เครื่องพล็อตแพทเทิร์นของสายอากาศ

ANTENNA PATTERN RECORDER

โดย

นาย อดิศักดิ์ แซ็งสาริกิจ 34.132138

นาย เพทาย สิงห์คราม 34.132158

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาอุตสาหกรรมศาสตร์บัณฑิต

สาขาเทคโนโลยีโทรคมนาคม ภาควิชา เทคนิคอุตสาหกรรม

สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหารลาดกระบัง

หัวข้อปริญญานิพนธ์ : เครื่องพลอตแพทเทิร์นของสายอากาศ

ผู้จัดทำ : นาย อดิศักดิ์ แข็งสาริกิจ

นาย เพทาย สิงห์คราม

อาจารย์ที่ปรึกษา : อ. กฤตากร กล่อมการ

ภาควิชา : เทคนิคอุตสาหกรรม

ปีการศึกษา : 2535

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหารลาดกระบัง อนุมัติให้นับปริญญานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาอุตสาหกรรมศาสตร์บัณฑิต

..... คณบดีคณะวิศวกรรมศาสตร์

( )

คณะกรรมการสอบปริญญานิพนธ์

..... ประธานกรรมการ

( )

..... กรรมการ

( )

..... กรรมการ

( )

..... กรรมการ

( )

ปีการศึกษา 2535

เครื่องพลือดแพทเทิร์นของสายอากาศ

โดย

นาย อดิศักดิ์ แข็งสาริกิจ 34.132138

นาย เพทาย สิงห์คราม 34.132158

อาจารย์ที่ปรึกษา

อ. กฤตากร กล่อมการ

ปริญญานิพนธ์ ปีการศึกษา 2535

ภาควิชา เทคนิคอุตสาหกรรม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหารลาดกระบัง

เรื่อง เครื่องพล็อตแพทเทิร์นของสายอากาศ (ANTENNA PATTERN RECORDER)

ผู้จัดทำ

นาย อดิศักดิ์ แข็งสาริกิจ 34.132138

นาย เพทาย สิงห์คราม 34.132158

..... อาจารย์ที่ปรึกษา

( )

หัวข้อปริญญานิพนธ์	:	เครื่องพล็อตแพทเทิร์นของสายอากาศ
ผู้จัดทำ	:	นาย อดิศักดิ์ แข็งสาริกิจ
		นาย เพทาย สิงห์คราม
อาจารย์ที่ปรึกษา	:	อ. กฤดากร กล่อมการ
ภาควิชา	:	เทคนิคอุตสาหกรรม
ปีการศึกษา	:	2535

บทคัดย่อ

ปริญญานิพนธ์นี้เป็นการนำเอาคอมพิวเตอร์มาช่วยในการวิเคราะห์หาคุณสมบัติของสายอากาศในรูปแบบของการแพร่กระจายคลื่น โดยมีหลักการให้สายอากาศที่จะทำการวิเคราะห์มาทำการหมุนและถูกต่อกับเครื่องวัดความแรงคลื่นวิทยุ จากนั้นก็นำเอาสัญญาณที่ได้มาแปลงเป็นสัญญาณดิจิตอลเพื่อนำค่าที่ได้มาทำขบวนการทางคณิตศาสตร์มาพล็อตแพทเทิร์นของสายอากาศ

สารบัญ

บทที่ 1

บทนำ

ในปัจจุบันการสื่อสารด้วยคลื่นแม่เหล็กไฟฟ้าได้มีความสำคัญต่อชีวิตประจำวัน อย่างมากดังจะเห็นได้จากการใช้อย่างแพร่หลายของโทรศัพท์ระบบเซลลูล่าร์ และการรับสัญญาณโทรทัศน์โดยตรงจากดาวเทียม เป็นต้น สายอากาศจัดว่าเป็นส่วนประกอบที่สำคัญชิ้นหนึ่งของระบบนี้ ซึ่งสายอากาศจะทำหน้าที่รับ - ส่งคลื่นแม่เหล็กไฟฟ้าโดยจะเปลี่ยนจากคลื่นแม่เหล็กไฟฟ้า เป็นสัญญาณทางไฟฟ้าในสายอากาศรับ และจะเปลี่ยนจากสัญญาณทางไฟฟ้า เป็นคลื่นแม่เหล็กไฟฟ้าแพร่กระจายออกไปในอากาศในสายอากาศส่ง ซึ่งจะต้องทำให้การรับ - ส่งคลื่นแม่เหล็กไฟฟ้าของสายอากาศมีประสิทธิภาพมากที่สุด โดยทั้งนี้ขึ้นอยู่กับส่วนประกอบต่าง ๆ มากมาย อาทิเช่น สายตัวนำ ชนิดของสายอากาศ โลหะที่ใช้ทำสายอากาศ และทิศทางในการรับ - ส่งคลื่นแม่เหล็กไฟฟ้า เป็นต้น แต่ในที่นี้จะกล่าวถึงการวิเคราะห์ทิศทาง ในการแพร่กระจายคลื่นแม่เหล็กไฟฟ้าของสายอากาศซึ่งเรียกว่า แพทเทิร์นของสายอากาศ

แพทเทิร์นของสายอากาศจะเป็นส่วนบ่งบอกถึงทิศทางการรับ - ส่งคลื่นแม่เหล็กไฟฟ้าว่าทิศทางใดรับ - ส่งคลื่นแม่เหล็กไฟฟ้าได้ดี ซึ่งจะวัดมีหน่วยเป็น dBi เทียบกับจุดที่แพร่กระจายคลื่นแม่เหล็กไฟฟ้าของสายอากาศของเครื่องส่ง ส่วนทิศทางใดที่ไม่มีการแพร่กระจายคลื่นแม่เหล็กไฟฟ้าซึ่งเรียกว่า จุดที่เกิดนัล (null) แพทเทิร์นของสายอากาศนั้นมีอยู่ด้วยกัน 3 แบบ คือ แบบวงกลม แบบสี่เหลี่ยม และแบบ 3 มิติ เป็นต้น

Project นี้จะเป็นการนำไมโครคอมพิวเตอร์ขนาด 16 บิทมาประยุกต์ใช้งาน ซึ่งจะมีหลักการทำงานโดยรวม ๆ คือ ในส่วนของด้านส่ง เครื่องไมโครคอมพิวเตอร์ขนาด 16 บิทจะทำงานเป็นส่วนควบคุมการหมุน ของสายอากาศทางด้านรับ โดยจะหมุนที่ละ 1 องศา และทางด้านรับจะต่อกับ Field Strieght ซึ่งเป็นเครื่องรับคลื่นแม่เหล็กไฟฟ้า จากเครื่องส่ง โดยจะมีลักษณะพิเศษคือจ่ายเอาท์พุทออกมาเป็นสัญญาณทางไฟฟ้าที่คงที่ วัดออกมาเป็น dBi จากนั้นเครื่องไมโครคอมพิวเตอร์ขนาด 16 บิทจะนำค่าที่ได้จากการรับจาก Field Strieght มาแปลงจากสัญญาณอนาลอกเป็นดิจิตอลเพื่อนำมาประมวลผลพล็อตเป็นแพทเทิร์นของสายอากาศ และเครื่องไมโครคอมพิวเตอร์ขนาด 16 บิท จะทำการหมุนของสายอากาศด้านรับ 1 องศา จากนั้นเครื่องไมโครคอมพิวเตอร์ขนาด 16 บิท จะนำค่ามาทำการพล็อตแพทเทิร์น ซึ่งเครื่องไมโครคอมพิวเตอร์ขนาด 16 บิทจะทำงานดังกล่าวไปจนครบ 360 องศาอย่างต่อเนื่อง

จากหลักการดังกล่าวสามารถแบ่งการทำงานส่วนใหญ่ออกเป็น 2 ประเภท คือทางฮาร์ดแวร์ และซอร์ฟแวร์ ซึ่งทางด้านฮาร์ดแวร์ จะแบ่งการทำงานออกเป็นส่วนย่อยอีกคือ การ์ดเชื่อมต่อกับ

เครื่องไมโครคอมพิวเตอร์ขนาด 16 บิท ภาคแปลงสัญญาณอนาลอกเป็นดิจิตอล และส่วนที่ใช้ขับชุดสเตปปิ้งมอเตอร์ เป็นต้น ส่วนทางด้านซอร์ฟแวร์ จะประกอบด้วยส่วนที่นำค่าจากสัญญาณดิจิตอลที่ได้จากการแปลงสัญญาณอนาลอกนำมาประมวลผลเป็นแพทเทิร์นของสายอากาศ และส่วนที่ทำหน้าที่ควบคุมสเตปปิ้งมอเตอร์

บทที่ 2

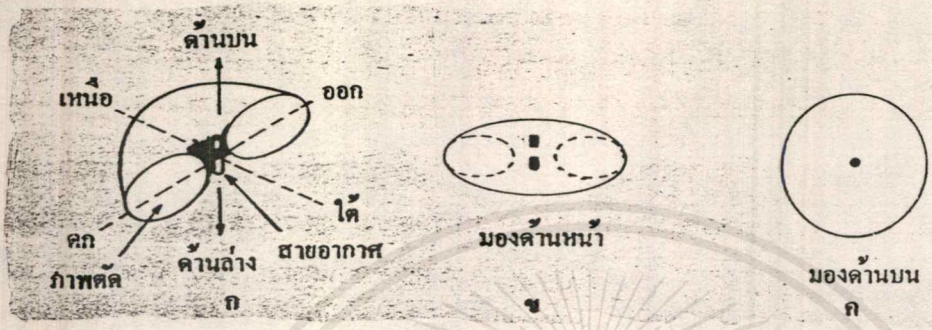
คุณสมบัติของสายอากาศทั่วไป

สายอากาศนั้นเป็นส่วนหนึ่งที่จะขาดไม่ได้ในวิทยุสื่อสารที่เกี่ยวข้องกับคลื่นวิทยุและโทรทัศน์ในปัจจุบัน ถ้าจะเปรียบเทียบระบบสายอากาศเข้ากับส่วนประกอบของไฟฟ้ากำลัง ก็เปรียบเสมือนกับสายส่งที่เชื่อมสถานีจ่ายไฟทางด้านต้นทางและสถานีรับปลายทาง จะต่างกันก็ตรงที่ว่าไม่ได้จ่ายหรือเป็นพลังงานไฟฟ้าผ่านทางสายส่ง แต่จ่ายและรับพลังงานในรูปที่เป็นคลื่นแม่เหล็กไฟฟ้าโดยมีอากาศเป็นตัวกลางเท่านั้น

เนื่องจากสายอากาศเป็นอุปกรณ์ทางไฟฟ้าประเภทพาสซีฟ (Passive) เพราะฉะนั้นการสึกหรอหรือเสื่อมคุณภาพ เมื่อเทียบกับหลอดหรือทรานซิสเตอร์แทบจะไม่มีเลย สาเหตุการสึกหรอจะเนื่องมาจากการผุกร่อนของโลหะเนื่องจากบรรยากาศที่เป็นกรดหรือด่าง หรือแรงลม เสถียรภาพของการติดตั้ง เพราะฉะนั้นการออกแบบ การติดตั้ง และการป้องกันการผุกร่อนจึงเป็นปัญหาสำคัญในการบำรุงรักษาสายอากาศ

รูปแบบของการแพร่คลื่นของสายอากาศ

ส่วนประกอบขั้นพื้นฐานของสายอากาศ ซึ่งจะส่งหรือรับคลื่นแม่เหล็กไฟฟ้าที่กระจายออกรอบตัวสายอากาศ ลักษณะรูปร่างของการกระจายพลังงานคลื่นแม่เหล็กไฟฟ้าเป็นรูปร่างต่าง ๆ กันเรียกว่าแพทเทิร์น ซึ่งจะเป็นตัวกำหนดค่าไดเรคติวิตี้ (Directivity) คือการกระจายพลังงานไปในทิศทางใดทิศทางหนึ่งมากกว่าในทิศทางอื่น ๆ ซึ่งมีความสำคัญในการรับหรือส่งในทิศทางเดียวและป้องกันการรบกวนจากสถานีอื่น ๆ

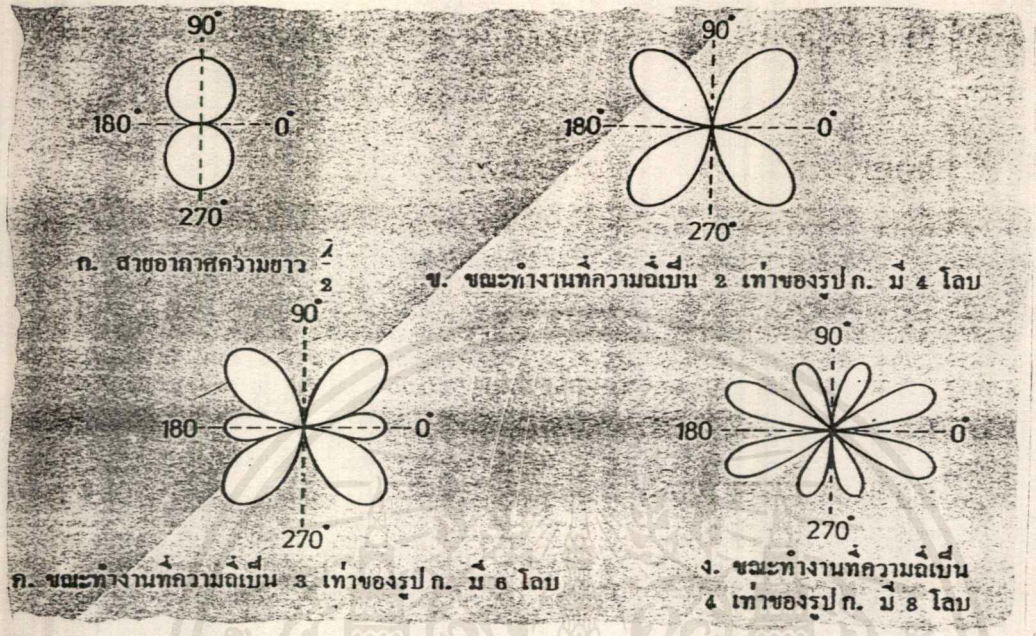


รูปที่ 1 แสดงรูปแบบของสายอากาศ

การหารูปร่างของแพทเทิร์นนั้นหาได้โดยวิธีทางคณิตศาสตร์ หรือ การบวกเวกเตอร์ที่ตำแหน่งต่าง ๆ ซึ่งจะไม่กล่าวถึงในตอนนี้ รูปแบบของการกระจายคลื่นของสายอากาศถ้าพลังงานถูกแพร่กระจายไปทั่ว ๆ ในทุกทิศทาง รูปแบบการแพร่กระจายคลื่นจะเป็นรูปทรงกลมซึ่งมีสายอากาศแบบไดโพลอยู่ตรงจุดศูนย์กลางดังรูปที่ 1 แสดงรูปแบบของสายอากาศความยาวคลื่นส่วนสองที่เรโซแนนซ์กับความถี่หนึ่ง ๆ

ความเข้มของการกระจายพลังงานจะลดลงในขณะที่ระยะทางจากสายอากาศเพิ่มขึ้น รูปแบบของสายอากาศแบบความยาวคลื่นส่วนสอง โดยทั่วไปจะคล้ายกับขนมโดนัท ซึ่งมีไดเรคติวีตี้และรูปแบบจะแปรผันกับความยาวทางไฟฟ้าของสายอากาศดังแสดงในรูปที่ 2 ซึ่งแสดงว่าไดโพลมีความยาวทางฟิสิกส์ (ความยาวจริง) เท่ากัน แต่ทำงานที่ความถี่ต่าง ๆ กัน จะสังเกตเห็นว่ารูปแบบที่ได้จะแตกต่างกันด้วย

โดยทั่วไปที่ความยาวของสายอากาศตัวหนึ่ง เมื่อให้ทำงานที่ความถี่มีค่ามากขึ้น จำนวนโลบ (Loop) จะมากขึ้นด้วย ซึ่งไม่เป็นผลดีนักในแง่การรับหรือส่ง เพราะกำลังที่ได้จะน้อยลง แต่ดีในแง่ของการติดตั้ง นอกจากนี้เพื่อปรับปรุงให้รูปแบบมีรูปร่างต่าง ๆ ตามที่ต้องการก็จะมีการเพิ่มส่วนประกอบอื่น ๆ รวมกับไดโพล เช่น ไดเรคเตอร์หรือตัวชี้นำ รีเฟรคเตอร์หรือตัวสะท้อน เป็นต้น



รูปที่ 2 แสดงรูปแบบของไดโพล

อัตราการขยายและสภาพการชี้นำของสายอากาศ

ในการวัดกำลังงาน เครื่องที่ใช้ในการวัดเป็นมิเตอร์วัดแรงดันที่โหลดที่รู้ค่าความต้านทาน แล้วคำนวณกำลังได้ว่า $P = V^2/R$ ถ้าคิดเป็นเดซิเบลจะได้ว่า

$$dB = 10 \log \left(\frac{vo^2 . R1}{vi^2 . R2} \right)$$

เมื่อ R2 เป็นความต้านทานทางด้านเอาพุทซึ่งเป็นโหลด

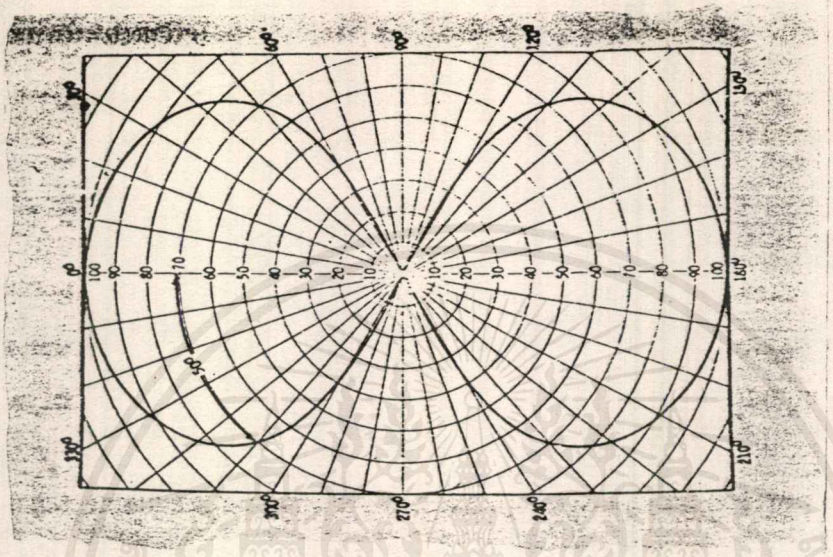
R1 เป็นความต้านทานทางด้านอินพุท

ถ้าให้ R2 เท่ากับ R1 จะได้ว่า

$$dB = 20 \log (vo/vi)$$

dB เป็นหน่วยที่วัดอัตราการขยายของสายอากาศโดยเทียบกับสายอากาศไอโซโทรปิค (isothopic antenna) ซึ่งสายอากาศแบบนี้เป็นจุดของต้นกำเนิดสัญญาณที่กระจายออกรอบทิศทาง หรือกล่าวอีกนัยหนึ่งว่า เป็นสายอากาศที่มีคุณสมบัติที่ให้ความเข้มของคลื่นสัญญาณ ณ จุดที่ห่างจากสายมีค่าเท่ากัน จะมีความเข้มของสัญญาณเท่ากัน โดยเปรียบเทียบสายอากาศแบบไอโซโทรปิคจะเปรียบได้กับหลอดไฟกลม ซึ่งให้ความสว่างเท่า ๆ กันทุก ๆ ทิศทาง โดยที่อาจจะออกแบบให้หลอดไฟสว่างเฉพาะจุดตามที่ต้องการให้มันสว่างได้ เช่น การใช้หลอดไฟยาวซึ่งจะให้แสงสว่างในด้านความยาวมากกว่าทางด้านข้าง สายอากาศมีตัวชี้นำ (director) ก็เสมือนกับใส่เลนส์นูนไว้ข้างหน้าหลอดไฟซึ่งก็จะพยายามบีบแสงให้เป็นลำพุ่งออกไป ทิศทางที่ลำแสงบีบออกไปนั้นสามารถจะทำให้เกิดความเข้มของแสงสว่างได้มากขึ้น สายอากาศที่ใส่ตัวสะท้อน (reflector) ก็เหมือนกับใส่กระจกไว้สะท้อนแสงให้พุ่งออกไปในทิศทางเดียว ผลก็จะทำให้ความเข้มของคลื่นวิทยุ ณ ทิศทางนั้นมีค่ามากขึ้น แต่มุมของการส่งก็จะแคบลง

สภาพการชี้นำเป็นคุณสมบัติที่สำคัญของสายอากาศ จะเป็นตัวบ่งบอกว่าสายอากาศจะรับสัญญาณได้ดีในทิศทางใดทิศทางหนึ่งหรืออาจเรียกได้ว่ารูปแบบของสภาพการชี้นำ ถึงแม้ว่าสายอากาศแบบยูนิไดเรคชั่นจะรับสัญญาณได้ดี สภาพการชี้นำจะแสดงการเปลี่ยนแปลงของความไวที่ทุก ๆ ทิศทาง โดยปกติจะแสดงโดยกราฟเชิงมุม รูปร่างโลบของแบบในเชิงมุม (Polar pattern) จะแสดงคุณสมบัติสภาพการชี้นำของสายอากาศไดโพลดังแสดงในรูปที่ 3 แสดงรูปแบบของสายอากาศแบบไดโพลในเทรมของแรงดันที่ตำแหน่งต่าง ๆ จะเป็นแบบสองทิศทาง (Bidirectional) มีโลบ (lobe) ที่มีขนาดเท่ากันทั้งด้านหน้าและด้านหลัง 2 ทิศทางที่ความไวมีค่ามากที่สุดอยู่ในแนวเส้นตั้งฉากกับสายอากาศ สังเกตเห็นว่าที่จุด 70.7% บนเส้นศูนย์องศาเรียกจุด ครึ่งกำลังงาน (Half power point) และจะเป็นตำแหน่งที่มีความกว้างของโลบมากที่สุด มุมที่แสดงที่จุดนี้เรียกว่าความกว้างของลำคลื่นเชิงมุม (beamwidth angle) นอกจากนี้ทิศทางที่รับได้น้อยที่สุดเรียกว่านัลโลบ (null lobe) หรืออาจกล่าวได้ว่าจุดที่รับสัญญาณไม่ได้เลย



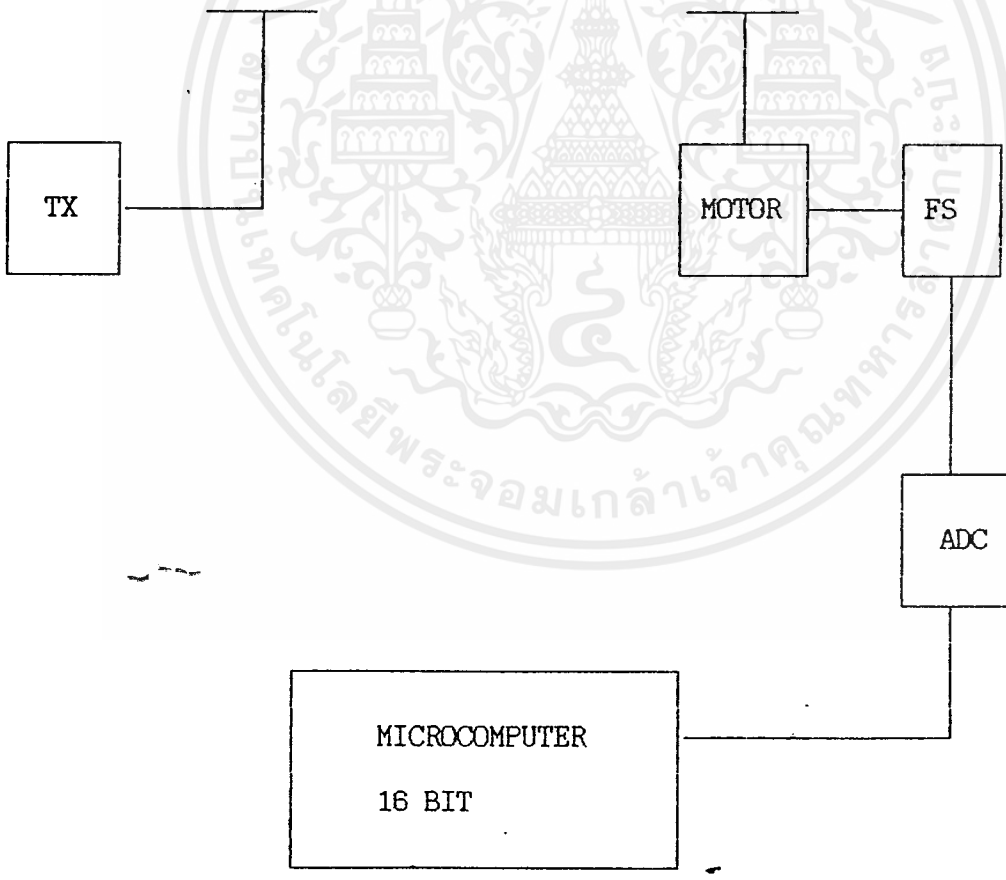
รูปที่ 3 แสดงรูปแบบของแพทเทิร์นของไดโพลมาตรฐาน

บทที่ 3

หลักการทำงานของ เครื่องพล็อตแพทเทิร์นของสายอากาศ

เครื่องพล็อตแพทเทิร์นของสายอากาศ โดยใช้เครื่องไมโครคอมพิวเตอร์ขนาด 16 บิทมาประยุกต์ใช้งานนั้นมีการทำงาน โดยส่วนใหญ่จะอยู่ที่เครื่องไมโครคอมพิวเตอร์ขนาด 16 บิท และส่วนของโปรแกรมที่ใช้ในการสั่งงานต่าง ๆ และเครื่องรับสัญญาณจาก เครื่องรับคลื่นแม่เหล็กไฟฟ้า (Field Strieght) รับค่ามาทำการพล็อตแพทเทิร์นของสายอากาศ เป็นต้น

ซึ่งการทำงานที่ได้กล่าวมาข้างต้นนี้ เราสามารถแบ่งการทำงานออก เป็นบล็อกไดอะแกรมได้ดัง รูปที่ 4



รูปที่ 4 แสดงบล็อกไดอะแกรมของ เครื่องพล็อตแพทเทิร์นของสายอากาศ

จากบล็อกไดอะแกรมของเครื่องพล็อตแพทเทิร์นของสายอากาศ ซึ่งมีหลักการทำงานโดยจะอธิบายออกมาเป็นบล็อกย่อย ๆ ได้ดังนี้ คือ

1. ทางด้านส่ง

1.1 เครื่องส่งคลื่นแม่เหล็กไฟฟ้า (TX)

เครื่องส่งคลื่นแม่เหล็กไฟฟ้า ซึ่งเป็นเครื่องส่งความถี่ในย่าน VHF ขึ้นไป โดยจะส่งคลื่นความถี่โดยที่ไม่มีสัญญาณเสียงหรือสัญญาณอื่น ๆ มามอดูเลทเข้าไปด้วยคือจะส่งเฉพาะความถี่พาหะ (RF) อย่างเดียว เนื่องจากจะได้มีการแพร่กระจายคลื่นแม่เหล็กไฟฟ้าที่ไม่มีการเปลี่ยนแปลงของความถี่พาหะ (RF) ซึ่งถ้านำสัญญาณเสียงหรือสัญญาณอื่น ๆ ที่มีการเปลี่ยนแปลงแล้วจะทำให้เครื่องรับคลื่นแม่เหล็กไฟฟ้า (Field Strieght) รับคลื่นแต่ละจุดมีค่าความแรงต่างกันจึงทำให้การวัด หรือการพล็อตแพทเทิร์นของสายอากาศนั้น มีการค่าต่างกันด้วยทั้งนี้เนื่องมาจากการมอดูเลทแต่ละชนิดนั้นมีค่าของเอาท์พุทที่มีการเปลี่ยนแปลงตลอด อาทิเช่น การมอดูเลทแบบ AM สัญญาณที่ออกมาทางเอาพุทนั้น ความถี่พาหะ (RF) จะมีการเปลี่ยนแปลงแอมปลิจูดของสัญญาณไปตามสัญญาณเสียงหรือสัญญาณอื่น ๆ ที่นำมามอดูเลทเข้าไป เป็นต้น

2. ทางด้านรับ

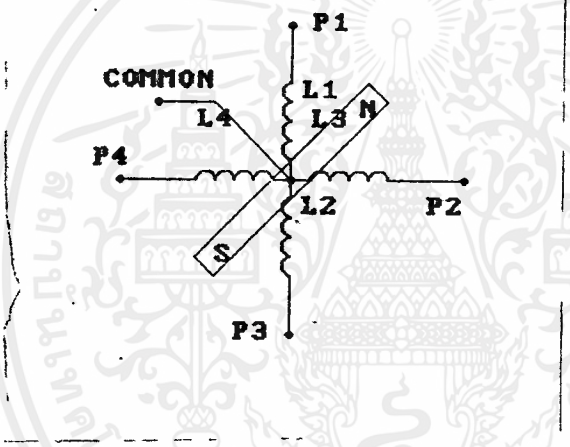
2.1 ชุด STEPPING MOTOR ที่ใช้ในการหมุนเสาสายอากาศ

สเตปปิงมอเตอร์เป็นมอเตอร์ทางาน โดยเปลี่ยนพลังงานไฟฟ้าแบบสวิทชิ่ง ซึ่งป้อนให้กับขดลวดสเตเตอร์ให้เป็นการเคลื่อนที่เชิงมุมบนแกนโรเตอร์ในลักษณะการหมุนทีละสเตปตามสัญญาณอินพุทของเฟสต่าง ๆ การเคลื่อนที่หรือหยุดในแต่ละสเตปจะมีความแม่นยำทางตำแหน่งสูงมาก แต่มีข้อจำกัดอยู่เหมือนกันคือขนาดของแรงบิดจะน้อยทำให้ไม่สามารถใช้ในงานหนักมากได้

