เครื่องตรวจสอบไอซีประเภท EEPROM MEMORY

MEMORY EEPROM EXERCISER

โดย

นายวรพจน์ บูรณพงศ์

นายวุฒิกร สมพัตสร

นายปรีชา จ้อยมาก

นายอธิราช กนกานนท์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

สาขา เทคโนโลยีการวัดคุมทางอุตสาหกรรม

สาขา เทคโนโลยีคอมพิวเตอร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง เครื่องตรวจสอบไอซีประเภท EEPROM MEMORY

MEMORY EEPROM EXERCISER

ผู้จัดทำ 1 นายวรพจน์ บูรณพงศ์ รหัสประจำตัว 34161226 ห้อง 2S

2 นายวุฒิกร สมพัตสร รหัสประจำตัว 34161229 ห้อง 2S

3 นายปรีชา จ้อยมาก รหัสประจำตัว 34162215 ห้อง 2Q

4 นายอธิราช กนกานนท์ รหัสประจำตัว 34162243 ห้อง 2Q

อาจารย์ที่ปรึกษา

(อาจารย์ สุพรรณ กุลพาณิชย์)

ปริญญานิพนธ์ ปีการศึกษา 2535

ภาควิชา เทคโนโลยีการวัดคุมทางอุตสาหกรรม

สาขาวิชา เทคโนโลยีการวัดคุมทางอุตสาหกรรม
เทคโนโลยีคอมพิวเตอร์

คณะ วิศวกรรมศาสตร์

สถาบัน เทคโนโลยีพระจอมเกล้า เจ้าคุณทหารลาดกระบัง

เรื่อง เครื่องตรวจสอบไอซีประเภท EEPROM MEMORY

MEMORY EEPROM EXERCISER

ผู้จัดทำ นายวรพจน์ บูรณพงศ์
นายวุฒิกร สมพัตสร
นายปรีชา จ้อยมาก
นายอธิราช กนกานนท์

อาจารย์ที่ปรึกษา อาจารย์ สุพรรณ กุลพาณิชย์

ปริญญานิพนธ์ MEMORY EEPROM EXERCISER

ปีการศึกษา 2535

นายวรพจน์ บูรณพงศ์
นายวุฒิกร สมพัตสร
นายปรีชา จ้อยมาก
นายอธิราช กนกานนท์

อาจารย์ที่ปรึกษา

อาจารย์ สุพรรณ กุลพาณิชย์

บทคัดย่อ

รายงานเรื่อง MEMORY EEPROM EXERCISER เป็นโครงงานที่ผู้จัดทำสร้างขึ้นมา เพื่อเป็นเครื่องตรวจสอบการทำงานของไอซีประเภท EEPROM MEMORY โดยอาศัยหลักการทำงานของไอซีตาม DATA BOOK ทั้งนี้เพื่อให้คุณภาพของผลิตภัณฑ์เป็น 100 % ก่อนถึงมือลูกค้า ที่นำผลิตภัณฑ์นี้ไปใช้ นอกจากทดสอบการทำงานในสภาวะปกติยังได้ใช้ SOFTWARE สร้างสภาวะที่ไม่เป็นปกติทดสอบตัวไอซีด้วย เช่น มีการลดระดับหรือเพิ่มระดับแรงดันไฟที่จ่ายให้กับตัวไอซี และมีการทดลองเขียนและลบ PROGRAM ที่ใส่ในตัวUNIT เป็นจำนวนหลายๆครั้งเพื่อทดสอบประสิทธิ์ภาพของตัวไอซีด้วยว่า มีความสามารถในด้านนี้ดีเพียงพอหรือไม่

ABSTRACT

THIS PROJECT USE FOR TEST AND EXERCISER INTREGRATE CIRCUIT(IC) IN EEPROM MEMORY TYPE. DESIGN BY REFERING TO DATA SHEET OF DEVICES. FOR MAKE SURE 100 % QUALITY BEFOR DEVIDER TO CUSTOMERS. THE METHOD TO TEST ARE FUNCTIONAL PARAMETRIC AND PROGRAM-ERASE EFFECIENCY THE FUNCTIONAL TEST PERFORM BY SIMULATE TIMING AND TEST PERFORM BY INCREASE OR DECREASE VOLTAGE SUPPLY FOR TESTED UNIT. THE PROGRAM-ERASE EFFECIENCY TEST BY PROGRAM AND ERASE TESTED UNIT IN MANY TIMES.

สารบัญ

คำนำ

การที่จะมีการผลิตผลิตภัณฑ์เพื่อจำหน่ายให้ลูกค้า จำเป็นอย่างยิ่งจะต้องคำนึงถึงคุณภาพของสินค้าที่ทำสำเร็จออกมา เนื่องจากการที่ผลิตภัณฑ์คุณภาพไม่ดี ย่อมก่อให้เกิดความเสียหายในภายหลังได้ ดังนั้นรายงานฉบับนี้จะอธิบายถึงการตรวจสอบไอซีประเภท EEPROM MEMORY โดยจะตรวจสอบในความสามารถทุกด้านของไอซี โดยทั้งนี้จะเน้นเรื่องของความเร็ว ความแม่นยำ และ สามารถใช้ได้จริงกับโรงงานอุตสาหกรรมผลิตไอซี โดยการตรวจสอบ จะทำใน 2 กลุ่มคือ

1 การตรวจสอบด้านการทำงาน FUNCTIONAL TEST

2 การตรวจสอบด้านความสามารถในการเขียนและลบ PROGRAM

คณะผู้จัดทำหวังเป็นอย่างยิ่งว่า รายงานฉบับนี้จะมีประโยชน์ต่อวงการอุตสาหกรรมผลิตไอซี และ สามารถนำหลักการทำงานไปใช้ประโยชน์ต่อไป

คณะผู้จัดทำ

18 มีนาคม 2535

บทนำ

วัตถุประสงค์

รายงานฉบับนี้มีวัตถุประสงค์เพื่อสร้างเครื่องตรวจสอบไอซีประเภท EEPROM MEMORY โดยใช้หลักการให้ง่าย และราคาถูก เพื่อใช้ทดแทนเครื่องตรวจสอบที่ต้องสั่งมาจากต่างประเทศ นอกจากนี้ยังมีวัตถุประสงค์ให้มีความยืดหยุ่นในการใช้โดยเครื่องเดียวสามารถตรวจสอบได้หลายๆ DEVICE

ขอบเขตของปริญญานิพนธ์

จะออกแบบให้เครื่องสามารถใช้ตรวจสอบได้ครั้งละประมาณ 40 ตัว โดยการตรวจสอบจะต้องทำได้ 3 ขั้นตอนการ TEST คือ

1 ต้องตรวจสอบการทำงานของตัวไอซี และบอกตำแหน่งของตัวไอซีที่เสียได้

2 ต้องสามารถตรวจสอบคุณสมบัติเมื่อแรงดันไฟลดและเพิ่มของตัวไอซีได้

3 สามารถตรวจสอบการเขียนและลบPROGRAM ได้ในรอบการทำงานที่ต้องการ

นอกจากนี้ยังมีความสามารถอื่นๆคือ

1 ต้องแสดงผลทางจอภาพได้

2 สามารถแสดงผลการ TEST ออกมาในลักษณะ HARD COPY ได้

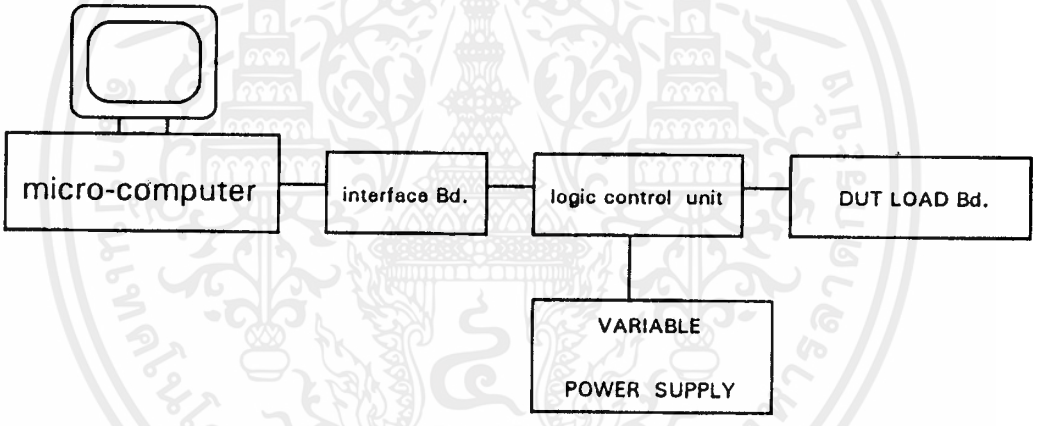
3 สามารถดัดแปลงและแก้ไขขั้นตอนการ TEST ได้ตามต้องการ

4 สามารถกำหนด PATTERN ของความจำที่จะเขียนลงหน่วยความจำได้

บทที่ 1

การทำงานของเครื่องโดยสังเขป

การทำงานของเครื่องอธิบาย BLOCK DIAGRAM ได้ดังนี้



1 PERSONAL COMPUTER

ทำหน้าสร้างสัญญาณ CS,SK,DI(DATA IN) โดยกระทำตามการส่งงานของ SOFTWARE ซึ่งเขียนโดยภาษา C การสร้างสัญญาณจะขึ้นอยู่กับ DEVICE ที่ต้องการ TEST โดยลักษณะของสัญญาณจะอ้างอิงจาก DATA BOOK แต่ละ DEVICE (อยู่ในภาคผนวกของรายงานฉบับนี้) และทำการตรวจสอบสัญญาณ DO(DATA OUT) จาก LOGIC CONTROL BOARD

2 LOGIC CONTROL BOARD

ทำหน้าที่ส่งต่อสัญญาณจาก BLOCK ที่ 1 ให้กับตัว DUT BOARD โดยจะมีการกำหนดตำแหน่งของ DUT(DEVICE UNDER TEST) และจะเลื่อนตำแหน่งโดยคำสั่งที่มาจาก COMPUTER ผ่านทาง INTERFACE BOARD นอกจากนี้ยังเป็น DRIVER BD เพื่อช่วยในการขับสัญญาณในกรณีที่สัญญาณนั้นต่อกับ DUT หลายๆตัว ในชุดLOGIC CONTROL BOARD แบ่งเป็นกลุ่มๆ คือ DI DRIVER,CS DRIVER และ DO RECEIVER และยังเป็นสัญญาณแสดงจังหวะการทำงานของเครื่องไปยัง FRONT PANEL

3 INTERFACE BOARD

เป็น OUTPUT PORT ที่จะทำการสั่งการให้กับ LOGIC CONTROL BOARD ทำงานตาม STEP ของ SOFTWARE และเป็นตัว INPUT PORT สัญญาณ DO(DATA OUT) จาก LOGIC CONTROL BOARD ให้กับ COMPUTER ด้วย

4 DUT BOARD

เป็น BOARD ที่ใช้ใส่UNITที่ต้องการตรวจสอบ โดยจะมีการจัดตำแหน่งของ DUT ใน BO ในลักษณะของ MATRIX เพื่อสามารถกำหนดตำแหน่งแต่ละตัวใน DUT BOARD รายละเอียดจะได้กล่าวในบทต่อไป

5 VARIABLE POWER SUPPLY

เป็น POWER SUPPLY ซึ่งสามารถจ่ายไฟให้กับ DUT ได้ตาม PROGRAM กำหนด

STEP การทำงาน

กรณี WRITE PROGRAM

1 COMPUTER จะทำการสร้างสัญญาณ CS,DI แล้วส่งออกทาง OUTPUT PORT ที่ BD INTERFACE และในขณะเดียวกันก็จะกำหนดตำแหน่งของ DUT ที่ต้องการ PROGRAM ด้วย โดยจะออกที่ OUTPUT PORT ที่ BD INTERFACE เช่นกัน

2 สัญญาณจากข้อ 1 จะส่งมายัง LOGIC CONTROL BOARD ซึ่ง BOARD นี้จะทำงานเป็น 2 ส่วนคือ สัญญาณกำหนดตำแหน่ง DUT จะถูก LATCH ไว้และจะจัดการเป็นสัญญาณ CS1-12 หรือ DI1-12 โดยขึ้นอยู่กับสัญญาณกำหนดตำแหน่งว่าเป็นเท่าใด และส่งออกไปยัง DUT BOARD ส่งสัญญาณ DI และ CS ที่แท้จริงก็จะถูกขับออกไปด้วยวงจร DRIVER ที่อยู่ใน BD LOGIC CONTROL นี้ และส่งไปยังตำแหน่ง DUT ที่ต้องการด้วย

กรณี READ PROGRAM

1 COMPUTER จะสร้างสัญญาณกำหนดตำแหน่ง DUT ที่ต้องการอ่านข้อมูลและส่งสัญญาณCS ออกมาส่ง OUTPUT PORT ที่ INTERFACE BOARD

2 INTERFACE BOARD ส่งสัญญาณไปยัง LOGIC CONTROL BOARD ซึ่ง BD นี้ ก็จะทำให้ DUT ในตำแหน่งที่กำหนดทำงาน DATA ภายใน UNIT ตัวนั้นจะออกมาเข้าที่ LOGIC CONTROL BD ซึ่ง LOGIC CONTROL BOARD ก็จะแปลง INPUT ซึ่งเป็นแบบ SERIAL ให้เป็น PARAWEL แล้วส่งออกไปให้ COMPUTER โดยผ่านทาง INPUT PORT ใน INTERFACE BOARD

3 COMPUTER จะทำการประมวลผลและแสดงผลบนจอภาพ

บทที่ 2

การทำงานของ IC ประเภท EEPROM

IC ประเภท EEPROM สามารถแบ่งเป็นกลุ่มใหญ่ๆ ได้ 2 กลุ่มคือ

1 ประเภท TRUE C

2 ประเภท NON TRUE C

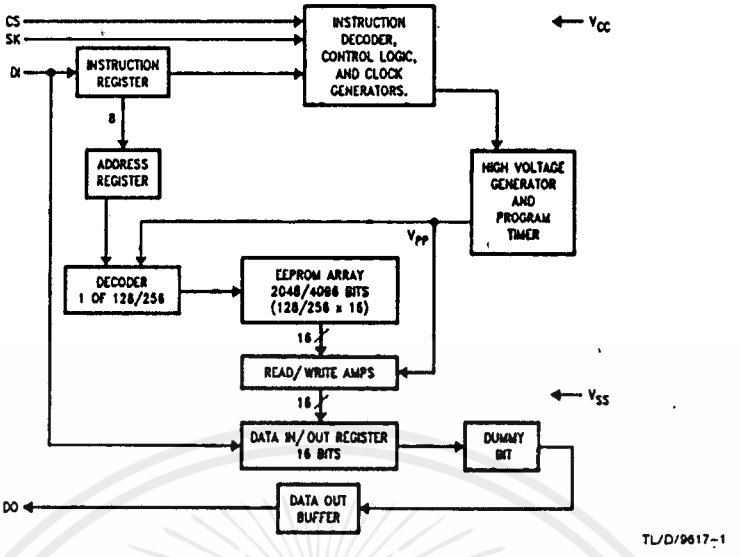
1 ประเภท TRUE C

NMC93C56/C66 2048-BIT/4096-BIT SERIAL ELECTRICALLY ERASABLE PROGRAMMABLE MEMORIES

NMC93C56/NMC93C66 เป็น ไอซีหน่วยความจำลบได้ด้วยไฟฟ้าแบบ CMOS ซึ่งมีหน่วยความจำ 2048 และ4096 บิท และแบ่งเป็น 128/256 16 บิท รีซิสเตอร์ การสร้างใช้กระบวนการ FLOATING-GATE CMOS ซึ่งใช้ได้ในความเร็วสูงและระดับแรงดันไฟต่ำ

ข้อมูลที่จะใส่เข้าไปยังหน่วยความจำจะป้อนแบบ SERIAL หลังจาก INSTRUCTION ADDRESS ที่ขา DI และ ข้อมูลที่อ่านจากหน่วยความจำจะออกมาจาก PIN DATA OUT (DO) ความถี่ SK (SERIAL DATA CLOCK) จะเปลี่ยนในรูปสัญญาณนาฬิกาจาก LOGIC"0" ไป LOGIC"1" เพื่อเลื่อนข้อมูลทั้งเข้าและออก การต่อ SERIAL INTERFACE แบบ MICROWIRE ซึ่งจะเหมือนกับ SERIAL INTERFACE ของ ไมโครคอนโทรลเลอร์และไมโครโพเซสเซอร์ทั่วไป มี INSTRUCTION CODE ที่ใช้อยู่ 7 CODE อ่าน ลบและเขียน ลบ ลบทั้งหมด เขียน เขียนทั้งหมด และห้ามลบและเขียน NMC93C56/66 ไม่จำเป็นต้องลบก่อนเขียนหรือเขียนทั้งหมด สัญญาณ BUSY บน DO PIN แสดงการสมบูรณ์ของการโปรแกรม UNIT ทุกตัวที่ส่งขายตามท้องตลาดจะมีสภาวะเป็นว่าง คือ จะมี LOGIC"1" ทุกบิท

Block Diagram



<u>การทำงานของตัว UNIT :</u>

7 INSTRUCTION CODE ที่แสดงไว้ในตารางที่ 2.2 ให้สังเกตุว่า MBS ของทุก INSTRUCTION จะเป็น LOGIC"1" ซึ่งจะเห็นเป็นบิทเริ่มแรกของ INTERFACE SEQUENCE และบิทต่อๆไป 10 บิท จะเป็น OP CODE และ 8 บิท ที่จะระบุ ADDRESS เพื่อเลือก REGISTER

<u>การอ่านข้อมูล READ :</u>

สภาวะ READ INSTURCTION จะเกิดข้อมูลส่งออกที่ขา DO หลังจากได้รับ READ INSTURCTION และ ADDRESS ที่ระบุจะมีข้อมูลส่งจากหน่วยความจำของ REGISTER ที่ถูกเลือกไปยัง 16 บิท SERIAL OUT REGISTER และมี DUMMY บิท เป็น LOGIC"0" จะถูกส่งออกมาก่อนและการเลื่อนของข้อมูลจะเป็นตามจังหวะการเปลี่ยนสภาวะจาก"0"เป็น"1" ของสัญญาณ SK

<u>การลบและการเขียน EARSE/WRITE ENABLE (EWEN) :</u>

เมื่อแรงดันไฟซัพพลายจ่ายให้กับตัวยูนิต จะเป็นการตั้งสภาวะ ERASE/WRITE DISABLE (EWDS) ดังนั้น ถ้ามีความต้องการโปรแกรมจะต้องให้คำสั่ง ERASE/WRITE (EWEN) ไปก่อนหนึ่งครั้งและสภาวะพร้อมโปรแกรม จะคงสภาวะเช่นนั้นจนกว่าจะมีคำสั่ง

ERASE/WRITE DIABLE (EWDS) หรือยกเลิกการจ่าแรงดันไฟ SUPPLY ให้กับตัว UNIT

การลบ ERASE :

การลบ จะโปรแกรมทุกบิทใน REGISTOR ที่ระบุด้วย LOGIC"1" และหลังจากบิทสุดท้ายของ ADDRESS สัญญาณ CHIP SELECT จะเปลี่ยนสภาวะเป็น "0"ซึ่ง TAILING EDGE ของสัญญาณ CS นี้จะสร้าง SELF-TIMED PROGRAMMING CYCLE และที่ขา DO จะแสดงสัญญาณ READY/BUSY เพื่อบอกสภาวะของตัว UNIT ถ้า CS เป็น HIGH หลังจาก 250NS(TCS) ถ้าขา DO เป็น LOGIC"0" จะแสดงว่าการโปรแกรมยังไม่สมบูรณ์ ถ้า DO เป็น LOGIC"1" แสดงว่า ADDRESS REGISTER ที่ระบุมีการโปรแกรมเรียบร้อยแล้ว และยูนิตพร้อมที่จะทำคำสั่งอื่นต่อไป

การลบทั้งหมด ERASE ALL (ERAL) :

คำสั่ง ERAL จะทำการโปรแกรมทุก REGISTER ในหน่วยความจำให้มี LOGIC เป็น"1"การลบทั้งหมดจะคล้ายกับคำสั่ง ERASE ยกเว้นความแตกต่างที่ OP-CODE และที่ขา DO ก็จะมีสัญญาณ READY/BUSY ซึ่งจะบอกสภาวะของตัวยูนิตหลังจากเวลา 250NS (TCS)

การเขียนทั้งหมด WRITE ALL (WRAL) :

คำสั่ง WRAL เป็นการโปรแกรมทุก REGISTER ด้วยรูปแบบข้อมูลซึ่งระบุในคำสั่งเหมือนกับการเขียน ขา DO จะแสดงสภาวะ READY/BUSY

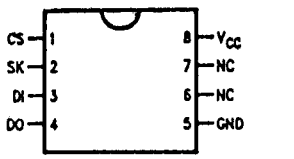
การยกเลิกการลบ/เขียน ERASE/WRITE DISABLE(EWDS) :

เพื่อป้องกันอุบัติเหตุหรือสิ่งผิดปกติที่จะรบกวนข้อมูลในตัว UNIT คำสั่ง ERASE/WRITE DISABLE (EWDS) จะยกเลิกคำสั่งโปรแกรมทุกอย่าง ยกเว้นคำสั่ง READ ซึ่งจะเป็นอิสระจากคำสั่ง EWEN และ EWDS

รายละเอียดเพิ่มเติมอยู่ในส่วน DATA SHEET

Connection Diagrams

Dual-In-Line Package (N)



TL/D/9617-2

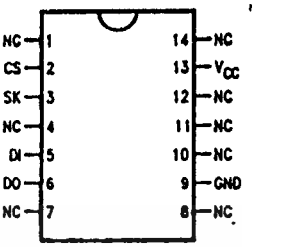
Top View

See NS Package Number N08E

Pin Names

CS	Chip Select
SK	Serial Data Clock
DI	Serial Data Input
DO	Serial Data Output
GND	Ground
V_{CC}	Power Supply

SO Package (M)



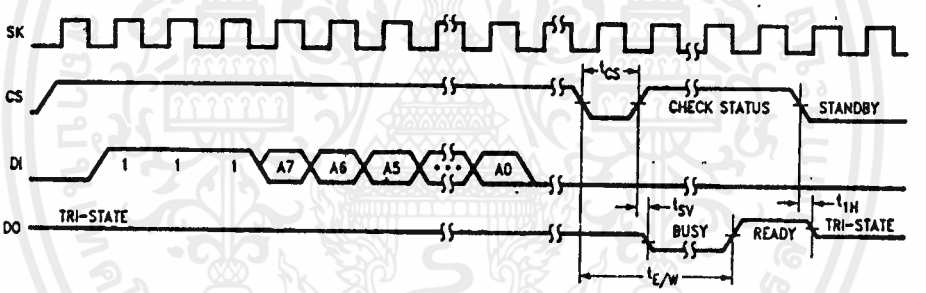
TL/D/9617-3

Top View

See NS Package Number M14A

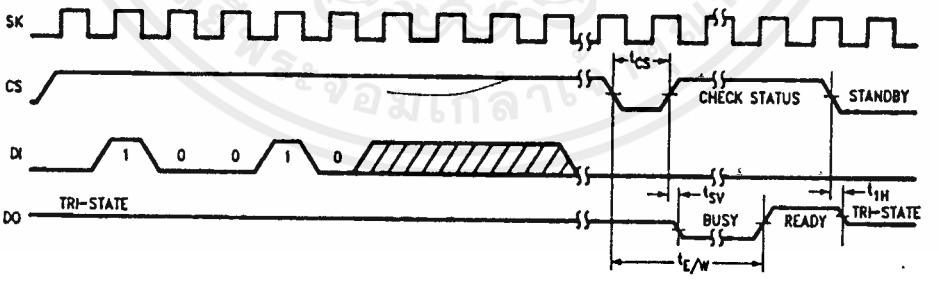
Timing Diagrams (Continued)

ERASE:



TL/D/9617-10

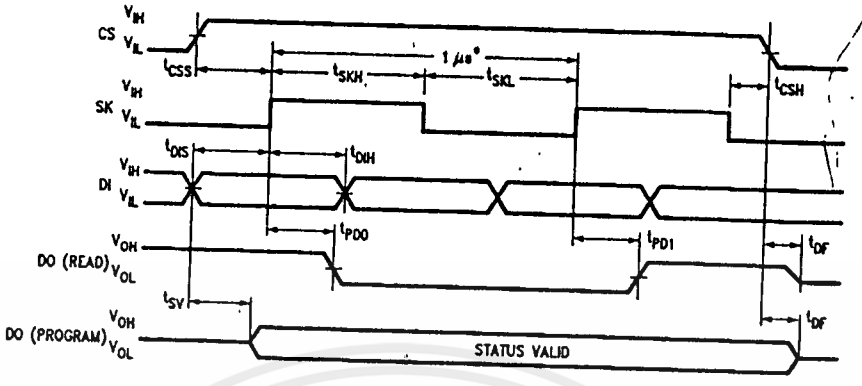
ERAL:



TL/D/9617-11

Timing Diagrams

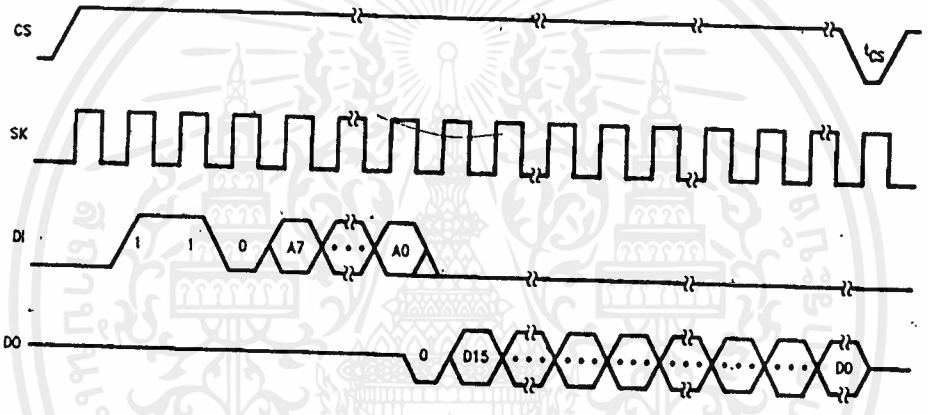
Synchronous Data Timing



*This is the minimum SK period (Note 2).

TL/D/9617–4

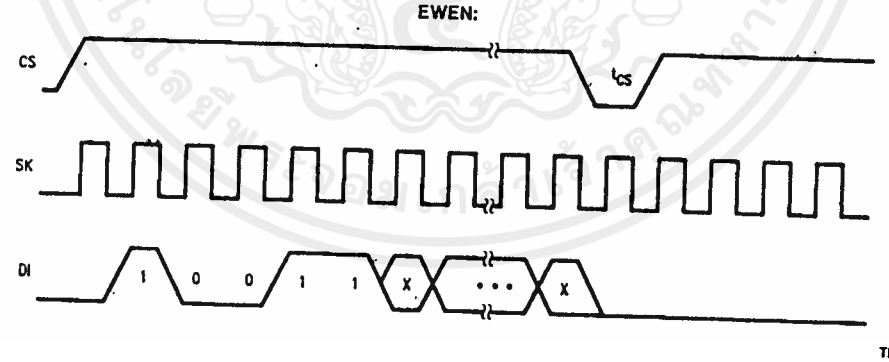
READ:



*Address bit A_7 becomes a "don't care" for NMC93C56.

TL/D/9617–5

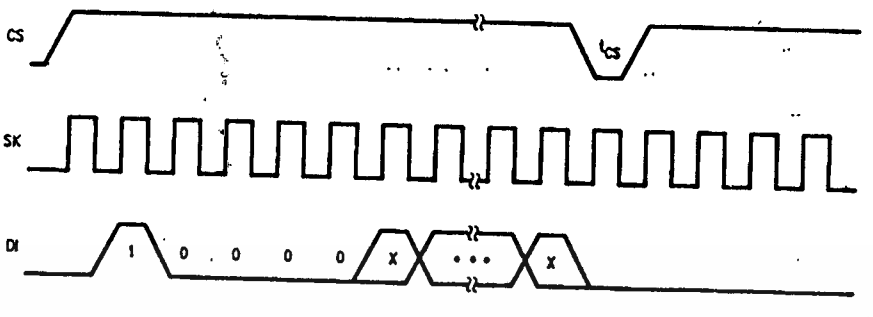
EWEN:



TL/D/9617–6

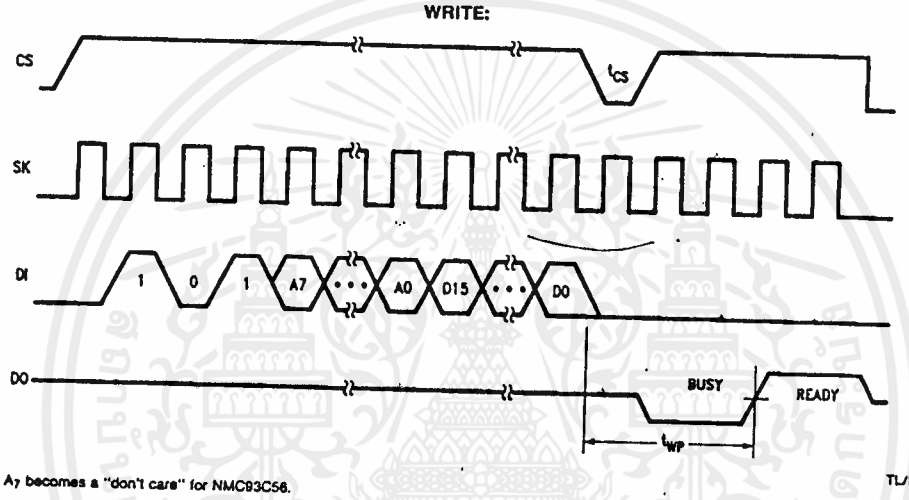
Timing Diagrams (Continued)

EWDS:



TL/D/9617–7

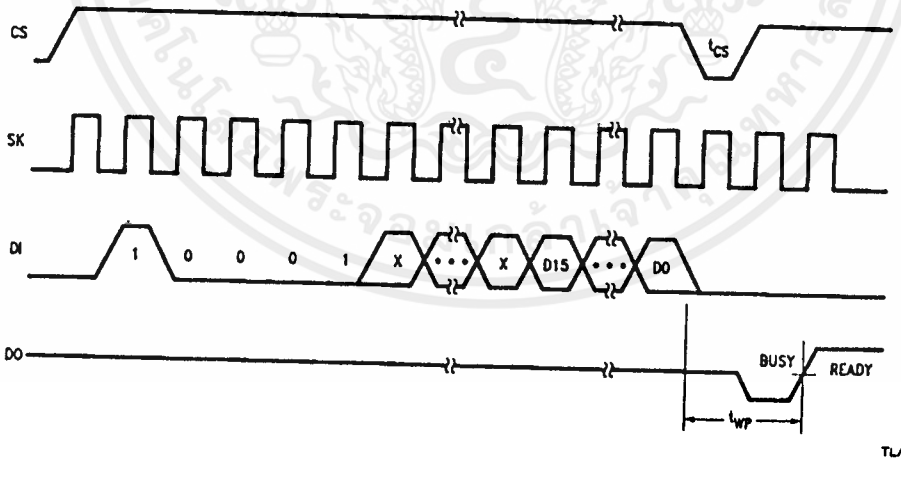
WRITE:



*Address bit A_7 becomes a "don't care" for NMC93C56.

TL/D/9617–8

WRAL:



TL/D/9617–9

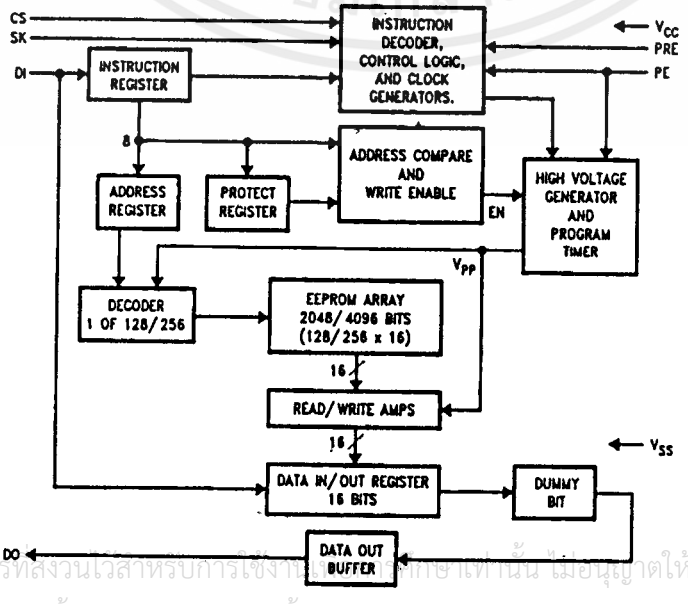
2 ประเภท NON TRUE C

NMC93CS56/CS66 เป็น 2048/4096 บิท 128/256 REGISTERS จะมี REGISTERS บางส่วน(ไม่มากกว่า REGISTER ทั้งหมดในตัว UNIT) สามารถป้องกันข้อมูลภายในโดยการโปรแกรมไว้ใน REGISTER ซึ่งเรียกหน่วยความจำส่วนนั้นว่า "PROTECT REGISTER" ADDRESS แรกใน REGISTER จะถูกป้องกัน ADDRESS นี้จะโดนล็อกไว้ใน DEVICE ดังนั้น REGISTER เหล่านั้นจะเป็นข้อมูลถาวรในตัว UNIT ดังนั้นภายหลังถ้ามีการโปรแกรมและมี ADDRESS ซึ่งกลายเป็น ADDRESS ที่ถูกป้องกันไว้แล้วจะไม่รับข้อมูลเหล่านั้น

คำสั่ง READ จะโหลด ADDRESS ของ REGISTER แรกซึ่งจะอ่านไปใน 8 บิท ADDRESS POINTER หลังจากนั้นข้อมูลจะส่งออกในลักษณะ SERIAL ออกทางขา DO โดยอัตโนมัติ และจะเลื่อนไปยัง REGISTER ถัดไป และจะส่งข้อมูลออกมาจนหมด ในทำนองเดียวกัน การป้อนข้อมูลสามารถทำเป็นแถวข้อมูลได้เช่นกัน

การเขียนสามารถทำได้เองโดยไม่จำเป็นต้อง ERASE ก่อน การเขียนจะทำได้เมื่อ PIN ที่ 6(PROGRAM RNABLE) อยู่ในสภาวะ HIGH ถ้า ADDRESS ของREGISTER ที่เขียนน้อยกว่า ADDRESS ใน "PROTECT REGISTER" ข้อมูลจะเขียน 16 บิท เข้าไป 1 REGISTER ของ 128/256 REGISTERS ถ้าสัญญาณ CS เป็น HIGH ภายหลังการเขียนโปรแกรม ขาDO จะแสดง READY/BUSY ข้อมูลภายในตัว UNITS สามารถคงอยู่มากกว่า 40 ปี

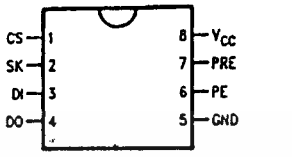
Block Diagram



Connection Diagrams

PIN OUT:

Dual-In-Line Package (N)



TL/D/9209-1

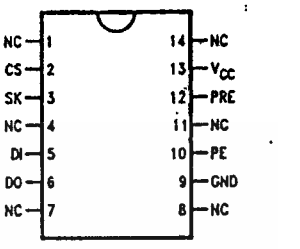
Top View

See NS Package Number N08E

Pin Names	
CS	Chip Select
SK	Serial Data Clock
DI	Serial Data Input
DO	Serial Data Output
GND	Ground
PE	Program Enable
PRE	Protect Register Enable
V_{CC}	Power Supply

PIN OUT:

SO Package (M)



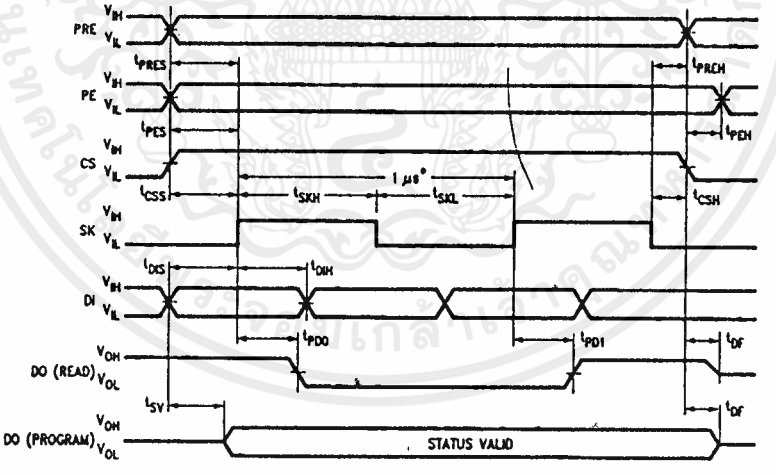
TL/D/9209-2

Top View

See NS Package Number M14A

Timing Diagrams

Synchronous Data Timing

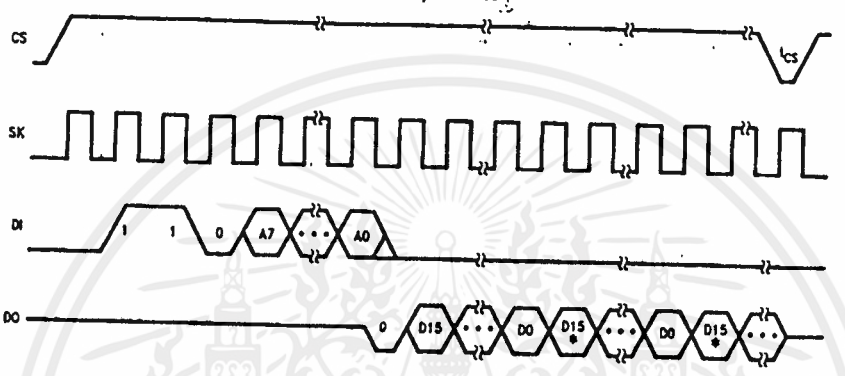


TL/D/9209-4

*This is the minimum SK period (See Note 2).

Timing Diagrams (Continued)

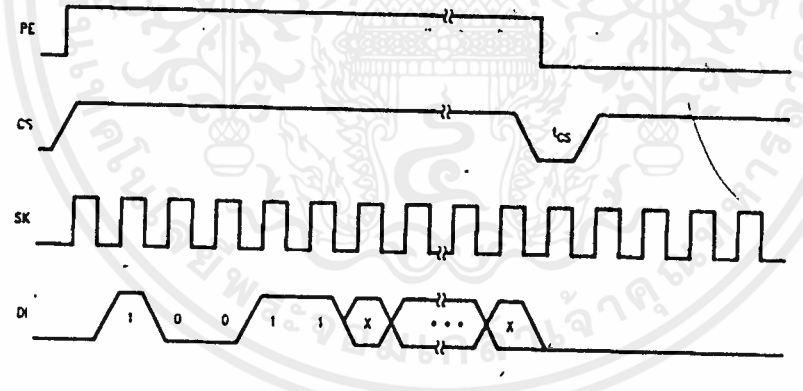
READ:*
PRE = 0, PE = X



*Address bit A_7 becomes a "don't care" for NMC93CS56.
*The memory automatically cycles to the next register.

TL/D/9209–5

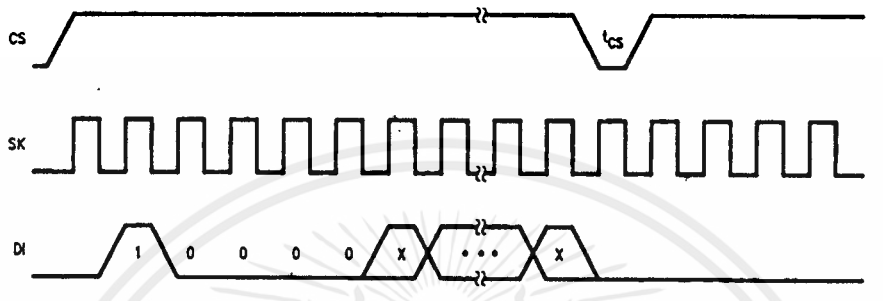
WEN:
PRE = 0, DO = TRI-STATE



TL/D/9209–6

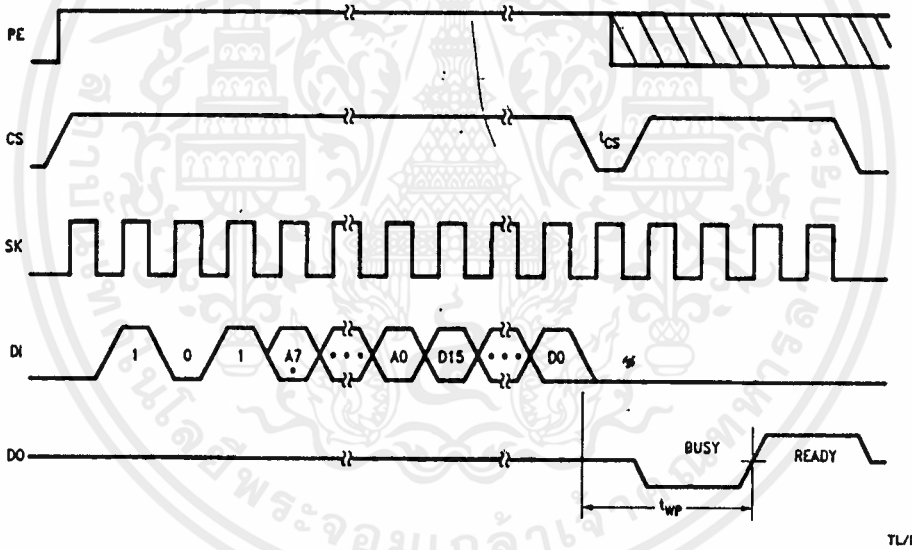
Timing Diagrams (Continued)

WDS:
PRE = 0, PE = X, DO = TRI-STATE



TL/D/9209–7

WRITE:*
PRE = 0

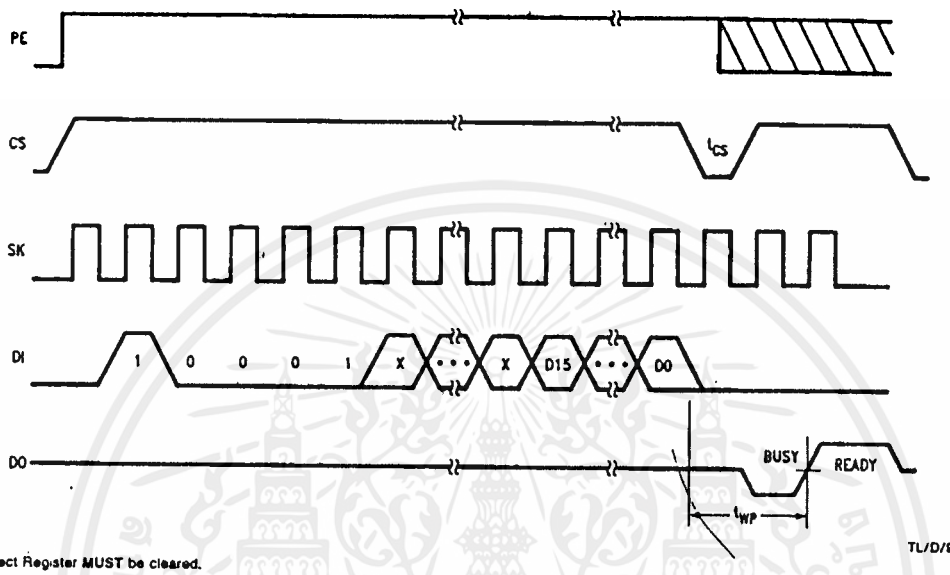


TL/D/9209–8

*Address bit A_7 becomes a "don't care" for NMC93CS56.

