หุ่นยนต์โปรแกรมและสั่งการโดยคอมพิวเตอร์ไร้สาย

Robot Control By Computer Wireless

จัดทำโดย

นาย นที	ทนุผล	รหัส	35103101
นาย บัณฑิต	ปิติยานุวัฒน์	รหัส	35103102
นาย สุทัศน์	เพิ่มพูลพานิช	รหัส	35103124
นาย อุทิศ	มิ่งมณี	รหัส	35103129

วิทยานิพนธ์ชิ้นนี้เป็นส่วนหนึ่งของหลักสูตร อุตสาหกรรมศาสตร์บัณฑิต ตามหลักสูตรวิชาชีพวิศวกรรม ภาควิชา เทคนิคอุตสาหกรรม สาขาเทคโนโลยีอิเลคทรอนิกส์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

หุ่นยนต์โปรแกรมและสั่งการโดย COMPUTOR ไร้สาย

จัดทำโดย

นาย นที	ทนุผล	รหัส	35103101
นาย บัณฑิต	ปิติยานุวัฒน์	รหัส	35103102
นาย สุทัศน์	เพิ่มพูลพานิช	รหัส	35103124
นาย อุทิศ	มิ่งมณี	รหัส	35103129

อาจารย์ที่ปรึกษา รศ.ดร. กนก เจนจิระพงษ์เวช

ภาค เทคนิคอุตสาหกรรม สาขา เทคโนโลยีอิเลคทรอนิคส์

คณะ วิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

คณะกรรมการผู้ตรวจสอบ ปริญญานิพนธ์

..... ประธานกรรมการ

( )

..... กรรมการ

( )

..... กรรมการ

( )

..... กรรมการ

( )

หุ่นยนต์โปรแกรมและสั่งการโดยคอมพิวเตอร์ไร้สาย

จัดทำโดย

1. นาย นที ทนุผล รหัส 35103101
2. นาย บัณฑิต ปิติยานุวัฒน์ รหัส 35103102
3. นาย สุทัศน์ เพิ่มพูลพานิช รหัส 35103124
4. นาย อุทิศ มิ่งมณี รหัส 35103129

อ.ที่ปรึกษา รศ.ดร. กนก เจนจิระพงศ์เวช

บทคัดย่อ

หุ่นยนต์ควบคุมโดยคอมพิวเตอร์ไร้สาย เป็นเทคโนโลยีของการสื่อสารที่ทันสมัย เช่นในโรงงานอุตสาหกรรมเครื่องจักร ที่ต้องควบคุมด้วยคอมพิวเตอร์ ถ้าต้องการระยะทางไกล ๆ จะเป็นการสิ้นเปลืองต้นทุน ปัญหานี้สามารถแก้ไขได้โดย การสื่อสารไร้สาย จึงได้มีการวิจัยสร้างหุ่นยนต์ควบคุมโดยคอมพิวเตอร์ไร้สายขึ้นมา เป็นหุ่นยนต์ที่ใช้โปรแกรมจากคอมพิวเตอร์ผ่านโมเด็ม และมีรูปแบบของการส่งข้อมูลให้หุ่นยนต์ เมื่อหุ่นยนต์ได้รับคำสั่งก็จะทำงานไปตามที่ละคำสั่ง

ABSTRACT

Robot control by computerized wireless is one of the advanced technology . It has been in modern industrial and machinry factories . For long distance cotroller , it is more convernience and more economical to controlled the operation of any equipment to via wireless communication . Herein , the robot is built up . The robot is controlled by computer program via electronic modem . When the robot get the computerized ,it get ready to work out.

กิตติกรรมประกาศ

ปริญญานิพนธ์สำหรับงานวิจัยชิ้นนี้ได้สำเร็จลุล่วงไปได้ด้วยดีเนื่องจากบุคคลหลายฝ่ายที่ให้ การสนับสนุน ช่วยเหลือและให้กำลังใจตลอดมา ขอขอบพระคุณ มารดา ที่อุตสาห์ส่งเสียให้มีโอกาส ได้ศึกษาตามที่ตั้งใจไว้ ขอขอบพระคุณ ผศ.ดร.กนก เจนจิระพงษ์เวช อาจารย์ที่ปรึกษา ตลอด จนไปถึง คุณชำนาญ เฉลิมยุทธ เพื่อนร่วมหอผู้ที่สร้างสมาธิในการออกแบบโครงงานวิจัย ชิ้นนี้ โครงงานวิจัยชิ้นนี้ได้สำเร็จได้ด้วยทีมงานดังต่อไปนี้

1. นายบัณฑิต ปิติยานุวัฒน์ ผู้ออกแบบวางระบบการทำงานของฮาร์ดแวร์และซอฟท์แวร์
2. นายนที ทนุผล ผู้ออกแบบแมคคานิคส์ของหุ่นยนต์และระบบการขับเคลื่อน
3. นายอุทิศ มิ่งมณี ผู้จัดการด้านจัดหาข้อมูลและผู้ประสานงานด้านซอฟท์แวร์
4. นายสุทัศน์ เพิ่มพูลพานิช ผู้ออกแบบด้านซอฟท์แวร์และจัดหาข้อมูล

คำนำ

ปัจจุบันวิวัฒนาการทางด้านเทคโนโลยีได้มีการพัฒนาไปมากโดยเฉพาะอย่างยิ่งใน โรงงานอุตสาหกรรม เราจะเห็นได้ว่ามีการนำหุ่นยนต์มาทำงานแทนมนุษย์ ซึ่งจะทำให้ผลผลิตที่ออกมามีประสิทธิภาพที่แน่นอน รวมทั้งช่วยลดต้นทุนในด้านการผลิตทั้งในด้านแรงงานและทางด้านทรัพยากร แต่ถึงกระนั้นก็ตาม การนำหุ่นยนต์หรือเครื่องมือเครื่องจักรบางอย่างมาใช้ก็อาจจะประสพปัญหาเกี่ยวกับการวางสายระบบบควบคุมระหว่างตัวควบคุม (Computer) กับเครื่องจักรกล(Robot) จะต้องอาศัยตัวกลางเป็นตัวต่อเชื่อมในการควบคุม ซึ่งเป็นสิ่งที่ไม่สามารถหลีกเลี่ยงได้ ถ้าพูดถึงงานควบคุมส่วนใหญ่แล้ว ยังคงอาศัยสายสื่อนำสัญญาณต่างๆ มาเป็นตัวกลางในการเชื่อมต่อระหว่างส่วนควบ คุมกับเครื่องจักรกล และถ้าหากว่าพิจารณากันให้ดีแล้วจะพบว่า การใช้สายไฟ หรือสาย สื่อนำสัญญาณมาเป็นตัวกลางเชื่อมอาจเป็นปัญหาหรืออุปสรรคต่อการทำงานได้ เราจึงใช้เทคโนโลยีการสื่อสารไร้สาย (Wireless Communication) มาแก้ปัญหาตรงจุดนี้ ปริญญานิพนธ์นี้จะเป็นประโยชน์อย่างมากแก่ผู้ที่ต้องการจะค้นคว้าเกี่ยวกับการสื่อสารไร้สาย ทีมงานผู้จัดทำขอขอบคุณท่านอาจารย์ที่ปรึกษา รศ.ดร.กนก เจนจิระพงศ์เวช มา ณ.ที่นี้ด้วย

6 พฤศจิกายน 2537

สารบัญ

วัดสภาพอุณหภูมิและเก็บ ข้อมูลนั้นได้อีกด้วย สำหรับลักษณะการควบคุมของคำสั่งหุ่นยนต์นั้นจะอาศัยการป้อนโปรแกรม คำสั่ง

ผ่าน Computer ซึ่งคำสั่งที่ใช้ในการควบคุมนั้นจะเป็นลักษณะText File และจะ ถูกส่งออกมาทาง port ขนานผ่านโมเดมซึ่งเป็นรหัส ASCII จากนั้นโมเดมก็จะทำการ ส่งสัญญาข้อมูลคำสั่งออกไป ซึ่งเป็นข้อมูลระบบดิจิตอลมีรูปแบบ PROTOCAL ที่ถูกออกแบบมาไว้ โดยเฉพาะ ส่งผ่านตัวกลางคือ "อากาศ" ผ่านไปยังหุ่นยนต์ (Robot) แล้วหุ่นยนต์จะทำ งานตามคำสั่งที่โปรแกรมเอาไว้ สำหรับหุ่นยนต์ที่เราทำการออกแบบนี้มีชื่อว่า "Continy" ซึ่งเป็นชื่อทีมงาน ของนักศึกษาสังกัดภาควิชา "เทคนิคอุตสาหกรรม" หลักสูตร อส.บ.

หลักการโดยทั่วไป

นับตั้งแต่มนุษย์เริ่มมีการนำเอาระบบไฟฟ้ามาใช้ในระบบมาใช้ในการติดต่อสื่อสาร จะเห็นได้ว่ามีการพัฒนาระบบการสื่อสารขึ้นเรื่อยๆ นับตั้งแต่ โทรเลข,โทรศัพท์ ตลอดไปถึงระบบการติดต่อสื่อสารผ่านดาวเทียม และจากอดีตจนถึงปัจจุบันการสื่อสารมีอยู่ด้วยการหลาย รูปแบบซึ่งสามารถแบ่งแยกรูปแบบตามหลักใหญ่ ๆได้ดังนี้

1.การสื่อสารระหว่างคนกับคนซึ่งได้แก่ โทรศัพท์ และวิทยุสมัครเล่น

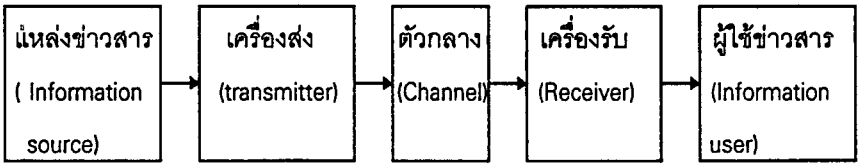
2.การสื่อสารระหว่างคนกับเครื่อง ในการสื่อสารประเภทนี้จำเป็นจะต้องมีการเก็บบันทึกข้อมูลไว้ในจานแม่เหล็กเพื่อสะดวกแก่การดึงข้อมูลที่ต้องการออกมาซึ่งอาจ ปรากฏบนจอ CRT หรือเครื่องพิมพ์ Printer นอกจากนี้ยังใช้ในการติดต่อสอบถาม ข้อมูลจากเครื่องคอมพิวเตอร์เชิงโต้ตอบ

3.การสื่อสารระหว่างเครื่องกับเครื่องเป็นการสื่อสารในลักษณะย้ายข้อมูลจากเครื่องหนึ่งไปยังอีกเครื่องหนึ่งวัตถุประสงค์ของการสื่อสารประเภทนี้ มุ่งที่จะใช้ ทรัพยากรร่วมให้มีประสิทธิภาพสูงขึ้น ซึ่งได้แก่ การสื่อสารระบบ LAN และ WAN

หลักการของระบบสื่อสาร

ระบบสื่อสารดังกล่าวข้างต้นสามารถอธิบายการทำงานอย่างง่ายๆตามรูปที่ 1 ดังนี้ ข่าวสารจากแหล่งข่าวจะถูกแปลงให้อยู่ในรูปของสัญญาณไฟฟ้าที่เรียกว่า สัญญาณโมดูเลติ้ง (Modulating signal) สัญญาณโมดูเลติ้งนี้จะถูกแปลงให้อยู่ในรูปของโค็ด ตามวิธีการ สื่อสารแบบดิจิตัล หรือส่งตรงเข้าเครื่องส่ง(transmitter)ตามวิธีการสื่อสารแบบอะนาล็อก ก็ได้ในเครื่องส่งจะมีเครื่องโมดูเลเตอร์(modulator) ที่ทำหน้าที่โมดูเลทสัญญาณโมดูเลติ้ง เข้ากับตัวพา (carrier) ตัวพานี้มีกำลังส่งสูงพอที่จะพาสัญญาณโมดูเลติ้งไปที่ไกล ๆได้ด้วย ความถี่สูงตามกระบวนการโมดูเลชั่น (modulation) จากนั้นสัญญาณจะถูกคัปปลิง(channel) ออกอากาศโดยเสาอากาศหรือส่งตามสายก็ได้ สัญญาณที่ผ่านทางตัวกลาง (channel) ซึ่ง ไม่ว่าจะเป็นอากาศหรือสายส่งก็ตามจะถูกรบกวนจากเสียงรบกวน (noise) หรือสัญญาณแทรก ที่ไม่พึงปรารถนา เมื่อสัญญาณไปถึงเครื่องรับของผู้ใช้ที่อยู่ปลายทาง เสาอากาศของเครื่องรับจะแปลงสัญญาณที่เป็นคลื่นแม่เหล็กไฟฟ้าเป็นกระแสไฟฟ้า แต่ถ้าส่งตามสายเครื่องรับจะรับจะ รับสัญญาณในรูปของกระแสไฟฟ้าหรือแรงดันไฟฟ้าได้ทันที จากนั้นเครื่องดีโมดูเลเตอร์ (demo dulator) ในเครื่องรับจะแปลงสัญญาณ

ที่มีความถี่สูงให้มีความถี่ต่ำลงและแยกสัญญาณโมดูเลติ้ง ออกจากตัวพา . ตามกระบวนการดีโมดูเลชั่น (demodulation) และถูกถอดโค้ด (decode) กลับเป็นสัญญาณอะนาล็อกตามเดิม ตามวิธีการสื่อสารแบบดิจิตัล



รูปที่ 1 ระบบสื่อสารทั่วไป

พารามิเตอร์ในการวัดความสามารถในการทำงานของระบบสื่อสาร

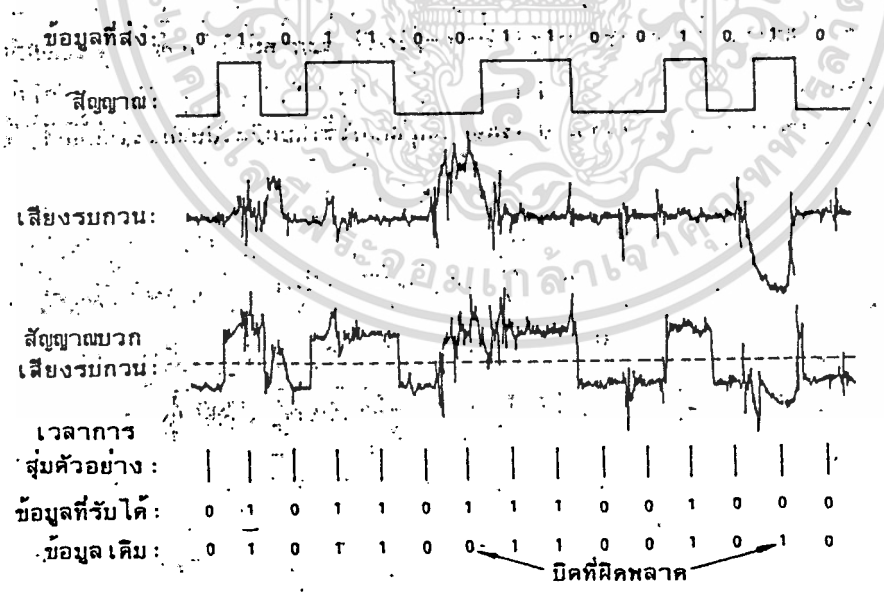
หัวใจสำคัญอีกข้อหนึ่งของการสื่อสารคือ ความสามารถในการสื่อสารของแต่ละระบบ สื่อสารตามปกติถ้าไม่มีเสียงรบกวนเราส่งสัญญาณไปอย่างไรก็มักจะได้รับสัญญาณอย่างนั้นแม้ว่าจะถูกลดกำลังด้วยระยะทางก็ตามแต่ถ้ามีเสียงรบกวนแล้วสัญญาณนั้นจะเพี้ยนไปหรือรับไม่ได้ถ้าเสียงรบกวนมีกำลังมากกว่าสัญญาณ ฉะนั้นจึงต้องออกแบบระบบสื่อสารที่มีการทำงาน ที่ดี หรือมีอัตราส่วนกำลังของสัญญาณต่อกำลังของเสียงรบกวน (signal-to- noise powerratio) หรือค่าเอสเอ็น (S/N) สูงสำหรับระบบสื่อสารแบบอะนาล็อก ถ้าเอสเอ็นมีค่า
มากก็แสดงว่าเครื่องมือสื่อสาร สามารถขจัดเสียงรบกวน (reject noise) ได้ดี สำหรับ พารามิเตอร์ในการวัดความสามารถในการทำงานของระบบสื่องานแบบดิจิตัลนั้นคือ อัตราความ ผิดพลาด (error rate) อัตราความผิดพลาดนี้จะแสดงให้เห็นว่า พัลส์ที่รับได้ทางภาครับ จะผิดไปกี่พัลส์จากจำนวนพัลส์ทั้งหมดที่ส่งมาโดยเฉลี่ย ฉะนั้นการออกแบบระบบการสื่อสาร จะต้องพยายามออกแบบ หรือจัดหาระบบสื่อสารที่มีค่าเอสเอ็นสูง หรือมีอัตราความผิด พลาดน้อย ในขณะเดียวกันระบบสื่อสารนั้นจะต้องใช้แถบความถี่แคบด้วย

ชนิดของสัญญาณ

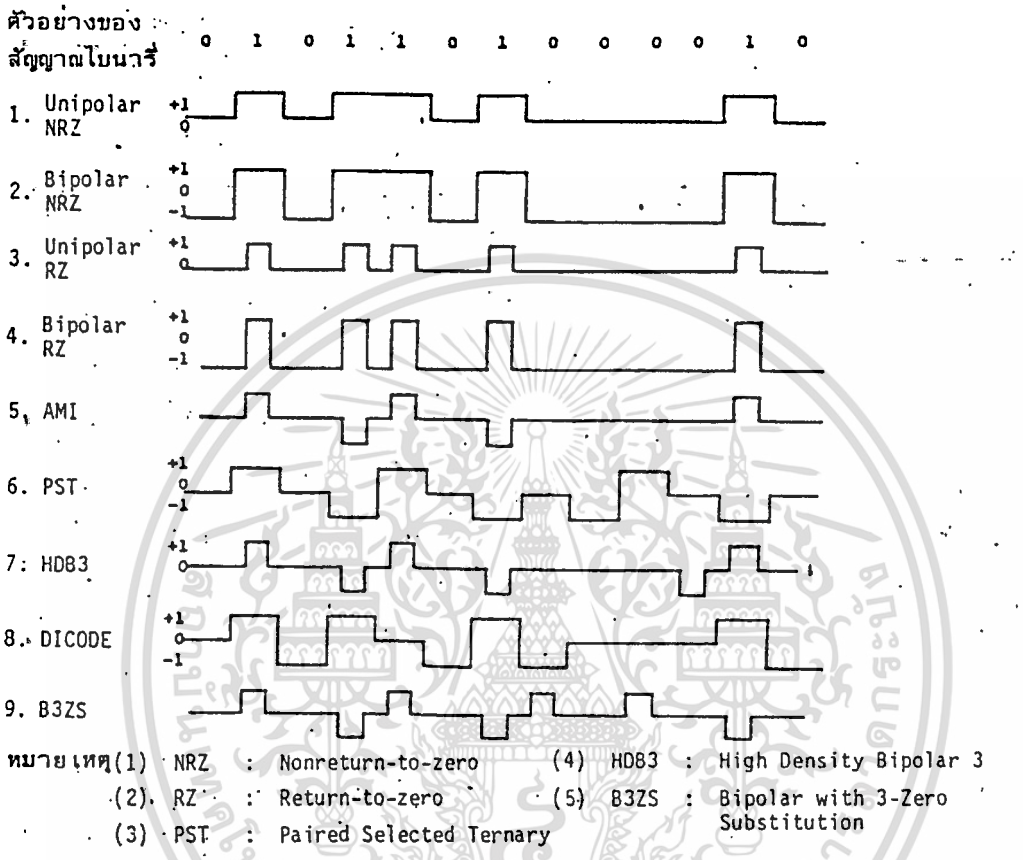
สัญญาณที่ใช้ในการส่งข่าวสาร มีตั้งแต่รูปคลื่นไซน์(sine wave)บริสุทธิ์เพียงความ ถี่เดียว หรือ สัญญาณคอมเพล็กซ์ที่ประกอบด้วยรูปคลื่นไซน์หลายความถี่มารวมกันจนถึงรูปพัลส์ เป็นต้น ในบางกรณีอาจมีการจัดรูปพัลส์ให้อยู่ในรูปที่จะลดความเพี้ยน (distortion) ต่างๆ ที่อาจเกิดขึ้นในวงจรสื่อสารให้พัลส์ผ่านได้โดยไม่เกิดความเพี้ยน

ปัจจุบันนี้ การสื่อสารนิยมใช้สัญญาณดิจิตัลแทนสัญญาณอะนาล็อกมากขึ้น ทั้งนี้เพราะสัญญาณ ดิจิตัล ทนต่อการถูกรบกวนโดยเสียงการรบกวนได้ดีกว่าระบบอะนาล็อก ดังรูปที่ 2 สัญญาณที่ เป็นอะนาล็อกเมื่อ ถูกเสียงรบกวนแล้ว ทางภาครับนอกจากจะแยกสัญญาณออกจากเสียงรบกวน แล้ว ยังจำเป็นต้องมีขนาดและ ความถี่เหมือนเดิมทุกประการ จึงจะไม่เกิดความเพี้ยนขึ้น ส่วน สัญญาณที่เป็นดิจิตัล แม้จะถูกเสียงรบกวน เนื่องจากเราต้องการทราบเพียงขนาดของพัลส์ว่าเป็น "1" หรือ "0" เท่านั้น จึงสร้างพัลส์เป็น "1" หรือ "0" ได้ง่าย โดยเทียบกับค่าที่ตั้งไว้ (threshold) เท่านั้น ความผิดพลาดจะเกิดขึ้นเฉพาะกรณีที่เสียงรบกวนสามารถเปลี่ยน ขนาด ของสัญญาณดิจิตัลจาก"1" เป็น "0" หรือ จาก "0" เป็น "1" เท่านั้น ดังนั้นจึงมีโอกาสเกิด ความผิดพลาด น้อยลง

สัญญาณดิจิตัลที่เป็นพัลส์อาจมีระดับขนาดอะไรก็ได้ไม่จำเป็นต้องเป็น "1" และ "0"เสมอ ไป แต่ถ้าเป็น "1" และ "0" เรียกว่าสัญญาณเลขฐานสองหรือสัญญาณไบนารี่ (binary) ซึ่งมีทั้งแบบไม่กลับศูนย์ (nonreturn-to-zero) และแบบกลับศูนย์ (return-to-zero)นอก จากนี้ยังมีพัลส์ไบโพล่าร์ที่แทน "1" ด้วย "+1" หรือ "-1" สลับกัน และแทน "0" ด้วย "0" นอกจากนี้ยังมีพัลส์แบบอื่นๆ อีกมากที่ใช้ในการสื่อสารดังแสดงในรูปที่3 พัลส์ไบโพล่าร์นี้ นิยมใช้ ในการรับส่งไปตามช่องสื่อสารเพราะไม่มีระดับกระแสตรง (dc level) ทำให้ไม่ถูกบล็อคโดย ตัวประจุในวงจรสื่อสาร



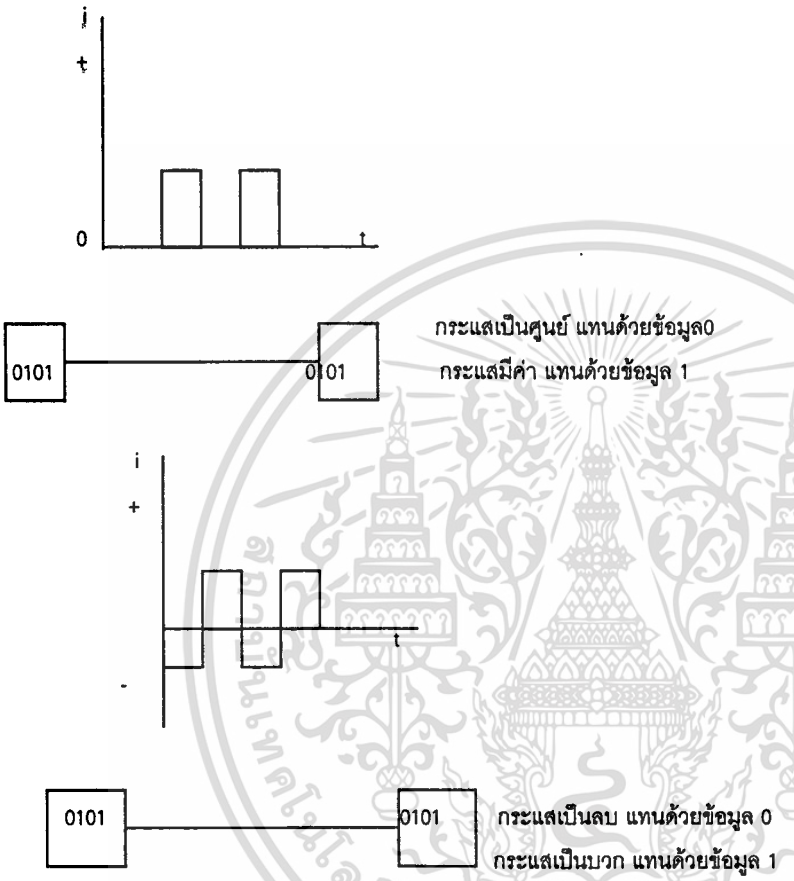
รูปที่ 2 แสดงการรับส่งสัญญาณดิจิตอล



รูปที่ 3 แสดงของรูปคลื่นรหัสต่าง ๆ

การสื่อสารในระบบคอมพิวเตอร์

การสื่อสารในระบบคอมพิวเตอร์นั้นจะมีค่าเพียงสองค่า คือ1กับ 0 ทดแทนลักษณะ ของการเปลี่ยนแปลงของกระแส่และแรงดัน เช่น กระแสไหล-ไม่ไหล , ค่ากระแสเป็น บวก -ลบ ,สภาวะเปิด-ปิด และแรงดัน 0 โวลท์ กับ 5 โวลท์ เป็นต้น ในทางดิจิตอลข้อมูล 0 คือ สถานะลอจิก "Low" และข้อมูล 1 คือสถานะลอจิก "high" ในรูปที่ 4 เป็นลักษณะการ กำหนดสภาวะข้อมูลในคอมพิวเตอร์



รูปที่ 4 การกำหนดสภาวะของข้อมูล 0 และ1 ในระบบสื่อสารดิจิตอล

ในระบบเลขฐานสองจะมีข้อมูล 2 ค่าคือ 0 และ 1 ข้อมูลใน 1 หลักหมายถึง 1 บิต ถ้าข้อมูลมี 3 หลัก จะหมายถึงข้อมูล 3 บิตเป็นต้น ถ้าหากนำข้อมูลมารวมกันหลาย ๆ บิต ก็จะมีชื่อเรียกแตกต่างกันออกไป ดังต่อไปนี้

ข้อมูล 4 บิต เรียกว่า 1 นิบเบิล (nibble)

ข้อมูล 8 บิต เรียกว่า 1 ไบต์(byte)

ข้อมูล 16 บิต เรียกว่า 1 เวิร์ด(word)

ข้อมูล 32 บิต เรียกว่า 1 ดับเบิ้ลเวิร์ด(double word)

ในข้อมูลไม่ว่าจะเป็นไบต์,เวิร์ดหรือดับเบิ้ลเวิร์ด บิตทางขวาสุด(หรือบิตสุดท้ายจะเป็น บิตที่มีนัยสำคัญต่ำสุด หรือ LSB (Least Significaunt Bit) และบิตทางซ้ายสุด (หรือบิตแรก) เป็นบิตที่มีนัยสำคัญสูงสุดหรือ MSB (Most Significant Bit)

รหัสของการสื่อสารข้อมูล

การสื่อสารข้อมูลระหว่างอุปกรณ์ 2 ประเภท เช่น คอมพิวเตอร์กับพรินเตอร์จะต้องมีการกำหนดรูปแบบของข้อมูลที่แน่นอน เพื่อประโยชน์ในการพัฒนาระบบยกตัวอย่างผู้ผลิตพรินเตอร์และเลเซอร์พรินเตอร์ จะต้องผลิตให้ผลิตภัณฑ์ทั้งสองสามารถใช้งานกับพอร์ต ขนานของคอมพิวเตอร์

รูปแบบของการกำหนดรหัสของข้อมูลอย่างง่ายๆเป็นดังนี้ ข้อมูลขนาด 7 บิตการ ที่จะทำเช่นนี้ ได้ ก็ต้องมาจากการกำหนดมาตรฐานของรหัสที่ใช้ในการส่งข้อมูลเหมือนกัน

สมมติให้ข้อมูล 1000001 แทนอักษร A และ 1000010 แทนอักษร B เมื่อต้องการส่งข้อมูลก็จะส่งข้อมูล Aไปก่อนแล้วตามด้วยข้อมูลของรหัสที่ทำหน้าที่แยกชุดข้อมูลออก จากกัน ซึ่งในที่นี้ใช้ช่องว่าง (space) ข้อมูลเป็นตัวแยกข้อมูล แล้วจึงส่งข้อมูล B ตามไป

นั่นคือที่ตัวส่งข้อมูลและตัวรับข้อมูลจะต้องมีการเข้ารหัสและถอดรหัสข้อมูลที่มีมาตรฐานเดียวกัน เพื่อจะได้ข้อมูลที่ปลายทางถูกต้อง

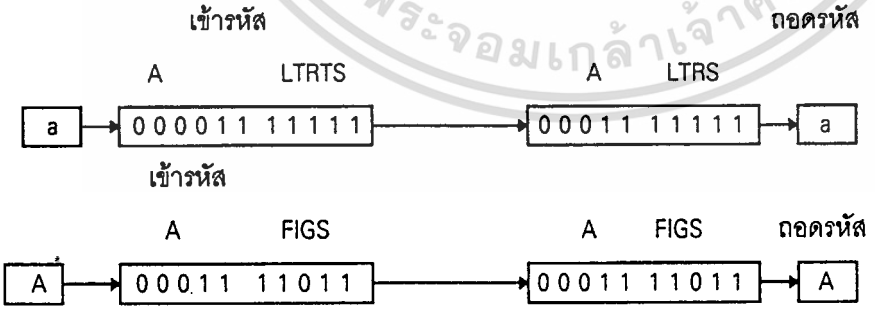
จำนวนของรหัสที่เกิดอาการแทนด้วยข้อมูลเลขฐานสอง จะหาได้จากสูตร2จำนวน บิตที่ใช้ เช่น ถ้าใช้ข้อมูล 7 บิต แทน 1 รหัส ก็จะได้รหัสทั้งสิ้น 2^7 = 128 รหัส สำหรับ รหัสของการสื่อสารข้อมูลที่ใช้เป็นมาตรฐานอยู่ในปัจจุบันมีอยู่หลายรหัส ดังต่อไปนี้

1.รหัสบาวดอต(baudot code)

ประกอบด้วยข้อมูล 5 บิตสามารถแทนตัวเลข 0-9 และตัวอักษร สัญลักษณ์ได้ 32 รูป แบบในตารางที่ แสดงข้อมูลต่างๆ ในรหัสบาวดอต นอกจากนี้ยังสามารถกำหนดลักษณะของ ตัวอักษร A-Z ให้เป็นตัวพิมพ์เล็กหรือตัวพิมพ์ใหญ่ได้โดยส่งรหัสเพิ่มเติมเข้าไป ถ้าหากต้องการ ตัวพิมพ์เล็กก็ทำการส่งรหัสLetter downshift (LTRS:11111)ในทางตรงข้ามถ้าต้องการ ตัวพิมพ์ใหญ่ก็ส่งรหัส Figures upshift (FIGS:11011) รูปแบบการส่งข้อมูลของรหัส บาวดอต แสดงดังรูปที่ 5 รหัสบาวดอตนี้ ใช้ในเครื่องเทเลพรินเตอร์ในสมัยก่อน

Character		Bit Pattern					Character Case		Bit Pattern				
Lower	Upper	5	4	3	2	1	Lower	Upper	5	4	3	2	1
A	-	0	0	0	1	1	Q	1	1	0	1	1	1
B	?	1	1	0	0	1	R	4	0	1	0	1	0
C	:	0	1	1	1	0	S	'	0	0	1	0	1
D	$	0	1	0	0	1	T	5	1	0	0	0	0
E	3	0	0	0	0	1	U	7	0	0	1	1	1
F	!	0	1	1	0	1	V	;	1	1	1	1	0
G	&	1	1	0	1	0	W	2	1	0	0	1	1
H	#	1	0	1	0	0	X	/	1	1	1	0	1
I	8	0	0	1	1	0	Y	6	1	0	1	0	1
J	Bell	0	1	0	1	1	Z	"	1	0	0	0	1
K	(	0	1	1	1	1	Letters(shift)		1	1	1	1	1
L	)	1	0	0	1	0	Figures(shift)		1	1	0	1	1
M	.	1	1	1	0	0	Space(sp)		0	0	1	0	0
N	,	0	1	1	0	0	Carriage return		0	1	0	0	0
O	9	1	1	0	0	0	Ling feed		0	0	0	1	0
P	0	1	0	1	1	0	Blank		0	0	0	0	0

ตารางที่ 1 รูปแบบของรหัสบาวดอต



รูปที่ 5 รูปแบบการส่งรหัสบาวดอต

2.รหัส EBCDIC

รหัส EBCDIC เป็นรหัสที่ถูกพัฒนาขึ้นโดยบริษัท IBM และกลุ่มทำอุปกรณ์เลียนแบบ ที่ใช้กับเครื่องคอมพิวเตอร์ของ IBM (หรือกลุ่มคอมแพตทิเบิล)ใช้สำหรับส่งข้อมูลระหว่างเครื่องคอมพิวเตอร์ขนาดใหญ่ๆ มีขนาด 8 บิต สามารถแทนตัวอักษรหรือสัญลักษณ์ได้ 256 ตัว มีรูปแบบของรหัสตามตารางที่ 2

Bit Position					4	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
					3	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
					2	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
8	7	6	5	1		0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	0	0	0			NUL	SOH	STX	ETX	PF	HT	LC	DEL			SMM	VT	FF	CR	SO	SI
0	0	0	1			DLE	DC1	DC2	DC3	RES	NL	BS	IL	CAN	EM	CC		IFS	IGS	IRS	IUS
0	0	1	0			DS	SOS	FS		BYP	LF	EOB	PRE			SM			ENQ	ACK	BEL
0	0	1	1					SYN		PN	RS	UC	EOT					DC4	NAK		SUB
0	1	0	0			SP										c	.	<	(	+	
0	1	0	1			&										!	$	*	)	;	
0	1	1	0			-	/										,	%	-	>	
0	1	1	1													:	#	@	'	=	
1	0	0	0				a	b	c	e	d	f	g	h	i						
1	0	0	1				j	k	l	m	n	o	p	g	r						
1	0	1	0					s	t	u	v	w	x	y	z						
1	0	1	1																		
1	1	0	0				A	B	C	E	D	F	G	G	I						
1	1	0	1				J	K	L	M	N	O	P	P	R						
1	1	1	0					S	T	U	V	W	X	X	Z						
1	1	1	1			0	1	2	3	4	5	6	7	8	9						

ตารางที่ 2 แสดงการแทนตัวอักษรและสัญลักษณ์ต่างๆด้วยรหัส EBCDIC

3. **รหัส** ASCII

ถูกพัฒนาขึ้นในระหว่างปี 1963-1967 และถูกกำหนดขึ้นเป็นมาตรฐานของ ANSI (American National Standards Institute) เพื่อใช้เป็นรหัสในการสื่อสารข้อมูล ด้วยคอมพิวเตอร์ รหัสนี้เป็นที่นิยมใช้อยู่ในปัจจุบัน

รหัส ASCII ในการพัฒนาขั้นแรกจะมีเพียง 7 บิต จะแทนอักขระได้เพียง 128 ตัว อักษร แต่ต่อมาในปี 1981 ระบบคอมพิวเตอร์ส่วนบุคคลของ IBM จะใช้รหัสแบบ 8 บิต ดัง นั้นจึงได้มีการพัฒนาโดยใช้บิตที่ 1-7 เป็ASCII มารตรฐาน ส่วนบิตที่ 8 จะเรียกว่าเป็น ส่วนขยายของรหัส ASCII ทำให้สามารถเก็บสัญลักษณ์พิเศษเพิ่มได้อีก ถึง 128 ตัว ตาราง ที่ 3 แสดงข้อมูลของรหัส ASCII

Bit Position	7	0	0	0	0	1	1	1	1
	6	0	0	1	1	0	0	1	1
4 3 2 1	5	0	1	0	1	0	1	0	1
0 0 0 0		NUL	DLT	SP	0	@	P		p
0 0 0 1		SOH	DC1	!	1	A	Q	a	q
0 0 1 0		STX	DC2	"	2	B	R	b	r
0 0 1 1		EXT	DC3	#	3	C	S	c	s
0 1 0 0		EOT	DC4	$	4	D	T	d	t
0 1 0 1		ENQ	NAK	%	5	E	U	e	u
0 1 1 0		ACK	SYN	&	6	F	V	f	v
0 1 1 1		BEL	ETB	'	7	G	W	g	w
1 0 0 0		BS	CAN	(	8	H	X	h	x
1 0 0 1		HT	EM	)	9	I	Y	I	y
1 0 1 0		LF	SUB	*	:	J	Z	j	z
1 0 1 1		VT	ESC	+	;	K	[	k	{
1 1 0 0		FF	FS	'	<	L	\	l	:
1 1 0 1		CR	GS	-	=	M	]	m	}
1 1 1 0		SO	RS	.	>	N	^	n	~
1 1 1 1		SI	US	/	?	O	-	o	DEL

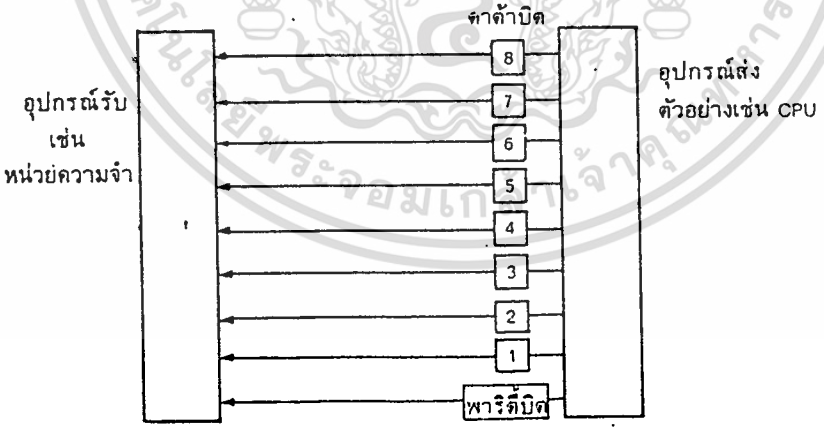
ตารางที่ 3 แสดงรูปแบบรหัส ASCII

ลักษณะของการสื่อสารข้อมูล

โดยทั่วไปหลักใหญ่ของการส่งข้อมูลในคอมพิวเตอร์ หรือระหว่างคอมพิวเตอร์ด้วยกันจะมีลัษณะของกาส่งข้อมูลอยู่ 2 แบบ คือ ส่งแบบขนาน และส่งแบบอนุกรม

1. การถ่ายโอนข้อมูลแบบขนาน

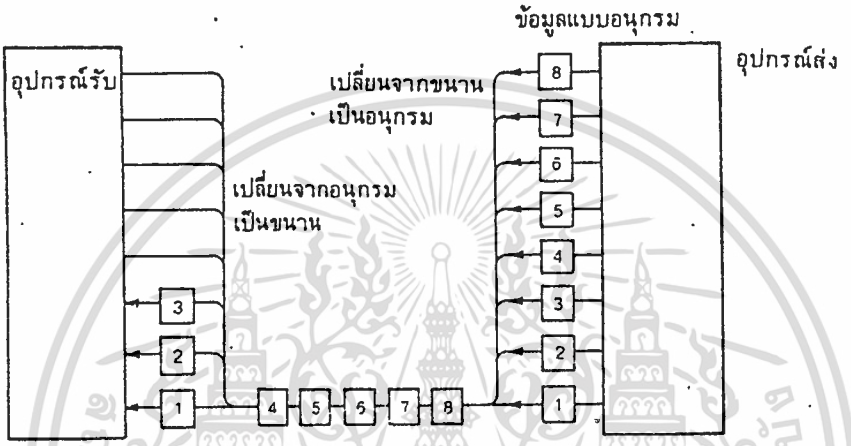
ลักษณะของการส่งข้อมูลแบบขนาน ทำได้โดยการส่งข้อมุลออกมาทีละ 1ไบท์ คือ 8 บิตจากอุปกรณ์ส่งไปยังอุปกรณ์รับ ตัวกลางระหว่าง 2 เครื่องจะต้องมีชองทางให้ข้อมูลเดินทางอย่างน้อย 8 ช่องทาง โดยมากจะเป็นสายขนานให้กระแสไฟฟ้าวิ่งมากกว่าจะเป็นตัวกลางชนิดอื่น เนื่องจากมีสัญญาณสูญหายไปกับความต้านทานของสาย ดังนั้น ระยะทางระหว่าง 2 เครื่องไม่ควรจะเกิน 100 ฟุต ปัญหาที่เกิดขึ้นถ้าหากระยะทางของสายมากกว่านี้ก็คือระดับของกราวด์ ในทางไฟฟ้าที่จุดรับผิดไปจากจุดส่งทำให้เกิดการผิดพลาดในการรับสัญญาณลอจิกทางฝ่ายรับนอกจากสายที่เป็ ทางเดินของข้อมูลแล้วอาจจะมีทางเดินของสัญญาณควบคุมอื่นๆ อีก เป็นต้นว่า บิตที่ บอกพาริตี้ของสัญญาณ เพื่อเป็นการตรวจสอบความผิดผลาดของการรับสัญญาณที่ปลายทาง หรือสายที่ควบคุมการโต้ตอบ (Hand-shake) ดังที่กล่าวมาแล้วจะเห็นว่าการส่งแบบขนานส่วนมากจะทำในระยะใกล้ ๆ เนื่องจากจะต้องมีช่องทางเดินของสัญญาณมากกว่า 8 สาย และอุปกรณ์ที่ติดต่อแบบขนานกับคอมพิวเตอร์ ได้แก่ เครื่องพิมพ์ , พล๊อตเตอร์ เป็นต้น



รูปที่ 6 การส่งข้อมูลแบบขนาน

2.การถ่ายโอนข้อมูลแบบอนุกรม

ในการถ่ายโอนข้อมูลแบบอนุกรม ข้อมูลถูกส่งออกมาทีละบิตระหว่างจุดส่งและจุดรับจะเห็นว่าการส่งข้อมูลแบบนี้จะช้ากว่าแบบขนาน เหตุผลที่ต้องส่งแบบนี้ก็คือ ตัวกลางการสื่อสารต้องการเพียงช่องเดียวหรือสายเพียงคู่เดียว ค่าใช้จ่ายในสื่อกลางจะต้องถูกกว่าแบบขนาน



รูปที่ 7 การส่งข้อมูลแบบอนุกรม

รูปที่ 7 แสดงให้เห็นการส่งข้อมูลแบบอนุกรม ข้อมูลจากจุดส่งจะถูกเปลี่ยนให้เป็น อนุกรมเสียก่อนแล้วค่อยทยอยส่งออกทีละบิตไปยังจุดรับ ณ ที่จุดรับจะต้องมีกลไกในการเปลี่ยน ข้อมูลที่ส่งมาทีละบิต ให้เป็นสัญญาณแบบขนานซึ่งลงตัวพอดีนั่นคือ บิต 1 ลงที่บัสข้อมูลเส้นที่ 1 พอดี การที่จะทำให้การแปลงสัญญาณจากอนุกรมทีละบิตให้ลงพอดีนั้นจำเป็นจะต้องมีกลไกที่เหมาะสม เพื่อป้องกันการผิดพลาดในการรับ กลไกที่ว่านี้แบ่งเป็น 2 แบบ คือ

1.การสื่อสารแบบซิงโครนัส

2.การสื่อสารแบบอะซิงโครนัส

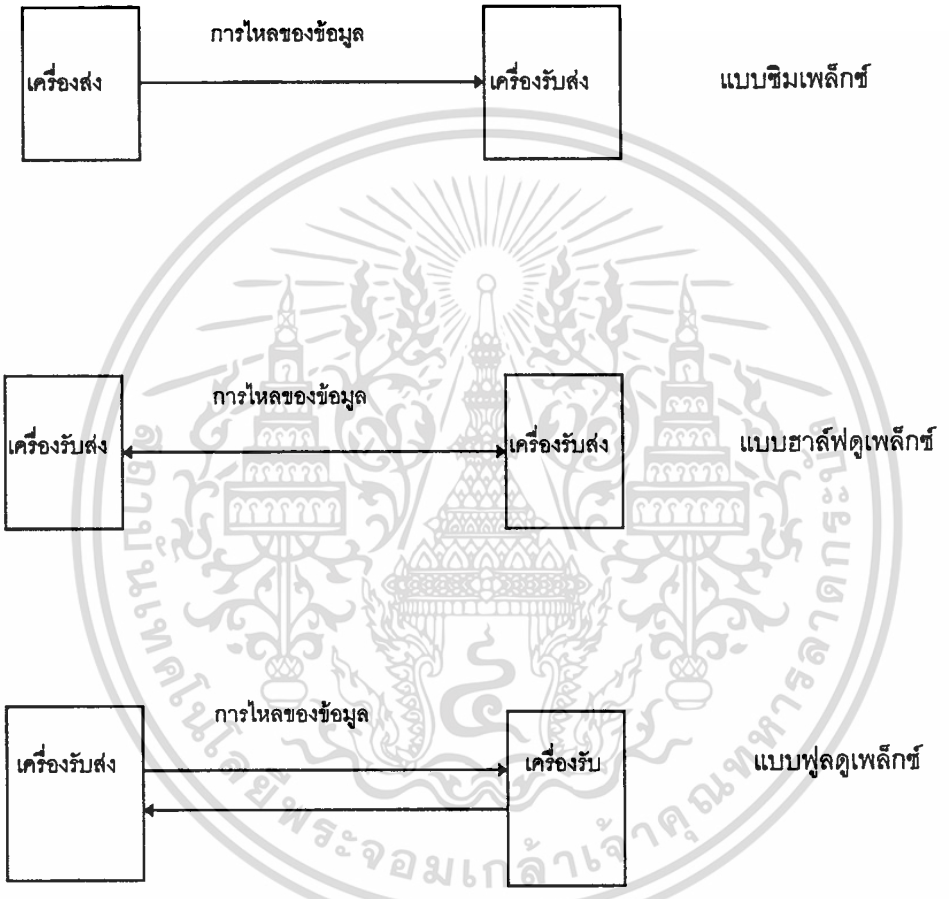
รูปแบบของการติดต่อสื่อสาร

การติดต่อแบบอนุกรมอาจจะแบ่งตามรูปลักษณะได้ 3 แบบ ตามรูปที่ 2.5

1.แบบซิมเพลกซ์(simplex) ข้อมูลส่งได้ในทางเดียวเท่านั้น บางครั้งก็เรียกว่า การส่งทิศทางเดียว (Unidirectional data bus)

2.แบบฮาล์ฟดูเพล็กซ์ (half duplex) ข้อมูลสามารถส่งได้ทั้งสองสถานี แต่จะต้องผลัดกันส่งและผลัดกันรับ จะส่งและรับพร้อมกันไม่ได้

3.แบบฟูลดูเพล็กซ์ (full duplex) ทั้งสองสถานีสามารถรับและส่งได้ในเวลาเดียวกัน



รูปที่ 8 รูปแบบของการติดต่อสื่อสารข้อมูลแบบอนุกรม

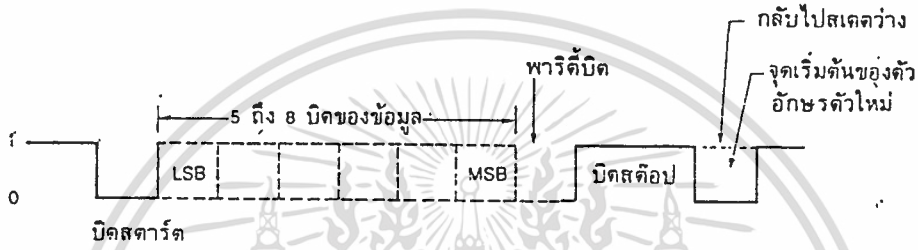
ความเร็วในการถ่ายโอนข้อมูลแบบอนุกรม

ความเร็วของการถ่ายโอนข้อมูลแบบอนุกรม หน่วยวัดเป็นบิตต่อวินาที(bps)หน่วยที่ บรรยายถึงการเปลี่ยนแปลงของสัญญาณใน 1 วินาที เรียกว่าบอดเรต(baud rate)หรือ อัตราบอด การเปลี่ยนแปลงของสัญญาณ 1 ครั้ง อาจจะแสดงถึงการส่งข้อมูลแบบอนุกรมมาก กว่า 1 บิต ถ้าเขียนในรูปของสมการทางคณิตศาสตร์เราก็จะได้

อัตราบิต(bit rate) = อัตราบอด(baud rate) * บิตใน 1 บอด

การสื่อสารแบบอะซิงโครนัส

การส่งแบบอะซิงโครนัสนี้ พัฒนามาจากการส่งโทรพิมพ์ในสมัยก่อน ลักษณะของสัญญาณ แสดงไว้ในรูปที่ 9 เพื่อเพิ่มกลไกในการรับส่งอย่างถูกต้อง สัญญาณอะซิงโครนัส จะประกอบ ด้วยบิตเริ่มต้นหรือบิตสตาร์ต (start) และบิตสิ้นสุดหรือบิตสต๊อป(stop bit)



รูปที่ 9 ฟอร์แมตการสื่อสารแบบอะซิงโครนัส

ขณะที่สภานะของการส่งเป็นแบบว่าง (Idle) คือยังไม่มีสัญญาณส่งออกมาจะมีสัญญาณหรือมีแรงดัน(หรือกระแส)ตลอดเวลา เพื่อความแน่ใจว่าฝ่ายรับยังติดต่ออยู่กับฝ่ายส่ง เมื่อเริ่มจะส่งข้อมูล สัญญาณของอะโครนัสจะเป็น 0 หนึ่งช่วงสัญญาณนาฬิกา บิตนี้เรียกว่า สตาร์ทบิต ตามหลังของสตาร์ตบิตก็จะเป็นข้อมูลสำหรับ 1 ตัวอักษร ซึ่งอาจจะมีขนาดตั้งแต่ 5 บิต จนถึง 8 บิต โดยบิตที่มีค่าน้อยที่สุด (LSB) จะส่งออกมาก่อนไล่ไปจนถึงบิตที่มีค่ามากที่สุด (MSB) การเข้ารหัสอักขระนี้ส่วนมากจะนิยมใช้รหัส ASCII แรกเริ่มทีเดียวในงานของโทรพิมพ์เขาใช้รหัส Baudot ซึ่งใช้ 5 บิต ในการแทนอักขระ 1 ตัว ตามหลังข้อมูล ก็จะเป็นพาริตี้บิต ซึ่งอาจจะใช้หรือไม่ใช้ก็ได้ พาริตี้บิตทำหน้าที่เป็นตัวตรวจสอบความถูกต้อง ของสัญญาณที่ได้รับ พาริตี้บิตอาจจะเป็นแบบคู่(Even)หรือแบบ คี่ (Odd)หมายความว่า ถ้าหากเป็นพาริตี้คู่ จำนวนบิตที่เป็น 1 ในช่วงบิตข้อมูลกับบิตพาริตี้รวมแล้วจะต้องเป็นจำนวนคู่ผู้ส่งจะต้องทำหน้าที่ตรวจสอบข้อมูลแล้วใส่พาริตี้บิตเอง ฝ่ายรับเมื่อรับแล้วก็จะต้องตรวจสอบ ดูว่าเป็นจริงดังสถานการณ์ที่ตั้งเอาไว้หรือไม่ หากผิดพลาดก็หมายความว่าสัญญาณที่รับนั้นผิดพลาดไปจากสถานีส่งออกมา ทั้งนี้ทั้งนั้นจะต้องผิดเป็นจำนวนคี่เท่านั้น คือ ผิดไป 1 บิต 3 บิต หรือ 5 บิต พร้อมกันจึงจะตรวจสอบได้ว่าผิด ย้อนกลับมาดูสัญญาณอะซิงโครนัสใหม่ หลังจากบิตพาริตี้แล้วก็จะต้องมีสต๊อปบิตซึ่งเป็น 1 ความกว้างของสต๊อปบิตอาจจะเป็น 1,1.5 หรือ 2 พัลส์ของสัญญาณนาฬิกา แล้วแต่ผู้รับและผู้ส่งจะตกลงใช้กันเอง การเริ่มใช้พอร์ตอนุกรม (ทางออกอนุกรม)จึงจำเป็นจะต้องตั้งค่าต่าง ๆ สำหรับเป็นการส่งแบบอนุกรมอันได้แก่

