การสื่อสารข้อมูล RS-485

RS-485 COMMUNICATION

โดย

นายพยัต ชินวิไล

นายวินัย เสือสีนวล

นายอดุลย์ เพ็ชร์ศิริพันธุ์

นายอนุชิต สิทธิสินธุ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

ภาควิชาเทคโนโลยีการวัดคุมทางอุตสาหกรรม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2534

ปริญญานิพนธ์ ปีการศึกษา 2534

ภาควิชา เทคโนโลยีการวัดคุมทางอุตสาหกรรม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีเจ้าคุณทหารลาดกระบัง

เรื่อง การสื่อสารข้อมูล RS - 485

ผู้จัดทำ

นาย	พยัต	ชินวิไล	33 . 161215
นาย	วินัย	เสือสีนวล	33 . 161219
นาย	อดุลย์	เพ็ชร์ศิริพันธุ์	33 . 161235
นาย	อนุชิต	สิทธิสินธุ์	33 . 161236

..... อาจารย์ที่ปรึกษา

(อ. สุพรรณ กุลพาณิชย์)

การสื่อสารข้อมูล RS - 485

นาย พยัต ชินวิไล เลขประจำตัว 33 . 161215

นาย วินัย เสือสีนวล เลขประจำตัว 33 . 161219

นาย อดุลย์ เพ็ชร์ศิริพันธุ์ เลขประจำตัว 33 . 161235

นาย อนุชิต สิทธิสินธุ์ เลขประจำตัว 33 . 161236

อาจารย์ที่ปรึกษา อ. สุนทรณ กุลพาณิชย์

ปีการศึกษา 2534

บทคัดย่อ

จุดประสงค์หลักของโครงงานนี้พอจะแบ่งเป็นหัวข้อได้ดังนี้

1. การศึกษาหลักการสื่อสาร ของอุปกรณ์รับ-ส่ง ข้อมูลระยะไกลโดยไม่ต้องใช้ MODEM

2. การออกแบบซอฟแวร์ซึ่งใช้ในการควบคุมการทำงานของระบบเครือข่ายสื่อสารข้อมูล

3. การออกแบบอุปกรณ์ที่มีความสามารถในการควบคุม การติดต่อสื่อสาร ของระบบเครือข่ายสื่อสารข้อมูลระหว่างเครื่องมาสเตอร์ (MASTER) กับเครื่องสเลฟ (SLAVE) ซึ่งในโครงการนี้เครื่องมาสเตอร์และเครื่องสเลฟ จะใช้บอร์ด CP-32 โดยที่กำหนด NO.00 เป็นเครื่องมาสเตอร์ และ NO.01 ถึง NO.31 เป็นเครื่องสเลฟ ในลักษณะการติดต่อจาก มาสเตอร์ไปยังสเลฟแบบจุดต่อจุด (POINT TO POINT) หรือแบบจุดไปหลาย (POINT TO MULTIPOINT) และในลักษณะการติดต่อจากสเลฟไปยังมาสเตอร์ แบบจุดต่อจุด โดยการอินเตอร์รัพท์ (INTERRUPT) ลักษณะการส่งข้อมูลจะเป็นแบบ HALF DUPLEX โดยใช้มาตรฐาน RS - 485

RS - 485 COMMUNICATION

MR. PHAYAT	CHINVILAI	NO.	33 . 161215
MR. WINAI	SOASINUAL	NO.	33 . 161219
MR. ADUL	PHETSIRIPAN	NO.	33 . 161235
MR. ANUCHIT	SITTISINT	NO.	33 . 161236
MR. SUPAN	KULPANIT	ADVISER	

1991

ABSTRACT

THE PERPOSES OF THIS PROJECT CAN BE PROVIDED IN 3 SECTION WHICH ARE

1. LEARNING THEORY OF REMOTE COMMUNICATION BETWEEN GENERATOR AND RECEIVER BY WITHOUT " MODEM ".

2. THE DESIGN OF SOFTWARE TO CONTROL THE NETWORK OPERATING SYSTEM.

3. THE DESIGN OF EQUIPMENT THERE IS CAPABLE OF CONTROLLING BETWEEN MASTERAND SLAVE , WHICH IS " CP - 32 BOARD ".

THE " CP - 32 BOARD " NO.00 IS MASTER AND NO.01 - NO.31 IS SLAVE. THE COMMUNICATION FROM MASTER TO SLAVE WAS DONE IN " POINT TO POINT " OR " POINT TO MULTIPOINT " MANNER BY SOFTWARE INTERRUPT. THE DATA COMMUNICATION IS HALF DUPLEX STANDARD RS - 485.

สารบัญ

สารบัญรูปภาพ

บทที่ 1
บทนำ

ในปัจจุบัน การติดต่อสื่อสารถือว่าสำคัญมาก โดยมนุษย์ได้นำการส่งถ่ายข้อมูลมาใช้ในโรงงานอุตสาหกรรมอย่างกว้างขวาง โดยเริ่มพัฒนาการส่งและการรับข้อมูลให้ดีขึ้นเรื่อย ๆ ในยุคแรก ๆ นั้น การส่งและรับข้อมูล มีประสิทธิภาพต่ำ เพราะระบบการควบคุมที่มนุษย์ออกแบบในยุคแรก ๆ ไม่สลับซับซ้อน การส่งข้อมูลและรับข้อมูลคือ กระทำด้วยการไม่ค่อยจะคำนึงถึงเวลามากนัก แต่เมื่อมนุษย์มีการพัฒนากลไกในงานอุตสาหกรรมมากขึ้น ระบบของการส่งและรับข้อมูล ก็เริ่มจะสลับซับซ้อนมากยิ่งขึ้น เพื่อให้ทันกับความต้องการของมนุษย์ ฉะนั้นมนุษย์จึงได้คิดค้นวิธีการต่าง ๆ เพื่อที่จะนำมาซึ่งการส่งข้อมูลให้ถูกต้องรวดเร็วแม้การส่งข้อมูลนั้นจะอยู่ในระยะทางที่ไกล ๆ ก็ตาม ฉะนั้นระบบหรือมาตราฐานต่าง ๆ จึงได้เกิดขึ้นมากมายเพื่อจะสนองความต้องการของมนุษย์

ปริญญานิพนธ์ฉบับนี้ เช่นกัน ได้กล่าวถึงมาตราฐาน RS-485 ที่ใช้ในการติดต่อสื่อสารข้อมูลโดยส่งแบบ MULTIPOINT ซึ่งจะส่งข้อมูลจาก SINGLE BOARD TO SINGLE BOARD โดยการอินเทอร์รัพท์ (INTERUPT) โดยมีรายละเอียดแต่ละบทดังต่อไปนี้

บทที่ 1 บทนำ กล่าวถึงวัตถุประสงค์โดยย่อของโครงงานนี้ และรวมถึงการแนะนำเนื้อหาในบทต่าง ๆ

บทที่ 2 ทฤษฎีเบื้องต้นในการสื่อสารข้อมูลแบบอนุกรม กล่าวถึง การสื่อสารเบื้องต้นลักษณะการสื่อสารตามมาตราฐานต่าง ๆ รวมทั้งมาตราฐาน RS-485

บทที่ 3 การออกแบบอุปกรณ์ควบคุมและการควบคุม ด้วย SOFTWARE กล่าวถึงแนวทางในการออกแบบทั้งทาง HARDWARE และ SOFTWARE

บทที่ 4 FUNCTION ต่าง ๆ และการใช้งาน กล่าวถึง หน้าที่ของฟังก์ชั่นต่าง ๆ และการใช้งานการสื่อสารตามมาตราฐาน RS-485

บทที่ 5 หาสรุปและข้อเสนอแนะ กล่าวถึงคุณสมบัติของโครงงานที่จัดทำขึ้นมาและข้อคิดต่าง ๆ ในการพัฒนาให้ดียิ่งขึ้น

ภาคผนวกที่ 1 แสดง PROGRAM MONITOR ที่ใช้ควบคุมการติดต่อสื่อสารระหว่าง MASTER และ SLAVE โดยใช้ PROGRAM ชุดเดียวกัน

ภาคผนวกที่ 2 แสดง DATA SHEET ของอุปกรณ์ที่ใช้ในโครงการนี้

บทที่ 2
ทฤษฎีเบื้องต้นในการสื่อสารข้อมูลแบบอนุกรม

การสื่อสารข้อมูลแบบอนุกรม

ถ้าขึ้นหัวข้อว่าการสื่อสารข้อมูลแบบอนุกรมแล้ว แสดงว่าจะต้องมีการสื่อสารข้อมูลแบบขนานด้วย การสื่อสารแบบขนานก็คือข้อมูลทุก ๆ บิตในแต่ละเวิร์ดจะถูกส่งออกไปพร้อม ๆ กัน ขึ้นอยู่กับว่าเวิร์ดดังกล่าวมีขนาดเท่าไหร่ โดยทั่วไปก็คือ 1 ไบต์หรือ 8 บิตนั่นเองการส่งข้อมูลแบบขนานนี้จะมีข้อจำกัดทางด้านระยะทางระหว่างต้นทางและปลายทาง โดยทั่วไปจะส่งได้ในระยะไม่เกิน 3-5 ฟุตเท่านั้น ทั้งนี้ขึ้นอยู่กับอัตราที่ใช้ในการส่งข้อมูลสูงก็จะส่งได้ระยะสั้นลงการส่งข้อมูลแบบขนานนั้นนิยมในระบบที่ต้องการความเร็วสูงมาก ๆ แต่อุปกรณ์อยู่ไม่ห่างกันมากนัก

ส่วนการส่งข้อมูลแบบอนุกรมนั้น ข้อมูลจะถูกทยอยส่งออกไปทีละบิตจนครบทั้งเวิร์ดในสายสัญญาณเพียงเส้นเดียว แต่ในการใช้งานจริงจะต้องมีสายสัญญาณอีกเส้นเป็นระดับ Ground ดังนั้นเมื่อเราส่งข้อมูลในแบบอนุกรมเราจะสามารถใช้สายสัญญาณอย่างน้อยที่สุดเพียง 2 เส้น ในขณะที่การส่งข้อมูลแบบขนานจะต้องใช้อย่างน้อยเท่ากับจำนวนบิตบวกกับสายสัญญาณระดับแรงดัน Ground อีก 1 เส้น นอกจากนี้การส่งข้อมูลแบบอนุกรมนั้น จะสามารถส่งข้อมูลได้ไกลกว่ามาก เช่น ถ้าส่งตามมาตรฐานของ RS-232 ที่จะกล่าวต่อไปในภายหลังจะสามารถส่งได้ไกลถึง 30 ถึง 40 ฟุตโดยไม่ต้องใช้อุปกรณ์ขับสัญญาณเพิ่มเติมแต่อย่างใดอย่างไรก็ดีการส่งข้อมูลจากข้อมูลแบบอนุกรมนั้น จะต้องมีส่วนที่ทำหน้าที่แปลงข้อมูลจากข้อมูลแบบขนานมาเป็นแบบอนุกรม ซึ่งสามารถเป็นได้ทั้งฮาร์ดแวร์และซอฟต์แวร์ และในการส่งยังมีข้อกำหนดบางประการเพื่อให้การส่งข้อมูลมีความถูกต้องมากยิ่งขึ้นอีกด้วย

SIMPLEX และ DUPLEX

ในการสื่อสารไม่ว่าจะเป็นการสื่อสารข้อมูลหรือการสื่อสารทั่วไปนั้น ย่อมจะต้องประกอบด้วยผู้รับและผู้ส่ง ผู้รับในขณะนี้อาจสามารถเป็นผู้ส่งในอนาคตได้ แต่มีบางกรณีสำหรับการสื่อสารข้อมูลที่ผู้รับและผู้ส่งจะตายตัวอยู่ตลอดเวลา เช่นการสื่อสารข้อมูลระหว่างเครื่องคอมพิวเตอร์กับเครื่องพิมพ์ เป็นต้น การสื่อสารของอุปกรณ์ที่มีผู้รับและผู้ส่งตายตัวนั้น เราเรียกว่าการสื่อสารแบบ Simplex กล่าวคือ การสื่อสารเป็นไปในลักษณะทิศทางเดียวตลอดเวลา ซึ่งจะมีที่ใช้น้อยมาก การสื่อสารโดยทั่วไปนั้นจะเป็นลักษณะ Duplex คือมีทิศทางการสื่อสาร 2 ทิศทางทั้งไปและกลับ การสื่อสารในลักษณะ Duplex คือมีทิศทางการสื่อสาร 2 ทิศทางทั้งไปและกลับ การสื่อสารในลักษณะ Duplex นี้ยังแบ่งออกได้เป็น 2 ชนิด คือ HalfDuplex (นิยมเขียนย่อว่า HDX) ซึ่งจะมีทิศทางการสื่อสารในลักษณะที่ผลัดกันเป็นผู้ส่งและผู้รับในเวลา

เดียวกันเราอาจเปรียบเทียบเพื่อให้เห็นภาพพจน์ชัดเจนยิ่งขึ้น โดยเปรียบเทียบการสื่อสารแบบ HDX หรือ Half Duplex เป็นทางรถไฟ ซึ่งจะมีรถไฟเพียงขบวนเดียวเท่านั้นที่วิ่งอยู่บนรางในเวลาหนึ่ง และเปรียบเทียบ FDX หรือ Full Duplex เป็นถนนที่รถสามารถวิ่งสวนกันได้ในเวลาเดียวกัน การสื่อสารระหว่างคอมพิวเตอร์มักจะอยู่ในลักษณะของ Duplex แบบใดแบบหนึ่ง

3 โปรโตคอลของการสื่อสารแบบอนุกรม

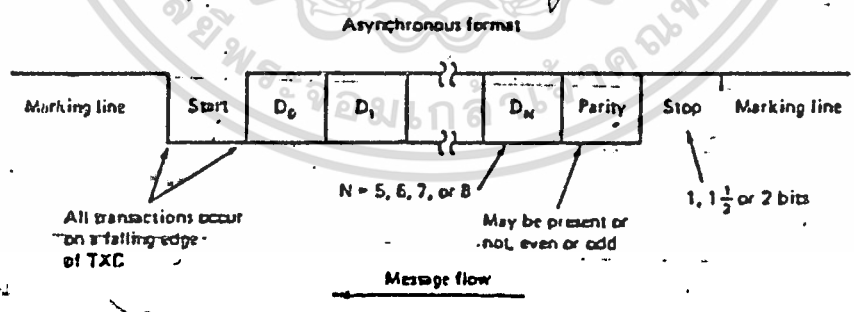
เมื่อพิจารณาการส่งข้อมูลในแบบอนุกรมให้ดีจะพบว่า ปัญหาหนึ่งที่ควรจะเกิดขึ้นอยู่เสมอก็คือ การตัดสินว่าข้อมูลที่ได้รับนั้นมีจุดเริ่มต้นที่ใด ดังนั้นจึงมีการกำหนดข้อตกลงในการสื่อสารขึ้นเพื่อแก้ปัญหานี้ ข้อตกลงดังกล่าวเราเรียกว่า โปรโตคอล (Protocol) ของการสื่อสารข้อมูลแบบอนุกรม สามารถแบ่งออกเป็น 2 ประเภทใหญ่ ๆ คือ โปรโตคอลสำหรับการสื่อสารข้อมูลอนุกรมแบบซิงโครนัส (Symchronous) และโปรโตคอลสำหรับการสื่อสารข้อมูลแบบอซิงโครนัส (Asynchronous) การสื่อสารแบบซิงโครนัสนั้น ข้อมูลจะถูกส่งออกไปอย่างสม่ำเสมอ ช่วงเวลาระหว่างบิตและระหว่างเวิร์ดจะมีค่าเท่ากันเสมอดังนั้นในการสื่อสารข้อมูลอนุกรมในแบบซิงโครนัสจึงต้องมีสายสัญญาณเพิ่มเติมเพื่อกำกับการส่งว่าควรจะส่งเมื่อใดและควรจะหยุดเมื่อใดระบบที่เป็นซิงโครนัสจะเป็นระบบที่มีความเร็วสูงแต่ก็ยังต่ำกว่าการสื่อสารแบบขนาน

ที่นี้เราจะมาดูการสื่อสารแบบอซิงโครนัสกันบ้าง การสื่อสารแบบอซิงโครนัสนี้ เป็นหัวใจของการสื่อสารข้อมูลผ่านทางสายโทรศัพท์ในปัจจุบัน การสื่อสารแบบนี้ช่วงระยะเวลาระหว่างบิตจะมีค่าเท่ากันเช่นเดียวกับซิงโครนัส แต่จะแตกต่างกันที่ระยะห่างระหว่างเวิร์ด ซึ่งไม่มีข้อกำหนดว่าจะห่างกันเป็นระยะเวลานานเท่าใด กล่าวคือ ระหว่างเวิร์ดนั้น จะห่างกันกี่วินาที กี่นาที กี่ชั่วโมง กี่วัน กี่เดือนหรือ กี่ปีก็ได้ทั้งสิ้นขึ้นอยู่กับว่าทางฝ่ายรับสามารถรอได้หรือไม่เท่านั้น

เมื่อไม่มีข้อกำหนดทางด้านระยะเวลาระหว่างเวิร์ดแล้ว ทางผู้ส่งและผู้รับจะเข้าใจตรงกันได้อย่างไรว่าที่ใดคือจุดเริ่มต้นและจุดสิ้นสุดของแต่ละเวิร์ด เพื่อแก้ปัญหานี้ จึงมีการกำหนดข้อตกลงเกี่ยวกับฟอร์แมตของข้อมูลที่จะส่งให้ทางผู้รับสามารถเข้าใจว่าจุดใดเป็นจุดเริ่มต้นของเวิร์ด ข้อกำหนดดังกล่าวกำหนดให้แต่ละเวิร์ดจะต้องขึ้นต้นด้วยบิตที่เรียกว่า Start Bit หรือ บิตเริ่มต้น ซึ่งจะต้องเป็นลอจิกศูนย์เสมอจากนั้นตามด้วยบิตข้อมูลที่ต้องการส่ง ซึ่งมีความยาว 5 ถึง 8 บิต ถัดจากบิตข้อมูลก็จะเป็นพาริตีบิตซึ่งทำหน้าที่เป็นบิตสำหรับตรวจสอบความถูกต้องของข้อมูลที่ได้รับว่ามีความถูกต้องหรือไม่ บิตพาริตีนี้มี 2 ประเภท คือ Even Parity ซึ่งจะเซ็ต (มีค่าเป็น 1) เมื่อจำนวนบิตที่เป็นลอจิก 1 ในบิตที่เป็นข้อมูลมีจำนวนเป็นคี่และ Odd Parity ซึ่งจะเซ็ตเมื่อจำนวนบิตที่เป็นลอจิก 1 ในบิตที่เป็นข้อมูลมีจำนวนเป็นคู่ ในการส่งข้อมูลบางครั้งอาจจะไม่มีการใช้บิตพาริตีก็ได้ ถ้าหากการสื่อสารในครั้งนั้นมีความน่าเชื่อถือสูง คือมี

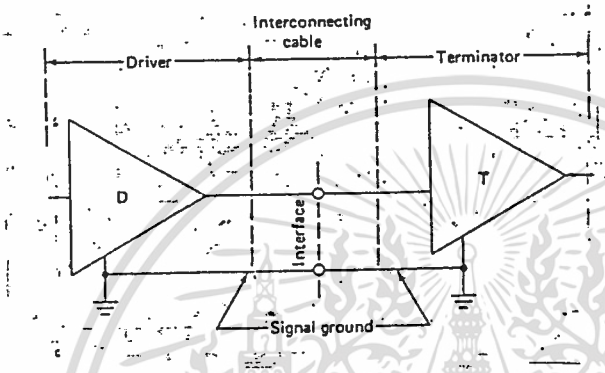
สัญญาณรบกวนต่ำเป็นการเพิ่มความเร็วในการสื่อสารได้ด้วย บิตสุดท้ายในฟอร์แมตก็คือ Stop Bit ทำหน้าที่บอกทางผู้รับว่าขณะนี้ข้อมูลที่ทางผู้รับได้รับนั้นครบเวิร์ดแล้วขอให้เตรียมตัวรับเวิร์ดต่อไปได้ Stop Bit นี้ถูกกำหนดให้เป็นลอจิก 1 เสมอทั้งนี้เพื่อให้ระบบสามารถตรวจสอบบิตเริ่มต้นได้ (บิตเริ่มต้นมีลอจิกเป็น 0) Stop Bit นี้อาจมีความยาวเป็น 1 บิต 1.5 บิต หรือ 2 บิต ก็ได้

จากฟอร์แมตดังกล่าว จะเห็นว่าเรามีฟอร์แมตสำหรับการสื่อสารมากมายไปหมด เช่น 5E1 (5Data bit, Even Parity, 1 Stop Bit), 7E1 (7Data bit, Even Parity, 1 stop Bit) และ 8N1 (8Data bit, No Parity, 1 Stop Bit) เป็นต้น ในการใช้งานทั่วไป เรานิยมใช้กันอยู่เพียง 2 ฟอร์แมต คือ 7E1 และ 8N1 จะเลือกใช้ฟอร์แมตใดขึ้นอยู่กับสภาพของสายส่งสัญญาณว่ามีสัญญาณรบกวนมากเพียงใด ถ้าหากสายส่งมีสัญญาณรบกวนมากก็ควรจะใช้ 7E1 แต่ถ้าสายส่งสัญญาณมีสภาพดีสัญญาณรบกวนต่ำการใช้ 8N1 จะเลือกใช้ฟอร์แมตใดขึ้นอยู่กับสภาพของสายส่งสัญญาณว่ามีสัญญาณรบกวนมากเพียงใด ถ้าหากสายส่งสัญญาณมีสภาพดีสัญญาณรบกวนต่ำการใช้ 8N1 จะเร็วกว่าเป็นต้น ทั้งนี้จะต้องมีการตกลงกันล่วงหน้าระหว่างผู้รับและผู้ส่งว่าจะใช้ฟอร์แมตใดในการสื่อสาร ลักษณะของข้อมูลที่ถูกส่งออกไปจะมีลักษณะดังรูป



มาตราฐานการสื่อสารข้อมูลแบบอนุกรมที่กำหนดโดย EIA (Electronics Industries Association) มาตราฐาน RS-232-C

มาตราฐาน RS-232-C ได้ถูกตีพิมพ์โดย EIA ในปี ค.ศ. 1969 ตัวอักษร RS แทน "Recomm Standard" 232 แทนหมายเลขของมาตราฐาน ส่วนอักษร C แสดงให้รู้ว่ามาตราฐานได้รับการแก้ไขกี่ครั้ง



รูปที่ 2.1 ลักษณะสมบัติทางไฟฟ้าของการอินเทอร์เฟสแบบ RS-232-C

- ถูกออกแบบให้ใช้กับอุปกรณ์พวก discrete
- ใช้การอินเทอร์เฟสแบบ Unbalanced
- ในแต่ละเซอร์กิตใช้ลวดนำในการนำสัญญาณ 1 เส้น และมีสายกราวนด์รวมของทุกเซอร์กิตเพียงเส้นเดียว
- อัตราเร็วในการส่งข้อมูลมีค่า < 20 kbps
- ระยะทางสูงสุดที่ใช้ในการส่งข้อมูลมีค่า < 15 m
- ทำให้เกิด crosstalk ที่มีค่ามาก

ตามมาตราฐาน RS-232-C ที่ถูกตีพิมพ์โดย EIA ได้กล่าวถึงการสื่อสารข้อมูลระหว่าง Data Terminal Equipment (DTE) และ Data Communication Equipment (DCE) (แต่ในปัจจุบันตัวย่อ DCE จะแทน data circuit termination equipment) คำจำกัดความของ DCE และ DTE ซึ่งแสดงไว้ข้างล่างได้คัดมาจากคำแปลศัพท์ (glossary) ในหนังสือ "Technical Aspect of Data communication" ซึ่งเขียนโดย John Mcnamara (Digital press, 1977) ดังนี้

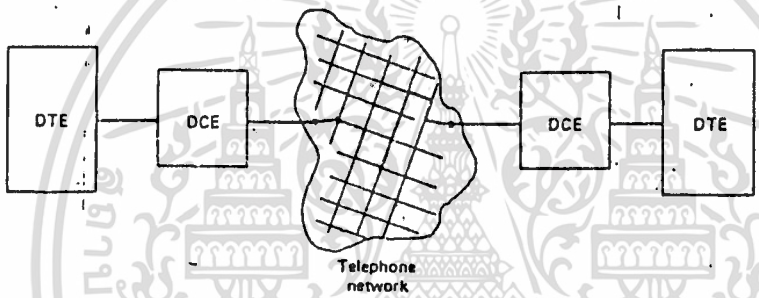
DCE : อุปกรณ์ที่มีฟังก์ชันการทำงานต่าง ที่ทำให้เกิดการเชื่อมต่อ ทำให้การเชื่อมต่อยังดำรงต่อไป และยุติการเชื่อมต่อ นอกจากนี้ยังใช้เปลี่ยนลักษณะของสัญญาณและสร้างรหัสสัญญาณต่าง ๆ ที่จำเป็นต้องใช้ในการสื่อสารข้อมูลระหว่าง DTE และ Data circuit โดย DCE อาจเป็นส่วนใดส่วนหนึ่งของคอมพิวเตอร์หรือไม่ก็ได้

DTE :

1. เป็นอุปกรณ์ที่ประกอบไปด้วยตัวส่งข้อมูล (data source) หรือตัวรับข้อมูล (data sink) หรือเป็นทั้งตัวส่งและตัวรับข้อมูลก็ได้

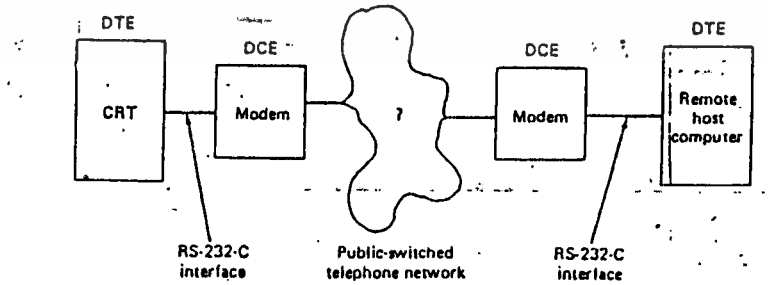
2. เป็นอุปกรณ์ที่ประกอบด้วย function unit ต่อไปนี้ control logic, buffer store และอุปกรณ์อินพุทหรือเอาท์พุทจำนวนหนึ่งตัวหรือมากกว่าก็ได้ หรือรวมเครื่องคอมพิวเตอร์เข้าไปด้วยก็ได้ DTE อาจจะรวมส่วน error control , synchronization และความสามารถในการบ่งหรือระบุว่าต้องการเกี่ยวข้องกับอุปกรณ์ตัวใด (station identification capability) เข้าไปด้วยก็ได้

ลักษณะของ DCE และ DTE ที่ใช้ในการสื่อสารข้อมูลได้แสดงไว้ในรูปที่ 2.2 ตามลักษณะการทำงานที่ได้อธิบายไว้ข้างต้น



รูปที่ 2.2 ลักษณะของ DCE และ DTE ที่ใช้ในวงจรการสื่อข้อมูล

ในความเป็นจริงแล้ว DTE มักจะแทนแหล่งกำเนิดข้อมูลแหล่งแรกและ/หรืออุปกรณ์ที่เป็นแหล่งรับข้อมูลแหล่งสุดท้าย (ดังแสดงในรูปที่ 2.2) เช่นเครื่องพิมพ์ (เป็นอุปกรณ์ที่รับข้อมูลได้เพียงอย่างเดียว) จะเป็น DTE เพราะเป็นอุปกรณ์ที่รับข้อมูลเป็นตัวสุดท้าย หรือ CRT/ คีย์บอร์ดเป็นทั้งตัวรับข้อมูลและตัวกำเนิดข้อมูล ส่วน DCE เป็นอุปกรณ์ที่ทำให้การสื่อสารข้อมูลระหว่างแหล่งกำเนิดกับตัวรับข้อมูลที่ปลายทางทำได้สะดวกขึ้น ตัวอย่างหนึ่งของ DCE ก็คือโมเด็ม



รูปที่ 2.3 ลักษณะของระบบที่ใช้แสดงเป็นตัวอย่าง

ถ้าเทอร์มินัลและคอมพิวเตอร์ของเราเป็น DCE และ DTE ทั้งคู่เราจะทำการสื่อสารข้อมูลได้อย่างไร ปัญหาที่เกิดขึ้นนี้เราแก้ไขได้โดยใช้เคเบิล (Cable) ที่เรียกว่า "null moderm"

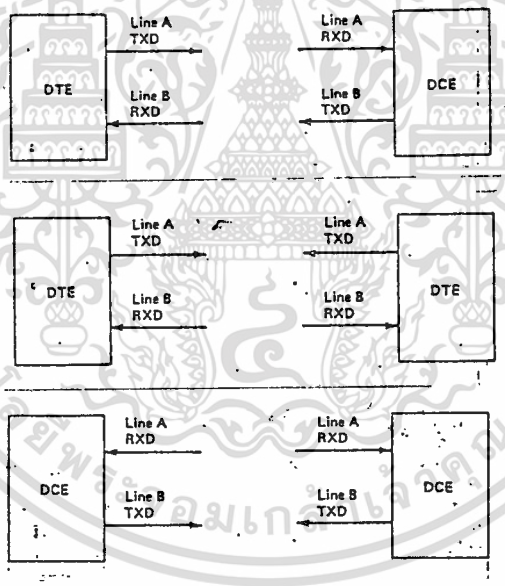
1. null หมายความว่า อุปกรณ์ตัวนี้ไม่สามารถทำงานอะไรได้เราใช้อุปกรณ์ตัวนี้เมื่อต้องการเปลี่ยนทิศทางการเคลื่อนที่ของข้อมูลเท่านั้น

2. modem แสดงว่าอุปกรณ์ตัวนี้เป็น DCE ด้วยเหตุนี้เราจึงใช้ null modrm แทรกเข้าไประหว่าง DTE สองตัว เพื่อทำให้เราสามารถทำการสื่อสารข้อมูลโดยผ่าน RS-232-C ได้

สำหรับลักษณะของสาย (line) ที่ใช้ในการรับและส่งข้อมูลของ DTE และ DCE เป็นดังนี้

1. สายที่ใช้ในการส่งและรับข้อมูลของ DTE และ DCE มีอยู่สองเส้นแต่ละเส้นจะมีทิศทางการเคลื่อนที่ของข้อมูลกำหนดได้ต่างทิศทางกัน

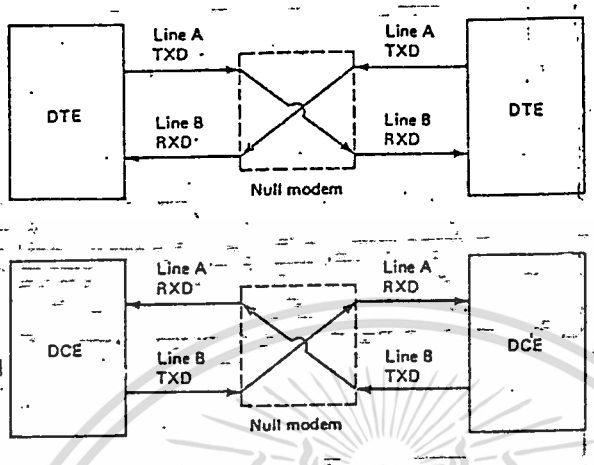
2. DTE จะส่งข้อมูลทาง line A และ DCE จะรับข้อมูลทาง line A เช่นเดียวกัน DCE จะส่งข้อมูลทาง line B และ DTE รับข้อมูลทาง line B ดังแสดงในรูปที่ 2.4



รูปที่ 2.4 ลักษณะของการส่งและรับข้อมูลของ DCE และ DTE

จากรูปที่ 4 ถ้า DTE 2 ตัว ทำการแลกเปลี่ยนข้อมูลกัน ข้อมูลจะถูกส่งออกทาง line A และรับทาง line B ทั้งคู่ ดังนั้นการสื่อสารข้อมูลจะไม่สามารถเกิดขึ้นได้สำหรับกรณีที่ DCE 2 ตัว ทำการแลกเปลี่ยนข้อมูลก็เป็นเช่นเดียวกัน เมื่อเราต่อ line A เข้าด้วยกัน และต่อ line B เข้าด้วยกัน ข้อมูลใน line B จะต้านกันเอง ส่วนใน line A ไม่มีข้อมูลที่จะรับ ปัญหานี้แก้ได้โดยใช้ null modem เข้าช่วย null modem cable จะทำการไขว้ line A เข้ากับ line B ดังรูปที่25 ดังนั้น DTE 1 สามารถรับ

ข้อมูลที่ส่งจาก DTE 2 ได้และ DTE 2 ก็สามารถรับข้อมูลจาก DTE 1 ได้เช่นกัน



รูปที่ 2.5 ลักษณะการทำงานของ null modrm ที่ใช้ในการอินเทอร์เฟสระหว่าง DTEหรือDCE สองตัว

เราใช้มาตราฐาน RS-232-C ในการสื่อสารข้อมูลแบบอนุกรมระหว่าง DCE กับ DTE โดยอัตราการส่งข้อมูลจะถูกกำหนดให้อยู่ระหว่าง 0 ถึง 2000 บิทต่อวินาที ในการประยุกต์ใช้งาน RS-232-C อัตราเร็วสูงสุดที่ใช้ควรจะมีค่าไม่เกิน 19.2 Kbps

มาตราฐานนี้ได้กำหนดความยาวของสายเคเบิลที่ใช้ในการสื่อสารข้อมูลไว้ไม่เกิน 50 ฟุต (ไม่ใช่ข้อกำหนดที่ตายตัว เนื่องจากระยะ 50 ฟุต นี้ได้มาจากประสบการณ์) เคเบิลอาจจะยาวกว่า 50 ฟุตก็ได้ เรารู้สภาพแวดล้อมของสายเคเบิลและอยู่ในเงื่อนไขที่ถูกกำหนดไว้ในมาตราฐาน

EIA ได้ระบุไว้ว่า เราไม่ควรใช้ RS-232-C ในกรณีที่ต้องการให้มีการแบ่งแยกทางไฟฟ้า (electrical isolation) ระหว่างอุปกรณ์ทั้งสองด้านของการอินเทอร์เฟสซึ่งคำเตือนนี้เป็นสิ่งสำคัญที่เราต้องจำไว้เสมอถ้าเราคิดจะใช้ RS-232-C ในการอินเทอร์เฟสเครื่องคอมพิวเตอร์ของเราเข้ากับอุปกรณ์ที่สร้างขึ้นเอง

วัตถุประสงค์ของการใช้ RS-232-C คือใช้ในการอินเทอร์เฟสระหว่าง DTE กับ DCE ในกรณีที่ต้องการสื่อสารข้อมูลในระยะทางไกล ๆ โดยผ่านทางเครือข่ายโทรศัพท์ แต่อย่างไรก็ตาม RS-232-C ก็ยังใช้ในการสื่อสารข้อมูลในระยะทางใกล้ ๆ เช่น ใช้ระหว่างคอมพิวเตอร์กับเทอร์มินัล คอมพิวเตอร์กับเครื่องพิมพ์และคอมพิวเตอร์กับดิสเก็ตไดรฟ์

คุณสมบัติของสัญญาณไฟฟ้า

สัญญาณที่ขาทุกขาที่คอนเน็คเตอร์ของ RS-232-C จะเป็นสภาวะใดสภาวะหนึ่งในแต่ละคู่ของคู่ต่อไปนี้

MARK/SPACE

ON/OFF

LOGIC 0/LOGIC 1

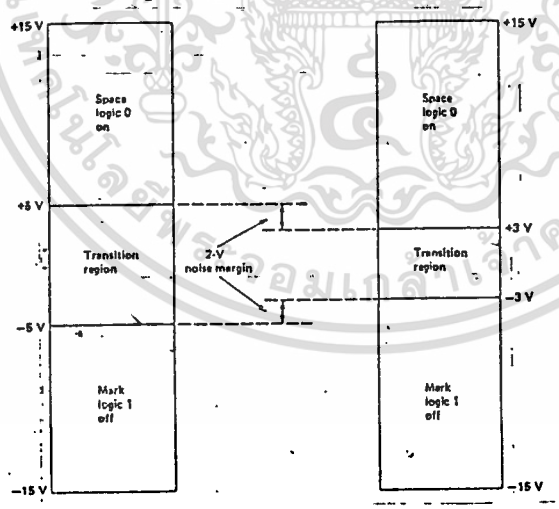
ความสัมพันธ์ระหว่างสถานะของสัญญาณคู่ต่าง ๆ กับระดับแรงดันได้แสดงไว้ในตารางที่ 1

Status	Signal Voltage	
	$-25V < V_1 < -3V$	$3V < V_1 < 25V$
Binary logic state	1	0
Signal condition	MARK	SPACE
Function	OFF	ON

ตารางที่ 1

ช่องของระดับแรงดันระหว่าง -3 ถึง +3 โวลต์ จะเป็นช่วงของการเปลี่ยนลอจิก ดังนั้นจึงไม่มีการระบุสภาวะของสัญญาณในช่วงนี้ ในการแทนลอจิก 1 หรือสถานะ mark ตัวขับสัญญาณ (driver) ต้องจ่ายแรงดันระหว่าง -5 ถึง -15 โวลต์ ส่วนในการแทนลอจิก 0 หรือ space ตัวขับสัญญาณต้องจ่ายแรงดันระหว่าง +5 ถึง +15 โวลต์