สเตปปิงมอเตอร์นั้นมีอยู่ด้วยกันหลายชนิดพอแยกได้เป็น 3 ประเภทคือ

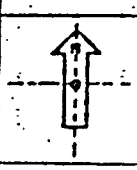
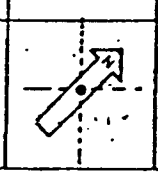
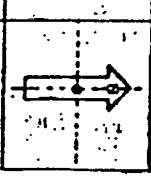
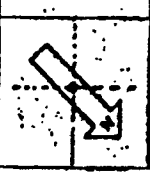
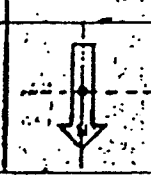
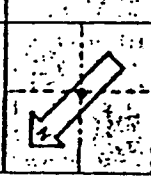
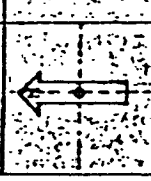
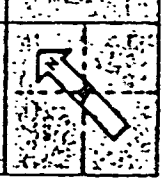
1. แบบแม่เหล็กถาวร (Permanent Magnet)
2. แบบแปรค่ารีลัคแตนซ์ (Variable reluctance)
3. แบบไฮบริด (Hybrid)

ส่วนรายระเอียดของสเตปปิ้งมอเตอร์ในแต่ละแบบนั้นสามารถศึกษา ได้จากหนังสือเฉพาะ เรื่อง แต่ในหัวข้อนี้เป็นการนำมาใช้ประโยชน์โดยการใช้พอร์ท 8255 การประยุกต์ใช้นั้น โดยการต่อกับพอร์ท 8255 เป็นตัวควบคุมการหมุนและทิศทางของมอเตอร์ (ในโหมด 0) ซึ่งรูปแบบของสเตปปิ้งมอเตอร์ที่ใช้งานมากพบบ่อย ๆ ก็คือ แบบไฮบริดโรเตอร์ทำด้วยแม่เหล็กถาวรและเป็นแบบชนิดขดลวดสเตเตอร์ 4 เฟสไฟฟ้าที่จ่ายให้กับขดลวดสเตเตอร์ ในแต่ละเฟสจะมีผลทำให้เกิดสนามแม่เหล็กในทิศทางนั้นๆ เกิดแรงผลักกับแม่เหล็กถาวรที่ตัวโรเตอร์ ไปในทิศทางหนึ่งการสั่งให้โรเตอร์หมุนในทิศทางใด ก็สามารถทำได้โดยการจ่ายกระแสไฟฟ้าแก่ขดสเตเตอร์ ในเฟสต่างๆ อย่างถูกต้องต่อไป รูปที่ 5 แสดงรูปโครงสร้างของสเตปปิ้งมอเตอร์ง่าย ๆ



รูปที่ 5 แสดงถึงโครงสร้างของสเตปปิ้งมอเตอร์ขนาด 4 เฟส และตำแหน่งของโรเตอร์ขณะจ่ายกระแสไฟฟ้าขดลวดที่ $\phi 1$ และ $\phi 2$

และจากสเตปปิ้งมอเตอร์มีขนาด 4 เฟสนี้ เราจะบังคับ ให้เกิดสเตปของการหมุนได้เป็น 3 ลักษณะขึ้นอยู่กับ การป้อนพัลซ์ดังแสดงในรูปที่ 6

เฟสที่จ่ายกระแสไฟฟ้า	ϕ_1	$\phi_1.\phi_2$	ϕ_2	$\phi_2.\phi_3$	ϕ_3	$\phi_3.\phi_4$	ϕ_4	$\phi_4.\phi_1$
ตำแหน่งโรเตอร์								

รูปที่ 6 แสดงมุมของโรเตอร์เมื่อมีการป้อนไฟแก่เฟสต่าง ๆ ของมอเตอร์ขนาด 4 เฟส

1) One - excitation หรือ half drive เป็นการจ่ายกระแสให้กับ สเตเตอร์ครั้งละหนึ่งเฟสคือ $\phi1,\phi2,\phi3,\phi4$ เรียงลำดับการหมุนเวียนกันไปแบบนี้แรงบิดจะน้อย ดังตาราง

No	$\phi1$	$\phi2$	$\phi3$	$\phi4$
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1

(1) One - excitation (กำลังครึ่งเดียว)

2) Two - excitaion หรือ full step เป็นการจ่ายกระแสให้กับ สเตเตอร์ครั้งละ 2 เฟสพร้อมกันได้แก่ $\phi1\phi2,\phi2\phi3,\phi3\phi4,\phi4\phi1$ หมุนเวียนกันไปแบบนี้แรงบิดที่ได้จะมากกว่าแบบแรก

No	$\phi1$	$\phi2$	$\phi3$	$\phi4$
1	1	1	0	0
2	0	1	1	0
3	0	0	1	1
4	1	0	0	1

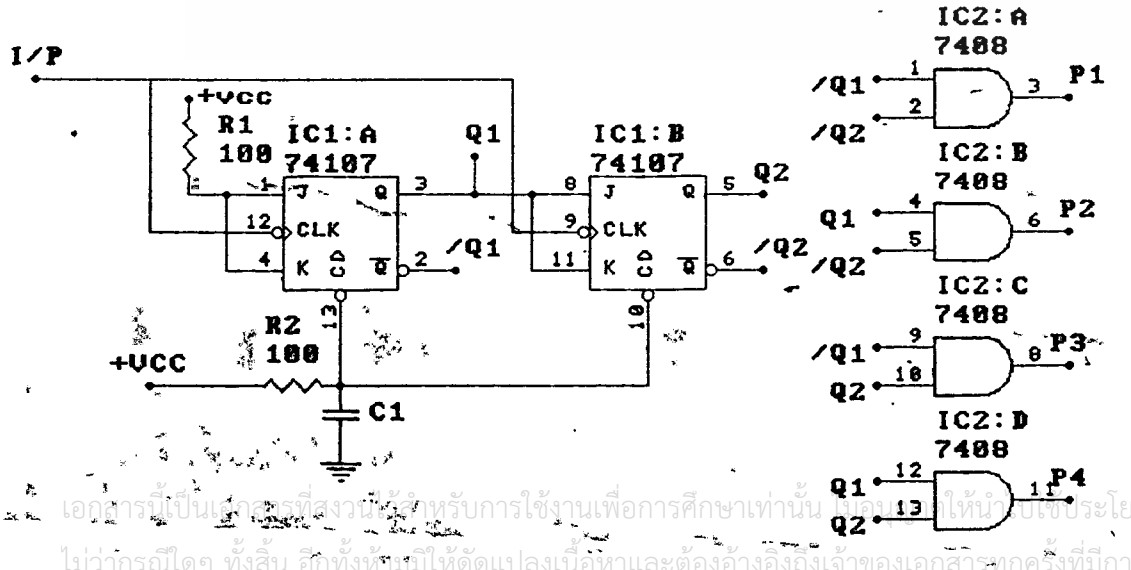
(2) Two-excitation(กำลังเต็ม)

3) one - two excitation หรือ half step เป็นการจ่ายกระกระแสให้กับขดลวดสเตเตอร์ 1 เฟสและ 2 เฟส สลับกันไป แบบนี้ทำให้จำนวนของสเตปเพิ่มขึ้นเป็น 2 เท่าของสองแบบแรก แต่แรงบิดจะน้อยกว่าแบบที่สองที่สอง เล็กน้อย

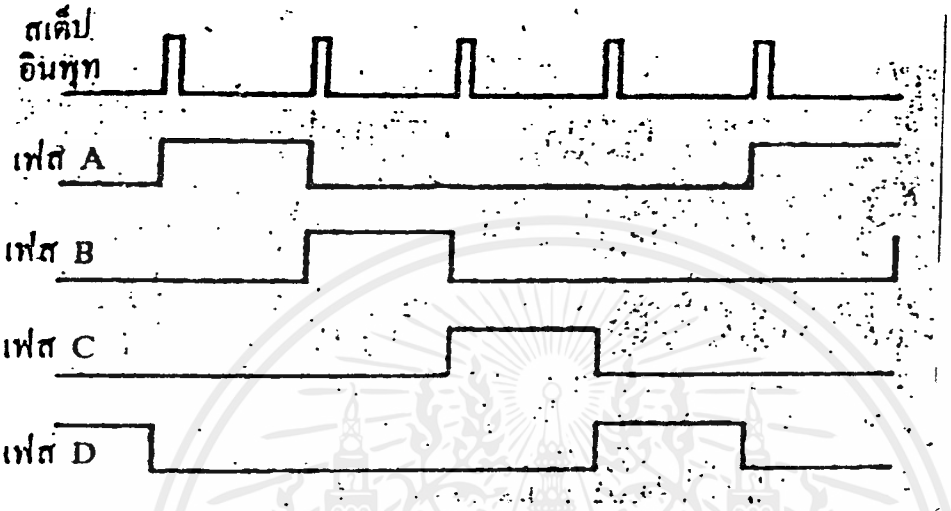
No	$\phi 1$	$\phi 2$	$\phi 3$	$\phi 4$
1	1	0	0	0
2	1	1	0	0
3	0	1	0	0
4	0	1	1	0
5	0	0	1	0
6	0	0	1	1
7	0	0	0	1
8	1	0	0	1

(3) one - two excitation

ในการควบคุมสเตปการทำงานของสเตปปิงมอเตอร์ในทั้ง 3 รูปแบบที่ทราบมานั้นเราสามารถทำได้เป็น 2 ลักษณะคือ ใช้หลักการวงจรชีเควันลอจิกและการไมโครโปรเซสเซอร์ควบคุม ในการใช้วงจรชีเควันลอจิกจะมีตัวอย่างง่าย ๆ โดยการใช้ฟลิบ - ฟลอบ มาทำการออกแบบให้ ได้พัลซ์ตามกำหนดดังวงจรข้างล่าง เป็นวงจรขับสเตปปิงมอเตอร์แบบ 4 เฟส โดยจัดให้มีขับแบบ one - ex--citation ในทิศทางเดียวกันดังแสดงดังรูปที่ 7 วงจรและรูปสัญญาณที่ได้



รูปที่ 7 แสดงวงจรชีเควันลอจิกที่ควบคุมการหมุนของสเตปปิงมอเตอร์แบบ 4 เฟส

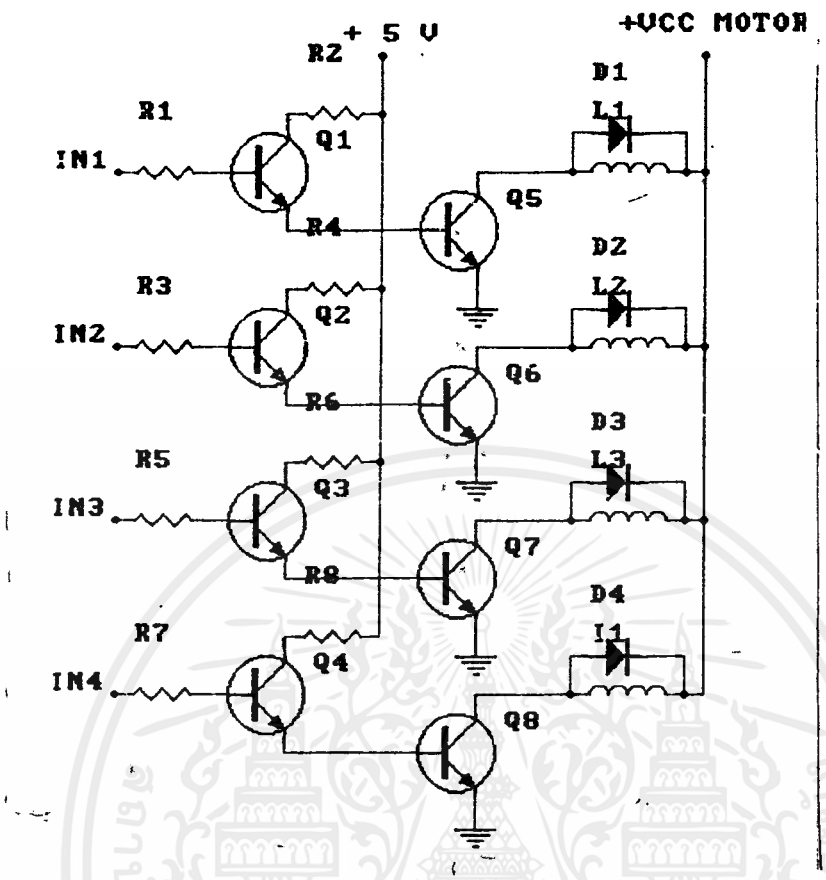


รูปที่ 8 แสดงถึงไดอะแกรมเวลาของพัลซ์ได้จากวงจรซีเควันลอจิก

ถ้าหากต้องการให้หมุนกลับก็สามารถทำได้โดยดัดแปลงวงจรเดิมเล็กน้อยให้พัลซ์ที่ได้เรียงกลับกันเท่านั้น

ลักษณะของการการควบคุมโดยใช้ไมโครโปรเซสเซอร์กำเนิดสัญญาณการกระตุ้นทั้งสามแบบที่กล่าวนั้น ซึ่งสามารถทำได้โดยสะดวกกว่าและเป็นที่นิยมมากกว่า เพราะไมโครโปรเซสเซอร์ได้มีราคาถูกลงมากในปัจจุบันและผู้ใช้ สามารถพลิกแพลงการใช้ได้ตามที่ต้องการ ดังตัวอย่างต่อไปนี้

ก่อนอื่นต้องให้เข้าใจ ส่วนของการขับสัญญาณกระตุ้น ที่ได้จากไมโครโปรเซสเซอร์ก่อน เพราะสัญญาณการกระตุ้นที่ได้จาก 8255 นั้นเราทราบว่าเป็นระดับลอจิก TTL ซึ่งมีค่าโดยประมาณ 4 V เท่านั้น ซึ่งหากต้องการใช้กับ สเตปปิงมอเตอร์ที่มีแรงดันสูง ขึ้นกระแสมากขึ้นต้องมีส่วนขับเพิ่มเข้ามาซึ่งแบบง่ายมักจะใช้เป็นแบบ ดาร์ลิงตัน แสดงดังรูปที่ 9



รูปที่ 9 แสดงถึงวงจรทรานซิสเตอร์ที่ใช้ขับมอเตอร์

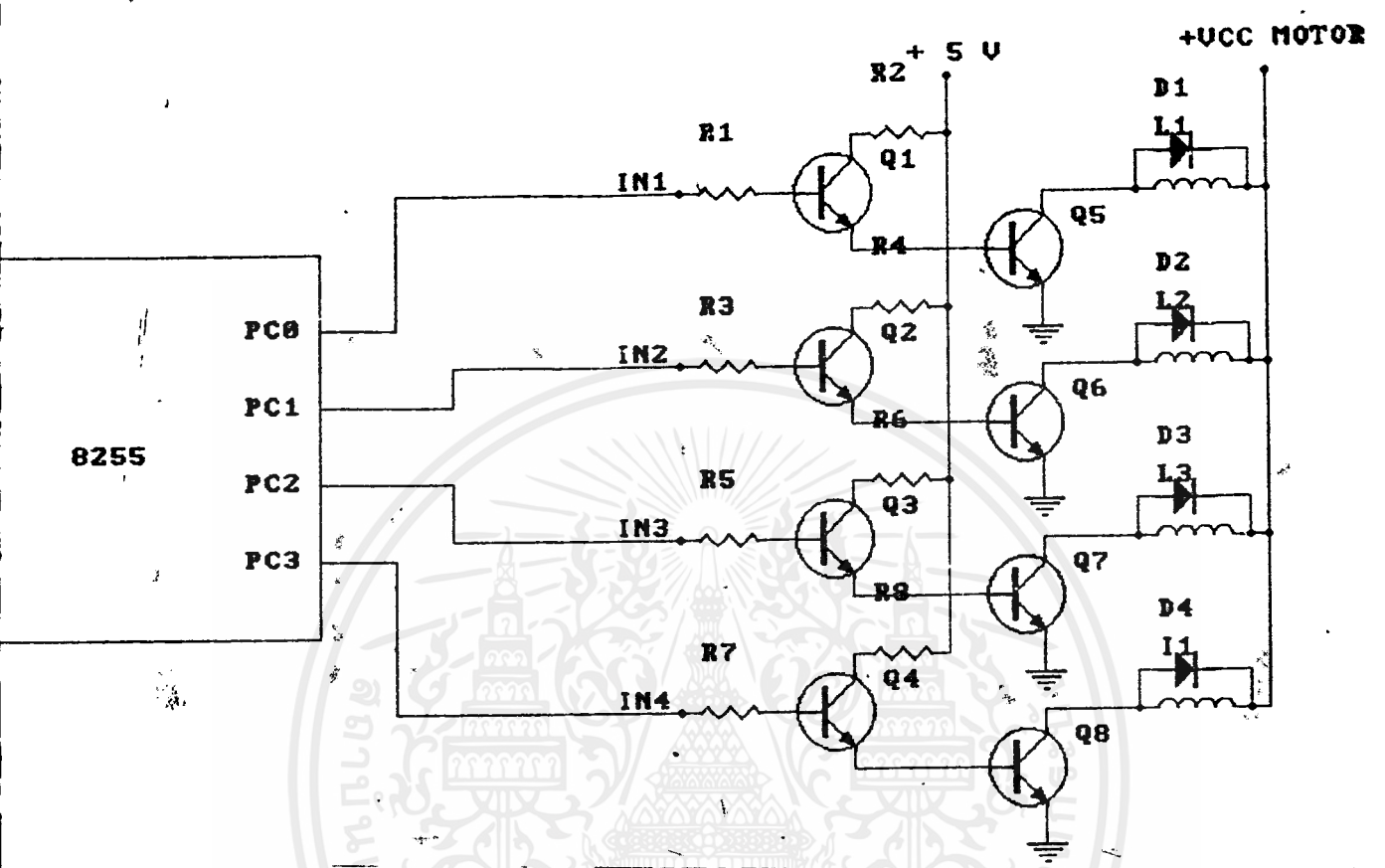
จากรูปที่ 9 R_2 เป็นตัวกำหนดกระแสเบสของ Q_2 ถ้าเราสมมุติให้ β_2 คือค่าอัตราการขยายกระแสต่ำสุดของ Q_2 และ IC_2 คือค่ากระแสขับสูงสุดแต่ละเฟสจะได้

$$R_2 \leq (5\text{-}VBE_2)/IB_2 = (5\text{-}VBE_2)*\beta_2/IC_2$$

สำหรับ R1 จะเป็นตัวกำหนดกระแสเบสของ Q1 ถ้าสมมุติให้ $\beta 1$ คือค่าอัตราการขยายกระแสต่ำสุดของ Q1 และ VIN นั้นมีค่าประมาณ 4 โวล์ท สำหรับลอจิก "1" เราจะได้

$$R_2 \leq (4\text{-}VBE_1\text{-}VBE_2)/IB_1 = 2.8\beta_1/IB_2$$

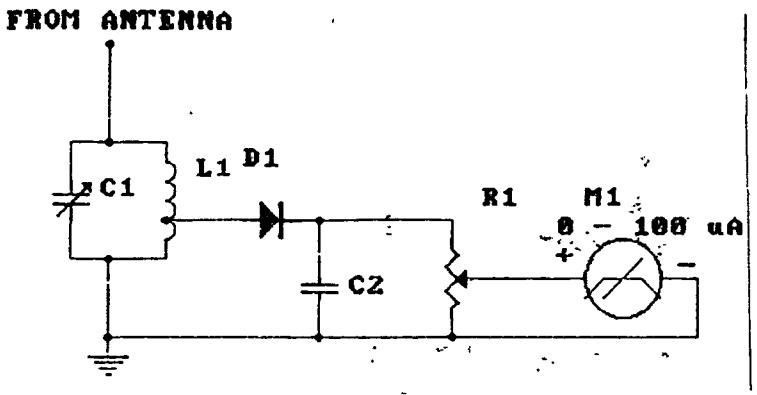
ด้วยสูตรข้างบนเราสามารถหาค่า ค.ต.ท โดยประมาณเมื่อใช้งานได้ เช่นหากมอเตอร์เราต้องการกระแสไฟต่อเฟสขนาด 1.5 A ที่ 15 V_{dc} และเราหาทรานซิสเตอร์ที่กำหนดค่า β ที่เท่ากับ 150,35 ของ Q_1,Q_2 ตามลำดับ (ค่าต่ำสุด) เราจะได้ค่าของ IC_2 = 1.5 A, IB_2 = $IC_2/*2$ = 0.0428 A เราก็ได้ค่าประมาณของ R1 = 12 kR, R2 = 100R เป็นต้น



รูปที่ 10 วงจร 8255 ขับสเตปปิงมอเตอร์แบบ 4 เฟส

2.2 เครื่องมือวัดความแรงของคลื่นวิทยุ

เครื่องวัดความแรงของคลื่นวิทยุ คือ เครื่องรับวิทยุขนาดเล็ก เครื่องหนึ่ง ที่มีมิเตอร์แสดงค่าความแรงของคลื่นวิทยุที่รับได้ รูปที่ 11 เป็นตัวอย่างวงจรเครื่องวัดความแรงคลื่นวิทยุอย่างง่ายแบบหนึ่ง L1 และ C1 ต่อกันเป็นวงจรรีโซแนนซ์แบบขนาน หรือที่เรียกว่าวงจรจูน (Tuned cir cuit) โดยจะยอมให้ความถี่ที่ตรงกับความถี่รีโซแนนซ์ผ่านไปได้มากที่สุด มี C1 ซึ่งเป็นตัวเก็บประจุเก็บค่าได้ตัวเล็ก ๆ หรือเรียกว่า ทริมเมอร์ ทำหน้าที่เป็นตัวปรับแต่งความถี่เรโซแนนซ์ของวงจรให้ตรงกับความถี่ที่เราต้องการใช้งาน

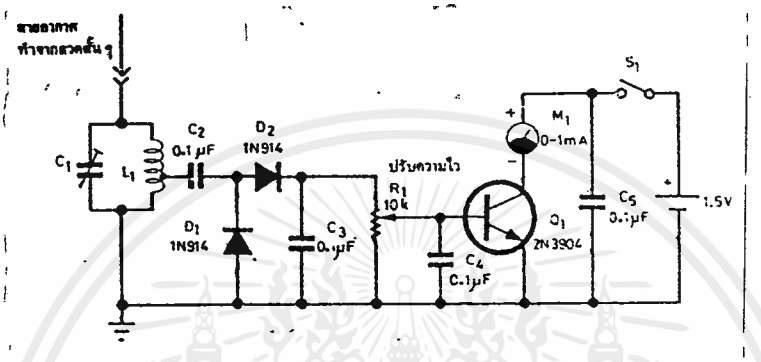


รูปที่ 11 แสดงวงจรเครื่องวัดความแรงของคลื่นวิทยุ

เมื่อคลื่นวิทยุที่มีความถี่ตรงกับความถี่เรโซแนนซ์เข้ามาทางสายอากาศ D1 ซึ่งทำหน้าที่เป็นวงจรดีเทคเตอร์ จะเปลี่ยนแรงดันไฟสลับของคลื่นวิทยุให้ออกมาเป็นแรงดันไฟตรง โดยมี C2 ช่วยทำให้แรงดันไฟตรงที่ออกมาเรียบขึ้น R1 เป็นตัวปรับความไวโดยกำหนดว่าจะให้สัญญาณแรงขนาดใดจึงจะทำให้เข็มของมิเตอร์ M1 ชี้เต็มสเกลได้ สายอากาศนี้อาจใช้เส้นลวดเล็ก ๆ ยาวสักคืบหนึ่งมาเสียบเข้าไปก็ได้ ยิ่งสายอากาศยาวก็จะยิ่งทำให้รับสัญญาณมาได้มากขึ้น

ขอให้สังเกตว่าไดโอด D1 นั้นมักจะถูกต่อเข้ากับ L1 ที่ตำแหน่งประมาณ 10 - 25 เปอร์เซนต์ของจำนวนรอบทั้งหมดเมื่อนับจากกราวนด์ขึ้นไป ที่ทำเช่นนี้เพื่อไม่ให้ D1,C2,R1 และ M1 ไปโหลด (ดึงกระแสจาก) วงจรจูนมากเกินไป ซึ่งจะมีผลทำให้ความคมในการเลือกความถี่เสียไป (แบนด์วิทธ์มากเกินไป) และมีความไวลดลง มิเตอร์ตามตัวอย่างในรูปที่ 11 นี้ใช้ขนาด 0 - 100 μA ซึ่งจริง ๆ แล้วจะใช้ค่าอื่นที่ต่างจากนี้ก็ได้ เช่น ถ้าใช้ขนาด 0 - 50 μA ก็จะทำให้ได้ความไวสูงขึ้น ถ้าใช้ขนาด 0 - 200 μA ก็จะทำให้ได้ความไวลดลง

เราอาจจะเพิ่มความไวของเครื่องวัดความแรงคลื่นวิทยุโดยวิธีอื่นแทนที่จะเพิ่มความไวของมิเตอร์ ได้โดยเพิ่มเติมวงจรขยายสัญญาณไฟตรงเข้าไปดังรูปที่ 12 ในที่นี้ Q1 ทำหน้าที่เป็นวงจรขยายสัญญาณไฟตรง เพื่อให้สามารถจ่ายกระแสให้แก่มิเตอร์ได้มากขึ้น โดยวิธีนี้จะทำให้เราใช้มิเตอร์ M1 เป็นแบบความไวต่ำอย่างไรก็ตามเรายังอาจใช้มิเตอร์ขนาด 0 - 200 μA เพื่อเพิ่มความไวให้สูงขึ้นอีกก็ได้ขอให้พิจารณาด้วยว่าวงจรนี้ใช้วงจรเรคติฟายเออร์เป็นแบบวงจรทวีแรงดัน (Voltage doubler) ซึ่งประกอบด้วย C2,D1,D2 และ C3 ซึ่งยิ่งทำให้ได้สัญญาณไฟตรงมาป้อนทรานซิสเตอร์ได้มากขึ้น เป็นการเพิ่มความไวอีกทาง เป็นต้น



รูปที่ 12 แสดงวงจรเครื่องวัดความแรงคลื่นวิทยุที่เพิ่มความไวขึ้น
โดยใช้วงจรหวีแรงดันและวงจรขยาย

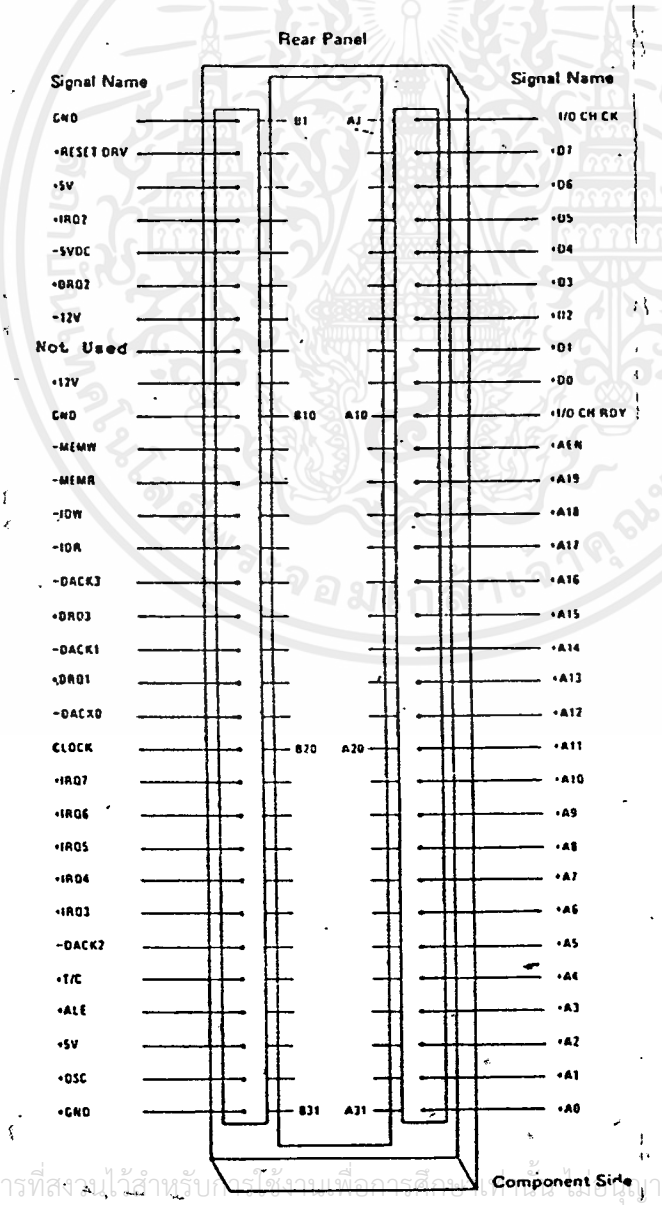
บทที่ 3

สัญญาณต่าง ๆ บนสล็อตของ IBM PC และการจัดแอดเดรส

ภายใน IBM/PC ได้มีการออกแบบให้สามารถที่จะเพิ่มเติมวงจรอินเทอร์เฟสเข้าไปในภายหลังได้ โดยผ่านทางสล็อตที่อยู่บนเมนบอร์ด (Main Board) สำหรับสล็อตบนเมนบอร์ดนี้จะมีจำนวน 5 สล็อต (สำหรับใน IBM PC/XT จะมี 8 สล็อต) ซึ่งแต่ละสล็อตจะมีจำนวนขาทั้งสิ้น 62 ขา แบ่งออกเป็น 2 ข้าง ๆ ละ 31 ขา ส่วนการเรียกตำแหน่งขาของสล็อตเหล่านี้จะขึ้นอยู่กับว่าขานั้นอยู่ข้างใด (ซ้ายหรือขวา) ของสล็อตโดยขาที่อยู่ทางด้านซ้ายของสล็อตจะเรียกโดยใช้อักษร "B" นำหน้าเลขตำแหน่งของขา เช่น ขา B16 ก็คือขาทางด้านซ้ายของสล็อตขาที่ 16 (นับจากทางด้านท้ายของเครื่อง) ส่วนขาที่อยู่ทางด้านขวาของสล็อตจะเรียกโดยใช้อักษร "A" นำหน้าเลขตำแหน่งของขา เช่น ขา A24 ก็คือขาทางด้านขวาของสล็อตขาที่ 24 (นับจาก