Timing Diagrams (Continued)

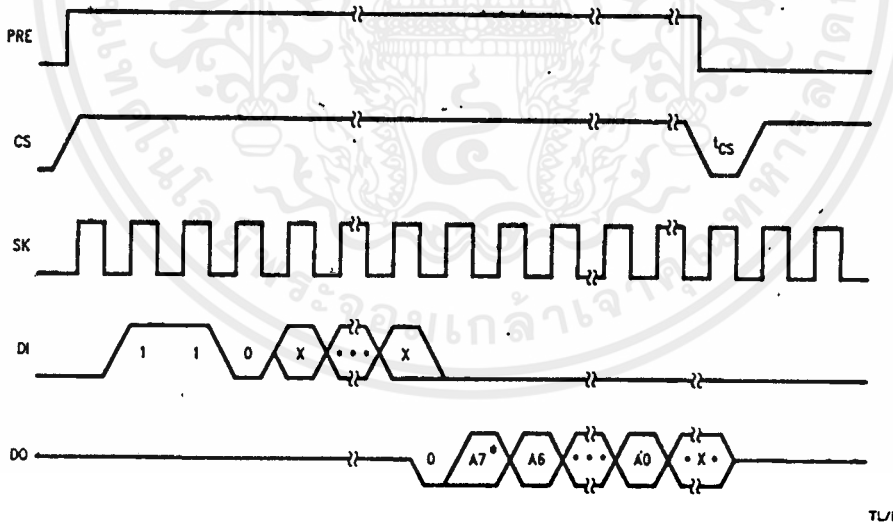
WRALL:*
PRE = 0



TL/D/9209–9

*Protect Register MUST be cleared.

PREAD:*
PE = X

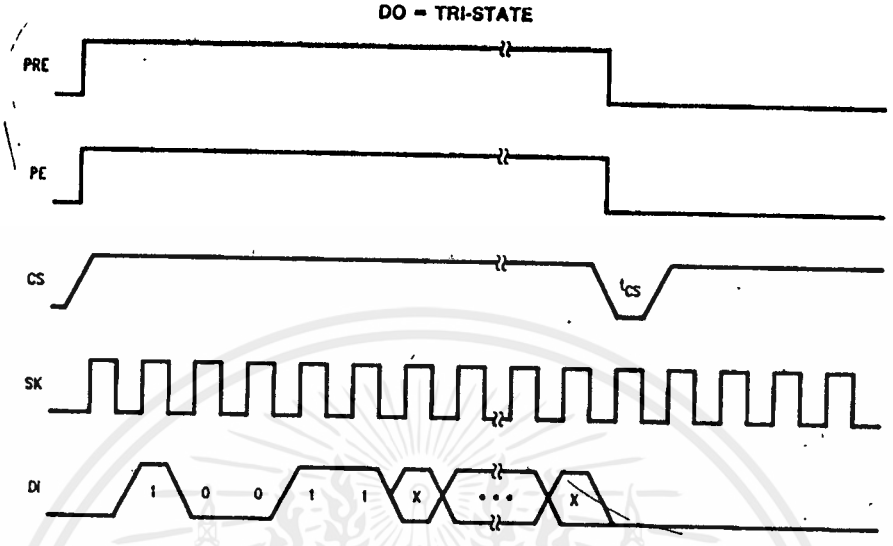


TL/D/9209–10

*Address bit A_7 becomes a "don't care" for NMC93CS56.

Timing Diagrams (Continued)

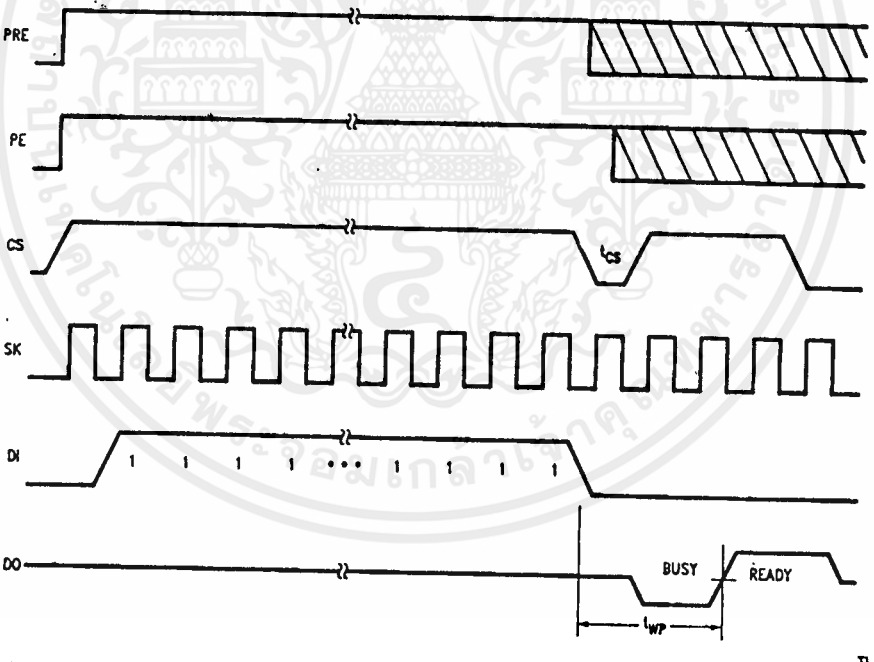
PREN:*
DO = TRI-STATE



*A WEN cycle must precede a PREN cycle.

TL/D/9209-1

PRCLEAR:*

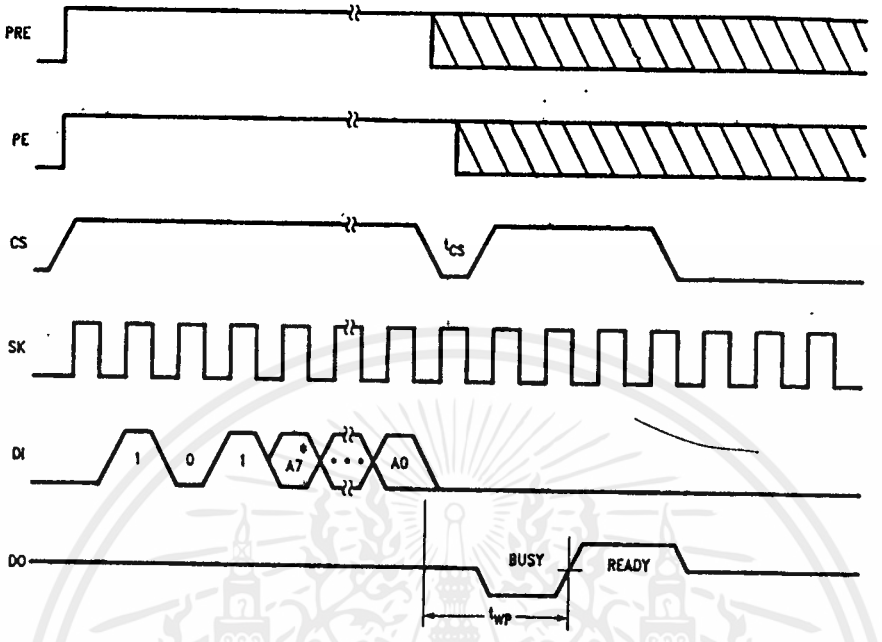


*A PREN cycle must immediately precede a PRCLEAR cycle.

TL/D/9209-1

Timing Diagrams (Continued)

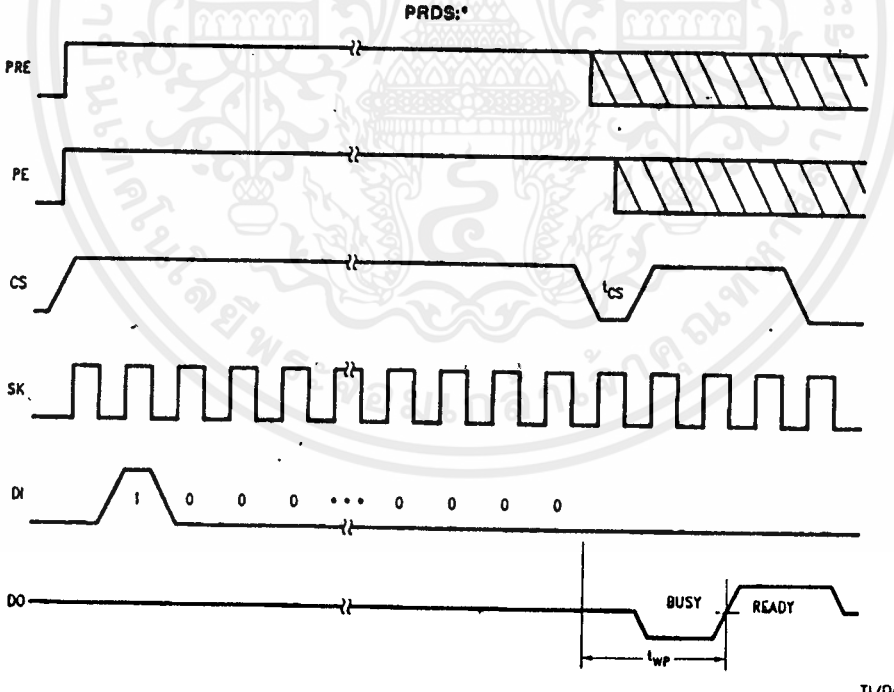
PRWRITE:†



TL/D/9209–13

*Address bit A_7 becomes a "don't care" for NMC93CS56.
†Protect Register MUST be cleared before a PRWRITE cycle. A PREN cycle must immediately precede a PRWRITE cycle.

PRDS:*



TL/D/9209–14

*ONE TIME ONLY instruction. A PREN cycle must immediately precede a PRDS cycle.

การทำงานของ UNIT

NMC93C556 และ NMC93C566 มี 10 INSTRUCTION ดังแสดงในตารางที่

2.3 การอ่านข้อมูล READ :

เหมือนกับ NMC93C56/C66

การเขียน WRITE :

เหมือนกับ NMC93CC56/66 แต่ขา PE จะต้องมี LOGIC"1" ขณะโหลด WRITE INSTRUCTION หลังจากนั้น PE จะเป็น DON'T CARE

การเขียนทั้งหมด WRITE ALL (WRALL) :

คำส่ง WRITE ALL จะใช้ได้ในกรณีที่ "PROTECT REGISTERS" ได้ถูกเคลียร์โดยคำสั่ง PRCLEAR คำสั่ง WRALL จะโปรแกรม REGISTERS ทุก REGISTERS ด้วยรูปแบบข้อมูลที่ระบุบอยู่ในคำสั่ง และเหมือนกับสภาวะ WRITE ขา PE ของ UNIT จะต้องอยู่ในสภาวะ "HIGH" ระหว่าง LOAD WRALL INSTRUCTION

การยกเลิกการเขียน WRITE DISABLE (WDS) :

เหมือนกับ NMC93C56/66

PROTECT REGISTER READ (PRREAD) :

คำสั่ง PRREAD จะเป็นคำสั่งซึ่งอ่านข้อมูลจาก ADDRESS ที่อยู่ในส่วนของ "PROTECT REGISTER" โดยส่งออกที่ขา DO โดย ขา PRE ต้องอยู่ในสภาวะ "HIGH" ขณะLOAD คำสั่งหลังจากนั้น 8 บิท ADDRESS ที่เก็บในหน่วยความจำส่วน PROTECT REGISTER จะส่งออกมายัง SERIAL OUT SHIFT REGISTER

PROTECT REGISTER ENABLE (PREN) :

คำสั่ง PROTECT REGISTER ENNABLE ใช้สำหรับ ENABLE คำสั่ง PRCLEAR, PRWRITE และ PRDS ก่อนจะใช้คำสั่ง PREN จะสั่งได้ตัว UNIT จะต้องอยู่ในสภาวะ WRITE ENABLE(WEN) ก่อนและขา PE และ PRE จะต้องอยู่ในสภาวะ HIGH ขณะ LOAD คำสั่ง และคำสั่ง PREN จะต้องมีขึ้นทันทีทันใด กับคำสั่ง PRCLEAR,PRWRITE หรือ PRDS

PROTECT REGISTER CLEAR (PRCLEAR) :

PRCLEAR จะเคลียร์ ADDRESS ที่เก็บอยู่ใน PROTECT REGISTER เพื่อที่จะ ANABLES REGISTER ทั้งหมดเพื่อที่จะ WRITE หรือ WRITE ALL ขา PRE และ PE ต้องอยู่สภาวะ HIGH ขณะ LOAD คำสั่ง

PROTECT REGISTER WRITE (PRWRITE) :

PRWRITE ใช้สำหรับเขียนข้อมูลลงในส่วนของ REGISTER ที่ PROTECT ไว้ ก่อนที่จะใช้คำสั่ง PRWRITE ได้ต้องใช้คำสั่ง PRCLEAR ก่อนและขา PRE และ PE ต้องอยู่ที่ HIGH

PROTECT REGISTER DISABLE (PRDS) :

เป็นคำสั่งที่สั่งเพียงครั้งเดียว โดยถ้าใช้คำสั่งนี้ REGISTER ที่ถูกระบุจะกลายเป็น ข้อมูลที่เก็บแบบ PERMANENTLY คือแบบถาวร ในขณะ LOAD PROGRAM ขา PE และ PRE ต้องอยู่ในสภาวะ HIGH

รายละเอียดเพิ่มเติมอยู่ในส่วน DATA SHEET

บทที่ 3

การทำงานของ LOGIC CONTROL BOARD

หลักการทำงานจะเป็นวงจร LOGIC ง่ายๆ เนื่องจาก IC MEMORY EEPROM นี้ DATA IN และ OUT จะเป็นแบบ SERIAL ส่วน PERSONEL COMPUTER จะรับและส่งสัญญาณในรูปของ PARALLEL ดังนั้น หน้าที่ที่สำคัญของ LOGIC CONTROL BOARD คือเปลี่ยนข้อมูลส่งแบบ PARALLEL จาก COMPUTER ให้เป็น SERIAL แล้วส่งไปยัง DUT BOARD และในทางกลับกัน LOGIC CONTROL BOARD จะทำหน้าที่เปลี่ยนสัญญาณ OUTPUT จาก DUT BOARD จากแบบ PARALLEL เป็นแบบ SERIAL ก่อนส่งให้ COMPUTER

3.1 การเปลี่ยนสัญญาณ SERIAL/PARALLEL

การเปลี่ยน PARALLEL TO SERIAL

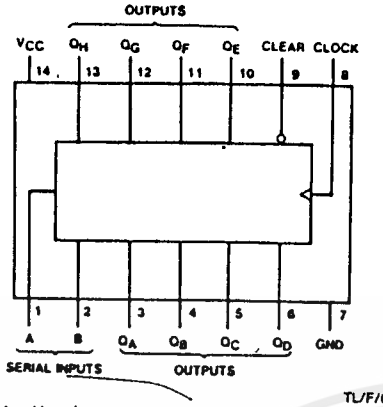
ใช้ IC ของบริษัทเนชั่นแนลเซมิคอนดักเตอร์ เบอร์ DM74LS165 8 BIT PARALLEL IN/SERIAL OUTPUT REGISTER โดย INPUT 8 BIT A-H ที่ขา 11-14 และขา 3-6 ตามลำดับ ส่วน SERIAL OUTPUT จะออกที่ขา 9 อนึ่งเพื่อให้การเลื่อนของข้อมูลสอดคล้องกับตัว UNIT จึงใช้สัญญาณ SK ที่ป้อนให้ตัว UNIT เป็น CLOCK ให้กับ 74LS165 ด้วย แต่ 74LS165ต้องการสัญญาณด้านขอบขาขึ้น ดังนั้นจึงใส่ INVERTER อีก 1 ตัวให้ก่อนป้อนเข้าที่ขา CLOCK คือขา 2 และขา 1 SWIFT/LOAD ก็จะต่อกับสัญญาณ ID LOAD ซึ่งจะสั่งโดย COMPUTER ผ่าน OUTPUT PORT ส่วนขา 15 CLOCK INHIBIT ขา 10 SERIAL OUTPUT จะต่อลง GROUND ตามรูปที่ 3.1 และ SERIAL OUTPUT จะเป็นตามตารางที่ 3.2 และTIMING ตามรูปที่ 3.3

การเปลี่ยน SERIAL TO PARALLEL

ใช้ IC ของบริษัทเนชั่นแนลเซมิคอนดักเตอร์ เบอร์ DM74LS164 8 BIT SERIAL IN/PARALLEL OUT SHIFT REGISTER โดย INPUT SERIAL จะเข้าที่ ขา1 และ 2 ส่วน OUTPUT QA-QH จะออกที่ขา 3-6และ 10-13 ตามลำดับ ในการเลื่อนข้อมูลใช้สัญญาณ CLOCK จาสัญญาณ SK เช่นเดียวกัน โดยก่อนต่อเข้า 74LS164 ต้องต่อ INVERTER ก่อน และต่อเข้าที่ขา 8 ขา 4 ต่อ VCC ตามรูป 3.4 และตารางการทำงานตารางที่ 3.5 และ TIMING DIAGRAM รูปที่ 3.6

Connection Diagram

Dual-In-Line Package



TL/F/6398–1

Order Number 54LS164DMQB, 54LS164FMQB, 54LS164LMQB, DM54LS164J, DM54LS164W, DM74LS164M or DM74LS164N
See NS Package Number E20A, J14A, M14A, N14A or W14B

Function Table

Inputs				Outputs			
Clear	Clock	A	B	Q_A	Q_B	...	Q_H
L	X	X	X	L	L	...	L
H	L	X	X	Q_{A0}	Q_{B0}	...	Q_{H0}
H	↑	H	H	H	Q_{An}	...	Q_{Gn}
H	↑	L	X	L	Q_{An}	...	Q_{Gn}
H	↑	X	L	L	Q_{An}	...	Q_{Gn}

H = High Level (steady state), L = Low Level (steady state)

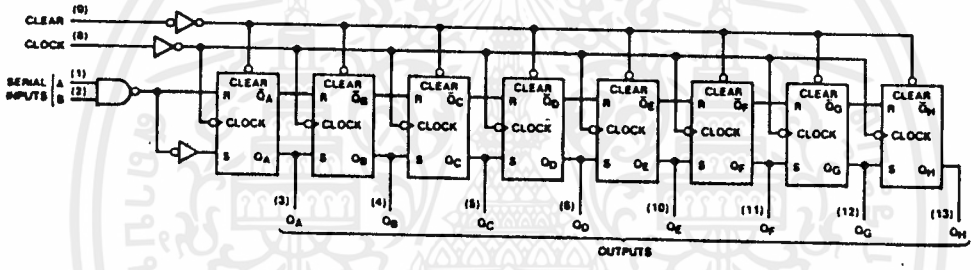
X = Don't Care (any input, including transitions)

↑ = Transition from low to high level

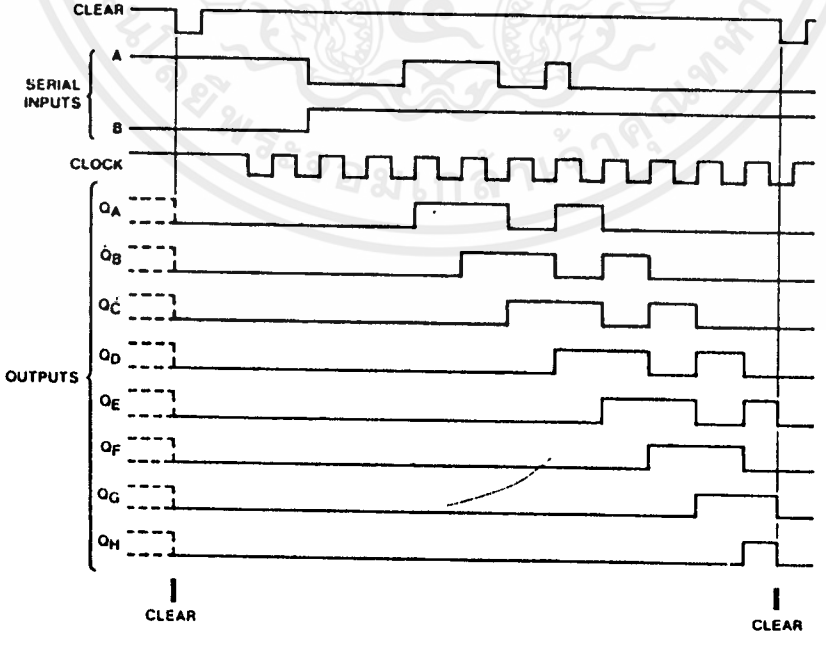
Q_{A0}, Q_{B0}, Q_{H0} = The level of Q_A, Q_B, or Q_H, respectively, before the indicated steady-state input conditions were established.

Q_{An}, Q_{Gn} = The level of Q_A or Q_G before the most recent ↑ transition of the clock; indicates a one-bit shift.

Logic Diagram

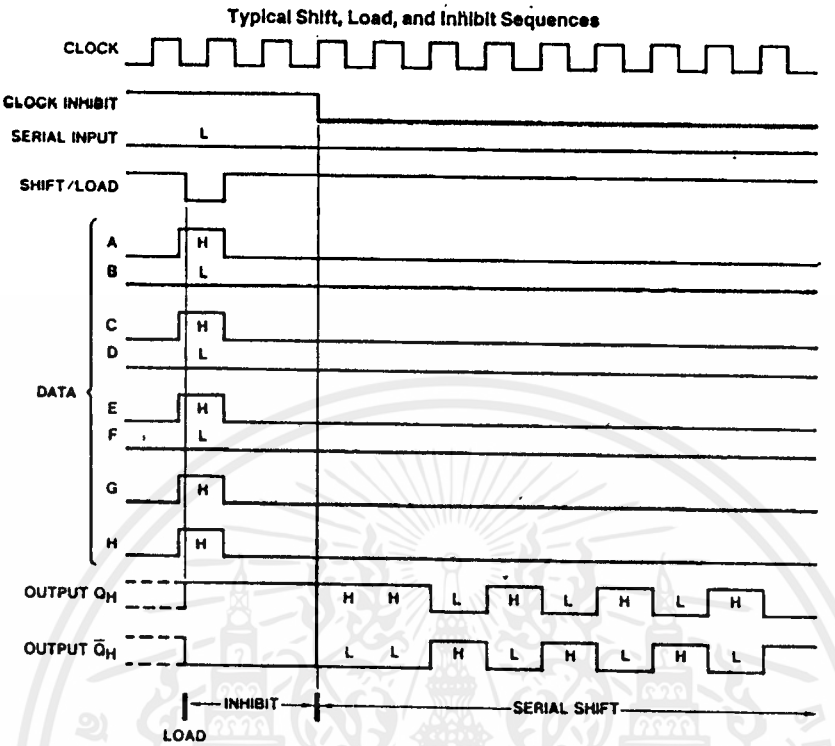


Timing Diagram

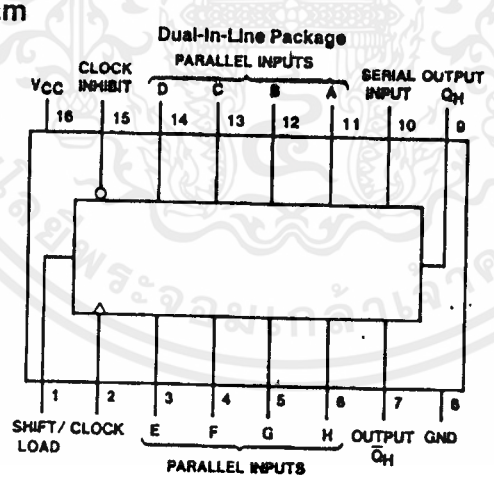


TL/F/631

Timing Diagram



Connection Diagram



Order Number 54LS165DMQB, 54LS165FMQB, DM74LS165WM or DM74LS165N
See NS Package Number J16A, M16B, N16E or W16A

Function Table

Inputs					Internal Outputs		Output
Shift/ Load	Clock Inhibit	Clock	Serial	Parallel A...H	Q_A	Q_B	Q_H
L	X	X	X	a...h	a	b	h
H	L	L	X	X	Q_{A0}	Q_{B0}	Q_{H0}
H	L	↑	H	X	H	Q_{An}	Q_{Gn}
H	L	↑	L	X	L	Q_{An}	Q_{Gn}
H	H	X	X	X	Q_{A0}	Q_{B0}	Q_{H0}

การที่ออกแบบดังนี้เพื่อให้เหมือนกับลักษณะของเลข DECIMAL และสามารถขยายสัญญาณ CS ออกไปได้ถึง 19CS และในกรณีต้องการให้ CS ทำงานทีละ 2CS ใช้การสั่ง OUT PORT ดังนี้

สัญญาณCS	BIT8	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1
CS1,2	L	H	L	H	L	L	L	H
CS3,4	L	H	L	H	L	L	H	L
CS5,6	L	H	L	H	L	L	H	H
CS7,8	L	H	L	H	L	H	L	L
CS9,10	L	H	L	H	L	H	L	H
CS11,12	L	H	L	H	L	H	H	L

และ CS SIGNAL จะใช้วงจร DRIVER โดยใช้ IC DS75361 ซึ่งเป็น DUAL TTL TO MOS DRIVER

รายละเอียดของ IC เบอร์นี้อยู่ในส่วน DATA SHEET

การที่ออกแบบดังนี้เพื่อให้เหมือนกับลักษณะของเลข DECIMAL และสามารถขยายสัญญาณ CS ออกไปได้ถึง 19CS และในกรณีต้องการให้ CS ทำงานทีละ 2CS ใช้การสั่ง OUT PORT ดังนี้

สัญญาณCS	BIT8	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1
CS1,2	L	H	L	H	L	L	L	H
CS3,4	L	H	L	H	L	L	H	L
CS5,6	L	H	L	H	L	L	H	H
CS7,8	L	H	L	H	L	H	L	L
CS9,10	L	H	L	H	L	H	L	H
CS11,12	L	H	L	H	L	H	H	L

และ CS SIGNAL จะใช้วงจร DRIVER โดยใช้ IC DS75361 ซึ่งเป็น DUAL TTL TO MOS DRIVER

รายละเอียดของ IC เบอร์นี้อยู่ในส่วน DATA SHEET

3.3 DI SIGNAL

DI SERIAL DATA IN จะถูกสร้างขึ้นโดย SOFTWARE ที่ RUN ใน COMPUTER โดยใช้คำสั่งให้เปลี่ยน STATE ในช่วงเวลาที่ต้องการใน LOGIC CONTROL BOARD สามารถเลือกสัญญาณ DI ได้ 12 สัญญาณ โดยระบุเป็น DI1-DI12 โดยคำสั่งของ COMPUTER ผ่าน PORT จะถูก LATCH โดย IC เบอร์ DM74LS374 โดยสัญญาณที่ส่งออกมาเป็น PARAUEL 8 BIT โดย 4 BIT บนจะเป็นคำสั่งเพื่อเลือกให้ DECODER ตัวใดทำงาน DECODER ที่ใช้ DM74LS154 3 ตัว โดยตัวที่ 1 จะทำระหว่าง DI1-DI9 และตัวที่ 3 ใช้ในกรณีที่ตัวทำเป็นกลุ่มๆ DI ทีละ 3 DI เลย

สัญญาณเลือก DI1-DI12

สัญญาณDI	BIT8	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1
DI1	L	L	L	L	L	L	L	H
DI2	L	L	L	L	L	L	H	L
DI3	L	L	L	L	L	L	H	H
DI4	L	L	L	L	L	H	L	L
DI5	L	L	L	L	L	H	L	H
DI6	L	L	L	L	L	H	H	L
DI7	L	L	L	L	L	H	H	H
DI8	L	L	L	L	H	L	L	L
DI9	L	L	L	L	H	L	L	H
DI10	L	L	L	H	L	L	L	H
DI11	L	L	L	H	L	L	H	L
DI12	L	L	L	H	L	L	H	H

และในกรณีต้องการ DI ทำงานทีละ 3DI เลย ใช้คำสั่ง OUTPORT ดังนี้

สัญญาณDI	BIT8	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1
DI1-3	L	H	L	H	L	L	L	H
DI4-6	L	H	L	H	L	L	H	L
DI7-9	L	H	L	H	L	L	H	H
DI10-12	L	H	L	H	L	H	L	L

และ DI SIGNAL จะใช้วงจร DRIVER โดยใช้ IC 74LS366A HEX TRI-STATE INVERTING BUFFERS

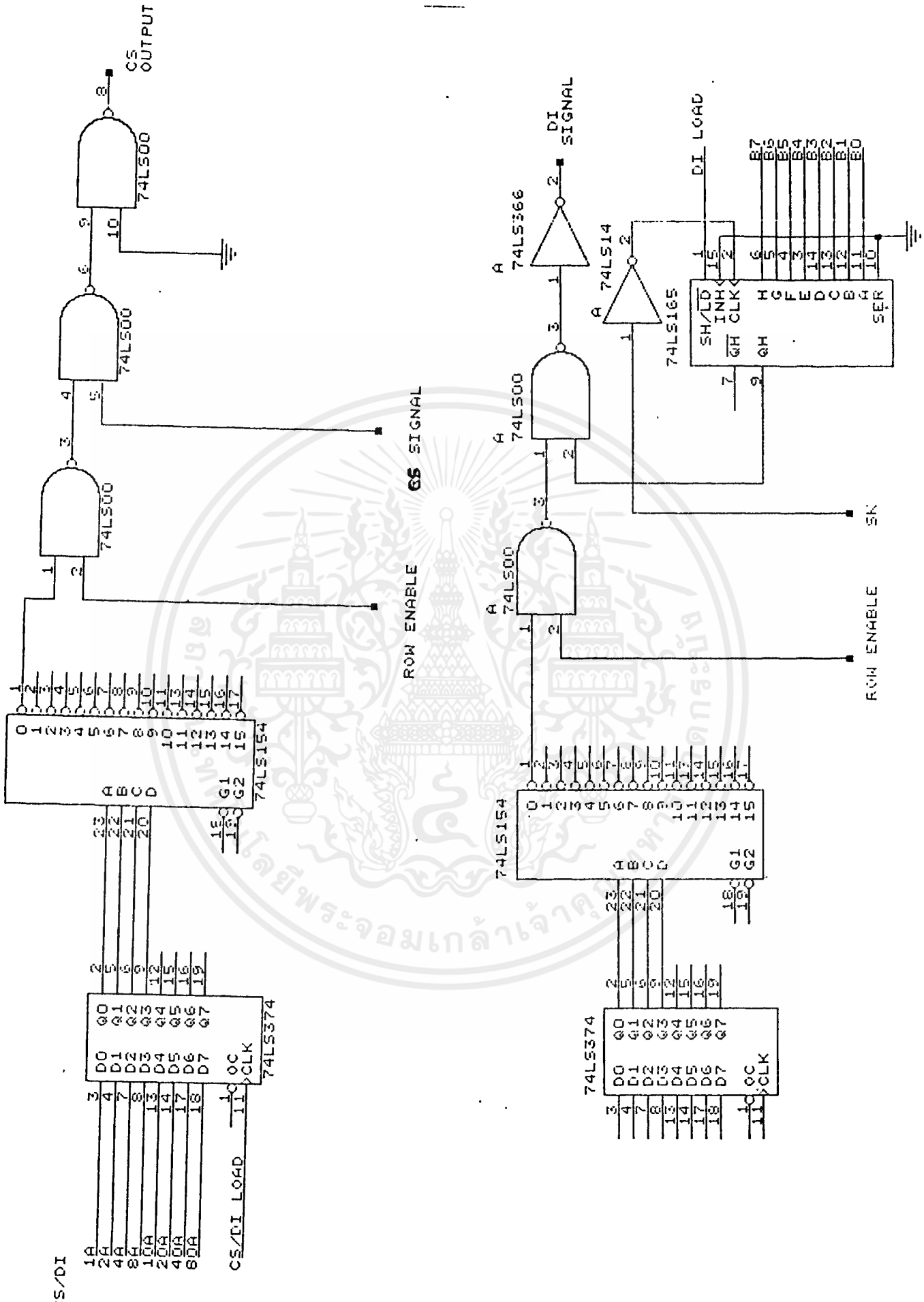
รายละเอียดของ IC เบอร์นี้อยู่ในส่วน DATA SHEET

3.4 DO SIGNAL

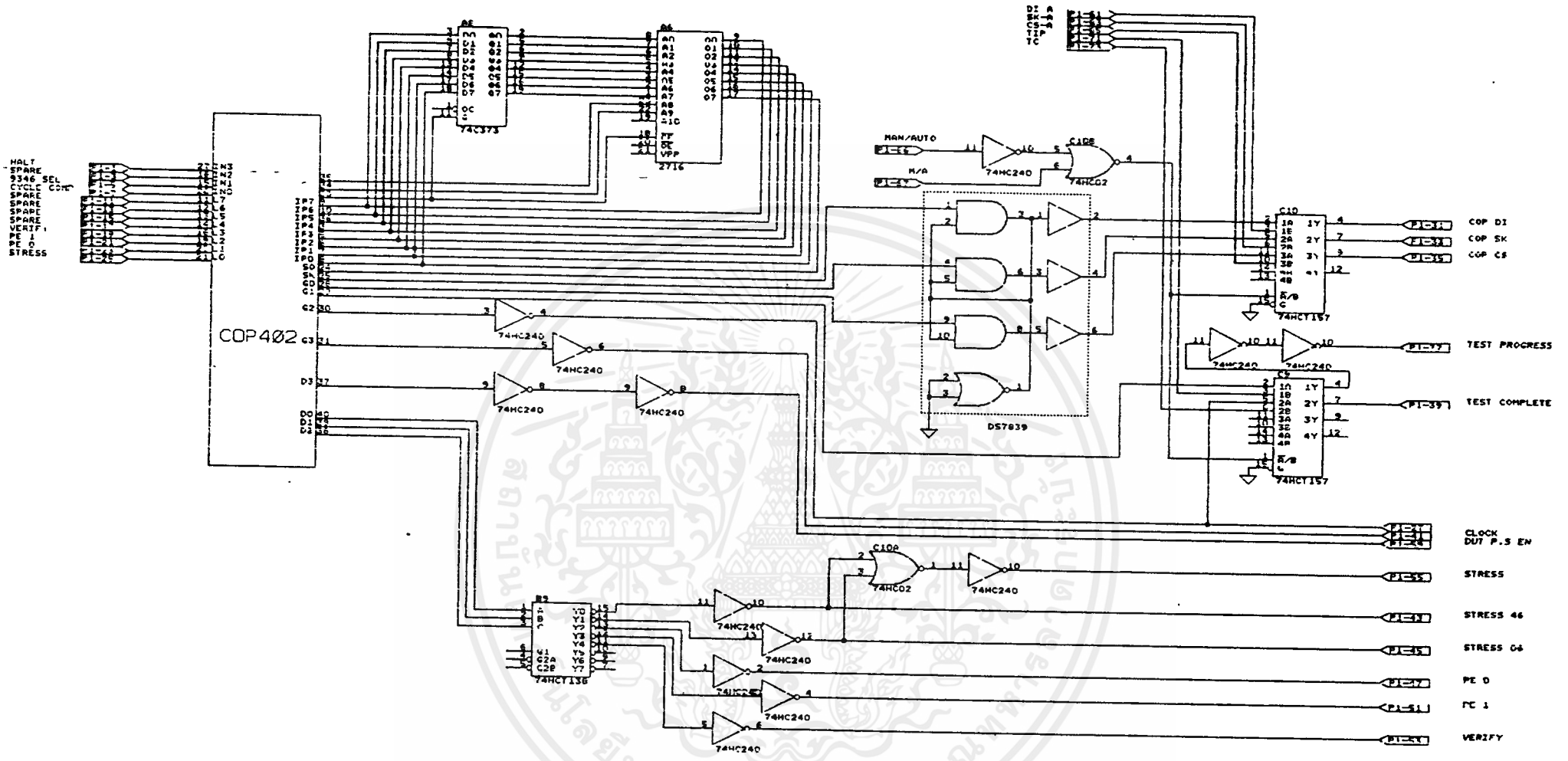
เป็นสัญญาณที่มาจาก DUT BOARD โดยมาในลักษณะของ SERIAL และจะมีหลายสัญญาณจำเป็นต้องใช้คำสั่งจาก COMPUTER เป็นตัวเลือกว่าจะเลือกสัญญาณ DO ชุดไหน่ไปยัง COMPUTER โดยใช้ IC เบอร์ 74LS151 DATA SELECTOR/MULTIPLEXER โดยคำสั่งของ COMPUTER จะออกมายัง PORT และเข้าที่ IC นี้ที่ขา DATA SELECT A-C ขา 11-9 ซึ่ง IC DM74LS151 ก็จะทำการเลือก DO ที่ต้องการเพียง DO เดียว จากชุดทดลองนี้ออกแบบให้มี DO ทั้งหมด 4 ชุดคือ DO1-DO4โดยก่อนเข้า DM74LS151 จะมี BUFFER DM74LS244 ต่ออยู่ หลังจากเลือกสัญญาณ DO ทีต้องการได้แล้วก็จะส่ง DO ไปยัง 74LS164 เพื่อเปลี่ยนสัญญาณ SERIAL เป็น PARALLEL ต่อไป

สัญญาณ OUTPUT PORT เพื่อสั่งเลือก DO1-DO4

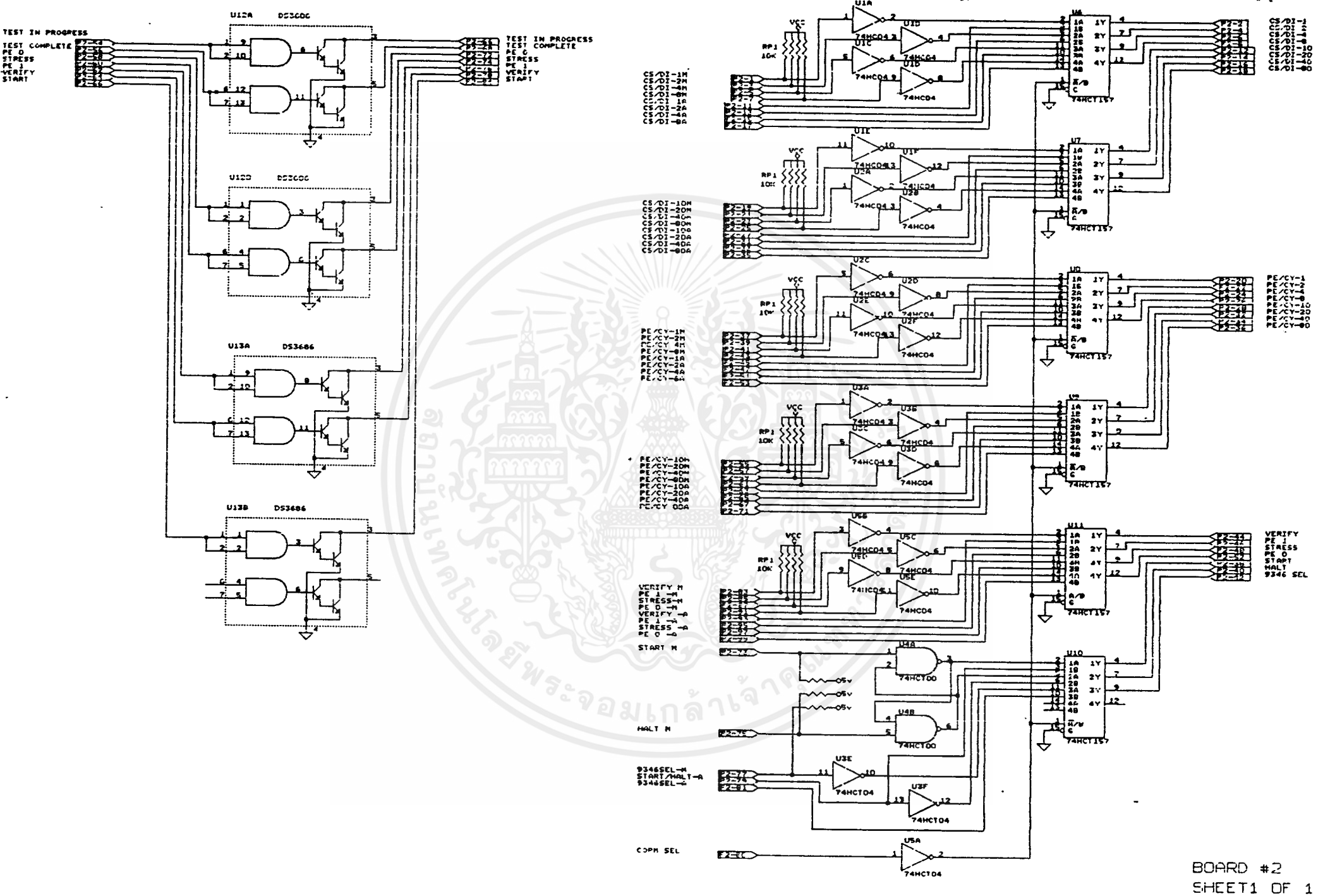
สัญญาณDO	C	B	A
DO1	L	L	L
DO2	L	L	H
DO3	L	H	L
DO4	L	H	H



