ระบบชุมสายโทรศัพท์อัตโนมัติแบบให้บริการหลายหน้าที่
Multi-function Private Automatic Branch Exchange

จรัสพรรณ พันธะ
JARUSPAN PANTA

อาจารย์ที่ปรึกษา
ผู้ช่วยศาสตราจารย์ ดร. บุญวัฒน์ อัตชู
ADVISOR
Asst. Prof. Dr. Boonwat Attachoo

วิทยานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิศวกรรมไฟฟ้า
บัณฑิตวิทยาลัย
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2536
ISBN 974-621-126-9

MULTI-FUNCTION PRIVATE AUTOMATIC BRANCH EXCHANGE

MR. JARUSPAN PANTA

ADVISOR

ASST. PROF. DR. BOONWAT ATTACHOO

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE
MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING
GRADUATE SCHOOL
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

ISBN 974-621-126-9

หัวข้อวิทยานิพนธ์ ระบบชุมสายโทรศัพท์อัตโนมัติแบบให้บริการหลายหน้าที่

นักศึกษา นายจรัสพรรณ พันธะ

อาจารย์ผู้ควบคุมวิทยานิพนธ์ ผศ.ดร. บุญวัฒน์ อัตชู

ระดับการศึกษา วิศวกรรมศาสตรมหาบัณฑิต สาขาวิศกรรมไฟฟ้า

ปีการศึกษา 2536

บทคัดย่อ

วิทยานิพนธ์ฉบับนี้นำเสนอการพัฒนาชุมสายโทรศัพท์ภายในขนาด 3 สายนอก 8 สายในซึ่งมีระบบตอบรับโทรศัพท์เป็นฟังก์ชันการทำงานหลักของระบบ โดยในการบันทึกและเล่นกลับเสียงจะกระทำผ่านฮาร์ดดิสต์ของไมโครคอมพิวเตอร์ ซึ่งมีแชนแนลของการตอบรับสายนอกโดยอัตโนมัติจำนวน 2 แชนแนล และแชนแนลสำหรับสายภายในจำนวน 1 แชนแนล และมีระบบการแจ้งข่าวสารจากส่วนกลางไปยังเครื่องไมโครคอมพิวเตอร์ผ่าน RS-232 เป็นฟังก์ชันเสริม ในส่วนของระบบตอบรับ แชนแนลทั้งสามสามารถทำงานอย่างเป็นอิสระต่อกันโดยการควบคุมของไมโครคอมพิวเตอร์ซึ่งทำงานในลักษณะแบ่งปันเวลาให้กับการควบคุมส่วนต่างๆของระบบตอบรับและการติดต่อกับชุมสายโทรศัพท์ โดยในวิทยานิพนธ์จะนำเสนอในรูปโครงสร้างข้อมูลและอัลกอริธึมธ์ที่ใช้ในกระบวนการดังกล่าวซึ่งระบบโดยรวมจะเน้นที่การใช้ฮาร์ดแวร์เท่าที่จำเป็นและให้งานส่วนใหญ่กระทำโดยซอฟต์แวร์

Title	Multi-function Private Automatic Branch Exchange
Student	MR. JARUSPAN PANTA
Thesis Advisor	Asst. Prof. DR. BOONWAT ATTACHOO
Level of study	MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING
Academic Year	1993

ABSTRACT

This thesis proposes a development of a PABX of 3 inlets and 8 outlets with automatic voice mail box as a main function of the system. Recording and playing use hardisk of microcomputer. Two channels are available for automatic answer of external lines and one for internal uses. Also, distribution of information from centre system by RS-232 is introduced as a supplement function. All three channels of automatic voice mail system can work independently under controling of a microcomputer working as time sharing to control all parts of voice mail system and PABX interfacing. Such a system is proposes as data structure and algorithm for controling and overall system is stressed on minimizing hardware and uses software as much as possible.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลงได้ด้วยความสนับสนุนในทุกๆด้านจากอาจารย์บุญวัฒน์ อัตชู อาจารย์ที่ปรึกษา และขอขอบคุณอาจารย์วัชระ ฉัตรวิริยะ และภาควิชาวิศวกรรมคอมพิวเตอร์ที่ได้กรุณาให้ยืมเครื่องที่จำเป็น และผู้อื่นที่ได้มีส่วนในการช่วยเหลือทุกท่าน ขอขอบพระคุณ

สารบัญ

สารบัญรูป

บทที่ 1
บทนำ

1.1 หลักการและเหตุผล

วิทยานิพนธ์ฉบับนี้เสนอการออกแบบและพัฒนาชุมสายโทรศัพท์ภายในขนาด 3 สายนอก 8 สายใน ซึ่งมีระบบตอบรับโทรศัพท์เป็นฟังก์ชั่นการทำงานหลักของระบบ โดยมีแชนแนลของการตอบรับสายนอกโดยอัตโนมัติจำนวน 2 แชนแนล และ แชนแนลสำหรับสายภายในจำนวน 1 แชนแนล และมีระบบการแจ้งข่าวสารจากส่วนกลางไปยังเครื่องคอมพิวเตอร์จำนวน 4 เครื่อง โดยผ่าน RS-232C เป็นฟังก์ชันเสริม

อุปกรณ์ประเภทระบบตอบรับโทรศัพท์โดยอัตโนมัติมีบทบาทอย่างมากในการติดต่อทางโทรศัพท์ เพราะช่วยลดปัญหาในการโทรแล้วไม่พบผู้รับซึ่งทำให้เสียเงินและเวลา โดยระบบตอบรับโทรศัพท์โดยอัตโนมัติจะรับฝากข้อความเสียงไว้ซึ่งผู้ถูกเรียกสามารถสอบถามข้อความนี้จากระบบได้ หรือผู้ใช้อาจจะฝากข้อความไว้เมื่อมีผู้โทรเข้ามาก็จะแจ้งข้อความดังกล่าวให้ คุณสมบัติดังกล่าวเป็นคุณสมบัติพื้นฐานที่ระบบระบบตอบรับโทรศัพท์โดยอัตโนมัติทั่วไปควรจะมี

ปัจจุบันระบบคอมพิวเตอร์ได้ถูกพัฒนาขึ้นมาก จึงถูกนำมาประยุกต์ใช้ในระบบตอบรับโทรศัพท์ทำให้มีขีดความสามารถเพิ่มมากขึ้น ซึ่งหากพิจารณาจากสินค้าประเภทนี้ที่มีขายอยู่ในตลาดอาจแบ่งได้เป็นสองกลุ่มใหญ่ๆ กลุ่มหนึ่งจะใช้เทคโนโลยีที่ไม่สูงนักซึ่งมักจะเป็นระบบเทป ที่ต้องต่อกับเทปบันทึกเสียง ซึ่งใช้งานไม่สะดวกและไม่เหมาะกับงานสำนักงานเพราะต้องเสียเวลาในการกลับเทปไปมา กลุ่มที่สองจะใช้เทคโนโลยีที่สูงขึ้นมาคือใช้ระบบคอมพิวเตอร์เข้ามาช่วยและมีระบบซอฟต์แวร์ที่มีประสิทธิภาพ

ถ้าหากพิจารณาระบบตอบรับโทรศัพท์อัตโนมัติที่ใช้ระบบคอมพิวเตอร์แล้วจะเห็นว่าสามารถพัฒนาประสิทธิภาพได้ง่าย โดยใช้เทคนิคทางซอฟต์แวร์ ซึ่งทำให้ระบบตอบรับโทรศัพท์อัตโนมัติไม่เป็นแต่อุปกรณ์ที่รับโทรศัพท์หรือฝากข้อความเท่านั้นแต่ยังสามารถทำให้เป็นอุปกรณ์ที่ให้บริการข่าวสารทางโทรศัพท์ได้อย่างมีประสิทธิภาพ

จุดมุ่งหมายของงานวิจัยชิ้นนี้เดิมทีเน้นที่การพัฒนาระบบตอบรับโทรศัพท์อัตโนมัติที่สามารถใช้กับชุมสายทั่วไปที่มีอยู่ในท้องตลาดได้ แต่ผลจากการพัฒนาพบว่า การที่จะได้ระบบตอบรับโทรศัพท์ที่สมบูรณ์กล่าวคือใช้งานได้กว้างขวางและหลากหลายนั้นควรจะต้องมีชุมสายโทรศัพท์ซึ่งสามารถทำงานเป็นระบบเดียวกันระบบตอบรับโทรศัพท์นั้นได้ ดังนั้นงานวิจัยชิ้นนี้จึงได้มุ่งพัฒนาทั้งระบบคือเป็นระบบชุมสายโทรศัพท์ภายในที่มีระบบตอบรับโทรศัพท์อัตโนมัติเป็นฟังก์ชั่นหลัก นอกจากนี้ผู้วิจัยได้เสริมระบบการแจ้งข่าวสารจากส่วนกลางไปยังคอมพิวเตอร์เครื่องลูกจำนวน 4 เครื่องผ่านทาง RS-232C เป็นฟังก์ชันย่อย เพื่อใช้ในการแจ้งข่าวสารประจำวันภายในสำนักงาน อย่างไรก็ตามระบบดังกล่าวเป็นระบบอิสระโดยไม่ใช้สายร่วมกับชุมสายโทรศัพท์ เพียงแต่ใช้ส่วนอินเทอร์เฟสกับไมโครคอมพิวเตอร์ร่วมกับระบบตอบรับโทรศัพท์

1 2 **<u>เนื้อหาโดยสังเขป</u>**

ดังได้กล่าวไว้ตอนต้นแล้วว่างานวิจัยนี้มุ่งที่การพัฒนาระบบตอบรับโทรศัพท์ แนวความคิดหลักในการออกแบบคือให้ระบบตอบรับโทรศัพท์เป็นส่วนเพิ่มขยาย (extension) จากชุมสายปกติ โดยจะนำเสนอการออกแบบให้ส่วนเชื่อมต่อระหว่างชุมสายโทรศัพท์กับระบบตอบรับเพื่อทำการรับส่งคำสั่งที่จำเป็น โดยนำเสนอในหลักการเพื่อให้ผู้สนใจที่พัฒนาเกี่ยวกับชุมสายโทรศัพท์อยู่แล้วสามารถนำระบบตอบรับโทรศัพท์นี้ไปเป็นส่วนเพิ่มขยายให้กับระบบชุมสายที่มีอยู่ได้โดยการเพิ่มเติมฮาร์ดแวร์และซอฟต์แวร์บางส่วนให้กับชุมสายโทรศัพท์ ถึงอย่างไรก็ตามในงานวิจัยนี้ก็ได้พัฒนาทั้งระบบไว้แล้วคือทั้งชุมสายโทรศัพท์และระบบตอบตอบรับโทรศัพท์ เพื่อเป็นต้นแบบสำหรับอ้างอิงแก่ผู้สนใจทั่วๆไป

ลักษณะของระบบชุมสายที่พัฒนาขึ้นมานี้มีการทำงานแบบแยกส่วนได้ กล่าวคือระบบทั้งสามส่วนอันได้แก่ชุมสายโทรศัพท์ ระบบตอบรับโทรศัพท์ และระบบแจ้งข่าวสารจากส่วนกลางสามารถแยกกันทำงานได้หรือมารวมกันทำงานก็ได้ เมื่อใช้ระบบตอบรับโทรศัพท์ร่วมกับชุมสายโทรศัพท์ที่พัฒนาขึ้น ผู้ใช้สามารถขอใช้บริการต่อไปนี้จากระบบตอบรับได้ คือฝากข้อความไว้ให้ผู้ที่โทรเข้ามาหาเมื่อจะไม่อยู่และโปรแกรมให้ระบบตอบรับรับฝากข้อความที่ผู้โทรเข้ามาฝากไว้ รับฟังข้อความที่มีผู้ฝากไว้ให้ และโปรแกรมให้ระบบโทรออกตามเวลาที่กำหนดโดยใช้ข้อความที่บันทึกไว้ก่อนแล้ว การจัดเก็บข้อมูลเสียงจะเก็บอยู่ในรูปไฟล์บนฮาร์ดดิสต์ของไมโครคอมพิวเตอร์ ในแง่ของผู้ใช้ไม่ได้ติดต่อกับไฟล์เหล่านี้โดยตรงเมื่อมีการบันทึกหรือเล่นกลับ แต่ซอฟต์แวร์ของระบบตอบรับซึ่งทำงานบนเครื่องไมโครคอมพิวเตอร์จะเป็นผู้จัดการเกี่ยวกับไฟล์ให้ การแจ้งข่าวสารจากส่วนกลางเป็นระบบอิสระโดยเป็นระบบที่ส่งข้อความสั้นๆขนาด 256 ไบท์จากเครื่องไมโครคอมพิวเตอร์เครื่องหนึ่งไปยังเครื่องคอมพิวเตอร์อีก 4 เครื่องผ่านพอร์ทอนุกรม โดยจะสลับกันส่งไปทีละเครื่องโดยเครื่องที่ทำหน้าที่รับข้อมูลจะต้องรันโปรแกรมสื่อสารข้อมูลเช่น Procom หรือ Telix ไว้ อย่างไรก็ตามระบบดังกล่าวเป็นส่วนเสริมไม่มีส่วนเกี่ยวข้องกับชุมสายโทรศัพท์หรือระบบตอบรับโทรศัพท์แต่อย่างใด

สำหรับผู้ใช้ที่โทรมาจากสายนอกจะมีแชนแนลสำหรับการตอบรับให้สองแชนแนล โดยแต่ละแชนแนลเมื่อตอบรับจะมีข้อความแจ้งให้ผู้ใช้ทราบ หากผู้ใช้ทราบเบอร์ภายในก็จะสามารถกดหมายเลขให้ระบบโอนสายให้ได้เลย หรือให้โอเปอร์เรเตอร์เป็นผู้โอนให้

ผู้ใช้สายในจะมองเห็นระบบตอบรับโทรศัพท์เป็นกล่องเสียงอันหนึ่งซึ่งเข้าถึงได้โดยการกดเลขหมายบริการที่กล่องเสียงนี้มีให้ หมายเลขบริการมีอยู่ 5 หมายเลขซึ่งให้บริการ 5 แบบด้วยกันคือ

1 ขอบันทึกเสียง ซึ่งใช้เมื่อต้องการจะฝากข้อความไว้ให้ผู้ที่โทรมาจากสายนอกทราบ

2. ขอรับฟังข้อความที่มีผู้โทรเข้ามาฝากไว้ โดยข้อความที่ฝากครั้งหลังสุดจะได้รับฟังก่อน และหลังจากรับฟังจบแล้วไฟล์ซึ่งบรรจุข้อความนั้นจะถูกลบโดยอัตโนมัติเพื่อไม่ให้เปลืองเนื้อที่ฮาร์ดดิสต์ การรับฟังจะได้หนึ่งไฟล์หรือหนึ่งข้อความต่อการโทรเข้าหาระบบหนึ่งครั้ง การกำหนดเช่นนี้เนื่องจากว่าได้กำหนดให้มีแชนแนลของการบันทึกและเล่นกลับเพียงแชนแนลเดียวในระบบ จึงต้องให้มีการแบ่งกันใช้อย่างทั่วถึงโดยไม่ให้มีการจองการใช้แชนแนลจากสายใดสายหนึ่งนานเกินไป (อย่างไรก็ตามกฎเกณฑ์อันนี้สามารถเปลี่ยนแปลงได้โดยแก้ไขซอฟต์แวร์ รายละเอียดดูได้จากบทที่ 5)

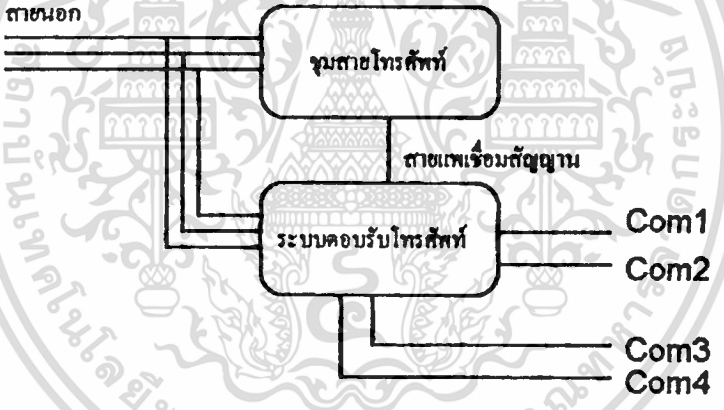
3. ตั้งโปรแกรมเพื่อการโทรออก บริการนี้จะให้ผู้ใช้บันทึกเสียงไว้และกำหนดเวลาและเลขหมายที่ต้องการให้ระบบโทรออกโดยแจ้งข้อความที่บันทึกไว้แจ้งให้ปลายทางทราบ

4. โปรแกรมให้ระบบโอนสายนอกเข้ามาได้เมื่อผู้โทรเข้าต้องการจะโอนสายมา

5. โปรแกรมให้ระบบแจ้งให้ผู้โทรเข้ามาฝากข้อความไว้โดยไม่ต้องโอนสายเข้ามา

แนวความคิดหลักในการพัฒนาระบบตอบรับโทรศัพท์คือให้ใช้ฮาร์ดแวร์เท่าที่จำเป็นแล้วใช้ซอฟต์แวร์ในควบคุมงานให้มากที่สุด ดังจะได้พบในเนื้อหาต่อไปว่าวงจรหลักๆของระบบตอบรับจะเป็นเพียงวงจรที่ใช้ในการแชมปลิ้งและวงจรที่ทำหน้าที่อินเทอร์เฟสกับสัญญานโทรศัพท์ โดยจุดที่นำเสนอจะอยู่ที่การจัดโครงสร้างข้อมูลและอัลกอริธึมย์ของซอฟต์แวร์บนเครื่องไมโครคอมพิวเตอร์เพื่อให้สามารถจัดการงานต่างๆได้พร้อมกันในมุมมองของผู้ใช้ ซึ่งก็คือการแบ่งปันเวลาของ CPU เพื่อทำงานต่างๆที่อาจเกิดขึ้นพร้อมกัน เช่นในขณะเล่นกลับในแชนแนลที่ 1 อาจมีการโปรแกรมจากผู้ใช้ในสายที่ 2 และในขณะเดียวกันอาจมีการบันทึกข้อความจากสายที่ 5 ซึ่งงานทั้งหมดนี้จะถูกจัดการโดย CPU บนเครื่องไมโครคอมพิวเตอร์ทั้งสิ้น

ลักษณะการต่อใช้งานเป็นดังรูป



แผนภาพการต่อใช้งานระบบ

จากรูป การต่อใช้ระบบตอบรับโทรศัพท์จะกระทำคร่อมสายนอกของสายองค์การโทรศัพท์ โดยระบบตอบรับโทรศัพท์ประกอบด้วยการ์ดอินเทอร์เฟสและเครื่องไมโครคอมพิวเตอร์พร้อมฮาร์ดดิสต์ ทำการรับส่งคำสั่งร่วมกับชุมสายโทรศัพท์ที่พัฒนาขึ้นผ่านทางสายแพ การส่งข้อความจากส่วนกลางทำโดยโหลดไฟล์ขนาด 256 ไบท์ในตอนเริ่มต้นทำงานแล้วข้อมูลจะถูกส่งไปยังไมโครคอมพิวเตอร์ 4 เครื่องที่ต่ออยู่เรียงกันไป โดยไมโครคอมพิวเตอร์ดังกล่าวจะต้องรันโปรแกรมสื่อสารข้อมูลไว้

1.3 **โครงสร้างของวิทยานิพนธ์**

วิทยานิพนธ์ฉบับนี้แบ่งเนื้อหาออกเป็น 6 บท บทที่ 1 ได้แก่บทนำ บทที่ 2 เป็นหลักการและความรู้พื้นที่ใช้ในวิทยานิพนธ์ บทที่ 3 เป็นฮาร์ดแวร์ของระบบ อันได้แก่ชุมสายศัพท์ ระบบตอบรับโทรศัพท์ ระบบแจ้งข่าวสารจากส่วนกลาง บทที่ 4 ได้แก่ซอฟต์แวร์ของระบบ อันได้แก่ซอฟต์แวร์ของส่วนทั้งสาม โดยจะเน้นที่เรื่องโครงส่ร้างข้อมูลและอัลกอริธึมธ์ ซึ่งช่วยให้เข้าใจได้ง่ายกว่าการยกโปรแกรมหรือส่วนของโปรแกรมมากล่าว บทที่ 5 ได้แก่ผลการสอบและแนวทางการพัฒนา บทที่ 6 เป็นบทสรุป สุดท้ายได้แก่ภาคผนวกซึ่งรวบรวมต้นฉบับโปรแกรม และรายละเอียดของไอซีที่เกี่ยวข้อง

บทที่ 2
หลักการที่เกี่ยวกับงานวิจัย

บทนี้จะกล่าวถึงหลักการและความรู้ขั้นพื้นฐานต่างๆ ที่ใช้ในการพัฒนางานวิจัยชิ้นนี้

2.1 พื้นฐานของระบบโทรศัพท์

ระบบในการติดต่อกับชุมสาย

ในปัจจุบันการติดต่อผ่านชุมสายโทรศัพท์ทำได้สองระบบ

1 ระบบหมุน (Rotary Dialer) ระบบนี้จะใช้การส่งสัญญาณพัลซ์ตั้งแต่ 1 ถึง 10 พัลซ์ เช่นถ้ามีการส่งพัลซ์ 1 พัลซ์ก็หมายถึงการหมุนหมายเลขศูนย์ ถ้าส่ง 2 พัลซ์ก็หมายถึงหมายเลขหนึ่ง ดังนั้นถ้าหมุนเลข 9 ก็จะมีการส่งพัลซ์จำนวน 10 พัลซ์ และความเร็วในการส่งก็คือ 10 พัลซ์ต่อวินาที

2. ระบบกดปุ่ม (Touchtone) ระบบนี้จะใช้สัญญาณความถี่คู่ในการติดต่อกับชุมสาย โดยทั่วไปจะมีปุ่มสำหรับการติดต่ออยู่ 12 ปุ่ม คือ 0-9 และ * และ # บางเครื่องอาจมีปุ่มเพิ่มเติมขึ้นมา 4 ปุ่มได้แก่ A, B, C , D

โทรศัพท์แบบนี้จะกำเนิดสัญญาณความถี่คู่ไปยังชุมสาย ซึ่งแต่ละปุ่มจะมีคู่ความถี่ของตัวเองอยู่ จึงเรียกระบบเช่นนี้ว่า Dual-Tone Multiple-Frequency (DTMF) คู่ความถี่มาตรฐานเป็นดังตารางที่ 2.1

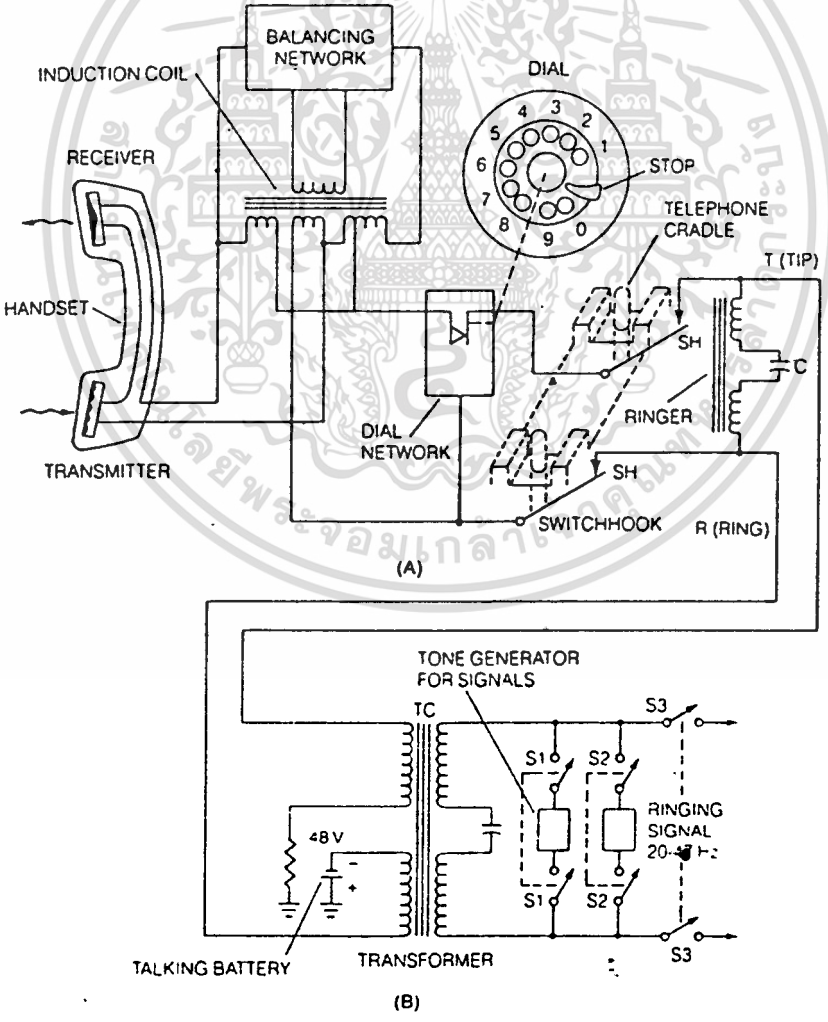
ตารางที่ 2.1 แสดงคู่ความถี่มาตรฐานของปุ่มต่างๆ

ปุ่ม	ความถี่ สูง (Hz)	ความถี่ ต่ำ (Hz)
1	1209	697
2	1336	697
3	1477	697
4	1209	770
5	1336	770
6	1477	770
7	1209	852
8	1336	852
9	1477	852
*	1209	941
0	1336	941
#	1477	941
A	1633	697
B	1633	770

C	1633	852
D	1633	941

2.1.1 **ชุมสายโทรศัพท์**

คู่สายของผู้ใช้บริการจะติดต่ออยู่กับชุมสายหลักซึ่งประกอบไปด้วยอุปกรณ์สวิทช์ซึ่ง อุปกรณ์สัญญาน และแหล่งกำเนิดสัญญานไปตรงที่จ่ายให้กับคู่สายที่ใช้บริการ รูปที่ 2.1 แสดงวงจรการเชื่อมต่อระหว่างชุมสายหลักกับคู่สายของผู้ใช้บริการในโทรศัพท์แบบหมุน แต่ละคู่สายจะต่ออยู่กับชุมสายหลักผ่านทางลูปภายใน (local loop) ซึ่งประกอบด้วยสัญญานสองเส้นคือ สาย T (trip) และ สาย R (ring) อุปกรณ์สวิทช์ซึ่งในชุมสายหลักจะทำหน้าที่เชื่อมต่อระหว่างผู้เรียกและผู้ถูกเรียก เมื่อติดต่อถึงกันได้ คู่สายทั้งสองจะสนทนากันผ่านลูปที่คัปปลิ้งโดยใช้หม้อแปลง (transformer-coupled loops) โดยใช้กระแสที่จ่ายให้โดยชุมสายหลัก



รูปที่ 2.1 แสดงวงจรระหว่างชุมสายหลักและคู่สายโทรศัพท์

การเริ่มการเรียก

เมื่อหูโทรศัพท์วางอยู่ เรียกว่าอยู่ในภาวะ on-hook วงจรระหว่างชุมสายหลักและโทรศัพท์จะเป็นวงจรเปิด ยกเว้นวงจรสัญญาณเรียก (ring circuit) ซึ่งจะต่ออยู่กับชุมสายหลักอยู่ตลอดเวลา โดยมีตัวเก็บประจุ C ทำหน้าที่กันไฟตรงไม่ให้ผ่านไปได้ ให้ผ่านได้เฉพาะสัญญาณเรียกซึ่งเป็นไฟสลับ (ดูรูปที่ 2.1)

เมื่อเรายกหูโทรศัพท์ขึ้น เรียกว่าอยู่ในภาวะ off-hook ซึ่งจะทำให้เกิดวงจรปิดระหว่างชุมสายหลักและคู่สายโทรศัพท์ ซึ่งชุมสายจะจ่ายการแสออกมาและรู้ว่าจะมีการติดต่อจากคู่สายที่ทำการยกหู

การส่งเลขหมาย เลขหมายที่ส่งมาจากคู่สายมีสองระบบคือระบบหมุนและระบบกดปุ่ม ซึ่งชุมสายในปัจจุบันสามารถรับได้ทั้งสองแบบ

การเชื่อมต่อ เมื่อเลขหมายที่คู่สายส่งมาเป็นเลขหมายที่มีอยู่จริง ชุมสายก็จะทำการเชื่อมคู่สายทั้งสองให้โดยผ่านอุปกรณ์สวิทช์ซึ่งซึ่งมีอยู่เป็นจำนวนหลายชุด แต่ถ้าหากไม่มีเลขหมายนั้นหรือเลขหมายปลายทางไม่อยู่ในภาวะ on-hook ก็จะส่งสัญญาณสายไม่ว่างไปให้กับคู่สายที่ทำการเรียกมา

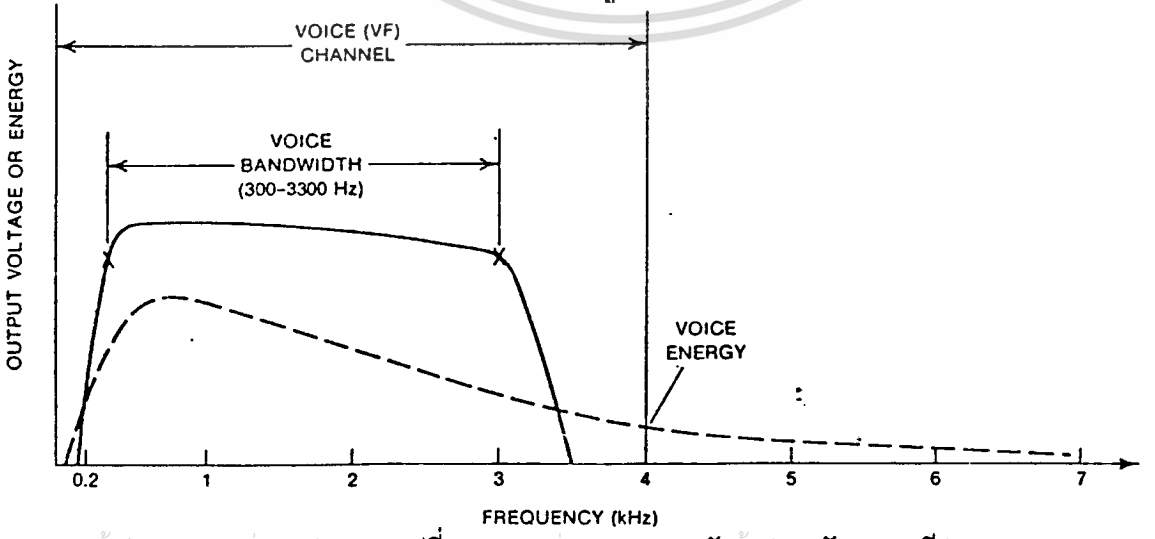
เมื่อจะทำการเชื่อมต่อคู่สายโดยผ่านอุปกรณ์สวิทช์ซึ่ง ชุมสายจะทำการส่งสัญญาณเรียกไปยังคู่สายปลายทาง และเมื่อคู่สายปลายทางทำการยกหูก็จะหยุดส่งสัญญาณเรียกนั้นแล้วจึงเชื่อมต่อคู่สายทั้งสองเข้าด้วยกัน

2.1 .2 **สัญญาณที่ส่งในคู่สายโทรศัพท์**

สัญญาณเสียงไม่ใช่สัญญาณชนิดเดียวที่ส่งในคู่สาย เพราะยังมีสัญญาณอื่นๆที่ใช้ในการติดต่ออันได้แก่ .สัญญาณเรียก สัญญาณให้หมุน สัญญาณ DTMF สัญญาณจากการหมุน สัญญาณเรียกกลับและสัญญาณสายไม่ว่าง สัญญาณต่างๆเหล่านี้เรียกว่าสัญญาณควบคุม

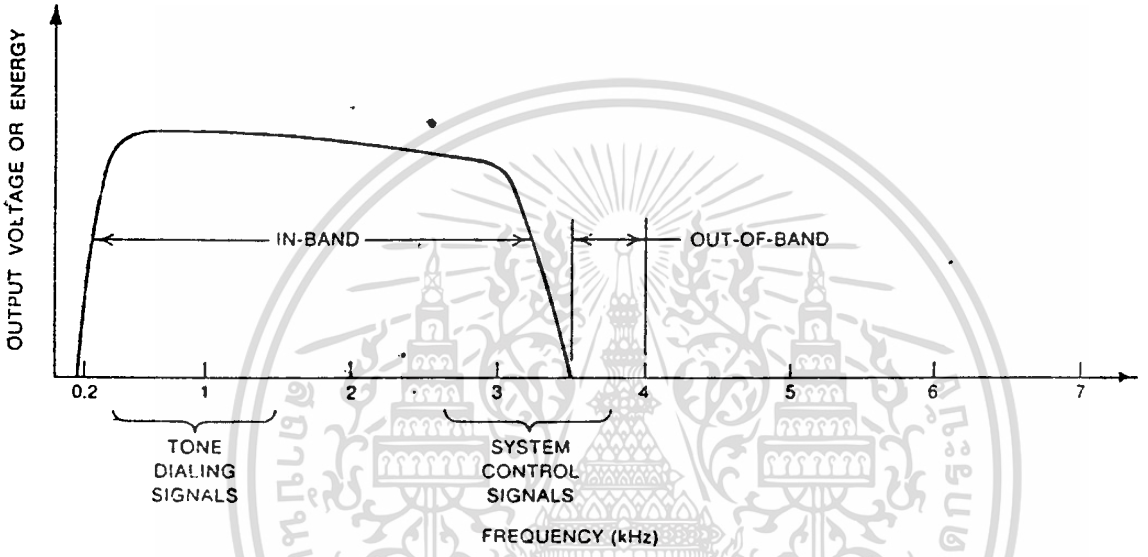
1 <u>สัญญาณเสียง</u>

สัญญาณเสียงมีความถี่ในช่วง 20 Hz ถึง 20,000 Hz และการกระจายของพลังงานเสียงแสดงในรูปที่ 2.2 จากรูปจะเห็นว่า พลังงานส่วนใหญ่จะอยู่ในช่วงที่ต่ำกว่า 100 Hz จนถึงเกินกว่า 6 kHz ถึงอย่างไรก็ตาม พบว่าพลังงานที่จำเป็นในการสนทนาโดยทั่วไปจะอยู่ในช่วง 200 Hz ถึง 4kHz



รูปที่ 2.2 แสดงการกระจายพลังงานของสัญญาณเสียง

เพื่อจำกัดสัญญาณที่ไม่ต้องการในขณะสนทนาและเพื่อไม่ให้เกิดความผิดพลาดขึ้นกับสัญญาณควบคุม วงจรที่ผ่านสัญญาณโทรศัพท์จะต้องให้สัญญาณในช่วงหนึ่งเท่านั้นที่ผ่านได้ ซึ่งเรียกว่า pass band ความถี่จาก ศูนย์จนถึง 4 kHz ถูกกำหนดให้เป็น pass band ของสัญญาณโทรศัพท์ ซึ่งเรียกช่วงความถี่นี้ว่า voice frequency (VF) ถึงอย่างไรก็ตามเราก็ไม่ใช้ช่วง VF นี้สำหรับสัญญาณเสียงทั้งหมด แต่จะเลือกช่วง 300-3300 Hz ดังแสดงในรูปที่ 2 3 สัญญาณในวงจรโทรศัพท์ที่อยู่ในช่วง 300-3300 Hz จะเรียกว่า สัญญาณ in-band และสัญญาณอื่นนอกจากนี้แต่ยังคงอยู่ใน VF จะเรียกว่าสัญญาณ out-of-band สัญญาณเสียงจะอยู่ใน in-band สัญญาณควบคุมจะมีอยู่ในทั้งสองแบนด์



รูปที่ 2 3 แสดงช่วงของสัญญาณ in-band และ out-of-band

ระดับความดังของสัญญาณเสียง

ระดับของสัญญาณเสียงแสดงอยู่ในรูปพลังงานของสัญญาณที่จ่ายให้กับโหลด ตัวอย่างเช่นในสายโทรศัพท์ซึ่งมีอิมพีแดนซ์ 600 โอมห์ พลังงานที่จ่ายให้กับคู่สายโทรศัพท์คือ

$P_{load} = e_s^2 / Z$

โดยที่

P_{load} = พลังงาน (วัตต์)

e_s = ระดับสัญญาณ (โวลท์)

Z = อิมพีแดนซ์ (โอมห์)

ในวงจรโทรศัพท์ระดับอ้างอิงคือพลังงาน 1 mW ที่จ่ายให้กับโหลด ดังนั้นถ้า P_{load} = 1 mW และ Z = 600 โอมห์ e_s = 0 775 โวลท์ ซึ่งก็คือระดับสัญญาณที่ทำให้เกิดพลังงาน 1 mW จ่ายให้กับสายโทรศัพท์

2. <u>สัญญาณควบคุม</u>

สัญญาณควบคุมอาจจะเป็นสัญญาณความถี่เดียวหรือหลายความถี่ นอกจากนี้ยังประกอบไปด้วยจังหวะเปิดและปิด สัณญาณเหล่านี้เป็นสัญญาณที่ส่งจากชุมสายไปยังคู่สายซึ่งสรุปความถี่และชนิดของสัญญาณดังตารางที่ 2.2

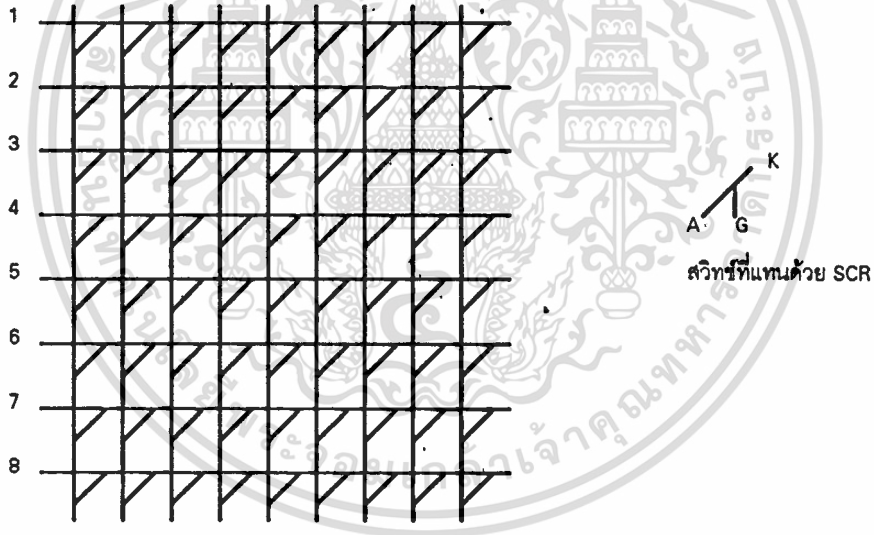
ตารางที่ 2.2 แสดงความถี่และจังหวะของสัญญานควบคุม

สัญญาน	ความถี่ (Hz)	ช่วงเปิด (sec)	ช่วงปิด (sec)
Dial	350 + 440	ต่อเนื่อง	
Busy	480 + 620	0.5	0.5
Ringback	440 + 480	1	4

สัญญานเรียกเป็นสัญญานที่มีความถี่ประมาณ 16-25 Hz มีขนาดประมาณ 90 Vrms หรือสูงกว่า สัญญานนี้จะส่งมาจากชุมสายเมื่อมีการเรียกมายังสายนั้น โทรศัพท์รุ่นเก่ามักใช้สัญญานนี้ขับกระดิ่งให้ดัง แต่โทรศัพท์รุ่นใหม่จะมีวงจรตรวจสัญญานนี้และกำเนิดเสียงเรียกเอง

2.1 .3 **สวิทช์ของชุมสาย (Crosspoint switching)**

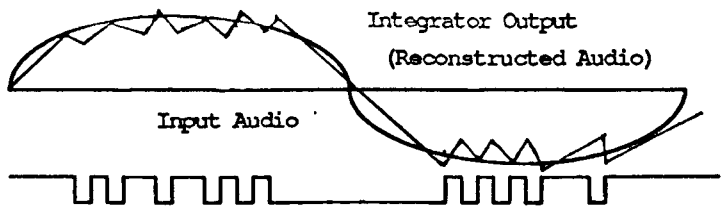
ส่วนนี้เป็นส่วนที่ทำหน้าที่เชื่อมต่อคู่สนทนาเข้าด้วยกัน โครงสร้างโดยทั่วไปเป็นดังรูปที่ 2.4 โดยโครงสร้างจะมีลักษณะ 8 คอลัมน์ 4 แถว ในปัจจุบันอุปกรณ์ที่นำมาทำเป็นสวิทช์มักเป็นสารกึ่งตัวนำ ซึ่งการเปิดปิดสวิทช์แต่ละตัวจะถูกควบคุมโดยไมโครโปรเซสเซอร์



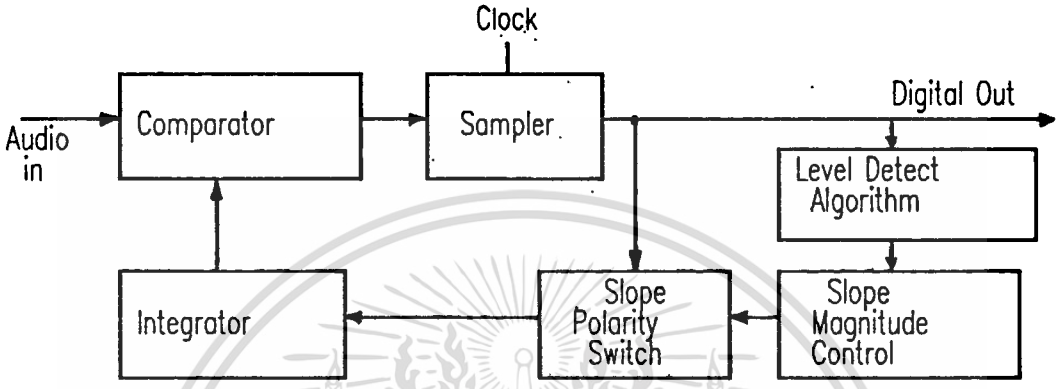
รูปที่ 2.4 แสดงโครงสร้างของ Crosspoint switch

2..2 **<u>การแปลงสัญญานระหว่างอะนาลอกและดิจิตอล</u>**

การแปลงสัญญานระหว่างดิจิตอลและอะนาลอกทำได้หลายวิธี ในที่นี้จะกล่าวแต่เฉพาะวิธีที่ใช้ในงานวิจัยชิ้นนี้คือ วิธี Continuously Variable Slope Delta modulator-demodulator (CVSDs) ตัวอย่างของการแปลงสัญญานรูปซายน์เป็นดิจิตอลอนุกรมเป็นดังรูปข้างล่างนี้ บล็อกไดอะแกรมของวิธีการนี้เป็นดังรูปที่ 2.5 สำหรับการเข้ารหัส และเป็นดังรูป 2.6 สำหรับการถอดรหัส

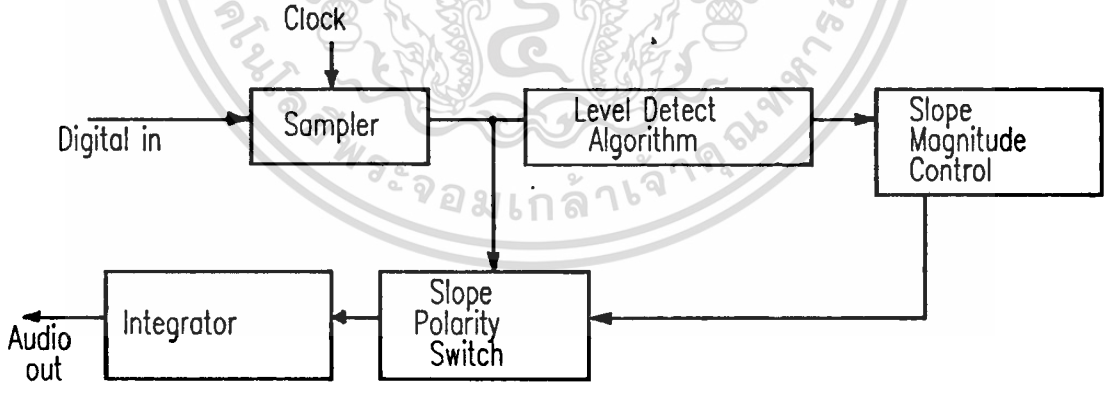


แสดงสัญญานดิจิตอลอนุกรมที่ได้จากการแปลงสัญญานซาย์



รูปที่ 2.5 แสดงการเข้ารหัสของวิธี CSVD

ส่วนทำหน้าที่เข้ารหัสของ CVSD ประกอบด้วย comparator และ integrator สัญญานอินพุทที่ป้อนให้กับ comparator คือสัญญานอะนาลอกที่จะทำการแปลงและเอาท์พุทจาก integrator ซึ่งจะถูกนำมาเปรียบเทียบกันโดยจะให้เอาท์พุทเป็นสัญญานดิจิตอล โดยมี clock เป็นตัวควบคุมจังหวะการเปรียบเทียบ



รูปที่ 2.6 แสดงการถอดรหัสของวิธี CSVD

กระบวนการถอดรหัสมีองค์ประกอบคล้ายๆกับการเข้ารหัส ในงานวิจัยนี้เลือกใช้ไอซีของโมโตโรลา เบอร์ MC3417 [21] ซึ่งมีการแปลงสัญญานโดยวิธี CVSD นี้ เนื่องจากเป็นไอซีที่สามารถหาได้ง่ายและรวมการถอดรหัสและเข้ารหัสไว้ในตัวเดียวกัน รายละเอียดโดยสมบูรณ์ศึกษาได้จากข้อมูลของไอซีตัวนี้ในภาคผนวก 5

2.3 การเชื่อมต่อกับเครื่องไมโครคอมพิวเตอร์ [6], [15], [19]

ในงานวิจัยนี้ใช้การรับส่งข้อมูลแบบ 8 บิท ในที่นี้จึงจะกล่าวแต่เฉพาะสัญญานบนสลอตของเครื่อง 8 บิท สัญญานที่เกี่ยวข้องบนสล็อตของเครื่องไมโครคอมพิวเตอร์ เป็นดังนี้

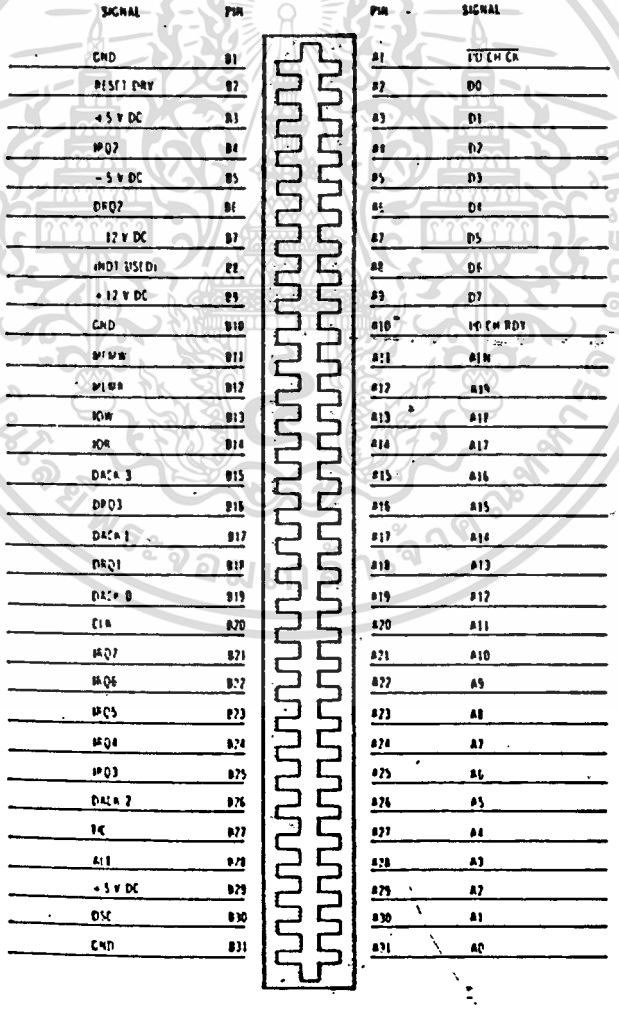
A0 - A19 เป็นแอดเดรสบิทที่ 0 ถึง 19 ขานี้จะแอคทีฟเมื่อขาสัญญาน ALE มีสถานะเป็น 1 และจะถูกแลทไว้ ตรงขอบขาลงของสัญญานนี้ สัญญานในส่วนนนี้ทำให้อ้างแอดเดรสได้ 1 MB

IOR เป็นขาสัญญานที่บอกให้อุปกรณ์ I/O ที่ต่ออยู่ ทำการส่งข้อมูลลงมาที่บัสข้อมูล

IOW เป็นขาสัญญานที่บอกให้อุปกรณ์ I/O ที่ต่ออยู่ ทำการเก็บข้อมูลจากบัสข้อมูลไป

RESET DRV เป็นขาสัญญานที่แอคทีฟตอนช่วงที่เราเริ่มจ่ายไฟให้กับระบบเพื่อใช้ในการรีเซต CPU และอุปกรณ์ต่างๆในระบบคอมพิวเตอร์ รวมทั้งอุปกรณ์ I/O ที่ต่ออยู่ด้วย

AEN ขาสัญญานนี้จะแอคทีฟเมื่อตัวควบคุม DMA ได้ทำการควบคุมบัสต่างๆของระบบแล้ว ดังนั้นการอ้างพอร์ทของอุปกรณ์ I/O จะต้องใช้สัญญานนี้ในการถอดรหัสด้วย เพื่อไม่ให้เกิดการดิดต่อระหว่างระบบกับอุปกรณ์ I/O ตัวอื่นยกเว้นตัวที่กำลังทำกระบวนการ DMA อยู่



รูปที่ 2.7 แสดงสล็อตและสัญญานต่างๆของเครื่องไมโครคอมพิวเตอร์

2.4 การรับส่งข้อมูลอะซิงโครนัสโดยใช้ไมโครคอนโทรลเลอร์ 8031

8031 เป็นไมโครคอนโทรลเลอร์ในตระกูล MCS-51 ซึ่งมีวงจรรับส่งสัญญานแบบอะซิงโครนัสอยู่ภายใน ซึ่งควบคุมการทำงานได้โดยคำสั่งของตัวไมโครคอนโทรลเลอร์เอง ทำให้สะดวกต่อการใช้งาน ในที่นี้ใช้ไมโครคอนโทรลเลอร์ตัวนี้ใช้การส่งข่าวสารซึ่งพีซีส่งมาให้จ่ายให้กับคอมพิวเตอร์จำนวน 4 เครื่องผ่านทางพอร์ท RS-232C ในที่นี้จะกล่าวแต่สถาปัตยกรรมของไมโครคอนโทรลเลอร์นี้ในส่วนที่เกี่ยวกับการสื่อสารแบบอะซิงโครนัส

การควบคุมพอร์ทอนุกรมของ MCS-51 กระทำผ่านฟังก์ชันรีจีสเตอร์ SCON ซึ่งมีรายละเอียดดังรูปที่ 2.8

SM0	SM1	SM2	REN	TBB	RBB	T	H

SM0	SM1	โหมด	ลักษณะการทำงาน	อัตราบอด
0	0	0	ชิฟต์รีจีสเตอร์	f (OSC) / 12
0	1	1	8 บิท UART	เปลี่ยนแปลงตามการเลือกไทม์เมอร์
1	0	2	9 บิท UART	f (OSC) / 64 หรือ f (OSC) / 32
1	1	3	9 บิท UART	เปลี่ยนแปลงได้อิสระ

SM2 ใช้เมื่อต้องการต่อร่วมกับหลายๆไมโครโปรเซสเซอร์

REN เป็น 1 เมื่อต้องการให้อีเนเบิ้ลการรับข้อมูล

TI เป็นอินเทอร์แฟลกในการส่งซึ่งถูกเซตด้วยฮาร์ดแวร์ หลังจากโปรแกรมส่งข้อมูลออกไปแล้วจะต้องเคลียร์บิทนี้เอง

RI เป็นอินเทอร์แฟลกในการรับซึ่งถูกเซตด้วยฮาร์ดแวร์ หลังจากโปรแกรมรับข้อมูลเข้ามาแล้วจะต้องเคลียร์บิทนี้เอง

รูปที่ 2.8 SCON รีจีสเตอร์ควบคุมพอร์ทอนุกรม

การกำเนิดสัญญานนาฬิกาเพื่อใช้ในการรับส่งข้อมูลเพื่อกำหนดอัตราบอด (baud rate) สามารถกำหนดจากตัวไทม์เมอร์ซึ่งมีอยู่ในตัว MSC-51 โหมดการทำงานในการส่งข้อมูลแบบอนุกรมมีได้ 4 โหมด ในที่นี้จะใช้เฉพาะโหมด 1

ในโหมด 1 ข้อมูลที่ใช้ส่งหรือรับ มีขนาด 10 บิท ซึ่งประกอบด้วยบิทเริ่มต้น (0) 1 บิท บิทข้อมูล 8 บิท การรับส่งข้อมูลกระทำผ่านขา TXD และ RXD ข้อมูลในการรับส่งจะถูกอ่านเขียนจาก SBUF ซึ่งถูกมองเป็นหน่วยความจำหนึ่งตำแหน่งหนึ่งในไมโครคอนโทรลเลอร์ การรับส่งจะกระทำโดยการเขียนคำสั่งเพื่ออ่านเขียน SBUF นี้

รายละเอียดเกี่ยวกับสถาปัตยกรรมและคำสั่งหาอ่านเพิ่มเติมได้จากหนังสืออ้างอิงท้ายเล่ม [11]

บทที่ 3
ฮาร์ดแวร์ของระบบ

บทนี้จะกล่าวถึงฮาร์ดแวร์ของวงจรทั้งหมด โดยเริ่มจากแนวคิดระดับบล็อกไดอะแกรม การทำงานในรายละเอียดของแต่ละส่วนจะถูกควบคุมโดยซอฟต์แวร์ซึ่งจะได้กล่าวถึงในบทที่ 4 ต่อไป

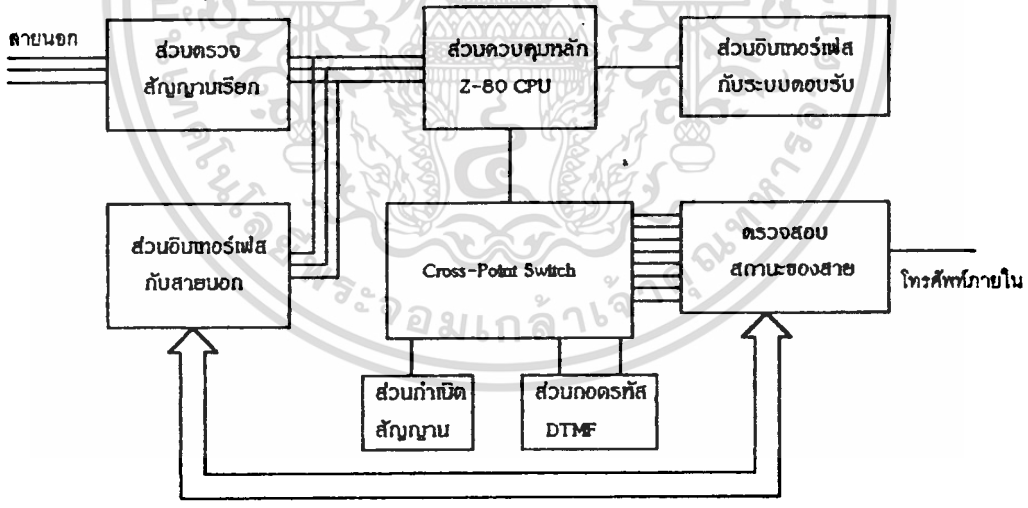
3.1 แนวความคิดในการออกแบบ

จุดมุ่งหมายในการออกแบบเน้นที่ความสามารถในการทำงานแบบแยกส่วนได้ กล่าวคือฮาร์ดแวร์ในส่วนทั้งสามสามารถทำงานแยกเป็นอิสระจากกันหรือนำมาประกอบกันทำงานร่วมเป็นระบบเดียวกันได้ ในบทนี้จะเริ่มจากโครงสร้างฮาร์ดแวร์ของชุมสาย ฮาร์ดแวร์ของระบบตอบรับโทรศัพท์ และฮาร์ดแวร์ของระบบแจ้งข่าวสารตามลำดับ

