ระบบรู้จำเสียงพูดอย่างง่าย

นายนุวัศ เจนเกียรติพู
นายสงกรานต์ คุ้มพูล

โครงงานพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตร์บัณฑิต
ภาควิชาฟิสิกส์ประยุกต์
คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ.2537

Simple Speech Recognition System

Mr. Nuwat Jainkeatfu

Mr. Songkran Khumphul

A Special Project Submitted in Partial Funfillment of the

Reguirement for the Degree of Bachelor of Science

Department of Applied Physics

Faculty of Science

King Mongkut's Institute of Technology Ladkrabang

1994

หัวข้อโครงงานพิเศษ	ระบบรู้จำเสียงพูดอย่างง่าย
โดย	นายนุวัศ เจนเกียรติฟู
	นายสงกรานต์ คุ้มพูล
ภาควิชา	ฟิสิกส์ประยุกต์
อาจารย์ที่ปรึกษา	รศ. สุรพล รักวิจัย
	ผศ. วิชิต ศิริโชติ

ภาควิชาฟิสิกส์ประยุกต์ คณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง อนุมัติให้นับโครงงานพิเศษฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตร์บัณฑิต

ลายเซ็น

.....

(ผศ. ดร. ปรีชา เทียนสมประสงค์) หัวหน้าภาควิชาฟิสิกส์ประยุกต์

คณะกรรมการโครงงานพิเศษ

.....

(รศ. สุรพล รักวิจัย) ประธานกรรมการ

.....

(ผศ. ดร. วราวุฒิ เถาลัดดา) กรรมการ

.....

(อ. วิชาญ เตชิตธีระ) กรรมการ

.....

(อ. รัชภาคย์ จิตย์อารี) กรรมการ

หัวข้อโครงงานพิเศษ	ระบบรู้จำเสียงพูดอย่างง่าย
โดย	นายนุวัศ เจนเกียรติพู
	นายสงกรานต์ คุ้มพูล
ภาควิชา	ฟิสิกส์ประยุกต์
อาจารย์ที่ปรึกษา	รศ. สุรพล รักวิจัย
	ผศ. วิชิต ศิริโชติ
ปีการศึกษา	2537

บทคัดย่อ

โครงงานพิเศษนี้เป็นการพัฒนาระบบรู้จำเสียงพูดแบบคำเดี่ยว บุคคลเดี่ยว สำหรับภาษาไทย ระบบนี้ประกอบด้วยไมโครคอนโทรเลอร์ซึ่งควบคุมการแปลงสัญญาณเสียง และมีอัลกอริธึมซึ่งเขียนด้วยภาษาปาสคาล การแปลงสัญญาณเสียงใช้วิธีการประมาณค่าหลายครั้ง (successive approximation) โดยใช้อัตราการสุ่มตัวอย่างเท่ากับ 8 kHz. ควบคุมโดยใช้ไมโครคอนโทรลเลอร์ แล้วส่งข้อมูลไปยังไมโครคอมพิวเตอร์ โดยผ่านทางพอร์ตอนุกรม RS-232C ข้อมูลนั้นจะผ่านการหาขอบเขตสัญญาณ ข้อมูลที่ได้จะนำไปหาค่าสัมประสิทธิ์การทำนายเชิงเส้น เพื่อสร้างแบบทดสอบและแบบอ้างอิง ซึ่งใช้เทคนิคการเข้ารหัสแบบทำนายเชิงเส้น โดยวิธีออโตคอรีเลชัน การรู้จำจะใช้แบบอ้างอิงและแบบทดสอบมาเปรียบเทียบโดยวิธีไดนามิกโปรแกรมมิง

จากการทดลองระบบรู้จำเสียงพูดภาษาไทยของตัวเลข 0-9 ผลที่ได้พบว่าอัตราการรู้จำและเวลาที่ใช้ในการรู้จำแปรผันตรงกับจำนวนแบบอ้างอิงและลำดับของการทำนาย

Special Project Title	Simple Speech Recognition System
Name	Mr. Nuwat Jainkeatfu
	Mr. Songkran Khumphul
Special Project Advisor	Assoc.Prof. Surapol Rakvijai
	Asst.Prof. Wichit Sirichote
Department	Applied Physics
Academic Year	1994

Abstract

A dependent-word-based speech recognition system for Thai language has been developed. The system consists of a microcontroller based voice digitizer, and a recognition algorithm written in Pascal Language. A time domain speech signal was digitized by a successive approximation analog-to-digital converter with sampling rate of 8 kHz. The digitized data was firstly processed, i.e. word correction, by the microcontroller and was sent to a PC via a serial port, RS232C. A Linear Precitive Coding technique, LPC, with an autocorrelation solving method was used to determine the predictive coeffficients both reference and test template, which were compared by a dynamic programming method.

Experiment was carried out with the number 0-9 pronouned in Thai language. Results have shown that both recognition rate and recognition time increases as the number of reference template and the order of LPC technique were increased.

กิตติกรรมประกาศ

โครงงานพิเศษชิ้นนี้สำเร็จลุล่วงได้ด้วยดี เนื่องด้วยความอนุเคราะห์จากบุคคลหลายฝ่ายดังนี้

พ่อแม่	ผู้ให้ร่างกาย จิตใจ สติปัญญา รวมทั้งความห่วงใยแม้ว่าสุขภาพของท่านจะน่าเป็นห่วงมากกว่า
รศ.สุรพล รักวิจัย และ	
ผศ.วิชิต ศิริโชติ	ผู้ให้คำแนะนำและเป็นอาจารย์ที่ปรึกษา
คุณไพศาล ธรรมโพธิ์ทอง และ	
คุณอาทร นันทิยกุล	ผู้ให้ข้อมูลในส่วนการวิเคราะห์และรู้จำ
คุณศิระ ริกุลสุรกาน	ผู้ให้ยืมคอมพิวเตอร์เพื่อทำโครงงานพิเศษ และขับรถพาไปกินข้าว
คุณวิทิดา เจนเกียรติพู	ผู้ช่วยพิมพ์รายงานโครงงานพิเศษ
คุณเจริญศักดิ์ มันตาดิลก	ผู้ให้ยืมคอมพิวเตอร์เพื่อพิมพ์รายงานโครงงานพิเศษ และเอื้อเฟื้อที่หลับนอน
คุณประจวบ ฉัตรกุลกวิน	ผู้ให้ยืมเครื่องพิมพ์
คุณอมร อมรศิลป์	ผู้ขับรถพาไปกินข้าว
ภาควิชาฟิสิกส์ประยุกต์	ที่ให้ความอนุเคราะห์อุปกรณ์ที่ใช้ในการทำโครงงานพิเศษ
คณะกรรมการทุกท่าน	ที่กรุณาตรวจทานรายงานโครงงานพิเศษ
พี่ๆ เพื่อนๆ น้องๆ	ผู้แวะเวียนมาถามไถ่และให้คำแนะนำ

รวมทั้งสิ่งที่ไม่มีตัวตนแต่มีชีวิตดังนี้

ค่ำคืน	ผู้ให้ความสว่างทางความคิด
เสียงเพลง	ผู้สร้างความผ่อนคลาย
ความอ่อนล้า	ผู้สร้างความตื่นตัว
ถ้อยคำกดดัน	ผู้สร้างความกดดัน

สารบาญเรื่อง

สารบาญรูป

สารบาญตาราง

บทที่1
บทนำ

1.1 ความเป็นมาของปัญหา

โดยปกติการติดต่อระหว่างคอมพิวเตอร์กับมนุษย์ มักกระทำโดยป้อนข้อมูลเข้าทางคีย์บอร์ด และ แสดงผลออกทางจอภาพหรือเครื่องพิมพ์ การติดต่อกับคอมพิวเตอร์ผ่านทางคีย์บอร์ดให้เกิดประสิทธิภาพ จำเป็นต้องใช้ความชำนาญ และการฝึกฝน การเพิ่มความสามารถในการติดต่อกับคอมพิวเตอร์ผ่านทางคีย์บอร์ด จึงไม่อาจพัฒนาต่อไปได้ด้วยความเร็วอันจำกัดของการป้อนข้อมูล ด้วยสาเหตุนี้จึงมีการคิดวิธีการป้อนข้อมูลในลักษณะต่างๆเช่น เมาท์, ปากกาคอมพิวเตอร์ และการป้อนข้อมูลทางเสียงพูด

การป้อนข้อมูลทางเสียงพูด จะเพิ่มขีดความสามารถการป้อนข้อมูลแทนคีย์บอร์ดได้ เนื่องจากไม่จำเป็นต้องฝึกฝนการออกเสียงพูด และการพูดจะกระทำได้เร็วกว่าการกดคีย์บอร์ดโดยเฉพาะอุปกรณ์ที่ต้องการข้อมูลทางเข้าไม่มากนัก และต้องการลดขนาดของอุปกรณ์ หรือลดจำนวนปุ่มกดต่างๆ รวมทั้งการสร้างความสะดวกสบายแก่คนพิการ ด้วยสาเหตุนี้จึงเกิดการศึกษาการป้อนข้อมูลทางเสียง เพื่อรองรับงานในลักษณะดังกล่าว

คอมพิวเตอร์จะเข้าใจข้อมูลที่ป้อนให้ได้ จะต้องเก็บเสียงพูดที่จะใช้งานเอาไว้ เพื่อรอการเปรียบเทียบเสียงพูดที่ผู้ใช้ป้อนเข้ามา นำมาเปรียบเทียบกับข้อมูลเสียงพูดที่เก็บไว้อยู่เดิมเรียกขบวนการข้างต้นว่า การรู้จำเสียงพูด (Speech Recognition)

1.2 วัตถุประสงค์ของโครงงานพิเศษ

1. เพื่อศึกษาระบบการวิเคราะห์ และการรู้จำเสียงพูด โดยวิธีพิจารณาคำเดี่ยว (Word Based Speech Recognition)
2. เพื่อสร้างวงจรแปลงสัญญาณอนาลอกเป็นสัญญาณดิจิตอล และวงจรส่งข้อมูลจากบอร์ดไปประมวลผลด้วยไมโครคอมพิวเตอร์
3. สามารถใช้ไมโครคอนโทรลเลอร์ควบคุมการทำงานส่วนฮาร์ดแวร์
4. เพื่อศึกษาอัลกอริธึมที่เหมาะสม สำหรับใช้ในการรู้จำเสียงพูด
5. เพื่อเป็นพื้นฐานในการนำไปประยุกต์ใช้งาน

1.3 ขอบเขตการทำงาน

1. สร้างวงจรแปลงสัญญาณเสียงพูดเป็นสัญญาณดิจิตอล
2. สร้างวงจรส่งข้อมูลจากบอร์ดไปประมวลผลด้วยคอมพิวเตอร์
3. ใช้ไมโครคอนโทรลเลอร์ควบคุมวงจรส่วนฮาร์ดแวร์
4. เขียนซอฟต์แวร์เพื่อใช้ควบคุมการทำงานบนบอร์ด
5. เขียนซอฟต์แวร์เพื่อใช้รับข้อมูลเข้ามาประมวลผล
6. เขียนซอฟต์แวร์เพื่อใช้รู้จำเสียงพูดแบบพิจารณาคำเดี่ยว โดยมีขอบเขตดังนี้
    - 6.1 ใช้ผู้บอกเสียงพูดเพียงคนเดียว
    - 6.2 สามารถรู้จำเสียงพูดเป็นตัวเลข 0-9

บทที่ 2
การวิเคราะห์และการรู้จำเสียงพูด

2.1 ลักษณะของเสียงพูด

มนุษย์เปล่งเสียงพูดด้วยอวัยวะที่ใช้ในการออกเสียง (Organ of Speech) โดยเปล่งเสียงตามที่มีอยู่ในระบบภาษาของตน แม้ว่าคนที่อยู่ในสังคมเดียวกัน จะใช้ภาษาเดียวกัน แต่ถ้าพิจารณาเสียงที่เปล่งออกมาจริงๆในแต่ละครั้ง จะให้เสียงพูดออกมาที่แตกต่างกัน ทำให้สามารถจดจำเสียงของ แต่ละบุคคลได้ เสียงที่พูดนี้มีลักษณะที่อธิบายได้ด้วยหลักเกณฑ์ทางวิทยาศาสตร์ ซึ่งเป็นภาษาสากล แม้ภาษาหนึ่งๆ จะมีเสียงที่แตกต่างกันไปมากบ้างน้อยบ้าง แต่สามารถอธิบายการออกเสียง และตำแหน่งที่เกิดเสียงได้ คำอธิบายนี้จะทำให้เข้าใจลักษณะเสียงพูด วิชาที่ว่าด้วยเสียงพูดเรียกว่า *วิชาสัทศาสตร์* (Phonetics)

การศึกษาเสียงพูดแบ่งออกเป็น 2 ลักษณะ ดังนี้

1) สรีรสัทศาสตร์ (Articulatory Phonetics) เป็นการศึกษาเสียงพูดจากอวัยวะ และการเคลื่อนไหวของอวัยวะที่ทำให้เกิดเสียง การอธิบายจะอาศัยลักษณะการเคลื่อนไหวของอวัยวะที่ทำให้เกิดการเปล่งเสียง

2) กลสัทศาสตร์ (Acoustic Phonetics) เป็นการศึกษาเสียงพูดจากลักษณะของคลื่นเสียงที่เปล่งออกมา การอธิบายจะอาศัยฟิสิกส์ และคณิตศาสตร์

2.1.1 อวัยวะที่ใช้ในการเปล่งเสียง (Organs of Speech)

อวัยวะที่ใช้ในการออกเสียงพูดมีหลายส่วน แต่ละส่วนสามารถทำให้เสียงพูดแตกต่างกันไปได้ อวัยวะเหล่านี้ได้แก่ ปาก และส่วนต่างๆในปาก ช่องคอ กล่องเสียง ช่องปาก และช่องจมูก ดังรายละเอียดในรูปที่ 2.1.1 อวัยวะที่ใช้ในการออกเสียงแบ่งได้เป็น 2 พวก คือ

1) อวัยวะที่ใช้ในการกระทำอาการ (Articulator) อวัยวะส่วนที่เคลื่อนไหว เพื่อผลักลมไปยังส่วนต่างๆ อวัยวะตัวกระทำอาการที่สำคัญคือ ลิ้น ซึ่งเป็นส่วนที่เคลื่อนไหวได้มากที่สุด

2) อวัยวะที่เป็นตำแหน่งที่เกิดเสียงต่างๆ (Point of Articulator) หมายถึงตำแหน่ง หรือฐานกรณ์ที่เกิดของเสียงต่างๆ เช่น ริมฝีปาก ฟัน เพดานส่วนต่างๆ เป็นต้น

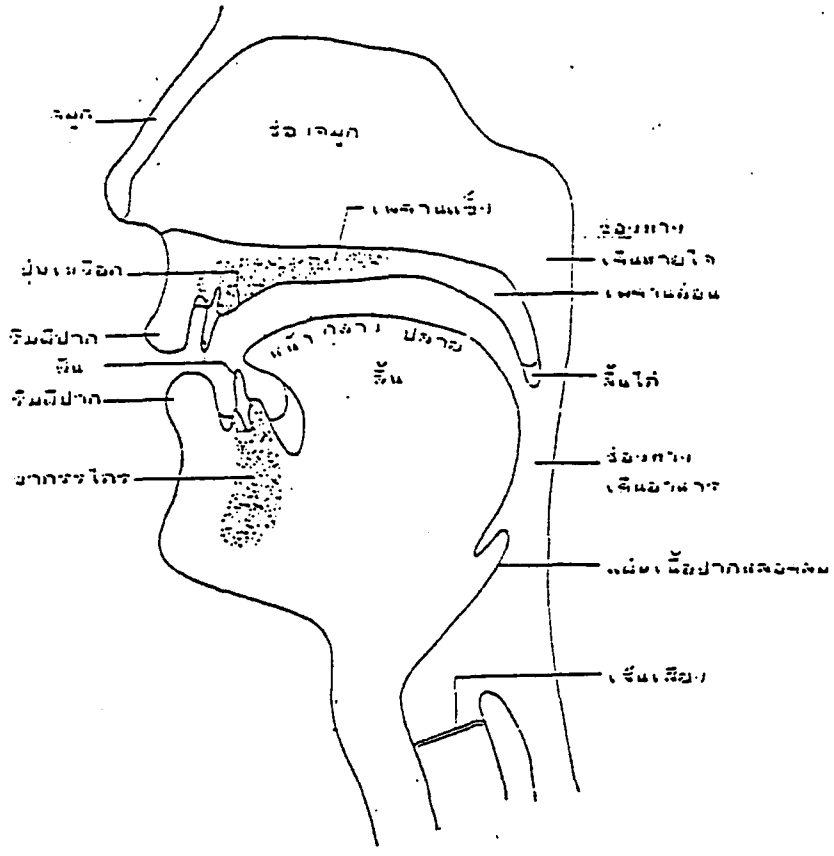
2.1.2 <u>การเกิดของเสียง</u> (Speech Production)

ขั้นตอนการเกิดของเสียงแบ่งออกได้เป็น 3 ขั้นตอนใหญ่ๆ ดังนี้

1) ขั้นเริ่มต้น (Initiation) ขั้นตอนนี้เป็นขั้นตอนที่ลมเริ่มถูกขับออกจากปอด เพื่อเข้าสู่ขั้นตอนที่ 2 ต่อไป

2) ขั้นตอนดัดแปลงลมที่เส้นเสียง (Phonation) เป็นขั้นตอนที่ลมจากปอดจะผ่านมายังหลอดลม และกล่องเสียง ซึ่งที่กล่องเสียงนี้ เส้นเสียงจะทำหน้าที่เป็นลิ้นปิด เปิด ทำให้เกิดเสียงได้ 2 ชนิดคือ *เสียงก้อง* (Voice Sounds) และ*เสียงไม่ก้อง* (Unvoice Sounds) อวัยวะที่ใช้ในขั้นตอนนี้คือส่วนที่ต่อจากปอดขึ้นมาจนถึงกล่องเสียง

3) ขั้นตอนเปลี่ยนแปลงลักษณะเสียง (Articulation)ในขั้นตอนนี้ลมที่ผ่านออกมาจากกล่องเสียงจะถูกแปลงให้เกิดเสียงในลักษณะต่างๆ อวัยวะที่ใช้ในขั้นตอนนี้คือส่วนที่ต่อจากกล่องเสียง จนถึงริมฝีปาก



รูปที่ 2.1.1 แสดงอวัยวะที่ใช้ในการออกเสียงพูด

เสียงที่เกิดจากขั้นตอนดังกล่าว จะถูกแบ่งออกเป็น 2 ประเภทใหญ่ๆ คือ

*1) **เสียงก้อง*** **(Voice Sounds)** เกิดจากการออกเสียงในขณะที่เส้นเสียงถูกดึงเข้ามาใกล้กันจนเกือบปิดช่องทางลมเสียสนิท ลมที่ดันขึ้นมาจากปอดจะทำให้เส้นเสียงสั่น ลมที่ออกมาจึงไม่สะดวก เพราะต้องบีบตัวผ่านช่องแคบ เป็นจังหวะๆ ทำให้เกิดเป็นเสียงก้อง

*2) **เสียงไม่ก้อง*** **(Unvoice Sounds)** เป็นการออกเสียงในขณะที่เส้นเสียงยังเปิดกว้าง โดยเปิดช่องระหว่างเส้นเสียง หรือช่องคอหอย (Glottis) ให้ลมหายใจผ่านเข้าออกสะดวก ทำให้เกิดเสียงไม่ก้อง

สามารถแบ่งวิธีการรู้จำเสียงพูดออกได้เป็น 2 วิธี ดังนี้

*1) **แบบพิจารณาทั้งหน่วยคำ*** (Word Based Speech Recognition) หน่วยคำที่ใช้พิจารณาในวิธีนี้ อาจเป็นพยางค์เดี่ยว คำ กลุ่มคำ วลี หรือประโยค

*2) **แบบพิจารณารายละเอียดของหน่วยเสียง*** (Phoneme Based Speech Recognition) เป็นการพิจารณารายละเอียดของหน่วยเสียงที่เล็กลงไป เช่น สระ พยัญชนะ หรืออรรถยุกต์ โดยใช้หน่วยเสียงเหล่านี้เป็นหลักในการเปรียบเทียบ

การรู้จำเสียงพูดแบบพิจารณารายละเอียดของหน่วยเสียง สามารถรู้จำคำได้จำนวนมากกว่าและให้ความแม่นยำได้มากกว่าการพิจารณาทั้งหน่วยคำ แต่ขอบเขตการศึกษาจะกว้าง และลึกซึ้งมากกว่าเช่นกัน โครงงานพิเศษนี้จึงใช้การรู้จำแบบพิจารณาทั้งหน่วยคำ โดยให้รู้จำตัวเลข 0 ถึง 9 เท่านั้น

2.2 <u>หลักการวิเคราะห์เสียง</u>

การวิเคราะห์เสียงพูดสามารถแบ่งออกได้เป็น 3 ลักษณะใหญ่ๆ คือ

1) การวิเคราะห์ในด้านของโดเมนเวลา

2) การวิเคราะห์ในด้านของโดเมนความถี่

3) การวิเคราะห์โดยการหาคาบของสัญญาณเสียง

2.2.1 <u>การวิเคราะห์ในโดเมนเวลา</u> (Time Domain)

การประมวลผลสัญญาณเสียงในโดเมนเวลา มีประโยชน์ในด้านการประมวลผลสัญญาณได้ไม่ยาก การคำนวณใช้เวลาไม่มากนักและง่ายในการวิเคราะห์

การวิเคราะห์ในโดเมนเวลา คือวิธีการประมวลผลสัญญาณเสียงพูดโดยยึดหลักพิจารณาจากรูปคลื่นสัญญาณที่แปรตามแกนเวลาซึ่งมีอยู่หลายวิธีด้วยกัน ถ้าเป็นการประมวลสัญญาณเสียงแบบดิจิตอล สามารถพิจารณาจากตัวอย่างที่คัดเข้ามา เช่น เอาท์พุทที่ได้จากเทคนิคการเข้ารหัสสัญญาณพีซีเอ็ม ผลของการวิเคราะห์สัญญาณเสียงในโดเมนเวลา เช่น พลังงานหรือแอมปลิจูดของเสียง การตรวจสอบว่าเสียงนั้นเป็นเสียงก้องหรือไม่ก้อง การหาคาบของสัญญาณเสียงเป็นต้น จากวิธีการต่างๆ เหล่านี้จะมีวิธีการที่เป็นพื้นฐานในการวิเคราะห์สัญญาณเสียงในโดเมนเวลา ดังจะกล่าวต่อไป

1) การวัดพลังงานของสัญญาณเสียง

พลังงานของสัญญาณเป็นปริมาณหนึ่งที่มักนำมาใช้ในการวิเคราะห์ลักษณะต่างๆ ของสัญญาณทั่วๆไป โดยพลังงานสัญญาณ s(n) ใดๆที่แปรตามเวลาจะนิยามได้ดังนี้

$$E = \sum_{n=-\infty}^{\infty} s^2(n) \qquad \text{..... 2.2.1}$$

แต่สัญญาณเสียงที่เป็นสัญญาณที่เปลี่ยนอยู่ตลอดเวลา ไม่มีเสถียรภาพตามเวลา จะต้องแบ่งสัญญาณออกมาพิจารณาเป็นช่วงๆ ตามแกนเวลา หรือที่เรียกว่าแบ่งเป็นเฟรม เช่น เฟรมละประมาณ 10-30 มิลลิวินาที ดังนั้นสามารถหาพลังงานของเสียงในแต่ละเฟรมได้เป็น

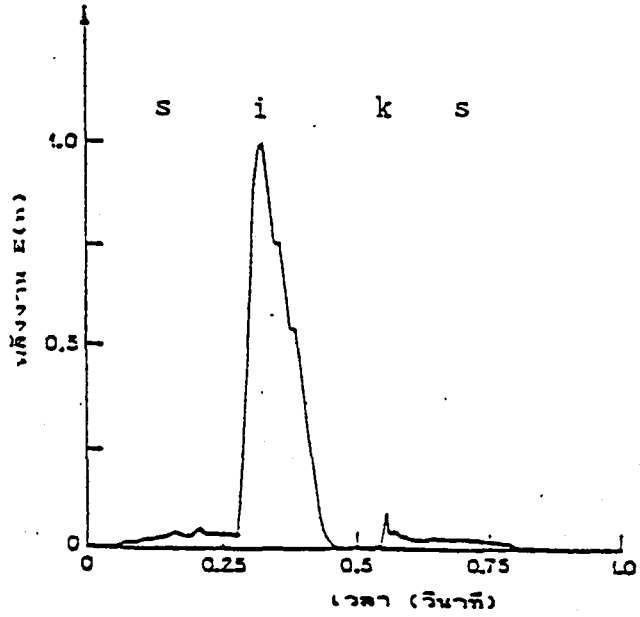
$$E(n) = \sum_{m=0}^{N-1} [w(m)s(n-m)]^2 \qquad \text{..... 2.2.2}$$

โดยที่ w(n) คือ *วินโดว์ฟังก์ชั่น* (Window Function) ที่ใช้กำหนดช่วงเวลาในการศึกษาสัญญาณเสียง s(n) ในหนึ่งเฟรม และ N คือจำนวนตัวอย่างของสัญญาณเสียงภายในหนึ่งเฟรม หรือภายในกรอบวินโดว์ฟังก์ชัน

การวัดค่าพลังงานในสมการที่ 2.2.2 มีข้อจำกัดคือ สมการจะมีความไวต่อสัญญาณเสียงที่มีขนาดใหญ่ๆ เนื่องจากใช้วิธียกกำลังสอง ดังนั้นการแก้ปัญหาอย่างหนึ่งคือการใช้ฟังก์ชันดังนี้

$$E(n) = \sum_{m=0}^{N-1} |w(m)s(n-m)| \qquad \text{..... 2.2.3}$$

นั่นคือการใช้การรวมผลบวกของค่าสัมบูรณ์ หรือที่เรียกว่า*ค่าแมกนิจูด* (magnitude) แทนผลรวมของค่ากำลังสอง จากรูปที่ 2.2.1 แสดงตัวอย่างรูปฟังก์ชันของพลังงานของคำว่า /six/ ภายในช่วงของคาบแบบสี่เหลี่ยมขนาด 10 มิลลิวินาที จะสังเกตได้ว่า สามารถเห็นช่วงเริ่มต้นของคำก่อนจะเริ่มต้นเสียง /s/ ซึ่งเป็นเสียงเสียดแทรก และช่วงหยุดก่อนจะเริ่มเสียง /k/ ที่มีระดับของพลังงานต่ำจนเกือบเป็นศูนย์



รูปที่ 2.2.1 รูปพลังงานของคำว่า /six/ [siks]

2) การวิเคราะห์สัญญาณเสียงในช่วงเวลาสั้นๆ (Short-time Speech Analysis)

เนื่องจากสัญญาณเป็นสัญญาณที่แปรตามเวลา มีการแปรเปลี่ยนที่ไม่แน่นอน แต่ส่วนใหญ่แล้วผู้พูดจะสามารถทำการควบคุมได้ เช่น ระหว่างเสียงพูดช่วงช้าๆ รูปร่างของช่องทางเดินเสียงรวมทั้งลักษณะรูปแบบของการกระตุ้นอาจมีความคงที่ได้นานที่สุดประมาณ 200 มิลลิวินาที แต่อย่างไรก็ตามในช่วงที่มีการเปลี่ยนแปลงอย่างรวดเร็ว เช่น สำหรับการออกเสียงสั้นๆ อาจมีช่วงคงที่ประมาณ 80 มิลลิวินาที

ดังนั้นเทคนิคในการวิเคราะห์เสียงพูดส่วนใหญ่ จะสมมติให้สัญญาณเสียงมีคุณสมบัติที่เปลี่ยนแปลง อย่างช้าๆ นั่นคือจะต้องแบ่งการทำงานพิจารณาพารามิเตอร์ของสัญญาณเสียงพูดในช่วงเวลาสั้นๆ เหมือนมองผ่านช่องแคบที่เรียกว่า *ช็อตไทม์ วินโดว์* (short-time window) เมื่อเปรียบตามเวลาที่เสียงมีการเปลี่ยนแปลง เพื่อจะมองเห็นเหมือนกับหาพารามิเตอร์นั้นๆ ได้มาจากสัญญาณเสียงที่อยู่ภายในช่องแคบๆ และมีเสถียรภาพในช่วงเวลานั้นๆ

เทคนิคส่วนใหญ่จะกำหนดให้พารามิเตอร์ได้มาจาก ค่าเฉลี่ยที่ได้มาจากการพิจารณาภายในช่วงเวลาแคบๆ สำหรับกรณีที่ต้องพิจารณาพารามิเตอร์ต่างๆ ที่มีการเปลี่ยนแปลง ก็จะทำการแบ่งสัญญาณเสียงออกเป็นช่องหลายๆ ช่อง หรืออาจจะเรียกว่า *กรอบการวิเคราะห์* (analysis frame) ดังนั้นพารามิเตอร์ต่างๆ ก็สามารถหาได้ทัน เพียงพอที่จะติดตามการเปลี่ยนแปลงที่เกิดขึ้นของสัญญาณเสียง เช่น เสียงสระที่ยาวๆ สัญญาณเสียงจะมีการเปลี่ยนแปลงในช่วงเวลาที่ช้า อาจกำหนดให้ช่องแคบมีขนาดใหญ่ประมาณ 100 มิลลิวินาที แต่ในทางตรงกันข้าม ถ้าในกรณีเสียงสั้น อาจต้องใช้ช่องแคบที่มีขนาดเล็กมากๆ ประมาณ 5-10 มิลลิวินาที เพื่อป้องกันการสูญหายใน รายละเอียดของการต่อเนื่องกันของเสียงถัดไป

3) รูปแบบของช่องวิเคราะห์ (Windowing)

รูปแบบของช่องวิเคราะห์ที่ใช้ควรมีลักษณะดังนี้

3.1) ช่องวิเคราะห์จะต้องสั้นพอ ที่จะทำให้คุณสมบัติของเสียงที่กำลังพิจารณาไม่มีการเปลี่ยนแปลงอย่างมีนัยสำคัญภายในช่องนั้น

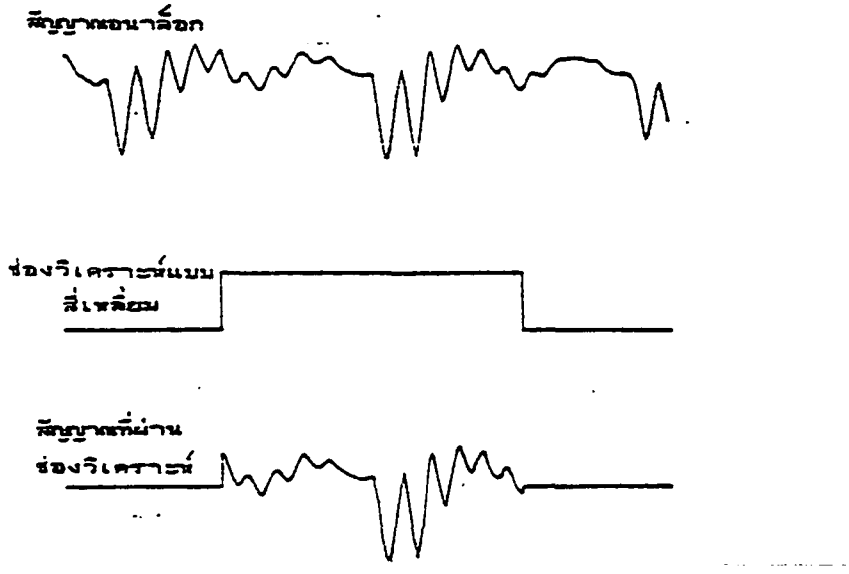
3.2) ช่องวิเคราะห์จะต้องยาวพอ ที่จะได้ตัวอย่างของเสียงเพื่อจะนำไปคำนวณหาพารามิเตอร์ตามที่ต้องการ เช่น กรณีที่มีสัญญาณรบกวนเข้ามาแทรกอยู่บางช่วงในสัญญาณเสียง ถ้าเลือกใช้ช่องแคบที่มีขนาดใหญ่กว่า เมื่อหาค่าพารามิเตอร์โดยเฉลี่ยแล้ว จะทำให้ส่วนประกอบของสัญญาณรบกวนถูกตัดทิ้งหรือมองข้ามไป

3.3) ช่องวิเคาะห์ต้องมีความยาวเหมาะสม คือไม่สั้นเกินไปกว่าช่วงหนึ่งคาบสัญญาณเสียงในช่วงที่กำลังพิจารณา เงื่อนไขนี้จะมีผลตอค่าเฟรมเรท (frame rate) หรือจำนวนครั้งต่อนาทีที่ทำการวิเคระห์สัญญาณเสียง ซึ่งเกิดขึ้นจากการขยับช่องวิเคราะห์ไปเป็นคาบๆ ตามแกนเวลามากกว่าขนาดของช่องวิเคราะห์ ตามปกติเฟรมเรทจะมีค่าประมาณ 2 เท่าของส่วนกลับของขนาดช่องวิเคราะห์ นั่นคือช่องวิเคราะห์ถัดๆ กันไปจะมีการซ้อนทับกัน 50 เปอร์เซ็นต์

การเอาฟังก์ชันของช่องวิเคาะห์ที่มีช่องขนาดจำกัด w(n) มาคูณเข้ากับสัญญาณ s(n) จะทำให้ได้กลุ่มตัวอย่างของเสียงพูดที่ถูกกำหนดน้ำหนัก ให้แปรไปตามรูปร่างของช่องวิเคราะห์ รูปแบบของช่องวิเคราะห์ที่ง่ายที่สุดคือ *ช่องวิเคราะห์แบบสี่เหลี่ยม* (rectangular shape window) ซึ่งมีนิยามดังนี้

$$w(n) = 1 \qquad \text{สำหรับ } 0 \le n \le N-1$$
$$0 \qquad \text{สำหรับ กรณีอื่นๆ} \qquad \text{..... 2.2.4}$$

ในสมการ 2.2.4 คือการกำหนดช่วงของการวิเคราะห์ให้อยู่ภายในตัวอย่าง N ตัวอย่างที่ถัดๆไป

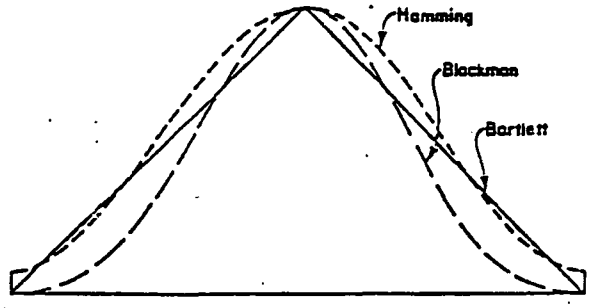


รูปที่ 2.2.2 ช่องวิเคราะห์แบบสี่เหลี่ยม

ในการประยุกต์ส่วนมาก จะใช้ช่องวิเคราะห์ที่มีช่วงของข้อมูลที่คงที่ และเน้นเฉพาะช่วงตรงกลางของช่องวิเคราะห์ให้เป็นส่วนของข้อมูลที่จะทำการพิจารณา เช่น ข้อมูลสัญญาณเสียงที่คงที่ในช่วงเวลา 10 มิลลิวินาที อาจจะใช้ช่องวิเคราะห์ขนาด 20 มิลลิวินาที โดยในช่วงกึ่งกลางช่องวิเคราะห์ขนาด 10 มิลลิวินาที จะมีการเน้นน้ำหนักให้มากกว่าช่วง 5 มิลลิวินาที ที่ปลายทั้งสองข้างของช่อง

รูปแบบของช่องวิเคราะห์ มีฟังก์ชันหลายลักษณะ เช่น Blackman , Barlett , Hamming เป็นต้น โดยฟังก์ชันของช่องวิเคราะห์ที่นิยมใช้กันมากในการวิเคราะห์สัญญาณเสียงคือ ฟังก์ชันของแฮมมิง ซึ่งมีรูปร่างตามลักษณะของโคซายน์พัลส์ (cosine pulse) มีนิยามดังนี้

$$w(n) = 0.54 - 0.46\cos(2\pi n)/(N-1) \qquad \text{สำหรับ} 0 \le n \le N-1 \qquad .....2.2.5$$



รูปที่ 2.2.3 ลักษณะของช่องวิเคราะห์บางชนิด

ส่วนที่เรียวลงไปตามขอบของช่องวิเคราะห์ มีผลทำให้เกิดการเลื่อนออกไปแบบเป็นคาบของเฟรมวิเคราะห์ตามสัญญาณอินพุท แต่ไม่มีผลกระทบต่อพารามิเตอร์ของเสียงที่เนื่องจากขอบเขตของพิทช์ (pitch periodic boundaries) หรือการเปลี่ยนแปลงอื่นๆ ที่มีลักษณะเดียวกันในสัญญาณเสียง

4) การวิเคราะห์ออโตคอรีเลชันในช่วงเวลาสั้น (Short-time Autocorrelation Analysis)

ออโตคอรีเลชันของสัญญาณเสียง คือค่าของ*สเปคตรอลแมกนิจูด* (spectral magnitude) ยกกำลังสอง ซึ่งมีความสัมพันธ์กับส่วนกลับของ*การแปลงฟูเรีย* (Fourier transform) ของพลังงานในรูปสเปคตรัม ออโตคอรีเลชันจะให้ข้อมูลเกี่ยวกับฮาร์โมนิคของสัญญาณเสียง ***แอมปลิจูดของฟอร์แมนท์*** ตลอดจนแสดงถึงความเป็นคาบของสัญญาณโดยไม่เกี่ยวข้องกับเรื่องของเฟส ดังนั้นจึงสามารถนำไปประยุกต์ใช้ในด้านการหาคาบของพิทช์ การดูว่าสัญญาณเสียงช่วงนั้นเป็นเสียงก้องหรือไม่ และการประยุกต์ใช้เทคนิคการทำนายเชิงเส้น ดังที่จะกล่าวต่อไป ออโตคอรีเลชันฟังก์ชันนิยามได้ดังนี้

$$\phi(m) = \frac{1}{N}\sum_{n=0}^{N-m} s(n)s(n+m) \qquad \text{สำหรับ } 0 \le m \le M_0 - 1 \qquad \text{..... 2.2.6}$$

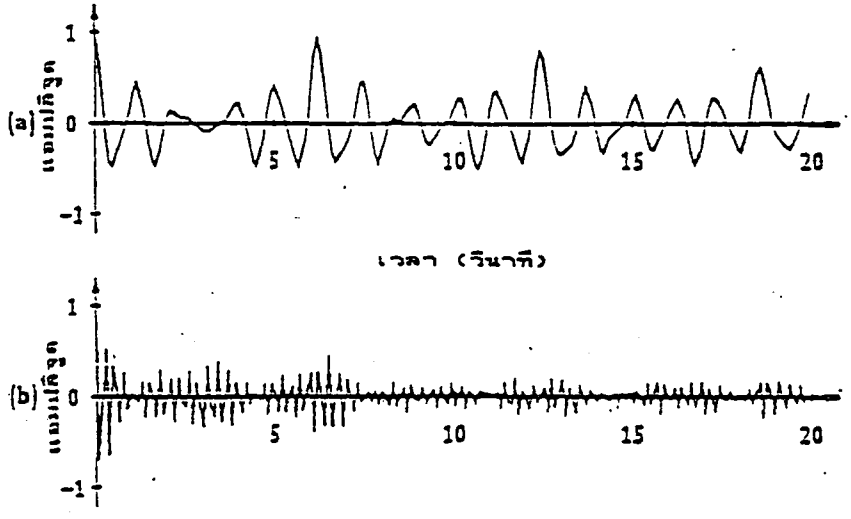
นั่นคือการรวมผลของการคูณสัญญาณตัวอย่างเข้ากับสัญญาณก่อนหน้า ภายในจำนวนตัวอย่าง ที่จำกัดจาก 0 ถึง N-m และแปลงให้เป็นค่าบรรทัดฐาน (normalized) โดยการหารด้วยจำนวนตัวอย่าง N โดยที่ M_0 เป็นค่าของการ lag ที่มากที่สุดที่จะให้มีได้สำหรับตัวอย่างที่กำลังพิจารณา เช่น ถ้าต้องการใช้สัญญาณที่เป็นคาบ P ค่า M_0 ก็ควรมีค่ามากกว่า P แต่ทั้งนี้ค่า m จะมีค่าอยู่ในช่วงเท่าไร ขึ้นอยู่กับการนำไปประยุกต์ใช้ในด้านต่างๆ ด้วย เช่น ในกรณีของเทคนิคการทำนายเชิงเส้น ค่า m จะมีค่าอยู่ในช่วง 0 ถึง 16 หรือตามจำนวนลำดับ (order) ของสมการดังที่จะกล่าวในเรื่องการทำนายเชิงเส้น ส่วนการประยุกต์ด้านความถี่พื้นฐาน อาจจะใช้ค่า m เป็นค่าประมาณเท่ากับคาบของพิทช์ที่คิดเป็นจำนวนครั้งของการคัดแต่ละตัวอย่าง แต่ถ้าในกรณีสัญญาณไม่เป็นคาบ อาจจะเริ่มจากคาบที่สั้นสุดที่เป็นไปได้คือ 3 มิลลิวินาที (ในกรณีเสียงของผู้หญิง) จนถึงคาบที่ยาวที่สุดคือ 20 มิลลิวินาที (ในกรณีเสียงของผู้ชาย) นั่นคือในกรณีที่คัดด้วยความถี่ 10 kHz หรือคัดทุกๆ 0.1 มิลลิวินาที ต้องคำนวณค่าออโตคอรีเลชันประมาณ 170 ครั้งขึ้นไปในแต่ละเฟรม ดังนี้ เป็นต้น จากสมการ 2.2.6 สามารถพิสูจน์ได้ว่า

$$\phi(m) = \phi(-m) \qquad \text{..... 2.2.7}$$

และฟังก์ชันออโตคอรีเลชันจะมีค่าสูงสุดเมื่อ m=0 และค่า $\phi(0)$ จะมีค่าเท่ากับค่าพลังงานของสัญญาณ ในกรณีที่สัญญาณมีลักษณะเป็นคาบที่มีคาบเท่ากับจำนวน P ตัวอย่าง จะได้ฟังก์ชันออโตคอรีเลชันของสัญญาณมีคาบเท่ากับ P เช่นกัน

$$\phi(m) = \phi(m+P) \qquad \text{..... 2.2.8}$$

หรือจะกล่าวได้ว่าค่าสูงสุดของ $\phi(m)$ จะเกิดขึ้นเมื่อ m = 0 , $\pm P$, $\pm 2P$, $\pm 3P$, ...



รูปที่ 2.2.4 ตัวอย่างแสดงฟังก์ชันออโตคอรีเลชัน
a) เสียงก้อง b) เสียงไม่ก้อง

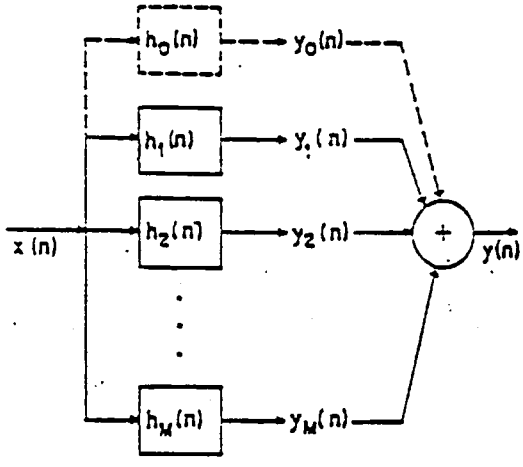
2.2.2 <u>การวิเคราะห์ในโดเมนความถี่</u> (Frequency Domain)

การวิเคราะห์ทางสเปคตรัมของสัญญาณเสียง ก็คือ การวิเคราะห์หรือพิจารณาสัญญาณเสียงในด้านของโดเมนความถี่นั่นเอง เหตุที่ต้องมีการวิเคราะห์ทางด้านสเปคตรัมนั้นก็เนื่องมาจากพารามิเตอร์บางตัวในการประมวลผลสัญญาณเสียงนั้น ส่วนใหญ่จะสามารถหาได้จากโดเมนความถี่ และสัญญาณเสียงที่สร้างออกมาจากการเปลี่ยนแปลงรูปร่างของช่องทางเดินเสียงของคนนั้น ก็จะสามารถพิจารณาเป็นรูปแบบในโดเมนของความถี่จะง่ายและคงที่กว่าการพิจารณาในโดเมนเวลาเป็นต้น

การวิเคราะห์ทางสเปคตรัมในช่วงเวลาสั้นๆ ก็เป็นวิธีการดั้งเดิมอันหนึ่งของการประมวลผลสัญญาณเสียงที่สำคัญ ด้วยข้อสมมติฐานที่ว่าสัญญาณเสียงในช่วงเวลายาวๆ จะมีการเปลี่ยนแปลงอยู่ตลอดเวลาไม่มีสภาพของการคงที่ ดังนั้นการทำการแปลงฟูเรียของสัญญาณเสียงช่วงสั้นๆ จะสามารถแทนเป็นค่าสเปคตรัมของสัญญาณเสียงในช่วงเวลาดังกล่าวได้เป็นอย่างดี การพิจารณาสเปคตรัมของสัญญาณเสียงในช่วงเวลาอันสั้นนั้น จะเป็นพื้นฐานของเทคนิคการวิเคราะห์เสียงต่างๆ เช่น เทคนิค channal vocoder , phase vocoder , spectrogram เป็นต้น

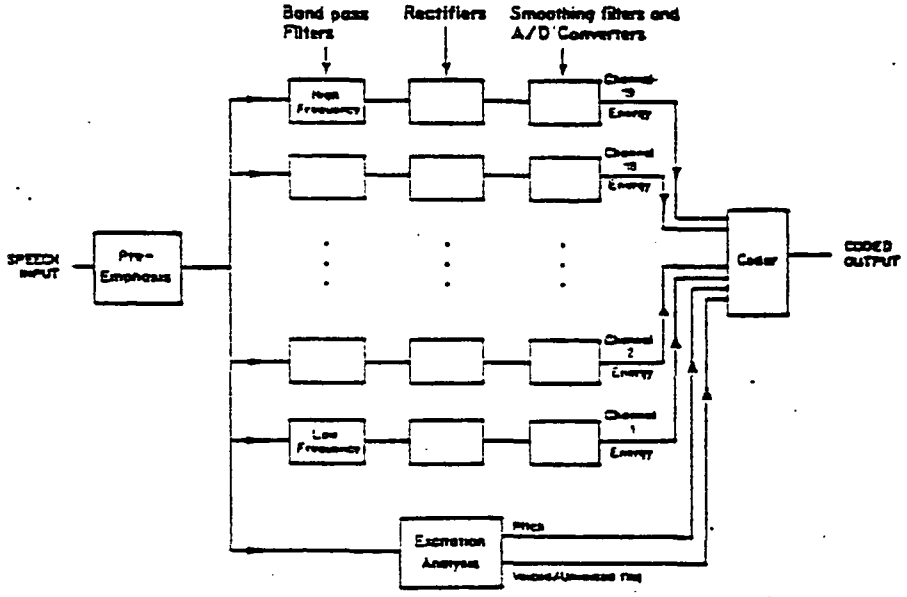
เทคนิคทางด้านสเปคตรัมที่จะนำมาอ้างถึงในที่นี้มีดังนี้

1) การวิเคราะห์ฟิลเตอร์แบงค์ (Filter Bank Analysis)



รูปที่ 2.2.5 การวิเคราะห์ฟิลเตอร์แบงค์

เป็นเทคนิคทางด้านการวิเคราะห์สัญญาณเสียง โดยใช้สเปคตรัมที่นิยมกันมากอันหนึ่ง เนื่องจากสามารถทำได้ง่ายแล้วได้ผลออกมาทันทีทันใดและไม่สิ้นเปลืองเวลามากนัก โดยการใช้กลุ่มของตัวกรองแบบเฉพาะย่านความถี่ผ่าน (bandpass filter) ซึ่งอาจจะเป็นได้ทั้งแบบอนาลอกหรือแบบดิจิตอลก็ได้ ตัวกรองแต่ละอันจะทำการกรองเอาความถี่เฉพาะที่ต่างช่วงกันมาทำการวิเคราะห์ ทั้งนี้ความถี่ทั้งหมดตลอดย่านจะต้องครอบคลุมแถบความถี่ที่สำคัญของสัญญาณอินพุท เทคนิคอันนี้เป็นพื้นฐานของระบบการติดต่อสื่อสารข้อมูล เช่น เทคนิคของแชลแนลโวโคดเดอร์ (channel vocoder) หรือเฟสโวโคดเดอร์(phase vocoder) เป็นต้น



รูปที่ 2.2.6 แชลแนลโวโคดเดอร์

2) การวิเคราะห์ฟูเรียในช่วงเวลาสั้นๆ (Short-time Fourier Analysis)

เป็นวิธีการดั้งเดิมของการวิเคราะห์สัญญาณทางด้านสเปคตรัม การวิเคราะห์แบบฟูเรียนี้จะสามารถแทนค่าสัญญาณเสียงออกมาได้ในเทอมของ แอมปลิจูดและเฟสซึ่งก็เป็นฟังก์ชันอันหนึ่งของความถี่ ถ้าเรามองช่องทางเดินเสียงของคนเป็นระบบเชิงเส้นระบบหนึ่ง การแปลงแบบฟูเรียของสัญญาณเสียงก็คือ ผลลัพธ์ที่เกิดจากการแปลงของการกระตุ้นจากการสั่นของเส้นเสียงกับการตอบสนองภายในช่องทางเดินเสียงนั่นเอง นิยามของการแปลงฟูเรียของสัญญาณ g(t) ใดๆ เป็นดังนี้

$$g(t) = df \qquad \text{..... 2.2.9}$$

$$G(f) = \int_{-\infty}^{\infty} g(t)e^{-j2\pi ft}\,dt \qquad \text{..... 2.2.10}$$

เป็นการแปลงระหว่างโดเมนเวลาที่ต่อเนื่อง กับโดเมนความถี่ที่ต่อเนื่อง จะสังเกตได้ว่าในบางครั้งจะมีการทำให้เป็นบรรทัดฐานกับสมการทั้งสอง โดยการคูณค่า $1/2\pi$ เข้าไปซึ่งจำเป็นในกรณีที่ตัวแปลความถี่มีหน่วยเป็น radians/s แต่ในรูปแบบในที่นี้เราจะใช้หน่วยเป็น Hz แทน

ฟูเรียซีรีส์(Fourier Series) คือการปฏิบัติการกับรูปคลื่นที่มีลักษณะเป็นคาบตามแกนเวลา ถ้ารูปคลื่นมีคาบอยู่ในช่วง [0,b) จะได้ฟังก์ชันที่มีนิยามดังนี้

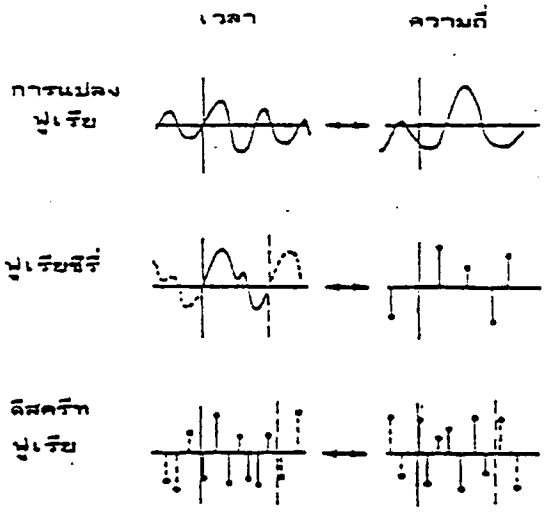
$$g(t)=\sum_{r=-\infty}^{\infty}G(r)e^{+j2\pi rt/b} \qquad \text{..... 2.2.11}$$

$$G(r)=\frac{1}{b}\int_{0}^{b}g(t)e^{-j2\pi rt/b}dt \qquad \text{..... 2.2.12}$$

ฟูเรียซีรีส์เป็นการแปลงระหว่างฟังก์ชันที่เป็นคาบในโดเมนเวลากับฟังก์ชันแบบไม่ต่อเนื่องในโดเมนความถี่ ถ้าเราทำการแปลงรูปคลื่นในโดเมนเวลาที่มีลักษณะเป็นฟังก์ชันแบบไม่ต่อเนื่องและมีช่วงที่จำกัด เราจะได้รูปคลื่นในโดเมนของความถี่ซึ่งมีช่วงที่จำกัดมีลักษณะเป็นคาบและเป็นฟังก์ชันที่ไม่ต่อเนื่อง ลักษณะการแปลงแบบนี้ก็คือ การแปลงแบบดีสครีทฟูเรีย (Discrete Fourier Transform) ซึ่งก็คือการทำฟังก์ชันที่เป็นคาบแบบไม่ต่อเนื่องในโดเมนเวลาไปเป็นฟังก์ชันที่เป็นคาบและไม่ต่อเนื่องในโดเมนความถี่นั่นเอง โดยมีนิยามกำหนดไว้ดังนี้

$$g(n)=\frac{1}{N}\sum_{r=0}^{N-1}G(r)e^{+j2\pi rn/N} \qquad \text{..... 2.2.13}$$

$$G(r)=\sum_{n=0}^{N-1}g(n)e^{-j2\pi rn/N} \qquad \text{..... 2.2.14}$$



รูปที่ 2.2.7 รูปแสดง การแปลงฟูเรีย ฟูเรียซีรีส์ และการแปลงดีสครีทฟูเรีย

3) การแสดงสเปคตรัมของเสียง

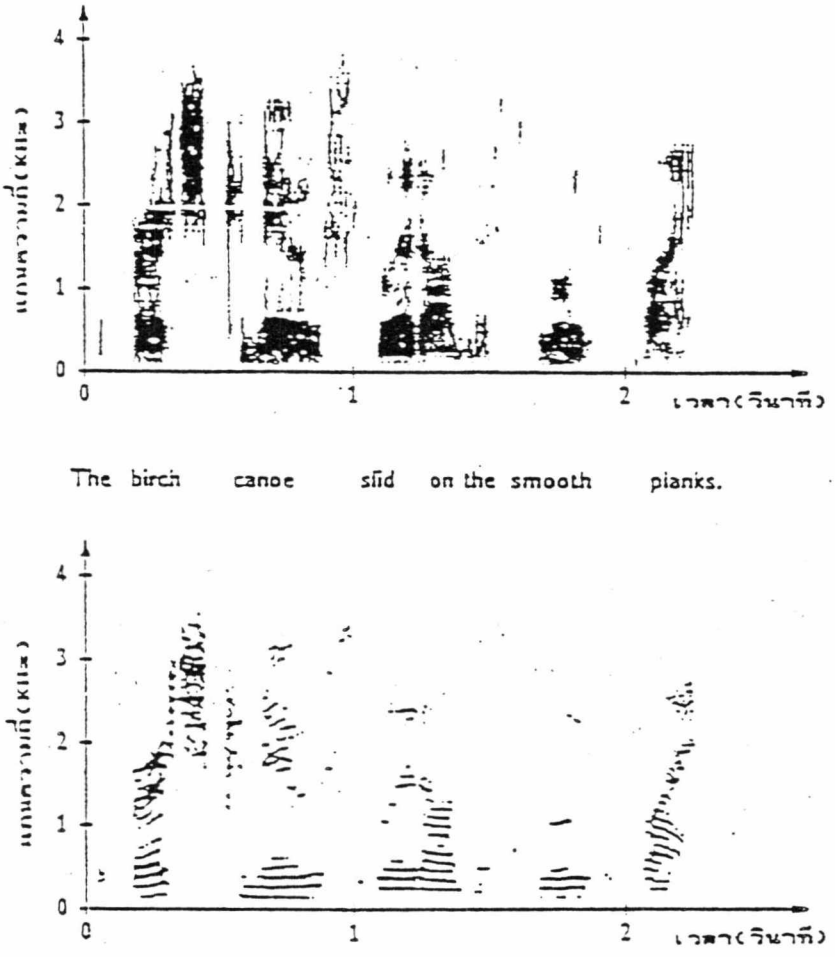
อุปกรณ์สำคัญในการวิเคราะห์สัญญาณเสียงซึ่งมีขึ้นเมื่อประมาณ 40 ปีที่แล้วก็คือ สเปคโตร-แกรม (spectrogram) หรือ ซาวด์เปคโตรกราฟ (sound spectrograph) ซึ่งจะแสดงให้เห็นถึงลักษณะเฉพาะของสัญญาณเสียงในรูปสามมิติภายในช่วงเวลาสั้นๆ ประมาณ 2-3 วินาที โดยการใช้การแปลงฟูเรียในช่วงเวลาสั้นๆ ของสัญญาณเสียงมาพล็อตรูปกราฟโดยมีแกนนอนเป็นแกนของเวลา และแกนตั้งเป็นแกนของความถี่ (ω จาก 0 ถึง π) และแสดงค่าของแมกนิจูด ด้วยความเข้มของการพิมพ์ออกมาดังรูปที่ 2.2.8

สเปคโตรแกรมแบบช่วงการกรองกว้าง (wideband spectrogram) จะแสดงให้เห็นถึงคาบของพิทช์ออกมาในแนวตั้ง ซึ่งจะสัมพันธ์กับช่วงที่แอมปลิจูดของเสียงมีค่ามากๆ ขณะที่เส้นเสียงกำลังจะปิด ดังนั้นในกรณีของเสียงก้องเราสามารถที่จะตรวจพบได้โดยง่ายจากการค้นหาเส้นในแนวตั้งที่มีลักษณะเป็นคาบนี้ การวิเคราะห์ความถี่แบบฟอร์แมนท์และแอมปลิจูดของฮาร์โมนิกภายใต้ ฟอร์-แมนท์ แต่ละฟอร์แมนท์จะอยู่ในช่วงแถบความถี่ประมาณ 300 Hz แสดงให้เห็นเป็นแถบสีดำครอบคลุมในแต่ละฟอร์แมนท์ทางแนวตั้ง ความถี่ตรงกึ่งกลางของแต่ละแถบสีดำก็คือความถี่ฟอร์แมนท์โดยประมาณนั่นเอง

สเปคโตรแกรมแบบช่วงการกรองแคบ (narrowband spectrogram) จะแสดงให้เห็นถึงฮาร์โม-นิกแต่ละอันแทนคาบของพิทช์ และมีประโยชน์ค่อนข้างน้อยในการวิเคราะห์สัญญาณเสียงออกเป็นส่วนๆ เนื่องจากให้รายละเอียดในด้านความถี่น้อยมาก แบบนี้จะใช้การวิเคราะห์ความถี่ในช่วง 45 Hz

เนื่องจากแอมปลิจูดของเสียงจะมีค่าลดลงประมาณ -6 dB ต่ออ็อกเตฟ (octave) เมื่อสัญ-ญาณเสียงออกมาจากริมฝีปาก และรายละเอียดด้านความถี่สูงจะปรากฏขึ้นมารบกวนเพิ่มขึ้นตาม ดังนั้นจึงมีการทำพรี-เอมฟาไซซิง (pre-emphasizing) กับสัญญาณเสียงโดยการทำดิฟเฟอร์เรนเชียลสัญญาณเสียงที่เป็นอนาลอก $S_a(t)$ ก่อนผ่านเข้าไป A/D เพื่อแปลงเป็นสัญญาณดิจิตอลอีกทีหนึ่ง หรือโดยการดิฟเฟอร์เรนเชียลสัญญาณที่ไม่ต่อเนื่องทางด้านเวลา $S(n) = S_a(nT)$ เพื่อเป็นการขดเชยลดสัญญาณความถี่สูงลงไป ถ้าสัญญาณเสียงที่มีการสังเคราะห์เสียงขึ้นมาจากข้อมูลที่ผ่านการ ***พรี-เอมฟาไซซิง*** แล้วนั้นในขั้นตอนสุดท้ายของการสังเคราะห์เสียงจะต้องทำการในทำนองกลับกันคือทำ ***ดี-เอมฟาไซซิง*** (de-emphasizing) โดยทำอินทิเกรชันกลับคืนด้วยเพื่อให้ได้คุณสมบัติทางไดนามิกเรนจ์ (dynamic range) กลับคืนมา ในการประมวลผลทางดิจิตอลสามารถทำการพรี-เอมฟาไซซิงด้วยสมการ

$$y(n) = s(n) - as(n-1) \qquad \text{..... 2.2.15}$$



รูปที่ 2.2.8 ภาพของสเปคโตรแกรม a) ไวด์แบนด์ b) นาโรแบนด์

โดยค่า a จะต้องมีค่าระหว่าง 0.9 ถึง 1 ตามปกติแล้วการทำพรี-เอมฟาไซซิงนี้จะใช้เฉพาะกับเสียงก้องเท่านั้น ทั้งนี้เนื่องจากเสียงไม่ก้องจะมีค่าสเปคตรัมราบเรียบที่ความถี่สูงๆ อยู่แล้ว ดังนั้นค่า a จะมีค่าเข้าใกล้ 0 ในกรณีเสียงไม่ก้องซึ่งเราสามารถจะทำให้ค่า a มีผลลัพธ์ออกมาใกล้เคียงตามที่กำหนดนี้ได้ โดยการให้ค่า a มีค่าเกี่ยวข้องกับค่าออโตคอรีเลชันของสัญญาณเสียงดังนี้

$$a = R(1)/R(0) \qquad \text{..... 2.2.16}$$

โดยค่า R(1) คือค่าคอรีเลชันของสัญญาณเสียงซึ่งมีการหน่วงเวลาไปหนึ่งครั้งของตัวอย่าง ค่า R(0) คือค่าคอรีเลชั่นของสัญญาณซึ่งไม่มีการหน่วงเวลา ซึ่งในกรณีของเสียงก้องแล้วการหาค่าคอรีเลชั่นของสัญญาณจะทำให้ได้ค่า R(1) มีค่าใกล้เคียงกับค่า R(0) มากดังนั้นก็จะได้ค่า a เข้าใกล้กับ 1 ส่วนในกรณีของเสียงไม่ก้องนั้นค่าคอรีเลชั่นของสัญญาณตัวอย่างหนึ่งกับตัวอย่างถัดไปจะมีค่าน้อยหรือไม่มีเลย ดังนั้นอัตราส่วนของ R(1) กับ R(0) จะมีค่าเข้าใกล้ศูนย์

2.2.3 <u>การหาคาบของพิทช์</u> (Pitch Detection)

อัลกอริธึมในการหาคาบของสัญญาณเสียงมีอยู่หลายวิธีด้วยกัน เริ่มตั้งแต่ประมาณปี ค.ศ. 1976 ที่มีชื่อเสียงก็เช่นวิธีการหาพิทช์ของ Sondhi และอัลกอริธึม SIFT (Simplified Inverse Filter Tracking) ของ Markel ในปี ค.ศ. 1972 เป็นต้น แต่ละวิธีการก็จะออกแบบเฉพาะตามความสามารถของฮาร์ดแวร์ของตนเพื่อทำให้เกิดการประมวลผลได้แบบทันทีทันใด ทั้งนี้เนื่องจากว่าอัลกอริธึมในการหาพิทช์นี้จะต้องใช้หลักการทางตรรกหลายขั้นตอน ทำให้ต้องเสียเวลาในการคำนวณมากถึงแม้ว่าการประยุกต์ในบางรูปแบบของอัลกอริธึมนี้จะสามารถประยุกต์โดยใช้โปรแกรมทั้งหมด แต่ก็จะต้องมีการใช้ตัวประมวลผลพิเศษมาช่วยเพื่อผลในการตอบสนองให้เป็นไปได้อย่างทันทีทันใด

ปัญหาสำคัญประการหนึ่งในการหาความถูกต้องของพิทช์ของเสียงในช่วงแถบความถี่ที่กว้างของสัญญาณนี้ก็คือ ผลของโครงสร้างฟอร์แมนท์ (formant structure) ที่มีการวัดสัมพันธ์กับคาบของรูปคลื่นสัญญาณ ดังนั้นเพื่อเพิ่มความถูกต้องในการหาพิทช์ของเสียงจะต้องทำการลดผลของฟอร์-แมนท์นี้ลงให้มากที่สุดเท่าที่จะทำได้ เทคนิคในการขจัดรูปร่างของสเปคตรัมในรูปคลื่นที่เกี่ยวข้องกับฟอร์แมนท์นี้เรียกว่า การทำสเปคตรัมให้ราบเรียบ (spectral flattening) Sondhi ได้เสนอวิธีการทำสเปคตรัมให้ราบเรียบอยู่สองวิธีคือ

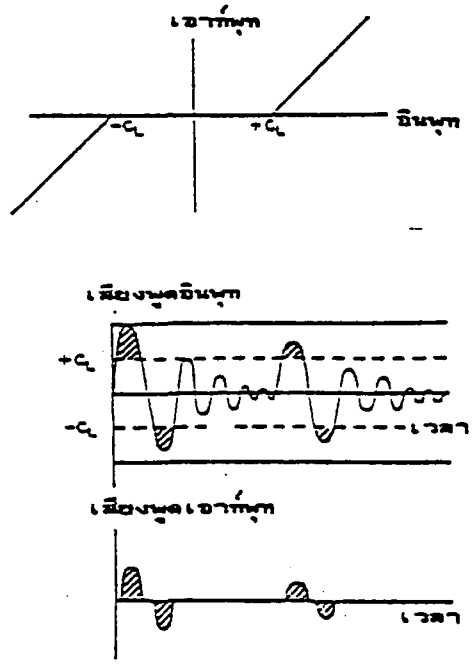
*1) **วิธีฟิลเตอร์แบงค์*** (filter bank method) วิธีการนี้คือการนำสัญญาณเสียงไปผ่านกลุ่มของตัวกรองสัญญาณแบบแถบความถี่ผ่าน (band pass filter) ที่มีแถบของความถี่แตกต่างกันไปตลอดย่านความถี่ของสัญญาณเสียง สัญญาณสุดท้ายที่ผ่านออกมาจากตัวกรองแต่ละตัวจะถูกทำให้เป็นบรรทัดฐานกับค่าแอมปลิจูดให้เป็น 1 หรือเป็นการทำสเปคตรัมให้ราบเรียบโดยการนำมาหารด้วยค่ากำลังงานในช่วงเวลาสั้นๆ สัญญาณรวมทั้งหมดที่มีการทำให้สเปคตรัมราบเรียบจะได้มาจาก นำสัญญาณสุดท้ายจากตัวกรองแต่ละตัวมารวมเข้าด้วยการหน่วงเวลาที่เหมาะสม

แม้ว่าวิธีนี้จะใช้ได้ดีในหลายๆ กรณีก็ตามแต่ก็ยังมีข้อเสียอยู่หลายอย่าง เช่น ต้องใช้ฮาร์ดแวร์ในส่วนที่เป็นวงจรตัวกรองและวงจรปรับระดับสัญญาณแต่ละความถี่ต่างๆ เป็นจำนวนมาก และในกรณีที่เกิดมีฮาร์โมนิกที่ไม่มีพิทช์อยู่ในตัวกรองในแถบความถี่ใดความถี่หนึ่งขึ้นมาจะมีผลทำให้สัญญาณที่ออกมาจากตัวกรองนั้นมีระดับสัญญาณที่ต่ำมาก แต่อย่างไรก็ตามการปรับระดับสัญญาณที่ความถี่นั้นก็ยังเป็นสิ่งที่จำเป็นอยู่ แต่จะมีผลในด้านการขจัดสัญญาณรบกวนซึ่งมีขนาดใหญ่เมื่อเทียบกับขนาดของสัญญาณจริงมากกว่าที่จะมีผลในด้านการตรวจจับพิทช์ของสัญญาณ

*2) **เทคนิคเซนเตอร์คลิปปิง*** (center clipping technique) วิธีการนี้คือการตรวจจับระดับของสัญญาณว่า ถ้ามีค่าต่ำกว่าระดับคลิปปิงที่กำหนดไว้เมื่อไรก็จะกำหนดให้สัญญาณตรงนั้นมีค่าเป็นศูนย์ ส่วนสัญญาณที่ยังมีค่าสูงกว่าระดับคลิปปิงก็จะเอาค่ามาหักลบด้วยค่าระดับคลิปปิงเหลือแต่ส่วนที่เกินไปจากระดับคลิปปิงเท่านั้น รูปที่ 2.3.9 แสดงถึงลักษณะสมบัติของสัญญาณอินพุท-เอาท์-พุทที่นำมาทำกระบวนการเซนเตอร์คลิปปิง ซึ่งจะเห็นได้ว่าถ้ามีการกำหนดระดับคลิปปิงที่เหมาะสมแล้วโครงสร้างของรูปคลื่นที่เป็นส่วนของฟอร์แมนท์ต่างๆ จะถูกกำจัดออกไปได้เกือบทั้งหมด ซึ่งจะมีผลทำให้สัญญาณที่สเปคตรัมราบเรียบที่ได้ออกมามองเห็นคาบของมันได้ง่ายกว่าสัญญาณที่ยังไม่ได้ผ่านกระบวนการสเปคตรัมราบเรียบ

วิธีการทำให้สเปคตรัมของสัญญาณราบเรียบทั้งสองวิธีข้างต้นนั้น วิธีการเซนเตอร์คลิปปิงจะเป็นวิธีที่มีการนำมาประยุกต์ใช้ได้ค่อนข้างจะสะดวกกว่า โดยการนำเอาฮาร์ดแวร์มาประยุกต์ใช้แทนอัลกอริธึมบางส่วนด้วย เพื่อให้ผลของการตอบสนองเป็นไปอย่างทันทีทันใด

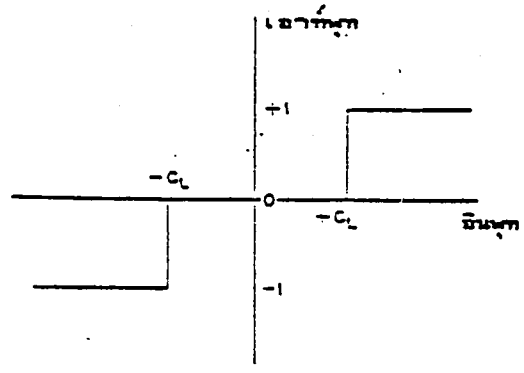
อัลกอริธึมของการหาพิทช์ของสัญญาณเสียงนี้ จะเริ่มจากการนำสัญญาณเสียงผ่านเข้าตัวกรองความถี่แบบต่ำผ่านที่มีช่วงกว้างของแถบความถี่อยู่ในช่วง 900 เฮิร์ตซ์ จากนั้นจะนำมาแปลงให้เป็นสัญญาณแบบดิจิตอลแล้วตัดแบ่งออกเป็นเฟรมๆ เฟรมละ 30 มิลลิวินาทีเพื่อนำไปวิเคราะห์และประมวลผลต่อไป



รูปที่ 2.2.9 ลักษณะสมบัติของสัญญาณเมื่อผ่านกระบวนการเซนเตอร์คลิปปิง

ขั้นแรกของการประมวลผลสำหรับแต่ละเฟรมนั้นก็จะทำการคำนวณระดับคลิปปิงของแต่ละเฟรม เนื่องจากเสียงเป็นสัญญาณที่มีช่วงของการเปลี่ยนแปลงกว้างมากดังนั้นการเลือกระดับคลิป-ปิง จะต้องทำด้วยความระมัดระวัง เพื่อป้องกันการสูญหายของข้อมูลที่สำคัญบางส่วนของรูปคลื่นในขณะที่รูปคลื่นมีแอมปลิจูดที่กำลังมีขนาดเพิ่มขึ้น หรือกำลังลดลงในแต่ละเฟรมนั้นๆ กรณีแบบนี้จะเกิดขึ้นในช่วงที่เสียงกำลังจะเริ่มต้นหรือกำลังจะจบลง หรืออาจจะเป็นตรงรอยต่อของช่วงการเปลี่ยน-แปลงของเสียง สมมติว่าเราเลือกให้ระดับคลิปปิงมีค่าเป็น C_L สัญญาณเสียงแต่ละเฟรมที่มีขนาด 30 มิลลิวินาทีจะถูกแบ่งออกเป็น 3 ส่วนๆ ละ 10 มิลลิวินาที ในส่วนของ 10 มิลลิวินาทีแรกกับส่วนของ 10 มิลลิวินาทีสุดท้ายอัลกอริธึมจะทำการหาค่าสมบูรณ์ที่จุดยอดสูงสุดเก็บเอาไว้สำหรับของแต่ละส่วน ค่าระดับคลิปปิงก็จะกำหนดเอาจากค่าเปอร์เซนต์ที่แน่นอนของค่าสมบูรณ์ที่จุดยอดสูงสุดที่น้อยที่สุดระหว่างสองค่านั้น ตามวิธีการของ Sondhi นั้นก็จะกำหนดค่าระดับคลิปปิงไว้ที่ 30 เปอร์เซนต์ของค่าสมบูรณ์ที่จุดยอด ซึ่งจะใช้เป็นระดับอ้างอิงกับข้อมูลของเสียงตลอดช่วง 30 มิลลิวินาทีเลย แต่ทั้งนี้เพื่อเป็นการหลีกเลี่ยงการสูญหายของข้อมูลของเสียงที่ระดับสัญญาณต่ำๆ ก็จำเป็นที่จะต้องใช้คลิปปิงให้ต่ำตามไปด้วย นั่นหมายถึงว่า การที่จะเลือกระดับคลิปปิงให้เหมาะสมกับระดับของสัญญาณเสียงเพื่อหลีกเลี่ยงปัญหาเหล่านี้เป็นเรื่องที่ไม่ง่ายนัก ไม่สามารถกำหนดตายตัวเป็นกฎเกณฑ์ที่ลงตัวได้

หลังจากการหาระดับคลิปปิงได้แล้ว จะนำสัญญาณมาตัดช่วงที่อยู่ตรงกลางระหว่างระดับคลิปปิงด้านบวกและด้านลบทิ้งไป ซึ่งลักษณะนี้เรียกว่า ***เซนเตอร์คลิปปิง*** (center clipping) ดังนั้นผลที่ได้ออกมาก็จะเป็นไปได้ 3 ค่าคือ ค่า +1,-1 สำหรับตัวอย่างสัญญาณที่มีค่าเกินกว่าระดับคลิปปิงด้านบวกและต่ำกว่าด้านลบตามลำดับ และค่า 0 สำหรับสัญญาณที่มีค่าอยู่ภายในช่วงระดับคลิปปิงระหว่างด้านบวกและด้านลบ จะเห็นได้ว่าไม่ว่าค่าของสัญญาณตัวอย่างจะมีค่ามากหรือน้อยแค่ไหนก็จะถูกทำให้มีค่าเท่ากับหนึ่งเหมือนกันหมด (ในแต่ละด้านบวกหรือลบ) ถ้าเกินค่าของระดับคลิปปิงแล้ว นั่นคือเป็นการทำการตัดยอดคลื่นที่มีค่ามากๆ (in-finite peak clipped) ด้วย รูปที่ 2.3.10 จะแสดงให้เห็นถึงลักษณะสมบัติของสัญญาณเสียงขาเข้าและขาออกเมื่อผ่านการเซนเตอร์คลิปปิง และการตัดยอดคลื่นที่มีค่ามากๆ ด้วย จากนั้นก็จะเอาผลมาทำการออโตคอรีเลชันต่อไป ผลของการเซนเตอร์คลิปปิงและการตัดยอดคลื่นนี้จะเป็นการลดความยุ่งยากในการคำนวณของขั้นออโตคอรีเลชันอย่างมาก เพราะไม่ต้องทำการคูณหรือบวกในการคำนวณฟังก์ชันออโตคอรีเลชันของสัญญาณที่ผ่านการตัดยอดคลื่นมานี้เลย



รูปที่ 2.2.10 ลักษณะสมบัติของเทคนิคเซนเตอร์คลิปปิง

ขั้นตอนของการประมวลผลต่อไปโดยการคำนวณออโตคอรีเลชันสำหรับสัญญาณใน 1 เฟรม หรือ 30 มิลลิวินาที ที่ผ่านการคลิปปิงแล้วนี้จะสามารถกำหนดได้เป็น

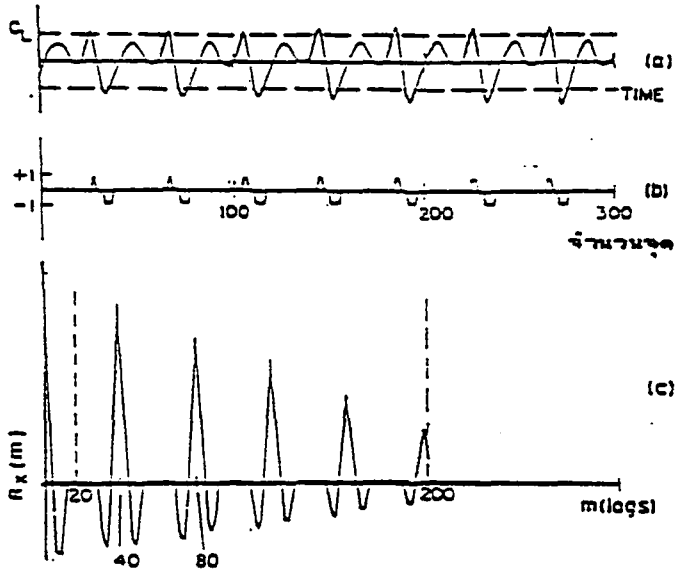
$$R_x(m) = \sum_{n=0}^{299-m} x(n)x(n+m) \qquad \text{สำหรับ } m = M_i, M_{i+1}, \ldots, M_f \qquad \ldots\ldots 2.2.17$$

โดยที่ M_i คือค่าเริ่มต้นและ M_f คือค่าสุดท้ายสำหรับการคำนวณฟังก์ชันออโตคอรีเลชัน ซึ่งตามปกติแล้วจะกำหนดให้ M_i = 25 และ M_f = 200 สำหรับเสียงที่มีช่วงคาบของเสียงอยู่ภายในช่วง 400 เฮิร์ตซ์ จนต่ำลงมาถึง 50 เฮิร์ตซ์ จากสมการ 2.2.17 เทอมแต่ละเทอมที่อยู่ในรูปของ x(n)x(n+m) ซึ่งค่าของ x(n) จะเป็นได้เพียงค่า +1,-1,0 เท่านั้น ดังนั้นเราสามารถเขียนเทอมนี้ใหม่ในรูปแบบดังนี้

$$x(n)x(m) = 0 \text{ ถ้า } x(n) = 0 \text{ หรือถ้า } x(n+m) = 0 \qquad \text{..... 2.2.18}$$
$$1 \text{ ถ้า } x(n) = x(n+m) = \pm 1$$
$$-1 \text{ ถ้า } x(n) = -x(n+m) = \pm 1$$

นั่นคือเราสามารถใช้เหตุผลทางตรรกมาใช้คำนวณเทอมแต่ละเทอมในฟังก์ชันออโตคอรีเลชันได้ ซึ่งถ้าทำการประยุกต์โดยใช้ฮาร์ดแวร์แล้วก็สามารถจะใช้วงจรนับธรรมดาแบบง่ายๆ มาใช้ในการหาออโตคอรีเลชันก็ได้แล้ว รูปที่ 2.2.11 เป็นตัวอย่างของการประมวลผลของสัญญาณใน 1 เฟรม รูปบนจะเป็นสัญญาณที่ผ่านวงจรกรองแบบต่ำผ่านมาแล้วและจะทำการเซนเตอร์คลิปปิงด้วยระดับสัญญาณคลิปปิงเท่ากับ $\pm C_L$ รูปกลางหลังจากการทำเซนเตอร์คลิปปิงแล้วและรูปล่างแสดงถึงฟังก์ชันออโตคอรีเลชันของสัญญาณเสียงที่ผ่านการคลิปปิงแล้ว ช่วงคาบหรือพิทช์ของเสียงนี้ถ้าดูจากรูปล่างก็จะพบว่ามีค่าอยู่ในช่วงของแนวเส้นประคือช่วงที่ m=20 จนถึง m=200 ซึ่งถ้าหาคาบของเสียงในเฟรมนี้ก็จะได้ค่าเท่ากับ 40 ตัวอย่างสัญญาณหรือ 4 มิลลิวินาที หรืออาจจะพูดได้ว่ามีความถี่ของพิทช์เท่ากับ 250 เฮิร์ตซ์

เราควรจะระลึกไว้เสมอว่าในการคำนวณฟังก์ชันออโตคอรีเลชันจากสมการที่ 2.2.17 นั้นเราสมมติว่า ตัวอย่างของสัญญาณตัวอย่างที่อยู่นอกเหนือไปจาก 30 มิลลิวินาทีที่เรากำลังพิจารณาอยู่นี้ถือว่าเป็นศูนย์หมด ผลกระทบอันนี้จะเป็นการถ่วงน้ำหนักฟังก์ชันออโตคอรีเลชันด้วยการลาดเอียงแบบเชิงเส้น (linear tapper) โดยเริ่มจากค่าเป็น 1 ที่จุด m=0 ไปจนถึงค่าเป็น 0 ที่จุด m=300 ผลกระทบอันนี้จะเห็นได้จากรูปที่ 2.2.11 ล่าง ซึ่งจะเห็นว่าจุดยอดของฟังก์ชันออโตคอรีเลชันจะมีการลาดเอียงลดหลั่นกันลงมาเป็นแบบเชิงเส้นจนถึงศูนย์ ประโยชน์ของการลาดเอียงแบบเชิงเส้นบนฟังก์ชันออโตคอรีเลชันนี้จะมีผลดีต่อจุดยอดแต่ละจุดที่เป็นจุดคาบของพิทช์ เพราะมันจะลดผลของจุดยอดอื่นๆ ที่ไม่ใช่จุดยอดตรงคาบของพิทช์ซึ่งอาจจะมีผลของฟังก์ชันออโตคอรีเลชันเด่นขึ้นมากกว่าจุดยอดตรงคาบของพิทช์จริงๆ ซึ่งทำให้การหาคาบของพิทช์ผิดพลาดได้



รูปที่ 2.2.11 ตัวอย่างการประมวลผลของสัญญาณใน 1 เฟรม

นอกจากนั้นในเรื่องระดับจุดยอดของสัญญาณ จะมีการกำหนดระดับอ้างอิงของสัญญาณในภาวะสงบ (silence level threshold) ซึ่งหามาจากการวัดระดับจุดยอดของสัญญาณในสภาวะสงบเป็นเวลา 50 มิลลิวินาที ระดับอ้างอิงต่ำสุดของสัญญาณในสภาวะสงบนี้จะถูกนำมา เทียบกับระดับสัญญาณในเฟรมนี้เพื่อทำการแยกแยะว่าสัญญาณในเฟรมที่กำลังทำการประมวลผลอยู่นี้เป็นข้อมูลของสัญญาณเสียงหรือไม่ ถ้าเป็นข้อมูลของสภาวะแวดล้อมในสภาวะสงบก็จะไม่ทำการคำนวณหาคาบพิทช์ ดังนั้นระดับอ้างอิงต่ำสุดของสัญญาณแวดล้อมในสภาวะสงบนี้จะต้องทำการเปลี่ยนแปลงค่าให้เหมาะสมตามสภาวะแวดล้อม เพื่อจะได้ทำการแยกแยะสัญญาณเสียงออกจากสัญญาณจากสภาวะแวดล้อมได้อย่างถูกต้อง ถ้าเราสามารถแยกได้แล้วว่าข้อมูลในเฟรมนั้นเป็นข้อ-มูลของเสียงจริงๆ ก็จะนำไปประมวลผลออโตคอรีเลชัน โดยจะทำการค้นหาค่าออโตคอรีเลชันที่มากที่สุดภายในช่วง $m=M_i$ จนถึง $m=M_f$ โดยจะทำการเก็บเอาไว้ทั้งตำแหน่งและค่าของจุดที่มีค่ามากที่สุดนี้ไว้ ถ้าค่าที่จุดนี้มีค่ามากกว่าระดับอ้างอิงต่ำสุดของการจำแนกว่าเป็นเสียงก้องหรือไม่ก้อง ก็จะถือว่าข้อมูลส่วนนี้เป็นข้อมูลของสัญญาณเสียงก้องและค่าคาบของพิทช์ก็จะเริ่มที่ตำแหน่งของจุดนี้ แต่ถ้าน้อยกว่าระดับอ้างอิงต่ำสุดก็จะถือว่าเป็นข้อมูลในช่วงนี้เป็นข้อมูลของสัญญาณเสียงไม่ก้อง

2.3 การหาขอบเขตของสัญญาณ

พื้นฐานของสัญญาณในการรับรู้เสียงพูดกลุ่มคำโดด คือ สัญญาณจะประกอบด้วยส่วนของเสียงพูด อาจมีสัญญาณรบกวน (Noise Signal) นำหน้าและต่อท้ายด้วย ดังนั้นเมื่อต้องการเฉพาะส่วนของสัญญาณ เราจึงจำเป็นต้องมีวิธีการแบ่งแยกส่วนของสัญญาณให้ถูกต้อง กระบวนการที่แยกส่วนของสัญญาณเสียงพูดออกจากส่วนที่เป็นสัญญาณรบกวนเรียกว่า การหาขอบเขตของสัญญาณ (Endpoint Detection) ความจำเป็นของการหาขอบเขตของสัญญาณในการรับรู้เสียงพูดกลุ่มคำโดดเนื่องจาก

ก. ความถูกต้องของการรับรู้เสียงพูด จะขึ้นโดยตรงกับความถูกต้องของการหาขอบเขตของสัญญาณ

ข. การหาขอบเขตสัญญาณที่ถูกต้อง จะทำให้เวลาที่ใช้ในการคำนวณน้อยที่สุด

พารามิเตอร์ที่ใช้ในการหาขอบเขตของสัญญาณ คือ *พลังงานของสัญญาณ* (Energy of Signal) โดยการเปรียบเทียบพลังงานของแต่ละเฟรมกับค่ากำหนด จุดเริ่มต้นของสัญญาณเสียงพูดจะได้จากเฟรมที่มีค่าพลังงานเกินค่ากำหนดเป็นเฟรมแรกและจุดสิ้นสุดจะได้จากเฟรมที่มีค่าพลังงานต่ำกว่าค่ากำหนดเป็นจุดแรกเช่นกัน

อย่างไรก็ตาม ได้มีการปรับปรุงการหาขอบเขตของสัญญาณ โดยอาศัยหลักการพื้นฐานดังกล่าวได้เป็นดังนี้ให้อินพุท คือ อะเรย์ของพลังงาน $R_l(0)$, $l=1,2,...,L$ โดย L เป็นจำนวนของเฟรมทั้งหมด จากนั้นนำอะเรย์ของพลังงานมาผ่านขั้นตอนดังนี้

2.3.1 การปรับปรุงระดับของพลังงาน (Adaptive Level Equalizer)

เป็นขั้นแรกของกระบวนการโดยจะทำการปรับปรุงระดับของข้อมูลเทียบกับค่าล็อกของระดับสัญญาณเบื้องหลัง (Background Signal) ได้ผลลัพธ์ คือ

$$R_l(0) = \log[R_l(0)] - Q \qquad \text{โดย } l = 1,2,...,L \qquad \text{..... 2.3.1}$$

โดย Q เป็นค่าเฉลี่ยของระดับสัญญาณเบื้องหลัง ซึ่งได้จากการหาค่า E_{min} ดังนี้

$$E_{min} = \min_{1 \le l \le L} \{\log[R_l(0)]\} \qquad \text{..... 2.3.2}$$

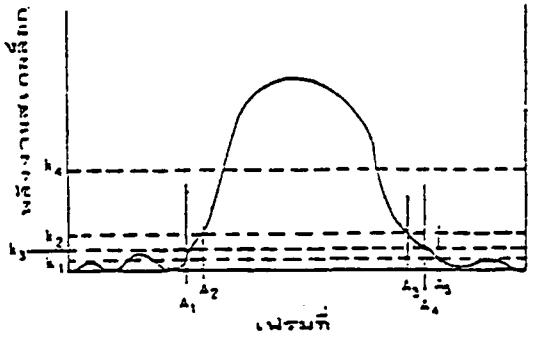
จากนั้นจะทำการสร้างแผนภูมิแท่ง (Histogram) ระหว่าง log[$R_I(0)$] กับค่า I โดยเลือกส่วนที่มีค่าระดับพลังงานสเกลล็อกต่ำกว่า 10 เดซิเบล แล้วจะทำการเลือกจุดเฉลี่ย 3 จุดจากแผนภูมิแท่งนี้ จะได้จุดยอดของแผนภูมิ ซึ่งจะใช้เป็นค่าของ Q นั่นเอง

2.3.2 **<u>การตรวจหารูปคลื่นพลังงาน</u>** (Energy Pulse Detection)

จากเอาท์พุทของการปรับปรุงระดับพลังงาน $R_I(0)$ เราจะทำการกำหนดระดับพลังงานอ้างอิง (Energy Thresholds) 4 ค่า คือ k_1 , k_2 , k_3 และ k_4 ดังรูปที่ 2.31 เพื่อใช้ในการหาขอบเขตของ รูปคลื่นพลังงาน สมมติฐานในการตรวจหารูปคลื่นพลังงาน คือ ในสัญญาณอินพุทที่เข้ามาจะมีรูปคลื่นพลังงานอย่างน้อย 1 รูป

ในการตรวจหารูปคลื่นพลังงาน จะทำจากซ้ายไปขวาโดยการเพิ่มขึ้นของค่า I เมื่อค่า $R_I(0)$ มีค่าเกินจุดอ้างอิงแรก k_1 เฟรมที่ A_1 จะถูกบันทึกไว้ ถ้าค่า $R_I(0)$ มีค่าเกินจุดอ้างอิง k_2 ก่อนที่จะตกกลับมาที่จุดอ้างอิง k_1 จุดเริ่มต้นของรูปคลื่นจะเป็นเฟรมที่ A_1 แต่ถ้าระยะเวลาจากเฟรม A_1 ถึง A_2 มีค่ายาวเกินไป จุดเริ่มต้นของรูปคลื่นจะถูกกำหนดเป็นเฟรมที่ A_2 แทน ส่วนจุดสิ้นสุดก็จะได้ในลักษณะเดียวกัน โดยใช้จุดอ้างอิงคือ จุด k_2 และ k_3 อย่างไรก็ตามถ้าระยะเวลาระหว่างเฟรมที่ A_3 และ A_4 มีค่ายาวเกินไป ซึ่งเกิดจากมีลมหายใจที่ปลายคำ เฟรมที่ A_3 จะเป็นเฟรมสุดท้ายของรูปคลื่นพลังงาน

การวิเคราะห์ทั้งสองขั้นตอนนี้ จะทำจนกว่าไม่พบรูปคลื่นพลังงานแล้ว โดยมีข้อยกเว้นคือ ถ้ารูปคลื่นมีค่าน้อยกว่าจุดอ้างอิง k_4 รูปคลื่นนี้ไม่ถูกนำมาคิด เนื่องจากถือว่าเป็นส่วนหนึ่งของคำ และถ้าระยะเวลาของรูปคลื่นมีค่าน้อยกว่า 5 เฟรม รูปคลื่นนี้จะไม่นำมาคิดเช่นกัน



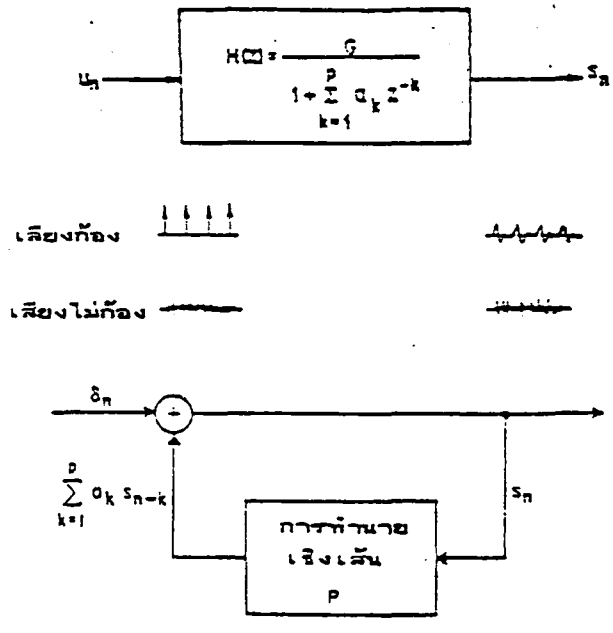
รูปที่ 2.3.1 แสดงถึงการกำหนดจุดอ้างอิงเพื่อหาจุดเริ่มต้นและสิ้นสุดของรูปคลื่นพลังงาน

2.4 การวิเคราะห์โดยใช้เทคนิคการทำนายแบบเชิงเส้น (Linear Predicton Code Analysis)

หลักการทำนายแบบเชิงเส้นปรากฏขึ้นครั้งแรกในหนังสือ Wiener's 1984 บทที่ 2 เรื่อง 'The Linear Prediction for a Single Time Series' จากนั้นมาได้มีการประยุกต์ใช้กันอย่างแพร่หลายในแขนงต่างๆกันไป นักวิจัยกลุ่มแรกที่นำวิธีการทำนายแบบเชิงเส้นมาประยุกต์ใช้ในด้านการวิเคราะห์ และสังเคราะห์เสียงพูดคือ Saito กับ Itakula ในปี ค.ศ. 1966 และ Atal กับ Schroeder ในปี ค.ศ. 1967 จากนั้นเทคนิคการทำนายแบบเชิงเส้นก็กลายเป็นเทคนิคที่นิยมใช้และแพร่หลายเป็นต้นมา