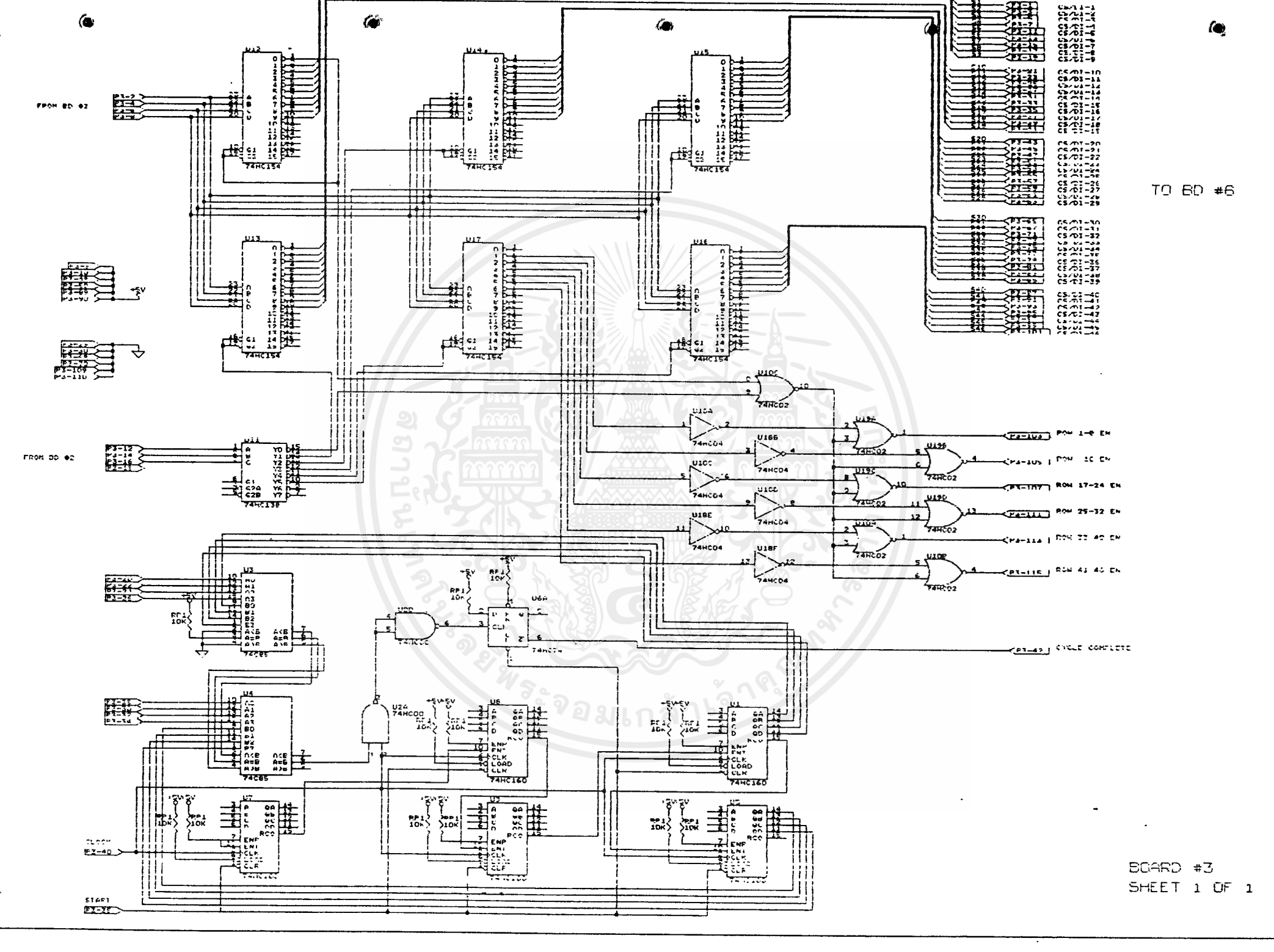
3.5 SCHEMATIC DIAGRAM

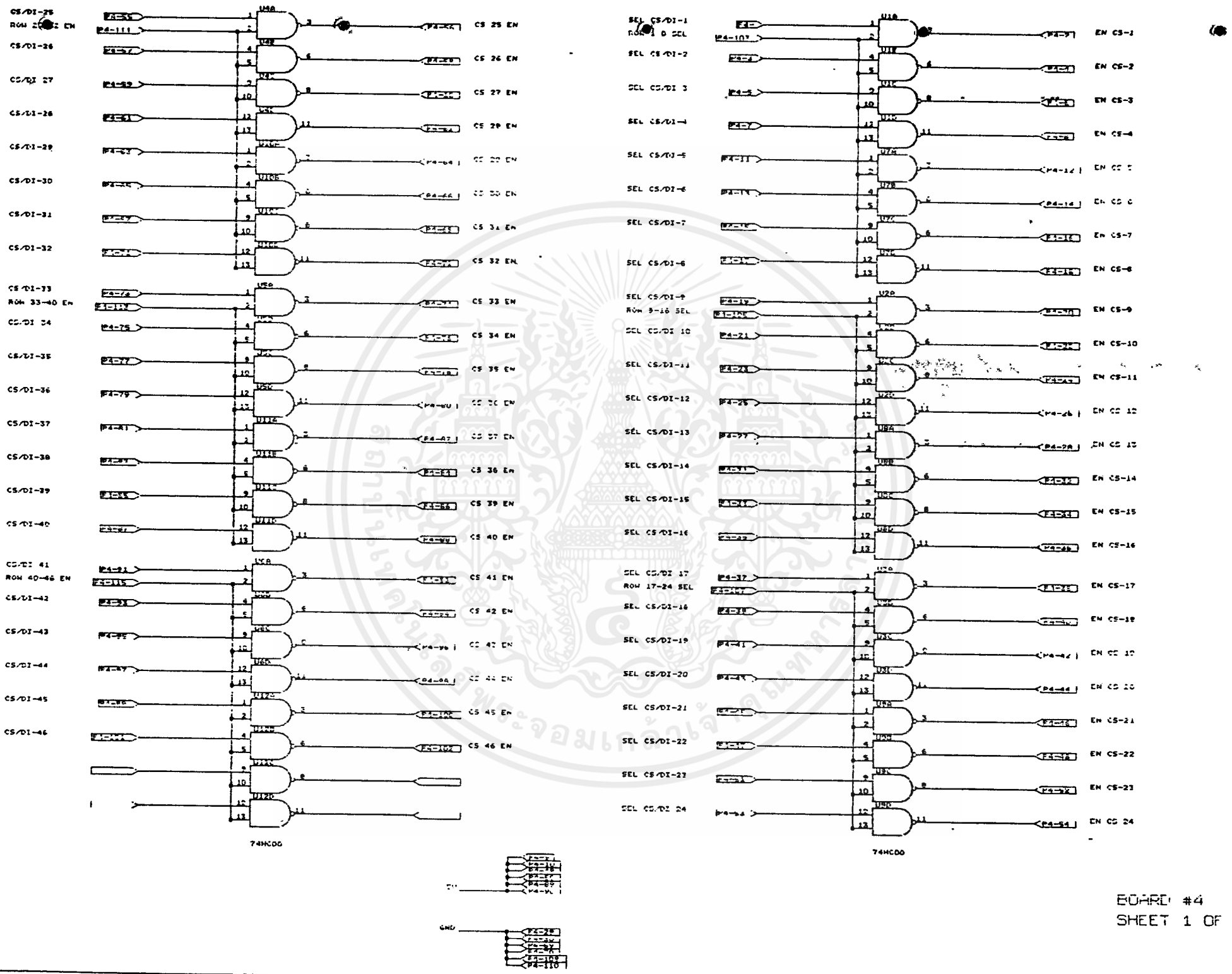


BOARD #1
SHEET 1 OF 1

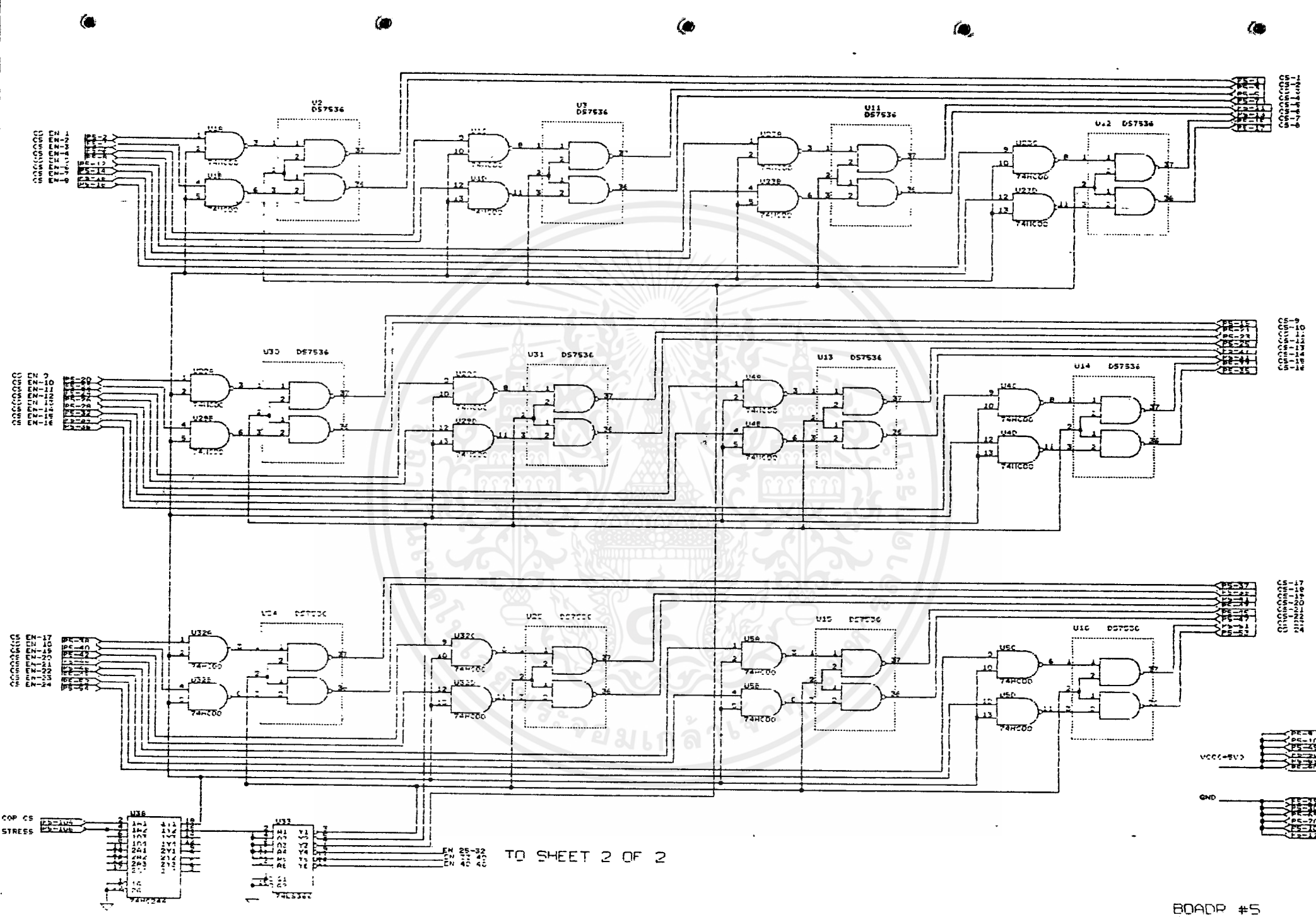


BOARD #2
SHEET1 OF 1

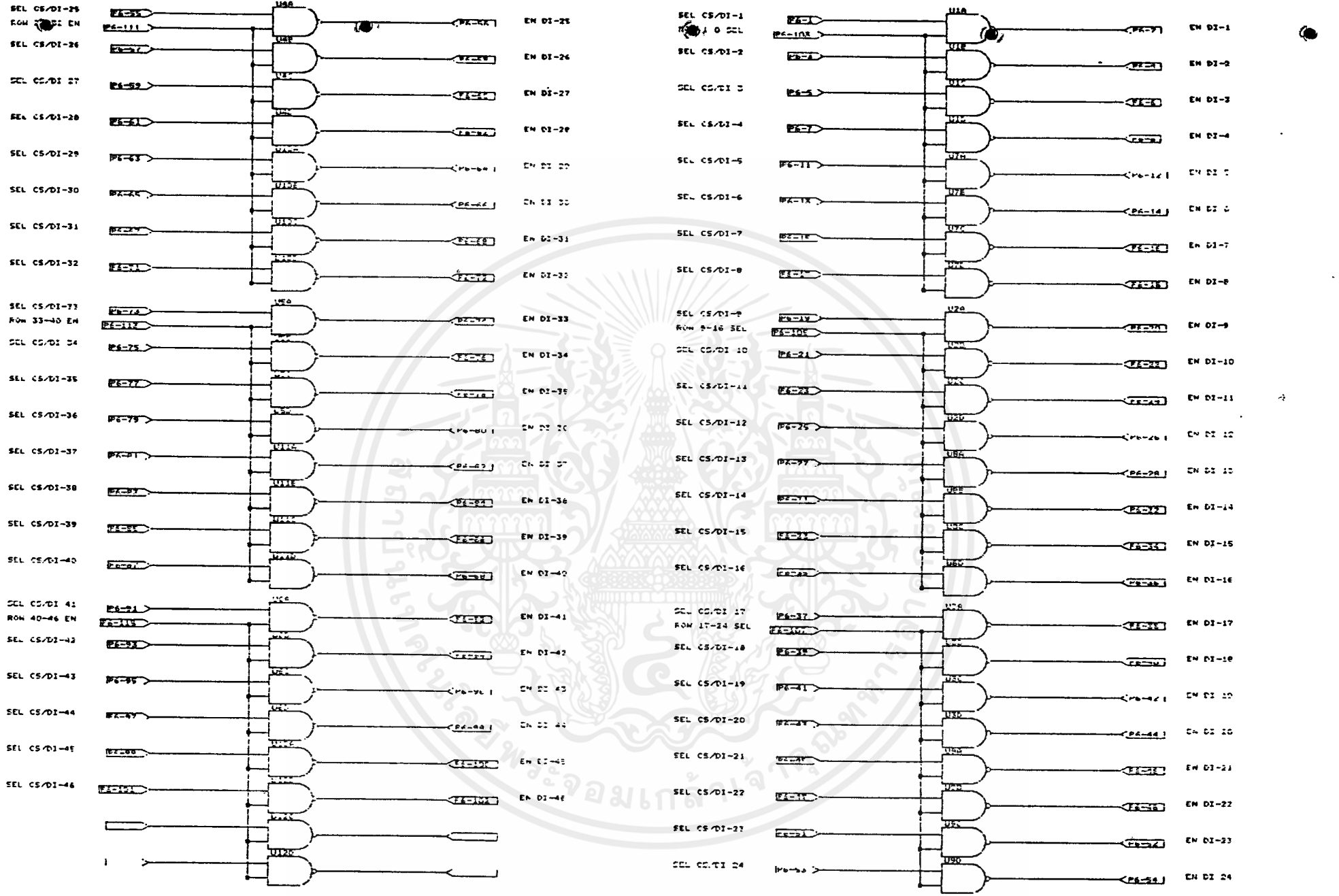

FROM BD #2
TO BD #6
74HC154
74HC138
74HC02
74HC04
74HC00
74HC160
74HC74
74C85
74C86
ROW 1-8 EN
ROW 9-16 EN
ROW 17-24 EN
ROW 25-32 EN
ROW 33-40 EN
ROW 41-48 EN
CYCLE COMPLETE
CLOCK
START
BOARD #3
SHEET 1 OF 1



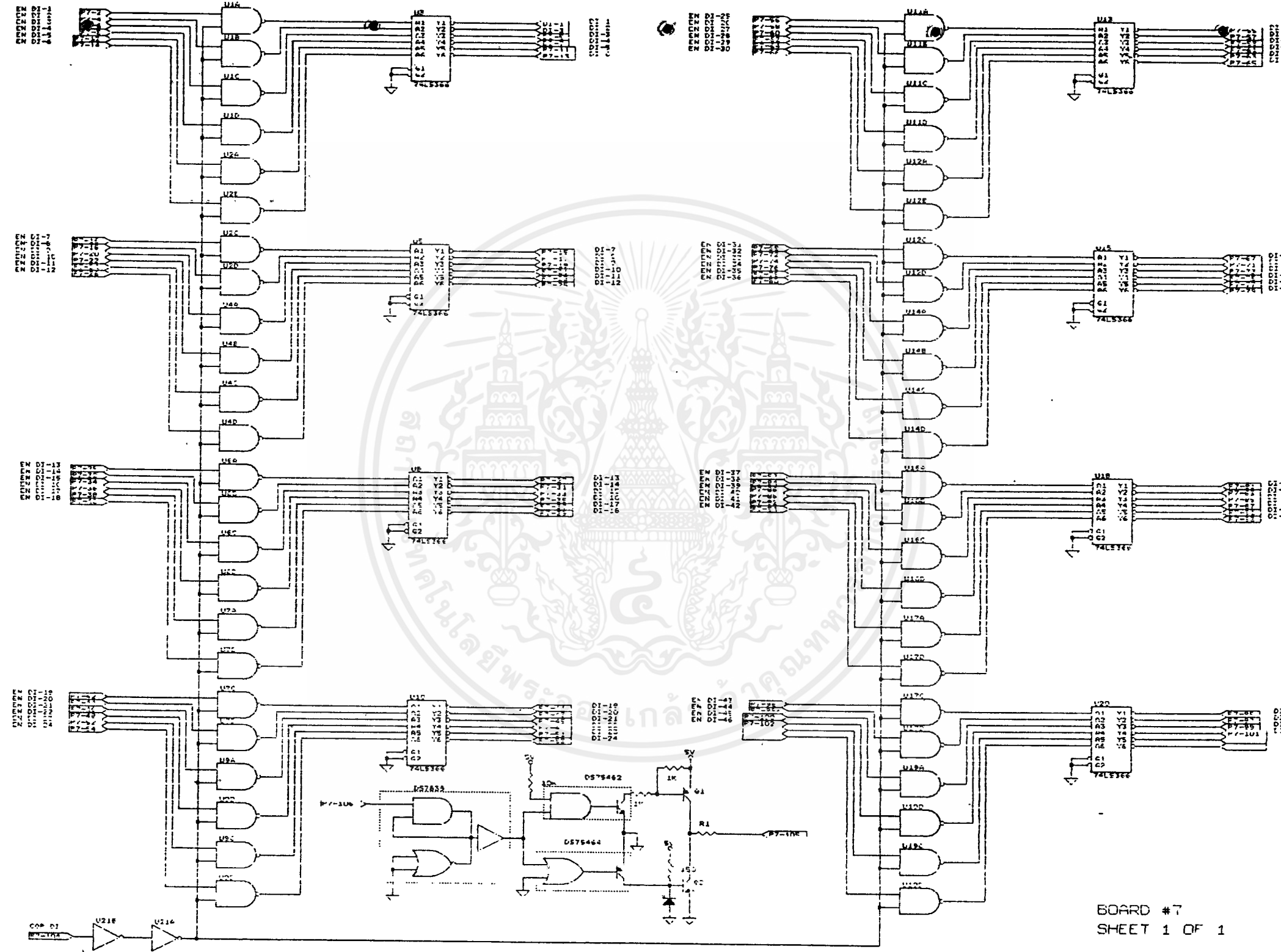
BOARD #4
SHEET 1 OF 1

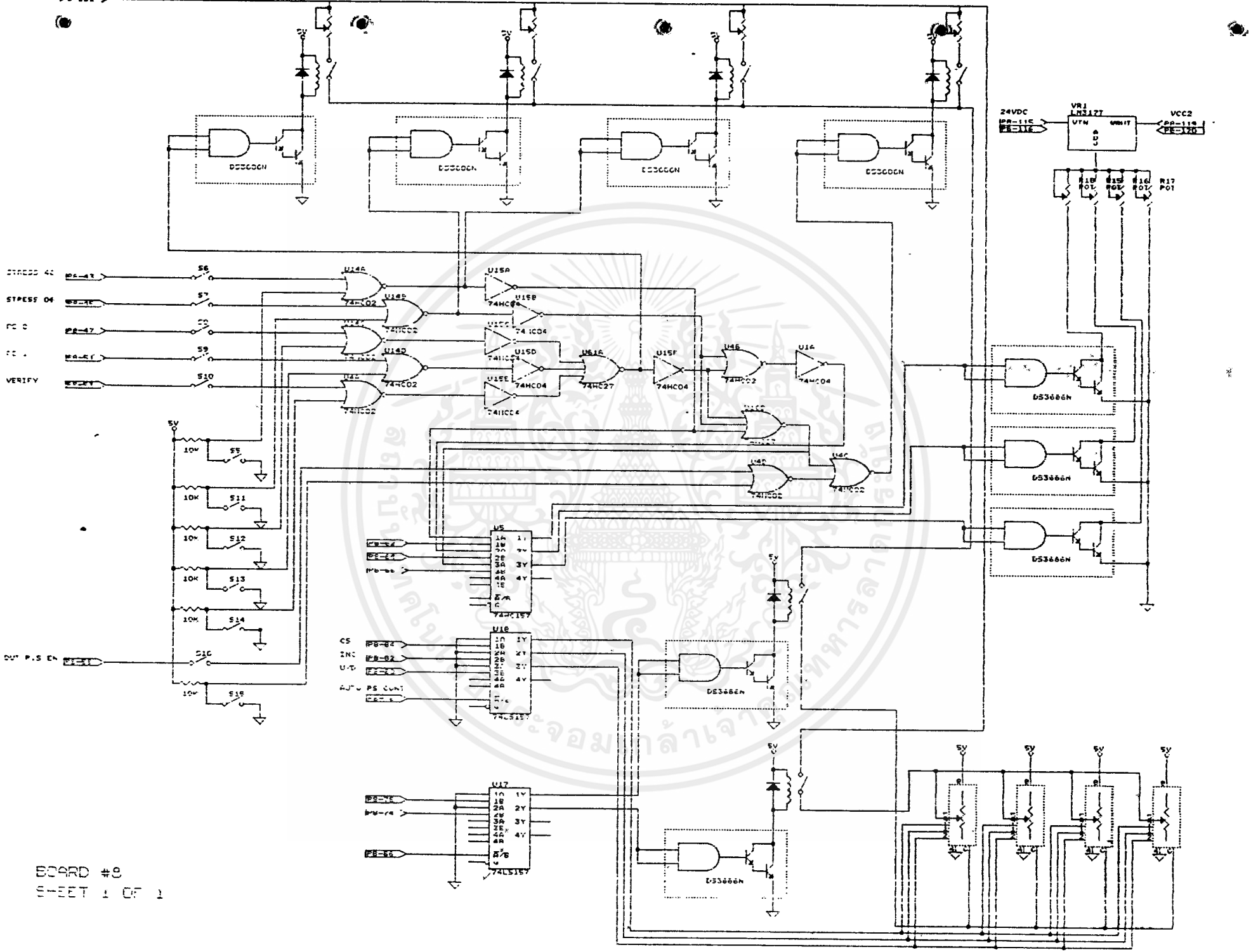


BOADR #5
SHEET 1 OF 2

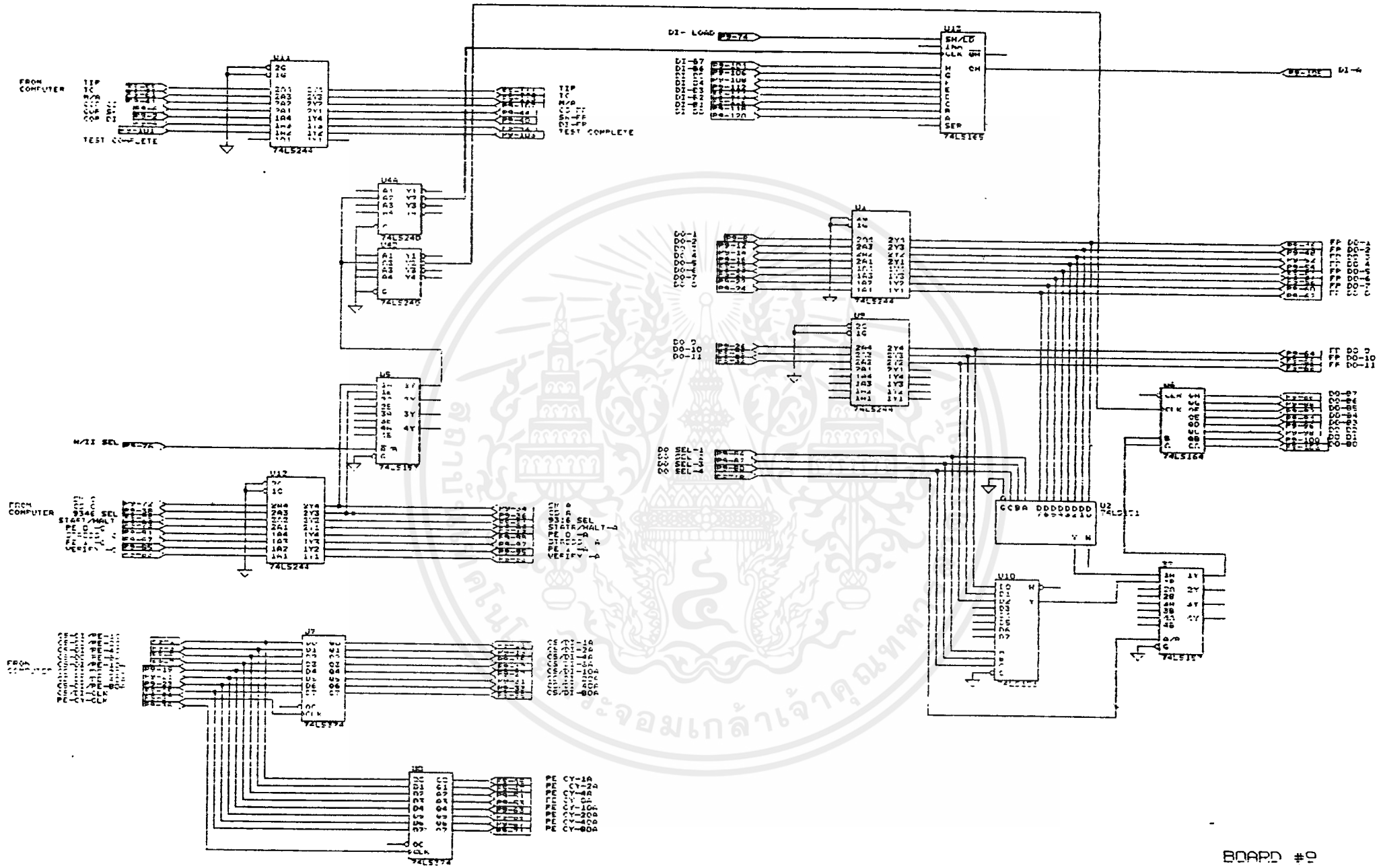


BOARD #6
SHEET 1 OF 1


DS7833
DS7464
DS7462
1K
R1
5V
74LS366
74LS367
74LS368
BOARD #7
SHEET 1 OF 1

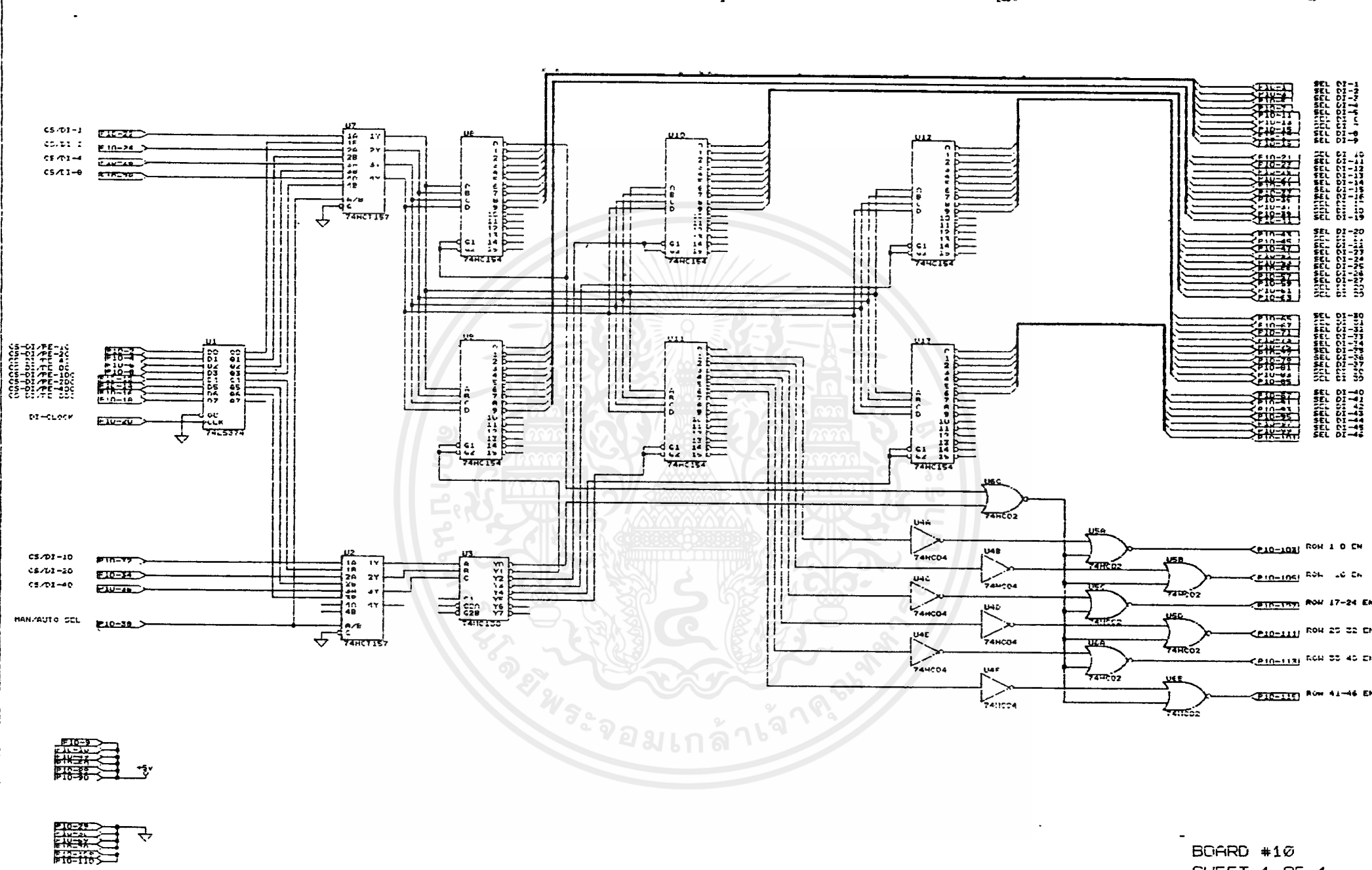


BOARD #8
SHEET 1 OF 1



BOARD #9

SHEET 1 OF 1


CS/DI-1
CS/DI-2
CS/DI-4
CS/DI-8
DI-CLOCK
CS/DI-10
CS/DI-20
CS/DI-40
MAN/AUTO SEL
74HCT157
74LS374
74HC154
74HC04
74HC02
+5V
BOARD #10
SHEET 1 OF 1

บทที่ 4

การทำงานของ INTERFACE BOARD

INTERFACE BOARD เป็น BOARD ที่ทำหน้าที่รับส่งสัญญาณระหว่าง LOGIC CONTROL BOARD กับ COMPUTER โดย INTERFACE BOARD ประกอบด้วย IC 8255A PROGRAMMABLE PERIPHERAL INTERFACE จำนวน 3 ตัวและ BUFFER DM74LS244 จำนวน 8 ตัวและ DM74LS138 จำนวน 1 ตัว

4.1 หลักการทำงาน

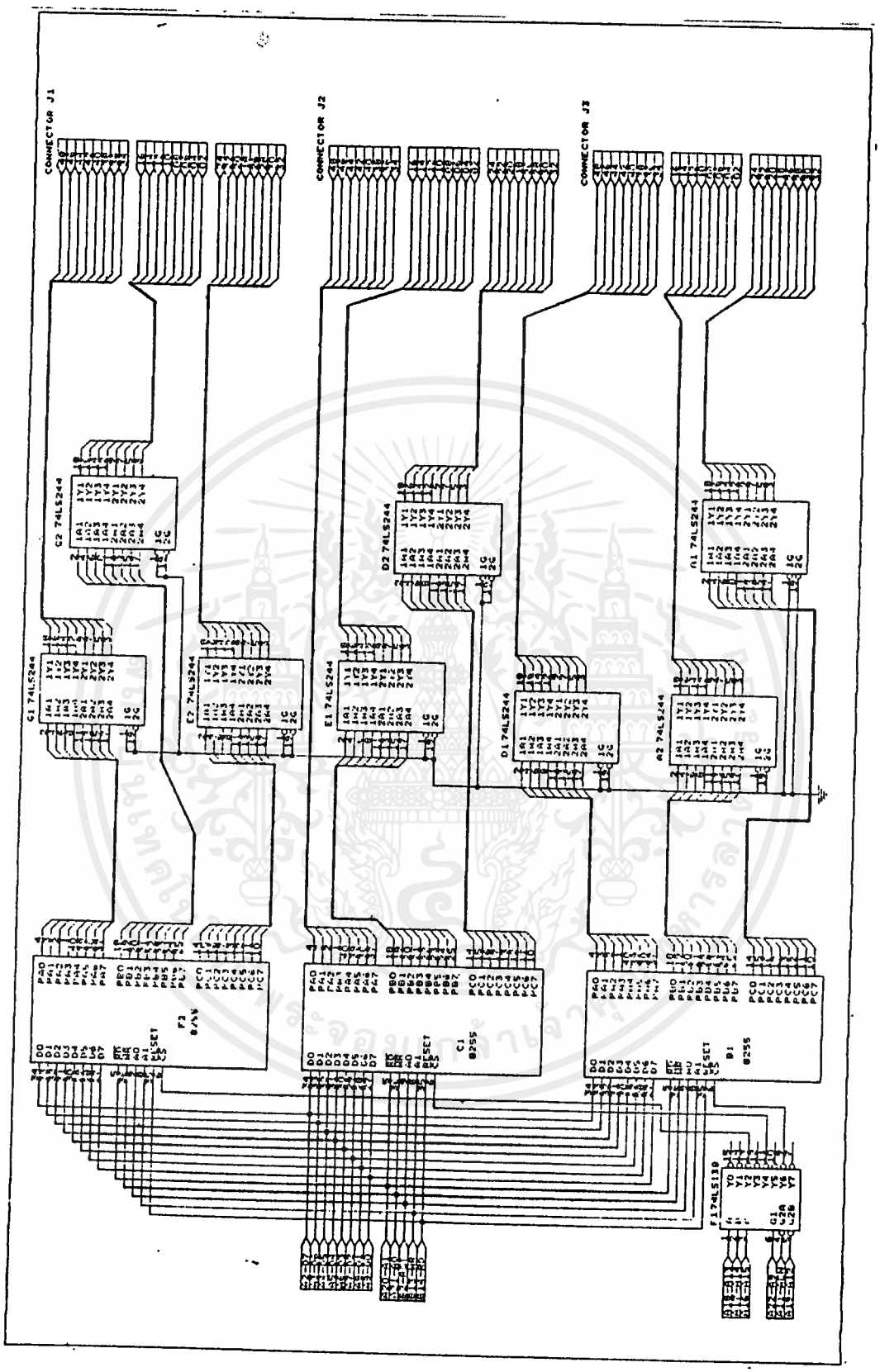
การสั่งงานจะสั่งงานของ COMPUTER โดยสั่งที่ขา D0-D7 และPROGRAMMING PORT ทั้ง 3 ตัว จะต่อขนานกันและที่ OUTPUT PORT แต่ละบิทจะต่อกับ BUFFER DM74LS244 ส่วน INPUT PORT จะต่อโดยตรงกับ LOGIC CONTROL BOARD

รายละเอียดของ 8255 มีอยู่ในส่วน DATA SHEET

การเลือกสั่งงาน PORT จะใช้บิท 13-บิท15 ต่อกับ DECODER DM74LS138 และ OUTPUTของ DM74LS138 จะต่อเป็นสัญญาณ ENABLE PORT แต่ละตัว LOGICที่ใช้ควบคุมการทำงานของ PORT แต่ละตัวดังแสดงในตารางที่ 4.1

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	PORT				
0	1	0	0	0	0	1	1	0	1	1	1	0	0	0	0	4	3	7	0	PORT 1
0	1	0	0	0	1	1	1	0	1	1	1	0	0	0	0	4	7	7	0	
0	1	0	0	1	0	1	1	0	1	1	1	0	0	0	0	4	B	7	0	
0	1	0	0	1	1	1	1	0	1	1	1	0	0	0	0	4	F	7	0	
1	0	0	0	0	0	1	1	0	1	1	1	0	0	0	0	8	3	7	0	PORT 2
1	0	0	0	0	1	1	1	0	1	1	1	0	0	0	0	8	7	7	0	
1	0	0	0	1	0	1	1	0	1	1	1	0	0	0	0	8	B	7	0	
1	0	0	0	1	1	1	1	0	1	1	1	0	0	0	0	8	F	7	0	
1	1	0	0	0	0	1	1	0	1	1	1	0	0	0	0	C	3	7	0	PORT 3
1	1	0	0	0	1	1	1	0	1	1	1	0	0	0	0	C	7	7	0	
1	1	0	0	1	0	1	1	0	1	1	1	0	0	0	0	C	B	7	0	
1	1	0	0	1	1	1	1	0	1	1	1	0	0	0	0	C	F	7	0	

4.2 SCHEMATIC DIAGRAM

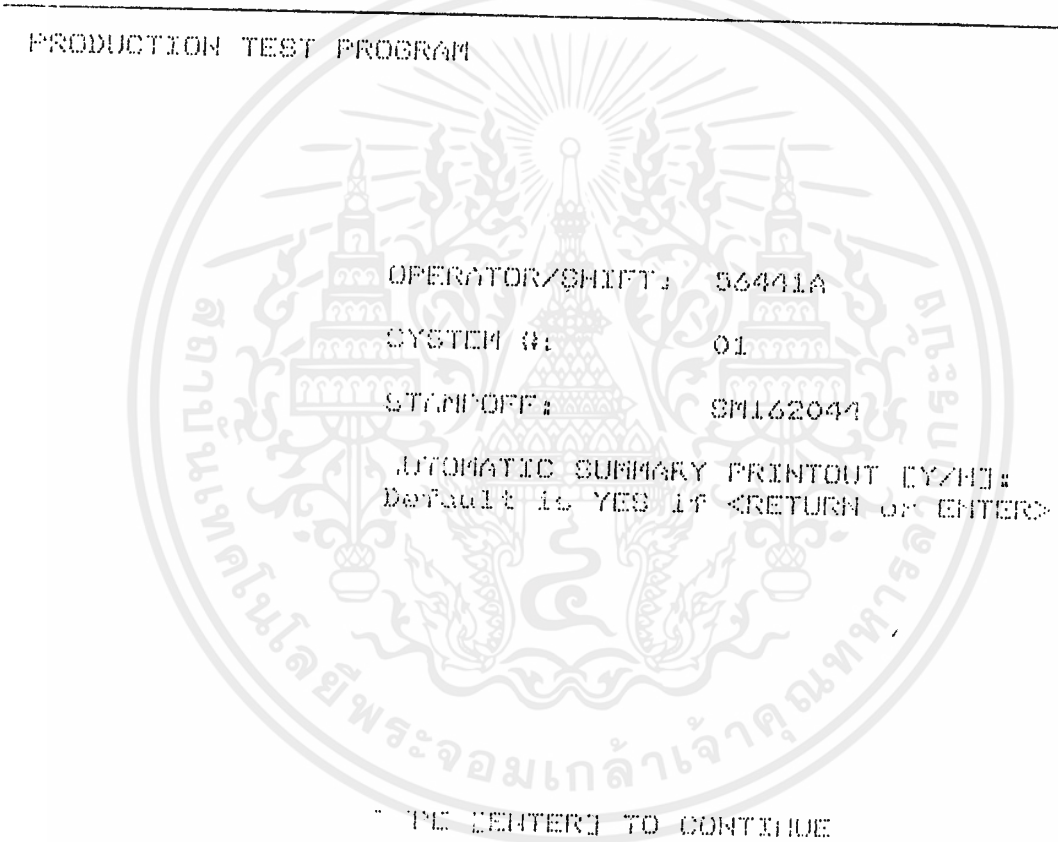


บทที่ 5

การใช้งาน

เพื่อความสะดวกจึงมีบทนี้ขึ้นเพื่อแสดงถึงหน้าจอของ COMPUTER ในกรณีที่เราจะใช้งานโดยจะอธิบายโดยใช้หน้าจอของ COMPUTER เป็นตัวแสดง

เมื่อเปิดเครื่องและRUN PROGRAM หน้าจอจะแสดง



ขั้นแรก เครื่องจะให้ป้อนข้อมูลต่างๆ

บรรทัดแรก ใส่ข้อมูล เลขประจำตัวของพนักงานที่ใช้เครื่อง แล้ว ENTER

บรรทัดที่สอง ใส่ข้อมูล หมายเลขเครื่อง แล้ว ENTER

บรรทัดที่สาม ใส่ข้อมูล หมายเลข STAMPOFF ถ้ามี ถ้าไม่มี ENTER ได้เลย

บรรทัดที่สี่ ใส่ข้อมูล ว่าเราต้องการพิมพ์ผลการ TEST หลังจาก TEST เสร็จแล้วหรือไม่โดยตอบ Y คือ YES ใช่ และตอบ N คือ NO

ไม่ แล้ว ENTER

ขั้นที่สอง เครื่องจะให้เราเลือกประเภทของ ไอซีที่เราทำการ TEST

```
PRODUCTION TEST PROGRAM

                    TYPE  [0] ..... NMOS

                    TYPE  [1] ..... CMOS


                    ENTER SELECTION HERE:
```

เลือก 0 เมื่อต้องการ TEST NMOS และเลือก 1 เมื่อต้องการ TEST CMOS

ขั้นที่ 3 จะเลือกเบอร์ DEVICE ที่ต้องการ TEST

```
PRODUCTION TEST PROGRAM

TYPE  [A] ..... NM  93C06        TYPE  [N] ..... NM  88C906
TYPE  [B] ..... NMC 93C06        TYPE  [P] ..... NM  88C946
TYPE  [C] ..... NM  93C006

                                 TYPE  [Q] ..... NM  93C46A
TYPE  [D] ..... NM  93C07        TYPE  [R] ..... NM  59C11

TYPE  [E] ..... NM  93C46        TYPE  [S] ..... NM  88C16
TYPE  [F] ..... NMC 93C46        TYPE  [T] ..... NM  59C16
TYPE  [G] ..... NM  93C946

                                 TYPE  [U] ..... NM  95C12
TYPE  [H] ..... NM  93C56
TYPE  [I] ..... NMC 93C56        TYPE  [V] ..... NM  93C13H
TYPE  [J] ..... NM  93C956
                                 TYPE  [W] ..... NM  93C14H
TYPE  [K] ..... NM  93C66
TYPE  [L] ..... NMC 93C66
TYPE  [M] ..... NM  93C966

                    ENTER SELECTION HERE:
```

การเลือกให้เลือกตัวอักษรที่อยู่หน้ารายการที่ต้องการเลือก

ขั้นที่ 4 เลือก PROGRAM ที่ใช้ TEST

```
PRODUCTION TEST PROGRAM

        TYPE  [0] ..... T93C46.1KT
        TYPE  [1] ..... T93C46.5KT
        TYPE  [2] ..... T93C46.1K
        TYPE  [3] ..... T93C46.10K
        TYPE  [4] ..... T93C46.50K
        TYPE  [5] ..... T93C46.VER
        TYPE  [E] ..... TO ENTER PROGRAM NAME

        ENTER SELECTION HERE:
```