ทางด้านซ้ายของเครื่อง)

แต่ละขาของสล็อตเหล่านี้จะเชื่อมต่อกับเส้นสัญญาณต่าง ๆ บนเมนบอร์ดทำให้การสร้างวงจรอินเทอร์เฟสกับ IBM/PC สามารถทำได้โดยสะดวก ซึ่งเส้นสัญญาณที่เชื่อมต่อกับขาของสล็อตเหล่านี้จะประกอบไปด้วย เส้นสัญญาณของบัสแอดเดรส (Address Bus), บัสข้อมูล (Data Bus), บัสควบคุมสำหรับการเขียน/อ่านข้อมูลจากหน่วยความจำ หรือพอร์ท I/O, เส้นสัญญาณสำหรับการขออินเทอร์รัพท์ของวงจรอินเทอร์เฟส, เส้นสัญญาณสำหรับการขอ DMA,สัญญาณฐานเวลา (Timing Signal) ต่าง ๆ ที่ใช้ในระบบ, เส้นสัญญาณแสดงการรีเฟรชหน่วยความจำ และสัญญาณสำหรับการตรวจสอบความผิดพลาด (I/O CHCK) นอกจากเส้นสัญญาณเหล่านี้แล้ว สล็อตบนเมนบอร์ดยังเชื่อมต่อกับแหล่งจ่ายไฟต่าง ๆ ที่ใช้ในระบบอีกด้วย คือ +5Vdc, -5Vdc, +12Vdc และ -12Vdc



รูปที่ 13 แสดงสล็อตของ IBM PC/XT

3.1 รายละเอียดเกี่ยวกับสัญญาณต่าง ๆ

OSC (Oscillator : ขา B30) :

ขานี้เป็นเอาท์พุทที่เชื่อมต่อกับสัญญาณคล็อกที่มีค่าความถี่สูงสุดบนเมนบอร์ดคือ 14.3181 MHz ซึ่งมีคาบเวลาประมาณ 70 nanosec, และมี Duty Cycle (ช่วงเวลาใน 1 คาบที่สัญญาณคล็อกมีลอจิกเป็น "1" หารด้วยคาบเวลาทั้งหมด)ประมาณ 50% สัญญาณคล็อกอื่น ๆ ของระบบ เช่น คล็อกที่ป้อนให้กับ 8088 หรือ ชิพชัพพอร์ตต่าง ๆ นั้นจะถูกสร้างขึ้นโดยการหารสัญญาณคล็อกนี้ อย่างไรก็ตามสิ่งหนึ่งที่จะต้องคำนึงถึงในการใช้งานสัญญาณ OSC ก็คือ สัญญาณนี้จะไม่ Synchronize กับสัญญาณอื่น ๆ บนบัสของระบบ ดังนั้นจึงไม่ควรที่จะนำสัญญาณจากขา OSC นี้ไปใช้เป็นสัญญาณคล็อกสำหรับวงจรภายนอกอื่น ๆ ที่ทำงานร่วมกับระบบ

CLK (Clock : ขา B20)

ขาสัญญาณนี้เป็นเอาท์พุท ซึ่งต่อกับสัญญาณคล็อกที่ถูกสร้างขึ้นโดยการหารสัญญาณ OSC ด้วย 3 ทำให้ได้ความถี่ประมาณ 4.77 MHz (14.31818 MHz/3) หรือมีช่วงเวลาใน 1 คาบ (ช่วงเวลาของคล็อก 1 ลูก) เท่ากับ 210 nanosec. (1/4.77 MHz) สำหรับค่า Duty Cycle ของสัญญาณนี้จะมีค่าประมาณ 1/3 คือใน 1 คาบจะมีช่วงเวลาที่เป็นลอจิก "1" เท่ากับ 1/3 ของคาบเวลาทั้งหมด หรือประมาณ 70 nancsec. และช่วงเวลาที่เป็นลอจิก "0"เท่ากับ 2/3 ของคาบเวลาทั้งหมด หรือประมาณ 140 nanosec. สัญญาณนี้เป็นสัญญาณที่ถูกใช้เป็นคล็อกของระบบ

RESET DRV (ขา B2) :

ขาสัญญาณนี้เป็นเอาท์พุท ซึ่งจะแอคทีฟ (ลอจิก "1")ในช่วงที่เราเริ่มจ่ายไฟให้กับระบบ และจะยังคงแอคทีฟไปจนกว่าระบบต่างๆ ภายใน IBM/PC จะพร้อมที่จะทำงานได้จากนั้นสัญญาณนี้ก็จะเปลี่ยนกลับเป็นลอจิก "0" นอกจากนี้ในระหว่างการทำงานของ IBM/PC ถ้าระดับแรงดันของแหล่งจ่ายไฟตกลง สัญญาณนี้ก็จะถูกทำให้แอคทีฟเช่นกัน โดยทั่วไปแล้วสัญญาณนี้จะถูกนำไปใช้ในการรีเซ็ทวงจรอินเทอร์เฟสหรืออุปกรณ์ I/O ต่าง ๆ ในช่วงที่เริ่มจ่ายไฟให้กับระบบซึ่งจะเป็นการทำให้วงจรหรืออุปกรณ์เหล่านั้นถูกปรับให้อยู่ในสภาวะที่แน่นอน ก่อนที่จะเริ่มต้นการทำงานของระบบ (สภาวะนี้เป็นสภาวะที่เราทราบ และต้องการให้วงจรทำงานในขณะที่ระบบถูกรีเซ็ท)

A0-A19 (Adress Bus : ขา A31-A12)

ขาสัญญาณทั้ง 20 ขานี้เป็นเอาท์พุท ซึ่งใช้สำหรับกำหนดแอดเดรสของหน่วยความจำหรืออุปกรณ์ I/O ที่ 8088 ต้องการติดต่อด้วย โดยที่สัญญาณ A0 จะมีนัยสำคัญต่ำสุด (Least Significant Bit) และ A19 จะมีนัยสำคัญสูงสุด (Most Significant Bit) สำหรับค่าแอดเดรสบนบัสแอดเดรส A0-A19 นี้ จะถูกกำหนดโดย 8088 ในระหว่างขบวนการอ่าน/เขียนข้อมูลลงในหน่วยความจำหรืออุปกรณ์ I/O แต่ในช่วงของขบวนการ DMA นั้น DMA - Controller จะเป็นผู้กำหนดค่าแอดเดรสบนบัสแอดเดรสเอง (ในระหว่างนี้ 8088 จะถูกตัดออกจากระบบ)

จะเห็นได้ว่าจำนวนเส้นแอดเดรสนี้จะมีอยู่ 20 เส้น ซึ่งสามารถที่จะอ้างแอดเดรสของหน่วยความจำได้ถึง 1 Mbyte แต่อย่างไรก็ตามจะมีแอดเดรสบางแอดเดรสที่ถูกใช้งานโดย IBM/PC อยู่ก่อนแล้ว คือแอดเดรสของหน่วยความจำ RAM บนเมนบอร์ดที่ถูกใช้โดยระบบจำนวน 64Kbyte (สำหรับ IBM PC/XT จะเป็นจำนวน 256Kbyte) และแอดเดรสสำหรับหน่วยความจำ ROM อีก 48Kbyte ซึ่งถูกจัดในช่วงของแอดเดรสบนสุดใน 1Mbyte คือ 0FC000H จนถึง 0FFFFFH (สำหรับ IBM PC/XT จะเป็น 64Kbyte) สำหรับการอ้างแอดเดรสของพอร์ท I/O นั้น จะใช้เส้นแอดเดรสเพียง 16 เส้น คือ A0-A15 ซึ่งจะทำให้อ้างแอดเดรสของพอร์ทได้ 64K พอร์ทโดยผ่านทางชุดคำสั่ง IN และ OUT ส่วนเส้นแอดเดรสที่เหลือคือ A16-A19 นั้น จะไม่ถูกใช้งาน อย่างไรก็ตามภายใน IBM/PC จะใช้เส้นแอดเดรสในการอ้างแอดเดรสของพอร์ทเพียง 10 เส้น คือจาก A0-A9 และค่าแอดเดรสที่ใช้งานจะต้องอยู่ในช่วง 0200H จนถึง 03FFH เท่านั้น

D0-D7 (Data Bus : ขา A9-A2)

ขาสัญญาณนี้จะเป็นแบบ Bi-Directional ซึ่งต่อกับบัสข้อมูลของระบบ เพื่อทำหน้าที่ในการส่งผ่านข้อมูลระหว่างพอร์ท I/O กับ IBM/PC โดยบิท D0 จะมีนัยสำคัญต่ำสุดและบิท D7 จะมีนัยสำคัญสูงสุดสำหรับในบัสไซเคิลของการเขียนข้อมูลที่สร้างขึ้นโดย 8088 นั้นข้อมูลจะถูกส่งออกมาบนบัสข้อมูล ก่อนที่สัญญาณ IOW (ในกรณีที่ต้องการส่งข้อมูลให้กับพอร์ท) หรือ MEMW (ในกรณีที่ต้องการส่งข้อมูลให้กับหน่วยความจำ) จะเปลี่ยนจากลอจิก "0" เป็นลอจิก "1" (ขอบขาขึ้น) ซึ่งโดยทั่วไปขอบขาขึ้นของสัญญาณ IOW หรือ MEMW นี้ จะถูกใช้เพื่อสั่งให้พอร์ท I/O หรือหน่วยความจำที่มีแอดเดรสตรงกับค่าแอดเดรสบนบัสแอดเดรสนั้นรับข้อมูลไปเก็บไว้

สำหรับในบัสไซเคิลของการอ่านข้อมูลที่สร้างขึ้นโดย 8088 นั้น พอร์ท I/O หรือหน่วยความจำที่ถูกอ้างถึงจะต้องส่งข้อมูลออกมาบนบัสข้อมูล ก่อนที่สัญญาณ IOR (ในกรณีที่ต้องการอ่าน

ข้อมูลจากหน่วยความจำ) จะเปลี่ยนจากลอจิก "0" เป็นลอจิก "1" (ขอบขาขึ้น)

ALE (Address Latch Enable : ขา B28)

ขาสัญญาณนี้เป็นสัญญาณเอาท์พุทที่ 8288 Bua Controller สร้างขึ้นเพื่อใช้สำหรับแสดงการเริ่มต้นของบัสไซเคิล และแสดงให้อุปกรณ์ภายนอกทราบว่าแอดเดรสที่ 8088 ต้องการจะติดต่อด้วยนั้นถูกส่งออกมาบนบัสแอดเดรสแล้ว ซึ่งที่สัญญาณ ALE นี้จะเปลี่ยนจากลอจิก "1" เป็น "0" เมื่อค่าแอดเดรสที่ถูกต้องถูกส่งออกมาบนบัสข้อมูลเรียบร้อยแล้ว ดังนั้นขอบขาลงสัญญาณ ALE นี้จะถูกใช้ในการแลทช์ค่าแอดเดรสจากบัสแอดเดรส/ข้อมูล (Adress/Data Bus : AD0-AD7) ของ 8088 ทำให้สามารถแยกค่าแอดเดรส (A0-A19) และข้อมูล (A0-A7) ออกจากกันได้ อย่างไรก็ตามสัญญาณ ALE จะแอคทีฟเฉพาะในบัสไซเคิลที่สร้างขึ้นโดย 8088 เท่านั้น โดยจะไม่แอคทีฟในระหว่างขบวนการ DMA

I/O CHCK (I/O Channel Check : ขา A1)

ขาสัญญาณนี้เป็นอินพุทที่ใช้ในการแสดงความผิดพลาด เกี่ยวกับพาริตี้ที่เกิดขึ้นในการทำงานของวงจรอินเทอร์เฟสหรืออุปกรณ์ I/O เมื่อขาสัญญาณนี้ได้รับลอจิก "0" จะทำให้ 8088 ถูกอินเทอร์รัพท์แบบ Non-Maskable (NMI) อย่างไรก็ตามเราสามารถที่จะกำหนดให้วงจรภายในของ IBM/PC ทำการขออินเทอรัพท์ (เมื่อได้รับสัญญาณ I/O CHCK) หรือไม่ก็ได้โดยการกำหนดลอจิกของบิทข้อมูลของพอร์ทที่ควบคุมการขออินเทอร์รัพท์แบบ NMIคือบิท D7 ของพอร์ท 00A0H ถูกเซ็ทเป็น "0" ก็จะเป็นการดิสเอเบิล (Disable) การขออินเทอร์รัพท์แบบ NMI ดังนี้

Enable : ใช้คำสั่ง OUT ส่งข้อมูล 80H ไปยังพอร์ท 00A0H

Disable : ใช้คำสั่ง OUT ส่งข้อมูล 80H ไปยังพอร์ท 00A0H

I/O CHRDY (I/O Channel Ready : ขา A10)

ขาสัญญาณนี้เป็นอินพุทที่ใช้เพิ่มช่วงเวลาในบัสไซเคิลในกรณีที่อุปกรณ์ I/O หรือหน่วยความจำเป็นที่เกี่ยวข้องกับขบวนการในบัสไซเคิลที่เกิดขึ้นนั้น ไม่สามารถทำงานทันตามช่วงเวลาปกติของบัสไซเคิลนั้นๆ ได้ (ช่วงเวลาของบัสไซเคิลที่เกี่ยวกับหน่วยความจำใช้ช่วงเวลาเท่ากับช่วงเวลาของคล็อก 4 ลูก หรือ 840 nanosec. ในขณะที่บัสไซเคิลที่เกี่ยวกับ I/O จะใช้ช่วงเวลาเท่ากับช่วงเวลาของคล็อก 5 ลูกหรือ 1.05 usec.) เมื่ออุปกรณ์ I/O หรือหน่วยความจำต้องการที่จะเพิ่มช่วงเวลาใน บัสไซเคิลให้นานขึ้นอีกนั้น จะสามารถทำได้โดยการป้อนลอ

จิก "0" ให้กับขา I/O CHRDY ในช่วงเวลาที่ I/O หรือหน่วยความจำที่ถูกกำหนดนั้น ได้รับสัญญาณจากการถอดแอดเดรส และสัญญาณ MEMR, MEMW, IOR หรือ IOW แอคทีฟ

IRQ2-IRQ7 (Interrupt Request 2 Through 7 ; ขา B4 และ B25-B21)

ขาสัญญาณทั้ง 6 นี้เป็นขาอินพุทที่ใช้สำหรับการขออินเทอร์รัพท์จาก 8088 โดยสัญญาณเหล่านี้จะต่อเข้ากับ 8259A บนเมนบอร์ดโดยตรง โปรแกรมในส่วน BIOS ของ IBM/PC จะทำการโปรแกรม 8259A ให้ IRQ2 มีลำดับความสำคัญสูงสุด (Highest Priority) และ IRQ7 มีลำดับความสำคัญต่ำสุด ในกรณีที่มีการขออินเทอร์รัพท์เกิดขึ้นคือ ระดับลอจิกที่ขา IRQ ขาใดขาหนึ่ง ถูกเปลี่ยนจากลอจิก "0" เป็นลอจิก "1" (ขอบขาขึ้น) 8259A ก็จะทำการส่งสัญญาณ INT ให้กับ 8088 เพื่อทำการขออินเทอร์รัพท์ซึ่งสำคัญในการขออินเทอร์รัพท์โดยผ่านทาง IRQ2-IRQ7 นี้ ก็คืออุปกรณ์ที่ทำการขออินเทอร์รัพท์โดยผ่านทาง IRQ ขาใดก็จะต้องรักษาระดับสัญญาณที่ขา IRQ นั้น ให้แอคทีฟ (ลอจิก "1") อยู่จนกว่าจะได้รับสัญญาณ INTA (Interrupt Acknowledge) จาก 8088 เสียก่อน ถ้าไม่เช่นนั้นการขออินเทอร์รัพท์จะถูกยกเลิก และอินเทอร์รัพท์ Level 7 (IRQ7) ก็จะถูกสร้างขึ้น โดยอัตโนมัติ ไม่ว่าการขออินเทอร์รัพท์ที่ถูกยกเลิกนั้นจะเป็นการขออินเทอร์รัพท์ ณ Level หรือขาใดแต่อย่างไรก็ตามสัญญาณ INTA นี้จะไม่ถูกต่อออกมาที่ขาของสล็อตด้วย ดังนั้นโปรแกรมที่ทำการตอบสนองต่อการขออินเทอร์รัพท์ (Interrupt Service Routine) จะต้องทำการรีเซ็ทสัญญาณ IRQ เอง โดยใช้คำสั่ง OUT ไปยังพอร์ท I/O ที่เกี่ยวข้อง

IOR (I/O Read ; ขา B14)

ขาสัญญาณนี้เป็นเอาท์พุทแอคทีฟที่ลอจิก "0" ที่สร้างขึ้นโดย 8288 Bus Controller เพื่อใช้ในการแสดงว่าบัสไซเคิลที่เกิดขึ้นนี้ เป็นบัสไซเคิลของการอ่านข้อมูลจากพอร์ท I/O เพื่อให้พอร์ท I/O ที่มีแอดเดรสตรงกับแอดเดรสบนบัสแอดเดรสนั้นส่งข้อมูลออกมาบนบัสข้อมูล โดยข้อมูลจะต้องถูกส่งออกมาบนบัสข้อมูลก่อนขอบขาขึ้นของสัญญาณ IOR ประมาณ 30 nanosec. เพื่อให้มั่นใจได้ว่า 8088 สามารถรับข้อมูลได้ถูกต้อง สำหรับในขบวนการ DMA 8237A-5 DMA Controller จะทำการสร้างสัญญาณ IOR เอง โดยที่ค่าแอดเดรสที่อยู่บนแอดเดรสจะเป็นค่าแอดเดรสของหน่วยความจำ (แทนที่จะเป็นแอดเดรสของพอร์ท I/O) ที่พอร์ท I/O ที่ขอ DMA ต้องการจะนำข้อมูลไปเก็บ การที่พอร์ทใดจะส่งข้อมูลออกมาบนบัสข้อมูลนั้น จะอาศัยสัญญาณ DACK จาก DMA Controller เป็นตัวกำหนด เช่นกรณีที่สัญญาณ DACK1 แอคทีฟก็จะแสดงว่าพอร์ท I/O ที่จะต้องส่งข้อมูลออกมาบนบัสข้อมูลก็คือพอร์ท I/O ที่ขอ DMA ผ่านทางแชนแนลที่ 1(DRQ1)

IOW (I/O Write ; ขา B13)

ขาสัญญาณนี้เป็นเอาท์พุทเอดทีพที่ลอจิก "0" ซึ่งถูกสร้างขึ้นโดย 8288 Bus Conrtol ler เพื่อใช้แสดงว่าบัสไซเคิลที่เกิดขึ้นนี้เป็นบัสไซเคิลของการเขียนข้อมูลลงบนพอร์ท I/O เพื่อให้พอร์ท I/O ที่มีแอดเดรสตรงกับแอดเดรสบนบัสแอดเดรสนั้น รับข้อมูลที่อยู่บนบัสข้อมูลไปเก็บไว้ อย่างไรก็ตามเนื่องจากในช่วงเวลาที่สัญญาณ IOW นี้แอดทีพ (ลอจิก "0") นั้นข้อมูลไปเก็บไว้ อย่างไรก็ตามเนื่องจากในช่วงเวลาที่สัญญาณ IOW นี้แอดทีพ (ลอจิก "0") นั้น ข้อมูลบนบัสข้อมูลอาจจะยังไม่สมบูรณ์ ดังนั้นในการออกแบบจึงควรใช้ขอบขาขึ้นของสัญญาณ IOW แทนขอบขาลงในการหาให้พอร์ท I/O ที่เกี่ยวข้องรับข้อมูลไปเก็บไว้ เพื่อให้ข้อมูลบนบัสข้อมูลสมบูรณ์เสียก่อน สำหรับในขบวนการ DMA นั้น DMA-Controller จะทำการสร้างสัญญาณ IOW เองโดยที่ค่าแอดเดรสที่อยู่บนบัสแอดเดรสจะเป็นค่าแอดเดรสของหน่วยความจำที่พอร์ท I/O ที่ขอ DMA ต้องการจะอ่านข้อมูล

MEMW (Memory Write ; ขา B11)

ขานี้เป็นเอาท์พุทแอดทีพที่ลอจิก "0" ซึ่ง 8288 Bus Controlle สร้างขึ้นในระหว่างบัสไซเคิลในการเขียนข้อมูลลงในหน่วยความจำของ 8088 สัญญาณ MEMW นี้จะถูกส่งออกมาเพื่อให้หน่วยความจำที่แอดเดรสตรงกับค่าแอดเดรสบนบัสแอดเดรสนั้น ทำการรับข้อมูลที่อยู่บนบัสข้อมูลไปเก็บไว้ โดยทั่วไปหน่วยความจำจะรับข้อมูลในช่วงขาขอบขึ้นของสัญญาณ MEMW สำหรับในระหว่างขบวนการ DMA นั้น 8237A-5 DMA-Controller จะทำการควบคุมบัสต่าง ๆ ของระบบแทน 8088 และสัญญาณ MEMW จะถูกใช้บัสไซเคิลของการเขียนข้อมูลลงในหน่วยความจำ (ข้อมูลถูกส่งจากอุปกรณ์ I/O ไปให้กับหน่วยความจำ)

MEMR (Memory Read ; ขา B12)

ขานี้เป็นเอาท์พุทจาก 8288 ซึ่งสัญญาณนี้จะแอดทีพ (ลอจิก "0") ในระหว่างบัสไซเคิลของการอ่านข้อมูลจากหน่วยความจำของ 8088 เพื่อให้หน่วยความจำที่มีแอดเดรสตรงกับค่าแอดเดรสบนบัสแอดเดรสนั้น ทำการส่งข้อมูลออกมาบนบัสข้อมูล โดยหน่วยความจำนั้นจะต้องส่งข้อมูลออกมาในช่วงเวลา 30 nanosec. ก่อนที่สัญญาณ MEMW จะกลับเป็นลอจิก "1"

สำหรับในระหว่างขบวนการ DMA นั้น DMA-Controller จะควบคุมบัสต่าง ๆ ของระบบแทน 8088 และสัญญาณ MEMR จะถูกใช้ในบัสไซเคิลของการอ่านข้อมูลจากหน่วยความจำ (ข้อมูลถูกส่งจากหน่วยความจำไปให้กับอุปกรณ์ I/O)

DRQ1-DRQ3 (DMA Request 1-3; ขา B18, B6 และ ขา B16)

ขาสัญญาณทั้งสามนี้เป็นสัญญาณอินพุทแอคทีฟที่ลอจิก "1" ซึ่งอุปกรณ์ภายนอกสามารถใช้ในการขอ DMA จากระบบโดยการป้อนระดับสัญญาณลอจิก "1" ให้กับขา DRQ ขาใดขาหนึ่ง (ขา DRQ ทั้งสามนี้จะต่อเข้ากับ DRQ1-DRQ3ของ 8237A-5) เมื่อ 8237A-5 ได้รับสัญญาณนี้แล้วก็จะตรวจสอบว่ามีการขอ DMA ในแชนแนลที่มีลำดับความสำคัญ (Priority) สูงกว่าหรือไม่ ถ้าไม่มีก็จะทำการขอ DMA จาก 8088 และตอบรับการขอ DMA จากอุปกรณ์ภายนอก (สัญญาณ DACK ของแชนแนลที่ขอ DMA จะแอคทีฟ)แต่ถ้ามี 8237A-5 ก็จะทำการขอ DMA ให้กับแชนแนลที่มีลำดับความสำคัญสูงกว่าก่อนแล้วจึงทำการขอ DMA ให้กับแชนแนลที่มีลำดับความสำคัญต่ำกว่า ภายใน ROM BIOS ของ IBM/PC จะโปรแกรม 8237A-5 ให้ DRQ1 มีลำดับความสำคัญสูงสุดและ DRQ3 มีลำดับความสำคัญต่ำสุด ดังนั้นถ้ามีการขอ DMA ของอุปกรณ์ภายนอกผ่านทางแชนแนลที่ (DRQ1) และแชนแนลที่ 2 (DRQ2) 8237A-5 ก็จะทำการขอ DMA ให้กับแชนแนลที่ 1 ก่อน จากนั้นเมื่อเสร็จจากขบวนการ DMA ของแชนแนลที่ 1 แล้ว จึงจะทำการขอ DMA ให้กับแชนแนลที่ 2 อย่างไรก็ตาม 8237A-5 ยังมีแชนแนลสำหรับการขอ DMA อยู่อีก 1 แชลแนล คือ แชนแนลที่ 0 (DRQ0) ซึ่งในความเป็นจริงแล้วแชนแนลนี้จะมีลำดับความสำคัญที่สูงกว่าแชนแนลที่ 1 แต่จะไม่ถูกต่อออกมายังขาของสล็อต เนื่องจาก IBM/PC จะใช้แชนแนลที่ 0 นี้ในการรีเฟรชหน่วยความจำที่เป็น Dynamic RAM ในการขอ DMA นั้นสัญญาณ DRQ นี้ จะต้องแอคทีฟอยู่ในช่วงระยะเวลาหนึ่งเท่านั้น ถ้าสัญญาณนี้แอคทีฟอยู่นานเกินไป จะทำให้เกิดขบวนการ DMA ขึ้นมากกว่า 1 ขบวนการได้ สำหรับวงจรที่ขอ DMA โดยทั่วไปแล้วจะใช้สัญญาณตอบรับการขอ DMA หรือสัญญาณ DACK ของแชนแนลที่ขอ DMA นั้น ในการรีเซ็ทสัญญาณ DRQ เช่นอุปกรณ์ภายนอกที่ขอ DMA ผ่านทางแชนแนลที่ 1 (DRQ1) ก็จะคอยตรวจสอบการตอบรับการขอ DMA จากสัญญาณDACK ของแชนแนลที่ (DACK1) เมื่อได้รับสัญญาณจาก DACK1 แล้ว ก็จะรีเซ็ทสัญญาณ DRQ1 (เปลี่ยนจากลอจิก "1" เป็น "0")

DACK0-DACK3 (DMA Acknowledge 0-3; ขา B19, B17, B26 และ B15):

สัญญาณทั้ง 4 นี้เป็นเอาท์พุทแอคทีฟที่ลอจิก "0" ซึ่ง 8237A-5 สร้างขึ้นเพื่อแสดงให้วงจรภายนอกที่ขอ DMA ทราบว่าการขอ DMA นั้นได้รับการตอบสนองแล้ว และ 8237A-5 จะเข้าสู่ขบวนการ DMA เพื่อให้การส่งผ่าน ข้อมูลระหว่างอุปกรณ์ I/O ที่ขอ DMA กับหน่วยความจำเกิดขึ้นได้โดยตรง (คือ ไม่ต้องผ่าน 8088) โดยสัญญาณ DACK นี้จะแอคทีฟในแชนแนลใดก็ขึ้นอยู่กับว่าขบวนการ DMA ที่จะเกิดขึ้นนั้น เป็นการตอบสนองต่อการขอ DMA ในแชนแนลใด เช่นถ้าขบวนการ DMA ที่จะเกิดขึ้นนั้นเป็นการตอบสนองต่อการขอ DMA ในแชนแนลที่ 2

(DRQ2) สัญญาณ DACK2 ก็จะแอคทีฟ เป็นต้น

ดังที่ได้กล่าวแล้วว่าสัญญาณ DRQ0 นั้น จะไม่ถูกต่อออกมายังขาของสล็อต ดังนั้นวงจรอินเทอร์เฟสจึงไม่สามารถจะขอ DMA ผ่านทางแชนแนล 0 ได้ แต่สัญญาณ DACK0 จะถูกต่อออกมายังสล็อตด้วย (ขา B19) ทั้งนี้ก็เพื่อที่จะแสดงให้วงจรอินเทอร์เฟสต่าง ๆ ทราบว่าขบวนการ DMA ที่เกิดขึ้นในช่วงเวลาที่ DACK0 แอคทีฟนั้น เป็นขบวนการที่ใช้สำหรับการรีเฟรชหน่วยความจำเป็น Dynamic RAM ซึ่งวงจรอินเทอร์เฟสที่ใช้หน่วยความจำประเภทนี้ นี้สามารถจะนำไปใช้ในการรีเฟรช Dynamic RAM ที่อยู่ในวงจรได้ ซึ่งการรีเฟรชหน่วยความจำนั้นจะเกิดขึ้นในทุก ๆ 15ม 12 usec. หรือทุก ๆ 72 คล็อก ดังนั้นสัญญาณ DACK0 นี้ก็จะแอคทีฟในทุก ๆ 15ม 12 usec.ด้วย

AEN (Address Enable ; ขา A11)

สัญญาณนี้เป็นเอาท์พุทที่ใช้ในการ แสดงว่าบัสไซเคิลที่เกิดขึ้นในช่วงเวลาที่สัญญาณ AEN แอคทีฟ (ลอจิก "1") นั้น เป็นบัสไซเคิลของขบวนการ DMA สำหรับบนเมนบอร์ดของ IBM/PC นั้น จะใช้สัญญาณนี้ในการดิสเอเบิล (Disable) 8288 Bus Controller และจะใช้ดิสเอเบิลพอร์ท I/O ต่าง ๆ ที่ไม่เกี่ยวข้องกับขบวนการ DMA ที่เกิดขึ้นนี้ ที่จำเป็นต้องทำเช่นนี้ก็เพราะในระหว่างขบวนการ DMA นั้น 8237A-5 จะส่งแอดเดรสของหน่วยความจำออกมาบนบัสแอดเดรส และจะทำให้สัญญาณ IOR หรือ IOW แอคทีฟด้วย ดังนั้นถ้าไม่ทำการดิสเอาเบิลพอร์ท I/O ที่ไม่เกี่ยวข้องไว้ ก็อาจจะทำให้พอร์ท I/O ที่มีแอดเดรสตรงกับค่าแอดเดรสบนบัสแอดเดรส (ซึ่งเป็นแอดเดรสของหน่วยความจำ) นั้น ทำการอ่านหรือส่งข้อมูลออกมาบนบัสข้อมูลทำให้เกิดความผิดพลาดขึ้นได้

T/C (Terminal Count ; ขา B27)

สัญญาณนี้ถูกสร้างขึ้นจากการนำเอาสัญญาณเอาท์พุทที่ขา EOP ของ 8237A-5 มากลับลอจิก (ซึ่งใช้เกท Inverter) ทำให้สัญญาณ T/C นี้แอคทีฟที่ลอจิก "1" สำหรับสัญญาณนี้จะแอคทีฟ เมื่อจำนวนไบท์ในการส่งผ่านข้อมูลของขบวนการ DMA ในแชนแนลใดแชนแนลหนึ่งครบตามจำนวนที่กำหนดไว้ ซึ่งโดยทั่วไปแล้วสัญญาณที่จะถูกใช้ในการสิ้นสุดขบวนการ DMA ที่ทำการส่งผ่านข้อมูลเป็นบล็อค เนื่องจากสัญญาณนี้จะแอคทีฟโดยไม่แสดงว่าเป็นสัญญาณของแชนแนลใด ดังนั้นจึงต้องทำการนำสัญญาณ T/C นี้ผ่านเกท Inverter แล้วนำไป OR กับสัญญาณ DACK เพื่อให้สามารถทราบได้ว่า สัญญาณ T/C ที่เกิดขึ้นนั้นเป็นสัญญาณของแชนแนลใด สำหรับในแชน

แนลที่ 0 นั้นสัญญาณ T/C จะแอคทีฟในช่วงเวลาที่คงที่คือ ทุกๆ 990.804 millisec. ซึ่งก็คือช่วงเวลาที่ใช้ในการรีเฟรชหน่วยความจำขนาด 64 Kbyte นั่นเอง

3.2 การจัดแอดเดรสสำหรับพอร์ท I/O

ในเรื่องนี้จะกล่าวถึงการจัดแอดเดรสสำหรับหน่วยความจำ และพอร์ท I/O ต่าง ๆ ภายใน IBM/PC ซึ่งจะแสดงถึงแอดเดรสต่าง ๆ ที่ถูกใช้งานโดยพอร์ท I/O (Input/ Output Port) และหน่วยความจำ นอกจากนี้จะได้กล่าวถึงเทคนิคการดีโค้ด (Decode) แอดเดรสในแบบต่าง ๆ ด้วย

- การจัดแอดเดรสสำหรับพอร์ท I/O ใน IBM/PC

ในหัวข้อนี้จะกล่าวถึงวิธีการอ้าง และใช้งานแอดเดรสต่าง ๆ ของพอร์ท I/O ที่ใช้งานอยู่ใน IBM/PC .