3.2 ฮาร์ดแวร์ของชุมสาย

ชุมสายโทรศัพท์ที่พัฒนาขึ้นนี้มีคุณสมบัติพื้นฐานเท่าที่จำเป็นในการใช้งาน คือสามารถโทรติดต่อกันภายในได้ โทรติดต่อกับสายนอกและโอนสายได้ จุดมุ่งหมายคือนำเสนอส่วนเสริมให้กับชุมสายโทรศัพท์เพื่อให้สามารถติดต่อกับระบบตอบรับโทรศัพท์ได้

ชุมสายโทรศัพท์เป็นขนาด 3 สายนอก 8 สายใน ควบคุมโดยไมโครโปรเซสเซอร์ ใช้งานกับโทรศัพท์แบบกดปุ่ม โครงสร้างของชุมสายในระดับบล็อกไดอะแกรมเป็นดังรูปที่ 3.1

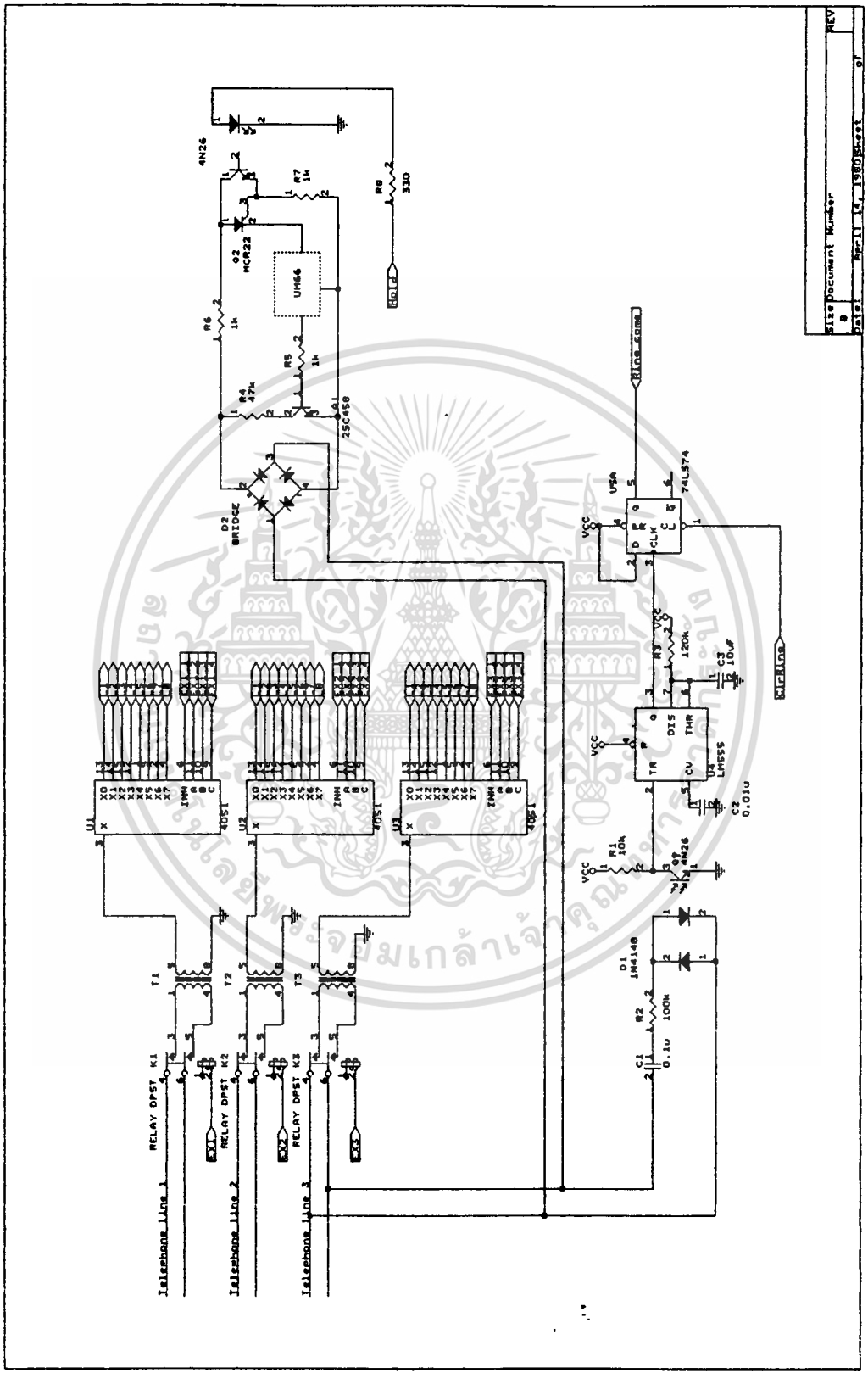


รูปที่ 3.1 บล็อกไดอะแกรมของชุมสาย

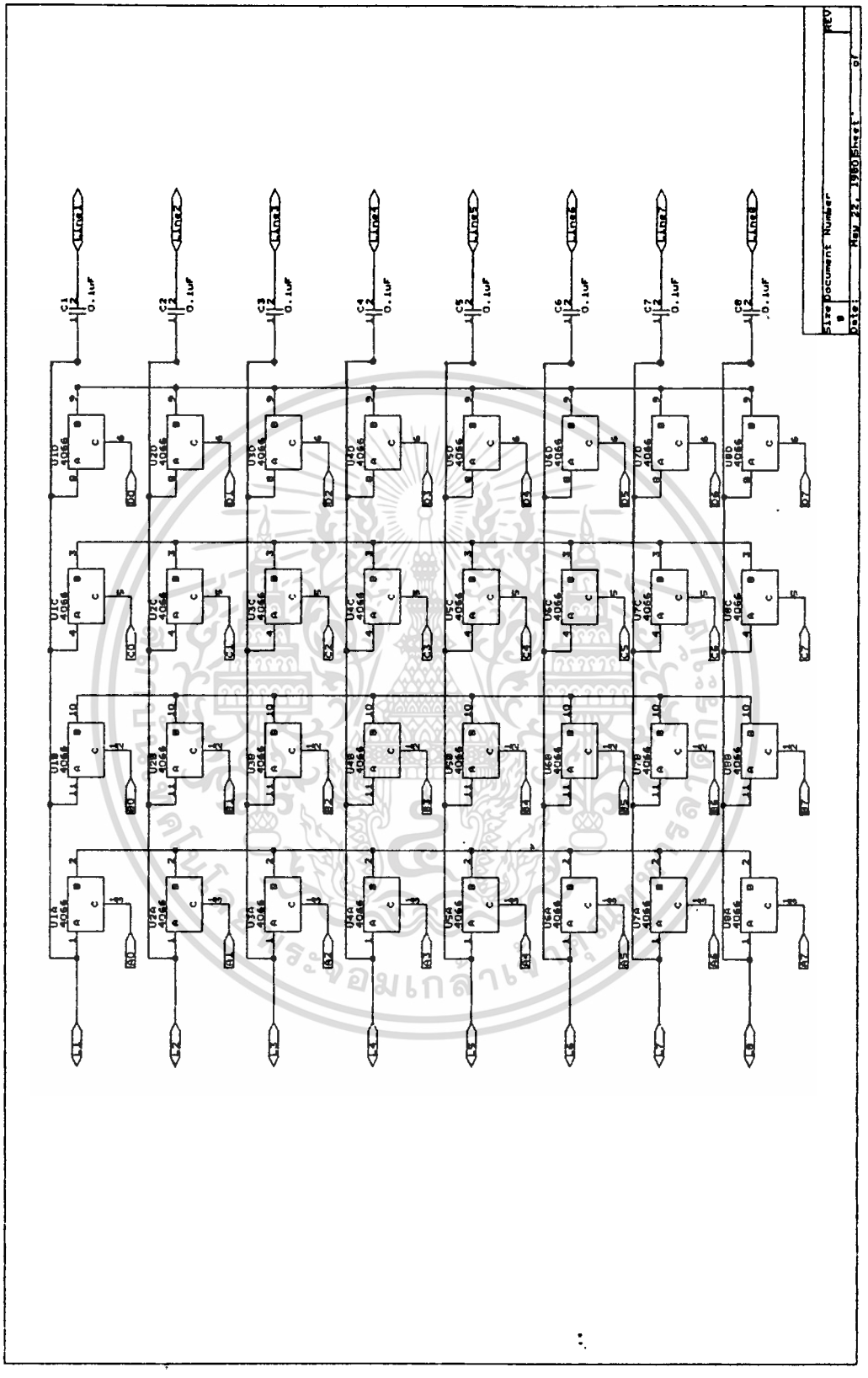
การทำงานของส่วนต่าง

3.2.1 ส่วนตรวจสอบสัญญาณเรียก

ส่วนนี้จะให้พัลซ์ที่มีความกว้างประมาณ 1 วินาทีตามจังหวะของสัญญาณเรียกจากสายนอกที่เข้ามา วงจรส่วนนี้มีจำนวน 3 วงจร

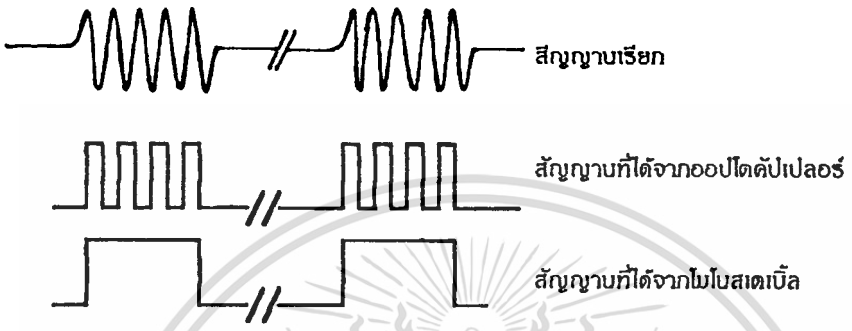


วงจรที่ 3.1 วงจรตรวจสัญญาณเรียก วงจรติดต่อกับสายนอก วงจรพักสายนอก



วงจรที่ 3.2 วงจรสวิทช์ของทุ่มสาย

เมื่อมีสัญญานเรียกซึ่งเป็นพัลซ์ที่มีความถี่ประมาณ 20 Hz และมีขนาดประมาณ 90 Vrms จะทำให้เกิดพัลซ์ระดับ TTL ที่มีความถี่เท่ากันที่ขาคอลเลกเตอร์ของออปโตคัปเปลอร์ ซึ่งจะไปทริกโมโนสเตเบิ้ล ได้เป็นพัลซ์ที่มีความกว้างประมาณ 1 วินาทีดังไดอะแกรมเวลาในรูป 3.2 และวงจรเป็นดังวงจรที่ 3.1 ซึ่งประกอบด้วย 4N26, U4 และ U5A รวมทั้งหมด 3 วงจร (ในรูปแสดงเพียง 1 วงจร ที่เหลือเป็นแบบเดียวกัน)



รูปที่ 3.2 ไดอะแกรมเวลาของสัญญาณที่เกี่ยวข้อง

3.2.2 **ส่วนสวิทช์ของชุมสาย**

วงจรในส่วนนี้ใช้ไอซี CMOS 4066 มาต่อกันเป็นอะเรย์วงจรเป็นดังวงจรที่ 3.2 เนื่องจากสายในมีจำนวน 8 สายจึงมีคู่สนทนาได้ 4 คู่ จึงจัดสวิทช์เป็น 8 แถว 4 หลัก โดยแต่ละแถวจะต่ออยู่กับสายในแต่ละสาย

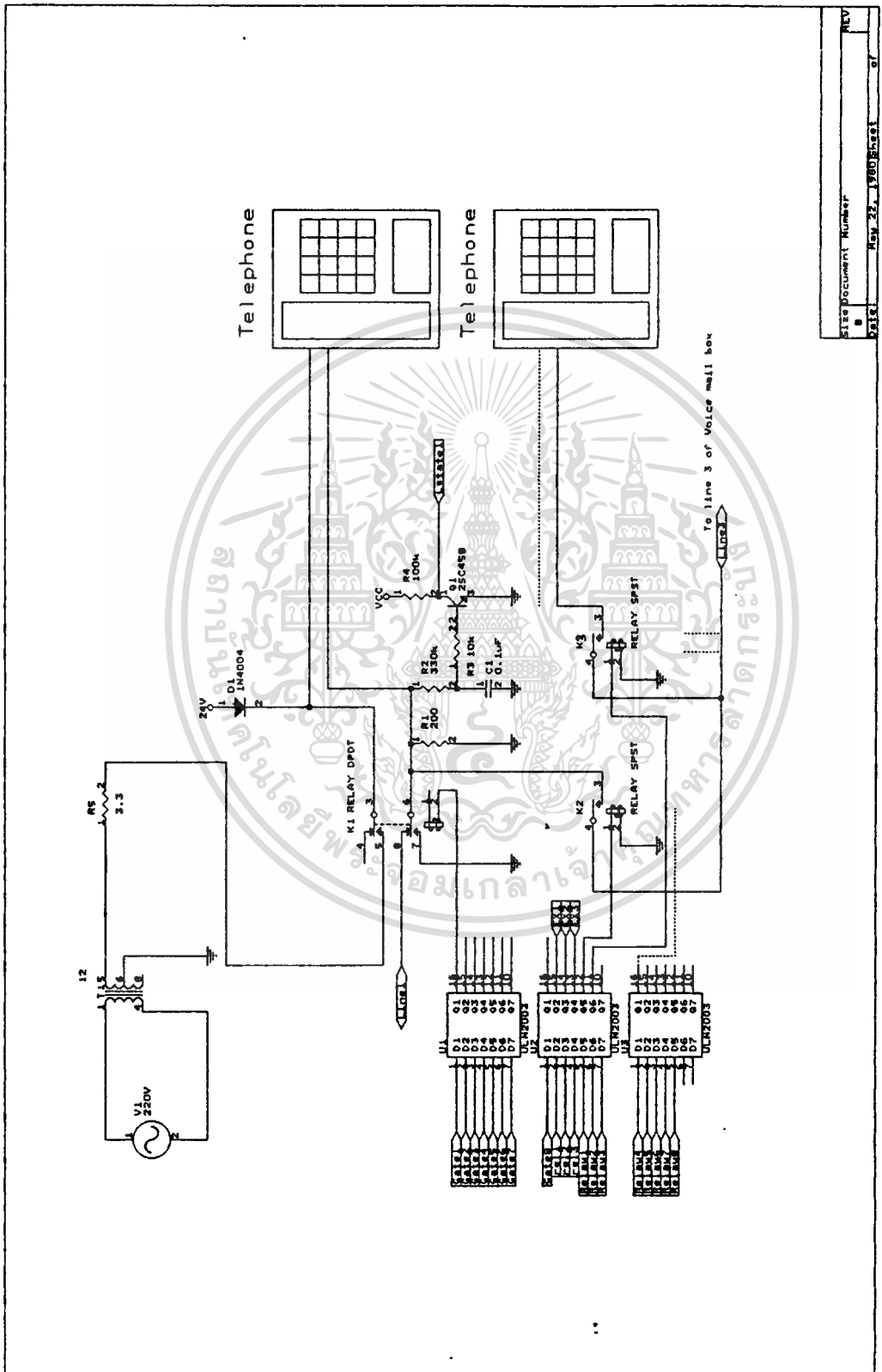
การต่อในลักษณะนี้ไม่ได้ใช้วงจรดีโคดในทางแถวแต่จะควบคุมการเปิดปิดสวิทช์แต่ละตัวด้วยบิทควบคุมของพอร์ทอเนกประสงค์ 8255 โดยถ้าปิดสวิทช์เชื่อมแถวสองแถวในหลักใดหลักหนึ่งก็จะเป็นการเชื่อมคู่สนทนาที่ต่ออยู่กับแถวทั้งสองนั้น

3.2.3 **ส่วนตรวจสอบสภาวะของสายใน**

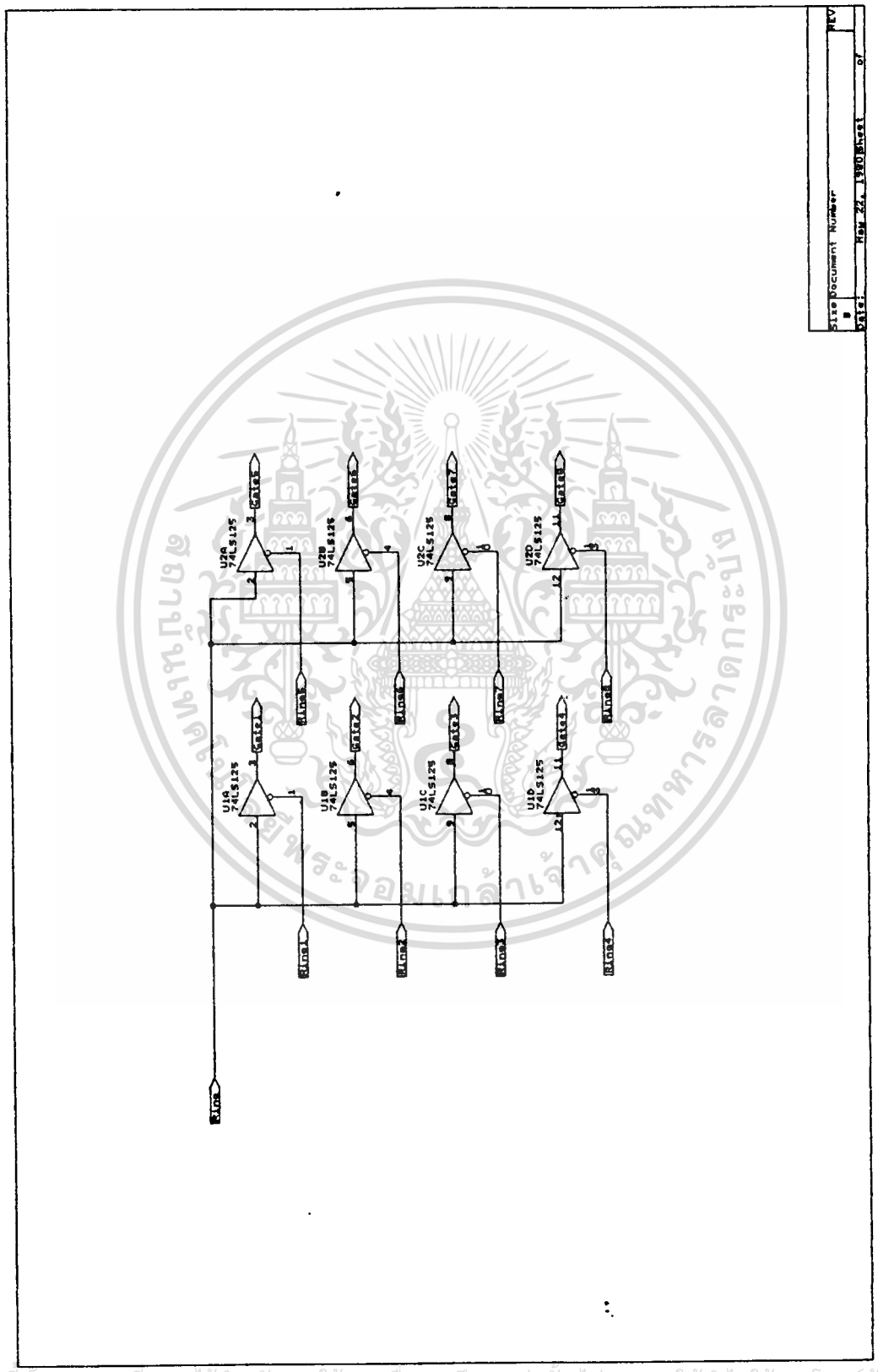
วงจรส่วนนี้จะบอกสถานะการยกหูและวางหูของแต่ละสายใน โดยจะให้ลอจิก 0 เมื่อมีการยกหู และ 1 เมื่อวางหูอยู่ วงจรส่วนนี้ยังประกอบด้วยรีเลย์ เพื่อทำหน้าที่ส่งสัญญานเรียกไปยังชุมสาย การแยกแยะว่าเป็นการยกหูหรือวางหูพิจารณาจากการเปลี่ยนจากการเปลี่ยน 0 ไปเป็น 1 หรือ 1 ไปเป็น 0 โดยจะกล่าวรายละเอียดในส่วนซอฟต์แวร์ต่อไป วงจรส่วนนี้ได้แก่วงจรของทรานซิสเตอร์ Q1 ในวงจรที่ 3.3

จากวงจรที่ 3.4 การส่งสัญญานเรียกไปยังโทรศัพท์เครื่องใดทำได้โดยการเปิดเกตของสายนั้น ซึ่งจะทำให้สัญญาน Ring ซึ่งเป็นสัญญานเรียกระดับ TTL ผ่านไปยัง ULN2003 เพื่อขับรีเลย์ให้เปิดปิดตามจังหวะของสัญญานเรียก โดยจังหวะที่รีเลย์ปิดจะมีสัญญานเรียกโวลเตจสูงผ่านไปยังโทรศัพท์เครื่องนั้น

สัญญานเรียกซึ่งเป็นไฟสลับสร้างจากการแปลงไฟ 220 V ผ่านหม้อแปลง T1 และ T2 ในวงจรที่ 3.3 ทำให้ได้เอาท์พุทประมาณ 100 V นำมาเป็นสัญญานเรียก



วงจรที่ 3.3 วงจรตรวจสถานะของสายใน วงจรสร้างสัญญาณเรียก วงจรเลือกสายในต่อเข้ากับระบบตอบรับ



วงจรที่ 3.4 วงจรกำหนดจังหวะของสัญญาณเรียก

3.2.4 ส่วนถอดรหัสสัญญาน DTMF

การถอดรหัสจะใช้ไอซีเบอร์ MT8870 ซึ่งรหัสที่ถอดได้เป็นเลข BCD ในวงจรใช้วงจรถอดรหัส 1 วงจร ต่อสายใน 4 สาย การมัลติเพล็กกระทำผ่านไอซี CMOS 4052 ซึ่งเป็นอะนาลอกมัลติเพล็กเซอร์แบบ 4x1 จำนวน 2 ตัวใน 1 แพ็คเกจ วงจรเป็นดังรูปที่ 3.5 ปุ่มที่กดและรหัสที่ถอดได้เป็นดังตารางที่ 3.2 การเลือกสายในเพื่อการถอดรหัสและการแก้ปัญหาจากการกดค้างเป็นหน้าที่ของโปรแกรมซึ่งรายละเอียดอยู่ในหัวข้อ 4.1.4

ตารางที่ 3.2 แสดงรหัส BCD ที่ได้จากการถอดรหัสของแต่ละปุ่ม

ปุ่มที่กด	รหัส BCD
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
0	1010
*	1011
#	1100

การถอดรหัสเริ่มโดยการสแกนมัลติเพล็กเซอร์โดย Z-80 จากแชนแนลที่ 1 ถึง 4 ในการสแกนแต่ละแชนแนลจะได้สัญญานจากสายใน 2 สายเข้าไปถอดรหัสพร้อมกัน หากมีสัญญาน DTMF เกิดขึ้น จะทำให้สัญญานที่ขา STD ของ MT8870 เป็น 1 ซึ่งจะไปทริก 74LS74 ให้แลทช์ไว้ เพื่อบอกให้ Z-80 ทราบว่ามีสัญญาน DTMF เกิดขึ้นในสายที่ทำการสแกนโดยที่ Z-80 จะอ่านค่ารหัสที่อ่านได้นี้ไปผ่านกระบวนการถัดไป

3.2.5 ส่วนอินเทอร์เฟสกับระบบตอบรับโทรศัพท์

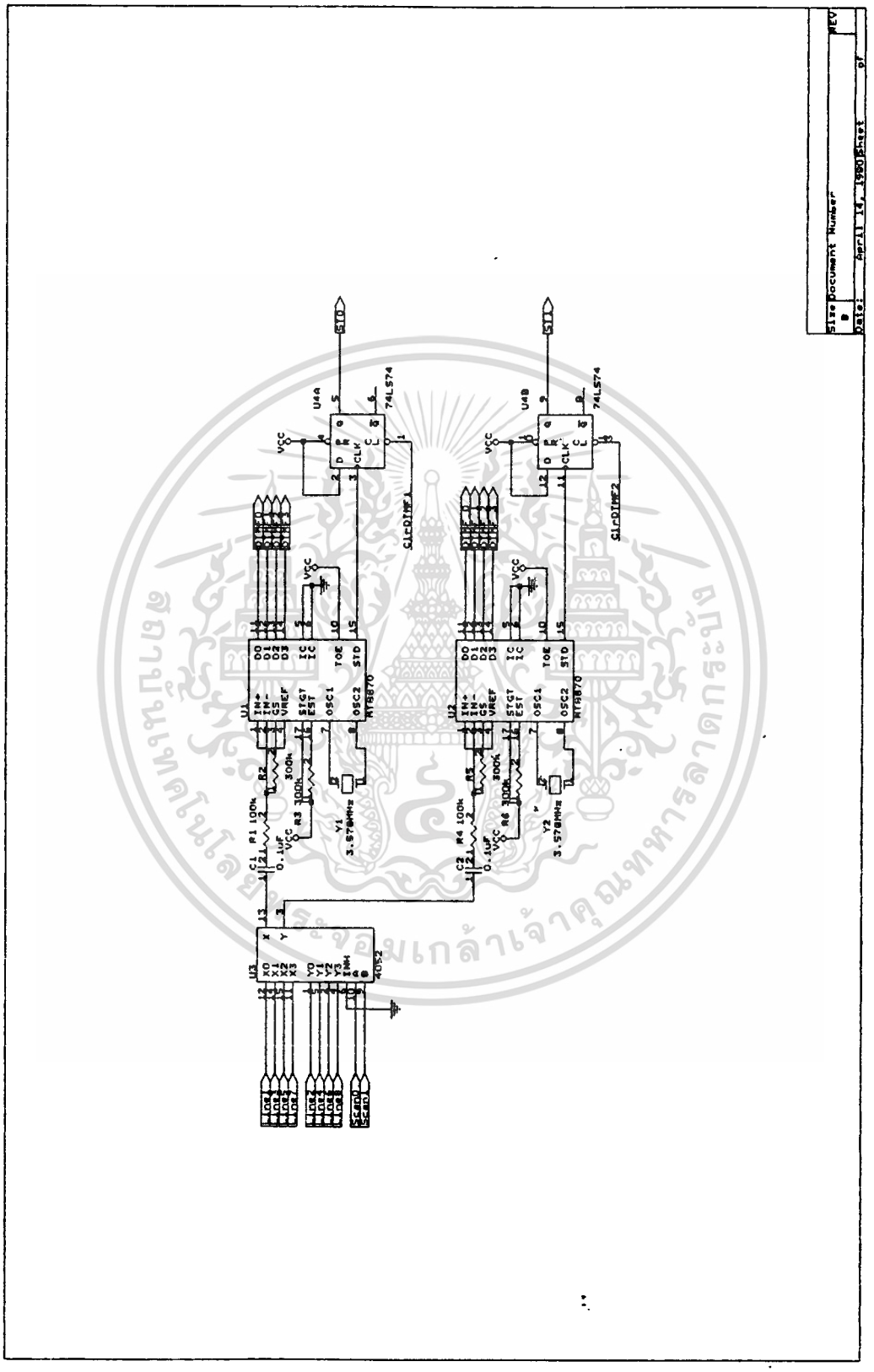
ส่วนนี้มีสองส่วน ส่วนแรกเป็นพอร์ทส่งและรับคำสั่งกับระบบตอบรับโทรศัพท์จำนวน 2 พอร์ท โดยใช้ 8255 ในวงจรที่ 3.7 พอร์ทส่งคำสั่งได้แก่พอร์ท C ของ U9 พอร์ทรับคำสั่งได้แก่ 4 บิทบนของพอร์ท B ของ U9 และ 3 บิทบนของพอร์ท B ของ U11

พอร์ทส่งคำสั่ง ส่งคำสั่งจากชุมสายไปยังระบบตอบรับโทรศัพท์ :

PC 7 บิทนี้ใช้ในกระบวนการรับส่งข้อมูลระหว่างพีซีและ 8031 (รายละเอียดอยู่ในหัวข้อ 3.3)

PC 4-6 เป็นเบอร์สายในที่ออกคำสั่งใน 4 บิทล่าง (มีค่า 0 ถึง 7)

PC 0-3 เป็นรหัสคำสั่งมีความหมายดังนี้



วงจรที่ 3.5 วงจรถอดรหัสสัญญาน DTMF

0000	สายใน 4 บิทล่างวางหูจากการใช้ระบบตอบรับ
0001	สายใน 4 บิทล่างต้องการจะบันทึกเสียง
0010	สายใน 4 บิทล่างต้องการจะเล่นกลับเสียง
0011	สายใน 4 บิทล่างต้องการจะโปรแกรมการโทรออก
0100	สายใน 4 บิทล่างต้องการให้ระบบตอบรับรับสายแทน
0101	สายใน 4 บิทล่างต้องการจะรับสายเอง
1001	ชุมสายไม่มีคำสั่งใดที่ส่งให้ระบบตอบรับ
1010	ชุมสายได้รับคำสั่งจากระบบตอบรับแล้ว

พอร์ทรับคำสั่ง รับคำสั่งจากระบบตอบรับโทรศัพท์ (มีขนาด 1 ไบท์ แต่ทั้ง 8 บิทไม่ได้อยู่ในไบท์เดียวกัน)

PB 7-6 ของ U9 เป็นรหัสคำสั่งจากระบบตอบรับมีความหมายดังนี้

00	ระบบตอบรับได้รับคำสั่งจากชุมสายแล้ว
01	ระบบตอบรับสั่งให้ชุมสายทำการโอนสายนอกที่อยู่ใน PB 5-4 ของ U9 ให้กับสายในที่อยู่ใน PC 7-5 ของ U11
10	ระบบตอบรับปฏิเสธคำสั่งที่ชุมสายส่งมา
11	ระบบตอบรับพร้อมรับคำสั่ง

6 บิทที่เหลือเป็นพาราเมตเตอร์ของคำสั่งที่มีรหัสเป็น 01 ดังกล่าวไว้ข้างต้น

ปกติเมื่อระบบตอบรับโทรศัพท์พร้อมที่จะรับคำสั่งจากชุมสายก็จะส่งรหัส 11 มายังพอร์ทรับคำสั่งของชุมสาย โดยก่อนที่ชุมสายจะส่งคำสั่งออกไปจะต้องอ่านพอร์ทนี้ก่อน ถ้าพบว่ามีค่าดังกล่าวจึงจะส่งคำสั่งออกไป โดยคำสั่งที่ส่งออกไปจะมีขนาด 1 ไบท์ 4 บิทล่างเป็นรหัสคำสั่ง และ 4 บิทบนเป็นหมายเลขของสายในที่ส่งคำสั่งนั้น แล้วชุมสายจะรอให้ระบบตอบรับตอบกลับมาว่าได้รับคำสั่งแล้ว โดยการอ่านที่พอร์ทรับคำสั่งอีกครั้ง ซึ่งหากคำสั่งได้รับเรียบร้อยแล้วก็จะส่งรหัส 00 กลับมาให้ เป็นอันจบกระบวนการส่งคำสั่งจากชุมสายไปยังระบบตอบรับ หากระบบตอบรับปฏิเสธคำสั่งนั้นก็จะส่งรหัส 10 มาให้ ซึ่งเป็นผลให้ชุมสายส่งสัญญานสายไม่ว่างไปให้กับสายที่ส่งคำสั่งนั้นมา

เมื่อระบบตอบรับต้องการจะส่งคำสั่งไปยังชุมสายบ้างก็จะใช้กระบวนการคล้ายๆกัน โดยจะอ่านพอร์ทส่งคำสั่งของชุมสายว่าส่งรหัส 1001 ซึ่งบ่งบอกว่าขณะนี้ชุมสายไม่มีคำสั่งจากชุมสายมาหรือไม่ ถ้าพบก็จะส่งคำสั่งที่ต้องการมายังชุมสาย แล้วรอให้ชุมสายตอบรับด้วยรหัส 1010 ซึ่งหมายความว่าชุมสายได้รับคำสั่งนั้นแล้ว

ส่วนที่สองของวงจรอินเทอร์เฟสกับระบบตอบรับโทรศัพท์เป็นวงจรเลือกสายในที่จะต่อเข้ากับแชนแนลที่ 3 ของระบบตอบรับโทรศัพท์ ซึ่งในวงจรใช้รีเลย์เป็นสวิทช์ในการเลือก วงจรนี้รวมอยู่ในวงจรที่ 3.3

3.2.6 **ส่วนต่อสายนอกเข้ากับสายใน**

ส่วนนี้ใช้ไอซี CMOS 4051 จำนวน 3 ตัวทำการเลือกสัญญานสายนอกแต่ละสายเข้ากับสายในแต่ละสาย วงจรเป็นดังวงจรที่ 3.1 ซึ่งประกอบด้วย U1, U2 และ U3

ส่วนที่เกี่ยวกับสายนอกอีกส่วนหนึ่งคือวงจรพักสาย โดยประกอบด้วย SCR เบอร์ MCR22 ไอซีเสียงเพลง UM66 ซึ่งการพักสายจะถูกควบคุมด้วยบิทคำสั่งจาก 8255 ผ่านทางออปโตคัปเปอร์ ซึ่งวงจรรวมอยู่ในรูปวงจรที่ 3.1

3.2.7 **ส่วนกำเนิดสัญญาน**

สัญญานที่สร้างขึ้นมี 4 แบบ ได้แก่ สัญญานให้หมุน (dial tone) สัญญานเรียกกลับ (ring back) และสัญญานสายไม่ว่าง (busy tone) และสัญญานเรียกระดับ TTL โดยสร้างขึ้นจากไอซีไทม์เมอร์ 556 สองตัว ดังรูปวงจรที่ 3.6 ไอซี CMOS 4051 ทำหน้าที่เลือกแบบของสัญญานที่จะส่งไปให้กับสายในแต่ละสาย ซึ่งการเลือกจะถูกควบคุมโดยบิทควบคุมของ 8255

3.2.8 **ส่วนควบคุมหลัก**

ส่วนนี้จะทำหน้าที่ควบคุมการทำงานทั้งหมดของชุมสาย ประกอบไปด้วย Z-80 CPU ทำงานที่ความถี่ 4 MHz หน่วยความจำ ROM ขนาด 32 KB หน่วยความจำ RAM ขนาด 8 KB พอร์ท 8255 จำนวน 5 พอร์ท วงจรเป็นดังรูปที่ 3.7 หน่วยความจำ RAM มีแอดเดรสตั้งแต่ 8000 ถึง 9fffH พอร์ท 8255 มีรายละเอียดดังนี้

<u>พอร์ท 1</u> ฐานอยู่ที่ตำแหน่ง 00H หมายเลขอ้างอิงในวงจร U6

หมายเลข 00H เป็นพอร์ทอินพุท แต่ละบิทจะอ่านสถานะการยกหูหรือวางหูของสายใน โดย 1 หมายถึงวางหู 0 หมายถึงยกหู

หมายเลข 01H เป็นพอร์ทเอาท์พุท แต่ละบิทจะใช้สั่งให้เชื่อมต่อสายในที่ตรงกับบิทนั้นๆเข้ากับแชนแนลที่ 3 ของระบบตอบรับโทรศัพท์

หมายเลข 02H เป็นพอร์ทอินพุท 4 บิทล่างใช้อ่านรหัส DTMF ของวงจรถอดรหัสตัวที่ 1 และ 4 บิทล่างใช้อ่านรหัส DTMF ของวงจรถอดรหัสตัวที่ 2

<u>พอร์ท 2</u> ฐานอยู่ที่ตำแหน่ง 04H หมายเลขอ้างอิงในวงจร U7

หมายเลข 04H เป็นพอร์ทเอาท์พุท แต่ละบิทจะใช้เปิดปิดสวิทช์ แต่ละตัวในหลักที่ 1 ของสวิทช์ชุมสาย

หมายเลข 05H เป็นพอร์ทเอาท์พุท แต่ละบิทจะใช้เปิดปิดสวิทช์ แต่ละตัวในหลักที่ 2 ของสวิทช์ชุมสาย

หมายเลข 06H เป็นพอร์ทเอาท์พุท แต่ละบิทจะใช้เปิดปิดสวิทช์ แต่ละตัวในหลักที่ 3 ของสวิทช์ชุมสาย โดยแต่ละบิทจะควบคุมการเปิดปิดของ 4066 โดย 1 หมายถึงเปิด

<u>พอร์ท 3</u> ฐานอยู่ที่ตำแหน่ง 08H หมายเลขอ้างอิงในวงจร U8

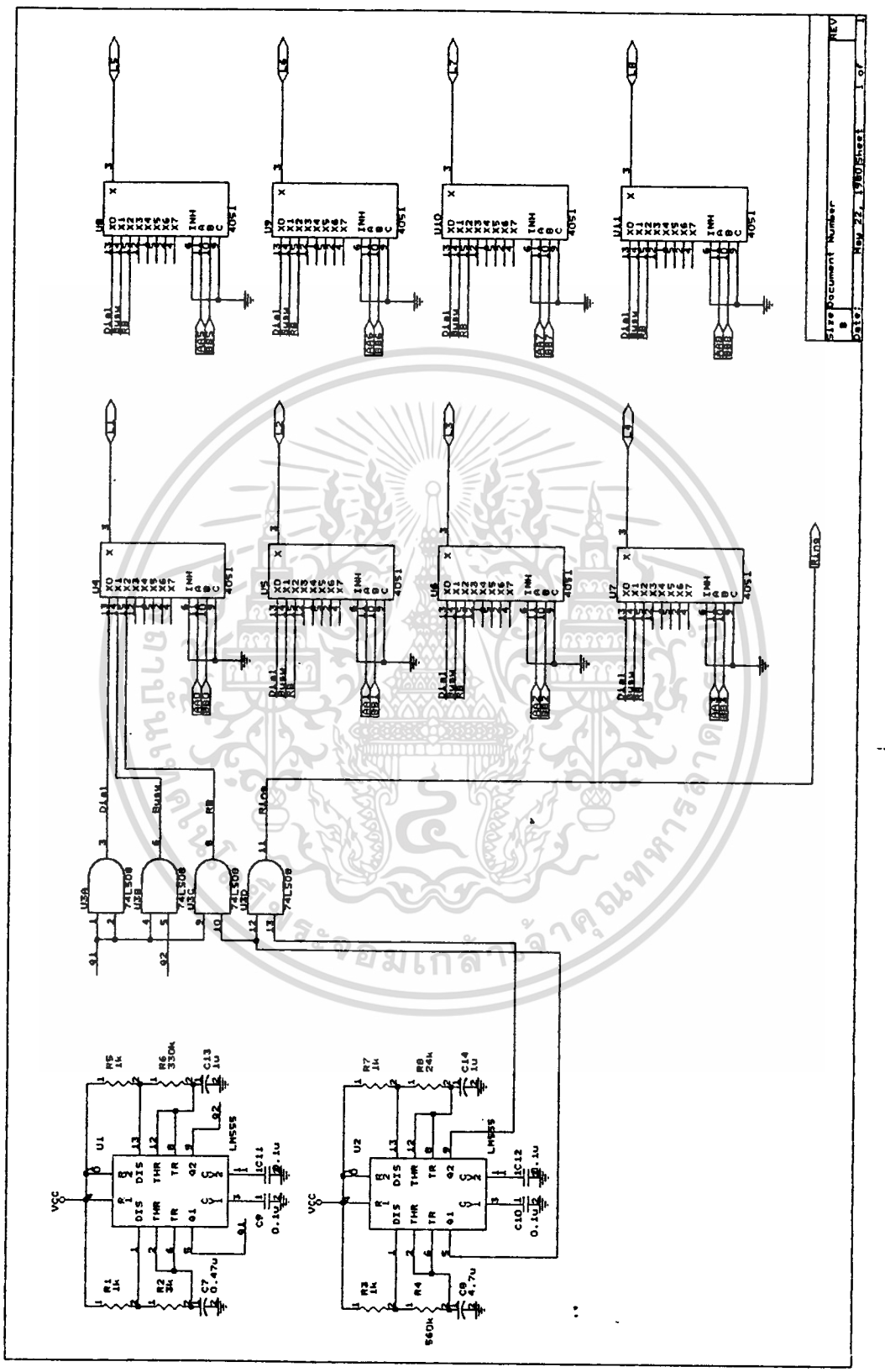
หมายเลข 08H เป็นพอร์ทเอาท์พุท แต่ละบิทจะใช้เปิดปิดสวิทช์ แต่ละตัวในหลักที่ 4 ของสวิทช์ชุมสาย

หมายเลข 09H เป็นพอร์ทเอาท์พุท แต่ละบิทจะเป็นบิทต่ำเป็นลอจิกที่ใช้เลือกสัญญานที่ส่งให้กับสายใน

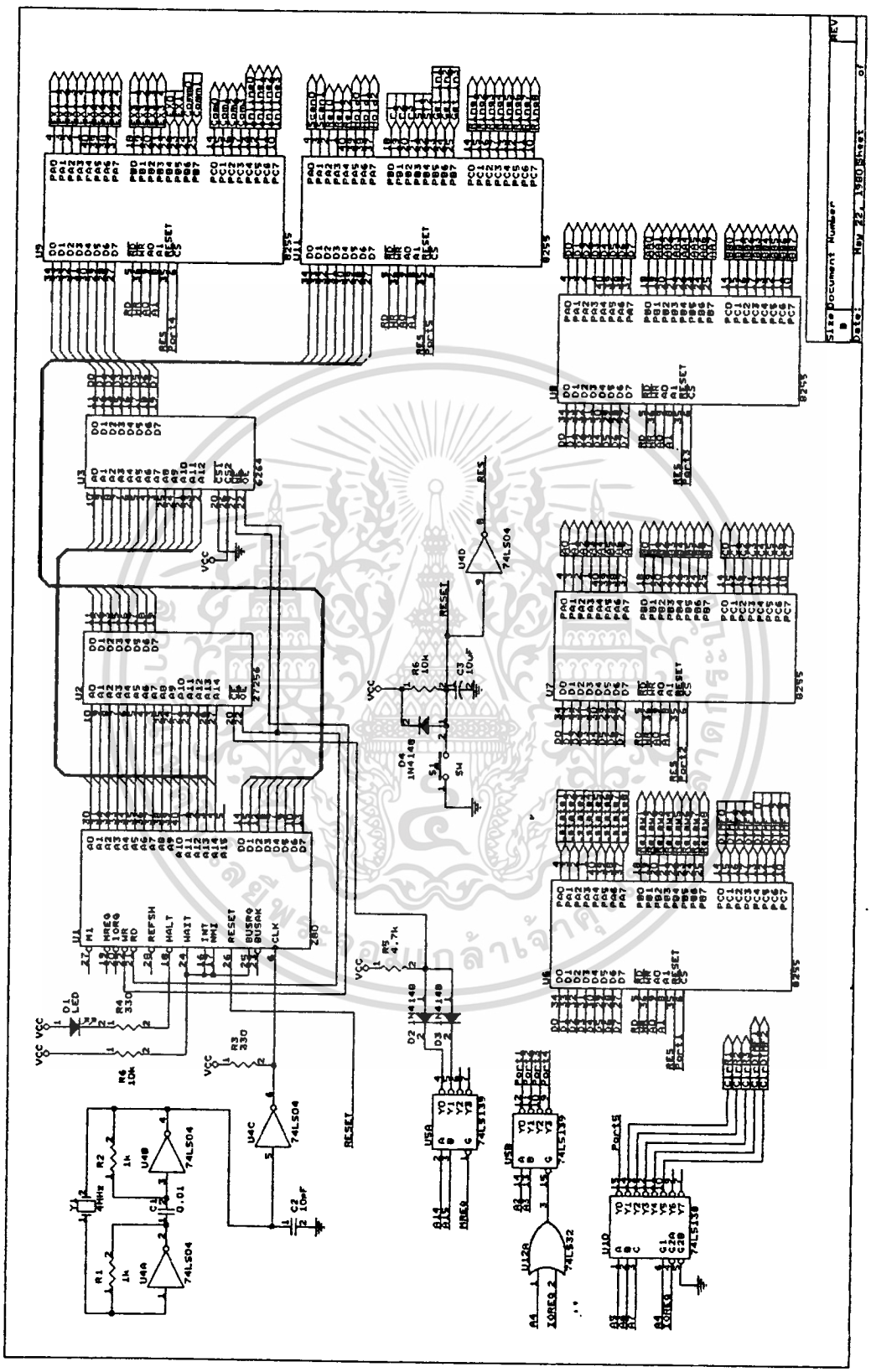
หมายเลข 0AH เป็นพอร์ทเอาท์พุท แต่ละบิทจะเป็นบิทสูงเป็นลอจิกที่ใช้เลือกสัญญานที่ส่งให้กับสายใน

<u>พอร์ท 4</u> ฐานอยู่ที่ตำแหน่ง 0CH หมายเลขอ้างอิงในวงจร U9

หมายเลข 0CH 4 บิทล่างเป็นพอร์ทเอาท์พุทใช้เลือกสายในที่จะต่อกับสายนอกที่ 1 และ 4 บิทล่างเป็นพอร์ทอินพุท 2 บิทแรกใช้รับคำสั่งจากชุมสาย 2 บิทต่อมาใช้ในกรณีที่ระบบตอบรับสั่งให้



วงจรที่ 3.6 วงจรกำเนิดสัญญาณ



วงจรที่ 3 7 วงจรควบคุมหลัก

ชุมสายทำการโอนสายให้ โดยจะบอกหมายเลขสายนอกที่จะโอน หมายเลขสายในดูจากพอร์ท 5

หมายเลข 0DH 4 บิทบนเป็นพอร์ทเอาท์พุทใช้เลือกสายในที่จะต่อกับสายนอกที่ 3 และ 4 บิทล่างเป็นพอร์ทอินพุทใช้อ่านคำสั่งจากระบบตอบรับโทรศัพท์

หมายเลข 0EH เป็นพอร์ทเอาท์พุท 4 บิทล่างใช้ส่งคำสั่งไปยังระบบตอบรับโทรศัพท์ 4 บิทล่างใช้ส่งพารามเตอร์ของคำสั่งนั้น

พอร์ท 5 ฐานอยู่ที่ตำแหน่ง 10H หมายเลขอ้างอิงในวงจร U11

หมายเลข 10H เป็นพอร์ทเอาท์พุท 2 บิทแรกใช้เลือกสัญญาน DTMF จากสายในจำนวน 4 สายมา 1 สาย เพื่อทำการถอดรหัส 3 บิทถัดมาใช้เปิดปิดรีเลย์ที่ต่อสายนอกเข้ากับวงจรของชุมสาย 3 บิทสุดท้ายใช้พักสายนอกทั้ง 3 สาย

หมายเลข 11H เป็นพอร์ทอินพุท 3 บิทแรกใช้อ่านดูว่าสายนอกใดมีสัญญานเรียก 2 บิทถัดมาใช้อ่านดูว่าตัวถอดรหัส DTMF ใดที่พบรหัส DTMF 3 บิทต่อมาเป็นเบอร์สายในที่ระบบตอบรับสั่งให้โอนสายให้

หมายเลข 12H เป็นพอร์ทเอาท์พุทใช้สั่งให้เกิดสัญญานเรียกในสายในแต่ละสาย

3.3 ฮาร์ดแวร์ของระบบตอบรับโทรศัพท์

ระบบตอบรับโทรศัพท์ที่พัฒนาขึ้นนี้มีแชนแนลของการบันทึกและเล่นกลับจำนวน 3 แชนแนล มีลักษณะเป็นอินเทอร์เฟสการ์ดของเครื่องไมโครคอมพิวเตอร์ที่มีช่องสำหรับต่อสายโทรศัพท์จำนวน 3 สาย การกำหนดการใช้งานในแต่ละแชนแนลทำได้โดยการเขียนโปรแกรมควบคุมภาษาแอสแซมบลี้บนเครื่องไมโครคอมพิวเตอร์ ในงานวิจัยนี้กำหนดให้แชนแนลที่ 1 และ 2 ใช้สำหรับตอบรับสายนอก โอนสาย แจ้งข้อความและรับฝากข้อความเมื่อผู้ที่โอนสายไปให้ไม่อยู่ แชนแนลที่ 3 ใช้สำหรับสายภายในเพื่อฝากข้อความไว้ให้ผู้ที่ทำการโทรเข้ามาติดต่อกับสายในในแต่ละสาย และใช้รับฟังข้อความที่มีผู้โทรจากสายนอกฝากไว้ให้ และใช้โปรแกรมเพื่อการโทรออก

ระบบมีองค์ประกอบหลักดังต่อไปนี้

1 ส่วนแปลงสัญญานเสียงระหว่างอะนาลอกและดิจิตอล

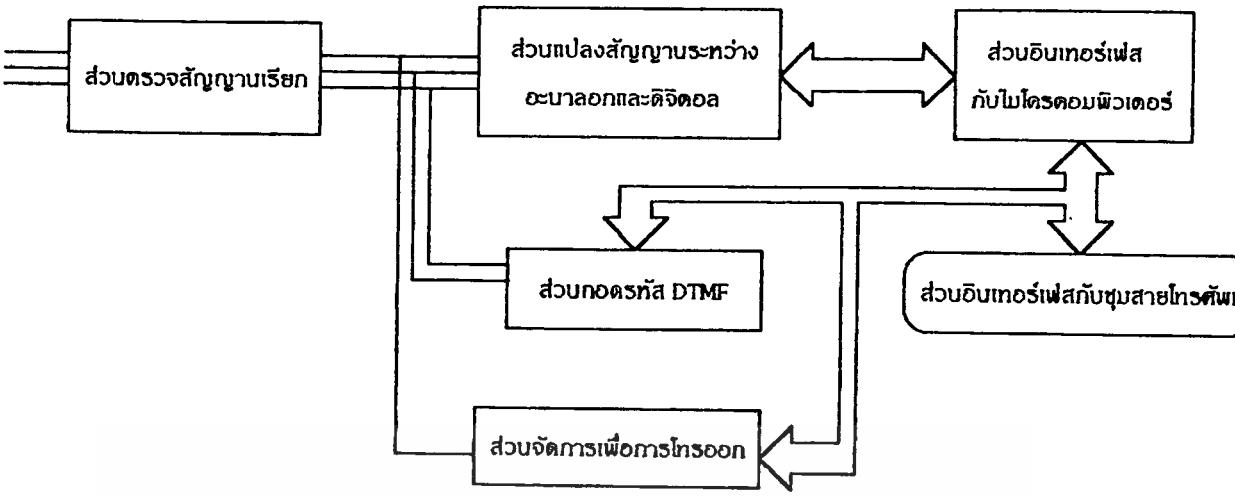
2. ส่วนตรวจสัญญานเรียก

3. ส่วนถอดรหัส DTMF

4. ส่วนอินเทอร์เฟสกับไมโครคอมพิวเตอร์

5. ส่วนอินเทอร์เฟสกับชุมสายโทรศัพท์

6. ส่วนจัดการเพื่อการโทรออก



รูปที่ 3.3 ไดอะแกรมของระบบตอบรับโทรศัพท์

การทำงานของส่วนต่างๆ

3.3.1 **ส่วนแปลงสัญญานระหว่างอะนาลอกและดิจิตอล**

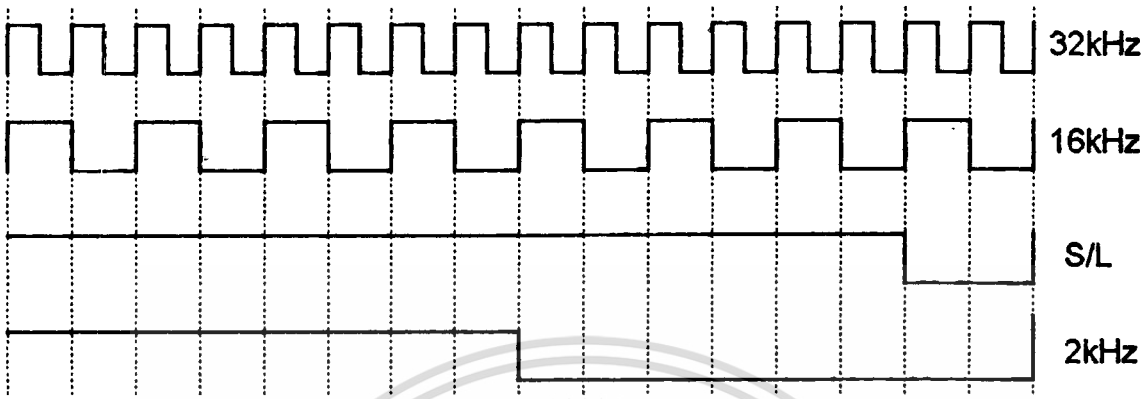
หัวใจของวงจรส่วนนี้ได้แก่ MC3417 ซึ่งเป็นไอซีที่ใช้แปลงสัญญานอะนาลอกเป็นดิจิตอลและจากดิจิตอลเป็นอะนาลอกแบบ CVSDs ซึ่งจะให้เอาท์พุท 1 บิท ต่อการแซมปลิ้ง 1 ครั้ง เพื่อให้ทำการรับส่งข้อมูลกับพีซีได้ จึงต้องมีวงจรซึ่งทำหน้าที่แปลงสัญญานจากอนุกรมเป็นขนานและจากขนานเป็นอนุกรม โดยในการบันทึกสัญญานอนุกรมจะผ่านไปสู่วงจรแปลงจากอนุกรมเป็นขนาน และในการเล่นกลับข้อมูลแบบขนานจากพีซีจะถูกแปลงเป็นอนุกรมเพื่อส่งให้กับไอซีนี้

วงจรในส่วนนี้มีจำนวน 3 ชุดหรือ 3 แชนแนล ซึ่งอินพุท (จุดที่สัญญานเสียงเข้าเพื่อทำการแปลงเป็นดิจิตอล) และ เอาท์พุท (จุดที่สัญญานเสียงออกที่ได้จากการแปลงจากดิจิตอล) ของแต่ละแชนแนลจะต่ออยู่กับสายนอกและสายในของชุมสายจำนวน 3 สายผ่านมัลติเพล็กเซอร์ซึ่งอยู่ในชุมสาย