1.ความเร็วในการส่ง

2.ความยาวรหัส 1 อักขระ

3.บิตตรวจสอบ

4.จำนวนสต๊อปบิต

ในการส่งโทรพิมพ์หรือโทรเลขเมื่อก่อนนี้ใช้ความเร็วแค่ 70 บอด และ 110 บอด สำหรับคอมพิวเตอร์ ความเร็วในการส่งมีให้เลือกตั้งแต่ 110,200,300,1200,2400,4800 ,9600 บอดและสูงไปกว่านั้นจะเห็นว่ากลไกในการซิงโครนัสของการสื่อสารอะซิงโครนัส มีลักษณะเป็นไปทีละตัวอักขระ จำนวนพัลส์ของสัญญาณที่ส่งออกยังมีบางส่วนใช้ในการควบคุมการส่งอยู่อันได้แก่ บิตสตาร์ต บิตสต๊อป และบิตพาริตี้ ทำให้ความเร็วการส่งอักขระต่อวินาทีน้อยลงไป การส่งสัญญาณด้วยความเร็ว 300 บอด สำหรับการเข้ารหัส 7 บิต ไม่ได้หมายความว่าส่งได้ 300 หารด้วย 7 อักขระต่อวินาที

โปรโตคอล

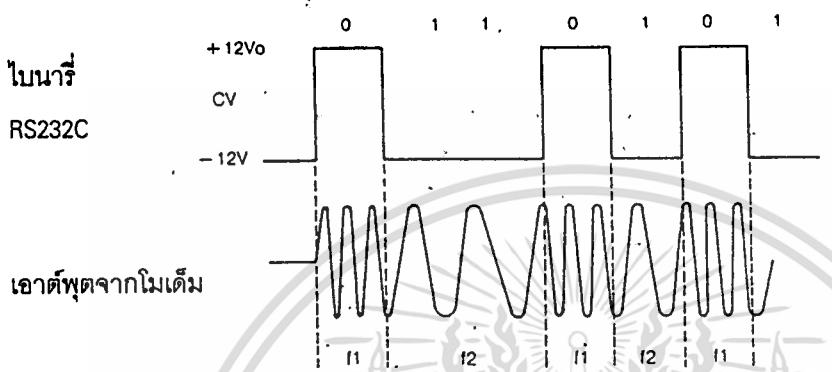
คือกฎข้อบังคับที่กำหนดรูปและขั้นตอนการทำงานของฮาร์ดแวร์ (Hard Wave)และซอฟต์แวร์ (Soft ware) เพื่อให้ทราบว่ามีการรับส่งข้อมูลเป็นอย่างไร เทคนิคทางด้านโปรโตคอลของสัญญาณจะต้องมีโปรโตคอลที่เหมือนกัน โดยอาจจะใช้อักขระควบคุมตามตารางของรหัส ASCII ในบางครั้งโปรโตคอลยังสามารถแก้ไขข้อผิดพลาดที่เกิดขึ้นในการรับส่งในการสื่อสารได้อีกด้วย เช่น โปรโตคอลที่ใช้ในระบบสื่อสารดาวเทียม

โมเด็ม

โมเด็มย่อมาจาก Modulator Demodulator ใช้ในการแปลงสัญญาณทางลอจิกให้เหมาะสมก่อนที่จะส่งผ่านตัวกลางที่มีความกว้างของแถบคลื่นของแถบคลื่นต่ำๆ อย่างเช่นสายโทรศัพท์ ทำไมสัญญาณทางลอจิกจึงส่งออกไปโดยตรงไม่ได้ เหตุผลเพราะว่าสัญญาณ ลอจิกมีลักษณะเป็นคลื่นสี่เหลี่ยม (square wave) "0"และ"1" ซึ่งอาจจะแทนด้วยค่าของ แรงดันสองค่า คลื่นรูปสี่เหลี่ยมประกอบด้วยคลื่นรูปซายน์หลายความถี่ ที่เป็นทวีคูณของความถี่พื้นฐาน หากผ่านตัวกลางที่มีแถบความกว้างของคลื่นต่ำแล้วความถี่สูง ๆ ก็จะหาย ไป เหลือสัญญาณที่ปลายทางผิดเพี้ยนไปจากเดิม ดังนั้นจำเป็นที่เราจะต้องเปลี่ยนสัญญาณลอจิกให้อยู่ในรูปแบบที่เหมาะสมก่อนที่จะส่งออกไป(ผจญกับโลกภายนอก) ข้างฝ่ายรับก็จำเป็น ต้องเปลี่ยนสญญาณที่ถูกแปลงมานี้กลับให้เป็นสัญญาณทางลอจิก และก็จะต้องมีขบวนการที่ตรงกันข้ามกับการฝ่ายส่ง อุปกรณ์ที่ทำหน้าที่ทั้งสองอันนี้จังเรียกว่า โมเด็ม(MODEM)

ลักษณะของ modem ที่มีใช้กันอยู่โดยทั่วไปนี้มีอยู่หลายรูปแบบโดยจะขึ้นอยู่กับ ความเร็วในการสื่อสาร, ลักษณะของการโมดูเลท คือ ว่าเป็นแบบใดซึ่งมีมากมาย เช่น ASK ,FSK ,PAK ฯลฯ ซึ่งแต่ละชนิดจะมีการแปลงรูปสัญญาณที่ต่างกันออกไป สำหรับลักษณะ Modem ที่จะกล่าวถึง คือ Modem ชนิด FSK (Fregvency shift Keying) คือ ใช้ความถี่ของเสียงสองความถี่สำหรับแทนสัญญาณลอจิก "0"และ "1" ฝ่ายรับก็พยา

ยามจับเอาสองความถี่ที่ว่านี้มาแปลงเป็นสัญญาณลอจิกกลับคืน ความถี่ของเสียงทั้งสองเสียง ต้องห่างกันพอที่จะแยกออกจากกันได้โดยวงจรอิเล็กทรอนิกส์ และก็จะต้องไม่ห่างเกินจนตก ขอบของแบนด์วิทด์ของวงจรที่จะนำพาไปได้รูปที่10 แสดงถึงหลักการทำงานของ modemFrequency Shift Keying เรียกย่อ ๆ ว่า FSK



รูปที่ 10 การแปลงสัญญาณของโมเด็ม

ลักษณะของการ Modularor กับสัญญาณ Carrier

ในระบบสื่อสาร สัญญาณที่จะนำไปโมดูเลทเรียกว่า สัญญาณโมดูเลติ้ง (Modulating signal)และคลื่นที่จะพาสัญญาณไปที่ไกล ๆ ได้เรียกว่า ตัวพา (Carrier)เมื่อสัญญาณโมดูเลติ้งโมดูเลทกับตัวพาแล้ว สัญญาณที่ได้เร็วกว่า สัญญาณถูกโมดูเลท(modulated signal) เหตุผลของการโมดูเลชั่นนั้นสามารถอธิบายประโยชน์ต่าง ๆดังนี้

1.ทำให้สัญญาณมีกำลังสูงทำให้สามารถเดินทางไปที่ไกล ๆ ได้

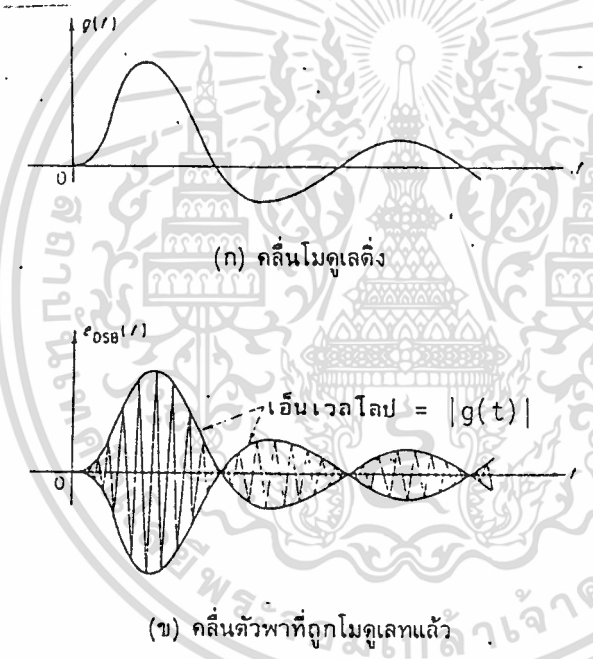
2.ทำให้สัญญาณมีความถี่สูงขึ้น ซึ่งเหมาะกับการรับส่งสัญญาณมากขึ้น เพราะใช้เสาอากาศที่สั้นลงได้

3.สามารถแบ่งความถี่ให้หลายๆ สัญญาณส่งพร้อมกันภายใต้ตัวพาตัวเดียวกันได้ที่ เรียกว่า มัลติเพล็กซ์ซิ่ง (multiplexing)

4.ทำให้สัญญาณมีภูมิต้านทางการรบกวนของเสียงรบกวน (noise) หรือของสัญญาณ รบกวน (interfering signal) ได้ดีขึ้น

ปัจจุบันนี้ระบบโมดูเลชั่นที่ใช้ในการสื่อสารนั้นมีอยู่ด้วยกันหลายระบบ แต่ละระบบมีชื่อเรียกตามลักษณะการโมดูเลทสัญญาณต่าง ๆ สำหรับระบบที่เราจะกล่าวถึงคือ ระบบ AM (Amplitude Modulation)

แอมพลิจูดโมดูเลชั่นนับเป็นวิธีการโมดูเลชั่นที่เก่าแก่ที่สุดของโลกวิธีหนึ่งเริ่มใช้ในปลาย ศตวรรษที่ 18 จนทุกวันนี้ก็ยังใช้อยู่ แอมพลิจูดโมดูเลชั่นที่ใช้กันอยู่ทุกวันนี้ส่วนมากใช้ในระบบ วิทยุกระจ่ายเสียง(broadcasting) การแพร่ภาพของทีวี การรับส่งสัญญาณผ่านคลื่นสั้น (short wave) หรือคลื่นความถี่สูง (high frequency) และในระบบเรดาร์ (radar) แบบง่าย ๆ เป็นต้น สำหรับลักษณะการ Mod ของระบบ AM นั้นเป็นอย่างไร เราจะเริ่ม อธิบายได้ดังต่อไปนี้ ถ้าให้สัญญาณโมดูเลติ้ง g(t) มีความถี่ต่ำกว่า f_m ($0 \le f \le f_m$) สัญญาณนี้เรียกว่า สัญญาณเบสแบนด์ (baseband signal) เพราะมีความถี่ต่ำถูกจำกัดแถบ ความถี่อยู่ (bandlimited) สัญญาณโมดูเลติ่งที่ผ่านเครื่องโมดูเลทแบบบาลานซ์ ซึ่งในอุปกรณ์ ไม่เชิงเส้น (nonlinear device) จะลบล้างไม่ให้ส่วนของตัวพา (carrier component) ปรากฏที่ขาออก DSB ที่ไม่มีส่วนของตัวพา (carrier component) นี้จะมีแถบความถี่อยู่ 2 ข้าง ของความถี่ของตัวพา จังเรียกว่า Double Sideband Suppressed Carrier (DSB-SC)



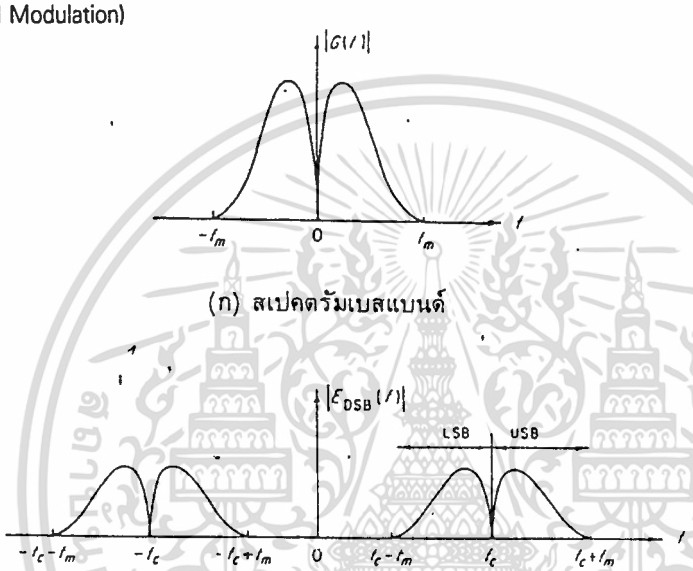
รูปที่ 11 สัญญาณแบสแบนด์และรูปคลื่นแบบไซด์แบนด์คู่

สมมติให้สัญญาณโมดูเลติ่งเป็น g(t) และตัวพาเป็นไซน์ยูซอยดัล สัญญาณขาออกของเครื่องโมดูเลทแบบบาลานซ์จะได้

$$e_{DSB}(t) = g(t)\,C(t) = g(t)\cos 2f_c t, \quad f_c >> f_m \qquad (4\text{-}1)$$

จะเห็นได้ว่าโมดูเลชั่นไซด์แบนด์คู่ก็คือ การคูณกันระหว่างตัวพากับสัญญาณโมดูเลติ่งนั่นเองรูปที่ 11 แสดงให้เห็นถึงผลคูณของสัญญาณเบสแบนด์กับคลื่นตัวพาแบบไซน์ยูซอยดัลตาม รูปสัญญาณที่ถูกโมดูแลทจะมีขนาด (amplitude) เปลี่ยนตามขนาดของสัญญาณโมดูเลติ่ง โดยมีความถี่เท่ากับความถี่ของตัวพา

ก่อนที่จะมีการโมดูเลชั่น สเปคตรัมของสัญญาณเบสแบนด์จะถูกจำกัดแถบความถี่อยู่ภายใน f_m เท่านั้น เมื่อโมดูเลทกับตัวพาแล้วจะถูกย้ายความถี่ไปที่ความถี่ตัวพาโดยขนาดของสเปคตรัมจะลดลงครึ่งหนึ่ง และแถบความถี่ที่อยู่เหนือความถี่ตัวพาเรียกว่า ไซด์แบนด์สูง (Upper Sidebannd(USB)) และที่อยู่ต่ำกว่าความถี่ตัวพาเรียกว่า ไซด์แบนด์ต่ำ(Lower Sideband (LSB))ฉะนั้นเราจึงเรียกชื่อโมดูเลชั่นนี้ว่า โมดูเลชั่นไซด์แบนด์คู่ (Double Sideband Modulation)



(ก) สเปคตรัมเบสแบนด์

(ข) สเปคตรัมไซด์แบนด์คู่

รูปที่ 12 สเปคตรัมของ double-sideband

ในระบบเอเอ็ม เอ็นเวลโลบของเอเอ็มจะเปลี่ยนตาม g(t) สัญญาณเบสแบนด์ทุก ขณะ และมีรูปร่างเหมือน g(t) ทุกประการ คลื่นเอเอ็มนั้นกำเนิดได้จากการบวกสัญญาณ dsb เข้ากับคลื่นตัวพา ผลที่ได้จะเท่ากับ

$$e_{AM}(t) = [1+g(t)] \cos 2f_c t \qquad (1)$$

ถ้าเพิ่ม m_a ซึ่งเป็นดัชนีการโมดูเลชั่น โดยให้

$$[g(t_{max})] =< 1,\ 0 < m_a < 1$$

ในสมการที่ (1) จะได้

$$e_{AM}(t) = [1 + m_a\, g(t)] \cos 2f_c t \qquad (2)$$

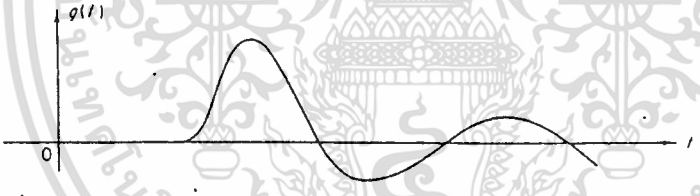
ตามปกติ g(t) จะมีความถี่ f =< f_m และ f_m << f_c , m_a จะเป็น

ตัววัดเปอร์เซนต์โมดูเลท

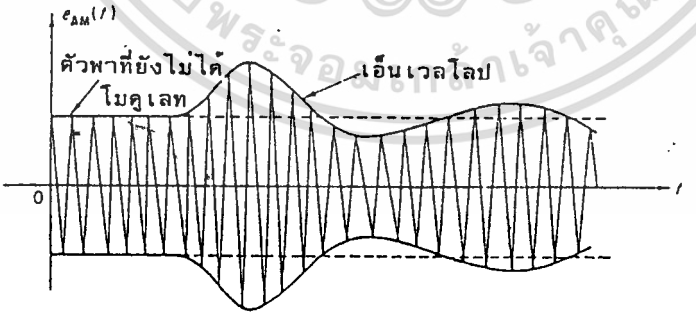
เปอร์เซนต์โมดูเลชั่น เป็นตัวแสดงดรีกรีของแอมปลิจูดโมดูเลชั่น ให้ A_c เป็น ขนาดของตัวพาที่ยังไม่ได้โมดูเลชั่น A_c (max) และ A_c (min) เป็นระดับตัวพาที่มากที่สุด และน้อยที่สุด ถ้าโมดูเลชั่นสมมาตรกัน เปอร์เซนต์การโมดูเลชั่นหาได้จากสูตร

$$P = \frac{A_c(max) - A_c}{A_c} * 100\%$$

$$= \frac{A_c - A_c(min)}{A_c} * 100\% = \frac{A_c(max) - A_c(min)}{2A_c} * 100\%$$



(ก) คลื่นโมดูเลติ่ง



(ข) ตัวพาที่ถูกโมดูเลทแล้ว

รูปที่ 13 รูปคลื่นเบสแบนด์และเอเอ็ม

สเปคตรัมความถี่ของคลื่นเอเอ็มได้จากฟูเรียทรานฟอร์ม ถ้าให้ G(t) เป็น สเปคตรัมความถี่ของสัญญาณเบสแบนด์ g(t) สเปคตรัมของเอเอ็มหาได้ดังนี้

$$e_{AM}(t) <=> E_{AM}(f) = [(f) + m_a G(t)] * [(f-fc)+(f+fc)]$$

$$= 1/2\ [(f-f_c)+(f+f_c)]+m_a/2[G(f-fc)+G(f+fc)]$$

ตัวพา + ไซด์แบนด์

กำลัง(power) ที่ส่วนของตัวพาและไซด์แบนด์ของเอเอ็ม โดยให้สัญญาณเป็น ไซน์เวฟ $g(t)= cos2f_ct$ สัญญาณเอเอ็มจะได้

$$e_{AM}(f)= A_c(1+m_a cos2f_m t)cos2f_c t$$

$$= A_c cos2fct + m_a Ac/2\ [cos2(fc+f)t+cos2(f_c-f_m)t]$$

USB + LSB

ทฤษฎีและการออกแบบ

สำหรับในหัวข้อนี้จะเป็นการหล่าวถึงลักษณะหลักทำงานพื้นฐานต่าง ๆ ตลอดไปจนถึงการ ออกแบบในส่วนทางฮาร์ดแวร์ และก่อนที่จะเริ่มอธิบายหลักการทำงานพื้นฐานและการออกแบบนั้น ควรทำความเข้าในการทำงานของหุ่นยนต์ ที่ถูกออกแบบไว้เสียก่อนมิฉะนั้นแล้วจะไม่เข้าใจในหลัก การออกแบบเลย เพราะการทำงานของมันจะอาศัยการทำงานของหุ่นยนต์ที่ถูกออกแบบมาเป็นหลัก สำหรับ'Continy'นั้นผมได้ออกแบบให้มันมีความสามารถการทำงานดังต่อไปนี้

1.จะปฏิบัติงานตามคำสั่งที่ผู้ป้อนโปรแกรมกำหนดไว้

2.สามารถเปลี่ยนโปรแกรมการทำงานอย่างอัตโนมัติเมื่อมีคำสั่งข้อมูลใหม่เข้ามา

3.มีระบบป้องกันการผิดผลาดของสัญญาณข่าวสารจึงสามารถปฏิบัติตามโปรแกรมที่ ถูกป้อนโดยผู้โปรแกรมได้อย่างถูกต้อง

4.มีระบบตอบรับด้วย Voice Synthesizer คือสามารถส่งเสียงพูดตอบรับการทำ งานของหุ่นยนต์

5.จะไม่ยอมปฏิบัติตามคำสั่งใด ๆ เลยหากไม่มีการส่งสัญญาณความพร้อมจากส่วนควบ คุม(ไม่ได้เปิดใช้งาน Modem)

6.มีระบบ Watch Dog คือป้องกันการเกิด Hank ของไมโครโปรเซสเซอร์ ภายใน หุ่นยนต์

สำหรับการออกแบบนั้นควรเริ่มต้นด้วยการออกแบบกำหนด โปรโตคอล ของการทำงานระหว่าง ส่วนควบคุม (Modem) กับหุ่นยนต์ (Continy) เสียก่อน

การออกแบบโปรโตคอล

สำหรับการออกแบบกำหนดโปรโตคอลที่ใช้ในการสื่อสารนั้นเราจะออกแบบไว้ให้มีลักษณะดังต่อไปนี้

1. รหัสเริ่ม
2. File โปรแกรมคำสั่ง
3. รหัสจบ

รหัสเริ่ม	File ข้อมูล	รหัสจบ

สำหรับการส่งผ่านข้อมูลจะเป็นลักษณะ Synchonouse คือจะมีระยะเวลาการส่งแต่ละ Byte มีคาบเวลาที่แน่นอน ในส่วนของ File คำสั่งต่างๆ นั้นผมได้ออกแบบไว้มีลักษณะโปร โตคอลดังต่อไปนี้

คำสั่งหุ่นยนต์	จำนวนคำสั่ง	รหัสคำสั่ง	
	(BYTE)	HEX	BINARY
เดินหน้า	2	01 xx	0000 0001 xxxx xxxx
เดินหน้าเลี้ยวขวา	2	02 xx	0000 0010 xxxx xxxx
เดินหน้าเลี้ยวซ้าย	2	03 xx	0000 0011 xxxx xxxx
ถอยหลัง	2	04 xx	0000 0100 xxxx xxxx
ถอยหลังเลี้ยวขวา	2	05 xx	0000 0101 xxxx xxxx
ถอยหลังเลี้ยวซ้าย	2	06 xx	0000 0110 xxxx xxxx
ตรวจอุณหภูมิ	1	20	0010 0000
คำสั่งทำงานซ้ำ	1	40	0100 0000
รหัสเริ่ม	1	AA	1010 1010
รหัสจบ	1	EE	1110 1110

XX หมายถึง ระยะทางหรืออาศาที่ถูกโปรแกรม

การออกแบบ Modem

ในส่วน Modem นี้จะเป็นส่วนที่ติดต่อกับ Computer โดยตรง โดยมันจะทำหน้าที่เป็นตัว สื่อกลางระหว่าง Computer กับหุ่นยนต์ ซึ่งภายใน Modem จะประกอบไปด้วยส่วนต่าง ๆ ดังต่อ
ไปนี้

- รับข้อมูลจาก Computer
- แปลงข้อมูลขนาน —>ข้อมูลอนุกรม
- เข้ารหัส (Encode) หรือ Modulating FSK
- Modulator กับสัญญาณ Carrier ส่งออกอากาศ

สำหรับการทำงานของModem นั้นจะเริ่มจากรับข้อมูลที่ถูกโปรแกรมไว้ในลักษณะ Pull Down เมนู จาก ตัวอย่างดังต่อไปนี้

โปรแกรมการควบคุมหุ่นยนต์
เมนูคำสั่ง
1. เดินหน้า xx เมตร
2. เดินหน้าเลี้ยวซ้าย xx องศา (45,90)
3. เดินหน้าเลี้ยวขวา xx องศา (45,90)
4. ถอยหลัง xx เมตร
5. ถอยหลังเลี้ยวขวา xx องศา (45,90)
6. ถอยหลังเลี้ยวซ้าย xx องศา (45,90)
7. ปฏิบัติซ้ำหรือไม่ (Y OR N)
กรุณากดหมายเลขลำดับคำสั่งของหุ่นยนต์
อันดับที่ 1 ?
อันดับที่ 2 ?
อันดับที่ 3 ?
อันดับที่ N ?
ท่านต้องการแก้ไขโปรแกรมหรือไม่ ? (Y OR N)

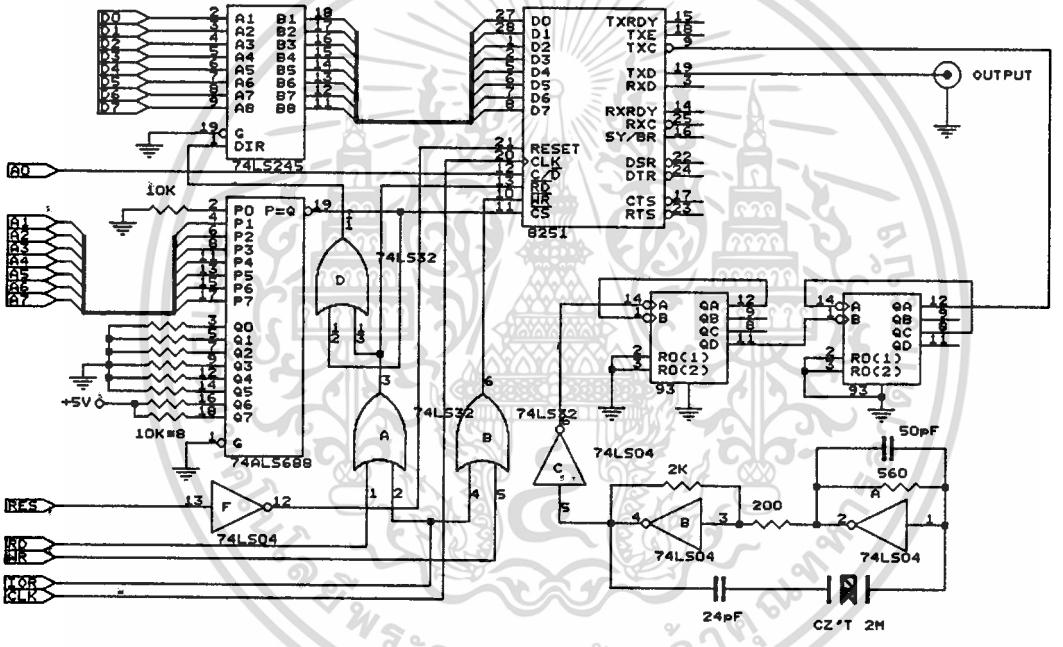
F1 อธิบายคำสั่งและวิธีการใช้	F2เคลียคำสั่งใหม่	F3เลิกการทำงาน

เมื่อมีการโปรแกรมเสร็จ จะมีการส่ง FILE เป็นลักษณะของโปรโคคอล ที่กล่าวไว้ในตอนต้นผ่าน PARALREL PORT ออกสู่ MODEM ต่อไป และ MODEM จะทำหน้าที่จัดทำขบวนการจัด
ส่งข้อมูลเป็นแบบอนุกรมส่งต่อออุกอากาศต่อไป

การออกแบบส่วนรับข้อมูลจาก Computer และแปลงข้อมูล

ในการออกแบบการทำงานในส่วนนี้ได้เลือกใช้ไมโครโปรเซสเซอร์ Z80 รวมกับ 8255 และ 8251เป็นตัวรับข้อมูลและแปลงข้อมูลซึ่ง8255 จะทำหน้าที่ถ่ายโอนข้อมูลแบบขนานมา เก็บไว้ในหน่วยความจำ พร้อมกันนั้นจะแสดงการทำงานโดย LED Minitor ส่วน 8251 จะทำ งานแปลงสัญญาณจากข้อมูลแบบขนานมาเป็นแบบอนุกรม ซึ่งมีการโปรแกรมกำหนดรูปแบบดังต่อไปนี้

1. กำหนดความเร็วในการส่ง Bit rate 300 bps
2. กำหนดความยาวของข้อมูล 8 บิต
3. กำหนดบิตการตรวจสอบและเช็คความถูกต้องของข้อมูล
4. กำหนดจำนวนบิต stop ไว้ที่ 2 บิต



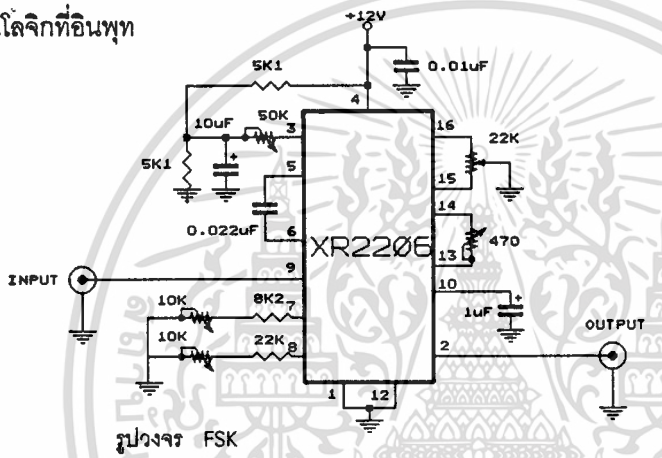
รูปวงจร แสดงการแปลงสัญญาณจากขนานเป็นอนุกรม

การออกแบบส่วนเข้ารหัส (Encode)

ในการออกแบบการทำงานในส่วนนี้จะใช้การเข้ารหัสของสัญญาณที่เป็นแบบ FSK (Freguency Shift Keying)โดยเอาท์พุท จะมีสัญญาณรูปซายน์ซึ่งมีความถี่ขึ้นอยู่กับ สภาวะโลจิกทางด้านอินพุท ซึ่งสามารถแสดงการแทนความถี่ที่ระดับสภาวะโลจิกต่าง ๆ
ได้ดังนี้

สัญญาณดิจิตอล	สัญญาณซายน์
Logic "0"	1200
Logic "1"	2400

การทำงานในส่วนนี้ได้ออกแบบเลือกใช้ไอซีเบอร์ XR-2206 เป็นไอซีที่ถูกออกแบบมาใช้สำหรับการเข้ารหัสลักษณะ FSK โดยเฉพาะ ภายในจะประกอบไปด้วยการทำงานระบบ phase lock Loop ซึ่งจะให้ความถี่ขึ้นอยู่กับสัญญาณโลจิกที่อินพุท

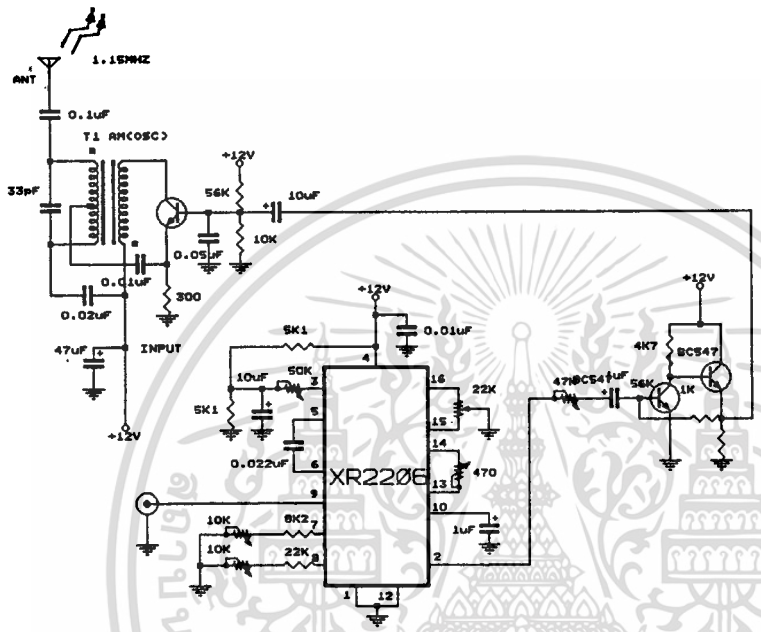


รูปวงจร FSK

การออกแบบส่วน Modulator กับสัญญาณคลื่นพาห์

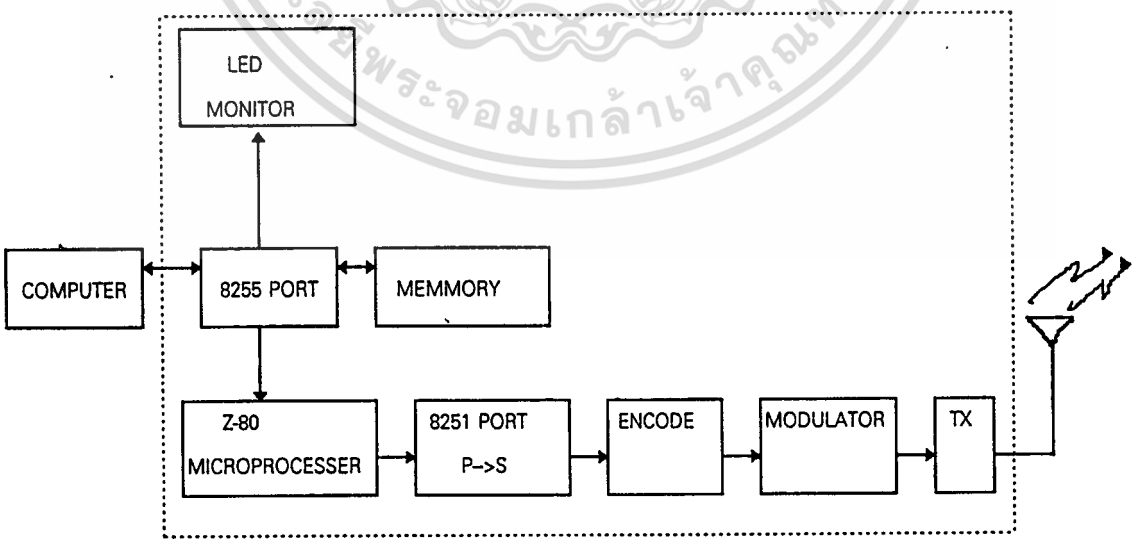
การทำงานในส่วนนี้เป็นส่วนที่จะทำการผสมสัญญาณข่าวสารฝากไปกับคลื่นพาห์ออกอากาศไป สำหรับระยะทางการรับส่งได้ดีแค่ไหน ขึ้นอยู่กับการออกแบบกำลังส่งและระดับ เปอร์เซนต์ ของการ Mod ซึ่งเป็นแบบ AM (Amplitude modlation) สำหรับการออก แบบนั้นจะใช้คลื่นความถี่คลื่นพาห์ประมาณ 1.15 MHz ซึ่งอยู่ในย่าน A.M บ้านเราพอดี เหตุที่ใช้ลักษณะการส่งที่อยู่ในย่านความถี่ต่ำซึ่งอยู่ในย่าน A.Mนี้ก็เพราะว่าต้องการลดอุปสรรคในการออกแบบการทำงานที่ภาครับ ซึ่งสามารถเลือกใช้ไอซีที่มีขายกันมากมายตามท้องตลาด สำหรับในการออกแบบของเรานั้นจะไม่คำนึงถึง SWR เหตุผลที่ไม่คำนึงถึง ค่า SWR ก็คือกำลังส่งของวงจรมีกำลังส่งไม่มากคือประมาณ 10 mw ดังนั้นจึงไม่คำนึง ถึงความเสียหายต่อทรานซิสเตอร์เพาวเวอร์อาร์เอฟ สำหรับระยะทางพื้นที่การับส่งข้อมูล ประมาณ พื้นที่รัศมี 40 เมตร สำหรับการออกแบบของการทำงานในภาคนี้จะใช้หลักการ mod และส่งสัญญาณคลื่นพาห์อย่างง่าย ๆ คือใช้การ Modที่ขา Base ของทรานซิสเตอร์ที่ทำหน้าออสซิเตอร์ซึ่งจะทำให้มีการเปลี่ยนแปลงของรูปคลื่นสัญญาณตามสัญญาณข่าวสารซึ่งมีความถี่คงที่แต่แอมปลิจูดเปลี่ยนแปลง จากวงจรทรานซิสเตอร์เบอร์ BC 1815 จะทำหน้าออสซิเล

เตอร์ความถี่ขึ้นมาโดยทำงานรวมกับ T1และตัวเก็บประจุซึ่งต่ออยู่ในลักษณะฮาร์ดเลย์ออสซิเล เตอร์ สำหรับความถี่ที่ผลิตขึ้นนั้นขึ้นอยู่กับการปรับสลักจูนของ T1 ส่วนทรานซิสเตอร์ BC 547 ทั้งสองตัวจะทำหน้าที่เป็นบัฟเพอร์ของสัญญาณอินพุทและทำหน้าที่ขยายสัญญาณด้วย โดยมี VR ค่า 47K เป็นตัวปรับระดับการ Mod ของสัญญาณ

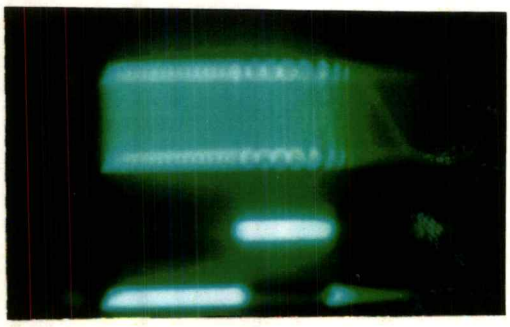


รูปวงจร การ Modulation

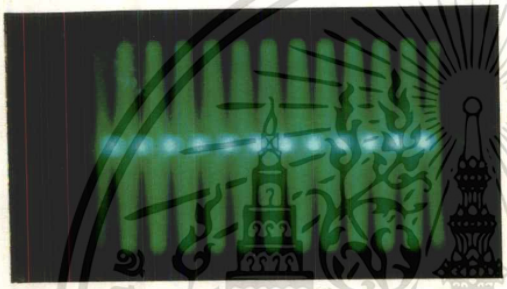
สำหรับ การทำงานต่าง ๆ ของ MODEM สามารถแสดงเป็น Block Diagram ได้ดังต่อไปนี้



ลักษณะรูปสัญญาณต่าง ๆ ของการทำงานแต่ละส่วนของวงจร



รูปแสดงสัญญาณการเข้ารหัสของ FSK



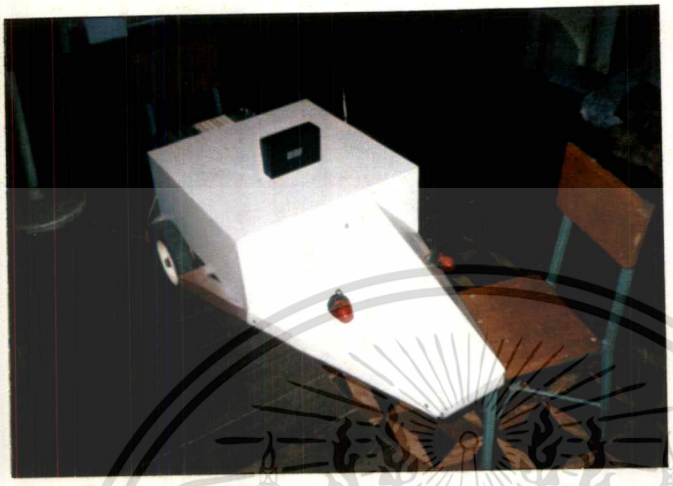
รูปแสดงสัญญาณการ MOD กับสัญญาณคลื่นพาห์

การออกแบบหุ่นยนต์(Continy)

จากที่กล่าวไปข้างต้นแล้วว่า "Continy" สามารถปฏิบัติงานอย่างไรบ้างจะไม่ขอกล่าวซ้ำแต่จะอธิบายถึงการออกแบบส่วนต่าง ๆ เริ่มตั้งแต่การออกแบบแมกคานิกส์ ไปจนถึง การออกแบบฮาร์ดแวร์และการอินเตอร์เฟส ตามลำดับ

การออกแบบแมกคานิกส์

ในส่วนนี้การออกแบบแมคคานิกส์นั้นเราจะไม่เน้นสักเท่าไรนักแต่จะไปเน้นทางระบบควบคุม, ระบบอิเล็กทรอนิกส์ ,ระบบการสื่อสารและคอมพิวเตอร์เป็นหลัก ลักษณะของ หุ่นยนต์(Continy) ได้มีการออกแบบไว้ดังนี้ คือสามารถเคลื่อนไปทางใดก็ได้ซึ่งจะมีระบบ ตรวจจับ (senser) ระยะทางและการเคลื่อนที่ สำหรับรูปร่างของหุ่นยนต์(continy) แสดงโดยดังรูป ซึ่งการขับเคลื่อนจะติดระบบเฟืองทดสามารถขับเคลื่อนบรรทุกส่งของได้ ไม่เกิน 60 kg โดยขึ้นอยู่กับสภาพแบตเตอร์รี่



รูปแสดงหน้าตาของหุ่น

การออกแบบทางด้านฮาร์แวร์และการอินเตอร์เฟส

สำหรับการออกแบบในส่วนนี้จะเป็นการออกแบบระบบ Senser และควบคุมการทำงาน ต่างๆ ภายในหุ่นยนต์ (Continy) ซึ่งมีการทำงานสามารถแบ่งแยกเป็นส่วน ๆ ได้ดังต่อไปนี้

ส่วนภาครับสัญญาณคลื่นวิทยุ

ส่วนกรองสัญญาณข่าวสารและกู้สัญญาณข่าวสาร

ส่วนของการถอดรหัสสัญญาณ (Decode)

ส่วนของการแปลงข้อมูลจากอนุกรมไปเป็นขนาน

ส่วนควบคุมการทำงาน

ส่วนขับเคลื่อนและ Senser ระยะทางการเคลื่อนที่

ส่วนตรวจจับอุณหภูมิและแสดงผล

ส่วน Voice Synthesizer

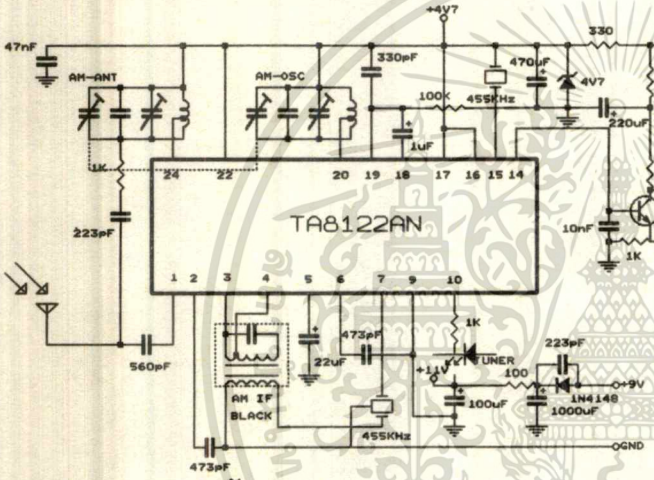
ส่วนของวงจร Wacth Dog

ส่วนของวงจรจับ Sync

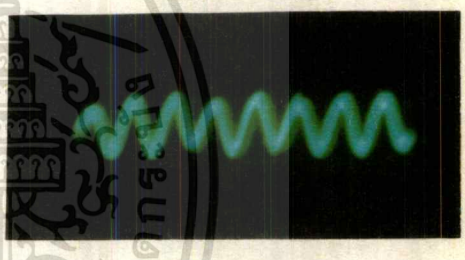
ส่วนจ่ายกำลัง

การออกแบบในส่วนภาครับสัญญาณคลื่นวิทยุ

การทำงานในส่วนนี้เป็นส่วนเริ่มการทำงานของหุ่นยนต์เลยทีเดียวเพราะคำสั่งต่างๆ ที่ถูกส่งมานั้นจะเป็นคลื่นแม่เหล็กไฟฟ้า ดังนั้นภาครับในส่วนนี้จะทำหน้าที่แปลงข่าวสารมาให้ อยู่ในรูปของคลื่นไฟฟ้าฉะนั้นการออกแบบการทำงานในส่วนต่างๆ นั้นจำเป็นต้องเลือกค่า การใช้งานให้มีเสถียรภาพมากที่สุดสำหรับการออกแบบการทำงานภาคนี้ ได้เลือกใช้ไอซีสำเร็จรูปเบอร์ TA 8122 N ซึ่งเป็นไอซีที่ภายในมีวงจรภาครับเอฟเอ็มและเอเอ็มในตัว เสร็จสรรพ ซึ่งใช้อุปกรณ์ต่อร่วมน้อย คุณสมบัติของค่าS/N ก็มีค่ามากกว่าเบอร์อื่นๆ ถ้าเทียบตระกูลไอซีที่ทำงานในภาคนี้แล้ว โดยในระดับราคาที่ใกล้เคียงกันเมื่อดูจากลักษณะของวงจร แล้วจะเห็นว่าอุปรกณ์ภายนอกที่นำมาต่อมีจำนวนน้อย สำหรับสัญญาณที่ออกมาจะมีกำลังประมาณ 0.005 w ซึ่งก็เพียงพอสำหรับการใช้งานซึ่งสามารถต่อเข้ากับวงจรได้เลยแต่สำหรับการใช้งาน แล้วเราจะออกแบบให้มีภาคบัฟเฟอร์เพิ่มเข้าไปอีก เพื่อให้วงจรมีเสถียรภาพที่ดีขึ้น



วงจร ภาครับ

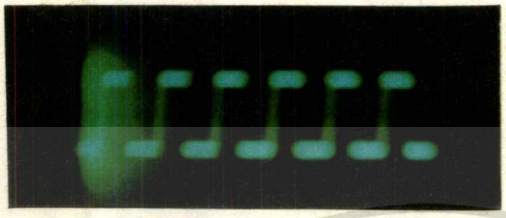


สัญญาณที่วัดได้จากออสซิลโลสโคป

การออกแบบส่วนกรองสัญญาณและกู้สัญญาณข่าวสาร

การออกแบบในส่วนภาคนี้ขึ้นมาก็เพื่อที่จะให้วงจรนี้มีเสถียรภาพการทำงานที่ดีขึ้น คือจะยอมให้สัญญาณข่าวสารเท่านั้น สามารถผ่านไปได้ ซึ่งสัญญาณที่ได้นั้นจะมีลักษณะเป็นสัญญาณซายน์บริสุทธิ์และเพื่อป้องกันการรบกวนของคลื่นแม่เหล็กที่อาจแฝงมากับสัญญาณข่าวสาร ขณะที่มอเตอร์กำลังหมุน สัญญาณที่ได้จากวงจรกรองสัญญาณจะถูกป้อนเข้าสู่วงจรกู้สัญญาณ ข่าวสารอีกครั้งหนึ่ง เพื่อให้ได้สัญญาณที่มีคาบเวลาของการสถานะคงที่ที่ความถี่ค่าหนึ่งซึ่งเป็น พัลซ์ระดับแรงดัน ดี ซี ซึ่งง่ายต่อการถอดรหัส สำหรับการออกแบบวงจรในส่วนนี้จะใช้เลือกใช้ไอซีเบอร์ LM 3900 N ซึ่งเป็นไอซีนอร์ตันออปแอมป์ ซึ่งเหมาะสมที่ใช้แหล่งพลังงานจากแบตเตอรี่ เพราะว่าถูกออกแบบมาสำหรับใช้แหล่งจ่ายแรงไฟทางเดียวคือ บวก กับกราวด์ การออกแบบในส่วนกรองสัญญาณนั้นจะใช้วงจร Filter แบบ 4 State เพื่อให้วงจรมี เสถียรภาพดีในการป้องกัน

สัญญาณรบกวน ในส่วนของวงจรกู้สัญญาณนั้นจะใช้การออกแบบลักษณะของวงจร Comparator ซึ่งจะให้สัญญาณ Output ลักษณะสัญญาณสี่เหลี่ยม

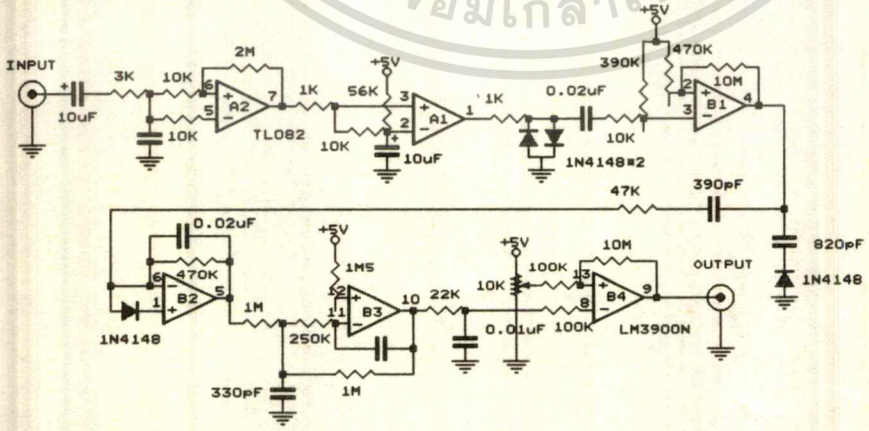


สัญญาณที่วัดได้จากออสซิลโลสโคป

การออกแบบส่วนของการถอดรหัสสัญญาณ (Decode)

หลักการออกแบบการทำงานของภาคนี้ผมได้ใช้หลักการของวงจรดังแสดงดังรูปต่อไปนี้

คือสัญญาณที่ถูกรับข่าวมาจะผ่านวงจร Fto V ซึ่งจะทำหน้าที่เปลี่ยนความถี่ให้เป็นแรงดันถ้าความถี่มากแรงดันยิ่งสูงถ้าความถี่น้อยแรงดันก็จะน้อย สัญญาณที่ได้ก็จะถูกนำไปเปรียบเทียบและเข้าวงจร Smittriger ต่อไป ซึ่งจะเป็นข้อมูลข่าวสารที่แท้จริง สำหรับการออกแบบนั้นได้เลือกใช้ ไอซี LM 3900 N อีกเช่นเคยเพราะเป็นออปแอมป์ไฟทางเดียว ที่ออกแบบมาให้สามารถทำงานได้อเนกประสงค์โดยออปแอมป์แต่ละตัวจะแทนการทำงานแต่ละ Block ของวงจรดังนี้

