กล้องรักษาความปลอดภัยควบคุมผ่านอินเตอร์เน็ต

SECURITY CAMERA CONTROLLED VIA INTERNET

โดย

นาย ปิยพัชร ทรัพย์ทวี

นาย พลกฤษณ์ เชยแจ้ง

นาย อุกฤษฎ์ วารีเพชร

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2549

กล้องรักษาความปลอดภัยควบคุมผ่านอินเตอร์เน็ต

SECURITY CAMERA CONTROLLED VIA INTERNET

โดย

นายปิยพัชร	ทรัพย์ทวี	46012020
นายพลกฤษณ์	เชยแจ้ง	46012022
นายอุกฤษฎ์	วารีเพชร	46012041

อาจารย์ที่ปรึกษา

ผศ.ตร. พรชัย ทรัพย์นิธิ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2549

ปริญญานิพนธ์ปีการศึกษา 2549

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง กล้องรักษาความปลอดภัยควบคุมผ่านอินเตอร์เน็ต

SECURITY CAMERA CONTROLLED VIA INTERNET

ผู้จัดทำ

1. นาย ปิยพัชร ทรัพย์ทวี 46012020
2. นาย พลกฤษณ์ เชยแจ้ง 46012022
3. นาย อุกฤษฎ์ วารีเพชร 46012041

..... อาจารย์ที่ปรึกษา

(ผศ.ดร. พรชัย ทรัพย์นิธิ)

กล้องรักษาความปลอดภัยควบคุมผ่านอินเตอร์เน็ต

SECURITY CAMERA CONTROLLED

VIA INTERNET

โดย	นาย ปิยพัชร	ทรัพย์ทวี	46012020
	นาย พลกฤษณ์	เชยแจ้ง	46012022
	นาย อุกฤษฎ์	วารีเพชร	46012041

อาจารย์ที่ปรึกษา ผศ.ดร. พรชัย ทรัพย์นิธิ

บทคัดย่อ

โครงงานนี้เป็นโครงงานที่ใช้คอมพิวเตอร์ควบคุมกล้องโดยผ่านทางเครือข่ายอินเตอร์เน็ต โดยโครงงานชิ้นนี้ใช้เครื่องคอมพิวเตอร์ 2 เครื่อง โดยเครื่องหนึ่งทำหน้าที่เป็นเซิร์ฟเวอร์และฐานข้อมูลของภาพที่รับได้จากกล้อง ส่วนอีกเครื่องหนึ่งทำหน้าที่สั่งงานการทำงานของกล้องได้โดยควบคุมผ่านทางเครือข่ายอินเตอร์เน็ตไปยังเครื่องคอมพิวเตอร์เซิร์ฟเวอร์ เพื่อส่งผ่านคำสั่งไปยัง สเต็ปปิงมอเตอร์ ที่ควบคุมโดย MCS-51 เพื่อทำหน้าที่ควบคุมการหมุนของกล้อง

Abstract

This project is designed by using computers to control direction of the video camera via internet. This project consists of two computers. One computer acts as a server computer and a database for keeping video data rom the camera. Another computer acts as a camera-controller by sending command data via internet to the server computer, then the server computer will pass the command data to the stepping-motor which is controlled by MCS-51 for controlling the direction of the camera in desired position.

สารบัญ

สารบัญ (ต่อ)

สารบัญ (ต่อ)

สารบัญรูปภาพ

สารบัญรูปภาพ (ต่อ)

สารบัญรูปภาพ (ต่อ)

สารบัญตาราง

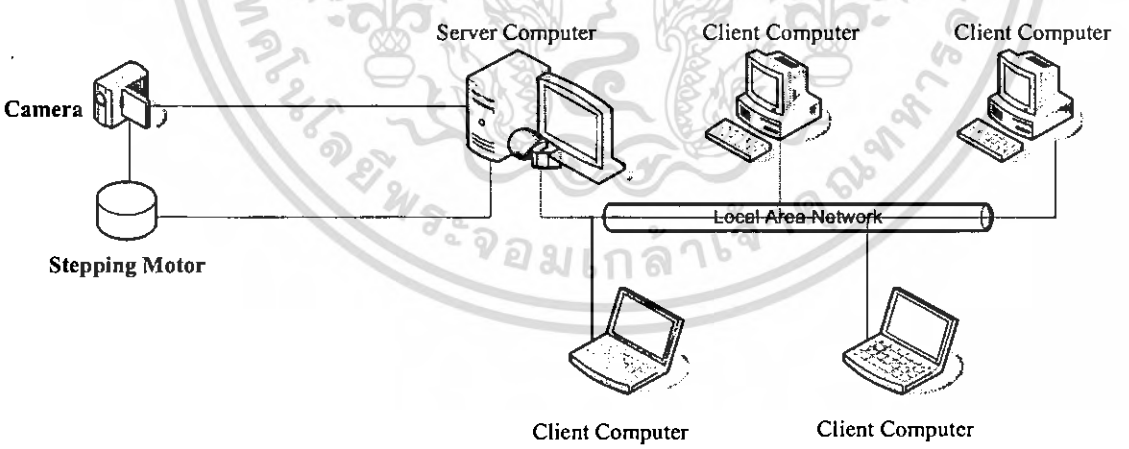
บทที่ 1
บทนำ

ในปัจจุบันเครือข่ายอินเทอร์เน็ตได้มีส่วนสำคัญในการสื่อสารอย่างมาก เนื่องจากเป็นเครือข่ายที่เชื่อมต่อกันได้ทั่วโลกมีบริการที่หลากหลายที่กระทำผ่านเครือข่าย และความเร็วในการส่งข้อมูลก็มีความเร็วอัตราข้อมูลเพิ่มขึ้น ด้วยเหตุนี้โครงงานนี้จึงได้นำเอาประโยชน์ของอินเทอร์เน็ตมาประยุกต์เข้ากับการรักษาความปลอดภัย โดยสามารถสังเกตการณ์พื้นที่ ที่ต้องการจากต่างพื้นที่ ที่สามารถเชื่อมโยงสู่เครือข่ายอินเทอร์เน็ตได้ โดยระบบประกอบด้วยกล้องที่ติดตั้งไว้ตามที่ต่างๆ ที่ส่งข้อมูลเข้าสู่คอมพิวเตอร์ที่ติดต่ออยู่กับเครือข่ายอินเทอร์เน็ต และส่งข้อมูลไปยังเครื่องปลายทางที่ต้องการดูภาพจากกล้องโดยผ่านเครือข่ายอินเทอร์เน็ตโดยเครื่องปลายทางจะต้องติดตั้งโปรแกรมที่ใช้เพื่อเรียกดูภาพจากกล้องโทรทัศน์วงจรปิด

1.1 วัตถุประสงค์ของโครงงาน

1. สร้างการเชื่อมต่อบนเครือข่ายระหว่างเครื่องคอมพิวเตอร์สองเครื่อง
2. เพื่อศึกษาการสื่อสารโดยส่งข้อมูลภาพเคลื่อนไหวผ่านเครือข่ายอินเตอร์เน็ต
3. เพื่อสร้างวงจรควบคุมสเต็ปปิ้งมอเตอร์ ที่ควบคุมโดย MCS-51 เพื่อทำหน้าที่บังคับการหมุนของฐานกล้อง

1.2 โครงสร้างโดยรวมของโครงงาน



รูปที่ 1.1 โครงสร้างของระบบ

หมายเหตุ Camera = กล้องโทรทัศน์วงจรปิด

Stepping Motor = มอเตอร์สเต็ปปิ้ง

Local Area Network = เครือข่ายท้องถิ่น

บทที่ 2
ทฤษฎีและหลักการ

2.1 กล้องโทรทัศน์วงจรปิด

โดยทั่วไปแล้วจะถูกแบ่งง่ายๆ ออกเป็นสองชนิดคือ กล้องสี และกล้องขาวดำซึ่งขึ้นอยู่กับความต้องการ ของผู้ใช้และสถานที่ในการติดตั้งซึ่งต้องประกอบไปด้วยลักษณะของการใช้งานจริง ยกตัวอย่างเช่น

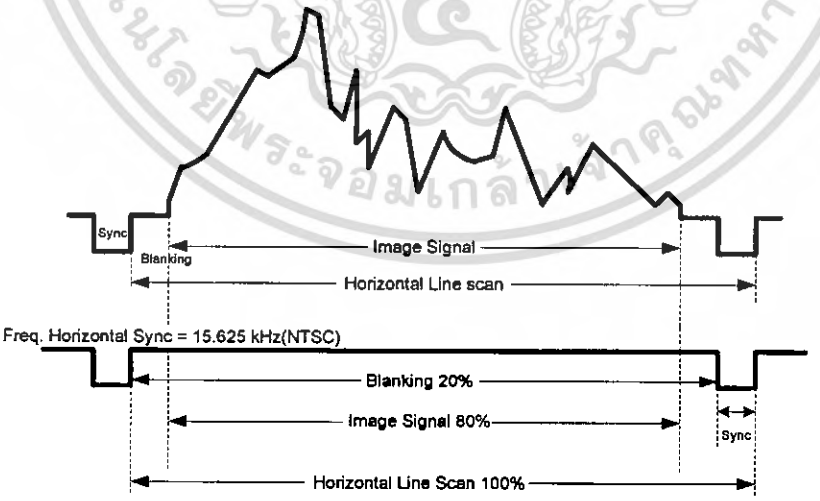
กล้องสี ควรใช้งานกับสถานที่ที่มีแสงสม่ำเสมอ เช่น ซูปเปอร์มาเก็ต, มินิมาร์ท, ร้านทอง เป็นต้น จากกลุ่มที่ยกตัวอย่างให้เห็นนั้นมีความเหมาะสมกล่าวคือ กล้องสีสามารถแยกแยะรายละเอียดหรือสีของสิ่งของได้ดี และในสถานที่ที่ยกตัวอย่างดังกล่าวก็มีการใช้แสงสว่างค่อนข้างมาก และสม่ำเสมอ ภาพที่มาปรากฏบนหน้าจอมอนิเตอร์ ก็จะมีความชัดเจน ซึ่งกล้องสีจะแบ่งออกได้เป็นสองชนิดคือ

กล้อง CCD (Charge Coupled Device) และกล้อง CMOS (Complementary Metal Oxide Semiconductor)

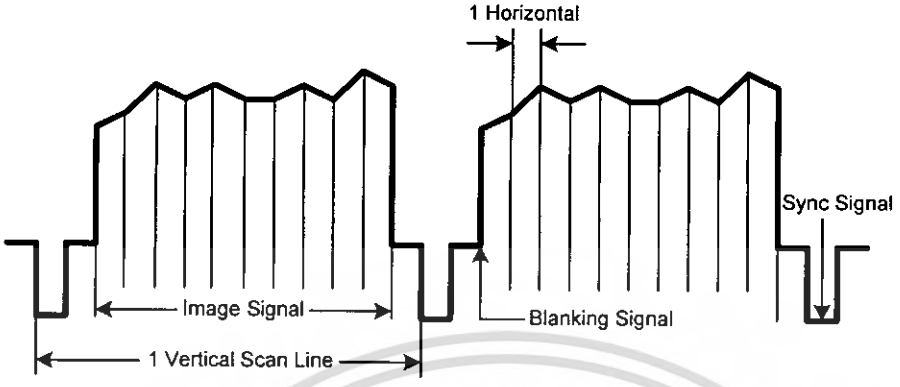
กล้องขาวดำ กล้องชนิดนี้เป็นกล้องที่ใช้แสงในการรับภาพต่ำมาก เหมาะอย่างยิ่งที่จะใช้งานในด้านการรักษาความปลอดภัยเนื่องจากสามารถดูในเวลากลางคืนได้ดีกว่ากล้องสี เหมาะสำหรับโรงงานอุตสาหกรรม เช่น ในอาคาร, คลังสินค้า, โรงงาน, กระบวนการผลิต, พื้นที่อันตราย, เคานเตอร์เก็บเงิน, ลานจอดรถ, ปั๊มน้ำมัน หรือสถานที่ที่ใช้อุปกรณ์ดูแลรักษาความปลอดภัย

2.2 ลักษณะของสัญญาณภาพ

สัญญาณภาพโดยทั่วไปจะมีลักษณะเป็นคอมโพสิทวีดีโอ ที่ประกอบไปด้วยข้อมูล สัญญาณภาพ (Image Signal), สัญญาณซิงค์ (Sync Signal) และสัญญาณแบลงกิ้ง (Blanking Signal) โดยลักษณะของสัญญาณมีดังรูปที่ 2.1 และรูปที่ 2.2

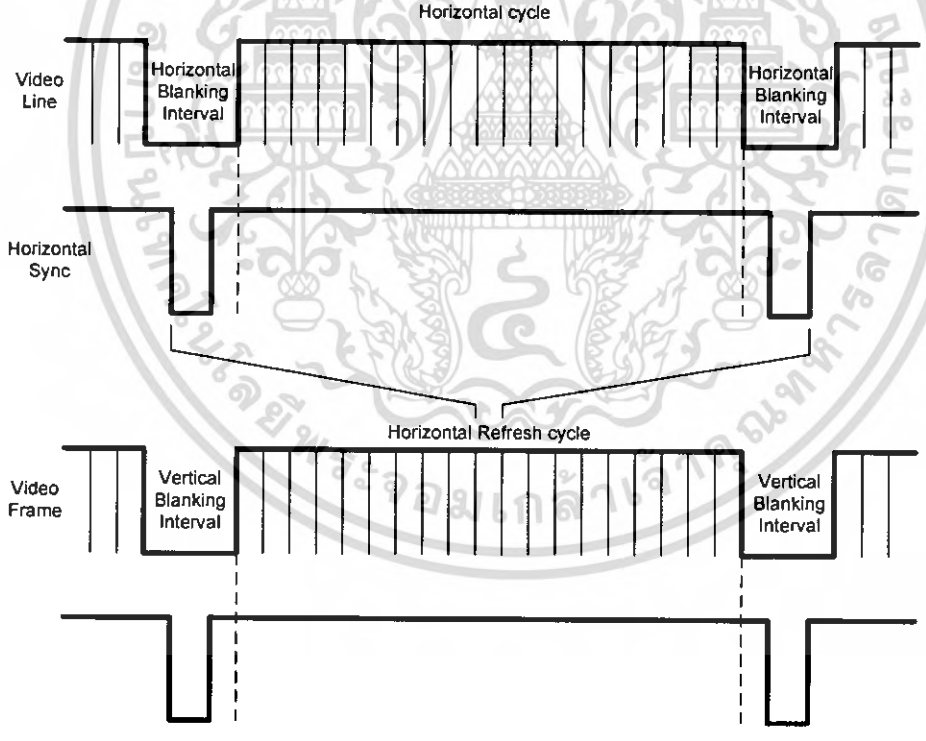


รูปที่ 2.1 ลักษณะของสัญญาณทางด้านแนวนอน



รูปที่ 2.2 ลักษณะของสัญญาณทางด้านแนวตั้ง

สัญญาณดังกล่าวจะถูกส่งไปยังมอนิเตอร์ ทำให้เกิดการสแกนที่หน้าจอมอนิเตอร์ซึ่งจะทำให้เกิดเป็นภาพขึ้นมา



รูปที่ 2.3 ความสัมพันธ์ระหว่างสัญญาณทางด้านแนวนอนและแนวตั้ง

2.2.1 องค์ประกอบของภาพ

ภาพนั้นมีองค์ประกอบมาจากจุดสีขาวและดำมากมาย มาเรียงกันประกอบขึ้นเป็นภาพ จุดเหล่านี้เองที่เรียกว่า เป็นองค์ประกอบของภาพ หรือ พิกเจอร์ อิลีเมนต์ หรือ พิกเซล ทำนองเดียวกัน ภาพที่ปรากฏทางจอภาพก็เอามาจากหลักการนี้ ภาพที่เกิดขึ้นบนจอภาพ หรือจอโทรทัศน์ประกอบด้วยเส้นขวางเล็กๆ ในแนวนอนเป็นจำนวนมากยิ่งภาพมีจำนวนเส้นมากเท่าไรรายละเอียดภาพก็จะยิ่งมากขึ้นเท่านั้น แต่ละเส้นนั้นมีทั้งส่วนที่ดำสนิท ส่วนที่จาง และส่วนที่สว่างร่วมกัน เส้นเหล่านี้เราได้มาจากการกวาดลำแสง(Scan) ความแตกต่างกันบนเส้นสแกนเหล่านี้เองที่จัดว่าเป็นองค์ประกอบของภาพ ซึ่งจากการทดสอบพบว่าภาพที่พอดูได้จะมีองค์ประกอบไม่ต่ำกว่า 200,000 พิกเซล

2.2.2 การสแกนภาพ

จุดที่เห็นสว่างในหลอดภาพของจอภาพ เกิดขึ้นเพราะ อิเล็กตรอนที่หลุดออกจากแคโทดและถูกดึงดูดให้วิ่งเป็นลำไปกระทบแอโนดหรือจอหลอดภาพ ซึ่งฉาบวัสดุเรืองแสงบางชนิดเอาไว้ การสแกนหลอดภาพมีสองวิธี คือ

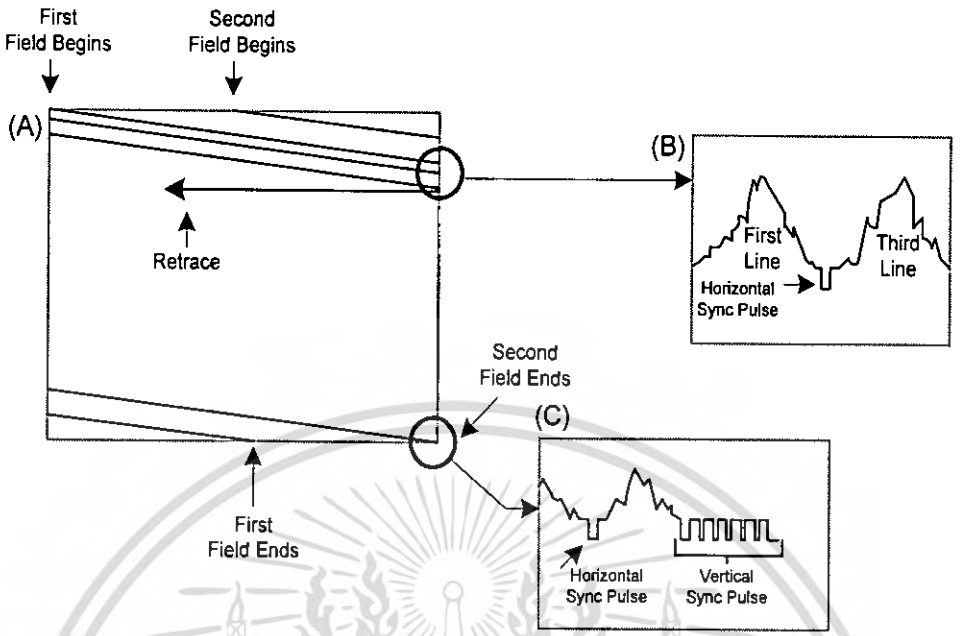
1. การสแกนแบบก้าวหน้า(Progressive scanning)
2. การสแกนแบบสลับเส้น(Interlaced scanning)

การที่จะให้การสแกนมีความต่อเนื่องขององค์ประกอบภาพจะต้องคำนึงถึงหลัก 3 ประการ คือ

1. ลำอิเล็กตรอนที่กวาดในแนวนอน (Horizontal scanning) ในแต่ละครั้งจะต้องครอบคลุมองค์ประกอบภาพทั้งหมดของเส้นนั้นๆ

2. ในแต่ละเส้นของการสแกนลำอิเล็กตรอน ลำแสงจะต้องกวาดไปทางด้านซ้ายเพื่อเริ่มต้นเส้นภาพทางแนวนอนลำดับต่อไป เวลาของการสะบัดกลับเราเรียกว่า "รีเทรซ" หรือ "ฟลายแบ็ค" ในกรณีดังกล่าวจะต้องไม่มีข้อมูลภาพใดๆ เพราะว่าทั้งกล้องถ่ายและหลอดภาพจะเกิดอาการแบลงค์เอาท์ ในขณะนั้น

3. ในขณะที่เส้นสแกนสะบัดกลับมาเพื่อเริ่มต้นทางซ้ายใหม่ ตำแหน่งทางแนวตั้งต้องต่ำกว่าตำแหน่งเดิมเพื่อทำให้การสแกนเส้นต่อไปไม่ทับกัน ทั้งนี้โดยการควบคุมของสัญญาณใน แนวตั้ง(Vertical scanning)



รูปที่ 2.4 (A) จุดเริ่มต้นสแกนครั้งที่ 1 และครั้งที่ 2
รูปที่ 2.5 (B) สัญญาณสแกนในแนวตั้ง บนลงล่าง
รูปที่ 2.6 (C) สัญญาณสแกนในแนวนอน ซ้ายไปขวา

การสแกนที่ใช้ในเครื่องรับโทรทัศน์ ถึงแม้เราจะพบว่าหากให้มีการเรียงภาพเกินกว่า 16ภาพต่อวินาทีแล้ว สายตาก็จะเห็นเป็นภาพต่อเนื่อง แต่แม้ว่าภาพที่เกิดขึ้นจะเกิด 24ภาพต่อวินาทีแล้วก็ตาม ก็ยังมีการกระพริบ(Flicker) เกิดขึ้นเนื่องจากว่าในขณะที่การสแกนเริ่มจากขอบบนลงมาด้านล่าง เมื่อเส้นสแกนลงมาถึงขอบด้านล่าง ในความรู้สึกของมนุษย์แสงทางด้านบนจะเริ่มมืดกว่าด้านล่าง เวลาที่ลำแสงทำการสแกนวกกลับไปด้านบน ด้านล่างก็เกิดปัญหาเช่นเดียวกัน จะรู้สึกว่าเกิดแสงกระพริบหรือวูบวาบขึ้น จึงต้องใช้การสแกนสลับเส้น โดยครั้งแรกจะสแกนฟิลด์คู่(Even line trace) เป็นการสแกนแบบเส้นเว้นเส้น นั่นหมายถึงว่า การได้ภาพหนึ่งภาพ หรือที่เรียกว่าภาพหนึ่งเฟรม ต้องใช้การสแกนถึง 2ครั้ง หรือ 2ฟิลด์

2.3 โมเดลสี

โมเดลแบบ RGB

โมเดลสีชนิดนี้ประกอบด้วยการรวมตัวกันของแม่สีหลักซึ่งได้แก่ แดง (R), เขียว (G) และ น้ำเงิน (B) ซึ่งค่าสีต่างๆในแถบสเปคตรัมของสีจะได้มาจากการผสมกันในอัตราส่วนที่แตกต่างกันของแม่สีทั้งสาม

โมเดลสี RGB นี้จะแสดงด้วยแกนของลูกบาศก์สามแกนในระนาบสามมิติ ซึ่งค่าสีแดง สีเขียว และสีน้ำเงิน จะอยู่ที่มุมทั้งสามของแต่ละแกนดังรูป ซึ่งจะเห็นว่าค่าสีดำจะอยู่ที่จุดกำเนิด (origin) สีขาวจะอยู่ที่

มุมตรงข้ามกับสีดำ ค่าของสีในช่วงระดับเทาจะอยู่ตามเส้นที่เชื่อมระหว่างค่าสีดำและค่าสีขาว จากรูปถ้าเป็นในระบบการแสดงสีแบบ 24 บิต (แบ่งออกเป็น 8 บิตต่อแม่สีหนึ่งสี) เช่น ค่าสีแดงจะถูกแทนด้วยค่า (255, 0, 0) เป็นต้น

โมเดลสี RGB ง่ายต่อการออกแบบและใช้งานในระบบคอมพิวเตอร์กราฟฟิก แต่ไม่เหมาะสำหรับประยุกต์ใช้งานอื่นๆ ค่าของ สีแดง สีเขียว และสีน้ำเงิน จะมีความสัมพันธ์กันอย่างมากซึ่งจะเป็นการยากที่จะนำไปประมวลผลเกี่ยวกับภาพ ดังนั้นจึงมีความจำเป็นที่จะต้องทำการแปลงภาพจากโมเดลสี RGB ให้อยู่ในรูปแบบ YCbCr เพื่อใช้ประโยชน์ในด้านการปรับความสว่างของภาพ

การแปลง RGB ให้อยู่ในรูป YCbCr

$$Y = 0.299*R + 0.587*G + 0.114*B$$

$$Cr = (R-Y)*0.713 + 128$$

$$Cb= (B-Y)*0.564 + 128$$

ปรับค่าความสว่างจากค่า Y

แปลงข้อมูลของภาพจาก YCbCr กลับไปเป็น RGB โดยใช้สมการ

$$R = Y + 1.403*(Cr - 128)$$

$$G = Y - 0.344*(Cr - 128) - 0.714*(Cb - 128)$$

$$B = Y + 1.773*(Cb - 128)$$

ในระบบดิจิตอลวีดีโอนั้น เรากล่าวถึงจำนวนบิตที่ใช้ในการสุ่มตัวอย่างสัญญาณแต่ละครั้งเป็นตัววัดความละเอียดของภาพที่ได้เป็นจำนวนบิตต่อพิกเซล (bit per pixel ; bpp)

เนื่องจากในการสุ่มตัวอย่างสัญญาณแต่ละครั้งนั้นจะได้เป็นหนึ่งพิกเซลโดยระบบ monochrome ที่มีคุณภาพสูงนั้นจะใช้ 8 bpp ซึ่งหมายถึงจำนวนระดับเทา(grey scale) ของภาพเท่ากับ 256 ระดับ แต่ในระบบการแสดงผลแบบสีนั้นเราต้องการหนึ่งช่องสัญญาณแบบ monochrome ต่อแม่สีแต่ละสี (แดง, เขียว และน้ำเงิน) ซึ่งจะต้องใช้จำนวนบิตทั้งหมดเป็น 24 บิต ซึ่งจะได้ค่าระดับสีที่เป็นไปได้คือ 16,777,216 สี ในระบบดิจิตอลวีดีโอบางระบบซึ่งจะใช้ 24 บิตต่อพิกเซล

2.4 เทคโนโลยีเครือข่าย

การพัฒนาอย่างรวดเร็วของเทคโนโลยีด้าน Internet Protocol (IP) ทำให้อินเทอร์เน็ตซึ่งเป็นเครือข่ายคอมพิวเตอร์ที่มีแหล่งข้อมูลที่ใหญ่ที่สุดในโลกปัจจุบัน มีความจำเป็น สำหรับประชาชนทุกคนที่สามารถเข้าค้นหาข้อมูลและข่าวสารต่างๆ ที่ต้องการได้อย่างรวดเร็ว และยังรวมถึงการใช้ติดต่อสื่อสารระหว่างบุคคล และแหล่งข้อมูลต่างๆ ได้ทั่วทุกมุมโลกตลอดเวลา

IP Network หรือ โครงข่ายไอ-พี เป็นโครงข่ายมาตรฐานที่นิยมใช้ในการติดต่อสื่อสาร มีความทันสมัย และความเร็วสูง สามารถใช้รับส่งสัญญาณได้ทั้งเสียง ข้อมูล ภาพวีดีโอ และภาพเคลื่อนไหว โดยอยู่ในรูปสัญญาณดิจิตอลที่เรียกว่า ไอ-พี แพ็กเกจ โครงข่ายนี้รองรับการใช้งานร่วมกับโครงข่ายโทรคมนาคม

ใหม่ ๆ และทันสมัยหลายแบบ ได้แก่ โครงข่ายใยแก้วความเร็วสูง แบบ SDH (Synchronous Digital Hierarchy), โครงข่ายดิจิตอลสวิทช์ที่รองรับหลายความเร็วแบบ ATM (Asynchronous Transfer Mode), โครงข่ายใยแก้วความเร็วสูงแบบ Optical Internet Protocol โดยส่งผ่านสัญญาณที่ระดับความเร็วสูงตั้งแต่ 2 Mbps - 2.5 Gbps และพร้อมพัฒนาใช้งานร่วมกับอินเทอร์เน็ตแบบไร้สาย ในโครงข่าย โทรศัพท์เคลื่อนที่ยุคที่ 3 (Third Generation Digital Mobile Communications Systems)

ความหมายของระบบเครือข่าย

ระบบเครือข่ายคอมพิวเตอร์ (Computer Network) หรือจะเรียกสั้นๆว่า ระบบเครือข่าย (Network) ประกอบไปด้วยเครื่องคอมพิวเตอร์หลายๆ เครื่องที่สามารถติดต่อกันเพื่อแลกเปลี่ยนข้อมูลกันได้ การติดต่อจะผ่านทางช่องการสื่อสารต่าง ๆ เช่น สายโทรศัพท์ สายไฟฟ้า หรือผ่านทางสื่อแบบอื่นๆ ได้แก่ โมเด็ม ไมโครเวฟ สัญญาณอินฟราเรด เป็นต้น ซึ่งโดยทั่วไปจะหมายถึง การที่เรานำเครื่องคอมพิวเตอร์อย่างน้อย 2 เครื่องมาเชื่อมต่อกัน วัตถุประสงค์ที่ต้องต่อกันนี้ มักเกิดจากความต้องการที่จะใช้ทรัพยากรของระบบร่วมกัน เช่น ใช้เนื้อที่เก็บข้อมูลในดิสก์ร่วมกัน ใช้งานเครื่องพิมพ์เลเซอร์ที่มีอยู่เครื่องเดียวร่วมกัน ต้องการส่งข้อมูลให้กับบุคคลอื่นในระบบไปใช้งาน หรือต้องการติดต่อสื่อสารระหว่างกัน เป็นต้น ฉะนั้น ระบบเครือข่าย Network คือ ระบบที่นำเอาเครื่องคอมพิวเตอร์ส่วนบุคคล (PC หรือ Personal Computer) แต่ละเครื่องมาต่อเชื่อมกันด้วยกลวิธีทางระบบคอมพิวเตอร์นั่นเอง

การแบ่งชนิดของระบบเครือข่าย สามารถดูได้จากลักษณะการติดตั้งใช้งานทางภูมิศาสตร์ จึงสามารถแบ่งระบบเครือข่ายได้เป็น 3 แบบ ด้วยกัน คือ

1. ระบบเครือข่ายระดับท้องถิ่น (Local Area Network หรือ LAN) เป็นระบบเครือข่ายที่ใช้งานอยู่ในบริเวณไม่กว้างนัก อาจใช้อยู่ภายในอาคารเดียวกันหรืออาคารที่อยู่ใกล้กัน เช่น เช่น ภายในมหาวิทยาลัย อาคารสำนักงาน คลังสินค้า หรือโรงงาน เป็นต้น การส่งข้อมูลสามารถทำได้ด้วยความเร็วสูง และมีข้อผิดพลาดน้อย ระบบเครือข่ายระดับท้องถิ่นนี้จึงถูกออกแบบมาให้ช่วยลดต้นทุนและเพื่อเพิ่มประสิทธิภาพในการทำงาน และใช้งานอุปกรณ์ต่าง ๆ ร่วมกัน

2. ระบบเครือข่ายระดับประเทศ (Wide Area Network หรือ WAN) เป็นระบบเครือข่ายที่ติดตั้งใช้งานอยู่ในบริเวณกว้าง เช่น ระบบเครือข่ายที่ติดตั้งใช้งานทั่วโลก โดยปกติมีอัตราการส่ง ข้อมูลที่ต่ำและมีโอกาสเกิดข้อผิดพลาด การส่งข้อมูลอาจใช้อุปกรณ์ในการสื่อสาร เช่น โมเด็ม มาช่วย

3. ระบบเครือข่ายระดับเมือง (Metropolitan Area Network หรือ MAN) เป็นระบบเครือข่ายที่มีขนาดอยู่ระหว่าง LAN และ WAN คือ เป็นระบบเครือข่ายที่ใช้ภายในเมืองหรือจังหวัดเท่านั้น LAN และ WAN คือ เป็นระบบเครือข่ายที่ใช้ภายในเมืองหรือจังหวัดเท่านั้น

2.5 ระบบอ้างอิงโมเดล OSI

องค์กรมาตรฐานสากลหรือ ISO (International Organization for Standardization) ได้มีการศึกษาและหาแนวทางในการกำหนดมาตรฐานของเครือข่าย ซึ่งจะทำให้การเชื่อมต่อระหว่างเครือข่ายสามารถทำได้ ISO ค้นพบว่า ระบบเครือข่ายจะมีกิจกรรมพื้นฐาน ต่าง ๆ เช่น การรับส่งข้อมูล การเข้าใช้งานในเครือข่าย การสั่งพิมพ์ที่เครื่องพิมพ์ในเครือข่าย เป็นต้น ดังนั้น ISO ได้จัดแบ่งกิจกรรมเหล่านั้นออกเป็นงานย่อย และกำหนดเป็นโมเดลแบ่งเป็นชั้น ๆ ตามลำดับ เรียกว่า มาตรฐาน Open System Interconnection หรือ OSI ด้วยวิธีการแบ่งกิจกรรมที่ซับซ้อนในเครือข่ายออกเป็นงานย่อยๆ เพื่อจะช่วยทำให้การออกแบบและใช้งานเครือข่าย รวมถึงการเชื่อมโยงกันเป็นไปได้ด้วยความสะดวก และมีวิธีการทำงานอยู่ในกรอบเดียว กัน โมเดล OSI นี้เป็นต้นแบบแนวคิดในการสร้างเครือข่าย

ระบบย่อยของการสื่อสารข้อมูล เป็นส่วนที่มีความซับซ้อนมาก ไม่ว่าจะเป็นทางด้านHardware หรือ Software ในสมัยแรกนั้น ๆ การพยายามสร้างโปรแกรม เพื่อการติดต่อสื่อสารนั้น จะสร้างโปรแกรมที่ค่อนข้างซับซ้อน ด้วยการมีส่วนประกอบหลายส่วนใน 1 โปรแกรม โปรแกรมประเภทนี้ โดยมากจะเขียนด้วยภาษาแอสแซมบลี ซึ่งผลมันก็คือ โปรแกรมเหล่านี้จะยุ่งยากในการทดสอบหรือปรับปรุงแก้ไข และเพื่อจะแก้ไขปัญหาที่เกิดขึ้นนี้ ISO จึงได้นำเอาวิธีการของระบบลำดับชั้น มาใช้ในเรื่องของ Reference Model ระบบย่อยการสื่อสาร ข้อมูลภายในเครื่องคอมพิวเตอร์จะถูกแบ่งออกเป็นลำดับชั้น หลาย ๆ ชั้น ซึ่งแต่ละขั้นก็จะทำหน้าที่ซึ่งได้กำหนดไว้โดยเฉพาะ

ISO Reference Model

ISO Reference Model ถูกออกแบบมาให้ประกอบด้วย 7 ลำดับชั้น ได้แก่

1.Physical Layer

จะให้บริการเกี่ยวกับวิธีการส่งข้อมูลเป็นบิต ระหว่างอุปกรณ์ 2 ชนิด ได้แก่ เครื่องมือและอุปกรณ์ทางฝ่ายผู้ใช้ และอุปกรณ์ของเครือข่ายโดยข้อมูลต่าง ๆ จะถูกส่งให้กับ Data Link Layer นอกจากนั้นยังรับผิดชอบดูแลในรายละเอียดการส่งข้อมูลในด้านฮาร์ดแวร์จริง เช่น การควบคุม Network Interface Card การส่งสัญญาณผ่านสายสัญญาณแบบต่าง ๆ การเชื่อมต่อเข้าเครือข่ายแบบต่าง ๆ โดย Physical Layer จะจัดสัญญาณทางไฟฟ้า สัญญาณ เสียง หรือสัญญาณแสงที่จำเป็นในการสื่อสารโดยตรง

2.Data Link Layer

จะจัดเตรียมข้อมูลเกี่ยวกับการเชื่อมต่อให้กับ Network Layer และจะทำหน้าที่เรียกใช้หรือกำหนดช่องทางในการส่งข้อมูลที่ถูกต้อง เช่น Ethernet , Token Ring หรือ FDDI เป็นต้น รวมถึงการควบคุมลำดับและอัตราการรับส่งข้อมูลและสถานที่ ที่จะส่งข้อมูลไป ทั้งนี้ Data Link Layer เป็นชั้นแรกที่จัดการแปลงข้อมูลจากบิตให้อยู่ในรูปของ Packet โดยจะมีการเพิ่มข้อมูลเพื่อตรวจสอบความ ถูกต้องในกรณีส่งข้อมูลออกไป หรือ

ในกรณีอ่านข้อมูลเข้ามาก็จะตรวจสอบส่วน Checksum เพื่อดูว่าข้อมูลที่ได้รับมาถูกต้องครบถ้วน และถ้าได้รับ Packet ข้อมูลที่ไม่ถูกต้องก็จะไม่เอาข้อมูลนั้นไปใช้งานต่อ และบอกไปยังต้นทางให้ส่งมาใหม่

ตัวอย่างเช่น คำสั่งในการตรวจสอบข้อผิดพลาด หรือมีข้อผิดพลาดแล้วต้องส่งข้อมูลใหม่ ก็จะมีคำสั่งในการส่งข้อมูลซ้ำ โดยปกติแล้วจะมีการให้บริการ ใน 2 รูปแบบ ได้แก่

- Connection Less ถ้าหากมีการตรวจสอบและพบข้อผิดพลาดก็จะตัดการสื่อสารทันที
- Connection Oriented ให้บริการในลักษณะพยายามไม่ให้เกิดข้อผิดพลาดใด ๆ ทั้งสิ้น

3. Network Layer

ทำหน้าที่ควบคุมวิธีการส่งผ่านข้อมูลระหว่างเครือข่ายให้ถูกต้อง และเป็นไปตามเส้นทางที่กำหนด ทำหน้าที่ในการเชื่อมต่อเส้นทางการสื่อสารระหว่าง Transport Layer ของเครือข่ายต้นทางและปลายทาง โดยจะจัดคำสั่งการทำงานเกี่ยวกับการค้นหาที่หมายปลายทางและควบคุมการไหลของข้อมูลในการเชื่อมต่อระหว่างเครื่องคอมพิวเตอร์กับเครือข่าย และในกรณีติดต่อระหว่างเครือข่าย Network Laycr ก็จะมีการจัดเตรียมคำสั่งการทำงาน เพื่อทำให้การติดต่อเป็นไปอย่างราบรื่นสมบูรณ์ในปัจจุบันเมื่อมีการใช้เครือข่ายมากขึ้น ขนาดใหญ่ขึ้น ซับซ้อนมากขึ้น จะมีการจัดการแบ่งเครือข่ายนั้นให้เป็นส่วนย่อยหรือ Network Segment หรือ เรียกว่า Subnetwork โดยใช้อุปกรณ์ Bridge หรือ Router ทั้งนี้ Network Layer จะจัดส่งผ่าน Packet ข้อมูล คือ ข้อมูลที่ถูกจัดให้อยู่ใน รูปที่กำหนดไว้แล้วนั่นเอง ผ่านอุปกรณ์ต่างๆ ไปยังเครือข่ายย่อยให้อย่างถูกต้องตามที่ต้องการ นอกจากนี้ Network Layer จะจัดการดูแลเส้นทางในการส่งข้อมูล (Routing Table) และกั้นหรือกรอง Packet ข้อมูลที่ส่งไปยังที่หมายภายในเครือข่ายย่อยเดียวกัน ไม่ให้ข้ามไปยังเครือข่ายย่อยอื่น ซึ่งจะช่วยลดปริมาณข้อมูลที่วิ่งบนเครือข่ายได้ส่วนหนึ่ง โปรโตคอล IP,TCP/IP และ IPX เป็นโปรโตคอล ที่ทำงานอยู่ใน Layer นี้

4.Transport Layer

ทำหน้าที่ควบคุมปริมาณและรายละเอียดวิธีการรับส่งข้อมูลให้เป็นไปตามกำหนดที่ได้ดังตั้งไว้ และจัดการให้การเชื่อมโยงเครือข่ายเป็นไปอย่างราบรื่น Transport Laycr เป็นชั้นสุดท้ายที่จัดการเรื่องของเส้นทางในการส่งข้อมูล และจัดการตรวจสอบความผิดพลาดของข้อมูล ซึ่งส่วนของ TCP (Transmission Control Protocol) ในโปรโตคอล TCP/IP แบบที่ใช้ในอินเตอร์เนต จะทำงาน ที่ระดับชั้นนี้

Transport Layer จะทำหน้าที่เป็นตัวเชื่อมระหว่าง 2 Layer คือ Application - Oriented Layer ซึ่งอยู่เหนือกว่า กับ Network - Dependent Protocol Layer ซึ่งอยู่ต่ำกว่า และ มีหน้าที่ในการเตรียมข้อความต่าง ๆ ในการสื่อสารแบบ Session Layer

5. Session Layer

เป็นชั้นที่จัดการในเรื่องของการสร้างการติดต่อแต่ละครั้ง หรือ Session ให้ระบุคอมพิวเตอร์ทั้งสองฝ่าย กล่าวง่าย ๆ คือ จะให้บริการแก่โปรแกรมสื่อสาร ได้จัดการรวบรวมข้อมูลต่าง ๆ ที่จะใช้ในการ

ติดต่อสื่อสาร โดยทำหน้าที่ตั้งแต่เริ่มการติดต่อ ดูแลให้การส่งผ่านข้อมูลในการติดต่อครั้งนั้น ๆ เป็นไปได้โดยไม่มีปัญหา จนถึงการเลิกติดต่อเมื่อเสร็จงาน ซึ่ง Session Layer นี้จะรับผิดชอบเกี่ยวกับการจัดสรรช่องการสื่อสารระหว่าง Application Layer ทั้งสองฝ่ายให้สามารถสื่อสารกันได้จนเสร็จสมบูรณ์ โดยต้องอาศัยความช่วยเหลือจาก Presentation Layer นอกจากนั้นยังให้บริการด้านต่าง ๆ คือ

- ให้บริการจัดเงื่อนไขโต้ตอบข้อมูลซึ่งจะให้บริการทั้งในแบบส่งข้อมูลไม่พร้อมกัน (Half-Duplex) และแบบพร้อมกัน (Full-Duplex)
- การสื่อสารในทิศทางเดียวกัน ในการสื่อสารผ่านระบบเครือข่ายระยะไกล ถ้าหากเกิดข้อผิดพลาดนั้น ระหว่างการสื่อสารในจุดใดจุดหนึ่ง Session Layer จะถูกอนุญาตให้ผู้ใช้เลือกที่จะทำการรับ-ส่ง ข้อมูลใหม่อีกครั้งในเวลาใดก็ได้

การรายงานเกี่ยวกับข้อผิดพลาดถ้าในระหว่างการสื่อสารเกิดข้อผิดพลาดที่ไม่สามารถแก้ไขได้ Session Layer จะทำการส่งสัญญาณเพื่อแจ้งให้ Application Layer รู้ถึงข้อผิดพลาดนั้น

6. Presentation Layer

เป็นชั้นที่มีการกำหนดหน้าที่ไม่ชัดเจนนัก และมีการนำไปใช้ไม่มาก หน้าที่หลัก คือ เป็นส่วนที่จัดรูปแบบและนำเสนอข้อมูลระหว่างการสื่อสาร ให้เป็นไปตามที่ต้องการ โดยมีการกำหนดรูปแบบการส่งข้อมูลสำหรับใช้ในการแลกเปลี่ยน รูปแบบที่จะมีการกำหนดไว้ใน 2 ลักษณะ ทั้งนี้ยังรวมไปถึงการจัดแปลงข้อมูลในรูปมาตรฐาน ASCII หรือ EBCDIC, การลดขนาดข้อมูล (Data Compression) การเข้ารหัสหรือถอดรหัส ข้อมูลเพื่อความปลอดภัยในการสื่อสารแต่ส่วนใหญ่แล้วแอปพลิเคชั่นจะเป็นตัวจัดการแทนได้

ตัวอย่างของ Presentation Layer ที่เข้าใจง่าย ๆ เช่น การพูดโทรศัพท์ระหว่างบุคคลที่พูดภาษาฝรั่งเศสและบุคคลที่พูดภาษาสเปน บุคคลทั้งสอง ใช้ล่ามในการแปรภาษา และภาษากลางที่ใช้ก็คือภาษาอังกฤษ ล่ามแต่ละฝ่ายก็จะแปรภาษาทางฝ่ายของตนเองออกเป็นภาษาอังกฤษ แล้วฝ่ายที่รับ ก็จะแปรภาษาอังกฤษนั้น กลับเป็นภาษาของตนเองอีกต่อหนึ่ง บุคคลผู้พูดทางโทรศัพท์ก็เปรียบเสมือนกับโปรแกรม 2 โปรแกรม ที่ต้องการสื่อสารกัน ตัวล่ามก็เปรียบเสมือนกับ Presentation Layer ในเครื่องคอมพิวเตอร์ทั้ง 2 ฝ่าย ภาษาฝรั่งเศสและภาษาสเปน เปรียบเสมือนกับ Abstract Data Syntax ส่วนภาษาอังกฤษเปรียบได้กับ Transfer or Concrete syntax ทั้งนี้ต้องนึกเสมอว่า ในการสื่อสารข้อมูลนั้น จะต้องมีการใช้ภาษากลางอันเป็นที่เข้าใจกันโดยทั่วไปภาษาหนึ่งเสมอ นอกจากนั้นตัวกลางในการแปรภาษา จากต้นทางและปลายทางให้เป็นภาษากลาง ก็ไม่จำเป็นจะต้องเข้าใจในความหมายของข้อมูลที่สื่อสารนั้นเสมอไป หน้าที่อีกประการหนึ่งของ Presentation Layer ก็คือ เรื่องความปลอดภัยของข้อมูลโดยมีการเข้ารหัสข้อมูล ก่อนส่งออกไป และเมื่ออีกฝ่ายหนึ่ง ได้รับข้อมูลแล้ว ก่อนนำมาใช้ก็จะมีการถอดรหัสตามรูปแบบตามที่ได้มีการกำหนดไว้ล่วงหน้าทั้งสองฝ่าย แม้ว่าวิธีการเข้ารหัสข้อมูลนี้จะมิได้มี การกำหนดไว้ในมาตรฐานใด ๆ เลยก็ตามแต่ก็ได้มีผู้นำมาใช้กันอย่างแพร่หลาย

7. Application Layer

เป็นชั้นบนสุดของโมเดลจะเป็นส่วนที่ทำการติดต่อระหว่างแอปพลิเคชั่นของเครือข่ายกับผู้ใช้เป็นไปตามที่ต้องการ ตัวอย่างแอปพลิเคชั่น ของเครือข่ายก็เช่น ระบบ จดหมายอิเล็กทรอนิกส์ การโอนถ่ายแฟ้มข้อมูล (File Transfer) การขอเข้าใช้ระบบคอมพิวเตอร์ในเครือข่าย (Host Terminal) การจัดแฟ้มข้อมูล ในลักษณะต่างๆ เป็นต้น นอกจากนั้นยังรวมถึงการบริการทางด้านการแลกเปลี่ยนเอกสารข้อความต่าง ๆ Application - Layer จะทำหน้าที่จัดการเรื่องต่าง ๆ ของเครือข่ายตามที่ผู้ใช้ต้องการนั่นเอง โดยจะอยู่ในระดับบนที่ใกล้ชิดกับผู้ใช้ที่สุด การเข้าไปใช้บริการระดับ Application Layer โดยปกติจะใช้โดยการพิมพ์คำสั่งต่าง ๆ ผ่านทางระบบโปรแกรมควบคุมเครื่อง (Operating System หรือ OS) โดย OS จะถือว่าอุปกรณ์ หรือโปรแกรมต่าง ๆ ในการสื่อสารนั่นเอง การบริการของลำดับชั้นนี้ จะแสดงให้ผู้ใช้ได้เข้าใจในทันที โดยไม่ว่าการสื่อสารจะประสบความสำเร็จหรือไม่ และหากมีข้อผิดพลาดประการใด ผู้ใช้ก็ สามารถทราบได้จากข้อความที่แสดงออกมา นอกเหนือจากบริการด้านต่าง ๆ ที่กล่าวข้างต้นแล้ว ยังมีบริการอื่น ๆ ที่ Application Layer จัดให้ได้ดังนี้ คือ

- การระบุระบบคอมพิวเตอร์ปลายทาง โดยอาจจะใช้วิธีการเรียกหรือเรียกตามที่อยู่ก็ได้
- กำหนดว่าเครื่องปลายทางที่ต้องการติดต่อนั้น อยู่ในสถานะที่พร้อมสำหรับการสื่อสาร
- กำหนดอำนาจหน้าที่ ความสำคัญ ในการติดต่อสื่อสาร
- การร่วมตกลงในระบบการใส่รหัสเพื่อความปลอดภัยของข้อมูล
- การรับรองเครื่องคอมพิวเตอร์ที่ต้องการสื่อสารด้วยว่า อยู่ในสถานะที่ต้องการจะสื่อสารได้หรือไม่
- กำหนดและเลือกข้อมูลประเภทต่าง ๆ ที่จำเป็นต้องใช้เมื่อมีการสื่อสาร
- การร่วมตกลงเกี่ยวกับความรับผิดชอบ กรณีการกู้ข้อมูลที่เกิดความเสียหาย
- กำหนดข้อจำกัดต่าง ๆ เกี่ยวกับรูปแบบของข้อมูลที่จะใช้ส่ง เช่น รูปแบบตัวอักษรที่จะใช้ หรือ โครงสร้างของข้อมูล

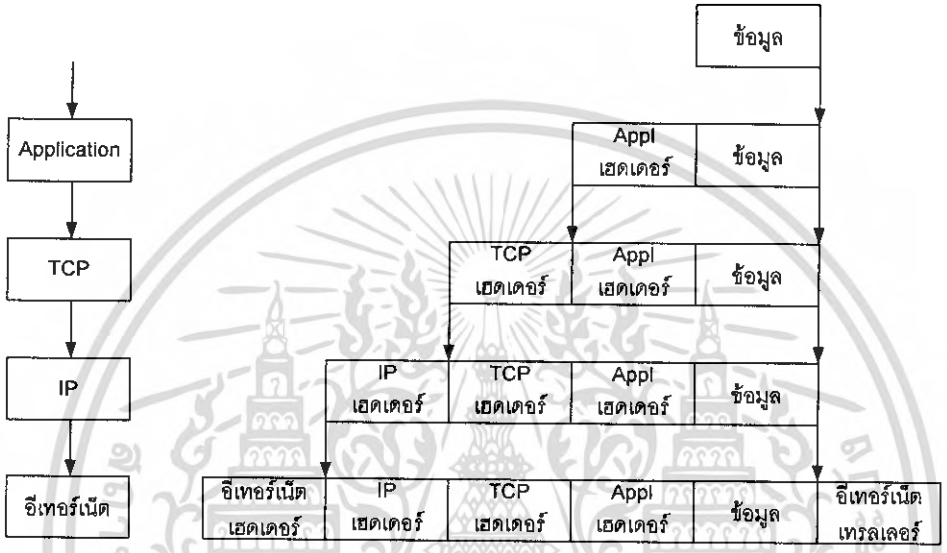
2.6 การส่งถ่ายข้อมูลระหว่างชั้น

โปรโตคอลในแต่ละชั้นล้วนมีหน้าที่เกี่ยวข้องในการส่งผ่านข้อมูลจากสถานีต้นทางไปยังสถานีปลายทาง ข้อมูลจะถูกส่ง ผ่านจากโปรโตคอลระดับบนสุดจากสถานีต้นทางไปยังระดับล่างจนกระทั่งข้อมูลถูกแปลงให้อยู่ในรูปของสัญญาณไฟฟ้าแล้วเดินทางผ่านสถานีเครือข่ายไปยังสถานีปลายทาง โปรโตคอลระดับล่างสุดที่สถานีปลายทางจะรับสัญญาณและส่งผ่ายขึ้นไปยังโปรโตคอลระดับบนต่อไป

เมื่อข้อมูลผ่านแต่ละระดับชั้น โปรโตคอลในชั้นนั้นจะผนวกข่าวสารกำกับการทำงานประจำโปร - โตคอลซึ่งเรียกว่าโปรโตคอลเฮดเดอร์ เข้ากับข้อมูลเฮดเดอร์และตัวข้อมูลจากระดับบนจะถูกส่งผ่านไปยังระดับล่าง โปรโตคอลระดับล่างจะมองเฮดเดอร์และตัวข้อมูลรวมเป็นเสมือนข้อมูลและเพิ่มเฮดเดอร์ประจำชั้น

เข้าไปข้อมูลเดิมจึงมีแค่เฮดเดอร์หุ้มเป็นชั้นๆ กระบวนการนี้เรียกว่า การเอ็นแคปซูเลต ตัวอย่างในรูปที่ 2.7 การแสดงเอ็นแคปซูเลตแพ็กเกจทีซีพี/ไอพีในอีเทอร์เน็ต

เมื่อสถานีปลายทางได้รับแพ็กเกจก็จะดำเนินการส่งไปตามลำดับชั้น โปรโตคอลประจำชั้นจะทำการถอดเฮดเดอร์ออกและส่งส่วนที่เหลือไปยังชั้นถัดไป เฮดเดอร์จะถูกถอดออกเหลือเฉพาะข้อมูลเมื่อถึงชั้นบนสุด กระบวนการนี้เรียกว่าการดีแคปซูเลต



รูปที่ 2.7 การห่อหุ้มข้อมูลตามลำดับโปรโตคอลแสตค

2.7 เลขหมายอินเทอร์เน็ต (IP Address)

หมายเลข IP เป็นหมายเลขขนาด 32bit ที่ใช้ระบุอุปกรณ์สื่อสารใน อินเทอร์เน็ตโดยเลขนี้จะไม่ซ้ำกันเลยหมายเลข IP จะใช้ระบุอุปกรณ์สื่อสาร (Network Interface) ของเครื่องคอมพิวเตอร์ เครื่องคอมพิวเตอร์เครื่องหนึ่งอาจมี อุปกรณ์สื่อสารหลายตัวก็ได้ เช่น ที่ทำหน้าที่เป็น Gateway อาจมี LAN card อยู่ 2 ถึง 3 card หมายเลข IP มักนิยมเขียนในรูปเลขฐาน 10 สี่กลุ่มโดยใช้จุดคั่นแต่ละช่วงมีค่าตั้งแต่ 0 ถึง 255 เช่น 161.246.18.44 เป็นต้นแต่เวลาเครื่องคอมพิวเตอร์นำเลขเหล่านี้ไปประมวลผลมันจะมองเป็นเลขฐานสอง 32 bit ต่อเนื่องกัน เลข IP นี้แบ่งออกเป็นสองส่วนคือส่วนที่ชี้ Network กับ ส่วนที่ชี้เครื่อง แต่ละส่วนจะใช้กี่ bit นั้นขึ้นอยู่กับ class ของเลข IP นั้นๆ ซึ่ง class ของเลข IP นั้นจะมีอยู่ 5 class ด้วยกันคือ

Class A: 0NNN.CCC.CCC.CCC (NNN ชุดแรก ตัวเลขอยู่ระหว่าง 1-126) เครือข่าย Class A สามารถแจกจ่าย IP Address ได้มากที่สุดถึง 16 ล้านหมายเลข

Class B: 10NNN.NNN.CCC.CCC (NNN ชุดแรก ตัวเลขอยู่ระหว่าง 128-191) เครือข่าย Class B สามารถแจกจ่าย IP Address ได้มากเป็นอันดับสอง คือ 65,000 หมายเลข

Class C: 110NNN.NNN.NNN.CCC (NNN ชุดแรก ตัวเลขอยู่ระหว่าง 192-233) เครือข่าย Class C สามารถแจกจ่าย IP Address ได้น้อยที่สุด คือ 256 หมายเลข

NNN หมายถึง Network Address ,CCC หมายถึง Computer Address(host part)

คลาส D และ คลาส E ไม่มีการจัดแบ่งเลขเครือข่ายและเลขโฮสต์โดยในคลาส D 3 บิตแรกเป็น 111 จึงมี แอดเดรสตั้งแต่ 224.0.0.0 ถึง 239.255.255.255 แอดเดรสในคลาสนี้เรียกว่ามัลติคาสต์แอดเดรส สำหรับคลาส E มีแอดเดรสจาก 240.0.0.0 ถึง 254.255.255.255 ซึ่งสำรองไว้เพื่อความจำเป็นเฉพาะงานในอนาคต

เนื่องจากเครือข่ายก็อาจจำเป็นต้องใช้ เลขหมายอินเทอร์เน็ตดังนั้น จึงจำเป็นต้องมีการจำกัดบางหมายเลขเพื่อใช้เป็นการภายใน ได้แก่

Class A ตั้งแต่ 10.0.0.0-10.255.255.255

Class B ตั้งแต่ 172.16.0.0 ถึง 172.31.255.255

Class C ตั้งแต่ 192.168.0.0 ถึง 192.168.255.255

2.8 การแบ่ง Subnet

เราสามารถแบ่งหนึ่ง เน็ตเวิร์ค ของเราออกเป็นโครงข่ายย่อยๆโดยเรามักจะทำเมื่อ IP ของเราสามารถอ้างอิงเครื่องภายในหนึ่งเน็ตเวิร์คได้มากกว่าความต้องการจริงๆ ของเรามากๆ

วิธีการแบ่ง Subnet นั้นเราจะอาศัยความรู้จากเรื่องของ เลขหมายอินเทอร์เน็ตที่ว่าเลข IP แบ่งออกเป็นสองส่วนคือ ส่วนชี้เน็ตเวิร์คกับส่วนชี้เครื่อง ในเมื่อจำนวนบิต ที่เราใช้ในการชี้เครื่องมากเกินไป เราก็ทำการลดจำนวนโดยนำไปเพิ่มให้กับส่วนใช้ชี้เน็ตเวิร์คทั้งหมดที่กล่าวมาจะสามารถทำได้โดยการใช้ เน็ตเวิร์ค ซึ่งเป็นเลข 32bit เหมือนเลขหมายอินเทอร์เน็ตมีหน้าที่ระบุขอบเขตของเลข IP ว่าบิตใดใช้ชี้เน็ตเวิร์คและ บิต ใดใช้ชี้เครื่อง ปกติ เน็ตเวิร์ค Class A,B,C จะมี เน็ตมาร์ค ที่ตายตัวอยู่แล้วคือ 255.0.0.0 , 255.255.0.0, และ 255.255.255.0ตามลำดับวิธีการใช้ เน็ตมาร์คนั้นเราจะนำมา And กับ IP Address ผลที่ได้จากการ And คือหมายเลข network เช่น 161.246.18.44 จะมี เน็ตมาร์ค มาตรฐานเป็น 255.255.0.0 (เพราะเป็น IP Class B) เมื่อนำมา And กันจะได้ 161.246.0.0 ซึ่งก็คือหมายเลข เน็ตเวิร์ค นั้นเอง bit ที่ถูกตัดทิ้งไป (18.44) ก็คือหมายเลขเครื่อง

เราสามารถทำการเปลี่ยนแปลง เน็ตมาร์ค มาตรฐานได้ โดยการเพิ่มจำนวน bit ที่ใช้ชี้ เน็ตเวิร์ค นั้นคือเราจะเพิ่มจำนวน bit ที่เป็น 1 ของ เน็ตมาร์ค เช่นเราอาจเพิ่ม bit ที่ใช้ในการชี้เน็ตเวิร์คของ IP Class B ขึ้นอีก 8 bit โดยดึงมาจาก bit ที่เคยใช้ในการชี้เครื่อง ในกรณีนี้เราจะได้ เน็ตมาร์คเป็น 255.255.255.0 ทำให้เมื่อนำไป And กับ IP 161.246.18.44จะได้ 161.246.12.18.0 เป็นหมายเลข เน็ตเวิร์ค จะเห็นว่าจะได้ เน็ตเวิร์ค เพิ่มขึ้นมาเป็น 256 เน็ตเวิร์ค และ แต่ละ เน็ตเวิร์ค มีหมายเลขเครื่อง2 54 เครื่อง

2.9 Visual C++ programming

Visual C++ ได้รับการพัฒนาขึ้นมาจาก Microsoft C/C++ ให้เป็น ID ที่ทำงานบนระบบปฏิบัติการวินโดวส์ได้อย่างเต็มที่ รองรับการพัฒนาโปรแกรมบนวินโดวส์โดยมี MFC (Microsoft Foundation Class) เป็นไลบรารี่ที่ช่วยอำนวยความสะดวกในการพัฒนาโปรแกรมบนวินโดวส์

ในยุคที่ MFC ยังไม่ถือกำเนิด เมื่อจะทำการพัฒนาโปรแกรมบนวินโดวส์เราจะต้องใช้ SDK (Software Development Kit) และคอมไพเลอร์ภาษา C เช่น Microsoft C++ , Borland C++ ช่วยในการเขียนโปรแกรม โค้ดโปรแกรมที่เขียนขึ้นด้วย SDK นี้ค่อนข้างซับซ้อนและสร้างความลำบากในการศึกษาทำความเข้าใจ จึงได้มีการสร้างคลาสขึ้นมาชุดหนึ่ง เขียนขึ้นโดยใช้โครงสร้างของ OOP (Object Oriented programming) ด้วยภาษา C++ ชื่อว่า MFC หรือ Microsoft Foundation Class ใช้สำหรับอำนวยความสะดวกในการเขียนโปรแกรมบนวินโดวส์ โดยเฉพาะโค้ดโปรแกรมที่เขียนขึ้นด้วย MFC นั้นจะมีขนาดเล็กและไม่ซับซ้อน ทำให้การพัฒนาโปรแกรมบนวินโดวส์สามารถทำได้ง่ายกว่าการใช้ SDK

2.9.1 OOP (Object Oriented Programming)

OOP หรือ Object Oriented Programming เป็นวิธีการเขียน โดยอาศัยแนวคิดของวัตถุชิ้นหนึ่ง ลักษณะการเขียนโปรแกรมจะเป็นแบบโครงสร้าง (Structure) การเขียนโปรแกรมแบบ OOP มีความสามารถในการปกป้องข้อมูล และการสืบทอดคุณสมบัติ ซึ่งทำให้แนวโน้มของ OOP ได้รับการยอมรับ และพัฒนามาใช้ในระบบต่างๆมากมาย เช่น ระบบปฏิบัติการวินโดวส์ เป็นต้น

2.9.2 แนวคิดของ OOP

แนวคิดของ OOP คือ "ธรรมชาติของวัตถุ" หมายความว่า OOP จะมองสิ่งแต่ละสิ่งถือเป็น "วัตถุชิ้นหนึ่ง" ซึ่งจะมีสีแดงเขียว หรือจะยาวหรือสั้น มันก็คือวัตถุชิ้นหนึ่งเหมือนกัน ซึ่งวัตถุแต่ละสิ่งนั้น ย่อมมีคุณสมบัติที่แตกต่างกัน แต่อาจมีบางอย่างที่เหมือนกันบ้าง และเราก็สามารถกำหนดประเภทหรือคลาสให้กับวัตถุเหล่านั้นได้ เช่น วัตถุสีแดงก็สามารถรวมอยู่ในกลุ่มเดียวกัน หรือวัตถุที่มีขนาดยาวก็มารวมอยู่ในกลุ่มเดียวกัน เป็นต้น

นอกจากนี้แล้ว เมื่อ OOP มองทุกสิ่งถือเป็นวัตถุชิ้นหนึ่งแล้ว ยังคงคิดต่อไปว่า "วัตถุแต่ละอย่างนั้น ต่างก็จะมีลักษณะและวิธีการใช้งานเป็นตัวของมันเอง" ประโยคนี้มี ความหมายว่า วัตถุแต่ละชนิดหรือแต่ละชิ้น ต่างก็มีรูปร่างลักษณะ และการใช้งาน(การกระทำ) แตกต่างกันออกไป ซึ่งเราจะเรียกคุณลักษณะของวัตถุว่า แอตทริบิวต์ (Attribute) และเราจะเรียกวิธีการใช้งานวัตถุว่า เมธอด (Method) ยกตัวอย่างเช่น "ดินสอเป็นวัตถุที่มีลักษณะยาวเรียว ภายในเป็นไส้ถ่านใช้สำหรับเขียน การใช้งานดินสอทำได้โดยใช้มือจับและเขียนลงบนวัสดุรองรับ" จากประโยคข้างต้น เราสามารถจับใจความได้ว่า คุณลักษณะของวัตถุ (Attribute) ก็คือ "ยาวเรียวภายในเป็นไส้ถ่าน" ส่วนการใช้งาน (Method) ก็คือ "ใช้มือจับและเขียนลงบนวัสดุรองรับ"

2.10 OpenCV (Open Source Computer Vision Library)

เป็นเครื่องมือสำหรับการจัดการข้อมูลภาพแบบ real-time ทำงานร่วมกับ Intel Image Processing Library (IPL) ซึ่งจัดการกับข้อมูลภาพแบบดิจิตอลแบบพื้นฐาน OpenCVนอกจากจะสามารถทำการกรองข้อมูลภาพ (Image Filtering) และการจัดการแบบ พีระมิด(Pyramid) แล้ว ยังสามารถ ทำหน้าที่เป็นไลบรารี่ขั้นสูงสำหรับการจัดการเกี่ยวกับกล้อง (Camera Calibrating) ,การตรวจจับการเคลื่อนไหวของภาพ,วิเคราะรูปร่างวัตถุ

ไลบารี่ OpenCV

จากที่ได้ทำการศึกษาข้อมูลจากเอกสารคู่มืออธิบายรายละเอียดของโครงสร้าง ,หน้าที่การทำงานและบทบาทของ OpenCV (the Open Source Computer Vision Library) สำหรับสถาปัตยกรรมแบบอินเทล จุดประสงค์หลักเพื่อใช้สำหรับคอมพิวเตอร์วิชั่นแบบเวลาจริง ได้แก่ การระบุวัตถุ การทำการตัดแยกวัตถุและการจดจำวัตถุ, การจดจำใบหน้า, การจดจำท่าทาง ฯลฯ การทำงานของ OpenCV กับภาพที่เป็นอินพุต มีประสิทธิภาพในการทำงานสูง และมีค่าใช้จ่ายต่ำ ซึ่งเอกสารนี้ได้กล่าวถึงรายละเอียดต่างๆ ดังนี้

1. ความสัมพันธ์ระหว่าง OpenCV กับ ไลบรารี่อื่นๆ

OpenCV ออกแบบมาให้ใช้งานร่วมกับอินเทล IPL (Image Processing Library) และขยายการทำงานไปยังการวิเคราะห์ภาพและรูปแบบ ดังนั้น OpenCV จึงสามารถใช้งานร่วมกับรูปแบบภาพอื่นๆ ที่ใกล้เคียงกับ IPL ได้ OpenCV ยังใช้ Intel Integrated Performance Primitives (IPP) ในระดับต่ำอีกด้วย IPP ปรากฏบนหลายสภาพแวดล้อมของระบบปฏิบัติการ รวมถึง IA32, IA64, และ StrongARM เป็นต้น OpenCV สามารถใช้งาน IPP บนสภาพแวดล้อมของระบบปฏิบัติการเหล่านี้ได้

2. ความต้องการด้านฮาร์ดแวร์และซอฟแวร์

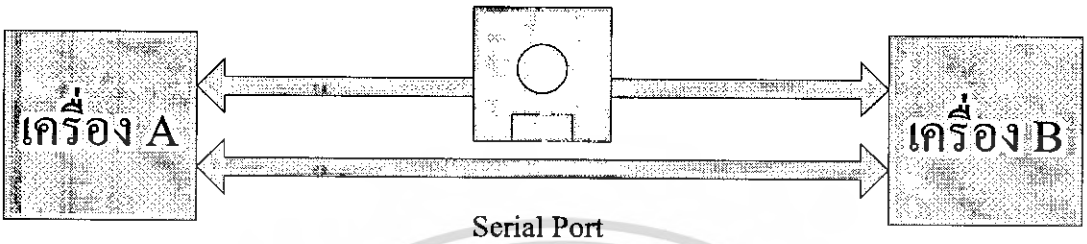
ซอฟต์แวร์ OpenCV ทำงานบนคอมพิวเตอร์ส่วนบุคคลซึ่งอยู่บนพื้นฐานของหน่วยประมวลผลสถาปัตยกรรมอินเทลและยังสามารถทำงานบนไมโครซอฟต์วินโดว์ 2000 ,98 ,95หรือ NT และสามารถใช้งานบนโปรแกรมประยุกต์ หรือ ไลบรารี่ในภาษา C/C++

2.11 TCP/IP Programming By Visual C++ V.6.0

2.11.1 การสื่อสารและ Network ในเชิง Programming

ในการสื่อสารข้อมูลระหว่างคอมพิวเตอร์จากโหนดหนึ่งไปโหนดหนึ่ง เช่นถ้าต้องการส่งไฟล์ 500K ถ้าไม่ใช่แผ่นดิสก์ ก็จะใช้สายอนุกรม (Serial Link) โดยต่อเข้าที่ช่องเชื่อมต่อแบบอนุกรม (Serial Port) ซึ่งถ้าใช้ Serial Port ก็จะเชื่อมต่อได้ แค่สองเครื่องเพราะสาย 1 เส้นจะมีหัวกับท้าย 2 หัว ถ้าจะเขียนโปรแกรมภาษา C/C++ เพื่อเชื่อมต่อเครื่องสองเครื่องผ่าน Serial Port เพื่อส่งไฟล์ข้อมูลระหว่างกันจะต้องทำการศึกษาการ