เลือกโดยพิมพ์หมายเลขหน้าโปรแกรม

ความหมายของแต่ละโปรแกรม คือ

T93C46.1KT หมายถึง TEST และทดสอบเขียนและลบ PROGRAM 1,000 ครั้ง

T93C46.5KT หมายถึง TEST และทดสอบเขียนและลบ PROGRAM 5,000 ครั้ง

T93C46.1K หมายถึง ทดสอบเขียนและลบ PROGRAM 1,000 ครั้ง แต่ไม่ TEST

T93C46.10K หมายถึง ทดสอบเขียนและลบ PROGRAM 10,000 ครั้ง แต่ไม่ TEST

T93C46.50K หมายถึง ทดสอบเขียนและลบ PROGRAM 50.000 ครั้ง แต่ไม่ TEST

T93C46.VER หมายถึง TEST อย่างเดียวไม่มีการทดสอบเขียนและลบ PROGRAM

ส่วน [E] เราสามารถเรียก PROGRAM ที่เราเขียนเองมาใช้ได้

ขั้นที่ 5 เลือกชนิดของ DUT LOAD BOARD ที่เราต้องการ

```
PRODUCTION TEST PROGRAM

                 SELECT BOARD TYPE

          BOARD SELECTION                          ENTER
STD       DIP  (N/J)   REV C                       [0]
STD       DIP  (N/J)   REV B                       [1]
STD       SOIC (M/M8)  REV A                       [2]
STD       SOIC (M8)    REV B                       [3]
STD       SOIC (M14)   REV A                       [4]
NM95C12   DIP  (N/J)   REV A                       [5]
NM95C12   SOIC (M)     REV A                       [6]
STD       SOIC (WM14)  REV A                       [7]
```

ชนิดของ LOAD BOARD จะขึ้นอยู่กับ PACKAGE ของยูนิตที่นำมา TEST ซึ่งประกอบด้วย

DIP(N/J) คือใช้กับ IC DIP PACKAGE

SOIC M/M8 คือใช้กับ IC SO PACKAGE 8 LEADS

SOIC M14 คือใช้กับ IC SO PACKAGE 14 LEADS

SOIC WM14 คือใช้กับ IC SO ขนาดกว้าง 14 LEADS

ขั้นที่ 6 หน้าจอจะแสดงสิ่งที่เราเลือกไปทั้งหมด

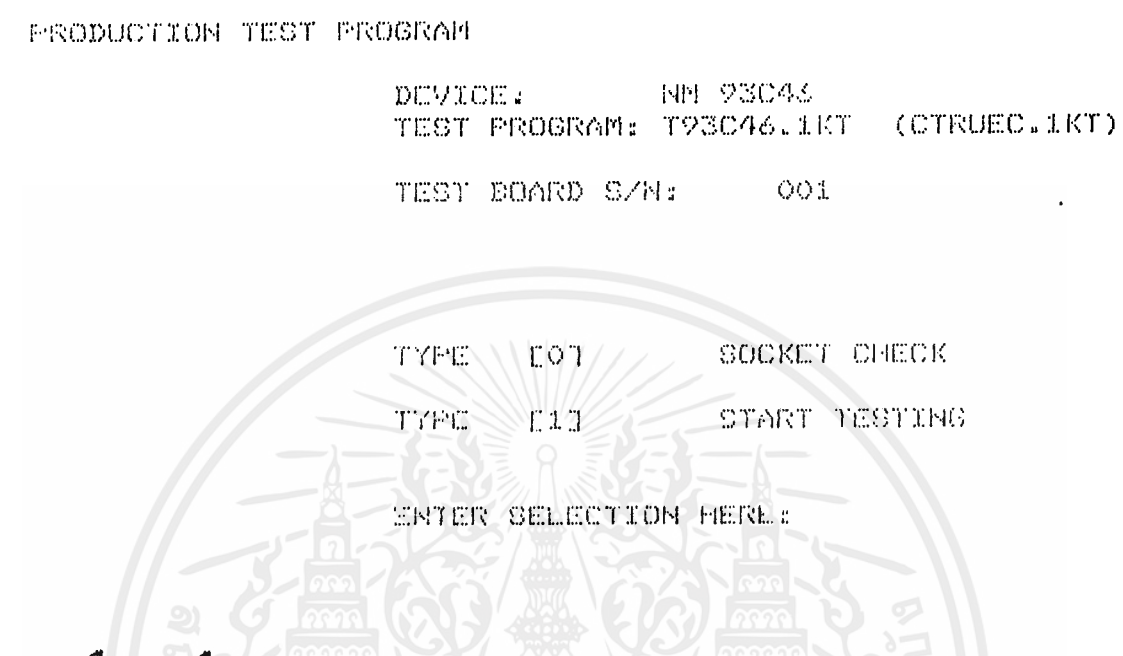
```
PRODUCTION TEST PROGRAM

DEVICE TYPE:          NM 93C46
TEST PROGRAM:         T93C46.1KT
LOAD BOARD:           STD        DIP   (N/J)    REV C
OPERATOR/SHIFT:       56441A
SYSTEM #              01
STAMPOFF:             SM162044
AUTOSUMMARY:          OFF

TYPE [X] TO CONTINUE ... OR PRESS [ENTER] TO START OVER
```

ถ้าสั่งที่จอภาพแสดงออกมาถูกต้องตามที่ต้องการให้กด X หรือถ้าการเลือกไม่ถูก เรากด ENTER เครื่องจะไปตั้งต้นขั้นที่ 1 ใหม่ เพื่อให้เราแก้ไขสิ่งที่ไม่ถูกต้อง
ขั้นที่ 7 เครื่องจะให้เราป้อนหมายเลขของ DUT LOAD BOARD

```
PRODUCTION TEST PROGRAM

        DEVICE:        NM 93C46
        TEST PROGRAM: T93C46.VER  (CTRUEC.VER)

        TEST BOARD S/N:
```

ให้ใส่หมายเลขของ DUT LOAD BOARD ในที่นี้มีเพียง BOARD เดียวให้ป้อน 001 แล้ว ENTER

ขั้นที่ 8 เครื่องจะให้เราเลือก TEST โดยแสดงรายละเอียดที่ป้อนไว้ด้วย

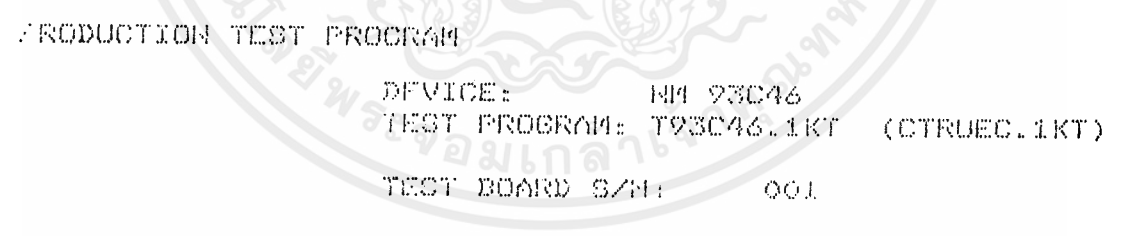


เลือก 0 หรือ 1

โดย 0 หมายถึงตรวจสอบว่าเราใส่ UNITS ลงใน SOCKET ถูกต้องดีหรือไม่

1 เพื่อ START TESTING

ใส่ 1 แล้ว ENTER ด้านล่างจอภาพจะแสดง



ให้กดตัว "x"

หลังจากกด "X"แล้ว เครื่องจะทำการ TEST และทำการพิมพ์ผลของการ TEST ดังแสดงในหน้า 49

หมายเหตุ

ถ้า UNIT ดีจะแสดงเป็นจุด (.)

ถ้า UNIT เสียจะแสดงเป็นเลข 5

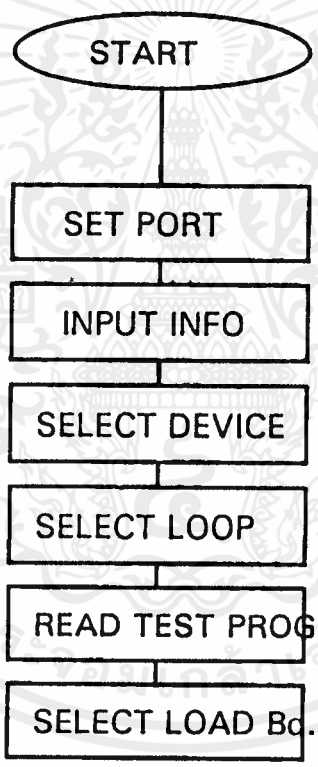
```
EXERCISOR TEST SUMMARY AND BOARD BINOUT MAP

PROGRAM REV:          SBT3KC
DATE/TIME:            Wed Jan 09 10:47:01 1980

DEVICE TYPE:          NM 93C46
TEST PROGRAM:         T93C46.VER  (CTRUEC.VER)
BOARD TYPE:           STD         DIP  (N/J)   REV C
BOARD S/N:            001
DEVICE STAMPOFF:      SM162044
       SYSTEM #:      01
OPERATOR/SHIFT:       56441

**************************************
*  BIN 1:          0  DEVICES
**************************************
*  BIN X:          0  DEVICES
**************************************
*  BIN 6 (%):      0%
**************************************

    0  0  0  0  1  1  1  1  1  2  2  2  2  2  3  3  3  3  3  4  4  4
    2  4  6  8  0  2  4  6  8  0  2  4  6  8  0  2  4  6  8  0  2  4
  ---------------------------------------------------------------------
1 |55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 |
2 |55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 |
3 |55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 |
4 |55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 |
5 |55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 |
6 |55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 |
7 |55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 |
8 |55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 |
9 |55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 |
10|55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 |
11|55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 55 |
  ---------------------------------------------------------------------

..... NUMBER OF REJECTS .....
BIN2  (CRAL) ..... 0
BIN3  (WRAL) ..... 0
BIN4  (CKBD) ..... 484
BIN5  (INITCHK) ..... 0
BIN6  (INITVEP) ..... 0    ... 0%
BIN7  (INVBITDIAG) ..... 0
BIN8  (EVENBYTE) ..... 0
BIN9  (ODDBYTE) ..... 0
BIN10 (BYTEDIAG) ..... 0
BIN11 (PEONES) ..... 0
BIN12 (PEZEROS) ..... 0
BIN13 (STRESS) ..... 0
BIN14 (EWDSTEST) ..... 0
BIN15 (REGWRITE) ..... 0
BIN16 (WRALL0) ..... 0
BIN17 (WRALL1) ..... 0
BIN20 (REGWRITE0) ..... 0
BIN21 (BITDIAG) ..... 0
BIN22 (INVCKBD) ..... 0
BIN23 (TINVCKBD) ..... 0
BIN24 (TCKBD) ..... 0
BIN X (PECKBD) ..... 0
BIN24 (SOCKETCHECK) ..... 0

.... TOTAL REJECTS ..... 484
```

บทที่ 7

SOFTWARE LISTING

การทำงานของเครื่องนี้ เราจะใช้ Computer เป็นตัวควบคุม โดยการทำการเขียน PROGRAM ใช้ภาษา C เพื่อใช้ในการควบคุมส่วน H/W ของเครื่อง EXERCISER โดยผ่าน Interface Bd. ดังรายละเอียดในบทที่ 4.