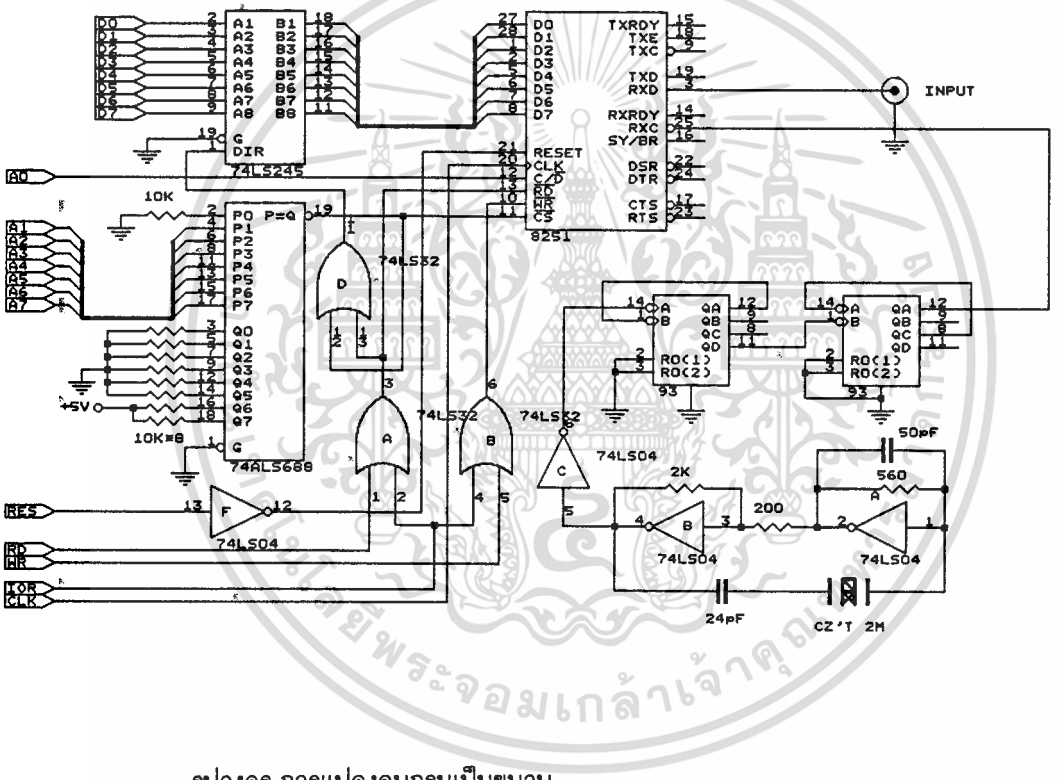


วงจร ถอดรหัสสัญญาณ (decode)

การออกแบบส่วนของการเปลี่ยนข้อมูลอนุกรมมาเป็นขนาน

การออกแบบในส่วนนี้ก็จะเป็นลักษณะเช่นเดียวกับในส่วนของการออกแบบ Modem ที่ 8251 คือ

1. กำหนดความเร็วในการส่ง Bit rate 300 bps
2. กำหนดความยาวของข้อมูล 8 บิท
3. กำหนดการตรวจสอบและเช็คความถูกต้องของข้อมูล
4. กำหนดจำนวนบิท stop ไว้ที่ 2 บิท

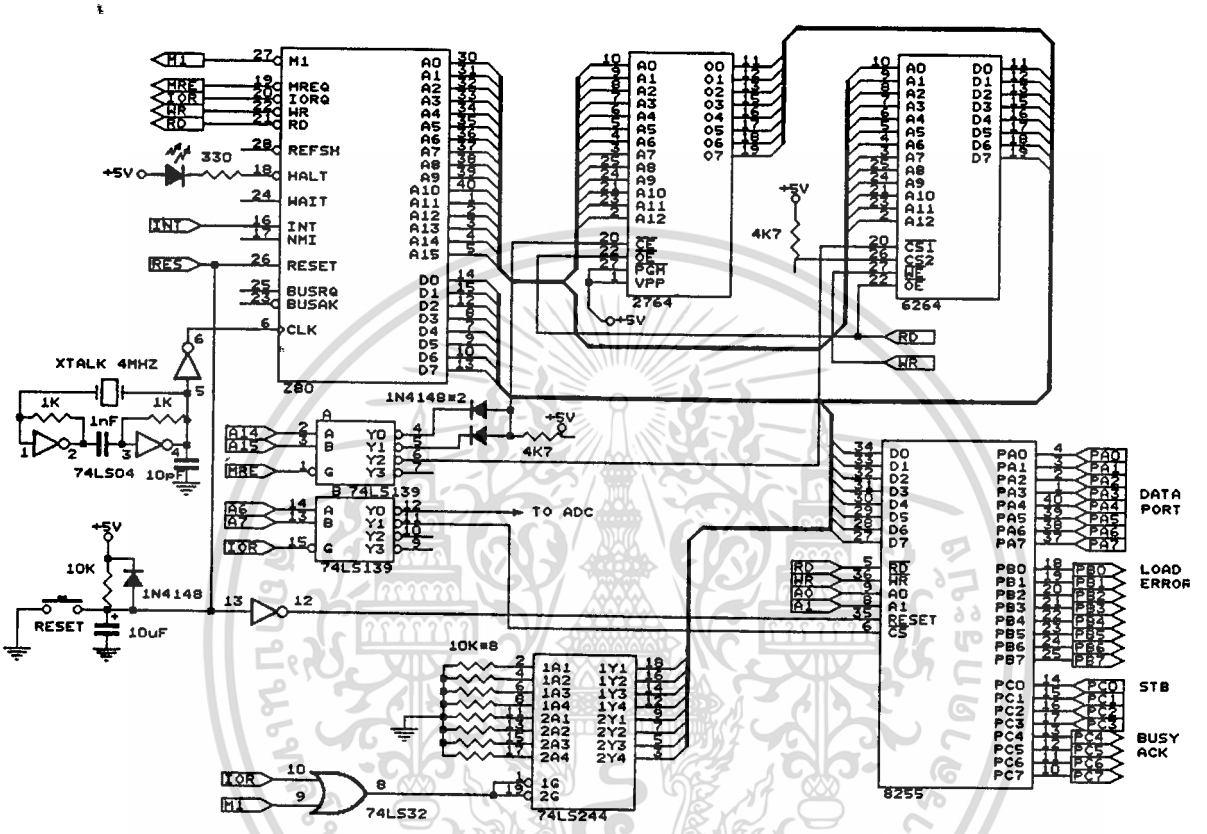


รูปวงจร การแปลงอนุกรมเป็นขนาน

การออกแบบส่วนของควบคุมการทำงาน

การควบคุมการทำงานนั้นได้เลือกใช้ไมโครโปรเซสเซอร์ Z80 มาเป็นตัวควบคุมการทำงานของหุ่นยนต์ (Continy) สำหรับอุปกรณ์เชื่อมต่อเพื่อใช้ 8255 เป็นตัวต่อเชื่อม

ซึ่งจะควบคุมการทำงานทุกส่วนในตัวหุ่นยนต์(Continy)



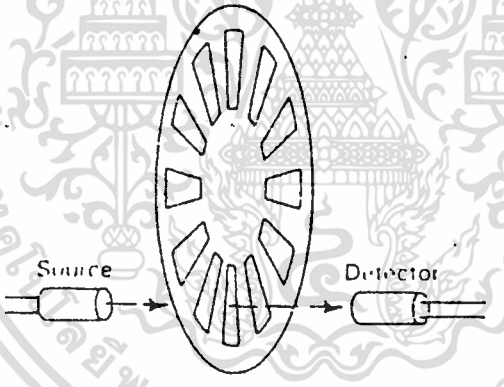
รูปวงจร ควบคุมการทำงาน

การออกแบบส่วนขับเคลื่อนและ Senser ระยะทางการเคลื่อนที่

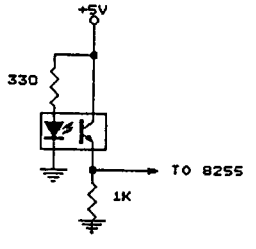
สำหรับการออกแบบอินเตอร์เฟสในส่วนนี้จะแบ่งเป็นสองส่วน คือ ส่วนขับเคลื่อนและส่วนตรวจจับระยะทาง (Senser) ในส่วนของการขับเคลื่อนนั้นจะอาศัยการทำงานของมอเตอร์ โดยควบคุมการทำงานและทิศทางการเคลื่อนที่ของมอเตอร์ ทั้งมอเตอร์ใน ส่วนขับเคลื่อนทางด้านซ้ายและในส่วนขับเคลื่อนมอเตอร์ด้าขวา ซึ่งสามารถอธิบายลักษณะ การควบคุมการคลื่อนที่ได้ดังต่อไปนี้

คำสั่ง	มอเตอร์ขับเคลื่อนด้านซ้าย	มอเตอร์ขับเคลื่อนด้านขวา
เดินหน้า	หมุนขวา	หมุนขวา
เดินหน้าเลี้ยวขวา	หมุนขวา	หยุด
เดินหน้าเลี้ยวซ้าย	หยุด	หมุนขวา
ถอยหลัง	หมุนซ้าย	หมุนซ้าย
ถอยหลังเลี้ยวขวา	หมุนซ้าย	หยุด
ถอยหลังเลี้ยวซ้าย	หยุด	หมุนซ้าย

ส่วนการออกแบบการตรวจจับ (Senser) ระยะทางการเคลื่อนที่นั้นจะอาศัยการออกแบบการนับจำนวนฟัลซ์ของการตรวจจับรูผ่านแสงเล็กๆ ที่ติดกับเพลาล้อ ซึ่ง 1 ฟัลซ์
ของสัญญาณที่ผ่านรูของแสงจะมีค่าเท่ากับประมาณ 2.5 เซนติเมตร การออกแบบการตรวจจับของรูผ่านแสงนั้นได้อาศัยการทำงานของออปเตอร์คับเปิ้ลมาใช้ซึ่งแสดงการติดตั้งและการ ทำงานดังนี้

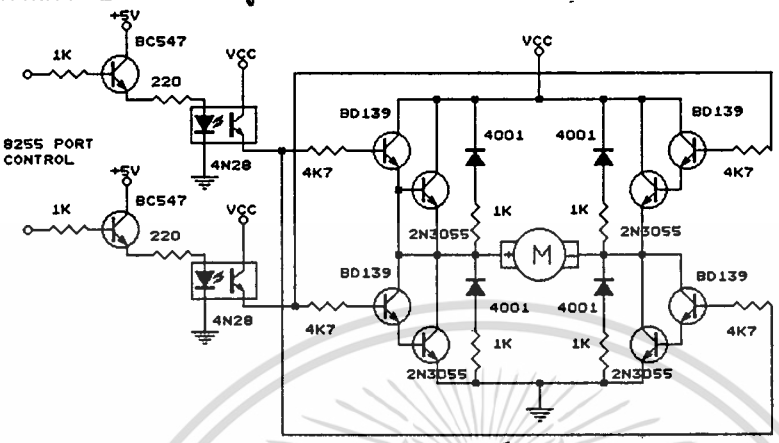


รูปลักษณะการติดตั้งจริง



รูปแสดงวงจร Sensor

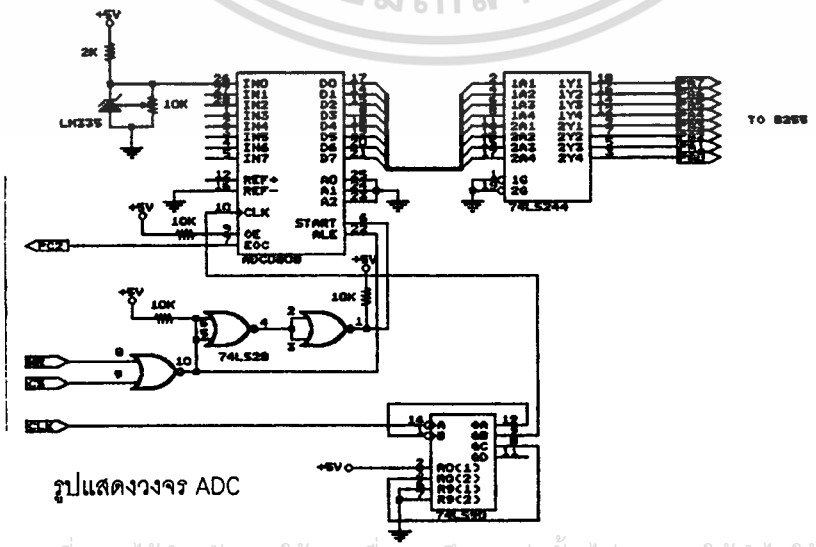
การออกแบบการขับเคลื่อนนั้นเราได้เลือกใช้มอเตอร์ชนิด ดี ซี มอเตอร์ซึ่งมีข้อดีกว่า Stepping Moter ที่ว่าสามารถรับโหลดที่มีค่ามาก ๆ ได้ สำหรับการควบคุมนั้นๆ จะอาศัย ออปโต้คับเปิ้ลมาเป็นตัวต่อเชื่อม คือ แยกระดับศักดิ์ไฟด้านต่ำ กับ ศักดิ์ไฟด้านสูงออกจากกันส่วนลักษณะการทำงานนั้นจะอาศัย 8255 มาเป็นตัวควบคุม



รูปแสดงวงจร ควบคุมมอเตอร์

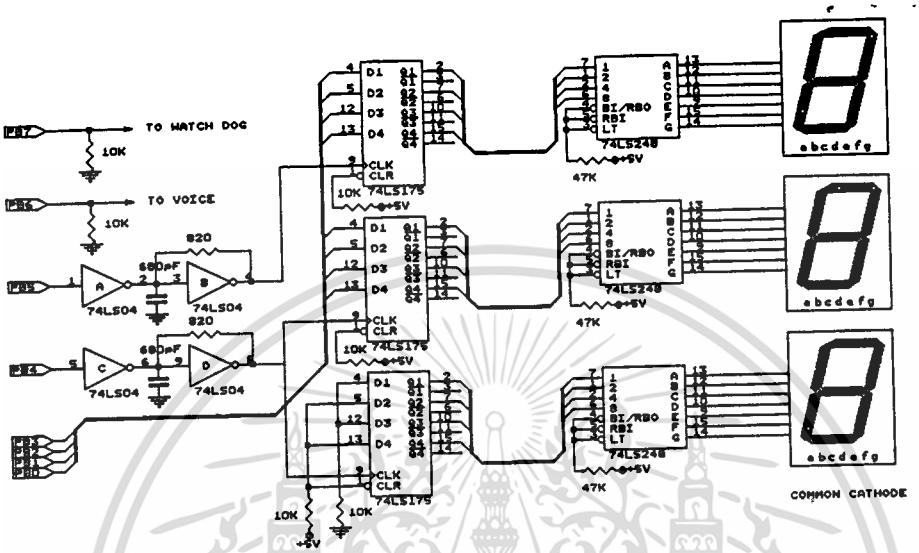
การออกแบบส่วนตรวจวัดอุณภูมิและแสดงผล

การออกแบบในส่วนนี้จะอาศัยอุปกรณ์ที่เรียกว่า ทรานสดิ้วเซอร์ คือจะทำหน้าที่เปลี่ยนค่าระดับอุณหภูมิมาเป็นแรงดัน สำหรับอุปกรณ์ทรานสดิวเตอร์นี้ได้เลือกใช้ไอซีสำเร็จรูปที่ออกแบบมาทำงานในด้านนี้โดยเฉพาะคือเบอร์ LM 335z คือมันจะให้ระดับแรงดันที่เอาท์พุท 10 mv/องศา K สำหรับในส่วนที่ทำหน้าที่เปลี่ยสัญญาณอนาล็อกมาเป็นสัญญาณดิจิตอลนั้นจะออกแบบชนิดให้มีค่าความละเอียดประมาณ $2^7 = 256$ ระดับซึ่งได้เลือกใช้ไอซีเบอร์ ADC 0808 จัดได้ว่าเป็นไอซีที่ถูกออกแบบมาให้มาทำงานด้านนี้โดยเฉพาะซึ่งระดับสภาวะโลจิกไบนารีทางด้านเอาท์พุทจะขึ้นอยู่กับค่าแรงดันทางด้านอินพุทที่รับเข้ามา สำหรับ ไอซีเบอร์นี้นั้นสามารถรับค่าสภาวะทางอินพุทได้ จำนวน 8 ช่องสัญญาณ แต่สำหรับการออก แบบนั้นจะเลือกใช้เพียงช่องเดียว สำหรับวงจรใช้งานแสดงดังรูป



รูปแสดงวงจร ADC

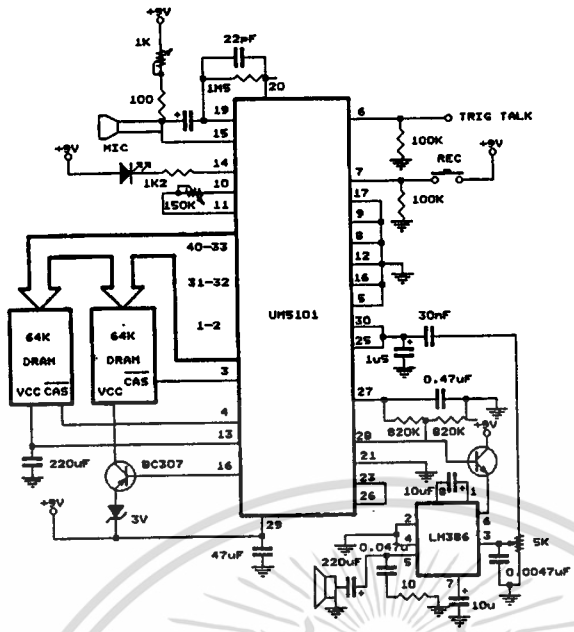
สำหรับในส่วนแสดงผลนั้นได้ออกแบบชนิดที่มีการ Lacth ข้อมูลค้างไว้เมื่อมันไปทำงานในส่วนอื่น ซึ่งข้อมูลจะไม่สูญหายจนกว่าจะมีคำสั่งที่ถูกโปรแกรมใหม่เข้ามา



รูปแสดงวงจร การแสดงผล

การออกแบบส่วน Voice Syrthesizer

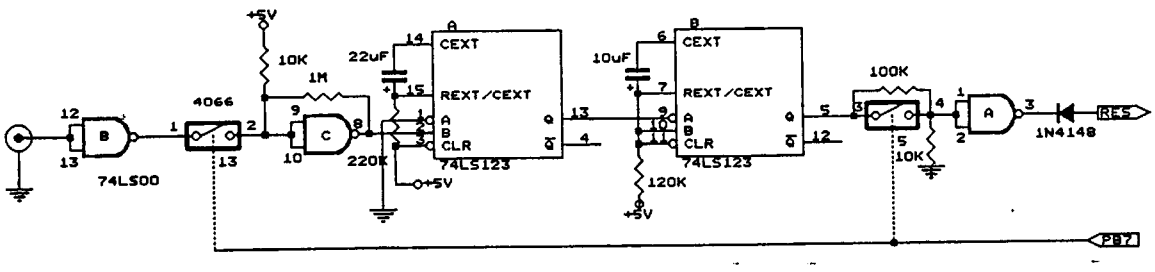
เหตุที่ออกแบบการทำงานในส่วนนี้ขึ้นมาก็เพื่อเป็นการตอบรับจากหุ่นยนต์ (Continy) ว่าได้รับข้อมูลโปรแกรมคำสั่งไว้แล้วพร้อมที่จะทำงาน การทำงานในส่วนนี้ได้ออกแบบลักษณะง่ายๆ คือ ใช้ไอซี สำเร็จรูปเบอร์ UM5101 ซึ่งตัวมันเองจะทำงานลักษณะไอซีตระกูล Voice Recording & Reproducing Voice ซึ่งการทำงานของมันจะมีวงจร Samping และผ่านขบวนการ ADC,Fitter และDAC ซึ่งอาศัยข้อมูลจากหน่วยความจำ Dinamic Ramภายในวงจรเป็นตัวถ่ายทอดและบันทึกสัญญาณเอาไว้ สำหรับการทำงานนั้นผมได้ออกแบบให้การควบคุมการเล่นการเพียงครั้งเดียวจาก ไมโครโปรเซสเซอร์ ผ่าน 8255 สำหรับข้อมูลที่ จะถูกถ่ายทอดออกมานั้นจะต้องถูกบันทึกไว้ก่อนโดยการTrig การทำงานที่ขา Rec



รูปวงจร Voice Synthesize

การออกแบบส่วนของวงจร Wacth Dog

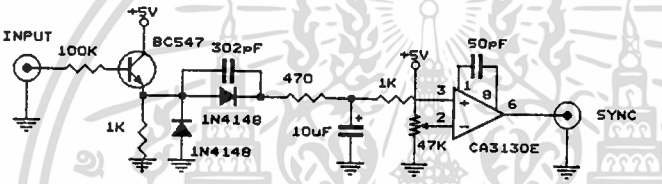
เหตุผลที่ต้องการออกแบบวงจรนี้ขึ้นมาก็เพื่อให้หุ่นยนต์(Continy)มีเสถียรภาพการทำงานให้ดียิ่งขึ้น ซึ่งจะป้องกันการเกิดสภาวะทำงานผิดผลาด (Hank) เมื่อหุ่นยนต์อยู่ในลูปของการรับข้อมูลจากโมเดม ซึ่งทุก ๆครั้งเมื่อมีข้อมูลส่งมาวงจรจะทำงานอัตโนมัติ โดยวงจรนี้จะ คอยทำการตรวจเช็คมีข้อมูลป้อนเข้ามา จากนั้นมันจะทำการตรวจจับว่าจำนวนข้อมูลที่ส่งออก มาหมดหรือยัง ถ้าหากว่าข้อมูลถูกส่งออกมาหมดแล้วแต่ไมโครโปรเซสเซอร์ยังคงหลงอยู่ในLoop การรับข้อมูลอยู่ วงจร Wacth Dog นี้จะไปรีเซตการทำงานของไมโครเซสเซอร์ให้เริ่มทำ งานใหม่ทันที สำหรับการออกแบบในส่วนนี้จะอาศัยการเปลี่ยนแปลงของพัลซซึ่งจะจับเวลา การทำงานของไมโครโปรเซสเซอร์ หากไมโครโปรเซสเซอร์ทำงานช้ากับเวลาที่ตั้งเอาไว้ วงจร Wacth Dog จะทำงานทันที



รูปแสดงวงจร Wacth Dog Time

การออกแบบในส่วนจับ Sync

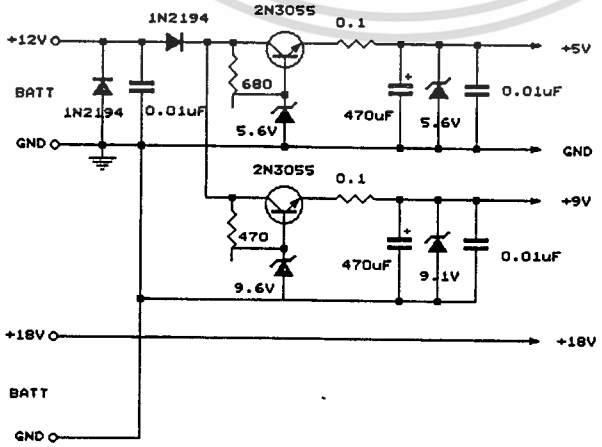
การออกแบบในส่วนนี้เป็นการออกแบบเพื่อให้หุ่นยนต์ (Continy) สามารถรับรู้ได้ว่ามีการเปิดเครื่องการทำงานที่ส่วนควบคุมหรือไม่ ซึ่งจะเป็นการป้องกันการสูญเสียเวลาในการตรวจจับข้อมูลว่าถูกส่งมาหรือยัง ซึ่งหุ่นยนต์อาจเอาเวลาว่าง ๆ นี้ทำงานอะไรไปก่อน ก็ได้ สำหรับการออกแบบในส่วนนี้จะอาศัยการถอดรหัสของวงจรถอดรหัส คือ ถ้าหากที่ Modem มีการเปิดเครื่องใช้งานมันจะทำการส่งสัญญาณ Sync ทันที ซึ่งก็คือสภาวะโลจิก "1" คงที่ ดังนั้นเราจึงอาศัยหลักการนี้เป็นตัวออกแบบการทำงานในส่วนนี้ และห้าหากยังไม่เปิดเครื่องจะมีแรงดันพัลซ์ขึ้นลงไปมาและจะถูกอินติเกรตโดยตัวเก็บประจุ 10F ซึ่งระดับ แรงดันจะไม่เกินระดับ Threshold ที่ตั้งไว้ถ้าเกิดเกินระดับ Threshold ภายในซอฟ แวร์ของการทำงานหุ่นยนต์ก็จะมีการตรวจสอบเช่นกันว่ามีสัญญาณ Sync ส่งมาหรือไม่



รูป วงจรจับ Sync

การออกแบบส่วนจ่ายกำลัง

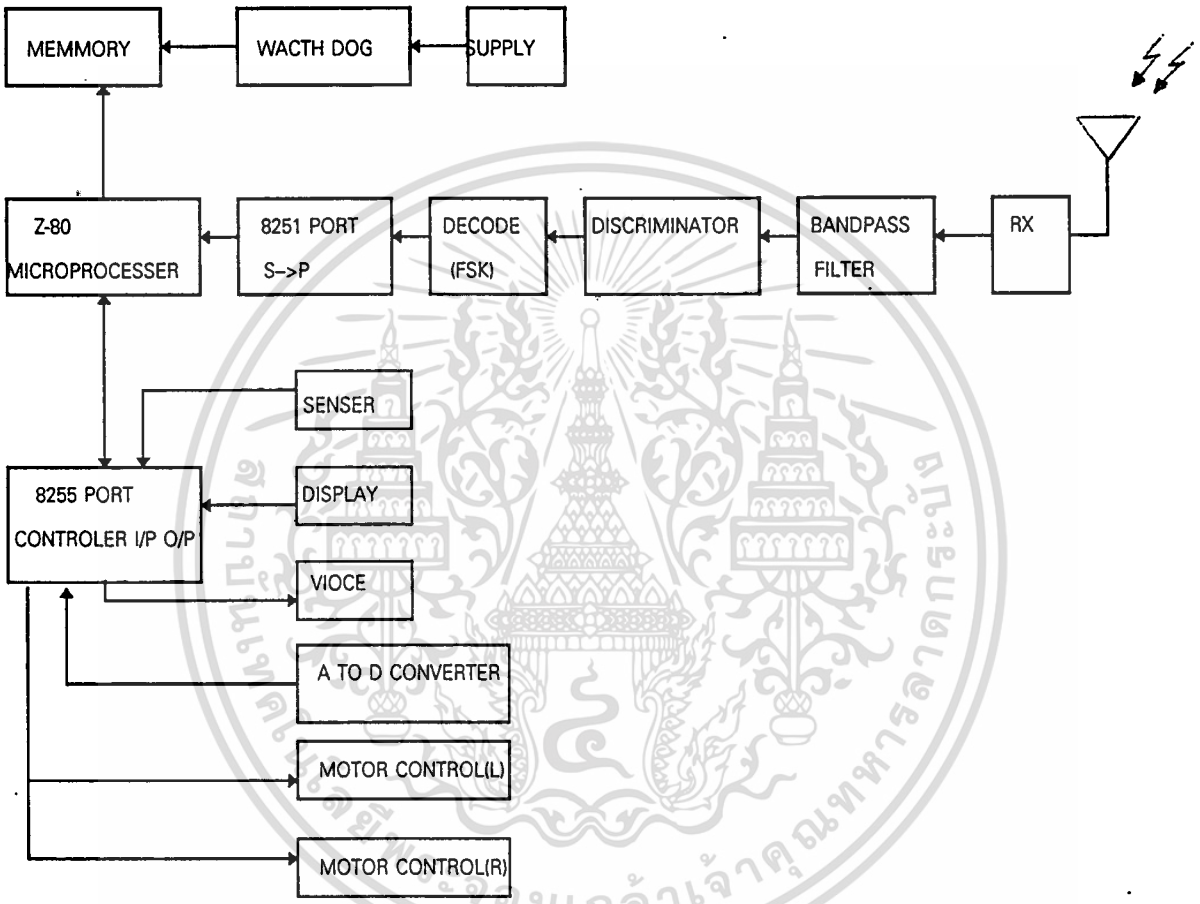
สำหรับการออกแบบในส่วนนี้จะใช้ระดับแรงดัน สองช่วงคือ ระดับแรงดันช่วงต่ำ 5 โวทล์ และ 9 โวทล์ ระดับแรงดันไฟสูงด้วยคือ 18 V. และเพื่อป้องกันการกระชาก ของการทำงานของหุ่นยนต์อันเกิดจากช่วงการทำงานของมอเตอร์ ซึ่งขณะ Start จะมีการ กระชากไฟ ซึ่งถ้าเราใช้แหล่งจ่ายไปรวมกันแล้วนำมาลดไฟลงไปเลี้ยงไฟด้านต่ำจะทำให้ วงจรควบคุมทำงาน ทำงานผิดพลาด ดังนั้นเราจึงได้แยกแหล่งจ่ายไฟออกเป็น สองส่วน คือ ใช้แบตเตอร์รี่ 2 ลูก ชุดหนึ่งเลี้ยงมอเตอร์ด้านไฟสูง และอีกชุดหนึ่งด้านไฟต่ำ ซึ่งแสดงวงจร การออกแบบดังนี้



รูป แสดงวงจร จ่ายกำลัง

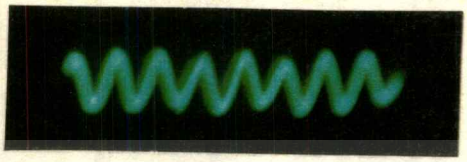
รูป แสดงวงจร จ่ายกำลัง

และจากการออกแบบส่วนต่าง ๆ สามารถนำมาเขียนลักษณะการทำงานเป็นบล็อกๆ ได้ดังต่อไปนี้



แสดงบล็อกไดอะแกรมการทำงานของหุ่นยนต์

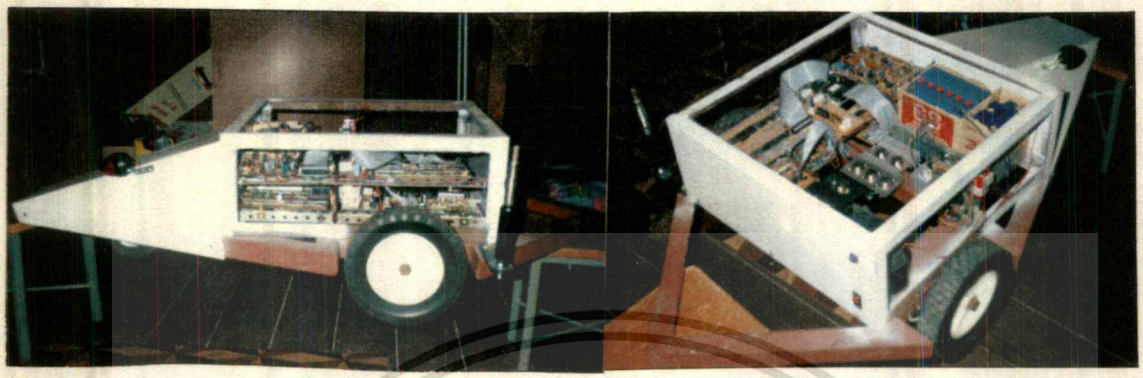
ลักษณะรูปสัญญาณจากการทำงานส่วนต่าง ๆ



รูปแสดงสัญญาณที่ได้จากภาครับ

รูปแสดงสัญญาณผ่านวงจร filter

รูปแสดงสัญญาณที่ผ่านวงจรกู้สัญญาณข่าวสาร

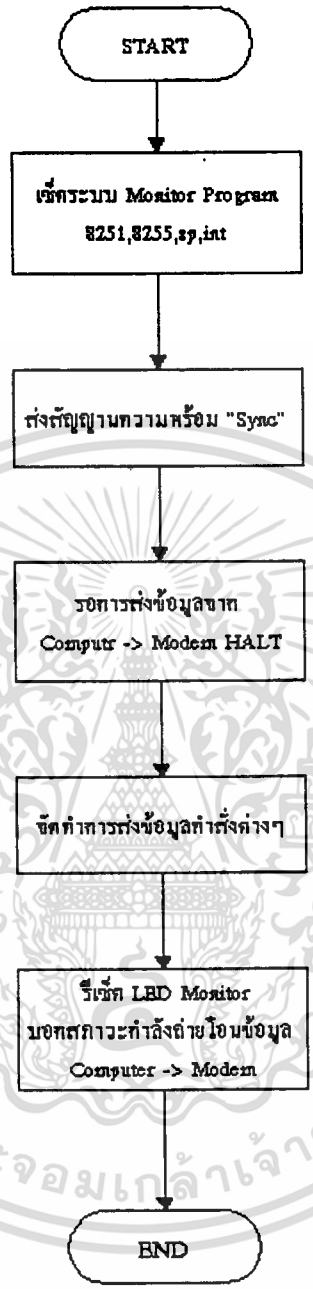


รูปถ่ายชิ้นงานที่ประกอบเสร็จ

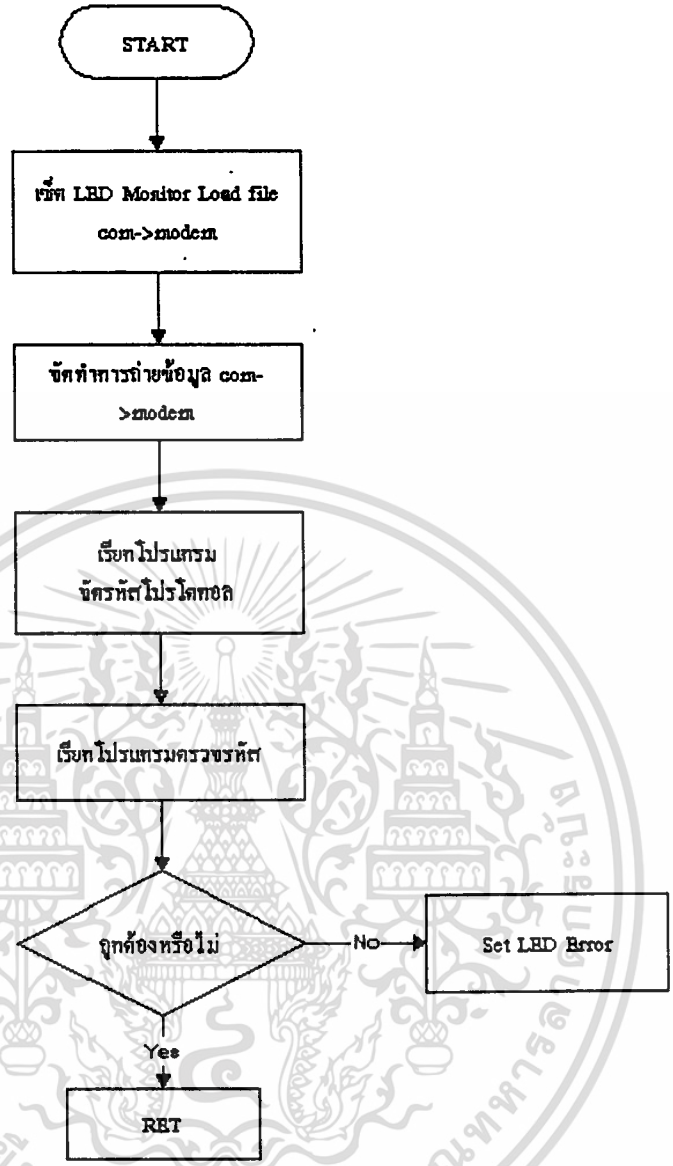
โปรแกรมควบคุมการทำงาน

ส่วนควบคุมการทำงาน(Modem)

ลักษณะของโปรแกรมควบคุมในส่วนนี้มีอยู่ 2 ลักษณะคือโปรแกรมหลักและโปรแกรมบริการอินเตอร์รัพท์ ซึ่งจะแสดง Flow Chart'การทำงานดังต่อไปนี้



แสดงลักษณะการทำงานโปรแกรมหลักของ Modem



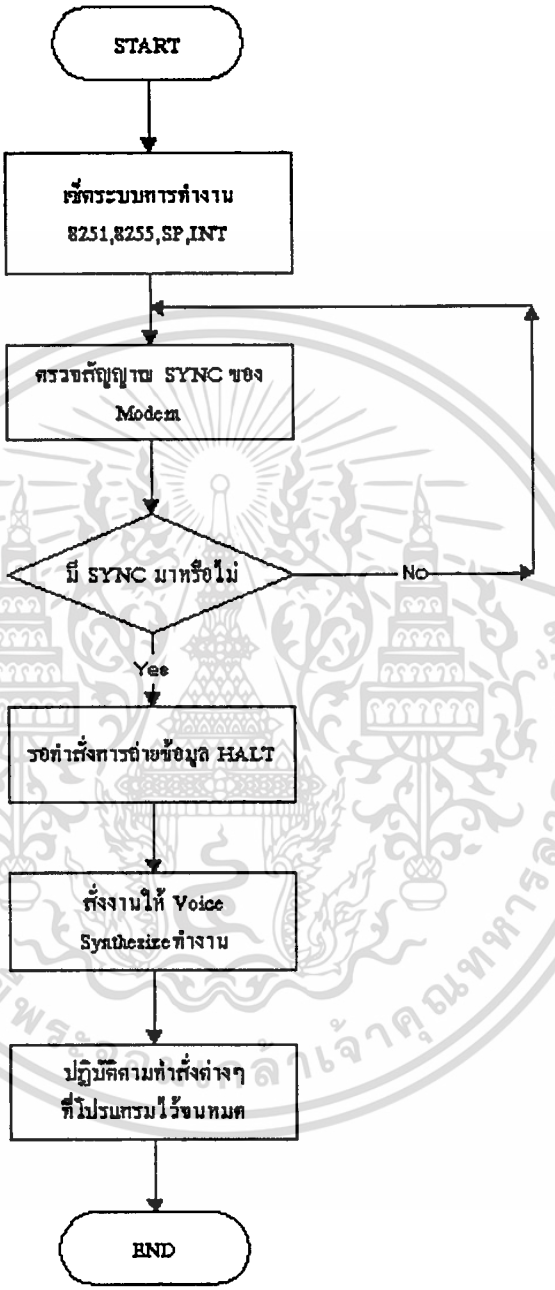
แสดงลักษณะการทำงานโปรแกรมบริการอินเตอร์รัพท์ของ Modem

สำหรับการทำงานของโปรแกรม Modem สามารถอธิบายได้ดังนี้ คือ เริ่มต้นด้วยไมโครโปรเซสเซอร์จะทำการเซตระบบการทำงานต่างๆ เช่น กำหนดค่าสแตก, กำหนดโหมด การทำงาน 8255,8251 เมื่อทำการเซตระบบการทำงานเรียบร้อยแล้ว Modem พร้อมจะส่งสัญญาณ Sync ไปให้หุ่นยนต์(Continy) เพื่อเป็นการบอกให้

ทราบว่า Modem พร้อมจะส่งสัญญาณแล้วให้หุ่นยนต์เตรียมพร้อมการทำงาน ทันทีเมื่อ Modem ส่งสัญญาณออกไปจากนั้น Modem จะอยู่ในสภาวะรอการถ่ายโอนข้อมูลจาก Computer และทันทีเมื่อเกิดการอินเตอร์รัพท์ คือมีคำสั่งโปรแกรมต่าง ๆ ถูกส่งมาจาก Computer Modem จะทำการถ่ายโอนข้อ มูลจาก Port ขนานที่ต่ออยู่กับ Computer ขณะเดียวกันไมโครโปสเซอร์จะทำการเซต LED Monitor บอกสภาวะว่ากำลังถ่ายโอนข้อมูล ข้อมูลต่าง ๆ จะถูกนำมาเก็บไว้ในหน่วยความจำเมื่อเสร็จขบวนการถ่ายโอนข้อมูลจาก Computer มายัง Modem เสร็จสิ้นลง Modem จะทำการนำข้อมูลเหล่านั้นมาทำการใส่ รหัส โปรโคคอลต่างๆ ที่ได้ออกแบบในตอนต้น จากนั้น จะนำ ข้อมูลไปตรวจความถูกต้องของรหัสคำสั่งต่างๆ ที่ได้ออกแบบไว้ ถ้าข้อมูลเกิดการผิดผลาด Modem จะแสดง LED Monitor บอกสภาวะ Error ให้ผู้ใช้ทำการรีเซตเครื่องให้ทำงานใหม่แต่ถ้าข้อ มูลถูกต้องโปรแกรมจะออกจากโปรแกรมบริการอินเตอร์รัพท์ จากนั้นจะกระโดดมาทำงานโปร แกรมหลักโดยจะทำการส่งสัญญาณข้อมูลเสร็จ ไมโครโปรเซสเซอร์จะทำการรีเซต LED Monitor การถ่ายโอนข้อมูลและกลับสู่การส่งสัญญาณ SYNC อีกครั้งหนึ่ง

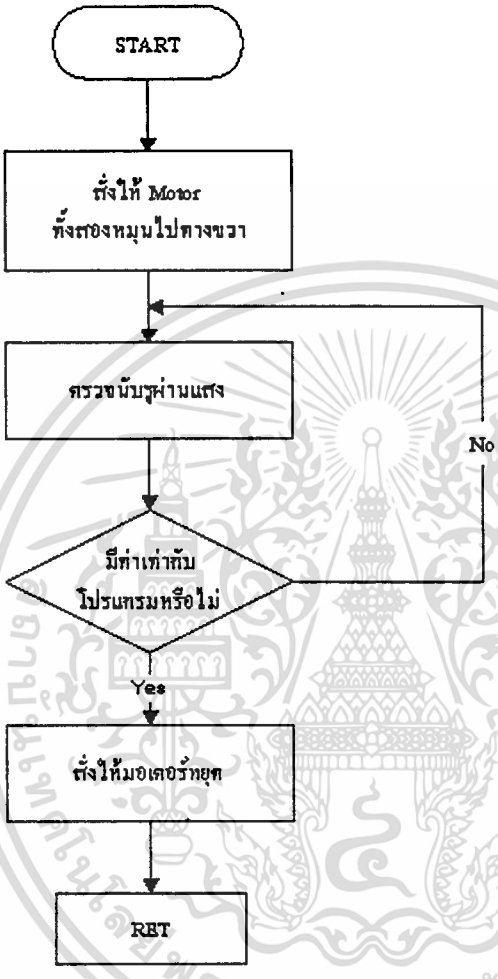
ส่วนการทำงานของหุ่นยนต์(Continy)

สำหรับโปรแกรมควบคุมการทำงานของหุ่นยนต์นั้นแบ่งได้เป็นสองลักษณะเช่นเดียวกันคือ โปรแกรมหลัก และโปรแกรมบริการ อินเตอร์รัพท์ โดยมีลักษณะโปรแกรมต่างๆดังต่อไปนี้

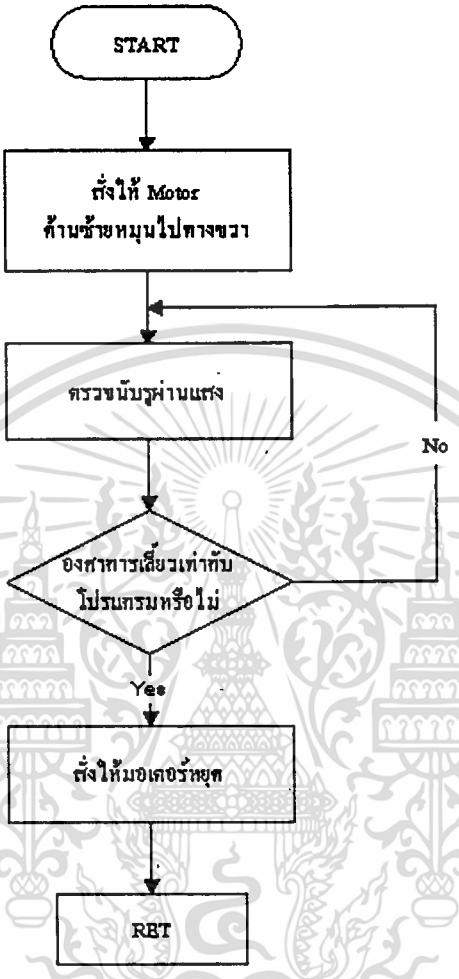


การทำงานของโปรแกรมหลักของหุ่นยนต์ (Continy)

