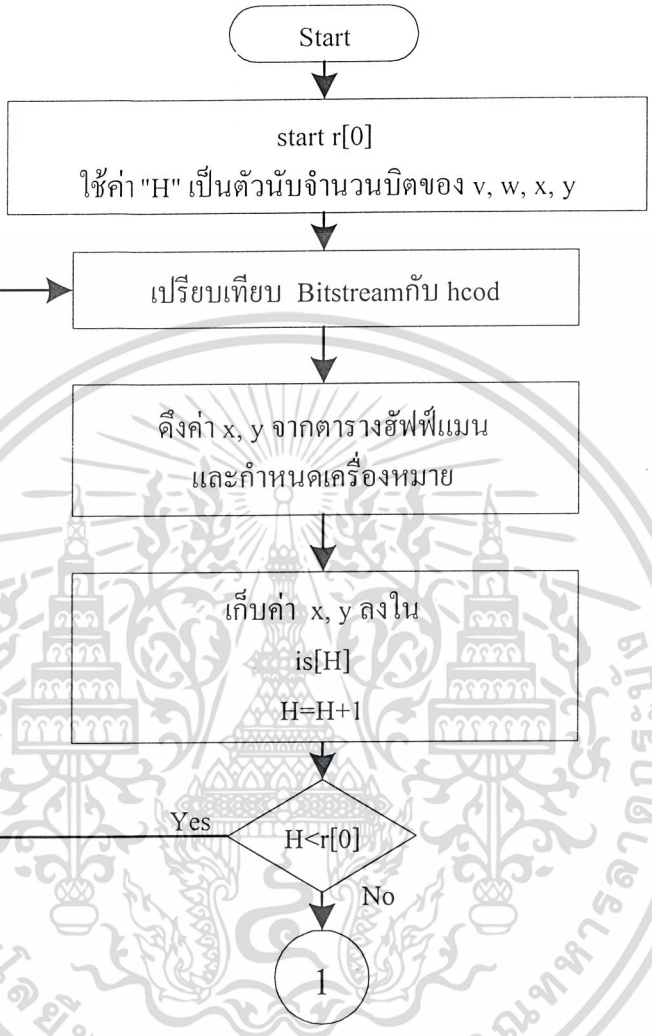
รูปที่ ค.6 ผังงานของโปรแกรมการดึงข้อมูลและค้นหาส่วนหัว



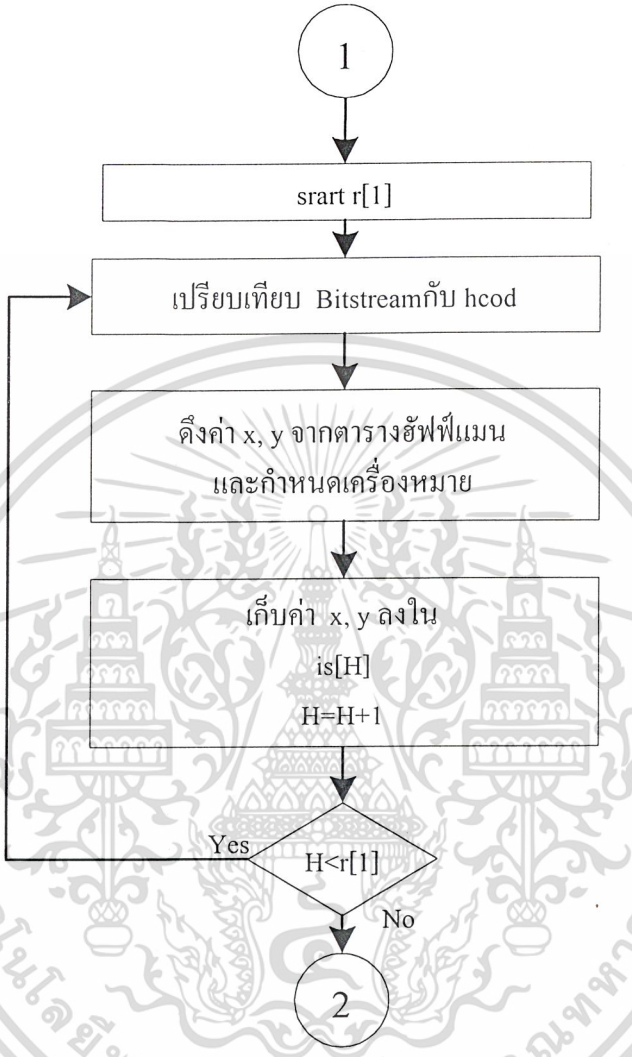
รูปที่ ค.7 ผังงานของโปรแกรมการถอดรหัสข้อมูลข้างเคียง



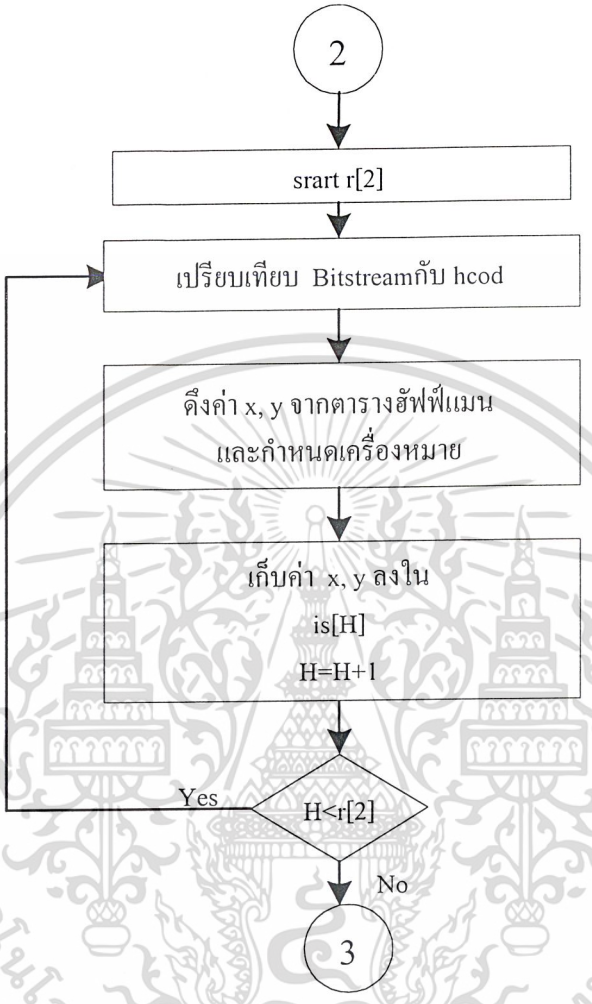
รูปที่ ค.8 ผังงานของโปรแกรมการกำหนดค่าของจุดสิ้นสุดของ 3 ภาคแรก



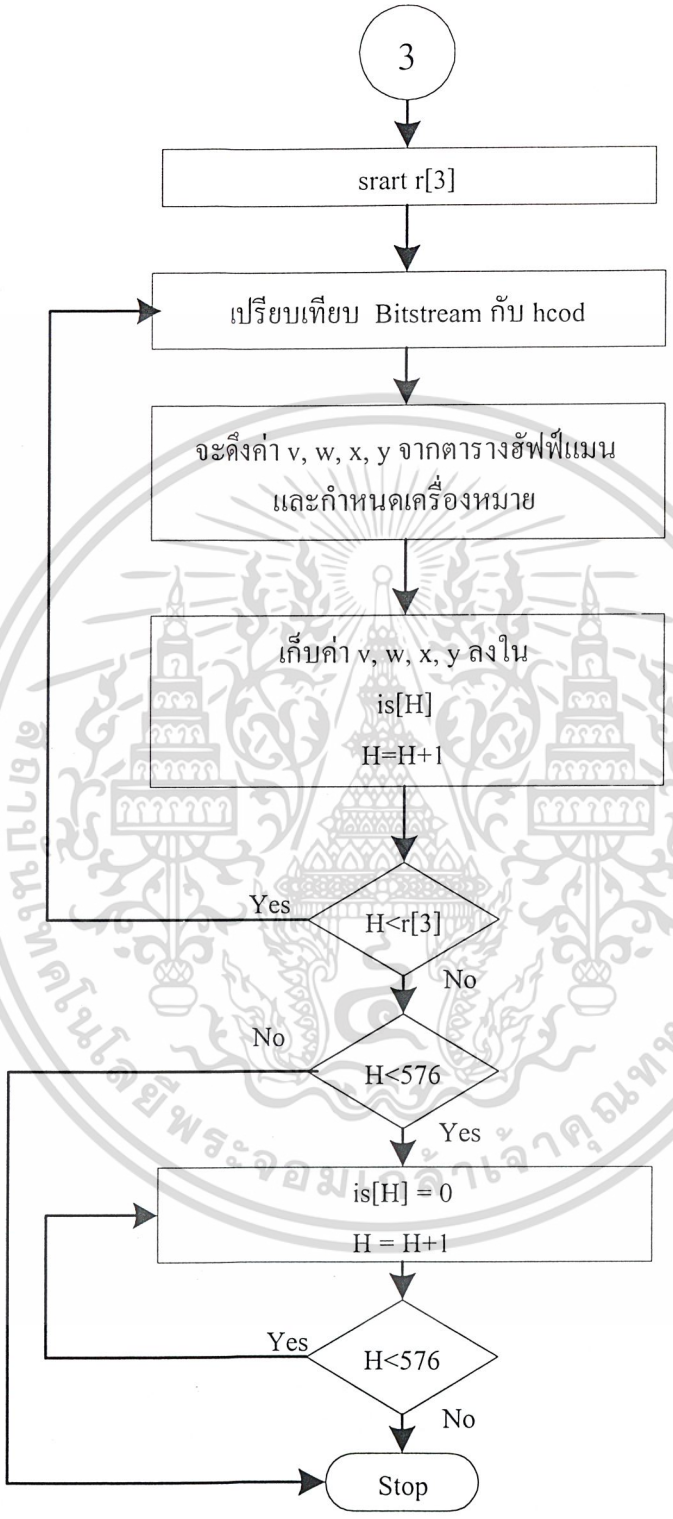
รูปที่ ค.9 ผังงานของโปรแกรมการถอดรหัสฮัฟฟ์แมน



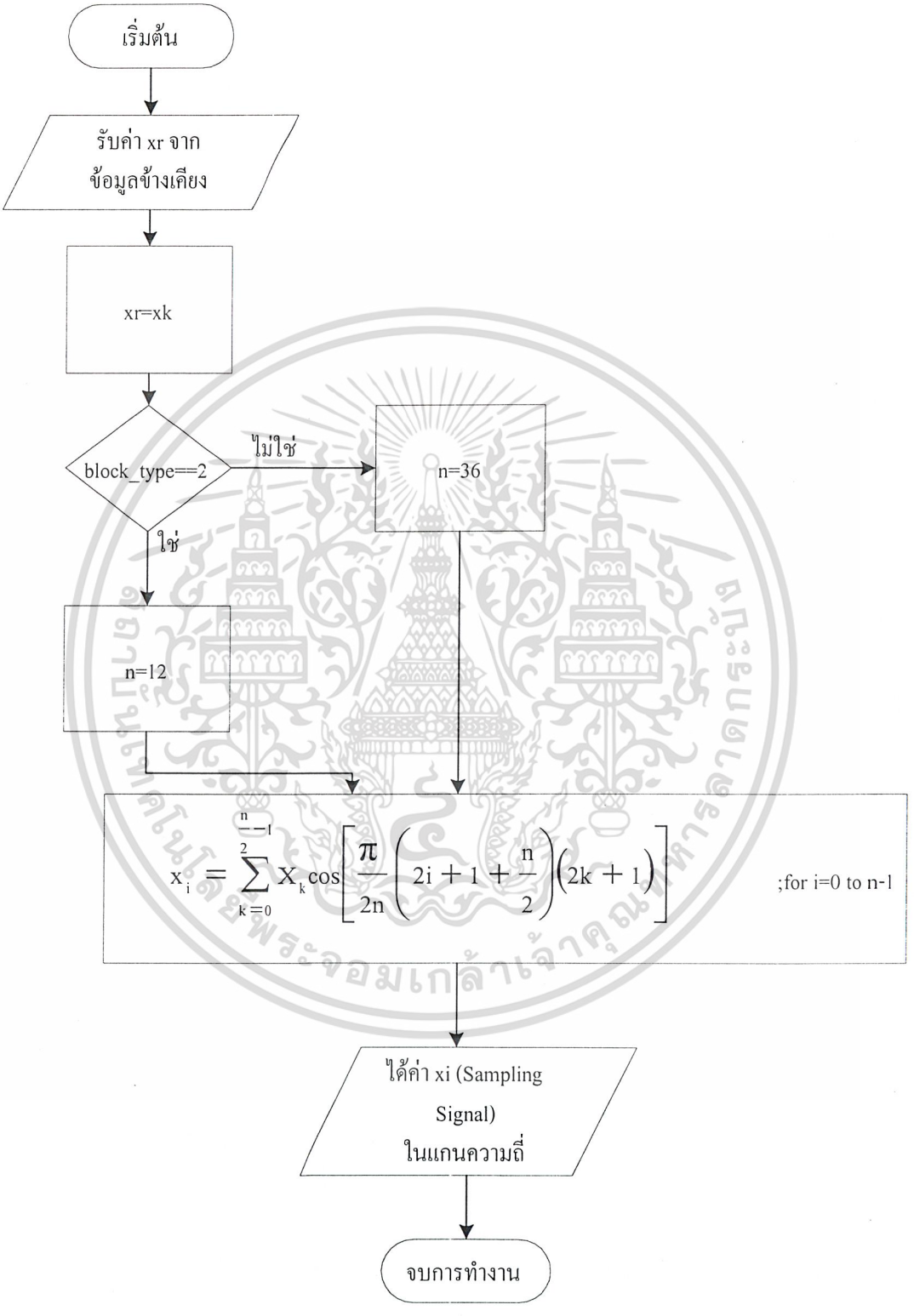
รูปที่ ค.10 ผังงานของโปรแกรมการถอดรหัสฮัฟฟ์แมน



รูปที่ ค.11 ผังงานของโปรแกรมการถอดรหัสฮัฟฟ์แมน



รูปที่ ค.12 ผังงานของโปรแกรมการถอดรหัสฮัฟฟ์แมน



รูปที่ ค.13 ผังงานของโปรแกรมการทำ IMDCT

รูปที่ ค.14 ผังงานของโปรแกรมการรวมสัญญาณจากแต่ละย่านความถี่ย่อย

```
/*Program  TESTING TMS320C31 EXTERNAL SRAM MEMORY    */
/*For Test EXTERNAL SRAM MEMORY Board MP3 Decorder Project */
/*Develope by ED.ENG 23 Electronics and Computer 2/1 */
/*MR.Chaloempong Khan-ngen  ID 44035319*/
/*MR.Montree     Phusatchum ID 44035341*/
/*MR.Montree     Srisa-nga  ID 44035342*/
#include "dsklib.h"
int testpattern(unsigned long address, unsigned long length, unsigned long data)
{
  unsigned long addr;
  unsigned long c31data;
  int ch;
  int i;
  int error = 0;
  addr = address;
  gotoxy(1,15);
  clreol();
  printf("Data 0x%.8lX is being written to C31 Memory ", data);
  for (i = 0; i < length; i++)
  {
    putmem(addr, 1, &data);
    addr++;
  }
  gotoxy(1,15);
  clreol();
  printf("Data is being read from C31 Memory ");
  gotoxy(1,16);
  addr = address;
  for (i = 0; i < length; i++)
```

```
{
    getmem(addr, 1, &c31data);
    printf("0x%.8lX = 0x%.8lX \r", addr, c31data);
    if (c31data != data)
    {
      printf("0x%.8lX = 0x%.8lX - FAILED ", addr, c31data);
      gotoxy(1,24);
      clreol();
      printf("Press 'c' to Continue or 'q' to quit.  ");
      while (!kbhit());
      ch = getch();
      gotoxy(1,24);
      clreol();
      gotoxy(1,16);
      clreol();
      error++;
      if ((ch == 'q') || (ch == 'Q')) return error;
    }
   addr++;
 }
 gotoxy(1,17);
 clreol();
 if (error == 0) printf("Memory Test PASSED");
 else printf("Memory Test encountered %d errors", error);
 gotoxy(1,24);
 clreol();
 printf("Press 'c' to Continue.  ");
 while (!kbhit());
 ch = getch();
 return error;
}
```

```
int testcntpat(unsigned long address, unsigned long length)
{
  unsigned long addr;
  unsigned long c31data;
  unsigned long data;
  int ch;
  int i;
  int error = 0;

  addr = address;
  gotoxy(1,15);
  clreol();
  printf("Data is being written to C31 Memory ");
  for (i = 0; i < length; i++)
  {
    data = addr;
    putmem(addr, 1, &data);
    addr++;
  }
  gotoxy(1,15);
  clreol();
  printf("Data is being read from C31 Memory ");
  gotoxy(1,16);
  addr = address;
  for (i = 0; i < length; i++)
  {
    getmem(addr, 1, &c31data);
    printf("0x%.8lX = 0x%.8lX \r", addr, c31data);
    if (c31data != addr)
    {
```

```
clreol();
        printf("Press 'c' to Continue or 'q' to quit.  ");
        while (!kbhit());
        ch = getch();
        gotoxy(1,24);
        clreol();
        gotoxy(1,16);
        clreol();
        error++;
        if ((ch == 'q') || (ch == 'Q')) return error;
     }
     addr++;
  }
  gotoxy(1,17);
  clreol();
  if (error == 0) printf("Memory Test PASSED");
  else printf("Memory Test encountered %d errors", error);
  gotoxy(1,24);
  clreol();
  printf("Press 'c' to Continue.  ");
  while (!kbhit());
  ch = getch();
  return error;
}

void main()
{
  int sel;
clrscr();
```

```
Detect_Windows();
Init_Communication(10000);
HALT_CPU();    // Put C31 into spin0 mode
do
{
  clrscr();
  gotoxy(1,5);
  printf("\t\t\tTESTING C31 EXTERNAL SRAM MEMORY \n\n");
  printf("\t\t\t  1).....0xAAAAAAAA    \n");
  printf("\t\t\t  2).....0x55555555    \n");
  printf("\t\t\t  3).....0x12345678    \n");
  printf("\t\t\t  4).....Data = Address \n");
  printf("\t\t\t  5).....Quit          \n\n");
  printf("\t\t\t  Select option number (1-4) :  ");
  while (!kbhit());
  sel = getch();
  switch (sel)
  {
    case '1': testpattern(0x100000L, 0x8000, 0xAAAAAAAAL);
              break;
    case '2': testpattern(0x100000L, 0x8000, 0x55555555L);
              break;
    case '3': testpattern(0x100000L, 0x8000, 0x12345678L);
              break;
    case '4': testcntpat(0x100000L, 0x8000);
              break;
  }
} while (sel != '5');
```