เทคนิคการทำนายแบบเชิงเส้น เป็นเทคนิคอันหนึ่งในการเข้ารหัสเสียงพูดโดยใช้อัตราการเก็บหน่วยความจำน้อย หลักการพื้นฐานของการทำนายแบบเชิงเส้นคือ สามารถประมาณค่าตัวอย่างของสัญญาณเสียง ได้ด้วยตัวอย่างของสัญญาณเสียงที่คัดตัวอย่างมาในอดีตนำมารวมกันแบบเชิงเส้น และด้วยการทำให้ผลต่างกำลังสองของสัญญาณเสียงตัวอย่างที่คัดมาจริง กับสัญญาณเสียงที่ประมาณขึ้นมา มีค่าน้อยที่สุด ทำให้หาค่าสัมประสิทธิ์การทำนาย (Prediction Coefficient) ที่จะนำมาใช้ในสมการรวมแบบเชิงเส้นได้

2.4.1 รูปแบบจำลองของการสร้างสัญญาณเสียงโดยเทคนิคการทำนายแบบเชิงเส้น



รูปที่ 2.4.1 รูปแบบจำลองการกำเนิดเสียงโดยใช้เทคนิคการทำนายแบบเชิงเส้น

จากรูปที่ 2.4.1 แสดงถึงรูปแบบการจำลองของการกำเนิดสัญญาณเสียงโดยใช้เทคนิคแอลพีซีทั้งในโดเมนของเวลา (บน) และในโดเมนความถี่ (ล่าง) ซึ่งในส่วนของตัวกรองดิจิตอลที่แปรตามสัญญาณ (time-varying digital filter) นั้นจะแทนครอบคลุมถึงผลกระทบต่าง ๆ ทางด้านสเปคตรัม เช่น การแผ่กระจายของคลื่นเสียง (radiation) ลักษณะของช่องทางเดินเสียง (vocal tract) และการสั่นของเส้นเสียง (glottal excitation) ดังนั้นจากรูปแบบจำลองดังกล่าวเราสามารถเขียนรูปแบบของการสังเคราะห์เสียงให้เป็นแบบออลโพล (all pole) ในรูปของการแปลงแบบแซด (z-tranform)ได้ดังนี้

$$S(z) = E(z)[1/A(z)] \quad \text{(รูปแบบของการสังเคราะห์เสียง)} \qquad \text{..... 2.4.1}$$

โดยมีการนิยามดังนี้

$$A(z) = \sum_{i=0}^{M} a_i z^{-1} \qquad (a_{0=1}) \qquad \text{..... 2.4.2}$$

ค่า M คืออันดับของการทำนายแบบเชิงเส้นหรือเทคนิคแอลพีซีนั่นเอง และ A(z) นี้เรียกว่าตัวกรองแบบออลซีโร (all zero filter) และจากสมการที่ 2.4.1 จะเรียกส่วนกลับของตัวกรองแบบออลซีโร (1/A(z)) นี้ว่าตัวกรองแบบออลโพล (all pole filter) ก็ได้ ความหมายของสมการที่ 2.4.1 นี้ก็คือถ้าทำการป้อนสัญญาณ E(z) เข้าไปสู่ตัวกรองแบบออลโพล ก็จะได้เอาท์พุทคือ S(z) ออกมาซึ่งก็คือการแปลงแบบแซดของรูปแบบจำลองของสัญญาณเสียงนั่นเอง

ถ้าเราคูณสมการ 2.4.1 ทั้งสองข้างด้วย A(z) ผลจะออกมาเป็น

$$E(z) = S(z)A(z) \qquad \text{(รูปแบบการวิเคราะห์เสียง)} \qquad \text{..... 2.4.3}$$

สมการนี้จะหมายถึงรูปแบบจำลองของการวิเคราะห์สัญญาณเสียง โดยถ้าเราให้ S(z) เป็นสัญญาณอินพุทเข้าไปยังส่วนกลับของตัวกรองแบบออลโพลหรือก็คือ A(z) (ซึ่งจะหาค่าสัมประสิทธิ์ของมันได้จากการวิเคราะห์รูปแบบคลื่นของสัญญาณเสียง) ก็จะได้เอาท์พุทออกมาเป็น E(z) หรือเป็นไดรฟวิงฟังก์ชัน (driving function) ของรูปแบบการสังเคราะห์เสียงนั่นเอง

2.4.2 <u>รูปแบบของการทำนายแบบเชิงเส้น</u> (Linear Prediction Model)

จากหลักการพื้นฐานของแอลพีซีดังที่ได้กล่าวมาในตอนต้นแล้วนั้น ถ้าสมมติให้สัญญาณเสียงตัวอย่างของเราที่ทำการคัดตัวอย่างที่เวลา n เป็น x(n) ต่อไปก็จะนำไปเปรียบเทียบกับสัญญาณเสียงตัวอย่างที่ทำการสุ่มมาก่อนหน้าคือ x(n-1) โดยคาดว่า x(n) ควรจะมีค่าใกล้เคียงกับ x(n-1) ที่สุด โดยการคูณสัญญาณตัวก่อนหน้านี้ด้วยค่าคงที่ a_1 ค่าผิดพลาดที่เกิดขึ้นจากการเปรียบเทียบสัญญาณจริงกับสัญญาณที่ได้มาจากการทำนายนี้จะเรียกว่า ค่าผิดพลาดของสัญญาณ (error signal) หรือค่าผิดพลาดจากการทำนาย (prediction error) มีสัญญลักษณ์เป็น e(n) ซึ่งสามารถเขียนออกมาเป็นสมการได้ดังนี้

$$e(n) = x(n) - a_1 x(n-1) \qquad \text{..... 2.4.4}$$

แต่เพื่อให้การทำนายนี้มีความแม่นยำยิ่งขึ้น ก็จะต้องนำสัญญาณตัวอย่างนี้ไปทำการเปรียบเทียบกับสัญญาณเสียงตัวอย่างก่อนๆ หลายๆ ค่า ในที่นี้สมมติให้ทำการเปรียบเทียบกับค่าสัญญาณตัวอย่างก่อนหน้าเป็นจำนวน M ค่าก็จะได้สมการเป็น

$$\begin{aligned} e(n) &= x(n) - a_1 x(n-1) - a_2 x(n-2) - \ldots - a_M x(n-M) \\ &= x(n) - \sum_{k=1}^{M} a_k x(n-k) \end{aligned} \qquad \text{..... 2.4.5}$$

จะเห็นได้ว่าค่าตัวคูณคงที่ a_k จะเป็นตัวปรับให้เกิดค่าผิดพลาดจากการทำนายลดน้อยลงได้ จะเรียกค่า a_k นี้ว่า สัมประสิทธิ์ของการทำนาย (prediction coefficients)

ถ้ากลับมาพิจารณาสมการที่ 2.4.3 โดยการแทนค่า A(z) จากสมการ 2.4.2 จะได้ว่า

$$E(z) = \sum_{i=0}^{M} a_i S(z) z^{-1} \qquad \text{..... 2.4.6}$$

หรือเขียนในรูปฟังก์ชันของเวลาจะได้เป็น

$$e(n) = -\sum_{i=0}^{M} a_i s(n-i) \qquad a_0 = 1 \qquad \text{..... 2.4.7}$$

$$= s(n) - \sum_{i=1}^{M} a_i s(n-i) \qquad \text{..... 2.4.8}$$

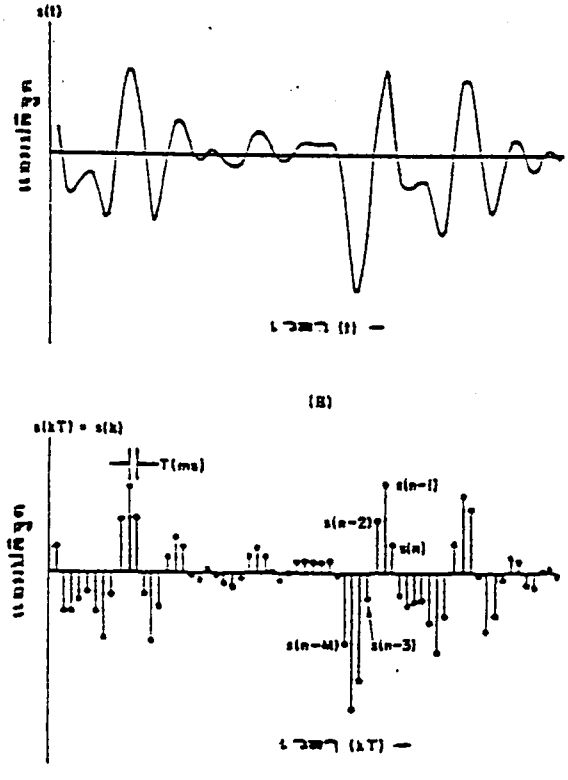
จะเห็นได้ว่าการทำนายที่มีอันดับเท่ากับ M ก็จะต้องใช้การรวมตัวอย่างแบบเชิงเส้นจากตัวอย่าง M ตัวอย่างก่อนหน้านั้น ถ้าเราให้ $\hat{s}(n)$ เป็นตัวอย่างที่เราทำนายขึ้นมาจะได้

$$e(n) = s(n) - \hat{s}(n) \qquad \text{..... 2.4.9}$$

โดยที่

$$\hat{s}(n) = -\sum_{i=1}^{M} a_i s(n-i) \qquad \text{..... 2.4.10}$$

นั่นคือไดรฟ์วิงฟังก์ชัน e(n) สามารถที่จะมองได้เป็นค่าผิดพลาดจากการทำนายระหว่างสัญญาณจริงกับสัญญาณที่เราทำนายขึ้นมา ค่า $-a_i$, i=1,2,...,M ก็คือค่าสัมประสิทธิ์ของการทำนายที่จะต้องทำการหา ส่วนเครื่องหมายลบทำให้ค่า e(n) เป็นค่าของความแตกต่างของสัญญาณทั้งสองตัว



รูปที่ 2.4.2 การคัดตัวอย่างของสัญญาณเสียงทุก ๆ ช่วงเวลา T

จากรูปแสดงถึงการคัดตัวอย่างของสัญญาณเสียงพูด s(t) ทุก ๆ ช่วงเวลา T มิลลิวินาทีทุก ๆ ตัวอย่างที่ n s(n) ก็จะได้มาจากการทำนายโดยการใช้การรวมกันแบบเชิงเส้นของสัญญาณก่อนหน้านั้น M ตัวอย่าง คือ s(n-1),s(n-2),...,s(n-M) สมการที่ 2.4.10 เขียนเป็นการแปลงแบบแซดได้เป็น

$$\hat{s}(z) = F(z)S(z) \qquad \text{..... 2.4.11}$$

โดยที่

$$F(z) = -\sum_{i=1}^{M} a_i z^{-1} \qquad \text{..... 2.4.12}$$

F(z) คือฟังก์ชันตัวกรองของการทำนายแบบเชิงเส้น ส่วน $\hat{s}(z) \leftrightarrow \hat{s}(n)$ และ $S(z) \leftrightarrow s(n)$ นั้นคือการนิยามตามความสัมพันธ์ระหว่างข้อมูลตัวอย่างกับคู่ของการแปลงแบบแซด จากสมการที่ 2.4.9 , 2.4.10 รูปแบบของการทำนายเชิงเส้นในโดเมนของการแปลงแบบแซดจะมีลักษณะดังนี้

$$E(z) = S(z)[1 - F(z)]$$

หรือ

$$= S(z)A(z) \qquad \text{..... 2.4.13}$$

โดยที่

$$A(z) = 1 - F(z)$$

$$= 1 + \sum_{i=1}^{M} a_i Z^{-1} \qquad \text{..... 2.4.14}$$

สำหรับรูปแบบในด้านการสังเคราะห์สัญญาณเสียงก็จะได้เป็น

$$S(z) = E(z)/[1 - F(z)]$$

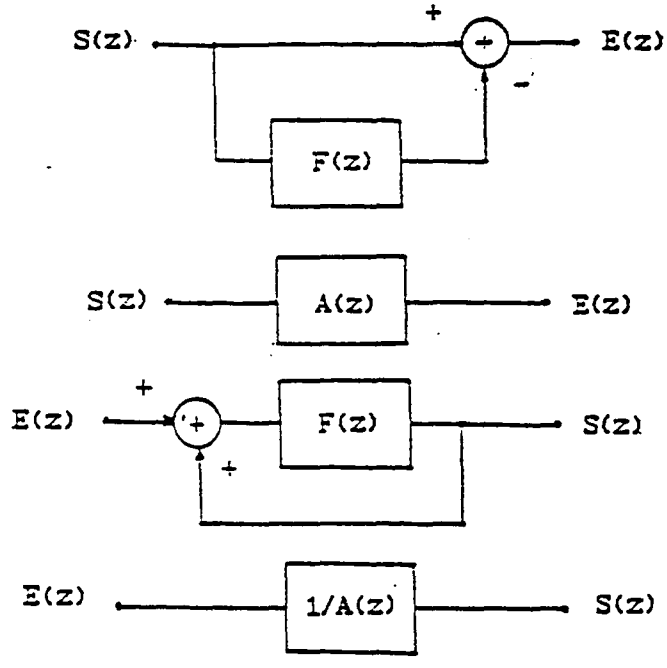
หรือ

$$= E(z)/A(z) \qquad \text{..... 2.4.15}$$

เขียนในโดเมนของเวลาได้เป็น

$$s(n) = e(n) - \sum_{i=1}^{M} a_i s(n-1) \qquad \text{..... 2.4.16}$$

ซึ่งทั้งสองลักษณะแสดงเป็นบล็อกไดอะแกรมได้ดังนี้



รูปที่ 2.4.3 บล็อกไดอะแกรมของแบบจำลองของการวิเคราะห์ (บน) และการสังเคราะห์ (ล่าง) ของการทำนายแบบเชิงเส้น

สมการในเทอมของตัวกรองการทำนายแบบเชิงเส้นเสนอขึ้นมาโดย Atal เมื่อปี ค.ศ. 1970 และในเทอมของส่วนกลับของตัวกรองการทำนายแบบเชิงเส้นโดย Markel เมื่อปี ค.ศ. 1971 โดยสรุปใจความสำคัญของการทำนายแบบเชิงเส้นว่า พารามิเตอร์ของฟังก์ชัน A(z) สามารถหาได้มาจากรูปคลื่นของสัญญาณเสียงโดยตรงจากการประยุกต์ใช้หลักการของค่าผิดพลาดกำลังสองที่น้อยที่สุด (least mean square error) จากสมการที่ 2.4.9 เนื่องจาก e(n) คือค่าผิดพลาดระหว่างสัญญาณจริงกับสัญญาณที่ทำขึ้นมา ดังนั้นจากหลักการดังกล่าวเราก็จะต้องหาสัมประสิทธิ์ a_i , i=1,2,...,M ที่จะทำให้ค่า e(n) มีค่าน้อยที่สุด แต่เนื่องจากเราจะทำการแบ่งสัญญาณเสียงมาพิจารณาเป็นเฟรม ๆ ไป ดังนั้นจึงกำหนดค่าผิดพลาดยกกำลังสองรวมขึ้นมาให้เป็น α ซึ่งมีนิยามดังนี้

$$\alpha = \sum_{n=n_0}^{n_1} e^2(n) \qquad \text{..... 2.4.17ก}$$

$$\alpha = \sum_{n=n_0}^{n_1} \left[\sum_{i=0}^{M} a_i s(n-i) \right]^2 \qquad \text{..... 2.4.17ข}$$

$$\alpha = \sum_{n=n_0}^{n_1} \sum_{i=0}^{M} \sum_{j=o}^{M} a_i s(n-i) s(n-j) a_j \qquad \text{..... 2.4.17ค}$$

ค่า n_0 กับ n_1 เป็นตัวกำหนดขอบเขตที่จะทำการพิจารณาซึ่งจะเกิดค่าผิดพลาดน้อยที่สุด ถ้าเรานิยามตัวแปรขึ้นมาตัวหนึ่งให้มีค่าดังนี้

$$c_{ij=} \sum_{n=n_0}^{n_1} s(n-i) s(n-j) \qquad \text{..... 2.4.18}$$

ก็จะทำให้เขียนสมการที่ 2.4.17 ได้ใหม่ว่า

$$\alpha = \sum_{i=0}^{M} \sum_{j=0}^{M} a_i c_{ij} a_j \qquad \text{..... 2.4.19}$$

ทำให้ค่าผิดพลาดยกกำลังสองมีค่าน้อยที่สุด โดยการดิฟเฟอร์เรนเชียลสมการ 2.4.19 ทั้งสองข้างเทียบกับ a_k ใด ๆ ที่ k=1,2,...,M แล้วจัดให้เท่ากับศูนย์ก็จะได้

$$\frac{\partial \alpha}{\partial a_k} = 0 = 2 \sum_{i=0}^{M} a_i c_{ik} \qquad \text{..... 2.4.20}$$

แต่ $a_0 = 1$ เพราะฉะนั้น

$$\sum_{i=1}^{M} a_i c_{ik} = -c_{0k} \qquad k = 1,2,...,M \qquad \text{..... 2.4.21}$$

สมการที่ 2.4.21 นี้เป็นสมการเชิงเส้นที่มีตัวไม่รู้ค่าอยู่ M ตัว $\{a_i\}$ ซึ่งจะต้องหาออกมาจากสมการเชิงเส้น M สมการ ค่า c_{ik} i=0,1,...,M นั้นเรารู้ค่าได้จากสมการที่ 2.4.18 อยู่แล้ว ถ้าเราหาค่า a_i i=0,1,...,M ออกมาได้จากสมการที่ 2.4.21 นี้ก็สามารถนำไปหาค่าผิดพลาด e(n) ในการทำการวิเคราะห์เสียงได้จากสมการที่ 2.4.8 และอาจนำไปสังเคราะห์เป็นสัญญาณเสียงกลับคืนมาได้ด้วย สมการที่ 2.4.16 สำหรับวิธีการแก้สมการที่ 2.4.21 เพื่อหาค่า a_i ออกมานั้นมีสามวิธีที่น่าสนใจและสำคัญคือ

*1) **วิธีโควาเรียนซ์*** (Covariance Method)

*2) **วิธีออโตคอรีเลชัน*** (Autocorrelation Method)

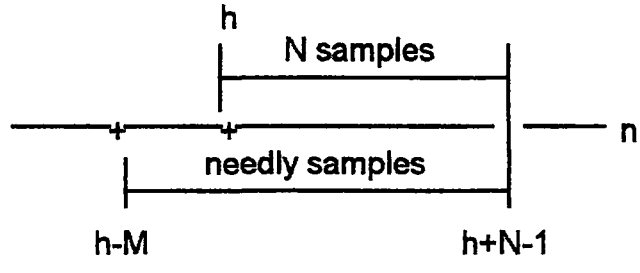
*3) **วิธีพาร์คอร์*** (PARCOR Method)

2.4.3 วิธีการหาค่าสัมประสิทธิ์ของการทำนาย

2.4.3.1 วิธีโควาเรียนซ์

เป็นแนวทางที่ใช้การนิยามสัญญาณเสียง s(n) ออกเป็นส่วน ๆ และมีการจำกัดขอบเขตในการรวม เพื่อให้พิจารณาอยู่เฉพาะภายในช่วงที่มีการหาผลรวมของค่าผิดพลาดยกกำลังสองเท่านั้น เช่น สมมติว่าถ้าเรามีตัวอย่างของสัญญาณเสียงอยู่ N ตัวอย่างคือ s(n) = s(0),s(1),s(2),...,s(N) ถ้าเริ่มต้นคำนวณที่จุด n=h ใด ๆ เพื่อทำการหาค่าสัมประสิทธิ์ของการทำนายระหว่างตัวอย่างที่ h ถึงตัวอย่างที่ h+n ดังนั้นการหาค่าผิดพลาดยกกำลังสองที่น้อยที่สุดก็จะอยู่ภายในกรอบ [h,h+N-1] นั่นคือ

$$\alpha = \sum_{n=h}^{h+N-1} e^2(n) \qquad \text{..... 2.4.22}$$



รูปที่ 2.4.4 ขอบเขตของตัวอย่างที่ใช้ในวิธีการโควาเรียนซ์

และนำเอาตัวอย่างมาใช้ในการคำนวณหาสัมประสิทธิ์ของโควาเรียนซ์เมทริกซ์ c_{ij} ดังนี้

$$\sum_{i=1}^{M} a_i c_{ij} = -c_{0j} \qquad j = 1,2,...,M \qquad \text{..... 2.4.23}$$

โดยที่ $$c_{ij} = \sum_{n=h}^{h+n-1} s(n-i)s(n-j) \qquad 0 \le i \le M, 1 \le j \le M \qquad \text{..... 2.4.24}$$

จากสมการที่ 2.4.23 เขียนในรูปเมทริกซ์ได้เป็น

$$\begin{bmatrix} c(1,1) & c(1,2) & c(1,3) & ..... & c(1,M) \\ c(2,1) & c(2,2) & c(2,3) & ..... & c(2,M) \\ ..... & ..... & ..... & ..... & ..... \\ ..... & ..... & ..... & ..... & ..... \\ c(M,1) & c(M,2) & c(M,3) & ..... & c(M,M) \end{bmatrix} \begin{bmatrix} a(1) \\ a(2) \\ ..... \\ .... \\ a(M) \end{bmatrix} = \begin{bmatrix} c(0,1) \\ c(0,2) \\ ..... \\ ..... \\ c(0,M) \end{bmatrix}$$

..... 2.4.25

จากสมการที่ 2.4.24 เราจะได้ว่า $c_{ij} = c_{ji}$ ดังนั้นเมทริกซ์ในสมการที่ 2.4.25 ก็จะเป็นเมทริกซ์ที่สมมาตรตามแนวเส้นทะแยงมุม เพราะฉะนั้นในการหาผลลัพธ์ของสมการหรือเมทริกซ์นี้ก็จะทำได้อย่างมีประสิทธิภาพสูงขึ้น เช่นการใช้ทฤษฎีของ Cholesky ในการหาผลลัพธ์ออกมา วิธีโควาเรียนซ์นี้ไม่ต้องมีการใช้ฟังก์ชันของช่องวิเคราะห์เข้ามากำหนดรูปร่างของสัญญาณเสียง และมันจะสามารถให้ค่าสัมประสิทธิ์ของการทำนายที่ค่อนข้างจะแม่นยำ ในขณะที่ใช้เฟรมวิเคราะห์เล็กกว่าวิธีออโตคอรีเลชันด้วย ซึ่งตามปกติก็จะใช้ขนาดของเฟรมประมาณ 50-100 ตัวอย่างสัญญาณเสียง และทำการคำนวณครั้งต่อไปทุก ๆ 100 ถึง 200 ตัวอย่างสัญญาณ

2.4.3.2 วิธีออโตคอรีเลชัน

จากสมการ 2.4.17 ถ้าเรากำหนดให้ $n_0 = -\infty$ และ $n = \infty$ และนิยามให้ s(n) มีค่าเท่ากับศูนย์ในกรณีที่ $n<0$ และ $n \ge N$ นั่นคือการกำหนดรูปแบบช่องแคบแบบสี่เหลี่ยมขนาด (0,N] จะทำให้สมการที่ 2.4.18 กลายเป็น

$$c_{i,j} = \sum_{n=-\infty}^{\infty} s(n-i)s(n-j)$$

$$c_{i,j} = \sum_{n=0}^{N-1-|i-j|} s(n)s(n+|i-j|)$$

$$c_{i,j} = r(|i-j|) \qquad \text{..... 2.4.26}$$

ในกรณีนี้แม้ว่าค่าผิดพลาด e(n) จะเป็นการหาค่าต่ำสุดภายในช่วงที่ไม่มีการจำกัดขอบเขตก็ตาม แต่ก็จะได้เช่นเดียวกับการหาค่าผิดพลาดต่ำสุดภายในช่วง [0,N+M-1] เหมือนกัน เนื่องจากว่าค่า s(n) เป็นศูนย์สำหรับ $n \geq N$ และ $n < 0$ นั่นเอง

$$\sum_{i=1}^{M} a_i r(|i-j|) = -r(j) \qquad j = 1,2,...,M \qquad \text{..... 2.4.27}$$

โดยที่

$$r(l) = \sum_{n=0}^{N-1-l} s(n)s(n+l) \qquad (l \geq 0) \qquad \text{..... 2.4.28}$$

และ

$$e(n) = \sum_{i=0}^{M} a_i s(n-i) \; a_0 = 1 \text{ และ } n = 0,1,...,N+M-1 \qquad \text{.....2.4.29}$$

จากสมการที่ 2.4.27 เขียนเมทริกซ์ได้เป็น

$$\begin{bmatrix} R(0) & R(1) & R(2) & ..... & R(M-1) \\ R(1) & R(0) & R(1) & ..... & R(M-2) \\ ..... & ..... & ..... & ..... & ..... \\ ..... & ..... & ..... & ..... & ..... \\ R(M-1) & R(M-2) & R(M-3) & ..... & R(0) \end{bmatrix} \begin{bmatrix} a(1) \\ a(2) \\ .. \\ .. \\ a(M) \end{bmatrix} = \begin{bmatrix} R(1) \\ R(2) \\ .. \\ .. \\ R(M) \end{bmatrix}$$

..... 2.4.30

จะได้เมทริกซ์ขนาด M*M สามารถแก้สมการหาผลลัพธ์ออกมาได้หลายวิธีด้วยกัน แต่วิธีอันหนึ่งที่นิยมใช้กันก็คือวิธีของ Durbin และ Levinson ซึ่งจะใช้ขั้นตอนหรือจำนวนครั้งในการคำนวณน้อยครั้งกว่าที่จะต้องใช้ในการแก้สมการตามวิธีพื้นฐานปกติ สำหรับวิธีออโตคอรีเลชันนี้จำเป็นที่จะต้องมีการใช้ฟังก์ชันของช่องแคบมากำหนดรูปแบบของสัญญาณ โดยอาศัยสัญญาณที่อยู่นอกช่องแคบจะมีค่าเป็นศูนย์หมด ในการคำนวณนั้นตามปกติจะใช้ช่องแคบที่มีขนาดประมาณ 100-250 ตัวอย่างสัญญาณเสียงเพื่อใช้ในการวิเคราะห์ในหนึ่งเฟรม

2.4.4.3 การเปรียบเทียบระหว่างวิธีโควาเรียนซ์กับออโตคอรีเลชัน

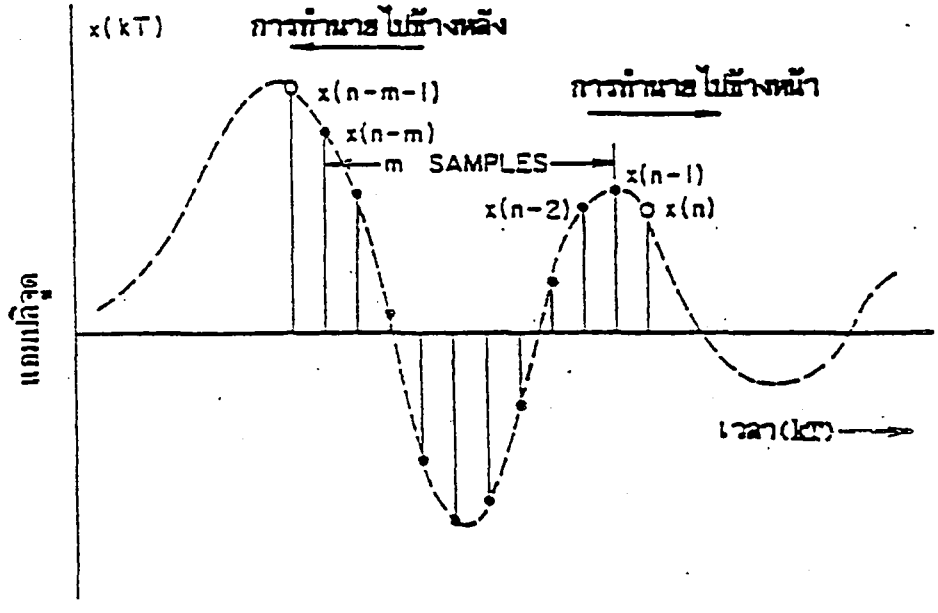
วิธีการของออโตคอรีเลชัน จำเป็นจะต้องมีการใช้การกำหนดขนาดของช่องแคบ หรือนำเอาวินโดว์ฟังก์ชันเข้ามาคูณกับสัญญาณเสียงด้วย ซึ่งหมายถึงว่าในทางปฏิบัติเราจะต้องพิจารณาจากตัวอย่างสัญญาณเป็นช่วงเวลาที่ยาวนานกว่าวิธีโควาเรียนซ์ โดยจะเห็นได้จากวิธีออโตคอรีเลชันจะกำหนดจำนวนตัวอย่างของสัญญาณภายในช่องแคบที่จะนำมาพิจารณามีค่า N ประมาณ 256 ตัวอย่าง และสำหรับวิธีโควาเรียนซ์จะกำหนดค่า N ประมาณ 100 ตัวอย่าง สำหรับในด้านของการคำนวณนั้นได้มีการเปรียบเทียบทั้งสองเทคนิคนี้โดย Witten ในปี ค.ศ.1980 ได้ว่าวิธีการหาผลลัพธ์จากการอินเวอร์สเมทริกซ์ของ Durbin และ Levinson สำหรับวิธีออโตคอรีเลชันจะมีประสิทธิภาพดีกว่าการหาผลลัพธ์โดยการแยกองค์ประกอบ (decomposition) ของ Cholesky สำหรับวิธีโควาเรียนซ์ที่จำนวนครั้งเท่า ๆ กัน

องค์ประกอบซึ่งใช้ในการให้น้ำหนักสำหรับวิธีโควาเรียนซ์นั้นยากในการที่จะแบ่งปริมาณเป็นระดับต่างๆ ในทันทีทันใดได้ แต่สำหรับวิธีออโตคอรีเลชันนั้นเราสามารถใช้การคำนวณทางคณิตศาสตร์เป็นแบบกำหนดความละเอียด (fix point) ได้เลย ทำให้สามารถนำไปดัดแปลงเป็นวงจรการคำนวณทางด้านฮาร์ดแวร์ได้โดยง่าย นอกจากนั้นสิ่งที่สำคัญสิ่งหนึ่งก็คือ วิธีโควาเรียนซ์ในบางครั้งอาจจะทำให้ได้ผลลัพธ์ออกมาที่ไม่เสถียรได้ ในขณะที่วิธีออโตคอรีเลชันนั้นสามารถจะแสดงให้เห็นได้ว่าผลลัพธ์ที่ออกมาจะเสถียรแน่นอน แต่กระนั้นก็ตามผลจากการประมาณคุณสมบัติของสัญญาณโดยการใช้ช่องแคบ และผลกระทบของการกระทำกับตัวอย่างของสัญญาณจำนวนมากของวิธีออโตคอรีเลชัน จะทำให้ค่าสัมประสิทธิ์ที่ใช้แทนสัญญาณเสียงที่ได้จากวิธีโควาเรียนซ์จะมีความแม่นยำค่อนข้างสูงกว่า ซึ่งเป็นข้อดีที่สำคัญของวิธีโควาเรียนซ์

2.4.3.4 พาร์คอร์ (PARCOR : Partial Correlation)

เป็นการประยุกต์เทคนิคการทำนายแบบเชิงเส้นกับสัญญาณเสียงแนวทางหนึ่งซึ่งเสนอขึ้นมาโดย Itakura และ Saito (1969-1972) โดยใช้ค่าสัมประสิทธิ์พาร์เชียลคอรีเลชัน (partial correlation coefficients) และแนวทางนี้ก็ได้นำมาใช้ในการกำหนดนิยามรายละเอียดภายในของส่วนกลับตัวกรอง A(z)

วิธีพาร์คอร์นี้จะทำการทำนายแบบเชิงเส้นที่มีอันดับเท่ากับ m ทั้งการทำนายไปข้างหน้า และทำนายย้อนกลับหลังดังรูปข้างล่าง โดยกำหนดให้ค่าผิดพลาดในการทำนายไปข้างหน้า m อันดับที่เวลา nใด ๆ มีค่าเป็น $x^{+}_{m}(n)$ โดยนิยามค่าทั้งสองได้ดังนี้



รูปที่ 2.4.5 หลักการของวิธีพาร์คอร์

$$x^{+}_{m}(n) = x(n) - \left[-\sum_{i=1}^{m} a_{mi} x(n-i) \right]$$

$$= \sum_{i=0}^{m} a_{mi} x(n-i) \qquad a_{m0} = 1 \qquad \text{..... 2.4.31}$$

x^{+}_{m} (n) ได้มาจากผลต่างของสัญญาณที่เวลา n กับสัญญาณที่ได้จากการทำนายที่เกิดจากสมการการรวมแบบเชิงเส้นของสัญญาณที่อยู่ก่อนหน้าไป m หน่วยของเวลา ส่วน x^{-}_{m} (n) ก็ได้มาจากผลต่างของสัญญาณที่เวลา n กับสัญญาณที่ได้จากการทำนายที่เกิดจากการรวมแบบเชิงเส้นของสัญญาณที่อยู่ถัดไปตามแกนเวลาโดยเริ่มคิดที่เวลา n-m-1

$$x^{-}_{m}(n) = x(n-m-1) - \left[-\sum_{i=1}^{m} b_{mi}x(n-i)\right]$$
$$= \sum_{i=1}^{m+1} b_{mi}x(n-i) \qquad b_{m,m+1} = 1 \qquad \text{..... 2.4.32}$$

ตามเทคนิคการทำนายแบบเชิงเส้นจะต้องหาผลรวมของค่าผิดพลาดกำลังสองของทั้งสองด้านได้เป็น

$$\alpha_m = \sum_{n=n0}^{n1} \left[x^{+}_{m}(n)\right]^2 \qquad m = 1,2,\ldots,M \qquad \text{.....}$$

2.4.33

$$\beta_m = \sum_{n=n0}^{n1} \left[x^{-}_{m}(n)\right]^2 \qquad m = 1,2,\ldots,M \qquad \text{..... 2.4.34}$$

และหาค่าต่ำสุดของผลรวมของค่าผิดพลาดกำลังสอง โดยการทำพาร์เชียลดิฟเฟอร์เรนเชียลกับพารามิเตอร์ที่สนใจแล้วจับให้เท่ากับศูนย์ได้

$$\frac{\partial \alpha_m}{\partial a_{mi}} = 0 \qquad \text{และ} \qquad \frac{\partial \beta_m}{\partial b_{mi}} = 0 \qquad i = 1,2,\ldots,m \qquad \text{..... 2.4.35}$$

แต่จากเงื่อนไขนี้ ยังไม่สามารถนำไปใช้แก้สมการเพื่อหาค่าสัมประสิทธิ์ของการทำนายได้โดยตรง Markel และ Gray ได้เสนอแนวทางในการหาคำตอบออกมาเป็นค่าสัมประสิทธิ์ของพาร์คอร์แทนด้วยสัญญลักษณ์ k_m โดยการใช้หลักของการอินเนอร์โปรดักท์และคุณสมบัติของออโธโกนอลดังที่จะกล่าวต่อไปนี้

2.4.3.5 หลักการของอินเนอร์โปรดักท์และคุณสมบัติออโธโกนอล

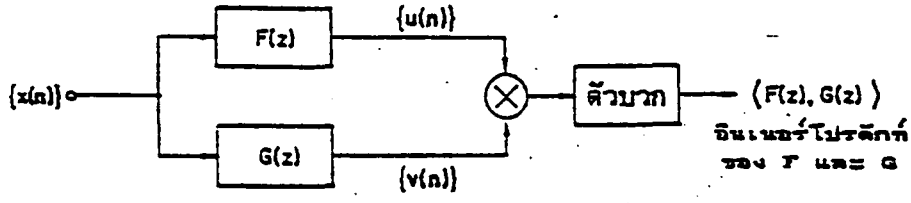
ค่าผิดพลาดจากการทำนายของสัญญาณทั้งสองด้านคือ $x^{+}_{m}(n)$ และ $x^{-}_{m}(n)$ สามารถจะพิจารณาให้เป็นเอาท์พุทที่ออกมาจากตัวกรอง $A_m(z)$ และ $B_m(z)$ ตามลำดับ โดยมีสัญญาณอินพุทเป็น x(n) โดยที่

$$A_m(z) = \sum_{i=0}^{m} a_{mi} z^{-1} \qquad \text{โดย} \quad a_{m0} = 1 \qquad \text{..... 2.4.36ก}$$

$$B_m(z) = \sum_{i=1}^{m+1} b_{mi} z^{-1} \qquad \text{โดย} \quad b_{m,m+1} = 1 \qquad \text{..... 2.4.36ข}$$

ผลรวมของค่าผิดพลาด α_m และ β_m แทนถึงพลังงานที่ออกจากตัวกรองภายในช่วงเวลาจาก $n=n_0$ ถึง $n=n_1$

หลักการของอินเนอร์โปรดักท์ เริ่มจากการสมมติให้สัญญาณเสียง x(n) ป้อนเข้าผ่านตัวกรองสองตัวคือ F(z) กับ G(z) ซึ่งมีค่าสัมประสิทธิ์เป็นจำนวนจริงดังรูป



รูปที่ 2.4.6 แสดงถึงอินเนอร์โปรดักท์ของตัวกรอง F(z) กับ G(z)

เอาท์พุทของตัวกรองแต่ละตัวคือ u(n) กับ v(n) ตามลำดับ จะถูกนำมาคูณและบวกกันตั้งแต่ตัวอย่างที่ $n=n_0$ ถึง $n=n_1$ ผลที่ได้ออกมาจะเรียกว่าอินเนอร์โปรดักท์ของ F(z) กับ G(z) โดยมีสัญลักษณ์เป็น $\langle F(z), G(z) \rangle$ และนิยามได้ดังนี้

$$\langle F(z), G(z) \rangle = \sum_{n=n0}^{n1} u(n) v(n) \qquad \text{..... 2.4.37}$$

ถ้าตัวกรอง F(z) และ G(z) อยู่ในรูปของ

$$F(z)=\sum_{i=0}^{\infty} f_i z^{-1} \qquad \text{และ} \qquad G(z)=\sum_{i=0}^{\infty} g_i z^{-1} \qquad \text{..... 2.4.38}$$

อินเนอร์โปรดักท์ของ F(z) กับ G(z) จะเท่ากับ

$$\langle F(z),G(z)\rangle=\sum_{i=0}^{\infty}\sum_{j=0}^{\infty} f_i\left[\sum_{n=n0}^{n1} x(n-i)x(n-j)\right]g_j \qquad \text{..... 2.4.39}$$

โดยที่

$$\left\langle z^{-i},z^{-j}\right\rangle=\sum_{n=n0}^{n1} x(n-i)x(n-j) \qquad \text{..... 2.4.40}$$

โดยวิธีโควาเรียนซ์ ซึ่งกำหนดให้ n_0 = M และ n_1 = N-1 จะได้

$$\left\langle z^{-i},z^{-j}\right\rangle=c_{ij}=c_{ji} \qquad \text{..... 2.4.41}$$

ส่วนวิธีออโตคอรีเลชันซึ่งมีขอบเขต $n_0 = -\infty$, $n_1 = \infty$ และค่าของข้อมูลจะไม่เป็นศูนย์เมื่ออยู่ภายในช่วง [0,N-1] เท่านั้นจะได้ว่า

$$\left\langle z^{-i},z^{-j}\right\rangle=r(i-j)=r(j-i) \qquad \text{..... 2.4.42}$$

จากการนิยามอินเนอร์โปรดักท์ข้างต้นค่าผิดพลาดจากการทำนายทั้งสองด้านคือ α_m กับ β_m ถ้าให้มันเป็นเอาท์พุทของตัวกรอง $A_m(z)$ กับ $B_m(z)$ ตามลำดับจะสามารถเขียนให้อยู่ในรูปแบบตามนิยามของอินเนอร์โปรดักท์ได้เป็น

$$\alpha_m=\langle A_m(z),A_m(z)\rangle \qquad \text{..... 2.4.43ก}$$

$$\beta_m=\langle B_m(z),B_m(z)\rangle \qquad \text{..... 2.4.43ข}$$

ซึ่งในกรณีนี้คือ อินเนอร์โปรดักท์ของตัวกรองตัวมันเอง จะเรียกใหม่ว่า นอร์มสแควร์ (norm square) และมีนิยามดังนี้

$$\|F(z)\|^2 = \langle F(z), F(z) \rangle \qquad \text{..... 2.4.44}$$

จุดวิกฤตของการหาค่าต่ำสุดของค่าผิดพลาดจากการทำนายสามารถอธิบายได้ด้วยการหาค่าต่ำสุดของนอร์มสแควร์ของตัวกรอง $A_m(z)$ และ $B_m(z)$ ซึ่งเป็นรูปแบบของสมการโพลิโนเมียลจาก m=1,2,...,M ต่อไปจะนำหลักการของอินเนอร์โปรดักท์นี้มาประยุกต์ใช้เพื่อแสดงถึงคุณสมบัติออโธโก-นอลดังนี้

ถ้าตัวกรอง $A_m(z)$ และ $B_m(z)$ เป็นตัวทำให้ได้ α_m และ β_m ที่น้อยที่สุดจริง ๆ แล้ว ถ้าเราบวกค่า cz^{-j} โดยที่ j=1,2,...,m และ c เป็นค่าคงที่เข้าไปในโพลิโนเมียลแล้ว นอร์มสแควร์หลังการบวกจะต้องใหญ่กว่านอร์มสแควร์ที่ยังไม่ได้บวก เพราะฉะนั้นจะได้

$$\|A_m(z) + cz^{-j}\|^2 \geq \|A_m(z)\|^2 \qquad \text{โดย } j = 1,2,...,m \qquad \text{..... 2.4.45}$$

ถ้าทำการกระจายอสมการ 2.4.45 ออกมาในรูปของอินเนอร์โปรดักท์จะได้

$$2c\langle A_m(z), z^{-j} \rangle + c^2 \langle z^{-j}, z^{-j} \rangle \geq 0 \qquad \text{..... 2.4.46}$$

ถ้า $\langle z^{-j}, z^{-j} \rangle$ ไม่ได้มีค่าเป็นศูนย์แล้ว ก็สามารถจะเลือกค่า c ให้มีค่าเป็น

$$c = -\langle A_m(z), z^{-j} \rangle / \langle z^{-j}, z^{-j} \rangle \qquad \text{..... 2.4.47}$$

จะทำให้อสมการที่ 2.4.47 กลายเป็น

$$-\left[\langle A_m(z), z^{-j} \rangle\right]^2 \geq 0 \qquad \text{..... 2.4.48}$$

แต่ถ้าเกิด $<z^{-j},z^{-j}>$ มีค่าเป็นศูนย์แล้ว ค่า c ก็สามารถที่จะเลือกให้มีค่าเป็น $c = - <A_m(z), z^{-j}>$ เพื่อให้ได้ผลลัพธ์ออกมาเท่ากันกับอสมการที่ 2.4.48 โดยที่ค่า j=1,2,...,m จากอสมการที่ 2.4.48 นั้นเนื่องจากตัวกรอง $A_m(z)$ เป็นตัวกรองจำนวนจริง ดังนั้นอสมการที่ 2.4.48 จะเป็นจริงได้ก็ต่อเมื่อ

$$\left\langle A_m(z), z^{-j} \right\rangle = 0 \qquad \text{..... 2.4.49}$$

และสำหรับตัวกรอง $B_m(z)$ ก็จะพิสูจน์ได้ผลในทำนองเดียวกันคือ

$$\left\langle B_m(z), z^{-j} \right\rangle = 0 \qquad \text{โดย } j = 1,2,...,m \qquad \text{..... 2.4.50}$$

ความสัมพันธ์ของทั้งสองสมการคือ 2.4.49 กับ 2.4.50 นี้จะแสดงให้เห็นถึงคุณสมบัติออโธโก-นอล ซึ่งจะนำไปประยุกต์ใช้กับการหาค่าน้อยที่สุดของค่าผิดพลาดจากการทำนายทั้งสองด้านคือ α_m และ β_m ต่อไป

ความสัมพันธ์กันแบบออโธโกนอลนี้สามารถแสดงในเทอมของสัญญาณผิดพลาดกับอินพุทที่มีการหน่วงเวลา ได้จากนิยามของอินเนอร์โปรดักท์และสัญญาณผิดพลาดในสมการที่ 2.4.23 กับสมการที่ 2.4.24 ได้ดังนี้

$$\left\langle A_m(z), z^{-j} \right\rangle = \sum_{n=n0}^{n1} x^{+}{}_m(n)x(n-j) = 0 \qquad \text{..... 2.4.51ก}$$

$$\left\langle B_m(z), z^{-j} \right\rangle = \sum_{n=n0}^{n1} x^{-}{}_m(n)x(n-j) = 0 \qquad \text{.....2.4.51ข}$$

2.4.3.6 การหาสัมประสิทธิ์พาร์คอร์และโครงสร้างแบบแลททิส

การหาผลลัพธ์โพลิโนเมียลของตัวกรอง $A_m(z)$ และ $B_m(z)$ โดยใช้คุณสมบัติความสัมพันธ์ออโธโกนอลในลักษณะของการเรียกใช้ตัวเองตามวิธีของ Levinson Recursion แสดงได้ดังต่อไปนี้

จากสมการที่ 2.4.49 , 2.4.50 จะได้ค่าเริ่มต้นของความสัมพันธ์เมื่อเริ่มประมวลผลตั้งแต่ m=1,2,...,M ดังนี้

$$A_0(z) = 1 \qquad \text{และ} \qquad B_0(z) = z^{-1} \qquad \text{..... 2.4.52}$$

เนื่องจากทั้ง $A_m(z)$ และ $B_m(z)$ เป็นฟังก์ชันโพลิโนเมียลของ z^{-j} เมื่อ j=0,1,...,m-1 ดังนั้นการรวมกันแบบเชิงเส้นในรูปของ $A_{m-1}(z) + k_m B_{m-1}(z)$ ก็จะเป็นฟังก์ชันโพลิโนเมียลที่ออโธโกนอลกับ z^{-j} โดยที่ j=1,2,...,m-1 ด้วย ถ้า K_m เป็นค่าที่เลือกมาเพื่อทำให้การรวมกันแบบเชิงเส้นออโธโกนอลกับ z^{-m} แล้ว $A_m(z)$ สามารถเขียนได้ในรูปดังนี้

$$A_m(z) = A_{m-1}(z) + k_m B_{m-1}(z) \qquad \text{..... 2.4.53}$$

และถ้าค่า k_m ทำให้ $A_m(z)$ ออโธโกนอลกับ z^{-m} แล้วจะทำให้

$$\left\langle A_m(z), z^{-m} \right\rangle = \left\langle A_{m-1}(z), z^{-m} \right\rangle + k_m \left\langle B_{m-1}(z), z^{-m} \right\rangle = 0 \qquad \text{..... 2.4.54}$$

และในทำนองเดียวกันทั้ง $A_{m-1}(z)$ และ $B_{m-1}(z)$ ก็ออโธโกนอลกับ z^{-m} เมื่อ j=1,2,...,m-1 ด้วย ดังนั้นเทอมในสมการที่ 2.4.54 สามารถเขียนกระจายได้เป็น

$$\left\langle A_{m-1}(z), z^{-m} \right\rangle = \left\langle A_{m-1}(z), B_{m-1}(z) \right\rangle = \left\langle 1, B_{m-1}(z) \right\rangle \qquad \text{..... 2.4.55}$$

$$\left\langle B_{m-1}(z), z^{-m} \right\rangle = \left\langle B_{m-1}(z), B_{m-1}(z) \right\rangle = \left\| B_{m-1}(z) \right\|^2 = \beta_{m-1} \qquad \text{..... 2.4.56}$$

จากสมการที่ 2.4.54 แทนค่าด้วยสมการที่ 2.4.56 กับ 2.4.51 ได้ว่า

$$k_m = -\frac{\langle A_{m-1}(z), z^{-m} \rangle}{\langle B_{m-1}(z), z^{-m} \rangle} \qquad \text{..... 2.4.57}$$

$$= -\frac{1}{\beta_{m-1}} \langle A_{m-1}(z), B_{m-1}(z) \rangle \qquad \text{..... 2.4.58}$$

$$= -\frac{1}{\beta_{m-1}} \sum_{n=n0}^{n1} x^{+}_{m-1}(n) x^{-}_{m-1}(n) \qquad \text{..... 2.4.59}$$

จากสมการ 2.4.59 นี้ก็จะหาค่าสัมประสิทธิ์ของพาร์คอร์ k_m ออกมาได้ นอกจากนี้เรายังอาจพิจารณาลงไปในรูปแบบของโครงสร้างแลททิสได้อีกด้วยจากสมการที่ 2.4.48 นำเอาสมการที่ 2.4.41 มาใช้จะได้เป็น

$$\langle A_m(z), z^{-j} \rangle = \sum_{i=0}^{m} a_{mi} \langle z^{-i}, z^{-j} \rangle$$

$$= \sum_{i=0}^{m} a_{mi} r(i-j) = 0 \quad \text{เมื่อ } j = 1,2,...,m \qquad \text{..... 2.4.60}$$

ถ้าทำการกลับอันดับของดัชนี โดยให้ l=m+1-j และ i=m+1-k สมการที่ 2.4.60 เขียนได้ใหม่เป็น

$$\sum_{k=1}^{m+1} a_{m,m+1-k} r(l-k) = 0 \qquad \text{เมื่อ } l = 1,2,...,m \qquad \text{..... 2.4.61}$$

$$b_{mk} = a_{m,m+1-k} \qquad \text{เมื่อ } k = 1,2,...,m+1 \qquad \text{..... 2.4.62}$$

การแปลงแบบแซดของสมการที่ 2.4.62 จะเป็น

$$B_m(z) = z^{-(m+1)} A_m(1/z) \qquad \text{..... 2.4.63}$$

ซึ่งจะทำให้ได้ตามหลักของออโธโกนอลดังนี้

$$\left\langle B_m(z), z^{-l} \right\rangle = 0 \qquad \text{เมื่อ } l = 1,2,\ldots,m \qquad \text{..... 2.4.64}$$

ดังนั้น $B_m(z)$ ก็คือโพลิโนเมียลที่มีค่าสัมประสิทธิ์เหมือนกับ $A_m(z)$ เพียงแต่ว่ามีอันดับที่กลับกัน เพราะฉะนั้นนำสมการที่ 2.4.53 มารวมกับสมการที่ 2.4.63 จะได้

$$B_m(z) = z^{-1}\left[k_m A_{m-1}(z) + B_{m-1}(z)\right] \qquad \text{..... 2.4.65}$$

ต่อจากนั้นถ้ามองโดยใช้การแปลงแบบแซดมาอธิบายลักษณะของตัวกรองโดยให้ X(z) แทนการแปลงแบบแซดของข้อมูลอินพุท x(n) จะได้

$$X^+_m(z) = A_m(z)X(z) \text{ และ } X^-_m(z) = B_m(z)X(z) \qquad \text{..... 2.4.66}$$

เพราะฉะนั้นจากสมการที่ 2.4.53 กับ 2.4.65 ก็จะได้

$$X^+_m(z) = X^+_{m-1}(z) + k_m X^-_{m-1}(z) \qquad \text{..... 2.4.67}$$

$$X^-_m(z) = z^{-1}\left[k_m X^+_{m-1}(z) + X^-_{m-1}(z)\right] \qquad \text{..... 2.4.68}$$

โดย $X^+_m(z)$ และ $X^-_m(z)$ แทนการแปลงแบบแซดของ $x^+_m(n)$ และ $x^-_m(n)$ ตามลำดับ ดังนั้นสมการที่ 2.4.67 และ 2.4.68 นี้เขียนใหม่ในรูปของโดเมนเวลาเป็น

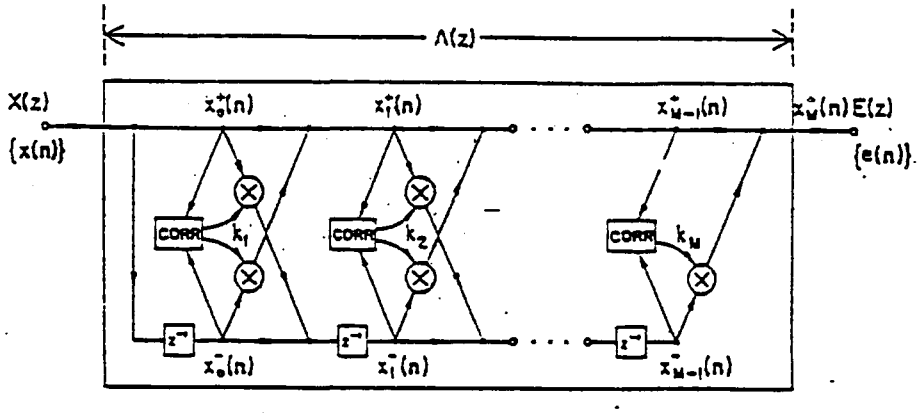
$$x^{+}_{m}(n) = x^{+}_{m-1}(n) + k_m x^{-}_{m-1}(n) \quad \text{..... 2.4.69}$$

$$x^{-}_{m}(n) = k_m x^{+}_{m-1}(n-1) + x^{-}_{m-1}(n-1) \quad \text{..... 2.4.70}$$

จากสมการที่ 2.4.23 และ 2.4.24 จะกำหนดขอบเขตของสองสมการข้างบนนี้ได้ดังนี้

$$x^{+}_{0}(n) = x(n) \text{ และ } x^{-}_{0}(n) = x(n-1) \quad \text{..... 2.4.71}$$

จากสมการที่ 2.4.69 , 2.4.70 และ 2.4.71 นี้สามารถแสดงให้เห็นในรูปของตัวกรองการวิเคราะห์ได้ดังรูป



รูปที่ 2.4.7 โครงสร้างของส่วนกลับของตัวกรอง {A(z)} ในรูปของโครงสร้างแลททิส

ในกรอบสี่เหลี่ยมที่เขียนว่า CORR คือขั้นตอนในการหาค่าสัมประสิทธิ์ k_m โดยการใช้สมการที่ 2.4.59

พารามิเตอร์ k_m ที่นิยามโดย Itakura และ Saito ในชื่อของค่าสัมประสิทธิ์พาร์คอร์นี้มีค่าเท่ากันทั้งด้านไปและด้านย้อนกลับ เพียงแต่ว่าเครื่องหมายต่างกันเท่านั้นเมื่อนำไปเปรียบเทียบกับวิธีออโตคอรีเลชัน สำหรับวิธีออโตคอรีเลชันนั้นจะได้ค่านอร์มสแควร์ของ $A_m(z)$ กับ $B_m(z)$ มีค่าเท่ากันหรือเป็นตัวเดียวกันนั่นเอง ดังนั้นสำหรับในวิธีออโตคอรีเลชันแล้วจะได้ว่า

$$\alpha_m = \beta_m = \sum_{n=-\infty}^{\infty} \left[x^+{}_m(n)\right]^2 = \sum_{n=-\infty}^{\infty} \left[x^-{}_m(n)\right]^2 \qquad \text{..... 2.4.72}$$

ทำให้ได้

$$k_m = -\frac{\sum_{n=-\infty}^{\infty} x^+{}_{m-1}(n)x^-{}_{m-1}(n)}{\sqrt{\sum_{n=-\infty}^{\infty}\left[x^+{}_{m-1}(n)\right]^2 \sum_{n=-\infty}^{\infty}\left[x^-{}_{m-1}(n)\right]^2}} \qquad \text{..... 2.4.73}$$

จากสมการนี้จะเห็นว่าค่าสัมประสิทธิ์ k_m จะมีค่าอยู่ระหว่าง ± 1 เสมอ นอกจากนี้ค่าสัมประสิทธิ์ของพาร์คอร์ยังมีส่วนเกี่ยวข้องคล้องจองกับค่าสัมประสิทธิ์การสะท้อนกลับ (reflection coefficients) ในแง่ของวิชา Acoustics ซึ่งกล่าวถึงการทำงานของช่องเสียงของมนุษย์ว่าทำการกรองสัญญาณเสียงได้อย่างไร พูดถึงรูปแบบของช่องทางเดินเสียง และแสดงให้เห็นว่าการแปรเปลี่ยนรูปร่างของช่องทางเดินเสียงนั้นสามารถจำลองให้เห็นเป็นลักษณะของท่อยาวที่มีเส้นผ่าศูนย์กลางแตกต่างกันออกไปขึ้นมา เมื่อรูปคลื่นผ่านจากท่ออันหนึ่งไปยังอีกอันหนึ่งนั้นมันจะมีการสร้างรูปคลื่นขึ้นมาใหม่ในทิศทางตรงกันข้ามซึ่งเรียกว่า รูปคลื่นสะท้อนกลับ (reflected waveforms) รูปคลื่นสะท้อนกลับนี้จะมีค่าของพารามิเตอร์ตัวหนึ่งที่เกี่ยวข้องอยู่ เรียกว่า สัมประสิทธิ์การสะท้อนกลับ (reflection coefficients) ซึ่งจะมีส่วนที่สัมพันธ์กันกับฟอร์แมนท์ (formant) หรือค่าสูงสุด (peak) ในสัญญาณเสียง ดังนั้นเราอาจจะนำค่าสัมประสิทธิ์การสะท้อนกลับนี้มาวิเคราะห์และใช้ในการเปลี่ยนแปลงแบบจำลองของช่องทางเดินเสียงได้ และจากการที่ค่าสัมประสิทธิ์พาร์คอร์สามารถเทียบเท่ากับค่าสัมประสิทธิ์การสะท้อนกลับดังกล่าวนี้เอง บางทีจึงมีการเรียกค่า k_m นี้ว่า ค่าสัมประสิทธิ์ของการสะท้อนกลับ

อย่างไรก็ตาม คุณลักษณะของตัวกรองนั้นเราอาจจะใช้พารามิเตอร์หรือค่าสัมประสิทธิ์ใดก็ได้ในระหว่างสัมประสิทธิ์การทำนายกับสัมประสิทธิ์การสะท้อนกลับมาใช้เป็นตัวกำหนดการแปรค่าของตัวกรอง แต่สำหรับค่าสัมประสิทธิ์การทำนายหรือ a_k (จากสมการที่ 2.4.21) นั้นอาจจะทำให้สถานะของตัวกรองเกิดไม่เสถียรขึ้นมาได้ เนื่องจากค่าสัมประสิทธิ์การทำนายมีค่าไม่แน่นอนไม่มีขอบเขตที่แน่ชัด ด้วยเหตุดังนี้สัมประสิทธิ์การทำนายจึงไม่เหมาะสมที่จะนำมาใช้ เพราะว่าการเปลี่ยนแปลงเพียงเล็กน้อยของมันสามารถที่จะมีผลทำให้เกิดการเปลี่ยนแปลงอย่างมากมายกับคุณลักษณะสมบัติของตัวกรองที่ใช้ใน การสังเคราะห์ ส่วนค่าสัมประสิทธิ์การสะท้อนจะมีค่าอยู่ในช่วง ± 1 หรือทำให้ตัวกรองมีสถานะเสถียรอยู่ตลอดเวลา

2.4.3.7 ขั้นตอนการคำนวณค่าสัมประสิทธิ์พาร์คอร์

จากข้างต้นในเรื่องของพาร์คอร์นั้น สามารถนำมาประยุกต์ใช้ได้โดยตรงทั้งวิธีออโต-คอรีเลชัน และวิธีโควาเรียนซ์ ขั้นตอนนี้จะเป็นการเข้าสู่รายละเอียดในการแก้สมการเพื่อนำไปสู่การหาสัมประสิทธิ์การทำนายเชิงเส้นและสัมประสิทธิ์ของพาร์คอร์ k_m ตามลำดับโดยใช้วิธีออโตคอรีเลชัน จากสมการที่ 2.4.53

$$A_m(z) = A_{m-1}(z) + k_m B_{m-1}(z) \qquad .....2.4.74$$

ทำการหาค่า Am(z) จาก $A_{m-1}(z)$ และ $B_{m-1}(z)$ และสามารถใช้สมการนี้หาค่า $A_m(z)$ ใดๆ ได้โดยการใช้วิธีคำนวณซ้ำ (recursive) โดยเริ่มต้นจากสมการที่ 2.4.52 คือ $A_0(Z) = 1$ เช่น

ที่ m = 1 $\quad A_1(z) = 1 + k_1 B_0(z)$

ที่ m = 2 $\quad A_2(z) = 1 + k_1 B_0(z) + k_2 B_1(z)$

หรือสรุปเป็นสมการทั่วๆไปได้ว่า

$$A_m(z) = 1 + \sum_{i=1}^{m} k_i B_{i-1}(z) \qquad \text{เมื่อ } m > 0 \qquad .....2.4.74$$

จากสมการที่ 2.4.55 เขียนใหม่ได้เป็น

$$\langle A_m(z), B_m(z) \rangle = \langle 1, B_m(z) \rangle = \langle A_m(z), z^{-(m+1)} \rangle \qquad .....2.4.75$$

จากสมการที่ 2.4.43 โดยอาศัยสมการที่ 2.4.75 จะได้ว่า

$$\alpha_m = \|A_m(z)\|^2 = \langle A_m(z), A_m(z) \rangle = \langle 1, A_m(z) \rangle \qquad .....2.4.76ก$$

$$\beta_m = \|B_m(z)\|^2 = \langle B_m(z), B_m(z) \rangle = \langle z^{-(m+1)}, B_m(z) \rangle \qquad .....2.4.76ข$$

นอร์มสแควร์ของสมการที่ 2.4.74 มีค่าเป็น

$$\|A_m(z)-1\|^2 = \sum_{i=1}^{m} k_i^2 \beta_{i-1} \qquad \text{.....2.4.77}$$

แตกค่าทางซ้ายมือของสมการที่ 2.4.77 ให้อยู่ในรูปแบบตามสมการที่ 2.4.44 ได้เป็น

$$\|A_m(z)-1\|^2 = \|A_m(z)\|^2 - 2\langle A_m(z),1\rangle + \|1\|^2$$

$$= \|1\|^2 - \alpha_m \qquad \text{.....2.4.78}$$

เทียบสมการที่ 2.4.77 กับ 2.4.78 จะได้ว่า

$$\alpha_m = \|1\|^2 - \sum_{i=1}^{m} k_1^2 \beta_{i-1} \qquad \text{.....2.4.79}$$

แทนค่า m ด้วย m+1 แล้วลบ α_{m+1} ออกด้วย α_m จะได้เป็น

$$\alpha_{m+1} = \alpha_m - k^2{}_{m+1}\beta_m$$

$$\alpha_{m+1} = (1-k_{m+1}^2)\alpha_m \qquad \text{เนื่องจาก} \alpha_m = \beta_m \qquad \text{.....2.4.80}$$

จากสมการที่ 2.4.53 เพิ่มค่าของดัชนีไปอีกหนึ่งเขียนใหม่ได้เป็น

$$A_{m+1}(z) = A_m(z) + k_{m+1}B_m(z) \qquad \text{.....2.4.81}$$

และจากสมการที่2.5.54 เพิ่มดัชนีไปอีกหนึ่งก็จะได้เป็น

$$\left\langle A_m(z), z^{-(m+1)} \right\rangle + k_{m+1}\beta_m = 0 \qquad \text{.....2.4.82}$$

จากสมการที่ 2.4.60 และถ้า β_m ไม่เท่ากับศูนย์แล้วได้ว่า

$$k_{m+1} = -\frac{1}{\alpha_m}\left\langle A_m(z), z^{-(m+1)}\right\rangle$$

$$= -\frac{1}{\alpha_m}\sum_{i=0}^{m} r(m+1-i)a_{mi} \qquad \text{.....2.4.83}$$

เพราะฉะนั้นจะสามารถหาสัมประสิทธิ์ของตัวกรองได้โดยการแทนค่า k_{m+1} ลงในสมการที่ 2.4.81 ได้เป็น

$$a_{m+1,0} = 1 \qquad \text{..... 2.4.84ก}$$

$$a_{m+1,i} = a_{mi} + k_{m+1}b_{mi} \qquad \text{..... 2.4.84ข}$$

$$a_{m+1,m+1} = k_{m+1} \qquad \text{..... 2.4.84ค}$$

ค่า b_{mi} ในสมการ 2.4.84ข อาจจะแทนได้ด้วยสมการที่ 2.4.61 ซึ่งเขียนใหม่ได้เป็น

$$b_{mi} = a_{m,m+1-i} \qquad \text{โดย } i = 1,2,\ldots,m+1 \qquad \text{..... 2.4.85}$$

2.4.3.8 สรุปขั้นตอนการคำนวณค่าสัมประสิทธิ์พาร์คอร์

ขั้นที่ 1 คำนวณค่าอัตสหสัมพันธ์

กำหนดให้ในหนึ่งเฟรมของการวิเคราะห์ มีจำนวนสัญญาณสุ่มเท่ากับ N ตัวอย่าง และฟิลเตอร์มีจำนวนออร์เดอร์เท่ากับ M ตามวิธีอัตสหสัมพันธ์ ค่าอัตสหสัมพันธ์ที่ใช้ในการคำนวณขั้นต่อ ๆ ไป จะมีจำนวนเท่ากับ M+1 คือ

$$r(k)=\sum_{n=0}^{N-1-k}x(n)x(n+k) \qquad \text{โดย } k=0,1,\ldots,M \qquad \ldots\ldots 2.4.86$$

โดย x(n) เมื่อ n = 0,1,...,N-1 คือสัญญาณเสียงที่สุ่มมาในหนึ่งเฟรม

ขั้นที่ 2 คำนวณค่าเริ่มต้น

จากสมการ (2.4.83) แทน m = 0 และ สมการ (2.4.84ค) จะได้

$$\alpha_0=\langle A_0(z),A_0(z)\rangle=r(0) \qquad \ldots\ldots 2.4.87$$

ซึ่งจากเงื่อนไขเริ่มต้นตามสมการ (2.4.36ก) a_{00} = 1 และ α_0 = r(0) ดังนั้น

$$k_i=-r(1)/r(0) \qquad \ldots\ldots 2.4.88$$

จากสมการ (2.4.84ก) และ (2.4.84ข) แทนค่า m = 0 ได้

$$a_{10}=1 \qquad \text{และ } a_{11}=k_1 \qquad \ldots\ldots 2.4.89$$

จากสมการ (2.4.80) แทน m = 0 จะได้

$$\alpha_1=(1-k_1^2)\alpha_0=(1-k_1^2)r(0) \qquad \ldots\ldots 2.4.90$$

ขั้นที่ 3 ทำการคำนวณแบบรีเคอร์ซีฟ

ขั้นตอนนี้จะนำค่าเริ่มต้นจากขั้นตอนที่แล้ว มาทำการคำนวณต่อ ๆ กัน แบบรีเคอร์ซีฟ โดยอาศัยสมการ (2.4.80) (2.4.83) (2.4.84) และ (2.4.53) ในการหาค่าสัมประสิทธิ์พาร์คอร์ จนครบจำนวนออร์เดอร์ คือ k_i โดย $i = 1,2,...,M$ และพร้อมกันนั้นก็จะได้ค่าสัมประสิทธิ์ของตัวทำนาย a_i โดย $i = 0,1,...,M$ จะต้องมีการคำนวณต่อ ๆ กันไปทั้งหมด M-1 รอบ คือ แทนค่า $m = 1,2,...,M-1$ เมื่อคำนวณเสร็จขั้นที่ $m = M-1$ จะได้คำตอบครบตามที่ต้องการ รวมทั้งค่า α_m หรือค่าผิดพลาดยกกำลังสองรวมในเฟรม

2.5 การเปรียบเทียบและการรู้จำเสียงพูด

ในการรับรู้เสียงพูด ก็เป็นลักษณะหนึ่งของการรับรู้รูปแบบ (Pattern Recognition) คือ จะเป็นการเปรียบเทียบระหว่าง แบบทดสอบ (Test Pattern) กับ แบบอ้างอิง (Reference Pattern) ซึ่งเป็นรูปแบบที่เราทราบและเก็บค่าไว้ล่วงหน้า

ขั้นตอนในการรับรู้รูปแบบแบ่งเป็น 2 ขั้นตอน คือ

ก ขั้นเรียนรู้ **(Learning)** จะเป็นการสร้างกลุ่มของแบบอ้างอิงในการรับรู้เสียงพูด ในขั้นนี้จะทำการวิเคราะห์เสียงพูดก่อน แล้วเก็บลักษณะของเสียงในรูปของพารามิเตอร์ พร้อมกับป้ายกำกับเพื่อจะใช้เปรียบเทียบในขั้นต่อไป รายละเอียดของการวิเคราะห์เสียงพูดได้กล่าวไว้ในบทที่ 2

ข ขั้นรับรู้ **(Recognition)** จะเป็นการทดสอบการรับรู้ ระหว่างแบบอ้างอิง กับ แบบทดสอบ โดยจะทำการเปรียบเทียบพารามิเตอร์ของแบบทดสอบ กับแบบอ้างอิงทั้งหมด แบบอ้างอิงที่ถูกเลือกคือ แบบอ้างอิงที่มีพารามิเตอร์ใกล้เคียงกับแบบทดสอบที่สุด เทคนิคในการเปรียบเทียบมีอยู่หลายวิธี ดังจะกล่าวต่อไป

2.5.1 การจัดกลุ่มเพื่อสร้างแบบอ้างอิง

การสร้างกลุ่มของแบบอ้างอิงเป็นขั้นตอนในส่วนของการเรียนรู้ โดยเริ่มต้นจากการนำข้อมูลสัญญาณเสียงที่ได้จากส่วนของการวิเคราะห์ มาทำการสร้างเป็นกลุ่มของแบบอ้างอิง ในระบบการรับรู้เสียงพูดแบบต่างบุคคล จะใช้แบบอ้างอิงของคำหนึ่ง ๆ จากผู้พูดจำนวนมาก เพื่อที่จะได้ครอบคลุมถึงความแปรปรวนต่าง ๆ ที่จะเกิดขึ้นระหว่างผู้พูดแต่ละคน แต่อย่างไรก็ตามผลที่เกิดขึ้นตามมาเมื่อจำเป็นจะต้องมีข้อมูลจำนวนมากในกลุ่มของแบบอ้างอิง คือ

ก เวลาที่ใช้ในการตอบสนอง ในการรับรู้เสียงพูด ขั้นตอนการเปรียบเทียบแบบทดสอบกับแบบอ้างอิงทั้งหมดที่มี เป็นขั้นตอนสำคัญในการรับรู้ เมื่อแบบอ้างอิงมีจำนวนมากการเปรียบเทียบจึงจำเป็นที่จะต้องอาศัยเวลาเพิ่มขึ้นด้วย

ข เนื้อที่ในหน่วยความจำสำรองที่ใช้ในการเก็บแบบอ้างอิง เมื่อมีแบบอ้างอิงจำนวนมากที่จำเป็นต้องใช้ในการเปรียบเทียบ เนื้อที่ในหน่วยความจำสำรองก็จำเป็นต้องมีเพิ่มขึ้นด้วย

ค ความถูกต้องในการรับรู้ เนื่องจากจำนวนแบบอ้างอิงในแต่ละคำไม่ได้เป็นสัดส่วนโดยตรงกับความถูกต้องในการรับรู้ กล่าวคือเมื่อเราเพิ่มจำนวนแบบอ้างอิงในแต่ละคำไปจนถึงระดับหนึ่ง ความถูกต้องในการรับรู้จะเริ่มคงที่ และในบางครั้งอาจจะมีค่าลดลงด้วย

จากเหตุผลดังกล่าว จึงทำให้ความพยายามที่จะจัดกลุ่มของแบบอ้างอิงในแต่ละคำใหม่เพื่อให้ได้แบบอ้างอิงในจำนวนที่พอเหมาะ และสามารถใช้เป็นตัวแทนของแบบอ้างอิงที่มีอยู่ทั้งหมด อัลกอริธึมที่ใช้ในการแบ่งกลุ่มมีอยู่เป็นจำนวนมาก สำหรับอัลกอริธึมที่จะกล่างถึง คือ การหาค่าเฉลี่ย K (K-means algorithm) ซึ่งประกอบด้วยขั้นตอนต่าง ๆ ดังนี้

2.5.1.1 การหาจุดศูนย์กลางของกลุ่ม (Center Cluster)

ในขั้นตอนนี้จะทำการกำหนดจำนวนกลุ่มของแบบอ้างอิงให้มีค่าเท่ากับ K และเริ่มต้นจากแบบอ้างอิงที่มีความยาวมากที่สุดเป็นจุดศูนย์กลางของกลุ่มแรก จากนั้นจะหาจุดศูนย์กลางของคำถัดไปได้จากแบบอ้างอิงที่มีระยะทางห่างจากจุดศูนย์กลางของกลุ่มแรกมากที่สุดและจุดศูนย์กลางของกลุ่มถัดไปจะได้จาก แบบอ้างอิงที่มีระยะทางห่างจากจุดศูนย์กลางของกลุ่มทั้งสองมากที่สุด ขั้นตอนนี้จะถูกทำซ้ำจนได้กลุ่มของแบบอ้างอิงครบตามจำนวน K กลุ่ม หรือค่าระยะทางที่มากที่สุดมีค่าน้อยกว่าระดับที่กำหนด วิธีนี้เรียกว่า*Farthest Neighbor*

2.5.1.2 การจัดกลุ่มใหม่ (Reclassification)

แบบอ้างอิงที่เหลืออยู่ในแต่ละคำ จะถูกจัดให้เข้าในแต่ละกลุ่ม โดยมีเงื่อนไขคือ ระยะทางของแบบอ้างอิงนี้กับแบบอ้างอิงในกลุ่มจะมีค่าเฉลี่ยน้อยที่สุด หลังจากที่แบบอ้างอิงทุกแบบถูกจัดเข้ากลุ่มเรียบร้อยแล้ว จะทำการหาจุดศูนย์กลางของแต่ละกลุ่มใหม่ ขั้นตอนนี้จะถูกทำซ้ำจนกว่าจุดศูนย์กลางของแต่ละกลุ่มจะไม่มีการเปลี่ยนแปลง อย่างไรก็ตามจากขั้นตอนดังกล่าวจะเห็นว่าการกำหนดค่าจำนวนกลุ่มที่เหมาะสมอาจจะทำได้ลำบาก ดังนั้นจึงได้มีการปรับปรุงเทคนิคนี้เป็นการกำหนดค่าระยะทางสูงสุดที่จะทำให้เกิดกลุ่มใหม่ขึ้น คือเมื่อแบบอ้างอิงใดที่ถูกเลือกให้เข้ากลุ่มด้วยระยะทางที่น้อยที่สุดแล้ว แต่ระยะทางนี้ยังคงมีค่ามากกว่าระดับที่กำหนด แบบอ้างอิงนี้จะถูกนำไปสร้างเป็นจุดศูนย์กลางของกลุ่มใหม่ทันที

2.5.2 การเปรียบเทียบแบบไดนามิคโปรแกรมมิง

เป็นที่ทราบกันดีอยู่แล้วว่า การเปลี่ยนแปลงอัตราเร็วในการเปล่งเสียง เป็นสาเหตุของการแกว่งไปมาของรูปแบบสัญญาณเสียงบนแกนเวลา การกำจัดผลของการแกว่งนี้เป็นปัญหาสำคัญประการหนึ่งในการรับรู้เสียงพูด ในยุคเริ่มแรกได้มีการเสนอวิธีการปรับปรุงเชิงเส้น (Linear Normalization) เพื่อที่จะกำจัดความแตกต่างของเวลาระหว่างรูปแบบสัญญาณเสียง 2 รูปแบบ โดยทำการเปลี่ยนรูปเชิงเส้น (Linear Transformation) บนแกนเวลา แต่อย่างไรก็ตามวิธีนี้ยังไม่ประสบความสำเร็จในการปรับปรุงความถูกต้องของการรับรู้เสียงพูด

เทคนิคของไดนามิคโปรแกรมมิงเป็นแนวความคิดใหม่ ที่จะใช้ในการกำจัดความแตกต่างของเวลาระหว่างรูปแบบสัญญาณ 2 รูปแบบ โดยการสร้างแบบจำลองของฟังก์ชันที่ไม่เป็นเชิงเส้น โดยให้ชื่อว่า ฟังก์ชันแรปปิง (Warpping Function) ความแตกต่างของเวลาจะถูกกำจัดออกไปได้โดยอาศัยแบบจำลองของฟังก์ชันนี้

2.5.2.1 ฟังก์ชันแรปปิง

เราสามารถแสดงสัญญาณเสียงออกมาให้อยู่ในรูปของเวคเตอร์ได้ คือ

$$A = a_1, a_2, \ldots, a_i, \ldots, a_I \qquad \ldots\ldots 2.5.1$$

$$B = b_1, b_2, \ldots, b_j, \ldots, b_J \qquad \ldots\ldots 2.5.2$$

ในการที่จะแก้ปัญหาความแตกต่างของเวลา ระหว่างรูปแบบของสัญญาณเสียง 2 รูปแบบนั้น สามารถทำได้โดย แทนที่จะอธิบายสัญญาณด้วยธรรมชาติของการแกว่งบนแกนเวลาหรือความแตกต่างของเวลา เราจะอธิบายด้วยระนาบ i-j ดังแสดงในรูปที่ 2.5.1 โดยรูปแบบ A หรือ B จะถูกแสดงบนแกน i และ j ตามลำดับ จากรูปจะได้ว่า ความแตกต่างของเวลาสามารถแสดงได้ด้วยลำดับของจุด c = (i , j) จะได้ว่า

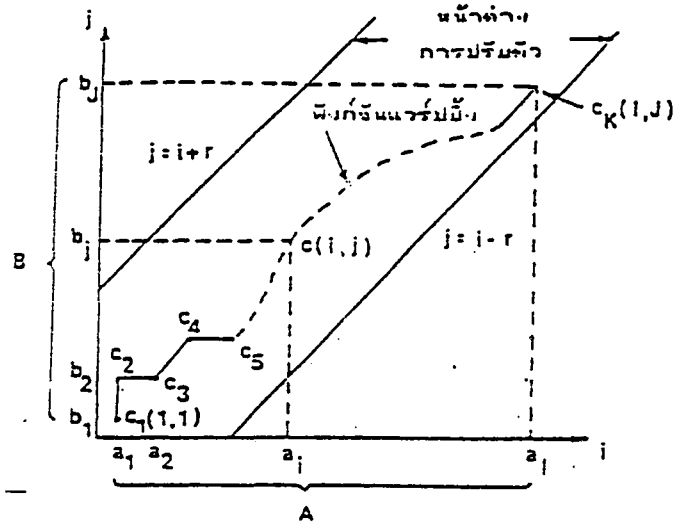
$$F = c(1), c(2), \ldots, c(k), \ldots, c(K) \qquad \ldots\ldots 2.5.3$$

โดย $c(k) = (i(k), j(k))$

จากคู่ลำดับนี้จะสามารถมองเป็นฟังก์ชัน ซึ่งจะโยงแกนเวลาของรูปแบบสัญญาณเสียง A ไปยัง B เราเรียกฟังก์ชันนี้ว่า ฟังก์ชันแรปปิง (Wrapping Function) ดังแสดงในรูปที่ 3.2.1 ถ้าไม่มีความแตกต่างของเวลาระหว่างสัญญาณทั้งสองแล้ว จะได้ว่าฟังก์ชันแรปปิงจะมีลักษณะเป็นเส้นทแยงมุม i = j ฟังก์ชันนี้จะมีการเปลี่ยนแปลงไปตามความแตกต่างของเวลาที่เกิดขึ้น

ให้ความแตกต่างระหว่างเวคเตอร์ a_i และ b_j เป็น

$$d(c) = d(i,j) = \|a_i - b_j\| \qquad \text{..... 2.5.4}$$



รูปที่ 2.5.1 แสดงให้เห็นลักษณะของฟังก์ชันแรปปิง

ดังนั้นจะได้ว่า ผลต่างรวมของฟังก์ชันแรปปิง F คือ

$$E(F) = \sum_{k=1}^{K} d(c(k))w(k) \qquad \text{..... 2.5.5}$$

โดยที่ w(k) เป็นสัมประสิทธิ์น้ำหนัก (Weight Coefficient) ซึ่งจะทำให้การวัด E(F) มีความยืดหยุ่นขึ้น ในการหาทางเดินของฟังก์ชันแรปปิงที่ดีที่สุดนั้น จะได้ค่าผลต่างรวมที่น้อยที่สุดตามทางเดินนั้นและเพื่อที่จะให้ได้ค่าคงตัวบนแกนเวลา เราสามารถเขียนระยะทางระหว่างรูปแบบสัญญาณเสียง A และ B ได้เป็น

$$D(A,B) = \underset{F}{MIN}\left[\frac{\sum_{k=1}^{K} d(ck))w(k)}{\sum_{k=1}^{K} w(k)}\right] \quad \text{.... 2.5.6}$$

โดยที่ตัวหาร $\sum$ w(k) จะเป็นตัวชดเชยผลของค่า k คือ จำนวนจุดของฟังก์ชันแรปปิง จากสมการที่ 2.5.6 จะได้รูปทั่วไปของระยะทางการปรับตัวตามแกนเวลา (Time-normalized Distance) เราเรียกวิธีการที่จะแก้สมการนี้ได้อย่างมีประสิทธิภาพว่า ไดนามิคโปรแกรมมิง (Dynamic Programming) อย่างไรก็ตามการหาฟังก์ชันแรปปิงที่เหมาะสมนั้น ยังขึ้นอยู่กับข้อกำหนดของฟังก์ชันแรปปิงและการระบุถึงสัมประสิทธิ์น้ำหนักอีกด้วย

ในการหาค่าฟังก์ชันแรปปิง ได้มีการกำหนดเงื่อนไขไว้ดังนี้ คือ

ก. เงื่อนไขโมโนโทนิค **(Monotonic Condition)**

$$i(k-1) \le i(k) \text{ และ } j(k-1) \le j(k) \quad \text{..... 2.5.7}$$

ข. เงื่อนไขความต่อเนื่อง **(Continuity Condition)**

$$i(k) - i(k-1) \le 1 \text{ และ } j(k) - j(k-1) \le 1 \quad \text{..... 2.5.8}$$

จากเงื่อนไขทั้ง 2 นี้ ทำให้ได้ความสัมพันธ์ระหว่างจุด 2 จุดคือ

$$c(k-1) = \begin{bmatrix} (i(k), j(k)-1) \\ (i(k)-1, j(k)-1) \\ (i(k)-1, j(k)) \end{bmatrix} \quad \text{..... 2.5.9}$$

ค. เงื่อนไขขอบเขต **(Boundary Condition)**

$$i(1) = 1, j(1) = 1 \text{ และ } i(K) = I, j(K) = J \quad \text{.....2.5.10}$$

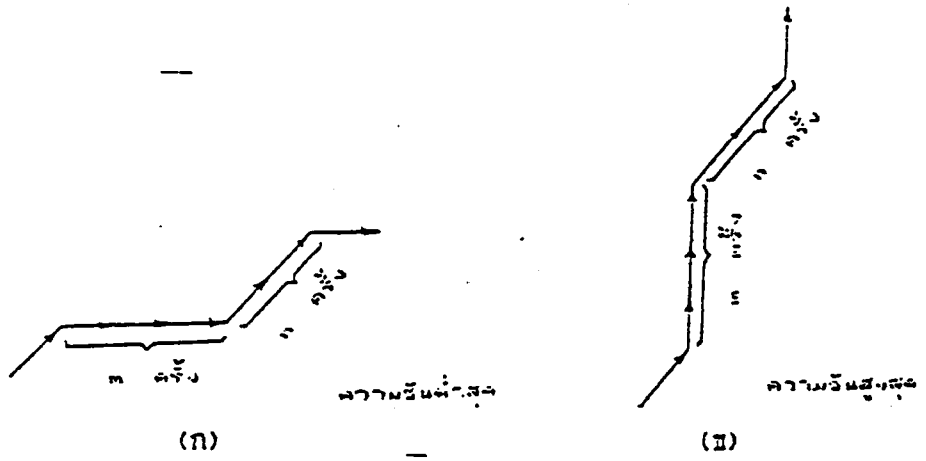
ง. *เงื่อนไขการปรับขอบเขต* **(Adjustment Window Condition)**

$$|i(k)-j(k)| \le r \qquad \text{..... 2.5.11}$$

โดยที่ r เป็นความกว้างของหน้าต่าง ซึ่งจะมีค่าเป็นเลขจำนวนเต็มบวก

จ. *เงื่อนไขการเปลี่ยนความชัน* **(Slope Constraint Condition)** เนื่องจากความแปรปรวนของความยาวในคำพูด สามารถเกิดขึ้นได้มาก เช่น เมื่อเราเปรียบเทียบคำพูด ก ซึ่งสั้นมาก เมื่อเทียบกับคำพูด ข ในกรณีนี้จะทำให้ได้ฟังก์ชันแรปปิงที่มีค่าผิดเพี้ยนไปมาก ดังนั้นจึงมีการกำหนดให้มีเงื่อนไขของการเปลี่ยนความชันขึ้น โดยเงื่อนไขนี้จะกำหนดให้จุด c(k) สามารถเคลื่อนที่ไปตามแกน i หรือ j โดยไม่เปลี่ยนทิศทางได้ m ครั้ง จากนั้นจุด c(k) จะไม่สามารถเคลื่อนที่ไปในทิศทางเดิมได้อย่างน้อย n ครั้ง ดังแสดงในรูปที่ (2.5.2ก) และ (2.5.2ข) เงื่อนไขนี้จะถูกเขียนออกมาในรูปของ

$$P = n/m \qquad \text{..... 2.5.12}$$



(ก) (ข)

รูปที่ 2.5.2 เงื่อนไขการเปลี่ยนความชันของฟังก์ชันแรปปิง

จากสมการที่ (2.5.5) ถ้าเราให้ตัวหาร ซึ่งเรียกว่าสัมประสิทธิ์การปรับตัว (Normalized Coefficient) ไม่ขึ้นอยู่กับฟังก์ชันแรปปิง จะทำให้เขียนสมการนี้ใหม่ได้เป็น

$$D(A,B) = (1/N)\underset{F}{MIN}\left[\sum_{k=1}^{K} d(c(k))w(k)\right] \quad \text{..... 2.5.13}$$

โดย $$N = \sum_{k=1}^{K} w(k) \quad \text{..... 2.5.14}$$

เพื่อที่จะทำให้สามารถแก้สมการที่ (2.5.13) ได้ง่ายขึ้น เราจึงมีการกำหนดค่าของ w(k) ซึ่งเรียกว่า สัมประสิทธิ์น้ำหนัก แบ่งออกได้เป็น 2 แบบ คือ

ก. แบบแกนเวลาสมมาตร **(Symmetric)** ในแบบนี้แกนเวลาทั้งสองจะถูกเปลี่ยนรูปให้อยู่บนแกนซั่วคราว ซึ่งกำหนดเป็นแกนร่วม ดังนั้นผลรวมของสัมประสิทธิ์น้ำหนักก็คือ ผลรวมของแกนร่วมนั่นเอง (I=I+j)

$$W(k) = (i(k) - i(k-1)) + (j(k) - j(k-1)) \quad \text{..... 2.5.15}$$

จะได้ว่า $$N = I + J \quad \text{..... 2.5.16}$$

โดย I และ J เป็นความยาวของสัญญาณ A และ B ตามลำดับ

ข. แบบแกนเวลาไม่สมมาตร **(Asymmertic)** ในแบบนี้เวลาจะถูกเปลี่ยนรูปจากแกนเวลาหนึ่งไปอยู่บนอีกแกนหนึ่ง ดังนั้นผลรวมของสัมประสิทธิ์น้ำหนักก็คือ ผลรวมของแกน i หรือ j เท่านั้น

$$W(k) = i(k) - i(k-1) \quad \text{..... 2.5.17}$$

จะได้ว่า $$N = I \quad \text{..... 2.5.18}$$

หรือ $w(k) = j(k) - j(k-1)$ จะได้ $N = J$

2.5.2.2 สมการไดนามิคโปรแกรมมิง

จากสมการที่ 3.2.10 สามารถแก้ได้โดยอาศัยทฤษฎีของไดนามิคโปรแกรมมิง ดังนี้

เงื่อนไขเริ่มต้น

$$g_1(c(1)) = d(c(1))w(1) \qquad \text{..... 2.5.19}$$

สมการไดนามิคโปรแกรมมิง

$$g_k(c(k)) = \min_{c(k-1)}\left[g_{k-1}(c(k-1)) + d(c(k))w(k)\right] \qquad \text{..... 2.5.20}$$

ระยะทางการปรับตัวตามแกนเวลา

$$D(A,B) = (1/N)g_k(c(K)) \qquad \text{..... 2.5.21}$$

จากสมการทั้งหมดนี้ สมมุติให้ c(0) = (0,0) และคิดสัมประสิทธิ์น้ำหนักแบบสมมาตร w(1) = 2 แทนค่าสมการ (2.5.15) ลงในสมการ (2.5.20) จะทำให้สามารถหาค่าต่าง ๆ ได้ เช่น ถ้ากำหนดให้เงื่อนไขของการเปลี่ยนความชัน P = 0 จะได้ว่า

เงื่อนไขเริ่มต้น

$$g(1,1) = 2d(1,1) \qquad \text{..... 2.5.22}$$

สมการไดนามิคโปรแกรมมิง

$$g(i,j) = \min\begin{bmatrix} g(i,j-1) + d(i,j) \\ g(i-1,j-1) + 2d(i,j) \\ g(i-1,j) + d(i,j) \end{bmatrix} \qquad \text{..... 2.5.23}$$

ระยะทางการปรับตัวตามแกนเวลา

$$D(A,B) = (1/N)g(I,J) \qquad \text{..... 2.5.24}$$

ในตารางที่ 2.5.1 จะสรุปให้เห็นถึงสมการไดนามิคโปรแกรมมิงในลักษณะต่าง ๆ ทั้งแบบแกนเวลาสมมาตรและแกนเวลาไม่สมมาตร กับการเปลี่ยนแปลงเงื่อนไขความชัน

2.5.3 <u>กฎการตัดสินใจ</u> (Decission Rules)

ผลที่ได้จากการเปรียบเทียบแบบไดนามิคโปรแกรมมิง คือระยะทางระหว่างแบบทดสอบกับแบบอ้างอิงแต่ละแบบ ระยะทางที่ได้ทั้งหมดจะนำมาผ่านขั้นตอนการตัดสินใจ เพื่อหาผลลัพธ์ของการรับรู้ กฎการตัดสินใจ (Decision Rules) ที่ใช้ในการรับรู้เสียงพูดมีอยู่ 3 แบบ คือ

***2.5.3.1 กฎ Nearest Neighbor* (NN)**

ผลลัพธ์ของการรับรู้จากกฎการตัดสินใจนี้ ได้แก่ แบบอ้างอิงที่มีระยะทางจากแบบทดสอบน้อยที่สุด กฎนี้เหมาะสำหรับระบบการรับรู้เสียงพูดแบบบุคคลเดี่ยว

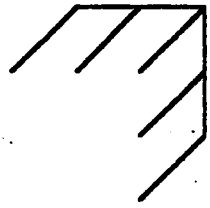
***2.5.3.2 กฎ K-Nearest Neighbor* (KNN)**

เป็นกฎที่ดัดแปลงมาจากกฎ NN อีกทีหนึ่ง เพื่อให้เหมาะสมกับระบบการรับรู้เสียงพูดแบบต่างบุคคล ซึ่งมีแบบอ้างอิงในแต่ละคำมากกว่า 1 แบบ กฎนี้ได้แบ่งออกเป็น 2 แบบ คือ

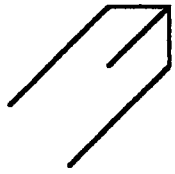
ก. แบบที่ 1 จะทำการหาแบบอ้างอิงจำนวน K แบบ ที่มีระยะทางห่างจากแบบทดสอบน้อยที่สุด ผลลัพธ์ของการรับรู้จะได้จาก แบบอ้างอิงที่มีจำนวนมากที่สุดในจำนวน K แบบที่ได้รับการเลือก

ข. แบบที่ 2 จะทำการหาแบบอ้างอิงจำนวน K แบบในแต่ละคำ โดยที่ทั้ง K แบบมีระยะทางห่างจากแบบทดสอบน้อยที่สุด ผลลัพธ์ของการเรียนรู้จะได้จากแบบอ้างอิงที่มีค่าเฉลี่ยของระยะทางในจำนวน K แบบที่ได้รับเลือกมาน้อยที่สุด วิธีการนี้จะได้ผลลัพธ์ที่ถูกต้องมากขึ้นโดยกำหนดค่า K เป็น 2 หรือ 3 เมื่อเทียบกับกฎ NN อย่างไรก็ตามวิธีนี้จะทำให้เวลาที่ใช้ในการรับรู้เพิ่มขึ้นด้วย

ในระบบการรับรู้เสียงพูดทั่วไป จะมีการกำหนดค่าระยะทางมากที่สุดที่ยอมรับได้ (Reject Value) เพื่อเพิ่มความถูกต้องของการรับรู้ในกรณีที่ผลลัพธ์ที่ได้จากการตัดสินใจมีค่าใกล้เคียงกันมาก วิธีการกำหนดค่านี้ต้องใช้ความระมัดระวังเป็นอย่างมาก เนื่องจากถ้าเรากำหนดค่านี้สูงเกินไปหรือไม่ได้กำหนด โอกาสที่จะทำให้ผลลัพธ์ของการรับรู้ผิดพลาดไปจะมีได้มาก แต่ถ้าเรากำหนดค่านี้ต่ำเกินไป จะทำให้ผลลัพธ์ของการรับรู้สำหรับแบบอ้างอิงที่ถูกต้องเกิดขึ้นได้ยาก ดังนั้นโดยทั่วไปค่าระยะทางนี้มักจะได้มาจากการทดลอง เพื่อหาค่าที่เหมาะสมสำหรับกลุ่มของแบบอ้างอิงหนึ่งๆ เท่านั้น