- การอ้างแอดเดรสของพอร์ท I/O

ในการควบคุมและตรวจสอบสภาวะการทำงาน รวมทั้งการอ่านข้อมูลจากอุปกรณ์ที่เป็นชิพชั้นพอร์ทหรือการ์ดต่าง ๆ ที่ใช้ในระบบของ IBM/PC นั้นจะกระทำโดยผ่านทางพอร์ท I/O ของระบบ ดังนั้นในการที่จะใช้งานหรือควบคุมการทำงานของอุปกรณ์เหล่านี้ จึงจำเป็นต้องศึกษาถึงวิธีการควบคุมพอร์ท I/O ต่าง ๆ ของระบบด้วย และเนื่องจากการควบคุมหรือติดต่อกับพอร์ทเหล่านี้ต้องกระทำโดยการอ้างถึงแอดเดรสของพอร์ท I/O เหล่านั้นโดยตรง เราจึงจำเป็นต้องศึกษาถึงหลักการอ้างแอดเดรสของ 8088 ใน IBM/PC ด้วยสำหรับแอดเดรสของพอร์ท I/O ต่าง ๆ นั้น จะเป็นแอดเดรสที่ถูกสร้างขึ้นโดย 8088 ซึ่งแอดเดรสเหล่านี้เป็นแอดเดรสที่จัดไว้สำหรับพอร์ท I/O โดยเฉพาะ คือแยกจากแอดเดรสของหน่วยความจำโดยเด็ดขาด ส่วนการส่งข้อมูลให้กับพอร์ทเหล่านี้จะทำได้โดยการใช้คำสั่ง OUT ของ 8088 ส่งข้อมูลนั้นไปยังแอดเดรสของพอร์ทที่ต้องการ และ สำหรับการตรวจสอบหรือการอ่านข้อมูลจากพอร์ท ก็จะทำได้โดยการใช้คำสั่ง IN ของ 8088 อ่านข้อมูลจากแอดเดรสของพอร์ทที่ต้องการเช่นกัน

ภายในไมโครโปรเซสเซอร์เบอร์ 8088 นี้จะมีแอดเดรสสำหรับใช้กับพอร์ท I/O อยู่ทั้งสิ้น 65,536 หรือ 64K แอดเดรส (ในขณะที่มีแอดเดรสสำหรับหน่วยความจำอยู่ 1Mbyte) ซึ่งทำให้การอ้างแอดเดรสของพอร์ท I/O ที่ทำงานร่วมกับ 8088 นั้นต้องใช้จำนวนเส้นแอดเดรสในบัสแอดเดรสทั้งสิ้น 16 เส้น คือ A0-A15 แต่สำหรับใน IBM/PC นี้ถูกออกแบบ

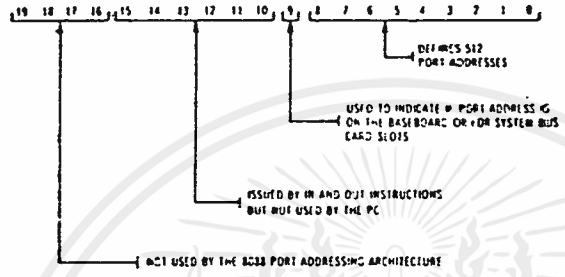
มาให้ใช้เส้นแอดเดรสเฉพาะ 10 เส้นล่าง คือ A0--A9 เท่านั้น ดังนั้นในการอ้างถึงแอดเดรสของพอร์ทของอุปกรณ์หรือชิพชัพพอร์ทต่าง ๆ ที่ใช้ร่วมกับ IBM/PC จึงใช้จำนวนเส้นแอดเดรสเพียง 10 เส้นด้วย โดยเส้นแอดเดรสที่เหลือคือ A 10-A15 นั้นจะไม่ถูกนำไปใช้งาน อย่างไรก็ตามถึงแม้ว่าเส้นแอดเดรส A10-A15 นี้จะไม่ถูกนำไปใช้งาน แต่ค่าแอดเดรสบนเส้นแอดเดรสเหล่านี้นั้งคงเปลี่ยนแปลงตามค่าแอดเดรสของพอร์ทมากำหนดไว้ในคำสั่ง OUT หรือ IN อยู่ด้วย เพียงแต่ไม่ได้ถูกนำมาถอดรหัสร่วมกับแอดเดรส A0-A9 เท่านั้น ตัวอย่างเช่นในการใช้คำสั่ง OUT ส่งข้อมูลไปยังพอร์ทที่ตรงกับแอดเดรส 0010H นั้นจะให้ผลเหมือนกับ การส่งข้อมูลไปยังพอร์ทที่ตรงกับแอดเดรส 0410H, 0810, 0C10H, ทั้งนี้เนื่องจากแอดเดรส 6 บิทบนไม่ได้ถูกใช้งานจึงทำให้การเปลี่ยนแปลงค่าแอดเดรสบนเส้นแอดเดรส A10-A15 นั้น ไม่ทำให้เกิดความแตกต่างใด ๆ ขึ้น

เนื่องจากใน IBM/PC ได้ใช้งานเส้นแอดเดรสเพียง 10 เส้น(คือA0-A9) ดังนั้นจึงสามารถที่จะอ้างแอดเดรสของพอร์ทได้สูงสุดเพียง 1024 พอร์ท (จากจำนวน 64 K พอร์ท) เท่านั้น นอกจากนี้ในกรณีที่เป็นการอ่านข้อมูลจากพอร์ทของ IBM/PC ข้อมูลในบิท A9 จะถูกจัดให้มีหน้าที่ในการแบ่งพอร์ททั้ง 1024 พอร์ทออกเป็น 2 ส่วน (ส่วนละ 512 พอร์ท) อีกด้วย กล่าวคือถ้าข้อมูลในบิท A9 เป็น "0" แล้ว เราจะทำการอ่านข้อมูลได้เฉพาะจากพอร์ทของอุปกรณ์หรือชิพชัพพอร์ทต่าง ๆ ที่อยู่บนเมนบอร์ด (Main Board) ของ IBM/PC เช่น 8253-5, 8237-5 หรือ 8259A เท่านั้น แต่ถ้าข้อมูลในบิท A9 นี้เป็น "1" ก็จะทำการอ่านข้อมูลได้เฉพาะจากพอร์ทที่อยู่บนการ์ดต่าง ๆ เท่านั้น

จากที่ได้กล่าวมานั้นจะสรุปได้ว่าพอร์ทบน IBM/PC ทั้ง 1024 พอร์ทถูกแบ่งออกเป็น 2 กลุ่ม โดยที่กลุ่มแรกเป็นกลุ่มของพอร์ทที่อยู่บนเมนบอร์ด และกลุ่มที่สองเป็นกลุ่มที่จัดเตรียมไว้สำหรับที่อยู่บนการ์ดต่าง ๆ

สำหรับในกรณีของการส่งข้อมูลให้กับพอร์ททั้ง 1024 พอร์ท เราสามารถที่จะเลือกส่งไปยังพอร์ทใด ๆ ใน IBM/PC ได้ ดังนั้นการเลือกแอดเดรสสำหรับพอร์ทที่อยู่บนการ์ดจึงสามารถทำได้โดยสะดวก แต่อย่างไรก็ตามสิ่งหนึ่งที่จะต้องคำนึงถึงก็คือ ถ้าแอดเดรสที่เราเลือกให้กับพอร์ทนี้ตรงกับค่าแอดเดรสเดิมที่มีอยู่บนเมนบอร์ดแล้ว เมื่อเราทำการส่งข้อมูลให้กับพอร์ทที่อยู่ในตำแหน่งแอดเดรสนี้ ก็จะเท่ากับเป็นการส่งข้อมูลให้กับทั้งพอร์ทที่อยู่บนเมนบอร์ดและพอร์ทที่อยู่บนการ์ดด้วย ซึ่งในกรณีเช่นนี้อาจจะก่อให้เกิดความผิดพลาดขึ้นได้เช่นกัน ดังนั้น ในการกำหนดค่าแอดเดรสให้กับพอร์ทที่ถูกสร้างขึ้นบนการ์ดต่าง ๆ จึงควรจะใช้ค่าแอดเดรสที่แอดเดรสบิท A9 มีค่าเป็น "1" คือ แอดเดรส 0FE00H จนถึง 0FFFFH เท่านั้น (แอดเดรสบิท A10-A15 ไม่ถูกใช้ในการถอดรหัส แต่เพื่อความสะดวกจึงกำหนดให้มีค่าเป็น "1" ในฐานสองทั้งหมด แต่ในการใช้

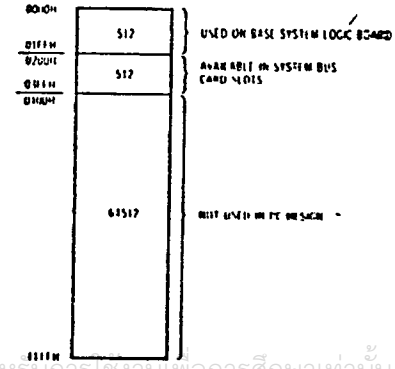
งานจริงเปลี่ยนให้แอดเดรส A10–A15 แต่ละบิทมีค่าเป็น "1" หรือ "0" ก็ได้)



รูปที่ 14 การใช้แอดเดรสบิทต่าง ๆ ในการอ้างแอดเดรสของพอร์ทใน IBM/PC

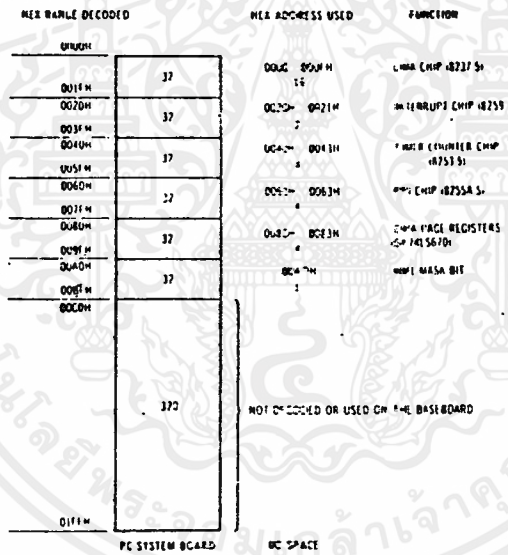
3.3 การใช้งานแอดเดรสสำหรับพอร์ท I/O ใน IBM/PC

จากที่ได้กล่าวไว้ในหัวข้อที่ผ่านมานั้น พอร์ท I/O ทั้ง 1024 พอร์ทใน IBM/PC จะถูกแบ่งออกเป็น 2 กลุ่ม ๆ ละ 512 พอร์ท สำหรับในหัวข้อนี้จะกล่าวถึงการใช้งานพอร์ทต่าง ๆ เหล่านี้ โดยจะแบ่งออกเป็น 2 กลุ่ม ตามที่ได้อธิบายไว้ในหัวข้อที่ผ่านมาดังนี้



รูปที่ 15 การใช้งานแอดเดรสของพอร์ทบน IBM/PC

1. ในกลุ่มแรกนี้ เป็นกลุ่มของพอร์ท I/O ที่อยู่บนเมนบอร์ดของ IBM/PC ซึ่งจะมีแอดเดรสอยู่ในตำแหน่ง 0000H จนถึง 01FFH (ขอให้ระลึกอยู่เสมอว่า A10-A15 นั้นไม่ถูกใช้งาน) หรือแอดเดรสที่มีบิท A9 เป็น "0" นั่นเอง สำหรับแอดเดรสของพอร์ท I/O ในกลุ่มนี้จะถูกใช้ในการอ้างแอดเดรสของชิพชัพพอร์ท และ อุปกรณ์ที่เป็น I/O ต่าง ๆ บนเมนบอร์ดของ IBM /PC เช่น แอดเดรส 0000H จนถึง 000FH จะถูกใช้เป็นแอดเดรสสำหรับ 8237-5 DMA Controller เป็นต้น ในรูปที่ 15 จะแสดงถึงการใช้งานแอดเดรสต่าง ๆ ตั้งแต่ 0000H จนถึง 01FFH ในการอ้างแอดเดรสของชิพชัพพอร์ทและอุปกรณ์ต่าง ๆ ที่ทำหน้าที่เป็น I/O บนเมนบอร์ดของ IBM/PC



รูปที่ 16 การใช้งานแอดเดรสต่าง ๆ สำหรับพอร์ท I/O ของ IBM/PC

จากรูปข้างต้นจะเห็นว่าแอดเดรส 00C0H จนถึงแอดเดรส 01FFH นั้นไม่ได้ถูกใช้งานบนเมนบอร์ดของ IBM/PC ดังนั้นในกรณีนี้เราก็สามารถที่จะใช้งานแอดเดรสต่าง ๆ เหล่านี้ได้ อย่างไรก็ตามแอดเดรสเหล่านี้ยังคงถูกดีโค้ดให้เป็นแอดเดรสที่ใช้ในการอ่านข้อมูลจากพอร์ท I/O บนเมนบอร์ดเท่านั้น ดังนั้นการใช้ค่าแอดเดรส 00C0H-01FFH กับพอร์ท I/O บนการ์ดหรือวงจรอินเทอร์เฟสที่เราสร้างขึ้นนั้น ต้องเป็นพอร์ทที่เอาท์พุทเพียงชนิดเดียวเท่านั้น กล่าวคือจะทำการอ่านข้อมูลจากพอร์ท I/O (ที่ไม่ได้อยู่บนเมนบอร์ด) ที่มีค่าแอดเดรสอยู่ในช่วง 00C0H-01FFH

ไม่ได้

2. ในกลุ่มที่สองนี้ จะเป็นกลุ่มของพอร์ท I/O ที่ถูกใช้งานอยู่บนการ์ดที่ใช้เสียบบนสล็อตต่าง ๆ ของ IBM/PC สำหรับแอดเดรสของพอร์ทเหล่านี้จะเริ่มต้นจากแอดเดรส 0200H จนถึง 03FFH ซึ่งก็คือแอดเดรสที่มีบิท A9 เป็น "1" นั่นเองสำหรับการใช้งานแอดเดรสของพอร์ท I/O ในกลุ่มนี้จะแสดงได้ดังรูปที่ 16

อย่างไรก็ตามการใช้งานแอดเดรสในกลุ่มนี้ อาจจะเปลี่ยนแปลงไปได้ ทั้งนี้ขึ้นอยู่กับการใช้งานการ์ดต่าง ๆ ร่วมกับ IBM/PC โดยการ์ดที่ถูกออกแบบผลิตขึ้นใหม่นั้นอาจจะใช้ค่าแอดเดรสต่าง ๆ ที่เหลืออยู่นี้ได้ ดังนั้นก่อนที่จะทำการออกแบบวงจรอินเทอร์เฟสที่จำเป็นต้องใช้ค่าแอดเดรสสำหรับพอร์ท I/O จึงควรจะตรวจสอบดูก่อนว่าการ์ดต่าง ๆ ที่ใช้อยู่ในระบบของ IBM/PC ที่เราใช้งานอยู่นั้นมีการ์ดใดบ้าง และการ์ดเหล่านั้นใช้งานแอดเดรสใดบ้าง จากนั้นจึงทำการออกแบบวงจรอินเทอร์เฟสโดยเลือกใช้เฉพาะแอดเดรสที่ยังไม่ถูกใช้งาน

3.4 เทคนิคในการถอดรหัสแอดเดรสสำหรับพอร์ท I/O

ในหัวข้อต่าง ๆ ที่ผ่านมาข้างต้นนั้น ได้กล่าวถึงการอ้างแอดเดรสและการใช้งานแอดเดรสต่าง ๆ ของพอร์ท I/O ใน IBM/PC สำหรับในหัวข้อนี้จะกล่าวถึงวิธีการต่าง ๆ ที่ใช้ในการถอดรหัสแอดเดรสต่าง ๆ ให้เป็นไปตามที่เราต้องการ

- การถอดรหัสแบบ Fixed

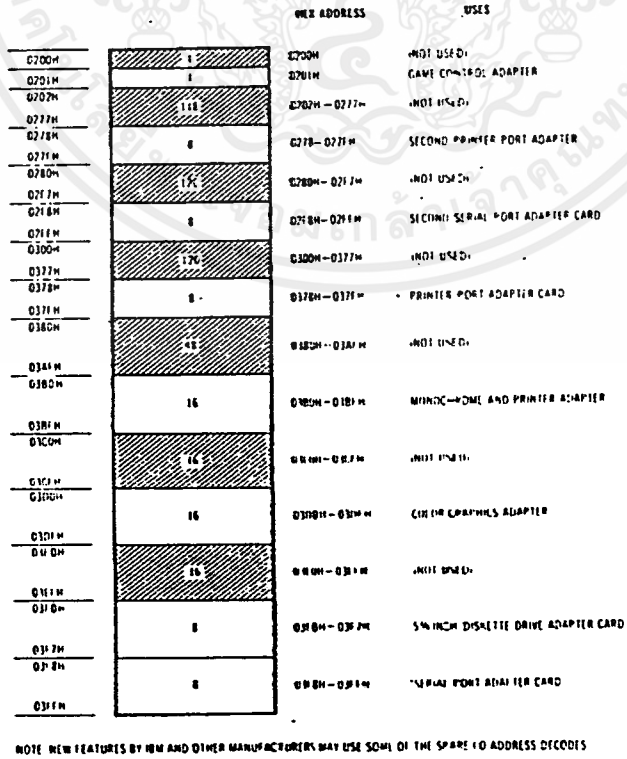
วิธีการถอดรหัสแบบนี้เป็นวิธีหนึ่งที่ง่าย และสะดวกในการถอดรหัสแอดเดรสหรือกลุ่มของแอดเดรสของพอร์ท I/O ซึ่งวิธีนี้เป็นการกำหนดจำนวนของแอดเดรสที่เราต้องการใช้ จากนั้นจึงทำการเลือกบล็อกของแอดเดรสที่ยังไม่ถูกใช้งานโดยการ์ดหรือวงจรอินเทอร์เฟสอื่น ๆ (บล็อกของแอดเดรสที่เลือกต้องมีจำนวนแอดเดรสเดียงพอกับจำนวนแอดเดรสที่เราต้องการใช้งาน) แล้วจึงออกแบบวงจรที่ทำการถอดรหัสแอดเดรสที่เราต้องการ สำหรับตัวอย่างวงจรที่ใช้ในการถอดรหัสแอดเดรสในแบบนี้จะแสดงได้ดังรูปที่ 17

จากรูปจะเห็นได้ว่า วงจรที่ใช้นี้เป็นวงจรที่สามารถทำการถอดรหัสแอดเดรสได้ 8 กลุ่ม โดยแต่ละกลุ่มจะมีจำนวนแอดเดรส 4 แอดเดรสซึ่งแอดเดรสทั้ง 8 กลุ่มจะแสดงได้ดังตารางข้างล่าง

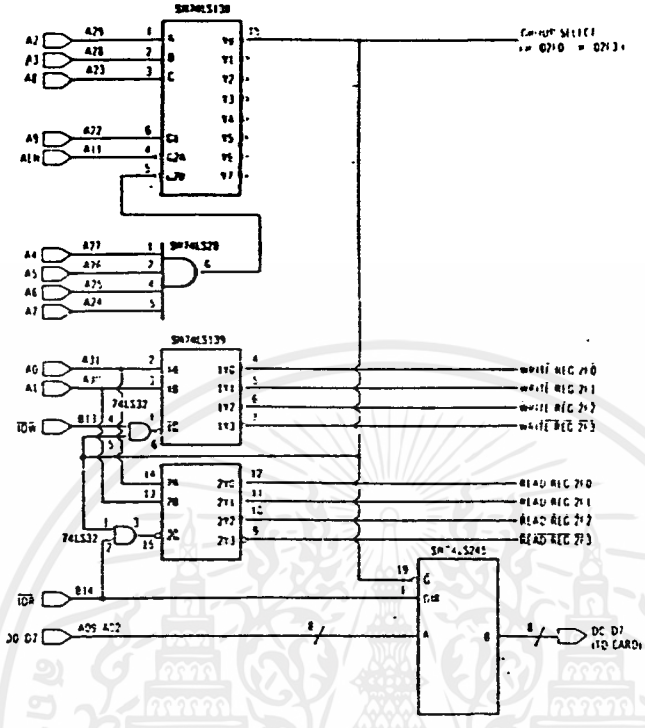
สำหรับในตัวอย่างนี้จะเลือกใช้การถอดรหัสแอดเดรสในกลุ่ม 0 (เริ่มจากแอดเดรส 02F0H จนถึง 02F3H) คือใช้สัญญาณเอาท์พุท (สัญญาณ GROUPSELECT) จากขา Y0 (ขา15)

กลุ่ม	แอดเดรส
0 (Y0)	02F0H-02F3H
1 (Y1)	02F4H-02F7H
2 (Y2)	02F8H-02FBH
3 (Y3)	02FCH-02FFH
4 (Y4)	03F0H-03F3H
5 (Y5)	03F4H-03F7H
6 (Y6)	03F8H-03FBH
7 (Y7)	03FCH-03FFH

ของ 74LS138 ไปทำการ OR กับสัญญาณ IOR และ IOW เพื่อสร้างเป็นสัญญาณอีนาเบิลวงจรดีโค้ด (74LS139)แอดเดรสอีก 4 แอดเดรส ซึ่งแบ่งเป็น 2 ชุดคือ ชุดที่เป็น WRITE REG ซึ่งจะแอคทีฟ (ลอจิก "0") เมื่อ CPU ต้องการจะส่งข้อมูลให้กับวงจรภายนอก (สัญญาณ IOW แอคทีฟ) และชุดที่เป็น READ REG ซึ่งจะแอคทีฟเมื่อ CPU ต้องการจะอ่านข้อมูลจากวงจรภายนอก (สัญญาณ IOR แอคทีฟ) สัญญาณ WRITE REG และ READ REG นี้โดยทั่วไปจะนำไปเป็นสัญ



รูปที่ 17 การใช้งานแอดเดรสสำหรับพอร์ท I/O บนการ์ดต่าง ๆ



รูปที่ 18 ตัวอย่างวงจรดีโค้ดแอดเดรสแบบ Fixed

สัญญาณสโตรบ (Strobe) ให้กับวงจรภายนอกที่เกี่ยวข้อง เพื่อให้สามารถส่งหรือรับข้อมูลจาก CPU ได้ในช่วงเวลาที่เหมาะสม นอกจากนี้สัญญาณ GROUPSELECT ยังถูกนำไปใช้ในการขับบัฟเฟอร์ 74LS245 ด้วย เพื่อให้ CPU สามารถส่งหรือรับข้อมูลจากอุปกรณ์ภายนอกได้เมื่อแอดเดรสในกลุ่มนี้ถูกเลือก สำหรับทิศทางของข้อมูลจะถูกควบคุมโดยสัญญาณ IOR ส่วนสัญญาณ AEN จะถูกนำมาใช้ในการดิสเอเบิลวงจรดีโค้ดโดยกำสัญญาณ AEN เป็น "1" ซึ่งเป็นช่วงเวลาของขบวนการ DMA นั้น 74LS138 จะถูกดิสเอเบิลทันที ทั้งนี้ก็เพื่อป้องกันความผิดพลาดที่อาจเกิดขึ้น เนื่องจากการดีโค้ดแอดเดรสของพอร์ทในระหว่างขบวนการ DMA นั่นเอง (ในระหว่างนี้แอดเดรสบนบัสแอดเดรสจะเป็นแอดเดรสของหน่วยความจำ คือ สัญญาณ MEMW หรือ MEMR จะแอคทีฟ แต่ในขณะเดียวกันสัญญาณ IOR หรือ IOW ก็จะแอคทีฟด้วย ดังนั้น ถ้าไม่มีดิสเอเบิลวงจรดีโค้ดไว้แล้ว อาจจะทำให้วงจรดีโค้ดคิดว่าแอดเดรสบนบัสแอดเดรสเป็นแอดเดรสของพอร์ท I/O ก็ได้) ในการดีโค้ดแอดเดรสของพอร์ท I/O เราจะต้องคำนึงถึงช่วงเวลาของสัญญาณที่เกิดขึ้นในขบวนการอ่านหรือเขียนข้อมูลลงบนพอร์ท I/O ดังนี้

1. ในช่วงเริ่มต้นของบัสไซเคิลที่เกี่ยวกับพอร์ท I/O นั้น ถ้าสัญญาณจากวงจรดีรหัสมีการหน่วงเวลา (Deley) มากเกินไป อาจจะทำให้สัญญาณดีรหัสนี้เกิดขึ้นหลังจากที่สัญญาณนี้แอดเดรสบนบัสแอดเดรสนั้นเปลี่ยนแปลงได้ตลอดเวลา ดังนั้นก่อนที่ค่าแอดเดรสที่ถูกต้องจะถูกส่งออกมาบนบัสแอดเดรสนั้น วงจรดีรหัสจะได้รับค่าแอดเดรสอื่น ๆ อยู่ ซึ่งถ้าหากวงจรดีรหัสมีการหน่วงเวลามากเกินไปแล้ว สัญญาณดีรหัสแอดเดรสที่ไม่ถูกต้องนี้อาจจะถูกหน่วงเวลาจนเกิดขึ้นในช่วงเวลาที่สัญญาณ IOR หรือ IOW เกิดขึ้นแล้วก็ได้ ทำให้ข้อมูลบนบัสข้อมูลนั้นถูกส่งไปยังพอร์ทที่ไม่ถูกต้อง สำหรับใน IBM/PC จะถูกออกแบบให้การหน่วงเวลาในวงจรดีรหัสนั้นมีค่าไม่เกิน 92 nanosec.

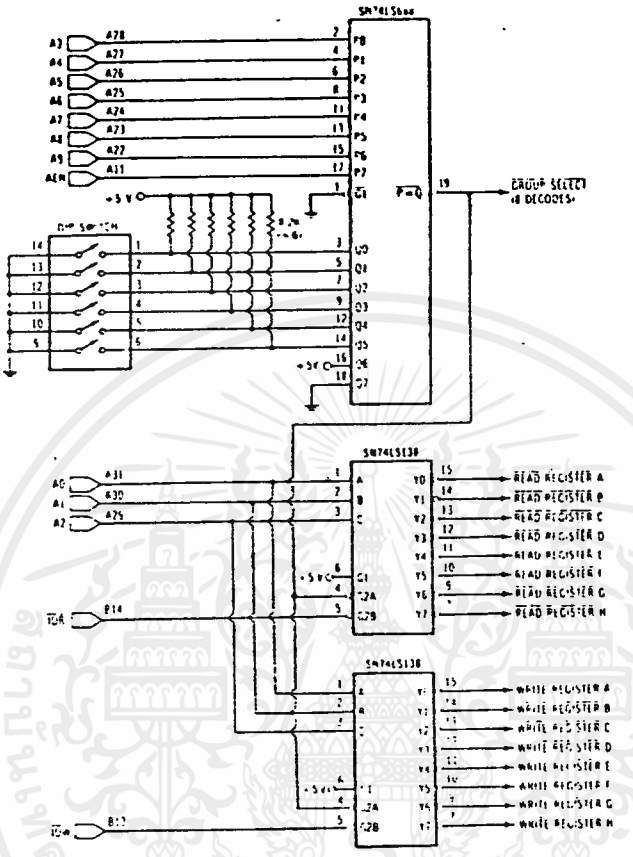
2. ในช่วงท้ายของบัสไซเคิลในการเขียนข้อมูลลงบนพอร์ท I/O นั้นถ้าสัญญาณ IOW มีการหน่วงเวลาออกไป และวงจรดีรหัสมีความเร็วในการทำงานสูงแล้ว อาจจะทำให้ข้อมูลในบัสไซเคิลนี้ถูกส่งให้กับพอร์ท I/Oที่มีแอดเดรสกับค่าแอดเดรสในบัสไซเคิลต่อไปก็ได้ สำหรับ IBM/PC สัญญาณ IOW จะมีการหน่วงเวลาไปไม่เกิน 200 nanosec.