รูปที่ ค.15 โปรแกรมทดสอบหน่วยความจำภายนอก

```
/*
 MP3 decoder Now Not Real Time ,!Now Not Finish
  Programmed by ED Engineer 23
  Layer 3 decoder                - version 1.00
  For save to Testmp.dec to polyphasefilter with MP3 Decoder By TMS 320c31
*/
#include "stdio.h"
#include <stdlib.h>
#include <conio.h>
#include <mem.h>
#include <math.h>
/* #define compile32bit */
/* Global constants */
unsigned char buffer[BUFFER_SIZE+BUFFER_AUX],_buffer[80];
int  scalefac_l[2][2][22];
int scaIefac_s[2][2][13][3];
int append,_bptr;
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};

/* layer 3 frame lengths
   formula = (bitrate[Kbits/s]/samp.freq[Khz/s])*(1152[samples/frame]/8[bits/byte]) */
/*const int layer3_framelength[3][15]={{0,104,130,156,182,208,261,313,365,417,522,626, 731,
835,1044},
                        {0, 96,120,144,168,192,240,288,336,384,480,576, 672, 768, 960},
                        {0,144,180,216,252,288,360,432,504,576,720,864,1008,1152,1440}};

/************************** Global variables ****************************/
FILE *filestream_in;
/* Filestream */
```

```
FILE *filestream_out;
/* Filestream */
unsigned char datastream_bitcounter;      /* Datastream bitcounter */
unsigned char datastream_temp;            /* Datastream current byte */
int windowbufferleft_index;               /* index in buffer */
int windowbufferright_index;              /* index in buffer */
int outputbufferleft_index;               /* index in buffer */
int outputbufferright_index;              /* index in buffer */
float *windowbufferleft_pointer;          /* start adres of buffer */
float *windowbufferright_pointer;         /* start adres of buffer */
int *outputbufferleft_pointer;            /* start adres of buffer */
int *outputbufferright_pointer;           /* start adres of buffer */
const  windowbuffer_mask;                 /* circular mask */
const outputbuffer_mask;                  /* circular mask */

int readfilestream(void);
int getbits(int numbits);
int getbyte(void);
int decode_frame(void);
/*****************************************************************************/
/************************** Function Pototype*********************************/
/******************************** Read File**********************************/
int readfilestream(void)
/* Read one byte from file
   in:  nothing
   out:  0 = OK
         -1 = error
*/
{
 if ((datastream_temp=fgetc(filestream_in))==EOF)
```

```
{
   return(-1);
 }
 datastream_bitcounter=8;                    /**** Read 8 bit Data *****/
 return(0);
}
/******************************** getbits ***********************************/
int getbits (int numbits)
/* Read a number of bits from the bitstream
  in:     numbits = number of bits
  out:  value
              -1      = error
*/
{
 unsigned int i,j;
 if (numbits==0)
  {
  return(0);
  }
j=0;
for (i=1;i<=numbits;i++)
 {
  if (datastream_bitcounter==0)
 {
    if (readfilestream()!=0)
      {
       return(-1);
      }
  }
   j=(j<<1)+((datastream_temp&0x80)>>7);
```

```
    datastream_temp=datastream_temp<<1;
    datastream_bitcounter--;
   }
  return(j);
}
/****************************** getbyte ********************************/
int getbyte (void)
/* Read byte from the bitstream
   in:     nothing
   out:    value
           -1= error
*/
{
 if (datastream_bitcounter!=0)
 {
   return(-1);
 }
 if (readfilestream()!=0)
 {
   return(-1);
  }
 datastream_bitcounter=0;
 return(datastream_temp);
}
/*************************** synthfilter ***************************/
void synthfilter (float *in,float *windowbuffer1_pointer,int windowbuffer1_index,int *out)
 Calculate the IDCT of the 32 input-samples, and perform windowing
  in:     pointer to float array with 32 input-samples
          pointer to float array where the 32 output-samples are to be located
```

```
{
int i,j;                        /* counters */
float sum1;                     /* temp var */
float sum2;                     /* temp var */
float    sum3;                  /* temp var */
float sum4;                     /* temp var */
float ee2[8];                   /* input butterfly - level 2, even,even */
float eo2[8];                   /* input butterfly - level 2, even,odd  */
float oe2[8];                   /* input butterfly - level 2, odd,even  */
float oo2[8];                   /* input butterfly - level 2, odd,odd   */
float IDCT_result[32];          /* temp var */
/************** calculate input butterflies - level 2 ******************/
for (i=0;i<8;i++)
 {
  ee2[i]=(*(in+i)+*(in+31-i))+(*(in+15-i)+*(in+16+i));
  eo2[i]=(*(in+i)+*(in+31-i))-(*(in+15-i)+*(in+16+i));
  oe2[i]=(*(in+i)-*(in+31-i))+(*(in+15-i)-*(in+16+i));
  oo2[i]=(*(in+i)-*(in+31-i))-(*(in+15-i)-*(in+16+i));
 }
/******************** do four IDCT 8x8 - level 2 ***********************/
for (i=0;i<8;i++)
 {
  sum1=0.0;
  sum2=0.0;
  sum3=0.0;
  sum4=0.0;
  for (j=0;j<8;j++)
  {
    sum1+=DCT08_table_eveneven[i][j]*ee2[j];
    sum2+=DCT08_table_evenodd [i][j]*eo2[j];
```

```
    sum3+=DCT08_table_oddeven [i][j]*oe2[j];
    sum4+=DCT08_table_oddodd  [i][j]*oo2[j];
   }
  IDCT_result[(i<<2)  ]=sum1;
  IDCT_result[(i<<2)+2]=sum2;
  IDCT_result[(i<<2)+1]=sum3;
  IDCT_result[(i<<2)+3]=sum4;

 }
now we have a 32x32 IDCT */
put this in the (circular) windowing buffer and expand to 32x64 IDCT */
for (i=0;i<16;i++)
 {
*(windowbuffer1_pointer+((windowbuffer1_index+i  )&&windowbuffer_mask))=IDCT_result
[i+16];

*(windowbuffer1_pointer+((windowbuffer1_index+i+17)&&windowbuffer_mask))=IDCT_resu
lt[31-i];
*(windowbuffer1_pointer+((windowbuffer1_index+i+32)&&windowbuffer_mask))=IDCT_resu
lt[16-i];
*(windowbuffer1_pointer+((windowbuffer1_index+i+48)&&windowbuffer_mask))=IDCT_resu
lt[i];
 }
*(windowbuffer1_pointer+((windowbuffer1_index +16)&&windowbuffer_mask))=0.0;
/* perform windowing and calculate the 32 samples */
for (i=0;i<31;i++)
 {
  sum1=0;
  for (j=0;j<16;j++)
  {
```

```
    sum1+=layer12_windowcoef[i*32+j]*(windowbuffer1_pointer+(windowbuffer1_index+i+
(j<<6)+((j&&0x01)<<5))&&windowbuffer_mask);
  }
   *(out+i)=sum1;
  }
return;
  }

/************************ decode frame ************************************/

 int decode_frame (void)

/* decode one frame
   in:     nothing
   out:  0 = OK
         1 = sync not found or Not Support(end of file)
         2 = Not ISO 11172-3 audio, layer 1-3
*/

{
 unsigned char layer;       /* layer number of frame 1-3      3=layer1
                                                              2=layer2
                                                              1=layer3          */
 unsigned char ID;          /* ID MPEG                        */
 unsigned char mpeg_version;     /* MPEG Audio version                */
 unsigned char protection_bit;   /* CRC for this frame?   0=Yes       */
 unsigned int bitrate_index;     /* bitrate index                     */
 unsigned int sampling_frequency;          /* sampling frequency      */
```

```
unsigned char padding_bit;                  /* padding bit? 1=yes      */
unsigned char private_bit;                  /* private bit             */
unsigned char mode;                         /* mode 0=stereo
                                                    1=joint stereo
                                                    2=dual channel
                                                    3=single channel */
unsigned char mode_extension;               /* mode extension          */
unsigned char copyright;                    /* copyright       1=yes   */
unsigned char original_copy;                /* original/copy  1=copy   */
unsigned char emphasis;                     /* emphasis                */
int  CRC_check;                             /* CRC data                */
int intensity_stereo;          /* intensity stereo       1=on          */
int ms_stereo;                 /* ms_stereo              1=on          */
unsigned char temp;            /*temp                                  */
/************** check sync word************************/
int i,j,sb,ch,s,gr;
i=0;
j=0;
while ((i=((i<<1)&0xfff)+j)!=0xfff)     /** 1111 1111 111 = 12Bits**/
  {
  if ((j=getbits(1))==-1)
  {
    printf("sync not found");
    return(1);
  }
  }
printf("\n\n Bits 1-12 = 1111 1111 1111  sync word OK\n");
temp=getbits(1);
```

```
/***************************** read header info ******************************/
 temp=getbits(1);
 if ((ID=temp)==1)          /******* Check ID (Old)**********/
  {
   printf(" ID      = 1          MPEG ISO 11172-3\n");
  }
 else
  {
   printf(" ID      = 0          MPEG Not ISO 11172-3\n");
   return(-2);
  }
                    /**** Check Layer description ****/
 temp=getbits(2);
 if ((layer=temp)==0)
 {
   printf(" Layer    = 00          Layer description reserved\n");
         return(-1);              /* if layer = 0 -> return with error */
  }
 else if ((layer=temp)==1)
  {
   printf(" Layer    = 01          Mpeg 1 Layer 3\n");
  }
 else if ((layer=temp)==2)
 {
   printf(" Layer    = 10          Mpeg 1 layer 2\n");
}
 else
 {
   printf(" Not For Standard MPEG\n");
 }
```

```
            /**** Check protection bit ****/
 temp=getbits(1);
 if((protection_bit=temp)==0)
  {
   printf(" error protection     = 0            No \n");
  }
 else
  {
   printf(" error protection     = 1            Yes \n");
 }
            /**** Check bitrate ****/
 temp=getbits(4);
 if ((bitrate_index=temp)==0)
  {
   printf(" bitrate   = 0000          Free \n");
  }
 else if ((bitrate_index=temp)==1)
  {
   printf(" bitrate   = 0001          32kbit/sec \n");
   bitrate_index = 32;
  }
 else if ((bitrate_index=temp)==2)
  {
   printf(" bitrate   = 0010          40kbit/sec \n");
   bitrate_index = 40;
  }
 else if ((bitrate_index=temp)==3)
  {
   printf(" bitrate   = 0011          48kbit/sec \n");
```

```
bitrate_index = 48;
  }
 else if ((bitrate_index=temp)==4)
  {
   printf(" bitrate   = 0100          56kbit/sec \n");
   bitrate_index = 56;
  }
 else if ((bitrate_index=temp)==5)
  {
   printf(" bitrate   = 0101          64kbit/sec \n");
   bitrate_index = 64;
  }
 else if ((bitrate_index=temp)==6)
  {
   printf(" bitrate   = 0110          80kbit/sec \n");
   bitrate_index = 80;
  }
 else if ((bitrate_index=temp)==7)
  {
   printf(" bitrate   = 0111          96kbit/sec \n");
   bitrate_index = 96;
  }
 else if ((bitrate_index=temp)==8)
  {
   printf(" bitrate   = 1000          112kbit/sec \n");
   bitrate_index = 112;
  }
 else if ((bitrate_index=temp)==9)
  {
   printf(" bitrate   = 1001          128kbit/sec \n");
```

```
bitrate_index = 128;
  }
 else if ((bitrate_index=temp)==10)
  {
   printf(" bitrate   = 1010          160kbit/sec \n");
   bitrate_index = 160;
  }
 else if ((bitrate_index=temp)==11)
  {
   printf(" bitrate   = 1011          192kbit/sec \n");
   bitrate_index = 192;
  }
 else if ((bitrate_index=temp)==12)
  {
   printf(" bitrate   = 1100          224kbit/sec \n");
   bitrate_index = 224;
  }
 else if ((bitrate_index=temp)==13)
  {
   printf(" bitrate   = 1101          256kbit/sec \n");
   bitrate_index = 256;
  }
 else if ((bitrate_index=temp)==14)
  {
   printf(" bitrate   = 1110          160kbit/sec \n");
   bitrate_index = 160;
  }
 else
  {
   printf(" bitrate   = 1111          Bad \n");
```

```
return(-1);
  }

                    /**** Check sampling frequency ****/
 temp=getbits(2);
 if ((sampling_frequency=temp)==0)
  {
    printf(" Sampling rate    = 00    44100Hz\n");
    sampling_frequency=441000;
  }
 else if ((sampling_frequency=temp)==1)
  {
    printf(" Sampling rate    = 01    48000Hz\n");
    sampling_frequency=48000;
  }
  else if ((sampling_frequency=temp)==2)
  {
    printf(" Sampling rate    = 10    32000Hz\n");
    sampling_frequency=32000;
  }
  else
  {
    printf(" reserv \n");
 }
                    /**** Check padding bit ****/
 temp=getbits(1);
 if ((padding_bit=temp)==0)

  {
```

```
printf(" padding bit = 0  frame is not padded\n");
 }
 else
 {
   printf(" padding bit = 1  frame is padded with one extra bit\n");
 }
                    /***** Check Private bit  *****/

 temp=getbits(1);
 private_bit=temp;

                  /***** Not ISO/IEC 11172-3 ********/

 printf (" Private bit = %d\n ",private_bit);

                  /***** Check mode  *****/

 temp=getbits(2);
if ((mode=temp)==0)
 {
   printf("Channel Mode    = 00    Stereo\n");
 }
else if ((mode=temp)==1)
 {
   printf("Channel Mode    = 01    Joint stereo (Stereo)\n");
 }
else if ((mode=temp)==2)
 {
   printf("Channel Mode    = 10    Dual channel (Stereo)\n");
 }
else
```

```
{
    printf("Channel Mode    = 11    Single channel (Mono)\n");
}

                    /***** Check Mode extension  *****/

temp=getbits(2);
printf(" Mode extension (Only Joint stereo) =");
if ((mode_extension=temp)==0)
 {
   printf(" 00 Intensity stereo = Off  MS stereo = Off');
 }
else if ((mode_extension=temp)==1)
 {
   printf(" 01 Intensity stereo = On  MS stereo = Off\n");
}
else if ((mode_extension=temp)==2)
 {
   printf(" 10 Intensity stereo = Off   MS stereo = On\n");
 }
else
 {
   printf(" 11 Intensity stereo = On   MS stereo = On\n");
 }

            /***** Check copyright bit  *****/

temp=getbits(1);
```

```
if ((copyright=temp)==0)
 {
   printf(" Copyright = 0  Audio is not copyrighted\n");
 }
else
 {
   printf(" Copyright = 1  Audio is copyrighted\n");
 }

                    /***** Check original copy bit  *****/
temp=getbits(1);
if ((original_copy=temp)==0)
 {
   printf(" Original = 0  Copy of original media\n");
 }
else
 {
   printf(" Original = 1  Original media\n");
}

          /***** Check emphasis *****/
temp=getbits(2);
if ((emphasis=temp)==0)
 {
   printf(" Emphasis = 00  none\n");
 }

else if ((emphasis=temp)==1)
```

```
{
    printf(" Emphasis = 01  50/15 ms\n");
}
 else if ((emphasis=temp)==2)
 {
    printf(" Emphasis = 10  reserved\n");
    return(-1);
 }
else
 {
   printf(" Emphasis = 11  CCIT J.17\n");
 }

/********************** read audio info + data + decode ************************/

switch (layer)
 {
  case 3 :                    /* LAYER 1 */
  {
    printf(" Not support Layer 1");
    break;
  }
  case 2 :                    /* LAYER 2 */
  {
    printf(" Not support Layer 2");
    break;
  }
  case 1 :                    /* ************  LAYER 3 For Decord   ************ */
```

```
{
    int  main_data_begin;
    int  private_bits;
    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][2];
    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];
    int  nch,slen1,slen2;
    int  ch,scfsi_band,gr,region,window,sfb;
    int  Size;                          /****** A Frame Leng ******/
    long int  i,j,k;
    int  sc[12];
/********************* Find Frame Size *****************/
/*************** How to calculate frame size  **********/
/****FrameSize = 144 * BitRate / SampleRate + Padding****/
 if (sampling_frequency = 0 )        /*** 41000 Hz ***/
  {
   Size = (((144*1000*bitrate_index) / 44100) + padding_bit);
```

```
   printf(" \n Frame Size = %d byte\n",Size);
   sampling_frequency = 441;
  }
 else if (sampling_frequency = 1 )    /*** 48000 Hz***/
  {
   Size = (((144*1000*bitrate_index) / 48000) + padding_bit);
   printf(" \n Frame Size = %d byte\n",Size);
   sampling_frequency = 480;
  }
 else                          /*** 3200 Hz***/
  {
   Size = (((144*1000*bitrate_index) / 32000) + padding_bit);
   printf(" \n Frame Size = %d byte\n",Size);
   sampling_frequency = 320;
  }

/******************** CRC Check ****************************/
 if (protection_bit = 1 )
  {
   printf(" CRC Check = No \n");
  }
 else
  {
   temp=getbits(16);     /******** CRC have 16 Bits *********/
   if ((CRC_check=temp)==0)
    {
    printf(" CRC Check = Yes \n");
    }
   else
    {
```

```
printf(" Error \n");
    }
  }
/***************** Decode side information **********************/
/************ Stereo Mode = 32 Bytes Or 256 Bits******************/
/************ Mono   Mode = 17 Bytes Or 136 Bits******************/
  getch();
  clrscr();
  temp=getbits(9);
  main_data_begin=temp;
  printf( " main_data_begin = %d From 9 bits \n",temp);
  if (mode==3)  /*** mono ***/          /* Discard 5 "private bits" */
   {
    temp=getbits(5);
    private_bits=temp;
    printf( " private_bits = %d From 5 bits \n",private_bits);
    nch=1;
   }
 else       /*** Stereo ***/   /* Discard 3 "private bits" */
  {
   temp=getbits(3);
   private_bits=temp;
   printf( " private_bits = %d From 3 bits \n",private_bits);
   nch=2;
  }
for (ch=0;ch<nch;ch++)
  {
   for (scfsi_band=0;scfsi_band<4;scfsi_band++)
     {
      scfsi[ch][scfsi_band]=getbits(1);
```

```
}
    }
  for (gr=0;gr<2;gr++)
    {
     for (ch=0;ch<nch;ch++)
       {
        part2_3_length[gr][ch]=getbits(12);
        big_values[gr][ch]=getbits(9);
        global_gain[gr][ch]=getbits(8);
        scalefac_compress[gr][ch]=getbits(4);
        if (window_switching_flag[gr][ch]=getbits(1))
         {          /* window switching !! */
          block_type[gr][ch]=getbits(2);
          mixed_block_flag[gr][ch]=getbits(1);
          for (region=0;region<2;region++)
            {
             table_select[gr][ch][region]=getbits(5);
            }
          for (window=0;window<3;window++)
            {
             subblock_gain[gr][ch][window]=getbits(3);
            }
       }
      else
       {          /* no window switching */
      block_type[gr][ch]=0;
      for (region=0;region<3;region++)
       {
        table_select[gr][ch][region]=getbits(5);
       }
```

```
region0_count[gr][ch]=getbits(4);
        region1_count[gr][ch]=getbits(3);
       }
      preflag[gr][ch]=getbits(1);
      scalefac_scale[gr][ch]=getbits(1);
      count1table_select[gr][ch]=getbits(1);
      }
   }

/******************** calculate length of header ****************************/
/* All header information is read. Now put remaining bits in the bitbucket
    Header size = 4+2*(1-Protection_Bit)+(nch==1?17:32) [bytes]
    framelength = 144*(bitrate/samplingfreq)+Padding_Bit [bytes]

    But first remember the current state of BitBucketPointerPut and BitBucketBitCounterPut, so
that the start
    of maindata of this frame can be calculated */

/*   PreviousBitBucketPointerPut=BitBucketPointerPut;
   PreviousBitBucketBitCounterPut=BitBucketBitCounterPut;          */

   printf(" bitrate = %d *1000 bits/s\n",bitrate_index);
   printf(" sampling_frequency = %d*100 Hz/s \n",sampling_frequency);
   printf(" padding_bit = %d bit\n",padding_bit);
   j=4+2*(1-protection_bit)+(nch==1?17:32);
   printf(" Header = %d bits\n",j);

   i=(((144*1000*bitrate_index)/sampling_frequency*100)+padding_bit) -
```

```
(4+2*(1-protection_bit)+(nch==1?17:32));
printf(" Main Audio Data Fram= %d Bits\n",i);
 break;

/*************************** Decode Scale Factor **************************/

 for (gr=0;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++)
          {
          sc1=scalefac_l[gr][ch][sfb];
          printf("scalefac_l = %d",sc1);

/*    GetBitBucket(Table_ScaleFac_CompressShort[1][scalefac_compress[gr][ch]][sfb]);  */
          }
         for (sfb=3;sfb<12;sfb++)
          {
          for (window=0;window<3;window++)
           {
          sc2=scalefac_s[gr][ch][sfb][window];
          printf("scalefac_s = %d",sc2);

 /*  GetBitBucket(Table_ScaleFac_CompressShort[2][scalefac_compress[gr][ch]][sfb]);   */
          }
```

```
    }
        for (window=0;window<3;window++)
         {
          scalefac_s[gr][ch][13][window]=1; /* @@@ is this correct? @@@ */
          printf("scalefac_s = 1");
         }
       }
      else
       {
        for (sfb=0;sfb<12;sfb++)
         {
          for (window=0;window<3;window++)
           {
           sc3=scalefac_s[gr][ch][sfb][window];
           printf("scalefac_s = %d",sc3);
        /*      GetBitBucket(Table_ScaleFac_CompressShort[0][scalefac_compress[gr][ch]]
[sfb]);          */
           }
         }
        for (window=0;window<3;window++)
         {
          scalefac_s[gr][ch][13][window]=1; /* @@@ is this correct? @@@ */
          printf("scalefac_s = 1");
         }
       }
     }
    else
     {
          if (gr==0)
        {
```

```
for (sfb=0;sfb<6;sfb++)
            {
             sc4=scalefac_l[gr][ch][sfb];
             printf("scalefac_l = %d",sc4);
        /*   GetBitBucket(Table_ScaleFac_Compress[scalefac_compress[gr][ch]][sfb]);   */
             }
           for (sfb=6;sfb<11;sfb++)
             {
             sc5=scalefac_l[gr][ch][sfb];
             printf("scalefac_l = %d",sc5);

        /*   GetBitBucket(Table_ScaleFac_Compress[scalefac_compress[gr][ch]][sfb]);   */
            }
           for (sfb=11;sfb<16;sfb++)
           {
            sc6=scalefac_l[gr][ch][sfb];
            printf("scalefac_l = %d",sc6);

        /*   GetBitBucket(Table_ScaleFac_Compress[scalefac_compress[gr][ch]][sfb]);   */
           }
          for (sfb=16;sfb<21;sfb++)
           {
            sc7=scalefac_l[gr][ch][sfb];
            printf("scalefac_l = %d",sc7);

        /*  GetBitBucket(Table_ScaleFac_Compress[scalefac_compress[gr][ch]][sfb]);   */
           }
          scalefac_l[gr][ch][22]=1; /* @@@ is this correct? @@@ */
        }
```

```
else
        {
        if (scfsi[ch][0]==0)
          {
          for (sfb=0;sfb<6;sfb++)
            {
            sc8=scalefac_l[1][ch][sfb];
            printf("scalefac_l = %d",sc8);

  /*   GetBitBucket(Table_ScaleFac_Compress[scalefac_compress[gr][ch]][sfb]);   */
            }
          }
        else
          {
          for (sfb=0;sfb<6;sfb++)
          {
          scalefac_l[1][ch][sfb]=scalefac_l[0][ch][sfb];
          }
        }
       if (scfsi[ch][1]==0)
        {
         for (sfb=6;sfb<11;sfb++)
          {
          sc9=scalefac_l[1][ch][sfb];
          printf("scalefac_l = %d",sc9);

  /*  GetBitBucket(Table_ScaleFac_Compress[scalefac_compress[gr][ch]][sfb]);   */
          }
        }
       else
```

```
{
            for (sfb=6;sfb<11;sfb++)
             {
              scalefac_l[1][ch][sfb]=scalefac_l[0][ch][sfb];
             }
           }
          if (scfsi[ch][2]==0)
           {
            for (sfb=11;sfb<16;sfb++)
             {
              sc10=scalefac_l[1][ch][sfb];
              printf("scalefac_l = %d",sc10);

/*    GetBitBucket(Table_ScaleFac_Compress[scalefac_compress[gr][ch]][sfb]);    */
             }
           }
          else
           {
            for (sfb=11;sfb<16;sfb++)
             {
              scalefac_l[1][ch][sfb]=scalefac_l[0][ch][sfb];
             }
           }
          if (scfsi[ch][3]==0)
           {
            for (sfb=16;sfb<21;sfb++)
             {
              sc11=scalefac_l[1][ch][sfb]=
              printf("scalefac_l = %d",sc11);
```

```
/*   GetBitBucket(Table_ScaleFac_Compress[scalefac_compress[gr][ch]][sfb]);    */
            }
          }
        else
          {
           for (sfb=16;sfb<21;sfb++)
            {
             scalefac_l[1][ch][sfb]=scalefac_l[0][ch][sfb];
            }
          }
       }
     }
      And now its time for decoding the Huffman data for this channel and this granule
     To do this, we must first set some variables for the Huffman decoder (what tables,
     what boundaries, where to put the data) */

     if (window_switching_flag[gr][ch])
      {
       if ((block_type[gr][ch]==2)&&(mixed_block_flag[gr][ch]==0))
        {
         HuffmanInput.reg0_bound=36;
        }
       else
        {
         HuffmanInput.reg0_bound=33; /* @@@ is this correct??? @@@ */
        }
       HuffmanInput.reg1_bound=576;
       /* no region 2 when short blocks */
      }
```

```
else
        {
         HuffmanInput.reg0_bound=Table_B8Long[Sampling_Frequency][region0_count[gr]
[ch]+1];
         HuffmanInput.reg1_bound=Table_B8Long[Sampling_Frequency][region0_count[gr]
[ch]+region1_count[gr][ch]+2];
        }
       if (HuffmanInput.reg0_bound>(big_values[gr][ch]<<1))
        {
         HuffmanInput.reg0_bound=
          HuffmanInput.reg1_bound=
           HuffmanInput.reg2_bound=big_values[gr][ch]<<1;
        }
       else {
        if (HuffmanInput.reg1_bound>(big_values[gr][ch]<<1))
         {
          HuffmanInput.reg1_bound=
           HuffmanInput.reg2_bound=big_values[gr][ch]<<1;
         }
        else {
         HuffmanInput.reg2_bound=big_values[gr][ch]<<1;
           }
         }
       HuffmanInput.table_select0=table_select[gr][ch][0];
       HuffmanInput.table_select1=table_select[gr][ch][1];
       HuffmanInput.table_select2=table_select[gr][ch][2];
i=(((PreviousBitBucketBitCounterPut<<5)+PreviousBitBucketBitCounterPut)+part2_3_length
[gr][ch])%8160;
```

```
    /* i=the position in the bitbucket where the current [gr][ch] ends. Counted in BITS */
 HuffmanInput.reg3_bound=((BitBucketValue+(i>>5))<<8)+(i%32);
    /* 24 MSB=adres of BitBucketPointerGet,LSB=bitcounter */


      HuffmanInput.table_select3=count1table_select[gr][ch];
      /*HuffmanInput.Spectrum=/* @@@@@????@@@@@ */


          HuffmanDecode(HuffmanInput);
     /*** and here the human data for [gr][ch] starts ***/
       }
   }
}
 } /* off leyer 3 */
    }   /* off switch  */
 return(0);
 } /* off decode_frame */

/*-------------------------------------------------------------*/
int main (void)
{
char filename_in[255]="test.mp3";
char filename_out[255]="testmp3.dec";
int i,j;
 clrscr();
 printf(" Program MP3 decoder For TMS320c31 by Electronic and Computer ID.ED23
KMITL\n\n");
 printf(" Only Layer 3 decoder                - version 1.00\n");
 printf(" Filename MP3 in : C:\test.mp3\n");
```

```
if ((filestream_in=fopen(filename_in,"r"))==NULL)
  {
   printf("Cannot open input file.\n");
   return(1);
  }
 if ((filestream_out=fopen(filename_out,"w"))==NULL)
  {
   printf("Cannot open output file.\n");
   return(1);
  }
 /****Show Header *****/
clrscr();
printf(" Header file have = \n");
for (j=0;j<=18;j++)
{
  for (i=0;i<=7;i++)
   {
    printf("%d",getbits(1));       /****** Show bits *******/
   }
  printf(" ");                     /****** Show byte *******/
 }
  decode_frame();
  fclose(filestream_in);
  fclose(filestream_out);
  printf(" \nComplete\n");         /******** Finish ********/
  return(0);
}
```