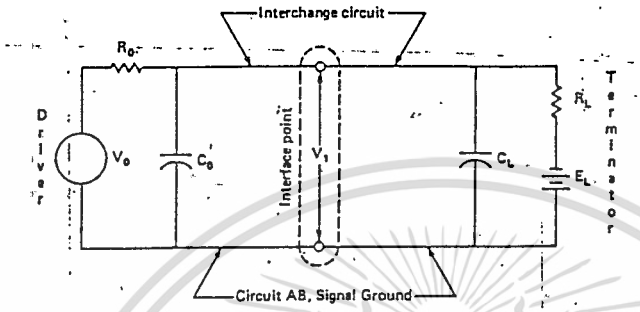
RS-232-C ยอมให้มี noise margin ได้ไม่เกิน 2 โวลต์ สำหรับความสัมพันธ์ระหว่างระดับแรงดันและสถานะของสัญญาณได้แสดงไว้ในรูปที่ 2.6



รูปที่ 2.6 คุณสมบัติทางไฟฟ้าของการอินเทอร์เฟสแบบ RS-232-C

จากรูปจะเห็นได้ว่า ถ้า line driver หรือตัวกำเนิดสัญญาณต้องการส่งลอจิก 0 line driver จะต้องจ่ายแรงดันระหว่าง +5 ถึง +15 โวลต์ ส่วน line receiver หรือตัวรับสัญญาณปลายทางจะถือว่า

แรงดันที่อยู่ภายในช่วง +3 ถึง +5 โวลต์ แทนลอจิก 0 จากการเปรียบเทียบระดับสัญญาณของตัวส่งและตัวรับจะเห็นว่า RS-232-C ยอมให้มีการ drop ของสัญญาณในช่วง 2 โวลต์เกิดขึ้นได้สำหรับในด้านการส่งลอจิก 1 ก็เป็นเช่นเดียวกัน



รูปที่ 2.7 RS-232-C Interface Circuit (EIA)

จากที่ได้อธิบายมาอาจมีข้อสงสัยว่า ทำไมไม่ใช้สถานะลอจิกแบบ TTL ซึ่งระดับของแรงดันมีค่าระหว่าง 0 ถึง +5 โวลต์และทำไมถึงต้องใช้ระดับแรงดันระหว่าง -15 ถึง -3 และ +3 ถึง +15 โวลต์ด้วย

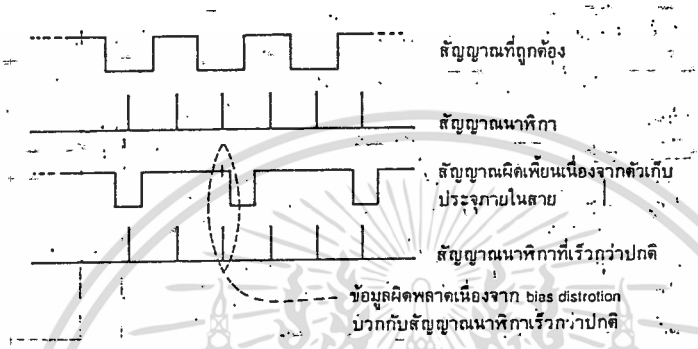
สาเหตุที่ไม่ใช้การแทนลอจิกแบบ TTL ก็เพราะสถานะลอจิกแบบ TTL ถูกรบกวนจากสัญญาณรบกวนต่าง ๆ ได้ง่าย นอกจากนั้นยังมีปัญหาเกี่ยวกับระยะทางที่สามารถทำการสื่อสารข้อมูลอีกด้วย สำหรับสาเหตุที่ต้องใช้แรงดันในช่วง -15 ถึง -3 และ +3 ถึง +15 ก็เพราะในขณะที่กำลังทำการพัฒนา RS-232-C ขึ้นนั้นในวงจรคอมพิวเตอร์ต่าง ๆ โดยทั่ว ๆ ไปมีการใช้ระดับแรงดันในช่วงเหล่านี้อยู่ อนึ่งทรานซิสเตอร์ที่มีขายกันทั่วไปสามารถทำงานได้ในช่วงแรงดันเหล่านี้ และยังทนต่อสัญญาณรบกวนต่าง ๆ ที่มีเข้ามาได้ นอกจากนี้ยังสามารถทำงานที่ความถี่สูง ๆ ได้ ถึง 20000 บิทต่อวินาที (bps) ยิ่งกว่านั้น MARK และสถานะ SPACE ยังถูกแทนด้วยการไหลของกระแสในทิศทางที่ตรงข้ามกัน และความแตกต่างของแรงดันที่สถานะ MARK และ SPACE มีค่าสูงถึง 6 โวลต์เป็นอย่างน้อย ข้อดีต่าง ๆ ที่กล่าวมานี้ช่วยให้การส่งข้อมูลมีเสถียรภาพ

จากรูปที่ 2.7 นั้น ตัวเก็บประจุ C1 ที่ต่อขนานกับอุปกรณ์รับข้อมูลปลายทางจะต้องมีค่าไม่เกิน 2500 pf โดยค่านี้ไม่รวมค่าความจุไฟฟ้าของสายเคเบิลเข้าไปด้วย (ด้วยเหตุนี้จึงทำให้ระยะทางที่สามารถใช้ทำการสื่อสารข้อมูลได้ต้องไม่เกิน 50 ฟุต)

ปัญหาของ RS-232-C พอสรุปได้ 3 ประการ

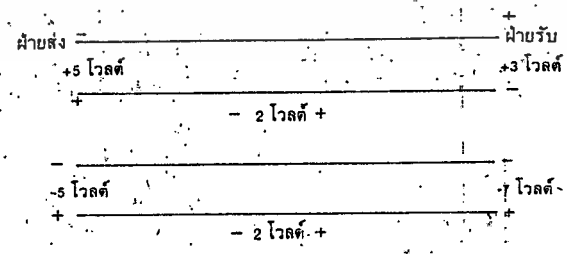
1. ใช้ระดับแรงดันไฟเลี้ยง + 15 V นอกเหนือจาก 5 โวลต์ ซึ่งใช้ในวงลอจิก
2. ค่าตัวเก็บประจุของอุปกรณ์รับสัญญาณ RS-232-C รวมทั้งตัวเก็บประจุสเตย์ (Stay Capacitance) สายจะต้องไม่มากกว่า 2500 pf สายที่รวมกันหลาย ๆ สาย ส่วนมากจะมีตัวเก็บประจุสเตย์ ประมาณ 40 - 50 pf ต่อ 1 ฟุต ดังนั้นสายนี้จะต่อได้ยาวสุด 50 ฟุต ก่อนที่ค่าตัวเก็บประจุสเตย์จะมากกว่า

2500 pf ถ้าหากตัวเก็บประจุสเตย์มากกว่าที่กำหนดนี้ ช่วงเวลาการเปลี่ยนแปลงระดับของสัญญาณมากกว่า 4 เปอร์เซนต์ตามที่ยอมให้ได้ในมาตราฐาน RS-232-C เมื่อเป็นเช่นนี้จะทำให้ฝ่ายรับตีความสัญญาณผิดไปจากความเป็นจริง มาร์กบิต (MARK bit) จะยาวกว่าสเปซบิต (SPACE bit) หรือสเปซยาวกว่ามาร์กบิตขึ้นอยู่กับวงจรการตรวจสอบการผิดเพี้ยน แบบนี้เรียกว่า "Bias distrotion"



รูปที่ 2.8 แสดงถึงการตีความข้อมูลที่ผิดของฝ่ายรับอันเนื่องจาก bias distrotion และสัญญาณนาฬิกาเร็วกว่าปกติ

3. ปัญหาทางสัญญาณไฟฟ้าก็คือปัญหาเรื่อง ground ที่แตกต่างกัน สัญญาณที่ส่งออกเทียบกับ ground ของตัวส่งเท่านั้น ถ้าหากตัวรับกับตัวส่งมีระดับแรงดัน ground ที่แตกต่างกัน สมมุติว่า 2 โวลต์ กระแสที่ไหลในเส้นที่เป็น ground ก็จะเกิดขึ้น สมมุติว่าความต้านทานของสายเป็น 0 ความต่างศักย์ที่เกิดจากกระแส ground ก็จะไม่มี ความต่างศักย์ของ ground ระหว่างตัวรับกับตัวส่งก็จะคงเท่าใดมีระดับของสัญญาณที่ฝ่ายส่งและฝ่ายรับ มองเห็นก็จะแตกต่างกัน สมมุติว่าระดับแรงดัน ground ต่างกัน 2 โวลต์ ตัวส่งป้อนแรงดันเข้าไป 5 โวลต์ ตัวรับก็จะมองเห็นแค่ 3 โวลต์เท่านั้นในทางกลับกัน ถ้าฝ่ายส่งป้อนแรงดัน -5 โวลต์จะมองเห็นเป็น -7V



รูปที่ 2.9 ผลของระดับสัญญาณ ground ที่แตกต่างกัน

ความต่างศักย์ของ ground จะคงที่ 2 โวลต์ไม่ว่าฝ่ายส่งจะใส่แรงดันเข้าไปเท่าไหร่ก็ตามผลของ ground ที่แตกต่างกันนี้อาจจะเกิดมาจากตัวรับและตัวส่งมีระบบไฟฟ้าที่ ground แตกต่างกันก็ได้

จากปัญหาต่าง ๆ ใน RS-232-C ทาง EIA จึงได้พัฒนาและกำหนดมาตราฐานใหม่ขึ้นอีกเป็น RS-422 RS-423 และ RS-485

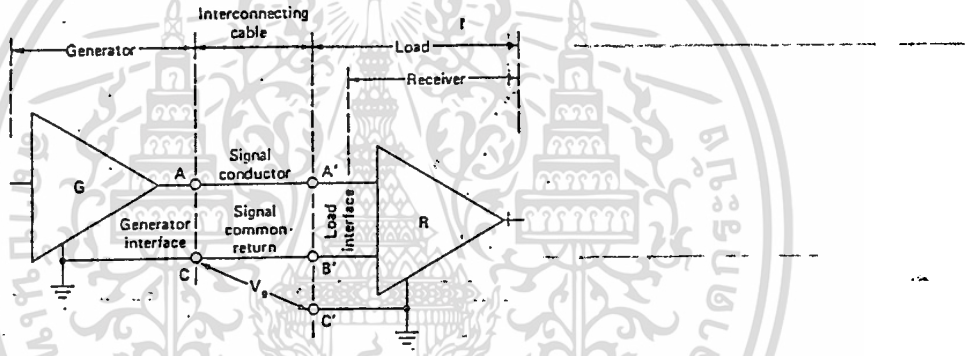
RS-422, RS-423 และ RS-485 มีข้อได้เปรียบ RS-232-C ตรงที่อินพุทของพวกมันเป็นแบบ" การขยายความแตกต่าง (differential input)" ต่อไปจะอธิบายว่ามันได้เปรียบอย่างไร ขอให้พิจารณาสภาพต่อไปนี้ : เราวางตำแหน่งของDCE และ DTE ของเราไว้ในสำนักงานเดียวกันแต่ใช้ระบบจ่ายไฟคนละระบบ จากสาเหตุนี้เองทำให้ระดับแรงดันที่สาย ground ของ DCE และ DTE มีระดับแรงดันแตกต่างกันโดยสมมุติให้ระดับแรงดันที่ ground ของ DTE มีค่ามากกว่า DCE เท่ากับ 5 โวลต์ จากมาตราฐาน RS-232-C ถ้าเราต้องการส่งลอจิก 1 เราต้องป้อนแรงดันในช่วง -5 ถึง -15 โวลต์เทียบกับ ground สัญญาณ(signal ground) ของตัวส่ง ดังนั้นถ้า DTE)ป้อนระดับแรงดันในช่วง -5 ถึง-7.9 โวลต์ เทียบกับ ground ของมัน(ของ DTE) เมื่อ DCE รับระดับแรงดันนี้เข้ามาโดยเทียบกับ ground ของมัน(ของDCE)ระดับแรงดันที่รับเข้ามาจะอยู่ในช่วง 0 ถึง -2.9 โวลต์แทน ซึ่งจากเหตุการณ์เช่นนี้ DTE จะคิดว่ามันส่งข้อมูลที่เป็นลอจิก 1 ออกมา แต่จริง ๆ กลายเป็นว่า DCE รับข้อมูลอยู่ในช่วงที่ไม่สามารถตัดสินได้ว่าเป็นลอจิกใดไปแทน(อยู่ใน transition region) ดังนั้นความแตกต่างของระดับแรงดันที่ groundสัญญาณของ DTE และ DCE ซึ่งมีค่าเท่ากับ 5 โวลต์เป็นตัวก่อให้เกิดความผิดพลาดข้อนี้ขึ้น

ตัวการอีกอย่างหนึ่ง ที่ทำให้ระดับแรงดันที่ส่งมีค่าเปลี่ยนแปลงไป คือ สัญญาณรบกวนทางไฟฟ้า (electrical noise) ถ้าเราวางเคเบิล RS-232-C ผ่านสนามแม่เหล็กไฟฟ้าที่แรงพอ สนามแม่เหล็กไฟฟ้านี้สามารถเหนี่ยวนำให้เกิดสัญญาณรบกวนที่มีค่ามากพอ ที่จะเปลี่ยนแปลงระดับลอจิกที่ส่งไปจากลอจิก 0 ไปเป็น 1 หรือลอจิก 1 ไปเป็น 0 ได้

เราสามารถลดปัญหาต่าง ๆ ที่กล่าวไว้ข้างต้นลงได้โดยใช้ตัวรับข้อมูลที่มีอินพุทเป็น แบบวงจรขยายความแตกต่าง (differential input) คุณสมบัติที่สำคัญของตัวรับข้อมูลแบบนี้คือ มันจะวัดระดับความแตกต่างของระดับแรงดันที่อินพุททั้งสอง ดังนั้นอินพุทหนึ่งจะถูกต่อเข้ากับตัวนำที่เป็นตัวนำสัญญาณที่ต้องการส่ง ส่วนอินพุทอีกตัวหนึ่งจะต่อเข้ากับสาย ground ของตัวส่ง การใช้งานลักษณะนี้ถูกใช้ในมาตราฐาน RS-423 แต่ในมาตราฐาน RS-422 และ RS-485 นั้น อินพุทของตัวรับข้อมูล ได้รับสัญญาณความแตกต่างที่ส่งมาจากตัวส่ง สำหรับระดับแรงดันที่ส่งจากตัวส่งจะมีค่าเท่ากับความแตกต่างของเอาท์พุททั้งสอง จากปัญหาที่กล่าวไว้ข้างต้น เราสามารถใช้เซอร์กิตแบบ balanced และ unbalanced แก้ไขได้ดังนี้ การอินเทอร์เฟสแบบ RS-423 สามารถแก้ไขปัญหาที่ระดับแรงดันที่ ground ของ DTE มีค่าสูงกว่า DCE อยู่ 5 โวลต์โดยอาศัยการต่อสาย ground ของ DTE เข้าที่อินพุทแบบขยายความแตกต่าง ของ DCE ส่วน RS-422 และ RS-485 แก้ปัญหานี้

โดยรับเป็นค่าความแตกต่างเข้ามา ซึ่งการแก้ปัญหาโดยใช้การอินเทอร์เฟสแบบนี้ไม่ต้องใช้สาย ground เลย สำหรับปัญหาเกี่ยวกับสัญญาณรบกวนนี้ถูกกำจัดไป เนื่องจากลวดตัวนำที่เป็นอินพุทของตัวรับข้อมูลทั้งสองอินพุทถูกวางผ่านสภาพแวดล้อมทางไฟฟ้าอย่างเดียวกันดังนั้นมันจึงได้รับสัญญาณรบกวนเหมือนกัน สมมุติให้สัญญาณรบกวนที่ได้รับมีแรงดันเป็น x โวลต์ ที่จุด A' และ B'(ในรูป และ) จะมีแรงดันเท่ากับ Va x และ Vb + x (สมมุติให้แรงดันที่จุด A' และ B' ในกรณีที่ไม่มีสัญญาณรบกวนมีค่าเป็น Va และ Vb ตามลำดับ) เนื่องจากินพุทของตัวรับข้อมูลเป็นแบบขยายความแตกต่างดังนั้นสัญญาณรบกวนจะไม่มีผลเนื่องจาก (Va + x) - (Vb + x) มีค่าเท่ากับ Va-Vb ซึ่งเป็นแรงดันที่ต้องการ

มาตราฐาน RS-423



รูปที่ 2.10 ลักษณะสมบัติทางไฟฟ้าของการอินเทอร์เฟสแบบ RS-423

-ตัวผลิตสัญญาณเป็นแบบ Unbalanced

-ตัวรับข้อมูลเป็นแบบขยายความแตกต่าง

-ในแต่ละเซอร์กิจใช้ลวดตัวนำในการส่งสัญญาณเพียง 1 สาย มีสาย ground แยกสำหรับการไหลของสัญญาณกลับในแต่ละทิศทาง

-อัตราเร็วในการส่งข้อมูลมีค่าสูงถึง 100 kbps

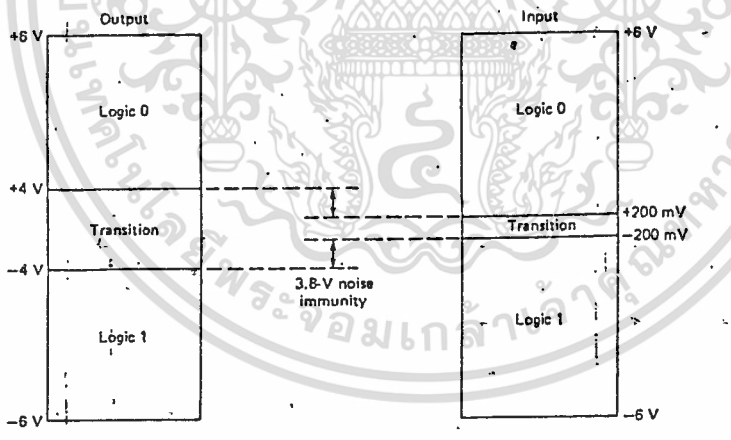
-ระยะทางที่ส่งได้ไกลถึง 4000 ฟุต

มาตราฐาน RS-423 เป็นมาตราฐานที่ได้รับการพัฒนามาจากมาตราฐาน RS-232-C ซึ่งใช้สายสัญญาณเส้นเดียวในการส่งสัญญาณ โดยสัญญาณที่ส่ง ๆ ไปได้ทิศทางเดียว ตัวรับข้อมูลแบบขยายความแตกต่างของสัญญาณระหว่างสาย ground กับตัวส่งสัญญาณ การทำเช่นนี้ช่วยแก้ปัญหาในกรณีที่เกิดความแตกต่าง

หว่างแรงดันที่ ground ของตัวรับข้อมูลกับตัวส่งสัญญาณ

ระดับแรงดันที่ใช้ใน RS-423 มีค่าน้อยกว่า RS-232-C ก็เป็นส่วนทำให้อัตราการส่งข้อมูลมีค่าสูงขึ้น หรือเรียกได้ว่าหนึ่งในส่วนที่เป็นตัวจำกัดอัตราการส่งข้อมูลก็คือ slew rate เพราะว่าระดับแรงดันของ RS-232-C นั้นสูงจึงจำเป็นต้องใช้ค่า slew rate ที่สูง เพื่อที่จะทำการส่งให้มีค่าอัตราการส่งข้อมูลสูงขึ้น แต่เรายังไม่สามารถให้ค่า slew rate นี้สูงได้ (เพราะมีข้อจำกัดอยู่หลายประการ) ดังนั้นเมื่อเราใช้ค่าระดับแรงดันในการส่งให้ต่ำลง ก็ทำให้เรานั้นใช้ค่า slew rate ต่ำลงด้วยสำหรับค่าอัตราการส่งข้อมูลที่เท่ากัน ดังนั้นที่ค่า slew rate เดียวกัน แต่ค่าระดับแรงดันที่ใช้ในการส่งนั้นลดลงมาก็สามารถที่จะทำให้ค่าอัตราการส่งข้อมูลสูงขึ้นได้

RS-423 ใช้ระดับแรงดัน 4 โวลต์ ถึง 6 โวลต์แทน ไบนารี่ 0 และ -4 โวลต์ถึง -6 โวลต์ไบนารี่แทน 1 เมื่อใช้ค่า slew rate ที่เท่ากับ RS-232-C ก็จะให้ค่าอัตราการส่งข้อมูลของ RS-423 ส่งกว่า RS-232 ประมาณ 4 เท่า นอกจากนั้นในการรับค่าไบนารี่ 1 หรือ 0 ที่ตัวรับข้อมูลของRS-423 ก็สามารถรับรู้ค่าได้ในที่ต่ำกว่า RS-232 คือถ้าที่ขาอินพุทของตัวรับข้อมูลมีระดับแรงดันแตกต่างกัน 200 มิลลิโวลต์ ก็จะรับรู้ได้ว่าเป็นไบนารี่ 0 และ -200 มิลลิโวลต์ ก็จะรับรู้เป็นไบนารี่ 1 ก็หมายความว่าการส่งข้อมูลแบบ RS-423 สามารถที่จะยอมให้เกิดค่าแรงดันสูญเสีย (Voltage loss)ในสายได้มากกว่าแบบ RS-232-C



รูปที่ 2.11 แสดงระดับแรงดันไฟฟ้าของการอินเทอร์เฟสแบบ RS-423

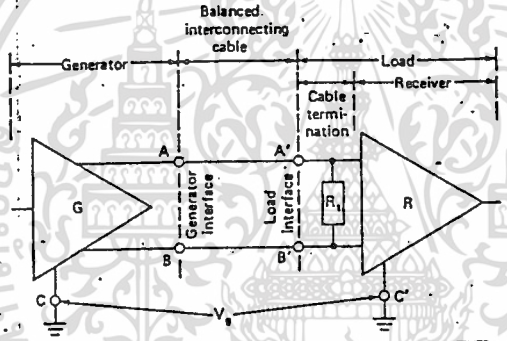
ตัวอย่างการคำนวณอย่างง่าย ๆ ที่จะแสดงผลของการส่งระดับแรงดันที่ตัวรับข้อมูลสามารถที่รับรู้ได้บนอัตราการส่งข้อมูล และ slew rate ที่สามารถทำได้ สมมุติว่า ตัวส่งข้อมูลส่งสัญญาณค่า slew rate 1000 v/s

ส่งแบบ RS-232-C	(-25v ถึง +25v)	50/1000	= 0.05	S
ส่งแบบ RS-423	(-6v ถึง +6V)	12/1000	= 0.012	S

ดังนั้นจะเห็นได้ว่าตัวส่งข้อมูลที่ค่า slew rate เดียวกันนั้น ถ้าส่งแบบ RS-423 จะสามารถทำให้ได้ค่าอัตราการส่งข้อมูลที่สูงกว่า เพราะว่าใช้ระดับแรงดันไฟฟ้าที่มีค่าต่ำกว่า

มาตราฐาน RS-422

มาตราฐาน RS-422 ได้พัฒนาจากมาตราฐาน RS-423 ทำให้อัตราเร็ว ในการส่งข้อมูล มีค่าสูงขึ้น และระยะทางที่ใช้ส่งข้อมูลระหว่างตัวส่งและตัวรับมีระยะไกลขึ้น

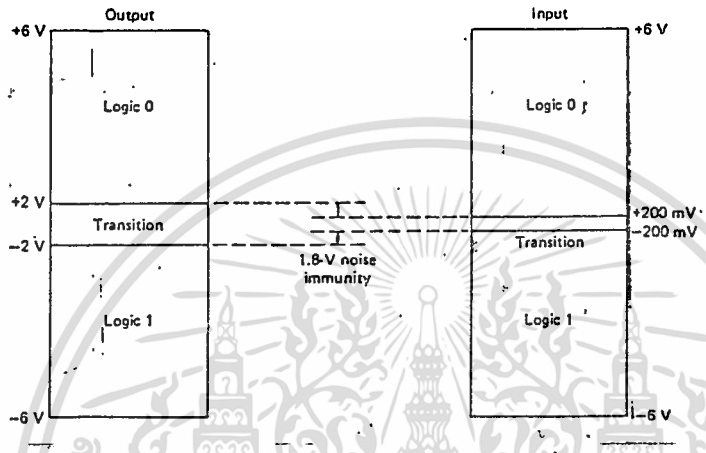


รูปที่ 2.12 ลักษณะสมบัติทางไฟฟ้าของการอินเทอร์เฟสแบบ RS-422
-ตัวส่งสัญญาณเป็นแบบ Balanced
-ตัวรับข้อมูลเป็นแบบขยายความแตกต่าง
-ในแต่ละเซอร์กิตใช้ลวดตัวนำในการส่งสัญญาณจำนวน 2 เส้น
-อัตราเร็วในการส่งข้อมูลสูงถึง 10 mbps
-ระยะทางที่ใช้ในการส่งข้อมูลได้ไกลถึง 4000 ฟุต

มาตราฐาน RS-422 ใช้การส่งข้อมูลในลักษณะของ one-way balanced-line ซึ่งมีตัวส่งข้อมูลบน line 1 ตัว และตัวรับข้อมูล 10 ตัวโดยอัตราเร็วในการส่งข้อมูลมีค่าสูงถึง 10 mbps ที่ระยะทางเท่ากับ 40 ฟุต ในกรณีที่ส่งข้อมูลในอัตราเร็วที่ต่ำกว่า 10 mbps ระยะทางที่ใช้ในการส่งข้อมูลสามารถขยายได้ถึง 4,000 ฟุต

ตัวส่งสัญญาณเป็นแบบ Balanced ระดับแรงดันที่ส่งจากตัวส่งสัญญาณจะมีค่าระหว่าง + 2 โวลต์ ถึง

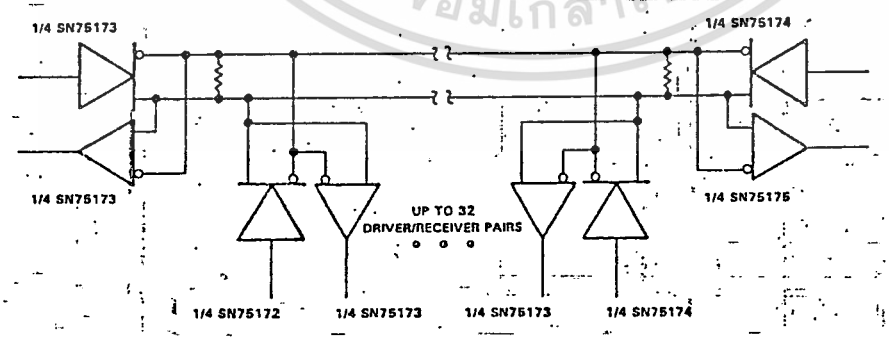
+ 6 โวลต์โดยที่ระดับแรงดัน 2 โวลต์ถึง 6 โวลต์แทนไบนารี่ 0 และ -2 โวลต์ถึง -6 โวลต์ แทนไบนารี่ 1 ซึ่งจะทำให้มีอัตราเร็วในการส่งข้อมูลสูงกว่าแบบ RS-423 และในการรับรู้ค่าไบนารี่ 1 หรือ 0 ที่ตัวรับข้อมูลสามารถจับสัญญาณที่มีระดับต่ำถึง + 200 มิลลิโวลต์ คือ ถ้าที่ขาอินพุทของตัวรับข้อมูลมีระดับแรงดันแตกต่างกัน + 200 มิลลิโวลต์ ก็จะรับรู้ได้ว่าเป็นไบนารี่ 0 และ ถ้าเป็น -200 มิลลิโวลต์ก็จะรับรู้เป็นไบนารี่ 1



รูปที่ 2.13 แสดงระดับแรงดันไฟฟ้า ของการอินเทอร์เฟสแบบ RS-422

มาตราฐาน RS-485 *

มาตราฐาน RS-485 นี้ก็มีการพัฒนามาจาก RS-422 คือผู้ผลิตบางบริษัทได้ทำวงจรขับสัญญาณ(ตัวส่ง) เป็นแบบ tri-state ทำให้เราสามารถส่งข้อมูลได้สองทิศทางบนสายคู่เดียว (single pair) คุณสมบัติข้อนี้จึงทำให้ ระบบส่งข้อมูลมีโครงสร้างเป็นแบบ multidrop ซึ่งอุปกรณ์หลาย ๆ ตัวสามารถรับและส่งข้อมูลแบบ half duplex บนสายคู่เดียวได้



รูปที่ 2.14 ลักษณะสมบัติทางไฟฟ้าของการอินเทอร์เฟสแบบ RS-485

มาตราฐาน RS-485 นี้ ทางบริษัทผู้ผลิตได้ออกแบบตัวส่ง และตัวรับให้สามารถต่อร่วมอยู่บนบัสได้ถึงอย่างละ 32 ตัว แต่การส่งข้อมูลนั้นจะส่งได้ที่ละตัวเท่านั้น ฉะนั้นจะต้องมีกรรมวิธีตรวจสอบบัสก่อนที่จะส่งข้อมูลไปในบัส เพื่อป้องกันการชนกันของข้อมูล ส่วนการรับข้อมูลนั้นสามารถรับได้ พร้อม ๆ กันถึง 32 ตัว แต่ถ้าเราต้องการกำหนดว่าจะให้ตัวรับข้อมูล ตัวไหนนั้นรับข้อมูล เราจะต้องมีการตรวจสอบก่อนที่จะรับข้อมูลว่าข้อมูลที่ส่งมานั้นใช่ของตนเองหรือไม่ ซึ่งมันก็ขึ้นอยู่กับเทคนิคของการเขียนโปรแกรมตรวจสอบของแต่ละบุคคล

ส่วนประกอบของมาตรฐาน RS-485 นี้ก็มี ตัวส่ง, ตัวรับ, สายเคเบิล และ R terminating ซึ่งค่าความต้านทานของ Rt ที่จะนำมาต่อนี้ก็จะขึ้นตัวส่งและตัวรับว่าเราจะใช้ TC เบอร์อะไร ตาม Data sheet ของ IC เบอร์นั้น ๆ จะกำหนดค่า Rt ไว้ให้แล้ว รายละเอียดและคุณสมบัติทางไฟฟ้าของตัวส่งและตัวรับจะถูกออกแบบและตรวจสอบโดยโรงงานผู้ผลิต ให้เป็นไปตามข้อกำหนดของทาง EIA RS-485 (APPIL 1983) ส่วนในเรื่องของการกำหนดรูปแบบการส่งข้อมูล, สายเคเบิล และเรื่องอื่นนั้นไม่ได้มีการกำหนดเอาไว้ ให้ผู้ใช้กำหนดและเลือกเองตามความเหมาะสมของงานนั้น ๆ

ลักษณะของ MCS-51

1. สร้างโดย HMOS และ CHMOS เทคโนโลยีและการทำงานด้วยแหล่งจ่ายไฟขนาดกระแสตรง 5 V. เพียงแหล่งเดียว
2. CPU มีขนาด 8 บิต
3. มีวงจรออสซิลเลเตอร์ และวงจรนาฬิกาบนชิป
4. ชุดแบงค์ (BANK) มี 4 ชุด แต่และชุดมีรีจิสเตอร์ 8 ตัว ทำงานเช่นเดียวกับ MCS-48
5. มีตัวจับเวลา/ตัวนับ ขนาด 16 บิต 2 ชุดและสำหรับเบอร์ 8032/8052 มีทั้ง 3 ชุด
6. มีพอร์ตไอโอแบบขนานสองทิศทาง จำนวน 4 พอร์ต ๆ ละ 8 รวมทั้งหมดเป็น 32 เส้น แต่จะเหลือเพียง 16 เส้น สำหรับเบอร์ 8031 อีก 16 เส้น ใช้ในการเข้าถึงทางแอสเดรสและข้อมูล
7. พอร์ตแบบอนุกรมสามารถที่จะโปรแกรมการรับส่งแบบ FULL DUPLEX ที่มีความเร็วสูง
8. หนึ่งวัฏจักรคำสั่งจะใช้เวลา 1 ไมโครวินาที ด้วยการใช้คริสตอล 12 เม็กกะเฮิรตซ์
9. แอดเดรสข้อมูลภายนอกได้ 64 กิโลไบต์
10. แอดเดรสโปรแกรมภายนอกได้ 64 กิโลไบต์
11. สามารถกำหนดเลขที่อยู่ข้อมูลขนาดไบต์หรือบิตได้
12. มีซอฟต์แวร์แฟลกสำหรับผู้ใช้ที่จะกำหนดเองได้ถึง 128 ตำแหน่งบิต
13. โครงสร้างอินเตอร์รัพท์ทำได้ 5 แหล่ง และ 6 แหล่ง สำหรับ 8032/8052 พร้อมด้วยการจัดไพโอริตี้ (PRIORITY) ได้สองระดับ
14. ตัวโปรเซสเซอร์สามารถใช้งานแบบบูลลีน (BOOLEAN) ได้ สำหรับใช้งานควบคุม
15. มีคำสั่งคูณ และ หาร ทางฮาร์ดแวร์ทำได้ภายใน 4 ไมโครวินาที
16. ตัวเลขทางคณิตศาสตร์ ใช้ได้ทั้งแบบไบนารีและ เดซิมัล
17. การใช้พื้นที่สแต็กสำหรับโปรแกรมย่อยต่าง ๆ ทำได้กว้างขึ้น
18. ชุดคำสั่งของ MCS-51 จะมีมากกว่า MCS-48

*** การต่อเชื่อมแบบอนุกรม (MCS-51) ***

พอร์ตอนุกรมเป็นแบบ FULL DUPLEX สามารถที่จะส่งและรับพร้อมกันได้ โดยทำหน้าที่เป็นบัฟเฟอร์การรับ หมายถึง พอร์ตสามารถที่จะรับไบต์ที่สอง ก่อนที่ตัวแรกจะถูกรับไปจากเรจิสเตอร์ตัวรับ อย่างไรก็ตาม ไบต์ตัวแรกจะต้องถูกอ่านไปก่อน ที่ช่วงเวลาการรับไบต์ตัวที่สองจะสิ้นสุด มิฉะนั้นไบต์ตัวแรก จะถูกซ้อนและสูญหายไปได้ในพอร์ตอนุกรมเรจิสเตอร์ตัวรับและส่งจะเข้าถึงติดต่อกันด้วยเรจิสเตอร์ SBUF ใน SFR แม้ว่าทางโครงสร้างเรจิสเตอร์ทั้งสองจะแยกกันอยู่ก็ตาม

พอร์ตอนุกรมสามารถที่จะเลือกทำงานในโหมดต่างๆได้สี่โหมด

โหมด 0 : ข้อมูลจะเข้าและออกผ่าน RXD TXD ด้วยการเลื่อนสัญญาณนาฬิกาเอาต์พุต ข้อมูลจะเป็นลักษณะแปดบิตในการรับและส่งแต่ละครั้งโดยที่ส่งค่า LSB ก่อนอัตราบ๊อดจะคงที่ที่1/2 ของความถี่ออสซิลเลเตอร์

โหมด 1 : จะเป็นการส่งข้อมูลขนาด 10 บิต ผ่านออก TXD หรือรับเข้ามาผ่าน RDX โดยรูปแบบบิตจะประกอบด้วย หนึ่งบิต start เป็น'0' แปดบิตข้อมูลโดย LSB เป็นตัวแรกที่รับและส่งข้อมูลนี้ และอีกหนึ่งบิต stop มีค่า'1' การรับบิต stop จะนำไปเก็บที่บิต RB8 ของ SFR เรจิสเตอร์ SCON อัตราบ๊อดแปรผันได้ตามการตั้งตัวจับเวลาซึ่งจะกล่าวในในหัวข้อต่อไป

โหมด 2 : เป็นการส่งข้อมูลขนาด 11 บิต ผ่านออกขา TXD หรือรับเข้ามาผ่านขา RXD ประกอบด้วย หนึ่งบิต start มีค่า'0' แปดบิตข้อมูลโดย LSB เป็นตัวแรกที่รับและส่งข้อมูล บิตที่เก้าของข้อมูลสามารถที่จะโปรแกรมเลือกได้ และบิต stop ค่า'1' อีกหนึ่งบิตในการส่งบิตที่เก้าที่อยู่ในบิต TB8 ของเรจิสเตอร์ SCON สามารถที่จะกำหนดเลือกเป็น'1' หรือ'0' ได้ตัวอย่างเช่น การใช้งานเป็นบิตพาริตี้ โดยการเลือกเอาบิต P ของ PSW มาไว้ใน TB8 เพื่อเป็นการส่งข้อมูลแบบมีการตรวจพาริตี้ของข้อมูลที่ส่ง ในการรับข้อมูลบิตที่เก้าจะเข้าไปเก็บที่ RB8 ใน SCON ขณะที่บิต stop จะไปรับเข้ามาเก็บ อัตราบ๊อดสามารถเลือกเป็น 1/32 หรือ 1/64 ของความถี่ออสซิลเลเตอร์

SCON เป็น SFR ที่ใช้ในการกำหนดโหมดการทำงานของพอร์ตอนุกรม เช่น การกำหนดค่า RB8 จะเป็นการใช้ตัวรับบิตที่เก้าด้วยหรือไม่ เป็นต้น รูปที่ 2.15 เป็นตารางการใช้ SCON ในการควบคุมการทำงานของพอร์ตอนุกรม

โหมด 3 : เป็นการส่งข้อมูลขนาด 11 บิตผ่านออกขา TXD หรือรับเข้ามาผ่านขา RXD ประกอบด้วยบิต start มีค่า 0 แปดบิตข้อมูลโดย LSB เป็นบิตแรกที่รับและส่งข้อมูลบิตที่เก้าของข้อมูลสามารถที่จะโปรแกรมเลือกได้ และบิต stop ค่า 1 อีกหนึ่งบิตในความเป็นจริง โหมด 3 จะคล้ายกับโหมด 2 ทุกประการ ยกเว้นอัตราบ๊อดในโหมด 3 จะแปรผันได้ไปตามการโปรแกรม