P	Schematic explanation	Symmetric/ Asymmetric	DP-equation $g(i,j) =$
0		Symmetric	$\min \begin{bmatrix} g(i,j-1)+d(i,j) \\ g(i-1,j-1)+2d(i,j) \\ g(i-1,j)+d(i,j) \end{bmatrix}$
		Asymmetric	$\min \begin{bmatrix} g(i,j-1) \\ g(i-1,j-1)+d(i,j) \\ g(i-1,j)+d(i,j) \end{bmatrix}$
1/2		Symmetric	$\min \begin{bmatrix} g(i-1,j-3)+2d(i,j-2)+d(i,j-1)+d(i,j) \\ g(i-1,j-2)+2d(i,j-1)+d(i,j) \\ g(i-1,j-1)+2d(i,j) \\ g(i-2,j-1)+2d(i-1,j)+d(i,j) \\ g(i-3,j-1)+2d(i-2,j)+d(i-1,j)+d(i,j) \end{bmatrix}$
		Asymmetric	$\min \begin{bmatrix} g(i-1,j-3)+\dfrac{d(i,j-2)+d(i,j-1)+d(i,j)}{3} \\ g(i-1,j-2)+\dfrac{d(i,j-1)+d(i,j)}{2} \\ g(i-1,j-1)+d(i,j) \\ g(i-2,j-1)+d(i-1,j)+d(i,j) \\ g(i-3,j-1)+d(i-2,j)+d(i-1,j)+d(i,j) \end{bmatrix}$
1		Symmetric	$\min \begin{bmatrix} g(i-1,j-2)+2d(i,j-1)+d(i,j) \\ g(i-1,j-1)+2d(i,j) \\ g(i-2,j-1)+2d(i-1,j)+d(i,j) \end{bmatrix}$
		Asymmetric	$\min \begin{bmatrix} g(i-1,j-2)+\dfrac{d(i,j-1)+d(i,j)}{2} \\ g(i-1,j-1)+d(i,j) \\ g(i-2,j-1)+d(i-1,j)+d(i,j) \end{bmatrix}$

ตารางที่ 2.5.1 แสดงสมการไดนามิคโปรแกรมมิงต่างๆ เมื่อเงื่อนไขการเปลี่ยนความชันเปลี่ยนไป

2		Symmetric	$\min\begin{bmatrix} g(i-1,j-3)+2d(i-1,j-2)+2d(i,j-1)+d(i,j) \\ g(i-1,j-1)+2d(i,j) \\ g(i-3,j-2),+2d(i-2,j-1)+2d(i-1,j)+d(i,j) \end{bmatrix}$
		Asymmetric	$\min\begin{bmatrix} g(i-2,j-3)+\dfrac{d(i-1,j-2)+d(i,j-1)+d(i,j)}{3} \\ g(i-1,j-1)+d(i,j) \\ g(i-3,j-2)+d(i-2,j-1)+d(i-1,j)+d(i,j) \end{bmatrix}$

ตารางที่ 2.5.1 แสดงสมการไดนามิคโปรแกรมมิงต่างๆ เมื่อเงื่อนไขการเปลี่ยนความชันเปลี่ยนไป(ต่อ)

บทที่ 3
ทฤษฎีการออกแบบส่วนฮาร์ดแวร์

3.1 การปรับสภาพสัญญาณ

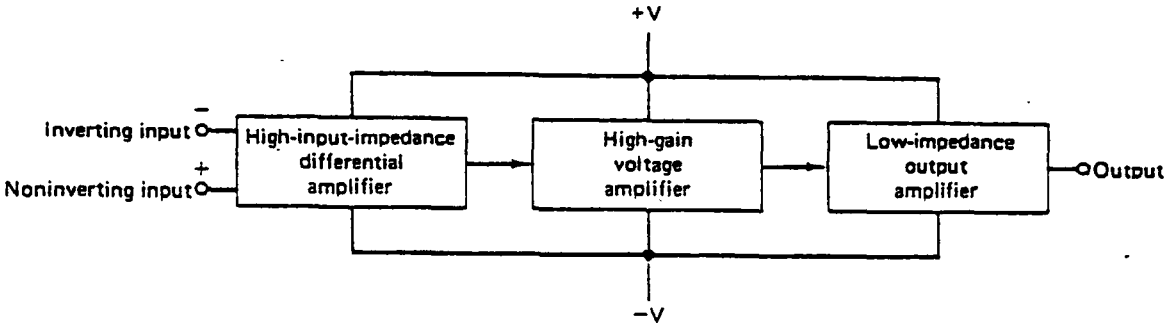
สัญญาณเสียงพูดที่ผ่านไมโครโฟนมาแล้ว จะเป็นสัญญาณที่มีแรงดัน สัญญาณนี้จะมีขนาดเล็ก มีสัญญาณรบกวน และที่สำคัญสัญญาณดังกล่าว ไม่สามารถเข้ากันได้กับวงจรการทำงานภาคต่อๆไป จึงต้องมีการปรับสภาพสัญญาณกันก่อน ซึ่งออปแอมป์เป็นอุปกรณ์ที่นิยมใช้ในการปรับสภาพสัญญาณ

3.1.1 คุณสมบัติของออปแอมป์

ออปแอมป์ คือ อุปกรณ์ชนิดหนึ่งที่ถูกออกแบบมาให้สามารถทำงานได้หลายรูปแบบ และยังเน้นความสะดวกในการนำไปใช้งานด้วย โดยสามารถประกอบเป็นวงจรได้โดยการต่อร่วมกับอุปกรณ์ภายนอกเพียงไม่กี่ตัวเท่านั้น ในอดีตออปแอมป์จะประกอบขึ้นจากอุปกรณ์หลายตัว ที่ถูกนำมาบรรจุไว้ในชิ้นเดียวกัน ซึ่งนอกจากจะทำให้ออปแอมป์มีขนาดใหญ่ และยังอาจมีประสิทธิภาพค่อนข้างต่ำด้วย แต่ปัจจุบันสามารถซื้อออปแอมป์ในรูปแบบวงจรรวม (IC) ได้ตามท้องตลาด และจากคุณสมบัติไอซีออปแอมป์ที่ได้พัฒนาขึ้นทำให้อุปกรณ์ชนิดนี้เป็นที่รู้จักกันอย่างแพร่หลาย

โดยทั่วไปแล้ว สามารถกล่าวได้ว่า ออปแอมป์คืออุปกรณ์โซลิดเสตท (Solid State) ชนิดหนึ่งซึ่งสามารถตรวจวัดระดับสัญญาณไฟตรง และ ไฟสลับได้ และยังสามารถนำไปใช้ขยายสัญญาณได้อีกด้วย ออปแอมป์พื้นฐาน จะต้องประกอบด้วยวงจรภายในภาคต่างๆดังนี้คือ

1. ดิฟเฟอเรนเชียลแอมป์ หรือวงจรขยายผลต่าง (Differential Amplifier) ที่มีอินพุทอิมพีแดนซ์สูงมาก
2. วงจรขยายแรงดันซึ่งมีอัตราขยายสูงมาก
3. วงจรขยายภาคเอาท์พุทที่มีเอาท์พุทอิมพีแดนซ์ต่ำมาก



รูปที่ 3.1.1 บล็อคไดอะแกรมของวงจรภายในภาคต่างๆ ของออปแอมป์

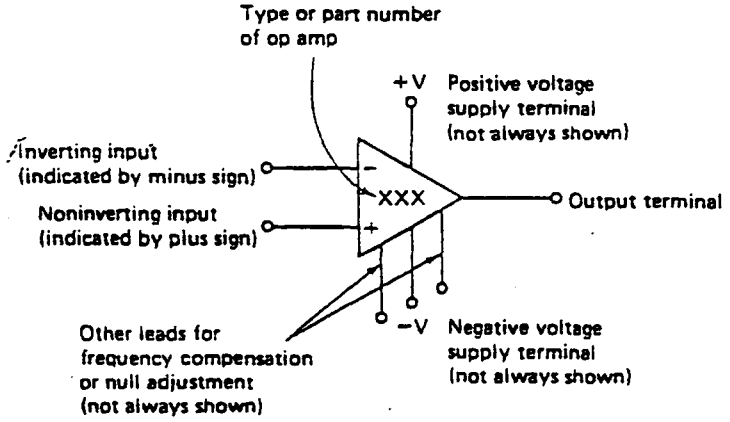
รูปที่ 3.1.1 แสดงบล็อคไดอะแกรมของวงจรภายในภาคต่างๆ ของออปแอมป์ดังกล่าวและจากรูปจะสังเกตว่า แรงดันไฟตรงที่จ่ายให้แก่ออปแอมป์มักประกอบไปด้วยไฟบวกและลบ เพื่อให้เอาท์พุทสามารถสวิงได้ทั้งซีกบวกและลบเทียบกับกราวด์

จากคุณสมบัติของออปแอมป์ดังที่กล่าวมา ทำให้สามารถสรุปคุณสมบัติที่สำคัญบางประการของออปแอมป์ในอุดมคติได้ดังนี้

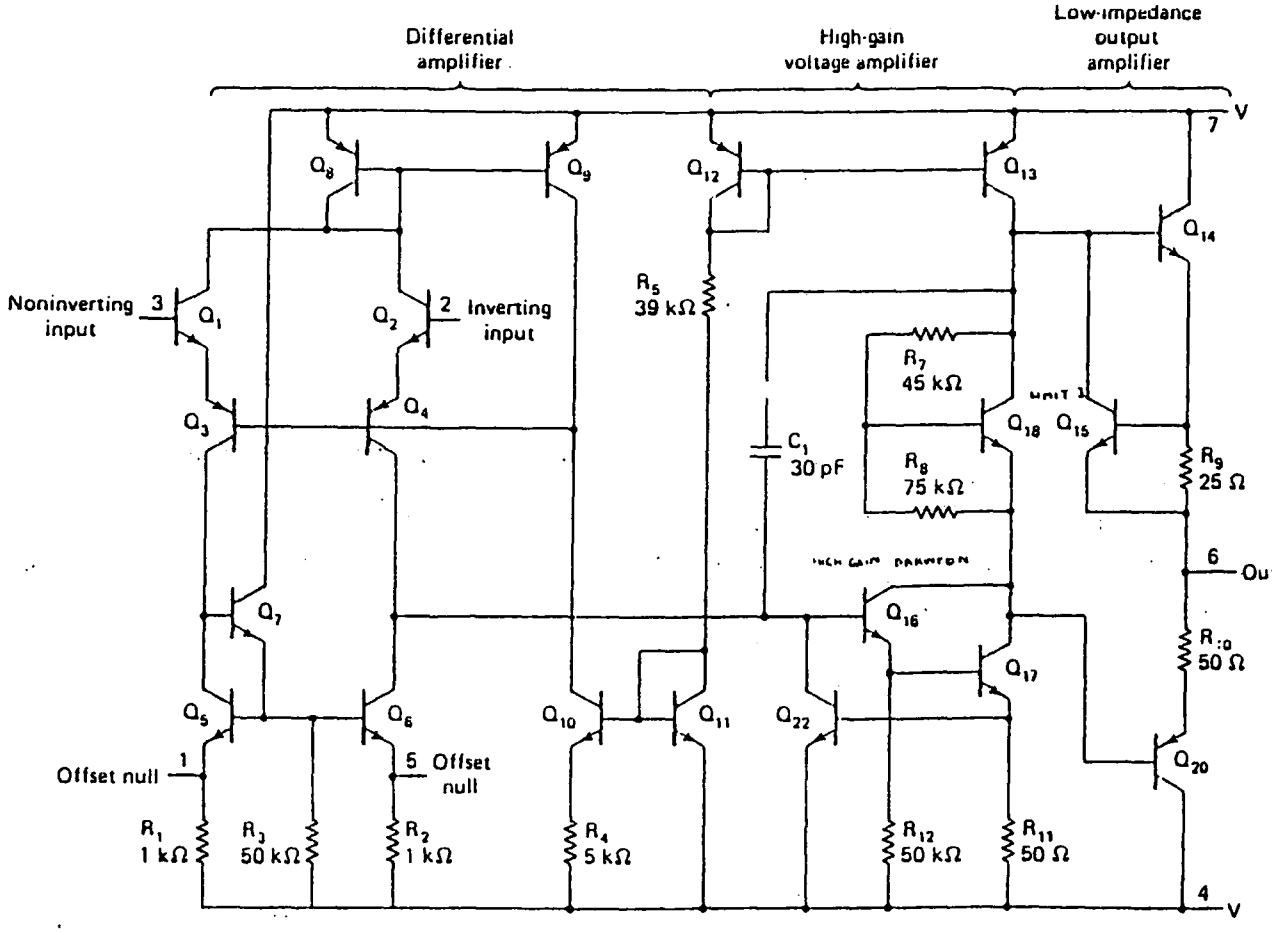
1.เนื่องจากอินพุทอิมพีแดนซ์ของออปแอมป์มีค่าสูงเป็นอนันต์ กระแสเข้าที่อินพุทจะต่ำจนเกือบเท่าศูนย์ หรืออีกนัยหนึ่ง ไม่มีกระแสอินพุทเข้าสู่ออปแอมป์เลย

2.อัตราขยายขณะเปิดลูป A (ขณะที่ยังไม่มีการป้อนกลับ) จะมีค่าสูงมาก ซึ่งหมายความว่าแรงดันระหว่างขั้วอินพุทควรมีค่าใกล้ศูนย์ (เนื่องจาก $V_{out} / A = V_{in}$)

3.เอาท์พุทอิมพีแดนซ์มีค่าต่ำมากจนไม่ทำตัวเป็นโหลดต่อภาคเอาท์พุทของวงจรขยาย



รูปที่ 3.1.2 แสดงสัญลักษณ์ทั่วไปของออปแอมป์



รูปที่ 3.1.3 วงจรภายในของออปแอมป์ เบอร์ 741

รูปที่ 3.1.2 แสดงสัญลักษณ์ทั่วไปของออปแอมป์ ซึ่งประกอบด้วยขั้วอินพุท 2 ขั้ว ขั้วสำหรับแหล่งจ่ายไฟ 2 ขั้ว ขั้วเอาท์พุท 1 ขั้ว และขั้วสำหรับปรับออฟเซ็ทหรือการชดเชยความถี่อีกสองขั้ว

ขั้วอินพุททั้งสองของออปแอมป์มีลักษณะต่างกันดังนี้ คือสำหรับขั้วลบ เมื่อป้อนไฟตรงหรือไฟสลับเข้าไป ในขณะที่ขั้วบวกต่อกับจุดอ้างอิงจุดหนึ่ง สัญญาณที่ออกมาที่เอาท์พุทจะกลับเฟสกับอินพุท 180 องศาส่วนการป้อนสัญญาณที่ขั้วบวก เอาท์พุทจะมีเฟสตรงกับอินพุท ดังนั้นจึงกล่าวได้ว่าเครื่องหมายที่อินพุท คือการแสดงเฟสของเอาท์พุทเทียบกับอินพุท ส่วนขั้วสำหรับออฟเซ็ท หรือชดเชยความถี่นั้นโดยมากมักจะไม่ถูกแสดงในวงจรทั่วไป

ในการนำออปแอมป์ไปใช้งานจริงนั้นเราไม่จำเป็นต้องศึกษาให้ลึกซึ้งถึงวงจรภายใน แต่อย่างไรก็ตามใน รูปที่ 3.1.3 จะแสดงวงจรภายในของออปแอมป์ เบอร์741 ซึ่งผู้สนใจสามารถศึกษารายละเอียดเพิ่มเติมจากสเป็ค หรือ รายละเอียดของผู้ผลิต จากวงจรในรูป 3.1.3 พบว่า IC ออปแอมป์ประกอบด้วยทรานซิสเตอร์หลายตัว และมีตัวเก็บประจุน้อยมาก โดยมีเหตุผลที่ว่า ตัวเก็บประจุจะกินเนื้อที่ค่อนข้างมาก และยังกันไม่ให้สัญญาณไฟตรงผ่านได้อีกด้วย แต่ตัวเก็บประ 30 PF ที่ต่อไว้ในวงจรนั้น มีหน้าที่ในการช่วยชดเชยความถี่เท่านั้น ซึ่งเราจะได้ศึกษาในบทต่อไป

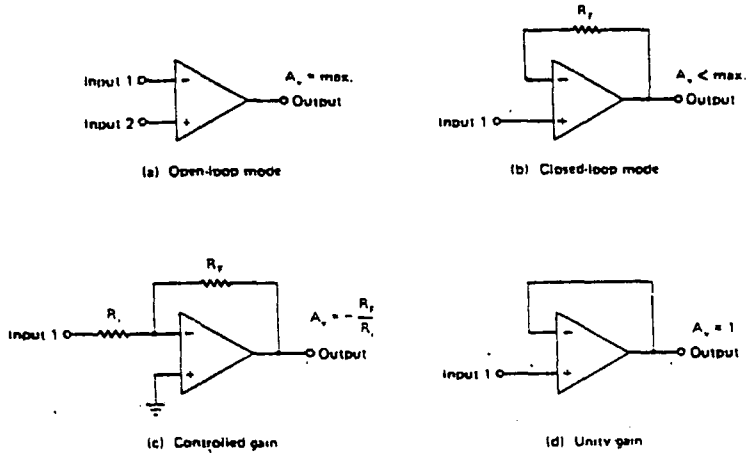
หากนำวงจรในรูป 3.1.3 มาเปรียบเทียบกับรูป 3.1.1 จะสามารถแยกวงจรออกเป็น 3 ภาพได้เช่นกัน โดยมีทรานซิสเตอร์ Q_1 และ Q_2 ทำหน้าที่เป็นดิฟเฟอร์เรนเชียลแอมป์ หรือทรานซิสเตอร์ Q_{16}, Q_{17} ซึ่งถูกต่อแบบดาร์ลิงตันทำหน้าที่เป็นวงจรขยายแรงดันที่มีอัตราขยายสูง ส่วนภาคเอาท์พุทจะประกอบด้วย Q_{14} และ Q_{20} ทรานซิสเตอร์ Q_{15} นั้นมีไว้สำหรับจำกัดกระแสและป้องกันออปแอมป์เสียหายเมื่อเอาท์พุทถูกลัดวงจร ส่วนอุปกรณ์ตัวอื่น ๆ จะมีหน้าที่ในการจัดไบอัส และช่วยในการขยายสัญญาณสำหรับออบแอมป์

3.1.2 <u>ลักษณะการทำงาน</u>

ออปแอมป์ในอุดมคติจะมีอัตราขยายเป็นอนันต์ แต่ในทางปฏิบัติ อัตราขยายอาจมีค่าสูงสุดเพียง 10,000 หรือ 1,000,000 เท่านั้น ซึ่งเรียกว่า อัตราขยายขณะเปิดลูป (AV) ดังรูปที่ 3.1.4(a) ในขณะที่เกิดความแตกต่างของแรงดันเพียงเล็กน้อยระหว่างขั้วอินพุททั้งสอง เอาท์พุทจะสามารถให้สัญญาณสูงขึ้นหลายเท่า (ตามค่าของอัตราขยาย AV) หากแต่จะถูกจำกัดด้วยขนาดของไฟเลี้ยงที่เราป้อนให้แก่ออปแอมป์ด้วย แต่ถึงเช่นนั้นก็ดี เอาท์พุทก็จะไม่สามารถมีค่าสูงสุดเท่ากับแรงดันจากแหล่งจ่ายไฟเลี้ยงได้จริง ทั้งนี้เกิดจากแรงดันที่ตกคร่อม Q_{14},R_9 หรือ R_{10},Q_{20} ในรูปที่ 3.1.3 ทำให้แรงดันเอาท์พุทสูงสุด อาจมีค่าประมาณ 90% ของแรงดันจากแหล่งจ่ายไฟเลี้ยงเท่านั้น

จากคุณสมบัติข้างต้น เราสมารถนำออปแอมป์ในขณะเปิดลูป ไปใช้งานเป็นคอมพาราเตอร์ (Comparator) หรือวงจรเปรียบเทียบแรงดันได้ โดยเอาท์พุทจะเปลี่ยนทันทีเมื่อมีความแตกต่างของแรงดันเกิดขึ้นระหว่างขั้วอินพุทของออปแอมป์

แต่ว่าการทำงานของออปแอมป์ยังไม่สิ้นสุดเพียงเท่านั้น นอกจากนี้จะพบว่าการใช้ออปแอมป์ในลักษณะของลูปปิด (มีการป้อนกลับ) จะทำให้ออปแอมป์ มีประโยชน์สูงมาก ขึ้น ดังรูปที่ 3.1.4 (b) การป้อนกลับในรูปใช้ตัวต้านทาน R_F เพียงตัวเดียว ซึ่งมีผลให้วงจรเสถียรภาพสูงขึ้น และมีสัญญาณรบกวนน้อยลง ในขณะเดียวกัน อัตราการขยายแรงดันจะลดลงด้วย

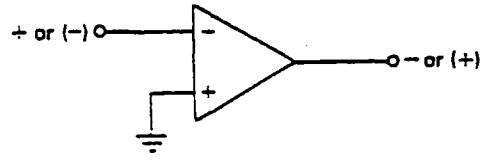


รูปที่ 3.1.4 วงจรพื้นฐานของออปแอมป์

วงจรในรูป 3.1.4 (c) แสดงการใช้ออปแอมป์โดยมีการป้อนสัญญาณเอาท์พุทกลับมายังอินพุท นอกจากนี้ยังสามารถควบคุมอัตราการขยายแรงดัน (ในขณะปิดลูป) ได้โดยอาศัยตัวต้านทาน 2 ตัว เท่านั้นทำให้อัตราการขยายแรงดัน Av มีค่าดังสมการ

$$A_v = -R_f / R_i \qquad \text{..... 3.1.1}$$

โดยที่เครื่องหมายลบแสดงถึงการกลับเฟสของเอาท์พุทเทียบกับอินพุท ส่วนวงจร 3.1.4 (d) แสดงการป้อนกลับในกรณีที่ A_V = 1 คุณสมบัติที่ได้กล่าวมาทั้งหมดนี้ คือคุณสมบัติทั่วไปของ ออป-แอมป์ ที่จะนำไปใช้ต่อไป และนอกจากนี้ คุณสมบัติอีกประการที่ควรศึกษาคือ เรื่องความสัมพันธ์ของขั้วต่างๆ ระหว่างอินพุทและเอาท์พุทของออปแอมป์ ซึ่งสรุปไว้แล้วใน รูปที่ 3.1.5



รูปที่ 3.1.5 อินพุท และเอาท์พุทของออปแอมป์

1. ถ้าขั้ว - มีศักดาเป็นบวกสูงกว่าขั้ว + ,เอาท์พุทเป็น -
2. ถ้าขั้ว - มีศักดาเป็นบวกต่ำกว่าขั้ว + ,เอาท์พุทเป็น -

3.1.3 <u>คุณสมบัติและพารามิเตอร์บางชนิดของออปแอมป์</u>

1) อินพุทอิมพีแดนซ์

ในทางอุดมคติควรมีค่าเท่ากับอนันต์ แต่ในความเป็นจริง อินพุทอิมพีแดนซ์จะมีค่าประมาณ 1 เมกะโอห์ม ($1*10^7$ โอห์ม) ค่าอินพุทอิมพีแดนซ์นี้ยิ่งมีค่ามากขึ้นเท่าใด ออปแอมป์ตัวนั้นก็จะทำงานได้ดีขึ้น นอกจากนี้ เมื่อนำออปแอมป์ไปใช้ในย่านความถี่สูง ควรระวังผลจากอินพุทคาปาซีแตนซ์ของวงจรด้วย ซึ่งมักมีค่าประมาณ 2 พิโคฟารัด ($2*10^2$ ฟารัด) เมื่อขั้วอินพุทขั้วหนึ่งต่อกับกราวด์

2) เอาท์พุทอิมพีแดนซ์

ดังที่ได้กล่าวมาแล้วว่า ออปแอมป์ในอุดมคติจะมีเอาท์พุทอิมพีแดนซ์เป็นศูนย์ แต่ในความเป็นจริง ค่านี้อาจมีได้ตั้งแต่ 25 ถึงหลายพันโอห์มขึ้นไป แต่อย่างไรก็ตามมักสมมติให้เอาท์พุท อิมพีแดนซ์ในวงจรมีค่าเป็นศูนย์ เพื่อง่ายต่อการคำนวณ และวิเคาะห์จากคุณสมบัติของออปแอมป์ที่มีอินพุทอิมพีแดนซ์สูง และมีเอาท์พุทอิมพิเดนซต่ำ ออปแอมป์จึงเปรียบเสมือนเป็นอุปกรณ์ซึ่งมีคุณสมบัติเป็นอิมพีแดนซ์แมทชิง (Impedance matching) ที่ดีตัวหนึ่ง

3) กระแสไบอัสด้านอินพุท

เนื่องจากอินพุทอิมพีแดนซ์ของออปแอมป์ไม่เป็นอนันต์ ดังนั้น จึงมีกระแสค่าน้อยๆ (ระดับเป็นนาโน (10^{9}) ถึง ไมโคร (10^{6}) แอมแปร์) ไหลผ่านขั้วอินพุททั้งสอง ซึ่งค่าเฉลี่ยของกระแสดังกล่าวถูกเรียกว่าเป็นกระแสไบอัสด้านอินพุท กระแสจะก่อให้เกิดความไม่สมดุลในวงจรภายใน ซึ่งจะเป็นผลกระทบต่อภาคเอาท์พุทด้วย ดังนั้น กระแสนี้ควรจำกัดให้มีค่าต่ำสุด อาจทำได้โดยการใช้ออปแอมป์ที่มีอินพุทเป็น FET

4) แรงดันออฟเซ็ท (OFFSET) ที่เอาท์พุท

แรงดันออฟเซ็ทที่เอาท์พุทเกิดขึ้นจากกระแสไบอัสด้านอินพุทซึ่งในทางอุดมคติ เมื่อแรงดันอินพุท ระหว่างขั้วทั้งสองมีค่าเท่ากัน แรงดันที่เอาท์พุทควรเป็นศูนย์ แต่โดยทั่วไปมักไม่เป็นเช่นนั้น คือ มักมีแรงดันค่าหนึ่งปรากฏที่เอาท์พุทขณะที่อินพุทเป็นศูนย์ ซึ่งสามารถแก้ไขได้โดย การป้อนแรงดันหรือกระแสออฟเซ็ทที่อินพุท แล้วปรับจนได้ $V_{out} = 0$

5) กระแสออฟเซ็ทที่อินพุท

ในการปรับแรงดันออฟเซ็ทที่เอาท์พุทให้มีค่าเป็นศูนย์ กระแสอินพุททั้งสองขั้วควรมีค่าเท่ากัน แต่ในทางปฏิบัติ จะพบว่าต้องจ่ายกระแสให้แก่อินพุทขั้วหนึ่งมากกว่าอีกขั้วหนึ่งเสมอ เพื่อทำให้แรงดันเอาท์พุทมีค่าเป็นศูนย์ ซึ่งกระแสออฟเซ็ทนี้อาจมีค่าประมาณ 20 มิลลิแอมป์

6) แรงดันออฟเซ็ทที่อินพุท

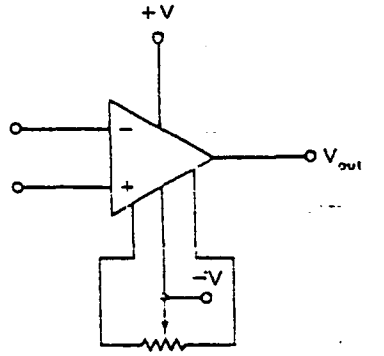
ในอุดมคติแรงดันเอาท์พุทจะเท่ากับศูนย์ก็ต่อเมื่อแรงดันระหว่างขั้วอินพุทมีค่าเป็นศูนย์เช่นกัน แต่ในทางปฏิบัติ ความไม่สมดุลภายในวงจรมักทำให้เราต้องป้อนแรงดันค่าหนึ่งแก่อินพุทใดๆเสมอ เพื่อทำให้แรงดันเอาท์พุทเป็นศูนย์

7) การปรับออฟเซ็ทให้เป็นศูนย์ (OFFSET NULLING)

ในการปรับแรงดันเอาท์พุทให้เป็นศูนย์หรือการปรับแรงดันออฟเซ็ทที่อินพุทให้เป็นศูนย์นั้นอาจใช้ขั้ว OFFSET NULLING ที่ผู้ผลิตได้เตรียมไว้ให้แล้ว โดยให้ศึกษาข้อมูลรายละเอียดของออปแอมป์นั้นๆ รูป 3.1.6 แสดงวงจรที่ใช้ในการตั้งค่าศูนย์ให้แก่ออปแอมป์ โดยมีขั้นตอนดังนี้

1. ตรวจสอบวงจรให้เรียบร้อยและตรวจอุปกรณ์ที่ต้องนำมาต่อ

2. ลดระดับสัญญาณอินพุทจนเหลือศูนย์ ถ้ามีตัวต้านทานต่ออนุกรมกับอินพุทอยู่ ให้ตรวจสอบ ดังนี้



รูปที่ 3.1.6 วงจรที่ใช้การตั้งค่าศูนย์ให้แก่ออปแอมป์

ก) ถ้าตัวต้านทานนั้นมีค่าสูงกว่าอิมพีแดนซ์ของแหล่งกำเนิดสัญญาณอินพุทตั้งแต่ 1 % ขึ้นไป ปล่อยตัวต้านทานไว้เช่นเดิม

ข) หากตัวต้านทานดังกล่าวมีค่าน้อยกว่าหรือเท่ากัน ให้นำแหล่งกำเนิดสัญญาณอินพุทนั้นออก แล้วต่อตัวต้านทานที่มีขนาดเท่ากับอินพุทอิมพีแดนซ์ของแหล่งกำเนิดเข้าแทน

ค) ต่อโหลดเข้ายังขั้วเอาท์พุท

ง) ป้อนไฟตรงให้วงจร

จ) วัดแรงดันที่เอาท์พุทด้วย โวลต์มิเตอร์ หรือออสซิลโลสโคป

ฉ) ปรับตัวต้านทานชนิดปรับค่าได้จน V_{out} มีค่าเป็นศูนย์

ช) ถอดอุปกรณ์ที่เพิ่มเข้าไปออก แล้วต่อวงจรดังเดิม แต่ห้ามปรับค่าตัวต้านทานปรับค่าได้

8) ผลของอุณหภูมิ

อุณหภูมิมีผลต่ออุปกรณ์โซลิดสเตททุกชนิด รวมทั้งออปแอมป์ ดังนั้น ผลกระทบจากอุณหภูมิจะทำให้กระแสและแรงดันออฟเซ็ทเปลี่ยนแปลงไป ซึ่งเรียกการเปลี่ยนแปลงอันเกิดขึ้นจากอุณหภูมิว่า ดริฟท์ ดังนั้นในขณะทำงานควรตรวจสอบด้วยว่าออปแอมป์จะมีเปอร์เซ็นต์การผิดพลาดมากน้อยเพียงใดหากอุณหภูมิเปลี่ยนไป

9) การชดเชยความถี่

ปัญหาที่เกิดขึ้นกับออปแอมป์ที่ใช้ในย่านความถี่สูงก็คือ การออสซิลเลท (Oscillation) ซึ่งเกิดจากอัตราขยายที่มีค่าค่อนข้างสูงของออปแอมป์เอง และยังเกิดจากการเลื่อนเฟส (Phase shift) ณ จุดต่างๆภายในวงจร เป็นผลให้เราไม่สามารถควบคุมอัตราขยายของสัญญาณป้อนกลับได้ วิธีแก้ปัญหาก็คือ ต่อตัวเก็บประจุชดเชยให้กับวงจร ซึ่งจะทำให้อัตราขยายของออปแอมป์มีขนาดลดลงเมื่อความถี่สูงขึ้น

10) อัตราสรูว์

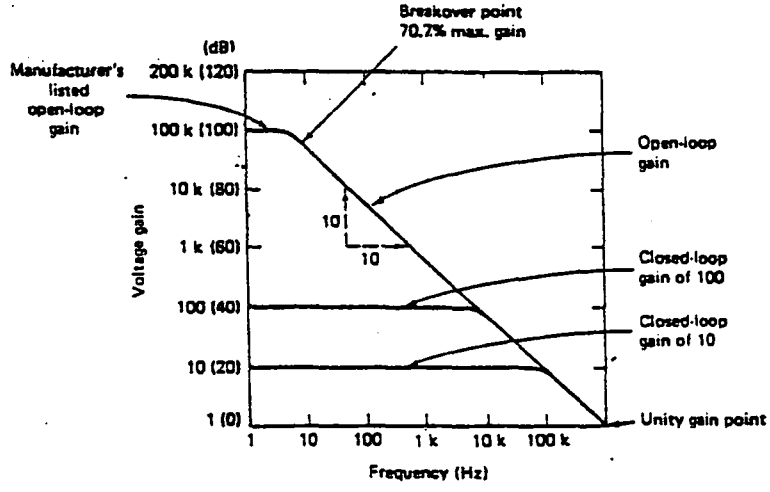
อัตราสรูว์ คือ อัตราการเปลี่ยนแปลงสูงสุดของแรงดันเอาท์พุทเทียบกับเวลา ดังสมการ

อัตราสรูว์ = การเปลี่ยนแปลงสูงสุดของแรงดันเอาท์พุท / การเปลี่ยนแปลงเวลา 3.1.2

11) การตอบสนองต่อความถี่

อัตราขยายของออปแอมป์จะลดลงเมื่อความถี่สูงขึ้น ดังแสดงในรูป 3.1.7 จะพบว่า อัตราขยายที่ผู้ผลิตแสดงไว้ในดาต้าชีท (data sheet) จะเป็นอัตราขยายที่ความถี่ 0 เฮิร์ตซ์ หรือไฟตรง ในการทำงานแบบลูปเปิดจะเห็นได้ว่า วงจรมีเสถียรภาพต่ำมาก นั่นคือ เมื่อความถี่สูงขึ้น 10 เท่า อัตราขยายจะลดลง 10 เท่าทันที

โดยทั่วไป แบนด์วิทของวงจรจะหมายถึงช่วงความถี่ที่อัตราขยายมีค่า 70.7 % ของอัตราขยายสูงสุด จากตัวอย่างในรูป แบนด์วิทของออปแอมป์ขณะเปิดลูป จะมีค่าประมาณ 10 เฮิร์ตซ์ ดังนั้น หากต้องการให้วงจรที่ออกแบบขึ้นสามารถทำงานได้ที่ความถี่อินพุทสูงๆ จะต้องป้อนสัญญาณจากเอาท์พุทกลับมายังอินพุท (การป้อนกลับแบบลบ) ซึ่งจะทำให้อัตราขยายของลูปปิดลดลง แต่ในขณะเดียวกัน ความถี่ที่อัตราขยายเท่ากับ 70.7 % ของอัตราขยายสูงสุดก็จะเพิ่มขึ้นด้วยอัตราเท่ากัน ส่วนจุดที่มีอัตราขยายเท่ากับหนึ่ง (unity-gain point) จะแสดงความถี่สูงสุดของออปแอมป์เมื่ออัตราขยายมีค่าเป็นหนึ่ง



รูปที่ 3.1.7 การตอบสนองความถี่ของออปแอมป์

12) ผลคูณของอัตราการขยายและแบนด์วิท

จากที่ได้อธิบายมาในหัวข้อที่แล้วว่า เมื่ออัตราขยายลดลงเท่าใด ความถี่ก็จะสูงขึ้นด้วยจำนวนเท่าของค่านั้นด้วย ซึ่งแสดงว่า ผลคูณระหว่างอัตราขยายและแบนด์วิทจะมีค่าคงที่เสมอไป และสามารถหาค่านั้นได้จากจุดซึ่งมีอัตราขยายเท่ากับหนึ่ง (ค่านี้จะหาได้จากดาตาชีท) ผลคูณนี้มีประโยชน์อย่างมากในการประมาณความถี่สูงสุดที่วงจรสามารถทำงานได้ตัวอย่างเช่น หากพบว่าความถี่ที่อัตราขยายมีค่าเท่ากับหนึ่ง (หาจาก DATA SHEET) มีค่าเป็น 1 เมกกะเฮิร์ตซ์ (ดังแสดงในรูป 3.1.7) และอัตราขยายของวงจรที่ออกแบบมีค่าเท่ากับ 100 ดังนั้น ความถี่สูงสุดที่ประมาณไว้ควรมีค่าอยู่ในราว

BW (แบนด์วิท) = 1,000,000 = 10 กิโลเฮิร์ตซ์ 3.1.3

13) อัตราการลดสัญญาณชนิดคอมมอนโหมด (CMRR)

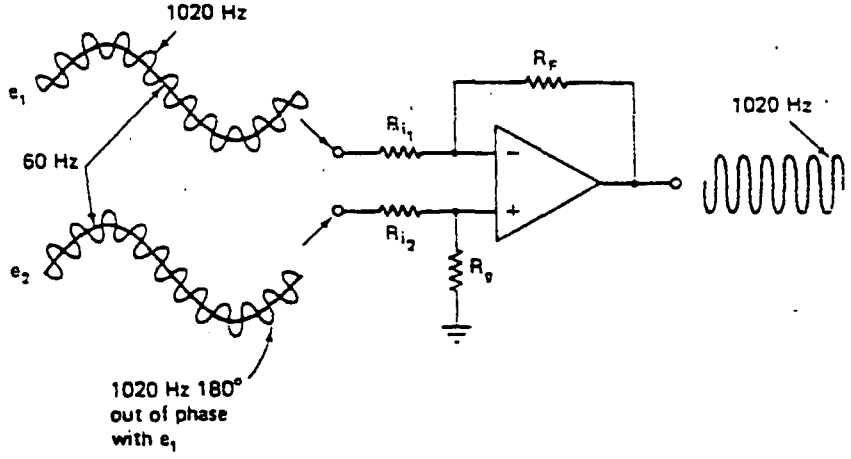
Commond Mode Rejection Ratio เป็นคุณสมบัติอย่างหนึ่งของออปแอมป์ที่ได้มาจากภาคอินพุทที่เป็นดิฟเฟอเรนเชียลแอมป์ หมายความว่า หากสัญญาณที่เข้ามายังขั้วอินพุททั้งสองนี้จะถูกกำจัดทิ้งไป ดังนั้น เอาท์พุทจะเป็นศูนย์ ส่วนสัญญาณที่มีขนาดต่างกันและมีเฟสต่างกันจะเรียกว่าอยู่ในดิฟเฟอเรนเชียลโหมด

ตัวอย่างเช่น วงจรในรูป 3.1.8 สัญญาณในดิฟเฟอเรนเชียลโหมดสองตัวซึ่งมีความถี่ 1020 เฮิร์ตซ์ ถูกป้อนเข้าไปยังขั้วอินพุทของออปแอมป์ แต่ในขณะเดียวกัน สัญญาณทั้งสองจะเก็บสัญญาณรบกวน 60 เฮิร์ตซ์ขึ้นมาในระหว่างเดินทาง ทำให้เกิดรูปคลื่นดังในภาพ ทว่าเมื่อป้อนสัญญาณเข้าไปแล้ว สัญญาณรบกวนความถี่ 60 เฮิร์ตซ์จะถูกกำจัดออกไป เนื่องจากเป็นสัญญาณชนิดคอมมอนโหมด ซึ่งความสามารถในการกำจัดสัญญาณคอมมอนโหมดนี้ เราเรียกย่อว่า CMRR หาได้จากสมการ

$$CMRR = \frac{A_d}{A_{cm}} \qquad \text{..... 3.1.4}$$

โดย A_d คือ อัตราขยายสำหรับสัญญาณแบบดิฟเฟอเรนเชียล

A_{cm} คือ อัตราขยายสำหรับสัญญาณคอมมอนโหมด ดังนั้นยิ่ง CMRR ของออปแอมป์มีค่าสูงเท่าไร หมายความว่า สัญญาณรบกวนจะถูกกำจัดลงมากขึ้นเท่านั้น



รูปที่ 3.1.8 สัญญาณในดิฟเฟอเรนเชียลโหมด

14) หน่วยเดซิเบล (dB)

การคำนวณอัตราขยายของวงจรใด ๆ สามารถหาได้จากสมการ

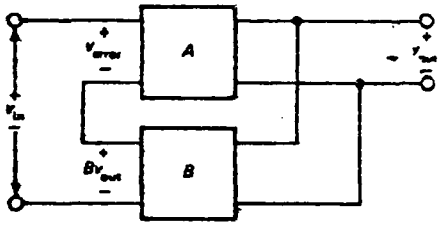
$$A_x = \frac{X_{OUT}}{X_{IN}} \qquad \text{..... 3.1.5}$$

โดยที่ x อาจเป็นแรงดัน หรือ กระแสก็ได้ และโดยทั่วไปนิยมหาอัตราขยายจากสมการ

3.1.4 การป้อนกลับแบบลบ

การป้อนกลับแบบลบหมายถึง สัญญาณบางส่วนของเอาท์พุทที่ถูกส่งไปยังอินพุทจะมีเฟสตรงข้ามกับสัญญาณอินพุท ประโยชน์ของการป้อนกลับแบบลบคือ ทำให้อัตราขยายคงที่ เปลี่ยนแปลงค่าอินพุท และเอาท์พุทอิมพีแดนซ์ให้ดีขึ้น ลดการผิดเพี้ยนไม่เป็นเชิงเส้น เพิ่มแบนด์วิทดังที่ได้กล่าวมาแล้วว่า ออปแอมป์จะไม่ใช้เป็น ลูปเปิด (open-loop) แต่จะใช้ลักษณะ ลูปปิด (close-loop) นั้นคือต้องมีการต่ออุปกรณ์ภายนอกเพิ่มเติม ซึ่งมักจะทำให้เกิดการป้อนกลับแบบลบ

รูปที่ 3.1.9 เป็นตัวอย่างของวงจรขยายป้อนกลับแบบลบ อิพพุทที่เข้าไปในวงจรขยาย (บล็อก A) เรียกว่า แรงดันผิดพลาด (ERROR VOLTAGE) ซึ่งเป็นผลต่างระหว่างสัญญาณอินพุท V_{in} กับสัญญาณป้อนกลับ BV_{out} ในระบบป้อนกลับแบบลบ แรงดันผิดพลาดจะมีค่าใกล้ศูนย์ เนื่องจากอัตราขยาย A มีค่าสูงประมาณ 10,000 ถึงมากกว่า 1,000,000 และแรงดันเอาท์พุทมักจะน้อยกว่า 10 V ดังนั้นแรงดันผิดพลาดบางทีมีค่าเป็นไมโครโวลท์ที่ความถี่ต่ำ เมื่อเทียบกับแรงดันอื่นๆ ทั้งหมดในวงจรแล้ว V_{error} จึงน้อยมาก



รูปที่3.1.9 การป้อนกลับแบบลบ

สมมติว่าอัตราขยายภายในวงจรเป็น A เพิ่มขึ้นเนื่องจากอุณหภูมิเปลี่ยนแปลงหรือเหตุผลอื่นๆ แรงดันเอาท์พุทก็จะเพิ่มตาม นั่นคือ แรงดันที่ถูกป้อนกลับไปยังอินพุทก็จะมีเพิ่มขึ้นด้วย บล็อก B จะเป็นตัวแบ่งแรงดันที่มีอัตราขยายเท่า B (ระหว่าง 0 ถึง 1) ดังนั้นสัญญาณป้อนกลับ Bv_{out} เป็นส่วนหนึ่งของแรงดันเอาท์พุท สัญญาณที่ป้อนกลับไปอินพุท จะถูกกลับเฟสให้ตรงข้ามกับสัญญาณอินพุท V_{in} จึงทำให้ V_{error} ลดลง ซึ่งจะไปขัดเขยกับการเพิ่มขึ้นของอัตราขยาย A พอดี ผลก็คือ V_{out} แทบจะไม่เพิ่มขึ้นเลย

ในกรณีอัตราขยาย A ลดลงก็เช่นเดียวกันคือ เมื่อ A ลดลง V_{out} จะลดลง BV_{out} ก็จะลดลงด้วยจึงทำให้ V_{error} เพิ่มขึ้น ซึ่งจะไปชดเชยกับการลดลงของอัตราขยาย A พอดี ผลก็คือ V_{out} ไม่ลดลงเลย หรืออาจจะลดลงก็เพียงเล็กน้อย

การพยายามที่จะเปลี่ยนแปลงแรงดันเอาท์พุทที่ป้อนกลับให้แก่อินพุท เป็นสาเหตุให้ต้องทำแรงดันผิดพลาดเปลี่ยนแปลงในทิศทางตรงกันข้าม คือ การป้อนกลับแบบลบ ผลที่เกิดขึ้นคือแรงดันเอาท์พุทจะเสมือนไม่ได้ขึ้นอยู่กับการเปลี่ยนแปลงภายในที่ทำให้อัตราขยาย A เปลี่ยนไป

สังเกตได้ว่า สัญญาณเอาท์พุทจะมีเฟสตรงกับสัญญาณอินพุท นั่นคือ ระบบเป็นวงจรขยายแบบไม่กลับเฟส (Noninverting Amplifier) ที่มีการป้อนกลับแบบลบ

ในรูปที่ 3.19 แรงดันผิดพลาดคือ ผลต่างของแรงดันอินพุทและแรงดันที่ป้อนกลับ

$$V_{error} = V_{in} - Bv_{out} \qquad \text{..... 3.1.6}$$

แรงดันเอาท์พุทเท่ากับแรงดัน. สมมดคูณกับอัตราขยายภายในวงจร

$$\begin{aligned} V_{out} &= AV_{error} \\ V_{out} &= A(V_{in} - Bv_{out}) \end{aligned} \qquad \text{..... 3.1.7}$$

เมื่อจัดรูปสมการใหม่จะได้

$$\frac{V_{out}}{V_{in}} = \frac{A}{1+AB} \qquad \text{..... 3.1.8}$$

สมการ 3.1.8 คือ อัตราขยายแรงดันของระบบทั้งหมด เนื่องจากเป็นอัตราส่วนของ V_{out} ต่อ V_{in}

ถ้าหากผลคูณของ AB มากกว่า 1 มากๆ อันเนื่องมาจาก อัตราขยาย A สูงมาก สมการ 3.1.8 จะลดลงเหลือ

$$\frac{V_{out}}{V_{in}} = \frac{1}{B} \qquad \text{..... 3.1.9}$$

สมการ 3.1.9 สำคัญมากเนื่องจากดูรูปสมการแล้วจะพบว่า อัตราขยายแรงดันของทั้งวงจรจะไม่ขึ้นกับอัตราขยายภายใน (A) เลย กลับขึ้นอยู่กับค่า B เท่านั้น และดังที่ได้กล่าวไปแล้วว่า วงจรย้อนกลับนั้นมักจะเป็นการแบ่งแรงดันจากเอาท์พุท ซึ่งตัววงจรสามารถใช้ค่าตัวต้านทานค่าหนึ่งที่ถูกต้องและแน่นอน อัตตราขยายแรงดันของวงจรขยายป้อนกลับจึงมีค่าเท่ากับส่วนกลับของ B

อัตราขยายภายใน (A) เรียกว่า อัตราขยาย ลูป-ปิด เพราะว่ามันคืออัตราขยายเมื่อเปิดลูปทางที่จะป้อนกลับ ในทางกลับกันอัตราขยายทั้งหมดที่มีการป้อนกลับด้วย เรียกว่า อัตราขยาย ลูปปิด เพราะว่ามันคือ อัตราขยายเมื่อปิดลูป หรือมีทางของสัญญาณทุกทางในวงจร ไม่ตัดวงจรส่วนใดออก ดังนั้นอาจจะพบสมการ 3.1.8 และ 3.1.9 ในรูปดังนี้บ่อยๆ

$$A_{CL} = \frac{A_{OL}}{1 + A_{OL}B}$$

$$A_{CL} \cong \frac{1}{B} \qquad \text{..... 3.1.10}$$

A_{CL} : อัตราขยายลูปปิด

A_{OL} : อัตราขยายลูปเปิด

ด้วยวิธีการคล้ายๆกับอัตราขยายแรงดัน สามารถพิสูจน์ได้ว่า อินพุทอิมพีแดนซ์ของวงจรขยายไม่กลับเฟสที่มีการป้อนกลับแบบลบมีค่าเป็น

$$Z_{in(CL)} = Z_{in(CL)} = (1 + A_{OL}B)Z_{in(OL)} \qquad \text{..... 3.1.11}$$

จะเห็นได้ว่า การป้อนกลับแบบลบจะเพิ่มอินพุทอิมพิแดนซ์

ตัวอย่างเช่น ถ้าวงจรขยายภายในมีอินพุทอิมพีแดนซ์ 1 โอห์ม และ 1 $A_{OL}B$ มีค่าเท่ากับ 10,000 เมื่อนั้นอินพุทอิมพีแดนซ์ของทั้งระบบ รวมการป้อนกลับด้วย มีค่าเป็น

$$Z_{in(CL)} = 10{,}000 \times 1K\Omega = 10M\Omega \qquad \text{..... 3.1.12}$$

และสามารถหาเอาท์พุทอิมพีแดนซ์

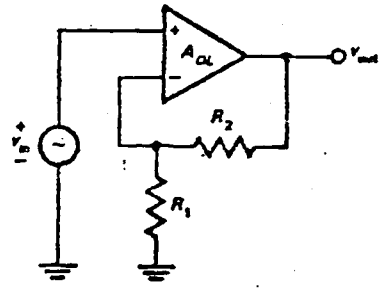
$$Z_{out(CL)} = \frac{Z_{out(OL)}}{1 + A_{OL}B} \qquad \text{..... 3.1.13}$$

การที่มีการป้อนกลับแบบลบ จะลดเอาท์พุทอิมพิแดนซ์ด้วยแฟคเตอร์ $1 + A_{OL}B$ อาจกล่าวได้ว่า เอาท์พุทอิมพีแดนซ์ของวงจรจะลดลง ตัวอย่างเช่น วงจรภายในมีเอาท์พุทอิมพีแดนซ์ $1K\Omega$ และ $1 + A_{OL}B = 10{,}000$

$$Z_{out(CL)} = \frac{1K\Omega}{10{,}000} = 0.1\Omega$$

3.1.5 <u>วงจรขยายไม่กลับเฟส</u>

รูปที่ 3.1.10 แสดงการต่อออปแอมป์เป็นวงจรขยายที่มีการป้อนกลับแบบลบ สัญญาณอินพุทจะเข้าที่อินพุทของออปแอมป์



รูปที่ 3.1.10 วงจรขยายไม่กลับเฟส

ออปแอมป์มีอัตราขยายภายใน A_{OL} และมีตัวต้านทานภายนอก R_1 และ R_2 เพื่อเป็นตัวแบ่งแรงดันให้ป้อนกลับ เนื่องจากแรงดันที่ป้อนให้ออปแอมป์อินพุทลบ ซึ่งตรงข้ามกับสัญญาณอินพุท หรือเรียกว่า เกิดการป้อนกลับแบบลบ

อัตราขยายแรงดัน หาได้จากค่า B ดังนี้

$$B = \frac{R_1}{R_1 + R_2}$$

ดังนั้นอัตราขยายลูปปิดประมาณ

$$A_{CL} \cong \frac{1}{B} = \frac{R_1 + R_2}{R_1}$$

$$A_{CL} \cong \frac{R_2}{R_1} + 1 \qquad \text{..... 3.1.14}$$

จะเห็นว่าอัตราขยายลูปปิด ขึ้นอยู่กับอัตราส่วนของตัวต้านทานที่ใช้ในการป้อนกลับ ดังที่ได้กล่าวมาแล้ว สามารถใช้ตัวต้านทานด้วยการกำหนดค่าได้ถูกต้อง จึงทำให้อัตราขยายลูปปิดที่ถูกต้องด้วย แม้ว่าอุณหภูมิหรือปัจจัยอื่นๆ จะเปลี่ยนแปลง อัตราขยายลูปปิดจะคงที่

ออปแอมป์ในรูป 3.1.10 จะไม่มีความถี่คัทออฟด้านล่าง (lower cutoff frequency) เพราะว่าต่อแบบคัปปลิงโดยตรง แต่จะมีความถี่คัทออฟด้านบนลูปเปิด (open-loop upper cutoff frequency : f_{OL}) ความถี่คัทออฟด้านบนของวงจรขยาย จะมากกว่า f_{OL} เนื่องจากมีการป้อนกลับแบบลบ ฉะนั้นเมื่อความถี่อินพุทสูงขึ้นจนขึ้นความถี่คัทออฟของวงจรภายใน f_{OL} ก่อน ซึ่งจะทำให้อัตราลูปเปิดลดลงเป็น 0.707 เท่าของอัตราขยายสูงสุด แต่ถ้ามีการป้อนกลับแบบลบ อัตราขยายลูปปิดจะไม่ลดลงเลย

ขณะที่ความถี่อินพุทเพิ่มขึ้นเรื่อยๆ อัตราขยายลูปเปิดจะลดลงเรื่อยๆ จนเท่ากับอัตราขยายลูปปิด เมื่อนั้นอัตราขยายลูปปิดจะลดลงอย่างรวดเร็ว ความถี่ขณะที่อัตราขยายลูปปิดลดลงเป็น 0.707 เท่าของอัตราขยายสูงสุดเรียกว่า ความถี่คัทออฟลูปปิด (closed-loop cutoff frequency : f_{CL})

$$f_{CL} = (1 + A_{OL}B) f_{OL} \qquad \text{..... 3.1.15}$$

ถ้า f_{OL} = 10 Hz และ $1+A_{OL}B$ = 10,000 f_{CL} = 10,000*10 Hz = 100 KHz

จากสมการ 3.1.10 เขียนใหม่ได้

$$1 + A_{OL}B = \frac{A_{OL}}{A_{CL}}$$

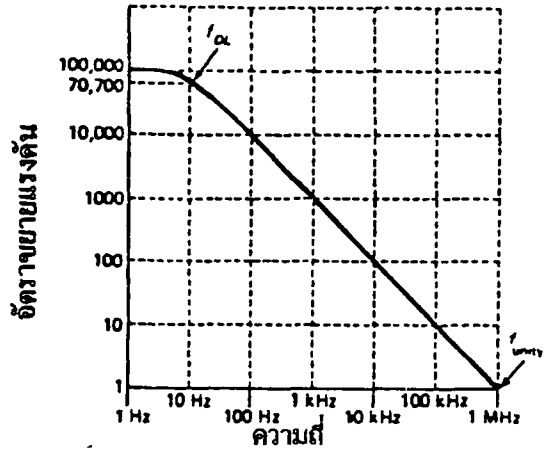
ฉะนั้น

$$f_{CL} = \frac{A_{OL}}{A_{CL}} f_{OL} \qquad \text{..... 3.1.16}$$

เนื่องจากการออกแบบต้องทราบค่า A_{OL} และ A_{CL} อยู่แล้วจึงสะดวกกว่าการใช้สมการ 3.1.16

รูปที่ 3.1.11 แสดงการตอบสนองของลูปเปิดของ 741C อัตราขยายลูปเปิดสูงสุดที่ 1,000,000 แต่เมื่อความถี่เพิ่มเป็น 10 Hz อัตรหาขยายลูปเปิดจะลดลงเป็น 0.707 เท่าของค่าสูงสุด เมื่อความถี่สูงขึ้น อัตราขยายจะลดลง

ความถี่ที่อัตราขยายเท่ากับหนึ่ง คือความถี่เพื่อขยายอัตราลูปเปิดเท่ากับหนึ่ง ในรูปที่ 3.1.11 f_{UNITY} เท่ากับ 1



รูปที่ 3.1.11 การตอบสนองลูปเปิดของ 741

เนื่องจากออปแอมป์จะไม่มีความถี่คัทออฟด้านต่ำ แบนด์วิทจึงเท่ากับความถี่คัทออฟด้านบน หรืออีกนัยหนึ่งคือแบนด์วิทลูปเปิดมีค่าเท่ากับ f_{OL} และแบนด์วิทลูปเปิดมีค่าเท่ากับ f_{CL} ซึ่งแบนด์วิทขนาดนี้เป็นแบนด์วิทสำหรับสัญญาณขนาดเล็ก ซึ่งไม่มีการผิดเพี้ยนเนื่องจากสรูว์เรท

เพื่อที่จะหาความสัมพันธ์ระหว่างแบนด์วิทสัญญาณขนาดเล็ก กับแบนด์วิทสัญญาณขนาดใหญ่ เขียนสมการได้ดังนี้

$$V_p = \frac{S_R}{2\pi f_{max}}$$

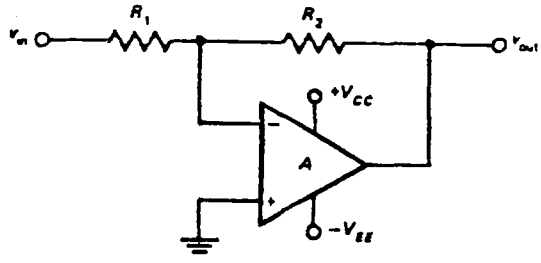
การทำงานที่สัญญาณขนาดเล็ก จะกลายเป็นสัญญาณขนาดใหญ่ที่จุดแบนด์วิทสัญญาณขนาดเล็ก f_{CL} เท่ากับแบนด์วิทสัญญาณขนาดใหญ่ f_{max} ดังนั้นเอาท์พุทสูงสุดจะเป็น

$$V_{p(max)} = \frac{S_R}{2\pi f_{CL}} \qquad \text{..... 3.1.17}$$

ตราบใดที่จุดยอดของเอาท์พุทมีค่าน้อยกว่า $V_{p(max)}$ ในสมการ 3.1.17 จะไม่เกิดความผิดพลาดเนื่องจากสรูว์เรท

3.1.6 วงจรขยายกลับเฟส

วงจรขยายกลับเฟสที่ใช้ออปแอมป์ สัญญาณอินพุทผ่าน R_1 เข้าออปแอมป์ที่อินพุทขาลบ ดังรูปที่ 3.1.12 สัญญาณเอาท์พุทจะมีขนาดใหญ่กว่าแรงดันผิดพลาด และทำให้สัญญาณเอาท์พุทมีเฟสต่างกับสัญญาณอินพุทอยู่ 180 องศา สัญญาณที่ป้อนกลับจะตรงข้ามกับสัญญาณอินพุท



รูปที่ 3.1.12 วงจรขยายกลับเฟส

อัตราขยายลูปเปิดของออปแอมป์มีค่าสูงมาก (741C มีค่าอัตราขยายลูปเปิด 100,000) ถ้าแรงดันเอาท์พุท 10 โวลท์ แรงดันผิดพลาดจะมีค่าเพียง 10/100,000 เท่ากับ 0.1mA นอกจากนี้อินพุทอิมพีแดนซ์ของ 741C มีค่าประมาณ 2 เมกกะโอห์ม ซึ่งจะให้กระแส 0.1/2 เท่ากับ 0.05 nA

เนื่องจากแรงดันผิดพลาดมีค่าน้อยมากเมื่อเทียบกับสัญญาณอื่น จึงประมาณว่าแรงดันผิดพลาด และกระแสผิดพลาดประมาณศูนย์

อินพุทกลับเฟสในรูปที่ 3.1.12 เป็นกราวด์เสมือน ซึ่งเสมือนกับว่าแรงดันผิดพลาดมีค่าเป็นศูนย์ ไม่ว่าแรงดันอินพุทจะเป็นเท่าไรก็ตาม อินพุทกลับเฟสไม่ใช่กราวด์ที่แท้จริง เพราะมีอินพุท อิมพีแดนซ์ 2 เมกกะโอห์ม ซึ่งมีกระแสน้อยมากไหลผ่านที่สามารถตัดทิ้งได้ กราวด์เสมือนใช้เปรียบเทียบจุดที่มีแรงดันเข้าใกล้ศูนย์

เนื่องจากอินพุทกลับเฟสเป็นกราวด์เสมือน จึงทำให้แรงดันอินพุททั้งหมดคร่อม R_1 สามารถหากระแสไหลผ่าน R_1 ได้ด้วยกฎของโอห์ม และกระแสจะไหลผ่าน R_2 ซึ่งจุดด้านซ้ายของ R_2 เป็นจุดเดียวกันกับอินพุทกลับเฟสซึ่งเป็นกราวด์เสมือน นั่นคือแรงดันตกคร่อม R_2 คือแรงดันเอาท์พุทนั่นเอง จึงเขียนได้ว่า

$$V_{out} = i_2 R_2 = i_1 R_1 = \frac{V_{in}}{R_1} R_2$$

จะได้ว่า

$$A_{CL} = \frac{R_2}{R_1} \qquad \text{..... 3.1.18}$$

และอินพุทอิมพีแดนซ์เมื่อมองจาก V_{in}

$$Z_{in(CL)} = R_1 \qquad \text{..... 3.1.19}$$

เหตุผลหนึ่งที่วงจรขยายกลับเฟสเป็นที่นิยมคือ มีอินพุทอิมพีแดนซ์และอัตราขยายแรงดันที่ถูกต้องและมีความคงที่ แม้ว่าอุณหภูมิหรือปัจจัยอื่นๆ จะเปลี่ยนแปลง ข้อดีอีกข้อหนึ่งของวงจรขยายกลับเฟสคือ สามารถมีอินพุทมากกว่า 1 อินพุทในเวลาเดียวกัน ดังในรูป 3.1.13

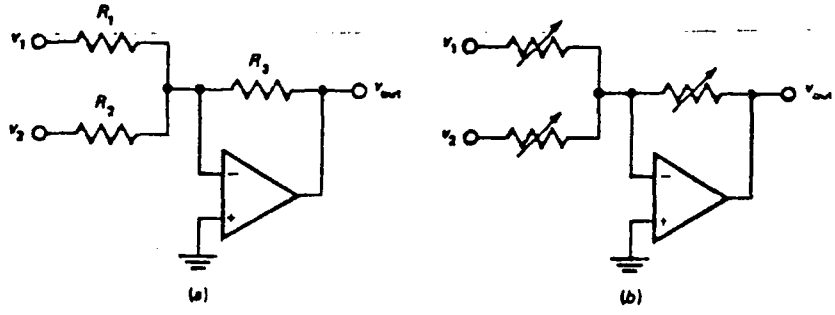
$$\begin{aligned} i_1 &= \frac{V_1}{R_1} \\ i_2 &= \frac{V_2}{R_2} \\ V_{out} &= (i_1 + i_2) R_3 = i_1 R_3 + i_2 R_3 \\ V_{out} &= \frac{R_3}{R_1} V_1 + \frac{R_3}{R_2} V_2 \end{aligned} \qquad \text{..... 3.1.20}$$

สำหรับค่าแบนด์วิทสัญญาณขนาดเล็ก สามารถเขียนเป็นสมการได้ดังนี้

$$f_{CL} = \frac{A_{OL}}{A_{CL}} f_{OL} \qquad \text{..... 3.1.21}$$

และเพื่อไม่ให้เกิดความผิดเพี้ยนเนื่องจากสรูว์เรท จุดยอดของอินพุทสูงสุดจะเป็น

$$V_{p(max)} = \frac{S_R}{2\pi f_{CL}} \qquad \text{..... 3.1.22}$$



รูปที่ 3.1.13 อินพุทของวงจรขยายกลับเฟส
(a) 2 อินพุท (b) ปรับค่าอินพุททั้งสอง

3.1.7 การประยุกต์ใช้งานออปแอมป์

ออปแอมป์ราคาถูกทำงานได้หลายอย่าง วิธีการใช้งานง่าย จึงทำให้ออปแอมป์ไม่เพียงแต่ใช้เป็นวงจรขยาย แต่ยังนิยมใช้เป็นวงจรจัดรูปคลื่น วงจรกรองความถี่

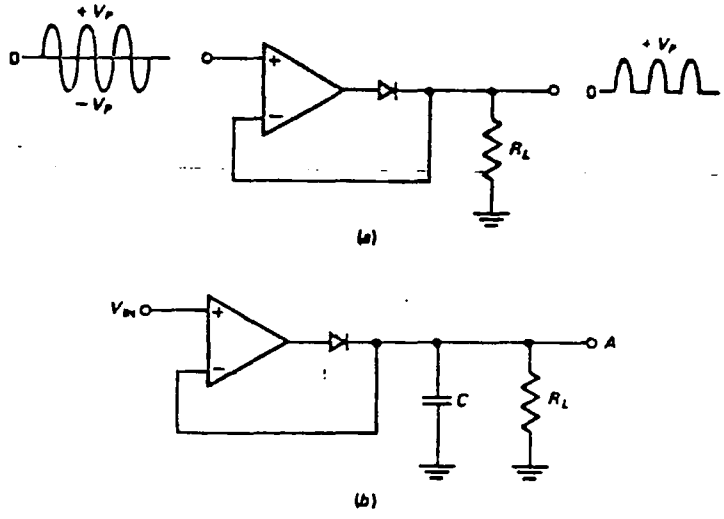
3.1.7.1 วงจรเรียงกระแสครึ่งคลื่น

ออปแอมป์สามารถเพิ่มบางสิ่งบางอย่างให้แก่ไดโอด เช่น การลดผลของแรงดันออฟเซทของไดโอด ทำให้สัญญาณขนาดเล็กๆ ที่น้อยกว่า 0.7 โวลท์ สามารถผ่านไดโอดได้ด้วยการเพิ่มวงจรออป-แอมป์เข้าไป

รูปที่ 3.1.14 (a) คือวงจรเรียงกระแสครึ่งคลื่นที่เพิ่มวงจรออปแอมป์ เมื่อสัญญาณอินพุทเป็นบวก เอาท์พุทจะเป็นบวก ไดโอดจะอยู่ในสภาพทำงาน วงจรขณะนี้ทำงานเหมือนวงจรโวลเทจฟอลโลเวอร์ จะมีแรงดันครึ่งบวกตกคร่อมตัวต้านทานโหลด R_L และเมื่อสัญญาณอินพุทเป็นลบ เอาท์พุทจะเป็นลบ ไดโอดจะไม่ทำงาน จะไม่มีแรงดันตกคร่อม นั่นคือเอาท์พุทจะเป็นสัญญาณครึ่งคลื่นที่สมบูรณ์

อัตราขยายที่มีค่ามากของออปแอมป์ ได้กำจัดผลอันเนื่องมาจากออฟเซทของไดโอด ตัวอย่างเช่น ถ้าแรงดันออฟเซท 0.7 โวลท์ และออปแอมป์มีอัตราขยายลูปเปิด 100,000 อินพุทที่สามารถทำให้ไดโอดมีสภาพทำงานมีค่า

$$V_{in} = \frac{0.7V}{100{,}000} = 7\mu V$$



รูปที่ 3.1.4 วงจรเรียงกระแสครึ่งคลื่น

เมื่ออินพุทมากกว่า 7 ไมโครโวลท์ ไดโอดจะทำงาน จึงสามารถลดออฟเซทได้ด้วยอัตราขยาย A

$$\phi' = \frac{\phi}{A} \qquad \text{..... 3.1.23}$$

เมื่อ ϕ' คือแรงดันออฟเซทโดยดูที่สัญญาณอินพุทย่อมแสดงว่า สามารถเรียงกระแสที่มีขนาดสัญญาณต่ำๆ ได้

3.1.7.2 วงจรกรองความถี่

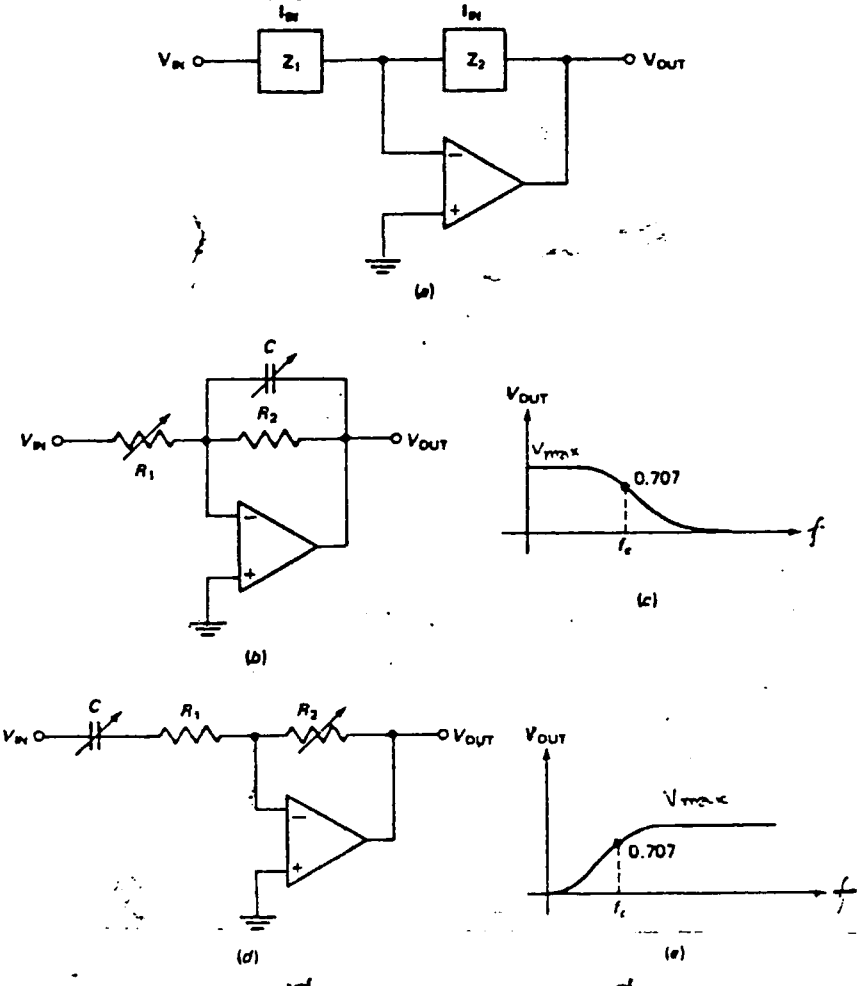
รูปที่ 3.1.25 (a) คือการใช้กราวด์เสมือนในวงจรกรองความถี่ ในวงจรแรงดันอินพุท และเอาท์พุท จะเป็นเฟสเซอร์คือ Z_1 และ Z_2 เป็นอิมพีแดนซ์เชิงซ้อน พิจารณากราวด์เสมือนจะได้

$$V_{in} = I_{in} Z_1$$
$$V_{out} = -I_{in} Z_2$$

เครื่องหมายลบหมายถึงเฟสตรงข้ามกัน

$$\frac{V_{out}}{V_{in}} = -\frac{Z_1}{Z_2} \qquad \text{..... 3.1.24}$$

สมการ 3.1.24 มีความหมายว่า อัตราขยายแรงดันเท่ากับอัตราส่วนระหว่างอิมพีแดนซ์ภายนอกทั้งสอง หากเลือกค่าอิมพีแดนซ์ที่เหมาะสมสามารถสร้างวงจรกรองได้ดังนี้



รูปที่ 3.1.15 วงจรกรองความถี่

รูปที่ 3.1.15 (b) เป็นวงจรกรรองความถี่ต่ำผ่าน ตัวเก็บประจุจะเสมือนเปิดวงจร วงจรจะทำหน้าที่เป็นวงจรขยายกลับเฟสที่มีอัตราขยายแรงดัน $-R_2/R_1$ เมื่อความถี่เพิ่มขึ้น รีแอคแตนซ์ของตัวเก็บประจุ X_C จะลดลง ทำให้อัตราขยายแรงดันลดลง

รูปที่ 3.1.15 (c) แสดงผลตอบสนองสัญญาณเอาท์พุทที่ความถี่ต่ำ เมื่อความถี่สูงขึ้นถึงความถี่คัทออฟ ขนาดสัญญาณเอาท์พุทจะลดลงเป็น 0.707 เท่าของค่าสูงสุดที่ความถี่ต่ำสุด สามารถหาความถี่คัทออฟได้จาก

$$f_c = \frac{1}{2\pi R_2 C} \qquad \text{..... 3.1.25}$$

ตัวเก็บประจุปรับค่าได้ในรูป 3.1.15 (b) ใช้ควบคุมความถี่คัทออฟ

ตัวต้านทานปรับค่าได้ในรูป 3.1.15 (b) ใช้ควบคุมอัตราขยายแรงดัน

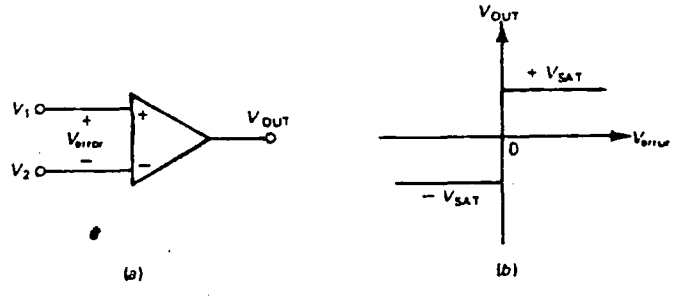
รูปที่ 3.1.15 (d) คือวงจรกรองความถี่สูงผ่าน ที่ความถี่ต่ำตัวเก็บประจุจะมีค่า X_C สูง เสมือนเปิดวงจร อัตราขยายแรงดันเท่ากับศูนย์ ที่ความถี่สูงตัวเก็บประจุจะมีค่า X_C ต่ำ เสมือนลัดวงจร ทำให้วงจรทำงานเป็นวงจรขยายกลับเฟส มีอัตราขยายแรงดัน $-R_2/R_1$ รูปที่ 3.1.15(e) เป็นผลตอบสนองสามารหาความถี่คัทออฟได้ดังนี้

$$f_c = \frac{1}{2\pi R_1 C} \qquad \text{..... 3.1.26}$$

3.1.7.3 ตัวเปรียบเทียบ

วิธีที่ง่ายที่สุดในการใช้ออปแอมป์คือ การใช้เป็นลูปเปิด ดังรูปที่ 3.1.16

เนื่องจากออปแอมป์มีอัตราขยายสูงมาก แรงดันผิดพลาดเพียงไมโครโวลท์ก็สามารถทำให้เกิดเอาท์พุทขนาดใหญ่ได้ เมื่อ V_1 มากกว่า V_2 แรงดันผิดพลาดจะเป็นบวก แรงดันเอาท์พุท จะเป็นค่าสูงสุดด้านบวก และถ้า V_1 น้อยยกว่า V_2 แรงดันผิดพลาดจะเป็นลบ แรงดันเอาท์พุท จะเป็นค่าสูงสุดทางด้านลบ



รูปที่ 3.1.16 ตัวเปรียบเทียบ

(a) วงจร (b) ผลตอบสนอง

3.2 การแปลงข้อมูล

แม้ว่าสัญญาณเสียงจะผ่านการปรับสภาพสัญญาณมาอย่างดีและเหมาะสมแล้วก็ตาม แต่การจะนำสัญญาณอนาลอกมาวิเคราะห์และประมวณผลทำได้ยากและไม่ดีนัก จึงต้องมีการแปลงสัญญาณอนาลอกเป็นสัญญาณดิจิตอลเสียก่อน ซึ่งทุกวันนี้เทคโนโลยีด้านดิจิตอลออดิโอทำให้เกิดการเปลี่ยนแปลงมากมายในวงการเครื่องเสียง การบันทึกเสียง การประมวลผลสัญญาณเสียง การเก็บข้อมูล รวมทั้งการเชื่อมโยงของระบบโทรศัพท์ เหล่านี้ล้วนต้องใช้เทคนิคทางดิจิตอลออดิโอ ซึ่งจะทำให้เสียงมีคุณภาพมากขึ้น

3.2.1 ขบวนการทางอนาลอกและดิจิตอล

ขบวนการทางอนาลอกเริ่มจากไมโครโฟนทำหน้าที่แปลงสัญญาณคลื่นซึ่งมีแอมปลิจูดเป็นสัญญาณทางไฟฟ้าซึ่งมีแรงดัน โดยมากจะมีแรงดันต่ำมากจึงต้องผ่าน*ภาคขยายสัญญาณ* และไม่ให้เกิดความผิดเพี้ยนจึงต้องผ่าน*ภาคกรองสัญญาณ* ทั้งสองภาคเรียกรวมกันว่า *การปรับสภาพสัญญาณ* ซึ่งได้อธิบายไว้แล้วในหัวข้อ 3.1 เมื่อผ่านขบวนการดังกล่าวแล้วมาบันทึกด้วยระบบอนาลอก จะเกิดความไม่เที่ยงตรง มีความผิดเพี้ยนของสัญญาณ เช่น เครื่องเล่นแผ่นเสียงในขณะที่เล่นกลับ ร่องแผ่นเสียงซึ่งแทนสัญญาณไฟฟ้าอาจมีฝุ่นละออง ทำให้สัญญาณทางไฟฟ้าผิดเพี้ยนไป หรือมีการเปลี่ยนรูปของสารที่ใช้เคลือบแผ่นเสียงทำให้เกิดเฮโซแนนซ์ ซึ่งจะทำให้หัวอ่านแผ่นเสียงแปลงสัญญาณกลับมาได้ไม่ถูกต้องขบวนการทางอนาลอกจึงเกิดความผิดเพี้ยนได้ง่าย

ขบวนการทางดิจิตอลจะแตกต่างออกไป แทนที่จะสนใจสัญญาณอนาลอกอย่างต่อเนื่องก็เลือกสนใจแอมปลิจูดแบบไม่ต่อเนื่องแต่เป็นช่วงๆ ไป โดยมีคาบเวลาที่คงที่ค่าหนึ่ง แอมปลิจูดที่วัดได้ในแต่ละช่วงจะถูกนำมาแทนรหัสด้วยเลขฐานสอง เพื่อจะนำไปบันทึกหรือประมวลผลต่อไป การที่คลื่นสัญญาณถูกแทนด้วยเลขฐานสองมีข้อดีมากมาย ที่เห็นได้ชัดก็คือ การสื่อสารข้อมูลด้วยเลขฐานสองมีความถูกต้องมากกว่า และไม่ง่ายต่อการถูกสัญญาณรบกวน ทำให้ส่งได้ไกลขึ้นด้วย

ประสิทธิภาพของสัญญาณดิจิตอลจะดีมากน้อยเพียงใด ขึ้นอยู่กับปัจจัย 2 ประการคือ *การสุ่มตัวอย่าง* (sampling) และ *การกำหนดขนาด* (quantisation)

3.2.2 ทฤษฎีการสุ่มตัวอย่าง

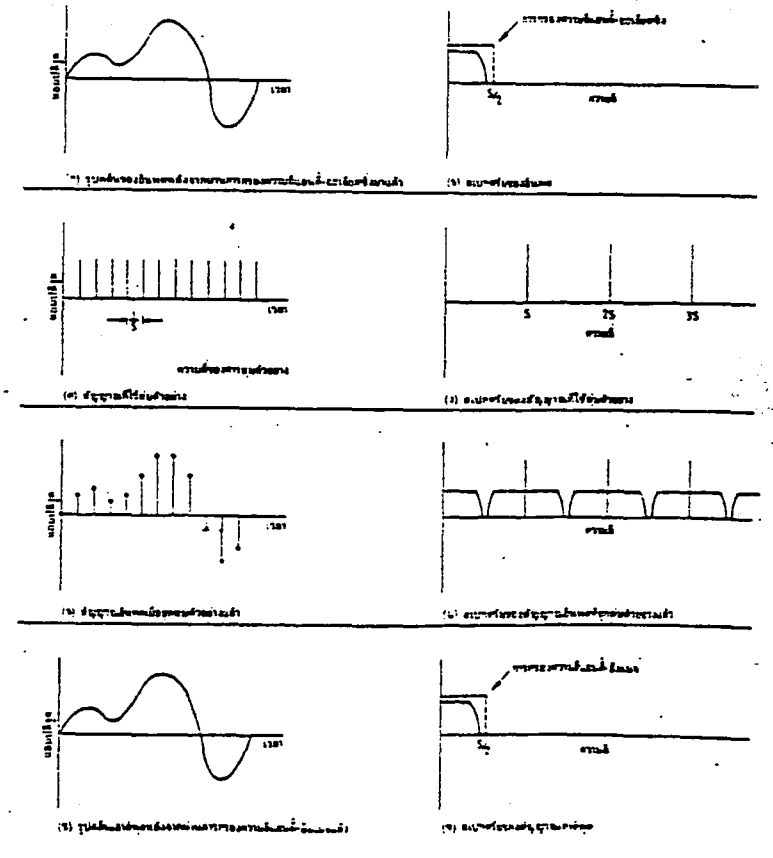
ทฤษฎีการสุ่มตัวอย่างได้ถูกกำหนดโดย นาย *ฮาร์รี่ ไนควิสต์* (Harry Nyquist) วิศวกรไฟฟ้าในปี ค.ศ. 1926 ว่า *ฟังก์ชันอนาลอกใดๆ สามารถนำมาสุ่มตัวอย่างค่าแอมปลิจูดในแต่ละช่วงเวลาที่มีคาบคงที่ ตัวอย่างที่ได้สามารถนำมาสร้างสัญญาณอนาลอกกลับมาได้ใหม่ โดยสุ่มตัวอย่างในอัตราที่เป็น 2 เท่า ของความถี่สูงสุดของสัญญาณอนาลอก*

ไนควิสต์กล่าวว่าภายใต้เงื่อนไขที่ถูกต้อง การสุ่มตัวอย่างจะไม่ทำให้ข้อมูลอนาลอกสูญหายไป ซึ่งเป็นหลักการพื้นฐานของระบบ***ดิจิไตซ์เซชัน*** (digitization)

แม้ว่าตัวอย่างที่ได้จะสร้างสัญญาณอนาลอกได้อย่างสมบูรณ์ก็จริง แต่ก็ต้องคำนึงถึงการตอบสนองความถี่ของระบบ ถ้าความถี่ของสัญญาณอนาลอกสูงกว่าความถี่ของการสุ่มสัญญาณอนาลอกที่กลับคืนมาจะไม่เกิดความสมบูรณ์ เป็นเหตุให้เกิดความเพี้ยนที่เรียกว่า ***อะเลียสซิง*** (aliasing) ขึ้นมาได้

การป้องกันการเกิดความเพี้ยนแบบอะเลียสซิง โดยการกำจัดความถี่ช่วงสัญญาณอินพุทโดยการผ่านวงจรกรองความถี่ต่ำผ่าน หรือเรียกว่า ***แอนตี้-อะเลียสซิง ฟิลเตอร์*** (anti-aliasing filter) ก่อนนำไปสุ่มตัวอย่าง เพื่อกำจัดความถี่สัญญาณอินพุตที่สูงกว่าครึ่งหนึ่งของความถี่สัญญาณสุ่มตัวอย่าง ซึ่งไม่เป็นไปตามทฤษฎีการสุ่มตัวอย่าง และไม่ต้องใช้ความถี่ของการสุ่มสูงเกินไป ลักษณะของวงจรต้องเป็นวงจรที่มีความชันมากๆ เช่น *วงจรกรองความถี่แบบบริควอลล์* (Brickwall filter)

ความถี่ของการสุ่มตัวอย่างเป็นตัวกำหนดช่วงการตอบสนองความถี่ของระบบดิจิไตซ์เซชัน เช่น ถ้าระบบมีการสุ่มตัวอย่าง S ครั้ง ใน 1 วินาที นั่นคือตัวอย่างที่สุ่มมาได้ เมื่อนำมาสร้างสัญญาณขึ้นมาใหม่ สามารถตอบสนองความถี่ที่ไม่เกินความถี่เท่ากับ S/2 เฮิร์ตซ์



รูปที่ 3.2.1 การสุ่มตัวอย่างในโดเมนเวลาและโดเมนความถี่

พิจารณาระบบดิจิไตซ์เซชันที่มีสัญญาณอินพุตรูปซายน์ 24 กิโลเฮิร์ตซ์ ความถี่ของการสุ่มตัวอย่างเป็น 48 กิโลเฮิร์ตซ์ ดังนั้นใน 1 คาบของสัญญาณอินพุท จะถูกสุ่มตัวอย่าง 2 ครั้ง ตัวอย่างที่ได้จะถูกนำไปสร้างเป็นสัญญาณขั้นบันได ซึ่งในกรณีนี้เป็นสัญญาณรูปสี่เหลี่ยมความถี่ 24 กิโลเฮิร์ตซ์ เพื่อให้ได้สัญญาณรูปซายน์ออกมาดังอินพุท ทางด้านเอาท์พุทของระบบดิจิไตซ์เซชันทุกระบบต้องมีส่วนของวงจรกรองความถี่ที่สูงกว่าครึ่งหนึ่งของความถี่การสุ่มตัวอย่าง แต่เหตุผลที่ต้องการมีแตกต่างกัน

เหตุที่ต้องกรองความถี่ทางด้านเอาท์พุท เพราะมีผลจากการสุ่มตัวอย่างมีการผลิตแถบความถี่เงาในย่านอุลตร้าโซนิก (ultrasoic image frequencies) ขึ้นมา ซึ่งอยู่เหนือช่วงความถี่ออดิโอ โดยมีค่าเป็นจำนวนเงาของความถี่ของการสุ่มตัวอย่าง แถบความถี่เงาที่เกิดขึ้นเป็นผลจากขบวนการมอดูเลตในขณะสุ่มตัวอย่าง

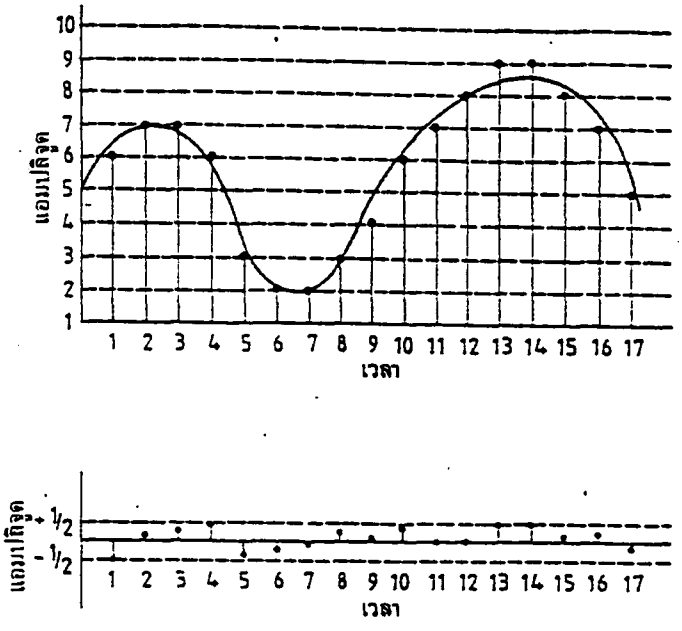
วงจรกรองความถี่ต่ำทางด้านเอาท์พุทเรียกว่า ***แอนตี้-อิมเมจ ฟิลเตอร์*** (anti-image filter) จะกำจัดแถบความถี่เงา ซึ่งเป็นฮาร์โมนิคอันดับต่างๆ ที่อยู่เหนือครึ่งหนึ่งของความถี่การสุ่มตัวอย่าง จากตัวอย่างที่ยกมาสัญญาณสี่เหลี่ยม 24 กิโลเฮิร์ตซ์ เมื่อผ่านวงจรกรองความถี่ต่ำ จะได้เอาท์พุทสัญญาณรูปซายน์ 4 กิโลเฮิร์ตซ์ ออกมา ข้อที่น่าสังเกตคือ ถ้าสัญญาณอินพุตไม่ได้มีรูปซายน์จะได้เอาท์พุทเป็นอย่างไร คำตอบก็คือ แม้ว่าสัญญาณอินพุทไม่ได้เป็นรูปซายน์ ที่สัญญาณที่ถูกสุ่มตัวอย่างเป็นส่วนประกอบของสัญญาณรูปซายน์เสมอ ไม่ว่าสัญญาณอินพุตจะเป็นรูปอะไร ก็ต้องผ่านการกรองความถี่ต่ำผ่านก่อน ส่วนประกอบความถี่สูงๆ ฮาร์โมนิกต่างๆ จะถูกกำจัดไปจนเหลือส่วนประกอบของสัญญาณรูปซายน์ที่จะถูกสุ่มตัวอย่างต่อไป

กล่าวย้ำอีกครั้งว่าความถี่ที่สูงกว่าครึ่งหนึ่งของความถี่การสุ่มตัวอย่างจะถูกกำจัดทิ้งไป ความถี่เหล่านี้หูคนไม่ได้ยินอยู่แล้ว การกำจัดทิ้งไปไม่ได้ทำให้คุณภาพของเสียงด้อยลงแต่อย่างใด

3.2.3 <u>การกำหนดขนาด</u>

ดังได้กล่าวมาแล้วว่า การสุ่มตัวอย่างเป็นวิธีที่เลือกวัดขนาดสัญญาณอนาลอกในแต่ละช่วงเวลา เมื่อการสุ่มตัวอย่างเกี่ยวข้องกับการกำหนดเวลาในการวัดขนาด ก็มีขั้นตอนต่อมาในการกำหนด ขนาดให้กับตัวอย่างที่สุ่มมาได้ด้วยเลขฐานสอง

สิ่งที่หลีกเลี่ยงไม่ได้ในการกำหนดขนาดคือ ความคาดเคลื่อนที่เกิดขึ้นเนื่องจากการจัดแบ่งย่านแอมปลิจูดออกเป็นระดับต่างๆ เพื่อใช้แทนค่าขนาดของสัญญาณที่สุ่มมาได้ ในการกำหนดขนาดตัวอย่างที่สุ่มมาได้ก็จะเลือกตัวเลขที่มีระดับใกล้เคียงที่สุดกับตัวอย่างนั้น จึงเป็นลักษณะของการประมาณค่าที่ใกล้เคียงกัน และนี่เป็นที่มาของความคลาดเคลื่อนอันเกิดจากขบวนการกำหนดขนาด ดังรูปที่ 3.2.2



รูปที่ 3.2.2 การเกิดความคลาดเคลื่อน
จากการกำหนดขนาดที่เวลาของการสุ่มตัวอย่าง

การแบ่งช่วงแอมปลิจูดออกเป็นระดับต่างๆนั้น จำนวนระดับมากน้อยอย่างไรขึ้นอยู่กับเลขฐานสองที่ใช้ ยิ่งมีจำนวนบิทมากก็ยิ่งแบ่งระดับออกได้มากขึ้น ตัวอย่างเช่น เลขฐานสองขนาด 2 บิท จะแบ่งได้เป็น 4 ระดับคือ 00, 01, 10 และ 11 ถ้าเป็นขนาด 16 บิท ก็แบ่งได้เป็น 65,536 ระดับ โดยจำนวนบิทเพิ่มขึ้น 1 บิท สามารถเพิ่มจำนวนระดับจากเดิมเป็น 2 เท่า

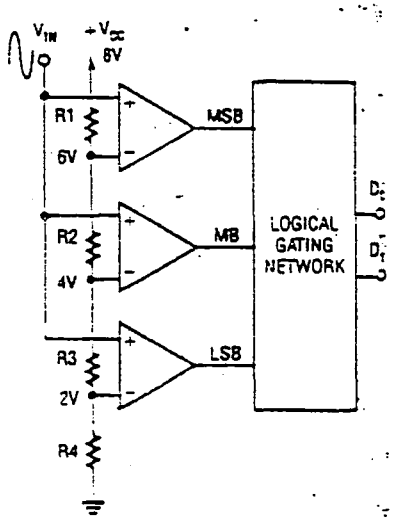
ประสิทธิภาพของขบวนการกำหนดขนาดเป็นตัวแปรสำคัญในการกำหนดคุณสมบัติของระบบ โดยมากแล้วขนาดของตัวอย่างที่สุ่มได้ มักจะไม่ค่อยตรงกับระดับต่างๆ ที่แบ่งไว้ในขบวนการกำหนดขนาด โดยเฉพาะเมื่อขนาดตัวอย่างอยู่ระหว่างกลางของระดับที่แบ่งไว้ ซึ่งเป็นจุดที่ทำให้เกิดความคลาดเคลื่อนมากที่สุด เช่น สมมุติว่าเลขฐานสอง 011000 แทนระดับสัญญาณ 1.20 โวลท์ และ 011001 แทนระดับสัญญาณ 1.30 โวลท์ แต่ขนาดตัวอย่างมีค่า 1.25 โวล์ท จะเห็นไม่มีตัวเลขที่แทนค่า 1.25 โวลท์ เพราะระหว่าง 011000 กับ 0111001 ไม่มีเลขฐานสองที่เป็นไปได้ จึงต้องเลือกเอาอย่างใดอย่างหนึ่งคือ 011000 หรือ 011001 ทำให้เกิดความคลาดเคลื่อนครึ่งหนึ่งของ 1 ระดับ แต่ถ้าจำนวนระดับมากขึ้น ก็อาจมีระดับที่สามารถแทนตัวเลขที่ต้องการได้

3.2.4 <u>การแปลงสัญญาณอนาลอกเป็นสัญญาณดิจิตอล</u> (Analog to Digital Converter)

การแปลงสัญญาณอนาลอกเป็นสัญญาณดิจิตอล หรือที่มักเรียกกันว่า ADC หรือ A/D ใช้สำหรับการแปลงสัญญาณอินพุทที่เป็นอนาลอก ให้เป็นดิจิตอล หรือเลขฐานสอง ผลลัพธ์ที่ได้จะอยู่ในรูปของเวิร์ด เทคนิคการแปลงสัญญาณของ A/D มีหลายแบบ แต่ละแบบก็มีข้อเสียแตกต่างกันไป จึงขอกล่าวรายละเอียดของ A/D แบบต่าง ดังต่อไปนี้