รูปที่ ค.16 โปรแกรมถอดรหัสส่วนหัวของสัญญาณเสียง MP3

```
;-------------------------------------------------------------------
; Define Port&Pin Name
;-------------------------------------------------------------------
LCD_EN                  BIT             P3.6    ; LCD Module Enable (Active High :
Level)
LCD_RS                  BIT             P3.7    ; LCD Module Register Select


;-------------------------------------------------------------------
; Define User Register
;-------------------------------------------------------------------
LCD_ADDR    EQU         030H    ; For keep LCD Address
LCD_DATA    EQU         031H    ; For keep LCD Data


;-------------------------------------------------------------------
; Main Program.
;-------------------------------------------------------------------
            ORG         0000H                   ; Reset Vector
            MOV         P0,#00000000B ; Clear Databus
            CLR         LCD_EN                  ; Clear LCD Enable
            CLR         LCD_RS                  ; Clear LCD RS
MAIN:       ACALLINIT_LCD                       ; Call LCD Initial subroutine
LOOP:       MOV         LCD_ADDR,#000H          ; Set Address 00H
            ACALLSET_ADDR_LCD       ;
            MOV         DPTR,#TITLE_1           ; Index Pointer ROM to Show LCD
            ACALLWRLINE_LCD         ; 00H-07H (Increase automatic)
            MOV         DPTR,#TITLE_3           ; Index Pointer ROM to Show LCD
            ACALLWRLINE_LCD         ; 08H-0FH (Increase automatic)
            MOV         LCD_ADDR,#040H          ; Set Address 40H
            ACALLSET_ADDR_LCD       ;
            MOV         DPTR,#TITLE_2           ; Index Pointer ROM to Show LCD
```

```
                ACALL WRLINE_LCD          ; 40H-47H (Increase automatic)
                MOV         DPTR,#TITLE_4     ; Index Pointer ROM to Show LCD
                ACALL WRLINE_LCD          ; 48H-4FH (Increase automatic)
                ACALL DELAY_1s            ; Delay
                ACALL DELAY_1s
                MOV         R4,#8             ; Set Loop 8 times
LOOP_LCD_L_SHF: ACALL   LCD_LSHF          ; Left Shift LCD Display
                ACALL DELAY_100ms         ; Delay
                ACALL DELAY_100ms
                DJNZ   R4,LOOP_LCD_L_SHF ; Do until 8 times
                MOV         LCD_ADDR,#000H    ; Set Address 00H
                ACALL SET_ADDR_LCD        ;
                MOV         DPTR,#TITLE_5     ; Index Pointer ROM to Show LCD
                ACALL WRLINE_LCD          ; 00H-07H (Increase automatic)

                MOV         LCD_ADDR,#040H    ; Set Address 40H
                ACALL SET_ADDR_LCD        ;
                MOV         DPTR,#TITLE_6     ; Index Pointer ROM to Show LCD
                ACALL WRLINE_LCD          ; 40H-47H (Increase automatic)

                ACALL DELAY_1s            ; Delay
                ACALL DELAY_1s

                MOV         R4,#8             ; Set Loop 8 times
LOOP_LCD_R_SHF: ACALL   LCD_RSHF          ; Right Shift LCD Display
                ACALL DELAY_100ms         ; Delay
                ACALL DELAY_100ms
                DJNZ   R4,LOOP_LCD_R_SHF ; Do until 8 times
                ACALL DELAY_1s            ; Delay
                ACALL DELAY_1s
```

```
ACALL LCD_CLR                    ; Clear LCD Display
MOV       LCD_ADDR,#000H         ; Set Address 00H
ACALL SET_ADDR_LCD
ACALL LCD_BLINK                  ; Blink Cursor
ACALL DELAY_1s

MOV       LCD_DATA,#'R'          ; Write Character 'R'
ACALL WRCHAR_LCD
ACALL LCD_BLINK                  ; Blink Cursor
ACALL DELAY_1s

MOV       LCD_DATA,#'e'          ; Write Character 'e'
ACALL WRCHAR_LCD
ACALL LCD_BLINK                  ; Blink Cursor
ACALL DELAY_1s                   ; Delay

MOV       LCD_DATA,#'a'          ; Write Character 'a'
ACALL WRCHAR_LCD
ACALL LCD_BLINK                  ; Blink Cursor
ACALL DELAY_1s                   ; Delay

MOV       LCD_DATA,#'d'          ; Write Character 'd'
ACALL WRCHAR_LCD
ACALL LCD_BLINK                  ; Blink Cursor
ACALL DELAY_1s                   ; Delay
MOV       LCD_DATA,#'y'          ; Write Character 'y'
ACALL WRCHAR_LCD
ACALL LCD_BLINK                  ; Blink Cursor
ACALL DELAY_1s                   ; Delay
```

```
MOV         LCD_DATA,#' ' ; Write Space
ACALL WRCHAR_LCD
MOV         LCD_DATA,#'T'       ; Write Character 'T'
ACALL WRCHAR_LCD
ACALL LCD_BLINK         ; Blink Cursor
ACALL DELAY_1s          ; Delay

MOV         LCD_DATA,#'o'       ; Write Character 'o'
ACALL WRCHAR_LCD

MOV         LCD_ADDR,#040H      ; Set Address 40H
ACALL SET_ADDR_LCD
ACALL LCD_BLINK         ; Blink Cursor
ACALL DELAY_1s          ; Delay

MOV         LCD_DATA,#' ' ; Write Space
ACALL WRCHAR_LCD

MOV         LCD_DATA,#'S'       ; Write Character 'S'
ACALL WRCHAR_LCD

MOV         LCD_DATA,#'t' ; Write Character 't'
ACALL WRCHAR_LCD
MOV         LCD_DATA,#'a'       ; Write Character 'a'
ACALL WRCHAR_LCD

MOV         LCD_DATA,#'r' ; Write Character 'r'
ACALL WRCHAR_LCD

MOV         LCD_DATA,#'t' ; Write Character 't'
```

```
                ACALL WRCHAR_LCD

                MOV             LCD_DATA,#' ' ; Write Space
                ACALL WRCHAR_LCD

                ACALL DELAY_1s              ; Delay

                ACALL LCD_OFF                       ; Off Display
                ACALL DELAY_1s              ; Delay

                ACALL LCD_ON                        ; On Display
                ACALL DELAY_1s              ; Delay

                ACALLLCD_OFF                        ; Off Display
                ACALL DELAY_1s              ; Delay

                ACALLLCD_ON                         ; On Display
                ACALL DELAY_1s              ; Delay
                ACALL DELAY_1s

                AJMP   LOOP                 ; Jump to loop
;-----------------------------------------------------------------
; LCD Initialize
;-----------------------------------------------------------------
INIT_LCD:       ACALL DELAY_100ms           ; Delay
                        CLR             LCD_RS                  ; Clear
LCD_RS Pin

                        MOV             P0,#00111000B ; 8bit Mode
                        ACALL LCD_CLK                   ; Pulse LCD Clock
```

```
                ACALL DELAY_10ms            ; Delay

                MOV         P0,#00111000B ; 8bit Mode
                ACALL LCD_CLK                   ; Pulse LCD Clock

                ACALL LCD_OFF                   ; Display Off

                ACALL LCD_CLR                   ; Clear Display

                MOV         P0,#00000110B ; Entry Mode
                ACALL LCD_CLK                   ; Pulse LCD Clock

                ACALL LCD_HOME          ; Return Home Display

;-----------------------------------------------------------
; LCD Clear Display
;-----------------------------------------------------------
LCD_CLR:    CLR         LCD_RS                  ; Clear LCD_RS Pin
            MOV         P0,#00000001B ; Display Clear
            ACALL LCD_CLK                   ; Pulse LCD Clock
            RET

;-----------------------------------------------------------------
; LCD Return Home
;-----------------------------------------------------------------
LCD_HOME:   CLR         LCD_RS                  ; Clear LCD_RS Pin
            MOV         P0,#00000010B ; Return Home
            ACALL LCD_CLK                   ; Pulse LCD Clock
            RET
```

```
;-----------------------------------------------------------------------
; LCD Display Off
;-----------------------------------------------------------------------
LCD_OFF:    CLR     LCD_RS                  ; Clear LCD_RS Pin
            MOV     P0,#00001000B ; Display Off
            ACALLLCD_CLK                    ; Pulse LCD Clock
            RET

;-----------------------------------------------------------------------
; LCD Clk
;-----------------------------------------------------------------------
LCD_CLK:    SETB  LCD_EN                    ; Pulse Clock to LCD_EN
            ACALLLCD_DELAY
            CLR     LCD_EN
            ACALLLCD_DELAY
            RET

;-----------------------------------------------------------------------
; LCD Display On
;-----------------------------------------------------------------------
LCD_ON:     CLR     LCD_RS                  ; Clear LCD_RS Pin
            MOV     P0,#00001100B ; Display On
            ACALLLCD_CLK                    ; Pulse LCD Clock
                    RET
;-----------------------------------------------------------------------
; LCD Cursor On
;-----------------------------------------------------------------------
LCD_BLINK:      CLR     LCD_RS                  ; Clear LCD_RS
Pin
                MOV     P0,#00001111B ; Display Cursor and Blink
```

```
                ACALL LCD_CLK                   ; Pulse LCD Clock
                RET

;-------------------------------------------------------------------
; LCD Left Shift Display
;-------------------------------------------------------------------
LCD_LSHF:       CLR     LCD_RS                  ; Clear LCD_RS Pin
                MOV     P0,#00011000B ; Left Shift Display
                ACALLLCD_CLK                    ; Pulse LCD Clock
                RET

;-------------------------------------------------------------------
; LCD Right Shift Display
;-------------------------------------------------------------------
LCD_RSHF:       CLR     LCD_RS                  ; Clear LCD_RS Pin
                MOV     P0,#00011100B ; Right Shift Display
                ACALLLCD_CLK                    ; Pulse LCD Clock
                RET

;-------------------------------------------------------------------
; Set LCD Address
; I/P:          LCD_ADDR
;-------------------------------------------------------------------
SET_ADDR_LCD:   CLR     LCD_RS                  ; Clear LCD_RS Pin
                MOV     A,LCD_ADDR      ; Move LCD_ADDR to ACC.
                SETB    ACC.7           ; Set bit ACC.7
                MOV     P0,A                    ; Move to DATABUS
                ACALLLCD_CLK                    ; Pulse LCD Clock
                RET
```

```
;-------------------------------------------------------------------------
; Write Character to show LCD
; I/P:          LCD_DATA
;-------------------------------------------------------------------------
WRCHAR_LCD:     SETB   LCD_RS                   ; Set LCD_RS Pin
                MOV    P0,LCD_DATA              ; Move LCD_DATA to
DATABUS
                ACALLLCD_CLK                    ; Pulse LCD Clock
                ACALLLCD_ON                     ; Display On
                RET

;-------------------------------------------------------------------
; Write Line of 8 Character from ROM
; I/P:          DPTR : Locate ROM Address
;-------------------------------------------------------------------
WRLINE_LCD:     MOV    R0,#0                    ; Clear loop counter
WRLINE_LCD_1:   SETB   LCD_RS                   ; Set LCD_RS Pin
                CLR    A                        ; Clear ACC.
                MOVC A,@A+DPTR                  ; Move data from @DPTR to ACC.
                MOV    P0,A                     ; Move ACC. to DATABUS
                ACALLLCD_CLK                    ; Pulse LCD Clock
                INC    DPTR                     ; Increase Pointer
                INC    R0                       ; Increase loop counter
                CJNE   R0,#8,WRLINE_LCD_1 ; Do until 8 times
                ACALLLCD_ON                     ; Display On
                RET
;-------------------------------------------------------------------------
; Dummy Delay time LCD_DELAY, 10m, 100m, 1s
;-------------------------------------------------------------------------
LCD_DELAY:      MOV    R7,#002                  ; Do 2 times
```

```
LCD_DELAY_2:    NOP
                NOP
                DJNZ   R6,LCD_DELAY_2
                DJNZ   R7,LCD_DELAY_1
                RET

DELAY_10ms:     MOV        R7,#010              ; Do 10 times
DELAY_10ms_1:   MOV        R6,#0E6H             ; Each loop = 1 ms
DELAY_10ms_2:   NOP
                NOP
                DJNZ   R6,DELAY_10ms_2
                DJNZ   R7,DELAY_10ms_1
                RET

DELAY_100ms:    MOV        R7,#100              ; Do 100 times
DELAY_100ms_1:  MOV        R6,#0E6H             ; Each loop = 1 ms
DELAY_100ms_2:  NOP
                NOP
                DJNZ   R6,DELAY_100ms_2
                DJNZ   R7,DELAY_100ms_1
                RET

DELAY_1s:       MOV        R5,#100              ; Do 100 times
DELAY_1s_1:     ACALL DELAY_10ms
                DJNZ   R5,DELAY_1s_1
                RET

;-----------------------------------------------------------------------
;Define Constant < Store in Flash EEPROM Program Memory >
;-----------------------------------------------------------------------
```