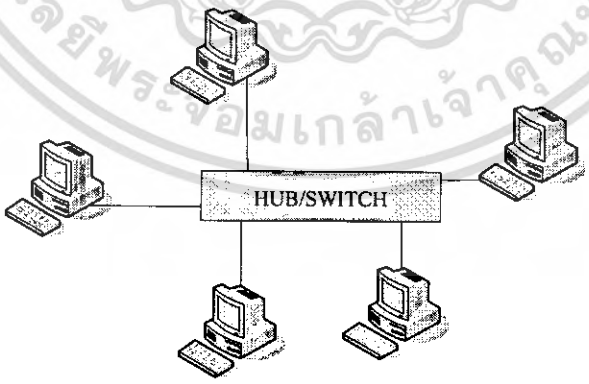
เขียนโปรแกรมออก Port และการทำงานของ Serial Port เพื่อดูวิธีการกำหนดค่าเริ่มต้นให้กับ Port กำหนดระดับบิตรับ/ส่ง พาริตี้บิต (Parity Bit) และทำการศึกษาชุดคำสั่งของภาษา C ที่ใช้ในการส่งข้อมูล ฯลฯ ที่ต้องศึกษาการทำงานของ Serial Port เพราะจะใช้ Serial Port เป็นสื่อ (media)



รูปที่ 2.8 เป็นการใช้สาย Serial Port เป็นสื่อในการส่ง

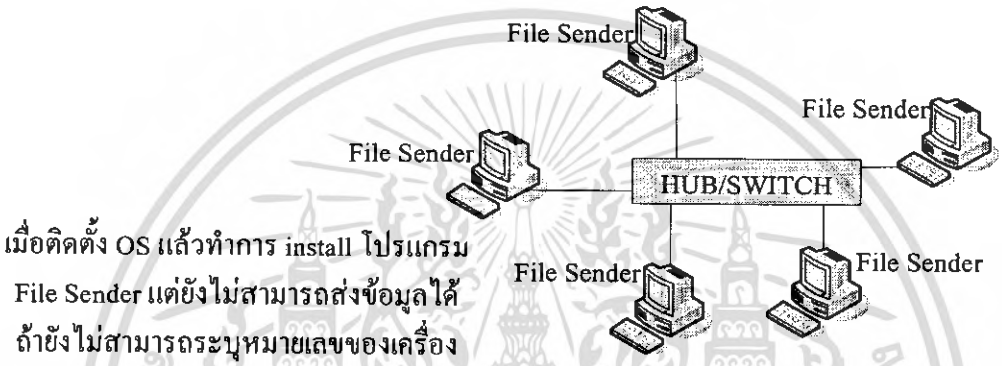
จากข้างต้นเป็นการเชื่อมต่อ2 คน ข่อมไม่เกิดปัญหา แต่ถ้าต้องการเชื่อมต่อกันหลายเครื่อง เช่น เชื่อมต่อ PC 5 เครื่องเข้าด้วยกันเพื่อ share ข้อมูลจะใช้ Serial Port ไม่ได้ และถ้าจะใช้ Disk หรือ Handy Drive ก็ไม่สามารถเก็บข้อมูลมากๆได้ เช่น หากจะทำการ share ไฟล์ 10 MB ให้กับเครื่องอีก 5 เครื่อง ก็คงไม่สามารถทำได้ จึงต้องใช้สื่อตัวอื่นที่เชื่อมต่อกันได้มากกว่านี้ ดีกว่า รวดเร็วกว่าและสะดวกกว่า นั่นก็คือเชื่อมต่อเป็นระบบเน็ตเวิร์คโดยใช้การ์ดระบบเครือข่าย และสายสัญญาณเชื่อมถึงกัน

การเชื่อมต่อคอมพิวเตอร์ 5 เครื่องเข้าด้วยกันจะต้องมีอุปกรณ์ชนิดหนึ่ง เรียกว่าLAN Card (LAN Card) หรือ Network Interface Card (NIC) โดยจะนำเอาการ์ดนี้เสียบเข้ากับ Main Board ของเครื่องคอมพิวเตอร์ จากนั้นก็ให้ใช้สายสัญญาณ UTP ที่มีหัวแบบ RJ-45 ต่อเข้ากับLAN Cardนั้น ส่วนปลายของสายสัญญาณอีกข้างก็เชื่อมเข้ากับ HUB หรือ Switch ก็จะทำให้เครื่องในกลุ่มนั้นมองเห็นกันได้ เชื่อมต่อถึงกันได้



รูปที่ 2.9 เป็นการแสดงการเชื่อมต่อคอมพิวเตอร์ 5 เครื่องโดยใช้ HUB/ SWITCH

จากข้างต้นนั้นเป็นการเชื่อมต่อทางด้าน Hardware หรือทางกายภาพเมื่อเครื่องคอมพิวเตอร์ เชื่อมต่อกันเป็นกลุ่มๆแบบนี้แล้ว การที่แต่ละเครื่องจะสื่อสารกันได้นั้นจะต้องมี Software เมื่อติดตั้ง OS ที่เป็น Window, Linux แล้วนั้น ตัว OS จะทำการจัดสรรและดู Hardware รวมทั้ง Hardware ที่เป็นชนิด Network ก็เช่นกัน และนอกจากจะต้องมี OS แล้ว ก็จะต้องมีโปรแกรมประยุกต์ เช่น โปรแกรม Chat หรือโปรแกรมส่งไฟล์หรือ share ไฟล์ให้แก่กัน เครื่องคอมพิวเตอร์ถึงจะส่งข้อมูลให้แก่กันได้ สมมติว่าโปรแกรมตัวนี้มีชื่อว่า File Sender ซึ่งเป็นโปรแกรมที่ใช้ในการส่งไฟล์ระหว่างเครื่องผ่านระบบเครือข่ายและจะต้องติดตั้งให้กับทุกเครื่อง



รูปที่ 2.10 เป็นการลงโปรแกรม File Sender ให้กับ คอมพิวเตอร์ทั้ง 5 เครื่อง

ตอนนี้เครื่องทั้ง 5 เครื่องที่ได้ยกตัวอย่างไป เมื่อติดตั้ง OS พร้อมแล้ว ติดตั้งโปรแกรม File Sender สำหรับส่งไฟล์แล้ว และจะต้องบอกว่าเครื่องใดเป็นเครื่องที่จะรับข้อมูล เช่น Com 1 จะเข้าเอาไปไฟล์ที่ Com 5 เปิด share ให้ Download เอาไว้ก็จะอ้างถึงคอมพิวเตอร์ Com5 โดยใช้หมายเลข MAC Address

Com 1 มี MAC Address คือ 08:0b:0e:12:b3:00
Com 2 มี MAC Address คือ 02:0f:0e:e7:b3:05
Com 3 มี MAC Address คือ 0e:02:ee:5e:b2:00
Com 4 มี MAC Address คือ 0e:08:2f:bb:b4:5b
Com 5 มี MAC Address คือ 0a:09:cc:04:e3:00

รูปที่ 2.11 เป็นการแสดงตัวอย่าง MAC Address

MAC Address นี้เป็นหมายเลขที่โรงงานผู้ผลิตได้กำหนดมาให้กับอุปกรณ์ซึ่งจะไม่ซ้ำกัน แต่ส่วนใหญ่ไม่เป็นที่นิยมใช้กันแต่จะใช้รูปแบบการสื่อสารให้ดูเหมือนว่าสื่อสารกันได้ง่ายๆ รูปแบบการสื่อสารนี้ ได้

ถูกกำหนดขึ้นมาในรูปของข้อกำหนดที่เรียกว่า "Protocol" เมื่อกำหนด Protocol มาเป็นมาตรฐานร่วมกันแล้ว ก็จะช่วยให้การสื่อสารโดยไม่ไปเกี่ยวข้องกับสิ่งที่เป็น Hardware มากนัก รูปแบบการสื่อสารของ Protocol จะครอบคลุมความยุ่งยากในการอ้างถึง MAC Address ในการสื่อสาร โดยสร้างตัวเลขที่จำง่ายๆขึ้นมาเหมือนกับเลขที่บ้าน โปรโตคอลที่ว่านี้เป็นโปรโตคอลที่นำไปใช้กันอย่างแพร่หลาย เพราะเป็นโปรโตคอล แบบเปิด ดังนั้นจึงกลายเป็นโปรโตคอลหลักของการสื่อสารบนอินเทอร์เน็ต ซึ่งก็คือ TCP/IP

TCP/ IP ย่อมาจาก Transmission Control Protocol / Internet Protocol ซึ่งในบทความนี้เนื้อหาเริ่มต้นก่อนการเข้าสู่การเขียนโปรแกรมบน Network ดังนี้

2.11.1.1 ทุกเครื่องต้องมี IP Address

TCP/IP บอกไว้ว่าทุกๆเครื่องจะต้องมีหมายเลข IP Address เป็นเสมือนกับเลขที่บ้าน คือ แทนที่จะเรียกหมายเลข MAC ก็จะเรียก เป็น IP แทน หมายเลข IP Address จะเป็นตัวเลข 4 ตัว คั่นด้วยเครื่องหมายจุด มีอยู่ด้วยกันทั้งหมด 5 กลุ่ม (Class) แต่ที่เห็นชัดๆ มีแค่ 3 กลุ่ม คือ คลาส A, B และ C ตัวอย่างหมายเลข IP Address มีดังนี้

192.168.10.1

202.44.35.80

54.94.128.3

หมายเลข IP Address จะถูก Bind เข้ากับอุปกรณ์เน็ตเวิร์คในตอนที่ OS เริ่มทำงาน การที่จะกำหนด IP Address นี้ ให้กับเครื่องคอมฯ ได้ จะต้องทำการติดตั้ง Protocol TCP/IP ลงไปก่อน ซึ่งในตอนที่ทำการติดตั้ง OS เช่น Windows หรือ Linux ตัวติดตั้งก็จะตรวจสอบว่ามี LAN Card ในเครื่องอยู่หรือไม่ ถ้ามีก็จะทำการติดตั้ง TCP/IP ให้โดยอัตโนมัติ

เมื่อ OS ได้ติดตั้ง TCP/IP แล้ว ก็สามารถกำหนดค่า IP Address ให้กับอุปกรณ์นั้นได้ เมื่อเปิดเครื่อง OS ถูกโหลดเข้าหน่วยความจำ ในตอนที่ OS ทำงานนี้ IP Address ก็จะถูก Bind เข้ากับหมายเลข MAC Address ซึ่งการ Bind นี้ก็คือการที่ทำให้เวลาเรียก IP Address แล้วอ้างถึงเครื่องๆ นั้นได้นั่นเอง ดังนั้น ถ้าหากไม่ทำการเปิดเครื่อง หรือ OS ไม่ทำงาน หรือเปิดเครื่องในแบบ Safe Mode ที่ไม่ได้โหลด Driver ของLAN Card หรือ LAN Cardเสียหาย หมายเลข IP Address ก็จะไม่ถูก Bind ทำให้เวลาใคร ping มาหาเรา หาไม่เจอ เพราะหมายเลข IP ของเราไม่สามารถอ้างถึงตำแหน่งเราใน Network ได้

2.11.1.2 Loopback IP Address

ถ้าติดตั้ง TCP/IP แล้ว ไม่ว่าจะกำหนดหมายเลข IP หรือไม่ก็ตาม สามารถอ้างถึงเครื่องของเราได้โดยใช้ IP Address ที่เรียกว่า Loop back (IP Address แบบตีกลับ) เช่นถ้าต้องการดูว่าการกำหนด IP ของเรานั้นยังทำงานเป็นปกติดีหรือไม่ เราก็สามารถทำการ ping ไปที่หมายเลข 127.0.0.1

การ ping นี้ไม่จำเป็นต้องต่อสายสัญญาณ เพราะเป็นการส่งสัญญาณไปทดสอบว่ายังทำงานเป็นปกติดีอยู่หรือเปล่า

2.11.1.3 Protocol ตัวอื่นๆที่ทำงานอยู่ภายใต้ TCP/IP

การสื่อสารกันภายใต้ TCP/IP นั้น สามารถทำได้หลายรูปแบบ เช่น ส่งไฟล์ เปิดไฟล์ข้อมูลธรรมดาๆ หรือจะส่งจดหมายถึงกัน ฯลฯ ซึ่งเป็นบริการหลายๆอย่างที่สามารถสร้างขึ้นมาได้ ซึ่งบริการเหล่านี้ ทั้งฝ่ายผู้ส่งและผู้รับจะต้องเข้าใจเหมือนๆกัน ยกตัวอย่างเช่น เวลาส่งไฟล์ข้อมูลถ้าฝ่ายส่งทำการส่งให้ทีละ 54 bytes ฝ่ายรับก็จะต้องรับทีละ 54 bytes เป็นต้น หรือเวลาจะส่งไฟล์ test.txt ไปให้กับอีกฝ่ายหนึ่ง ฝ่ายส่งจะต้องส่งชื่อไฟล์ test.txt ไปให้ก่อน จากนั้นจึงตามด้วยการส่งข้อมูลในไฟล์นั้น ขั้นตอนและกระบวนการเหล่านี้ จะต้องทำการตกลงกันให้ดี ดังนั้น จึงจะต้องมีการตั้งข้อกำหนดในการสื่อสารแบบต่าง ๆ เช่น แบบเว็บ, แบบรับส่งไฟล์, แบบพูดคุย ฯลฯ ตามจุดประสงค์ของการใช้งานและมีการสร้างข้อกำหนด (Protocol) ของบริการต่างๆนั้น เช่น

Protocol	ย่อมาจาก	ใช้สำหรับ
HTTP	Hyper Text Transfer Protocol	เปิดดูโฮมเพจ,เว็บเพจ
FTP	File Transfer Protocol	Download/Upload ไฟล์
POP	Post Office Protocol	รับ E-mail (pop v.3)
SMTP	Simple Mail Transfer Protocol	ส่ง E-mail
IRC	Internet Really Chat	พูดคุย

นอกจากนี้ ยังมี Protocol ที่ทำงานอยู่เบื้องหลังเพื่อสนับสนุนให้การทำงานของ บริการต่างๆ ภายใต้ TCP/IP เป็นไปด้วยดี เช่น

Protocol	ย่อมาจาก	ใช้สำหรับ
DNS	Domain Name System	แปลง IP ให้เป็นชื่อที่จำง่ายๆ
SNMP	Simple Network Management Protocol	ควบคุมและตรวจสอบอุปกรณ์
DHCP	Dynamic Host Configuration	แจกจ่ายข้อมูลของระบบเครือข่ายให้กับเครื่องลูกข่าย

บริการต่าง ๆ เหล่านี้ ทำงานภายใต้ TCP/IP ทั้งสิ้น อย่าง Protocol ที่อยู่เบื้องหลังนี้จะช่วยให้ผู้ใช้บริการทำงานได้ง่ายขึ้น เช่น DNS เป็นระบบที่ใช้แปลงหมายเลข IP ให้เป็นชื่อที่จำง่าย เช่นแทนที่จะจดจำชื่อ IP เวลาต้องการเชื่อมต่อก็ใช้ว่า myhost.com

ในการทำงานของบริการเหล่านี้ จะต้องมี 2 ฝั่งด้วยกันนั่นก็คือ ฝั่งที่เป็นผู้ให้บริการ (Server) และฝั่งที่เป็นผู้รับบริการ (Client) โดยผู้ให้บริการเหล่านี้ จะต้องมี Software ที่เรียกว่า Server และฝั่ง Client ก็จะต้องมี Software ที่เรียกว่า Client

ในการทำงานของโปรแกรม Server นั้น จะมีการกำหนดหมายเลข Port เอาไว้ เพื่อใช้เป็นหมายเลขอ้างอิงเวลาเชื่อมต่อ โดยหมายเลขนี้จะต้องไม่ซ้ำกันดังนี้

ตารางที่ 2.1 การกำหนดหมายเลข Port

Protocol	Port
HTTP	80
FTP	20,21
POP3	110
SMTP	25
IRC	6667

เมื่อ Server เปิดให้บริการที่ Port หมายเลขนั้นแล้ว ทางฝั่ง Client ที่เชื่อมต่อเข้ามา ก็จะต้องเชื่อมต่อเข้ามาที่ Port ที่ทาง Server กำหนดเอาไว้

2.11.1.4 การเชื่อมต่อ

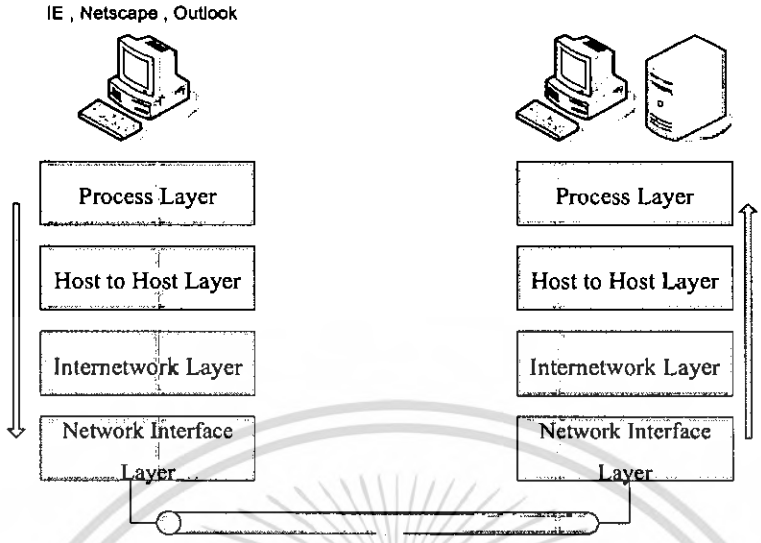
การเชื่อมต่อเพื่อขอเข้าไปใช้บริการ จะมีอยู่ด้วยกัน 2 ลักษณะคือ

1. Connection-Oriented (โปรโตคอล TCP) คือ จะต้องเชื่อมต่อกันก่อนเป็นการสร้างช่องทางเพื่อการสื่อสารขึ้น เมื่อเชื่อมต่อกันได้แล้วก็จะทำการส่งข้อมูลโดยในการส่งข้อมูลนั้น จะทำการแตกข้อมูลนั้นออกเป็นชิ้นเล็กๆจากนั้นนำไปประกอบกันที่ฝั่งตรงข้าม โดยผู้ที่จะทำหน้าที่ประกอบกันก็คือผู้เขียนโปรแกรมนั่นเอง

2. Connectionless (โปรโตคอล UDP) คือ สามารถส่งข้อมูลถึงกันได้เลยโดยไม่ต้องมีการสร้างช่องทางก่อน แต่ข้อมูลอาจสูญหายได้ เพราะวิธีนี้จะไม่มีการส่งสัญญาณตอบกลับมาว่าได้รับหรือไม่

บริการต่างๆ และโปรโตคอลที่ทำงานภายใต้ TCP/IP จะมีการเชื่อมต่อกันอยู่ใน 2 ลักษณะนี้ บางบริการก็ใช้การเชื่อมต่อ 2 ลักษณะ

กลไกการทำงานของการสื่อสารภายใต้ TCP/IP นั้นสามารถอธิบายได้โดยแบ่งเป็นลำดับชั้นที่เรียกว่า TCP/IP Stack เป็นการจำลองลำดับชั้นขึ้นมาจากกลไกและขั้นตอนการทำงาน โดยจะไล่จากระดับ Software ลงไปหา Hardware หรือทางกายภาพ (Physical) และเมื่อถึงฝ่ายรับก็ย้อนกับขึ้นกลับไปสู่ระดับของ Software หรือ Application ดังแสดงในรูปที่ 2.12



รูปที่ 2.12 TCP/IP Stack

จากรูปที่ 2.12 จะเห็นได้ว่าแต่ละฝ่ายจะอยู่ที่ชั้น Process Layer ของ TCP/IP Stack ซึ่งก็จะใช้โปรแกรมของตนเองไป หรือถ้าเป็นอีกฝั่งชั้นหนึ่งที่เป็นรูปเครื่อง Server ก็เป็น Application ที่คอยตอบรับตัว Client ตามปกติ คือ ทั้งสองฝั่งทำงานโดยจะไม่คำนึงถึงการส่งข้อมูลในสายส่ง Hardware เมื่อสื่อสารกันนี้ จะทำการเชื่อมต่อกับอีกฝั่งหนึ่งโดยกระบวนการในชั้น Host-to-Host Layer นี้ จะทำการเชื่อมต่อระหว่าง Host ต้นทางกับปลายทาง เรื่องของ Port และ Socket ที่ใช้เขียนโปรแกรมกันก็จะทำงานอยู่ชั้นนี้ ก็คือ Transport layer หรือ UDP คือจะเชื่อมต่อแบบ Connection Oriented หรือ Connection Less จากนั้นก็จะส่งต่อไปยัง Internetwork Layer ซึ่งจะมีการ Protocol IP , ICMP และ ARP ทำงานอยู่ภายในชั้นนี้ จากนั้นก็จะลงไปยังชั้น Network Interface Layer ซึ่งเป็นชั้นของการสื่อสารในระดับ Hardware แล้วเมื่อถึงปลายทางมันก็จะไล่ขึ้นไปทีละชั้นของ Process Layer

2.11.2 การให้บริการบนฝั่ง Server

การพัฒนาโปรแกรมบนระบบเครือข่าย โปรโตคอล TCP/IP เป็นโปรโตคอลหลักของการทำงานในระบบอินเตอร์เน็ต จากที่ได้กล่าวมา โปรโตคอล TCP/IP เป็นข้อกำหนดร่วมกันหรือข้อตกลงร่วมกันในการสื่อสารว่า ถ้าเราจะสื่อสารกันให้รู้เรื่อง จะต้องมีข้อตกลงร่วมกัน บริการต่างๆในอินเตอร์เน็ตก็จะมีโปรโตคอลที่ควบคุมการทำงานเป็นของตัวเอง เช่น การเปิดโฮมเพจ ใช้ข้อกำหนดโปรโตคอล HTTP แต่ถ้าจะเปิด Email ก็จะใช้โปรโตคอล POP3 ซึ่งโปรแกรมที่ทำหน้าที่เป็น Client ของบริการเหล่านี้จะต้องดำเนินการตามข้อกำหนดของโปรโตคอลนั้นๆ สังเกตได้ว่าโปรแกรมที่เป็นชนิด Web Browser หรือ Email Client เช่น Outlook หรือ Eudora นั้น สามารถเชื่อมต่อเข้าไปยัง Server ที่ให้บริการนั้น ๆ ได้โดย ไม่ผิดพลาด หรืออย่าง Web Browser เวลาทำการสั่งเปิดเว็บๆหนึ่ง สามารถทำได้โดยไม่ต้องคำนึงถึงว่า Server นั้นจะใช้ OS อะไร แต่

ก็สามารถเปิดเว็บได้ อ่านเมล์ได้ ที่เป็นเช่นนี้ก็เพราะว่า โปรโตคอล หรือ ข้อตกลงในการสื่อสารนี้ จะไม่คำนึงว่าผู้ให้บริการเป็น LINUX ผู้เข้ามาขอบริการจะต้องเป็น LINUX ด้วย จะต้องใช้ร่วมกันได้ทั้งหมด

ดังนั้น จากหลักการนี้ ถ้าท่านต้องการที่จะสร้างโปรแกรม Client เพื่อขอใช้บริการใดๆ ในอินเตอร์เน็ต จะต้องเชื่อมต่อเข้าไปยัง Server ที่เปิดให้บริการๆนั้น ซึ่งใน Server หนึ่งๆ สามารถเปิดให้บริการต่างๆ ได้ โดยการติดตั้งโปรแกรม Server ของบริการนั้นๆลงไป

ยกตัวอย่างเช่น ถ้าต้องการสร้าง Web Server เมื่อทำการ install OS เช่น Windows 2000 หรือ LINUX แล้ว ตอนนี้จะยังไม่เป็น Web Server จะต้องใช้ IE มาเปิดเว็บของเครื่องๆนี้ยังไม่ได้ จนกว่าจะติดตั้งโปรแกรม Web Server และสั่งให้ทำงานก่อน โปรแกรม Web Server ที่ติดตั้งกันใน LINUX ก็คือ Apache หรือถ้าจะติดตั้งบน Windows ก็จะใช้ IIS หรือ Personal Web Server ได้ โปรแกรม Web Server จะทำงานที่ Port หมายเลข 80คำว่า Port นี้ไม่ใช่ Port อนุกรมหรือ USB Port หากแต่เป็นหมายเลขที่ใช้อ้างถึงเวลาเชื่อมต่อเพื่อให้รู้ว่ากำลังจะเข้าไปใช้บริการนี้ ถ้า Web Server ทำงานที่ Port 82แต่ Web Client เชื่อมต่อเข้ามาที่ Port 80 ก็ไม่สามารถจะทำการเชื่อมต่อกันได้ เนื่องจากหมายเลข Port ไม่ตรงกัน

2.11.3 การสร้างโปรแกรม Client/Server แบบที่เป็นของตัวเอง

สามารถสร้างบริการอื่นๆ ที่เป็นของตัวเองได้ โดยในการสร้างโปรแกรม.exe ให้ทำงานภายใต้ TCP/IP นั้น จะประกอบไปด้วย 2 ฝั่ง คือ ฝั่ง Client และ Server โดยเมื่อเขียนโปรแกรมที่เป็น Server นั่นจะต้องกำหนดหมายเลข Port ขึ้นมา เพื่อให้เป็นช่องทางในการเข้ามาใช้บริการที่เป็นของตนเอง และก็จะต้องทำการออกแบบ โปรแกรมที่เป็น Client ด้วย

ยกตัวอย่างเช่น ICQ จะมีทั้งเป็น ICQ Server และ ICQ Client โดย ICQ Server จะติดตั้งเอาไว้ที่เครื่อง Server และเปิดทิ้งไว้ 24 ช.ม. รอการเชื่อมต่อ จากนั้นก็จะปล่อย ICQ Client ให้ผู้สนใจ Download ไปติดตั้ง โดยสื่อสารของ ICQ ทั้ง Server และ Client จะอยู่ในข้อกำหนดการสื่อสารของ ICQ เอง คือ มีหมายเลข Port ที่ไม่ไปชนกันกับบริการอื่นๆ ในอินเตอร์เน็ต การที่จะเขียนโปรแกรม ICQ Client เอง จะต้องเข้าใจและรู้วิธีการเชื่อมต่อและการใช้คำสั่งประมวลผล เช่น การ Authentication หมายถึง UIN และ Password ฯลฯ ถ้ารู้ ก็สามารถเขียนโปรแกรม ICQ Client ได้ และเช่นเดียวกัน สำหรับโปรโตคอล HTTP เป็นโปรโตคอลที่ใช้กันอย่างแพร่หลาย ใคร ๆ ก็สามารถเรียนรู้ได้เนื่องจากเป็นโปรโตคอลที่เปิดเผยเหมือน TCP/IP การเขียน Web Browser จึงไม่ใช่เรื่องยาก

จากข้างต้น ถ้าจะสร้างโปรแกรมประเภท Messenger หรือหากมี Idea สร้างโปรแกรมเฉพาะงานสักอย่าง เช่น โปรแกรมสื่อสารและรับส่งไฟล์ภายในกลุ่มก็จะต้องสร้างตัว Server ขึ้นมา กำหนด Port เอาไว้หมายเลขหนึ่ง และสั่งรันโปรแกรม Server เอาไว้ที่ใดที่หนึ่ง ผู้ที่ได้รับโปรแกรม Client ไป เมื่อรันมันก็จะเชื่อมต่อเข้ามาที่ Server และมาพบปะกับบุคคลอื่นได้ และแลกเปลี่ยนไฟล์กันได้, chat ได้ หากผู้ใดต้องการเข้ามา Join ต้องทำการ Download โปรแกรม Client ตัวนี้มาติดตั้ง จึงจะสามารถ Join กันได้

2.11.4 ใช้ Visual C++ และ MFC สร้างโปรแกรม Network

Visual C++ เป็นชุดพัฒนาที่ได้รับความนิยมมากตัวหนึ่ง สามารถใช้พัฒนาโปรแกรมบน Windows ได้ทุกประเภท รวมทั้งโปรแกรมบนระดับเครือข่ายด้วย

MFC เป็นชุดคำสั่งที่นิยมนำไปใช้ในการสร้างโปรแกรมบนวินโดวส์ และใน MFC นั้นจะมีชุดคำสั่งที่สามารถใช้สร้างโปรแกรมบนระบบเครือข่ายได้ นั่นก็คือ

1.คลาส CSocket และ CAsyncSocket

2.คลาส Winlnet

การเขียนโปรแกรมเชื่อมต่อกับระบบเครือข่ายนั้น แต่เดิมนั้นจะใช้ชุดคำสั่ง Socket Library แต่เมื่อความต้องการในการพัฒนาโปรแกรมที่รวดเร็วมีสูงขึ้น ชุดคำสั่งที่เป็นคลาสจึงถูกพัฒนาโปรแกรมที่รวดเร็วมีสูงขึ้น ชุดคำสั่งที่เป็นคลาสจึงถูกพัฒนาขึ้นเพื่อให้ครอบคลุมความยุ่งยากในการเขียนโปรแกรม ทำให้การสร้างโปรแกรมเหล่านั้นทำได้เร็วมากขึ้น แทนที่จะทำการประกาศตัวแปรโครงสร้างหลายตัวถึงจะเขียนโปรแกรม Client ของบริการ HTTP หรือ FTP ได้แล้ว

2.12 โปรโตคอล TCP/IP

โปรโตคอล TCP/IP มีกลไกการทำงานเป็นชั้นหรือ Layer เรียงต่อกัน โดยในแต่ละ Layer จะมีการทำงานได้เทียบได้กับ มาตรฐาน OSI Model แต่ในบาง Layer ของโปรโตคอล TCP/IP จะทำงานเทียบได้กับ OSI หลาย Layer ปนกัน ซึ่งในแต่ละ Layer ของโปรโตคอล TCP/IP จะประกอบด้วย

- Process layer
- Host-to-Host layer
- Internetwork layer
- Network Interface layer

2.12.1 Process layer

การทำงานของโปรโตคอล TCP/IP เทียบกับมาตรฐาน OSI model นั้น ในชั้นบนสุดเรียกว่า Process layer ทำงาน 2 หน้าที่เทียบได้กับ Application layer และ Presentation layer ในชั้นนี้จะรองรับการทำงานของ Application ต่างๆ ที่ทำงาน เป็นโปรเซส อยู่ในเครื่อง server ให้บริการและ client ซึ่งจะติดต่อกันผ่านโปรโตคอลเฉพาะแอพพลิเคชั่นอีกทีหนึ่ง ยกตัวอย่างเช่น เมื่อผู้ใช้งานอินเตอร์เน็ตต้องการถ่ายโอนไฟล์ หรือ Download ข้อมูลจากเครื่อง server ที่ให้บริการ โดยอาจเรียกใช้โปรแกรม FTP client ทั่วไป เช่น โปรแกรม WS FTP ติดต่อกับโปรเซส FTP ที่กำลังให้บริการอยู่ที่เครื่อง server จากนั้นตัวโปรเซส FTP ก็จะเรียกใช้โปรโตคอล FTP (File Transfer Protocol) เพื่อทำการถ่ายโอนไฟล์นี้

การทำงานของแอพพลิเคชั่นต่างๆ จะอยู่ที่ Process layer นี้ และมีการติดต่อกันตามแต่โปรโตคอลเฉพาะแล้วแต่แอพพลิเคชั่นที่ใช้งาน

โปรโตคอลหลักๆที่ทำงานใน Process layer ซึ่งผู้ใช้มักจะคุ้นเคยกันดี ได้แก่ FTP (File Transfer Protocol), Telnet, HTTP (Hyper Text Transfer Protocol) และ SMTP (Simple Mail Transfer Protocol) นอกจากนี้ยังมีโปรโตคอลที่อยู่เบื้องหลัง ซึ่งทำงานโดยที่ผู้ใช้ไม่สามารถมองเห็นได้จากโปรแกรมหรือไม่ได้มีการใช้งานโดยตรง เช่น

- โปรโตคอล DNS (Domain Name System) ทำหน้าที่แปลงข้อมูล domain name หรือชื่อ Website ทั้งหลายให้เป็น IP Address หรือ ทำการแปลง IP Address ให้เป็นชื่อ domain name

- โปรโตคอล SNMP (Simple Network Management Protocol) ใช้ในการควบคุมและตรวจสอบอุปกรณ์ที่อยู่ในเครือข่าย

- โปรโตคอล DHCP (Dynamic Host Configuration Protocol) ทำหน้าที่แจกจ่ายข้อมูลพารามิเตอร์ของเครือข่ายให้กับเครื่องลูกข่ายที่เชื่อมต่ออยู่

2.12.2 Host – to – Host layer

การทำงานที่ชั้น Host-to-Host layer นี้จะมีบทบาทในการจัดการต่อจาก Process layer บางครั้งเราเรียกชั้น Host-to-Host layer นี้ว่าเป็น Transport layer ซึ่งไม่ใช่ชั้นของ Transport layer ใน OSI model การทำงานของ Host-to-Host layer นี้จะมีการสร้าง connection หรือการเชื่อมต่อกันระหว่างแอพพลิเคชั่นกับ Host-to-Host layer โดยที่จุดที่เชื่อมต่อกันเพื่อรับส่งข้อมูลนี้เรียกว่า port (คำว่า portในที่นี้ไม่ได้หมายถึง port ในทาง hardware) และในแอพพลิเคชั่น ก็จะสร้างการเชื่อมต่อผ่าน port ได้พร้อมกันหลายแอพพลิเคชั่น ซึ่งการใช้งาน port ของแต่ละแอพพลิเคชั่นที่อยู่ในชั้น Process layer จะแตกต่างกันตามหมายเลขที่กำหนดไว้ และแต่ละโปรโตคอลจะมีการใช้งาน port หมายเลขต่างๆ ไม่ซ้ำกัน

เมื่อแอพพลิเคชั่นทำงานผ่านโปรโตคอลในชั้น Process layer จะมีการส่งผ่านข้อมูลไปยัง Host-to-Host layer ที่ชั้นนี้จะมีการเชื่อมต่อผ่าน port ที่กำหนด ทำให้การรับส่งข้อมูลในแต่ละโปรโตคอลทำได้อย่างถูกต้อง ถึงแม้ว่าในเครื่อง server ที่ให้บริการจะมีการทำงานอยู่หลายโปรเซสที่แตกต่างกันก็ตาม

2.12.3 โปรโตคอล TCP

โปรโตคอล TCP (Transmission Control Protocol) เป็นโปรโตคอลที่มีการรับส่งข้อมูลแบบ stream oriented protocol หมายความว่า การรับส่งข้อมูลจะไม่คำนึงถึงปริมาณข้อมูลที่ส่งไป แต่จะแบ่งข้อมูลออกเป็นส่วนย่อยๆก่อน แล้วจึงจะส่งไปยังปลายทางอย่างต่อเนื่องเป็นลำดับข้อมูล ในกรณีที่ข้อมูลส่วนใดส่วนหนึ่งสูญหายไปก็จะส่งข้อมูลส่วนนั้นใหม่อีกครั้ง สำหรับปลายทางก็จะทำหน้าที่จัดเรียงส่วนของข้อมูล Datagram ใหม่ให้ต่อเนื่องกันและประกอบกลับเป็นข้อมูลทั้งหมดได้ ซึ่งจะแยกข้อมูลที่ไม่ถูกต้องออก ดังนั้น แอพพลิเคชั่นหรือโปรเซสเซอร์ใดที่อาศัยการส่งผ่านข้อมูลด้วยโปรโตคอล TCP จะต้องใช้หน่วยความจำและขนาดของช่องสัญญาณ (Baudwidth) มากกว่าแบบ UDP

การติดต่อระหว่างกันจะเป็นแบบ Connection-oriented คือ ต้องมีการสร้างการติดต่อกันเป็น session ทั้ง 2 ด้านเสียก่อน แล้วจึงจะรับส่งข้อมูลไปได้พร้อมกัน (full duplex) เหมือนกับการใช้โทรศัพท์ติดต่อกัน เมื่อผู้ติดต่อต้นทางเรียกให้ฝั่งตรงข้ามรับสายแล้ว จึงเริ่มต้นการสนทนา เมื่อแอพพลิเคชั่นต้องการส่งผ่านข้อมูลจะใช้โปรโตคอลที่เหมาะสมในชั้น Process layer ติดต่อไปและมีการสร้างช่องทางส่งข้อมูลผ่าน port ที่กำหนดเพื่อส่งผ่านข้อมูลไปยังโปรโตคอล TCP

ในระหว่างการรับส่งข้อมูลนี้ โปรโตคอล TCP จะเพิ่มกระบวนการสอบถามข้อมูล (acknowledgement) และส่งข้อมูลใหม่อีกครั้ง หากปลายทางยังไม่ได้รับข้อมูลหรือเกิดความผิดพลาดขึ้น

2.12.4 Internetwork layer

ในระดับล่างต่อมาในชั้น Internetwork layer มีหน้าที่ส่งผ่านข้อมูลระหว่างเครือข่าย โดยมีโปรโตคอลที่ทำงานเป็นกลไกสำคัญในการส่งผ่านข้อมูลไปยังเครือข่ายใดๆ บนอินเตอร์เน็ต คือ โปรโตคอล IP (Internet Protocol)

2.12.5 โปรโตคอล IP

โปรโตคอล IP ทำหน้าที่ให้บริการส่งผ่านข้อมูลที่มาจาก Host-to-Host layer เพื่อส่งข้ามไปยังเครือข่ายใดๆ ได้อย่างถูกต้อง แม้ว่าจะมีเครือข่ายเชื่อมต่อกันอยู่ในอินเตอร์เน็ตเป็นล้านๆ เครือข่ายก็ตาม เนื่องจากโปรโตคอล IP มีข้อมูลตำแหน่ง IP ปลายทางที่จะส่งข้อมูลไป โดยทำงานร่วมกับอุปกรณ์ Router เพื่อส่งข้อมูลผ่าน switch ไปยังปลายทาง ตัววงจรผ่าน หรือ switch นี้อาจเป็น Gateway หรือ Router ในระบบเครือข่ายก็ได้ ซึ่งในข้อมูลของโปรโตคอล IP จะมีข้อมูลของหมายเลข IP ให้เป็นหมายเลข hardware และจำเครื่องที่ถูกต้องด้วยโปรโตคอล ARP (Address Resolution Protocol)

2.12.6 กลไกของโปรโตคอล IP

ในการส่งผ่านข้อมูล หรือ IP Datagram ไปยังเครือข่ายอินเตอร์เน็ตนั้น โปรโตคอล IP จะทำหน้าที่พิจารณาว่าปลายทางในการส่ง IP datagram นั้นจะเป็นภายในเครือข่ายของตนเองหรือจะต้องส่งข้อมูลข้ามเครือข่ายไปอีก โดยการพิจารณานี้โปรโตคอล IP address ต้นทางหรือไม่ ถ้าตรงกันแสดงว่าการส่งข้อมูลอยู่ภายในเครือข่ายเดียวกัน แต่ถ้าต่างกันแสดงว่าต้องส่งข้อมูลไปยังปลายทางที่อยู่คนละเครือข่ายกัน
การส่งข้อมูลภายในเครือข่ายเดียวกันมีกลไกดังนี้

1. โปรโตคอล IP จะเรียกใช้บริการโปรโตคอล ARP (Address Resolution Protocol) เพื่อแปลงหมายเลข IP ปลายทาง ให้เป็นค่าหมายเลข hardware เช่น MAC address
2. เมื่อโปรโตคอล IP รับค่าหมายเลข hardware แล้ว ก็จะส่งข้อมูลนั้นไปยัง hardware ที่ระบุไว้

การส่งข้อมูลข้ามเครือข่ายมีกลไกดังนี้

1.โปรโตคอล IP จะตรวจสอบพบว่า หมายเลข IP address ปลายทางอยู่คนละเครือข่ายกัน โดยโปรโตคอล IP จะอ่านค่า IP address ของ Router เพื่อเตรียมส่งข้อมูลไปยัง Router แทนซึ่งในที่นี้จะมีการกำหนดเป็น default Router

2.โปรโตคอล IP จะเรียกใช้บริการโปรโตคอล ARP เพื่อแปลงค่า IP address ของ Router ให้เป็นค่าหมายเลข hardware

3.โปรโตคอล IP ส่งข้อมูล IP datagram ไปยัง Router ที่กำหนดเอาไว้ จากนั้น Router ส่งข้อมูลข้ามเครือข่ายไปตามขั้นตอน

2.13 ไมโครคอนโทรลเลอร์ MCS-51

ปัจจุบันเครื่องใช้ไฟฟ้าอุปกรณ์อิเล็กทรอนิกส์ตลอดจนระบบโรงงานอุตสาหกรรมจะใช้ไมโครคอนโทรลเลอร์ซึ่งติดตั้งอยู่ภายในเป็นตัวควบคุมเกือบทั้งหมด ทำให้ไมโครคอนโทรลเลอร์กลายเป็นอีกหนึ่งอุปกรณ์ที่ผู้ผลิตสารกึ่งตัวนำหลายๆ บริษัทให้ความสนใจ และมีการแข่งขันสูงมาก อุปกรณ์ที่ใช้ไมโครคอนโทรลเลอร์ควบคุมมีมากมายหลายชนิด เช่น อุปกรณ์เครื่องใช้ภายในบ้าน สัญญาณไฟจราจร รถยนต์ตลอดจนระบบอุตสาหกรรม PLC , CNC , Robot เป็นต้น

ปัจจุบันไมโครคอนโทรเลอร์ (MCS-51) ซึ่งเป็นไมโครคอมพิวเตอร์แบบชิปเดี่ยว (Single Chip Microcontroller) ถูกใช้งานอย่างแพร่หลาย การศึกษาเกี่ยวกับไมโครคอนโทรลเลอร์จึงมีองค์ประกอบหลายอย่างเช่น สถาปัตยกรรมของไมโครคอนโทรลเลอร์ วงจรอิเล็กทรอนิกส์ ตลอดจนภาษาที่จะใช้ในการควบคุมอีกด้วย

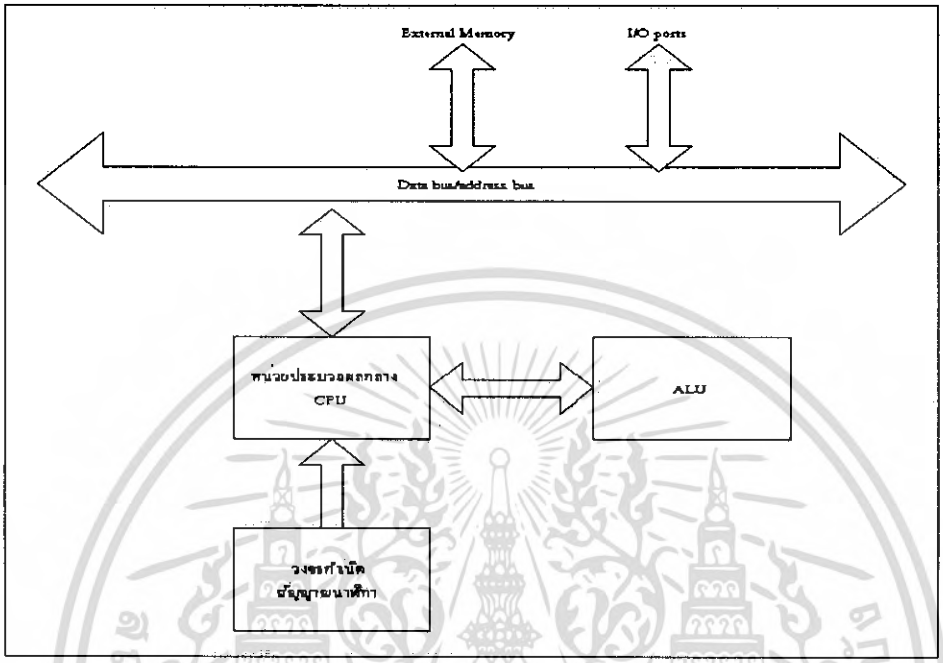
2.13.1 ความหมายของไมโครคอนโทรลเลอร์

ไมโครคอนโทรลเลอร์ คือ อุปกรณ์อิเล็กทรอนิกส์อย่างหนึ่งซึ่งภายในประกอบด้วยวงจรอื่นๆหลายวงจรและทำงานร่วมกัน เช่น หน่วยประมวลผลกลาง (CPU: Central Processing Unit) หน่วยคำนวณทางคณิตศาสตร์และลอจิก (ALU: Arithmetic Logic Unit) วงจรออสซิลเลเตอร์ หน่วยความจำ (Memory: ROM, RAM) วงจรรับสัญญาณอินพุทและขับสัญญาณเอาต์พุท (I/O Port) เป็นต้น ด้วยเหตุนี้ไมโครคอนโทรลเลอร์จึงสามารถนำไปประยุกต์ใช้งานควบคุมต่างๆได้ดี เนื่องจากสามารถเขียนโปรแกรมควบคุมได้อย่างอิสระแล้วแต่เราต้องการควบคุมอะไร

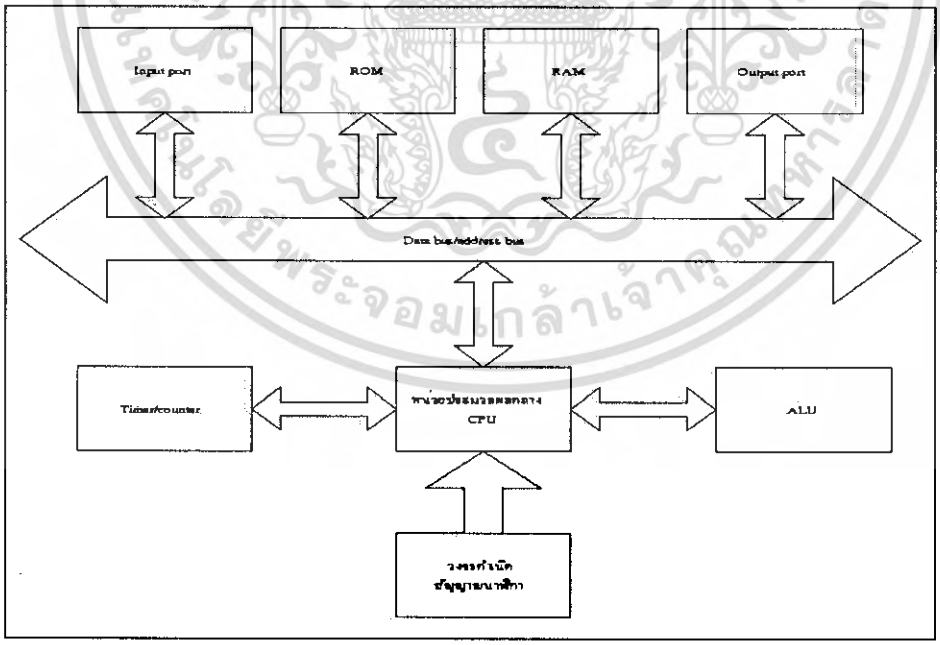
2.13.2 ข้อแตกต่างระหว่างไมโครโปรเซสเซอร์กับไมโครคอนโทรลเลอร์

ไมโครโปรเซสเซอร์ที่ใช้อยู่ในปัจจุบัน เช่น ซีพียูเบอร์ Z80 เป็นต้น จะไม่มีหน่วยความจำ RAM , ROM และ Port อยู่ในตัวชิป ทำให้ต้องต่อหน่วยความจำโปรแกรมภายนอกเพิ่มและต้องใช้ ICs ขยายพอร์ตเพิ่มเติมข้อดีคือ สามารถเพิ่มหน่วยความจำได้ตลอด ส่วนไมโครคอนโทรลเลอร์จะมีวงจรพื้นฐานประกอบอยู่

ภายในชิป เช่นหน่วยความจำ RAM ,ROM และ I/O Port ดังนั้น ในระบบไมโครคอนโทรลเลอร์จึงมีขนาดเล็กกว่าและราคาถูกกว่าระบบไมโครโปรเซสเซอร์



รูปที่ 2.13 โครงสร้างพื้นฐานของไมโครโปรเซสเซอร์



รูปที่ 2.14 โครงสร้างพื้นฐานของไมโครคอนโทรเลอร์

2.13.3 โครงสร้างของไมโครคอนโทรลเลอร์ (MCS-51)

Microcontroller MCS-51 มีโครงสร้างหลายลักษณะ ทั้ง 20pin และ 40pin โดยมีหน่วยความจำแบบ ROM หรือ EPROM ภายในมีขนาดไม่เกิน 4K byte และมีหน่วยความจำแบบ RAM 256 byte โดย Microcontroller MCS-51 แบบ Flash มีพอร์ตให้ใช้งานทั้งสิ้น 4พอร์ต คือ P0 , P1 , P2 และ P3 แต่ละพอร์ตจะมีขนาด 8บิต เป็นพอร์ตแบบมี 2ทิศทาง คือ สามารถเป็นได้ทั้งอินพุทและเอาต์พุท โดยมีคุณลักษณะดังต่อไปนี้

- มีพอร์ต I/O ขนาด 8บิต 4พอร์ต
- มีหน่วยความจำ ROM 4K byte
- มีหน่วยความจำ RAM 128 byte
- อ้างตำแหน่งหน่วยความจำโปรแกรมภายนอก 64K byte
- อ้างตำแหน่งหน่วยความจำข้อมูลภายนอก 64K byte
- Timer/Counter 16 bit 2 ตัว
- วงจรสื่อสารแบบอนุกรมแบบฟูลดูเพล็กซ์ (Full Duplex)
- วงจรควบคุมการอินเตอร์รัปต์จากแหล่งกำเนิดสัญญาณ 6ประเภท
- วงจรออสซิสเลเตอร์ภายใน
- มีหน่วยความจำโปรแกรมแบบ Flash สามารถเขียนและลบได้พันครั้ง

โครงสร้างทางไมโครคอนโทรลเลอร์ MCS-51 แบบ Flash จะประกอบด้วยส่วนต่างๆดังต่อไปนี้

1.หน่วยประมวลผลกลาง (CPU: Central Processing Unit)

CPU เปรียบได้กับสมองของคนเรานั่นเอง เพราะการคำนวณต่างๆ เกิดขึ้นที่ CPU ประกอบด้วยวงจรต่างๆ หลายวงจร เช่น Decoder, Register, Counter, Adder, Subtraction, Buffer, Oscillator เป็นต้น

2.หน่วยความจำ (Memory)

ในการเขียนโปรแกรมด้วยภาษา C ให้กับไมโครคอนโทรลเลอร์นั้นต้องคำนึงถึงชนิดของหน่วยความจำและวิธีการเข้าถึงด้วย ซึ่งต่างจากการเขียนบน PC ที่สนใจเพียงชนิดของตัวแปรว่าจะใช้เก็บข้อมูลประเภทใด สำหรับหน่วยความจำในระบบไมโครคอนโทรลเลอร์นั้นแบ่งเป็น 2 ประเภท ดังนี้

2.1 หน่วยความจำข้อมูลภายใน (Internal RAM)

หน่วยความจำส่วนนี้มีไว้ใช้เก็บข้อมูลขณะประมวลผลโปรแกรม สามารถอ่านและเขียนข้อมูลได้ขณะมีไฟเลี้ยง แต่เมื่อไม่จ่ายไฟเลี้ยงข้อมูลต่างๆ จะสลายไป หากหน่วยความจำส่วนนี้ไม่พอใช้งานจะต้องต่อหน่วยความจำแรมภายนอก (External RAM หรือ Data Memory) ปัจจุบันเทคโนโลยีก้าวหน้าขึ้นมากชิปบางตัวจะมีการบรรจุหน่วยความจำประเภท Data Memory รวมเข้าไปในชิปเลย

2.2 หน่วยความจำโปรแกรม (Program Memory หรือ ROM)

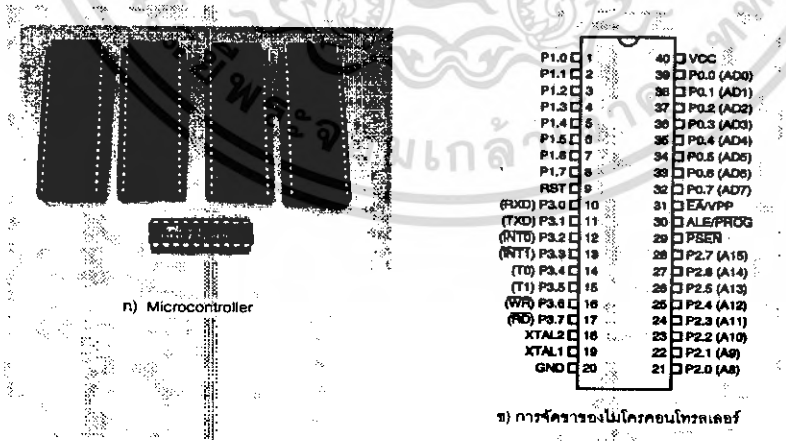
หน่วยความจำส่วนนี้ใช้เก็บโปรแกรมที่เราเขียนมา สามารถอ่านได้อย่างเดียวขณะประมวลผล ถ้าจะเขียนข้อมูลลง ROM จะต้องใช้เครื่องโปรแกรม เมื่อก่อนจะเก็บไว้ใน EPROM แต่ปัจจุบันจะใช้ Flash ROM ที่สามารถเขียนและลบได้ด้วยกระแสไฟฟ้า ซึ่งสะดวกและรวดเร็วมาก

3.พอร์ตอินพุท/เอาต์พุท (I/O port)

พอร์ตมีหน้าที่ทำให้ระบบไมโครคอนโทรลเลอร์สามารถติดต่อสื่อสารกับอุปกรณ์ภายนอกได้ แล้วแต่วัตถุประสงค์ในการใช้งานและคุณสมบัติของพอร์ต เช่น สามารถติดต่อสื่อสารกับอุปกรณ์ Pushbutton, Keypad, Sensor, LCD เป็นต้น

ตารางที่ 2.2 Microcontroller MCS-51

เบอร์	หน่วยความจำโปรแกรม	หน่วยความจำข้อมูล	Timer/Counter
AT89C1051	1K byte	64 byte	1
AT89C2051	2K byte	128 byte	2
AT89C51	4K byte	128 byte	2
AT89C52	8K byte	256 byte	3
AT89C55	20K byte	256 byte	3
AT89S53	12K byte	256 byte	3

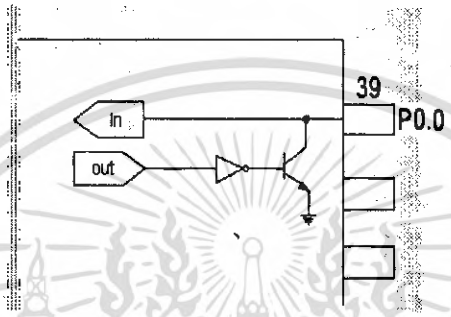


รูปที่ 2.15 ไมโครคอนโทรลเลอร์ตระกูล MCS-51

● **ขา 40 "Vcc":** ขานี้ต้องต่อไฟเลี้ยง 5+V DC

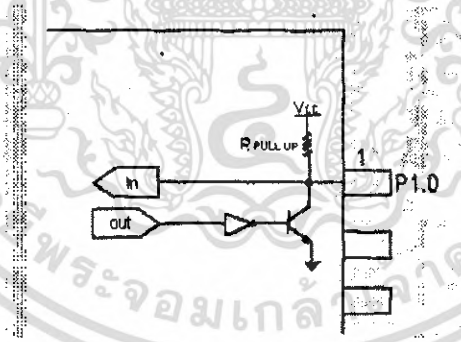
● **ขา 20"GND":** ต่อลงกราวด์ของระบบ

● **ขา 39-32"Port 0":** ขาทั้ง 8 ขานี้มีหน้าที่ใช้งานได้ทั้งอินพุทและเอาต์พุทสำหรับใช้ควบคุมอุปกรณ์ภายนอก โดยถ้าต้องการใช้ขาเป็นอินพุทต้องเขียนลอจิก 1 ไปที่ขาที่ต้องการ (ที่ขา port 0 มีสถานะ High Impedance เพราะไม่มีการต่อ R pull-up ไว้ภายใน) นอกจากนี้ยังใช้ติดต่อกับขา address ไบต์ต่ำของหน่วยความจำภายนอก (A0-A7) และขาข้อมูล (D0-D7) โดยวิธีการมัลติเพล็กซ์ เพื่อสลับหน้าที่การทำงาน



รูปที่ 2.16 โครงสร้างอย่างง่ายภายใน Port 0

● **ขา 8-1"Port 1":** ขานี้มีหน้าที่ใช้งานได้ทั้งอินพุทและเอาต์พุทสำหรับใช้ควบคุมอุปกรณ์ภายนอกภายในมีการต่อ R pull-up อยู่แล้ว



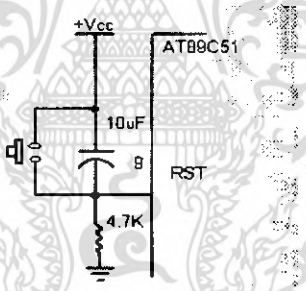
รูปที่ 2.17 โครงสร้างอย่างง่ายภายใน Port 1

● **ขา 28-21"Port 2":** ขาทั้ง 8 ขานี้มีหน้าที่ใช้งานได้ทั้งอินพุทและเอาต์พุทสำหรับใช้ควบคุมอุปกรณ์ภายนอก นอกจากนี้ยังใช้ติดต่อกับขา Address ไบต์สูงหน่วยความจำภายนอก (A8-A15) โครงสร้างพอร์ตภายในเหมือนพอร์ต 1

● **ขา 17-10"Port 3":** ขาทั้ง 8 ขานี้มีหน้าที่ใช้งานได้ทั้งอินพุทและเอาต์พุทสำหรับใช้ควบคุมอุปกรณ์ภายนอก นอกจากนี้ยังมีหน้าที่พิเศษในการติดต่อรับสัญญาณควบคุมการทำงานของไมโครคอนโทรลเลอร์ดังต่อไปนี้

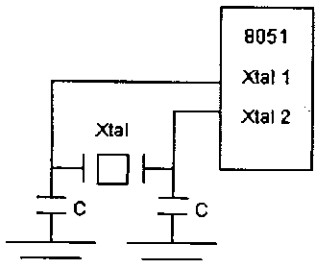
- P3.0 (RXD) ใช้รับสัญญาณจาก Serial Port หรือ RS-232
- P3.1 (TXD) ใช้ส่งสัญญาณทาง Serial port
- P3.2 (INT0) ใช้รับสัญญาณ interrupt หมายเลข 0
- P3.3 (INT1) ใช้รับสัญญาณ interrupt หมายเลข 1
- P3.4 (T0) เป็นขารับสัญญาณ pulse หมายเลข 0เข้าวงจร Counter
- P3.5 (T1) เป็นขารับสัญญาณ pulse หมายเลข 1เข้าวงจร Counter
- P3.6 (WR) เป็นขาสัญญาณเขียน RAM
- P3.7 (RD) เป็นขาสัญญาณอ่าน RAM

● **ขา 9"Reset":** ขานี้มีไว้สำหรับรีเซ็ตไมโครคอนโทรลเลอร์ โดยระบบจะรีเซ็ตเมื่อขานี้ได้รับสัญญาณพัลส์ (Pulse) บวกเป็นเวลาอย่างน้อย 2 แมชชีนไซเคิล (Machine Cycle) ดังนั้นจะต้องสร้างวงจรรีเซ็ตให้กับขานี้ โดยใช้วงจร RC หรือใช้ IC reset โดยเฉพาะก็ได้ เมื่อเริ่มจ่ายไฟ C จะเริ่มเก็บประจุ ขณะนี้ C เปรียบเสมือน short circuit แรงดันตกคร่อม C เป็น 0โวลต์ แต่แรงดันตกคร่อม R = 5V เมื่อเวลาผ่านไป C ก็เก็บประจุมากขึ้นๆ ตรงกันข้ามแรงดันคร่อม R = 0V ด้วยเหตุนี้เองทำให้เกิดสัญญาณพัลส์ที่ขา 9ส่วนความกว้างของพัลส์นั้นขึ้นอยู่กับค่า R , C ซึ่งเป็นไปตามหลักการของวงจร RC ส่วนสวิตช์ที่คร่อม C นั้นมีหน้าที่ discharge ให้กับ C หรือเป็นการ reset MCS-51



รูปที่ 2.18 วงจร Reset

● **ขา 9 ,18"X ' tal 1 , X ' tal 2":** ขาทั้งสองข้างนี้มีไว้ต่อกับวงจรสร้างสัญญาณนาฬิกาให้กับ MCS-51 ค่าความถี่ให้ใช้ ตามสเปกของแต่ละเบอร์ หากต้องการสื่อสารแบบอนุกรมด้วยค่าที่เหมาะสม คือ 11.0592 MHz เพราะค่านี้จะตรงกับมาตรฐาน RS-232 ดังรูปที่ 1.7ส่วนค่า C ให้ใช้ได้ตั้งแต่ 35-20 pF กรณีที่งานที่จะออกแบบไม่มีการสื่อสารอนุกรมก็สามารถใช้ X' tal ค่าอื่นก็ได้ที่อยู่ในช่วงที่ชิปรับได้ เช่น 24-4MHz



รูปที่ 2.19 การต่อวงจรสร้างสัญญาณนาฬิกา

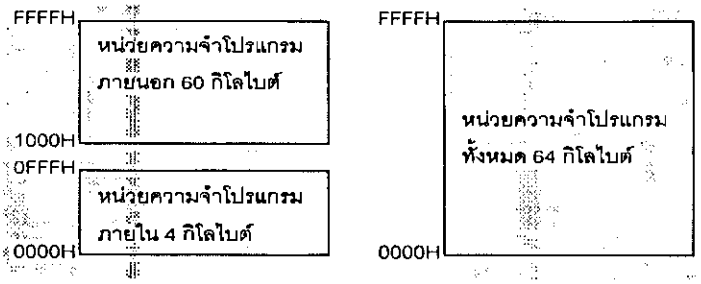
- ขา 30"ALE/PROG": เป็นขาที่บอกว่ามีสัญญาณแอดเดรส A0-A7 ออกมาจากพอร์ต 0 byteต่ำ เพื่อให้ไอซีแลตช์ (IC Latch)ทำการเก็บค่าแอดเดรสดังกล่าวไว้ก่อนที่จะหายไป เนื่องจากการมัลติเพล็กซ์เป็นดาต้าบัส (Data Bus)
- ขา 29"PSEN": Program Store Enable ขานี้จะใช้ติดต่อกับหน่วยความจำโปรแกรมภายนอกโดยจะส่งสัญญาณมาที่ขานี้ 2 ครั้ง ในแต่ละ แมชชีนไซเคิล
- ขา 31"EA / Vpp": เป็นขาเลือกว่าจะใช้หน่วยความจำโปรแกรมภายในหรือภายนอก โดยถ้าขานี้ได้รับลอจิก 1จะเป็นการใช้หน่วยความจำโปรแกรมภายใน

2.13.4 การจัดการหน่วยความจำของไมโครคอนโทรลเลอร์ ตระกูล MCS-51 แบบแฟลช

ในไมโครคอนโทรลเลอร์ตระกูล MCS-51 แบบแฟลชมีหน่วยความจำอยู่ 2 ส่วนคือหน่วยความจำโปรแกรม(Program Memory) และหน่วยความจำข้อมูล (Data Memory) ซึ่งชิปแต่ละเบอร์ก็จะมีขนาดและราคาไม่เท่ากัน ทั้งนี้ขึ้นอยู่กับความต้องการของผู้ใช้

1.หน่วยความจำโปรแกรม (Program Memory)

หน่วยความจำชนิดนี้ใช้เก็บ โปรแกรมควบคุมการทำงานของไมโครคอนโทรลเลอร์หรือโปรแกรมมอนิเตอร์ (Monitor Program) อ่านได้อย่างเดียว ซึ่งสามารถติดต่อหน่วยความจำโปรแกรมได้สูงสุด 64 กิโลไบต์ ถ้าเป็นเบอร์ AT89C51 จะมีหน่วยความจำโปรแกรม 4กิโลไบต์ หากไม่เพียงพอก็อาจต่อหน่วยความจำโปรแกรมภายนอกเพิ่มหรือใช้เบอร์ที่มีขนาดหน่วยความจำโปรแกรมใหญ่ขึ้น เช่น AT89C52 ในบางเบอร์ก็มีหน่วยความจำโปรแกรมภายในถึง 64 กิโลไบต์ แต่ราคาสูงขึ้นมาก



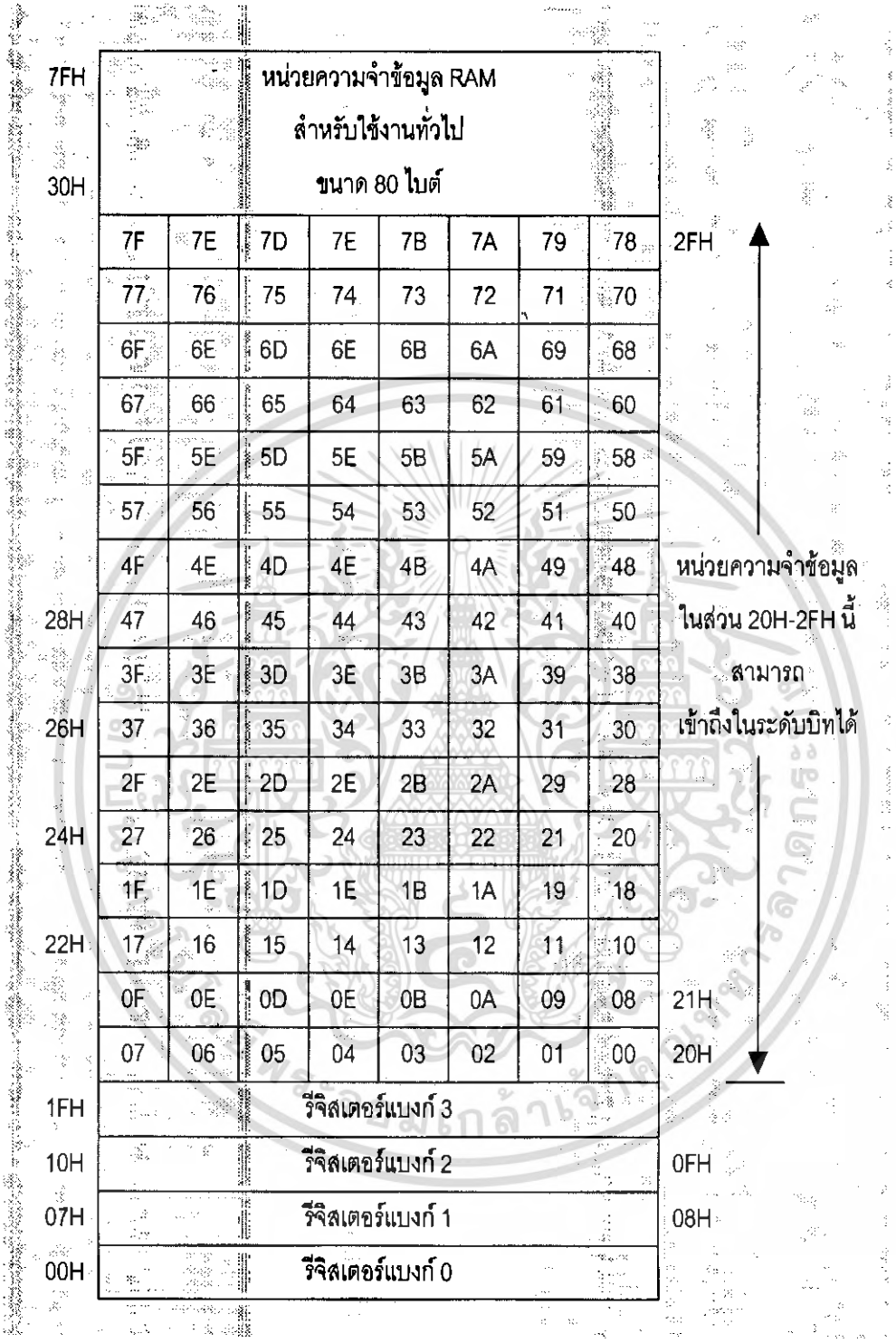
รูปที่ 2.20 แสดงการจัดพื้นที่หน่วยความจำโปรแกรมของชิปเบอร์ AT89C51

2.หน่วยความจำข้อมูล (Data Memory)

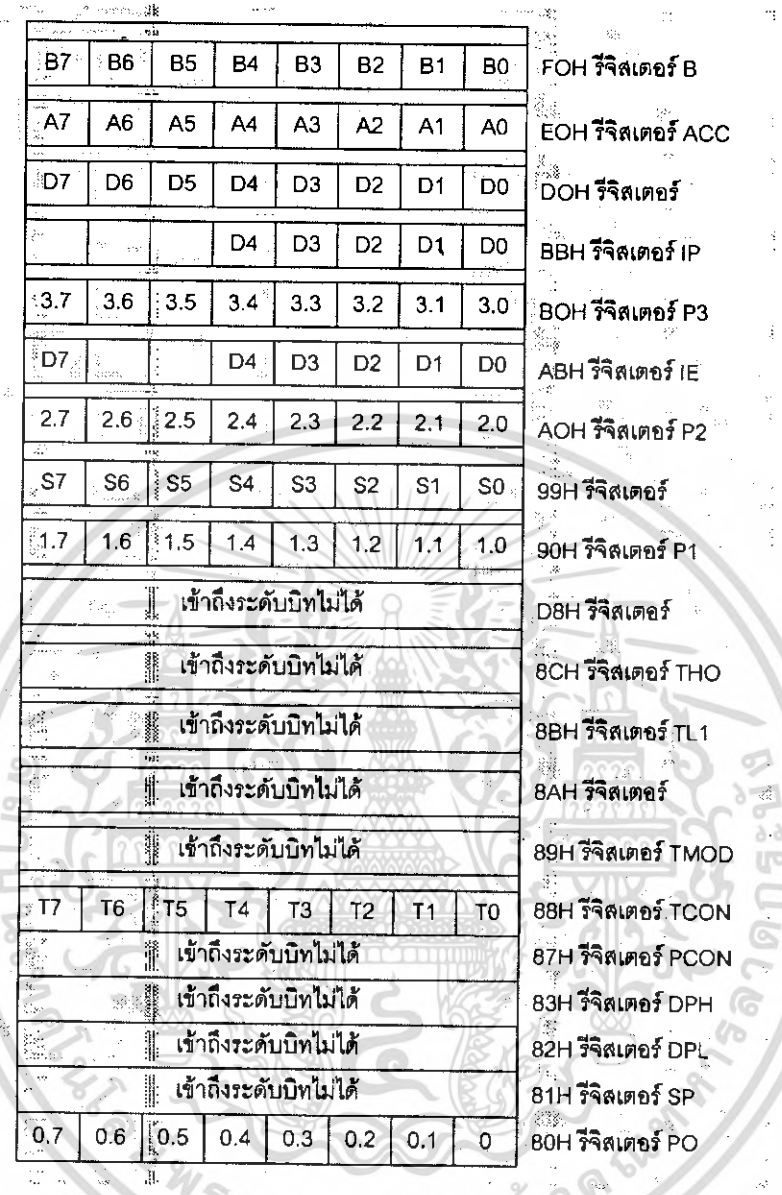
หรือเรียกอีกอย่างหนึ่งว่า หน่วยความจำแรม หน่วยความจำชนิดนี้สามารถอ่านและเขียนได้ซึ่งมีด้วยกัน 2แบบคือ หน่วยความจำข้อมูลภายใน (Internal Ram) และหน่วยความจำข้อมูลภายนอก (External RAM) ขนาดก็แตกต่างกันในแต่ละเบอร์