อย่างไรก็ตามช่วงเวลาที่ต้องสนใจมากอีกช่วงเวลาหนึ่ง ก็คือ ช่วงเวลาระหว่างขอบขาขึ้นสัญญาณ IOW กับช่วงเวลาที่ข้อมูลที่ถูกต้องถูกส่งออกมาบนบัสข้อมูล ถ้าสัญญาณ IOW ถูกหน่วงเวลาไปเกินกว่า 120 nanosec. แล้วอาจจะทำให้พอร์ท I/O ได้รับข้อมูลที่ไม่ถูกต้องก็ได้และสำหรับสัญญาณ IOR นั้นถ้ามีการหน่วงเวลาเกิดขึ้นแล้ว ก็จะทำให้ความเร็วในการอ่านข้อมูลถูกลดลง

- การดีรหัสโดยใช้สวิทช์เลือก

การดีรหัสในแบบ Fixed ที่ได้กล่าวไว้ในหัวข้อที่ผ่านมานั้น มีข้อเสียอยู่บางประการคือแอดเดรสที่เราเลือกใช้งานไว้นั้นอาจจะชำกับแอดเดรสของการ์ดอื่น ๆ ที่เรานำมาเพิ่มเข้าไปในระบบในภายหลังก็ได้ ซึ่งกรณีเช่นนี้เราต้องแก้ไขวงจรเพื่อหลีกเลี่ยงไปใช้แอดเดรสอื่นที่ยังว่างอยู่ และ ไม่ถูกใช้งานโดยการ์ดที่จะเพิ่มเข้าไปใหม่ซึ่งยุ่งยากและต้องเสียเวลามากขึ้น ปัญหาเช่นนี้เราสามารถแก้ไขได้โดยใช้วงจรดีรหัสที่สามารถเปลี่ยนแปลงค่าแอดเดรสได้ โดยเพียงแต่เปลี่ยนตำแหน่งของสวิทช์ (ในที่นี้คือ DIP Switch) ที่เซ็ทไว้ในวงจรเท่านั้น

จากรูปเป็นวงจรที่ทำการดีรหัสการ์ดสุ่มแอดเดรสขนาด 8 แอดเดรสซึ่งการเลือกการ์ดสุ่มแอดเดรสที่จะทำการดีรหัสนี้จะทำได้โดยการเซ็ท DIP Switch ที่ขา Q0-Q5 ของ 74LS688 สำหรับหน้าที่ของ 74LS688 นี้จะทำการเปรียบเทียบค่าของอินพุท 2 ชุดที่ถูกส่งเข้ามาทางขา P0-P7และขา Q0-Q7 ถ้าอินพุททั้ง 2 ชุดนี้เท่ากันแล้ว เอาท์พุทที่ขา P=Q จะให้เอาท์พุทเป็นลอจิก "0" จากในวงจรขา P0-P6 ของ 74LS688 ต่อกับแอดเดรสบัส A3-A9 ในขณะที่ขา Q0-Q5 ต่อกับความต้านทานที่ทำหน้าที่เป็น Pull Up (รักษาระดับแรงดันให้เป็นลอจิก "1" ไว้ในกรณีที่ไม่มีอินพุทใด



รูปที่ 19 ตัวอย่างวงจรดีโคดแอดเดรสโดยใช้สวิทช์เลือก

ๆ เข้ามา) และ ขา Q0-Q5 นี้จะต่อกับบลายข้างหนึ่งของ DIP Switch ด้วย ส่วนบลายอีกข้างหนึ่งของ DIP Switch นั้นจะต่อลง Ground (ลอจิก "0") ไว้ ในขณะที่ทำ DIP Switch ที่ต่อกับขาใดถูก "OFF" ขานั้นก็จะได้รับลอจิก "1" และเนื่องจากอินพุทที่ขา P0-P5 (แอดเดรส A3-A9) ต้องเท่ากับอินพุทที่ขาQ0-Q5 ดังนั้นถ้าเราเปลี่ยนแปลงการเซ็ท DIP Switch เหล่านี้ก็จะทำให้แอดเดรสบิท A3-A5ซึ่งต่อกับขา P0-P5 นั้นต้องเปลี่ยนแปลง ตามไปด้วยจึงจะทำให้เอาท์พุทของ 74LS688 แอคทีฟได้ ทำให้เราสามารถเปลี่ยนค่าแอดเดรสที่ต้องการจะดีโคดได้ง่ายกว่าวิธีการดีโคดแบบ Fixed สำหรับขา Q6 นั้นจะต่อกับลอจิก "1" (+5 V) และขา P6 ต่อกับแอดเดรสบิท A9 ในการนี้เช่นนี้จึงเท่ากับเป็นการบังคับให้แอดเดรสที่จะทำการดีโคดได้นั้น จะต้องมีแอดเดรสบิท A9 เป็น "1" เท่านั้น สำหรับขา P7 จะต่อกับสัญญาณ AEN ซึ่งมีขา Q7 ต่อกับลอจิก "0" การต่อในลักษณะนี้ก็เพื่อป้องกันไม่ให้ 74LS688 ทำการดีโคดในระหว่าง

ขบวนการ DMA นั่นเอง เอาท์พุทจากขา P=Q ของ 74LS688 นี้ จะถูกนำไปใช้ในการอินาเบิล 74LS138 ซึ่งทำหน้าที่ในการดีโค้ดแอดเดรส 8 แอดเดรสของกลุ่มแอดเดรสที่เราเลือกา

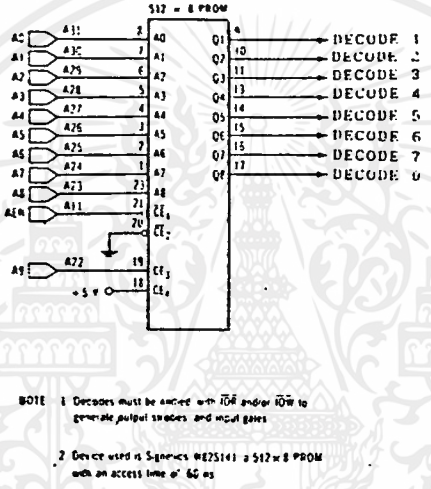
วงจรในลักษณะนี้ เราสามารถจะนำไปใช้เป็นการดีโค้ดในแบบ Fixed ได้โดยการนำเอา DIP Switch ออก จากนั้นถ้าอินพุทใดต้องการลอจิก "0" จึงจะใช้ตัวนำเชื่อมต่อระหว่างขั้วทั้งสองแทนการเซ็ท DIP Switch ให้ "ON" แต่ถ้าอินพุทใดต้องการลอจิก "1" ก็ปล่อยขั้วทั้งสองนั้นไว้

- การดีโค้ดโดยใช้ PROM

การดีโค้ดในแบบต่าง ๆ ที่ได้กล่าวมาแล้วนั้น เป็นการดีโค้ดในลักษณะที่แอดเดรสของพอร์ทต่าง ๆ อยู่รวมกันเป็นกลุ่ม แต่ในบางกรณีพอร์ทที่เราใช้นั้นมีแอดเดรสแยกกันอย่างเป็นอิสระ เช่นในการนำเอาหน้าที่การทำงานที่อยู่บนการ์ดต่าง ๆ มารวมไว้บนการ์ดเพียงการ์ดเดียว และมีความจำเป็นต้องคงค่าแอดเดรสของพอร์ทเดิม (ที่อยู่บนการ์ดเดิม) ไว้ด้วย ทำให้ไม่สามารถใช้การดีโค้ดในแบบต่าง ๆ ที่ผ่านมาได้ เนื่องจากการใช้วิธีการดีโค้ดในแบบที่ผ่านมานั้นจะทำให้ต้องใช้อุปกรณ์ที่ทำการดีโค้ดนั้นมากเกินไป ในกรณีเช่นนี้เราจำเป็นต้องใช้การดีโค้ดอีกแบบหนึ่งซึ่งจะได้กล่าวถึงในหัวข้อนี้ คือการดีโค้ดโดยใช้ PROM (Programmble Read Only Memory)ดังในรูปที่ 20

จากรูปข้างต้นเป็นวิธีการง่าย ๆ แบบหนึ่งในการดีโค้ดโดยใช้ PROM ซึ่งจะเห็นได้ว่าเราใช้เส้นแอดเดรส A0-A8 ของระบบต่อเข้ากับเส้นแอดเดรส A0-A8 ของ PROM และใช้บัสข้อมูลทั้ง 8 ของ PROM คือ Q1-Q8 เป็นเอาท์พุท สำหรับใช้เป็นสัญญาณดีโค้ดให้กับบอร์ดต่าง ๆ 8 พอร์ท อย่างไรก็ตามสัญญาณดีโค้ดทั้ง 8 เส้น คือ DECODE 1-DECODE 8 นี้ยังคงต้องนำไป OR กับสัญญาณ IOR หรือ IOW ก่อนที่จะนำไปอินาเบิลพอร์ทที่มีแอดเดรสตรงกับแอดเดรสที่ป้อนให้กับ PROM นั้น จากที่ได้กล่าวมานั้นจะเห็นได้ว่าส่วนของวงจรดีโค้ดนั้น จะมี PROM เพียงตัวเดียวเท่านั้น ซึ่ง PROM ที่จะนำมาใช้งานนี้จะต้องถูกโปรแกรมมาก่อน แล้ว โดยข้อมูลที่โปรแกรมให้กับแอดเดรสต่าง ๆ ของ PROM นั้น จะต้องสัมพันธ์กับสัญญาณดีโค้ดที่เราต้องการ กล่าวคือเราจะต้องทราบเสียก่อนว่าค่าแอดเดรสของพอร์ททั้ง 8 ที่เราต้องการจะดีโค้ดนั้นมีแอดเดรสใดบ้าง แล้วจึงกำหนดว่าพอร์ทใดจะใช้สัญญาณดีโค้ดเส้นใด จากนั้นจึงโปรแกรมข้อมูลให้กับ PROM โดยแอดเดรสใดถ้าต้องการให้สัญญาณดีโค้ดใดแอคทีฟ (ในที่นี้จะกำหนดให้สัญญาณดีโค้ดแอคทีฟที่ลอจิก "0") ก็กำหนดให้ข้อมูลในบิทที่ตรงกับสัญญาณดีโค้ดนั้นเป็น "0" เช่นถ้าเราต้องการกำหนดให้แอดเดรสของพอร์ทที่เราต้องการจะดีโค้ดเป็น 0393H และเลือกใช้สัญญาณ DECODE5 เราก็จะต้องทำการโปรแกรมให้แอดเดรส 0193H ของ PROM (เหตุที่แอด

เดรสของPROM เป็น 0193H แทนที่จะเป็นแอดเดรส 0393H เหมือนกับแอดเดรสของพอร์ท ก็เพราะแอดเดรสของ PROM มีเพียง 9 บิท คือ A0-A8 เท่านั้น ส่วนบิท A9 จะถูกต่อเข้ากับ PROM ในภายหลัง เพื่อยืนยาเบิล PROM เมื่อข้อมูลในบิท A9 นี้เป็น "1" เท่านั้น) มีข้อมูลในบิท Q5 (ถ้านับเริ่มจากบิท D0 ก็คือบิท D4)เป็น "0" ส่วนบิทอื่น ๆ นั้นมีค่าเป็น"1" ทั้งหมด ดังนั้น การโปรแกรมแอดเดรส 0193H ของ PROM จึงต้องโปรแกรมด้วยข้อมูล DEFH เป็นต้น สำหรับข้อมูลในแอดเดรสอื่น ๆ ที่นอกเหนือจากแอดเดรสทั้ง 8 ที่กำหนดแล้วจะต้องโปรแกรมให้ข้อมูลทุกบิทเป็น "1" ทั้งหมด ซึ่งก็คือโปรแกรมด้วยข้อมูล OFFH นั่นเอง



รูปที่ 20 การใช้ EPROM ในการดีโค้ด

บทที่ 4

การใช้งาน 8255 PIO

เป็นพอร์ตอินพุท/เอาท์พุทแบบขนาน (Parallel Input/Output port) ทำงานได้ 3 โหมด ดังต่อไปนี้

โหมด 0 พอร์ตทุกตัวเป็นพอร์ตอินพุท/เอาท์พุทพื้นฐาน

โหมด 1 พอร์ต A และ B เป็น พอร์ตอินพุท/เอาท์พุท พอร์ต C เป็น handshake

เช่น A เป็น พอร์ตอินพุท C บน เป็น Acknowledge

B เป็น พอร์ตเอาท์พุท C ล่างเป็น Strobe

โหมด 2 ทำได้เฉพาะพอร์ต A ให้เป็นพอร์ตอินพุท/เอาท์พุท เป็นบัสสองทิศทาง และมี handshake ทั้งคู่ (พอร์ต C เป็น handshake) เมื่อโปรแกรมพอร์ต A แล้ว พอร์ต B ทำงานอิสระ เป็น พอร์ตอินพุท/เอาท์พุท โหมด 1 หรือ โหมด 0 ก็ได้

สถานะขาต่างๆในการควบคุมการทำงาน

- ต้องการให้อ่านข้อมูล ขา RD ต้องเป็น '0'
- ต้องการให้เขียนข้อมูล ขา WR ต้องเป็น '0'
- ขณะทำงาน ขา CS ต้องเป็น '0'
- การรีเซต ขา RESET แอคทีฟที่ลอจิก '1'
- การเลือกใช้พอร์ต ทำได้โดยควบคุมที่ขา A_0 และ A_1

00 เลือก พอร์ต A

01 เลือก พอร์ต B

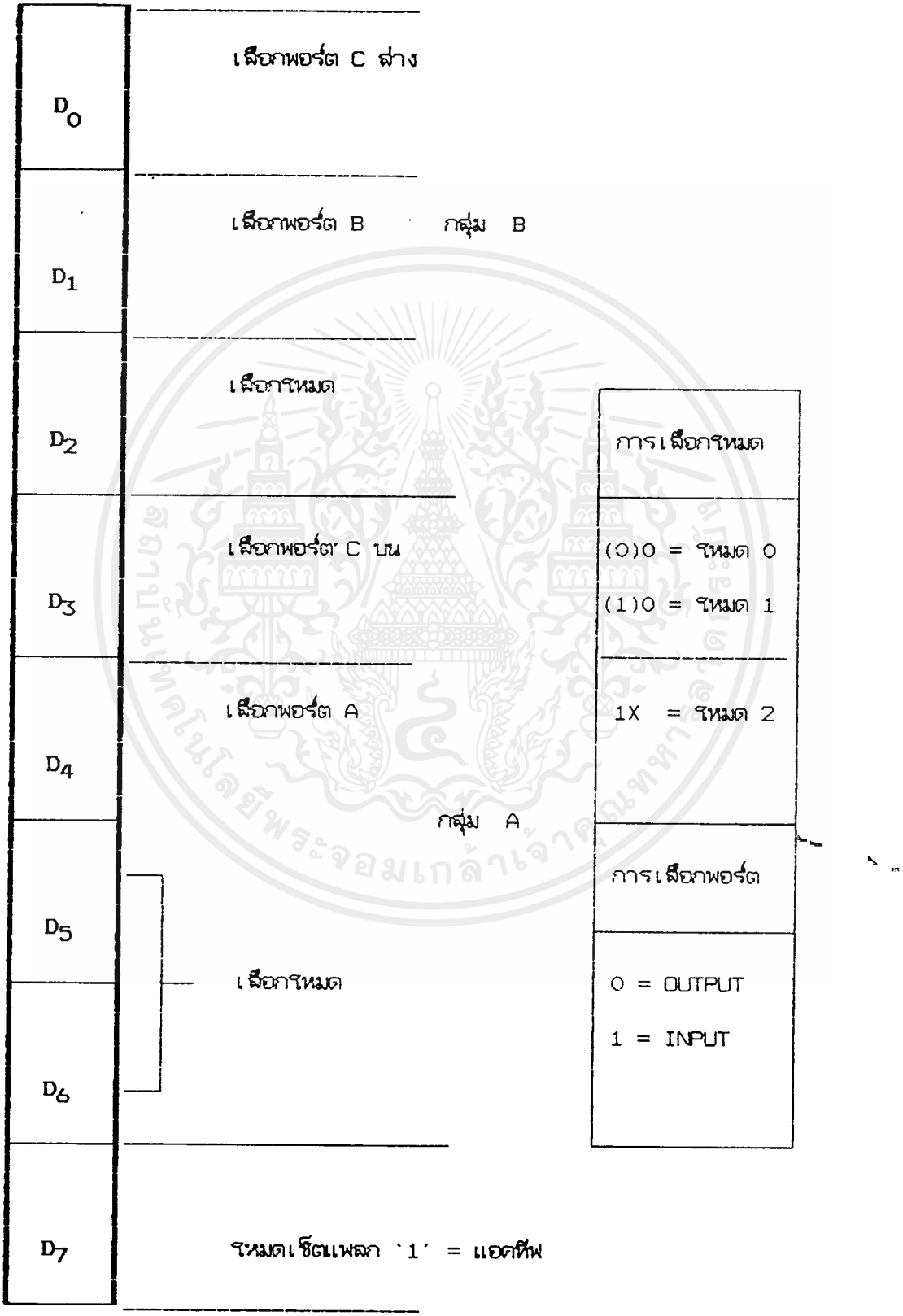
10 เลือก พอร์ต B

11 ให้เขียนรหัสควบคุม (มีความหมายเฉพาะขา WR แอคทีฟเท่านั้น)

รายละเอียดอื่น ๆ ของ 8255 ศึกษาได้จากภาคผนวก

การเขียนรหัสควบคุม

จะต้องเขียนลงบนขา D_0-D_7 สั่งให้แต่ละพอร์ตทำงาน โดยแต่ละบิทมีความหมายดังต่อไปนี้



บทที่ 5

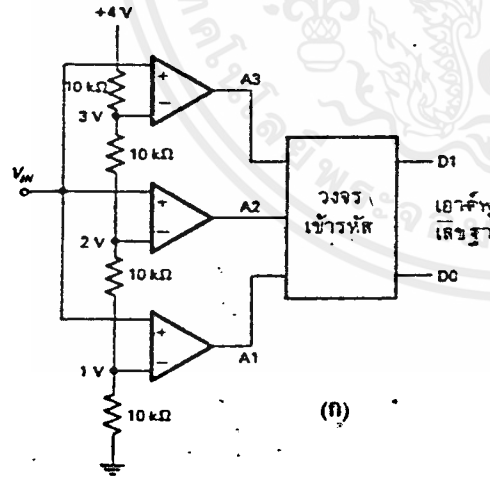
พื้นฐานวงจร เอทูดี

วงจรเปลี่ยนสัญญาณอะนาลอกเป็นดิจิตอล (analog to digital converter) ที่ใช้กันอยู่ทั่ว ๆ ไป มีหลายแบบคือ

5.1 แบบใช้วงจรเปรียบเทียบขนานหรือแบบ "แฟลช"

(Parallel comparator Simultaneous 1"Flash" A/D Converter)

วงจรเอทูดีแบบนี้ใช้หลักการง่าย ๆ อีกทั้งยังเป็นวิธีที่รวดเร็วที่สุด คือใช้วงจรเปรียบเทียบต่อขนานกัน ดังรูปที่ 21 ประกอบด้วยออปแอมป์ที่ต่อเป็นวงจรเปรียบเทียบและตัวต้านทานต่อไว้เพื่อแบ่งแรงดันที่ขาอินพุตแบบกลับ (inverting) ให้มีขนาดต่าง ๆ กัน



(ก)

แรงดันอินพุต V_{in} (โวลต์)	เอาต์พุตของวงจรเปรียบเทียบ A1	A2	A3	เอาต์พุตเลขฐานสอง D1	D0
0 – 1	0	0	0	0	0
1 – 2	1	0	0	0	1
2 – 3	1	1	0	1	0
3 – 4	1	1	1	1	1

(ข)

รูปที่ 21 (ก) แสดงการต่อวงจร Parallel comparator A/D converter

(ข) ตารางความสัมพันธ์ระหว่างแรงดันอินพุทที่เป็นอนาลอกกับเอาพุทที่เป็นดิจิตอล

จากหลักการของวงจรเปรียบเทียบทั่วไป เมื่อแรงดันอินพุตที่ขาอินพุตแบบไม่กลับ (noninverting) มีค่าสูงกว่าที่ขาอินพุตแบบกลับ เอาต์พุตจะได้แรงดันค่าสูง ดูได้จากตารางรูปที่ 21 จะเข้าใจยิ่งขึ้นว่า ที่แรงดันค่าต่าง ๆ มีผลต่อเอาต์พุตของวงจรเปรียบเทียบแต่ละตัว

อย่างไร ซึ่งเอาต์พุตที่ได้จากวงจร เปรียบเทียบนี้จะนำไปเข้ารหัสให้เป็นเลขฐานสองต่อไป จำนวนของวงจรเปรียบเทียบที่ต้องใช้ในวงจรขึ้นอยู่ กับขนาดของสัญญาณอะนาลอกที่อินพุตจากวงจร รูปที่ 21 ถ้าแรงดันอินพุตมีค่า 1 ระดับต่ำไม่เพียงพอที่จะทำให้วงจรเปรียบเทียบตัวใดให้ค่า เอาต์พุต เป็น "high"

ที่แรงดันระหว่าง 1 ถึง 2 ระดับวงจรเปรียบเทียบที่มีระดับเทรชโฮลด์ (threshold)ต่ำสุด ก็จะให้เอาต์พุตเป็น "high"

แรงดัน 2-3 ระดับ วงจรเปรียบเทียบทั้ง A_1 และ A_2 ให้เอาต์พุตเป็น "high" ถ้าแรงดัน อินพุตมากกว่า 3 ระดับ วงจรเปรียบเทียบก็จะให้เอาต์พุตเป็น "high" ทั้งหมด

เมื่อต้องการวงจรที่มีความละเอียดสูงขึ้น จำเป็นต้องใช้วงจรเปรียบเทียบเพิ่มขึ้น เช่น ถ้าต้องการความละเอียด 3 บิต ต้องใช้วงจรเปรียบเทียบ 7 ตัวความละเอียด 4 บิตต้องใช้วงจรเปรียบเทียบ 15 ตัว (16 ระดับ) โดยหาจำนวนวงจรเปรียบเทียบได้จาก 2^N-1 เมื่อ N แทนจำนวนบิต หรือ ความละเอียดที่ต้องการ

จะเห็นได้ว่าที่ความละเอียด 8 บิต ต้องใช้วงจรเปรียบเทียบมากถึง 255 ตัว ซึ่งเป็นข้อเสียของวงจรเอดีแบบนี้

ข้อเสียอีกประการหนึ่งคือ เอาต์พุตที่ได้ไม่เป็นเลขฐานสอง ต้องมีวงจรเพิ่มเติมไปทำการเข้ารหัส

ข้อดีของวงจรเอดีแบบขนานนี้คือ ความเร็วสูงมาก บางครั้งจึงเรียกวงจรเอดีแบบนี้ว่า "แฟลช" (flash type A/D converter) วงจรเอดีชนิดนี้ใช้เวลาในการแปลงได้เร็วในระดับนาโนวินาทีทีเดียว

5.2 วงจรเอดีที่ใช้การอินทีเกรต

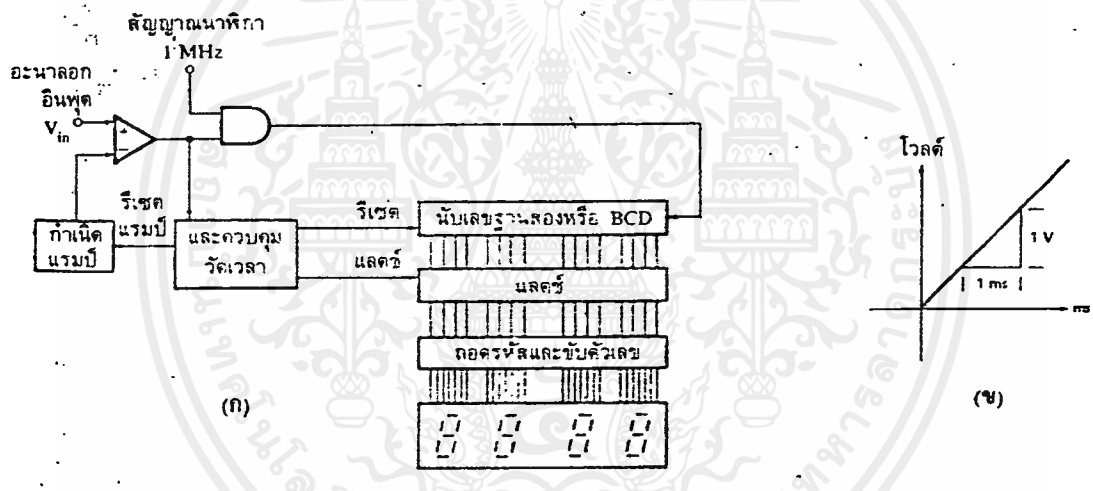
วงจรเปลี่ยนสัญญาณเอดีที่ใช้เทคนิคการอินทีเกรตสัญญาณมี 3 แบบ คือ

- แบบสโลปเดี่ยวหรือแบบแรมป์

(Single Ramp หรือ Single Slope A/D Converter)

วงจรเอดีแบบนี้แสดงไว้ดังรูปที่ 13 ประกอบด้วยวงจรกำเนิดสัญญาณแรมป์,วงจรเปรียบเทียบ,วงจรนับ BCD หรือนับเลขฐานสอง เมื่อเริ่มทำการเปลี่ยนสัญญาณสัญญาณแรมป์และวงจรนับจะถูกรีเซตให้เป็น 0 แรงดันอะนาลอกถูกป้อน ไปยังวงจรเปรียบเทียบทางขาอินพุตแบบไม่กลับเมื่อ

แรงดันอินพุตที่ขานี้เป็นบวกมากกว่าที่ขาอินพุตแบบกลับ วงจรเปรียบเทียบก็จะให้เอาต์พุตเป็นระดับ "high" ทำให้เอนด์เกตปล่อยสัญญาณนาฬิกา ผ่านไปยังวงจรนับได้ และทำให้เริ่มเกิดสัญญาณแรมป์ สัญญาณแรมป์มีแรงดันเป็นบวกขึ้นเรื่อย ๆ จนมากกว่าระดับแรงดันอินพุตเอาต์พุตของวงจรเปรียบเทียบก็ตกลงมาเป็นระดับ "low" ปิดเอนด์เกตไม่มีสัญญาณผ่านไปให้วงจรนับวงจรนับจะหยุดนับและ เก็บค่าไว้ที่วงจรแลตช์ จากนั้นจึงทำการรีเซตวงจรนับและวงจรก็เกิด สัญญาณแรมป์



รูปที่ 22 วงจรเปลี่ยนสัญญาณเอนูดีแบบสรุปเดี่ยว

(ก) แสดงบล็อกไดอะแกรม (ข) ความชันของสัญญาณแรมป์

สมมติให้สัญญาณนาฬิกามีความถี่ 1 MHz วงจรนับ BCD 4 หลัก แรงดันอินพุต V_{in} 2 โวลต์ สัญญาณแรมป์มีความชัน 1 V/ms ดังแสดงในรูป ข.