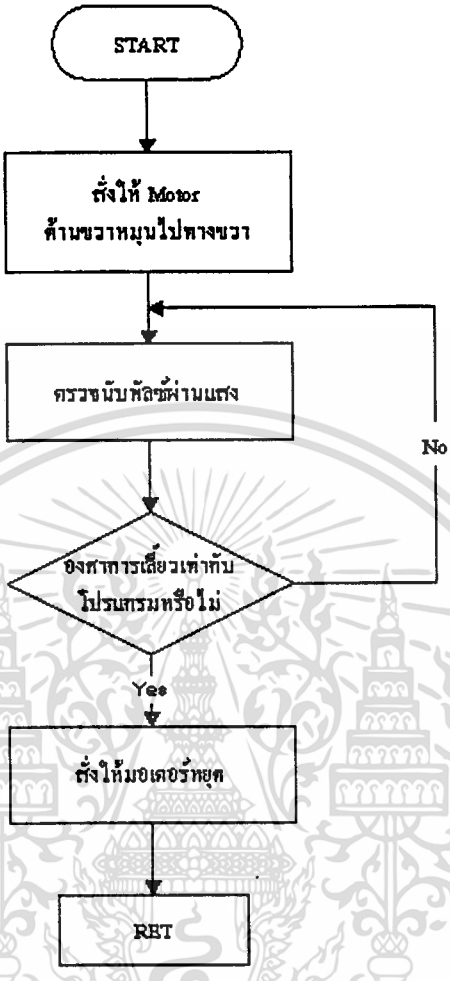
โปรแกรมคำสั่งเดินหน้า



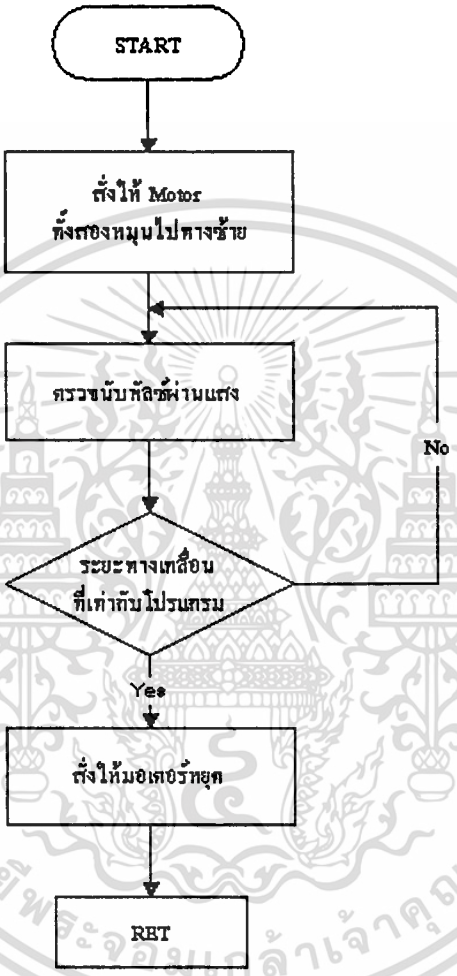
โปรแกรมเดินหน้าเลี้ยวขวา



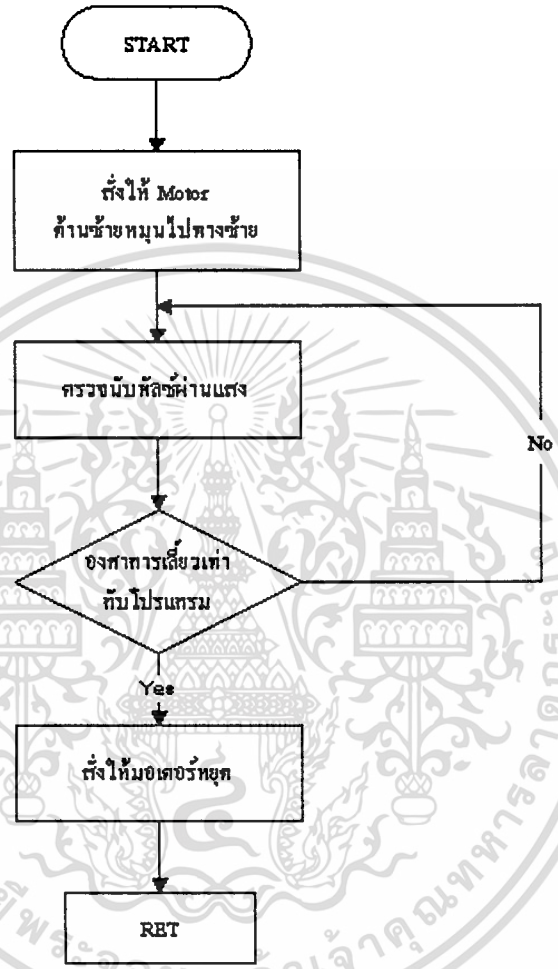
โปรแกรมเดินหน้าเลี้ยวซ้าย



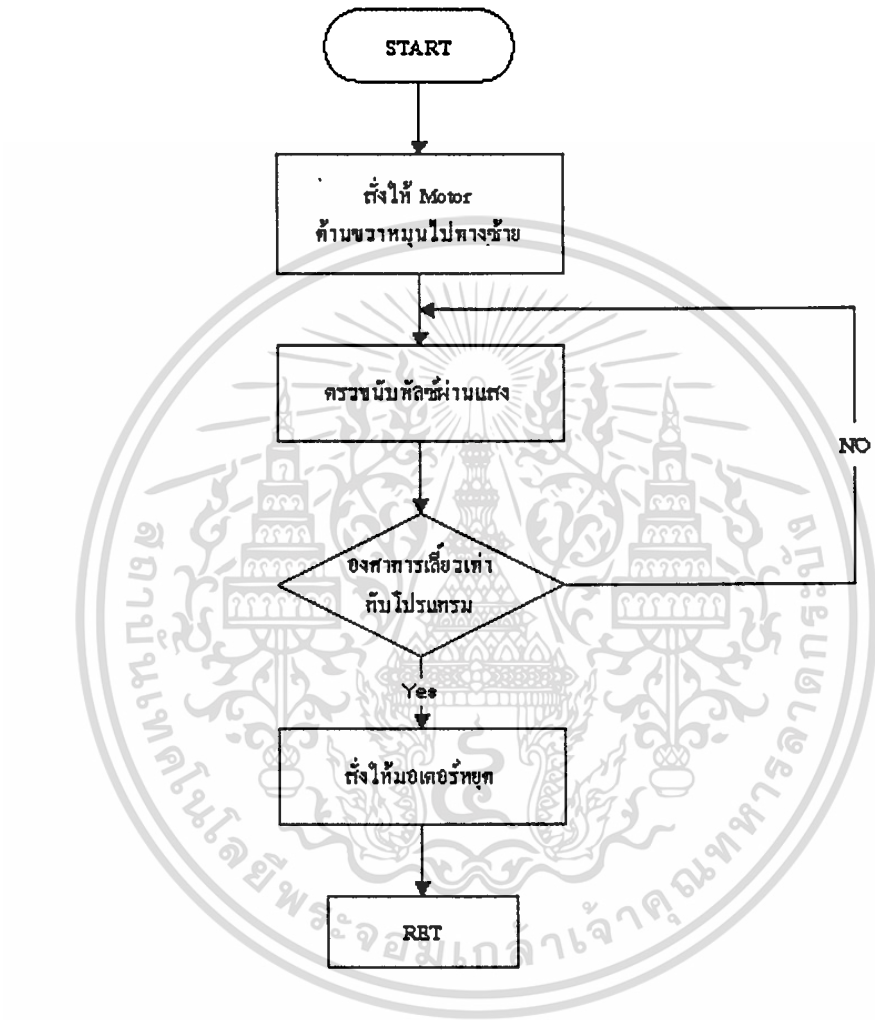
โปรแกรมถอยหลัง



โปรแกรมถอยหลังไปทางขวา

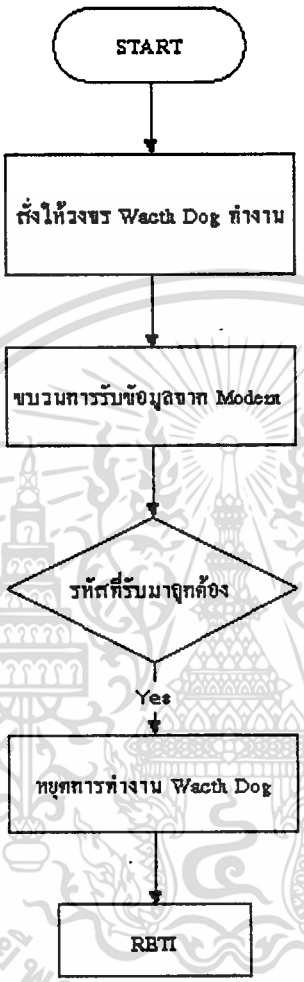


โปรแกรมถอยหลังไปทางซ้าย



โปรแกรมตรวจอุณหภูมิ

โปรแกรมบริการการอินเตอร์รัพท์ของหุ่นยนต์

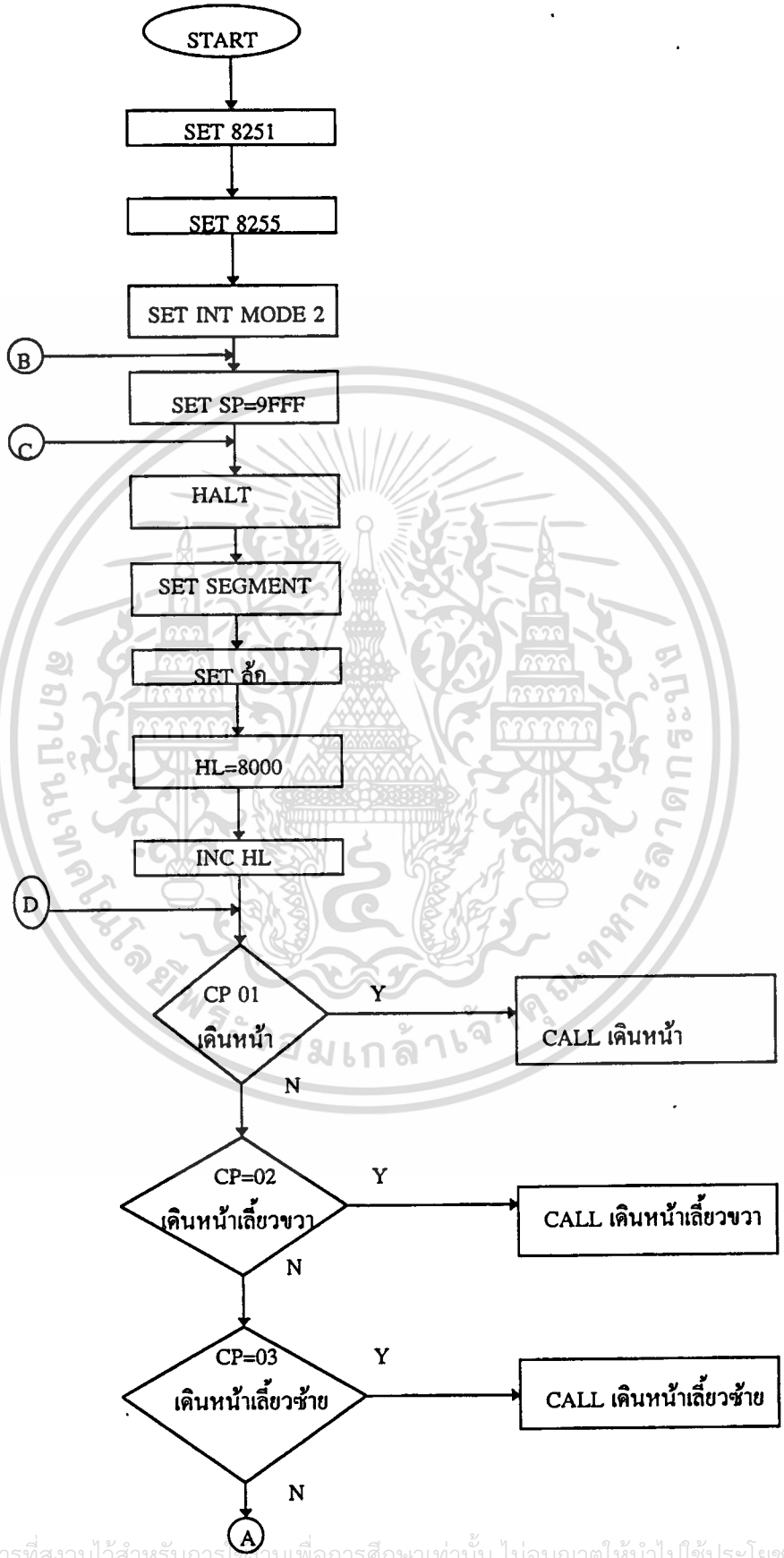


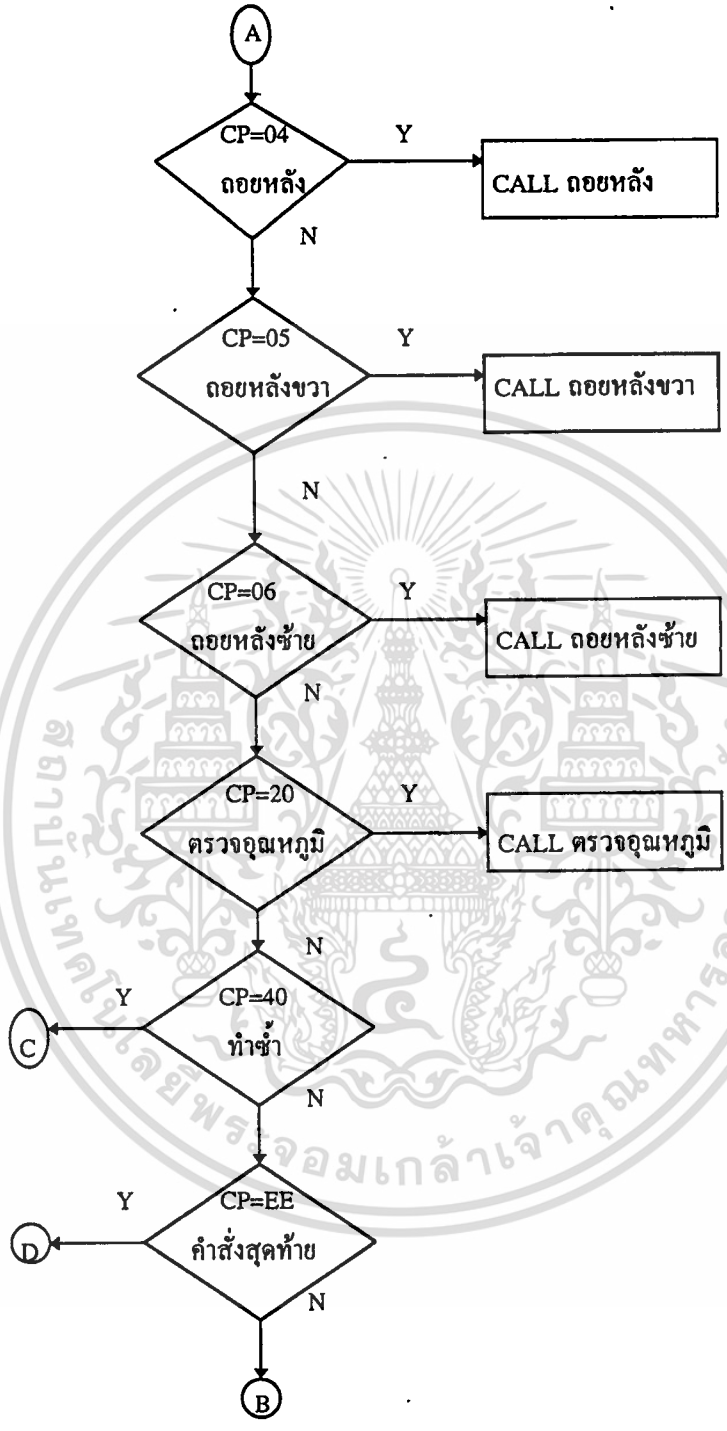
สำหรับกาทำงานของโปรแกรมหุ่นยนต์(Continy) นั้นสามารถอธิบายได้ดังต่อไปนี้ เริ่มด้วยเปิดเครื่องไมโครโปรเซสเซอร์ Z-80 จะทำการเซตระบบการทำงานต่าง ๆ เช่น โหมดการ INT,สัญญาณควบคุมของ 8251 และ 8255 จากนั้นมันจะทำการตรวจสอบความ พร้อมของ Modem โดยการตรวจสอบว่ามีการส่งสัญญาณ Sync มาหรือไม่ ถ้าไม่มีการส่งสัญญาณหุ่นยนต์จะไม่ทำงานใดๆทั้งสิ้น จนกว่าจะมีการเปิดเครื่อง Modem หรือ Modem อยู่ในสภาวะที่พร้อมจะทำงาน เมื่อหุ่นยนต์ได้ตรวจสอบพบว่ามีการส่งสัญญาณ Sync มาจาก Modem

มันก็จะอยู่ในสภาวะที่พร้อมทำงานทุกเมื่อ ทันทีเมื่อ Modem ได้ทำการส่งข้อมูลมาแล้วมันจะ ทำการกระโดดเข้าสู่โปรแกรมบริการอินเตอร์รัพท์ทันที สำหรับในส่วนของโปรแกรมบริการ อินเตอร์รัพท์นั้นจะทำการสั่งให้วง

จร Wacth Dog ทำงานขณะเดียวกันก็จัดทำขบวนการรับข้อ มูลคำสั่งจาก Modem เมื่อข้อมูลที่รับได้ถูกต้องหมดมันจะหยุดการทำงานของวงจร Wacth Dog แต่ถ้าหากว่าเกิดการรับข้อมูลผิดพลาดเกิดขึ้นหุ่นยนต์จะทำงานแสดงผล Error ขณะเดียวกัน วงจร Wacth Dog จะทำการรีเซตการทำงานของหุ่นยนต์โดยอัตโนมัติ ซึ่งจะป้องกันการ "Hank" ของหุ่นยนต์ ถ้าข้อมูลถูกต้องหมดโปรแกรมจะกลับเข้าสู่โปรแกรมหลัก คือจะนำคำสั่งต่างๆ ที่ถูกโปรแกรมไว้มาปฏิบัติงานโดยการเรียกโปรแกรมย่อยต่างๆ เมื่อปฏิบัติตามคำสั่ง เสร็จสิ้นแล้วมันจะกลับมาเช็คสัญญาณ Sync อีกครั้งหนึ่ง

Flowchart of ROBOT





โปรแกรมหลักของโมเด็ม

```
        LD      DE,0000
###:    DEC     DE
        LD      A,D
        OR      E
        JRNZ,   ###
        LD      SP,9FFF
        LD      A, FF
        OUT     ( C1 ) ,A
        LD      A, 45
        OUT     ( C1 ),A
        LD      A,FF
        OUT     ( C1 ),A
        LD      A,05
        OUT     ( C1 ),A
        LD      A,91
        OUT     ( 43 ) ,A
        IN      A, ( 42 )
        BIT     1,A
        JNZ,    AAA
LOOP1:  LD      A, AA
        OUT     ( C0 ),A
LOOP2:  IN      A, ( C1 )
        BIT     O,A
        JR Z,   LOOP 2
        JR      LOOP 1
Chacle: LD      DE, 8000
        LD      A, ( DE )
        CP      AA
        JR      NZ,error
FIN :   INC     DE
NEW :   LD      A, ( DE )
```

```
                CP      EE
                JRZ,    pass
                CP      20
                JR Z,   FIN
                CP      40
                JRE,    FIN
                LD      HL, 0200
                LD      BC, 0007
                CP      IR
                LD      A,B
                OR      C
                JR Z,   error
                INC     DE
                INC     DE
                JR,     NEW
PASS :          LD      A, 40
                OUT     ( 42 ), A
                LD      DE, 8000
NEO :           LD      A, ( DE )
                CP      EE
                JR      Z, END
                LD      BC , 1000
DELAY : DEC     BC
                LD      A,B
                OR      C
                JRNZ ,  DELAY
                LD      A, ( DE )
                OUT     ( CO ) ,A
   ASK : IN     A, ( C1 )
                BIT     O,A
                JR Z,   ASK
                INC     DE
```

```
                JR ,    NEO
END :           LD      BC , 1000
DELAY 2 :       DEC     BC
                DEC     BC
                LD      A,B
                OR      C
                SRNZ ,  DELAY 2
                LD      A, ( DE )
                OUT     ( CO ) , A
ASK 2 :  IN     A, ( C1 )
                BIT     2, A
                JR Z,   ASK2
                LD      A, 00
                OUT     ( 42 ) , A
                JP ,    AAA
ERROR : LD      A, 20
                OUT     ( 42 ) , A
&&& :           NOP
                JR ,    &&&
AAA :           LD      HL, 8000
STB :           IN      A, ( 42 )
                BIT     O,A
                JANZ ,  STB
                LD      A, 10
                OUT     ( 42 ) ,A
                IN      A, ( 40 )
                LD      ( HL ) , A
                CP      OA
                JR Z,   RET
                INC     HL
                LD      A,00
                OUT     ( 42 ) ,A
                LD      B, 10
```

```
LOK :       DEC    B
            JRNZ,  LOK
            JR,    STB
RET :       LD     A, 00
            OUT    ( 42 ) , A
            LD     HL , 8000
            LD     D,02
JJJ :       LD     IX 0300
            LD     B , 00
            LD     A, ( HL )
            CP     30
            JR     C, error
            CP     40
            JR     C, KKK
            CP     41
            JR Z,  KKK
            CP     45
            JR Z,  KKK
            JR,    errror
KKK :       SUB    30H
            LD     C,A
            ADD    IX+BC
            LD     A, ( IX )
            CP     OE
            JR Z,  LLL
MMM : LD    ( HL ) , A
            INC    HL
            JR,    JJJ
LLL :       DEC    D
            JRNZ ,  MMM
            LD     ( HL ) , A
* :        LD     HL , 8000
            LD     DE , 8000
```

```
QQQ :     LD    A, ( HL )
          CALL  POP
          LD    B, EE
          CP    B
          JRZ   RRR
          INC   DE
          INC   HL
          INC   HL
          JR ,  QQQ
error :   LD    A, 20
          OUT   ( 42 ) , A
GGG :     NOP
          JR ,  GGG
          LD    B , 04
PPP :     SLA   A
          DJNZ ,  PPP
          INC   HL
          LD    C , ( HL )
          OR    C
          DEC   HL
          LD    ( DE ), A
          RET
```

```
; MAIN PROGRAM ของหุ่นยนต์
        LD      DE,0000
AAA :   DEC     DE
        LD      A,D
        OR      E
        JRNZ    AAA
        LD      A,FF
        OUT     (C1),A
        LD      A,45
        OUT     (C1),A
        LD      A,FF
        OUT     (C1),A
        LD      A,05
        OUT     (C1),A
        LD      A,91
        OUT     (43),A
*  :    IN      A,(42)
        BIT     3,A
        JR Z    *
        LD      DE,000
BBB :   DEC     DE
        LD      A,D
        OR      E
        JR NZ   BBB
        IN      A,(42)
        BIT     3,A
        JR Z    *
        IM2
        LD      A,06
        LD      I,A
THAI :  LD      SP,9FFF
        EI
        HALT
```

```
        LD      A,3F
        OUT     (41),A
        LD      A,00
        OUT     (41),A
        LD      A,00
        OUT     (42),A
        LD      A,40
        OUT     (41),A
        LD      DE,3000
CCC :   DEC     DE
        LD      A,D
        OR      E
        JR NZ   CCC
        LD      A,00
        OUT     (41),A
        LD      B,08
DDD :   LD      DE,0000
EEE :   DEC     DE
        LD      A,D
        OR      E
        JR NZ   EEE
        DJNZ    DDD
FFF :   LD      HL,8000
        INC     HL
        LD      A,(HL)
GGG :   CP      01
        CALL Z,เดินหน้า
        CP      02
        CALL Z,เดินหน้าเลี้ยวขวา
        CP      03
        CALL Z,เดินหน้าเลี้ยวซ้าย
        CP      04
        CALL Z,ถอยหลัง
```

```
                CP      05
                CALL Z,ถอยหลังขวา
                CP      06
                CALL Z,ถอยหลังซ้าย
                CP      20
                CALL Z,อุณหภูมิ
                CP      40
                JR NZ   HHH
                JR      FFF
HHH :           CP      EE
                JR NZ   GGG
                JR      THAI
ERROR :         LD      A,3E
                OUT     (41),A
                LD      A,00
                OUT     (41),A
                JUMP    THAI

; Check ล้อซ้าย
A00 :           IN      A,(42)
                BIT     1,A
                JR NZ   A00
A01 :           IN      A,(42)
                BIT     1,A
                JR Z    A01
A02 :           IN      A,(42)
                BIT     1,A
                JR NZ   A02
                DJNZ    A01
                RET
```

```
; Check ล้อขวา
A0 :        IN      A,(42)
            BIT     0,A
            JR NZ   A0
A1 :        IN      A,(42)
            BIT     0,A
            JR Z    A1
A2 :        IN      A,(42)
            BIT     0,A
            JR NZ   A2
            DJNZ    A1
            RET

; แปลงฐาน 10 เป็นฐาน 16
            LD      C,08
START :     XOR     A
            LD      A,D
            RR      A
            PUSH    AF
            BIT     7,A
            JR Z    COR1
            SUB     30H
COR1 :      BIT     3,A
            JR Z    COR2
            SUB     03H
COR2 :      LD      D,A
            POP     AF
            RR      E
            DEC     C
            JR NZ   START
            RET
```

```
; บริการ Interupt
                LD      HI,003B
                EX      (SP),HL
                LD      A,80
                OUT     (41),A
                LD      DE,8000
ASK :           IN      A,(C1)
                BIT     1,A
                JRZ     ASK
                IN      A,(C0)
                LD      (DE),A
                CP      EE
                JR Z    ASK
                INC     DE
                JR      ASK
CHECK :         LD      A,00
                OUT     (41),A
                LD      DE,8000
                LD      A,(DE)
                CP      AA
                JR NZ   ERROR
FIN :           INC     DE
NEW :           LD      A,(DE)
                CP      EE
                JR Z    END
                CP      20
                JR Z    FIN
                CP      40
                JR Z    FIN
                LD      HL,0100
                LD      BC,0007
                CP      IR
                LD      A,B
```

```
            OR      C
            JR Z    ERROR
            INC     DE
            INC     DE
            JR      NEW
ERROR :     LD      HL,0200
            EX      (SP),HL
            RETI
END :       EI
            RETI
```

; เดินหน้า

```
            INC     HL
            LD      A,(HL)
            LD      D,A
            CALL    แปลงเลขฐาน
            LD      A,A0
            OUT     (42),A
GAK :       LD      B,35
            CALL    CHECK ล้อขวา
            DEC     E
            JR NZ   GAK
            LD      A,00
            OUT     (42),A
            INC     HL
            LD      A,(HL)
            RET
```

; เดินหน้าเลี้ยวขวา

```
            INC     HL
            LD      A,(HL)
            CP      45
            JR NZ   BOB
```

```
        LD      A,80
        OUT     (42),A
        LD      B,19
        CALL    CHECK ล้อซ้าย
        LD      A,00
        OUT     (42),A
        INC     HL
        LD      A,(HL)
        RET
BOB :   LD      A,80
        OUT     (42),A
        LD      B,30
        CALL    CHECK ล้อซ้าย
        LD      A,00
        OUT     (42),A
        INC     HL
        LD      A,(HL)
        RET

; เดินหน้าเลี้ยวซ้าย
        INC     HL
        LD      A,(HL)
        CP      45
        JR NZ   BOOB
        LD      A,20
        OUT     (42),A
        LD      B,19
        CALL    CHECK ล้อขวา
        LD      A,00
        OUT     (42),A
        INC     HL
        LD      A,(HL)
        RET
```

```
BOOB :    LD    A,20
          OUT   (42),A
          LD    B,30
          CALL  CHECK ล้อขวา
          LD    A,00
          OUT   (42),A
          INC   HL
          LD    A,(HL)
          RET

; โปรแกรมถอยหลัง
          INC   HL
          LD    A,(HL)
          LD    D,A
          CALL  แปลงเลขฐาน
          LD    A,50
          OUT   (42),A
GOAK :    LD    B,35
          CALL  CHECK ล้อขวา
          DEC   E
          JR NZ GOAK
          LD    A,00
          OUT   (42),A
          INC   HL
          LD    A,(HL)
          RET

; โปรแกรมถอยหลังเลี้ยวขวา
          INC   HL
          LD    A,(HL)
          CP    45
          JR NZ DOB
          LD    A,40
```

```
        OUT     (42),A
        LD      B,19
        CALL    CHECK ล้อซ้าย
        LD      A,00
        OUT     (42),A
        INC     HL
        LD      A,(HL)
        RET
DOB :   LD      A,40
        OUT     (42),A
        LD      B,30
        CALL    CHECK ล้อซ้าย
        LD      A,00
        OUT     (42),A
        INC     HL
        LD      A,(HL)
        RET
```

; โปรแกรมถอยหลังเลี้ยวซ้าย

```
        INC     HL
        LD      A,(HL)
        CP      45
        JR NZ   DOOD
        LD      A,10
        OUT     (42),A
        LD      B,19
        CALL    CHECK ล้อขวา
        LD      A,00
        OUT     (42),A
        INC     HL
        LD      A,(HL)
        RET
DOOD :  LD      A,10
```

```
                OUT     (42),A
                LD      B,30
                CALL    CHECK ล้อขวา
                LD      A,00
                OUT     (42),A
                INC     HL
                LD      A,(HL)
                RET

;โปรแกรมตรวจอุณหภูมิ
                LD      A,91
                OUT     (43),A
NOO :           LD      A,00
                OUT     (80),A
LOOP :          IN      A,(42)
                BIT     2,A
                JR Z    LOOP
                IN      A,(40)
                SUB     8B
                LD      IX,0E00
                LD      C,A
                LD      B,00
                ADD     IX,BC
                LD      A,(IX+0)
                LD      B,A
                AND     0F
                OR      10
                OUT     (41),A
                AND     0F
                OUT     (41),A
                LD      A,B
                SRL     A
                SRL     A
```

```
SRL     A
SRL     A
AND     0F
0R      20
OUT     (41),A
AND     0F
OUT     (41),A
INC     HL
LD      A,(HL)
RET
```

โปรแกรมส่งข้อมูลจากคอมพิวเตอร์

```
#include<stdio.h>
#include<dos.h>
#include<conio.h>
#include<string.h>

#define ESC  0x11B
#define F1   0x3B00
#define F2   0x3C00
#define F10  0x4400
#define LPT1 0x378
#define BLACK         0
#define BLUE          1
#define GREEN         2
#define CYAN          3
#define RED           4
#define MAGENTA       5
#define BROWN         6
#define LIGHTGRAY     7
#define DARKGRAY      8
#define LIGHTBLUE     9
#define LIGHTGREEN    10
#define LIGHTCYAN     11
#define LIGHTRED      12
#define LIGHTMAGENTA 13
#define YELLOW        14
#define WHITE         15
#define BLINK         128
#define return_key    '\x0d'
#define esc_key       '\x1b'
#define bs_key        '\x08'
#define home_key      '\x47'
#define up_key        '\x48'
#define lt_key        '\x4b'
```

```
#define rt_key        '\x4d'
#define end_key       '\x4f'
#define ins_key       '\x52'
#define del_key       '\x53'
#define nodisplay  0x00
#define lowdisplay0x07
#define highdisplay          0x0f
#define underlinelow         0x01
#define underlinehigh        0x09
#define reverselow0x70
#define reversehigh          0x78
#define blinklow   0x87
#define blinkhigh  0x8f
#define undblinklow          0x81
#define undblinkhigh         0x89
#define revblinklow          0xf0
#define revblinkhigh         0xf8
#define true                 1
#define false                0
typedef unsigned char byte;
typedef char string80[81];
int funckey;
char key;

  char *menu_msg[]={
        "เดินหน้า(0-99)เมตร \n",
        "เลี้ยวขวา(45,90)องศา \n",
        "เลี้ยวซ้าย(45,90)องศา \n",
        "ถอยหลัง(0-99)เมตร \n",
        "ถอยเลี้ยวขวา(45,90)องศา \n",
        "ถอยเลี้ยวซ้าย(45,90)องศา \n",
        "รายงานอุณหภูมิ(เซลเซียส) \n",
```

```
        "จบโปรแกรม \n",
        };
  char *yesno=" คุณต้องการทำซ้ำอีกหรือไม่(Y/N) ";
  char *yn_msg[]={ " YES ", " NO ",};
  char *menu="                    โปรแกรมควบคุมหุ่นยนต์                    ";
  char *num[]={ "1.","2.","3.","4.","5.","6.","7.","8.",};
  char *func="   F1                F2                ESC              ";
  char *h=" Help ";
  char data[100];
  int  ctrl_robot=0;
  int  l,e,s;


main()
 {
   int x=10,y=8;
   clrscr();
   textmode(3);
   background();
   cursoroff();
   display_block(1,1,80,24);
   display_text(x,y);
   control();
   select_menu();
 }
/*-----------------------------------*/
readfunckey()
{
   key =getch();
   if(key == '\x0')
   {
      funckey = true;
         key     =getch();
   }
```

```
    else
    {
          if((key == esc_key)||(key == bs_key))
             funckey = true;
        else
             funckey = false;
    }
}
setcursor(byte top,byte bottom)
{
   union REGS regs;
   regs.h.ah=1;
   regs.h.ch=top;
   regs.h.cl=bottom;
   int86(0x10,&regs,&regs);
}
cursoron()
{
   setcursor(12,13);
}
cursoroff()
{
   setcursor(0x20,1);
}

inschar(char *str,char ch,int x)
{
    string80 str1;
    strcpy(str1,str+x);
    str[x]=ch;str[x+1]='\0';
    strcat(str,str1);
}
delchar(char *str,int x)
{
```

```
    string80 str1;
    strcpy(str1,str);
    str[x]='\0';
    strcat(str,str1+x+1);
}
clrline(int len)
{
    int x,y,i;
    x=wherex();y=wherey();
    for(i=0;i<len;i++)
    {
       gotoxy(x+i,y);cprintf(" ");
    }
    gotoxy(x,y);
}

readstr(char *str,int len)
{
    int x,new=false,endloop=false,xx,y;
    string80 strsave;
    xx=wherex();y=wherey();
    strcpy(strsave,str);
    x=strlen(str);cursoron();

    textattr(reverselow);
    gotoxy(xx,y);clrline(len);
    textattr(lowdisplay);
    gotoxy(xx,y);cprintf("%s",str);
    textattr(reverselow);
    do{
       readfunckey(key);
       if(funckey){
            gotoxy(xx,y);cprintf("%s",str);new=true;
            switch(key){
               case lt_key    : x--;
                                if(x<0)
```

```
                    x=0;
                gotoxy(xx+x,y);
                break;
case rt_key     : if(x<strlen(str))
                    x++;


                gotoxy(xx+x,y);
                break;
case del_key    : if(x<strlen(str)){
                    delchar(str,x);
                    gotoxy(xx,y);clrline(len);
                    gotoxy(xx,y);cprintf("%s",str);
                    gotoxy(xx+x,y);
                }
                break;
case bs_key     : if(x>0){
                    delchar(str,x-1);
                    x--;
                    gotoxy(xx,y);clrline(len);
                    gotoxy(xx,y);cprintf("%s",str);
                    gotoxy(xx+x,y);
                }
                break;
case home_key   : x=0;
                gotoxy(xx+x,y);
                break;
case end_key    : x=strlen(str);
                gotoxy(xx+x,y);
                break;
```

```
        case esc_key   : strcpy(str,strsave);
                         endloop=true;
                         break;
      }
}
else
{
    if(key!=return_key)
     {
      if(key>='0'&&key<='9')
       if(new==false)
       {
        new=true;
        str[0]=key;str[1]='\0';x=1;
        gotoxy(xx,y);clrline(len);
        gotoxy(xx,y);cprintf("%s",str);
        gotoxy(xx+x,y);
       }
      else{
      if(strlen(str)<len){
        inschar(str,key,x);
        x++;
        gotoxy(xx,y);cprintf("%s",str);
        gotoxy(xx+x,y);
        if(strlen(str)>=len)
          gotoxy(xx+x-1,y);
       }
      }
    }
    else
        endloop=true;
 }
```

```
    }while(endloop==false);
    gotoxy(xx,y);cprintf("%s",str);
    textattr(lowdisplay);
    cursoroff();
}
/*-----------------------------------*/
control()
{
  int i,j,k;
  int h=45,l=7;
  draw_border(45,7,75,16);
    for(j=1;j<=8;j++)
      {
          for(k=1;k<=29;k++)
            {
              textbackground(LIGHTGRAY);
              gotoxy(h+k,l+j);
              cputs(" ");
            }
      }
   for(i=0;i<8;i++)
    {
      textcolor(RED);
      gotoxy(h+1,l+1+i);
      cputs(num[i]);
    }
    textcolor(WHITE);
 }
 /*-----------------------------------*/
 background()
 {
  int i,k;
     for(i=4;i<=23;i++)
       {
```

```
        for(k=3;k<=78;k++)
          {
            textbackground(BLUE);
            gotoxy(k,i);
            cputs(" ");
          }
      }
}
/*-----------------------------------*/
display_text(int x,int y)
  {
   int i;
    textbackground(LIGHTGRAY);
    textcolor(RED);
    gotoxy(3,2);
    cputs(menu);
    textbackground(LIGHTGRAY);
    textcolor(RED);
    gotoxy(2,25);
    cputs(func);
    textbackground(LIGHTGRAY);
    textcolor(BLUE);
    gotoxy(7,25);
    cputs("=การใช้งาน");
    gotoxy(34,25);
    cputs("=ลบโปรแกรม");
    gotoxy(63,25);
    cputs("=ออกจากโปรแกรม");
    normvideo();
    highvideo();
    textcolor(YELLOW);
    textbackground(BLUE);
    gotoxy(9,17);
```

```
    cputs("กรุณาเลือกคำสั่งให้หุ่นยนต์ทำงาน(1-8)");
    normvideo();
    for(i=0;i<8;i++)
     {
      textcolor(RED);
      textbackground(BLUE);
      highvideo();
      gotoxy(x-2,y+i);
      cputs(num[i]);
      normvideo();
      textcolor(WHITE);
      textbackground(BLUE);
      gotoxy(x,y+i);
      cputs(menu_msg[i]);
     }
  }
/*-----------------------------------------------*/
display_block(startx,starty,endx,endy)
  int startx,starty,endx,endy;
  {
    register int i;
    textcolor(LIGHTGRAY);
    for(i=startx+1;i<endx;i++)
      {
       gotoxy(i,starty);
       cprintf("%c",205);          /* อ */
       gotoxy(i,endy);
       cprintf("%c",196);          /* ฤ */
       gotoxy(i,starty+2);
       cprintf("%c",205);
      }
    for(i=starty;i<endy;i++)
      {
```

```
      gotoxy(startx+1,i);
      cprintf("%c",179);          /* ณ */
      gotoxy(endx-1,i);
      cprintf("%c",179);
     }
      gotoxy(startx+1,starty);cprintf("%c",213);  /* ี */
      gotoxy(endx-1,starty);cprintf("%c",184);    /* ธ */
      gotoxy(startx+1,endy);cprintf("%c",192);    /* ภ */
      gotoxy(endx-1,endy);cprintf("%c",217);      /* ู */
      gotoxy(startx+1,starty+2);cprintf("%c",198); /* ฦ */
      gotoxy(endx-1,starty+2);cprintf("%c",181);  /* ต */
    }
 /*-----------------------------------------*/
draw_border(startx,starty,endx,endy)
  int startx,starty,endx,endy;
{
    register int i;
    for(i=startx+1;i<endx;i++)
      {
       textcolor(BLUE);
       textbackground(WHITE);
       gotoxy(i,starty);
       cprintf("%c",196);
       gotoxy(i,endy);
       cprintf("%c",196);
      }
    for(i=starty+1;i<endy;i++)
      {
       gotoxy(startx,i);
       cprintf("%c",179);
       gotoxy(endx,i);
       cprintf("%c",179);
      }
```

```
        gotoxy(startx,starty);cprintf("%c",218);
        gotoxy(endx,starty);cprintf("%c",191);
        gotoxy(startx,endy);cprintf("%c",192);
        gotoxy(endx,endy);cprintf("%c",217);
}
/*-----------------------------------------------*/
select_menu()
 {
   int n,k=0,x;
   int in;
   for(;;)
    {
     in=bioskey(0);
          switch(in)
           {
            case 0x231:/*กด '1'*/
                 n=0;k++;input(n,k);break;
            case 0x332:/*กด '2'*/
                 n=1;k++;input(n,k);break;
            case 0x433:/*กด '3'*/
                 n=2;k++;input(n,k);break;
            case 0x534:/*กด '4'*/
                 n=3;k++;input(n,k);break;
            case 0x635:/*กด '5'*/
                 n=4;k++;input(n,k);break;
            case 0x736:/*กด '6'*/
                 n=5;k++;input(n,k);break;
            case 0x837:/*กด '7'*/
                 n=6;k++;input(n,k);break;
            case 0x938:/*กด '8'*/
                 n=7;k++;input(n,k);break;
            case F1:help();break;
```

```
case F2:control();
        for(x=3;x<=75;x++)
         {
           textbackground(BLUE);
           gotoxy(x,20);
           cputs(" ");
         }
        k=0;
        ctrl_robot=0;
        break;
case F10:textmode(-1);
         clrscr();
         textcolor(YELLOW);
         highvideo();
         cputs("Good bye! form KMIT'L\n\n\r");
         normvideo();
         textcolor(LIGHTCYAN);
         cputs("Press any key to continue...");
         getch();
         textmode(-1);
         clrscr();
         cursoron();
         exit(0);
case ESC:textmode(-1);
         clrscr();
         textcolor(YELLOW);
         highvideo();
         cputs("Good bye! form KMIT'L\n\n\r");
         normvideo();
         textcolor(LIGHTCYAN);
         cputs("Press any key to continue...");
         getch();
         textmode(-1);
         clrscr();
```

```
                cursoron();
                exit(0);

        /*default:return;*/
        }
    }
  }
 /*---------------------------------------*/
input(int n,int k)
{
   int x=48,y=7;
   textcolor(BLUE);
   textbackground(WHITE);
      switch(k)
         {
          case 1:
                chk_data(k,n);
                break;
          case 2:
                chk_data(k,n);
                break;
          case 3:
                chk_data(k,n);
                break;
          case 4:
                chk_data(k,n);
                break;
          case 5:
                chk_data(k,n);
                break;
          case 6:
                chk_data(k,n);
                break;
```

```
            case 7:
                    chk_data(k,n);
                    break;
            case 8:
                    gotoxy(x,y+k);
                    cprintf(menu_msg[n]);
                    yes_no(ctrl_robot);
                    break;
            default:return;
          }
        textcolor(WHITE);
}
/*---------------------------------------*/
windows(startx,starty,endx,endy,color,disp)
  int startx,starty,endx,endy,color;
  char *disp;
{
    register int i,j;
    for(i=startx+1;i<endx;i++)
      {
       textcolor(YELLOW);
       textbackground(color);
       gotoxy(i,starty);
       cprintf("%c",196);
       gotoxy(i,endy);
       cprintf("%c",196);
      }
    for(i=starty+1;i<endy;i++)
      {
       gotoxy(startx,i);
       cprintf("%c",179);
```

```
    gotoxy(endx,i);
    cprintf("%c",179);
   }
    gotoxy(startx,starty);cprintf("%c",218);
    gotoxy(endx,starty);cprintf("%c",191);
    gotoxy(startx,endy);cprintf("%c",192);
    gotoxy(endx,endy);cprintf("%c",217);
    gotoxy(((endx-startx)/2)+startx-2,starty);
    cprintf(disp);
    window(startx+1,starty+1,endx-1,endy-1);
  for(j=1;j<(endy-starty);j++)
   {
    for(i=1;i<(endx-startx);i++)
     {
          textbackground(color);
          gotoxy(i,j);
          cputs(" ");
     }
   }

}
/*-----------------------------------------------*/
help()
{
 char ch;
 char buf[40*20*2];
 gettext(25,5,62,21,buf);
 do{
   windows(25,5,60,20,CYAN,h);
   gotoxy(3,1);
   cprintf("เลือกหมายเลขตามลำดับให้หุ่นยนต์");
   gotoxy(3,2);
   cprintf("ทำงานโดยใส่หมายเลขตามที่อยู่ใน");
```

```
  gotoxy(3,3);
  cprintf("วงเดือน");
  gotoxy(4,4);
  cprintf("เช่น 1.เลี้ยวซ้าย(0-99)เมตร 50");
  gotoxy(4,5);
  cprintf("    2.เลี้ยวขวา(0-99)เมตร 60");
  gotoxy(4,6);
  cprintf("    3.จบโปรแกรม");
  ch=getch();
  }while(ch!=0x1b);
puttext(25,5,62,21,buf);
window(1,1,80,25);
}
/*-------------------------------------------------*/
windo(startx,starty,endx,endy)
 int startx,starty,endx,endy;
{
   register int i,j;
   for(i=startx+1;i<endx;i++)
     {
      textcolor(YELLOW);
      textbackground(RED);
      gotoxy(i,starty);
      cprintf("%c",196);
      gotoxy(i,endy);
      cprintf("%c",196);
      }
   for(i=starty+1;i<endy;i++)
      {
       gotoxy(startx,i);
       cprintf("%c",179);
       gotoxy(endx,i);
       cprintf("%c",179);
      }
```

```
    gotoxy(startx,starty);cprintf("%c",218);
    gotoxy(endx,starty);cprintf("%c",191);
    gotoxy(startx,endy);cprintf("%c",192);
    gotoxy(endx,endy);cprintf("%c",217);
    gotoxy(((endx-startx)/2)+startx-2,starty);
  for(j=starty+1;j<endy;j++)
   {
    for(i=startx+1;i<endx;i++)
     {
        textbackground(RED);
        gotoxy(i,j);
        cputs(" ");
     }
   }

}
/*********************************************/
yes_no(int m)
{

   register int i;
   char ch,c;
   char start[2];
   char end[2];
   char loop[2];
   char buf[35*10*2];
     gettext(28,19,63,23,buf);
     windo(28,19,63,23);
     textcolor(WHITE);
     gotoxy(31,21);
     cprintf(yesno);
     ch=getch();
```

```
    if((ch=='y')||(ch=='Y')){
       puttext(28,19,63,23,buf);
       start[s]='A';
       fprintf(stdprn,"%c",start[s]);
       s++;start[s]='A';
       fprintf(stdprn,"%c",start[s]);
         for(i=1;i<=m;i++)
          {
              fprintf(stdprn,"%c",data[i]);
          }
       loop[l]='4';
       fprintf(stdprn,"%c",loop[l]);
       l++;loop[l]='0';
       fprintf(stdprn,"%c",loop[l]);
       end[e]='E';
       fprintf(stdprn,"%c",end[e]);
       e++;end[e]='E';
       fprintf(stdprn,"%c\n",end[e]);
    }
else if((ch=='n')||(ch=='N'))
       {
       puttext(28,19,63,23,buf);
       start[s]='A';
       fprintf(stdprn,"%c",start[s]);
       s++;start[s]='A';
       fprintf(stdprn,"%c",start[s]);
         for(i=1;i<=m;i++)
          {
              fprintf(stdprn,"%c",data[i]);
          }
       end[e]='E';
       fprintf(stdprn,"%c",end[e]);
       e++;end[e]='E';
```

```
            fprintf(stdprn,"%c\n",end[e]);
            }
}
/*****************************************************/
chk_data(int k;int n)
{
  int c,x=48,y=7,i;
  int z=47;
  char input_str[10]="";

    if(n==0){
            ctrl_robot++;
               data[ctrl_robot]='0';
               ctrl_robot++;
               data[ctrl_robot]='1';
            gotoxy(x,y+k);
               cprintf(menu_msg[n]);
               cursoron();
               gotoxy(z+strlen(menu_msg[n]),y+k);
               strcpy(input_str,"00");
               readstr(input_str,2);
               for(i=0;i<strlen(input_str);i++)
               {
              ctrl_robot++;
                    data[ctrl_robot]=input_str[i];
               }
              }
               else if(n==1){
             ctrl_robot++;
               data[ctrl_robot]='0';
               ctrl_robot++;
               data[ctrl_robot]='2';
             gotoxy(x,y+k);
```

```
    cprintf(menu_msg[n]);
    cursoron();
    gotoxy(z+strlen(menu_msg[n]),y+k);
    strcpy(input_str,"00");
    readstr(input_str,2);
    for(i=0;i<strlen(input_str);i++)
    {
  ctrl_robot++;
        data[ctrl_robot]=input_str[i];
    }
  }
  else if(n==2){
 ctrl_robot++;
    data[ctrl_robot]='0';
    ctrl_robot++;
    data[ctrl_robot]='3';
 gotoxy(x,y+k);
    cprintf(menu_msg[n]);
    cursoron();
    gotoxy(z+strlen(menu_msg[n]),y+k);
    strcpy(input_str,"00");
    readstr(input_str,2);
    for(i=0;i<strlen(input_str);i++)
    {
  ctrl_robot++;
        data[ctrl_robot]=input_str[i];
    }
  }
   else if(n==3){
 ctrl_robot++;
    data[ctrl_robot]='0';
    ctrl_robot++;
    data[ctrl_robot]='4';
```

```
gotoxy(x,y+k);
    cprintf(menu_msg[n]);
    cursoron();
    gotoxy(z+strlen(menu_msg[n]),y+k);
    strcpy(input_str,"00");
    readstr(input_str,2);
    for(i=0;i<strlen(input_str);i++)
    {
  ctrl_robot++;
        data[ctrl_robot]=input_str[i];
    }
  }
  else if(n==4){
ctrl_robot++;
    data[ctrl_robot]='0';
    ctrl_robot++;
    data[ctrl_robot]='5';
 gotoxy(x,y+k);
    cprintf(menu_msg[n]);
    cursoron();
    gotoxy(z+strlen(menu_msg[n]),y+k);
    strcpy(input_str,"00");
    readstr(input_str,2);
    for(i=0;i<strlen(input_str);i++)
    {
  ctrl_robot++;
        data[ctrl_robot]=input_str[i];
    }
  }
  else if(n==5){
 ctrl_robot++;
    data[ctrl_robot]='0';
    ctrl_robot++;
```

```
    data[ctrl_robot]='6';
gotoxy(x,y+k);
    cprintf(menu_msg[n]);
    cursoron();
    gotoxy(z+strlen(menu_msg[n]),y+k);
    strcpy(input_str,"00");
    readstr(input_str,2);
    for(i=0;i<strlen(input_str);i++)
    {
  ctrl_robot++;
        data[ctrl_robot]=input_str[i];
    }
  }
  else if(n==6){
ctrl_robot++;
    data[ctrl_robot]='2';
    ctrl_robot++;
    data[ctrl_robot]='0';
 gotoxy(x,y+k);
    cprintf(menu_msg[n]);
    cursoron();
    gotoxy(z+strlen(menu_msg[n]),y+k);
    strcpy(input_str,"");
    readstr(input_str,2);
    for(i=0;i<strlen(input_str);i++)
    {
  ctrl_robot++;
        data[ctrl_robot]=input_str[i];
    }
  }
  else if(n==7){
 gotoxy(x,y+k);
    cprintf(menu_msg[n]);
```

```
            yes_no(ctrl_robot);
            }

            cursoroff();

}
```

ปัญหาที่เกิดและแนวทางการแก้ไข

ในการจัดสร้างออกแบบงานวิจัยชิ้นนี้ ทางผู้จัดทำได้ดำเนินการสร้าง ทดลอง แก้ปัญหา และทำการปรับปรุงมาเรื่อยๆ เพื่อให้งานวิจัยโครงงานชิ้นนี้ มีประสิทธิภาพมากที่สุด จึงขอสรุป ปัญหาที่เกิดขึ้น แนวทางการแก้ให้ปัญหาที่ได้ดำเนินไป และผลที่ได้รับดังต่อไปนี้

1.ปัญหาจากสัญญาณรบกวนจากการทำงานของไมโครโปรเซสเซอร์คือ จะมีการสอด แทรกสัญญาณที่มีความถี่สูงๆ สอดแทรกเข้ามากับสัญญาณข่าวสารด้วย หนทางแก้ไขได้ใส่วงจรกรองย่านความถี่ของสัญญาณข่าวสารผลที่ได้รับคือ สามารถลดการรบกวน

จากไมโครโปรเซสเซอร์ได้พอสมควร

2.ปัญหาจากระยะทางการรับส่งของข้อมูล จากวงจรที่ออกแบบไว้ยังมีกำลังส่งต่ำดังนั้นหากต้องการวบคุมการทำงานของหุ่นยนต์ให้มีประสิทธิภาพแล้วควรเพิ่มกำลังส่งให้สูงขึ้น

สำหรับแนวทางที่ได้ดำเนินไปคือ เพิ่มไฟเลี้ยงให้กับวงจร แต่สำหรับแนวทางแก้ไขให้ดีนั้น จะต้องออกแบบย่านความถี่การรับส่งให้สูงขึ้น ทั้งนี้วงจรที่ได้ทำการวิจัยนั้นได้ใช้ความถี่ที่ต่ำซึ่งไม่สามารถออกแบบให้มีกำลังส่งสูงได้เพราะจะต้องใช้สายอากาศที่ยาวมาก ๆ ซึ่งเป็นปัญหาเกี่ยวกับ SWR เมื่อออกแบบเพิ่มกำลังส่งให้สูงขึ้น

3.ปัญหาทางด้านการถอดรหัสสัญญาณ Decode ในการออกแบบในตอนแรกนั้นได้เลือกใช้ไอซี XR 2211 ซึ่งจะทำงานลักษณะ PLL แต่เมื่อมาใช้งานจริงแล้วพบว่าเมื่อ ทำการถอดสัญญาณแล้วบางครั้งไม่แน่นอน ซึ่งเป็นเพราะว่าการทำงานของ XR 2211

มีช่วง Band With ของการตอบสนองอยู่ในช่วงที่จำกัดและยังไวต่อสัญญาณรบกวนที่อาจสอดแทรกเข้ามาได้ง่ายจึงทำให้ข้อมูลมีการผิดพลาด

4.ปัญหาทางด้านวงจรจ่ายกำลังในตอนแรกได้เลือกใช้แหล่งจ่ายพลังงาน คือ แบตเตอร์รี่ลูกเดียวซึ่งจะแบ่งจ่ายแรงดันไปให้กับส่วนควบคุม และส่วนขับมอเตอร์ ผลการทำงานพบว่าเมื่อหุ่นยนต์มีการเคลื่อนที่จะมีการดึงกระแสในตอน Start สูงซึ่งทำให้หุ่นยนต์ทำงานผิดพลาดเกิดขึ้น สำหรับหนทางแก้ไขนั้นสามารถแก้ไขได้ 2 วิธี สำหรับวิธีแรกคือเพิ่มแบตเตอรี่ให้จ่ายกระแสให้สูงขึ้นและออกแบบระบบวงจร Stabirizer ด้วย ส่วน

วิธีที่ สอง คือ เพิ่มแบตเตอร์รี่อีกลูก เพื่อต้องการแยกแหล่งจ่ายไฟทั้งสองส่วนออกจาก กันโดยเด็ดขาด สำหรับในการออกแบบการทำงานนั้น ได้ตัดสินใจเลือกใช้แบตเตอรี่สองลูกแทนการใช้แบตเตอรี่ลูกเดียวที่จ่ายกระแสได้สูง เพราะว่าได้เปรียบเทียบกันกับระดับงบประมาณราคาของทั้งสองกรณีแล้ว

สรุปและวิจารณ์ผลการทำงาน

ผลจากการวิจัยโดยทำการออกแบบในส่วนModemและในส่วนหุ่นยนต์ เมื่อทดลองส่ง ข้อมูลเชื่อมต่อกันพบว่าสามารถทำงานได้อย่างมีประสิทธิภาพอยู่ในระดับที่เป็นที่น่าพอใจ คือ หาก ข้อมูลถูกส่งมาอย่างไรหุ่นยนต์ก็จะปฏิบัติงานตามโปรแกรมคำสั่งนั้น ถ้าหากคำสั่งนั้นถูกต้องแต่ถ้า หากข้อมูลของคำสั่งเปลี่ยนไปหุ่นยนต์จะไม่ทำงาน ซึ่งจากการทดลองในพื้นที่รัศมี 40 เมตร พบว่าการทำงานของหุ่นยนต์สามารถทำงานได้อย่างมีประสิทธิภาพ 99% แต่ถ้าหากระยะทางยิ่งห่างไกลออกไปจะทำให้ประสิทธิภาพการทำงานลดลง สำหรับพื้นที่รัศมีการรับส่งของข้อมูลนั้นยังพบว่าขึ้นอยู่กับอุณหภูมิและความชื้นของอากาศด้วยซึ่งจะสังเกตจากช่วงเวลาของการทดสอบ (เช้า,กลางวัน,กลางคืน)

เอกสารอ้างอิง

1.ไมโครโปรเซสเซอร์ไมโครคอมพิวเตอร์ ยืน ภู่วรวรรณ

2.ทฤษฎีการประยุกต์ ไมโครโปรเซสเซอร์ Z-80 ยืน ภู่วรวรรณ

3.การสื่อสารข้อมูล ชูชัย ธนสารตั้งเจริญ ทินกร ดุ๊ก

4.การสื่อสารข้อมูลและไมโครคอมพิวเตอร์เนตเวอร์ท ไพศาล สรวนหมู่ ยืน ภู่วรวรรณ

5.หลักการระบบสื่อสาร ด.ร. ประสิทธิ์ ประพิณมงคลการ

6.Modem สุพจน์ ปุณณชัยยะ

7.การใช้งาน Z-80 ทินกร ดุ๊ก

8.ELECTROINC COMMUNICATION SYSTEM FRAMK R.

9.LINEAR & INTERFACE INTEGARATE CIRCUIT,MOTOROLA

10.ROBOT Bulider's BONAZA GORDON,MCCOM

Silicon Gate MOS 8255

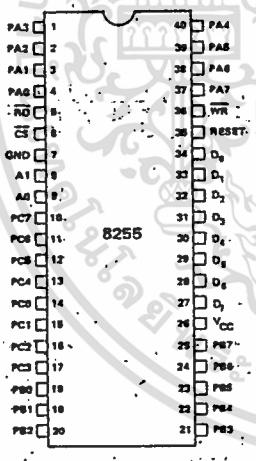
PROGRAMMABLE PERIPHERAL INTERFACE

- 24 Programmable I/O Pins
- Completely TTL Compatible
- Fully Compatible with MCS™-8 and MCS™-80 Microprocessor Families
- Direct Bit Set/Reset Capability Easing Control Application Interface
- 40 Pin Dual In-Line Package
- Reduces System Package Count

The 8255 is a general purpose programmable I/O device designed for use with both the 8008 and 8080 microprocessors. It has 24 I/O pins which may be individually programmed in two groups of twelve and used in three major modes of operation. In the first mode (Mode 0), each group of twelve I/O pins may be programmed in sets of 4 to be input or output. In Mode 1, the second mode, each group may be programmed to have 8 lines of input or output. Of the remaining four pins three are used for handshaking and interrupt control signals. The third mode of operation (Mode 2) is a Bidirectional Bus mode which uses 8 lines for a bidirectional bus; and five lines, borrowing one from the other group, for handshaking.