การเลือกตัวจับเวลา

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

โดย SM0,SM1 เป็นตัวกำหนดใช้โหมดต่าง ๆ ของพอร์ตอนุกรมดังนี้

SM0	SM1	โหมด	ลักษณะการทำงาน	อัตราบ๊อด
0	0	0	เลื่อนเรจิสเตอร์	*fosc*/12
0	1	1	8-บิต UART	แปรผันได้ตามการเลือกตัวจับเวลา
1	0	2	9-บิต UART	*fosc*/64 หรือ *fosc*/32
1	1	3	9-บิต UART	แปรผัน

* UART : Universal Asynchronous Receiver/Tranceiver

SM2 ควบคุมอีนาเบิ้ล การใช้โพรเซสเซอร์หลายตัวในการสื่อสารซึ่งกันและกัน
ในโหมด 2 และ 3 ถ้า SM2 เซตเป็น 1 ดั้งนั้น RI จะต้องไม่แอ็คทีฟ ถ้ามีการรับบิตที่เก้า ทำให้บิต RB8 นี้เป็น 0
ในโหมด 1 ถ้า SM2 เซตเป็น 1 ดั้งนั้น RI จะไม่แอ็คทีฟถ้า stop บิตไม่ถูกรับ
ในโหมด 0 SM2 ควรมีค่าเท่ากับ 0

REN ตัวอีนาเบิ้ลอนุกรมการรับ เซตเป็น 1 ด้วยโปรแกรมการเลือกอีนาเบิ้ลการรับและเป็น 0 ด้วยโปรแกรม เมื่อให้เป็นดิสเอเบิ้ล

TB8 เป็นข้อมูลบิตที่เก้า ซึ่งจะถูกส่งในโหมด 2 และ 3 ซึ่งจะให้เป็น 1 หรือ 0 ได้ด้วยโปรแกรม

RB8 ในโหมด 2 และ 3 ข้อมูลบิตที่เก้าจะถูกรับไป
ในโหมด 1 ถ้า SM2=0 RB8 จะกลายเป็น stop บิตที่ถูกรับไป
ในโหมด 0 RB8 ไม่ใช้

TI เป็นแฟลกอินเตอร์รัพต์การส่ง เซตด้วยฮาร์ดแวร์คือสัญญาณที่ปลายช่วงเวลาของบิตที่แปด ในโหมด 0 หรือที่จุดเริ่มต้นของบิต stop ในโหมดอื่น ในการส่งแบบอนุกรมของทุกโหมดจะต้องเคลียร์บิตนี้ด้วยโปรแกรมหลังการส่ง

RI เป็นแฟลกอินเตอร์รัพต์การรับ เซตด้วยฮาร์ดแวร์คือสัญญาณที่ปลายช่วงเวลาของบิตที่แปดในโหมด 0 หรือที่จุดครึ่งทางของช่วงบิต stop ในโหมดอื่น ในการรับแบบอนุกรมยกเว้นกรณีใช้ SM2 จะต้องเคลียร์บิตด้วยโปรแกรหลังการรับ

รูปที่ 2.15 SCON : เรเตอร์ควบคุมพอร์ต อนุกรม

ทั้งสี่โหมดนี้ การส่งข้อมูลจะถูก Initiated ด้วยคำสั่งใด ๆ ที่ใช้ตัวเรจิสเตอร์ SBUF เป็นเรจิสเตอร์ตัวรับข้อมูลจากซีพียู และในโหมด 0 การรับข้อมูลจะ Initiated ด้วยการใช้สถานะ RI=0 REN=1 ส่วนในโหมดอื่นการรับข้อมูล จะถูก Initiated ด้วยการรับบิต start เข้ามาตรวจสอบ ถ้า REN=1

การทำงานทั้งสี่โหมดของพอร์ตอนุกรมได้สรุปไว้ในรูปที่ 2.15 ต่อไปเป็นการทำงานโดยละเอียดในโหมดต่าง ๆ 4 โหมดของพอร์ตอนุกรม

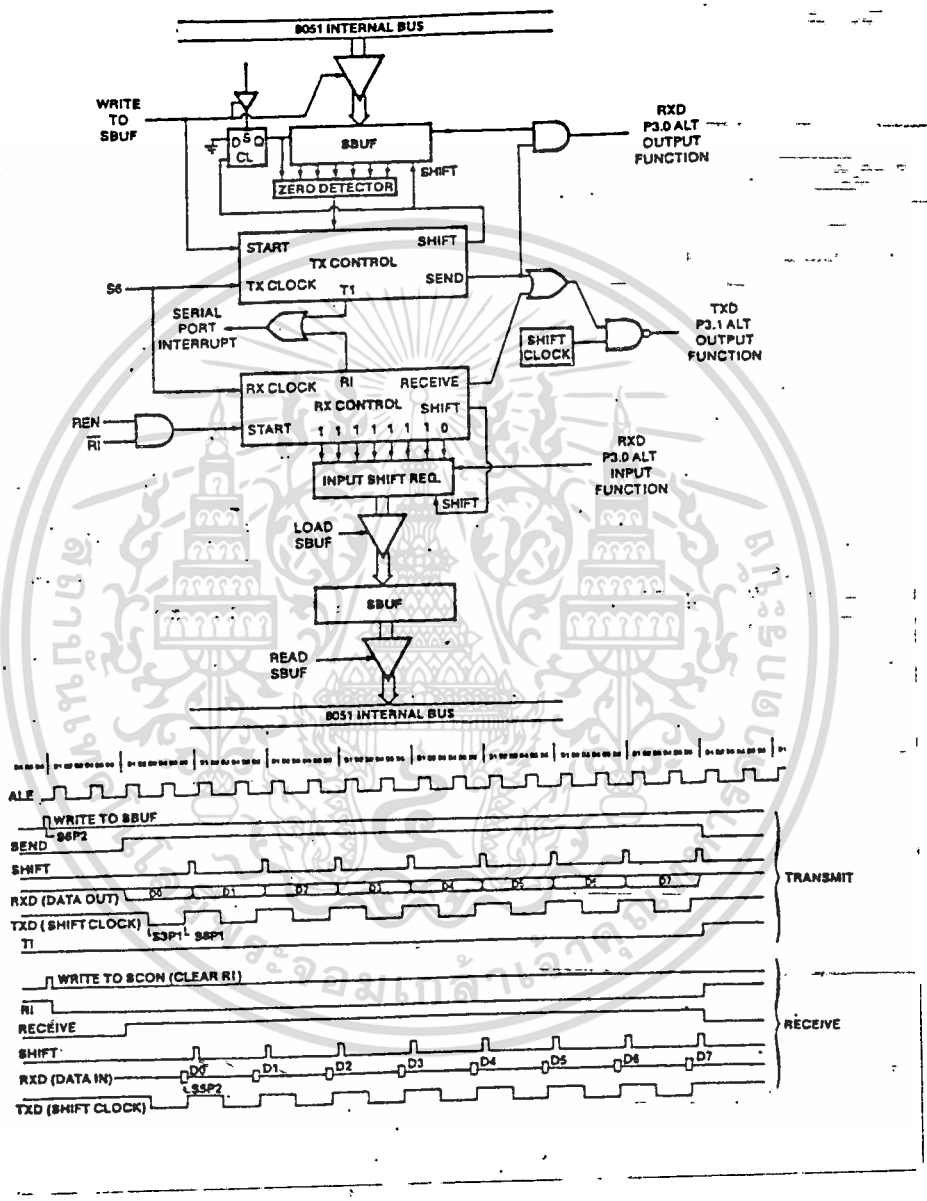
โหมด 0

ข้อมูลอนุกรมจะส่งและรับผ่านตัวเอาต์พุต TXD,RXD แบบเลื่อนข้อมูลเป็นอนุกรม 8 บิตที่จะส่งหรือรับข้อมูลขนาด 8 บิต โดยเลื่อนบิต LSB เป็นตัวแรกอัตราบ๊อดคงที่ที่ 1/12 ของความถี่ออสซิลเลเตอร์ รูปที่ 2.16 แสดงถึงแผนภูมิการใช้งานในโหมด 0 พร้อมกับแผนภูมิเวลาสำหรับการส่งและรับ

การส่งจะถูกเริ่ม (Initiated) ด้วยคำสั่งต่าง ๆ ในการใช้ SBUF เป็นเรจิสเตอร์รับข้อมูล (Destination) สัญญาณคำสั่ง 'write to SBUF' เกิดที่ S6P2 และจะบรรจุค่า'1' เข้าที่ตำแหน่งบิตที่เก้าของตัวเรจิสเตอร์การเลื่อนส่ง และบอกให้ส่วนควบคุมการส่ง(TX Control Block) ให้ทำงานการส่ง ภายในช่วงเวลาหนึ่งวัฏจักรแมชชีนจะครอบคลุมให้เกิดพัลส์ 'write to SBUF' ที่ S6P2 และสัญญาณ SEND ต้องแอ็กทีฟสูงในช่วงการส่ง

การอินาเบิ้ล SEND ที่มีแอ็กทีฟสูงจะเป็นการควบคุมให้ข้อมูลส่งออกจากตัวเรจิสเตอร์การเลื่อนบิตออกที่ขา p3.0 และตัวเลื่อนสัญญาณนาฬิกา (Shift Clock) ก็จะถูกให้ส่งออกที่ขา P3.1 โดยที่ลักษณะของสัญญาณ Shift Clock จะมีระดับต่ำที่ S3, S4 และ S5 ของทุกวัฏจักรแมชชีน และมีระดับสูงช่วง S6 S1 และ S2 ในช่วง S6P2 ของทุกวัฏจักรแมชชีนขณะที่ SEND แอ็กทีฟ ค่าข้อมูลในเรจิสเตอร์การเลื่อนส่งจะถูกเลื่อนไปทางขวาหนึ่งตำแหน่งหรือบิต คือ เป็นการส่งที่ขา P3.0 ไปหนึ่งบิตขณะนั้น

ในขณะที่เลื่อนบิตไปทางขวา ค่า "0" จะเข้าแทนที่ทางซ้าย เมื่อเลื่อนจน MSB เลื่อนออกจากตัวเรจิสเตอร์การเลื่อน ดังนั้นค่า "1" จะเริ่มถูกบรรจุเข้าไปที่ตำแหน่งที่เก้าที่ต่อจาก MSB ที่ถูกเลื่อนออกไป และทุกตำแหน่งทางซ้ายมือจะเป็น "0" หมดทุกตัว สถานะของแฟลกในส่วนควบคุมการส่ง จะทำการเลื่อนครั้งสุดท้าย และให้สัญญาณ SEND กลับคืนจากสภาพแอ็กทีฟเป็นระดับต่ำ และปรับ TI เป็นระดับสูงด้วย ส่งออกไปที่พอร์ตอนุกรมการอินเตอร์รัพต์ ทั้งสองสัญญาณที่ปรับระดับนี้ จะเกิดช่วง S1P1 ของวัฏจักรแมชชีนที่สิบ หลังจากที่สัญญาณ "Write to SBUF" เริ่มสโตรบ



รูปที่ 2.16 เป็นการแสดงถึงฟังก์ชันแผนภูมิแบบธรรมดาของพอร์ตอนุกรมในโหมด 0 และช่วงเวลาที่เกิดขึ้น

การรับข้อมูลจะถูกควบคุมด้วยการโปรแกรมเริ่มแรก การตั้งข้อแม้ให้ REN=1 และ RI=0 ที่ S6P2 ของวัฏจักรแมชชีนตัวใหม่ หน่วยควบคุม RX จะเขียนบิตค่า 1111 1110$_2$ ไปยังเรจิสเตอร์การเลื่อนรับ และที่เฟส clock ตัวใหม่ จะแอ็กทีฟสัญญาณ Receive ให้สูง

เมื่อ Receive ถูกอีนาเบิ้ลให้สูง ก็จะทำให้สัญญาณ Shift clock ส่งฟังช์ชั่นต่าง ๆ ออกที่ขา p3.1 Shift clock จะเปลี่ยนสถานะที่ S3P1 และ S6P1 ของทุกวัฏจักรแมชชีนที่ S6P2 ของวัฏจักรแมชชีนตัวแรก สัญญาณ Receive เริ่มแอ็กทีฟสูงและค่าในเรจิสเตอร์การเลื่อนรับจะรับข้อมูลเข้ามา และเลื่อนมาทางซ้ายหนึ่งตำแหน่ง และทุกค่าที่เข้ามาจากทางขวาจะเป็นค่าที่ถูก Sample เข้าที่ขา p3.0 ที่ช่วง S5P2 ของทุกวัฏจักรแมชชีน

ขณะที่ข้อมูลบิตต่ำเข้ามาทางขวา ค่า "1" จะเลื่อนออกไปทางซ้าย เมื่อค่า "0" ถูกบรรจุเริ่มแรกที่เข้ามาทางตำแหน่งขวาสุด ถูกเลื่อนมาอยู่ตำแหน่งซ้ายสุดในเรจิสเตอร์การเลื่อนมันจะมีผลให้ค่าแฟลกในหน่วยควบคุมการรับ (RX Control Block) ให้ทำการเลื่อนเป็นค่าสุดท้าย และเริ่มบรรจุข้อมูลทั้งหมดเข้าใน SBUF ที่ช่วง S1P1 ของวัฏจักรแมชชีนที่สิบ หลังจากเริ่มส่งสัญญาณสโตรบ "write to SCON"ของ RI เคลียร์ ต่อจากนั้น สัญญาณ Receive จะเคลียร์ และ RI จะปรับเป็น 1

โหมด 1

จำนวนสิบบิตจะถูกส่งผ่าน TXD หรือรับผ่าน RXD ที่ประกอบด้วยบิต start บิตข้อมูล 8 บิต และบิต stop การรับบิต stop จะส่งเข้า RB8 ในเรจิสเตอร์ SCON การตั้งอัตราความเร็วของบ๊อดจะแปรผันได้ ทั้งตัวจับเวลา 1 หรือ 2 อาจใช้เป็นสัญญาณนาฬิกาสำหรับพอร์ตอนุกรมโดยสร้างอัตราความเร็วแปรผันด้วยการตั้งหรือเคลียร์ค่าบิตใน T2CON เป็น TCLK และ RCLK รูปที่ 2.17 สแดงถึงแผนภูมิการใช้งานในโหมด 1 พร้อมกับแผนภูมิเวลาสำหรับสัญญาณการส่งและรับ

การส่งและเริ่มต้นงานด้วยคำสั่งที่ใช้ SBUF เป็นเรจิสเตอร์รับข้อมูลสัญญาณ "write to SBUF" ก็จะบรรจุค่า 1 เข้าไปเป็นตำแหน่งที่เก้าในเรจิสเตอร์การเลื่อนส่ง และแฟลกในหน่วยควบคุมการส่ง (TX Control Block) ก็จะแสดงการร้องขอให้ส่งข้อมูล การส่งข้อมูลจะส่งที่ช่วง S1P1 ของวัฏจักรแมชชีน และจะตามด้วยบิตตัวต่อมา ในช่วงเวลาของสัญญาณนาฬิกาที่หารด้วย 16 ที่ถูกตั้งที่ตัวนับ ดังนั้น แต่ละบิตจะถูกชิงค์ด้วยการหาร 16 ของตัวนับ ไม่ใช่ด้วยสัญญาณ "write to SBUF"

การส่งจะเริ่มด้วยการส่งแอ็กทีฟสัญญาณ SEND และใส่บิต start เข้าที่ TXD ช่วงเวลาหลังจากนั้นหนึ่งบิต สัญญาณของข้อมูลจะแอ็กทีฟ ซึ่งก็จะอีนาเบิ้ลการส่งบิตออกจากเรจิสเตอร์การเลื่อนส่งออกไปยังขา TXD พัลส์เลื่อนตัวบิตแรกจะเกิดขึ้นหลังเวลาทำงานแล้วหนึ่งบิต

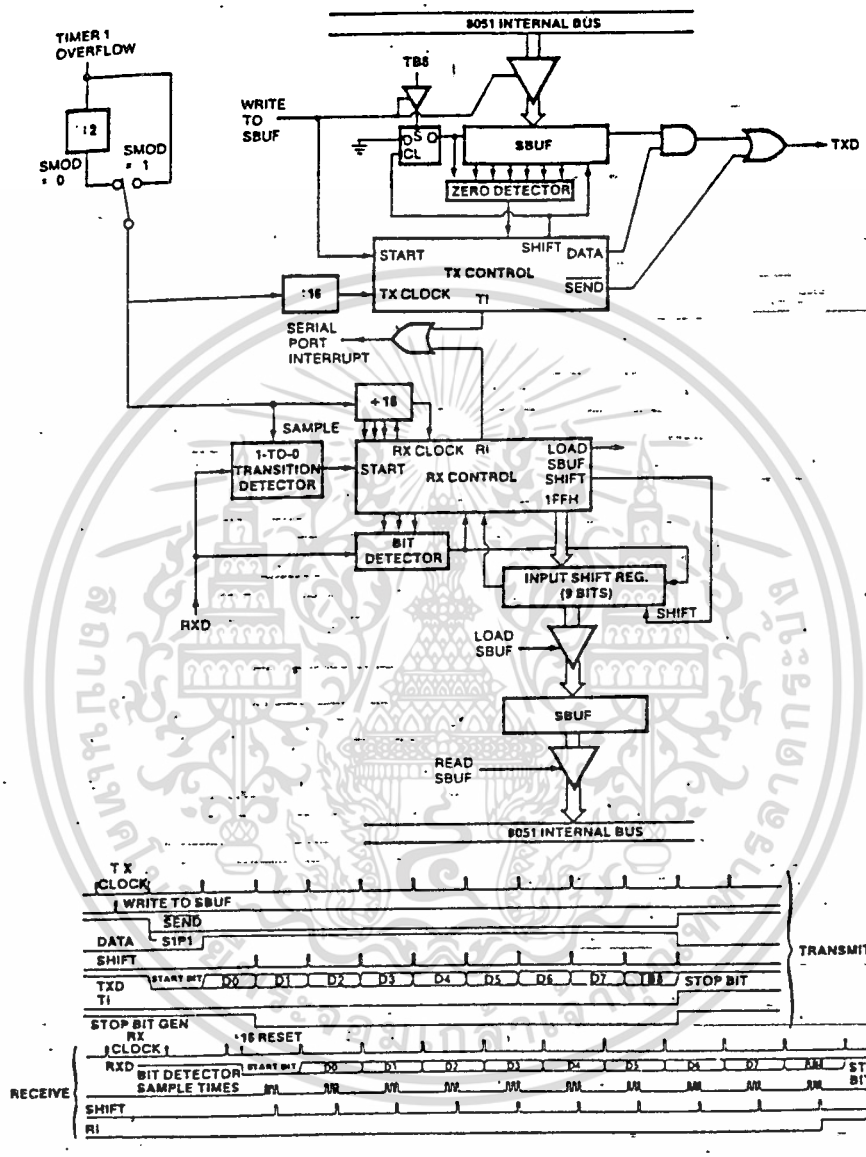
ขณะข้อมูลเลื่อนออกทางขวา ค่า 0 จะถูกใส่เข้าทางซ้าย เมื่อ MSB ของข้อมูลหนึ่งไบต์อยู่ที่ตำแหน่งเอาต์พุตของเรจิสเตอร์ตัวเลื่อน ขณะนั้นค่า 1 จะเริ่มบรรจุเข้าเป็นตำแหน่งที่เก้าที่เอาต์พุตหลังจากที่ MSB ถูกส่งออกไป และทุกตำแหน่งเมื่อถูกส่งไปแล้ว ที่เหลือในเรจิตเตอร์การเลื่อนจะเป็น 0 หมด สถานะแฟลกในหน่วยควบคุมการส่ง (TX Control Unit) ก็จะเลื่อนเป็นตัวสุดท้าย และส่งสัญญาณ SEND ดิสเอเบิ้ล และการเซต TI จะเกิดขึ้นในช่วงลูกที่สิบของการหาร 16 หลังการส่งสัญญาณสโตรบ "write to SBUF"

การรับจะถูกเริ่มงานด้วยการกระตุ้นจากการเปลี่ยนแปลง 1 เป็น 0 ที่ RXD สำหรับจุดนี้ RXD จะถูก Sample ด้วยอัตรา 16 เท่าของอัตราบ๊อดที่ถูกกำหนดเริ่มแรก เมื่อการส่งข้อมูลถูกรับได้ ตัวนับหาร 16 ก็จะถูกรีเซต และค่า 01FFH ก็จะถูกเขียนเข้าไปในเรจิสเตอร์ตัวเลื่อน การรีเซตตัวนับหาร 16 ก็จะเป็นการตั้งวนรอบด้วยการให้ขอบเขตของช่วงเวลาแต่ละบิตที่เข้ามา

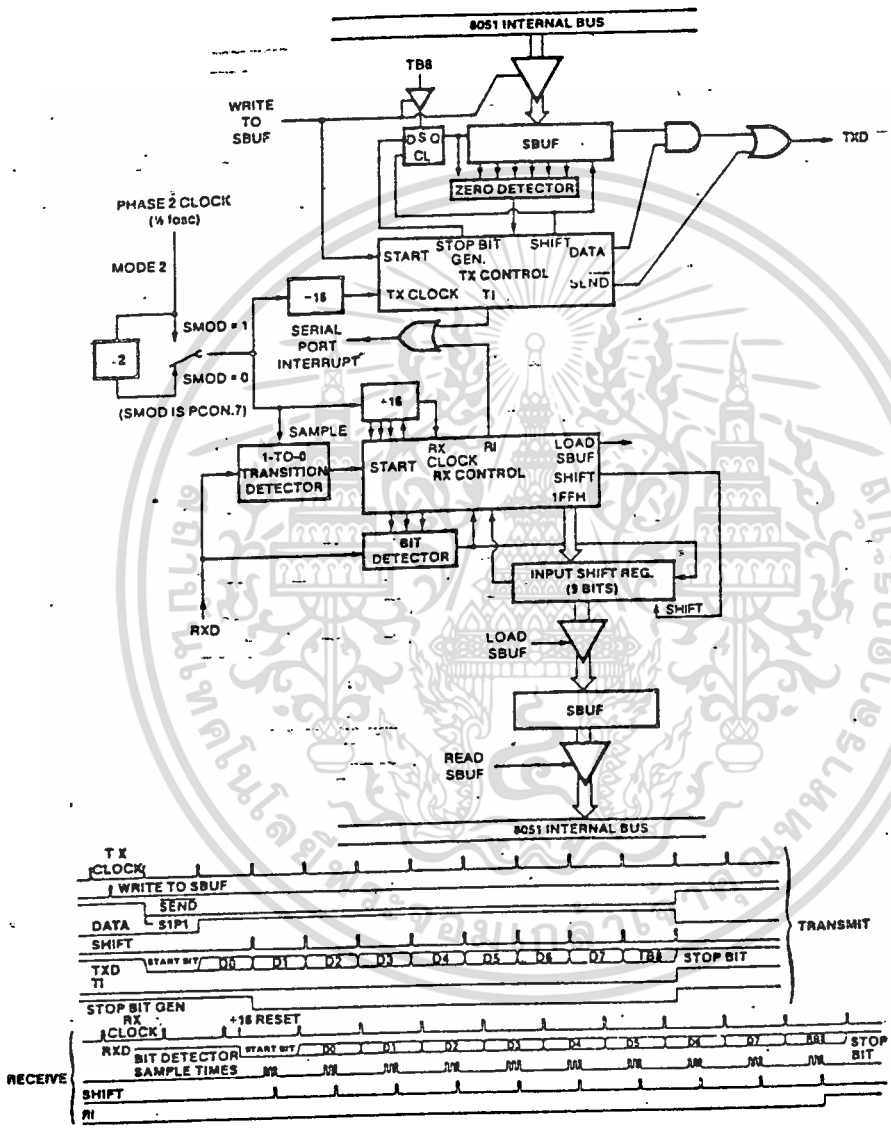
16 คาบเวลาของตัวนับแต่ละบิต จะเป็นเวลาที่เข้ามาในคาบที่ 16 ที่ตัวนับนับคาบที่ 7 8 และ 9 จะเป็นช่วงเวลาของบิต เป็นการรับข้อมูลแต่ละบิตที่ Sample ค่าที่เข้ามาทาง RXD และค่าที่รับเข้ามาถูก Sample อย่างน้อย 2-3 ครั้ง การทำเช่นนี้จะเป็นการขจัด Noise ออกไป ถ้าค่าข้อมูลถูกรับในระหว่างช่วงเวลาบิตแรกที่ไม่ใช่ค่า 0 วงจรการรับจะถูกรีเซตและหน่วยรับก็จะกลับไปตรวจการเปลี่ยนแปลงจาก 0 -> 1 ใหม่ ลักษณะงานเช่นนี้จะเป็นการป้องกันรับบิต start ที่ผิดพลาดเข้ามาได้ ถ้าบิต start ถูกรับเข้ามาถูกต้อง มันก็จะถูกเลื่อนเข้าเรจิสเตอร์ตัวเลื่อน และการรับข้อมูลก็จะเริ่มขึ้น

ขณะที่ข้อมูลเข้ามาจากทางขวา ค่า 1 จะถูกเลื่อนออกไปทางซ้ายเมื่อค่าบิต start ถูกเลื่อนมาถึงทางซ้ายสุดในเรจิสเตอร์ตัวเลื่อน มันก็จะแฟลกในหน่วยควบคุมการรับ (RX Control Block) ให้เลื่อนอีกหนึ่งครั้งเป็นครั้งสุดท้าย และก็จะบรรจุข้อมูลเข้า SBUF และ RB8 (เพราะมีเก้าบิต) และเซต RI สัญญาณการบรรจุเข้า SBUF , RB8 และการเซต RI เป็น 1 จะปรากฏ ถ้าเพียงแต่กรณีใดต่อไปนี้ปรากฏในช่วงเวลาพัลส์การเลื่อนสุดท้ายเกิดขึ้น คือ

1. RI = 0
2. SM2 = 0 หรือการรับ stop บิต = 1

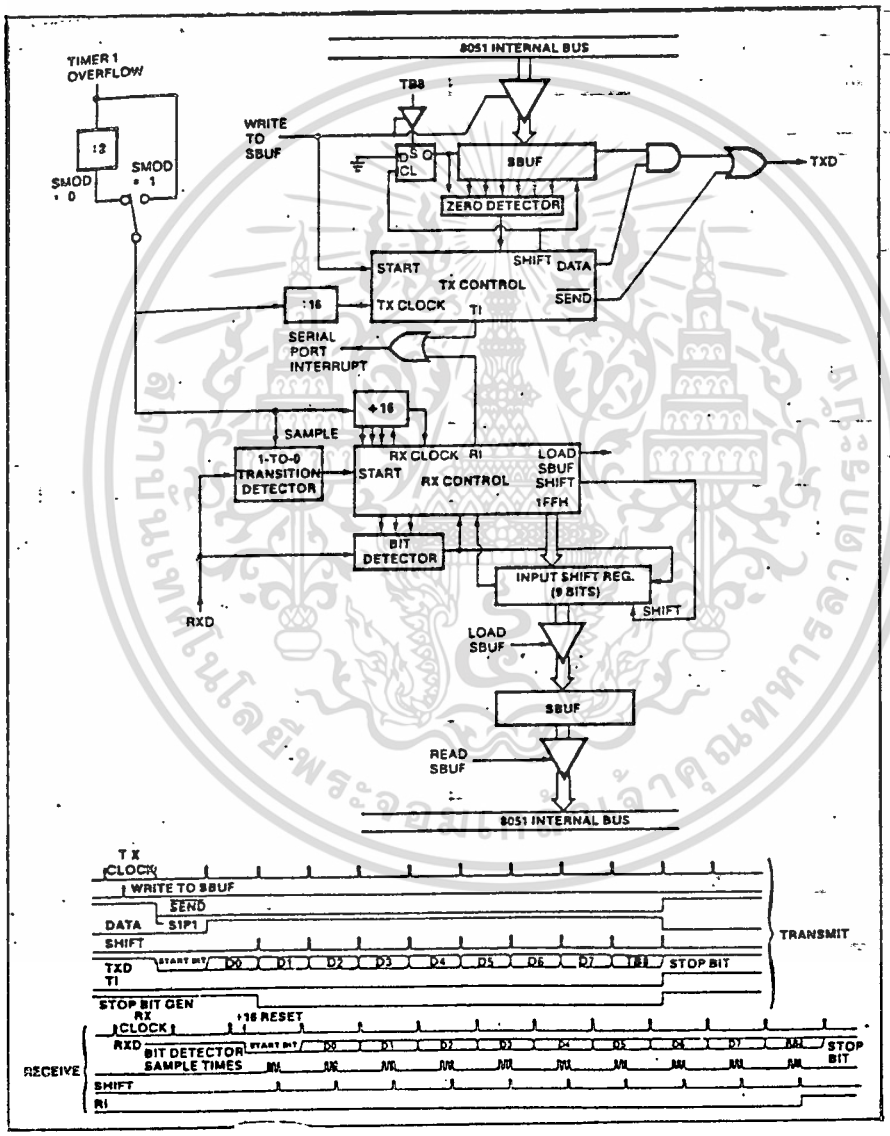


รูปที่ 2.17 แสดงถึงแผนภูมิการใช้งานในโหมด 1 พร้อมกับแผนภูมิเวลาสำหรับสัญญาณการส่งและรับ



รูปที่ 2.18 แสดงแผนภูมิฟังก์ชันของพอร์ตอนุกรมในโหมด 2

ถ้าไม่เกิดทั้งสองกรณีการ รับข้อมูลไบต์ก็จะล้มเหลว ถ้าเกิดทั้งสองกรณี ตัว stop บิตก็จะไปเก็บที่ RB8 และข้อมูลไบต์ก็จะเข้า SBUF และ RI จะแอ็กทีฟสูงช่วงเวลานี้ไม่ว่าจะเกิดขึ้นทั้งสองกรณีหรือไม่ หน่วยควบคุมการรับก็จะกับไปตรวจการเปลี่ยนแปลง 1-0 ของการส่งข้อมูลผ่าน RXD ใหม่



รูปที่ 2.19 แสดงแผนภูมิฟังก์ชันของพอร์ตอนุกรมในโหมด 3

โหมด 2 และ 3

จำนวน 11 บิตจะส่งออกที่ TXD. และรับเข้าที่ RXD โดยมีบิต start มีค่า 0 ข้อมูล 8 บิตมี LSB เป็นบิตแรกและโปรแกรมบิตที่เก้าได้ และบิต stop มีค่า 1 การส่งข้อมูลบิตที่เก้าใช้ TB8 เป็นตัวกำหนดค่า 0 หรือ 1 การรับข้อมูลบิตที่เก้าใช้ RB8 ใน SCON เป็นตัวรับ อัตราบ็อดสามารถโปรแกรมเลือกได้ทั้งแบบ 1/32 หรือ 1/64 ของความถี่ออสซิลเลเตอร์ในโหมด 2 แต่โหมด 3 จะใช้ตัวแปรหลายค่าของอัตราบ็อด เกิดจากการใช้ตัวจับ 1 หรือ 2 ขึ้นอยู่กับสถานะ TCLK และ RCLK

รูปที่ 2.18 และ รูปที่ 2.19 แสดงแผนภูมิฟังก์ชันของพอร์ตอนุกรมในโหมด 2 และ 3 ส่วนของตัวรับจะทำงานเหมือนกับโหมด 1 ส่วนของตัวส่งต่างจากโหมด 1 เพียงบิตที่เก้าของเรจิสเตอร์ตัวเลื่อนการส่ง

การส่งจะเริ่มด้วยคำสั่งใด ๆ ที่ใช้ SBUF เป็นเรจิสเตอร์ตัวรับข้อมูล สัญญาณ "write to SBUF"ก็จะบรรจุ TB8 เข้าไปเป็นบิตที่เก้าของตำแหน่งในเรจิสเตอร์ตัวเลื่อนการส่งและแฟลกในหน่วยควบคุมการส่งก็จะให้การส่งถูกร้องขอ ช่วงการส่งจะเริ่มขึ้นที่ S1P1 ของวัฏจักรแมชชีนต่อ ๆ มา ในการใช้ตัวนับหาร 16 ดังนั้นช่วงแต่ละบิตจะซิงค์โคไนส์กับตัวนับหาร 16 โดยไม่ใช้สัญญาณของ "write to SBUF"

การส่งเริ่มด้วยสัญญาณ SEND แอ็กทีฟต่ำ ใส่บิต start ออกที่ TXD หลังจากนั้นบิตแต่ละตัวของข้อมูลในสัญญาณ DATA ก็จะแอ็กทีฟ ซึ่งจะอีนาเบิ้ลให้บิตของเรจิสเตอร์ตัวเลื่อนส่งออกที่ TXD ตามมา โดยตัวเลื่อนพัลส์ตัวแรกจะเกิดหลังจากนั้นเล็กน้อย ที่สัญญาณนาฬิกาเลื่อนตัวแรก ถ้า 1 ถือเป็นบิต stop จะใส่เข้าไปที่เรจิสเตอร์ตัวเลื่อนทางซ้ายสุด เป็นตำแหน่งบิตที่เก้า ดังนั้นหลังจากสัญญาณนาฬิกาเข้าตัวต่อมาจะใส่ค่า 0 เข้าไปเท่านั้น และ ทุกสัญญาณนาฬิกาจะเลื่อนเอาบิตออกทางขวา และใส่ค่า 0 เข้าทางซ้ายเมื่อ TB8 อยู่ที่ตำแหน่งเอาต์พุตของเรจิสเตอร์ตัวเลื่อน ดังนั้นบิต stop จะส่งออกต่อ TB8 และทุกตำแหน่งในเรจิสเตอร์ตัวเลื่อนที่เหลือจะเป็นศูนย์หมด ด้วยสถานะเช่นนี้จะทำให้แฟลกในหน่วยควบคุมการส่งเลื่อนเป็นครั้งสุดท้าย และให้สัญญาณ SEND ดิสเอเบิ้ลสูง และ เซต TI ด้วย ซึ่งจะเกิดขึ้นที่พัลส์ลูกที่ 11 ของตัวหาร 16 สัญญาณนาฬิกาหลังจากส่งสโตรบสัญญาณ "write to SBUF"

การรับจะเริ่มทำงานด้วยการจับสัญญาณที่ขา RXD ช่วงการเปลี่ยนจาก 1 เป็น 0 สำหรับโหมดนี้ ตัว RXD จะถูก Sampled ที่อัตรา 16 เท่าของอัตราบ็อดที่ถูกกำหนด เมื่อสภาวะการเปลี่ยนแปลงถูกรับได้ ตัวหาร 16 ก็จะเริ่มเซตทันทีและค่า 1FFH ก็จะเขียนเข้าไปที่เรจิสเตอร์ตัวเลื่อน

ที่ช่วงการนับลูกที่ 7,8 และ 9 ของแต่ละช่วงบิต ตัวดีเทคเตอร์จะ Sample ค่าบิตของสัญญาณที่เข้า RXD ค่าที่รับมาจะเป็นค่าที่คล้ายกับว่ามีการรับเข้า Sampling อย่างน้อย 2 ถึง 3 Sample ถ้าค่าถูกรับเข้าช่วงระหว่างบิตแรกไม่เป็น 0 วงจรตัวรับก็จะรีเซต และหน่วยควบคุมก็จะกลับตรวจการเปลี่ยนแปลงจาก 1 เป็น 0 ใหม่ ถ้าบิต start ถูกพิสูจน์ว่าถูกต้องมันก็จะเลื่อนเข้าสู่เรจิสเตอร์ตัวเลื่อนและการรับก็จะรับจนครบ frame ของข้อมูลในโหมดนี้

ขณะที่ข้อมูลเข้ามาจากทางขวา ค่า '1' ก็จะถูกเลื่อนไปทางซ้ายออกไป เมื่อบิต start ถูกเลื่อนมาถึงตำแหน่งทางซ้ายสุดของเรจิสเตอร์ตัวเลื่อน โดยในโหมด 2 และ 3 จะมีเรจิสเตอร์เก้าบิต มันจะแฟลกให้หน่วยควบคุมการรับทำการเลื่อครั้งสุดท้าย แล้วบรรจุค่าใน SBUF และ RB8 และเซต RI สัญญาณการบรรจุ SBUF และ RB8 และการเซต RI เป็น '1' จะถูกสร้างขึ้น ถ้าเพียงแต่ เกิดกรณีใดกรณีหนึ่งต่อไปนี้ ปรากฏในช่วงเวลาพัลส์การเลื่อนลูกสุดท้ายคือ

1. RI = 0
2. SM2 = 0 หรือและการรับบิตที่เก้ามีค่า = 1

ถ้าทั้งสองกรณีไม่เกิดขึ้น การรับ Frame ของข้อมูลก็จะสูญหาย และ RI จะไม่เซต ถ้าทั้งสองกรณีเกิดขึ้น การรับบิตที่เก้าจะรับเข้า RB8 และแปดบิตแรกจะบรรจุเข้าใน SBUF ช่วงเวลาหนึ่งบิต หลังจากนั้น ไม่ว่าจะได้รับข้อมูลหรือข้อมูลสูญหาย หน่วยควบคุมก็จะตรวจสอบการเปลี่ยนแปลงค่า 1 เป็น 0 ที่อินพุตของ RXD ใหม่