รูปที่ ค.14 โปรแกรมส่วนแสดงผล LCD 16 ตัวอักษร 1 บรรทัด

```
;                                   01234567
TITLE_1:        DB          'MP3 Deco'
TITLE_2:        DB          'der By    '

TITLE_3:        DB          'DSK TMS '
TITLE_4:        DB          '320C31  '

TITLE_5:        DB          'Mpeg 1  '
TITLE_6:        DB          'Layer 3
```

รูปที่ ค.17 โปรแกรมถอดรหัสส่วนหัวของสัญญาณเสียง MP3

ภาคผนวก ง
รายการอุปกรณ์

รายการอุปกรณ์

1. อุปกรณ์ภาคจ่ายไฟ

รายการอุปกรณ์	จำนวน
1.1 ไดโอด เบอร์ 1N4007	4 ตัว
1.2 LED สีแดง	2 ตัว
1.3 ตัวต้านทาน ค่า 330 โอห์ม	2 ตัว
1.4 IC#7805	1 ตัว
1.5 IC#7809	1 ตัว
1.6 Connector 2 Pin	4 ตัว
1.7 หม้อแปลงไฟฟ้า 500 mA 12V	1 ตัว
1.8 คาปาซิเตอร์ ชนิดอิเล็กโทรไลต์ 2200 µF/16V	2 ตัว
1.9 คาปาซิเตอร์ ชนิดอิเล็กโทรไลต์ 220 µF/16V	2 ตัว
1.10 คาปาซิเตอร์ ชนิดเซรามิค 0.1 µF	6 ตัว

2. อุปกรณ์วงจร Boot Loader

รายการอุปกรณ์	จำนวน
2.1 ไอซีเบอร์ 74LS32	1 ตัว
2.2 ไอซีเบอร์ 74139	1 ตัว
2.3 ไอซีเบอร์ 74LS08	1 ตัว
2.4 ไอซีเบอร์ 74LS04	1 ตัว
2.5 Flash Memory #AT29C010A	1 ตัว
2.6 ตัวต้านทาน ค่า 10 กิโลโอห์ม	5 ตัว
2.7 คาปาซิเตอร์ ชนิดเซรามิค 0.1 µF	8 ตัว
2.8 คาปาซิเตอร์ ชนิดอิเล็กโทรไลต์ 10 µF/16V	1 ตัว

รายการอุปกรณ์ (ต่อ)

2. อุปกรณ์วงจร Boot Loader (ต่อ)

รายการอุปกรณ์	จำนวน
2.9 Connector 34 ขา	3 ตัว
2.10 Socket 14 ขา	3 ตัว
2.11 Socket 16 ขา	1 ตัว
2.12 สวิตซ์	1 ตัว
2.13 Connector 2 ขา	1 ตัว
2.14 Jumper	4 ตัว
2.15 ไอซี 7474	3 ตัว

3. อุปกรณ์วงจรขยายหน่วยความจำภายนอก (RAM)

รายการอุปกรณ์	จำนวน
3.1 ไอซี 74LS32	1 ตัว
3.2 ไอซี 74LS08	1 ตัว
3.3 ไอซี 74LS04	1 ตัว
3.4 SRAM #K6T1008C2E	8 ตัว
3.5 Connector 34 ขา	3 ตัว
3.6 Socket 14 ขา	3 ตัว
3.7 Socket 16 ขา	1 ตัว
3.8 Connector 2 ขา	2 ตัว

ภาคผนวก จ

รายละเอียด และคุณสมบัติของอุปกรณ์

Features

- Fast Read Access Time – 70 ns
- 5-volt Only Reprogramming
- Sector Program Operation
  - Single Cycle Reprogram (Erase and Program)
  - 1024 Sectors (128 Bytes/sector)
  - Internal Address and Data Latches for 128 Bytes
- Two 8K Bytes Boot Blocks with Lockout
- Internal Program Control and Timer
- Hardware and Software Data Protection
- Fast Sector Program Cycle Time – 10 ms
- $\overline{\text{DATA}}$ Polling for End of Program Detection
- Low Power Dissipation
  - 50 mA Active Current
  - 100 µA CMOS Standby Current
- Typical Endurance > 10,000 Cycles
- Single 5V ± 10% Supply
- CMOS and TTL Compatible Inputs and Outputs
- Commercial and Industrial Temperature Ranges

1-Megabit (128K x 8) 5-volt Only Flash Memory

AT29C010A

Description

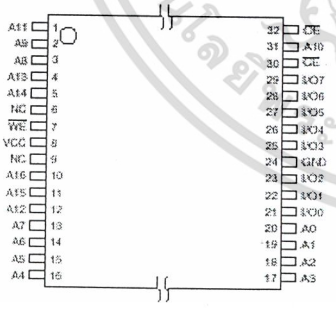
The AT29C010A is a 5-volt-only in-system Flash programmable and erasable read only memory (PEROM). Its 1 megabit of memory is organized as 131,072 words by 8 bits. Manufactured with Atmel's advanced nonvolatile CMOS technology, the device offers access times to 70 ns with power dissipation of just 275 mW over the commercial temperature range. When the device is deselected, the CMOS standby current is

Pin Configurations

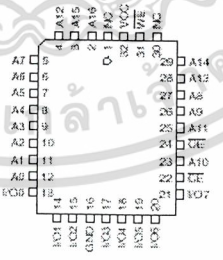
Pin Name	Function
A0 - A16	Addresses
$\overline{\text{CE}}$	Chip Enable
$\overline{\text{OE}}$	Output Enable
$\overline{\text{WE}}$	Write Enable
I/O0 - I/O7	Data Inputs/Outputs
NC	No Connect

DIP Top View

TSOP Top View
Type 1



PLCC Top View



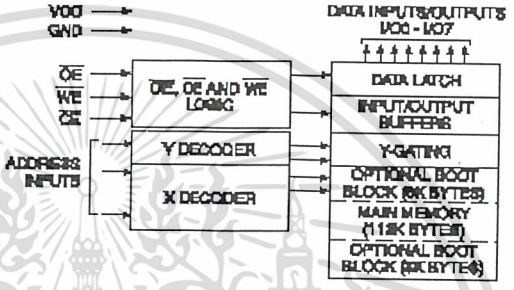
Rev. 0394D–FLASH–05/02

less than 100 µA. The device endurance is such that any sector can typically be written to in excess of 10,000 times.

To allow for simple in-system reprogrammability, the AT29C010A does not require high input voltages for programming. Five-volt-only commands determine the operation of the device. Reading data out of the device is similar to reading from an EPROM. Reprogramming the AT29C010A is performed on a sector basis; 128 bytes of data are loaded into the device and then simultaneously programmed.

During a reprogram cycle, the address locations and 128 bytes of data are internally latched, freeing the address and data bus for other operations. Following the initiation of a program cycle, the device will automatically erase the sector and then program the latched data using an internal control timer. The end of a program cycle can be detected by $\overline{\text{DATA}}$ polling of I/O7. Once the end of a program cycle has been detected, a new access for a read or program can begin.

Block Diagram



Device Operation

READ: The AT29C010A is accessed like an EPROM. When $\overline{\text{CE}}$ and $\overline{\text{OE}}$ are low and $\overline{\text{WE}}$ is high, the data stored at the memory location determined by the address pins is asserted on the outputs. The outputs are put in the high impedance state whenever $\overline{\text{CE}}$ or $\overline{\text{OE}}$ is high. This dual-line control gives designers flexibility in preventing bus contention.

BYTE LOAD: Byte loads are used to enter the 128 bytes of a sector to be programmed or the software codes for data protection. A byte load is performed by applying a low pulse on the $\overline{\text{WE}}$ or $\overline{\text{CE}}$ input with $\overline{\text{CE}}$ or $\overline{\text{WE}}$ low (respectively) and $\overline{\text{OE}}$ high. The address is latched on the falling edge of $\overline{\text{CE}}$ or $\overline{\text{WE}}$, whichever occurs last. The data is latched by the first rising edge of $\overline{\text{CE}}$ or $\overline{\text{WE}}$.

PROGRAM: The device is reprogrammed on a sector basis. If a byte of data within a sector is to be changed, data for the entire sector must be loaded into the device. The data in any byte that is not loaded during the programming of its sector will be indeterminate. Once the bytes of a sector are loaded into the device, they are simultaneously programmed during the internal programming period. After the first data byte has been loaded into the device, successive bytes are entered in the same manner. Each new byte to be programmed must have its high to low transition on $\overline{\text{WE}}$ (or $\overline{\text{CE}}$) within 150 µs of the low to high transition of $\overline{\text{WE}}$ (or $\overline{\text{CE}}$) of the preceding byte. If a high to low transition is not detected within 150 µs of the last low to high transition, the load period will end and the internal programming period will start. A7 to A16 specify the sector address.

The sector address must be valid during each high to low transition of $\overline{WE}$ (or $\overline{CE}$). A0 to A6 specify the byte address within the sector. The bytes may be loaded in any order; sequential loading is not required. Once a programming operation has been initiated, and for the duration of t_{WC}, a read operation will effectively be a polling operation.

SOFTWARE DATA PROTECTION: A software controlled data protection feature is available on the AT29C010A. Once the software protection is enabled a software algorithm must be issued to the device before a program may be performed. The software protection feature may be enabled or disabled by the user; when shipped from Atmel, the software data protection feature is disabled. To enable the software data protection, a series of three program commands to specific addresses with specific data must be performed. After the software data protection is enabled the same three program commands must begin each program cycle in order for the programs to occur. All software program commands must obey the sector program timing specifications. Once set, the software data protection feature remains active unless its disable command is issued. Power transitions will not reset the software data protection feature, however the software feature will guard against inadvertent program cycles during power transitions.

Once set, software data protection will remain active unless the disable command sequence is issued.

After setting SDP, any attempt to write to the device without the 3-byte command sequence will start the internal write timers. No data will be written to the device; however, for the duration of t_{WC}, a read operation will effectively be a polling operation.

After the software data protection's 3-byte command code is given, a byte load is performed by applying a low pulse on the $\overline{WE}$ or $\overline{CE}$ input with $\overline{CE}$ or $\overline{WE}$ low (respectively) and $\overline{OE}$ high. The address is latched on the falling edge of $\overline{CE}$ or $\overline{WE}$, whichever occurs last. The data is latched by the first rising edge of $\overline{CE}$ or $\overline{WE}$. The 128 bytes of data must be loaded into each sector by the same procedure as outlined in the program section under device operation.