จากจุดเริ่มต้นจนถึงแรงดันสูงสุด (2 โวลต์) สัญญาณแรมป์ใช้เวลา 2 ms หลังจากนี้จึงปิดสัญญาณนาฬิกาที่ส่งไปให้วงจรนับ ในช่วง 2 ms นี้ มีการส่งพัลส์ไปให้วงจรนับเพื่อทำการนับถึง 2000 พูด เอาต์พุตของวงจรเปรียบเทียบระดับ "high" เป็นการสั่งสัญญาณให้วงจรแลตช์ส่งค่าที่นับได้ไปยังภาคแสดงผล และเติมจุดทศนิยมที่ตำแหน่งที่เหมาะสมของตัวแสดงผลได้เป็นค่า 2,000 ที่

แรงดันอินพุต 2 โวลต์ วงจรแบบนี้เป็นหลักการเบื้องต้นของดิจิตอลโวลต์มิเตอร์ ซึ่งถ้าใช้วงจรนับเลขฐานสองแทนแบบ BCD เอาต์พุตก็จะอ่านได้ค่าเลขฐานสองโดยตรง วงจรลักษณะนี้มักนำไปใช้งานในการเปลี่ยนเวลาเป็นขนาดของสัญญาณ (time to amplitude conversion) หรืออาจใช้ในดิจิตอลโวลต์มิเตอร์ แต่ไม่ใช้กับงานที่ต้องการความถูกต้องสูง เนื่องจากการเปลี่ยนแปลงในแหล่งกำเนิดสัญญาณแรมป์ขึ้นกับอุณหภูมิและ ผลตอบสนองต่อสัญญาณอินพุต ทำให้ไม่มี ความคงที่ดังนั้นจึงมีการปรับปรุงให้ดีขึ้นกลายเป็นแบบ สโลปคู่ (dual-slope)

- แบบสโลปคู่

(Dual-Slope A/D converters)

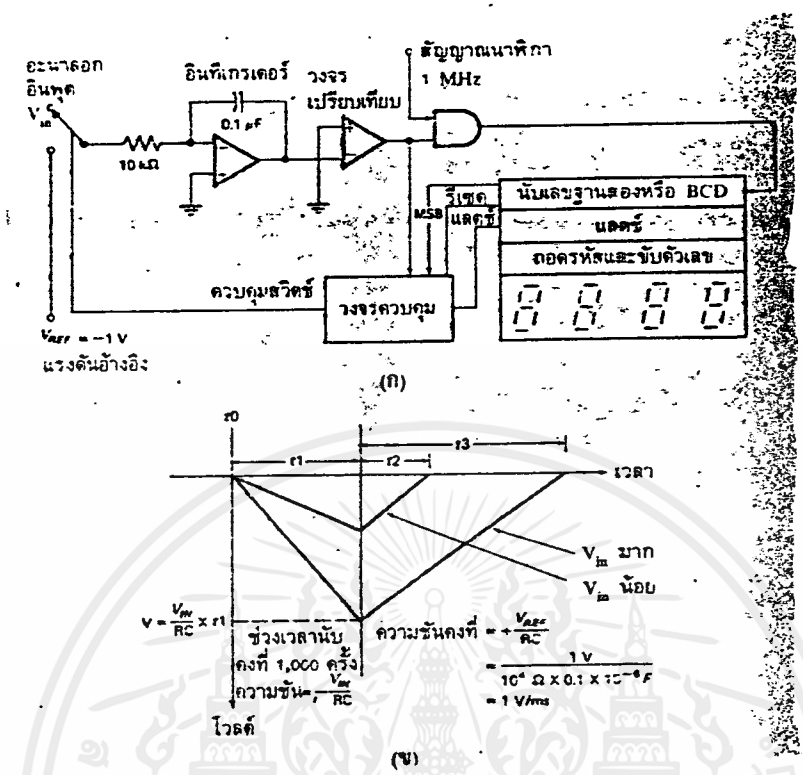
รูปที่ 23 ก. แสดงบล็อกไดอะแกรมของวงจรเอดีแบบสโลปคู่ ซึ่งวงจรส่วนใหญ่คล้ายกันแบบสโลปเดี่ยว แต่มีสวิตช์ที่อินพุตเพิ่มขึ้นเพื่อทำการเลือกระหว่างแรงดันอินพุตกับแรงดันอ้างอิง (วงจรเปรียบเทียบต่อขาสัญญาณอินพุตกลับกันกับแบบสโลปเดี่ยว) ส่วนแรกของวงจรคือ วงจรกำเนิดสัญญาณแรมป์หรือวงจรอินทีเกรเตอร์นั่นเอง ที่อินพุตแบบกลับของออปแอมป์มีสภาพเป็นการาวด์เทียม (virtual ground) ถ้ามีแรงดันอินพุต 2 โวลต์ จะได้กระแสไหลผ่านตัวต้านทาน 10 k เท่ากับ 0.2 mA ไปยังจุดรวม (summing point) เนื่องจากค่าความต้านทานอินพุตของออปแอมป์นั้นสูงมาก กระแสที่ไหลจึงเกิดขึ้นผ่านตัว เก็บประจุ ขณะที่ตัวเก็บประจุทำการชาร์จ (รับประจุ) แรงดันที่เอาต์พุตของออปแอมป์ก็จะยิ่งเป็น ลบมากขึ้นเรื่อย ๆ เพื่อรักษาระดับกระแสให้คงที่ แรงดันคร่อมตัวเก็บประจุจึงได้เป็นสัญญาณแรมป์ที่เป็นเชิงเส้น (linear ramp)

ถ้าแรงดันอินพุตเป็นบวก วงจรอินทีเกรเตอร์จะให้เอาต์พุตเป็นสัญญาณแรมป์ทางลบ ดังแสดงไว้ในช่วง t_1 รูปที่ 23 ข. หากแรงดันอินพุตเป็นลบก็จะทำให้เอาต์พุตได้แรมป์ทางบวก ความชันของสัญญาณแรมป์ สามารถคำนวณได้จากความสัมพันธ์ของประจุ $q = cv$ และ $q = It$ โดยจับสองสมการมาเท่ากัน

$$\frac{dv}{dt} = \frac{I}{C}$$

เมื่อรู้ว่ากระแสเท่ากับ $\frac{V_{in}}{R}$ เราก็รู้ว่า

$$\frac{dV}{dt} = \frac{Vin}{RC}$$



รูปที่ 23 วงจรเปลี่ยนสัญญาณเอทูดีแบบสองสไลปคู่

(ก) แสดงบล็อกไดอะแกรม (ข) เอาพุทของวงจรอินทีเกรตเมื่อกับเวลา

จากรูปถ้าแรงดันอินพุต +2 โวลต์ก็จะได้ความชันของสัญญาณแรมป์ทางเอาต์พุตเท่ากับ −2V/ms จากวงจรในรูปที่ 23.ข อธิบายได้ดือ เมื่อสวิตช์ต่อกับสัญญาณอินพุตจะทำให้มีแรงดันบวกจากอินพุตป้อนเข้าสู่วงจรอินทีเกรเตอร์ ได้เอาต์พุตออกมาเป็นแรมป์ทางลบ วงจรเปรียบเทียบก็จะได้แรงดันลบจากวงจรอินทีเกรเตอร์ แล้วทำให้เอาต์พุตเป็นบวกทำการเปิดแอนด์เกตทำให้สัญญาณนาฬิกาผ่านเข้าไปสู่วงจรนับ วงจรนับจะนับไปยังค่าที่กำหนดไว้คงที่ (t_1) แล้วทำการสับสวิตช์ต่อเข้ากันกับแรงดันอ้างอิง ในช่วงที่วงจรนับด้วยค่าคงที่นั้นวงจรอินทีเกรตจะทำให้สัญญาณแรมป์ทางลบที่มีค่าต่ำสุงสุดตามแต่ระดับแรงดันอินพุต เมื่อทำการสวิตช์อินพุตของวงจรอินทีเกรเตอร์ทำให้ไปที่แรงดันอ้างอิงค่าลบ เอาต์พุตของวงจรจึงได้เป็นแรมป์ทางบวกคือ ช่วง t_2 รูปที่ 23 ข. พร้อม ๆ กับรีเซตค่าของวงจรนับลงเป็นศูนย์เพื่อเริ่มนับใหม่

เมื่อเอาต์พุตของวงจรอินทีเกรเตอร์เพิ่มขึ้นถึงแรงดัน 0 อีกครั้ง เอาต์พุตของวงจรเปรียบเทียบก็จะเป็นลบ (หรือเป็นศูนย์) วงจรควบคุมจับการเปลี่ยนแปลงอันนี้ได้ก็ส่งสัญญาณสวิตรบกวนวงจรนับเก็บค่าที่ได้ไว้ที่วงจรแลตช์จากนั้นจึงรีเซตให้เป็นศูนย์แล้วทำการสวิตช์ให้อินพุตเป็นการเริ่มทำการเปลี่ยนสัญญาณอีกรอบหนึ่ง จำนวนที่นับได้ที่เก็บไว้ในวงจรแลตช์ก็จะเป็น สัดส่วนโดยกับแรงดัน

อินพุต V_m สัญญาณแรมป์ทางเอาต์พุตของวงจรอินทีเกรเตอร์ในช่วงเวลาคงที่ t_1 จะลดลงสู่แรงดัน V ซึ่ง

$$V = \frac{(Vin \times t_1)}{RC}$$

เพื่อให้กลับไปสู่ระดับ 0 หน้าที่ของวงจรอินทีเกรเตอร์จึงต้องสร้างแรมป์ทั้งทางบวกทางลบให้เพิ่มขึ้นเท่า ๆ กันในช่วงเวลา t_2 (ที่เกิดจากแรงดันอินพุตอ้างอิง) แรงดัน V เท่ากับ

$$V = \frac{(Vref \times t_2)}{RC}$$

สูตรทั้ง 2 ของ V สามารถจับมาเท่ากันได้เป็น

$$\frac{Vin \times t_1}{RC} = \frac{Vref \times t_2}{RC}$$

$$V_{in} \times t_1 = V_{REF} \times t_2$$

$$t_2 = V_{in} \times \frac{t1}{V_{REF}}$$

เห็นได้ว่า RC ปรากฏอยู่ทั้ง 2 ข้างของสมการ จึงสมารถตัดทิ้งได้ หมายถึง ว่าเมื่อช่วงเวลาอินทีเกรตสัญญาณและช่วงเวลาอินทีเกรตอ้างอิง ใช้ตัวต้านทานและตัวเก็บประจุค่าเดียวกัน การเปลี่ยนแปลงของค่าทั้งสองนี้ก็จะไม่มีผลต่อความถูกต้องของสัญญาณเอาต์พุต ซึ่งเป็นข้อดีที่เหนือกว่าแบบสโลปเดียว (single slope) คือค่าที่ได้ไม่ขึ้นกับความถี่ของรอบการทำงานสมการท้ายสุดแสดงให้เห็นว่าเอาต์พุตของวงจรนับในช่วงเวลา t_2 เป็นสัดส่วนโดยตรงกับแรงดันอินพุต V_{in}

เมื่อ V_{REF} และ t_1 คงที่

จากวงจรในรูปที่ 23 t_1 เท่ากับ 1000 รอบ

เมื่อป้อนสัญญาณนาฬิกา 1 MHz (= $\frac{1}{1000}$ = 1 ms)

และ V_{REF} มีค่า -1 โวลต์

ถ้าสัญญาณอินพุตมีขนาด 2 โวลต์จะได้ช่วงเวลา $t_2 = (\frac{2V}{} \times 1000) = 2000$ รอบ(รอบของการนับ) จุดทศนิยมที่อยู่ทางขวามาทำให้ได้ผลลัพธ์ที่ภาคแสดงผล 2.000 กราฟในรูป 23 ข.แสดงว่าเมื่อสัญญาณอินพุตน้อยกว่านี้จะมีการเปลี่ยนแปลงอย่างไรบ้าง

เช่น อินพุต 0.8 โวลต์ t_2 จะได้ $(\frac{0.8\ V}{1\ V}) \times 1000$ เท่ากับ 800

รอบก็จะอ่านค่าได้ 0.800 หลักการเช่นนี้ถูกนำไปใช้อย่างแพร่หลายในดิจิตอลโวลต์มิเตอร์และเครื่องมืออื่น ๆ อีกหลายชนิดt_1 วงจรนับจึงถูกรีเซตให้เป็น 0 อินพุตของวงจรอินทีเกรเตอร์ก็จะถูกสวิตช์ต่อกลับมาที่แรงดันอ้างอิง(ที่มีแรงดันคงที่) ทำให้ความชันของสัญญาณแรมป์คงที่เพิ่มค่าขึ้นไปจนถึงระดับ 0 ช่วงเวลา t_2 นี้เป็น สัดส่วนโดยตรงกับสัญญาณอินพุต ถ้าดูรูป 23 ข. อีกครั้งพิจารณา ช่วง t_1 ซึ่งเป็นช่วงเวลาคงที่ และ t_2 ซึ่งความชันคงที่แล้วจะเข้าใจยิ่งขึ้น

ข้อดีของวงจรเปลี่ยนสัญญาณแบบสรุปคู่นี้คือ ความถูกต้องสูง ราคาถูก เสถียรภาพทางด้านอุณหภูมิ ข้อเสียคือความเร็วต่ำ ในการเปลี่ยนสัญญาณ 1 ครั้งอาจใช้เวลาถึง 100 ms (ในขณะที่แบบ "แฟลช" ใช้เวลาประมาณ 30 ns)

- แบบชาร์จบาลานซ์

  (Charge Balnace A/D Converter)

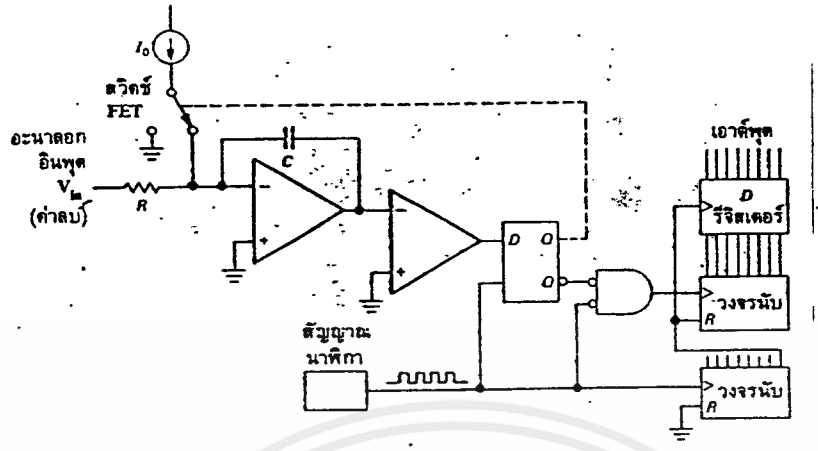
วงจรเปลี่ยนสัญญาณเอทูดีแบบชาร์จบาลานซ์ใช้วงจรลักษณะคล้ายกับแบบสรุปคู่นั่นเอง แต่แทนที่จะ ทำให้อินพุตสวิตช์ไปมาระหว่างแรงดันที่ไม่รู้ค่ากับแรงดันอ้างอิง ก็ทำการแหรกพัลส์ของกระแสอ้างอิงมา ตรง ๆ ที่จุดรวม (summing point) ของวงจรอินทีเกรเตอร์ในช่วงเวลาที่คงที่ โดยที่จำนวนของพัลส์ จะเป็นสัดส่วนโดยตรงกับแรงดันอินพุตที่ไม่รู้ค่า

ประโยชน์ของเทคนิคนี้คือแรงดันตกคร่อมตัวเก็บประจุของวงจรอินทีเกรเตอร์จะมีค่าเกือบเท่ากับ 0 V ดังนั้นจึงไม่เกิดความผิดพลาดจากผลของกระแสเสถียรไหล เอทูดีชนิดนี้ จึงมีความถูกต้องสูงกว่าแบบสรุปคู่

- แบบเดลต้า-ซิกม่า

  (Delta-Sigma A/D Converter)

จากวงจรรูปที่ 24 เมื่อมีแรงดันอินพุตป้อนเข้าไปที่วงจรอินทีเกรเตอร์จะให้เอาต์พุตเข้าไปในวงจร เปรียบเทียบ เปรียบเทียบกับแรงดันคงที่ (จากรูปคือ กราวด์) พัลส์ของกระ



รูปที่ 24 วงจรเปลี่ยนสัญญาณแบบเดลต้า-ซิกม่า

แสที่ได้ขึ้นอยู่กับเอาต์พุตของวงจรเปรียบเทียบ โดยสวิตช์ที่ทำขึ้นจากเฟตจะควบคุมให้กระแสเข้าไปยังที่จุดรวมหรือลงกราวด์ไป ส่วนวงจรนับจะนับจำนวนพัลส์ด้วยหลักการที่คล้ายกัน

ข้อสรุปของเอดูซีแบบอินทีเกรตสัญญาณ

จุดสำคัญของอินทีเกรติ้งเทคนิคคือ อินพุตที่ให้กับวงจรอินทีเกรเตอร์ต้องเป็นกระแส ไอซีคอนเวอร์เตอร์บางตัวอาจมีอินพุตให้สองขา แต่จะมีขาหนึ่งต่อตรงกับจุด summing point ใช้กับอุปกรณ์ที่เป็นแหล่ง จ่ายกระแสโดยตรง

ถ้าให้อินพุตเป็นกระแสก็ไม่ต้องคำนึงถึงแรงดันออฟเซตของวงจรอินทีเกรเตอร์ แต่ถ้าหากใช้กับอินพุตที่เป็นแรงดัน (ที่ต้องมีตัวต้านทานต่ออนุกรมอยู่เพื่อให้ได้เป็นกระแส) ต้องปรับออฟเซตของออปแอมป์เสียก่อน

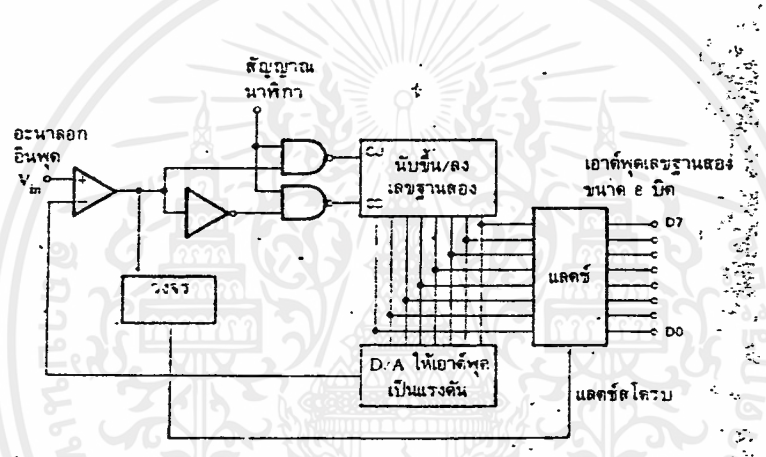
การใช้อินพุตเป็นกระแสทำให้ง่ายการใช้งานทางไฟสลับกว้างไอซีแบบชาร์จ-บาลานซ์มักประกอบด้วยวงจรแปลงแรงดันเป็นความถี่อยู่ด้วย ดังนั้นถ้าหากต้องการเอาต์พุตเป็นความถี่ก็สามารถเลือกได้

5.3 วงจรเปลี่ยนสัญญาณเอดูซีที่ใช้วงจรนับและวงจรดีทูเอประกอบกัน

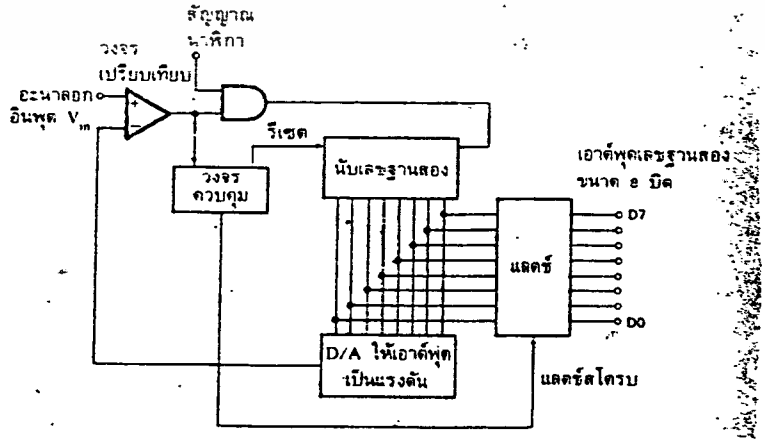
- แบบวงจรนับเดี่ยว (Single counter)

แท้ที่จริงแล้วสัญญาณแรมป์เชิงเส้น (linear ramp) อาจจะประกอบขึ้นด้วยสัญญาณขั้นบันไดเล็ก ๆ จำนวนมากที่เกิดจากการต่อเอาต์พุตของวงจรนับเข้ากับวงจรแปลงดีทูเอ ซึ่งขนาดของขั้นบันไดแต่ละขั้น ขึ้นอยู่กับจำนวนบิตหรือความละเอียดของวงจรดีทูเอนั้น ๆ

รูปที่ 25 แสดงการกำเนิดสัญญาณแรมป์เดี่ยวด้วยวงจรนับ และวงจรดีทูเอ (แหนวงจรอินทีเกรเตอร์) เมื่อเริ่มแปลงสัญญาณวงจรนับจะถูกรีเซต เอาต์พุตของวงจรดีทูเอมีระดับ 0 เมื่อแรงดันขาเข้าไปยัง อินพุตของวงจรเปรียบเทียบ เอาต์พุตก็จะขึ้นสู่ระดับ "high" และเปิดสัญญาณนาฬิกาไปสู่วงจรนับแต่ละพัลส์ของสัญญาณนาฬิกา ทำให้เกิดการนับและเพิ่มแรงดันขึ้น 1 ขั้น



รูปที่ 25 วงจรเอทูดีแบบวงจรนับเดี่ยวที่สร้างขึ้นจากวงจรนับขึ้น และวงจรดีทูเอ



รูปที่ 26 วงจรเอทูดีที่สร้างขึ้นจากวงจรนับขึ้นลง และวงจรดีทูเอ

เมื่อเอาต์พุตของดีทูเอมีค่ามากกว่าอินพุต V_{in} เอาต์พุตของวงจรเปรียบเทียบก็จะกลายเป็น "low" ทำให้สัญญาณนาฬิกาไม่อาจผ่านไปยังวงจรนับได้ ดังนั้นวงจรควบคุมจะทำการแลตช์เอาต์พุตของวงจรนับ และรีเซตวงจรนับก่อให้เริ่มต้น รอบใหม่อีกครั้งหนึ่ง

- แบบเทรกกิ้ง (Tracking A/D Converter)

การทำงานจะคล้ายกับแบบใช้วงจรนับเดี่ยว แต่การนับจะไม่ได้เริ่มจากศูนย์ แต่จะทำการนับขึ้น หรือนับลงจากค่าล่าสุดไปยังค่าใหม่ แล้วแต่ว่าแรงดันอินพุตในรอบใหม่มีค่าสูงกว่าหรือต่ำกว่าค่าที่แล้ว ข้อดีของเอดีแบบเทรกกิ้งคือทำงานได้เร็วขึ้น

5.4 วงจรเปลี่ยนสัญญาณเอดีแบบใช้การประมาณค่า
(Successive Approximation A/D Converter)

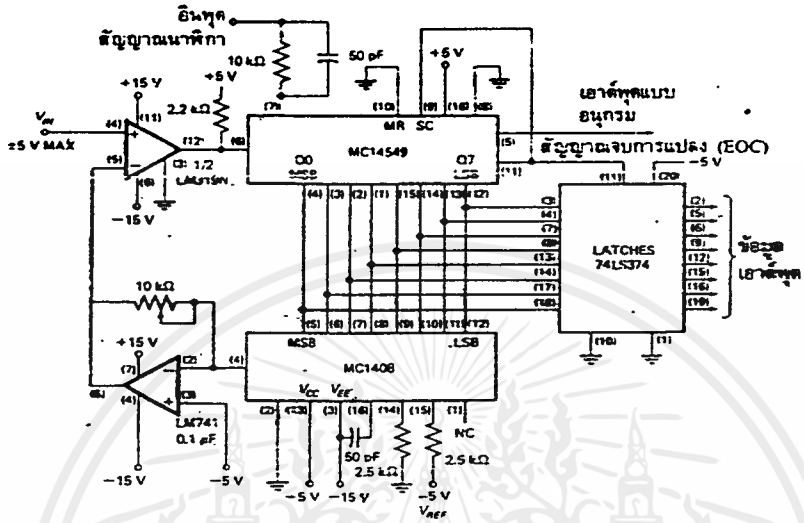
วงจรเอดีแบบนี้มีข้อได้เปรียบทางด้านความละเอียด เพราะความละเอียด n บิต สามารถกำหนดได้จากสัญญาณนาฬิกา n ลูก ตัวอย่างเช่น วงจรแปลงขนาด 8 บิต ต้องการพัลส์ของสัญญาณนาฬิกาเพียง 8 ลูก ในขณะที่แบบใช้วงจรนับต้องใช้พัลส์ถึง 256 ลูก วงจร SA (Successive-Approximation) นี้แสดงไว้ดังรูปที่ 7 หัวใจของวงจรคือ successive approximationregister (SAR) เช่น เบอร์ MC14549 ที่มีการทำงานดังต่อไปนี้

เมื่อเริ่มการเปลี่ยนสัญญาณพัลส์ลูกแรกจะทำการส่งบิตที่มีนัยสำคัญสูงสุดไปยังดีทูเอเบอร์ MC1408 โดย SAR จะรอสัญญาณจากวงจรเปรียบเทียบ LM319 ซึ่งทำการตรวจสอบว่าเอาต์พุตของวงจรดีทูเอ มากกว่าหรือน้อยกว่าแรงดันอินพุต V_{in} ถ้าเอาต์พุตของวงจรเปรียบเทียบมีระดับ "high" เอาต์พุตของดีทูเอจึงต่ำกว่า V_{in} SAR จะทำการเก็บบิตที่มีนัยสำคัญสูงสุดไว้ ถ้าเอาต์พุตของวงจรเปรียบเทียบเป็นระดับ "low" เอาต์พุตของดีทูเอจึงมากกว่า V_{in} SAR ทำการรีเซตบิตที่มีนัยสำคัญสูงสุดนั้น

พัลส์ลูกต่อมาก็ทำงานเช่นเดียวกัน โดยบิตที่ได้คือ บิตที่มีนัยสำคัญรองลงมา SAR ทำงานแบบนี้ไปจนถึงบิตที่มีนัยสำคัญต่ำสุด แต่ละบิตใช้สัญญาณนาฬิกาเพียงลูกเดียวครบทุกบิตแล้ว SAR ก็ทำการส่งสัญญาณ EOC (end of conversion) ออกไป

สัญญาณ EOC เป็นตัวบอกว่าสายสัญญาณเอาต์พุตที่ขนานกันมาทำเสนมีข้อมูลดิจิตอลของสัญญาณอินพุต ครบถ้วนแล้ว ถ้าสัญญาณ EOC ถูกต่อไปยังอินพุตที่เป็นจุดเริ่มการเปลี่ยนสัญญาณการเปลี่ยนสัญญาณก็จะเกิดขึ้นอย่างต่อเนื่อง

MC14549 ยังสามารถส่งเอาต์พุตดิจิตอลแบบอนุกรมได้อีกด้วย ขึ้นอยู่กับการควบคุมของ SAR



รูปที่ 27 วงจรเปลี่ยนสัญญาณเอหูดีแบบ Successive approximation

วงจรในรูปที่ 27 ใช้แรงดันอินพุตสูงสุด 5 โวลต์ อินพุตแบบไม่กลับของออปแอมป์ที่ทำหน้าที่เปลี่ยนกระแสเป็นแรงดัน (current to voltage converter) ต่อเข้ากับ −5 โวลต์ แทนที่จะต่อลงกราวด์ เป็นการยกระดับแรงดันอะนาลอกจาก −5 โวลต์ ถึง +5 โวลต์ แทน 0 ถึง 10 โวลต์ สัญญาณไพสลับรูปชายน์จึงสามารถต่อโดยตรงเข้ากับอินพุตของเอหูดีวงจรนี้ได้

วงจรแปลงเอหูดีชนิดนี้มีความเร็วสูง และความละเอียดสูง จึงเป็นวงจรที่นำมาใช้กันอย่างแพร่หลาย

- การสุ่มและการคงค่า (Sample and Holds)

วงจรเอหูดีต้องการเวลาในการแปลงสัญญาณหรือที่เรียกว่า conversion time เพื่อเปลี่ยนสัญญาณอะนาลอกเป็นสัญญาณดิจิตอลที่เหมาะสม ถ้าสัญญาณอะนาลอกมีการเปลี่ยนแปลงในระหว่างช่วงเวลาการแปลง เอาต์พุตของวงจรเปลี่ยนสัญญาณอาจเกิดความผิดพลาดจึงต้องมีการป้องกันด้วยวงจรสุ่มและคงค่า สัญญาณ (sample and hold) เพื่อใช้จับสัญญาณอะนาลอกที่จุดเริ่มของการเปลี่ยนสัญญาณ แล้วเก็บไว้ใน ตัวเก็บประจุระหว่างช่วงเวลาการแปลง หลังจากที่การเปลี่ยนสัญญาณเสร็จสิ้น จึงจับสัญญาณอะนาลอก ค่าใหม่มาเก็บไว้อีกครั้งเป็นเช่นนี้ต่อ ๆ ไป

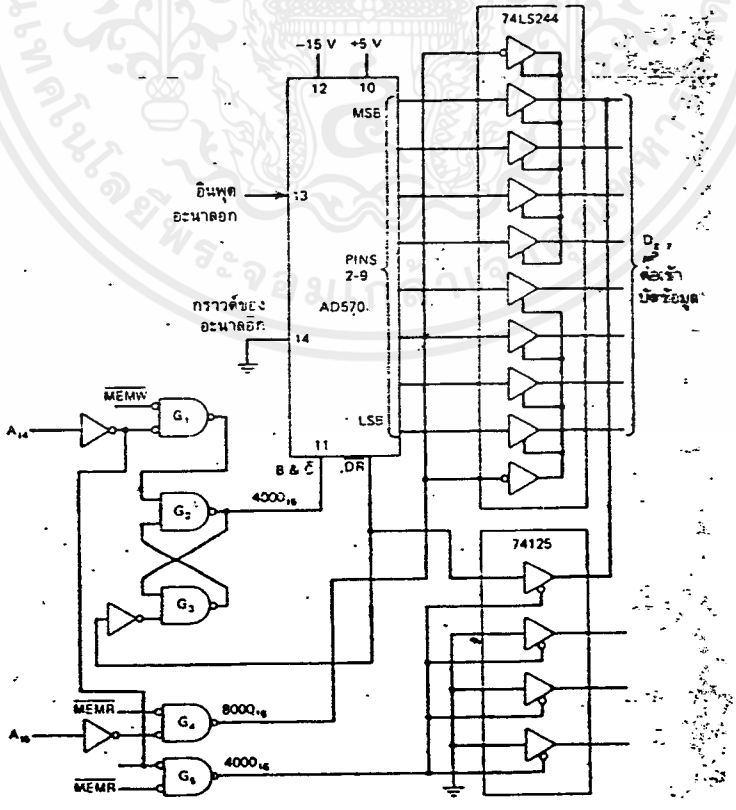
เราจึงมักได้พบเห็นวงจรสุ่มและคงค่าสัญญาณปรากฏคู่กับวงจร เอทูดีอยู่เสมอ