การสื่อสาร Multiprocessor

โหมด 2 และ 3 มีการใช้ทำงานพิเศษสำหรับการสื่อสารทางมัลติโพรเซสเซอร์ ทั้งสองโหมดนี้ การรับบิตที่เก้าของข้อมูลจะรับเข้า RB8 แล้วจึงจะตามมาด้วยบิต stop พอร์ตสามารถถูกโปรแกรมเมื่อบิต stop ถูกรับเข้ามา การอินเตอร์รัพต์ทางพอร์ตอนุกรมจะแอ็กทีฟเท่านั้นถ้า RB8 = 1 การทำงานลักษณะนี้จะอีนาเบิ้ลได้ด้วยการเซตบิต SM2 ในเรจิสเตอร์ SCON เพื่อให้ใช้งานระบบมัลติโพรเซอร์ตามลำดับดังนี้

เมื่อตัวโพรเซสเซอร์หลักต้องการส่งข้อมูลจำนวนหนึ่งไปยังโพรเซสเซอร์ตัวลูกทั่ว ๆ ไป มันจะต้องส่งไบต์แอดเดรสกำหนดเป้าหมายตัวลูกทั่ว ๆ ไปก่อน ไบต์แอดเดรสต่างจากไบต์ข้อมูลที่บิตที่เก้ามีค่าเป็น '1' ขณะที่บิตที่เก้าของข้อมูลมีค่าเป็น '0' SM2 = 1 เป็นการอีนาเบิ้ลกลุ่มโพรเซสเซอร์ลูกไม่ให้ถูกอินเตอร์รัพต์ด้วยไบต์ข้อมูล อย่างไรก็ตามไบต์แอดเดรสจะอินเตอร์รัพต์โพรเซสเซอร์ลูกทั้งหมด การทำเช่นนี้จะเป็นการช่วยให้โพรเซสเซอร์ลูกแต่ละตัวตรวจสอบไบต์ที่รับเข้า ถ้ามันเป็นโพรเซสเซอร์ลูกเป้าหมายที่ถูกกำหนดด้วยไบต์แอดเดรส มันจะเคลียร์ค่าบิต SM2 และเตรียมรับไบต์ข้อมูลที่จะเข้ามาต่อไป ตัวโพรเซสเซอร์ลูกตัวอื่นที่ไม่ได้ถูกไบต์แอดเดรสกำหนด ก็ยังคงเซตค่า SM2 และยังคงทำงานในส่วนเฉพาะของตัวเอง ในขณะเดียวกันก็จะ

ไม่มีการรับข้อมูลไบต์ที่ส่งมา ค่า SM2 ควรจะเคลียร์สำหรับการทำงานในโหมด 0

อัตราบ๊อด

อัตราบ๊อดในโหมด 0 ของการใช้พอร์ตอนุกรมจะคงที่ที่ความถี่ของออสซิลเลเตอร์คือ

อัตราบ๊อดในโหมด 0 = ความถี่ออสซิลเลเตอร์/12

อัตราบ๊อดในโหมด 2 จะขึ้นอยู่กับการปรับค่าบิตใน SMOD ของ SFR ในเรจิสเตอร์ PCON ถ้า SMOD = 0 ซึ่งจะเป็นค่าที่ถูกรีเซตแต่แรก หลังการรีเซต อัตราบ๊อดจะเป็น 1/64 ความถี่ออสซิลเลเตอร์ มีสูตรเป็น

SMOD บิต

อัตราบ๊อดโหมด 2 = 2 * ความถี่ออสซิลเลเตอร์ /64

ใน MCS-51 อัตราบ๊อดในโหมด 1 และ 3 ถูกกำหนดได้ด้วยอัตรา Overflow ที่เกิดขึ้น จากการกำหนดค่าในเรจิสเตอร์ TH1 ของตัวจับเวลา 1 ส่วนใน 8052 อัตราบ๊อดเหล่านี้สามารถคำนวณได้จากตัวจับเวลา 1 หรือ ตัวจับเวลา 2 หรือใช้ทั้งสองตัวโดยตัวหนึ่งสำหรับส่งและอีกตัวสำหรับรับ

การใช้ตัวจับเวลา 1 เป็นตัวสร้างอัตราบ๊อด

เมื่อใช้ตัวจับเวลา 1 เป็นตัวสร้างอัตราบ๊อด อัตราบ๊อดในโหมด 1 และ 3 จะถูกคำนวณด้วยอัตรา Overflow ที่เกิดขึ้นในตัวจับเวลา 1 และค่าบิตใน SMOD ซึ่งสูตรการคำนวณเป็นดังนี้

SMOD

อัตราบ๊อดในโหมด 1,3 = 2 *อัตรา Overflow ของตัวจับเวลา 1 /32

การอินเตอร์รัพต์ตัวจับเวลา 1 ควรที่จะดิสเอเบิ้ลในการใช้งานแบบนี้ ตัวจับเวลาในตัวมันเองสามารถที่จะถูกกำหนดให้ใช้เป็นตัวจับเวลาหรือตัวนับการทำงานในโหมด 3 ในการใช้งานในลักษณะนี้ มันจะถูกกำหนดให้ทำงานเป็นตัวจับเวลาในโหมดแบบบรรจุอัตโนมัติ (โดยตั้งให้ NIGH NIBBLE ของ TMOD = 0010B) ในกรณีนี้ อัตราบ๊อดคำนวณได้ดังสูตร

SMOD

อัตราบ๊อดในโหมด 1,3 = 2 /32 * ความถี่ออสซิลเลเตอร์/(12*(256-TH1))

ในการให้อัตราบ๊อดมีค่าต่ำมาก ก็สามารถที่จะทำได้โดยการตั้งตัวจับเวลา 1ให้สามารถรับการอินเตอร์รัพต์ได้ และกำหนดให้ตัวจับเวลาทำงานเป็น 16 บิต (โดยตั้งค่า HIGH NIBBLE ของ TMOD = 0001B) และใช้ตัวจับเวลา 1 ให้ทำการอินเตอร์รัพต์เมื่อเกิด

overflow และบรรจุค่า 16 บิตไปใหม่ ในกรณีนี้ต้องการอินเตอร์รัพต์ที่ตัวจับเวลา 1 จึงให้ IE.3 = 1

ในกรณีถ้าตัวจับเวลา 1 กำลังทำงานที่บิต C/T = 0 อัตราการนับเป็น 1/12 ของความถี่ออสซิลเลเตอร์ ถ้าตัวจับเวลาทำงานที่บิต C/T = 1 อัตราการนับจะใช้ความถี่ภายนอกที่ส่งเข้ามา ซึ่งจะมีความถี่สูงสุดที่จะใช้ได้คือ 1/24 ของความถี่ออสซิลเลเตอร์

รูปที่ 2.20 เป็นรายการอัตราบ๊อดที่ใช้ทั่วไป และจะคำนวณค่าต่าง ๆ ของการใช้ตัวจับเวลา 1 มีแสดงในตาราง

			TIMMER 1		
BAUD RATE	*fosc*	SMOD	C/T	MODE	RELOAD VALUE
MODE 0 MAX. 1MHz	12 MHz	X	X	X	X
MODE 2 MAX. 375K	12 MHz	1	X	X	X
MODES 1,3 62.5K	12 MHz	1	0	2	FFH
19.2K	11.059MHZ	1	0	2	FDH
9.6K	11.059MHZ	0	0	2	FDH
4.8K	11.059MHZ	0	0	2	FAH
2.4K	11.059MHZ	0	0	2	F4H
1.2K	11.059MHZ	0	0	2	E8H
137.5	11.059MHZ	0	0	2	1DH
110	6 MHZ	0	0	2	72H
110	12 MHZ	0	0	1	FEEBH

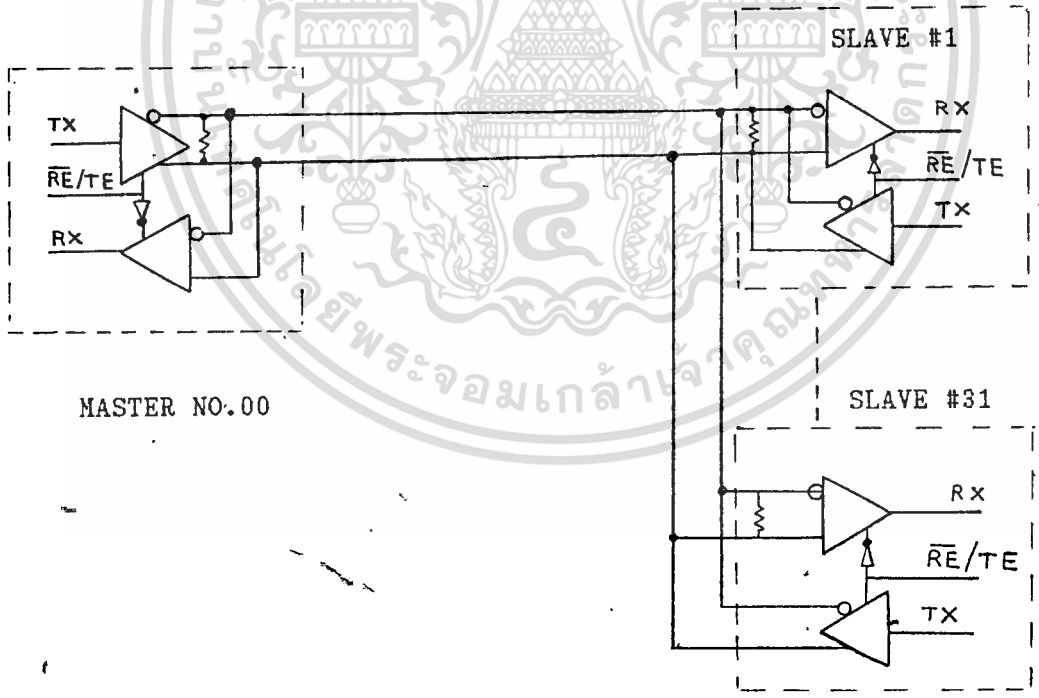
รูปที่ 2.20 เป็นรายการอัตราบ๊อดที่ใช้ตัวจับเวลา 1

บทที่ 3

การออกแบบอุปกรณ์ควบคุมและการควบคุมด้วยซอฟแวร์

แนวความคิดในการออกแบบ

ในการออกแบบระบบการสื่อสารตามมาตราฐาน RS-485 นี้ นับว่าเป็นการออกแบบซึ่งยุ่งยากพอสมควร เพราะการส่งข้อมูลตามมาตราฐานนี้ ใช้การส่งข้อมูลแบบ HAFT DUPLEX ซึ่งซอฟแวร์ที่จัดการระบบต้องออกแบบให้สัมพันธ์กัน โดยทำงานคนละเวลา และมาตราฐาน RS-485 นี้ใช้สำหรับส่งข้อมูลในระยะทางไกล ๆ ฉะนั้นอุปกรณ์ที่ใช้นี้ต้องมีความเที่ยงตรงและแน่นอนสูง เพราะถ้าเกิดการผิดพลาดในการส่งข้อมูลก็จะทำให้ข้อมูลใน MEMORY ของ MASTER และ SLAVE ต่างกัน ฉะนั้นอุปกรณ์ที่ใช้ในการส่งข้อมูลก็มีความสำคัญอย่างยิ่ง ในที่นี้เราอาจจะแยกระบบออกเป็น 2 ส่วนด้วยกันคือ HARD WARE และ SOFT WARE มาตราฐานการต่อใช้งานดูได้ดังรูป 3.1



รูปที่ 3.1

1.ส่วน HARD WARE ประกอบด้วย

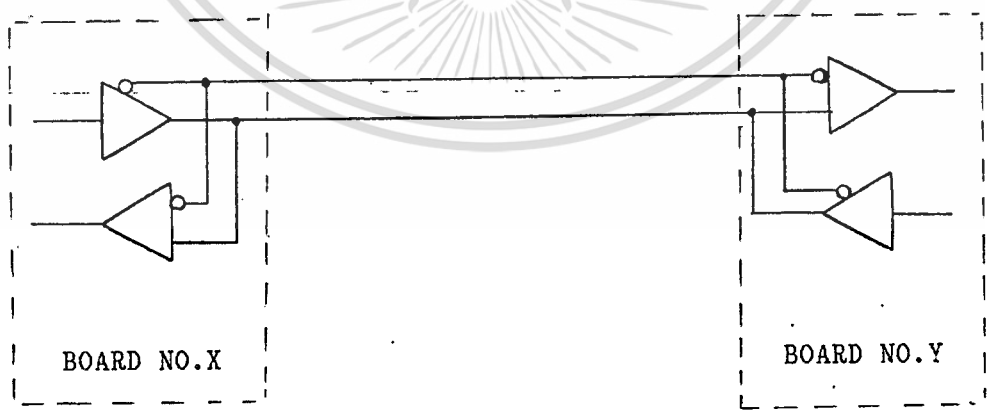
- CPU BOARD
- KEY BOARD LCD, DRIVER

CPU BOARD ประกอบด้วย

1. CPU เบอร์ MCS-80C31 X-TAL 11.0592 MHz
2. 2764 EPROM (PROGRAM MONITOR & KBYTEI)
3. RS-232 SERIAL PORT
4. 8255 PORT NO. E0E0H - E0E3H
5. POWER SUPPLY + 5v

การทำงาน

ใน CPU BOARD การทำหน้าที่เป็นตัวจัดการการทำงานทั้งหมดของเครื่องโดย CPU จะทำการอ่านข้
มูลมาจาก EPROM ซึ่งเป็น MONITOR PROGRAM ออกมา และกระทำตามคำสั่งที่เราเขียนไว้ โดยใน CP
BOARD จะประกอบด้วย 8255 PORT ซึ่งต่อกับ LCD และ CPU BOARD ยังประกอบด้วย RS-232 ซึ่งจ
ต้องนำมาเปลี่ยนให้เป็น RS-485 ใน BOARD ที่มี KEY BOARD ซึ่งจะมี IC เบอร์ 75173,75174 เป็นตั
เปลี่ยนโดยจะทำหน้าที่ส่งข้อมูลอยู่ในรูป CURRENT LOOP ข้อมูลจะถูกส่งไปตามสายและเข้าไปยัง BOARD รั
อีกตัวหนึ่ง ดังแสดงในรูป 3.2



รูปที่ 3.2

KEY BOARD DRIVER

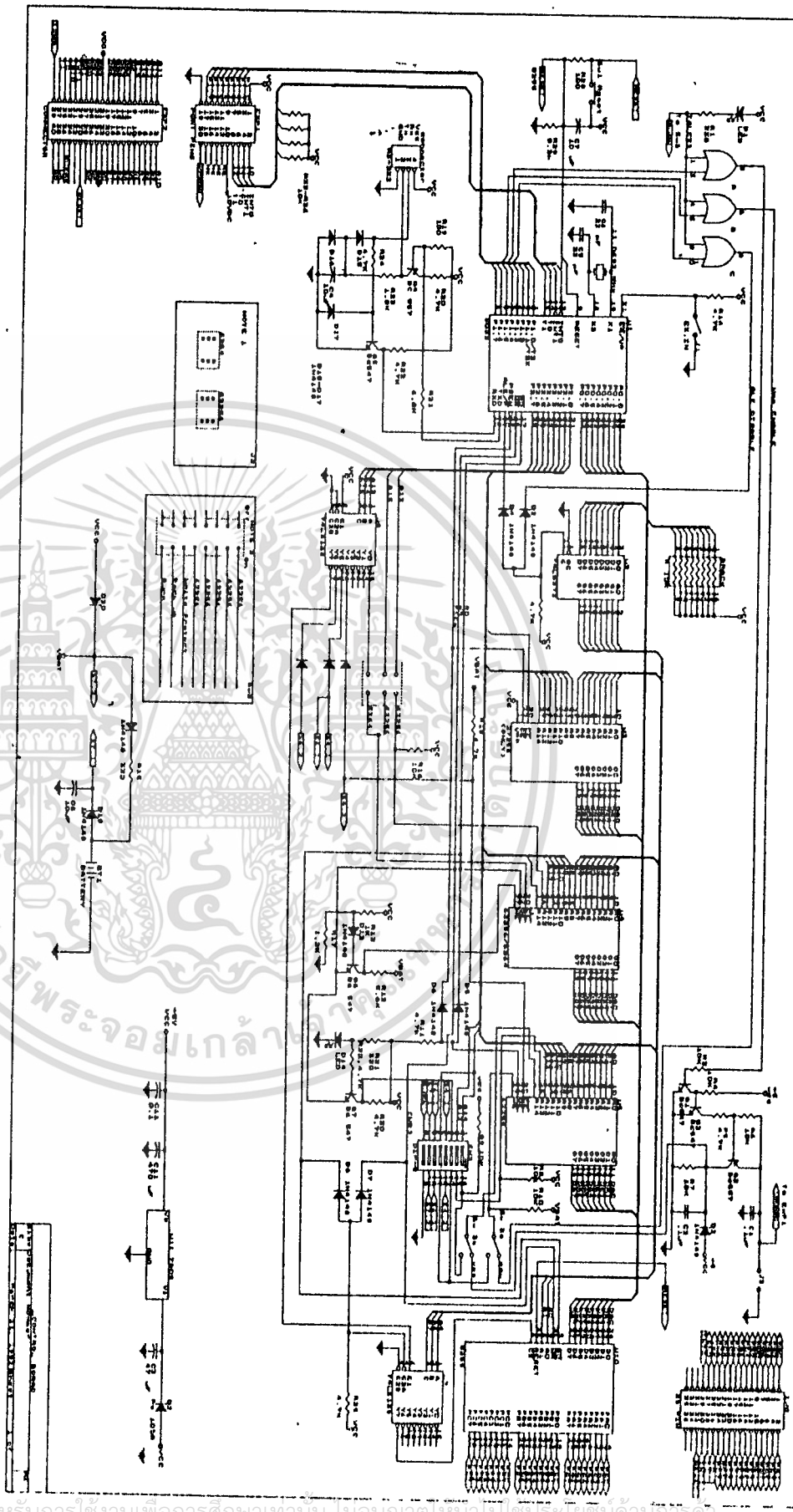
ส่วนของ KEY BOARD จะทำหน้าที่รับการกด KEY เพื่อไปทำงานตาม FUNCTION ที่กำหนดโดยใ
8255 PORT ทำงานร่วมกับ IC เบอร์ 74LS147 เพื่อใช้ในการ SCANKEY และรับ KEY เมื่อมีการก
KEY จึงทำให้ CPU รู้ว่าจะต้องไปกระทำที่ FUNCTION อะไร

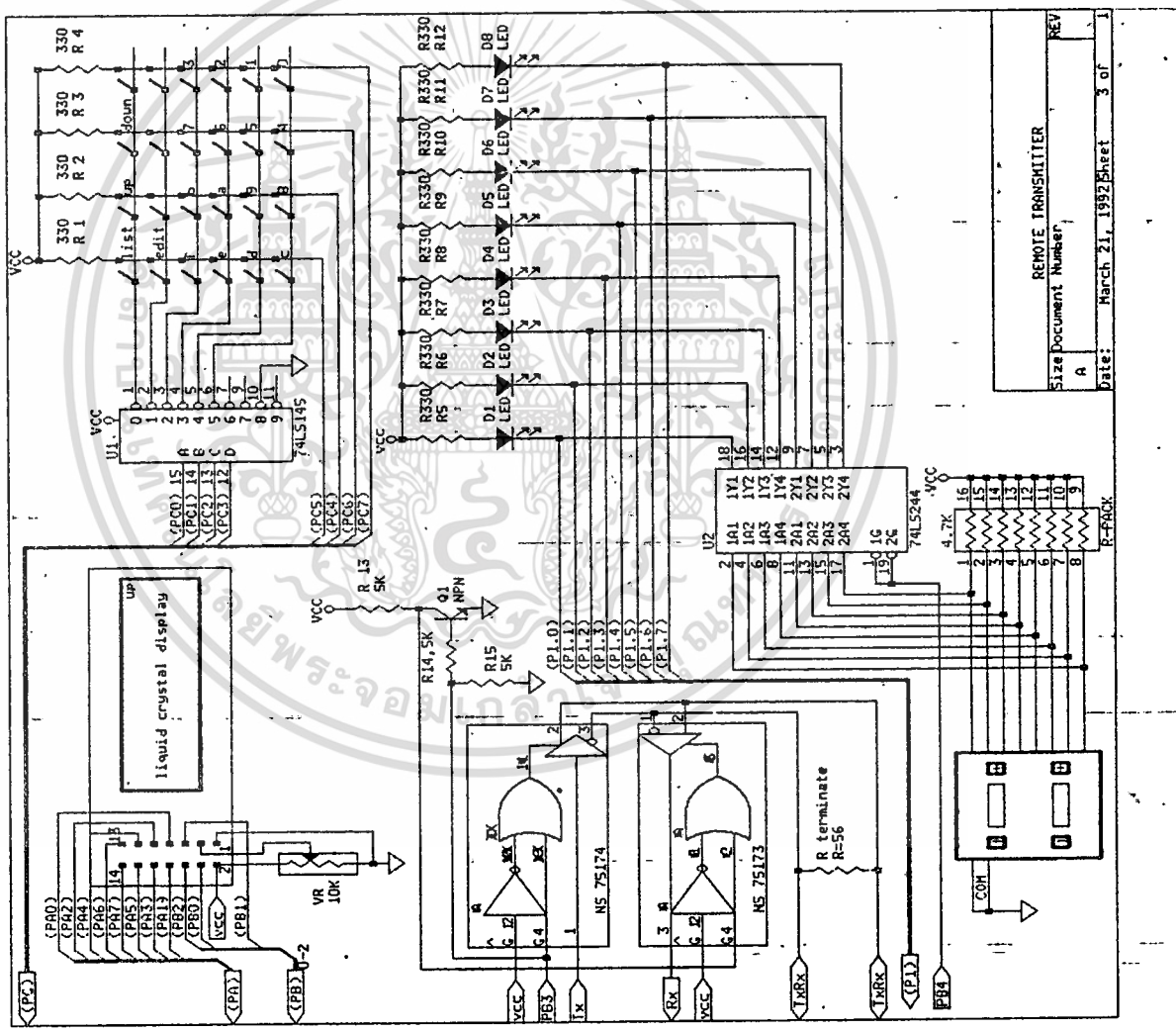
ส่วนของ DRIVER จำทำหน้าที่เปลี่ยน RS-232 จาก CPU BOARD มาเป็น RS-485 โดยใช้ I
เบอร์ 75172 และ 75174 ทำหน้าที่เป็นตัว TRANSMITTER และ RECIEVER ตามลำดับ

ตำแหน่งของ FUNTION KEY BOARD

C	D	E	F	EDIT CHAN	LIST CHAN
8	9	A	B	EDIT RAM	LIST RAM INC
4	5	6	7	SET BAUD RATE	NO. BOARD DEC
0	1	2	3	INSTALL	ENTER

KEY BOARD





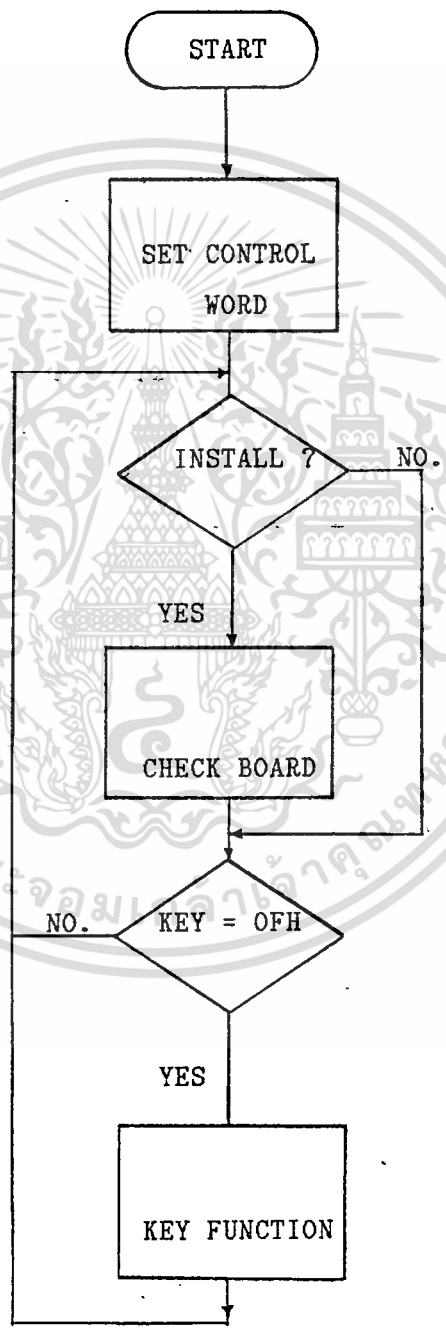
2. ส่วนของ SOFT WARE

เราจะใช้ PROTOCAL เป็นตัวกำหนดรูปแบบในการส่งหรือรับ เพื่อใช้ในการเขียนโปรแกรมใ
สามารถจัดการระบบต่าง ๆ ได้ เพื่อทำให้การส่งข้อมูลไม่มีการผิดพลาด ดังเสนอในตาราง

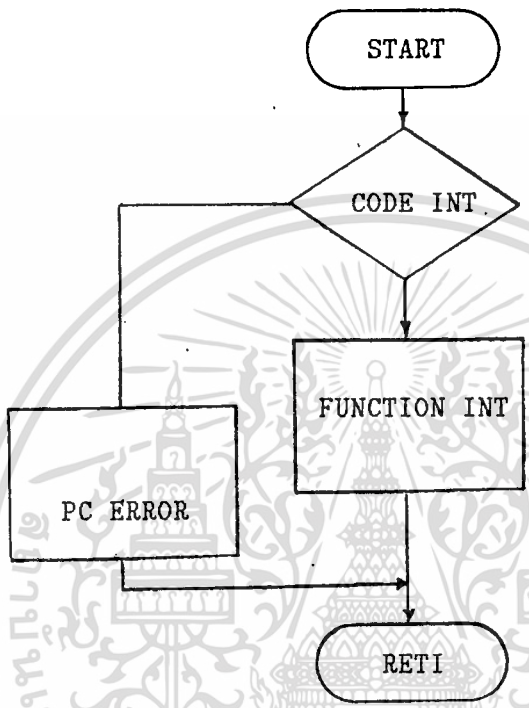
FUNCTION	COMMAND	FORMAT
SYSTEM RESET	40 H	40 H | SLAVE NO. | ODH
PC INSTALL	41 H	41 H | ODH
PC ERROR DATA RCADOUT/VERIFICATION	42 H	42 H | ODH
PC ERROR DATA TRASMITION	43 H	43 H | ODH
PC TRASMITTION	44 H	44 H | START | MASSAGE | STOP | ODH

* ODH = ENTER

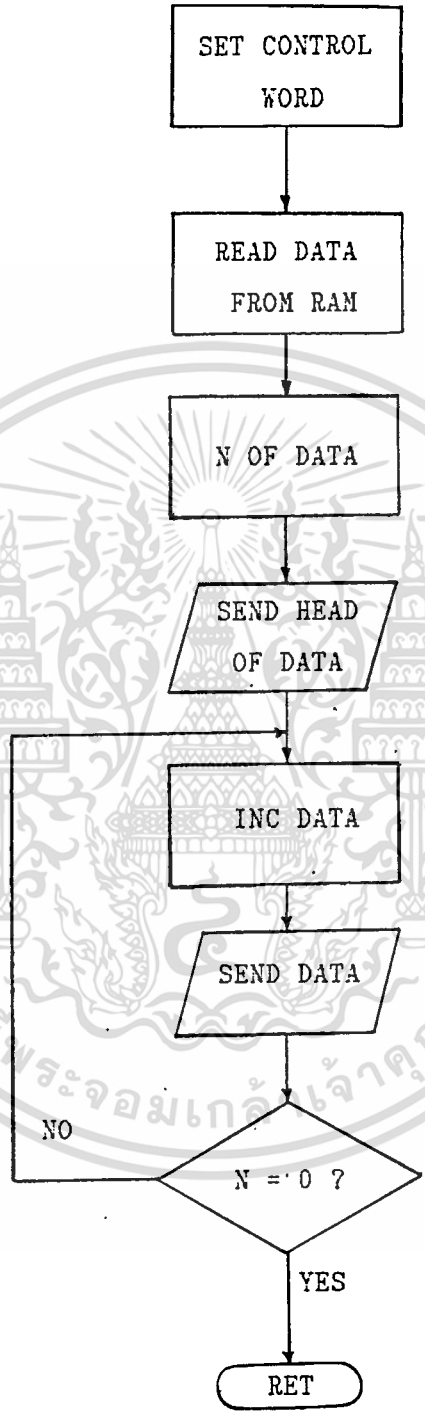
MAIN PROGRAM



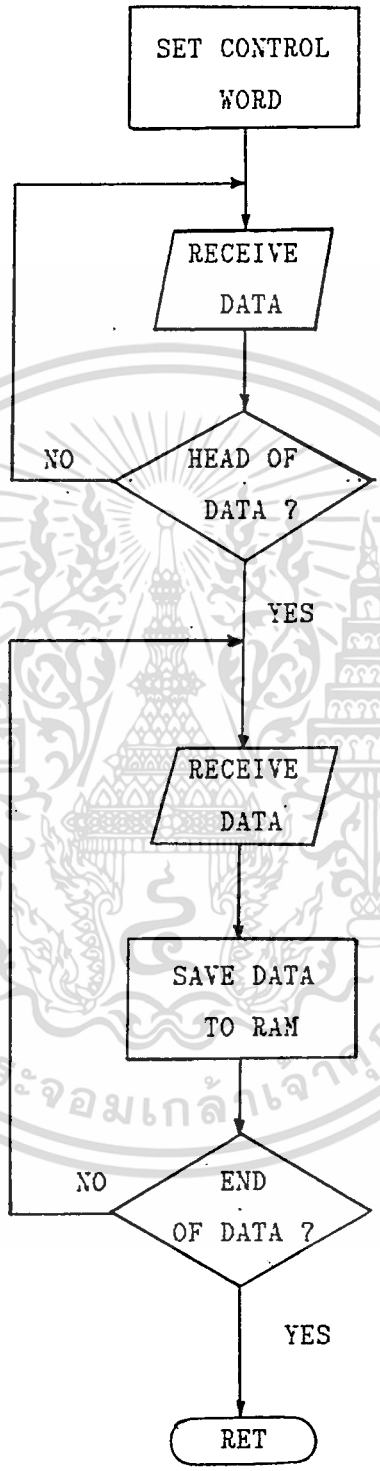
INTERUPT ROUTINE



FLOW CHART INTERUPT



FLOW CHART TRANSMIT ROUTAINE



FLOW CHART RECEIVE ROUTINE

บทที่ 4

FUNCTION และการใช้งาน

ในบทนี้ จะแสดงรายละเอียดต่าง ๆ เกี่ยวกับการติดตั้งอุปกรณ์ทางฮาร์ดแวร์และการใช้งานร่วมกับ
ซอฟแวร์ ในที่นี้จะกล่าวถึงเฉพาะ BOARD ที่มี KEY BOARD อยู่ด้วยและ CPU BOARD เป็น BOARD สำเร็จ
รูป คือ CP-32

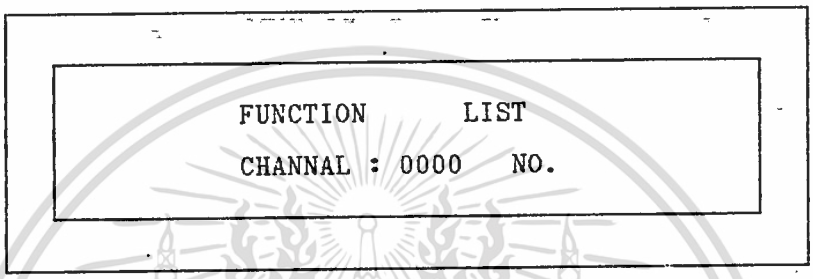
ตำแหน่งของ KEY BOARD และ FUNCTION การใช้งาน

C	D	E	F	EDIT CHAN	LIST CHAN
8	9	A	B	EDIT RAM	LIST RAM INC
4	5	6	7	SET BAUD RATE	NO. BOARD DEC
0	1	2	3	INSTALL	ENTER

รูปที่ 4.1 แสดงตำแหน่งของ KEY BOARD

หน้าที่ของ KEY ต่าง ๆ และการแสดงผล

LIST CHAN ทำหน้าที่ LIST ดูข้อมูลโดยสามารถ LIST ดูข้อมูลที่เก็บไว้ได้สำหรับการแสดงผลเมื่อ
ทำการกด KEY นี้ จะแสดงผลที่ LCD ดังนี้

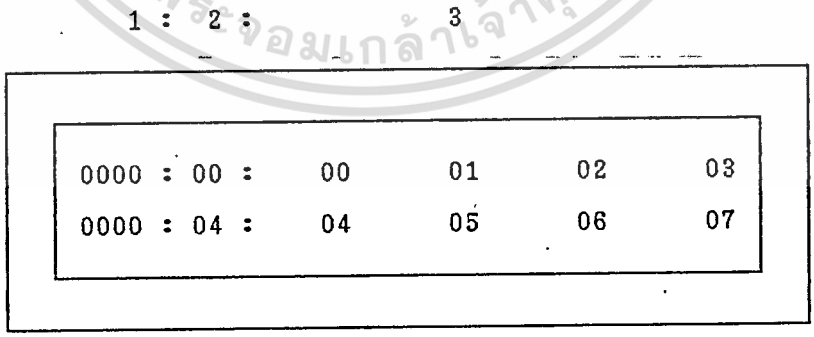


รูปที่ 4.2

สำหรับการ LIST ข้อมูลด้วยวิธีนี้จะมีอยู่ 2 กรณี

1) ดูทีละ CHANNAL (8 BYTE)

เมื่อกด KEY LIST แล้วได้ตามรูป 4.2 เราต้องป้อนตัวเลข จาก KEY ตัวเลข 16 KEY ดูตามรูป 4.1 โดย 1 CHANNAL จะมีข้อมูล ถึง 8 BYTE และเรากด KEY ENTER 2 ครั้งจะเกิดการแสดงผลดังรูป



รูปที่ 4.3

โดย

ส่วนที่ 1 หมายถึง CHANNAL ที่ต้องการ LIST

ส่วนที่ 2 หมายถึง NUMBER ที่ต้องการ LIST

ส่วนที่ 3 หมายถึง ข้อมูลที่อยู่ที่ CHANNAL นั้นและ NUMBER นั้น ๆ

2) ดูทีละ NUMBER (1 BYTE BINARY)

เมื่อกด KEY LIST แล้วได้ตามรูป 4.2 และเมื่อป้อนตัวเลขเพื่อระบุ CHANNAL ที่ต้องการ LIST และกด ENTER 1 ครั้ง CURSOR จะเลื่อนมาอยู่ที่ NO. แล้วป้อนค่าตัวเลข NO. ที่ต้องการตรวจดู ตัวอย่างเช่น ถ้าต้องการดูที่ NO. 4 ตามรูป 4.3 จะได้การแสดงผลดังนี้

```
CHANNAL : 000  NO : 0
BINARY  : 00000100
```

รูปที่ 4.4

EDIT CHAN ทำหน้าที่แก้ไขข้อมูล โดย จะแก้ไขข้อมูลเป็น CHANNAL และการแสดงผลจะเป็นดังรูป

```
    1 :  2 :          3
0000 : 00 :   00   01   02   03
0005 : 00 :   04   05   06   07
```

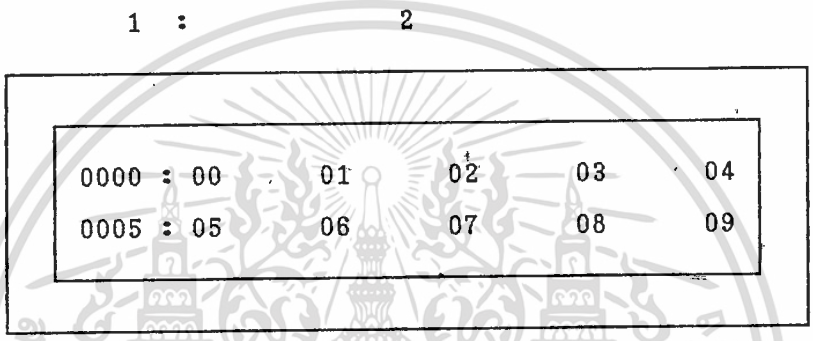
รูปที่ 4.5

ส่วนที่ 1 หมายถึง CHANAL ที่ต้องการ EDIT

ส่วนที่ 2 หมายถึง NUMBER ที่ต้องการ EDIT

ส่วนที่ 3 หมายถึง ข้อมูลที่ต้องการแก้ไขในตอนเริ่มแรก CURSOR จะปรากฏอยู่ที่ NO.