จากรูป 3.8 สัญญาน ในสายโทรศัพท์ซึ่งผ่านมาจากรีเลย์ในส่วนของวงจรตรวจสัญญานเรียกจะคัปปลิ้งผ่านหม้อแปลง T1 ไปยังขดลวดที่สอง ผ่าน U1A และ U2 เข้าสู่ U3 ในกรณีที่จะทำการบันทึกเสียง ในทางกลับกันเมื่อจะเล่นกลับ สัญญานเสียงจะออกจาก U3 ไปสู่ U4, U10 และ U1B คัปปลิ้งผ่าน T1 และเข้าสู่สายโทรศัพท์ U1 เป็นอะนาลอกสวิทช์ ซึ่งจะเปิดปิดสลับกันตามการควบคุมของพีซีเพื่อป้องกันไม่ให้สัญญานที่คัปปลิ้งผ่าน T1 ออสซิลเลต U2 และ U4 ทำหน้าที่กรองสัญญานเสียงโดยมีแบนด์วิทช์ 4 kHz U5 ทำหน้าที่ขยายกำลังของสัญญานที่ได้จากการแปลงจากดิจิตอลเป็นอะนาลอกเพื่อป้อนเข้าสู่สายโทรศัพท์

จากรูปที่ 3.9 สัญญานนาฬิกาที่ใช้ในการแซมปลิ้งกำเนิดจากไอซีไทม์เมอร์ 555 U1 โดยมีความถี่ประมาณ 32 kHz นำมาหาร 2 โดย U2 ได้เป็นความถี่ 16 kHz ซึ่งเป็นความถี่ในการแซมปลิ้งของ MC3417 และเป็นความถี่ในการเลื่อนข้อมูลของ 74LS164 และ 74LS166 นอกจากนี้ยังนำมาหาร 16 ได้เป็นความถี่ประมาณ 2 kHz เพื่อป้อนให้กับพีซีเป็นอัตราการรับส่งข้อมูลและเป็น**สัญญานนาฬิกาหลัก**สำหรับควบคุมลูปของซอฟต์แวร์ในหัวข้อ 4.2

อัตราการแซมปลิ้งที่ได้ประมาณ 2000 ไบท์ต่อวินาทีหรือ 1 ไบท์ทุกๆ 500 us ไดอะแกรมของสัญญานนาฬิกาที่เกี่ยวข้องเป็นดังนี้



รูปที่ 3.4 ไดอะแกรมเวลาของสัญญานนาฬิกาที่ใช้ในระบบ

พิจารณาจากรูปที่ 3.8 อีกครั้งหนึ่ง U5, U6 และ U7 ใช้ในการบันทึกเสียง โดย U5 จะแปลงสัญญานอนุกรมให้เป็นขนาน โดยมีอัตราการเลื่อนข้อมูลเป็น 16 kHz เมื่อเลื่อนได้ครบ 8 บิท ข้อมูลจะ ถูกแลทช์ใน U6 โดยมีอัตราการแลทช์ 2 kHz U7 เป็นบัฟเฟอร์เพื่อรอให้พีซีมาอ่านไป

U8, U12 ใช้ในการเล่นกลับเสียง โดย U8 จะแปลงสัญญานขนานจาก U12 เป็นอนุกรม โดยมีอัตราการเลื่อนข้อมูลเป็น 16 kHz เมื่อเลื่อนได้ครบ 8 บิท ข้อมูลจะถูกโหลดใหม่ ซึ่งจะต้องมีข้อมูลจากพีซีเขียนลงบน U12 อยู่แล้ว โดยพีซีจะต้องเขียนข้อมูลบน U12 ในอัตราประมาณ 2000 ไบท์ต่อวินาที

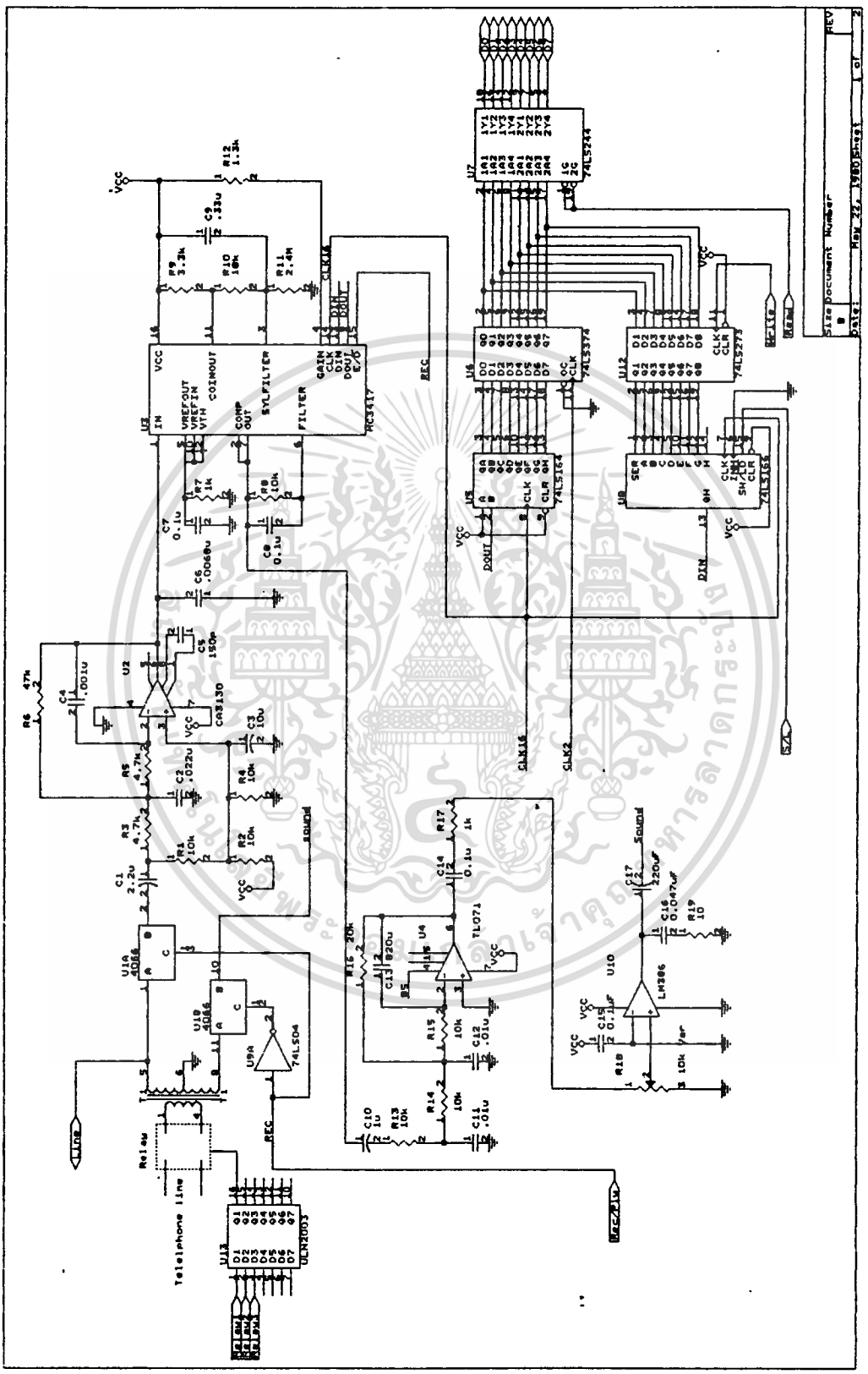
การอ่านข้อมูลและเขียนข้อมูลในการแซมปลิ้งของพีซีจะต้องทำทุกๆประมาณ 500 us ซึ่งพีซีจะต้องทำการโพลเพื่อหาขอบขาขึ้นของสัญญานนาฬิกาที่มีความถี่ 2 kHz เพื่อทำการอ่านเขียนข้อมูล ดังนั้นพีซีจึงมีเวลาประมาณ 500 us ก่อนการอ่านและเขียนครั้งถัดไป ซึ่งเราจะนำเวลาในช่วงนี้มาให้พีซีทำหน้าที่อย่างอื่นได้ ซึ่งรวมถึงการบันทึกและเล่นกลับเสียงทั้งสามแชนแนลพร้อมกัน รายละเอียดในส่วนนี้จะกล่าวถึงในบทซอฟต์แวร์ต่อไป

3.3.2 **ส่วนตรวจสอบสัญญานเรียก**

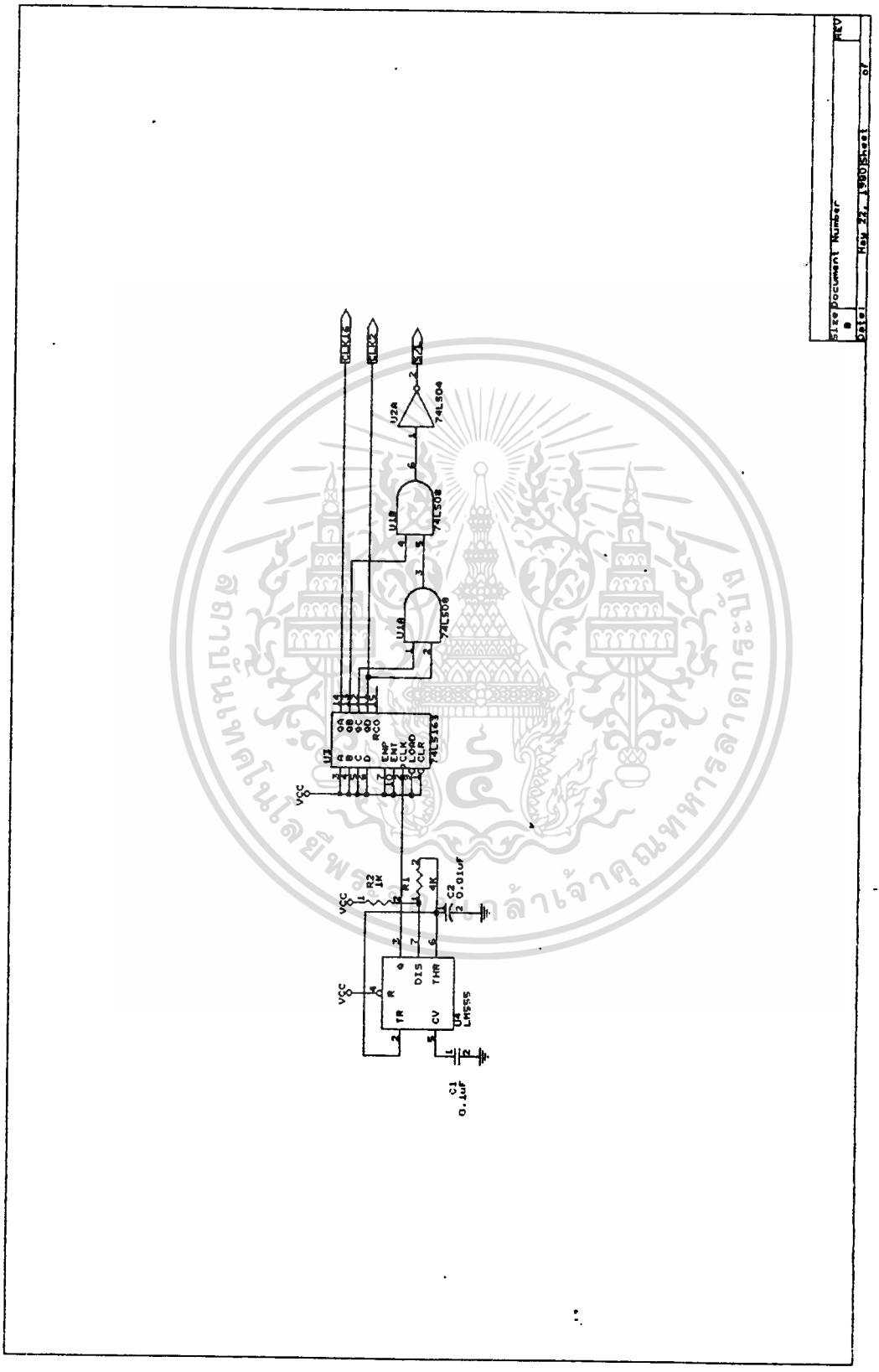
วงจรส่วนนี้มีการทำงานเช่นเดียวกับวงจรตรวจสัญญานเรียกของชุมสายโทรศัพท์ในหัวข้อ 3.2.1 และสำหรับระบบตอบรับโทรศัพท์มีวงจรส่วนนี้ 2 ชุด โดยใช้ตรวจสอบสัญญานเรียกจากสายที่ 1 และ 2 ซึ่งจะเชื่อมโยงอยู่กับแชนแนลที่ 1 และ 2 ของระบบตอบรับ ดังวงจรที่ 3.10

3.3.3 **ส่วนถอดรหัส DTMF**

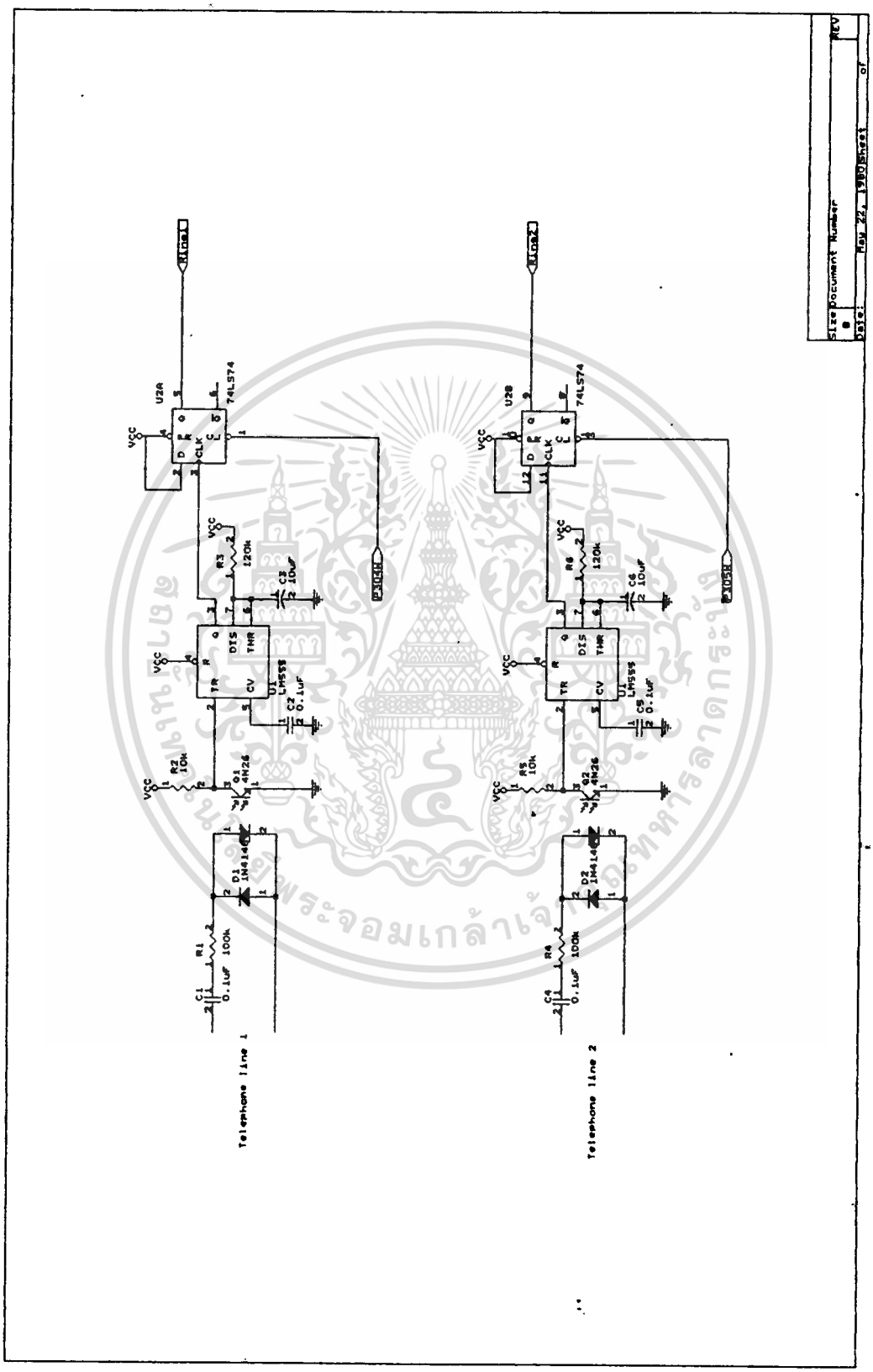
วงจรส่วนนี้มีการทำงานเช่นเดียวกับวงจรถอดรหัสของชุมสายโทรศัพท์ในหัวข้อ 3.2.4 และสำหรับระบบตอบรับนี้มีวงจรส่วนนี้ 2 วงจร ซึ่งใช้สำหรับถอดรหัสจากการกดปุ่มของผู้ใช้สายนอกที่โทรเข้ามา



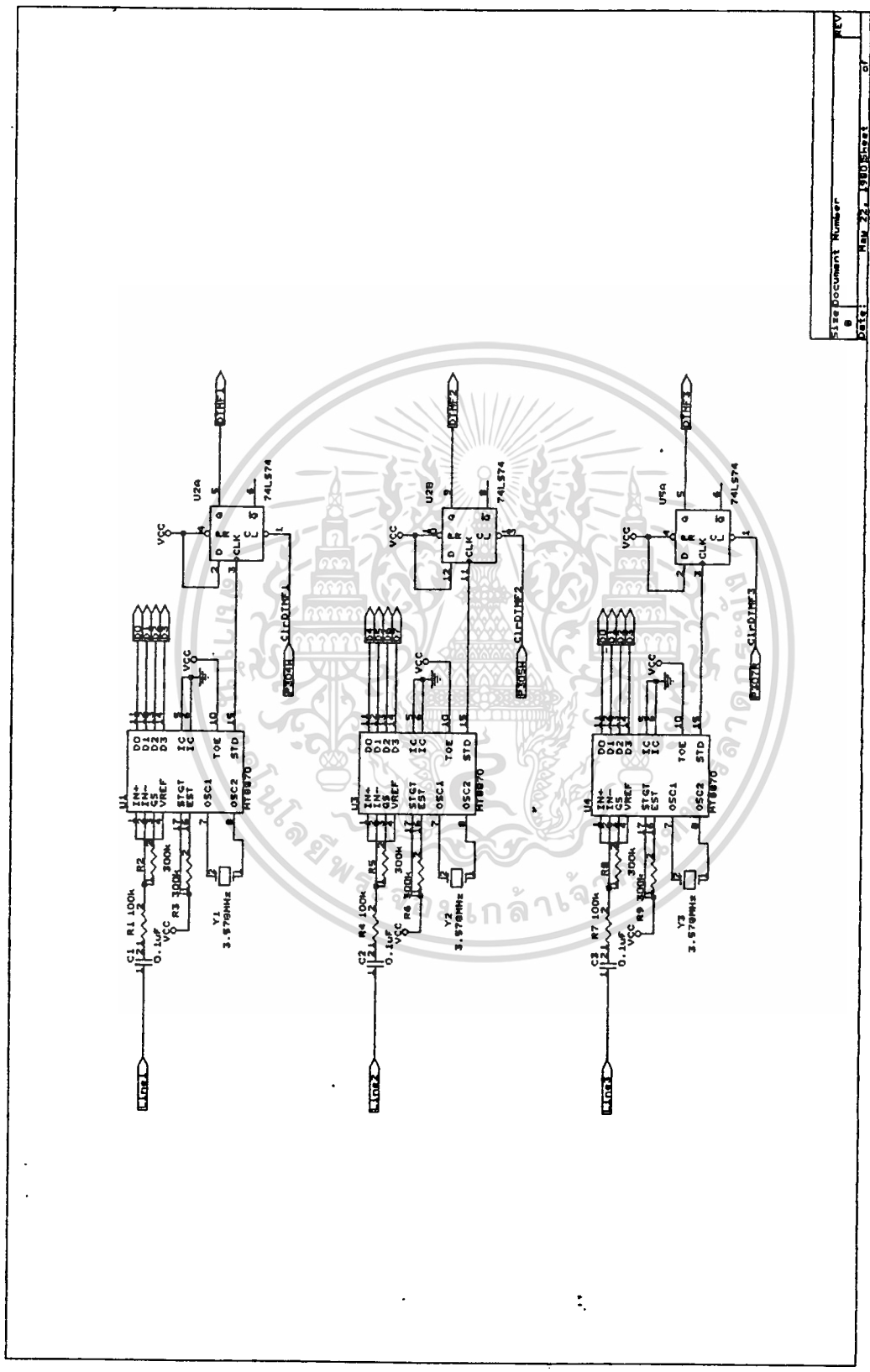
วงจรที่ 38 วงจรแปลงสัญญาณในแต่ละแชนแนล



วงจรที่ 3.9 วงจรกำเนิดสัญญาณควบคุมการแปลงสัญญาน



รูปที่ 3.10 วงจรตรวจสอบสัญญาณเรียก



วงจรที่ 3 11 วงจรถอดรหัสสัญญาน DTMF

3.3.4 **ส่วนอินเทอร์เฟสกับไมโครคอมพิวเตอร์**

วงจรส่วนนี้ทำหน้าที่ถอดรหัสพอร์ทของเครื่องพีซี โดยมีพอร์ทแอดเดรสเริ่มที่ 300H วงจรเป็นดังรูปที่ 3.12 และ 3.13 หมายเลขพอร์ทและความหมายเป็นดังนี้

พอร์ทอ่าน

300H อ่านข้อมูลจาก 74LS244 เพื่อการบันทึกในแชนแนลที่ 1

301H อ่านข้อมูลจาก 74LS244 เพื่อการบันทึกในแชนแนลที่ 2

302H ใช้อ่านข้อมูลเกี่ยวกับสัญญานเรียก และสัญญาน DTMF ดังรายละเอียดต่อไปนี้

บิท 0 ต่อเข้ากับสัญญานนาฬิกาความถี่ 2KHz ซึ่งเป็นสัญญานนาฬิกาหลัก

บิท 1-2 เป็น 1 หมายความว่ามีสัญญานเรียกในแชนแนลที่ 1 และ 2 ตามลำดับ

บิท 3 เป็น 1 หมายถึงได้รับสัญญานเรียกกลับ เป็น 0 หมายถึงได้รับสัญญานสายไม่ว่าง ในกระบวนการโทรออก (ดูหัวข้อ 3.3.6)

บิท 4-6 เป็น 1 หมายถึงถอดรหัส DTMF ได้จากแชนแนลที่ 1 และ 2 ตามลำดับ

บิท 7 ต่อเข้ากับพัลซ์ที่มีช่วงเปิดปิด 500 ms ซึ่งใช้ในกระบวนการโทรออก (ดูหัวข้อ 3.3.6)

303H 4 บิทล่างใช้อ่านรหัส DTMF ที่ถอดได้จากแชนแนลที่ 1

4 บิทบนใช้อ่านรหัส DTMF ที่ถอดได้จากแชนแนลที่ 2

304H 4 บิทล่างใช้อ่านรหัส DTMF ที่ถอดได้จากแชนแนลที่ 3

305H อ่านข้อมูลจาก 74LS244 เพื่อการบันทึกในแชนแนลที่ 3

306H อ่าน**พอร์ทส่งคำสั่ง**ของชุมสาย (ดูรายละเอียดในหัวข้อ 3.2.5 และ 4.2.4)

307H ใช้เคลียร์ฟลิปฟลอปที่ถูกทริกจากสัญญานเรียกและสัญญาน DTMF ในแชนแนลที่ 3

พอร์ทเขียน

300H เขียนข้อมูลลง 74LS273 เพื่อการเล่นกลับในแชนแนลที่ 3

301H ใช้ควบคุมการเปิดปิดรีเลย์และเลือกการบันทึกและเล่นกลับในแชนแนลทั้งสาม

บิท 0 เป็น 1 หมายถึงปิดรีเลย์ในสายที่ 1

บิท 1 เป็น 1 หมายถึงให้บันทึกในแชนแนลที่ 1 เป็น 0 หมายถึงให้เล่นกลับในแชนแนลที่ 1

บิท 2 เป็น 1 หมายถึงปิดรีเลย์ในสายที่ 2

บิท 3 เป็น 1 หมายถึงให้บันทึกในแชนแนลที่ 1 เป็น 0 หมายถึงให้เล่นกลับในแชนแนลที่ 2

บิท 4 เป็น 1 หมายถึงปิดรีเลย์ในสายที่ 3

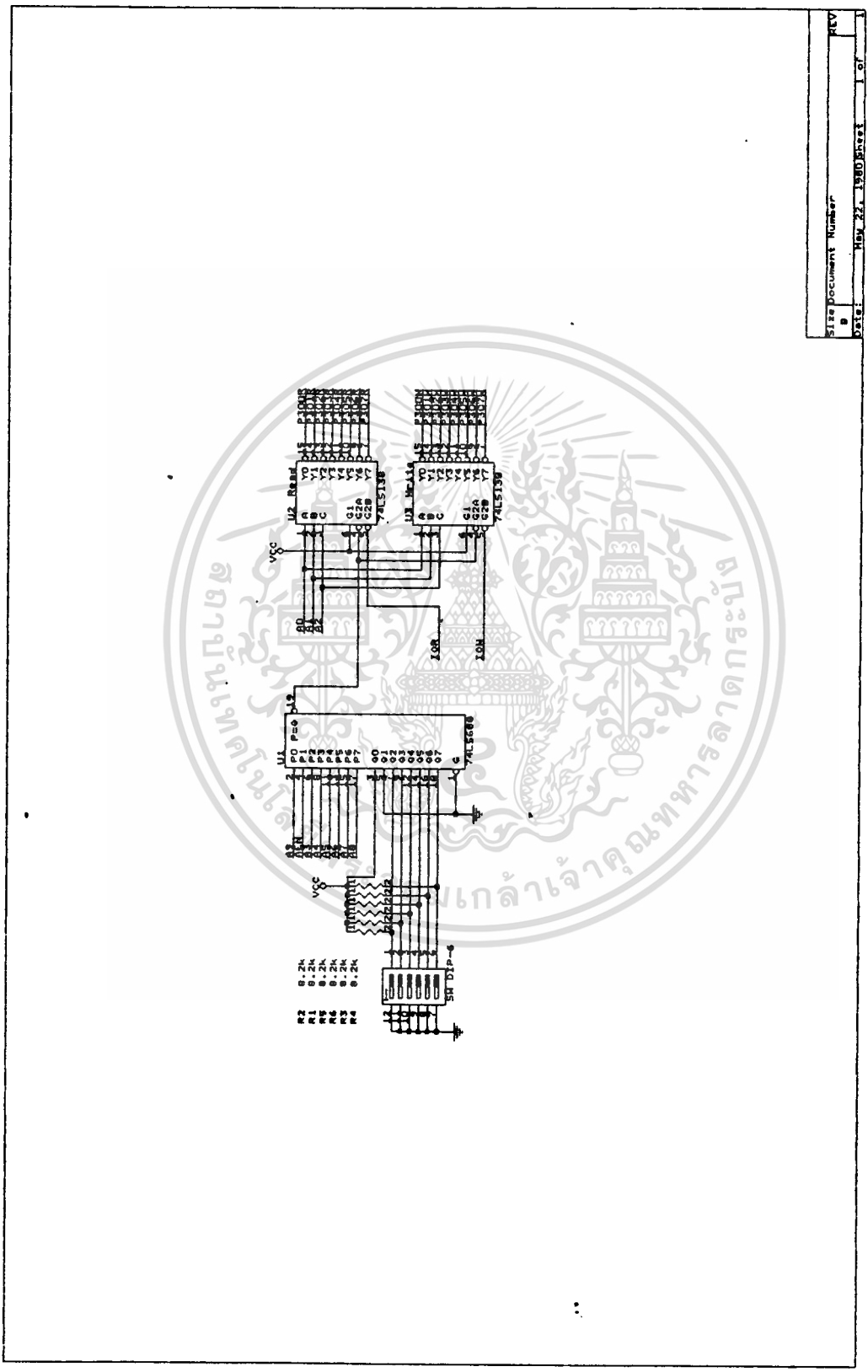
บิท 5 เป็น 1 หมายถึงให้บันทึกในแชนแนลที่ 1 เป็น 0 หมายถึงให้เล่นกลับในแชนแนลที่ 3

302H เขียนข้อมูลลง 74LS273 เพื่อการเล่นกลับในแชนแนลที่ 1

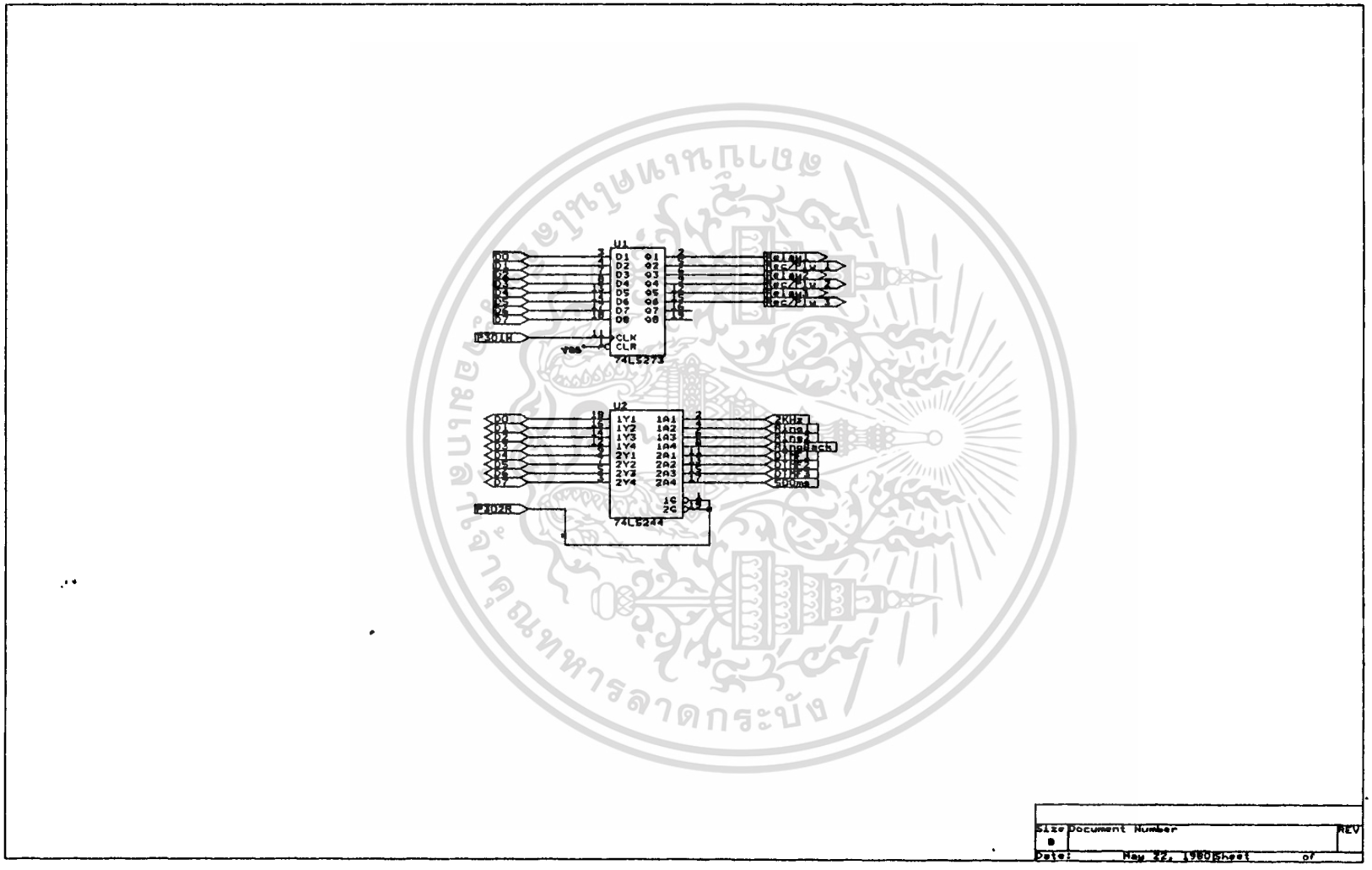
303H เขียนข้อมูลลง 74LS273 เพื่อการเล่นกลับในแชนแนลที่ 2

304H ใช้เคลียร์ฟลิปฟลอปที่ถูกทริกจากสัญญานเรียกและสัญญาน DTMF ในแชนแนลที่ 1

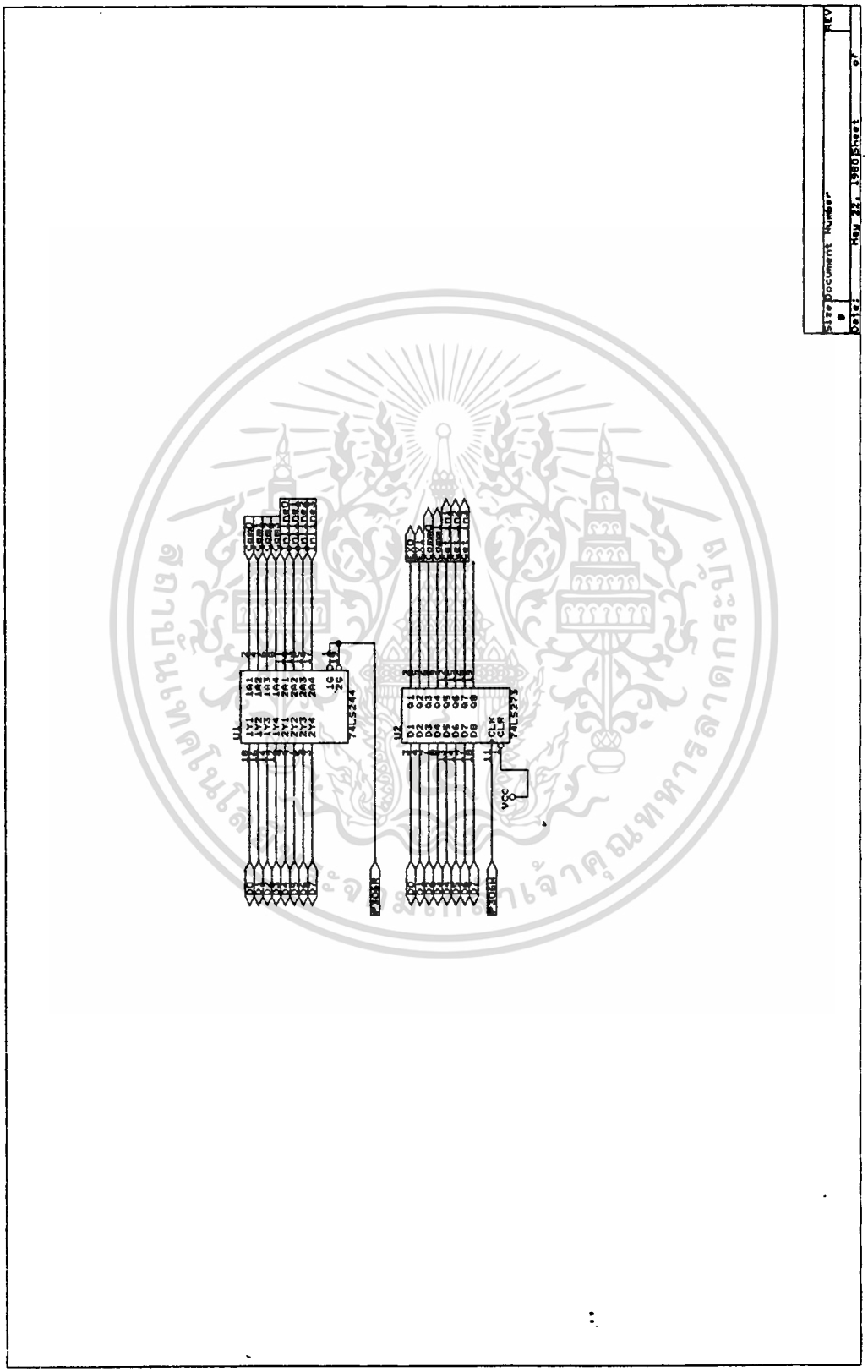
305H ใช้เคลียร์ฟลิปฟลอปที่ถูกทริกจากสัญญานเรียกและสัญญาน DTMF ในแชนแนลที่ 2



วงจรที่ 3.12 วงจรอินเทอร์เฟสกับไมโครคอมพิวเตอร์



วงจรที่ 3.13 วงจรของพอร์ทเขียน 301H และพอร์ทอ่าน 302H
เพื่อควบคุมการบันทึกและเล่นกลับและอ่านภาวะของระบบ



รูปที่ 3.14 วงจรอินเทอร์เฟสกับชุมสายโทรศัพท์

306H เขียน**พอร์ทรับคำสั่ง**ของชุมสาย (ดูรายละเอียดในหัวข้อ 3.2.5, 3.3.5 และ 4.2.4)

307H เขียนรหัสเลขหมายในกระบวนการโทรออก (ดูตารางที่ 3.3)

3.3.5 **ส่วนอินเทอร์เฟสกับชุมสายโทรศัพท์**

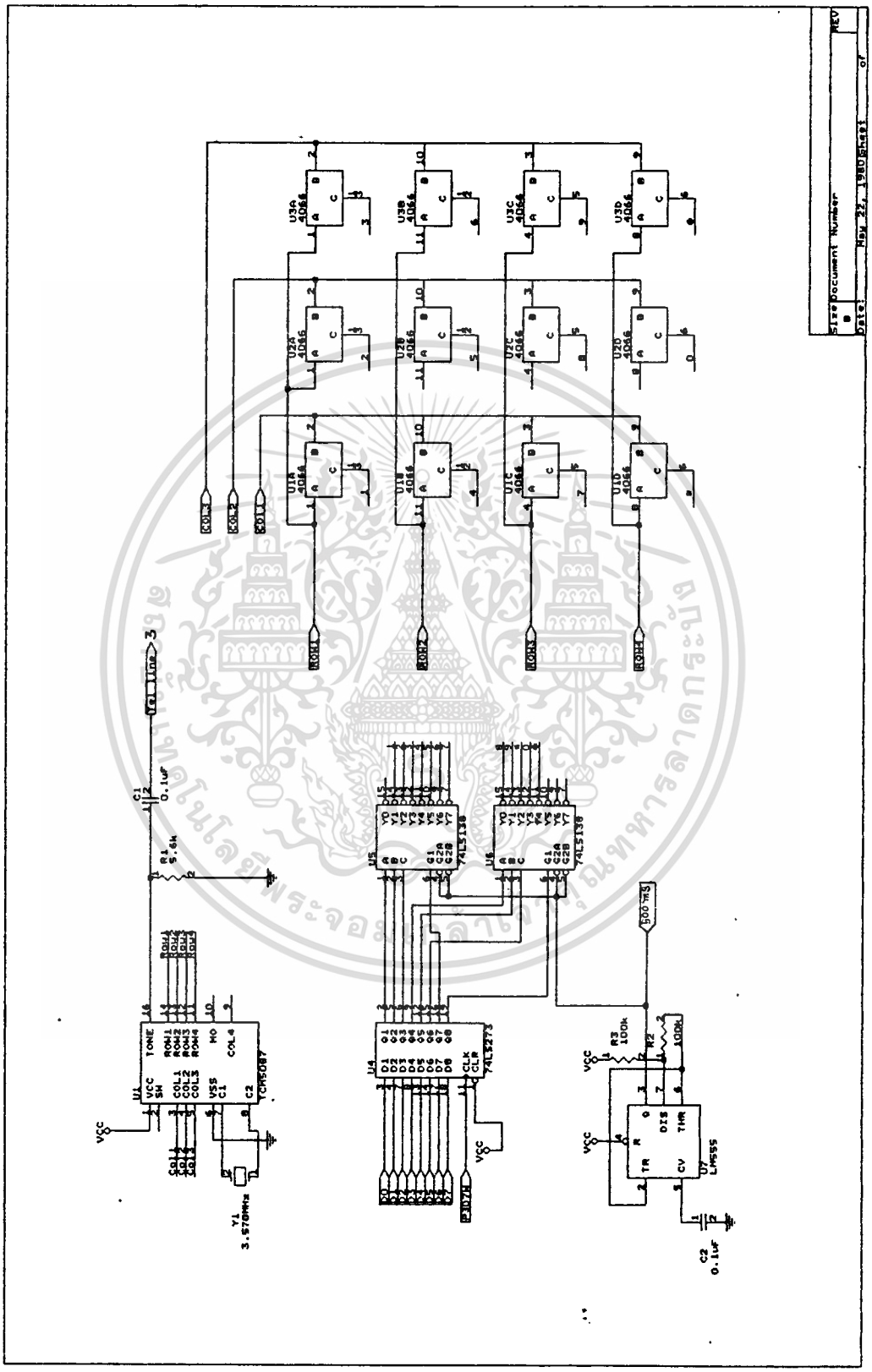
ส่วนนี้ประกอบด้วยพอร์ทสองพอร์ท ทำหน้าที่อ่านและส่งคำสั่งให้กับชุมสายโทรศัพท์ในกรณีที่ใช้ระบบตอบรับโทรศัพท์ร่วมกับชุมสายที่พัฒนาขึ้น โดยพอร์ทรับคำสั่งจะต่ออยู่กับพอร์ทส่งคำสั่งของชุมสาย และพอร์ทส่งคำสั่งจะต่ออยู่กับพอร์ทรับคำสั่งของชุมสาย ดังวงจรที่ 3.14

3.3.6 **ส่วนจัดการเพื่อการโทรออก**

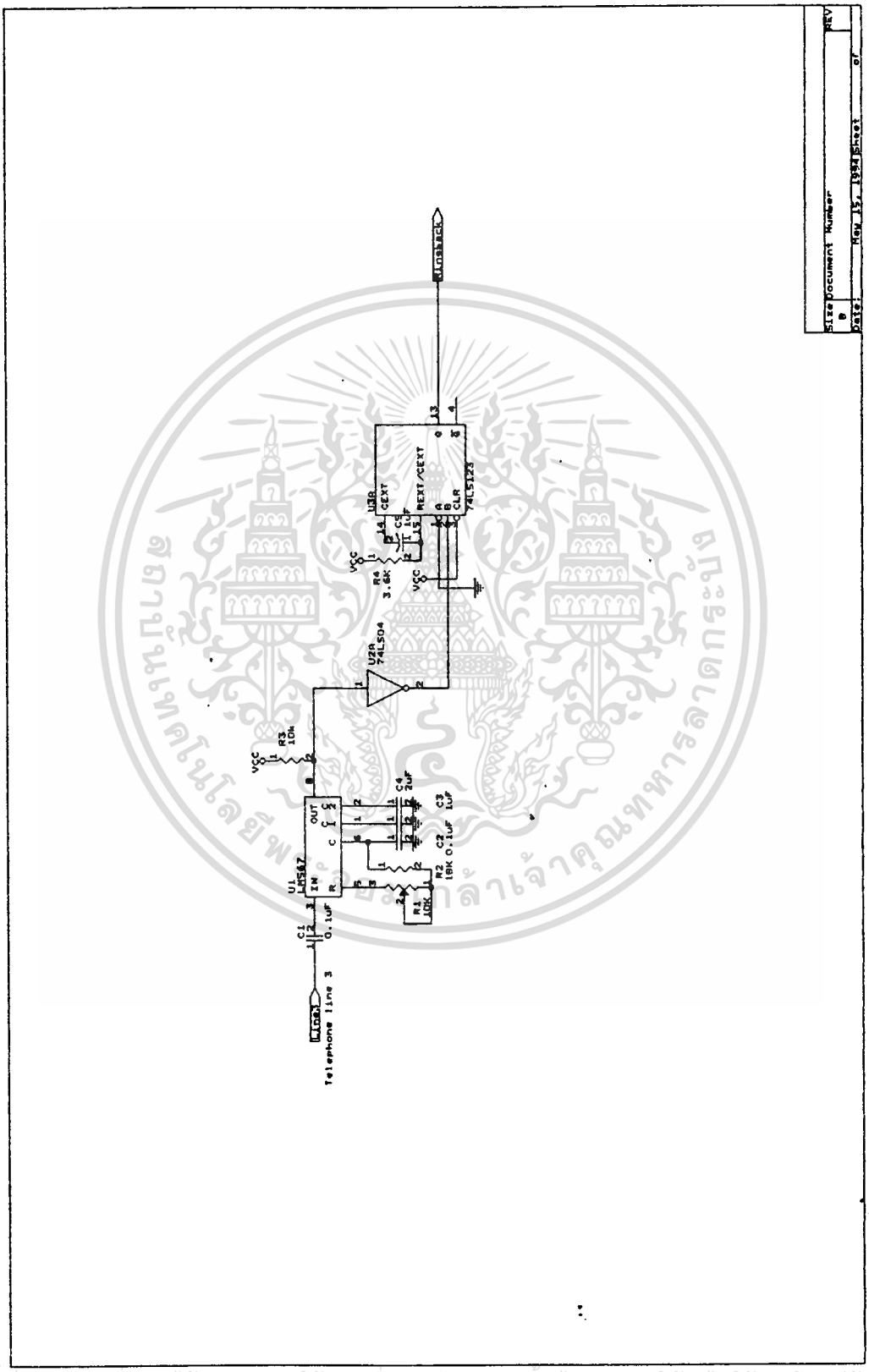
วงจรส่วนนี้ประกอบไปด้วยวงจรส่งสัญญาณ DTMF และวงจรตรวจสัญญานเรียกกลับ ดังวงจรที่ 3.15 และ 3.16 ตามลำดับ

วงจรที่ 3.15 เป็นวงจรส่งสัญญาน DTMF ซึ่งใช้ไอซีเบอร์ TCM5087 แผงคีย์บอร์ดประกอบด้วยไอซี CMOS 4066 ซึ่งจะถูกควบคุมการเปิดปิดโดยดีโคดเดอร์ 74LS138 สองตัว ซึ่งข้อมูลที่นำมาดีโคดจะได้จากพีซีซึ่งถูกแลทช์ไว้ใน 74LS273 การส่งสัญญาน DTMF ออกไปจะถูกควบคุมโดยพัลซ์ที่มีช่วงเปิดปิดประมาณ 500 ms ซึ่งจะทำหน้าที่อีเนเบิ้ล 74LS138 ทั้งสองตัวที่ระดับลอจิก 0 ซึ่งสัญญาน DTMF จะถูกส่งออกไปยังสายโทรศัพท์เมื่อพัลซ์นี้มีลอจิกเป็น 0 และข้อมูลซึ่งเป็นรหัสแทนสัญญาน DTMF จะถูกเขียนลงบน 74LS273 เมื่อระดับลอจิกของพัลซ์นี้เป็น 1 รหัสที่ต้องเขียนลงบน 74LS273 สำหรับหมายเลขต่างๆเป็นดังตารางต่อไปนี้ ตารางที่ 3.3 แสดงรหัสที่ต้องเขียนลงบน 74LS273 สำหรับรหัส DTMF ต่างๆ

รหัส	เลขหมาย
41	1
42	2
43	3
44	4
45	5
46	6
47	7
80	8
88	9
90	*
98	0
A0	#



วงจรที่ 3.15 วงจรจัดการเพื่อการโทรออก



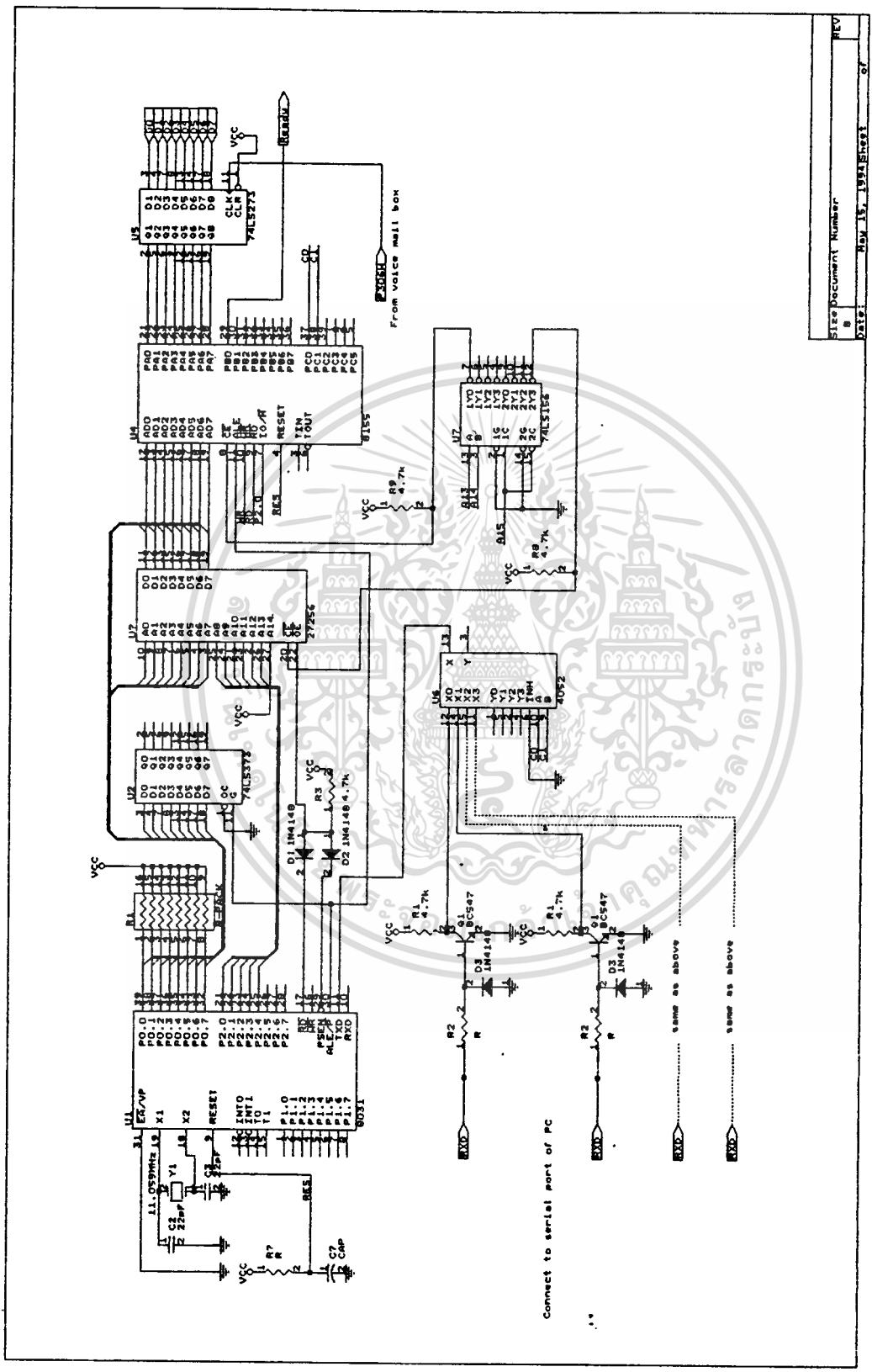
วงจรที่ 3.16 วงจรตรวจสัญญาณเรียกเข้า

หลังจากที่เราทำการส่งรหัส DTMF ออกไปแล้ว สัญญาณที่จะส่งกลับออกมาจากชุมสายโทรศัพท์จะเป็นสัญญาณเรียกกลับหรือสัญญาณสายไม่ว่างเท่านั้น (ในกรณีที่ชุมสายทำงานตามปกติ) พิจารณาจากวงจรรูปที่ 3.16 ไอซีเฟสล็อกลูป LM567 ซึ่งถูกตั้งไว้ให้ดีเทคความถี่ประมาณ 400Hz เมื่อนำเอาท์พุทมาผ่านอินเวอร์เตอร์และโมโนสเตเบิ้ลจะให้สัญญาณที่เป็น 1 และ 0 ตามจังหวะของสัญญาณเรียกกลับ ซึ่งสัญญาณนี้จะถูกนำไปตรวจสอบจังหวะโดยซอฟต์แวร์เพื่อพิจารณาว่าเป็นสัญญาณเรียกกลับหรือไม่

3.4 **ฮาร์ดแวร์ส่วนบริการข่าวสารจากส่วนกลาง**

ฮาร์ดแวร์ในส่วนนี้มีหลักการง่ายๆคือรับข้อมูลขนาด 256 ไบท์จากพีซีแล้วส่งออกแบบอนุกรมไปยังพอร์ทอนุกรมของไมโครคอมพิวเตอร์ 4 เครื่องโดยใช้ไอซี CMOS 4052 ซึ่งเป็นตัวมัลติเพล็กซ์/ดีมัลติเพล็กซ์ วงจรหลักได้แก่ 8031, 8155 ดังวงจรรูปที่ 3.17

ข้อมูลที่พีซีส่งมาให้จะถูกเก็บอยู่ในหน่วยความจำของพอร์ท 8251 ซึ่งมีขนาด 256 ไบท์ และในวงจรถูกดีโคดให้เริ่มที่แอดเดรส E000H การส่งข้อมูลจากเครื่องพีซีมาเก็บไว้ในหน่วยความจำนี้จะกระทำผ่านพอร์ทเขียนหมายเลข 306H ของระบบตอบรับโทรศัพท์ โดยพอร์ทนี้จะเชื่อมต่ออยู่กับอินพุทพอร์ท A ของ 8155 ซึ่งอยู่ที่ตำแหน่ง E101H บอร์ด 8031 นี้จะใช้บิทที่ 0 ของเอาท์พุทพอร์ท B ซึ่งอยู่ที่ตำแหน่ง E102H ในการกำหนดจังหวะในการรับส่งข้อมูล โดย 1 หมายความว่า 8031 พร้อมที่จะรับข้อมูลจากพีซี โดยบิทนี้จะต่ออยู่กับบิทที่ 7 ของพอร์ทอ่านหมายเลข 306H ของระบบตอบรับ การสแกน 4052 ในขณะที่มีการส่งข้อมูลจากหน่วยความจำจะใช้ 2 บิทของเอาท์พุทพอร์ท C ของ 8155 ซึ่งอยู่ที่ตำแหน่ง E103H รายละเอียดในการรับส่งข้อมูลจะอยู่ในหัวข้อซอฟต์แวร์ที่ 4.3



วงจรที่ 3.17 วงจรระบบแจ้งข่าวสาร

บทที่ 4
ซอฟต์แวร์ของระบบ

ซอฟต์แวร์เป็นส่วนสำคัญของระบบ ในงานวิจัยนี้ซอฟต์แวร์ทั้งหมดเป็นภาษาแอสเซมบลี้ โดยเป็นภาษาของ Z-80 สำหรับชุมสายโทรศัพท์ และ 8088 สำหรับระบบตอบรับโทรศัพท์ และ MCS-51 สำหรับระบบแจ้งข่าวสารจากส่วนกลาง เนื้อหาจะเน้นที่โครงสร้างข้อมูลและอัลกอริธึมธ์ เพื่อให้ง่ายต่อการทำความเข้าใจและปรับปรุงการทำงานในภายหลัง

4.1 ซอฟต์แวร์ของชุมสายโทรศัพท์

ซอฟต์แวร์นี้เป็นภาษาแอสเซมบลี้ Z-80 โครงสร้างหลักเป็นแบบวนลูปเพื่อทำการโพลสถานะของสายต่างๆ และทำงานให้กับสายนั้นๆ อัลกอริธึมธ์หลักเป็นดังนี้

```
loopline:   Select pair of line to decode DTMF code;
            Update count;
            Time out for external ringing;
            Delay
            Check hanging up or down of each line;
            Check if external line comes;
            Check if avm send a command;
            Decode DTMF code;
            Jmp    loopline
```