3.2.4.1 การแปลงแบบแฟลช (Flash Converter)

การแปลงสัญญาณอนาลอกเป็นสัญญาณดิจิตอลแบบแฟลช เป็นการแปลงที่มีความเร็ว เนื่องจากใช้ตัวเปรียบเทียบขนานกัน จึงอาจเรียกการแปลงแบบนี้ว่า*การแปลงแบบขนาน*ก็ได้



รูปที่ 3.2.3 วงจรการแปลงแบบแฟลช

วงจรในรูปที่ 3.2.3 มีตัวต้านทานที่ต่ออนุกรมกันอยู่ในวงจรแบ่งแรงดัน ที่ตกคร่อมตัวเปรียบเทียบแต่ละตัว แรงดันอินพุตสูงสุดจะขึ้นอยู่กับค่าของไฟเลี้ยง ที่ป้อนให้กับอุปกรณ์ A/D สัญญาณขาออกจากตัวเปรียบเทียบแต่ละตัวจะเป็น "1" หรือ "0" ซึ่งเป็นระดับสัญญาณลอจิกของวงจรดิจิตอล

เมื่อไม่มีแรงดันอินพุทเข้ามา ที่ขาออกของตัวเปรียบเทียบแต่ละตัวจะเป็นลอจิก "0" เมื่อเพิ่มแรงดันอินพุทเรื่อยๆ ที่เอาท์พุทของตัวเปรียบเทียบแต่ละตัวจะเป็นลอจิก "1" เมื่อแรงดันอินพุทมากกว่าแรงดันอ้างอิงแต่ละค่าที่ถูกตั้งไว้โดยวงจรแบ่งแรงดัน เนตเวอร์คของดิจิตอลเกท ถูกใช้ในการเรียงลำดับของสัญญาณจากตัวเปรียบเทียบ ให้อยู่ในรหัสเลขฐานสอง ซึ่งเป็นการสร้างรหัสที่เอาท์พุทของตัวแปลงสัญญาณ

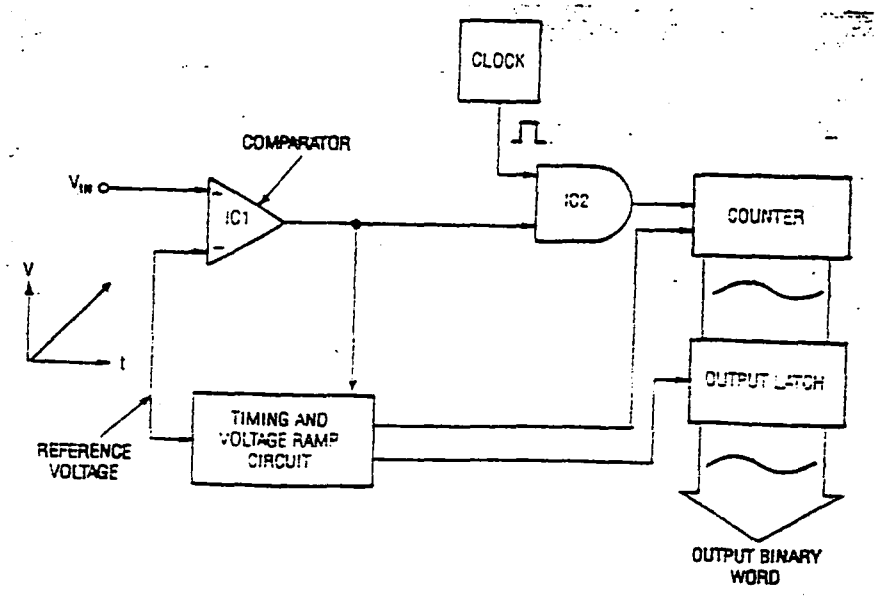
ตัวอย่างที่ยกมาให้ดูในรูปที่ 3.2.3 เป็น A/D ขนาด 2 บิท ซึ่งไม่สามารถนำไปใช้งานจริงได้ เนื่องจากความละเอียดต่ำเกินไปแต่เป็นแนวคิดที่สำคัญในการสร้าง A/D แบบแฟลชได้เป็นอย่างดี จะเห็นว่าใช้ตัวเปรียบเทียบ 2^n-1 ตัว ในการแสดงความละเอียด จากตัวอย่างเป็นตัวแปลงสัญญาณขนาด 2 บิท จึงใช้ตัวเปรียบเทียบ 2^2-1 = 3 ตัว ถ้าเป็นตัวแปลงสัญญาณขนาด 4 บิท จะใช้ตัวเปรียบเทียบ 2^4-1 = 15 ตัว เช่นเดียวกันถ้าเป็นตัวแปลงสัญญาณขนาด 8 บิท จะใช้ตัวเปรียบเทียบ 2^8-1 = 255 ตัว จะเห็นได้ว่าขนาดของตัวแปลงสัญญาณมากเท่าไร จำนวนตัวเปรียบเทียบจะมากขึ้นเป็นทวีคูณ และความละเอียดจะเพิ่มมากขึ้น แต่ความซับซ้อนในการสร้างก็สูงขึ้น และยังทำให้ตัวอุปกรณ์มีขนาดใหญ่ขึ้นด้วย เหล่านี้จึงเป็นข้อเสียของ A/D แบบแฟลช

ความเร็วของ A/D แบบแฟลช จะให้สัญญาณอนาลอกทางขาเข้าจ่ายให้แก่ตัวเปรียบเทียบแต่ละตัวพร้อมๆ กัน ช่วงเวลาในการเปลี่ยน (convertion time) จึงมีค่าเท่ากับเวลาหน่วงในตัวเปรียบเทียบแต่ละตัวแต่วงจรเกทในวงจรเท่านั้น ซึ่งใช้เวลาไม่กี่ไมโครวินาทีเท่านั้น

การนำ A/D แบบแฟลชไปใช้งานจึงควรเป็นงานที่ต้องการความเร็วในการแปลงสัญญาณสูงๆ แต่ไม่ต้องการความละเอียดมากนัก

3.2.4.2 การแปลงแบบความชันเดียว (Single Slope Convertion)

วิธีการแปลงสัญญาณอนาลอกเป็นสัญญาณดิจิตอลที่มีประสิทธิภาพสูงวิธีหนึ่งคือการแปลงแบบความชันเดี่ยว หรือเรียกว่าแบบแรมป์เดี่ยว (Single-ramp A/D) ดังแสดงไว้ในรูปที่ 3.2.4



รูปที่ 3.2.4 แผนผังการทำงานของการแปลงแบบความชันเดียว

จากแผนผังการทำงานของวงจรแบบความชันเดี่ยว เริ่มต้นที่การรีเซ็ตวงจรนับ (counter) และแรงดันแรมป์อยู่ที่ศูนย์ แรงดันเอาท์พุทของตัวเปรียบเทียบที่จุดเริ่มต้นเป็นศูนย์ ดังนั้นจึงไม่มีสัญญาณนาฬิกาจ่ายให้กับวงจรนับ เมื่อแรงดันอินพุทถูกจ่ายให้กับตัวแปลงสัญญาณอินพุทนอนอินเวอร์ติง (non-inverting) จะมีแรงดันเกินกว่าแรงดันที่อินพุทอินเวอร์ติง (inverting) ดังนั้นที่ขาออกของตัวเปรียบเทียบจึงเป็นลอจิก "1" สัญญาณลอจิก "1" จะได้เอนาเบิลให้แอนด์เกทท่างาน ยอมให้พัลส์ของสัญญาณนาฬิการผ่านไปยังวงจรนับเลขฐานสองให้ทำงานในขณะเดียวกันวงจรไทม์มิง (timing) จะขับให้วงจรแรมป์เพิ่มขึ้น ซึ่งทำให้แรงดันอ้างอิงที่ขาอินเวอร์ติงของตัวเปรียบเทียบเพิ่มขึ้นอย่างรวดเร็ว เมื่อแรงดันแรมป์อ้างอิงเริ่มมากขึ้นกว่าแรงดันอินพุท ที่แรงดันขาเอาท์พุทของตัวเปรียบเทียบจะตกลงมาเป็น "0" อีกครั้ง พัลส์ของสัญญาณนาฬิกาจึงหยุด เนื่องจากไม่สามารถผ่านวงจรแอนด์เกทได้และ วงจรไทม์มิงส่งสัญญาณไปยังวงจรนับเลขฐานสองเกิดการค้าง (latch) ค่าที่นับไว้ในขณะนั้น เมื่อทำการรีเซ็ตตัวนับอีกครั้งก็จะเกิดการทำงานเช่นเดิมต่อเนื่องกันไป

เมื่อแรงดันแรมป์อ้างอิงมีค่าเท่ากับแรงดันที่อินพุท วงจรนับจะถูกกระตุ้นให้นับเลขฐานสองในขณะเดียวกันค่าที่นับได้จึงเป็นสัญญาณดิจิตอลของสัญญาณอนาลอกทางด้านเอาท์พุท จะสังเกตได้ว่าความเร็วของสัญญาณนาฬิกา อัตราการเพิ่มขึ้นในลักษณะแรงดันแรมป์จะต้องมีความสัมพันธ์กันอย่างถูกต้อง เพื่อให้วงจรนับทำงานตามหน้าที่ได้อย่างถูกต้องนั่นเอง

ค่าเวลาที่ต้องการเปลี่ยนขึ้นอยู่กับระดับของสัญญาณอนาลอกทางอินพุท เพราะวงจรนับและแรงดันแรมป์อ้างอิงทั้งคู่เริ่มต้นจากศูนย์ที่ทุกๆ การรีเซ็ต จึงใช้เวลาค่อนข้างนานที่จะทำให้แรงดันอ้างอิงเท่ากับแรงดันอินพุท ในทางตรงกันข้ามถ้าแรงดันอินพุทมีค่าน้อย ช่วงเวลาที่แรงดันแรมป์อ้างอิงเพิ่มขึ้นจนเท่ากับแรงดันอินพุท จะใช้เวลาน้อยกว่ากรณีที่แรงดันอินพุทมีค่ามาก

แรงดันแรมป์อ้างอิงสามารถเปลี่ยนแปลงเพิ่มขึ้นจนเท่ากับแรงดันอินพุทได้เร็วกว่า 1 โวลท์ ต่อ 1/1000 วินาที เช่น ถ้าแรงดันอินพุทเป็น 2 โวลท์ ถูกจ่ายให้กับวงจรในรูปที่ 8 วงจรจะใช้เวลา 2*1 โวลท์/มิลลิวินาที ซึ่งเท่ากับ 2 มิลลิวินาที สำหรับแรงดันแรมป์ที่เพิ่มขึ้น จนมีระดับแรงดันเท่ากันระดับแรงดันอินพุท การนับเลขฐานสองจะการทำหลังจาก 2 มิลลิวินาทีไปแล้วความเร็วในช่วงนี้ขึ้นอยู่กับสัญญาณนาฬิกา ถ้าความเร็วของสัญญาณนาฬิกามาก จะทำให้จังหวะการนับเร็วขึ้น

เนื่องจากการทำงานของสัญญาณนาฬิกาขึ้นอยู่กับแรงดันแรมป์ จึงเป็นลักษณะพิเศษของการแปลงแบบความชันเดี่ยว ที่มีสัญญาณเอาท์พุทออกมาเป็นเลขฐานสองโดยตรง ไอซีเครื่องมือวัดบางตัวที่ใช้เทคนิคแบบความชันเดี่ยว จะแปลงรหัส BCD ไปขับภาคแสดงผล 7 เซกเมนท์ได้โดยตรง ซึ่งทำให้สะดวก และได้เปรียบกว่าการใช้เทคนิคแบบอื่น

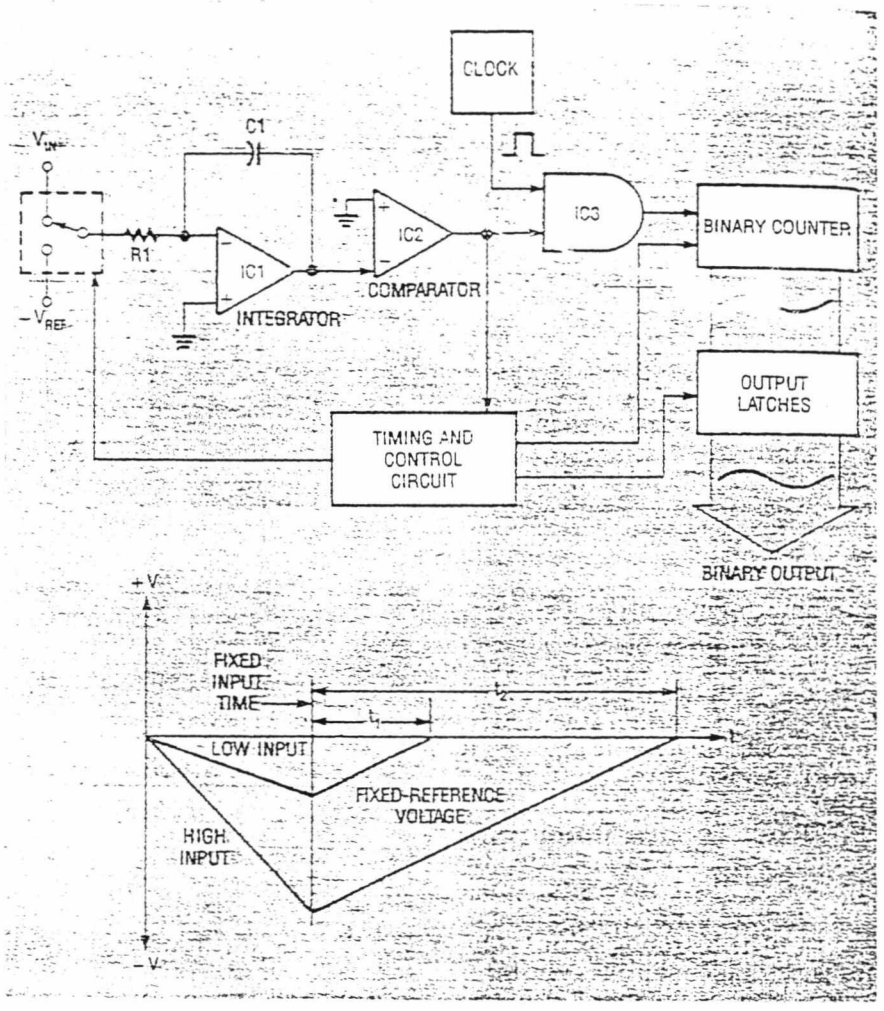
ข้อเสียของการแปลงแบบความชันเดี่ยว คือการทำงานที่ไม่เสถียรภาพเมื่อใช้งานไปนานๆ โดยไม่มีการประสานจังหวะ (synchronization) ระหว่างวงจรผลิตสัญญาณนาฬิกา และวงจรสร้างสัญญาณแรมป์ ทุกๆ การเลื่อนของความเร็วสัญญาณนาฬิกา หรือแรงดันแรมป์ เป็นเหตุให้เกิดความผิดพลาดขึ้นที่รหัสทางเอาท์พุท จึงเป็นสาเหตุสำคัญที่ทำให้ การแปลงแบบความชันเดี่ยวไม่นำไปใช้งานที่ต้องการความถูกต้องสูงๆ

3.2.4.3 การแปลงแบบความชันคู่ (Double Slope Convertion)

เทคนิคการแปลงสัญญาณอนาลอกเป็นสัญญาณดิจิตอลแบบความชันคู่ เป็นเทคนิคที่ให้ข้อดีในด้านเสถียรภาพของการแปลงสัญญาณ เมื่อสัญญาณอินพุทมีการเปลี่ยนแปลงอย่างรวดเร็ว วงจรผลิตสัญญาณแรมป์อ้างอิง ได้ปรับปรุงขึ้นโดยตัดเอาผลกระทบของการเลื่อนไหลเมื่อใช้วงจรไปนานๆ

สัญญาณอินพุทของตัวแปลงสัญญาณแบบความชันคู่ จะป้อนให้กับวงจรอินทิเกรตเตอร์ เมื่อสัญญาณอินพุทที่เป็นบวกถูกป้อนเข้ามาที่ตัวแปลงสัญญาณ ความชันของแรงดันแรมป์ทางเอาท์พุทของวงจรอินทิเกรตเตอร์จะมีทิศทางเป็นลบ และจะมีค่าเป็นลบ เพราะป้อนอินพุทเข้าทางอินเวอร์ติงของออปแอมป์ ด้วยแรงดันลบที่ได้นี้ทำให้เอาท์พุทของวงจรเปรียบเทียบเป็น "1" ด้วยเหตุนี้จึงเป็นการกระตุ้นให้เกิดสัญญาณนาฬิกาป้อนเข้าไปยังขั้วอินพุทของวงจรนับ ซึ่งจะเป็นการเริ่มต้นนับขึ้นไปเรื่อยๆ วงจรอินทิเกรตเตอร์จะให้สัญญาณแรมป์เพียงคาบเวลาที่คงที่ขณะหนึ่งเท่านั้น หลังจากช่วงเวลานี้แล้ววงจรควบคุมจะเคลียร์วงจรนับ และเปลี่ยนอินพุทของวงจรอินทิเกรตเตอร์ไปต่อกับแรงดันอ้างอิงที่มีค่าเป็นลบ ดังนั้นแรงดันลบในขณะนี้ถูกป้อนให้กับวงจรอินทิเกรตเตอร์ ความชันของสัญญาณแรมป์ทางเอาท์พุทกลับมามีทิศทางเป็นบวก วงจรนับจะเริ่มนับใหม่จนกระทั่งเอาท์พุทของวงจรอินทิเกรตเตอร์ ตกลงมาเป็นศูนย์ ที่จุดนี้เอาท์พุทของวงจรเปรียบเทียบกลายเป็น "0" ซึ่งทำให้สัญญาณนาฬิกาที่ป้อนให้วงจรนับหยุดลง วงจรควบคุมจะตรวจสอบซึ่งเปลี่ยนและแลทช์การนับที่เอาท์พุทไว้ แล้วเคลียร์วงจรนับอีกครั้ง การแปลงสัญญาณแบบความชันคู่นี้ สัญญาณที่นับได้ครั้งสุดท้ายจะแทนแรงดันอินพุทอนาลอกที่เข้ามา

อัตราการอินทิเกรตขึ้นอยู่กับขนาดของแรงดันอินพุท เช่นเดียวกับค่า R_1 และ C_1 ดังนั้นแรงดันที่ต่ำๆ จะลดเอาท์พุทของวงจรอินทิเกรตเตอร์ ให้น้อยกว่าแรงดันอินพุทที่มีค่าสูงๆ ในช่วงคาบเวลาอินพุทที่แน่นอนของคอนเวอร์ชันไซเคิล



รูปที่ 3.2.5 แผนผังและช่วงเวลาการทำงานของการแปลงแบบความชันคู่

เมื่อแรงดันลบอ้างอิงที่มีค่าคงที่ถูกป้อนเข้ามา (ค่าของ R_1 และ C_1 ยังคงเหมือนเดิม) เวลาที่ต้องการสำหรับเอาท์พุทของวงจรอินทิเกรตเตอร์เริ่มเข้าสู่ศูนย์ เป็นอัตราแปรผันโดยตรงต่อขนาดเดิมของแรงดันอินพุท ทุกๆการเปลี่ยนแปลง เพราะฉะนั้นในวงจรอินทิเกรตเตอร์ เวลาหรืออุณหภูมิมีผลต่อการทำงานของวงจรจะถูกตัดออกโดยอัตโนมัติ ดังนั้นการแปลงแบบความชันคู่จึงมีเสถียรภาพเหมาะสำหรับการประยุกต์ใช้งานที่มีความแม่นยำสูง

การแปลงแบบความชันคู่มีสิ่งที่คล้ายกับการแปลงแบบความชันเดี่ยว คือสัญญาณอินพุทสามารถถูกเปลี่ยนให้อยู่ในรูป BCD หรืออยู่ในรูปอื่นๆ ได้โดยตรง เช่น รหัสไบนารี่ ในดิจิตอลโวลท์มิเตอร์โดยมากจะใช้เทคนิคแบบความชันคู่

ข้อเสียของการแปลงแบบความชันคู่ คือคาบเวลาที่ขยายออกไปในการแปลงสัญญาณ ต้องการคาบเวลาที่มากกว่า 100 มิลลิวินาที ต่อการเปลี่ยนแปลงสัญญาณอินพุทที่มีแรงดันสูงๆ ให้อยู่ในรูปสัญญาณดิจิตอล

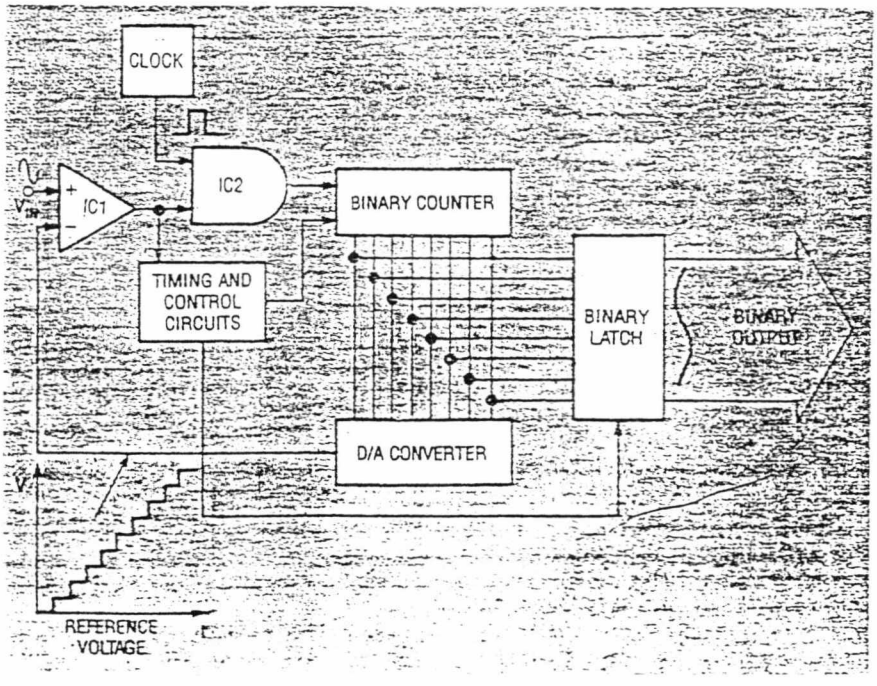
3.2.4.4 การแปลงแบบป้อนกลับ (Feedback Convertion)

การแปลงที่ใช้สัญญาณป้อนกลับมาเป็นสัญญาณอ้างอิงที่วงจรเปรียบเทียบมี 2 ชนิด ดังนี้

1) วงจรนับแบบเดี่ยว (Single counter)

วงจรของการแปลงสัญญาณแบบวงจรนับเดี่ยว ได้มีการพัฒนาจนมีลักษณะคล้ายการแปลงแบบความชันเดี่ยว ตลอดจนการทำงานของวงจรทั้งสองยังมีความคล้ายกับด้วย การแปลงแบบวงจรนับเดี่ยวจะอ่านการนับของสัญญาณนาฬิกาที่ได้จากวงจรนับเลขฐานสอง แล้วทำให้เป็นแรงดันป้อนกลับไปยังวงจรเปรียบเทียบแทนวงจรอินทิเกรตเตอร์ หรือแหล่งจ่ายแรงดันแรมป์อื่นๆ

เมื่อสัญญาณอนาลอกทางอินพุทถูกจ่ายให้วงจรเปรียบเทียบ เอาท์พุทจะมีสถานะเป็น "1" ดังนั้นวงจรควบคุมจึงยอมให้สัญญาณนาฬิกาผ่านเข้าไปวงจรนับ วงจรนับจะนับเลขฐานสองขึ้นไปเรื่อยๆ ซึ่งแรงดันเอาท์พุทของ D/A จะเพิ่มขึ้นตามไปด้วย เอาท์พุทของ D/A ที่ได้นี้จะถูกป้อนกลับไปยังอินพุทที่ขาลบของวงจรเปรียบเทียบ เมื่อระดับแรงดันเอาท์พุทของ D/A เริ่มมีค่ามากกว่าระดับแรงดันอินพุทที่เข้ามา เอาท์พุทของวงจรเปรียบเทียบจะมีสถานะเป็น "0" คือวงจรเปรียบเทียบหยุดทำงาน วงจรควบคุมก็จะส่งสัญญาณไปยังวงจรแลทช์เลขฐานสอง ให้ค้างค่าที่ได้จากเอาท์พุทไว้ หลังจากนั้นวงจรควบคุมจะรีเซ็ตวงจรนับ แล้วทำงานต่อเนื่องไปเรื่อยๆ

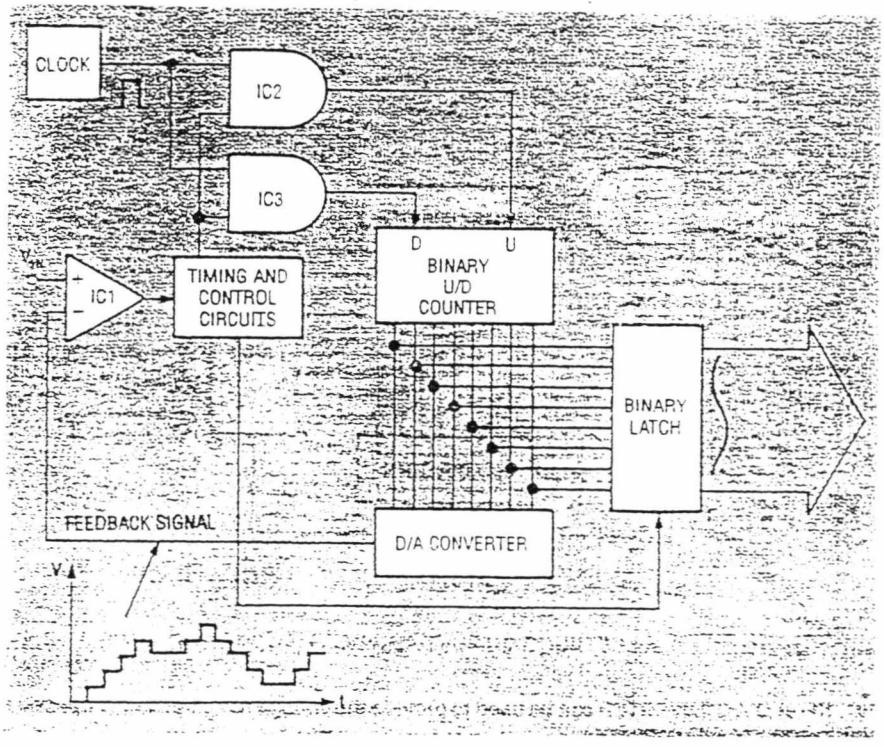


รูปที่ 3.2.6 การทำงานของ A/D แบบวงจรนับเดี่ยว

ถึงแม้ว่าการแปลงแบบวงจรนับเดี่ยวเป็นวิธีที่เร็วกว่าวิธีความชันคู่ แต่ความแม่นยำของวงจรก็ขึ้นอยู่กับ D/A ที่ใช้ในวงจร ถ้าต้องการให้ A/D แบบวงจรนับเดียวมีความแม่นยำสูงด้วย เพื่อให้ได้สัญญาณป้อนกลับที่เอาท์พุทของ D/A มีความแม่นยำ ป้อนกลับไปยังวงจรเปรียบเทียบ วงจร A/D แบบวงจรนับเดี่ยวยังต้องเริ่มจากศูนย์ ทุกๆการรีเซ็ต ดังนั้นจึงมีความเป็นไปได้ที่จะมีการสูญหายของข้อมูลเป็นบิทของเวลาไปทุกๆ รอบ โดยเฉพาะอย่างยิ่งถ้าแรงดันที่ถูกแปลงนั้นมีค่าใกล้เคียงกับระดับแรงดันสูงสุดของ A/D

2) วงจรนับแบบแทร็คกิง (Tracking counter)

เทคนิคแบบวงจรนับแทร็คกิงสามารถแปลงได้เร็วกว่าวงจรนับเดี่ยว เนื่องจากวงจรแบบแทร็คกิงใช้วงจรนับเลขฐานสองแบบขึ้น/ลงได้ (binary up/down , up/down counter) แทนวงจรนับขึ้นอย่างเดียว เหมือนกับตัวอย่างที่ผ่านมา วงจรนับสามารถเพิ่มค่าขึ้น หรือลดค่าลงได้ ขึ้นอยู่กับสถานะเอาท์พุทของวงจรเปรียบเทียบ ซึ่งจะทำให้รหัสไบนารี่ที่ได้มีความเป็นจริงต่อสัญญาณอนาลอกมากขึ้น



รูปที่ 3.2.7 การทำงานของ A/D แบบวงจรนับแทร็คกิง

รอบการทำงานของ A/D แบบวงจรนับแทร็คกิง เริ่มต้นที่สัญญาณอนาลอกถูกป้อนกลับมายังอินพุทของวงจรเปรียบเทียบ การนับบนวงจรเลขฐานสองขึ้น/ลง อาจจะเริ่มที่ค่าใดๆก็ได้ หมายความว่า แรงดันป้อนกลับที่มาจากตัว D/A อาจจะมากกว่า หรือน้อยกว่าสัญญาณอนาลอกทางอินพุทก็ได้ ถ้าแรงดันป้อนกลับมีค่ามากกว่าสัญญาณอนาลอกทางอินพุท เอาท์พุทของวงจรเปรียบเทียบจะมีสถานะเป็น "0" และวงจรควบคุมจะส่งสัญญาณไปเปิดเกทให้พัลส์ของสัญญาณนาฬิกาผ่านไปยังวงจรนับ แต่การนับครั้งนี้จะเป็นการนับลง ดังนั้นจึงเป็นการลดค่าเลขฐานสองซึ่งเป็นเอาท์พุทของวงจรนับ และเป็นการลดแรงดันป้อนกลับที่วงจรเปรียบเทียบลง เมื่อแรงดันป้อนกลับตกลงต่ำกว่าแรงดันทางอินพุท เอาท์พุทของวงจรเปรียบเทียบจะมีสถานะเป็น "1" ทันที และวงจรควบคุมส่งสัญญาณไปยังวงจรแลทช์ทางเอาท์พุทให้ค้างข้อมูลทางเอาท์พุทไว้ วงจรเกทจะส่งสัญญาณนาฬิกาไปเพิ่มอินพุทของวงจรนับขึ้น (ซึ่งวงจรนับจะไม่ถูกรีเซ็ต)และเป็นเหตุให้วงจรนับทำการนับค่าขึ้นอีกครั้ง สำหรับการเปลี่ยนแปลงสถานะช่วงต่อไป

ถ้าสัญญาณอินพุทยังมีค่าคงที่อยู่ เอาท์พุทที่ได้มักเกิดจากการออสซิลเลตขึ้น 1*LSB คล้ายกับตัวแปลงสัญญาณพยายามปรับค่าให้เข้าสู่ตัวกลางของมัน ปัญหาของการออสซิลเลตจึงเป็นปัญหาสำคัญของ A/D แบบวงจรนับแทร็กกิง และกลายมาเป็นข้อเสียของ การแปลงแบบนี้ แต่การแปลงแบบวงจรนับแทร็กกิง มีข้อดีที่มีความเร็วกว่าการแปลงแบบวงจรนับเดี่ยว

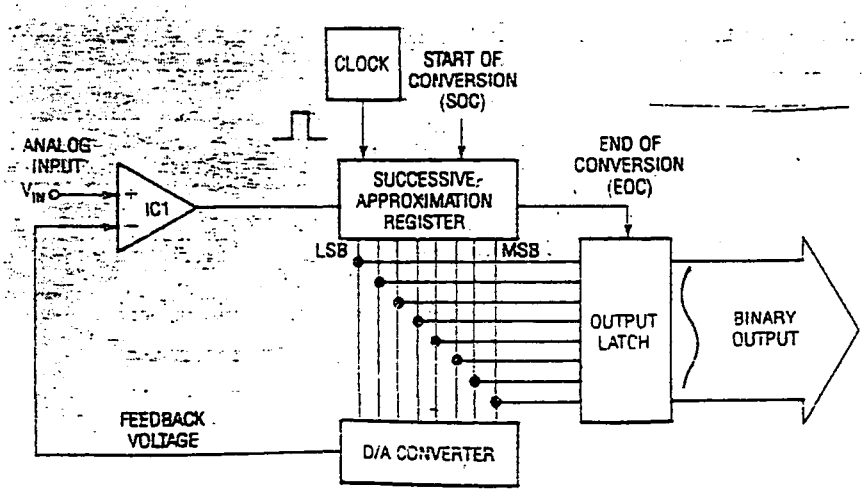
3.2.4.5 การแปลงแบบประมาณค่าหลายครั้ง (Successive-approximation Converter)

เทคนิคแบบการประมาณค่าหลายครั้งมีชื่อย่อว่า SA ซึ่งย่อมาจากคำว่า Successive Approximation เป็นเทคนิคที่น่าเลือกใช้เพราะมีราคาถูก มีความละเอียดพอสมควร และเป็นตัวแปลงสัญญาณที่มีความเร็วสูง มีประสิทธิภาพสูง และใช้งานได้ดี เพราะไม่เกิดการออสซิลเลต แต่กระบวนการของเทคนิคแบบนี้จะเข้าใจได้ยากกว่าเทคนิคแบบวงจรนับที่กล่าวมาแล้ว

หัวใจของการประมาณค่าหลายครั้ง คืออุปกรณ์ที่เรียกว่า Successive Approximation Register (SAR) ซึ่งเป็นอุปกรณ์ที่มีจุดประสงค์ต่างจากวงจรนับทั่วไป ดังรูปที่ 3.2.8

วัฏจักรการแปลงผันเริ่มต้นเมื่อสัญญาณอนาลอกถูกป้อนให้กับตัวแปลงสัญญาณ และพัลส์ของการแปลงเริ่มขึ้น (start convertion pluse : SOC) ถูกป้อนให้กับตัว SAR จะทำงานเอาท์พุทของบิทนับสูงสุด ดังนั้นจึงเป็นการปรับให้เอาท์พุทของ D/A เป็น 50 เปอร์เซ็นต์ของแรงดันเอาท์พุทของอินเวอร์เตอร์ ตัว SAR จะมองไปยังเอาท์พุทของวงจรเปรียบเทียบว่า เอาท์พุทของ D/A มีค่ามากกว่าหรือน้อยกว่าสัญญาณอนาลอกทางเอาท์พุท ถ้าแรงดัน D/A มีค่ามากกว่าวงจรเปรียบเทียบจะยังคงสภาวะไม่ทำงาน ดังนั้นตัว SAR จะ 'OFF' บิทนัยสำคัญสูงสุดลง และ

ให้ชื่อว่าสภาวะ '0' ถ้าแรงดันของ D/A มีค่าน้อยกว่าสัญญาณอนาลอก ทางเอาท์พุทของวงจรเปรียบเทียบจะยังคงทำงานอยู่ ดังนั้น SAR จะยังคงให้บิทนัยสำคัญสูงสุดทำงานอยู่ เรียกสภาวะนี้ว่า '1' ซึ่งสภาวะ '0' หรือ '1' จะกระทำภายในพัลส์ของสัญญาณนาฬิกาเพียงลูกเดียว บนสัญญาณนาฬิกาถัดไป ตัว SAR จะทำงาน บิทนัยสำคัญอันดับสอง และทำการตรวจสอบผลลัพธ์ที่ได้อีกครั้งหนึ่งจากวงจรเปรียบเทียบ ถ้าสัญญาณจาก D/A ครั้งใหม่มีค่ามากกว่าแรงดันอิน-พุทเอาท์พุทของวงจรเปรียบเทียบจะยังคงไม่มี ดังนั้นตัว SAR จะไม่ทำงานบิทนัยสำคัญสูงสุดอันดับสองเรียกว่า '0' แต่ถ้าสัญญาณจาก D/A มีค่าน้อยกว่า วงจรเปรียบเทียบจะทำงาน และตัว SAR จะปล่อยให้บิทนัยสำคัญสูงสุดอันดับสองทำงาน



รูปที่ 3.2.8 แผนผังการทำงานของ
การแปลงแบบการประมาณค่าหลายครั้ง

ตัว SAR จะพิจารณาแต่ละบิทด้วยวิธีเดียวกันจนครบทุกบิท เนื่องจากหนึ่งบิทหาค่าได้ภายในหนึ่งพัลส์ ฉะนั้น A/D ขนาด 8 บิท จึงใช้สัญญาณนาฬิกาเพียง 8 พัลส์ ก็สามารถทำการแปลงได้จนครบ เมื่อบิทนัยสำคัญต่ำสุดถูกพิจารณาแล้ว ตัว SAR จะส่งสัญญาณการแปลง (End of Converter : EOC) ไปทำการค้างผลลัพธ์ที่ได้ซึ่งเป็นเลขฐานสองทางเอาท์พุทไว้

3.2.5 **<u>การแปลงสัญญาณดิจิตอลเป็นสัญญาณอนาลอก</u>** (Digital to Analog Converter)

การแปลงสัญญาณดิจิตอลเป็นสัญญาณอนาลอกมักนิยมเรียกกันว่า DAC หรือ D/A เป็นตัวแปลงรหัสเลขฐานสองจากวงจรดิจิตอล ให้กลายเป็นระดับแรงดันอนาลอกที่เอาท์พุท ตัว D/A สามารถใช้ขับอุปกรณ์ที่เป็นอนาลอกได้ เช่น มิเตอร์ , มอเตอร์ หรือวงจรที่เกี่ยวกับสัญญาณเสียง เช่น เครื่องเล่นคอมแพคดิสก์ ให้เป็นเครื่องเล่นที่มีคุณภาพยิ่งขึ้น

แนวความคิดที่สำคัญของ D/A คือความละเอียดของ D/A หรือการกำหนดขนาดดังที่ได้กล่าวไว้ในตอนต้น นั้นคือ กำหนดขนาดไว้มาก ความละเอียดยิ่งมาก คุณภาพก็มีสูงขึ้นด้วย

นอกจากจะคำนึงถึงความละเอียดแล้ว D/A ที่ดี ควรพิจารณา*เวลาเข้าสู่สภาวะคงตัว* (setting time) เวลาเข้าสู่สภาวะคงตัว เป็นค่าของเวลาที่ระดับแรงดันเอาท์พุทเข้าสู่สภาวะคงที่ เมื่อสัญญาณไบนารีทางอินพุทเปลี่ยนแปลงไป โดยปกติจะคิดที่สัญญาณเอาท์พุทคงที่ ในช่วงของบวกลบ 1/2 ของบิทนัยสำคัญต่ำสุดของค่าที่คาดว่าจะเป็นหลังจากที่ค่าไบนารีทางอินพุทเปลี่ยนแปลงไป หมายความว่า เงื่อนไขการปฏิบัติงานจริง มีความสัมพันธ์กับค่าที่เป็นอยู่ขณะนั้น ถ้า D/A ขนาด 8 บิท มีช่วงแรงดันทางเอาท์พุทอยู่ในช่วง 0-10 ค่าเวลาเข้าสู่สภาวะคงตัวมีค่าน้อยกว่า 10 ไมโครวินาที

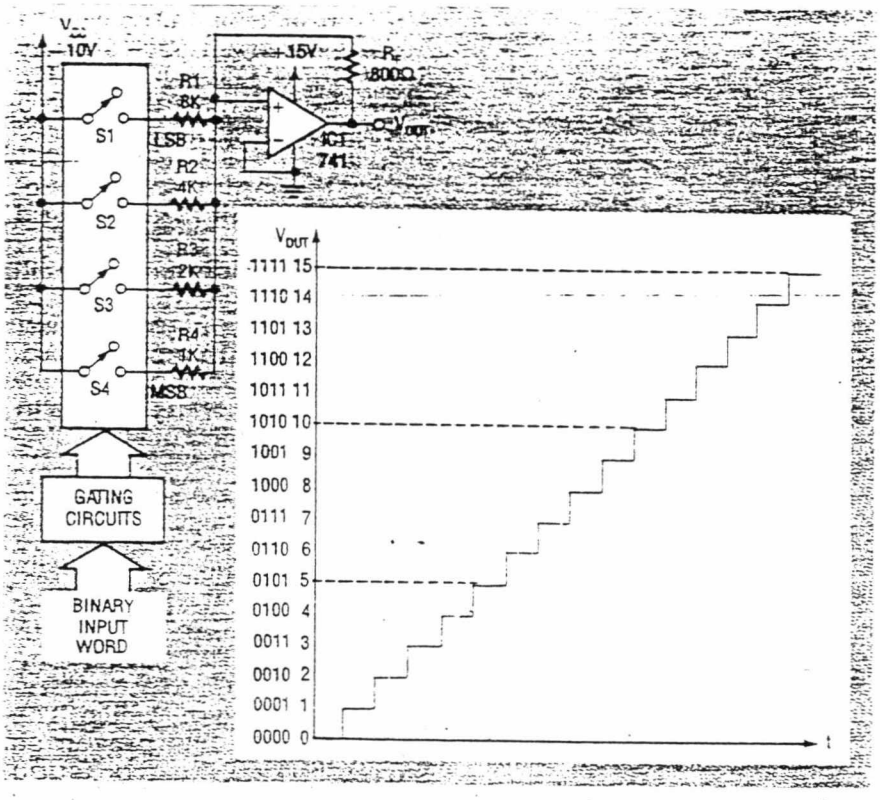
ค่าความแม่นยำเป็นปัจจัยที่สำคัญอีกตัวหนึ่ง ในเงื่อนไขปกติ ค่าความแม่นยำของ D/A คือ ± ทุกๆ ตำแหน่งจาก 1/2 ถึง 2 ค่าของบิทนัยสำคัญต่ำสุดของ D/A ซึ่งมีค่าความแม่นยำ ±1 แรงดันเอาท์พุทสามารถเปลี่ยนแปลงไปได้ในทาง บวก หรือลบ 1 บิท ถ้า D/A มีแรงดันเอาท์พุทอยู่ในชวง 0-5 โวลท์ มีความละเอียดเท่ากับ 12 บิท บิทนัยสำคัญต่ำสุดควรมีค่าเป็น $5/2^{12}$ หรือ 0.00122 โวลท์ สำหรับทุกๆ ค่าของไบนารีทางด้านเอาท์พุท แรงดันอาจจะสูง หรือต่ำกว่าค่าที่คากหมายไว้ 0.00122 โวลท์ ถ้า D/A ตัวเดียวกันมีค่าความแม่นยำเท่ากับ 1/2 ค่าความถูกต้องบิทนัยสำคัญต่ำสุด ค่าเอาท์-พุทสามารถผิดพลาดไปได้ ±0.00122/2 หรือ ±0.00061 โวลท์ ยิ่งค่าความแม่นยำน้อยเท่าไร ค่าความละเอียดยิ่งมากขึ้นเท่านั้น และจะมีค่าใกล้เคียงกับค่าเอาท์พุทที่คาดไว้

การแปลงสัญญาณดิจิตอลเป็นสัญญาณอนาลอก มี 2 วิธี ดังนี้

3.2.5.1 การแปลงแบบจัดรหัสน้ำหนักของไบนารี (Binary-weighted resistor)

เป็นเทคนิคที่ง่ายที่สุดของการแปลงสัญญาณดิจิตอลเป็นสัญญาณอนาลอก ดังแสดงไว้ในรูปที่ 3.2.9

รหัสไบนารีจะถูกป้อนให้ที่ขาเกทของอนาลอกสวิทช์ เมื่อรหัสไบนารีเป็น 0000 ถูกป้อนให้ที่ขาเกท อนาลอกสวิทช์ทั้งหมดจะเปิดวงจร ดังนั้นจึงไม่มีแรงดันเอาท์พุทจ่ายให้กับออปแอมป์ เอาท์พุทของออปแอมป์จึงเป็นศูนย์ เมื่อรหัสไบนารีเป็น 0001 สวิทช์ S_1 จะปิดลง จึงมีแรงดันขนาด 10



รูปที่ 3.2.9 วงจร D/A แบบเทคนิคการจัดรหัสน้ำหนักไบนารี

โวลท์จ่ายให้กับ R_1 เพราะเอาท์พุทของออปแอมป์จะมองว่าเป็นกราวด์เสมือน (virtual ground) เป็นผลให้แรงดัน 10 โวลท์ ตกคร่อมที่ตัวต้านทาน 8 กิโลโอห์ม ทำให้เกิดกระแส 1.25 มิลลิแอมป์ ไหลผ่านความต้านทานป้อนกลับ (R_F) ค่า 800 โอห์ม แรงดันตกคร่อม R_F ควรมีเป็นผลคูณของค่า 800 โอห์ม กับ 1.25 มิลลิแอมป์ หรือมีค่าเท่ากับ 1 โวลท์

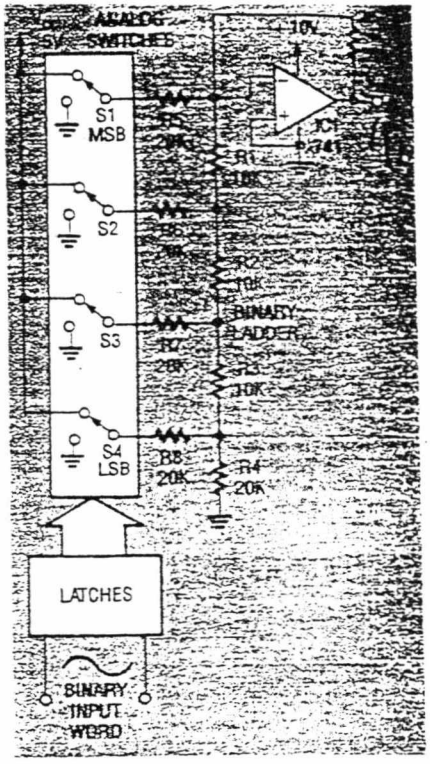
เมื่อรหัสไบนารีเปลี่ยนเป็น 0010 S_1 จะเปิด และ S_2 จะปิด เป็นเหตุให้กระแส 2.5 มิลลิแอมป์ ไหลผ่าน R_2 แรงดันตกคร่อม R_F มีค่าเท่ากับผลคูณของ 800 โอห์ม กับ 2.5 มิลลิแอมป์ หรือ 2 โวลท์ รหัสไบนารี 0100 จะให้แรงดันเอาท์พุทเท่ากับ 4 โวลท์ และถ้ารหัสไบนารีเป็น 1000 แรงดันเอาท์พุทจะมีค่าเป็น 8 โวลท์ จะสังเกตได้ว่า รหัสทางอินพุทของค่า R_F มีผลต่อระดับแรงดันทางด้านเอาท์พุท

สวิทช์แต่ละตัวสามารถปิดวงจรพร้อมกันได้ เมื่อทำการรวมกันเพื่อสร้างสัญญาณอนาลอกทางเอาท์พุท ที่มีค่าจาก 0 ถึง 15 โวลท์

ถึงแม้ว่าเทคนิคการจัดรหัสน้ำหนักของไบนารี จะมีลักษณะวงจรที่ง่าย ตรงไปตรงมา แค่สะดวกในการนำไปใช้งาน ถ้าต้องการความละเอียดของ D/A มากกว่า 4 บิท เพราะว่าค่าตัวต้านทานที่ใช้มากมายหลายค่าเกินไป ซึ่งต่างจากเทคนิคที่จะกล่าวต่อไป

3.2.5.2 การแปลงแบบแลดเดอร์เน็ทเวอร์ค (Ladder network)

เทคนิคแลดเดอร์เน็ทเวอร์ค สามารถสร้างแรงดันสมน้ำหนักของรหัสไบนารี โดยอาศัยความต้านทานเพียง 2 ค่า ที่จัดไว้ในลักษณะวงจรแบ่งแรงดัน หรือที่เรียกว่า *ไบนารี แลดเดอร์* (binary ladder) ดังแสดงในรูปที่ 3.2.10



รูปที่ 3.2.10 วงจร D/A แบบไบนารีแลดเดอร์

ถึงแม้ว่าวงจร D/A แบบแลดเดอร์เน็ทเวอร์ค ดูผ่านๆ แล้วค่อนข้างจะยาก แต่การทำงานคล้ายกับแบบการจัดรหัสน้ำหนักของไบนารี เกทที่ต่อในลักษณะอนุกรมถูกใช้ สำหรับขับอนาลอกสวิทช์ เมื่อรหัสไบนารี 0000 ถูกส่งมายังเกท อนาลอกสวิทช์ทั้งหมดจะเปิดออก ดังนั้นแรงดันเอาท์-พุทที่ได้จากออปแอมป์ จึงมีค่าเป็นศูนย์ สวิทช์ S_1 จะปิดลง เมื่อเกทได้รับรหัสไบนารีเป็น 1000 เป็นผลให้เกิดแรงดันออกมา 5 โวลท์ ที่ออปแอมป์ ถ้ารหัสไบนารีเป็น 0010 สวิทช์ S_3 ก็จะปิด และทำให้เกิดแรงดัน 1.25 โวลท์ ที่เอาท์พุทและสุดท้าย ถ้ารหัสไบนารีเป็น 0001 สวิทช์ S_4 จะปิดลง ออปแอมป์จะให้แรงดันเอาทุพุทออกมา 0.625 โวลท์ จะสังเกตได้ง่าย แต่ละแรงดันเอาท์พุทอยู่ในรูปอันดับของไบนารี คือเอาท์พุทสามารถเปลี่ยนจาก 0 ถึง 10 โวลท์ พิ่มขึ้นทีละ 0.625 โวลท์

ข้อดีของแลดเดอร์ เน็ทเวอร์ค คือ สามารถแบ่งได้ง่าย เนื่องจากใช้ความต้านทานเพียง 2 ค่า ปัจจุบัน บริษัทผู้ผลิต D/A เกือบทั้งหมด จะใช้เทคนิคแลดเดอร์ เน็ทเวอร์คแทบทั้งสิ้น

3.3 ไมโครคอนโทรลเลอร์

ไมโครคอนโทรลเลอร์ตระกูล MCS-51 เป็นไมโครคอนโทรลเลอร์แบบชิพเดี่ยว (Single Chip Microcontroller) คือไมโครโปรเซสเซอร์ที่รวมอยู่บนแผงวงจรรวม ที่มีส่วนหน่วยความจำ ส่วนควบคุม และส่วนอินพุท/เอาท์พุท รวมอยู่บนชิพตัวเดียวกัน โดยมีลักษณะดังจะกล่าวต่อไป

3.3.1 ลักษณะทั่วไปของ MCS-51

ลักษณะหลักทั่ว ๆ ไปของ MCS-51 จะประกอบด้วย

1. ใช้ HMOS และ CHMOS เทคโนโลยีในการสร้าง และทำงานด้วยแหล่งจ่ายไฟขนาด 5 V เพียงแหล่งเดียว

2. ซีพียูมีขนาดคำ 8 บิท

3. มีวงจรออสซิลเลเตอร์ และวงจรนาฬิกาบนชิป

4. ชุดแบงค์ (BANK) รีจิสเตอร์มี 4 ชุด แต่ละชุดมีรีจิสเตอร์ 8 ตัว ทำงานเช่นเดียวกับ MCS-48

5. มีตัวจับเวลา/ตัวนับ ขนาด 16 บิท 2 ชุด และสำหรับเบอร์ 8032/8052 มี 3 ชุด

6. มีพอร์ตไอโอแบบขนานสองทิศทางจำนวน 4 พอร์ต พอร์ตละ 8 บิท รวมทั้งหมดเป็น 32 เส้น แต่จะเหลือเพียง 16 เส้น สำหรับเบอร์ 8031 อีก 16 เส้น จะใช้ในการเข้าถึงทางแอดเดรสและข้อมูล

7. พอร์ตแบบอนุกรมสามารถที่จะโปรแกรมการรับส่งแบบ Full Duplex ที่ความเร็วสูง

8. หนึ่งวัฏจักรคำสั่งจะกินเวลา 1 ไมโครวินาที ด้วยการใช้คริสทัล 12 เมกกะเฮิร์ตซ์

9. แอดเดรสข้อมูลภายนอกได้ 64 กิโลไบท์

10. แอดเดรสโปรแกรมภายนอกได้ 64 กิโลไบท์

11. สามารถกำหนดเลขที่อยู่ข้อมูลขนาดไบท์ หรือบิทได้โดยตรง

12. มีซอฟต์แวร์บิทแฟลกสำหรับผู้ใช้ที่จะกำหนดเองได้ 128 ตำแหน่งบิท

13. โครงสร้างอินเตอร์รัพต์จะติดตั้งได้ 5 แหล่ง และ 6 แหล่ง สำหรับ 8032/8052 พร้อมด้วยการจัดไพรโอริตี (Priority) ได้ 2 ระดับ

14. ตัวโพรเซสเซอร์สามารถใช้งานแบบบูลีน (Boolean) ได้ สำหรับการใช้กับกระบวนการงานควบคุม

15. มีคำสั่งคูณ และหารทางฮาร์ดแวร์ที่ทำได้ภายใน 4 ไมโครวินาที

16. ตัวเลขทางคณิตศาสตร์ ใช้ได้ทั้งระบบไบนารี และเดซิมอล

17. การใช้พื้นที่สแตกสำหรับโปรแกรมย่อยต่าง ๆ ทำได้กว้างกว่า MCS-48

18. ชุดคำสั่งของ MCS-51 จะมีความสามารถสูงกว่าคำสั่งของ MCS-48

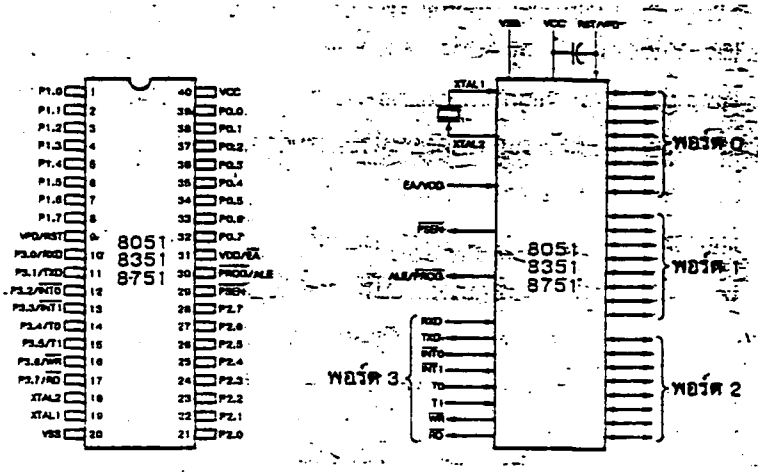
ตระกูล MCS-51 จะมีทั้งแบบมี ROM ในตัว หรือไม่มี ROM หรือมี EPROM บนชิปเดียวกัน แสดงถึงตารางรายละเอียดของเบอร์ต่าง ๆ ในตระกูล MCS-51 ที่มีจำหน่ายในท้องตลาด

Device	ROMless Version	EPROM Version	ROM Bytes	RAM Bytes	8-Bit I/O Ports	16-Bit Timer/ Counters	Programmable Counter Array (PCA)	UART	Serial Expansion Port (SEP)	Global Serial Channel (GSC)	DMA Channels	A/D Channels	Interrupt Sources/ Vectors	Power Down and Idle Modes
8051	8031	--	4K	128	4	2		✓					6/5	
8051AH	8031AH	87C51 87C51B	4K	128	4	2		✓					6/6	
8052	8032AH	87C52B	8K	256	4	3		✓					8/6	
80C51BH	80C31BH	87C51	4K	128	4	2		✓					6/5	✓
83C51FA	80C51FA	87C51FA	8K	256	4	3	✓	✓					14/7	✓
83C51FB	80C51FA	87C51FB	16K	256	4	3	✓	✓					14/7	✓
83C51GA	80C51GA	87C51GA	4K	128	4	2		✓	✓			8	8/7	✓
83C152JA	80C152JA	--	8K	256	6	2		✓		✓	2		19/11	✓
--	80C152JB	--	--	256	7	2		✓		✓	2		19/11	✓
83C152JC	80C152JC	--	8K	256	5	2		✓		✓	2		19/11	✓
--	80C152JD	--	--	256	7	2		✓		✓	2		19/11	✓
83C451	80C451	--	4K	128	7	2		✓					6/5	✓
83C452	80C452	87C452P	8K	256	5	2		✓					9/8	✓

ตารางที่ 3.3.1 ตารางรายละเอียดของตระกูล MCS-51

8751H อยู่ในกลุ่มรุ่นเดียวกับ 8051AH ที่สามารถโปรแกรมได้ด้วยระบบไฟสามารถลบโปรแกรมออกได้ด้วยแสงอุลตราไวโอเลต นอกเหนือจากไอซีที่แสดงในตารางข้างบนที่ใช้เทคโนโลยี HMOS แล้วยังมีตระกูลอื่นที่ใช้เทคโนโลยี CHMOS ที่ประหยัดพลังงานได้มากกว่า 4 เท่าของ CMOS ที่มีจำหน่ายขณะนี้คือ เบอร์ 80C51,80C31 และ 87C51

3.3.2 การจัดขาของ MCS-51



รูปที่ 3.3.1 : ลักษณะการจัดขาของ MCS-51

รูปที่ 3.3.1 แสดงการจัดขาตามลักษณะภายนอกของชิป MCS-51 ซึ่งจะมีการแบ่งกลุ่มการจัดขาตามสถาปัตยกรรมของ MCS-51 อยู่ 4 กลุ่ม คือ

1) กลุ่มขารับแหล่งจ่ายไฟฟ้า และระบบสัญญาณนาฬิกา

2) กลุ่มขาแอดเดรสและข้อมูล

3) กลุ่มขาควบคุม

4) กลุ่มขาพอร์ตแบบขนานและอนุกรม

ขาบางขาจะมีหน้าที่ได้สองหน้าที่ขึ้นอยู่กับการติดตั้งด้วยซอฟต์แวร์หรือฮาร์ดแวร์ เช่น ขาที่ 32-39 จะทำหน้าที่ได้เป็นกลุ่มขาแอดเดรสและข้อมูลหรือจะทำหน้าที่เป็นกลุ่มขาพอร์ตแบบขนานเป็นต้น รายละเอียดหน้าที่ขาแต่ละขาจะมีดังนี้

ขา *Vss* : ขาที่ 20

เป็นขาสำหรับต่อลงดิน

ขา *Vcc* : ขาที่ 40

เป็นขาที่ต่อแรงดันไฟกระแสตรงขนาด 5 V และใช้สำหรับการโปรแกรม

***ขา PORT 0* : ขาที่ 32-39**

ทำหน้าที่เป็นพอร์ตไอโอ 8 บิทแบบ Open Drain (P0.0 - P0.7/Bidirectional สามารถที่จะรับโหลดทีทีแอลได้ 8 ตัว การเขียนค่า '1' ไปที่พอร์ตนี้ จะเป็นการปล่อยลอย (Float) ขาของพอร์ตนี้ ทำให้มันทำงานเป็นอินพุท มีสถานะอิมพีแดนซ์สูง ในการให้พอร์ตนี้บริการแบบไอโอ และอีกหน้าที่หนึ่งของพอร์ต0 จะทำงานเป็นมัลติเพลกซ์ ด้วยสัญญาณแอดเดรสไบท์ต่ำกับบัสข้อมูล สำหรับการใช้งานด้านหน่วยความจำภายนอก ในการใช้งานแบบนี้จะใช้ลักษณะภายในเป็นตัวพูลอัพ นอกจากหน้าที่หลัก 2 หน้าที่ดังกล่าวแล้ว พอร์ต0 ยังใช้งานพิเศษเป็นตัวส่งข้อมูลออกทางพอร์ตนี้ เมื่อใช้บริการทางด้านการตรวจสอบโปรแกรม ROM ภายใน และการโปรแกรมตัว EPROM ภายใน ถ้าใช้งานในลักษณะนี้การพูลอัพจากภายนอกจะต้องต่อด้วยค่า 10 กิโลโอห์ม

***ขา PORT 1* : ขาที่ 1-8**

เป็นพอร์ตไอโอ 8 บิทแบบ Open Drain Bidirectional พร้อมด้วยการพูลอัพภายใน ถ้าเป็นพอร์ตเอาท์พุท บัฟเฟอร์สามารถขับโหลดทีทีแอลตระกูลแอลเอสไอได้ 4 ตัว พอร์ต1 เมื่อถูกเขียนค่า '1' ด้วยโปรแกรมจะมีสถานะสูงด้วยการพูลอัพภายใน การให้สถานะเช่นนี้จะเป็นการ Initial ใช้งานพอร์ตนี้ให้เป็นอินพุท ขณะที่พอร์ต1 เป็นอินพุท การให้สัญญาณลงต่ำจะเป็นการจ่ายกระแสออกเนื่องจากการพูลอัพภายในในเบอร์ 8052 ขา P1.0 และ P1.7 จะใช้งานเป็น T2 และ T2EX โดยขา T2 จะทำหน้าที่รับสัญญาณจากภายนอกให้ตัวจับเวลา2 ทำงาน และขา T2EX จะเป็นอินพุทผ่านเข้าตัวจับเวลา2 ถูกกระตุ้นให้ทำงานแบบปกติตามโปรแกรมที่ติดตั้งไว้ หรือ เค็ปเจอร์ (Capture)

***ขา PORT 2* : ขาที่21-28**

เป็นพอร์ตไอโอ 8 บิทแบบ Open drain Bidirectional ด้วยการพูลอัพภายใน พอร์ต2 ที่ทำหน้าที่เป็นบัฟเฟอร์ เอาท์พุทสามารถจ่ายโหลดทีทีแอลตระกูลแอลเอสได้ 4 ตัว อีกหน้าที่หนึ่งของพอร์ตจะถูกใช้งานเป็นตัวส่งแอดเดรสไบท์สูงด้วย เมื่อใช้งานร่วมกับหน่วยความจำภายนอก เพื่อให้แอด--เดรสได้ถึง 16 บิท ด้วยการใช้งานแบบนี้มันจะมีพลูอัพภายในที่ช่วยให้การส่งค่า '1' ได้ระดับที่แน่นอน นอกจากการใช้งานสำหรับแอดเดรสอันดับสูงยังใช้เป็นขาควบคุมในการใช้งานตรวจสอบ และเขียนโปรแกรมเบอร์ 8751 และตรวจสอบโปรแกรมภายใน 8051

ขา *PORT 3* : ขาที่ 10-17

เป็นพอร์ตไอโอ 8 บิทแบบพูลอัพภายใน นอกจากทำเป็นพอร์ตไอโอที่สามารถโหลดทีทีแอลพวกตระกูลแอลเอสได้ 4 ตัวแล้วยังมีอีกหน้าที่หนึ่งของตระกูล MCS-51ตามรายการข้างล่างนี้ด้วย ขาพอร์ต ขาการทำงานตามฟังก์ชันพิเศษ

P3.0 RxD : **ขาที่ 10** พอร์ตอนุกรมอินพุท

P3.1 TxD : **ขาที่ 11** พอร์ตอนุกรมเอาท์พุท

P3.2 INT0 : **ขาที่ 12** อินเตอร์รัพต์ภายนอกตัวที่1

P3.3 INT1 : **ขาที่ 13** อินเตอร์รัพต์ภายนอกตัวที่2

P3.4 T0 : **ขาที่ 14** T0 สัญญาณกระตุ้นเข้าที่ตัวจับเวลา/ตัวนับ0

P3.5 T1 : **ขาที่ 15** สัญญาณกระตุ้นเข้าที่ตัวจับเวลา/ตัวนับ1

P3.6 WR : **ขาที่ 16** สัญญาณควบคุมการเขียน

P3.7 RD : **ขาที่ 17** สัญญาณควบคุมการอ่าน การที่จะให้ทำงานตามฟังก์ชันข้างบนได้จะต้องติดตั้งโปรแกรมด้วยการส่งค่า '1' ไปแลทช์ไว้ก่อนที่ให้ทำงานตามฟังก์ชันข้างบน

ขา *RST* : ขาที่ 9

ต้องคงสถานะค่าสูงเป็นเวลาประมาณอย่างน้อยสองวัฏจักรระหว่างที่ออสซิเลเตอร์ทำงานขณะที่ต้องการรีเซตทั้งระบบงาน โดยจะต่อรีซิสเตอร์พูลดาวน์ (8.2 กิโลโอห์ม) จากขา RST ไปลงดินและเพื่อให้ตัวชิปรีเซตได้โดยอัตโนมัติ ขณะเปิดไฟจะใช้คาปาซิเตอร์ (10 ไมโครฟารัด) ต่อคร่อมระหว่างขา RST กับขา Vcc

ขา *XTAL1* : ขาที่ 19

ใช้เป็นตัวอินพุทเข้าสู่ตัวออสซิเลเตอร์ขยายแบบ Invert

ขา *XTAL2* : ขาที่ 18

ใช้เป็นตัวเอาท์พุทจากตัวออสซิเลเตอร์ขยายแบบ Invert

ขา *PSEN* : ขาที่ 29

Program Strorage Enable เป็นสโตรบอ่านข้อมูลจากโปรแกรมหน่วยความจำภายนอก เมื่อชิปทำงานด้วยโปรแกรมภายนอก ขา PSEN จะสร้างสโตรบต่ำสองครั้งภายในแต่ละวัฏจักรแมชชีน สัญญาณจะมีสถานะสูงหรือพัลส์ต่ำทั้งสองลูกจะหายไป เมื่อทำงานในช่วงการอ่านหรือเขียนข้อมูลจากหน่วยความจำข้อมูลภายนอก และ PSEN จะไม่มีพัลส์ส่งออกถ้าชิปทำงานด้วยโปรแกรมหน่วยความจำภายใน

ขา *ALE/PROG* : ขาที่ 30

เป็นขาแอดเดรสแลทช์อีนาเบิลด้วยการส่งพัลส์ออกไปใช้สำหรับแลทช์ค่าแอดเดรสไบท์ต่ำจากพอร์ต0 ในระหว่างเข้าถึงข้อมูลจากหน่วยความจำภายใน ALE จะถูกส่งสัญญาณนาฬิกาออกมาในอัตราความเร็วคงที่ที่ 1/8 ของความถี่ออสซิเลเตอร์ตลอดเวลา แม้ว่าบางช่วงจังหวะจะไม่มีการเข้าถึงข้อมูลจากภายใน ดังนั้นจึงสามารถที่จะใช้สัญญาณจากขานี้เป็นตัวจับเวลาภายนอกหรือเป็นความถี่ช้าลงไปเท่าหนึ่งระหว่างการทำงานแบบการเข้าถึงของหน่วยความจำข้อมูลภายนอก ขานี้ยังจะใช้เป็นสัญญาณพัลส์เข้าสำหรับการควบคุมการโปรแกรม EPROM ภายในชิป

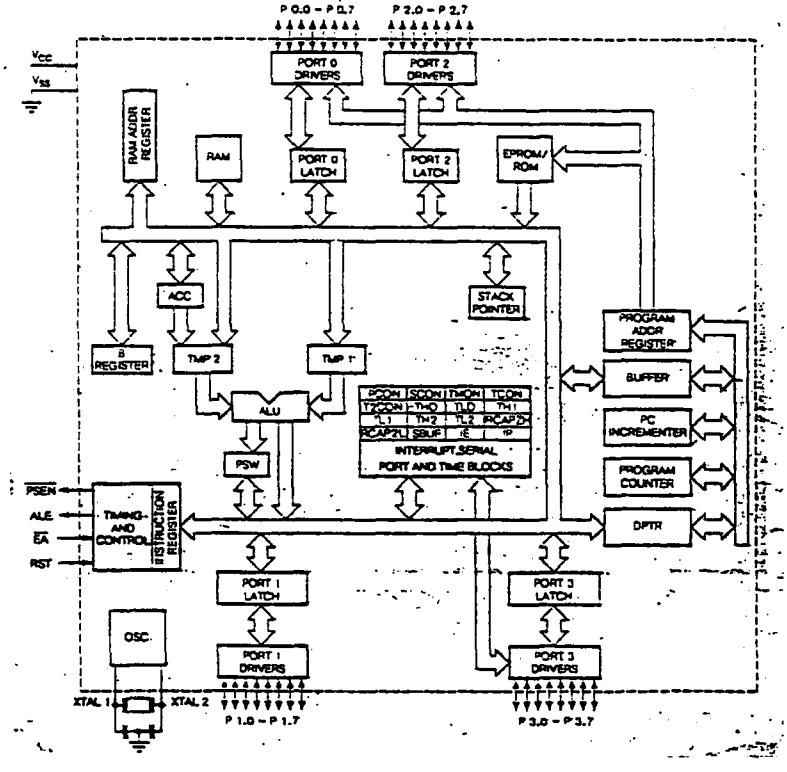
ขา *EA/Vpp* : ขาที่ 31

มีสถานะสูง ตัวซีพียูในชิปจะทำงานตามโปรแกรมที่อยู่ในหน่วยความจำภายใน (โดยที่โปรแกรมจะต้องไม่ยาวกว่า 4 กิโลไบท์ สำหรับเบอร์ 8051 AH และ 8 กิโลไบท์ สำหรับเบอร์ 8052 AH) การทำให้ EA มีสถานะต่ำจะเป็นการควบคุมให้ซีพียูทำงานตามโปรแกรมหน่วยความจำภายนอก ซึ่งขยายโปรแกรมได้ยาวถึง 64 กิโลไบท์ ในตัว 8031 AH และ 8032 AH ขา EA จะต้องต่อลงดินเช่นกันแม้ว่าจะไม่มี ROM อยู่ภายในก็ตาม ในตัว 8751 H จะใช้ขานี้จ่ายแรงดันขนาด 21 V ขณะทำการเขียนโปรแกรมเข้า EPROM ของชิป 8751H ตัวนี้

ตามตารางที่ 3.3.1 MCS-51 ทั้งสามกลุ่ม คือ กลุ่มที่มี ROM ไม่มี ROM และพวก EPROM จะมีขาใช้งานเหมือนกันหมด ยกเว้นขา 1 จะใช้งานเป็น T2 และขา 2 เป็น T2EX ในเบอร์ 8032/8052 ตลอดถึงจังหวะเวลา (Timing Diagram) และคุณสมบัติทางไฟฟ้าทั้งสามจะแตกต่างกันเฉพาะการโปรแกรมบนชิป MCS-51 เท่านั้น ซึ่งแต่ละแบบจัดไปตามความต้องการของผู้ใช้ เช่น 8751 จะมี 4 กิโลไบท์ของ Ultraviolet-Erasable Programmable Read Only Memory (EPROM) เหมาะสำหรับการพัฒนาเครื่องต้นแบบ และการผลิตอุปกรณ์ที่จำนวนจำกัด เมื่อต้องการจะเขียนโปรแกรมเข้า EPROM จะมีตัวเขียนโปรแกรมพิเศษสำหรับเขียนโปรแกรมที่ผู้ออกแบบเขียนขึ้นมาได้

ถ้าโปรแกรมมีบักหรือส่วนผิดพลาดที่ต้องการจะแก้ไข ก็สามารถแก้ไขได้โดยการนำตัว 8751 นี้ ไปล้างโปรแกรมเดิมออกด้วยแสงอุลตร้าไวโอเลต และอัดข้อมูลโปรแกรมที่ได้แก้ไขแล้วเข้าไปใหม่ ทำเช่นนี้จนกระทั่งได้โปรแกรมสมบูรณ์ และเมื่อต้องการผลิตจำนวนมากก็สามารถที่จะใช้ MCS-51 เบอร์ 8051 ที่มี 4 กิโลไบท์ของ ROM ซึ่งจะถูกอัดข้อมูลโปรแกรมตามความต้องการของผู้ออกแบบ โดยโรงงานผู้ผลิตชิปเบอร์นี้ การผลิตลักษณะนี้จะถูกกว่าการใช้เบอร์ 8751 แต่โปรแกรมภายในจะไม่สามารถลบ และโปรแกรมใหม่ได้หลังการผลิตไปแล้ว

ส่วนเบอร์ 8031 จะไม่มีหน่วยความจำของโปรแกรมบนชิป แต่อาจต่อหน่วยความจำโปรแกรมจากภายนอกด้วย ROM EPROM หรือ PROM ได้ถึง 64 กิโลไบท์ ดังนั้น 8031 จึงเหมาะสำหรับการใช้งานที่โปรแกรมมีขนาดใหญ่กว่าสี่กิโลไบท์ และสำหรับผู้ออกแบบที่ต้องการแยกส่วนของโปรแกรมออกจากชิป



รูปที่ 3.3.2 : โครงสร้างสถาปัตยกรรมภายในของ MCS-51

3.3.3 สถาปัตยกรรมภายในของ MCS-51

รูปที่ 3.3.2 เป็นบล็อกไดอะแกรมที่แบ่งตามลักษณะงานทางสถาปัตยกรรมภายในของ MCS-51 โดยชิงเกิลชิปแต่ละตัวของตระกูลนี้จะประกอบด้วยหน่วยศูนย์กลางประมวลผล หน่วยความจำสองชนิด คือ แบบ RAM กับ ROM หรือ EPROM พอร์ตเอาท์พุท อินพุท โหมดรีจิสเตอร์สถานะและข้อมูล ส่วนวงจรตรรกในการ RANDOM ที่จำเป็นสำหรับตัวแปรของฟังก์ชันการต่อพ่วงส่วนต่างๆ ที่กล่าวนี้จะติดต่อกันด้วยบัสข้อมูลขนาด 8 บิท และจะมีบัฟเฟอร์สำหรับการติดต่อข้อมูลกับภายนอกผ่านพอร์ตไอโอ เมื่อต้องการขยายหน่วยความจำหรือพอร์ตไอโอ

3.3.4 หน่วยศูนย์กลางประมวลผล

ซีพียูเป็นมันสมองของระบบไมโครคอมพิวเตอร์ การอ่านโปรแกรม และทำงานตามคำสั่งโปรแกรมจะกระทำที่ส่วนนี้ โดยการใช้ส่วนคณิตศาสตร์ และตรรกศาสตร์ทำงานร่วมกับรีจิสเตอร์ A,B,PSW (Program Status Word),SP (Stack Pointer) ตัวนับโปรแกรม (PC:Program Counter) ขนาด 16 บิท และตัวชี้ตำแหน่งข้อมูล (DPTR:DataPointer) ส่วนคณิตศาสตร์และตรรกศาสตร์ด้วยตัวแปรต่างๆ ขนาด 8 บิท ที่มีลักษณะการทำงานทางคณิตศาสตร์เป็น บวก ลบ คูณ หาร รวมทั้งทางตรรกศาสตร์ เช่น AND OR XOR รวมทั้งการเลื่อนและวนรอบบิท การเคลียร์ค่าและกลับค่า (Complement) เป็นต้น ALU ยังสามารถที่จะตัดสินใจในการให้กระโดดไปทำคำสั่งของโปรแกรมในส่วนอื่นๆ ตามเงื่อนไขที่ตั้งขึ้นและยังแบ่งรีจิสเตอร์ชั่วคราวใช้สำหรับเป็นทางผ่านชั่วคราวของข้อมูลในการถ่ายเทภายในระบบคำสั่งอื่นที่มีการใช้ ALU ALU ยังมีความสามารถที่จะเพิ่มค่าในรีจิสเตอร์ในลักษณะการบวกด้วยหนึ่ง (Increment) หรือ คำนวณเลขที่อยู่ของข้อมูลที่จะนำไปเก็บ หรือการลดค่าลงครั้งละหนึ่ง ในลักษณะการลบด้วยค่าหนึ่ง (Decrement) โดยอัตโนมัติ หรือใช้ในการเปรียบเทียบค่าของตัวแปรทั้งสอง

สิ่งสำคัญในการทำงานทางสถาปัตยกรรมของ MCS-51 คือ ความสามารถในการทำงานสำหรับข้อมูลขนาด 8 บิท และ 1 บิท การใช้งานในระดับบิทในการเซต เคลียร์หรือกลับค่า การเคลื่อนย้าย การทดสอบ และใช้ในการคำนวณทางตรรกขนาด 1 บิท ความสามารถเช่นนี้เหมาะสำหรับใช้ในงานควบคุมของสัญญาณเข้าและออกที่มีการคิดและออกแบบทางตรรกด้วยพีชคณิต Boolean ซึ่งโดยปกติทำได้ลำบากสำหรับไมโครโพรเซสเซอร์ทั่วๆ ไป งานในลักษณะเช่นนี้จึงได้ชื่ออีกอย่างหนึ่งว่า *ตัวประมวลผลบูลีน* (Boolean Processor)

***แอกคูมิวเลเตอร์* (Accumlator : ACC)**

MCS-51 ก็เช่นเดียวกับ MCS-48 ที่ใช้ ACC ที่มีขนาด 8 บิท เป็นแอกคูมิวเลเตอร์คำสั่งส่วนใหญ่จะอ้างถึงตัวรีจิสเตอร์นี้ โดยถือค่าภายในเป็นค่าตัวตั้ง และรับค่าผลลัพธ์ที่ได้จากคำสั่งทางคณิตศาสตร์ เช่น บวก ลบ คูณ หาร เข้ามาเก็บไว้ ตัว ACC ยังสามารถใช้เป็นตัวแหล่งกระทำหรือถูกกระทำในการทำงานทางตรรก และใช้เป็นตัวกลางในการถ่ายเทข้อมูล ในการติดต่อกับอุปกรณ์ภายนอกไอโอ และหน่วยความจำภายนอก รวมถึงการตรวจสอบตารางข้อมูล

รีจิสเตอร์ B

เป็นรีจิสเตอร์พิเศษที่ใช้งานสำหรับคำสั่งของการคูณและหาร โดยใช้เป็นที่เก็บตัวคูณหรือตัวหาร และเป็นที่เก็บผลลัพธ์ตัวที่สองหลังการคูณและเศษหลังการหาร

***ตัวชี้สแตก* (Stack Pointer : SP)**

MCS-51 จะใช้ RAM ภายในเป็นบริเวณสแตกทางฮาร์ดแวร์สำหรับการเชื่อมต่อระหว่างโปรแกรมหลัก สแตกการผ่านพารามิเตอร์ระหว่างงานในแต่ละส่วนโปรแกรม และสแตกเก็บตัวแปรข้อมูลชั่วคราว หรือสแตกการเก็บสถานะระหว่างการบริการงานอินเตอร์รัพต์ไว้ภายในชิป โดยที่ SP จะมีขนาด 8 บิท จะเพิ่มค่าขึ้นโดยอัตโนมัติก่อนที่ข้อมูลจะนำมาเก็บในหน่วยความจำระหว่างการใช้คำสั่ง PUSH และ CALL และจะลดค่าของ SP ลงหลังจากที่ได้ถ่ายเทข้อมูลออกไปแล้วในคำสั่ง POP หรือ RETURN โดยทฤษฎีทางสถาปัตยกรรม MCS-51 สามารถใช้สแตกให้มีเนื้อที่ถึง 128 ไบท์ แต่ในทางปฏิบัติสำหรับโปรแกรมทั่วไปจะใช้น้อยกว่านี้ SP จะเริ่มที่ตำแหน่ง 07H ดังนั้น สแตกจะเริ่มบรรจุข้อมูลที่ตำแหน่ง 08H MCS-51 สามารถเปลี่ยนแปลงค่าใน SP ได้ ซึ่งจะเป็นการเปลี่ยนตำแหน่งสแตกไปยังที่ใดๆ ของ RAM ภายในชิป

***ตัวชี้ข้อมูล* (Data Pointer : DPTR)**

DPTR รีจิสเตอร์ขนาด 16 บิท ที่ประกอบด้วยไบท์สูง (DPH) และ ไบท์ต่ำ (DPL) ที่สามารถเลือกแบ่งออกเป็น รีจิสเตอร์ 8 บิทสองตัวที่ใช้ได้อย่างอิสระ หรือจะใช้รวมกันทั้ง 16 บิท ก็ได้ ในการ Increment หรือ Decrement เพื่อประโยชน์ในการใช้เป็นฐานของเลขที่อยู่ในรีจิสเตอร์ในการกระโดดโดยทางอ้อมในการใช้คำสั่งเกี่ยวกับตารางข้อมูลและชี้ตำแหน่งของหน่วยความจำภายนอก

รีจิสเตอร์คำแสดงสถานะโปรแกรม (Program Status Word : PSW)

(MSB) **(LSB)**

CY	AC	F0	RS1	RS0	OV	-	P

สัญลักษณ์	ตำแหน่ง	ข้อกำหนดการทำงาน
CY	PSW7	แฟลกตัวทด จะเซต/เคลียร์ด้วยฮาร์ดแวร์หรือ ซอฟต์แวร์ ระหว่างผลลัพธ์ หลังการใช้คำสั่งทางคณิตศาสตร์ หรือ ตรรกศาสตร์ที่แน่นอน
AC	PSW6	แฟลกตัวทดของ Auxiliary จะเซต/เคลียร์ด้วยฮาร์ดแวร์ ระหว่างการบวก และลบ ที่แสดงผลจากการทดหรือยืมจากบิทที่ 3 ของ ACC
F0	PSW5	แฟลก 0 จะเซต/เคลียร์ด้วยซอฟต์แวร์ที่ผู้ใช้กำหนดสถานะแฟลกนี้เอง
RS1	PSW4	รีจิสเตอร์ตัวควบคุมการเลือกแบงค์ด้วยค่า RS1 และ RS0
RS0	PSW3	จะเซต/เคลียร์ด้วยซอฟต์แวร์ เพื่อเลือกกลุ่มรีจิสเตอร์ทำงานในแต่ละแบงค์ โดยปรับค่าใน RS1 และ RS0 ให้อินาเบิลคลุมลักษณะการเลือกแบงค์ต่อไปนี้

RS1	RS0	เลือกแบงค์	ค่าแอดเดรส
0	0	แบงค์ 0	00H - 07H
0	1	แบงค์ 1	08H - 0FH
1	0	แบงค์ 2	10H - 17H
1	1	แบงค์ 3	18H - 1FH

OV	PSW2	แฟลก Overflow จะเซต/เคลียร์ด้วยฮาร์ดแวร์ระหว่างการใช้คำสั่งที่แสดงผลถึงการเกิดลักษณะ Overflow ทางคณิตศาสตร์
X	PSW1	บิทสำรอง จะไม่สามารถเซต/เคลียร์ด้วยผู้ใช้เพราะ สำรองไว้สำหรับโรงงานผู้สร้าง
P	PSW0	แฟลกพาริตี จะเซต/เคลียร์ด้วยฮาร์ดแวร์ในแต่ละวัฏจักรคำสั่ง

แสดงถึงตัวเลขค่า '1' ในแต่ละบิทของแอกคูมิวเลเตอร์ เช่น '1' มี 6 ตัวจะเป็นพาริตีคู่ P บิทจะเท่ากับ 0

หมายเหตุ ความหมายของฮาร์ดแวร์ และซอฟต์แวร์ในตารางต่างๆ ที่จะกล่าวต่อไปนี้ ในแต่ละบิทของตัวรีจิสเตอร์ การที่บิทจะเซตหรือเคลียร์นั้น ถ้าเกิดขึ้นจากฮาร์ดแวร์จะหมายถึงว่า ค่าบิทในรีจิสเตอร์จะเกิดเซตตัวเองเนื่องจากผลของความหมายของการทำงานตามคำสั่งของบิทนั้น เช่น TI สามารถจะเซตตัวเองด้วยฮาร์ดแวร์ เมื่อการส่งข้อมูลได้สิ้นสุดถึง STOP บิทแล้ว ช่วยให้สามารถตรวจสอบได้ว่าการส่งข้อมูลครั้งละไบท์นั้นสิ้นสุดหรือยัง ถ้ายังจะได้รอต่อไปก่อน หรือมีการคำนวณแล้วผลลัพธ์เกิด Overflow ใน PSW ก็จะเซตตัวเองที่บิท OV ส่วนทางซอฟแวร์ หมายถึงว่าสามารถที่จะเซต หรือเคลียร์ได้ด้วยการใช้คำสั่งโปรแกรมต่างๆ ในการเซตหรือเคลียร์ในบิทแต่ละบิท ของรีจิสเตอร์เป็นลักษณะทางซอฟต์แวร์

รีจิสเตอร์ PSW เป็นรีจิสเตอร์ที่แสดงผลที่ได้หลังจากการใช้คำสั่งต่าง ๆ และใช้เป็นตัวเลือกกลุ่มการทำงานของรีจิสเตอร์กลุ่มต่าง ๆ

พอร์ต 0 ถึง 3

รีจิสเตอร์ P0,P1,P2 และ P3 ของกลุ่มรีจิสเตอร์ฟังก์ชันพิเศษ (Special Function Register : SFR) จะเป็นตัวรีจิสเตอร์ที่แลทช์ค่าของพอร์ต 0,1,2 และ 3 ตามลำดับ ในขณะใช้งาน

***บัฟเฟอร์ข้อมูลอนุกรม* (Serial Data Buffer : SBUF)**

บัฟเฟอร์ข้อมูลอนุกรมแบ่งออกเป็นรีจิสเตอร์สองตัว ตัวหนึ่งเป็นบัฟเฟอร์การส่งและอีกตัวเป็นบัฟเฟอร์การรับ เมื่อข้อมูลถ่ายเทเข้า SBUF มันจะถ่ายเข้าบัฟเฟอร์ส่งซึ่งเป็นตัวจัดการส่งข้อมูลอนุกรม วิธีการเคลื่อนย้ายเข้า SBUF ขึ้นอยู่กับการติดตั้งโปรแกรม (Initial) การส่งเมื่อข้อมูลย้ายออกจาก SBUF จะเป็นการรับข้อมูลจากบัฟเฟอร์ตัวรับ

รีจิสเตอร์ CAPTURE

ไอซีเบอร์ 8032/8052 จะมีคู่รีจิสเตอร์ (RCAP2H,RCAP2L) เพิ่มเติมเป็นรีจิสเตอร์เค็ปเจอร์สำหรับตัวจับเวลา 2 ในโหมดการใช้งานของรีจิสเตอร์ตัวนี้จะรับการเปลี่ยนแปลงที่เข้ามาในขา T2EX ตัว TH2 และ TL2 จะลอกข้อมูลเข้าไปในรีจิสเตอร์คู่ RCAP2H และ RCAP2L ด้วยการใช้ตัวจับเวลา จะมีโหมดการบรรจุอัตโนมัติขนาด 16 บิท สำหรับการใช้ตัวจับเวลา/ตัวนับ2 ซึ่งจะมีรายละเอียดในหัวข้อต่อไป

MCS-51 จะจัดแบ่งตำแหน่งสำหรับ SFR ให้ทำงานเป็นรีจิสเตอร์ต่าง ๆ ดังนี้

SPECTIAL FUNCTION REGISTERS	ตำแหน่ง
* ACC Accumulator	0E0H
* B รีจิสเตอร์	0F0H
* PSW Program Status Word	0D0H
SP Stack Pointer	081H
DPTR ตัวชี้ข้อมูล ประกอบด้วย	
DPH	083H
DPL	082H
* P0 พอร์ต 0	080H
* P1 พอร์ต 1	090H
* P2 พอร์ต 2	0A0H
* P3 พอร์ต 3	0B0H
* IP ตัวควบคุมการอินเตอร์รัพต์ตามลำดับ	0B8H
* IE ตัวควบคุมการอินเตอร์รัพต์อีนาเบิ้ล	0A8H
TMOD ตัวควบคุมการเลือกโหมดตัวตั้งเวลา/ตัวนับ	089H
* T2CON ตัวควบคุมตัวตั้งเวลา/ตัวนับ 2	088H
TCON ตัวควบคุมตัวตั้งเวลา/ตัวนับ	0C8H
TH0 รีจิสเตอร์ตัวตั้งเวลา/ตัวนับ 0 (ไบท์สูง)	08CH
TL0 รีจิสเตอร์ตั้งเวลา/ตัวนับ 0 (ไบท์ต่ำ)	08AH
TH1 รีจิสเตอร์ตัวตั้งเวลา/ตัวนับ 1 (ไบท์สูง)	08DH
TL1 รีจิสเตอร์ตัวตั้งเวลา/ตัวนับ 1 (ไบท์ต่ำ)	08BH

+ TH2 รีจิสเตอร์ตัวตั้งเวลา/ตัวนับ 2 (ไบท์สูง)	0CDH
+ TL2 รีจิสเตอร์ตัวตั้งเวลา/ตัวนับ 2 (ไบท์ต่ำ)	0CCH
+ RLDH รีจิสเตอร์ตัวตั้งเวลา/ตัวนับ 2 ประจุใหม่อัตโนมัติ (ไบท์สูง)	0CBH
+ RLDL รีจิสเตอร์ตัวตั้งเวลา/ตัวนับ 2 ประจุใหม่อัตโนมัติ(ไบท์ต่ำ)	0CAH
* SCON ควบคุมการส่งข้อมูลอนุกรม	098H
SBUF บัฟเฟอร์ข้อมูลการส่งอนุกรม	099H
PCON ควบคุมการใช้พลังงาน (Power)	097H

เครื่องหมาย * หน้าตัวรีจิสเตอร์แสดงว่า รีจิสเตอร์นั้นสามารถที่จะแอดเดรสข้อมูลได้ทั้งข้อมูลขนาดไบท์และบิท

เครื่องหมาย + นั้นแสดงว่าจะมีเฉพาะในเบอร์ 8032/8052 เท่านั้น

รีจิสเตอร์ควบคุม **(Control Register)**

กลุ่ม SFR ที่เป็น IP,IE,TMOD,TCON,T2CON,SCON และ PCON จะประกอบด้วยบิทที่ใช้ในการควบคุม และแสดงสถานะของการใช้งานในระบบอินเตอร์รัพต์ ตัวจับเวลา/ตัวนับและพอร์ต อนุกรมซึ่งจะอธิบายโดยละเอียดในหัวข้อต่อไป

3.3.5 <u>การจัดหน่วยความจำ</u>

ตัว MCS-51 จะแยกแอดเดรสสำหรับหน่วยความจำของโปรแกรม และหน่วยความจำของข้อมูลออกจากกัน หน่วยความจำของโปรแกรมขยายได้ถึง 64 กิโลไบท์ และจำนวนไบท์ต่ำ 4 กิโลไบท์จะอยู่ใน 8051 หน่วยความจำของข้อมูลภายในมี 128 ไบท์ (256 ไบท์สำหรับ 8032/8052) บนชิป และอีก 128 ไบท์ใช้สำหรับรีจิสเตอร์ฟังก์ชันพิเศษ (Special Function Register : SFR) และเข้าถึงหน่วยความจำข้อมูลภายนอกได้อีก 64 กิโลไบท์

3.4 ไมโครคอมพิวเตอร์กับการสื่อสารข้อมูล

เมื่อข้อมูลผ่านการแปลงสัญญาณเป็นข้อมูลทางดิจิตอลแล้ว ข้อมูลส่วนนี้ถูกนำมาวิเคราะห์บนไมโครคอมพิวเตอร์ ซึ่งจะต้องมีการส่งผ่านข้อมูลจากบอร์ดมาเก็บไว้ในไมโครคอมพิวเตอร์ การ ส่งผ่านข้อมูลหรือการถ่ายโอนข้อมูลสามารถกระทำได้ทั้งแบบขนานและอนุกรม โดยที่การถ่ายโอนข้อมูลแบบขนานอาจผ่านทางพอร์ตพริ นเตอร์ซึ่งต้องทำความเข้าสัญญาณต่างๆ ที่ใช้ติดต่อกับพริ นเตอร์ แต่หากทำการถ่ายโอนข้อมูลแบบอนุกรม สามารถทำการถ่ายโอนได้หลายแบบตามแต่ระยะทาง และความเร็วที่ต้องการ หรืออาจถ่ายโอนข้อมูลด้วยการทำเป็นการ์ดเสียบสล็อทคอมพิวเตอร์ก็ได้ แต่จะมีข้อเสียที่มีสัญญาณรบกวนมากกว่า

ในหัวข้อนี้จะกล่าวรูปแบบการสื่อสารต่างๆ โดยเน้นที่การสื่อสารแบบอนุกรมเป็นหลัก เนื่องจากในโครงงานชิ้นนี้ใช้การสื่อสารแบบอนุกรม

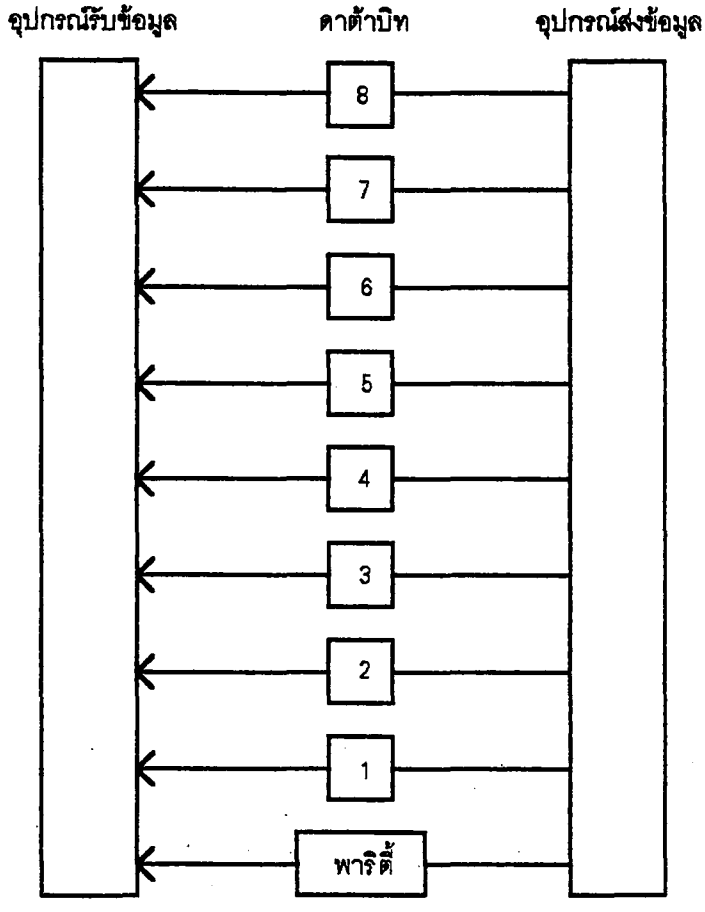
3.4.1 วิธีการถ่ายโอนข้อมูล

ก่อนที่กล่าวถึงการสื่อสารระหว่างเครื่อง ลองมาศึกษาว่าสัญญาณที่ส่งและรับนั้น มีวิธีการอย่างไร ดังนี้