Other features of the 8255 include bit set and reset capability and the ability to source 1mA of current at 1.5 volts. This allows darlington transistors to be directly driven for applications such as printers and high voltage displays.

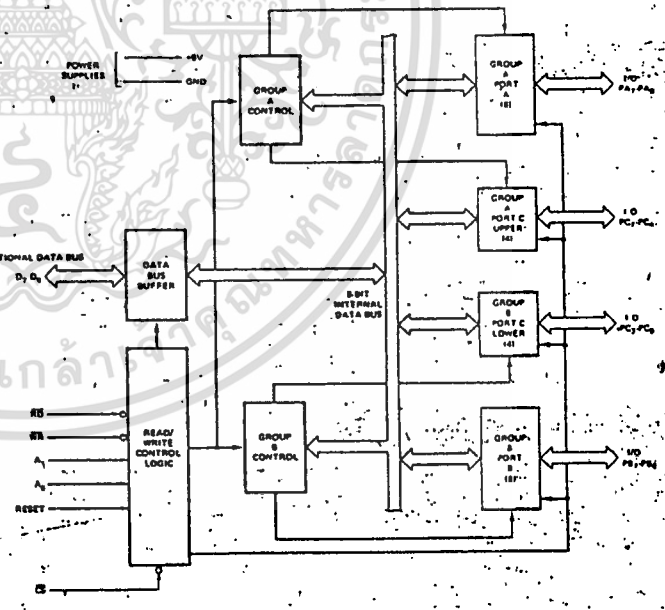
PIN CONFIGURATION



PIN NAMES

D_7–D_0	DATA BUS (BI-DIRECTIONAL)
RESET	RESET INPUT
$\overline{CS}$	CHIP SELECT
$\overline{RD}$	READ INPUT
$\overline{WR}$	WRITE INPUT
A0, A1	PORT ADDRESS
PA7-PA0	PORT A (BIT)
PB7-PB0	PORT B (BIT)
PC7-PC0	PORT C (BIT)
V_{CC}	+5 VOLTS
GND	0 VOLTS

8255 BLOCK DIAGRAM



8255 BASIC FUNCTIONAL DESCRIPTION

General

The 8255 is a Programmable Peripheral Interface (PPI) device designed for use in 8080 Microcomputer Systems. Its function is that of a general purpose I/O component to interface peripheral equipment to the 8080 system bus. The functional configuration of the 8255 is programmed by the system software so that normally no external logic is necessary to interface peripheral devices or structures.

Data Bus Buffer

This 3-state, bi-directional, eight bit buffer is used to interface the 8255 to the 8080 system data bus. Data is transmitted or received by the buffer upon execution of INput or OUTput instructions by the 8080 CPU. Control Words and Status information are also transferred through the Data Bus buffer.

Read/Write and Control Logic

The function of this block is to manage all of the internal and external transfers of both Data and Control or Status words. It accepts inputs from the 8080 CPU Address and Control busses and in turn, issues commands to both of the Control Groups.

($\overline{CS}$)

Chip Select: A "low" on this input pin enables the communication between the 8255 and the 8080 CPU.

($\overline{RD}$)

Read: A "low" on this input pin enables the 8255 to send the Data or Status information to the 8080 CPU on the Data Bus. In essence, it allows the 8080 CPU to "read from" the 8255.

($\overline{WR}$)

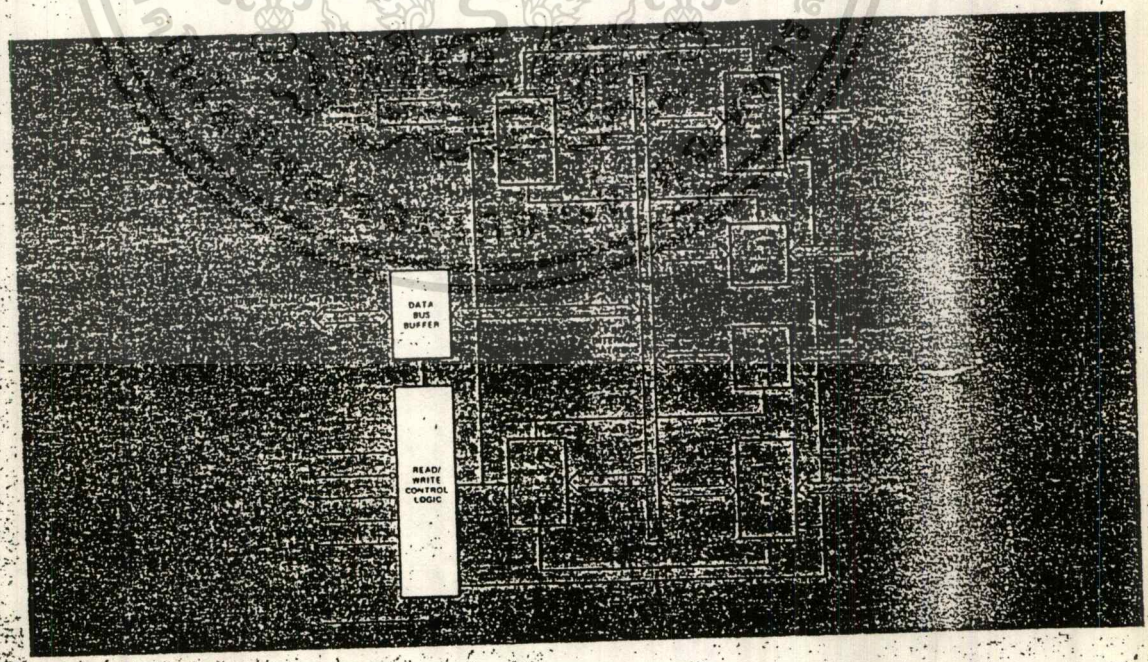
Write: A "low" on this input pin enables the 8080 CPU to write Data or Control words into the 8255.

(A_0 and A_1)

Port Select 0 and Port Select 1: These input signals, in conjunction with the $\overline{RD}$ and $\overline{WR}$ inputs, control the selection of one of the three ports or the Control Word Register. They are normally connected to the least significant bits of the Address Bus (A_0 and A_1).

8255 BASIC OPERATION

A_1	A_0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	INPUT OPERATION (READ)
0	0	0	1	0	PORT A → DATA BUS
0	1	0	1	0	PORT B → DATA BUS
1	0	0	1	0	PORT C → DATA BUS
					OUTPUT OPERATION (WRITE)
0	0	1	0	0	DATA BUS → PORT A
0	1	1	0	0	DATA BUS → PORT B
1	0	1	0	0	DATA BUS → PORT C
1	1	1	0	0	DATA BUS → CONTROL
					DISABLE FUNCTION
X	X	X	X	1	DATA BUS → 3-STATE
1	1	0	1	0	ILLEGAL CONDITION



8255 Block Diagram

(RESET)

Reset: A "high" on this input clears all internal registers including the Control Register and all ports (A, B, C) are set to the input mode.

Group A and Group B Controls

The functional configuration of each port is programmed by the systems software. In essence, the 8080 CPU "outputs" a control word to the 8255. The control word contains information such as "mode", "bit set", "bit reset", etc. that initializes the functional configuration of the 8255.

Each of the Control blocks (Group A and Group B) accepts "commands" from the Read/Write Control Logic, receives "control words" from the internal data bus and issues the proper commands to its associated ports.

Control Group A — Port A and Port C upper (C7-C4)
Control Group B — Port B and Port C lower (C3-C0)

The Control Word Register can Only be written into. No Read operation of the Control Word Register is allowed.

Ports A, B, and C

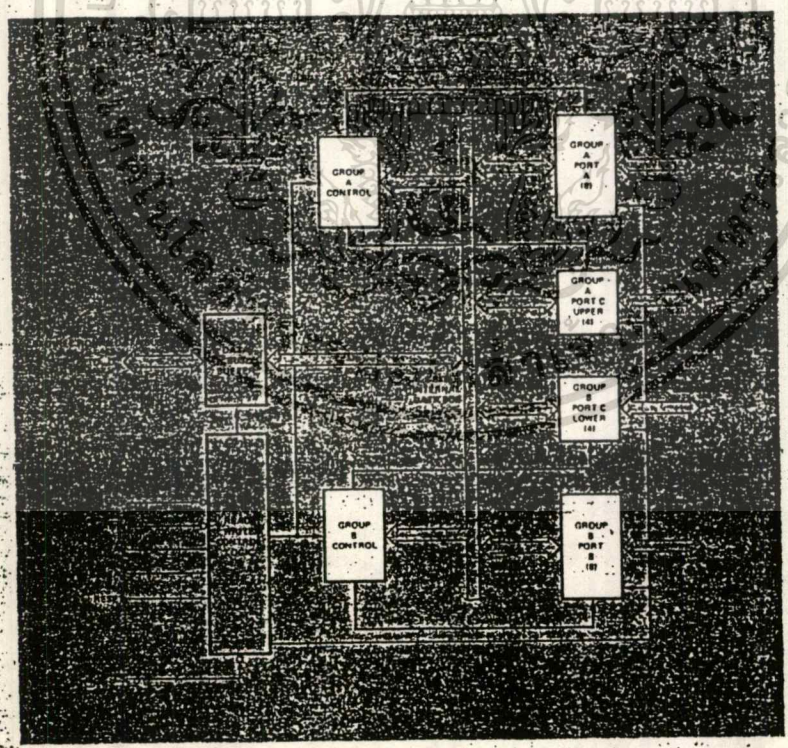
The 8255 contains three 8-bit ports (A, B, and C). All can be configured in a wide variety of functional characteristics by the system software but each has its own special features or "personality" to further enhance the power and flexibility of the 8255.

Port A: One 8-bit data output latch/buffer and one 8-bit data input latch.

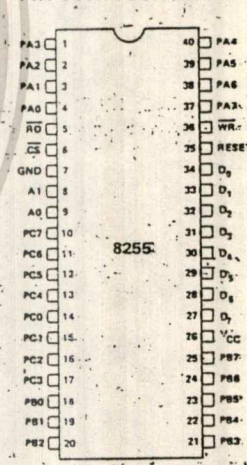
Port B: One 8-bit data input/output latch/buffer and one 8-bit data input buffer.

Port C: One 8-bit data output latch/buffer and one 8-bit data input buffer (no latch for input). This port can be divided into two 4-bit ports under the mode control. Each 4-bit port contains a 4-bit latch and it can be used for the control signal outputs and status signal inputs in conjunction with Ports A and B.

8255 BLOCK DIAGRAM



PIN CONFIGURATION



PIN NAMES

D_7–D_0	DATA BUS (BI-DIRECTIONAL)
RESET	RESET INPUT
$\overline{CS}$	CHIP SELECT
$\overline{RD}$	READ INPUT
$\overline{WR}$	WRITE INPUT
A0, A1	PORT ADDRESS
PA7-PA0	PORT A (BIT)
PB7-PB0	PORT B (BIT)
PC7-PC0	PORT C (BIT)
V_{CC}	+5 VOLTS
GND	0 VOLTS

8255 DETAILED OPERATIONAL DESCRIPTION

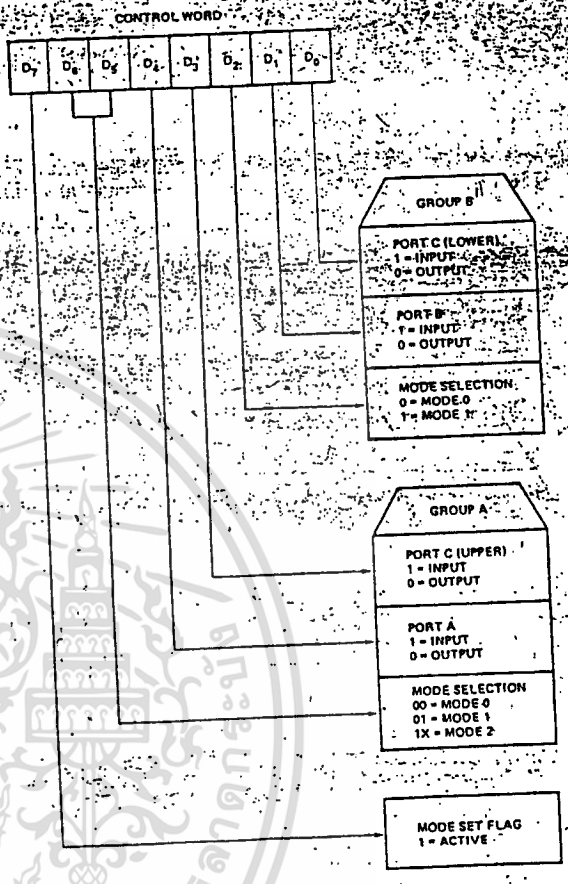
Mode Selection

There are three basic modes of operation that can be selected by the system software:

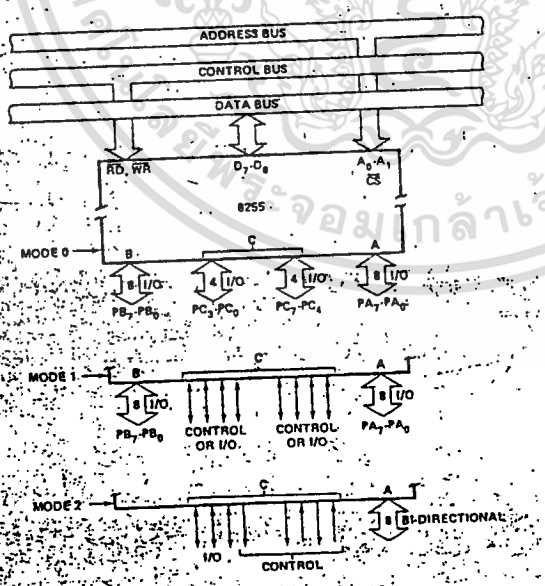
Mode 0 — Basic Input/Output
Mode 1 — Strobed Input/Output
Mode 2 — Bi-Directional Bus

When the RESET input goes "high" all ports will be set to the input mode (i.e., all 24 lines will be in the high impedance state). After the RESET is removed, the 8255 can remain in the input mode with no additional initialization required. During the execution of the system program any of the other modes may be selected using a single OUTput instruction. This allows a single 8255 to service a variety of peripheral devices with a simple software maintenance routine.

The modes for Port A and Port B can be separately defined, while Port C is divided into two portions as required by the Port A and Port B definitions. All of the output registers, including the status flip-flops, will be reset whenever the mode is changed. Modes may be combined so that their functional definition can be "tailored" to almost any I/O structure. For instance; Group B can be programmed in Mode 0 to monitor simple switch closings or display computational results, Group A could be programmed in Mode 1 to monitor a keyboard or tape reader on an interrupt-driven basis.



Mode Definition Format

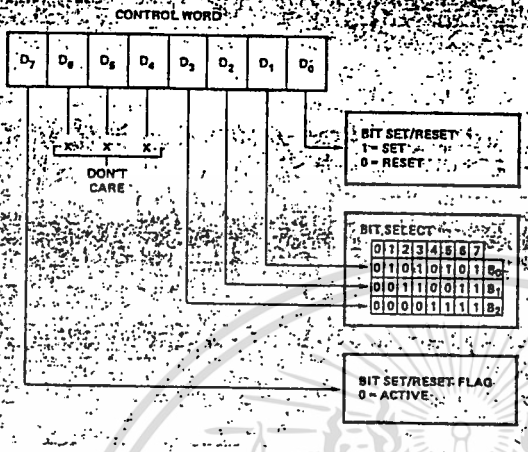


Basic Mode Definitions and Bus Interface

The Mode definitions and possible Mode combinations may seem confusing at first but after a cursory review of the complete device operation a simple, logical I/O approach will surface. The design of the 8255 has taken into account things such as efficient PC board layout, control signal definition vs PC layout and complete functional flexibility to support almost any peripheral device with no external logic. Such design represents the maximum use of the available pins.

Single Bit Set/Reset Feature

Any of the eight bits of Port C can be Set or Reset using a single OUTput instruction. This feature reduces software requirements in Control-based applications.



Bit Set/Reset Format

When Port C is being used as status/control for Port A or B, these bits can be set or reset by using the Bit Set/Reset operation just as if they were data output ports.

Interrupt Control Functions

When the 8255 is programmed to operate in Mode 1 or Mode 2, control signals are provided that can be used as interrupt request inputs to the CPU. The interrupt request signals, generated from Port C, can be inhibited or enabled by setting or resetting the associated INTE flip-flop, using the Bit set/reset function of Port C.

This function allows the Programmer to disallow or allow a specific I/O device to interrupt the CPU without effecting any other device in the interrupt structure.

INTE flip-flop definition:

(BIT-SET) — INTE is SET — Interrupt enable

(BIT-RESET) — INTE is RESET — Interrupt disable

Note: All Mask flip-flops are automatically reset during mode selection and device Reset.

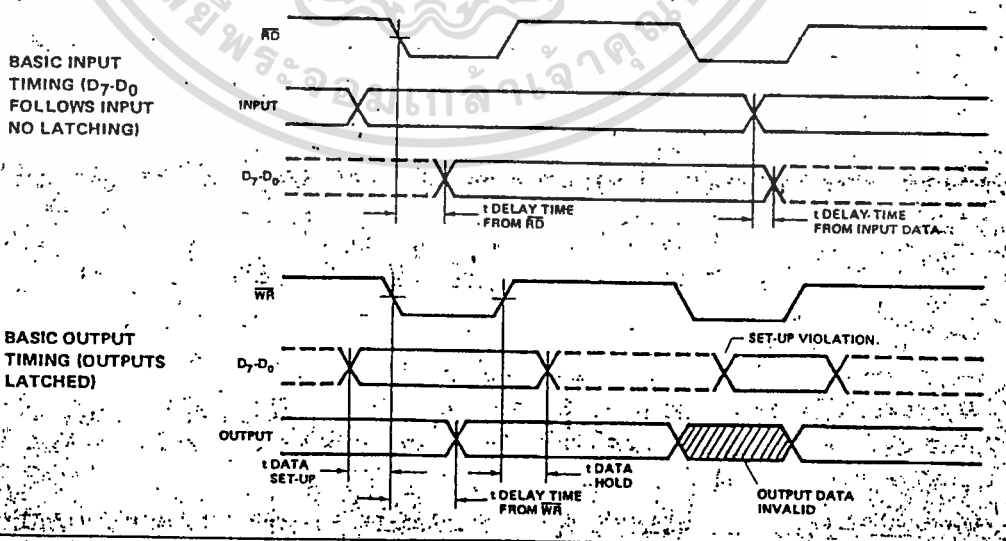
Operating Modes

Mode 0 (Basic Input/Output)

This functional configuration provides simple Input and Output operations for each of the three ports. No "handshaking" is required, data is simply written to or read from a specified port.

Mode 0 Basic Functional Definitions:

- Two 8-bit ports and two 4-bit ports.
- Any port can be input or output.
- Outputs are latched.
- Inputs are not latched.
- 16 different Input/Output configurations are possible in this Mode.



Mode 0 Timing

MODE 0 PORT DEFINITION CHART

	A			B	GROUP A			GROUP B	
D_4	D_3	D_1	D_0	PORT A	PORT C (UPPER)	#	PORT B	PORT C (LOWER)
0	0	0	0	OUTPUT	OUTPUT	0	OUTPUT	OUTPUT
0	0	0	1	OUTPUT	OUTPUT	1	OUTPUT	INPUT
0	0	1	0	OUTPUT	OUTPUT	2	INPUT	OUTPUT
0	0	1	1	OUTPUT	OUTPUT	3	INPUT	INPUT
0	1	0	0	OUTPUT	INPUT	4	OUTPUT	OUTPUT
0	1	0	1	OUTPUT	INPUT	5	OUTPUT	INPUT
0	1	1	0	OUTPUT	INPUT	6	INPUT	OUTPUT
0	1	1	1	OUTPUT	INPUT	7	INPUT	INPUT
1	0	0	0	INPUT	OUTPUT	8	OUTPUT	OUTPUT
1	0	0	1	INPUT	OUTPUT	9	OUTPUT	INPUT
1	0	1	0	INPUT	OUTPUT	10	INPUT	OUTPUT
1	0	1	1	INPUT	OUTPUT	11	INPUT	INPUT
1	1	0	0	INPUT	INPUT	12	OUTPUT	OUTPUT
1	1	0	1	INPUT	INPUT	13	OUTPUT	INPUT
1	1	1	0	INPUT	INPUT	14	INPUT	OUTPUT
1	1	1	1	INPUT	INPUT	15	INPUT	INPUT

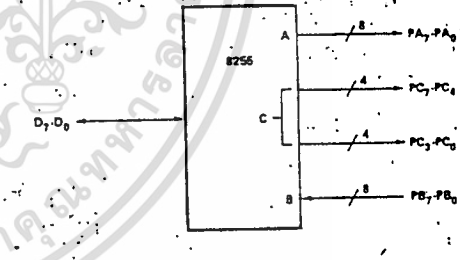
MODE 0 CONFIGURATIONS

CONTROL WORD #0

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	0	0	0	0

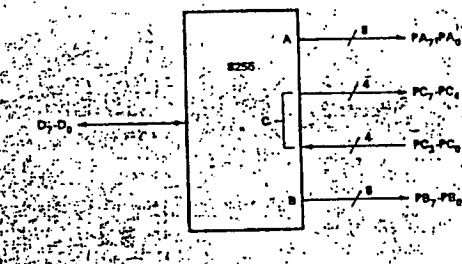
CONTROL WORD #2

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	0	0	1	0



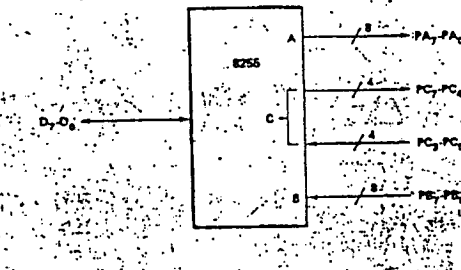
CONTROL WORD #1

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	0	0	0	1



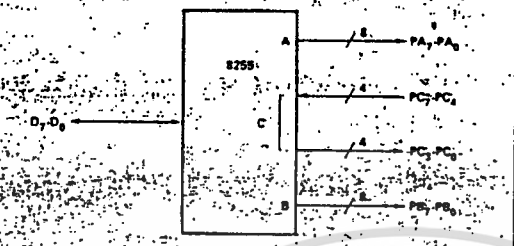
CONTROL WORD #3

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	0	0	1	1



CONTROL WORD #6

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	0	0



CONTROL WORD #8

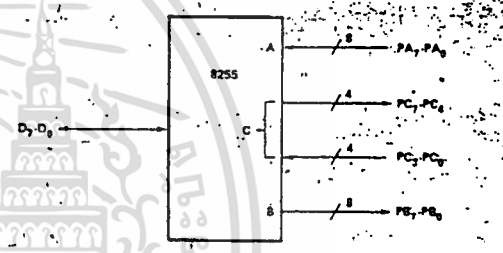
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	0	0

CONTROL WORD #9

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	0	1

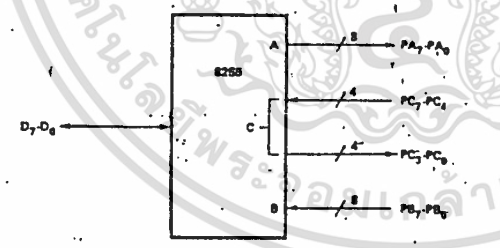
CONTROL WORD #10

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	1	0



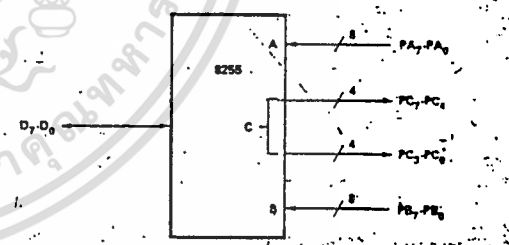
CONTROL WORD #7

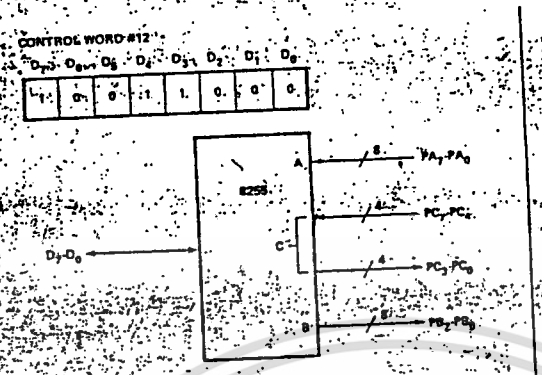
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	1	1

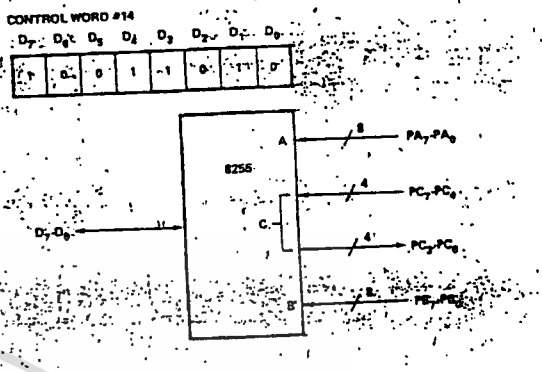


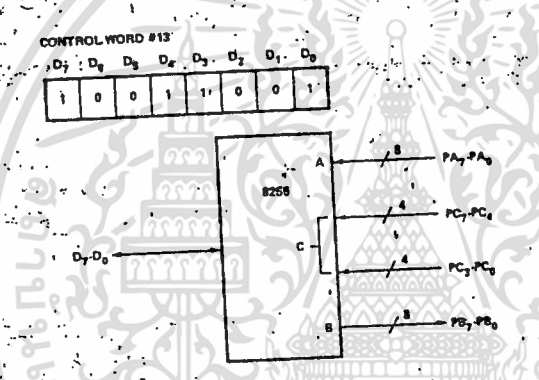
CONTROL WORD #11

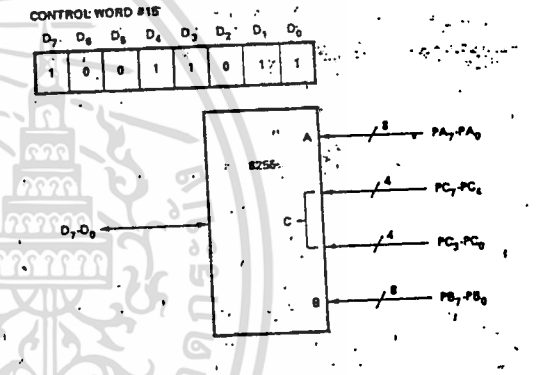
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	1	1











Operating Modes

Mode 1 (Strobed Input/Output)

This functional configuration provides a means for transferring I/O data to or from a specified port in conjunction with strobes or "handshaking" signals. In Mode 1, Port A and Port B use the lines on Port C to generate or accept these "handshaking" signals.

Mode 1 Basic Functional Definitions:

- Two Groups (Group A and Group B)
- Each group contains one 8-bit data port and one 4-bit control/data port.
- The 8-bit data port can be either input or output. Both inputs and outputs are latched.
- The 4-bit port is used for control and status of the 8-bit data port.

Input Control Signal Definition

$\overline{STB}$ (Strobe Input)

A "low" on this input loads data into the input latch.

IBF (Input Buffer Full F/F)

A "high" on this output indicates that the data has been loaded into the input latch; in essence, an acknowledgement. IBF is set by the falling edge of the STB input and is reset by the rising edge of the RD input.

INTR (Interrupt Request)

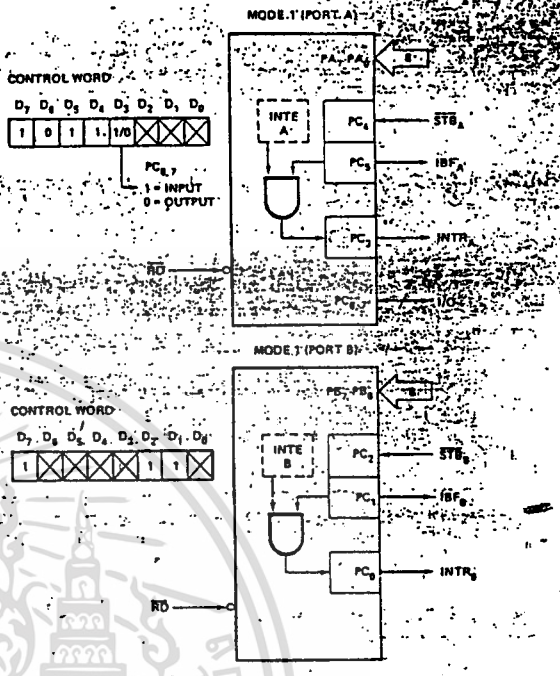
A "high" on this output can be used to interrupt the CPU when an input device is requesting service. INTR is set by the rising edge of $\overline{STB}$ if IBF is a "one" and INTE is a "one". It is reset by the falling edge of $\overline{RD}$. This procedure allows an input device to request service from the CPU by simply strobing its data into the port.

INTE A

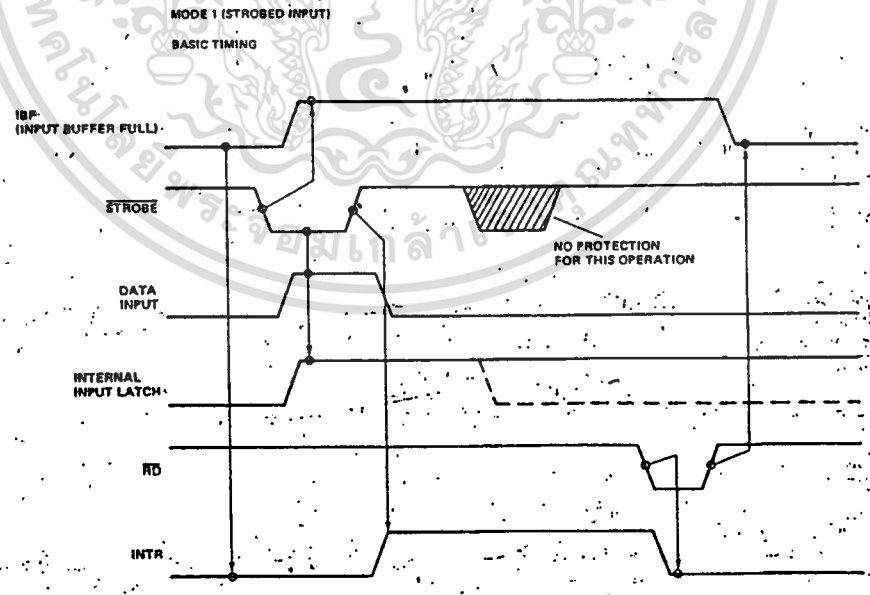
Controlled by bit set/reset of PC_4.

INTE B

Controlled by bit set/reset of PC_2.



Mode 1 Input



Basic Timing Input

Output Control Signal Definition

$\overline{OBF}$ (Output Buffer Full F/F)

The $\overline{OBF}$ output will go "low" to indicate that the CPU has written data out to the specified port. The OBF F/F will be set by the rising edge of the WR input and reset by the falling edge of the $\overline{ACK}$ input signal.

$\overline{ACK}$ (Acknowledge Input)

A "low" on this input informs the 8255 that the data from Port A or Port B has been accepted. In essence, a response from the peripheral device indicating that it has received the data output by the CPU.

INTR (Interrupt Request)

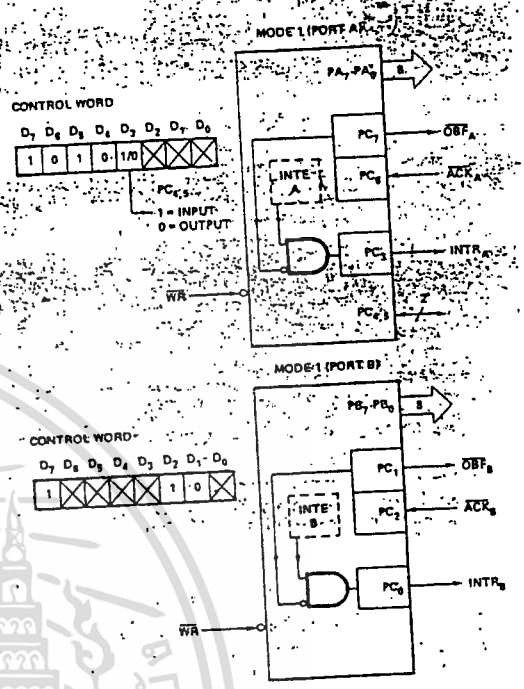
A "high" on this output can be used to interrupt the CPU when an output device has accepted data transmitted by the CPU. INTR is set by the rising edge of $\overline{ACK}$ if $\overline{OBF}$ is a "one" and INTE is a "one". It is reset by the falling edge of $\overline{WR}$.

INTE A

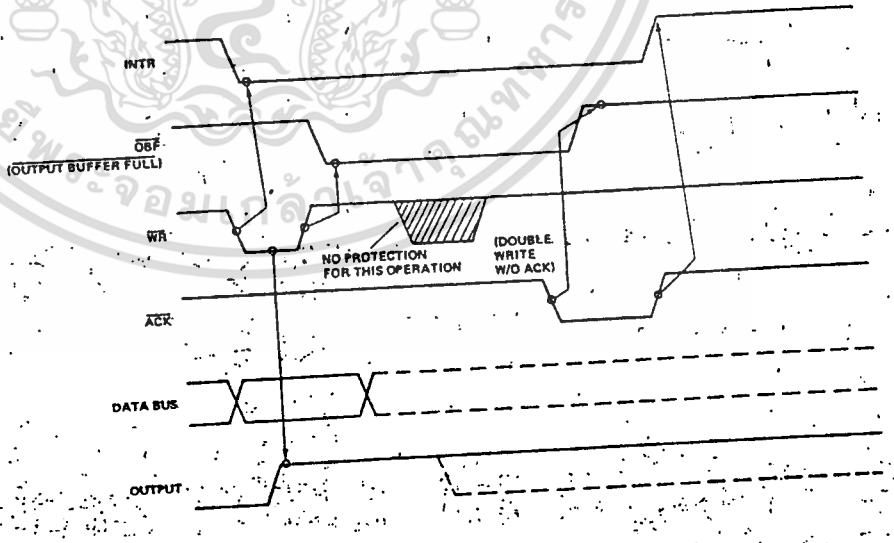
Controlled by bit set/reset of PC_6.

INTE B

Controlled by bit set/reset of PC_2.



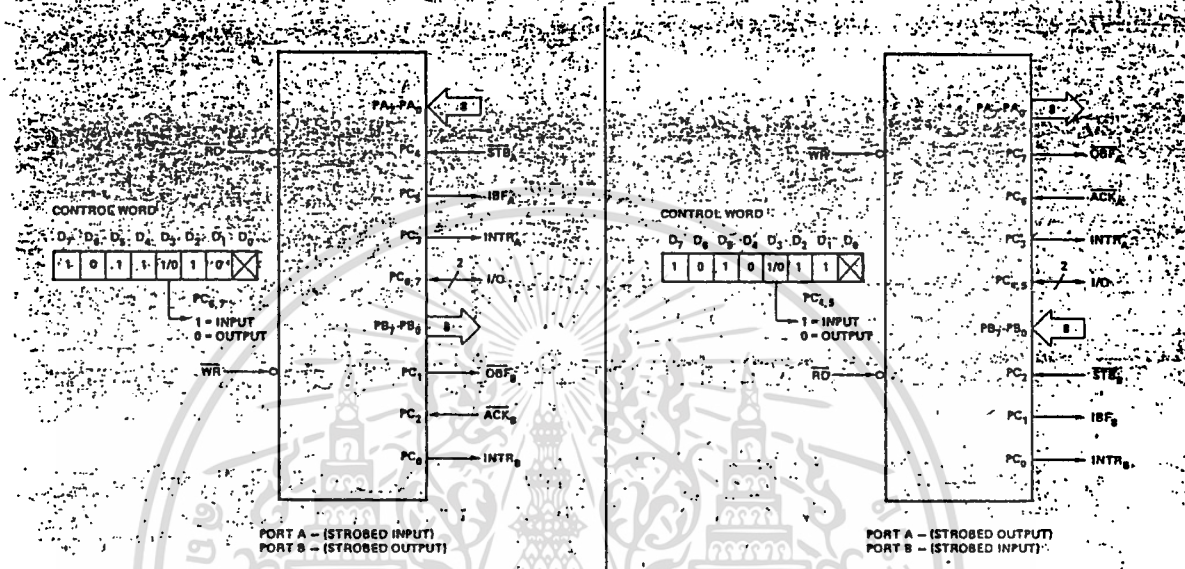
Mode 1 Output



Basic Timing Output

Combinations of Mode 1

Port A and Port B can be individually defined as input or output in Mode 1 to support a wide variety of strobed I/O applications.



Operating Modes

Mode 2 (Strobed Bi-Directional Bus I/O)

This functional configuration provides a means for communicating with a peripheral device or structure on a single-8-bit bus for both transmitting and receiving data (bi-directional bus I/O). "Handshaking" signals are provided to maintain proper bus flow discipline in a similar manner to Mode 1. Interrupt generation and enable/disable functions are also available.

Mode 2 Basic Functional Definitions:

- Used in Group A only.
- One 8-bit, bi-directional bus Port (Port A) and a 5-bit control Port (Port C).
- Both inputs and outputs are latched.
- The 5-bit control port (Port C) is used for control and status for the 8-bit, bi-directional bus port (Port A).

Bi-Directional Bus I/O Control Signal Definition

INTR (Interrupt Request)

A high on this output can be used to interrupt the CPU for both input or output operations.

$\overline{OBF}$ (Output Buffer Full)

The $\overline{OBF}$ output will go "low" to indicate that the CPU has written data out to Port A.

$\overline{ACK}$ (Acknowledge)

A "low" on this input enables the tri-state output buffer of Port A to send out the data. Otherwise, the output buffer will be in the high-impedance state.

INTE 1 (The INTE Flip-Flop associated with $\overline{OBF}$)

Controlled by bit set/reset of PC_6.

Input Operations

$\overline{STB}$ (Strobe Input)

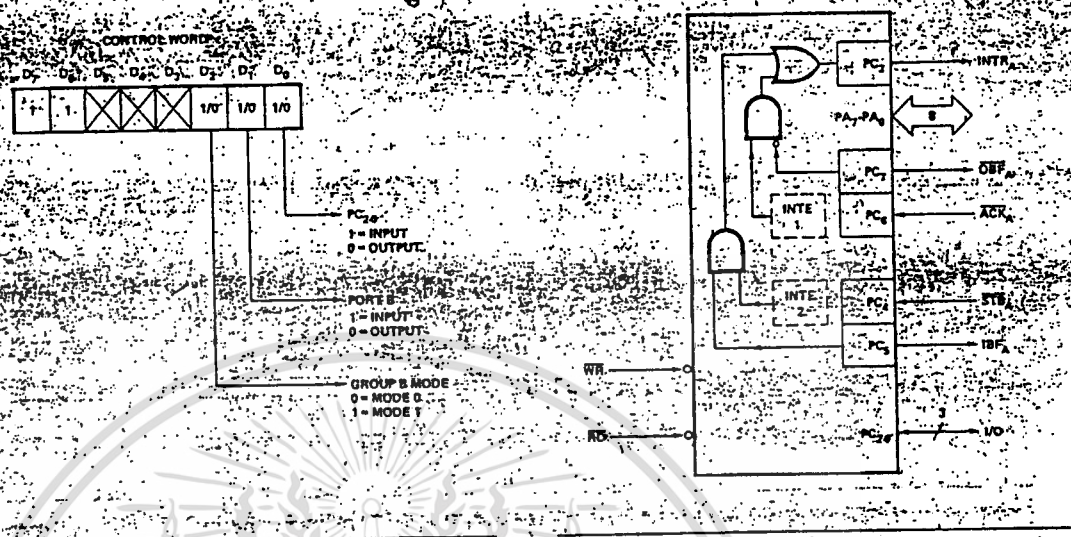
A "low" on this input loads data into the input latch.

IBF (Input Buffer Full F/F)

A "high" on this output indicates that data has been loaded into the input latch.

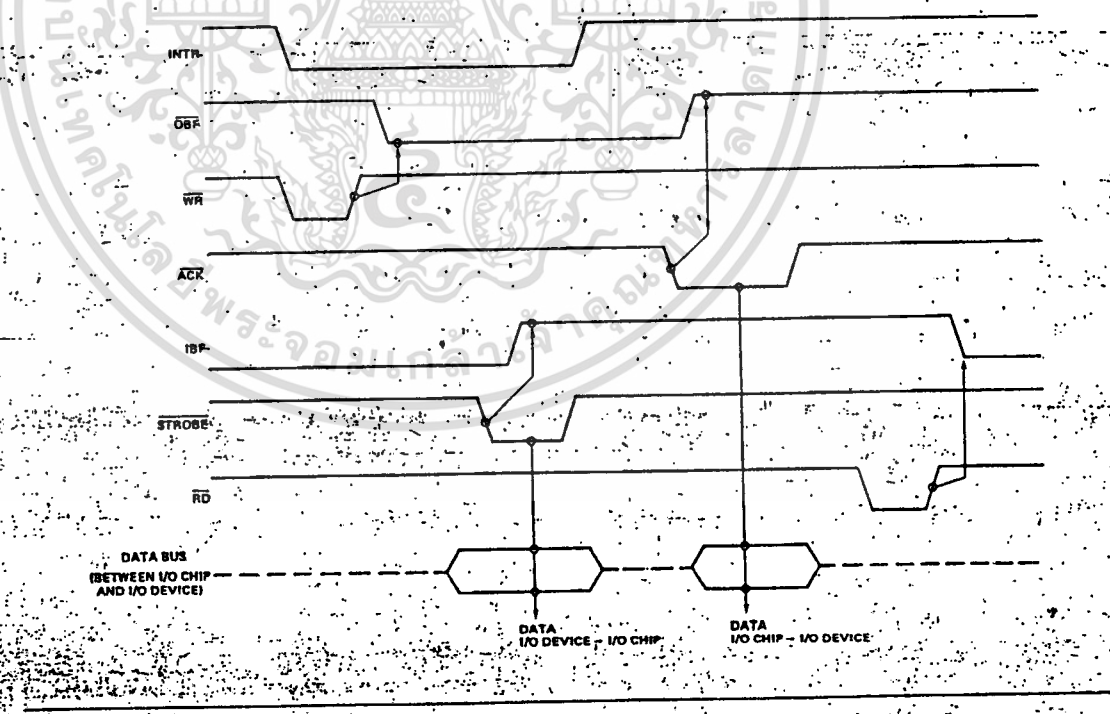
INTE 2 (The INTE Flip-Flop associated with IBF)

Controlled by bit set/reset of PC_4.



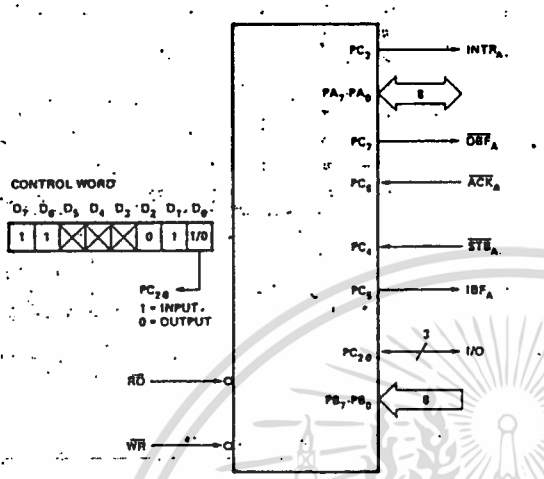
Mode 2 Control Word

Mode 2

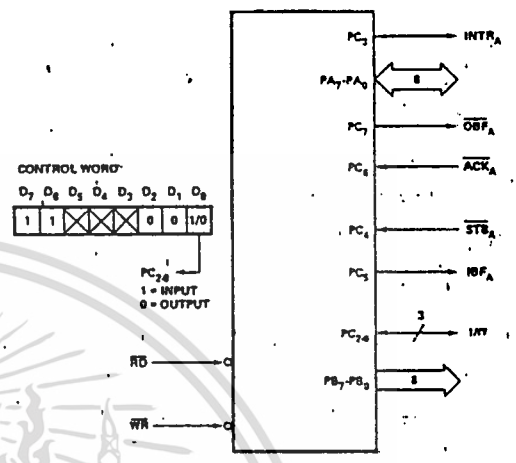


Mode 2 (Bi-directional) Timing

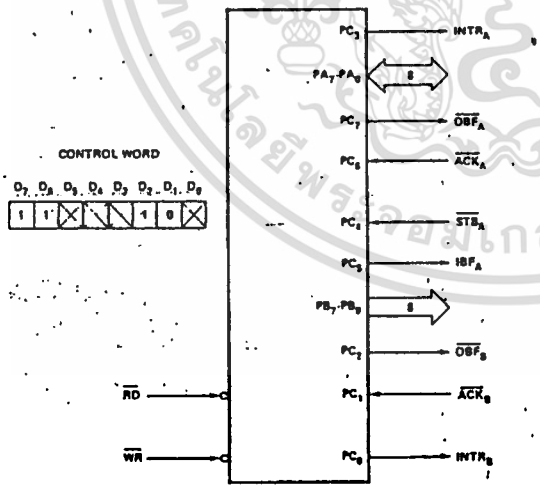
MODE 2 AND MODE 0 (INPUT)



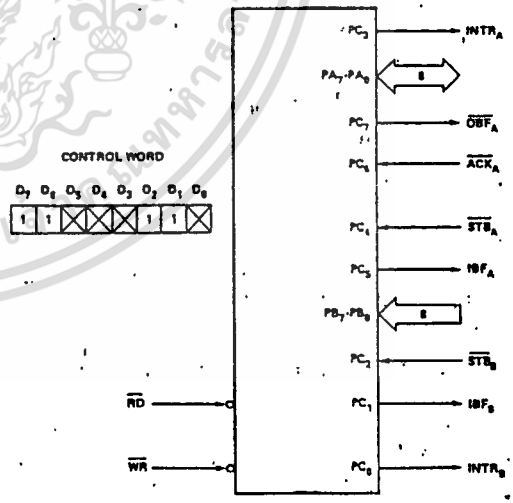
MODE 2 AND MODE 0 (OUTPUT)



MODE 2 AND MODE 1 (OUTPUT)



MODE 2 AND MODE 1 (INPUT)



Mode 2 Combinations

MODE DEFINITION SUMMARY TABLE

	MODE 0		MODE 1		MODE 2
	IN	OUT	IN	OUT	GROUP A ONLY
PA_0	IN	OUT	IN	OUT	↔
PA_1	IN	OUT	IN	OUT	↔
PA_2	IN	OUT	IN	OUT	↔
PA_3	IN	OUT	IN	OUT	↔
PA_4	IN	OUT	IN	OUT	↔
PA_5	IN	OUT	IN	OUT	↔
PA_6	IN	OUT	IN	OUT	↔
PA_7	IN	OUT	IN	OUT	↔
PB_0	IN	OUT	IN	OUT	—
PB_1	IN	OUT	IN	OUT	—
PB_2	IN	OUT	IN	OUT	—
PB_3	IN	OUT	IN	OUT	—
PB_4	IN	OUT	IN	OUT	—
PB_5	IN	OUT	IN	OUT	—
PB_6	IN	OUT	IN	OUT	—
PB_7	IN	OUT	IN	OUT	—
PC_0	IN	OUT	$INTR_B$	$INTR_B$	I/O
PC_1	IN	OUT	IBF_B	$\overline{OBF}_B$	I/O
PC_2	IN	OUT	$\overline{STB}_B$	$\overline{ACK}_B$	I/O
PC_3	IN	OUT	$INTR_A$	$INTR_A$	$INTR_A$
PC_4	IN	OUT	$\overline{STB}_A$	I/O	$\overline{STB}_A$
PC_5	IN	OUT	IBF_A	I/O	IBF_A
PC_6	IN	OUT	I/O	$\overline{ACK}_A$	$\overline{ACK}_A$
PC_7	IN	OUT	I/O	$\overline{OBF}_A$	$\overline{OBF}_A$

MODE 0 OR MODE 1 ONLY

Special Mode Combination Considerations

There are several combinations of modes when not all of the bits in Port C are used for control or status. The remaining bits can be used as follows:

If Programmed as Inputs —
All input lines can be accessed during a normal Port C read.

If Programmed as Outputs —
Bits in C upper (PC_7-PC_4) must be individually accessed using the bit set/reset function.

Bits in C lower (PC_3-PC_0) can be accessed using the bit set/reset function or accessed as a threesome by writing into Port C.

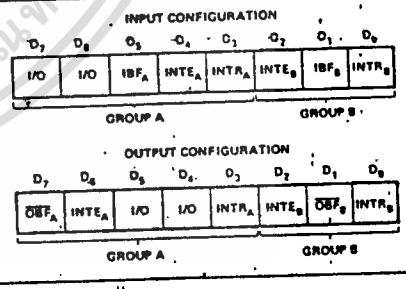
Source Current Capability on Port B and Port C

Any set of eight output buffers, selected randomly from Ports B and C can source 1mA at 1.5 volts. This feature allows the 8255 to directly drive Darlington type drivers and high-voltage displays that require such source current.

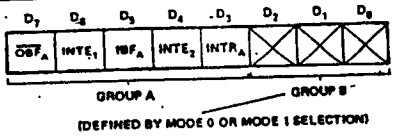
Reading Port C Status

In Mode 0, Port C transfers data to or from the peripheral device. When the 8255 is programmed to function in Modes 1 or 2, Port C generates or accepts "hand-shaking" signals with the peripheral device. Reading the contents of Port C allows the programmer to test or verify the "status" of each peripheral device and change the program flow accordingly.

There is no special instruction to read the status information from Port C. A normal read operation of Port C is executed to perform this function.



Mode 1 Status Word Format

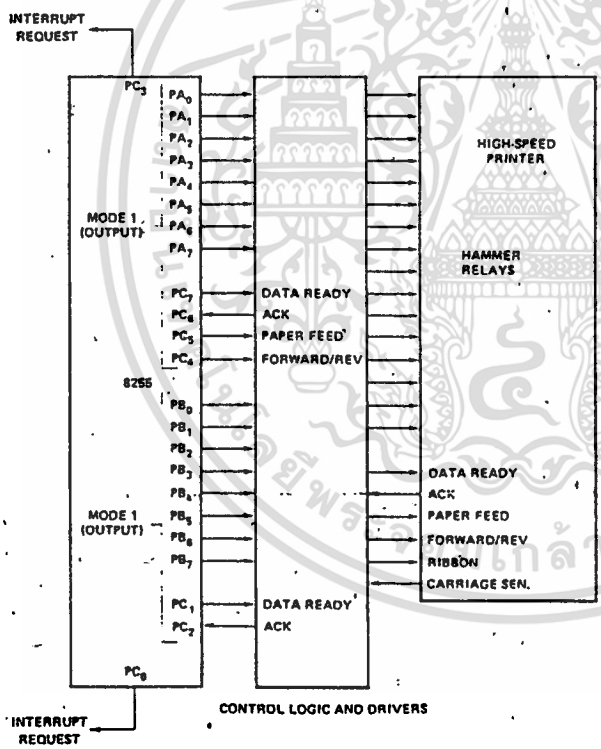


Mode 2 Status Word Format

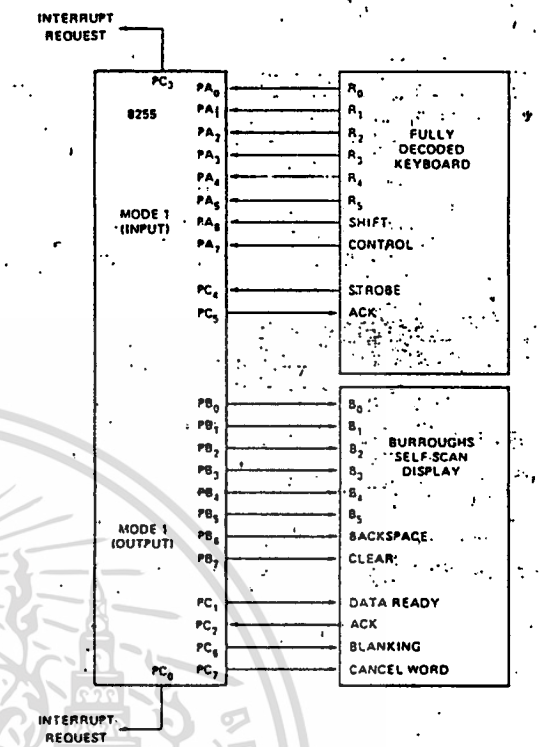
APPLICATIONS OF THE 8255

The 8255 is a very powerful tool for interfacing peripheral equipment to the 8080 microcomputer system. It represents the optimum use of available pins and is flexible enough to interface almost any I/O device without the need for additional external logic.