ลูปเริ่มโดยการเลือกสายใน 1 คู่เพื่อทำการถอดรหัส DTMF จากสายคู่นั้น (ถ้ามี) โดยการถอดรหัสจะเริ่มกระทำที่ปลายลูปเนื่องจากวงจรถอดรหัสต้องการให้สัญญาน DTMF คงที่เป็นระยะเวลาหนึ่งก่อนที่จะถอดรหัสได้ จึงต้องมีการหน่วงเวลาของลูปหลังจากเลือกคู่สายที่จะทำการถอดรหัสด้วย

ส่วนที่เหลือของลูปจะตรวจสอบการยกหูหรือวางหูของสายภายใน ตรวจสอบสัญญานเรียกจากสายนอก และตรวจสอบคำสั่งจากระบบตอบรับโทรศัพท์ในกรณีที่ใช้งานร่วมกัน ซึ่งรายละเอียดของซอฟต์แวร์ในส่วนต่างๆเป็นดังนี้

4.1.1 การติดตามสถานะของแต่ละสายใน

เพื่อให้สามารถติดตามการทำงานของโทรศัพท์แต่ละเครื่องได้ จึงกำหนดให้มีตัวแปรสถานะของโทรศัพท์แต่ละเครื่อง โดยตัวแปรมีขนาดมีขนาด 1 ไบท์ต่อโทรศัพท์สายใน 1 เครื่อง แต่ละไบท์จะบรรจุรหัสสถานะของแต่ละสายไว้ โดยสถานะต่างๆที่มีได้เป็นดังนี้

downstate เป็นสถานะเริ่มต้น สายในใดที่วางหูลง หรือ ไม่ได้ยกหูโทรศัพท์จะถูกกำหนดให้มีสถานะนี้

dialstate	เป็นสถานะของสายในเมื่อได้ทำการยกหูเพื่อที่จะทำการโทร โดยสายนั้นจะได้รับสัญญานให้หมุน
off_dialstate	เป็นสถานะที่ต่อจากสถานะข้างต้น โดยถ้าสายในนั้นทำการกดปุ่มแรก สัญญานให้หมุนจะหายไป
calledstate	เป็นสถานะที่กำหนดให้กับสายที่ถูกเรียก โดยที่สายนั้นอยู่ในสถานะ downstate
callerstate	เป็นสถานะที่กำหนดให้กับสายในที่ทำการเรียกสายอื่น โดยที่สายที่เรียกไปนั้นอยู่ในสถานะ downstate มาก่อน
conver_caller	เป็นสถานะที่กำหนดให้ขณะทำการสนทนาโดยเป็นฝ่ายโทรหาคู่สนทนา
conver_called	เป็นสถานะที่กำหนดให้ขณะทำการสนทนาโดยเป็นฝ่ายถูกโทรเรียก
holdstate	เป็นสถานะที่ถูกกำหนดโดยการกดปุ่มพักสายนอก (#)
conver_ex	เป็นสถานะที่ถูกกำหนดในขณะที่สนทนากับสายนอก
use_avmstate	เป็นสถานะที่ถูกกำหนดในขณะใช้บริการระบบตอบรับโทรศัพท์
ex_calledstate	เป็นสถานะของสายในที่ 1 เมื่อถูกเรียกจากสายนอกใดสายหนึ่งใน 3 สาย

4.1.2 **การตรวจสอบการยกหูและวางหูของสายภายใน**

ส่วนที่เกี่ยวข้อง Check hanging up or down of each line;

การตรวจสอบทำได้โดยการพิจารณาการเปลี่ยนแปลงของตัวแปรสองตัวคือ **prestate** และ **nowstate** ซึ่งเป็นตัวแปรขนาด 1 ไบท์ ซึ่งแต่ละบิทจะเกี่ยวข้องกับสายในแต่ละสาย

ตัวแปร prestate ใช้บอกสถานะการยกหูหรือวางหูของแต่ละสายก่อนที่จะมีการตรวจสอบ โดย 1 หมายถึงวางหู และ 0 หมายถึงยกหู ตัวแปร nowstate ใช้บอกสถานะการยกหูหรือวางหูในขณะที่มีการตรวจสอบ การแยกแยะว่าสายในใดทำการยกหูหรือวางหูกระทำโดยสองกระบวนการ หนึ่งจะต้องตรวจสอบดูก่อนว่ามีสายใดมีการยกหูหรือวางหูหรือไม่ซึ่งทำได้โดยนำ prestate มา XOR กับ nowstate ผลที่ได้จะนำมาเก็บไว้ในตัวแปร changestate โดยถ้าผลของการ XOR ปรากฏว่าบิทของสายใดเป็น 1 แสดงว่าสายนั้นมีการยกหูหรือวางหู แต่ถ้าเป็น 0 แสดงว่าไม่มีการเปลี่ยนแปลง สองนำค่าใน prestate มาเปรียบเทียบกับ nowstate เฉพาะสายที่มีบิทใน change state เป็น 1 เพื่อหาว่าการเปลี่ยนแปลงนั้นเป็นการยกหูหรือวางหู ถ้าเป็นการยกหู ค่าใน nowstate จะเป็น 0 ในขณะที่ค่าใน prestate เป็น 1 และถ้าเป็นการวางหู ค่าใน nowstate จะเป็น 1 ในขณะที่ค่า prestate เป็น 0

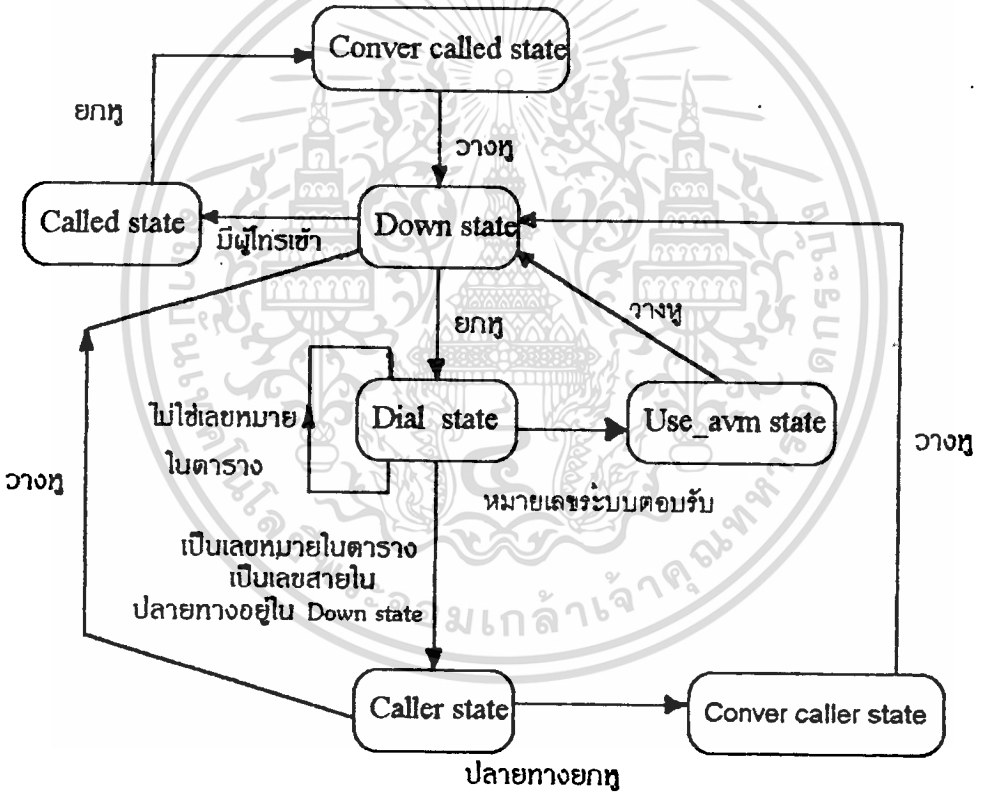
1	0	1	1	1	0	0	0	**prestate**
								XOR
0	1	0	1	1	1	0	1	**nowstate**
1	1	1	0	0	1	0	1	**changestate**

จากรูปจะสรุปได้ว่าสายที่วางหูคือสายที่ 1, 3 และ 7 สายที่ยกหูคือสายที่ 6 และ 8 สายที่ไม่มีการเปลี่ยนแปลงคือสายที่ 2, 4, 5 (บิทขวาสุดคือสายที่ 1)

หลังจากตรวจสอบได้ว่าสายใดทำการยกหูหรือวางหูแล้วก็จะนำไปพิจารณาต่อไปว่าเป็นการยกหูหรือวางหูในกรณีใด การยกหูนั้นอาจมาจากการรับสัญญาณเรียกหรือต้องการจะโทรออก เช่นเดียวกับการวางหูซึ่งอาจจะมาจากการจบการสนทนาหรือยกเลิกการโทรออก การทำงานในส่วนนี้เกี่ยวข้องกับการเปลี่ยนสถานะของโทรศัพท์แต่ละเครื่องซึ่งจะกล่าวในหัวข้อถัดไป

4.1.3 **<u>การเปลี่ยนสถานะของสายในเมื่อมีการโทรติดต่อกัน</u>**

การโทรติดต่อกันภายในเป็นการใช้งานขั้นพื้นฐานของชุมสายโทรศัพท์โดยทั่วไป ในขณะที่มีการใช้เริ่มตั้งแต่มีการยกหูเพื่อทำการโทรออก จนถึงการสนทนาแล้ววางหู ซอฟต์แวร์จะทำการกำหนดสถานะของแต่ละสายให้ซึ่งเป็นไปดังไดอะแกรมต่อไปนี้



รูปที่ 4.1 ไดอะแกรมสถานะของซอฟต์แวร์

เมื่อสายใดวางหูอยู่สายนั้นจะอยู่ในสภาวะ down state และเมื่อผู้ใช้ทำการยกหูเพื่อจะโทรติดต่อสายนั้นจะถูกเปลี่ยนให้อยู่ในสถานะ dial state ซึ่งเป็นสถานะที่รอการกดปุ่มเลขหมายของผู้ใช้ หากเลขหมายที่กดไม่มีอยู่ในตารางเลขหมาย (รายละเอียดอยู่ในหัวข้อ 4.1.5) สายนั้นก็ยังคงอยู่ในสถานะเดิม หากพบว่าเป็นเลขหมายภายใน (21 ถึง 28) และสถานะของสายของเลขหมายนั้นเป็น down state ก็จะเปลี่ยนสถานะเป็น caller state ในขณะเดียวกันสถานะของเลขหมายที่ถูกเรียกก็จะถูกเปลี่ยนเป็น called state หากผู้ถูกเรียกทำการยกหู ผู้เรียกก็จะเปลี่ยนจาก caller state เป็น conver caller state และ ผู้ถูกเรียกก็จะเปลี่ยนสถานะเป็น conver called

state ซึ่ง ณ ขณะนี้เป็นเวลาที่ทั้งคู่ทำการสนทนากันอยู่ และเมื่อทำการวางหูก็จะกลับเข้าสู่สถานะ down state อีกครั้งหนึ่ง

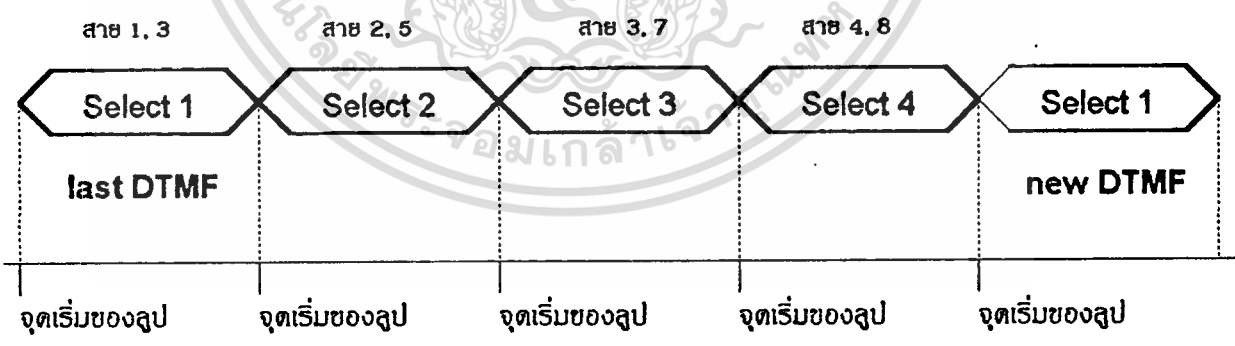
ในขณะที่อยู่ในสถานะ dial state หากเลขหมายปลายทางไม่ว่าง ผู้เรียกจะได้รับสัญญานสายไม่ว่าง และในขณะที่คู่สนทนาอยู่ในสถานะ conver caller state หรือ conver called state หากฝ่ายใดฝ่ายหนึ่งวางหูคู่สนทนาก็จะได้รับสัญญานว่าคู่สนทนาได้วางหูแล้ว (ซึ่งเป็นสัญญานเดียวกับสัณณานสายไม่ว่าง)

หากหมายเลขที่กดในขณะที่อยู่ใน dial state เป็น 51 ถึง 55 ซึ่งเป็นเลขหมายของระบบตอบรับโทรศัพท์และสามารถติดต่อกับระบบตอบรับได้ สายนั้นก็จะถูกเปลี่ยนสถานะเป็น use_avm state ซึ่งหมายความว่าขณะนี้กำลังใช้บริการของระบบตอบรับอยู่ และเมื่อทำการวางหูสายนั้นก็จะกลับไปเป็น down state พร้อมๆ กับส่งคำสั่งว่าได้วางหูจากการติดต่อแล้วไปยังระบบตอบรับด้วย

4.1.4 **การเลือกคู่สายในเพื่อทำการถอดรหัส DTMF**

ส่วนที่เกี่ยวข้อง : Select pair of line to decode DTMF code;
Update count;
Delay;
Decode DTMF;

ดังได้กล่าวในหัวข้อ 3.2.4 แล้วว่าวงจรในส่วนนี้ใช้วงจรถอดรหัส DTMF หนึ่งวงจรต่อสายใน 4 สาย โดยการเลือกหนึ่งครั้งจะได้สายที่จะถอดรหัสออกมา 2 สาย เพื่อส่งให้กับวงจรถอดรหัส 2 วงจรเพื่อทำการถอดรหัสพร้อมกัน การเลือกจะกระทำที่ต้นลูปและการอ่านค่าที่ได้จากการถอดรหัสจะกระทำที่ปลายลูป ไดอะแกรมเวลาในการเลือกสายในเพื่อการถอดรหัสเป็นดังรูปที่ 4.2



รูปที่ 4.2 ไดอะแกรมเวลาของการเลือกคู่สายเพื่อทำการถอดรหัส DTMF

ปัญหาที่อาจเกิดขึ้นจากการออกแบบในลักษณะนี้คือเมื่อผู้ใช้กดปุ่มเลขหมายค้างไว้จะทำให้ซอฟต์แวร์อ่านเลขหมายได้หลายตัวทั้งๆที่เป็นการกดเลขตัวเดียว ซึ่งแก้ไขได้โดยกำหนดว่าถ้าอ่านได้เลขหมายเดิมในสองครั้งที่ติดกันให้ถือว่าเป็นการกดค้าง แต่จะทำให้เกิดข้อจำกัดในกรณีที่ผู้ใช้กดเลขหมายเดิมสองครั้งติดกันแล้วไปตรงกับจุดเริ่มของลูปที่ทำการเลือกสายนั้นพอดีก็จะทำให้อ่านได้เลขหมายเดียวทั้งๆที่กดสองครั้ง

การแยกแยะว่า new DTMF เกิดขึ้นหลังจาก last DTMF ทันทีหรือไม่ทำได้โดยกำหนดให้ทุกๆครั้งที่อ่านรหัส DTMF ได้ให้เริ่มนับตัวแปร count ของสายนั้น (ซึ่งถูกเซตไว้เป็น 0 ในตอนเริ่มต้น) โดยในรูทีน Update count จะเพิ่มค่า count ขึ้น 1 ทุกครั้งในหนึ่งรอบของลูปหลักเมื่อค่าของ count ไม่เท่ากับ 0 จากรูปที่จะสังเกตเห็นได้ว่า ถ้า new DTMF เกิดหลังจาก last DTMF แล้วค่า count จะเท่ากับ 5

อัลกอริธึมธ์เพื่อแก้ปัญหาจากการกดค้างเป็นดังนี้ (เป็นส่วนที่อยู่ใน Decode DTMF)

```
If (STD=1) {          rem :  ถ้า STD =1 หมายความว่าวงจรถอดรหัสถอดรหัส DTMF ได้
        If (count <> 0)
                If (new DTMF = last DTMF)
                        If (count = 5) { count = 1; exit; }
        Keep new DTMF;
        count = 1;
}
```

และอัลกอริธึมธ์ใน Update count เป็นดังนี้

```
If count <> 0 Then          rem : ตัวแปร count เป็นอะเรย์ขนาด 8 ไบท์, 1 ไบท์ต่อ 1 สายใน
{       Increment count;    rem : อัลกอริธึมธ์นี้จะกระทำต่อทุกตัวแปร count
        If count = 6 Then   rem : count = 6 หมายความว่าไม่มีการกดค้างหรือกดปุ่มติดกัน
                count = 0;  }
```

4.1.5 **<u>การนำเลขหมายที่อ่านได้ไปเทียบกับตารางเลขหมาย</u>**

<u>ส่วนที่เกี่ยวข้อง</u> Decode DTMF code;

รหัส DTMF ที่ได้จะถูกนำมาเปรียบเทียบกับตารางเลขหมายทันที หากพบว่าเป็นเลขหมายในตารางก็จะปฏิบัติงานนั้นทันที ตารางเลขหมายเป็นตารางขนาด 20 แถว 8 หลัก หลักละ 1 ไบท์ ในหนึ่งหลักจะบรรจุรหัส DTMF 1 ค่า ตารางทั้งหมดบรรจุอยู่ใน EPROM รวมอยู่กับโปรแกรม โดยแต่ละแถวจะหมายถึงเลขหมาย 1 ชุด เลขหมายภายในที่กำหนดไว้คือ 21 ถึง 28 เลขหมายเพื่อตัดสายนอกทั้งสามสาย 91 ถึง 93 เลขหมายเพื่อการติดต่อกับระบบตอบรับโทรศัพท์ 51 ถึง 55 และปุ่มเพื่อการพักสายนอกคือ #

4.1.6 **<u>การติดต่อกับสายนอก</u>**

<u>ส่วนที่เกี่ยวข้อง</u> : Time out for external ringing;
Check if external line comes;
Decode DTMF code,

การควบคุมการติดต่อกับสายนอกของซอฟต์แวร์ประกอบไปด้วยสามส่วนได้แก่ การตรวจสอบสัญญาณเรียกและการรับสายนอก การพักสายและการโอนสาย และการตัดสายนอก

1 การตรวจสอบสัญญานเรียกและการรับสายนอก

ดังได้กล่าวไว้ในหัวข้อ 3.2.1 แล้วว่าเมื่อเกิดสัญญานเรียกวงจรในส่วนนี้จะกำเนิดพัลซ์ที่มีความกว้างประมาณ 1 วินาทีตามจังหวะของสัญญานเรียก พัลซ์นี้จะถูกโพลโดยซอฟต์แวร์ในส่วน Check if external line comes เพื่อดูว่ามีสายนอกเข้ามาหรือไม่ หากพบก็จะส่งสัญญานเรียกไปยังสายที่ 1 ซึ่งถูกกำหนดให้เป็นโทรศัพท์หลัก ดังอัลกอริธึมต์ต่อไปนี้

```
If ( external ringing on line 3 found ) {
        exuse [3] = 2;          ; REM · กรณีมีสัญญานเรียกจากสายนอกที่ 3
        time_out [3] = 0;
        If ( line 1 in Down state ) {
                ringing line 1;
                make line 1 to be ex_call state;        }
        else    {       make line 1 to be ex_call state;        }
}
```

ตัวแปร time_out เป็นอะเรย์ขนาด 3 ไบท์ซึ่งใช้จับเวลาประมาณ 5 วินาทีของสายนอกแต่ละสายเพื่อใช้ในการเคลียร์ exuse ของสายนั้น รายละเอียดจะได้กล่าวต่อไป

ตัวแปร exuse เป็นอะเรย์ขนาด 3 ไบท์ แต่ละไบท์จะบอกสถานะของสายนอกทั้งสาม โดยมีความหมายดังต่อไปนี้

exuse = 00 สายนอกนั้นว่างไม่มีสายนอกเข้าและไม่มีการใช้โดยสายใน

exuse = 01 สายนอกนั้นถูกจองโดยสายในในขณะตัดสายนอกเพื่อโทรออก

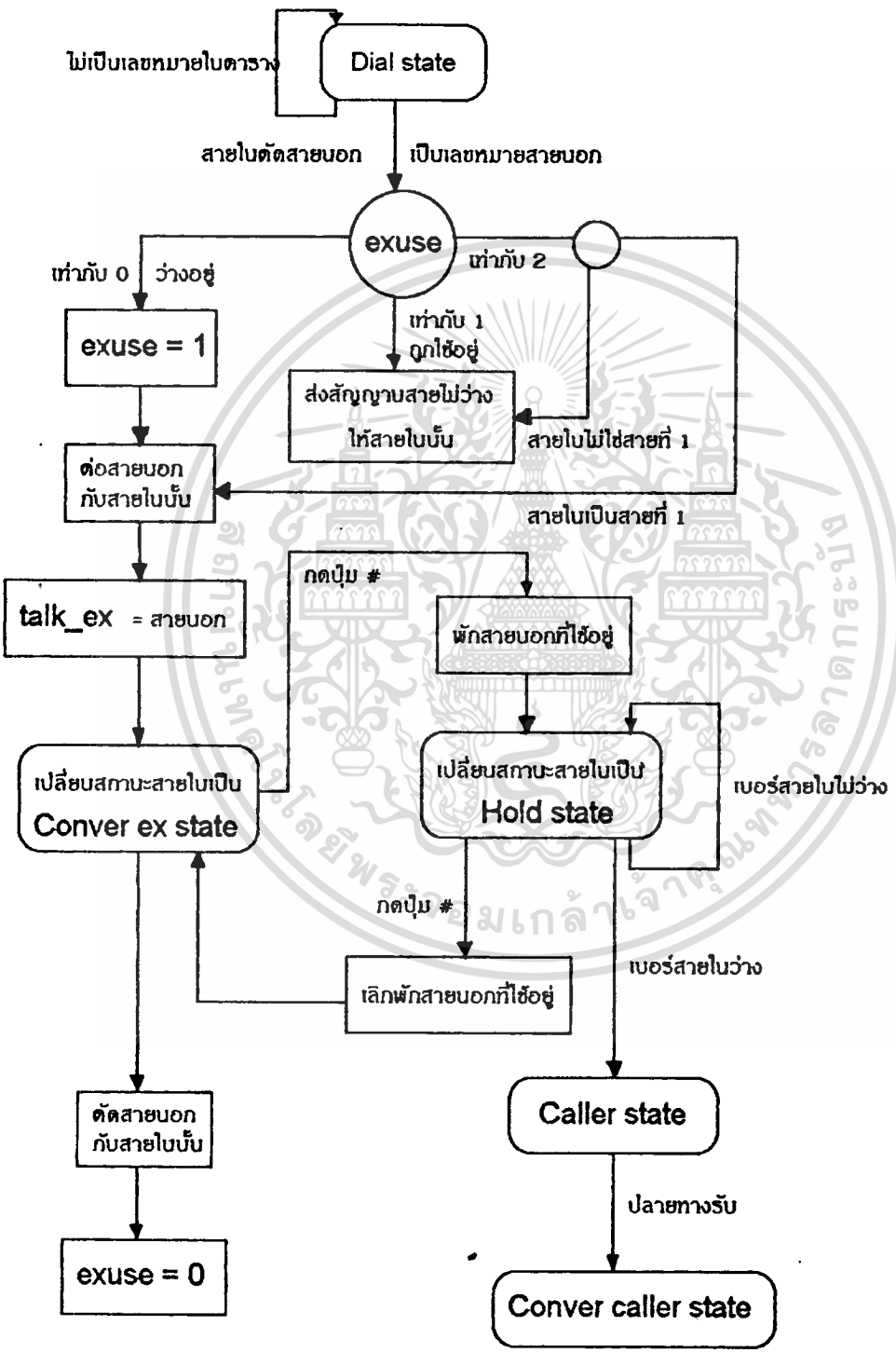
exuse = 02 มีสัญญานเรียกเข้ามาที่สายนอกนั้น

เมื่อมีสัญญาณเรียก exuse ของสายนอกที่ถูกเรียกนั้นจะถูกเปลี่ยนเป็น 2 และการรับสายนอกจะกระทำโดยสายที่ 1 ซึ่งหากมีสัญญานเรียกเข้ามาและสาย 1 อยู่ใน down state อยู่ก่อนแล้วก็จะถูกเปลี่ยนเป็น ex call state ซึ่งหมายถึงสถานะที่ถูกเรียกโดยสายนอก สายที่ 1 เมื่อจะทำการรับสายนอกจะต้องกดหมายเลขตัดสายนอก (91 ถึง 93) ของสายนอกที่เข้ามานั้น (บนกล่องชุมสายจะมี LED 3 ดวงซึ่งกระพริบตามสายนอกที่เข้ามา) เมื่อสาย 1 รับสายนอกนั้นแล้วตัวแปร exuse ของสายนั้นจะเป็น 1 ซึ่งหมายความว่าสายนอกนั้นถูกใช้อยู่ และสาย 1 นั้นก็จะถูกเปลี่ยนสถานะเป็น conver ex state ซึ่งหมายความว่ากำลังสนทนากับสายนอกอยู่

ถ้าตัวแปร exuse มีค่าเท่ากับ 2 ซึ่งเป็นกรณีที่มีสายนอกเข้ามา ก่อนที่จะเกิดพัลซ์จากสัญญานเรียกลูกถัดไป ตัวแปรนี้จะได้รับการเคลียร์ให้เป็น 0 ก่อน มิฉะนั้นจะเกิดกรณีที่สัญญานเรียกหยุดไปแล้วแต่ exuse ยังคงเป็น 2 อยู่ซึ่งทำให้ตัดสายนอกนั้นไม่ได้ การเคลียร์นี้อยู่ในส่วน Time out for external ringing อัลกอริธึมต์ในส่วนนี้เป็นดังนี้

```
If ( exuse = 2 ) Then
{       Increment time_out [3] by 1;
        If ( time_out means to 5 min. ) exuse = 0;
```

กระบวนการที่เกี่ยวข้องกับสายนอกเป็นดังได้อะแกรมต่อไปนี้



รูปที่ 4 3 แสดงไดอะแกรมของกระบวนการที่เกี่ยวกับสายนอก

ตัวแปร talk_ex เป็นอะเรย์ขนาด 8 ไบท์ 1 ไบท์ต่อ 1 สายในใช้เพื่อบอกว่าสายในแต่ละสายกำลังติดต่อกับสายนอกใดอยู่ โดยค่าในแต่ละไบท์จะเป็น 0, 1 และ 2 ซึ่งหมายถึงสายนอกทั้งสามสาย และ ค่า 4 หมายถึงไม่มีการติดต่อกับสายนอกใดเลย

2. การพักสายและการโอนสาย

เมื่อสายที่ 1 รับสายนอกที่เข้ามาแล้ว เมื่อต้องการจะโอนสายก็เริ่มด้วยการกดปุ่มพักสาย # ซึ่งจะทำให้วงจรพักสายทำงานและสายหนึ่ง 1 จะถูกเปลี่ยนสถานะเป็น hold state ซึ่งมีภาวะคล้ายๆกับ dial state เมื่อสาย 1 โทรติดต่อกับสายที่จะโอนไปได้ ตัวเองก็จะถูกเปลี่ยนสถานะเป็น caller state และเมื่อปลายทางรับก็จะเปลี่ยนสถานะเป็น conver caller state เมื่อจะโอนสายที่กำลังสนทนาด้วยให้กับสายนอกที่พักอยู่ก็วางหูลง

การแยกแยะว่าเป็นการวางหูเพื่อการโอนสายหรือเป็นการวางหูจากกรณีอื่นๆทำได้โดยการพิจารณาจากค่าในตัวแปร talk_ex หากมีค่าเท่ากับ 4 หมายความว่าไม่ใช่เป็นการวางหูเพื่อการโอนสาย

3. การตัดสายนอก

การกดหมายเลขเพื่อตัดสายนอก (91 ถึง 93) ได้หรือไม่ได้พิจารณาจากค่าของ exuse หากมีค่าเป็น 0 หมายความว่าสายนอกนั้นว่าง ซึ่งต่อมาจะถูกเปลี่ยนเป็น 1 ซึ่งเป็นการจองสายนอกนั้นไว้ กระบวนการถัดไปก็เช่นเดียวกับการรับสายนอกของสายที่ 1 เพราะสายในนั้นได้ต่อตรงเข้ากับสายนอกนั้นแล้ว (ดูไดอะแกรมรูปที่ 4.3 ประกอบ)

4.2 ซอฟต์แวร์ของระบบตอบรับโทรศัพท์

4.2.1 หลักการโดยทั่วไป

ซอฟต์แวร์ในส่วนนี้เป็นส่วนที่อยู่บนเครื่องพีซี เขียนด้วยภาษาแอสเซมบลี้ 8088 เนื่องจากเป็นงานที่ต้องการความรวดเร็วเพราะเกี่ยวข้องกับการแซมปลิ้งสัญญาน ซึ่งซอฟต์แวร์ส่วนนี้สามารถนำไปดัดแปลงเพื่อควบคุมให้ระบบตอบรับทำงานตามวัตถุประสงค์อื่นได้อีก ในหัวข้อนี้เป็นส่วนสำคัญที่นำเสนอ โดยจะแสดงโครงสร้างข้อมูลและอัลกอริธึมต่างๆที่จำเป็นในระบบตอบรับโทรศัพท์ซึ่งสามารถนำไปเรียบเรียงใหม่เพื่อให้ระบบตอบรับทำงานได้หลากหลายมากขึ้น

เนื่องจากการบันทึกและเล่นกลับเสียงเป็นกระบวนการที่ต้องการให้ CPU จัดการให้ด้วยเวลาที่แน่นอน คือประมาณทุกๆ 500 us ดังนั้นส่วนหลักของโปรแกรมจึงเป็นโปรแกรมลูปที่เริ่มต้นด้วยการตรวจสอบขอบขาขึ้นของสัญญานนาฬิกาหลัก (มีความถี่ 2KHz ดูรายละเอียดในหัวข้อ 3.3.1) เมื่อพบก็จะทำการอ่านและเขียนให้กับแชนแนลทั้งสาม แต่เนื่องจากคำสั่งที่ใช้ในกระบวนการดังกล่าวมีเพียงไม่กี่คำสั่งซึ่งแต่ละคำสั่งใช้เวลาไม่กี่ us จากที่มีอยู่ประมาณ 500 us ก่อนที่จะมีการแซมปลิ้งครั้งถัดไป ดังนั้นเราสามารถนำเวลาที่เหลือนี้มาให้ CPU ทำงานอย่างอื่นเช่น อ่านรหัส DTMF ตรวจสอบสัญญานเรียก อ่านคำสั่งจากชุมสายโทรศัพท์ ตรวจสอบเวลาเพื่อการโทรออก และโทรออกอัตโนมัติ และอื่นๆ การทำงานให้กับงานต่างๆเหล่านี้จะต้องเสมือนกับว่าเกิดขึ้นพร้อมกันในทัศนะของผู้ใช้ กล่าวคือไม่ควรมีการหยุดรอของงานใดงานหนึ่งในขณะที่เกิดอีกงานหนึ่ง ดังนั้นจึงต้องมีการวางอัลกอริธึมธ์ที่เหมาะสม

ดังที่ได้กล่าวในส่วนของฮาร์ดแวร์แล้วว่า ระบบตอบรับโทรศัพท์นี้มีแชนแนลสำหรับการบันทึกและเล่นกลับจำนวน 3 แชนแนล จุดประสงค์หลักของการเขียนซอฟต์แวร์ในส่วนนี้คือให้แต่ละแชนแนลมีการทำงานที่เป็นอิสระต่อกัน กล่าวคือ ในขณะที่ทำการบันทึกแชนแนลหนึ่งก็สามารถทำการบันทึกหรือเล่นกลับในแชนแนลอื่นๆได้พร้อมกันไปด้วย ซึ่งการบันทึกหรือเล่นกลับนี้จะต้องทำการอ่านและเขียนข้อมูลลงบนหน่วยความจำหรือฮาร์ดดิสต์ของเครื่องพีซี พร้อมๆกันไป นอกจากนี้ ในขณะที่ทำการบันทึกหรือเล่นกลับในทั้ง 3 แชนแนล ซอฟต์แวร์จะต้องทำการตรวจสอบคำสั่งต่างๆ ที่ส่งมาจากชุมสาย หรือส่งคำสั่งไปยังชุมสาย เช่นสั่งให้โอนสายอีกด้วยเป็นต้น ซึ่งจะเห็นว่า เป็นการให้ปฏิบัติงานหลายอย่างพร้อมๆกัน

จากลักษณะของปัญหาจะเห็นว่าโปรแกรมจะต้องทำการแบ่งปันเวลาในการทำงานให้กับงานแต่ละส่วนอย่างทั่วถึง โดยเฉพาะการอ่านเขียนข้อมูลในกระบวนการแชมปลิ้งซึ่งต้องการเวลาที่แน่นอน ในที่นี้ทำโดยแบ่งงานต่างๆที่ CPU จะต้องทำให้อยู่ในรูปรูทีน ซึ่งบรรดารูทีนต่างๆเหล่านี้จะบรรจุอยู่ในลูป โดยที่จุดเริ่มต้นของลูปจะทำการตรวจสอบขอบขาขึ้นของสัญญาณนาฬิกาหลัก หากพบจึงจะทำทำบรรดารูทีนต่างๆที่อยู่ถัดไป แต่ละรูทีนจะมีแฟลกซึ่งเป็นตัวบอกให้รู้ว่ารูทีนนั้นถูกอีเนเบิ้ลหรือดีสเอเบิ้ลไว้ ที่ต้นรูทีนเหล่านี้จะทำการตรวจสอบแฟลกของตนถ้าพบว่าอีเนเบิ้ลก็จะทำงาน หากพบว่าดีสเอเบิ้ลก็จะข้ามไปรูทีนถัดไป

อัลกอริธมธ์หลักและรูทีนต่างๆที่ใช้ในโปรแกรมมีดังต่อไปนี้

```
Check_edge.     While ( NOT rising edge );
                        Check ringing;
                        Check DTMF and transfer;
                        Recording external line;
                        Check PBX command;
                        Check time;
                        Phone out;
                        Sampling;
                Jmp     Check_edge
```

ในที่นี้ได้กำหนดการใช้งานระบบตอบรับโทรศัพท์ร่วมกับชุมสายที่พัฒนาขึ้นโดยมีกฎเกณฑ์ดังต่อไปนี้

1. แชนแนลที่ 1 และ 2 ใช้สำหรับตอบรับโทรศัพท์จากสายนอก โอนสายและรับฝากข้อความเมื่อผู้ที่ต้องการโอนสายไปให้ไม่อยู่ หรือโอนไปยังสาย 1 เพื่อให้โอเปอร์เรเตอร์เป็นผู้โอนสายให้ในกรณีที่ผู้โทรเข้ามาไม่รู้เบอร์ภายใน โดยทั้งสองแชนแนลทำงานเสมือนกับว่าถูกควบคุมโดยโปรแกรมสองโปรแกรมที่เหมือนกันแต่ทำงานพร้อมกันหรือเหลื่อมกันได้

2. แชนแนลที่ 3 ใช้สำหรับสายภายในทั้ง 8 สาย เพื่องานต่อไปนี้

2.1 ฝากข้อความไว้ให้ผู้ที่โทรเข้ามาหาเมื่อจะไม่อยู่

2.2 รับฟังข้อความที่มีผู้ฝากไว้ให้ โดยข้อความใดที่รับฟังแล้วระบบก็จะทำการลบให้โดยอัตโนมัติ

2.3 โปรแกรมการโทรออกจากข้อความที่บันทึกไว้ โดยสามารถกำหนดเวลาและหมายเลขที่จะให้ระบบเป็นผู้โทรออกให้

3 โปรแกรมระบบตอบรับโทรศัพท์ให้โอนสายเข้ามาได้เลยหรือให้ระบบเป็นผู้รับแทน

4.2.2 **ตารางสถานะของแต่ละแชนแนล**

เพื่อให้การบันทึกและเล่นกลับของทั้ง 3 แชนแนลสามารถเกิดขึ้นพร้อมกันได้ จึงต้องมีตารางเพื่อบรรจุสถานะของแต่ละแชนแนล เพื่อให้รู้ว่าการบันทึกหรือเล่นกลับในแชนแนลนั้นถึงจุดไหน

ในแต่ละตารางของแต่ละแชนแนลมีโครงสร้างข้อมูลดังต่อไปนี้

```
state     structure
          state, filename, handle, buffer, index, del, count, time, misc, line
state     ends
```

ความหมายในแต่ละฟิลด์เป็นดังนี้

state	ใช้บอกสถานะของแชนแนลนั้นว่ากำลังทำอะไรอยู่โดยมีค่าคงที่ที่มีความหมายดังนี้
REC = 01	หมายถึงแชนแนลนี้กำลังใช้เพื่อการบันทึก
PLY = 02	หมายถึงแชนแนลนี้กำลังใช้เพื่อการเล่นกลับ
NONE = 00	แชนแนลนี้ว่าง
filename	บรรจุชื่อไฟล์ที่ใช้ในการเล่นกลับหรือบันทึก จะต้องเป็นชื่อไฟล์ตามระบบของดอส
handle	หมายถึงตัวจัดการไฟล์ (file handle) ของดอสซึ่งใช้ควบคู่กับชื่อไฟล์ ค่าในฟิลด์นี้จะได้จากดอสหลังจากที่เปิด หรือสร้างไฟล์ ซึ่งเป็นหน้าที่ของผู้เขียนโปรแกรมที่จะเปิดไฟล์เสียงเพื่อการเล่นกลับหรือสร้างไฟล์เพื่อการบันทึกจะต้องเรียกฟังก์ชันของดอส โดยดอสจะส่งตัวจัดการไฟล์มาให้
buffer	เป็นบัฟเฟอร์สำหรับการอ่านและเขียนข้อมูลจากฮาร์ดดิสก์ กำหนดให้มีขนาดเท่ากับ 2000 ไบท์ บัฟเฟอร์นี้เป็นบันเฟอร์ที่ฟังก์ชันการอ่านหรือเขียนไฟล์ของดอสจะใช้
index	เป็นตัวชี้เพื่อใช้บอกว่าทำการแซมปลิ้งไบท์ที่เท่าไรของ buffer อยู่ ค่านี้จะถูกเพิ่มขึ้นทุกๆครั้งที่มีการอ่านหรือเขียนข้อมูลในขณะที่มีการแซมปลิ้ง
del	ใช้ในการเล่นกลับเพื่อบอกใช้บอกให้ลบไฟล์ที่เล่นกลับอยู่หลังจากจบไฟล์แล้วหรือไม่ ไฟล์ที่ไม่ต้องการให้ลบจะเป็นไฟล์เสียงของระบบ ไฟล์ที่ให้ลบคือไฟล์ที่ใช้ชั่วคราว เช่นไฟล์เสียงที่มีผู้ฝากข้อความไว้
count	ใช้นับว่าบันทึกข้อมูลลง buffer จนเต็มไปกี่ครั้งแล้ว ใช้ในการกำหนดเวลาในการบันทึกหรือเล่นกลับ โดยจะเพิ่มขึ้น 1 ทุกๆ 1 buffer หรือทุกๆ 1 วินาที

time กำหนดเวลาที่ใช้ในการบันทึกเพื่อเปรียบเทียบกับ count มีหน่วยเป็นวินาที

misc กำหนดว่าหลังจากเล่นกลับจนจบไฟล์แล้วจะให้ทำอะไรต่อ ค่าในนี้แล้วแต่ผู้เขียนโปรแกรมจะกำหนดขึ้น

line ใช้เป็นพาราเมตเตอร์ของ misc

4.2.3 **ตารางสถานะของแต่ละสาย**

ระบบตอบรับโทรศัพท์จะมีข้อมูลของสายในแต่ละสายอยู่ โครงสร้างข้อมูลที่เก็บไว้ในแต่ละสายเป็นดังนี้

```
line    structure
        filename, pointer, message, phonefile, time, number, state
line    ends
```

โดยข้อมูลในแต่ละฟิลด์มีความหมายดังต่อไปนี้

filename จะบรรจุชื่อไฟล์ที่ใช้เก็บข้อความที่มีผู้โทรเข้ามาแล้วฝากไว้ โดยจะเก็บเรียงกันไป การตั้งชื่อไฟล์จะใช้หมายเลขของสายแล้วตามด้วยลำดับการฝากข้อความ โดยใช้ตัวอักษร A, B, C. เรียงต่อกันไป เช่น ข้อความที่ฝากไว้ให้กับสายที่ 2 จากคนที่ 3 ที่โทรมาฝากจะเก็บไว้ในไฟล์ชื่อ 2C เป็นต้น

pointer จะบรรจุจำนวนคนที่โทรเข้ามาฝากข้อความไว้ให้สายนั้น โดยจะเพิ่มขึ้น 1 เมื่อมีการฝากหนึ่งครั้ง และลดลง 1 เมื่อมีการรับฟังข้อความนั้นไปแล้ว การฝากข้อความและการรับฟังข้อความจะมีลักษณะเป็นสแตก (stack) กล่าวคือ ข้อความที่ฝากไว้ทีหลังเมื่อจะทำการรับฟังจะเป็นข้อความแรกที่ถูกอ่านออกมา

message จะบรรจุชื่อไฟล์ที่สายภายในฝากไว้ให้กับผู้โทรเข้ามา

phonefile บรรจุชื่อไฟล์ที่สายภายในฝากไว้เพื่อให้โทรออก

time เป็นเวลาที่สายภายในโปรแกรมไว้ให้เพื่อโทรออกโดยใช้เสียงจากไฟล์ข้างต้น

number หมายเลขที่ใช้โทรออก

state บอกสถานะของสายนั้น มีอยู่ 4 สถานะคือ

- REC ทำการบันทึกเสียงอยู่
- PLY ทำการเล่นกลับอยู่
- ISIN อยู่ ถ้ามีสายนอกให้โอนมาได้
- ISOUT ไม่อยู่ ถ้ามีสายนอกให้ฝากข้อความไว้

4.2.4 **การตรวจสอบสัญญาณเรียก**

รูทีนที่เกี่ยวข้อง Check ringing;

หลังจากตรวจพบขอบขาขึ้นของสัญญาณนาฬิกาหลักก็จะทำการตรวจสอบว่ามีสัญญาณเรียกในแชนแนลที่ 1 หรือ 2 หรือไม่ หากพบว่ามีก็จะทำการเล่นกลับไฟล์แนะนำระบบ (ไฟล์ Intro) โดยการเขียนข้อมูล

ที่เกี่ยวข้องลงในตารางสถานะของแชนแนลที่มีสัญญาณเรียกเข้ามา เสร็จแล้วก็จะกลับไปตรวจสอบขอบขาขึ้นของสัญญาณนาฬิกาหลักอีกครั้ง หากไม่พบก็จะข้ามไปตรวจสอบรูทีนถัดไป (Check DTMF) ค่าที่ใส่ลงในตารางสถานะเมื่อพบว่ามีสัญญาณเรียกในแชนแนลที่ 1 จะเป็นดังนี้

```
If (ringing on channel 1) {
        Close relay channel 1,
        Handle = Open (Intro1),
        states[1].state = PLY;
        states[1].handle = Handle;
        states[1].misc = 00;
        states[1].index = 0000;
        states[1].count = 00;
        states[1].time = 30d;        ; REM  เล่นกลับเป็นเวลา 30 วินาทีหรือจบไฟล์
        states[1].del = 00;
}
jmp Check_edge;
```

4.2.5 **การตรวจสัญญาณ DTMF**

รูทีนที่เกี่ยวข้อง : Check DTMF and transfer;

การตรวจสัญญาณ DTMF นี้ใช้ในกรณีที่รอให้ผู้ใช้สายนอกกดเลขหมายเพื่อทำการโอนสายซึ่งจะทำการตรวจสอบหลังจากตรวจพบสัญญาณเรียกในแชนแนลที่เกี่ยวข้อง อัลกอริธึมธ์เป็นดังนี้

```
If ( DTMF on channel 1 Exist ) {
        If (First Time) {          rem: First Time is first initialized to .F.
                Read Code to X;
                Jmp Clock_edge,
                First Time = .F.,
                Jmp Check_edge,
        }
        else {
                First time = T.;
                Read Code to Y;
                If ( XY is internal line number ) {
                        If (The internal line is programed by service number 55) {
                                PlayBack (M?);    rem: M? is message file that prefered
```

```
                    Recording channel 1 = T ,
                }
                else {
                    Transfer to the internal line,
                    Jmp Clock_edge:          }
            else {
                transfer to internal line number 1,
                Jmp Clock_edge;          }
        }
```

4.2.6 **การบันทึกเสียงจากสายนอก**

รูทีนที่เกี่ยวข้อง . Recording external line;

เมื่อผู้ใช้สายนอกกดเลขหมายเพื่อให้ระบบโอนสายไปให้สายภายในแต่สายในนั้นโปรแกรมให้ระบบตอบรับรับสายแทน ระบบจะเล่นกลับไฟล์ M? ที่สายในนั้นบันทึกไว้ ซึ่งหลังจากเล่นกลับจบแล้วรูทีนการบันทึกเสียงก็จะถูกอีเนเบิ้ลผ่านฟิลด์ misc ของตารางสถานะ และเขียนเบอร์สายในที่จะถูกฝากข้อความไว้ให้ลงในฟิลด์ line ซึ่งในการบันทึกเสียงทำโดยใส่ข้อมูลลงในตารางสถานะให้ครบ การค้นหาชื่อไฟล์ที่ใช้ในการบันทึกทำได้โดยอ่านชื่อไฟล์ที่ฟิลด์ pointer ของสายในที่จะถูกฝากข้อความไว้ให้โดยอ่านจากฟิลด์ pointer ของตารางสถานะของแต่ละสาย (lines) เมื่อได้ชื่อไฟล์แล้วจึงนำมาหาค่าตัวจัดการไฟล์โดยผ่านดอส หลังจากบันทึกแล้วค่าในฟิลด์ pointer จะเพิ่มขึ้น 1 ซึ่งจะบอกตำแหน่งชื่อไฟล์ที่จะถูกบันทึกครั้งต่อไป ในรูปเป็นตัวอย่างกรณีที่จะบันทึกข้อมูลลงไฟล์ชื่อ 2D

Filename													pointer				message		
2	A	0	2	B	0	2	C	0	2	D	0	2	E	0	0	0	0	3	M	2	0

```
If ( Recording channel 1 = T ) {
        Filename = File pointed by pointer,
        Handle = Create (Filename);
        states[1] state = REC;
        states[1].handle = Handle;
        states[1].misc = 00;
        states[1].index = 0000,
        states[1].count = 00,
        states[1] time = 30d;
```

```
states[1].del = 00.
}
```

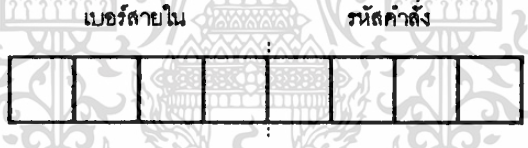
การบันทึกข้อความจากสายนอกจะเป็นลักษณะสแตกคือ ฟิลด์ pointer จะชี้ที่ชื่อไฟล์ถัดไปที่จะถูกบันทึก ซึ่งในการเล่นกลับก็จะทำการเล่นกลับจากไฟล์หลังสุดที่ถูกบันทึกไว้โดยการลดค่า pointer ลง 1 ก็จะได้ชื่อไฟล์ที่จะถูกเล่นกลับ

4.2.7 **การรับคำสั่งจากชุมสายโทรศัพท์**

รูทีนที่เกี่ยวข้อง . Check PBX command,

คำสั่งที่ส่งจากชุมสายไปยังระบบตอบรับจะเกิดขึ้นในกรณีที่ผู้ใช้กดปุ่มเลขหมายเพื่อขอใช้บริการ (เลขหมาย 51 ถึง 55) โดยบริการที่ต้องใช้แชนแนลที่ 3 คือ 51, 52 และ 53 ในกรณีนี้ระบบตอบรับจะตรวจสอบดูว่าแชนแนลดังกล่าวถูกใช้โดยผู้ใช้คนอื่นหรือไม่ หากถูกใช้อยู่ผู้ที่กดเลขหมายดังกล่าวจะได้ยินสัญญาณสายไม่ว่าง ส่วนบริการ 54 และ 55 ไม่จำเป็นต้องใช้แชนแนลดังกล่าว กระบวนการในการตอบโต้กันระหว่างชุมสายและระบบตอบรับเป็นดังรายละเอียดที่กล่าวในหัวข้อ ต่อไปนี้จะกล่าวรายละเอียดของอัลกอริธึมธ์ในรูทีน Check PBX command

1 คำสั่งที่ส่งมาจากชุมสายเป็นคำสั่งขนาด 1 ไบท์โดย 4 บิทล่างเป็นรหัสคำสั่งและ 4 บิทบนเป็นหมายเลขของสายใน (0 ถึง 7) ที่ส่งคำสั่งนั้นออกมา



รูทีนในส่วนนี้จะอ่านพอร์ทรับคำสั่งของระบบตอบรับเพื่อดูว่ามีรหัส 00, 01, 02, 03, 04, 05 ซึ่งหมายถึงชุมสายต้องการบันทึกเสียง เล่นกลับเสียง โปรแกรมโทรออก ให้ชุมสายโอนสายนอกเข้ามา ให้ระบบรับสายแทนตามลำดับหรือไม่ หากพบว่าเป็นคำสั่งดังกล่าวก็จะกระทำงานที่เกี่ยวข้องกับคำสั่งนั้นแล้วกลับไปตรวจสอบขอบขาขึ้นของสัญญาณนาฬิกาหลักใหม่ หากไม่พบก็จะไปกระทำรูทีนถัดไป (Check time) อัลกอริธึมธ์ในการตรวจสอบเป็นดังนี้

```
Check_PBX_command   Read Command from PBX to X;
                    store low nibble to Y;
                    Case Y of
                            00 : do_hang_down;
                            01   do_recording;
                            02 . do_playing;
                            03 : do_phone_out;
                            04 : do_in_office;
```

```
05 . do_out_office;
default : Jmp Check time;
Endcase
```

รายละเอียดในแต่ละงานเป็นดังนี้

1.1 **do_recording** การบันทึกเสียงนี้จะบันทึกไว้ในไฟล์ซึ่งมีชื่ออยู่ในฟิลด์ M* เช่น ถ้าสายที่ 3 ขอบันทึกเสียง เสียงนั้นจะถูกบันทึกไว้ในไฟล์ M3 เสียงที่บันทึกนี้เป็นข้อความที่ผู้ใช้สายในต้องการที่จะฝากข้อความให้ผู้โทรเข้ามา อัลกอริธึมธ์ในส่วนนี้จะเริ่มที่การตรวจสอบว่าแชนแนลที่ 3 ว่างหรือไม่ หากว่างก็จะหาว่าสายใดเป็นผู้ออกคำสั่งโดยอ่าน 4 บิทบนของคำสั่งที่ส่งออกมาจากชุมสาย จากนั้นก็จะอ่านเรคคอร์ดของสายนั้นจากตารางข้อมูลของสายใน โดยจะทำการอ่านฟิลด์ message เพื่อที่จะนำชื่อไฟล์ที่ต้องการจะบันทึกไปหาตัวจัดการไฟล์โดยผ่านดอส เมื่อดอสให้ค่ามาก็จะนำค่านั้นไปใส่ในฟิลด์ handle ในตารางสถานะของสายที่ 3

```
do_recording:   Store high nibble  to X;          rem :  X คือสายในที่ออกคำสั่ง
                If ( states [3].state = 00 ) {
                    Filename := lines [X].filename;
                    Handle := CreateFile (Filename);
                    lines [X] state := REC;
                    states[1].state = REC;
                    states[1].handle = Handle;
                    states[1].misc = 00;
                    states[1].index = .0000;
                    states[1] count = 00;
                    states[1].time = 30d;
                    states[1].del = 00,
                }
                else { Tell PBX that command not accepted }
```

1.2 **do_playing** การเล่นกลับเสียงจะเล่นกลับจากไฟล์ที่มีชื่อปรากฏอยู่ในฟิลด์ filename ซึ่งถูกชี้โดยฟิลด์ pointer อีกทีหนึ่ง ในตัวอย่างเป็นเรคคอร์ดของสายที่ 2 ในตารางสถานะของสายใน ซึ่งแสดงกรณีที่มีผู้บันทึกเสียงฝากข้อความไว้ให้สายที่ 2 จำนวน 3 คน ซึ่งขณะนี้ pointer จะชี้ที่ชื่อไฟล์ 2D ซึ่งหมายถึงชื่อไฟล์ที่จะถูกบันทึกครั้งต่อไป แต่เมื่อต้องการเล่นกลับจะเล่นกลับจากไฟล์ที่บันทึกหลังสุดก่อนซึ่งก็คือ 2C ซึ่งทำได้โดยลดหา pointer ลงหนึ่งเพื่อให้ชี้ที่ไฟล์ดังกล่าว

Filename														pointer				message		
2	A	0	2	B	0	2	C	0	2	D	0	2	E	0	0	0	0	3	M	2	0

สรุปอัลกอริธึมธ์สำหรับการเล่นกลับได้ดังนี้

```
do_ playing :    Store high nibble  to X;              rem . X คือสายในที่ออกคำสั่ง
                 If ( states [3].state = 00 ) {
                        Filename .= lines [X].filename,
                        Handle .= OpenFile (Filename);
                        lines [X] state := PLY;
                        states[1].state = PLY;
                        states[1].handle = Handle,
                        states[1].misc = 00;
                        states[1].index = 0000;
                        states[1].count = 00;
                        states[1].time = 30d;
                        states[1].del = 01,
                 }
                 else { Tell PBX that command not accepted }
```

สังเกตว่าในฟิลด์ del จะใส่ค่า 01 ไว้ ซึ่งเป็นการบอกว่าให้รูทีนส่วนอ่านเขียนสัญญาณจากการแซมปลิ้งทำการลบไฟล์นี้ออกไปเมื่อเล่นกลับจบไฟล์แล้ว

1.3 **do hangdown** เมื่อผู้ใช้สายในกำลังใช้บริการของระบบตอบรับอยู่ หากทำการวางหูจะถือว่าเป็นการสิ้นสุดการขอใช้บริการ ซึ่งในกรณีนี้ชุมสายจะส่งรหัสคำสั่งมาบอกระบบตอบรับเพื่อให้ระบบตอบรับปิดไฟล์ที่กำลังบันทึกอยู่ในกรณีเป็นการบันทึกเสียงหรือลบไฟล์ที่กำลังเล่นกลับอยู่ในกรณีที่กำลังเล่นกลับไฟล์ข้อความที่มีผู้ฝากไว้ให้สายนั้น (ซึ่งขณะในบริการของระบบตอบรับชุมสายสายนั้นจะอยู่ในสถานะ use_avmstate โดยจะตรวจสอบว่าถ้าอยู่ในสถานะนี้และมีการวางหูก็ให้แจ้งให้ระบบตอบรับทราบ)