3.4.1.1 การถ่ายโอนข้อมูลแบบขนาน

ลักษณะการถ่ายโอนข้อมูลแบบขนาน ทำได้โดยการส่งข้อมูลออกมาทีละ 1 ไบท์ คือ 8 บิท จากอุปกรณ์ส่งไปยังอุปกรณ์รับ ตัวกลางระหว่างเครื่อง 2 เครื่อง จะต้องมีช่องทางให้ข้อมูลเดินทางได้อย่างน้อย 8 ช่องทาง โดยมากจะเป็นสายขนานให้กระแสวิ่งผ่านไปมากกว่าจะเป็นอุปกรณ์อย่างอื่น เนื่องจากมีสัญญาณสูญหาย ไปกับความต้านทานของสาย ระยะทางระหว่างเครื่องจึงไม่ควรเกิน 100 ฟุต ปัญหาที่เกิดขึ้นหากระยะทางมากกว่านี้คือ ระดับของกราวด์ทางไฟฟ้าที่จุดรับผิดไปจากจุดส่ง ทำให้เกิดความผิดพลาดในการรับสัญญาณลอจิกทางฝ่ายรับ

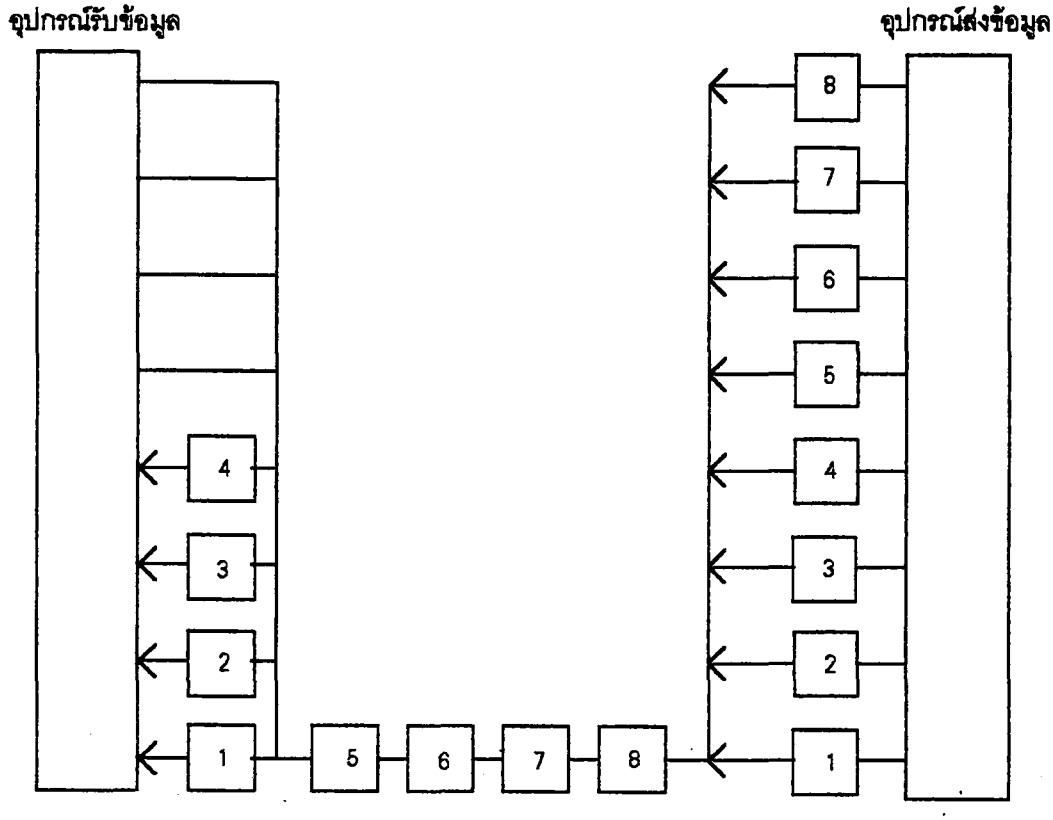
นอกจากสายที่เป็นทางเดินของข้อมูลแล้วอาจจะมีทางเดินของสัญญาณควบคุมอื่นๆ อีก เป็นต้นว่า บิทที่บอกพาริ ตีของสัญญาณ เพื่อเป็นการตรวจสอบความผิดพลาดของการรับสัญญาณที่ปลายทาง หรือสายที่ควบคุมการโต้ตอบ (Hand-shake) ดังที่กล่าวมาแล้ว



รูปที่ 3.4.1 การส่งข้อมูลแบบขนาน

3.4.1.2 การถ่ายโอนข้อมูลแบบอนุกรม

ในการถ่ายโอนข้อมูลแบบอนุกรม ข้อมูลถูกส่งออกมาทีละบิท ระหว่างจุดส่งและจุดรับจะเห็นว่าการส่งข้อมูลแบบนี้จะช้ากว่าแบบขนานที่กล่าวมาแล้วแน่นอน แล้วทำไมต้องส่งแบบนี้ละ คำตอบก็คือ ต้องการสายเพียงคู่เดียว ค่าใช้จ่ายในสื่อกลางจะถูกกว่าแบบขนานอย่างแน่นอน สำหรับการส่งระยะทางไกลๆ โดยเฉพาะเมื่อมีระบบการสื่อสารทางโทรศัพท์ไว้ใช้งานอยู่แล้วย่อมจะเป็นการประหยัดกว่าที่จะทำการติดต่อสื่อสารทีละ 8 ช่อง เพื่อการถ่ายโอนข้อมูลแบบขนานอย่างแน่นอน



รูปที่ 3.4.2 การส่งข้อมูลแบบอนุกรม

รูปที่ 3.4.2 แสดงให้เห็นการส่งข้อมูลแบบอนุกรม ข้อมูลจากจุดส่งจะถูกเปลี่ยนให้เป็นอนุกรมเสียก่อนแล้วค่อยทยอยส่งทีละบิทไปยังจุดรับ ณ ที่จุดรับจะต้องมีกลไกในการเปลี่ยนข้อมูลที่ส่งมาทีละบิท ให้เป็นสัญญาณแบบขนานซึ่งลงตัวพอดี นั่นคือ บิท 1 ลงที่บัสข้อมูลเส้นที่ 1 พอดี การที่จะทำให้การแปลงสัญญาณจากอนุกรมทีละบิทให้ลงพอดีนั้นจำเป็นต้องมีกลไกที่เหมาะสม เพื่อป้องกันการผิดพลาดในการรับ กลไกที่ว่านี้แบ่งเป็น 2 แบบ คือ

1. การสื่อสารแบบซิงโครนัส

2. การสื่อสารแบบอะซิงโครนัส

รายละเอียดของทั้งสองแบบจะกล่าวในหัวข้อถัดไป

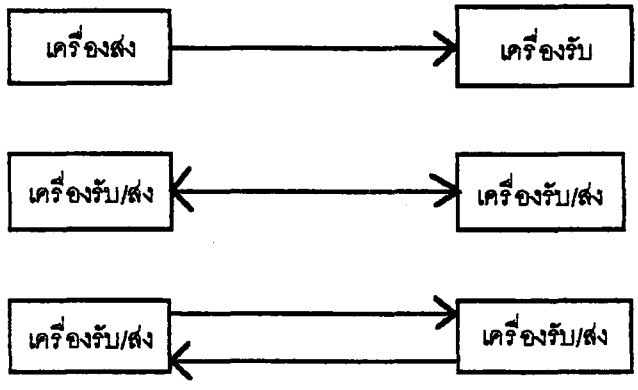
3.4.2 <u>รูปแบบการติดต่อสื่อสารแบบอนุกรม</u>

การติดต่อแบบอนุกรมอาจจะแบ่งตามรูปลักษณะได้ 3 รูปแบบ ตามรูปที่ 3.4.3

1) แบบซิมเพล็กซ์ **(Simplex)** ข้อมูลส่งได้ในทางเดียวเท่านั้น บางครั้งเรียกว่าการส่งทิศทางเดียว (Unidirectional dats bus)

2) แบบฮาล์ฟดูเพล็กซ์ (Half duplex) ข้อมูลสามารถส่งได้ทั้งสองสถานี แต่จะต้องผลัดกันส่งและผลัดกันรับ จะส่งและรับพร้อมกันไม่ได้

3) แบบฟลูดูเพล็กซ์ (Full duplex) ทั้งสองสถานีสามารถรับและส่งได้ในเวลาเดียวกัน



รูปที่ 3.4.3 รูปแบบของการติดต่อสื่อสารข้อมูลแบบอนุกรม

การส่งแบบฟูลดูเพล็กซ์และฮาล์ฟดูเพล็กซ์ ไม่ขึ้นอยู่กับจำนวนของสายในการติดต่อ บางครั้งคำว่า ทูไวร์ (two wire) หรือสองเส้น และโฟร์ไวร์ (four wire) หรือ 4 เส้น ใช้ในการบรรยายถึงลักษณะการสื่อสารข้อมูลซึ่งอาจจะทำให้เข้าใจและฮาล์ฟดูเพล็กซ์ สายโทรศัพท์ทั่วไปเป็นแบบ 2 เส้น ส่วนในสายที่เป็นแบบเช่า (lease line) นั้นส่วนมากจะเป็น 4 เส้น

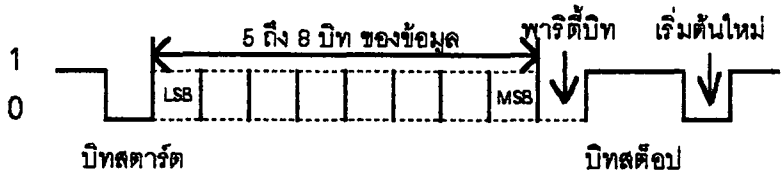
3.4.3 <u>ความเร็วในการถ่ายโอนข้อมูลแบบอนุกรม</u>

ความเร็วของการถ่ายโอนข้อมูลแบบอนุกรม หน่วยวัดเป็นบิทต่อวินาที (bps) หน่วยที่บรรยายถึงการเปลี่ยนแปลงของสัญญาณใน 1 วินาที เรียกว่าบอดเรต (baud rate) หรืออัตราบอด หลายคนยังเข้าใจสับสนระหว่างอัตราบอดและอัตราบิท (bit rate) การเปลี่ยนแปลงของสัญญาณ 1 ครั้ง อาจจะแสดงถึงการส่งข้อมูลแบบอนุกรมมากกว่า 1 บิท (รายละเอียดเกี่ยวกับเรื่องนี้จะได้กล่าวถึงอีกครั้งในเรื่องของโมเด็ม) ถ้าเขียนในรูปของสมการทางคณิตศาสตร์เราก็จะได้

อัตราบิท(bit rate) = อัตราบอด(baud rate) * บิทใน 1 บอด

3.4.4 การสื่อสารแบบอะซิงโครนัส

การส่งแบบอะซิงโครนัสนี้ พัฒนามาจากการส่งโทรพิมพ์ในสมัยก่อน ลักษณะของสัญญาณแสดงไว้ในรูปที่ 3.4.4 เพื่อเพิ่มกลไกในการรับส่งอย่างถูกต้อง สัญญาณอะซิงโครนัส จะประกอบด้วยบิท เริ่มต้นหรือบิทสตาร์ท (start) และบิทสิ้นสุดหรือบิทสต๊อป (stop bit)



รูปที่ 3.4.4 รูปแบบของการสื่อสารแบบอะซิงโครนัส

ขณะที่สถานะของการส่งเป็นแบบว่าง (Idle) คือ ยังไม่มีสัญญาณส่งออกมา จะมีสัญญาณหรือมีแรงดัน (หรือกระแส) ตลอดเวลา เพื่อความแน่ใจว่าฝ่ายรับยังติดต่ออยู่กับฝ่ายส่ง เมื่อเริ่มจะส่งข้อมูล สัญญาณของอะซิงโครนัสจะเป็น 0 หนึ่งช่วงสัญญาณนาฬิกา บิทนี้เรียกว่า สตาร์ทบิท ตามหลังของสตาร์ทบิทก็จะเป็นข้อมูลสำหรับ 1 ตัวอักษร ซึ่งอาจจะมีขนาดตั้งแต่ 5 บิท จนถึง 8 บิท โดยบิทที่มีค่าน้อยที่สุด (LSB) จะถูกส่งออกมาก่อนไล่ไปจนถึงบิทที่มีค่ามากที่สุด (MSB) การเข้ารหัสอักขระนี้ส่วนมากจะนิยมใช้รหัส ASCII แรกเริ่มทีเดียวในงานของโทรพิมพ์เขาใช้รหัส Baudot ซึ่งใช้ 5 บิท ในการแทนอักขระ 1 ตัว ตามหลังข้อมูลก็จะเป็นพาริตีบิท ซึ่งอาจจะใช้หรือไม่ใช้ก็ได้ พาริตีบิททำหน้าที่เป็นตัวตรวจสอบความถูกต้องของสัญญาณที่ได้รับ พาริตีบิทอาจจะเป็นแบบ คู่ (Even) หรือแบบคี่ (Odd) หมายความว่าถ้าหากเป็นพาริตีคู่ จำนวนบิทที่เป็น 1 ในช่วงบิทข้อมูลกับบิทพาริตีรวมแล้วจะต้องเป็นจำนวนคู่ ผู้ส่งจะต้องทำหน้าที่ตรวจสอบข้อมูลแล้วใส่พาริตีบิทเอง ฝ่ายรับเมื่อรับแล้วก็จะต้องตรวจสอบดูว่าเป็นจริงดังสถานการณ์ที่ตั้งเอาไว้หรือไม่ หากผิดพลาดก็หมายความว่าสัญญาณที่รับนั้นผิดพลาดไปจากสถานีส่งส่งออกมา ทั้งนี้ทั้งนั้นจะต้องผิดเป็นจำนวนคี่เท่านั้น คือผิดไป 1 บิท 3 บิท หรือ 5 บิท พร้อมกันจึงจะตรวจสอบได้ว่าผิด มองเห็นง่าย ๆ ว่าถ้าผิดเป็นจำนวนคู่ ผลรวมของจำนวนหนึ่งก็ยังเป็นคู่อยู่ดี ทั้งนี้ทั้งนั้นไม่ได้หมายความว่าพาริตีคี่ (Odd Parety) จะตรวจสอบการผิดพลาดเป็นจำนวนคี่ ความจริงแล้วตรวจสอบความผิดพลาดได้เหมือนกับพาริตีคู่ (Even Parity) แต่แทนที่จะตรวจสอบดูว่าสัญญาณที่รับเข้ามีจำนวนคู่ ก็ตรวจสอบดูว่ามีจำนวนคี่หรือเปล่าอย่างไรก็ตามโอกาสที่จะผิดพลาด 2 บิทพร้อมกันมีน้อยมาก

ย้อนกลับมาดูสัญญาณอะซิงโครนัสใหม่ หลังจากบิทพาริตีแล้วก็จะต้องมีสต๊อปบิทซึ่งเป็น 1 ความกว้างของสต๊อปบิทอาจจะเป็น 1,1.5 หรือ 2 พัลส์ของสัญญาณนาฬิกา แล้วแต่ผู้รับและผู้ส่งจะตกลงใช้กันเอง การเริ่มใช้พอร์ตอนุกรม (ทางออกอนุกรม) จึงจำเป็นจะต้องตั้งค่าต่าง ๆ สำหรับเป็นการส่งแบบอนุกรมอันได้แก่

1. ความเร็วในการส่ง
2. ความยาวรหัส 1 อักขระ
3. บิทตรวจสอบ
4. จำนวนสต๊อปบิท

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

จะเห็นว่ากลไกในการซิงโครนัสของการสื่อสารอะซิงโครนัส มีลักษณะเป็นไปทีละตัวอักขระ จำนวนพัลส์ของสัญญาณที่ส่งออกยังมีบางส่วนใช้ในการควบคุมการส่งอยู่อันได้แก่ บิทสตาร์ท บิทสต๊อป และบิทพาริตี ทำให้ความเร็วการส่งอักขระต่อวินาทีน้อยลงไป การส่งสัญญาณด้วยความเร็ว 300 บอด สำหรับการเข้ารหัส 7 บิท ไม่ได้หมายความว่าส่งได้ 300 หารด้วย 7 อักขระต่อวินาที

3.4.5 <u>การสื่อสารแบบอะซิงโครนัสที่มีการแมตช์ความเร็ว</u>

ได้กล่าวถึงการถ่ายโอนข้อมูลจากแบบอนุกรมจากอุปกรณ์เครื่องหนึ่งไปยังอีกเครื่องหนึ่ง ซึ่งอาจจะเป็นคอมพิวเตอร์หรืออุปกรณ์สื่อสารชนิดอื่น โดยสมมติฐานว่าความเร็วในการเปลี่ยนสัญญาณจากขนานเป็นอนุกรมได้เร็วพอ และฝ่ายรับเปลี่ยนจากอนุกรมเป็นขนานแล้วนำไปแสดงบนจอพิมพ์ออกที่เครื่องพิมพ์หรือเก็บไว้ในดิสค์ทันทีได้ทันเวลาด้วยความเร็วในแต่ละการทำงานเท่ากันทั้งฝ่ายรับและฝ่ายส่ง ไม่มีการหน่วงเวลาหรือการอินเตอร์รัพต์ระหว่างตัวกลาง อย่างไรก็ตามสมมติฐานนี้ย่อมไม่เป็นความจริง ฝ่ายส่งทำหน้าที่ส่งอย่างเดียวแต่ฝ่ายรับอาจต้องทำหน้าที่หลายอย่าง เช่น รับ แสดงผล เก็บ พิมพ์ เป็นต้น ความเร็วของฝ่ายรับหากไม่เพียงพอที่จะทำหลายอย่างให้ทันกับฝ่ายส่ง (แน่นอนย่อมขึ้นอยู่กับความเร็วในการส่ง) ก็จำเป็นจะต้องมีกลไกในการควบคุมการถ่ายโอน เทคนิคในการควบคุมความเร็วในการส่งมีอยู่หลายรูปแบบซึ่งอาจจะแบ่งออกได้เป็น 2 ลักษณะ คือ ส่งข้อ-มูลบอกการออนออฟการทำงาน (on-off data flow toggle) และหาที่เก็บข้อมูลชั่วคราวหรือสร้างบัฟเฟอร์ (Temporary data storage mechanism)

3.4.6 การควบคุมความเร็วในการรับส่งข้อมูลที่ไม่เท่ากัน

เนื่องจากการใช้ภาษาในระดับสูงเขียนเป็นโปรแกรมสำหรับการควบคุมการทำงานของการถ่ายโอนข้อมูลแบบอนุกรม อาจจะใช้เวลามากกว่าที่จะรับข้อมูลเข้ามาได้ทันกับสถานีส่งข้อมูลมา จำเป็นจะต้องมีวิธีการควบคุมไม่ให้เกิดการสูญหายของข้อมูลที่สถานีส่งมา วิธีการดังกล่าวนี้มีอยู่หลายวิธี คือ

3.4.6.1 การมีบัฟเฟอร์ในการสื่อสารข้อมูล

บัฟเฟอร์สำหรับการสื่อสารก็คือหน่วยความจำในคอมพิวเตอร์ซึ่งแบ่งแยกออกมาจากหน่วยความจำหลักสำหรับเก็บพักข้อมูลในการติดต่อชั่วคราว บัฟเฟอร์สำหรับการสื่อสารนี้ส่วนมากใช้สำหรับฝ่ายรับเท่านั้น เนื่องจากฝ่ายรับจำเป็นจะต้องตามฝ่ายส่งให้ทัน ถ้าหากฝ่ายรับใช้ภาษาแอส-แซมบลี้ควบคุม ถ้ามีความเร็วพออาจไม่จำเป็นต้องใช้บัฟเฟอร์สำหรับการสื่อสารเนื่องจากภาษาแอส-แซมบลี้มีความเร็วสูง

ข้อมูลที่จัดส่งให้คอมพิวเตอร์ที่เป็นฝ่ายรับ ส่วนมากจะอ่านมาจากแฟ้มที่บันทึกไว้ในดิสค์ หากพิจารณาระหว่างการส่งข้อมูลออก ข้อมูลที่อ่านมาจากดิสค์จะมีลักษณะเป็นกลุ่มได้รับการนำมาสู่บัฟเฟอร์ การอ่านแต่ละกลุ่มดำเนินไปจนกระทั่งบัฟเฟอร์เต็ม การอ่านจะหยุดลงจนกระทั่งบัฟเฟอร์ถูกส่งออกไปหมดในลักษณะของเข้าก่อนออกก่อน ข้อมูลก็จะถูกอ่านออกมาใส่ในบัฟเฟอร์ส่งอีกครั้ง โดยปกติบัฟเฟอร์ส่งจะมีขนาด 255 ตัวอักษรหรือประมาณ 3 บรรทัดของ 80 อักษร

บัฟเฟอร์ของฝ่ายรับมีผลกระทบต่อการรับ-ส่งข้อมูลมากกว่าบัฟเฟอร์ส่ง บัฟเฟอร์รับทำหน้าที่เช่นเดียวกับบัฟเฟอร์ส่ง แต่ทิศทางของการไหลของข้อมูลอยู่ในทางตรงกันข้าม ฝ่ายรับข้อมูลเข้ามาเก็บไว้ในบัฟเฟอร์รับก่อนจนกว่าโปรแกรมควบคุมการสื่อสารจะนำข้อมูลออกไปจากบัฟเฟอร์รับเพื่อไปแสดงหรือพิมพ์หรือเก็บไว้ในแฟ้มก็แล้วแต่ ขอเพิ่มเติมอีกนิดว่าในระบบควบคุมการทำงานของไมโครคอมพิวเตอร์อย่างเช่น IBM PC มีกลไกบัฟเฟอร์รับส่งนี้ไว้อยู่ โปรแกรมในระดับสูงจึงเพียงแต่ทำหน้าที่ดึงเอาข้อมูลจากบัฟเฟอร์นี้ไปใช้ เราจะเห็นได้ชัดถึงความจำเป็นในการใช้บัฟเฟอร์เมื่อความเร็วในการส่งสูงเกินกว่า 600 บอด ภาษาในระดับสูง เช่น ภาษาเบสิกไม่สามารถที่จะรับข้อมูลจากพอร์ตอนุกรมได้ทันแน่ๆ ระบบควบคุมการทำงานจึงถูกออกแบบมาเพื่อการสื่อสารข้อมูล โดยการใช้อินเตอร์รัพต์เข้าช่วยเมื่อมีข้อมูลเข้ามาที่พอร์ตอนุกรทมเมื่อไร ระบบควบคุมจะอินเตอร์รัพต์การทำงานเพื่อดึงข้อมูลไปใส่ในบัฟเฟอร์รับทันที เพื่อไม่ให้ข้อมูลที่รับหายไปก่อนเมื่อมีตัวใหม่ส่งมาทับที่พอร์ตอนุกรม

หน้าที่ของโปรแกรมควบคุมการรับส่งก็คือ การอ่านข้อมูลจากบัฟเฟอร์รับไปใช้เมื่อถูกอ่านจากบัฟเฟอร์รับไปแล้ว ตัวที่อ่านออกไปก็จะหายไปจากบัฟเฟอร์ ลองนึกภาพดูจะเห็นว่าฝ่ายหนึ่งคือระบบควบคุมการทำงาน (OS) รับข้อมูลจากพอร์ตอนุกรมใส่บัฟเฟอร์ อีกฝ่ายหนึ่งคือโปรแกรมควบคุมการรับส่งดึงข้อมูลออกจากบัฟเฟอร์ เปรียบเสมือนคนหนึ่งตักน้ำใส่ตุ่ม อีกคนหนึ่งตักน้ำออกจากตุ่ม ถ้าฝ่ายที่ตักออกมีความเร็วมากกว่าตุ่มก็อาจจะมีโอกาสแห้ง ในทางตรงกันข้ามถ้าฝ่ายที่ตักออกช้ากว่าฝ่ายที่ตักเข้าโอกาสที่จะล้นตุ่มก็ย่อมจะมี ในทางสื่อสารเรียกว่าบัฟเฟอร์รับโอเวอร์โฟลว์ (receive buffer overflow) การไหลล้นดังกล่าวทำให้ข้อมูลที่รับหายไป

โปรแกรมที่เขียนด้วยภาษาแอสแซมบลี้สามารถทำงานได้เร็ว และอาจไม่จำเป็นต้องใช้บัฟเฟอร์ สำหรับการรับส่งเลยก็ได้ แต่ถ้าหากเขียนด้วยภาษาเบสิกจำเป็นจะต้องมีบัฟเฟอร์อย่างน้อย 1024 ไบท์ สำหรับการสื่อสารที่ความเร็วไม่เกิน 600 บอด

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

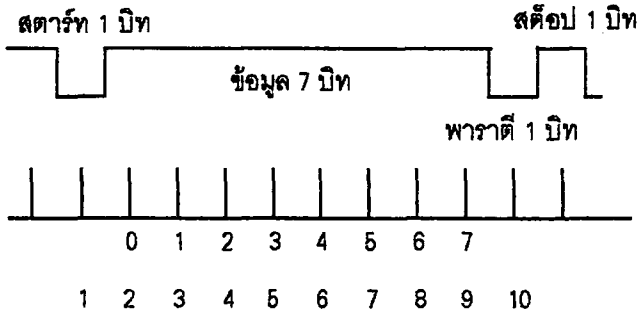
เป็นการกำหนดบัฟเฟอร์การสื่อสารด้วยขนาด 4 กิโลไบท์ ถ้าหากเราไม่กำหนดค่าบัฟเฟอร์สื่อสารเอาไว้ DOS จะตั้งค่าให้เท่ากับ 256 ไบท์ โดยการใส่ / C: ทำให้เราสามารถกำหนดบัฟเฟอร์ได้จาก 0 ถึง 32,767 ไบท์ อย่างไรก็ตามภาษาเบสิกใช้หน่วยความจำได้ 64 กิโลไบท์ การเรากำหนดค่าบัฟเฟอร์สื่อสารเอาไว้มากย่อมจะมีหน่วยความจำเหลือสำหรับใช้อย่างอื่นน้อยลง เมื่อเข้าสู่ภาษาเบสิกแล้วแรกเริ่มภาษาเบสิกจะบอกจำนวนหน่วยความจำเหลือใช้ทำงานจริงใ ในขณะที่โหลดโปรแกรมภาษาเบสิกเข้ามา ขนาดของโปรแกรมย่อมจะเข้าไปอาศัยที่ในหน่วยความจำที่เหลือ ทำให้หน่วยความจำที่จะใช้งานสำหรับเก็บตัวแปรต่างๆ น้อยลงไปอีก เราสามารถใช้คำสั่งโดยตรงสำหรับการตรวจดูขนาดของหน่วยความจำที่ว่างโดยการพิมพ์คำสั่ง PRINT PRE(0)

3.4.6.2) การควบคุมโดยใช้ XON/XOOF

ถึงแม้ว่าเราจะมีบัฟเฟอร์สำหรับการสื่อสารแล้วก็ตาม ในบางครั้งการถ่ายโอนข้อมูลด้วยความเร็วสูงและด้วยขนาดของแฟ้มที่จะทำการถ่ายโอนมีขนาดใหญ่กว่าบัฟเฟอร์สื่อสาร โอกาสที่ข้อมูลจะหายไปมีอยู่มาก ลองมาคำนวณดูง่ายๆ สมมติว่าเราใช้ความเร็วในการถ่ายโอนข้อมูล 9600 บิทต่อวินาที สต๊อปบิทเป็น 1 บิท ข้อมูล 7 บิท และพาริตีเป็น 1 คู่ ใน 1 ตัวอักษร จะต้องใช้ 11บิท

เพราะฉะนั้นฝ่ายรับจะต้องอ่านข้อมูลจากพอร์ตอนุกรมทุก 110/9600 ได้ผลประมาณ .001 วินาที หรือ 1 มิลลิวินาที ถ้าเปรียบเป็นพัลส์ได้ประมาณ 5000 พัลส์นาฬิกา (ความเร็วนาฬิกาของ IBM PC = 4.77 MHz หรือประมาณ .2 ไมโครวินาทีต่อหนึ่งพัลส์ 1000/.2 = 5000) ในเมื่อบัฟเฟอร์ ไม่เพียงพอ ก็จำเป็นต้องควบคุมการรับส่ง โดยการบอกให้ฝ่ายส่งหยุดส่งชั่วคราว (XOFF) จนกว่าฝ่าย

รับจะจัดการเอาข้อมูลออกจากบัฟเฟอร์สื่อสารหมดเสียก่อน จึงบอกให้ฝ่ายส่งจัดการส่งต่อไป (XON) ในรหัส ASCII XON มีค่าเท่ากับ 17 XOFF มีค่าเท่ากับ 19 เป็นหน้าที่ของนักเขียนโปรแกรมที่จะต้องจัดการส่ง XOFF ออกไปให้ฝ่ายส่งได้รับรู้ก่อนที่บัฟเฟอร์สื่อสารจะเต็มเสียก่อน



รูปที่ 3.4.5 รูปแบบของข้อมูล 1 ตัวอักษร

คอมพิวเตอร์เมนเฟรมส่วนมากจะมีระบบ XON/XOFF ให้ สำหรับการเชื่อมต่อทางด้านความเร็ว (Speed Matching) แต่โปรแกรมสื่อสารที่มีขายสำหรับ IBM PC ไม่มี XON/XOFF ทุกตัว โดยมากโปรแกรมที่เขียนโดยภาษาแอสแซมบลี้จะมี XON/XOFF ให้ แต่โปรแกรมที่เขียนโดยภาษาเบสิกจะมีเพียงบางโปรแกรมเท่านั้น โปรแกรมที่เขียนโดยภาษาแอสแซมบลี้มีความเร็วพอที่จะคอยตรวจสอบดู XOFF ที่ส่งมาจากฝ่ายรับ แต่ถ้าเป็นภาษาเบสิก ความเร็วอาจจะไม่เพียงพอต่อการตรวจสอบ XOFF ที่ฝ่ายส่ง

3.4.6.3 การใช้โปรโตคอล

อีกเทคนิคหนึ่งในการควบคุมการรับส่งก็คือ การส่งโปรโตคอล (Protocol Transfer) เทคนิคนี้จำเป็นจะต้องมีเหมือนกันทั้งฝ่ายรับและฝ่ายส่ง โดยการใช้อักขระควบคุมในตารางของ ASCII สำหรับควบคุมการส่งข้อมูลออกมาเป็นกลุ่มที่มีขนาดคงที่

การใช้โปรโตคอลอาจจะใช้อักขระต่อไปนี้ ในการควบคุมการส่งข้อมูลเป็นกลุ่มๆ

ETB End of Transmission block

ETB มีค่าเท่ากับ 23 ในตาราง ASCII เป็นการบอกฝ่ายรับว่าขณะนี้สิ้นสุดการส่งข้อมูลกลุ่มที่หนึ่งแล้ว

ETX End of Text

ETX มีค่าเท่ากับ 03 ในตาราง ASCII เป็นการบอกฝ่ายรับว่าขณะนี้สิ้นสุดการส่งแล้ว

ENQ Enquiry

ENQ มีค่าเท่ากับ 05 ในตาราง ASCII เป็นอักขระที่ส่งมาจากฝ่ายรับขอให้ฝ่ายส่งส่งข้อมูลมา

NAK Negative Acknowledge

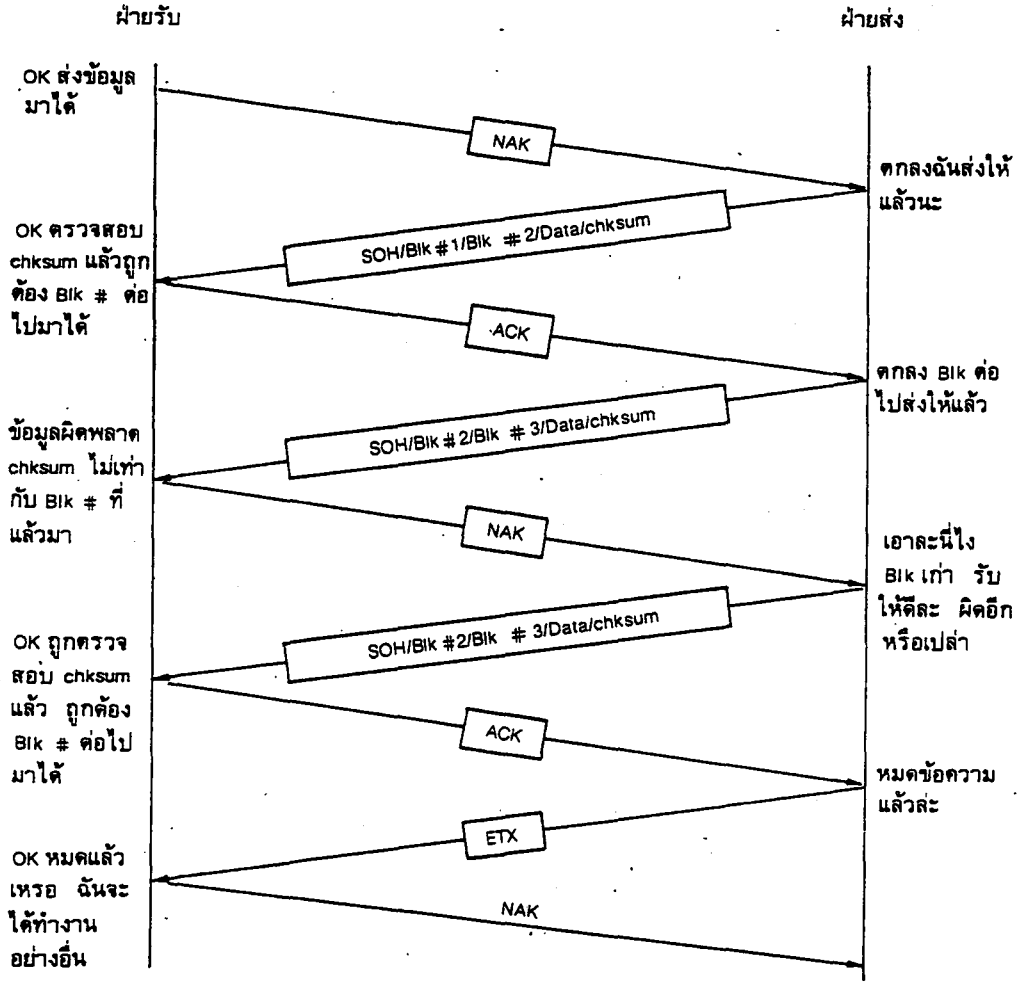
ในตารางรหัส ASCII มีค่าเท่ากับ 021 เป็นการบอกฝ่ายส่งว่าข้อมูลที่ได้รับนั้นผิดพลาด

ACK Acknowledge

เป็นการบอกฝ่ายส่งว่าข้อมูลที่ได้รับนั้นถูกต้องแล้วเป็นรหัส ASCII เท่ากับ 06

โปรโตคอลที่ใช้ใน IBM PC ส่วนมากจะเป็นโปรโตคอลที่มีชื่อว่า XMODEM ลักษณะการทำงานแสดงไว้ในรูปที่ 3.4.6

ฝ่ายส่งจะยังไม่ส่งข้อมูลจนกว่าจะได้รับ NAK จากฝ่ายรับ ฝ่ายส่งจะส่งข้อมูลออกไปโดยมีรูปแบบเริ่มด้วย SOH ตามด้วยอักขระ 2 ตัว สำหรับบอกกลุ่มของข้อมูลที่ส่งและตามด้วยส่วนเติมเต็ม 1 (1's complement) ของกลุ่มต่อไปที่จะส่ง ต่อจากนั้นก็จะเป็นข้อมูล 128 ไบท์ ตามหลังด้วยการตรวจสอบข้อผิดพลาดโดยวิธีตรวจสอบผลบวก (checksum) การ checksum คำนวณมาจากการบวกค่า ASCII ของข้อมูลที่ส่งออกไปทั้งหมด 128 ไบท์ แล้วหารด้วย 255 เศษที่เหลือก็คือค่า checksum ข้างฝ่ายรับเมื่อแยกเอา SOH และหมายเลขบล็อก (Block Number) ทั้งสองออกไปแล้วก็จะเอาข้อมูลทั้ง 128 ไบท์มารวมกันเพื่อหาค่า checksum เอา checksum ที่หาได้เปรียบเทียบกับค่าที่ได้รับ หากตรงกันก็ถือว่าข้อมูลที่ได้รับถูกต้องจึงส่งสัญญาณ ACK ไปให้ฝ่ายส่งได้รู้ว่าขณะนี้ได้รับข้อมูลไว้ถูกต้องแล้วส่งกลุ่มของข้อมูลต่อไปมาได้ ถ้าหากค่า checksum ไม่ถูกต้อง ฝ่ายรับก็จะส่ง NACK ให้ฝ่ายส่งเพื่อเป็นการบอกให้รู้ว่าข้อมูลที่ได้รับผิดพลาด ช่วยส่งกลุ่มของข้อมูลอันเก่ามาให้ทีเถอะ ฝ่ายส่งก็จะส่งข้อมูลกลุ่มเก่ามาให้ใหม่ การส่งใหม่จะดำเนินไป 9 ครั้ง หากยังคงได้รับแต่สัญญาณ NAK ฝ่ายส่งจะหยุดทำงานแสดงว่าตัวกลางการสื่อสารไม่ดี



รูปที่ 3.4.6 โปรโตคอล XMODEM

การที่ XMODEM ใช้เลขบอกกลุ่ม (Block Number) 2 ตัว (ตัวหนึ่งบอกกลุ่มที่ส่งขณะนี้ อีกตัวหนึ่งเป็นส่วนเติมเต็ม 1 ของกลุ่มต่อไป) เพื่อความแน่นอนว่ากลุ่มเดียวกันจะไม่ถูกส่งออกไปสองครั้ง ถ้าหากอักขระควบคุมการส่งเกิดสูญหายไประหว่างการส่ง ฝ่ายรับจะตรวจสอบดูว่ากลุ่มของข้อมูลที่ส่งมาเป็นกลุ่มที่ฝ่ายรับต้องการหรือไม่ ถ้าหากกลุ่มเก่าเกิดส่งมาใหม่อีกด้วยความผิดพลาด จาก ACK เป็น NACK ของฝ่ายส่ง ฝ่ายรับก็จะจับข้อมูลที่รับมาโยนทิ้งไปเมื่อทุกอย่างดำเนินไปอย่างเรียบร้อยจนสิ้นสุดแฟ้มที่จะส่ง ฝ่ายส่งก็จะส่ง ETX เป็นการบอกฝ่ายรับว่าหมดข้อความที่จะส่งแล้วละ

XMODEM เหมาะสำหรับ IBM PC เหนือโปรโตคอลชนิดอื่น 3 ข้อ คือ

1. ใช้อักขระควบคุมที่มีอยู่แล้วใน ASCII
2. สามารถจะใช้ภาษาในระดับสูงควบคุมได้ เช่น ภาษาเบสิก ภาษาปาสคาล
3. ต้องการบัฟเฟอร์สื่อสารแค่ 256 ไบท์
4. ระบบบริการข่าวสารด้วยคอมพิวเตอร์ โดยทั่วไปใช้โปรโตคอล XMODEM

ข้อดีของระบบการควบคุมการรับส่งแบบอนุกรมโดยการใช้ระบบการใช้โปรโตคอล ก็คือ

1. โปรโตคอลบางชนิดสามารถเลือกขนาดของกลุ่มข้อมูลได้

2. สามารถส่งข้อมูลที่ไม่ใช่ ASCII ได้ โดยไม่ต้องกลัวว่ารหัสนั้นจะไปทับกับรหัสควบคุมของ ASCII

3. การตรวจสอบโดยวิธี checksum มีความสามารถตรวจสอบความผิดพลาดได้ดีกว่า ใช้บิทพาริตีในอะซิงโครนัสในขณะที่บิทพาริตีสามารถให้ประสิทธิผลได้ 95 เปอร์เซ็นต์แต่ checksum สามารถให้ประสิทธิผลถึง 99.5 เปอร์เซ็นต์ หากพบการผิดพลาดด้วยบิทพาริตี ไม่ทำให้เกิดการส่งใหม่เกิดขึ้นแต่ข้อผิดพลาดจาก checksum ทำให้เกิดการส่งข้อมูลมาใหม่

3.4.7 <u>การสื่อสารแบบซิงโครนัส</u>

ข้อแตกต่างระหว่างวงจรส่งข้อมูลอนุกรมแบบซิงโครนัสและอะซิงโครนัสก็คือ ความต่อเนื่องของข้อมูลที่ส่ง ในแบบซิงโครนัสข้อมูลที่ส่งออกมาแบบต่อเนื่อง ไม่มีบิทสตาร์ทหรือบิทสต๊อป หรือแม้กระทั่งบิทพาริตีโปรโตคอลที่ใช้ในการส่งแบบซิงโครนัส จึงแตกต่างไปจากโปรโตคอลแบบอะซิงโครนัส มีโปรโตคอลหลายแบบที่ใช้ในการส่งแบบซิงโครนัสดังจะกล่าวต่อไปนี้

3.4.7.1 โปรโตคอลไบซิงก์ (Bisyn Protocal)

Bisyn ย่อมาจาก binary synchronous communication เป็นผลผลิตของบริษัท IBM Bisync เป็นโปรโตคอลในระดับอักขระซึ่งหมายความว่าอักขระแต่ละตัวมีขอบเขตที่แน่นอน แต่อักขระไม่มีสตาร์ทบิทหรือสต๊อปบิทเหมือนกับอะซิงโครนัส การซิงโคนัสกระทำกันที่จุดเริ่มต้นของการส่งข้อมูลเลยทีเดียว สถานีส่งจะส่งสัญญาณที่เรียกว่า leading pad character ไปยังสถานีรับก่อนที่จะเริ่มส่งข้อมูล ตัวนำอักษร (leading pad character) จะประกอบด้วย 0 และ 1 สลับกันเพื่อให้สถานีรับจัดสัญญาณนาฬิกาให้ตรงกัน นอกจากนั้นก่อนข้อมูลจะส่งออกมาจะต้องมีอักขระที่เรียกว่า syn ตามหลัง PAD มาก่อน และสถานีส่งจำเป็นจะต้องบอกความยาวของข้อมูลมาในกลุ่มนี้และเครื่องหมายที่เป็นตัวบอกจุดเริ่มต้นของข้อมูลมาด้วย

อักขระ syn ในโปรโตคอล Bisyn ทำหน้าที่คล้ายกับบิทเริ่มต้นในอะซิงโครนัสซึ่งทำหน้าที่ "ปลุก"สถานีรับให้ตื่นมารับข้อมูล ขณะที่สถานีรับกำลังรอรับสัญญาณจากสถานีส่ง เครื่องรับอยู่ในสถานภาพที่เรียกว่า "Hunt" บิททุกบิทที่ผ่านเข้ามาจะถูกค้นหาอักขระ syn ก่อน เมื่อได้รับอักขระ syn แล้วจึงจะเริ่มต้นนับบิทที่เข้ามาเพื่อจุดเริ่มต้นของสัญญาณ เพื่อป้องกันโอกาสที่จะเกิดความผิดพลาดอักขระ syn จะส่งมา 2 ตัวก่อนที่จะเริ่มส่งข้อมูล

3.4.7.2 โปรโตคอล SDLC และ HDLC

SDLC ย่อมาจาก Synchronous Data Link Control และ HDLC ย่อมาจาก High Level Data Link Control โปรโตคอลทั้งสองนี้ไม่เหมาะสมสำหรับมือสมัครเล่นเพราะราคาเริ่มต้นแพง แต่ทั้งสองนี้เป็นมาตรฐานโปรโตคอลที่ใช้ในงานธุรกิจสำหรับเครื่องระดับเมนเฟรม อย่างไรก็ตามไมโครคอมพิวเตอร์ยังจำเป็นที่จะต้องติดต่อกับเมนเฟรม จำเป็นอยู่ดีที่ไมโครคอมพิวเตอร์จะต้องใช้โปรโตคอลทั้งสอง จึงขอกล่าวถึงโปรโตคอลทั้งสองไว้ย่อๆ ด้วยกัน ข้อแตกต่างที่เห็นชัดจะบอกเอาไว้ในขณะที่กล่าว

ข้อมูลที่ส่งโดย SDLC/HDLC เรียกว่า Information field และฟิลด์ดังกล่าวก็คือ เลขฐานสองที่วิ่งต่อเนื่องกันมาแบบอนุกรม ขนาดของฟิลด์อาจจะมีตั้งแต่ 0 จนถึงค่าสูงสุดที่หน่วยความจำจะรับได้ สายของข้อมูลดังกล่าวจะอยู่ในรูปของบิทข้อมูล Bisyn เป็นโปรโตคอลแบบเชิงตัวอักษร ซึ่งหมายความว่าไม่มีขอบเขตระหว่างอักษร ถ้าหากข้อมูลเป็นอักขระฝ่ายรับแยกออกเอาเองเมื่อรับข้อมูลทั้งหมดแล้ว

การต่อเนื่องของข้อมูลใน SDLC/HDLC แตกต่างไปจาก Bisyn การหยุดชั่วขณะของการส่งสามารถทำได้ใน Bisyn แต่ SDLC/HDLC ไม่ยอมให้ทำอย่างนั้น ความต่อเนื่องของสายข้อมูลจะต้องต่อเนื่องไปจนสิ้นสุด Information field ถ้าหากมีการหยุดหรือเบรกเกิดขึ้น ก่อนที่จะสิ้นสุด Information field ถือว่ามีความผิดพลาดเกิดขึ้น

ใน SDLC/HDLC สถานีรับส่งแบ่งเป็น 2 ฝ่าย คือ ฝ่ายปฐมภูมิและฝ่ายทุติยภูมิ สถานีปฐมภูมิจะเป็นตัวควบคุมการเชื่อมต่อข้อมูล

รูปแบบของ SDLC และ HDLC แสดงในรูปที่ 3.4.7

รูปแบบ SDLC

Begining flag	Address	Control	Information	Frame check	Ending flag
01111110					01111110
8 bit	8 bit	8 bit		16 bit	8 bit

รูปแบบ HDLC

Begining flag	Address	Control	Packet heading	Infomation	Frame check	Ending flag

รูปที่ 3.4.7 รูปแบบของ SDLC และ HDLC

แฟล็กเริ่มต้น (Beginning flag) เป็นตัวอ้างอิงถึงตำแหน่งเริ่มต้นสำหรับฟิลด์แอดเดรส (Address field) และฟิลด์ควบคุม (control field) แฟล็กเริ่มต้นประกอบด้วย 01011110 ทำนองเดียวกันแฟล็กท้ายขบวน (Ending flag) เป็นตัวบอกว่า 16 บิทที่รับก่อนหน้านี้เป็นการตรวจสอบเฟรม ในขณะเดียวกันถ้าหากเฟรมที่ส่งต่อเนื่องกัน แฟล็กท้ายขบวน (Ending flag) จะทำหน้าที่ทั้งเป็นแฟล็กเริ่มต้นและแฟล็กท้ายขบวน จะเห็นว่าฮาร์ดแวร์ของโปรโตคอล SDLC/HDLC จะต้องคอยตรวจสอบรูปแบบของ 01111110 อยู่เสมอ เพราะถือว่าเป็นแฟล็ก เพื่อป้องกันความผิดพลาดในการตีความข้อมูลเป็นแฟล็ก SDLC/HDLC จะจัดการแทรกบิท "0" ลงในข้อมูลที่มี "1" ต่อเนื่องมากกว่า 5 บิท ฝ่ายรับจะต้องคอยแยก "0" ที่แทรกใส่เข้าไปเอาเองด้วยเหตุผลนี้เองราคาของฮาร์แวร์จึงแพง

พิจารณาฟิลด์แอดเดรส (Address field) ซึ่งมีอยู่ 8 บิท เป็นการบอกสถานีทุติยภูมิว่าต้องการส่งให้ตัวไหน ฟิลด์ควบคุม (control field) ประกอบไปด้วย 8 บิท ซึ่งจะคอยแยกการส่งข้อความเป็น 3 รูปแบบใหญ่ ๆ คือ

1. ฟอร์แมตการโอนย้ายข้อมูล
2. ฟอร์แมต supervisory
3. ฟอร์แมต non sequence

รูปแบบที่ 1 ใช้ทั่วไปในการส่งข้อมูล ในจำนวน 8 บิทนี้จะมีจำนวนเฟรมที่กำลังส่งและจำนวนเฟรมที่ได้รับอยู่ด้วย

3.4.8 <u>รหัสในการสื่อสารและการควบคุม</u>

รหัสในการสื่อสารเป็นมาตรฐานที่มีความหมายอย่างเดียวกันสำหรับผู้รับส่ง หากไม่มีมาตรฐานเสียแล้วรหัสเหล่านี้ก็ไม่มีความหมายอะไร แต่ละเครื่องที่ใช้ในการสื่อสารก็จะเข้ากันไม่ได้โดยเฉพาะเมื่อมีผู้ผลิตหลายยี่ห้อหลายสถานที่

มีรหัสที่ใช้กันอย่างแพร่หลายอยู่ 2 รหัสคือ ASCII (อ่านว่า แอสกี) ซึ่งย่อมาจาก American Standard Code for Information Interchange ซึ่งใช้กันอย่างแพร่หลายทั่วโลก อีกรหัสหนึ่งเป็นของบริษัทไอบีเอ็ม ซึ่งใช้กับเครื่องคอมพิวเตอร์ขนาดใหญ่และขนาดกลางมานาน รหัสที่ว่านั้นก็คือ EBCDIC (อ่านว่า เอบซิดิค) ซึ่งย่อมาจาก Extended Binary Coded Decimal Interchange Code อย่างไรก็ตามยังมีรหัสอีกอย่างหนึ่งที่ใช้กันมานานในงานโทรเลขคือ Baudot (อ่านว่า โบดอต)

เนื่องจาก ASCII เป็นรหัสที่ใช้กันอย่างแพร่หลายในเครื่องระดับไมโคร ซึ่งแม้แต่ IBM PC ก็ยังใช้รหัส ASCII จึงขอกล่าวถึงรหัส ASCII อย่างละเอียด

3.4.9 อักขระในชุดของแอสกี

ใน 128 ตัวแรกของรหัส ASCII ได้รับการกำหนดมาตรฐานเอาไว้ใน ANSI X3.4 ปี ค.ศ. 1977 (standard ASCII character set) เครื่องไมโครคอมพิวเตอร์เกือบทั่วโลกใช้รหัสนี้

ตารางที่ 3.4.1 เป็นตารางที่แสดงถึงรหัส 128 ตัวแรกของ ASCII ซึ่งประกอบด้วยอักขระภาษาอังกฤษ 26 ตัวใหญ่ 26 ตัวเล็ก เลขโรมัน 10 ตัว และอักขระพิเศษที่มีอยู่ตามเครื่องพิมพ์ดีดทั่วไป รหัสการสื่อสารถูกรวมเข้าไปใช้อยู่ในที่นี้ด้วย

ถึงแม้ว่ารหัส ASCII ใช้แค่ 7 บิท ส่วนมากไมโครคอมพิวเตอร์จะใช้ 8 บิท ในการเข้ารหัสอักขระ รหัสที่เป็นมาตรฐานของ ASCII จะมีบิทที่ 8 เป็น 0 นั่นคือ 128 ตัวแรกในจำนวน 256 ตัวจะเป็นรหัสมาตรฐาน ASCII ส่วนอีก 128 ตัวหลังจึงเริ่มจาก 128 ถึง 255 ซึ่งมีบิทที่ 8 เป็น 1 นั้น ไมโครคอมพิวเตอร์ส่วนมากใช้เป็นรหัสสำหรับสัญลักษณ์ทางด้านกราฟิก ซึ่งแต่ละยี่ห้อของคอมพิวเตอร์จะไม่เหมือนกัน สำหรับคนไทยนำเอา 128 ตัวหลังนี้มาเข้ารหัสเป็นอักขระภาษาไทย

ANSI (American National Standarad Institute) กำหนดมาตรฐานของรหัส ASCII ออกเป็นพวกใหญ่ ๆ ตามลักษณะการใช้งานคือ อักขระที่ทำให้เกิดข้อความที่อ่านเข้าใจได้เรียกว่า ตัวอักขระกราฟิก (Graphic character) และอักขระที่ใช้ทำให้เกิดการควบคุม เรียกว่าอักขระควบคุม (Control characters) ชื่อของพวกแรกนั้นอาจจะทำให้เข้าใจผิดได้ เนื่องจากอักขระกราฟิกก็คืออักขระที่เป็นพวกที่สามารถอ่านได้เข้าใจจริงๆ ไม่รวมถึงพวกที่จะทำให้เกิดรูปกราฟิกตามชื่อ พวกที่เป็นกราฟิกในไมโครคอมพิวเตอร์ทั่วไปควรจะอยู่ในจำพวกตัวอักขระใช้งานเฉพาะ (special extended characters)

อักขระควบคุมของ ASCII แบ่งออกเป็นกลุ่มย่อยตามการใช้งาน คือ

1. รหัสที่ใช้ควบคุมการสื่อสารข้อมูล (communication control)
2. รหัสจัดรูปฟอร์แมต (format effectors)
3. รหัสไว้ใช้แยกข้อความ (information separator)
4. รหัสควบคุมพิเศษ (special control characters)

อักขระแอสกีได้แสดงไว้ในตารางที่ 3.4.1

3.4.10 มาตรฐาน RS-232C

โดยปกติไมโครคอมพิวเตอร์จะมีพอร์ตที่เป็นแบบอนุกรม เรียกชื่อกันว่า RS 232 อยู่ในตัวเองอยู่แล้ว หลายเครื่องไม่มีมากับเครื่อง อย่างเช่น IBM PC จำเป็นจะต้องมีการ์ดที่เรียกว่าอะซิงโครนัสอะแดปเตอร์ (Asynchronous Communication Adapter) มาเสียบใส่

ตารางที่ 3.4.1 ตารางรหัส ASCII

HEX	Character	HEX	Character	HEX	Character	HEX	Character
00	NUL	20	(SPACE)	40	@	60	`
01	SOH	21	!	41	A	61	a
02	STX	22	"	42	B	62	b
03	ETX	23	#	43	C	63	c
04	EOT	24	$	44	D	64	d
05	ENQ	25	%	45	E	65	e
06	ACK	26	&	46	F	66	f
07	BEL	27	'	47	G	67	g
08	BS	28	(	48	H	68	h
09	HT	29	)	49	I	69	i
0A	LF	2A	*	4A	J	6A	j
0B	VT	2B	+	4B	K	6B	k
0C	FF	2C	,	4C	L	6C	l
0D	CR	2D	-	4D	M	6D	m
0E	SO	2E	.	4E	N	6E	n
0F	SI	2F	/	4F	O	6F	o
10	DLE	30	0	50	P	70	p
11	DC1	31	1	51	Q	71	q
12	DC2	32	2	52	R	72	r
13	DC3	33	3	53	S	73	s
14	DC4	34	4	54	T	74	t
15	NAK	35	5	55	U	75	u
16	SYN	36	6	56	V	76	v
17	ETB	37	7	57	W	77	w
18	CAN	38	8	58	X	78	x
19	EM	39	9	59	Y	79	y
1A	SUB	3A	:	5A	Z	7A	z
1B	ESC	3B	;	5B	[	7B	{
1C	FS	3C	<	5C	\	7C	|
1D	GS	3D	=	5D	]	7D	}
1E	RS	3E	>	5E	^	7E	~
1F	US	3F	?	5F	_	7F	(DEL)

พอร์ต RS 232C นี้ทำหน้าที่รับและส่งข้อมูลในแบบอนุกรมเรียกว่า Universal Asynchronous Adapter เหตุที่มีชื่อเรียกว่า RS 232C ก็เนื่องจากสมาคมผู้ผลิตอุปกรณ์อิเล็กทรอนิกส์ของอเมริกาหรือ EIA ได้กำหนดมาตรฐานของอุปกรณ์การสื่อสารแบบอนุกรมเอาไว้ภายใต้ชื่อว่า RS 232C ความจริงมาตรฐานของการส่งข้อมูลแบบอนุกรมมีหลายมาตรฐาน แต่ที่นิยมกันมากที่สุดสำหรับไมโครคอมพิวเตอร์ก็คือ RS 232C

หน้าที่สำคัญของการสื่อสารแบบอะซิงโครนัสก็คือ

รับสัญญาณ

1. เปลี่ยนสัญญาณเข้ามาแบบอนุกรมให้เป็นแบบขนาน
2. ตรวจสอบความผิดพลาดของสัญญาณที่รับ

3. ตัดสต๊อปบิทและพาริตีบิทออก

4. ส่งสัญญาณให้ซีพียูรู้ว่ารับสัญญาณไว้แล้ว

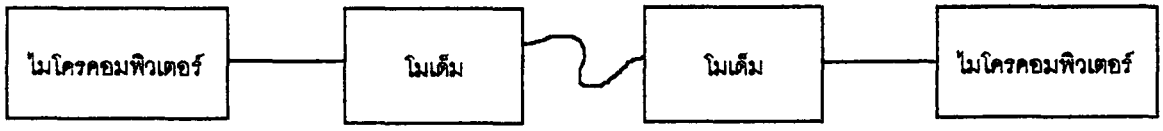
ส่งสัญญาณ

1. เปลี่ยนสัญญาณแบบขนานจากซีพียูค่อยทยอยส่งออกเป็นแบบอนุกรม

2. เพิ่มสต๊อปบิทและพาริตี

3. เพิ่มสัญญาณควบคุมโมเดมที่ต่อเชื่อม (ถ้ามี)

มาตรฐาน RS 232C ได้จัดพิมพ์ขึ้นเมื่อ ปี ค.ศ. 1969 โดยสมาคมผู้ผลิตอุปกรณ์อิเล็กทรอนิกส์แห่งสหรัฐอเมริกา RS ย่อมาจาก Recommened Standard ส่วน 232 เป็นหมายเลขบ่งบอกของมาตรฐานตัวนี้ C เป็นหมายเลขของฉบับสุดท้ายของมาตรฐานตัวนี้ จุดประสงค์ของมาตรฐานตัวนี้ก็เพื่อบรรยายคุณลักษณะของการเชื่อมต่ออุปกรณ์รับส่งข้อมูลปลายทาง (Data Terminal Equipment DTE) กับอุปกรณ์สื่อสารข้อมูล (Data Communication Equipment DCE) สำหรับผู้ใช้ไมโครคอมพิวเตอร์ DTE หมายถึง ตัวไมโครคอมพิวเตอร์ และ DCE ก็หมายถึง โมเดม อุปกรณ์อื่น ๆ เช่น เครื่องพิมพ์ที่รับสัญญาณแบบอนุกรมอาจจะเป็นได้ทั้ง DTE และ DCE ขึ้นอยู่กับผู้ผลิต ข้อแตกต่างของ DTE และ DCE จะเห็นได้จากรูปที่ 3.4.8 จากรูปนี้เราจะเห็นได้ว่า RS 232C มีส่วนสำคัญอย่างใหญ่หลวงสำหรับการสื่อสารข้อมูลระหว่างไมโครคอมพิวเตอร์

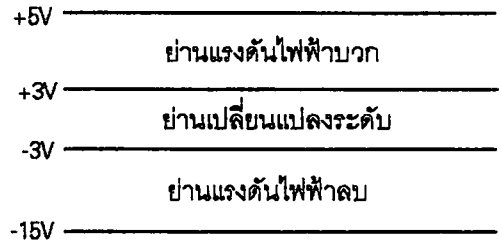


รูปที่ 3.4.8 การใช้ RS 232C เชื่อมต่ออุปกรณ์

ความจริงอีกประการหนึ่งของ RS 232C ก็คือ ความเร็วและระยะทางของการเชื่อมต่อ RS 232C สามารถเชื่อมต่อการถ่ายโอนข้อมูลได้จาก 0-20,000 บิทต่อวินาที ซึ่งเพียงพอสำหรับไมโครคอมพิวเตอร์ที่มีขนาดอัตราบอด 110 ถึง 9600 บอด ความยาวของสายเชื่อมต่อโดยสัญญาณตามมาตรฐานของ RS 232 จำกัดอยู่แค่ 50 ฟุต ซึ่งเพียงพอสำหรับการสื่อสารไมโครคอมพิวเตอร์กับอุปกรณ์รอบนอก เพื่อเป็นหลักประกันว่าข้อมูลถูกส่งออกไปอย่างถูกต้อง จำเป็นจะต้องมีข้อตกลงกันในเรื่องของสัญญาณที่ใช้ มาตรฐาน RS 232C กำหนดย่านของแรงดันไฟฟ้าในสัญญาณเพื่อสนองจุดประสงค์ข้างบน ดังแสดงในตารางที่ 3.4.2 และรูปที่ 3.4.9

ตารางที่ 3.4.2 มาตรฐานของแรงดันไฟฟ้าใน RS-232-C

แรงดันไฟฟ้า	สถานภาพลอจิก	สถานภาพของสัญญาณ	ฟังก์ชันในการควบคุม
บวก	0	สเปซ	ON
ลบ	2	มาร์ค	OFF



รูปที่ 3.4.9 ย่านของแรงดันไฟฟ้าที่ใช้ในสัญญาณ RS-232C

สำหรับไมโครคอมพิวเตอร์บางเครื่อง ใช้แต่สัญญาณลอจิกออกมาเป็นสัญญาณของ RS 232 เลย อย่างเช่น อะซิงโครนัสอะแดปเตอร์ของ IBM PC ในกรณีเช่นนี้ระยะทางของสายที่เชื่อมต่ออาจจะไปได้สั้นกว่า 50 ฟุต ดังที่กล่าวเอาไว้เนื่องจากระดับของกราวด์เปลี่ยนแปลงไป อันเนื่องจากการสูญเสียไปในความต้านทานของสาย ผู้ที่เคยใช้ IBM PC อาจจะเคยประสบกับปัญหานี้มาแล้วว่า ทำไมต่อสัญญาณ RS 232C เกินกว่า 10 ฟุต แล้วใช้งานไม่ได้ แต่อย่างไรก็ตาม RS 232 ของ IBM PC ยังมีโอกาสให้เลือกใช้ 20 มิลลิแอมแปร์ กระแสวนกลับแทนแรงดันไฟฟ้า ในทางฟิสิกส์แล้ว มาตรฐานของ RS 232C กำหนดข้อต่อแบบ DB-25 แต่ละขาของข้อต่อกำหนดไว้ดังในรูปที่ 3.4.10 อย่างไรก็ตามผู้ผลิตไมโครคอมพิวเตอร์อาจจะต้องใช้ข้อต่อชนิดอื่นที่นอกเหนือไปจาก DB-25

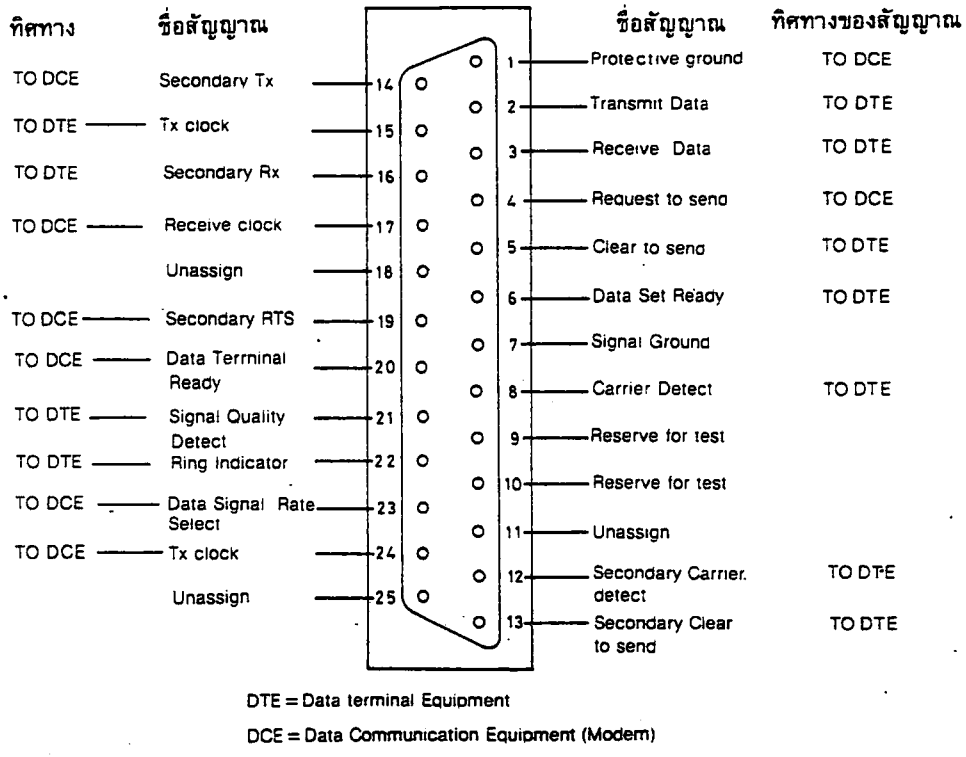
สัญญาณต่าง ๆ ถูกมอบหมายให้ทำหน้าที่ดังนี้

***Transmit Data : TD* ขาที่ 2**

เป็นสัญญาณที่ส่งออกจาก DTE (หรือตัวไมโครคอมพิวเตอร์) ไปยังโมเด็มหรือต่อเข้าโดยตรงกับไมโครคอมพิวเตอร์ตัวอื่น หรือเครื่องพิมพ์ เมื่อไม่มีสัญญาณส่งออกสถานภาพของลอจิกที่ขานี้จะมีค่าเท่ากับ "1" หรือเทียบเท่ากับสต๊อปบิท

***Receive Data : RD* ขาที่ 3**

เป็นทางของสัญญาณเข้าไปยัง DTE หรือไมโครคอมพิวเตอร์เมื่อไม่มีสัญญาณรับเข้ามา ขานี้จะมีสถานภาพทางลอจิกเป็น "1"



รูปที่ 3.4.10 การกำหนดของข้อต่อ RS 232

***Request To Send : RTS* ขาที่ 4**

ใช้สำหรับส่งสัญญาณไปยังโมเด็มหรือเครื่องพิมพ์เป็นการเรียกร้องที่จะส่งสัญญาณมาทางขา 2 สัญญาณนี้ใช้คู่กับ CTS หรือ Clear to send อุปกรณ์รับหากได้รับสัญญาณ RTS จะตรวจสอบตัวเองว่าพร้อมจะรับสัญญาณได้หรือยัง หากพร้อมที่จะรับก็ส่งสัญญาณออกไปที่สาย CTS

***Clear To Send : CTS* ขาที่ 5**

ดังอธิบายไว้ใน RTS เมื่อสัญญาณนี้อยู่ในสถานะไมทำงาน (negative voltage หรือลอจิก "1") หมายความว่า อุปกรณ์รับกำลังบอกว่าพร้อมที่จะรับข้อมูลแล้ว

***Data Set Ready : DSR* ขาที่ 6**

เมื่อสัญญาณสายนี้อยู่ในสถานะทำงาน (หรือลอจิก 0) เป็นการบอกไมโครคอมพิวเตอร์หรือฝ่ายส่งว่า โมเด็มต่อเข้ากับสายโทรศัพท์เรียบร้อยแล้วและพร้อมที่จะส่งได้แล้ว โมเด็มที่มีการหมุนหมายเลขอัตโนมัติจะส่งสัญญาณสายนี้ไปบอกให้คอมพิวเตอร์รู้ว่าต่อโทรศัพท์ได้สำเร็จแล้ว

***Signal Ground : SG* ขาที่ 7**

SG ทำหน้าที่เป็นระดับแรงดันอ้างอิงสำหรับทุกๆ สายของสัญญาณ จะมีแรงดันเป็น "0" เมื่อเทียบกับสัญญาณตัวอื่น

***Carrier Detect : CD* ขาที่ 8**

โมเด็มจะส่งสัญญาณที่อยู่ในสถานะทำงาน (ลอจิก "0") ไปบอกไมโครคอมพิวเตอร์ เมื่อได้รับสัญญาณจากโมเด็มของอีกฝ่ายหนึ่ง สัญญาณนี้จะนำไปจุด LED บอกว่าได้รับสัญญาณจากโมเด็มอีกฝ่ายหนึ่งแล้ว ไฟ LED จะอยู่บนหน้าปัทม์ของโมเด็มเอง

***Data Terminal Ready : DTR* ขาที่ 20**

คอมพิวเตอร์เปิดสัญญาณสายนี้ให้ทำงาน (ลอจิก "0") เมื่อพร้อมที่จะติดต่อกับโมเด็ม โมเด็มส่วนมากจะไม่รายงานสถานภาพของตัวเอง (CD,DSR และ CTS) ให้คอมพิวเตอร์รู้ หากคอมพิวเตอร์ไม่เปิดสัญญาณ DTR

***Ring Indicator : RI* ขาที่ 22**

สัญญาณนี้ใช้ในโมเด็มที่เป็นระบบตอบได้อัตโนมัติ (Auto-answer) สัญญาณนี้จะทำงานเมื่อมีสัญญาณกระดิ่งมา และไม่ทำงานระหว่างเสียงดังของกระดิ่ง

อาจจะสับสนระหว่างสถานภาพของลอจิกกับสถานภาพของสัญญาณ โดยปกติจะคุ้นเคยอยู่กับความรู้สึกที่ว่า เมื่อแรงดันเป็นบวก หรือสัญญาณทำงานลอจิกน่าจะเป็น "1" สำหรับสัญญาณ ที่กล่าวมานี้จะมีลักษณะตรงกันข้าม ทำไมจึงกำหนดกฎเกณฑ์ออกมาอย่างนี้ ก็เพราะว่าแต่เดิมนั้นการติดต่อกันทางโทรเลขการทำงานของสัญญาณจะต้องครบวงจรทั้งฝ่ายส่งและฝ่ายรับ เมื่อลอจิกเป็น "0" หรือขณะที่ไม่อะไรส่งควรจะมีสัญญาณทางไฟฟ้าครบวงจรอยู่ตลอดเวลา จะได้รู้ว่าวงจรไม่ขาดระหว่างทางตรงไหน ควรจะรู้ว่าวงจรครบอยู่ตลอดเวลาก็โดยการให้ค่าแรงดันที่ฝ่ายส่ง ดังนั้นจึงถือกันว่าสัญญาณไฟบวกใช้เป็นลอจิก "0"

3.4.11 <u>มาตรฐาน RS 422 และ RS 423</u>

จุดอ่อนของ EIA RS232C พอสรุปได้ 3 ประการ

1. ใช้ระดับแรงดันไฟฟ้าเลี้ยง -15 โวลต์ นอกเหนือ -5 โวลต์ ซึ่งใช้ในวงจรลอจิก

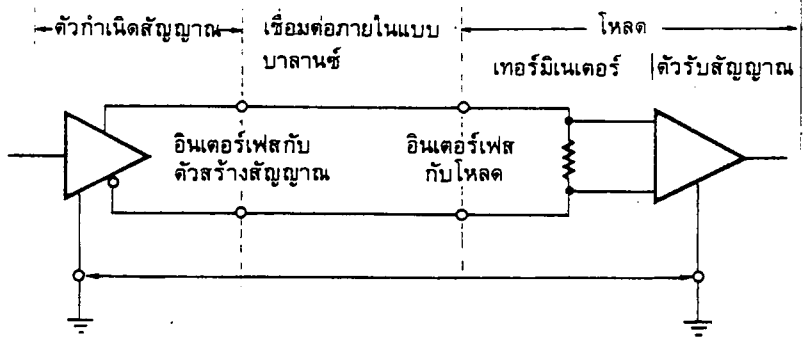
2. ค่าตัวเก็บประจุของอุปกรณ์รับสัญญาณ RS 232 รวมทั้งตัวเก็บประจุสเตย์ (Stay capacitance) ในสายจะต้องไม่มากกว่า 2500 pf สายที่รวมกันหลาย ๆ สายส่วนมากจะมีตัวเก็บประจำสเตย์ประมาณ 40-50 pf ต่อ 1 ฟุต ดังนั้นสายนี้จะต่อได้ยาวสุด 50 ฟุต ก่อนที่ค่าตัวเก็บประจุสเตย์จะมากกว่า 2500 pf ถ้าหากตัวเก็บประจุสเตย์มากกว่าที่กำหนดนี้ ช่วงเวลาการเปลี่ยนแปลงระดับของสัญญาณจะมากกว่า 4 เปอร์เซ็นต์ตามที่ยอมให้ได้ในมาตรฐาน RS 232C เมื่อเป็นเช่นนี้จะทำให้ฝ่ายรับตีความสัญญาณผิดไปจากความเป็นจริง มาร์กบิท (MARK bit) จะยาวกว่าสเปซบิท (SPACE bit) หรือ สเปซบิทยาวกว่ามาร์กบิท ขึ้นอยู่กับวงจร การตรวจสอบการผิดเพี้ยนแบบนี้เรียกว่า "Bias distrotion"

3. ปัญหาที่ 3 เป็นปัญหาทางสัญญาณไฟฟ้า ที่ EIA ไม่แก้เอาไว้ สำหรับวงจรที่ใช้ IC ก็คือ ปัญหาเรื่องกราวด์ที่แตกต่างกันตามมาตรฐาน EIA สัญญาณที่ส่งออกเทียบกับกราวด์ของเครื่องส่งเท่านั้น ถ้าหากเครื่องรับกับเครื่องส่งมีระดับแรงดันกราวด์แตกต่างกัน สมมติว่า 2 โวลต์ กระแสที่ไหลในเส้นที่เป็นกราวด์ (ขา 7) ก็จะเกิดขึ้น สมมติว่าความต้านทานของสายเป็น 0 ความต่างศักย์ที่เกิดจากกระแสกราวด์ก็จะไม่มี ความต่างศักย์ของกราวด์ระหว่างเครื่องรับและเครื่องส่งก็จะคงเท่าเดิม ระดับของสัญญาณที่ฝ่ายส่งและฝ่ายรับมองเห็นก็จะแตกต่างกัน เช่น สมมติว่าระดับของแรงดันกราวด์ต่างกัน 2 โวลต์ ฝ่ายส่งใส่แรงดันเข้าไป 5 โวลต์ ฝ่ายรับจะมองเห็นแค่ 3 โวลต์เท่านั้น ในทางกลับกันถ้าฝ่ายส่งใส่แรงดัน -5 โวลต์ ฝ่ายรับจะมองเห็นเป็น -7 โวลต์

ความต่างศักย์ของกราวด์จะคงที่ 2 โวลต์ ไม่ว่าฝ่ายส่งจะใส่แรงดันเข้าไปเท่าไรก็ตาม ผลของกราวด์ที่แตกต่างกันนี้อาจจะเกิดมาจากสถานีรับและสถานีส่งมีระบบไฟฟ้าที่กราวด์แตกต่างกันก็ได้

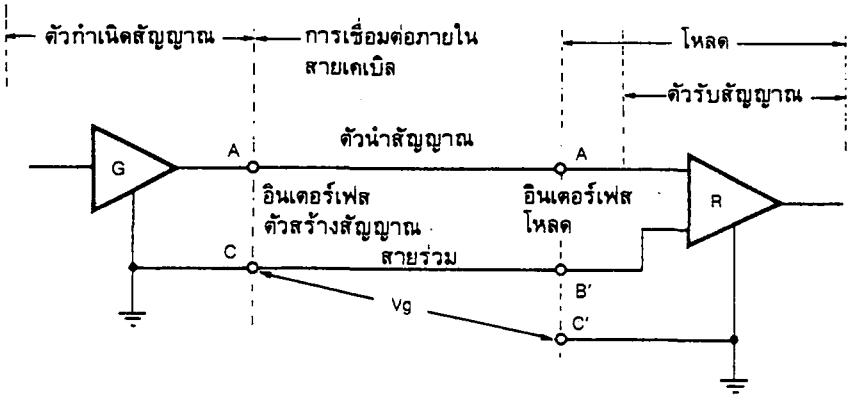
เนื่องจากตระหนักถึงปัญหาเหล่านี้ EIA ได้ออกมาตรฐานออกมาใหม่ 2 มาตรฐาน คือ RS 422 และ RS 423

ใน RS 422 แก้ปัญหา RS 232 โดยการส่งสัญญาณแบบแรงดันบาลานซ์ (Balance voltage) โดยกฎเกณฑ์สรุปไว้ในรูปที่ 3.4.11



รูปที่ 3.4.11 RS-422 วงจรการเชื่อมต่อสัญญาณดิจิตอลแบบบาลานซ์

RS 423 เป็นการแก้ปัญหาอีกวิธีหนึ่งโดยการส่งสัญญาณแบบแรงดันไม่บาลานซ์ (Unbalance voltage) กฎเกณฑ์สรุปเอาไว้ในรูปที่ 3.4.12



รูปที่ 3.4.12 RS-423 วงจรการเชื่อมต่อสัญญาณดิจิตอลแบบไม่บาลานซ์

A,C = อินเตอร์เฟสกับตัวกำเนิดสัญญาณ

A',B' = อินเตอร์เฟสกับโหลด

C' = กราวด์ของวงจรโหลด

C = กราวด์ของวงจรสร้างสัญญาณ

Vg = ศักดาแตกต่างระหว่างกราวด์ของโหลดและตัวสร้างสัญญาณ

ในเครื่อง IBM PC หากใช้อะแดปเตอร์สื่อสารข้อมูลแบบอะซิงโครนัส มีหนทางให้เลือกในการเชื่อมต่อกับอุปกรณ์อื่น 2 แบบ คือ แบบแรงดันตามกฎเกณฑ์ของ RS 232C (แต่แรงดันแค่ 0 ถึง 5 โวลต์) และแบบกระแสวนรอบ เพื่อขยายระยะทางของสายให้มากขึ้น

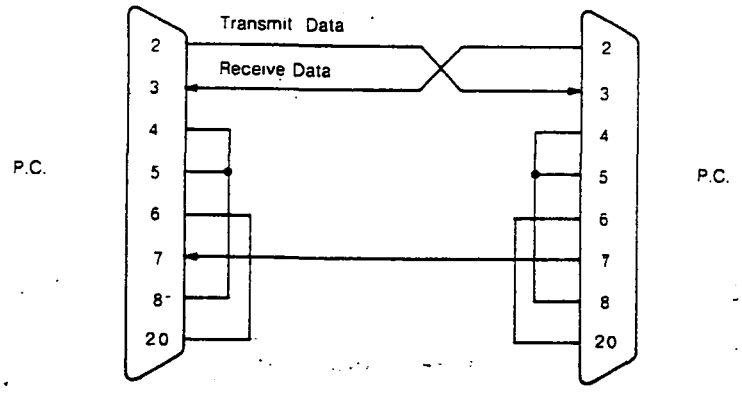
3.4.12 <u>การเชื่อมต่อกับคอมพิวเตอร์โดยตรง</u>

ในบางครั้งอาจต้องการย้ายข้อมูล จากคอมพิวเตอร์เครื่องหนึ่งไปยังอีกเครื่องหนึ่ง แน่นอนถ้าหากว่าไมโครคอมพิวเตอร์ที่ใช้มีดิสเกตต์และเป็นชนิดเดียวกัน ก็เพียงแต่ก๊อปปี้ลงบนดิสเกตต์แล้วก็นำไปใช้อีกเครื่องหนึ่ง ถ้าหากว่าเป็นคนละยี่ห้อ คนละระบบปฏิบัติการ วิธีดังกล่าวใช้ไม่ได้ผลเพราะการบันทึกลงบนแผ่นดิสค์ของคอมพิวเตอร์แต่ละยี่ห้อหรือระบบปฏิบัติการ แต่ละชนิดส่วนมากจะไปด้วยกันไม่ได้ คือ อีกเครื่องหนึ่งจะอ่านไม่ออก ขอยกตัวอย่างเช่น ถ้าสร้างแฟ้มโดย CP/M86 ของ IBM PC จะใช้ PC DOS บนเครื่องเดียวกันอ่านไม่ได้ อาจจะซื้อ IBM PC มาใหม่แล้วข้อมูลเก่าที่เคยใช้กับเครื่องเก่าจะทำยังไง

หนทางที่แก้ปัญหาข้างบน ก็คือการต่อเครื่องคอมพิวเตอร์เข้าด้วยกันโดยใช้ RS 232C แล้วถ่ายโอนข้อมูลจากเครื่องหนึ่งไปอีกเครื่องหนึ่ง วิธีการต่อแบบนี้เรียกว่า Null Modem คือ ไม่ใช้โมเด็มนั่นเอง (ในกรณีที่ระยะทางไม่เกิน 50 ฟุต)

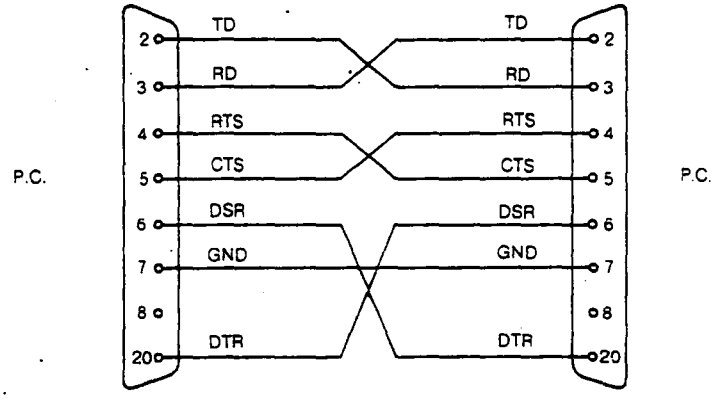
วิธีการต่อ RS 232C เข้าระหว่างเครื่องคอมพิวเตอร์โดยตรงมีอยู่หลายวิธีตามแต่ขบวนการที่จะใช้ ถ้าไม่ต้องการมีการตรวจสอบสัญญาณกันก็ต่อ RD เข้า TD ของอีกเครื่องหนึ่ง สายกราวด์ต่อถึงกันดังรูปที่ 3.4.13 ก็สามารถใช้งานถ่ายโอนข้อมูลได้แล้ว

ปกติระบบปฏิบัติการที่ให้บริการเกี่ยวกับพอร์ต RS 232 จะส่งสัญญาณ RTS ออกมาที่ขา 4 ก่อน เมื่อ CS ที่ขา 5 เป็นลอจิก "1" จึงเริ่มทำการส่งข้อมูลที่โอเปอเรเตอร์บอกให้ส่งออกไปที่ขา 2 ในกรณีที่เป็นการต่อแบบง่าย ๆ ในรูปที่ 3.4.13 จึงถือว่าเป็นการหลอกคอมพิวเตอร์โดยเอาขา 4 RTS ต่อเข้ากับขา 5 หรือ CTS เพื่อให้คอมพิวเตอร์ส่งข้อมูลได้ทันทีโดยไม่ต้องการความเรียบร้อยของฝ่ายรับ สำหรับขา 6 DSR ต่อเข้ากับขาที่ 20 DTR ก็ทำนองเดียวกัน โดยปกติคอมพิวเตอร์จะถามอุปกรณ์ที่มาต่อพ่วงกับ RS 232 ว่าพร้อมที่จะส่งแล้วยัง โดยส่งสัญญาณถามที่ขา 20 และรอคำตอบที่ขา 6



รูปที่ 3.4.13 การเชื่อมต่อแบบ RS-232 กับคอมพิวเตอร์อย่างง่าย

การต่อแบบนี้จะไม่ปลอดภัยหากว่าฝ่ายใดฝ่ายหนึ่งไม่มีความพร้อมที่จะรับหรือส่งข้อมูล จะควรมีการตรวจสอบสัญญาณที่เรียกว่า สัญญาณโต้ตอบ (hand shack) ดังรูปที่ 3.4.14 โดยใช้การตรวจสอบสัญญาณ RTS , CTS , DSR , DTR กล่าวคือเมื่อฝ่ายรับยังไม่พร้อมที่จะรับ ก็จะยังไม่มีสัญญาณ RTS ออกมา ฝ่ายส่งซึ่งถือเอา RTS ของฝ่ายรับเป็น CTS ก็จะยังไม่ส่งข้อมูลออกไป



รูปที่ 3.4.14 การเชื่อมต่อแบบ RS-232 กับคอมพิวเตอร์ แบบมีสัญญาณโต้ตอบ

บทที่ 4
การทำงานของระบบรู้จำเสียงพูด

ระบบรู้จำเสียงพูดที่ได้พัฒนาขึ้น ได้แบ่งการทำงานออกเป็น 2 ส่วนคือ

1) ส่วนแปลงและส่งผ่านข้อมูล

2) ส่วนวิเคราะห์และรู้จำเสียงพูด

โดยส่วนแปลงและส่งผ่านข้อมูล จะเป็นการทำงานบนบอร์ด ที่ใช้แปลงสัญญาณเสียงพูดให้เป็นข้อมูลดิจิตอล และส่งผ่านข้อมูลไปเก็บไว้ในไมโครคอมพิวเตอร์ โดยผ่านทางพอร์ตอนุกรม สำหรับส่วนวิเคราะห์และรู้จำเสียงพูด จะเป็นการทำงานบนไมโครคอมพิวเตอร์ ซึ่งพัฒนาด้วยภาษาปาสคาล ทำหน้าที่วิเคราะห์และรู้จำเสียงพูด โดยใช้หลักการและทฤษฎีดังรายละเอียดในบทที่ 2

4.1 การทำงานของส่วนแปลงและส่งผ่านข้อมูล

การทำงานของส่วนแปลง และส่งผ่านข้อมูล จะใช้ไมโครคอนโทรลเลอร์เบอร์ 8051 เป็นตัวควบคุมการทำงานทั้งหมดบนบอร์ด คือ การแปลงสัญญาณเสียงพูดเป็นข้อมูลดิจิตอล และส่งผ่านข้อมูลดิจิตอลให้กับไมโครคอมพิวเตอร์ โดยผ่านทางพอร์ตอนุกรม การทำงานบนบอร์ดแบ่งออกเป็น 2 ส่วน คือ ส่วนฮาร์ดแวร์ และส่วนโปรแกรมควบคุม

4.1.1 ส่วนฮาร์ดแวร์

ฮาร์ดแวร์ของบอร์ดที่ใช้ในการแปลง และส่งผ่านข้อมูล แบ่งออกเป็น 2 ส่วน คือ

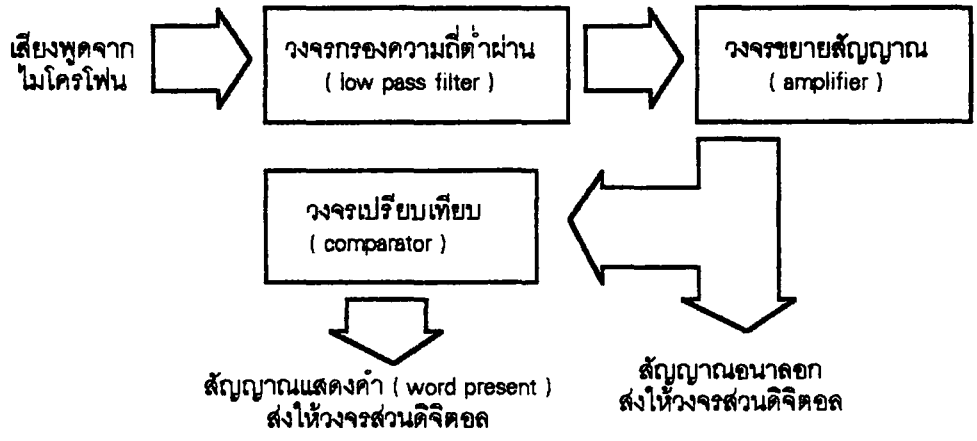
1) วงจรส่วนอนาลอก

2) วงจรส่วนดิจิตอล

โดยแต่ละส่วนมีรายละเอียดวงจร และการทำงานดังนี้

4.1.1.1 วงจรส่วนอนาลอก

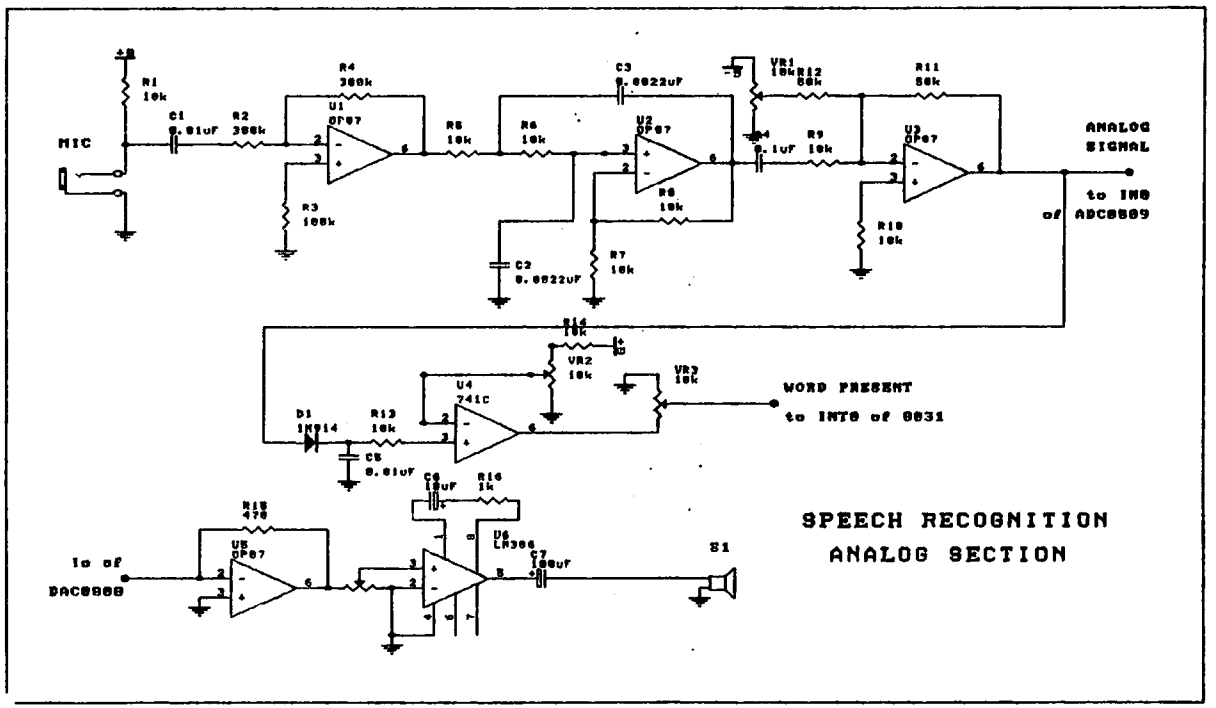
วงจรส่วนอนาลอกทำหน้าที่ปรับสภาพสัญญาณเสียงพูดให้มีความเหมาะสมก่อน แล้วจึงส่งให้วงจรส่วนดิจิตอล โดยวงจรส่วนอนาลอกจะสร้าง *สัญญาณแสดงคำ* (word present) เพื่อสั่งให้วงจรส่วนดิจิตอลเริ่มแปลงสัญญาณ บล็อกไดอะแกรมการทำงานเป็นดังรูปที่ 4.1.1



รูปที่ 4.1.1 บล็อกไดอะแกรมการทำงานของวงจรส่วนอนาลอก

การทำงานของวงจรส่วนอนาลอก เริ่มจากการรับเสียงพูดผ่านทางไมโครโฟน แล้วผ่านวงจรกรองความถี่ต่ำผ่าน (low pass filter) ซึ่งมีความถี่คัทออฟที่ 4 กิโลเฮิร์ตซ์ สัญญาณที่ได้จะส่งให้วงจรขยายสัญญาณ และยกระดับสัญญาณ เพื่อให้ได้สัญญาณที่มีการเปลี่ยนแปลงในช่วง 0 ถึง 5 โวลท์

การสร้างสัญญาณแสดงคำ (word present) ทำได้โดยการนำสัญญาณเสียงพูด ที่ออกจากส่วนขยายสัญญาณมาทำการเรียงกระแส ให้เป็นแรงดันกระแสตรง โดยใช้ไดโอด และตัวเก็บประจุ แล้วนำแรงดันนี้ป้อนให้กับวงจรเปรียบเทียบ เพื่อเปรียบเทียบกับแรงดันอ้างอิงที่กำหนดไว้ที่ค่าหนึ่ง ถ้าแรงดันที่มาจากเสียงพูด มีค่ามากกว่าแรงดันอ้างอิง จะทำให้เอาท์พุทของวงจรเปรียบเทียบเปลี่ยนสถานะจากลอจิก 0 เป็นลอจิก 1 วงจรสมบูรณ์ของส่วนอนาลอกเป็นดังรูปที่ 4.1.2