รูปที่ 2.21 พื้นที่หน่วยความจำภายใน (Internal RAM)

จากรูปที่ 2.21 จะเห็นว่ามีการแบ่งพื้นที่หน่วยความจำข้อมูลภายใน (Internal RAM) ออกเป็น 3ส่วน คือ หน่วยความจำข้อมูลส่วนล่างซึ่งมีขนาด 128ไบต์ (00H-7FH) ส่วนบนขนาด 128ไบต์ (80H-FFH) และรีจิสเตอร์ฟังก์ชั่นพิเศษ (SFR: Special Function Register) ซึ่งพื้นที่ส่วนนี้จะซ้อนทับกับหน่วยความจำข้อมูลส่วนบน แต่มีวิธีการเข้าถึงแตกต่างกัน



รูปที่ 2.22 แสดงพื้นที่หน่วยความจำข้อมูลภายใน (Internal RAM) ขนาด 128 ไบต์



รูปที่ 2.23 การจัดพื้นที่จองรีจิสเตอร์ฟังก์ชั่นพิเศษ (SFR)

2.14 อินเตอร์รัปต์ (Interrupt)

ในหน่วยนี้จะศึกษาเกี่ยวกับการอินเตอร์รัปต์ (Interrupt) ว่าเกิดขึ้นได้อย่างไร มีหลักการทำงานอย่างไร มีความสำคัญอย่างไรและจะเกิดอะไรขึ้นเมื่อมีการอินเตอร์รัปต์ สำหรับไมโครคอนโทรลเลอร์ MCS-51 จะมีการเกิดสัญญาณอินเตอร์รัปต์ ได้หลายลักษณะคือ

1. สัญญาณการเกิดอินเตอร์รัปต์จากภายนอก (External Interrupt)
2. สัญญาณการเกิดอินเตอร์รัปต์จากภายใน (Internal Interrupt)

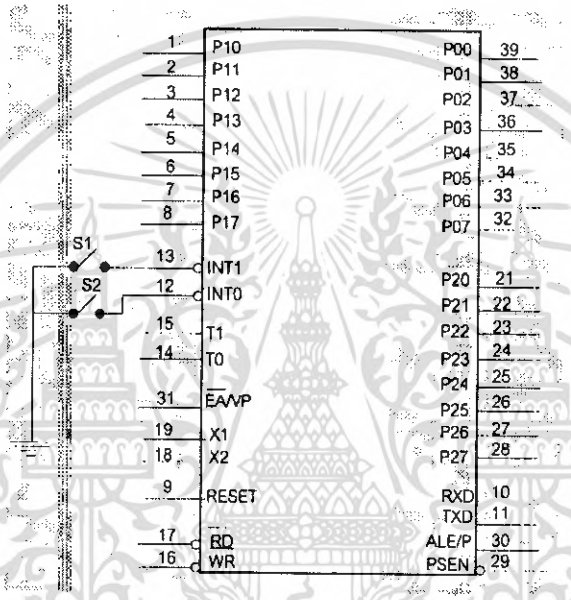
3. สัญญาณการเกิดอินเตอร์รัปต์จากวงจรนับเวลา (Time Interrupt)

4. สัญญาณการเกิดอินเตอร์รัปต์จากการรับส่งข้อมูลอนุกรม (Serial Interrupt)

โดยในหน่วยนี้จะกล่าวถึงการเกิดอินเตอร์รัปต์จากภายนอก จากวงจรนับเวลา และจากการรับส่งข้อมูลอนุกรม ดังต่อไปนี้

2.14.1 อินเตอร์รัปต์ภายนอก (External Interrupt)

อินเตอร์รัปต์ภายนอก หมายถึง การเกิดอินเตอร์รัปต์ภายนอกของไมโครคอนโทรลเลอร์ ไม่ได้เกิดจากภายใน โดยจะเกิดขึ้นที่ขาสัญญาณ INT0 (P3.2) และขาสัญญาณ INT1 (P3.3) ของพอร์ต 3 นั่นเอง



รูปที่ 2.24 การอินเตอร์รัปต์ภายนอก

จากรูปสามารถนำเอา Switch S1, S2 หรือ Sensor มาต่อเข้ากับสัญญาณ INT0 หรือ INT1 ได้โดยตรง ส่วนอีกด้านหนึ่งก็จะต่อลงกราวด์ ทั้ง 2 ขานี้ จะทำงานก็ต่อเมื่อเราป้อนลอจิก "0" ให้กับตัวมัน เพราะขานี้จะทำงานที่ลอจิก "0" ซึ่งที่ขานี้ไม่จำเป็นต้องต่อ R pull-up เพราะภายในพอร์ตมี R pull-up ภายในอยู่แล้ว

● **Interrupt มีหลักการอย่างไร**

Interrupt ก็คือ การขัดจังหวะของการทำงานในช่วงเวลาหนึ่ง แล้วให้ไปทำงานอีกอย่างหนึ่ง เมื่อทำเสร็จแล้วก็จะกลับไปทำงานเดิม ตัวอย่างเช่น สมมติว่าขณะที่เรากำลังดูทีวีอยู่นั้น เกิดมีเสียงโทรศัพท์ดังขึ้น เราก็จำเป็นที่จะต้องหยุดดูทีวีเพื่อมารับโทรศัพท์ชั่วขณะก่อน เมื่อคุยเสร็จแล้วจึงกลับไปดูทีวีต่อ ลักษณะนี้โทรศัพท์เป็นตัว Interrupt

● **ชนิดของการ Interrupt**

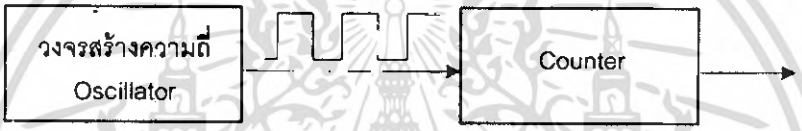
การเกิดอินเตอร์รัปต์ก็สามารถแบ่งออกตามลักษณะการทำงานได้ 2 ลักษณะ ดังนี้

- Software Interrupt หมายถึง การ Interrupt จากคำสั่งที่เราเขียนเช่น คำสั่ง CALL ในภาษาแอสเซมบลี ส่วนภาษาซีก็ใช้การเขียนฟังก์ชั่น Interrupt ขึ้นมา

- Hardware Interrupt หมายถึง การ Interrupt จากวงจรภายนอกหรือวงจรภายใน เช่น External Interrupt, Timer Interrupt, Serial Interrupt เป็นต้น

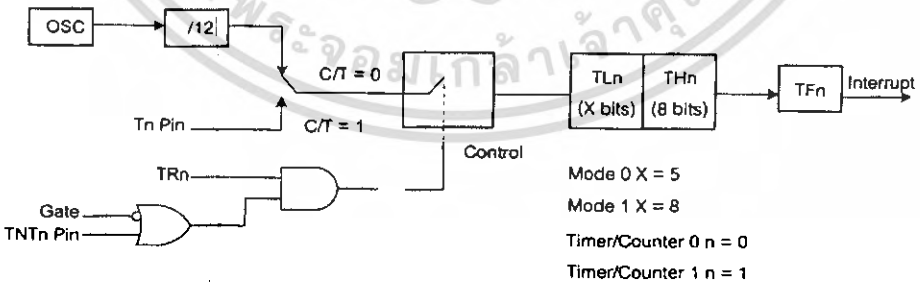
2.14.2 Timer Interrupt

ในหน่วยนี้จะกล่าวถึงการทำงานพื้นฐานของวงจร Timer ว่าทำงานอย่างไร และมีรีจิสเตอร์อะไรบ้างที่เกี่ยวข้องกับ Timer อีกทั้งวิธีคำนวณหาค่าความถี่ของ Timer ที่ต้องการ



รูปที่ 2.25 แสดงวงจร Timer พื้นฐาน

จากรูปที่ 2.25 เป็นวงจรพื้นฐานของวงจร Timer ใน MCS-51 สามารถแบ่งออกเป็น 2 ส่วนหลักๆ คือ วงจร Oscillator ซึ่งทำหน้าที่กำเนิดสัญญาณนาฬิกาขึ้นมา จากนั้นก็จะส่งสัญญาณนาฬิกาไปให้วงจร Counter เพื่อทำหน้าที่หารความถี่ ซึ่งขึ้นอยู่กับค่าหารของวงจร Counter ว่ามีค่าเท่าไร สมมติให้วงจรหารด้วย 50หมายความว่า ต้องมีสัญญาณนาฬิกาเข้าที่วงจร Counter จำนวน 50 ลูก จึงจะมีสัญญาณนาฬิกาออกที่เอาท์พุทของวงจร Counter 1 ลูก



รูปที่ 2.26 แสดงวงจรการทำงานของวงจร Timer

ตารางที่ 2.3 TMOD (Timer/Counter Mode Control Register) ตำแหน่งที่ 89H

Gate1	C/T1	M1	M0	Gate0	C/T0	M1	M0

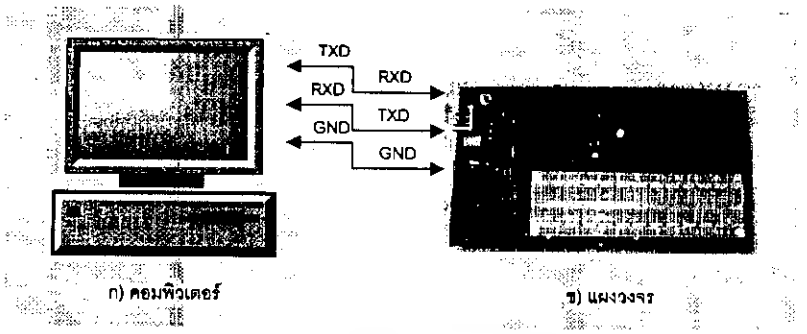
บิต	ความหมาย
Gate1	Timer/Counter จะทำงานเมื่อบิต TR ใน TCON = 1และสถานะลอจิกที่ขา INT0, INT1 เป็น "1" เป็นการควบคุมทาง Hardware
C/T1	เลือกให้ทำงานเป็น Counter จะรับอินพุทจากภายนอกเข้าทางขา T0 หรือขา T1
M1	เป็นบิตบนสำหรับเลือก Mode การทำงานของ Timer 1
M0	เป็นบิตล่างสำหรับเลือก Mode การทำงานของ Timer 1
Gate0	Timer/Counter จะทำงานเมื่อบิต TR ใน TCON = 1 เป็นการควบคุมทาง Software
C/T0	เลือกให้ทำงานเป็น Timer0
M1	เป็นบิตบนสำหรับเลือก Mode การทำงานของ Timer 0
M0	เป็นบิตบนสำหรับเลือก Mode การทำงานของ Timer 0

ตารางที่ 2.4 การเลือก Mode Timer/Counter

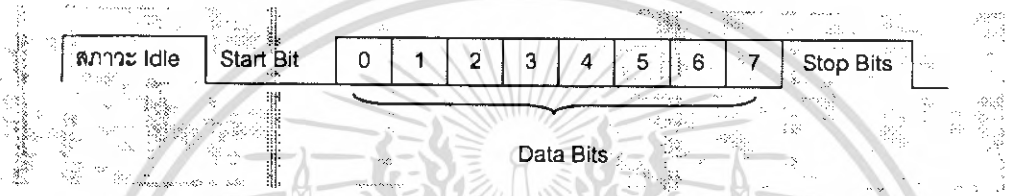
M1	M0	Mode	การทำงาน
0	0	0	ใช้งาน Timer/Counter แบบ 13Bit
0	1	1	ใช้งาน Timer/Counter แบบ 16Bit
1	0	2	ใช้งานแบบ 8Bit Auto Reload
1	1	3	ใช้งานแบบแยกบิตโดยแบ่งเป็นแบบ 8Bit 2 ตัว

2.14.3 พอร์ตอนุกรม Serial Port

จากรูปที่ 2.27 แสดงการเชื่อมต่อสายสัญญาณระหว่างคอมพิวเตอร์กับบอร์ดไมโครคอนโทรลเลอร์ โดยลักษณะสัญญาณในการเชื่อมกันนั้นจะเป็นแบบ RS232 ซึ่งมีระดับสัญญาณลอจิก "1" ที่ระดับแรงดัน -3V ถึง -12V และลอจิก "0" ที่แรงดัน 3V ถึง 15V



รูปที่ 2.27 แสดงการเชื่อมต่อแบบ RS-232



รูปที่ 2.28 สัญญาณ RS-232

รูปข้างบนแสดงถึงลักษณะของสัญญาณที่ใช้ในการติดต่อ ผ่านมาตรฐาน RS-232 เพื่อให้อุปกรณ์ตัวหนึ่งกับอีกตัวหนึ่งสามารถสื่อสารกันได้ จึงต้องกำหนดมาตรฐานการสื่อสารขึ้นมา โดยลักษณะของสัญญาณนั้นจะประกอบด้วย 4 ส่วน คือ

Start Bit เป็นบิตสำหรับการเริ่มต้นในการติดต่อสื่อสาร (logic 0 เสมอ)

Data Bits เป็นบิตสำหรับเป็นข้อมูล มีทั้งหมด 8 บิต โดยเริ่มต้นด้วยบิตต่ำ

Stop Bits เป็นบิตสำหรับการจบในการติดต่อสื่อสาร (logic 0 เสมอ)

Idle เป็นสภาวะที่ขา RXD ของ MCS-51 รออยู่

ในแต่ละบิตจะมีการกำหนดความถี่ขึ้นมา เพื่อให้สามารถสื่อสารกันได้อย่างถูกต้อง โดยจะกำหนดให้ความกว้างของแต่ละบิตมีความถี่ค่าหนึ่ง ความถี่ค่านี้เรียกว่า Baud Rate หรืออัตราในการส่งข้อมูล เช่น Baud Rate 9600 หมายความว่า ใช้ความถี่ในการติดต่อสื่อสารที่ความถี่ 9600 Hz หรือสามารถส่งข้อมูลได้สูงสุด 9600 Bit/Sec นั่นเอง

- **การทำงานของวงจรพอร์ตอนุกรม**

เริ่มต้นจากสร้างความถี่ในการสื่อสาร (Baud Rate) ก่อนโดย MCS-51 จะบังคับให้ใช้ Timer 1 เท่านั้น ซึ่งเราจะต้องกำหนดให้ Timer 1 ทำงานที่โหมด 2 ซึ่งคล้ายกับโหมด 1 ของ Timer 0 ในหน่วยที่ผ่านมา ซึ่งโหมด1 เป็นแบบ 16 บิต แต่โหมด 2 จะทำงานแบบ 8 Bit Auto Reload หมายความว่า เมื่อสัญญาณนาฬิกาที่ได้จากวงจร Oscillator ถูกหารด้วย 12 แล้วป้อนเข้าที่รีจิสเตอร์ TL1 ค่าของ TL1 จะถูกเพิ่มค่าไปเรื่อยๆ จนเป็น

00H ค่าที่อยู่ใน รีจิสเตอร์ TH1 ใหม่จะถูกใส่เข้าไปที่รีจิสเตอร์ TL1 ทันที แล้ว TL1 ก็จะถูกเพิ่มไปเรื่อยๆจน เป็น 00H ใหม่ ทำให้เอาต์พุทของ TL1 เป็นสัญญาณนาฬิกาค่าหนึ่ง

หลังจากที่ได้สัญญาณนาฬิกาจากวงจร Timer 1 แล้วสัญญาณจะถูกส่งไปยังวงจรหารความถี่ด้วย 16 กับ 32 โดยมีบิต SMOD ของรีจิสเตอร์ PCON เป็นตัวเลือกวงจรหาร หลังจากนั้นเราก็จะได้ความถี่ Baud Rate ป้อนให้กับวงจร Shift Register ในการส่งข้อจาก Serial Port ผ่านทางขา TxD และ RxD ซึ่งจะมีความสัมพันธ์กับรีจิสเตอร์ SCON

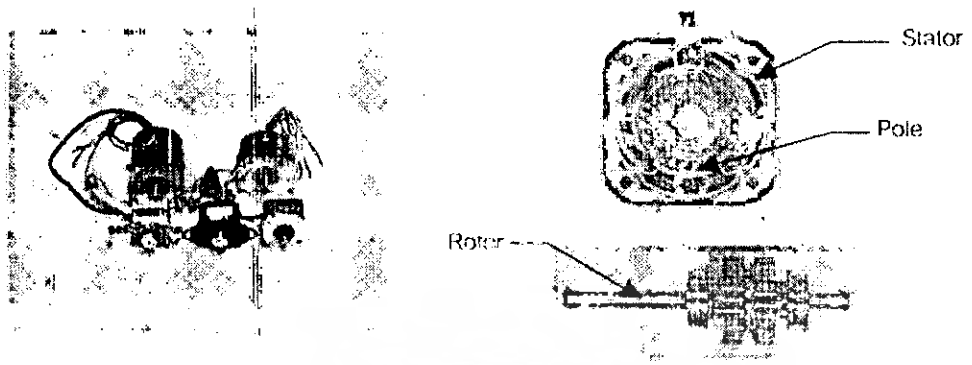
2.15 มอเตอร์สเต็ปปิ้ง (Stepping Motor)

โดยทั่วไปแล้วมอเตอร์ไฟฟ้ากระแสตรง (DC Motor) หรือมอเตอร์ไฟฟ้ากระแสสลับ (AC Motor) จะมีการหมุนที่ต่อเนื่อง ไม่สามารถควบคุมการหมุนที่เป็นองศาหรือระยะทางได้ Stepping Motor จึงเป็นทางเลือกอีกทางเลือกหนึ่งที่สามารถควบคุมการหมุนตามความต้องการได้ ลักษณะการทำงานของ Stepping Motor จะต้องป้อนสัญญาณพัลส์ (Pulse) ให้กับขดสเตเตอร์ (Stator) ทำให้เกิดแรงผลักที่โรเตอร์ (Rotor) จึงเกิดการหมุน โดยปกติแล้วการหมุนของมอเตอร์ จะหมุนเป็นจังหวะ (Step) ตามสัญญาณพัลส์ที่ป้อนเข้ามา การหมุนของ Stepping Motor เมื่อหมุนครบ 1 รอบ เท่ากับ 360 องศา ถ้า Stepping Motor มีการหมุนเท่ากับ 5 องศาต่อสเต็ป ดังนั้นความละเอียดการหมุนของ Stepping Motor ตัวนี้เท่ากับ 72 สเต็ปต่อรอบ โดยทั่วไปแล้ว Step Motor ถูกใช้งานอย่างแพร่หลายในปัจจุบัน เพราะสามารถควบคุมการหมุนตำแหน่งใดก็ได้ เช่น หัวอ่าน CD ROM , Tape Classes ตลอดจนอุตสาหกรรมการผลิตต่างๆ เช่น หุ่นยนต์อุตสาหกรรม ,ระบบสายพาน เป็นต้น

Stepping Motor ที่ใช้อยู่ทั่วไปสามารถแบ่งออกได้เป็น 3 ประเภท คือ

- แบบแม่เหล็กถาวร (PM : Permanent Magnet)
- แบบเปลี่ยนแปลงค่ารีลักแทนต์ (VR : Variable Reluctance)
- แบบผสม (Hybrid)

Stepping Motor แบบ PM นี้ ตัวโรเตอร์จะทำด้วยแม่เหล็กถาวร และ สเตเตอร์จะประกอบด้วยกลุ่มของขดลวด เมื่อจ่ายกระแสไฟฟ้าทำให้เกิดแรงผลักที่โรเตอร์ จึงเกิดการหมุนที่เป็นองศาเท่าๆกัน ส่วน Stepping Motor แบบ VR ชนิดนี้ จะมีการเปลี่ยนแปลงความต้านทานในวงจรแม่เหล็ก ส่วน Rotor จะหมุนได้อย่างอิสระและมีความเฉื่อยน้อย โดยทั่วไปแล้ว Stepping Motor แบบ VR จะมีความเร็วสูงกว่าแบบ PM และ Stepping Motor แบบ Hybrid เป็นการผสมกันระหว่างแม่เหล็กถาวร PM กับเปลี่ยนแปลงค่ารีลักแทนต์ VR ทำให้ Hybrid มีแรงบิดสูงกว่า PM และ VR



ก) Stepping Motor ข) โครงสร้างภายใน

รูปที่ 2.29 แสดง Stepping Motor และโครงสร้างภายใน

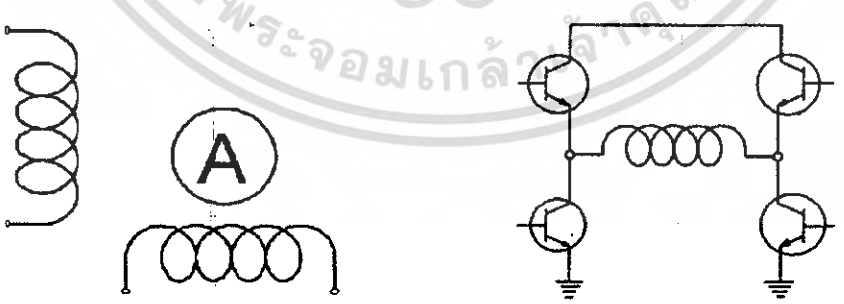
2.15.1 การพันขดลวดของ Stepping Motor

Stepping Motor โดยทั่วไปมีการพันขดลวด 2 ชนิด คือ

- แบบไบโพลาร์ (Bipolar)
- แบบยูนิโพลาร์ (Unipolar)

แบบไบโพลาร์ (Bipolar)

Stepping Motor ชนิดนี้มีการพันขดลวด 1 ขด บนขั้วแม่เหล็กของสเตเตอร์ ขั้วแม่เหล็กของสเตเตอร์จะถูกกำหนดโดยทิศทางการไหลของกระแสไฟฟ้า ซึ่งการทำให้ขั้วแม่เหล็กเกิดทิศทางตรงกันข้ามโดยการกลับทิศทางการไหลของกระแสไฟฟ้า โดยทั่วไปแล้วแบบ Bipolar จะทำให้เกิดแรงบิด (Torque) มากกว่าแบบ Unipolar

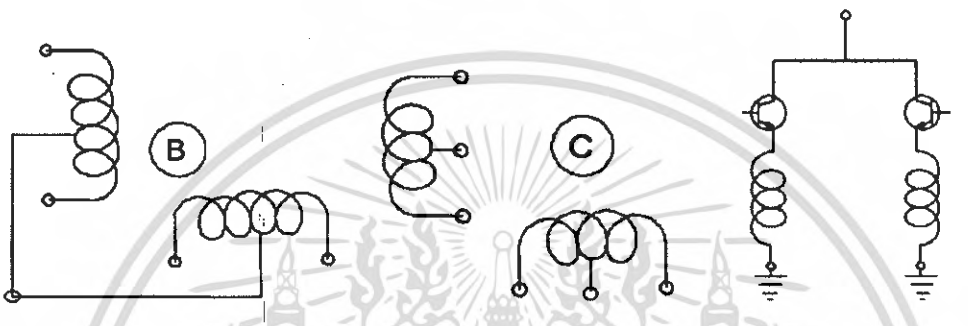


ก) แบบ Bipolar ข) วงจรสวิทชิ่ง

รูปที่ 2.30 แสดงการพันขดลวดและวงจรสวิทชิ่งแบบ Bipolar

แบบยูนิโพลาร์ (Unipolar)

Stepping Motor ชนิดนี้มีการพันขดลวด 2 ขด บนแต่ละขั้วแม่เหล็กของสเตเตอร์ แต่ละขดจะแบ่งออกเป็น 2 เฟส รวมทั้งหมดมี 4 เฟส คือ เฟส 1, 2, 3 และ 4 โดยทั่งไปแล้ว Stepping Motor ชนิดนี้จะมีแบบ 5 สาย 4 เฟส และ 6 สาย 4 เฟส โดยแบบ 5 สาย 4 เฟส จะมีสายออกมาจากมอเตอร์ 6 เส้น โดยสาย 4 เส้นแรกจะต่อที่เฟสของมอเตอร์ และสายอีก 2 เส้น (Common) จะนำมาต่อรวมกัน



ก) แบบ Unipolar 5 สาย ข) แบบ Unipolar 6 สาย ค) วงจรสวิทชิ่ง

รูปที่ 2.31 แสดงการพันขดลวดและการควบคุมวงจรสวิ่ทชิ่งแบบ Unipolar

2.15.2 การควบคุมการหมุนของ Stepping Motor

การให้ Stepping Motor หมุนได้นั้น จะต้องมีการป้อนสัญญาณพัลส์ ที่ขดลวดแต่ละเฟสของ Stepping Motor ให้ไปทิศทางเดียวกันหรือเรียงลำดับ (Sequence) และถ้าต้องการให้ Stepping Motor หมุนกลับทิศทางก็จะต้องป้อนสัญญาณพัลส์ ในทิศตรงข้าม

การควบคุมการหมุนของStepping Motor สามารถแบ่งออกเป็น 3 ลักษณะดังนี้

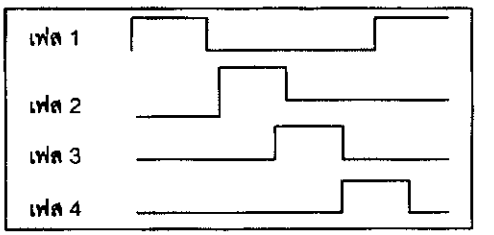
1. แบบฟูลสเต็ปหนึ่งเฟส (Full-Step 1 Phase)
2. แบบฟูลสเต็ปสองเฟส (Full-Step 2 Phase)
3. แบบฮาล์ฟสเต็ป (Half Step)

แบบฟูลสเต็ปหนึ่งเฟส (Full-Step 1 Phase)

การควบคุมลักษณะนี้จะเป็นการกระตุ้นขดลวดทีละขดเรียงตามลำดับ1, 2, 3, 4 การกระตุ้นจะมีขดลวดขดเดียวในเวลาหนึ่งที่ถูกกระตุ้นเท่านั้น เช่น ขดที่ 1, 2, 3, 4หรือถ้าต้องการให้หมุนสวนทิศทางกันก็จะกระตุ้นขดลวดที่ 4, 3, 2, 1 เป็นต้น การกระตุ้นแบบนี้จะทำงานง่ายที่สุดและกินกระแสไฟฟ้าน้อยที่สุดด้วย

สเต็ปที่	เฟสที่ 1	เฟสที่ 2	เฟสที่ 3	เฟสที่ 4
1	ทำงาน	-	-	-
2	-	ทำงาน	-	-
3	-	-	ทำงาน	-
4	-	-	-	ทำงาน

ก) Step Motor



ข) สัญญาณพัลส์ (Pulse)

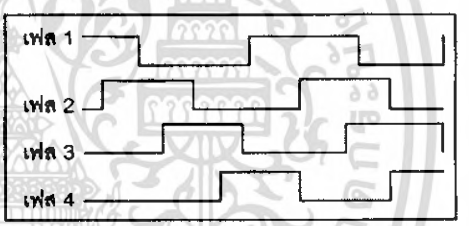
รูปที่ 2.32 แสดงการควบคุม Stepping Motor แบบ Full-Step 1 Phase

แบบฟูลสเต็ปสองเฟส (Full-Step 2 Phase)

การควบคุมลักษณะนี้จะเป็นการกระตุ้นขดลวดที่ละ 2 ขด ที่อยู่ใกล้กันให้ทำงานพร้อมกัน และเรียงลำดับกันไป เช่น ขดที่ 12,23,34,41 หรือถ้าต้องการให้หมุนสวนทิศทางกันก็จะกระตุ้นขดลวดที่ 12,23,34,41 เป็นต้น ข้อดีของการกระตุ้นแบบนี้ Stepping Motor จะมีแรงบิดมากกว่าแบบ 1 เฟส ส่วนข้อเสียของการกระตุ้นแบบนี้ จะต้องการใช้กำลังไฟฟ้าเพิ่มขึ้นเป็น 2 เท่าของแบบ 1 เฟส เพราะต้องกระตุ้นขดลวด 2 ขดพร้อมกัน

สเต็ปที่	เฟสที่ 1	เฟสที่ 2	เฟสที่ 3	เฟสที่ 4
1	ทำงาน	ทำงาน	-	-
2	-	ทำงาน	ทำงาน	-
3	-	-	ทำงาน	ทำงาน
4	ทำงาน	-	-	ทำงาน

ก) Step Motor



ข) สัญญาณพัลส์ (Pulse)

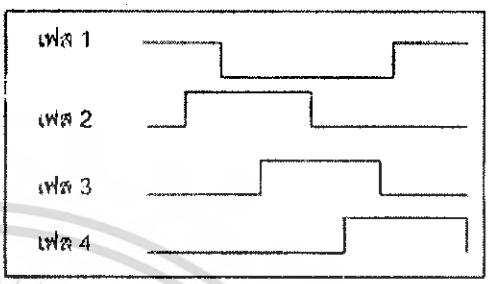
รูปที่ 2.33 แสดงการควบคุม Stepping Motor แบบ Full-Step 2 Phase

แบบฮาล์ฟสเต็ป (Half Step)

การควบคุมลักษณะนี้เป็นการกระตุ้นแบบ 1 เฟส กับ 2 เฟส มารวมกัน โดยการกระตุ้นจะเรียงลำดับกันไป เช่น ขดที่ 1,21,2,32,3,43,4,41 หรือถ้าต้องการให้หมุนสวนทางกัน ก็จะกระตุ้นขดลวด ที่ 1,21,2,32,3,43,4,41 เป็นต้น ข้อดีของการกระต้นแบบนี้ Stepping Motor จะมีแรงบิด เพิ่มมากขึ้น ละเอียดขึ้น และการควบคุมตำแหน่งถูกต้องมากยิ่งขึ้น ส่วนข้อเสียการกระตุ้นแบบนี้จะต้องใช้กำลังไฟฟ้าเพิ่มขึ้นเป็น 2 เท่า เหมือนกับการกระตุ้นแบบ 2 เฟสนั่นเอง

สเต็ปที่	เฟสที่ 1	เฟสที่ 2	เฟสที่ 3	เฟสที่ 4
1	ทำงาน	-	-	
2	ทำงาน	ทำงาน	-	
3	-	ทำงาน	-	
4	-	ทำงาน	ทำงาน	
	-	-	ทำงาน	
6	-	-	ทำงาน	ทำงาน
7	-	-	-	ทำงาน
8	ทำงาน	-	-	ทำงาน

ก) Step Motor

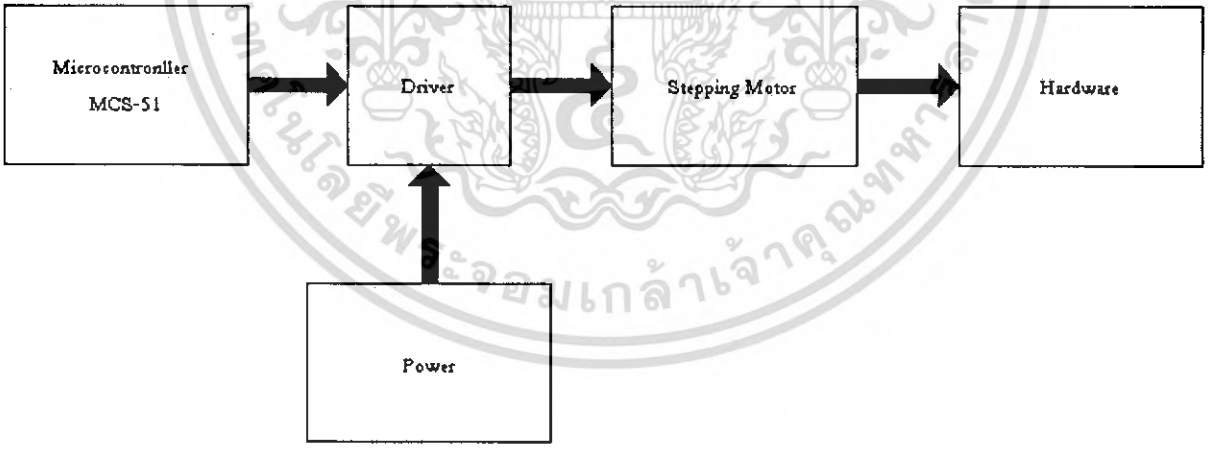


ข) สัญญาณพัลส์ (Pulse)

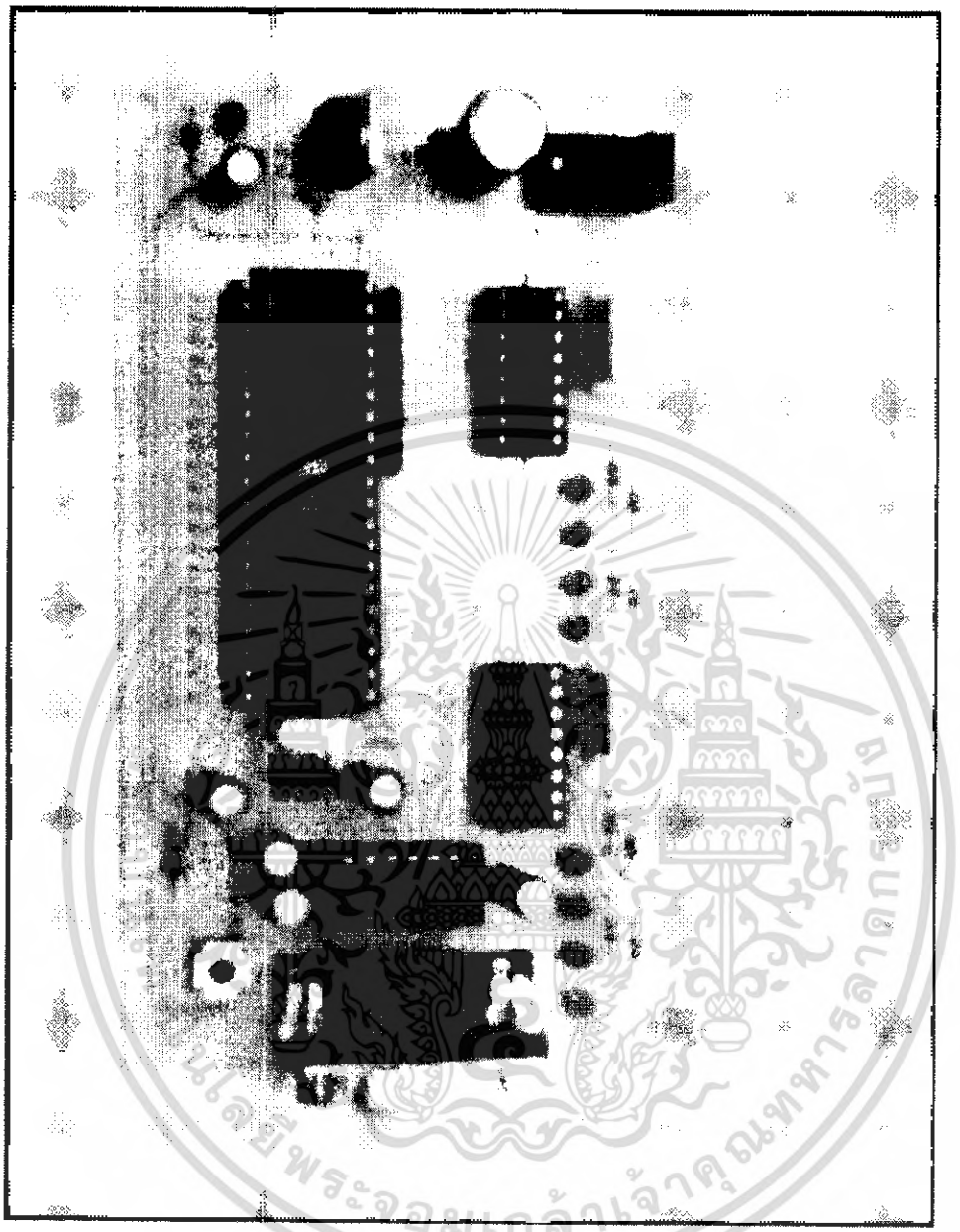
รูปที่ 2.34 แสดงการควบคุม Stepping Motor แบบ Half Step

2.15.3 การควบคุมการทำงานของ Stepping Motor

ในหน่วยนี้จะใช้ Microcontroller เป็นตัวควบคุมการทำงาน เพื่อให้ Stepping Motor ทำงานโดยจะผ่านชุดขับเพื่อเป็นตัวสร้างสัญญาณขับมอเตอร์ในแต่ละเฟส ให้ทำงานได้ตามต้องการ โดยสามารถให้มีการเคลื่อนที่เป็นองศาหรือระยะทาง การควบคุม Stepping Motor สามารถเขียนเป็นไดอะแกรมได้ดังนี้



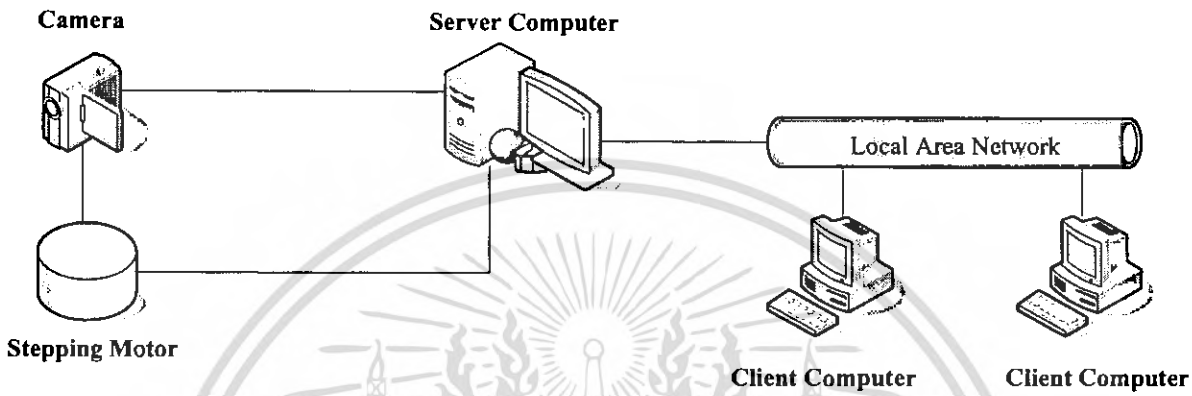
รูปที่ 2.35 แสดงไดอะแกรมการควบคุม Stepping Motor



รูปที่ 2.36 แสดงชุด Microcontroller ควบคุม Stepping Motor

บทที่ 3
การคำนวณและการสร้าง

กล้องรักษาความปลอดภัยผ่านเครือข่าย IP มีบล็อกไดอะแกรมทางภาคส่งดังนี้

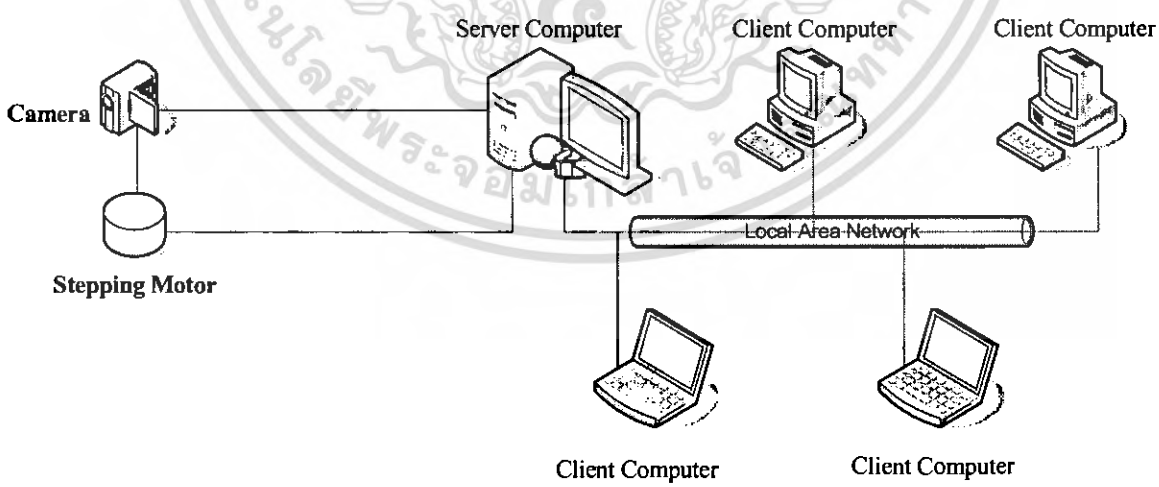


รูปที่ 3.1 บล็อกไดอะแกรมภาคส่ง

แบ่งการทำงานออกเป็นส่วนประกอบ 3 ส่วนคือ

1. วงจรสเต็ปปิงมอเตอร์ ที่ควบคุมโดย MCS-51 เพื่อทำหน้าที่เป็นฐานบังคับการหมุนของกล้อง
2. โปรแกรมด้าน Client
3. โปรแกรมของด้าน Server

3.1 ภาพรวมของระบบ



รูปที่ 3.2 โครงสร้างของระบบ

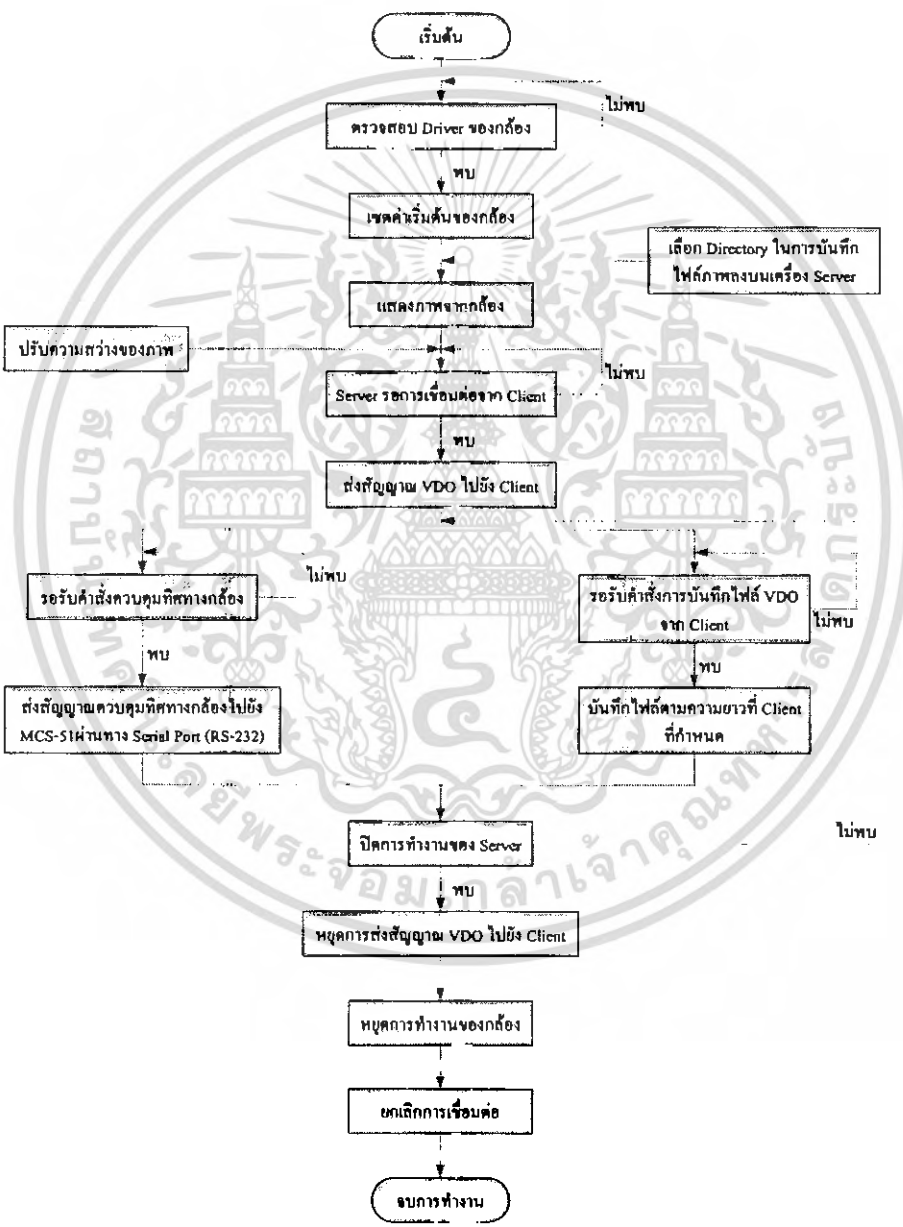
3.2 โปรแกรมส่วนติดต่อระหว่างเครือข่าย

โปรแกรมที่ออกแบบนั้นแบ่งออกเป็น 2 โปรแกรม ดังนี้

3.2.1 .โปรแกรม Server

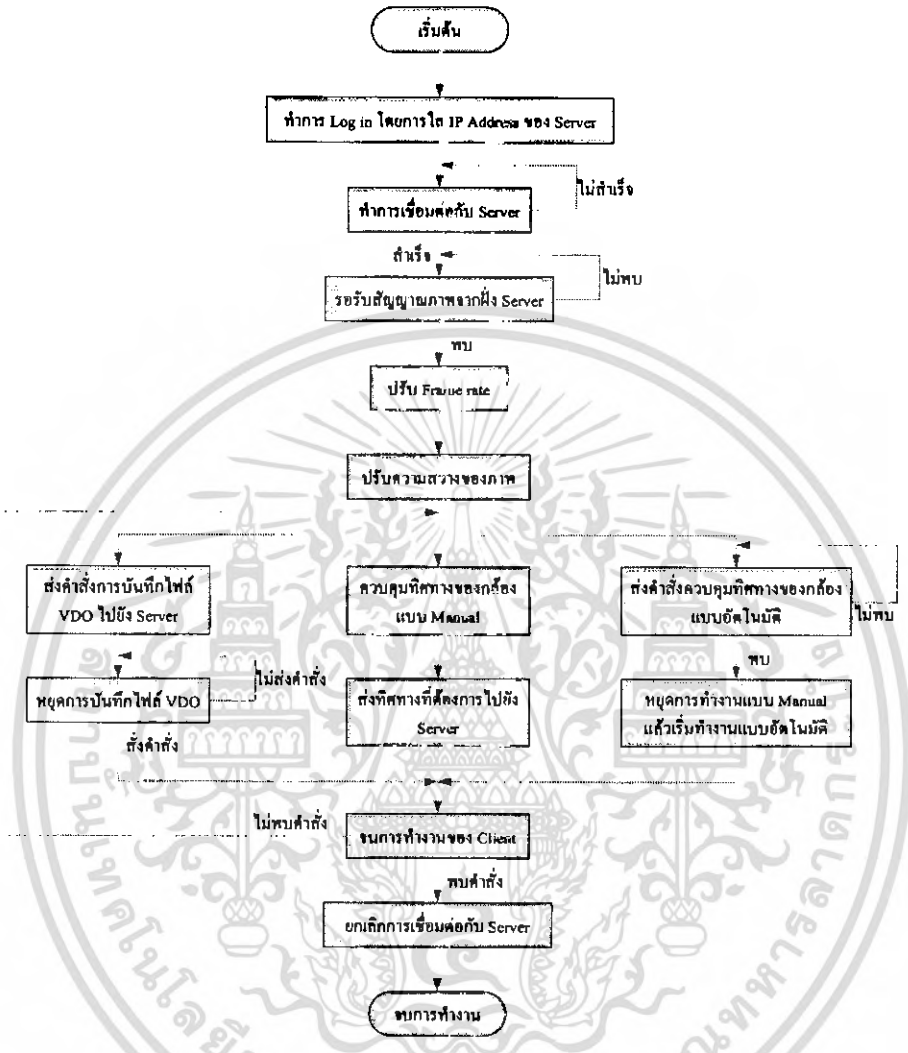
3.2.2 .โปรแกรม Client

3.2.1 โปรแกรม Server



รูปที่ 3.3 แผนภาพแสดงแผนผังการทำงานของโปรแกรมด้าน Server

3.2.2 โปรแกรม Client



รูปที่ 3.4 แผนภาพแสดงแผนผังการทำงานของโปรแกรมด้าน Client

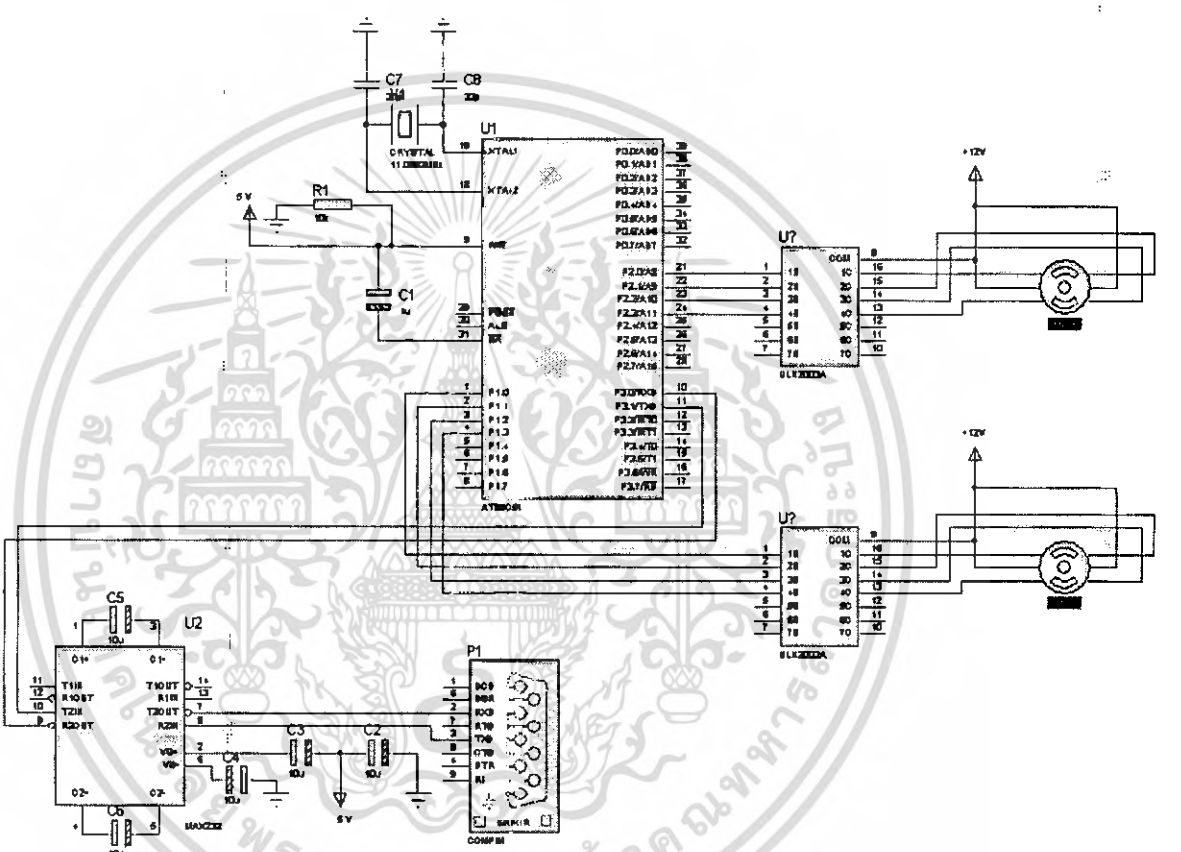
3.3 วงจรไมโครคอนโทรลเลอร์และวงจรขับกระแสสเต็ปปิงมอเตอร์

วงจรไมโครคอนโทรลเลอร์และวงจรขับกระแสสเต็ปปิงมอเตอร์นี้นั้นแบ่งออกเป็น 2 ตอน ดังนี้

3.2.1 .วงจรไมโครคอนโทรลเลอร์และวงจรขับกระแสสเต็ปปิงมอเตอร์

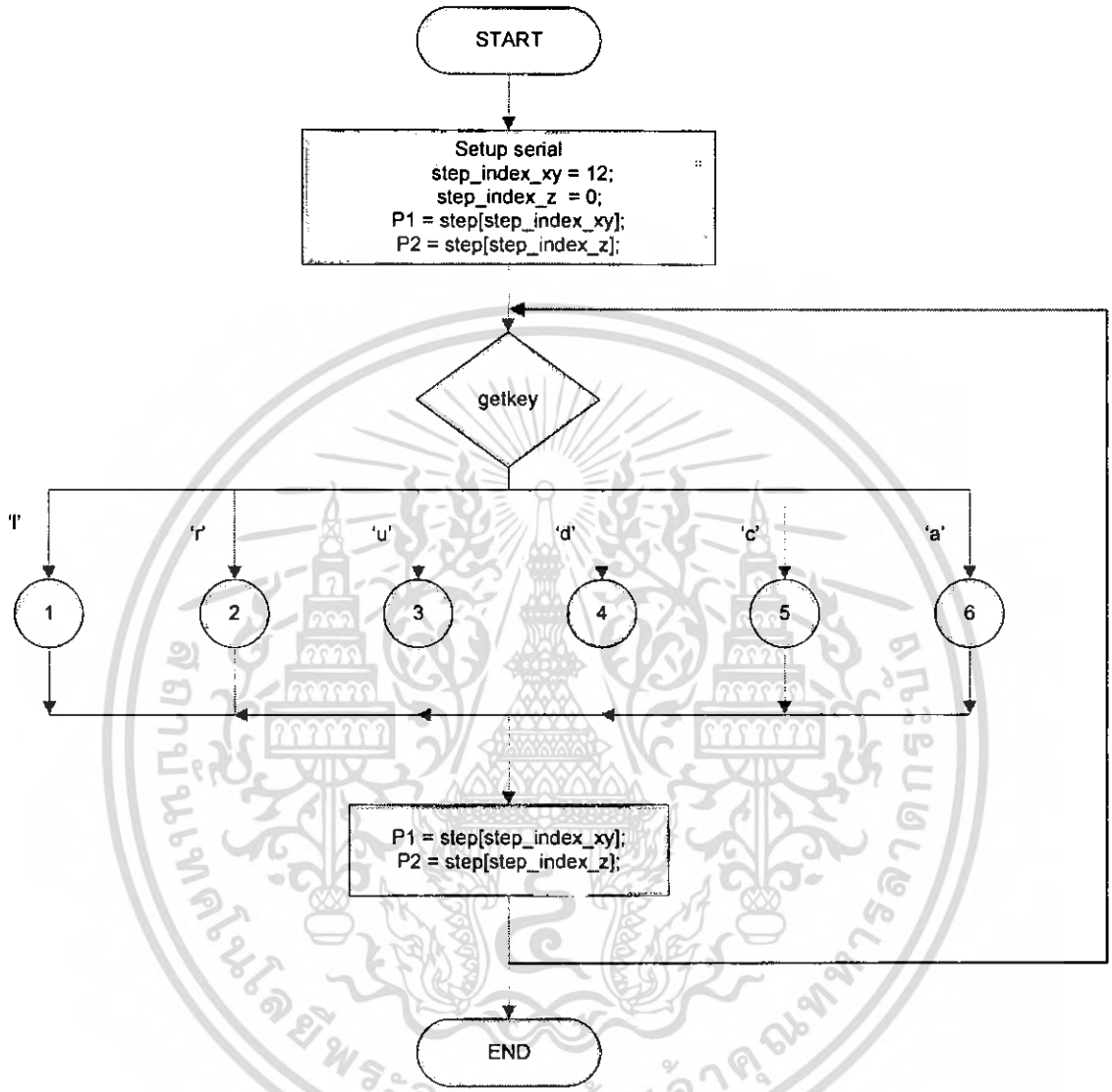
3.2.2 . โปรแกรมที่ใช้ควบคุมของกล้อง

3.2.1 วงจรไมโครคอนโทรลเลอร์และวงจรขับกระแสสเต็ปปิงมอเตอร์

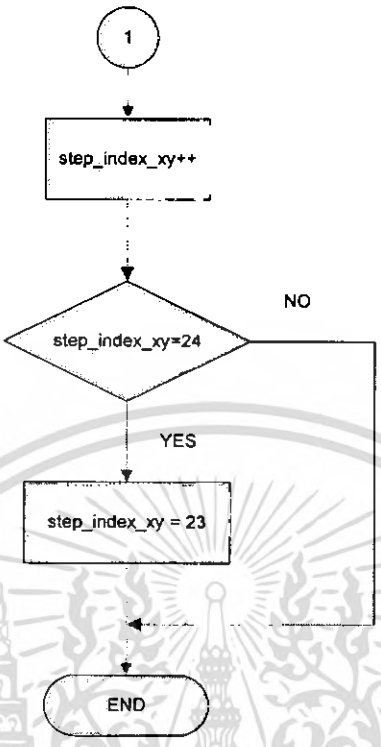


รูปที่ 3.5 วงจรไมโครคอนโทรลเลอร์และวงจรขับกระแสสเต็ปปิงมอเตอร์

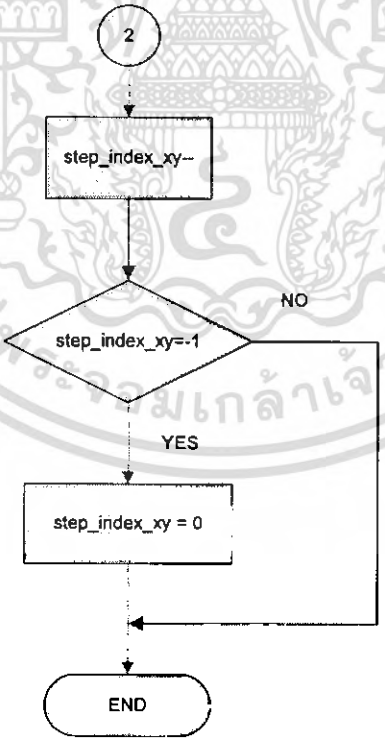
3.2.2 .โปรแกรมที่ใช้ควบคุมการหมุนของกล้อง



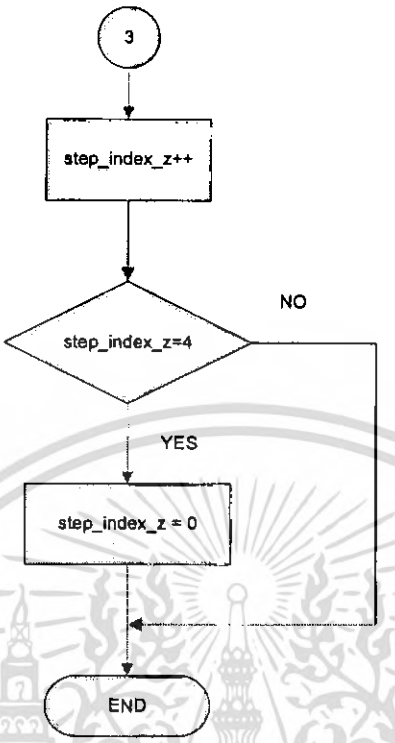
รูปที่ 3.6 แผนภาพแสดงแผนผังการทำงานของโปรแกรมควบคุมการหมุนของมอเตอร์



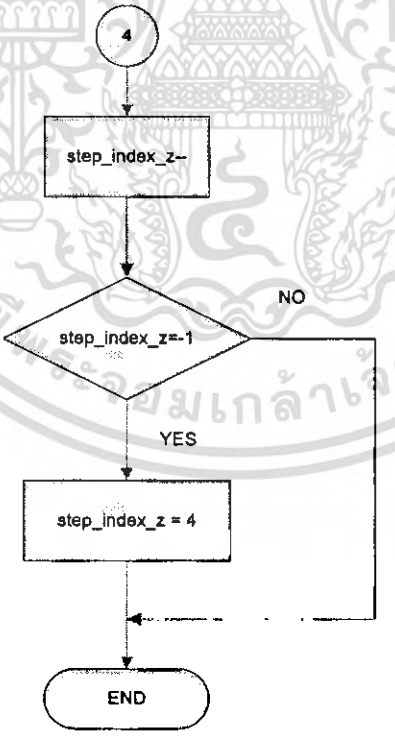
รูปที่ 3.7 แผนภาพแสดงแผนผังคำสั่งให้มอเตอร์หมุนซ้าย



รูปที่ 3.8 แผนภาพแสดงแผนผังคำสั่งให้มอเตอร์หมุนขวา

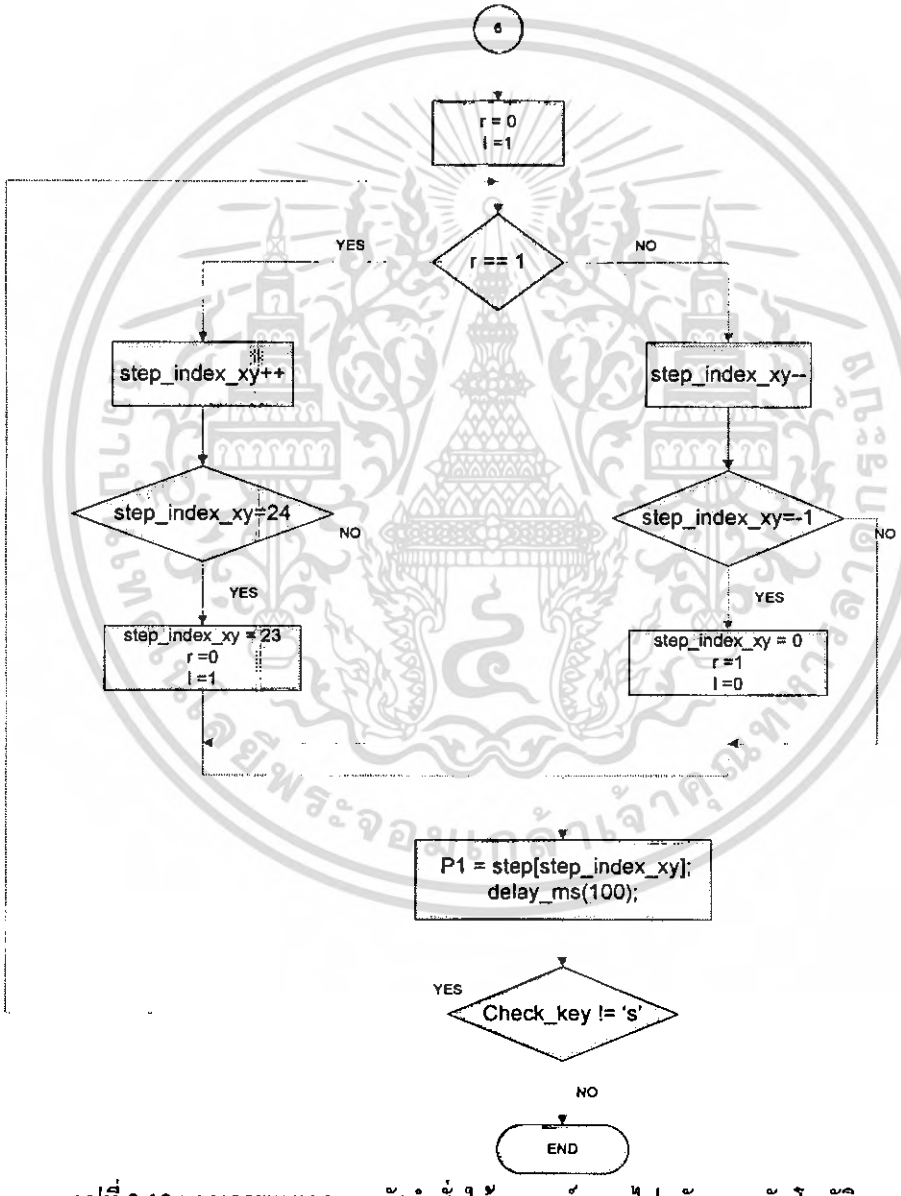


รูปที่ 3.9 แผนภาพแสดงแผนผังคำสั่งให้มอเตอร์หมุนขึ้น



รูปที่ 3.10 แผนภาพแสดงแผนผังคำสั่งให้มอเตอร์หมุนลง

รูปที่ 3.11 แผนภาพแสดงแผนผังคำสั่งให้ calibrate ตำแหน่งของมอเตอร์ในแกน XY



รูปที่ 3.12 แผนภาพแสดงแผนผังคำสั่งให้มอเตอร์หมุนไปกลับแบบอัตโนมัติ

บทที่ 4
การทดลองและผลการทดลอง

4.1 โปรแกรม Server - Client

วัตถุประสงค์

- ทดสอบการทำงานของโปรแกรมด้าน Server เชื่อมต่อกับด้าน Client

ขั้นตอนการทดลอง

4.1.1 ทำการรันโปรแกรม Server

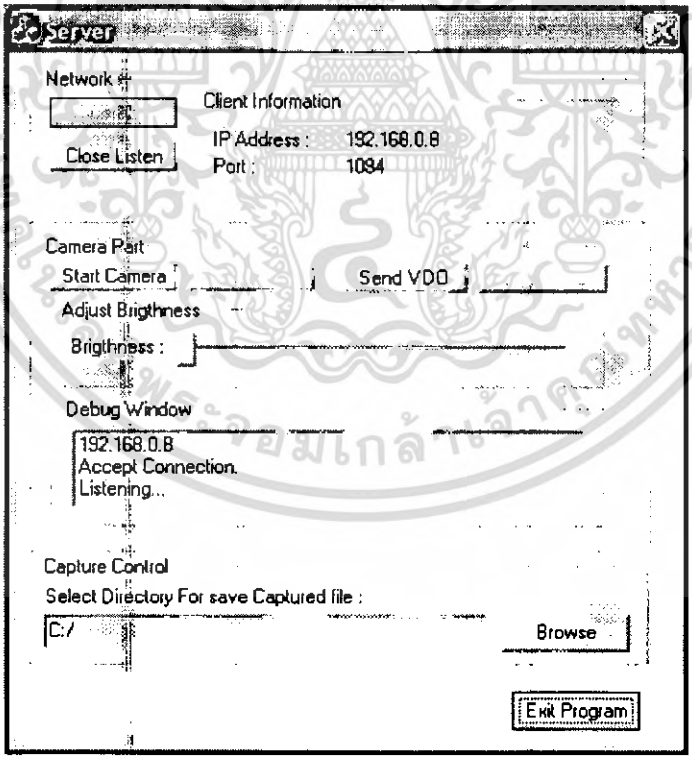
4.1.2 ทำการ listen รอรับการเชื่อมต่อจาก Client