LIST ทำหน้าที่ LIST ข้อมูลเช่นเดียวกับ LIST CHAN แต่ต่างกันตรงที่ในการ LIST จะ LIST ตามตำ
แหน่งของ RAM และสามารถดูข้อมูลได้ทีละ 10 BYTE

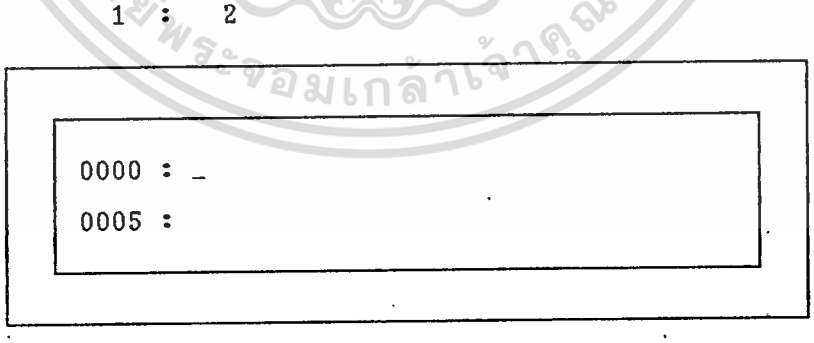


รูปที่ 4.6

ส่วนที่ 1 หมายถึง ตำแหน่งของ RAM ที่จะ LIST

ส่วนที่ 2 หมายถึง ข้อมูลที่ตำแหน่งของ RAM นั้น ๆ

EDIT ทำหน้าที่แก้ไขข้อมูล เช่นเดียวกับ EDIT CHAN ต่างกันตรงที่ในการ EDIT เราจะ EDIT ตามตำ
แหน่งของ RAM การแสดงผลดังรูป

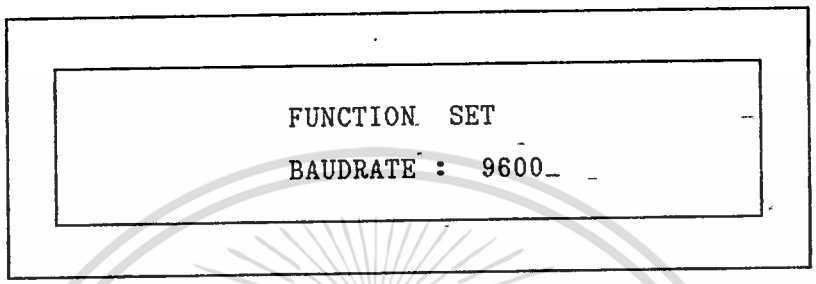


รูปที่ 4.7

ส่วนที่ 1 หมายถึง ตำแหน่งของ ROM ที่จะ EDIT

ส่วนที่ 2 หมายถึง ข้อมูลที่จะทำการแก้ไขในตำแหน่งของ RAM นั้น ๆ

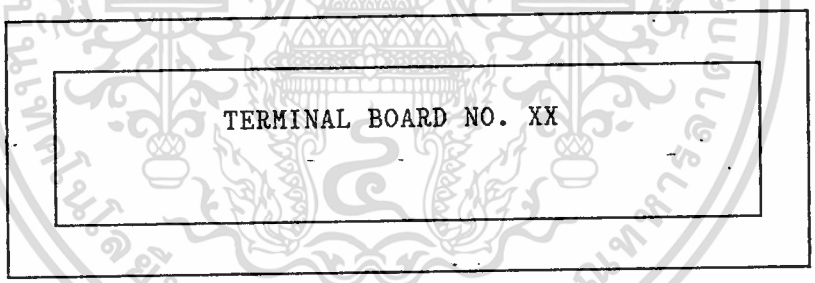
BAUD RATE เป็นฟังก์ชั่น การ SET BAUD RATE ที่จะทำการ COMMUNICATE โดยการ SET BAUD RATE สามารถ SET ได้ 4 ค่า คือ 9600, 4800, 2400, 1200 และการแสดงผลเป็นดังนี้



รูปที่ 4.8

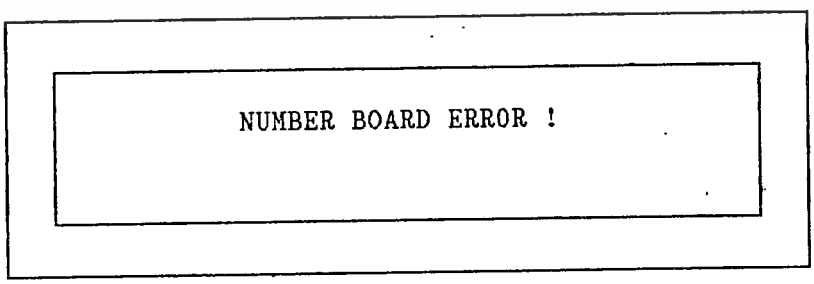
เมื่อกด KEY INC หรือ DEC ค่าที่ SET BAUD RATE จะเปลี่ยนไปตามค่าที่กำหนดไว้ข้างต้น

NO. BOARD เป็นฟังก์ชั่นใช้ในการตรวจดู ว่า BOARD ที่ใช้ SET NUMBER ไว้เกินกว่าที่กำหนดหรือไม่ เมื่อทำการกด THRUM WHEEL จะแสดงดังนี้



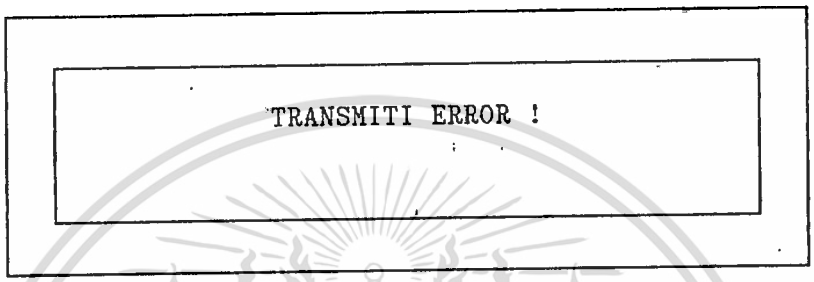
รูปที่ 4.9

ถ้าอักษร XX เกินกว่า 31 BOARD จะแสดงค่าออกมาว่า

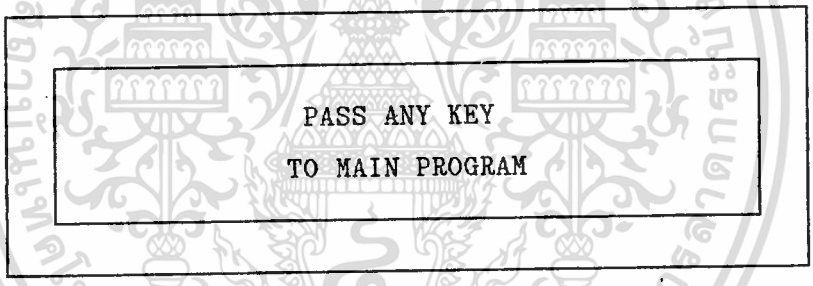


รูปที่ 4.10

INSTALL ทำหน้าที่ติดตั้ง BOARD เพื่อทำการ จัดแบ่ง MEMORY หรือทำการ CHECK BOARD เมื่อมีการ INSTALL แล้ว FUNCTION นี้จะไม่มีการแสดงผลใด ๆ แต่จะแสดงผลเมื่อมีการส่งข้อมูล ERROR ขึ้น ดังรูป



รูปที่ 4.11



รูปที่ 4.12

ENTER เป็นตำแหน่งที่จะเข้าหรือออก FUNCTION ใด ๆ ของ BOARD การใช้งานของระบบ

การทำงานของระบบ

เมื่อมีการต่อ BOARD เข้าดังรูปไม่ว่าจะต่อ SLAVE กี่ตัวก็ตาม ต้องมีการ INSTALL ระบบก่อนโดย เรากำหนดให้ BOARD NO.00 เป็น MASTER และ BOARD ที่เหลือจะเป็น SLAVE หมด ในการ INSTALL ถ้ากรณีต่อระบบเข้าไปโดยไม่มีการ INSTALL การ WRITE หรือแก้ไขข้อมูลในหน่วยความจำจะสามารถ แก้ไขได้ตั้งแต่ CHANNEL 0000H - 0FFFH หรือ ADDRESS 0000H - 7FFFH และจะสามารถ READ DATA ได้ตั้งแต่ CHANNEL 0000 -13FFH หรือ ADDRESS 0000H ถึง ADDRESS 9FFFH แต่เมื่อมีการ INSTALL ระบบ (ต้อง SET BAUD RATE ให้เท่ากัน) MASTER ก็จะเป็นตัวจัดหน่วยความจำว่าให้ SLAVE NO. อะไร สามารถแก้ไขข้อมูลได้ในส่วนไหน ซึ่งก็หมายถึงการแบ่ง MEMORY กันนั้นเอง แต่ทุก BOARD จะสามารถอ่านข้อมูลได้ทั้งหมด ไม่ว่า MEMORY ส่วนนั้นจะไม่สามารถแก้ไขข้อมูลได้ก็ตามและหน่วยความจำที่ใช้ทั้งหมด = 32 K BYTE ซึ่งการแบ่ง MEMORY ก็ขึ้นอยู่กับจำนวน SLAVE ที่ต่อกับตัว MASTER ถ้าจำนวน SLAVE มีจำนวนมาก MEMORY ที่ จัดให้แต่ละ SLAVE ก็จะมีค่าน้อยแต่ถ้ามีจำนวน SLAVE น้อย จำนวน MEMORY ที่จัดให้ SLAVE ทำการแก้ไว้ก็จะมีค่ามากขึ้นคือแสดงไว้ในตาราง ซึ่งจะบอกถึงจำนวน SLAVE ที่มาทำการต่อ และ MEMORY ที่ MASTER ทำการจัดสรรให้เมื่อมีการ INSTALL ระบบ ในที่นี้กำหนดให้ 1 UNIT = 1 K BYTE เมื่อเป็นการง่ายแก่การอธิบายในตารางซึ่งเมื่อดูแล้วจะพบว่าถ้าระบบถูกต่อจนครบทุกตัวแล้วทุกตัวจะได้ MEMORY ที่สามารถแก้ไขได้ BOARD ละ 1 K BYTE พอดี แต่ถ้าต่อน้อยกว่าก็จะกำหนดดังในตาราง

NO. BOARD	17 TO 32	9 TO 16	5 TO 8	3 TO 4	2
MEMORY (UNIT)					
1	MASTER				
2	#1	MASTER			
3	#2		MASTER		
4	#3	#1		MASTER	
5	#4				
6	#5	#2			
7	#6				
8	#7	#3	#1		MASTER
9	#8				
10	#9	#4			
11	#10				
12	#11	#5	#2	#1	
13	#12				
14	#13	#6			
15	#14				

ตารางจัดแบ่ง MEMORY เมื่อมี SLAVE จำนวนต่าง ๆ กัน

NO. BOARD	17 TO 32	9 TO 16	5 TO 8	3 TO 4	2
MEMORY (UNIT)					
16	#15	#7	#3		
17	#16				
18	#17	#8			
19	#18				
20	#19	#9	#4	#2	
21	#20				
22	#21	#10			
23	#22				
24	#23	#11	#5		#1
25	#24				
26	#25	#12			
27	#26				
28	#27	#13	#6	#3	
29	#28				
30	#29	#14			
31	#30				
32	#31	#15	#7		

ตารางจัดแบ่ง MEMORY เมื่อมี SLAVE จำนวนต่าง ๆ กัน

สำหรับ SLAVE เมื่อทำการ INSTALL ไว้หลัง เมื่อมีการถอดออกจากระบบจะทำให้ การส่งข้อมู
ผิดพลาด MASTER จะทำการแจ้งให้ผู้ใช้ทราบทันทีว่าระบบถูกถอดออกหรือไม่สามารถทำการรับส่งข้อมูลกันไ
MASTER จะแจ้งไปถึงที่จนกว่าจะมีการตอบรับจาก SLAVE ที่ติดตั้งหรือทำการ RESET เพื่อติดตั้งระบบใหม่

บทที่ 5
บทสรุปและข้อเสนอแนะ

บทสรุป

จะเห็นว่าการส่งตามมาตรฐาน RS - 485 ที่ได้ทำการพัฒนา เป็นการสื่อสารได้ดีในการส่งข้อมูลระยะไกล ๆ โดยที่ข้อมูลมีโอกาสผิดพลาดได้น้อยมาก เมื่อเทียบกับมาตรฐานอื่น เช่น RS - 232 แต่ข้อเสียของการส่งตามมาตรฐาน RS - 485 ก็คือการส่งและรับในเวลาเดียวกันไม่ได้ แต่ถ้าไม่มีการค่านึงถึงเวลามากนัก การส่งมาตรฐาน RS - 485 นับว่าเป็นการส่งที่ดี และมีประสิทธิภาพมาก ดังจะเห็นได้จากการนิยมใช้กันอย่างกว้างขวางในโรงงานอุตสาหกรรม เช่น PLC เป็นต้น

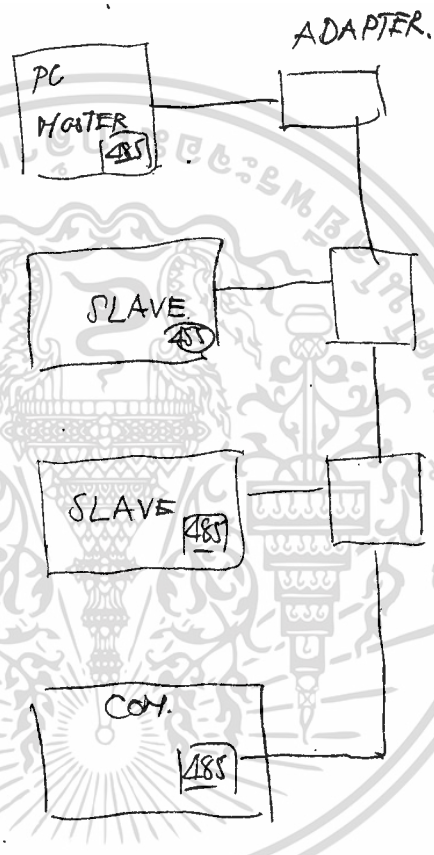
ข้อเสนอแนะ

ในโรงงานอุตสาหกรรม ที่ใช้ระบบควบคุมการทำงานของ PROCESS โดยมีหน่วยประมวลผล และส่งสัญญาณไปควบคุม PROCESS นั้น ๆ เพื่อทำให้การทำงานของ PROCESS แต่ละ PROCESS ต่อเนื่องกันทั้งระบบ จะทำให้ประสิทธิภาพในการทำงานของระบบดีขึ้น

ในการรับส่งข้อมูลระหว่างหน่วยประมวลผลกลางกับ PROCESS ต่าง ๆ ควรจะใช้มาตรฐาน RS - 485 เพราะเมื่อเทียบกับมาตรฐานอื่นแล้ว จะเห็นว่ามาตรฐาน RS - 485 จะให้ประสิทธิภาพในการรับส่งข้อมูลดีกว่ามาตรฐานอื่น ๆ