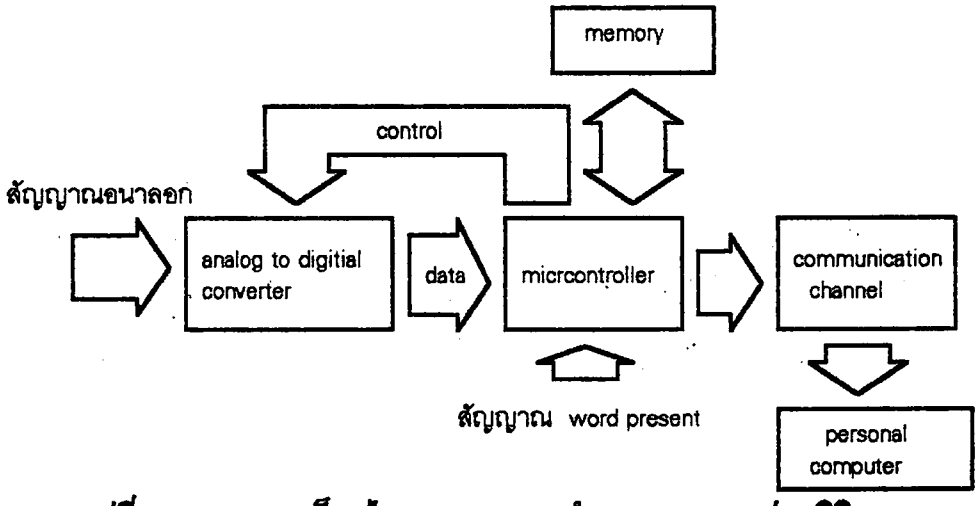


รูปที่ 4.1.2 วงจรส่วนอนาลอก

4.1.1.2 วงจรส่วนดิจิตอล

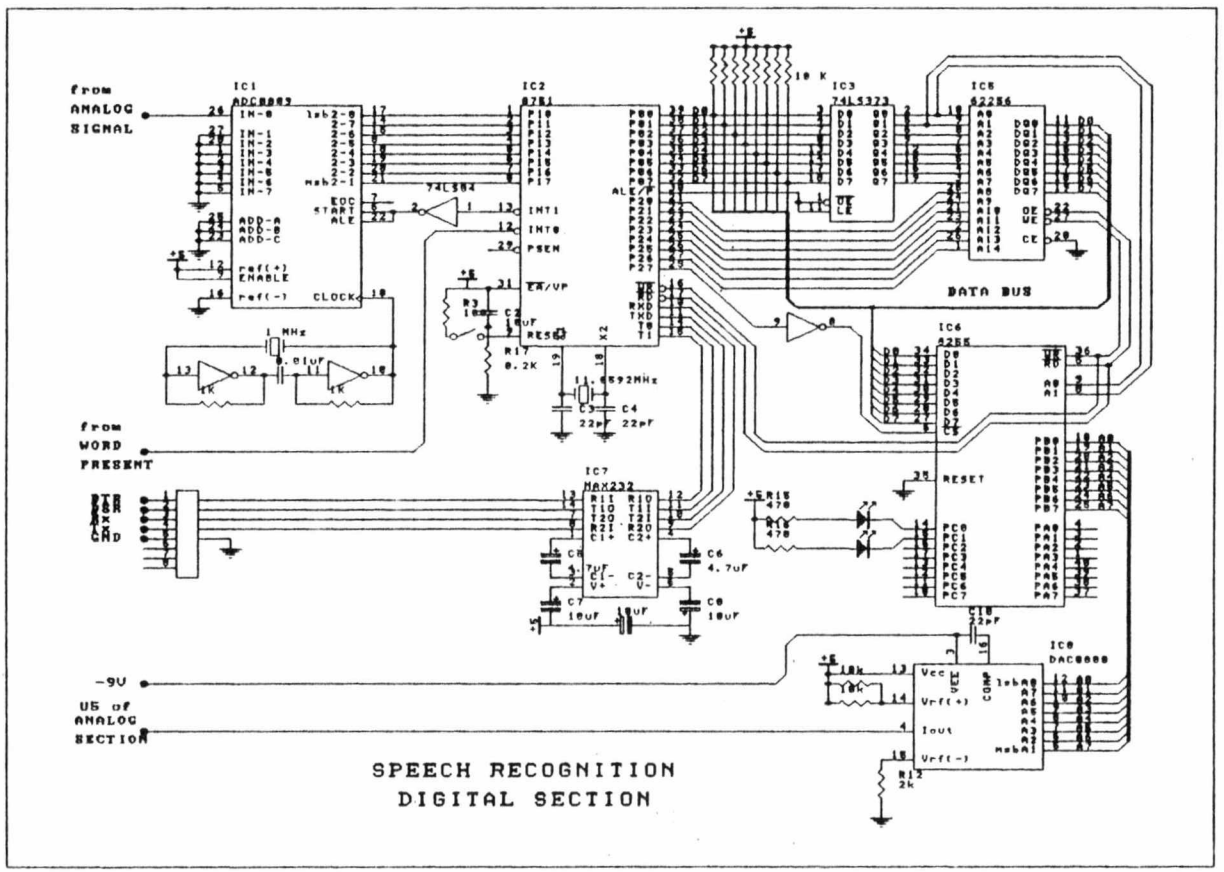
วงจรส่วนดิจิตอลทำหน้าที่แปลงสัญญาณเสียงพูด เป็นข้อมูลดิจิตอล และส่งข้อมูลดิจิตอลให้กับไมโครคอมพิวเตอร์ โดยใช้ไมโครคอนโทรลเลอร์เป็นตัวควบคุมการทำงาน บล็อกไดอะแกรมการทำงานเป็นดังรูปที่ 4.1.3

การแปลงสัญญาณเสียงพูดเป็นข้อมูลดิจิตอล จะใช้ไอซี ADC 0809 เป็นตัวแปลงสัญญาณ โดยใช้อัตราสุ่มตัวอย่าง (sampling rate) เท่ากับ 8 กิโลเฮิร์ตซ์ และเก็บข้อมูลภายในเวลา 1 วินาที เมื่อแปลงเสร็จแล้ว จะเก็บข้อมูลไว้ในหน่วยความจำบนบอร์ดชั่วคราวเพื่อรอส่งไปเก็บหรือวิเคราะห์ในไมโครคอมพิวเตอร์



รูปที่ 4.1.3 แสดงบล็อกไดอะแกรมการทำงานของวงจรส่วนดิจิตอล

การส่งผ่านข้อมูลให้กับไมโครคอมพิวเตอร์ จะใช้พอร์ตอนุกรมของไมโครคอนโทรลเลอร์ ซึ่งมีรายละเอียดการใช้งานในบทที่ 2 โดยใช้มาตรฐาน RS-232 และใช้ไอซีเบอร์ MAX 232 เป็นตัวขับ การส่งข้อมูลจะใช้อัตราการส่งเท่ากับ 19,200 บิทต่อวินาที และมีสายสัญญาณที่ใช้ตรวจสอบความพร้อมของการรับ-ส่งข้อมูล 2 เส้น คือ DSR (data set ready) และ DTR (data terminal ready) โดยสัญญาณ DSR เป็นสัญญาณบอกความพร้อมในการส่งข้อมูลของบอร์ด และสัญญาณ DTR เป็นสัญญาณบอกความพร้อมในการรับข้อมูลของไมโครคอมพิวเตอร์ วงจรสมบูรณ์ของส่วนดิจิตอลเป็นดังรูปที่ 4.1.4



รูปที่ 4.1.4 วงจรส่วนดิจิตอล

4.1.2 <u>ส่วนโปรแกรมควบคุม</u>

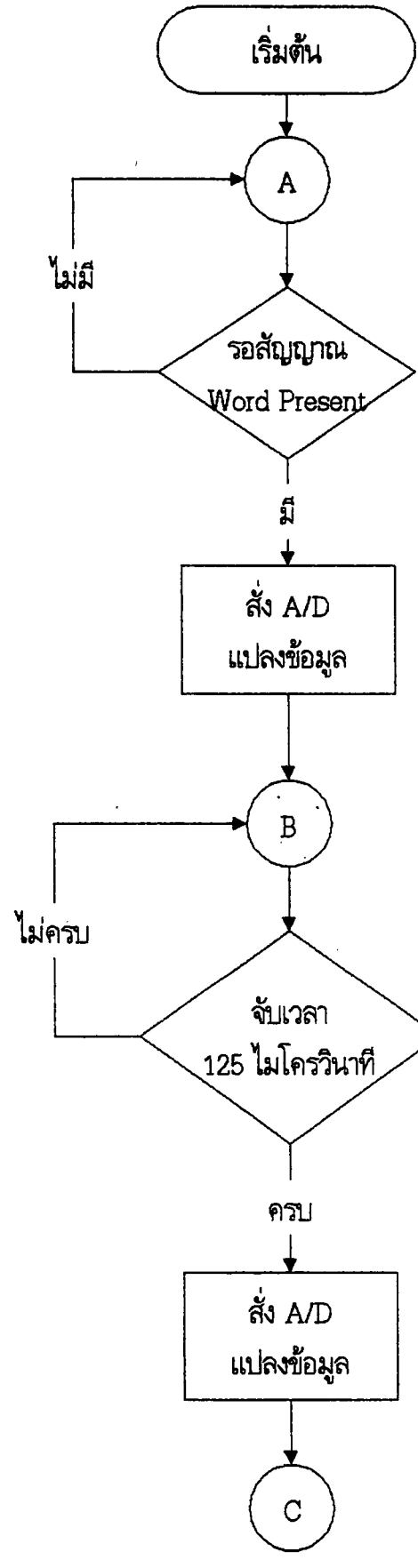
ส่วนของโปรแกรมควบคุมแบ่งหน้าที่ออกเป็น 2 ส่วน คือ

1) โปรแกรมควบคุมการแปลงข้อมูล

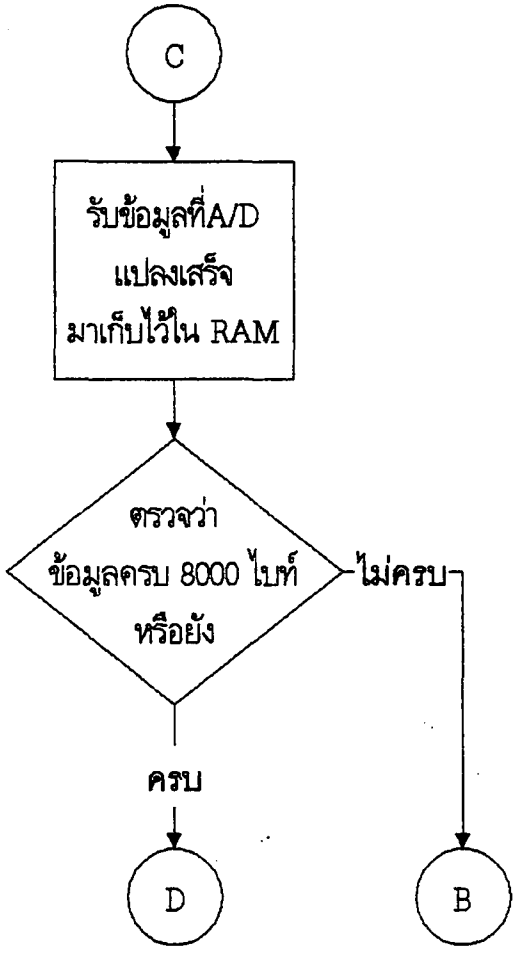
2) โปรแกรมควบคุมการรับ-ส่งข้อมูล

4.1.2.1 โปรแกรมควบคุมการแปลงข้อมูล

ทำหน้าที่แปลงสัญญาณเสียงพูดเป็นข้อมูลดิจิตอล โดยกำหนดอัตรากการสุ่มตัวอย่างเท่ากับ 8 กิโลเฮิร์ตซ์ หรือทำการแปลงข้อมูลทุกๆ 125 ไมโครวินาที ซึ่งจะใช้วงจรไทม์เมอร์ภายในไมโครคอนโทรลเลอร์เป็นตัวจับเวลา ขั้นตอนการทำงานของโปรแกรมคือโปรแกรมจะตรวจสอบบสถานะของสัญญาณ แสดงคำ จนกว่าจะมีสถานะลอจิกเป็น '1' เมื่อพบว่ามีสัญญาณแสดงคำมาแล้ว โปรแกรมจะสั่งให้ไอซี ADC 0809 ทำการแปลงข้อมูลโดยส่งพัลส์ไปที่ขา START และ ALE ทุกๆ ช่วงเวลา 125 ไมโครวินาที เป็นเวลา 1 วินาที และเก็บข้อมูลที่ได้ไว้ในหน่วยความจำชั่วคราว เพื่อรอส่งให้กับไมโครคอมพิวเตอร์ แผนภาพการทำงานของโปรแกรมเป็นดังรูปที่ 4.1.5



รูปที่ 4.1.5 แผนภาพการทำงานของโปรแกรมควบคุมการแปลงข้อมูล



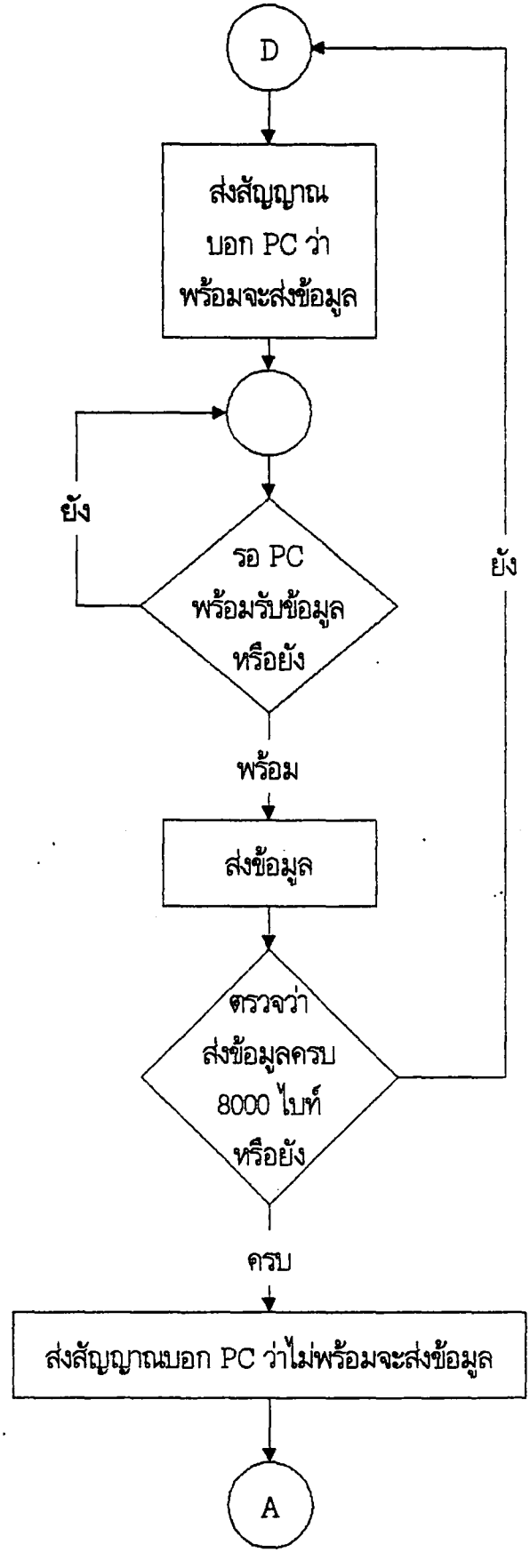
รูปที่ 4.1.5 แผนภาพการทำงานของโปรแกรมควบคุมการแปลงข้อมูล (ต่อ)

4.1.2.2 โปรแกรมควบคุมการรับ-ส่งข้อมูล

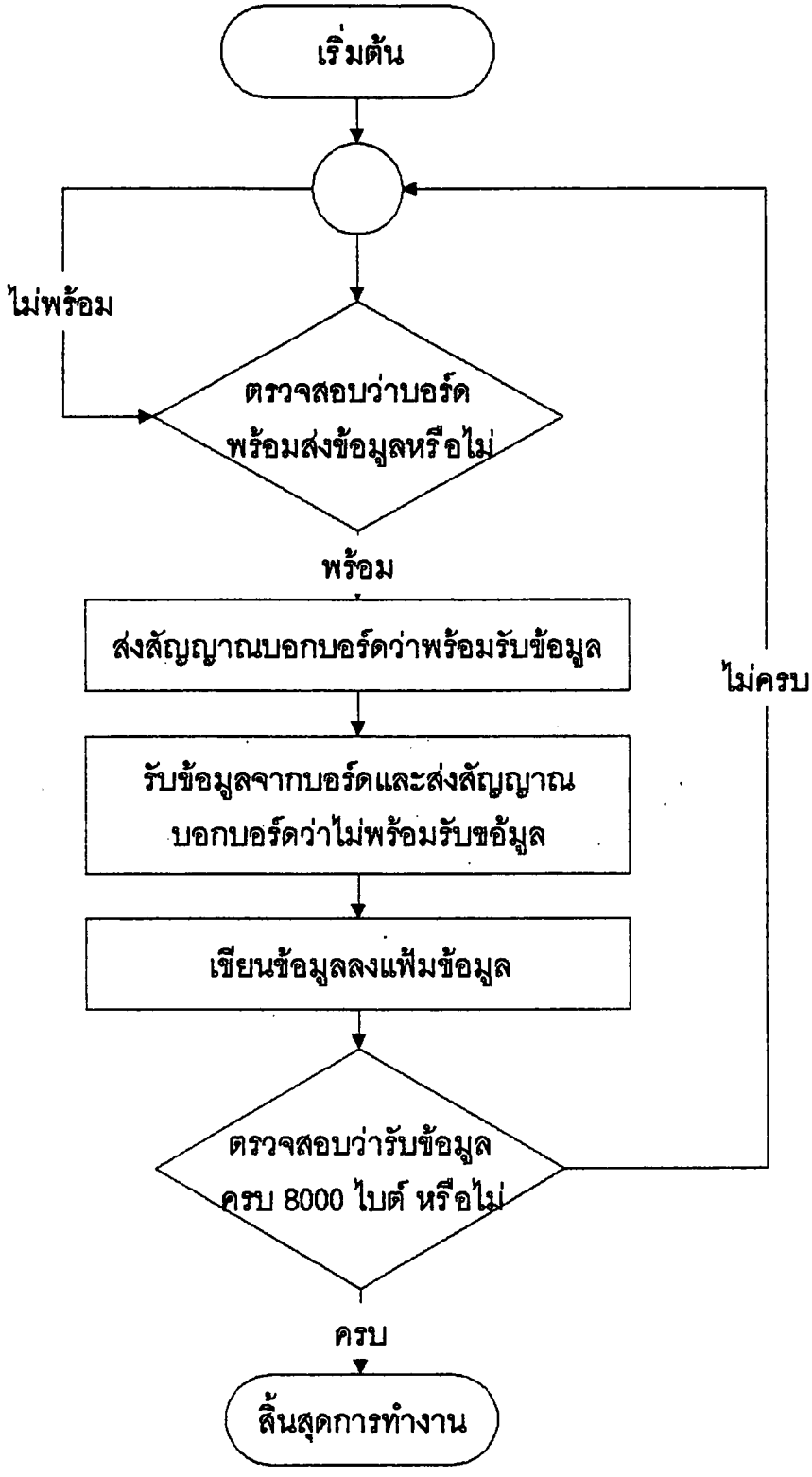
โปรแกรมที่ใช้ควบคุมการรับ-ส่งข้อมูลแบ่งออกเป็น 2 ส่วน คือ โปรแกรมที่พัฒนาด้วยภาษาเครื่องของไมโครคอนโทรลเลอร์ 8051 ซึ่งทำหน้าที่ควบคุมการส่งข้อมูล และโปรแกรมที่พัฒนาด้วยภาษาปาสคาล ซึ่งทำหน้าที่ควบคุมการรับข้อมูลบนไมโครคอมพิวเตอร์

1. โปรแกรมควบคุมการส่งข้อมูล มีการทำงานคือโปรแกรมจะส่งสัญญาณบอกไมโครคอมพิวเตอร์ ว่าพร้อมที่จะส่งข้อมูล โดยให้สัญญาณ DSR เป็นลอจิก '0' หลังจากนั้นจะรอสัญญาณตอบกลับจากไมโครคอมพิวเตอร์ ว่าพร้อมที่จะรับข้อมูล โดยตรวจสอบสัญญาณ DTR จนกว่าจะมีลอจิกเป็น '0' เมื่อไมโครคอมพิวเตอร์พร้อมที่จะรับข้อมูล โปรแกรมจะส่งข้อมูลให้กับไมโครคอมพิวเตอร์ ครั้งละ 1 ไบท์ จนกว่าจะครบ 8000 ไบท์ โดยการส่งข้อมูลแต่ละไบท์จะมีการตรวจสอบความพร้อม ในการรับข้อมูลของไมโครคอมพิวเตอร์ทุกครั้งเพื่อป้องกันการสูญหายของข้อมูล แผนภาพการทำงานของโปรแกรมเป็นดังรูปที่ 4.1.6

2. โปรแกรมควบคุมการรับข้อมูล มีการทำงานคือโปรแกรมจะตรวจสอบความพร้อมในการส่งข้อมูลของบอร์ด โดยการตรวจสอบสัญญาณ DSR จนกว่าจะมีสถานะเป็นลอจิก 0 'เมื่อพบว่าสัญญาณ DSR มีลอจิก 0 แล้วจะส่งสัญญาณบอกบอร์ดว่าพร้อมที่จะรับข้อมูล โดยทำให้สัญญาณ DTR เป็นลอจิก 0 หลังจากนั้นโปรแกรมจะรอรับข้อมูลจากบอร์ดโดยการตรวจสอบสถานะของ บัฟเฟอร์ รับข้อมูลว่ามีข้อมูลหรือไม่ ถ้าพบว่ามีข้อมูลอยู่ในบัฟเฟอร์โปรแกรมจะอ่านข้อมูลจากบัฟเฟอร์แล้วเขียนลงแฟ้มข้อมูล โปรแกรมจะทำงานตามขั้นตอนดังกล่าวจนกระทั่งรับข้อมูลครบ 8000 ไบท์ แผนภาพการทำงานโปรแกรมเป็นดังรูปที่ 4.1.7



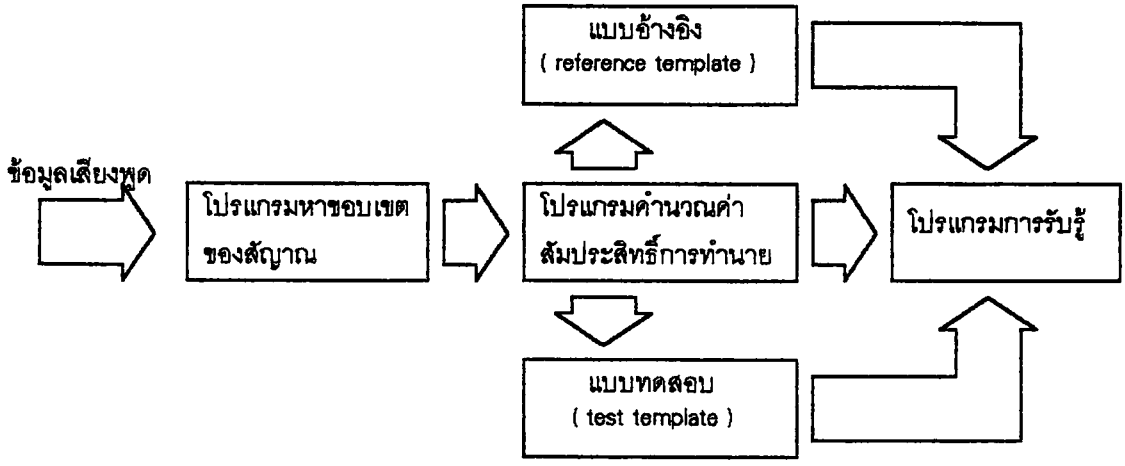
รูปที่ 4.1.6 แผนภาพการทำงานของโปรแกรมควบคุมการส่งข้อมูล



รูปที่ 4.1.7 แผนภาพการทำงานของโปรแกรมรับข้อมูลบนไมโครคอมพิวเตอร์

4.2 การทำงานส่วนวิเคราะห์และรู้จำเสียงพูด

ส่วนวิเคราะห์และรู้จำเสียงพูด เป็นโปรแกรมซึ่งพัฒนาด้วยภาษาปาสคาล โดยใช้ทฤษฎีตามที่ได้กล่าวไว้ในบทที่ 2 แผนภาพการทำงานของส่วนวิเคราะห์และรู้จำเสียงพูดเป็นดังรูปที่ 4.2.1 และการทำงานจะแบ่งเป็นส่วนๆ ดังนี้



รูปที่ 4.2.1 บล็อคไดอะแกรมการทำงานของส่วนวิเคราะห์และรู้จำเสียงพูด

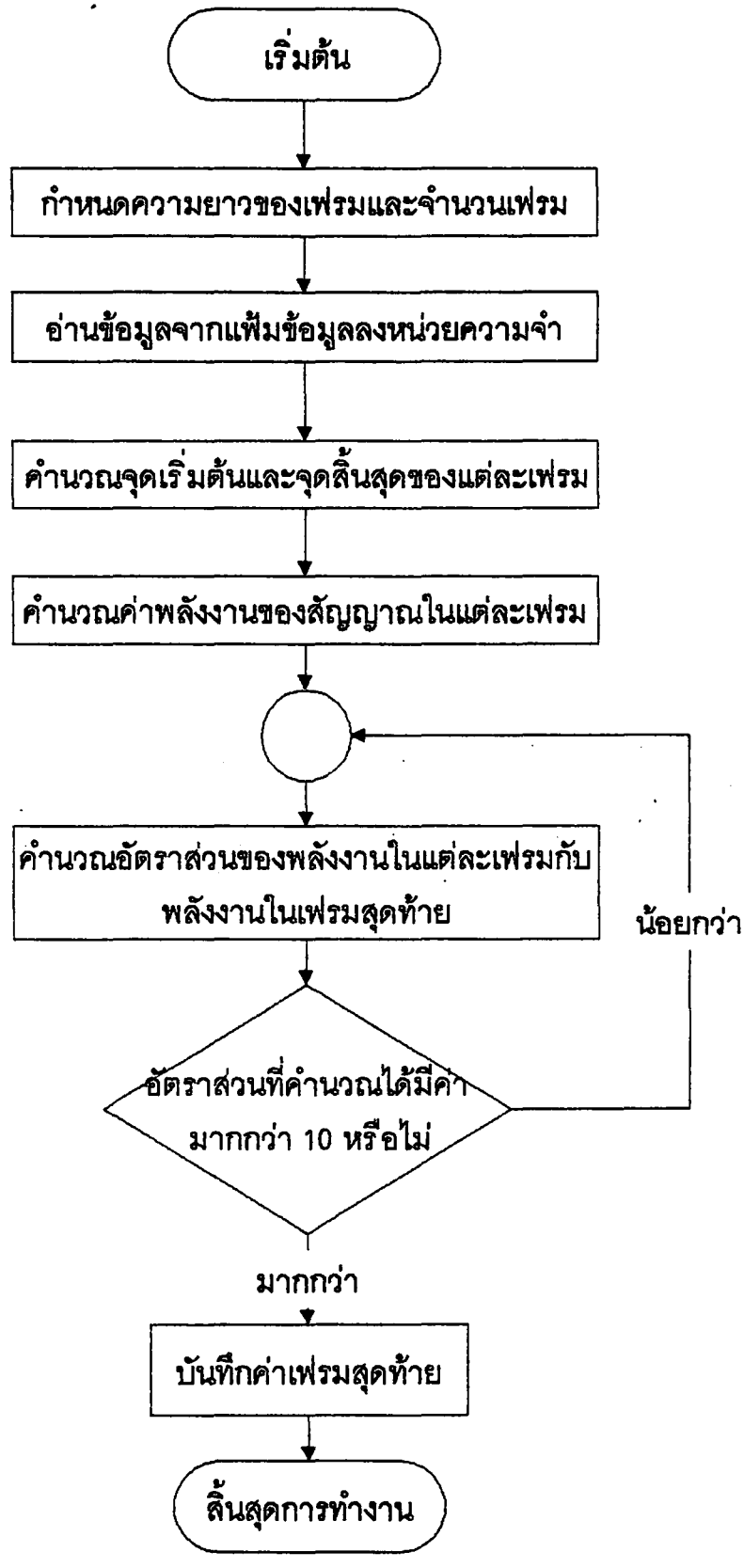
1) โปรแกรมหาขอบเขตของสัญญาณ ทำหน้าที่ในการตรวจสอบหาขอบบเขตของสัญญาณ และนำมาเฉพาะส่วนของสัญญาณที่ได้หาขอบเขตแล้วไปวิเคราะห์

2) โปรแกรมหาสัมประสิทธิ์การทำนาย ทำหน้าที่หาสัมประสิทธิ์การทำนาย ของสัญญาณเสียงที่ได้ผ่านการหาขอบเขตแล้ว เพื่อใช้เป็นแบบอ้างอิงหรือแบบทดสอบตามต้องการ

3) โปรแกรมการรู้จำ ทำหน้าที่รู้จำเสียงพูดโดยอาศัยสัมประสิทธิ์การทำนายที่ได้จากโปร---แกรมการหาสัมประสิทธิ์การทำนาย

4.2.1 โปรแกรมหาขอบเขตของสัญญาณ

เนื่องจากสัญญาณที่รับเข้ามาจากไมโครโฟน อาจมีส่วนของสัญญาณรบกวนก่อนและต่อท้ายเสียงพูดจริง จึงต้องกำจัดสัญญาณรบกวนทั้ง 2 ส่วนออกไป เพื่อให้ได้ข้อมูลของเสียงพูดที่ต้องการจริงๆ การกำจัดสัญญาณรบกวนก่อนเสียงพูดจริง จะใช้วงจรทางฮาร์ดแวร์ ที่เรียกว่าวงจรแสดงคำ (word present) สำหรับสัญญาณต่อท้ายเสียงพูดจริง จะถูกกำจัดด้วยโปรแกรมทางซอฟท์แวร์ คือ โปรแกรมหาขอบเขตของสัญญาณ ซึ่งพิจารณาจากพลังงานของสัญญาณ ดังรายละเอียดในบทที่ 2 โดยมีแผนภาพการทำงานดังรูปที่ 4.2.2 และการทำงานของโปรแกรมเป็นดังนี้



รูปที่ 4.2.2 แผนภาพการทำงานของโปรแกรมหาขอบเขตสัญญาณ

1) กำหนดให้ความยาวของเฟรม และจำนวนของเฟรมเป็น 160 ตัวอย่าง และ 50 เฟรม ตามลำดับ โดยความยาวของเฟรมจะเป็นค่าเดียวกับในโปรแกรมหาสัมประสิทธิ์การทำนาย

2) อ่านข้อมูลของคำที่ต้องการจากแฟ้มข้อมูลเสียงพูดที่ต้องการลงหน่วยความจำ

3) คำนวณจุดเริ่มต้น และจุดสิ้นสุดของแต่ละเฟรม

4) คำนวณค่าพลังงานของสัญญาณในแต่ละเฟรม

5) คำนวณหาอัตราส่วนของพลังงานในแต่ละเฟรม กับพลังงานในเฟรมสุดท้าย จุดสิ้นสุดของสัญญาณ คือ เฟรมที่มีอัตราส่วนมากกว่าค่าที่กำหนดไว้ค่าหนึ่ง โดยในโปรแกรมที่กำหนดมีค่าเท่ากับ 10

6) ส่งข้อมูลเสียงพูดที่หาขอบเขตแล้วให้กับโปรแกรมหาสัมประสิทธิ์การทำนาย

4.2.2 <u>โปรแกรมหาสัมประสิทธิ์การทำนาย</u>

เป็นโปรแกรมที่ใช้ในการคำนวณหาสัมประสิทธิ์การทำนาย โดยใช้วิธีออโตคอรีเลชัน ดังรายละเอียดในบทที่ 2 และสามารถกำหนดได้ว่าจะใช้ผลลัพธ์ที่ได้เป็นแบบทดสอบ (test template) หรือแบบอ้างอิง(reference template) แผนภาพการทำงานของโปรแกรมเป็นดังรูปที่ 4.2.3 และการทำงานของโปรแกรมเป็นดังนี้

1) กำหนดพารามิเตอร์ที่ใช้ในการคำนวณ คือ ออร์เดอร์ของการทำนาย โดยกำหนดให้มีค่าเท่ากับ 6

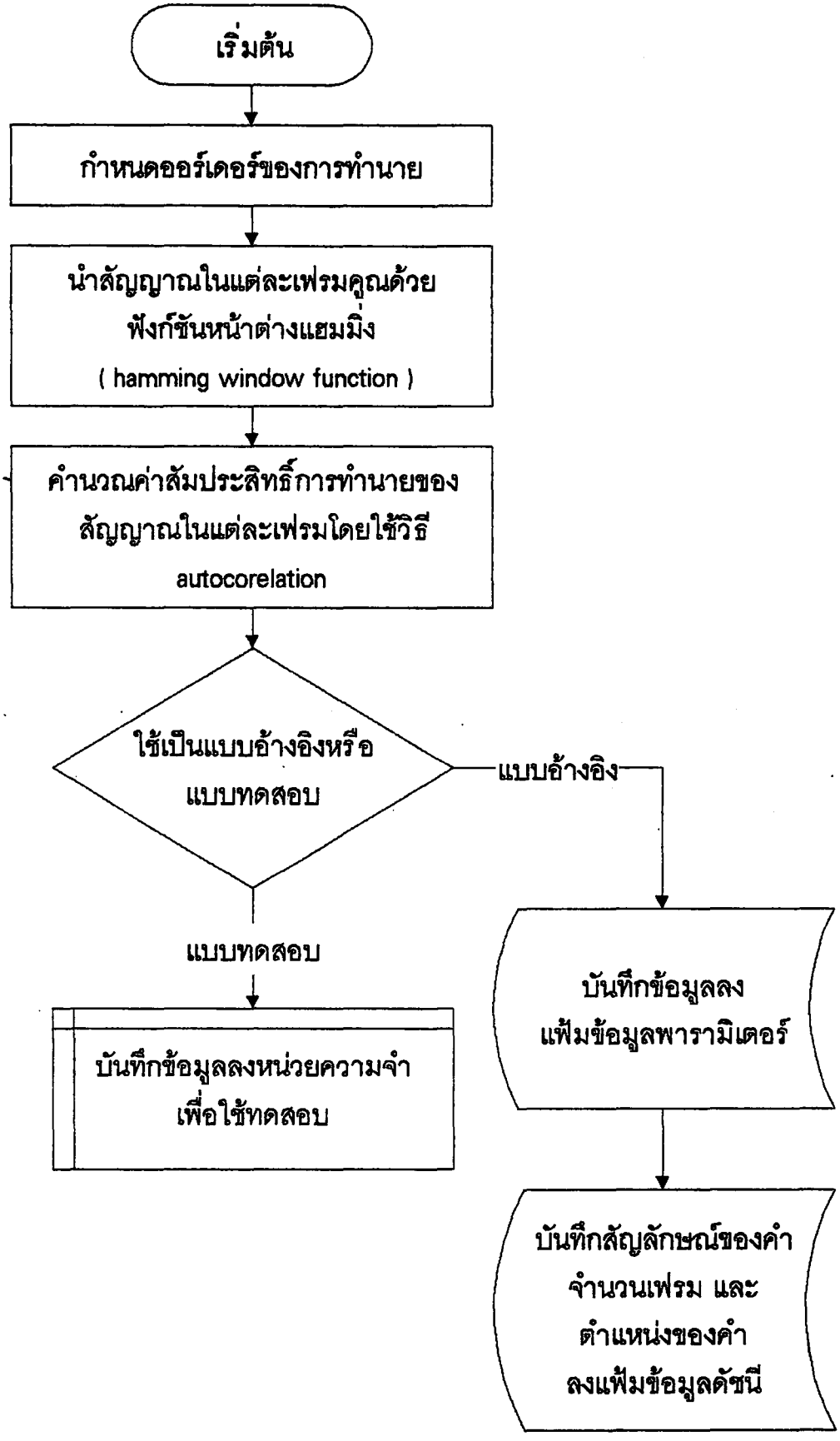
2.) กำหนดว่าจะเก็บผลลัพธ์ที่ได้เป็นแบบอ้างอิงหรือแบบทดสอบ ถ้าเป็นแบบอ้างอิงจะเก็บผลลัพธ์ที่ได้ลงแฟ้มข้อมูลซึ่งเรียกว่าแฟ้มข้อมูลพารามิเตอร์ ถ้าเป็นแบบทดสอบจะเก็บผลลัพธ์ที่ได้ลงหน่วยความจำเพื่อใช้ในการทดสอบ

4) นำสัญญาณที่ได้จากโปรแกรมหาขอบเขตของสัญญาณในแต่ละเฟรมคูณด้วยฟังก์ชันหน้าต่างแฮมมิ่ง (hamming window function)

3) คำนวณค่าสัมประสิทธิ์การทำนายของสัญญาณในแต่ละเฟรม โดยเรียกใช้โปรแกรมย่อย CALCOEFF

4) บันทึกผลลัพธ์ของการคำนวณลงแฟ้มข้อมูลพารามิเตอร์ในกรณีที่กำหนดให้เป็นแบบอ้างอิง หรือ เก็บลงหน่วยความจำในกรณีที่กำหนดให้เป็นแบบทดสอบ

5) ถ้ากำหนดให้ผลลัพธ์ที่ได้เป็นแบบอ้างอิง โปรแกรมจะทำการบันทึกข้อมูลที่เป็นรายละเอียดของคำ คือ สัญลักษณ์ของคำ จำนวนเฟรมของคำนั้น และตำแหน่งของคำในแฟ้มข้อมูล ลงในแฟ้มข้อมูลดัชนี



รูปที่ 4.2.3 แผนภาพการทำงานของโปรแกรมคำนวณค่าสัมประสิทธิ์การทำนาย

4.2.3 โปรแกรมการรู้จำ

เป็นโปรแกรมที่ใช้ในการรู้จำเสียงพูด โดยทำการเปรียบเทียบพารามิเตอร์ ของแบบทดสอบ (test template) กับแบบอ้างอิง (reference template) ทีละคำโดยใช้การเปรียบเทียบแบบไดนามิก โปรแกรมมิ่ง ดังรายละเอียดในบทที่ 2 ผลลัพธ์ที่ได้จะเป็นระยะทาง ระหว่างแบบทดสอบ กับแบบอ้างอิงแต่ละคำ จากระยะทางที่ได้จะนำไปผ่านกฎการตัดสินใจ เพื่อเลือกคำที่ถูกต้อง และแสดงสัญลักษณ์ที่ได้จากการรู้จำ แผนภาพการทำงานของโปรแกรมรู้จำเสียงพูดเป็นดังรูปที่ 4.2.4 และขั้นตอนการทำงานของโปรแกรมเป็นดังนี้

1)กำหนดพารามิเตอร์ที่ใช้ในการคำนวณ คือ ความกว้างของหน้าต่างการปรับตัว (adjust window) ซึ่งในโปรแกรมกำหนดให้มีค่าเท่ากับ 6

2) กำหนดแฟ้มข้อมูลที่ใช้เป็นแบบอ้างอิง

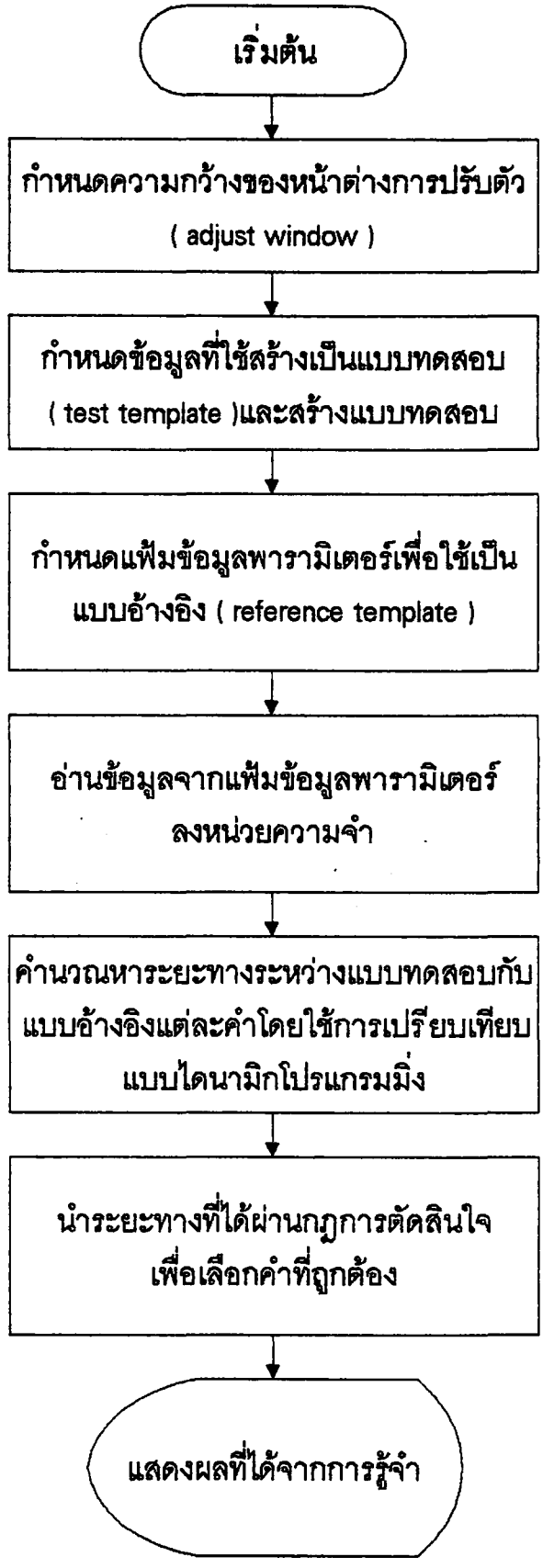
3) กำหนดข้อมูลที่จะใช้สร้างแบบทดสอบ โดยสามารถกำหนดได้ว่า จะนำข้อมูลจากแฟ้มข้อมูลเสียงพูดมาสร้างเป็นแบบทดสอบ หรือจะนำข้อมูลจากบอร์ด ซึ่งเป็นเสียงพูดในขณะทดสอบมาสร้างเป็นแบบทดสอบ

4) สร้างแบบทดสอบโดยเรียกใช้โปรแกรมหาสัมประสิทธิ์การทำนาย

5) อ่านข้อมูลแฟ้มข้อมูลพารามิเตอร์ลงหน่วยความจำเพื่อใช้เป็นแบบอ้างอิง

6) เรียกใช้โปรแกรมย่อย DISTANCE เพื่อหาระยะทางระหว่างแบบบทดสอบ กับแบบบอ้างอิงแต่ละคำ

7) นำระยะทางที่ได้ผ่านกฎการตัดสินใจ เพื่อเลือกคำที่ถูกต้องและแสดงสัญลักษณ์ของคำนั้น



รูปที่ 4.2.4 แผนภาพการทำงานของโปรแกรมรู้จำเสียงพูด

บทที่ 5
ผลการทดลอง

โครงงานพิเศษระบบรู้จำเสียงพูดอย่างง่าย ได้แบ่งการทดลองออกเป็น 2 ส่วน ดังนี้

1. การทดลองส่วนหาขอบเขตสัญญาณเสียงพูด
2. การทดลองส่วนรู้จำเสียงพูด

ผลการทดลองทั้ง 2 ส่วน จะนำเสนอออกมาในรูปของกราฟ ซึ่งต่อการตีความ และสรุปผล

5.1 ผลการทดลองส่วนหาขอบเขตสัญญาณเสียงพูด

ผลการทดลองในส่วนนี้จะแบ่งเป็น 2 ส่วน คือ ส่วนหาขอบเขตสัญญาณที่ถูกต้อง และส่วนหาขอบเขตสัญญาณที่ผิดพลาด

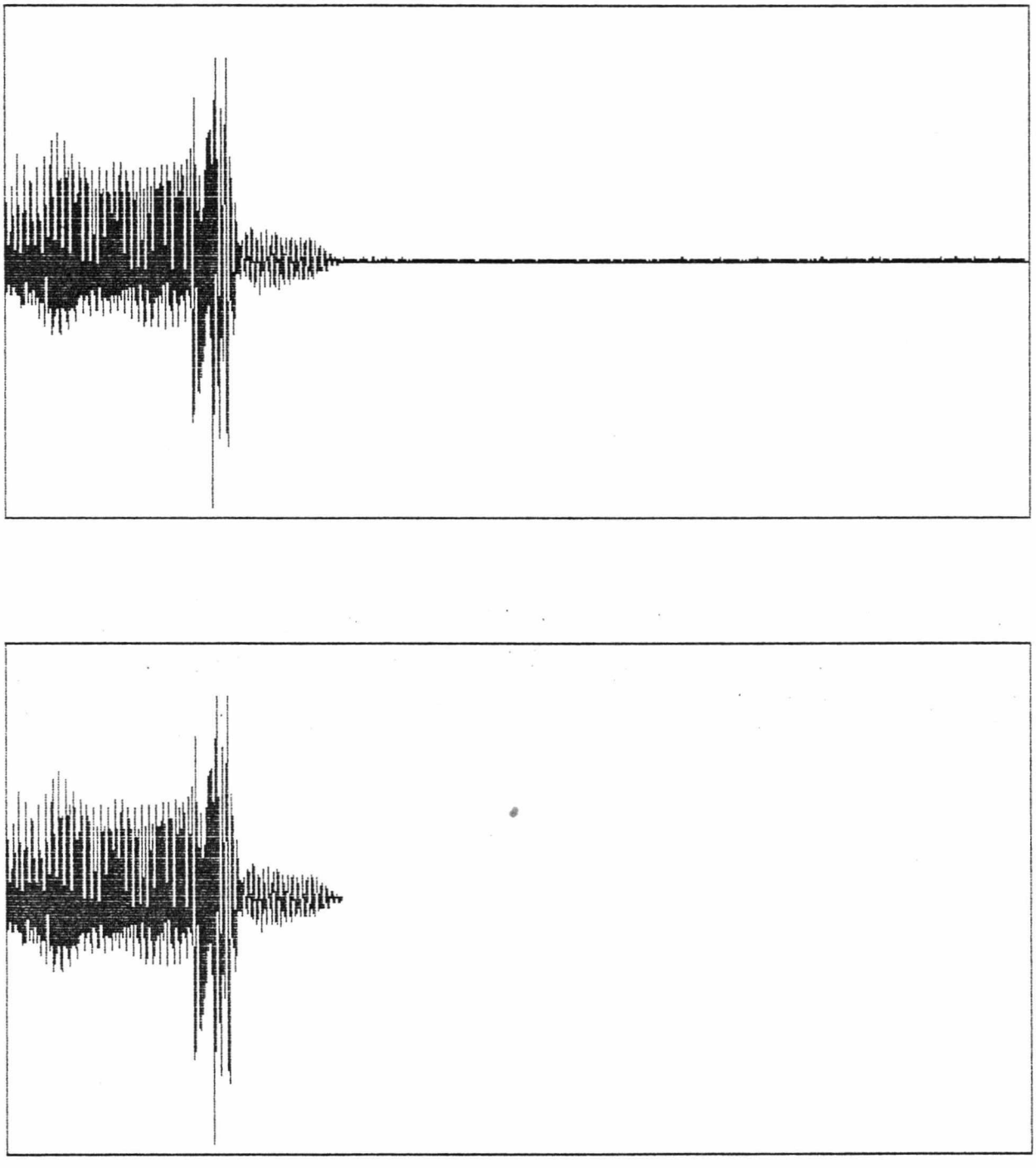
แต่ละคำจะแสดงกราฟ 2 รูป รูปบนคือ รูปของสัญญาณตลอด 1 วินาที รูปล่างคือรูปที่ผ่านการหาขอบเขตสัญญาณแล้ว

ขอบเขตสัญญาณที่ถูกต้องได้แสดงทั้ง 10 คำ คือ 0-9 ดังแสดงในรูปที่ 5.1.1-5.1.10 ส่วนขอบเขตสัญญาณที่ผิดพลาด ได้แสดงในรูปที่ 5.1.11-5.1.13

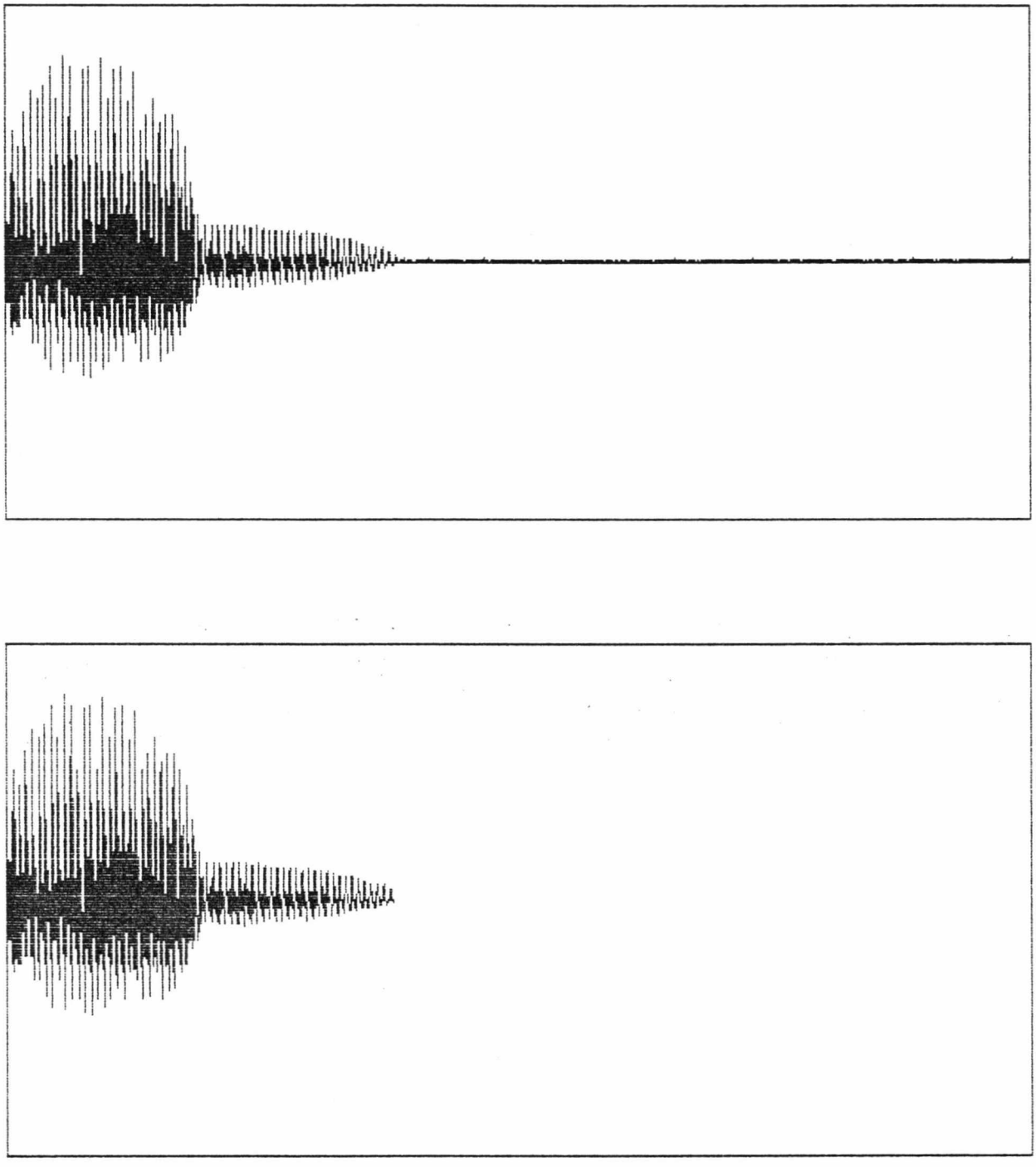
การหาขอบเขตสัญญาณที่ผิดพลาดในรูปที่ 5.1.11 เกิดจากมีสัญญาณรบกวนที่ปลายสัญญาณในช่วง 1 วินาที มีขนาดใหญ่ ซึ่งจะตัดสัญญาณที่แท้จริงบางส่วนหายไป

การหาขอบเขตสัญญาณที่ผิดพลาดในรูปที่ 5.1.12 เป็นกรณีเดียวกับรูปที่ 5.1.11 แต่สัญญาณเสียงตลอดทั้งหนดมีขนาดเล็ก และไม่มีตำแหน่งใดที่มีขนาดสัญญาณสูงกว่าเป็นจำนวนเท่า (ซึ่งกำหนดไว้ในโปรแกม) ของตำแหน่งสุดท้าย จึงทำให้โปรแกรมไม่สามารถตัดสัญญาณได้

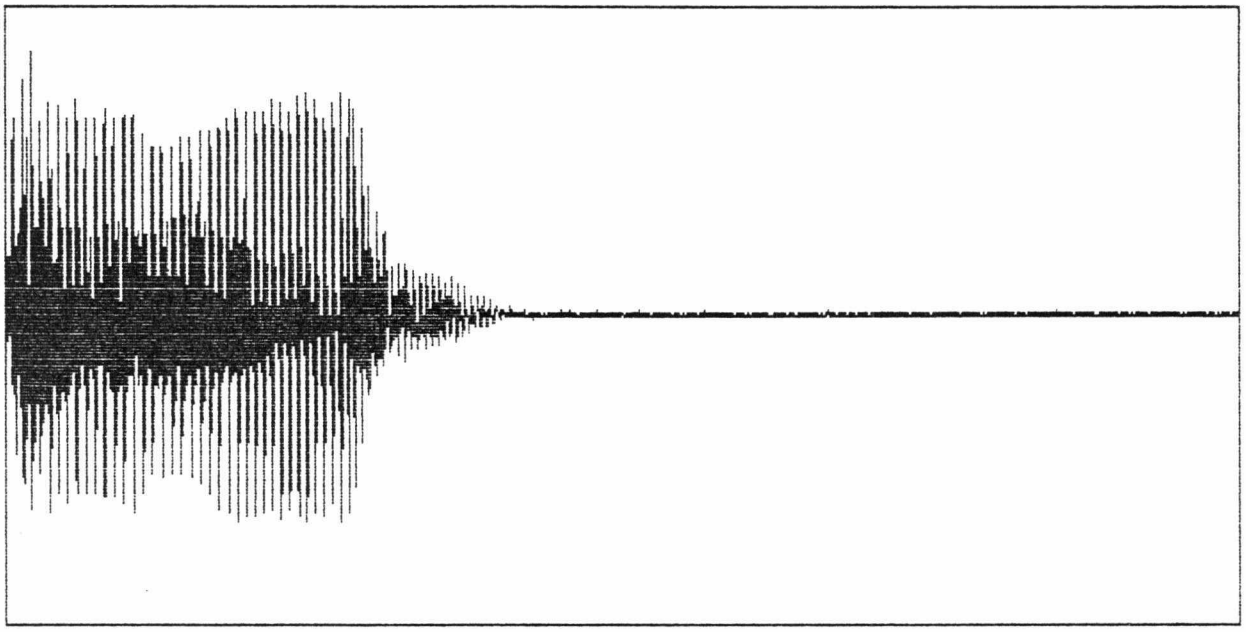
การหาขอบเขตสัญญาณที่ผิดพลาดในรูปที่ 5.1.13 เกิดจากมีสัญญาณรบกวนสูงเป็นช่วงๆ ซึ่งจะถือว่าสัญญาณเหล่านั้นเป็นส่วนหนึ่งของคำด้วย

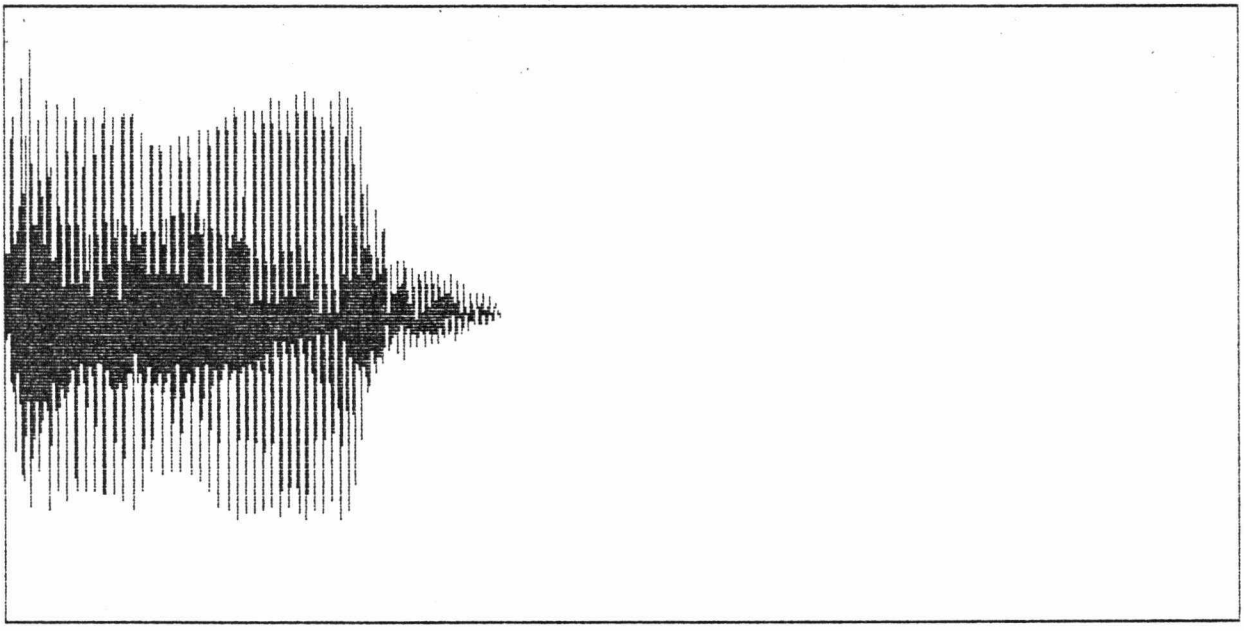


รูปที่ 5.1.1 แสดงขอบเขตสัญญาณที่ถูกต้องของคำว่า "ศูนย์"

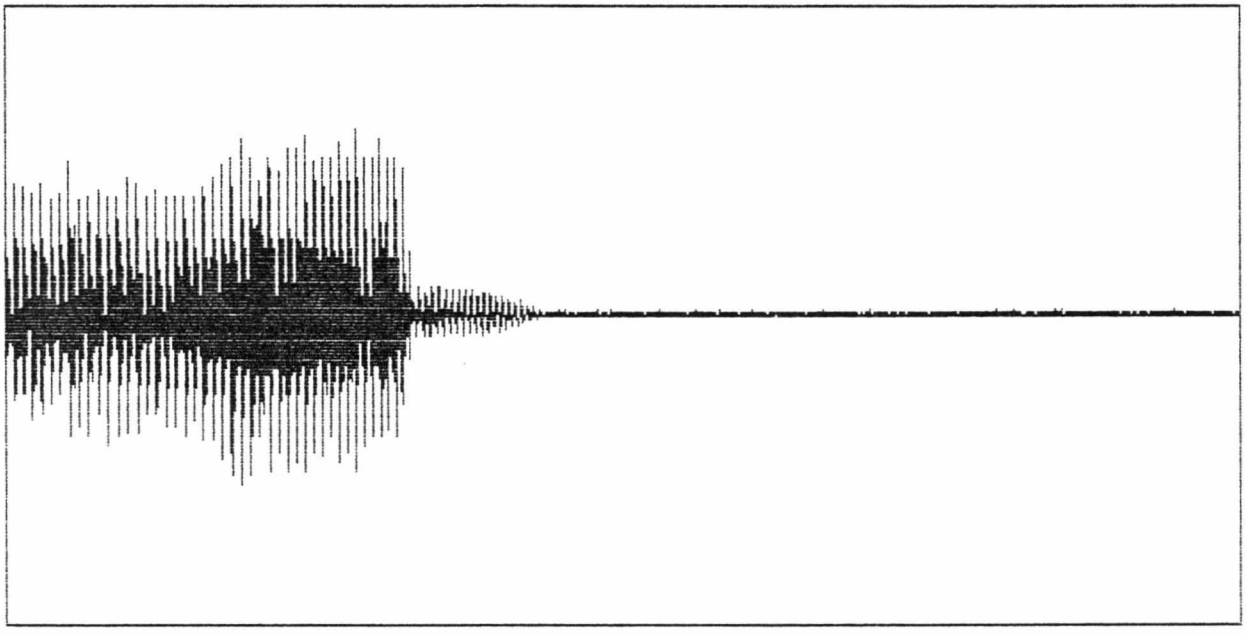


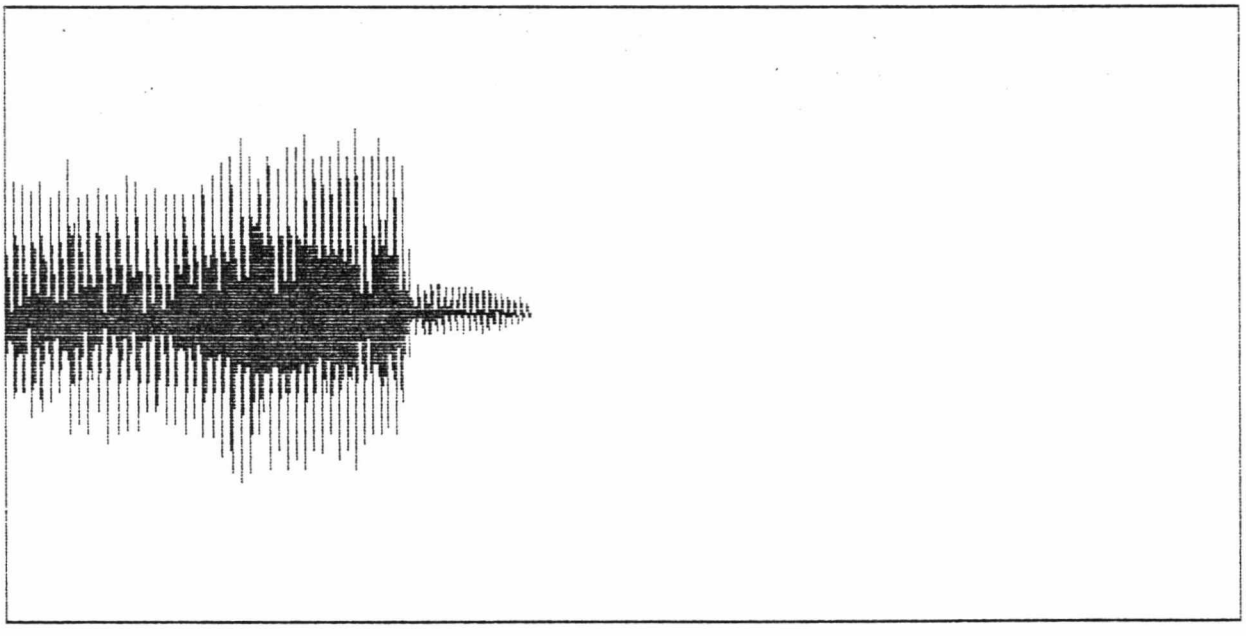
รูปที่ 5.1.2 แสดงขอบเขตสัญญาณที่ถูกต้องของคำว่า "หนึ่ง"



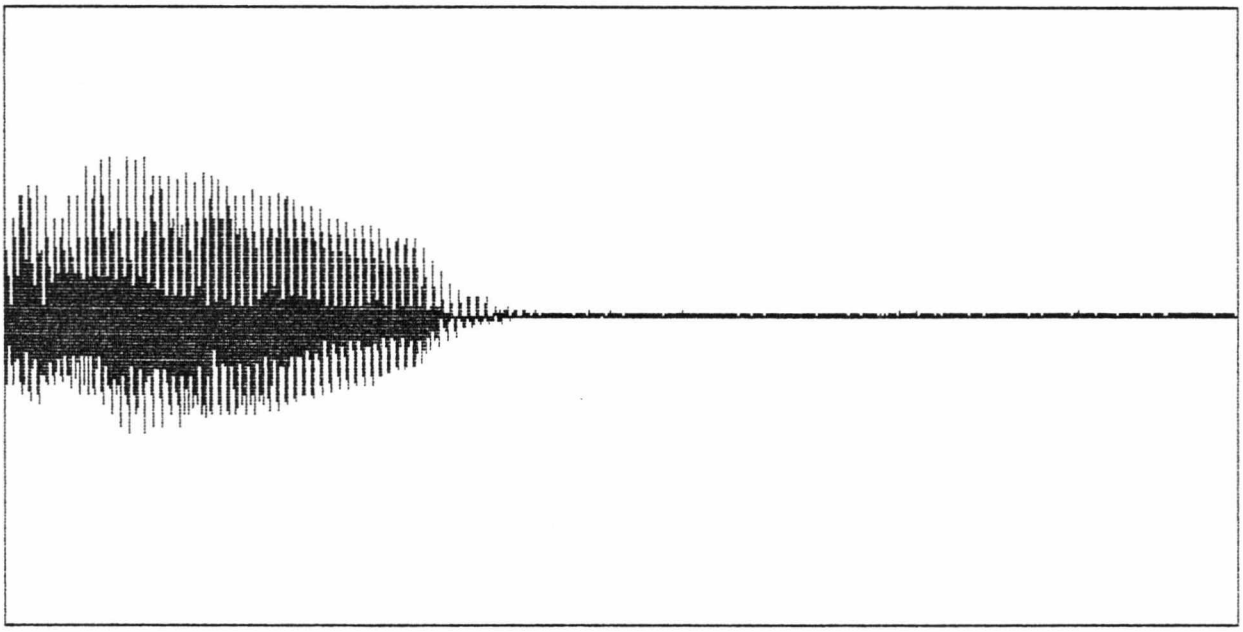


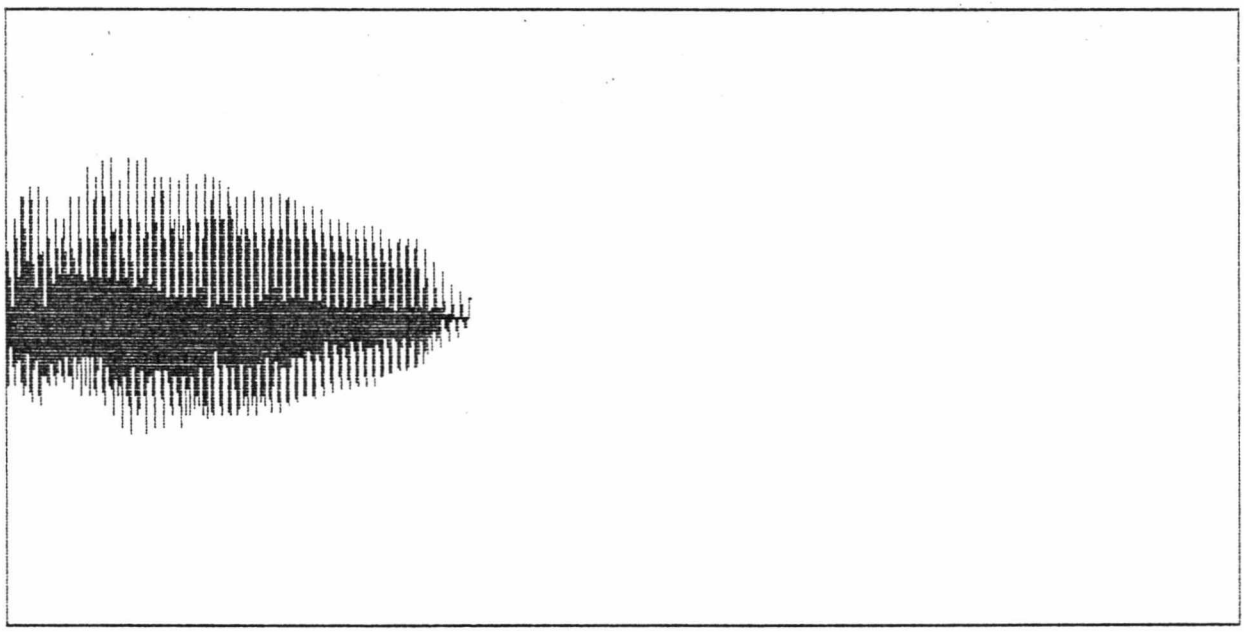
รูปที่ 5.1.3 แสดงขอบเขตสัญญาณที่ถูกต้องของคำว่า "สอง"



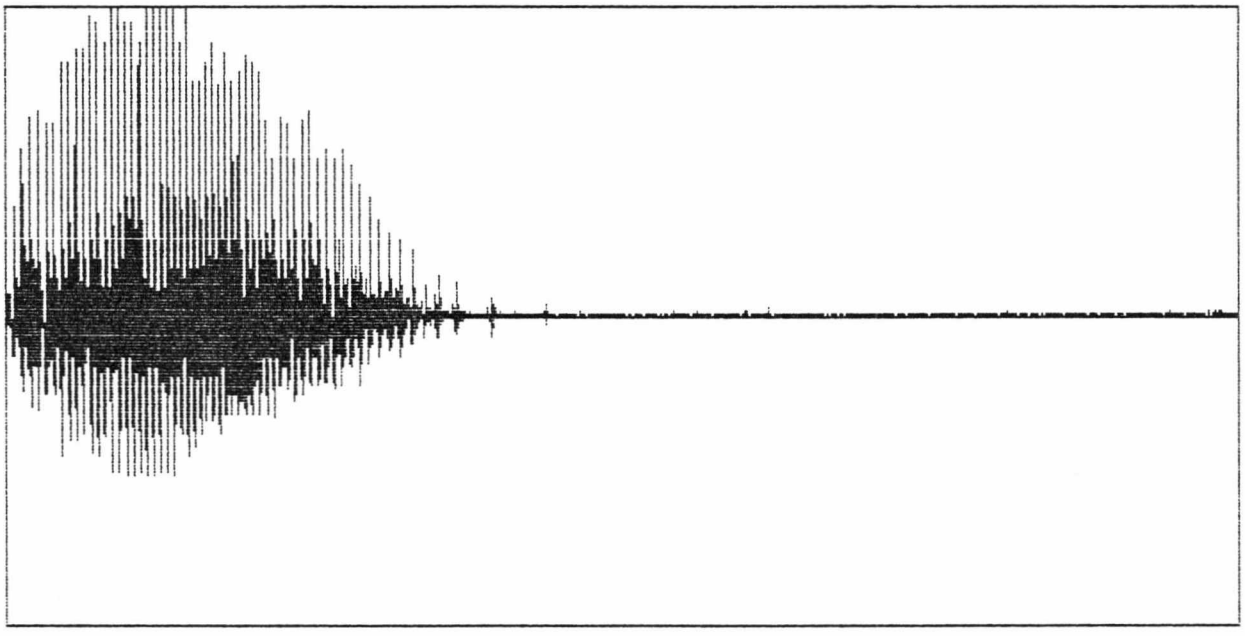


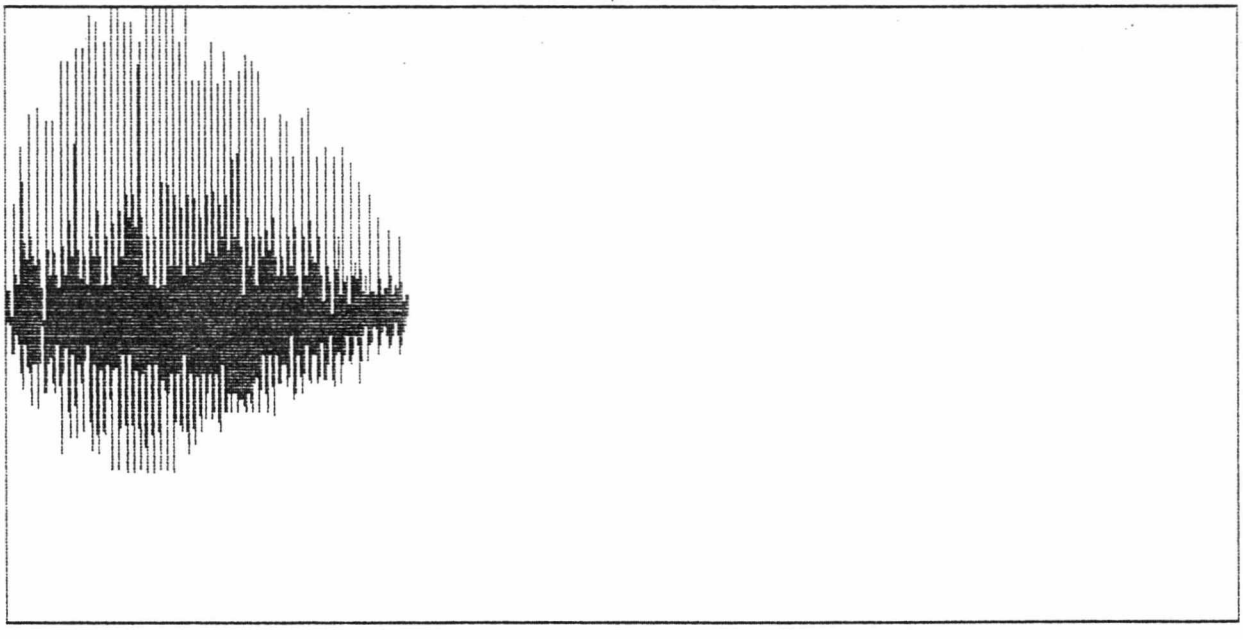
รูปที่ 5.1.4 แสดงขอบเขตสัญญาณที่ถูกต้องของคำว่า "สาม"



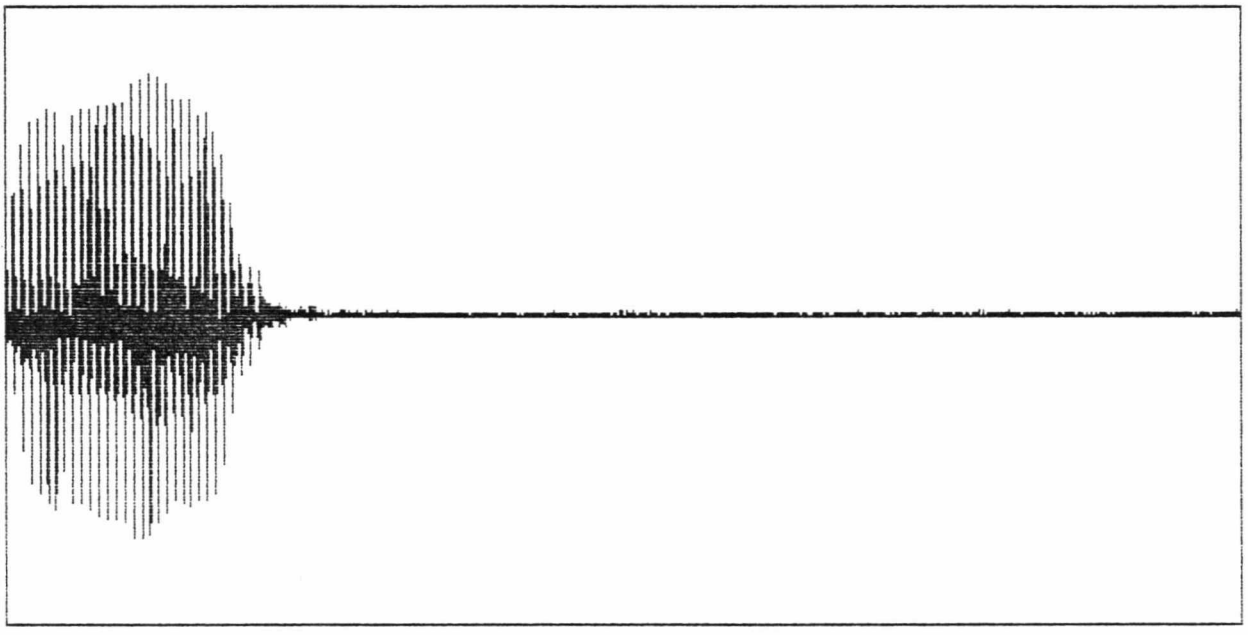


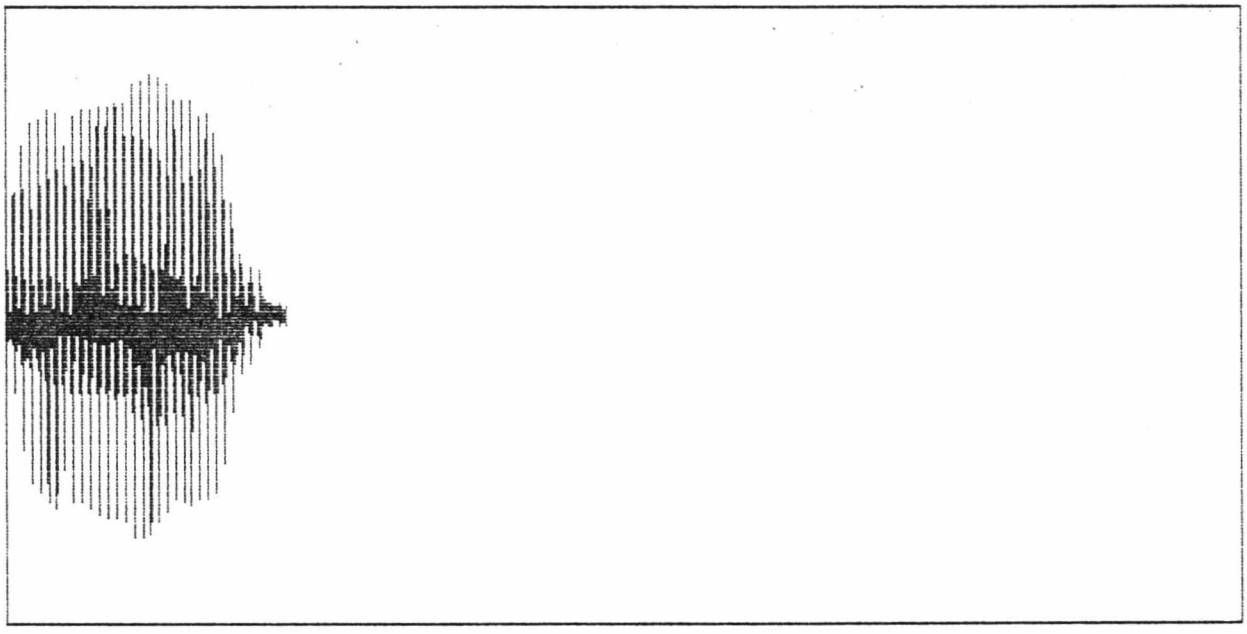
รูปที่ 5.1.5 แสดงขอบเขตสัญญาณที่ถูกต้องของคำว่า "สี่"



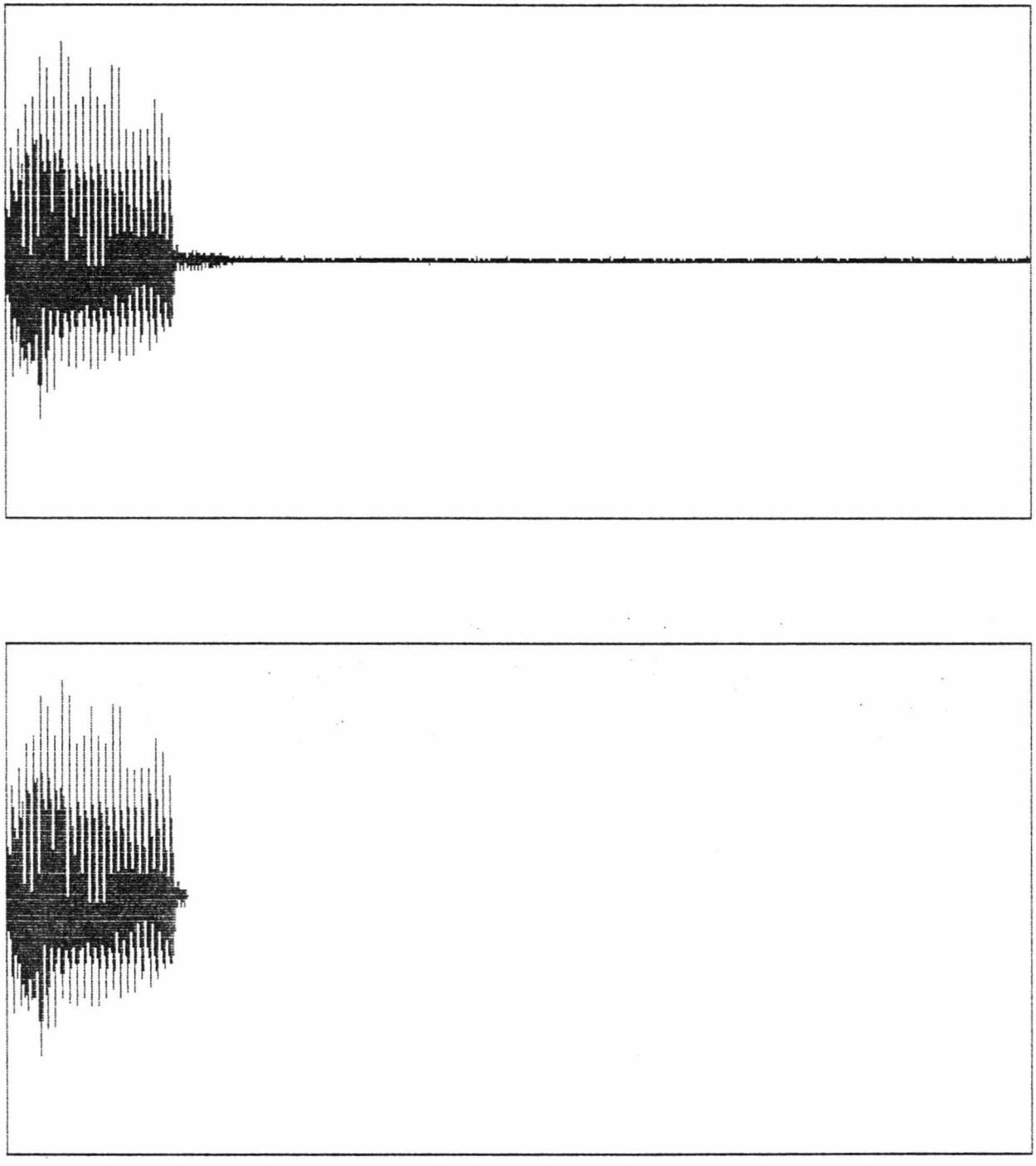


รูปที่ 5.1.6 แสดงขอบเขตสัญญาณที่ถูกต้องของคำว่า "ห้า"

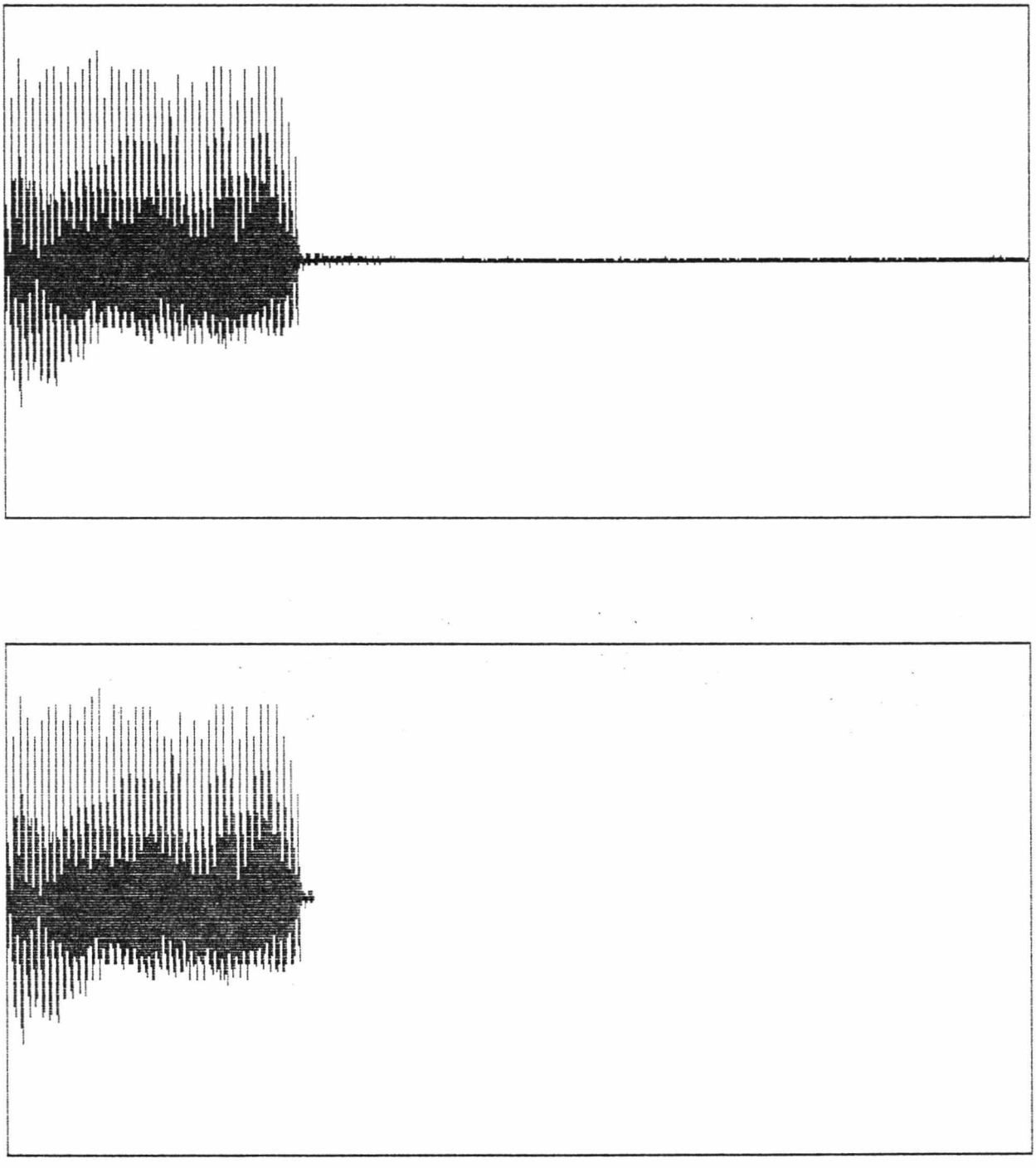




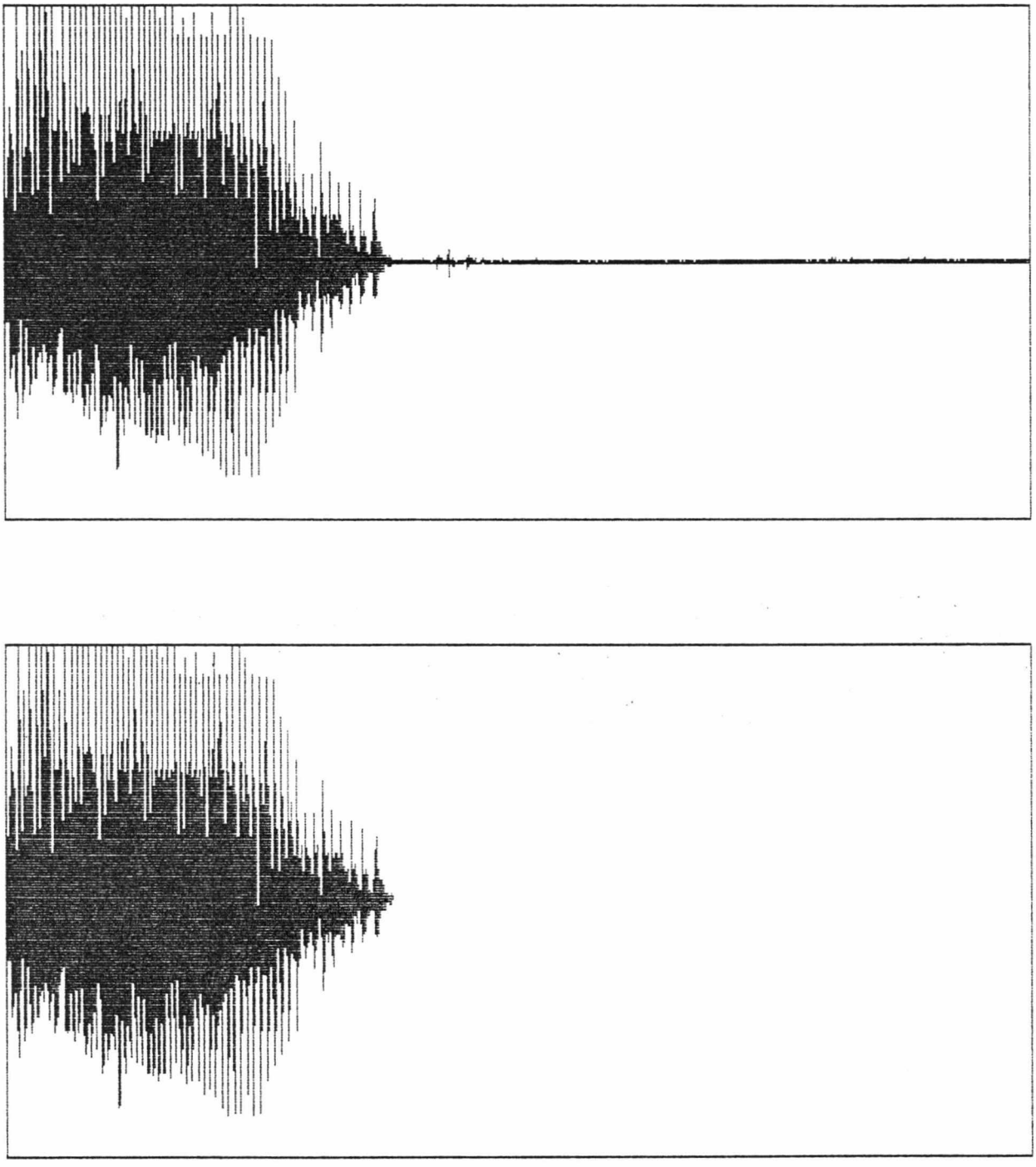
รูปที่ 5.1.7 **แสดงขอบเขตสัญญาณที่ถูกต้องของคำว่า "หก"**



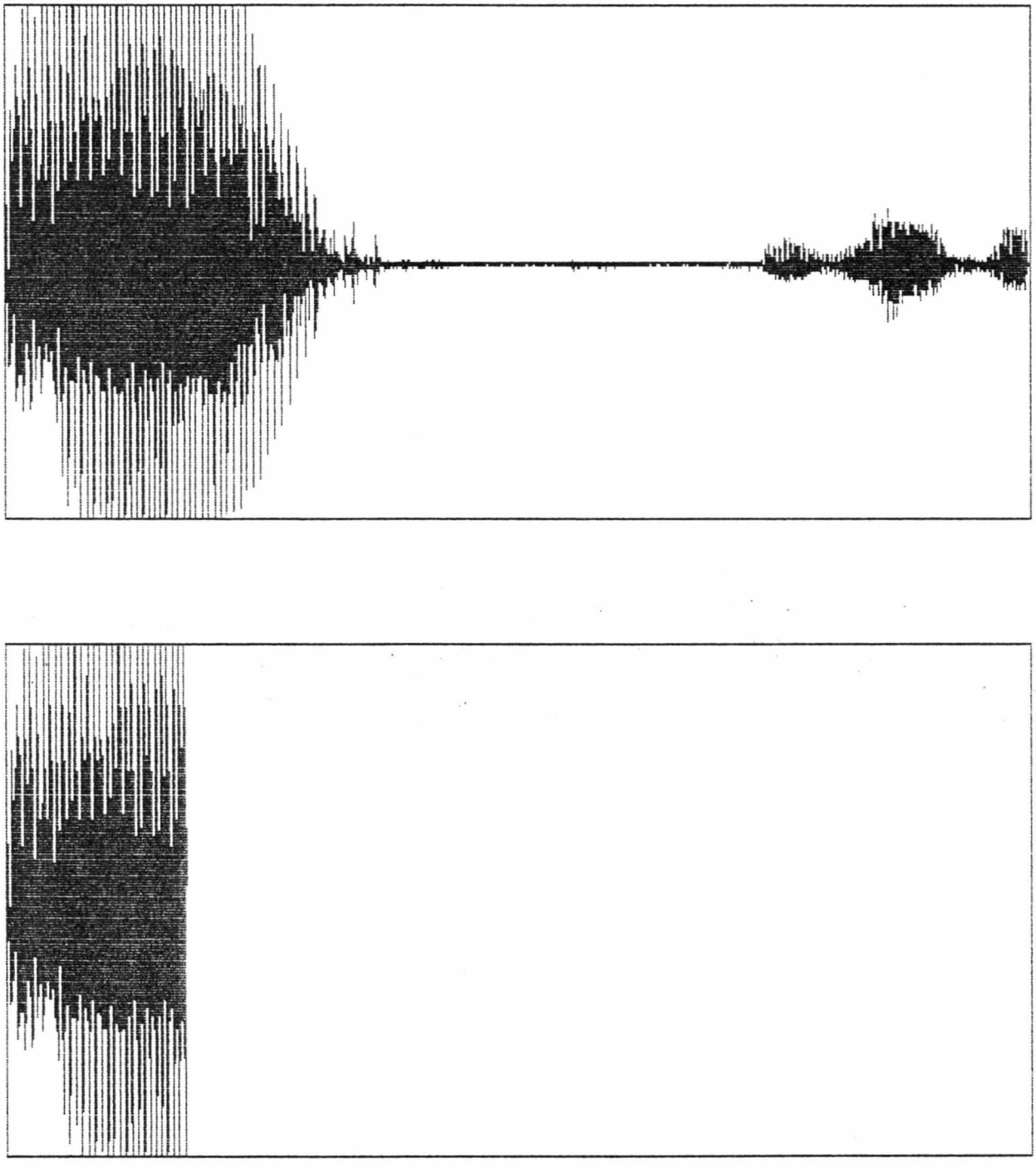
รูปที่ 5.1.8 แสดงขอบเขตสัญญาณที่ถูกต้องของคำว่า "เจ็ด"