HARDWARE DATA PROTECTION: Hardware features protect against inadvertent programs to the AT29C010A in the following ways: (a) V_{CC} sense – if V_{CC} is below 3.8V (typical), the program function is inhibited; (b) V_{CC} power on delay – once V_{CC} has reached the V_{CC} sense level, the device will automatically time out 5 ms (typical) before programming; (c) Program inhibit – holding any one of $\overline{OE}$ low, $\overline{CE}$ high or $\overline{WE}$ high inhibits program cycles; and (d) Noise filter—pulses of less than 15 ns (typical) on the $\overline{WE}$ or $\overline{CE}$ inputs will not initiate a program cycle.

PRODUCT IDENTIFICATION: The product identification mode identifies the device and manufacturer as Atmel. It may be accessed by hardware or software operation. The hardware operation mode can be used by an external programmer to identify the correct programming algorithm for the Atmel product. In addition, users may wish to use the software product identification mode to identify the part (i.e. using the device code), and have the system software use the appropriate sector size for program operations. In this manner, the user can have a common board design for 256K to 4-megabit densities and, with each density's sector size in a memory map, have the system software apply the appropriate sector size.
For details, see Operating Modes (for hardware operation) or Software Product Identification. The manufacturer and device code is the same for both modes.

$\overline{DATA}$ POLLING: The AT29C010A features $\overline{DATA}$ polling to indicate the end of a program cycle. During a program cycle an attempted read of the last byte loaded will result in the complement of the loaded data on I/O7. Once the program cycle has been completed, true data is valid on all outputs and the next cycle may begin. $\overline{DATA}$ polling may begin at any time during the program cycle.

TOGGLE BIT: In addition to $\overline{\text{DATA}}$ polling the AT29C010A provides another method for determining the end of a program or erase cycle. During a program or erase operation, successive attempts to read data from the device will result in I/O6 toggling between one and zero. Once the program cycle has completed, I/O6 will stop toggling and valid data will be read. Examining the toggle bit may begin at any time during a program cycle.

OPTIONAL CHIP ERASE MODE: The entire device can be erased by using a 6-byte software code. Please see Software Chip Erase application note for details.

BOOT BLOCK PROGRAMMING LOCKOUT: The AT29C010A has two designated memory blocks that have a programming lockout feature. This feature prevents programming of data in the designated block once the feature has been enabled. Each of these blocks consists of 8K bytes; the programming lockout feature can be set independently for either block. While the lockout feature does not have to be activated, it can be activated for either or both blocks.

These two 8K memory sections are referred to as *boot blocks*. Secure code which will bring up a system can be contained in a boot block. The AT29C010A blocks are located in the first 8K bytes of memory and the last 8K bytes of memory. The boot block programming lockout feature can therefore support systems that boot from the lower addresses of memory or the higher addresses. Once the programming lockout feature has been activated, the data in that block can no longer be erased or programmed; data in other memory locations can still be changed through the regular programming methods. To activate the lockout feature, a series of seven program commands to specific addresses with specific data must be performed. Please see Boot Block Lockout Feature Enable Algorithm.

If the boot block lockout feature has been activated on either block, the chip erase function will be disabled.

BOOT BLOCK LOCKOUT DETECTION: A software method is available to determine whether programming of either boot block section is locked out. See Software Product Identification Entry and Exit sections. When the device is in the software product identification mode, a read from location 00002 will show if programming the lower address boot block is locked out while reading location 1FFF2 will do so for the upper boot block. If the data is FE, the corresponding block can be programmed; if the data is FF, the program lockout feature has been activated and the corresponding block cannot be programmed. The software product identification exit mode should be used to return to standard operation.

Absolute Maximum Ratings*

Temperature Under Bias	-55°C to +125°C
Storage Temperature	-65°C to +150°C
All Input Voltages (including NC Pins) with Respect to Ground	-0.6V to +6.25V
All Output Voltages with Respect to Ground	-0.6V to V_{CC} + 0.6V
Voltage on $\overline{\text{OE}}$ with Respect to Ground	-0.6V to +13.5V

*NOTICE: Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

DC and AC Operating Range

		AT29C010A-70	AT29C010A-90	AT29C010A-12	AT29C010A-15
Operating Temperature (Case)	Com.	0°C - 70°C	0°C - 70°C	0°C - 70°C	0°C - 70°C
	Ind.		-40°C - 85°C	-40°C - 85°C	-40°C - 85°C
V_{CC} Power Supply		5V ± 5%	5V ± 10%	5V ± 10%	5V ± 10%

Note: Not recommended for New Designs.

Operating Modes

Mode	$\overline{CE}$	$\overline{OE}$	$\overline{WE}$	Ai	I/O
Read	V_{IL}	V_{IL}	V_{IH}	Ai	D_{OUT}
Program[2]	V_{IL}	V_{IH}	V_{IL}	Ai	D_{IN}
5V Chip Erase	V_{IL}	V_{IH}	V_{IL}	Ai	
Standby/Write Inhibit	V_{IH}	X[1]	X	X	High Z
Program Inhibit	X	X	V_{IH}		
Program Inhibit	X	V_{IL}	X		
Output Disable	X	V_{IH}	X		High Z
Product Identification					
Hardware	V_{IL}	V_{IL}	V_{IH}	A1 - A16 = V_{IL}, A9 = V_H,[3] A0 = V_{IL}	Manufacturer Code[4]
				A1 - A16 = V_{IL}, A9 = V_H,[3] A0 = V_{IH}	Device Code[4]
Software[5]				A0 = V_{IL}	Manufacturer Code[4]
				A0 = V_{IH}	Device Code[4]

Notes:
1. X can be V_{IL} or V_{IH}.
2. Refer to AC Programming Waveforms.
3. V_H = 12.0V ± 0.5V.
4. Manufacturer Code: 1F, Device Code: 5D.
5. See details under Software Product Identification Entry/Exit.

DC Characteristics

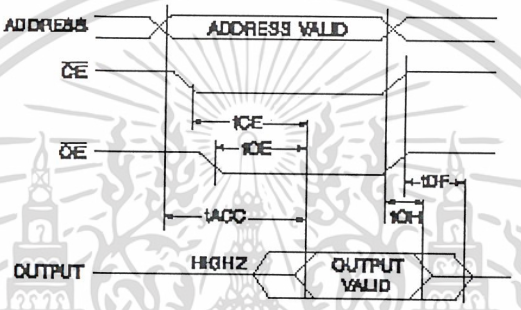
Symbol	Parameter	Condition		Min	Max	Units
I_{LI}	Input Load Current	V_{IN} = 0V to V_{CC}			10	µA
I_{LO}	Output Leakage Current	$V_{I/O}$ = 0V to V_{CC}			10	µA
I_{SB1}	V_{CC} Standby Current CMOS	$\overline{CE}$ = V_{CC} - 0.3V to V_{CC}	0° - 40°C		30	µA
			Com.		100	µA
			Ind.		300	µA
I_{SB2}	V_{CC} Standby Current TTL	$\overline{CE}$ = 2.0V to V_{CC}			3	mA
I_{CC}	V_{CC} Active Current	f = 5 MHz; I_{OUT} = 0 mA			50	mA
V_{IL}	Input Low Voltage				0.8	V
V_{IH}	Input High Voltage			2.0		V
V_{OL}	Output Low Voltage	I_{OL} = 2.1 mA			0.45	V
V_{OH1}	Output High Voltage	I_{OH} = -400 µA		2.4		V
V_{OH2}	Output High Voltage CMOS	I_{OH} = -100 µA; V_{CC} = 4.5V		4.2		V

AC Read Characteristics

Symbol	Parameter	AT29C010A-70		AT29C010A-90		AT29C010A-12		AT29C010A-15		Units
		Min	Max	Min	Max	Min	Max	Min	Max	
t_{ACC}	Address to Output Delay		70		90		120		150	ns
t_{CE}[1]	$\overline{CE}$ to Output Delay		70		90		120		150	ns
t_{OE}[2]	$\overline{OE}$ to Output Delay	0	35	0	40	0	50	0	70	ns
t_{DF}[3][4]	$\overline{CE}$ or $\overline{OE}$ to Output Float	0	25	0	25	0	30	0	40	ns
t_{OH}	Output Hold from $\overline{OE}$, $\overline{CE}$ or Address, whichever occurred first	0		0		0		0		ns

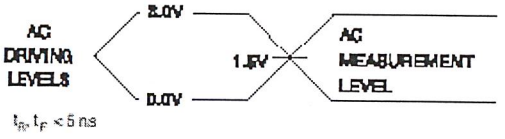
Note: Not recommended for New Designs.

AC Read Waveforms[1][2][3][4]



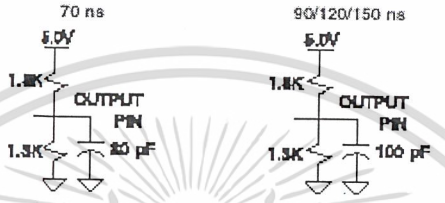
Notes:
1. $\overline{CE}$ may be delayed up to $t_{ACC} - t_{CE}$ after the address transition without impact on t_{ACC}.
2. $\overline{OE}$ may be delayed up to $t_{CE} - t_{OE}$ after the falling edge of $\overline{CE}$ without impact on t_{CE} or by $t_{ACC} - t_{OE}$ after an address change without impact on t_{ACC}.
3. t_{DF} is specified from $\overline{OE}$ or $\overline{CE}$ whichever occurs first (CL = 5 pF).
4. This parameter is characterized and is not 100% tested.

Input Test Waveforms and Measurement Level



$t_R, t_F < 5$ ns

Output Test Load



Pin Capacitance

f = 1 MHz, T = 25°C[1]

Symbol	Typ	Max	Units	Conditions
C_{IN}	4	6	pF	V_{IN} = 0V
C_{OUT}	8	12	pF	V_{OUT} = 0V

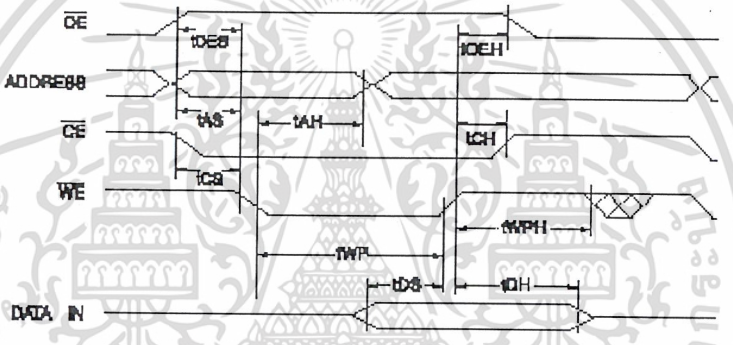
Note: 1. This parameter is canharacterized and is not 100% tested.

AC Byte Load Characteristics

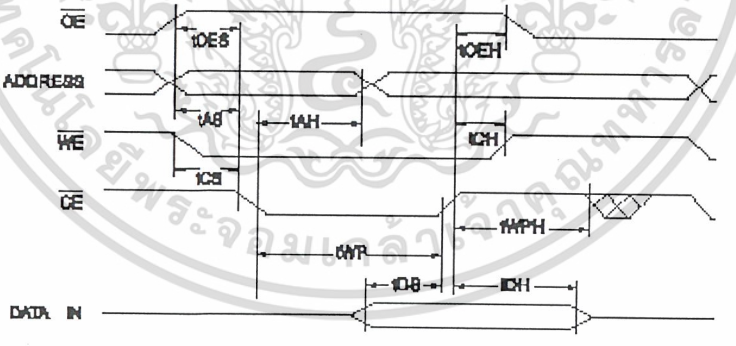
Symbol	Parameter	Min	Max	Units
t_{AS}, t_{OES}	Address, $\overline{OE}$ Set-up Time	0		ns
t_{AH}	Address Hold Time	50		ns
t_{CS}	Chip Select Set-up Time	0		ns
t_{CH}	Chip Select Hold Time	0		ns
t_{WP}	Write Pulse Width ($\overline{WE}$ or $\overline{CE}$)	90		ns
t_{DS}	Data Set-up Time	35		ns
t_{DH}, t_{OEH}	Data, $\overline{OE}$ Hold Time	0		ns
t_{WPH}	Write Pulse Width High	100		ns

AC Byte Load Waveforms

$\overline{WE}$ Controlled



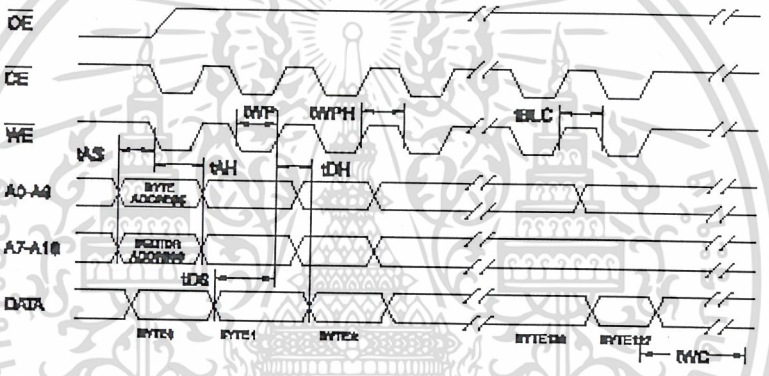
$\overline{CE}$ Controlled



Program Cycle Characteristics

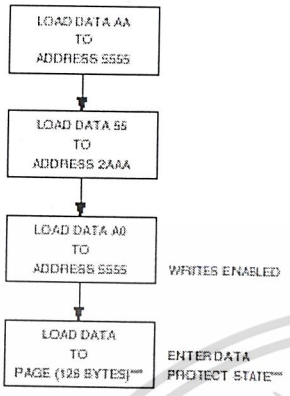
Symbol	Parameter	Min	Max	Units
t_{WC}	Write Cycle Time		10	ms
t_{AS}	Address Set-up Time	0		ns
t_{AH}	Address Hold Time	50		ns
t_{DS}	Data Set-up Time	35		ns
t_{DH}	Data Hold Time	0		ns
t_{WP}	Write Pulse Width	90		ns
t_{BLC}	Byte Load Cycle Time		150	µs
t_{WPH}	Write Pulse Width High	100		ns

Program Cycle Waveforms[1][2][3]

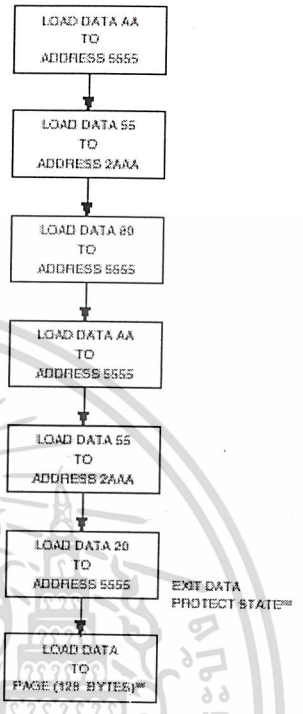


Notes:
1. A7 through A16 must specify the sector address during each high to low transition of $\overline{WE}$ (or $\overline{CE}$).
2. $\overline{OE}$ must be high when $\overline{WE}$ and $\overline{CE}$ are both low.
3. All bytes that are not loaded within the sector being programmed will be indeterminate.

Software Data Protection Enable Algorithm[1]

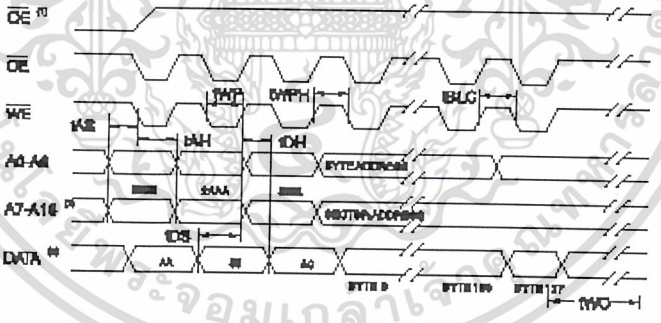


Software Data Protection Disable Algorithm[1]



Notes:
1. Data Format: I/O7 - I/O0 (Hex); Address Format: A14 - A0 (Hex).
2. Data Protect state will be activated at end of program cycle.
3. Data Protect state will be deactivated at end of program period.
4. 128 bytes of data MUST BE loaded.

Software Protected Program Cycle Waveform[1][2][3]



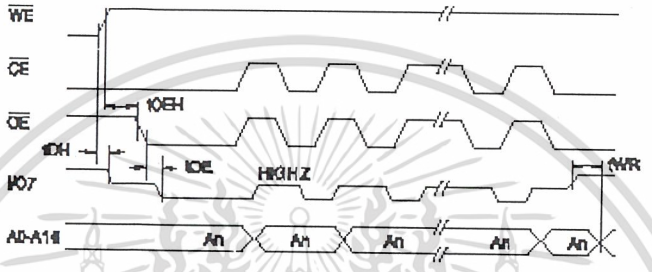
Notes:
1. A7 through A16 must specify the sector address during each high to low transition of $\overline{WE}$ (or $\overline{CE}$) after the software code has been entered.
2. $\overline{OE}$ must be high when $\overline{WE}$ and $\overline{CE}$ are both low.
3. All bytes that are not loaded within the sector being programmed will be indeterminate.

$\overline{\text{Data}}$ Polling Characteristics[1]

Symbol	Parameter	Min	Typ	Max	Units
t_{DH}	Data Hold Time	10			ns
t_{OEH}	$\overline{OE}$ Hold Time	10			ns
t_{OE}	$\overline{OE}$ to Output Delay[2]				ns
t_{WR}	Write Recovery Time	0			ns

Notes: 1. These parameters are characterized and not 100% tested.
2. See t_{OE} spec in AC Read Characteristics.