ภาคผนวกที่ 1

ในภาคผนวกที่ 1 นี้ได้แสดงโปรแกรมควบคุมการติดต่อสื่อสาร ทั้ง มาสเตอร์ (MASTER) และสเลฟ (SLAVE) ไว้โดยละเอียด



```
;17/3/92
;*********************************************************
;
;THESE ARE EXAMPLE SUBROUTINES FOR THE 8051 MICRO-
;CONTROLLER FAMILY.  THEY ARE TAKEN FROM THE 1982
;INTEL MICROCONTROLLER USER'S MANUAL ON PAGES 9-1
;& 9-2.  AFTER ONLY FORMAT MODIFICATIONS FOR CROSS-16,
;THEY ARE PASSED ON AS EXAMPLES, FREE OF CHARGE, BY
;UNIVERSAL CROSS-ASSEMBLERS.

;ALSO INCLUDED, IS A LIST OF THE REGISTER AND BIT NAMES
;WITH THEIR CORRESPONDING ADDRESSES.  THESE MAY
;BE SPECIFIED AS PART OF THE ASSEMBLER CODE, OR
;MOVED TO THE END OF THE 8051 TABLE FOR GREATER
;TRANSPARENCY.

        CPU     "8051.TBL"
        HOF     "BIN8"
;
;*********************************************************
;
;MCS-51 INTERNAL REGISTERS
;
B:      EQU     0F0H    ;B REGISTER
ACC:    EQU     0E0H    ;ACCUMULATOR
PSW:    EQU     0D0H    ;PROGRAM STATUS WORD
IPC:    EQU     0B8H    ;INTERRUPT PRIORITY
P3:     EQU     0B0H    ;PORT 3
IEC:    EQU     0A8H    ;INTERRUPT ENABLE
P2:     EQU     0A0H    ;PORT 2
SBUF:   EQU     99H     ;SEND BUFFER
SCON:   EQU     98H     ;SERIAL CONTROL
P1:     EQU     90H     ;PORT 1
TH1:    EQU     8DH     ;TIMER 1 HIGH
TH0:    EQU     8CH     ;TIMER 0 HIGH
TL1:    EQU     8BH     ;TIMER 1 LOW
TL0:    EQU     8AH     ;TIMER 0 LOW
TMOD:   EQU     89H     ;TIMER MODE
TCON:   EQU     88H     ;TIMER CONTROL
PCON:   EQU     87H     ;POWER CONTROL REGISTER
DPH:    EQU     83H     ;DATA POINTER HIGH
DPL:    EQU     82H     ;DATA POINTER LOW
SP:     EQU     81H     ;STACK POINTER
P0:     EQU     80H     ;PORT 0
;
;MCS-51 INTERNAL BIT ADDRESSES
;
CY:     EQU     0D7H    ;CARRY FLAG
AC:     EQU     0D6H    ;AUXILIARY-CARRY FLAG
F0:     EQU     0D5H    ;USER FLAG 0
RS1:    EQU     0D4H    ;REGISTER SELECT MSB
RS0:    EQU     0D3H    ;REGISTER SELECT LSB
OV:     EQU     0D2H    ;OVERFLOW FLAG
P:      EQU     0D0H    ;PARITY FLAG
PS:     EQU     0BCH    ;PRIORITY SERIAL PORT
```

```
PT1:      EQU       0BBH      ;PRIORITY TIMER 1
PX1:      EQU       0BAH      ;PRIORITY EXTERNAL 1
PT0:      EQU       0B9H      ;PRIORITY TIMER 0
PX0:      EQU       0B8H      ;PRIORITY EXTERNAL 0
EA:       EQU       0AFH      ;ENABLE ALL INTERRUPT
ES:       EQU       0ACH      ;ENABLE SERIAL INTERRUPT
ET1:      EQU       0ABH      ;ENABLE TIMER 1 INTERRUPT
EX1:      EQU       0AAH      ;ENABLE EXTERNAL 1 INTERR
ET0:      EQU       0A9H      ;ENABLE TIMER 0 INTERRUPT
EX0:      EQU       0A8H      ;ENABLE EXTERNAL 0 INTERR
SM0:      EQU       09FH      ;SERIAL MODE 0
SM1:      EQU       09EH      ;SERIAL MODE 1
SM2:      EQU       09DH      ;SERIAL MODE 2
REN:      EQU       09CH      ;SERIAL RECEPTION ENABLE
TB8:      EQU       09BH      ;TRANSMITT BIT 8
RB8:      EQU       09AH      ;RECEIVE BIT 8
TI:       EQU       099H      ;TRANSMIT INTERRUPT FLAG
RI:       EQU       098H      ;RECEIVE INTERRUPT FLAG
TF1:      EQU       08FH      ;TIMER 1 OVERFLOW FLAG
TR1:      EQU       08EH      ;TIMER 1 RUN CONTROL BIT
TF0:      EQU       08DH      ;TIMER 0 OVERFLOW FLAG
TR0:      EQU       08CH      ;TIMER 0 RUN CONTROL BIT
IE1:      EQU       08BH      ;EXT INTERR. 1 EDGE FLAG
IT1:      EQU       08AH      ;EXT INTERR. 1 TYPE FLAG
IE0:      EQU       089H      ;EXT INTERR. 0 EDGE FLAG
IT0:      EQU       088H      ;EXT INTERR. 0 TYPE FLAG
ACC0:     EQU       0E0H      ;ACCUMULATOR BIT 0
ACC1:     EQU       0E1H      ;ACCUMULATOR BIT 1
ACC2:     EQU       0E2H      ;ACCUMULATOR BIT 2
ACC3:     EQU       0E3H      ;ACCUMULATOR BIT 3
ACC4:     EQU       0E4H      ;ACCUMULATOR BIT 4
ACC5:     EQU       0E5H      ;ACCUMULATOR BIT 5
ACC6:     EQU       0E6H      ;ACCUMULATOR BIT 6
ACC7:     EQU       0E7H      ;ACCUMULATOR BIT 7
;*****************************************************

pdata:          equ     0e0e0h
psign:          equ     0e0e1h
pscan:          equ     0e0e2h
pctl:           equ     0e0e3h
shiftl:         equ     10h
shiftr:         equ     14h
pushl:          equ     18h
pushr:          equ     1ch
trans_l:        equ     2eh
trans_h:        equ     2fh
ascl:           equ     30h     ;ascii  low
asch:           equ     31h     ;ascii  high
n_chr:          equ     32h
func:           equ     33h     ;function  set LCD
addl:           equ     34h
addh:           equ     35h
start:          equ     36h     ;position  dd ram
n_list:         equ     37h     ;number of char  function list
edd:            equ     38h
```

```
buff1:          equ     39h
number:         equ     3ah
posit1:         equ     3bh     ;position 4 digit
posit2:         equ     3ch     ;position 4 digit
baud:           equ     3dh     ;set baud rate
buffh:          equ     3fh
d_buff:         equ     40h
numb:           equ     41h
start1:         equ     42h
stall:          equ     43h
sub_n:          equ     44h
dela:           equ     45h
s_addh:         equ     46h
s_addl:         equ     47h
check_b:        equ     2bh
command:        equ     2ch
code:           equ     2dh     ;code interupt serial
buff_dis:       equ     8700h
trandat:        equ     9000h
trandat1:       equ     trandat+1
trandat2:       equ     trandat+2
st_add1:        equ     trandat+3
st_add2:        equ     trandat+4
stp_add1:       equ     trandat+5
stp_add2:       equ     trandat+6
buff_b:         equ     trandat+7
buffer1:        equ     trandat+8
buffer2:        equ     trandat+9
buffer3:        equ     trandat+10
buffer4:        equ     trandat+11
add_numb:       equ     trandat+12
stc_add1:       equ     trandat+13
stc_add2:       equ     trandat+14
stpc_add1:      equ     trandat+15
stpc_add2:      equ     trandat+16
const1:         equ     4000h
const2:         equ     2000h
const3:         equ     1000h
const4:         equ     0800h
const5:         equ     0400h
const6:         equ     0800h
const7:         equ     0400h
const8:         equ     0200h
const9:         equ     0100h
const10:        equ     0080h
                ;*****************************
                ;    CODE KEYBOARD
                ;*****************************
k1:             equ     0cah
k2:             equ     0cbh
k3:             equ     0cch
k4:             equ     0cdh
k5:             equ     0ceh
k6:             equ     0cfh
k7:             equ     0dah
```

```
k8:             equ     0dbh
k9:             equ     0dch
k10:            equ     0ddh
k11:            equ     0deh
k12:            equ     0dfh
k13:            equ     0aah
k14:            equ     0abh
k15:            equ     0ach
k16:            equ     0adh
k17:            equ     0aeh
k18:            equ     0afh
k19:            equ     06ah
k20:            equ     06bh
k21:            equ     06ch
k22:            equ     06dh
k23:            equ     06eh
k24:            equ     06fh
;*****************************
;   MAIN    PROGRAM
;*****************************
                org     0000h
                sjmp    0040h
                org     0023h
                clr     es
                mov     p1,#7fh
                ljmp    try   (110)
                org     0040h
                mov      sp,#50h
                lcall   delay
                lcall   c_ram1
                mov     dptr,#8500h
                mov     a,#0dh
                movx    @dptr,a
                lcall   set_e
                lcall   lcd
                lcall   initlcd
                lcall   clr_f
                lcall   test1
                mov     dela,#1
                mov     check_b,#00
                mov     baud,#0fdh   ;set baud rate=9600
                mov     stall,#0
                mov     scon,#0f0h
                mov     tmod,#20h
                mov     th1,baud
                setb     sm2
                setb     tr1
                setb     ea
                setb     es
                lcall   numb00
                setb     es
                lcall   kout
                lcall   clr_f
main:           lcall   main2
                mov     a,#0                ;t
```

```
                cjne    a,number,main1      ;t
                cjne    a,stall,main1       ;t
                lcall   b_con
main1:          lcall   t1
                setb    es
;                cjne    r3,#0f0h,main
                lcall   numb00
                mov     a,#0                ;t
                cjne    a,number,main1      ;t
                cjne    a,stall,main1       ;t
                lcall   b_con
                sjmp    main1
main2:          lcall   reset
                lcall   reset_char
                ret
        ;*******************************
t1:             lcall    sck                ;SCAN KEY BOARD
list_k:         cjne     r3,#k12,list_chan  ;key list?
                lcall   gen
                lcall   clr_f
                lcall    list_dat
                lcall   gen1
                ret
list_chan:      cjne     r3,#k6,no_board    ;key list channel?
                lcall   gen
                mov     posit1,#4ah
                mov     posit2,#4dh
                lcall    list_dat1
                lcall   gen1
                ret
no_board:       cjne     r3,#k18,edit       ;key list channel binary?
                lcall   gen
                lcall    numbb     ;chan_bin
                lcall   gen1
                ret
edit:           cjne    r3,#k11,edit_chn        ;key edit?
                lcall   gen
                lcall   clr_f
                lcall   edit_dat
                lcall   gen1
                ret
edit_chn:       cjne    r3,#k5,baudrate   ;key edit chan?
                lcall   gen
                mov     posit1,#4ah
                mov     posit2,#4dh
                lcall   e_chan
                lcall   gen1
                ret
baudrate:       cjne    r3,#k17,transs      ;key baudrate
                lcall   gen
                lcall   rate
                lcall   gen1
                ret
transs:         cjne    r3,#k23,t11
                mov     a,#0
```

```
                    cjne    a,number,t11
                    lcall   gen
                    lcall   numb00
                    mov     a,#0h
                    cjne    a,number,transs1
                    lcall   inst
                    mov     check_b,#0h
transs1:            lcall   kout
                    lcall   gen1
                    ret
;trans:             cjne    r3,#k17,clr_ram
;                   lcall   gen
;                   lcall   trans_dat
;                   lcall   gen1
;                   ret
;clr_ram:           cjne    r3,#k23,t11
;                   lcall   gen
;                   lcall   clear0
;                   lcall   gen1
;                   ret
;edit_chr:          cjne    r3,#k11,asc_list
;                   lcall   gen
;                   lcall   ascii_dat
;                   lcall   gen1
;                   ret
;asc_list:          cjne    r3,#k12,t11
;                   lcall   gen
;                   lcall   list_page
;                   lcall   gen1
;                   ret
out_t:              lcall   error
                    lcall   any_k
t11:                ret
gen:                mov     number,#0aah
                    lcall   wait            ;t
                    lcall   wait            ;t
                    lcall   wait            ;t
                    lcall   wait            ;t
                    lcall   wait            ;t
                    lcall   wait
                    lcall   wait            ;t
                    lcall   wait            ;t
                    lcall   wait
                    ret
gen1:               lcall   main2
                    mov     command,#45h ;t
                    push    iec
                    push    dph
                    push    dpl
                    push    acc
                    clr     es
                    clr     et1
                    setb    ti
                    lcall    ee
                    clr     ri
```

```
                clr     ti
                pop     acc
                pop     dpl
                pop     dph
                pop     iec
                ret
ee:             mov     dptr,#psign
                mov     a,#08h
                movx    @dptr,a
                lcall   comm
                mov     a,#0ddh
                jnb     ti,$
                clr     ti
                mov     sbuf,a
                jnb     ti,$
                clr     ti
                mov     dptr,#psign
                mov     a,#08h
                movx    @dptr,a
                mov     dptr,#psign
                mov     a,#0h
                movx    @dptr,a
                clr     ti
                setb    es
                ret
              ;*****************************
              ;  TEST RAM FUNCTION
              ;*****************************
test1:          mov     dptr,#data33
                mov     start,#0
                mov     n_chr,#20
                lcall   list
                mov     dptr,#data32
                mov     start,#48h
                mov     n_chr,#4
                lcall   list
                mov     start,#60h
                lcall   goto
                mov     dptr,#0000h
                mov     r0,#00h
                mov     r1,#0a0h
                mov     r4,#0
test2:          movx    a,@dptr
                mov     number,a
                lcall   test3
                inc     dptr
;                lcall   waittt
;                push    4
;                mov     a,r0
;                lcall   h_2asc
;                mov     start,#48h
;                lcall   goto
;                lcall   inc_add1
;                pop     4
                djnz    r0,test2
```

```
                inc     r4
                lcall   inc_add
                djnz    r1,test2
                mov     dptr,#data34
                mov     start,#40h
                mov     n_chr,#20
                lcall   list
vat1:            mov     r1,#8
vat:            lcall   delay000
                djnz    r1,vat
                ret
test3:        ;  lcall    waittt
                mov     a,#0aah
                movx    @dptr,a
                mov     a,#0
                movx    a,@dptr
                cjne    a,#0aah,r_err
                mov     a,number
                movx    @dptr,a
                ret
inc_add:        mov     a,r4
                lcall   h_2asc
                mov     start,#48h
                lcall   goto
inc_add1:       mov     r4,asch
                lcall   wrbyte
                mov     r4,ascl
                lcall   wrbyte
                mov     r4,a
                mov     start,#60h
                lcall   goto
                ret
r_err:          push    dph
                push    dpl
                mov     dptr,#data35
                mov     start,#40h
                mov     n_chr,#20
                lcall   list
                sjmp    $
                pop     dpl
                pop     dph
                ret
waittt:         push    0
                mov     r0,#8h
                djnz    r0,$
                pop     0
                ret
            ;*******************************
            ;     RESET FUNCTION
            ;*******************************
reset:          mov      addh,#0
                mov      addl,#0
                mov      edd,#0
                lcall    lcd            ;init lcd
                lcall    initlcd
```

```
                lcall   delay
                lcall   initlcd
                lcall   delay
                lcall   initlcd
                lcall   delay
                ret
c_ram1:         mov     r2,#80h
                mov     r3,#00
                mov     dptr,#0000h
c_ram:          mov     a,#0
                movx    @dptr,a
                inc     dptr
                djnz    r3,c_ram
                djnz    r2,c_ram
                ret
        ;*****************************
        ;       RESET CHARACTOR
        ;*****************************
reset_char:     lcall   initlcd
                mov     dptr,#data2
                mov     n_chr,#20
                mov     start,#0
                lcall   list            ;"RS-485 COMUNICATION"
                mov     dptr,#data3
                mov     n_chr,#20
                mov     start,#40h      ;" copyright (c) 1991"
                lcall   list
                lcall   kout
                ret
        ;************************
clr_f:          mov     posit1,#10
                mov     posit2,#13
                ret
set_e:          mov     a,#0
                mov     dptr,#9003h
                mov     r1,#2
cc1:            movx    @dptr,a
                inc     dptr
                djnz    r1,cc1
                mov     a,#7fh
                movx    @dptr,a
                inc     dptr
                mov     a,#0ffh
                movx    @dptr,a
                mov     dptr,#trandat+13
                mov     a,#0
                mov     r1,#2
cc2:            movx    @dptr,a
                inc     dptr
                djnz    r1,cc2
                mov     a,#0fh
                movx    @dptr,a
                inc     dptr
                mov     a,#0ffh
                movx    @dptr,a
```

```
                ret
            ;*****************************
            ;       COMMAND  MODE
            ;*****************************
comm:           setb    tb8
                mov     a,command
                jnb     ti,$
                clr     ti
                mov     sbuf,a
                clr     tb8
                ret
            ;*****************************
            ;       CHECK BOARD
            ;*****************************
chk_b:          mov     command,#42h
chk_b0:         push    iec
                push    6
                push    dph
                push    dpl
                push    acc
                clr     es
                clr     et1
                setb    ti
                lcall   chk_b2
                clr     ri
                clr     ti
                pop     acc
                pop     dpl
                pop     dph
                pop     6
                pop     iec
chk_b1:         ret
chk_b2:         mov     dptr,#psign
                mov     a,#08h
                movx    @dptr,a
                lcall   comm
                mov     dptr,#trandat
                movx    a,@dptr
                jnb     ti,$
                clr     ti
                mov     sbuf,a
                inc     dptr
                movx    a,@dptr
                jnb     ti,$
                clr     ti
                mov     sbuf,a
                inc     dptr
                movx    a,@dptr
                jnb     ti,$
                clr     ti
                mov     sbuf,a
                mov     a,#0dh
                jnb     ti,$
                clr     ti
                mov     sbuf,a
```

```
                jnb     ti,$
                clr     ti
                 mov    dptr,#psign
                 mov    a,#08h
                 movx   @dptr,a
                 mov    dptr,#psign
                 mov    a,#0h
                movx    @dptr,a
                 clr    ti
                 ret
               ;****************************
               ;       INSTALL BOARD
               ;****************************
inst:           mov     command,#41h
                mov     code,#01
                lcall   numb00
                mov     a,#00
                cjne    a,number,inst0
inst1:          mov     stall,#00h
                lcall   ins_b0      ;inst3
                mov     a,#32h
                lcall   wait
                lcall   wait
                lcall   wait
                lcall   wait
                lcall   wait
                lcall   wait
                lcall   wait
                lcall   wait
                lcall   wait
                lcall   wait
                lcall   wait
                lcall   wait
                lcall   wait
                lcall   wait
                lcall   wait
                lcall   wait
                lcall   wait
                cjne    a,code,inst1
                lcall   add_e
                lcall   c_ram1
inst0:          lcall   kout
                ret
wait:           push    0
                mov     r0,#0ffh
                djnz    r0,$
                pop     0
                ret

               ;****************************
               ;       CHECK BOARD
               ;****************************
b_con:          mov     command,#44h
                lcall   numb00
                mov     a,#0
                cjne    a,number,b_con1
```

```
                mov     dptr,#8500h
b_con0:         movx    a,@dptr
 ;               mov     r7,#3
                cjne    a,#0dh,b_con2
b_con1:         mov     dptr,#buff_b
                mov     a,#0ffh
                movx    @dptr,a
;               lcall   main2
                ret
b_con2:         mov     code,a
te0:            lcall   ins_b0
                mov     r7,dela
te:             lcall   wait
                lcall   wait
                lcall   wait
                lcall   wait
                lcall   wait
                lcall   wait
                lcall   wait
                lcall   wait
                lcall   wait
                lcall   wait
                lcall   wait
                lcall   wait
                djnz    r7,te
                push    dph
                push    dpl
                mov     dptr,#buff_b
                movx    a,@dptr
                dec     code
                cjne    a,code,inc_s
b_con21:        pop     dpl
                pop     dph
                inc     dptr
                ljmp    b_con0
inc_s:          cjne    a,#0aah,b_con3  ;t
                inc     stall           ;t
                sjmp    b_con21         ;t
b_con3:  ;       dec     r7
         ;       cjne    r7,#0,te0
                lcall   error
                lcall   any_k
                pop     dpl
                pop     dph
                lcall   main2
                ret
;*****************************************
ins_b:          mov     command,#41h
                lcall   numb00
                mov     a,#0
                cjne    a,number,ins_b1
ins_b0:         push    iec
                push    6
                push    dph
                push    dpl
```

```
                push    acc
                clr     es
                clr     et1
                setb    ti
                lcall    ins_b2
                clr     ri
                clr     ti
                pop     acc
                pop     dpl
                pop     dph
                pop     6
                pop     iec
ins_b1:         ret
ins_b2:         mov     dptr,#psign
                mov     a,#08h
                movx    @dptr,a
                lcall   comm
                mov     a,code
                jnb     ti,$
                clr     ti
                mov     sbuf,a
                inc     code
                jnb     ti,$
                clr     ti
                mov     dptr,#psign
                mov     a,#08h
                movx    @dptr,a
                mov     dptr,#psign
                mov     a,#0h
                movx    @dptr,a
                clr     ti
                ret
            ;*************************
            ;     address to edit
            ;*************************
add_e:          mov     a,check_b
                mov     r5,check_b
                cjne    a,#0h,div2
                ret
div2:           cjne    a,#1h,div4
                mov     r0,#07h
                mov     r1,#0ffh
                mov     r2,#3fh
                mov     r3,#0ffh
                lcall   add_m
                mov     dptr,#const1
                lcall   div_e
                ret
div4:           mov     a,#3h
                clr     cy
                subb     a,check_b
                jc      div8
                mov     r0,#03h
                mov     r1,#0ffh
                mov     r2,#1fh
```

```
                mov     r3,#0ffh
                lcall   add_m
                mov     dptr,#const2
                lcall   div_e
                ret
div8:           mov     a,#7h
                clr     cy
                subb     a,check_b
                jc      div16
                mov     r0,#01h
                mov     r1,#0ffh
                mov     r2,#0fh
                mov     r3,#0ffh
                lcall   add_m
                mov     dptr,#const3
                lcall   div_e
                ret
div16:           mov     a,#15
                clr     cy
                subb     a,check_b
                jc      div32
                mov     r0,#00h
                mov     r1,#0ffh
                mov     r2,#07h
                mov     r3,#0ffh
                lcall   add_m
                ret
div32:           mov     a,#31
                clr     cy
                subb     a,#check_b
                jc      div_o
                mov     r0,#00h
                mov     r1,#07fh
                mov     r2,#03h
                mov     r3,#0ffh
                lcall   add_m
div_o:           ret
        ;*******************************
add_m:          mov     dptr,#stp_add1
                mov     a,r2
                movx    @dptr,a
                inc     dptr
                mov     a,r3
                movx    @dptr,a
                mov     dptr,#900fh
                mov     a,r0
                movx    @dptr,a
                inc     dptr
                mov     a,r1
                movx    @dptr,a
                ret
div_e:          mov     r0,dph
                mov     r1,dpl
div_g1:         lcall   add16
                lcall   send_e
```

```
                djnz    r5,div_g1
                ret
add16:          mov     a,r0
                mov     dptr,#buffer1
                movx    @dptr,a
                mov     a,r1
                mov     dptr,#buffer2
                movx    @dptr,a
                clr     cy
                addc    a,r3
                jnc     add17
                mov     dptr,#buffer4
                movx    @dptr,a
                inc     r2
                sjmp    add18
add17:          mov     dptr,#buffer4
                movx    @dptr,a
add18:          mov     r1,a
                mov     a,r0
                addc    a,r2
                mov     dptr,#buffer3
                movx    @dptr,a
                mov     r0,a
                mov     dph,r0
                mov     dpl,r1
                inc     dptr
                mov     r0,dph
                mov     r1,dpl
                ret
send_e:         mov     command,#46h
send_e1:        push    iec
                push    6
                push    dph
                push    dpl
                push    acc
                clr     es
                clr     et1
                setb    ti
                lcall   send_e2
                clr     ri
                clr     ti
                pop     acc
                pop     dpl
                pop     dph
                pop     6
                pop     iec
                ret
send_e2:        mov     dptr,#psign
                mov     a,#08h
                movx    @dptr,a
                lcall   comm
                mov     dptr,#buffer1
                movx    a,@dptr
                jnb     ti,$
                clr     ti
```

```
                mov     sbuf,a
                inc     dptr
                movx    a,@dptr
                jnb     ti,$
                clr     ti
                mov     sbuf,a
                inc     dptr
                movx    a,@dptr
                jnb     ti,$
                clr     ti
                mov     sbuf,a
                inc     dptr
                movx    a,@dptr
                jnb     ti,$
                clr     ti
                mov     sbuf,a
                mov     a,#0dh
                jnb     ti,$
                clr     ti
                mov     sbuf,a
                jnb     ti,$
                clr     ti
                 mov    dptr,#psign
                 mov    a,#08h
                 movx   @dptr,a
                 mov    dptr,#psign
                 mov    a,#0h
                movx    @dptr,a
                 clr    ti
                 ret
              ;*****************************
              ;       EDIT DATA
              ;*****************************
edit_dat:        lcall   clr_f
                 lcall    initicd
                 mov      dptr,#data4
                 mov      n_chr,#05h
                 mov      start,#05
                 lcall    list                ;"EDIT"
                 lcall    kout                ;key  not  on?
                 mov     a,#5h
                 addc    a,start
                 xch     a,start.
                 lcall    digit4             ;in key 4 digit
                 lcall   e_data              ;edit data xram
                 cjne    r7,#1,sgy
                 sjmp    edit_dat
sgy:             ret
              ;************************
e_data:          lcall    edata       ;list data external ram line1
                 cjne    r7,#3h,ch_err
                 ret
ch_err:          cjne    r7,#1,jj
                 lcall   eadderr
```

```
                ret
jj:             lcall   sck
                cjne    r3,#k24,e_data1
                mov     r7,#2
                ret
e_data1:        lcall    edata2      ;list data external ram line2
                cjne    r7,#1,jj1
                lcall   eadderr
                ret
jj1:            lcall   sck
                cjne    r3,#k24,e_data2
                mov     r7,#2
                ret
e_data2:        lcall    kout
                clr     cy
                mov     a,#05h
                addc    a,add1
                jc      eo
                mov     add1,a
                ljmp    e_data
eo:             mov     add1,a
                inc     addh
          ;     lcall    edec       ;list inc & dec data external ram
                lcall    sck
                cjne    r3,#k24,e_data
                ret
          ;*****************************
          ;    BOARD  NUMBER
          ;*****************************
numbb:          lcall   numb0
numb1:          mov     dptr,#add_numb
                mov     a,number
                movx    @dptr,a
                lcall   numb00
                mov     dptr,#add_numb
                movx    a,@dptr
                cjne    a,number,numb11
                lcall   sck
                cjne    r3,#k24,numb1
                ret
numb11:         mov     a,#31h
                clr     cy
                subb    a,number
                jc      numb12
                mov     start,#17
                lcall   goto
                lcall   numb011
                sjmp    numb1
numb12:         lcall   data_err
numb13:         lcall   numb00
        ;        mov     dptr,#add_numb
         ;       movx    a,@dptr
          ;      cjne    a,number,numb14
           ;     sjmp    numb13
numb14:         mov     a,#31h
```

```
                clr     cy
                subb    a,number
                jc      numb13
                ljmp    numbb
numb0:          lcall   numb00
                lcall   numb01
                ret
numb00:         mov     p1,#0ffh
                mov     a,#10h
                mov     dptr,#psign
                movx    @dptr,a
                mov     a,p1
                cpl     a
                mov     number,a
                mov     a,#00h          ;
                mov     dptr,#psign
                movx    @dptr,a
                mov     sub_n,number
                ret
numb01:         mov     a,#31h
                clr     cy
                subb    a,number
                jc      numb02
                lcall   initlcd
                mov     dptr,#data21
                mov     dptr,#data21
                mov     n_chr,#17
                mov     start,#0
                lcall   list
numb011:        mov     a,number
                lcall   h_2asc
                mov     r4,asch
                lcall   wrbyte
                mov     r4,ascl
                lcall   wrbyte
                lcall   kout
                ret
numb02:         lcall   data_err
                ret
        ;*******************************
        ;       TRANSMIT    ERROR
        ;*******************************
error:          lcall   initlcd
                mov     dptr,#data7     ;" TRANSMIT  ERROR"
                mov     n_chr,#20
                mov     start,#0
                lcall   list
                ret

        ;*******************************
        ;       DATA    ERROR
        ;*******************************
data_err:       lcall   initlcd
                mov     dptr,#data22    ;" DATA  ERROR"
                mov     n_chr,#20
```

```
                mov     start,#0
                lcall   list
                ret
          ;*****************************
          ;  PASS ANY KEY TO CONTINUE
          ;*****************************
any_k:          mov     r3,#0fh
                lcall   delay00
                lcall   initlcd
                mov     dptr,#data8
                mov     n_chr,#20
                mov     start,#0
                lcall   list                ;"PASS ANY KEY"
                mov     dptr,#data9
                mov     n_chr,#20
                mov     start,#40h          ;" TO CONTINUE "
                lcall   list
any_k1:         lcall   sck
                cjne    r3,#0f0h,anyo
                sjmp    any_k1
anyo:           ret
          ;*****************************
          ;  FUNCTION    LIST    DATA
          ;*****************************
list_daf:       lcall   initlcd
                mov     dptr,#data1
                mov     n_chr,#05h
                mov     start,#05
                lcall   list                ;"LIST"
                lcall   kout                ;key  not  on?
                mov     a,#5h
                addc    a,start
                xch     a,start
                lcall   digit4              ;in key 4 digit
                mov     a,#9fh
                clr     cy
                subb    a,addh
                jc      ang1
                cjne    a,#0,ang
                mov     a,#0f6h
                clr     cy
                subb    a,add1
                jc      ang1
ang:            mov     p1,addh
                lcall   l_data              ;list data to lcd
                ret
ang1:           lcall   adderr
                ljmp    list_dat
          ;************************************
          ;  FUNCTION    LIST    CHANNEL  DATA
          ;************************************
list_dat1:      mov     posit1,#4ah
                mov     posit2,#4dh
                lcall   initlcd
```

```
                lcall   name1
                lcall   name2
                mov     a,#13h
                clr     cy
                subb    a,addh
                jc      dang
nel:            lcall   sck
                cjne    r3,#k24,nel2
nel1:           mov     p1,addh
                lcall   l_data1          ;list data to lcd
                mov     p1,#55h
                ret
dang:           sjmp    list_dat1
nel2:           cjne    r3,#k19,nel3
                sjmp    nel10
nel3:           cjne    r3,#k20,nel4
                sjmp    nel10
nel4:           cjne    r3,#k21,nel5
                sjmp    nel10
nel5:           cjne    r3,#k22,nel6
                sjmp    nel10
nel6:           cjne    r3,#k13,nel7
                sjmp    nel10
nel7:           cjne    r3,#k14,nel8
                sjmp    nel10
nel8:           cjne    r3,#k15,nel9
                sjmp    nel10
nel9:           cjne    r3,#k16,nel
                sjmp    nel10
                ;lcall   kout
nel10:          lcall   chan_bin
                ret
        ;*******sub name********
name1:          mov      dptr,#data23
                mov      n_chr,#20
                mov      start,#0
                lcall   list             ;"FUNCTION LIST"
                ret
name2:          mov      dptr,#data19
                mov      n_chr,#08h
                mov      start,#42h
                lcall   list             ;"CHANNEL"
                lcall   kout
                mov     a,#8h
                addc    a,start
                xch     a,start
                lcall   digit4           ;in key 4 digit
                mov     a,#13h
                clr     cy
                subb    a,addh
                jc      arr
                lcall   kout
                mov     dptr,#data20
                mov     n_chr,#03h
                mov     start,#4fh
```

```
                lcall   list            ;"NO."
                lcall   kout            ;key  not  on?
                ret
arr:            lcall   chanerr
                ret
name3:          mov      dptr,#data19
                mov      n_chr,#08h
                mov     start,#42h
                lcall   list            ;"CHANNEL"
                lcall   kout
                mov     a,#8h
                addc    a,start
                xch     a,start
                lcall   digit4          ;in key 4 digit
                lcall   name7
                cjne    r7,#1h,name4
                sjmp    name6
name4:          lcall   chan_ex
                lcall   chk_stt
                cjne    r7,#1,name5
                mov     p1,#5h
                lcall   echanerr
                ret
name5:          mov     r7,#2
name6:          ret
name7:           mov     a,#0fh
                clr     cy
                subb    a,addh
                jc      arr1
                lcall   kout
arr11:          ret
arr1:           lcall   echanerr
                mov     r7,#1
                ret
            ;*******************************
l_data:         lcall   ldata       ;list data external ram line1
                lcall   ldata2      ;list data external ram line2
                lcall   kout
                lcall   ldec        ;list inc & dec data external ram
                lcall   sck
                cjne    r3,#k24,l_data
                ret
            ;*******************************
l_data1:        lcall   lchan       ;list data external ram line1
                lcall   lchan2      ;list data external ram line2
                lcall   kout
                lcall   d_chan        ;list inc & dec data external ram
                lcall   sck
                cjne    r3,#k24,l_data1
                ret
            ;*******************************
            ;    SUB    LIST    DATA
            ;*******************************
ldata:          push    1
                push    4
```

```
                push    acc
                lcall   initlcd
                mov     start,#0
                lcall   goto
                lcall   ldata0
                lcall   channel11
                pop     acc
                pop     4
                pop     1
                ret
ldata0:         mov     r1,#2
                mov     a,addh
ldata1:         mov     p1,a
                lcall   h_2asc
                mov     r4,asch
                lcall   wrbyte
                mov     r4,ascl
                lcall   wrbyte
                mov     a,addl
                djnz    r1,ldata1
                mov     r4,#3ah
                lcall   wrbyte
                ret
ldata2:         mov     start,#40h
                lcall   goto
                mov     a,addh
                push    1
                push    4
                push    acc
;               lcall   initlcd
                lcall   ldata00
                mov     addl,s_addl
                mov     addh,s_addh
                lcall   channel22
                pop     acc
                pop     4
                pop     1
                ret
ldata00:        mov     a,addl
                clr     cy
                addc    a,#5
                jnc     ldata000
                inc     addh
ldata000:       mov     addl,a
                mov     r1,#2
                mov     a,addh
ldata10:        mov     p1,a
                lcall   h_2asc
                mov     r4,asch
                lcall   wrbyte
                mov     r4,ascl
                lcall   wrbyte
                mov     a,addl
                djnz    r1,ldata1
                mov     r4,#3ah
```

```
                lcall   wrbyte
                lcall   kout
                ret
            ;*********************************
            ;  READ   DATA  EXT.RAM
            ;*********************************
channel11:      push    2
                push    4
                push    5
                push    6
                push    dph
                push    dpl
                push    acc
                mov     dph,addh
                mov     dpl,addl
                lcall   chan_ex11
                mov     dph,addh
                mov     dpl,addl
                movx    a,@dptr
                mov     r2,a
                inc     dptr
                movx    a,@dptr
                mov     r4,a
                inc     dptr
                movx    a,@dptr
                mov     r5,a
                inc     dptr
                movx    a,@dptr
                mov     r6,a
                inc     dptr
                movx    a,@dptr
                mov     r7,a
                mov     dptr,#buff_dis
                mov     a,r2
                movx    @dptr,a
                inc     dptr
                mov     a,r4
                movx    @dptr,a
                inc     dptr
                mov     a,r5
                movx    @dptr,a
                inc     dptr
                mov     a,r6
                movx    @dptr,a
                inc     dptr
                mov     a,r7
                movx    @dptr,a
                pop     acc
                pop     dpl
                pop     dph
                pop     6
                pop     5
                pop     4
                pop     2
                ret
```

```
channel22:      push    2
                push    4
                push    5
                push    6
                push    dph
                push    dpl
                push    acc
                mov     dph,addh
                mov     dpl,addl
                lcall   chan_ex00
                lcall   chan_ex11
                mov     dph,addh
                mov     dpl,addl
                lcall   chan_ex00
                movx    a,@dptr
                mov     r2,a
                inc     dptr
                movx    a,@dptr
                mov     r4,a
                inc     dptr
                movx    a,@dptr
                mov     r5,a
                inc     dptr
                movx    a,@dptr
                mov     r6,a
                inc     dptr
                movx    a,@dptr
                mov     r7,a
                mov     dptr,#buff_dis+5
                mov     a,r2
                movx    @dptr,a
                inc     dptr
                mov     a,r4
                movx    @dptr,a
                inc     dptr
                mov     a,r5
                movx    @dptr,a
                inc     dptr
                mov     a,r6
                movx    @dptr,a
                inc     dptr
                mov     a,r7
                movx    @dptr,a
                pop     acc
                pop     dpl
                pop     dph
                pop     6
                pop     5
                pop     4
                pop     2
                ret
chan_ex00:      inc     dptr
                inc     dptr
                inc     dptr
                inc     dptr
```

```
                inc     dptr
                ret
chan_ex11:       mov     r2,#5
chan_ex22:       movx    a,@dptr
                lcall   h_2asc
                mov     r4,asch
                lcall   wrbyte
                mov     r4,ascl
                lcall   wrbyte
                mov     func,#shiftr
                lcall   set
                inc     dptr
                djnz    r2,chan_ex22
                ret
            ;*************************************
            ;    FUNCTION  LIST DEC & INC CHANNEL
            ;*************************************
ldec:           lcall   chg_ramm
                cjne    r2,#2h,nid1
                lcall   sck
                cjne    r3,#k18,inc_chan01
                mov     a,#0
                cjne    a,add1,you1
                cjne    a,addh,you1
                mov     add1,#0f6h
                mov     addh,#9fh
                mov     s_add1,add1
                mov     s_addh,addh
                ret
you1:           clr     cy
                mov     a,add1
                subb    a,#10
                jc      d_chan11
                mov     add1,a
                mov     s_add1,a
nid1:           ret
d_chan11:       dec     s_addh
                dec     addh
                mov     add1,a
                mov     s_add1,a
                ret
inc_chan01:     cjne    r3,#k12,out_chan1
                mov     a,#9fh
                clr     cy
                subb    a,addh
                jc      me1
                cjne    a,#0,nott
                mov     a,#0f6h
                cjne    a,add1,nott
me1:            mov     add1,#0
                mov     addh,#0
                mov     s_add1,#0
                mov     s_addh,#0
                ret
nott:           clr     cy
```

```
                mov     a,#10
                addc    a,addl
                jc      inc_chan11
                mov     addl,a
                mov     s_addl,a
                ret
inc_chan11:     inc     s_addh
                inc     addh
                mov     addl,a
                mov     s_addl,a
                ret
out_chan1:       cjne    r3,#k24,ldec
                ret
chg_ramm:        mov    dph,addh
                mov     dpl,addl
chg_ram11:       mov     r2,#0
                lcall   chggg
chg_ram22:       mov     dptr,#buff_dis
                movx    a,@dptr
                cjne    a,10h,quit11
                inc     dptr
                movx    a,@dptr
                cjne    a,11h,quit11
                inc     dptr
                movx    a,@dptr
                cjne    a,12h,quit11
                inc     dptr
                movx    a,@dptr
                cjne    a,13h,quit11
                inc     dptr
                movx    a,@dptr
                cjne    a,14h,quit11
                inc     dptr
                movx    a,@dptr
                cjne    a,15h,quit11
                inc     dptr
                movx    a,@dptr
                cjne    a,16h,quit11
                inc     dptr
                movx    a,@dptr
                cjne    a,17h,quit11
                inc     dptr
                movx    a,@dptr
                cjne    a,18h,quit11
                inc     dptr
                movx    a,@dptr
                cjne    a,19h,quit11
                mov     r2,#2h
quit11:          ret
chggg:           movx    a,@dptr
                mov     10h,a
                inc     dptr
                movx    a,@dptr
                mov     11h,a
                inc     dptr
```

```
                movx    a,@dptr
                mov     12h,a
                inc     dptr
                movx    a,@dptr
                mov     13h,a
                inc     dptr
                movx    a,@dptr
                mov     14h,a
                inc     dptr
                movx    a,@dptr
                mov     15h,a
                inc     dptr
                movx    a,@dptr
                mov     16h,a
                inc     dptr
                movx    a,@dptr
                mov     17h,a
                inc     dptr
                movx    a,@dptr
                mov     18h,a
                inc     dptr
                movx    a,@dptr
                mov     19h,a
                ret
;*******************************
;    SUB    LIST  CHANNEL  DATA
;*******************************
lchan:          push    1
                push    4
                push    acc
                lcall   initlcd
                lcall   lchan0
                ljmp    lchan11
lchan0:         mov     r1,#2
                mov     a,addh
lchan1:         mov     p1,a
                lcall   h_2asc
                mov     r4,asch
                lcall   wrbyte
                mov     r4,ascl
                lcall   wrbyte
                mov     a,addl
                djnz    r1,lchan1
                mov     r4,#3ah
                lcall   wrbyte
                mov     r4,#30h
                lcall   wrbyte
                ret
lchan11:        mov     r4,#30h
                lcall   wrbyte
lchan111:       mov     r4,#3ah
                lcall   wrbyte
                lcall   channel1
                pop     acc
                pop     4
```

```
                pop     1
                ret
lchan2:         push    1
                push    4
                push    acc
                clr     cy
                mov     a,#00h
                addc    a,addl
                jc      chanin_2
                mov     addl,a
                mov     start,#40h
                lcall   goto
                lcall   lchan0
                mov     r4,#34h
                lcall   wrbyte
                ljmp    lchan22
chanin_2:       mov     addl,a
                inc     addh
                mov     start,#40
                lcall   goto
                lcall   lchan0
                mov     r4,#34h
                lcall   wrbyte
lchan22:        mov     r4,#3ah
                lcall   wrbyte
                lcall   channel2
                pop     acc
                pop     4
                pop     1
                ret
               ;*******************************
               ; READ CHANNEL  DATA  EXT.RAM
               ;*******************************
channel1:       push    2
                push    4
                push    5
                push    6
                push    dph
                push    dpl
                push    acc
                lcall   chan_ex
                lcall   chan_ex1
                lcall   chan_ex
                movx    a,@dptr
                mov     r2,a
                inc     dptr
                movx    a,@dptr
                mov     r4,a
                inc     dptr
                movx    a,@dptr
                mov     r5,a
                inc     dptr
                movx    a,@dptr
                mov     r6,a
                mov     dptr,#buff_dis
```

```
                mov     a,r2
                movx    @dptr,a
                inc     dptr
                mov     a,r4
                movx    @dptr,a
                inc     dptr
                mov     a,r5
                movx    @dptr,a
                inc     dptr
                mov     a,r6
                movx    @dptr,a
                pop     acc
                pop     dpl
                pop     dph
                pop     6
                pop     5
                pop     4
                pop     2
                ret
channel2:       push    2
                push    4
                push    5
                push    6
                push    dph
                push    dpl
                push    acc
                lcall   chan_ex
                lcall   chan_ex0
                lcall   chan_ex1
                lcall   chan_ex
                lcall   chan_ex0
                movx    a,@dptr
                mov     r2,a
                inc     dptr
                movx    a,@dptr
                mov     r4,a
                inc     dptr
                movx    a,@dptr
                mov     r5,a
                inc     dptr
                movx    a,@dptr
                mov     r6,a
                mov     dptr,#buff_dis+4
                mov     a,r2
                movx    @dptr,a
                inc     dptr
                mov     a,r4
                movx    @dptr,a
                inc     dptr
                mov     a,r5
                movx    @dptr,a
                inc     dptr
                mov     a,r6
                movx    @dptr,a
                pop     acc
```

```
                pop     dpl
                pop     dph
                pop     6
                pop     5
                pop     4
                pop     2
                ret
chan_ex:        mov     a,addl
                mov     b,#8h
                mul     ab
                mov     r5,a
                mov     r6,b
                mov     a,addh
                mov     b,#8
                mul     ab
                addc    a,r6
                mov     r6,a
                mov     dph,r6
                mov     dpl,r5
                ret
chan_ex0:       inc     dptr
                inc     dptr
                inc     dptr
                inc     dptr
                ret
chan_ex1:       mov     r2,#4
chan_ex2:       movx    a,@dptr
                lcall   h_2asc
                mov     r4,asch
                lcall   wrbyte
                mov     r4,ascl
                lcall   wrbyte
                mov     func,#shiftr
                lcall   set
                inc     dptr
                djnz    r2,chan_ex2
                ret
            ;***********************************
            ;    FUNCTION  LIST DEC & INC CHANNEL
            ;***********************************
d_chan:         lcall   chg_ram
                cjne    r2,#2h,nid
                lcall   sck
                cjne    r3,#k18,inc_chan
                mov     a,#0
                cjne    a,addl,you
                cjne    a,addh,you
                mov     addl,#0ffh
                mov     addh,#13h
nid:            ret
you:            clr     cy
                mov     a,addl
                subb    a,#1
                jc      d_chan1
                mov     addl,a
```

```
                ret
d_chan1:        dec     addh
                mov     addl,a
                ret
inc_chan:       cjne    r3,#k12,out_chan
                mov     a,#13h
                clr     cy
                subb    a,addh
                jc      me
                cjne    a,#0,noth
                mov     a,#0ffh
                cjne    a,addl,noth
me:             mov     addl,#0
                mov     addh,#0
                ret
noth:           clr     cy
                mov     a,#01h
                addc    a,addl
                jc      inc_chan1
                mov     addl,a
                ret
inc_chan1:      inc     addh
                mov     addl,a
                ret
out_chan:       cjne    r3,#k24,d_chan
                mov     a,addl
                subb    a,#4
                jc      out1_chan
                mov     addl,a
                ret
out1_chan:      dec     addh
                mov     addl,a
                ret
chg_ram:        lcall   chan_ex
chg_ram1:       mov     r2,#0
                lcall   chgg
chg_ram2:       mov     dptr,#buff_dis
                movx    a,@dptr
                cjne    a,10h,quit1
                inc     dptr
                movx    a,@dptr
                cjne    a,11h,quit1
                inc     dptr
                movx    a,@dptr
                cjne    a,12h,quit1
                inc     dptr
                movx    a,@dptr
                cjne    a,13h,quit1
                inc     dptr
                movx    a,@dptr
                cjne    a,14h,quit1
                inc     dptr
                movx    a,@dptr
                cjne    a,15h,quit1
                inc     dptr
```

```
                movx    a,@dptr
                cjne    a,16h,quit1
                inc     dptr
                movx    a,@dptr
                cjne    a,17h,quit1
                mov     r2,#2h
quit1:          ret
chgg:           movx    a,@dptr
                mov     10h,a
                inc     dptr
                movx    a,@dptr
                mov     11h,a
                inc     dptr
                movx    a,@dptr
                mov     12h,a
                inc     dptr
                movx    a,@dptr
                mov     13h,a
                inc     dptr
                movx    a,@dptr
                mov     14h,a
                inc     dptr
                movx    a,@dptr
                mov     15h,a
                inc     dptr
                movx    a,@dptr
                mov     16h,a
                inc     dptr
                movx    a,@dptr
                mov     17h,a
                ret
            ;*****************************
            ;   CHARACTOR  BINARY
            ;*****************************
chan_bin0:      push    dph
                push    dpl
                lcall   initlcd
                mov     dptr,#data19
                mov     start,#2
                mov     n_chr,#8
                lcall   list
                mov     a,addh
                lcall   m1
                mov     a,addl
                lcall   m1
                mov     dptr,#data20
                mov     start,#15
                mov     n_chr,#3
                lcall   list
                mov     a,numb
                lcall   h_2asc
                lcall   m2
                mov     dptr,#data31
                mov     start,#42h
                mov     n_chr,#7
```

```
                lcall   list
                pop     dpl
                pop     dph
                ret
m1:             lcall   h_2asc
                mov     r4,asch
                lcall   wrbyte
m2:             mov     r4,ascl
                lcall   wrbyte
                ret
          ;*****************************
          ;  LIST CHANNNEL BINARY
          ;*****************************
chan_bin:       lcall   chan_ex
                mov     start1,#52h
                lcall   digit1
                lcall   chan_bin0
                mov     start,#4ah
                lcall   goto
                mov     r5,#2
chan_bin1:      mov     a,numb
                clr     cy
                addc    a,dpl
                jc      chan_bin2
                mov     dpl,a
                ljmp    chan_bin3
chan_bin2:      inc     dph
                mov     dpl,a
chan_bin3:      lcall   in_hex
                lcall   kout
lod:            movx    a,@dptr
                cjne    a,20h,chan_bin4
                lcall   sck
                cjne    r3,#k24,lod
                ret
chan_bin4:      mov     start,#4ah
                lcall   goto
                sjmp    chan_bin3
in_hex:         movx    a,@dptr
                mov     20h,a
                jnb     7,set_7
                lcall   bin1
                sjmp    bit6
set_7:          lcall   bin0
                sjmp    bit6
bit6:           jnb     6,set_6
                lcall   bin1
                sjmp    bit5
set_6:          lcall   bin0
                sjmp    bit5
bit5:           jnb     5,set_5
                lcall   bin1
                sjmp    bit4
set_5:          lcall   bin0
                sjmp    bit4
```

```
bit4:           jnb     4,set_4
                lcall   bin1
                sjmp    bit3
set_4:          lcall   bin0
                sjmp    bit3
bit3:           jnb     3,set_3
                lcall   bin1
                sjmp    bit2
set_3:          lcall   bin0
                sjmp    bit2
bit2:           jnb     2,set_2
                lcall   bin1
                sjmp    bit1
set_2:          lcall   bin0
                sjmp    bit1
bit1:           jnb     1,set_1
                lcall   bin1
                sjmp    bit0
set_1:          lcall   bin0
                sjmp    bit0
bit0:           jnb     0,set_0
                lcall   bin1
                ret
set_0:          lcall   bin0
                ret

bin0:           mov     r4,#30h
                lcall   wrbyte
                ret
bin1:           mov     r4,#31h
                lcall   wrbyte
                ret
          ;*****************************
          ;  address or channel error
          ;*****************************
adderr:           push    dph
                  push    dpl
                  lcall   initlcd
                  mov     dptr,#data36
                  mov     start,#0
                  mov     n_chr,#20
                  lcall   list
                  mov     dptr,#data37
                  mov     start,#40h
                  mov     n_chr,#20
                  lcall   list
                  mov     r1,#10
                  lcall   vat     ;delay
                  pop     dpl
                  pop     dph
                  ret
eadderr:          push    dph
                  push    dpl
                  lcall   initlcd
                  mov     dptr,#data36
```

```
                mov     start,#0
                mov     n_chr,#20
                lcall   list
                mov      dptr,#data40
                mov     start,#40h
                mov     n_chr,#20
                lcall   list
                mov     start,#43h
                lcall   goto
                mov     r0,#2
                mov     dptr,#st_add1
eadderr1:       movx    a,@dptr
                lcall   h_2asc
                mov     r4,asch
                lcall   wrbyte
                mov     r4,ascl
                lcall   wrbyte
                inc     dptr
                djnz    r0,eadderr1
                mov     start,#4bh
                lcall   goto
                mov     r0,#2
eadderr2:       movx    a,@dptr
                lcall   h_2asc
                mov     r4,asch
                lcall   wrbyte
                mov     r4,ascl
                lcall   wrbyte
                inc     dptr
                djnz    r0,eadderr2
                mov     start,#55h
                lcall   goto
                mov     r1,#10
                lcall   vat
                pop     dpl
                pop     dph
                ret
chanerr:        push    dph
                push     dpl
                lcall   initlcd
                mov      dptr,#data38
                mov     start,#0
                mov     n_chr,#20
                lcall   list
                mov      dptr,#data39
                mov     start,#40h
                mov     n_chr,#20
                lcall   list
                mov     r1,#10
                lcall   vat
                pop     dpl
                pop     dph
                ret
echanerr:       push    dph
                push     dpl
```

```
                lcall   initlcd
                mov      dptr,#data38
                mov     start,#0
                mov     n_chr,#20
                lcall   list
                mov      dptr,#data40
                mov     start,#40h
                mov     n_chr,#20
                lcall   list
                mov     start,#43h
                lcall   goto
                mov     r0,#2
                mov     dptr,#stc_add1
echanerr1:      movx    a,@dptr
                lcall   h_2asc
                mov     r4,asch
                lcall   wrbyte
                mov     r4,ascl
                lcall   wrbyte
                inc     dptr
                djnz    r0,echanerr1
                mov     start,#4bh
                lcall   goto
                mov     r0,#2
echanerr2:      movx    a,@dptr
                lcall   h_2asc
                mov     r4,asch
                lcall   wrbyte
                mov     r4,ascl
                lcall   wrbyte
                inc     dptr
                djnz    r0,echanerr2
                mov     start,#55h
                lcall   goto
                mov     r1,#10
                lcall   vat
                pop     dpl
                pop     dph
                ret

        ;*****************************
        ;   CHECK   1  DIGIT
        ;*****************************
digit1:         mov     numb,#00
                mov     start,start1
;                lcall    kout
check1:        cjne    r3,#k24,check2
                ret
check2:         lcall   goto
                mov     r5,#16
                lcall   dig00
                cjne    r4,#00,check1
                cjne    r5,#0,check3
                sjmp    check10
check3:         cjne    r5,#1,check4
```

```
                sjmp    check10
check4:         cjne    r5,#2,check5
                sjmp    check10
check5:         cjne    r5,#3,check6
                sjmp    check10
check6:         cjne    r5,#4,check7
                sjmp    check10
check7:         cjne    r5,#5,check8
                sjmp    check10
check8:         cjne    r5,#6,check9
                sjmp    check10
check9:         cjne    r5,#7,check1
check10:        mov     numb,r5
                mov     a,numb
                lcall   h_2asc
                mov     r4,asc1
                lcall   wrbyte
                mov     func,#shift1
                lcall   set
                lcall   kout
                lcall   sck
                ljmp    check1

              ;*****************************
              ; CHACK  4   DIGIT
              ;*****************************
digit4:         mov     dptr,#data32
                mov     n_chr,#4h
                lcall   list
                mov     r4,#4
                mov     func,#shift1
digit44:        lcall   set
                djnz    r4,digit44
                mov     addh,#0
                mov     addl,#0
                mov     r6,#4
digi:           mov     dptr,#data18
                mov     r5,#16
                lcall   dig00
                lcall   dis0
                lcall   incc
                lcall   sck
                cjne    r3,#k24,digi
                mov     s_addh,addh
                mov     s_addl,addl
                ret
incc:           cjne    r3,#k18,decc
                cjne    r6,#1,incc1
                mov     r6,#4
                mov     start,posit1
                lcall   goto
                lcall   kout
                ret
incc1:          dec     r6
                mov     func,#shiftr
```

```
                lcall   set
                lcall   kout
                ret
decc:           cjne    r3,#k17,o_list
                cjne    r6,#4,decc1
                mov     r6,#1
                mov     start,posit2
                lcall   goto
                lcall   kout
                ret
decc1:          inc     r6
                mov     func,#shift1
                lcall   set
                lcall   kout
                ret
o_list:         ret
dis0:           cjne    r4,#0aah,dis1
                sjmp    dis2
dis1:           mov     a,r5
                movc    a,@a+dptr
                mov     r4,a
                lcall   wrbyte
                dec     r6
                lcall   table1
                lcall   kout
                ret
dis2:
table1:         cjne    r6,#3,table2
                anl     addh,#0fh
                mov     a,r5
                swap    a
                orl     addh,a
                ret
table2:         cjne    r6,#2,table3
                anl     addh,#0f0h
                mov     a,r5
                orl     addh,a
                ret
table3:         cjne    r6,#1,table4
                anl     addl,#0fh
                mov     a,r5
                swap    a
                orl     addl,a
                ret
table4:         cjne    r6,#0,dis2
                anl     addl,#0f0h
                mov     a,r5
                orl     addl,a
                mov     r6,#4
                mov     start,posit1
                lcall   goto
                ret
dig00:          lcall   sck
                cjne    r3,#k19,dig1
                sjmp    y1
dig1:           cjne    r3,#k20,dig2
```

```
                sjmp    y2
dig2:           cjne    r3,#k21,dig3
                sjmp    y3
dig3:           cjne    r3,#k22,dig4
                sjmp    y4
dig4:           cjne    r3,#k13,dig5
                sjmp    y5
dig5:           cjne    r3,#k14,dig6
                sjmp    y6
dig6:           cjne    r3,#k15,dig7
                sjmp    y7
dig7:           cjne    r3,#k16,dig8
                sjmp    y8
dig8:           cjne    r3,#k7,dig9
                sjmp    y9
dig9:           cjne    r3,#k8,dig10
                sjmp    y10
dig10:          cjne    r3,#k9,dig11
                sjmp    y11
dig11:          cjne    r3,#k10,dig12
                sjmp    y12
dig12:          cjne    r3,#k1,dig13
                sjmp    y13
dig13:          cjne    r3,#k2,dig14
                sjmp    y14
dig14:          cjne    r3,#k3,dig15
                sjmp    y15
dig15:          cjne    r3,#k4,dig16
                sjmp    y16
dig16:          cjne    r3,#k24,dig17
dig17:          mov     r4,#0aah
                ret
y1:             dec     r5
y2:             dec     r5
y3:             dec     r5
y4:             dec     r5
y5:             dec     r5
y6:             dec     r5
y7:             dec     r5
y8:             dec     r5
y9:             dec     r5
y10:            dec     r5
y11:            dec     r5
y12:            dec     r5
y13:            dec     r5
y14:            dec     r5
y15:            dec     r5
y16:            dec     r5
                mov     r4,#00
                ret
        ;*******************************
        ;       ENTER   FUNCTION
        ;*******************************
enter:            ret
```

```
        ;*******************************
        ;       SCAN  KEY  BOARD
        ;        key  =  r3
        ;*******************************
sck:            push    dph
                push    dpl
                push    1
                push    acc
scan:           mov     r3,#7
scank2:         dec     r3
                mov     a,r3
                mov     dptr,#pscan
                movx    @dptr,a
                movx    a,@dptr
                anl     a,#0f0h
                mov     r1,a
                cjne    a,#0f0h,keyin
                cjne    r3,#0,scank2
                mov     r3,#0f0h
                ljmp    out
keyin:          mov     a,r1
                jnb     acc7,keyi
                jnb     acc6,keyi
                jnb     acc5,keyi
                jnb     acc4,keyi
                ljmp    out
keyi:           subb    a,r3
                mov     r3,a
out:            pop     acc
                pop     1
                pop     dpl
                pop     dph
                lcall   delay
                lcall   delay
                ret
              ;***********************
kout:           push    dph
                push    dpl
                push    acc
                mov     dptr,#pscan
key1:           movx    a,@dptr
                anl     a,#0f0h
                cjne    a,#0f0h,key1
                pop     acc
                pop     dpl
                pop     dph
                ret
            ;*************************
            ;     LCD DISPLAY
            ;*************************

LCD:            push    dph
```

```
                push    dpl
                push    1
                push    2
                push    acc
                mov     r2,#04h
lcd1:           mov     dptr,#pctl
                mov     a,#88h
                movx    @dptr,a
                pop     acc
                pop     2
                pop     1
                pop     dph
                pop     dpl
                ret
            ;************************
initlcd:        push    dph
                push    dpl
                push    1
                push    2
                push    acc
                mov     r2,#14h
lcd2:           mov     a,#00h
                mov     dptr,#psign
                movx    @dptr,a
                mov     a,#38h
                mov     dptr,#pdata
                movx    @dptr,a
                lcall   empluse
                mov     a,#0fh
                mov     dptr,#pdata
                movx    @dptr,a
                lcall   empluse
                mov     a,#06h
                mov     dptr,#pdata
                movx    @dptr,a
                lcall   empluse
                mov     a,#01h
                mov     dptr,#pdata
                movx    @dptr,a
                lcall   empluse
                djnz    r2,lcd2
                pop     acc
                pop     2
                pop     1
                pop     dpl
                pop     dph
                ret
            ;*****************************
            ;    START=POSITION
            ;*****************************
goto:           push    acc
                push    1
                push    dph
                push    dpl
                mov     a,#80h
```

```
goto1:          orl     a,start
                mov     dptr,#pdata
                movx    @dptr,a
                mov     a,#00h
                mov     dptr,#psign
                movx    @dptr,a
                mov     r1,#1fh
ep11:           nop
                nop
                nop
                djnz    r1,ep11
                mov     dptr,#psign
                clr     a
                movx    a,@dptr
                orl     a,#04h
                movx    @dptr,a
                mov     r1,#20h
ep22:           nop
                nop
                djnz    r1,ep22
                anl     a,#0fbh
                movx    @dptr,a
                pop     dpl
                pop     dph
                pop     1
                pop     acc
                ret
        ;*******************************
        ;       r4 = data
        ;*******************************
wrbyte:         push    psw
                push    dph
                push    dpl
                push    acc
                mov     a,#01h
                mov     dptr,#psign
                movx    @dptr,a
                mov     a,r4
                mov     dptr,#pdata
                movx    @dptr,a
                lcall   empluse
                pop     acc
                pop     dpl
                pop     dph
                pop     psw
                ret
        ;*****************************

delay:          mov     r0,#0h
dl1:            nop
                nop
                nop
                nop
                djnz    r0,dl1
                ret
```

```
          ;*****************************
empluse:        push    dph
                push    dpl
                push    1
                mov     r1,#1fh
ep1:            nop
                nop
                nop
                djnz    r1,ep1
                mov     dptr,#psign
                clr     a
                movx    a,@dptr
                orl     a,#04h
                movx    @dptr,a
                mov     r1,#20h
ep2:            nop
                nop
                djnz    r1,ep2
                anl     a,#0fbh
                movx     @dptr,a
                pop     1
                pop     dpl
                pop     dpl
                ret
               ;*************************
               ;  FUNCTION  SET  LCD
               ;*************************
set:            push    dph
                push    dpl
                push    acc
                mov     dptr,#pdata
                mov     a,func
                movx    @dptr,a
                mov     dptr,#psign
                mov     a,#00h
                movx    @dptr,a
                lcall   empluse
                pop     acc
                pop     dpl
                pop     dph
                ret
        ;*******************************
        ;       CLEAR    XRAM
        ;*******************************
clear0:         push    1
                push    2
                push    3
                push    dph
                push    dpl
                push    acc
clear00:        mov     r1,#1
                lcall    initlcd
                mov      dptr,#data12
                mov      n_chr,#20
                mov      start,#0
```

```
                lcall   list            ;"FUNCTION"
                mov     dptr,#data13
                mov     n_chr,#20
                mov     start,#40h
                lcall   list            ;" CLEAR  XRAM"
                mov     r3,#0fh
                lcall   delay00
clear000:       lcall   initlcd
                mov     dptr,#data10
                mov     n_chr,#20
                mov     start,#0
                lcall   list            ;" ARE  YOU  SURE ?"
                mov     dptr,#data11
                mov     n_chr,#20
                mov     start,#40h
                lcall   list            ;" YES OR NO ?"
                lcall   kout
clear:          lcall   sck
                cjne    r3,#k18,clear1
                pop     acc
                pop     dpl
                pop     dph
                pop     3
                pop     2
                pop     1
                ret
clear1:         cjne    r3,#k12,clear2
                mov     a,#0
                mov     r2,#00h
                mov     r3,#0h
                mov     dptr,#8000h
clear11:        mov     a,#0
                movx    @dptr,a
                inc     dptr
                djnz    r3,clear11
                djnz    r2,clear11
                lcall   kout
                pop     acc
                pop     dpl
                pop     dph
                pop     3
                pop     2
                pop     1
                ret
clear2:         cjne    r3,#0f0h,clear000
                sjmp    clear
        ;*********************************
        ;       EDIT    CHANNEL
        ;*********************************

e_chan:         mov     posit1,#4ah
                mov     posit2,#4dh
                lcall   initlcd
                mov     dptr,#data24
                mov     n_chr,#20
```

```
                mov     start,#0
                lcall   list
                mov     r7,#2
                lcall   name3
                cjne    r7,#1,e_chan00
                ljmp    e_chan
                lcall   chan_ex
e_chan0:        lcall   chk_stt
                cjne    r7,#1,e_chan00
                ret
e_chan00:       lcall   lchan       ;list data external ram line1
                lcall   lchan2      ;list data external ram line2
                mov     start,#08h
                lcall   goto
                mov     r5,#1
                mov     r6,#0
comp0:          lcall   echar0
                cjne    r3,#k24,comp1
                ret
comp1:          mov     a,edd
                lcall   h_2asc
                cjne    r5,#1,comp2
                mov     r4,asch
                lcall   wrbyte
                mov     r5,#2
                lcall   kout
                sjmp    comp0
comp2:          cjne    r5,#2,comp1
                mov     a,edd
                lcall   h_2asc
                mov     r4,ascl
                lcall   wrbyte
                mov     r5,#1
                mov     func,#shiftr
                lcall   set
                inc     dptr
                inc     r6
                lcall   kout
                cjne    r6,#4,comp3
                mov     start,#48h
                lcall   goto
comp3:          cjne    r6,#8,comp0
                inc     add1
                mov     a,#0
                cjne    a,add1,e_chan0
                inc     addh
                sjmp    e_chan0
e_chan1:        lcall   sck
                cjne    r3,#k24,e_chan1
                ret
        ;*********************************
        ;         CHACK ADDRESS
        ;*********************************
chk_stt:        push    acc
                mov     posit1,dph
```

```
                mov     posit2,dpl
                mov     dptr,#st_add1
                movx    a,@dptr
                mov     dph,posit1
                xch     a,posit1
                clr     cy
                subb    a,posit1
                jc      chk_out
                mov     posit1,dph
                mov     dptr,#st_add2
                movx    a,@dptr
                mov     dpl,posit2
                xch     a,posit2
                clr     cy
                subb    a,posit2
                jnc     chk_stp
chk_out:        mov     dph,posit1
                mov     dpl,posit2
                mov     r7,#1
                pop     acc
                ret
chk_stp:        mov     posit2,dpl
                mov     dph,posit1
                mov     dptr,#stp_add1
                movx    a,@dptr
                mov     dph,posit1
                clr     cy
                subb    a,posit1
                jc      chk_out
                cjne    a,#0,chk_pss
                mov     posit1,dph
                mov     dptr,#stp_add2
                movx    a,@dptr
                mov     dpl,posit2
                clr     cy
                subb    a,posit2
                jc      chk_out
                mov     posit2,dpl
chk_pss:        mov     dpl,posit2
                mov     dph,posit1
                mov     r7,#2
                pop     acc
                ret

        ;*********************************
        ;       EDIT
        ;*********************************
edata:          push    1
                push    4
                push    acc
                mov     dpl,addl
                mov     dph,addh
                lcall   chk_stt
                cjne    r7,#1,edatag
                pop     acc
```

```
                pop     4
                pop     1
                ret
edatag:         lcall   initlcd
edata0:         mov     r1,#2
                mov     a,addh
edata1:         mov     p1,a
                lcall   h_2asc
                mov     r4,asch
                lcall   wrbyte
                mov     r4,ascl
                lcall   wrbyte
                mov     a,addl
                djnz    r1,edata1
                mov     r4,#3ah
                lcall   wrbyte
                lcall   wd_ex
                mov     dph,addh
                mov     dpl,addl
                inc     dptr
                lcall   chk_stt
                cjne    r7,#1,goo
                mov     r7,#3h
goo:            pop     acc
                pop     4
                pop     1
                ret
edata2:         push    1
                push    4
                push    acc
                clr     cy
                mov     a,#05h
                addc    a,addl
                jc      ein_2
                mov     addl,a
                mov     start,#40h
                lcall   goto
                ljmp    edata0
ein_2:           mov     addl,a
                inc     addh
                mov     start,#40
                lcall   goto
                ljmp    edata0

              ;*********************************
              ;    WRITE  DATA  EXT.RAM
              ;*********************************
wd_ex:          push    2
                push    4
                push    5
                push    6
                push    dph
                push    dpl
                push    acc
                lcall   kout
```

```
                mov     r6,#5
                mov     dph,addh
                mov     dpl,addl
wd_ex1:         mov     r5,#1
                lcall   chk_stt
                cjne    r7,#1,rem
                sjmp    rem1
rem:            lcall   echar0      ;char edit
                lcall   sck
                cjne    r3,#k24,wd_ex2
rem1:           pop     acc
                pop     dpl
                pop     dph
                pop     6
                pop     5
                pop     4
                pop     2
                ret
wd_ex2:         lcall   h_2asc
                mov     r4,ascl
                lcall   wrbyte
                lcall   kout
                mov     r5,#2
                lcall   echar0      ;char edit
                lcall   chk_stt
                cjne    r7,#1,em
                sjmp    em1
em:             lcall   sck
                cjne    r3,#k24,wd_ex3
em1:            pop     acc
                pop     dpl
                pop     dph
                pop     6
                pop     5
                pop     4
                pop     2
                ret
wd_ex3:         lcall   h_2asc
                mov     r4,ascl
                lcall   wrbyte
                lcall   kout
                mov     func,#shiftr
                lcall   set
                inc     dptr
                lcall   kout
                djnz    r6,wd_ex1
                mov     p1,#55h
                pop     acc
                pop     dpl
                pop     dph
                pop     6
                pop     5
                pop     4
                pop     2
                ret
```

```
                ;*********************************
                ;         HEX TO ASCII
                ;         INPUT = A
                ;        OUTPUT = ASCH+ASCL
                ;*********************************
h_2asc:         push    0
                mov     r0,#asch
                mov     @r0,a
                push    acc
                lcall   htoa
                mov     ascl,a
                mov     a,@r0
                swap    a
                lcall   htoa
                mov     asch,a
                pop     acc
                pop     0
                ret
htoa:           push    psw
                clr     cy
                anl     a,#0fh
                cjne    a,#09h,htoa1
                setb    cy
htoa1:          jnb     cy,htoa2
                orl     a,#30h
                sjmp    end_htoa
htoa2:          subb    a,#09h
                orl     a,#40h
end_htoa:       pop     psw
                ret
                ;*******************************
                ;     SHOW  CHARECTOR
                ;     INPUT = DPTR
                ;     N = N_CHR
                ;*******************************
list:           push    psw
                push    2
                push    4
                push    5
                push    6
                push    acc
                mov     r5,n_chr
list1:          mov     r2,start
                lcall   goto
                mov     r6,#0h
list0:          mov     a,r6
list2:          movc    a,@a+dptr
                mov     r4,a
                lcall   wrbyte
                inc     r6
                djnz    r5,list0
                pop     acc
                pop     6
                pop     5
                pop     4
```

```
                pop     2
                pop     psw
                ret
        ;*********************************
        ;       CHARACTER   EDIT
        ;*********************************
echar0:         lcall    sck
echar1:         cjne     r3,#k1,echar2
                mov     r2,#0ch
                sjmp    enn0
echar2:         cjne     r3,#k2,echar3
                mov     r2,#0dh
enn0:           sjmp    enn
echar3:         cjne     r3,#k3,echar4
                mov     r2,#0eh
                sjmp    enn
echar4:         cjne     r3,#k4,echar5
                mov     r2,#0fh
                sjmp    enn
echar5:         cjne     r3,#k7,echar6
                mov     r2,#08h
                sjmp    enn
echar6:         cjne     r3,#k8,echar7
                mov     r2,#09h
                sjmp    enn
echar7:         cjne     r3,#k9,echar8
                mov     r2,#0ah
                sjmp    enn
echar8:         cjne     r3,#k10,echar9
                mov     r2,#0bh
                sjmp    enn
echar9:         cjne     r3,#k13,echar10
                mov     r2,#04h
                sjmp    enn
echar10:        cjne     r3,#k14,echar11
                mov     r2,#05h
                sjmp    enn
echar11:        cjne     r3,#k15,echar12
                mov     r2,#06h
                sjmp    enn
echar12:        cjne     r3,#k16,echar13
                mov     r2,#07h
                sjmp    enn
echar13:        cjne     r3,#k19,echar14
                mov     r2,#0h
                sjmp    enn
echar14:        cjne     r3,#k20,echar15
                mov     r2,#01h
                sjmp    enn
echar15:        cjne     r3,#k21,echar16
                mov     r2,#02h
                sjmp    enn
echar16:        cjne     r3,#k22,echar17
                mov     r2,#03h
                sjmp    enn
```

```
echar17:        cjne    r3,#k24,echar199

                ret
echar199:       ljmp    echar0
enn:            ; mov     a,#1
                cjne    r5,#1,enn1
                mov     a,r2
                swap    a
                mov     edd,a
                swap    a
                ret
enn1:           ;  mov     a,#2
                cjne    r5,#2,enn2
                mov     a,r2
                orl     a,edd
                mov     edd,a
                movx    @dptr,a
                mov     trans_h,dph
                mov     trans_l,dpl
                mov     dptr,#trandat
                mov     a,trans_h
                movx    @dptr,a
                inc     dptr
                mov     a,trans_l
                movx    @dptr,a
                inc     dptr
                mov     a,edd
                movx    @dptr,a
                mov     command,#42h
                lcall   chk_b
                mov     a,edd
                mov     dph,trans_h
                mov     dpl,trans_l
                mov     a,r2
                mov     edd,a
                ret
enn2:           ljmp    echar0
;*****************************************************
;             write ascii to lcd
;*****************************************************
ascii_dat:      lcall   list_page0
                push    7
                push    6
                push    5
                push    4
                push    3
                mov     p1,#7fh
                mov     start,#0
                lcall   goto
                lcall   sck
                lcall   kout
ascii:          mov     r7,#0
                mov     buffh,#80h
                mov     buffl,#0
                mov     d_buff,#0
```

```
                lcall   kout
ascii0:          mov     r6,#5
ascii00:         mov     r5,#22
                cjne    r6,#5,dat15
                mov     dptr,#data14
                mov     p1,#7fh
                ljmp    wrlcd
dat15:          cjne    r6,#4,dat16
                mov     dptr,#data15
                mov     p1,#0bfh
                ljmp    wrlcd
dat16:          cjne    r6,#3,dat17
                mov     dptr,#data16
                mov     p1,#0dfh
                ljmp    wrlcd
dat17:          cjne    r6,#2,contr
                mov     dptr,#data17
                mov     p1,#0efh
                ljmp    wrlcd
contr:          cjne    r6,#1,ascii
                mov     p1,#0f7h
                lcall   sck
left:          cjne    r3,#k17,space
                cjne    r7,#0,left0
                mov     r7,#53h
                mov     d_buff,#39
                mov     start,#53h
                lcall   goto
                lcall   kout
                sjmp    space
left0:          cjne    r7,#40h,left1
                mov     r7,#13h
                mov     d_buff,#19
                mov     start,#13h
                lcall   goto
                lcall   kout
                sjmp    space
left1:          mov     func,#shift1
                lcall   set
                dec     r7
                dec     d_buff
                lcall   kout
space:          cjne    r3,#k18,con_out
space1:         cjne    r7,#53h,right
                mov     r7,#0h
                mov     d_buff,#0
                mov     start,#0h
                lcall   goto
                lcall   kout
                sjmp    con_out
right:          cjne    r7,#13h,right1
                mov     r7,#40h
                mov     d_buff,#20
                mov     start,#40h
                lcall   goto
```

```
                lcall   kout
                sjmp    con_out
right1:         mov     func,#shiftr
                lcall   set
                inc     r7
                inc     d_buff
                lcall   kout
con_out:        cjne    r3,#k23,contr
                mov     p1,#0f7h
                lcall   kout
                ljmp    ascii0
wrlcd:          mov     r4,#0ffh
                lcall   asciii
                cjne    r4,#0aah,disp
                cjne    r3,#k23,shift2
                dec     r6
                lcall   kout
shift1:         ljmp    ascii00
shift2:         cjne    r3,#k24,shift1
                pop     3
                pop     4
                pop     5
                pop     6
                pop     7
                ret
;*********dipplay lcd**********
disp:           mov     a,r5
                movc    a,@a+dptr
                mov     r4,a
                lcall   wrbyte
                lcall   kout
                lcall   save
                mov     r4,#0ffh
                inc     d_buff
                inc     r7
row2:           cjne    r7,#54h,row1
                mov     r7,#0h
                mov     d_buff,#0
                mov     start,#0h
                lcall   goto
                lcall   kout
                sjmp    row_out
row1:           cjne    r7,#14h,row_out
                mov     r7,#40h
                mov     d_buff,#20
                mov     start,#40h
                lcall   goto
                lcall   kout
row_out:        ;  inc     r7
                ljmp    ascii00
;*****************************
asciii:         lcall   sck
                cjne    r3,#k1,ascii1
                sjmp    x1
ascii1:         cjne    r3,#k2,ascii2
```

```
                sjmp    x2
ascii2:         cjne    r3,#k3,ascii3
                sjmp    x3
ascii3:         cjne    r3,#k4,ascii4
                sjmp    x4
ascii4:         cjne    r3,#k5,ascii5
                sjmp    x5
ascii5:         cjne    r3,#k6,ascii6
                sjmp    x6
ascii6:         cjne    r3,#k7,ascii7
                sjmp    x7
ascii7:         cjne    r3,#k8,ascii8
                sjmp    x8
ascii8:         cjne    r3,#k9,ascii9
                sjmp    x9
ascii9:         cjne    r3,#k10,ascii10
                sjmp    x10
ascii10:        cjne    r3,#k11,ascii11
                sjmp    x11
ascii11:        cjne    r3,#k12,ascii12
                sjmp    x12
ascii12:        cjne    r3,#k13,ascii13
                sjmp    x13
ascii13:        cjne    r3,#k14,ascii14
                sjmp    x14
ascii14:        cjne    r3,#k15,ascii15
                sjmp    x15
ascii15:        cjne    r3,#k16,ascii16
                sjmp    x16
ascii16:        cjne    r3,#k17,ascii17
                sjmp    x17
ascii17:        cjne    r3,#k18,ascii18
                sjmp    x18
ascii18:        cjne    r3,#k19,ascii19
                sjmp    x19
ascii19:        cjne    r3,#k20,ascii20
                sjmp    x20
ascii20:        cjne    r3,#k21,ascii21
                sjmp    x21
ascii21:        cjne    r3,#k22,shift
                sjmp    x22
shift:          mov     r4,#0aah
shift0:         ret
x1:             dec     r5
x2:             dec     r5
x3:             dec     r5
x4:             dec     r5
x5:             dec     r5
x6:             dec     r5
x7:             dec     r5
x8:             dec     r5
x9:             dec     r5
x10:            dec     r5
x11:            dec     r5
x12:            dec     r5
```

```
x13:            dec     r5
x14:            dec     r5
x15:            dec     r5
x16:            dec     r5
x17:            dec     r5
x18:            dec     r5
x19:            dec     r5
x20:            dec     r5
x21:            dec     r5
x22:            dec     r5
                ret
        ;******save data to ram *****
save:           push    dph
                push    dpl
                mov     dph,buffh
                mov     dpl,buffl
                mov     a,d_buff
                clr     cy
                addc    a,dpl
                jnc     save1
                inc     dph
save1:          mov     dpl,a
                mov     a,r4
                movx    @dptr,a
                pop     dpl
                pop     dph
                ret
        ;************************************
        ;       LIST ASCII FROM RAM TO LCD
        ;************************************
list_page:      lcall   list_page0
inc_p:          lcall   sck
out_p:          cjne    r3,#k24,inc_p
                ret
list_page0:     mov     buffh,#80h
                mov     buffl,#0
                mov     dph,buffh
                mov     dpl,buffl
list_page1:     lcall   dat1
                lcall   dat2
                lcall   kout
                ret
        ;*************************
dat1:           mov     start,#0h
sub_dat:        mov     n_chr,#20
                lcall   list_asc
                ret
        ;*************************
dat2:           mov     start,#40h
                sjmp    sub_dat
        ;*************************
list_asc:       push    psw
                push    2
                push    4
                push    5
```

```
                push    acc
                mov     r5,n_chr
list1_asc:      mov     r2,start
                lcall   goto
list0_asc:      movx    a,@dptr
                mov     r4,a
                lcall   wrbyte
                inc     dptr
                djnz    r5,list0_asc
                pop     acc
                pop     5
                pop     4
                pop     2
                pop     psw
                ret

        ;*********************************
        ;       baud  rate
        ;*********************************
rate:           lcall   initlcd
                mov     start,#0
                lcall   goto
                mov     dptr,#data25
                mov     n_chr,#20
                mov     start,#0
                lcall   list
                mov     dptr,#data26
                mov     n_chr,#13
                mov     start,#40h
                lcall   list
                lcall   dis_baud
b_dec:          lcall   sck
                cjne    r3,#k12,b_inc
                lcall   inc_b
                lcall   dis_baud
                lcall   kout
b_inc:          cjne    r3,#k18,out_b
                lcall   dec_b
                lcall   dis_baud
                lcall   kout
out_b:          cjne    r3,#k24,b_dec
                mov     th1,baud
                ret
              ;**********************
inc_b:          mov     a,#0fdh
                cjne    a,baud,inc_b1
                mov     baud,#0e8h
                mov     dela,#10
                ret
inc_b1:         mov     a,#0fah
                cjne    a,baud,inc_b2
                mov     baud,#0fdh
                mov     dela,#1
                ret
inc_b2:         mov     a,#0f4h
```

```
                cjne    a,baud,inc_b3
                mov     baud,#0fah
                mov     dela,#5
                ret
inc_b3:         mov     a,#0e8h
                cjne    a,baud,inc_b4
                mov     baud,#0f4h
                mov     dela,#7
inc_b4:         ret
dec_b:          mov     a,#0fdh
                cjne    a,baud,dec_b1
                mov     baud,#0fah
                mov     dela,#5
                ret
dec_b1:         mov     a,#0fah
                cjne    a,baud,dec_b2
                mov     baud,#0f4h
                mov     dela,#7
                ret
dec_b2:         mov     a,#0f4h
                cjne    a,baud,dec_b3
                mov     baud,#0e8h
                mov     dela,#10
                ret
dec_b3:         mov     a,#0e8h
                cjne    a,baud,dec_b4
                mov     baud,#0fdh
                mov     dela,#1
dec_b4:         ret


                ;***************
dis_baud:       push    acc
                push    dph
                push    dpl
                mov     a,baud
b1:             cjne    a,#0fdh,b2
                mov     dptr,#data27
                sjmp    show
b2:             cjne    a,#0fah,b3
                mov     dptr,#data28
                sjmp    show
b3:             cjne    a,#0f4h,b4
                mov     dptr,#data29
                sjmp    show
b4:             cjne    a,#0e8h,b5
                mov     dptr,#data30
                sjmp    show
b5:             mov     dptr,#data27
                mov     baud,#0fdh
show:           mov     n_chr,#4
                mov     start,#4eh
                lcall   list
                pop     dpl
```

```
                pop     dph
                pop     acc
                ret

        ;*************************

trans_dat:      mov     dptr,#8000h
                ret
          ;*** display to port1 ***
delay000:       mov     r3,#01h
delay00;        mov     r4,#0ffh
delay1:         mov     r5,#0ffh
delay2:         djnz    r5,delay2
                djnz    r4,delay1
                djnz    r3,delay00
                ret
try:            push    acc
                push    dph
                push    dpl
                push    0
                mov     dptr,#8000h
try1:           lcall   receive
                movx    @dptr,a
                cjne    a,#41h,try2   ;check  board
                clr     sm2
                lcall   receive
                cjne    a,number,try11
                inc     dptr
                movx    @dptr,a
                lcall   wait
                lcall   chai
try11:          ljmp    try_n
try2:           cjne    a,#42h,try_3   ;receive edit data
                clr     sm2
                lcall   receive
                mov     dph,a
                lcall   receive
                mov     dpl,a
                lcall   receive
                movx    @dptr,a
                lcall   receive
                cjne    a,#0dh,try_ee
                ljmp    try_n
try_ee:         ljmp    try_e
try_3:           cjne    a,#43h,try_4  ;the main board
                clr     sm2
                mov     dptr,#8500h
                mov     dpl,check_b
                lcall   receive
                movx    @dptr,a
                inc     dptr
                mov     a,#0dh
                movx    @dptr,a
                inc     check_b
                mov     dptr,#9999h
```

```
                mov     a,check_b
                movx    @dptr,a
                ljmp    try_n
try_4:          cjne    a,#44h,try_5
                clr     sm2
                lcall   receive
                cjne    a,sub_n,try_42
try_41:         mov     dptr,#9000h
                mov     a,#90h
                movx    @dptr,a
                inc     dptr
                mov     a,#07
                movx    @dptr,a
                inc     dptr
                mov     a,number
                movx    @dptr,a
                lcall   wait
                lcall   chk_b
                lcall   chk_b
                lcall   chk_b
try_42:         ljmp    try_n
try_5:          cjne    a,#45h,try_6
                clr     sm2
                lcall   receive
                cjne    a,#0ddh,try_n
                mov     stall,#0
                 dec     stall
;               ljmp    try_n
try_6:          cjne    a,#46h,try_n  ;the terminal board
                clr     sm2           ;address  edit
                mov     dptr,#trandat+3
                lcall   receive
                movx    @dptr,a
                inc     dptr
                lcall   receive
                movx    @dptr,a
                inc     dptr
                lcall   receive
                movx    @dptr,a
                inc     dptr
                lcall   receive
                movx    @dptr,a
                inc     dptr
                lcall   receive
                movx    @dptr,a
                lcall   receive
                cjne    a,#0dh,try_e
                lcall   mov_add
                clr     ri
                clr     ti
                pop     0
                pop     dpl
                pop     dph
                pop     acc
                setb    sm2
                setb    es
                lcall   c_ram1
                reti
```

```
try_e:          lcall   error
try_n:          clr     ri
                clr     ti
                pop     0
                pop     dpl
                pop     dph
                pop     acc
                setb    sm2
                setb    es
                reti
        ;*** receive routine ***
receive:        jnb     ri,$
                clr     ri
                mov     a,sbuf
                ret

        ;*** transmitt routine ***
sbyte:          clr     es
                clr     ea
                jnb     ti,$
                clr     ti
                mov     sbuf,a
                ret
        ;**************************
        ;*     MOV ADDRESS        *
        ;**************************
mov_add:        mov     dptr,#9003h
                movx    a,@dptr
                mov     r0,a
                mov     dptr,#9005h
                movx    a,@dptr
                subb    a,r0
                cjne    a,#3fh,m_add4
                mov     dptr,#900dh
                mov     a,#08h
                movx    @dptr,a
                mov     a,#0h
                inc     dptr
                movx    @dptr,a
                inc     dptr
                mov     a,#0fh
                movx    @dptr,a
                inc     dptr
                mov     a,#0ffh
                movx    @dptr,a
                ret
m_add4:         cjne    a,#1fh,m_add8
                mov     dptr,#900dh
                mov     a,#0ch
                movx    @dptr,a
                mov     a,#00h
                inc     dptr
                movx    @dptr,a
                inc     dptr
                mov     a,#0fh
```

```
                movx    @dptr,a
                inc     dptr
                mov     a,#0ffh
                movx    @dptr,a
                ret
m_add8:         cjne    a,#0fh,m_add16
                mov     dptr,#900dh
                mov     a,#0eh
                movx    @dptr,a
                mov     a,#00h
                inc     dptr
                movx    @dptr,a
                inc     dptr
                mov     a,#0fh
                movx    @dptr,a
                inc     dptr
                mov     a,#0ffh
                movx    @dptr,a
                ret
m_add16:        cjne    a,#7h,m_add32
                mov     dptr,#900dh
                mov     a,#0fh
                movx    @dptr,a
                mov     a,#00h
                inc     dptr
                movx    @dptr,a
                inc     dptr
                mov     a,#0fh
                movx    @dptr,a
                mov     a,#0ffh
                inc     dptr
                movx    @dptr,a
                ret
m_add32:        cjne    a,#03h,m_addo
                mov     dptr,#900dh
                mov     a,#0fh
                movx    @dptr,a
                mov     a,#80h
                inc     dptr
                movx    @dptr,a
                inc     dptr
                mov     a,#0fh
                movx    @dptr,a
                inc     dptr
                mov     a,#0ffh
                movx    @dptr,a
m_addo:         ret
        ;*****************************
        ;       CHECK BOARD
        ;*****************************
chai:           mov     command,#43h
                lcall   numb00
chai_b0:         push    iec
                clr     es
                clr     et1
```

```
                setb    ti
                lcall   chai_b2
                clr     ri
                clr     ti
                pop     iec
chai_b1:         ret
chai_b2:         mov     dptr,#psign
                mov     a,#08h
                movx    @dptr,a
                lcall   comm
                mov     a,number
                jnb     ti,$
                clr     ti
                mov     sbuf,a
                jnb     ti,$
                clr     ti
                mov     dptr,#psign
                mov     a,#08h
                movx    @dptr,a
                mov     dptr,#psign
                mov     a,#0h
                movx    @dptr,a
                clr     ti
                ret
;*****************************
data1:          dfb     "LIST:"
data2:          dfb   "RS-485 COMMUNICATION"
data3:          dfb   " copyright(C)  1991 "
data4:          dfb     "EDIT:"
data5:          dfb   " TRANSFER COMPLATE! "
data6:          dfb   " NOW! TRANSMIT DATA "
data7:          dfb   "  TRANSMIT  ERROR!  "
data8:          dfb   "*** PASS ANY KEY ***"
data9:          dfb   "** TO MAINPROGRAM **"
data10:         dfb   "   ARE YOU SURE ?   "
data11:         dfb   "    YES! OR NO!     "
data12:         dfb   "***** FUNCTION *****"
data13:         dfb   "**** CLEAR XRAM ****"
data14:         dfb   "abcdefghijklmnopqrstuv"
data15:         dfb   "ABCDEFGHIJKLMNOPQRSTUV"
data16:         dfb   "wxyz0123456789+-*/!@#$"
data17:         dfb   "WXYZ%&()<>{}[]:;,.=?"
                dfb   22h,20h
data18:         dfb   "0123456789ABCDEF"
data19:         dfb   "CHANNEL:"
data20:         dfb   "NO."
data21:         dfb   " TERMINAL NUMBER:"
data22:         dfb   "  NO.BOARD ERROR!  "
data23:         dfb   "   FUNCTION  LIST  "
data24:         dfb   "   FUNCTION  EDIT  "
data25:         dfb   "   FUNCTION  SET   "
data26:         dfb   "   baud rate:"
data27:         dfb   "9600"
data28:         dfb   "4800"
```

```
data29:         dfb  "2400"
data30:         dfb  "1200"
data31:         dfb  "BINARY:"
data32:         dfb  "0000"
data33:         dfb  " ** TEST  EX-RAM ** "
data34:         dfb  "     EX-RAM PASS!    "
data35:         dfb  "    EX-RAM  ERROR!   "
data36:         dfb  "    ADDRESS ERROR!   "
data37:         dfb  "   (0000H---9FF6H)   "
data38:         dfb  "    CHANNEL ERROR!   "
data39:         dfb  "   (0000H---13FFH)   "
data40:         dfb  "   (-    H---    H)  "

                END
```