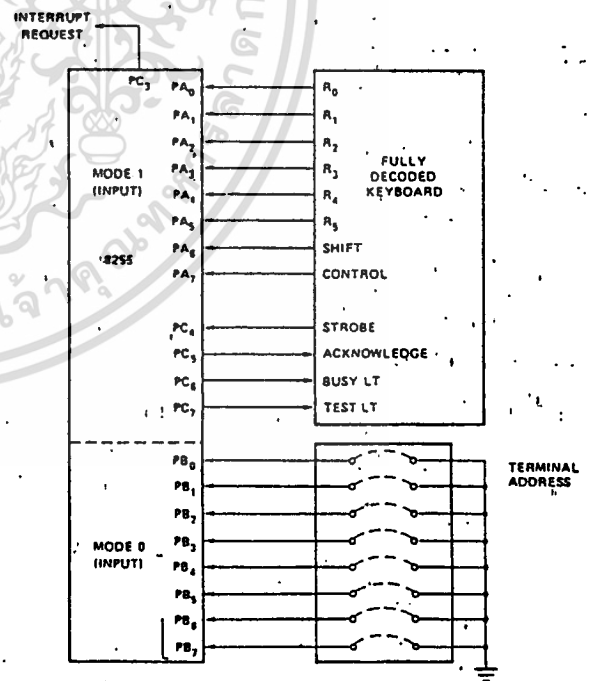
Each peripheral device in a Microcomputer system usually has a "service routine" associated with it. The routine manages the software interface between the device and the CPU. The functional definition of the 8255 is programmed by the I/O service routine and becomes an extension of the systems software. By examining the I/O devices interface characteristics for both data transfer and timing, and matching this information to the examples and tables in the Detailed Operational Description, a control word can easily be developed to initialize the 8255 to exactly "fit" the application. Here are a few examples of typical applications of the 8255.



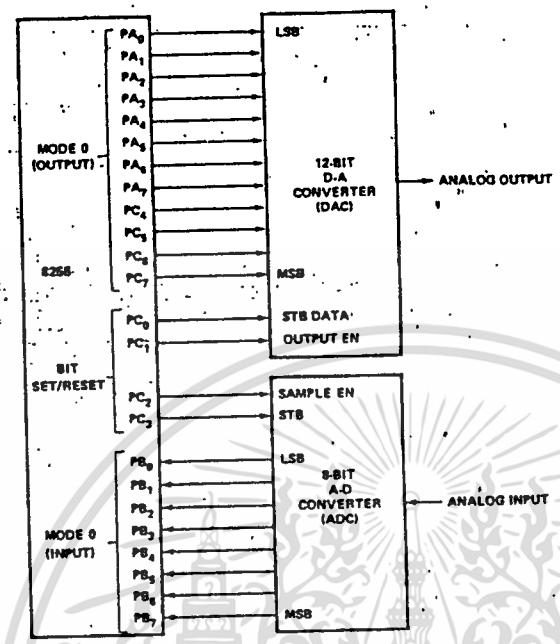
Printer Interface



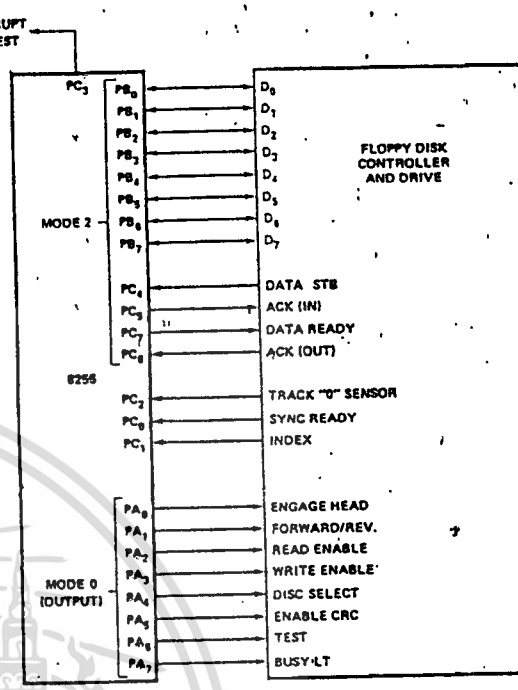
Keyboard and Display Interface



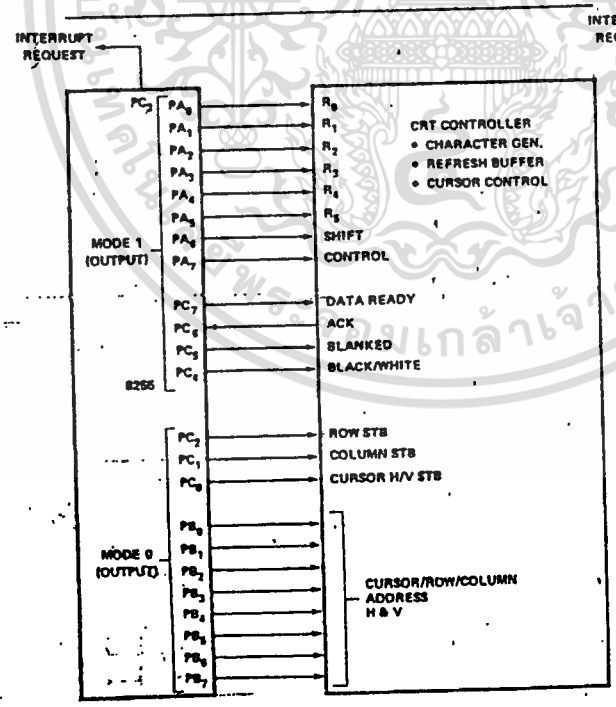
Keyboard and Terminal Address Interface



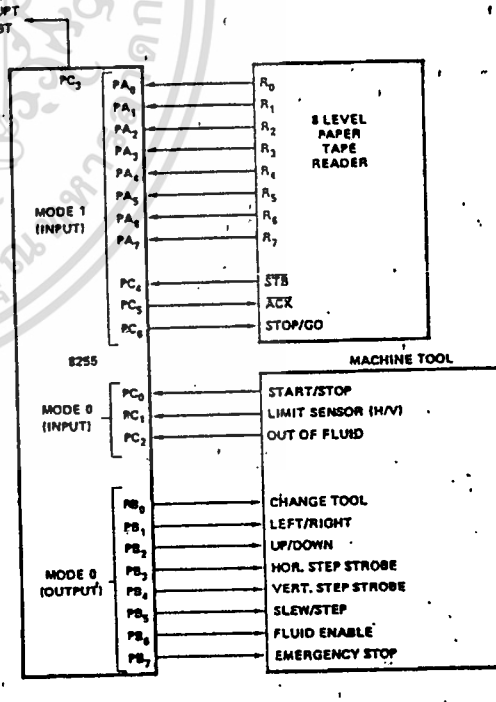
Digital to Analog, Analog to Digital



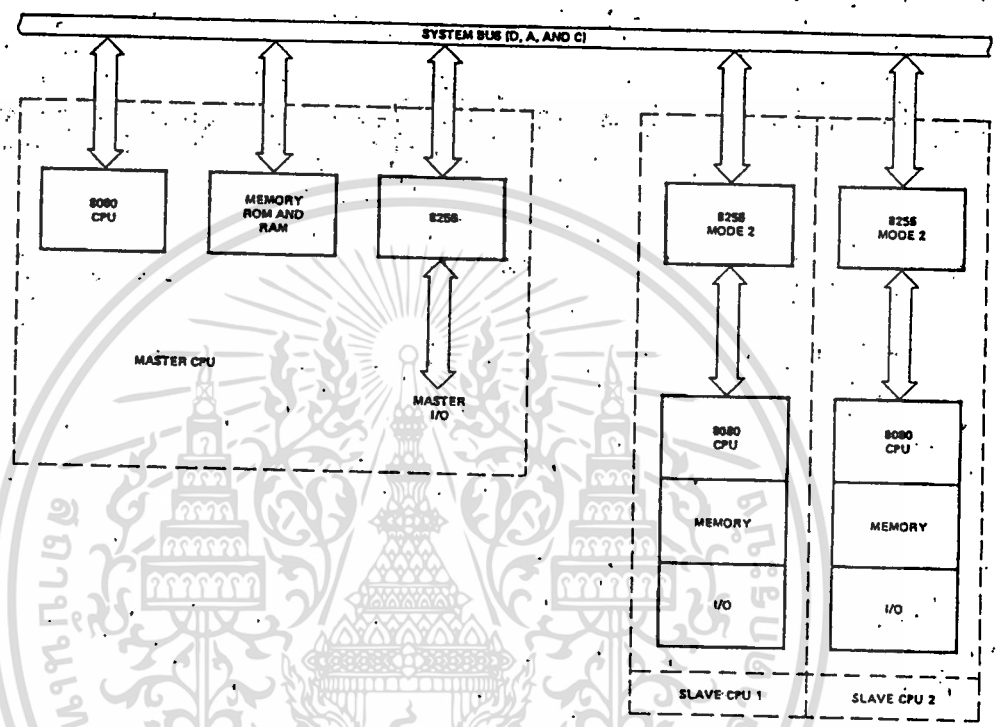
Basic Floppy Disc Interface



Basic CRT Controller Interface



Machine Tool Controller Interface



Distributed Intelligence Multi-Processor Interface

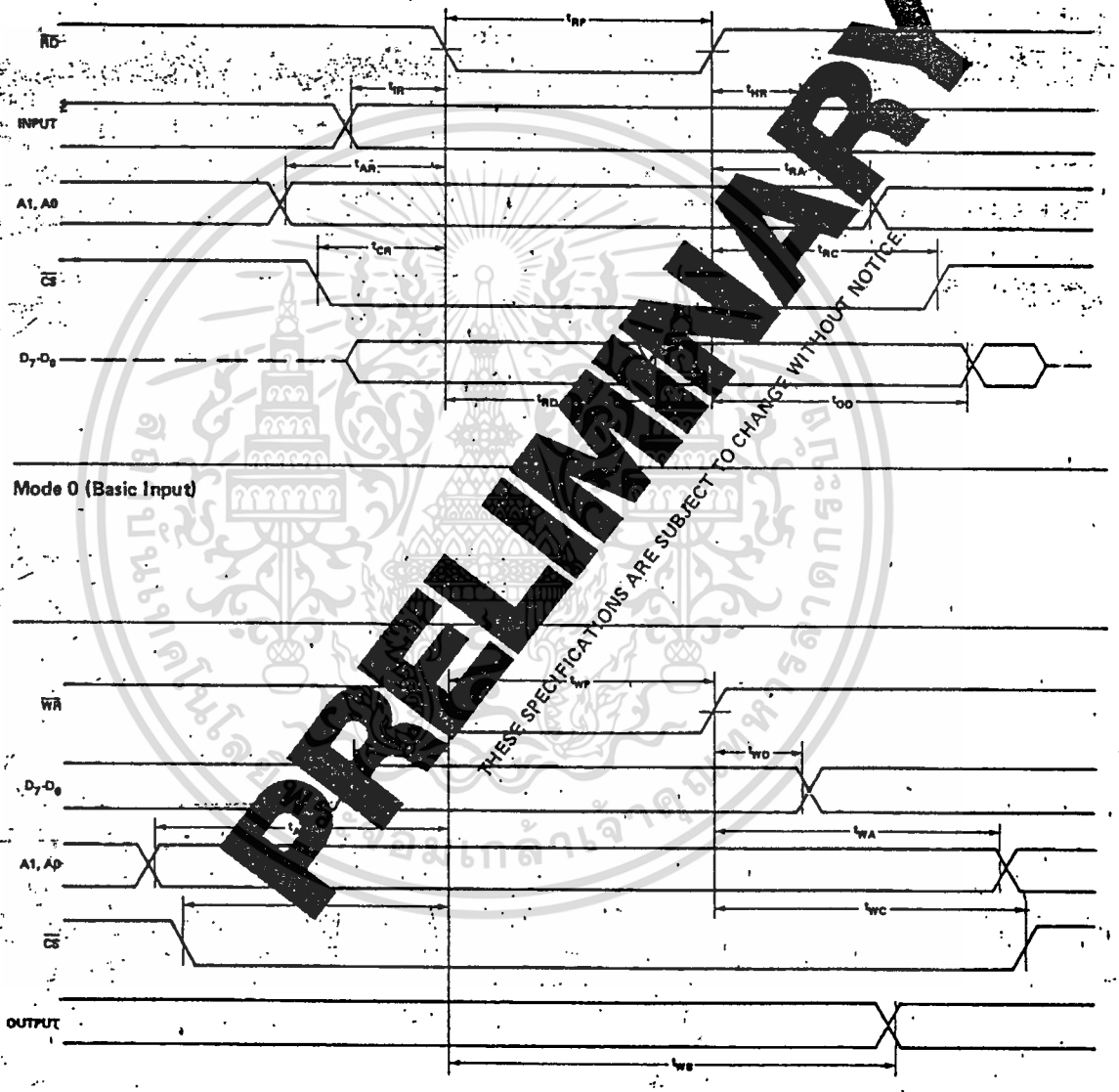
D.C. CHARACTERISTICS T_A = 0°C to 70°C; V_{CC} = +5V ±5%; V_{SS} = 0V

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Conditions
V_{IL}	Input Low Voltage			.8	V	
V_{IH}	Input High Voltage	2.0			V	
V_{OL}	Output Low Voltage			.4	V	I_{OL} = 1.6mA
V_{OH}	Output High Voltage	2.4			V	I_{OH} = -50μA (-200μA for D.B. Port)
I_{OH}[1]	Darlington Drive Current		2.0		mA	V_{OH} = 1.5V, R_{EXT} = 390Ω
I_{CC}	Power Supply Current		40		mA	

NOTE:
1. Available on 8 pins only.

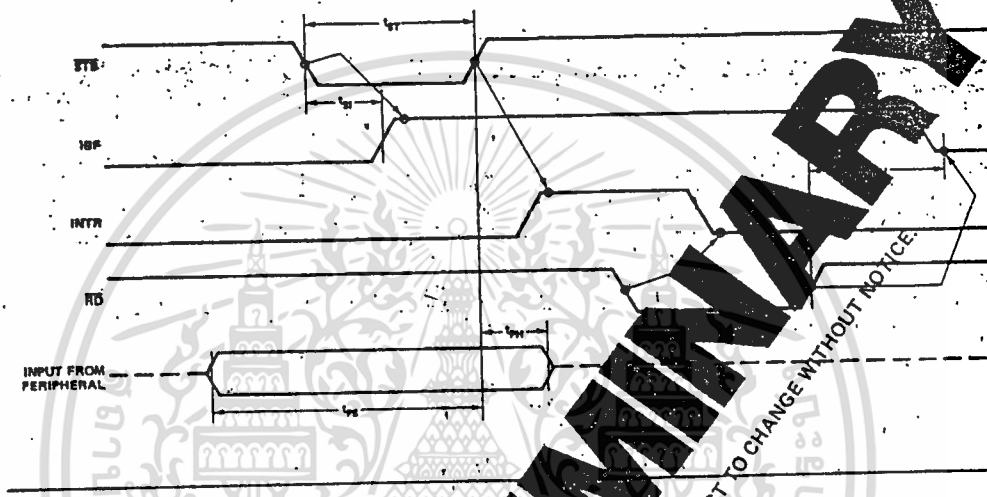
A.C. CHARACTERISTICS T_A = 0°C to 70°C; V_{CC} = +5V ±5%; V_{SS} = 0V

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Condition
t_{WP}	Pulse Width of $\overline{WR}$		250		ns	
t_{DW}	Time D.B. Stable Before $\overline{WR}$		10		ns	
t_{WD}	Time D.B. Stable After $\overline{WR}$		10		ns	
t_{AW}	Time Address Stable Before $\overline{WR}$		25		ns	
t_{WA}	Time Address Stable After $\overline{WR}$		10		ns	
t_{CW}	Time CS Stable Before $\overline{WR}$		25		ns	
t_{WC}	Time CS Stable After $\overline{WR}$		10		ns	
t_{WB}	Delay From $\overline{WR}$ To Output		200		ns	
t_{RP}	Pulse Width of $\overline{RD}$		300		ns	
t_{IR}	$\overline{RD}$ Set-Up Time		50		ns	
t_{HR}	Input Hold Time		10		ns	
t_{RD}	Delay From $\overline{RD}$ = 0 To System Bus		200		ns	
t_{OD}	Delay From $\overline{RD}$ = 1 To System Bus		100		ns	
t_{AR}	Time Address Stable Before $\overline{RD}$		25		ns	
t_{CR}	Time $\overline{CS}$ Stable Before $\overline{RD}$		25		ns	
t_{AK}	Width Of $\overline{ACK}$ Pulse		100		ns	
t_{ST}	Width Of $\overline{STB}$ Pulse		100		ns	
t_{PS}	Set-Up Time For Peripheral		200		ns	
t_{PH}	Hold Time For Peripheral		10		ns	
t_{RA}	Hold Time for A_1A_0 After $\overline{RD}$ = 1		10		ns	
t_{RC}	Hold Time For CS After $\overline{RD}$ = 1		10		ns	
t_{AD}	Time From $\overline{ACK}$ = 0 To Output (Mode 2)		200		ns	
t_{KD}	Time From $\overline{ACK}$ = 1 To Output Floating		250		ns	
t_{WO}	Time From $\overline{WR}$ = 1 To $\overline{OBF}$ = 0		50		ns	
t_{AO}	Time From $\overline{ACK}$ = 0 To $\overline{OBF}$ = 1		200		ns	
t_{SI}	Time From $\overline{STB}$ = 0 To IBF		200		ns	
t_{RI}	Time From $\overline{RD}$ = 1 To IBF = 0		200		ns	

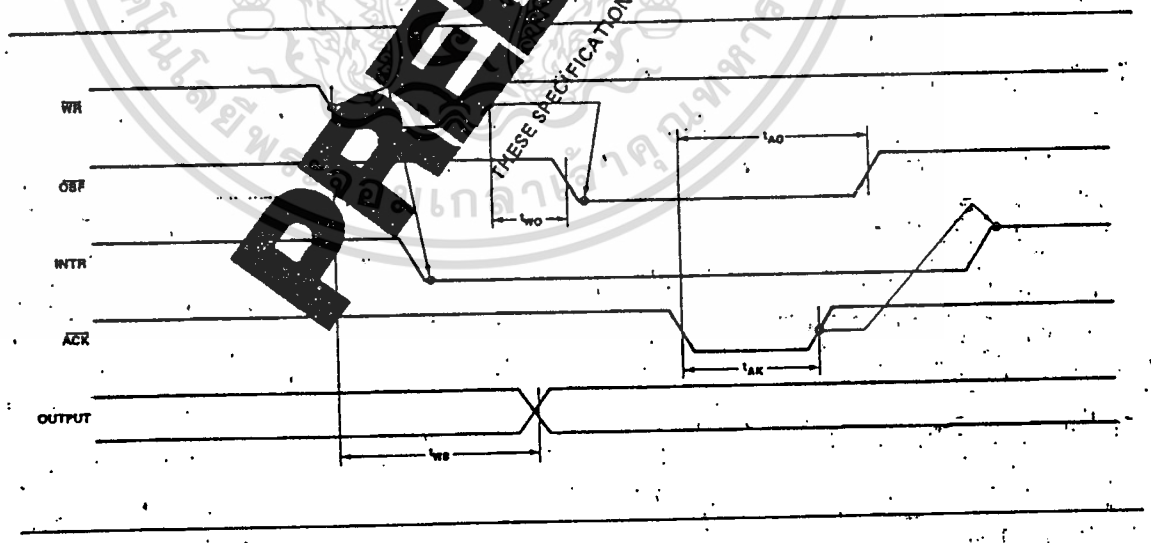


Mode 0 (Basic Input)

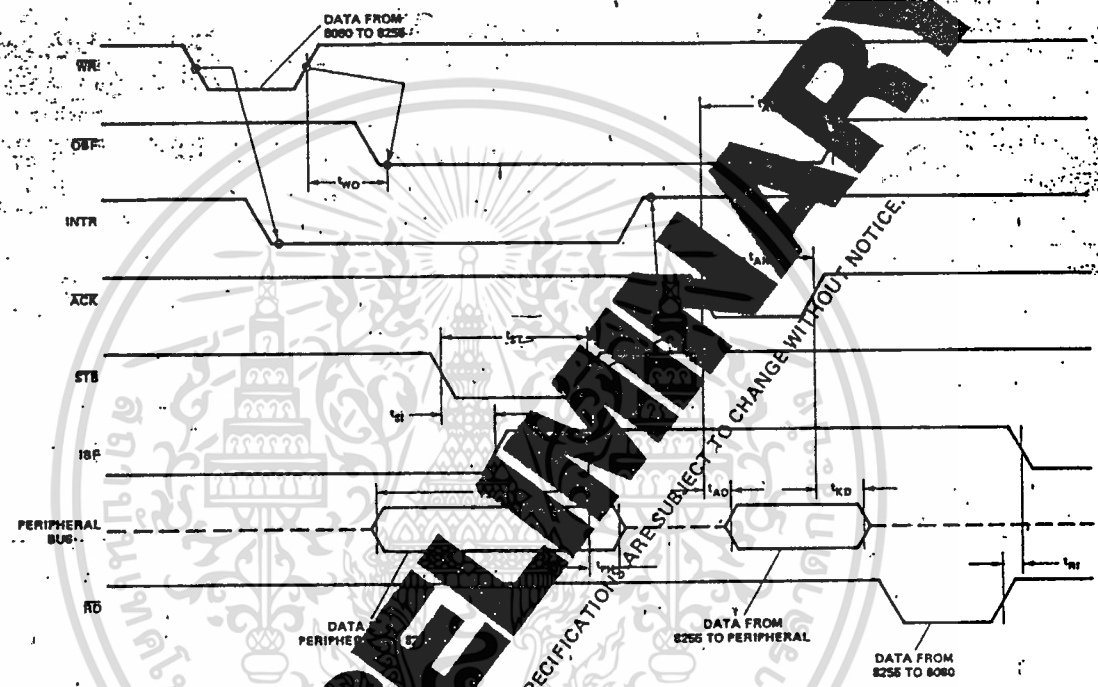
Mode 0 (Basic Output)



Mode 1 (Strobed Input)



Mode 1 (Strobed Output)



Mode 2 (Bi-directional)

intel

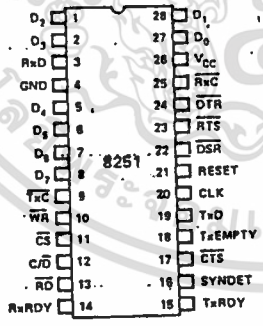
Silicon Gate MOS 8251

PROGRAMMABLE COMMUNICATION INTERFACE

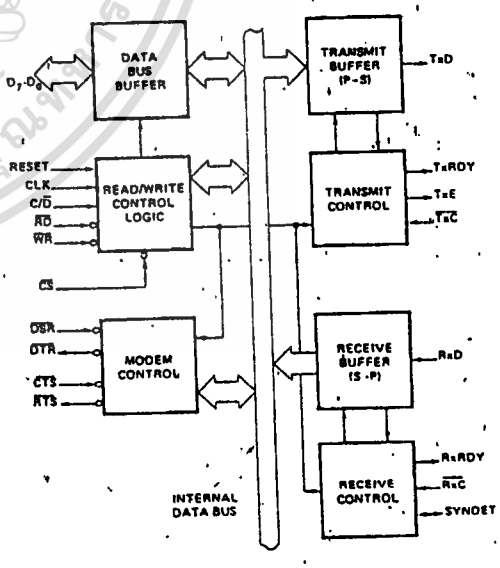
- **Synchronous and Asynchronous Operation**
  - **Synchronous:**
    - 5-8 Bit Characters
    - Internal or External Character Synchronization
    - Automatic Sync Insertion
  - **Asynchronous:**
    - 5-8 Bit Characters
    - Clock Rate — 1, 16 or 64 Times Baud Rate
    - Break Character Generation
    - 1, 1½, or 2 Stop Bits
    - False Start Bit Detection
- **Baud Rate — DC to 56k Baud (Sync Mode)**
  DC to 9.6k Baud (Async Mode)
- **Full Duplex, Double Buffered, Transmitter and Receiver**
- **Error Detection — Parity, Overrun, and Framing**
- **Fully Compatible with 8080 CPU**
- **28-Pin DIP Package**
- **All Inputs and Outputs Are TTL Compatible**
- **Single 5 Volt Supply**
- **Single TTL Clock**

The 8251 is a Universal Synchronous/Asynchronous Receiver/Transmitter (USART) Chip designed for data communications in microcomputer systems. The USART is used as a peripheral device and is programmed by the CPU to operate using virtually any serial data transmission technique presently in use (including IBM Bi-Sync). The USART accepts data characters from the CPU in parallel format and then converts them into a continuous serial data stream for transmission. Simultaneously it can receive serial data streams and convert them into parallel data characters for the CPU. The USART will signal the CPU whenever it can accept a new character for transmission or whenever it has received a character for the CPU. The CPU can read the complete status of the USART at any time. These include data transmission errors and control signals such as SYNDET, TxEMPT. The chip is constructed using N-channel silicon gate technology.

PIN CONFIGURATION



BLOCK DIAGRAM



Pin Name	Pin Function
D_7-D_0	Data Bus (8 bits)
C/$\overline{D}$	Control or Data is to be Written or Read.
$\overline{RD}$	Read Data Command
$\overline{WR}$	Write Data or Control Command
$\overline{CS}$	Chip Enable
CLK	Clock Pulse (TTL)
RESET	Reset
$\overline{TxC}$	Transmitter Clock
TxD	Transmitter Data
$\overline{RxC}$	Receiver Clock
RxD	Receiver Data
RxRDY	Receiver Ready (has character for 8080)
TxRDY	Transmitter Ready (ready for char. from 8080)

Pin Name	Pin Function
$\overline{DSR}$	Data Set Ready
$\overline{DTR}$	Data Terminal Ready
SYNDET	Sync Detect
$\overline{RTS}$	Request to Send Data
$\overline{CTS}$	Clear to Send Data
TxE	Transmitter Empty
Vcc	+5 Volt Supply
GND	Ground

8251 BASIC FUNCTIONAL DESCRIPTION

General

The 8251 is a Universal Synchronous/Asynchronous Receiver/Transmitter designed specifically for the 8080 Microcomputer System. Like other I/O devices in the 8080 Microcomputer System its functional configuration is programmed by the systems software for maximum flexibility. The 8251 can support virtually any serial data technique currently in use (including IBM "bi-sync").

In a communication environment an interface device must convert parallel format system data into serial format for transmission and convert incoming serial format data into parallel system data for reception. The interface device must also delete or insert bits or characters that are functionally unique to the communication technique. In essence, the interface should appear "transparent" to the CPU, a simple input or output of byte-oriented system data.

Data Bus Buffer

This 3-state, bi-directional, 8-bit buffer is used to interface the 8251 to the 8080 system Data Bus. Data is transmitted or received by the buffer upon execution of INput or OUTput instructions of the 8080 CPU. Control words, Command words and Status information are also transferred through the Data Bus Buffer.

Read/Write Control Logic

This functional block accepts inputs from the 8080 Control bus and generates control signals for overall device operation. It contains the Control Word Register and Command Word Register that store the various control formats for device functional definition.

RESET (Reset)

A "high" on this input forces the 8251 into an "Idle" mode. The device will remain at "Idle" until a new set of control words is written into the 8251 to program its functional definition.

CLK (Clock)

The CLK input is used to generate internal device timing and is normally connected to the Phase 2 (TTL) output of the 8224 Clock Generator. No external inputs or outputs are referenced to CLK but the frequency of CLK must be greater than 30 times the Receiver or Transmitter clock inputs for synchronous mode (4 times for asynchronous mode).

$\overline{WR}$ (Write)

A "low" on this input informs the 8251 that the CPU is outputting data or control words, in essence, the CPU is writing out to the 8251.

$\overline{RD}$ (Read)

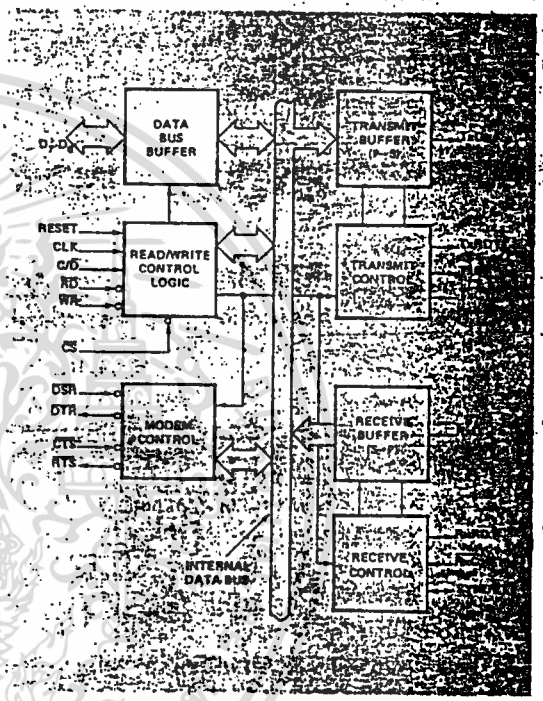
A "low" on this input informs the 8251 that the CPU is inputting data or status information, in essence, the CPU is reading from the 8251.

C/$\overline{D}$ (Control/Data)

This input, in conjunction with the $\overline{WR}$ and $\overline{RD}$ inputs informs the 8251 that the word on the Data Bus is either a data character, control word or status information.
1 = CONTROL 0 = DATA

$\overline{CS}$ (Chip Select)

A "low" on this input enables the 8251. No reading or writing will occur unless the device is selected.



C/$\overline{D}$	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	
0	0	1	0	8251 → DATA BUS
0	1	0	0	DATA BUS → 8251
1	0	1	0	STATUS → DATA BUS
1	1	0	0	DATA BUS → CONTROL
X	X	X	1	DATA BUS → 3-STATE

Modem Control

The 8251 has a set of control inputs and outputs that can be used to simplify the interface to almost any Modem. The modem control signals are general purpose in nature and can be used for functions other than Modem control, if necessary.

$\overline{DSR}$ (Data Set Ready)

The $\overline{DSR}$ input signal is general purpose in nature. Its condition can be tested by the CPU using a Status Read operation. The $\overline{DSR}$ input is normally used to test Modem conditions such as Data Set Ready.

$\overline{DTR}$ (Data Terminal Ready)

The $\overline{DTR}$ output signal is general purpose in nature. It can be set "low" by programming the appropriate bit in the Command Instruction word. The $\overline{DTR}$ output signal is normally used for Modem control such as Data Terminal Ready or Rate Select.

$\overline{RTS}$ (Request to Send)

The $\overline{RTS}$ output signal is general purpose in nature. It can be set "low" by programming the appropriate bit in the Command Instruction word. The $\overline{RTS}$ output signal is normally used for Modem control such as Request to Send.

$\overline{CTS}$ (Clear to Send)

A "low" on this input enables the 8251 to transmit data (serial) if the Tx EN bit in the Command byte is set to a "one."

Transmitter Buffer

The Transmitter Buffer accepts parallel data from the Data Bus Buffer, converts it to a serial bit stream, inserts the appropriate characters or bits (based on the communication technique) and outputs a composite serial stream of data on the TxD output pin.

Transmitter Control

The Transmitter Control manages all activities associated with the transmission of serial data. It accepts and issues signals both externally and internally to accomplish this function.

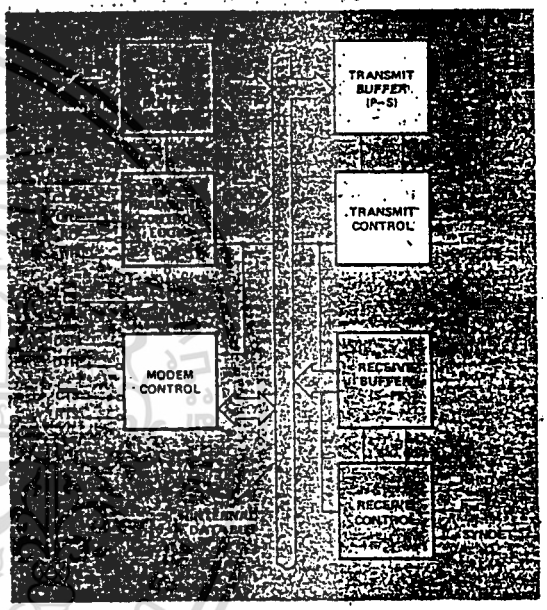
TxRDY (Transmitter Ready)

This output signals the CPU that the transmitter is ready to accept a data character. It can be used as an interrupt to the system or for the Polled operation the CPU can check TxRDY using a status read operation. TxRDY is automatically reset when a character is loaded from the CPU.

TxE (Transmitter Empty)

When the 8251 has no characters to transmit, the TxE output will go "high". It resets automatically upon receiving a character from the CPU. TxE can be used to indicate the end of a transmission mode, so that the CPU "knows" when to "turn the line around" in the half-duplexed operational mode.

In SYNChronous mode, a "high" on this output indicates that a character has not been loaded and the SYNC character or characters are about to be transmitted automatically as "fillers".



$\overline{TxC}$ (Transmitter Clock)

The Transmitter Clock controls the rate at which the character is to be transmitted. In the Synchronous transmission mode, the frequency of $\overline{TxC}$ is equal to the actual Baud Rate (1X). In Asynchronous transmission mode, the frequency of $\overline{TxC}$ is a multiple of the actual Baud Rate. A portion of the mode instruction selects the value of the multiplier; it can be 1x, 16x or 64x the Baud Rate.

For Example:

If Baud Rate equals 110 Baud,
$\overline{TxC}$ equals 110 Hz (1x)
$\overline{TxC}$ equals 1.76 kHz (16x)
$\overline{TxC}$ equals 7.04 kHz (64x).
If Baud Rate equals 9600 Baud,
$\overline{TxC}$ equals 614.4 kHz (64x).

The falling edge of $\overline{TxC}$ shifts the serial data out of the 8251.

Receiver Buffer

The Receiver accepts serial data, converts this serial input to parallel format, checks for bits or characters that are unique to the communication technique and sends an "assembled" character to the CPU. Serial data is input to the RxD pin.

Receiver Control

This functional block manages all receiver-related activities.

RxRDY (Receiver Ready)

This output indicates that the 8251 contains a character that is ready to be input to the CPU. RxRDY can be connected to the interrupt structure of the CPU or for Polled operation the CPU can check the condition of RxRDY using a status read operation. RxRDY is automatically reset when the character is read by the CPU.

$\overline{RxC}$ (Receiver Clock)

The Receiver Clock controls the rate at which the character is to be received. In Synchronous Mode, the frequency of $\overline{RxC}$ is equal to the actual Baud Rate (1x). In Asynchronous Mode, the frequency of $\overline{RxC}$ is a multiple of the actual Baud Rate. A portion of the mode instruction selects the value of the multiplier; it can be 1x, 16x or 64x the Baud Rate.

For Example: If Baud Rate equals 300 Baud,
$\overline{RxC}$ equals 300 Hz (1x)
$\overline{RxC}$ equals 4800 Hz (16x)
$\overline{RxC}$ equals 19.2 kHz (64x).
If Baud Rate equals 2400 Baud,
$\overline{RxC}$ equals 2400 Hz (1x)
$\overline{RxC}$ equals 38.4 kHz (16x)
$\overline{RxC}$ equals 153.6 kHz (64x).

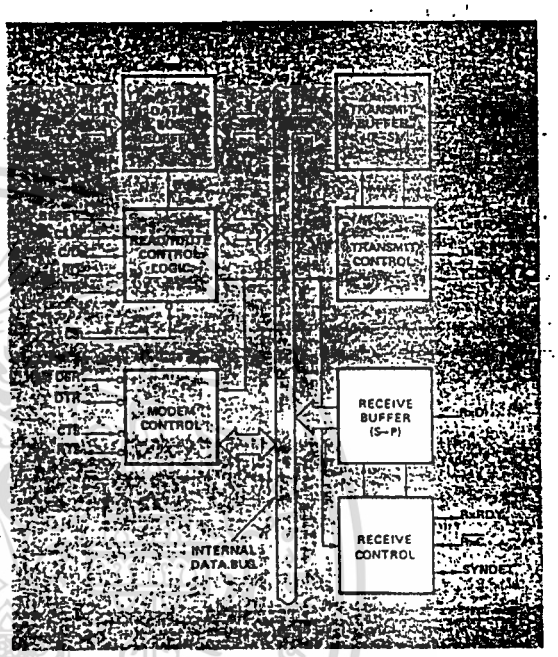
Data is sampled into the 8251 on the rising edge of $\overline{RxC}$.

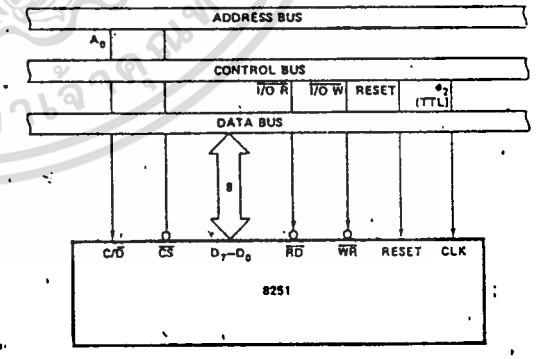
NOTE: In most communications systems, the 8251 will be handling both the transmission and reception operations of a single link. Consequently, the Receive and Transmit Baud Rates will be the same. Both $\overline{TxC}$ and $\overline{RxC}$ will require identical frequencies for this operation and can be tied together and connected to a single frequency source (Baud Rate Generator) to simplify the interface.

SYNDET (SYNC Detect)

This pin is used in SYNChronous Mode only. It is used as either input or output, programmable through the Control Word. It is reset to "low" upon RESET. When used as an output (internal Sync mode), the SYNDET pin will go "high" to indicate that the 8251 has located the SYNC character in the Receive mode. If the 8251 is programmed to use double Sync characters (bi-sync), then SYNDET will go "high" in the middle of the last bit of the second Sync character. SYNDET is automatically reset upon a Status Read operation.

When used as an input, (external SYNC detect mode), a positive going signal will cause the 8251 to start assembling data characters on the falling edge of the next $\overline{RxC}$. Once in SYNC, the "high" input signal can be removed. The duration of the high signal should be at least equal to the period of $\overline{RxC}$.





8251 Interface to 8080 Standard System Bus

DETAILED OPERATION DESCRIPTION

General

The complete functional definition of the 8251 is programmed by the systems software. A set of control words must be sent out by the CPU to initialize the 8251 to support the desired communications format. These control words will program the: BAUD RATE, CHARACTER LENGTH, NUMBER OF STOP BITS, SYNCHRONOUS or ASYNCHRONOUS OPERATION, EVEN/ODD PARITY etc. In the Synchronous Mode, options are also provided to select either internal or external character synchronization.

Once programmed, the 8251 is ready to perform its communication functions. The TxRDY output is raised "high" to signal the CPU that the 8251 is ready to receive a character. This output (TxRDY) is reset automatically when the CPU writes a character into the 8251. On the other hand, the 8251 receives serial data from the MODEM or I/O device, upon receiving an entire character the RxRDY output is raised "high" to signal the CPU that the 8251 has a complete character ready for the CPU to fetch. RxRDY is reset automatically upon the CPU read operation.

The 8251 cannot begin transmission until the TxEN(Transmitter Enable) bit is set in the Command Instruction and it has received a Clear To Send (CTS) input. The TxD output will be held in the marking state upon Reset.

Programming the 8251

Prior to starting data transmission or reception, the 8251 must be loaded with a set of control words generated by the CPU. These control signals define the complete functional definition of the 8251 and must immediately follow a Reset operation (internal or external).

The control words are split into two formats:

1. Mode Instruction
2. Command Instruction

Mode Instruction

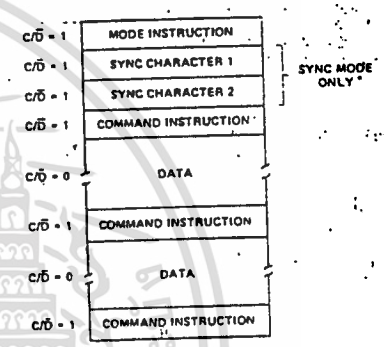
This format defines the general operational characteristics of the 8251. It must follow a Reset operation (internal or external). Once the Mode instruction has been written into the 8251 by the CPU, SYNC characters or Command instructions may be inserted.

Command Instruction

This format defines a status word that is used to control the actual operation of the 8251.

Both the Mode and Command instructions must conform to a specified sequence for proper device operation. The Mode Instruction must be inserted immediately following a Reset operation, prior to using the 8251 for data communication.

All control words written into the 8251 after the Mode Instruction will load the Command Instruction. Command Instructions can be written into the 8251 at any time in the data block during the operation of the 8251. To return to the Mode Instruction format a bit in the Command Instruction word can be set to initiate an internal Reset operation which automatically places the 8251 back into the Mode Instruction format. Command Instructions must follow the Mode Instructions or Sync characters.



Typical Data Block

Mode Instruction Definition

The 8251 can be used for either Asynchronous or Synchronous data communication. To understand how the Mode instruction defines the functional operation of the 8251 the designer can best view the device as two separate components sharing the same package. One Asynchronous the other Synchronous. The format definition can be changed "on the fly" but for explanation purposes the two formats will be isolated.

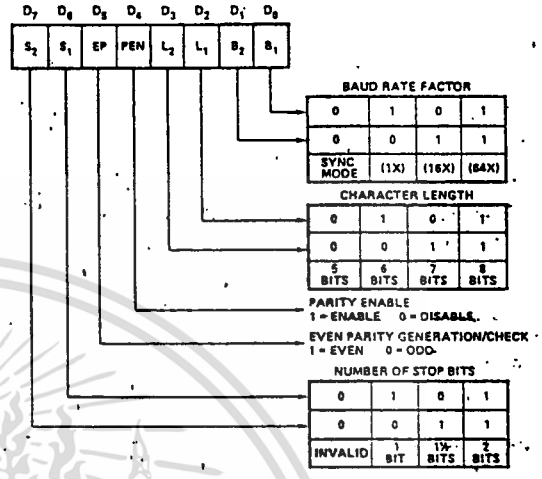
Asynchronous Mode (Transmission)

Whenever a data character is sent by the CPU the 8251 automatically adds a Start bit (low level) and the programmed number of Stop bits to each character. Also, an even or odd Parity bit is inserted prior to the Stop bit(s), as defined by the Mode Instruction. The character is then transmitted as a serial data stream on the TxD output. The serial data is shifted out on the falling edge of $\overline{TxC}$ at a rate equal to 1, 1/16, or 1/64 that of the $\overline{TxC}$, as defined by the Mode Instruction. BREAK characters can be continuously sent to the TxD if commanded to do so.

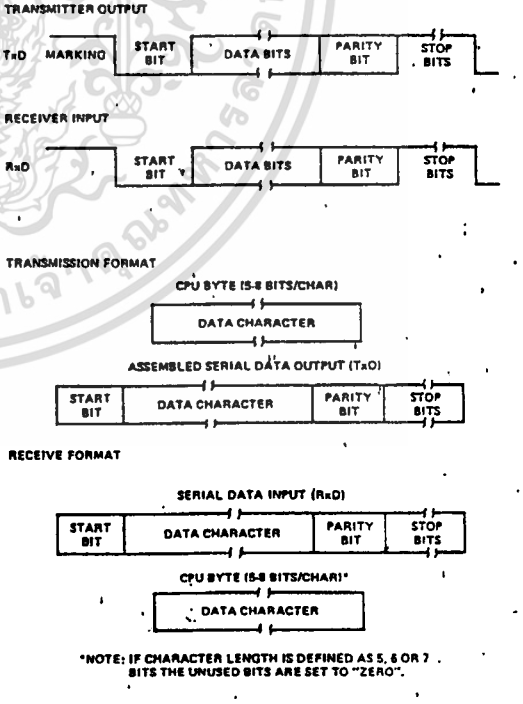
When no data characters have loaded into the 8251 the TxD output remains "high" (marking) unless a Break (continuously low) has been programmed.

Asynchronous Mode (Receive)

The RxD line is normally high. A falling edge on this line triggers the beginning of a START bit. The validity of this START bit is checked by again strobing this bit at its nominal center. If a low is detected again, it is a valid START bit, and the bit counter will start counting. The bit counter locates the center of the data bits; the parity bit (if it exists) and the stop bits. If parity error occurs, the parity error flag is set. Data and parity bits are sampled on the RxD pin with the rising edge of $\overline{RxC}$. If a low level is detected as the STOP bit, the Framing Error flag will be set. The STOP bit signals the end of a character. This character is then loaded into the parallel I/O buffer of the 8251. The RxRDY pin is raised to signal the CPU that a character is ready to be fetched. If a previous character has not been fetched by the CPU, the present character replaces it in the I/O buffer, and the OVERRUN flag is raised (thus the previous character is lost). All of the error flags can be reset by a command instruction. The occurrence of any of these errors will not stop the operation of the 8251.



Mode Instruction Format, Asynchronous Mode



Asynchronous Mode

Synchronous Mode (Transmission)

The TxD output is continuously high until the CPU sends its first character to the 8251 which usually is a SYNC character. When the $\overline{CTS}$ line goes low, the first character is serially transmitted out. All characters are shifted out on the falling edge of $\overline{TxC}$. Data is shifted out at the same rate as the $\overline{TxC}$.

Once transmission has started, the data stream at TxD output must continue at the $\overline{TxC}$ rate. If the CPU does not provide the 8251 with a character before the 8251 becomes empty, the SYNC characters (or character if in single SYNC word mode) will be automatically inserted in the TxD data stream. In this case, the TxEMPTY pin is raised high to signal that the 8251 is empty and SYNC characters are being sent out. The TxEMPTY pin is internally reset by the next character being written into the 8251.

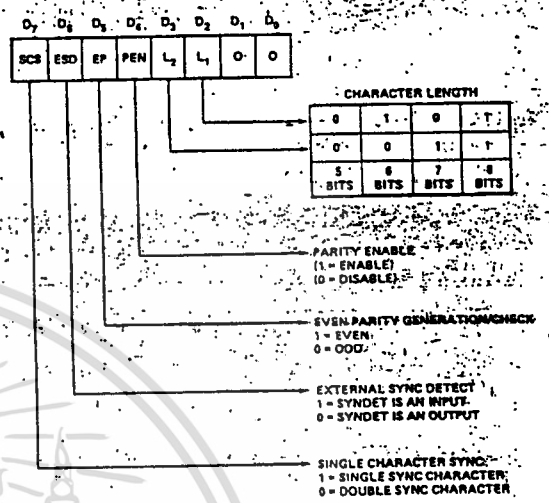
Synchronous Mode (Receive)

In this mode, character synchronization can be internally or externally achieved. If the internal SYNC mode has been programmed, the receiver starts in a HUNT mode. Data on the RxD pin is then sampled in on the rising edge of $\overline{RxC}$. The content of the Rx buffer is continuously compared with the first SYNC character until a match occurs. If the 8251 has been programmed for two SYNC characters, the subsequent received character is also compared; when both SYNC characters have been detected, the USART ends the HUNT mode and is in character synchronization. The SYNDET pin is then set high, and is reset automatically by a STATUS READ.

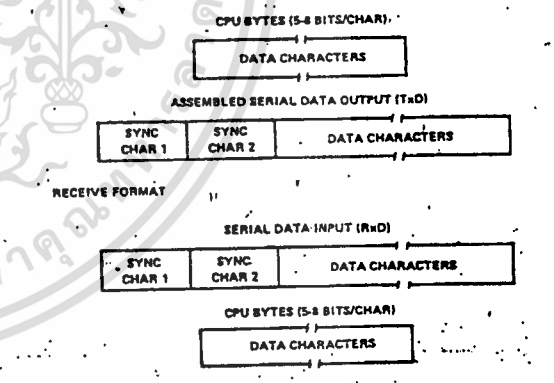
In the external SYNC mode, synchronization is achieved by applying a high level on the SYNDET pin. The high level can be removed after one $\overline{RxC}$ cycle.

Parity error and overrun error are both checked in the same way as in the Asynchronous Rx mode.

The CPU can command the receiver to enter the HUNT mode if synchronization is lost.



Mode Instruction Format, Synchronous Mode



Synchronous Mode, Transmission Format

COMMAND INSTRUCTION DEFINITION

Once the functional definition of the 8251 has been programmed by the Mode Instruction and the Sync Characters are loaded (if in Sync Mode) then the device is ready to be used for data communication. The Command Instruction controls the actual operation of the selected format. Functions such as: Enable Transmit/Receive, Error Reset and Modem Controls are provided by the Command Instruction.

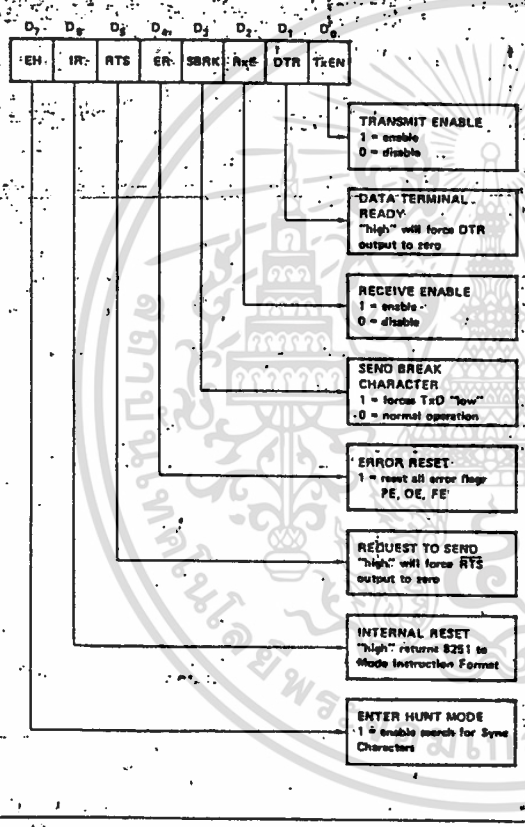
Once the Mode Instruction has been written into the 8251 and Sync characters inserted, if necessary, then all further "control writes" ($C/\overline{D}$ = 1) will load the Command Instruction. A Reset operation (internal or external) will return the 8251 to the Mode Instruction Format.