1.4 **do phone out** การตั้งโปรแกรมเริ่มจากผู้ใช้กดเลขหมาย 53 จากเครื่องโทรศัพท์ ซึ่งระบบจะเล่นกลับไฟล์ Phone ซึ่งจะบรรยายวิธีการโปรแกรมให้ผู้ใช้ทราบ จากนั้นก็จะรอให้ผู้ใช้กดเลขหมายโดย 4 ตัวแรก เป็นชั่วโมงและนาทีที่ต้องการให้โทรออกและจากนั้นเป็นเลหมายที่ต้องการให้โทรโดยจบด้วยปุ่ม # จากนั้นระบบจะเล่นกลับไฟล์ Phonerec ซึ่งจะบอกให้ผู้ใช้กล่าวข้อความที่จะให้โทรออกแล้วจึงวางหูเป็นการจบการโปรแกรม หากแชนแนลที่ 3 ซึ่งใช้ในงานนี้ไม่ว่างผู้ใช้จะได้ยินเสียงสัญญาณสายไม่ว่าง ซึ่งผู้ใช้จะต้อง

ต้องต่อเข้ามาใหม่ อนึ่งการโปรแกรมจะโปรแกรมได้ครั้งละ 1 สายเนื่องจากในการโปรแกรม 1 ครั้งแซนแนลที่ 3 จะถูกจองเพื่อใช้งาน อัลกอริธึมธ์เป็นดังนี้

```
do_phoneout     Store high nibble  to X,              rem  X คือสายในที่ออกคำสั่ง
                If ( states [3] state = 00 ) {
                        Handle  = OpenFile (Phonerec),
                        lines [X] state .= PLY,
                        states[1] state = PLY;
                        states[1].handle = Handle,
                        states[1] misc = 00;
                        states[1].index = 0000,
                        states[1] count = 00,
                        states[1] time = 30d;
                        states[1].del = 00;
                        check DTMF channel 3 = .T.,
                }
                else { Tell PBX that command not accepted }
```

หลังจากนี้จะเป็นกระบวนการโปรแกรมเวลาและหมายเลขที่ต้องการจะโทรออก

If (check DTMF channel 3) {

1 5 **do in office** เมื่อผู้ใช้สายในกดเลขหมาย 54 ระบบตอบรับจะเขียนรหัส ISIN ลงไปในฟิลด์ state ของตารางสถานะของสายใน

1 6 **do out office** เมื่อผู้ใช้สายในกดเลขหมาย 55 ระบบตอบรับจะเขียนรหัส ISOUT ลงไปในฟิลด์ state ของตารางสถานะของสายใน

4 2 8 **<u>กระบวนการโทรออก</u>**

รูทีนที่เกี่ยวข้อง Check time,

Phoneout,

1 <u>ตรวจสอบเวลาเพื่อจะทำการโทรออก</u>

รูทีนนี้จะทำการตรวจสอบฟิลด์ time ของตารางสถานะของสายใน โดยเริ่มตั้งแต่สายที่ 1 ไปจนถึงสายที่ 8 โดยจะนำมาเปรียบเทียบกับเวลาที่ได้จากฟังก์ชัน 2CH ของดอส ฟิลด์นี้มีขนาด 2 ไบท์ ไบท์แรกเป็นชั่วโมง มีค่าอยู่ในช่วง 0 ถึง 24 ไบท์ที่สองเป็นนาทีมีค่าอยู่ในช่วง 0 ถึง 59 การเปรียบเทียบจะเปรียบเทียบว่าเวลาที่ได้จากดอสอยู่ในช่วงเวลาที่กำหนดในฟิลด์นี้บวกกับอีก 15 นาทีหรือไม่ ถ้าอยู่ในช่วงดังกล่าวก็จะเริ่มกระบวนการโทรออก แต่ถ้าไม่ก็จะเปรียบเทียบสายถัดไปในอีกรอบของลูป กล่าวคือในหนึ่งรอบของลูปจะทำการโทรออกให้กับสายเดียวเท่านั้น

2 การส่งรหัส DTMF

เมื่อผ่านขั้นตอนที่ 1 คือพบว่าสายที่ i โปรแกรมให้โทรออกในขณะเวลานี้ ก็จะปิดรีเลย์เพื่อต่อวงจรของระบบตอบรับเข้ากับชุมสายภายนอก จากนั้นจะรอประมาณ 5 วินาทีจึงทำการตรวจสอบสัญญานให้หมุนในสายที่ 3 หากไม่มีก็จะจบกระบวนการ หากพบว่ามีก็จะเริ่มส่งสัญญาน DTMF จากเลขหมายที่ระบุในฟิลด์ number ในตารางสถานะของสายนั้น ดังได้กล่าวในหัวข้อฮาร์ดแวร์ที่ 3.3.6 แล้วว่า การส่งเลขหมายมีสองขั้นตอนคือ หนึ่งต้องส่งรหัสเลหมาย (ตารางที่ 3.3) ไปแลทช์ไว้ในช่วงขอบขาขึ้นของพัลซ์ 500 ms จากนั้นรหัสดังกล่าวจะถูกนำไปดีโคดเป็นสัญญาน DTMF ส่งไปในสายโทรศัพท์ในช่วงระดับลอจิก 0 ของพัลซ์ดังกล่าว การส่งสัญญานจะกระทำไปเรื่อยๆจนกว่าจะพบอักขระ '#' ในฟิลด์ number ซึ่งบอกจุดสิ้นสุดของการส่ง

3. การตรวจการรับสาย

เมื่อส่งสัญญาน DTMF ซึ่งบ่งบอกปลายทางที่ต้องการติดต่อได้แล้วก็จะเข้าสู่กระบวนการตรวจการรับสายของปลายทาง ซึ่งแบ่งเป็นสองขั้นตอน ขั้นตอนแรกจะตรวจสอบว่าได้รับสัญญานตอบกลับเป็นสัญญานสายไม่ว่างหรือสัญญานเรียกกลับ หากได้รับสัญญานสายไม่ว่างก็จะจบกระบวนการโทรออกของสายนั้น แต่หากได้รับสัญญานเรียกกลับก็จะผ่านไปขั้นตอนที่สองคือตรวจสอบว่าปลายทางรับสายแล้วหรือยัง ซึ่งขั้นตอนนี้จะตรวจสอบโดยซอฟต์แวร์

เมื่อพบว่าปลายทางรับสายแล้ว โปรแกรมก็จะเล่นกลับเสียงจากไฟล์ที่ชื่อปรากฏอยู่ในฟิลด์ phonefile ของสายที่กำลังโทรออก ซึ่งเป็นการจบกระบวนการโทรออก

4.2.9 **การอ่านเขียนข้อมูลที่ได้จากการแซมปลิ้ง**

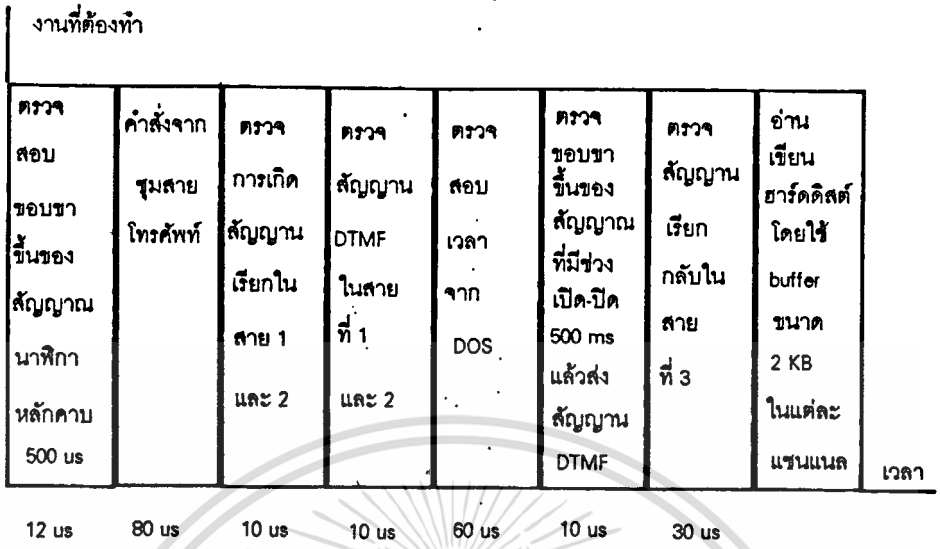
ส่วนนี้จะถูกตรวจสอบและกระทำที่ปลายลูป ซึ่งรูทีนในส่วนนี้จะทำการตรวจสอบฟิลด์ state ของตารางสถานะของแชนแนลว่ามีการจองการใช้แชนแนลนั้นเพื่อการอ่านหรือบันทึกจากรูทีนข้างต้นหรือไม่ ถ้ามีก็จะทำการอ่านหรือเขียนข้อมูลขนาด 1 ไบท์โดยอาศัยข้อมูลในฟิลด์อื่นๆประกอบ การอ่านและเขียนจะกระทำผ่านฮาร์ดดิสต์ของไมโครคอมพิวเตอร์ โดยจะใช้หน่วยความจำเป็นบัฟเฟอร์ขนาด 2 KB ก่อนที่จะมีการอ่านหรือเขียนลงบนฮาร์ดดิสต์ สรุปเป็นอัลกอริธึมธ์ในแต่แชนแนลได้ดังนี้

```
If (states [1].state = REC) {
        Select hardware channel 1 for recording;
        Read 1 byte data from recording hardware;
        Save data to states.buffer pointed by index;
        increment index by 1,
        If ( index = 2KB) {
                Save buffer to file indicated by states [1].handle;
                increment states [1].count by 1;
                reset index;
                If (states [1].count = states [1] time) {
```

```
                    CloseFile (states [1].handle);
                    reset states [1].count;
                    states [1].state = NONE;
                }
            }
    If (states [1].state = PLY) {
            If ( states [1].index = 0000 ) {
                    ReadFile ( states [1].handle ) with 2KB;
                    If ( EOF ) {
                            CloseFile ( states [1].handle );
                            states [1].state = NONE;
                            If ( states [1].del = 01 ) DeleteFile ( state [1].handle);
                    }
                    else {
                            Write 1 byte from buffer pointed by index;
                            Increment index;
                    }
}
```

4.2.10 **สรุปส่วนต่าง ๆของซอฟต์แวร์**

ดังได้กล่าวไว้แล้วว่าในหนึ่งวงรอบของลูปจะถูกควบคุมโดยสัญญาณนาฬิกาหลักซึ่งมีคาบ 500 us แต่ในทางปฏิบัติ CPU อาจจะทำงานในส่วนใดส่วนหนึ่งนานเกินกว่า 500 us ซึ่งจะเป็นผลให้ CPU ใช้เวลาในหนึ่งรอบเกินกว่า 500 us ซึ่งอาจจะเป็น 1000us, 1500us หรือเพิ่มขึ้นทีละ 500 us ซึ่งจะเกิดขึ้นในกรณีที่มีการบันทึกหรืออ่านข้อมูลจากฮาร์ดดิสต์ ถ้าพิจารณาจากต้นฉบับซอฟต์แวร์ในภาคผนวกที่ 1 ซึ่งเป็นภาษาแอสแซมบลี้ ประมาณจากเวลาเฉลี่ยในหนึ่งคำสั่ง 2 us (CPU 386DX 40 Mhz) จะได้เวลาเฉลี่ยที่ใช้ในแต่ละส่วนได้ดังนี้



แผนภาพแสดงเวลาเฉลี่ยในแต่ละส่วนของซอฟต์แวร์

โดยเวลาที่ใช้ในส่วนอ่านเขียนฮาร์ดดิสต์ขึ้นอยู่กับปัจจัยหลายอย่างได้แก่ ชนิดของไมโครคอมพิวเตอร์ ชนิดของคอนโทรลเลอร์ ชนิดของฮาร์ดดิสต์ และการกระจัดกระจายของข้อมูลซึ่งอยู่ในรูปไฟล์ ซึ่งทำให้ไม่สามารถกำหนดเวลาที่ใช้ในการอ่านหรือบันทึกได้อย่างแน่นอน แต่มีแนวโน้มในลักษณะที่ถ้าขนาดของบัฟเฟอร์มาก เวลาที่ใช้ในส่วนนี้จะมีค่ามากขึ้น ซึ่งจะทำให้เวลาที่เกินไปในแต่ละลูปอยู่ในระยะ 500 us ดังได้กล่าวไว้ข้างต้น

4.3 ซอฟต์แวร์ของส่วนแจ้งข่าวสารจากส่วนกลาง

ซอฟต์แวร์นี้แบ่งเป็นสองส่วน ส่วนแรกเป็นส่วนที่อยู่บนเครื่องไมโครคอมพิวเตอร์เขียนด้วยแอสแซมบลี้ 8088 ทำหน้าที่ส่งข้อมูลขนาด 256 ไบท์จากไฟล์ที่ชื่อ inform ไปยังบอร์ด 8031 โดยซอฟต์แวร์บนพีซีจะอ่านบิทที่ 7 ของพอร์ท 306H โดยถ้ามีค่าเป็น 1 แสดงว่าบอร์ด 8031 พร้อมที่จะรับข้อมูล พีซีก็จะส่งข้อมูลขนาด 1 ไบท์จากไฟล์ที่ชื่อ inform ไปยังพอร์ทหมายเลข 306H แล้วรอให้บอร์ด 8031 อ่านข้อมูลไปเก็บไว้ในหน่วยความจำ กระบวนการนี้จะดำเนินไปเรื่อยๆจนครบ 256 ไบท์ โปรแกรมส่วนนี้อยู่ในภาคผนวก 3

ส่วนที่สองเป็นซอฟต์แวร์มอนิเตอร์ของ 8031 ซึ่งจะทำงานสอดคล้องกับส่วนแรก โดยจะส่งสัญญานบอกให้พีซีส่งข้อมูลมาให้ เมื่อพีซีส่งข้อมูลมาแล้วก็ส่งสัญญานบอกพีซีว่าได้รับข้อมูลแล้ว จนกระทั่งได้ข้อมูลครบ 256 ไบท์ หลังจากนั้นจึงเริ่มส่งข้อมูลนั้นผ่านไปทางพอร์ทอนุกรมโดยจะเริ่มคอมพิวเตอร์เครื่องที่ 1 ไปจนถึงเครื่องที่ 4 และวนอย่างนี้ไปเรื่อยๆ โดยเครื่องคอมพิวเตอร์ที่รับข้อมูลจะต้องรันโปรแกรมประเภทสื่อสารข้อมูลเช่น Telix หรือ Procomm ไว้ โปรแกรมส่วนนี้อยู่ในภาคผนวก 4

บทที่ 5
การทดสอบและวิจารณ์ผล

ในบทนี้จะกล่าวถึงผลการทดสอบระบบชุมสายโทรศัพท์ที่พัฒนาขึ้น โดยจะเริ่มจากการทดสอบชุมสายในกรณีที่ทำงานเดี่ยวๆ การทำงานของระบบตอบรับโทรศัพท์ การทำงานของชุมสายโทรศัพท์เมื่อทำงานร่วมกับระบบตอบรับ และผลการทดสอบระบบแจ้งข่าวสารตามลำดับ ข้อเสนอแนะและข้อวิจารณ์จะแทรกไว้ในหัวข้อนั้นๆ

5.1 อุปกรณ์หลักที่ใช้ในงานวิจัย

1 ชุมสายโทรศัพท์พานาโซนิค KX-T30810 ขนาด 3 สายนอก 8 สายใน

2. เครื่องไมโครคอมพิวเตอร์ 386DX ความถี่ 40 Mhz ฮาร์ดดิสก์ 170 MHz

5.2 การทดสอบการทำงานของชุมสายโทรศัพท์

การทดสอบในส่วนแรกเป็นการทดสอบการทำงานของระบบชุมสายเมื่อใช้งานตามปกติโดยไม่ได้ต่อร่วมกับระบบตอบรับโทรศัพท์ ประเด็นหลักที่ทำการทดสอบมีดังนี้

5.2.1 ความเร็วในการตอบสนองต่อการกดปุ่ม

เนื่องจากได้กล่าวรายละเอียดในหัวข้อ 3.2.4 และ 4.1.4 แล้วว่าวงจรถอดรหัส DTMF มีอยู่ 2 วงจรหรือมีการใช้วงจรถอดรหัส 1 วงจรต่อสายใน 4 สาย ซึ่งมีผลดีในแง่ประหยัดวงจรถอดรหัสแต่มีข้อเสียคือการตอบสนองต่อการกดปุ่มของแต่ละสายจะลดลงเมื่อเมื่อเทียบกับการใช้หนึ่งวงจรต่อหนึ่งสาย ดังนั้นจึงมุ่งการทดสอบประสิทธิภาพในส่วนนี้

เวลาที่ใช้ในหนึ่งรอบของลูป loopline (ดูหัวข้อ 4.1) ถูกกำหนดโดยเวลาที่ถูกหน่วงในรูทีน delay ซึ่งบรรจุอยู่ในลูปนั้น เวลาที่เราต้องการคือเวลาที่สั้นที่สุดที่ทำให้การถอดรหัสจากแต่ละสายเป็นไปอย่างถูกต้อง ซึ่งเวลานี้ได้จากการทดลองโดยเปลี่ยนจำนวนรอบในลูปของ delay จนกว่าจะได้ค่าที่เหมาะสมที่สุด รูทีน delay ดังกล่าวเป็นดังนี้

```
delay   push   de
        ld     de, 3000H
ttt.    dec    de
        ld     a, d
        or     e
        jr     nz, ttt
        pop    de
        ret
```

ซึ่งตัวเลข 3000H ได้มาจากการทดลอง และประมาณจากการทดลองกดปุ่มพบว่าผู้ใช้ต้องกดปุ่มเป็นเวลาประมาณครึ่งวินาทีจึงจะสามารถถอดรหัสได้เป็นอย่างช้า (การถอดรหัสได้เร็วเกิดขึ้นเมื่อจังหวะที่กดตรงกับจังหวะที่สายนั้นถูกเลือกให้ต่อเข้ากับวงจรถอดรหัสพอดี)

อย่างไรก็ตามหากต้องการให้การตอบสนองเป็นไปอย่างรวดเร็ว อาจเลือกใช้วงจรถอดรหัสหนึ่งวงจรต่อสายในหนึ่งสายซึ่งสามารถตัดอัลกอริธึมธ์ในหัวข้อ 4.1.4 ออกไปจากโปรแกรมได้ แต่จะต้องใช้ไอซีถอดรหัส MT8870 เท่ากับจำนวนสายในซึ่งก็คือ 8 ตัว

5.2.2 <u>การติดต่อกันภายใน</u>

เนื่องจากการจัดการให้แต่สายในของ CPU นั้นใช้การวนลูปเป็นหลัก ซึ่ง CPU จะคอยติดตามสถานะของสายในแต่ละสายและพิจารณาปัจจัยที่ทำให้สายต่างๆเปลี่ยนสถานะ การทำงานจึงเป็นแบบทีละสายไล่จากสายที่ 1 ไปจนถึงสายที่ 8 และกลับมาสายที่ 1 อีก การทดสอบจะดูว่านับตั้งแต่การยกหูขึ้นโทรจนกระทั่งสนทนาแล้ววางหูลง ความเร็วในการตอบสนองเป็นอย่างไร มีการหน่วงเวลาในขณะใช้งานจนผิดปกติวิสัยในการใช้งานโดยทั่วไปหรือไม่ จุดมุ่งหมายคือทดสอบประสิทธิภาพของอัลกอริธึมธ์ที่จัดการในส่วนดังกล่าว โดยตั้งแบบแผนการทดสอบดังนี้

1. ให้โทรศัพท์ 4 เครื่องทำการโทรออกพร้อมกัน และให้ผู้รับ 4 คนรับสายพร้อมกัน สนทนาแล้ววางหูพร้อมกัน การทดสอบนี้ต้องการจะดูผลเมื่อผู้ใช้ทั้งหมดทำกระบวนการโทรออก สนทนาและวางหูพร้อมกัน

2. ให้สาย 1 โทรเข้าหาสายที่ 2 และขณะที่สนทนากันอยู่ให้สายที่ 3 โทรเข้าหาสายที่ 4 ขณะเดียวกันให้คู่สายที่ 1 และ 2 จบการสนทนา และสายอื่นก็กระทำลักษณะเป็นลูกโซ่ต่อไปเรื่อยๆ การทดสอบนี้ต้องการจะดูผลเมื่อผู้ใช้ทำกระบวนการโทรออก สนทนาและวางหูในเวลาที่เหลื่อมกัน

จากทั้งสองกรณีซึ่งเป็นกรณีที่ CPU จะต้องทำงานให้กับทุกสายพบว่าการตอบสนองในแต่ละสายเป็นไปตามปกติ กล่าวคือผู้ใช้ที่กำลังใช้สายหนึ่งจะไม่รู้สึกว่า CPU ถูกแย่งเวลาไปเมื่อมีผู้ใช้คนอื่นอยู่ด้วย ดังนั้นหากขยายสายในให้มากขึ้น (โดยการเปลี่ยนแปลงตัวเลขการวนลูปในโปรแกรมบางจุด) ก็ไม่น่าจะมีผลต่อความเร็วในการตอบสนองต่อการใช้ของผู้ใช้แต่ละคน

5.3 **<u>การทำงานของระบบตอบรับโทรศัพท์</u>**

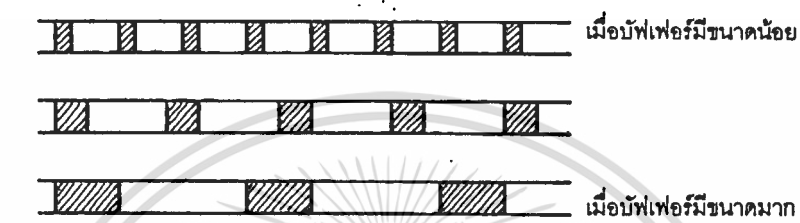
เนื่องจากซอฟต์แวร์ที่ใช้ควบคุมระบบตอบรับโทรศัพท์ที่เสนอในงานวิจัยนี้เป็นกรณีที่ใช้ร่วมกับชุมสายโทรศัพท์ที่พัฒนาขึ้น ดังนั้นการทดสอบจึงเป็นการทดสอบร่วมของอุปกรณ์ทั้งสอง ประเด็นหลักในการทดสอบเป็นดังนี้คือ

1. <u>เสียงที่ได้จากการบันทึกและเล่นกลับ</u>

ในกระบวนการบันทึกและเล่นกลับ การอ่านเขียนฮาร์ดดิสต์จะกระทำเมื่อบัฟเฟอร์ขนาด 2KB ได้รับการอ่านหรือเขียนจนเต็มหรือประมาณ 1 วินาทีต่อครั้ง จากการทดลองพบว่าช่วงเวลาที่ใช้ในการบันทึกหรือเขียนลงบนฮาร์ดดิสต์เป็นเวลาที่สั้นมาก (ภายใต้ฮาร์ดแวร์ข้างต้น) เสียงที่ได้จึงมีความต่อเนื่องไม่ขาดหายไป หากใช้กับเครื่องที่มีความเร็วต่ำกว่านี้ พบว่าเสียงมีความขาดหายเป็นห้วงๆในขณะบันทึกหรืออ่านจากฮาร์ดดิสต์ จากการทดลองพบว่าความถี่ต่ำสุดที่ให้เสียงชัดในระดับจับใจความได้คือ 33MHz (ทดสอบจาก

ซอฟต์แวร์ของระบบตอบรับทั้งหมด หากซอฟต์แวร์มีแต่ส่วนที่ใช้ในการแซมปลิ้งจะใช้เครื่องที่มีความถี่ต่ำกว่านี้ได้อีก)

ขนาดของบัฟเฟอร์นั้นสามารถเปลี่ยนแปลงได้ โดยถ้าบัฟเฟอร์มีขนาดใหญ่กว่า 2KB จะทำให้การอ่านเขียนฮาร์ดดิสต์ถี่น้อยลง เสียงที่ได้จะเรียบเป็นช่วงและขาดหายเป็นช่วง แต่ถ้าบัฟเฟอร์มีขนาดน้อยกว่า 2 KB การอ่านเขียนฮาร์ดดิสต์จะถี่ขึ้น เสียงที่ได้เหมือนกับว่าขาดหายเป็นช่วงเวลาน้อยๆ ฟังดูแล้วขาดความสม่ำเสมอ จากการทดลองพบว่าบัฟเฟอร์ขนาด 2 KB เป็นขนาดที่เหมาะที่สุดสำหรับวิธีการที่นำเสนอนี้



แผนภาพแสดงการกระจายของเวลาเมื่อมีการเลือกบัฟเฟอร์ขนาดต่างๆ

แผนภาพข้างต้นแสดงการกระจายของเวลาเมื่อเลือกใช้บัฟเฟอร์ที่ต่างขนาดกัน

2. คุณภาพของเสียงเมื่อมีการบันทึกและเล่นกลับในทั้งสามแชนแนลพร้อมกัน

เมื่อมีการใช้งานทั้งสามแชนแนลพร้อมกันจะทำให้มีการอ่านและบันทึกฮาร์ดดิสต์ถี่ขึ้น จากการทดลองกับเครื่องไมโครคอมพิวเตอร์ที่มีความถี่ 40MHz พบว่าเสียงจะขาดหายเป็นช่วงสั้นๆ แต่ก็ยังคงความชัดเจนไว้ได้ ดังนั้นข้อจำกัดของการใช้ซอฟต์แวร์ในการจัดการงานทั้งหมดในแบบที่เสนอในวิทยานิพนธ์นี้จึงอยู่ที่ต้องการเครื่องไมโครคอมพิวเตอร์ที่มีความเร็วค่อนข้างสูง โดยเฉพาะอย่างยิ่งถ้าต้องการที่จะเพิ่มแชนแนลของการเล่นกลับและบันทึกให้มากขึ้น ความเร็วของเครื่องก็จะต้องสูงมากกว่า 40 Mhz

คุณภาพของเสียงกับความเร็วของเครื่องไมโครคอมพิวเตอร์จะเป็นปัญหาในกรณีที่เราต้องการจะบันทึกข้อมูลหรืออ่านข้อมูลจากฮาร์ดดิสต์ในลักษณะกึ่งเวลาจริง (real time) ในแบบที่นำเสนอในที่นี้เท่านั้น หากเรากำหนดเงื่อนไขใหม่ว่าในการบันทึกแต่ละครั้งจะจำกัดเวลาไว้ (แบบที่นำเสนอเป็นการบันทึกแบบไม่จำกัดเวลา คือทุกๆครั้งที่บัฟเฟอร์ขนาด 2 KB เต็มก็จะนำข้อมูลนั้นไปเก็บในฮาร์ดดิสต์ นั่นคือสามารถเก็บข้อมูลได้จนกระทั่งฮาร์ดดิสต์เต็ม) ระยะหนึ่งซึ่งข้อมูลที่ได้จากการแซมปลิ้งนั้นจะไม่เกินหน่วยความจำที่มีอยู่ (ซึ่งอาจจะเป็นหน่วยความจำแบบ convention หรือ extention) ในกรณีบัฟเฟอร์จะต้องมีขนาดใหญ่พอที่จะเก็บข้อมูลเสียงในระยะเวลาที่ต้องการได้ การบันทึกจะกระทำเมื่อบัฟเฟอร์นั้นเต็ม เช่นเดียวกับการอ่านซึ่งจะเริ่มเมื่อข้อมูลถูกอ่านลงสู่บัฟเฟอร์จนเต็มแล้ว

3. การทดสอบการรับส่งคำสั่งระหว่างชุมสายและระบบตอบรับ

การทดสอบทำโดยให้แชนแนลที่ 1 หรือ 2 ของระบบตอบรับถูกใช้โดยสายนอกแล้วให้สายในทดลองโทรเข้าหาชุมสายเพื่อขอใช้บริการทั้ง 5 แบบ เพื่อที่จะดูว่าโปรแกรมของระบบตอบรับสามารถให้บริการแก่สายภายในในขณะที่ทำงานให้กับสายนอกได้หรือไม่ จากการทดลองพบว่าทำงานได้ตามวัตถุประสงค์

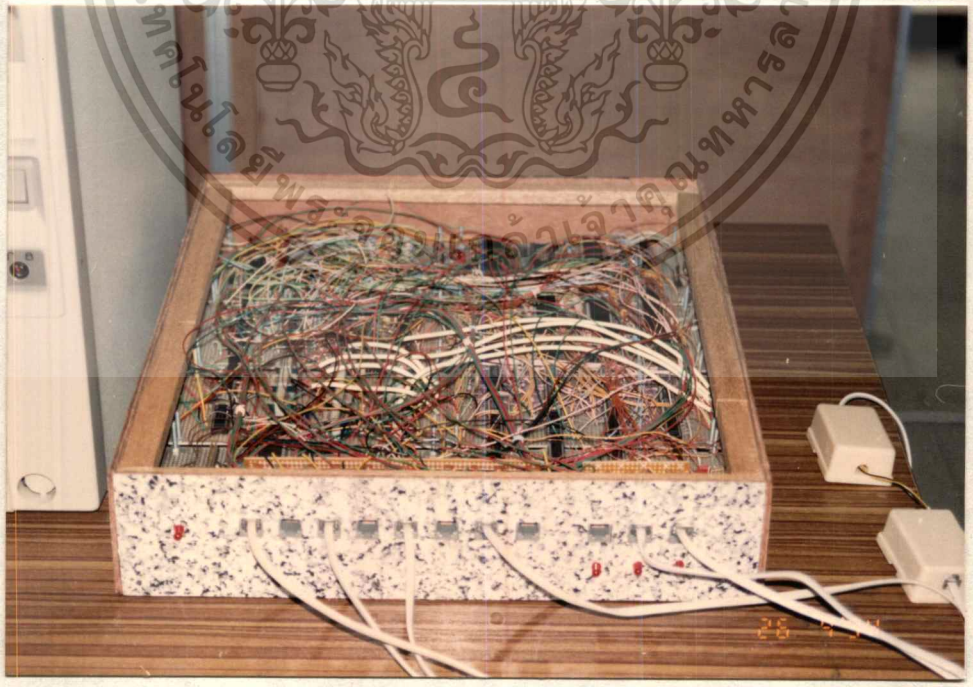
4. การทดสอบการโทรออก เนื่องจากอัลกอริธึมธ์ในการโทรออกจะทำการตรวจสอบว่าได้รับสัญญานสายไม่ว่างหรือเรียกกลับในครั้งแรกหลังจากส่งสัญญาน DTMF ออกไปแล้ว โดยการตรวจสอบว่าได้รับขอบขาขึ้นของสัญญานตอบกลับสองครั้งติดกันห่างกันประมาณ 5 วินาทีหรือไม่ ถ้าใช่จะถือว่าเป็นสัญญานเรียกกลับ แต่หากไม่ใช่จะถือว่าเป็นสัญญานสายไม่ว่าง ซึ่งจะเกิดข้อเสียในกรณีที่ผู้รับยกหูทันทีที่ได้รับสัญญานเรียก ซึ่งระบบจะตีความว่าเป็นสัญญานสายไม่ว่าง

5.4 การทำงานของระบบแจ้งข่าวสาร

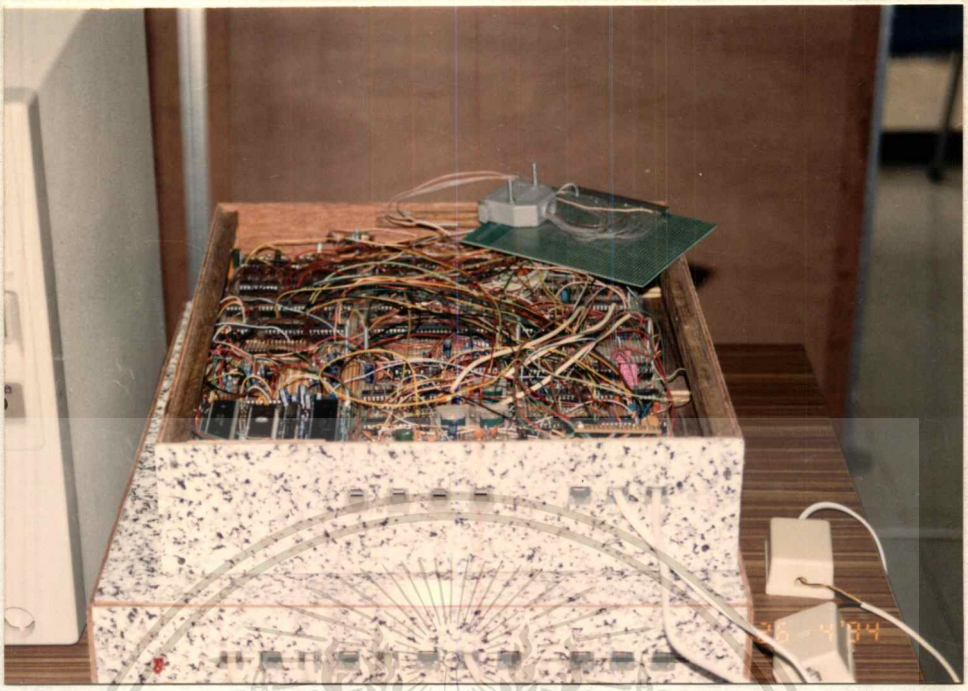
เนื่องจากระบบนี้มีโครงสร้างแบบง่ายๆการทดสอบจึงมีเพียงการต่อไมโครคอมพิวเตอร์ 4 เครื่องเข้ากับระบบซึ่งแต่ละเครื่องจะรันซอฟต์แวร์ Telix อยู่ การรับส่งกระทำที่อัตราบอด 9600 ข้อความที่ปรากฏบนจอขนาด 256 ไบท์ จะปรากฏวนจากคอมพิวเตอร์เครื่องที่ 1 ถึง 4 แล้วกลับมาที่เครื่องที่ 1 อีกครั้ง แสดงการต่อทดลองดังรูปที่ 5.4

5.5 ต้นแบบของชุมสายที่พัฒนา

ต้นแบบจริงที่พัฒนาขึ้นมีสองส่วนคือ ชุมสายโทรศัพท์ และระบบตอบรับโทรศัพท์ โดยใช้ไวร์แล็ปบนแผ่นวงจรเอนกประสงค์

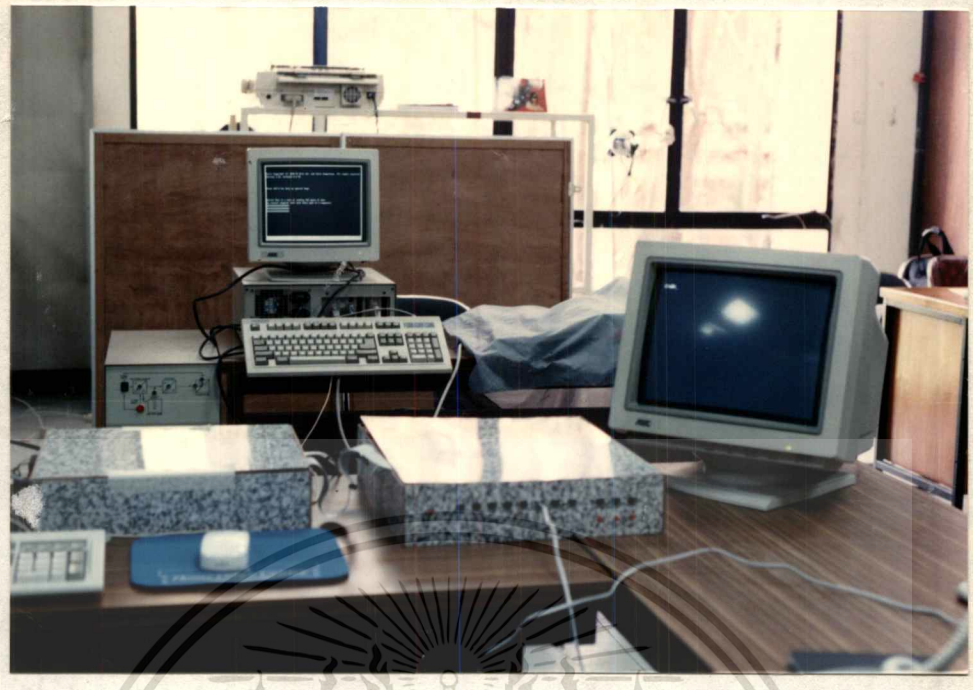


รูปที่ 5.1 ต้นแบบชุมสายโทรศัพท์



รูปที่ 5.2 ต้นแบบระบบตอบรับโทรศัพท์

รูปที่ 5.3 แสดงการต่อใช้งานทั้งสองระบบร่วมกัน



รูปที่ 5.4 การต่อใช้ระบบแจ้งข่าวสาร

บทที่ 6
บทสรุป

งานวิจัยนี้มีวัตถุประสงค์ที่จะพัฒนาชุมสายโทรศัพท์ที่มีระบบตอบรับโทรศัพท์เป็นฟังก์ชันการทำงานหลักของระบบ โดยการสร้างชุมสายโทรศัพท์ที่มีส่วนเพิ่มขยายที่ใช้ในการติดต่อภายในกับระบบตอบรับ โดยการต่อใช้งานระบบตอบรับจะต่อคร่อมสายนอกของชุมสายโทรศัพท์ และใช้สายแพเป็นทางผ่านของการควบคุมระหว่างชุมสายและระบบตอบรับ การออกแบบในลักษณะนี้ช่วยให้ระบบตอบรับเป็นอิสระจากชุมสายซึ่งหมายความว่าเราสามารถนำระบบตอบรับนี้ไปใช้กับชุมสายโทรศัพท์อื่นๆได้โดยไม่จำเป็นต้องใช้กับชุมสายที่พัฒนาขึ้น แต่มีข้อเสียตรงที่ว่าจะต้องเสียสายภายในไปจำนวน 3 สายหากต้องการใช้ระบบตอบรับทั้งสามแชนแนลซึ่งจะทำให้เหลือสายในที่ใช้น้อยลงไป แต่หากใช้กับชุมสายที่พัฒนาขึ้นมาก็จะไม่เสียสายในไป

ในโปรแกรมมอนิเตอร์ของชุมสายโทรศัพท์จะเพิ่มหมายเลขบริการที่ใช้ในการติดต่อกับระบบตอบรับ โดยจะเป็นสะพานส่งคำสั่งจากผู้ใช้ไปยังระบบตอบรับ การพัฒนาระบบตอบรับโทรศัพท์จะใช้ฮาร์ดแวร์เท่าที่จำเป็น และใช้ซอฟต์แวร์ซึ่งพัฒนาบนเครื่องไมโครคอมพิวเตอร์ให้มากที่สุด ซึ่งจะทำให้เกิดข้อดีที่สำคัญสองประการคือหนึ่งทำให้ต้นทุนทางฮาร์ดแวร์ต่ำและใช้อุปกรณ์น้อยซึ่งเป็นผลดีในการผลิตเพื่อนำไปใช้งานจริง สองทำให้การเปลี่ยนแปลงฟังก์ชันการทำงานของระบบตอบรับทำได้ง่ายเพราะสามารถเปลี่ยนแปลงได้จากซอฟต์แวร์โดยตรง

โครงสร้างข้อมูลและอัลกอริธึมที่นำเสนอในวิทยานิพนธ์นี้ได้ครอบคลุมส่วนงานต่างๆที่ควรจะมีสำหรับระบบตอบรับโทรศัพท์ ซึ่งผู้ที่สนใจสามารถนำหลักการและตัดตอนโปรแกรมต้นฉบับไปประยุกต์ใช้ในงานอื่นได้ เช่นหากต้องการโปรแกรมให้ระบบตอบรับทำหน้าที่เป็นระบบการแจ้งข่าวสารอย่างเดียวในทั้ง 3 แชนแนลโดยไม่มีการฝากข้อความ ก็สามารถตัดตอนจากโปรแกรมในส่วนตอบรับสายนอกแล้วอ่านไฟล์ที่ต้องการแจ้งให้ผู้โทรเข้ามาทราบแทน หรือหากต้องการความซับซ้อนยิ่งขึ้นโดยให้ผู้โทรเข้ากดเลขหมายเพื่อเลือกรายการข้อความที่ต้องการทราบก็ทำได้โดยไม่ยาก เพราะการเล่นกลับไฟล์ใดๆก็คือการเขียนข้อมูลที่เกี่ยวข้องลงในตารางสถานะของแชนแนลที่มีสายนอกเข้ามา โดยในขณะเขียนโปรแกรมเราไม่จำเป็นต้องพะวงกับลำดับเหตุการณ์ที่จะเกิดขึ้นในแชนแนลอื่น เพราะรูทีนส่วนการแซมปลิ้งสัญญาณจะคอยควบคุมการบันทึกหรือโหลดเสียงในแต่ละแชนแนลให้ ซึ่งทำให้สะดวกในการพัฒนาโปรแกรม

อย่างไรก็ตามการเขียนซอฟต์แวร์ตามวิธีการที่เสนอมานี้ก็ยังมีข้อยุ่งยากที่เห็นได้ชัดคือต้องเขียนด้วยภาษาแอสแซมบลี้ แต่ข้อยุ่งยากอันนี้สามารถแก้ไขด้วยวิธีการทางซอฟต์แวร์เช่นกัน โดยการสร้างภาษาต้นแบบซึ่งมีรูปแบบ (syntax) ที่ให้ผู้ใช้กำหนดการทำงานของระบบตอบรับด้วยภาษาง่ายๆ และพัฒนาคอมไพเลอร์ที่ทำหน้าที่แปลงภาษาต้นแบบนั้นให้เป็นภาษาแอสแซมบลี้โดยใช้โครงสร้างข้อมูลและอัลกอริธึมที่นำเสนอ ซึ่งเป็นวิธีที่ผู้วิจัยคิดว่าน่าจะเป็นแนวทางที่เหมาะสมในการพัฒนาระบบตอบรับโทรศัพท์ที่สมบูรณ์แบบเพื่อให้ทำงานได้หลากหลาย

เอกสารอ้างอิง

1. การใช้งาน Z-80 , ศูนย์ภาษาคอมพิวเตอร์, พีสิกส์เซ็นเตอร์การพิมพ์

2. คู่มือเทียบเบอร์ไอซี TTL , บริษัทซีเอ็ดยูเคชั่นจำกัด, พ.ศ.2532

3. คู่มือเทียบเบอร์ไอซี CMOS , บริษัทซีเอ็ดยูเคชั่นจำกัด, พ.ศ.2532

4. คู่มือไอซี ซิพรัพพอร์ทและหน่วยความจำ, บริษัทซีเอ็ดยูเคชั่นจำกัด, พ.ศ.2529

5. จรัสพรรณ พันธะ, บุญวัฒน์ อัตชู, ระบบตอบรับโทรศัพท์บนไมโครคอมพิวเตอร์ , การประชุมทางวิชาการทางวิศวกรรมศาสตร์ ครั้งที่ 16, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

6. บัณฑิต จามรภูติ, ฮาร์ดแวร์ไมโครคอมพิวเตอร์ 8088, 80286, 80386 ฉบับปรับปรุงและเพิ่มเติม, บริษัทซีเอ็ดยูเคชั่นจำกัด

7. ดร.บุญวัฒน์ อัตชู, ทฤษฎีและการใช้งานไมโครโปรเซสเซอร์, โครงการตำราคณะวิศวกรรมศาสตร์สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

8. ยืน ภู่วรวรรณ, เทคนิคการประยุกต์และใช้งานไอซีทีทีแอล, บริษัทซีเอ็ดยูเคชั่นจำกัด, พ.ศ.2528

9. ยืน ภู่วรวรรณ, ทฤษฎีและการประยุกต์ไมโครโปรเซสเซอร์ Z-80, บริษัทซีเอ็ดยูเคชั่นจำกัด

10. พัชรี ปรัชญาถาวรกุล, สุชฟ้า ศรีงางาม, เครื่องชุมสายขนาดเล็ก , ปริญญานิพนธ์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, พ.ศ. 2533

11. Microprocessor Databook, MCS-51

12. น.ต. ธวัชชัย เลื่อนฉวี, เทคโนโลยีโทรศัพท์ , บรรเทิงการพิมพ์, พ.ศ.2533Microcontrollers, INTEL

13. DOGAN A. TUGAL, OSMAN TUGAL, Data Transmission , McGrawHill Book Company, 1989, 1982

14. DAVID F. STOUT, MILTON KAUFMAN, Handbook of Operational Amplifier Circuit Design, McGrawHill Book Company, 1976.

15. JOHN UFFENBECK, THE 8086/8088 FAMILY: design, programming, and interfacing ,Prentice-Hall International Editions, 1987.

16. JOHN ANGERMEYER, KEVIN JAEGER, MS-DOS Developer's Guide, The Waite Group Howard W.Sams & Co. 1986.

17. JOHN F. WAKERLY, DIGITAL DESIGN PRINCIPLES AND PRACTICES, Prentice-Hall International Editions, 1990

18. Linear Circuits Amplifiers, Comparators, and Special Functions, Data Book Volume 1, Texas Instruments.

19.. LEWIS C. EGGEBRECHT, INTERFACING TO THE IBM PERSONAL COMPUTER, Howard W. Sams & Co., Inc, 1983.