4.1.3 ทำการรันโปรแกรม Client

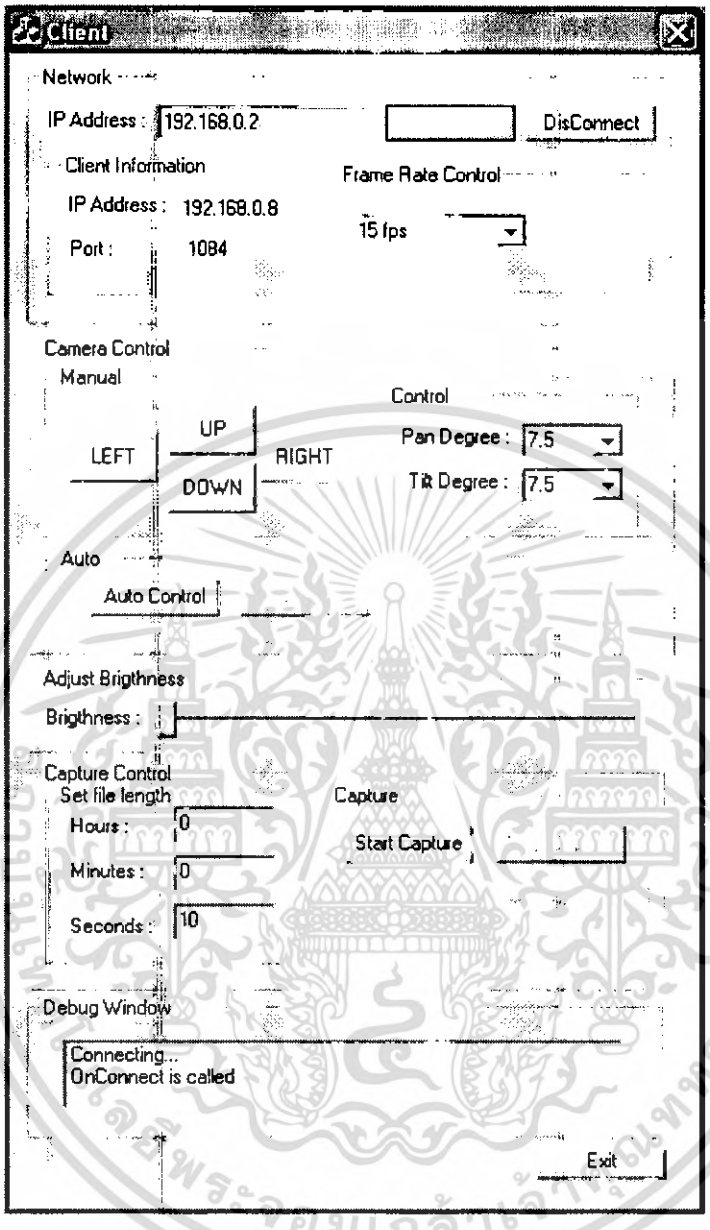
4.1.4 ทำการ connect ไปยัง Server

4.1.5 Server ทำการเริ่มต้นส่งสัญญาณวีดีโอไปยัง Client

ผลการทดลอง



รูปที่ 4.1 โปรแกรม Server



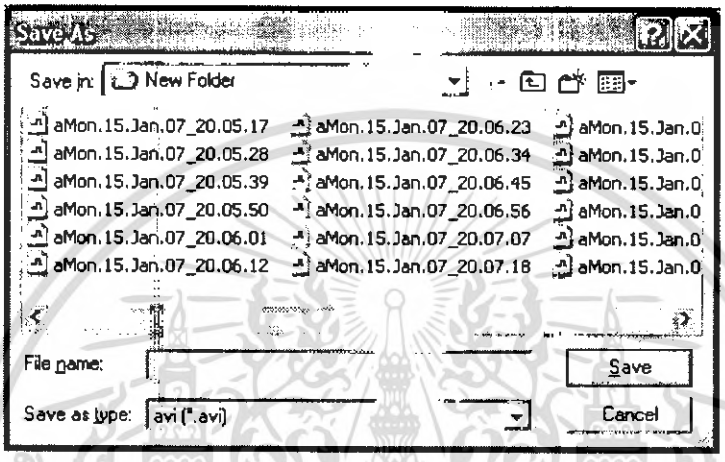
รูปที่ 4.2 โปรแกรม Client

4.2 การบันทึกไฟล์วีดีโอ

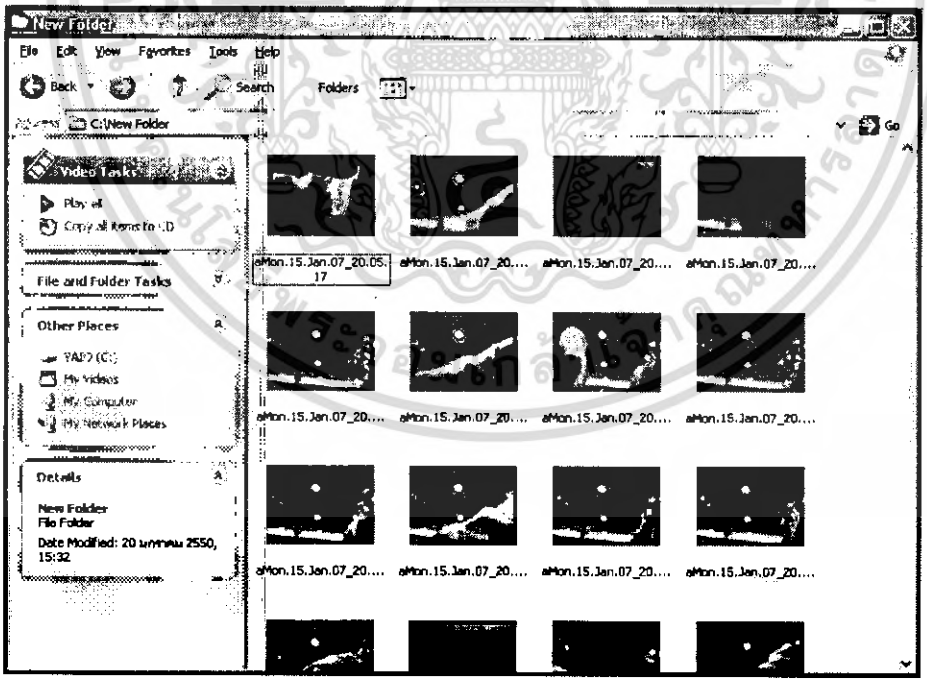
4.2.1 ทำการ browse ที่โปรแกรมด้าน server เพื่อเลือก save path

4.2.2 ทำการเริ่มบันทึกโดยส่งคำสั่งจากฝั่ง Client

ผลการทดลอง



รูปที่ 4.3 หน้าต่าง save file



รูปที่ 4.4 ไฟล์วีดีโอที่ได้จากการ save

4.3 การปรับความสว่าง (brightness)

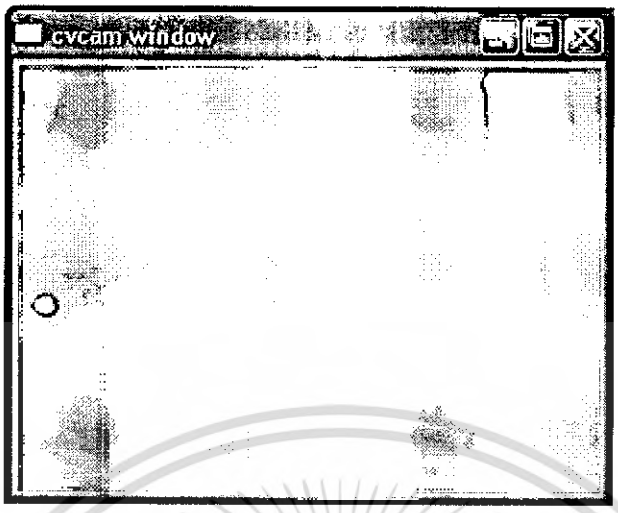
4.3.1 เมื่อโปรแกรม Client ได้รับข้อมูลวีดีโอจาก Server แล้วทำการปรับ brightness

4.3.2 โปรแกรม Server ทำการปรับ brightness ได้เช่นกัน

ผลการทดลอง

รูปที่ 4.5 ภาพในขณะก่อนทำการปรับความสว่าง

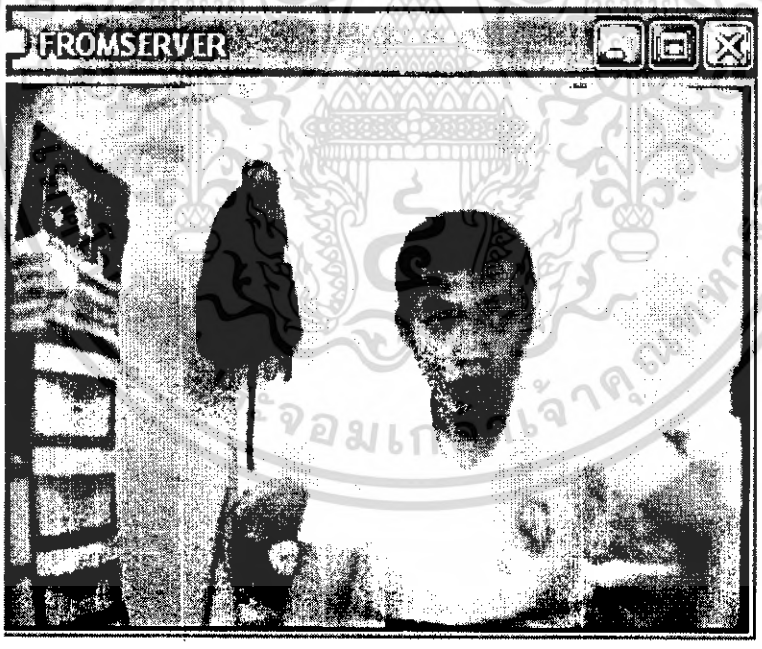
รูปที่ 4.6 ภาพเมื่อปรับค่าสว่างในระดับกลาง



รูปที่ 4.7 ภาพเมื่อทำการปรับความสว่างสูงสุด

4.4 การหมุนของกล้อง

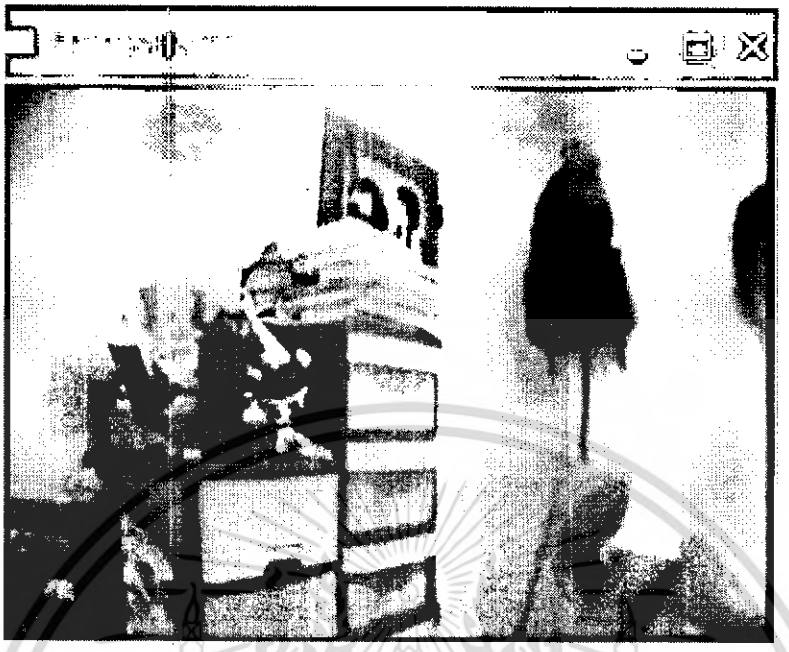
ผลการทดลอง



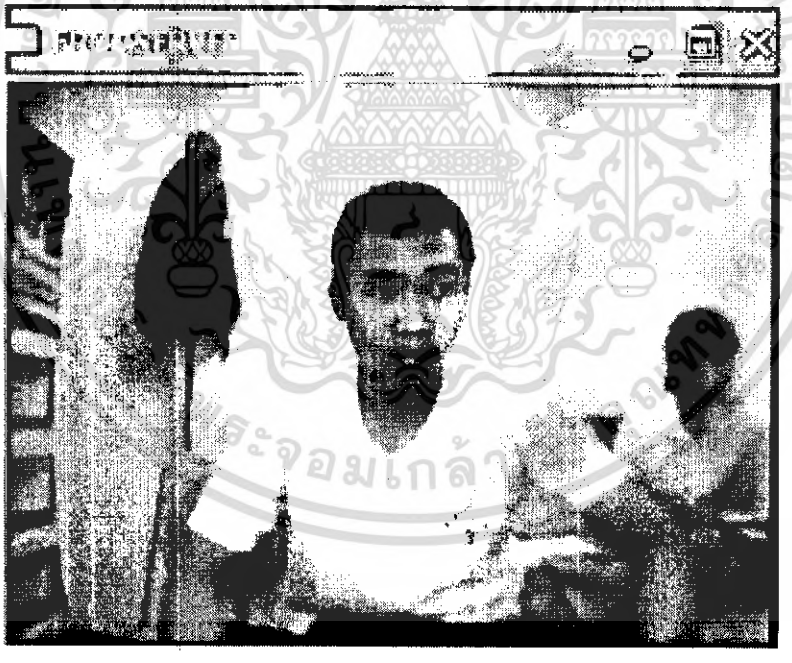
รูปที่ 4.8 ภาพขณะกล้องสถานะปกติ

รูปที่ 4.9 เมื่อทำการสั่งให้กล้องหมุนซ้ายไป 30 องศา

รูปที่ 4.10 เมื่อทำการสั่งให้กล้องหมุนซ้ายไป 60 องศา



รูปที่ 4.11 เมื่อทำการสั่งให้กล้องหมุนซ้ายไป 90 องศา

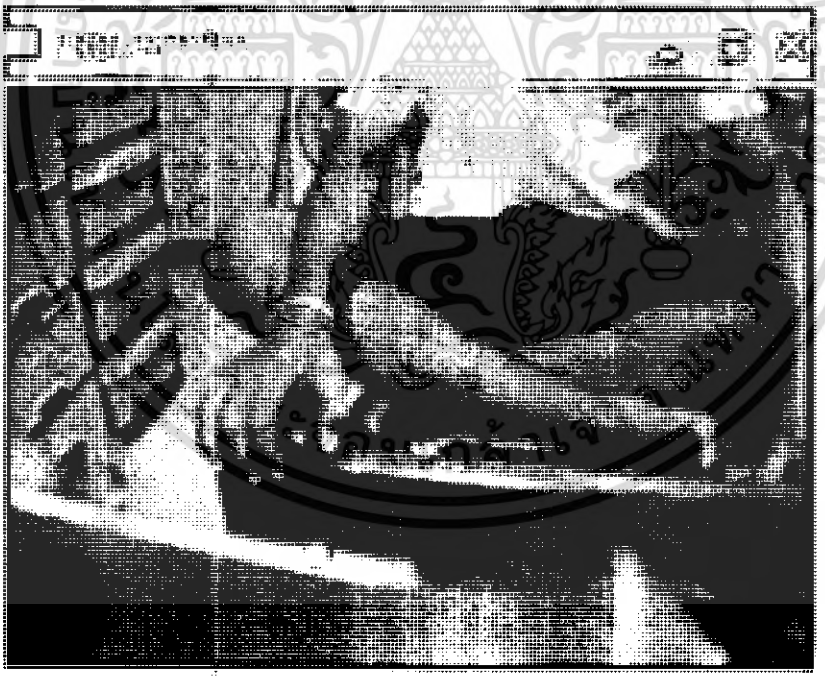


รูปที่ 4.12 เมื่อทำการสั่งให้กล้องหมุนขวาไป 30 องศา

รูปที่ 4.13 เมื่อทำการสั่งให้กล้องหมุนขวาไป 60 องศา

รูปที่ 4.14 เมื่อทำการสั่งให้กล้องหมุนขวาไป 90 องศา

รูปที่ 4.15 เมื่อทำการสั่งให้กล้องหมุนขึ้นไป 30 องศา



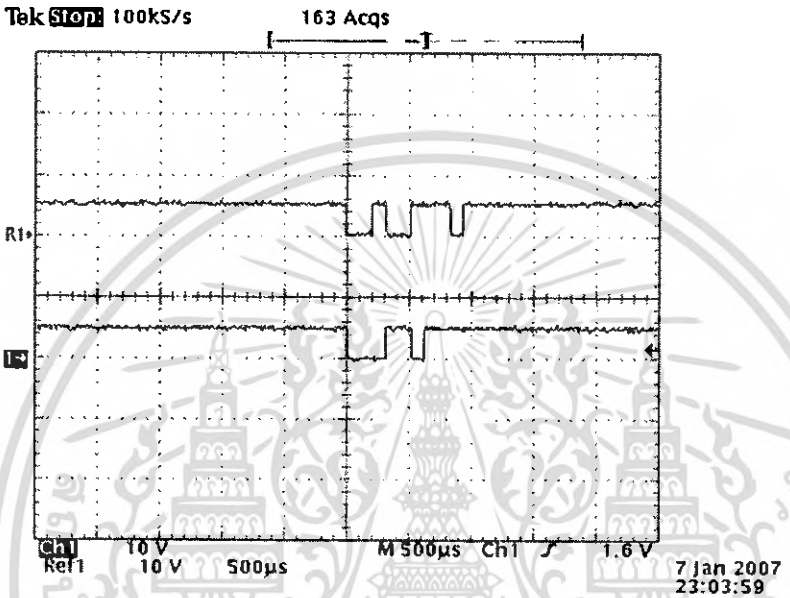
รูปที่ 4.16 เมื่อทำการสั่งให้กล้องหมุนลงไป 30 องศา

4.5 ทดสอบการส่งข้อมูลผ่านพอร์ตอนุกรม

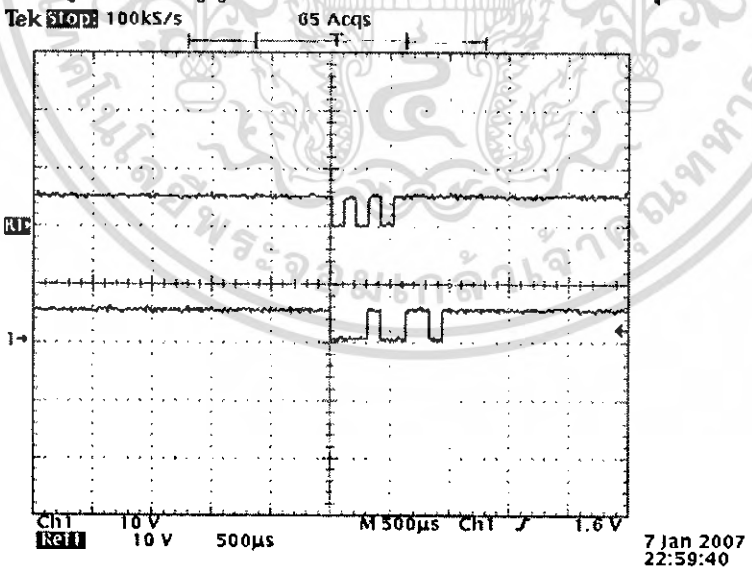
4.5.1 ส่งค่า (R, L, U, D) จากเครื่องคอมพิวเตอร์ไปยัง พอร์ตอนุกรม

4.5.2 ทำการวัดสัญญาณอินพุทที่เข้า MCS-51 ที่ขา RXD ของ MCS-51

ผลการทดลอง



รูปที่ 4.17 สัญญาณค่า R, L จาก server ที่ส่งไปยังพอร์ตอนุกรม



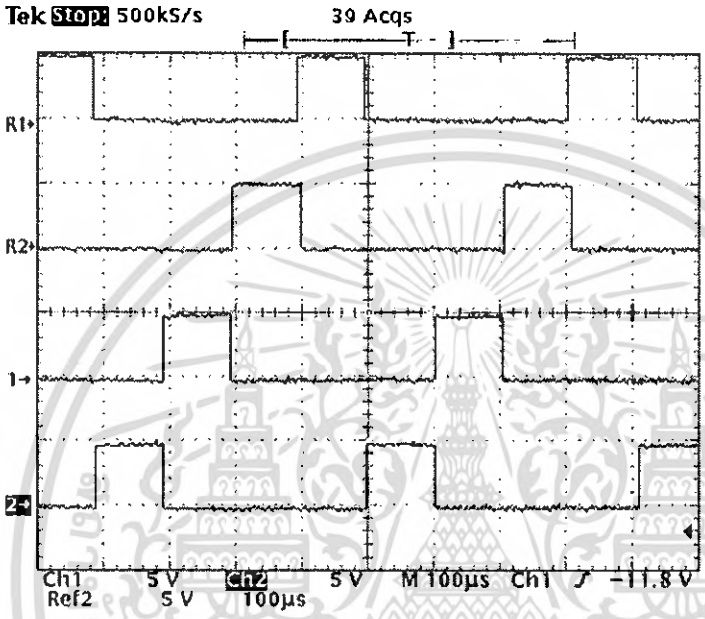
รูปที่ 4.18 สัญญาณค่า U, D จาก server ที่ส่งไปยังพอร์ตอนุกรม

4.6 ทดสอบการทำงานของ MCS-51

4.6.1 ทำการส่งสัญญาณคำสั่งไปยังพอร์ตอนุกรมแล้ว พอร์ตอนุกรมก็ทำการส่งสัญญาณไปยัง MCS- 51

4.6.2 ทำการวัดสัญญาณ output ที่ขาของ MCS-51

ผลการทดลอง



รูปที่ 4.19 แสดงการป้อนสัญญาณให้แต่เฟสของมอเตอร์ให้หมุนซ้าย

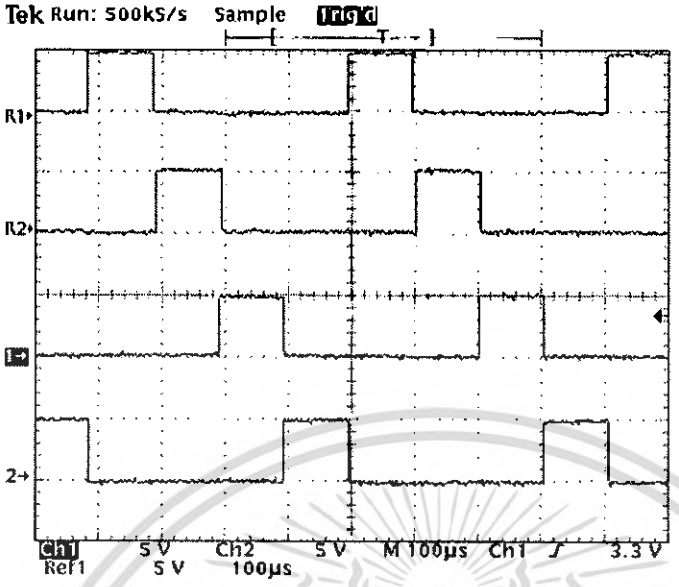
R1: แสดงรูปสัญญาณที่วัดได้จากพอร์ตที่ P1.0 ของ MCS-51

R2: แสดงรูปสัญญาณที่วัดได้จากพอร์ตที่ P1.1 ของ MCS-51

1: แสดงรูปสัญญาณที่วัดได้จากพอร์ตที่ P1.2 ของ MCS-51

2: แสดงรูปสัญญาณที่วัดได้จากพอร์ตที่ P1.3 ของ MCS-51

เมื่อเราป้อนคำสั่งให้มอเตอร์หมุนซ้าย แล้วทำการวัดสัญญาณเอาต์พุทที่พอร์ต 1 จะเห็นว่าสัญญาณที่ได้จะออกมาเป็นสัญญาณพัลส์สลับกันไปแต่ละเฟสของมอเตอร์ ดังแสดงในรูปที่ 4.19



รูปที่ 4.20 แสดงการป้อนสัญญาณให้แต่เฟสของมอเตอร์ให้หมุนขวา

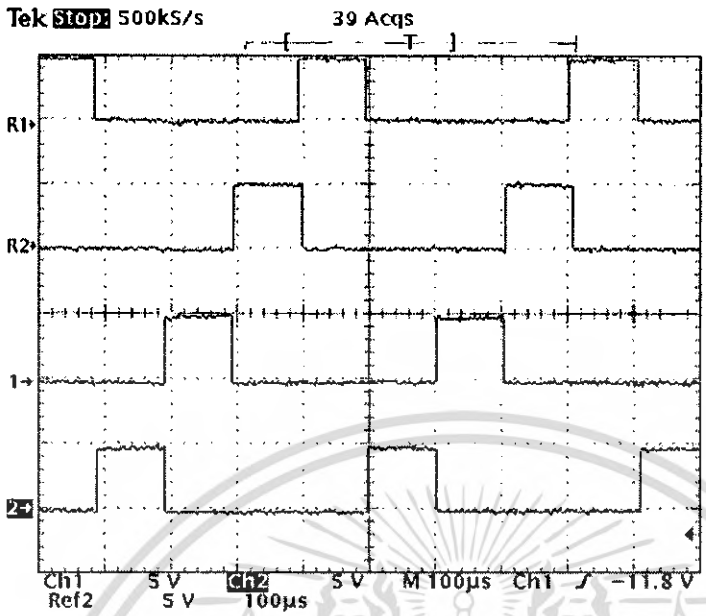
R1: แสดงรูปสัญญาณที่วัดได้จากพอร์ตที่ P1.0 ของ MCS-51

R2: แสดงรูปสัญญาณที่วัดได้จากพอร์ตที่ P1.1 ของ MCS-51

1: แสดงรูปสัญญาณที่วัดได้จากพอร์ตที่ P1.2 ของ MCS-51

2: แสดงรูปสัญญาณที่วัดได้จากพอร์ตที่ P1.3 ของ MCS-51

เมื่อเราป้อนคำสั่งให้มอเตอร์หมุนขวา แล้วทำการวัดสัญญาณเอาต์พุทที่พอร์ต 1 จะเห็นว่าสัญญาณที่ได้จะออกมาเป็นสัญญาณพัลส์สลับกันไปในแต่ละเฟสของมอเตอร์ โดยการสลับของพัลส์จะเป็นการสลับในทิศทางที่ตรงข้ามกันกับในกรณีที่เราป้อนคำสั่งให้มอเตอร์หมุนซ้าย ดังแสดงในรูปที่ 4.20



รูปที่ 4.21 แสดงการป้อนสัญญาณให้แต่เฟสของมอเตอร์ให้หมุนขึ้น

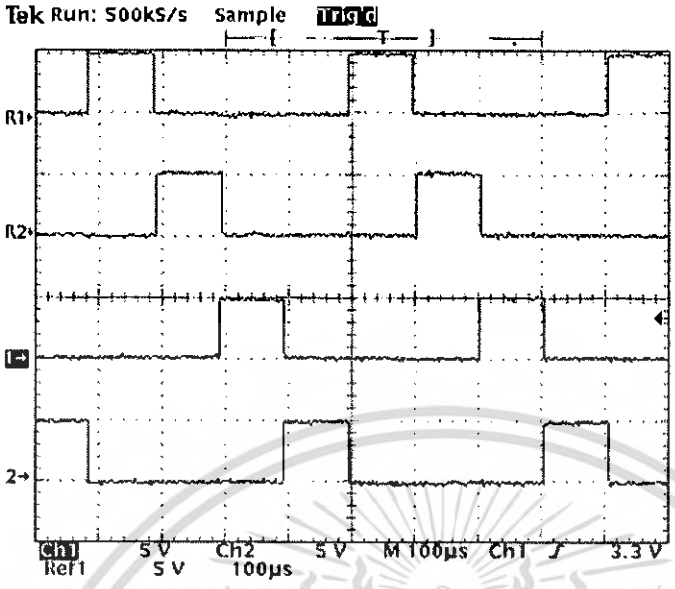
R1: แสดงรูปสัญญาณที่วัดได้จากพอร์ตที่ P2.0 ของ MCS-51

R2: แสดงรูปสัญญาณที่วัดได้จากพอร์ตที่ P2.1 ของ MCS-51

1: แสดงรูปสัญญาณที่วัดได้จากพอร์ตที่ P2.2 ของ MCS-51

2: แสดงรูปสัญญาณที่วัดได้จากพอร์ตที่ P2.3 ของ MCS-51

เมื่อเราป้อนคำสั่งให้มอเตอร์หมุนขึ้น แล้วทำการวัดสัญญาณเอาต์พุทที่พอร์ต 2 จะเห็นว่าสัญญาณที่ได้จะออกมาเป็นสัญญาณพัลส์สลับกันไปแต่ละเฟสของมอเตอร์ ดังแสดงในรูปที่ 4.21



รูปที่ 4.22 แสดงการป้อนสัญญาณให้แต่เฟสของมอเตอร์ให้หมุนลง

R1: แสดงรูปสัญญาณที่วัดได้จากพอร์ตที่ P2.0 ของ MCS-51

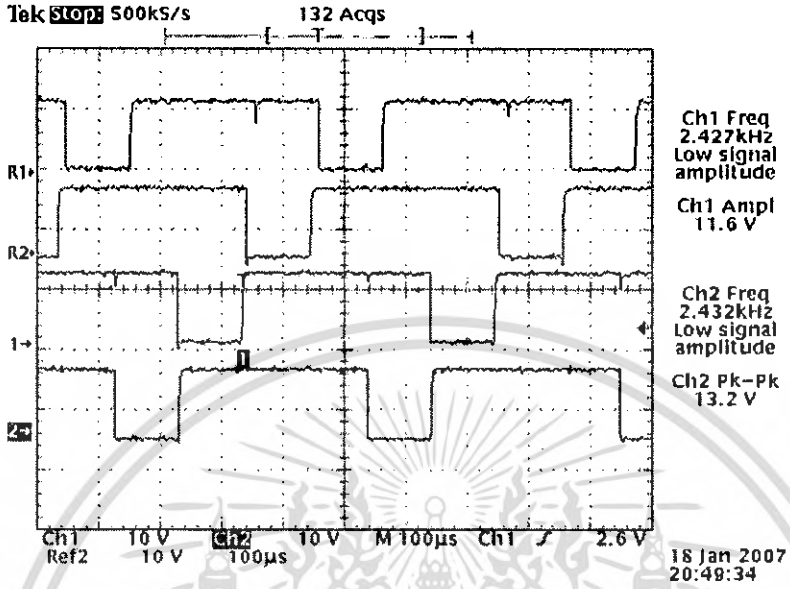
R2: แสดงรูปสัญญาณที่วัดได้จากพอร์ตที่ P2.1 ของ MCS-51

1: แสดงรูปสัญญาณที่วัดได้จากพอร์ตที่ P2.2 ของ MCS-51

2: แสดงรูปสัญญาณที่วัดได้จากพอร์ตที่ P2.3 ของ MCS-51

เมื่อเราป้อนคำสั่งให้มอเตอร์หมุนลง แล้วทำการวัดสัญญาณเอาต์พุทที่พอร์ต 2 จะเห็นว่าสัญญาณที่ได้จะออกมาเป็นสัญญาณพัลส์สลับกันไปในแต่ละเฟสของมอเตอร์ โดยการสลับของพัลส์จะเป็นการสลับในทิศทางที่ตรงข้ามกันกับในกรณีที่เราป้อนคำสั่งให้มอเตอร์หมุนขึ้น ดังแสดงในรูปที่ 4.22

4.6.3 ทำการทดลองวัดแรงดันที่เอาต์พุทของ ULN 2003 โดยสั่งให้มอเตอร์หมุนซ้าย



รูปที่ 4.23 สัญญาณเอาต์พุทจาก ULN2003

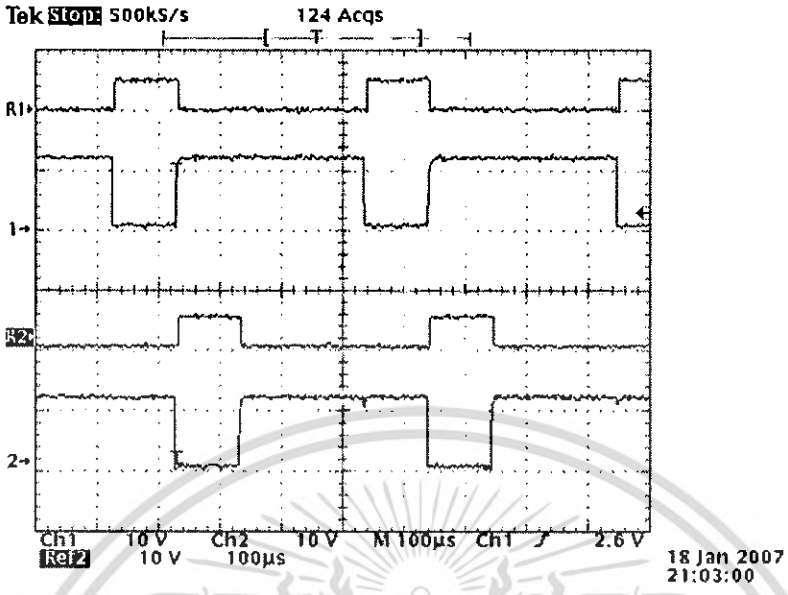
R1: แสดงรูปสัญญาณที่วัดได้จากขาที่ 16 ของ ULN2003

R2: แสดงรูปสัญญาณที่วัดได้จากขาที่ 15 ของ ULN2003

1: แสดงรูปสัญญาณที่วัดได้จากขาที่ 14 ของ ULN2003

2: แสดงรูปสัญญาณที่วัดได้จากขาที่ 13 ของ ULN2003

เมื่อป้อนคำสั่งให้มอเตอร์หมุนซ้ายสัญญาณที่ได้จะมีการกลับเฟสจากสัญญาณอินพุตที่ป้อนให้ ULN 2003และแรงดันเพิ่มขึ้นเป็น 12 V ดังแสดงในรูปที่ 4.23



รูปที่ 4.24 สัญญาณเอาต์พุทจาก MCS เทียบกับ สัญญาณเอาต์พุทจาก ULN2003

R1: แสดงรูปสัญญาณที่วัดได้จากพอร์ตที่ P1.0 ของ MCS-51

1: แสดงรูปสัญญาณที่วัดได้จากขาที่ 16 ของ ULN2003

R2: แสดงรูปสัญญาณที่วัดได้จากพอร์ตที่ P1.1 ของ MCS-51

2: แสดงรูปสัญญาณที่วัดได้จากขาที่ 15 ของ ULN2003

บทที่ 5
บทสรุปและบทวิจารณ์

5.1 สรุปผลการทดลอง

โครงงานนี้เป็นการสร้างระบบกล้องรักษาความปลอดภัย ซึ่งสามารถใช้สังเกตการณ์ในบริเวณต่างๆที่ต้องการได้ ผ่านเครือข่ายอินเทอร์เน็ต โดยส่วนฮาร์ดแวร์ ในส่วนของวงจรสเต็ปปิ้งมอเตอร์ ที่ควบคุมโดย MCS-51 เพื่อทำหน้าที่เป็นฐานบังคับการหมุนของกล้อง ที่ต่ออยู่ก่อนส่งสัญญาณต่อไปยังคอมพิวเตอร์ โดยสามารถทำการหมุนกล้องได้180 องศา และ ทำงานทั้งในโหมดธรรมดาและ ในแบบอัตโนมัติ

- การทำการสื่อสารกันระหว่าง เครื่องคอมพิวเตอร์ฝั่ง server และ ฝั่ง client นั้นสามารถทำได้ตามวัตถุประสงค์
- การส่งข้อมูลภาพจากฝั่ง server ไปยัง client นั้นทำได้ตามวัตถุประสงค์
- เครื่อง server และ client สามารถปรับค่าความสว่างของภาพได้
- เครื่อง client สามารถปรับอัตราการส่งเฟรมของเครื่องserverได้ทำให้ข้อมูลภาพที่ได้รับมีความต่อเนื่องยิ่งขึ้น
- สามารถบันทึกข้อมูลภาพ เป็นไฟล์วีดีโอได้โดยเป็นไฟล์ .avi มีการระบุ วัน เวลาไว้โดยละเอียดทำให้เกิดความสะดวก เวลาต้องการเปิดไฟล์ที่บันทึกขึ้นมาตรวจสอบ

5.2 วิจารณ์ผลการทดลอง

- จากการทดลอง เครื่องคอมพิวเตอร์ server และ client สามารถสื่อสารกันได้ราบรื่น
- การปรับอัตราการส่งเฟรมของข้อมูลภาพนั้นทำได้แต่ประสบปัญหาจากความล่าช้าของเครือข่ายบ้าง ทำให้ข้อมูลภาพที่ได้นั้นอาจมีการกระตุกได้ในบางครั้ง
- สามารถทำการปรับค่าความสว่างของภาพได้
- การควบคุมการหมุนของกล้องนั้นทำได้ทั้งในแบบธรรมดาและในแบบอัตโนมัติ
- การบันทึกไฟล์วีดีโอนั้นทำได้อย่างถูกต้องตามวัตถุประสงค์และสามารถเปิดดูได้ในภายหลัง
- เนื่องจากในส่วนของฮาร์ดแวร์ไม่มีเซนเซอร์ในการตรวจสอบให้ตัวกล้องหมุนกับโดยอัตโนมัติ แต่ทำโดยการให้โปรแกรมจำตำแหน่งในการหมุนของกล้องแต่ละสเต็ปไว้ทำให้เมื่อเวลาใช้งานเริ่มต้นแต่ละครั้งต้องทำการกำหนดตำแหน่งกึ่งกลางของกล้องไว้ทุกครั้ง

ภาคผนวก

โปรแกรมส่วนติดต่อระหว่างเครือข่าย

-โปรแกรม Server

```
//////////////////////////////////////////////////////////////////////////
////// CMySocket member functions
void CMySocket::OnAccept(int nErrorCode)
{
      if (nErrorCode == 0)
      ((CSocket002Dlg *)maindlg)->OnAccept();
      CAsyncSocket::OnAccept(nErrorCode);
}
void CMySocket::OnClose(int nErrorCode)
{
      if (nErrorCode == 0)
      ((CSocket002Dlg *)maindlg)->OnClose();
            CAsyncSocket::OnClose(nErrorCode);
}
void CMySocket::OnConnect(int nErrorCode)
{
      if (nErrorCode == 0)
      ((CSocket002Dlg *)maindlg)->OnConnect();
      CAsyncSocket::OnConnect(nErrorCode);
}
void CMySocket::OnSend(int nErrorCode)
{
      if (nErrorCode == 0)
      ((CSocket002Dlg *)maindlg)->OnSend();
      CAsyncSocket::OnSend(nErrorCode);
}
void CMySocket::OnReceive(int nErrorCode)
{
      if (nErrorCode == 0)
      ((CSocket002Dlg *)maindlg)->OnReceive();
      CAsyncSocket::OnReceive(nErrorCode);
}

//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by Socket002.rc
//
#define IDM_ABOUTBOX                    0x0010
#define IDD_ABOUTBOX                    100
#define IDS_ABOUTBOX                    101
#define IDD_SOCKET002_DIALOG            102
#define IDR_MAINFRAME                   128
#define IDC_LIST1                       1000
#define IDC_EDIT1                       1001
#define IDC_BUTTON1                     1002
#define IDC_BUTTON2                     1003
#define IDC_BUTTON3                     1004
#define IDC_BUTTON4                     1005
#define IDC_BUTTON7                     1008
#define IDC_SLIDER1                     1009
#define IDC_BUTTON5                     1010
#define IDC_SLIDER2                     1011
#define IDC_STATIC1                     1012
#define IDC_STATIC2                     1013
#define IDC_BUTTON8                     1015
#define IDC_BUTTON9                     1016
#define IDC_BUTTON6                     1019
// Next default values for new objects
```

```
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE          129
#define _APS_NEXT_COMMAND_VALUE           32771
#define _APS_NEXT_CONTROL_VALUE           1020
#define _APS_NEXT_SYMED_VALUE             101
#endif
#endif

// Socket002.h : main header file for the SOCKET002 application
//
#if
!defined(AFX_SOCKET002_H__5962AAFA_B93B_4C98_A6C9_D7504671CCC7__INCLUDED_
)
#define AFX_SOCKET002_H__5962AAFA_B93B_4C98_A6C9_D7504671CCC7__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
#ifndef __AFXWIN_H__
#error include 'stdafx.h' before including this file for PCH
#endif
#include "resource.h"       // main symbols
/////////////////////////////////////////////////////////////////////////////
////
// CSocket002App:
// See Socket002.cpp for the implementation of this class
//
class CSocket002App : public CWinApp
{
public:
      CSocket002App();
// Overrides
      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CSocket002App)
      public:
      virtual BOOL InitInstance();
      //}}AFX_VIRTUAL
// Implementation
//{{AFX_MSG(CSocket002App)
            // NOTE - the ClassWizard will add and remove member
functions here.
            //    DO NOT EDIT what you see in these blocks of generated
code !
      //}}AFX_MSG
      DECLARE_MESSAGE_MAP()
};
/////////////////////////////////////////////////////////////////////////////
////
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.
#endif //
!defined(AFX_SOCKET002_H__5962AAFA_B93B_4C98_A6C9_D7504671CCC7__INCLUDED_
)

// Socket002Dlg.h : header file
//
#if
!defined(AFX_SOCKET002DLG_H__1162EDEC_DD8F_4FD1_AC68_06FE3FD33B68__INCLUD
ED_)
```

```
#define
AFX_SOCKET002DLG_H__1162EDEC_DD8F_4FD1_AC68_06FE3FD33B68__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
#include "MySocket.h"
#include "cv.h"
#include "highgui.h"
#include "cxcore.h"
#include "cvcam.h"
#include <time.h>
////////////////////////////////////////////////////////////////////////////
////
// CSocket002Dlg dialog
class CSocket002Dlg : public CDialog
{
	// Construction
public:
	CSocket002Dlg(CWnd* pParent = NULL);	// standard constructor
	CMySocket server,client;
	void OnAccept();
	void OnConnect();
	void OnClose();
	void OnSend();
	void OnReceive();
// Dialog Data
	//{{AFX_DATA(CSocket002Dlg)
	enum { IDD = IDD_SOCKET002_DIALOG };
	CSliderCtrl m_Slide1;
	CListBox	m_message;
	CString	m_ipcl;
	CString	m_portcl;
	CString	m_dir;
	//}}AFX_DATA
// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CSocket002Dlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV
support
	//}}AFX_VIRTUAL
// Implementation
protected:
	HICON m_hIcon;
// Generated message map functions
	//{{AFX_MSG(CSocket002Dlg)
	virtual BOOL OnInitDialog();
	afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
	afx_msg void OnPaint();
	afx_msg HCURSOR OnQueryDragIcon();
	afx_msg void OnStart();
	virtual void OnCancel();
	afx_msg void OnSendData();
	afx_msg void OnCloseListen();
	afx_msg void OnCheckCamera();
	afx_msg void OnStopCamera();
	afx_msg void OnReleasedcaptureSlider1(NMHDR* pNMHDR, LRESULT*
pResult);
	afx_msg void OnStopSend();
	afx_msg void OnBrowse();
	afx_msg void OnButton6();
	//}}AFX_MSG
```

```
	DECLARE_MESSAGE_MAP()
};
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.
#endif //
!defined(AFX_SOCKET002DLG_H__1162EDEC_DD8F_4FD1_AC68_06FE3FD33B68__INCLUD
ED_)

// stdafx.h : include file for standard system include files,
//  or project specific include files that are used frequently, but
//      are changed infrequently
//
#if
!defined(AFX_STDAFX_H__9918A076_C2AD_4724_B5BB_3700602AAEBC__INCLUDED_)
#define AFX_STDAFX_H__9918A076_C2AD_4724_B5BB_3700602AAEBC__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
#define VC_EXTRALEAN		// Exclude rarely-used stuff from Windows
headers
#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions
#include <afxdisp.h>        // MFC Automation classes
#include <afxdtctl.h>		// MFC support for Internet Explorer 4
Common Controls
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>			// MFC support for Windows Common
Controls
#endif // _AFX_NO_AFXCMN_SUPPORT
#include <afxsock.h>
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.
#endif //
!defined(AFX_STDAFX_H__9918A076_C2AD_4724_B5BB_3700602AAEBC__INCLUDED_)

// MySocket.cpp : implementation file
//
#include "stdafx.h"
#include "Socket002.h"
#include "MySocket.h"
#include "Socket002Dlg.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
//////////////////////////////////////////////////////////////////////////
////
// CMySocket
CMySocket::CMySocket()
{
}
CMySocket::~CMySocket()
{
}
// Do not edit the following lines, which are needed by ClassWizard.
#if 0
BEGIN_MESSAGE_MAP(CMySocket, CAsyncSocket)
	//{{AFX_MSG_MAP(CMySocket)
```

```
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()
#endif	// 0
/////////////////////////////////////////////////////////////////////////////
////
// CMySocket member functions
void CMySocket::OnAccept(int nErrorCode)
{
	// TODO: Add your specialized code here and/or call the base class
	if (nErrorCode == 0)
	((CSocket002Dlg *)maindlg)->OnAccept();
	CAsyncSocket::OnAccept(nErrorCode);
}
void CMySocket::OnClose(int nErrorCode)
{
	// TODO: Add your specialized code here and/or call the base class
	if (nErrorCode == 0)
	((CSocket002Dlg *)maindlg)->OnClose();
	CAsyncSocket::OnClose(nErrorCode);
}
void CMySocket::OnConnect(int nErrorCode)
{
	// TODO: Add your specialized code here and/or call the base class
	if (nErrorCode == 0)
	((CSocket002Dlg *)maindlg)->OnConnect();
	CAsyncSocket::OnConnect(nErrorCode);
}
void CMySocket::OnSend(int nErrorCode)
{
	// TODO: Add your specialized code here and/or call the base class
	if (nErrorCode == 0)
	((CSocket002Dlg *)maindlg)->OnSend();
	CAsyncSocket::OnSend(nErrorCode);
}
void CMySocket::OnReceive(int nErrorCode)
{
	// TODO: Add your specialized code here and/or call the base class
	if (nErrorCode == 0)
	((CSocket002Dlg *)maindlg)->OnReceive();
	CAsyncSocket::OnReceive(nErrorCode);
}

// Socket002.cpp : Defines the class behaviors for the application.
//
#include "stdafx.h"
#include "Socket002.h"
#include "Socket002Dlg.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
/////////////////////////////////////////////////////////////////////////////
////
// CSocket002App
BEGIN_MESSAGE_MAP(CSocket002App, CWinApp)
	//{{AFX_MSG_MAP(CSocket002App)
		// NOTE - the ClassWizard will add and remove mapping macros
here.
		//    DO NOT EDIT what you see in these blocks of generated
code!
```

```
	//}}AFX_MSG
	ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()
////////////////////////////////////////////////////////////////////////////
////
// CSocket002App construction
CSocket002App::CSocket002App()
{
	// TODO: add construction code here,
	// Place all significant initialization in InitInstance
}
////////////////////////////////////////////////////////////////////////////
////
// The one and only CSocket002App object
CSocket002App theApp;
////////////////////////////////////////////////////////////////////////////
////
// CSocket002App initialization
BOOL CSocket002App::InitInstance()
{
	if(!AfxSocketInit())
	{
		return FALSE;
	}
	AfxEnableControlContainer();
// Standard initialization
	// If you are not using these features and wish to reduce the size
	//  of your final executable, you should remove from the following
	//  the specific initialization routines you do not need.
#ifdef _AFXDLL
	Enable3dControls();			// Call this when using MFC in
a shared DLL
#else
	Enable3dControlsStatic();	// Call this when linking to MFC
statically
#endif
CSocket002Dlg dlg;
	m_pMainWnd = &dlg;
	int nResponse = dlg.DoModal();
	if (nResponse == IDOK)
	{
		// TODO: Place code here to handle when the dialog is
		//  dismissed with OK
	}
	else if (nResponse == IDCANCEL)
	{
		// TODO: Place code here to handle when the dialog is
		//  dismissed with Cancel
	}
// Since the dialog has been closed, return FALSE so that we exit the
	//  application, rather than start the application's message pump.
	return FALSE;
}

// Socket002Dlg.cpp : implementation file
//
#include "stdafx.h"
#include "Socket002.h"
#include "Socket002Dlg.h"
#ifdef _DEBUG
#define new DEBUG_NEW
```

```
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
//////////////////////////////////////////////////////////////////////////////
////
// CAboutDlg dialog used for App About
unsigned int global_br;
unsigned int global_ct;
int frame_delay;
char * Rec_ptr;
bool start_status;
UINT MyThreadFunct(LPVOID pParam);
void PreviewCallback(IplImage* previm);
IplImage *vdo;
IplImage *ycbcr;
//////////////
CvVideoWriter *writer;
unsigned StartCap=0;
int stat = 0;          // chronometer status: 0 = off, 1 = on
FILE *ptr;
time_t start_time, cur_time ,rawtime;
char filename[512];
char filedir[256];
long limit_time = 10;
struct tm * timeinfo;
char buffer [80];
CString     l_strFileName;
////////////////
// Serial I/F
HANDLE hComp;
DCB dcb;
char * chCommPort = "COM1";
int valReturn;
char chBuffer;
unsigned long nReadPort;
unsigned long nWrittenPort;
//////////////////////////
class CAboutDlg : public CDialog
{
public:
      CAboutDlg();
// Dialog Data
      //{{AFX_DATA(CAboutDlg)
      enum { IDD = IDD_ABOUTBOX };
      //}}AFX_DATA
// ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CAboutDlg)
      protected:
      virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
      //}}AFX_VIRTUAL
// Implementation
protected:
      //{{AFX_MSG(CAboutDlg)
      //}}AFX_MSG
      DECLARE_MESSAGE_MAP()
};
CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
      //{{AFX_DATA_INIT(CAboutDlg)
      //}}AFX_DATA_INIT
```

```
}
void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}
BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
		// No message handlers
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()
////////////////////////////////////////////////////////////////////////////
////
// CSocket002Dlg dialog
CSocket002Dlg::CSocket002Dlg(CWnd* pParent /*=NULL*/)
	: CDialog(CSocket002Dlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CSocket002Dlg)
	m_ipcl = _T("");
	m_portcl = _T("");
	m_dir = _T("C:/");
	//}}AFX_DATA_INIT
	// Note that LoadIcon does not require a subsequent DestroyIcon in
Win32
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}
void CSocket002Dlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CSocket002Dlg)
	DDX_Control(pDX, IDC_SLIDER1, m_Slide1);
	DDX_Control(pDX, IDC_LIST1, m_message);
	DDX_Text(pDX, IDC_STATIC1, m_ipcl);
	DDX_Text(pDX, IDC_STATIC2, m_portcl);
	DDX_Text(pDX, IDC_EDIT1, m_dir);
	//}}AFX_DATA_MAP
}
BEGIN_MESSAGE_MAP(CSocket002Dlg, CDialog)
	//{{AFX_MSG_MAP(CSocket002Dlg)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_BN_CLICKED(IDC_BUTTON2, OnStart)
	ON_BN_CLICKED(IDC_BUTTON3, OnSendData)
	ON_BN_CLICKED(IDC_BUTTON1, OnCloseListen)
	ON_BN_CLICKED(IDC_BUTTON4, OnCheckCamera)
	ON_BN_CLICKED(IDC_BUTTON7, OnStopCamera)
	ON_NOTIFY(NM_RELEASEDCAPTURE, IDC_SLIDER1,
OnReleasedcaptureSlider1)
	ON_BN_CLICKED(IDC_BUTTON5, OnStopSend)
	ON_BN_CLICKED(IDC_BUTTON9, OnBrowse)
	ON_BN_CLICKED(IDC_BUTTON6, OnButton6)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()
////////////////////////////////////////////////////////////////////////////
////
// CSocket002Dlg message handlers
BOOL CSocket002Dlg::OnInitDialog()
{
	CDialog::OnInitDialog();
```

```
// Add "About..." menu item to system menu.
// IDM_ABOUTBOX must be in the system command range.
      ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
      ASSERT(IDM_ABOUTBOX < 0xF000);
CMenu* pSysMenu = GetSystemMenu(FALSE);
      if (pSysMenu != NULL)
      {
            CString strAboutMenu;
            strAboutMenu.LoadString(IDS_ABOUTBOX);
            if (!strAboutMenu.IsEmpty())
            {
                  pSysMenu->AppendMenu(MF_SEPARATOR);
                  pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
strAboutMenu);
            }
      }
// Set the icon for this dialog.  The framework does this automatically
      //  when the application's main window is not a dialog
      SetIcon(m_hIcon, TRUE);             // Set big icon
      SetIcon(m_hIcon, FALSE);            // Set small icon
      // TODO: Add extra initialization here
      server.SetMainwindow(this);
      client.SetMainwindow(this);
      m_Slide1.SetRange(1,100,0);
      m_Slide1.SetPos(0);
      m_Slide1.SetTicFreq(1);
global_br = 0;
      start_status = FALSE ;
frame_delay = 66 ; //Default = 0
//initial serial port
      hComp = CreateFile( chCommPort,
                  GENERIC_READ | GENERIC_WRITE,
                  0,
                  NULL,
                  OPEN_EXISTING,
                  0,
                  NULL
                  );
if (hComp == INVALID_HANDLE_VALUE)
      {
MessageBox("Create Data File Fail");
            exit(0);
      }
valReturn = GetCommState(hComp, &dcb);
if (!valReturn)
      {
            MessageBox("Check state Fail");
            exit(0);
      }
//comment
      // CBR_110 CBR_19200
      // CBR_300 CBR_38400
      // CBR_600 CBR_56000
      // CBR_1200 CBR_57600
      // CBR_2400 CBR_115200
      // CBR_4800 CBR_128000
      // CBR_9600 CBR_256000
      // CBR_14400
      dcb.BaudRate = CBR_9600; //baud rate
      dcb.ByteSize = 8; // data size
      dcb.Parity = NOPARITY; // Parity
```

```
	// EVENPARITY		Even
	// MARKPARITY		Mark
	// NOPARITY		No parity
	// ODDPARITY		Odd
	// SPACEPARITY		Space
dcb.StopBits = ONESTOPBIT;
//ONESTOPBIT	1 stop bit
	//ONE5STOPBITS  1.5 stop bits
	//TWOSTOPBITS	2 stop bits
valReturn = SetCommState(hComp, &dcb);
if (!valReturn)
	{
		MessageBox("Port Define Fail");
		exit(0);
	}
//calibrate
	char Tmp;
	Tmp = 'c';
	WriteFile(hComp,&Tmp,8,&nWrittenPort, NULL);
return TRUE;  // return TRUE  unless you set the focus to a control
}
void CSocket002Dlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialog::OnSysCommand(nID, lParam);
	}
}
// If you add a minimize button to your dialog, you will need the code
below
//  to draw the icon.  For MFC applications using the document/view
model,
//  this is automatically done for you by the framework.
void CSocket002Dlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this); // device context for painting
SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);
// Center icon in client rectangle
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;
// Draw the icon
		dc.DrawIcon(x, y, m_hIcon);
	}
	else
	{
		CDialog::OnPaint();
	}
}
// The system calls this to obtain the cursor to display while the user
drags
```

```
//  the minimized window.
HCURSOR CSocket002Dlg::OnQueryDragIcon()
{
	return (HCURSOR) m_hIcon;
}
void CSocket002Dlg::OnAccept()
{
	if (server.Accept(client))
	{
		UpdateData(TRUE);
		m_message.AddString("Accept Connection.");
		UINT port;
		client.GetPeerName(m_ipcl,port); //ip address into m_ipcl and
port no. into port
		m_portcl.Format("%d",port); // transform int port to string
port and save in %d
		m_message.AddString(m_ipcl); // show ip address in Debug
windows
GetDlgItem(IDC_BUTTON3)->EnableWindow(TRUE);
		UpdateData(FALSE);
}
	else
		MessageBox("Error accept connection");
}
void CSocket002Dlg::OnConnect()
{
	m_message.AddString("OnConnect is called");
}
void CSocket002Dlg::OnClose()
{
	m_message.AddString("Socket is closed");
}
void CSocket002Dlg::OnSend()
{
}
void CSocket002Dlg::OnReceive() // receives command from client
{
	char *buff;
	char Tmp[513];
int size=1024;
	buff=new char[1025];
int r=client.Receive(buff,size);
	memcpy(Tmp,buff,513);
if (r==SOCKET_ERROR)
	{
		m_message.AddString("Error");
	}
	else
	{
		if ( Tmp[1] == 'F') // _FPS_NO_ that's mean F is a command
for adjust frame rate
		{
			Tmp[513] = NULL;
			m_message.AddString(Tmp);
			if	(Tmp[8] == '0') // 10 fps
				frame_delay = 100;
else if		(Tmp[8] == '1') // 15 fps
				frame_delay = 66;
else if		(Tmp[8] == '2') // 20 fps
				frame_delay = 50;
else if (Tmp[8] == '3') // 25 fps
```

```
                        frame_delay = 40;
                  else                              //30 fps
                        frame_delay = 33;
}
            else if (Tmp[1] == 'D') // _DIRECT_ D is a direction cammand
by sending U R L D
            {
                  Tmp[513] = NULL;
                  m_message.AddString(Tmp);
WriteFile(hComp,&Tmp[8],8,&nWrittenPort, NULL);
                  //call function for control step motor
            }
            else if ( Tmp[1] == 'C')
            {
                  char Strtime[512];
Tmp[513] = NULL;
                  m_message.AddString(Tmp);
                  // Start Capture File
                  strcpy(Strtime,Tmp+8);
sscanf(Strtime,"%ld",&limit_time);
                  StartCap = 1 ;
}
            else if ( Tmp[1] == 'S')
            {
                  Tmp[513] = NULL;
                  m_message.AddString(Tmp);
                  //    Stop Capture File
                  StartCap = 0 ;
            }
      }
}
void CSocket002Dlg::OnStart() //Listen button
{
      UpdateData(TRUE);
      if (server.Create(4000))
      {
            m_message.AddString("Listening...");
            server.Listen();
GetDlgItem(IDC_BUTTON2)->EnableWindow(FALSE);
            GetDlgItem(IDC_BUTTON1)->EnableWindow(TRUE);
}
      else
            m_message.AddString("Error starting Server");
      UpdateData(FALSE);
}
void CSocket002Dlg::OnCancel()
{
      //    server.Close();
      //    client.Close();
      // TODO: Add extra cleanup here
      CDialog::OnCancel();
}
void CSocket002Dlg::OnSendData() // send vdo button
{
      UpdateData(TRUE);
start_status = TRUE;  ////////////start and stop button***
      AfxBeginThread(MyThreadFunct,this, THREAD_PRIORITY_NORMAL, 0, 0);
// this means this class
GetDlgItem(IDC_BUTTON3)->EnableWindow(FALSE);
      GetDlgItem(IDC_BUTTON5)->EnableWindow(TRUE);
UpdateData(FALSE);
```

```
}
void CSocket002Dlg::OnCloseListen() //Close listen button
{
	// TODO: Add your control notification handler code here
	UpdateData(TRUE);
	server.ShutDown();
	server.Close();
	m_message.AddString("Stop Listen");
GetDlgItem(IDC_BUTTON2)->EnableWindow(TRUE);
	GetDlgItem(IDC_BUTTON1)->EnableWindow(FALSE);
UpdateData(FALSE);
}
void CSocket002Dlg::OnCheckCamera() // Start camera button
{
UpdateData(TRUE);
	int ncams = cvcamGetCamerasCount();//returns the number of
available cameras
	if (ncams < 1)
	{
		AfxMessageBox("Please Connect Camera");
	}
	else
	{
		//Initial property
		// Set 1 or 0 for select camera
		//cvcamSetProperty(0, CVCAM_VIDEOFORMAT, NULL);
		cvcamSetProperty(0, CVCAM_PROP_ENABLE, CVCAMTRUE); //enable
1st found camera
		cvcamSetProperty(0, CVCAM_PROP_RENDER, CVCAMTRUE) ;  //
render vdo from camera
		cvcamSetProperty(0, CVCAM_PROP_CALLBACK, PreviewCallback);
		cvcamInit(); // call the camera's default setting window
		cvcamStart(); // show vdo stream
GetDlgItem(IDC_BUTTON7)->EnableWindow(TRUE);
		GetDlgItem(IDC_BUTTON4)->EnableWindow(FALSE);
	UpdateData(FALSE);
}
void CSocket002Dlg::OnStopCamera() // stop camera button
{
	// TODO: Add your control notification handler code here
	UpdateData(TRUE);
	cvcamStop(); // stop rendering vdo stream from camera
	cvcamExit(); // shutdown the camera
GetDlgItem(IDC_BUTTON7)->EnableWindow(FALSE);
	GetDlgItem(IDC_BUTTON4)->EnableWindow(TRUE);
UpdateData(FALSE);
}
void CSocket002Dlg::OnReleasedcaptureSlider1(NMHDR* pNMHDR, LRESULT*
pResult) // brightness control slider
{
	// TODO: Add your control notification handler code here
	global_br   = m_Slidel.GetPos();
	*pResult = 0;
}
void PreviewCallback(IplImage* previm) // works everytime when program
gets 1 frame
{
	int i,j;
cvCvtColor(previm,previm,CV_BGR2YCrCb); //convert to YCrCb for adjust
brigthness
for(i=0; i<previm->height; i++)
```

```
	{
		for(j=(previm->widthStep)*i; j<(previm->widthStep)*(i+1);
j+=previm->nChannels)
		{
			if ( global_br + (unsigned char)previm->imageData[j] >
255 )
				previm->imageData[j] = (char)255;
			else
				previm->imageData[j] = global_br + (unsigned
char)previm->imageData[j];
		}
}
cvCvtColor(previm,previm,CV_YCrCb2BGR);// Convert back to BGR for display
	// Capture Zone
	//
	if (StartCap == 1)
	{
		if ( stat == 0 )
		{
			//initailization
				time(&start_time); // tick time
			stat = 1;
time ( &rawtime ); // parameter for saving time data
			timeinfo = localtime ( &rawtime ); // turns int into
daytime data
strftime (buffer,80,"%a.%d.%b.%y_%H.%M.%S",timeinfo); // get time
information from computer
	sprintf(filename,"%s%s.avi",filedir,buffer); // save into file path
		writer = cvCreateVideoWriter(filename,
			CV_FOURCC('D', 'I', 'B', ' '), 30, cvSize(320, 240));
				cvWriteFrame(writer,previm);
		}
		else
		{
			cvWriteFrame(writer,previm);
			time(&cur_time);
			if ( cur_time - start_time > limit_time )
			{
				cvReleaseVideoWriter(&writer);
				stat = 0;
			}
		}
	}
	else
	{
		if ( stat < 2)
		{
			cvReleaseVideoWriter(&writer);
			stat = 2;
		}
	}
if (start_status == TRUE)
		{
Rec_ptr = previm->imageData;  //Rec_ptr is a pointer that keeps vdo data
as buffer
		}
}
UINT MyThreadFunct(LPVOID pParam) //
{
	CSocket002Dlg* pControl = (CSocket002Dlg*)pParam;
	while(start_status == TRUE)
```

```
            {
                  Sleep(frame_delay);
                  //pControl->client.Send(Rec_ptr,57600);//57600 come
from 120*160*3
                  pControl->client.Send(Rec_ptr,230400);//230400 come
from 240*320*3
            }
return 0;
}
void CSocket002Dlg::OnStopSend()
{
      // TODO: Add your control notification handler code here
      UpdateData(TRUE);
      start_status = FALSE;
GetDlgItem(IDC_BUTTON3)->EnableWindow(TRUE);
      GetDlgItem(IDC_BUTTON5)->EnableWindow(FALSE);
      UpdateData(FALSE);
}
void CSocket002Dlg::OnBrowse() // Browse button
{
      // TODO: Add your control notification handler code here
      UpdateData(TRUE);
      CFileDialog
l_SampleDlg(FALSE,NULL,NULL,OFN_ENABLESIZING|OFN_EXPLORER|OFN_FILEMUSTEXI
ST
            ,"avi (*.avi)|*.avi",this);
      int iRet = l_SampleDlg.DoModal();
  if ( iRet == IDOK )
      {
            l_strFileName = l_SampleDlg.GetPathName();
            sprintf(filedir,"%s",l_strFileName);
      m_dir = l_strFileName;
      }
      else
      {
            sprintf(filedir,"C:/");
            m_dir = "C:/"; //Default
      }
      UpdateData(FALSE);
}
void CSocket002Dlg::OnButton6()
{
      // TODO: Add your control notification handler code here
      ReadFile(hComp,&chBuffer,8,&nReadPort,NULL);
      CString tmp;
      tmp = chBuffer;
      m_message.AddString(tmp);
}

// stdafx.cpp : source file that includes just the standard includes
//    Socket002.pch will be the pre-compiled header
//    stdafx.obj will contain the pre-compiled type information
#include "stdafx.h"
```