STATUS READ DEFINITION

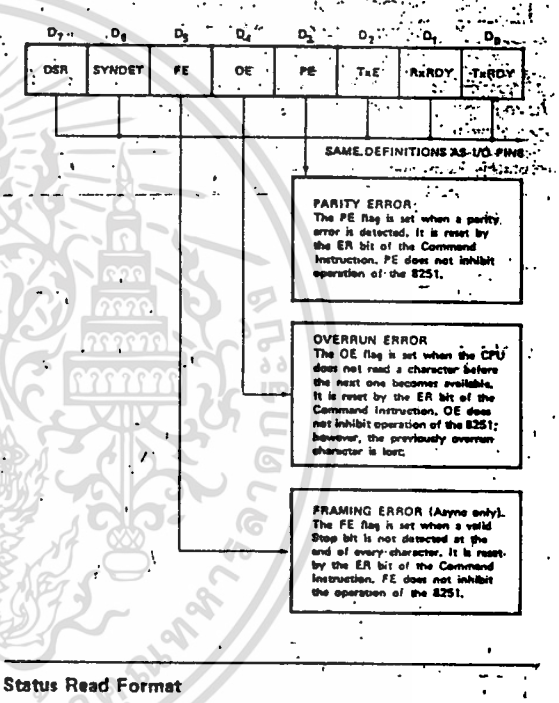
In data communication systems it is often necessary to examine the "status" of the active device to ascertain if errors have occurred or other conditions that require the processor's attention. The 8251 has facilities that allow the programmer to "read" the status of the device at any time during the functional operation.

A normal "read" command is issued by the CPU with the C/D input at one to accomplish this function.

Some of the bits in the Status Read Format have identical meanings to external output pins so that the 8251 can be used in a completely Polled environment or in an interrupt driven environment.

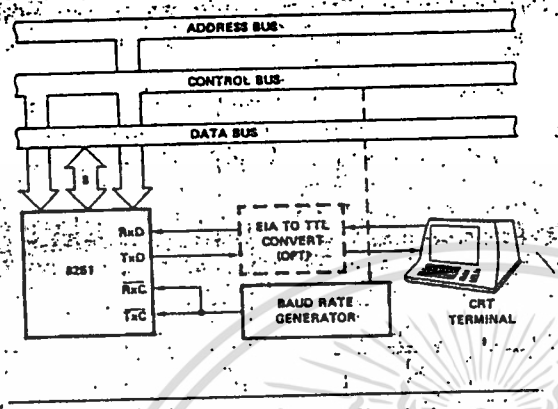


Command Instruction Format

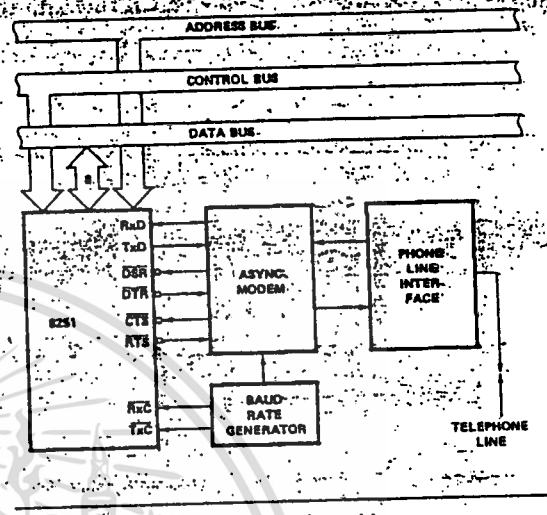


Status Read Format

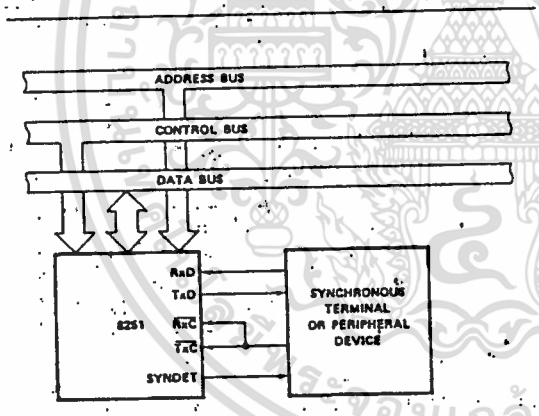
APPLICATIONS OF THE 8251



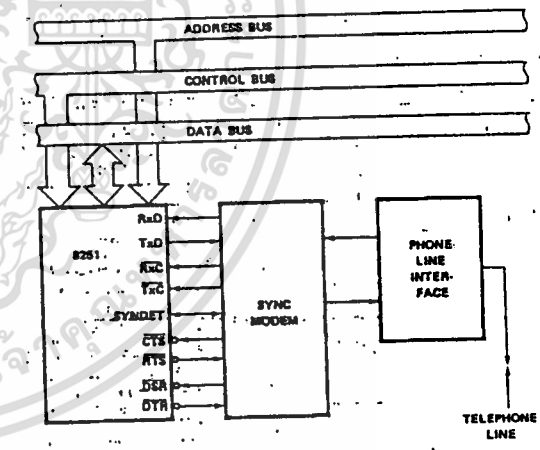
Asynchronous Serial Interface to CRT Terminal, DC-9600 Baud



Asynchronous Interface to Telephone Lines



Synchronous Interface to Terminal or Peripheral Device



Synchronous Interface to Telephone Lines

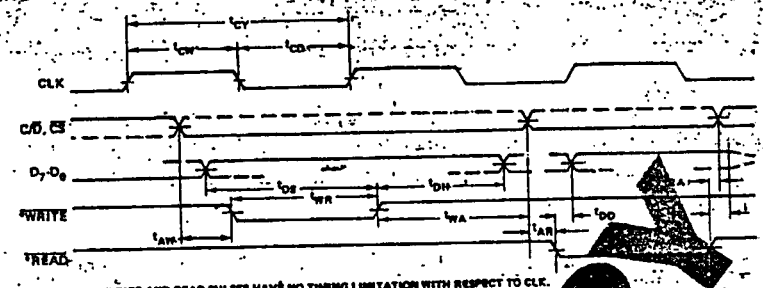
D.C. Characteristics: T_A = 0°C to 70°C, V_{CC} = +5V ±5%, V_{SS} = 0V

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Conditions
V_{IL}	Input Low Voltage			.8	V	
V_{IH}	Input High Voltage	2.0			V	
V_{OL}	Output Low Voltage			.4	V	I_{OL}=2.0mA (DB_{0-7}), 1.6mA (Others)
V_{OH}	Output High Voltage	2.4			V	I_{OH}=-150μA (DB_{0-7}), -100μA (Others)
I_{CC}	Power Supply Current		80		mA	
I_{LI}	Input Load Current		10		μA	V_{IN}=0V to 5.25V
I_{LOL}	Output Leakage Current (DB Low)		-100		μA	V_{OUT}=0.4V
I_{LOH}	Output Leakage Current (DB High)		+10		μA	V_{OUT}=V_{CC}

A.C. Characteristics: T_A = 0°C to 70°C, V_{CC} = +5V ±5%, V_{SS} = 0V

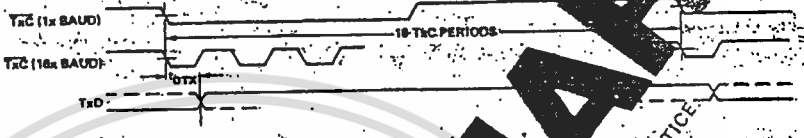
Symbol	Parameter	Min.		Max.	Unit	Test Conditions
t_{CY}	Clock Period		[illegible]		ns	
t_{CW}	Clock Pulse Width		[illegible]		ns	
t_{CD}	Clock Pulse Delay		[illegible]		ns	
$t_{R,F}$	Clock Rise and Fall Time		[illegible]		ns	
t_{WR}	Write Pulse Width		400		ns	
t_{DS}	Data Set-Up Time for WRITE		300		ns	
t_{DH}	Data Hold Time for WRITE		20		ns	
t_{DD}	Data Delay from READ			350	ns	C_L = 150pF
t_{FD}	READ to Data Floating			160	ns	C_L = 150pF
t_{AW}	Address Stable before WRITE		0		ns	
t_{WA}	Address Hold Time for WRITE			40	ns	
t_{AR}	Address Stable before READ			0	ns	
t_{RA}	Address Hold Time for READ			0	ns	
t_{DTx}	TxD Delay from Falling Edge of TxC			300	ns	
t_{SRx}	Rx Data Set-Up Time to Sampling Pulse			500	ns	
t_{HRx}	Rx Data Hold Time to Sampling Pulse			6	CLK Period	
f_{Tx}	Transmitter Input Clock Frequency 1X Baud Rate 16X and 64X Baud Rate			 56 615	 KHz KHz	
f_{Rx}	Receiver Input Clock Frequency 1X Baud Rate 16X and 64X Baud Rate			 56 615	 KHz KHz	
t_{Tx}	TxRDY Delay from Center of Data Bit			6	CLK Period	C_L=50pF
t_{Rx}	RxRDY Delay from Center of Data Bit			6	CLK Period	
t_{IS}	Internal SYNDET Delay from Center of Data Bit			6	CLK Period	
t_{ES}	External SYNDET Set-Up Time before Rising Edge of RxC			6	CLK Period	

READ AND WRITE TIMING



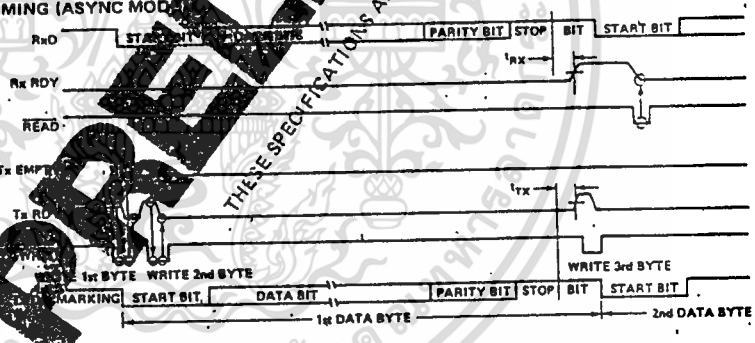
*WRITE AND READ PULSES HAVE NO TIMING LIMITATION WITH RESPECT TO CLK.

TRANSMITTER CLOCK AND DATA

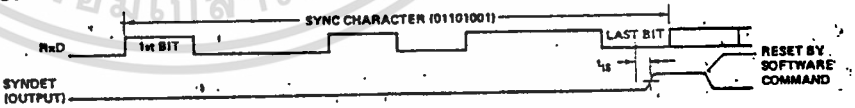


RECEIVER CLOCK AND DATA

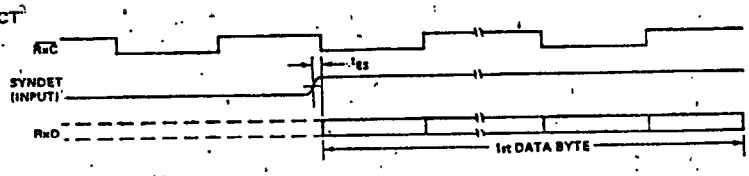
Tx RDY AND Rx RDY TIMING (ASYNC MODE)



INTERNAL SYNC DETECT



EXTERNAL SYNC DETECT



Z80-CPU™
Z80A-CPU™

Zilog

Product Specification

The Zilog Z80 product line is a complete set of microcomputer components, development systems and support software. The Z80 microcomputer component set includes all of the circuits necessary to build high-performance microcomputer systems with virtually no other logic and a minimum number of low cost standard memory elements.

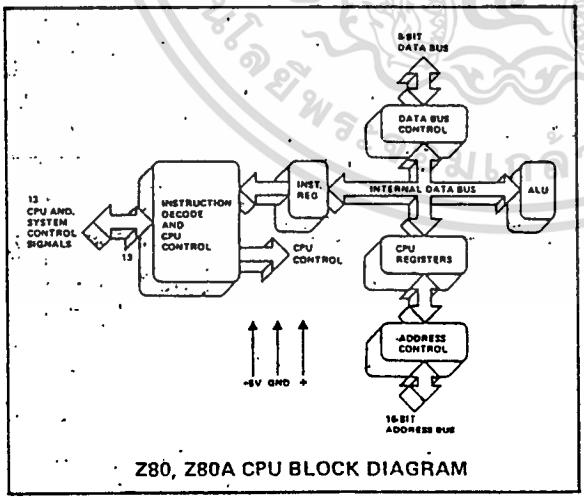
The Z80 and Z80A CPU's are third generation single chip microprocessors with unrivaled computational power. This increased computational power results in higher system through-put and more efficient memory utilization when compared to second generation microprocessors. In addition, the Z80 and Z80A CPU's are very easy to implement into a system because of their single voltage requirement plus all output signals are fully decoded and timed to control standard memory or peripheral circuits. The circuit is implemented using an N-channel, ion implanted, silicon gate MOS process.

Figure 1 is a block diagram of the CPU, Figure 2 details the internal register configuration which contains 208 bits of Read/Write memory that are accessible to the programmer. The registers include two sets of six general purpose registers that may be used individually as 8-bit registers or as 16-bit register pairs. There are also two sets of accumulator and flag registers. The programmer has access to either set of main or alternate registers through a group of exchange instructions. This alternate set allows foreground/background mode of operation or may be reserved for very fast Interrupt response. Each CPU also contains a 16-bit stack pointer which permits simple implementation of multiple level interrupts, unlimited subroutine nesting and simplification of many types of data handling.

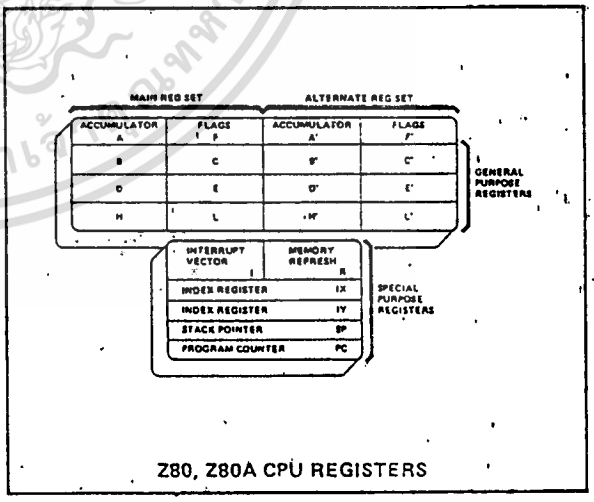
The two 16-bit index registers allow tabular data manipulation and easy implementation of relocatable code. The Refresh register provides for automatic, totally transparent refresh of external dynamic memories. The I register is used in a powerful interrupt response mode to form the upper 8 bits of a pointer to a interrupt service address table; while the interrupting device supplies the lower 8 bits of the pointer. An indirect call is then made to this service address.

FEATURES

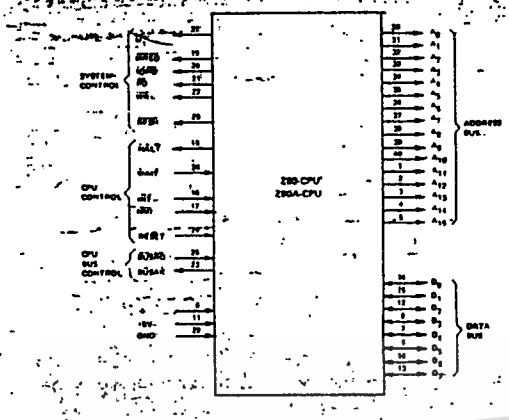
- Single chip, N-channel Silicon Gate CPU.
- 158 instructions—includes all 78 of the 8080A instructions with total software compatibility. New instructions include 4-, 8- and 16-bit operations with more useful addressing modes such as indexed, bit and relative.
- 17 internal registers.
- Three modes of fast interrupt response plus a non-maskable interrupt.
- Directly interfaces standard speed static or dynamic memories with virtually no external logic.
- 1.0 μs instruction execution speed.
- Single 5 VDC supply and single-phase 5 volt Clock.
- Out-performs any other single chip microcomputer in 4-, 8-, or 16-bit applications.
- All pins TTL Compatible
- Built-in dynamic RAM refresh circuitry.



Z80, Z80A CPU BLOCK DIAGRAM



Z80, Z80A CPU REGISTERS



Z80, Z80A CPU PIN CONFIGURATION

A_0-A_{15} (Address Bus) Tri-state output, active high. A_0-A_{15} constitute a 16-bit address bus. The address bus provides the address for memory (up to 64K bytes) data exchanges and for I/O device data exchanges.

D_0-D_7 (Data Bus) Tri-state input/output, active high. D_0-D_7 constitute an 8-bit bidirectional data bus. The data bus is used for data exchanges with memory and I/O devices.

$\overline{M_1}$ (Machine Cycle one) Output, active low. $\overline{M_1}$ indicates that the current machine cycle is the OP code fetch cycle of an instruction execution.

$\overline{MREQ}$ (Memory Request) Tri-state output, active low. The memory request signal indicates that the address bus holds a valid address for a memory read or memory write operation.

$\overline{IORQ}$ (Input/ Output Request) Tri-state output, active low. The $\overline{IORQ}$ signal indicates that the lower half of the address bus holds a valid I/O address for a I/O read or write operation. An $\overline{IORQ}$ signal is also generated when an interrupt is being acknowledged to indicate that an interrupt response vector can be placed on the data bus.

$\overline{RD}$ (Memory Read) Tri-state output, active low. $\overline{RD}$ indicates that the CPU wants to read data from memory or an I/O device. The addressed I/O device or memory should use this signal to gate data onto the CPU data bus.

$\overline{WR}$ (Memory Write) Tri-state output, active low. $\overline{WR}$ indicates that the CPU data bus holds valid data to be stored in the addressed memory or I/O device.

$\overline{RFSH}$ (Refresh) Output, active low. $\overline{RFSH}$ indicates that the lower 7 bits of the address bus contain a refresh address for dynamic memories and the current $\overline{MREQ}$ signal should be used to do a refresh read to all dynamic memories.

$\overline{HALT}$ (Halt state) Output, active low. $\overline{HALT}$ indicates that the CPU has executed a HALT software instruction and is awaiting either a non-maskable or a maskable interrupt (with the mask enabled) before operation can resume. While halted, the CPU executes NOP's to maintain memory refresh activity.

$\overline{WAIT}$ (Wait) Input, active low. $\overline{WAIT}$ indicates to the Z-80 CPU that the addressed memory or I/O devices are not ready for a data transfer. The CPU continues to enter wait states for as long as this signal is active.

$\overline{INT}$ (Interrupt Request) Input, active low. The Interrupt Request signal is generated by I/O devices. A request will be honored at the end of the current instruction if the internal software controlled interrupt enable flip-flop (IFF) is enabled.

$\overline{NMI}$ (Non Maskable Interrupt) Input, active low. The non-maskable interrupt request line has a higher priority than $\overline{INT}$ and is always recognized at the end of the current instruction, independent of the status of the interrupt enable flip-flop. $\overline{NMI}$ automatically forces the Z-80 CPU to restart to location 0066_H.

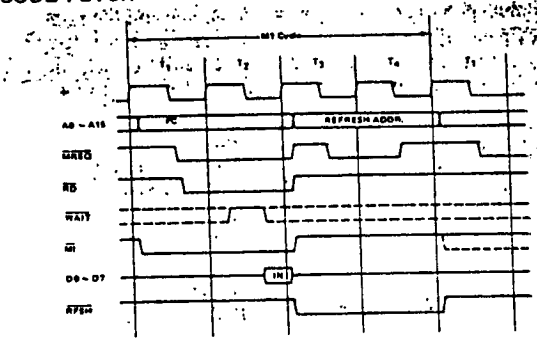
$\overline{RESET}$ Input, active low. $\overline{RESET}$ initializes the CPU as follows: reset interrupt enable flip-flop, clear PC and registers I and R and set interrupt to 8080A mode. During reset time, the address and data bus go to a high impedance state and all control output signals go to the inactive state.

$\overline{BUSRQ}$ (Bus Request) Input, active low. The bus request signal has a higher priority than $\overline{NMI}$ and is always recognized at the end of the current machine cycle and is used to request the CPU address bus, data bus and tri-state output control signals to go to a high impedance state so that other devices can control these busses.

$\overline{BUSAK}$ (Bus Acknowledge) Output, active low. Bus acknowledge is used to indicate to the requesting device that the CPU address bus, data bus and tri-state control bus signals have been set to their high impedance state and the external device can now control these signals.

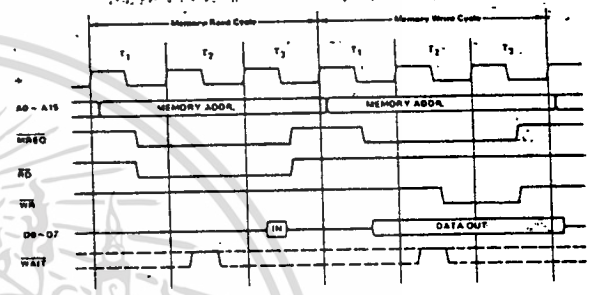
INSTRUCTION OP CODE FETCH

The program counter content (PC) is placed on the address bus immediately at the start of the cycle. One half clock time later $\overline{MREQ}$ goes active. The falling edge of $\overline{MREQ}$ can be used directly as a chip enable to dynamic memories. $\overline{RD}$ when active indicates that the memory data should be enabled onto the CPU data bus. The CPU samples data with the rising edge of the clock state T_3. Clock states T_3 and T_4 of a fetch cycle are used to refresh dynamic memories while the CPU is internally decoding and executing the instruction. The refresh control signal $\overline{RFSH}$ indicates that a refresh read of all dynamic memories should be accomplished.



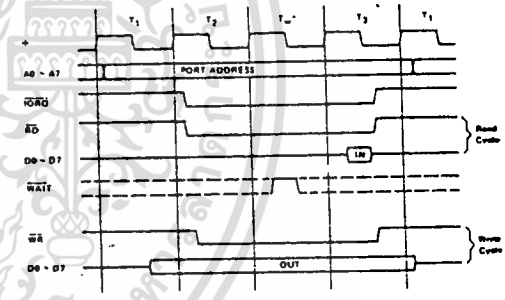
MEMORY READ OR WRITE CYCLES

Illustrated here is the timing of memory read or write cycles other than an OP code fetch (M_1 cycle). The $\overline{MREQ}$ and $\overline{RD}$ signals are used exactly as in the fetch cycle. In the case of a memory write cycle, the $\overline{MREQ}$ also becomes active when the address bus is stable so that it can be used directly as a chip enable for dynamic memories. The $\overline{WR}$ line is active when data on the data bus is stable so that it can be used directly as a R/W pulse to virtually any type of semiconductor memory.



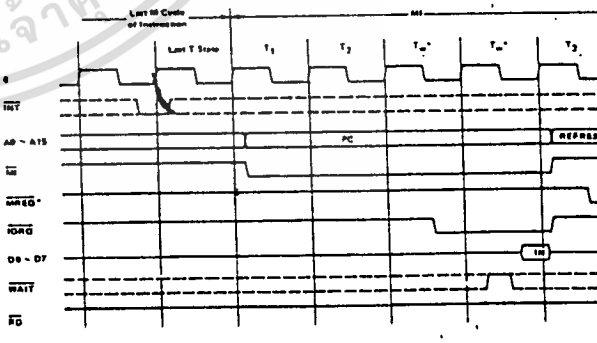
INPUT OR OUTPUT CYCLES

Illustrated here is the timing for an I/O read or I/O write operation. Notice that during I/O operations a single wait state is automatically inserted (Tw*). The reason for this is that during I/O operations this extra state allows sufficient time for an I/O port to decode its address and activate the $\overline{WAIT}$ line if a wait is required.



INTERRUPT REQUEST/ACKNOWLEDGE CYCLE

The interrupt signal is sampled by the CPU with the rising edge of the last clock at the end of any instruction. When an interrupt is accepted, a special M_1 cycle is generated. During this M_1 cycle, the $\overline{IORQ}$ signal becomes active (instead of $\overline{MREQ}$) to indicate that the interrupting device can place an 8-bit vector on the data bus. Two wait states (Tw*) are automatically added to this cycle so that a ripple priority interrupt scheme, such as the one used in the Z80 peripheral controllers, can be easily implemented.



Z80, Z80A Instruction Set

The following is a summary of the Z80, Z80A instruction set showing the assembly language mnemonic and the symbolic operation performed by the instruction. A more detailed listing appears in the Z80-CPU technical manual, and assembly language programming manual. The instructions are divided into the following categories:

- 8-bit loads
- 16-bit loads
- Exchanges
- Memory Block Moves
- Memory Block Searches
- 8-bit arithmetic and logic
- 16-bit arithmetic
- General purpose Accumulator & Flag Operations
- Miscellaneous Group
- Rotates and Shifts
- Bit Set, Reset and Test
- Input and Output
- Jumps
- Calls
- Restarts
- Returns

In the table the following terminology is used.

- b ≡ a bit number in any 8-bit register or memory location
- cc ≡ flag condition code
  - NZ ≡ non zero
  - Z ≡ zero
  - NC ≡ non carry
  - C ≡ carry
  - PO ≡ Parity odd or no over flow
  - PE ≡ Parity even or over flow
  - P ≡ Positive
  - M ≡ Negative (minus)
- d ≡ any 8-bit destination register or memory location
- dd ≡ any 16-bit destination register or memory location
- e ≡ 8-bit signed 2's complement displacement used in relative jumps and indexed addressing
- L ≡ 8 special call locations in page zero. In decimal notation these are 0, 8, 16, 24, 32, 40, 48 and 56
- n ≡ any 8-bit binary number
- nn ≡ any 16-bit binary number
- r ≡ any 8-bit general purpose register (A, B, C, D, E, H, or L)
- s ≡ any 8-bit source register or memory location
- s_b ≡ a bit in a specific 8-bit register or memory location
- ss ≡ any 16-bit source register or memory location
- subscript "L" ≡ the low order 8 bits of a 16-bit register
- subscript "H" ≡ the high order 8 bits of a 16-bit register
- ( ) ≡ the contents within the ( ) are to be used as a pointer to a memory location or I/O port number

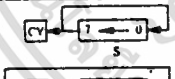
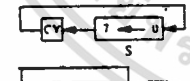
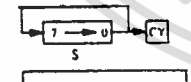
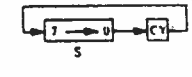
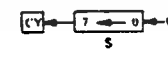
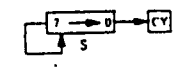
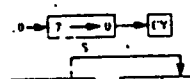
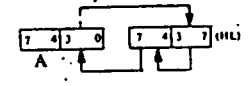
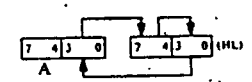
8-bit registers are A, B, C, D, E, H, L, I and R.
16-bit register pairs are AF, BC, DE and HL
16-bit registers are SP, PC, IX and IY

Addressing Modes implemented include combinations of the following:

- Immediate
- Immediate extended
- Modified Page Zero
- Relative
- Extended
- Indexed
- Register
- Implied
- Register Indirect
- Bit

	Mnemonic	Symbolic Operation	Comments
8-BIT LOADS	LD r, s	r ← s	s ≡ r, n, (HL), (IX+e), (IY+e)
	LD d, r	d ← r	d ≡ (HL), r (IX+e), (IY+e)
	LD d, n	d ← n	d ≡ (HL), (IX+e), (IY+e)
	LD A, s	A ← s	s ≡ (BC), (DE), (nn), I, R
	LD d, A	d ← A	d ≡ (BC), (DE), (nn), I, R
16-BIT LOADS	LD dd, nn	dd ← nn	dd ≡ BC, DE, HL, SP, IX, IY
	LD dd, (nn)	dd ← (nn)	dd ≡ BC, DE, HL, SP, IX, IY
	LD (nn), ss	(nn) ← ss	ss ≡ BC, DE, HL, SP, IX, IY
	LD SP, ss	SP ← ss	ss = HL, IX, IY
	PUSH ss	$(SP-1) \leftarrow ss_H$; $(SP-2) \leftarrow ss_L$	ss = BC, DE, HL, AF, IX, IY
	POP dd	$dd_L \leftarrow (SP)$; $dd_H \leftarrow (SP+1)$	dd = BC, DE, HL, AF, IX, IY
EXCHANGES	EX DE, HL	DE ↔ HL	
	EX AF, AF'	AF ↔ AF'	
	EXX	$\begin{pmatrix}BC\\DE\\HL\end{pmatrix} \leftrightarrow \begin{pmatrix}BC'\\DE'\\HL'\end{pmatrix}$	
	EX (SP), ss	$(SP) \leftrightarrow ss_L$, $(SP+1) \leftrightarrow ss_H$	ss ≡ HL, IX, IY

	Mnemonic	Symbolic Operation	Comments
MEMORY BLOCK MOVES	LDI	(DE) ← (HL), DE ← DE+1 HL ← HL+1, BC ← BC-1	
	LDIR	(DE) ← (HL), DE ← DE+1 HL ← HL+1, BC ← BC-1 Repeat until BC = 0	
	LDD	(DE) ← (HL), DE ← DE-1 HL ← HL-1, BC ← BC-1	
	LDDR	(DE) ← (HL), DE ← DE-1 HL ← HL-1, BC ← BC-1 Repeat until BC = 0	
MEMORY BLOCK SEARCHES	CPI	A-(HL), HL ← HL+1 BC ← BC-1	
	CPIR	A-(HL), HL ← HL+1 BC ← BC-1, Repeat until BC = 0 or A = (HL)	A-(HL) sets the flags only. A is not affected
	CPD	A-(HL), HL ← HL-1 BC ← BC-1	
	CPDR	A-(HL), HL ← HL-1 BC ← BC-1, Repeat until BC= 0 or A = (HL)	
8-BIT ALU	ADD s	A ← A + s	
	ADC s	A ← A + s + CY	CY is the carry flag
	SUB s	A ← A - s	
	SBC s	A ← A - s - CY	s ≡ r, n, (HL) (IX+e), (IY+e)
	AND s	A ← A ∧ s	
	OR s	A ← A ∨ s	
	XOR s	A ← A ⊕ s	

Mnemonic	Symbolic Operation	Comments
8-BIT ARITHMETIC		
CP s	A − s	s = r, n, (HL), (IX+e), (IY+e)
INC d	d ← d + 1	d = r, (HL), (IX+e), (IY+e)
DEC d	d ← d − 1	
16-BIT ARITHMETIC		
ADD HL, ss	HL ← HL + ss	ss ≡ BC, DE, HL, SP
ADC HL, ss	HL ← HL + ss + CY	
SBC HL, ss	HL ← HL − ss − CY	
ADD IX, ss	IX ← IX + ss	ss ≡ BC, DE, IX, SP
ADD IY, ss	IY ← IY + ss	ss ≡ BC, DE, IY, SP
INC dd	dd ← dd + 1	dd ≡ BC, DE, HL, SP, IX, IY
DEC dd	dd ← dd − 1	dd ≡ BC, DE, HL, SP, IX, IY
GP ACC. & F-LAG		
DAA	Converts A contents into packed BCD following add or subtract.	Operands must be in packed BCD format
CPL	$A \leftarrow \overline{A}$	
NEG	A ← 00 − A	
CCF	$CY \leftarrow \overline{CY}$	
SCF	CY ← 1	
MISCELLANEOUS		
NOP	No operation	
HALT	Halt CPU	
DI	Disable Interrupts	
EI	Enable Interrupts	
IM 0	Set interrupt mode 0	8080A mode
IM 1	Set interrupt mode 1	Call to 0038_H
IM 2	Set interrupt mode 2	Indirect Call
ROTATES AND SHIFTS		
RLC s		s ≡ r, (HL), (IX+e), (IY+e)
RL s		
RRC s		
RR s		
SLA s		
SRA s		
SRL s		
RLD		
RRD		

Mnemonic	Symbolic Operation	Comments
BIT S, R, & T		
BIT b, s	$Z \leftarrow \overline{s_b}$	Z is zero flag
SET b, s	$s_b \leftarrow 1$	s ≡ r, (HL), (IX+e), (IY+e)
RES b, s	$s_b \leftarrow 0$	
INPUT AND OUTPUT		
IN A, (n)	A ← (n)	
IN r, (C)	r ← (C)	Set flags
INI	(HL) ← (C), HL ← HL + 1, B ← B − 1	
INIR	(HL) ← (C), HL ← HL + 1, B ← B − 1, Repeat until B = 0	
IND	(HL) ← (C), HL ← HL − 1, B ← B − 1	
INDR	(HL) ← (C), HL ← HL − 1, B ← B − 1, Repeat until B = 0	
OUT(n), A	(n) ← A	
OUT(C), r	(C) ← r	
OUTI	(C) ← (HL), HL ← HL + 1, B ← B − 1	
OTIR	(C) ← (HL), HL ← HL + 1, B ← B − 1, Repeat until B = 0	
OUTD	(C) ← (HL), HL ← HL − 1, B ← B − 1	
OTDR	(C) ← (HL), HL ← HL − 1, B ← B − 1, Repeat until B = 0	
JUMPS		
JP nn	PC ← nn	cc ≡ NZ, Z, NC, C, PO, PE, P, M
JP cc, nn	If condition cc is true PC ← nn, else continue	
JR e	PC ← PC + e	kk ≡ NZ, Z, NC, C
JR kk, e	If condition kk is true PC ← PC + e, else continue	ss = HL, IX, IY
JP (ss)	PC ← ss	
DJNZ e	B ← B − 1, if B = 0 continue, else PC ← PC + e	
CALLS		
CALL nn	$(SP-1) \leftarrow PC_H$, $(SP-2) \leftarrow PC_L$, PC ← nn	cc ≡ NZ, Z, NC, C, PO, PE, P, M
CALL cc, nn	If condition cc is false continue, else same as CALL nn	
RESTARTS		
RST L	$(SP-1) \leftarrow PC_H$, $(SP-2) \leftarrow PC_L$, $PC_H \leftarrow 0$, $PC_L \leftarrow L$	
RETURNS		
RET	$PC_L \leftarrow (SP)$, $PC_H \leftarrow (SP+1)$	cc ≡ NZ, Z, NC, C, PO, PE, P, M
RET cc	If condition cc is false continue, else same as RET	
RETI	Return from interrupt, same as RET	
RETN	Return from non-maskable interrupt	

T_A = 0°C to 70°C, Vcc = +5V ± 5%, Unless Otherwise Noted.

Signal	Symbol	Parameter	Min	Max	Unit	Test Condition
Φ	t_c	Clock Period	250	1000	nsec	
	$t_{w(\Phi H)}$	Clock Pulse Width, Clock High	180	[E]	nsec	
	$t_{w(\Phi L)}$	Clock Pulse Width, Clock Low	180	2000	nsec	
	$t_{r,f}$	Clock Rise and Fall Time		30	nsec	
A_{0-15}	$t_{D(AD)}$	Address Output Delay		145	nsec	C_L = 50pF
	$t_{F(AD)}$	Delay to Float		110	nsec	
	t_{acm}	Address Stable Prior to MREQ (Memory Cycle)	[1]		nsec	
	t_{aci}	Address Stable Prior to IORQ, RD or WR (I/O Cycle)	[2]		nsec	
	t_{ca}	Address Stable from RD, WR, IORQ or MREQ	[3]		nsec	
	t_{caf}	Address Stable From RD or WR During Float	[4]		nsec	
D_{0-7}	$t_{D(D)}$	Data Output Delay		230	nsec	C_L = 50pF
	$t_{F(D)}$	Delay to Float During Write Cycle		90	nsec	
	$t_{S\Phi(D)}$	Data Setup Time to Rising Edge of Clock During M1 Cycle	50		nsec	
	$t_{S\overline{\Phi}(D)}$	Data Setup Time to Falling Edge of Clock During M2 to M5	60		nsec	
	t_{dcm}	Data Stable Prior to WR (Memory Cycle)	[5]		nsec	
	t_{dci}	Data Stable Prior to WR (I/O Cycle)	[6]		nsec	
	t_{cdf}	Data Stable From WR	[7]		nsec	
	t_{ll}	Any Hold Time for Setup Time	0		nsec	
MREQ	$t_{DL\overline{\Phi}(MR)}$	MREQ Delay From Falling Edge of Clock, MREQ Low		100	nsec	C_L = 50pF
	$t_{DH\Phi(MR)}$	MREQ Delay From Rising Edge of Clock, MREQ High		100	nsec	
	$t_{DH\overline{\Phi}(MR)}$	MREQ Delay From Falling Edge of Clock, MREQ High		100	nsec	
	$t_{w(MRL)}$	Pulse Width, MREQ Low	[8]		nsec	
	$t_{w(MRH)}$	Pulse Width, MREQ High	[9]		nsec	
IORQ	$t_{DL\Phi(IR)}$	IORQ Delay From Rising Edge of Clock, IORQ Low		90	nsec	C_L = 50pF
	$t_{DL\overline{\Phi}(IR)}$	IORQ Delay From Falling Edge of Clock, IORQ Low		110	nsec	
	$t_{DH\Phi(IR)}$	IORQ Delay From Rising Edge of Clock, IORQ High		100	nsec	
	$t_{DH\overline{\Phi}(IR)}$	IORQ Delay From Falling Edge of Clock, IORQ High		110	nsec	
RD	$t_{DL\Phi(RD)}$	RD Delay From Rising Edge of Clock, RD Low		100	nsec	C_L = 50pF
	$t_{DL\overline{\Phi}(RD)}$	RD Delay From Falling Edge of Clock, RD Low		130	nsec	
	$t_{DH\Phi(RD)}$	RD Delay From Rising Edge of Clock, RD High		100	nsec	
	$t_{DH\overline{\Phi}(RD)}$	RD Delay From Falling Edge of Clock, RD High		110	nsec	
WR	$t_{DL\Phi(WR)}$	WR Delay From Rising Edge of Clock, WR Low		80	nsec	C_L = 50pF
	$t_{DL\overline{\Phi}(WR)}$	WR Delay From Falling Edge of Clock, WR Low		90	nsec	
	$t_{DH\overline{\Phi}(WR)}$	WR Delay From Falling Edge of Clock, WR High		100	nsec	
	$t_{w(WRL)}$	Pulse Width, WR Low	[10]		nsec	
M1	$t_{DL(M1)}$	M1 Delay From Rising Edge of Clock, M1 Low		130	nsec	C_L = 50pF
	$t_{DH(M1)}$	M1 Delay From Rising Edge of Clock, M1 High		130	nsec	
RFSH	$t_{DL(RF)}$	RFSH Delay From Rising Edge of Clock, RFSH Low		180	nsec	C_L = 50pF
	$t_{DH(RF)}$	RFSH Delay From Rising Edge of Clock, RFSH High		150	nsec	
WAIT	$t_{s(WT)}$	WAIT Setup Time to Falling Edge of Clock	70		nsec	
HALT	$t_{D(HT)}$	HALT Delay Time From Falling Edge of Clock		300	nsec	C_L = 50pF
INT	$t_{s(IT)}$	INT Setup Time to Rising Edge of Clock	80		nsec	
NMI	$t_{w(NML)}$	Pulse Width, NMI Low	80		nsec	
BUSRQ	$t_{s(BQ)}$	BUSRQ Setup Time to Rising Edge of Clock	80		nsec	
BUSAK	$t_{DL(BA)}$	BUSAK Delay From Rising Edge of Clock, BUSAK Low		120	nsec	C_L = 50pF
	$t_{DH(BA)}$	BUSAK Delay From Falling Edge of Clock, BUSAK High		110	nsec	
RESET	$t_{s(RS)}$	RESET Setup Time to Rising Edge of Clock	90		nsec	
	$t_{F(C)}$	Delay to Float (MREQ, IORQ, RD and WR)		100	nsec	
	t_{mr}	M1 Stable Prior to IORQ (Interrupt Ack.)	[11]		nsec	

[1] $t_{acm} = t_{w(\Phi H)} + t_f - 75$

[2] $t_{aci} = t_c - 80$

[3] $t_{ca} = t_{w(\Phi L)} + t_r - 40$

[4] $t_{caf} = t_{w(\Phi L)} + t_r - 60$

[5] $t_{dcm} = t_c - 210$

[6] $t_{dci} = t_{w(\Phi L)} + t_r - 210$

[7] $t_{cdf} = t_{w(\Phi L)} + t_r - 80$

[8] $t_{w(MRL)} = t_c - 40$

[9] $t_{w(MRH)} = t_{w(\Phi H)} + t_r - 30$

[10] $t_{w(WRL)} = t_c - 40$

[11] $t_{mr} = 3t_c + t_{w(\Phi H)} + t_f - 80$

NOTES:

A. Data should be enabled onto the CPU data bus when RD is active. During interrupt acknowledge data should be enabled when M1 and IORQ are both active.

B. All control signals are internally synchronized, so they may be totally asynchronous with respect to the clock.

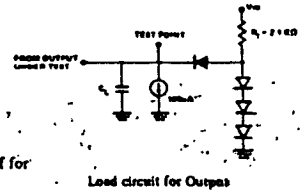
C. The RESET signal must be active for a minimum of 3 clock cycles.

D Output Delay vs. Loaded Capacitance

TA = 70°C Vcc = +5V ±5%

Add 10nsec delay for each 50pf increase in load up to a maximum of 200pf for the data bus & 100pf for address & control lines

E. Although static by design, testing guarantees $t_{w(\Phi H)}$ of 200 μsec maximum

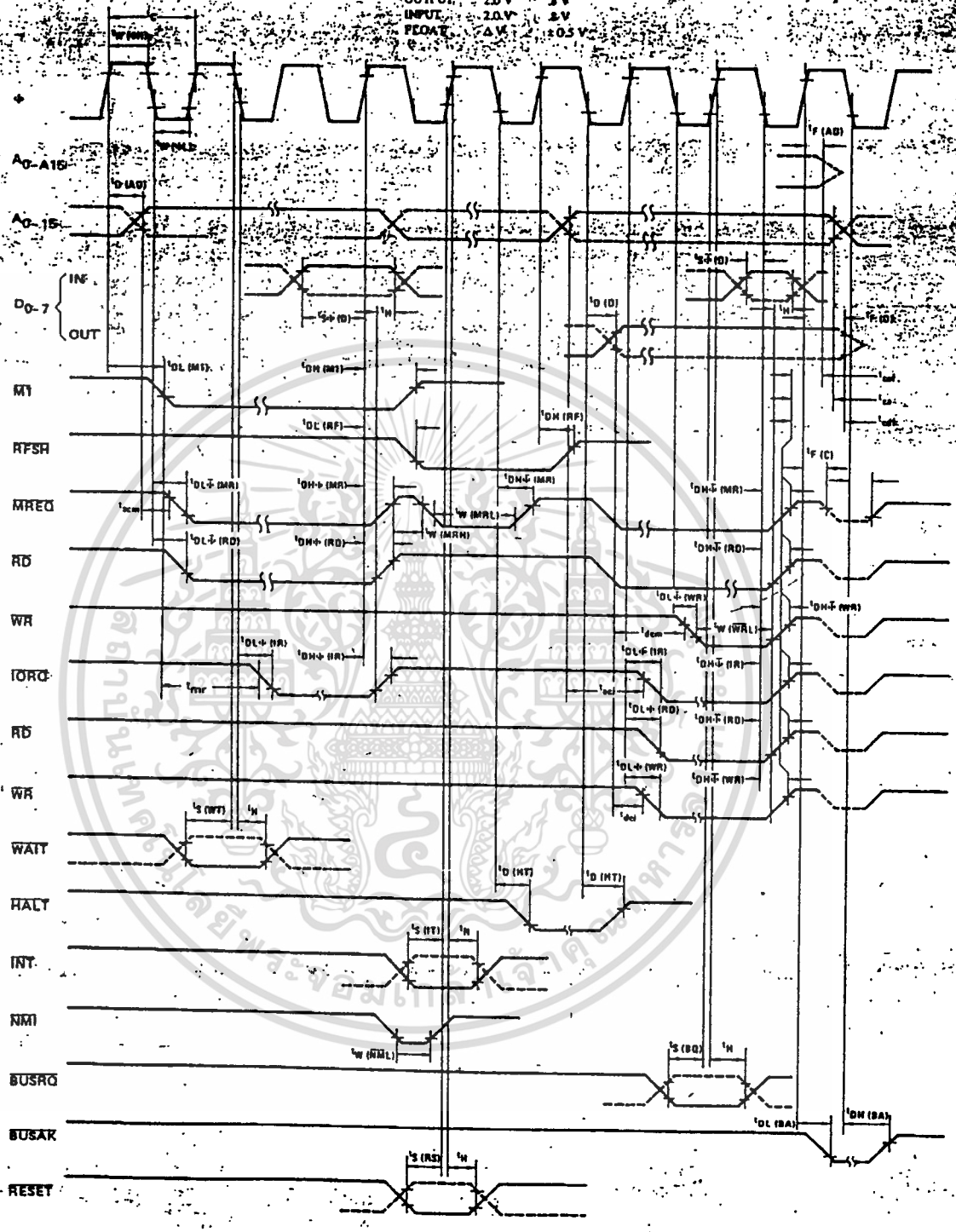


Load circuit for Outputs

Timing Diagram

Timing measurements are made at the following voltages unless otherwise specified:

	"1"	"0"
CLOCK	V_{CC} -.6V	.45V
OUTPUT	2.0 V	.8 V
INPUT	2.0 V	.8 V
FLOAT	$\Delta V = \pm 0.5$ V	



Absolute Maximum Ratings

Temperature Under Bias	Specified operating range.
Storage Temperature	-65°C to +150°C
Voltage On Any Pin with Respect to Ground	-0.3V to +7V
Power Dissipation	1.5W

*Comment

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

Note: For Z80-CPU all AC and DC characteristics remain the same for the military grade parts except I_{CC} = 200 mA.

Z80-CPU D.C. Characteristics

T_A = 0°C to 70°C, V_{CC} = 5V ± 5% unless otherwise specified

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Condition
V_{ILC}	Clock Input Low Voltage	-0.3		0.45	V	
V_{IHC}	Clock Input High Voltage	V_{CC} -.6		V_{CC}+.3	V	
V_{IL}	Input Low Voltage	-0.3		0.8	V	
V_{IH}	Input High Voltage	2.0		V_{CC}	V	
V_{OL}	Output Low Voltage			0.4	V	I_{OL}=1.8mA
V_{OH}	Output High Voltage	2.4			V	I_{OH} = -250μA
I_{CC}	Power Supply Current			150	mA	
I_{LI}	Input Leakage Current			10	μA	V_{IN}=0 to V_{CC}
I_{LOH}	Tri-State Output Leakage Current in Float			10	μA	V_{OUT}=2.4 to V_{CC}
I_{LOL}	Tri-State Output Leakage Current in Float			-10	μA	V_{OUT}=0.4V
I_{LD}	Data Bus Leakage Current in Input Mode			±10	μA	$0 \le V_{IN} \le V_{CC}$

Capacitance

T_A = 25°C, f = 1 MHz,
unmeasured pins returned to ground

Symbol	Parameter	Max.	Unit
C_ϕ	Clock Capacitance	35	pF
C_{IN}	Input Capacitance	5	pF
C_{OUT}	Output Capacitance	10	pF

Z80-CPU Ordering Information

C – Ceramic
P – Plastic
S – Standard 5V ±5% 0° to 70°C
E – Extended 5V ±5% -40° to 85°C
M – Military 5V ±10% -55° to 125°C

Z80A-CPU D.C. Characteristics

T_A = 0°C to 70°C, V_{CC} = 5V ± 5% unless otherwise specified

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Condition
V_{ILC}	Clock Input Low Voltage	-0.3		0.45	V	
V_{IHC}	Clock Input High Voltage	V_{CC} -.6		V_{CC}+.3	V	
V_{IL}	Input Low Voltage	-0.3		0.8	V	
V_{IH}	Input High Voltage	2.0		V_{CC}	V	
V_{OL}	Output Low Voltage			0.4	V	I_{OL}=1.8mA
V_{OH}	Output High Voltage	2.4			V	I_{OH} = -250μA
I_{CC}	Power Supply Current		90	200	mA	
I_{LI}	Input Leakage Current			10	μA	V_{IN}=0 to V_{CC}
I_{LOH}	Tri-State Output Leakage Current in Float			10	μA	V_{OUT}=2.4 to V_{CC}
I_{LOL}	Tri-State Output Leakage Current in Float			-10	μA	V_{OUT}=0.4V
I_{LD}	Data Bus Leakage Current in Input Mode			±10	μA	$0 \le V_{IN} \le V_{CC}$

Capacitance

T_A = 25°C, f = 1 MHz,
unmeasured pins returned to ground

Symbol	Parameter	Max.	Unit
C_ϕ	Clock Capacitance	35	pF
C_{IN}	Input Capacitance	5	pF
C_{OUT}	Output Capacitance	10	pF

Z80A-CPU Ordering Information

C – Ceramic
P – Plastic
S – Standard 5V ±5% 0° to 70°C

T_A = 0°C to 70°C, Vcc = +5V ± 5%, Unless Otherwise Noted.