ภาคผนวกที่ 2

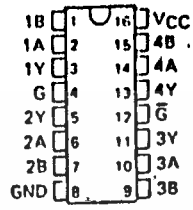
ในภาคผนวกที่ 2 แสดง DATA SHEET ของ IC เบอร์ SN75173 และ เบอร์ SN75174

SN75173
QUADRUPLE DIFFERENTIAL LINE RECEIVER

D2600, OCTOBER 1980 – SEPTEMBER 1980

- Meets EIA Standards RS-422-A, RS-423-A, and RS-485
- Meets CCITT Recommendations V.10, V.11, X.26, and X.27
- Designed for Multipoint Bus Transmission on Long Bus Lines in Noisy Environments
- 3-State Outputs
- Common-Mode Input Voltage Range . . . – 12 to 12 V
- Input Sensitivity . . . ± 200 mV
- Input Hysteresis . . . 50 mV Typ
- High Input Impedance . . . 12 kΩ Min
- Operates from Single 5-Volt Supply
- Low Power Requirements
- Plug-In Replacement for AM26LS32

D, J, OR N
DUAL-IN-LINE PACKAGE
(TOP VIEW)

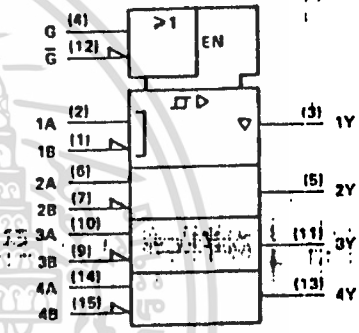


description

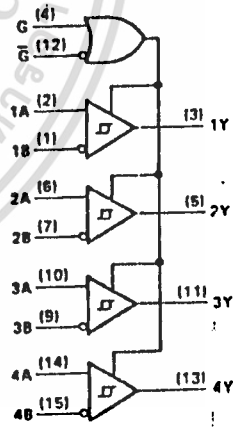
The SN75173 is a monolithic quadruple differential line receiver with three-state outputs. It is designed to meet the requirements of EIA Standards RS-422-A, RS-423-A, and RS-485 and several CCITT recommendations. The device is optimized for balanced multipoint bus transmission at rates up to 10 megabits per second. Each of the two pairs of receivers has a common active-high enable. The device features high input impedance, input hysteresis for increased noise immunity, and input sensitivity of ± 200 millivolts over a common-mode input voltage range of – 12 to 12 volts. The SN75173 is designed for optimum performance when used with the SN75172 or SN75174 quadruple differential line drivers.

The SN75173 is characterized for operation from 0°C to 70°C.

logic symbol



logic diagram (positive logic)



SN75173
QUADRUPLE DIFFERENTIAL LINE RECEIVER

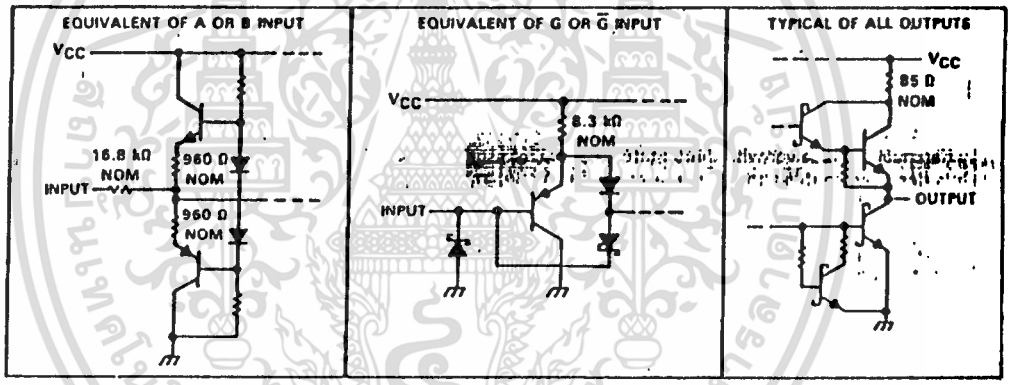
FUNCTION TABLE (EACH RECEIVER)

DIFFERENTIAL A–B	ENABLES G	ENABLES $\overline{G}$	OUTPUT Y
$V_{ID} \geq 0.2$ V	H	X	H
	X	L	H
-0.2 V $< V_{ID} <$ 0.2 V	H	X	?
	X	L	?
$V_{ID} \leq -0.2$ V	H	X	L
	X	L	L
X	L	H	Z

H = high level
L = low level
X = irrelevant
? = indeterminate
Z = high impedance (off)

4

schematics of inputs and outputs



TEXAS INSTRUMENTS
POST OFFICE BOX 655012 • DALLAS, TEXAS 75265

absolute maximum ratings over operating free-air temperature range (unless otherwise noted)

Supply voltage, V_{CC} (see Note 1) . . . . . 7 V
Input voltage, A or B inputs . . . . . ± 25 V
Differential input voltage (see Note 2) . . . . . ± 25 V
Enable input voltage . . . . . 7 V
Low-level output current . . . . . 50 mA
Continuous total dissipation at (or below) 25°C free-air temperature (see Note 3):
D Package . . . . . 950 mW
J Package . . . . . 1025 mW
N Package . . . . . 1150 mW
Operating free-air temperature range . . . . . 0°C to 70°C
Storage temperature range . . . . . −65°C to 150°C
Lead temperature 1,6 mm (1/16 inch) from case for 60 seconds: J package . . . . . 300°C
Lead temperature 1,6 mm (1/16 inch) from case for 10 seconds: D or N package . . . . . 260°C

NOTES: 1. All voltage values, except differential input voltage, are with respect to network ground terminal.
2. Differential input voltage is measured at the noninverting input with respect to the corresponding inverting input.
3. For operation above 25°C free-air temperature, derate the D package to 608 mW at 70°C at the rate of 7.6 mW/°C, the J package to 656 mW at 70°C at the rate of 8.2 mW/°C, and the N package to 736 mW at 70°C at the rate of 9.2 mW/°C. In the J package, SN75173 chips are glass mounted.

recommended operating conditions

	MIN	NOM	MAX	UNIT
Supply voltage, V_{CC}	4.75	5	5.25	V
Common-mode input voltage, V_{IC}			± 12	V
Differential input voltage, V_{ID}			± 12	V
High-level input voltage, V_{IH}	2			V
Low-level input voltage, V_{IL}			0.8	V
High-level output current, I_{OH}			−400	µA
Low-level output current, I_{OL}			16	mA
Operating free-air temperature, T_A	0		70	°C

4 Line Drivers/Receivers

electrical characteristics over recommended ranges of common-mode input voltage, supply voltage, and operating free-air temperature (unless otherwise noted)

PARAMETER		TEST CONDITIONS		MIN	TYP†	MAX	UNIT
V_{TH}	Differential-input high-threshold voltage	V_O = 2.7 V,	I_O = −0.4 mA			0.2	V
V_{TL}	Differential-input low-threshold voltage	V_O = 0.5 V,	I_O = 16 mA	−0.2‡			V
V_{hys}	Hysteresis§				50		mV
V_{IK}	Enable-input clamp voltage	I_I = −18 mA				−1.5	V
V_{OH}	High-level output voltage	V_{ID} = 200 mV,	I_{OH} = −400 μA	2.7			V
V_{OL}	Low-level output voltage	V_{ID} = −200 mV,	I_{OL} = 8 mA			0.45	V
			I_{OL} = 16 mA			0.5	
I_{OZ}	High-impedance-state output current	V_O = 0.4 V to 2.4 V				±20	μA
I_I	Line input current	Other input at 0 V, See Note 4	V_I = 12 V			1	mA
			V_I = −7 V			−0.8	
I_{IH}	High-level enable-input current	V_{IH} = 2.7 V				20	μA
I_{IL}	Low-level enable-input current	V_{IL} = 0.4 V				−100	μA
r_I	Input resistance			12			kΩ
I_{OS}	Short-circuit output current¶			−15		−85	mA
I_{CC}	Supply current	Outputs disabled				70	mA

† All typical values are at V_{CC} = 5 V, T_A = 25°C.
‡ The algebraic convention, in which the less positive (more negative) limit is designated minimum, is used in this data sheet for threshold voltage levels only.
§ Hysteresis is the difference between the positive-going input threshold voltage, V_{T+}, and the negative-going input threshold voltage, V_{T-}. See Figure 4.
¶ Not more than one output should be shorted at a time and the duration of the short-circuit should not exceed one second.
NOTE 4: Refer to EIA Standard RS-422-A and RS-423-A for exact conditions.

switching characteristics, V_{CC} = 5 V, T_A = 25°C

PARAMETER		TEST CONDITIONS		MIN	TYP	MAX	UNIT
t_{PLH}	Propagation delay time, low-to-high-level output	V_{ID} = −1.5 V to 1.5 V, C_L = 15 pF, See Figure 1			20	35	ns
t_{PHL}	Propagation delay time, high-to-low-level output				22	35	ns
t_{PZH}	Output enable time to high level	C_L = 15 pF,	See Figure 2		17	22	ns
t_{PZL}	Output enable time to low level	C_L = 15 pF,	See Figure 3		20	25	ns
t_{PHZ}	Output disable time from high level	C_L = 5 pF,	See Figure 2		21	30	ns
t_{PLZ}	Output disable time from low level	C_L = 5 pF,	See Figure 3		30	40	ns

4 Line Drivers/Receivers

PARAMETER MEASUREMENT INFORMATION

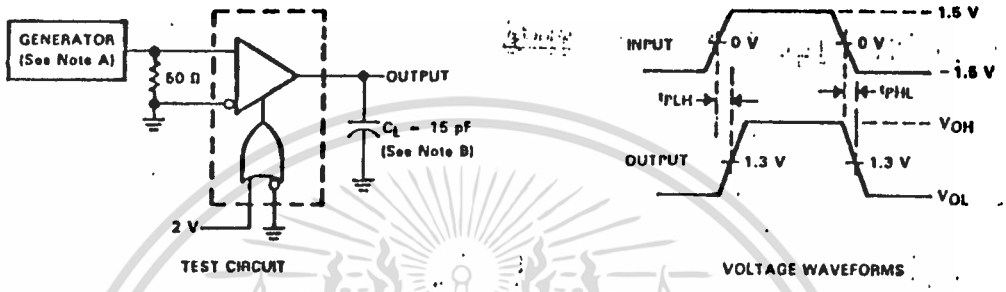


FIGURE 1. t_{PLH}, t_{PHL}

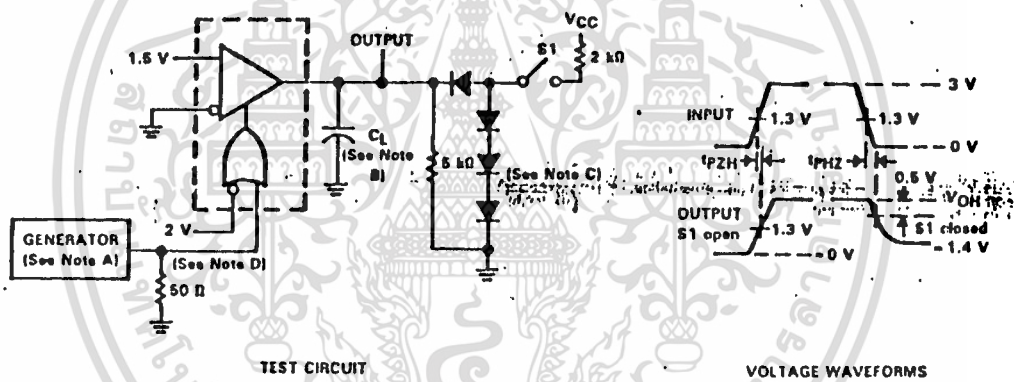


FIGURE 2. t_{PHZ}, t_{PZH}

NOTES: A. The input pulse is supplied by a generator having the following characteristics: PRR ≤ 1 MHz, duty cycle = 50%, t_r ≤ 6 ns, t_f ≤ 6 ns, Z_{out} = 50 Ω.
B. C_L includes probe and jig capacitance.
C. All diodes are 1N916 or equivalent.
D. To test the active-low enable $\overline{G}$, ground G and apply an inverted input waveform to $\overline{G}$.