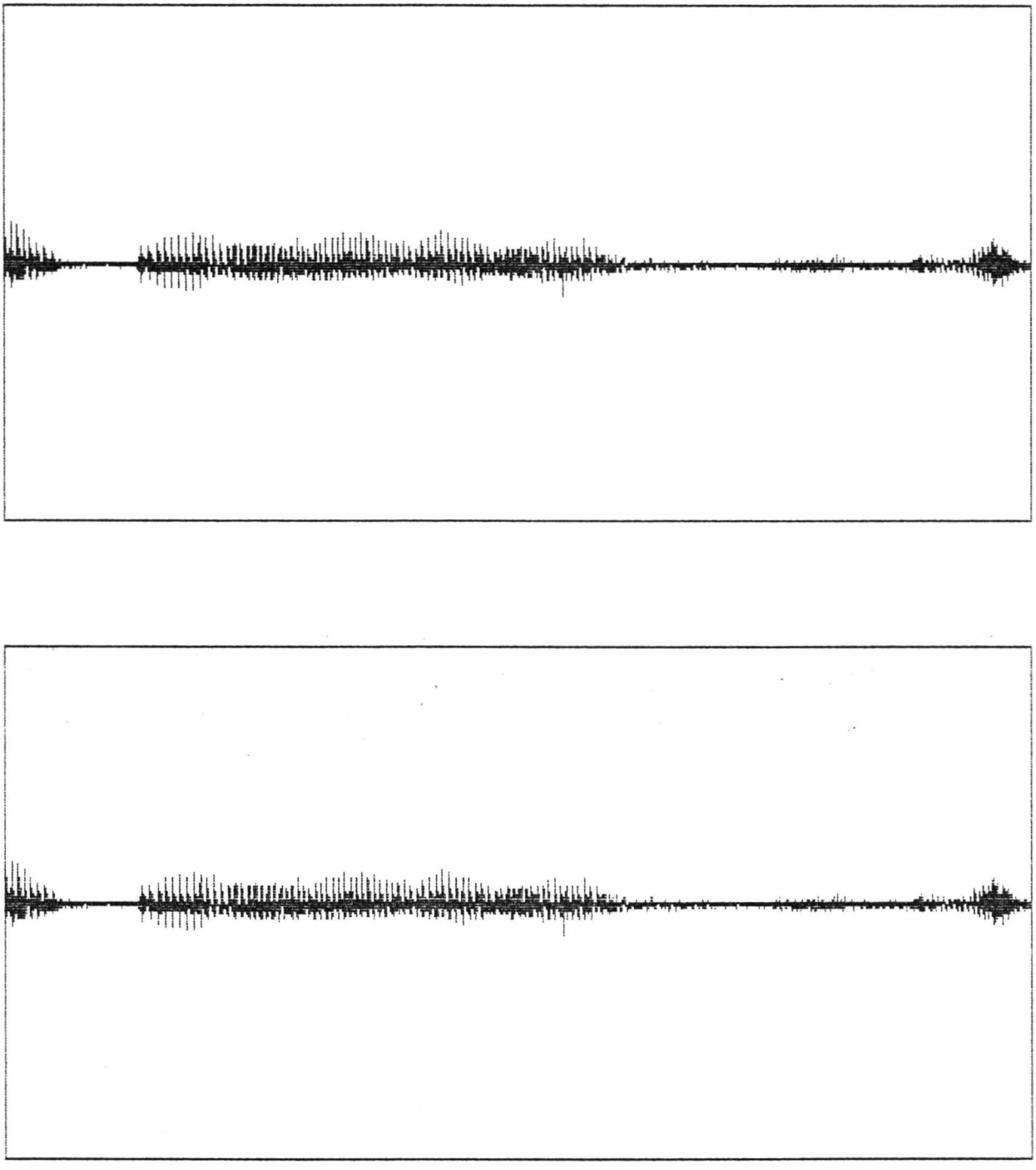
รูปที่ 5.1.9 แสดงขอบเขตสัญญาณที่ถูกต้องของคำว่า "แปด"



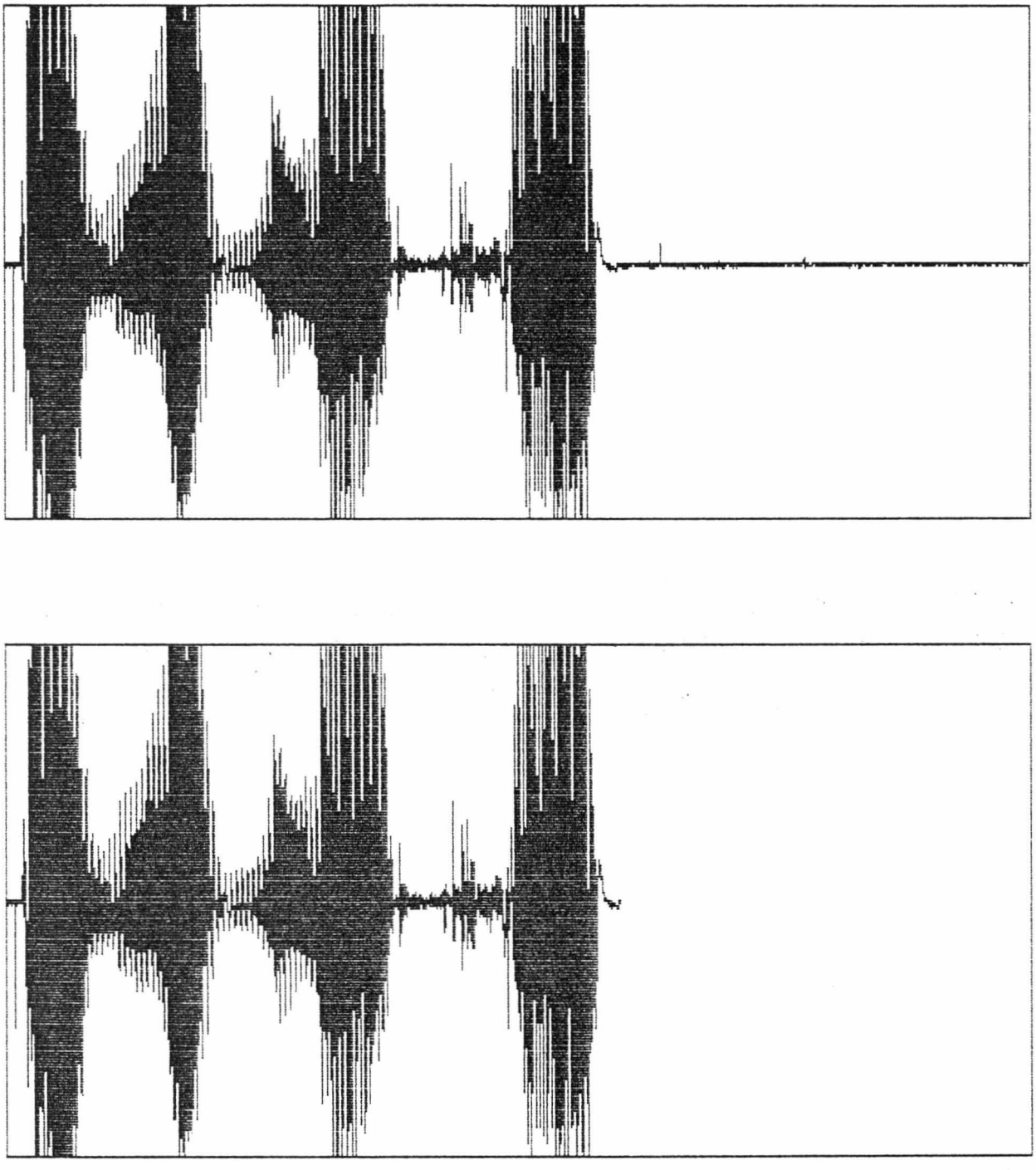
รูปที่ 5.1.10 แสดงขอบเขตสัญญาณที่ถูกต้องของคำว่า "เก้า"



รูปที่ 5.1.11 แสดงขอบเขตสัญญาณที่มีสัญญาณรบกวนขนาดใหญ่



รูปที่ 5.1.12 แสดงกรณีที่ไม่สามารถหาขอบเขตได้



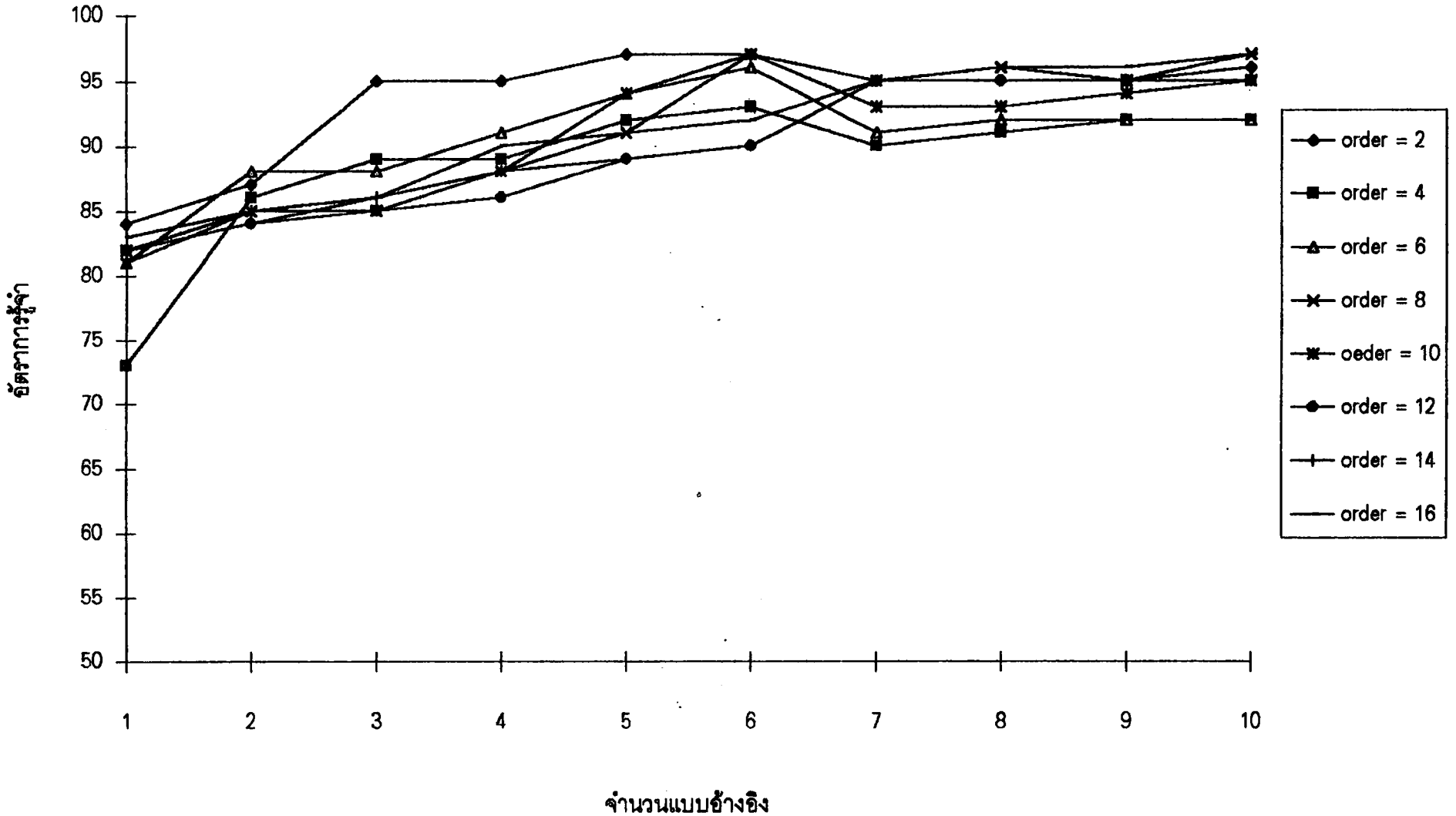
รูปที่ 5.1.13 แสดงขอบเขตสัญญาณที่มีสัญญาณรบกวนเป็นช่วง ๆ

5.2 ผลการทดลองส่วนรู้จำเสียงพูด

ผลการทดลองส่วนรู้จำเสียงพูดได้แสดงเป็นตารางระหว่างลำดับการทำนายและจำนวนอ้างอิง ดังตารางที่ 5.2.1 และแสดงเป็นกราฟระหว่างจำนวนแบบอ้างอิง และอัตราการรู้จำ โดยมีลำดับการทำนายเป็นอีกตัวแปรหนึ่งที่นำมาพิจารณา ดังรูปที่ 5.2.1

ลำดับการทำนาย	จำนวนแบบอ้างอิง									
	1	2	3	4	5	6	7	8	9	10
2	84	87	95	95	97	97	95	95	95	96
4	73	86	89	89	92	93	90	91	92	92
6	81	88	88	91	94	96	91	92	92	92
8	81	85	85	88	91	97	95	96	95	97
10	82	85	85	88	94	97	93	93	94	95
12	82	84	85	86	89	90	95	95	95	95
14	82	84	86	88	89	90	95	95	95	96
16	83	85	86	90	91	92	95	96	96	97

ตารางที่ 5.2.1 แสดงร้อยละของอัตราการรู้จำเสียงพูด



รูปที่ 5.2.1 แสดงผลการรู้จำเสียงพูด

บทที่ 6
สรุปผลและแนวทางการพัฒนา

6.1 สรุปผลการทดลอง

1. สามารถส่งข้อมูลจากบอร์ดไปยังคอมพิวเตอร์ทางพอร์ตอนุกรมด้วยความเร็ว 19,200 บิทต่อวินาที ได้อย่างถูกต้อง

2 จากการหาขอบเขตของสัญญาณ พบว่า ความถูกต้องของการหาขอบเขตของสัญญาณขึ้นอยู่กับสัญญาณรบกวน กล่าวคือ ถ้าระดับของสัญญาณรบกวนมีค่าต่ำจะสามารถหาขอบเขตของสัญญาณได้ถูกต้อง แต่ถ้าระดับของสัญญาณรบกวนมีค่าสูง สัญญาณเสียงพูดบางส่วนจะถูกตัดหายไป

3 จากผลการรู้จำเสียงพูด พบว่า อัตราการรู้จำแปรผันตรงกับจำนวนแบบอ้างอิงที่สร้างขึ้น และมีแนวโน้มเพิ่มขึ้นตามออร์เดอร์ของการทำนาย

4 เวลาที่ใช้ในการรู้จำเสียงพูด แปรผันตรงกับจำนวนแบบอ้างอิง

6.2 แนวทางการพัฒนา

1. เพิ่มความเร็วการส่งข้อมูลจากบอร์ดไปคอมพิวเตอร์ โดยอาจพัฒนาเป็นการส่งแบบขนาน

2. เพื่อให้ได้ผลการรู้จำที่ถูกต้องมากยิ่งขึ้น ควรเปลี่ยนวิธีการหาสัมประสิทธิ์การทำนาย และวิธีการรู้จำ

3. พัฒนาเป็นการรู้จำแบบต่างบุคคล

4. พัฒนาระบบรู้จำให้สามารถนำไปใช้งานได้จริง

ภาคผนวก ก.

โปรแกรมควบคุมการทำงานบนบอร์ด

```
;
;*******************************
;
        CPU     "8051.TBL"
        HOF     "INT8"
;
;*******************************
;
;MCS-51 INTERNAL REGISTERS
;
B:      EQU     0F0H    ;B REGISTER
ACC:    EQU     0E0H    ;ACCUMULATOR
PSW:    EQU     0D0H    ;PROGRAM STATUS WORD
IPC:    EQU     0B8H    ;INTERRUPT PRIORITY
P3:     EQU     0B0H    ;PORT 3
IEC:    EQU     0A8H    ;INTERRUPT ENABLE
P2:     EQU     0A0H    ;PORT 2
SBUF:   EQU     99H     ;SEND BUFFER
SCON:   EQU     98H     ;SERIAL CONTROL
P1:     EQU     90H     ;PORT 1
TH1:    EQU     8DH     ;TIMER 1 HIGH
TH0:    EQU     8CH     ;TIMER 0 HIGH
TL1:    EQU     8BH     ;TIMER 1 LOW
TL0:    EQU     8AH     ;TIMER 0 LOW
TMOD:   EQU     89H     ;TIMER MODE
TCON:   EQU     88H     ;TIMER CONTROL
PCON:   EQU     87H     ;POWER CONTROL REGISTER
DPH:    EQU     83H     ;DATA POINTER HIGH
DPL:    EQU     82H     ;DATA POINTER LOW
SP:     EQU     81H     ;STACK POINTER
P0:     EQU     80H     ;PORT 0
;
;*******************************
;
CY:     EQU     0D7H    ;CARRY FLAG
AC:     EQU     0D6H    ;AUXILIARY-CARRY FLAG
F0:     EQU     0D5H    ;USER FLAG 0
RS1:    EQU     0D4H    ;REGISTER SELECT MSB
RS0:    EQU     0D3H    ;REGISTER SELECT LSB
OV:     EQU     0D2H    ;OVERFLOW FLAG
P:      EQU     0D0H    ;PARITY FLAG
PS:     EQU     0BCH    ;PRIORITY SERIAL PORT
PT1:    EQU     0BBH    ;PRIORITY TIMER 1
PX1:    EQU     0BAH    ;PRIORITY EXTERNAL 1
PT0:    EQU     0B9H    ;PRIORITY TIMER 0
PX0:    EQU     0B8H    ;PRIORITY EXTERNAL 0
EA:     EQU     0AFH    ;ENABLE ALL INTERRUPT
ES:     EQU     0ACH    ;ENABLE SERIAL INTERRUPT
ET1:    EQU     0ABH    ;ENABLE TIMER 1 INTERRUPT
EX1:    EQU     0AAH    ;ENABLE EXTERNAL 1 INTERR
ET0:    EQU     0A9H    ;ENABLE TIMER 0 INTERRUPT
EX0:    EQU     0A8H    ;ENABLE EXTERNAL 0 INTERR
SM0:    EQU     09FH    ;SERIAL MODE 0
SM1:    EQU     09EH    ;SERIAL MODE 1
```

```
SM2:        EQU         09DH        ;SERIAL MODE 2
REN:        EQU         09CH        ;SERIAL RECEPTION ENABLE
TB8:        EQU         09BH        ;TRANSMITT BIT 8
RB8:        EQU         09AH        ;RECEIVE BIT 8
TI:         EQU         099H        ;TRANSMIT INTERRUPT FLAG
RI:         EQU         098H        ;RECEIVE INTERRUPT FLAG
TF1:        EQU         08FH        ;TIMER 1 OVERFLOW FLAG
TR1:        EQU         08EH        ;TIMER 1 RUN CONTROL BIT
TFO:        EQU         08DH        ;TIMER O OVERFLOW FLAG
TRO:        EQU         08CH        ;TIMER O RUN CONTROL BIT
IE1:        EQU         08BH        ;EXT INTERR. 1 EDGE FLAG
IT1:        EQU         08AH        ;EXT INTERR. 1 TYPE FLAG
IEO:        EQU         089H        ;EXT INTERR. O EDGE FLAG
ITO:        EQU         088H        ;EXT INTERR. O TYPE FLAG
;
;*******************************
;
WORD_IN:    EQU         0B2H        ; PORT 3.2 FROM WORD PRESENT
TRIG:       EQU         0B3H        ; PORT 3.3 TO START OF A/D
DSR:        EQU         0B4H        ; USE PORT 3.4 AS DATA SET READY
DTR:        EQU         0B5H        ; USE PORT 3.5 AS DATA TERMINAL REA
FULL_MEM:   EQU         0D5H        ; USE USER FLAG TO CHECK DATA
;
;*******************************
;
            ORG         00H
            AJMP        INIT
;
;*******************************
;
            ORG         000BH
            AJMP        ADC
;
;*******************************
;
            ORG         0023H
            AJMP        RECEIVE
;
;*******************************
;
INIT:       MOV         TL0,#141
            MOV         TH0,#141
            MOV         TH1,#253
            MOV         TMOD,#00100010B
            SETB        SM1
            MOV         IEC,#10010010B
            MOV         IPC,#00000010B
            MOV         PCON,#10000000B
            MOV         SP,#30H
            SETB        TR1
            SETB        DSR
            SETB        REN
            MOV         DPTR,#8003H
            MOV         A,#80H
```

```
            MOVX        @DPTR,A
            MOV         DPTR,#8002H
            MOV         A,#0FFH
            MOVX        @DPTR,A
;
;*******************************
;
MAIN:       CLR         FULL_MEM
AUDIO:      JNB         WORD_IN,AUDIO
            MOV         IEC,#10000010B
            CLR         REN
            MOV         DPTR,#8003H
            MOV         A,#00H
            MOVX        @DPTR,A
            MOV         DPTR,#0000H
            ACALL       START
CONVERSE:   JNB         FULL_MEM,CONVERSE
            CLR         TR0
            ACALL       SEND
            MOV         IEC,#10010010B
            SETB        REN
            MOV         DPTR,#8003H
            MOV         A,#01H
            MOVX        @DPTR,A
            AJMP        MAIN
;
;*******************************
;
START:      CLR         TRIG
            NOP
            SETB        TRIG
            SETB        TR0
            RET
;
;*******************************
;
GET_DATA:   MOV         A,P1
            MOVX        @DPTR,A
            RET
;
;*******************************
;
SEND:       CLR         DSR
            MOV         DPTR,#0000H
LOOP:       MOVX        A,@DPTR
WAIT:       JB          DTR,WAIT
            MOV         SBUF,A
READY:      JNB         TI,READY
            CLR         TI
            INC         DPTR
            MOV         R0,DPL
            CJNE        R0,#40H,LOOP
            MOV         R0,DPH
            CJNE        R0,#1FH,LOOP
```

```
            SETB        DSR
            RET
;
;*******************************
;
ADC:        CLR         TR0
            ACALL       START
            ACALL       GET_DATA
            INC         DPTR
            MOV         A,DPL
            CJNE        A,#40H,EXIT
            MOV         A,DPH
            CJNE        A,#1FH,EXIT
            SETB        FULL_MEM
EXIT:       RETI
;
;*******************************
;
RECEIVE:    MOV         DPTR,#8003H
            MOV         A,#02H
            MOVX        @DPTR,A
            MOV         IEC,#10000010B
            MOV         DPTR,#0000H
            CLR         RI
            MOV         A,SBUF
            MOVX        @DPTR,A
            INC         DPTR
            JB          DTR,PLAY
READ:       CLR         DSR
READ1:      JNB         RI,READ1
            SETB        DSR
            CLR         RI
            MOV         A,SBUF
            MOVX        @DPTR,A
            INC         DPTR
            JB          DTR,PLAY
            AJMP        READ
PLAY:       MOV         00H,DPL
            MOV         01H,DPH
            MOV         DPTR,#0000H
PLAY1:      MOVX        A,@DPTR
            PUSH        DPL
            PUSH        DPH
            MOV         DPTR,#8001H
            MOVX        @DPTR,A
            POP         DPH
            POP         DPL
            ACALL       DELAY
            INC         DPTR
            MOV         A,DPL
            CJNE        A,00H,PLAY1
            MOV         A,DPH
            CJNE        A,01H,PLAY1
            MOV         IEC,#10010010B
```

```
            MOV         DPTR,#8003H
            MOV         A,#03H
            MOVX        @DPTR,A
            RETI
;
;*******************************
;
DELAY:      MOV         R0,#19H
DELAY1:     NOP
            NOP
            DJNZ        R0,DELAY1
            RET
;
;*******************************
;
```

ภาคผนวก ข.

โปรแกรมการวิเคราะห์และรู้จำเสียงพูด

```
program recognized;
uses crt,dos,graph,win;

const MaxMenu = 6;

const  Quit : boolean = false;

const menu : array[1..MaxMenu] of MenuRec =
             ((win    : (5,4,21,14);
               col    : (7,7,7,7,7,7,7,0,0,0);
               row    : (6,7,8,9,10,11,12,0,0,0);
               WinCol : (15,3,norm);
               AcCol  : (15,4,high);
               head   : (' MAIN MENU ');
               msg    : (' IMPORT DATA ',' EDIT DATA ',' WAVEFORM ',
                         ' PLAYBACK ',' TEMPLATE ',' RECOG ',' QUIT ',
                         '','','');
               CurAc  : 1;
               MaxC   : 7),

               (win    : (15,11,51,15);
               col    : (17,0,0,0,0,0,0,0,0,0);
               row    : (13,0,0,0,0,0,0,0,0,0);
               WinCol : (11,2,norm);
               AcCol  : (11,1,high);
               head   : (' IMPORT DATA ');
               msg    : (' FILENAME : ','',
                         '','','','','','','','');
               CurAc  : 1;
               MaxC   : 1),

              (win    : (16,9,53,16);
               col    : (18,18,0,0,0,0,0,0,0,0);
               row    : (11,14,0,0,0,0,0,0,0,0);
               WinCol : (11,7,norm);
               AcCol  : (11,2,high);
               head   : ('');
               msg    : (' FILENAME : ',' NUMBER : ',
                         '','','','','','','','');
               CurAc  : 1;
               MaxC   : 2),

              (win    : (16,9,59,16);
               col    : (18,18,0,0,0,0,0,0,0,0);
               row    : (11,14,0,0,0,0,0,0,0,0);
               WinCol : (11,7,norm);
               AcCol  : (11,1,high);
               head   : (' TEMPLATE ');
               msg    : ('.VOI FILENAME  : ','.TEM FILENAME  : ',
                         '','','','','','','','');
               CurAc  : 1;
               MaxC   : 2),

              (win    : (16,11,31,16);
```

```
                col    : (18,18,0,0,0,0,0,0,0,0);
                row    : (13,14,0,0,0,0,0,0,0,0);
                WinCol : (11,7,norm);
                AcCol  : (11,1,high);
                head   : (' RECOG ');
                msg    : (' FROM FILE ',' FROM BOARD ',
                          '','','','','','','','');
                CurAc  : 1;
                MaxC   : 2),

               (win    : (25,8,66,15);
                col    : (27,27,0,0,0,0,0,0,0,0);
                row    : (10,13,0,0,0,0,0,0,0,0);
                WinCol : (11,2,norm);
                AcCol  : (11,1,high);
                head   : (' RECOG FROM FILE ');
                msg    : ('.VOI FILENAME : ','.TEM FILENAME : ',
                          '','','','','','','','');
                CurAc  : 1;
                MaxC   : 2));

const bit0 = 1;
      bit1 = 2;
      bit2 = 4;
      bit3 = 8;
      bit4 = 16;
      bit5 = 32;
      bit6 = 64;
      bit7 = 128;

const buff = $3F8;
      DLLSB = $3F8;
      DLMSB = $3F9;
      IER = $3F9;
      IIR = $3FA;
      LCR = $3FB;
      LSR = $3FD;
      MCR = $3FC;
      MSR = $3FE;

const pxl = 200;
      frm = 40;
      order = 4;

const AdjustWindow = 6;

type a8000 = array[1..8000] of byte;
     s8000 = array[1..8000] of real;

type GraphType = record
                   MinimumX,MaximumX : integer;
                   MinimumY,MaximumY : integer;
                   wx1,wy1           : integer;
                   wx2,wy2           : integer;
```

```
                  end;

type cvec = array[1..order] of real;
     CoeffPtrType = ^cvec;
     CoeffAddressType = ^CoeffPtrType;

type rvec = array[0..order] of real;

type InfType = record
                    number : byte;
                    start  : integer;
                    NumOfFrm : integer;
                end;
     InfPtrType = ^InfType;
     InfAddressType = ^InfPtrType;

type gType = array[1..((AdjustWindow*2)+1)] of real;
     gPtrType = ^gType;

var coeff : cvec;
    CoeffFile : File of cvec;

var CoeffPtr1,CoeffPtr2 : CoeffPtrType;
    InfPtr1,InfPtr2 : InfPtrType;

var gPtr : gPtrType;

var CoeffAddress1,CoeffAddress2 : CoeffAddressType;
    InfAddress1,InfAddress2 : InfAddressType;

var voice : a8000;
    signal : s8000;
    VoiceFile : file of a8000;

var inf : InfType;
    InfFile : File of InfType;

var LastFrm,LastPxl : integer;
    num : longint;

var msg : string;
    percent : real;

var FileN,p : string;
    FileN1,p1 : string;

var dp : pathstr;
    d,d1 : dirstr;
    n,n1 : namestr;
    e,e1 : extstr;

var key3,key4 : char;

function difference(i,j,order : integer) : real;
```

```
var temp1,temp2 : CoeffPtrType;
    count : integer;
    x,y,z : real;
begin
     temp1 := CoeffPtr1;
     temp2 := CoeffPtr2;
     if i > 1 then for count := 1 to i-1 do inc(temp1);
     if j > 1 then for count := 1 to j-1 do inc(temp2);
     x := temp1^[order];
     y := temp2^[order];
     difference :=  sqrt((sqr(i-j))+(sqr(x-y)));
end;

function index(i,j : integer) : integer;
begin
     index := (AdjustWindow+1)-(j-i);
end;

function case1(i,j,order : integer) : real;
var temp : gPtrType;
    count,k : integer;
begin
     temp := gPtr;
     if j-2 > 1 then for count := 1 to (j-2)-1 do inc(temp);
     k := index(i-1,j-2);
     if temp^[k] = -1 then case1 := -1
     else
     case1 :=  temp^[k]+(2*difference(i,j-1,order))+difference(i,j,order);
end;

function case2(i,j,order : integer) : real;
var temp : gPtrType;
    count,k : integer;
begin
     temp := gPtr;
     if j-1 > 1 then for count := 1 to (j-1)-1 do inc(temp);
     k := index(i-1,j-1);
     if temp^[k] = -1 then case2 := -1
     else
     case2 := temp^[k]+(2*difference(i,j,order));
end;

function case3(i,j,order : integer) : real;
var temp : gPtrType;
    count,k : integer;
begin
     temp := gPtr;
     if j-1 > 1 then for count := 1 to j-1-1 do inc(temp);
     k := index(i-2,j-1);
     if temp^[k] = -1 then case3 := -1
     else
     case3 := temp^[k]+(2*difference(i-1,j,order))+difference(i,j,order);
end;
```

```
function g(i,j,order : integer) : real;
var min,min1 : real;
    i1,j1,i2,j2 : integer;
begin
     i1 := i-1;
     j1 := j-1;
     i2 := i-2;
     j2 := j-2;
     if (i = 1) and (j = 1) then min := difference(1,1,order)
     else
     begin
          if (not (abs(i1-j2) > AdjustWindow)) and
             (not ((i1 <= 0) or (j2 <= 0))) then
          begin
               if (not (abs(i1-j1) > AdjustWindow)) and
                  (not ((i1 <= 0) or (j1 <= 0))) then
               begin
                    if (not (abs(i2-j1) > AdjustWindow)) and
                       (not ((i2 <= 0) or (j1 <= 0))) then
                    begin
                         min :=  case1(i,j,order);
                         min1 := case2(i,j,order);
                         if (min <> -1) and (min1 <> -1) then
                         begin
                              if min1 < min then min := min1;
                         end
                         else
                             if min = -1 then min := min1;
                         min1 := case3(i,j,order);
                         if (min <> -1) and (min1 <> -1) then
                         begin
                              if min1 < min then min := min1;
                         end
                         else
                             if min = -1 then min := min1;
                    end
                    else
                    begin
                         min := case1(i,j,order);
                         min1 := case2(i,j,order);
                         if (min <> -1) and (min1 <> -1) then
                         begin
                              if min1 < min then min := min1;
                         end
                         else
                             if min = -1 then min := min1;
                    end;
               end
               else
               begin
                    if (not (abs(i2-j1) > AdjustWindow)) and
                       (not ((i2 <= 0) or (j1 <= 0))) then
                       begin
                            min := case1(i,j,order);
```

```
                        min1 := case3(i,j,order);
                        if (min <> -1) and (min1 <> -1) then
                        begin
                              if min1 < min then min := min1;
                        end
                        else
                        begin
                            if min = -1 then min := min1;
                        end;
                    end
                    else
                    begin
                        min := case1(i,j,order);
                    end;
               end;
          end
          else
          begin
               if (not (abs(i1-j1) > AdjustWindow)) and
                  (not ((i1 <= 0) or (j1 <= 0))) then
               begin
                    if (not (abs(i2-j1) > AdjustWindow)) and
                       (not ((i2 <= 0) or (j1 <= 0))) then
                       begin
                        min := case2(i,j,order);
                        min1 := case3(i,j,order);
                        if (min <> -1) and (min1 <> -1) then
                        begin
                              if min1 < min then min := min1;
                        end
                        else
                            if min = -1 then min := min1;
                    end
                    else
                    begin
                        min := case2(i,j,order);
                    end;
               end
               else
               begin
                    if (not (abs(i2-j1) > AdjustWindow)) and
                       (not ((i2 <= 0) or (j1 <= 0))) then
                    begin
                       min := case3(i,j,order);
                    end
                    else
                    begin
                         min := -1;
                    end;
               end;
          end;
     end;
     g := min;
end;
```

```
function distance(ii,jj,order : integer) : real;
var i,j : integer;
    temp : gPtrType;
begin
     if (abs(ii-jj) > AdjustWindow) then distance := -1
     else
     begin
          getmem(gPtr,(6*((AdjustWindow*2)+1)*jj));
          temp := gPtr;
          for j := 1 to jj do
          begin
               for i := 1 to ((AdjustWindow*2)+1) do temp^[i] := -1;
               inc(temp);
          end;
          temp := gPtr;
          i := 1;
          j := 1;
          while j <= jj do
          begin
               if i > (j + AdjustWinDow) then
               begin
                    inc(j);
                    inc(temp);
                    i := j - AdjustWindow;
               end
               else
               begin
                    if (i <= ii) and (i > 0) then
                    begin
                         temp^[index(i,j)] := g(i,j,order);
                    end;
                    inc(i);
                    end;
               end;
          temp := gPtr;
          for i := 1 to jj-1 do inc(temp);
          distance := temp^[index(ii,jj)]/(ii+jj);
          freemem(gPtr,(6*((AdjustWindow*2)+1)*jj));
     end;
end;

procedure average(var av : longint);
var p : integer;
begin
     av := 0;
     for p := 7801 to 8000 do av := av + voice[p];
     av := round(av/200);
end;

procedure DcFilter(av : longint);
var i,last : integer;
begin
     for i := 1 to LastPxl do
```

```
     signal[i] := voice[i]-av;
end;

procedure boundary;
var count,f,p,temp : integer;
    SumPxl : array[1..frm] of integer;
    found : boolean;
    av : longint;
begin
     average(av);
     count := 0;
     for f := 1 to frm do
     begin
          SumPxl[f] := 0;
          for p := 1 to pxl do
          begin
               inc(count);
               temp := abs(voice[count]-av);
               SumPxl[f] := SumPxl[f] + temp;
          end;
     end;
     f := 1;
     LastFrm := frm;
     found := false;
     repeat
           if (SumPxl[frm-f] / SumPxl[frm]) >= 8 then
           begin
                found := true;
                LastFrm := frm - f;
           end
           else
           begin
                inc(f);
                if f = frm then found := true;
           end;
     until found;
     LastPxl := pxl*LastFrm;
     DcFilter(av);
end;

procedure WindowSignal(frame : integer);
var i,j : integer;
begin
     i := 0;
     for j := (frame*pxl)-(pxl-1) to (frame*pxl) do
     begin
          signal[j] := signal[j]*(0.54-(0.46*(cos((2*pi*i)/(pxl-1)))));
          inc(i);
     end;
end;

procedure auto(frame : integer; var r : rvec);
var i,j : integer;
begin
```

```
     for i := 0 to order do
     begin
          r[i] := 0;
          for j := (frame*pxl)-(pxl-1) to (frame*pxl)-i do
          r[i] := r[i] + (signal[j] * signal[j+i]);
     end;
end;

procedure CalCoeff(frame : integer);
var ee : real;
    i,j : byte;
    r,temp : rvec;
begin
     WindowSignal(frame);
     auto(frame,r);
     ee := r[0];
     coeff[1] := r[1]/ee;
     for i := 2 to order do
     begin
          ee := (1-coeff[i-1]*coeff[i-1])*ee;
          coeff[i] := r[i];
          for j := 1 to i-1 do coeff[i] := coeff[i] - r[i-j]*coeff[j];
          coeff[i] := coeff[i]/ee;
          for j := 1 to i-1 do temp[j] := coeff[j] - coeff[i]*coeff[i-j];
          for j := 1 to i-1 do coeff[j] := temp[j];
     end;
end;

procedure InitSerialPort;
var status : byte;
begin
     port[LCR] := $80;          { access to DLLSM and DLMSB register }
     port[DLLSB] := $06;        { set baud rate = 19200 bit/sec }
     port[DLMSB] := $00;
     port[LCR] := $07;          { access to THR,RBR and IER register }
     port[MCR] := $00;          { pc not ready }
end;

procedure DispStatus(msg : string);
begin
     PrintStr(msg,2,25,15,4,norm);
end;

procedure ClearStatus;
var i : byte;
begin
     for i := 1 to 78 do
     PrintStr(' ',i+1,25,1,7,norm);
end;

procedure receive;
var status : byte;
    count : integer;
    EndReceive : boolean;
```

```
begin
     status := port[buff];       { clear data and data ready flag }
     status := port[LSR];        { clear status }
     EndReceive := false;
     port[MCR] := $01;           { pc ready }
     repeat
     status := port[MSR];
     until ((status and bit5) <> 0);
     count := 1;
     repeat
     status := port[MSR];                        { check board ready or not
}
     if ((status and bit5) <> 0) then            { board ready to send }
     begin
          port[MCR] := $01;                      { pc ready }
          repeat
          status := port[LSR];                   { check data valid or not }
          if (status and  bit0) <> 0 then        { data valid }
          begin
               port[MCR] := $00;
               percent := ((count-1)/8000)*100;
               str(percent:5:2,msg);
               DispStatus(' COMPLETE : '+msg+' % ');
               voice[count] := port[buff];
               inc(count);
          end;
          until (status and bit0) <> 0;
     end
     else
         EndReceive := true;
     until EndReceive;
     ClearStatus;
end;

procedure play;
var status : byte;
    i : integer;
    SendReady,BoardReady : boolean;
begin
     port[MCR] := $01;
     BoardReady := false;
     SendReady := false;
     port[buff] := voice[1];
     for i := 2 to LastPxl do
     begin
          repeat
          status := port[LSR];
          if (status and bit5 <> 0) then
          begin
               if (status and bit6 <> 0) then SendReady := true;
          end;
          until SendReady;
          SendReady := false;
```

```
          repeat
          status := port[MSR];
          if (status and bit5) <> 0 then
          begin
               if (status and bit1) <> 0 then BoardReady := true;
          end;
          until BoardReady;
          BoardReady := false;
          percent := ((i-1)/LastPxl)*100;
          str(percent:5:2,msg);
          DispStatus(' COMPLETE : '+msg+' % ');
          port[buff] := voice[i];
     end;
     port[MCR] := $00;
     ClearStatus;
end;

procedure OpenGraph(var GrError : integer);
var GrDriver,GrMode : integer;
begin
     detectgraph(GrDriver,GrMode);
     initgraph(GrDriver,GrMode,'c:\work\tp\bgi');
     GrError := graphresult;
end;

procedure SetLimit(var g : GraphType; MinX,MaxX,MinY,MaxY : integer);
begin
     with g do
     begin
          MinimumX := minX; MaximumX := MaxX;
          MinimumY := MinY; MaximumY := MaxY;
     end;
end;

procedure SetWindow(var g : GraphType; x1,y1,x2,y2 : integer);
begin
     with g do
     begin
          wx1 := x1; wx2 := x2;
          wy1 := y1; wy2 := y2;
     end;
end;

procedure PlotWindow(g : GraphType);
begin
     with g do rectangle(wx1,wy1,wx2,wy2);
end;

procedure plot(g : GraphType; x,y : integer; option : byte);
begin
     with g do
     begin
          x := wx1 + round((wx2-wx1)*((x-MinimumX)/(MaximumX-MinimumX)));
          y := wy2 - round((wy2-wy1)*((y-MinimumY)/(MaximumY-MinimumY)));
```

```
     end;
     case option of
     0 : begin
              PutPixel(x,y,yellow);
              MoveTo(x,y);
         end;
     1 : begin
              SetColor(yellow);
              LineTo(x,y);
         end;
     end;
end;

function OpenError(FileN : string) : boolean;
begin
     OpenError := false;
     assign(VoiceFile,FileN);
     {$I-}
     rewrite(VoiceFile);
     {$I+}
     if IoResult <> 0 then OpenError := true
end;

function DiskSpaceError : boolean;
begin
     DiskSpaceError := false;
     if DiskFree(0) < (SizeOf(voice)*10) then DiskSpaceError := true;
end;

procedure ErrorWin(x1,y1,x2,y2 : byte);
begin
     SetWin(14,4,norm);
     SetHeader(' ERROR ');
     OpenWin(x1,y1,x2,y2);
end;

procedure OverWrite(var OverW : boolean; x1,y1,x2,y2 : byte);
var msg : string;
begin
     ErrorWin(x1,y1,x2,y2);
     msg := 'FILE ALREADY EXIST';
     PrintStr(msg,center(msg,x1,x2),y1+2,WinColor[1],WinColor[2],WinColor[3
]);
     msg := 'REPLACE? (Y/N)';
     PrintStr(msg,center(msg,x1,x2),y1+4,WinColor[1],WinColor[2],WinColor[3
]);
     write(#7);
     repeat
     key3 := readkey;
     if key3 = #00 then key4 := readkey;
     key3 := upcase(key3);
     until key3 in ['Y','N'];
     case key3 of
     'Y' : OverW := true;
```

```
     'N' : OverW := false;
     end;
     CloseWin;
end;

procedure update(var upd : boolean; x1,y1,x2,y2 : byte);
var msg : string;
begin
     ErrorWin(x1,y1,x2,y2);
     msg := 'FILE ALREADY EXIST';
     PrintStr(msg,center(msg,x1,x2),y1+2,WinColor[1],WinColor[2],WinColor[3
]);
     msg := 'UPDATE OR NEW? (U/N)';
     PrintStr(msg,center(msg,x1,x2),y1+4,WinColor[1],WinColor[2],WinColor[3
]);
     write(#7);
     repeat
     key3 := readkey;
     if key3 = #00 then key4 := readkey;
     key3 := upcase(key3);
     until key3 in [#27,'U','N'];
     case key3 of
     'U' : upd := true;
     'N' : upd := false;
     end;
     CloseWin;
end;

procedure error(ErrorCode : byte; x1,y1,x2,y2 : byte);
var msg : string;
    key : char;
begin
     ErrorWin(x1,y1,x2,y2);
     case ErrorCode of
     1 : msg := 'PATH NOT FOUND';
     2 : msg := 'DISK FULL';
     3 : msg := 'CANNOT SAVE DATA';
     4 : msg := 'FILE NOT FOUND';
     5 : msg := 'GRAPHIC ERROR';
     6 : msg := 'NOT ENOUGH MEMORY';
     7 : msg := 'CANNOT RECOGNIZED';
     end;
     PrintStr(msg,center(msg,x1,x2),y1+2,WinColor[1],WinColor[2],WinColor[3
]);
     msg := 'PRESS ANY KEY TO CONTINUE';
     PrintStr(msg,center(msg,x1,x2),y1+4,WinColor[1],WinColor[2],WinColor[3
]);
     write(#7);
     repeat until keypressed;
     key3:= readkey;
     if key3= #00 then key4 := readkey;
     CloseWin;
end;
```

```
procedure background;
var x,y : byte;
begin
     TextColor(9);
     TextBackground(7);
     clrscr;
     for x := 1 to 80 do
     for y := 2 to 24 do
     VRam[y,x,1] := #219;
     PrintStr(' SPEECH RECOGNITION ',2,1,15,4,high);
end;

procedure MainMenu;
begin
     with menu[1] do
     begin
          active(menu[1],CurAc);
          repeat
               key1 := readkey;
               UnActive(menu[1],CurAc);
               if key1 = #00 then
               begin
                    key2 := readkey;
                    case key2 of
                    #72 : begin
                              dec(CurAc);
                              if CurAC < 1 then CurAc := MaxC;
                          end;
                    #80 : begin
                              inc(CurAC);
                              if CurAc > MaxC then CurAc := 1;
                          end;
                    #45 : CurAc := MaxC;
                    end;
               end
               else
               begin
                    key1 := upcase(key1);
                    case key1 of
                    'I' : CurAc := 1;
                    'E' : CurAc := 2;
                    'W' : CurAc := 3;
                    'P' : CurAc := 4;
                    'T' : CurAc := 5;
                    'R' : CurAc := 6;
                    end;
               end;
               active(menu[1],CurAc);
          until ((key1 in [#13,'I','E','W','P','T','R']) or
               ((key1 = #00) and (key2 = #45)));
     end;
end;

procedure input1(x,y : byte);
```

```
begin
     repeat
     ReadStr(p,x,y,20);
     until key1 in [#13,#27];
end;

procedure NumWin(num : byte; x1,y1,x2,y2 : byte);
var msg : string;
begin
     SetWin(15,1,norm);
     SetHeader('');
     OpenWin(x1,y1,x2,y2);
     BoxLine(x1+1,y1+1,x2-1,y2-4);
     BoxLine(x1+1,y1+4,x2-1,y2-1);
     msg := 'NUMBER : ';
     PrintStr(msg,x1+2,y1+2,WinColor[1],WinColor[2],WinColor[3]);
     PrintInt(num,x1+11,y1+2,1);
     msg := 'PRESS ANY KEY TO START';
     PrintStr(msg,center(msg,x1,x2),y1+5,WinColor[1],WinColor[2],WinColor[3
]);
end;

procedure WaitWin(x1,y1,x2,y2 : byte);
var msg : string;
begin
     SetWin(1,3,norm);
     SetHeader('');
     OpenWin(x1,y1,x2,y2);
end;

procedure WaitMsg(i : byte; x1,y1,x2,y2 : byte);
var j : byte;
begin
     case i of
     1 : msg := 'transfering data ...';
     2 : msg := 'saving template ...';
     3 : msg := 'loading template...';
     4 : msg := 'calculating...';
     end;
     for i := 1 to 26 do
     PrintStr(' ',x1+1+i,y1+2,WinColor[1],WinColor[2],WinColor[3]);
     PrintStr(msg,center(msg,x1,x2),y1+2,WinColor[1]+blink,WinColor[2],WinC
olor[3]);
end;

procedure ImportData;
var cancel,OverW : boolean;
    count : byte;
begin
     cancel := false;
     with menu[2] do
     begin
          head := ' IMPORT DATA ';
          win[1] := 15;
```

```
     win[2] := 11;
     win[3] := 51;
     win[4] := 15;
     col[1] := 17;
     row[1] := 13;
end;
MenuWin(menu[2]);
repeat
input1(29,13);
fsplit(p,d,n,e);
FileN := n+'.VOI';
if key1 = #13 then
begin
     if DirError(d) then error(1,25,13,56,19)
     else
     begin
         if  FileExist(FileN) then
         begin
              OverWrite(OverW,25,13,56,19);
              if OverW = true then
              begin
                   assign(VoiceFile,FileN);
                   erase(VoiceFile);
              end;
         end
         else
              OverW := true;
         if OverW = true then
         if DiskSpaceError then error(2,25,13,56,19)
         else
             if OpenError(FileN) then error(3,25,13,56,19)
             else
             begin
                  for count := 0 to 9 do
                  begin
                       if (not cancel) then
                       begin
                            NumWin(count,26,12,56,19);
                            repeat until keypressed;
                            key3 := readkey;
                            if key3 = #00 then key4 := readkey;
                            if key3 = #27 then
                            begin
                                 close(VoiceFile);
                                 erase(VoiceFile);
                                 CloseWin;
                                 cancel := true;
                            end
                            else
                            begin
                                 CloseWin;
                                 WaitWin(26,13,56,17);
                                 WaitMsg(1,26,13,56,17);
                                 receive;
```

```
                                        write(VoiceFile,voice);
                                        CloseWin;
                                   end;
                              end;
                         end;
                         if (not cancel) then
                         begin
                              close(VoiceFile);
                         end
                         else
                             cancel := false;
                    end;
               end;
               chdir(dp);
          end;
          until (key1 = #27);
          CloseWin;
end;

procedure InputWin(i : byte);
begin
     with menu[3] do
     begin
          case i of
          1 : head := ' EDIT DATA ';
          2 : head := ' WAVEFORM ';
          3 : head := ' PLAYBACK ';
          end;
          MenuWin(menu[3]);
          BoxLine(win[1]+1,win[2]+1,win[3]-1,win[4]-4);
          BoxLine(win[1]+1,win[2]+4,win[3]-1,win[4]-1);
     end;
end;

procedure input2(x1,y1,x2,y2 : byte);
var i : byte;
begin
     i := 1;
     PrintInt(num,x2,y2,1);
     repeat
     case i of
     1 : ReadStr(p,x1,y1,20);
     2 : ReadInt(num,x2,y2,1);
     end;
     if key1 = #00 then
     case key2 of
     #72 : begin
                dec(i);
                if i < 1 then i := 2;
           end;
     #80 : begin
                inc(i);
                if i > 2 then i := 1;
           end;
```

```
     end;
     until (key1 = #13) or (key1 = #27);
end;

procedure PlotVoice;
var first : boolean;
    GrError : integer;
    g1,g2 : GraphType;
    i : integer;
begin
     SetLimit(g1,1,8000,0,255);
     SetWindow(g1,round(GetMaxX/20),round(GetMaxY/20),
                        round((GetMaxX*9.5)/10),round((GetMaxY*4.5)/10));
     PlotWindow(g1);
     SetLimit(g2,1,8000,0,255);
     SetWindow(g2,round(GetMaxX/20),round((GetMaxY*5.5)/10),
                        round((GetMaxX*9.5)/10),round((GetMaxY*9.5)/10));
     PlotWindow(g2);
     first := true;
     for i := 1 to 8000 do
     begin
          if first then
          begin
               plot(g1,i,voice[i],0);
               first := false;
          end
          else
              plot(g1,i,voice[i],1);
     end;
     first := true;
     for i := 1 to LastPxl do
     begin
          if first then
          begin
               plot(g2,i,voice[i],0);
               first := false;
          end
          else
              plot(g2,i,voice[i],1);
     end;
     repeat until keypressed;
     key3 := readkey;
     if key3 = #00 then key4 := readkey;
     CloseGraph;
end;

procedure waveform;
var GrError : integer;
begin
     InputWin(2);
     repeat
     input2(30,11,28,14);
     if key1 = #13 then
     begin
```

```
     fsplit(p,d,n,e);
     FileN := n+'.VOI';
     if DirError(d) then error(1,25,13,56,19)
     else
         if (not FileExist(FileN)) then error(4,25,13,56,19)
         else
             begin
                  CloseWin;
                  CloseWin;
                  OpenGraph(GrError);
                  if GrError <> 0 then
                  begin
                       background;
                       MenuWin(menu[1]);
                       InputWin(2);
                       error(5,25,13,56,19);
                  end
                  else
                  begin
                       assign(VoiceFile,FileN);
                       reset(VoiceFile);
                       seek(VoiceFile,num);
                       read(VoiceFile,voice);
                       boundary;
                       PlotVoice;
                       close(VoiceFile);
                       background;
                       MenuWin(menu[1]);
                       InputWin(2);
                  end;
             end;
     end;
     chdir(dp);
     until (key1 = #27);
     CloseWin;
end;

procedure EditData;
begin
     InputWin(1);
     repeat
     input2(30,11,28,14);
     if key1 = #13 then
     begin
     fsplit(p,d,n,e);
     FileN := n+'.VOI';
     if DirError(d) then error(1,25,13,56,19)
     else
         if (not FileExist(FileN)) then error(4,25,13,56,19)
         else
             begin
                  begin
                       NumWin(num,26,12,56,19);
                       repeat until keypressed;
```

```
                    key3 := readkey;
                    if key3 = #00 then key4 := readkey;
                    if key3 <> #27 then
                    begin
                         CloseWin;
                         WaitWin(26,13,56,17);
                         WaitMsg(1,26,13,56,17);
                         receive;
                         CloseWin;
                         assign(VoiceFile,FileN);
                         reset(VoiceFile);
                         seek(VoiceFile,num);
                         write(VoiceFile,voice);
                         close(VoiceFile);
                    end
                    else
                        CloseWin;
               end;
          end;
     end;
     chdir(dp);
     until (key1 = #27);
     CloseWin;
end;

procedure PlayBack;
begin
     InputWin(3);
     repeat
     input2(30,11,28,14);
     if key1 = #13 then
     begin
     fsplit(p,d,n,e);
     FileN := n+'.VOI';
     if DirError(d) then error(1,25,13,56,19)
     else
         if (not FileExist(FileN)) then error(4,25,13,56,19)
         else
             begin
                  begin
                       WaitWin(26,13,56,17);
                       WaitMsg(1,26,13,56,17);
                       assign(VoiceFile,FileN);
                       reset(VoiceFile);
                       seek(VoiceFile,num);
                       read(VoiceFile,voice);
                       close(VoiceFile);
                       boundary;
                       play;
                       CloseWin;
                  end;
             end;
     end;
     chdir(dp);
```

```
     until (key1 = #27);
     CloseWin;
end;

procedure input3(x1,y1,x2,y2 : byte);
var i : byte;
begin
     i := 1;
     PrintStr(p,x1,y1,WinColor[1],winColor[2],WinColor[3]);
     PrintStr(p1,x2,y2,WinColor[1],winColor[2],WinColor[3]);
     repeat
     case i of
     1 : ReadStr(p,x1,y1,20);
     2 : ReadStr(p1,x2,y2,20);
     end;
     if key1 = #00 then
     case key2 of
     #72 : begin
               dec(i);
               if i < 1 then i := 2;
           end;
     #80 : begin
               inc(i);
               if i > 2 then i := 1;
           end;
     end;
     until (key1 = #13) or (key1 = #27);
end;

procedure SaveTemplate(var DiskFull : boolean);
var i : byte;
    temp,NumRec,j : integer;
begin
     WaitWin(26,14,56,18);
     WaitMsg(2,26,14,56,18);
     i := 0;
     NumRec := FileSize(CoeffFile);
     DiskFull := false;
     repeat
     percent := i*10;
     str(percent:5:2,msg);
     DispStatus(' COMPLETE : '+msg+' % ');
     chdir(d);
     read(VoiceFile,voice);
     boundary;
     chdir(d1);
     if DiskFree(0) < ((sizeof(coeff)*LastFrm)+(sizeof(inf)*10)) then
     begin
          DiskFull := true;
     end
     else
     begin
          for j := 1 to LastFrm do
          begin
```

```
                CalCoeff(j);
                write(CoeffFile,coeff);
            end;
            with inf do
            begin
                number := i;
                start := NumRec;
                NumOfFrm := LastFrm;
            end;
            write(InfFile,inf);
            NumRec := NumRec + LastFrm;
        end;
        inc(i);
        until (i = 10) or DiskFull;
        ClearStatus;
        CloseWin;
end;

procedure TemplateWin;
begin
        MenuWin(menu[4]);
        with menu[4] do
        begin
            BoxLine(win[1]+1,win[2]+1,win[3]-1,win[4]-4);
            BoxLine(win[1]+1,win[2]+4,win[3]-1,win[4]-1);
        end;
end;

procedure OpenFile(i : byte);
begin
        assign(VoiceFile,FileN);
        chdir(d);
        reset(VoiceFile);
        assign(CoeffFile,FileN1);
        assign(InfFile,n1+'.INF');
        chdir(d1);
        case i of
        1 : begin
                rewrite(CoeffFile);
                rewrite(InfFile);
            end;
        2 : begin
                reset(CoeffFile);
                reset(InfFile);
                seek(CoeffFile,FileSize(CoeffFile));
                seek(InfFile,FileSize(InFFile));
            end;
        end;
end;

procedure CloseFile;
begin
        chdir(d);
        close(VoiceFile);
```

```
     chdir(d1);
     close(CoeffFile);
     close(InfFile);
end;

procedure template;
var upd,DiskFull : boolean;
begin
     TemplateWin;
     repeat
     input3(36,11,36,14);
     if key1 = #13 then
     begin
     fsplit(p,d,n,e);
     fsplit(p1,d1,n1,e1);
     FileN := n+'.VOI';
     if DirError(d) then error(1,25,13,56,19)
     else
         if (not FileExist(FileN)) then error(4,25,13,56,19)
         else
             begin
                  FileN1 := n1+'.TEM';
                  if DirError(d1) then error(1,25,13,56,19)
                  else
                      if (not FileExist(FileN1)) then
                      begin
                           if OpenError(FileN1) then error(3,25,13,56,19)
                           else
                           begin
                                OpenFile(1);
                                SaveTemplate(DiskFull);
                                CloseFile;
                                if DiskFull then
                                begin
                                     error(2,25,13,56,19);
                                     erase(CoeffFile);
                                     erase(InfFile);
                                end;
                           end
                      end
                      else
                      begin
                           if (not FileExist(n1+'.INF')) then
                           begin
                                error(4,25,13,56,19);
                           end
                           else
                           begin
                                update(upd,25,13,56,19);
                                if key3 <> #27 then
                                begin
                                     case upd of
                                     true : OpenFile(2);
                                     false : OpenFile(1);
```

```
                                        end;
                                        SaveTemplate(DiskFull);
                                        CloseFile;
                                        if DiskFull then
                                        begin
                                             error(2,25,13,56,19);
                                        end;
                                   end;
                              end;
                         end;
               end;
     end;
     chdir(dp);
     until (key1 = #27);
     CloseWin;
end;

procedure LoadTemplate(var NumOfTem : integer);
var temp1 : CoeffAddressType;
    temp2 : InfAddressType;
    temp3 : CoeffPtrType;
    i,j : integer;
begin
     chdir(d1);
     assign(InfFile,n1+'.INF');
     reset(InfFile);
     NumOfTem := FileSize(InfFile);
     getmem(InfAddress1,(4*NumOfTem));
     temp2 := InfAddress1;
     for i := 1 to NumOfTem do
     begin
          getmem(temp2^,sizeof(inf));
          read(InfFile,temp2^^);
          inc(temp2);
     end;
     close(InfFile);
     assign(CoeffFile,n1+'.TEM');
     reset(CoeffFile);
     getmem(CoeffAddress1,(4*NumOfTem));
     temp1 := CoeffAddress1;
     temp2 := InfAddress1;
     for i := 1 to NumOfTem do
     begin
          getmem(temp1^,(sizeof(coeff)*temp2^^.NumOfFrm));
          temp3 := temp1^;
          for j := 1 to temp2^^.NumOfFrm do
          begin
               read(CoeffFile,temp3^);
               inc(temp3);
          end;
          inc(temp1);
          inc(temp2);
     end;
     close(CoeffFile);
```

```
end;

procedure LoadTestTem(source : byte);
var i,j : integer;
    temp : CoeffPtrType;
    temp1 : InfAddressType;
    temp2 : CoeffAddressType;
begin
     case source of
     1 : begin
              getmem(InfAddress2,40);
              getmem(CoeffAddress2,40);
              temp1 := InfAddress2;
              temp2 := CoeffAddress2;
              chdir(d);
              assign(VoiceFile,FileN);
              reset(VoiceFile);
              for j := 0 to 9 do
              begin
                   percent := j*10;
                   str(percent:5:2,msg);
                   DispStatus(' COMPLETE : '+msg+' % ');
                   read(VoiceFile,voice);
                   boundary;
                   getmem(temp2^,(sizeof(coeff)*LastFrm));
                   temp := temp2^;
                   for i := 1 to LastFrm do
                   begin
                        CalCoeff(i);
                        temp^ := coeff;
                        inc(temp);
                   end;
                   inc(temp2);
                   getmem(temp1^,sizeof(inf));
                   temp1^^.number := j;
                   temp1^^.NumOfFrm := LastFrm;
                   inc(temp1);
              end;
              close(VoiceFile);
         end;
     2 : begin
              receive;
              boundary;
              getmem(CoeffPtr2,(sizeof(coeff)*LastFrm));
              temp := CoeffPtr2;
              for i := 1 to LastFrm do
              begin
                   CalCoeff(i);
                   temp^ := coeff;
                   inc(temp);
              end;
              getmem(InfPtr2,sizeof(inf));
              InfPtr2^.NumOfFrm := LastFrm;
         end;
```

```
     end;
end;

procedure RemoveTemplate(NumOfTem : integer);
var temp1 : CoeffAddressType;
    temp2 : InfAddressType;
    j : integer;
begin
     temp1 := CoeffAddress1;
     temp2 := InfAddress1;
     for j := 1 to NumOfTem do
     begin
          freemem(temp1^,(sizeof(coeff)*temp2^^.NumOfFrm));
          inc(temp1);
          inc(temp2);
     end;
     freemem(CoeffAddress1,(4*NumOfTem));
     temp2 := InfAddress1;
     for j := 1 to NumOfTem do
     begin
          freemem(temp2^,sizeof(inf));
          inc(temp2);
     end;
     freemem(InfAddress1,(4*NumOfTem));
end;

procedure RemoveTestTem(i : byte);
var temp1 : CoeffAddressType;
    temp2 : InfAddressType;
    j : byte;
begin
     case i of
     1 : begin
               temp1 := CoeffAddress2;
               temp2 := Infaddress2;
               for j := 1 to 10 do
               begin
                    freemem(temp1^,(sizeof(coeff)*temp2^^.NumOfFrm));
                    inc(temp1);
                    inc(temp2);
               end;
               freemem(CoeffAddress2,(40));
               temp2 := InfAddress2;
               for j := 1 to 10 do
               begin
                    freemem(temp2^,sizeof(inf));
                    inc(temp2);
               end;
               freemem(InfAddress2,40);
         end;
     2 : begin
               freemem(CoeffPtr2,(sizeof(coeff)*InfPtr2^.NumOfFrm));
               freemem(InfPtr2,sizeof(inf));
         end;
```

```
     end;
end;

procedure input4(x1,y1,x2,y2 : byte);
var i : integer;
begin
     i := 1;
     PrintStr(p,x1,y1,WinColor[1],winColor[2],WinColor[3]);
     PrintStr(p1,x2,y2,WinColor[1],winColor[2],WinColor[3]);
     repeat
     case i of
     1 : ReadStr(p,x1,y1,20);
     2 : ReadStr(p1,x2,y2,20);
     end;
     if key1 = #00 then
     case key2 of
     #72 : begin
               dec(i);
               if i < 1 then i := 2;
           end;
     #80 : begin
               inc(i);
               if i > 2 then i := 2;
           end;
     end;
     until (key1 = #13) or (key1 = #27);
end;

procedure FromFileWin;
begin
     MenuWin(menu[6]);
     with menu[6] do
     begin
          BoxLine(win[1]+1,win[2]+1,win[3]-1,win[4]-4);
          BoxLine(win[1]+1,win[2]+4,win[3]-1,win[4]-1);
     end;
end;

procedure OutputWin(number : byte; x1,y1,x2,y2 : byte);
begin
     Setwin(15,5,norm);
     SetHeader(' OUTPUT ');
     OpenWin(x1,y1,x2,y2);
     PrintStr('RECOG AS NUMBER : ',x1+2,y1+2,WinColor[1],WinColor[2],WinCol
or[3]);
     PrintInt(number,x1+20,y1+2,1);
     repeat until keypressed;
     key3 := readkey;
     if key3 = #00  then key4 := readkey;
     CloseWin;
end;

procedure OutputWin1(x1,y1,x2,y2 : byte);
var j : byte;
```

```
begin
     Setwin(15,5,high);
     SetActive(15,7,low);
     SetHeader(' OUTPUT ');
     OpenWin(x1,y1,x2,y2);
     PrintStr('INPUT NUMBER ',x1+2,y1+2,WinColor[1],WinColor[2],WinColor[3]
);
     PrintStr('RECOG AS NUMBER ',x1+20,y1+2,WinColor[1],WinColor[2],WinColo
r[3]);
     for j := 0 to 9 do
     PrintInt(j,x1+7,y1+4+j,1);
end;

procedure OutputNum(i,number : byte; x1,y1 : byte);
var NumStr : string;
begin
     if number = 10 then NumStr := 'x' else str(number,NumStr);
     if i = number then
     PrintStr(NumStr,x1+26,y1+4+i,15,5,high)
     else
         PrintStr(NumStr,x1+26,y1+4+i,15,1,low);
end;

procedure RecogFromFile;
var i,j,k,NumOfTem : integer;
    temp1,temp3 : CoeffAddressType;
    temp2,temp4 : InfAddressType;
    x,sum : real;
    LoadTest,LoadTem,first : boolean;
    min,min1 : real;
    RecogNum : byte;
    OldP,OldP1 : string;
begin
     FromFileWin;
     LoadTem := false;
     LoadTest := false;
     repeat
     OldP := p;
     OldP1 := p1;
     input4(43,10,43,13);
     fsplit(p,d,n,e);
     fsplit(p1,d1,n1,e1);
     if key1 = #13 then
     begin
     ClearStatus;
     FileN := n+'.VOI';
     FileN1 := n1+'.TEM';
     if DirError(d) then error(1,27,13,58,19)
     else
     if (not FileExist(FileN)) then error(4,27,13,58,19)
     else
         begin
              if DirError(d1) then error(1,27,13,58,19)
              else
```

```
                if (not FileExist(FileN1)) then error(4,27,13,58,19)
                else
                    if (not FileExist(n1+'.INF')) then error(4,27,13,58,1
9)
                    else
                    begin
                         OutputWin1(35,4,71,19);
                         WaitWin(20,16,50,20);
                         if (p <> OldP) or (not LoadTest) then
                         begin
                              WaitMsg(3,20,16,50,20);
                              if LoadTest then RemoveTestTem(1);
                              LoadTestTem(1);
                              LoadTest := true;
                         end;
                         if (p1 <> OldP1) or (not LoadTem) then
                         begin
                              WaitMsg(3,20,16,50,20);
                              if LoadTem then RemoveTemplate(NumofTem);
                              LoadTemplate(NumOfTem);
                              LoadTem := true;
                         end;
                         temp3 := CoeffAddress2;
                         temp4 := InfAddress2;
                         for k := 0 to 9 do
                         begin
                         WaitMsg(4,20,16,50,20);
                         first := true;
                         temp1 := CoeffAddress1;
                         temp2 := InfAddress1;
                         for i := 1 to NumOfTem do
                         begin
                              percent := ((i-1)/NumOfTem)*100;
                              str(percent:5:2,msg);
                              DispStatus(' COMPLETE : '+msg+' % ');
                              CoeffPtr1 := temp1^;
                              InfPtr1 := temp2^;
                              CoeffPtr2 := temp3^;
                              InfPtr2 := temp4^;
                              sum := 0;
                              for j := 1 to order do
                              begin
                                   x := distance(InfPtr1^.NumOfFrm,InfPtr
2^.NumOfFrm,j);
                                   if x = -1 then
                                   begin
                                        sum := -order;
                                        break;
                                   end
                                   else
                                       sum := sum+x;
                              end;
                              min1 := sum/order;
                              if first then
```

```
                                begin
                                     min := min1;
                                     RecogNum := InfPtr1^.number;
                                     first := false;
                                end
                                else
                                begin
                                     if((min<> -1) and (min1 <> -1))  then
                                     begin
                                          if min1 < min then
                                          begin
                                               min := min1;
                                               RecogNum := InfPtr1^.number;
                                          end;
                                     end
                                     else
                                     begin
                                          if min = -1 then
                                          begin
                                               min := min1;
                                               RecogNum := InfPtr1^.number;
                                          end;
                                     end;
                                end;
                                inc(temp1);
                                inc(temp2);
                           end;
                           ClearStatus;
                           inc(temp3);
                           inc(temp4);
                           if min = -1 then RecogNum := 10;
                           OutputNum(k,RecogNum,35,4);
                           end;
                           CloseWin;
                           repeat until keypressed;
                           key3 := readkey;
                           if key3 = #00 then key3 := readkey;
                           CloseWin;
                      end;
          end;
     end;
     chdir(dp);
     until (key1 = #27);
     if LoadTem then RemoveTemplate(NumOfTem);
     if LoadTest then RemoveTestTem(1);
     CloseWin;
     ClearStatus;
end;

procedure FromBoardWin;
begin
     with menu[2] do
     begin
          head := ' RECOG FROM BOARD ';
```

```
          win[1] := 19; win[2] := 10;
          win[3] := 55; win[4] := 14;
          col[1] := 20;
          row[1] := 12;
     end;
     MenuWin(menu[2]);
end;

procedure RecogFromBoard;
var i,j,NumOfTem : integer;
    temp1 : CoeffAddressType;
    temp2 : InfAddressType;
    x,sum : real;
    LoadTem,first : boolean;
    min,min1 : real;
    RecogNum : byte;
    OldP1,OldP2 : string;
begin
     FromBoardWin;
     OldP1 := p;
     repeat
     p := p1;
     OldP2 := p1;
     input1(32,12);
     p1 := p;
     fsplit(p1,d1,n1,e1);
     FileN1 := n1+'.TEM';
     if key1 = #13 then
     begin
          if DirError(d1) then error(1,25,13,56,19)
          else
              if (not FileExist(FileN1)) then error(4,25,13,56,19)
              else
                  if (not FileExist(n1+'.INF')) then error(4,25,13,56,19)
                  else
                  begin
                       WaitWin(28,14,58,18);
                       WaitMsg(1,28,14,58,18);
                       LoadTestTem(2);
                       if (p1 <> OldP2) or (not LoadTem) then
                       begin
                            WaitMsg(3,28,14,58,18);
                            if LoadTem then
                            begin
                                 RemoveTemplate(NumofTem);
                                 RemoveTestTem(2);
                            end;
                            LoadTemplate(NumOfTem);
                            LoadTem := true;
                       end;
                       WaitMsg(4,28,14,58,18);
                       temp1 := CoeffAddress1;
                       temp2 := InfAddress1;
                       first := true;
```

```
                    for i := 1 to NumOfTem do
                    begin
                         percent := ((i-1)/NumOfTem)*100;
                         str(percent:5:2,msg);
                         DispStatus(' COMPLETE : '+msg+' % ');
                         CoeffPtr1 := temp1^;
                         InfPtr1 := temp2^;
                         sum := 0;
                         for j := 1 to order do
                         begin
                              x := distance(InfPtr1^.NumOfFrm,InfPtr2^.N
umOfFrm,j);
                              if x = -1 then
                              begin
                                   sum := -order;
                                   break;
                                   end
                              else
                                  sum := sum+x;
                         end;
                         min1 := sum/order;
                         if first then
                         begin
                              min := min1;
                              RecogNum := InfPtr1^.number;
                              first := false;
                         end
                         else
                         begin
                              if((min<> -1) and (min1 <> -1))  then
                              begin
                                   if min1 < min then
                                   begin
                                        min := min1;
                                        RecogNum := infPtr1^.number;
                                   end;
                              end
                              else
                              begin
                                   if min = -1 then
                                   begin
                                        min := min1;
                                        RecogNum := infPtr1^.number;
                                   end;
                              end;
                         end;
                         inc(temp1);
                         inc(temp2);
                    end;
                    CloseWin;
                    ClearStatus;
                    if min = -1 then error(7,25,13,56,19)
                    else
                        OutputWin(RecogNum,29,13,52,17);
```

```
                end;
    end;
    until (key1 = #27);
    CloseWin;
    if LoadTem then
    begin
        RemoveTemPlate(NumOfTem);
        RemoveTestTem(2);
    end;
    p := OldP1;
end;

procedure SelectSource;
begin
    with menu[5] do
    begin
        active(menu[5],CurAc);
        repeat
            key3 := readkey;
            UnActive(menu[5],CurAc);
            if key3 = #00 then
            begin
                key4 := readkey;
                case key4 of
                #72 : begin
                          dec(CurAc);
                          if CurAC < 1 then CurAc := MaxC;
                      end;
                #80 : begin
                          inc(CurAC);
                          if CurAc > MaxC then CurAc := 1;
                      end;
                end;
            end
            else
            begin
                key3 := upcase(key3);
                case key3 of
                'F' : CurAc := 1;
                'B' : CurAc := 2;
                end;
            end;
            active(menu[5],CurAc);
        until key3 in [#13,#27,'F','B'];
    end;
end;

procedure recog;
var go : boolean;
begin
    go := false;
    MenuWin(menu[5]);
    repeat
    SelectSource;
```

```
     if key3 <> #27 then
     with menu[5] do
     begin
          case CurAc of
          1 : RecogFromFile;
          2 : RecogFromBoard;
          end;
     end
     else
         go := true;
     until go;
     CloseWin;
end;

procedure QuitProgram;
begin
     Quit := true;
end;

begin
     InitSerialPort;
     getdir(0,dp);
     p := 'c:\voice\';
     p1 := 'c:\voice\';
     num := 0;
     background;
     MenuWin(menu[1]);
     repeat
           MainMenu;
           with menu[1] do
           begin
                case CurAc of
                1 : ImportData;
                2 : EditData;
                3 : waveform;
                4 : playback;
                5 : template;
                6 : recog;
                7 : QuitProgram;
                end;
           end;
     until Quit;
     CloseWin;
     CursorOn;
     clrscr;
end.
```

ภาคผนวก ค.

การใช้โปรแกรมวิเคราะห์และรู้จำเสียงพูด

(Recognized)

การใช้โปรแกรม Recognized

1. ลักษณะทั่วไป

โปรแกรม RECOGNIZED เป็นโปรแกรมที่ใช้สำหรับการรู้จำเสียงพูด ซึ่งพัฒนาด้วยภาษาปาส-คาล โดยใช้เทอร์โบปาสคาล เวอร์ชัน 7.0 มีขนาด 54,449 ไบท์ เป็นโปรแกรมที่ยังไม่ได้คอมไพล์เป็นexecutable file (.EXE) ซึ่งผู้ใช้สามารถเข้ามาเปลี่ยนแปลงพารามิเตอร์บางตัวได้เช่น จำนวนเฟรม (frm) จำนวนตัวอย่างใน 1 เฟรม (pxl) หรือจำนวนออร์เดอร์ (order) เป็นต้น การเรียกใช้โปรแกรมจึงต้องเรียกผ่านโปรแกรมเทอร์โบปาสคาลปาสคาล และเรียกแฟ้มข้อมูลชื่อ RECOG.PAS ยูนิตที่ใช้ประกอบด้วย crt, dos, graph, win

2. เซลล์ของโปรแกรม

เซลล์ของโปรแกรมประกอบด้วยเมนูหลัก และมีเมนูย่อยหรือไดอะล็อคบล็อคซ้อนทับกันไป การเลือกเมนูทำได้โดยการกดปุ่มลูกศรขึ้นลงแล้วกด Enter หรือกดปุ่มอักษรตัวแรกของเมนูยกเว้นเมนู QUIT

การออกจากเมนูย่อยหรือไดอะล็อคบล็อค ทำได้โดยการกดปุ่ม Esc ส่วนการออกจากโปร-แกรมหลัก ทำได้โดยเลือกเมนู QUIT หรือกดปุ่ม Alt+X

3. การใช้เมนู IMPORT DATA

โปรแกรมจะสามารถรู้จำเสียงพูดได้โดยการเปรียบเทียบระหว่างแบบอ้างอิง (reference template) ซึ่งได้จากแฟ้มข้อมูลแบบอ้างอิง(.TEM) เพื่อเปรียบเทียบกับแบบทดสอบ (test template) ซึ่งสามารถทำมาได้ 2 ทาง คือ ทางหนึ่ง นำมาจากแฟ้มข้อมูลเสียงพูด (.VOI) ที่ผ่านการหาขอบเขตสัญญาณแล้ว อีกทางหนึ่ง นำมาจากเสียงพูดปัจจุบันที่ผ่านไมโครโฟน ซึ่งสามารถเลือกได้จากเมนู RECOG แต่ในขณะนี้ขอกล่าวถึงการเก็บเสียงพูดโดยใช้เมนู IMPORT DATA ก่อน

เมื่อเลือกเมนู IMPORT DATA จะปรากฏไดอะล็อคบล็อค ให้ป้อนชื่อแฟ้มข้อมูล แล้วกด Enter จะปรากฏไดอะล็อคบล็อคเพื่อบอกว่าให้พูดคำที่เท่าไหร่ (พูดอะไรก็ได้ แต่ในการทดลองพูดคำว่า 'ศูนย์' ถึง 'เก้า') เมื่อพูดผ่านไมโครโฟน (สังเกตไฟสีแดงที่บอร์ดจะสว่าง ซึ่งแสดงว่ามีการส่งข้อมูลจากบอร์ดไปที่ไมโครคอมพิวเตอร์) ให้กดปุ่มอะไรก็ได้ ยกเว้นปุ่ม Esc หรือคีย์ฟังก์ชัน โปรแกรมจะรอรับข้อมูลจนครบ 1 วินาที (สังเกตไฟสีแดงจะดับ) โปรแกรมจะรอรับคำใหม่ เป็นเช่นนี้จนครบ 10 คำ

หากมีการป้อนชื่อแฟ้มข้อมูลที่มีอยู่แล้ว โปรแกรมจะถามว่าต้องการให้ทับแฟ้มข้อมูลเดิมหรือไม่

เมื่อเก็บเสียงพูดครบ 10 คำแล้ว ข้อมูลเสียงเหล่านี้จะเก็บไว้ในไฟล์ .VOI ซึ่งจะมีขนาด 80,000 ไบท์ (คำละ 8,000 ไบท์)

4. <u>การใช้เมนู EDIT DATA</u>

บางครั้งการเก็บเสียงด้วยเมนู IMPORT DATA อาจมีรูปร่างเสียงที่ไม่ดีนัก (ดูได้จากเมนู WAVEFORM) เช่นมีสัญญาณรบกวนมากเกินไป หรือเสียงพูดแปลกแตกต่างจากเสียงพูดปกติ เสียงพูดที่ผิดพลาดเหล่านี้ อาจทำให้การรู้จำผิดพลาดด้วย จึงควรแก้ไขด้วยเมนู EDIT DATA

เมื่อเลือกเมนู EDIT DATA จะปรากฏไดอะล็อคบล็อค เพื่อให้ป้อนชื่อแฟ้มข้อมูลและคำที่จะแก้ไข เมื่อป้อนเสร็จ และกดปุ่ม Enter จะปรากฏไดอะล็อคบล็อคเพื่อให้พูดเช่นเดียวกับการ IMPORT DATA

5. <u>การใช้เมนู WAVEFORM</u>

เพื่อการรู้จำที่ถูกต้อง เมื่อเก็บเสียงด้วยการ IMPORT DATA แล้ว ควรมาดูรูปร่างของสัญญาณเสียงที่เก็บไว้ว่ามีสัญญาณรบกวนเพียงไร มีการหาขอบเขตของสัญญาณถูกต้องหรือไม่ หรือมีรูปร่างเสียงที่ไม่ผิดเกินไปนัก เป็นต้น

เมื่อเลือกเมนู WAVEFORM จะปรากฏไดอะล็อคบล็อค เพื่อให้ป้อนชื่อแฟ้มข้อมูลและคำที่จะขอดูรูปร่างสัญญาณเสียง และกดปุ่ม Enter เซลล์ของโปรแกรมจะหายไปเป็นรูปร่างสัญญาณเสียง 2 รูป รูปบนเป็นสัญญาณเสียงพูดในช่วงเวลา 1 วินาที (8,000 ไบท์) ส่วนรูปล่างเป็นสัญญาณเสียงที่ผ่านการหาขอบเขตสัญญาณแล้ว กดปุ่มอะไรก็ได้เพื่อกลับเข้าสู่เซลล์ของโปรแกรม

6. <u>การใช้เมนู PLAYBACK</u>

นอกจากจะดูความถูกต้องของการเก็บเสียงและการหาขอบเขตสัญญาณด้วยเมนู WAVEFORM แล้ว ยังสามารถฟังได้จากเมนู PLAYBACK ซึ่งจะเล่นเสียงพูดที่ผ่านการหาขอบเขตสัญญาณออกมาทางลำโพงได้อีกด้วย

เมื่อเลือกเมนู PLAYBACK จะปรากฏไดอะล็อคบล็อค เพื่อให้ป้อนชื่อแฟ้มข้อมูลและคำที่ต้องการจะฟัง เมื่อป้อนเสร็จ และกดปุ่ม Enter โปรแกรมจะส่งข้อมูลเสียงพูดที่ผ่านการหาขอบเขตสัญญาณ ไปยังบอร์ดเพื่อแปลงกลับออกมาเป็นเสียงพูด จะสังเกตได้ว่าไฟสีเขียวจสว่าง ซึ่งแสดงว่ามีข้อมูลจากไมโครคอมพิวเตอร์มายังบอร์ด ข้อควรระวังคือ ไม่ควรให้มีการเล่นกลับขณะที่มีการส่งข้อมูลจากบอร์ดไปยังไมโครคอมพิวเตอร์ (ไฟสีแดงสว่าง) ถ้ามีให้กดปุ่มสีแดงเพื่อรีเซ็ทเสียก่อน

7. <u>การใช้เมนู TEMPLATE</u>

เสียงพูดที่ผ่านการหาขอบเขตของสัญญาณ จะถูกนำมาหาสัมประสิทธิ์การทำนายเชิงเส้นเพื่อสร้างเป็นแฟ้มข้อมูลแบบอ้างอิง (.TEM) ตามทฤษฎียิ่งมีจำนวนแบบอ้างอิงมาก นั่นคือมีการสร้างแบบอ้างอิงในแฟ้มข้อมูลแบบอ้างอิงมาก การรู้จำก็จะยิ่งมีความถูกต้องมากยิ่งขึ้น แต่เวลาที่ใช้ก็มากขึ้นด้วย

เมื่อเลือกเมนู TEMPLATE จะปรากฏไดอะล็อคบล็อค เพื่อให้ใส่ชื่อแฟ้มข้อมูลเสียงพูด(.VOI) ที่จะนำมาสร้างเป็นแฟ้มข้อมูลแบบอ้างอิง และป้อนชื่อแฟ้มข้อมูลแบบอ้างอิง(.TEM) ที่จะสร้าง แล้วกดปุ่ม Enter หากแฟ้มข้อมูลแบบอ้างอิงมีการสร้างไว้ก่อน โปรแกรมจะถามว่าต้องการสร้างแฟ้มข้อมูลแบบอ้างอิงใหม่ (NEW) หรือสร้างแบบอ้างอิงเพิ่มเติม (UPDATE) เข้าไปในแฟ้มอิงข้อมูลเดิม ถ้าต้องการสร้างแฟ้มข้อมูลใหม่ให้กดปุ่มอักษร N แต่ถ้าต้องการสร้างแบบอ้างอิงเพิ่มเติมให้กดปุ่มอักษร U

ขณะที่โปรแกรมกำลังสร้างแฟ้มข้อมูลอ้างอิง จะปรากฏข้อความที่มุมซ้ายล่างว่ากำลังสร้างแฟ้มข้อมูลอ้างอิงอยู่เป็นเปอร์เซ็นต์

การใช้เมนู TEMPLATE จะเกิดไฟล์ขึ้นมา 2 ไฟล์ คือ แฟ้มข้อมูลแบบอ้างอิง (.TEM) และแฟ้มข้อมูลดัชนี (.INF) แฟ้มข้อมูลแบบอ้างอิง คือแฟ้มข้อมูลที่เก็บสัมประสิทธิ์การทำนายของแต่ละคำ ส่วนแฟ้มข้อมูลดัชนี คือแฟ้มข้อมูลที่เก็บรายละเอียดของแบบอ้างอิงที่มีอยู่ในแฟ้มข้อมูลแบบอ้างอิง ซึ่งใน 1 คำ จะประกอบด้วย จำนวนเฟรมของคำนั้น 2 ไบท์ ตำแหน่งเริ่มต้นของคำในแฟ้มข้อมูลแบบอ้างอิง 2 ไบท์ และ สัญญลักษณ์ของคำ 1 ไบท์ รวมเป็น 5 ไบท์ ฉะนั้นการสร้างแฟ้มข้อมูลอ้างอิง 1 ครั้ง จะได้แฟ้มข้อมูลดัชนี (.INF) ขนาด 50 ไบท์ ถ้ามีการสร้างเพิ่มเติม แฟ้มข้อมูลแบบอ้างอิงก็จะมีขนาดใหญ่ขึ้น และแฟ้มข้อมูลดัชนีก็จะมีขนาดใหญ่ขึ้นครั้งละ 50 ไบท์

8. <u>การใช้เมนู RECOG</u>

สัมประสิทธิ์การทำนายที่หาได้จะถูกนำมาเปรียบเทียบกันระหว่างแบบอ้างอิงกับแบบทดสอบ ซึ่งแบบอ้างอิงได้จากแฟ้มข้อมูลแบบอ้างอิง (.TEM) และแบบทดสอบสามารถสร้างได้จาก 2 ทาง ดังที่ได้กล่าวไว้แล้ว ประโยชน์ที่แตกต่างกันของทั้ง 2 ทางคือ ถ้าแบบทดสอบมาจากแฟ้มข้อมูลเสียง เสียงพูดจะเหมาะกับการทดลอง แต่ถ้าแบบทดสอบมาจากเสียงพูดปัจจุบันจะเหมาะกับการนำไปประยุกต์ใช้งาน

เมื่อเลือกเมนู RECOG จะปรากฏเมนูย่อย FROM FILE และ FROM BOARD

เมื่อเลือกเมนูย่อย FROM FILE จะเป็นการเปรียบเทียบระหว่างแบบอ้างอิงกับแบบทดสอบที่ได้จากแฟ้มข้อมูลเสียงพูด (.VOI) ซึ่งจะปรากฏไดอะล็อคบล็อคให้ป้อนชื่อแฟ้มข้อมูลเสียงพูด (.VOI) ที่จะนำมาเปรียบเทียบ กับแฟ้มข้อมูลแบบอ้างอิง (.TEM) เมื่อกดปุ่ม Enter จะปรากฏตารางแสดงผลการรู้จำ และมีข้อความทางมุมซ้ายล่างแสดงเปอร์เซ็นต์ของการคำนวณในแต่ละคำ

ตารางแสดงผลการรู้จำ คำที่จำได้ถูกต้องจะแสดงตัวเลขที่ RECOG AS NUMBER ตรงกับ INPUT NUMBER ถ้าไม่ถูกต้องแต่ใกล้เคียงกับคำอื่น ก็จะแสดงตัวเลขที่ใกล้เคียงและมีไฮท์ไลท์ แต่ถ้าไม่ตรงกับคำใดเลย จะแสดงเป็นกากบาท

เมื่อเลือกเมนูย่อย FROM BOARD จะเป็นการเปรียบเทียบระหว่างแบบอ้างอิงกับแบบทดสอบที่ได้จากเสียงพูดปัจจุบันที่ผ่านไมโครโฟน ซึ่งจะปรากฏไดอะล็อคบล็อคให้ป้อนชื่อแฟ้มข้อมูลแบบอ้างอิง (.TEM) กดปุ่ม Enter โปรแกรมจะรอรับข้อมูลจากบอร์ด (ไฟสีแดงจะสว่าง) แล้วทำการเปรียบเทียบและแสดงผลการรู้จำออกมา

ภาคผนวก ง.

รายละเอียดอุปกรณ์ทางอิเลคทรอนิกส์

PRELIMINARY

AUGUST 1988

DATA SHEET 80C51-L / 80C31-L

CMOS SINGLE-CHIP 8 BIT 3V-MICROCONTROLLER

- **80C51-L - CMOS SINGLE-SHIP 8-BIT MICROCONTROLLER with factory mask-programmable ROM**
- **80C31-L - CMOS SINGLE-CHIP 8-BIT CONTROL-ORIENTED CPU with RAM and I/O**
- **80C51-L/C31-L: 0 TO 6 MHz, VCC = 2.7V TO 6V**

FEATURES

- **POWER CONTROL MODES**
- **128 x 8 BIT RAM**
- **32 PROGRAMMABLE I/O LINES**
- **TWO 16-BIT TIMER/COUNTERS**
- **64K PROGRAM MEMORY SPACE**
- **FULLY STATIC DESIGN**
- **HIGH PERFORMANCE SAJI VI CMOS PROCESS**
- **BOOLEAN PROCESSOR**
- **5 INTERRUPT SOURCES**
- **PROGRAMMABLE SERIAL PORT**
- **64K DATA MEMORY SPACE**
- **TEMPERATURE RANGE: 0 TO 70°C**

DESCRIPTION

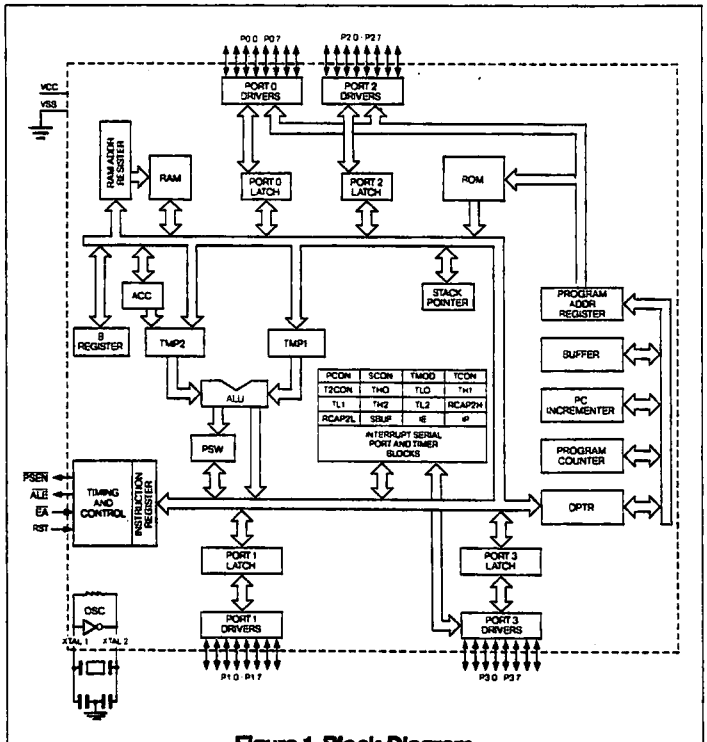


Figure 1. Block Diagram

MHS's 80C51 and 80C31 are high performance CMOS versions of the 8051/8031 NMOS single chip 8 bit μC and is manufactured using a self-aligned silicon gate CMOS process (SAJI VI).

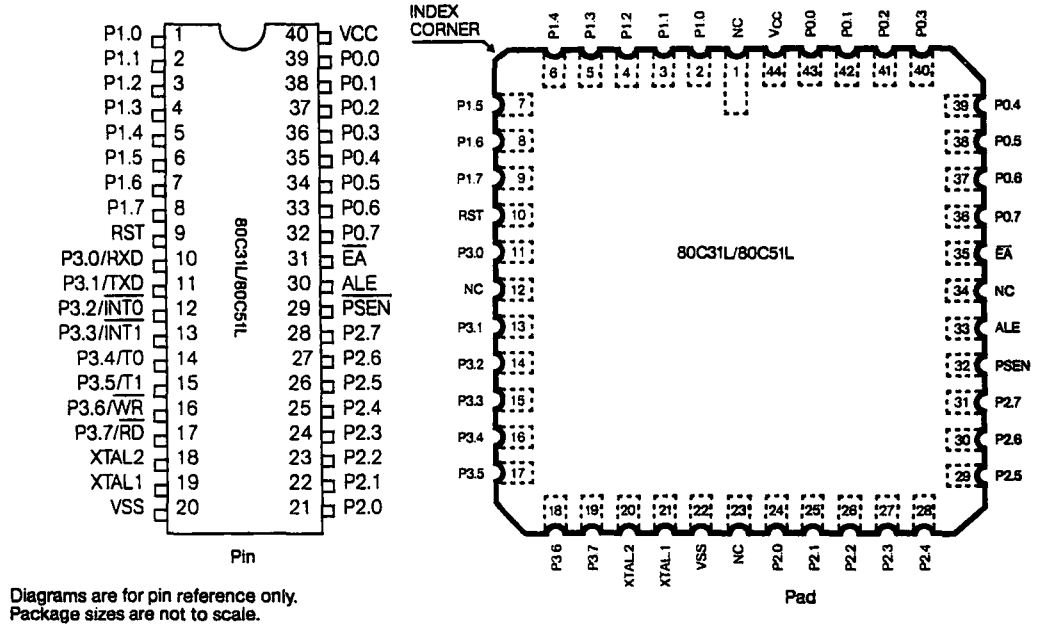
The fully static design of the MHS 80C51/80C31 allows to reduce system power consumption by bringing the clock frequency down to any value, even DC, without loss of data.

The 80C51 retains all the features of the 8051: 4K bytes of ROM; 128 bytes of RAM; 32 I/O lines; two 16 bit timers; a 5-source 2-level interrupt structure; a full duplex serial port; and on-chip oscillator and clock circuits.