Data Polling Waveforms

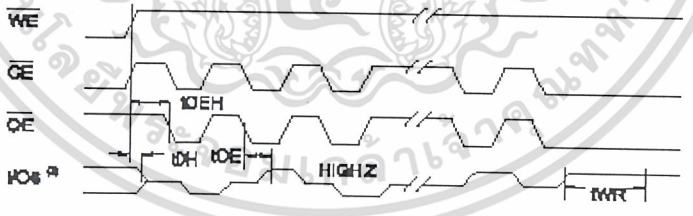


Toggle Bit Characteristics[1]

Symbol	Parameter	Min	Typ	Max	Units
t_{DH}	Data Hold Time	10			ns
t_{OEH}	$\overline{OE}$ Hold Time	10			ns
t_{OE}	$\overline{OE}$ to Output Delay[2]				ns
t_{OEHP}	$\overline{OE}$ High Pulse	150			ns
t_{WR}	Write Recovery Time	0			ns

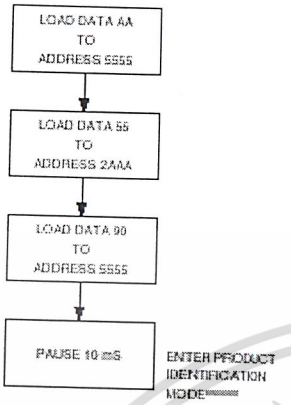
Notes: 1. These parameters are characterized and not 100% tested.
2. See t_{OE} spec in AC Read Characteristics.

Toggle Bit Waveforms[1][2][3]

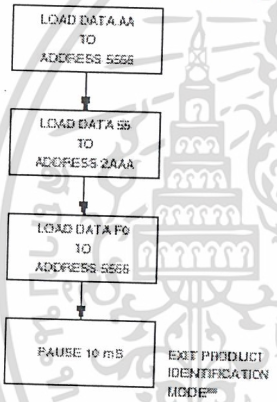


Notes: 1. Toggling either $\overline{OE}$ or $\overline{CE}$ or both $\overline{OE}$ and $\overline{CE}$ will operate toggle bit.
2. Beginning and ending state of I/O6 will vary.
3. Any address location may be used but the address should not vary.

Software Product Identification Entry[1]

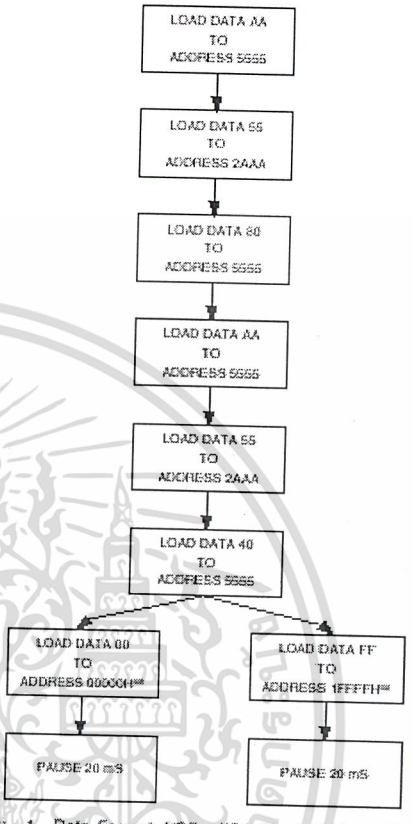


Software Product Identification Exit[1]

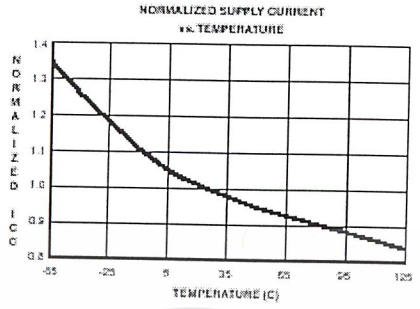


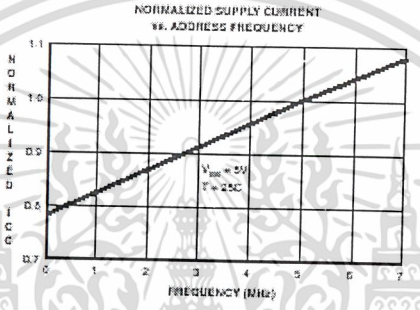
Notes:
1. Data Format: I/O7 - I/O0 (Hex); Address Format: A14 - A0 (Hex).
2. A1 - A15 = V_{IL}.
   Manufacturer Code is read for A0 = V_{IL}; Device Code is read for A0 = V_{IH}.
3. The device does not remain in identification mode if powered down.
4. The device returns to standard operation mode.
5. Manufacturer Code is 1F. The Device Code is D5.

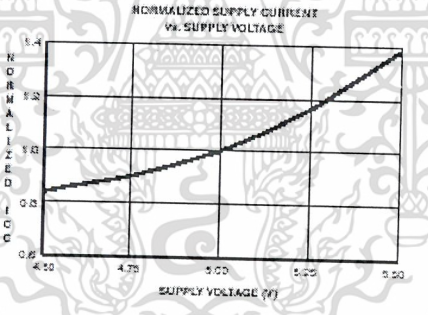
Boot Block Lockout Feature Enable Algorithm[1]



Notes:
1. Data Format: I/O7 - I/O0 (Hex); Address Format: A14 - A0 (Hex).
2. Lockout feature set on lower address boot block.
3. Lockout feature set on higher address boot block.







Ordering Information

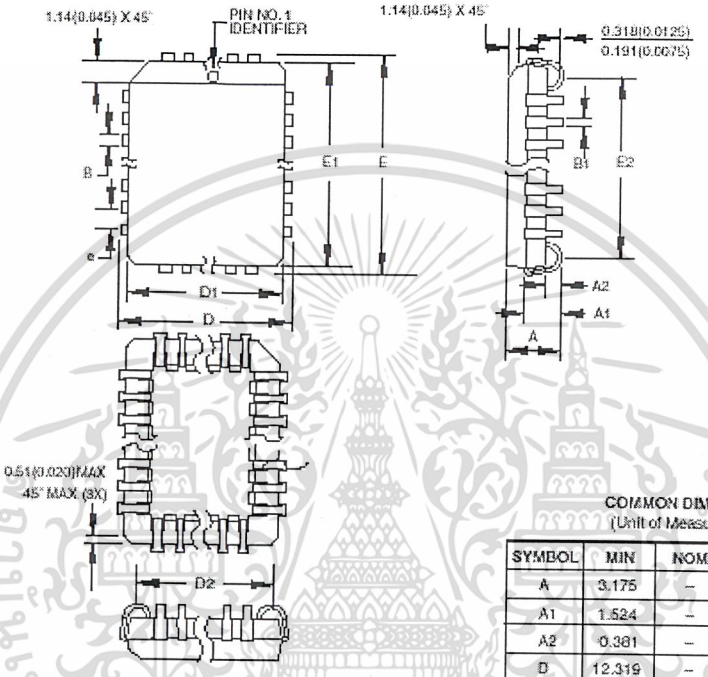
t_{ACC} (ns)	I_{CC} (mA)		Ordering Code	Package	Operation Range
	Active	Standby			
70	50	0.1	AT29C010A-70JC AT29C010A-70PC AT29C010A-70TC	32J 32P6 32T	Commercial (0° to 70°C)
90	50	0.1	AT29C010A-90JC AT29C010A-90PC AT29C010A-90TC	32J 32P6 32T	Commercial (0° to 70°C)
	50	0.3	AT29C010A-90JI AT29C010A-90PI AT29C010A-90TI	32J 32P6 32T	Industrial (-40° to 85°C)
120	50	0.1	AT29C010A-12JC AT29C010A-12PC AT29C010A-12TC	32J 32P6 32T	Commercial (0° to 70°C)
	50	0.3	AT29C010A-12JI AT29C010A-12PI AT29C010A-12TI	32J 32P6 32T	Industrial (-40° to 85°C)
150	50	0.1	AT29C010A-15JC AT29C010A-15PC AT29C010A-15TC	32J 32P6 32T	Commercial (0° to 70°C)
	50	0.3	AT29C010A-15JI AT29C010A-15PI AT29C010A-15TI	32J 32P6 32T	Industrial (-40° to 85°C)

Note: Not recommended for New Designs.

Package Type	
32J	32-lead, Plastic J-leaded Chip Carrier (PLCC)
32P6	32-lead, 0.600" Wide, Plastic Dual Inline Package (PDIP)
32T	32-lead, Thin Small Outline Package (TSOP)

Packaging Information

32J – PLCC



COMMON DIMENSIONS
(Unit of Measure – mm)

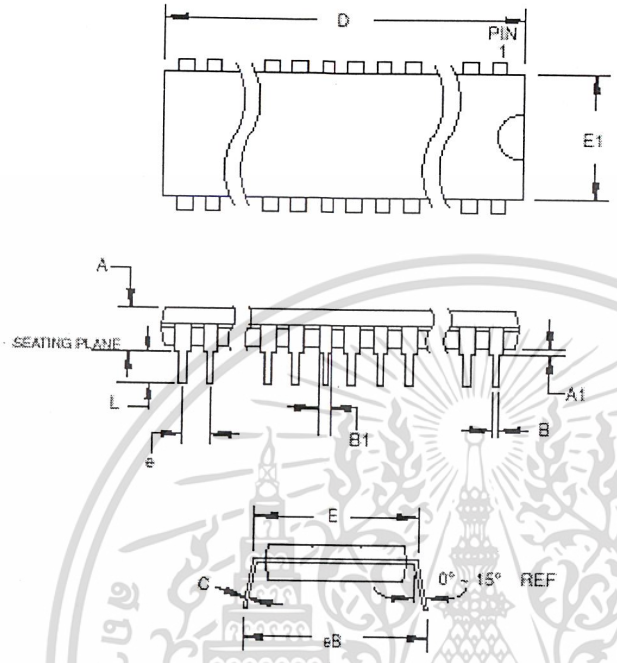
SYMBOL	MIN	NOM	MAX	NOTE
A	3.175	–	3.556	
A1	1.524	–	2.413	
A2	0.381	–	–	
D	12.319	–	12.573	
D1	11.354	–	11.506	Note 2
D2	9.906	–	10.922	
E	14.859	–	15.113	
E1	13.894	–	14.046	Note 2
E2	12.471	–	13.487	
B	0.660	–	0.813	
B1	0.330	–	0.533	
e	1.270 TYP			

Notes:
1. This package conforms to JEDEC reference MS-016, Variation AE.
2. Dimensions D1 and E1 do not include mold protrusion. Allowable protrusion is .010"(0.254 mm) per side. Dimension D1 and E1 include mold mismatch and are measured at the extreme material condition at the upper or lower parting line.
3. Lead coplanarity is 0.004" (0.102 mm) maximum.

10/04/01

	TITLE	DRAWING NO.	REV.
2325 Orchard Parkway San Jose, CA 95131	32J, 32-lead, Plastic J-leaded Chip Carrier (PLCC)	32J	B

32P6 – PDIP



COMMON DIMENSIONS
(Unit of Measure – mm)

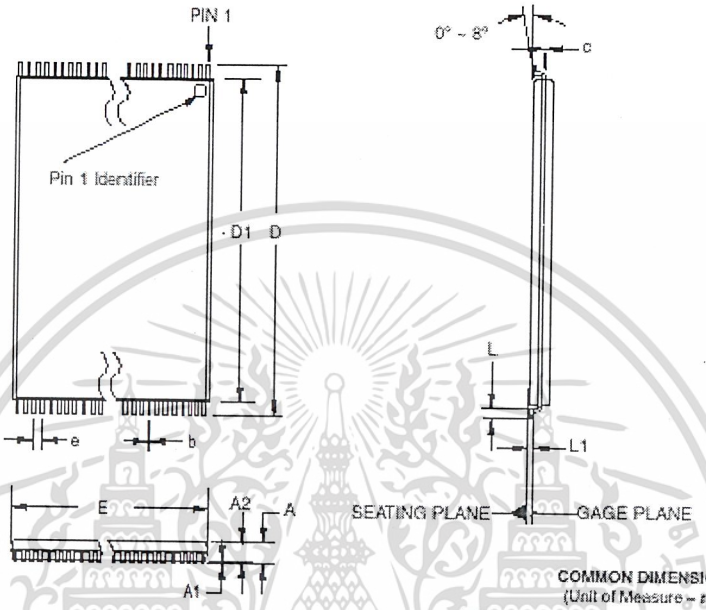
SYMBOL	MIN	NOM	MAX	NOTE
A	–	–	4.826	
A1	0.381	–	–	
D	41.783	–	42.291	Note 1
E	15.240	–	15.875	
E1	13.462	–	13.970	Note 1
B	0.356	–	0.559	
B1	1.041	–	1.651	
L	3.048	–	3.556	
C	0.203	–	0.381	
eB	15.494	–	17.526	
e	2.540 TYP			

Note: 1. Dimensions D and E1 do not include mold Flash or Protrusion. Mold Flash or Protrusion shall not exceed 0.25 mm (0.010").

09/28/01

	TITLE	DRAWING NO.	REV.
2325 Orchard Parkway San Jose, CA 95131	32P6, 32-lead (0.600"/15.24 mm Wide) Plastic Dual Inline Package (PDIP)	32P6	B

32T – TSOP



COMMON DIMENSIONS
(Unit of Measure – mm)

SYMBOL	MIN	NOM	MAX	NOTE
A	—	—	1.20	
A1	0.05	—	0.15	
A2	0.95	1.00	1.05	
D	19.80	20.00	20.20	
D1	18.30	18.40	18.50	Note 2
E	7.90	8.00	8.10	Note 2
L	0.50	0.60	0.70	
L1	0.25 BASIC			
b	0.17	0.22	0.27	
c	0.10	—	0.21	
e	0.50 BASIC			

Notes:
1. This package conforms to JEDEC reference MO-142, Variation BD.
2. Dimensions D1 and E do not include mold protrusion. Allowable protrusion on E is 0.15 mm per side and on D1 is 0.25 mm per side.
3. Lead coplanarity is 0.10 mm maximum.

10/18/01

2325 Orchard Parkway San Jose, CA 95131	TITLE 32T, 32-lead (8 x 20 mm Package) Plastic Thin Small Outline Package, Type I (TSOP)	DRAWING NO. 32T	REV. B

Atmel Headquarters

Corporate Headquarters
2325 Orchard Parkway
San Jose, CA 95131
TEL 1(408) 441-0311
FAX 1(408) 487-2600

Europe
Atmel Sarl
Route des Arsenaux 41
Case Postale 80
CH-1705 Fribourg
Switzerland
TEL (41) 26-426-5555
FAX (41) 26-426-5500

Asia
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
TEL (852) 2721-9778
FAX (852) 2722-1369

Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
TEL (81) 3-3523-3551
FAX (81) 3-3523-7581

Atmel Operations

Memory
2325 Orchard Parkway
San Jose, CA 95131
TEL 1(408) 441-0311
FAX 1(408) 436-4314

Microcontrollers
2325 Orchard Parkway
San Jose, CA 95131
TEL 1(408) 441-0311
FAX 1(408) 436-4314

La Chantrerie
BP 70602
44306 Nantes Cedex 3, France
TEL (33) 2-40-18-18-18
FAX (33) 2-40-18-19-60

ASIC/ASSP/Smart Cards
Zone Industrielle
13106 Rousset Cedex, France
TEL (33) 4-42-53-60-00
FAX (33) 4-42-53-60-01

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906
TEL 1(719) 576-3300
FAX 1(719) 540-1759

Scottish Enterprise Technology Park
Maxwell Building
East Kilbride G75 0QR, Scotland
TEL (44) 1355-803-000
FAX (44) 1355-242-743

RF/Automotive
Theresienstrasse 2
Postfach 3535
74025 Heilbronn, Germany
TEL (49) 71-31-67-0
FAX (49) 71-31-67-2340

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906
TEL 1(719) 576-3300
FAX 1(719) 540-1759

Biometrics/Imaging/Hi-Rel MPU/
High Speed Converters/RF Datacom
Avenue de Rochepleine
BP 123
38521 Saint-Egreve Cedex, France
TEL (33) 4-76-58-30-00
FAX (33) 4-76-58-34-80

e-mail
literature@atmel.com

Web Site
http://www.atmel.com

0394D–FLASH–05/02 xM

Document Title

128Kx8 bit Low Power CMOS Static RAM

Revision History

Revision No.	History	Draft Data	Remark
0.0	Design target	October 12, 1998	Preliminary
1.0	Finalize - Improve twr form 55ns to 50ns for 70ns product. - Remove 55ns speed bin from industrial product.	August 30, 1999	Final
1.01	Errata correction	December 1, 1999	
2.0	Revise	February 14, 2000	Final
3.0	Revise - Add 55ns parts to industrial products.	March 3, 2000	Final

128Kx8 bit Low Power CMOS Static RAM

FEATURES

- Process Technology: TFT
- Organization: 128Kx8
- Power Supply Voltage: 4.5~5.5V
- Low Data Retention Voltage: 2V(Min)
- Three state output and TTL Compatible
- Package Type: 32-DIP-600, 32-SOP-525, 32-TSOP1-0820F/R

GENERAL DESCRIPTION

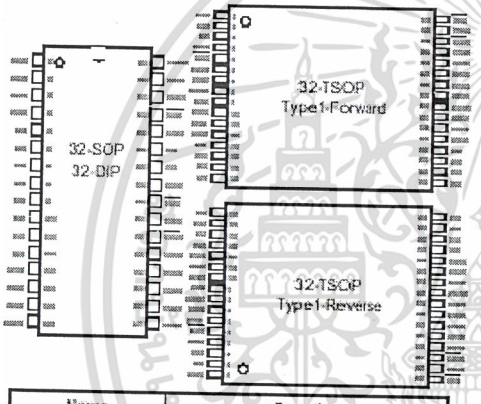
The K6T1008C2E families are fabricated by SAMSUNG's advanced CMOS process technology. The families support various operating temperature ranges and have various package types for user flexibility of system design. The families also support low data retention voltage for battery back-up operation with low data retention current.