-โปรแกรม Client

```
#if
!defined(AFX_MYSOCKET_H__0D78AB7C_74AD_449A_B566_55DE62AABBEE__INCLUDED_)
#define AFX_MYSOCKET_H__0D78AB7C_74AD_449A_B566_55DE62AABBEE__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// MySocket.h : header file
//
////////////////////////////////////////////////////////////////////////////
////
// CMySocket command target
class CMySocket : public CAsyncSocket
{
      CDialog *maindlg;
            public:
                  void SetMainwindow(CDialog*dlg)
                  {
                        maindlg=dlg;
                  }
      // Attributes
public:
// Operations
public:
      CMySocket();
      virtual ~CMySocket();
// Overrides
public:
      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CMySocket)
      public:
      virtual void OnAccept(int nErrorCode);
      virtual void OnClose(int nErrorCode);
      virtual void OnConnect(int nErrorCode);
      virtual void OnSend(int nErrorCode);
      virtual void OnReceive(int nErrorCode);
      //}}AFX_VIRTUAL
// Generated message map functions
      //{{AFX_MSG(CMySocket)
            // NOTE - the ClassWizard will add and remove member
functions here.
      //}}AFX_MSG
// Implementation
protected:
};
////////////////////////////////////////////////////////////////////////////
////
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.
#endif //
!defined(AFX_MYSOCKET_H__0D78AB7C_74AD_449A_B566_55DE62AABBEE__INCLUDED_)


//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by Socket002.rc
//
#define IDM_ABOUTBOX                    0x0010
#define IDD_ABOUTBOX                    100
```

```
#define IDS_ABOUTBOX                    101
#define IDD_SOCKET002_DIALOG            102
#define IDR_MAINFRAME                   128
#define IDC_LIST1                       1000
#define IDC_EDIT1                       1001
#define IDC_BUTTON1                     1002
#define IDC_BUTTON2                     1003
#define IDC_BUTTON3                     1004
#define IDC_BUTTON4                     1005
#define IDC_BUTTON5                     1006
#define IDC_BUTTON6                     1007
#define IDC_COMBO1                      1010
#define IDC_STATIC1                     1012
#define IDC_STATIC2                     1013
#define IDC_SLIDER1                     1014
#define IDC_BUTTON7                     1015
#define IDC_BUTTON8                     1016
#define IDC_EDIT2                       1017
#define IDC_EDIT3                       1018
#define IDC_EDIT4                       1019
#define IDC_COMBO3                      1021
#define IDC_COMBO4                      1022
#define IDC_BUTTON9                     1023
#define IDC_BUTTON10                    1024
#define IDC_BUTTON11                    1025
// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        131
#define _APS_NEXT_COMMAND_VALUE         32771
#define _APS_NEXT_CONTROL_VALUE         1026
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif

// Socket002.h : main header file for the SOCKET002 application
//
#if
!defined(AFX_SOCKET002_H__5962AAFA_B93B_4C98_A6C9_D7504671CCC7__INCLUDED_
)
#define AFX_SOCKET002_H__5962AAFA_B93B_4C98_A6C9_D7504671CCC7__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
 #ifndef __AFXWIN_H__
      #error include 'stdafx.h' before including this file for PCH
#endif
#include "resource.h"         // main symbols
////////////////////////////////////////////////////////////////////////////
////
// CSocket002App:
// See Socket002.cpp for the implementation of this class
//
class CSocket002App : public CWinApp
{
public:
      CSocket002App();
// Overrides
      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CSocket002App)
```

```
	public:
	virtual BOOL InitInstance();
	//}}AFX_VIRTUAL
// Implementation
//{{AFX_MSG(CSocket002App)
		// NOTE - the ClassWizard will add and remove member
functions here.
		//    DO NOT EDIT what you see in these blocks of generated
code !
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
/////////////////////////////////////////////////////////////////////////////
////
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.
#endif //
!defined(AFX_SOCKET002_H__5962AAFA_B93B_4C98_A6C9_D7504671CCC7__INCLUDED_
)

// Socket002Dlg.h : header file
//
#if
!defined(AFX_SOCKET002DLG_H__1162EDEC_DD8F_4FD1_AC68_06FE3FD33B68__INCLUD
ED_)
#define
AFX_SOCKET002DLG_H__1162EDEC_DD8F_4FD1_AC68_06FE3FD33B68__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
#include "MySocket.h"
#include "cv.h"
#include "highgui.h"
#include "cxcore.h"
#include "cvcam.h"
/////////////////////////////////////////////////////////////////////////////
////
// CSocket002Dlg dialog
class CSocket002Dlg : public CDialog
{
	CMySocket server,client;
	// Construction
public:
	CSocket002Dlg(CWnd* pParent = NULL);	// standard constructor
void OnAccept();
	void OnConnect();
	void OnClose();
	void OnSend();
	void OnReceive();
long m_showRcv ;
	char *NetRec_ptr;
	IplImage *r_dat;
// Dialog Data
	//{{AFX_DATA(CSocket002Dlg)
	enum { IDD = IDD_SOCKET002_DIALOG };
	CComboBox	m_tilt;
	CComboBox	m_pan;
	CSliderCtrl m_Slide1;
	CComboBox	m_fps;
	CListBox	m_message;
```

```
  CString    m_ipaddress;
      CString     m_ipsv;
      CString     m_portsv;
      UINT  m_sec;
UINT  m_minute;
      UINT  m_hrs;
      //}}AFX_DATA
// ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CSocket002Dlg)
      protected:
      virtual void DoDataExchange(CDataExchange* pDX);      // DDX/DDV
support
      //}}AFX_VIRTUAL
// Implementation
protected:
      HICON m_hIcon;
// Generated message map functions
      //{{AFX_MSG(CSocket002Dlg)
      virtual BOOL OnInitDialog();
      afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
      afx_msg void OnPaint();
      afx_msg HCURSOR OnQueryDragIcon();
      afx_msg void OnOK();
      virtual void OnCancel();
      afx_msg void OnUpData();
      afx_msg void OnDisConnect();
      afx_msg void OnSelchangeCombo1();
      afx_msg void OnReleasedcaptureSlider1(NMHDR* pNMHDR, LRESULT*
pResult);
      afx_msg void OnRIGHT();
      afx_msg void OnLEFT();
      afx_msg void OnDOWN();
      afx_msg void OnCapture();
      afx_msg void OnStopCap();
      afx_msg void OnAuto();
      afx_msg void OnStopAuto();
      afx_msg void OnCali();
      //}}AFX_MSG
      DECLARE_MESSAGE_MAP()
};
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.
#endif //
!defined(AFX_SOCKET002DLG_H__1162EDEC_DD8F_4FD1_AC68_06FE3FD33B68__INCLUD
ED_)

// stdafx.h : include file for standard system include files,
//  or project specific include files that are used frequently, but
//      are changed infrequently
//
#if
!defined(AFX_STDAFX_H__9918A076_C2AD_4724_B5BB_3700602AAEBC__INCLUDED_)
#define AFX_STDAFX_H__9918A076_C2AD_4724_B5BB_3700602AAEBC__INCLUDED_
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
#define VC_EXTRALEAN          // Exclude rarely-used stuff from Windows
headers
#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions
```

```
#include <afxdisp.h>        // MFC Automation classes
#include <afxdtctl.h>         // MFC support for Internet Explorer 4
Common Controls
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>                    // MFC support for Windows Common
Controls
#endif // _AFX_NO_AFXCMN_SUPPORT
#include <afxsock.h>
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
before the previous line.
#endif //
!defined(AFX_STDAFX_H__9918A076_C2AD_4724_B5BB_3700602AAEBC__INCLUDED_)

// MySocket.cpp : implementation file
//
#include "stdafx.h"
#include "Socket002.h"
#include "MySocket.h"
#include "Socket002Dlg.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
//////////////////////////////////////////////////////////////////////
////
// CMySocket
CMySocket::CMySocket()
{
}
CMySocket::~CMySocket()
{
}
// Do not edit the following lines, which are needed by ClassWizard.
#if 0
BEGIN_MESSAGE_MAP(CMySocket, CAsyncSocket)
      //{{AFX_MSG_MAP(CMySocket)
      //}}AFX_MSG_MAP
END_MESSAGE_MAP()
#endif      // 0
//////////////////////////////////////////////////////////////////////
////
// CMySocket member functions
void CMySocket::OnAccept(int nErrorCode)
{
      // TODO: Add your specialized code here and/or call the base class
      if (nErrorCode == 0)
      ((CSocket002Dlg *)maindlg)->OnAccept();
      CAsyncSocket::OnAccept(nErrorCode);
}
void CMySocket::OnClose(int nErrorCode)
{
      // TODO: Add your specialized code here and/or call the base class
      if (nErrorCode == 0)
      ((CSocket002Dlg *)maindlg)->OnClose();
      CAsyncSocket::OnClose(nErrorCode);
}
void CMySocket::OnConnect(int nErrorCode)
{
      // TODO: Add your specialized code here and/or call the base class
```

```
	if (nErrorCode == 0)
	((CSocket002Dlg *)maindlg)->OnConnect();
	CAsyncSocket::OnConnect(nErrorCode);
}
void CMySocket::OnSend(int nErrorCode)
{
	// TODO: Add your specialized code here and/or call the base class
	if (nErrorCode == 0)
	((CSocket002Dlg *)maindlg)->OnSend();
	CAsyncSocket::OnSend(nErrorCode);
}
void CMySocket::OnReceive(int nErrorCode)
{
	// TODO: Add your specialized code here and/or call the base class
	if (nErrorCode == 0)
	((CSocket002Dlg *)maindlg)->OnReceive();
	CAsyncSocket::OnReceive(nErrorCode);
}


// Socket002.cpp : Defines the class behaviors for the application.
//
#include "stdafx.h"
#include "Socket002.h"
#include "Socket002Dlg.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
/////////////////////////////////////////////////////////////////////////////
////
// CSocket002App
BEGIN_MESSAGE_MAP(CSocket002App, CWinApp)
	//{{AFX_MSG_MAP(CSocket002App)
		// NOTE - the ClassWizard will add and remove mapping macros
here.
		//    DO NOT EDIT what you see in these blocks of generated
code!
	//}}AFX_MSG
	ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()
/////////////////////////////////////////////////////////////////////////////
////
// CSocket002App construction
CSocket002App::CSocket002App()
{
	// TODO: add construction code here,
	// Place all significant initialization in InitInstance
}
/////////////////////////////////////////////////////////////////////////////
////
// The one and only CSocket002App object
CSocket002App theApp;
/////////////////////////////////////////////////////////////////////////////
////
// CSocket002App initialization
BOOL CSocket002App::InitInstance()
{
	if(!AfxSocketInit())
	{
		return FALSE;
```

```
	}
	AfxEnableControlContainer();
// Standard initialization
	// If you are not using these features and wish to reduce the size
	//  of your final executable, you should remove from the following
	//  the specific initialization routines you do not need.
#ifdef _AFXDLL
	Enable3dControls();			// Call this when using MFC in
a shared DLL
#else
	Enable3dControlsStatic();	// Call this when linking to MFC
statically
#endif
CSocket002Dlg dlg;
	m_pMainWnd = &dlg;
	int nResponse = dlg.DoModal();
	if (nResponse == IDOK)
	{
		// TODO: Place code here to handle when the dialog is
		//  dismissed with OK
	}
	else if (nResponse == IDCANCEL)
	{
		// TODO: Place code here to handle when the dialog is
		//  dismissed with Cancel
	}
// Since the dialog has been closed, return FALSE so that we exit the
	//  application, rather than start the application's message pump.
	return FALSE;
}

// Socket002Dlg.cpp : implementation file
//
#include "stdafx.h"
#include "Socket002.h"
#include "Socket002Dlg.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
/////////////////////////////////////////////////////////////////////////////
////
// CAboutDlg dialog used for App About
unsigned int global_br;
class CAboutDlg : public CDialog
{
public:
	CAboutDlg();
// Dialog Data
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	//}}AFX_DATA
// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAboutDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV
support
	//}}AFX_VIRTUAL
// Implementation
protected:
```

```
	//{{AFX_MSG(CAboutDlg)
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}
void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}
BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
		// No message handlers
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()
////////////////////////////////////////////////////////////////////////////
////
// CSocket002Dlg dialog

CSocket002Dlg::CSocket002Dlg(CWnd* pParent /*=NULL*/)
	: CDialog(CSocket002Dlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CSocket002Dlg)
	m_ipaddress = _T("127.0.0.1");
	m_ipsv = _T("");
	m_portsv = _T("");
	m_sec = 10;
	m_minute = 0;
	m_hrs = 0;
	//}}AFX_DATA_INIT
	// Note that LoadIcon does not require a subsequent DestroyIcon in
Win32
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}
void CSocket002Dlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CSocket002Dlg)
	DDX_Control(pDX, IDC_COMBO4, m_tilt);
	DDX_Control(pDX, IDC_COMBO3, m_pan);
	DDX_Control(pDX, IDC_SLIDER1, m_Slide1);
	DDX_Control(pDX, IDC_COMBO1, m_fps);
	DDX_Control(pDX, IDC_LIST1, m_message);
	DDX_Text(pDX, IDC_EDIT1, m_ipaddress);
	DDX_Text(pDX, IDC_STATIC1, m_ipsv);
	DDX_Text(pDX, IDC_STATIC2, m_portsv);
	DDX_Text(pDX, IDC_EDIT2, m_sec);
	DDX_Text(pDX, IDC_EDIT3, m_minute);
	DDX_Text(pDX, IDC_EDIT4, m_hrs);
	//}}AFX_DATA_MAP
}
BEGIN_MESSAGE_MAP(CSocket002Dlg, CDialog)
	//{{AFX_MSG_MAP(CSocket002Dlg)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
```

```
	ON_BN_CLICKED(IDC_BUTTON3, OnUpData)
	ON_BN_CLICKED(IDC_BUTTON2, OnDisConnect)
	ON_CBN_SELCHANGE(IDC_COMBO1, OnSelchangeCombo1)
	ON_NOTIFY(NM_RELEASEDCAPTURE, IDC_SLIDER1,
OnReleasedcaptureSlider1)
	ON_BN_CLICKED(IDC_BUTTON5, OnRIGHT)
	ON_BN_CLICKED(IDC_BUTTON6, OnLEFT)
	ON_BN_CLICKED(IDC_BUTTON4, OnDOWN)
	ON_BN_CLICKED(IDC_BUTTON7, OnCapture)
	ON_BN_CLICKED(IDC_BUTTON8, OnStopCap)
	ON_BN_CLICKED(IDC_BUTTON9, OnAuto)
	ON_BN_CLICKED(IDC_BUTTON10, OnStopAuto)
	ON_BN_CLICKED(IDC_BUTTON1, OnOK)
	ON_BN_CLICKED(IDC_BUTTON11, OnCali)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()
////////////////////////////////////////////////////////////////////////////
////
// CSocket002Dlg message handlers
BOOL CSocket002Dlg::OnInitDialog()
{
	CDialog::OnInitDialog();
// Add "About..." menu item to system menu.
// IDM_ABOUTBOX must be in the system command range.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);
CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		CString strAboutMenu;
		strAboutMenu.LoadString(IDS_ABOUTBOX);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
strAboutMenu);
		}
	}
// Set the icon for this dialog.  The framework does this automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon
	// TODO: Add extra initialization here
	server.SetMainwindow(this);
	client.SetMainwindow(this);
cvvNamedWindow( "FROMSERVER", 1 );
   cvMoveWindow( "FROMSERVER", 50, 50 );
   cvResizeWindow("FROMSERVER" , 320, 240 );
NetRec_ptr = new char[230400];
	m_showRcv = 0;
r_dat=cvCreateImageHeader( cvSize(320, 240 ), IPL_DEPTH_8U, 3 );
	cvInitImageHeader( r_dat, cvSize(320, 240 ), IPL_DEPTH_8U,
3,IPL_ORIGIN_BL, 4 );
	cvCreateData(r_dat);
	m_Slide1.SetRange(1,100,0);
	m_Slide1.SetPos(0);
	m_Slide1.SetTicFreq(1);
global_br = 0;
	//Set Combo box
	m_fps.SetCurSel(0);
	m_pan.SetCurSel(0);
```

```
	m_tilt.SetCurSel(0);
	return TRUE;  // return TRUE  unless you set the focus to a control
}
void CSocket002Dlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialog::OnSysCommand(nID, lParam);
	}
}
// If you add a minimize button to your dialog, you will need the code
below
//  to draw the icon.  For MFC applications using the document/view
model,
//  this is automatically done for you by the framework.
void CSocket002Dlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this); // device context for painting
	SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);
	// Center icon in client rectangle
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;
	// Draw the icon
		dc.DrawIcon(x, y, m_hIcon);
	}
	else
	{
		CDialog::OnPaint();
	}
}
// The system calls this to obtain the cursor to display while the user
drags
//  the minimized window.
HCURSOR CSocket002Dlg::OnQueryDragIcon()
{
	return (HCURSOR) m_hIcon;
}
void CSocket002Dlg::OnAccept()
{
	if (server.Accept(client))
	{
		m_message.AddString("Accept Connection.");
		CString ip;
		UINT port;
		client.GetPeerName(ip,port);
		m_message.AddString(ip);
}
	else
		MessageBox("Error accept connection");
}
```

```
void CSocket002Dlg::OnConnect()
{
	UINT port;
	UpdateData(TRUE);
	m_message.AddString("OnConnect is called");
	client.GetSockName(m_ipsv,port);
	m_portsv.Format("%lu",port);
	UpdateData(FALSE);
}
void CSocket002Dlg::OnClose()
{
	m_message.AddString("Socket is closed");
}
void CSocket002Dlg::OnSend()
{
}
void CSocket002Dlg::OnReceive()
{
char buffer[45000]; //Maximum size of TCP/IP
   long r=0;
	long nRecv;
int height = 240;
	int width = 320;
nRecv = client.Receive(buffer,45000);
	for (r=0; r< nRecv; r++ )
	{
	 *(NetRec_ptr + ( m_showRcv + r)) =  buffer[r] ; //arReceive
>>*(NetRec_ptr+ii); m_ShowReceive
	}
	m_showRcv += nRecv;
   if(   m_showRcv  >= 230400)
	{
		m_showRcv =0;
		r_dat->imageData = NetRec_ptr;
		//Adjust Brightness
	  int i,j;
	cvCvtColor(r_dat,r_dat,CV_BGR2YCrCb);//convert to YCrCb for adjust
brigthness
	for(i=0; i<r_dat->height; i++)
		{
			for(j=(r_dat->widthStep)*i; j<(r_dat->widthStep)*(i+1);
j+=r_dat->nChannels)
			{
				if ( global_br + (unsigned char)r_dat-
>imageData[j] > 255 )
					r_dat->imageData[j] = (char)255;
				else
					r_dat->imageData[j] = global_br + (unsigned
char)r_dat->imageData[j];
			}
	}
	cvCvtColor(r_dat,r_dat,CV_YCrCb2BGR);// Convert back to BGR for
display
	//
	cvShowImage( "FROMSERVER", r_dat );
delete NetRec_ptr;
NetRec_ptr = new char[230400];
	}
}
void CSocket002Dlg::OnOK() // Connect button
{
```

```
	UpdateData(TRUE);
	if (client.Create())
	{
		m_message.AddString("Connecting...");
		client.Connect(m_ipaddress,4000);
		GetDlgItem(IDC_BUTTON1)->EnableWindow(FALSE);
		GetDlgItem(IDC_BUTTON2)->EnableWindow(TRUE);
	}
	else
		m_message.AddString("Error creating client");
	UpdateData(FALSE);
}
void CSocket002Dlg::OnCancel()
{
server.Close();
	client.Close();
	CDialog::OnCancel();
}
void CSocket002Dlg::OnUpData() // Up direction button
{
	CString str;
	int tilt = m_tilt.GetCurSel(); //Start at 0..end Combo box
	str = "_DIRECT_u";
for ( int i = 0 ; i < tilt + 1 ; i++ )
	{
		int r=client.Send(LPCTSTR(str), str.GetLength());
		if (r==SOCKET_ERROR)
			TRACE("Error Sending\n");
		else
			TRACE("Sending OK\n");
		Sleep(250);
	}
}
void CSocket002Dlg::OnDisConnect() // Disconnect Button
{
	UpdateData(TRUE);
	client.Close();
GetDlgItem(IDC_BUTTON1)->EnableWindow(TRUE);
	GetDlgItem(IDC_BUTTON2)->EnableWindow(FALSE);
	UpdateData(FALSE);
}
void CSocket002Dlg::OnSelchangeCombo1() // Frame rate control
{
	int fps = m_fps.GetCurSel(); //Start at 0..end Combo box
	CString str;
	CString Tmp;
	Tmp.Format("%d",fps);
str = "_FPS_NO_" + Tmp ;
int r=client.Send(LPCTSTR(str), str.GetLength()); // LPCTSTR is char's
array
	if (r==SOCKET_ERROR)
		TRACE("Error Sending\n");
	else
		TRACE("Sending OK\n");
}
void CSocket002Dlg::OnReleasedcaptureSlider1(NMHDR* pNMHDR, LRESULT*
pResult) // Brightness control
{
	*pResult = 0;
	global_br   = m_Slide1.GetPos();
}
```

```
void CSocket002Dlg::OnRIGHT() // Right direction button
{
	CString str;
	int pan = m_pan.GetCurSel(); //Start at 0..end Combo box
	//CString Tmp;
	//Tmp.Format("%d",pan);
	//m_pan.GetLBText( pan, Tmp );
	str = "_DIRECT_r";
for ( int i = 0 ; i < pan +1 ; i++ )
	{
		int r=client.Send(LPCTSTR(str), str.GetLength());
		if (r==SOCKET_ERROR)
			TRACE("Error Sending\n");
		else
			TRACE("Sending OK\n");
		Sleep(250);
	}
}
void CSocket002Dlg::OnLEFT() // Left direction button
{
	CString str;
	int pan = m_pan.GetCurSel(); //Start at 0..end Combo box
	//CString Tmp;
	//Tmp.Format("%d",pan);
	//m_pan.GetLBText( pan, Tmp );
	str = "_DIRECT_l";
for ( int i = 0 ; i < pan +1 ; i++ )
	{
		int r=client.Send(LPCTSTR(str), str.GetLength());
		if (r==SOCKET_ERROR)
			TRACE("Error Sending\n");
		else
			TRACE("Sending OK\n");
		Sleep(250);
	}
}
void CSocket002Dlg::OnDOWN() // Down direction button
{
	CString str;
	int tilt = m_tilt.GetCurSel(); //Start at 0..end Combo box
	str = "_DIRECT_d";
for ( int i= 0 ; i < tilt +1 ; i++ )
	{
		int r=client.Send(LPCTSTR(str), str.GetLength());
		if (r==SOCKET_ERROR)
			TRACE("Error Sending\n");
		else
			TRACE("Sending OK\n");
		Sleep(250);
	}
}
void CSocket002Dlg::OnCapture() // Start capture vdo button
{
	unsigned long time ;
	char strTime[1024];
	UpdateData(TRUE);
	time = 3600*(m_hrs) + 60 * ( m_minute ) + (m_sec);
	sprintf(strTime,"_CAPTIM_%lu",time);
int r=client.Send(LPCTSTR(strTime), 1024);
	if (r==SOCKET_ERROR)
		TRACE("Error Sending\n");
```

```
	else
		TRACE("Sending OK\n");
	GetDlgItem(IDC_BUTTON7)->EnableWindow(FALSE);
	GetDlgItem(IDC_BUTTON8)->EnableWindow(TRUE);
	UpdateData(FALSE);
}
void CSocket002Dlg::OnStopCap()
{
	UpdateData(TRUE);
	CString str;
	str = "_STOP_";
int r=client.Send(LPCTSTR(str), str.GetLength());
	if (r==SOCKET_ERROR)
		TRACE("Error Sending\n");

else
		TRACE("Sending OK\n");
	GetDlgItem(IDC_BUTTON7)->EnableWindow(TRUE);
	GetDlgItem(IDC_BUTTON8)->EnableWindow(FALSE);
UpdateData(FALSE);
}
void CSocket002Dlg::OnAuto() // Auto button
{
CString str;
	str = "_DIRECT_a";
int r=client.Send(LPCTSTR(str), str.GetLength());
	if (r==SOCKET_ERROR)
		TRACE("Error Sending\n");
	else
		TRACE("Sending OK\n");
GetDlgItem(IDC_BUTTON9)->EnableWindow(FALSE);
	GetDlgItem(IDC_BUTTON10)->EnableWindow(TRUE);
}
void CSocket002Dlg::OnStopAuto() // Manual button
{
CString str;
	str = "_DIRECT_s";
int r=client.Send(LPCTSTR(str), str.GetLength());
	if (r==SOCKET_ERROR)
		TRACE("Error Sending\n");
	else
		TRACE("Sending OK\n");
GetDlgItem(IDC_BUTTON9)->EnableWindow(TRUE);
	GetDlgItem(IDC_BUTTON10)->EnableWindow(FALSE);
}
void CSocket002Dlg::OnCali() // Calibrate button
{
	CString str;
	str = "_DIRECT_c";
int r=client.Send(LPCTSTR(str), str.GetLength());
	if (r==SOCKET_ERROR)
		TRACE("Error Sending\n");
	else
		TRACE("Sending OK\n");
}

// stdafx.cpp : source file that includes just the standard includes
//	Socket002.pch will be the pre-compiled header
//	stdafx.obj will contain the pre-compiled type information
#include "stdafx.h"
```

โปรแกรมที่ใช้ควบคุมของกล้อง

```
#include<reg51.h>
#include<stdio.h>
code unsigned char step[24]={0X01,0X02,0X04,0X08,
        0X01,0X02,0X04,0X08,
        0X01,0X02,0X04,0X08,
        0X01,0X02,0X04,0X08,
        0X01,0X02,0X04,0X08,
        0X01,0X02,0X04,0X08};
static char step_index_xy;
static char step_index_z;
void init_serial( void );
char check_char(void);
void auto_mode(void);
void delay_ms(unsigned int time);
void main( void )
{
    init_serial();
      P1=0X00;
      P2=0X00;
      step_index_xy = 13;
    step_index_z  = 0;
      P1=step[step_index_xy];
      P2=step[step_index_z];
  while(1){
            switch(_getkey())
            {
            case 'l':
                        step_index_xy++;
                        if(step_index_xy==24)
                        {
                              step_index_xy = 23;

                        }
                  break;
            case 'r':
                      step_index_xy--;
                      if(step_index_xy==-1)
                        {
                              step_index_xy = 0;
                        }
                  break;
            case 'u':
                        step_index_z++;
                        if(step_index_z==4)
                        {
                              step_index_z = 0;
                        }
                  break;
            case 'd':
                     step_index_z--;
                       if(step_index_z==-1)
                       {
                              step_index_z = 3;
                       }
                  break;
```

```
            case 'c':
                 step_index_xy=13;
                  break;

            case 'a':
                          auto_mode();
            break;
                  default: break;
            }
      P1=step[step_index_xy];
      P2=step[step_index_z];
      }
}
void init_serial( void ){
      SCON = 0x52;
      TMOD = 0x20;
      TH1 =0xFD;
      TR1 =1;
      TI    = 1;
}
char check_char(void)
{
     if(RI) {
         RI = 0;
         return SBUF;
       }
}
void auto_mode(void)
{
       unsigned char r,l;
       r = 0;
       l = 1;
       do{
              if(r == 1){
                   step_index_xy--;
                    if(step_index_xy==-1)
                         {
                               step_index_xy = 0;
                               r = 0;
                               l =1;
                         }
                }else{
                         step_index_xy++;
                         if(step_index_xy==24)
                         {
                               step_index_xy = 23;
                               r = 1;
                               l =0;
                         }
                }
         P1=step[step_index_xy];
         delay_ms(200);
             }while(check_char()!='s');
}
void delay_ms(unsigned int time)
{
```

```
        unsigned int i;
      while(time--)
        {
            for(i=0;i<100;i++);
        }
}
```

Features

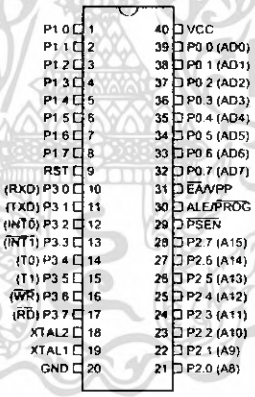
- Compatible with MCS-51™ Products
- 4K Bytes of In-System Reprogrammable Flash Memory
  - Endurance: 1,000 Write/Erase Cycles
- Fully Static Operation: 0 Hz to 24 MHz
- Three-level Program Memory Lock
- 128 x 8-bit Internal RAM
- 32 Programmable I/O Lines
- Two 16-bit Timer/Counters
- Six Interrupt Sources
- Programmable Serial Channel
- Low-power Idle and Power-down Modes

Description

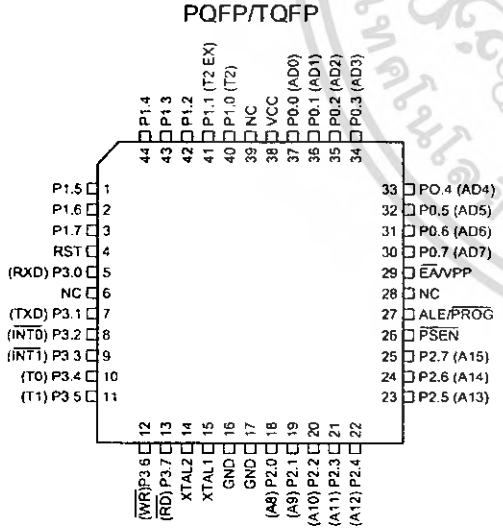
The AT89C51 is a low-power, high-performance CMOS 8-bit microcomputer with 4K bytes of Flash programmable and erasable read only memory (PEROM). The device is manufactured using Atmel's high-density nonvolatile memory technology and is compatible with the industry-standard MCS-51 instruction set and pinout. The on-chip Flash allows the program memory to be reprogrammed in-system or by a conventional nonvolatile memory programmer. By combining a versatile 8-bit CPU with Flash on a monolithic chip, the Atmel AT89C51 is a powerful microcomputer which provides a highly-flexible and cost-effective solution to many embedded control applications.

Pin Configurations

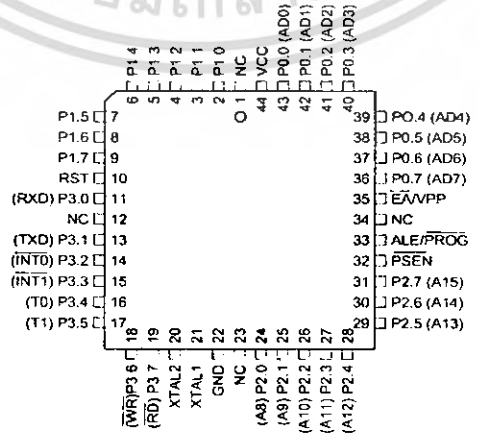
PDIP



PQFP/TQFP



PLCC



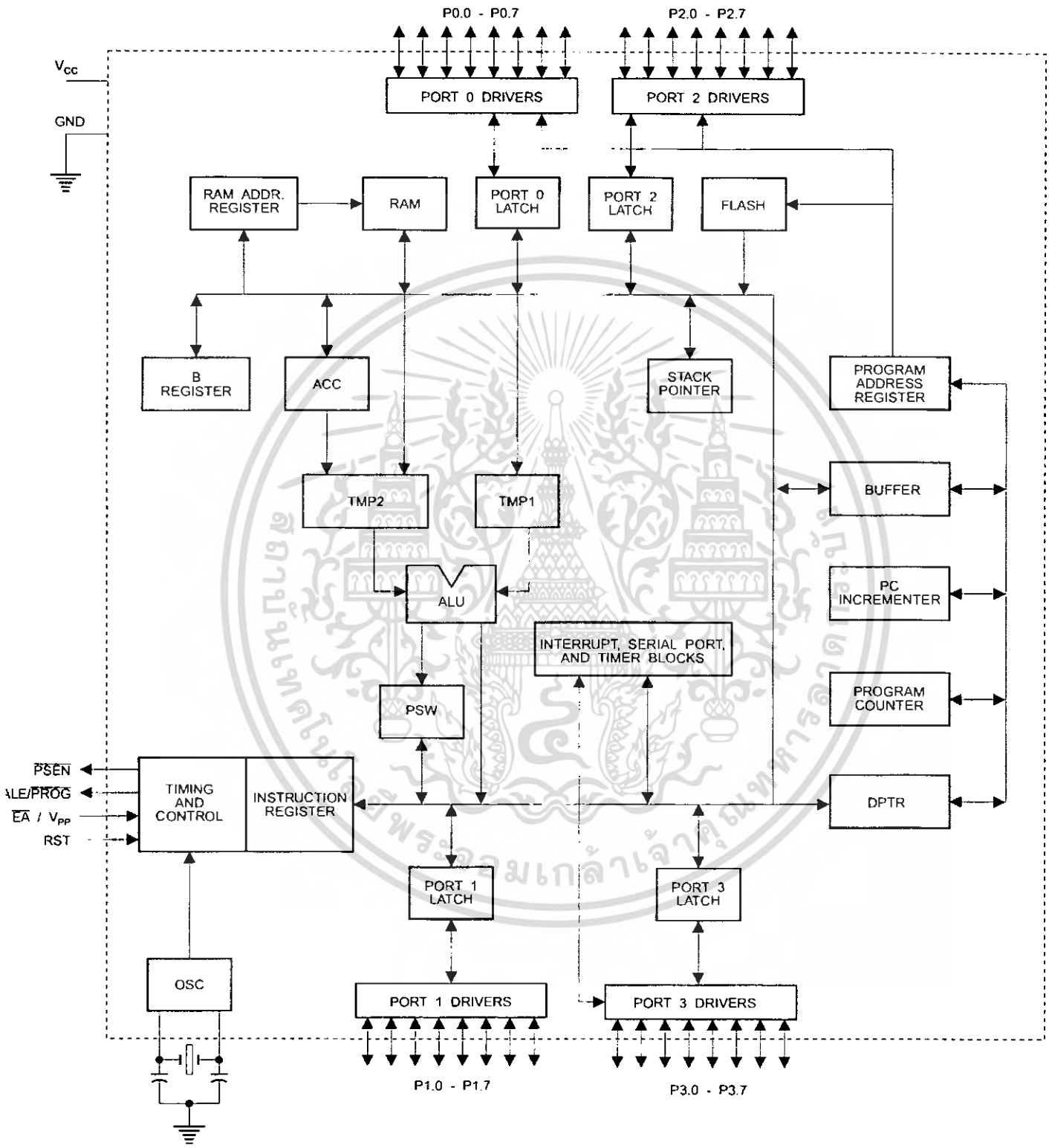
8-bit Microcontroller with 4K Bytes Flash

AT89C51

Not Recommended for New Designs. Use AT89S51.

Rev. 0265G–02/00

Block Diagram



The AT89C51 provides the following standard features: 4K bytes of Flash, 128 bytes of RAM, 32 I/O lines, two 16-bit timer/counters, a five vector two-level interrupt architecture, a full duplex serial port, on-chip oscillator and clock circuitry. In addition, the AT89C51 is designed with static logic for operation down to zero frequency and supports two software selectable power saving modes. The Idle Mode stops the CPU while allowing the RAM, timer/counters, serial port and interrupt system to continue functioning. The Power-down Mode saves the RAM contents but freezes the oscillator disabling all other chip functions until the next hardware reset.

Pin Description

VCC

Supply voltage.

GND

Ground.

Port 0

Port 0 is an 8-bit open-drain bi-directional I/O port. As an output port, each pin can sink eight TTL inputs. When 1s are written to port 0 pins, the pins can be used as high-impedance inputs.

Port 0 may also be configured to be the multiplexed low-order address/data bus during accesses to external program and data memory. In this mode P0 has internal pullups.

Port 0 also receives the code bytes during Flash programming, and outputs the code bytes during program verification. External pullups are required during program verification.

Port 1

Port 1 is an 8-bit bi-directional I/O port with internal pullups. The Port 1 output buffers can sink/source four TTL inputs. When 1s are written to Port 1 pins they are pulled high by the internal pullups and can be used as inputs. As inputs, Port 1 pins that are externally being pulled low will source current (I_{IL}) because of the internal pullups.

Port 1 also receives the low-order address bytes during Flash programming and verification.

Port 2

Port 2 is an 8-bit bi-directional I/O port with internal pullups. The Port 2 output buffers can sink/source four TTL inputs. When 1s are written to Port 2 pins they are pulled high by the internal pullups and can be used as inputs. As inputs, Port 2 pins that are externally being pulled low will source current (I_{IL}) because of the internal pullups.

Port 2 emits the high-order address byte during fetches from external program memory and during accesses to external data memory that use 16-bit addresses (MOVX @ DPTR). In this application, it uses strong internal pullups when emitting 1s. During accesses to external data memory that use 8-bit addresses (MOVX @ RI), Port 2 emits the contents of the P2 Special Function Register.

Port 2 also receives the high-order address bits and some control signals during Flash programming and verification.

Port 3

Port 3 is an 8-bit bi-directional I/O port with internal pullups. The Port 3 output buffers can sink/source four TTL inputs. When 1s are written to Port 3 pins they are pulled high by the internal pullups and can be used as inputs. As inputs, Port 3 pins that are externally being pulled low will source current (I_{IL}) because of the pullups.

Port 3 also serves the functions of various special features of the AT89C51 as listed below:

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

Port 3 also receives some control signals for Flash programming and verification.

RST

Reset input. A high on this pin for two machine cycles while the oscillator is running resets the device.

ALE/$\overline{PROG}$

Address Latch Enable output pulse for latching the low byte of the address during accesses to external memory. This pin is also the program pulse input ($\overline{PROG}$) during Flash programming.

In normal operation ALE is emitted at a constant rate of 1/6 the oscillator frequency, and may be used for external timing or clocking purposes. Note, however, that one ALE

pulse is skipped during each access to external Data Memory.

If desired, ALE operation can be disabled by setting bit 0 of SFR location 8EH. With the bit set, ALE is active only during a MOVX or MOVC instruction. Otherwise, the pin is weakly pulled high. Setting the ALE-disable bit has no effect if the microcontroller is in external execution mode.

$\overline{\text{PSEN}}$

Program Store Enable is the read strobe to external program memory.

When the AT89C51 is executing code from external program memory, $\overline{\text{PSEN}}$ is activated twice each machine cycle, except that two $\overline{\text{PSEN}}$ activations are skipped during each access to external data memory.

$\overline{\text{EA}}$/VPP

External Access Enable. $\overline{\text{EA}}$ must be strapped to GND in order to enable the device to fetch code from external program memory locations starting at 0000H up to FFFFH. Note, however, that if lock bit 1 is programmed, $\overline{\text{EA}}$ will be internally latched on reset.

$\overline{\text{EA}}$ should be strapped to V_{CC} for internal program executions.

This pin also receives the 12-volt programming enable voltage (V_{PP}) during Flash programming, for parts that require 12-volt V_{PP}.

XTAL1

Input to the inverting oscillator amplifier and input to the internal clock operating circuit.

XTAL2

Output from the inverting oscillator amplifier.

Oscillator Characteristics

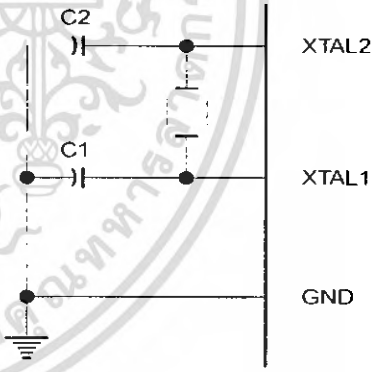
XTAL1 and XTAL2 are the input and output, respectively, of an inverting amplifier which can be configured for use as an on-chip oscillator, as shown in Figure 1. Either a quartz crystal or ceramic resonator may be used. To drive the device from an external clock source, XTAL2 should be left unconnected while XTAL1 is driven as shown in Figure 2. There are no requirements on the duty cycle of the external clock signal, since the input to the internal clocking circuitry is through a divide-by-two flip-flop, but minimum and maximum voltage high and low time specifications must be observed.

Idle Mode

In idle mode, the CPU puts itself to sleep while all the on-chip peripherals remain active. The mode is invoked by software. The content of the on-chip RAM and all the special functions registers remain unchanged during this mode. The idle mode can be terminated by any enabled interrupt or by a hardware reset.

It should be noted that when idle is terminated by a hard ware reset, the device normally resumes program execution, from where it left off, up to two machine cycles before the internal reset algorithm takes control. On-chip hardware inhibits access to internal RAM in this event, but access to the port pins is not inhibited. To eliminate the possibility of an unexpected write to a port pin when Idle is terminated by reset, the instruction following the one that invokes Idle should not be one that writes to a port pin or to external memory.

Figure 1. Oscillator Connections

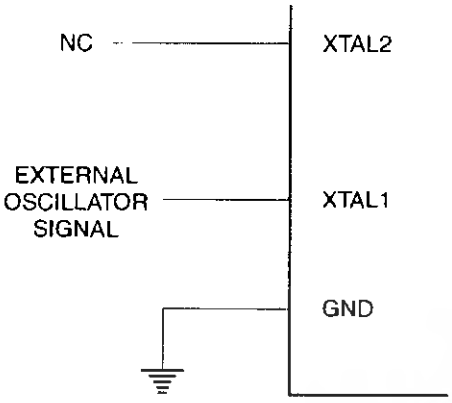


Note: C1, C2 = 30 pF ± 10 pF for Crystals
= 40 pF ± 10 pF for Ceramic Resonators

Status of External Pins During Idle and Power-down Modes

Mode	Program Memory	ALE	$\overline{\text{PSEN}}$	PORT0	PORT1	PORT2	PORT3
Idle	Internal	1	1	Data	Data	Data	Data
Idle	External	1	1	Float	Data	Address	Data
Power-down	Internal	0	0	Data	Data	Data	Data
Power-down	External	0	0	Float	Data	Data	Data

Figure 2. External Clock Drive Configuration



Power-down Mode

In the power-down mode, the oscillator is stopped, and the instruction that invokes power-down is the last instruction executed. The on-chip RAM and Special Function Registers retain their values until the power-down mode is terminated. The only exit from power-down is a hardware reset. Reset redefines the SFRs but does not change the on-chip RAM. The reset should not be activated before V_{CC} is restored to its normal operating level and must be held active long enough to allow the oscillator to restart and stabilize.

Program Memory Lock Bits

On the chip are three lock bits which can be left unprogrammed (U) or can be programmed (P) to obtain the additional features listed in the table below.

When lock bit 1 is programmed, the logic level at the $\overline{EA}$ pin is sampled and latched during reset. If the device is powered up without a reset, the latch initializes to a random value, and holds that value until reset is activated. It is necessary that the latched value of $\overline{EA}$ be in agreement with the current logic level at that pin in order for the device to function properly.

Lock Bit Protection Modes

Program Lock Bits				
	LB1	LB2	LB3	Protection Type
1	U	U	U	No program lock features
2	P	U	U	MOVC instructions executed from external program memory are disabled from fetching code bytes from internal memory, $\overline{EA}$ is sampled and latched on reset, and further programming of the Flash is disabled
3	P	P	U	Same as mode 2, also verify is disabled
4	P	P	P	Same as mode 3, also external execution is disabled

Programming the Flash

The AT89C51 is normally shipped with the on-chip Flash memory array in the erased state (that is, contents = FFH) and ready to be programmed. The programming interface accepts either a high-voltage (12-volt) or a low-voltage (V_{CC}) program enable signal. The low-voltage programming mode provides a convenient way to program the AT89C51 inside the user's system, while the high-voltage programming mode is compatible with conventional third-party Flash or EPROM programmers.

The AT89C51 is shipped with either the high-voltage or low-voltage programming mode enabled. The respective top-side marking and device signature codes are listed in the following table.

	V_{PP} = 12V	V_{PP} = 5V
Top-side Mark	AT89C51 xxxx yyww	AT89C51 xxxx-5 yyww
Signature	(030H) = 1EH (031H) = 51H (032H) =F FH	(030H) = 1EH (031H) = 51H (032H) = 05H

The AT89C51 code memory array is programmed byte-by-byte in either programming mode. *To program any non-blank byte in the on-chip Flash Memory, the entire memory must be erased using the Chip Erase Mode.*

Programming Algorithm: Before programming the AT89C51, the address, data and control signals should be set up according to the Flash programming mode table and Figure 3 and Figure 4. To program the AT89C51, take the following steps.

1. Input the desired memory location on the address lines.
2. Input the appropriate data byte on the data lines.
3. Activate the correct combination of control signals.
4. Raise $\overline{EA}/V_{PP}$ to 12V for the high-voltage programming mode.
5. Pulse ALE/$\overline{PROG}$ once to program a byte in the Flash array or the lock bits. The byte-write cycle is self-timed and typically takes no more than 1.5 ms. Repeat steps 1 through 5, changing the address and data for the entire array or until the end of the object file is reached.

$\overline{Data}$ Polling: The AT89C51 features $\overline{Data}$ Polling to indicate the end of a write cycle. During a write cycle, an attempted read of the last byte written will result in the complement of the written datum on PO.7. Once the write cycle has been completed, true data are valid on all outputs, and the next cycle may begin. $\overline{Data}$ Polling may begin any time after a write cycle has been initiated.

Ready/$\overline{Busy}$: The progress of byte programming can also be monitored by the RDY/$\overline{BSY}$ output signal. P3.4 is pulled low after ALE goes high during programming to indicate BUSY. P3.4 is pulled high again when programming is done to indicate READY.

Program Verify: If lock bits LB1 and LB2 have not been programmed, the programmed code data can be read back via the address and data lines for verification. The lock bits cannot be verified directly. Verification of the lock bits is achieved by observing that their features are enabled.

Chip Erase: The entire Flash array is erased electrically by using the proper combination of control signals and by holding ALE/$\overline{PROG}$ low for 10 ms. The code array is written with all "1"s. The chip erase operation must be executed before the code memory can be re-programmed.

Reading the Signature Bytes: The signature bytes are read by the same procedure as a normal verification of locations 030H, 031H, and 032H, except that P3.6 and P3.7 must be pulled to a logic low. The values returned are as follows.

(030H) = 1EH indicates manufactured by Atmel
(031H) = 51H indicates 89C51
(032H) = FFH indicates 12V programming
(032H) = 05H indicates 5V programming

Programming Interface

Every code byte in the Flash array can be written and the entire array can be erased by using the appropriate combination of control signals. The write operation cycle is self-timed and once initiated, will automatically time itself to completion.

All major programming vendors offer worldwide support for the Atmel microcontroller series. Please contact your local programming vendor for the appropriate software revision.

Flash Programming Modes

Mode		RST	PSEN	ALE/PROG	EA/V_{PP}	P2.6	P2.7	P3.6	P3.7
Write Code Data		H	L		H/12V	L	H	H	H
Read Code Data		H	L	H	H	L	L	H	H
Write Lock	Bit - 1	H	L		H/12V	H	H	H	H
	Bit - 2	H	L		H/12V	H	H	L	L
	Bit - 3	H	L		H/12V	H	L	H	L
Chip Erase		H	L	(1)	H/12V	H	L	L	L
Read Signature Byte		H	L	H	H	L	L	L	L

Note: 1. Chip Erase requires a 10 ms PROG pulse.

Figure 3. Programming the Flash

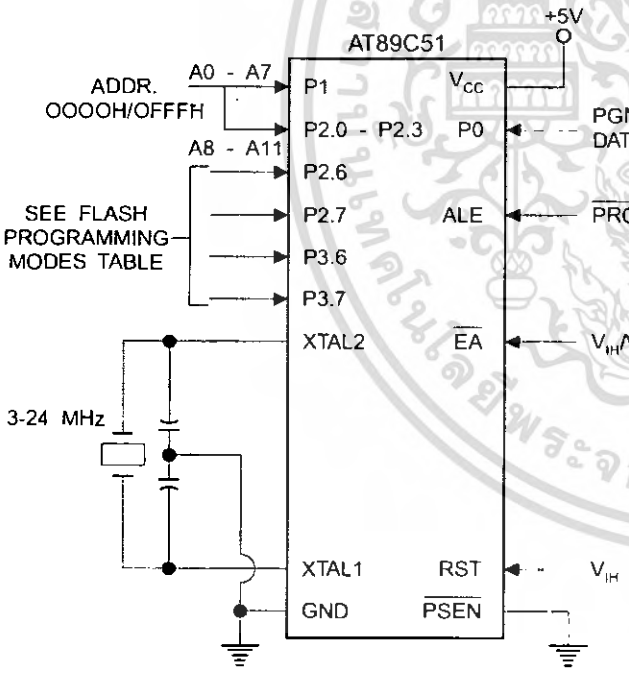
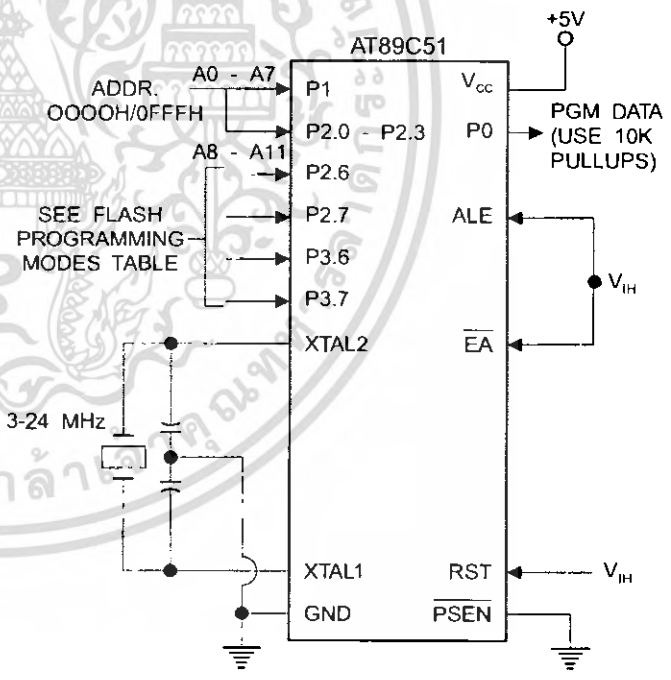
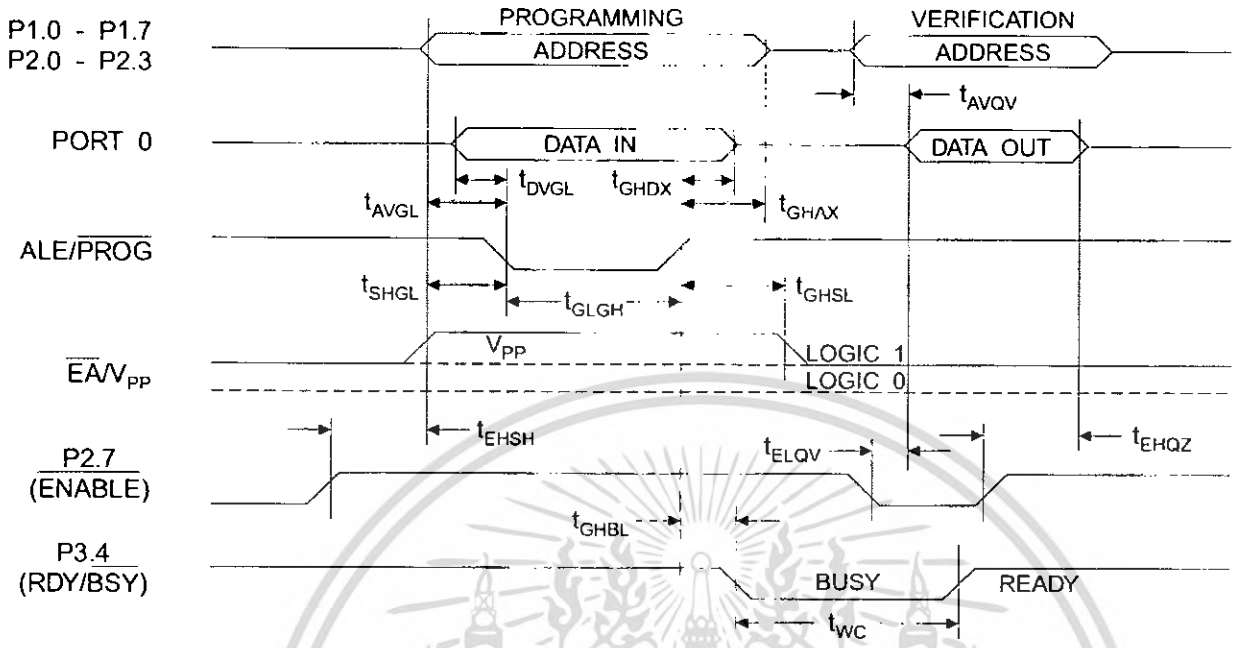


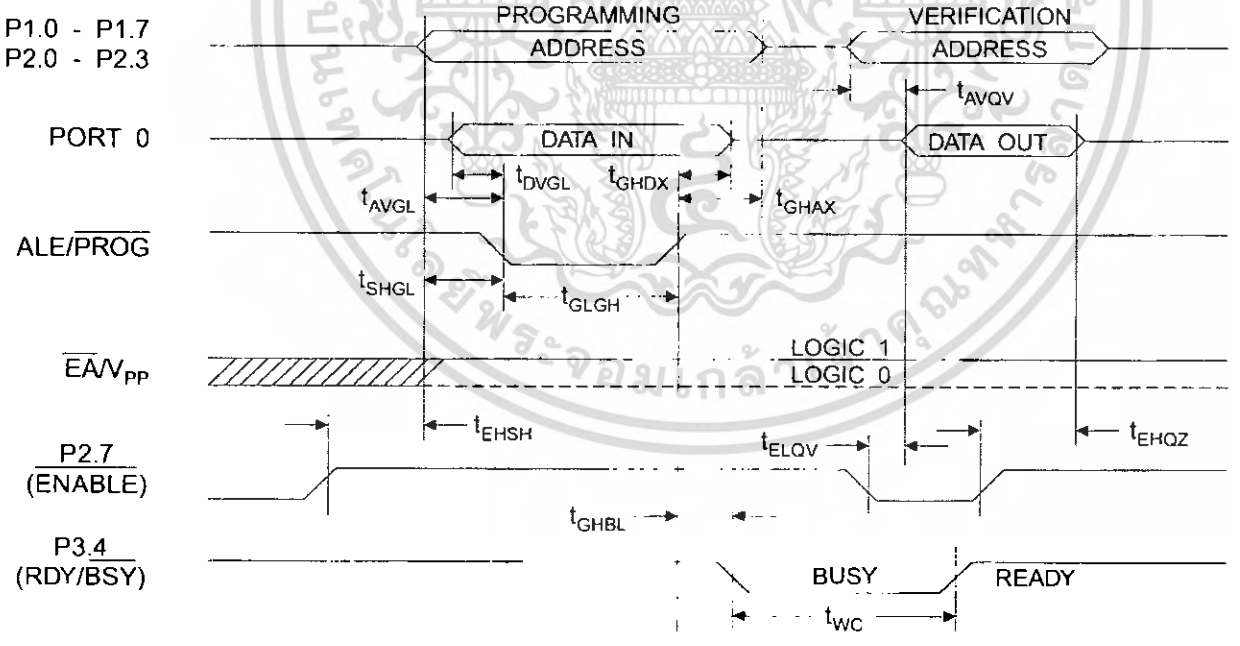
Figure 4. Verifying the Flash



Flash Programming and Verification Waveforms - High-voltage Mode (V_{PP} = 12V)



Flash Programming and Verification Waveforms - Low-voltage Mode (V_{PP} = 5V)



Flash Programming and Verification Characteristics

T_A = 0°C to 70°C, V_{CC} = 5.0 ± 10%

Symbol	Parameter	Min	Max	Units
V_{PP}[1]	Programming Enable Voltage	11.5	12.5	V
I_{PP}[1]	Programming Enable Current		1.0	mA
$1/t_{CLCL}$	Oscillator Frequency	3	24	MHz
t_{AVGL}	Address Setup to $\overline{PROG}$ Low	$48t_{CLCL}$		
t_{GHAX}	Address Hold after $\overline{PROG}$	$48t_{CLCL}$		
t_{DVGL}	Data Setup to $\overline{PROG}$ Low	$48t_{CLCL}$		
t_{GHDX}	Data Hold after $\overline{PROG}$	$48t_{CLCL}$		
t_{EHSH}	P2.7 ($\overline{ENABLE}$) High to V_{PP}	$48t_{CLCL}$		
t_{SHGL}	V_{PP} Setup to $\overline{PROG}$ Low	10		µs
t_{GHSL}[1]	V_{PP} Hold after $\overline{PROG}$	10		µs
t_{GLGH}	$\overline{PROG}$ Width	1	110	µs
t_{AVQV}	Address to Data Valid		$48t_{CLCL}$	
t_{ELQV}	$\overline{ENABLE}$ Low to Data Valid		$48t_{CLCL}$	
t_{EHQZ}	Data Float after $\overline{ENABLE}$	0	$48t_{CLCL}$	
t_{GHBL}	$\overline{PROG}$ High to $\overline{BUSY}$ Low		1.0	µs
t_{WC}	Byte Write Cycle Time		2.0	ms

Note: 1. Only used in 12-volt programming mode.

Absolute Maximum Ratings*

Operating Temperature	-55°C to +125°C
Storage Temperature	-65°C to +150°C
Voltage on Any Pin with Respect to Ground	-1.0V to +7.0V
Maximum Operating Voltage	6.6V
DC Output Current	15.0 mA

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

DC Characteristics

T_A = -40°C to 85°C, V_{CC} = 5.0V ±20% (unless otherwise noted)

Symbol	Parameter	Condition	Min	Max	Units
V_{IL}	Input Low-voltage	(Except $\overline{EA}$)	-0.5	0.2 V_{CC} - 0.1	V
V_{IL1}	Input Low-voltage ($\overline{EA}$)		-0.5	0.2 V_{CC} - 0.3	V
V_{IH}	Input High-voltage	(Except XTAL1, RST)	0.2 V_{CC} + 0.9	V_{CC} + 0.5	V
V_{IH1}	Input High-voltage	(XTAL1, RST)	0.7 V_{CC}	V_{CC} + 0.5	V
V_{OL}	Output Low-voltage[1] (Ports 1,2,3)	I_{OL} = 1.6 mA		0.45	V
V_{OL1}	Output Low-voltage[1] (Port 0, ALE, $\overline{PSEN}$)	I_{OL} = 3.2 mA		0.45	V
V_{OH}	Output High-voltage (Ports 1,2,3, ALE, $\overline{PSEN}$)	I_{OH} = -60 µA, V_{CC} = 5V ±10%	2.4		V
		I_{OH} = -25 µA	0.75 V_{CC}		V
		I_{OH} = -10 µA	0.9 V_{CC}		V
V_{OH1}	Output High-voltage (Port 0 in External Bus Mode)	I_{OH} = -800 µA, V_{CC} = 5V ±10%	2.4		V
		I_{OH} = -300 µA	0.75 V_{CC}		V
		I_{OH} = -80 µA	0.9 V_{CC}		V
I_{IL}	Logical 0 Input Current (Ports 1,2,3)	V_{IN} = 0.45V		-50	µA
I_{TL}	Logical 1 to 0 Transition Current (Ports 1,2,3)	V_{IN} = 2V, VCC = 5V ±10%		-650	µA
I_{LI}	Input Leakage Current (Port 0, $\overline{EA}$)	0.45 < V_{IN} < V_{CC}		±10	µA
RRST	Reset Pull-down Resistor		50	300	KΩ
C_{IO}	Pin Capacitance	Test Freq. = 1 MHz, T_A = 25°C		10	pF
I_{CC}	Power Supply Current	Active Mode, 12 MHz		20	mA
		Idle Mode, 12 MHz		5	mA
	Power-down Mode[2]	V_{CC} = 6V		100	µA
		V_{CC} = 3V		40	µA

Notes:
1. Under steady state (non-transient) conditions, I_{OL} must be externally limited as follows:
   Maximum I_{OL} per port pin: 10 mA
   Maximum I_{OL} per 8-bit port: Port 0: 26 mA
   Ports 1, 2, 3: 15 mA
   Maximum total I_{OL} for all output pins: 71 mA
   If I_{OL} exceeds the test condition, V_{OL} may exceed the related specification. Pins are not guaranteed to sink current greater than the listed test conditions.
2. Minimum V_{CC} for Power-down is 2V.

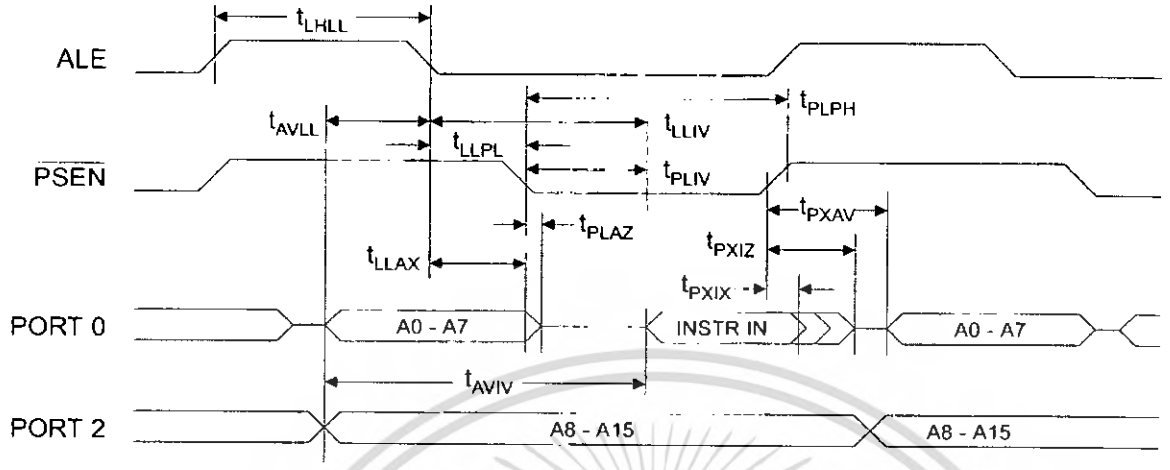
AC Characteristics

Under operating conditions, load capacitance for Port 0, ALE/$\overline{PROG}$, and $\overline{PSEN}$ = 100 pF; load capacitance for all other outputs = 80 pF.

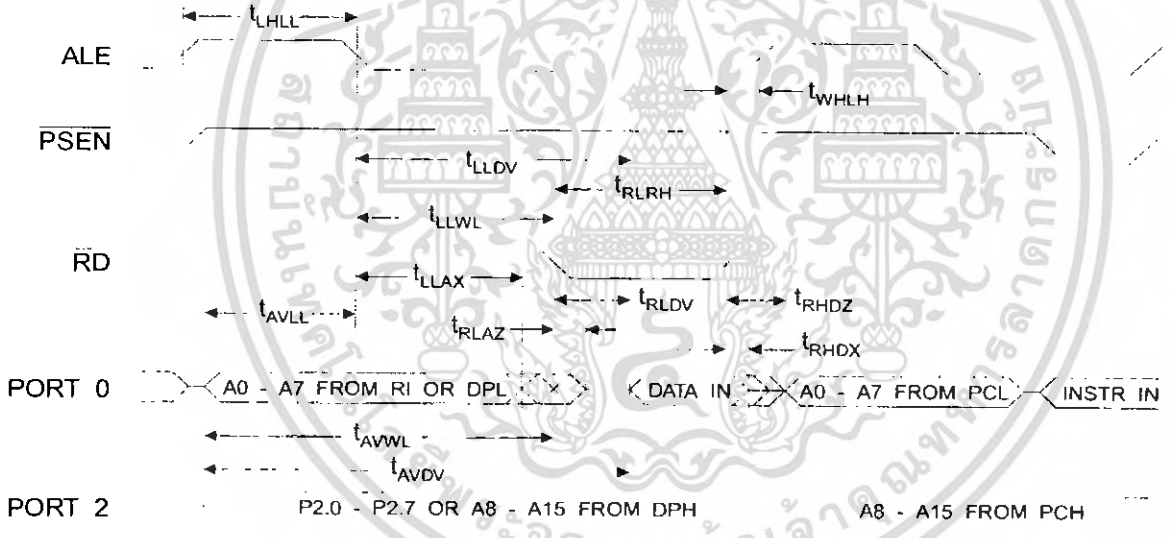
External Program and Data Memory Characteristics

Symbol	Parameter	12 MHz Oscillator		16 to 24 MHz Oscillator		Units
		Min	Max	Min	Max	
$1/t_{CLCL}$	Oscillator Frequency			0	24	MHz
t_{LHLL}	ALE Pulse Width	127		$2t_{CLCL}$-40		ns
t_{AVLL}	Address Valid to ALE Low	43		t_{CLCL}-13		ns
t_{LLAX}	Address Hold after ALE Low	48		t_{CLCL}-20		ns
t_{LLIV}	ALE Low to Valid Instruction In		233		$4t_{CLCL}$-65	ns
t_{LLPL}	ALE Low to $\overline{PSEN}$ Low	43		t_{CLCL}-13		ns
t_{PLPH}	$\overline{PSEN}$ Pulse Width	205		$3t_{CLCL}$-20		ns
t_{PLIV}	$\overline{PSEN}$ Low to Valid Instruction In		145		$3t_{CLCL}$-45	ns
t_{PXIX}	Input Instruction Hold after $\overline{PSEN}$	0		0		ns
t_{PXIZ}	Input Instruction Float after $\overline{PSEN}$		59		t_{CLCL}-10	ns
t_{PXAV}	$\overline{PSEN}$ to Address Valid	75		t_{CLCL}-8		ns
t_{AVIV}	Address to Valid Instruction In		312		$5t_{CLCL}$-55	ns
t_{PLAZ}	$\overline{PSEN}$ Low to Address Float		10		10	ns
t_{RLRH}	$\overline{RD}$ Pulse Width	400		$6t_{CLCL}$-100		ns
t_{WLWH}	$\overline{WR}$ Pulse Width	400		$6t_{CLCL}$-100		ns
t_{RLDV}	$\overline{RD}$ Low to Valid Data In		252		$5t_{CLCL}$-90	ns
t_{RHDX}	Data Hold after $\overline{RD}$	0		0		ns
t_{RHDZ}	Data Float after $\overline{RD}$		97		$2t_{CLCL}$-28	ns
t_{LLDV}	ALE Low to Valid Data In		517		$8t_{CLCL}$-150	ns
t_{AVDV}	Address to Valid Data In		585		$9t_{CLCL}$-165	ns
t_{LLWL}	ALE Low to $\overline{RD}$ or $\overline{WR}$ Low	200	300	$3t_{CLCL}$-50	$3t_{CLCL}$+50	ns
t_{AVWL}	Address to $\overline{RD}$ or $\overline{WR}$ Low	203		$4t_{CLCL}$-75		ns
t_{QVWX}	Data Valid to $\overline{WR}$ Transition	23		t_{CLCL}-20		ns
t_{QVWH}	Data Valid to $\overline{WR}$ High	433		$7t_{CLCL}$-120		ns
t_{WHQX}	Data Hold after $\overline{WR}$	33		t_{CLCL}-20		ns
t_{RLAZ}	$\overline{RD}$ Low to Address Float		0		0	ns
t_{WHLH}	$\overline{RD}$ or $\overline{WR}$ High to ALE High	43	123	t_{CLCL}-20	t_{CLCL}+25	ns

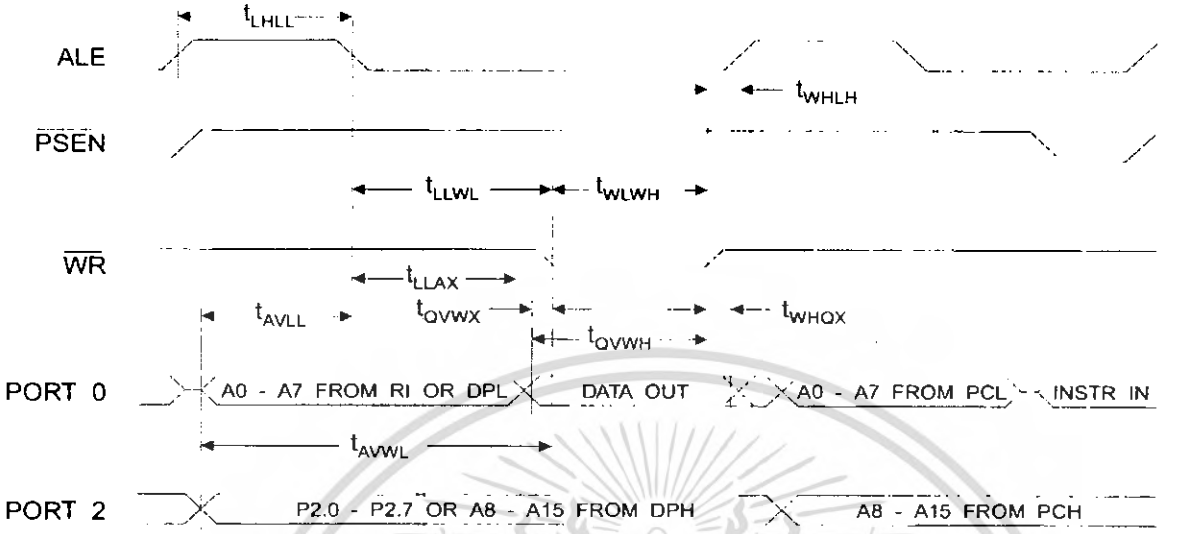
External Program Memory Read Cycle



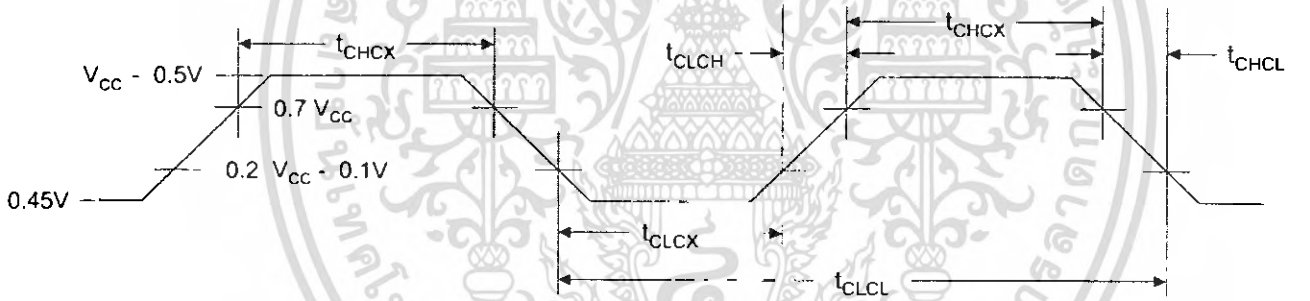
External Data Memory Read Cycle



External Data Memory Write Cycle



External Clock Drive Waveforms



External Clock Drive

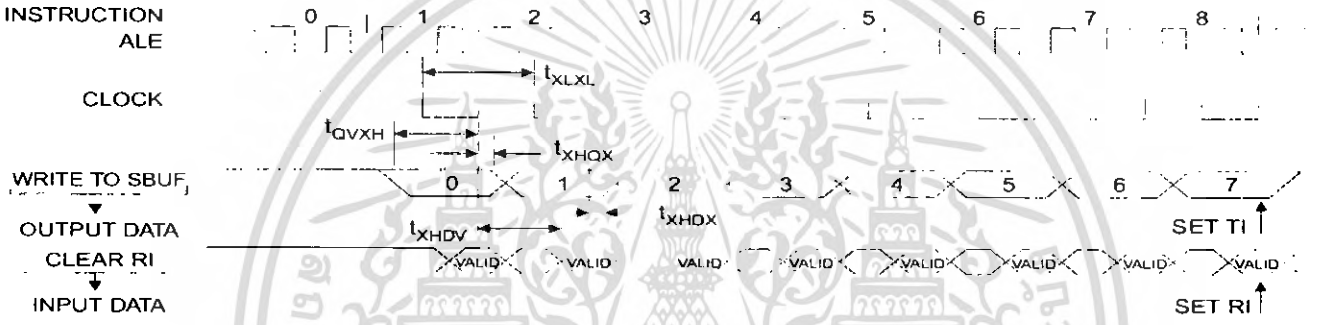
Symbol	Parameter	Min	Max	Units
$1/t_{CLCL}$	Oscillator Frequency	0	24	MHz
t_{CLCL}	Clock Period	41.6		ns
t_{CHCX}	High Time	15		ns
t_{CLCX}	Low Time	15		ns
t_{CLCH}	Rise Time		20	ns
t_{CHCL}	Fall Time		20	ns

Serial Port Timing: Shift Register Mode Test Conditions

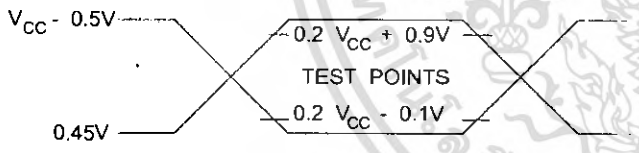
(V_{CC} = 5.0 V ±20%; Load Capacitance = 80 pF)

Symbol	Parameter	12 MHz Osc		Variable Oscillator		Units
		Min	Max	Min	Max	
t_{XLXL}	Serial Port Clock Cycle Time	1.0		$12t_{CLCL}$		µs
t_{QVXH}	Output Data Setup to Clock Rising Edge	700		$10t_{CLCL}$-133		ns
t_{XHQX}	Output Data Hold after Clock Rising Edge	50		$2t_{CLCL}$-117		ns
t_{XHDX}	Input Data Hold after Clock Rising Edge	0		0		ns
t_{XHDV}	Clock Rising Edge to Input Data Valid		700		$10t_{CLCL}$-133	ns

Shift Register Mode Timing Waveforms

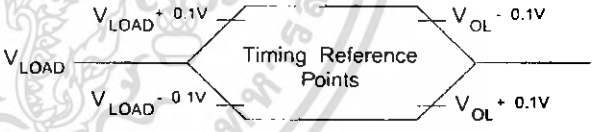


AC Testing Input/Output Waveforms[1]



Note: 1. AC Inputs during testing are driven at V_{CC} - 0.5V for a logic 1 and 0.45V for a logic 0. Timing measurements are made at V_{IH} min. for a logic 1 and V_{IL} max. for a logic 0.

Float Waveforms[1]



Note: 1. For timing purposes, a port pin is no longer floating when a 100 mV change from load voltage occurs. A port pin begins to float when 100 mV change from the loaded V_{OH}/V_{OL} level occurs.