20. Steven Holzner, ADVANCED ASSEMBLY LANGUAGE on the IBM PC, A Brady Book Published by Prentice Hall Press, 1987.

21. TELECOMMUNICATIONS DEVICE DATA, Sries C, Motorola, Third Printing, 1989.

ภาคผนวก ก โปรแกรมต้นฉบับของชุมสายโทรศัพท์

```
                CPU     "Z80.TBL"
                HOF     "INT8"
;---------- state variable ----------
dialstate:      equ     01h
busystate:      equ     02h
rbstate:        equ     03h
off_dialstate:  equ     04h
calledstate:    equ     05h
callerstate:    equ     06h
conver_caller:  equ     07h
conver_called:  equ     08h
downstate:      equ     09h
holdstate:      equ     0ah
conver_ex:      equ     0bh
use_avmstate:   equ     0ch
ex_calledstate: equ     0dh
avm_callstate:  equ     0fh
telstate:       equ     00h
offtone:        equ     01h
dtmfport:       equ     02h
lineselect:     equ     02h
port1:          equ     00h
port2:          equ     10h
port3:          equ     30h
port4:          equ     50h
lowbyte:        equ     51h
enable:         equ     52h
port5:          equ     70h
port6:          equ     0ch
inrelay:        equ     70h
highbyte:       equ     71h
outrelay:       equ     72h
clrring1:       equ     90h
clrring2:       equ     0b0h
clrring3:       equ     0d0h
clrdtmf1:       equ     0f0h
clrdtmf2:       equ     08h
status:         equ     04h

                org     00h
;*************************************************************
main:           ld      hl, 00h     ; power on delay
st1:            dec     hl
                ld      a, l
                or      h
                jr      nz, st1
                ld      sp, 09ffffh
                call    setup
                call    en_ring
loopline:
                ld      a, (port5C)
                ld      hl, scanline
                or      (hl)
                out     (outrelay), a
                out     (port1+1), a
                ld      (port5C), a
                call    update_count
                call    state                   ; do when line change
                call    external_line
                call    avm_command
                call    delay
                call    dtmf
; scan next group
                ld      a, (port5C)
                and     11111100b
                ld      (port5C), a
```

```
                ld      a, (scanline)
                inc     a
                ld      (scanline), a
                cp      00000100b
                jr      nz, outmain
                ld      a, 00h
                ld      (scanline), a
outmain:
                jr      loopline
;*****************************************************
update_count:   ld      a, 00h
                ld      b, 08h
                ld      hl, count
ser_count:      cp      (hl)
                jr      z, next_count
                inc     (hl)
                ld      a, 06h
                cp      (hl)
                jr      nz, next_count
                ld      a, 00h
                ld      (hl), a
next_count:     inc     hl
                djnz    ser_count
                ret
;-----------------------------------------------------
state:          ld      hl, prestate
                in      a, (telstate)
                ld      (nowstate), a
                ld      d, a            ; save nowstate
                xor     (hl)
                ret     z
                ld      (changestate), a
                ld      (hl), d
                call    do_changing
                ret
;-----------------------------------------
external_line:  in      a, (status)             ; check ex1
                ld      b, a
                and     00000001b
                jr      z, ex2
                push    bc
                call    ex1_in
                pop     bc
ex2:            ld      a, b
                and     00000010b
                jr      z, ex3
                push    bc
                call    ex2_in
                pop     bc
ex3:            ld      a, b
                and     00000100b
                ret     z
                call    ex3_in
                ret

ex1_in:         in      a, (cirring1)
                ld      ix, exuse
                ld      a, 02h
                ld      (ix+0), a
                ld      hl, linestate       ; see if line 1 downstate
                ld      a, (hl)
                cp      downstate
                ret     nz
                ld      c, 00h              ; line 1 is called by external line
                ld      e, ex_calledstate
                call    newstate
```

```
                ld      hl, talk_ex         ; tell ex line that comes in
                ld      a, 00h
                ld      (hl), a
                ld      c, 00h
                call    ringing_on          ; send ring to line 1
                ret

ex2_in:         in      a, (cirring2)
                ld      ix, exuse
                ld      a, 02h
                ld      (ix+1), a
                ld      hl, linestate       ; see if line 1 downstate
                ld      a, (hl)
                cp      downstate
                ret     nz
                ld      c, 00h              ; line 1 is called by external line
                ld      e, ex_calledstate
                call    newstate
                ld      hl, talk_ex         ; tell ex line that comes in
                ld      a, 01h
                ld      (hl), a
                ld      c, 00h
                call    ringing_on          ; send ring to line 1
                ret

ex3_in:         in      a, (cirring3)
                ld      ix, exuse
                ld      a, 02h
                ld      (ix+2), a
                cp      downstate
                ret     nz
                ld      c, 00h              ; line 1 is called by external line
                ld      e, ex_calledstate
                call    newstate
                ld      hl, talk_ex         ; tell ex line that comes in
                ld      a, 02h
                ld      (hl), a
                ld      c, 00h
                call    ringing_on          ; send ring to line 1
                ret
;-----------------------------------------
avm_command:    in      a, (port6+2)
                ld      d, a
                and     11000000b
                cp      01000000b           ; if not transfer command, do nothing
                ret     nz
                ld      a, 1010b ; tell avm that command accepted
                out     (port6), a
                in      a, (status)         ; get internal line
                and     11100000b
                srl     a
                srl     a
                srl     a
                srl     a
                srl     a
                out     (port1+1), a
                ld      e, a
                ld      hl, linestate       ; see if target in downstate
                ld      b, 00h
                ld      c, a
                add     hl, bc
                ld      a, (hl)
                cp      downstate
                jr      z, avm_transfer
                ld      b, 110b     ; not in downstate tell avm
                call    send_command
```

```
                ret
avm_transfer:   ld      a, d                    ; get external line
                and     00110000b
                srl     a
                srl     a
                srl     a
                srl     a
                ld      hl, exuse               ; this ex line is in ex ( 02 )
                ld      b, 00h
                ld      c, a
                add     hl, bc
                push    af                      ; save external line in a
                ld      a, 02h
                ld      (hl), a
                push    de                      ; save internal line in e
                ld      c, e            ; send ringing to in line
                call    ringing_on
                pop     de
                ld      hl, talk_ex     ; tell in line talking with which ex line
                ld      b, 00h
                ld      c, e
                add     hl, bc
                pop     af                      ; restore external line in a
                ld      (hl), a
                ld      c, e                    ; make in line avm call state
                ld      e, avm_callstate
                call    newstate
                call    delay
                ld      a, 1001b  ; tell avm that transfered line free
                out     (port6), a
                call    send_command
                ret
;-----------------------------------
delay:          push    de
                ld      de, 3000h
ttt:            dec     de
                ld      a, d
                or      e
                jr      nz, ttt
                pop     de
                ret
;-------------------------------------------------
dtmf:           in      a, (status)
                and     00001000b
                jp      z, gr2
                in      a, (clrdtmf1)
                in      a, (dtmfport)
                srl     a
                srl     a
                srl     a
                srl     a
                ld      (x), a
                ld      a, (scanline)
                cp      00h
                jp      z, l1
                cp      01h
                jp      z, l3
                cp      02h
                jp      z, l5

l7:             ld      b, 00h
                ld      a, (ptr7)
                ld      c, a
                ld      hl, dtmf7
                add     hl, bc
                push    hl                      ; hl is where to put DTMF
```

```
                ld      b, 00h          ; see if count<>0
                ld      c, 06h
                ld      hl, count
                add     hl, bc
                ld      a, 00h
                cp      (hl)
                jr      z, savelt7
                ld      a, (x)          ; see if new DTMF = last DTMF
                ld      hl, ptr7
                ld      c, (hl)
                dec     c
                ld      hl, dtmf7
                add     hl, bc
                cp      (hl)
                jr      nz, savelt7     ; not equal keep it
                ld      b, 00h          ; equal if count=5, count=1
                ld      c, 06h
                ld      hl, count
                add     hl, bc
                ld      a, 05h
                cp      (hl)
                jr      nz, savelt7
                ld      a, 01h
                ld      (hl), a
                pop     hl
                jp      gr2
savelt7:        pop     hl              ; keep in dtmf array
                ld      a, (x)
                ld      (hl), a
                ld      hl, count       ; count = 1
                ld      b, 00h
                ld      c, 06h
                add     hl, bc
                ld      a, 01h
                ld      (hl), a
                ld      ix, ptr7        ; if first button, off dial
                ld      a, 00h
                cp      (ix+0)
                jr      nz, j7
                ld      b, 04h
                ld      c, 06h
                call    signal
j7:             inc     (ix+0)          ; point to next byte
                ld      ix, dtmf7
                ld      a, 06h
                ld      (y), a
                call    check_number
                jp      gr2

l5:             ld      b, 00h
                ld      a, (ptr5)
                ld      c, a
                ld      hl, dtmf5
                add     hl, bc
                push    hl              ; hl is where to put DTMF
                ld      b, 00h          ; see if count<>0
                ld      c, 04h
                ld      hl, count
                add     hl, bc
                ld      a, 00h
                cp      (hl)
                jr      z, savelt5
                ld      a, (x)          ; see if new DTMF = last DTMF
                ld      hl, ptr5
                ld      c, (hl)
                dec     c
```

```
                ld      hl, dtmf5
                add     hl, bc
                cp      (hl)
                jr      nz, savelt5             ; not equal keep it
                ld      b, 00h                  ; equal if count=5, count=1
                ld      c, 04h
                ld      hl, count
                add     hl, bc
                ld      a, 05h
                cp      (hl)
                jr      nz, savelt5
                ld      a, 01h
                ld      (hl), a
                pop     hl
                jp      gr2
savelt5:        pop     hl                      ; keep in dtmf array
                ld      a, (x)
                ld      (hl), a
                ld      hl, count       ; count = 1
                ld      b, 00h
                ld      c, 04h
                add     hl, bc
                ld      a, 01h
                ld      (hl), a
                ld      ix, ptr5        ; if first button, off dial
                ld      a, 00h
                cp      (ix+0)
                jr      nz, j5
                ld      b, 04h
                ld      c, 04h
                call    signal
j5:             inc     (ix+0)                  ; point to next byte
                ld      ix, dtmf5
                ld      a, 04h
                ld      (y), a
                call    check_number
                  jp    gr2

l3:             ld      b, 00h
                ld      a, (ptr3)
                ld      c, a
                ld      hl, dtmf3
                add     hl, bc
                push    hl                      ;hl is where to put DTMF
                ld      b, 00h                  ; see if count<>0
                ld      c, 02h
                ld      hl, count
                add     hl, bc
                ld      a, 00h
                cp      (hl)
                jr      z, savelt3
                ld      a, (x)                  ; see if new DTMF = last DTMF
                ld      hl, ptr3
                ld      c, (hl)
                dec     c
                ld      hl, dtmf3
                add     hl, bc
                cp      (hl)
                jr      nz, savelt3             ; not equal keep it
                ld      b, 00h                  ; equal if count=5, count=1
                ld      c, 02h
                ld      hl, count
                add     hl, bc
                ld      a, 05h
                cp      (hl)
                jr      nz, savelt3
```

```
                ld      a, 01h
                  ld    (hl), a
                pop     hl
                jp      gr2
savelt3:        pop     hl                      ; keep in dtmf array
                ld      a, (x)
                ld      (hl), a
                ld      hl, count       ; count = 1
                ld      b, 00h
                ld      c, 02h
                add     hl, bc
                ld      a, 01h
                ld      (hl), a
                ld      ix, ptr3        ; if first button, off dial
                ld      a, 00h
                cp      (ix+0)
                jr      nz, j3
                ld      b, 04h
                ld      c, 02h
                  call  signal
j3:             inc     (ix+0)                  ; point to next byte
                ld      ix, dtmf3
                ld      a, 02h
                ld      (y), a
                call    check_number
                  jp    gr2

l1:             ld      b, 00h
                ld      a, (ptr1)
                  ld    c, a
                ld      hl, dtmf1
                  add   hl, bc
                push    hl                      ; hl is where to put DTMF
                ld      b, 00h                  ; see if count<>0
                ld      c, 00h
                ld      hl, count
                  add   hl, bc
                ld      a, 00h
                cp      (hl)
                jr      z, savelt1
                ld      a, (x)                  ; see if new DTMF = last DTMF
                ld      hl, ptr1
                ld      c, (hl)
                dec     c
                ld      hl, dtmf1
                add     hl, bc
                cp      (hl)
                jr      nz, savelt1             ; not equal keep it
                ld      b, 00h                    ; equal if count=5, count=1
                ld      c, 00h
                ld      hl, count
                add     hl, bc
                ld      a, 05h
                cp      (hl)
                jr      nz, savelt1
                ld      a, 01h
                ld      (hl), a
                pop     hl
                jp      gr2
savelt1:        pop     hl                      ; keep in dtmf array
                ld      a, (x)
                ld      (hl), a
                ld      hl, count       ; count = 1
                ld      b, 00h
                ld      c, 00h
                add     hl, bc
```

```
                ld      a, 01h
                ld      (hl), a
                ld      ix, ptr1        ; if first button, off dial
                  ld    a, 00h
                  cp    (ix+0)
                jr      nz, j1
                  ld    b, 04h
                ld      c, 00h
                  call  signal
j1:             inc     (ix+0)                  ; point to next byte
                ld      ix, dtmf1
                ld      a, 00h
                ld      (y), a
                  call  check_number

gr2:            in      a, (status)
                  and   00010000b
                  ret   z
                  in    a, (clrdtmf2)
                  in    a, (dtmfport)
                  and   0fh
                  ld    (x), a
                  ld    a, (scanline)
                  cp    00h
                jp      z, l2
                  cp    01h
                  jr    z, l4
                  cp    02h
                  jr    z, l6

l8:             ld      b, 00h
                  ld    a, (ptr8)
                  ld    c, a
                  ld    hl, dtmf8
                  add   hl, bc
                  ld    a, (x)
                  ld    (hl), a
                ld      ix, ptr8
                ld      a, 00h
                cp      (ix+0)
                jr      nz, j8
                ld      b, 04h
                ld      c, 07h
                call    signal
j8:
                inc     (ix+0)
                ld      ix, dtmf8
                ld      a, 07h
                ld      (y), a
                  call  check_number
                ret
l6:             ld      b, 00h
                  ld    a, (ptr6)
                  ld    c, a
                  ld    hl, dtmf6
                add     hl, bc
                  ld    a, (x)
                  ld    (hl), a
                  ld    ix, ptr6 -
                ld      a, 00h
                  cp    (ix+0)
                jr      nz, j6
                  ld    b, 04h
                ld      c, 05h
                  call  signal
j6:
```

```
                pop     ix
                pop     hl
                ld      a, c
                out     (port1+1), a
                call    cleardtmf       ; ix point to dtmf to be clear
                call    do_button       ; do C for line in (y)
                ret
nextinter:      pop     ix
                pop     hl
                ld      de, 08h
                add     hl, de
                inc     c
                ld      a, 20d
                cp      c
                ret     z
                jr      comp
;-----------------------------------------------
cleardtmf:      ld      b, 08h
                ld      a, 00h
clear:          ld      (ix+0), a
                inc     ix
                djnz    clear
                ld      hl, ptr1
                ld      d, 00h          ; clear ptr
                ld      a, (y)
                ld      e, a
                add     hl, de
                ld      (hl), d
                ret
;-----------------------------------------------
; input C = line called
;         (y) = line call
;-----------------------------------------------
do_button:      ld      a, c
                ld      (x), a                  ; save line called in (x)
                cp      08d
                jp      c, inline
                cp      11d
                jp      c, outline
;                ;-----------------
;                ;       SERVICE
;                ;-----------------
;
;           HOLD LINE BY PRESSING #
;
                cp      11d             ; # pressed
                jp      z, record
                cp      12d             ; 51 pressed
                jp      z, play
                cp      13d             ; 52 pressed
                jp      z, telout
                cp      14d             ; 53 pressed
                jp      z, comein
                cp      15d             ; 54 pressed
                jp      z, go_out
                cp      16d             ; 55 pressed
                jp      z, hold_service
                cp      17d             ; *1 pressed
                jp      z, avm1
                cp      18d             ; *2 pressed
                jp      z, avm2
                ret

hold_service:   ld      hl, linestate   ; see if conver_ex
                ld      b, 00h
                ld      a, (y)
```

```
                ld      c, a
                add     hl, bc
                ld      a, (hl)
                cp      conver_ex
                jr      nz, checknext       ; see if in hold state
                ld      hl, talk_ex         ; hold line talking with
                add     hl, bc
                ld      a, (hl)
                ld      (line), a
                call    hold_line
                ld      a, (line)    ; disconnect in and ext line
                ld      a, (y)
                ld      e, a
                call    disconnect_ex
                ld      e, holdstate        ; make this line hold state
                ld      a, (y)
                ld      c, a
                call    newstate
                ret
checknext:      cp      holdstate           ; if not in hold state do nothing
                ret     nz
                ld      hl, talk_ex         ; unhold line talking with
                add     hl, bc
                ld      a, (hl)
                ld      (line), a
                call    unhold_line
                ld      a, (line)    ; connect in and ext line
                ld      a, (y)
                ld      e, a
                call    connect_ex
                ld      e, conver_ex        ; make this line hold state
                ld      a, (y)
                ld      c, a
                call    newstate
                ret
;
;       SET AVM 1
;
avm1:           ld      a, (y)              ; if not line 1, do nothing
                cp      00h
                ret     nz
                ld      a, 01h              ; ex line 1 use avm
                ld      ix, use_avm
                ld      (ix+0), a
                ret
;
;       SET AVM 2
;
avm2:           ld      a, (y)              ; if not line 1, do nothing
                cp      00h
                ret     nz
                ld      a, 01h              ; ex line 2 use avm
                ld      ix, use_avm
                ld      (ix+1), a
                ret
;
;       LINE IN (Y) WANT TO RECORD
;
record:         ld      b, 01h
                ld      a, (y)
                ld      c, a
                call    send_command
                ret     c                   ; get busy
                ld      a, (y)
                ld      b, a
                inc     b
```

```
                inc     (ix+0)
                ld      ix, dtmf6
                ld      a, 05h
                ld      (y), a
                call    check_number
                ret
l4:             ld      b, 00h
                ld      a, (ptr4)
                  ld    c, a
                  ld    hl, dtmf4
                  add   hl, bc
                  ld    a, (x)
                  ld    (hl), a
                ld      ix, ptr4
                ld      a, 00h
                cp      (ix+0)
                jr      nz, j4
                ld      b, 04h
                ld      c, 03h
                call    signal
j4:
                inc     (ix+0)
                ld      ix, dtmf4
                ld      a, 03h
                ld      (y), a
                  call  check_number
                  ret
l2:             ld      b, 00h
                  ld    a, (ptr2)
                ld      c, a
                  ld    hl, dtmf2
                  add   hl, bc
                  ld    a, (x)
                  ld    (hl), a
                  ld    ix, ptr2
                  ld    a, 00h
                  cp    (ix+0)
                  jr    nz, j2
                  ld    b, 04h
                  ld    c, 01h
                  call  signal
j2:
                  inc   (ix+0)
                ld      ix, dtmf2
                ld      a, 01h
                ld      (y), a
                  call  check_number
                ret
;---------------------------------------
; input IX = dtmf?
;     (y) = line number
; output C = 0-7 internal line found
;        = 8, 9, 10 external line found
;        = 11, 12, 13 service found
check_number:   ld      hl, interline1
                ld      c, 00h
comp:
                  push    hl
                push    ix
                ld      b, 08h
continue:       ld      a, (ix+0)
                  cp    (hl)
                  jr    nz, nextinter
                  inc   ix
                  inc   hl
                  djnz  continue
```

```
                ld      a, 10000000b    ; mask 1 for line that wants recording
distribute:     rlca
                djnz    distribute
                push    af
                call    delay
                call    delay
                call    delay
                call    delay
                call    delay
                pop     af
                out     (enable), a
                ret
;
;       LINE IN (Y) WANT TO PLAY
;
play:           ld      b, 02h
                ld      a, (y)
                ld      c, a
                call    send_command
                ret     c                       ; get busy
                ld      a, (y)
                ld      b, a
                inc     b
                ld      a, 10000000b    ; mask 1 for line that wants playing
distri:         rlca
                djnz    distri
                out     (enable), a
                ret
;
;       LINE IN (Y) WANT TO PHONE OUT
;
telout:         ld      b, 03h
                ld      a, (y)
                ld      c, a
                call    send_command
                ret     c
                ld      a, (y)
                ld      b, a
                inc     b
                ld      a, 10000000b    ; mask 1 for line that wants programe
ditri:          rlca
                djnz    ditri
                out     (enable), a
                ret

go_out:         ld      b, 04h
                ld      a, (y)
                ld      c, a
                call    send_command
                ret

comein:         ld      b, 05h
                ld      a, (y)
                ld      c, a
                call    send_command
                ret
;---------------------------------------
; INPUT: B command line for line in C
;       C line that sent command
;---------------------------------------
send_command:   in      a, (port6+2)    ; see if avm ready
                and     11000000b
                cp      0c0h
                jr      nz, voice_busy
                ld      a, c
                sla     a
```

```
                sla     a
                sla     a
                sla     a
                or      b
                out     (port6), a          ;send it
wait_avm:       in      a, (port6+2)        ;see if avm accepts command
                  and     11000000b
                cp      10000000b           ; void
                jr      z, void_this
                cp      00h
                jr      nz, wait_avm        ; wait until avm accept
                in      a, (status)         ; get line
                srl     a
                srl     a
                srl     a
                srl     a
                srl     a
                cp      c                   ; see if this line
                  jr      nz, wait_avm
                ld      a, 00001001b        ; COMMAND OK!
                  out     (port6), a        ; cancel previous command
                  ld      e, use_avmstate   ; accept, make conver voice state
                  call      newstate
                  and       a               ; clear carry flag
                ret
void_this:      in      a, (status)         ; see if this line void
                srl     a
                srl     a
                srl     a
                srl     a
                srl     a
                cp      c
                jr      nz, wait_avm
voice_busy:     ld      b, 02h      ; send busy tone
                call    signal
                ld      a, 00001001b
                out     (port6), a          ; cancel previous command
                scf
                ret
;--------------------
;  INTERNAL LINE
;--------------------
inline:         ld      hl, linestate       ; see if that line in down state
                ld      b, 00h
                add     hl, bc
                ld      a, (hl)
                cp      downstate
                jr      z, sendrb           ; if so, book and send RB
sendbusy:       ld      b, 02h              ; send busy to itself
                ld      a, (y)
                ld      c, a
                call    signal
                ret
sendrb:         ld      a, calledstate      ; book as called state
                ld      (hl), a
                call    ringing_on          ; send ring to called line
                ld      b, 03h              ; send ring back to itself
                ld      a, (y)
                ld      c, a
                call    signal
                ld      hl, linestate       ; make this line caller state
                ld      b, 00h
                ld      a, (y)
                ld      c, a
                add     hl, bc
                ld      a, callerstate
```

```
                ld      (hl), a
                ld      hl, caller    ; write called line in caller
                ld      b, 00h
                ld      a, (y)
                ld      c, a
                add     hl, bc
                ld      a, (x)
                ld      (hl), a
                ld      hl, called    ; write call line in called
                ld      b, 00h
                ld      a, (x)
                ld      c, a
                add     hl, bc
                ld      a, (y)
                ld      (hl), a
                ret
                ;---------------------
                ;  EXTERNAL LINE
                ;---------------------
outline:        cp      8d
                jr      z, exline1
                cp      9d
                jr      z, exline2
                cp      10d
                jr      z, exline3
                ret
exline1:        ld      a, 0h
                call    check_exuse
                ret
exline2:        ld      a, 1h
                call    check_exuse
                ret
exline3:        ld      a, 2h
                call    check_exuse
                ret
;-------------------------------------
check_exuse:    ld      (line), a
                ld      hl, exuse
                ld      b, 00h
                ld      c, a
                add     hl, bc
                ld      a, (hl)
                cp      00h
                jr      z, book  ; that external line available
                cp      01h
                jr      z, exout  ; that external line is phoned out
                cp      02h
                jr      z, exin   ; line in
                ret

book:           ld      a, 01h              ; book this external line
                ld      (hl), a
                jr      con_external
exout:          ld      b, 02h              ; send busy
                ld      a, (y)
                ld      c, a
                call    signal
                ret
exin:           ld      a, (y)              ; see if line 1
                cp      00h
                jr      nz, exout
con_external:   ld      a, (y)                ; connect in line and out line
                ld      e, a
                ld      a, (line)
                call    connect_ex
                ld      a, (line)    ; close relay
```

```
                call    close_relay
                ld      hl, talk_ex             ; talk_ex (this line) = ex line
                ld      d, 00h
                ld      a, (y)
                ld      e,  a
                add     hl, de
                ld      a, (line)
                ld      (hl), a
converex:       ld      a, (y)                  ; this line conver_ex state
                ld      c, a
                ld      e, conver_ex
                call    newstate
                ret
;---------------------------------------
close_relay:    push    bc
                push    ix
                push    hl
                push    af
                push    de
                ld      a, (line)
                cp      00h
                jr      z, ll_1
                cp      01h
                jr      z, ll_2
                cp      02h
                jr      z, ll_3
                ret
ll_1:           ld      a, 100000b
                jr      put_it
ll_2:           ld      a, 1000000b
                jr      put_it
ll_3:           ld      a, 10000000b
put_it:         ld      hl, port5C
                or      (hl)
                out     (outrelay), a
                ld      (hl), a
                pop     de
                pop     af
                pop     hl
                pop     ix
                pop     bc
                ret
;---------------------------------------
open_relay:     push    bc
                push    ix
                push    hl
                push    af
                push    de
                ld      a, (line)
                cp      00h
                jr      z, lin_1
                cp      01h
                jr      z, lin_2
                cp      02h
                jr      z, lin_3
                ret
lin_1:          ld      a, 100000b
                jr      put_it2
lin_2:          ld      a, 1000000b
                jr      put_it2
lin_3:          ld      a, 10000000b
put_it2:        ld      hl, port5C
                cpla
                and     (hl)
                out     (outrelay), a
                ld      (hl), a
```

```
                pop     de
                pop     af
                pop     hl
                pop     ix
                pop     bc
                ret
;-----------------------------------------------
hold_line:      push    bc
                push    ix
                push    hl
                ld      a, (line)
                cp      00h
                jr      z, line_1
                cp      01h
                jr      z, line_2
                cp      02h
                jr      z, line_3
                ret
line_1:         ld      a, 100b
                jr      put_there
line_2:         ld      a, 1000b
                jr      put_there
line_3:         ld      a, 10000b
put_there:      ld      hl, port5C
                ld      b, (hl)
                or      b
                out     (outrelay), a
                ld      (hl), a
                call    delay
                call    open_relay
                pop     hl
                pop     ix
                pop     bc
                ret
;-----------------------------------------------
unhold_line:    push    bc
                push    ix
                push    hl
                ld      a, (line)
                cp      00h
                jr      z, line1
                cp      01h
                jr      z, line2
                cp      02h
                jr      z, line3
                ret
line1:          ld      a, 100b
                jr      put_there2
line2:          ld      a, 1000b
                jr      put_there2
line3:          ld      a, 10000b
put_there2:     ld      hl, port5C
                cpla
                and     (hl)
                out     (outrelay), a
                ld      (hl), a
                call    close_relay
                pop     hl
                pop     ix
                pop     bc
                ret
;---------------------------------------
en_ring:        ld      a, (port3B)
                or      10000000b
                out     (port3+1), a
                ld      (port3B), a
```

```
                ret

dis_ring:       ld      a, (port3B)
                and     01111111b
                out     (port3+1), a
                ld      (port3B), a
                ret

;---------------------------------------
; c=line number,
; b=1=dial  2=busy      3=ringback;  4=off_dial
signal:         ld      h, b            ; save b
                ld      b, c
                inc     b
                ld      a, 01111111b    ; mask 0 for that line means enable
rotate:         rlca
                djnz    rotate
                ld      d, a            ; d reg mask for enable line = 0
                ld      b, h                    ; return b
                ld      a, b                    ; select signal to send
                cp      01h
                jr      z, dial
                cp      02h
                jr      z, busy
                cp      03h
                jr      z, ringback
                cp      04h
                jr      z, off_dial
                ret
ringback:       ld      a, (port4B)     ; 0
                and     d
                out     (lowbyte), a
                ld      (port4B), a
                ld      a, d            ; 1
                neg
                ld      h, a
                ld      a, (port5B)
                or      h
                out     (highbyte), a
                ld      (port5B), a
                ret
dial:           ld      a, (port4B)   ; 0
                and     d
                out     (lowbyte), a
                ld      (port4B), a
                ld      a, (port5B)   ; 0
                and     d
                out     (highbyte), a
                ld      (port5B), a
                ret
busy:           ld      a, d            ; 1
                cpla
                ld      h, a
                ld      a, (port4B)
                or      h
                out     (lowbyte), a
                ld      (port4B), a
                ld      a, (port5B)   ; 0
                and     d
                out     (highbyte), a
                ld      (port5B), a
                ret
off_dial: ld    a, d            ; 1
                cpla
                ld      h, a
                ld      a, (port4B)
```

```
                or      h
                out     (lowbyte), a
                ld      (port4B), a
                ld      a, d                    ; 1
                cpla
                ld      h, a
                ld      a, (port5B)
                or      h
                out     (highbyte), a
                ld      (port5B), a
                ret
;-----------------------------------------------
ringing_on:     ld      b, c
                inc     b
                ld      a, 01111111b        ; mask 0 for that line means enable
jet1:           rlca
                djnz    jet1
                ld      d, a            ; d reg mask for enable line = 0
                ld      a, (port5A)             ; close relay for this line
                and     d
                out     (inrelay), a
                  ld    (port5A), a
                  ret
;-----------------------------------------------
ringing_off:    ld      b, c
                inc     b
                ld      a, 10000000b        ; mask 1 for that line means disenable
jet2:           rlca
                djnz    jet2
                ld      d, a
                ld      a, (port5A)
                or      d
                out     (inrelay), a
                  ld    (port5A), a
                  ret
;-----------------------------------------------
; subsequent calls must preserve hl, ix, c
;-----------------------------------------------
do_changing:    ld      hl, changestate     ; see which line changes state
                ld      ix, nowstate
                ld      c, 00h
shift:          srl     (hl)
                jr      nc, nextbit     ; no change, see next line
                srl     (ix+0)          ; change!, see in which case?
                jr      c, down         ; now=1, so prev=0, 0-->1
                call    hangup
                jr      exit
down:           call    hangdown
                jr      exit
                call    nc, hangup              ; now=0, so prev=1, 1-->0
nextbit:        srl     (ix+0)
exit:           inc     c
                ld      a, 08h
                cp      c
                jr      nz, shift
                ret
                  ;----------------------
                  ;       HANG DOWN
                  ;----------------------
hangdown:       push    hl
                ld      hl, linestate           ; see down in which case
                ld      b, 00h
                add     hl, bc
                ld      a, (hl)
                cp      use_avmstate
                jr      z, down_voice
```

```
                cp      conver_ex       ; **** only line 1 in this state
                jr      z, down_ex
                cp      conver_caller
                jr      z, read_caller
                cp      conver_called
                jr      z, read_called
                cp      callerstate
                jp      z, endcall
                cp      dialstate
                jp      z, enddial
                cp      holdstate
                jp      z, endhold
                pop     hl
                ret

down_voice:     ld      b, c
                inc     b
                ld      a, 01111111b    ; mask 0 for line that hangdown
dis:            rlca
                djnz    dis
                ld      a, 00h
                out     (enable), a
                ld      b, 00h                  ; tell avm hang down
                call    send_command
                jp      enddial

down_ex:        ld      d, 00b          ; check what external line talking to
                ld      e, c
                ld      hl, talk_ex
                add     hl, de
                ld      a, (hl)
                ld      hl, exuse               ; exuse of talking line = 00
                ld      e, a
                add     hl, de
                ld      (hl), d
                ld      e, c                    ; disconnect
                call    disconnect_ex
                ld      (line), a
                call    open_relay
                jp      enddial

read_caller:    ld      hl, caller
                jr      close_conver
read_called:    ld      hl, called
close_conver:   push    hl                      ; if this line talked with some ex line
                ld      hl, talk_ex
                ld      b, 00h
                add     hl, bc
                ld      a, (hl)
                cp      04h
                jr      z, normalclose
transfex:
                ld      (line), a
                call    unhold_line
                pop     hl                      ; restore called line
                add     hl, bc
                ld      e, (hl)
                call    connect_ex
                ld      hl, talk_ex     ; now external a talking with this partner
                ld      d, 00h
                add     hl, de
                ld      (hl), a
                push    bc
                ld      c, e
                ld      e, conver_ex
                call    newstate
```

```
                pop     bc
                jr      enddial

normalclose:    pop     hl
                ld      b, 00h                  ; get partner in d
                add     hl, bc
                ld      d, (hl)
                ld      e, c
                push    de
                push    bc
                ld      b, 02                   ; send busy to partner
                ld      c, d
                call    signal
                pop     bc
                pop     de
                call    disconnect_line
                jr      enddial

endhold:        ld      hl, talk_ex             ; exuse of line while hold = 00
                ld      b, 00h
                add     hl, bc
                ld      a, (hl)
                ld      hl, exuse
                ld      d, 00h
                ld      e, a
                add     hl, de
                ld      (hl), d
                ld      (line), a   ; unhold and open relay ex line used
                call    unhold_line
                call    open_relay
                jr      enddial

endcall:        ld      b, 04h          ; off signal to this line
                call    signal
                ld      hl, caller      ; make called line down state
                ld      b, 00h
                add     hl, bc
                ld      a, (hl)         ; a = line called
                push    bc
                ld      c, a
                ld      e, downstate
                call    newstate
                call    ringing_off             ; open inrelay of called line
                pop     bc
enddial:        ld      e, downstate        ; make this line downstate
                call    newstate
                ld      hl, talk_ex             ; make talk_ex this line zero
                ld      b, 00h
                add     hl, bc
                ld      a, 04h
                ld      (hl), a
                ld      hl, ptr1    ; make ptr for this line 0
                ld      b, 00h
                add     hl, bc
                ld      a, 00h
                ld      (hl), a
                ld      hl, new     ; new for this line 0
                add     hl, bc
                ld      (hl), a
                ld      hl, dtmf1   ; dtmf for this line 0
                ld      de, 08h
                ld      b, c
inc8:           ld      a, 00h
                cp      c
                jr      z, notinc
                add     hl, de
```

```
                djnz    inc8
notinc:         ld      b, 08h
dtmf00:         ld      (hl), a     ; dtmf=0
                inc     hl
                djnz    dtmf00
                pop     hl
                ret

                ;--------------------
                ;       HANG UP
                ;--------------------
hangup:         push    hl
                ld      hl, linestate       ;see up in which case
                ld      b, 00
                add     hl, bc
                ld      a, (hl)
                cp      avm_callstate
                jp      z, do_avmcall
                cp      ex_calledstate
                jp      z, do_ex_call
                cp      calledstate         ;** called_state
                jp      z, do_called
                cp      downstate
                jp      z, do_down
                pop     hl
                ret

do_avmcall:     call    ringing_off
                ld      hl, talk_ex         ; get external line coming in
                ld      b, 00h
                add     hl, bc
                ld      a, (hl)
                ld      e, c
                ld      (line), a
                call    close_relay
                call    connect_ex
                ld      e, conver_ex        ; this line conver_ex state
                call    newstate
                ld      b, 1000b
                call    send_command
                pop     hl
                ret

; this code will work for line 1 if voice mail not used
; if voice mail used, calling made by avm
do_ex_call:     call    ringing_off
                ld      hl, talk_ex         ; get external line coming in
                ld      b, 00h
                add     hl, bc
                ld      a, (hl)
                ld      e, c
                ld      (line), a
                call    close_relay
                call    connect_ex
                ld      e, conver_ex        ; this line conver_ex state
                call    newstate
                ld      b, 04h              ; quiet signal
                call    signal
                pop     hl
                ret

do_called:      call    ringing_off
                ld      hl, called          ; find caller
                ld      b, 00h
                add     hl, bc
                ld      d, (hl)     ; 1st arg for line connection
```

```
                push    de                  ; save 1st argument
                push    bc
                ld      e, conver_caller . ; make caller conver state
                ld      c, d
                call    newstate
                ld      b, 04h              ; off signal of caller
                call    signal
                pop     bc
                ld      b, 04h              ; off all signal to itself
                call    signal
                ld      e, conver_called    ; make itself conver state
                call    newstate
                pop     de                  ;restore 1st argument
                ld      e, c                ; 2st arg for line connection
                call    connect_line
                pop     hl
                ret

do_down:        ld      b, 01h          ; ** down state, send dial tone
                call    signal
                ld      e, dialstate
                call    newstate
                pop     hl
                ret
;-----------------------------------------------
;  INPUT: D = first line, E = second line
;-----------------------------------------------
connect_line:   push    ix
                push    bc
                ld      b, d                ; converse first line
                inc     b
                ld      a, 10000000b        ; mask 1 for that line
rotate1:  rlca
                djnz    rotate1
                ld      d, a                ; save in d
                ld      b, e                ; converse second line
                inc     b
                ld      a, 10000000b        ; mask 1 for that line
rotate2:        rlca
                djnz    rotate2
                or      d                   ; or first and second
                ld      e, a                ; save in e
                ld      hl, portnumber      ; search for empty column
                ld      ix, portvalue
                ld      b, 05h
                ld      a, 00h
findcol:        cp      (ix+0)
                jr      z, found
                inc     hl
                inc     ix
                djnz    findcol
found:          ld      c, (hl)         ; read port number
                out     (c), e
                ld      (ix+0), e
                pop     bc
                pop     ix
                ret

;-----------------------------------------------
;  INPUT: D = firstline, E = second line
;-----------------------------------------------
disconnect_line:
                push    ix
                push    bc
                ld      b, d                ; converse first line
                inc     b
```

```
                ld      a, 10000000b        ; mask 1 for that line
rotate3:        rlca
                djnz    rotate3
                ld      d, a                ; save in d
                ld      b, e                ; converse second line
                inc     b
                ld      a, 10000000b        ; mask 1 for that line
rotate4:        rlca
                djnz    rotate4
                or      d                   ; or first and second
                ld      hl, portvalue       ; search for pair to disconnect
                ld      ix, portnumber
                ld      b, 05h
findcol1:       cp      (hl)
                jr      z, found1
                inc     hl
                inc     ix
                djnz    findcol1
found1:         ld      e, 00h
                ld      c, (ix+0)   ; read port number
                out     (c), e
                ld      (hl), e
                pop     bc
                pop     ix
                ret
;-------------------------------------
newstate:       ld      hl, linestate
                ld      b, 00h
                add     hl, bc
                ld      (hl), e
                ret

;-------------------------------------
setup:          ld      a, 10011001b ; port1 A in, B out, C high in, C low out
                out     (port1+3), a
                ld      a, 10000000b    ; port 2 to 5 all out
                out     (port2+3), a
                out     (port3+3), a
                out     (port4+3), a
                out     (port5+3), a
                ld      a, 10001000b
                out     (port6+3), a
                ld      a, 00h                      ; reset port 2 to 5 all 0
                out     (port1+1), a
                out     (port2), a
                out     (port3), a
                out     (port4), a
                ld      a, 0ffh
                out     (port5), a
                ld      a, 00001001b        ; tell avm do nothing
                out     (port6), a
                ld      a, 00h
                out     (port2+1), a
                out     (port3+1), a
                ld      a, 0ffh
                out     (port4+1), a
                out     (port5+1), a
                ld      a, 0ffh
                out     (port6+1), a
                ld      a, 00h
                out     (port2+2), a
                out     (port3+2), a
                out     (port4+2), a            ; enable port
                ld      a, 00h
                out     (port5+2), a
                ld      a, 0ffh     ; all ex3 0fh
```

```
                out     (port6+2), a
                ld      a, 00h
                ld      (port2A), a
                ld      (port2B), a
                ld      (port2C), a
                ld      (port3A), a
                ld      (port3B), a
                ld      (port3C), a
                ld      (port4A), a
                ld      (port4B), a
                ld      (port4C), a
                ld      a, 0ffh
                ld      (port5A), a
                ld      a, 00h
                ld      (port5B), a
                ld      (port5C), a
                ld      (port6A), a
                ld      a, 0ffh
                ld      (port6B), a
                ld      (port6C), a
                ld      a, 00h
                ld      (scanline), a   ; start get dtmf from line 0
                in      a, (clrring1)       ; clear all interrupts
                in      a, (clrring2)
                in      a, (clrring3)
                in      a, (clrdtmf1)
                in      a, (clrdtmf2)
                ld      a, 0ffh   ; assume all lines hang down
                ld      (prestate), a
                ld      hl, linestate   ; assume all lines in downstate
                ld      b, 08h
                ld      a, downstate
linezero:       ld      (hl), a
                inc     hl
                djnz    linezero
                ld      hl, ptr1        ; make all ptr* zero
                ld      a, 00h
                ld      b, 08h
zero:           ld      (hl), a
                inc     hl
                djnz    zero
                ld      hl, new         ; make all new zero
                ld      a, 00h
                ld      b, 08h
zero1:          ld      (hl), a
                inc     hl
                djnz    zero1
                ld      hl, dtmf1       ; make all dtmf zero
                ld      a, 00h
                ld      b, 64d
zero2:          ld      (hl), a
                inc     hl
                djnz    zero2
                ld      hl, portnumber      ; init port number
                ld      a, 10h
                ld      (hl), a
                inc     hl
                ld      a, 11h
                ld      (hl), a
                inc     hl
                ld      a, 12h
                ld      (hl), a
                inc     hl
                ld      a, 30h
                ld      (hl), a
                inc     hl
```

```
                ld      a, 50h
                ld      (hl), a
                ld      b, 05h          ; init all port value zero
                ld      hl, portvalue
                ld      a, 00h
zero4:          ld      (hl), a
                inc     hl
                djnz    zero4
                jp      linedata
data:           dfb     02h, 01h, 00h, 00h, 00h, 00h, 00h, 00h
                dfb     02h, 02h, 00h, 00h, 00h, 00h, 00h, 00h
                dfb     02h, 03h, 00h, 00h, 00h, 00h, 00h, 00h
                dfb     02h, 04h, 00h, 00h, 00h, 00h, 00h, 00h
                dfb     02h, 05h, 00h, 00h, 00h, 00h, 00h, 00h
                dfb     02h, 06h, 00h, 00h, 00h, 00h, 00h, 00h
                dfb     02h, 07h, 00h, 00h, 00h, 00h, 00h, 00h
                dfb     02h, 08h, 00h, 00h, 00h, 00h, 00h, 00h
                dfb     09h, 01h, 00h, 00h, 00h, 00h, 00h, 00h
                dfb     09h, 02h, 00h, 00h, 00h, 00h, 00h, 00h
                dfb     09h, 03h, 00h, 00h, 00h, 00h, 00h, 00h
                dfb     05h, 01h, 00h, 00h, 00h, 00h, 00h, 00h      ; #
                dfb     05h, 02h, 00h, 00h, 00h, 00h, 00h, 00h      ; *1
                dfb     05h, 03h, 00h, 00h, 00h, 00h, 00h, 00h      ; *2
                dfb     05h, 04h, 00h, 00h, 00h, 00h, 00h, 00h
                dfb     05h, 05h, 00h, 00h, 00h, 00h, 00h, 00h
                dfb     0Ch, 00h, 00h, 00h, 00h, 00h, 00h, 00h
                dfb     0Bh, 01h, 00h, 00h, 00h, 00h, 00h, 00h
                dfb     0Bh, 02h, 00h, 00h, 00h, 00h, 00h, 00h
                dfb     0Bh, 03h, 00h, 00h, 00h, 00h, 00h, 00h
linedata:       ld      b, 160d
                ld      hl, data
                ld      ix, interline1
putdata:        ld      a, (hl)
                ld      (ix+0), a
                inc     hl
                inc     ix
                djnz    putdata
                ld      a, 00h
clrtalk:        ld      (talking), a
clrexuse:       ld      ix, exuse
                ld      (ix+0), a
                ld      (ix+1), a
                ld      (ix+2), a
                ld      a, 04h
                ld      b, 08h
                ld      hl, talk_ex
clrtalk_ex:     ld      (hl), a
                djnz    clrtalk_ex
clruse_avm:     ld      a, 00h
                ld      ix, use_avm
                ld      (ix+0), a
                ld      (ix+1), a
                ld      (ix+2), a
                ld      (voice_use), a
                ld      b, 08h
                ld      hl, channel_use
clrchannel:     ld      (hl), a
                inc     hl
                djnz    clrchannel
                ld      b, 08h
                ld      hl, count
clrcount:ld     (hl), a
                inc     hl
                ret
;---------------------------------------------------------
; INPUT: a = external line 0, 1, 2
```

```
;           e = internal line 0..7
connect_ex:     push    af
                push    de
                cp      00h
                jr      z, exterline1
                cp      01h
                jr      z, exterline2
                cp      02h
                jr      z, exterline3
                pop     de
                pop     af
                ret
exterline1:     ld      a, (port6B)
                and     11110000b
                sla     e
                add     a, e
                out     (port6+1), a
                ld      (port6B), a
                pop     de
                pop     af
                ret
exterline2:     ld      a, (port6B)
                and     00001111b
                sla     e
                sla     e
                sla     e
                sla     e
                sla     e
                add     a, e
                out     (port6+1), a
                ld      (port6B), a
                pop     de
                pop     af
                ret
exterline3:     ld      a, (port6C)
                and     11110000b
                sla     e
                add     a, e
                out     (port6+2), a
                ld      (port6B), a
                pop     de
                  pop     af
                ret
;-------------------------------------------------------
; INPUT: a = external line 0, 1, 2
;           e = internal line 0..7
disconnect_ex:  push    af
                cp      00h
                jr      z, exteline1
                cp      01h
                jr      z, exteline2
                cp      02h
                jr      z, exteline3
                pop     af
                ret
exteline1:      ld      a, (port6B)
                or      00000001b
                out     (port6+1), a
                ld      (port6B), a
                pop     af
                ret
exteline2:      ld      a, (port6B)
                or      00010000b
                out     (port6+1), a
                ld      (port6B), a
                pop     af
```

```
                ret
exteline3:      ld      a, (port6C)
                or      00000001b
                out     (port6+2), a
                ld      (port6B), a
                pop     af
                ret

                org     9000h
port2A:  dfb    00h     ; sw line 1
port2B:  dfb    00h     ; sw line 2
port2C:  dfb    00h     ; sw line 3
port3A:  dfb    00h     ; sw line 4
port3B:  dfb    00h     ; bit 7 enable/disable ringing
port3C:  dfb    00h
port4A:  dfb    00h
port4B:  dfb    00h     ; low byte
port4C:  dfb    00h     ; enable
port5A:  dfb    00h     ; inrelay
port5B:  dfb    00h     ; high byte
port5C:  dfb    00h     ; ....1111  gate ring, ex3, ex2, ex1
port6A:  dfb    00h
port6B:  dfb    00h
port6C:  dfb    00h
linestate:      dfb     00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h
prestate:       dfb     11111111b
nowstate:       dfb     00h
changestate:    dfb     00000000b
scanline:       dfb     00h
dtmf1:          dfb     00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h
dtmf2:          dfb     00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h
dtmf3:          dfb     00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h
dtmf4:          dfb     00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h
dtmf5:          dfb     00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h
dtmf6:          dfb     00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h
dtmf7:          dfb     00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h
dtmf8:          dfb     00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h
ptr1:           dfb     00h
ptr2:           dfb     00h
ptr3:           dfb     00h
ptr4:           dfb     00h
ptr5:           dfb     00h
ptr6:           dfb     00h
ptr7:           dfb     00h
ptr8:           dfb     00h
new:
new1:           dfb     00h
new2:           dfb     00h
new3:           dfb     00h
new4:           dfb     00h
new5:           dfb     00h
new6:           dfb     00h
new7:           dfb     00h
new8:           dfb     00h
old1:           dfb     00h
old2:           dfb     00h
old3:           dfb     00h
old4:           dfb     00h
old5:           dfb     00h
old6:           dfb     00h
old7:           dfb     00h
old8:           dfb     00h
count:          dfb     00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h

x:              dfb     00h         ; x, y temporary storage
y:              dfb     00h
```

```
;--------- Line number ------------
interline1:     dfb     02h, 01h, 00h, 00h, 00h, 00h, 00h, 00h
interline2:     dfb     02h, 02h, 00h, 00h, 00h, 00h, 00h, 00h
interline3:     dfb     02h, 03h, 00h, 00h, 00h, 00h, 00h, 00h
interline4:     dfb     02h, 04h, 00h, 00h, 00h, 00h, 00h, 00h
interline5:     dfb     02h, 05h, 00h, 00h, 00h, 00h, 00h, 00h
interline6:     dfb     02h, 06h, 00h, 00h, 00h, 00h, 00h, 00h
interline7:     dfb     02h, 07h, 00h, 00h, 00h, 00h, 00h, 00h
interline8:     dfb     02h, 08h, 00h, 00h, 00h, 00h, 00h, 00h
outline1:       dfb     09h, 01h, 00h, 00h, 00h, 00h, 00h, 00h
outline2:       dfb     09h, 02h, 00h, 00h, 00h, 00h, 00h, 00h
outline3:       dfb     09h, 03h, 00h, 00h, 00h, 00h, 00h, 00h
hold:           dfb     0Ch, 00h, 00h, 00h, 00h, 00h, 00h, 00h
transfer:       dfb     0Bh, 00h, 00h, 00h, 00h, 00h, 00h, 00h
callpartner:    dfb     05h, 03h, 00h, 00h, 00h, 00h, 00h, 00h
confer:         dfb     05h, 04h, 00h, 00h, 00h, 00h, 00h, 00h
split:          dfb     05h, 05h, 00h, 00h, 00h, 00h, 00h, 00h
ser1:           dfb     05h, 06h, 00h, 00h, 00h, 00h, 00h, 00h
ser2:           dfb     05h, 07h, 00h, 00h, 00h, 00h, 00h, 00h
ser3:           dfb     05h, 08h, 00h, 00h, 00h, 00h, 00h, 00h
ser4:           dfb     05h, 09h, 00h, 00h, 00h, 00h, 00h, 00h
called:         dfb     00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h
caller:         dfb     00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h
portnumber:     dfb     10h, 11h, 12h, 30h, 50h
portvalue:      dfb     00h, 00h, 00h, 00h, 00h
; external line interface
talking:        dfb     00h
exuse:          dfb     00h, 00h, 00h
line:           dfb     00b
talk_ex:        dfb     00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h
; voice mail interface
use_avm:        dfb     00h
voice_use:      dfb     00h
channel_use:    dfb     00h, 00h, 00h, 00h, 00h, 00h, 00h, 00h
                end
```