ดังมี FLOW CHART การทำงานของโปรแกรมดังนี้



```
/*------------------------------ MAX DEFINITIONS ------------------------------*/
#define         MAX_BLOCKS      50              /*    Maximum test blocks          */
#define         MAX_BOARDS      10              /*    Number of board types        */
#define         MAX_FLOWS       35              /*    Maximum test plans in a file */
#define         MAX_BINS        51              /*    Maximum number of bin outs   */

/*------------------------------------------ TESTS ------------------------------------------*
#define         PASS            1               /*    Passbin definition                     *

#define         ERAL            2               /*    Erase: Array = all 1's                 *
#define         WRAL            3               /*    Write: Array = all 0's                 *
#define         CKBD            4               /*    Checkerboard: Even bits = 1            *
#define         INITCHK         5               /*    Initial read ID code 0101   */
#define         INITVER         6               /*    Initial Verify: R1 = 0101; others = 0000 *
#define         INVBITDIAG      7               /*    Inverse Bit Diagonal:  0's over 1's    *
#define         EVENBYTE        8               /*    Even registers set to 1's              *
#define         ODDBYTE         9               /*    Odd registers set to 1's               *
#define         BYTEDIAG        10              /*    Bytes diagonally erased                *
#define         PEONES          11              /*    Bin # is Kcycles.  Ends in ERAL        *
#define         PEZEROES        12              /*    Bin # is Kcycles.  Ends in WRAL        *
#define         STRESS          13              /*    Use preset SBT2000 stress level        *
#define         EWDSTEST        14              /*    Write A1233456C789D000 to reg01        *
#define         REGWRITE        15              /*    Register Write test (Ricoh 9346)       *
#define         WRALL0          16              /*    Write all 0s to 93CSx6 device          *
#define         WRALL1          17              /*    Write all 1s to 93CSx6 device          *
#define         PRCLEAR         18              /*    Prtect Register CLEAR                  *
#define         PREN            19              /*    Protect Register ENABLE                *
#define         REGWRITE0       20              /*    REGWRITE test for 0s                   *
#define         BITDIAG         21              /*    Bit Diagonal:  1's over 0's            *
#define         INVCKBD         22              /*    Inverse Checkerboard: Odd bits = 1     *
#define         TINVCKBD        23              /*    Inverse Checkerboard: Odd bits = 1     *
#define         TCKBD           24              /*    Inverse Checkerboard: Odd bits = 1     *
#define         PECKBD          25              /*    Write/Read CKBD 'K' number of cycles   *
#define         SOCKETCHECK     26              /*    Socketcheck Register 15 w/CKBD         *
#define         PROGREJN        27              /*    Program pattern for reject unit(nmos)  *
#define         PROGREJC        28              /*    Program pattern for reject unit(cmos)  *

#define         VDO             47
#define         MITSU           48
#define         DATATEST        49

#define         TOTALBAD        50              /*    Total number of failures               *

#define         I2CZEROS        29
#define         I2CONES         30
#define         I2CCKBD         31
#define         I2CBZEROS       32
#define         I2CBONES        33
#define         I2CPE1          34
#define         I2CPE0          35
#define         I2CICKBD        36
#define         I2CB55          37
#define         I2CBAA          38
#define         I2CINIVER       39
#define         I2CINICHK       40

#define         RANDOM          99              /*    Random pattern                         *

/*----------------  DUTs  ----------------*/
```

```
#define         NONE                0

#define         NMC9306             1
#define         NMC9307             2
#define         NMC9313             3
#define         NMC9346             4
#define         NMC9314             5

#define         NM93C06             6
#define         NMC93C06            7
#define         NM93CS06            8

#define         NM88CS06            9

#define         NM93C07             10

#define         NM93C46             11
#define         NM93C46A            12
#define         NMC93C46            13
#define         NM93CS46            14

#define         NM88CS46            15

#define         NM59C11             16

#define         NM93C56             17
#define         NMC93C56            18
#define         NM93CS56            19

#define         NM93C66             20
#define         NMC93C66            21
#define         NM93CS66            22

#define         NM88C16             23
#define         NM59C16             24

#define         NM95C12             25

#define         NM93C13N            26
#define         NM93C14N            27

#define         NM24C02             28
#define         NM24C04             29
#define         NM24C08             30
#define         NM24C16             31

#define         CTRUE               98
#define         CSTRUE              99

/*--------------  BOARD TYPES  --------------*/
#define         DIPC44              0
#define         DIPB46              1
#define         SOA                 2
#define         SO8B                3
#define         SO14A               4
#define         NM95C12DIPA         5
#define         NM95C12SOICA        6
#define         SO14WMA             7
#define         FLATPACK            8
#define         I2C46               9
```

```
/*-------------- SCREEN COLORS ------------*/
#define         BLACK           0
#define         BLUE            1
#define         GREEN           2
#define         CYAN            3
#define         RED             4
#define         MAGENTA         5
#define         BROWN           6
#define         GRAY            7
#define         WHITE           15

/*------------ MODE FLAGS ------------*/
#define         ENABLE          0
#define         DISABLE         1

#define         APON            0
#define         APOFF           1

#define         SCOFF           1
#define         SCON            0

#define         ENGMODE         0
#define         TESTMODE        1

#define         ESC             27
#define         CR              13
#define         SC              0
#define         TEST            1

#define         TOPOLOGICAL     1

#define         SET_62          1

#define         OFF             0
#define         ON              1

/*------------------------------ HARDWARE PORTS ------------------------*/
#define         A1              0x4370
#define         B1              0x4770
#define         C1              0x4B70

#define         A2              0x8370
#define         B2              0x8770
#define         C2              0x8B70          /*  DATA OUT (From DUT)  */
#define         A3              0xC370          /*  ROW ENABLE           */
#define         B3              0xC770
#define         C3              0xCB70

#define         CW1             0x4F70          /*  CS/COLUMN SELECT     */
#define         CW2             0x8F70          /*  DATA IN (To DUT)     */
#define         CW3             0xCF70          /*  VCC ADJUST           */

/*------------------------------ MACRO DEFINITIONS ---------------------*/

/*-------------- OPERATION SETUP: 8255 INITIALIZED IN THE PC ------------*/
/*  Don't ask why this works.  It is used to set up the automatic operating  */
/*  mode for the exercisor.  DON'T CHANGE ANY PARTS OF IT!  The delays were  */
/*  included after the exercisor couldn't keep up with other direct outports */
/*  commands.  The delays are in milliseconds.                                */
```

```
#define     opr_set         outportb(CW1,0x80);\
                            delay(5);\
                            outportb(CW2,0x82);\
                            delay(5);\
                            outportb(CW3,0x80);\
                            delay(5);\
                            outportb(CW1,0x7);\
                            delay(5);\
                            outportb(CW1,0x4)

#define     tip_off         outportb(B1,0x40)    /*  Test-in-progress light  OFF */
#define     tip_on          outportb(B1,0x20)    /*  Test-in-progress light ON   */

#define     cs_high         outportb(CW2,0xF);\
                            outportb(CW2,0xF);\
                            outportb(CW2,0xF);\
                            outportb(CW2,0xF);\
                            outportb(CW2,0xF);\
                            outportb(CW2,0xF);\
                            outportb(CW2,0xF);\
                            outportb(CW2,0xF);\
                            outportb(CW2,0xF);\
                            outportb(CW2,0xF)

#define     cs_low          outportb(CW2,0xE);\
                            outportb(CW2,0xE);\
                            outportb(CW2,0xE);\
                            outportb(CW2,0xE);\
                            outportb(CW2,0xE);\
                            outportb(CW2,0xE);\
                            outportb(CW2,0xE);\
                            outportb(CW2,0xE);\
                            outportb(CW2,0xE);\
                            outportb(CW2,0xE)

#define     sk_toggle       outportb(CW2,0xD);\
                            outportb(CW2,0xC)

#define     clock(f)        for(loop = 0; loop < f; loop++)\
                            {\
                            outportb(CW2,0xD);\
                            outportb(CW2,0xC);\
                            }

#define     latch(f)        outportb(A2,f);\
                            outportb(CW2,0xA);\
                            outportb(CW2,0x8)

#define     column(f)       outportb(A1,f);\
                            outportb(CW1,0x0);\
                            outportb(CW1,0x1)

#define     di_dec(f)       outportb(A1,f);\
                            outportb(CW1,0xA);\
                            outportb(CW1,0x8);\

#define     start_bit       outportb(A2,128);\
                            outportb(CW2,0xA);\
                            outportb(CW2,0x8);\
```

```
                        outportb(CW2,0xD);\
                        outportb(CW2,0xC)

#define     i2c_mode    outportb(CW3,0xf)

#define     uw_mode     outportb(CW3,0xe)

#define     scl_5v      outportb(A3,4)

#define     scl_12v     outportb(A3,1)

#define     scl_5v_12v  outportb(A3,5)

#define     startc      outportb(CW2,0xe);\
                        outportb(A2,0x80);\
                        outportb(CW2,0xa);\
                        outportb(CW2,0xb);\
                        outportb(CW2,0xf);\
                        outportb(A2,0);\
                        outportb(CW2,0xa);\
                        outportb(CW2,0xb);\
                        outportb(CW2,0xe)

#define     stop        outportb(CW2,0xe);\
                        outportb(A2,0x0);\
                        outportb(CW2,0xa);\
                        outportb(CW2,0xb);\
                        outportb(CW2,0xf);\
                        outportb(A2,0x80);\
                        outportb(CW2,0xa);\
                        outportb(CW2,0xb)

#define     data_write(d) outportb(A2,d);\
                        outportb(CW2,0xa);\
                        outportb(CW2,0xb);\
                        for (c=0; c<8; c++)\
                            { outportb(CW2,0xf);\
                              outportb(CW2,0xf);\
                              outportb(CW2,0xe);\
                            }

#define     ack_write   outportb(A2,0x80);\
                        outportb(CW2,0xa);\
                        outportb(CW2,0xb);\
                        outportb(CW2,0xf);\
                        outportb(CW2,0xf);\
                        outportb(CW2,0xe)

#define     data_read   outportb(A2,0xff);\
                        outportb(CW2,0xa);\
                        outportb(CW2,0xb);\
                        for (c=0; c<8; c++)\
                            { outportb(CW2,0xf);\
                              outportb(CW2,0xf);\
                              outportb(CW2,0xe);\
                            }

#define     ack_read    outportb(A2,0);\
                        outportb(CW2,0xa);\
                        outportb(CW2,0xb);\
```

```
                        outportb(CW2,0xf);\
                        outportb(CW2,0xf);\
                        outportb(CW2,0xe)

/*---------------------------  GLOBAL VARIABLES  ----------------------------*/
unsigned    int         d_buffer[2048];             /*   Collects register data          */
unsigned    int         t_vector[2048];             /*   Expect data buffer              */
unsigned    int         tss_vector[64];             /*   Bitmap single-step test vector  */
unsigned    int         p_vector[2048];             /*   To store random pattern         */
unsigned    int         b_mask[16];                 /*   Bit masks 0x1,0x2,0x4,0x8...    */
unsigned    int         f;                          /*   Universal macro variable        */
unsigned    int         dut_reg;                    /*   Number of DUT registers         */
unsigned    int         reg_rotate;                 /*   reg rotated for DUT sending     */
unsigned    int         reg2_rotate;                /*   reg2 rotated for DUT sending    */
unsigned    int         reg2;                       /*   reg2                            */
unsigned    int         lk;                         /*   Data moving                     */
unsigned    int         lpn,Nprt;                   /*   Bitmap autoprint option         */
unsigned    int         cycles;                     /*   Cycle counter in PECKBD         */

int         twp;                        /*   Device twp or tWP                  */
int         dut_offset;                 /*   Number of shifts to send reg to DUT    */
int         dut_adrbits;                /*   Number of device address bits      */
int         dut_opbits;                 /*   Number of device opcode bits       */
int         true_dut;                   /*   Actual device family               */
int         dut_type;                   /*   Device type selected               */
int         board_type;                 /*   Board type selected                */

int         print_stat;                 /*   Board type selected                */
int         run_stat;                   /*   Bitmap/Calibration flag            */
int         C12_set_stat;               /*   95C12 Register62 set to 0s flag    */
int         bit_stat;                   /*   Bitmap topological/logical flag    */
int         screen_stat;                /*   Topological Bitmap screen          */
int         step_stat;                  /*   Single_step using instruction      */
int         peckbd_stat;                /*   PECKBD is testing if ON            */
int         peckbd_1st_stat;            /*   Set for 1st time thru PECKBD       */
int         socket_stat;                /*   Socketcheck flag                   */

int         test_type;                  /*   Test Plan selected                 */
int         loop;                       /*   Universal macro loop counter       */
unsigned int  loop1;
unsigned int high;
int         low;
int         count;                      /*   Max blocks in selected plan        */
int         start;
int         column_set;                 /*   Controls go_test_whole_board       */
int         i,j,k,dn,lp,lpt,rsterr;     /*   Bitmap stuff                       */

int         row,done,next;
int         page,slaveadd,wordadd,datab,loop2;

int         enable_send;
int         disable_send;
int         read_send;
int         eral_send;
int         wral_send;
int         erase_send;
int         write_send;
int         wrall_send;

int         bin_count[MAX_BINS];        /*   Board bin out counters             */
```

```
int         vccindex[9];                /*   Vcc calibration counters          */
int         dut_board[512];             /*   Loadboard Bin out results array   */
int         autoprint;                  /*   Used for print screen in bit map  */

int         vccpoint[8];                /*   Vcc calibration set points        */

int         max_row[10];

int         max_col[10];

int         max_duts[10];

int          c,d,e,chip;

float       m;                          /*   Slope of Vcc line                 */
float       b;                          /*   Intercept of Vcc line             */
float       percent;                    /*   Calculates % failures             */
long        sec_now;
long         ff;

struct      date    today;
struct      time    now;
struct      tm *tm_now;

struct      t_blk                       /*   Test block structure              */
{
    int         bin;                    /*   Bin #, 0 means program cycle      */
    int         volt;                   /*   Volts in millivolts               */
    int         number;                 /*   Test Block Index number           */
    char        name[13];               /*   Test Block name                   */
};

struct    t_blk    t[MAX_BLOCKS];       /*   Array of MAX_BLOCK test blocks    */

char    test[MAX_FLOWS][13];

char    stampoff[20];                   /*   Stampoff                          */
char    tbsn[10];                       /*   Test Board S/N                    */
char    opr[20];                        /*   Operator ID                       */
char    sysn[5];                        /*   System number                     */
char    device[20];                     /*   Actual device type base           */
char    input[20];                      /*   Misc. input                       */
char    test_input[20];                 /*   Single-step test input            */
char    testprogram[20];                /*   Test program selected             */
char    dut_test[20];                   /*   Device type base                  */
char    opr_test[20];                           /*   Test program operator sees */
char    bit[4096];                      /*   bit map                           */
char    ran[20];                        /*   random pattern name               */
char    out_name[20];
char    ntemp[20];

char    CS_vector[64];                  /*   Single-step CS vector to DUT      */
char    SK_vector[64];                  /*   Single-step SK vector to DUT      */
char    DIN_vector[64];                 /*   Single-step DIN vector to DUT     */

char    password[20];                   /*   Calibration password              */
char    calpass[20];                    /*   Calibration password IN FILE      */

char    plk[20];
```

```
char    p1kt[20];
char    p5kt[20];
char    p10k[20];
char    p50k[20];
char    pver[20];
char    bin_1[5];                       /*    BIN 1 counter for printing    */

char    ch;

char    *rev;                           /*    Test Program Revision Level   */
char    *str_now;

char    *board[10];

char    *boardenter[10];

FILE    *fopen(),*infile;           /*  Standard I/O definitions    */
FILE    *outfile;                   /*  Standard I/O definitions    */


/* ---------------------------  FUNCTIONS  ---------------------------*/
void    test_set();                     /* Converts string input to TEST numbers  */
int     bitmap();                       /* Bitmap                                 */
void    vcc_cal();                      /* Vcc calibration                        */
void    ask_for_id();                   /* Ask for operator information           */
int     select_mos();                   /* Select nmos or cmos type               */
void    select_nmos_device();
void    select_cmos_device();
void    get_op_code();                  /* Setup opcodes                          */
void    select_i2c_device();
void    select_test_program();          /* Select test program to use             */
int     read_test_program();            /* Read test program contents             */
void    select_board_type();            /* Select DUT board to be used            */
int     verify_selections();            /* Verify selections entered              */
int     enter_dut_board_no();
void    print_summary();
void    display_board_map();
void    topological_display_bit_map();
void    logical_display_bit_map();
void    print_fail_dut_locations();
void    socket_check();
void    go_test_whole_board();
int     chose_map_or_quit();
void    map();
void    print_screen();
void    save_pattern();
void    able();
void    strprint();
```

```
/*----------  INCLUDE FILES  ---------*/
#include    <stdio.h>
#include    <dos.h>
#include    <conio.h>
#include    <dir.h>
#include    <math.h>
#include    <io.h>
#include    <sys/stat.h>
#include    <time.h>
#include    <string.h>
#include    <\mill\variable.h>
#include    <\mill\function.h>

main()
{

int         i;                          /*    General Purpose Counter          */
int         j;                          /*    General Purpose Counter          */
int         k;                          /*    Temporary data storage           */
int         l;                          /*    Temporary data storage           */
int         n;                          /*    Test data                        */

int         mos;
char        *p1k,*p1kt,*p5kt,*p10k,*p50k,*pver;
twp=0;

f=0;
dut_reg=0;
twp=0;
dut_offset=0;
dut_adrbits=0;
dut_opbits=0;
true_dut=0;
dut_type=0;
board_type=0;
print_stat=0;
run_stat=0;
C12_set_stat=0;
bit_stat=0;
screen_stat=0;
step_stat=0;
peckbd_stat=0;
test_type=0;
loop=0;
count=0;
vccpoint[0]=4000;
vccpoint[1]=4200;
vccpoint[2]=4400;
vccpoint[3]=5000;
vccpoint[4]=5500;
vccpoint[5]=5600;
vccpoint[6]=5800;
vccpoint[7]=7500;
vccpoint[8]=8000;
max_row[0]=11;
```

```
max_row[1]=11;
max_row[2]=6;
max_row[3]=6;
max_row[4]=6;
max_row[5]=7;
max_row[6]=6;
max_row[7]=6;
max_row[8]=4;
max_row[9]=11;
max_col[0]=44;
max_col[1]=46;
max_col[2]=34;
max_col[3]=36;
max_col[4]=18;
max_col[5]=23;
max_col[6]=18;
max_col[7]=18;
max_col[8]=20;
max_col[9]=46;
max_duts[0]=484;
max_duts[1]=506;
max_duts[2]=204;
max_duts[3]=216;
max_duts[4]=108;
max_duts[5]=161;
max_duts[6]=108;
max_duts[7]=108;
max_duts[8]=80;
max_duts[9]=506;
rev="SBT4KA";
board[0]="STD44      DIP(N/J)    REV C";
board[1]="STD46      DIP(N/J)    REV B";
board[2]="STD34      SOIC (M/M8) REV A";
board[3]="STD36      SOIC (M8)   REV B";
board[4]="STD18      SOIC (M14)  REV A";
board[5]="NM95C12    DIP  (N/J)  REV A";
board[6]="NM95C12    SOIC (M)    REV A";
board[7]="STD18      SOIC (WM14) REV A";
board[8]="STD20      FLATPACK    REV D";
board[9]="I2C46      DIP  (N/J)  REV A";
boardenter[0]="[0]";
boardenter[1]="[1]";
boardenter[2]="[2]";
boardenter[3]="[3]";
boardenter[4]="[4]";
boardenter[5]="[5]";
boardenter[6]="[6]";
boardenter[7]="[7]";
boardenter[8]="[8]";
boardenter[9]="[9]";

chdir("c:/sbt");                        /*    Get to the correct directory    */
textattr(WHITE+(BLUE<<4));              /*    Text = white; background = blue */
clrscr();
_chmod("stat.dat",S_IREAD,FA_HIDDEN);

if((infile = fopen("stat.dat","rb")) == NULL)
{
    printf("Error:  Cannot open stat file!  Recreating it....\n\n");
    creat("c:/sbt/stat.dat",S_IREAD);
```

```
    printf("NOTE:  All calibration and twp data must be re-entered.\n\n");
    printf("Have an engineer make the proper data calibration.\n\n");
    printf("Press ENTER to continue");
    getch();
    vcc_cal();
}

for(i = 0; i<9; i++) fread(&vccindex[i],sizeof(vccindex[i]),1,infile);
fclose(infile);

opr_set;                                                    /*  Operation setup     */
uw_mode;
scI_5v;
vcc_set();                                                  /*  VCC setup           */
delay(200);
set_vcc(0);
save_vcc();

/*  The program has a lookup table for the following Vcc levels:
    4.0V, 4.2V, 4.4V, 5.0V, 5.5V, 5.6V, 5.8V, 7.5V and 8.0V.
    Any test that require one of these levels will use the lookup
    table entry to set the Vcc.  If the Vcc for the test is not in the
    table, then it will use a linear formula based on two of the values.
    This insures that the system will always have a setting for any given
    voltage.  M is the slope of the function, and b is the intercept.
                                                                */
m = (1.0 * vccindex[5] - 1.0 * vccindex[1])/1.4;
b = 1.0 * vccindex[5] - m*5.6;

window(1,1,80,25);
clrscr();
textcolor(YELLOW);
highvideo();
gotoxy(1,1);
cprintf("PRODUCTION TEST PROGRAM");
gotoxy(73,1);
cprintf("(%s)",rev);
lowvideo();
textcolor(WHITE);
window(1,2,80,25);
clrscr();

if(chdir("c:/sbt/tests4") == -1)
{
    gotoxy(20,20);
    printf("FATAL ERROR(2):  UNABLE TO LOCATE TEST DIRECTORY");
    printf("\n\nPRESS <ENTER> TO EXIT");
    getch();
    exit(2);
}
/*;-------------------------------------------------------------------------*/
while(1)                                                    /* Main program loop
{

set_vcc(0);
save_vcc();
    while(1)                                                /* Operator interface loop
    {
        print_stat   = APON;                                /* 0 = Automatic Summary Printout
        run_stat     = ENGMODE;                             /* 0 = Production Test Mode
```

```
        while(run_stat == ENGMODE) ask_for_id();   /* Screen #1 to enter operator information
        mos=select_mos();
        if (mos == 0)  select_nmos_device();                    /* Screen #2 for chose
        if (mos == 1)  select_cmos_device();                    /* Screen #3
        if (mos == 2) select_i2c_device();
        select_test_program();
        count=read_test_program();                              /* Screen #4,select tes
        select_board_type();                                    /* Read test program co
        if(verify_selections()) break;                          /* Screen #5, select DU
    }                                                           /* Screen #6, verify se

/*************************************************************/
/*  This is the grand test loop.  The program repeats to */
/*  here when the user selects "T" to test another board */
/*************************************************************/
    while(1)
    {
        start=enter_dut_board_no();                 /*  Enter bd# and start test
        set_vcc(0);
        save_vcc();
        column_set = 0;
                                                    /*  0 = Test Whole Board
        bin_count[PASS] = max_duts[board_type];     /*  Set BIN 1 counter to #DUTs on board
        for(i = 2; i<MAX_BINS; i++) bin_count[i] = 0;   /*  Initialize BIN counters to 0

        for(i = 0; i < max_duts[board_type]+1; i++) dut_board[i] = 1;

        if(start==SC)  socket_check();
        else  go_test_whole_board();

        tip_off;
        set_vcc(0);                                 /*  Turn off TIP light      */
        save_vcc();                                 /*  Turn off the power      */
        percent = bin_count[TOTALBAD] + bin_count[PASS] - bin_count[INITCHK];       /*  PASS +
        if (percent!=0)  percent = 100*(bin_count[INITVER]/percent);

        if(print_stat == APON) print_summary();
        if (chose_map_or_quit()) break;             /* if return 1, go back to  */
    }                                               /* main loop and start over */
}                                                   /* end of main loop         */

}                                                   /* end of main              */

/*------------------------ CONVERT STRING INPUT TO TEST NUMBERS ------------------------*/
void test_set(int i)
{
    if(!stricmp(t[i].name,"INITCHK"))       t[i].number = INITCHK;
    if(!stricmp(t[i].name,"INITVER"))       t[i].number = INITVER;
    if(!stricmp(t[i].name,"ERAL"))          t[i].number = ERAL;
    if(!stricmp(t[i].name,"WRAL"))          t[i].number = WRAL;
    if(!stricmp(t[i].name,"CKBD"))          t[i].number = CKBD;
    if(!stricmp(t[i].name,"INVCKBD"))       t[i].number = INVCKBD;
    if(!stricmp(t[i].name,"TINVCKBD"))      t[i].number = TINVCKBD;
    if(!stricmp(t[i].name,"TCKBD"))         t[i].number = TCKBD;
    if(!stricmp(t[i].name,"BITDIAG"))       t[i].number = BITDIAG;
    if(!stricmp(t[i].name,"INVBITDIAG"))    t[i].number = INVBITDIAG;
    if(!stricmp(t[i].name,"EVENBYTE"))      t[i].number = EVENBYTE;
    if(!stricmp(t[i].name,"ODDBYTE"))       t[i].number = ODDBYTE;
    if(!stricmp(t[i].name,"BYTEDIAG"))      t[i].number = BYTEDIAG;
```

```
    if(!stricmp(t[i].name,"PEONES"))        t[i].number = PEONES;
    if(!stricmp(t[i].name,"PEZEROES"))      t[i].number = PEZEROES;
    if(!stricmp(t[i].name,"STRESS"))        t[i].number = STRESS;
    if(!stricmp(t[i].name,"EWDSTEST"))      t[i].number = EWDSTEST;
    if(!stricmp(t[i].name,"REGWRITE"))      t[i].number = REGWRITE;
    if(!stricmp(t[i].name,"WRALL0"))        t[i].number = WRALL0;
    if(!stricmp(t[i].name,"WRALL1"))        t[i].number = WRALL1;
    if(!stricmp(t[i].name,"PECKBD"))        t[i].number = PECKBD;
    if(!stricmp(t[i].name,"PROGREJN"))      t[i].number = PROGREJN;
    if(!stricmp(t[i].name,"PROGREJC"))      t[i].number = PROGREJC;
    if(!stricmp(t[i].name,"I2CZEROS"))      t[i].number = I2CZEROS;
    if(!stricmp(t[i].name,"I2CONES"))       t[i].number = I2CONES;
    if(!stricmp(t[i].name,"I2CCKBD"))       t[i].number = I2CCKBD;
    if(!stricmp(t[i].name,"I2CBZEROS"))     t[i].number = I2CBZEROS;
    if(!stricmp(t[i].name,"I2CBONES"))      t[i].number = I2CBONES;
    if(!stricmp(t[i].name,"I2CPE1"))        t[i].number = I2CPE1;
    if(!stricmp(t[i].name,"I2CPE0"))        t[i].number = I2CPE0;
    if(!stricmp(t[i].name,"DATATEST"))      t[i].number = DATATEST;
    if(!stricmp(t[i].name,"VDD"))           t[i].number = VDD;
    if(!stricmp(t[i].name,"MITSU"))         t[i].number = MITSU;
    if(!stricmp(t[i].name,"I2CICKBD"))      t[i].number = I2CICKBD;
    if(!stricmp(t[i].name,"I2CB55"))        t[i].number = I2CB55;
    if(!stricmp(t[i].name,"I2CBAA"))        t[i].number = I2CBAA;
    if(!stricmp(t[i].name,"I2CINIVER"))     t[i].number = I2CINIVER;
    if(!stricmp(t[i].name,"I2CINICHK"))     t[i].number = I2CINICHK;
}

/*------------------------------- MAIN OPTION SCREEN -------------------------------*/
int     chose_map_or_quit()
{
    while(1)
    {
        textattr(WHITE+(BLUE<<4));          /*   Text = white; background = blue   */
        input[0] = '\0';                    /*   Reset to 0                        */
        clrscr();

        gotoxy(10,5);
        cprintf("ENTER  [0]     DISPLAY BOARD MAP AND TEST SUMMARY");
        gotoxy(10,7);
        cprintf("ENTER  [1]     PRINT THE BOARD MAP AND TEST SUMMARY");
        gotoxy(10,9);
        cprintf("ENTER  [2]     GET A HARD COPY OF THE FAIL DUT LOCATIONS");
        gotoxy(10,11);
        cprintf("ENTER  [3]     TEST ANOTHER BOARD");
        gotoxy(10,13);
        cprintf("ENTER  [4]     CHANGE THIS DUT/BOARD/PLAN SELECTION");
        gotoxy(10,15);
        cprintf("ENTER  [Q]     QUIT");

        gotoxy(10,20);
        cprintf("SELECTION:      ");
        gets(input);

        if(!stricmp(input,"Q"))  {chdir("c:/sbt"); exit(0);}
        if(!stricmp(input,"0"))  display_board_map();
        if(!stricmp(input,"1"))  print_summary();
        if(!stricmp(input,"2"))  print_fail_dut_locations();
        if(!stricmp(input,"3"))  return(0);                 /* Return 0 to test another board */
        if(!stricmp(input,"4"))  return(1);                 /* Return 1 to go back to begining */
    }                                                       /* End of while loop */
```

```
}                                                       /* End of choose_map_or_quit */

/*---------------------------------- PERFORM TESTS ----------------------------------*/
void go_test_whole_board()
{
        int i;

        textattr(WHITE+(GREEN<<4));                     /*  Text = white; background = green */
        clrscr();
        gotoxy(20,1);
        cprintf("TEST PROGRAM %s (%s) IN PROGRESS",opr_test,testprogram);
        gotoxy(20,3);
        cprintf("#");
        gotoxy(25,3);
        cprintf("NAME");
        gotoxy(39,3);
        cprintf("Vcc (V)");
        gotoxy(50,3);
        cprintf("BIN");
        gotoxy(20,4);
        cprintf("-");
        gotoxy(25,4);
        cprintf("----");
        gotoxy(39,4);
        cprintf("--------");
        gotoxy(50,4);
        cprintf("---");
        window(1,6,80,25);

        column(column_set);

        tip_on;                                         /*  Test in Progress        */
        i = 0;                                          /*  Insure initial values   */

        for(i=0; i<count; i++)                          /*  Do all test blocks      */
        {
            if (i == 0) C12_set_stat = SET_62;          /*  Program Register62 to 0s    */
            else        C12_set_stat = 0;

            if (i%18==0) clrscr();                      /* new screen if more than 18 tests */
            gotoxy(20,1+i%18);
            printf("%-2d",i);
            gotoxy(25,1+i%18);
            printf("%s",t[i].name);
            gotoxy(40,1+i%18);
            printf("%2.3f",t[i].volt/1000.0);                       /*  Print DUT VCC     */
            gotoxy(51,1+i%18);
            printf("%d",t[i].bin);
            gotoxy(57,1+i%18);
            if(bin_count[TOTALBAD]==max_duts[board_type]) break;
            set_vcc(t[i].volt);                                     /*  Set VCC for tests

            switch(t[i].number)
            {
               case     PEONES:     pecycles1(t[i].bin);            break;
               case     PEZEROES:   pecycles0(t[i].bin);            break;
               case     PECKBD:     peckbd(t[i].volt);              break;
               case     STRESS:     stress();                       break;
               case     REGWRITE:   regwrite(t[i].volt);            break;
```

```
                case    PROGREJN:   progrejn(t[i].volt);        break;
                case    PROGREJC:   progrejc(t[i].volt);        break;
                case    I2CPE1:     i2cpeones(t[i].bin);        break;
                case    I2CPE0:     i2cpezeros(t[i].bin);       break;
                case    DATATEST:   datatest(t[i].bin);         break;
                case    VDO:        vdo(t[i].bin);              break;
                case    MITSU:      mitsu(t[i].bin);            break;

                default:        if(t[i].bin == 0) program(t[i].number);   /* Bin = 0 means program
                                else
                                {
                                    printf("Verify \n");                /* bin != 0 means verify    *
                                    verify(t[i].number);
                                }
            }
            tip_on;                                                     /* End of switch statment   *
        }                                                               /* Test in Progress         *
        window(1,2,80,25);                                              /* End of for loop          *
}                                                                   /*  End of go_test_whole_board */

/*------------------------------ PRINT TEST SUMMARY AND BOARD MAP ------------------------------
void    print_summary()
{
    char    *message    =   "                                                                   ";
    int i,k,col;

    time(&sec_now);
    tm_now = localtime(&sec_now);
    str_now = asctime(tm_now);

    strprint("          EXERCISOR TEST SUMMARY AND BOARD BINOUT MAP");

    if(Nprt>0)
    {
        itoa(lp+1,message,10);
        strprint("  (*** LOOP #:  ");
        strprint(message);
        strprint(" ***)");
    }

    biosprint(0,10,0);
    biosprint(0,10,0);

    strprint("          PROGRAM REV:      ");
    strcpy(message,rev);
    strprint(message);
    biosprint(0,10,0);

    strprint("          DATE/TIME:        ");
    strprint(str_now);
    biosprint(0,10,0);

    strprint("          DEVICE TYPE:      ");
    strcpy(message, device);
    strprint(message);
    biosprint(0,10,0);

    strprint("          TEST PROGRAM:     ");
        if(socket_stat == SCON)   strprint("Socketcheck");
    else
```

```
{
    strprint(opr_test);
    strprint("  (");
    strprint(testprogram);
    strprint(")");
}
biosprint(0,10,0);

strprint("          BOARD TYPE:       ");
strcpy(message,board[board_type]);
strprint(message);
biosprint(0,10,0);

strprint("          BOARD S/N:        ");
strcpy(message,tbsn);
strprint(message);
biosprint(0,10,0);

strprint("          DEVICE STAMPOFF:  ");
strcpy(message,stampoff);
strprint(message);
biosprint(0,10,0);

strprint("                SYSTEM #:   ");
strcpy(message,sysn);
strprint(message);
biosprint(0,10,0);
strprint("          OPERATOR/SHIFT:   ");
strcpy(message,opr);
strprint(message);
biosprint(0,10,0);
biosprint(0,10,0);

strprint("          ***************************************");
biosprint(0,10,0);
strprint("          *  BIN 1:           ");
itoa(bin_count[PASS],message,10);
strprint(message);
strprint("  DEVICES");
biosprint(0,10,0);

strprint("          ***************************************");
biosprint(0,10,0);
strprint("          *  BIN X:           ");
itoa(bin_count[PECKBD],message,10);
strprint(message);
strprint("  DEVICES");
biosprint(0,10,0);

strprint("          ***************************************");
biosprint(0,10,0);
strprint("          *  BIN 6 (%):       ");
strprint(gcvt(percent,3,message));
strprint("%");
biosprint(0,10,0);
strprint("          ***************************************");
biosprint(0,10,0);

biosprint(0,10,0);
if((board_type==DIPB46) || (board_type==I2C46))
```

```
strprint("      0   0   0   0   1   1   1  -1   1   2   2   2   2   2   3   3   3   3   3   4   4   4   4");
if(board_type==DIPC44)
strprint("      0   0   0   0   1   1   1   1   1   2   2   2   2   2   3   3   3   3   3   4   4   4");
if(board_type==SO8B)
strprint("      0   0   0   0   1   1   1   1   1   2   2   2   2   2   3   3   3   3");
if(board_type==SOA)
strprint("      0   0   0   0   1   1   1   1   1   2   2   2   2   2   3   3   3");
if(board_type==NM95C12DIPA)
strprint("      0   0   0   0   1   1   1   1   1   2   2");
if(board_type==FLATPACK)
strprint("      0   0   0   0   1   1   1   1   1   2");
if((board_type==SO14A) || (board_type==NM95C12SOICA) || (board_type==SO14WMA))
strprint("      0   0   0   0   1   1   1   1   1");
biosprint(0,10,0);
if((board_type==DIPB46) || (board_type==I2C46))
strprint("      2   4   6   8   0   2   4   6   8   0   2   4   6   8   0   2   4   6   8   0   2   4   6");
if(board_type==DIPC44)
strprint("      2   4   6   8   0   2   4   6   8   0   2   4   6   8   0   2   4   6   8   0   2   4");
if(board_type==SO8B)
strprint("      2   4   6   8   0   2   4   6   8   0   2   4   6   8   0   2   4   6");
if(board_type==SOA)
strprint("      2   4   6   8   0   2   4   6   8   0   2   4   6   8   0   2   4");
if(board_type==NM95C12DIPA)
strprint("      2   4   6   8   0   2   4   6   8   0   2");
if(board_type==FLATPACK)
strprint("      2   4   6   8   0   2   4   6   8   0");
if((board_type==SO14A) || (board_type==NM95C12SOICA) || (board_type==SO14A))
strprint("      2   4   6   8   0   2   4   6   8");
biosprint(0,10,0);

for(i = 0; i<4; i++) biosprint(0,32,0);

for(i=0; i<(max_col[board_type]/2)*3; i++)
biosprint(0,'-',0);

biosprint(0,10,0);

for(row = 1; row < max_row[board_type]+1; row++)
    {
    biosprint(0,32,0);

    if(row < 10)
        {
        biosprint(0,48+row,0);
        biosprint(0,32,0);
        }
    if(row == 10)   strprint("10");
    if(row == 11)   strprint("11");
    biosprint(0,'|',0);

    for(col = 1; col<=max_col[board_type]; col++)
        {
        k = dut_board[(row-1)*max_col[board_type] + col];

        if(k == 1 ) biosprint(0,'.',0);
        else if(k == 6 ) biosprint(0,'6',0);
        else if(k == 25 ) biosprint(0,'X',0);
        else biosprint(0,'5',0);
        if((col%2) == 0) biosprint(0,32,0);
        }
```

```
        biosprint(0,'|',0);
        biosprint(0,10,0);
        }

    for(i = 0; i<4; i++) biosprint(0,32,0);
    for(i=0; i< (max_col[board_type]/2)*3; i++)
    biosprint(0,'-',0);

    biosprint(0,10,0);

    for(i = 0; i<4; i++) biosprint(0,32,0);
    strprint("..... NUMBER OF REJECTS .....  ");
    biosprint(0,10,0);

if(board_type != I2C46)
  {

    for(i = 0; i<4; i++) biosprint(0,32,0);
    strprint("BIN2  (ERAL) ..... ");
    itoa(bin_count[ERAL],message,10);
    strprint(message);
    biosprint(0,10,0);

    for(i = 0; i<4; i++) biosprint(0,32,0);
    strprint("BIN3  (WRAL) ..... ");
    itoa(bin_count[WRAL],message,10);
    strprint(message);
    biosprint(0,10,0);

    for(i = 0; i<4; i++) biosprint(0,32,0);
    strprint("BIN4  (CKBD) ..... ");
    itoa(bin_count[CKBD],message,10);
    strprint(message);
    biosprint(0,10,0);

    for(i = 0; i<4; i++) biosprint(0,32,0);
    strprint("BIN5  (INITCHK) ..... ");
    itoa(bin_count[INITCHK],message,10);
    strprint(message);
    biosprint(0,10,0);

    for(i = 0; i<4; i++) biosprint(0,32,0);
    strprint("BIN6  (INITVER) ..... ");
    itoa(bin_count[INITVER],message,10);
    strprint(message);
    strprint("  ... ");
    strprint(gcvt(percent,3,message));
    biosprint(0,37,0);
    biosprint(0,10,0);

    for(i = 0; i<4; i++) biosprint(0,32,0);
    strprint("BIN7  (INVBITDIAG) ..... ");
    itoa(bin_count[INVBITDIAG],message,10);
    strprint(message);
    biosprint(0,10,0);

    for(i = 0; i<4; i++) biosprint(0,32,0);
    strprint("BIN8  (EVENBYTE) ..... ");
    itoa(bin_count[EVENBYTE],message,10);
    strprint(message);
```

```
biosprint(0,10,0);

for(i = 0; i<4; i++) biosprint(0,32,0);
strprint("BIN9  (ODDBYTE) ........ ");
itoa(bin_count[ODDBYTE],message,10);
strprint(message);
biosprint(0,10,0);

for(i = 0; i<4; i++) biosprint(0,32,0);
strprint("BIN10 (BYTEDIAG) ....(A) ");
itoa(bin_count[BYTEDIAG],message,10);
strprint(message);
biosprint(0,10,0);

for(i = 0; i<4; i++) biosprint(0,32,0);
strprint("BIN11 (PEONES) ......(B) ");
itoa(bin_count[PEONES],message,10);
strprint(message);
biosprint(0,10,0);

for(i = 0; i<4; i++) biosprint(0,32,0);
strprint("BIN12 (PEZEROS) .....(C) ");
itoa(bin_count[PEZEROES],message,10);
strprint(message);
biosprint(0,10,0);

for(i = 0; i<4; i++) biosprint(0,32,0);
strprint("BIN13 (STRESS) ......(D) ");
itoa(bin_count[STRESS],message,10);
strprint(message);
biosprint(0,10,0);

for(i = 0; i<4; i++) biosprint(0,32,0);
strprint("BIN14 (EWDSTEST) ....(E) ");
itoa(bin_count[EWDSTEST],message,10);
strprint(message);
biosprint(0,10,0);

for(i = 0; i<4; i++) biosprint(0,32,0);
strprint("BIN15 (REGWRITE) ....(F) ");
itoa(bin_count[REGWRITE],message,10);
strprint(message);
biosprint(0,10,0);

for(i = 0; i<4; i++) biosprint(0,32,0);
strprint("BIN16 (WRALL0) ......(G) ");
itoa(bin_count[WRALL0],message,10);
strprint(message);
biosprint(0,10,0);

for(i = 0; i<4; i++) biosprint(0,32,0);
strprint("BIN17 (WRALL1) ......(H) ");
itoa(bin_count[WRALL1],message,10);
strprint(message);
biosprint(0,10,0);

for(i = 0; i<4; i++) biosprint(0,32,0);
strprint("BIN20 (REGWRITE0) ...(K) ");
itoa(bin_count[REGWRITE],message,10);
strprint(message);
```

```
    biosprint(0,10,0);

    for(i = 0; i<4; i++) biosprint(0,32,0);
    strprint("BIN21 (BITDIAG) .....(L) ");
    itoa(bin_count[BITDIAG],message,10);
    strprint(message);
    biosprint(0,10,0);

    for(i = 0; i<4; i++) biosprint(0,32,0);
    strprint("BIN22 (INVCKBD) .....(M) ");
    itoa(bin_count[INVCKBD],message,10);
    strprint(message);
    biosprint(0,10,0);

    for(i = 0; i<4; i++) biosprint(0,32,0);
    strprint("BIN23 (TINVCKBD) ....(N) ");
    itoa(bin_count[TINVCKBD],message,10);
    strprint(message);
    biosprint(0,10,0);

    for(i = 0; i<4; i++) biosprint(0,32,0);
    strprint("BIN24 (TCKBD) .......(O) ");
    itoa(bin_count[TCKBD],message,10);
    strprint(message);
    biosprint(0,10,0);

    for(i = 0; i<4; i++) biosprint(0,32,0);
    strprint("BIN X (PECKBD) ......... ");
    itoa(bin_count[PECKBD],message,10);
    strprint(message);
    if(bin_count[PECKBD] > 0) {strprint("  *** BIN X 50K P/E FAILURE(S) IN LOT ***");}
    biosprint(0,10,0);

    for(i = 0; i<4; i++) biosprint(0,32,0);
    strprint("BIN26 (SOCKETCHECK) .(Q) ");
    itoa(bin_count[SOCKETCHECK],message,10);
    strprint(message);
    biosprint(0,10,0);
  }

if(board_type==I2C46)
  { for(i=0; i<4; i++) biosprint(0,32,0);
    strprint("BIN  2 (I2CONES) .....(U) ");
    itoa(bin_count[I2CONES],message,10);
    strprint(message);
    biosprint(0,10,0);

    for(i=0; i<4; i++) biosprint(0,32,0);
    strprint("BIN  3 (I2CZEROS) ....(T) ");
    itoa(bin_count[I2CZEROS],message,10);
    strprint(message);
    biosprint(0,10,0);

    for(i=0; i<4; i++) biosprint(0,32,0);
    strprint("BIN  4 (I2CCKBD) .....(V) ");
    itoa(bin_count[I2CCKBD],message,10);
    strprint(message);
    biosprint(0,10,0);

    for(i=0; i<4; i++) biosprint(0,32,0);
```

```
    strprint("BIN  5 (I2CINICHK) ...(_) ");
    itoa(bin_count[I2CINICHK],message,10);
    strprint(message);
    biosprint(0,10,0);

    for(i=0; i<4; i++) biosprint(0,32,0);
    strprint("BIN  6 (I2CINIVER) ...(^) ");
    itoa(bin_count[I2CINIVER],message,10);
    strprint(message);
    biosprint(0,10,0);

    for(i=0; i<4; i++) biosprint(0,32,0);
    strprint("BIN 11 (I2CPE1) ......(Y) ");
    itoa(bin_count[I2CPE1],message,10);
    strprint(message);
    biosprint(0,10,0);

    for(i=0; i<4; i++) biosprint(0,32,0);
    strprint("BIN 12 (I2CPE0) ......(Z) ");
    itoa(bin_count[I2CPE0],message,10);
    strprint(message);
    biosprint(0,10,0);

    for(i=0; i<4; i++) biosprint(0,32,0);
    strprint("BIN 16 (I2CBZEROS) ...(W) ");
    itoa(bin_count[I2CBZEROS],message,10);
    strprint(message);
    biosprint(0,10,0);

    for(i=0; i<4; i++) biosprint(0,32,0);
    strprint("BIN 17 (I2CBONES) ....(X) ");
    itoa(bin_count[I2CBONES],message,10);
    strprint(message);
    biosprint(0,10,0);

    for(i=0; i<4; i++) biosprint(0,32,0);
    strprint("BIN 22 (I2CICKBD) ....(E) ");
    itoa(bin_count[I2CICKBD],message,10);
    strprint(message);
    biosprint(0,10,0);

    for(i=0; i<4; i++) biosprint(0,32,0);
    strprint("BIN 26 (SOCKETCHECK) .(Q) ");
    itoa(bin_count[SOCKETCHECK],message,10);
    strprint(message);
    biosprint(0,10,0);

    for(i=0; i<4; i++) biosprint(0,32,0);
    strprint("BIN 37 (I2CB55) ......(/) ");
    itoa(bin_count[I2CB55],message,10);
    strprint(message);
    biosprint(0,10,0);

    for(i=0; i<4; i++) biosprint(0,32,0);
    strprint("BIN 38 (I2CBAA) ......(]) ");
    itoa(bin_count[I2CBAA],message,10);
    strprint(message);
    biosprint(0,10,0);
}
```

```
    for(i = 0; i<4; i++) biosprint(0,32,0);
    strprint("--------------------------");
    biosprint(0,10,0);
    for(i = 0; i<4; i++) biosprint(0,32,0);
    strprint("..... TOTAL REJECTS ..... ");
    itoa(bin_count[TOTALBAD],message,10);
    strprint(message);
    biosprint(0,12,0);
    }
                                        /* end of function print_summary */

/*---------------------------------- DISPLAY BOARD MAP ----------------------------------*/
void    display_board_map()
{
    int i,k,col;

    clrscr();
    gotoxy(30,1);
    cprintf("BOARD SUMMARY BINOUT MAP");
    gotoxy(15,3);
    cprintf("DEVICE TYPE :      %s",device);
    gotoxy(15,4);
    if(socket_stat == SCON)
            cprintf("TEST PROGRAM:      Socketcheck");
    else    cprintf("TEST PROGRAM:      %s  (%s)",opr_test,testprogram);
    gotoxy(15,5);
    cprintf("BOARD TYPE  :      %s",board[board_type]);
    gotoxy(15,6);
    cprintf("# OF BIN 1  :      %d",bin_count[PASS]);
    gotoxy(6,8);
    if((board_type==DIP846) || (board_type==I2C46))
    cprintf("     0  0  0  0  1  1  1  1  1  2  2  2  2  2  3  3  3  3  3  4  4  4  4");
    if(board_type==DIPC44)
    cprintf("     0  0  0  0  1  1  1  1  1  2  2  2  2  2  3  3  3  3  3  4  4  4");
    if(board_type==SO8B)
    cprintf("     0  0  0  0  1  1  1  1  1  2  2  2  2  2  3  3  3  3");
    if(board_type==SOA)
    cprintf("     0  0  0  0  1  1  1  1  1  2  2  2  2  2  3  3  3");
    if(board_type==NM95C12DIPA)
    cprintf("     0  0  0  0  1  1  1  1  1  2  2");
    if(board_type==FLATPACK)
    cprintf("     0  0  0  0  1  1  1  1  1  2");
    if((board_type==SO14A) || (board_type==NM95C12SOICA) || (board_type==SO14WMA))
    cprintf("     0  0  0  0  1  1  1  1  1");
    gotoxy(6,9);
    if((board_type==DIP846) || (board_type==I2C46))
    cprintf("     2  4  6  8  0  2  4  6  8  0  2  4  6  8  0  2  4  6  8  0  2  4  6");
    if(board_type==DIPC44)
    cprintf("     2  4  6  8  0  2  4  6  8  0  2  4  6  8  0  2  4  6  8  0  2  4");
    if(board_type==SO8B)
    cprintf("     2  4  6  8  0  2  4  6  8  0  2  4  6  8  0  2  4  6");
    if(board_type==SOA)
    cprintf("     2  4  6  8  0  2  4  6  8  0  2  4  6  8  0  2  4");
    if(board_type==NM95C12DIPA)
    cprintf("     2  4  6  8  0  2  4  6  8  0  2");
    if(board_type==FLATPACK)
    cprintf("     2  4  6  8  0  2  4  6  8  0");
    if((board_type==SO14A) || (board_type==NM95C12SOICA) || (board_type==SO14WMA))
    cprintf("     2  4  6  8  0  2  4  6  8");
    gotoxy(9,10);
    for(i=0; i<(max_col[board_type]/2)*3; i++)
```

```
    cprintf("-");

for(row = 1; row < max_row[board_type]+1; row++)
{
    if(row <= 9)
    {
        gotoxy(5,10+row);
        cprintf("%d",row);
        cprintf("  |");
    }
    else
    {
        gotoxy(5,10+row);
        cprintf("%d",row);
        cprintf(" |");
    }

    for(col = 1; col<=max_col[board_type]; col++)
    {
        k = dut_board[(row-1)*max_col[board_type] + col];
        if(k == 1)  cprintf("_");
        else
        {
            if(k > 1 && k < 10) cprintf("%c",48+k);
            if(k >= 10)         cprintf("%c",55+k);
        }
        if((col%2) == 0)    cprintf(" ");
    }
    cprintf("|");
}

gotoxy(9,11 + max_row[board_type]);
for(i=0; i<(max_col[board_type]/2)*3; i++)
cprintf("-");

gotoxy(5,24);
cprintf("PRESS ANY KEY TO DISPLAY TEST SUMMARY");
getch();
clrscr();

if(board_type == I2C46)
  { gotoxy(5,3);
    cprintf("BIN   2  (I2CONES) ...(U) %d",bin_count[I2CONES]);
    gotoxy(5,4);
    cprintf("BIN   3  (I2CZEROS) ..(T) %d",bin_count[I2CZEROS]);
    gotoxy(5,5);
    cprintf("BIN   4  (I2CCKBD) ...(V) %d",bin_count[I2CCKBD]);
    gotoxy(5,6);
    cprintf("BIN   5  (I2CINICHK) .(_) %d",bin_count[I2CINICHK]);
    gotoxy(5,7);
    cprintf("BIN   6  (I2CINIVER) .(^) %d",bin_count[I2CINIVER]);
    gotoxy(5,8);
    cprintf("BIN  11  (I2CPE1) ....(Y) %d",bin_count[I2CPE1]);
    gotoxy(5,9);
    cprintf("BIN  12  (I2CPE0) ....(Z) %d",bin_count[I2CPE0]);
    gotoxy(5,10);
    cprintf("BIN  16  (I2CBZEROS) .(W) %d",bin_count[I2CBZEROS]);
    gotoxy(5,11);
    cprintf("BIN  17  (I2CBONES) ..(X) %d",bin_count[I2CBONES]);
    gotoxy(5,12);
```

```
    cprintf("BIN  22  (I2CICKBD) ..(I) %d",bin_count[I2CICKBD]);
    gotoxy(5,13);
    cprintf("BIN  26  (SOCKETCHECK)(Q) %d",bin_count[SOCKETCHECK]);
    gotoxy(5,14);
    cprintf("BIN  37  (I2CB55) ....(/) %d",bin_count[I2CB55]);
    gotoxy(5,15);
    cprintf("BIN  38  (I2CBAA) ....(;) %d",bin_count[I2CBAA]);
  }

if(board_type != I2C46)
{
gotoxy(5,3);
cprintf("BIN  2   (ERAL) ......... %d",bin_count[ERAL]);
gotoxy(5,4);
cprintf("BIN  3   (WRAL) ......... %d",bin_count[WRAL]);
gotoxy(5,5);
cprintf("BIN  4   (CKBD) ......... %d",bin_count[CKBD]);
gotoxy(5,6);
cprintf("BIN  5   (INITCHK) ...... %d",bin_count[INITCHK]);
gotoxy(5,7);
cprintf("BIN  6   (INITVER) ...... %d  ...(%3.2f%%)",bin_count[INITVER],percent);

gotoxy(5,8);
cprintf("BIN  7   (INVBITDIAG) ... %d",bin_count[INVBITDIAG]);
gotoxy(5,9);
cprintf("BIN  8   (EVENBYTE) ..... %d",bin_count[EVENBYTE]);
gotoxy(5,10);
cprintf("BIN  9   (ODDBYTE) ...... %d",bin_count[ODDBYTE]);
gotoxy(5,11);
cprintf("BIN 10   (BYTEDIAG) ..(A) %d",bin_count[BYTEDIAG]);
gotoxy(5,12);
cprintf("BIN 11   (PEONES) ....(B) %d",bin_count[PEONES]);
gotoxy(5,13);
cprintf("BIN 12   (PEZEROS) ...(C) %d",bin_count[PEZEROES]);
gotoxy(5,14);
cprintf("BIN 13   (STRESS) ....(D) %d",bin_count[STRESS]);
gotoxy(5,15);
cprintf("BIN 14   (EWDSTEST) ..(E) %d",bin_count[EWDSTEST]);
gotoxy(5,16);
cprintf("BIN 15   (REGWRITE) ..(F) %d",bin_count[REGWRITE]);
gotoxy(5,17);
cprintf("BIN 16   (WRALL0) ....(G) %d",bin_count[WRALL0]);
gotoxy(40,3);
cprintf("BIN 17   (WRALL1) ....(H) %d",bin_count[WRALL1]);
gotoxy(40,4);
cprintf("BIN 20   (REGWRITE0) .(K) %d",bin_count[REGWRITE0]);
gotoxy(40,5);
cprintf("BIN 21   (BITDIAG) ...(L) %d",bin_count[BITDIAG]);
gotoxy(40,9);
cprintf("BIN 22   (INVCKBD) ...(M) %d",bin_count[INVCKBD]);
gotoxy(40,10);
cprintf("BIN 23   (TINVCKBD) ..(N) %d",bin_count[TINVCKBD]);
gotoxy(40,11);
cprintf("BIN 24   (TCKBD) .....(O) %d",bin_count[TCKBD]);
gotoxy(40,12);
cprintf("BIN 25   (PECKBD) ..... %d  (BIN X)",bin_count[PECKBD]);
gotoxy(40,13);
cprintf("BIN 26   (SOCKETCHECK)(Q) %d",bin_count[SOCKETCHECK]);
}
```

```
    gotoxy(10,18);
    cprintf("------------------------------------------------------------");
    gotoxy(10,20);
    cprintf("TOTAL FAILURES ......... %d",bin_count[TOTALBAD]);
    gotoxy(20,24);
    cprintf("PRESS ANY KEY TO EXIT");
    getch();
}

/*--------------------- PRINT FAIL DUT LOCATIONS --------------------------*/
void    print_fail_dut_locations()
{
    char    *message    =   "                                                ";
    int i,n,row,col,k;

    time(&sec_now);
    tm_now = localtime(&sec_now);
    str_now = asctime(tm_now);

    message = "                    FAIL DUT LOCATIONS";
    strprint(message);
    for(i = 0; i<4; i++) biosprint(0,10,0);

    strprint("            PROGRAM REV:      ");
    strcpy(message,rev);
    strprint(message);
    biosprint(0,10,0);

    strprint("            DATE/TIME:        ");
    strprint(str_now);
    biosprint(0,10,0);
    biosprint(0,10,0);

    strprint("            DEVICE TYPE:      ");
    strcpy(message, device);
    strprint(message);
    biosprint(0,10,0);

    strprint("            TEST PROGRAM:     ");
    if(socket_stat == SCON)
    strprint("Socketcheck");
    else
    {
        strprint(opr_test);
        strprint("  (");
        strprint(testprogram);
        strprint(")");
    }
    biosprint(0,10,0);

    strprint("            BOARD TYPE:       ");
    strcpy(message,board[board_type]);
    strprint(message);
    biosprint(0,10,0);

    strprint("            BOARD S/N:        ");
    strcpy(message,tbsn);
    strprint(message);
    biosprint(0,10,0);
```

```
    strprint("          DEVICE STAMPOFF:  ");
    strcpy(message,stampoff);
    strprint(message);
    biosprint(0,10,0);

    strprint("                  SYSTEM #:  ");
    strcpy(message,sysn);
    strprint(message);
    biosprint(0,10,0);

    strprint("          OPERATOR/SHIFT:    ");
    strcpy(message,opr);
    strprint(message);
    biosprint(0,10,0);
    biosprint(0,10,0);

    biosprint(0,10,0);
    biosprint(0,10,0);

    for(row = 0; row < max_row[board_type]; row++)
    {
        biosprint(0,10,0);
        for(col = 1; col <= max_col[board_type];)
        {
            for(k=0; (k<=7)&&(col<=max_col[board_type]); col++,k++)
            {
                n = dut_board[row * max_col[board_type] + col];
                if(n>1)
                {
                    biosprint(0,' ',0);
                    biosprint(0,' ',0);
                    biosprint(0,' ',0);
                    biosprint(0,' ',0);
                    itoa(row+1,message,10);
                    strprint(message);
                    biosprint(0,',',0);
                    biosprint(0,32,0);
                    itoa(col,message,10);
                    strprint(message);
                }
            }
            biosprint(0,10,0);
        }
    }
    biosprint(0,12,0);
}

/* ---------------------- Operator Entry Screen ----------------------------*/
void    ask_for_id()
{
    int i;

    run_stat = TESTMODE;                    /*  Set flag                    */

    window(1,1,80,25);
    clrscr();
    textcolor(YELLOW);
    highvideo();
    gotoxy(1,1);
    cprintf("PRODUCTION TEST PROGRAM");
```

```
gotoxy(73,1);
cprintf("(%s)",rev);
lowvideo();
textcolor(WHITE);
window(1,2,80,25);
clrscr();

gotoxy(20,7);
cprintf("OPERATOR/SHIFT:  ");
gotoxy(20,9);
cprintf("SYSTEM #:        ");
gotoxy(20,11);
cprintf("STAMPOFF:        ");
gotoxy(20,13);
cprintf("AUTOMATIC SUMMARY PRINTOUT [Y/N]:");

textcolor(YELLOW);
highvideo();
gotoxy(1,21);   cprintf("%c",218);
                for(i = 0; i<32; i++) cprintf("%c",196);    cprintf("%c",194);
                for(i = 0; i<45; i++) cprintf("%c",196);    cprintf("%c",191);
gotoxy(1,23);   cprintf("%c",192);
                for(i = 0; i<32; i++) cprintf("%c",196);    cprintf("%c",193);
                for(i = 0; i<45; i++) cprintf("%c",196);    cprintf("%c",217);

gotoxy(1,22);   cprintf("%c  ENGINEERING BITMAP:          B  %c",179,179);
gotoxy(38,22);  cprintf("MAINTENANCE VCC CALIBRATION:       CAL  %c",179);
lowvideo();
textcolor(WHITE);

opr[0]=0;
while(opr[0]==0)
{
    gotoxy(20,7);
    cprintf("OPERATOR/SHIFT:  ");
    gets(opr);
}
if(!stricmp(opr,"CAL")) vcc_cal();                  /*   VCC calibration      */
if(!stricmp(opr,"B")) bitmap();                     /*   Bitmap routine       */

if(run_stat == TESTMODE)
{
    sysn[0]=0;
    while(sysn[0]==0)
    {
        gotoxy(1,21); delline(); delline(); delline();
        gotoxy(20,9);
        cprintf("SYSTEM #:        ");
        gets(sysn);
    }
    gotoxy(20,11);
    cprintf("STAMPOFF:        ");
    gets(stampoff);

    gotoxy(20,13);
    cprintf("AUTOMATIC SUMMARY PRINTOUT [Y/N]: ");
    gotoxy(20,14);
    cprintf("Default is YES if <RETURN or ENTER>   ");
    textcolor(YELLOW);
    gotoxy(20,24);
```

```
        cprintf("TYPE [ENTER] TO CONTINUE");
        textcolor(WHITE);
        gotoxy(60,13);
        gets(input);
        if(!stricmp(input,"N"))                    /*     APOFF = Automatic Summary OFF        */
            print_stat = APOFF;
    }
}

/*************************************************************************/

/*              m2.c                */
#include    <stdio.h>
#include    <dos.h>
#include    <conio.h>
#include    <dir.h>
#include    <math.h>
#include    <io.h>
#include    <sys/stat.h>
#include    <time.h>
#include    <string.h>
#include    <\mill\variable.h>
#include    <\mill\function.h>

/*************************************************************************/
/* ----------------------- Select NMOS or CMOS ---------------------------*/
int select_mos()
{
    int     mos;

    mos=0;
    ch=0;

    clrscr();
    gotoxy(25,5);
    cprintf("TYPE  [0] ..... NMOS");
    gotoxy(25,7);
    cprintf("TYPE  [1] ..... CMOS");
    gotoxy(25,9);
    cprintf("TYPE  [2] ..... I2C");
    gotoxy(25,12);
    cprintf("ENTER SELECTION HERE: ");

    while((ch!='0') && (ch!='1') && (ch!='2'))
    {
        ch=getch();
        if (ch=='0')
        {
            gotoxy(25,5);
            mos=0;
            cprintf("TYPE  [0] ..... NMOS");
        }

        if (ch=='1')
        {
            gotoxy(25,7);
            mos=1;
            cprintf("TYPE  [1] ..... CMOS");
        }
        if (ch=='2')
```

```
        {
            gotoxy(25,9);
            mos=2;
            cprintf("TYPE  [2] ..... I2C");
        }
    }
    return(mos);
}

/* ---------------------- Select NMOS Devices ----------------------------*/
void    select_nmos_device()
{
    ch=0;

    clrscr();
    while((ch<'0') || (ch>'4'))
    {
        gotoxy(25,5);
        cprintf("TYPE  [0] ..... NMC9306");
        gotoxy(25,6);
        cprintf("TYPE  [1] ..... NMC9307");
        gotoxy(25,7);
        cprintf("TYPE  [2] ..... NMC9313");

        gotoxy(25,9);
        cprintf("TYPE  [3] ..... NMC9346");
        gotoxy(25,10);
        cprintf("TYPE  [4] ..... NMC9314");
        ch=0;
        gotoxy(25,13);
        cprintf("ENTER SELECTION HERE:  ");
        ch=getch();
    }

    if (ch == '0')
        {
        strcpy(device,"NMC9306");
        strcpy(p1kt,"9306.1KT");
        strcpy(p5kt,"9306.5KT");
        strcpy(p1k,"9306.1K");
        strcpy(p10k,"9306.10K");
        strcpy(p50k,"9306.50K");
        strcpy(pver,"9306.VER");
        dut_type=NMC9306;
        true_dut=NMC9306;
        strcpy(dut_test,"9306");
        twp=20;
        dut_reg=16;
        dut_opbits=2;
        dut_adrbits=6;
        dut_offset=2;                   /*  Shifts required to move reg into DUT    */
        get_op_code();
        }

    if (ch == '1')
        {
        strcpy(device,"NMC9307");
        strcpy(p1kt,"9307.1KT");
        strcpy(p5kt,"9307.5KT");
        strcpy(p1k,"9307.1K");
```

```
        strcpy(p10k,"9307.10K");
        strcpy(p50k,"9307.50K");
        strcpy(pver,"9307.VER");
        dut_type=NMC9307;
        true_dut=NMC9306;
        strcpy(dut_test,"9306");
        twp=20;
        dut_reg=16;
        dut_adrbits=6;
        dut_opbits=2;
        dut_offset=2;                   /*  Shifts required to move reg into DUT    */
        get_op_code();
        }

    if (ch == '2')
        {
        strcpy(device,"NMC9313");
        strcpy(p1kt,"9313.1KT");
        strcpy(p5kt,"9313.5KT");
        strcpy(p1k,"9313.1K");
        strcpy(p10k,"9313.10K");
        strcpy(p50k,"9313.50K");
        strcpy(pver,"9313.VER");
        dut_type=NMC9313;
        true_dut=NMC9306;
        strcpy(dut_test,"9306");
        twp=20;
        dut_reg=16;
        dut_opbits=2;
        dut_adrbits=6;
        dut_offset=2;                   /*  Shifts required to move reg into DUT    */
        get_op_code();
        }

    if (ch == '3')
        {
        strcpy(device,"NMC9346");
        strcpy(p1kt,"9346.1KT");
        strcpy(p5kt,"9346.5KT");
        strcpy(p1k,"9346.1K");
        strcpy(p10k,"9346.10K");
        strcpy(p50k,"9346.50K");
        strcpy(pver,"9346.VER");
        dut_type=NMC9346;
        true_dut=NMC9346;
        strcpy(dut_test,"9346");
        twp=20;
        dut_reg=64;
        dut_opbits=2;
        dut_adrbits=6;
        dut_offset=2;                   /*  Shifts required to move reg into DUT    */
        get_op_code();
        }

    if (ch == '4')
        {
        strcpy(device,"NMC9314");
        strcpy(p1kt,"9314.1KT");
        strcpy(p5kt,"9314.5KT");
        strcpy(p1k,"9314.1K");
```

```
        strcpy(p10k,"9314.10K");
        strcpy(p50k,"9314.50K");
        strcpy(pver,"9314.VER");
        dut_type=NMC9314;
        true_dut=NMC9346;