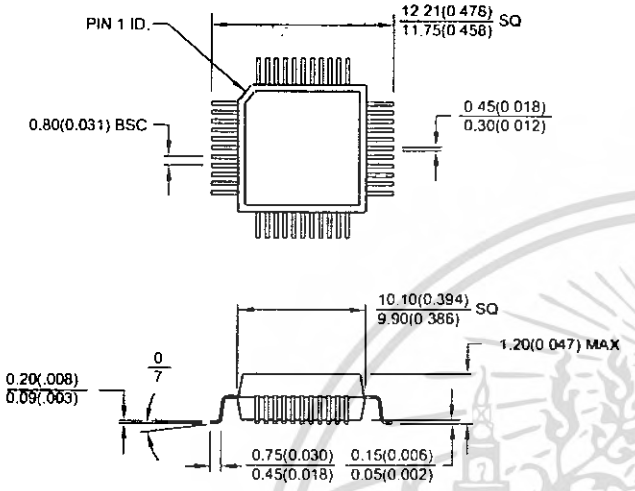
Ordering Information

Speed (MHz)	Power Supply	Ordering Code	Package	Operation Range
12	5V ±20%	AT89C51-12AC	44A	Commercial (0° C to 70° C)
		AT89C51-12JC	44J	
		AT89C51-12PC	40P6	
		AT89C51-12QC	44Q	
		AT89C51-12AI	44A	Industrial (-40° C to 85° C)
		AT89C51-12JI	44J	
		AT89C51-12PI	40P6	
		AT89C51-12QI	44Q	
16	5V ±20%	AT89C51-16AC	44A	Commercial (0° C to 70° C)
		AT89C51-16JC	44J	
		AT89C51-16PC	40P6	
		AT89C51-16QC	44Q	
		AT89C51-16AI	44A	Industrial (-40° C to 85° C)
		AT89C51-16JI	44J	
		AT89C51-16PI	40P6	
		AT89C51-16QI	44Q	
20	5V ±20%	AT89C51-20AC	44A	Commercial (0° C to 70° C)
		AT89C51-20JC	44J	
		AT89C51-20PC	40P6	
		AT89C51-20QC	44Q	
		AT89C51-20AI	44A	Industrial (-40° C to 85° C)
		AT89C51-20JI	44J	
		AT89C51-20PI	40P6	
		AT89C51-20QI	44Q	
24	5V ±20%	AT89C51-24AC	44A	Commercial (0° C to 70° C)
		AT89C51-24JC	44J	
		AT89C51-24PC	40P6	
		AT89C51-24QC	44Q	
		AT89C51-24AI	44A	Industrial (-40° C to 85° C)
		AT89C51-24JI	44J	
		AT89C51-24PI	40P6	
		AT89C51-24QI	44Q	

Package Type	
44A	44-lead, Thin Plastic Gull Wing Quad Flatpack (TQFP)
44J	44-lead, Plastic J-leaded Chip Carrier (PLCC)
40P6	40-lead, 0.600" Wide, Plastic Dual Inline Package (PDIP)
44Q	44-lead, Plastic Gull Wing Quad Flatpack (PQFP)

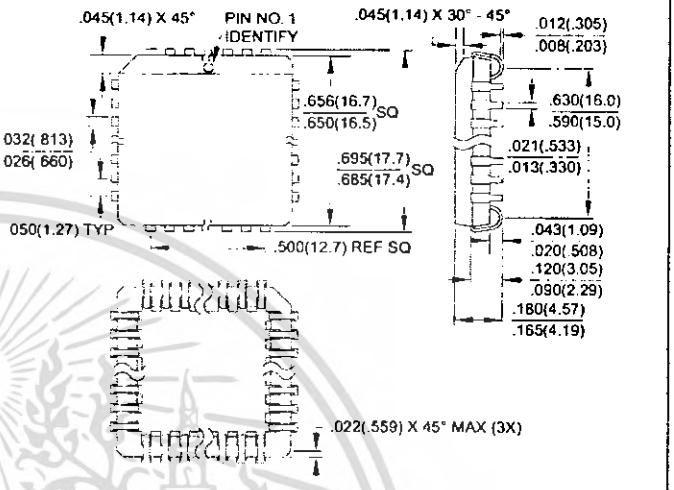
Packaging Information

44A, 44-lead, Thin (1.0 mm) Plastic Gull Wing Quad Flatpack (TQFP)
Dimensions in Millimeters and (Inches)*
JEDEC STANDARD MS-026 ACB

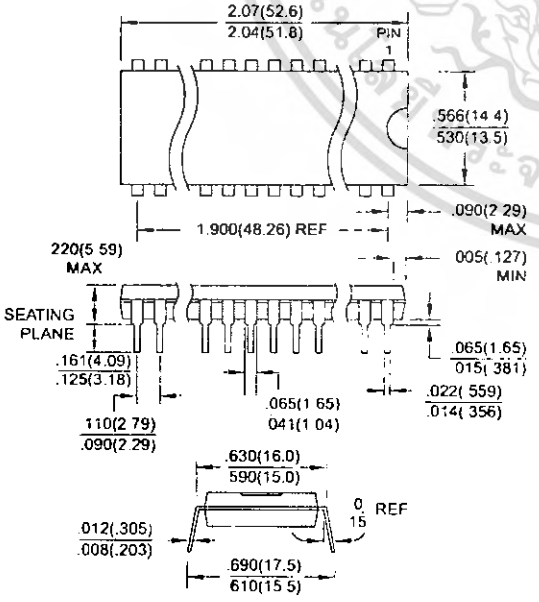


Controlling dimension: millimeters

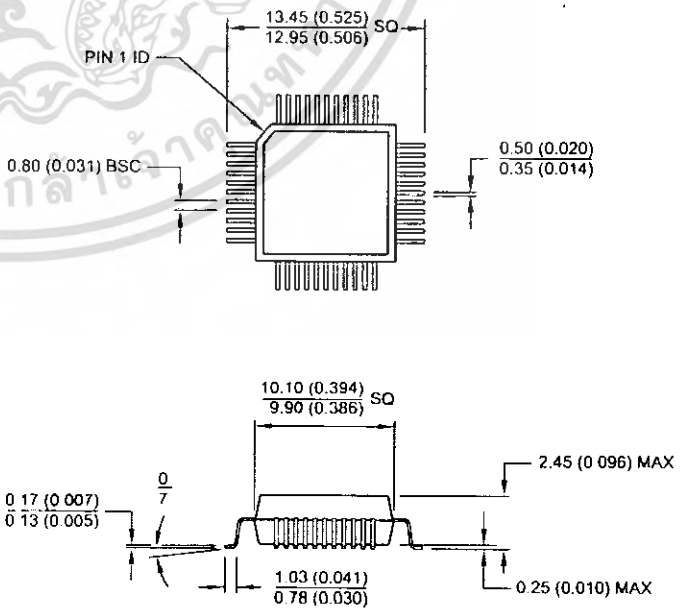
44J, 44-lead, Plastic J-leaded Chip Carrier (PLCC)
Dimensions in Inches and (Millimeters)
JEDEC STANDARD MS-018 AC



40P6, 40-lead, 0.600" Wide, Plastic Dual Inline Package (PDIP)
Dimensions in Inches and (Millimeters)



44Q, 44-lead, Plastic Quad Flat Package (PQFP)
Dimensions in Millimeters and (Inches)*
JEDEC STANDARD MS-022 AB



Controlling dimension: millimeters

Atmel Headquarters

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

Europe
Atmel U.K., Ltd.
Coliseum Business Centre
Riverside Way
Camberley, Surrey GU15 3YL
England
TEL (44) 1276-686-677
FAX (44) 1276-686-697

Asia
Atmel Asia, Ltd.
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimhatsui
East Kowloon
Hong Kong
TEL (852) 2721-9778
FAX (852) 2722-1369

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

Atmel Operations

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

Atmel Rousset
Zone Industrielle
13106 Rousset Cedex
France
TEL (33) 4-4253-6000
FAX (33) 4-4253-6001

Fax-on-Demand
North America:
1-(800) 292-8635
International:
1-(408) 441-0732

e-mail
literature@atmel.com

Web Site
http://www.atmel.com

BBS
1-(408) 436-4309

0265G–02/00/xM

19-4323, Rev 7b, 11/97

MAXIM

+5V-Powered, Multichannel RS-232 Drivers/Receivers

General Description

The MAX220–MAX249 family of line drivers/receivers is intended for all EIA/TIA-232E and V.28/V.24 communications interfaces, particularly applications where ±12V is not available.

These parts are especially useful in battery-powered systems, since their low-power shutdown mode reduces power dissipation to less than 5µW. The MAX225, MAX233, MAX235, and MAX245/MAX246/MAX247 use no external components and are recommended for applications where printed circuit board space is critical.

Applications

Portable Computers

Low-Power Modems

Interface Translation

Battery-Powered RS-232 Systems

Multi-Drop RS-232 Networks

Features

Superior to Bipolar

- **Operate from Single +5V Power Supply (+5V and +12V—MAX231/MAX239)**
- **Low-Power Receive Mode in Shutdown (MAX223/MAX242)**
- **Meet All EIA/TIA-232E and V.28 Specifications**
- **Multiple Drivers and Receivers**
- **3-State Driver and Receiver Outputs**
- **Open-Line Detection (MAX243)**

Ordering Information

PART	TEMP. RANGE	PIN-PACKAGE
MAX220CPE	0°C to +70°C	16 Plastic DIP
MAX220CSE	0°C to +70°C	16 Narrow SO
MAX220CWE	0°C to +70°C	16 Wide SO
MAX220C/D	0°C to +70°C	Dice*
MAX220EPE	-40°C to +85°C	16 Plastic DIP
MAX220ESE	-40°C to +85°C	16 Narrow SO
MAX220EWE	-40°C to +85°C	16 Wide SO
MAX220EJE	-40°C to +85°C	16 CERDIP
MAX220MJE	-55°C to +125°C	16 CERDIP

Ordering Information continued at end of data sheet.

**Contact factory for dice specifications.*

Selection Table

Part Number	Power Supply (V)	No. of RS-232 Drivers/Rx	No. of Ext. Caps	Nominal Cap. Value (µF)	SHDN & Three-State	Rx Active in SHDN	Data Rate (kbps)	Features
MAX220	+5	2/2	4	4.7/10	No		120	Ultra-low-power, industry-standard pinout
MAX222	+5	2/2	4	0.1	Yes	—	200	Low-power shutdown
MAX223 (MAX213)	+5	4/5	4	1.0 (0.1)	Yes	✓	120	MAX241 and receivers active in shutdown
MAX225	+5	5/5	0	—	Yes	✓	120	Available in SO
MAX230 (MAX200)	+5	5/0	4	1.0 (0.1)	Yes	—	120	5 drivers with shutdown
MAX231 (MAX201)	+5 and +7.5 to +13.2	2/2	2	1.0 (0.1)	No		120	Standard +5/+12V or battery supplies; same functions as MAX232
MAX232 (MAX202)	+5	2/2	4	1.0 (0.1)	No	—	120 (64)	Industry standard
MAX232A	+5	2/2	4	0.1	No	—	200	Higher slew rate, small caps
MAX233 (MAX203)	+5	2/2	0	-	No		120	No external caps
MAX233A	+5	2/2	0	—	No		200	No external caps, high slew rate
MAX234 (MAX204)	+5	4/0	4	1.0 (0.1)	No	—	120	Replaces 1488
MAX235 (MAX205)	+5	5/5	0	—	Yes	—	120	No external caps
MAX236 (MAX206)	+5	4/3	4	1.0 (0.1)	Yes		120	Shutdown, three state
MAX237 (MAX207)	+5	5/3	4	1.0 (0.1)	No	—	120	Complements IBM PC serial port
MAX238 (MAX208)	+5	4/4	4	1.0 (0.1)	No	—	120	Replaces 1488 and 1489
MAX239 (MAX209)	+5 and +7.5 to +13.2	3/5	2	1.0 (0.1)	No	—	120	Standard +5/+12V or battery supplies; single-package solution for IBM PC serial port
MAX240	+5	5/5	4	1.0	Yes		120	DIP or flatpack package
MAX241 (MAX211)	+5	4/5	4	1.0 (0.1)	Yes	—	120	Complete IBM PC serial port
MAX242	+5	2/2	4	0.1	Yes	✓	200	Separate shutdown and enable
MAX243	+5	2/2	4	0.1	No		200	Open-line detection simplifies cabling
MAX244	+5	8/10	4	1.0	No	—	120	High slew rate
MAX245	+5	8/10	0	- -	Yes	✓	120	High slew rate, int. caps, two shutdown modes
MAX246	+5	8/10	0	- -	Yes	✓	120	High slew rate, int. caps, three shutdown modes
MAX247	+5	8/9	0	—	Yes	✓	120	High slew rate, int. caps, nine operating modes
MAX248	+5	8/8	4	1.0	Yes	✓	120	High slew rate, selective half-chip enables
MAX249	+5	6/10	4	1.0	Yes	✓	120	Available in quad flatpack package

ABSOLUTE MAXIMUM RATINGS—MAX220/222/232A/233A/242/243

Supply Voltage (V_{CC}) ..... -0.3V to +6V
Input Voltages
T_{IN} ..... -0.3V to (V_{CC} + 0.3V)
R_{IN} ..... ±30V
T_{OUT} (Note 1) ..... ±15V
Output Voltages
T_{OUT} ..... ±15V
R_{OUT} ..... -0.3V to (V_{CC} + 0.3V)
Driver/Receiver Output Short Circuited to GND ..... Continuous
Continuous Power Dissipation (T_A = +70°C)
16-Pin Plastic DIP (derate 10.53mW/°C above +70°C) 842mW
18-Pin Plastic DIP (derate 11.11mW/°C above +70°C) 889mW
20-Pin Plastic DIP (derate 8.00mW/°C above +70°C) 440mW
16-Pin Narrow SO (derate 8.70mW/°C above +70°C) 696mW
16-Pin Wide SO (derate 9.52mW/°C above +70°C) 762mW
18-Pin Wide SO (derate 9.52mW/°C above +70°C) 762mW
20-Pin Wide SO (derate 10.00mW/°C above +70°C) 800mW
20-Pin SSOP (derate 8.00mW/°C above +70°C) 640mW
16-Pin CERDIP (derate 10.00mW/°C above +70°C) 800mW
18-Pin CERDIP (derate 10.53mW/°C above +70°C) 842mW
Operating Temperature Ranges
MAX2__AC__, MAX2__C__ ..... 0°C to +70°C
MAX2__AE__, MAX2__E__ ..... -40°C to +85°C
MAX2__AM__, MAX2__M__ ..... -55°C to +125°C
Storage Temperature Range ..... -65°C to +160°C
Lead Temperature (soldering, 10sec) ..... +300°C

Note 1: Input voltage measured with T_{OUT} in high-impedance state, $\overline{SHDN}$ or V_{CC} = 0V.

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of the specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

ELECTRICAL CHARACTERISTICS—MAX220/222/232A/233A/242/243

(V_{CC} = +5V ±10%, C1–C4 = 0.1µF, T_A = T_{MIN} to T_{MAX}, unless otherwise noted.)

PARAMETER	CONDITIONS		MIN	TYP	MAX	UNITS
RS-232 TRANSMITTERS						
Output Voltage Swing	All transmitter outputs loaded with 3kΩ to GND		±5	±8		V
Input Logic Threshold Low				1.4	0.8	V
Input Logic Threshold High			2	1.4		V
Logic Pull-Up/Input Current	Normal operation			5	40	µA
	$\overline{SHDN}$ = 0V, MAX222/242 shutdown			±0.01	±1	
Output Leakage Current	V_{CC} = 5.5V, $\overline{SHDN}$ = 0V, V_{OUT} = ±15V, MAX222/242			±0.01	±10	µA
	V_{CC} = $\overline{SHDN}$ = 0V, V_{OUT} = ±15V			±0.01	±10	
Data Rate	All except MAX220, normal operation			200	116	kbits/sec
	MAX220			22	20	
Transmitter Output Resistance	V_{CC} = V_+ = V_- = 0V, V_{OUT} = ±2V		300	10M		Ω
Output Short-Circuit Current	V_{OUT} = 0V		±7	±22		mA
RS-232 RECEIVERS						
RS-232 Input Voltage Operating Range					±30	V
RS-232 Input Threshold Low	V_{CC} = 5V	All except MAX243 $R2_{IN}$	0.8	1.3		V
		MAX243 $R2_{IN}$ (Note 2)	-3			
RS-232 Input Threshold High	V_{CC} = 5V	All except MAX243 $R2_{IN}$		1.8	2.4	V
		MAX243 $R2_{IN}$ (Note 2)		-0.5	-0.1	
RS-232 Input Hysteresis	All except MAX243, V_{CC} = 5V, no hysteresis in shdn		0.2	0.5	1	V
	MAX243			1		
RS-232 Input Resistance			3	5	7	kΩ
TTL/CMOS Output Voltage Low	I_{OUT} = 3.2mA			0.2	0.4	V
TTL/CMOS Output Voltage High	I_{OUT} = -1.0mA		3.5	V_{CC} - 0.2		V
TTL/CMOS Output Short-Circuit Current	Sourcing V_{OUT} = GND		-2	-10		mA
	Shrinking V_{OUT} = V_{CC}		10	30		
TTL/CMOS Output Leakage Current	$\overline{SHDN}$ = V_{CC} or $\overline{EN}$ = V_{CC} ($\overline{SHDN}$ = 0V for MAX222), 0V ≤ V_{OUT} ≤ V_{CC}			±0.05	±10	µA

ELECTRICAL CHARACTERISTICS—MAX220/222/232A/233A/242/243 (continued)

(V_{CC} = +5V ±10%, C1–C4 = 0.1µF, T_A = T_{MIN} to T_{MAX}, unless otherwise noted.)

PARAMETER	CONDITIONS		MIN	TYP	MAX	UNITS
$\overline{EN}$ Input Threshold Low	MAX242			1.4	0.8	V
$\overline{EN}$ Input Threshold High	MAX242		2.0	1.4		V
Operating Supply Voltage			4.5		5.5	V
V_{CC} Supply Current ($\overline{SHDN}$ = V_{CC}), Figures 5, 6, 11, 19	No load	MAX220		0.5	2	mA
		MAX222/232A/233A/242/243		4	10	
	3kΩ load both inputs	MAX220		12		
		MAX222/232A/233A/242/243		15		
Shutdown Supply Current	MAX222/242	T_A = +25°C		0.1	10	µA
		T_A = 0°C to +70°C		2	50	
		T_A = -40°C to +85°C		2	50	
		T_A = -55°C to +125°C		35	100	
$\overline{SHDN}$ Input Leakage Current	MAX222/242				±1	µA
$\overline{SHDN}$ Threshold Low	MAX222/242			1.4	0.8	V
$\overline{SHDN}$ Threshold High	MAX222/242		2.0	1.4		V
Transition Slew Rate	C_L = 50pF to 2500pF, R_L = 3kΩ to 7kΩ, V_{CC} = 5V, T_A = +25°C, measured from +3V to -3V or -3V to +3V	MAX222/232A/233A/242/243	6	12	30	V/µs
		MAX220	1.5	3	30	
Transmitter Propagation Delay TLL to RS-232 (normal operation), Figure 1	t_{PHLT}	MAX222/232A/233A/242/243		1.3	3.5	µs
		MAX220		4	10	
	t_{PLHT}	MAX222/232A/233A/242/243		1.5	3.5	
		MAX220		5	10	
Receiver Propagation Delay RS-232 to TLL (normal operation), Figure 2	t_{PHLR}	MAX222/232A/233A/242/243		0.5	1	µs
		MAX220		0.6	3	
	t_{PLHR}	MAX222/232A/233A/242/243		0.6	1	
		MAX220		0.8	3	
Receiver Propagation Delay RS-232 to TLL (shutdown), Figure 2	t_{PHLS}	MAX242		0.5	10	µs
	t_{PLHS}	MAX242		2.5	10	
Receiver-Output Enable Time, Figure 3	t_{ER}	MAX242		125	500	ns
Receiver-Output Disable Time, Figure 3	t_{DR}	MAX242		160	500	ns
Transmitter-Output Enable Time ($\overline{SHDN}$ goes high), Figure 4	t_{ET}	MAX222/242, 0.1µF caps (includes charge-pump start-up)		250		µs
Transmitter-Output Disable Time ($\overline{SHDN}$ goes low), Figure 4	t_{DT}	MAX222/242, 0.1µF caps		600		ns
Transmitter + to - Propagation Delay Difference (normal operation)	t_{PHLT} - t_{PLHT}	MAX222/232A/233A/242/243		300		ns
		MAX220		2000		
Receiver + to - Propagation Delay Difference (normal operation)	t_{PHLR} - t_{PLHR}	MAX222/232A/233A/242/243		100		ns
		MAX220		225		

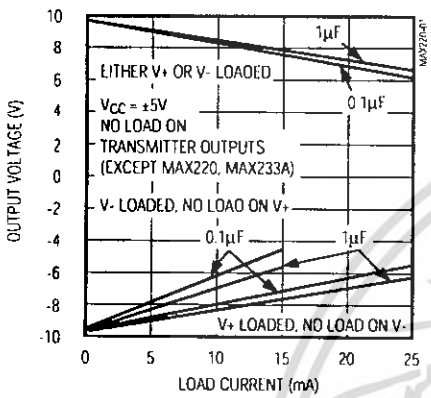
Note 2: MAX243 $R2_{OUT}$ is guaranteed to be low when $R2_{IN}$ is ≥ 0V or is floating.

+5V-Powered, Multichannel RS-232 Drivers/Receivers

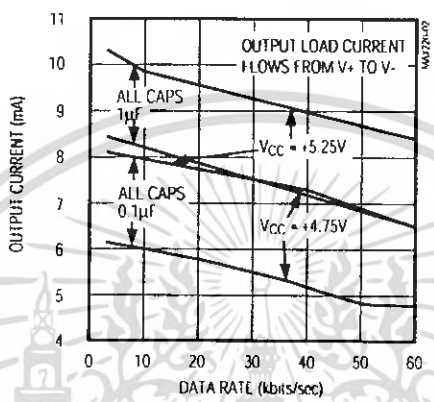
Typical Operating Characteristics

MAX220/MAX222/MAX232A/MAX233A/MAX242/MAX243

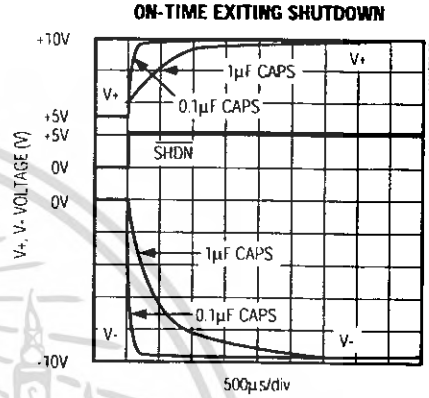
OUTPUT VOLTAGE vs. LOAD CURRENT



AVAILABLE OUTPUT CURRENT vs. DATA RATE



MAX222/MAX242 ON-TIME EXITING SHUTDOWN



ABSOLUTE MAXIMUM RATINGS—MAX223/MAX230–MAX241

V_{CC} ..... -0.3V to +6V
V+ ..... (V_{CC} - 0.3V) to +14V
V- ..... +0.3V to -14V
Input Voltages
T_{IN} ..... -0.3V to (V_{CC} + 0.3V)
R_{IN} ..... ±30V
Output Voltages
T_{OUT} ..... (V+ + 0.3V) to (V- - 0.3V)
R_{OUT} ..... -0.3V to (V_{CC} + 0.3V)
Short-Circuit Duration, T_{OUT} ..... Continuous
Continuous Power Dissipation (T_A = +70°C)
14-Pin Plastic DIP (derate 10.00mW/°C above +70°C) 800mW
16-Pin Plastic DIP (derate 10.53mW/°C above +70°C) 842mW
20-Pin Plastic DIP (derate 11.11mW/°C above +70°C) 889mW
24-Pin Narrow Plastic DIP
(derate 13.33mW/°C above +70°C) 1.07W
24-Pin Plastic DIP (derate 9.09mW/°C above +70°C) 500mW
16-Pin Wide SO (derate 9.52mW/°C above +70°C) 762mW
20-Pin Wide SO (derate 10.00mW/°C above +70°C) 800mW
24-Pin Wide SO (derate 11.76mW/°C above +70°C) 941mW
28-Pin Wide SO (derate 12.50mW/°C above +70°C) 1W
44-Pin Plastic FP (derate 11.11mW/°C above +70°C) 889mW
14-Pin CERDIP (derate 9.09mW/°C above +70°C) 727mW
16-Pin CERDIP (derate 10.00mW/°C above +70°C) 800mW
20-Pin CERDIP (derate 11.11mW/°C above +70°C) 889mW
24-Pin Narrow CERDIP
(derate 12.50mW/°C above +70°C) 1W
24-Pin Sidebraze (derate 20.0mW/°C above +70°C) 1.6W
28-Pin SSOP (derate 9.52mW/°C above +70°C) 762mW
Operating Temperature Ranges
MAX2_ _ C _ _ ..... 0°C to +70°C
MAX2_ _ E _ _ ..... -40°C to +85°C
MAX2_ _ M _ _ ..... -55°C to +125°C
Storage Temperature Range ..... -65°C to +160°C
Lead Temperature (soldering, 10sec) ..... +300°C

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of the specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

ELECTRICAL CHARACTERISTICS—MAX223/MAX230–MAX241

(MAX223/230/232/234/236/237/238/240/241, V_{CC} = +5V ±10, MAX233/MAX235, V_{CC} = 5V ±5%, C1–C4 = 1.0µF; MAX231/MAX239, V_{CC} = 5V ±10%; V+ = 7.5V to 13.2V, T_A = T_{MIN} to T_{MAX}; unless otherwise noted.)

PARAMETER	CONDITIONS		MIN	TYP	MAX	UNITS
Output Voltage Swing	All transmitter outputs loaded with 3kΩ to ground		±5.0	±7.3		V
V_{CC} Power-Supply Current	No load, T_A = +25°C	MAX232/233		5	10	mA
		MAX223/230/234–238/240/241		7	15	
		MAX231/239		0.4	1	
V+ Power-Supply Current		MAX231		1.8	5	mA
		MAX239		5	15	
Shutdown Supply Current	T_A = +25°C	MAX223		15	50	µA
		MAX230/235/236/240/241		1	10	
Input Logic Threshold Low	T_{IN}; EN, $\overline{SHDN}$ (MAX233), $\overline{EN}$, SHDN (MAX230/235–241)				0.8	V
Input Logic Threshold High	T_{IN}		2.0			V
	EN, $\overline{SHDN}$ (MAX223), $\overline{EN}$, SHDN (MAX230/235/236/240/241)		2.4			
Logic Pull-Up Current	T_{IN} = 0V			1.5	200	µA
Receiver Input Voltage Operating Range			-30		30	V

+5V-Powered, Multichannel RS-232 Drivers/Receivers

ELECTRICAL CHARACTERISTICS—MAX223/MAX230–MAX241 (continued)

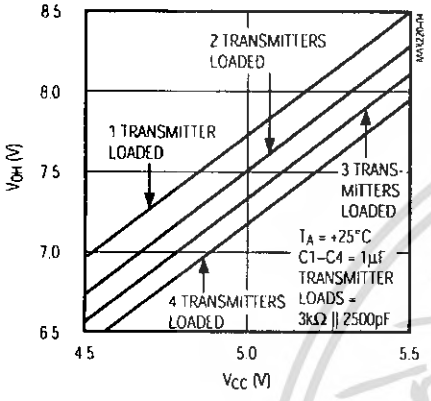
(MAX223/230/232/234/236/237/238/240/241, V_{CC} = +5V ±10, MAX233/MAX235, V_{CC} = 5V ±5%, C1–C4 = 1.0µF; MAX231/MAX239, V_{CC} = 5V ±10%, V+ = 7.5V to 13.2V; T_A = T_{MIN} to T_{MAX}, unless otherwise noted.)

PARAMETER	CONDITIONS			MIN	TYP	MAX	UNITS
RS-232 Input Threshold Low	T_A = +25°C, V_{CC} = 5V	Normal operation $\overline{SHDN}$ = 5V (MAX223) SHDN = 0V (MAX235/236/240/241)		0.8	1.2		V
		Shutdown (MAX223) $\overline{SHDN}$ = 0V, EN = 5V ($R4_{IN}$, $R5_{IN}$)		0.6	1.5		
RS-232 Input Threshold High	T_A = +25°C, V_{CC} = 5V	Normal operation $\overline{SHDN}$ = 5V (MAX223) SHDN = 0V (MAX235/236/240/241)			1.7	2.4	V
		Shutdown (MAX223) $\overline{SHDN}$ = 0V, EN = 5V ($R4_{IN}$, $R5_{IN}$)			1.5	2.4	
RS-232 Input Hysteresis	V_{CC} = 5V, no hysteresis in shutdown			0.2	0.5	1.0	V
RS-232 Input Resistance	T_A = +25°C, V_{CC} = 5V			3	5	7	kΩ
TTL/CMOS Output Voltage Low	I_{OUT} = 1.6mA (MAX231/232/233, I_{OUT} = 3.2mA)					0.4	V
TTL/CMOS Output Voltage High	I_{OUT} = -1mA			3.5	V_{CC} - 0.4		V
TTL/CMOS Output Leakage Current	0V ≤ R_{OUT} ≤ V_{CC}; EN = 0V (MAX223), $\overline{EN}$ = V_{CC} (MAX235–241)				0.05	±10	µA
Receiver Output Enable Time	Normal operation	MAX223			600		ns
		MAX235/236/239/240/241			400		
Receiver Output Disable Time	Normal operation	MAX223			900		ns
		MAX235/236/239/240/241			250		
Propagation Delay	RS-232 IN to TTL/CMOS OUT, C_L = 150pF	Normal operation			0.5	10	µs
		$\overline{SHDN}$ = 0V (MAX223)	t_{PHLS}		4	40	
			t_{PLHS}		6	40	
Transition Region Slew Rate	MAX223/MAX230/MAX234–241, T_A = +25°C, V_{CC} = 5V, R_L = 3kΩ to 7kΩ, C_L = 50pF to 2500pF, measured from +3V to -3V or -3V to +3V			3	5.1	30	V/µs
	MAX231/MAX232/MAX233, T_A = +25°C, V_{CC} = 5V, R_L = 3kΩ to 7kΩ, C_L = 50pF to 2500pF, measured from +3V to -3V or -3V to +3V				4	30	
Transmitter Output Resistance	V_{CC} = V+ = V- = 0V, V_{OUT} = ±2V			300			Ω
Transmitter Output Short-Circuit Current					±10		mA

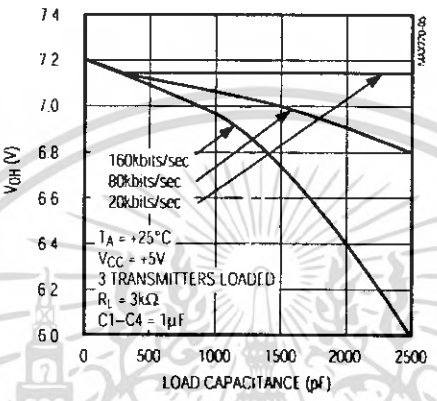
Typical Operating Characteristics

MAX223/MAX230–MAX241

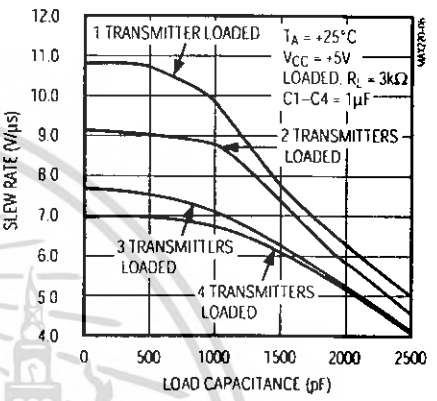
TRANSMITTER OUTPUT VOLTAGE (V_{OH}) vs. V_{CC}



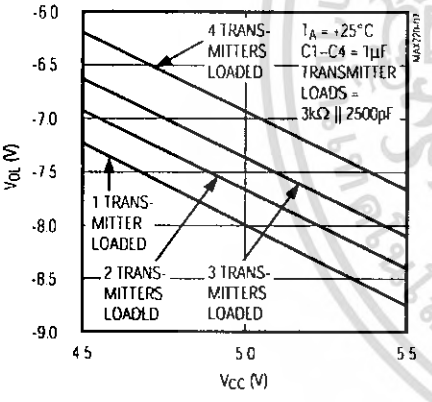
TRANSMITTER OUTPUT VOLTAGE (V_{OH}) vs. LOAD CAPACITANCE AT DIFFERENT DATA RATES



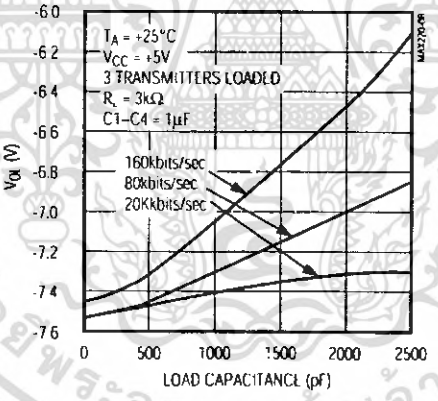
TRANSMITTER SLEW RATE vs. LOAD CAPACITANCE



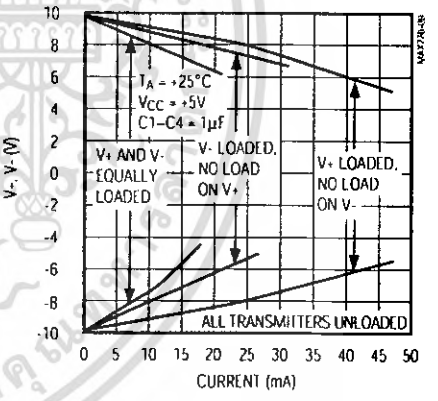
TRANSMITTER OUTPUT VOLTAGE (V_{OL}) vs. V_{CC}



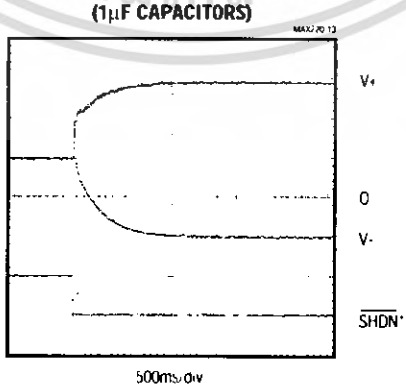
TRANSMITTER OUTPUT VOLTAGE (V_{OL}) vs. LOAD CAPACITANCE AT DIFFERENT DATA RATES



TRANSMITTER OUTPUT VOLTAGE (V+, V-) vs. LOAD CURRENT



V+, V- WHEN EXITING SHUTDOWN (1µF CAPACITORS)



*SHUTDOWN POLARITY IS REVERSED FOR NON MAX241 PARTS

+5V-Powered, Multichannel RS-232 Drivers/Receivers

ABSOLUTE MAXIMUM RATINGS—MAX225/MAX244–MAX249

Supply Voltage (V_{CC}) ..... -0.3V to +6V
Input Voltages
T_{IN}, $\overline{ENA}$, $\overline{ENB}$, $\overline{ENR}$, $\overline{ENT}$, $\overline{ENRA}$, $\overline{ENRB}$, $\overline{ENTA}$, $\overline{ENTB}$ ..... -0.3V to (V_{CC} + 0.3V)
R_{IN} ..... ±25V
T_{OUT} (Note 3) ..... ±15V
R_{OUT} ..... -0.3V to (V_{CC} + 0.3V)
Short Circuit (one output at a time)
T_{OUT} to GND ..... Continuous
R_{OUT} to GND ..... Continuous

Continuous Power Dissipation (T_A = +70°C)
28-Pin Wide SO (derate 12.50mW/°C above +70°C) 1W
40-Pin Plastic DIP (derate 11.11mW/°C above +70°C) ... 611mW
44-Pin PLCC (derate 13.33mW/°C above +70°C) 1.07W
Operating Temperature Ranges
MAX225C_ _, MAX24_C_ _ ..... 0°C to +70°C
MAX225E_ _, MAX24_E_ _ ..... -40°C to +85°C
Storage Temperature Range ..... -65°C to +160°C
Lead Temperature (soldering,10sec) ..... +300°C

Note 3: Input voltage measured with transmitter output in a high-impedance state, shutdown, or V_{CC} = 0V.

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of the specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

ELECTRICAL CHARACTERISTICS—MAX225/MAX244–MAX249

(MAX225, V_{CC} = 5.0V ±5%; MAX244–MAX249, V_{CC} = +5.0V ±10%, external capacitors C1–C4 = 1µF, T_A = T_{MIN} to T_{MAX}; unless otherwise noted.)

PARAMETER	CONDITIONS		MIN	TYP	MAX	UNITS
RS-232 TRANSMITTERS						
Input Logic Threshold Low				1.4	0.8	V
Input Logic Threshold High			2	1.4		V
Logic Pull-Up/Input Current	Tables 1a–1d	Normal operation		10	50	µA
		Shutdown		±0.01	±1	
Data Rate	Tables 1a–1d, normal operation			120	64	kbits/sec
Output Voltage Swing	All transmitter outputs loaded with 3kΩ to GND		±5	±7.5		V
Output Leakage Current (shutdown)	Tables 1a–1d	$\overline{ENA}$, $\overline{ENB}$, $\overline{ENT}$, $\overline{ENTA}$, $\overline{ENTB}$ = V_{CC}, V_{OUT} = ±15V		±0.01	±25	µA
		V_{CC} = 0V, V_{OUT} = ±15V		±0.01	±25	
Transmitter Output Resistance	V_{CC} = V+ = V- = 0V, V_{OUT} = ±2V (Note 4)		300	10M		Ω
Output Short-Circuit Current	V_{OUT} = 0V		±7	±30		mA
RS-232 RECEIVERS						
RS-232 Input Voltage Operating Range					±25	V
RS-232 Input Threshold Low	V_{CC} = 5V		0.8	1.3		V
RS-232 Input Threshold High	V_{CC} = 5V			1.8	2.4	V
RS-232 Input Hysteresis	V_{CC} = 5V		0.2	0.5	1.0	V
RS-232 Input Resistance			3	5	7	kΩ
TTL/CMOS Output Voltage Low	I_{OUT} = 3.2mA			0.2	0.4	V
TTL/CMOS Output Voltage High	I_{OUT} = -1.0mA		3.5	V_{CC} - 0.2		V
TTL/CMOS Output Short-Circuit Current	Sourcing V_{OUT} = GND		-2	-10		mA
	Shrinking V_{OUT} = V_{CC}		10	30		
TTL/CMOS Output Leakage Current	Normal operation, outputs disabled, Tables 1a–1d, 0V ≤ V_{OUT} ≤ V_{CC}, $\overline{ENR}_{\_}$ = V_{CC}			±0.05	±0.10	µA

ELECTRICAL CHARACTERISTICS—MAX225/MAX244–MAX249 (continued)

(MAX225, V_{CC} = 5.0V ±5%; MAX244–MAX249, V_{CC} = +5.0V ±10%, external capacitors C1–C4 = 1µF; T_A = T_{MIN} to T_{MAX}, unless otherwise noted.)

PARAMETER	CONDITIONS		MIN	TYP	MAX	UNITS
POWER SUPPLY AND CONTROL LOGIC						
Operating Supply Voltage		MAX225	4.75		5.25	V
		MAX244–MAX249	4.5		5.5	
V_{CC} Supply Current (normal operation)	No load	MAX225		10	20	mA
		MAX244–MAX249		11	30	
	3kΩ loads on all outputs	MAX225		40		
		MAX244–MAX249		57		
Shutdown Supply Current	T_A = +25°C			8	25	µA
	T_A = T_{MIN} to T_{MAX}				50	
Control Input	Leakage current				±1	µA
	Threshold low			1.4	0.8	V
	Threshold high		2.4	1.4		
AC CHARACTERISTICS						
Transition Slew Rate	C_L = 50pF to 2500pF, R_L = 3kΩ to 7kΩ, V_{CC} = 5V, T_A = +25°C, measured from +3V to -3V or -3V to +3V		5	10	30	V/µs
Transmitter Propagation Delay TLL to RS-232 (normal operation), Figure 1	t_{PHLT}			1.3	3.5	µs
	t_{PLHT}			1.5	3.5	
Receiver Propagation Delay TLL to RS-232 (normal operation), Figure 2	t_{PHLR}			0.6	1.5	µs
	t_{PLHR}			0.6	1.5	
Receiver Propagation Delay TLL to RS-232 (low-power mode), Figure 2	t_{PHLS}			0.6	10	µs
	t_{PLHS}			3.0	10	
Transmitter + to - Propagation Delay Difference (normal operation)	t_{PHLT} - t_{PLHT}			350		ns
Receiver + to - Propagation Delay Difference (normal operation)	t_{PHLR} - t_{PLHR}			350		ns
Receiver-Output Enable Time, Figure 3	t_{ER}			100	500	ns
Receiver-Output Disable Time, Figure 3	t_{DR}			100	500	ns
Transmitter Enable Time	t_{ET}	MAX246–MAX249 (excludes charge-pump start-up)		5		µs
		MAX225/MAX245–MAX249 (includes charge-pump start-up)		10		ms
Transmitter Disable Time, Figure 4	t_{DT}			100		ns

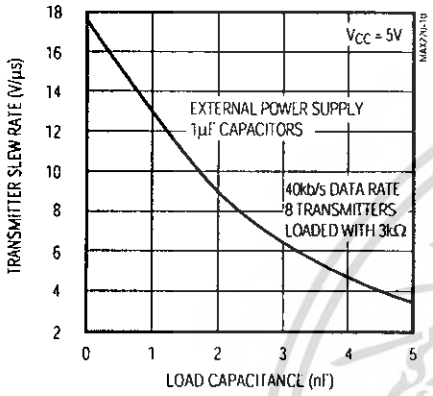
Note 4: The 300Ω minimum specification complies with EIA/TIA-232E, but the actual resistance when in shutdown mode or V_{CC} = 0V is 10MΩ as is implied by the leakage specification

+5V-Powered, Multichannel RS-232 Drivers/Receivers

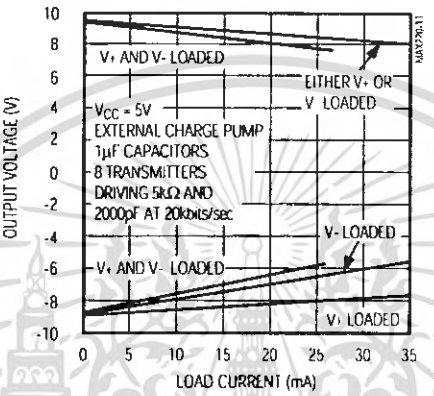
Typical Operating Characteristics

MAX225/MAX244–MAX249

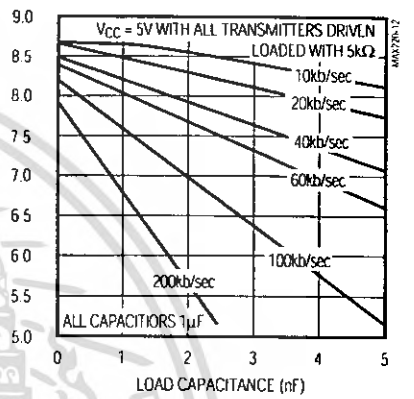
TRANSMITTER SLEW RATE vs. LOAD CAPACITANCE



OUTPUT VOLTAGE vs. LOAD CURRENT FOR V+ AND V-



TRANSMITTER OUTPUT VOLTAGE (V+, V-) vs. LOAD CAPACITANCE AT DIFFERENT DATA RATES



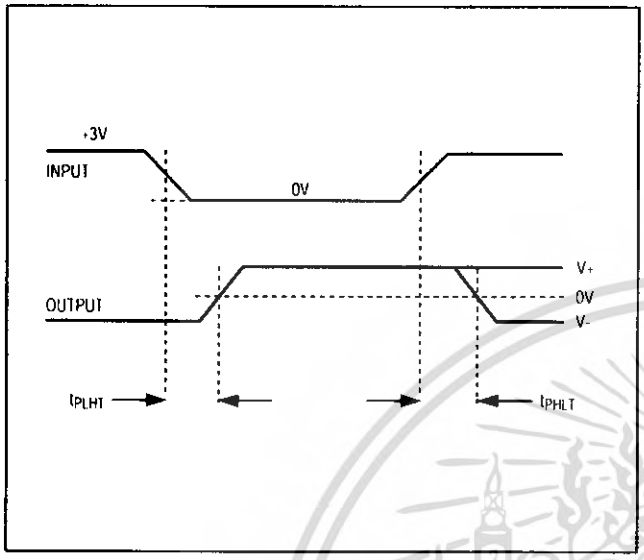


Figure 1. Transmitter Propagation-Delay Timing

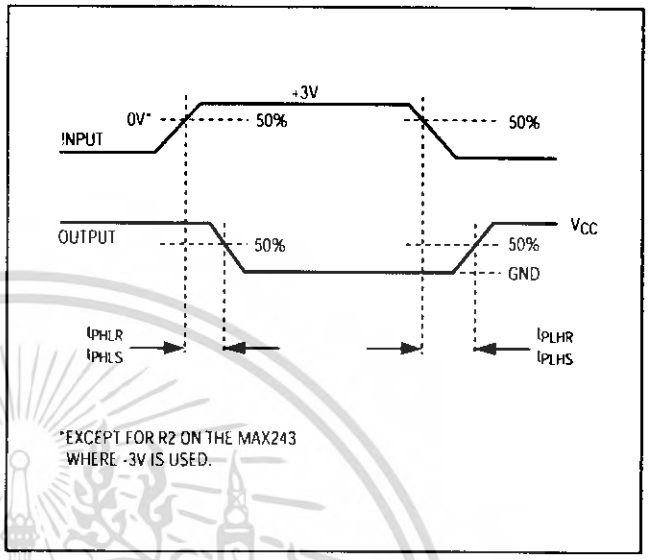


Figure 2. Receiver Propagation-Delay Timing

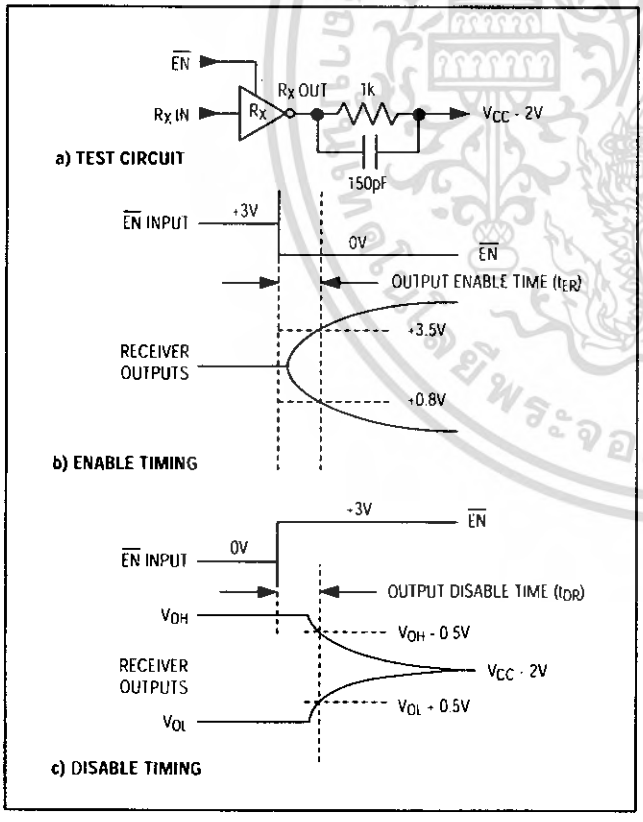


Figure 3 Receiver-Output Enable and Disable Timing

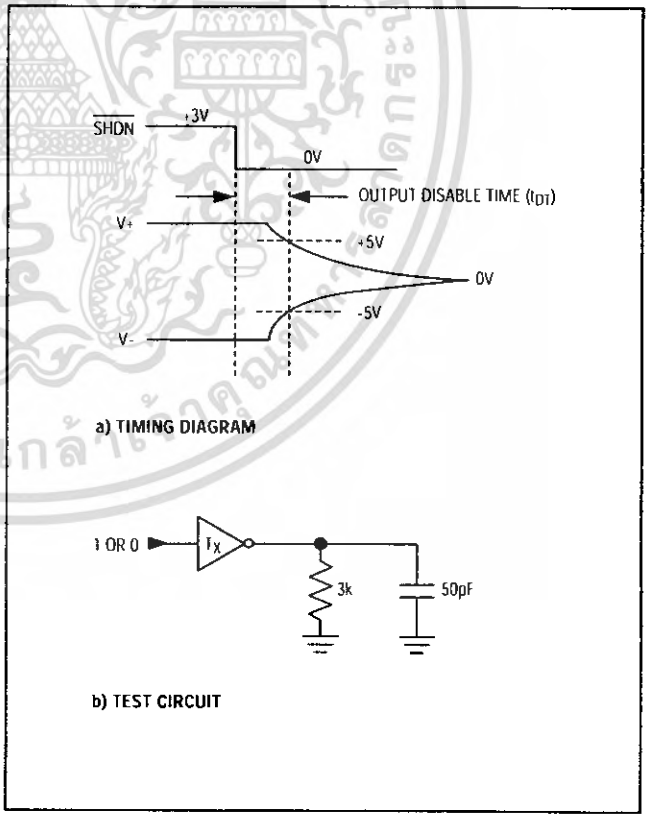


Figure 4 Transmitter-Output Disable Timing

Table 1a. MAX245 Control Pin Configurations

$\overline{ENT}$	$\overline{ENR}$	OPERATION STATUS	TRANSMITTERS	RECEIVERS
0	0	Normal Operation	All Active	All Active
0	1	Normal Operation	All Active	All 3-State
1	0	Shutdown	All 3-State	All Low-Power Receive Mode
1	1	Shutdown	All 3-State	All 3-State

Table 1b. MAX245 Control Pin Configurations

$\overline{ENT}$	$\overline{ENR}$	OPERATION STATUS	TRANSMITTERS		RECEIVERS	
			TA1–TA4	TB1–TB4	RA1–RA5	RB1–RB5
0	0	Normal Operation	All Active	All Active	All Active	All Active
0	1	Normal Operation	All Active	All Active	RA1–RA4 3-State, RA5 Active	RB1–RB4 3-State, RB5 Active
1	0	Shutdown	All 3-State	All 3-State	All Low-Power Receive Mode	All Low-Power Receive Mode
1	1	Shutdown	All 3-State	All 3-State	RA1–RA4 3-State, RA5 Low-Power Receive Mode	RB1–RB4 3-State, RB5 Low-Power Receive Mode

Table 1c. MAX246 Control Pin Configurations

$\overline{ENA}$	$\overline{ENB}$	OPERATION STATUS	TRANSMITTERS		RECEIVERS	
			TA1–TA4	TB1–TB4	RA1–RA5	RB1–RB5
0	0	Normal Operation	All Active	All Active	All Active	All Active
0	1	Normal Operation	All Active	All 3-State	All Active	RB1–RB4 3-State, RB5 Active
1	0	Shutdown	All 3-State	All Active	RA1–RA4 3-State, RA5 Active	All Active
1	1	Shutdown	All 3-State	All 3-State	RA1–RA4 3-State, RA5 Low-Power Receive Mode	RB1–RB4 3-State, RA5 Low-Power Receive Mode

Table 1d. MAX247/MAX248/MAX249 Control Pin Configurations

$\overline{ENTA}$	$\overline{ENTB}$	$\overline{ENRA}$	$\overline{ENRB}$	OPERATION STATUS		TRANSMITTERS		RECEIVERS	
					MAX247	TA1–TA4	TB1–TB4	RA1–RA4	RB1–RB5
					MAX248	TA1–TA4	TB1–TB4	RA1–RA4	RB1–RB4
					MAX249	TA1–TA3	TB1–TB3	RA1–RA5	RB1–RB5
0	0	0	0	Normal Operation		All Active	All Active	All Active	All Active
0	0	0	1	Normal Operation		All Active	All Active	All Active	All 3-State, except RB5 stays active on MAX247
0	0	1	0	Normal Operation		All Active	All Active	All 3-State	All Active
0	0	1	1	Normal Operation		All Active	All Active	All 3-State	All 3-State, except RB5 stays active on MAX247
0	1	0	0	Normal Operation		All Active	All 3-State	All Active	All Active
0	1	0	1	Normal Operation		All Active	All 3-State	All Active	All 3-State, except RB5 stays active on MAX247
0	1	1	0	Normal Operation		All Active	All 3-State	All 3-State	All Active
0	1	1	1	Normal Operation		All Active	All 3-State	All 3-State	All 3-State, except B5 stays active on MAX247
1	0	0	0	Normal Operation		All 3-State	All Active	All Active	All Active
1	0	0	1	Normal Operation		All 3-State	All Active	All Active	All 3-State, except RB5 stays active on MAX247
1	0	1	0	Normal Operation		All 3-State	All Active	All 3-State	All Active
1	0	1	1	Normal Operation		All 3-State	All Active	All 3-State	All 3-State, except RB5 stays active on MAX247
1	1	0	0	Shutdown		All 3-State	All 3-State	Low-Power Receive Mode	Low-Power Receive Mode
1	1	0	1	Shutdown		All 3-State	All 3-State	Low-Power Receive Mode	All 3-State, except RB5 stays active on MAX247
1	1	1	0	Shutdown		All 3-State	All 3-State	All 3-State	Low-Power Receive Mode
1	1	1	1	Shutdown		All 3-State	All 3-State	All 3-State	All 3-State, except RB5 stays active on MAX247

+5V-Powered, Multichannel RS-232 Drivers/Receivers

Detailed Description

The MAX220–MAX249 contain four sections: dual charge-pump DC-DC voltage converters, RS-232 drivers, RS-232 receivers, and receiver and transmitter enable control inputs.

Dual Charge-Pump Voltage Converter

The MAX220–MAX249 have two internal charge-pumps that convert +5V to ±10V (unloaded) for RS-232 driver operation. The first converter uses capacitor C1 to double the +5V input to +10V on C3 at the V+ output. The second converter uses capacitor C2 to invert +10V to -10V on C4 at the V- output.

A small amount of power may be drawn from the +10V (V+) and -10V (V-) outputs to power external circuitry (see the *Typical Operating Characteristics* section), except on the MAX225 and MAX245–MAX247, where these pins are not available. V+ and V- are not regulated, so the output voltage drops with increasing load current. Do not load V+ and V- to a point that violates the minimum ±5V EIA/TIA-232E driver output voltage when sourcing current from V+ and V- to external circuitry.

When using the shutdown feature in the MAX222, MAX225, MAX230, MAX235, MAX236, MAX240, MAX241, and MAX245–MAX249, avoid using V+ and V- to power external circuitry. When these parts are shut down, V- falls to 0V, and V+ falls to +5V. For applications where a +10V external supply is applied to the V+ pin (instead of using the internal charge pump to generate +10V), the C1 capacitor must not be installed and the $\overline{SHDN}$ pin must be tied to V_{CC}. This is because V+ is internally connected to V_{CC} in shutdown mode.

RS-232 Drivers

The typical driver output voltage swing is ±8V when loaded with a nominal 5kΩ RS-232 receiver and V_{CC} = +5V. Output swing is guaranteed to meet the EIA/TIA-232E and V.28 specification, which calls for ±5V minimum driver output levels under worst-case conditions. These include a minimum 3kΩ load, V_{CC} = +4.5V, and maximum operating temperature. Unloaded driver output voltage ranges from (V+ -1.3V) to (V- +0.5V).

Input thresholds are both TTL and CMOS compatible. The inputs of unused drivers can be left unconnected since 400kΩ input pull-up resistors to V_{CC} are built in. The pull-up resistors force the outputs of unused drivers low because all drivers invert. The internal input pull-up resistors typically source 12µA, except in shutdown mode where the pull-ups are disabled. Driver outputs turn off and enter a high-impedance state—where leakage current is typically microamperes (maximum 25µA)—when in shutdown mode, in three-state mode, or when device power is removed. Outputs can be driven to ±15V. The power-supply current typically drops to 8µA in shutdown mode.

The MAX239 has a receiver three-state control line, and the MAX223, MAX225, MAX235, MAX236, MAX240, and MAX241 have both a receiver three-state control line and a low-power shutdown control. Table 2 shows the effects of the shutdown control and receiver three-state control on the receiver outputs.

The receiver TTL/CMOS outputs are in a high-impedance, three-state mode whenever the three-state enable line is high (for the MAX225/MAX235/MAX236/MAX239–MAX241), and are also high-impedance whenever the shutdown control line is high.

When in low-power shutdown mode, the driver outputs are turned off and their leakage current is less than 1µA with the driver output pulled to ground. The driver output leakage remains less than 1µA, even if the transmitter output is backdriven between 0V and (V_{CC} + 6V). Below -0.5V, the transmitter is diode clamped to ground with 1kΩ series impedance. The transmitter is also zener clamped to approximately V_{CC} + 6V, with a series impedance of 1kΩ.

The driver output slew rate is limited to less than 30V/µs as required by the EIA/TIA-232E and V.28 specifications. Typical slew rates are 24V/µs unloaded and 10V/µs loaded with 3Ω and 2500pF.

RS-232 Receivers

EIA/TIA-232E and V.28 specifications define a voltage level greater than 3V as a logic 0, so all receivers invert. Input thresholds are set at 0.8V and 2.4V, so receivers respond to TTL level inputs as well as EIA/TIA-232E and V.28 levels.

The receiver inputs withstand an input overvoltage up to ±25V and provide input terminating resistors with nominal 5kΩ values. The receivers implement Type 1 interpretation of the fault conditions of V.28 and EIA/TIA-232E.

Table 2. Three-State Control of Receivers

PART	SHDN	$\overline{SHDN}$	EN	$\overline{EN(R)}$	RECEIVERS
MAX223	—	Low High High	X Low High	—	High Impedance Active High Impedance
MAX225	—	—	—	Low High	High Impedance Active
MAX235 MAX236 MAX240	Low Low High		—	Low High X	High Impedance Active High Impedance

The receiver input hysteresis is typically 0.5V with a guaranteed minimum of 0.2V. This produces clear output transitions with slow-moving input signals, even with moderate amounts of noise and ringing. The receiver propagation delay is typically 600ns and is independent of input swing direction.

Low-Power Receive Mode

The low-power receive-mode feature of the MAX223, MAX242, and MAX245–MAX249 puts the IC into shutdown mode but still allows it to receive information. This is important for applications where systems are periodically awakened to look for activity. Using low-power receive mode, the system can still receive a signal that will activate it on command and prepare it for communication at faster data rates. This operation conserves system power.

Negative Threshold—MAX243

The MAX243 is pin compatible with the MAX232A, differing only in that RS-232 cable fault protection is removed on one of the two receiver inputs. This means that control lines such as CTS and RTS can either be driven or left floating without interrupting communication. Different cables are not needed to interface with different pieces of equipment.

The input threshold of the receiver without cable fault protection is -0.8V rather than +1.4V. Its output goes positive only if the input is connected to a control line that is actively driven negative. If not driven, it defaults to the 0 or "OK to send" state. Normally, the MAX243's other receiver (+1.4V threshold) is used for the data line (TD or RD), while the negative threshold receiver is connected to the control line (DTR, DTS, CTS, RTS, etc.).

Other members of the RS-232 family implement the optional cable fault protection as specified by EIA/TIA-232E specifications. This means a receiver output goes high whenever its input is driven negative, left floating, or shorted to ground. The high output tells the serial communications IC to stop sending data. To avoid this, the control lines must either be driven or connected with jumpers to an appropriate positive voltage level.

Shutdown—MAX222–MAX242

On the MAX222, MAX235, MAX236, MAX240, and MAX241, all receivers are disabled during shutdown. On the MAX223 and MAX242, two receivers continue to operate in a reduced power mode when the chip is in shutdown. Under these conditions, the propagation delay increases to about 2.5µs for a high-to-low input transition. When in shutdown, the receiver acts as a CMOS inverter with no hysteresis. The MAX223 and MAX242 also have a receiver output enable input ($\overline{EN}$ for the MAX242 and EN for the MAX223) that allows receiver output control independent of $\overline{SHDN}$ (SHDN for MAX241). With all other devices, $\overline{SHDN}$ (SHDN for MAX241) also disables the receiver outputs.

The MAX225 provides five transmitters and five receivers, while the MAX245 provides ten receivers and eight transmitters. Both devices have separate receiver and transmitter-enable controls. The charge pumps turn off and the devices shut down when a logic high is applied to the ENT input. In this state, the supply current drops to less than 25µA and the receivers continue to operate in a low-power receive mode. Driver outputs enter a high-impedance state (three-state mode). On the MAX225, all five receivers are controlled by the $\overline{ENR}$ input. On the MAX245, eight of the receiver outputs are controlled by the $\overline{ENR}$ input, while the remaining two receivers (RA5 and RB5) are always active. RA1–RA4 and RB1–RB4 are put in a three-state mode when $\overline{ENR}$ is a logic high.

Receiver and Transmitter Enable Control Inputs

The MAX225 and MAX245–MAX249 feature transmitter and receiver enable controls.

The receivers have three modes of operation: full-speed receive (normal active), three-state (disabled), and low-power receive (enabled receivers continue to function at lower data rates). The receiver enable inputs control the full-speed receive and three-state modes. The transmitters have two modes of operation: full-speed transmit (normal active) and three-state (disabled). The transmitter enable inputs also control the shutdown mode. The device enters shutdown mode when all transmitters are disabled. Enabled receivers function in the low-power receive mode when in shutdown.

+5V-Powered, Multichannel RS-232 Drivers/Receivers

Tables 1a–1d define the control states. The MAX244 has no control pins and is not included in these tables.

The MAX246 has ten receivers and eight drivers with two control pins, each controlling one side of the device. A logic high at the A-side control input ($\overline{ENA}$) causes the four A-side receivers and drivers to go into a three-state mode. Similarly, the B-side control input ($\overline{ENB}$) causes the four B-side drivers and receivers to go into a three-state mode. As in the MAX245, one A-side and one B-side receiver (RA5 and RB5) remain active at all times. The entire device is put into shutdown mode when both the A and B sides are disabled ($\overline{ENA}$ = $\overline{ENB}$ = +5V).

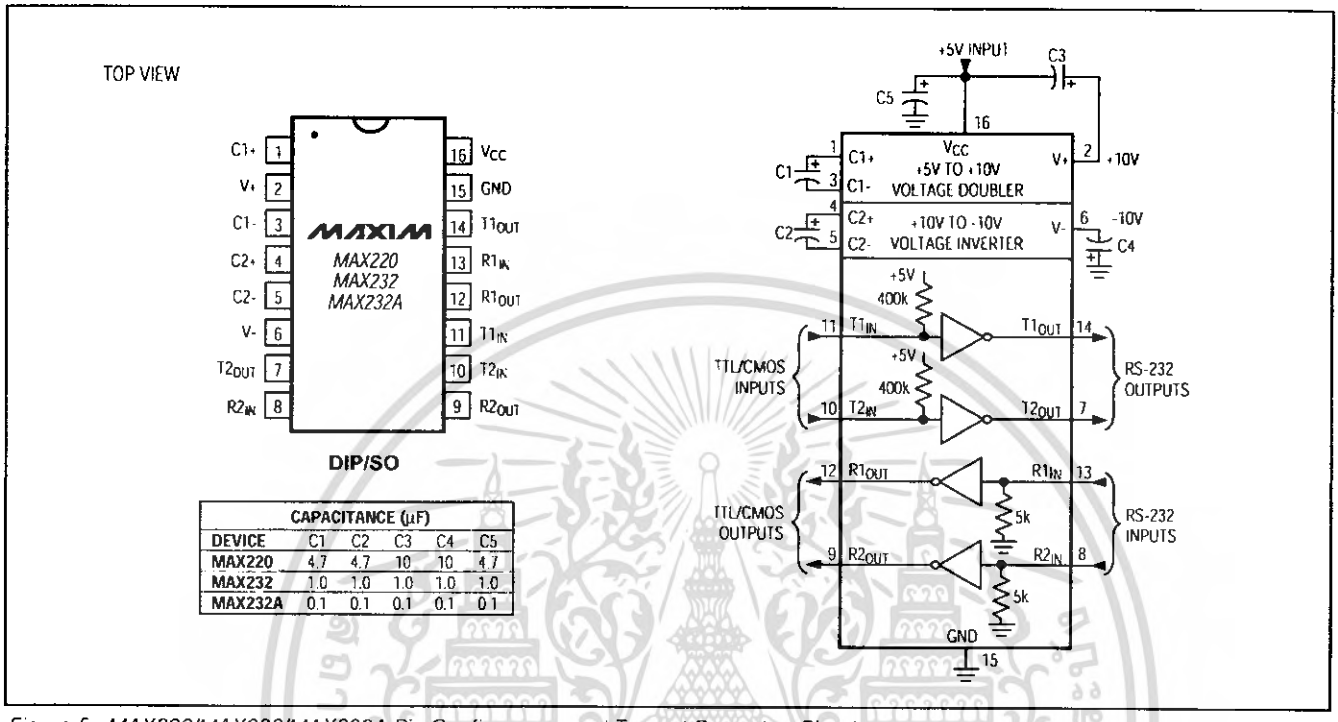
The MAX247 provides nine receivers and eight drivers with four control pins. The $\overline{ENRA}$ and $\overline{ENRB}$ receiver enable inputs each control four receiver outputs. The $\overline{ENTA}$ and $\overline{ENTB}$ transmitter enable inputs each control four drivers. The ninth receiver (RB5) is always active. The device enters shutdown mode with a logic high on both $\overline{ENTA}$ and $\overline{ENTB}$.

The MAX248 provides eight receivers and eight drivers with four control pins. The $\overline{ENRA}$ and $\overline{ENRB}$ receiver enable inputs each control four receiver outputs. The $\overline{ENTA}$ and $\overline{ENTB}$ transmitter enable inputs control four drivers each. This part does not have an always-active receiver. The device enters shutdown mode and transmitters go into a three-state mode with a logic high on both $\overline{ENTA}$ and $\overline{ENTB}$.

The MAX249 provides ten receivers and six drivers with four control pins. The $\overline{ENRA}$ and $\overline{ENRB}$ receiver enable inputs each control five receiver outputs. The $\overline{ENTA}$ and $\overline{ENTB}$ transmitter enable inputs control three drivers each. There is no always-active receiver. The device enters shutdown mode and transmitters go into a three-state mode with a logic high on both $\overline{ENTA}$ and $\overline{ENTB}$. In shutdown mode, active receivers operate in a low-power receive mode at data rates up to 20kbits/sec.

Applications Information

Figures 5 through 25 show pin configurations and typical operating circuits. In applications that are sensitive to power-supply noise, V_{CC} should be decoupled to ground with a capacitor of the same value as C1 and C2 connected as close as possible to the device.