PRODUCT FAMILY

Product Family	Operating Temperature	Vcc Range	Speed	Power Dissipation		PKG Type
				Standby (Isb1, Max)	Operating (Icc2, Max)	
K6T1008C2E-L	Commercial(0~70°C)	4.5~5.5V	55*/70ns	50μA	50mA	32-DIP-600, 32-SOP-525 32-TSOP1-0820F/R
K6T1008C2E-B				10μA		
K6T1008C2E-P	Industrial(-40~85°C)			50μA		32-SOP -525 32-TSOP1-0820F/R
K6T1008C2E-F				15μA		

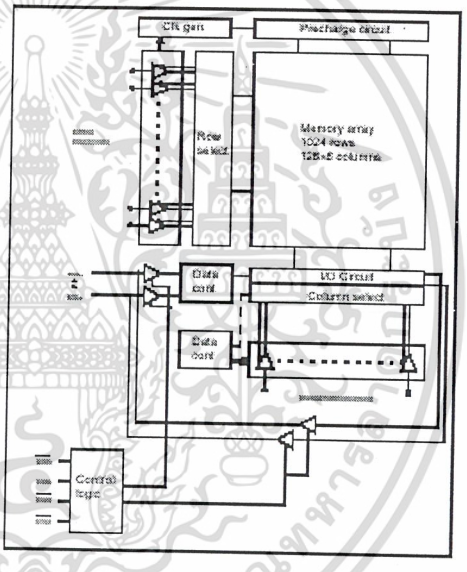
1. The parameters are tested with 50pF test load

PIN DESCRIPTION



Name	Function
$\overline{CS_1}$, CS_2	Chip Select Input
$\overline{OE}$	Output Enable Input
$\overline{WE}$	Write Enable Input
I/O_1~I/O_8	Data Inputs/Outputs
A_0~A_{16}	Address Inputs
Vcc	Power
Vss	Ground
N.C.	No Connection

FUNCTIONAL BLOCK DIAGRAM



SAMSUNG ELECTRONICS CO., LTD. reserves the right to change products and specifications without notice.

PRODUCT LIST

Commercial Temperature Products(0~70°C)		Industrial Temperature Products(-40~85°C)	
Part Name	Function	Part Name	Function
K6T1008C2E-DL55	32-DIP, 55ns, Low Power	K6T1008C2E-GP55	32-SOP, 55ns, Low Power
K6T1008C2E-DL70	32-DIP, 70ns, Low Power	K6T1008C2E-GP70	32-SOP, 70ns, Low Power
K6T1008C2E-DB55	32-DIP, 55ns, Low Low Power	K6T1008C2E-GF55	32-SOP, 55ns, Low Low Power
K6T1008C2E-DB70	32-DIP, 70ns, Low Low Power	K6T1008C2E-GF70	32-SOP, 70ns, Low Low Power
K6T1008C2E-GL55	32-SOP, 55ns, Low Power	K6T1008C2E-TF55	32-TSOP F, 55ns, Low Low Power
K6T1008C2E-GL70	32-SOP, 70ns, Low Power	K6T1008C2E-TF70	32-TSOP F, 70ns, Low Low Power
K6T1008C2E-GB55	32-SOP, 55ns, Low Low Power	K6T1008C2E-RF55	32-TSOP R, 55ns, Low Low Power
K6T1008C2E-GB70	32-SOP, 70ns, Low Low Power	K6T1008C2E-RF70	32-TSOP R, 70ns, Low Low Power
K6T1008C2E-TB55	32-TSOP F, 55ns, Low Low Power		
K6T1008C2E-TB70	32-TSOP F, 70ns, Low Low Power		
K6T1008C2E-RB55	32-TSOP R, 55ns, Low Low Power		
K6T1008C2E-RB70	32-TSOP R, 70ns, Low Low Power		

FUNCTIONAL DESCRIPTION

$\overline{CS_1}$	CS_2	$\overline{OE}$	$\overline{WE}$	I/O	Mode	Power
H	X[1]	X[1]	X[1]	High-Z	Deselected	Standby
X[1]	L	X[1]	X[1]	High-Z	Deselected	Standby
L	H	H	H	High-Z	Output Disabled	Active
L	H	L	H	Dout	Read	Active
L	H	X[1]	L	Din	Write	Active

1. X means don't care (Must be in high or low states)

ABSOLUTE MAXIMUM RATINGS[1]

Item	Symbol	Ratings	Unit	Remark
Voltage on any pin relative to Vss	V_{IN},V_{OUT}	-0.5 to 7.0	V	-
Voltage on Vcc supply relative to Vss	V_{CC}	-0.5 to 7.0	V	-
Power Dissipation	P_D	1.0	W	-
Storage temperature	T_{STG}	-65 to 150	°C	-
Operating Temperature	T_A	0 to 70	°C	K6T1008C2E-L/-B
		-40 to 85	°C	K6T1008C2E-P/-F

1. Stresses greater than those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. Functional operation should be restricted to recommended operating condition. Exposure to absolute maximum rating conditions for extended periods may affect reliability.

RECOMMENDED DC OPERATING CONDITIONS[1]

Item	Symbol	Product	Min	Typ	Max	Unit
Supply voltage	V_{CC}	K6T1008C2E Family	4.5	5.0	5.5	V
Ground	V_{SS}	All Family	0	0	0	V
Input high voltage	V_{IH}	K6T1008C2E Family	2.2	-	V_{CC}+0.5[2]	V
Input low voltage	V_{IL}	K6T1008C2E Family	-0.5[3]	-	0.8	V

Note:
1. Commercial Product: T_A=0 to 70°C
   Industrial Product: T_A=-40 to 85°C, otherwise specified.
2. Overshoot: V_{CC}+3.0V in case of pulse width≤30ns.
3. Undershoot: -3.0V in case of pulse width≤30ns.
4. Overshoot and undershoot are sampled, not 100% tested.

CAPACITANCE[1] (f=1MHz, TA=25°C)

Item	Symbol	Test Condition	Min	Max	Unit
Input capacitance	C_{IN}	V_{IN}=0V	-	6	pF
Input/Output capacitance	C_{IO}	V_{IO}=0V	-	8	pF

1. Capacitance is sampled, not 100% tested

DC AND OPERATING CHARACTERISTICS

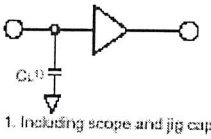
Item	Symbol	Test Conditions	Min	Typ	Max	Unit
Input leakage current	I_{LI}	V_{IN}=Vss to V_{CC}	-1	-	1	μA
Output leakage current	I_{LO}	$\overline{CS_1}$=V_{IH} or CS_2=V_{IL} or $\overline{OE}$=V_{IH} or $\overline{WE}$=V_{IL}, V_{IO}=Vss to V_{CC}	-1	-	1	μA
Operating power supply current	I_{CC}	I_{IO}=0mA, $\overline{CS_1}$=V_{IL}, CS_2=V_{IH}, V_{IN}=V_{IH} or V_{IL}, Read	-	-	10	mA
Average operating current	I_{CC1}	Cycle time=1μs, 100%duty, I_{IO}=0mA, $\overline{CS_1}$≤0.2V, CS_2≥V_{CC}-0.2V, V_{IN}≤0.2V or V_{IN}≥V_{CC}-0.2V	-	-	7	mA
	I_{CC2}	Cycle time=Min, 100% duty, I_{IO}=0mA, $\overline{CS_1}$=V_{IL}, CS_2=V_{IH}, V_{IN}=V_{IL} or V_{IH}	-	-	50	mA
Output low voltage	V_{OL}	I_{OL}=2.1mA	-	-	0.4	V
Output high voltage	V_{OH}	I_{OH}=-1.0mA	2.4	-	-	V
Standby Current(TTL)	I_{SB}	$\overline{CS_1}$=V_{IH}, CS_2=V_{IL}, Other inputs=V_{IH} or V_{IL}	-	-	3	mA
Standby Current(CMOS)	I_{SB1}	$\overline{CS_1}$≥V_{CC}-0.2V, CS_2≥V_{CC}-0.2V or CS_2≤0.2V, Other inputs=0~V_{CC}	-	-	50[1]	μA

1. 50μA for Low power product. In case of Low Low power products are comercial=10μA, industrial=15μA

AC OPERATING CONDITIONS

TEST CONDITIONS(Test Load and Input/Output Reference)

Input pulse level: 0.8 to 2.4V
Input rising and falling time: 5ns
Input and output reference voltage: 1.5V
Output load(see right): C_L=100pF+1TTL
C_L=50pF+1TTL



1. Including scope and jig capacitance

AC CHARACTERISTICS (V_{CC}=4.5~5.5V, Commercial Product: T_A=0 to 70°C, Industrial Product: T_A=-40 to 85°C)

Parameter List		Symbol	Speed Bins				Units
			55ns		70ns		
			Min	Max	Min	Max	
Read	Read Cycle Time	t_{RC}	55	-	70	-	ns
	Address Access Time	t_{AA}	-	55	-	70	ns
	Chip Select to Output	t_{CO}	-	55	-	70	ns
	Output Enable to Valid Output	t_{OE}	-	25	-	35	ns
	Chip Select to Low-Z Output	t_{LZ}	10	-	10	-	ns
	Output Enable to Low-Z Output	t_{OLZ}	5	-	5	-	ns
	Chip Disable to High-Z Output	t_{HZ}	0	20	0	25	ns
	Output Disable to High-Z Output	t_{OHZ}	0	20	0	25	ns
	Output Hold from Address Change	t_{OH}	10	-	10	-	ns
Write	Write Cycle Time	t_{WC}	55	-	70	-	ns
	Chip Select to End of Write	t_{CW}	45	-	60	-	ns
	Address Set-up Time	t_{AS}	0	-	0	-	ns
	Address Valid to End of Write	t_{AW}	45	-	60	-	ns
	Write Pulse Width	t_{WP}	40	-	50	-	ns
	Write Recovery Time	t_{WR}	0	-	0	-	ns
	Write to Output High-Z	t_{WHZ}	0	20	0	25	ns
	Data to Write Time Overlap	t_{DW}	20	-	25	-	ns
	Data Hold from Write Time	t_{DH}	0	-	0	-	ns
	End Write to Output Low-Z	t_{OW}	5	-	5	-	ns

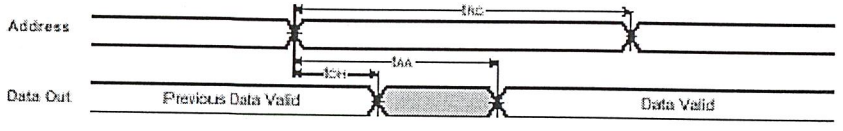
DATA RETENTION CHARACTERISTICS

Item	Symbol	Test Condition		Min	Typ	Max	Unit
V_{CC} for data retention	V_{DR}	$\overline{CS}_1 \geq V_{CC}-0.2V$[1]		2.0	-	5.5	V
Data retention current	I_{DR}	V_{CC}=3.0V, $\overline{CS}_1 \geq V_{CC}-0.2V$[1]	K6T1008C2E-L	-	-	20	µA
			K6T1008C2E-B	-	-	10	
			K6T1008C2E-P	-	-	25	
			K6T1008C2E-F	-	-	10	
Data retention set-up time	t_{SDR}	See data retention waveform		0	-	-	ms
Recovery time	t_{RDR}			5	-	-	

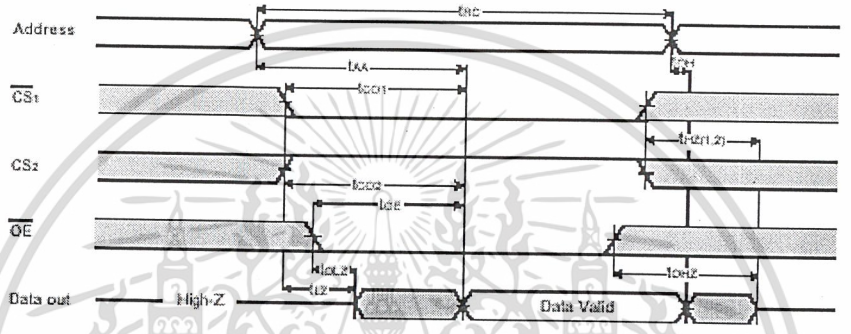
1. $\overline{CS}_1 \geq V_{CC}-0.2V$, $CS_2 \geq V_{CC}-0.2V$($\overline{CS}_1$ controlled) or $CS_2 \leq 0.2V$(CS_2 controlled)

TIMMING DIAGRAMS

TIMING WAVEFORM OF READ CYCLE(1) (Address Controlled, $\overline{CS1}$=$\overline{OE}$=V_{IL}, CS2=$\overline{WE}$=V_{IH})



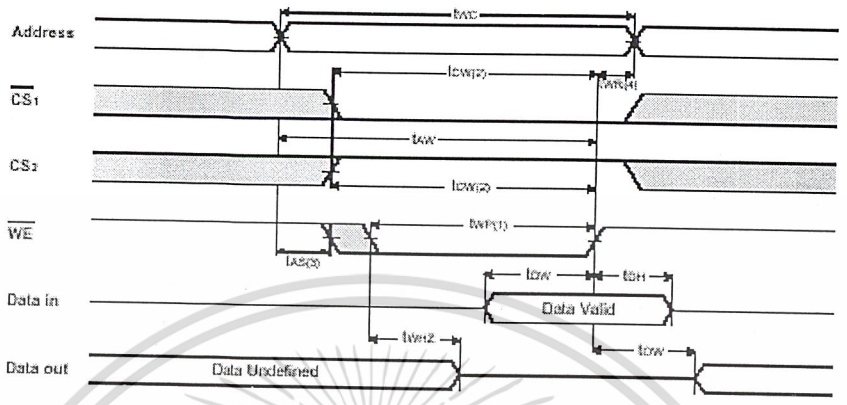
TIMING WAVEFORM OF READ CYCLE(2) ($\overline{WE}$=V_{IH})



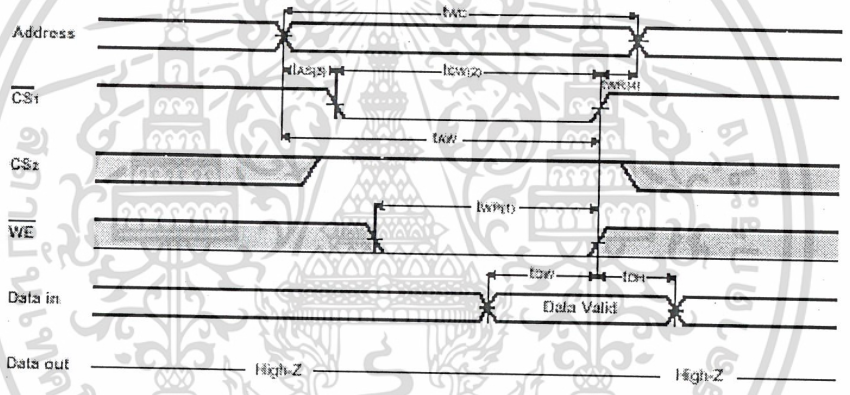
NOTES (READ CYCLE)

1. tHZ and tOHZ are defined as the time at which the outputs achieve the open circuit conditions and are not referenced to output voltage levels.
2. At any given temperature and voltage condition, tHZ(Max.) is less than tLZ(Min.) both for a given device and from device to device interconnection.

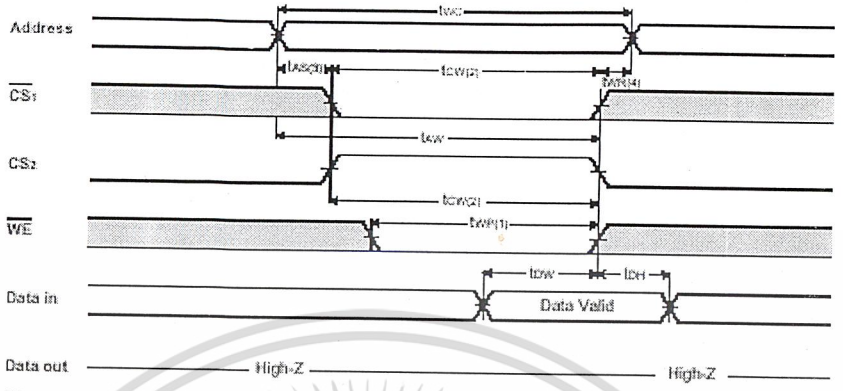
TIMING WAVEFORM OF WRITE CYCLE(1) ($\overline{WE}$ Controlled)



TIMING WAVEFORM OF WRITE CYCLE(2) ($\overline{CS_{1,2}}$ Controlled)



TIMING WAVEFORM OF WRITE CYCLE(3) ($\overline{CS_1}$ Controlled)

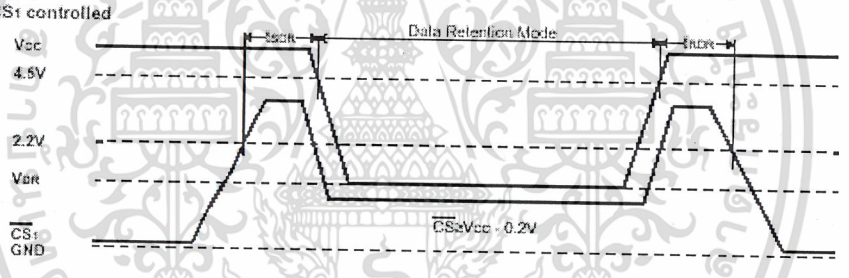


NOTES (WRITE CYCLE)

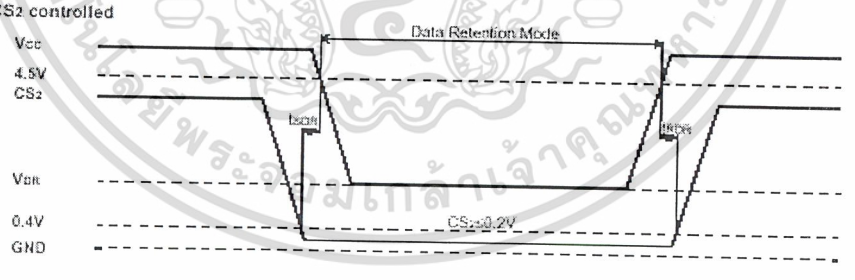
1. A write occurs during the overlap of a low $\overline{CS_1}$, a high CS_2 and a low $\overline{WE}$. A write begins at the latest transition among $\overline{CS_1}$ goes low, CS_2 going high and $\overline{WE}$ going low. A write end at the earliest transition among $\overline{CS_1}$ going high, CS_2 going low and $\overline{WE}$ going high. t_{WP} is measured from the beginning of write to the end of write.
2. t_{CW} is measured from the $\overline{CS_1}$ going low or CS_2 going high to the end of write.
3. t_{AS} is measured from the address valid to the beginning of write.
4. t_{WR} is measured from the end of write to the address change. t_{WR2} applied in case a write ends as $\overline{CS_1}$ or $\overline{WE}$ going high t_{WR} applied in case a write ends as CS_2 going to low.