ใช้ไมโครโปรเซสเซอร์ควบคุมวงจรเอทูดีแบบประมาณค่า

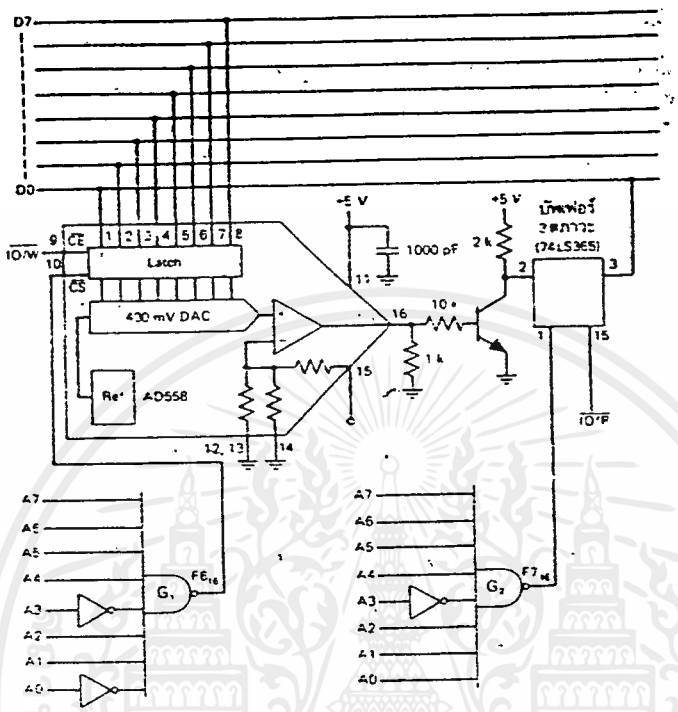
เอทูดีแบบประมาณค่า (successive approximation) ประกอบด้วยดีทูเอรีจิสเตอร์ (SAR) และวงจรเปรียบเทียบ การใช้ไมโครโปรเซสเซอร์ควบคุมเอทูดีทำได้โดยใช้ซอฟต์แวร์ควบ SAR วงจรในรูปที่ 11 แสดงการใช้ 8085 ควบคุมเอทูดีแบบ successive approximation วงจรประกอบด้วยดีทูเอ เบอร์ AD558 บัฟเฟอร์สามสภาวะ สัญญาณควบคุม IO/R และ IO/W และ วงจรถอดรหัส 8 บิต

วงจรนี้ใช้การต่อกับ I/O โดยตรงตัวดีทูเอใช้ตำแหน่งพอร์ต F6H และวงจรเปรียบเทียบใช้ตำแหน่งพอร์ต F7H โดยใช้แนนด์เกต 8 อินพุต 2 ตัว

เมื่อไมโครโปรเซสเซอร์ให้พอร์ต F6H ต่อกับแอดเดรสบัส เอาต์พุตของแนนด์เกตตัวที่ 1 ก็จะมีระดับ "low" เป็นการเลือก chip select (CS) เป็นสัญญาณควบคุม IO/W ทำให้

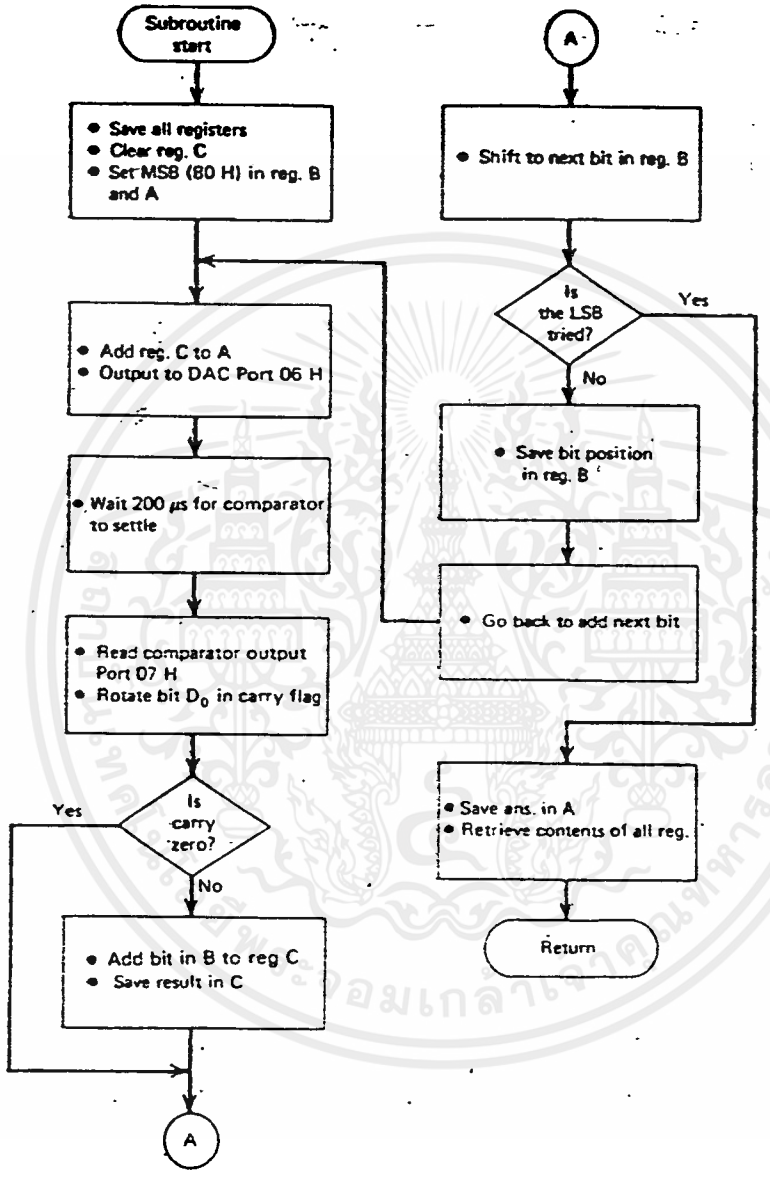


รูปที่ 28 การอินเตอร์เฟสเอทูดีเข้ากับไมโครโปรเซสเซอร์



รูปที่ 29 การใช้ไมโครโปรเซสเซอร์ควบคุมวงจรเอดีซีแบบ successive approximation

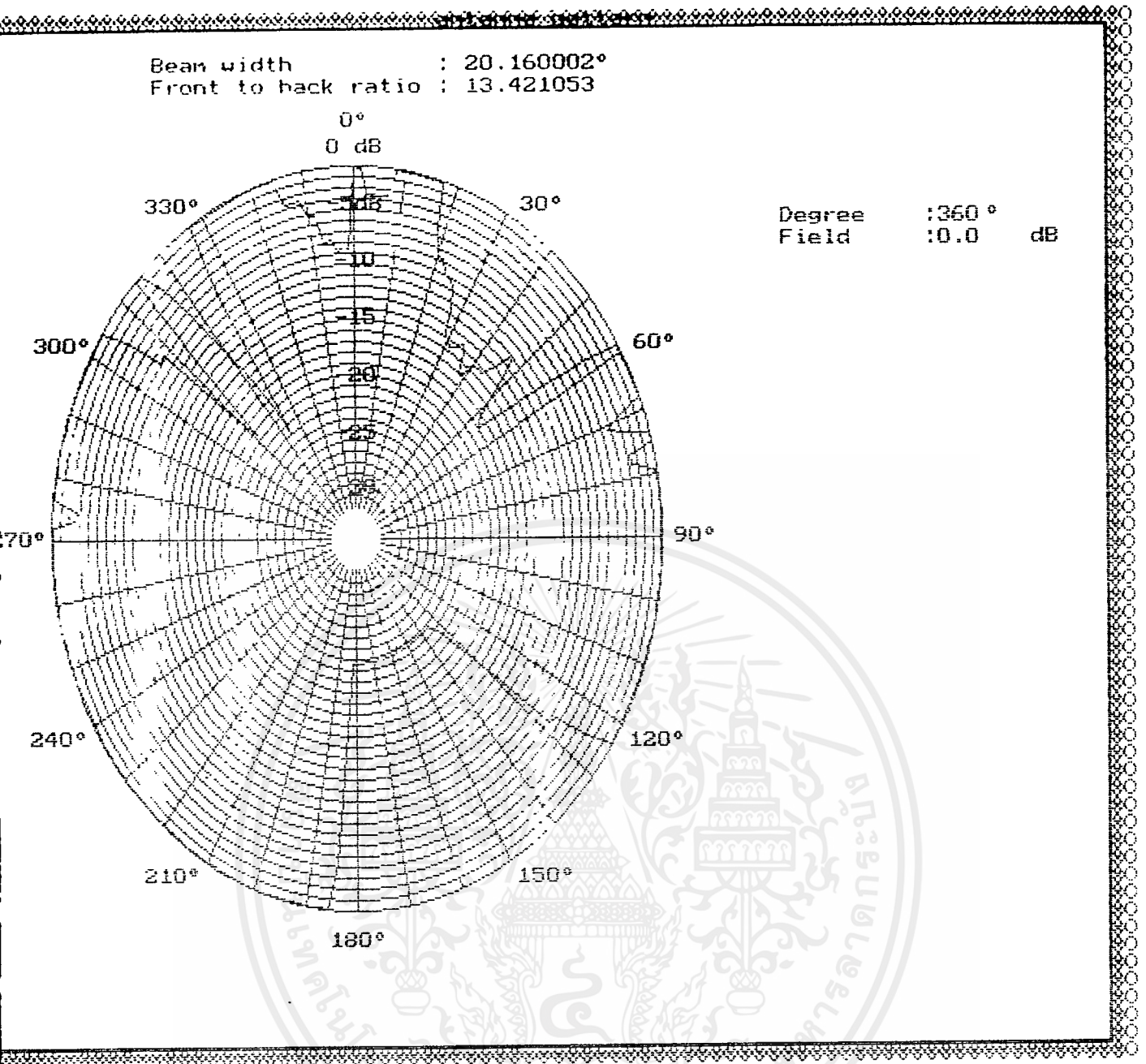
เอดีซีทำงาน เมื่อไมโครโปรเซสเซอร์ส่งตำแหน่ง F7H เข้ากับแอดเดรสบัส แนนด์เกตตัวที่ 2 จะมีเอาต์พุต "low" ทำให้ขา IO/R ทำงาน ทำให้บัฟเฟอร์ 3 สภาวะทำงานอัลกอริทึมของเทคนิคแบบ successiveapproximation นี้ แสดงไว้ในรูปที่ 30



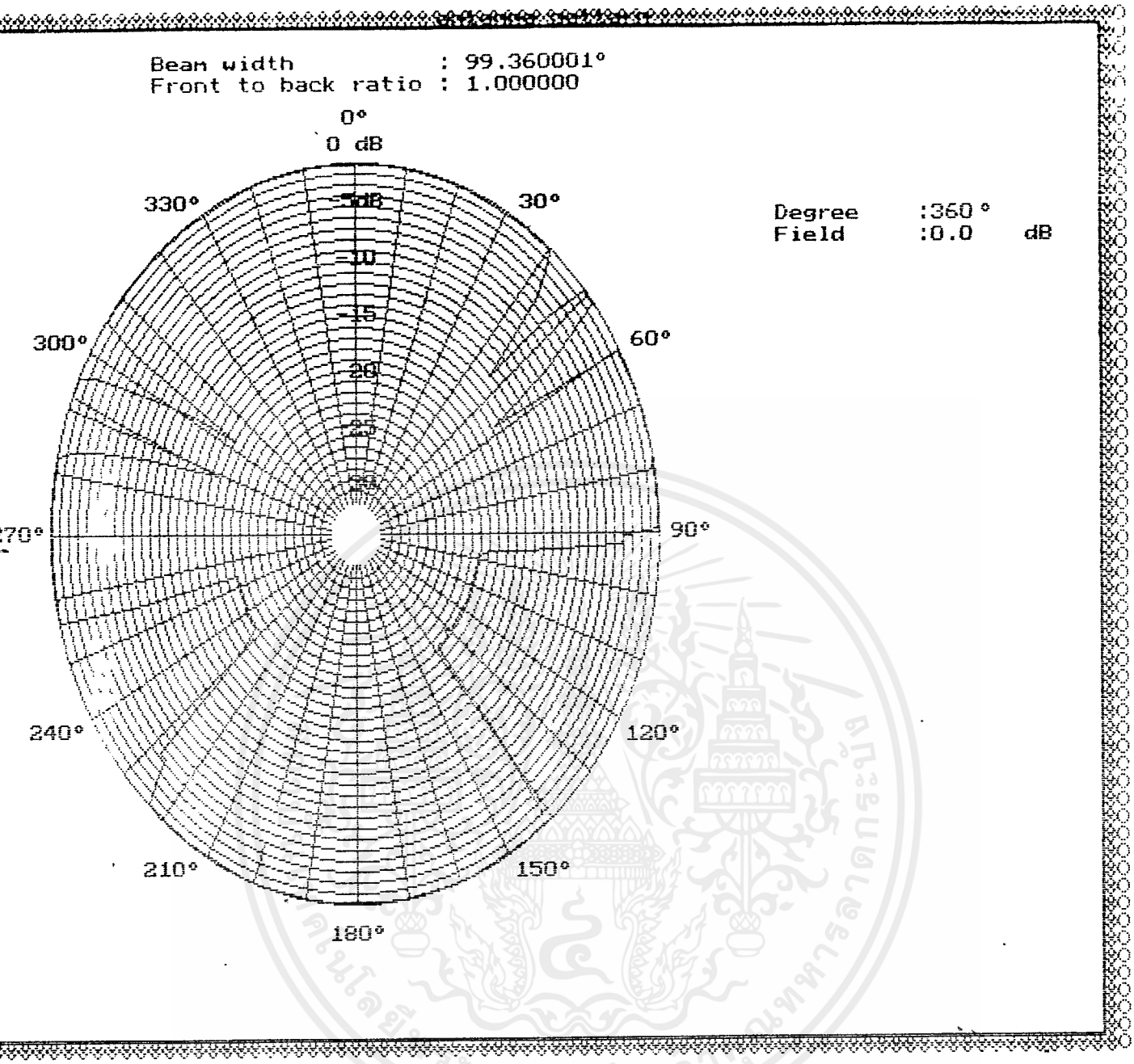
รูปที่ 30 แผนผังการทำงานของเอดีซีแบบ Successive approximation

ผลการทดลอง

ผลที่ได้ใกล้เคียงกับทางทฤษฎี ทั้งนี้เนื่องการสะท้อนของคลื่นภายห้องทำให้ผลที่ออกมาอาจแตกต่างกับทางทฤษฎีไปบ้างเล็กน้อย ในการทดลองที่ใช้นี้โดยการรับเอาความถี่ของโทรทัศน์ช่อง 5,7