        strcpy(dut_test,"9346");
        twp=30;
        dut_reg=64;
        dut_opbits=2;
        dut_adrbits=6;
        dut_offset=2;                   /*  Shifts required to move reg into DUT    */
        get_op_code();
        }
}

/*------------------------- SELECT CMOS DEVICES ----------------------------*/
void select_cmos_device()
{
    ch=0;

    clrscr();

    while ( ((ch<'a') || (ch>'u')) && ((ch<'A') || (ch>'U')) )
    {
        gotoxy(5,2);
        cprintf("TYPE  [A] ..... NM  93C06");
        gotoxy(5,3);
        cprintf("TYPE  [B] ..... NMC 93C06");
        gotoxy(5,4);
        cprintf("TYPE  [C] ..... NM  93CS06");
        gotoxy(5,6);
        cprintf("TYPE  [D] ..... NM  93C07");
        gotoxy(5,8);
        cprintf("TYPE  [E] ..... NM  93C46");
        gotoxy(5,9);
        cprintf("TYPE  [F] ..... NMC 93C46");
        gotoxy(5,10);
        cprintf("TYPE  [G] ..... NM  93CS46");
        gotoxy(5,12);
        cprintf("TYPE  [H] ..... NM  93C56");
        gotoxy(5,13);
        cprintf("TYPE  [I] ..... NMC 93C56");
        gotoxy(5,14);
        cprintf("TYPE  [J] ..... NM  93CS56");
        gotoxy(5,16);
        cprintf("TYPE  [K] ..... NM  93C66");
        gotoxy(5,17);
        cprintf("TYPE  [L] ..... NMC 93C66");
        gotoxy(5,18);
        cprintf("TYPE  [M] ..... NM  93CS66");
        gotoxy(40,2);
        cprintf("TYPE  [N] ..... NM  88CS06");
        gotoxy(40,3);
        cprintf("TYPE  [P] ..... NM  88CS46");
        gotoxy(40,5);
        cprintf("TYPE  [Q] ..... NM  93C46A");
        gotoxy(40,6);
        cprintf("TYPE  [R] ..... NM  59C11");
        gotoxy(40,8);
        cprintf("TYPE  [S] ..... NM  88C16");
```

```
        gotoxy(40,9);
        cprintf("TYPE  [T] ..... NM  59C16");
        gotoxy(40,11);
        cprintf("TYPE  [U] ..... NM  95C12");
        gotoxy(40,13);
        cprintf("TYPE  [V] ..... NM  93C13N");
        gotoxy(40,15);
        cprintf("TYPE  [W] ..... NM  93C14N");
        gotoxy(25,24);
        cprintf("ENTER SELECTION HERE:   ");
        ch=getch();
}

if ((ch == 'A') || (ch == 'a'))
{
        strcpy(device,"NM 93C06");
        strcpy(p1kt,"T93C06.1KT");
        strcpy(p5kt,"T93C06.5KT");
        strcpy(p1k,"T93C06.1K");
        strcpy(p10k,"T93C06.10K");
        strcpy(p50k,"T93C06.50K");
        strcpy(pver,"T93C06.VER");
        dut_type=NM93C06;
        true_dut=CTRUE;
        strcpy(dut_test,"CTRUE");
        twp=20;
        dut_reg=16;
        dut_opbits=2;
        dut_adrbits=6;
        dut_offset=2;                   /*  Shifts required to move reg into DUT    */
        get_op_code();
}
if ((ch == 'B') || (ch == 'b'))
{
        strcpy(device,"NMC 93C06");
        strcpy(p1kt,"93C06.1KT");
        strcpy(p5kt,"93C06.5KT");
        strcpy(p1k,"93C06.1K");
        strcpy(p10k,"93C06.10K");
        strcpy(p50k,"93C06.50K");
        strcpy(pver,"93C06.VER");
        dut_type=NMC93C06;
        true_dut=CSTRUE;
        strcpy(dut_test,"CSTRUE");
        twp=20;
        dut_reg=16;
        dut_opbits=2;
        dut_adrbits=6;
        dut_offset=2;                   /*  Shifts required to move reg into DUT    */
        get_op_code();
}

if ((ch == 'C') || (ch == 'c'))
{
        strcpy(device,"NM 93CS06");
        strcpy(p1kt,"93CS06.1KT");
        strcpy(p5kt,"93CS06.5KT");
        strcpy(p1k,"93CS06.1K");
        strcpy(p10k,"93CS06.10K");
        strcpy(p50k,"93CS06.50K");
```

```
        strcpy(pver,"93CS06.VER");
        dut_type=NM93CS06;
        true_dut=CSTRUE;
        strcpy(dut_test,"CSTRUE");
        twp=20;
        dut_reg=16;
        dut_opbits=2;
        dut_adrbits=6;
        dut_offset=2;                   /*  Shifts required to move reg into DUT    */
        get_op_code();
}

if ((ch == 'D') || (ch == 'd'))
{
        strcpy(device,"NM 93C07");
        strcpy(p1kt,"T93C07.1KT");
        strcpy(p5kt,"T93C07.5KT");
        strcpy(p1k,"T93C07.1K");
        strcpy(p10k,"T93C07.10K");
        strcpy(p50k,"T93C07.50K");
        strcpy(pver,"T93C07.VER");
        dut_type=NM93C06;
        true_dut=CTRUE;
        strcpy(dut_test,"CTRUE");
        twp=20;
        dut_reg=16;
        dut_opbits=2;
        dut_adrbits=6;
        dut_offset=2;                   /*  Shifts required to move reg into DUT    */
        get_op_code();
}
if ((ch == 'E') || (ch == 'e'))
{
        strcpy(device,"NM 93C46");
        strcpy(p1kt,"T93C46.1KT");
        strcpy(p5kt,"T93C46.5KT");
        strcpy(p1k,"T93C46.1K");
        strcpy(p10k,"T93C46.10K");
        strcpy(p50k,"T93C46.50K");
        strcpy(pver,"T93C46.VER");
        dut_type=NM93C46;
        true_dut=CTRUE;
        strcpy(dut_test,"CTRUE");
        twp=20;
        dut_reg=64;
        dut_opbits=2;
        dut_adrbits=6;
        dut_offset=2;                   /*  Shifts required to move reg into DUT    */
        get_op_code();
}
if ((ch == 'F') || (ch == 'f'))
{
        strcpy(device,"NMC 93C46");
        strcpy(p1kt,"93C46.1KT");
        strcpy(p5kt,"93C46.5KT");
        strcpy(p1k,"93C46.1K");
        strcpy(p10k,"93C46.10K");
        strcpy(p50k,"93C46.50K");
        strcpy(pver,"93C46.VER");
        dut_type=NMC93C46;
```

```
        true_dut=CSTRUE;
        strcpy(dut_test,"CSTRUE");
        twp=20;
        dut_reg=64;
        dut_opbits=2;
        dut_adrbits=6;
        dut_offset=2;                   /*  Shifts required to move reg into DUT    */
        get_op_code();
}
if ((ch == 'G') || (ch == 'g'))
{
        strcpy(device,"NM 93CS46");
        strcpy(p1kt,"93CS46.1KT");
        strcpy(p5kt,"93CS46.5KT");
        strcpy(p1k,"93CS46.1K");
        strcpy(p10k,"93CS46.10K");
        strcpy(p50k,"93CS46.50K");
        strcpy(pver,"93CS46.VER");
        dut_type=NM93CS46;
        true_dut=CSTRUE;
        strcpy(dut_test,"CSTRUE");
        twp=20;
        dut_reg=64;
        dut_adrbits=6;
        dut_opbits=2;
        dut_offset=2;                   /*  Shifts required to move reg into DUT    */
        get_op_code();
}
if ((ch == 'H') || (ch == 'h'))
{
        strcpy(device,"NM 93C56");
        strcpy(p1kt,"T93C56.1KT");
        strcpy(p5kt,"T93C56.5KT");
        strcpy(p1k,"T93C56.1K");
        strcpy(p10k,"T93C56.10K");
        strcpy(p50k,"T93C56.50K");
        strcpy(pver,"T93C56.VER");
        dut_type=NM93C56;
        true_dut=CTRUE;
        strcpy(dut_test,"CTRUE");
        twp=20;
        dut_reg=128;
        dut_opbits=2;
        dut_adrbits=8;
        dut_offset=0;                   /*  Shifts required to move reg into DUT    */
        get_op_code();
}
if ((ch == 'I') || (ch == 'i'))
{
        strcpy(device,"NMC 93C56");
        strcpy(p1kt,"93C56.1KT");
        strcpy(p5kt,"93C56.5KT");
        strcpy(p1k,"93C56.1K");
        strcpy(p10k,"93C56.10K");
        strcpy(p50k,"93C56.50K");
        strcpy(pver,"93C56.VER");
        dut_type=NMC93C56;
        true_dut=CSTRUE;
        strcpy(dut_test,"CSTRUE");
        twp=20;
```

```
        dut_reg=128;
        dut_opbits=2;
        dut_adrbits=8;
        dut_offset=0;                   /*  Shifts required to move reg into DUT    */
        get_op_code();
}
if ((ch == 'J') || (ch == 'j'))
{
        strcpy(device,"NM 93CS56");
        strcpy(p1kt,"93CS56.1KT");
        strcpy(p5kt,"93CS56.5KT");
        strcpy(p1k,"93CS56.1K");
        strcpy(p10k,"93CS56.10K");
        strcpy(p50k,"93CS56.50K");
        strcpy(pver,"93CS56.VER");
        dut_type=NM93CS56;
        true_dut=CSTRUE;
        strcpy(dut_test,"CSTRUE");
        twp=20;
        dut_reg=128;
        dut_opbits=2;
        dut_adrbits=8;
        dut_offset=0;                   /*  Shifts required to move reg into DUT    */
        get_op_code();
}
if ((ch == 'K') || (ch == 'k'))
{
        strcpy(device,"NM 93C66");
        strcpy(p1kt,"T93C66.1KT");
        strcpy(p5kt,"T93C66.5KT");
        strcpy(p1k,"T93C66.1K");
        strcpy(p10k,"T93C66.10K");
        strcpy(p50k,"T93C66.50K");
        strcpy(pver,"T93C66.VER");
        dut_type=NM93C66;
        true_dut=CTRUE;
        strcpy(dut_test,"CTRUE");
        twp=20;
        dut_reg=256;
        dut_opbits=2;
        dut_adrbits=8;
        dut_offset=0;                   /*  Shifts required to move reg into DUT    */
        get_op_code();
}
if ((ch == 'L') || (ch == 'l'))
{
        strcpy(device,"NMC 93C66");
        strcpy(p1kt,"93C66.1KT");
        strcpy(p5kt,"93C66.5KT");
        strcpy(p1k,"93C66.1K");
        strcpy(p10k,"93C66.10K");
        strcpy(p50k,"93C66.50K");
        strcpy(pver,"93C66.VER");
        dut_type=NMC93C66;
        strcpy(dut_test,"CSTRUE");
        true_dut=CSTRUE;
        twp=20;
        dut_reg=256;
        dut_adrbits=8;
        dut_opbits=2;
```

```
        dut_offset=0;                   /*  Shifts required to move reg into DUT    */
        get_op_code();
}
if ((ch == 'M') || (ch == 'm'))
{
        strcpy(device,"NM 93CS66");
        strcpy(p1kt,"93CS66.1KT");
        strcpy(p5kt,"93CS66.5KT");
        strcpy(p1k,"93CS66.1K");
        strcpy(p10k,"93CS66.10K");
        strcpy(p50k,"93CS66.50K");
        strcpy(pver,"93CS66.VER");
        dut_type=NM93CS66;
        true_dut=CSTRUE;
        strcpy(dut_test,"CSTRUE");
        twp=20;
        dut_reg=256;
        dut_opbits=2;
        dut_adrbits=8;
        dut_offset=0;                   /*  Shifts required to move reg into DUT    */
        get_op_code();
}
if ((ch == 'N') || (ch == 'n'))
{
        strcpy(device,"NM 88CS06");
        strcpy(p1kt,"88CS06.1KT");
        strcpy(p5kt,"88CS06.5KT");
        strcpy(p1k,"88CS06.1K");
        strcpy(p10k,"88CS06.10K");
        strcpy(p50k,"88CS06.50K");
        strcpy(pver,"88CS06.VER");
        dut_type=NM88CS06;
        true_dut=CSTRUE;
        strcpy(dut_test,"CSTRUE");
        twp=20;
        dut_reg=16;
        dut_opbits=2;
        dut_adrbits=6;
        dut_offset=2;                   /*  Shifts required to move reg into DUT    */
        get_op_code();
}
if ((ch == 'P') || (ch == 'p'))
{
        strcpy(device,"NM 88CS46");
        strcpy(p1kt,"88CS46.1KT");
        strcpy(p5kt,"88CS46.5KT");
        strcpy(p1k,"88CS46.1K");
        strcpy(p10k,"88CS46.10K");
        strcpy(p50k,"88CS46.50K");
        strcpy(pver,"88CS46.VER");
        dut_type=NM88CS46;
        true_dut=CSTRUE;
        strcpy(dut_test,"CSTRUE");
        twp=20;
        dut_reg=64;
        dut_opbits=2;
        dut_adrbits=6;
        dut_offset=2;                   /*  Shifts required to move reg into DUT    */
        get_op_code();
}
```

```
if ((ch == 'Q') || (ch == 'q'))
{
        strcpy(device,"NM 93C46A");
        strcpy(p1kt,"93C46A.1KT");
        strcpy(p5kt,"93C46A.5KT");
        strcpy(p1k,"93C46A.1K");
        strcpy(p10k,"93C46A.10K");
        strcpy(p50k,"93C46A.50K");
        strcpy(pver,"93C46A.VER");
        dut_type=NM93C46A;
        true_dut=CTRUE;
        strcpy(dut_test,"CTRUE");
        tmp=20;
        dut_reg=64;
        dut_opbits=2;
        dut_adrbits=6;
        dut_offset=2;                   /*  Shifts required to move reg into DUT    */
        get_op_code();
}
if ((ch == 'R') || (ch == 'r'))
{
        strcpy(device,"NM 59C11");
        strcpy(p1kt,"59C11.1KT");
        strcpy(p5kt,"59C11.5KT");
        strcpy(p1k,"59C11.1K");
        strcpy(p10k,"59C11.10K");
        strcpy(p50k,"59C11.50K");
        strcpy(pver,"59C11.VER");
        dut_type=NM59C11;
        true_dut=CTRUE;
        strcpy(dut_test,"59C11");
        tmp=20;
        dut_reg=64;
        dut_opbits=4;
        dut_adrbits=6;
        dut_offset=2;                   /*  Shifts required to move reg into DUT    */
        get_op_code();
}
if ((ch == 'S') || (ch == 's'))
{
        strcpy(device,"NM 88C16");
        strcpy(p1kt,"88C16.1KT");
        strcpy(p5kt,"88C16.5KT");
        strcpy(p1k,"38C16.1K");
        strcpy(p10k,"88C16.10K");
        strcpy(p50k,"88C16.50K");
        strcpy(pver,"88C16.VER");
        dut_type=NM89C16;
        true_dut=CTRUE;
        strcpy(dut_test,"CTRUE");
        tmp=20;
        dut_reg=1024;
        dut_opbits=2;
        dut_adrbits=10;
        dut_offset=6;                   /*  Shifts required to move reg into DUT    */
        get_op_code();
}
if ((ch == 'T') || (ch == 't'))
{
        strcpy(device,"NM 59C16");
```

```
        strcpy(p1kt,"59C16.1KT");
        strcpy(p5kt,"59C16.5KT");
        strcpy(p1k,"59C16.1K");
        strcpy(p10k,"59C16.10K");
        strcpy(p50k,"59C16.50K");
        strcpy(pver,"59C16.VER");
        dut_type=NM59C16;
        true_dut=CTRUE;
        strcpy(dut_test,"59C11");
        twp=20;
        dut_reg=1024;
        dut_opbits=4;
        dut_adrbits=10;
        dut_offset=0;                   /*  Shifts required to move reg into DUT    */
        get_op_code();
}
if ((ch == 'U') || (ch == 'u'))
{
        strcpy(device,"NM 95C12");
        strcpy(p1kt,"95C12.1KT");
        strcpy(p5kt,"95C12.5KT");
        strcpy(p1k,"95C12.1K");
        strcpy(p10k,"95C12.10K");
        strcpy(p50k,"95C12.50K");
        strcpy(pver,"95C12.VER");
        dut_type=NM95C12;
        true_dut=CSTRUE;
        strcpy(dut_test,"CSTRUE");
        twp=20;
        dut_reg=62;                     /*  Due to switch control in Reg63/64       */
        dut_opbits=2;
        dut_adrbits=6;
        dut_offset=2;                   /*  Shifts required to move reg into DUT    */
        get_op_code();
}
if ((ch == 'V') || (ch == 'v'))
{
        strcpy(device,"NM 93C13N");
        strcpy(p1kt,"93C13N.1KT");
        strcpy(p5kt,"93C13N.5KT");
        strcpy(p1k,"93C13N.1K");
        strcpy(p10k,"93C13N.10K");
        strcpy(p50k,"93C13N.50K");
        strcpy(pver,"93C13N.VER");
        dut_type=NM93C13N;
        true_dut=CTRUE;
        strcpy(dut_test,"C13C14N");
        twp=20;
        dut_reg=16;
        dut_opbits=2;
        dut_adrbits=6;
        dut_offset=2;
        get_op_code();
}
        if ((ch == 'W') || (ch == 'w'))
{
        strcpy(device,"NM 93C14N");
        strcpy(p1kt,"93C14N.1KT");
        strcpy(p5kt,"93C14N.5KT");
        strcpy(p1k,"93C14N.1K");
```

```
                strcpy(p10k,"93C14N.10K");
                strcpy(p50k,"93C14N.50K");
                strcpy(pver,"93C14N.VER");
                dut_type=NM93C14N;
                true_dut=CTRUE;
                strcpy(dut_test,"C13C14N");
                twp=20;
                dut_reg=64;
                dut_opbits=2;
                dut_adrbits=6;
                dut_offset=2;
                get_op_code();
        }

}

void    get_op_code()
{
    if(dut_type == NM88CS06 || dut_type == NM88CS46)
    {
        enable_send  = 0x70;
        disable_send = 0x40;
        wral_send    = 0x50;
        wrall_send   = 0x50;
        write_send   = 0x00;
        read_send    = 0xC0;
    }
    else
    {
        enable_send  = 0x30;
        disable_send = 0x00;
        eral_send    = 0x20;
        wral_send    = 0x10;
        wrall_send   = 0x10;
        erase_send   = 0xC0;
        write_send   = 0x40;
        read_send    = 0x80;
    }
}

void    select_i2c_device()
{
  ch=0;
  clrscr();
  while((ch < '0') || (ch > '3'))
       {
        gotoxy(25,5);
        cprintf("TYPE  [0] ..... NM24C02");
        gotoxy(25,7);
        cprintf("TYPE  [1] ..... NM24C04");
        gotoxy(25,9);
        cprintf("TYPE  [2] ..... NM24C08");
        gotoxy(25,11);
        cprintf("TYPE  [3] ..... NM24C16");
        ch=0;
        gotoxy(25,15);
        cprintf("ENTER SELECTION HERE:  ");
        ch=getch();
       }
   if (ch=='0')
```

```
        {
         strcpy(device,"NM24C02");
         strcpy(p1kt,"24C02.1KT");
         strcpy(p5kt,"24C02.5KT");
         strcpy(p1k,"24C02.1K");
         strcpy(p10k,"24C02.10K");
         strcpy(p50k,"24C02.50K");
         strcpy(pver,"24C02.VER");
         strcpy(dut_test,"24C02");
         dut_type=NM24C02;
         twp=20;
         dut_reg=256;
        }

    if (ch=='1')
        {
         strcpy(device,"NM24C04");
         strcpy(p1kt,"24C04.1KT");
         strcpy(p5kt,"24C04.5KT");
         strcpy(p1k,"24C04.1K");
         strcpy(p10k,"24C04.10K");
         strcpy(p50k,"24C04.50K");
         strcpy(pver,"24C04.VER");
         strcpy(dut_test,"24C04");
         dut_type=NM24C04;
         twp=20;
         dut_reg=512;
        }

    if (ch=='2')
        {
         strcpy(device,"NM24C08");
         strcpy(p1kt,"24C08.1KT");
         strcpy(p5kt,"24C08.5KT");
         strcpy(p1k,"24C08.1K");
         strcpy(p10k,"24C08.10K");
         strcpy(p50k,"24C08.50K");
         strcpy(pver,"24C08.VER");
         strcpy(dut_test,"24C08");
         dut_type=NM24C08;
         twp=20;
         dut_reg=1024;
        }
    if (ch=='3')
        {
         strcpy(device,"NM24C16");
         strcpy(p1kt,"24C16.1KT");
         strcpy(p5kt,"24C16.5KT");
         strcpy(p1k,"24C16.1K");
         strcpy(p10k,"24C16.10K");
         strcpy(p50k,"24C16.50K");
         strcpy(pver,"24C16.VER");
         strcpy(dut_test,"24C16");
         dut_type=NM24C16;
         twp=20;
         dut_reg=2048;
        }
}

/*------------------------ OPERATOR SELECTS CORRECT TEST PLAN IN TESTS4 ------------------*/
```

```
void select_test_program()
{
    struct    ffblk    test;
    int       i;

    ch=0;
    clrscr();
    testprogram[0]=0;                                   /*  Set testprogram = 0

    gotoxy(25,3);
    cprintf("TYPE  [0] ..... %s",p1kt);
    gotoxy(25,5);
    cprintf("TYPE  [1] ..... %s",p5kt);
    gotoxy(25,7);
    cprintf("TYPE  [2] ..... %s",p1k);
    gotoxy(25,9);
    cprintf("TYPE  [3] ..... %s",p10k);
    gotoxy(25,11);
    cprintf("TYPE  [4] ..... %s",p50k);
    gotoxy(25,13);
    cprintf("TYPE  [5] ..... %s",pver);
    gotoxy(25,15);
    cprintf("TYPE  [E] ..... TO ENTER PROGRAM NAME");

    gotoxy(25,20);
    cprintf("ENTER SELECTION HERE:      ");
    while(((ch<'0') || (ch>'5')) && (ch!=ESC) && (ch!='E') && (ch!='e'))
    {
        ch=getch();
        if ((ch>='0') && (ch<='4')) cprintf("%c",ch);

        if (ch=='0')
        {
            gotoxy(25,3);
            cprintf("TYPE  [0] ..... %s",p1kt);
                    strcpy(opr_test,p1kt);
            strcpy(testprogram,dut_test);
            strcat(testprogram,"*.1KT");
            if(!findfirst(testprogram,&test,0)) strcpy(testprogram,test.ff_name);
        }
        if (ch=='1')
        {
            gotoxy(25,5);
            cprintf("TYPE  [1] ..... %s",p5kt);
            strcpy(opr_test,p5kt);
            strcpy(testprogram,dut_test);
            strcat(testprogram,"*.5KT");
            if(!findfirst(testprogram,&test,0)) strcpy(testprogram,test.ff_name);
        }
        if (ch=='2')
        {
            gotoxy(25,7);
            cprintf("TYPE  [2] ..... %s",p1k);
            strcpy(opr_test,p1k);
            strcpy(testprogram,dut_test);
            strcat(testprogram,"*.1K");
            if(!findfirst(testprogram,&test,0)) strcpy(testprogram,test.ff_name);
        }
        if (ch=='3')
        {
```

```
            gotoxy(25,9);
            cprintf("TYPE  [3] ..... %s",p10k);
            strcpy(opr_test,p10k);
            strcpy(testprogram,dut_test);
            strcat(testprogram,"*.I0K");
            if(!findfirst(testprogram,&test,0)) strcpy(testprogram,test.ff_name);
        }
        if (ch=='4')
        {
            gotoxy(25,11);
            cprintf("TYPE  [4] ..... %s",p50k);
            strcpy(opr_test,p50k);
            strcpy(testprogram,dut_test);
            strcat(testprogram,"*.50K");
            if(!findfirst(testprogram,&test,0)) strcpy(testprogram,test.ff_name);
        }
        if (ch=='5')
        {
            gotoxy(25,13);
            cprintf("TYPE  [5] ..... %s",pver);
            strcpy(opr_test,pver);
            strcpy(testprogram,dut_test);
            strcat(testprogram,"*.VER");
            if(!findfirst(testprogram,&test,0)) strcpy(testprogram,test.ff_name);
        }
        if ((ch=='E')||(ch=='e'))
        {
            gotoxy(25,20);
            cprintf("                              ");
            gotoxy(25,15);
            cprintf("                                    ");
            gotoxy(25,15);
            cprintf("ENTER PROGRAM NAME:   ");
            gets(opr_test);
            strcpy(testprogram,opr_test);
        }
        if(findfirst(testprogram,&test,FA_RDONLY)) testprogram[0]=0;
    }

    if (ch==ESC)                        /* ESC pressed; check password before exit */
    {
        gotoxy(1,1);
        cprintf("Enter password: ");
        i=0;
        while((password[i]=getch())!=CR) i++;
        password[i]='\0';
        if (!stricmp(password,"junk1"))
        {
            normvideo();
            window(1,1,80,25);
            clrscr();
            chdir("c:/sbt");
            exit(2);
        }
    }
}

/*---------------------- OPEN SELECTED TEST PLAN AND READ TESTS ----------------------*/
```

```
int read_test_program()
{
    int i;

    infile = fopen(testprogram,"rt");

    i = 0;
    while(fscanf(infile,"%d %d %s",&t[i].bin,&t[i].volt,t[i].name) != EOF)
    {
        test_set(i);
        i++;
    }
    fclose(infile);
    return(i);
}


/*------------------------ OPERATOR SELECTS DUT BOARD TYPE ------------------------*/
void select_board_type()
{
    int     i;

    ch=0;

    clrscr();
    gotoxy(20,2);
    cprintf("%c-----------------------------------------%c",218,191);
    gotoxy(20,3);
    cprintf("|          SELECT BOARD TYPE              |");
    gotoxy(20,4);
    cprintf("%c-----------------------------------------%c",192,217);

    gotoxy(20,7);
    cprintf("       BOARD SELECTION                ENTER");
    gotoxy(20,8);
    cprintf("-----------------------------------------");
    for(i=0;i<MAX_BOARDS;i++)
    {
        gotoxy(20,10 + i);
        cprintf("%s                %s",board[i],boarden[i]);
    }

    while((ch<'0') || (ch>'9'))
    {
        ch=getch();
        if (ch == '0') board_type = DIPC44;
        if (ch == '1') board_type = DIP346;
        if (ch == '2') board_type = SOA;
        if (ch == '3') board_type = SO8B;
        if (ch == '4') board_type = SO14A;
        if (ch == '5') board_type = NM95C12DIPA;
        if (ch == '6') board_type = NM95C12SOICA;
        if (ch == '7') board_type = SO14WMA;
        if (ch == '8') board_type = FLATPACK;
        if (ch == '9') board_type = I2C46;
    }
}

/*------------------------ OPERATOR VERIFIES ALL SELECTIONS ----------------------*/
int verify_selections()
```

```
{
    clrscr();

    if (count==0)                          /*  Bad test program selection  */
    {
        textcolor(YELLOW);
        highvideo();
        gotoxy(20,5);
        cprintf("%c-------------------------------------%c",218,191);
        gotoxy(20,6);
        cprintf("| ..... PROGRAM SELECTION ERROR .... |");
        gotoxy(20,7);
        cprintf("|                                     |");
        gotoxy(20,8);
        cprintf("| ... HIT ANY KEY TO START AGAIN ... |");
        gotoxy(20,9);
        cprintf("%c-------------------------------------%c",192,217);

        lowvideo();
        textcolor(WHITE);

        gotoxy(20,11);
        cprintf("PROGRAM %s NOT ALLOWED",opr_test);
        getch();
        return(0);
    }

    gotoxy(20,5);
    cprintf("DEVICE TYPE:      %s",device);
    gotoxy(20,7);
    textcolor(YELLOW);
    cprintf("TEST PROGRAM:     %s",opr_test);
    textcolor(WHITE);
    gotoxy(20,9);
    cprintf("LOAD BOARD:       %s",board[board_type]);
    gotoxy(20,11);
    cprintf("OPERATOR/SHIFT:   %s",opr);
    gotoxy(20,13);
    cprintf("SYSTEM #          %s",sysn);
    gotoxy(20,15);
    cprintf("STAMPOFF:         %s",stampoff);
    gotoxy(20,17);
    cprintf("AUTOSUMMARY:");
    gotoxy(38,17);
    if(print_stat == APON) cprintf("ON");
    if(print_stat != APON) cprintf("OFF");

    gotoxy(10,24);
    textcolor(YELLOW);
    cprintf("TYPE [X] TO CONTINUE ... OR PRESS [ENTER] TO START OVER");
    textcolor(WHITE);
    ch=getch();
    if ((ch=='X') || (ch=='x')) return(1);
        else return(0);
}

/* ------------------------------------------------------------------- */
/* screen for selecting DUT test board# and to start test or socket check */
/* ------------------------------------------------------------------- */
int enter_dut_board_no()
```

```
{
    int    option;

    ch=0;
    while((ch!='X') && (ch!='x'))
    {
        clrscr();
        textcolor(YELLOW);
        gotoxy(20,2);
        cprintf("DEVICE:        %s",device);
        gotoxy(20,3);
        cprintf("TEST PROGRAM: %s  (%s)",opr_test,testprogram);
        textcolor(WHITE);

        input[0] = '\0';
        socket_stat = SCOFF;
        gotoxy(20,5);
        cprintf("TEST BOARD S/N:     ");
        gets(tbsn);

        gotoxy(20,10);
        cprintf("TYPE  [0]       SOCKET CHECK");
        gotoxy(20,12);
        cprintf("TYPE  [1]       START TESTING");

        gotoxy(20,15);
        cprintf("ENTER SELECTION HERE: ");
        ch=0;
        while((ch!='0') && (ch!='1'))
        {
            ch=getch();
            if ((ch=='0') || (ch=='1'))
                printf("%c",ch);
            textcolor(YELLOW);
            highvideo();
            if (ch=='0')
            {
                gotoxy(20,10);
                option=SC;
                cprintf("TYPE  [0]       SOCKET CHECK");
            }
            if (ch=='1')
            {
                gotoxy(20,12);
                option=TEST;
                cprintf("TYPE  [1]       START TESTING");
            }
        }
        textcolor(YELLOW);
        gotoxy(10,24);
        cprintf("TYPE [X] TO CONTINUE ... OR PRESS [ENTER] TO START OVER");
        ch=getch();
        textcolor(WHITE);
    }
    return(option);
}

void   program(int new_test)
{
    register    int        reg;            /*  Register variables for speed        */
```

```
    register  int         col;
    int block;
    long int tt;

    get_vector(new_test);

if (board_type==I2C46)
   { i2c_mode;
     column(0);
     di_dec(0);

     if (new_test==I2CZEROS)
        { for (reg=0; reg<256; reg++)
             { startc;
               data_write(0xa0);
               ack_write;
               data_write(reg);
               ack_write;
               data_write(0);
               ack_write;
               stop;
               delay(20);
             }
         if(dut_reg==512)
           { for (reg=0; reg<256; reg++)
                { startc;
                  data_write(0xa2);
                  ack_write;
                  data_write(reg);
                  ack_write;
                  data_write(0);
                  ack_write;
                  stop;
                  delay(20);
                }
           }
        }

     if (new_test==I2CONES)
        { for (reg=0; reg<256; reg++)
             { startc;
               data_write(0xa0);
               ack_write;
               data_write(reg);
               ack_write;
               data_write(0xff);
               ack_write;
               stop;
               delay(20);
             }
         if(dut_reg==512)
           { for (reg=0; reg<256; reg++)

                { startc;
                  data_write(0xa2);
                  ack_write;
                  data_write(reg);
                  ack_write;
                  data_write(0xff);
                  ack_write;
```

```
                stop;
                delay(20);
              }
          }
      }

  if (new_test==I2CCK8D)
     { for(page=0; page < (dut_reg/16);   page++)
          { slaveadd=0xa0+((page/16)*2);
            wordadd=page*16;
            if((page%2)==0)
               datab=0x55;
            else
               datab=0xaa;

            startc;
            data_write(slaveadd);
            ack_write;
            data_write(wordadd);
            ack_write;
            for(loop2=0; loop2 <16; loop2++)
               { data_write(datab);
                 ack_write;
               }
            stop;
            delay(20);
          }
     }

 if (new_test==I2CICK8D)
    { for(page=0; page < (dut_reg/16);   page++)
         { slaveadd=0xa0+((page/16)*2);
           wordadd=page*16;
           if((page%2)==0)
              datab=0xaa;
           else
              datab=0x55;

           startc;
           data_write(slaveadd);
           ack_write;
           data_write(wordadd);
           ack_write;
           for(loop2=0; loop2 <16; loop2++)
              { data_write(datab);
                ack_write;
              }
           stop;
           delay(20);
         }
    }

  if(new_test==I2CBZERO8)
  { for(block=81; block<87; block++)
    { column(block);
      startc;
      data_write(0xa0);
      ack_write;
      data_write(0);
      ack_write;
```

```
      for(reg=0; reg<16; reg++)
         { data_write(0);
           ack_write;
         }
      scl_5v_12v;
      delay(4);
      scl_12v;
      delay(4);
      stop;
      delay(20);
      cs_low;
      latch(0);
      scl_5v_12v;
      delay(12);
      scl_5v;
      delay(100);
    }
  }

if(new_test==I2CBONES)
{ for(block=81; block<87; block++)
  { column(block);
    startc;
    data_write(0xa0);
    ack_write;
    data_write(0);
    ack_write;
    for(reg=0; reg<16; reg++)
       { data_write(0xff);
         ack_write;
       }
    scl_5v_12v;
    delay(4);
    scl_12v;
    delay(4);
    stop;
    delay(20);
    cs_low;
    latch(0);
    scl_5v_12v;
    delay(12);
    scl_5v;
    delay(100);
  }
}

if(new_test==I2CB55)
{ for(block=81; block<87; block++)
  { column(block);
    startc;
    data_write(0xa0);
    ack_write;
    data_write(0);
    ack_write;
    for(reg=0; reg<16; reg++)
       { data_write(0x55);
         ack_write;
       }
    scl_5v_12v;
    delay(4);
```

```
        scl_12v;
        delay(4);
        stop;
        delay(20);
        cs_low;
        latch(0);
        scl_5v_12v;
        delay(12);
        scl_5v;
        delay(100);
    }
  }

  if(new_test==I2CBAA)
  { for(block=81; block<87; block++)
    { column(block);
      startc;
      data_write(0xa0);
      ack_write;
      data_write(0);
      ack_write;
      for(reg=0; reg<16; reg++)
         { data_write(0xaa);
           ack_write;
         }
      scl_5v_12v;
      delay(4);
      scl_12v;
      delay(4);
      stop;
      delay(20);
      cs_low;
      latch(0);
      scl_5v_12v;
      delay(12);
      scl_5v;
      delay(100);
    }
  }

  if(new_test==I2CINIVER || new_test==I2CINICHK)
  { for(block=81; block<87; block++)
    { column(block);
      startc;
      data_write(0xa0);
      ack_write;
      data_write(0);
      ack_write;
      for(reg=0; reg<16; reg++)
         { data_write(0);
           ack_write;
         }
      scl_5v_12v;
      delay(4);
      scl_12v;
      delay(4);
      stop;
      delay(20);
      cs_low;
      latch(0);
```

```
            scl_5v_12v;
            delay(12);
            scl_5v;
            delay(100);

            startc;
            data_write(0xa0);
            ack_write;
            data_write(0);
            ack_write;
            data_write(0xa0);
            ack_write;
            stop;
            delay(20);
        }
    }

    uw_mode;
    column(0);
    return;
}

    column(column_set);

    able(ENABLE);                       /*  Find IBM Japan rejects  */
    able(DISABLE);

    able(ENABLE);

    if(dut_type == NM95C12)             /*  Write all 0s to Register62 (Switch control) *
    {
        reg = 62;
        reg_rotate=_rotl(reg,dut_offset);   /*  Rotates reg for DUT sending         */

        cs_high;
        sk_toggle;
        start_bit;
        latch(write_send);
        clock(dut_opbits);
        latch(reg_rotate);
        clock(dut_adrbits);
        latch(0);
        clock(16);
        cs_low;
    }

    if(new_test == ERAL)
    {
        cs_high;
        if(true_dut == NMC9306) {sk_toggle;}
        start_bit;
        latch(eral_send);
        clock(4+(dut_adrbits-2));
        if(dut_type == NM59C11 || dut_type == NM59C16) clock(2);
        cs_low;
    }

    else if(new_test == WRAL || new_test == WRALL0)
    {
        cs_high;
```

```
    if(true_dut == NMC9306 || dut_type == NM95C12) {sk_toggle;}
    start_bit;
    latch(wral_send);
    clock(4+(dut_adrbits-2));
    latch(0);
    clock(16);
    cs_low;
}

else if(new_test == WRALL1)
{
    cs_high;
    if(dut_type == NM95C12) {sk_toggle;}
    start_bit;
    latch(wrall_send);
    clock(4+(dut_adrbits-2));
    latch(0xFF);
    clock(8);
    latch(0xFF);
    clock(8);
    cs_low;
}

else if(new_test == BYTEDIAG)
{
    if(dut_adrbits == 10)
    for(reg = 0; reg < dut_reg; reg+=17)
    {
        reg_rotate=_rotl(reg,2);                    /*  Rotates reg for DUT sending

        cs_high;
        start_bit;
        latch(erase_send);
        clock(dut_opbits);
        latch(reg);                                 /*  Send leading 2 bits          */
        clock(2);
        latch(reg_rotate);
        clock(8);
        cs_low;
        delay(twp);
    }
    else for(reg = 0; reg < dut_reg; reg+=5)
    {
        reg_rotate=_rotl(reg,dut_offset);           /*  Rotates reg for DUT sending */

        cs_high;
        start_bit;
        latch(erase_send);
        clock(dut_opbits);
        latch(reg_rotate);
        clock(dut_adrbits);
        cs_low;
        delay(twp);
    }
}

else if(new_test == EVENBYTE)
{
    if(dut_adrbits == 10)
    for(reg = 0; reg < dut_reg; reg+=2)
```

```
    {
        reg_rotate=_rotl(reg,2);                    /*  Rotates reg for DUT sending

        cs_high;
        start_bit;
        latch(erase_send);
        clock(dut_opbits);
        latch(reg);
        clock(2);                                   /*  Send leading 2 bits         */
        latch(reg_rotate);
        clock(8);
        cs_low;
        delay(twp);
    }
    else for(reg = 0; reg<dut_reg; reg +=2)
    {
        reg_rotate=_rotl(reg,dut_offset);           /*  Rotates reg for DUT sending */

        cs_high;
        if(true_dut == NMC9306) {sk_toggle;}
        start_bit;
        latch(erase_send);
        clock(dut_opbits);
        latch(reg_rotate);
        clock(dut_adrbits);
        cs_low;
        delay(twp);
    }
}

else if(new_test == ODDBYTE)
{
    if(dut_adrbits == 10)
    for(reg = 1; reg < dut_reg; reg+=2)
    {
        reg_rotate=_rotl(reg,2);                    /*  Rotates reg for DUT sending

        cs_high;
        start_bit;
        latch(erase_send);
        clock(dut_opbits);
        latch(reg);
        clock(2);                                   /*  Send leading 2 bits         */
        latch(reg_rotate);
        clock(8);
        cs_low;
        delay(twp);
    }
    else for(reg = 1; reg<dut_reg; reg +=2)
    {
        reg_rotate=_rotl(reg,dut_offset);           /*  Rotates reg for DUT sending */

        cs_high;
        if(true_dut == NMC9306) {sk_toggle;}
        start_bit;
        latch(erase_send);
        clock(dut_opbits);
        latch(reg_rotate);
        clock(dut_adrbits);
        cs_low;
```

```
            delay(twp);
        }
    }

    else if(dut_adrbits == 10)
    for(reg = 0; reg < dut_reg; reg++)                    /*  ALL BIT MANIPULATION TESTS  */
    {
        reg_rotate=_rotl(reg,2);                          /*  Rotates reg for DUT sending */

        cs_high;
        start_bit;
        latch(write_send);
        clock(dut_opbits);
        latch(reg);
        clock(2);                                         /*  Send leading 2 bits         */
        latch(reg_rotate);
        clock(8);
        lk = *((char *)&t_vector[reg]+1);
        latch(lk);
        clock(8);
        lk = *((char *)&t_vector[reg]);
        latch(lk);
        clock(8);
        cs_low;
        delay(twp);
    }
    else for(reg = 0; reg < dut_reg; reg++)               /* All Bit manipulation tests   */
    {
        reg_rotate=_rotl(reg,dut_offset);                 /*  Rotates reg for DUT sending */

        cs_high;
        if(true_dut == NMC9306 || dut_type == NM95C12) sk_toggle;
        start_bit;
        latch(write_send);
        clock(dut_opbits);
        latch(reg_rotate);
        clock(dut_adrbits);
        lk = *((char *)&t_vector[reg]+1);
        latch(lk);
        clock(8);
        lk = *((char *)&t_vector[reg]);
        latch(lk);
        clock(8);
        cs_low;
        delay(twp);
    }

    delay(twp);

    cs_high;
    cs_low;

    able(DISABLE);
}

/*****************************************************************/

/*              m3.c                */
#include <stdio.h>
```

```
#include <dos.h>
#include <conio.h>
#include <dir.h>
#include <math.h>
#include <io.h>
#include <sys/stat.h>
#include <time.h>
#include <string.h>
#include <\mill\variable.h>
#include <\mill\function.h>

/*********************************************************/

void   verify(int   new_test)
{
    register    int         reg;            /*  Register variables for speed        */
    register    int         col;

    int         ll,di;
    int         temp = 0;

    char    *message    =   "                                                       ";

    get_vector(new_test);

if(board_type==I2C46)
  { i2c_mode;
    for(row=0; row<max_row[board_type]; row++)
       { ll=0;
         for(col=1; col<=max_col[board_type]; col++)
            { chip=0;
              ll++;
              if((ll & 10) >= 10) ll += 6;
              outportb(C2,row);
              column(ll);
              di_dec(((row+1)/10)*16 + (row+1)%10);

            if(dut_board[row * max_col[board_type] + col] != 1) continue;
              {
              startc;
              data_write(0xa0);
              ack_write;
              data_write(0);
              ack_write;
              startc;
              data_write(0xa1);
              ack_write;
              data_read;
              d_buffer[0]=inportb(B2);
              for(reg=1; reg<dut_reg; reg++)
                 { ack_read;
                   data_read;
                   d_buffer[reg]=inportb(B2);
                 }
              stop;

              if(new_test!=I2CINICHK)
                { if(d_buffer[0] != t_vector[0])
                     chip=1;
                }
```

```
            if(new_test==I2CINIVER)
              { for(reg=0; reg<dut_reg; reg++)
                  { if(d_buffer[reg] != t_vector[reg])
                      chip=1;
                  }
              }

            if(new_test != I2CINICHK || new_test != I2CINIVER)
              { for(reg=0; reg<dut_reg; reg++)
                  { if(d_buffer[reg] != t_vector[reg])
                      chip=1;
                  }
              }

            if(chip==1)
              { bin_count[PASS] -= 1;
                dut_board[row * max_col[board_type] + col]=new_test;
                bin_count[new_test] += 1;
                bin_count[TOTALBAD] += 1;
              }
          }
        }
      }
    uw_mode;
    return;
}

    if(new_test == INITCHK)
    for(row = 0; row < max_row[board_type]; row++)          /*  CMOS parts only */
    {
        ll = 0;
        for(col = 1; col<=max_col[board_type]; col++)
        {
            ll++;
            if((ll & 10) >= 10)  ll += 6;

            outportb(C2,row);                               /*  row number */
            column(ll);                                     /*  column number */
            if(board_type==DIPC44)
              { di=(4*(row+1))-3;
                if(col>0 && col<12)     di=di;
                if(col>11 && col<23)    di=di+1;
                if(col>22 && col<34)    di=di+2;
                if(col>33 && col<45)    di=di+3;
                di=(di/10)*16 + di%10;
                di_dec(di);
              }

            if(board_type > 0 && board_type < 8)
              {
                di_dec(ll);
              }

            if(board_type==FLATPACK)
              { di=(5*(row+1))-4;
                if(col>0 && col<5)      di=di;
                if(col>4 && col<9)      di=di+1;
                if(col>8 && col<13)     di=di+2;
                if(col>12 && col<17)    di=di+3;
```

```
            if(col>16 && col<21)   di=di+4;
            di=(di/10)*16 + di%10;
            di_dec(di);
          }

        if(dut_board[row*max_col[board_type] + col] != 1) continue;
        {
            cs_high;
            if(dut_type == NM95C12) {sk_toggle;}
            start_bit;
            latch(read_send);
            clock(dut_opbits);
            latch(0);
            clock(dut_adrbits);
            clock(8);
            temp = 256 * inportb(82);
            clock(8);
            d_buffer[0] = temp+inportb(82);          /* Clock out byte2 */
            cs_low;                                  /* Save byte2      */

            if(d_buffer[0] != t_vector[0])
            {
                bin_count[PASS] -= 1;                /*  DECrement BIN 1 count */
                dut_board[row*max_col[board_type] + col] = new_test;
                bin_count[new_test] += 1;            /*  INcrement failbin  */
                bin_count[TOTALBAD] += 1;            /* INcrement Total fail count     */
            }
        }
    }
}
else for(row = 0; row < max_row[board_type]; row++)
{
    ll = 0;
    for(col = 1; col<=max_col[board_type]; col++)
    {
        ll++;
        if((ll & 10) >= 10)  ll += 6;
        outportb(C2,row);                            /*  row number */
        column(ll);                                  /*  column number */

     if(board_type==DIPC44)
       {
        di=(4*(row+1))-3;
        if(col>0  &&  col<12)   di=di;
        if(col>11  &&  col<23)  di=di+1;
        if(col>22  &&  col<34)  di=di+2;
        if(col>33  &&  col<45)  di=di+3;
        di=(di/10)*16 + di%10;
        di_dec(di);
       }

     if(board_type > 0 && board_type < 8)
       {
        di_dec(ll);
       }

     if(board_type== FLATPACK)
       { di=(5*(row+1))-4;
         if(col>0 && col<5)   di=di;
         if(col>4 && col<9)   di=di+1;
```

```
    if(col>8 && col<13)  di=di+2;
    if(col>12 && col<17)  di=di+3;
    if(col>16 && col<21)  di=di+4;
    di=(di/10)*16 + di%10;
    di_dec(di);
}

if(dut_board[row*max_col[board_type] + col] != 1) continue;

if(true_dut == NMC9306 || dut_type == NM95C12)          /*  9306 required  */
for(reg = 0; reg < dut_reg; reg++)
{
     cs_high;
     sk_toggle;
     start_bit;
     latch(read_send+reg);
     clock(16);
     temp = 256 * inportb(B2);
     clock(8);
     d_buffer[reg] = temp+inportb(B2);          /* Clock out byte2 */
     cs_low;                                    /* Save byte2      */

     if(d_buffer[reg] != t_vector[reg])
     {
          if(peckbd_stat == ON)
          {
               if(peckbd_1st_stat == ON)
               {
                    biosprint(0,10,0);
                    strprint("[BIN X]  ROW   COLUMN   LOOP (K)   REG   EXPECTED   ACTUA
                    biosprint(0,10,0);
                    strprint("         ------------------------------------------------
                    biosprint(0,10,0);
               }
               if(row < 10) strprint("       ");
                    else strprint("      ");
               itoa(row+1,message,10);
               strprint(message);

               if(col < 10) strprint("    ");
                    else strprint("   ");
               itoa(col,message,10);
               strprint(message);

               if((cycles*dut_reg)/1000 < 10) strprint("         ");
                    else strprint("        ");
               itoa(((cycles*dut_reg)/1000)+1,message,10);
               strprint(message);

               if(reg < 10)                 strprint("         ");
               if(reg > 9 && reg <100) strprint("        ");
               if(reg > 99)                 strprint("       ");
               itoa(reg,message,10);
               strprint(message);

               strprint("     ");
               itoa(t_vector[reg],message,16);
               strprint(message);

               strprint("         ");
```

```
            itoa(d_buffer[reg],message,16);
            strprint(message);
            biosprint(0,10,0);

            peckbd_1st_stat = OFF;          /*    Reset to prevent header printing
        }

        bin_count[PASS] -= 1;                   /* DECrement BIN 1 count          */
        dut_board[row*max_col[board_type] + col] = new_test;
        bin_count[new_test] += 1;               /* INcrement failbin              */
        bin_count[TOTALBAD] += 1;               /* INcrement Total fail count     */
        if(new_test != PECK3D) break;
    }
}
else if (dut_adrbits == 8 || dut_type == NM59C11)
for(reg = 0; reg < dut_reg; reg++)
{
    cs_high;
    start_bit;
    latch(read_send);
    clock(2);
    latch(reg);
    clock(16);
    temp = 256 * inportb(B2);
    clock(8);
    d_buffer[reg] = temp+inportb(B2);       /* Clock out byte2 */
    cs_low;                                 /* Save byte2      */

    if(d_buffer[reg] != t_vector[reg])
    {
        if(peckbd_stat == ON)
        {
            if(peckbd_1st_stat == ON)
            {
                biosprint(0,10,0);
                strprint("[BIN X]  ROW   COLUMN   LOOP (K)   REG   EXPECTED   ACTUA
                biosprint(0,10,0);
                strprint("         ------------------------------------------
                biosprint(0,10,0);
            }
            if(row < 10) strprint("          ");
              else strprint("         ");
            itoa(row+1,message,10);
            strprint(message);

            if(col < 10) strprint("      ");
              else strprint("     ");
            itoa(col,message,10);
            strprint(message);

            if((cycles*dut_reg)/1000 < 10) strprint("          ");
              else strprint("         ");
            itoa(((cycles*dut_reg)/1000)+1,message,10);
            strprint(message);

            if(reg < 10)              strprint("          ");
            if(reg > 9 && reg <100)   strprint("         ");
            if(reg > 99)              strprint("        ");
            itoa(reg,message,10);
            strprint(message);
```

```
                strprint("     ");
                itoa(t_vector[reg],message,16);
                strprint(message);

                strprint("          ");
                itoa(d_buffer[reg],message,16);
                strprint(message);
                biosprint(0,10,0);

                peckbd_1st_stat = OFF;              /*    Reset to prevent header printin
            }

            bin_count[PASS] -= 1;                    /*  DECrement BIN 1 count */
            dut_board[row*max_col[board_type] + col] = new_test;
            bin_count[new_test] += 1;                /*  INcrement failbin  */
            bin_count[TOTALBAD] += 1;                /* INcrement Total fail count    */
            if(new_test != PECKBD) break;
        }
    }
    else if (dut_adrbits == 6)
    for(reg = 0; reg < dut_reg; reg++)
    {
        cs_high;
        start_bit;
        latch(read_send+reg);
        clock(16);
        temp = 256 * inportb(82);
        clock(8);
        d_buffer[reg] = temp+inportb(82);        /* Clock out byte2 */
        cs_low;                                  /* Save byte2      */

        if(d_buffer[reg] != t_vector[reg])
        {
            if(peckbd_stat == ON)
            {
                if(peckbd_1st_stat == ON)
                {
                    biosprint(0,10,0);
                    strprint("[BIN X]  ROW   COLUMN   LOOP (K)   REG   EXPECTED   ACTUA
                    biosprint(0,10,0);
                    strprint("          ----------------------------------------------
                    biosprint(0,10,0);
                }
                if(row < 10) strprint("       ");
                    else strprint("        ");
                itoa(row+1,message,10);
                strprint(message);

                if(col < 10) strprint("      ");
                    else strprint("     ");
                itoa(col,message,10);
                strprint(message);

                if((cycles*dut_reg)/1000 < 10) strprint("          ");
                    else strprint("         ");
                itoa(((cycles*dut_reg)/1000)+1,message,10);
                strprint(message);

                if(reg < 10)              strprint("          ");
```

```
                    if(reg > 9 && reg <100) strprint("      ");
                    if(reg > 99)            strprint("       ");
                    itoa(reg,message,10);
                    strprint(message);

                    strprint("     ");
                    itoa(t_vector[reg],message,16);
                    strprint(message);

                    strprint("          ");
                    itoa(d_buffer[reg],message,16);
                    strprint(message);
                    biosprint(0,10,0);

                    peckbd_1st_stat = OFF;          /*    Reset to prevent header printing
                }

                bin_count[PASS] -= 1;                /*  DECrement BIN 1 count */
                dut_board[row*max_col[board_type] + col] = new_test;
                bin_count[new_test] += 1;            /*  INcrement failbin  */
                bin_count[TOTALBAD] += 1;            /* INcrement Total fail count    */
                if(new_test != PECKBD) break;
            }
        }
        else if (dut_adrbits == 10)
        for(reg = 0; reg < dut_reg; reg++)
        {
            reg_rotate =_rotl(reg,2);                /*  Rotates reg for DUT sending */
            cs_high;
            start_bit;
            latch(read_send);
            clock(dut_opbits);
            latch(reg);
            clock(2);                                /*  Send leading 2 bits         */
            latch(reg_rotate);
            clock(16);
            temp = 256 * inportb(B2);
            clock(8);
            d_buffer[reg] = temp+inportb(B2);        /* Clock out byte2 */
            cs_low;                                  /* Save byte2      */

            if(d_buffer[reg] != t_vector[reg])
            {
                bin_count[PASS] -= 1;                /*  DECrement BIN 1 count */
                dut_board[row*max_col[board_type] + col] = new_test;
                bin_count[new_test] += 1;            /*  INcrement failbin  */
                bin_count[TOTALBAD] += 1;            /* INcrement Total fail count    */
                if(new_test != PECKBD) break;
            }
        }
    }
  }
  di_dec(0);
}

void socket_check()
{
    register    int     reg;
    register    int     col;
```

```
    int         ll,di;
    int         temp = 0;

    socket_stat = SCON;                     /*  Socketcheck flag set          */
    textattr(WHITE+(GREEN<<4));             /*  Text = white; background = cyan */
    clrscr();
    gotoxy(25,10);
    printf("SOCKET CHECK IN PROGRESS");

if(board_type==I2C46)
  { i2c_mode;
    set_vcc(5000);
    column(0);
    di_dec(0);

    startc;
    data_write(0xa0);
    ack_write;
    data_write(0);
    ack_write;
    data_write(0xaa);
    ack_write;
    stop;
    delay(20);

    for(row=0; row<max_row[board_type]; row++)
       { ll=0;
         for(col=1; col<=max_col[board_type]; col++)
            { ll++;
              if((ll & 10) >= 10)  ll += 6;
              outportb(C2,row);
              column(ll);
              di_dec(((row+1)/10)*16 + (row+1)%10);

              startc;
              data_write(0xa0);
              ack_write;
              data_write(0);
              ack_write;
              startc;
              data_write(0xa1);
              ack_write;
              data_read;
              d_buffer[0]=inportb(B2);
              stop;

              if(d_buffer[0] != 0xaa)
                { bin_count[PASS] -= 1;
                  dut_board[row*max_col[board_type]+col]=SOCKETCHECK;
                  bin_count[SOCKETCHECK] += 1;
                  bin_count[TOTALBAD] += 1;
                }
            }
       }
    uw_mode;
    return;
  }

    column(0);
    di_dec(0);                              /*  Entire board                  */
```

```
set_vcc(vpoint[3]);                          /*  Set VCC = 5.0                    */
if(true_dut == NMC9306) {program(ERAL);}     /*                                   */
able(ENABLE);                                /*  Enable the part                  */

if(dut_type == NM95C12)                      /*  Write all 0s to Register62 (Switch control) */
{
    reg = 62;
    reg_rotate=_rotl(reg,dut_offset);        /*  Rotates reg for DUT sending */

    cs_high;
    sk_toggle;
    start_bit;
    latch(write_send);
    clock(dut_opbits);
    latch(reg_rotate);
    clock(dut_adrbits);
    latch(0);
    clock(16);
    cs_low;

    delay(twp);

    cs_high;
    cs_low;
}

reg = 15;
reg_rotate=_rotl(reg,dut_offset);            /*  Socketcheck with register 15     */
                                             /*  Rotates reg for DUT sending */

cs_high;
if(true_dut == NMC9306 || dut_type == NM95C12) {sk_toggle;}     /*  9306/95C12 required  */
start_bit;
latch(write_send);
clock(dut_opbits);
latch(reg_rotate);
clock(dut_adrbits);
latch(0xAA);
clock(8);                                                       /*  Send AA      */
latch(0xAA);
clock(8);                                                       /*  Send AA      */
cs_low;
delay(twp);

for(row = 0; row < max_row[board_type]; row++)
{
    ll = 0;
    for(col = 1; col<=max_col[board_type]; col++)
    {
        ll++;
        if((ll & 10) >= 10)  ll += 6;

        outportb(C2,row);                                       /*  row number       */
        column(ll);                                             /*  column number    */
        if(board_type==DIPC44)
          { di=(4*(row+1))-3;
            if(col>0 && col<12)   di=di;
            if(col>11 && col<23)  di=di+1;
            if(col>22 && col<34)  di=di+2;
            if(col>33 && col<45)  di=di+3;
            di=(di/10)*16 + di%10;
```

```
            di_dec(di);
          }
        if(board_type>0 && board_type<8)
          di_dec(11);
        if(board_type==FLATPACK)
          { di=(5*(row+1))-4;
            if(col>0 && col<5)      di=di;
            if(col>4 && col<9)      di=di+1;
            if(col>8 && col<13)     di=di+2;
            if(col>12 && col<17)    di=di+3;
            if(col>16 && col<21)    di=di+4;
            di=(di/10)*16 + di%10;
            di_dec(di);
          }

        if(dut_board[row*max_col[board_type] + col] != 1) continue;
        {
            cs_high;
            if(true_dut == NMC9306 || dut_type == NM95C12) {sk_toggle;}          /*  9306 r
            start_bit;
            latch(read_send);
            clock(dut_opbits);
            latch(reg_rotate);
            clock(dut_adrbits);
            clock(8);
            temp = 256 * inportb(B2);
            clock(8);                                          /*  Clock out byte2 */
            d_buffer[reg] = temp+inportb(B2);                  /*  Save byte2      */
            cs_low;

            if(d_buffer[reg] != 0xAAAA)
            {
                bin_count[PASS] -= 1;                          /* DECcrement BIN 1 count */
                dut_board[row*max_col[board_type] + col] = SOCKETCHECK;
                bin_count[SOCKETCHECK] += 1;                   /* INcrement failbin */
                bin_count[TOTALBAD] += 1;                      /* INcrement Total fail count
            }
        }
    }
}
column(0);
di_dec(0);                                                     /*  Entire board    */

if(true_dut == NMC9306 || true_dut == NMC9346)                 /*  NMOS ends in 1s */
{
    program(ERAL);
    verify(ERAL);
}
else
{                                                              /*  CMOS parts      */
    able(ENABLE);                                              /*  Enable the part

    cs_high;
    if(dut_type == NM95C12) {sk_toggle;}
    start_bit;
    latch(write_send);
    clock(dut_opbits);
    latch(reg_rotate);
    clock(dut_adrbits);
    latch(0);
```

```
        clock(16);
        cs_low;
        delay(twp);
    }
    able(DISABLE);
}                                                   /*  Disable the damn part

void   peckbd(int voltset)
{
    register    int     reg;
    register    int     col;

    int         temp = 0;
    int ll,k,di;

    column(0);                                      /*  Entire board                    */
    di_dec(0);

    set_vcc(vccpoint[3]);                           /*  Set VCC = 5.0V                  */

    peckbd_stat = ON;                               /*  Set PECKBD flag for verify fail */
    peckbd_1st_stat = ON;                           /*  1st time PECKBD print fail title block  */

    able(ENABLE);                                   /*  Enable the part                 */

    reg = 15;                                       /*  Socketcheck with register 15    */
    reg_rotate=_rotl(reg,dut_offset);               /*  Rotates reg for DUT sending */

    cs_high;
    if(dut_type == NM95C12) {sk_toggle;}
    start_bit;
    latch(write_send);
    clock(dut_opbits);
    latch(reg_rotate);
    clock(dut_adrbits);
    latch(0xAA);                                    /*  Send AA     */
    clock(8);
    latch(0xAA);                                    /*  Send AA     */
    clock(8);
    cs_low;
    delay(twp);

    for(row = 0; row < max_row[board_type]; row++)
    {
        ll = 0;
        for(col = 1; col<=max_col[board_type]; col++)
        {
            ll++;
            if((ll & 10) >= 10)  ll += 6;

            outportb(C2,row);                       /*  row number          */
            column(ll);                             /*  column number       */
            if(board_type==DIPC44)
              { di=(4*(row+1))-3;
                if(col>0 && col<12)   di=di;
                if(col>11 && col<23)  di=di+1;
                if(col>22 && col<34)  di=di+2;
                if(col>33 && col<45)  di=di+3;
                di=(di/10)*16 + di%10;
                di_dec(di);
```

```
    if(dut_board[row*max_col[board_type] + col] != 1)
    {
        if(dut_board[row*max_col[board_type] + col] != SOCKETCHECK)
        {dut_board[row*max_col[board_type] + col] = PECKBD;}
    }

ount[PECKBD] = bin_count[CKBD]+bin_count[INVCKBD]; /*  Set PECKBD bin_count          */
ount[CKBD] = 0;                                    /*  Reset CKBD bin_count = 0      */
ount[INVCKBD] = 0;                                 /*  Reset INVCKBD bin count = 0   */

d_stat = OFF;                                      /*  Reset PECKBD flag             */

int vcc)

S TEST ERASES, READS (VERIFYs) AND PROGRAMS 0s FOR EVERY DUT REGISTER    */

ter     int     reg;                               /*  Register variables for speed
ter     int     col;
        row;

        ll,di;
        temp = 0;

cc(5000);
am(ERAL);
cc(5600);
am(WRAL);
y(WRAL);                                           /*  Put background into all 0s

cc(vcc);
ector(REGWRITE);
n(0);
c(0);                                              /*  Entire board     */

ENABLE);

eg = 0; reg < dut_reg; reg++)

eg_rotate=_rotl(reg,dut_offset);                   /*  Rotates reg for DUT sending */

TER ERASE (FFFF) FOR REGISTER "reg" TO ENTIRE BOARD    */
olumn(0);
i_dec(0);                                          /*    Entire board          */

s_high;
f(true_dut == NMC9306 || dut_type == NM95C12) (sk_toggle;)          /*  9306 required
tart_bit;
atch(erase_send);
lock(dut_opbits);
atch(reg_rotate);
lock(dut_adrbits);
s_low;

elay(tmp);

BOARD FOR REGISTER "reg" = FFFF     */
or(row = 0; row < max_row[board_type]; row++)
    {
```

```
    11 = 0;
    for(col = 1; col<=max_col[board_type]; col++)
        {
        11++;
        if((11 & 10) >= 10) 11 += 6;

        if(dut_board[row*max_col[board_type] + col] != 1) continue;
        outportb(C2,row);                                   /*  row number          */
        column(11);                                         /*    column number       */
        if(board_type==DIPC44)
          { di=(4*(row+1))-3;
            if(col>0 && col<12)   di=di;
            if(col>11 && col<23)  di=di+1;
            if(col>22 && col<34)  di=di+2;
            if(col>33 && col<45)  di=di+3;
            di=(di/10)*16 + di%10;
            di_dec(di);
          }
        if(board_type>0 && board_type<8)
          { di_dec(11);
          }
        if(board_type==FLATPACK)
          { di=(5*(row+1))-4;
            if(col>0 && col<5)    di=di;
            if(col>4 && col<9)    di=di+1;
            if(col>8 && col<13)   di=di+2;
            if(col>12 && col<17)  di=di+3;
            if(col>16 && col<21)  di=di+4;
            di=(di/10)*16 +di%10;
            di_dec(di);
          }

        cs_high;
        if(true_dut == NMC9306 || dut_type == NM95C12) {sk_toggle;}          /*  9306 r
        start_bit;
        latch(read_send);
        clock(dut_opbits);
        latch(reg_rotate);
        clock(dut_adrbits);
        clock(8);
        temp = 256 * inportb(82);
        clock(9);
        d_buffer[reg] = temp+inportb(82);                   /*    Clock out byte2     */
        cs_low;                                             /*    Save byte2          */

        if(d_buffer[reg] != t_vector[reg])
        {
            bin_count[PASS] -= 1;                           /*    DECcrement bin_count BIN 1 co
            dut_board[row*max_col[board_type] + col] = REGWRITE;
            bin_count[REGWRITE] += 1;                       /*    INcrement bin_count of failbi
        }
    }


ISTER WRITE (0000) FOR REGISTER "reg" TO ENTIRE BOARD    */
olumn(0);
i_dec(0);

s_high;
f(true_dut == NMC9306 || dut_type == NM95C12) {sk_toggle;}           /*  9306 required
```

```
        start_bit;
        latch(write_send);
        clock(dut_opbits);
        latch(reg_rotate);
        clock(dut_adrbits);
        latch(0);
        clock(16);
        cs_low;

        delay(twp);
    }
/*    ALL PARTS ON THE BOARD ARE NOW PROGRAMMED WITH 0s    */

/*    CHECK ACTUAL VS. EXPECTED DATA FOR ALL REGISTERS AND SET FAILBINS    */
    cs_high;
    cs_low;
    able(DISABLE);
    printf("Verify \n");
    verify(REGWRITE0);
    return;
}
/*-------------------------------------*/
/*      PROGRAM REJECT UNIT(NMOS)      */
/*-------------------------------------*/
void   progrejn(int vcc)
 {
    int             dix=1;
    int             csx;
    int             col,row,k,dox;
    set_vcc(vcc);
      for(row = 1;row<max_row[board_type]+1;row++)
          {
           csx = 0;
           for(col = 1; col<=max_col[board_type]; col++)
              {
               k = dut_board[(row-1)*max_col[board_type] + col];
               csx++;

               if(board_type==DIPB46)
                 { printf("ERROR - WRONG BOARD TYPE \n");
                 }

               if(board_type==DIPC44)
                 { dix=(4*row)-3;
                   if(col>0  && col<12) dix=dix;
                   if(col>11 && col<23) dix=dix+1;
                   if(col>22 && col<34) dix=dix+2;