CAPACITANCE (µF)					
DEVICE	C1	C2	C3	C4	C5
MAX220	4.7	4.7	10	10	4.7
MAX232	1.0	1.0	1.0	1.0	1.0
MAX232A	0.1	0.1	0.1	0.1	0.1

Figure 5. MAX220/MAX232/MAX232A Pin Configuration and Typical Operating Circuit

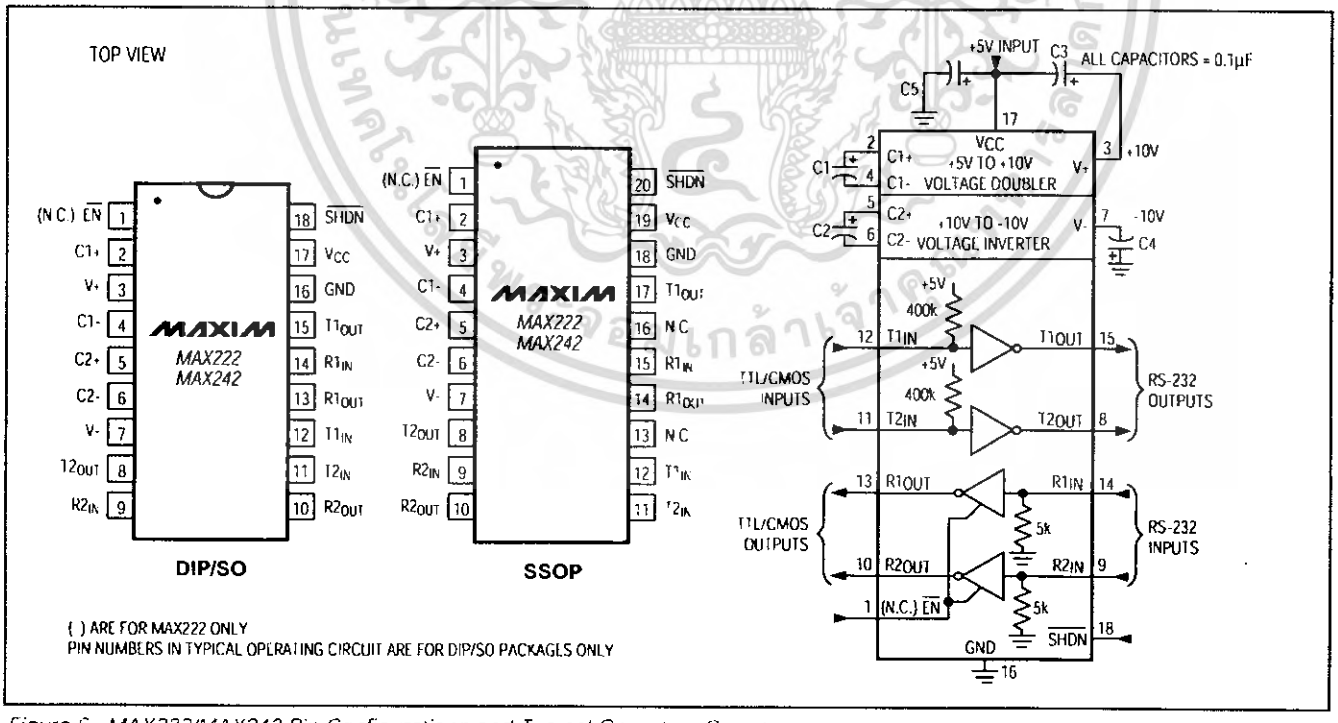


Figure 6. MAX222/MAX242 Pin Configurations and Typical Operating Circuit

+5V-Powered, Multichannel RS-232 Drivers/Receivers

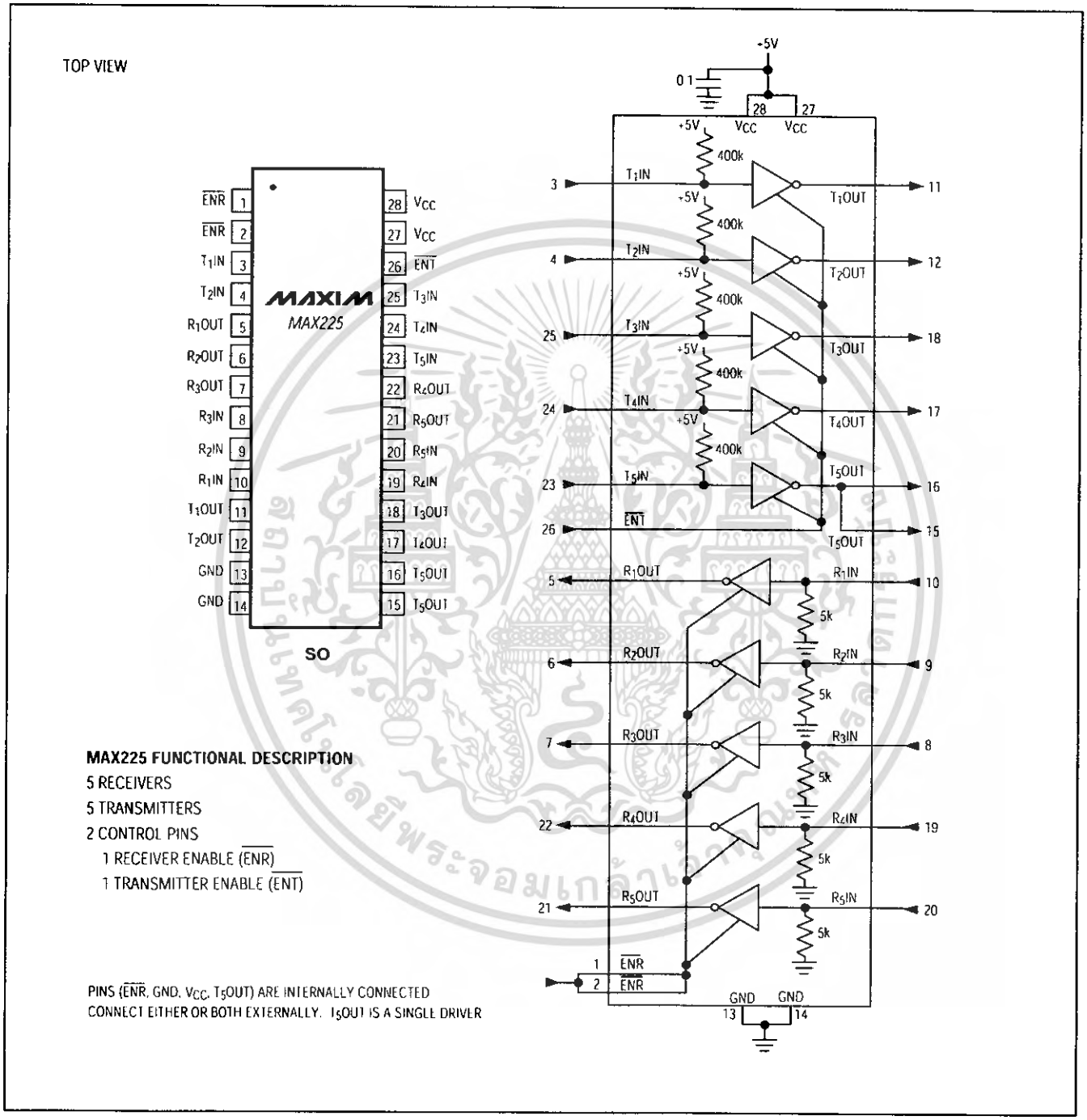


Figure 7. MAX225 Pin Configuration and Typical Operating Circuit

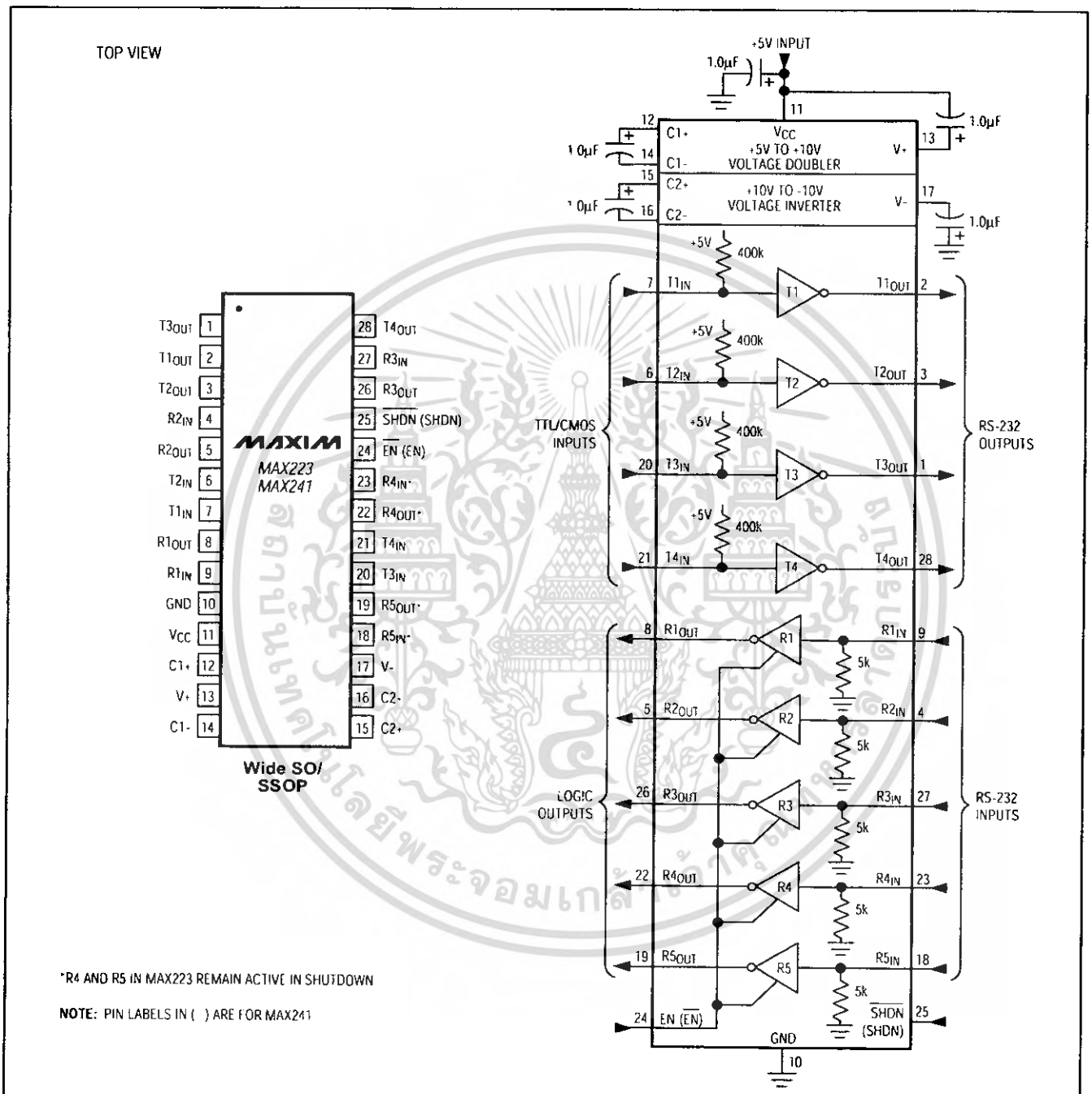


Figure 8. MAX223/MAX241 Pin Configuration and Typical Operating Circuit

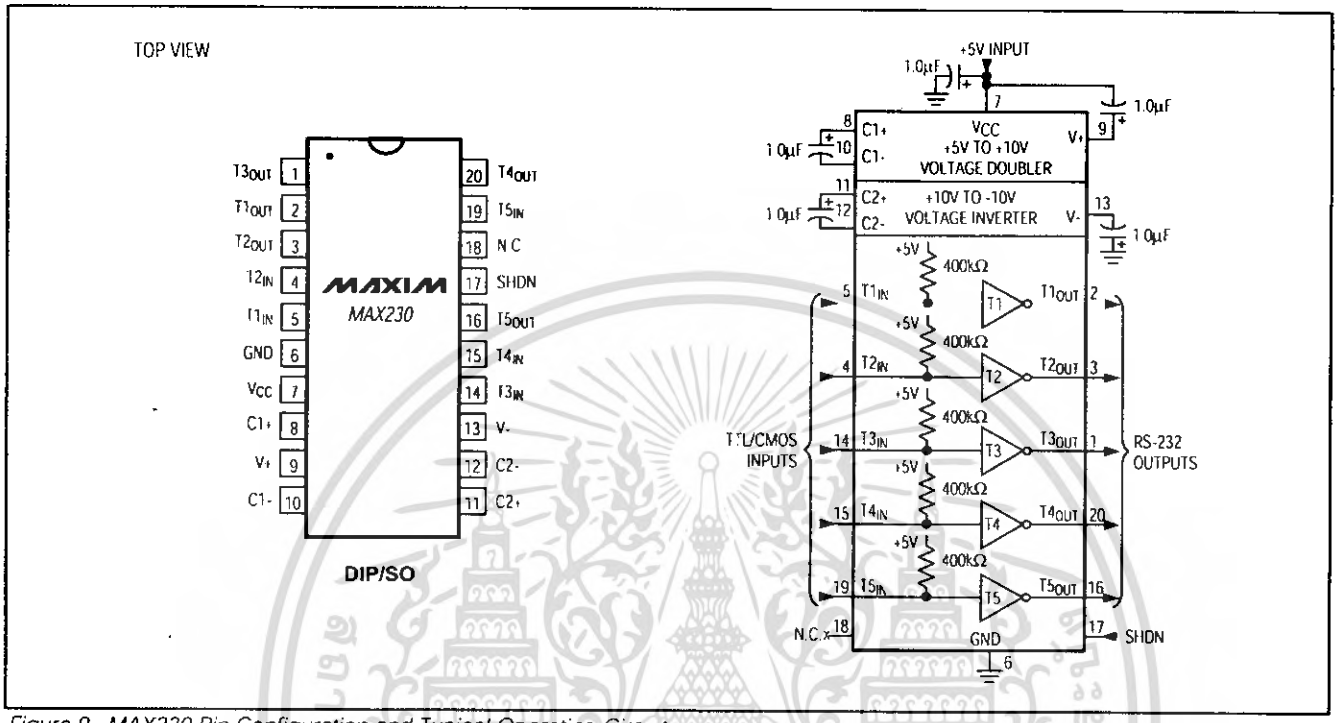


Figure 9. MAX230 Pin Configuration and Typical Operating Circuit

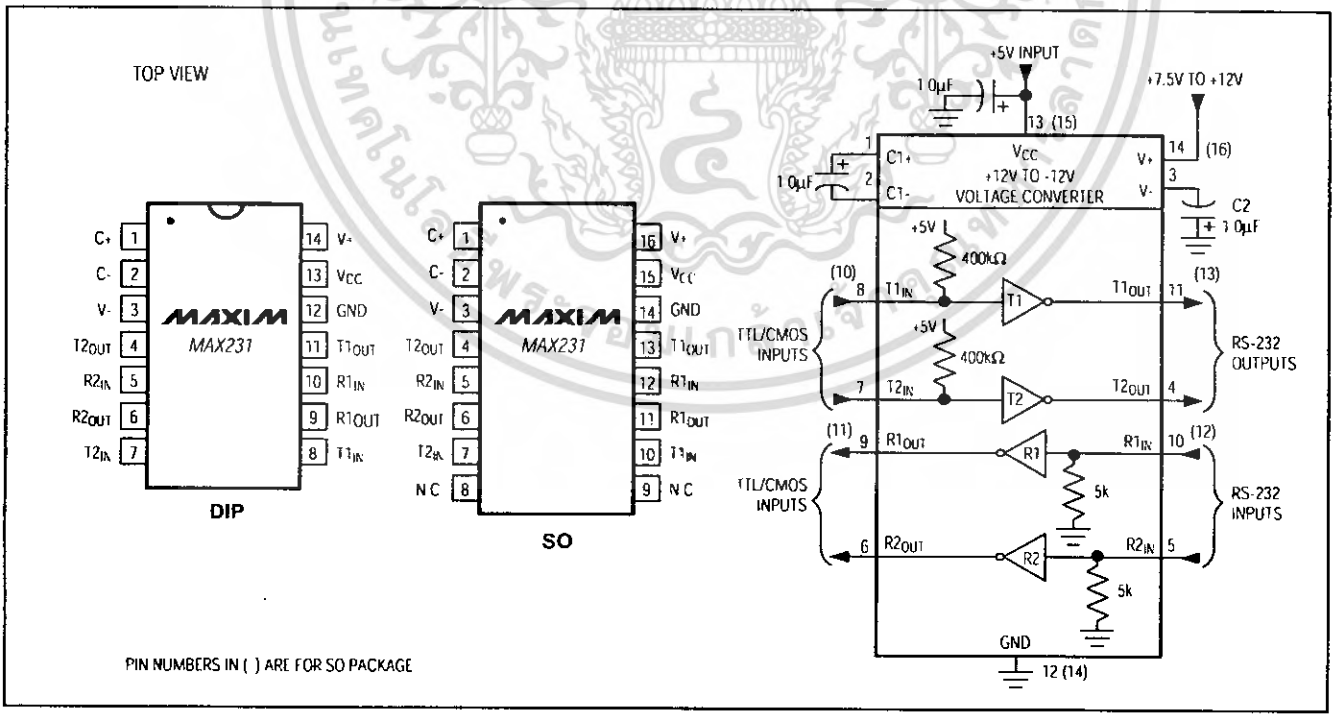


Figure 10. MAX231 Pin Configurations and Typical Operating Circuit

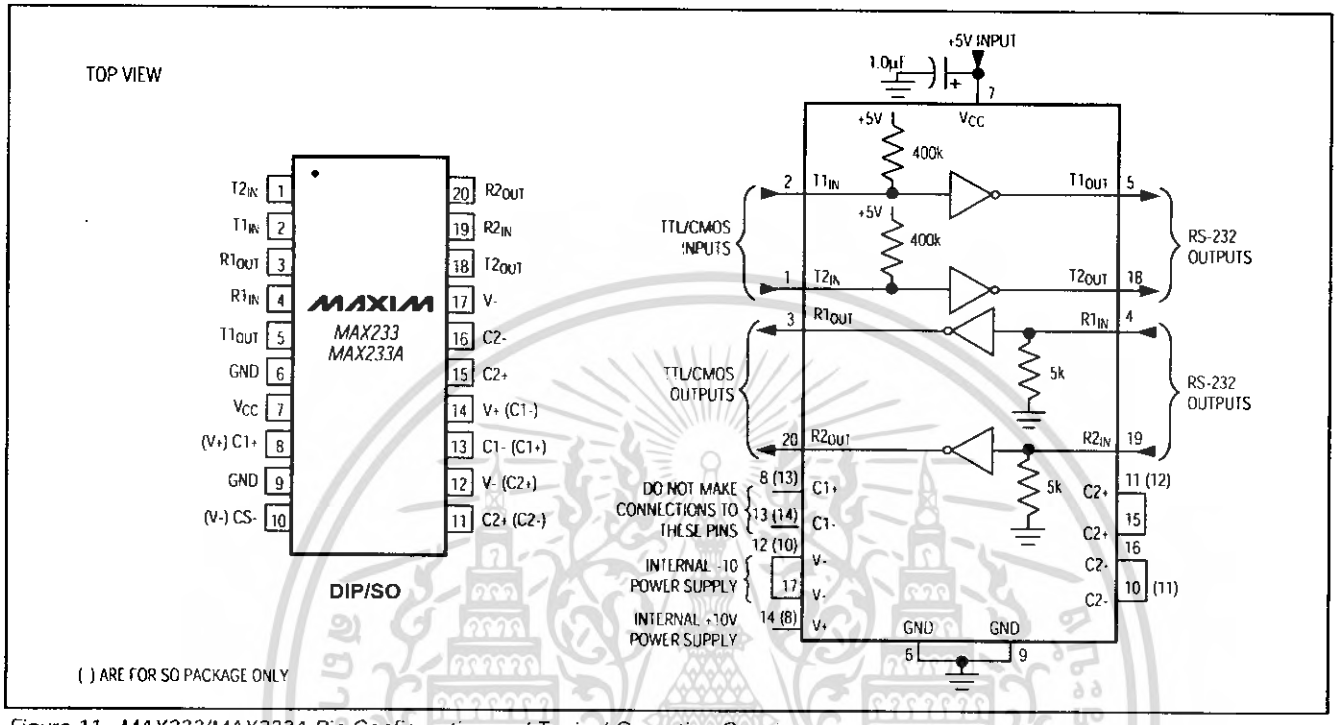


Figure 11. MAX233/MAX233A Pin Configuration and Typical Operating Circuit

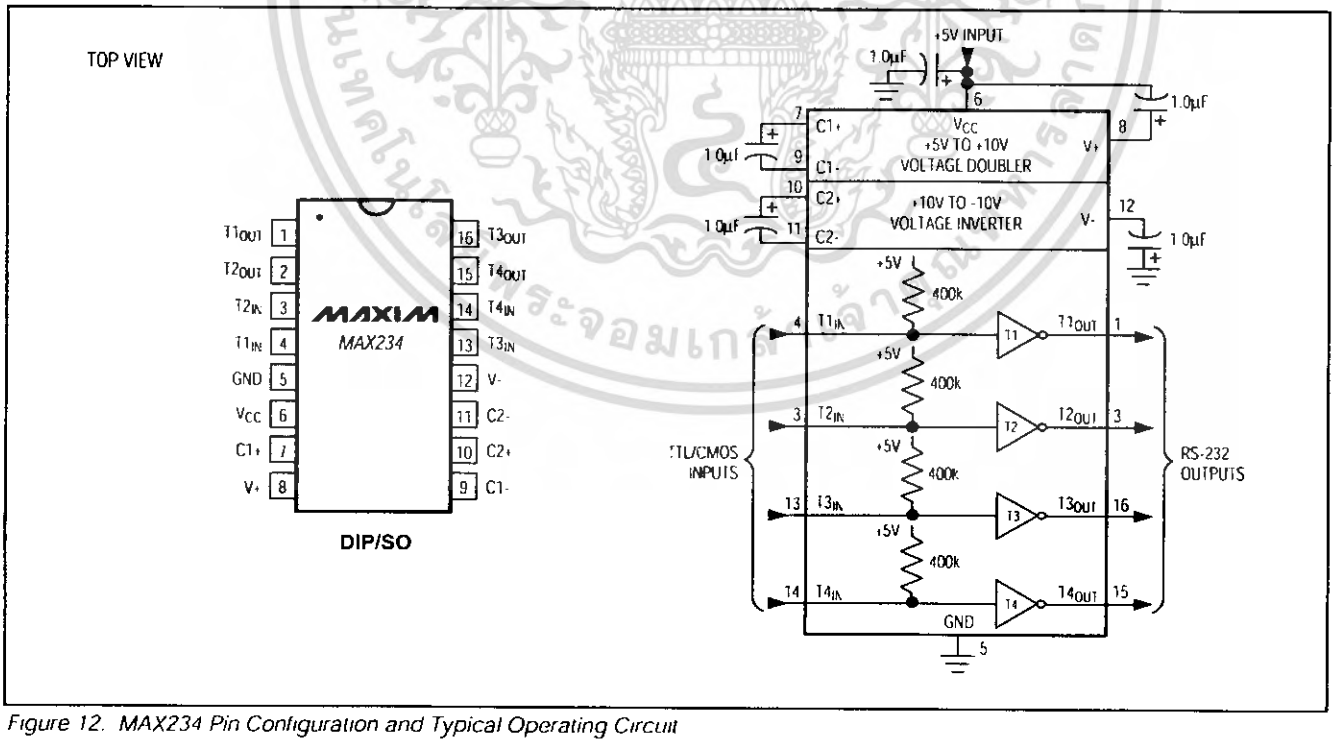


Figure 12. MAX234 Pin Configuration and Typical Operating Circuit

+5V-Powered, Multichannel RS-232 Drivers/Receivers

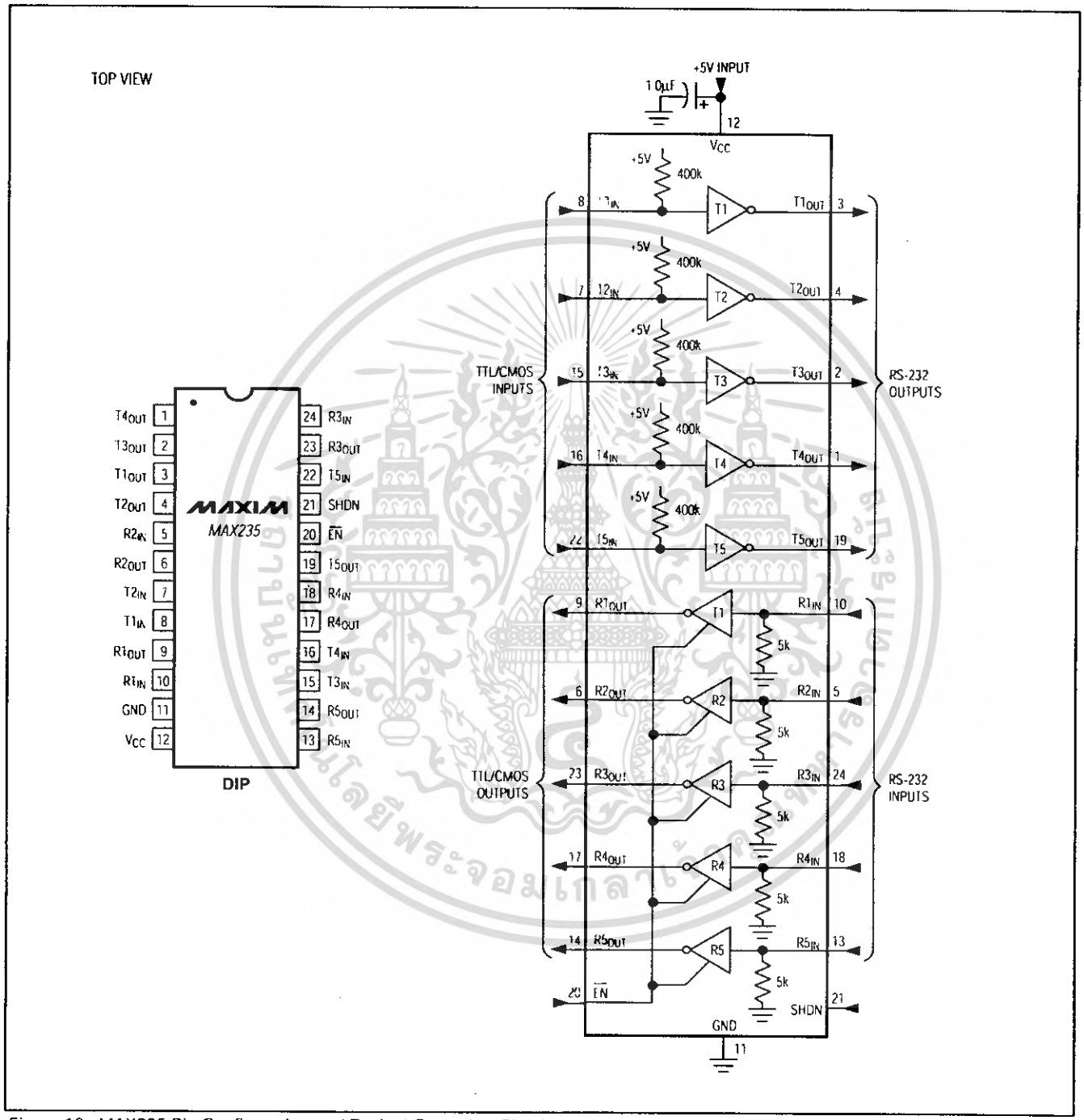


Figure 13. MAX235 Pin Configuration and Typical Operating Circuit

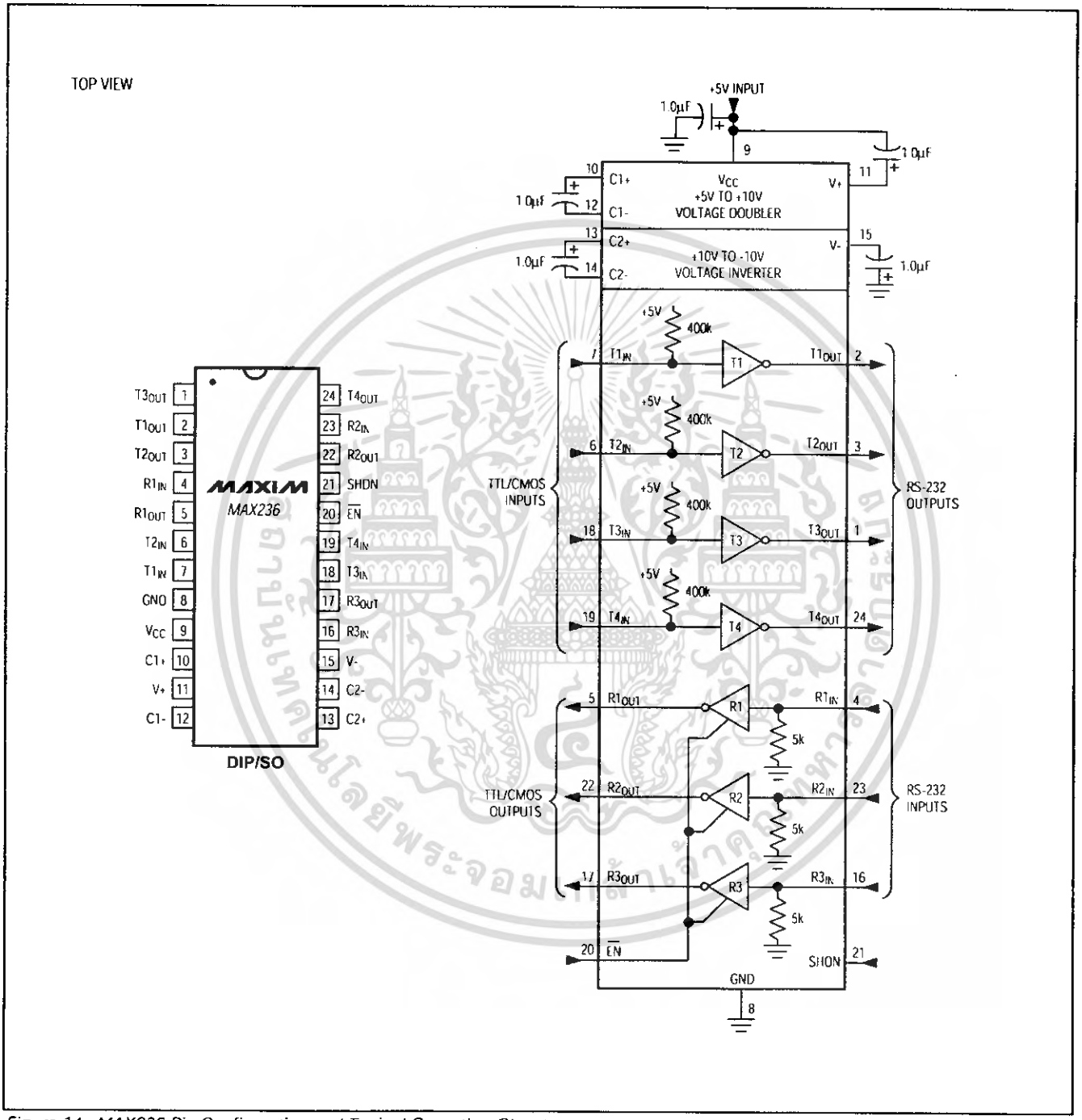


Figure 14. MAX236 Pin Configuration and Typical Operating Circuit

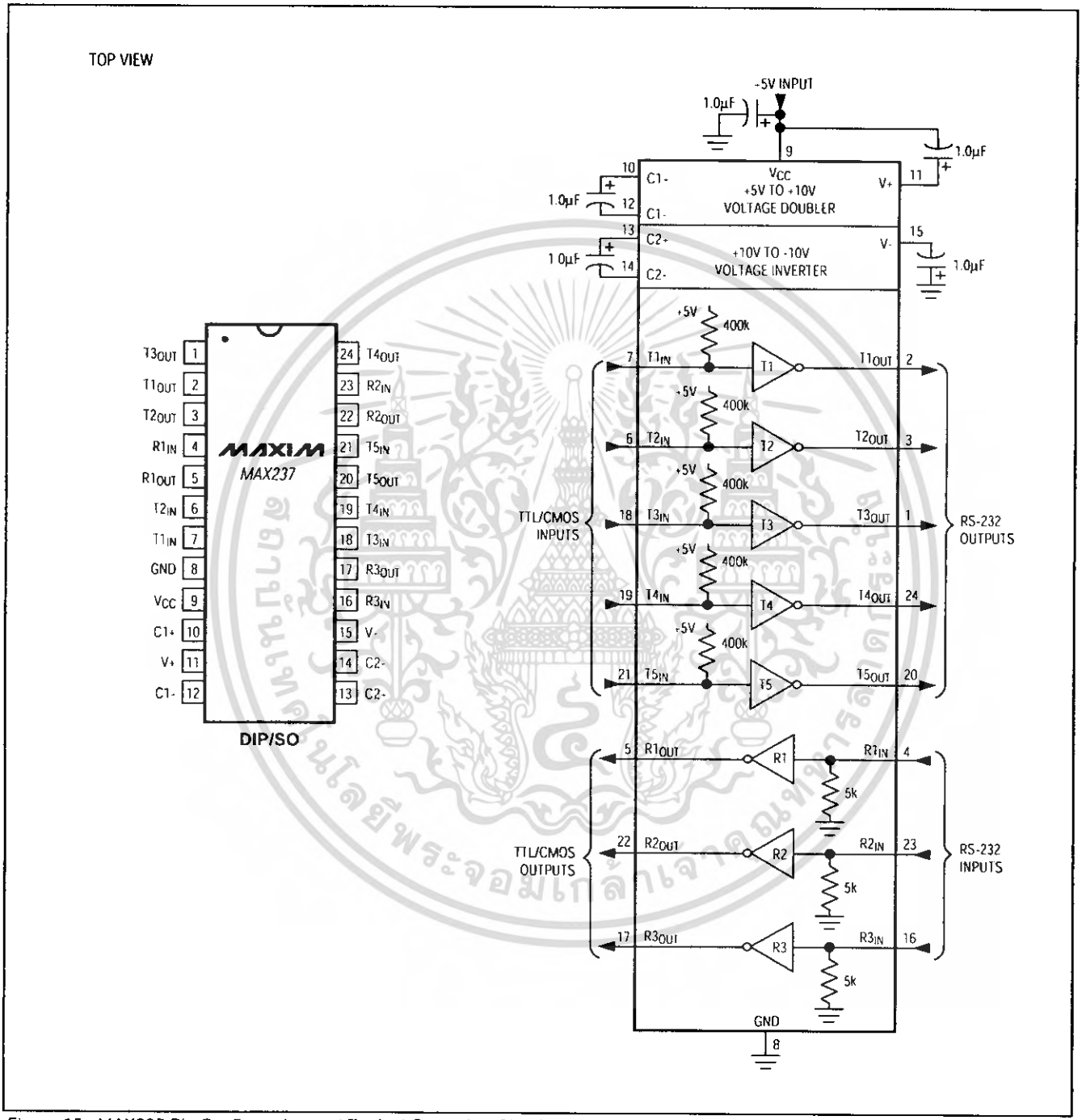


Figure 15. MAX237 Pin Configuration and Typical Operating Circuit

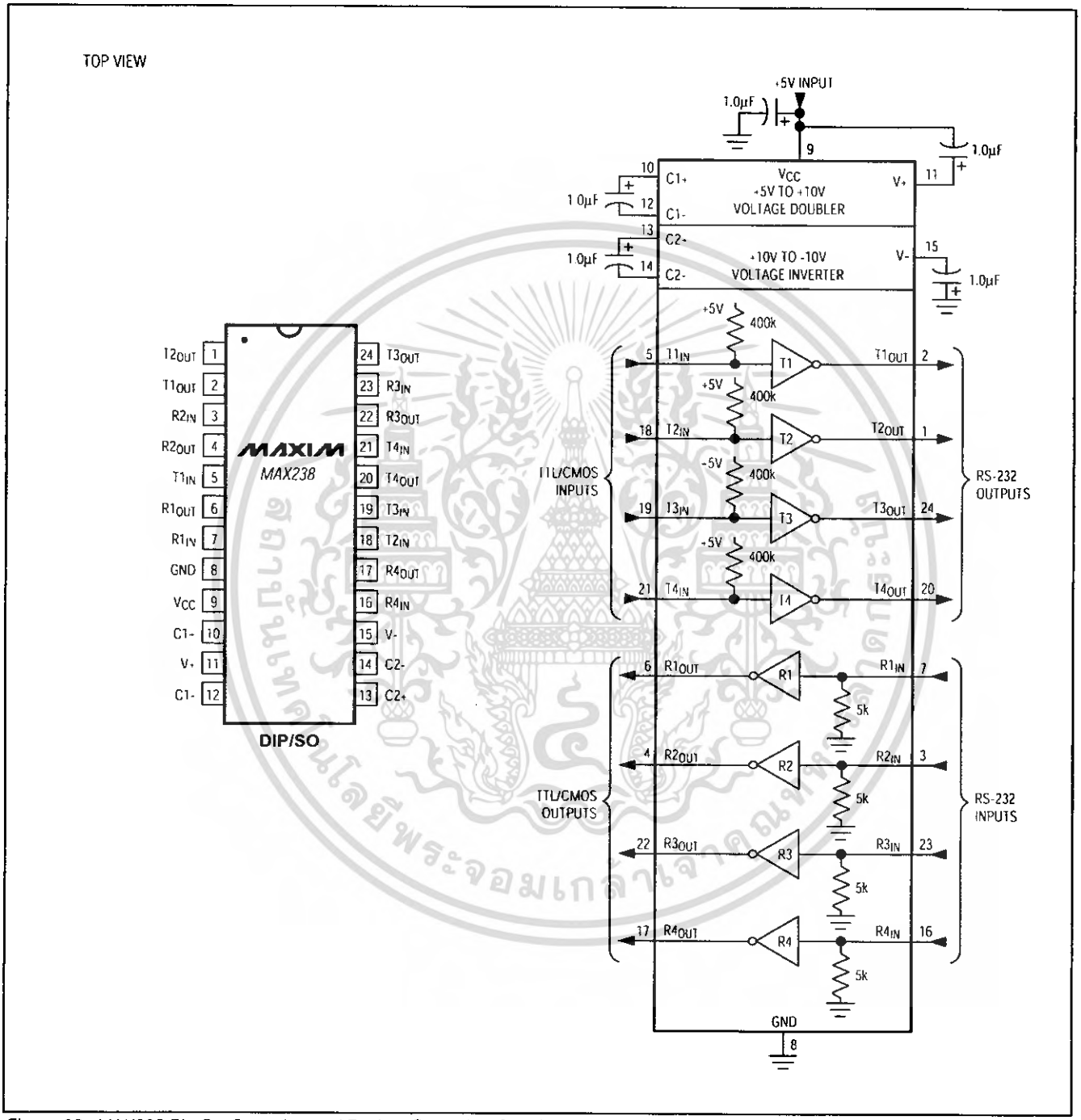


Figure 16. MAX238 Pin Configuration and Typical Operating Circuit

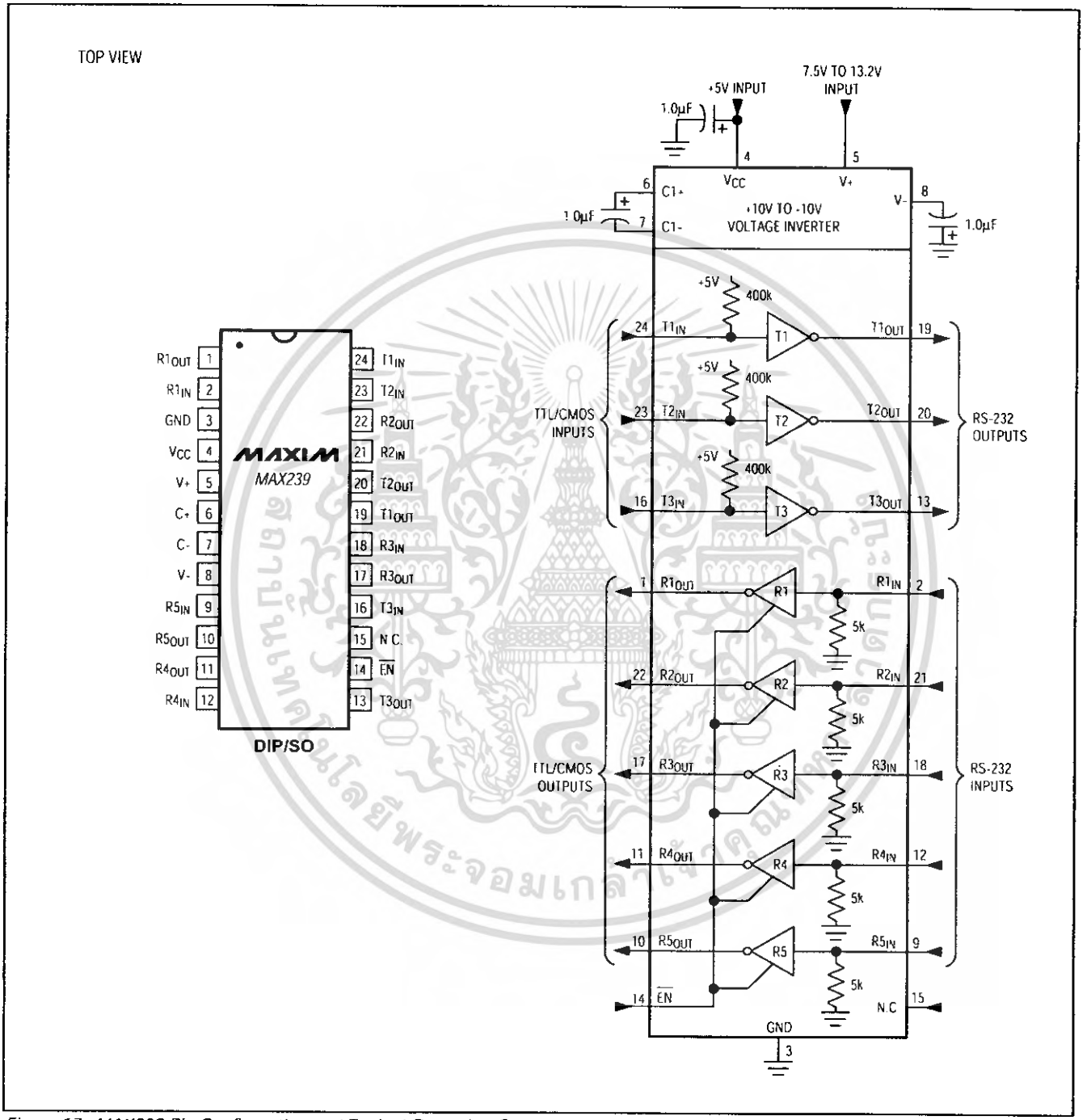


Figure 17. MAX239 Pin Configuration and Typical Operating Circuit

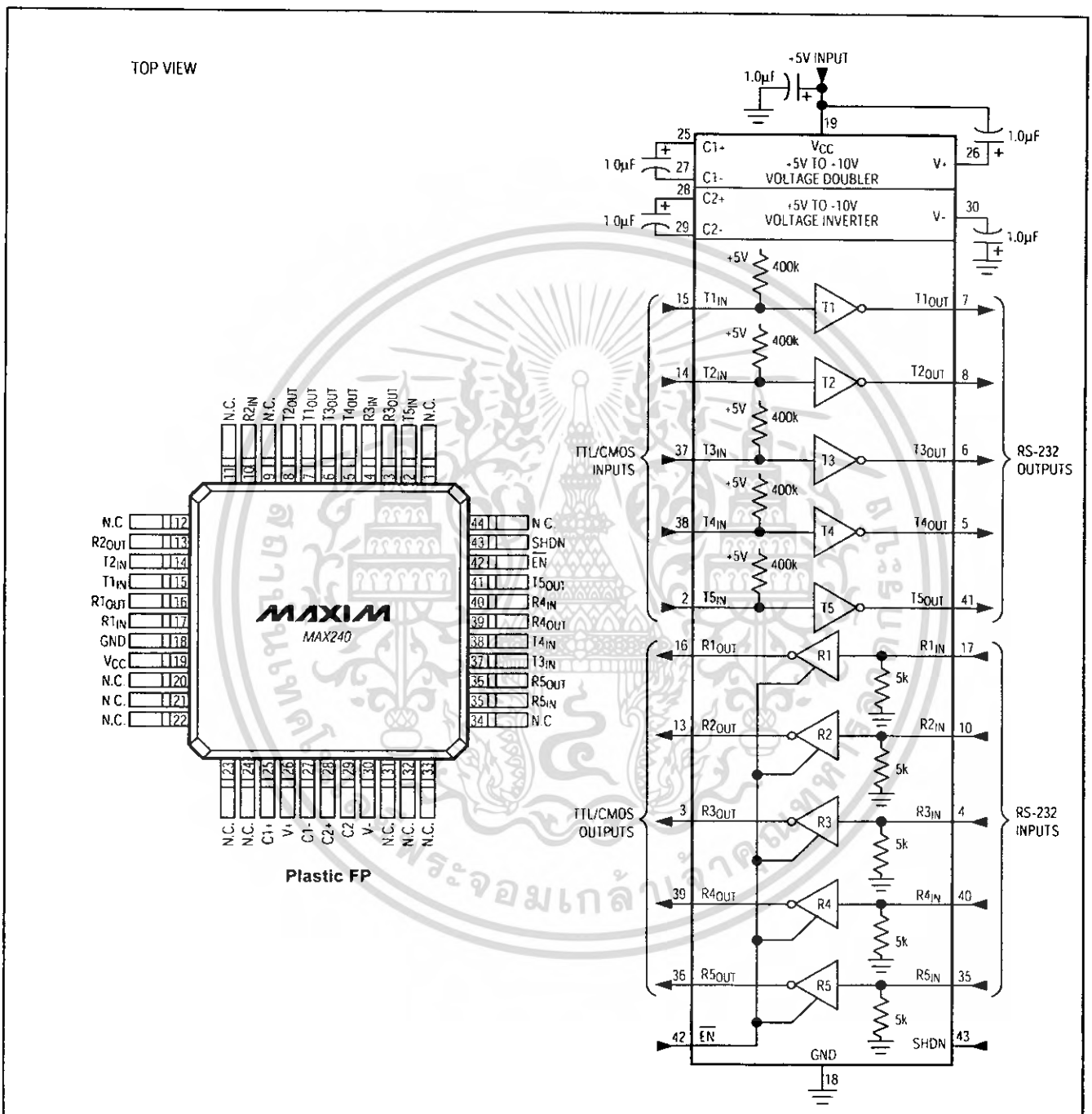


Figure 18. MAX240 Pin Configuration and Typical Operating Circuit

+5V-Powered, Multichannel RS-232 Drivers/Receivers

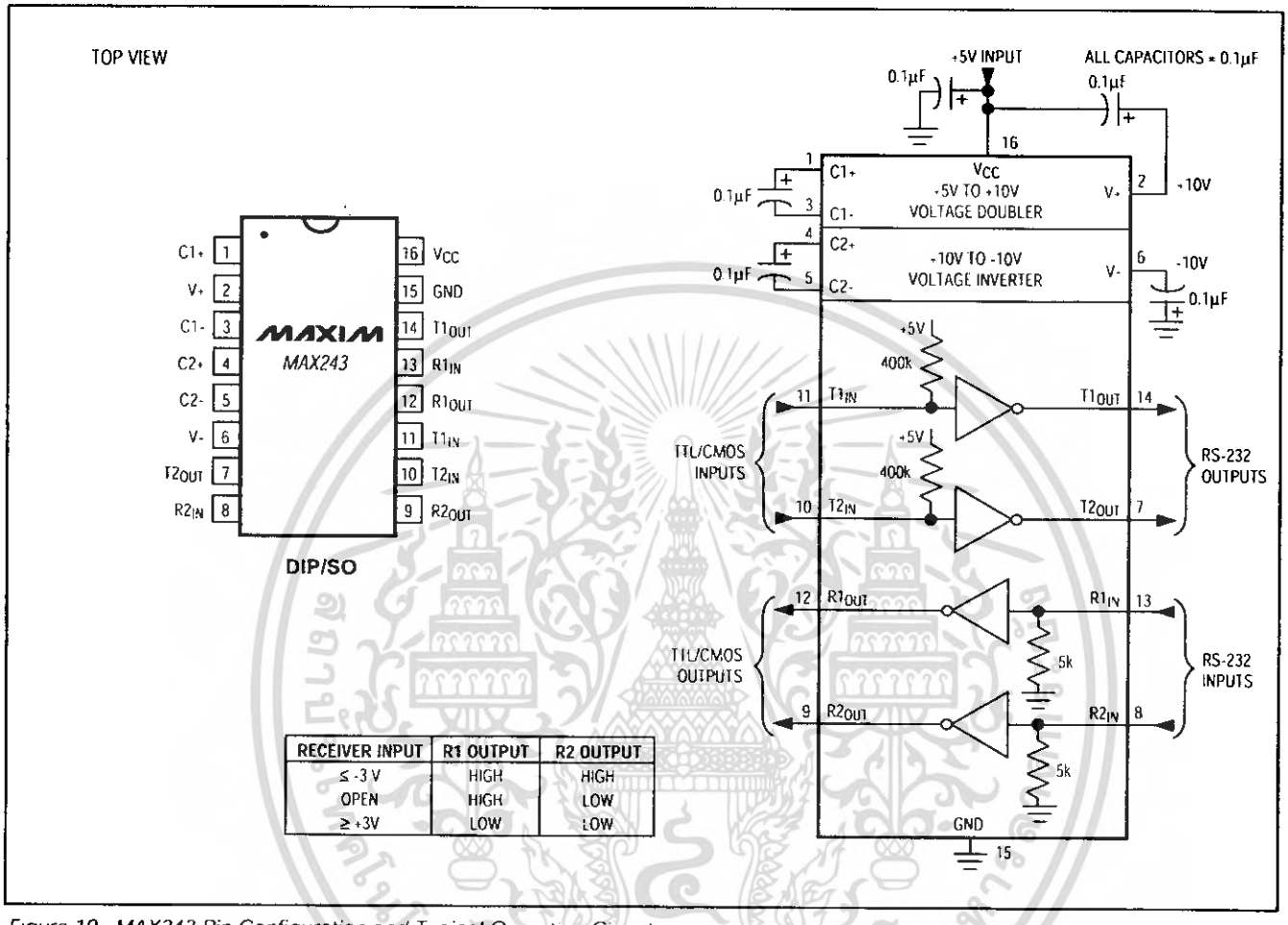


Figure 19. MAX243 Pin Configuration and Typical Operating Circuit

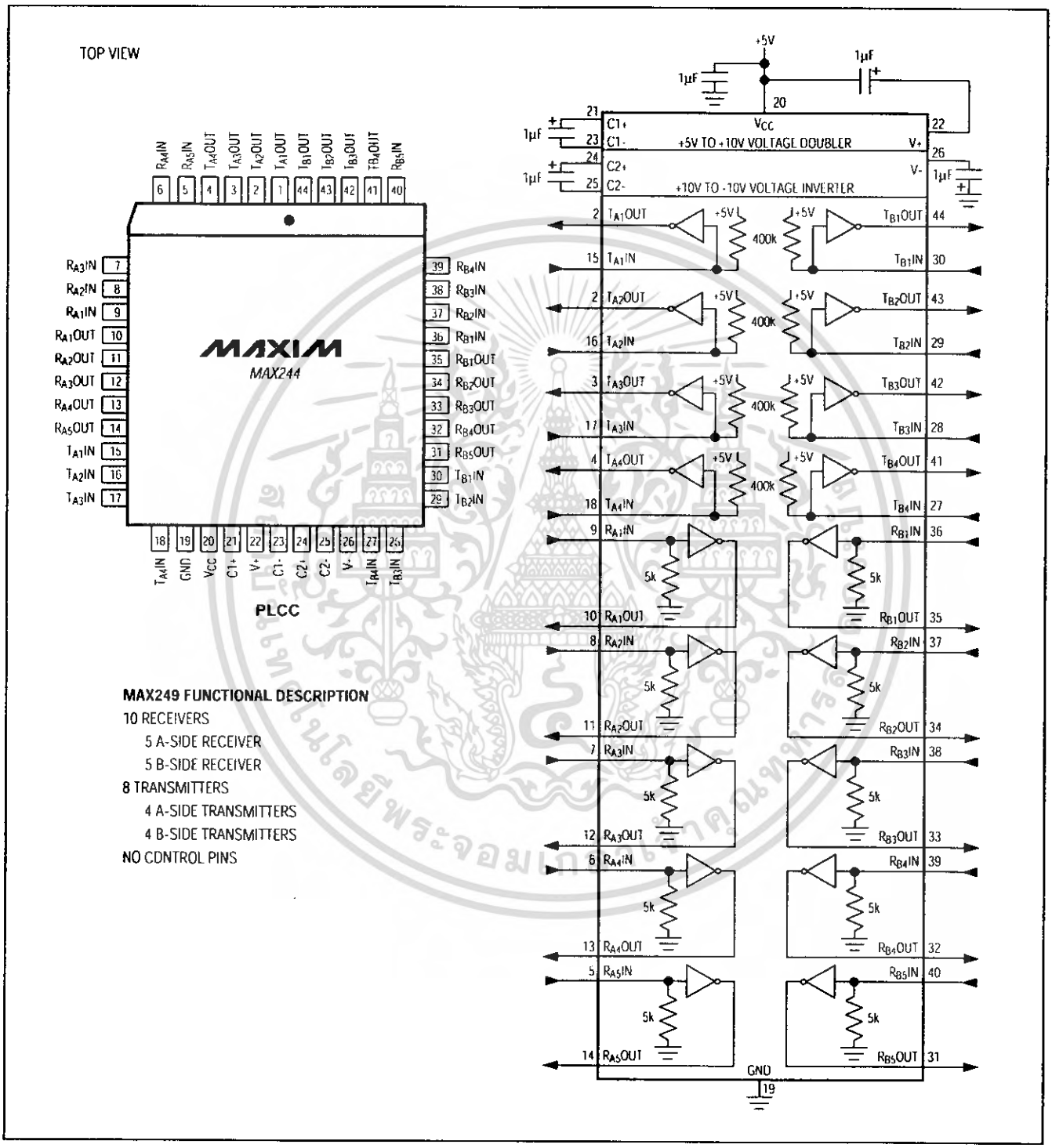
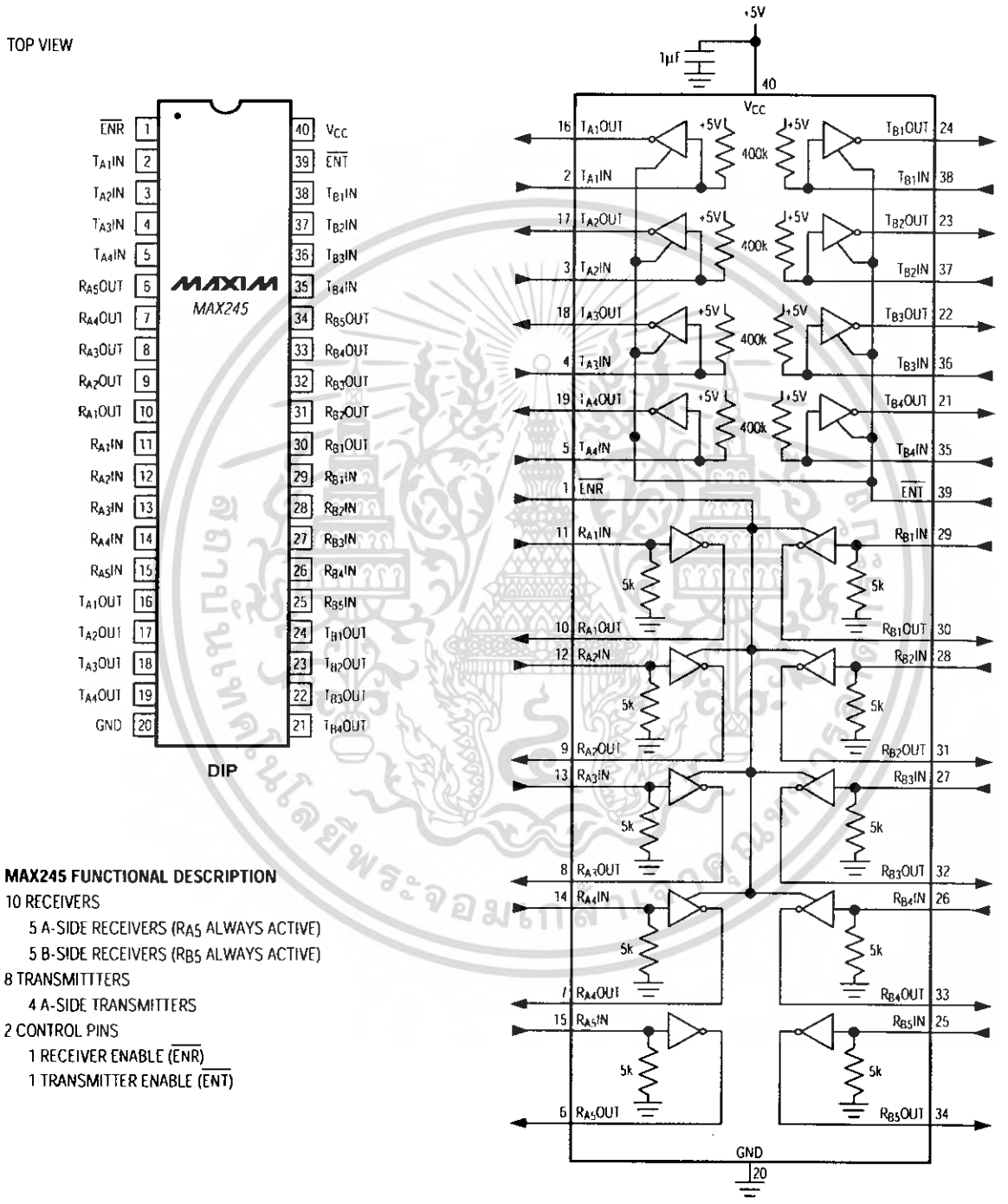


Figure 20. MAX244 Pin Configuration and Typical Operating Circuit

+5V-Powered, Multichannel RS-232 Drivers/Receivers



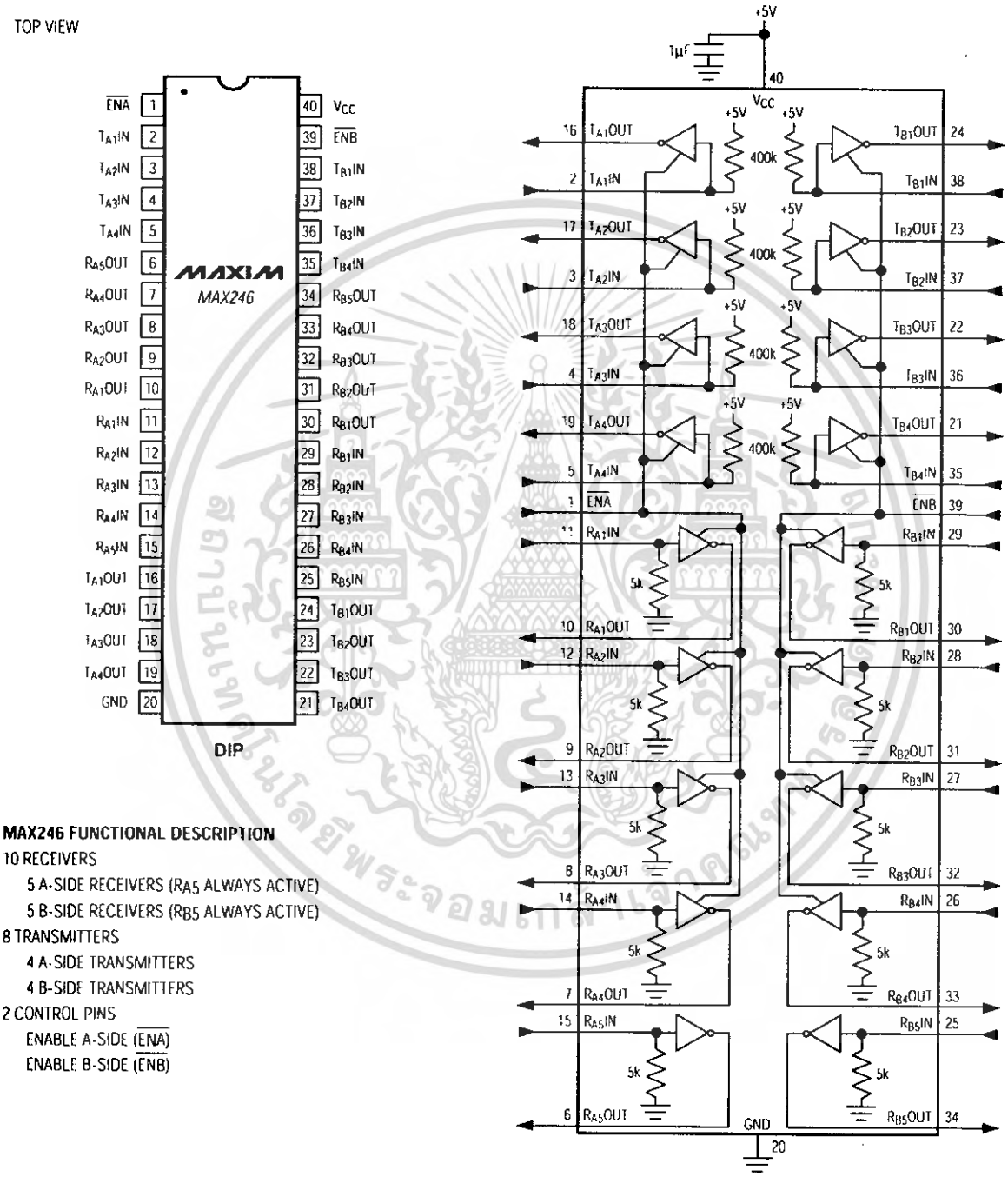
MAX245 FUNCTIONAL DESCRIPTION

10 RECEIVERS
- 5 A-SIDE RECEIVERS (R_{A5} ALWAYS ACTIVE)
- 5 B-SIDE RECEIVERS (R_{B5} ALWAYS ACTIVE)

8 TRANSMITTTERS
- 4 A-SIDE TRANSMITTERS

2 CONTROL PINS
- 1 RECEIVER ENABLE ($\overline{ENR}$)
- 1 TRANSMITTER ENABLE ($\overline{ENT}$)

Figure 21. MAX245 Pin Configuration and Typical Operating Circuit



MAX246 FUNCTIONAL DESCRIPTION

10 RECEIVERS
- 5 A-SIDE RECEIVERS (R_{A5} ALWAYS ACTIVE)
- 5 B-SIDE RECEIVERS (R_{B5} ALWAYS ACTIVE)

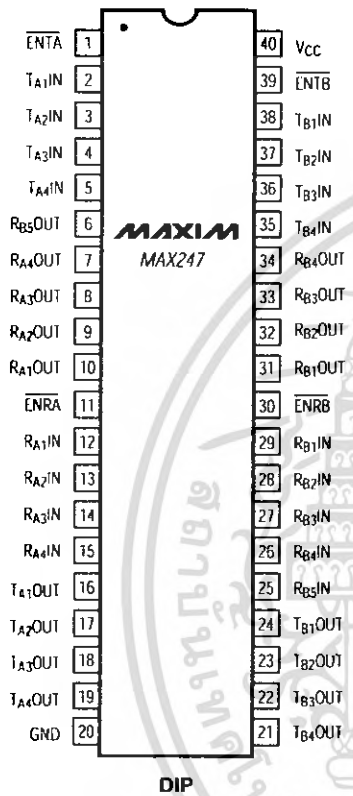
8 TRANSMITTERS
- 4 A-SIDE TRANSMITTERS
- 4 B-SIDE TRANSMITTERS

2 CONTROL PINS
- ENABLE A-SIDE ($\overline{ENA}$)
- ENABLE B-SIDE ($\overline{ENB}$)

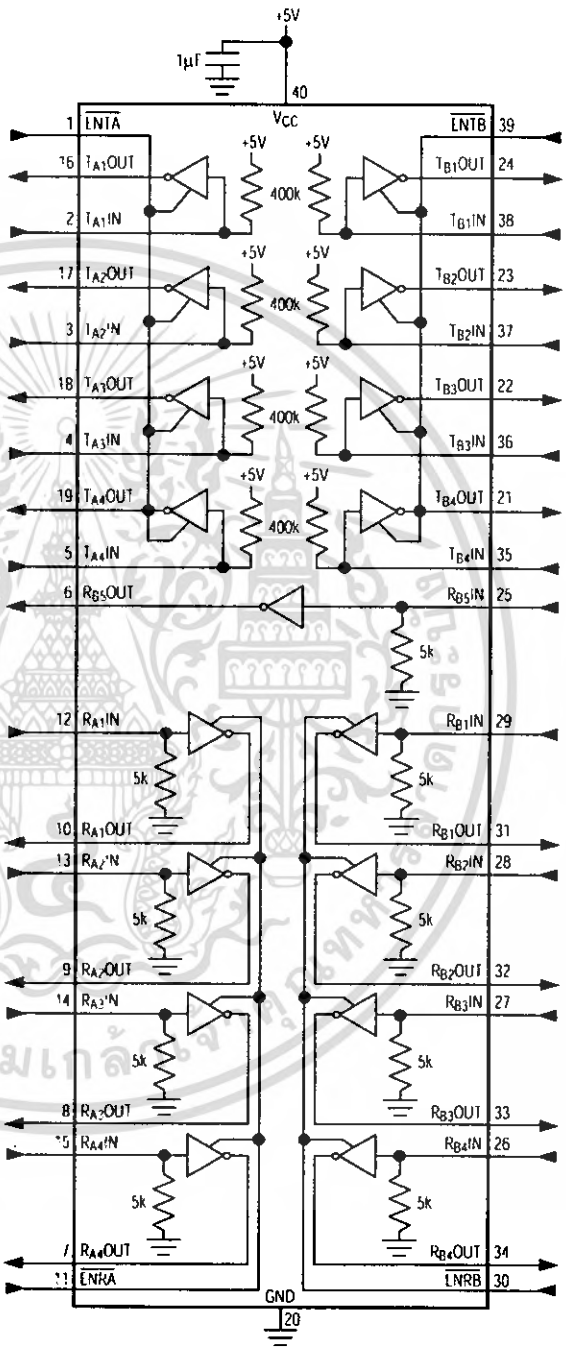
Figure 22. MAX246 Pin Configuration and Typical Operating Circuit

+5V-Powered, Multichannel RS-232 Drivers/Receivers

TOP VIEW



DIP



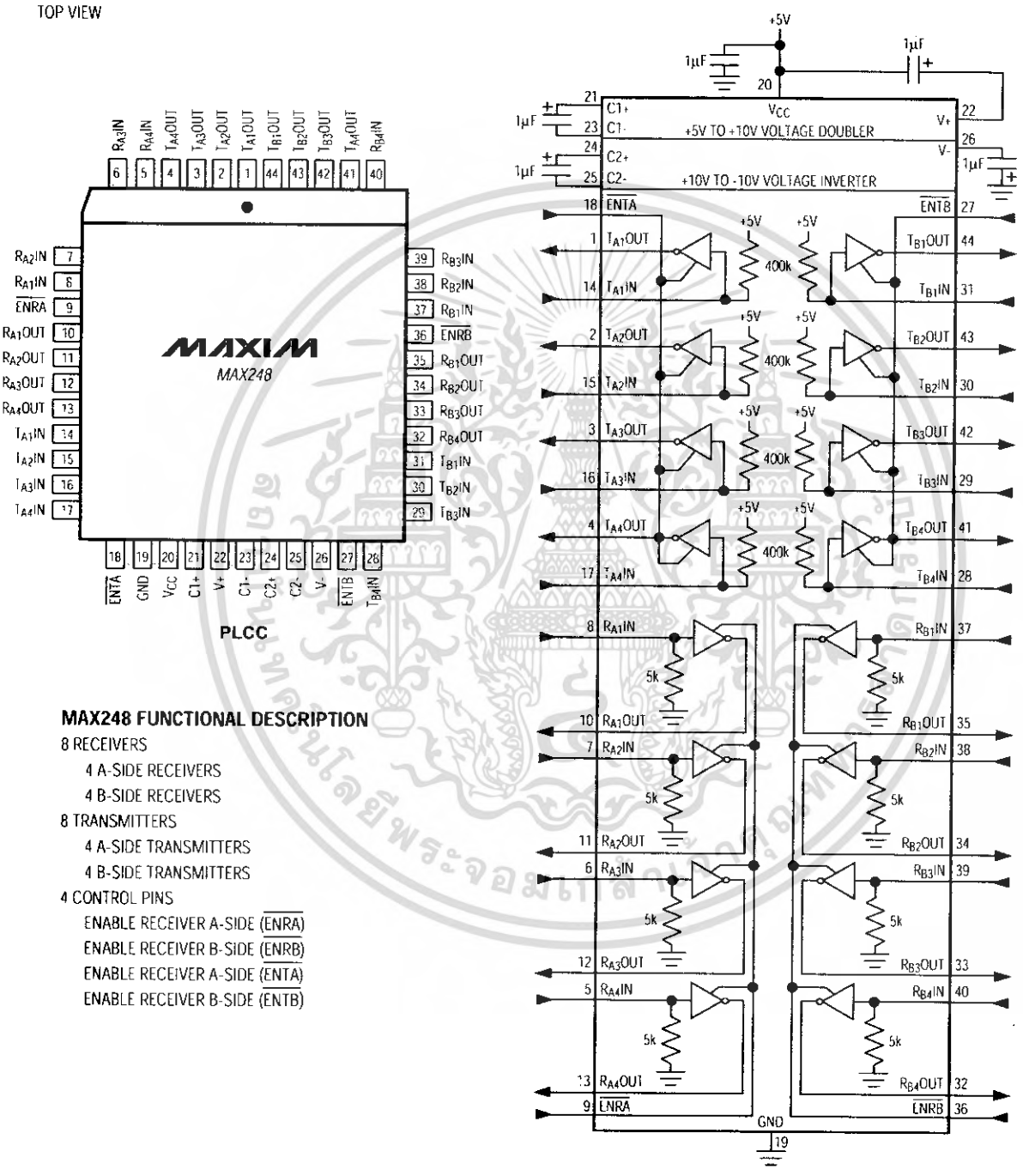
MAX247 FUNCTIONAL DESCRIPTION

9 RECEIVERS
- 4 A-SIDE RECEIVERS
- 5 B-SIDE RECEIVERS (R_{B5} ALWAYS ACTIVE)

8 TRANSMITTERS
- 4 A-SIDE TRANSMITTERS
- 4 B-SIDE TRANSMITTERS

4 CONTROL PINS
- ENABLE RECEIVER A-SIDE ($\overline{ENRA}$)
- ENABLE RECEIVER B-SIDE ($\overline{ENRB}$)
- ENABLE RECEIVER A-SIDE ($\overline{ENTA}$)
- ENABLE RECEIVERr B-SIDE ($\overline{ENTB}$)

Figure 23. MAX247 Pin Configuration and Typical Operating Circuit



MAX248 FUNCTIONAL DESCRIPTION

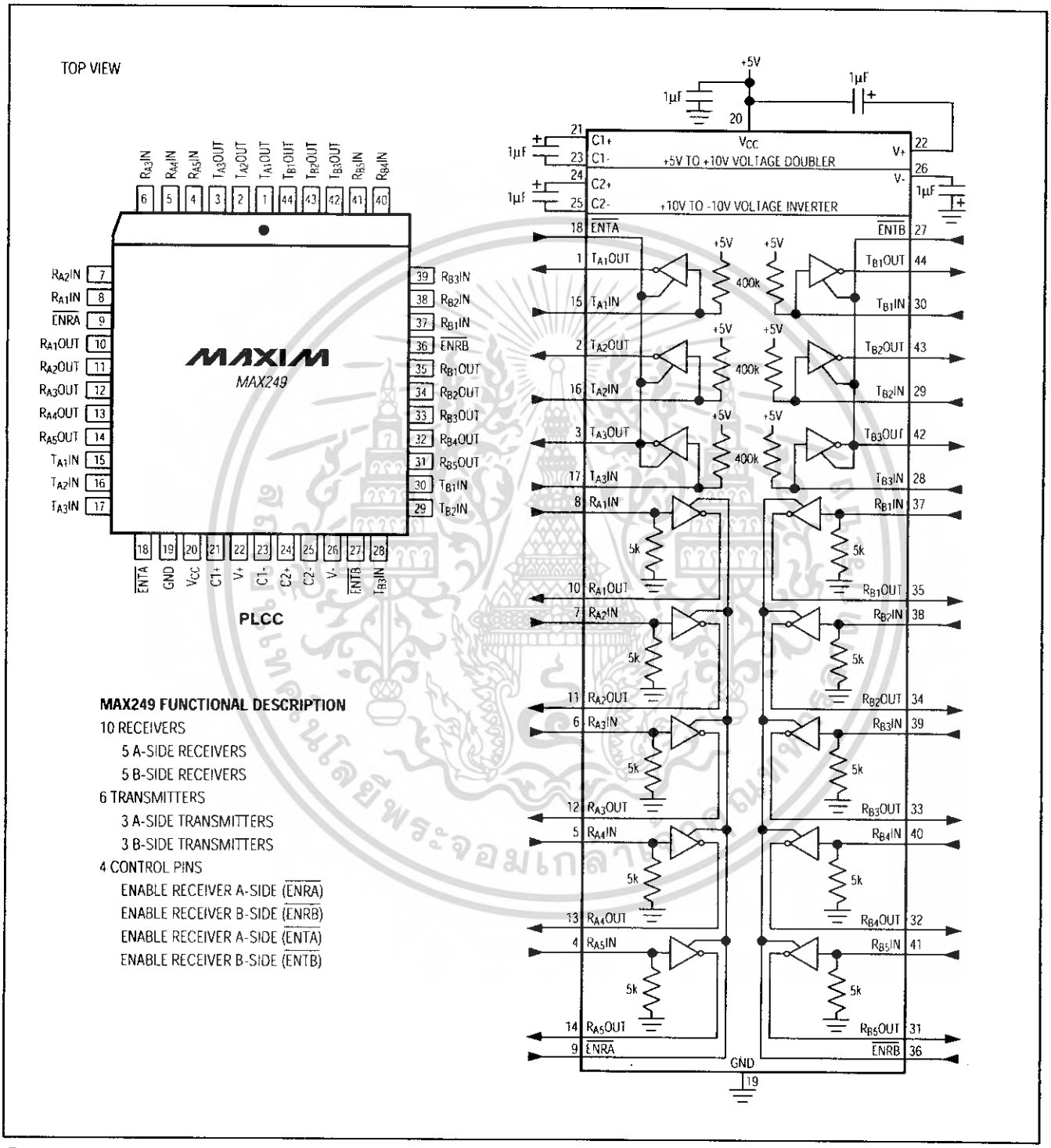
8 RECEIVERS
- 4 A-SIDE RECEIVERS
- 4 B-SIDE RECEIVERS