Signal	Symbol	Parameter	Min	Max	Unit	Test Condition	
Φ	t_c	Clock Period	.25	[12]	μsec		[12] $t_c = t_{w(\Phi H)} + t_{w(\Phi L)} + t_r + t_f$
	$t_{w(\Phi H)}$	Clock Pulse Width, Clock High	110	[E]	nsec		
	$t_{w(\Phi L)}$	Clock Pulse Width, Clock Low	110	2000	nsec		
	t_r, t_f	Clock Rise and Fall Time		30	nsec		
A_{0-15}	$t_{D(AD)}$	Address Output Delay		110	nsec	C_L = 50pF	
	$t_{F(AD)}$	Delay to Float		90	nsec		
	t_{acm}	Address Stable Prior to $\overline{MREQ}$ (Memory Cycle)	[1]		nsec		[1] $t_{acm} = t_{w(\Phi H)} + t_f - 65$
	t_{aci}	Address Stable Prior to $\overline{IORQ}$, $\overline{RD}$ or $\overline{WR}$ (I/O Cycle)	[11]		nsec		
	t_{ca}	Address Stable from $\overline{RD}$, $\overline{WR}$, $\overline{IORQ}$ or $\overline{MREQ}$	[2]		nsec		[2] $t_{aci} = t_c - 70$
	t_{caf}	Address Stable From $\overline{RD}$ or $\overline{WR}$ During Float	[3]		nsec		[3] $t_{ca} = t_{w(\Phi L)} + t_r - 50$
			[4]		nsec		[4] $t_{caf} = t_{w(\Phi L)} + t_r - 45$
D_{0-7}	$t_{D(D)}$	Data Output Delay		150	nsec	C_L = 50pF	
	$t_{F(D)}$	Delay to Float During Write Cycle		90	nsec		
	$t_{S\Phi(D)}$	Data Setup Time to Rising Edge of Clock During M1 Cycle	-35		nsec		[5] $t_{dcm} = t_c - 170$
	$t_{S\overline{\Phi}(D)}$	Data Setup Time to Falling Edge of Clock During M2 to M5	50		nsec		[6] $t_{dci} = t_{w(\Phi L)} + t_r - 170$
	t_{dcm}	Data Stable Prior to $\overline{MREQ}$ (Memory Cycle)	[5]		nsec		[7] $t_{cdf} = t_{w(\Phi L)} + t_r - 70$
	t_{dci}	Data Stable Prior to $\overline{WR}$ (I/O Cycle)	[6]		nsec		
	t_{cdf}	Data Stable From $\overline{WR}$	[7]				
	t_H	Any Hold Time for Setup Time		0	nsec		
$\overline{MREQ}$	$t_{DL\Phi(MR)}$	$\overline{MREQ}$ Delay From Falling Edge of Clock, $\overline{MREQ}$ Low		85	nsec	C_L = 50pF	
	$t_{DH\Phi(MR)}$	$\overline{MREQ}$ Delay From Rising Edge of Clock, $\overline{MREQ}$ High		85	nsec		
	$t_{DH\overline{\Phi}(MR)}$	$\overline{MREQ}$ Delay From Falling Edge of Clock, $\overline{MREQ}$ High		85	nsec		
	$t_{w(\overline{MRL})}$	Pulse Width, $\overline{MREQ}$ Low	[8]		nsec		[8] $t_{w(\overline{MRL})} = t_c - 30$
	$t_{w(\overline{MRH})}$	Pulse Width, $\overline{MREQ}$ High	[9]		nsec		[9] $t_{w(\overline{MRH})} = t_{w(\Phi H)} + t_f - 20$
$\overline{IORQ}$	$t_{DL\Phi(IR)}$	$\overline{IORQ}$ Delay From Rising Edge of Clock, $\overline{IORQ}$ Low		75	nsec	C_L = 50pF	
	$t_{DL\overline{\Phi}(IR)}$	$\overline{IORQ}$ Delay From Falling Edge of Clock, $\overline{IORQ}$ Low		85	nsec		
	$t_{DH\Phi(IR)}$	$\overline{IORQ}$ Delay From Rising Edge of Clock, $\overline{IORQ}$ High		85	nsec		
	$t_{DH\overline{\Phi}(IR)}$	$\overline{IORQ}$ Delay From Falling Edge of Clock, $\overline{IORQ}$ High		85	nsec		
$\overline{RD}$	$t_{DL\Phi(RD)}$	$\overline{RD}$ Delay From Rising Edge of Clock, $\overline{RD}$ Low		85	nsec	C_L = 50pF	
	$t_{DL\overline{\Phi}(RD)}$	$\overline{RD}$ Delay From Falling Edge of Clock, $\overline{RD}$ Low		95	nsec		
	$t_{DH\Phi(RD)}$	$\overline{RD}$ Delay From Rising Edge of Clock, $\overline{RD}$ High		85	nsec		
	$t_{DH\overline{\Phi}(RD)}$	$\overline{RD}$ Delay From Falling Edge of Clock, $\overline{RD}$ High		85	nsec		
$\overline{WR}$	$t_{DL\Phi(WR)}$	$\overline{WR}$ Delay From Rising Edge of Clock, $\overline{WR}$ Low		65	nsec	C_L = 50pF	
	$t_{DL\overline{\Phi}(WR)}$	$\overline{WR}$ Delay From Falling Edge of Clock, $\overline{WR}$ Low		80	nsec		
	$t_{DH\overline{\Phi}(WR)}$	$\overline{WR}$ Delay From Falling Edge of Clock, $\overline{WR}$ High		80	nsec		
	$t_{w(\overline{WRL})}$	Pulse Width, $\overline{WR}$ Low	[10]		nsec		[10] $t_{w(\overline{WRL})} = t_c - 30$
$\overline{M1}$	$t_{DL(M1)}$	$\overline{M1}$ Delay From Rising Edge of Clock, $\overline{M1}$ Low		100	nsec	C_L = 50pF	
	$t_{DH(M1)}$	$\overline{M1}$ Delay From Rising Edge of Clock, $\overline{M1}$ High		100	nsec		
$\overline{RFSH}$	$t_{DL(RF)}$	$\overline{RFSH}$ Delay From Rising Edge of Clock, $\overline{RFSH}$ Low		130	nsec	C_L = 50pF	
	$t_{DH(RF)}$	$\overline{RFSH}$ Delay From Rising Edge of Clock, $\overline{RFSH}$ High		120	nsec		
$\overline{WAIT}$	$t_{s(WT)}$	$\overline{WAIT}$ Setup Time to Falling Edge of Clock	70		nsec		
$\overline{HALT}$	$t_{D(HT)}$	$\overline{HALT}$ Delay Time From Falling Edge of Clock		300	nsec	C_L = 50pF	
$\overline{INT}$	$t_{s(IT)}$	$\overline{INT}$ Setup Time to Rising Edge of Clock	80		nsec		
$\overline{NMI}$	$t_{w(\overline{NML})}$	Pulse Width, $\overline{NMI}$ Low	80		nsec		
$\overline{BUSRQ}$	$t_{s(BQ)}$	$\overline{BUSRQ}$ Setup Time to Rising Edge of Clock	50		nsec		
$\overline{BUSAK}$	$t_{DL(BA)}$	$\overline{BUSAK}$ Delay From Rising Edge of Clock, $\overline{BUSAK}$ Low		100	nsec	C_L = 50pF	
	$t_{DH(BA)}$	$\overline{BUSAK}$ Delay From Falling Edge of Clock, $\overline{BUSAK}$ High		100	nsec		
$\overline{RESET}$	$t_{s(RS)}$	$\overline{RESET}$ Setup Time to Rising Edge of Clock	60		nsec		
	$t_{F(C)}$	Delay to Float ($\overline{MREQ}$, $\overline{IORQ}$, $\overline{RD}$ and $\overline{WR}$)		80	nsec		
	t_{mr}	M1 Stable Prior to IORQ (Interrupt Ack.)	[11]		nsec		[11] $t_{mr} = 2t_c + t_{w(\Phi H)} + t_f - 65$

NOTES:

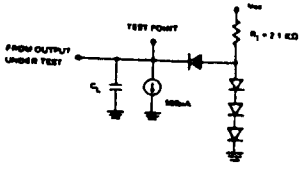
A. Data should be enabled onto the CPU data bus when $\overline{RD}$ is active. During interrupt acknowledge data should be enabled when $\overline{M1}$ and $\overline{IORQ}$ are both active.

B. All control signals are internally synchronized, so they may be totally asynchronous with respect to the clock.

C. The $\overline{RESET}$ signal must be active for a minimum of 3 clock cycles.

D. Output Delay vs. Loaded Capacitance
TA = 70°C Vcc = +5V ±5%
Add 10nsec delay for each 50pf increase in load up to maximum of 200pf for data bus and 100pf for address & control lines.

E. Although static by design, testing guarantees $t_{w(\Phi H)}$ of 200 μsec maximum



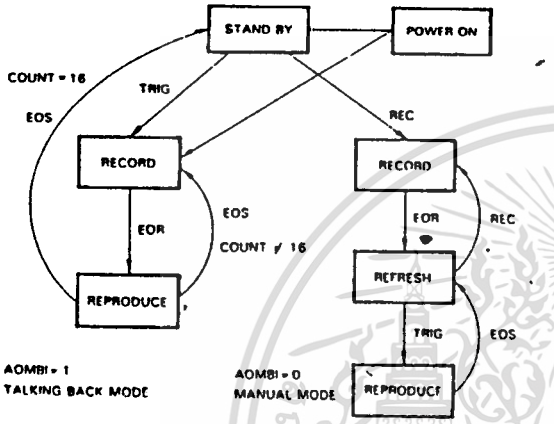
Load circuit for Output

Electrical Characteristics

Operation Voltage 9V
Stand-by Current ..... Typical 2μA at V_{DD} = 5V
Output Buffer Current
LEDO 6mA
IGPO 5mA
MICG 1.5mA
DRAMPD 2.5mA
Others 0.5mA

Functional Description

The operation of UM5101 is very simple. It can be generally described in the following state diagram below:



REC = RECIN pin is triggered
TRIG = TRIG pin is triggered
EOR = End Of Record
EOS = End Of Speech

Talking Back Mode

It will enter the record state after power is on or the TRIG pin is stimulated in stand-by state. When the recording has been completed, it will automatically change to reproduce state. If the counting number is less than sixteen, it will go to record state again after reproducing is finished. This procedure will continue until the counting number equals sixteen. After that, it goes to standby state.

Manual Mode

In this mode it will stay in stand-by only after power on. The trigger of the REC pin makes it enter the record state. After the end of the record, it goes to refresh state in order to keep the DRAM data. It can enter the reproduce state when it is in refresh state and TRIG pin is stimulated. However it must return to refresh state after the end of recording or reproducing.

Pin Description

AMPI, AMPO

Microphone amplifier input and output. Amplifier gain can be controlled via external resistor connected between these two pins.

LSCTR

Low speed control pin is used to guarantee the RAS width meets the requirement of DRAM. When low sample rate is desired, this pin should be in "H" level and the sample rate is equal to $f_{OSC}/32$ (f_{OSC} = Oscillating frequency). When normal this pin is in "L" level and the sample rate is $f_{OSC}/16$. However this pin should be in "H" when sample rate is below 12.5KHz.

SPEED

Speed controls the output voice rate. As this pin is in "H" level, the output rate will increase by two during voice reproduction.

TWN

If this pin is in "H" level, the voice output will be reproduced twice after each trigger.

AOMBI

Automatic or manual mode control pin indicate the current mode of this IC. When this pin is in "H", it is in talking back mode. After a trigger it will start to record the voice and then reproduce it. The process will be carried out 16 times. When this pin is in "L", it is in manual mode. It will stay in stand-by after power is on. If REC pin is triggered, it will start to record and go to refresh state after recording end. Then it will enter record or reproduce state by stimulating REC pin or TRIG pin.

OTB

This pin is used to indicate one or two DRAMs being connected externally. When OTB is "H", one DRAM is connected. It should be "L" if two DRAMs are connected.

OSCI, OSCO

Oscillator input and output pin. A resitor is placed between these two pins in order to oscillate. If 11 pin OSCO is triggered low, the talking back function will pause for a period as long as trigger time.

STOP

If talking back operates over two times and the STOP pin is triggered by high, then talking back function will immediately stop.

RECIN

In manual mode, it will enter record state when this pin is triggered.

TRIG

In talking back mode, it will carry out record and reproduce 16 times, when this pin is triggered. In manual mode, it will enter reproduce state when this pin is triggered, a CdS can be used to trigger this pin.

V_{DD}, V_{SS}

Power pins.

A0 ~ A7

DRAM address pins.

WEB, RASB, CAS1B, CAS2B

DRAM control timing pins.

DI, DO

DRAM data input and output pins.

LEDO

N-channel open drain output. It will stay in "L" during the record state.

MICG

N-channel open drain output. It will stay in "L" during the record state.

DRAMPD

N-channel open drain output. It will stay in "L" when it is not in standby state.

TGFO

P-channel open drain output. It will stay in "H" during the reproduce state.

PAIP, PABA, PAIN, PAOP, PAOC, PAON

Power amplifier input and output pins. PAIP is positive terminal. PAOP is used to drive PNP transistor. PAON is used to drive NPN transistor. PAOC is connected to output push-pull. PAIN is negative terminal. PABA is bias input pin.

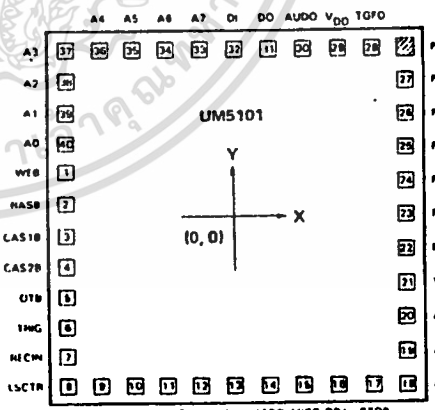
AUDO

D/A reproducing output.

V_{CC}

Power supply for logic-level conversion. The high level of output pins: A0-A7, DI, RASB, CAS1B, CAS2B, WEB is "V_{CC}". V_{CC} should less than or equal to V_{DD}.

Bonding Diagram

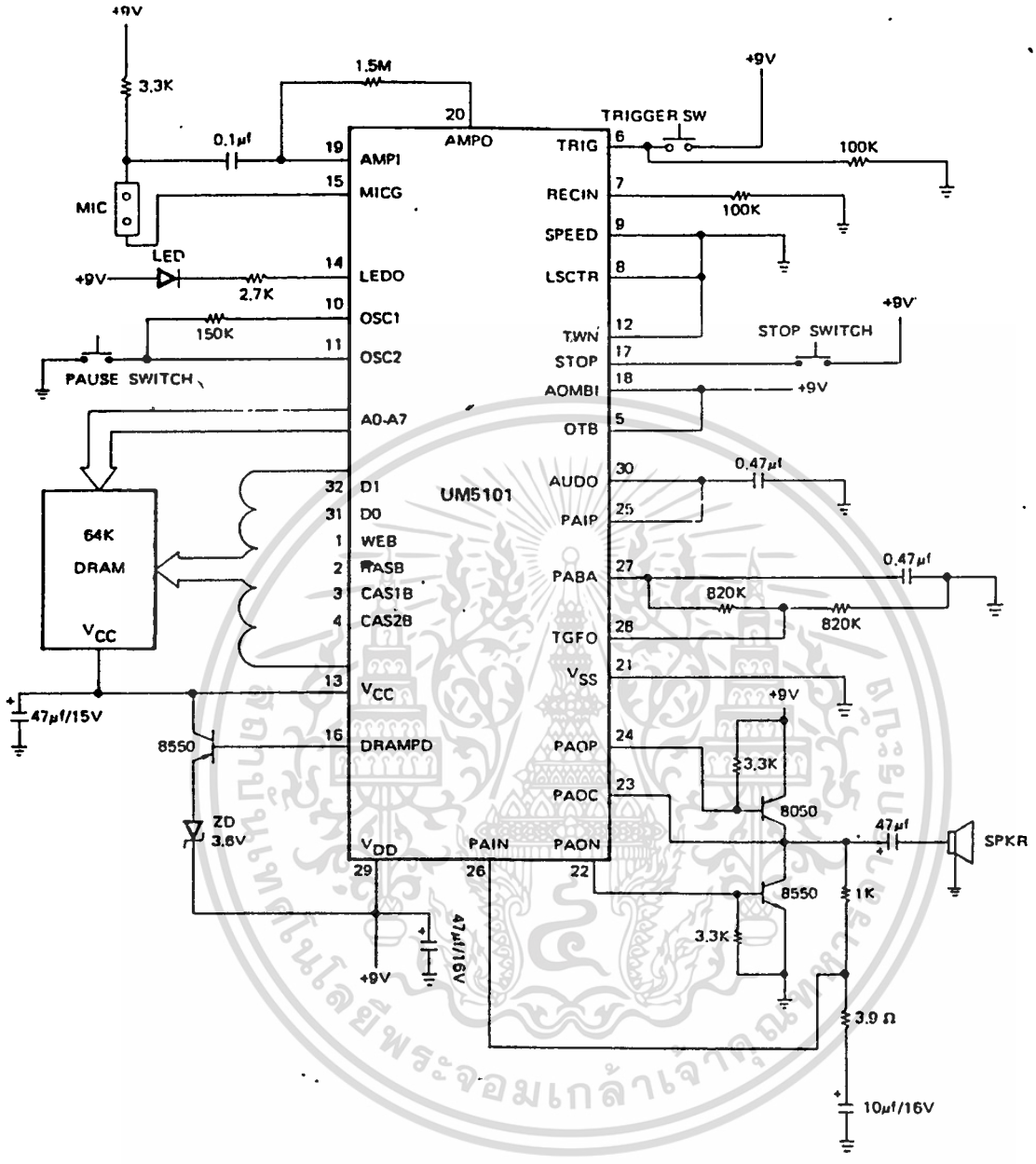


unit: μm

Pad No.	X	Y	Pad No.	X	Y
1	-1352.04	485.90	22	1389.89	-263.40
2	-1360.93	274.57	23	1368.04	-27.43
3	-1352.04	24.38	24	1368.04	226.06
4	-1352.04	-204.47	25	1368.04	454.91
5	-1352.04	-440.94	26	1368.04	694.44
6	-1360.93	-680.47	27	1368.04	937.51
7	-1360.93	-927.10	[pad]	1368.04	1163.57
8	-1312.93	-1180.59	28	1003.05	1183.89
9	-1082.55	-1173.48	29	781.56	1180.59
10	-859.47	-1180.59	30	494.54	1180.59
11	-595.88	-1169.92	31	277.62	1191.01
12	-370.41	-1173.48	32	48.09	1180.59
13	77.47	-1253.49	33	-168.12	1183.89
14	288.00	-1253.49	34	-391.92	1183.89
15	550.93	-1253.49	35	-609.09	1183.89
16	764.03	-1253.49	36	-826.52	1180.59
17	1038.10	-1197.61	37	-1052.58	1183.89
18	1263.90	-1187.45	38	-1352.04	1152.91
19	1368.04	-958.09	39	1360.93	930.40
20	1368.04	-711.45	40	1352.04	708.41
21	1368.04	-489.46			

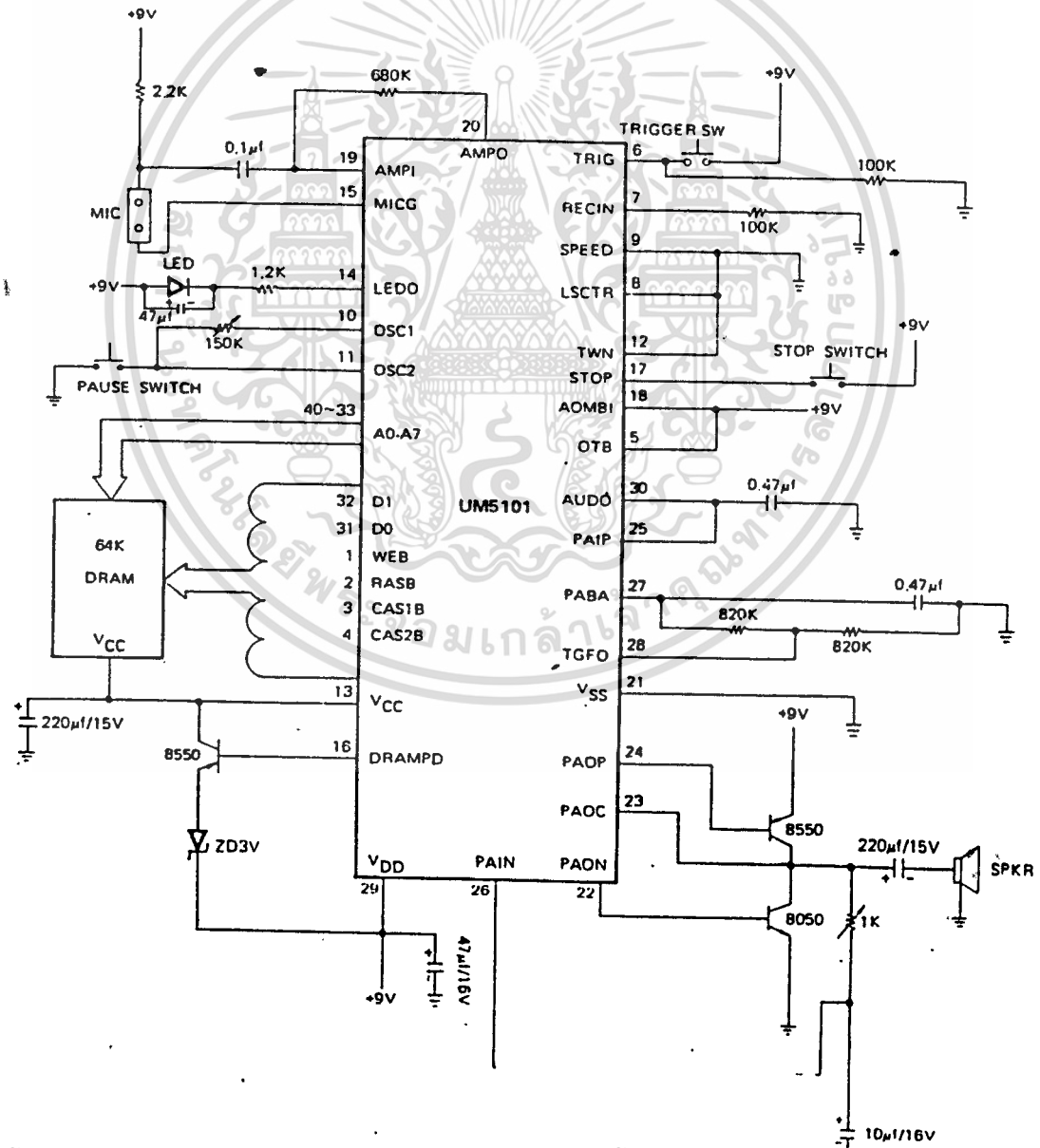
Application Circuit

(For Talking Back Mode)



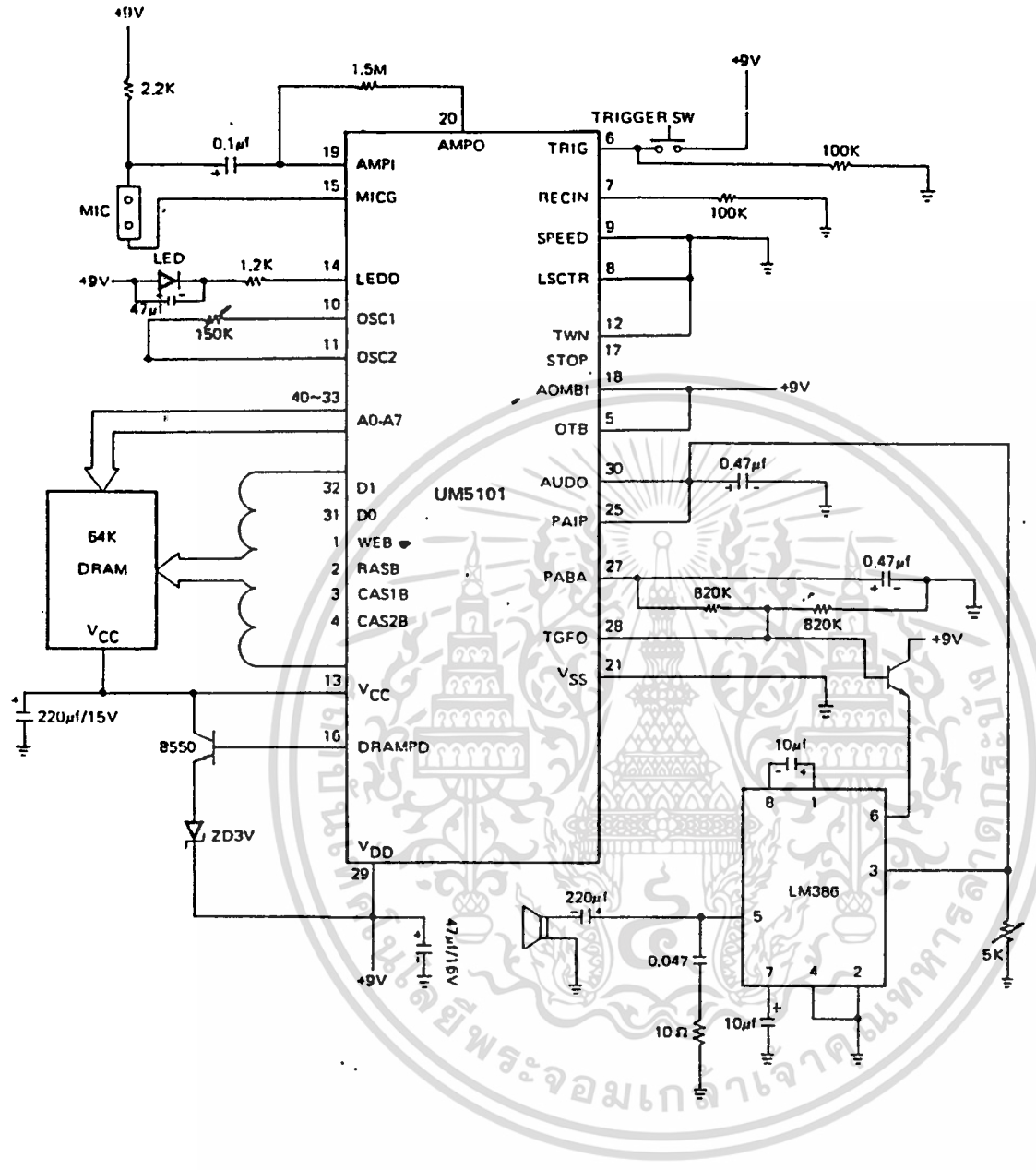
Application Circuit

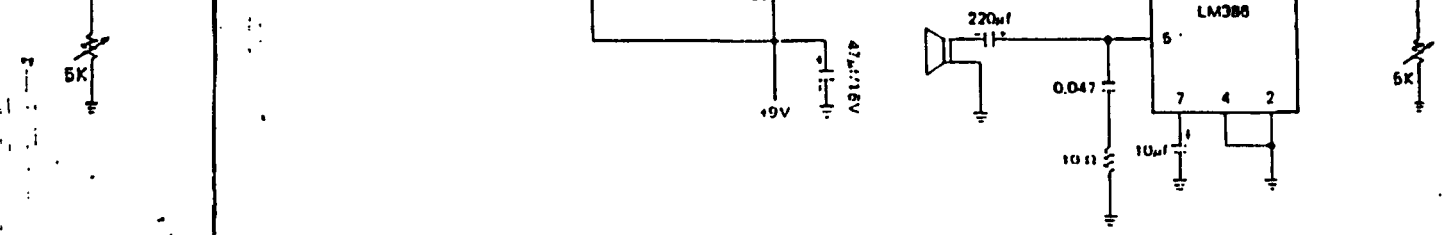
(For Talking Back Mode)



Application Circuit

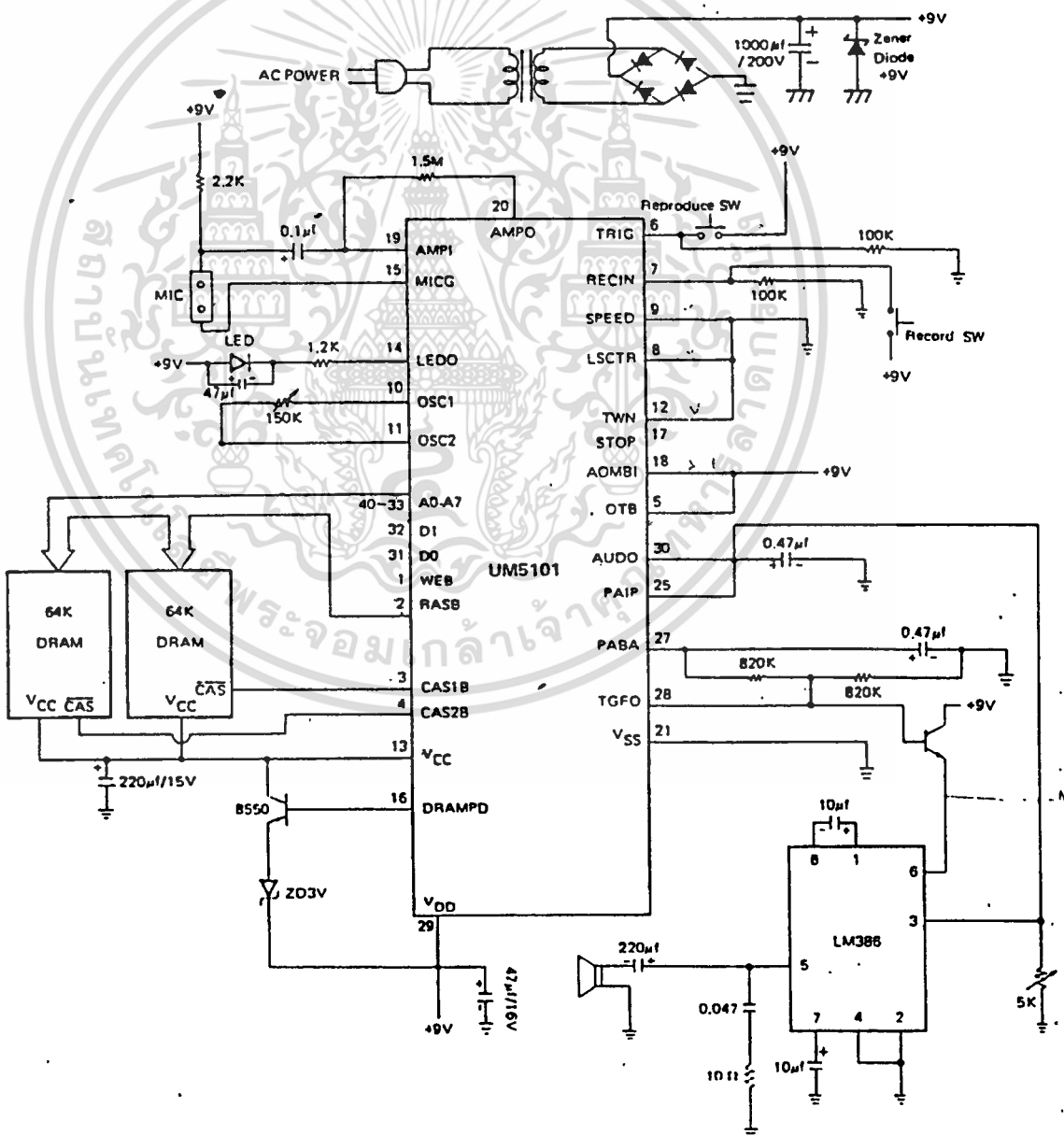
(For Talking Back Mode)





Application Circuit

(For Message Box with AC Power)



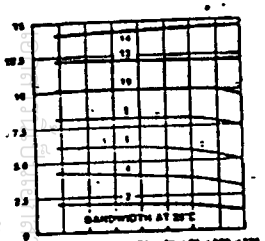


Figure 12. Bandwidth Variation With Temperature

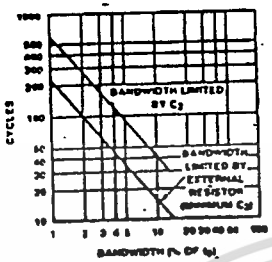


Figure 13. Greatest Number of Cycles Before Output

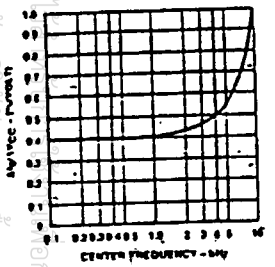


Figure 14. Power Supply Dependence of Center Frequency

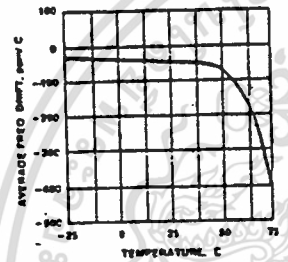


Figure 15. Typical Center Frequency Drift With Temperature (V^+ = 5V, R_1 = 80 kΩ, f_o = 1 kHz)

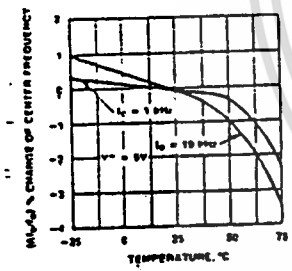


Figure 16. Typical Frequency Drift as a Function of Temperature

Monolithic Function Generator

GENERAL DESCRIPTION

The XR-2206 is a monolithic function generator integrated circuit capable of producing high quality sine, square, triangle, ramp, and pulse waveforms of high-stability and accuracy. The output waveforms can be both amplitude and frequency modulated by an external voltage. Frequency of operation can be selected externally over a range of 0.01 Hz to more than 1 MHz.

The circuit is ideally suited for communications, instrumentation, and function generator applications requiring sinusoidal tone, AM, FM, or FSK generation. It has a typical drift specification of 20 ppm/°C. The oscillator frequency can be linearly swept over a 2000:1 frequency range, with an external control voltage, having a very small effect on distortion.

FEATURES

Low Sine Wave Distortion	0.5%, Typical
Excellent Temperature Stability	20 ppm/°C, Typical
Wide Sweep Range	2000:1, Typical
Low Supply Sensitivity	0.01%/V, Typical
Linear Amplitude Modulation	
TTL Compatible FSK Controls	
Wide Supply Range	10V to 26V
Adjustable Duty Cycle	1% to 99%

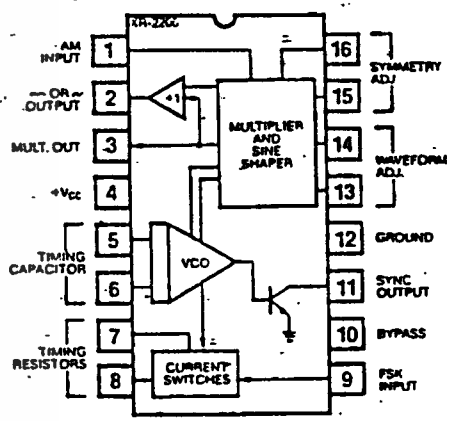
APPLICATIONS

Waveform Generation
Sweep Generation
AM/FM Generation
V/F Conversion
FSK Generation
Phase-Locked Loops (VCO)

ABSOLUTE MAXIMUM RATINGS

Power Supply	26V
Power Dissipation	750 mW
Derate Above 25°C	5 mW/°C
Total Timing Current	6 mA
Storage Temperature	−65°C to +150°C

FUNCTIONAL BLOCK DIAGRAM



ORDERING INFORMATION

Part Number	Package	Operating Temperature
XR-2206M	Ceramic	−55°C to +125°C
XR-2206N	Ceramic	0°C to +70°C
XR-2206P	Plastic	0°C to +70°C
XR-2206ON	Ceramic	0°C to +70°C
XR-2206CP	Plastic	0°C to +70°C

SYSTEM DESCRIPTION

The XR-2206 is comprised of four functional blocks: a voltage-controlled oscillator (VCO), an analog multiplier and sine-shaper; a unity gain buffer amplifier; and a set of current switches.

The VCO actually produces an output frequency proportional to an input current, which is produced by a resistor from the timing terminals to ground. The current switches route one of the timing pins current to the VCO controlled by an FSK input pin, to produce an output frequency. With two timing pins, two discrete output frequencies can be independently produced for FSK Generation Applications.

ELECTRICAL CHARACTERISTICS

Test Conditions: Test Circuit of Figure 1, $V^+ = 12V$, $T_A = 25°$, $C = 0.01\ \mu F$, $R_1 = 100\ k\Omega$, $R_2 = 10\ k\Omega$, $R_3 = 25\ k\Omega$ unless otherwise specified. S_1 open for triangle, closed for sine wave.

PARAMETERS	XR-2206M			XR-2206C			UNITS	CONDITIONS
	MIN	TYP	MAX	MIN	TYP	MAX		
GENERAL CHARACTERISTICS								
Single Supply Voltage	10		26	10		26	V	
Split-Supply Voltage	±5		±13	±5		±13	V	
Supply Current		12	17		.14	20	mA	$R_1 \geq 10\ k\Omega$
OSCILLATOR SECTION								
Max. Operating Frequency	0.5	1		0.5	1		MHz	$C = 1000\ pF$, $R_1 = 1\ k\Omega$
Lowest Practical Frequency		0.01			0.01		Hz	$C = 50\ \mu F$, $R_1 = 2\ M\Omega$
Frequency Accuracy		±1	±4		±2		% of f_o	$f_o = 1/R_1C$
Temperature Stability		±10	±50		±20		ppm/°C	0°C ≤ T_A ≤ 70°C, $R_1 = R_2 = 20\ k\Omega$
Supply Sensitivity		0.01	0.1		0.01		%/V	$V_{LOW} = 10V$, $V_{HIGH} = 20V$. $R_1 = R_2 = 20\ k\Omega$
Sweep Range	1000:1	2000:1			2000:1		$I_H = I_L$	f_H @ $R_1 = 1\ k\Omega$ f_L @ $R_1 = 2\ M\Omega$
Sweep Linearity								
10:1 Sweep		2			2		%	$f_L = 1\ kHz$, $f_H = 10\ kHz$
1000:1 Sweep		8			8		%	$f_L = 100\ kHz$, $f_H = 100\ kHz$
FM Distortion		0.1			0.1		%	±10% Deviation
Recommended Timing Components								
Timing Capacitor: C	0.001		100	0.001		100	μF	See Figure 4.
Timing Resistors: R_1 & R_2	1		2000	1		2000	kΩ	
Triangle Sine Wave Output								See Note 1, Figure 2.
Triangle Amplitude		160			160		mV/k Ω	Figure 1, S_1 Open
Sine Wave Amplitude	40	60	80		60		mV/k Ω	Figure 1, S_1 Closed
Max. Output Swing		6			6		V p-p	
Output Impedance		600			600		Ω	
Triangle Linearity		1			1		%	
Amplitude Stability		0.5			0.5		dB	For 1000:1 Sweep
Sine Wave Amplitude Stability		4800			4800		ppm/°C	See Note 2.
Sine Wave Distortion								
Without Adjustment		2.5			2.5		%	$R_1 = 30\ k\Omega$
With Adjustment		0.4	1.0		0.5	1.5	%	See Figures 6 and 7.
Amplitude Modulation								
Input Impedance	50	100		50	100		kΩ	
Modulation Range		100			100		%	
Carrier Suppression		55			55		dB	
Linearity		2			2		%	For 95% modulation
Square-Wave Output								
Amplitude		12			12		V p-p	Measured at Pin 11.
Rise Time		250			250		nsec	$C_L = 10\ pF$
Fall Time		50			50		nsec	$C_L = 10\ pF$
Saturation Voltage		0.2	0.4		0.2	0.6	V	$I_L = 2\ mA$
Leakage Current		0.1	20		0.1	100	μA	$V_{11} = 26V$
FSK Keying Level (Pin 9)	0.8	1.4	2.4	0.8	1.4	2.4	V	See section on circuit controls
Reference Bypass Voltage	2.9	3.1	3.3	2.5	3	3.5	V	Measured at Pin 10.

Note 1: Output amplitude is directly proportional to the resistance, R_3, on Pin 3. See Figure 2.
Note 2: For maximum amplitude stability, R_3 should be a positive temperature coefficient resistor.

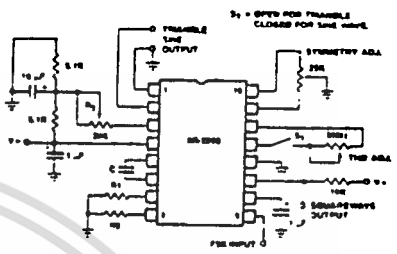


Figure 1. Basic Test Circuit.

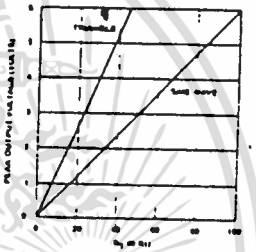


Figure 2. Output Amplitude as a Function of the Resistor, R_3, at Pin 3.

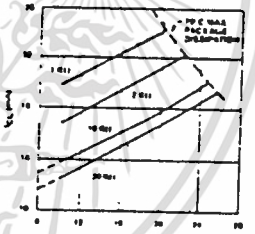


Figure 3. Supply Current versus Supply Voltage, Timing, R.

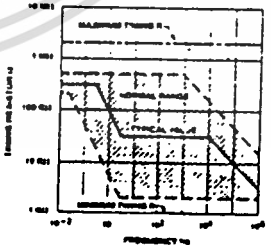


Figure 4. R versus Oscillation Frequency.

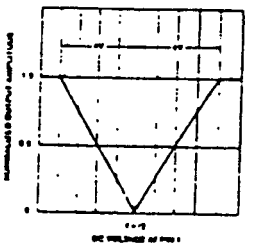


Figure 5. Normalized Output Amplitude versus DC Bias and AM Input (Pin 1).

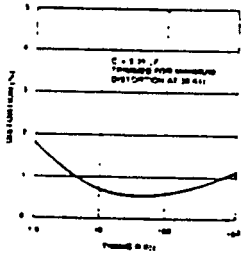


Figure 6. Trimmed Distortion versus Timing Resistor.

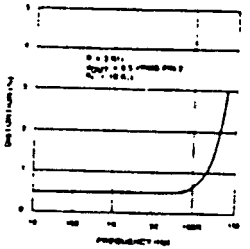


Figure 7. Sine Wave Distortion versus Operating Frequency with Timing Capacitors Varied.

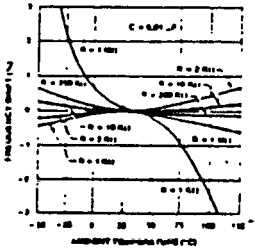


Figure 8. Frequency Drift versus Temperature.

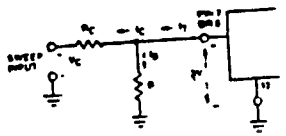


Figure 9. Circuit Connection for Frequency Sweep.

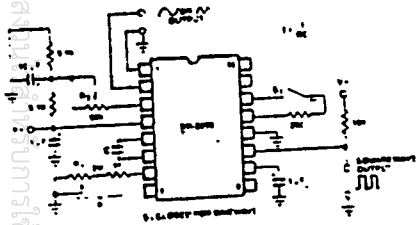


Figure 10. Circuit for Sine Wave Generation without External Adjustment. (See Figure 2 for Choice of R_3).

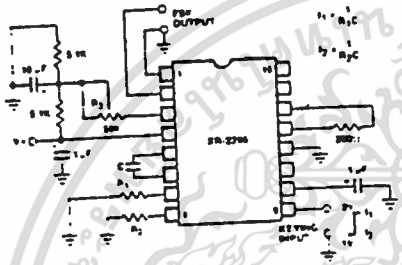


Figure 12. Sinusoidal FSK Generator.

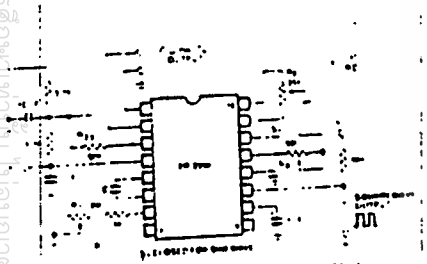


Figure 11. Circuit for Sine Wave Generation with Minimum Harmonic Distortion. (R_3 Determines Output Swing—See Figure 2.)

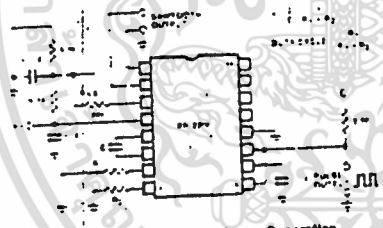


Figure 13. Circuit for Pulse and Ramp Generation.

Frequency-Shift Keying:

The XR-2206 can be operated with two separate timing resistors, R_1 and R_2, connected to the timing Pin 7 and 8 respectively, as shown in Figure 12. Depending on the polarity of the logic signal at Pin 9, either one or the other of these timing resistors is activated. If Pin 9 is open-circuited or connected to a bias voltage ≥ 2V, only R_1 is activated. Similarly, if the voltage level at Pin 9 is ≤1V, only R_2 is activated. Thus, the output frequency can be keyed between two levels, f_1 and f_2, as:

$$f_1 = 1/R_1C \text{ and } f_2 = 1/R_2C$$

For split-supply operation, the keying voltage at Pin 9 is referenced to V^-.

Output DC Level Control:

The dc level at the output (Pin 2) is approximately the same as the dc bias at Pin 3. In Figures 10, 11 and 12, Pin 3 is biased midway between V^+ and ground, to give an output dc level of $\approx V^+/2$.

APPLICATIONS INFORMATION

Sine Wave Generation

Without External Adjustment:

Figure 10 shows the circuit connection for generating a sinusoidal output from the XR-2206. The potentiometer, R_1 at Pin 7, provides the desired frequency tuning. The maximum output swing is greater than $V^+/2$, and the typical distortion (THD) is <2.5%. If lower sine wave distortion is desired, additional adjustments can be provided as described in the following section.

The circuit of Figure 10 can be converted to split-supply operation, simply by replacing all ground connections with V^-. For split-supply operation, R_3 can be directly connected to ground.

With External Adjustment:

The harmonic content of sinusoidal output can be reduced to ≈0.5% by additional adjustments as shown in Figure 11. The potentiometer, R_A, adjusts the sine-shaping resistor, and R_B provides the fine adjustment for the waveform symmetry. The adjustment procedure is as follows:

1. Set R_B at midpoint, and adjust R_A for minimum distortion.
2. With R_A set as above, adjust R_B to further reduce distortion.

Triangle Wave Generation

The circuits of Figures 10 and 11 can be converted to triangle wave generation, by simply open-circuiting Pin 13 and 14 (i.e., S_1 open). Amplitude of the triangle is approximately twice the sine wave output.

FSK Generation

Figure 12 shows the circuit connection for sinusoidal FSK signal operation. Mark and space frequencies can be independently adjusted, by the choice of timing resistors, R_1 and R_2; the output is phase-continuous during transitions. The keying signal is applied to Pin 9. The circuit can be converted to split-supply operation by simply replacing ground with V^-.

Pulse and Ramp Generation

Figure 13 shows the circuit for pulse and ramp waveform generation. In this mode of operation, the FSK keying terminal (Pin 9) is shorted to the square-wave output (Pin 11), and the circuit automatically frequency-shift keys itself between two separate frequencies during the positive-going and negative-going output waveforms. The pulse width and duty cycle can be adjusted from 1% to 99%, by the choice of R_1 and R_2. The values of R_1 and R_2 should be in the range of 1 kΩ to 2 MΩ.

PRINCIPLES OF OPERATION

Description of Controls

Frequency of Operation:

The frequency of oscillation, f_o, is determined by the external timing capacitor, C, across Pin 5 and 6, and by the timing resistor, R, connected to either Pin 7 or 8. The frequency is given as:

$$f_o = \frac{1}{RC} \text{ Hz}$$

and can be adjusted by varying either R or C. The recommended values of R, for a given frequency range, as shown in Figure 4. Temperature stability is optimum for 4 kΩ < R < 200 kΩ. Recommended values of C are from 1000 pF to 100 μF.

Frequency Sweep and Modulation:

Frequency of oscillation is proportional to the total timing current, I_T, drawn from Pin 7 or 8:

$$f = \frac{320\ I_T\ (\text{mA})}{C\ (\mu\text{F})} \text{ Hz}$$

Timing terminals (Pin 7 or 8) are low-impedance points, and are internally biased at +3V, with respect to Pin 12. Frequency varies linearly with I_T, over a wide range of current values, from 1 μA to 3 mA. The frequency can be controlled by applying a control voltage, V_C, to the activated timing pin as shown in Figure 9. The frequency of oscillation is related to V_C as:

$$f = \frac{1}{RC}\,1 + \frac{R}{R_C}\left(1 - \frac{V_C}{3}\right) \text{ Hz}$$

where V_C is in volts. The voltage-to-frequency conversion gain, K, is given as:

$$K = \Delta f / \Delta VC = -\frac{0.32}{R_C C} \text{ Hz/V}$$

CAUTION: For safety operation of the circuit, I_T should be limited to ≤3 mA.

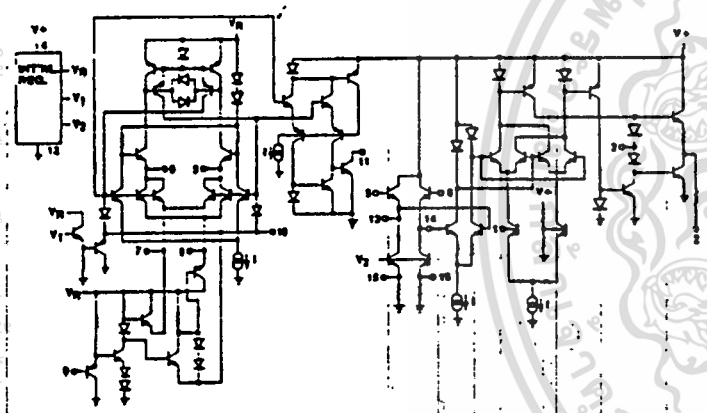
Output Amplitude:

Maximum output amplitude is inversely proportional to the external resistor, R_3, connected to Pin 3 (see Figure 2). For sine wave output, amplitude is approximately 60 mV peak per kΩ of R_3; for triangle, the peak amplitude is approximately 160 mV peak per kΩ of R_3. Thus, for example, R_3 = 50 kΩ would produce approximately ±3V sinusoidal output amplitude.

Amplitude Modulation:

Output amplitude can be modulated by applying a dc bias and a modulating signal to Pin 1. The internal impedance at Pin 1 is approximately 100 kΩ. Output amplitude varies linearly with the applied voltage at Pin 1, for values of dc bias at this pin, within ±4 volts of $V^+/2$ as shown in Figure 5. As this bias level approaches $V^+/2$, the phase of the output signal is reversed, and the amplitude goes through zero. This property is suitable for phase-shift keying and suppressed-carrier AM generation. Total dynamic range of amplitude modulation is approximately 55 dB.

CAUTION: AM control must be used in conjunction with a well-regulated supply, since the output amplitude now becomes a function of V^+.



EQUIVALENT SCHEMATIC DIAGRAM

Voltage-Controlled Oscillator

GENERAL DESCRIPTION

The XR-2207 is a monolithic voltage-controlled oscillator (VCO) integrated circuit featuring excellent frequency stability and a wide tuning range. The circuit provides simultaneous triangle and squarewave outputs over a frequency range of 0.01 Hz to 1 MHz. It is ideally suited for FM, FSK, and sweep or tone generation, as well as for phase-locked loop applications.

The XR-2207 has a typical drift specification of 20 ppm/°C. The oscillator frequency can be linearly swept over a 1000:1 range with an external control voltage; and the duty cycle of both the triangle and the squarewave outputs can be varied from 0.1% to 99.9% to generate stable pulse and sawtooth waveforms.

FEATURES

Excellent Temperature Stability (20 ppm/°C)
Linear Frequency Sweep
Adjustable Duty Cycle (0.1% to 99.9%)
Two or Four Level FSK Capability
Wide Sweep Range (1000:1 Min)
Logic Compatible Input and Output Levels
Wide Supply Voltage Range (±4V to ±13V)
Low Supply Sensitivity (0.1%/V)
Wide Frequency Range (0.01 Hz to 1 MHz)
Simultaneous Triangle and Squarewave Outputs

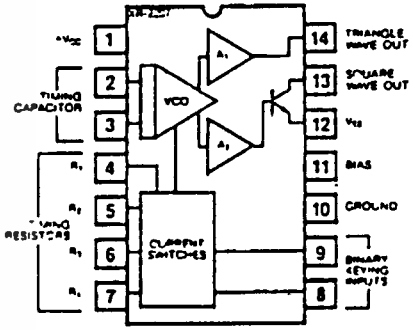
APPLICATIONS

FSK Generation
Voltage and Current-to-Frequency Conversion
Stable Phase-Locked Loop
Waveform Generation
  Triangle, Sawtooth, Pulse, Squarewave
FM and Sweep Generation

ABSOLUTE MAXIMUM RATINGS

Power Supply	26V
Power Dissipation (package limitation)	
Ceramic package	750 mW
Derate above +25°C	6.0 mW/°C
Plastic package	625 mW
Derate above +25°C	5 mW/°C
Storage Temperature Range	-65°C to +150°C

FUNCTIONAL BLOCK DIAGRAM



ORDERING INFORMATION

Part Number	Package	Operating Temperature
XR2207M	Ceramic	-55°C to +125°C
XR2207N	Ceramic	0°C to +70°C
XR2207P	Plastic	0°C to +70°C
XR2207CN	Ceramic	0°C to +70°C
XR2207CP	Plastic	0°C to +70°C

SYSTEM DESCRIPTION

The XR-2207 utilizes four main functional blocks for frequency generation. These are a voltage controlled oscillator (VCO), four current switches which are activated by binary keying inputs, and two buffer amplifiers for triangle and squarewave outputs. The VCO is actually a current controlled oscillator which gets its input from the current switches. As the output frequency is proportional to the input current, the VCO produces four discrete output frequencies. Two binary input pins determine which timing currents are channelled to the VCO. These currents are set by resistors to ground from each of the four timing terminals.

The triangle output buffer provides a low impedance output (100 TYP) while the squarewave is an open-collector type. A programmable reference point allows the XR-2207 to be used in either single or split supply configurations.

9