                   if(col>33 && col<45) dix=dix+3;
                   if((csx & 10) >= 10) csx += 6;
                   dix=(dix/10)*16 + dix%10;
                   outportb(C2,(row-1));
                   column(csx);
                   di_dec(dix);
                 }

               if(board_type==FLATPACK)
                 { dix=(5*row)-4;
                   if(dix>0  && dix<5) dix=dix;
                   if(dix>4  && dix<9) dix=dix+1;
                   if(dix>8  && dix<13) dix=dix+2;
```

```
    if(dix>12 && dix<17) dix=dix+3;
    if(dix>16 && dix<21) dix=dix+4;
    if((csx & 10) >= 10) csx += 6;
    dix=(dix/10)*16 + dix%10;
    outportb(C2,(row-1));
    column(csx);
    di_dec(dix);
  }

if(k != 1)
    {
     if(true_dut == NMC9306)                          /*  9306 parts requ
         {
          cs_high;
          sk_toggle;                                  /* ENABLE cycle  */
          start_bit;
          latch(enable_send);
          clock(4+(dut_adrbits-2));
          cs_low;

          delay(1);

          cs_high;
          sk_toggle;                                  /*  WRAL cycle  */
          start_bit;
          latch(wral_send);
          clock(4+(dut_adrbits-2));
          latch(0);
          clock(16);
          cs_low;

          delay(tmp);                                 /* Programming time */

          cs_high;                                    /*  DISABLE cycle
          sk_toggle;
          start_bit;
          latch(disable_send);
          clock(4+(dut_adrbits-2));
          cs_low;
          delay(1);
         }

    else
         {
          cs_high;                                    /*  ENABLE cycle */
          start_bit;
          latch(enable_send);
          clock(4+(dut_adrbits-2));
          cs_low;

          delay(1);

          cs_high;                                    /*  WRAL cycle    */
          start_bit;
          latch(wral_send);
          clock(4+(dut_adrbits-2));
          latch(0);
          clock(16);
          cs_low;
```

```
                        delay(twp);                                    /*  Programming time  *

                        cs_high;                                       /*  DISABLE cycle
                        start_bit;
                        latch(disable_send);
                        clock(4+(dut_adrbits-2));
                        cs_low;
                        delay(1);
                       }
                    }
                 else;
               }
          }
 }
/*---------------------------------------*/
/*      PROGRAM REJECT UNIT(CMOS)        */
/*---------------------------------------*/
void   progrejc(int vcc)
 {
    unsigned int ii;
    int             dix=1;
    int             csx;
    int             col,row,k,dox;
    set_vcc(vcc);
      for(row = 1;row<max_row[board_type]+1;row++)
          {
           csx = 0;
           for(col = 1; col<=max_col[board_type]; col++)
              {
               k = dut_board[(row-1)*max_col[board_type] + col];
               csx++;

               if(board_type==DIP846)
                 { printf("ERROR - WRONG BOARD TYPE \n");
                 }

               if(board_type==DIPC44)
                 { dix=(4*row)-3;
                   if(col>0  && col<12) dix=dix;
                   if(col>11 && col<23) dix=dix+1;
                   if(col>22 && col<34) dix=dix+2;
                   if(col>33 && col<45) dix=dix+3;
                   if((csx & 10) >= 10) csx += 6;
                   dix=(dix/10)*16 + dix%10;
                   outportb(C2,(row-1));
                   column(csx);
                   di_dec(dix);
                 }

               if(board_type==FLATPACK)
                 { dix=(5*row)-4;
                   if(dix>0  && dix<5)  dix=dix;
                   if(dix>4  && dix<9)  dix=dix+1;
                   if(dix>8  && dix<13) dix=dix+2;
                   if(dix>12 && dix<17) dix=dix+3;
                   if(dix>16 && dix<21) dix=dix+4;
                   if((csx & 10) >= 10) csx += 6;
                   dix=(dix/10)*16 + dix%10;
                   outportb(C2,(row-1));
```

```
        column(csx);
        di_dec(dix);
    }

if(k != 1)
    {
      if(dut_type == NM95C12)
        {
          cs_high;                                          /*  ENABLE cycle  */
          sk_toggle;
          start_bit;
          latch(enable_send);
          clock(4+(dut_adrbits-2));
          cs_low;
          delay(1);

          cs_high;                                          /*  WRALL1 (=EPAL) cy
          sk_toggle;
          start_bit;
          latch(wrall_send);
          clock(4+(dut_adrbits-2));
          latch(0xFF);
          clock(8);
          latch(0xFF);
          clock(8);
          cs_low;
          delay(twp);                                       /*  Programming time

          cs_high;                                          /*  DISABLE CYCLE */
          sk_toggle;
          start_bit;
          latch(disable_send);
          clock(4+(dut_adrbits-2));
          cs_low;
          delay(1);
        }
      else if(true_dut == CTRUE)
        {
          cs_high;                                          /*  ENABLE cycle  */
          start_bit;
          latch(enable_send);
          clock(4+(dut_adrbits-2));
          if(dut_type == NM59C11 || dut_type == NM59C16) clock(2);
          cs_low;
          delay(1);

          cs_high;                                          /*  ERAL cycle  */
          start_bit;
          latch(eral_send);
          clock(4+(dut_adrbits-2));
          if(dut_type == NM59C11 || dut_type == NM59C16) clock(2);
          cs_low;
          delay(twp);                                       /*  Programming time

          cs_high;                                              /*  DISABLE cycle
          start_bit;
          latch(disable_send);
          clock(4+(dut_adrbits-2));
          if(dut_type == NM59C11 || dut_type == NM59C16) clock(2);
          cs_low;
```

```
                    delay(1);
                }
            else                                                    /*  "CSTRUE"  */
                {
                    cs_high;                                        /*  ENABLE CYCLE    */
                    start_bit;
                    latch(enable_send);
                    clock(4+(dut_adrbits-2));
                    cs_low;
                    delay(1);

                    cs_high;                                        /*  WRALL1 (=ERAL) cy
                    start_bit;
                    latch(wrall_send);
                    clock(4+(dut_adrbits-2));
                    latch(0xFF);
                    clock(8);
                    latch(0xFF);
                    clock(8);
                    cs_low;
                    delay(twp);                                     /*  Programming time

                    cs_high;                                            /*  DISABLE CYCLE
                    start_bit;
                    latch(disable_send);
                    clock(4+(dut_adrbits-2));
                    cs_low;
                    delay(1);
                }
            }
        }
    }
}


void    stress()
{
    int ii = 0;

    if(true_dut != CSTRUE)
    {
        outportb(B1,0x21);                                  /* TIP light, Verify */
        delay(5);
        while((inportb(C1) & 16) == 0)                      /* If not in HALT mode */
        {
            delay(5);
            outportb(CW1,0x6);                              /* Prepare */
            delay(5);
            outportb(CW1,0x7);                              /*  HALT!              */
            for(ii = 0; ii<200 ; ii++)                      /*  Do it 200 time to */
            {
                delay(5);
                outportb(CW1,0x7);                          /*  Make sure it stops */
            }
            delay(5);
        }
        delay(5);
        outportb(CW1,0x7);                                  /*  Unknown command  */
        delay(5);
        outportb(CW1,0x5);                                  /*  Unknown command  */
```

```
        delay(5);
        column(0);                                  /*  Do entire board  */
        di_dec(0);
        delay(5);

        if(true_dut == NMC9306)
            outportb(p1,0x34);                      /*  TIP, 9306, stress   */
        else
            outportb(p1,0x24);                      /*  TIP, 9346, stress   */
        delay(5);

        outportb(CW1,0x6);                          /*  Prepare             */
        delay(5);
        outportb(CW1,0x7);                          /*  Halt                */
        delay(5);

        while((inportb(C1) & 16) == 0);             /*  Wait until halted   */

        outportb(CW1,0x4);                          /*  Return to auto mode */
        printf("Verify \n");                        /*  Show "Verify"       */
        set_vcc(vccpoint[2]);                       /*  Set VCC = 4.4V      */
        verify(STRESS);                             /*  Verify WRALL condition */
    }

    if(true_dut == CSTRUE)
    {
        column(0);                                  /*  Send to entire board    */
        di_dec(0);

        cs_high;                                    /*  (1) Disable part        */
        if(dut_type == NM95C12) {sk_toggle;}
        start_bit;
        latch(disable_send);
        clock(4+(dut_adrbits-2));
        cs_low;
        delay(1);

        cs_high;
        if(dut_type == NM95C12) {sk_toggle;}
        start_bit;
        latch(wral_send);                           /*  (2) WRALL0              */
        clock(4+(dut_adrbits-2));
        set_vcc(vccpoint[8]);                       /*  (3) Set VCC = 8.0V      */
        delay(1000);
        set_vcc(vccpoint[3]);                       /*  (4) Return VCC to 5.0V  */
        cs_low;                                     /*  Abort stress mode       */

        printf("Verify \n");
        verify(STRESS);                             /*  (5) Read 0s             */
    }
}

/*************************************************/

/*              m412.c              */
#include <stdio.h>
#include <dos.h>
#include <conio.h>
#include <dir.h>
#include <math.h>
```

```
#include <io.h>
#include <sys/stat.h>
#include <time.h>
#include <string.h>
#include <\mill\variable.h>
#include <\mill\function.h>

/*************************************************/

void datatest()
{
 int temp,test_row,endt;
 char *mess ="                    ";

able(ENABLE);
di_dec(1);
column(1);
for(loop1 = 0xffff; loop1 >0; loop1--)
   { high=loop1 & 0xff00;
     high=high / 256;
     low=loop1 & 0xff;

     if((loop1 % 0x100)==0)
       { printf("%x %x    \n",high,low);
       }

     if(kbhit())
       { endt=getch();
         if(endt=='x' || endt=='X')
            return;
       }

     cs_high;
     start_bit;
     latch(write_send);
     clock(dut_adrbits);
     latch(high);
     clock(8);
     latch(low);
     clock(8);
     cs_low;
     delay(20);
     cs_high;
     cs_low;

     for(test_row=0; test_row< 1; test_row++)
        { outportb(C2,test_row);
          cs_high;
          start_bit;
          latch(read_send);
          clock(16);
          temp=256*inportb(32);
          clock(8);
          d_buffer[0]=temp+inportb(32);
          cs_low;

          if(d_buffer[0] != loop1)
            { printf("Cycle Failure at location(col,row) %d,1\n",test_row+1);
              printf("Cycle Failure at data %x%x     Actual Data  %x \n",high,low,d_buffer[0]);
              strprint("Cycle failure at location(col,row) ");
```

```
                itoa(test_row+1,mess,10);
                strprint(mess);
                strprint(",1");
                biosprint(0,10,0);
                strprint("Cycle failure at data  ");
                itoa(high,mess,16);
                strprint(mess);
                strprint(" ");
                itoa(low,mess,16);
                strprint(mess);
                strprint("     ");
                strprint("Actual Data  ");
                itoa(c_buffer[0],mess,16);
                strprint(mess);
                biosprint(0,10,0);
              }
            }
          }
    able(DISABLE);
}

void vdo()
{
 int temp,chip1,reg_add,endt;
 char *mess = "                    ";

 di_dec((36/10)*16+36%10);
 column((36/10)*16+36%10);
outportb(C2,0);

  for(loop1 = 0xffff; loop1 > 0;  loop1--)
     { high=loop1 & 0xff00;
       high=high / 256;
       low=loop1 & 0xff;
       if((loop1 % 0x010) ==0)
     { printf("%x %x    \n",high,low);
     }

if(kbhit())
   { endt=getch();
     if(endt=='x' || endt=='X')
        return;
   }

for(reg_add=0x2e; reg_add<=0x3d; reg_add++)
    { able(ENABLE);

        cs_high;
        start_bit;
        latch(erase_send);
        clock(dut_opbits);
        latch(reg_add * 4);
        clock(dut_adrbits);
        cs_low;
        delay(30);
        cs_high;
        cs_low;

        able(DISABLE);
```

```
for(chip1=0; chip1< 6; chip1++)
   { outportb(C2,chip1);

     cs_high;
     start_bit;
     latch(read_send+reg_add);
     clock(16);
     temp=256*inportb(82);
     clock(8);
     d_buffer[chip1]=temp+inportb(82);
     cs_low;
   }

for(chip1=0; chip1< 6; chip1++)
   { if(d_buffer[chip1] != 0xffff)
       { printf("Cycle failure at location(col,row) %d,36\n",chip1+1);
         printf("Cycle failure at data  FFFF    Actual Data    %x   \n",d_buffer[chip1]);
         printf("Cycle failure at register  %x  \n",reg_add);
         strprint("Cycle failure at location(col,row)  ");
         itoa(chip1+1,mess,10);
         strprint(mess);
         strprint(",36");
         biosprint(0,10,0);
         strprint("Cycle failure at data  FFFF");
         strprint("    Actual Data  ");
         itoa(d_buffer[chip1],mess,16);
         strprint(mess);
         biosprint(0,10,0);
         strprint("Cycle failure at register  ");
         itoa(reg_add,mess,16);
         strprint(mess);
         biosprint(0,10,0);
       }
    }

    able(ENABLE);

    cs_high;
    start_bit;
    latch(write_send);
    clock(dut_opbits);
    latch(reg_add*4);
    clock(dut_adrbits);
    latch(high);
    clock(8);
    latch(low);
    clock(8);
    cs_low;
    delay(36);
    cs_high;
    cs_low;

    able(DISABLE);

    for(chip1=0; chip1< 6; chip1++)
       { outportb(C2,chip1);

         cs_high;
         start_bit;
         latch(read_send+reg_add);
```

```
                clock(16);
                temp=256*inportb(52);
                clock(8);
                d_buffer[chip1]=temp+inportb(32);
                cs_low;
              }

          for(chip1=0; chip1< 5; chip1++)
             { if(d_buffer[chip1] != loop1)
                 { printf("Cycle failure at location(col,row) %d,36\n",chip1+1);
                   printf("Cycle failure at data  %x %x   Actual Data  %x \n",high,low,d_buffer[
                   printf("Cycle failure at register  %x\n",reg_add);
                   strprint("Cycle failure at location(col,row)  ");
                   itoa(chip1+1,mess,10);
                   strprint(mess);
                   strprint(",36");
                   biosprint(0,10,0);
                   strprint("Cycle failure at data  ");
                   itoa(high,mess,16);
                   strprint(mess);
                   strprint(" ");
                   itoa(low,mess,16);
                   strprint(mess);
                   strprint("    ");
                   strprint("Actual Data  ");
                   itoa(d_buffer[chip1],mess,16);
                   strprint(mess);
                   biosprint(0,10,0);
                   strprint("Cycle failure at register  ");
                   itoa(reg_add,mess,16);
                   strprint(mess);
                   biosprint(0,10,0);
                 }
              }
           }
        }
}

void mitsu()
{
int temp,chip1,reg_add,datap,endt;
char *mess ="                    ";

di_dec((36/10)*16+36%10);
column((36/10)*16+36%10);
outportb(C2,0);

t_vector[1]=0xfffe;
t_vector[2]=0xfffc;
t_vector[3]=0xfff8;
t_vector[4]=0xfff0;
t_vector[5]=0xffe0;
t_vector[6]=0xffc0;
t_vector[7]=0xff80;
t_vector[8]=0xff00;
t_vector[9]=0xfe00;
t_vector[10]=0xfc00;
t_vector[11]=0xf800;
t_vector[12]=0xf000;
t_vector[13]=0xe000;
```

```
t_vector[14]=0xc000;
t_vector[15]=0x8000;
t_vector[16]=0x0000;

for(loop1=0; loop1<255; loop1++)
   { printf("%d  \n",loop1);

     if(kbhit())
       { endt=getch();
         if(endt=='x' || endt=='X')
            return;
       }

     for(reg_add=1; reg_add < 4; reg_add++)
        { able(ENABLE);

          cs_high;
          start_bit;
          latch(erase_send);
          clock(dut_opbits);
          latch(reg_add * 4);
          clock(dut_adrbits);
          cs_low;
          delay(20);
          cs_high;
          cs_low;

          able(DISABLE);
        }

     for(reg_add=1; reg_add<4; reg_add++)
        { for(chip1=0; chip1< 6; chip1++)
             { outportb(C2,chip1);

               cs_high;
               start_bit;
               latch(read_send+reg_add);
               clock(16);
               temp=256*inportb(32);
               clock(8);
               d_buffer[chip1]=temp+inportb(32);
               cs_low;
             }

          for(chip1=0; chip1< 6; chip1++)
             { if(d_buffer[chip1] != 0xffff)
                  { printf("Cycle failure at location(col,row) %d,36\n",chip1+1);
                    printf("Cycle failure at data FFFF    Actual Data    %x    \n",d_buffer[chip1
                    printf("Cycle failure at register  %x  \n",reg_add);
                    strprint("Cycle failure at location(col,row)  ");
                    itoa(chip1+1,mess,10);
                    strprint(mess);
                    strprint(",36");
                    biosprint(0,10,0);
                    strprint("Cycle failure at data  FFFF");
                    strprint("    Actual Data  ");
                    itoa(d_buffer[chip1],mess,16);
                    strprint(mess);
                    biosprint(0,10,0);
                    strprint("Cycle failure at register  ");
```

```
                itoa(reg_add,mess,16);
                strprint(mess);
                biosprint(0,10,0);
              }
          }
      }
for(datap=1; datap<17; datap++)
   { high=t_vector[datap] & 0xff00;
     high=high/256;
     low=t_vector[datap] & 0xff;

     for(reg_add=1; reg_add<4; reg_add++)
        { able(ENABLE);

          cs_high;
          start_bit;
          latch(write_send);
          clock(dut_opbits);
          latch(reg_add*4);
          clock(dut_adrbits);
          latch(high);
          clock(8);
          latch(low);
          clock(8);
          cs_low;
          delay(20);
          cs_high;
          cs_low;

          able(DISABLE);
        }
     for(reg_add=1; reg_add<4; reg_add++)
        { for(chip1=0; chip1< 6; chip1++)
            { outportb(C2,chip1);

              cs_high;
              start_bit;
              latch(read_send+reg_add);
              clock(16);
              temp=256*inportb(82);
              clock(8);
              d_buffer[chip1]=temp+inportb(82);
              cs_low;
            }

          for(chip1=0; chip1< 6; chip1++)
             { if(d_buffer[chip1] != t_vector[datap])
                 { printf("Cycle failure at location(col,row) %d,36\n",chip1+1);
                   printf("Cycle failure at data  %x %x    Actual Data   %x \n",high,low,d_b
                   printf("Cycle failure at register  %x\n",reg_add);
                   strprint("Cycle failure at location(col,row)  ");
                   itoa(chip1+1,mess,10);
                   strprint(mess);
                   strprint(",36");
                   biosprint(0,10,0);
                   strprint("Cycle failure at data  ");
                   itoa(high,mess,16);
                   strprint(mess);
                   strprint(" ");
                   itoa(low,mess,16);
```

```
                        strprint(mess);
                        strprint("    ");
                        strprint("Actual Data  ");
                        itoa(d_buffer[chip1],mess,16);
                        strprint(mess);
                        biosprint(0,10,0);
                        strprint("Cycle failure at register  ");
                        itoa(reg_add,mess,16);
                        strprint(mess);
                        biosprint(0,10,0);
                      }
                  }
              }
          }
      }
}

void  i2cpeones(unsigned int cycles)
{
  register int reg;
  unsigned int ii;
  int k,block;
  i2c_mode;
  column(0);
  di_dec(0);

  for(ii=0; ii<cycles; ii++)
  for(k=0; k<1000; k++)
  for(block=81; block<87; block++)
     { column(block);
       startc;
         data_write(0xa0);
       ack_write;
       data_write(0);
       ack_write;
         for(reg=0; reg<16; reg++)
          { data_write(0);
            ack_write;
          }
       scl_5v_12v;
       delay(4);
       scl_12v;
       delay(4);
       stop;
       delay(20);
       cs_low;
       latch(0);
       scl_5v_12v;
       delay(12);
       scl_5v;
       delay(100);

       startc;
       data_write(0xa0);
         ack_write;
       data_write(0);
       ack_write;
       for(reg=0; reg<16; reg++)
          { data_write(0xff);
            ack_write;
```

```
        }
      scl_5v_12v;
      delay(4);
        scl_12v;
        delay(4);
      stop;
        delay(20);
        cs_low;
      latch(0);
        scl_5v_12v;
        delay(12);
        scl_5v;
        delay(100);
    }

  uw_mode;
}

void  i2cpezeros(unsigned int cycles)
{
  register int reg;
  unsigned int ii;
  int k,block;

  i2c_mode;
  column(0);
  di_dec(0);

  for(ii=0; ii<cycles; ii++)
  for(k=0; k<1000; k++)
  for(block=81; block<97; block++)
     { column(block);
       startc;
         data_write(0xa0);
         ack_write;
       data_write(0);
         ack_write;
       for(reg=0; reg<16; reg++)
          { data_write(0xff);
            ack_write;
          }
       scl_5v_12v;
         delay(4);
         scl_12v;
         delay(4);
         stop;
       delay(20);
         cs_low;
       latch(0);
         scl_5v_12v;
         delay(12);
         scl_5v;
         delay(100);

         startc;
         data_write(0xa0);
         ack_write;
         data_write(0);
         ack_write;
         for(reg=0; reg<16; reg++)
```

```
            { data_write(0);
              ack_write;
            }
          scl_5v_12v;
          delay(4);
          scl_12v;
          delay(4);
          stop;
        delay(20);
          cs_low;
        latch(0);
          scl_5v_12v;
          delay(12);
          scl_5v;
          delay(100);
      }

  uw_mode;
}

void   pecycles1(unsigned int  cycles)                  /*    P/E Cycles ends in 1s    */
{
    unsigned int ii;
    int k;
    column(0);                                          /*  Entire board */
    di_dec(0);

    if(true_dut == NMC9306)                             /*  9306 parts require a 01 OPCODE
    {
        for(ii = 0; ii<cycles; ii++)
        for(k=0; k<1000; k++)
        {
            cs_high;                                    /* ENABLE cycle  */
            sk_toggle;
            start_bit;
            latch(enable_send);
            clock(4+(dut_adrbits-2));
            cs_low;

            delay(1);

            cs_high;                                    /*  WRAL cycle   */
            sk_toggle;
            start_bit;
            latch(wral_send);
            clock(4+(dut_adrbits-2));
            latch(0);
            clock(16);
            cs_low;

            delay(twp);                                 /* Programming time */

            cs_high;                                    /* ERAL cycle   */
            sk_toggle;
            start_bit;
            latch(eral_send);
            clock(4+(dut_adrbits-2));
            cs_low;
            delay(twp);                                 /* Programming time */
        }
```

```
        cs_high;                                        /*  DISABLE cycle  */
        sk_toggle;
        start_bit;
        latch(disable_send);
        clock(4+(dut_adrbits-2));
        cs_low;
        delay(1);
    }

    else
    {
        for(ii = 0; ii<cycles; ii++)
        for(k=0; k<1000; k++)
        {
            cs_high;                                    /*  ENABLE cycle */
            start_bit;
            latch(enable_send);
            clock(4+(dut_adrbits-2));
            cs_low;

            delay(1);

            cs_high;                                    /*  WRAL cycle     */
            start_bit;
            latch(wral_send);
            clock(4+(dut_adrbits-2));
            latch(0);
            clock(16);
            cs_low;
            delay(twp);                                 /*  Programming time  */

            cs_high;                                    /*  ERAL cycle */
            start_bit;
            latch(eral_send);
            clock(4+(dut_adrbits-2));
            cs_low;
            delay(twp);                                 /*  Programming time  */
        }

        cs_high;                                        /*  DISABLE cycle     */
        start_bit;
        latch(disable_send);
        clock(4+(dut_adrbits-2));
        cs_low;
        delay(1);
    }
}

void   pecycles0(unsigned int  cycles)                  /*     P/E Cycles ends in 0s     */
{                                                       /*  CMOS devices: 0s go into Bake   */
    unsigned int ii;
    int k;

    column(0);                                          /*  Entire board */
    di_dec(0);

    if(dut_type == NM95C12)
    {
        for(ii = 0; ii<cycles; ii++)
```

```
    for(k=0; k<1000; k++)
    {
        cs_high;                                        /*  ENABLE cycle  */
        sk_toggle;
        start_bit;
        latch(enable_send);
        clock(4+(dut_adrbits-2));
        cs_low;
        delay(1);

        cs_high;                                        /*  WRALL1 (=ERAL) cycle    */
        sk_toggle;
        start_bit;
        latch(wrall_send);
        clock(4+(dut_adrbits-2));
        latch(0xFF);
        clock(8);
        latch(0xFF);
        clock(8);
        cs_low;
        delay(twp);                                     /*  Programming time  */

        cs_high;                                        /*  WRALL0 (=WRAL) cycle    */
        sk_toggle;
        start_bit;
        latch(wrall_send);
        clock(4+(dut_adrbits-2));
        latch(0);
        clock(16);
        cs_low;
        delay(twp);                                     /*  Programming time */
    }
    cs_high;                                            /*  DISABLE CYCLE */
    sk_toggle;
    start_bit;
    latch(disable_send);
    clock(4+(dut_adrbits-2));
    cs_low;
    delay(1);
}
else if(true_dut == CTRUE)
{
    for(ii = 0; ii<cycles; ii++)
    for(k=0; k<1000; k++)
    {
        cs_high;                                        /*  ENABLE cycle  */
        start_bit;
        latch(enable_send);
        clock(4+(dut_adrbits-2));
        if(dut_type == NM59C11 || dut_type == NM59C16) clock(2);
        cs_low;
        delay(1);

        cs_high;                                        /*  ERAL cycle  */
        start_bit;
        latch(eral_send);
        clock(4+(dut_adrbits-2));
        if(dut_type == NM59C11 || dut_type == NM59C16) clock(2);
        cs_low;
        delay(twp);                                     /*  Programming time  */
```

```
            cs_high;                                        /*  WRAL cycle          */
            start_bit;
            latch(wral_send);
            clock(4+(dut_adrbits-2));
            latch(0);
            clock(16);
            cs_low;
            delay(twp);                                     /*  Programming time    */
        }
        cs_high;                                            /*  DISABLE cycle       */
        start_bit;
        latch(disable_send);
        clock(4+(dut_adrbits-2));
        if(dut_type == NM59C11 || dut_type == NM59C16) clock(2);
        cs_low;
        delay(1);
    }
    else                                                    /*  "CSTRUE"            */
    {
        for(ii = 0; ii<cycles; ii++)
        for(k=0; k<1000; k++)
        {
            cs_high;                                        /*  ENABLE CYCLE        */
            start_bit;
            latch(enable_send);
            clock(4+(dut_adrbits-2));
            cs_low;
            delay(1);

            cs_high;                                        /*  WRALL1 (=ERAL) cycle    */
            start_bit;
            latch(wrall_send);
            clock(4+(dut_adrbits-2));
            latch(0xFF);
            clock(8);
            latch(0xFF);
            clock(8);
            cs_low;
            delay(twp);                                     /*  Programming time    */

            cs_high;                                        /*  WRALL0 (=WRAL) cycle    */
            start_bit;
            latch(wrall_send);
            clock(4+(dut_adrbits-2));
            latch(0);
            clock(16);
            cs_low;
            delay(twp);                                     /*  Programming time    */
        }
        cs_high;                                            /*  DISABLE CYCLE       */
        start_bit;
        latch(disable_send);
        clock(4+(dut_adrbits-2));
        cs_low;
        delay(1);
    }
}

void    able(int  mode)
```

```
{
    cs_high;
    if(true_dut == NMC9306 || dut_type == NM95C12) {sk_toggle;}          /*  9306 required  */
    start_bit;
    if(mode == ENABLE)      {latch(enable_send);}
    if(mode == DISABLE)     {latch(disable_send);}
    clock(4);                                                           /*  Send EWEN/EWDS          */
    latch(0);
    clock(dut_adrbits-2);
    if(dut_type == NM59C11 || dut_type == NM59C16) clock(2);
    cs_low;
    delay(1);
}

void   set_vcc(int  volts)
{
    int     ii;
    int     jj = 0;

    outportb(CW3,0x0);

    outportb(CW3,0x4);

    for(ii = 0; ii<120; ii++)
    {
        outportb(CW3,0x3);
        outportb(CW3,0x3);
        outportb(CW3,0x3);
        outportb(CW3,0x3);
        outportb(CW3,0x3);
        outportb(CW3,0x2);
        outportb(CW3,0x2);
        outportb(CW3,0x2);
        outportb(CW3,0x2);
    }


    if(volts == 0)  return;

    for(ii = 0; ii<9 ; ii++)
        if(volts == vccpoint[ii])  jj = vccindex[ii];

    if(jj == 0)
        jj = (int)(m * (volts/1000.0) + b);

    outportb(CW3,0x5);
    for(ii = 0; ii<jj; ii++)
    {
        outportb(CW3,0x3);
        outportb(CW3,0x3);
        outportb(CW3,0x3);
        outportb(CW3,0x3);
        outportb(CW3,0x3);
        outportb(CW3,0x2);
        outportb(CW3,0x2);
        outportb(CW3,0x2);
        outportb(CW3,0x2);
    }
    delay(400);
```

```
    outportb(CW3,0x4);
}


void save_vcc()
{
 outportb(CW3,0x3);
 outportb(CW3,0x3);

 outportb(CW3,0x1);
 delay(40);
}

void   vcc_set()
{
    int i;
/*-------------------------------------------------------------------------*/
/*  Don't ask why this works.  It is used to set up the automatic VCC mode  */
/*  for the exercisor.  DON'T CHANGE ANY OF THIS!!!                         */
/*-------------------------------------------------------------------------*/
    outportb(CW3,0x9);                                  /*    AUTO VCC mode; DECREASE
    outportb(CW3,0x0);

    for(i = 0; i<100; i++)                              /*    Lower VCC to 0.0V
    {
        outportb(CW3,0x3);
        outportb(CW3,0x3);
        outportb(CW3,0x3);
        outportb(CW3,0x3);
        outportb(CW3,0x3);
        outportb(CW3,0x2);
        outportb(CW3,0x2);
        outportb(CW3,0x2);
        outportb(CW3,0x2);
    }

    outportb(CW3,0x5);                                  /*    AUTO VCC mode; INCREASE

    for(i = 0; i<32; i++)                               /*    Set VCC to 5.0V
    {
        outportb(CW3,0x3);
        outportb(CW3,0x3);
        outportb(CW3,0x3);
        outportb(CW3,0x3);
        outportb(CW3,0x3);
        outportb(CW3,0x2);
        outportb(CW3,0x2);
        outportb(CW3,0x2);
        outportb(CW3,0x2);
    }

    outportb(CW3,0x8);                                  /*    Update Digital pots

    delay(400);

    outportb(CW3,0x7);                                  /*    Manual VCC mode

    outportb(CW3,0x4);                                  /*    DECREASE direction
}
```

```
void   vcc_cal()                                    /*    If stat.dat is not found or c
{
    int      j = 0;
    int      k = 0;
    char     temp;

    strcpy(password,"ZAP");
    strcpy(calpass,"ZIP");

    window(1,1,80,25);
    clrscr();
    textcolor(YELLOW);
    highvideo();
    gotoxy(1,1);
    cprintf("MAINTENANCE VCC CALIBRATION");
    gotoxy(73,1);
    cprintf("(%s)",rev);
    lowvideo();
    textcolor(WHITE);
    window(1,2,80,25);

    while(stricmp(password,calpass))                /*  Password not matched    */
    {
        gotoxy(20,3);
        printf("PLEASE ENTER THE CALIBRATION PASSWORD:   ");
        gets(password);

        _chmod("c:/sbt/password.dat",S_IREAD,FA_HIDDEN);

        if((infile = fopen("c:/sbt/password.dat","rb")) == NULL)
        {
            gotoxy(20,5);
            printf("Error:  PASSWORD FILE NOT PRESENT\n");
            gotoxy(20,7);
            printf("...FILE GENERATION IN PROGRESS...\n\n");
            gotoxy(20,8);
            printf("PLEASE ENTER NEW PASSWORD:   ");
            gets(password);

            creat("c:/sbt/password.dat",S_IREAD|S_IWRITE);
            if((infile = fopen("c:/sbt/password.dat","r+b")) == NULL)
            {
                printf("FATAL ERROR:   CANNOT OPEN PASSWORD FILE!");
                delay(2000);
                exit(2);
            }

            fwrite(password,sizeof(password),1,infile);
            fclose(infile);
            chmod("c:/sbt/password.dat",S_IREAD);
        }
        else
        {
            fread(calpass,sizeof(calpass),1,infile);
            fclose(infile);
        }
    }
    if(stricmp(password,calpass))  vcc_cal();       /*  Password not matched    */
```

```
gotoxy(20,15);
printf("DO YOU WANT TO CHANGE THE PASSWORD (Y/N)?");
gets(input);
if(!stricmp(input,"Y"))
{
     chmod("c:/sbt/password.dat",S_IREAD|S_IWRITE);
     fopen("c:/sbt/password.dat","r+b");

     gotoxy(20,17);
     printf("ENTER NEW PASSWORD:      ");
     gets(password);

     fwrite(password,sizeof(password),1,infile);
     fclose(infile);
     chmod("c:/sbt/password.dat",S_IREAD);
}

clrscr();

outportb(CW3,0x0);

gotoxy(29,3);
printf("VCC calibration mode...");
gotoxy(23,5);
printf("Press the <u> key to increase the Vcc");
gotoxy(23,6);
printf("Press the <d> key to decrease the Vcc");
gotoxy(13,8);
printf("Press the <ENTER> key when the Vcc is closest to      mV");

for(j = 0; j<100; j++)
{
     outportb(CW3,0x3);
     outportb(CW3,0x3);
     outportb(CW3,0x3);
     outportb(CW3,0x3);
     outportb(CW3,0x3);
     outportb(CW3,0x2);
     outportb(CW3,0x2);
     outportb(CW3,0x2);
     outportb(CW3,0x2);
}

outportb(CW3,0x5);
for(j = 0; j<9; j++)
{
     while(1)
     {
          gotoxy(62,8);
          printf("%d",vccpoint[j]);

          temp = getch();

          if(temp == 'd' || temp == 'D')
          {
               outportb(CW3,0x4);
               k--;
               outportb(CW3,0x3);
               outportb(CW3,0x3);
               outportb(CW3,0x3);
```

```
            outportb(CW3,0x3);
            outportb(CW3,0x3);
            outportb(CW3,0x2);
            outportb(CW3,0x2);
            outportb(CW3,0x2);
            outportb(CW3,0x2);
        }
        if(temp == 'u' || temp == 'U')
        {
            outportb(CW3,0x5);
            k++;
            outportb(CW3,0x3);
            outportb(CW3,0x3);
            outportb(CW3,0x3);
            outportb(CW3,0x3);
            outportb(CW3,0x3);
            outportb(CW3,0x2);
            outportb(CW3,0x2);
            outportb(CW3,0x2);
            outportb(CW3,0x2);
        }
        if(temp == 13)
        {
            vccindex[j] = k;
            break;
        }
    }
}

outportb(CW3,0x4);

for(j = 0; j<100; j++)
{
    outportb(CW3,0x3);
    outportb(CW3,0x3);
    outportb(CW3,0x3);
    outportb(CW3,0x3);
    outportb(CW3,0x3);
    outportb(CW3,0x3);
    outportb(CW3,0x2);
    outportb(CW3,0x2);
    outportb(CW3,0x2);
    outportb(CW3,0x2);
}
outportb(CW3,0x5);

save_vcc();

chmod("c:/sbt/stat.dat",S_IREAD|S_IWRITE);
if((infile = fopen("c:/sbt/stat.dat","r+b")) == NULL)
{
    printf("Fatal Error:   Cannot open stat file!  Contact Engineer");
    delay(2000);
    exit(2);
}

for(j = 0; j<9; j++) fwrite(&vccindex[j],sizeof(vccindex[j]),1,infile);

fclose(infile);
chmod("c:/sbt/stat.dat",S_IREAD);
```

```
    run_stat = ENGMODE;
}

void   strprint(char  *item)
{
    int i;
    for(i = 0; i<strlen(item); i++)
        biosprint(0,item[i],0);
}

/*-------------------------------- EXPECTED DATA & PROGRAMMED DATA PATTERNS ------------------
void   get_vector(int  test)
{
    int          i = 0;
    register     int reg;                                  /*  Register variables for speed     */
    unsigned     int b_mask_BIT = 0x11*256 + 0x11;         /*     Bit mask = 0001 0001 0001 0001    */


    for(i = 0; i<16; i++)                                  /*  Generate the bit masks           */
        b_mask[i] = (int)pow(2,i);                         /*  Coercion to int is required      */

if(test==I2CZEROS || test==I2CBZEROS)
   for(reg=0; reg<dut_reg; reg++)
       t_vector[reg]=0;

if(test==I2CONES || test==I2CBONES)
   for(reg=0; reg<dut_reg; reg++)
       t_vector[reg]=0xff;

if(test==I2CCKBD)
   for(reg=0; reg<dut_reg; reg++)
       { if(((reg/16)%2)==0)
           t_vector[reg]=0x55;
         else
         t_vector[reg] = 0xaa;
       }

if(test==I2CICKBD)
   for(reg=0; reg<dut_reg; reg++)
       { if(((reg/16)%2)==0)
           t_vector[reg]=0xaa;
         else
           t_vector[reg]=0x55;
       }

if(test==I2C055)
   for(reg=0; reg<dut_reg; reg++)
       { t_vector[reg]=0x55;
       }

if(test==I2CBAA)
   for(reg=0; reg<dut_reg; reg++)
       { t_vector[reg]=0xaa;
       }

if(test==I2CINIVER || test==I2CINICHK)
    { t_vector[0]=0xa0;
      for(reg=1; reg<dut_reg; reg++)
         { t_vector[reg]=0;
         }
```

```
}

 if(test == ERAL || test == REGWRITE ||  test == WRALL1 ||  test == PROGREJC)
 for(reg = 0; reg < dut_reg; reg++)
     t_vector[reg] = 0xFF*256 + 0xFF;

 if(test == WRAL || test == WRALL0 || test == STRESS || test == REGWRITE0 || test == PROGREJ
 for(reg = 0; reg < dut_reg; reg++)
     t_vector[reg] = 0*256 + 0;

 if(test == INITCHK) {t_vector[0] = 0x01*256 + 0x01;}      /*  ID code = 0000 0001 0000 0001
                                                           /*            16        7        0

 if(test == INITVER)
 {
     t_vector[0] = 0x01*256 + 0x01;                        /*  ID code = 0000 0001 0000 0001
     for(reg = 1; reg < dut_reg; reg++)                    /*            16        7        0
             t_vector[reg] = 0*256 + 0;
 }

 if(test == CKBD || test == TCKBD)
 {
     if(true_dut == NMC9306)
     for(reg = 0; reg < dut_reg; reg++)
     {
         if(!(reg & 1)) t_vector[reg] = 0x55*256 + 0xAA;
         else
         t_vector[reg] = 0xAA*256 + 0x55;
     }
     else if(true_dut == NMC9346)
     for(reg = 0; reg < dut_reg; reg++)
     {
         if(!(reg & 1)) t_vector[reg] = 0x55*256 + 0x55;
         else
         t_vector[reg] = 0xAA*256 + 0xAA;
     }
     else if(dut_type == NM59C16 || dut_type == NM88C16)
     for(reg = 0; reg < dut_reg; reg++)
     {
         if(!(reg & 0x10)) t_vector[reg] = 0x55*256 + 0x55;
         else
         t_vector[reg] = 0xAA*256 + 0xAA;
     }
     else if(true_dut == CTRUE || true_dut == CSTRUE)
     for(reg = 0; reg < dut_reg; reg+=8)
     {
         t_vector[reg]   =   0xFF*256 + 0xFF;
         t_vector[reg+1] =   0x00*256 + 0x00;
         t_vector[reg+2] =   0xFF*256 + 0xFF;
         t_vector[reg+3] =   0x00*256 + 0x00;
         t_vector[reg+4] =   0x00*256 + 0x00;
         t_vector[reg+5] =   0xFF*256 + 0xFF;
         t_vector[reg+6] =   0x00*256 + 0x00;
         t_vector[reg+7] =   0xFF*256 + 0xFF;
     }
}

if(test == INVCKBD || test == TINVCKBD)
{
    if(true_dut == NMC9306)
```

```
    for(reg = 0; reg < dut_reg; reg++)
    {
        if(!(reg & 1)) t_vector[reg] = 0xAA*256 + 0x55;
        else
        t_vector[reg] = 0x55*256 + 0xAA;
    }
    else if(true_dut == NMC9346)
    for(reg = 0; reg < dut_reg; reg++)
    {
        if(!(reg & 1)) t_vector[reg] = 0xAA*256 + 0xAA;
        else
        t_vector[reg] = 0x55*256 + 0x55;
    }
    else if(dut_type == NM59C16 || dut_type == NM88C16)
    for(reg = 0; reg < dut_reg; reg++)
    {
        if(!(reg & 0x10)) t_vector[reg] = 0xAA*256 + 0xAA;
        else
        t_vector[reg] = 0x55*256 + 0x55;
    }
    else if(true_dut == CTRUE || true_dut == CSTRUE)
    for(reg = 0; reg < dut_reg; reg+=8)
    {
        t_vector[reg]   =   0x00*256 + 0x00;
        t_vector[reg+1] =   0xFF*256 + 0xFF;
        t_vector[reg+2] =   0x00*256 + 0x00;
        t_vector[reg+3] =   0xFF*256 + 0xFF;
        t_vector[reg+4] =   0xFF*256 + 0xFF;
        t_vector[reg+5] =   0x00*256 + 0x00;
        t_vector[reg+6] =   0xFF*256 + 0xFF;
        t_vector[reg+7] =   0x00*256 + 0x00;
    }
}
if(test == BITDIAG)
{
    if(true_dut == NMC9306)
    for(reg = 0; reg < dut_reg; reg++)
    {
        if(reg < 0x8) t_vector[reg] = b_mask[0x7 - reg]*256 + 0;
        if(reg > 0x7) t_vector[reg] = 0*256 + b_mask[reg - 0x8];
    }
    else if(true_dut == NMC9346)
    for(reg = 0; reg < dut_reg; reg++)
    {
        if(!(reg & 0x3)) t_vector[reg] = 0*256 + b_mask[reg%0x3];
        else
        t_vector[reg] = b_mask[reg%0x3]*256 + 0;
    }
    else if(dut_type == NM59C16 || dut_type == NM88C16)
    {
        for(reg = 0; reg < dut_reg; reg++) t_vector[reg] = 0xffff;

        for(reg = 0; reg < dut_reg; reg+=16)
        {
            t_vector[reg+3-next] = 0xffff & (~b_mask[(reg-256*next)/16]);
            t_vector[reg+7] =  t_vector[reg+3-next];
            t_vector[reg+11] = t_vector[reg+3-next];
            t_vector[reg+15] = t_vector[reg+3-next];

            done++;
```

```
            if(done == (16*(next+1)-1))    next++;
            else next=next;
        }
    }
    else if(true_dut == CTRUE || true_dut == CSTRUE)
    {
        for(reg = 0; reg < dut_reg; reg++) t_vector[reg] = 0xFFFF;

        if(dut_reg > 64)
        {
            for(reg = 0; reg < dut_reg; reg+=16)
            {
                t_vector[reg] = 0xFFFF & ~b_mask[reg/16];
                t_vector[reg+5] =  t_vector[reg];
                t_vector[reg+10] = t_vector[reg];
                t_vector[reg+15] = t_vector[reg];
            }
        }
        if(dut_reg < 128)
        {
            for(reg = 0; reg < dut_reg; reg+=16)
            {
                t_vector[reg] = 0xFFFF & ~b_mask_BIT;
                t_vector[reg+5] =  t_vector[reg];
                t_vector[reg+10] = t_vector[reg];
                t_vector[reg+15] = t_vector[reg];
                b_mask_BIT = _rotl(b_mask_BIT,1);
            }
        }
    }
}

if(test == INVBITDIAG)
{
    if (true_dut == NMC9306)
    for(reg = 0; reg < dut_reg; reg++)
    {
        if(reg < 0x8) t_vector[reg] = ~(b_mask[0x7 - reg]*256 + 0);
        if(reg > 0x7) t_vector[reg] = ~(0*256 + b_mask[reg - 0x8]);
    }
    else if(true_dut == NMC9346)
    for(reg = 0; reg < dut_reg; reg++)
    {
        if(!(reg & 0x8)) t_vector[reg] = ~(0*256 + b_mask[reg%0x8]);
        else
        t_vector[reg] = ~(b_mask[reg%0x8]*256 + 0);
    }
    else if(dut_type == NM59C16 || dut_type == NM88C16)
    {
        for(reg = 0; reg < dut_reg; reg++) t_vector[reg] = 0;

        for(reg = 0; reg < dut_reg; reg+=16)
        {
            t_vector[reg+3-next] = b_mask[(reg-256*next)/16];
            t_vector[reg+7] =  t_vector[reg+3-next];
            t_vector[reg+11] = t_vector[reg+3-next];
            t_vector[reg+15] = t_vector[reg+3-next];

            done++;
            if(done == (16 * (next+1)-1))  next++;
```

```
            else next=next;
        }
    }
    else if(true_dut == CTRUE || true_dut == CSTRUE)
    {
        for(reg = 0; reg < dut_reg; reg++) t_vector[reg] = 0;

        if(dut_reg > 64)
        {
            for(reg = 0; reg < dut_reg; reg+=16)
            {
                t_vector[reg] = b_mask[reg/16];
                t_vector[reg+5] = t_vector[reg];
                t_vector[reg+10] = t_vector[reg];
                t_vector[reg+15] = t_vector[reg];
            }
        }
        if(dut_reg < 128)
        {
            for(reg = 0; reg < dut_reg; reg+=16)
            {
                t_vector[reg] = b_mask_BIT;
                t_vector[reg+5] = t_vector[reg];
                t_vector[reg+10] = t_vector[reg];
                t_vector[reg+15] = t_vector[reg];
                b_mask_BIT = _rotl(b_mask_BIT,1);
            }
        }
    }
}

if(test == EVENBYTE)
for(reg = 0; reg < dut_reg; reg++)
    t_vector[reg] = (0xFF * !(reg & 1)) * 256 + 0xFF * !(reg & 1);

if(test == ODDBYTE)
for(reg = 0; reg < dut_reg; reg++)
    t_vector[reg] = (0xFF * (reg & 1)) * 256 + 0xFF * (reg & 1);

if(test == BYTEDIAG)
{
    if(dut_type == NM59C16 || dut_type == NM88C16)
    for(reg = 0; reg < dut_reg; reg++)
    {
        if(reg == next)
        {
            t_vector[reg]   =   0xFF*256 + 0xFF;
            next += 17;
        }
        else  t_vector[reg]   =   0*256 + 0;
    }
    else if(true_dut == CTRUE)
   {
        for(reg = 0; reg < dut_reg; reg++)
        {
            t_vector[reg] = 0*256 + 0;
            if(reg%5 == 0) t_vector[reg] = 0xFF*256 + 0xFF;
        }
    }
}
```

```
    if(test == RANDOM)
        for(reg = 0; reg < dut_reg;reg++)
                t_vector[reg]   =   p_vector[reg];
}

bitmap()
{
    register    int         reg;            /*  Register variables for speed        */

    int     mos,i;
    int     row,col,Vcc;
    char    pt[20];


    for(i = 0; i < MAX_FLOWS; i++)          /*  Reset all t[i].* stuff      */
    {
        t[i].name[0] = '\0';
        t[i].number  = '\0';
        t[i].bin     = '\0';
    }

    while (1)
    {
        run_stat = ENGMODE;

        window(1,1,80,25);
        clrscr();
        textcolor(YELLOW);
        highvideo();
        gotoxy(1,1);
        cprintf("ENGINEERING BITMAP");
        gotoxy(73,1);
        cprintf("(%s)",rev);
        lowvideo();
        textcolor(WHITE);
        window(1,2,80,25);

        mos=select_mos();                                       /* Screen #2 for chose
        if (mos == 0)  select_nmos_device();                    /* Screen #3
        if (mos == 1)  select_cmos_device();
        if (mos == 2)  select_i2c_device();
        select_board_type();                                    /* Screen #5, select B

        clrscr();
        gotoxy(20,10);
        cprintf("Enter the PT# : ");
        gets(pt);
        gotoxy(20,1);
        textcolor(YELLOW);
        highvideo();
        cprintf("DEVICE... %s",device);
        cprintf("    PT#... %s",pt);
        lowvideo();
        textcolor(WHITE);
        window(1,3,80,25);

        row=1;  col=1;
        Vcc=5000;

        for(i=0; i<16; i++)             /* Generate the bit masks      */
```

```
        b_mask[i] = (int)pow(2,i);    /* Coercion to int is required */

    while (1)
    {
        socket_stat = SCOFF;
        clrscr();
        gotoxy(20,5);   cprintf("Type [B]  to  Bitmap device");
        gotoxy(20,6);   cprintf("Type [P]  to  Program pattern");
        gotoxy(20,7);   cprintf("Type [D]  to  Display boardmap");
        gotoxy(20,8);   cprintf("Type [L]  to  Load a test program");
        gotoxy(20,9);   cprintf("Type [E]  to  Edit/create a custom pattern");
        gotoxy(20,10);  cprintf("Type [F]  to  load a custom pattern from File");
        gotoxy(20,11);  cprintf("Type [C]  to  Copy one dut data to another");
        gotoxy(20,12);  cprintf("Type [R]  to  Re-select device");
        gotoxy(20,13);  cprintf("Type [S]  to  Single Step device");
        gotoxy(20,14);  cprintf("Type [X]  to  eXit");
        gotoxy(20,18);  cprintf("Enter selection here:   ");

        textcolor(YELLOW);
        highvideo();
        gotoxy(1,20);   cprintf("%c",218);
                        for(i = 0; i<39; i++) cprintf("%c",196);    cprintf("%c",194);
                        for(i = 0; i<38; i++) cprintf("%c",196);    cprintf("%c",191);
        gotoxy(1,22);   cprintf("%c",192);
                        for(i = 0; i<39; i++) cprintf("%c",196);    cprintf("%c",193);
                        for(i = 0; i<38; i++) cprintf("%c",196);    cprintf("%c",217);

        gotoxy(1,21);   cprintf("%c          TOPOLOGICAL MAP = B          %c",179,179);
        gotoxy(42,21);  cprintf("        LOGICAL BITMAP = b          %c",179);
        lowvideo();
        textcolor(WHITE);
        gotoxy(45,18);
        ch=getch();

        if      (ch=='B' || ch=='b') map(&row,&col,&Vcc);
        else if(ch=='P' || ch=='p') pattern(&row,&col,&Vcc);
        else if(ch=='D' || ch=='d') display_board_map();
        else if(ch=='L' || ch=='l') load_test_program();
        else if(ch=='E' || ch=='e') random_pattern(&row,&col,&Vcc);
        else if(ch=='F' || ch=='f') load_random_file(&Vcc);
        else if(ch=='C' || ch=='c') copy_dut(&row,&col,&Vcc);
        else if(ch=='R' || ch=='r') break;
        else if(ch=='S' || ch=='s') single_step(&row,&col,&Vcc);
        else if(ch=='X' || ch=='x') return;
    }
  }
}

void   map(int *row, int *col, int *Vcc)
{
        int     i,j,k,reg,dn,co,di,dix;
        char    ctemp[5];

        if(ch == 'B')
        {
            switch(dut_type)
            {
                case    NM93C06:    bit_stat = TOPOLOGICAL; break;
                case    NMC93C06:   bit_stat = TOPOLOGICAL; break;
                case    NM93CS06:   bit_stat = TOPOLOGICAL; break;
```

```
        case    NM88CS06:   bit_stat = TOPOLOGICAL; break;
        case    NM88CS46:   bit_stat = TOPOLOGICAL; break;

        case    NM93C07:    bit_stat = TOPOLOGICAL; break;

        case    NM93C46:    bit_stat = TOPOLOGICAL; break;
        case    NMC93C46:   bit_stat = TOPOLOGICAL; break;
        case    NM93CS46:   bit_stat = TOPOLOGICAL; break;

        case    NM93C46A:   bit_stat = TOPOLOGICAL; break;
        case    NM59C11:    bit_stat = TOPOLOGICAL; break;

        case    NM93C56:    bit_stat = TOPOLOGICAL; break;
        case    NMC93C56:   bit_stat = TOPOLOGICAL; break;
        case    NM93CS56:   bit_stat = TOPOLOGICAL; break;

        case    NM93C66:    bit_stat = TOPOLOGICAL; break;
        case    NMC93C66:   bit_stat = TOPOLOGICAL; break;
        case    NM93CS66:   bit_stat = TOPOLOGICAL; break;
        case    NM95C12:    bit_stat = TOPOLOGICAL; break;

        default:    bit_stat = 0;
    }
}

for(i=0; i<4095; i++) bit[i]=' ';

if(dut_type == NM95C12) dut_reg=64;                 /*  Due to switch control in Reg63.

autoprint=0;
clrscr();
gotoxy(20,3);
cprintf("Current DUT location: (%d,%d), VCC = %d",*row,*col,*Vcc);
gotoxy(20,7);
cprintf("Change to new DUT location (y/n)? ");
if((ch=getch())=='y' || (ch=='Y'))
{   putch(ch);
    gotoxy(25,8);
    if(board_type==DIPB46 || board_type==I2C46 || board_type==DIPC44)
    cprintf("Enter col# (1..11): ");
    if(board_type==NM95C12DIPA)
    cprintf("Enter col# (1..7): ");
    if(board_type==SO8B || board_type==SOA || board_type==SO14A)
    cprintf("Enter col# (1..6): ");
    if(board_type==NM95C12SOICA || board_type==SO14WMA)
    cprintf("Enter col# (1..6): ");
    if(board_type==FLATPACK)
    cprintf("Enter col# (1..4): ");
    *row=atoi(gets(input));
    gotoxy(25,9);
    if(board_type==DIPB46 || board_type==I2C46)
    cprintf("Enter row# (1..46): ");
    if(board_type==DIPC44)
    cprintf("Enter row# (1..44): ");
    if(board_type==SO8B)
    cprintf("Enter row# (1..36): ");
    if(board_type==SOA)
    cprintf("Enter row# (1..34): ");
    if(board_type==NM95C12DIPA)
```

```
            cprintf("Enter row# (1..23): ");
            if(board_type==FLATPACK)
            cprintf("Enter row# (1..20): ");
            if(board_type==SO14A || board_type==NM95C12SOICA || board_type==SO14WMA)
            cprintf("Enter row# (1..18): ");
            *col=atoi(gets(input));
        }

        gotoxy(20,11);
        cprintf("Change VCC (y/n)? ");
        if((ch=getch())=='y'|| (ch=='Y'))
        {
            putch(ch);
            gotoxy(25,12);
            cprintf("Enter Vcc (mv): ");
            gets(input);
            i=atoi(input);
            if(i<2700) i=2700;
            *Vcc=i;
        }

        dn=1;
        gotoxy(20,14);
        cprintf("Auto mode (auto-increment row#)? y/n ");
        if((ch=getch())=='y'|| (ch=='Y'))
        {   putch(ch);
            gotoxy(25,15);
            cprintf("Enter #of parts in current col: ");
            dn=atoi(gets(ctemp));
            gotoxy(25,16);
            cprintf("Auto-print screen for each part? (y/n) ");
            if((ch=getch())=='y'||(ch=='Y')) autoprint=1;
        }

        set_vcc(*Vcc);

if(board_type==I2C46)
  { i2c_mode;
    di_dec((*row)/10*16 + (*row)%10);
    column((*col/10) * 16 + *col%10);
    outportb(C2,(*row)-1);

    startc;
    data_write(0xa0);
    ack_write;
    data_write(0);
    ack_write;
    startc;
    data_write(0xa1);
    ack_write;
    data_read;
    d_buffer[0]=inportb(82);

    for(reg=1; reg<dut_reg; reg++)
       { ack_read;
         data_read;
         d_buffer[reg]=inportb(82);
       }
    stop;
    um_mode;
```

```
    clrscr();

    i=2;
    for(reg=0; reg<dut_reg; reg++)
       { if((reg !=0) && (reg % 512 ==0))
           { i=2;
             gotoxy(25,23);
             cprintf("PRESS ANY KEY TO CONTINUE");
             getch();
             clrscr();
           }
         if(reg%32 == 0)
           {
             gotoxy(9,i++);
           }
         cprintf("%2x",d_buffer[reg]);

       }
    gotoxy(25,23);
    cprintf("PRESS ANY KEY TO CONTINUE");
    getch();
  }
set_vcc(0);
save_vcc();

if(board_type==I2C46)  return;

        if(dut_type == NM95C12)                 /*  Write all 0s to Register62 (Switch control)
        {
            column(0);                          /* column number */
            di_dec(0);

            reg = 62;
            reg_rotate=_rotl(reg,dut_offset);   /*  Rotates reg for DUT sending */

            cs_high;
            sk_toggle;
            start_bit;
            latch(write_send);
            clock(dut_opbits);
            latch(reg_rotate);
            clock(dut_adrbits);
            latch(0);
            clock(16);
            cs_low;
            delay(twp);
        }

        for(j=0;j<dn;j++)
        {
            dix=(*col)+j;
            co=(*col)+j;
            co=(co/10)*16 + co%10;                  /* Decode column #      */

            outportb(C2,(*row)-1);                  /* Row number           */

            if(board_type>0 && board_type<8)
              { column(co);
                di_dec(co);
              }
```

```
        if(board_type==DIPC44)
          { di=(4*(*row))-3;
            if(dix>0  && dix<12) di=di;
            if(dix>11 && dix<23) di=di+1;
            if(dix>22 && dix<34) di=di+2;
            if(dix>33 && dix<45) di=di+3;
            di=(di/10)*16 + di%10;
            column(co);                          /* Column number  */
            di_dec(di);
          }

        if(board_type==FLATPACK)
          { di=(5*(*row))-4;
            if(dix>0  && dix<5)  di=di;
            if(dix>4  && dix<9)  di=di+1;
            if(dix>8  && dix<13) di=di+2;
            if(dix>12 && dix<17) di=di+3;
            if(dix>16 && dix<21) di=di+4;
            di=(di/10)*16 + di%10;
            column(co);                         /* Column number  */
            di_dec(di);
          }

        read_dut_data();

        if(bit_stat == TOPOLOGICAL)                  /*  Topological bitmap          */
        {
            for (k=0; k<(dut_reg/4); k++)            /*  Set Row count = dut_reg/4   */
            {
                reg=(4*k);
                for(i=0; i<16; i++)                  /*  Go thru 16 bits             */
                {
                    if ((d_buffer[reg]   & b_mask[i]) != 0) bit[  (4*i)+(k*64)]='1';
                    if ((d_buffer[reg+1] & b_mask[i]) != 0) bit[1+(4*i)+(k*64)]='1';
                    if ((d_buffer[reg+2] & b_mask[i]) != 0) bit[2+(4*i)+(k*64)]='1';
                    if ((d_buffer[reg+3] & b_mask[i]) != 0) bit[3+(4*i)+(k*64)]='1';
                }
            }
            topological_display_bit_map(*row,(*col)+j,*Vcc);
        }
        else                                         /*  Logical bitmap      */
        {
            for (reg=0,k=0; reg<dut_reg; reg++)
            for(i=0;i<16;i++,k++)
            {
                if ((d_buffer[reg] & b_mask[i]) == 0) bit[k]=' ';
                else bit[k]='1';
            }
            logical_display_bit_map(*row,(*col)+j,*Vcc);
        }
    }

    if(dut_type == NM95C12) dut_reg=62;              /*  Due to switch control in Reg63*/
    set_vcc(0);
    save_vcc();
    bit_stat = 0;
}
/*************************************************/
```

```
/*            m5.c                */
#include <stdio.h>
#include <dos.h>
#include <conio.h>
#include <dir.h>
#include <math.h>
#include <io.h>
#include <sys/stat.h>
#include <time.h>
#include <string.h>
#include <\mill\variable.h>
#include <\mill\function.h>

/*************************************************/
void   single_step(int *row, int *col, int *Vcc)
{
    int          i,j,k,co,di;
    int          SK_count;                    /*  SK counter                  */

    unsigned     int          temp = 0;
    register     int          reg;           /*  Register variables for speed */

    char         ctemp[5];

    step_stat = 0;                            /*  Set flag for manual bit entry */

    clrscr();
    gotoxy(20,3);    cprintf("Current DUT location: (%d,%d), VCC = %d",*row,*col,*Vcc);
    gotoxy(20,5);    cprintf("Change to new DUT location (y/n)? ");
    if((ch=getch())=='y' || (ch=='Y'))
    {
        putch(ch);
        gotoxy(25,6);
        if(board_type==DIPB46 || board_type==I2C46 || board_type==DIPC44)
        gotoxy(25,6);    cprintf("Enter col# (1..11): ");
        if(board_type==NM95C12DIPA)
        cprintf("Enter col# (1..7): ");
        if(board_type==SO8B || board_type==SOA || board_type==SO14A)
        cprintf("Enter col# (1..6): ");
        if(board_type==NM95C12SOICA || board_type==SO14WMA)
        cprintf("Enter col# (1..6): ");
        if(board_type==FLATPACK)
        cprintf("Enter col# (1..4): ");
        *row=atoi(gets(input));
        gotoxy(25,7);
        if(board_type==DIPB46 || board_type==I2C46)
        cprintf("Enter row# (1..46): ");
        if(board_type==DIPC44)
        cprintf("Enter row# (1..44): ");
        if(board_type==SO8B)
        cprintf("Enter row# (1..36): ");
        if(board_type==SOA)
        cprintf("Enter row# (1..34): ");
        if(board_type==NM95C12DIPA)
        cprintf("Enter row# (1..23): ");
        if(board_type==FLATPACK)
        cprintf("Enter row# (1..20): ");
        if(board_type==SO14A || board_type==NM95C12SOICA || board_type==SO14WMA)
        cprintf("Enter row# (1..18): ");
```

```
    *col=atoi(gets(input));

    gotoxy(25,6);   cprintf("                    ");          /*  Clear line  */
    gotoxy(25,7);   cprintf("                    ");          /*  Clear line  */
}
co=(*col/10)*16 + *col%10;                      /* Decode column &      */
outportb(C2,(*row)-1);                          /*  Row number           */

if(board_type>0 && board_type<3)
  { column(co);
    di_dec(co);
  }

if(board_type==DIPC44)
  { di=(4*(*row))-3;
    if(*col>0  && *col<12) di=di;
    if(*col>11 && *col<23) di=di+1;
    if(*col>22 && *col<34) di=di+2;
    if(*col>33 && *col<45) di=di+3;
    di=(di/10)*16 + di%10;
    column(co);
    di_dec(di);
  }

if(board_type==FLATPACK)
  { di=(5*(*row))-4;
    if(*col>0  && *col<5)  di=di;
    if(*col>4  && *col<9)  di=di+1;
    if(*col>8  && *col<13) di=di+2;
    if(*col>12 && *col<17) di=di+3;
    if(*col>16 && *col<21) di=di+4;
    di=(di/10)*16 + di%10;
    column(co);
    di_dec(di);
  }

gotoxy(20,5);   cprintf("                                   "); /*  Clear line  */
gotoxy(20,5);   cprintf("Change VCC (y/n)? ");
if((ch=getch())=='y'|| (ch=='Y'))
{
    putch(ch);
    gotoxy(25,6);   cprintf("Enter Vcc (mv): ");
    i=atoi(gets(input));
    if(i<2700) i=2700;
    *Vcc=i;
}
set_vcc(*Vcc);

gotoxy(20,3);
cprintf("Current DUT location: (%d,%d), VCC = %d",*row,*col,*Vcc);

if(dut_type == NM95C12)                 /*  Write all 0s to Register62 (Switch control) */
{
    column(0);                          /* Set column number for whole board    */
    di_dec(0);

    reg = 62;
    reg_rotate=_rotl(reg,dut_offset);   /*  Rotates reg for DUT sending         */

    cs_high;
```

```
      sk_toggle;
      start_bit;
      latch(write_send);
      clock(dut_opbits);
      latch(reg_rotate);
      clock(dut_adrbits);
      latch(0);
      clock(16);
      cs_low;
      delay(twp);

      co=(*col/10)*16 + *col%10;                    /* Decode column #      */

      if(board_type>0 && board_type<8)
        { column(co);
          di_dec(co);
        }

      if(board_type==DIPC44)
        { di=(4*(*row))-3;
          if(*col>0  && *col<12) di=di;
          if(*col>11 && *col<23) di=di+1;
          if(*col>22 && *col<34) di=di+2;
          if(*col>33 && *col<45) di=di+3;
          di=(di/10)*16 + di%10;
          column(co);
          di_dec(di);
        }

      if(board_type==FLATPACK)
        { di=(5*(*row))-4;
          if(*col>0  && *col<5)  di=di;
          if(*col>4  && *col<9)  di=di+1;
          if(*col>8  && *col<13) di=di+2;
          if(*col>12 && *col<17) di=di+3;
          if(*col>16 && *col<21) di=di+4;
          di=(di/10)*16 + di%10;
          column(co);
          di_dec(di);
        }

}

gotoxy(10,5);   cprintf("%c",218);
                for(i = 0; i<26; i++) cprintf("%c",196);     cprintf(" CLOCK CYCLES ");
                for(i = 0; i<26; i++) cprintf("%c",196);     cprintf("%c",191);

gotoxy(10,6);   cprintf("%c",179);
gotoxy(22,6);   for(j=1; j<6; j++)
                for(i=0; i<10; i++)  cprintf("%d",j);
                cprintf("%d%d%d%d",6,6,6,6);
gotoxy(77,6);   cprintf("%c",179);

gotoxy(10,7);   cprintf("%c",179);
gotoxy(12,7);   for(j=0; j<6; j++)
                for(i=0; i<10; i++) cprintf("%d",i);
                cprintf("%d%d%d%d",0,1,2,3);
gotoxy(77,7);   cprintf("%c",179);

gotoxy(10,8);   cprintf("%c",192);
```