In addition, the 80C51 has two software-selectable modes of reduced activity for further reduction in power consumption. In the Idle Mode the CPU is frozen while the RAM, the timers, the serial port, and the interrupt system continue to function. In the Power Down Mode the RAM is saved and all other functions are inoperative.

The 80C31 is identical to the 80C51 except that it has no on-chip ROM.

Figure 2. Configurations



Diagrams are for pin reference only.
Package sizes are not to scale.

IDLE AND POWER DOWN OPERATION

Figure 3 shows the internal Idle and Power Down clock configuration. As illustrated, Power Down operation stops the oscillator. Idle mode operation allows the interrupt, serial port, and timer blocks to continue to function while the clock to the CPU is gated off.
These special modes are activated by software via the Special Function Register, Its hardware address is 87H. PCON is not bit addressable.

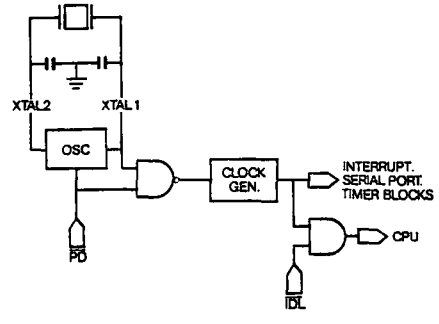


Figure 3. Idle and Power Down Hardware

PCON: Power Control Register

(MSB) (LSB)

SMOD	-	-	-	GF1	GF0	PD	IDL

Symbol	Position	Name and Function
SMOD	PCON.7	Double Baud rate bit. When set to a 1, the baud rate is doubled when the serial port is being used in either modes 1, 2 or 3.
-	PCON.6	(Reserved)
-	PCON.5	(Reserved)
-	PCON.4	(Reserved)
GF1	PCON.3	General-purpose flag bit.
GF0	PCON.2	General-purpose flag bit.
PD	PCON.1	Power Down bit. Setting this bit activates power down operation.
IDL	PCON.0	Ide mode bit.Setting this bit activates idle mode operation.

If 1's are written to PD and IDL at the same time. PD takes precedence. The reset value of PCON is (OXXX0000).

Table 1. Status of the external pins during Idle and Power Down modes

Mode	Program Memory	ALE	PSEN	PORT0	PORT1	PORT2	PORT3
Idle	Internal	1	1	Port Data	Port Data	Port Data	Port Data
Idle	External	1	1	Floating	Port Data	Address	Port Data
Power Down	Internal	0	0	Port Data	Port Data	Port Data	Port Data
Power Down	External	0	0	Floating	Port Data	Port Data	Port Data

IDLE MODE

The instruction that sets PCON.0 is the last instruction executed before the Idle mode is activated. Once in the Idle mode the CPU status is preserved in its entirety: the Stack Pointer, Program Counter, Program Status Word, Accumulator, RAM, and all other registers maintain their data during Idle. *Table 1* describes the status of the external pins during Idle mode.

There are two ways to terminate the Idle mode. Activation of any enabled interrupt will cause PCON.0 to be cleared by hardware, terminating Idle mode. The interrupt is serviced, and following RETI, the next instruction to be executed will be the one following theinstruction that wrote a 1 to PCON.0.

The flag bits GF0 and GF1 may be used to determine whether the interrupt was received during normal execution or during the Idle mode. For example, the instruction that writes to PCON.0 can also set or clear one or both flag bits. When Idle mode is terminated by an enabled interrupt, the service routine can examine the status of the flag bits.

The second way of terminating the Idle mode is with a hardware reset. Since the oscillator is still running, the hardware reset needs to be active for only 2 machine cycles (24 oscillator periods) to complete the reset operation.

POWER DOWN MODE

The instruction that sets PCON.1 is the last executed prior to entering power down. Once in power down, the oscillator is stopped. The contents of the onchip RAM and the Special Function Register is saved during power down mode. A hardware reset is the only way of exiting the power down mode. The hardware reset initiates the Special Function Register *(see Table 1)*.

In the Power Down mode. V_{CC} may be lowered to minimize circuit power consumption. Care must be taken to ensure the voltage is not reduced until the power down mode is entered, and that the voltage is restored before the hardware reset is applied which frees the oscillator. Reset should not be released until the oscillator has restarted and stabilized.

Table 1 describes the status of the external pins while in the power down mode. It should be noted that if the power down mode is activated while in external program memory, the port data that is held in the Special Function Register P2 is restored to Port 2. If the data is a 1, the port pin is held high during the power down mode by the strong pullup, T1, shown in *Figure 4*.

STOP CLOCK MODE

Due to static design, the MHS 80C31/C51 clock speed can be reduced until 0 MHz without any data loss in memory or registers. This mode allows step by step utilization, and permits to reduce system power consumption by bringing the clock frequency down to any value. At 0 MHz, the power consumption is the same as in the Power Down Mode.

80C51 I/O PORTS

The I/O port drive of the 80C51 is similar to the 8051. The I/O buffers for Ports 1, 2, and 3 are implemented as shown in *figure 4*.

When the port latch contains a 0, all pFETS in *figure 4* are off while the nFET is turned on. When the port latch makes a 0-to-1 transition, the nFET turns off. The strong pullup pFET, T1, turns on for two oscillator periods, pulling the output high very rapidly. As the output line is drawn high, pFET T3 turns on through the inverter to supply the I_{OH} source current. This inverter and T3 form a latch which holds the 1 and is supported by T2. When Port 2 is used as an address port, for access to external program of data memory, any address bit that contains a 1 will have his strong pullup turned on for the entire duration of the external memory access.

When an I/O pin on Ports 1, 2, or 3 is used as an input, the user should be aware that the external circuit must sink current during the logical 1-to-0 transition. The maximum sink current is specified as ITL under the D.C.Specifications. When the input goes below approximately 2V, T3 turns off to save ICC current. Note, when returning to a logical 1, T2 is the only internal pullup that is on. This will result in a slow rise time if the user's circuit does not force the input line high.

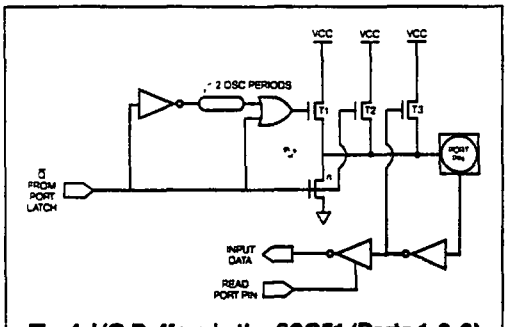


Fig. 4. I/O Buffers in the 80C51 (Ports 1, 2, 3)

80C51 PIN DESCRIPTIONS

V_{SS}
Circuit ground potential

V_{CC}
Supply voltage during normal, Idle, and Power Down operation.

Port 0
Port 0 is an 8-bit open drain bi-directional I/O port. Port 0 pins that have 1's written to them float, and in that state can be used as high-impedance inputs.

Port 0 is also the multiplexed low-order address and data bus during accesses to external Program and Data Memory. In this application it uses strong internal pullups when emitting 1's. Port 0 also outputs the code bytes during program verification in the 80C51. External pullups are required during program verification. Port 0 can sink eight LS TTL inputs.

Port 1
Port 1 is an 8-bit bi-directional I/O port with internal pullups. Port 1 pins that have 1's written to them are pulled high by the internal pullups, and in that state can be used as inputs. As inputs, Port 1 pins that are externally being pulled low will source current (IIL, on the data sheet) because of the internal pullups.

Port 1 also receives the low-order address bytes during program verification. In the 80C51, Port 1 can sink/source three LS TTL inputs. It can drive CMOS inputs without external pullups.

Port 2
Port 2 is an 8-bit bi-directional I/O port with internal pullups. Port 2 pins that have 1's written to them are pulled high by the internal pullups, and in that state can be used as inputs. As inputs, Port 2 pins that are externally being pulled low will source current (IIL, on the data sheet) because of the internal pullups. Port 2 emits the high-order address byte during fetches from external Program Memory and during accesses to external Data Memory that use 16-bit addresses (MOVX @ DPTR). In this application, it uses strong internal pullups when emitting 1's. During accesses to external Data Memory that uses 8-bit addresses (MOVX @ Ri), Port 2 emits the contents of the P2 Special Function Register.

It also receives the high-order address bits and control signals during progam verification in the 80C51. Port 2 can sink/source three LS TTL inputs. It can drive CMOS inputs without external pullups.

Port 3
Port 3 is an 8-bit bi-directional I/O port with internal pullups. Port 3 pins that have 1's written to them are pulled high by the internal pullups, and in that state can be used as inputs. As inputs, Port 3 pins that are externally being pulled low will source current (IIL, on the data sheet) because of the pullups. It also serves the functions of various special features of the MCS-51 Family, as listed below.

Port Pin	Alternate Function
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	INT0 (external interrupt 0)
P3.3	INT1 (external interrupt 1)
P3.4	T0 (Timer 0 external input)
P3.5	T1 (Timer 1 external input)
P3.6	$\overline{WR}$ (external Data Memory write strobe)
P3.7	$\overline{RD}$ (external Data Memory read strobe)

Port 3 can sink/source three LS TTL inputs. It can drive CMOS inputs without external pullups.

RST
A high level on this for two machine cycles while the oscillator is running resets the device. An internal pull-down resistor permits Power-On reset using only a capacitor connected to V_{CC}.

ALE
Address Latch Enable output for latching the low byte of the address during accesses to external memory. ALE is activated as though for this purpose at a constant rate of 1/6 the oscillator frequency except during an external data memory access at which time one ALE pulse is skipped. ALE can sink/source 8 LS TTL inputs. It can drive CMOS inputs without an external pullup.

$\overline{PSEN}$
Program Store Enable output is the read strobe to external Program Memory. PSEN is activated twice each machine cycle during fetches from external Program Memory. (However, when executing out of external Program Memory, two activations of PSEN are skipped during each access to external Data Memory). PSEN is not activated during fetches from internal Program Memory. PSEN can sink/source 8 LS TTL inputs. It can drive CMOS inputs without an external pullup.

$\overline{EA}$
When EA is held high, the CPU executes out of internal Program Memory (unless the Program Counter exceeds OFFFH). When EA is held low, the CPU executes only out of external Program Memory. EA must not be floated.

XTAL1
Input to the inverting amplifier that forms the oscillator. Receives the external oscillator signal when an external oscillator is used.

XTAL2
Output of the inverting amplifier that forms the oscillator, and input to the internal clock generator. This pin should be floated when an external oscillator is used.

OSCILLATOR CHARACTERISTICS

XTAL1 and XTAL2 are the input and output respectively, of an inverting amplifier which is configured for use as an on-chip oscillator, as shown in figure 5. Either a quartz crystal or ceramic resonator may be used.
To drive the device from an external clock source, XTAL1 should be driven while XTAL2 is left unconnected as shown in *figure 6*. There are no requirements on the duty cycle of the external clock signal, since the input to the internal clocking circuitry is through a divide-by-two flip-flop, but munimum and maximum high and low times specified on the Data Sheet must be observed.

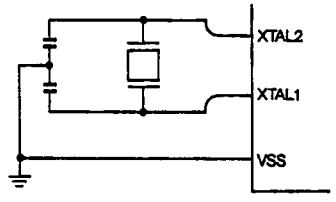


Figure 5. Crystal Oscillator

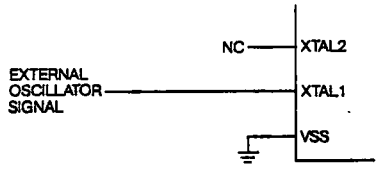


Figure 6. External Drive Configuration

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias:
Commercial ..... 0°C to 70°C
Industrial ..... -40°C to 85°C
Storage Temperature ..... -65°C to +150°C
Voltage on V_{CC} to V_{SS} ..... -0.5V to +7V
Voltage on Any Pin to V SS..... -0.5V to V_{CC} + 0.5V
Power Dissipation 1W*

*This value is based on the maximum allowable die temperature and the thermal resistance of the package.

***NOTICE:**

Stresses at or above 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 above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions may affect device reliability.

DC CHARACTERISTICS

TA = -40°C to 85°C; VCc = 2.7V to 6V; VSS = 0V; F = 0 to 6 MHz

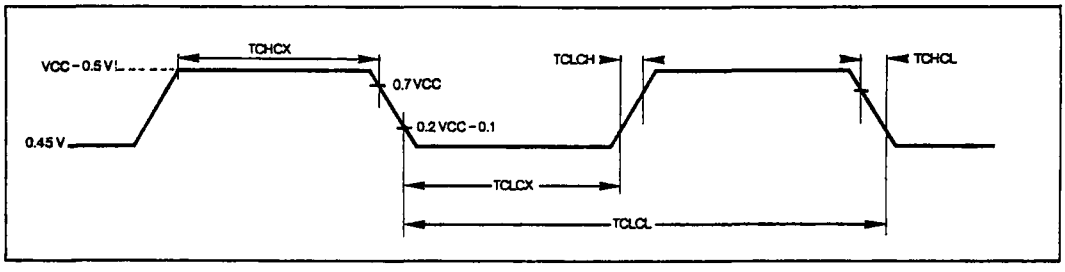
Symbol	Parameter	Min	Max	Unit	Test Conditions
VIL	Input Low Voltage	-0.5	0.2 V_{CC} -0.1	V	
VIH	Input High Voltage (Except XTALs and RST)	0.2 V_{CC} +0.9	V_{CC} +0.5	V	
VIH1	Input High Voltage to RST for Reset	0.7 V_{CC}	V_{CC} +0.5	V	
VIH2	Input High Voltage To XTAL 1	0.7 V_{CC}	V_{CC} +0.5	V	
VPD	Power Down Voltage To V_{CC} in PD Mode	2.0	6.0	V	
VOL	Output Low Voltage (Ports 1, 2, 3)		0.45	V	IOL = 1.6mA (note 1)
VOL1	Output Low Voltage Port 0, ALE, $\overline{PSEN}$		0.45	V	IOL = 3.2mA (note 1)
VOH	Output High Voltage Ports 1, 2, 3	0.9 V_{CC}		V	IOH = -10μa
		2.4		V	IOH = -60μA V_{CC} = 5V ± 10%
VOH1	Output High Voltage (Port 0 in External in External Bus Mode), ALE, $\overline{PSEN}$	0.9 V_{CC}		V	IOH = -40μA
		2.4		V	IOH = -400μA V_{CC} = 5V ± 10%
IIL	Logical 0 Input Current Ports 1,2,3		-50	μA	Vin = 0.45V
ILI	Input Leakage Current		± 10	μA	0.45 < Vin < V_{CC}
ITL	Logical 1 to 0 Transition Current (Ports 1, 2, 3)		-500	μA	Vin = 2.0V
ICCPD	Power Supply Current (Power Down Mode)	50	10	μA	V_{CC} = 2.0V to 5.5V (note 2)
RRST	RST Pulldown Resistor	50	150	kΩ	
CIO	Capacitance of I/O Buffer		10	pF	f_C = 1MHz, T_A = 25°C

Note 1:

Capacitive loading on Ports 0 and 2 may cause spurious noise pulses to be superimposed on the V_{OL}S of ALE and Ports 1 and 3. The noise is due to external bus capacitance discharging into the Port 0 and Port 2 pins when these pins make 1-to-0 transitions during bus operations. In the worst cases (capacitive loading 100 pF), the noise pulse on the ALE line may exceed 0.45V with maxi V_{OL} peak 0.6V. A Schmitt Trigger use is not necessary.

EXTERNAL CLOCK DRIVE CHARACTERISTICS (XTAL 1)

Symbol	Parameter	Variable Clock freq = 0 to 6 MHz		Unit
		Min	Max	
TCLCL	Oscillator Period	166		ns
TCHCX	High Time	20		ns
TCLCX	Low Time	20		ns
TCLCH	Rise Time		20	ns
TCHCL	Fall Time		20	ns



AC CHARACTERISTICS

(T_A = -40°C to 85°C, V_{CC} = 2.7V to 6V, V_{SS} = 0V)
(Load Capacitance for Port 0, ALE, and $\overline{PSEN}$ = 100pf; Load Capacitance for All Other Outputs = 80pf).

EXTERNAL PROGRAM MEMORY CHARACTERISTICS

Symbol	Parameter	Min	Max	Units
TLHLL	ALE Pulse Width	2TCLCL - 40		ns
TAVLL	Address Valid to ALE	TCLCL - 55		ns
TLLAX	Address Hold After ALE	TCLCL - 35		ns
TLLIV	ALE to Valid instr In		4TCLCL - 170	ns
TLLPL	ALE to $\overline{PSEN}$	TCLCL - 25		ns
TPLPH	$\overline{PSEN}$ Pulse Width	3TCLCL - 35		ns
TPLIV	$\overline{PSEN}$ to Valid Instr In		3TCLCL - 220	ns
TPXIX	Input Instr Hold After $\overline{PSEN}$	0		ns
TPXIZ	Input Instr Float After $\overline{PSEN}$		TCLCL - 20	ns
TPXAV	$\overline{PSEN}$ to Address Valid	TCLCL - 8		ns
TAVIV	Address to Valid Instr In		5TCLCL - 220	ns
TPLAZ	$\overline{PSEN}$ Low to Address Float		0	ns

See next page for External Data Memory Characteristics.

EXTERNAL DATA MEMORY CHARACTERISTICS

Symbol	Parameter	Min	Max	Units
TRLRH	$\overline{RD}$ Pulse Width	6TCLCL-100		ns
TWLWH	$\overline{WR}$ Pulse Width	6TCLCL-100		ns
TLLAX	Data Address Hold After ALE	TCLCL-35		ns
TRLDV	$\overline{RD}$ to Valid Data In		5TCLCL-165	ns
TRHDX	Data Hold After $\overline{RD}$	0		ns
TRHDZ	Data Float After $\overline{RD}$		2TCLCL-70	ns
TLLDV	ALE to Valid Data In		8TCLCL-150	ns
TAVDV	Address to Valid Data In		9TCLCL-165	ns
TLLWL	ALE to $\overline{WR}$ or $\overline{RD}$	3TCLCL-50	3TCLCL+50	ns
TAVWL	Address to $\overline{WR}$ or $\overline{RD}$	4TCLCL-130		ns
TQVWX	Data Valid to $\overline{WR}$ Transition	TCLCL-60		ns
TQVWH	Data Setup to $\overline{WR}$ High	7TCLCL-150		ns
TWHQX	Data Hold After $\overline{WR}$	TCLCL-50		ns
TRLAZ	$\overline{RD}$ Low to Address Float		0	ns
TWHLH	$\overline{RD}$ or $\overline{WR}$ High to ALE High	TCLCL-40	TCLCL-40	ns

MAXIMUM I_{CC} (mA)

	Operating (Note 3)			Idle (Note 4)		
Freq. V_{CC}	2.7V	5V	6V	2.7V	5V	6V
1 MHz	0.8 mA	1.5 mA	1.8 mA	400 µA	800 µA	1 mA
6 MHz	4 mA	8 mA	10 mA	1.2 mA	3.5 mA	3.8 mA

Note 2:

Power Down I_{CC} is measured will all output pins disconnected; EA=Port 0 = V_{CC}; XTAL2 N.C.; RST= V_{SS}

Note 3:

I_{CC} is measured with all output pins disconnected; XTAL1 driven with TCLCH, TCHCL = 5 ns, V_{IL} = V_{SS} + 0.5V; V_{IH} V_{CC} - 0.5V; XTAL2 N.C.; EA=RST=Port 0= V_{CC}. I_{CC} would be slightly higher if a crystal oscillator used.

Note 4:

Idle I_{CC} is measured with all output pins disconnected; XTAL1 driven TCLCH, TCHCL=5 ns, V_{IL} = V_{SS} + 0.5V; V_{IH} = V_{CC} - 0.5V; XTAL2 N.C.; Port 0= V_{CC}; EA=RST=V_{SS}.

EXPLANATION OF THE AC SYMBOLS

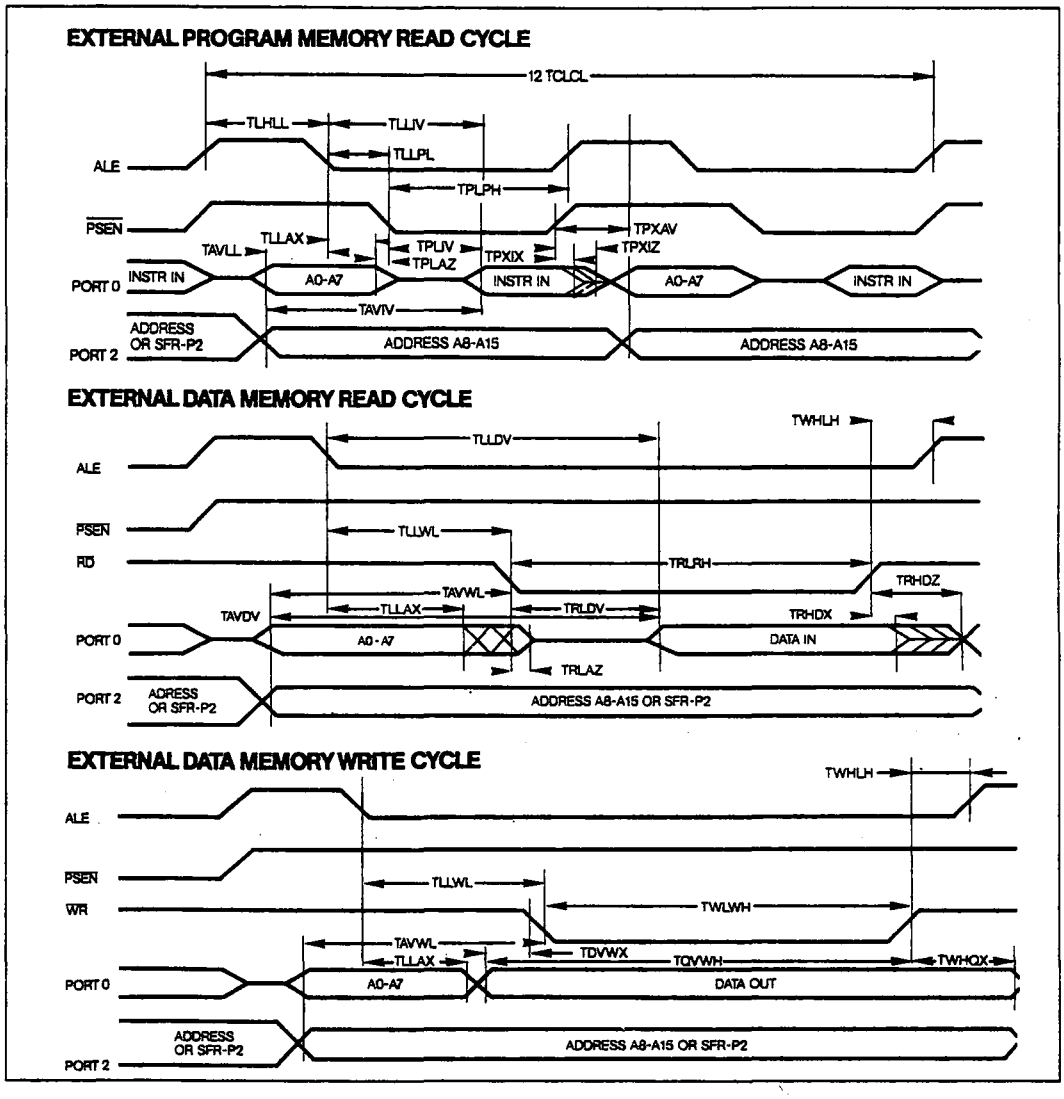
Each timing symbol has 5 characters. The first character is always a 'T' (stands for time). The other characters, depending on their positions, stand for the name of a signal or the logical status of that signal. The following is a list all the characters and what they stand for.

EXAMPLE:

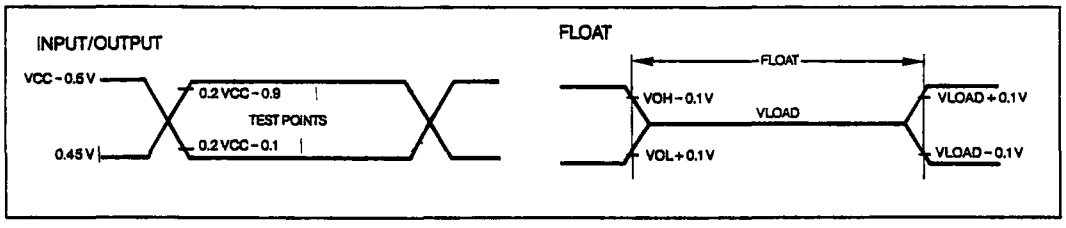
TAVLL=Time for Address Valid to ALE low.
TLLPL=Time for ALE low to $\overline{PSEN}$ low.

A: Address.
C: Clock.
D: Input data.
H: Logic level HIGH.
I: Instruction (program memory contents).
L: Logic level LOW, or ALE.
P: PSEN
Q: Output data.
R: READ signal.
T: Time.
V: Valid.
W: WRITE signal
X: No longer a valid logic level.
Z: Float.

AC TIMING DIAGRAMS



AC TESTING INPUT/OUTPUT, FLOAT WAVEFORMS



AC inputs during testing are driven at V_{CC} – 0.5 for a logic "1" and 0.45V for a logic "0". Timing measurements are made at VIH min for a logic "1" and VIL max for a logic "0". For timing purposes a port pin is no longer floating when a 100 mV change from load voltage occurs and begins to float when a 100 mV change from the loaded VOH/VOL level occurs. Iol/IoH ≥ ± 20 Ma.

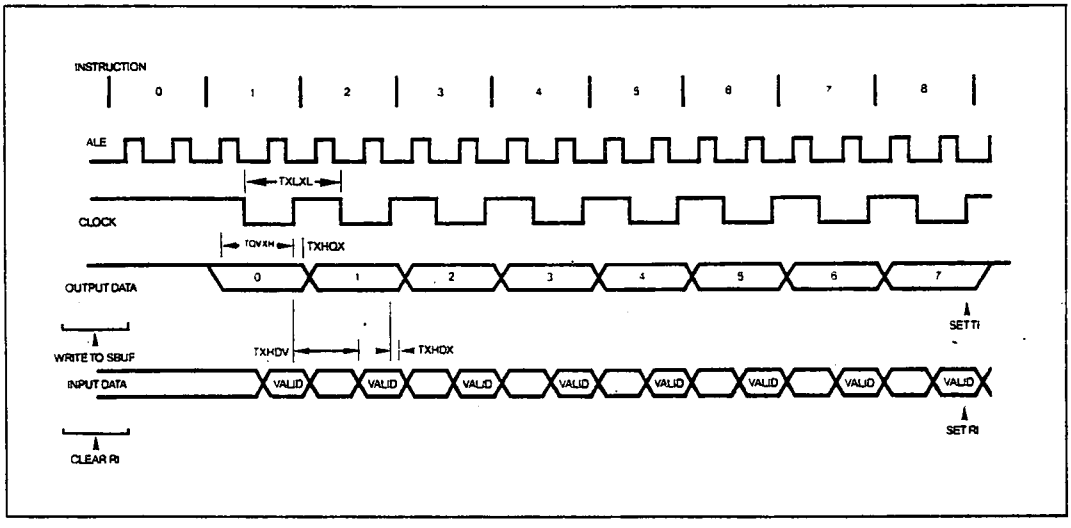
SERIAL PORT TIMING - SHIFT REGISTER MODE

A.C. CHARACTERISTICS:

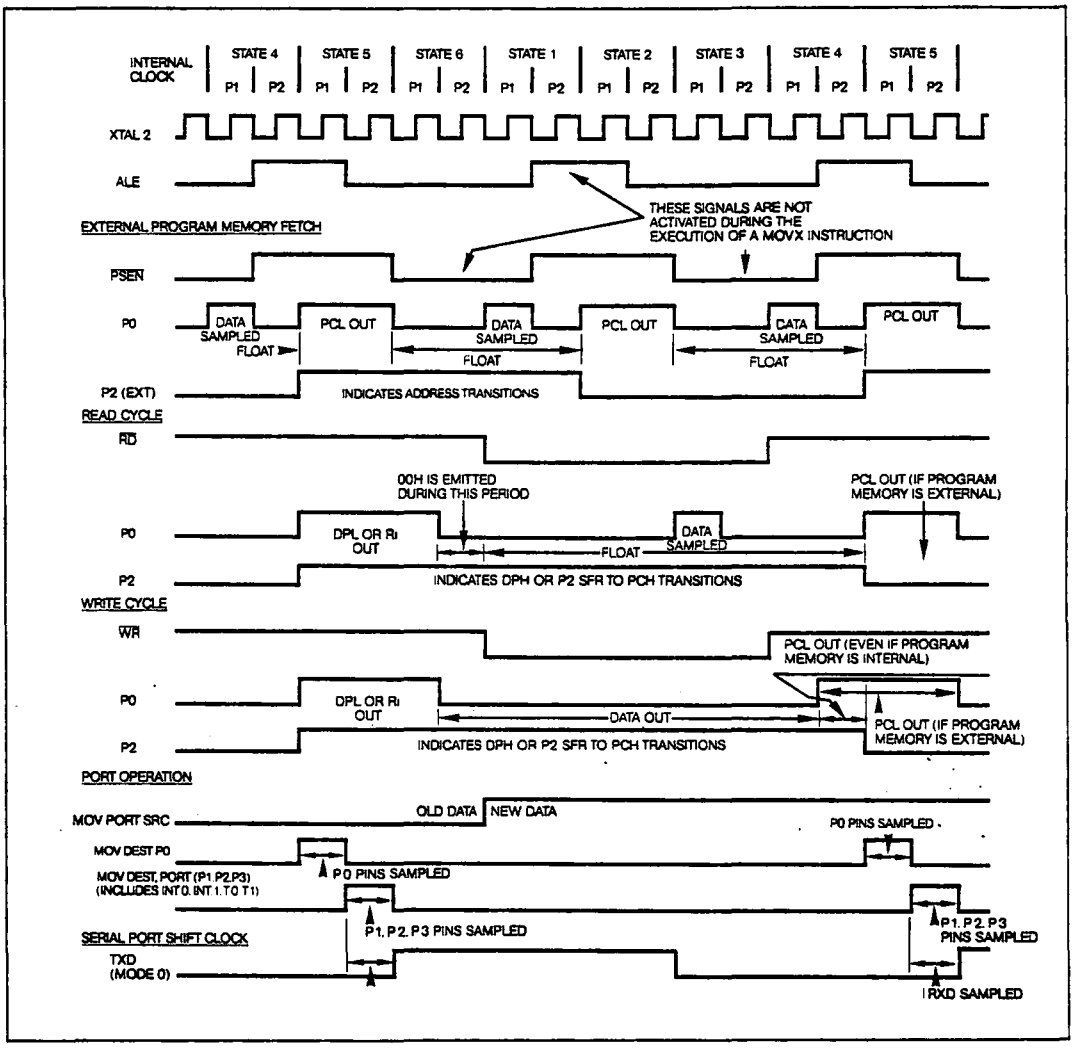
(T_A = 0°C to 70°C; V_{SS} = 0V; V_{CC} = 2.7V to 6V; Load Capacitance = 80 pF)

Symbol	Parameter	Min	Max	Units
TXLXL	Serial Port Clock Cycle Time	12TCLCL		µs
TQVXH	Output Data Setup to Clock Rising Edge	10TCLCL-133		ns
TXHQX	Output Data Hold After Clock Rising Edge	2TCLCL-117		ns
TXHDX	Input Data Hold After Clock Rising Edge	0		ns
TXHDV	Clock Rising Edge to Input Data Valid		10TLCL-133	ns

SHIFT REGISTER TIMING WAVEFORMS



CLOCK WAVEFORMS



This diagram indicates when signals are clocked internally. The time it takes the signals to propagate to the pins, however, ranges from 25 to 125 ns. This propagation delay is dependent on variables such as temperature and pin loading. Propagation also varies from output to output and component. Typically though ($T_A = 25°C$ fully loaded) RD and WR propagation delays are approximately 50 ns. The other signals are typically 85 ns. Propagation delays are incorporated in the AC specifications.

Table 1. MCS©-51 Instruction Set Description

ARITHMETIC OPERATIONS

Mnemonic		Description	Byte	Cyc
ADD	A,Rn	Add register to Accumulator	1	1
ADD	A,direct	Add direct byte to Accumulator	2	1
ADD	A,@Ri	Add indirect RAM to Accumulator	1	1
ADD	A,#data	Add immediate data to Accumulator	2	1
ADDC	A,Rn	Add register to Accumulator with Carry	1	1
ADDC	A,direct	Add direct byte to A with Carry flag	2	1
ADDC	A,@Ri	Add indirect RAM to A with Carry flag	1	1
ADDC	A,#data	Add immediate data to A with Carry flag	2	1
SUBB	A,Rn	Subtract register from A with Borrow	1	1
SUBB	A,direct	Subtract direct byte from A with Borrow	2	1
SUBB	A,@Ri	Subtract indirect RAM from A with Borrow	1	1
SUBB	A,#data	Subtract immed. data from A with Borrow	2	1
INC	A	Increment Accumulator	1	1
INC	Rn	Increment register	1	1
INC	direct	Increment direct byte	2	1
INC	@Ri	Incriment indirect RAM	1	1
INC	DPTR	Incriment Data Pointer	1	2
DEC	A	Decrement Accumulator	1	1
DEC	Rn	Decrement register	1	1
DEC	direct	Decrement direct byte	2	1
DEC	@Ri	Decrement indirect RAM	1	1
MUL	AB	Multiply A & B	1	4
DIV	AB	Divide A by B	1	4
DA	A	Decimal Adjust Accumulator	1	1

LOGICAL OPERATIONS

Mnemonic		Destination	Byte	Cyc
ANL	A,Rn	AND register to Accumulator	1	1
ANL	A,direct	AND direct byte to Accumulator	2	1
ANL	A,@Ri	AND indirect RAM to Accumulator	1	1
ANL	A,#data	AND immediate data to Accumulator	2	1
ANL	direct,A	AND Accumulator to direct byte	2	1
ANL	direct,#data	AND immediate data to direct byte	3	2
ORL	A,Rn	OR register to Accumulator	1	1
ORL	A,direct	OR direct byte to Accumulator	2	1
ORL	A,@Ri	OR indirect RAM to Accumulator	1	1
ORL	A,#data	OR immediate data to Accumulator	2	1
ORL	direct,A	OR Accumulator to direct byte	2	1
ORL	direct,#data	OR immediate data to direct byte	3	2
XRL	A,Rn	Exclusive-OR register to Accumulator	1	1
XRL	A,direct	Exclusive-OR direct byte to Accumulator	2	1
XRL	A,@Ri	Exclusive-OR indirect RAM to A	1	1
XRL	A,#data	Exclusive-OR immediate data to A	2	1
XRL	direct,A	Exclusive-OR Accumulator to direct byte	2	1
XRL	direct,#data	Exclusive-OR immediate data to direct	3	2
CLR	A	Clear Accumulator	1	1
CPL	A	Complement Accumulator	1	1
RL	A	Rotate Accumulator Left	1	1
RLC	A	Rotate A Left through the Carry flag	1	1
RR	A	Rotate Accumulator Right	1	1
RRC	A	Rotate A Right through Carry flag	1	1
SWAP	A	Swap nibbles within the Accumulator	1	1

Table 1. (Cont.)

DATA TRANSFER

Mnemonic		Description	Byte	Cyc
MOV	A,Rn	Move register to Accumulator	1	1
MOV	A,direct	Move direct byte to Accumulator	2	1
MOV	A,@Ri	Move indirect RAM to Accumulator	1	1
MOV	A,#data	Move immediate data to Accumulator	2	1
MOV	Rn,A	Move Accumulator to register	1	1
MOV	Rn,direct	Move direct byte to register	2	2
MOV	Rn,#data	Move immediate data to register	2	1
MOV	direct,A	Move Accumulator to direct byte	2	1
MOV	direct,Rn	Move register to direct byte	2	2
MOV	direct,direct	Move direct byte to direct	3	2
MOV	direct,@Ri	Move indirect RAM to direct byte	2	2
MOV	direct,#data	Move immediate data to direct byte	3	2
MOV	@Ri,A	Move Accumulator to indirect RAM	1	1
MOV	@Ri,direct	Move direct byte to indirect RAM	2	2
MOV	@Ri,#data	Move immediate data to indirect RAM	2	1
MOV	DPTR,#data 16	Load Data Pointer with a 16-bit constant	3	2
MOVC	A,@A+DPTR	Move Code byte relative to DPTR to A	1	2
MOVC	A,@A+PC	Move Code byte relative to PC to A	1	2
MOVX	A,@Ri	Move External RAM (8-bit addr) to A	1	2
MOVX	A,@DPTR	Move External RAM (16-bit addr) to A	1	2
MOVX	@Ri,A	Move A to External RAM (8-bit addr)	1	2
MOVX	@DPTR,A	Move A to External RAM (16-bit addr)	1	2
PUSH	direct	Push direct byte onto stack	2	2
POP	direct	Pop direct byte form stack	2	2
XCH	A,Rn	Exchange register with Accumulator	1	1
XCH	A,direct	Exchange direct byte with Accumulator	2	1
XCH	A,@Ri	Exchange indirect RAM with A	1	1
XCHD	A,@Ri	Exchange low-order nibble ind RAM with A	1	1

BOOLEAN VARIABLE MANIPULATION

Mnemonic		Description	Byte	Cyc
CLR	C	Clear Carry flag	1	1
CLR	bit	Clear direct bit	2	1
SETB	C	Set Carry flag	1	1
SETB	bit	Set direct Bit	2	1
CPL	C	Complement Carry flag	1	1
CPL	bit	Complement direct bit	2	1
ANL	C,bit	AND direct bit to Carry flag	2	2
ANL	C,1 bit	AND complement of direct bit to Carry	2	2
ORL	C/bit	OR direct bit to Carry flag	2	2
ORL	C,1 bit	OR complement of direct bit to Carry	2	2
MOV	C/bit	Move direct bit to Carry flag	2	1
MOV	bit,C	Move Carry flag to direct bit	2	2

PROGRAM AND MACHINE CONTROL

Mnemonic		Description	Byte	Cyc
ACALL	addr 11	Absolute Subroutine Call	2	2
LCALL	addr 16	Long Subroutine Call	3	2
RET		Return from subroutine	1	2
RETI		Return from interrupt	1	2
AJMP	addr 11	Absolute Jump	2	2
LJMP	addr 16	Long Jump	3	2
SJMP	rel	Short Jump (relative addr)	2	2
JMP	@A+DPTR	Jump indirect relative to the DPTR	1	2
JZ	rel	Jump if Accumulator is Zero	2	2
JNZ	rel	Jump if Accumulator is Not Zero	2	2
JC	rel	Jump if Carry flag is set	2	2
JNC	rel	Jump if No Carry flag	2	2

Table 1. (Cont.)

PROGRAM AND MACHINE CONTROL (cont.)

Mnemonic		Description	Byte	Cyc
JB	bit,rel	Jump if direct Bit set	3	2
JNB	bit,rel	Jump if direct Bit Not set	3	2
JBC	bit,rel	Jump if direct Bit is set & Clear bit	3	2
CJNE	A,direct,rel	Compare direct to A & Jump if Not Equal	3	2
CJNE	A,#data,rel	Comp. immed. to A & Jump if Not Equal	3	2
CJNE	Rn,#data,rel	Comp. immed. to reg & Jump if Not Equal	3	2
CJNE	@Ri,#data. rel	Comp. immed. to ind. & Jump if Not Equal	3	2
DJNZ	Rn,rel	Decrement register & Jump if Not Zero	2	2
DJNZ	direct. rel	Decrement direct & Jump if Not Zero	3	2
NOP		No operation	1	1

Notes on data addressing modes:

Rn	– Working register R0-R7
direct	– 128 internal RAM locations, any I/O port, control or status register
@Ri	– Indirect internal RAM location addressed by register R0 or R1
#data	– 8-bit constant included in instruction
#data 16	– 16-bit constant included as bytes 2 & 3 of instruction
bit	– 128 software flags, any I/O pin, control or status bit

Notes on program addressing modes:

addr 16	– Destination address for LCALL & LJMP may be anywhere within the 64-k program memory address space
Addr 11	– Destination address for ACALL & AJMP will be within the same 2-k page of program memory as the first byte of the following instruction
rel	– SJMP and all conditional jumps include an 8-bit offset byte. Range is +127–128 bytes relative to first byte of the following instruction.

Table 2. Instruction Opcodes in Hexadecimal Order

Hex Code	Number of Bytes	Mnemonic	Operands
00	1	NOP	
01	2	AJMP	code addr
02	3	LJMP	code addr
03	1	RR	A
04	1	INC	A
05	2	INC	data addr
06	1	INC	@R0
07	1	INC	@R1
08	1	INC	R0
09	1	INC	R1
0A	1	INC	R2
0B	1	INC	R3
0C	1	INC	R4
0D	1	INC	R5
OE	1	INC	R6
0F	1	INC	R7
10	3	JBC	bit addr,code addr
11	2	ACALL	code addr
12	3	LCALL	code addr
13	1	RRC	A
14	1	DEC	A
15	2	DEC	data addr
16	1	DEC	@R0
17	1	DEC	@R1
18	1	DEC	R0
19	1	DEC	R1
1A	1	DEC	R2
1B	1	DEC	R3
1C	1	DEC	R4
1D	1	DEC	R5
1E	1	DEC	R6
1F	1	DEC	R7
20	3	JB	bit addr,code addr
21	2	AJMP	code addr
22	1	RET	
23	1	RL	A
24	2	ADD	A,data
25	2	ADD	A,data addr
26	1	ADD	A,@R0
27	1	ADD	A,@R1
28	1	ADD	A,R0
29	1	ADD	A,R1
2A	1	ADD	A,R2
2B	1	ADD	A,R3
2C	1	ADD	A,R4
2D	1	ADD	A,R5
2E	1	ADD	A,R6
2F	1	ADD	A,R7
30	3	JNB	bit addr,code addr
31	2	ACALL	code addr
32	1	RETI	

Hex Code	Number of Bytes	Mnemonic	Operands
33	1	RLC	A
34	2	ADDC	A,#data
35	2	ADDC	A,data addr
36	1	ADDC	A,@R0
37	1	ADDC	A,@R1
38	1	ADDC	A,R0
39	1	ADDC	A,R1
3A	1	ADDC	A,R2
3B	1	ADDC	A,R3
3C	1	ADDC	A,R4
3D	1	ADDC	A,R5
3E	1	ADDC	A,R6
3F	1	ADDC	A,R7
40	2	JC	code addr
41	2	AJMP	code addr
42	2	ORL	data addr,A
43	3	ORL	data addr,#data
44	2	ORL	A,#data
45	2	ORL	A,data addr
46	1	ORL	A,@R0
47	1	ORL	A,@R1
48	1	ORL	A,R0
49	1	ORL	A,R1
4A	1	ORL	A,R2
4B	1	ORL	A,R3
4C	1	ORL	A,R4
4D	1	ORL	A,R5
4E	1	ORL	A,R6
4F	1	ORL	A,R7
50	2	JNC	code addr
51	2	ACALL	code addr
52	2	ANL	data addr,A
53	3	ANL	data addr,#data
54	2	ANL	A,#data
55	2	ANL	A,data addr
56	1	ANL	A,@R0
57	1	ANL	A,@R1
58	1	ANL	A,R0
59	1	ANL	A,R1
5A	1	ANL	A,R2
5B	1	ANL	A,R3
5C	1	ANL	A,R4
5D	1	ANL	A,R5
5E	1	ANL	A,R6
5F	1	ANL	A,R7
60	2	JZ	code addr
61	2	AJMP	code addr
62	2	XRL	data addr A
63	3	XRL	data addr,#data
64	2	XRL	A,#data
65	2	XRL	A,data addr

Table 2. (Cont.)

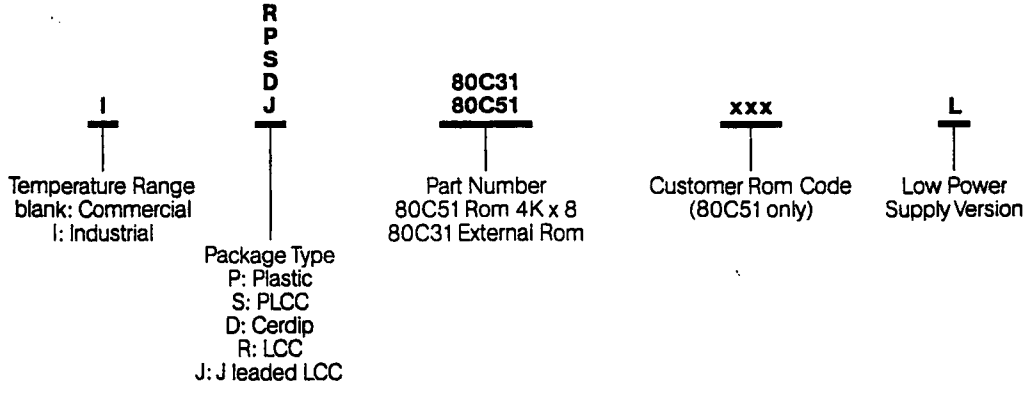
Hex Code	Number of Bytes	Mnemonic	Operands
66	1	XRL	A,@R0
67	1	XRL	A,@R1
68	1	XRL	A,R0
69	1	XRL	A,R1
6A	1	XRL	A,R2
6B	1	XRL	A,R3
6C	1	XRL	A,R4
6D	1	XRL	A,R5
6E	1	XRL	A,R6
6F	1	XRL	A,R7
70	2	JNZ	code addr
71	2	ACALL	code addr
72	2	ORL	C, bit addr
73	1	JMP	@A+DPTR
74	2	MOV	A,#data
75	3	MOV	data addr,#data
76	2	MOV	@R0,#data
77	2	MOV	@R1,#data
78	2	MOV	R0,#data
79	2	MOV	R1,#data
7A	2	MOV	R2,#data
7B	2	MOV	R3,#data
7C	2	MOV	R4,#data
7D	2	MOV	R5,#data
7E	2	MOV	R6,#data
7F	2	MOV	R7,#data
80	2	SJMP	code addr
81	2	AJMP	code addr
82	2	ANL	C,bit addr
83	1	MOVC	A,@A+PC
84	1	DIV	AB
85	3	MOV	data addr,data addr
86	2	MOV	data addr,@R0
87	2	MOV	data addr,@R1
88	2	MOV	data addr,R0
89	2	MOV	data addr,R1
8A	2	MOV	data addr,R2
8B	2	MOV	data addr,R3
8C	2	MOV	data addr,R4
8D	2	MOV	data addr,R5
8E	2	MOV	data addr,R6
8F	2	MOV	data addr,R7
90	3	MOV	DPTR,#data
91	2	ACALL	code addr
92	2	MOV	bit addr,C
93	1	MOVC	A,@A+DPTR
94	2	SUBB	A,#data
95	2	SUBB	A,data addr
96	1	SUBB	A,@R0
97	1	SUBB	A,@R1
98	1	SUBB	A,R0

Hex Code	Number of Bytes	Mnemonic	Operands
99	1	SUBB	A,R1
9A	1	SUBB	A,R2
9B	1	SUBB	A,R3
9C	1	SUBB	A,R4
9D	1	SUBB	A,R5
9E	1	SUBB	A,R6
9F	1	SUBB	A,R7
A0	2	ORL	C,bit addr
A1	2	AJMP	code addr
A2	2	MOV	C,bit addr
A3	1	INC	DPTR
A4	1	MUL	AB
A5		reserved	
A6	2	MOV	@R0,data addr
A7	2	MOV	@R1,data addr
A8	2	MOV	R0,data addr
A9	2	MOV	R1,data addr
AA	2	MOV	R2,data addr
AB	2	MOV	R3,data addr
AC	2	MOV	R4,data addr
AD	2	MOV	R5,data addr
AE	2	MOV	R6,data addr
AF	2	MOV	R7,data addr
B0	2	ANL	C,bit addr
B1	2	ACALL	code addr
B2	2	CPL	bit addr
B3	1	CPL	C
B4	3	CJNE	A,#data,code addr
B5	3	CJNE	A,data addr,code addr
B6	3	CJNE	@R0,#data,code addr
B7	3	CJNE	@R1,#data,code addr
B8	3	CJNE	R0,#data,code addr
B9	3	CJNE	R1,#data,code addr
BA	3	CJNE	R2,#data,code addr
BB	3	CJNE	R3,#data,code addr
BC	3	CJNE	R4,#data,code addr
BD	3	CJNE	R5,#data,code addr
BE	3	CJNE	R6,#data,code addr
BF	3	CJNE	R7,#data,code addr
C0	2	PUSH	data addr
C1	2	AJMP	code addr
C2	2	CLR	bit addr
C3	1	CLR	C
C4	1	SWAP	A
C5	2	XCH	A,data addr
C6	1	XCH	A,@R0
C7	1	XCH	A,@R1
C8	1	XCH	A,R0
C9	1	XCH	A,R1
CA	1	XCH	A,R2
CB	1	XCH	A,R3

Table 2. (Cont.)

Hex Code	Number of Bytes	Mnemonic	Operands
CC	1	XCH	A,R4
CD	1	XCH	A,R5
CE	1	XCH	A,R6
CF	1	XCH	A,R7
D0	2	POP	data addr
D1	2	ACALL	code addr
D2	2	SETB	bit addr
D3	1	SETB	C
D4	1	DA	A
D5	3	DJNZ	data addr,code addr
D6	1	XCHD	A,@R0
D7	1	XCHD	A,@R1
D8	2	DJNZ	R0,code addr
D9	2	DJNZ	R1,code addr
DA	2	DJNZ	R2,code addr
DB	2	DJNZ	R3,code addr
DC	2	DJNZ	R4,code addr
DD	2	DJNZ	R5,code addr
DE	2	DJNZ	R6,code addr
DF	2	DJNZ	R7,code addr
E0	1	MOVX	A,@DPTR
E1	2	AJMP	code addr
E2	1	MOVX	A,@R0
E3	1	MOVX	A,@R1
E4	1	CLR	A
E5	2	MOV	A,data addr

Hex Code	Number of Bytes	Mnemonic	Operands
E6	1	MOV	A,@R0
E7	1	MOV	A,@R1
E8	1	MOV	A,R0
E9	1	MOV	A,R1
EA	1	MOV	A,R2
EB	1	MOV	A,R3
EC	1	MOV	A,R4
ED	1	MOV	A,R5
EE	1	MOV	A,R6
EF	1	MOV	A,R7
F0	1	MOVX	@DPTR,A
F1	2	ACALL	code addr
F2	1	MOVX	@R0,A
F3	1	MOVX	@R1,A
F4	1	CPL	A
F5	2	MOV	data addr,A
F6	1	MOV	@R0,A
F7	1	MOV	@R1,A
F8	1	MOV	R0,A
F9	1	MOV	R1,A
FA	1	MOV	R2,A
FB	1	MOV	R3,A
FC	1	MOV	R4,A
FD	1	MOV	R5,A
FE	1	MOV	R6,A
FF	1	MOV	R7,A



National Semiconductor

ADC0808/ADC0809 8-Bit μP Compatible A/D Converters with 8-Channel Multiplexer

General Description

The ADC0808, ADC0809 data acquisition component is a monolithic CMOS device with an 8-bit analog-to-digital converter, 8-channel multiplexer and microprocessor compatible control logic. The 8-bit A/D converter uses successive approximation as the conversion technique. The converter features a high impedance chopper stabilized comparator, a 256R voltage divider with analog switch tree and a successive approximation register. The 8-channel multiplexer can directly access any of 8-single-ended analog signals.

The device eliminates the need for external zero and full-scale adjustments. Easy interfacing to microprocessors is provided by the latched and decoded multiplexer address inputs and latched TTL TRI-STATE® outputs.

The design of the ADC0808, ADC0809 has been optimized by incorporating the most desirable aspects of several A/D conversion techniques. The ADC0808, ADC0809 offers high speed, high accuracy, minimal temperature dependence, excellent long-term accuracy and repeatability, and consumes minimal power. These features make this device ideally suited to applications from process and machine control to consumer and automotive applications. For 16-channel multiplexer with common output (sample/hold port) see ADC0816 data sheet. (See AN-247 for more information.)

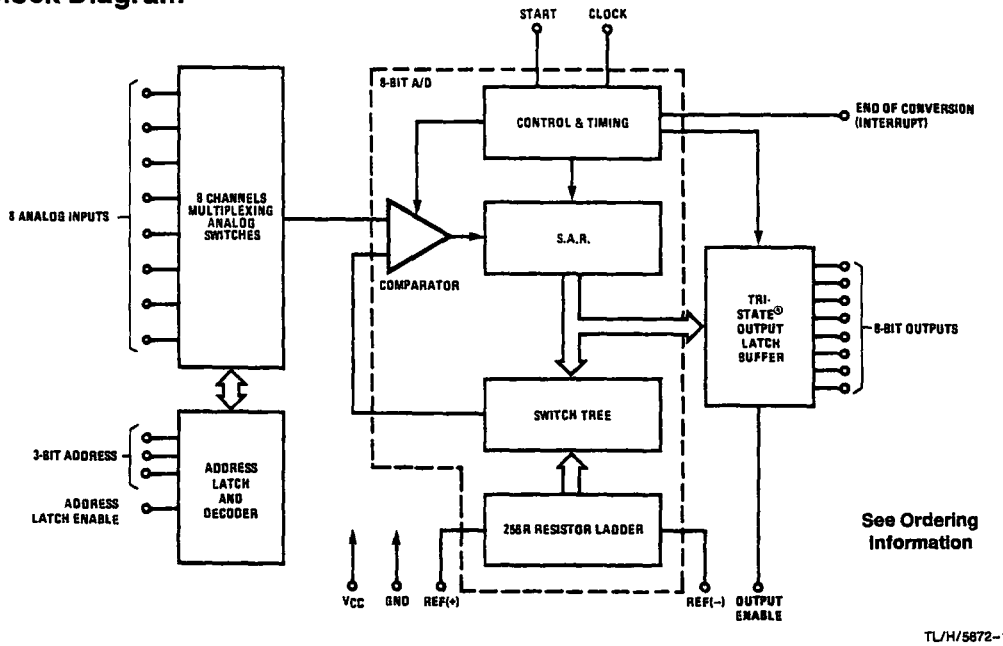
Features

- Easy interface to all microprocessors
- Operates ratiometrically or with 5 V_{DC} or analog span adjusted voltage reference
- No zero or full-scale adjust required
- 8-channel multiplexer with address logic
- 0V to 5V input range with single 5V power supply
- Outputs meet TTL voltage level specifications
- Standard hermetic or molded 28-pin DIP package
- 28-pin molded chip carrier package
- ADC0808 equivalent to MM74C949
- ADC0809 equivalent to MM74C949-1

Key Specifications

■ Resolution	8 Bits
■ Total Unadjusted Error	±½ LSB and ±1 LSB
■ Single Supply	5 V_{DC}
■ Low Power	15 mW
■ Conversion Time	100 μs

Block Diagram



See Ordering Information

TL/H/5872–1

Absolute Maximum Ratings (Notes 1 & 2)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Supply Voltage (V_{CC}) (Note 3)	6.5V
Voltage at Any Pin Except Control Inputs	−0.3V to (V_{CC} + 0.3V)
Voltage at Control Inputs (START, OE, CLOCK, ALE, ADD A, ADD B, ADD C)	−0.3V to +15V
Storage Temperature Range	−65°C to +150°C
Package Dissipation at T_A = 25°C	875 mW
Lead Temp. (Soldering, 10 seconds)	
Dual-In-Line Package (plastic)	260°C
Dual-In-Line Package (ceramic)	300°C
Molded Chip Carrier Package	
Vapor Phase (60 seconds)	215°C
Infrared (15 seconds)	220°C
ESD Susceptibility (Note 11)	400V

Operating Conditions (Notes 1 & 2)

Temperature Range (Note 1)	$T_{MIN} \le T_A \le T_{MAX}$
ADC0808CJ	−55°C ≤ T_A ≤ +125°C
ADC0808CCJ, ADC0808CCN, ADC0809CCN	−40°C ≤ T_A ≤ +85°C
ADC0808CCV, ADC0809CCV	−40°C ≤ T_A ≤ +85°C
Range of V_{CC} (Note 1)	4.5 V_{DC} to 6.0 V_{DC}

Electrical Characteristics

Converter Specifications: V_{CC} = 5 V_{DC} = V_{REF+}, $V_{REF(-)}$ = GND, $T_{MIN} \le T_A \le T_{MAX}$ and f_{CLK} = 640 kHz unless otherwise stated.

Symbol	Parameter	Conditions	Min	Typ	Max	Units
	ADC0808					
	Total Unadjusted Error	25°C			±½	LSB
	(Note 5)	T_{MIN} to T_{MAX}			±¾	LSB
	ADC0809					
	Total Unadjusted Error	0°C to 70°C			±1	LSB
	(Note 5)	T_{MIN} to T_{MAX}			±1¼	LSB
	Input Resistance	From Ref(+) to Ref(−)	1.0	2.5		kΩ
	Analog Input Voltage Range	(Note 4) V(+) or V(−)	GND−0.10		V_{CC}+0.10	V_{DC}
$V_{REF(+)}$	Voltage, Top of Ladder	Measured at Ref(+)		V_{CC}	V_{CC}+0.1	V
$\frac{V_{REF(+)}+V_{REF(-)}}{2}$	Voltage, Center of Ladder		V_{CC}/2-0.1	V_{CC}/2	V_{CC}/2+0.1	V
$V_{REF(-)}$	Voltage, Bottom of Ladder	Measured at Ref(−)	−0.1	0		V
I_{IN}	Comparator Input Current	f_c = 640 kHz, (Note 6)	−2	±0.5	2	μA

Electrical Characteristics

Digital Levels and DC Specifications: ADC0808CJ 4.5V ≤ V_{CC} ≤ 5.5V, −55°C ≤ T_A ≤ +125°C unless otherwise noted ADC0808CCJ, ADC0808CCN, ADC0808CCV, ADC0809CCN and ADC0809CCV, 4.75 ≤ V_{CC} ≤ 5.25V, −40°C ≤ T_A ≤ +85°C unless otherwise noted

Symbol	Parameter	Conditions	Min	Typ	Max	Units
ANALOG MULTIPLEXER						
$I_{OFF(+)}$	OFF Channel Leakage Current	V_{CC} = 5V, V_{IN} = 5V,				
		T_A = 25°C		10	200	nA
		T_{MIN} to T_{MAX}			1.0	μA
$I_{OFF(-)}$	OFF Channel Leakage Current	V_{CC} = 5V, V_{IN} = 0,				
		T_A = 25°C	−200	−10		nA
		T_{MIN} to T_{MAX}	−1.0			μA

Electrical Characteristics (Continued)

Digital Levels and DC Specifications: ADC0808CJ $4.5V \le V_{CC} \le 5.5V$, $-55°C \le T_A \le +125°C$ unless otherwise noted ADC0808CCJ, ADC0808CCN, ADC0808CCV, ADC0809CCN and ADC0809CCV, $4.75 \le V_{CC} \le 5.25V$, $-40°C \le T_A \le +85°C$ unless otherwise noted

Symbol	Parameter	Conditions	Min	Typ	Max	Units
CONTROL INPUTS						
$V_{IN(1)}$	Logical "1" Input Voltage		$V_{CC}-1.5$			V
$V_{IN(0)}$	Logical "0" Input Voltage				1.5	V
$I_{IN(1)}$	Logical "1" Input Current (The Control Inputs)	$V_{IN}=15V$			1.0	μA
$I_{IN(0)}$	Logical "0" Input Current (The Control Inputs)	$V_{IN}=0$	−1.0			μA
I_{CC}	Supply Current	$f_{CLK}=640$ kHz		0.3	3.0	mA
DATA OUTPUTS AND EOC (INTERRUPT)						
$V_{OUT(1)}$	Logical "1" Output Voltage	$I_O=-360$ μA	$V_{CC}-0.4$			V
$V_{OUT(0)}$	Logical "0" Output Voltage	$I_O=1.6$ mA			0.45	V
$V_{OUT(0)}$	Logical "0" Output Voltage EOC	$I_O=1.2$ mA			0.45	V
I_{OUT}	TRI-STATE Output Current	$V_O=5V$ $V_O=0$	 −3		3	μA μA

Electrical Characteristics

Timing Specifications $V_{CC}=V_{REF(+)}=5V$, $V_{REF(-)}=GND$, $t_r=t_f=20$ ns and $T_A=25°C$ unless otherwise noted.

Symbol	Parameter	Conditions	Min	Typ	Max	Units
t_{WS}	Minimum Start Pulse Width	*(Figure 5)*		100	200	ns
t_{WALE}	Minimum ALE Pulse Width	*(Figure 5)*		100	200	ns
t_S	Minimum Address Set-Up Time	*(Figure 5)*		25	50	ns
t_H	Minimum Address Hold Time	*(Figure 5)*		25	50	ns
t_D	Analog MUX Delay Time From ALE	$R_S=0\Omega$ *(Figure 5)*		1	2.5	μS
t_{H1}, t_{H0}	OE Control to Q Logic State	$C_L=50$ pF, $R_L=10$k *(Figure 8)*		125	250	ns
t_{1H}, t_{0H}	OE Control to Hi-Z	$C_L=10$ pF, $R_L=10$k *(Figure 8)*		125	250	ns
t_C	Conversion Time	$f_C=640$ kHz, *(Figure 5)* (Note 7)	90	100	116	μS
f_C	Clock Frequency		10	640	1280	kHz
t_{EOC}	EOC Delay Time	*(Figure 5)*	0		8+2 μS	Clock Periods
C_{IN}	Input Capacitance	At Control Inputs		10	15	pF
C_{OUT}	TRI-STATE Output Capacitance	At TRI-STATE Outputs, (Note 12)		10	15	pF

Note 1: Absolute Maximum Ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications do not apply when operating the device beyond its specified operating conditions.

Note 2: All voltages are measured with respect to GND, unless otherwise specified.

Note 3: A zener diode exists, internally, from V_{CC} to GND and has a typical breakdown voltage of 7 V_{DC}.

Note 4: Two on-chip diodes are tied to each analog input which will forward conduct for analog input voltages one diode drop below ground or one diode drop greater than the V_{CC} supply. The spec allows 100 mV forward bias of either diode. This means that as long as the analog V_{IN} does not exceed the supply voltage by more than 100 mV, the output code will be correct. To achieve an absolute $0V_{DC}$ to $5V_{DC}$ input voltage range will therefore require a minimum supply voltage of 4.900 V_{DC} over temperature variations, initial tolerance and loading.

Note 5: Total unadjusted error includes offset, full-scale, linearity, and multiplexer errors. See *Figure 3.* None of these A/Ds requires a zero or full-scale adjust. However, if an all zero code is desired for an analog input other than 0.0V, or if a narrow full-scale span exists (for example: 0.5V to 4.5V full-scale) the reference voltages can be adjusted to achieve this. See *Figure 13.*

Note 6: Comparator input current is a bias current into or out of the chopper stabilized comparator. The bias current varies directly with clock frequency and has little temperature dependence *(Figure 6).* See paragraph 4.0.

Note 7: The outputs of the data register are updated one clock cycle before the rising edge of EOC.

Note 8: Human body model, 100 pF discharged through a 1.5 kΩ resistor.

Functional Description

Multiplexer. The device contains an 8-channel single-ended analog signal multiplexer. A particular input channel is selected by using the address decoder. Table I shows the input states for the address lines to select any channel. The address is latched into the decoder on the low-to-high transition of the address latch enable signal.

TABLE I

SELECTED ANALOG CHANNEL	ADDRESS LINE		
	C	B	A
IN0	L	L	L
IN1	L	L	H
IN2	L	H	L
IN3	L	H	H
IN4	H	L	L
IN5	H	L	H
IN6	H	H	L
IN7	H	H	H

CONVERTER CHARACTERISTICS

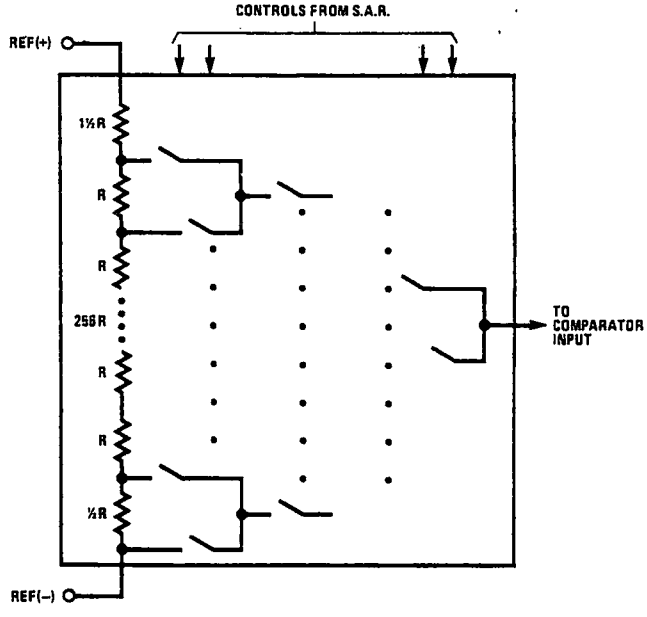
The Converter

The heart of this single chip data acquisition system is its 8-bit analog-to-digital converter. The converter is designed to give fast, accurate, and repeatable conversions over a wide range of temperatures. The converter is partitioned into 3 major sections: the 256R ladder network, the successive approximation register, and the comparator. The converter's digital outputs are positive true.

The 256R ladder network approach *(Figure 1)* was chosen over the conventional R/2R ladder because of its inherent monotonicity, which guarantees no missing digital codes. Monotonicity is particularly important in closed loop feedback control systems. A non-monotonic relationship can cause oscillations that will be catastrophic for the system. Additionally, the 256R network does not cause load variations on the reference voltage.

The bottom resistor and the top resistor of the ladder network in *Figure 1* are not the same value as the remainder of the network. The difference in these resistors causes the output characteristic to be symmetrical with the zero and full-scale points of the transfer curve. The first output transition occurs when the analog signal has reached $+\frac{1}{2}$ LSB and succeeding output transitions occur every 1 LSB later up to full-scale.

The successive approximation register (SAR) performs 8 iterations to approximate the input voltage. For any SAR type converter, n-iterations are required for an n-bit converter. *Figure 2* shows a typical example of a 3-bit converter. In the ADC0808, ADC0809, the approximation technique is extended to 8 bits using the 256R network.



TL/H/5672–2

FIGURE 1. Resistor Ladder and Switch Tree

Functional Description (Continued)

The A/D converter's successive approximation register (SAR) is reset on the positive edge of the start conversion (SC) pulse. The conversion is begun on the falling edge of the start conversion pulse. A conversion in process will be interrupted by receipt of a new start conversion pulse. Continuous conversion may be accomplished by tying the end-of-conversion (EOC) output to the SC input. If used in this mode, an external start conversion pulse should be applied after power up. End-of-conversion will go low between 0 and 8 clock pulses after the rising edge of start conversion.

The most important section of the A/D converter is the comparator. It is this section which is responsible for the ultimate accuracy of the entire converter. It is also the comparator drift which has the greatest influence on the repeatability of the device. A chopper-stabilized comparator provides the most effective method of satisfying all the converter requirements.

The chopper-stabilized comparator converts the DC input signal into an AC signal. This signal is then fed throught a high gain AC amplifier and has the DC level restored. This technique limits the drift component of the amplifier since the drift is a DC component which is not passed by the AC amplifier. This makes the entire A/D converter extremely insensitive to temperature, long term drift and input offset errors.

Figure 4 shows a typical error curve for the ADC0808 as measured using the procedures outlined in AN-179.

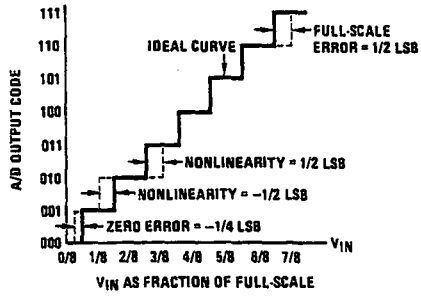


FIGURE 2. 3-Bit A/D Transfer Curve

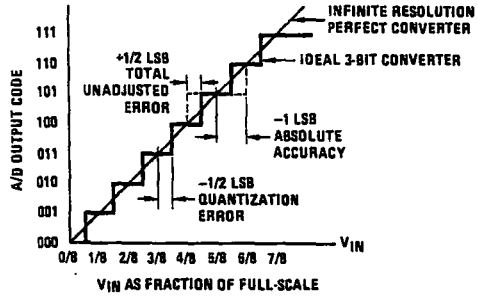
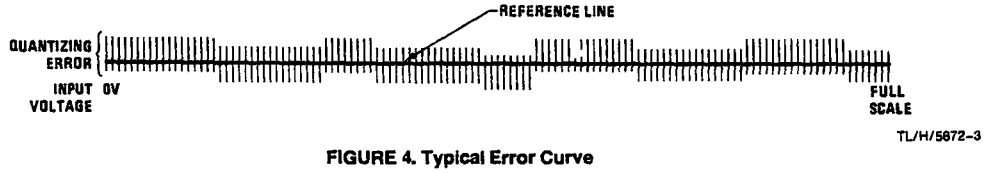


FIGURE 3. 3-Bit A/D Absolute Accuracy Curve

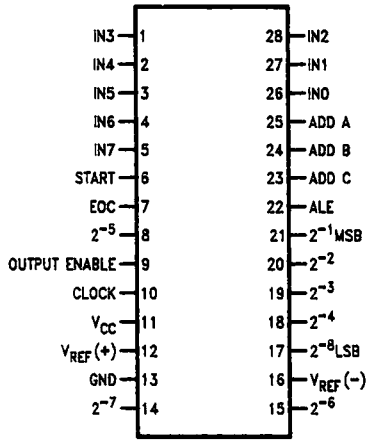


TL/H/5672-3

FIGURE 4. Typical Error Curve

Connection Diagrams

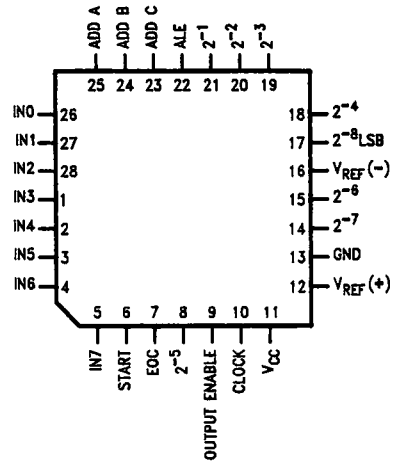
Dual-In-Line Package



TL/H/5672–11

Order Number ADC0808CCN, ADC0809CCN, ADC0808CCJ or ADC0808CJ
See NS Package J28A or N28A

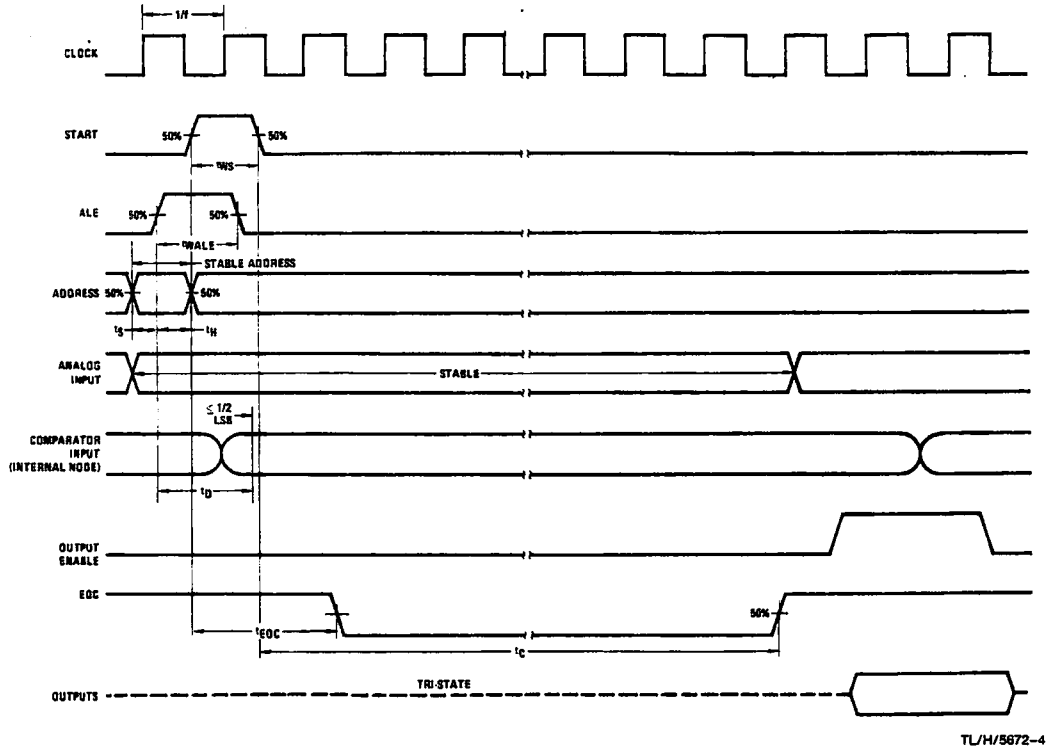
Molded Chip Carrier Package



TL/H/5672–12

Order Number ADC0808CCV or ADC0809CCV
See NS Package V28A

Timing Diagram



TL/H/5672–4

FIGURE 5

Typical Performance Characteristics

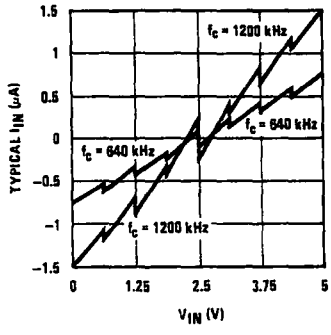
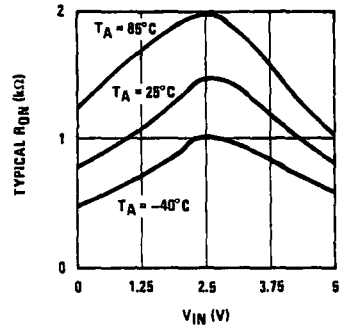


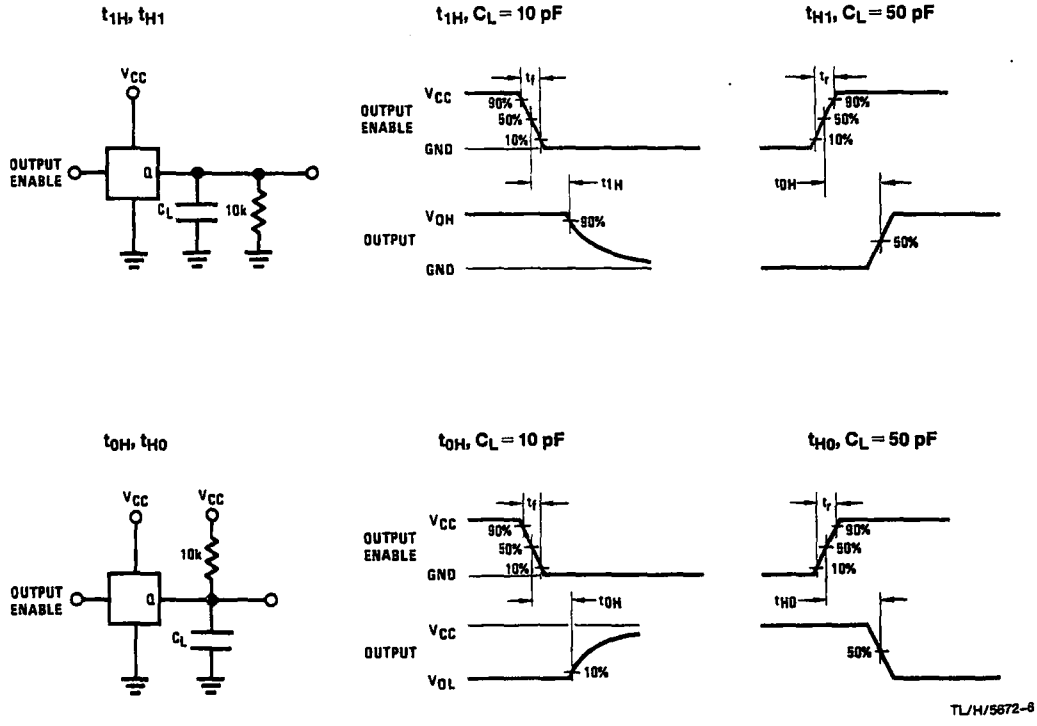
FIGURE 6. Comparator I_{IN} vs V_{IN} ($V_{CC} = V_{REF} = 5V$)



TL/H/5672–5

FIGURE 7. Multiplexer R_{ON} vs V_{IN} ($V_{CC} = V_{REF} = 5V$)

TRI-STATE Test Circuits and Timing Diagrams



TL/H/5672–6

FIGURE 8

Applications Information

OPERATION

1.0 RATIOMETRIC CONVERSION

The ADC0808, ADC0809 is designed as a complete Data Acquisition System (DAS) for ratiometric conversion systems. In ratiometric systems, the physical variable being measured is expressed as a percentage of full-scale which is not necessarily related to an absolute standard. The voltage input to the ADC0808 is expressed by the equation

$$\frac{V_{IN}}{V_{fs}-V_Z}=\frac{D_X}{D_{MAX}-D_{MIN}} \quad (1)$$

V_{IN} = Input voltage into the ADC0808

V_{fs} = Full-scale voltage

V_Z = Zero voltage

D_X = Data point being measured

D_{MAX} = Maximum data limit

D_{MIN} = Minimum data limit

A good example of a ratiometric transducer is a potentiometer used as a position sensor. The position of the wiper is directly proportional to the output voltage which is a ratio of the full-scale voltage across it. Since the data is represented as a proportion of full-scale, reference requirements are greatly reduced, eliminating a large source of error and cost for many applications. A major advantage of the ADC0808, ADC0809 is that the input voltage range is equal to the supply range so the transducers can be connected directly across the supply and their outputs connected directly into the multiplexer inputs, *(Figure 9)*.

Ratiometric transducers such as potentiometers, strain gauges, thermistor bridges, pressure transducers, etc., are suitable for measuring proportional relationships; however, many types of measurements must be referred to an absolute standard such as voltage or current. This means a system reference must be used which relates the full-scale voltage to the standard volt. For example, if $V_{CC}=V_{REF}=5.12V$, then the full-scale range is divided into 256 standard steps. The smallest standard step is 1 LSB which is then 20 mV.

2.0 RESISTOR LADDER LIMITATIONS

The voltages from the resistor ladder are compared to the selected into 8 times in a conversion. These voltages are coupled to the comparator via an analog switch tree which is referenced to the supply. The voltages at the top, center and bottom of the ladder must be controlled to maintain proper operation.

The top of the ladder, Ref(+), should not be more positive than the supply, and the bottom of the ladder, Ref(−), should not be more negative than ground. The center of the ladder voltage must also be near the center of the supply because the analog switch tree changes from N-channel switches to P-channel switches. These limitations are automatically satisfied in ratiometric systems and can be easily met in ground referenced systems.

Figure 10 shows a ground referenced system with a separate supply and reference. In this system, the supply must be trimmed to match the reference voltage. For instance, if a 5.12V is used, the supply should be adjusted to the same voltage within 0.1V.

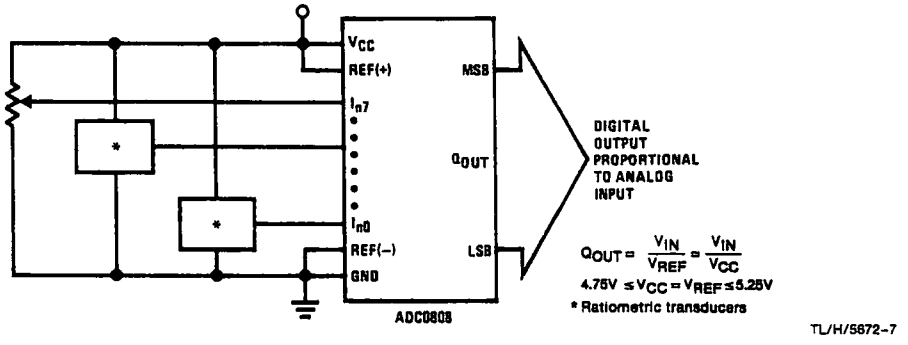


FIGURE 9. Ratiometric Conversion System

Applications Information (Continued)

The ADC0808 needs less than a milliamp of supply current so developing the supply from the reference is readily accomplished. In *Figure 11* a ground referenced system is shown which generates the supply from the reference. The buffer shown can be an op amp of sufficient drive to supply the milliamp of supply current and the desired bus drive, or if a capacitive bus is driven by the outputs a large capacitor will supply the transient supply current as seen in *Figure 12*. The LM301 is overcompensated to insure stability when loaded by the 10 μF output capacitor.

The top and bottom ladder voltages cannot exceed V_{CC} and ground, respectively, but they can be symmetrically less than V_{CC} and greater than ground. The center of the ladder voltage should always be near the center of the supply. The sensitivity of the converter can be increased, (i.e., size of the LSB steps decreased) by using a symmetrical reference system. In *Figure 13*, a 2.5V reference is symmetrically centered about $V_{CC}/2$ since the same current flows in identical resistors. This system with a 2.5V reference allows the LSB bit to be half the size of a 5V reference system.

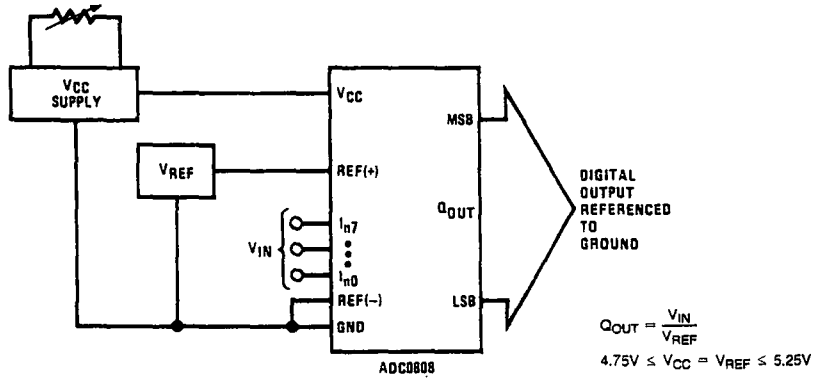


FIGURE 10. Ground Referenced Conversion System Using Trimmed Supply

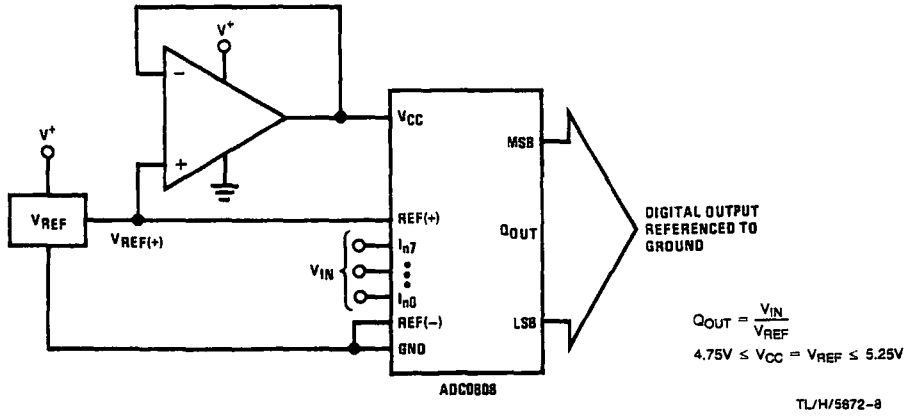


FIGURE 11: Ground Referenced Conversion System with Reference Generating V_{CC} Supply

Applications Information (Continued)

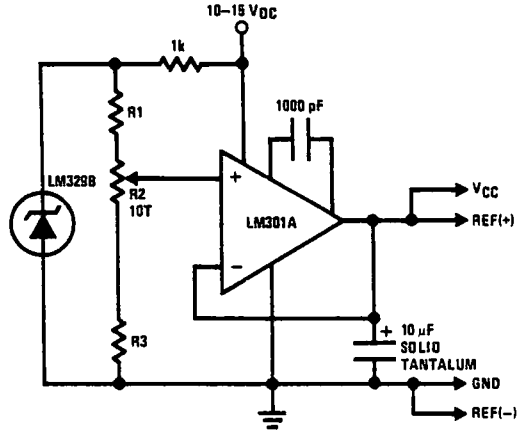
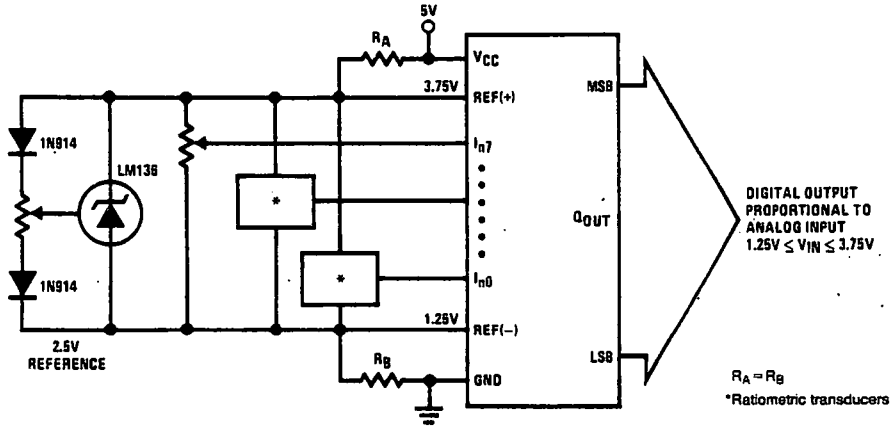


FIGURE 12. Typical Reference and Supply Circuit



TL/H/5872–9

FIGURE 13. Symmetrically Centered Reference

3.0 CONVERTER EQUATIONS

The transition between adjacent codes N and N + 1 is given by:

$$V_{IN} = \left\{ (V_{REF(+)} - V_{REF(-)}) \left[\frac{N}{256} + \frac{1}{512} \right] \pm V_{TUE} \right\} + V_{REF(-)} \quad (2)$$

The center of an output code N is given by:

$$V_{IN} \left\{ (V_{REF(+)} - V_{REF(-)}) \left[\frac{N}{256} \right] \pm V_{TUE} \right\} + V_{REF(-)} \quad (3)$$

The output code N for an arbitrary input are the integers within the range:

$$N = \frac{V_{IN} - V_{REF(-)}}{V_{REF(+)} - V_{REF(-)}} \times 256 \pm \text{Absolute Accuracy} \quad (4)$$

where: V_{IN} = Voltage at comparator input

$V_{REF(+)}$ = Voltage at Ref(+)

$V_{REF(-)}$ = Voltage at Ref(−)

V_{TUE} = Total unadjusted error voltage (typically $V_{REF(+)} \div 512$)

4.0 ANALOG COMPARATOR INPUTS

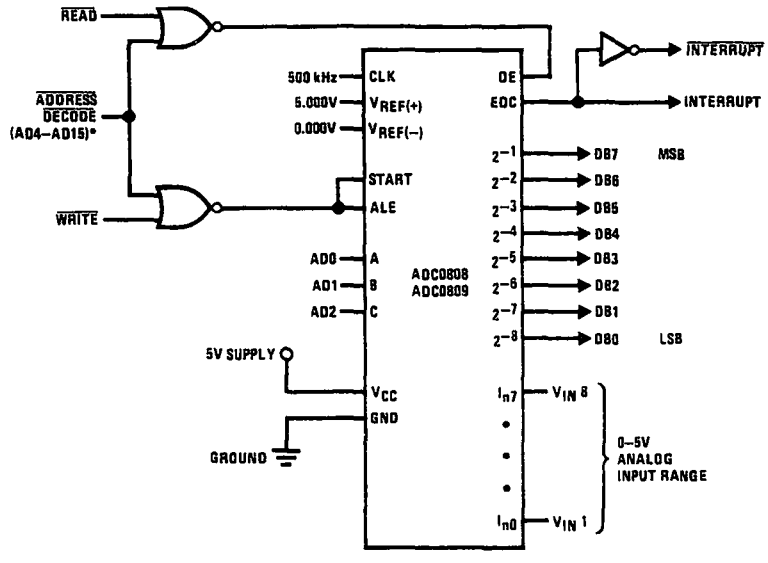
The dynamic comparator input current is caused by the periodic switching of on-chip stray capacitances. These are connected alternately to the output of the resistor ladder/switch tree network and to the comparator input as part of the operation of the chopper stabilized comparator.

The average value of the comparator input current varies directly with clock frequency and with V_{IN} as shown in *Figure 6*.

If no filter capacitors are used at the analog inputs and the signal source impedances are low, the comparator input current should not introduce converter errors, as the transient created by the capacitance discharge will die out before the comparator output is strobed.

If input filter capacitors are desired for noise reduction and signal conditioning they will tend to average out the dynamic comparator input current. It will then take on the characteristics of a DC bias current whose effect can be predicted conventionally.

Typical Application



TL/H/5672–10

*Address latches needed for 8085 and SC/MP interfacing the ADC0808 to a microprocessor

MICROPROCESSOR INTERFACE TABLE

PROCESSOR	READ	WRITE	INTERRUPT (COMMENT)
8080	$\overline{\text{MEMR}}$	$\overline{\text{MEMW}}$	INTR (Thru RST Circuit)
8085	$\overline{\text{RD}}$	$\overline{\text{WR}}$	INTR (Thru RST Circuit)
Z-80	$\overline{\text{RD}}$	$\overline{\text{WR}}$	$\overline{\text{INT}}$ (Thru RST Circuit, Mode 0)
SC/MP	NRDS	NWDS	SA (Thru Sense A)
6800	VMA•ϕ2•R/W	VMA•ϕ•$\overline{\text{R/W}}$	$\overline{\text{IRQA}}$ or $\overline{\text{IRQB}}$ (Thru PIA)

Ordering Information

TEMPERATURE RANGE		−40°C to +85°C			−55°C to +125°C
Error	±½ LSB Unadjusted	ADC0808CCN	ADC0808CCV	ADC0808CCJ	ADC0808CJ
	±1 LSB Unadjusted	ADC0809CCN	ADC0809CCV		
Package Outline		N28A Molded DIP	V28A Molded Chip Carrier	J28A Ceramic DIP	J28A Ceramic DIP

เอกสารอ้างอิง

ไพศาล ธรรมโพธิ์ทอง, "ระบบการรู้จำเสียงพูดต่างบุคคล," คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย, 2533.

อาทร นันทิยกุล, "การสังเคราะห์เสียงพูดจากข้อความภาษาไทย," คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย, 2533.

Teja, Edward R., "Teaching your computer to talk : a manual of command and respond," TAB BOOKS Inc.; 1981.

Witten, I.H., "Principle of Computer Speech," Academic Press Inc. (London) Ltd. 1982.

Sakoe, H. and Chiba, S., "Dynamic Programming Algorithm Optimization for Spoken Word Recognition," IEEE Trans. ASSP, Vol. ASSP-26, No. 1, Feb. 1978.

กลุ่ม CNS, "ออปแอมป์," หจก.สำนักพิมพ์ฟิสิกส์เซ็นเตอร์, กรุงเทพมหานคร.

วันชัย คุณากรวงศ์, บัณฑิต บัวบูชา, รัฐวุฒิ ประทุมราช และรุ่งแสง เครือไวศวรรณ, "การคำนวณวงจรทรานซิสเตอร์," หจก.สำนักพิมพ์ฟิสิกส์เซ็นเตอร์, กรุงเทพมหานคร.

วิสุทธิ์ อัศวนนทวงศ์, "พื้นฐาน A/D และ D/A คอนเวทเตอร์," วารสารคอมพิวเตอร์อิเลคทรอนิคส์เวิลด์ เล่มที่ 135 หน้า 102-115, ศูนย์การพิมพ์ดวงกมล, กรุงเทพมหานคร.

พฤติ สุนากร, "ดิจิตอลออดิโอเทคนิค," วารสารเซมิคอนดักเตอร์อิเลคทรอนิคส์, เล่มที่ 99 หน้า 241-246, ส.เอเชียเพรส (1989), กรุงเทพมหานคร.

น.ต.ดร. ไพศาล สงวนหมู่, รศ. ยืน ภู่วรวรรณ, "การสื่อสารข้อมูล และไมโครคอมพิวเตอร์เนตเวอร์ค," บริษัทซีเอ็ดยูเคชัน จำกัด, กรุงเทพมหานคร.

ประวัติผู้จัดทำโครงงานพิเศษ

นายนุวัศ เจนเกียรติพู เกิดวันที่ 10 มกราคม 2516 สำเร็จการศึกษาระดับมัธยมศึกษาตอนต้นและตอนปลาย จากโรงเรียนอัสสัมชัญธนบุรี จากนั้นเข้าศึกษาต่อในระดับปริญญาตรี สาขาฟิสิกส์ประยุกต์ คณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง และสำเร็จการศึกษาในปีการศึกษา 2537

นายสงกรานต์ คุ้มพูล เกิดวันที่ 13 เมษายน 2516 สำเร็จการศึกษาระดับมัธยมศึกษาตอนต้นและตอนปลาย จากโรงเรียนพระโขนงพิทยาลัย จากนั้นเข้าศึกษาต่อในระดับปริญญาตรี สาขาฟิสิกส์ประยุกต์ คณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง และสำเร็จการศึกษาในปีการศึกษา 2537