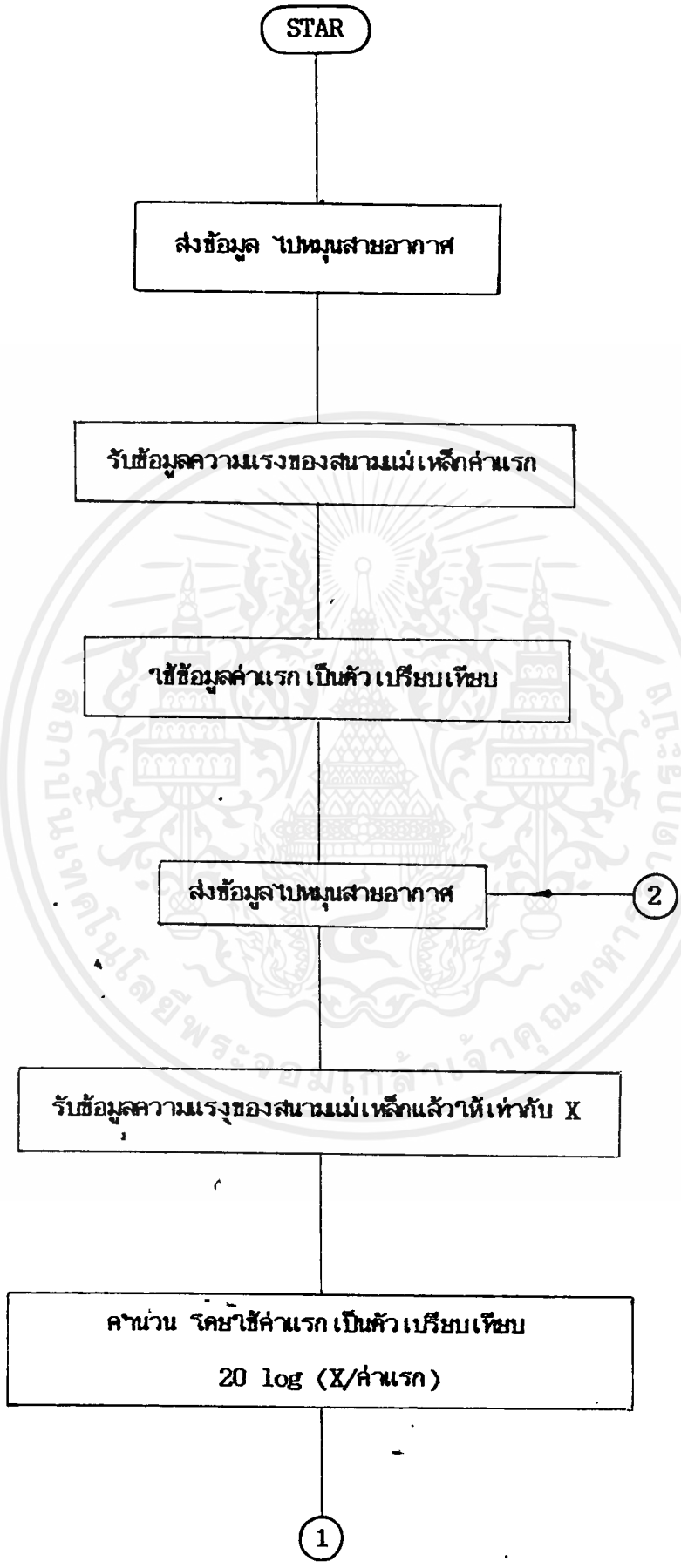
แพทเทิร์นของช่อง 5

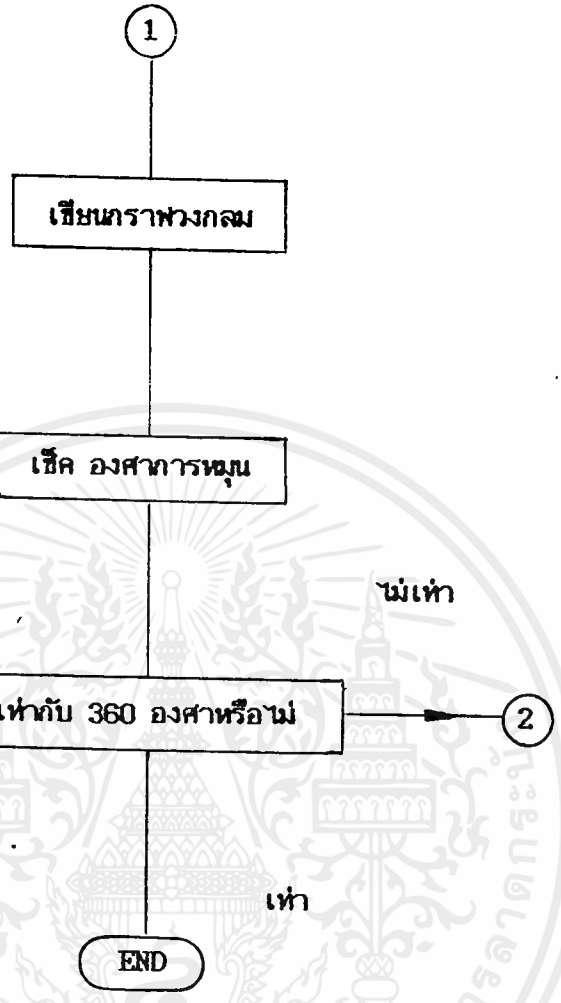


แพทเทิร์นของช่อง 7 ที่ความถี่ 177 MHz

ภาคผนวก

โปรแกรม





รูป แสดง FLOW CHART ของ SOFT WARE

```
typedef struct { int x; int y; char data[13]; int status; } DATA_LIST;
extern DATA_LIST data_list[80];
extern void list_file(int x,int y,int bg,int fg,int line_list,char *list);

#include <graphics.h>
#include <math.h>
#include <stdio.h>
#include <stdarg.h>
#include <dos.h>
#include <conio.h>
#include <stdlib.h>
#include <alloc.h>
#include <process.h>
#include <bios.h>
#include <string.h>
#include <dir.h>

#define  ESC    0x1b
#define  UP     0x48
#define  DOWN   0x50
#define  LEFT   0x4b
#define  RIGHT  0x4d
#define  ENTER  0x0d
#define  S      0x73
#define  P      0x70
#define  L      0x6C
#define  D      0x64
#define  O      0x6F
#define  Q      0x71
#define  P_UP   0x49
#define  P_DOWN 0x51
#define  NUM_MENU 6

void far *sav1;
void far *sav2;
void far *sav3;
int  used_sav1=0,used_sav2=0,used_sav3=0;
int     GraphDriver;/* The Graphics device driver*/
int     GraphMode;/* The Graphics mode value*/
int     MaxX, MaxY;/* The maximum resolution of the screen */
int     color;/* used to set old color when back to main */
float   dB[359];                /* for save data on disks */
float   dBv[359];
char namebuf[21];
```

```
int    fxplot,fyplot;            /*  first position   */
char  *menuarg[6] ={
"Print",
"Save",
"Load",
"Drive",
"Rectang patt",
"Quit",    };


int left;
int top;




void Initialize(void);
void MainWindow(int left,int top,int right,int bottom);
void line_circle(int h,int k,int radiusi,int radiuso);
void arc_circle(int h,int k,int side);
void plot(int degree,float dB);
void far*Saveimage(int left,int top,int right,int bottom);
void stor_win(int left,int top,int right,int bottom);
void restor_win(int left,int top,int status);
int  gprintf( int *xloc, int *yloc, char *fmt, ... );
void dpwork(int left, int top, int right, int bottom);
void txt_menu(int left,int top, int right, int bottom);
void bars(int line_select);
int  shift_bar(int line_select,int key);
int  HotKey(int key1,int line_select);
void antest(void);
void ploting(void);
void openwin(int left,int top,int right,int bottom);
void closewin(int left,int top);
void getname(int numtxt);
void save(void);
void rectang_patt(void);
void load(void);
void dirve(void);
void copydot(int scan);
void copybyte(int byte);
void hardcopy(int left);
void status(int sta);
int  front_to_back(void);
```

```
int  beam_width(void);
void tranfer(void);
void clear(void);
void Report(void);
void ant(void);
void plot_rec(void);
void list(void);
int resp_used(/*int,int,int,int*/void);
int forscan_key(int);

exit_all()
{
   _AX = 0x03;
   geninterrupt(0x10);
   exit(1);
   return;
}

/*----- function main -------*/
main(void)

{
    Initialize();
    cleardevice();
    MainWindow(7,7,MaxX-7,MaxY-7);
    arc_circle(MaxX/3,MaxY/2,170);
    line_circle(MaxX/3,MaxY/2,15,170);
    dpwork(100,100,185,220);
    getch();
    cleardevice();
    closegraph();
    return;

}



void Initialize(void)
{
  GraphDriver = DETECT; /* Request auto-detection*/
  registerbgidriver(EGAVGA_driver);
  initgraph( &GraphDriver, &GraphMode, "" );
  MaxX=getmaxx(); MaxY=getmaxy();
}
```

```
void MainWindow(left,top,right,bottom)
int left,top,right,bottom;
{
   char patterns_old[8][8];
   char patterns[][8] = {
    { 0x81, 0x00, 0x10, 0x38, 0x7c, 0x38, 0x10, 0x81 }
  };
      color=getcolor();

      setcolor(LIGHTGRAY);
rectangle(left,top,right,bottom);

      setcolor(DARKGRAY);
rectangle(left+1,top+1,right-1,bottom-1);
      getfillpattern(&patterns_old[0][0]);
      setfillpattern( &patterns[0][0], LIGHTBLUE);
bar(0,0,right+1,top-1);
bar(right+1,0,MaxX,MaxY);
bar(left-1,bottom+1,right+1,MaxY);
bar(0,top-1,left-1,MaxY);

      setcolor(YELLOW);
      settextjustify(CENTER_TEXT,CENTER_TEXT);
quttextxy(MaxX/2,textheight("a")/2,"antenna pattern");

      window(left,top,right,bottom);
      MaxX = getmaxx();
      MaxY = getmaxy();/* Read size of screen*/
      setfillpattern(&patterns_old[0][0],color);
      setcolor(color);
  }


void arc_circle(h,k,side)
int h,k,side;

{
int count=5,radius;
int color;
color = getcolor();
    for(radius=20;radius<=side;radius+=5)
```

```
{
if(count==5)
{
 count=0;
 setcolor(LIGHTGRAY);
 circle(h,k,radius);
 setcolor(DARKGRAY);
} else circle(h,k,radius);
count++;
} setcolor(color);

 }

void line_circle(h,k,radiusi,radiuso)
#define PINE 3.141592654
int h,k,radiusi,radiuso;
{
   int color;
   int count;
   int hor,ver,hor1,ver1;
   int angle,yo,xo,yi,xi;
   char *textdB[7]={"-30","-25","-20","-15","-10","-5dB","0 dB"};
   char *degree[12]={"0ø"," 30ø"," 60ø"," 90ø","120ø","150ø","180ø","210ø","240ø"
   float zeata;

   color = getcolor();
   for(angle=0;angle<=359;angle=angle+=10)
   {
      zeata=PINE*angle/180;
      ver=radiuso*sin(zeata);
      hor=radiuso*cos(zeata);
      ver1=radiusi*sin(zeata);
      hor1=radiusi*cos(zeata);

      yo= k-hor;     xo= h+ver;
      yi= k-hor1;    xi= h+ver1;

      setcolor(LIGHTGRAY);
      line(xo,yo,xi,yi);

      if(angle==0) {
 setcolor(LIGHTGRAY);
 settextjustify(CENTER_TEXT,CENTER_TEXT);
 for(count=0; count<7; count++)
```

```
    outtextxy(xo,yi-(count)*(yi-yo)/6-8,textdB[count]);
  setcolor(LIGHTGRAY);
      }


      if(((angle%30) == 0)&&(angle!=359)) {
switch(angle) {
   case   0: settextjustify(CENTER_TEXT,CENTER_TEXT); yo-=20; break;
   case  30: settextjustify(LEFT_TEXT,BOTTOM_TEXT); break;
   case  90: settextjustify(LEFT_TEXT,CENTER_TEXT); break;
   case 120: settextjustify(LEFT_TEXT,TOP_TEXT);yo+=4;xo+=4; break;
   case 150: settextjustify(LEFT_TEXT,TOP_TEXT);yo+=4;xo+=4; break;
   case 180: settextjustify(CENTER_TEXT,CENTER_TEXT);yo+=15; break;
   case 210: settextjustify(RIGHT_TEXT,TOP_TEXT);yo+=3;xo-=3; break;
   case 240: settextjustify(RIGHT_TEXT,TOP_TEXT);yo+=3;xo-=3; break;
   case 270: settextjustify(RIGHT_TEXT,CENTER_TEXT);xo-=3;  break;
   case 300: settextjustify(RIGHT_TEXT,BOTTOM_TEXT); break;
}
setcolor(LIGHTCYAN);
outtextxy(xo,yo,degree[angle/30]);
setcolor(LIGHTGRAY);
      }
      setcolor(color);
   }
}


/**/
/*GPRINTF: Used like PRINTF except the output is sent to the*/
/*screen in graphics mode at the specified co-ordinate.*/
/**/

int gprintfs( int *xloc, int *yloc, char *fmt, ... )
{
  va_list  argptr;/* Argument list pointer*/
  char str[140];/* Buffer to build sting into*/
  int cnt;/* Result of SPRINTF for return */

  va_start( argptr,fmt );/* Initialize va_ functions*/

  cnt = vsprintf( str, fmt, argptr );/* prints string to buffer*/
  outtextxy( *xloc, *yloc, str );/* Send string in graphics mode */
/* *yloc += textheight( "H" ) + 2;       /* Advance to next line          */
```

```
   va_end( argptr );/* Close va_ functions*/

   return( cnt );/* Return the conversion count*/

}

int gprintf( int *xloc, int *yloc, char *fmt, ... )
{
   va_list  argptr;/* Argument list pointer*/
   char str[140];/* Buffer to build sting into*/
   int cnt;/* Result of SPRINTF for return */

   va_start( argptr, fmt );/* Initialize va_ functions*/

   cnt = vsprintf( str, fmt, argptr );/* prints string to buffer*/
   outtextxy( *xloc, *yloc, str );/* Send string in graphics mode */
  *yloc += textheight( "H" ) + 2;        /* Advance to next line          */

   va_end( argptr );/* Close va_ functions*/

   return( cnt );/* Return the conversion count*/

}


/*------- plot value of dB to pattrun ------*/
/* this program call just check degree 0 if degree not 0 don't call*/
/*  and this program just call after program inport data of dB once check 360 rou


void plot(int degree,float dBin)
#define PINE  3.141592554

{
int h=MaxX/3;int k=MaxY/2;            /* center of circular for chart */
int xplot,yplot;
int ver,hor;       /* distance from center */
float zeata;       /* angle in radius */
char *buffer;

buffer=farmalloc(20);
if(!buffer) exit_all(); else;
color=getcolor();
```

```
     if(degree==0)
       {
zeata=PINE*degree/180;
ver=(170+dBin*5)*sin(zeata);   /* 170 is value max radius to call  void line_circ
hor=(170+dBin*5)*cos(zeata);   /* max 34 dB */
fxplot=h+ver;
fyplot=k-hor;
 }
      else
{
                  zeata=PINE*degree/180;
 ver=(170+dBin*5)*sin(zeata);   /* 170 is value max radius to call  void line_cir
 hor=(170+dBin*5)*cos(zeata);   /* max 34 dB */
 xplot=h+ver;
 yplot=k-hor;
 setcolor(LIGHTCYAN);
 sound(7000);
 line (xplot,yplot,fxplot,fyplot);
 nosound();

/* firt position */

 fxplot=xplot;
 fyplot=yplot;
  }

/* set win  off text */

     setcolor(BROWN);

     setviewport(538,90,578,108,1);
     clearviewport();
     setviewport(0,0,MaxX,MaxY,1);

/* display value */
     settextjustify(LEFT_TEXT,TOP_TEXT);
     sprintf(buffer,"Degree    :%d",degree);
     outtextxy(450,90,buffer);
     sprintf(buffer,"ω");
     outtextxy(565,90,buffer);
     sprintf(buffer,"Field     :%.1f",dBin);
     outtextxy(450,100,buffer);
     sprintf(buffer,"dB");
     outtextxy(590,100,buffer);
```

```
        setcolor(color);
        dB[degree]=dBin;

         }

void far *Saveimage(int left,int top,int right,int bottom)
{
    void far *image;
    long size;
    size = imagesize(left,top,right,bottom);
    image=farmalloc(size);
    if(!image) exit_all(); else;
    getimage(left,top,right,bottom,image);
    putimage(left,top,image,XOR_PUT);
    return(image);
 }

void dpwork(int left, int top, int right, int bottom)

{
  void far *image_buffer;
  int line_select;
  char  patterns[][8]={ 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff};
  color=getcolor();
  image_buffer=Saveimage(left,top,right,bottom);
  setcolor(DARKGRAY);
  rectangle(left+1,top+1,right-1,bottom-1);
  setcolor(DARKGRAY);
  rectangle(left+2,top+2,right-2,bottom-2);
  setcolor(LIGHTGRAY);
  rectangle(left+3,top+3,right-3,bottom-3);
  setcolor(LIGHTGRAY);
  rectangle(left+4,top+4,right-4,bottom-4);
  setfillpattern(&patterns[0][0],YELLOW);
  floodfill(120,120,LIGHTGRAY);
  txt_menu(left+4,top+4,right-4,bottom-4);
  line_select=resp_used(/*left,top,right,line_height*/);
  putimage(left,top,image_buffer,COPY_PUT);
  farfree(image_buffer);
  switch(line_select)
{
 case 0:antest(); break;
 case 1:hardcopy(1); break;
 case 2:save();     break;
```

```
  case 3:load();      break;
  case 4:dirve();     break;
  case 5:rectang_patt();   break;
  case 6:closegraph();  exit(0);
 }

   antest();
 }

 void txt_menu(int left, int top, int right, int bottom )


 {
 int line_height;int height_H;        /* height of char */
 char buffer[20];
 int line=0;
 int color;

 color=getcolor();
 height_H=textheight("H");
 line_height=height_H;
 settextjustify(LEFT_TEXT,TOP_TEXT);

 setviewport( left, top, right, bottom,1);
 settextstyle(DEFAULT_FONT, HORIZ_DIR, 1);
         setcolor(RED);
 for (line=0;line<=5;++line)
 {
 sprintf(buffer,menuarg[line]);
 outtextxy(height_H,line_height,buffer);
 rectangle(height_H-3,line_height-2,72,line_height+height_H);
 line_height=line_height+height_H*2;
 }
 setviewport(0,0,MaxX,MaxY,1);
 setcolor(color);

 }


 void bars(int line_select)     /* change size and position in function  */

  {
    int left =   110;
    int top  =   95;
```

```
    int right=   175;

    int height_H;
    color=getcolor();
    height_H=textheight("H");
    setfillstyle(SOLID_FILL,DARKGRAY);
    if(used_sav2) farfree(sav2); else;
    sav2=Saveimage(left, top+height_H*line_select*2, right,
     top+2*height_H*line_select+height_H);
    used_sav2 = 1;
    setcolor(RED);

    putimage(left, top+height_H*line_select*2,sav2,NOT_PUT);
    setcolor(color);
   }


int shift_bar(int line_select,int key)
{
 if(key==UP || key==DOWN)
       {
 if(key==UP){

if(line_select==1)line_select=6;
else line_select--;
}
 else
 {
  if(key==DOWN){
       if(line_select==6)line_select=1;
       else line_select++;
       }

 }

      }
           return(line_select);
}


char key1;
int resp_used(void)

{
```

```
int line_select=1;
key1=0;                    /* clear key1 */
bars(line_select);
      do
       {
if(key1==ENTER) return(line_select);
key1=getch();
switch(key1){
     case ENTER:bars(line_select); return(line_select);
     case 0    :line_select=forscan_key(line_select); break;
    }
if((key1==P)||(key1==S)||(key1==L)||(key1==D)||(key1==0)||(key1==Q))
   {
   line_select=HotKey(key1,line_select);
   return(line_select);
   }

}

  while(key1!=ESC);

  return(0);
}


int HotKey(int key1,int line_select)
    {
      switch(key1){
 case P:line_select= 1; break;
 case S:line_select= 2; break;
 case L:line_select= 3; break;
 case D:line_select= 4; break;
 case 0:line_select= 5; break;
 case Q:line_select= 6; break;
      }
      return(line_select);
     }

int forscan_key(line_select)

{
int key2;
int height_H;
```

```
height_H=textheight("H");

   do{
 key2=getch();
 if(!(key2==UP || key2==DOWN)) return(line_select);
 line_select=shift_bar(line_select,key2);
 if (key2==DOWN)
     {
      if(line_select==1)

  putimage(110,79+height_H*(line_select+6)*2,sav2,COPY_PUT);
      else
  putimage(110,79+height_H*line_select*2,sav2,COPY_PUT);

       }
 else
      {
       if(line_select==6)
  putimage(110,79+height_H*(line_select-4)*2,sav2,COPY_PUT);
      else
  putimage(110,79+height_H*(line_select+2)*2,sav2,COPY_PUT);
 }

 bars(line_select);
 do {
    key1=getch();
    if(key1==ENTER) return(line_select);
    if((key1==P)||(key1==S)||(key1==L)||(key1==D)||(key1==O)||(key1==Q))
      {
      line_select=HotKey(key1,line_select);
      return(line_select);
      }
    }
 while((key1!=0)&&(key1!=ESC));
      }
   while(key1!=ESC);
   return(0);
   }


char *buff;
void  antest(void)

   {
```

```
    void far *image_buff;
    int left=230,top=455,right=390,bottom=475;
    int color;
    char test;
    /* clear all sav1..3 */
    if(used_sav1) farfree(sav1); else;
    if(used_sav2) farfree(sav2); else;
    if(used_sav3) farfree(sav3); else;
    used_sav1 = 0;
    used_sav2 = 0;
    used_sav3 = 0;
    buff=farmalloc(10);
    if(!buff) exit_all(); else;
    image_buff=Saveimage(left-3,top-3,right+3,bottom+3);
    color=getcolor();
    setcolor(LIGHTGRAY);
    rectangle(left,top,right,bottom);
    setcolor(WHITE);
    rectangle(left-1,top-1,right+1,bottom+1);
    setcolor(LIGHTGRAY);
    rectangle(left-2,top-2,right+2,bottom+2);
    setcolor(DARKGRAY);
    rectangle(left-3,top-3,right+3,bottom+3);
    setfillstyle(SOLID_FILL,DARKGRAY);
    floodfill(left+5,top+5,WHITE);
    sprintf(buff,"TEST Y or N");
    settextjustify(CENTER_TEXT,CENTER_TEXT);
    setcolor(RED);
    outtextxy(left+(right-left)/2,top+(bottom-top)/2,buff);
    do
      {
      test=getch();
      if((test=='Y')||(test=='y'))
{
clear();
ploting();
        dpwork(100,100,185,220);
}
      if((test=='N')||(test=='n')) break;
      }
    while(test!=ESC);
    putimage(left-3,top-3,image_buff,COPY_PUT);
    setviewport(0,0,MaxX,MaxY,1);
    setcolor(color);
```

```
      farfree(image_buff);
      dpwork(100.100,185,220);
      }


void clear(void
     {
     cleardevice();
     MainWindow(7,7,MaxX-7,MaxY-7);
     arc_circle(MaxX/3,MaxY/2,170);
     line_circle(MaxX/3,MaxY/2,15,170);
     }


void ploting(void)
     {
     int count;
     /*                  */
     /* insert program trun mortor and input data voltage */
     /*                                                   */
      tranfer();
      for(count=0;count<=359;count++)
        {
plot(count,dB[count]);
}
Report();
     }

char *buf;
int beam;
int FtoB;
void Report(void)
{
  int color;
  color=getcolor();
  settextjustify( LEFT_TEXT, TOP_TEXT );
  sprintf(buf,"Beam width          : %dº",beam);
  setcolor(BROWN);
  outtextxy(100,20,buf);
  sprintf(buf,"Front to back ratio : %d",FtoB);
  outtextxy(100,30,buf);
  setcolor(color);
}
```

```
int beam_width(void)
   {
   int count;
   int seta1,seta2;
   int beam;

   for(count=0;count<=359;count++)
      {
      if(dB[count]<-3)
{
seta1=count;
break;
}
      }
   for(count=359;count>=0;count--)
      {
      if(dB[count]<-3)
{
seta2=count;
break;
}
      }
   beam=seta1+(360-seta2);
   return(beam);
   }
int front_to_back (void)
  {
  int FtoB;
  FtoB=dBv[0]/dBv[180];
  return(FtoB);
  }

void openwin(left1,top1,right1,bottom1)
{
  int color;
  if(used_sav1) farfree(sav1); else;
  sav1=Saveimage(left1,top1,right1,bottom1);
  used_sav1=1;
  setviewport(left1,top1,right1,bottom1,1);
  clearviewport();
  setviewport(0,0,MaxX,MaxY,1);
  rectangle(left1+1,top1+1,right1-1,bottom1-1);
  color=getcolor();
  floodfill(left1+2,top1+3,color);
```

```
 }

void closewin(left1,top1)
{
  putimage(left1,top1,sav1,COPY_PUT);
}

void save(void)
   {
      FILE *fp;
      int count;
      getname(1);
      for(count=0;count<20;count++){ if(namebuf[count]==ESC) return; }
      if((fp=fopen(namebuf,"wb"))==NULL)
    {
    getname(4);
    return;
     }

if(360==fwrite(&dBv[0],sizeof(float),360,fp))
 {
 status(0);
 }

     fclose(fp);
    }

void load(void)
   {
      FILE *fp;
      int count,i=0;
      char *test;
      char data_cmp[4];
      getname(0);

      if(namebuf[0]==NULL) return;
      for(count=0;count<20;count++){ if(namebuf[count]==ESC) return; }
      for(count=0;count<20;count++)  if(namebuf[count]=='.') break; else;
      if(count==20){
  getname(4);
  return; }else;
      for(count=count++;count<20;count++) {
 data_cmp[i]=toupper(namebuf[count]);
 i++;
```

```
   if(i==3) break; else;
        }
        data_cmp[3] = NULL;
        if(strcmp(data_cmp,"ATN")){getname(4); return; }else;

        if((fp=fopen(namebuf,"rb"))==NULL)
      {
      getname(4);
       return;
       }
        if(360==fread(&dBv[0],sizeof(float),360,fp))
 {
 status(0);
 }
       fclose(fp);
       tranfer();

      clear();
      ploting();

      }

void tranfer(void)
      {
      int count;

      if(dBv[0]==0)
  {
  closegraph();
  printf("Firth voltage  at 0@ is zero");
  exit(0);
  }
    for(count=0;count<=359;count++)
{
if(dBv[count]==0)
  {
  status(1);
  }
dB[count]=(20*log10(dBv[count]/dBv[0]));
if(dB[count]<-30) dB[count]=-30;
}
      beam=beam_width();
      FtoB=front_to_back();
      }
```

```
void status(sta)
{
int sizeln=0;
char *text[2]={
"!! SUCCESSFUL !!",
"!! NO LOAD DATA OR DATA ERROR !!",
};

 if(used_sav2) farfree(sav2); else;
 sav2=Saveimage(100,50,360,70);
 used_sav2 = 1;
 color=getcolor();
 setcolor(WHITE);
 rectangle(101,51,359,69);
 setfillstyle(SOLID_FILL,DARKGRAY);
 floodfill(102,52,WHITE);
 setviewport(100,50,360,70,1);
 settextjustify(LEFT_TEXT,TOP_TEXT);
 sizeln=strlen(text[sta])*8;
 outtextxy((359-101-sizeln)/2,7,text[sta]);
 settextjustify(LEFT_TEXT,CENTER_TEXT);
 setviewport(0,0,MaxX,MaxY,1);
 delay(1000);
 putimage(100,50,sav2,COPY_PUT);
 setcolor(color);

    }
char *text[6]={
"Load File :",
"Save File :",
"Change Drive to:",
"Quit Y or N :",
"Can't Done ! ",
};

void getname(numtxt)
 {
 int posX,count,sizel;
 int i;
 int left1=100,top1=50,right1=360,bottom1=70;
 struct viewporttype vp;

 settextstyle(DEFAULT_FONT,HORIZ_DIR,1);  /* character size 8-8 */
 settextjustify(LEFT_TEXT,TOP_TEXT);
```

```
 sprintf(namebuf,text[numtxt]);     /* 17 character */
 size1 = strlen(text[numtxt]);
 getviewsettings(&vp);
 openwin(left1,top1,right1,bottom1);
 outtextxy(left1+5,top1+7,namebuf);
 posX=left1+8*size1+5;
 for(count=0;count<20;count++) namebuf[count]=0x00; /* clear buffer */
 setviewport(vp.left,vp.top,vp.right,vp.bottom,vp.clip);
 for(count=0;count<20;count++)
{
if(numtxt==4) break;
namebuf[count]=getch();
do{
  switch(namebuf[count]){
     case 0x09 : namebuf[count]=getch();  break;          /* key TAB */
     case 0x1b : closewin(left1,top1); return;           /* key ESC */
     case 0x00 : getch(); namebuf[count]=getch(); break; /* Scan code */
     }
   }
while(namebuf[count]==0x00 || namebuf[count]==0x1b || namebuf[count]==0x09);
if (namebuf[count]==ENTER) break;
if(namebuf[count]==0x08 || count>20)  /* key Back Space */
   {
      namebuf[count]=0;
      if(count>0)
      {count-=2;}
      if(count<=0)
       {
       for(i=0;i<20;i++) namebuf[i]=0;
       count=-1;
       }
   posX=posX-8;
   if(posX < left1+8*size1+5) posX= left1+8*size1+5;
   setviewport(posX,top1+7,posX+8,top1+7+8,1);
   clearviewport();
   setviewport(vp.left,vp.top,vp.right,vp.bottom,vp.clip);
   color=getcolor();
   floodfill(posX+1,top1+8,color);

   }
 else
    {
      outtextxy(posX,top1+7,&namebuf[count]);
      posX=posX+8;
```

```
      }
  if(numtxt==2)
  {
  if(!((65<= (int)namebuf[count] &&(int)namebuf[count]<=72) ||
       (97<= (int)namebuf[count] && (int)namebuf[count]<=104 )))
       {
       closewin(left1,top1);
       getname(2);
       }
  count=20;
  }
      }
      if(numtxt==4) delay(1000);

      if(count<=0 && numtxt!=4)
{
closewin(left1,top1);
getname(numtxt); /* count==0 begin again */
}
      while(namebuf[count]=='\r')
       {
       if(numtxt==4) break;
       namebuf[count]=0;
       }
      list();
      closewin(left1,top1);

 }

int  pointx,pointy,pointx1=0,pointy1=0;

void list(void)
{
   void far  *image_buffer1;
   void far  *image_buffer2;
   void far  *image_buffer3;
   DATA_LIST old_data;
   int x=0,y=0,line=0,count,line1=0;
   int key_push,key_push2;
   int end = 0;
   char nbuffer[20];
   for(count=0;count<20;count++) nbuffer[count]=namebuf[count];
   for(count=0;count<20;count++)  /* test namebuf */
      if((namebuf[count]=='*')||(namebuf[count]=='?')) break; else;
```

```
    if(count==20) return; else;
    setviewport(0,0,MaxX,MaxY,1);
    image_buffer1=Saveimage(100,75,293,319);
    image_buffer2=Saveimage(292,75,488,319);
    setcolor(LIGHTGRAY);
    rectangle(100,75,488,319);
    list_file(102,77,DARKGRAY,WHITE,line,namebuf);
    if(data_list[0].status==0) {  /* exit when no file dir. */
       end = 1; namebuf[0] = NULL;
    } else;
    old_data = data_list[(y*4)+x];
    setcolor(RED);
    setviewport(102,77,102+384,(77+240),1);
    image_buffer3 = \
       Saveimage(old_data.x,old_data.y,old_data.x+95,old_data.y+7);
    putimage(old_data.x,old_data.y,image_buffer3,NOT_PUT);
    strcpy(namebuf,old_data.data);
    for(;!end;) {
       key_push = getch();
       if(key_push == ENTER) { end = 1; continue; } else;
       if(key_push == ESC) { end = 1; namebuf[0] = NULL; } else;
       if(key_push==0) {
 key_push2=getch();
 if(data_list[79].status && (key_push2==P_DOWN)) line+=19;
 if(key_push2==P_UP)
 line-=19;
 switch(key_push2) {
    case LEFT : x--; if(x<0) { x=0;
} else;
if((!data_list[(y*4)+x].status)&&(x!=0)) x++; else;
break;
    case RIGHT: x++; if(x>3) x=3; else;
if((!data_list[(y*4)+x].status)&&(x!=0)) x--; else;
                        break;
    case UP   : y--; if(y<0)  y=0;  else;
if((!data_list[(y*4)+x].status)&&(y!=0)) y++; else;
break;
    case DOWN : y++; if(y>=20)  y=19; else;
if((!data_list[(y*4)+x].status)&&(y!=0)) y--; else;
break;
   case P_DOWN: if(data_list[79].status){
    list_file(102,77,DARKGRAY,WHITE,line,nbuffer);
    y=0; x=0; farfree(image_buffer3);
    setviewport(102,77,102+384,(77+240),1);
```

```
      image_buffer3=Saveimage(old_data.x,old_data.y\
      ,old_data.x+95,old_data.y+7);
      putimage(old_data.x,old_data.y,image_buffer3\
      ,NOT_PUT);} else; break;
     caseP_UP  :if(line>=0 && line1!=line){
      list_file(102,77,DARKGRAY,WHITE,line,nbuffer);
      y=0; x=0; farfree(image_buffer3);
      setviewport(102,77,102+384,(77+240),1);
      image_buffer3=Saveimage(old_data.x,old_data.y\
      ,old_data.x+95,old_data.y+7);
      putimage(old_data.x,old_data.y,image_buffer3\
      ,NOT_PUT);} else line1=line;line=0;break;
      default   : break;
  }
  setviewport(102,77,102+384,(77+240),1);
  if(old_data.status)
     putimage(old_data.x,old_data.y,image_buffer3,COPY_PUT);
  else;
  old_data = data_list[(y*4)+x];
  farfree(image_buffer3);
  image_buffer3 = \
     image_buffer3 = Saveimage(old_data.x,old_data.y,old_data.x+95,old_data.y+7);
  putimage(old_data.x,old_data.y,image_buffer3,NOT_PUT);
  strcpy(namebuf,old_data.data);
       } else;
    }
    farfree(image_buffer3);
    setviewport(0,0,MaxX,MaxY,1);
    putimage(292,75,image_buffer2,COPY_PUT);
    putimage(100,75,image_buffer1,COPY_PUT);
    farfree(image_buffer1);
    farfree(image_buffer2);
    return;
}

void rectang_patt(void)
{
  ant();
  plot_rec();
}

void dirve()
     {
     int dir;
```

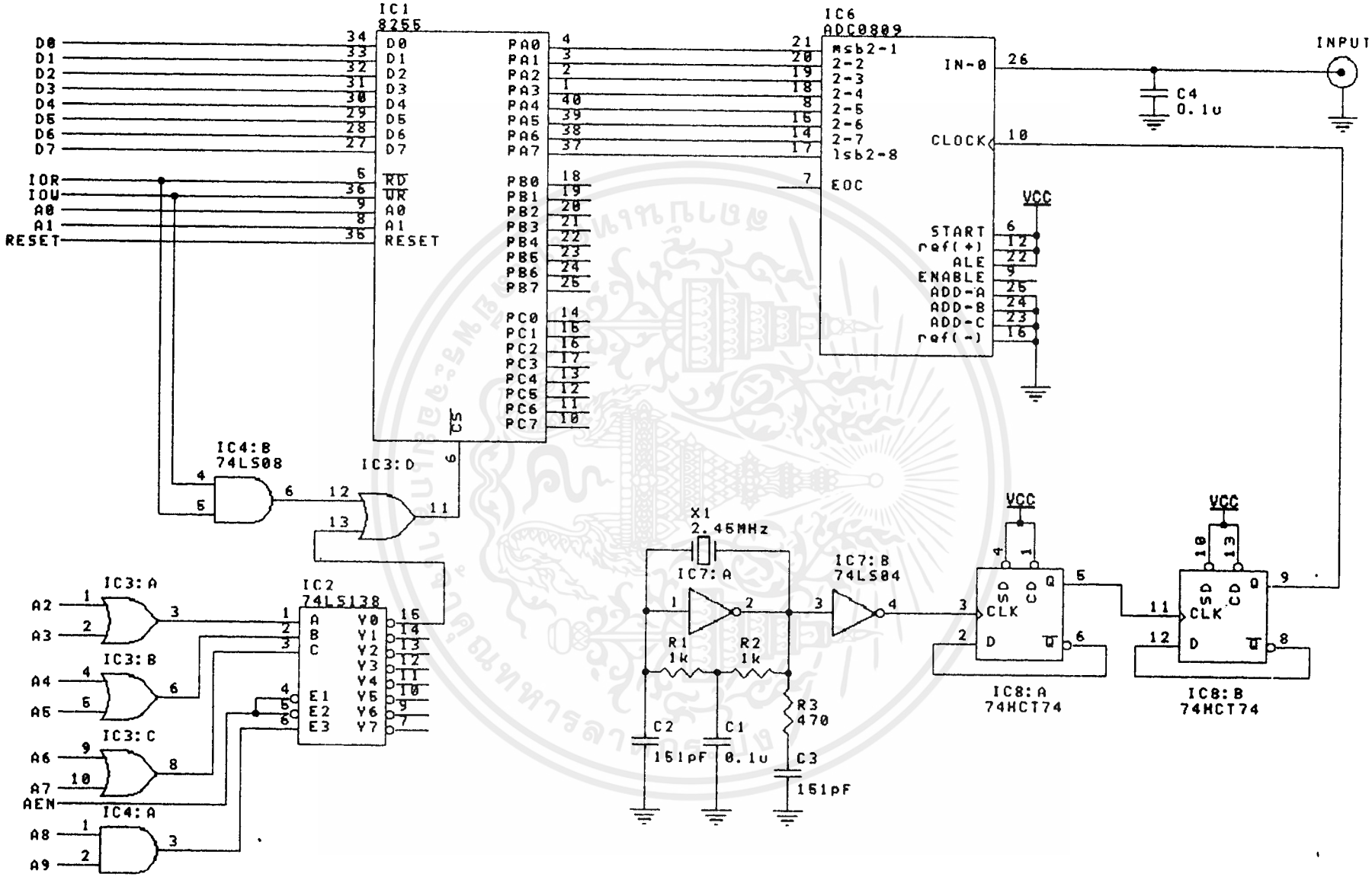
```
    char buf;
    char name[20];

    dir=getdisk();
    switch(dir)
  {
  case 0:buf='A'; break;
  case 1:buf='B'; break;
  case 2:buf='C'; break;
  case 3:buf='D'; break;
  case 4:buf='E'; break;
  case 5:buf='F'; break;
  case 6:buf='G'; break;
  case 7:buf='H'; break;
  }
    if(used_sav2) farfree(sav2); else;
    sav2=Saveimage(0,0,140,10);
    used_sav2=1;
    setviewport(0,0,140,10,1);
    sprintf(name,"Current Dir :%s",&buf);
    outtext(name);
    setviewport(0,0,MaxX,MaxY,1);
    getname(2);
    putimage(0,0,sav2,COPY_PUT);
    switch(*namebuf){
 case 'a':dir=0; break;
 case 'A':dir=0; break;
 case 'b':dir=1; break;
 case 'B':dir=1; break;
 case 'c':dir=2; break;
 case 'C':dir=2; break;
 case 'd':dir=3; break;
 case 'D':dir=3; break;
 case 'e':dir=4; break;
 case 'E':dir=4; break;
 case 'f':dir=5; break;
 case 'F':dir=5; break;
 case 'g':dir=6; break;
 case 'G':dir=6; break;
 case 'h':dir=7; break;
 case 'H':dir=7; break;
 }
   setdisk(dir);
   }
```

```
void copybyte(int byte)
 {
   biosprint(0,byte,0);
 }

void copydot(int scan)
 {
  int bits[8] = {128,64,32,16,8,4,2,1};
  int bit,printbyte,maxx,xpix,n1,n2,exit;

    maxx = getmaxx()+1;
    do
      {
 printbyte=0; maxx=maxx--; exit=27;
for (bit=0;bit<=7;bit++)
 {
  if (getpixel(maxx,scan+bit)!=0)
   { printbyte=printbyte+bits[bit]; }
 }
  exit=printbyte;
  if (maxx==0) { exit=27;}
      } while (exit==0);

     if (printbyte!=0)
      { maxx=maxx++;
n2=maxx/256; n1=maxx-n2*256;
copybyte(27);   copybyte(51);   copybyte(24);
copybyte(27);   copybyte(42);   copybyte(6);
copybyte(n1);   copybyte(n2);
for (xpix=0;xpix<=maxx;xpix++)
   {  printbyte=0;
      for (bit=0;bit<=7;bit++)
       {
if (getpixel(xpix,scan+bit)!=0)
   { printbyte=printbyte+bits[bit]; }
       }
      copybyte(printbyte);
   }
      }
 }

void hardcopy(int left)
 {
```

```
  int scanline;
  copybyte(27); copybyte(108); copybyte(left);
  scanline=0;
    do
      {
       copydot(scanline); scanline=scanline+8;
       copybyte(10);
      }
    while (scanline <(getmaxy()+1));
    copybyte(27); copybyte(2);
}

char *buf;
void ant(void)
   {
    int orX,orY;
    int count;
    float i;

    orX=MaxX/2; orY=MaxY/2+75;
    cleardevice();
    line((MaxX-361)/2,MaxY-(MaxY-150)/2,MaxX-(MaxX-361)/2,MaxY-(MaxY-150)/2);
    line(MaxX/2,(MaxY-150)/2,MaxX/2,MaxY-(MaxY-150)/2);
    settextjustify(CENTER_TEXT,TOP_TEXT);
    outtextxy(orX,orY+7,"0@");
    settextjustify(CENTER_TEXT,BOTTOM_TEXT);
    outtextxy(orX,orY-160,"dB");
    settextjustify(LEFT_TEXT,TOP_TEXT);
    for(count=0;count<=180;count+=90)
       {
       if(count)
{
line(orX+count,MaxY-(MaxY-150)/2,orX+count,MaxY-(MaxY-150)/2+5);
sprintf(buf,"%d@",count);
outtextxy(orX+count,MaxY-(MaxY-150)/2+7,buf);
}
}
    settextjustify(RIGHT_TEXT,TOP_TEXT);
    for(count=0;count<=180;count+=90)
       {
       if(count)
{
line(orX-count,MaxY-(MaxY-150)/2,orX-count,MaxY-(MaxY-150)/2+5);
sprintf(buf,"%d@",count);
```

```
  outtextxy(orX-count,MaxY-(MaxY-150)/2+7,buf);
  }
        }
     settextjustify(RIGHT_TEXT,CENTER_TEXT);
     for(count=0;count<=150;count+=25)
        {
        if(count)
  {
  line(MaxX/2,orY-count,MaxX/2-5,orY-count);
  i=count*0.2;
  sprintf(buf,"-%d",(int)i);
  outtextxy(MaxX/2-7,orY-count,buf);
  }
        }
    }

void plot_rec(vcid)
      {
      int orX,orY;
      int count;
      orX = MaxX/2; orY = MaxY/2+75;
      for(count=1;count<=180;count++)
{
line(orX+count-1,orY+dB[count-1]/0.2,orX+count,orY-dB[count]/0.2);
}
      line(orX,orY+dB[0],orX-1,orY+dB[359]);
      for(count=358;count>=180;count--)
{
line(orX-360+count+1,orY+dB[count+1]/0.2,orX-360+count,orY+dB[count]/0.2);
}
      Report();
       }
```

วงจรของเครื่องพล๊อตแพทเทิร์นสายอากาศ


IC1
8255
D0
D1
D2
D3
D4
D5
D6
D7
IOR
IOW
A0
A1
RESET
RD
WR
CS
PA0
PA1
PA2
PA3
PA4
PA5
PA6
PA7
PB0
PB1
PB2
PB3
PB4
PB5
PB6
PB7
PC0
PC1
PC2
PC3
PC4
PC5
PC6
PC7
IC6
ADC0809
msb2-1
2-2
2-3
2-4
2-5
2-6
2-7
lsb2-8
EOC
IN-0
CLOCK
START
ref(+)
ALE
ENABLE
ADD-A
ADD-B
ADD-C
ref(-)
VCC
INPUT
C4
0.1u
IC4:B
74LS08
IC3:D
IC3:A
IC3:B
IC3:C
IC4:A
IC2
74LS138
A
B
C
E1
E2
E3
Y0
Y1
Y2
Y3
Y4
Y5
Y6
Y7
A2
A3
A4
A5
A6
A7
AEN
A8
A9
X1
2.45MHz
IC7:A
IC7:B
74LS04
R1
1k
R2
1k
R3
470
C2
151pF
C1
0.1u
C3
151pF
IC8:A
74HCT74
IC8:B
74HCT74
SD
CD
CLK
D
Q