ภาคผนวก ข โปรแกรมต้นฉบับของระบบตอบรับโทรศัพท์

```
cr              equ     0dh
lf              equ     0ah
recsize         equ     2048d
rec             equ     01h
ply             equ     02h
isin            equ     03h
isout           equ     04h
waitrec         equ     05h
phoneout        equ     06h
recordphone     equ     07h             ; record for phone out
fnum            equ     5d
time1           equ     10d
time2           equ     10d

data        segment
en_ring1 db   00h
en_ring2 db   00h
timedo1         dw    0000h
timedo2 dw    0000h
count1          db    00h
count2          db    00h
p301w           db    00
intro1          db    'c:\ja\intro1', 0
intro2          db    'c:\ja\intro2', 0
recph           db    'c:\ja\recp', 0
recphone        db    00h
dtmf1           db    3 dup(00)
dtmf2           db    3 dup(00)
dtmf3           db    3 dup(00)
delay_time      dw    0000
flr_wait1 dw    0000h
sec_wait1       dw    0000h
flr_wait2 dw    0000h
sec_wait2       dw    0000h
flr_wait_flag1  db    00h
sec_wait_flag1  db    00h
flr_wait_flag2  db    00h
sec_wait_flag2  db    00h
ptr1            db    00h
ptr2            db    00h
ptr3            db    00h
tone            label word
line            db    02h, 01h
                db    02h, 02h
                db    02h, 03h
                db    02h, 04h
                db    02h, 05h
                db    02h, 06h
                db    02h, 07h
                db    02h, 08h

track           dw    0000h
low_state       db    00h               ; set up with any value except 01h
prog_line       db    0ffh
num_ptr         dw    0000h
chk_tone        db    00h
time_flag       db    00h
temp            db    ?
chk_sig         dw    0000h
rb_busy dw    0000h
prog            db    'c:\ja\prog', 0
pho2            db    00h
afterclose      db    00h
sound           db    00h, 41h, 42h, 43h, 44h, 45h, 46h, 47h, 80h, 88h
buttoncode      db    00h, 01h, 02h, 03h, 04h, 05h, 06h, 07h, 08h, 09h, 00h
```

```
lines       struc
                    namelines       db      fnum*3 dup (00)
                    pointer         dw      0000h
                    mess            db      3 dup (00)
                    p_file          db      3 dup (00)
                    p1_time         db      0ffh        ; minite
                    p2_time         db      0ffh        ; hour
                    number          db      10 dup ('#')
                    statlines       db      isin        ; rec, ply, isin, isout
lines       ends
linesvar    lines   8 dup ( <> )
linessize   equ       size linesvar/8
linesptr            db          00
                    db          linessize
                    db          2*linessize
                    db          3*linessize
                    db          4*linessize
                    db          5*linessize
                    db          6*linessize
                    db          7*linessize

state       struc
                    statstate       db      00
                    namestate       db      3 dup (00)
                    handstate       dw      0000h
                    buffer          db      recsize dup (00)
                    index           dw      0000h
                    del             db      00
                    count           db      00
                    time            db      30
                    misc            db      00
                    linestate       dw      0000h
state       ends
statevar    state           3 dup (<>)
statesize   equ             size statevar/3

;^^^^^^^^ error message ^^^^^^^^^^
error1              db          'can t create file for voice mail 1$', 13, 10, '$'
error2              db          'can t create file for voice mail 2$', 13, 10, '$'
error3              db          'can t open intro file 1$', 13, 10, '$'
error4              db          'can t open intro file 2$', 13, 10, '$'
error5              db          'can t open file for message 1$', 13, 10, '$'
error6              db          'can t open file for message 2$', 13, 10, '$'
error7              db          'invalid file handle for closing in hangdown$', 13, 10, '$'
error8              db          'can t delete file$'
error9              db          'can t create file for recording$', 13, 10, '$'
error10             db          'can t open file for playing$', 13, 10, '$'
error11             db          'write file ch1 error!$', 13, 10, '$'
error12             db          'invalid handle in closing record file ch1$', 13, 10, '$'
error13             db          'read file ch1 error!$', 13, 10, '$'
error14             db          'invalid handle in closing play file ch1$', 13, 10, '$'
error15             db          'delete file error after playing ch1$', 13, 10, '$'
error16             db          'write file ch2 error!$', 13, 10, '$'
error17             db          'invalid handle in closing record file ch2$', 13, 10, '$'
error18             db          'read file ch2 error!$', 13, 10, '$'
error19             db          'invalid handle in closing play file ch2$', 13, 10, '$'
error20             db          'delete file error after playing ch2$', 13, 10, '$'
error21             db          'write file ch3 error!$', 13, 10, '$'
error22             db          'invalid handle in closing record file ch3$', 13, 10, '$'
error23             db          'read file ch3 error!$', 13, 10, '$'
error24             db          'invalid handle in closing play file ch3$', 13, 10, '$'
error25             db          'delete file error after playing ch3$', 13, 10, '$'
error26             db          'can t open message for programme phone out$', 13, 10
error27             db          'can t open message in phone out$', 13, 10
error28             db          'can t open record message for phone out$', 13, 10
error29             db          'can t create record file for phone out$', 13, 10
```

```
error30         db      'can t close file after phone out$', 13, 10
error31         db      'can t delete file after phone out$', 13, 10
error32         db      'can t close file after phone rec$', 13, 10
data    ends

code            segment
                assume  cs:code
                assume  ds:data

main            proc    far
start:
                mov     ax, data    ; make DS = ES = data
                mov     ds, ax
                mov     ax, ds
                mov     es, ax

;------------ SET UP file name ---------------
                mov     dx, 307h
                mov     al, 00h
                out     dx, al

                mov     si, 00h
                mov     al, 00h     ; al counts fnum
                mov     bx, 00h
                mov     dl, 49d     ; dl starts from '1'
                mov     dh, 65d     ; dh starts from 'A'
                mov     cl, 00h     ; cl counts line
setup:          mov     byte ptr linesvar [si][bx], dl
                inc     bx
                mov     byte ptr linesvar [si][bx], dh
                inc     bx
                mov     byte ptr linesvar [si][bx], 00h
                inc     bx
                inc     al
                cmp     al, fnum
                jz      nextline
                inc     dh
                jmp     setup
nextline: mov   al, 00h
                add     si, linessize
                inc     dl                  ; next line
                mov     dh, 65d     ; run from 'A'
                mov     bx, 00h
                inc     cl
                cmp     cl, 08h
                jz      finish
                jmp     setup
finish:
                mov     al, 0ffh    ; tell PBX ready to get command
                mov     dx, 306h
                out     dx, al

                mov     dx, 301h            ; open all relays
                mov     al, 00h
                out     dx, al
; setup file name of message to tell caller
                mov     word ptr linesvar.mess, 314dh                   ; 'm1'
                mov     word ptr linesvar [linessize].mess, 324dh   ; 'm2'
                mov     word ptr linesvar [2*linessize].mess, 334dh  ; 'm3'
                mov     word ptr linesvar [3*linessize].mess, 344dh  ; 'm4'
                mov     word ptr linesvar [4*linessize].mess, 354dh  ; 'm5'
                mov     word ptr linesvar [5*linessize].mess, 364dh  ; 'm6'
                mov     word ptr linesvar [6*linessize].mess, 374dh  ; 'm7'
                mov     word ptr linesvar [7*linessize].mess, 384dh  ; 'm8'
; setup file name of message to phone out
                mov     word ptr linesvar.p_file, 3150h             ; 'p1'
```

```
                mov     word ptr linesvar [linessize].p_file, 3250h    ; 'p2'
                mov     word ptr linesvar [2*linessize].p_file, 3350h  ; 'p3'
                mov     word ptr linesvar [3*linessize].p_file, 3450h  ; 'p4'
                mov     word ptr linesvar [4*linessize].p_file, 3550h  ; 'p5'
                mov     word ptr linesvar [5*linessize].p_file, 3650h  ; 'p6'
                mov     word ptr linesvar [6*linessize].p_file, 3750h  ; 'p7'
                mov     word ptr linesvar [7*linessize].p_file, 3850h  ; 'p8'
; clear all flag
                mov     al, 00h
                mov     dx, 304h            ; clear dtmf1
                out     dx, al
                inc     dx                  ; clear dtmf2
                out     dx, al
                mov     dx, 307h            ; clear dtmf3
                in      al, dx

;------------ START PROGRAME LOOP ---------------
clock:          mov     dx, 302h            ; read 2 kHz clock
again1:         in      al, dx
                and     al, 01h
                jnz     again1   ; again if 1 found
again2:         in      al, dx
                and     al, 01h
                jz      again2          ; again if 0 found
;-----------------------------------------
; time out before accepting ring on line 1
;-----------------------------------------
rrr1:           cmp     count1, 01h
                jnz     rrr2
                inc     timedo1
                cmp     timedo1, 2500d
                jnz     rrr2
                mov     en_ring1, 01h
                mov     timedo1, 00h
                mov     count1, 00h
                jmp     do_ring1
;-----------------------------------------
; time out before accepting ring on line 2
;-----------------------------------------
rrr2:           cmp     count2, 01h
                jnz     ttt1
                inc     timedo2
                cmp     timedo2, 2500d
                jnz     ttt1
                mov     en_ring2, 01h
                mov     timedo2, 00h
                mov     count2, 00h
                jmp     do_ring2
;-----------------------------
; time out for line 1
;-----------------------------
ttt1:           cmp     fir_wait_flag1, 01h
                jnz     ttt2
                cmp     sec_wait_flag1, 01h
                jz      pq
                inc     fir_wait1
                cmp     fir_wait1, 0ffffh
                jnz     ttt2
                mov     sec_wait_flag1, 01h
pq:             inc     sec_wait1
                cmp     sec_wait1, 65535d
                jnz     ttt2
                mov     fir_wait_flag1, 00h
                mov     sec_wait_flag1, 00h
                mov     fir_wait1, 00h
                mov     sec_wait1, 00h
```

```
                mov     dx, 301h
                and     p301w, 11111110b
                mov     al, p301w
                out     dx, al
;---------------------------
; time out for line 2
;---------------------------
ttt2:           cmp     flr_wait_flag2, 01h
                jnz     dt3
                cmp     sec_wait_flag2, 01h
                jz      pq2
                inc     flr_wait2
                cmp     flr_wait2, 0ffffh
                jnz     dt3
                mov     sec_wait_flag2, 01h
pq2:            inc     sec_wait2
                cmp     sec_wait2, 65535d
                jnz     dt3
                mov     flr_wait_flag2, 00h
                mov     sec_wait_flag2, 00h
                mov     flr_wait2, 00h
                mov     sec_wait2, 00h
                mov     dx, 301h
                and     p301w, 11111011b
                mov     al, p301w
                out     dx, al

dt3:            jmp     do_dtmf3

;---------------------------------------
; Check user break
;---------------------------------------
check_break:    mov     ah, 0bh
                int     21h
                cmp     al, 0ffh
                jnz     line_command
                jmp     stop
;---------------------------------------
; check hardware ringing on line 1 and 2
;---------------------------------------
line_command:   mov     dx, 302h
                in      al, dx
                test    al, 00000010b   ; see if line 1 rings
                jz      ri2
                mov     count1, 01h
                mov     dx, 304h        ; clear ring1
                out     dx, al
                jmp     phone_command
ri2:            test    al, 00000100b   ; see if line 2 rings
                jz      checkdtmf
                mov     count2, 01h
                mov     dx, 305h        ; clear ring2
                out     dx, al
                jmp     phone_command
;---------------------------------
; check DTMF on line 1 and 2
;---------------------------------
checkdtmf:      test    al, 00010000b   ; see if a button pushed on line 1
                jz      d2
                jmp     do_dtmf1
d2:             test    al, 00100000b   ; see if a button pushed on line 2
                jz      phone_command
                jmp     do_dtmf2
;------------- CHECK COMMAND ----------------
;  command from internal line PABX
;--------------------------------------------
```

```
phone_command: mov     dx, 306h
               in      al, dx
               mov     ah, al      ; save command in ah
               and     al, 0fh
               cmp     al, 00h
               jz      hang_down
               cmp     al, 01h
               jnz     pl
               jmp     recording
pl:            cmp     al, 02h
               jnz     ph
               jmp     playing
ph:            cmp     al, 03h
               jnz     is_in
               jmp     phone_out
is_in:         cmp     al, 04
               jnz     is_out
               jmp     in_office
is_out:        cmp     al, 05
               jnz     cancel
               jmp     out_office
cancel:        mov     al, 0fh       ; tell PBX ready to get command
               out     dx, al
               jmp     checkstate
;--------------- HANG DOWN ----------------
hang_down:     mov     al, ah
               mov     dx, 306h          ; tell PBX that command accepted
               and     al, 01110000b
               out     dx, al
               mov     cl, 4
               shr     al, cl       ; see hanging down in which process
               mov     ah, 00h
               mov     bx, ax
               mov     bl, linesptr [bx]
               mov     bh, 00h
               mov     si, bx
               cmp     byte ptr linesvar [si].statlines, rec
               jz      end_rec
               cmp     byte ptr linesvar [si].statlines, ply
               jz      end_ply
               cmp     byte ptr linesvar [si].statlines, phoneout
               jnz     op
               jmp     end_phoneout
op:            cmp     byte ptr linesvar [si].statlines, recordphone
               jnz     sdh
               jmp     end_recphone
sdh:           jmp     clock

end_rec:       mov     byte ptr linesvar [si].statlines, isin
               mov     byte ptr statevar [2*statesize].statstate, 00
               mov     byte ptr statevar [2*statesize].misc, 00h
               cmp     byte ptr statevar [2*statesize].del, 01h
               jz      e1                    ; file has been closed
               mov     ah, 3eh   ; close recording file
               mov     bx, word ptr statevar [2*statesize].handstate
               int     21h
               jnc     e1
               lea     dx, error7
               call    write
               jmp     stop
e1:            jmp     clock

end_ply:       mov     byte ptr linesvar [si].statlines, isin
               mov     byte ptr statevar [2*statesize].statstate, 00
               mov     byte ptr statevar [2*statesize].misc, 00h
               cmp     byte ptr statevar [2*statesize].del, 02h
```

```
                jz      e3
                mov     ah, 3eh   ; close playing file
                mov     bx, word ptr statevar [2*statesize].handstate
                int     21h
                jnc     e2
                lea     dx, error7
                call    write
                jmp     stop
e2:             cmp     byte ptr statevar [2*statesize].del, 00h
                jz      e3
                cmp     byte ptr statevar [2*statesize].del, 01h
                jz      e5
                jmp     clock
e5:             mov     ah, 41h   ; delete playing file
                lea     dx, word ptr statevar [2*statesize].namestate
                int     21h
                jnc     e3
                lea     dx, error8
                call    write
                jmp     stop
e3:             jmp     clock

end_phoneout:   mov     byte ptr linesvar [si].statlines, isin
                mov     byte ptr statevar [2*statesize].statstate, 00
                mov     chk_tone, 00h
                mov     num_ptr, 00h
                mov     byte ptr linesvar [si].p1_time, 00h
                mov     byte ptr linesvar [si].p2_time, 00h
                mov     prog_line, 0ffh
                jmp     clock

end_recphone:   mov     byte ptr linesvar [si].statlines, isin
                mov     byte ptr statevar [2*statesize].statstate, 00
                mov     chk_tone, 00h
                mov     time_flag, 01h
                mov     num_ptr, 00h
                mov     prog_line, 0ffh
                mov     byte ptr statevar [2*statesize].misc, 00h
                cmp     byte ptr statevar [2*statesize].del, 01h
                jz      er3               ; file has been closed
                mov     ah, 3eh   ; close recording file
                mov     bx, word ptr statevar [2*statesize].handstate
                int     21h
                jnc     er3
                lea     dx, error32
                call    write
                jmp     stop
er3:            jmp     clock

;-------------- RECORDING ------------------
recording:      mov     al, ah
                cmp     byte ptr statevar [2*statesize].statstate, 00h
                jnz     no_chan
                and     al, 01110000b     ; tell PBX that command accepted
                mov     dx, 306h
                out     dx, al
                mov     cl, 4             ; find the line requesting
                shr     al, cl
                mov     ah, 00h
                mov     bx, ax            ; bx = line number
                mov     bl, linesptr [bx]
                mov     bh, 00h
                mov     si, bx
                mov     ah, 3ch   ; create file
                mov     cx, 00h
                lea     dx, linesvar [si].mess
```

```
                int     21h
                jnc     l1
                lea     dx, error9
                call    write
                jmp     stop
l1:             mov     byte ptr linesvar [si].statlines, rec
                mov     byte ptr statevar [2*statesize].statstate, rec
                mov     word ptr statevar [2*statesize].handstate, ax
                mov     byte ptr statevar [2*statesize].misc, 00
                mov     byte ptr statevar [2*statesize].del, 00
                mov     word ptr statevar [2*statesize].index, 0000h
                mov     byte ptr statevar [2*statesize].count, 00h
                mov     byte ptr statevar [2*statesize].time, 30d
                jmp     clock
no_chan:        and     al, 01110000b
                mov     dx, 306h            ; tell PBX that command void
                or      al, 00000010b
                out     dx, al
                jmp     clock
;--------------- PLAYING ----------------
playing: mov    al, ah
                cmp     byte ptr statevar [2*statesize].statstate, 00h
                jnz     no_chan
                and     al, 01110000b       ; see if file to play exist
                mov     cl, 4               ; find the line requesting
                shr     al, cl
                mov     ah, 00h
                mov     bx, ax              ; bx = line number
                mov     bl, linesptr [bx]
                mov     bh, 00h
                mov     si, bx
                cmp     word ptr linesvar [si].pointer, 00h
                jnz     l2
                mov     cl, 4               ; al = line requested
                shl     al, cl
                jmp     no_chan             ; no file to read, same as no channel
l2:             mov     dx, 306h            ; tell PBX that command accepted
                mov     cl, 4
                shl     al, cl
                out     dx, al
                dec     word ptr linesvar [si].pointer      ; decrement pointer by 1
                mov     bx, word ptr linesvar [si].pointer
                mov     dx, bx              ; save ptr in DX
                shl     bx, 1                               ; bx = 3*bx
                add     bx, dx
                mov     ax, 3d00h                           ; open file for reading
                lea     dx, linesvar [si].namelines [bx]
                int     21h
                jnc     x1
                lea     dx, error10
                call    write
                jmp     stop
x1:             mov     byte ptr linesvar [si].statlines, ply
                mov     byte ptr statevar [2*statesize].statstate, ply
                mov     word ptr statevar [2*statesize].handstate, ax
                mov     ax, word ptr linesvar [si].namelines [bx]
                mov     word ptr statevar [2*statesize].namestate, ax
                mov     word ptr statevar [2*statesize].index, 0000h
                mov     byte ptr statevar [2*statesize].del, 01h
                mov     byte ptr statevar [2*statesize].misc, 00
                jmp     clock
;--------------- PHONE OUT ----------------
phone_out:      cmp     prog_line, 0ffh     ; see if no body programing
                jz      con_p
                mov     al, ah
                jmp     no_chan
```

```
con_p:          mov     al, ah
                cmp     byte ptr statevar [2*statesize].statstate, 00h
                jz      con_p2
                jmp     no_chan
con_p2: and     al, 01110000b       ; tell PBX that command accepted
                mov     dx, 306h
                out     dx, al
                mov     cl, 4                   ; find the line requesting
                shr     al, cl
                mov     prog_line, al       ; line in programing
                mov     ah, 00h
                mov     bx, ax      ; bx = line number
                mov     bl, linesptr [bx]
                mov     bh, 00h
                mov     si, bx      ; si = pointer to linesvar
                mov     ax, 3d00h           ; open programe message
                lea     dx, prog
                int     21h
                jnc     pr1
                lea     dx, error26
                call    write
                jmp     stop
pr1:            mov     byte ptr linesvar [si].statlines, phoneout
                mov     statevar [2*statesize].handstate, ax
                mov     statevar [2*statesize].statstate, ply
                mov     statevar [2*statesize].index, 0000h
                mov     statevar [2*statesize].misc, phoneout
                mov     statevar [2*statesize].del, 00
                jmp     clock
;--------------- IN OFFICE ---------------
in_office:      mov     al, ah
                mov     dx, 306h        ; tell PBX that command accepted
                and     al, 01110000b
                out     dx, al
                mov     cl, 4
                shr     al, cl
                mov     ah, 00h
                mov     bx, ax
                mov     bl, linesptr [bx]
                mov     bh, 00h
                mov     byte ptr linesvar [bx].statlines, isin
                jmp     tr
;--------------- OUT OFFICE ---------------
out_office:     mov     al, ah
                mov     dx, 306h        ; tell PBX that command accepted
                and     al, 01110000b
                out     dx, al
                mov     cl, 4
                shr     al, cl
                mov     ah, 00h
                mov     bx, ax
                mov     bl, linesptr [bx]
                mov     bh, 00h
                mov     byte ptr linesvar [bx].statlines, isout
                jmp     clock

;--------------- DO RING 1 ---------------
do_ring1:       or      p301w, 01h          ; close relay
                mov     al, p301w
                mov     dx, 301h
                out     dx, al
                mov     en_ring1, 00h       ; disable ring 1
tr:             mov     ax, 3d00h       ; open intro file 1
                lea     dx, intro1
                int     21h
                jnc     rr1
```

```
                lea     dx, error3
                call    write
                jmp     stop
rr1:            mov     statevar.handstate, ax
                mov     statevar.statstate, ply
                mov     statevar.index, 0000h
                mov     statevar.misc, 00
                mov     statevar.del, 00
                mov     flr_wait_flag1, 01h
                jmp     clock
;--------------- DO DTMF 1 ----------------
do_dtmf1:       mov     dx, 304h            ; clear dtmf1
                out     dx, al
                mov     dx, 303h            ; read dtmf1
                in      al, dx
                and     al, 0fh
                cmp     ptr1, 00h       ; see if first button
                jnz     transfer1           ; second do it
                mov     dtmf1, al           ; first button
                inc     ptr1
                jmp     clock
transfer1:      mov     ptr1, 00
                mov     dtmf1 [1], al       ; save second button
                mov     cx, 08h     ; search for internal line
                mov     si, 00
                mov     bl, 00
                mov     ax, word ptr dtmf1
compare:        cmp     tone [si], ax
                jz      do_transfer1    ; if found transfer
                inc     si
                inc     si
                inc     bl
                loop    compare
                mov     al, 00000001b       ; not found transfer to line 1
                mov     dx, 306h
                out     dx, al
                jmp     clock
; transfer external line 1 to internal line in bl
do_transfer1:   mov     bh, 00h     ; see if the line absent
                mov     al, bl
                mov     bl, linesptr [bx]
                cmp     byte ptr linesvar [bx].statlines, isout
                jnz     transf1
                mov     ah, 00              ; tell caller mailing
                mov     statevar.linestate, ax
                mov     ax, 3d00h
                lea     dx, linesvar [bx].mess
                int     21h
                jnc     t1
                lea     dx, error5
                call    write
                jmp     stop
t1:             mov     statevar.statstate, ply
                mov     statevar.handstate, ax
                mov     statevar.index, 0000h
                mov     statevar.del, 00
                mov     statevar.misc, waitrec
                jmp     clock
transf1:        push    ax                  ; make sure that PBX send no commamd
                mov     dx, 306h
sure:           in      al, dx
                and     al, 0fh
                cmp     al, 1001b
                jz      tell_pbx
                cmp     al, 1010b
                jnz     sure
```

```
tell_pbx: pop     ax
                  mov     cl, 4                   ; tell PBX to transfer
                  shl     al, cl
                  or      al, 00000001b
                  out     dx, al
wait_ac: in       al, dx        ; wait until PBX accept command
                  and     al, 0fh
                  cmp     al, 1010b
                  jnz     wait_ac
                  jmp     clock

;---------- DO RING 2 ----------
do_ring2:         or      p301w, 04h          ; close relay
                  mov     al, p301w
                  mov     dx, 301h
                  out     dx, al
                  mov     en_ring2, 00h       ; disable ring 2
trr:              mov     ax, 3d00h         ; open intro file 2
                  lea     dx, intro2
                  int     21h
                  jnc     rr2
                  lea     dx, error4
                  call    write
                  jmp     stop
rr2:              mov     statevar [statesize].handstate, ax
                  mov     statevar [statesize].statstate, ply
                  mov     statevar [statesize].index, 0000h
                  mov     statevar [statesize].misc, 00
                  mov     statevar [statesize].del, 00
                  mov     fir_wait_flag2, 01h
                  jmp     clock
;---------- DO DTMF 2 ----------
do_dtmf2:         mov     dx, 305h              ; clear dtmf1
                  out     dx, al
                  mov     dx, 303h            ; read dtmf2
                  in      al, dx
                  and     al, 0f0h
                  mov     cl, 4
                  shr     al, cl
                  cmp     ptr2, 00h           ; see if first button
                  jnz     transfer2           ; second do it!
                  mov     dtmf2, al             ; first
                  inc     ptr2
                  jmp     clock
transfer2:        mov     ptr2, 00
                  mov     dtmf2 [1], al       ; save second button
                  mov     cx, 08h    ; search for internal line
                  mov     si, 00
                  mov     bl, 00
                  mov     ax, word ptr dtmf2
compare2:         cmp     tone [si], ax
                  jz      do_transfer2        ; if found transfer
                  inc     si
                  inc     si
                  inc     bl
                  loop    compare2
                  mov     al, 00000001b       ; not found transfer to line 1
                  mov     dx, 306h
                  out     dx, al
                  jmp     clock
; transfer external line 1 to internal line in bl
do_transfer2:     mov     bh, 00h     ; see if the line absent
                  mov     al, bl
                  mov     bl, linesptr [bx]
                  cmp     byte ptr linesvar [bx].statlines, isout
                  jnz     transf2
```

```
                mov     ah, 00                  ; tell caller mailing
                mov     statevar [statesize].linestate, ax
                mov     ax, 3d00h
                lea     dx, linesvar [bx].mess
                int     21h
                jnc     t2
                lea     dx, error5
                call    write
                jmp     stop
t2:             mov     statevar [statesize].statstate, ply
                mov     statevar [statesize].handstate, ax
                mov     statevar [statesize].index, 0000h
                mov     statevar [statesize].del, 00
                mov     statevar [statesize].misc, waitrec
                jmp     clock
transf2:        push    ax                      ; make sure that PBX send no commamd
                mov     dx, 306h
sure_2:  in      al, dx
                and     al, 0fh
                cmp     al, 1001b
                jz      tell_pbx2
                cmp     al, 1010b
                jnz     sure_2
tell_pbx2:      pop     ax
                mov     cl, 4                   ; tell PBX to transfer
                shl     al, cl
                or      al, 00000001b
                out     dx, al
wait_ac2:       in      al, dx          ; wait until PBX accept command
                and     al, 0fh
                cmp     al, 1010b
                jnz     wait_ac2
                jmp     clock

;---------- CHECK STATE ----------
checkstate:     cmp     statevar.statstate, 00      ; see if wait recording
                jnz     nextchan
                cmp     statevar.misc, waitrec
                jnz     nextchan
do_state0:      mov     bx, statevar.linestate
                mov     bl, linesptr [bx]
                mov     bh, 00h
                mov     si, bx
                mov     bx, word ptr linesvar [si].pointer
                mov     ax, bx      ; save bx in ax
                shl     bx, 1       ; bx=bx*3
                add     bx, ax
                mov     ah, 3ch
                mov     cx, 00h
                lea     dx, linesvar [si].namelines [bx]
                int     21h
                jnc     c1
                lea     dx, error1
                call    write
                jmp     stop
c1:             inc     word ptr linesvar [si].pointer
                mov     statevar.statstate, rec
                mov     statevar.handstate, ax
                mov     statevar.index, 0000h
                mov     statevar.count, 00
                mov     statevar.time, time1        ; record for 15 sec
                mov     statevar.misc, 00
                jmp     chan1

nextchan:       cmp     byte ptr statevar [statesize].statstate, 00  ; see if wait recording
                jz      t
```

```
                jmp     chan1
t:              cmp     byte ptr statevar [statesize].misc, waitrec
                jz      do_state2
                jmp     chan1
do_state2:      mov     bx, word ptr statevar [statesize].linestate
                mov     bl, linesptr [bx]
                mov     bh, 00h
                mov     si, bx
                mov     bx, word ptr linesvar [si].pointer
                mov     ax, bx      ; bx=bx*3
                shl     bx, 1
                add     bx, ax
                mov     ah, 3ch
                mov     cx, 00h
                lea     dx, linesvar [si].namelines [bx]
                int     21h
                jnc     c2
                lea     dx, error2
                call    write
                jmp     stop
c2:             inc     word ptr linesvar [si].pointer
                mov     byte ptr statevar [statesize].statstate, rec
                mov     word ptr statevar [statesize].handstate, ax
                mov     word ptr statevar [statesize].index, 0000h
                mov     byte ptr statevar [statesize].count, 00
                mov     byte ptr statevar [statesize].time, time2
                mov     byte ptr statevar [statesize].misc, 00
                jmp     chan1

;------------- DO DTMF 3 ----------------
do_dtmf3:       cmp     chk_tone, 00h       ; if not in programing, do nothing
                jnz     chkphone
                jmp     rec_phone
chkphone:       mov     dx, 302h    ; see if there is tone on ch3
                in      al, dx
                test    al, 01000000b
                jnz     tone3_in
                jmp     rec_phone
tone3_in:       mov     dx, 307h        ; clear tone
                in      al, dx
                mov     dx, 304h    ; read tone
                in      al, dx
                and     al, 0fh     ; al = tone decoded
                cmp     chk_tone, 01h
                jnz     more2
                mov     ah, 00h                 ; convert al
                mov     bx, ax
                mov     cl, buttoncode [bx]
                mov     temp, cl                ; 1st button
                inc     chk_tone
                jmp     rec_phone

more2:          cmp     chk_tone, 02h
                jnz     more3
                mov     ah, 00h                 ; convert al
                mov     bx, ax
                mov     al, buttoncode [bx]
                mov     bl, temp    ; 2nd button
                mov     ah, bl                  ; bl*10
                mov     cl, 3
                shl     bl, cl
                add     ah, ah
                add     ah, bl
                add     ah, al
                mov     bl, prog_line
                mov     bh, 00h
```

```
                mov     bl, linesptr [bx]
                mov     byte ptr linesvar [bx].p2_time, ah
                inc     chk_tone
                jmp     rec_phone

more3:          cmp     chk_tone, 03h
                jnz     more4
                mov     ah, 00h                 ; convert al
                mov     bx, ax
                mov     cl, buttoncode [bx]
                mov     temp, cl                ; 1st button
                inc     chk_tone
                jmp     rec_phone

more4:          cmp     chk_tone, 04h
                jnz     more5
                mov     ah, 00h                 ; convert al
                mov     bx, ax
                mov     al, buttoncode [bx]
                mov     bl, temp    ; 2nd button
                mov     ah, bl                  ; bl*10
                mov     cl, 3
                shl     bl, cl
                add     ah, ah
                add     ah, bl
                add     ah, al
                mov     bl, prog_line
                mov     bh, 00h
                mov     bl, linesptr [bx]
                mov     byte ptr linesvar [bx].p1_time, ah
                inc     chk_tone
                jmp     rec_phone

more5:          cmp     al, 1100b               ; if # button found
                jz      rty
                mov     si, num_ptr
                mov     bl, prog_line
                mov     bh, 00h
                mov     bl, linesptr [bx]
                mov     byte ptr linesvar [bx].number [si], al
                mov     byte ptr linesvar [bx].statlines, recordphone
                inc     num_ptr
                inc     chk_tone
                jmp     rec_phone
rty:            mov     ax, 3d00h               ; open recording message
                lea     dx, recph
                int     21h
                jnc     prr1
                lea     dx, error28
                call    write
                jmp     stop
prr1:           mov     statevar [2*statesize].handstate, ax
                mov     statevar [2*statesize].statstate, ply
                mov     statevar [2*statesize].index, 0000h
                mov     statevar [2*statesize].misc, recordphone
                mov     statevar [2*statesize].del, 00
                jmp     rec_phone

;---------- RECORD VOICE FOR PHONE OUT ----------
rec_phone:      cmp     recphone, 01h
                jz      sd
                jmp     check_time
sd:             mov     bl, prog_line
                mov     bh, 00h
                mov     bl, linesptr [bx]
                mov     ah, 3ch  ; create file
```

```
                mov     cx, 00h
                lea     dx, linesvar [bx].p_file
                int     21h
                jnc     lsd1
                lea     dx, error29
                call    write
                jmp     stop
lsd1:           mov     byte ptr linesvar [bx].statlines, recordphone
                mov     byte ptr statevar [2*statesize].statstate, rec
                mov     word ptr statevar [2*statesize].handstate, ax
                mov     byte ptr statevar [2*statesize].misc, 00
                mov     byte ptr statevar [2*statesize].del, 00
                mov     word ptr statevar [2*statesize].index, 0000h
                mov     byte ptr statevar [2*statesize].count, 00h
                mov     byte ptr statevar [2*statesize].time, 30d
                mov     recphone, 00h                   ; disable this routine
                jmp     check_time

;--------------- CHECK TIME FOR PHONE OUT ----------------
check_time:     cmp     time_flag, 01h
                jz      sdf
                jmp     check_signal
sdf:            cmp     afterclose, 01h
                jz      he
                cmp     pho2, 01h       ; see if sending tone not in process
                jnz     chk_time
                jmp     gf
chk_time:       mov     ah, 2ch         ; no, read time
                int     21h
search_time:    mov     si, track       ; see if tracked line to phone out
                mov     dl, linesptr [si]
                mov     dh, 00h
                mov     si, dx
                cmp     word ptr linesvar [si].p1_time, cx
                jz      do_phone
                inc     track                   ; no, next line
                cmp     track, 08h
                jnz     jj
                mov     track, 00h
jj:             jmp     check_signal
do_phone:       cmp     num_ptr, 0000h
                jnz     cmp_next
                mov     afterclose, 01h
                mov     dx, 301h
                or      p301w, 10010000b        ; close relay
                mov     al, p301w
                out     dx, al
he:             inc     delay_time              ; delay 4 sec. after closing relay
                cmp     delay_time, 8000d
                jz      cmp_next
                jmp     check_signal
cmp_next:       mov     dx, 304h                ; clear ring 1, BUG!
                out     dx, al
                mov     afterclose, 00h
                mov     delay_time, 0000h
                mov     pho2, 01h
gf:             mov     dx, 302h                ; see if pulse low or high
                in      al, dx
                and     al, 80h
                jnz     send_tone
                mov     low_state, 01h          ; pulse low, set low_state flag
                jmp     check_signal
send_tone:      cmp     low_state, 01h          ; pulse high, see if high from low
                jz      send_it
                jmp     check_signal
send_it:        mov     low_state, 00h          ; reset low_state flag
```

```
                mov     bx, num_ptr
                mov     si, track
                mov     cl, linesptr [si]
                mov     ch, 00h
                mov     si, cx
                mov     al, byte ptr linesvar [si].number [bx]
                cmp     al, '#'         ; see if last tone
                jnz     num
                mov     dx, 307h                ; off tone
                mov     al, 00h
                out     dx, al
                mov     al, '#'         ; yes, make all tone buffer '#'
                lea     di, linesvar [si].number
                mov     cx, 10d
                rep     stosb
                mov     chk_sig, 01h                    ; enable check_signal routine
                mov     pho2, 00h
                mov     num_ptr, 00h                    ; reset buffer pointer
                mov     time_flag, 00h  ; disable this routine
                jmp     check_signal
num:            inc     num_ptr         ; send tone on rising edge
                mov     dx, 307h
                mov     bl, al      ; convert code
                mov     bh, 00h
                mov     al, sound [bx]
                out     dx, al
                jmp     check_signal

;-------------- CHECK RESPONSE ---------------
check_signal:   cmp     chk_sig, 00h            ; see if sending tone finished
                jnz     five_minite
                jmp     check_break
five_minite:    inc     chk_sig         ; count chk_sig until 7 sec.
                cmp     chk_sig, 15000d
                jz      say
                jmp     check_break
say:            mov     si, track  ; 0 found, ring back detected
                mov     bl, linesptr [si]
                mov     bh, 00h
                mov     si, bx
                mov     ax, 3d00h               ; open file to phone out
                lea     dx, linesvar [si].p_file
                int     21h
                jnc     sss
                lea     dx, error27
                call    write
                jmp     stop
sss:            mov     statevar [2*statesize].handstate, ax
                mov     statevar [2*statesize].statstate, ply
                mov     statevar [2*statesize].index, 0000h
                mov     statevar [2*statesize].misc, phoneout
                mov     statevar [2*statesize].del, 00
                mov     chk_sig, 00h            ; disable this routine
                jmp     check_break

                include  c:\ja\channels.asm

code        ends
                end     start
```

```
;*************** Include file ******************
; READ/WRITE IN SAMPLING PROCESS
;--------------- CHANNELS.ASM----------------
chan1:          cmp     statevar.statstate, rec
                jnz     r1
                jmp     rec1
r1:             cmp     statevar.statstate, ply
                jnz     chan2
                jmp     ply1
chan2:          cmp     byte ptr statevar [statesize].statstate, rec
                jnz     r2
                jmp     rec2
r2:             cmp     byte ptr statevar [statesize].statstate, ply
                jnz     chan3
                jmp     ply2
chan3:          cmp     byte ptr statevar [2*statesize].statstate, rec
                jnz     r3
                jmp     rec3
r3:             cmp     byte ptr statevar [2*statesize].statstate, ply
                jnz     p3
                jmp     ply3
p3:             jmp     clock
;--------------- CHANNEL 1 ------------------
rec1:           or      p301w, 00000010b
                mov     dx, 301h                ; select record channel 1
                mov     al, p301w
                out     dx, al
                mov     si, statevar.index
                dec     dx                      ; read voice ch 1
                in      al, dx
                mov     statevar.buffer [si], al    ; save in voice array
                inc     si
                mov     statevar.index, si
                cmp     si, recsize             ; see if buffer full
                jnz     notfull1
                mov     ah, 40h             ; full! write to file
                mov     bx, statevar.handstate
                mov     cx, recsize
                lea     dx, statevar.buffer
                int     21h
                jnc     m1
                lea     dx, error11
                call    write
                jmp     stop
m1:             mov     statevar.index, 00      ; see if time out
                inc     statevar.count
                mov     al, statevar.time
                cmp     statevar.count, al
                jnz     notfull1
                and     p301w, 11111110b        ; open relay 1
                mov     al, p301w
                mov     dx, 301h
                out     dx, al
                mov     statevar.statstate, 00  ; time out, close file
                mov     statevar.misc, 00       ; end of recording
                mov     statevar.count, 00
                mov     ah, 3eh
                mov     bx, statevar.handstate
                int     21h
                jnc     notfull1
                lea     dx, error12
                call    write
                jmp     stop
notfull1:       jmp     chan2

ply1:           and     p301w, 11111101b
```

```
                mov     al, p301w
                mov     dx, 301h            ; play channel 1
                out     dx, al
                cmp     statevar.index, 0000h       ; if start, read a section
                jne     |_read1                 ; if not, play remainder
                mov     ax, 3f00h           ; read file
                mov     bx, statevar.handstate
                mov     cx, recsize
                lea     dx, statevar.buffer
                int     21h
                jnc     m2
                lea     dx, error13
                call    write
                jmp     stop
m2:             cmp     ax, recsize
                jnz     end_of_file1            ; end of file!
|_read1: mov    si, statevar.index
                mov     al, statevar.buffer [si]
                mov     dx, 302h        ; write voice ch 1
                out     dx, al
                inc     si
                mov     statevar.index, si
                cmp     si, recsize             ; see if play all of this record
                jz      con1                ; not all, not load record later
                jmp     chan2
con1:           mov     statevar.index, 00 ; all, reset index for reading next
                jmp     chan2
end_of_file1:   mov     statevar.statstate, 00
                mov     ah, 3eh             ; close file
                mov     bx, statevar.handstate
                int     21h
                jnc     cmp1
                lea     dx, error14
                call    write
                jmp     stop
cmp1:           cmp     statevar.del, 01h
                jz      del1
                jmp     chan2
del1:           mov     ah, 41h             ; delete file
                lea     dx, statevar.namestate
                int     21h
                jnc     ee1
                lea     dx, error15
                call    write
                jmp     stop
ee1:            jmp     chan2
;---------- CHANNEL 2 ----------
rec2:           or      p301w, 00001000b
                mov     dx, 301h                    ; select record channel 1
                mov     al, p301w
                out     dx, al
                mov     si, word ptr statevar[statesize].index
                in      al, dx              ; read voice ch 2
                mov     byte ptr statevar[statesize].buffer [si], al        ; save in voice array
                inc     si
                mov     word ptr statevar [statesize].index, si
                cmp     si, recsize                 ; see if buffer full
                jnz     notfull2
                mov     ah, 40h             ; full!  write to file
                mov     bx, word ptr statevar [statesize].handstate
                mov     cx, recsize
                lea     dx, statevar [statesize].buffer
                int     21h
                jnc     m3
                lea     dx, error16
                call    write
```

```
                jmp     stop
m3:             mov     word ptr statevar [statesize].index, 00     ; see if time out
                inc     byte ptr statevar [statesize].count
                mov     al, byte ptr statevar [statesize].time
                cmp     byte ptr statevar [statesize].count, al
                jnz     notfull2
                and     p301w, 11111011b        ; open relay 2
                mov     al, p301w
                mov     dx, 301h
                out     dx, al
                mov     byte ptr statevar [statesize].statstate, 00     ; time out, close file
                mov     byte ptr statevar [statesize].misc, 00      ; end of recording
                mov     byte ptr statevar [statesize].count, 00
                mov     ah, 3eh
                mov     bx, word ptr statevar [statesize].handstate
                int     21h
                jnc     notfull2
                lea     dx, error17
                call    write
                jmp     stop
notfull2:       jmp     chan3

ply2:           and     p301w, 11110111b
                mov     al, p301w
                mov     dx, 301h            ; play channel 2
                out     dx, al
                cmp     word ptr statevar [statesize].index, 0000h  ; if start, read a section
                jne     j_read2             ; if not, play remainder
                mov     ax, 3f00h           ; read file
                mov     bx, word ptr statevar [statesize].handstate
                mov     cx, recsize
                lea     dx, word ptr statevar [statesize].buffer
                int     21h
                jnc     m4
                lea     dx, error18
                call    write
                jmp     stop
m4:             cmp     ax, recsize
                jnz     end_of_file2        ; end of file!
j_read2: mov    si, word ptr statevar [statesize].index
                mov     al, byte ptr statevar [statesize].buffer [si]
                mov     dx, 303h            ; write voice ch 2
                out     dx, al
                inc     si
                mov     word ptr statevar [statesize].index, si
                cmp     si, recsize         ; see if play all of this record
                jz      con2                ; not all, not load record later
                jmp     chan3
con2:           mov     word ptr statevar [statesize].index, 00 ; all, reset index for reading next
                jmp     chan3
end_of_file2:   mov     byte ptr statevar [statesize].statstate, 00
                mov     ah, 3eh             ; close file
                mov     bx, word ptr statevar [statesize].handstate
                int     21h
                jnc     cmp2
                lea     dx, error19
                call    write
                jmp     stop
cmp2:           cmp     statevar [statesize].del, 01h
                jz      del2
                jmp     chan3
del2:           mov     ah, 41h             ; delete file
                lea     dx, word ptr statevar [statesize].namestate
                int     21h
                jnc     ee2
                lea     dx, error20
```

```
                call    write
                jmp     stop
ee2:            jmp     chan3

;-------------- CHANNEL 3 -------------------
rec3:           or      p301w, 10000000b
                mov     dx, 301h                ; select record channel 3
                mov     al, p301w
                out     dx, al
                mov     si, word ptr statevar[2*statesize].index
                mov     dx, 305h
                in      al, dx          ; read voice ch 3
                mov     byte ptr statevar[2*statesize].buffer [si], al    ; save in voice array
                inc     si
                mov     word ptr statevar [2*statesize].index, si
                cmp     si, recsize             ; see if buffer full
                jnz     notfull3
                mov     ah, 40h         ; full!  write to file
                mov     bx, word ptr statevar [2*statesize].handstate
                mov     cx, recsize
                lea     dx, statevar [2*statesize].buffer
                int     21h
                jnc     m5
                lea     dx, error21
                call    write
                jmp     stop
m5:             mov     word ptr statevar [2*statesize].index, 0000  ; see if time out
                inc     byte ptr statevar [2*statesize].count
                mov     al, byte ptr statevar [2*statesize].time
                cmp     byte ptr statevar [2*statesize].count, al
                jnz     notfull3
; make del = 01 means this routine has closed the file
                mov     byte ptr statevar [2*statesize].del, 01h
                mov     byte ptr statevar [2*statesize].statstate, 00
                mov     byte ptr statevar [2*statesize].count, 00
                mov     ah, 3eh  ; close file
                mov     bx, word ptr statevar [2*statesize].handstate
                int     21h
                jnc     notfull3
                lea     dx, error22
                call    write
                jmp     stop
notfull3:       cmp     byte ptr statevar [2*statesize].misc, recordphone
                jnz     ue
                mov     time_flag, 01h
ue:             jmp     clock

ply3:           and     p301w, 01111111b
                mov     al, p301w
                mov     dx, 301h            ; play channel 3
                out     dx, al
                cmp     word ptr statevar [2*statesize].index, 0000h  ; if start, read a section
                jne     j_read3                 ; if not, play remainder
                mov     ax, 3f00h           ; read file
                mov     bx, word ptr statevar [2*statesize].handstate
                mov     cx, recsize
                lea     dx, statevar [2*statesize].buffer
                int     21h
                jnc     m6
                lea     dx, error23
                call    write
                jmp     stop
m6:             cmp     ax, recsize
                jnz     end_of_file3        ; end of file!
j_read3: mov    si, word ptr statevar [2*statesize].index
                mov     al, byte ptr statevar [2*statesize].buffer [si]
```

```
                mov     dx, 300h            ; write voice ch 3
                out     dx, al
                inc     si
                mov     word ptr statevar [2*statesize].index, si
                cmp     si, recsize         ; see if play all of this record
                jz      con3                ; not all, not load record later
                jmp     clock
con3:           mov     word ptr statevar [2*statesize].index, 00 ; all, reset index for reading next
                jmp     clock
end_of_file3:   cmp     byte ptr statevar [2*statesize].misc, phoneout
                jnz     ku
                mov     chk_tone, 01h
                mov     dx, 307h    ; clear tone ch3
                in      al, dx
                mov     dx, 301h    ; off noise
                or      p301w, 10000000b
                mov     al, p301w
                out     dx, al
ku:             cmp     byte ptr statevar [2*statesize].misc, recordphone
                jnz     gd
                mov     recphone, 01h
gd:             mov     byte ptr statevar [2*statesize].statstate, 00
                mov     ah, 3eh             ; close file
                mov     bx, word ptr statevar [2*statesize].handstate
                int     21h
                jnc     cmp3
                lea     dx, error24
                call    write
                jmp     stop
cmp3:           cmp     byte ptr statevar [2*statesize].del, 01h
                jz      del3
                jmp     open_relay
del3:           mov     ah, 41h             ; delete file
                lea     dx, word ptr statevar [2*statesize].namestate
                int     21h
                jnc     open_relay
                lea     dx, error25
                call    write
                jmp     stop
open_relay:     mov     byte ptr statevar [2*statesize].del, 02h
                mov     dx, 301h
                and     p301w, 11101111b
                mov     al, p301w
                out     dx, al
                jmp     clock

stop:           mov     dx, 301h
                mov     al, 00h
                out     dx, al
                mov     dx, 307h
                out     dx, al
                mov     ax, 4c00h
                int     21h
main        endp

write           proc    near
                mov     ah, 09h
                int     21h
                ret
write           endp
```

ภาคผนวก ค โปรแกรมต้นฉบับของระบบแจ้งข่าวสารบนเครื่องพีซี

```
;*****************************************************************
;*              Programe for sending inform file to 8031         *
;*****************************************************************
;
cr              equ     0ah
lf              equ     0dh
                assume  cs:code
                assume  ds:data
com1            equ     3f8h
data        segment
inform          db      'inform', 0
inf             db      256 dup(?)
data        ends

code            segment
main            proc    far
start:          mov     ax, data
                mov     ds, ax
                ; open inform file
                mov     ah, 3d00h
                lea     dx, inform
                int     21h
                mov     ah, 3fh         ; read data
                mov     bx, ax
                mov     cx, 256d
                lea     dx, inf
                int     21h

                mov     dx, 301h
                mov     al, 00h
                out     dx, al
                dec     dx
                out     dx, al

                mov     ax, 00F3h       ; 9600 baud 1, stop bit, none, 8 bits
                mov     dx, 00h         ; com1
                int     14h

                mov     si, 00h
                mov     cx, 255d
get:            mov     dx, 300h                ; send null
                mov     al, 00h
                out     dx, al
                mov     dx, 306h                ; see if board ready
                in      al, dx
                test    al, 80h
                jz      get
                mov     dx, 300h                ; ready, send new data
                mov     al, inf [si]
                out     dx, al
                mov     ah, 02h
                mov     dl, al
                int     21h
                mov     dx, 306h                ; wait till jr31 accept data
waitt:          in      al, dx
                test    al, 80h
                jnz     waitt
                inc     si
                loop    get
                jmp     stop
stop:           mov     ax, 4c00h
```

```
                int         21h
main            endp

; INPUT: BL=character to send
send            proc        near
                mov         dx, com1+5
s:              in          al, dx
                test        al, 20h
                jz          s
                mov         al, bl
                mov         dx, com1
                out         dx, al
                ret
send            endp

; OUTPUT: AL=character received
recev           proc        near
                mov         dx, com1+5
r:              in          al, dx
                test        al, 1
                jz          r
                mov         dx, com1
                in          al, dx
                ret
recev           endp

delay           proc        near
                push        dx
                mov         dx, 8000d       ; 5000d
von:            dec         dx
                cmp         dx, 0000h
                jnz         von
                pop         dx
                ret
delay           endp

code            ends
                end         start
```

ภาคผนวก ง โปรแกรมต้นฉบับของระบบแจ้งข่าวสารบนบอร์ดฮาร์ดแวร์

```
        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


;*****************************************************
                org     00h
contrl:         equ     0e100h
porta:          equ     0e101h
portb:          equ     0e102h
portc:          equ     0e103h
mem:            equ     0e000h  ; 256 byte E000h to E0FFh

start:          ljmp    init
                org     100h
init:           mov     sp, #30
                lcall   delay
                lcall   delay
                mov     a, #0fdh  ; baud rate = 9600
setbaud:        mov     th1, a
                mov     tmod, #00100000b  ; timer mode 1, 8 bit auto-reload
                setb    tr1                     ; enable timer1
                mov     scon, #01010010b        ; set serial mode 1
; serial mode 1, 1 start bit + 8 bit data + 1 stop bit
st1:            mov     dptr, #contrl           ; 8155 mode a, c output
                mov     a, #00001101b           ; b input
                movx    @dptr, a
                mov     a, #00h   ; test port a
                mov     dptr, #porta
                movx    @dptr, a

                lcall   read_data           ; read data from PC
                mov     a, #00h     ; start from station 1
send:           mov     dptr, #portc
                movx    @dptr, a
                push    acc
                lcall   send_to_PC
                pop     acc
                inc     acc
                cjne    a, #04h, send
                mov     a, #00h
                ljmp    send
```

```
read_data:      mov     dptr, #mem
                push    dpl             ;.save dptr to mem
                push    dph
                mov     r0, #0d
null:           mov     dptr, #porta    ; tell PC that ready to get data
                mov     a, #0ffh
                movx    @dptr, a
                mov     dptr, #portb    ; see if a character found
                movx    a, @dptr
                cjne    a, #00d, char
                ljmp    null
char:           pop     dph             ; found! save in 8155 memory
                pop     dpl
                movx    @dptr, a
                inc     dptr            ; point to next address
                push    dpl
                push    dph
                mov     dptr, #porta    ; tell PC data accepted
                mov     a, #00h
                movx    @dptr, a
                lcall   tx_delay        ; wait until PC waiting for new data
                lcall   tx_delay
                inc     r0
                cjne    r0, #255d, null
                pop     dph
                pop     dpl
                ret

send_to_PC:     mov     dptr, #mem
                mov     r0, 255d
sendPC:         movx    a, @dptr
                lcall   tx_byte
                lcall   tx_delay
                inc     dptr
                djnz    r0, sendPC
                ret

tx_byte:        jnb     ti, $           ; ti=1 when presvious byte has been sent
                clr     ti
                mov     sbuf, a
                ret

rx_byte:        jnb     ri, $
                clr     ri
                mov     a, sbuf
                ret

delay:          mov     r1, #70h
dl1:            mov     r2, #0f0h
dl2:            nop
                djnz    r2, dl2
                djnz    r1, dl1
                ret

tx_delay:       mov   r1, #00h
dll:            mov   r2, #10h
dl:             nop
                djnz  r2, dl
                djnz  r1, dll
                ret

                end
```

ภาคผนวก ฉ รายละเอียดไอซี MC3417

MC3417, MC3517
MC3418, MC3518

2

Specifications and Applications Information

CONTINUOUSLY VARIABLE SLOPE DELTA MODULATOR/DEMODULATOR

Providing a simplified approach to digital speech encoding/decoding, the MC3517/18 series of CVSDs is designed for military secure communication and commercial telephone applications. A single IC provides both encoding and decoding functions.

- Encode and Decode Functions on the Same Chip with a Digital Input for Selection
- Utilization of Compatible I^2L – Linear Bipolar Technology
- CMOS Compatible Digital Output
- Digital Input Threshold Selectable ($V_{CC}/2$ reference provided on chip)
- MC3417/MC3517 has a 3-Bit Algorithm (General Communications)
- MC3418/MC3518 has a 4-Bit Algorithm (Commercial Telephone)

CONTINUOUSLY VARIABLE SLOPE DELTA MODULATOR/DEMODULATOR

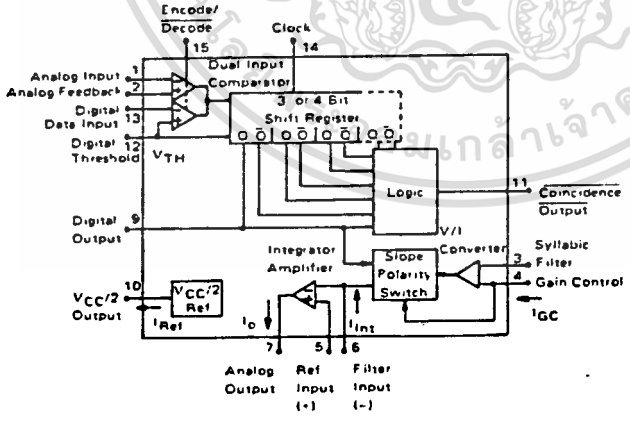
LASER-TRIMMED INTEGRATED CIRCUIT

L SUFFIX
CERAMIC PACKAGE
CASE 620-10

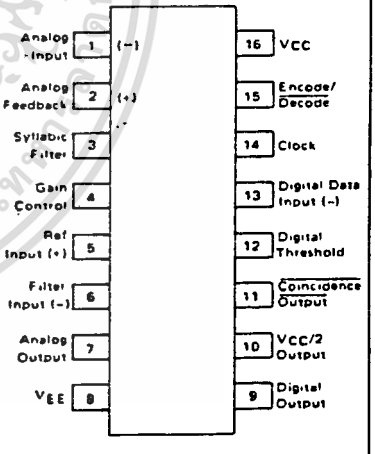
P SUFFIX
PLASTIC PACKAGE
CASE 648-08

DW SUFFIX
PLASTIC PACKAGE
CASE 751G-01
SO-16L

CVSD BLOCK DIAGRAM



PIN CONNECTIONS



ORDERING INFORMATION

Device	Package	Temperature Range
MC3417L	Ceramic DIP	0°C to +70°C
MC3418DW	Plastic SOIC	0°C to +70°C
MC3418L	Ceramic DIP	0°C to +70°C
MC3418P	Plastic DIP	0°C to +70°C
MC3517L	Ceramic DIP	–55°C to +125°C
MC3518L	Ceramic DIP	–55°C to +125°C

MAXIMUM RATINGS

(All voltages referenced to V_{EE}. T_A = 25°C unless otherwise noted.)

Rating	Symbol	Value	Unit
Power Supply Voltage	V_{CC}	−0.4 to +18	Vdc
Differential Analog Input Voltage	V_{ID}	±5.0	Vdc
Digital Threshold Voltage	V_{TH}	−0.4 to V_{CC}	Vdc
Logic Input Voltage (Clock, Digital Data, Encode/$\overline{\text{Decode}}$)	V_{Logic}	−0.4 to +18	Vdc
Coincidence Output Voltage	$V_{O(Con)}$	−0.4 to +18	Vdc
Syllabic Filter Input Voltage	$V_{I(Syl)}$	−0.4 to V_{CC}	Vdc
Gain Control Input Voltage	$V_{I(GC)}$	−0.4 to V_{CC}	Vdc
Reference Input Voltage	$V_{I(Ref)}$	$V_{CC}/2$ − 1.0 to V_{CC}	Vdc
$V_{CC}/2$ Output Current	I_{Ref}	−25	mA

ELECTRICAL CHARACTERISTICS

(V_{CC} = 12 V, V_{EE} = Gnd, T_A = 0°C to +70°C for MC3417/18, T_A = −55°C to +125°C for MC3517/18 unless otherwise noted.)

Characteristic	Symbol	MC3417/MC3517			MC3418/MC3518			Unit
		Min	Typ	Max	Min	Typ	Max	
Power Supply Voltage Range (Figure 1)	V_{CCR}	4.75	12	16.5	4.75	12	16.5	Vdc
Power Supply Current (Figure 1)	I_{CC}							mA
(@ Idle Channel)								
(V_{CC} = 5.0 V, All except MC3418P,DW)		—	3.7	5.0	—	3.7	5.0	
(V_{CC} = 5.0 V, MC3418P,DW)		—	—	—	—	3.7	5.5	
(V_{CC} = 15 V, All except MC3418P,DW)		—	6.0	10	—	6.0	10	
(V_{CC} = 15 V, MC3418P,DW)		—	—	—	—	6.0	11	
Gain Control Current Range (Figure 2)	I_{GCR}	0.002	—	3.0	0.002	—	3.0	mA
Analog Comparator Input Range (Pins 1 and 2) (4.75 V ≤ V_{CC} ≤ 16.5 V)	V_I	1.3	—	V_{CC} − 1.3	1.3	—	V_{CC} − 1.3	Vdc
Analog Output Range (Pin 7) (4.75 V ≤ V_{CC} ≤ 16.5 V, I_O = ±5.0 mA)	V_O	1.3	—	V_{CC} − 1.3	1.3	—	V_{CC} − 1.3	Vdc
Input Bias Currents (Figure 3)	I_{IB}							μA
(Comparator in Active Region)								
Analog Input (I1)		—	0.5	1.5	—	0.25	1.0	
Analog Feedback (I2)		—	0.5	1.5	—	0.25	1.0	
Syllabic Filter Input (I3)		—	0.06	0.5	—	0.06	0.3	
Reference Input (I5)		—	−0.06	−0.5	—	−0.06	−0.3	
Input Offset Current	I_{IO}							μA
(Comparator in Active Region)								
Analog Input/Analog Feedback |I1 − I2| — Figure 3		—	0.15	0.6	—	0.05	0.4	
Integrator Amplifier |I5 − I6| — Figure 4		—	0.02	0.2	—	0.01	0.1	
Input Offset Voltage	V_{IO}	—	2.0	6.0	—	2.0	6.0	mV
V/I Converter (Pins 3 and 4) — Figure 5								
Transconductance	gm							mA/mV
V/I Converter, 0 to 3.0 mA		0.1	0.3	—	0.1	0.3	—	
Integrator Amplifier, 0 to ±5.0 mA Load		1.0	10	—	1.0	10	—	
Propagation Delay Times (Note 1)								μs
Clock Trigger to Digital Output	t_{PLH}	—	1.0	2.5	—	1.0	2.5	
(C_L = 25 pF to Gnd)	t_{PHL}	—	0.8	2.5	—	0.8	2.5	
Clock Trigger to Coincidence Output	t_{PLH}	—	1.0	3.0	—	1.0	3.0	
(C_L = 25 pF to Gnd) (R_L = 4.0 kΩ to V_{CC})	t_{PHL}	—	0.8	2.0	—	0.8	2.0	
Coincidence Output Voltage — Low Logic State ($I_{OL(Con)}$ = 3.0 mA)	$V_{OL(Con)}$	—	0.12	0.25	—	0.12	0.25	Vdc
Coincidence Output Leakage Current — High Logic State (V_{OH} = 15 V, 0°C ≤ T_A ≤ 70°C)	$I_{OH(Con)}$	—	0.01	0.5	—	0.01	0.5	μA

NOTE 1. All propagation delay times measured 50% to 50% from the negative going (from V_{CC} to +0.4 V) edge of the clock.

2

ELECTRICAL CHARACTERISTICS (continued)

Characteristic	Symbol	MC3417/MC3517 Min	MC3417/MC3517 Typ	MC3417/MC3517 Max	MC3418/MC3518 Min	MC3418/MC3518 Typ	MC3418/MC3518 Max	Unit
Applied Digital Threshold Voltage Range (Pin 12)	V_{TH}	+ 1.2	—	V_{CC} – 2.0	+ 1.2	—	V_{CC} – 2.0	Vdc
Digital Threshold Input Current ($1.2 V \leq V_{th} \leq V_{CC} - 2.0 V$)	$I_{I(th)}$							μA
(V_{IL} applied to Pins 13, 14 and 15)		—	—	5.0	—	—	5.0	
(V_{IH} applied to Pins 13, 14 and 15)		—	– 10	– 50	—	– 10	– 50	
Maximum Integrator Amplifier Output Current	I_O	± 5.0	—	—	± 5.0	—	—	mA
$V_{CC}/2$ Generator Maximum Output Current (Source only)	I_{Ref}	+ 10	—	—	+ 10	—	—	mA
$V_{CC}/2$ Generator Output Impedance (0 to + 10 mA)	z_{Ref}	—	3.0	6.0	—	3.0	6.0	Ω
$V_{CC}/2$ Generator Tolerance ($4.75 V \leq V_{CC} \leq 16.5 V$)	er	—	—	± 3.5	—	—	± 3.5	%
Logic Input Voltage (Pins 13, 14 and 15)								Vdc
Low Logic State	V_{IL}	Gnd	—	V_{th} – 0.4	Gnd	—	V_{th} – 0.4	
High Logic State	V_{IH}	V_{th} + 0.4	—	18	V_{th} + 0.4	—	18	
Dynamic Total Loop Offset Voltage (Note 2) — Figures 3, 4 and 5	ΣV_{offset}							mV
I_{GC} = 12 μA, V_{CC} = 12 V								
T_A = 25°C (All except 3418P,DW)		—	—	—	—	± 0.5	± 1.5	
(MC3418P,DW)		—	—	—	—	± 0.5	± 3.0	
0°C ≤ T_A ≤ +70°C (MC3417/18L)		—	—	—	—	± 0.75	± 2.3	
(MC3418P,DW)		—	—	—	—	± 0.75	± 3.8	
– 55°C ≤ T_A ≤ +125°C (MC3517/18)		—	—	—	—	± 1.5	± 4.0	
I_{GC} = 33 μA, V_{CC} = 12 V								
T_A = 25°C		—	± 2.5	± 5.0	—	—	—	
0°C ≤ T_A ≤ +70°C (MC3417/18)		—	± 3.0	± 7.5	—	—	—	
– 55°C ≤ T_A ≤ +125°C (MC3517/18)		—	± 4.5	± 10	—	—	—	
I_{GC} = 12 μA, V_{CC} = 5.0 V								
T_A = 25°C (All except MC3418P,DW)		—	—	—	—	± 1.0	± 2.0	
(MC3418P,DW)		—	—	—	—	± 1.0	± 3.5	
0°C ≤ T_A ≤ +70°C (MC3417/18L)		—	—	—	—	± 1.3	± 2.8	
(MC3418P,DW)		—	—	—	—	± 1.3	± 4.3	
– 55°C ≤ T_A ≤ +125°C (MC3517/18)		—	—	—	—	± 2.5	± 5.0	
I_{GC} = 33 μA, V_{CC} = 5.0 V								
T_A = 25°C		—	± 4.0	± 6.0	—	—	—	
0°C ≤ T_A ≤ +70°C (MC3417/18)		—	± 4.5	± 8.0	—	—	—	
– 55°C ≤ T_A ≤ +125°C (MC3517/18)		—	± 5.5	± 10	—	—	—	
Digital Output Voltage								Vdc
(I_{OL} = 3.6 mA)	V_{OL}	—	0.1	0.4	—	0.1	0.4	
(I_{OH} = – 0.35 mA)	V_{OH}	V_{CC} – 1.0	V_{CC} – 0.2	—	V_{CC} – 1.0	V_{CC} – 0.2	—	
Syllabic Filter Applied Voltage (Pin 3) (Figure 2)	$V_{I(Syl)}$	+ 3.2	—	V_{CC}	+ 3.2	—	V_{CC}	Vdc
Integrating Current (Figure 2)	I_{Int}							
(I_{GC} = 12 μA)		8.0	10	12	8.0	10	12	μA
(I_{GC} = 1.5 mA) (All except 3418P,DW)		1.45	1.5	1.55	1.45	1.5	1.55	mA
(MC3418P,DW)		—	—	—	1.42	1.5	1.58	mA
(I_{GC} = 3.0 mA)		2.75	3.0	3.25	2.75	3.0	3.25	mA
Dynamic Integrating Current Match	$V_{O(Ave)}$							mV
(I_{GC} = 1.5 mA) Figure 6								
(All except MC3418P,DW)		—	± 100	± 250	—	± 100	± 250	
(MC3418P,DW)		—	—	—	—	± 100	± 280	
Input Current — High Logic State	I_{IH}							μA
(V_{IH} = 18 V)								
Digital Data Input		—	—	+ 5.0	—	—	+ 5.0	
Clock Input		—	—	+ 5.0	—	—	+ 5.0	
Encode/$\overline{\text{Decode}}$ Input		—	—	+ 5.0	—	—	+ 5.0	
Input Current — Low Logic State	I_{IL}							μA
(V_{IL} = 0 V)								
Digital Data Input		—	—	– 10	—	—	– 10	
Clock Input		—	—	– 360	—	—	– 360	
Encode/$\overline{\text{Decode}}$ Input		—	—	– 36	—	—	– 36	
Clock Input, V_{IL} = 0.4 V		—	—	– 72	—	—	– 72	

NOTE 2 Dynamic total loop offset (ΣV_{offset}) equals V_{IO} (comparator) (Figure 3) minus V_{IOX} (Figure 5). The input offset voltages of the analog comparator and of the integrator amplifier include the effects of input offset current through the input resistors. The slope polarity switch current mismatch appears as an average voltage across the 10 k integrator resistor. For the MC3417/MC3517, the clock frequency is 16 kHz. For the MC3418/MC3518, the clock frequency is 32 kHz. Idle channel performance is guaranteed if this dynamic total loop offset is less than one-half of the change in integrator output voltage during one clock cycle (ramp step size). Laser trimming is used to insure good idle channel performance.

DEFINITIONS AND FUNCTION OF PINS

Pin 1 — Analog Input

This is the analog comparator inverting input where the voice signal is applied. It may be ac or dc coupled depending on the application. If the voice signal is to be level shifted to the internal reference voltage, then a bias resistor between Pins 1 and 10 is used. The resistor is used to establish the reference as the new dc average of the ac coupled signal. The analog comparator was designed for low hysteresis (typically less than 0.1 mV) and high gain (typically 70 dB).

Pin 2 — Analog Feedback

This is the noninverting input to the analog signal comparator within the IC. In an encoder application it should be connected to the analog output of the encoder circuit. This may be Pin 7 or a low pass filter output connected to Pin 7. In a decode circuit Pin 2 is not used and may be tied to $V_{CC}/2$ on Pin 10, ground or left open.

The analog input comparator has bias currents of 1.5 μA max, thus the driving impedances of Pins 1 and 2 should be equal to avoid disturbing the idle channel characteristics of the encoder.

Pin 3 — Syllabic Filter

This is the point at which the syllabic filter voltage is returned to the IC in order to control the integrator step size. It is an NPN input to an op amp. The syllabic filter consists of an RC network between Pins 11 and 3. Typical time constant values of 6.0 ms to 50 ms are used in voice codecs.

Pin 4 — Gain Control Input

The syllabic filter voltage appears across C_S of the syllabic filter and is the voltage between V_{CC} and Pin 3. The active voltage to current (V–I) converter drives Pin 4 to the same voltage at a slew rate of typically 0.5 V/μs. Thus the current injected into Pin 4 (I_{GC}) is the syllabic filter voltage divided by the R_X resistance. Figure 7 shows the relationship between I_{GC} (x-axis) and the integrating current, I_{Int} (y-axis). The discrepancy, which is most significant at very low currents, is due to circuitry within the slope polarity switch which enables trimming to a low total loop offset. The R_X resistor is then varied to adjust the loop gain of the codec, but should be no larger than 5.0 kΩ to maintain stability.

Pin 5 — Reference Input

This pin is the noninverting input of the integrator amplifier. It is used to reference the dc level of the output signal. In an encoder circuit it must reference the same voltage as Pin 1 and is tied to Pin 10.

Pin 6 — Filter Input

This inverting op amp input is used to connect the integrator external components. The integrating current (I_{Int}) flows into Pin 6 when the analog input (Pin 1) is high with respect to the analog feedback (Pin 2) in the encode mode or when the digital data input (Pin 13) is high in the decode mode. For the opposite states, I_{Int} flows out of Pin 6. Single integration systems require a capacitor and resistor between Pins 6 and 7. Multipole configurations will have different circuitry. The resistance between Pins 6 and 7 should always be between 8.0 kΩ and 13 kΩ to maintain good idle channel characteristics.

Pin 7 — Analog Output

This is the integrator op amp output. It is capable of driving a 600-ohm load referenced to $V_{CC}/2$ to +6.0 dBm and can otherwise be treated as an op amp output. Pins 5, 6, and 7 provide full access to the integrator op amp for designing integration filter networks. The slew rate of the internally compensated integrator op amp is typically 0.5 V/μs. Pin 7 output is current limited for both polarities of current flow at typically 30 mA.

Pin 8 — V_{EE}

The circuit is designed to work in either single or dual power supply applications. Pin 8 is always connected to the most negative supply.