PARAMETER MEASUREMENT INFORMATION

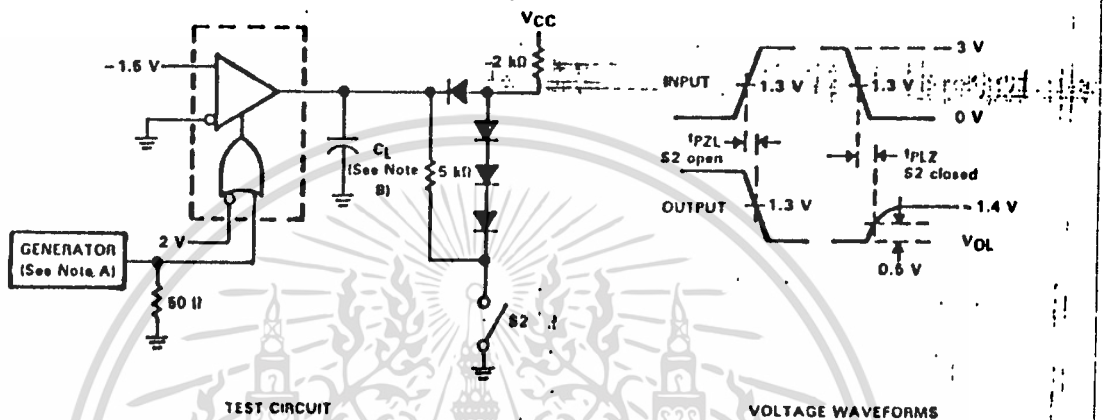


FIGURE 3. t_{PZL}, t_{PLZ}

NOTES: A. The input pulse is supplied by a generator having the following characteristics: PRR ≤ 1 MHz, duty cycle = 50%, t_r ≤ 6 ns, t_f ≤ 6 ns, Z_{out} = 50 Ω.
B. C_L includes probe and jig capacitance.
C. All diodes are 1N916 or equivalent.
D. To test the active-low enable $\overline{G}$, ground G and apply an inverted input waveform to $\overline{G}$.

TYPICAL CHARACTERISTICS

OUTPUT VOLTAGE
vs
DIFFERENTIAL INPUT VOLTAGE

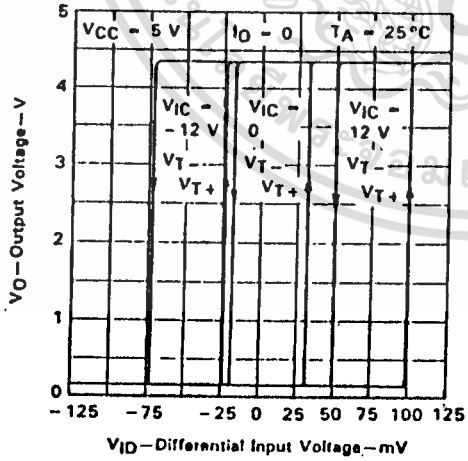


FIGURE 4

HIGH-LEVEL OUTPUT VOLTAGE
vs
HIGH-LEVEL OUTPUT CURRENT

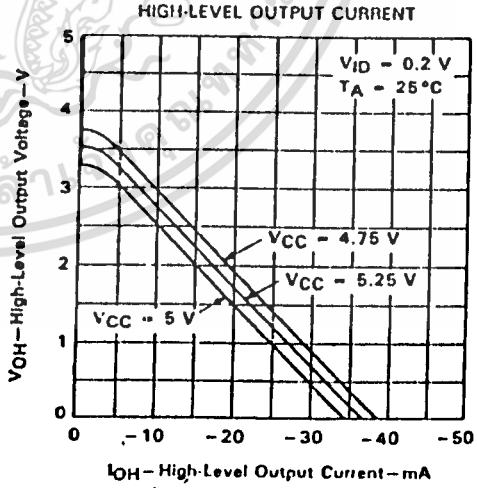


FIGURE 5

TYPICAL CHARACTERISTICS

HIGH-LEVEL OUTPUT VOLTAGE
vs
FREE-AIR TEMPERATURE

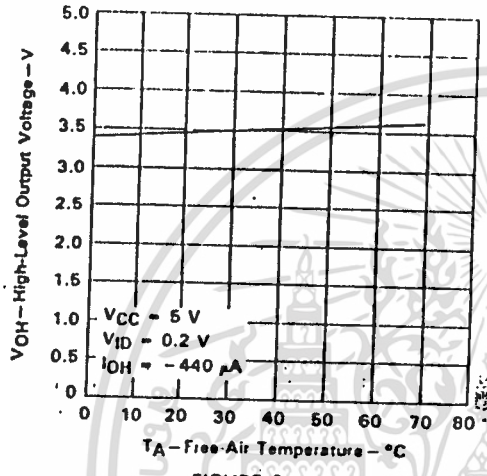


FIGURE 6

LOW-LEVEL OUTPUT VOLTAGE
vs
LOW-LEVEL OUTPUT CURRENT

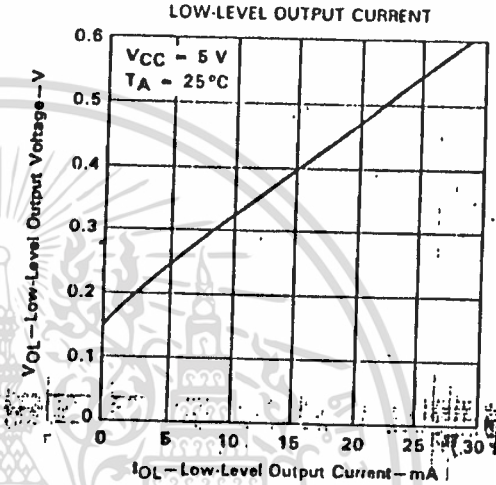


FIGURE 7

LOW-LEVEL OUTPUT VOLTAGE
vs
FREE-AIR TEMPERATURE

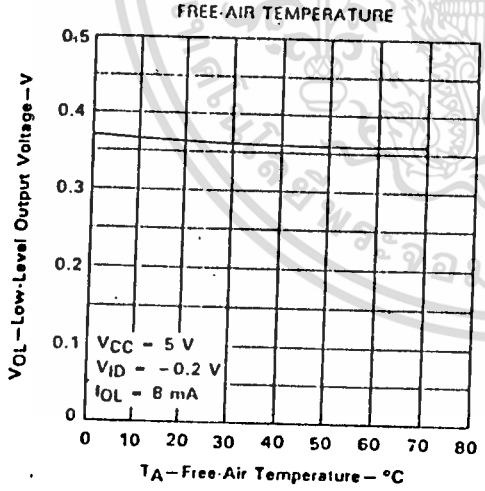


FIGURE 8

OUTPUT VOLTAGE
vs
ENABLE G VOLTAGE

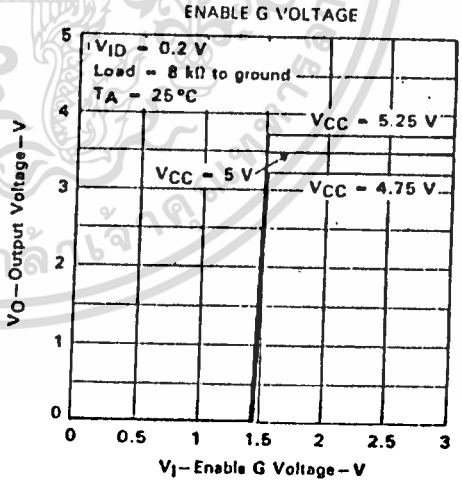


FIGURE 9

TYPICAL CHARACTERISTICS

OUTPUT VOLTAGE
vs
ENABLE G VOLTAGE

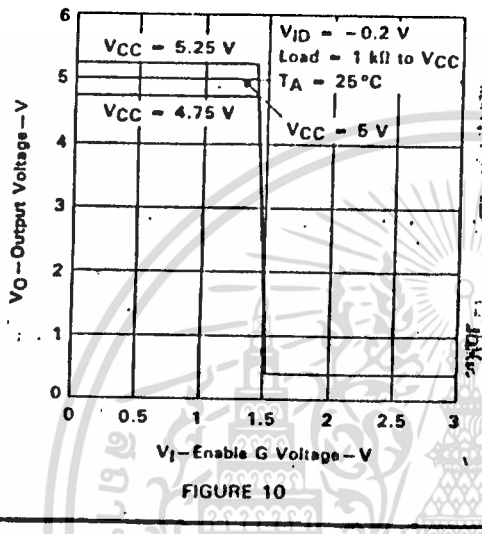


FIGURE 10

INPUT CURRENT
vs
INPUT VOLTAGE

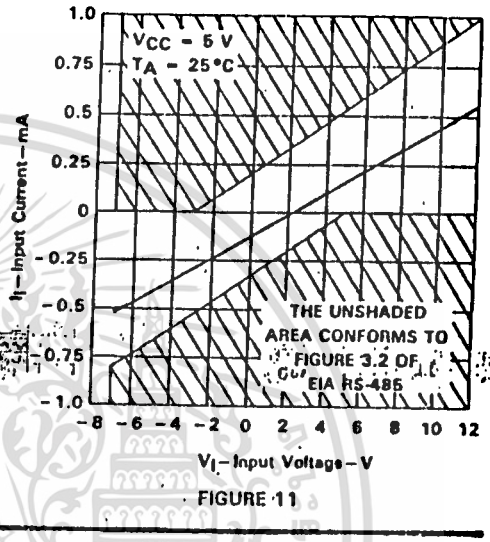
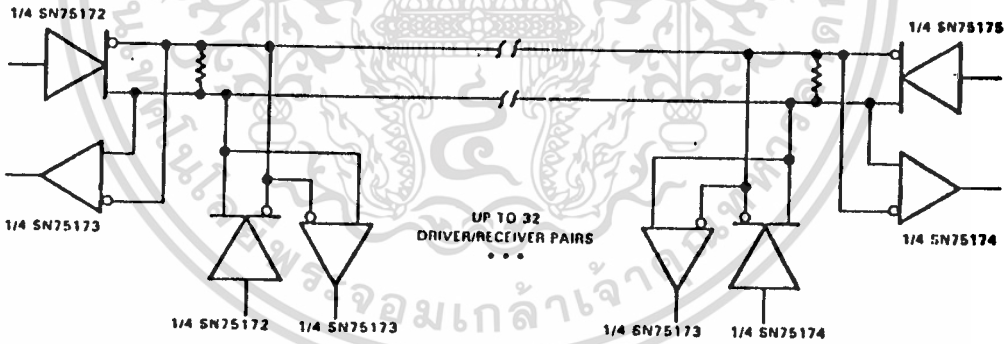


FIGURE 11

TYPICAL APPLICATION



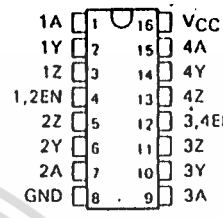
NOTE 4: The line should be terminated at both ends in its characteristic impedance. Stub lengths off the main line should be kept as short as possible.

SN75174
QUADRUPLE DIFFERENTIAL LINE DRIVER

D2601, OCTOBER 1980 – REVISED OCTOBER 1986

- Meets EIA Standards RS-422-A and RS-485 and CCITT Recommendations V.11 and X.27
- Designed for Multipoint Transmission on Long Bus Lines in Noisy Environments
- 3-State Outputs
- Common-Mode Output Voltage Range of −7 V to 12 V
- Active-High Enable
- Thermal Shutdown Protection
- Positive- and Negative-Current Limiting
- Operates from Single 5-V Supply
- Low Power Requirements
- Functionally Interchangeable with MC3487

J OR N DUAL-IN-LINE PACKAGE
(TOP VIEW)



FUNCTION TABLE (EACH DRIVER)

INPUT	ENABLE	OUTPUTS	
		Y	Z
H	H	H	L
L	H	L	H
X	L	Z	Z

H = TTL high level,
L = TLL low level,
X = irrelevant,
Z = High impedance (off)

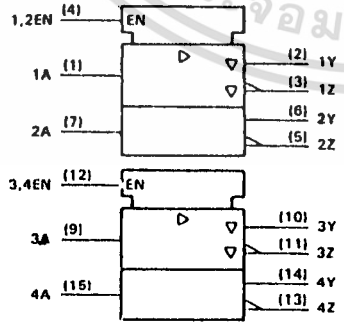
description

The SN75174 is a monolithic quadruple differential line driver with three-state outputs. It is designed to meet the requirements of EIA Standards RS-422-A and RS-485 and CCITT Recommendations V.11 and X.27. The device is optimized for balanced multipoint bus transmission at rates up to 4 megabaud. Each driver features wide positive and negative common-mode output voltage ranges making it suitable for party-line applications in noisy environments.

The SN75174 provides positive- and negative-current limiting and thermal shutdown for protection from line fault conditions on the transmission bus line. Shutdown occurs at a junction temperature of approximately 150°C. This device offers optimum performance when used with the SN75173 or SN75175 quadruple differential line receivers.

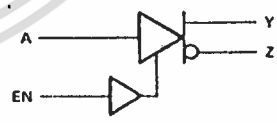
The SN75174 is characterized for operation from 0°C to 70°C.

logic symbol†

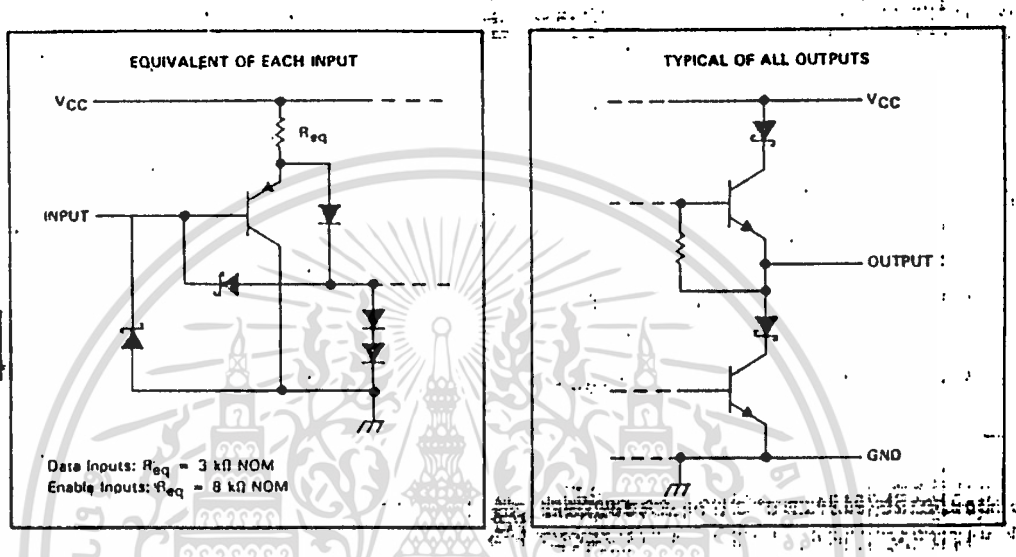


† This symbol is in accordance with ANSI/IEEE Std 91-1984 and IEC Publication 617-12.

logic diagram, each driver (positive logic)



schematics of inputs and outputs



absolute maximum ratings over operating free-air temperature (unless otherwise noted)

Supply voltage, V_{CC} (see Note 1) . . . . . 7 V
Input voltage . . . . . 5.5 V
Continuous total dissipation at (or below) 25°C free-air temperature (see Note 2):
J package . . . . . 1375 mW
N package . . . . . 1625 mW
Operating free-air temperature . . . . . 0°C to 70°C
Storage temperature range . . . . . −65°C to 150°C
Lead temperature 1,6 mm (1/16 inch) from case for 60 seconds: J package . . . . . 300°C
Lead temperature 1,6 mm (1/16 inch) from case for 10 seconds: N package . . . . . 260°C

NOTES: 1. All voltage values are with respect to the network terminal.
2. For operation above 25°C free-air temperature, derate the J package to 880 mW at 70°C at the rate of 11.0 mW/°C and the N package to 1040 mW at 70°C at the rate of 13.0 mW/°C.

recommended operating conditions

	MIN	NOM	MAX	UNIT
Supply voltage, V_{CC}	4.75	5	5.25	V
High-level input voltage, V_{IH}	2			V
Low-Level input voltage, V_{IL}			0.8	V
Common-mode output voltage, V_{OC}			−7 to 12	V
High-level output current, I_{OH}			−60	mA
Low-level output current, I_{OL}			60	mA
Operating free-air temperature, T_A	0		70	°C

electrical characteristics over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted)

PARAMETER		TEST CONDITIONS		MIN	TYP†	MAX	UNIT
V_{IK}	Input clamp voltage	I_I = −18 mA				−1.5	V
V_{OH}	High-level output voltage	V_{IH} = 2 V, I_{OH} = −33 mA	V_{IL} = 0.8 V,		3.7		V
V_{OL}	Low-level output voltage	V_{IH} = 2 V, I_{OL} = 33 mA	V_{IL} = 0.8 V,		1.1		V
V_O	Output voltage	I_O = 0		0		6	V
$\|V_{OD1}\|$	Differential output voltage	I_O = 0		1.5		6	V
$\|V_{OD2}\|$	Differential output voltage	R_L = 100 Ω,	See Figure 1	½ V_{OD1}			V
		R_L = 54 Ω,	See Figure 1	1.5	2.5	5	V
V_{OD3}	Differential output voltage	See Note 3		1.5		5	V
$\Delta\|V_{OD}\|$	Change in magnitude of differential output voltage‡	R_L = 54 Ω or 100 Ω,	See Figure 1			±0.2	V
V_{OC}	Common mode output voltage					+3 −1	V
$\Delta\|V_{OC}\|$	Change in magnitude of common mode output voltage‡					±0.2	V
I_O	Output current with power off	V_{CC} = 0,	V_O = −7 V to 12 V			±100	μA
I_{OZ}	High-impedance state output current	V_O = −7 V to 12 V				±100	μA
I_{IH}	High-level input current	V_I = 2.7 V				20	μA
I_{IL}	Low-level input current	V_I = 0.5 V				−360	μA
I_{OS}	Short-circuit output current	V_O = −7 V				−250	mA
		V_O = V_{CC}				180	
		V_O = 12 V				500	
I_{CC}	Supply current (all drivers)	No load	Outputs enabled		38	60	mA
			Oututs disabled		18	40	

† All typical values are at V_{CC} = 5 V and T_A = 25°C.
‡ $\Delta|V_{OD}|$ and $\Delta|V_{OC}|$ are the changes in magnitude of V_{OD} and V_{OC}, respectively, that occur when the input is changed from a high level to a low level.
NOTE 3: See EIA Standard RS-485 Figure 3.5, Test Termination Measurement 2.

switching characteristics, V_{CC} = 5 V, T_A = 25°C

PARAMETER		TEST CONDITIONS		MIN	TYP	MAX	UNIT
t_{DD}	Differential-output delay time	R_L = 54 Ω,	See Figure 2		45	65	ns
t_{TD}	Differential-output transition time				80	120	ns
t_{PZH}	Output enable time to high level	R_L = 110 Ω,	See Figure 3		80	120	ns
t_{PZL}	Output enable time to low level	R_L = 110 Ω,	See Figure 4		55	80	ns
t_{PHZ}	Outut disable time from high level	R_L = 110 Ω,	See Figure 3		75	115	ns
t_{PLZ}	Output disable time from low level	R_L = 110 Ω,	See Figure 4		18	30	ns

4 Line Drivers/Receivers

SYMBOL EQUIVALENTS

DATA SHEET PARAMETER	RS-422-A	RS-485
V_O	V_{oa}, V_{ob}	V_{oa}, V_{ob}
$\lvert V_{OD1}\rvert$	V_o	V_o
$\lvert V_{OD2}\rvert$	V_t (R_L = 100 Ω)	V_t (R_L = 54 Ω)
$\lvert V_{OD3}\rvert$		V_t (Test Termination Measurement 2)
$\Delta\lvert V_{OD}\rvert$	$\lvert\lvert V_t\rvert - \lvert\overline{V}_t\rvert\rvert$	$\lvert\lvert V_t\rvert - \lvert\overline{V}_t\rvert\rvert$
V_{OC}	$\lvert V_{os}\rvert$	$\lvert V_{os}\rvert$
$\Delta\lvert V_{OC}\rvert$	$\lvert V_{os} - \overline{V}_{os}\rvert$	$\lvert V_{os} - \overline{V}_{os}\rvert$
I_{OS}	$\lvert I_{sa}\rvert$, $\lvert I_{sb}\rvert$	
I_O	$\lvert I_{xa}\rvert$, $\lvert I_{xb}\rvert$	I_{ia}, I_{ib}

4 Line Drivers/Receivers

PARAMETER MEASUREMENT INFORMATION

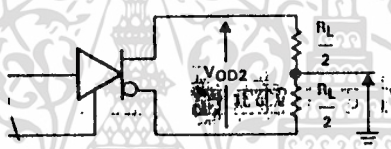
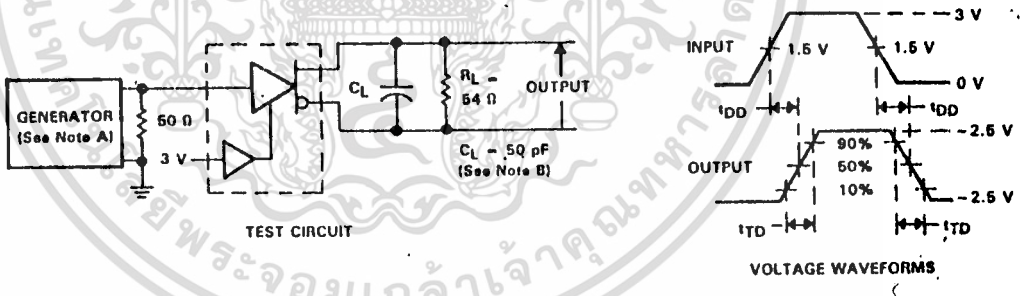


FIGURE 1. DIFFERENTIAL AND COMMON-MODE OUTPUT VOLTAGES



NOTES: A. The input pulse is supplied by a generator having the following characteristics: $t_r \le 5$ ns, $t_f \le 5$ ns, PRR ≤ 1 MHz, duty cycle = 50%, Z_O = 50 Ω.
B. C_L includes probe and stray capacitance.

FIGURE 2. DIFFERENTIAL-OUTPUT DELAY AND TRANSITION TIMES

TEXAS INSTRUMENTS
POST OFFICE BOX 655012 • DALLAS, TEXAS 75265

PARAMETER MEASUREMENT INFORMATION

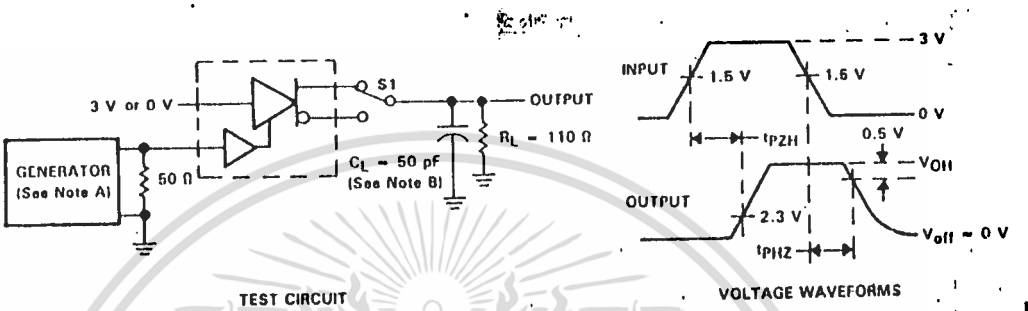


FIGURE 3. t_{PZH} AND t_{PHZ}

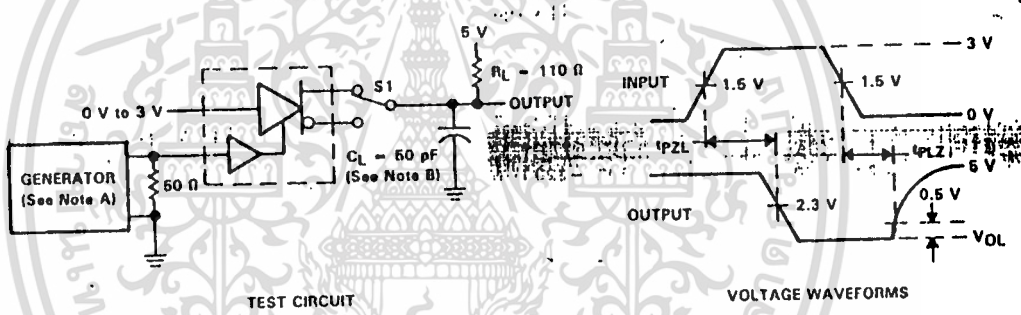


FIGURE 4. t_{PZL} AND t_{PLZ}

NOTES: A. The input pulse is supplied by a generator having the following characteristics: PRR ≤ 1 MHz, duty cycle = 50%, t_r ≤ 5 ns, t_f ≤ 5 ns, Z_o = 50 Ω.
B. C_L includes probe and stray capacitance.

4 Line Drivers/Receivers

TYPICAL CHARACTERISTICS

HIGH-LEVEL OUTPUT VOLTAGE
vs
HIGH-LEVEL OUTPUT CURRENT

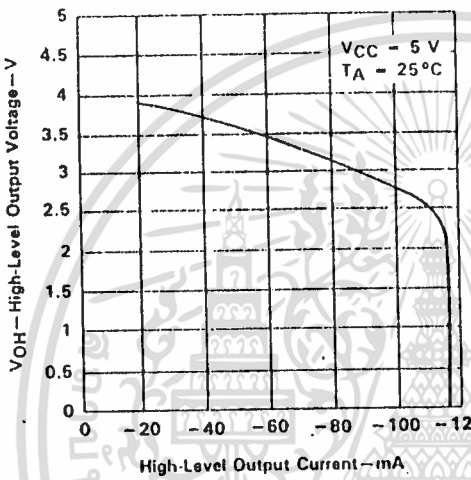


FIGURE 5

LOW-LEVEL OUTPUT VOLTAGE
vs
LOW-LEVEL OUTPUT CURRENT

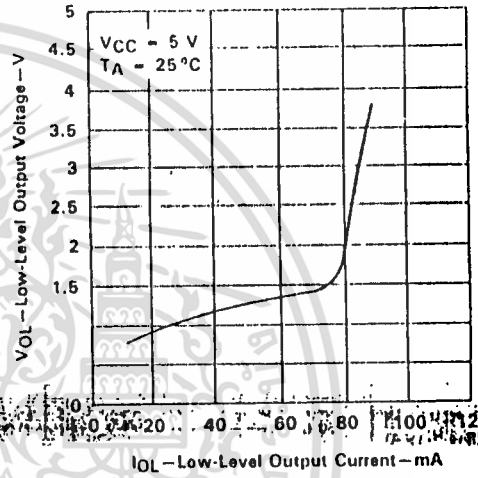


FIGURE 6

DIFFERENTIAL OUTPUT VOLTAGE
vs
OUTPUT CURRENT

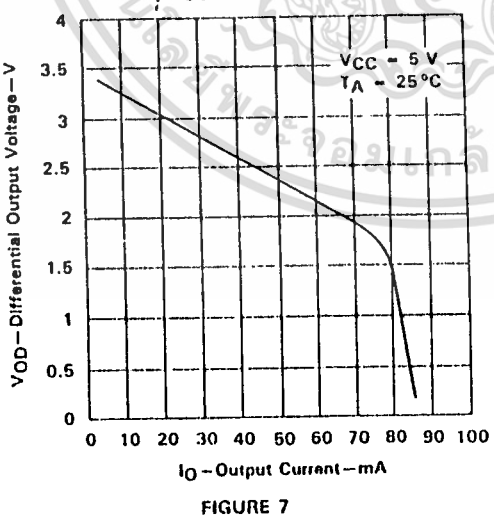


FIGURE 7

OUTPUT CURRENT
vs
OUTPUT VOLTAGE

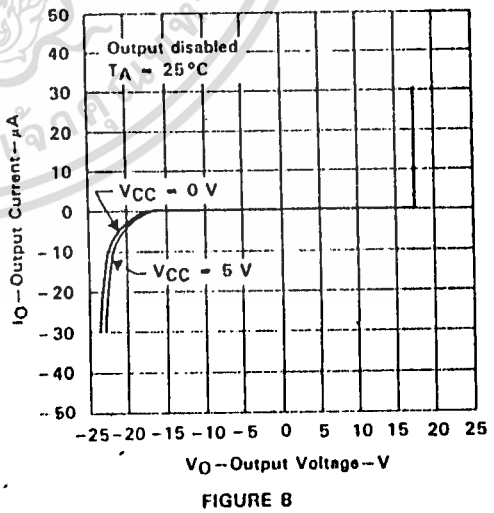


FIGURE 8

TYPICAL CHARACTERISTICS

SUPPLY CURRENT
vs
SUPPLY VOLTAGE

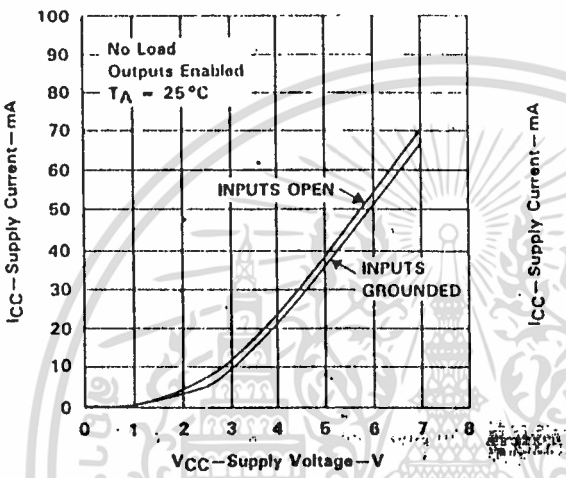


FIGURE 9

SUPPLY CURRENT
vs
SUPPLY VOLTAGE

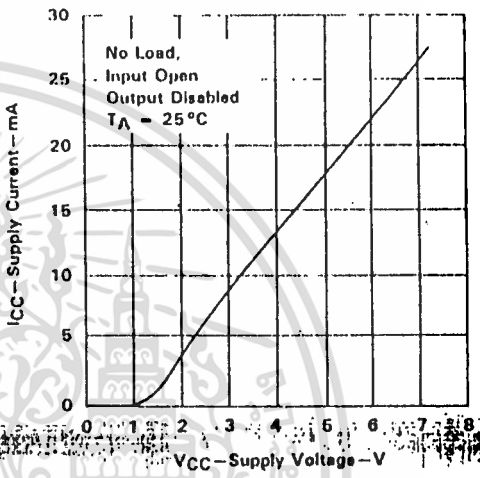
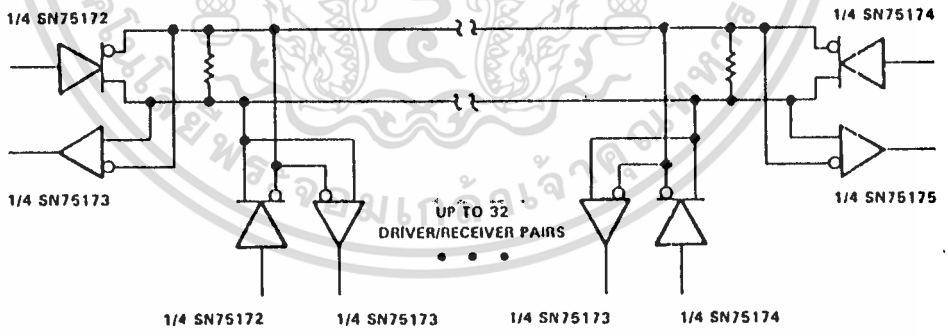


FIGURE 10

TYPICAL APPLICATION



NOTE: The line length should be terminated at both ends in its characteristic impedance. Stub lengths off the main line should be kept as short as possible.

FIGURE 11

กิตติกรรมประกาศ

ขอขอบคุณ อาจารย์ สุพรรณ กุลพาณิชย์ ที่กรุณาให้คำปรึกษาและสนับสนุนเครื่องมือและอุปกรณ์ จนโครงงานนี้สำเร็จและ เสร็จลุล่วงไปด้วยดี

ขอขอบคุณ องค์การโทรศัพท์แห่งประเทศไทย ที่เอื้อเฟื้อข้อมูล จนโครงงานนี้สำเร็จและ เสร็จลุล่วงไปด้วยดี

ขอขอบคุณ คุณแม่ คุณพ่อ ที่ให้กำลังใจและสนับสนุนในการศึกษาแก่ผู้จัดทำมาโดยตลอด

บรรณานุกรม

1. ธานินทร์ ถาวรศาสนวงศ์ , ทินกร ดุ๊ก , การอินเทอร์เฟส (INTERFACE) IBM/PC
2. ชุชัย ธนสารตั้งเจริญ , ทินกร ดุ๊ก , การสื่อสารข้อมูล
3. วารสารคอมพิวเตอร์ (COMPUTER REVIEW),ฉบับที่ 82,มิถุนายน 2534
4. SIGNETICS , Microcontroller
5. OMRON , Sysmac,Mini H-type PCs C20H,C28H,C40H Operation Manual
6. OMRON , Sysmac, C200H-LK401/C500-LK009-V1 PC Link,System Manual