8 TRANSMITTERS
- 4 A-SIDE TRANSMITTERS
- 4 B-SIDE TRANSMITTERS

4 CONTROL PINS
- ENABLE RECEIVER A-SIDE ($\overline{ENRA}$)
- ENABLE RECEIVER B-SIDE ($\overline{ENRB}$)
- ENABLE RECEIVER A-SIDE ($\overline{ENTA}$)
- ENABLE RECEIVER B-SIDE ($\overline{ENTB}$)

Figure 24. MAX248 Pin Configuration and Typical Operating Circuit

+5V-Powered, Multichannel RS-232 Drivers/Receivers



MAX249 FUNCTIONAL DESCRIPTION

10 RECEIVERS
- 5 A-SIDE RECEIVERS
- 5 B-SIDE RECEIVERS

6 TRANSMITTERS
- 3 A-SIDE TRANSMITTERS
- 3 B-SIDE TRANSMITTERS

4 CONTROL PINS
- ENABLE RECEIVER A-SIDE ($\overline{ENRA}$)
- ENABLE RECEIVER B-SIDE ($\overline{ENRB}$)
- ENABLE RECEIVER A-SIDE ($\overline{ENTA}$)
- ENABLE RECEIVER B-SIDE ($\overline{ENTB}$)

Figure 25. MAX249 Pin Configuration and Typical Operating Circuit

Ordering Information (continued)

PART	TEMP. RANGE	PIN-PACKAGE
MAX222CPN	0°C to +70°C	18 Plastic DIP
MAX222CWN	0°C to +70°C	18 Wide SO
MAX222C/D	0°C to +70°C	Dice*
MAX222EPN	-40°C to +85°C	18 Plastic DIP
MAX222EWN	-40°C to +85°C	18 Wide SO
MAX222EJN	-40°C to +85°C	18 CERDIP
MAX222MJN	-55°C to +125°C	18 CERDIP
MAX223CAI	0°C to +70°C	28 SSOP
MAX223CWI	0°C to +70°C	28 Wide SO
MAX223C/D	0°C to +70°C	Dice*
MAX223EAI	-40°C to +85°C	28 SSOP
MAX223EWI	-40°C to +85°C	28 Wide SO
MAX225CWI	0°C to +70°C	28 Wide SO
MAX225EWI	-40°C to +85°C	28 Wide SO
MAX230CPP	0°C to +70°C	20 Plastic DIP
MAX230CWP	0°C to +70°C	20 Wide SO
MAX230C/D	0°C to +70°C	Dice*
MAX230EPP	-40°C to +85°C	20 Plastic DIP
MAX230EWP	-40°C to +85°C	20 Wide SO
MAX230EJP	-40°C to +85°C	20 CERDIP
MAX230MJP	-55°C to +125°C	20 CERDIP
MAX231CPD	0°C to +70°C	14 Plastic DIP
MAX231CWE	0°C to +70°C	16 Wide SO
MAX231CJD	0°C to +70°C	14 CERDIP
MAX231C/D	0°C to +70°C	Dice*
MAX231EPD	-40°C to +85°C	14 Plastic DIP
MAX231EWE	-40°C to +85°C	16 Wide SO
MAX231EJD	-40°C to +85°C	14 CERDIP
MAX231MJD	-55°C to +125°C	14 CERDIP
MAX232CPE	0°C to +70°C	16 Plastic DIP
MAX232CSE	0°C to +70°C	16 Narrow SO
MAX232CWE	0°C to +70°C	16 Wide SO
MAX232C/D	0°C to +70°C	Dice*
MAX232EPE	-40°C to +85°C	16 Plastic DIP
MAX232ESE	-40°C to +85°C	16 Narrow SO
MAX232EWE	-40°C to +85°C	16 Wide SO
MAX232EJE	-40°C to +85°C	16 CERDIP
MAX232MJE	-55°C to +125°C	16 CERDIP
MAX232MLP	-55°C to +125°C	20 LCC
MAX232ACPE	0°C to +70°C	16 Plastic DIP
MAX232ACSE	0°C to +70°C	16 Narrow SO
MAX232ACWE	0°C to +70°C	16 Wide SO
MAX232AC/D	0°C to +70°C	Dice*
MAX232AEPE	-40°C to +85°C	16 Plastic DIP
MAX232AESE	-40°C to +85°C	16 Narrow SO
MAX232AEWE	-40°C to +85°C	16 Wide SO
MAX232AEJE	-40°C to +85°C	16 CERDIP
MAX232AMJE	-55°C to +125°C	16 CERDIP
MAX232AMLP	-55°C to +125°C	20 LCC
MAX233CPP	0°C to +70°C	20 Plastic DIP
MAX233EPP	-40°C to +85°C	20 Plastic DIP
MAX233ACPP	0°C to +70°C	20 Plastic DIP
MAX233ACWP	0°C to +70°C	20 Wide SO
MAX233AEPP	-40°C to +85°C	20 Plastic DIP
MAX233AEWP	-40°C to +85°C	20 Wide SO
MAX234CPE	0°C to +70°C	16 Plastic DIP
MAX234CWE	0°C to +70°C	16 Wide SO
MAX234C/D	0°C to +70°C	Dice*
MAX234EPE	-40°C to +85°C	16 Plastic DIP
MAX234EWE	-40°C to +85°C	16 Wide SO
MAX234EJE	-40°C to +85°C	16 CERDIP
MAX234MJE	-55°C to +125°C	16 CERDIP
MAX235CPG	0°C to +70°C	24 Wide Plastic DIP
MAX235EPG	-40°C to +85°C	24 Wide Plastic DIP
MAX235EDG	-40°C to +85°C	24 Ceramic SB
MAX235MDG	-55°C to +125°C	24 Ceramic SB
MAX236CNG	0°C to +70°C	24 Narrow Plastic DIP
MAX236CWG	0°C to +70°C	24 Wide SO
MAX236C/D	0°C to +70°C	Dice*
MAX236ENG	-40°C to +85°C	24 Narrow Plastic DIP
MAX236EWG	-40°C to +85°C	24 Wide SO
MAX236ERG	-40°C to +85°C	24 Narrow CERDIP
MAX236MRG	-55°C to +125°C	24 Narrow CERDIP
MAX237CNG	0°C to +70°C	24 Narrow Plastic DIP
MAX237CWG	0°C to +70°C	24 Wide SO
MAX237C/D	0°C to +70°C	Dice*
MAX237ENG	-40°C to +85°C	24 Narrow Plastic DIP
MAX237EWG	-40°C to +85°C	24 Wide SO
MAX237ERG	-40°C to +85°C	24 Narrow CERDIP
MAX237MRG	-55°C to +125°C	24 Narrow CERDIP
MAX238CNG	0°C to +70°C	24 Narrow Plastic DIP
MAX238CWG	0°C to +70°C	24 Wide SO
MAX238C/D	0°C to +70°C	Dice*
MAX238ENG	-40°C to +85°C	24 Narrow Plastic DIP

** Contact factory for dice specifications.*

+5V-Powered, Multichannel RS-232 Drivers/Receivers

Ordering Information (continued)

PART	TEMP. RANGE	PIN-PACKAGE
MAX238EWG	-40°C to +85°C	24 Wide SO
MAX238ERG	-40°C to +85°C	24 Narrow CERDIP
MAX238MRG	-55°C to +125°C	24 Narrow CERDIP
MAX239CNG	0°C to +70°C	24 Narrow Plastic DIP
MAX239CWG	0°C to +70°C	24 Wide SO
MAX239C/D	0°C to +70°C	Dice*
MAX239ENG	-40°C to +85°C	24 Narrow Plastic DIP
MAX239EWG	-40°C to +85°C	24 Wide SO
MAX239ERG	-40°C to +85°C	24 Narrow CERDIP
MAX239MRG	-55°C to +125°C	24 Narrow CERDIP
MAX240CMH	0°C to +70°C	44 Plastic FP
MAX240C/D	0°C to +70°C	Dice*
MAX241CAI	0°C to +70°C	28 SSOP
MAX241CWI	0°C to +70°C	28 Wide SO
MAX241C/D	0°C to +70°C	Dice*
MAX241EAI	-40°C to +85°C	28 SSOP
MAX241EWI	-40°C to +85°C	28 Wide SO
MAX242CAP	0°C to +70°C	20 SSOP
MAX242CPN	0°C to +70°C	18 Plastic DIP
MAX242CWN	0°C to +70°C	18 Wide SO
MAX242C/D	0°C to +70°C	Dice*
MAX242EPN	-40°C to +85°C	18 Plastic DIP
MAX242EWN	-40°C to +85°C	18 Wide SO
MAX242EJN	-40°C to +85°C	18 CERDIP
MAX242MJN	-55°C to +125°C	18 CERDIP

MAX243CPE	0°C to +70°C	16 Plastic DIP
MAX243CSE	0°C to +70°C	16 Narrow SO
MAX243CWE	0°C to +70°C	16 Wide SO
MAX243C/D	0°C to +70°C	Dice*
MAX243EPE	-40°C to +85°C	16 Plastic DIP
MAX243ESE	-40°C to +85°C	16 Narrow SO
MAX243EWE	-40°C to +85°C	16 Wide SO
MAX243EJE	-40°C to +85°C	16 CERDIP
MAX243MJE	-55°C to +125°C	16 CERDIP
MAX244CQH	0°C to +70°C	44 PLCC
MAX244C/D	0°C to +70°C	Dice*
MAX244EQH	-40°C to +85°C	44 PLCC
MAX245CPL	0°C to +70°C	40 Plastic DIP
MAX245C/D	0°C to +70°C	Dice*
MAX245EPL	-40°C to +85°C	40 Plastic DIP
MAX246CPL	0°C to +70°C	40 Plastic DIP
MAX246C/D	0°C to +70°C	Dice*
MAX246EPL	-40°C to +85°C	40 Plastic DIP
MAX247CPL	0°C to +70°C	40 Plastic DIP
MAX247C/D	0°C to +70°C	Dice*
MAX247EPL	-40°C to +85°C	40 Plastic DIP
MAX248CQH	0°C to +70°C	44 PLCC
MAX248C/D	0°C to +70°C	Dice*
MAX248EQH	-40°C to +85°C	44 PLCC
MAX249CQH	0°C to +70°C	44 PLCC
MAX249EQH	-40°C to +85°C	44 PLCC

** Contact factory for dice specifications.*

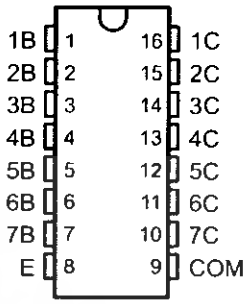
ULN2001A, ULN2002A, ULN2003A, ULN2004A, ULQ2003A, ULQ2004A
HIGH-VOLTAGE HIGH-CURRENT DARLINGTON TRANSISTOR ARRAY

The ULN2001A is obsolete and is no longer supplied.

SLRS027G – DECEMBER 1976 – REVISED JUNE 2004

- **500-mA-Rated Collector Current (Single Output)**
- **High-Voltage Outputs . . . 50 V**
- **Output Clamp Diodes**
- **Inputs Compatible With Various Types of Logic**
- **Relay-Driver Applications**

ULN2001A . . . D OR N PACKAGE
ULN2002A . . . N PACKAGE
ULN2003A . . . D, N, NS, OR PW PACKAGE
ULN2004A . . . D, N, OR NS PACKAGE
ULQ2003A, ULQ2004A . . . D OR N PACKAGE
(TOP VIEW)



description/ordering information

The ULN2001A, ULN2002A, ULN2003A, ULN2004A, ULQ2003A, and ULQ2004A are high-voltage, high-current Darlington transistor arrays. Each consists of seven npn Darlington pairs that feature high-voltage outputs with common-cathode clamp diodes for switching inductive loads. The collector-current rating of a single Darlington pair is 500 mA. The Darlington pairs can be paralleled for higher current capability. Applications include relay drivers, hammer drivers, lamp drivers, display drivers (LED and gas discharge), line drivers, and logic buffers. For 100-V (otherwise interchangeable) versions of the ULN2003A and ULN2004A, see the SN75468 and SN75469, respectively.

ORDERING INFORMATION

T_A	PACKAGE†		ORDERABLE PART NUMBER	TOP-SIDE MARKING
−20°C to 70°C	PDIP (N)	Tube of 25	ULN2002AN	ULN2002AN
			ULN2003AN	ULN2003AN
			ULN2004AN	ULN2004AN
	SOIC (D)	Tube of 40	ULN2003AD	ULN2003A
		Reel of 2500	ULN2003ADR	
		Tube of 40	ULN2004AD	ULN2004A
		Reel of 2500	ULN2004ADR	
	SOP (NS)	Reel of 2000	ULN2003ANSR	ULN2003A
			ULN2004ANSR	ULN2004A
	TSSOP (PW)	Tube of 90	ULN2003APW	UN2003A
		Reel of 2000	ULN2003APWR	
−40°C to 85°C	PDIP (N)	Tube of 25	ULQ2003AN	ULQ2003A
			ULQ2004AN	ULQ2004AN
	SOIC (D)	Tube of 40	ULQ2003AD	ULQ2003A
		Reel of 2500	ULQ2003ADR	ULQ2003A
		Tube of 40	ULQ2004AD	ULQ2004A
		Reel of 2500	ULQ2004ADR	ULQ2004A

† Package drawings, standard packing quantities, thermal data, symbolization, and PCB design guidelines are available at www.ti.com/sc/package.

Please be aware that an important notice concerning availability, standard warranty, and use in critical applications of Texas Instruments semiconductor products and disclaimers thereto appears at the end of this data sheet.

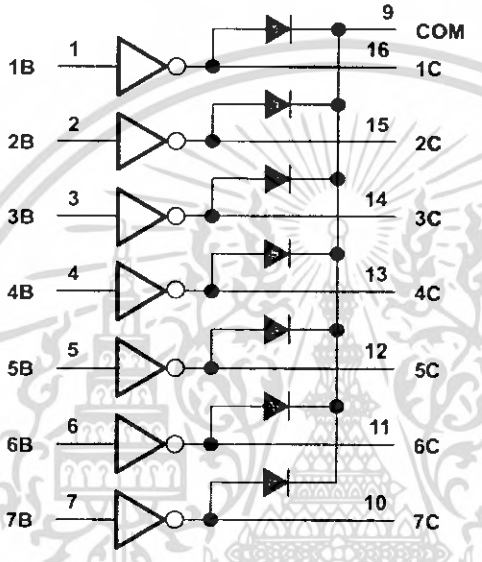
TEXAS INSTRUMENTS
POST OFFICE BOX 655303 • DALLAS, TEXAS 75265

The ULN2001A is obsolete and is no longer supplied.

description/ordering information (continued)

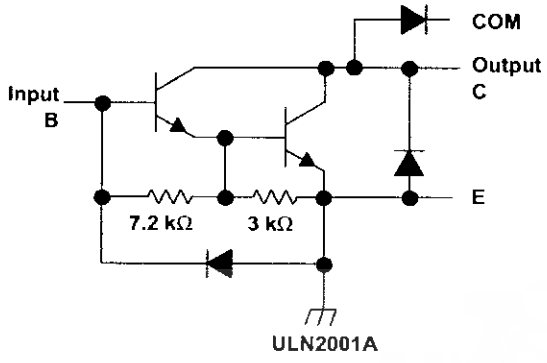
The ULN2001A is a general-purpose array and can be used with TTL and CMOS technologies. The ULN2002A is designed specifically for use with 14-V to 25-V PMOS devices. Each input of this device has a Zener diode and resistor in series to control the input current to a safe limit. The ULN2003A and ULQ2003A have a 2.7-kΩ series base resistor for each Darlington pair for operation directly with TTL or 5-V CMOS devices. The ULN2004A and ULQ2004A have a 10.5-kΩ series base resistor to allow operation directly from CMOS devices that use supply voltages of 6 V to 15 V. The required input current of the ULN/ULQ2004A is below that of the ULN/ULQ2003A, and the required voltage is less than that required by the ULN2002A.

logic diagram

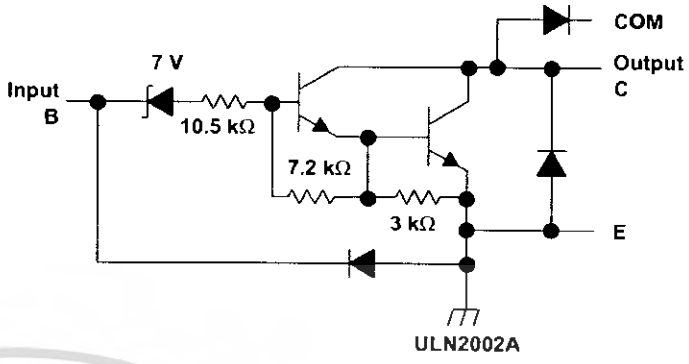


The ULN2001A is obsolete and is no longer supplied.

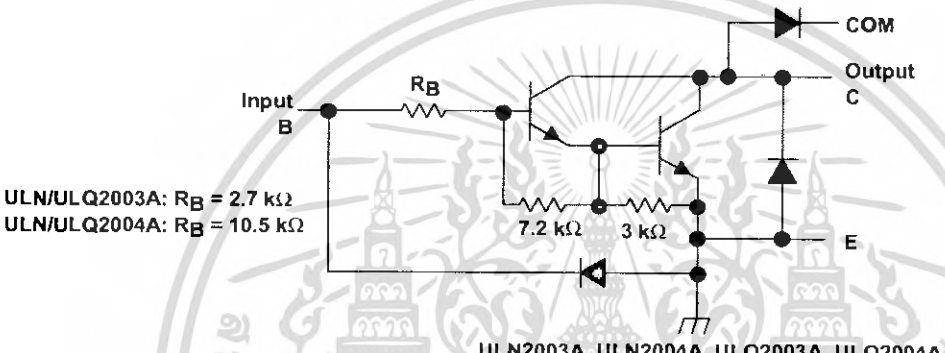
schematics (each Darlington pair)



ULN2001A



ULN2002A



ULN2003A, ULN2004A, ULQ2003A, ULQ2004A

All resistor values shown are nominal.

The ULN2001A is obsolete and is no longer supplied.

absolute maximum ratings at 25°C free-air temperature (unless otherwise noted)†

Collector-emitter voltage ..... 50 V
Clamp diode reverse voltage (see Note 1) ..... 50 V
Input voltage, V_I (see Note 1) ..... 30 V
Peak collector current (see Figures 14 and 15) ..... 500 mA
Output clamp current, I_{OK} ..... 500 mA
Total emitter-terminal current ..... −2.5 A
Operating free-air temperature range, T_A, ULN200xA ..... −20°C to 70°C
ULQ200xA ..... −40°C to 85°C
ULQ200xAT ..... −40°C to 105°C
Package thermal impedance, θ_{JA} (see Notes 2 and 3): D package ..... 73°C/W
N package ..... 67°C/W
NS package ..... 64°C/W
PW package ..... 108°C/W
Package thermal impedance, θ_{JC} (see Notes 4 and 5): D package ..... 36°C/W
N package ..... 54°C/W
Operating virtual junction temperature, T_J ..... 150°C
Lead temperature 1,6 mm (1/16 inch) from case for 10 seconds ..... 260°C
Storage temperature range, T_{stg} ..... −65°C to 150°C

† Stresses beyond those listed under "absolute maximum ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated under "recommended operating conditions" is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

NOTES: 1. All voltage values are with respect to the emitter/substrate terminal E, unless otherwise noted.
2. Maximum power dissipation is a function of $T_J(max)$, θ_{JA}, and T_A. The maximum allowable power dissipation at any allowable ambient temperature is $P_D = (T_J(max) - T_A)/\theta_{JA}$. Operating at the absolute maximum T_J of 150°C can affect reliability.
3. The package thermal impedance is calculated in accordance with JESD 51-7.
4. Maximum power dissipation is a function of $T_J(max)$, θ_{JC}, and T_C. The maximum allowable power dissipation at any allowable case temperature is $P_D = (T_J(max) - T_C)/\theta_{JC}$. Operating at the absolute maximum T_J of 150°C can affect reliability.
5. The package thermal impedance is calculated in accordance with MIL-STD-883.

electrical characteristics, T_A = 25°C (unless otherwise noted)

PARAMETER		TEST FIGURE	TEST CONDITIONS		ULN2001A			ULN2002A			UNIT
					MIN	TYP	MAX	MIN	TYP	MAX	
$V_{I(on)}$	On-state input voltage	6	V_{CE} = 2 V,	I_C = 300 mA						13	V
$V_{CE(sat)}$	Collector-emitter saturation voltage	5	I_I = 250 µA,	I_C = 100 mA		0.9	1.1		0.9	1.1	V
			I_I = 350 µA,	I_C = 200 mA		1	1.3		1	1.3	
			I_I = 500 µA,	I_C = 350 mA		1.2	1.6		1.2	1.6	
V_F	Clamp forward voltage	8	I_F = 350 mA			1.7	2		1.7	2	V
I_{CEX}	Collector cutoff current	1	V_{CE} = 50 V,	I_I = 0			50			50	µA
		2	V_{CE} = 50 V, T_A = 70°C	I_I = 0			100			100	
				V_I = 6 V						500	
$I_{I(off)}$	Off-state input current	3	V_{CE} = 50 V, T_A = 70°C	I_C = 500 µA,	50	65		50	65		µA
I_I	Input current	4	V_I = 17 V						0.82	1.25	mA
I_R	Clamp reverse current	7	V_R = 50 V,	T_A = 70°C			100			100	µA
			V_R = 50 V				50			50	
h_{FE}	Static forward-current transfer ratio	5	V_{CE} = 2 V,	I_C = 350 mA	1000						
C_i	Input capacitance		V_I = 0,	f = 1 MHz		15	25		15	25	pF

The ULN2001A is obsolete and is no longer supplied.

electrical characteristics, T_A = 25°C (unless otherwise noted) (continued)

PARAMETER		TEST FIGURE	TEST CONDITIONS		ULN2003A			ULN2004A			UNIT
					MIN	TYP	MAX	MIN	TYP	MAX	
$V_{I(on)}$	On-state input voltage	6	V_{CE} = 2 V	I_C = 125 mA						5	V
				I_C = 200 mA			2.4			6	
				I_C = 250 mA			2.7				
				I_C = 275 mA						7	
				I_C = 300 mA			3				
				I_C = 350 mA						8	
$V_{CE(sat)}$	Collector-emitter saturation voltage	5	I_I = 250 µA,	I_C = 100 mA		0.9	1.1		0.9	1.1	V
			I_I = 350 µA,	I_C = 200 mA		1	1.3		1	1.3	
			I_I = 500 µA,	I_C = 350 mA		1.2	1.6		1.2	1.6	
I_{CEX}	Collector cutoff current	1	V_{CE} = 50 V,	I_I = 0			50			50	µA
		2	V_{CE} = 50 V, T_A = 70°C	I_I = 0			100			100	
				V_I = 1 V						500	
V_F	Clamp forward voltage	8	I_F = 350 mA			1.7	2		1.7	2	V
$I_{I(off)}$	Off-state input current	3	V_{CE} = 50 V, T_A = 70°C	I_C = 500 µA,	50	65		50	65		µA
I_I	Input current	4	V_I = 3.85 V			0.93	1.35				mA
			V_I = 5 V						0.35	0.5	
			V_I = 12 V						1	1.45	
I_R	Clamp reverse current	7	V_R = 50 V				50			50	µA
			V_R = 50 V,	T_A = 70°C			100			100	
C_i	Input capacitance		V_I = 0,	f = 1 MHz		15	25		15	25	pF

The ULN2001A is obsolete and is no longer supplied.

electrical characteristics over recommended operating conditions (unless otherwise noted)

PARAMETER		TEST FIGURE	TEST CONDITIONS		ULQ2003A			ULQ2004A			UNIT
					MIN	TYP	MAX	MIN	TYP	MAX	
$V_{I(on)}$	On-state input voltage	6	V_{CE} = 2 V	I_C = 125 mA						5	V
				I_C = 200 mA			2.7			6	
				I_C = 250 mA			2.9				
				I_C = 275 mA						7	
				I_C = 300 mA			3				
				I_C = 350 mA						8	
$V_{CE(sat)}$	Collector-emitter saturation voltage	5	I_I = 250 µA,	I_C = 100 mA		0.9	1.2		0.9	1.1	V
			I_I = 350 µA,	I_C = 200 mA		1	1.4		1	1.3	
			I_I = 500 µA,	I_C = 350 mA		1.2	1.7		1.2	1.6	
I_{CEX}	Collector cutoff current	1	V_{CE} = 50 V,	I_I = 0			100			50	µA
		2	V_{CE} = 50 V	I_I = 0						100	
				V_I = 1 V						500	
V_F	Clamp forward voltage	8	I_F = 350 mA			1.7	2.3		1.7	2	V
$I_{I(off)}$	Off-state input current	3	V_{CE} = 50 V,	I_C = 500 µA	50	65		50	65		µA
I_I	Input current	4	V_I = 3.85 V			0.93	1.35				mA
			V_I = 5 V						0.35	0.5	
			V_I = 12 V						1	1.45	
I_R	Clamp reverse current	7	V_R = 50 V,	T_A = 25°C			100			50	µA
			V_R = 50 V				100			100	
C_i	Input capacitance		V_I = 0,	f = 1 MHz		15	25		15	25	pF

switching characteristics, T_A = 25°C

PARAMETER		TEST CONDITIONS	ULN2001A, ULN2002A, ULN2003A, ULN2004A			UNIT
			MIN	TYP	MAX	
t_{PLH}	Propagation delay time, low- to high-level output	See Figure 9		0.25	1	µs
t_{PHL}	Propagation delay time, high- to low-level output	See Figure 9		0.25	1	µs
V_{OH}	High-level output voltage after switching	V_S = 50 V, I_O = 300 mA, See Figure 10	V_S–20			mV

switching characteristics over recommended operating conditions (unless otherwise noted)

PARAMETER		TEST CONDITIONS	ULQ2003A, ULQ2004A			UNIT
			MIN	TYP	MAX	
t_{PLH}	Propagation delay time, low- to high-level output	See Figure 9		1	10	µs
t_{PHL}	Propagation delay time, high- to low-level output	See Figure 9		1	10	µs
V_{OH}	High-level output voltage after switching	V_S = 50 V, I_O = 300 mA, See Figure 10	V_S–500			mV

The ULN2001A is obsolete and is no longer supplied.

PARAMETER MEASUREMENT INFORMATION

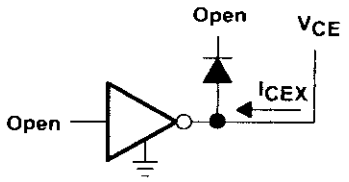


Figure 1. I_{CEX} Test Circuit

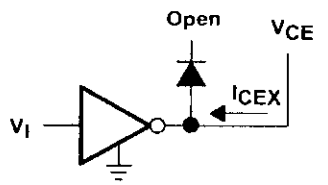


Figure 2. I_{CEX} Test Circuit

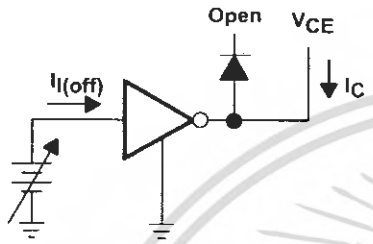


Figure 3. $I_{I(off)}$ Test Circuit

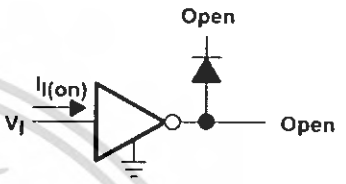
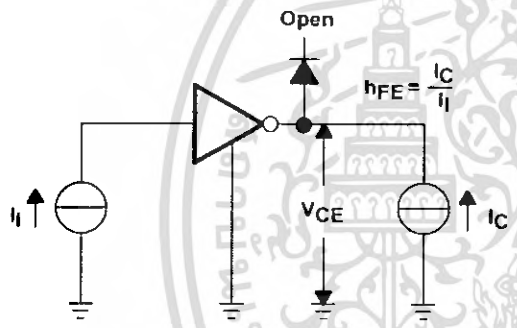


Figure 4. I_I Test Circuit



NOTE: I_I is fixed for measuring $V_{CE(sat)}$, variable for measuring h_{FE}.

Figure 5. h_{FE}, $V_{CE(sat)}$ Test Circuit

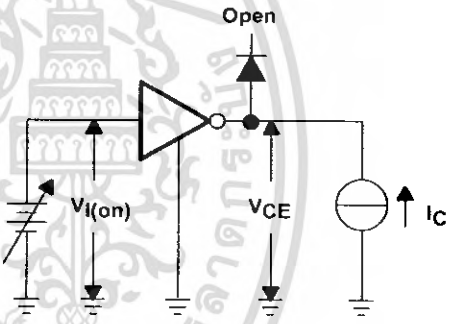


Figure 6. $V_{I(on)}$ Test Circuit

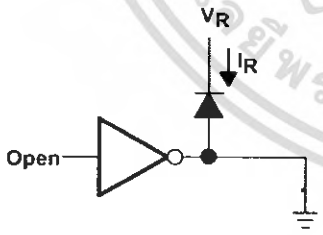


Figure 7. I_R Test Circuit

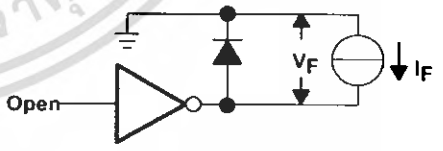


Figure 8. V_F Test Circuit

The ULN2001A is obsolete and is no longer supplied.

PARAMETER MEASUREMENT INFORMATION

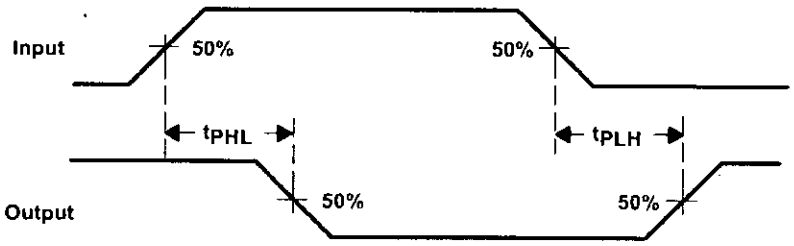
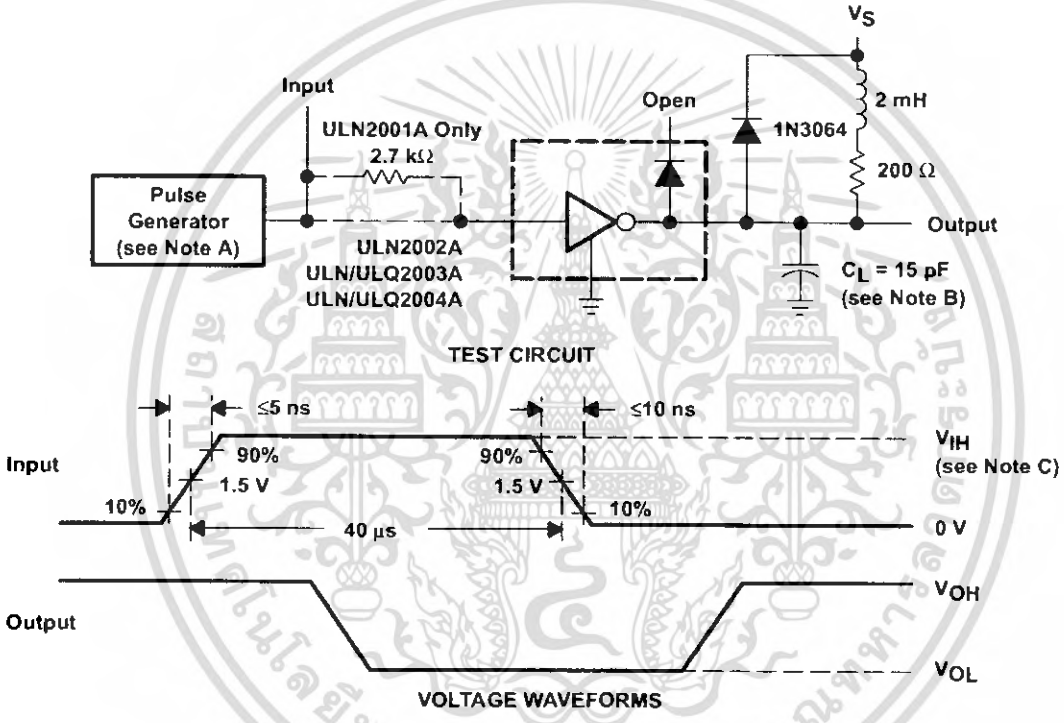


Figure 9. Propagation Delay-Time Waveforms



NOTES: A. The pulse generator has the following characteristics: PRR = 12.5 kHz, Z_O = 50 Ω.
B. C_L includes probe and jig capacitance.
C. For testing the ULN2001A, the ULN2003A, and the ULQ2003A, V_{IH} = 3 V; for the ULN2002A, V_{IH} = 13 V; for the ULN2004A and the ULQ2004A, V_{IH} = 8 V.

Figure 10. Latch-Up Test Circuit and Voltage Waveforms

The ULN2001A is obsolete and is no longer supplied.

TYPICAL CHARACTERISTICS

COLLECTOR-EMITTER SATURATION VOLTAGE vs COLLECTOR CURRENT (ONE DARLINGTON)

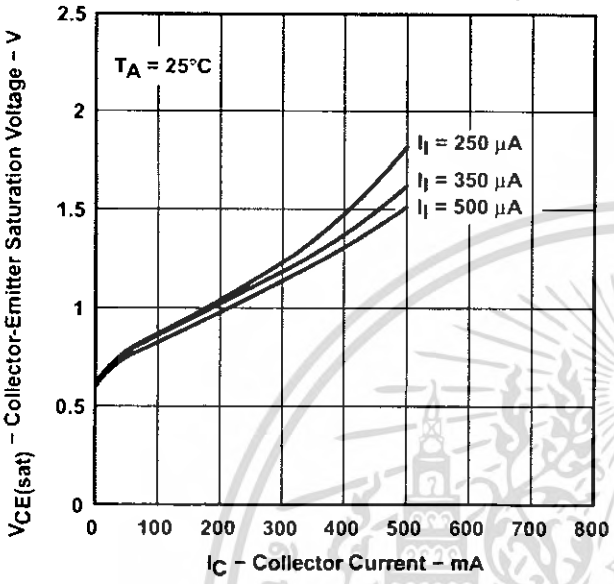


Figure 11

COLLECTOR-EMITTER SATURATION VOLTAGE vs TOTAL COLLECTOR CURRENT (TWO DARLINGTONS IN PARALLEL)

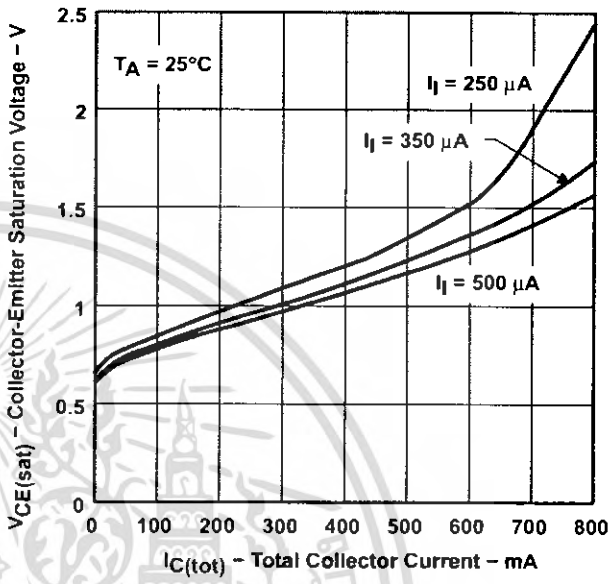


Figure 12

COLLECTOR CURRENT vs INPUT CURRENT

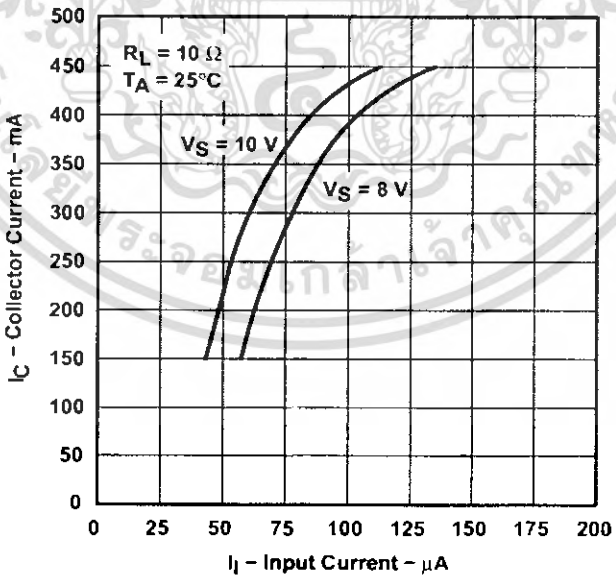


Figure 13

The ULN2001A is obsolete and is no longer supplied.

THERMAL INFORMATION

D PACKAGE
MAXIMUM COLLECTOR CURRENT
vs
DUTY CYCLE

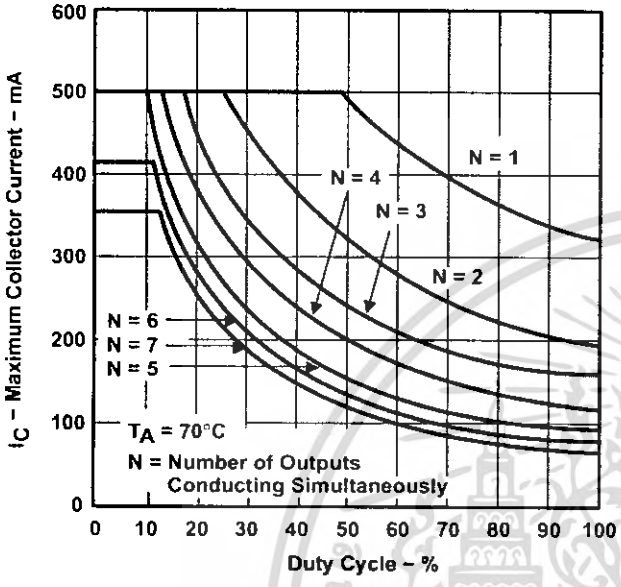


Figure 14

N PACKAGE
MAXIMUM COLLECTOR CURRENT
vs
DUTY CYCLE

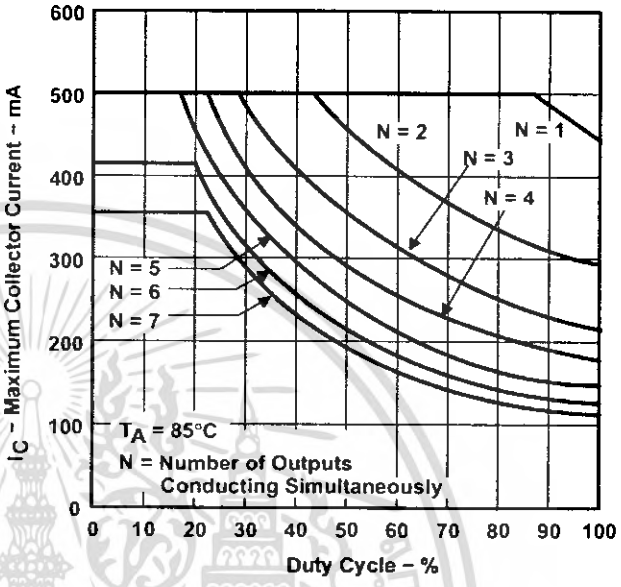


Figure 15

The ULN2001A is obsolete and is no longer supplied.

APPLICATION INFORMATION

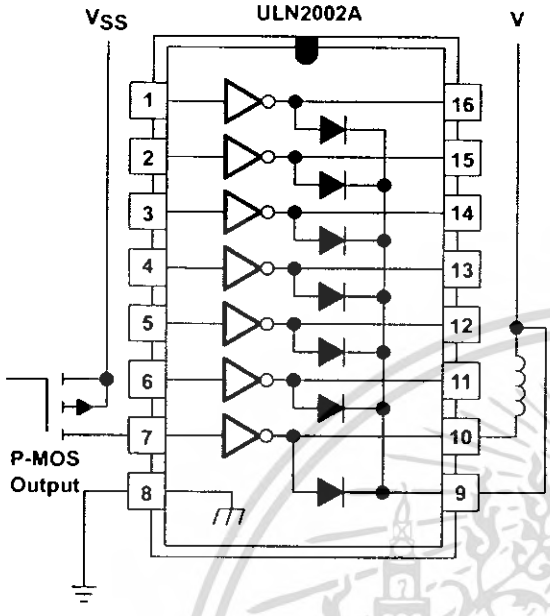


Figure 16. P-MOS to Load

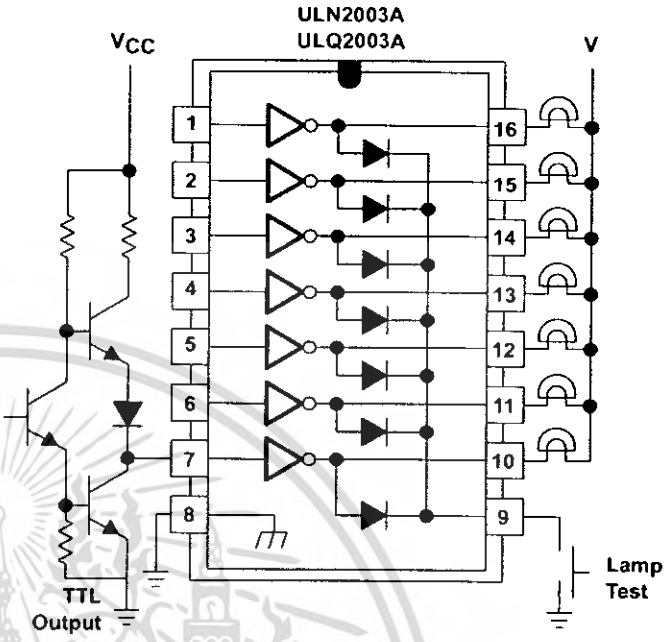


Figure 17. TTL to Load

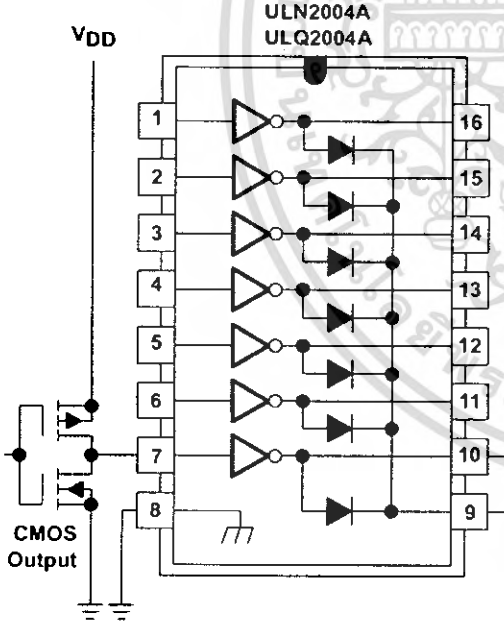


Figure 18. Buffer for Higher Current Loads

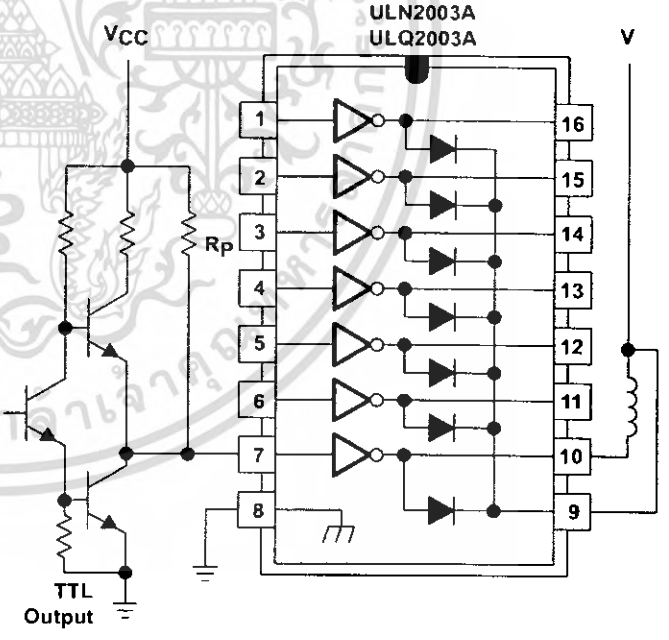


Figure 19. Use of Pullup Resistors to Increase Drive Current

PACKAGING INFORMATION

Orderable Device	Status [1]	Package Type	Package Drawing	Pins	Package Qty	Eco Plan [2]	Lead/Ball Finish	MSL Peak Temp [3]
ULN2001AD	OBSOLETE	SOIC	D	16		None	Call TI	Call TI
ULN2001ADR	OBSOLETE	SOIC	D	16		None	Call TI	Call TI
ULN2001AN	OBSOLETE	PDIP	N	16		None	Call TI	Call TI
ULN2002AD	OBSOLETE	SOIC	D	16		None	Call TI	Call TI
ULN2002AN	ACTIVE	PDIP	N	16	25	Pb-Free (RoHS)	CU NIPDAU	Level-NC-NC-NC
ULN2003AD	ACTIVE	SOIC	D	16	40	Pb-Free (RoHS)	CU NIPDAU	Level-2-260C-1 YEAR/ Level-1-235C-UNLIM
ULN2003ADR	ACTIVE	SOIC	D	16	2500	Green (RoHS & no Sb/Br)	Call TI	Level-1-260C-UNLIM
ULN2003AJ	OBSOLETE	CDIP	J	16		None	Call TI	Call TI
ULN2003AN	ACTIVE	PDIP	N	16	25	Pb-Free (RoHS)	CU NIPDAU	Level-NC-NC-NC
ULN2003ANSR	ACTIVE	SO	NS	16	2000	Pb-Free (RoHS)	CU NIPDAU	Level-2-260C-1 YEAR/ Level-1-235C-UNLIM
ULN2003APW	ACTIVE	TSSOP	PW	16	90	Pb-Free (RoHS)	CU NIPDAU	Level-1-250C-UNLIM
ULN2003APWR	ACTIVE	TSSOP	PW	16	2000	Pb-Free (RoHS)	CU NIPDAU	Level-1-250C-UNLIM
ULN2004AD	ACTIVE	SOIC	D	16	40	Pb-Free (RoHS)	CU NIPDAU	Level-2-260C-1 YEAR/ Level-1-235C-UNLIM
ULN2004ADR	ACTIVE	SOIC	D	16	2500	Green (RoHS & no Sb/Br)	Call TI	Level-1-260C-UNLIM
ULN2004AN	ACTIVE	PDIP	N	16	25	Pb-Free (RoHS)	CU NIPDAU	Level-NC-NC-NC
ULN2004ANSR	ACTIVE	SO	NS	16	2000	Pb-Free (RoHS)	CU NIPDAU	Level-2-260C-1 YEAR/ Level-1-235C-UNLIM
ULQ2003AD	ACTIVE	SOIC	D	16	40	Pb-Free (RoHS)	CU NIPDAU	Level-2-250C-1 YEAR/ Level-1-235C-UNLIM
ULQ2003ADR	ACTIVE	SOIC	D	16	2500	Pb-Free (RoHS)	CU NIPDAU	Level-2-250C-1 YEAR/ Level-1-235C-UNLIM
ULQ2003AN	ACTIVE	PDIP	N	16	25	None	Call TI	Level-NC-NC-NC
ULQ2004AD	ACTIVE	SOIC	D	16	40	Pb-Free (RoHS)	CU NIPDAU	Level-2-250C-1 YEAR/ Level-1-235C-UNLIM
ULQ2004ADR	ACTIVE	SOIC	D	16	2500	Pb-Free (RoHS)	CU NIPDAU	Level-2-250C-1 YEAR/ Level-1-235C-UNLIM
ULQ2004AN	ACTIVE	PDIP	N	16	25	None	Call TI	Level-NC-NC-NC

[1] The marketing status values are defined as follows:
ACTIVE: Product device recommended for new designs.
LIFEBUY: TI has announced that the device will be discontinued, and a lifetime-buy period is in effect.
NRND: Not recommended for new designs. Device is in production to support existing customers, but TI does not recommend using this part in a new design.
PREVIEW: Device has been announced but is not in production. Samples may or may not be available.
OBSOLETE: TI has discontinued the production of the device.

[2] Eco Plan - May not be currently available - please check http://www.ti.com/productcontent for the latest availability information and additional product content details.
None: Not yet available Lead (Pb-Free).
Pb-Free (RoHS): TI's terms "Lead-Free" or "Pb-Free" mean semiconductor products that are compatible with the current RoHS requirements

for all 6 substances, including the requirement that lead not exceed 0.1% by weight in homogeneous materials. Where designed to be soldered at high temperatures, TI Pb-Free products are suitable for use in specified lead-free processes.
Green (RoHS & no Sb/Br): TI defines "Green" to mean "Pb-Free" and in addition, uses package materials that do not contain halogens, including bromine (Br) or antimony (Sb) above 0.1% of total product weight.

(3) MSL, Peak Temp. – The Moisture Sensitivity Level rating according to the JEDECindustry standard classifications, and peak solder temperature.

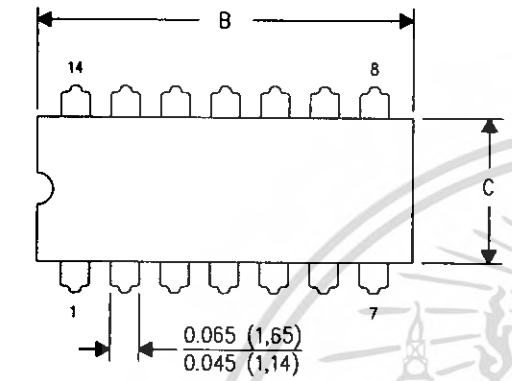
Important Information and Disclaimer: The information provided on this page represents TI's knowledge and belief as of the date that it is provided. TI bases its knowledge and belief on information provided by third parties, and makes no representation or warranty as to the accuracy of such information. Efforts are underway to better integrate information from third parties. TI has taken and continues to take reasonable steps to provide representative and accurate information but may not have conducted destructive testing or chemical analysis on incoming materials and chemicals. TI and TI suppliers consider certain information to be proprietary, and thus CAS numbers and other limited information may not be available for release.

In no event shall TI's liability arising out of such information exceed the total purchase price of the TI part(s) at issue in this document sold by TI to Customer on an annual basis.

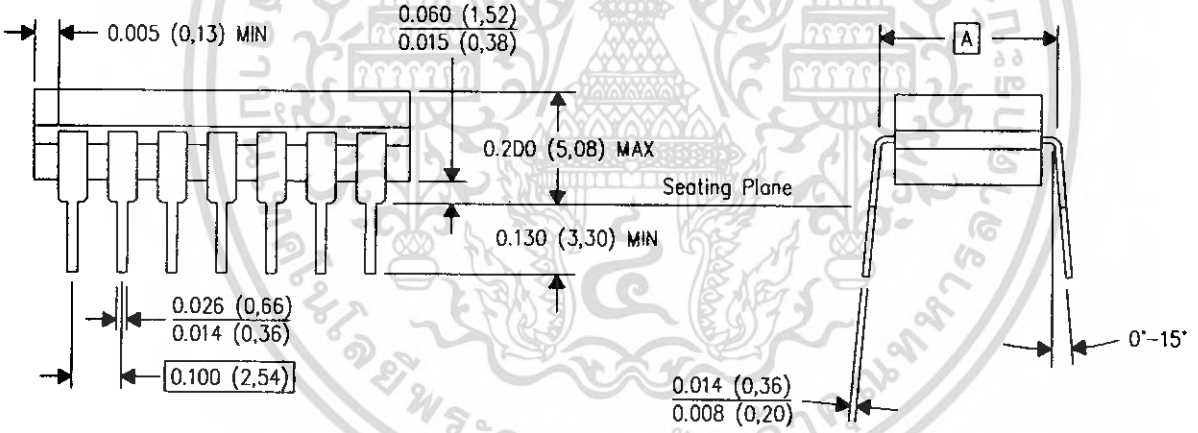
J (R–GDIP–T**)

14 LEADS SHOWN

CERAMIC DUAL IN–LINE PACKAGE



DIM \ PINS **	14	16	18	20
A	0.300 (7,62) BSC	0.300 (7,62) BSC	0.300 (7,62) BSC	0.300 (7,62) BSC
B MAX	0.785 (19,94)	.840 (21,34)	0.960 (24,38)	1.060 (26,92)
B MIN	—	—	—	—
C MAX	0.300 (7,62)	0.300 (7,62)	0.310 (7,87)	0.300 (7,62)
C MIN	0.245 (6,22)	0.245 (6,22)	0.220 (5,59)	0.245 (6,22)



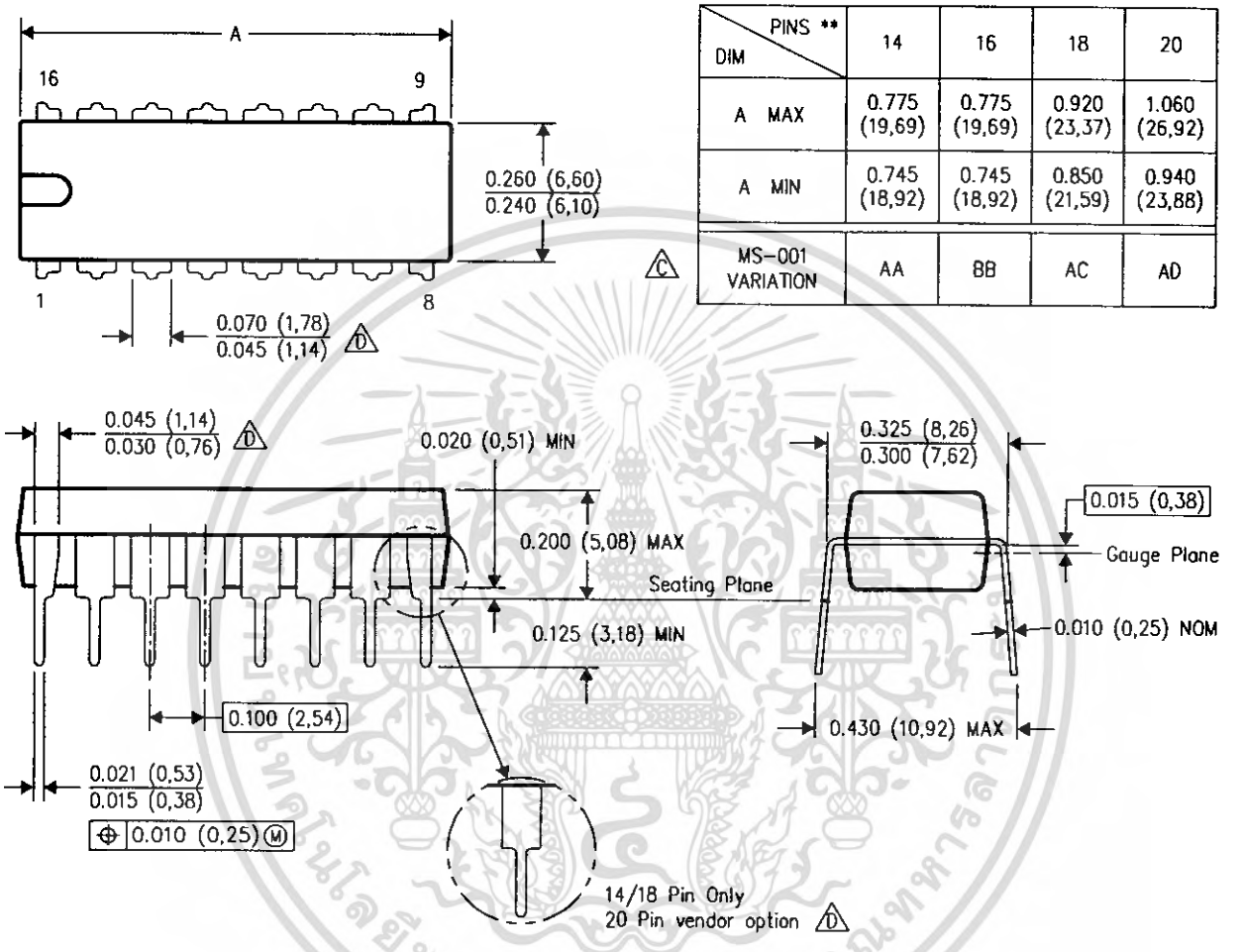
4040083/F 03/03

NOTES:
A. All linear dimensions are in inches (millimeters).
B. This drawing is subject to change without notice.
C. This package is hermetically sealed with a ceramic lid using glass frit.
D. Index point is provided on cap for terminal identification only on press ceramic glass frit seal only.
E. Falls within MIL STD 1835 GDIP1–T14, GDIP1–T16, GDIP1–T18 and GDIP1–T20.

N (R–PDIP–T**)

16 PINS SHOWN

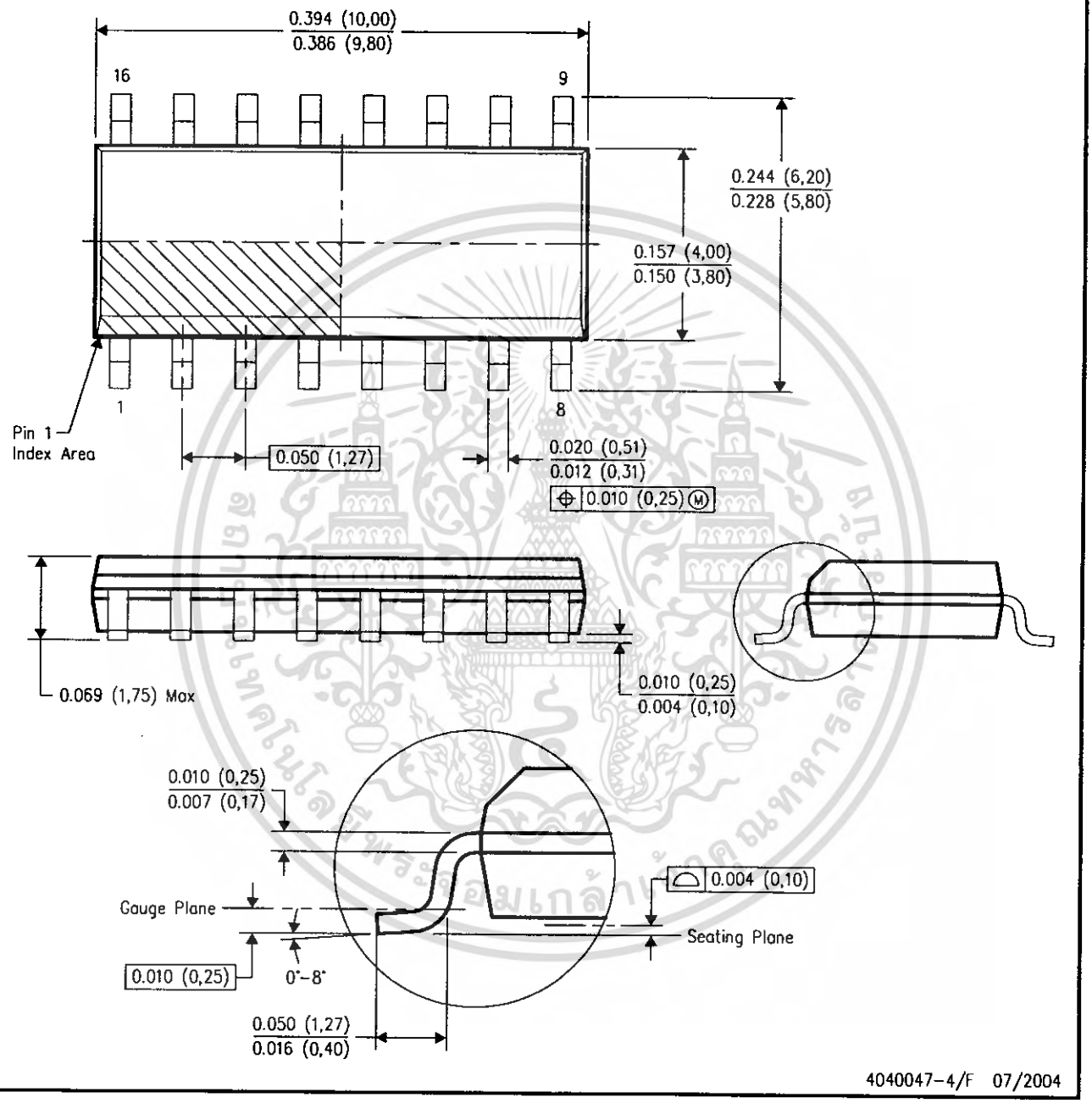
PLASTIC DUAL–IN–LINE PACKAGE



DIM \ PINS **	14	16	18	20
A MAX	0.775 (19,69)	0.775 (19,69)	0.920 (23,37)	1.060 (26,92)
A MIN	0.745 (18,92)	0.745 (18,92)	0.850 (21,59)	0.940 (23,88)
MS–001 VARIATION	AA	BB	AC	AD

NOTES:
A. All linear dimensions are in inches (millimeters).
B. This drawing is subject to change without notice.
C. Falls within JEDEC MS–001, except 18 and 20 pin minimum body length (Dim A).
D. The 20 pin end lead shoulder width is a vendor option, either half or full width.

D (R–PDSO–G16) PLASTIC SMALL–OUTLINE PACKAGE

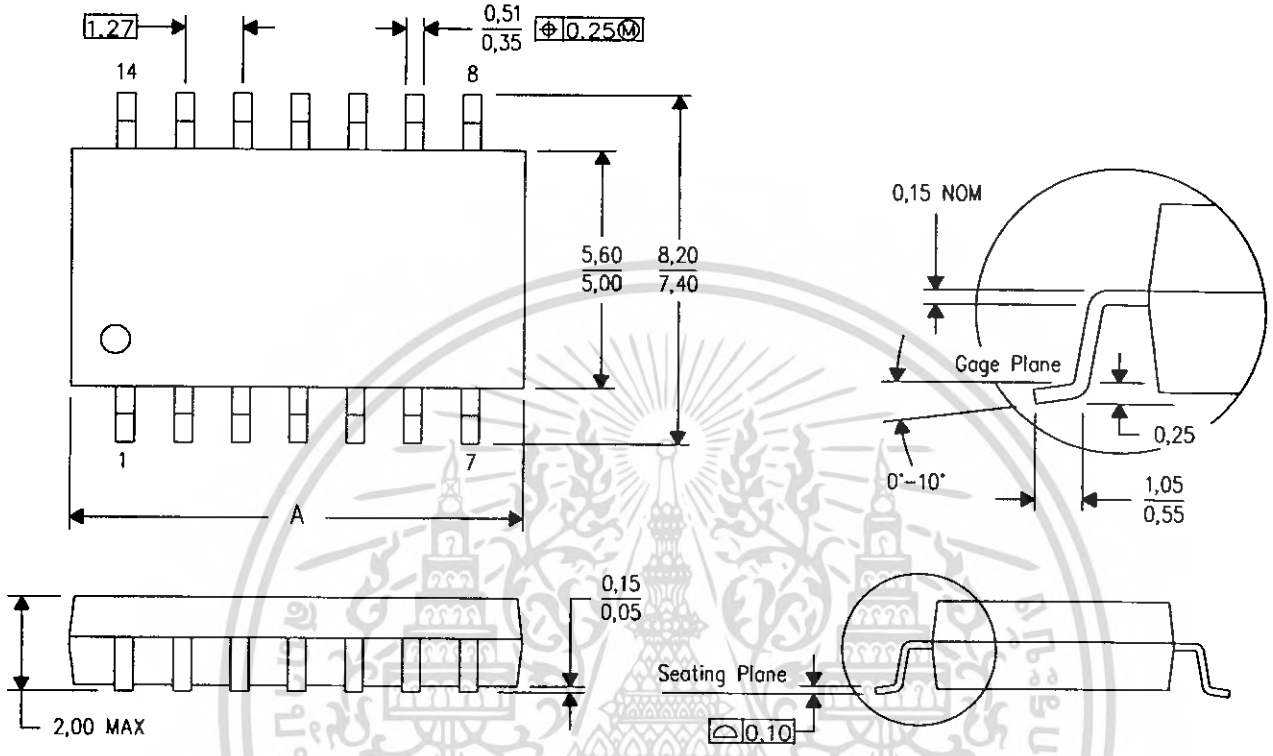


NOTES:
A. All linear dimensions are in inches (millimeters).
B. This drawing is subject to change without notice.
C. Body dimensions do not include mold flash or protrusion not to exceed 0.006 (0,15).
D. Falls within JEDEC MS–012 variation AC.

MECHANICAL DATA

NS (R-PDSO-G)** **PLASTIC SMALL-OUTLINE PACKAGE**

14-PINS SHOWN



DIM \ PINS **	14	16	20	24
A MAX	10,50	10,50	12,90	15,30
A MIN	9,90	9,90	12,30	14,70

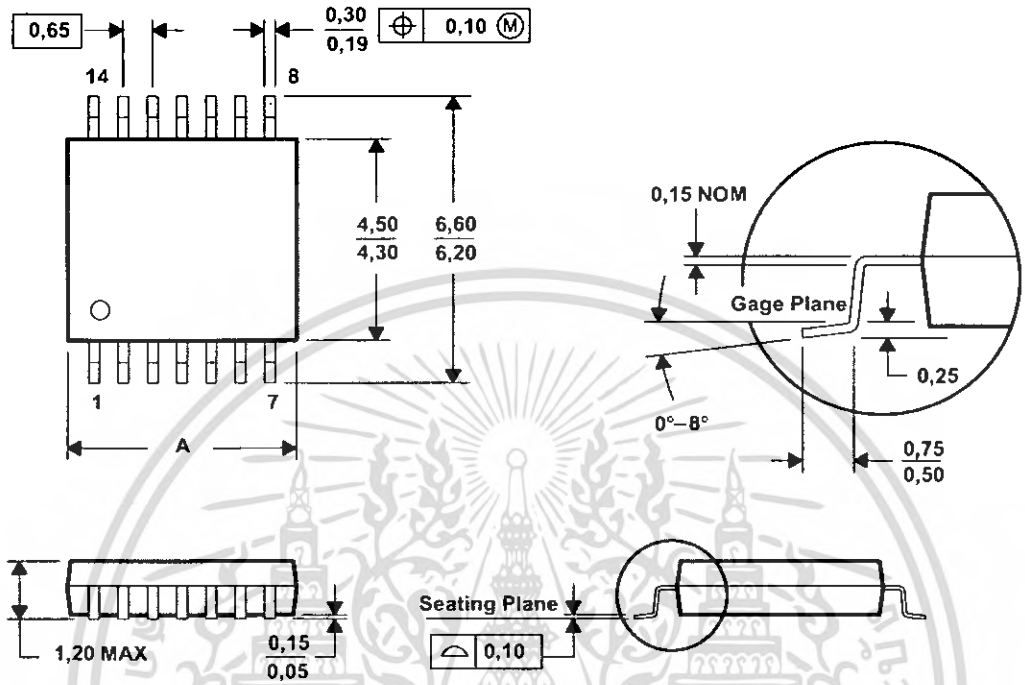
4040062/C 03/03

NOTES:
A. All linear dimensions are in millimeters.
B. This drawing is subject to change without notice.
C. Body dimensions do not include mold flash or protrusion, not to exceed 0,15.

MTSS001C – JANUARY 1995 – REVISED FEBRUARY 1999

PW (R-PDSO-G)** **PLASTIC SMALL-OUTLINE PACKAGE**

14 PINS SHOWN



DIM \ PINS **	8	14	16	20	24	28
A MAX	3,10	5,10	5,10	6,60	7,90	9,80
A MIN	2,90	4,90	4,90	6,40	7,70	9,60

4040064/F 01/97

NOTES: A. All linear dimensions are in millimeters.
B. This drawing is subject to change without notice.
C. Body dimensions do not include mold flash or protrusion not to exceed 0,15.
D. Falls within JEDEC MO-153

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265

หนังสืออ้างอิง

1. นิรุธ อำนวยศิลป์, "Network and Protocols Programming using C/C++"

2. นิรุธ อำนวยศิลป์, ""C++ การออกแบบชุดคำสั่ง และการประยุกต์ Design andits applications

3. นิรุธ อำนวยศิลป์, "C Programming เขียนโปรแกรมภาษาซี"

4. นิรุธ อำนวยศิลป์, "Visual C++ and MFC Programming เขียนโปรแกรมบนวินโดวส์ ด้วย Visual C++ และ MFC"

5. ยุทธนา ลีลาศวัฒนกุล, "คู่มือการเขียนวินโดว์ขั้นสูงด้วย Visual C++", หจก.ไทยเจริญการพิมพ์, พิมพ์ครั้งที่ 1/สิงหาคม 2546

6. ธีรบูลย์ หล่อวิเชียรรุ่ง, ปฏิบัติการไมโครคอนโทรลเลอร์ MCS-51 ด้วยโปรแกรมภาษา C

7. วรพจน์ กรแก้ววัฒนกุล, เรียนรู้และปฏิบัติการไมโครคอนโทรลเลอร์ MCS-51

8. รศ.สมยศ จุณณะปิยะม, การประยุกต์ใช้งานไมโครคอนโทรลเลอร์