DATA RETENTION WAVE FORM

$\overline{CS_1}$ controlled



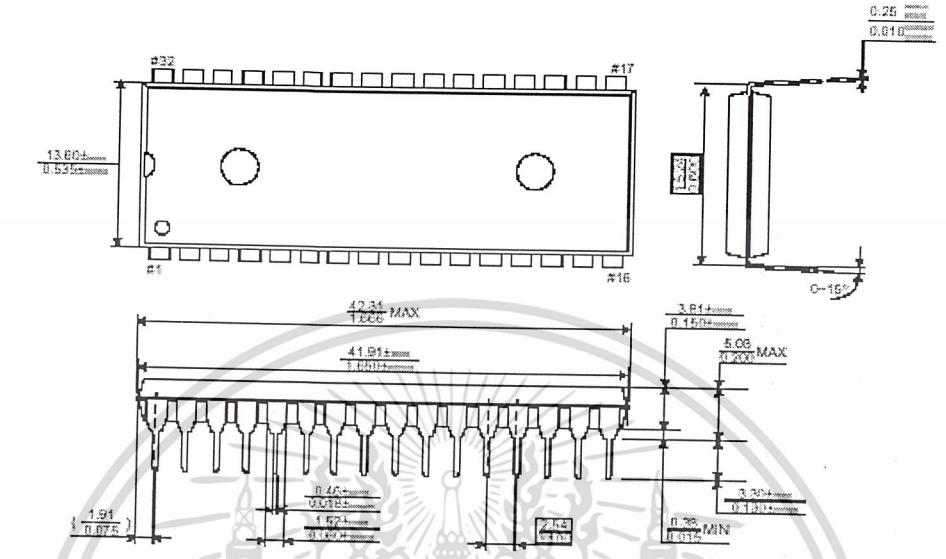
CS_2 controlled



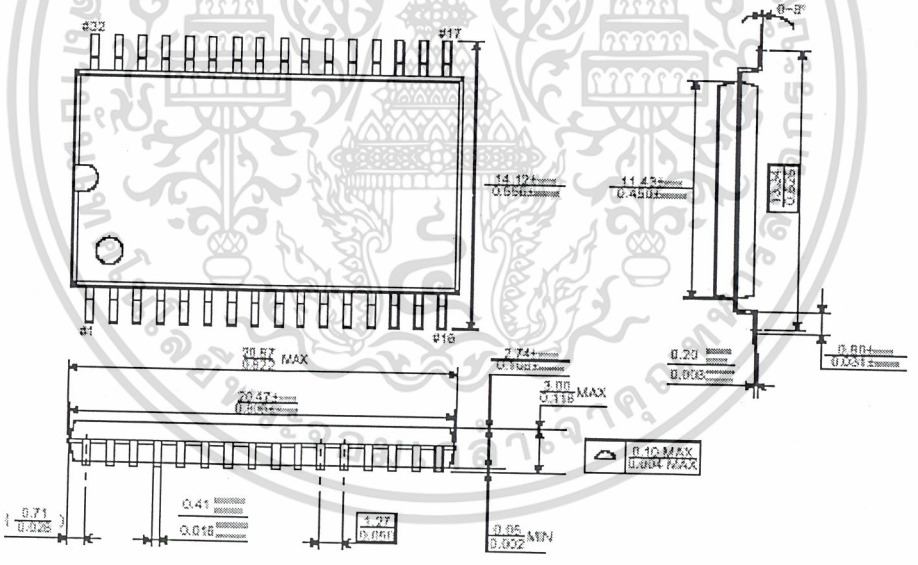
PACKAGE DIMENSIONS

32 DUAL INLINE PACKAGE (600mil)

Units: millimeters(inches)



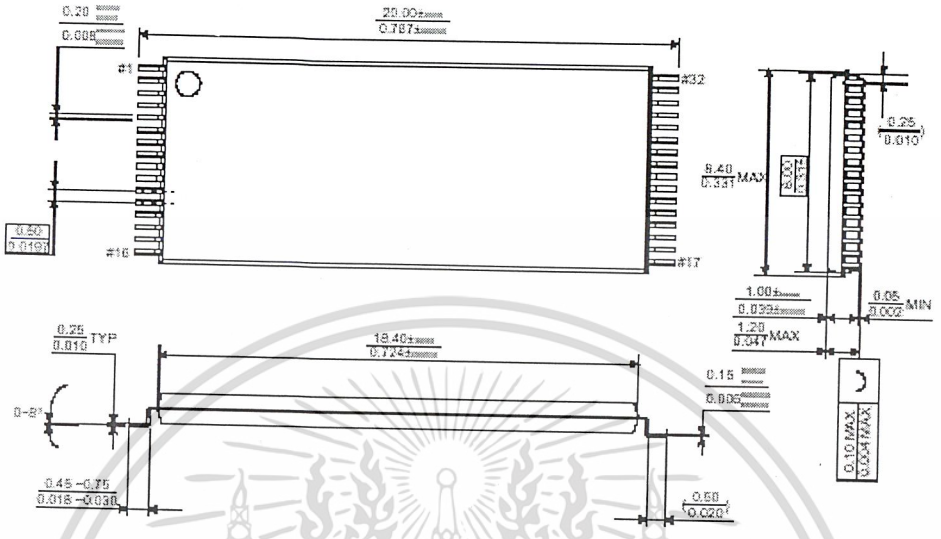
32 PLASTIC SMALL OUTLINE PACKAGE (525mil)



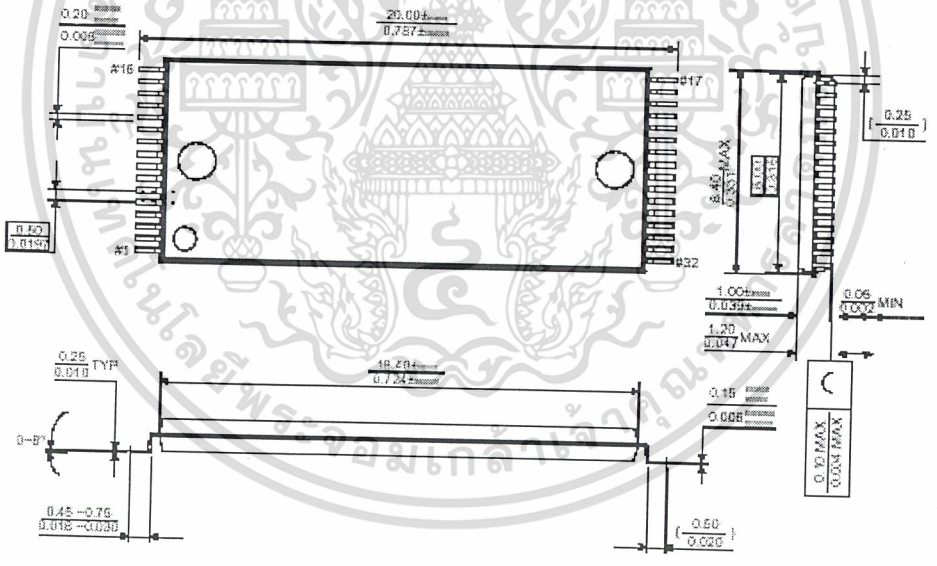
PACKAGE DIMENSIONS

Units: millimeters(inches)

32 PIN THIN SMALL OUTLINE PACKAGE TYPE I (0820F)



32 THIN SMALL OUTLINE PACKAGE TYPE I (0820R)



บรรณานุกรม

จรีลักษณ์ ประสิทธิ์มณีรัตน์, บรรยงค์ ด้วงนุ่ม. **"การบีบอัดข้อมูลเสียงพูดด้วยการแปลงเวฟเล็ต โดย DSP TMS320C31."** ปริญญานิพนธ์ครุศาสตร์อุตสาหกรรมบัณฑิต สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง. 2543

พิพัฒพงศ์ เกื้อศิริกุล, สุวสิต วิทยวิจักษณ์. **"ตัวถอดรหัสเสียงเอ็มเป็ก."** ปริญญานิพนธ์วิศวกรรมศาสตร์บัณฑิต สาขาวิชาวิศวกรรมอิเล็กทรอนิกส์, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง. 2541

ลภน สุภาพ. **"รู้จักและสร้างเครื่องถอดรหัส MP3 ด้วยมือของคุณ ตอน 1 มารู้จักกับ MP3."** วารสารเซมิคอนดักเตอร์อิเล็กทรอนิกส์. 215 (พ.ย. 43): หน้า 147 – 154 . 2543

ลภน สุภาพ. **"รู้จักและสร้างเครื่องถอดรหัส MP3 ด้วยมือของคุณ ตอน 2 สร้างบอร์ดถอดรหัส MP3."** วารสารเซมิคอนดักเตอร์อิเล็กทรอนิกส์. 216 (EBG 2001): หน้า 140 – 148 . 2543

ลภน สุภาพ. **"รู้จักและสร้างเครื่องถอดรหัส MP3 ด้วยมือของคุณ ตอน 3 วงจรทดลองส่วนเชื่อมต่อกับแหล่งข้อมูล."** วารสารเซมิคอนดักเตอร์อิเล็กทรอนิกส์. 217 (ธ.ค. 43): หน้า 147 – 156. 2543

ลภน สุภาพ. **"รู้จักและสร้างเครื่องถอดรหัส MP3 ด้วยมือของคุณ ตอน 4 ทำความเข้าใจกับ I^2C และมาตราฐานการเชื่อมต่อ."** วารสารเซมิคอนดักเตอร์อิเล็กทรอนิกส์. 218 (ม.ค. 44): หน้า 156 – 164 . 2544

ลภน สุภาพ. **"รู้จักและสร้างเครื่องถอดรหัส MP3 ด้วยมือของคุณ ตอน 5 การทำงานของบอร์ดควบคุมซีดีรอมและบอร์ดพาแนลดิสเพลย์."** วารสารเซมิคอนดักเตอร์อิเล็กทรอนิกส์. 219 (ก.พ. 44): หน้า 147 – 155 . 2544

ลภน สุภาพ. **"รู้จักและสร้างเครื่องถอดรหัส MP3 ด้วยมือของคุณ ตอนจบ การสร้างของบอร์ดควบคุมซีดีรอมและบอร์ดพาแนลดิสเพลย์."** วารสารเซมิคอนดักเตอร์อิเล็กทรอนิกส์. 220 (มี.ค. 44): หน้า 156 – 161 . 2544

CHASSAING, R. **Digital Signal Processing Laboratory Experiment Using C and the TMS320C31**. New York: University of Massachusetts.1999

บรรณานุกรม (ต่อ)

Morgan, Don. **Practical DSP Modeling, Techniques, and Programming in C**. New York: John Wiley, c1994

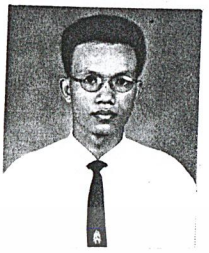
Sorensen, Henrik V. **A Digital Signal Processing Laboratory using the TMS320C30**. Upper Saddle River, NJ : Prentice Hall, c1997

Texas Instruments. **TMS320LC31 DIGITAL SIGNAL PROCESSORS.**[Online]. Available : http://www.usna.edu/EE/ee462/hardware/TMS320C31Datasheet.pdf. 1997

ประวัติผู้แต่ง

ชื่อผู้ทำปริญญานิพนธ์	นายเฉลิมพงษ์ ขันเงิน
วันเดือนปีเกิด	19 กุมภาพันธ์ พ.ศ. 2523
สถานที่เกิด	จังหวัดลำปาง
ภูมิลำเนาเดิม	42/2 หมู่ 2 ตำบลบ้านเอื้อม อำเภอเมือง จังหวัดลำปาง 52100
ที่อยู่ปัจจุบัน	312/109 หมู่บ้านเครือทิพย์ ซอยแพปลา ถนนอ่อนนุช – ลาดกระบัง แขวงลาดกระบัง เขตลาดกระบัง กรุงเทพฯ 10520
โทรศัพท์	0-2739-1834
ประวัติการศึกษา	
ประถมศึกษา	โรงเรียนวัดบ้านฮ่อง จังหวัดลำปาง
มัธยมศึกษาตอนต้น	โรงเรียนเขลางค์นคร จังหวัดลำปาง
ประกาศนียบัตรวิชาชีพ (ปวช.)	วิทยาลัยเทคนิคลำปาง
ประกาศนียบัตรวิชาชีพชั้นสูง (ปวส.)	วิทยาลัยเทคนิคลำปาง
ปริญญาตรี	สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์ ภาควิชาครุศาสตร์วิศวกรรม คณะครุศาสตร์อุตสาหกรรม สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ผลงานที่ได้รับรางวัล	-
ทุนการศึกษา	-
คติพจน์	ฝันไกลแค่ไหนก็ขอให้เดินไปให้ถึงความฝันของตนเอง

ประวัติผู้แต่ง



ชื่อผู้ทำปริญญานิพนธ์	นายชัยพัฒนพงษ์ ภูสัตย์คำ
วันเดือนปีเกิด	2 กุมพาพันธ์ พ.ศ. 2524
สถานที่เกิด	จังหวัดกาฬสินธุ์
ภูมิลำเนาเดิม	9 หมู่ 3 ตำบลนาดี อำเภอยางตลาด จังหวัดกาฬสินธุ์ 46120
ที่อยู่ปัจจุบัน	3 หมู่ 2 หอพักนักศึกษา ถนนฉลองกรุง แขวงลำปลาทิว เขตลาดกระบัง กรุงเทพฯ 10520
โทรศัพท์	0-9140-5596
ประวัติการศึกษา	
ประถมศึกษา	โรงเรียนปอแดงวิทยา จังหวัดกาฬสินธุ์
มัธยมศึกษาตอนต้น	โรงเรียนปอแดงวิทยา จังหวัดกาฬสินธุ์
ประกาศนียบัตรวิชาชีพ (ปวช.)	วิทยาลัยเทคนิคกาฬสินธุ์
ประกาศนียบัตรวิชาชีพชั้นสูง (ปวส.)	วิทยาลัยเทคนิคกาฬสินธุ์
ปริญญาตรี	สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์ ภาควิชาครุศาสตร์วิศวกรรม คณะครุศาสตร์อุตสาหกรรม สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหาร ลาดกระบัง
ผลงานที่ได้รับรางวัล	-
ทุนการศึกษา	กองทุนกู้ยืมรัฐบาล
คติพจน์	**กล้าคิด กล้าพูด กล้านำ กล้าทำในสิ่งที่ถูกต้อง**

ประวัติผู้แต่ง

ชื่อผู้ทำปริญญานิพนธ์	นายมนตรี ศรีสง่า
วันเดือนปีเกิด	23 สิงหาคม พ.ศ. 2523
สถานที่เกิด	จังหวัดตาก
ภูมิลำเนาเดิม	179 ถนนจอมพล ตำบลระแหง อำเภอเมือง จังหวัดตาก 63000
ที่อยู่ปัจจุบัน	260/81 ซอยดับเพลิง ถนนหลวงแพ่ง เขตลาดกระบัง กรุงเทพฯ 10520
โทรศัพท์	0-6132-2565
ประวัติการศึกษา	
ประถมศึกษา	โรงเรียนเทศบาล 4 รัตนวิทยานุสรณ์ จังหวัดตาก
มัธยมศึกษาตอนต้น	โรงเรียนตากพิทยาคม จังหวัดตาก
ประกาศนียบัตรวิชาชีพ (ปวช.)	สถาบันเทคโนโลยีราชมงคล วิทยาเขตตาก
ประกาศนียบัตรวิชาชีพชั้นสูง (ปวส.)	สถาบันเทคโนโลยีราชมงคล วิทยาเขตตาก
ปริญญาตรี	สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์ ภาควิชาครุศาสตร์วิศวกรรม คณะครุศาสตร์อุตสาหกรรม สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหาร ลาดกระบัง
ผลงานที่ได้รับรางวัล	-
ทุนการศึกษา	กองทุนกู้ยืมรัฐบาล