Pin 9 — Digital Output

The digital output provides the results of the delta modulator's conversion. It swings between V_{CC} and V_{EE} and is CMOS or TTL compatible. Pin 9 is inverting with respect to Pin 1 and non-inverting with respect to Pin 2. It is clocked on the falling edge of Pin 14. The typical 10% to 90% rise and fall times are 250 ns and 50 ns respectively for V_{CC} = 12 V and C_L = 25 pF to ground.

Pin 10 — $V_{CC}/2$ Output

An internal low impedance mid-supply reference is provided for use of the MC3417/18 in single supply applications. The internal regulator is a current source and must be loaded with a resistor to insure its sinking capability. If a +6.0 dBmo signal is expected across a 600 ohm input bias resistor, then Pin 10 must sink 2.2 V/600 Ω = 3.66 mA. This is only possible if Pin 10 sources 3.66 mA into a resistor normally and will source only the difference under peak load. The reference load resistor is chosen accordingly. A 0.1 μF bypass capacitor from Pin 10 to V_{EE} is also recommended. The $V_{CC}/2$ reference is capable of sourcing 10 mA and can be used as a reference elsewhere in the system circuitry.

Pin 11 — $\overline{\text{Coincidence Output}}$

The duty cycle of this pin is proportional to the voltage across C_S. The coincidence output will be low whenever the content of the internal shift register is all 1s or all 0s. In the MC3417 the register is 3 bits long while the MC3418 contains a 4 bit register. Pin 11 is an open collector of an NPN device and requires a pull-up resistor.

2

If the syllabic filter is to have equal charge and discharge time constants, the value of R_P should be much less than R_S. In systems requiring different charge and discharge constants, the charging constant is $R_S C_S$ while the decaying constant is $(R_S + R_P)C_S$. Thus longer decays are easily achievable. The NPN device should not be required to sink more than 3.0 mA in any configuration. The typical 10% to 90% rise and fall times are 200 ns and 100 ns respectively for R_L = 4.0 kΩ to +12 V and C_L = 25 pF to ground.

Pin 12 — Digital Threshold

This input sets the switching threshold for Pins 13, 14, and 15. It is intended to aid in interfacing different logic families without external parts. Often it is connected to the $V_{CC}/2$ reference for CMOS interface or can be biased two diode drops above V_{EE} for TTL interface.

Pin 13 — Digital Data Input

In a decode application, the digital data stream is applied to Pin 13. In an encoder it may be unused or may be used to transmit signaling message under the control of Pin 15. It is an inverting input with respect to Pin 9. When Pins 9 and 13 are connected, a toggle flip-flop is formed and a forced idle channel pattern can be transmitted. The digital data input level should be maintained for 0.5 μs before and after the clock trigger for proper clocking.

Pin 14 — Clock Input

The clock input determines the data rate of the codec circuit. A 32K bit rate requires a 32 kHz clock. The switching threshold of the clock input is set by Pin 12. The shift register circuit toggles on the falling edge of the clock input. The minimum width for a positive-going pulse on the clock input is 300 ns, whereas for a negative-going pulse, it is 900 ns.

Pin 15 — Encode/$\overline{\text{Decode}}$

This pin controls the connection of the analog input comparator and the digital input comparator to the internal shift register. If high, the result of the analog comparison will be clocked into the register on the falling edge at Pin 14. If low, the digital input state will be entered. This allows use of the IC as an encoder/decoder or simplex codec without external parts. Furthermore, it allows non-voice patterns to be forced onto the transmission line through Pin 13 in an encoder.

Pin 16 — V_{CC}

The power supply range is from 4.75 to 16.5 volts between Pin V_{CC} and V_{EE}.

FIGURE 1 — POWER SUPPLY CURRENT

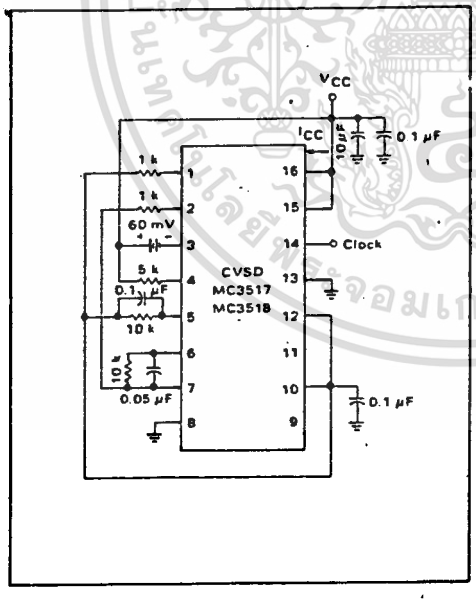
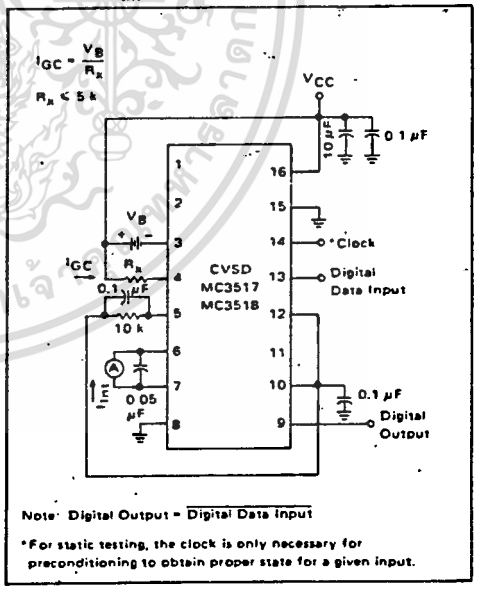


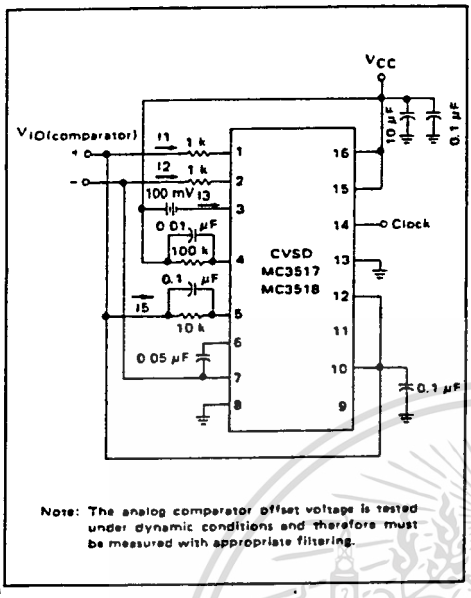
FIGURE 2 — I_{GCR}, GAIN CONTROL RANGE and I_{int} — INTEGRATING CURRENT



Note: Digital Output = $\overline{\text{Digital Data Input}}$

*For static testing, the clock is only necessary for preconditioning to obtain proper state for a given input.

FIGURE 3 – INPUT BIAS CURRENTS, ANALOG COMPARATOR OFFSET VOLTAGE AND CURRENT



Note: The analog comparator offset voltage is tested under dynamic conditions and therefore must be measured with appropriate filtering.

FIGURE 4 – INTEGRATOR AMPLIFIER OFFSET VOLTAGE AND CURRENT

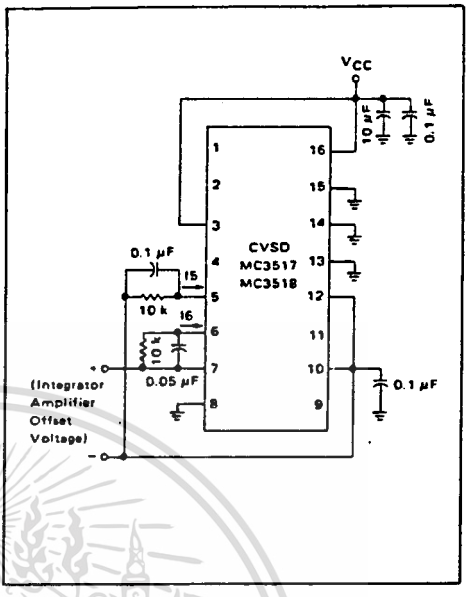
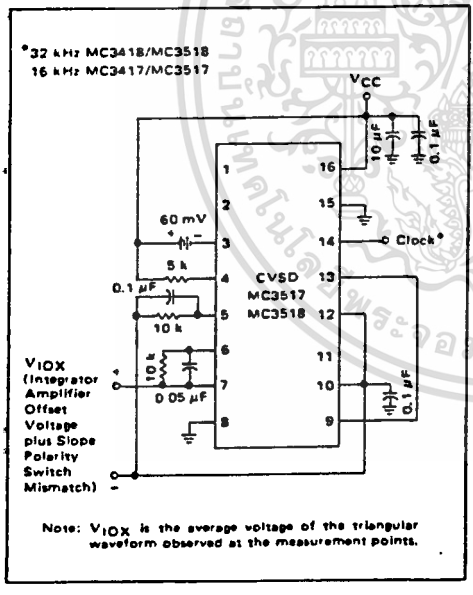
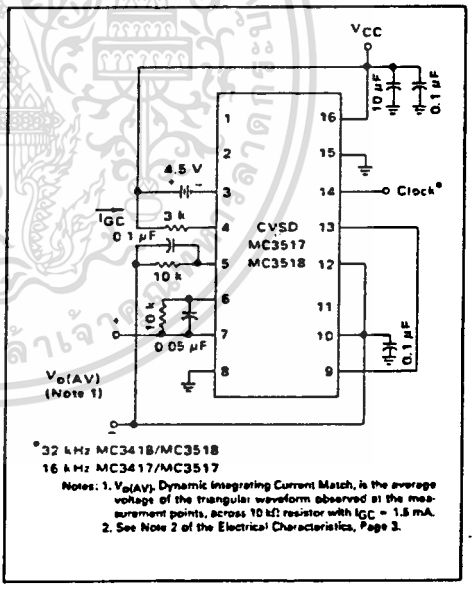


FIGURE 5 – V/I CONVERTER OFFSET VOLTAGE, V_{IO} and V_{IOX}



Note: V_{IOX} is the average voltage of the triangular waveform observed at the measurement points.

FIGURE 6 – DYNAMIC INTEGRATING CURRENT MATCH



Notes: 1. $V_{o(AV)}$, Dynamic Integrating Current Match, is the average voltage of the triangular waveform observed at the measurement points, across 10 kΩ resistor with I_{GC} = 1.5 mA.
2. See Note 2 of the Electrical Characteristics, Page 3.

2

TYPICAL PERFORMANCE CURVES

FIGURE 7 – TYPICAL I_{Int} versus I_{GC} (Mean ± 2σ)

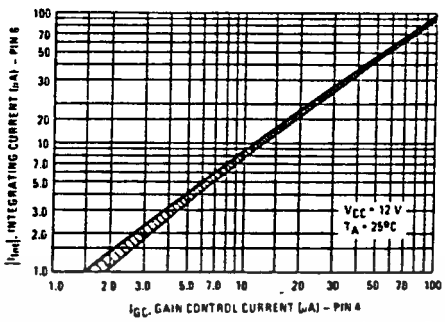


FIGURE 8 – NORMALIZED DYNAMIC INTEGRATING CURRENT MATCH versus V_{CC}

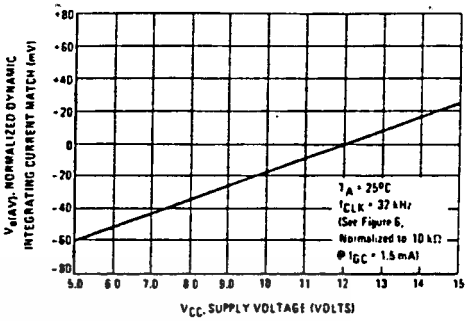


FIGURE 9 – NORMALIZED DYNAMIC INTEGRATING CURRENT MATCH versus CLOCK FREQUENCY

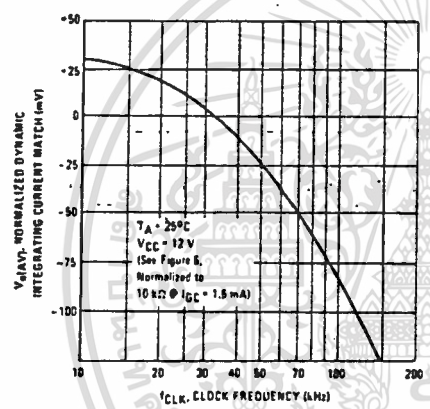


FIGURE 10 – DYNAMIC TOTAL LOOP OFFSET versus CLOCK FREQUENCY

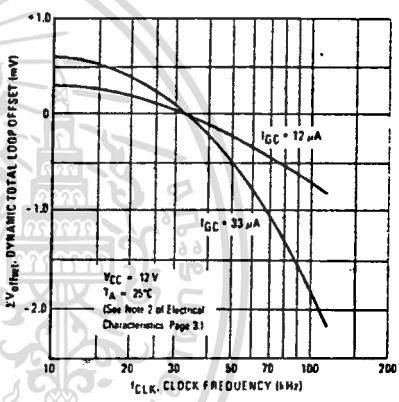


FIGURE 11 – BLOCK DIAGRAM OF THE CVSD ENCODER

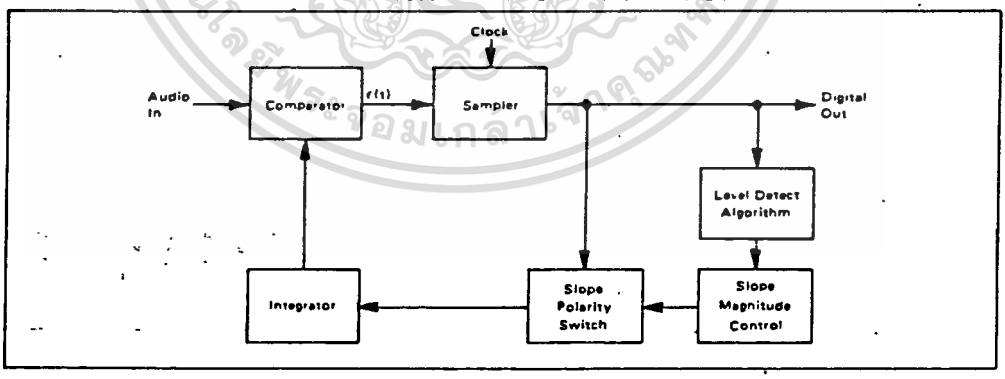


FIGURE 12 – CVSD WAVEFORMS

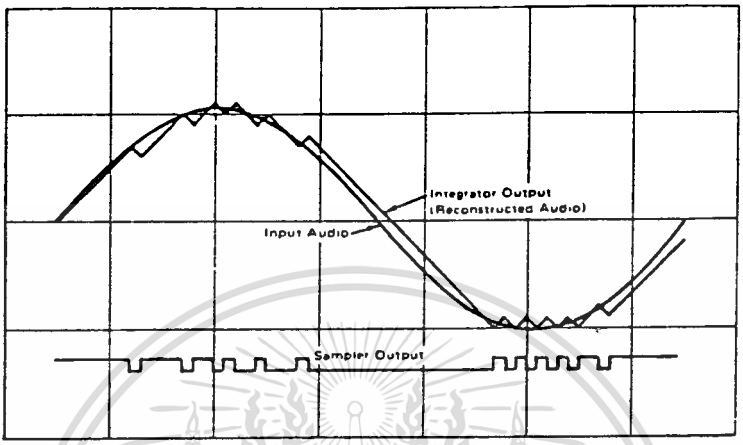
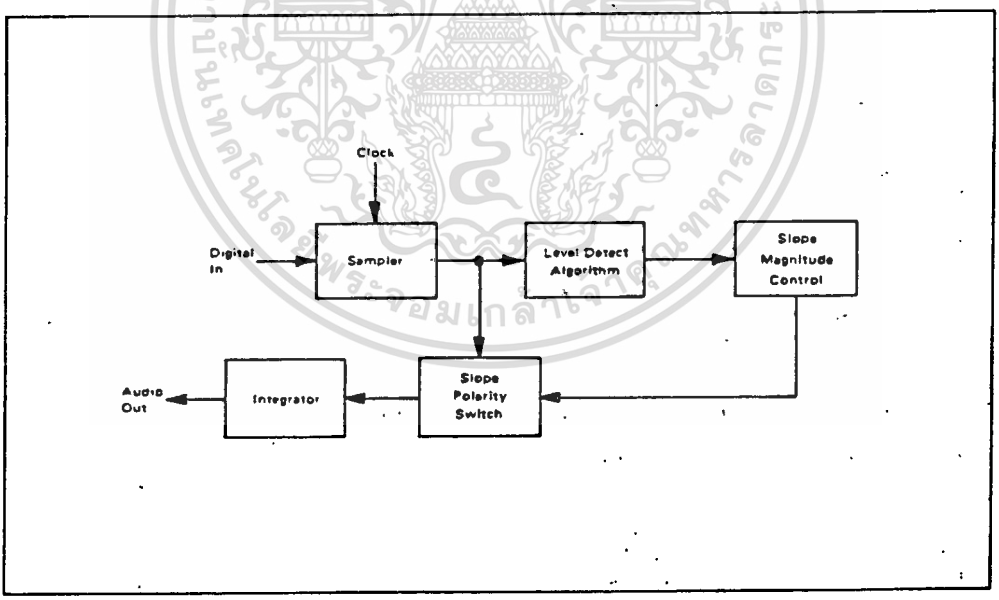
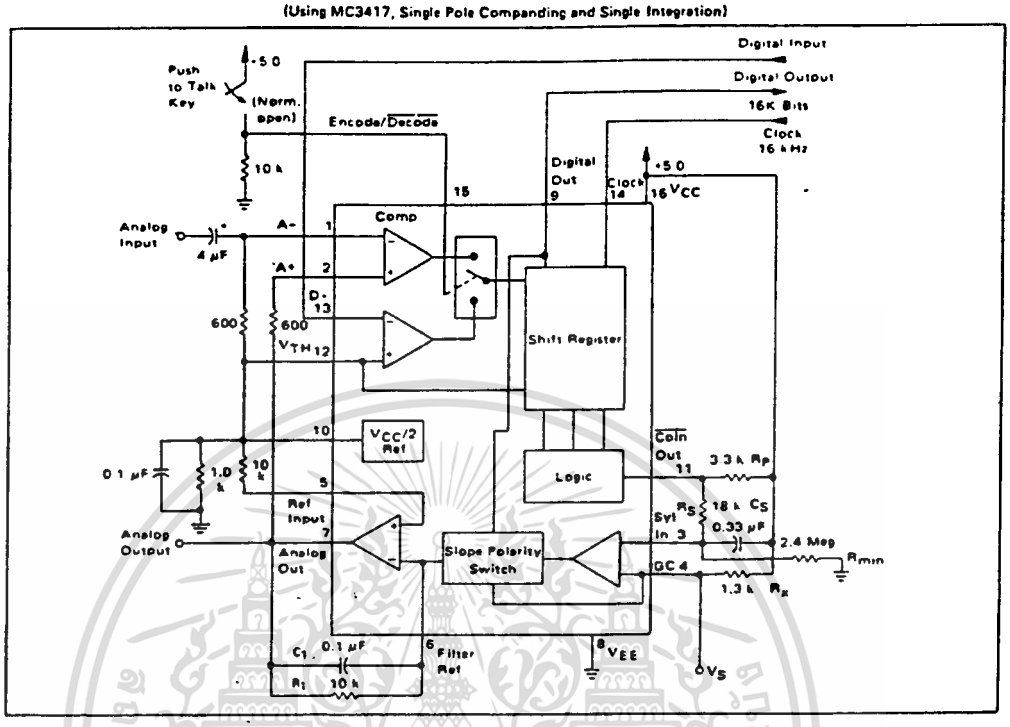


FIGURE 13 – BLOCK DIAGRAM OF THE CVSD DECODER



2

FIGURE 14 – 16 kHz SIMPLEX VOICE CODEC
(Using MC3417, Single Pole Companding and Single Integration)



CIRCUIT DESCRIPTION

The continuously variable slope delta modulator (CVSD) is a simple alternative to more complex conventional conversion techniques in systems requiring digital communication of analog signals. The human voice is analog, but digital transmission of any signal over great distance is attractive. Signal/noise ratios do not vary with distance in digital transmission and multiplexing, switching and repeating hardware is more economical and easier to design. However, instrumentation A to D converters do not meet the communications requirements. The CVSD A to D is well suited to the requirements of digital communications and is an economically efficient means of digitizing analog inputs for transmission.

The Delta Modulator

The innermost control loop of a CVSD converter is a simple delta modulator. A block diagram CVSD Encoder is shown in Figure 11. A delta modulator consists of a comparator in the forward path and an integrator in the feedback path of a simple control loop. The inputs to the comparator are the input analog signal and the integrator output. The comparator output reflects the sign of the difference between the input voltage and the integrator output. That sign bit is the digital output and also controls the direction of ramp in the integrator. The comparator is normally clocked so as to produce a synchronous and band limited digital bit stream.

If the clocked serial bit stream is transmitted, received, and delivered to a similar integrator at a remote point, the remote integrator output is a copy of the transmitting control loop integrator output. To the extent that the integrator at the transmitting locations tracks the input signal, the remote receiver reproduces the input signal. Low pass filtering at the receiver output will eliminate most of the quantizing noise, if the clock rate of the bit stream is an octave or more above the bandwidth of the input signal. Voice bandwidth is 4 kHz and clock rates from 8 k and up are possible. Thus the delta modulator digitizes and transmits the analog input to a remote receiver. The serial, unframed nature of the data is ideal for communications networks. With no input at the transmitter, a continuous one zero alternation is transmitted. If the two integrators are made leaky, then during any loss of contact the receiver output decays to

CIRCUIT DESCRIPTION (continued)

zero and receive restart begins without framing when the receiver reacquires. Similarly a delta modulator is tolerant of sporadic bit errors. Figure 12 shows the delta modulator waveforms while Figure 13 shows the corresponding CVSD decoder block diagram.

The Companding Algorithm

The fundamental advantages of the delta modulator are its simplicity and the serial format of its output. Its limitations are its ability to accurately convert the input within a limited digital bit rate. The analog input must be band limited and amplitude limited. The frequency limitations are governed by the nyquist rate while the amplitude capabilities are set by the gain of the integrator.

The frequency limits are bounded on the upper end; that is, for any input bandwidth there exists a clock frequency larger than that bandwidth which will transmit the signal with a specific noise level. However, the amplitude limits are bounded on both upper and lower ends. For a signal level, one specific gain will achieve an optimum noise level. Unfortunately, the basic delta modulator has a small dynamic range over which the noise level is constant.

The continuously variable slope circuitry provides increased dynamic range by adjusting the gain of the integrator. For a given clock frequency and input bandwidth the additional circuitry increases the delta modulator's dynamic range. External to the basic delta modulator is an algorithm which monitors the past few outputs of the delta modulator in a simple shift register. The register is 3 or 4 bits long depending on the application. The accepted CVSD algorithm simply monitors the contents of the shift register and indicates if it contains all 1s or 0s. This condition is called coincidence. When it occurs, it indicates that the gain of the integrator is too small. The coincidence output charges a single pole low pass filter. The voltage output of this syllabic filter controls the integrator gain through a pulse amplitude modulator whose other input is the sign bit or up/down control.

The simplicity of the all ones, all zeros algorithm should not be taken lightly. Many other control algorithms using the shift register have been tried. The key to the accepted algorithm is that it provides a measure of the average power or level of the input signal. Other techniques provide more instantaneous information about the shape of the input curve. The purpose of the algorithm is to control the gain of the integrator and to increase the dynamic range. Thus a measure of the average input level is what is needed.

The algorithm is repeated in the receiver and thus the level data is recovered in the receiver. Because the algorithm only operates on the past serial data, it changes the nature of the bit stream without changing the channel bit rate.

The effect of the algorithm is to compand the input signal. If a CVSD encoder is played into a basic delta modulator, the output of the delta modulator will reflect the shape of the input signal but all of the output will be at an equal level. Thus the algorithm at the output is needed to restore the level variations. The bit stream in the channel is as if it were from a standard delta modulator with a constant level input.

The delta modulator encoder with the CVSD algorithm provides an efficient method for digitizing a voice input in a manner which is especially convenient for digital communciations requirements.

APPLICATIONS INFORMATION

CVSD DESIGN CONSIDERATIONS

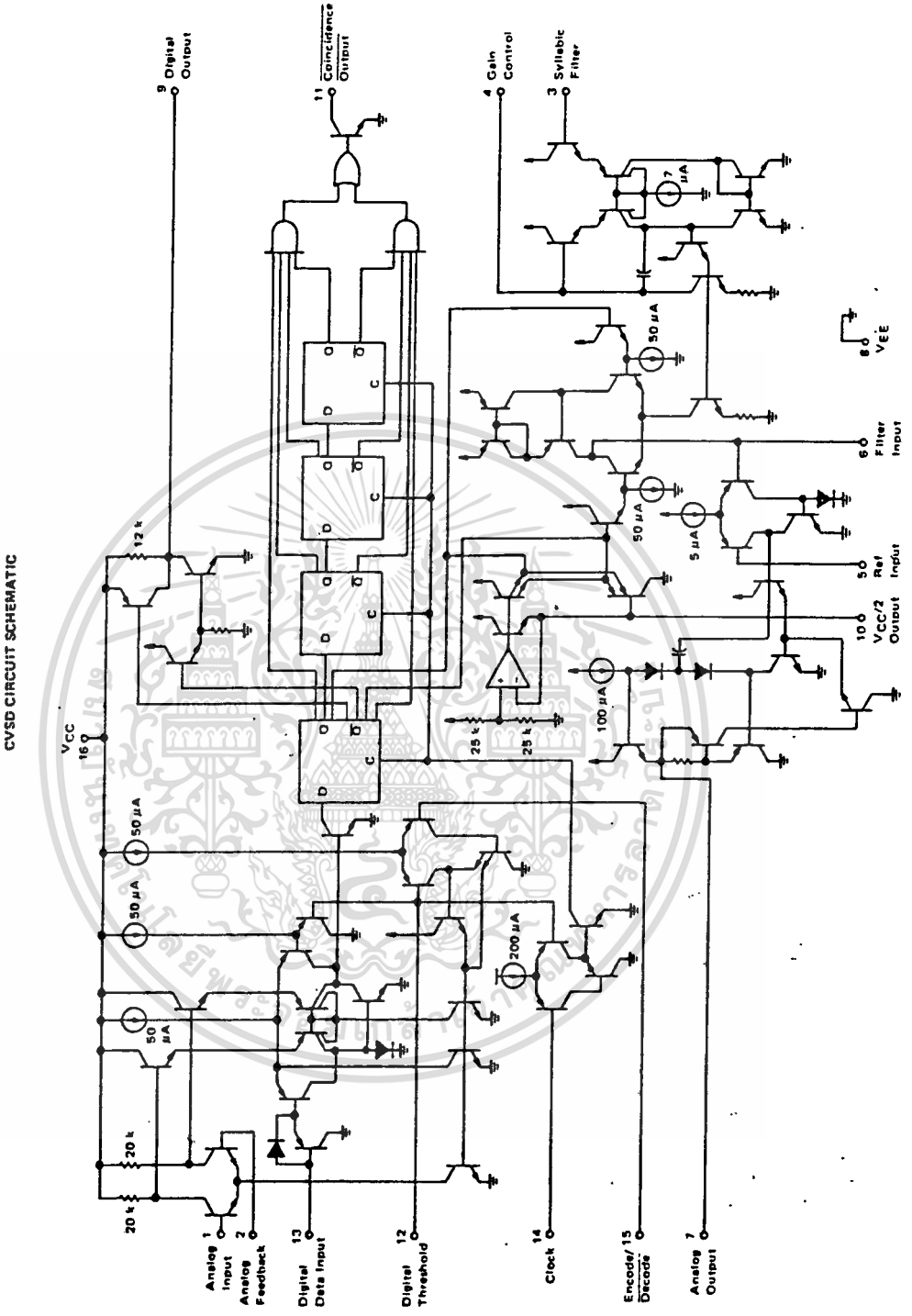
A simple CVSD encoder using the MC3417 or MC3418 is shown in Figure 14. These ICs are general purpose CVSD building blocks which allow the system designer to tailor the encoder's transmission characteristics to the application. Thus, the achievable transmission capabilities are constrained by the fundamental limitations of delta modulation and the design of encoder parameters. The performance is not dictated by the internal configuration of the MC3417 and MC3418. There are seven design considerations involved in designing these basic CVSD building blocks into a specific codec application, and they are as follows:

1. Selection of clock rate
2. Required number of shift register bits
3. Selection of loop gain
4. Selection of minimum step size
5. Design of integration filter transfer function
6. Design of syllabic filter transfer function
7. Design of low pass filter at the receiver

The circuit in Figure 14 is the most basic CVSD circuit possible. For many applications in secure radio or other intelligible voice channel requirements, it is entirely sufficient. In this circuit, items 5 and 6 are reduced to their simplest form. The syllabic and integration filters are both single pole networks. The selection of items 1 through 4 govern the codec performance.

2

CVSD CIRCUIT SCHEMATIC



CVSD DESIGN CONSIDERATIONS (continued)

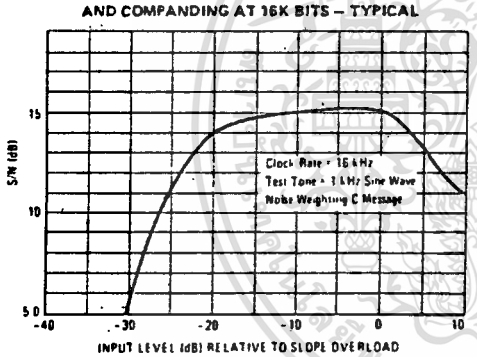
Layout Considerations

Care should be exercised to isolate all digital signal paths (Pins 9, 11, 13, and 14) from analog signal paths (Pins 1–7 and 10) in order to achieve proper idle channel performance.

Clock Rate

With minor modifications the circuit in Figure 14 may be operated anywhere from 9.6 kHz to 64 kHz clock rates. Obviously the higher the clock rate the higher the S/N performance. The circuit in Figure 14 typically produces the S/N performance shown in Figure 15. The selection of clock rate is usually dictated by the bandwidth of the transmission medium. Voice bandwidth systems will require no higher than 9600 Hz. Some radio systems will allow 12 kHz. Private 4-wire telephone systems are often operated at 16 kHz and commercial telephone performance can be achieved at 32K bits and above. Other codecs may use bit rates up to 200K bits/sec.

FIGURE 15 – SIGNAL-TO-NOISE PERFORMANCE OF MC3417 WITH SINGLE INTEGRATION, SINGLE-POLE AND COMPANDING AT 16K BITS – TYPICAL



Shift Register Length (Algorithm)

The MC3417 has a three-bit algorithm and the MC3418 has a four-bit algorithm. For clock rates of 16 kHz and below, the 3-bit algorithm is well suited. For 32 kHz and higher clock rates, the 4-bit system is preferred. Since the algorithm records a fixed past history of the input signal, a longer shift register is required to obtain the same internal history. At 16 bits and below, the 4-bit algorithm will produce a slightly wider dynamic range at the expense of level change response. Basically the MC3417 is designed for low bit rate systems and the MC3418 is intended for high performance, high bit rate system. At bit rates above 64K bits either part will work well.

Selection of Loop Gain

The gain of the circuit in Figure 14 is set by resistor R_X. R_X must be selected to provide the proper integrator step size for high level signals such that the companding ratio does not exceed about 25%. The companding ratio is the active low duty cycle of the coincidence output on Pin 11 of the codec circuit. Thus the system gain is dependent on:

1. The maximum level and frequency of the input signal.
2. The transfer function of the integration filter.

For voice codecs the typical input signal is taken to be a sine wave at 1 kHz of 0 dBmo level. In practice, the useful dynamic range extends about 6 dB above the design level. In any system the companding ratio should not exceed 30%.

To calculate the required step size current, we must describe the transfer characteristics of the integration filter. In the basic circuit of Figure 14, a single pole of 160 Hz is used.

$$R_1 = 10\ k\Omega,\ C_1 = 0.1\ \mu F$$

$$\frac{V_o}{I_i} = \frac{1}{C(S + 1/RC)} \equiv \frac{K}{S + \omega_o}$$

$$\omega_o = 2\pi f$$

$$10^3 = \omega_o = 2\pi f$$

$$f = 159.2\ Hz$$

Note that the integration filter produces a single-pole response from 300 to 3 kHz. The current required to move the integrator output a specific voltage from zero is simply:

$$I_i = \frac{V_o}{R_1} + \left(C_1 \times \frac{dV_o}{dt}\right)$$

Now a 0 dBmo sine wave has a peak value of 1.0954 volts. In 1/8 of a cycle of a sine wave centered around the zero crossing, the sine wave changes by approximately its peak value. The CVSD step should trace that change. The required current for a 0 dBm 1 kHz sine wave is:

$$I_i = \frac{1.1\ V}{*2(10\ k\Omega)} + \frac{0.1\ \mu F(1.1)}{0.125\ ms} = 0.935\ mA$$

*The maximum voltage across R_1 when maximum slew is required is:

$$\frac{1.1\ V}{2}$$

Now the voltage range of the syllabic filter is the power supply voltage, thus:

$$R_X = 0.25(V_{CC})\ \frac{1}{0.935\ mA}$$

A similar procedure can be followed to establish the proper gain for any input level and integration filter type.

Minimum Step Size

The final parameter to be selected for the simple codec in Figure 14 is idle channel step size. With no input signal, the digital output becomes a one-zero alternating pattern and the analog output becomes a small triangle wave. Mismatches of internal currents and offsets limit the minimum step size which will produce a perfect idle channel pattern. The MC3417 is tested to ensure that a 20 mVp-p minimum step size at 16 kHz will attain a proper idle channel. The idle channel step size must be twice the specified total loop offset if a one-zero idle pattern is desired. In some applications a much smaller minimum step size (e.g., 0.1 mV) can produce quiet performance without providing a 1-0 pattern.

To set the idle channel step size, the value of R_{min} must be selected. With no input signal, the slope control algorithm is inactive. A long series of ones or zeros never occurs. Thus, the voltage across the syllabic filter capacitor (C_S) would decay to zero. However, the voltage divider of R_S and R_{min} (see Figure 14) sets the minimum allowed voltage across the syllabic filter capacitor. That voltage must produce the desired ramps at the analog output. Again we write the filter input current equation:

$$I_i = \frac{V_o}{R_1} + C\frac{dV_o}{dt}$$

For values of V_o near $V_{CC}/2$ the V_o/R term is negligible; thus

$$I_i = C_S\frac{\Delta V_o}{\Delta T}$$

where ΔT is the clock period and ΔV_o is the desired peak-to-peak value of the idle output. For a 16K-bit system using the circuit in Figure 14

$$I_i = \frac{0.1\,\mu F \;\; 20\,mV}{62.5\,\mu s} = 33\,\mu A$$

The voltage on C_S which produces a 33 μA current is determined by the value of R_x.

$I_i R_x = V_{Smin}$; for 33 μA, V_{Smin} = 41.6 mV

In Figure 14 R_S is 18 kΩ. That selection is discussed with the syllabic filter considerations. The voltage divider of R_S and R_{min} must produce an output of 41.6 mV.

$$V_{CC}\frac{R_S}{R_S + R_{min}} = V_{Smin} \qquad R_{min} \simeq 2.4\,M\Omega$$

Having established these four parameters – clock rate, number of shift register bits, loop gain and minimum step size – the encoder circuit in Figure 14 will function at near optimum performance for input levels around 0 dBm.

INCREASING CVSD PERFORMANCE

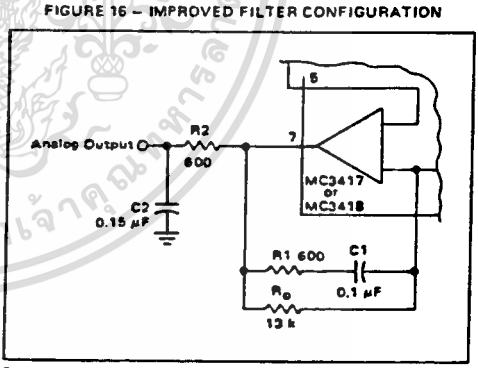
Integration Filter Design

The circuit in Figure 14 uses a single-pole integration network formed with a 0.1 μF capacitor and a 10 kΩ resistor. It is possible to improve the performance of the circuit in Figure 14 by 1 or 2 dB by using a two-pole integration network. The improved circuit is shown.

The first pole is still placed below 300 Hz to provide the 1/S voice content curve and a second pole is placed somewhere above the 1 kHz frequency. For telephony circuits, the second pole can be placed above 1.8 kHz to exceed the 1633 touchtone frequency. In other communication systems, values as low as 1 kHz may be selected. In general, the lower in frequency the second pole is placed, the greater the noise improvement. Then, to ensure the encoder loop stability, a zero is added to keep the phase shift less than 180°. This zero should be placed slightly above the low-pass output filter break frequency so as not to reduce the effectiveness of the second pole. A network of 235 Hz, 2 kHz and 5.2 kHz is typical for telephone applications while 160 Hz, 1.2 kHz and 2.8 kHz might be used in voice only channels. (Voice only channels can use an output low-pass filter which breaks at about 2.5 kHz.) The two-pole network in Figure 16 has a transfer function of:

$$\frac{V_o}{I_i} = \frac{R_0R_1\left(S + \frac{1}{R_1C_1}\right)}{R_2C_2(R_0 + R_1)\left(S + \frac{1}{(R_0 + R_1)C_1}\right)S + \left(\frac{1}{R_2C_2}\right)}$$

FIGURE 16 – IMPROVED FILTER CONFIGURATION



These component values are for the telephone channel circuit poles described in the text. The R2, C2 product can be provided with different values of R and C. R2 should be chosen to be equal to the termination resistor on Pin 1.

INCREASING CVSD PERFORMANCE (continued)

Thus the two poles and the zero can be selected arbitrarily as long as the zero is at a higher frequency than the first pole. The values in Figure 16 represent one implementation of the telephony filter requirement.

The selection of the two-pole filter network effects the selection of the loop gain value and the minimum step size resistor. The required integrator current for a given change in voltage now becomes:

$$I_i = \frac{V_o}{R_0} + \left(\frac{R_2C_2}{R_0} + \frac{R_1C_1}{R_0} + C_1\right)\frac{\Delta V_o}{\Delta T} + \left(R_2C_2C_1 + \frac{R_1C_1R_2C_2}{R_0}\right)\frac{\Delta V_o{}^2}{\Delta T^2}$$

The calculation of desired gain resistor R_x then proceeds exactly as previously described.

Syllabic Filter Design

The syllabic filter in Figure 14 is a simple single-pole network of 18 kΩ and 0.33 μF. This produces a 6.0 ms time constant for the averaging of the coincidence output signal. The voltage across the capacitor determines the integrator current which in turn establishes the step size. The integrator current and the resulting step size determine the companding ratio and the S/N performance. The companding ratio is defined as the voltage across C_S/V_{CC}.

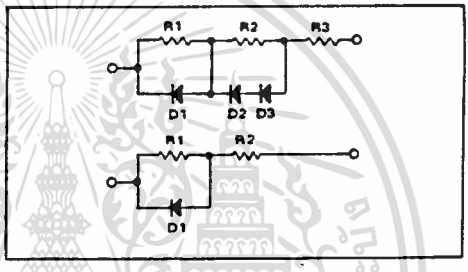
The S/N performance may be improved by modifying the voltage to current transformation produced by R_x. If different portions of the total R_x are shunted by diodes, the integrator current can be other than $(V_{CC} - V_S)/R_x$. These breakpoint curves must be designed experimentally for the particular system application. In general, one would wish that the current would double with input level. To design the desired curve, supply current to Pin 4 of the codec from an external source. Input a signal level and adjust the current until the S/N performance is optimum. Then record the syllabic filter voltage and the current. Repeat this for all desired signal levels. Then derive the resistor diode network which produces that curve on a curve tracer.

Once the network is designed with the curve tracer, it is then inserted in place of R_x in the circuit and the forced optimum noise performance will be achieved from the active syllabic algorithm.

Diode breakpoint networks may be very simple or moderately complex and can improve the usable dynamic range of any codec. In the past they have been used in high performance telephone codecs.

Typical resistor-diode networks are shown in Figure 17.

FIGURE 17 – RESISTOR-DIODE NETWORKS



If the performance of more complex diode networks is desired, the circuit in Figure 18 should be used. It simulates the companding characteristics of nonlinear R_x elements in a different manner.

Output Low Pass Filter

A low pass filter is required at the receiving circuit output to eliminate quantizing noise. In general, the lower the bit rate, the better the filter must be. The filter in Figure 20 provides excellent performance for 12 kHz to 40 kHz systems.

TELEPHONE CARRIER QUALITY CODEC USING MC3418

Two specifications of the integrated circuit are specifically intended to meet the performance requirements of commercial telephone systems. First, slope polarity switch current matching is laser trimmed to guarantee proper idle channel performance with 5 mV minimum step size and a typical 1% current match from 15 μA to 3 mA. Thus a 300 to 1 range of step size variation is possible. Second, the MC3418 provides the four-bit algorithm currently used in subscriber loop telephone systems. With these specifications and the circuit of Figure 18, a telephone quality codec can be mass produced.

The circuit in Figure 18 provides a 30 dB S/Nc ratio over 50 dB of dynamic range for a 1 kHz test tone at a 37.7K bit rate. At 37.7K bits, 40 voice channels may be multiplexed on a standard 1.544 megabit T1 facility. This codec has also been tested for 10^{-7} error rates with asynchronous and synchronous data up to 2400 baud and for reliable performance with DTMF signaling. Thus, the design is applicable in telephone quality subscriber loop carrier systems, subscriber loop concentrators and small PABX installations.

The Active Companding Network

The unique feature of the codec in Figure 18 is the step size control circuit which uses a companding ratio reference, the present step size, and the present syllabic filter output to establish the optimum companding ratios and step sizes for any given input level. The companding ratio of a CVSD codec is defined as the duty cycle of the coincidence output. It is the parameter measured by the syllabic filter and is the voltage across C_S divided by the voltage swing of the coincidence output. In Figure 18, the voltage swing of Pin 11 is 6.0 volts. The operating companding ratio is analoged by the voltage between Pins 10 and 4 by means of the virtual short across Pins 3 and 4 of the V to I op amp within the integrated circuit. Thus, the instantaneous companding ratio of the codec is always available at the negative input of A1.

The diode D1 and the gain of A1 and A2 provide a companding ratio reference for any input level. If the output of A2 is more than 0.7 volts below $V_{CC}/2$, then the positive input of A1 is $(V_{CC}/2 - 0.7)$. The on diode drop at the input of A1 represents a 12% companding ratio (12% = 0.7 V/6.0 V).

The present step size of the operating codec is directly related to the voltage across R_X, which established the integrator current. In Figure 18, the voltage across R_X is amplified by the differential amplifier A2 whose output is single ended with respect to Pin 10 of the IC.

For large signal inputs, the step size is large and the output of A2 is lower than 0.7 volts. Thus D1 is fully on. The present step size is not a factor in the step size control. However, the difference between 12% companding ratio and the instantaneous companding ratio at Pin 4 is amplified by A1. The output of A1 changes the voltage across R_X in a direction which reduces the difference between the companding reference and the operating ratio by changing the step size. The ratio of R4 and R3 determines how closely the voltage at Pin 4 will be forced to 12%. The selection of R3 and R4 is initially experimental. However, the resulting companding control is dependent on R_X, R3, R4, and the full diode drop D1. These values are easy to reproduce from codec to codec.

For small input levels, the companding ratio reference becomes the output of A2 rather than the diode drop. The operating companding ratio on Pin 4 is then compared to a companding ratio smaller than 12% which is determined by the voltage drop across R_X and the gain of A2 and A1. The gain of A2 is also experimentally determined, but once determined, the circuitry is easily

FIGURE 18 – TELEPHONE QUALITY DELTAMOD CODER

(Both double integration and active companding control are used to obtain improved CVSD performance. Laser trimming of the integrated circuit provides reliable idle channel and step size range characteristics.)

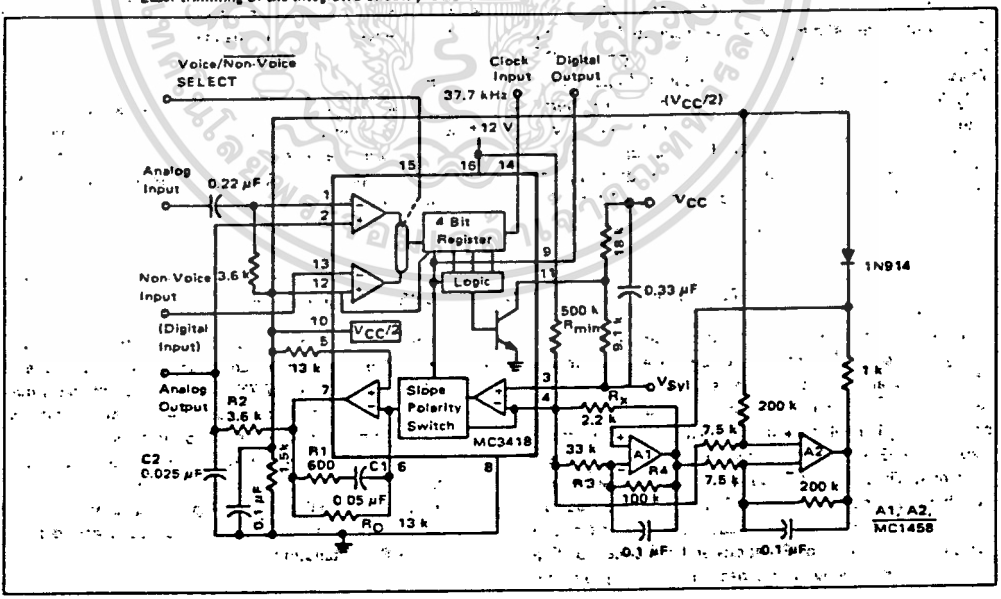
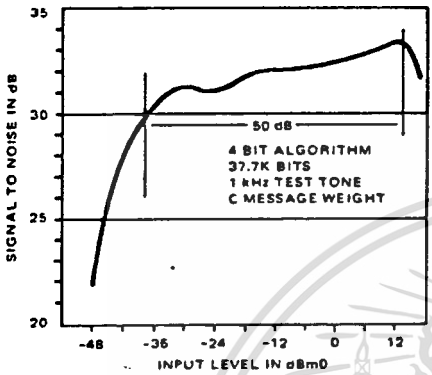
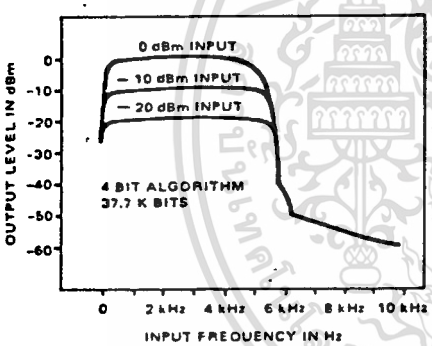


FIGURE 19 – SIGNAL-TO-NOISE PERFORMANCE AND FREQUENCY RESPONSE
(Showing the improvement realized with the circuit in Figure 18.)

a SIGNAL-TO-NOISE PERFORMANCE OF TELEPHONY QUALITY DELTAMODULATOR



b. FREQUENCY RESPONSE versus INPUT LEVEL (SLOPE OVERLOAD CHARACTERISTIC)



repeated.

With no input signal, the companding ratio at Pin 4 goes to zero and the voltage across R_X goes to zero. The voltage at the output of A2 becomes zero since there is no drop across R_X. With no signal input, the actively controlled step size vanished.

The minimum step size is established by the 500 k resistor between V_{CC} and $V_{CC}/2$ and is therefore independently selectable.

The signal to noise results of the active companding network are shown in Figure 19. A smooth 2 dB drop is realized from +12 dBm to –24 under the control of A1. At –24 dBm, A2 begins to degenerate the companding reference and the resulting step size is reduced so as to extend the dynamic range of the codec by 20 dBm.

The slope overload characteristic is also shown. The active companding network produces improved performance with frequency. The 0 dBm slope overload point is raised to 4.8 kHz because of the gain available in controlling the voltage across R_X. The curves demonstrate that the level linearity has been maintained or improved.*

The codec in Figure 18 is designed specifically for 37.7K bit systems. However, the benefits of the active companding network are not limited to high bit rate systems. By modifying the crossover region (changing the gain of A2), the active technique may be used to improve the performance of lower bit rate systems.

The performance and repeatability of the codec in Figure 18 represents a significant step forward in the art and cost of CVSD codec designs.

*A larger value for C2 is required in the decoder circuit than in the encoder to adjust the level linearity with frequency. In Figure 18, 0.050 μF would work well.

FIGURE 20 – HIGH PERFORMANCE ELLIPTIC FILTER FOR CVSD OUTPUT

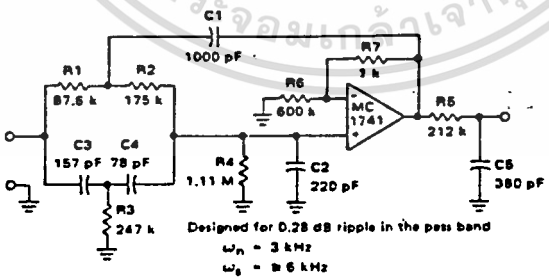
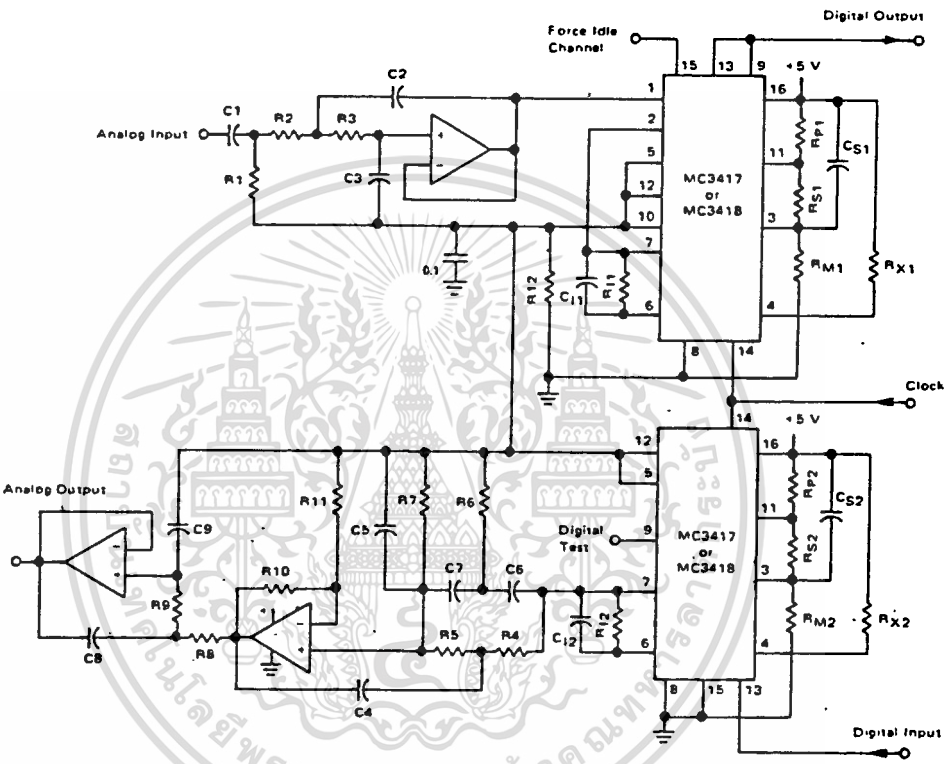


FIGURE 21 – FULL DUPLEX/32K BIT CVSD VOICE CODEC USING MC3517/18 AND MC3503/6 OP AMP



Codec Components

R_{X1}, R_{X2} – 3.3 kΩ
R_{P1}, R_{P2} – 3.3 kΩ
R_{S1}, R_{S2} – 100 kΩ
R_{I1}, R_{I2} – 20 kΩ
R_{12} – 1 kΩ
R_{M1}, R_{M2} – 5 MΩ (MC3417)
Minimum step size = 20 mV
R_{M1}, R_{M2} – 15 MΩ (MC3418)
Minimum step size = 6 mV

C_{S1}, C_{S2} – 0.05 µF
C_{I1}, C_{I2} – 0.05 µF

2 MC3417 (or MC3418)
1 MC3403 (or MC3406)

Note: All Res. 5%
All Cap. 5%

Input Filter Specifications

12 dB/Octave Rolloff above 3.3 kHz
6 dB/Octave Rolloff below 50 Hz

Output Filter Specifications

Break Frequency – 3.3 kHz
Stop Band – 9 kHz
Stop Band Atten. – 50 dB
Rolloff – > 40 dB/Octave

Filter Components

R1 – 965 Ω	C1 – 3.3 µF
R2 – 72 kΩ	C2 – 837 pF
R3 – 72 kΩ	C3 – 536 pF
R4 – 63.46 kΩ	C4 – 1000 pF
R5 – 127 kΩ	C5 – 222 pF
R6 – 365.5 kΩ	C6 – 77 pF
R7 – 1.645 MΩ	C7 – 38 pF
R8 – 72 kΩ	C8 – 837 pF
R9 – 72 kΩ	C9 – 536 pF
R10 – 29.5 kΩ	
R11 – 72 kΩ	

Note All Res 0.1% to 1%.
All Cap. 1.0%

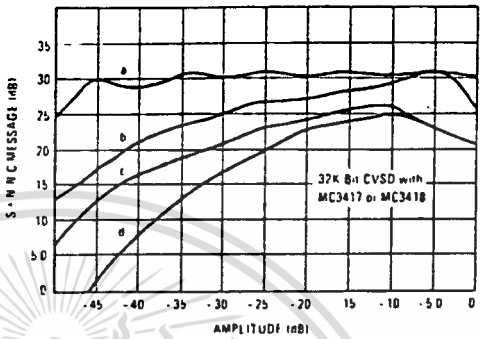
COMPARATIVE CODEC PERFORMANCE

The salient feature of CVSD codecs using the MC3517 and MC3518 family is versatility. The range of codec complexity tradeoffs and bit rate is so wide that one cannot grasp the interdependency of parameters for voice applications in a few pages.

Design of a specific codec must be tailored to the digital channel bandwidth, the analog bandwidth, the quality of signal transmission required and the cost objectives. To illustrate the choices available, the data in Figure 22 compares the signal-to-noise ratios and dynamic range of various codec design options at 32K bits. Generally, the relative merits of each design feature will remain intact in any application. Lowering the bit rate will reduce the dynamic range and noise performance of all techniques. As the bit rate is increased, the overall performance of each technique will improve and the need for more complex designs diminishes.

Non-voice applications of the MC3517 and MC3518 are also possible. In those cases, the signal bandwidth and amplitude characteristics must be defined before the specification of codec parameters can begin. However, in general, the design can proceed along the lines of the voice applications shown here, taking into account the different signal bandwidth requirements.

FIGURE 22 – COMPARATIVE CODEC PERFORMANCE – SIGNAL-TO-NOISE RATIO FOR 1 kHz TEST TONE



These curves demonstrate the improved performance obtained with several codec designs of varying complexity.

Curve a — Complex companding and double integration (Figure 18 — MC3418)
Curve b — Double integration (Figure 14 using Figure 16 — MC3418)
Curve c — Single integration (Figure 14 — MC3418) with 6.0 mV step size
Curve d — Single integration (Figure 14 — MC3417) with 25 mV step size