บทที่ 7

ผลการทดลอง

7.1 INTERFACE BOARD

วงจรชุดนี้ จะเป็นวงจรที่ต่อเชื่อม LOGIC CONTROL BOARD กับ COMPUTER ซึ่งจะส่งผ่านและรับสัญญาณทุกอย่างให้กันและกัน โดยใช้ PROGRAMMABLE PORT 8255A โดยมี PORT ต่อใช้งานทั้งหมด 3 PORT ดังนี้

PORT 1 ใช้เป็น PORT ต่อสัญญาณควบคุมและสัญญาณเลือกการทำงาน CS หรือ DI เป็นกลุ่มและสัญญาณที่แสดงผลที่ FRONT PANEL

PORT 2 ใช้เป็น PORT ต่อสัญญาณ DO,DI,CS และ SK

PORT 3 ใช้เป็น PORT สำหรับควบคุมการทำงานของ PROGRAMMING POWER SUPPLY

ขั้นตอนการทดลองวงจร

1 เปิดเครื่องให้วงจรทำงาน ตอนนี้สัญญาณที่ส่งจาก COMPUTER จะเป็น LOGIC"LOW" ทุก BIT ดังนั้น OUTPUT ของ PORT จะเป็น"LOW"หลังจากผ่าน BUFFER OUTPUT ทุก PIN ของ PORT ทั้ง 3 PORTS จะเป็น LOGIC LOW ทุกบิท

2 ให้สมมุติว่า เราต้องการให้ DUT ในตำแหน่งที่ 1 ทำงานซึ่ง DUT ในตำแหน่งที่ 1จะต่ออยู่กับสัญญาณ CS1 และสัญญาณ DI1 ดังนั้น COMPUTER จะส่งสัญญาณเลือก DI/CS ตาม PATTERN โดยการ OUT PORT 2A BIT 0 ให้เป็น HIGH (DI0) และ OUT PORT 1A BIT 0 ให้ HIGH (CS0) การสั่งดังนี้เป็นการเลือก UNIT ตำแหน่งที่ 1 บน LOADBOARD

7.2 LOGIC CONTROL BOARD

วงจรนี้เป็น HARDWARE ซึ่งเลือกสัญญาณที่จะส่งไปให้ยัง DUT BOARD และรับสัญญาณจาก DUT BOARD แล้วส่งกลับไปวัดที่ COMPUTER

ขั้นตอนการทดสอบวงจร

1 ขณะเปิดเครื่องให้ทำงาน LOGIC ที่ DI และ CS ทุกสัญญาณเป็น LOW DUTทุกตัวไม่ทำงานนอกจากมีไฟ SUPPLY จ่ายให้โดยผ่าน SWITCH VCC DUT โดย DUT SUPPLY นี้จะเป็น PROGRAMMING POWER SUPPLY

2 สมมุติว่าเราต้องการให้ DUT ที่ตำแหน่งที่ 1 ทำงาน ซึ่ง DUT ตำแหน่งที่ 1 ต่ออยู่กับสัญญาณ DI1,DO1 และ CS1 ดังนั้นคำสั่งเลือกสัญญาณ DI และ CS จะถูกส่งออกมาโดย COMPUTER ด้วย PATTERN

DI/CS 1A HIGH

DI/CS 2A LOW

DI/CS 4A LOW

DI/CS 8A LOW

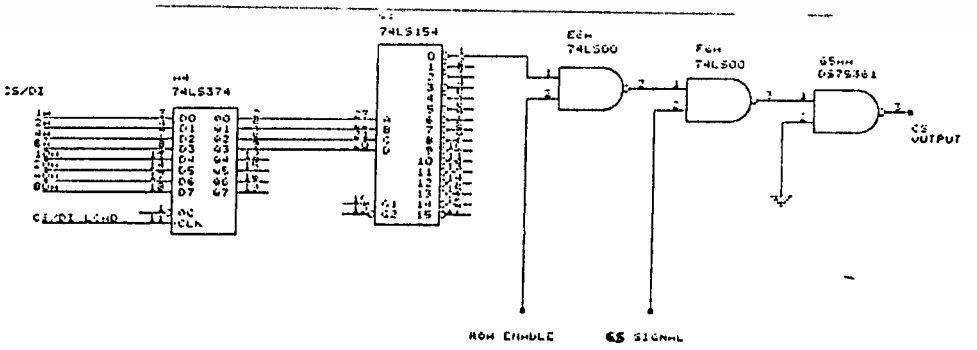
DI/CS 10A LOW

DI/CS 20A LOW

DI/CS 40A LOW

DI/CS 80A LOW

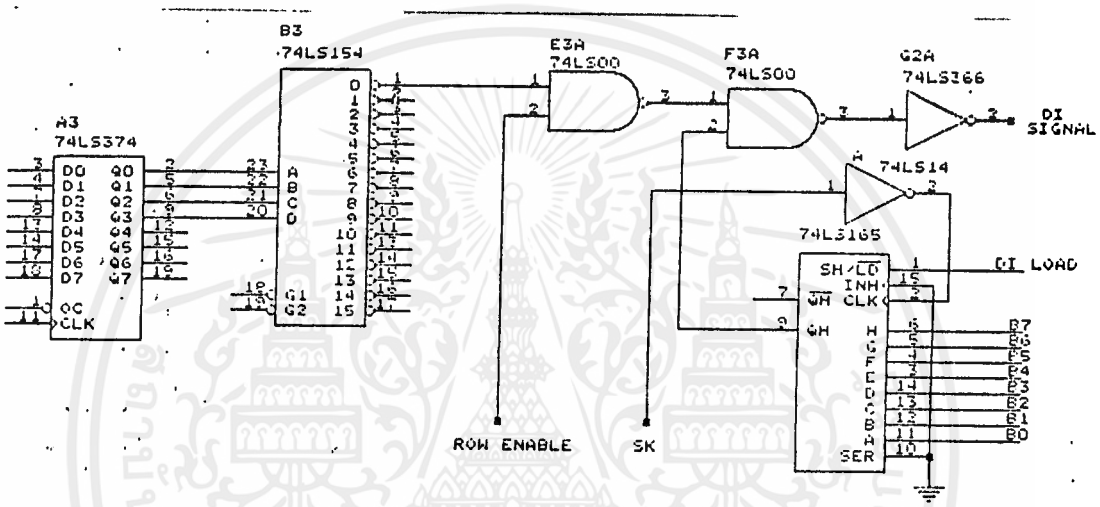
สัญญาณจะถูก LATCH โดยสัญญาณ CS/DI CLK จะเปลี่ยนจาก LOW เป็น HIGH PATTERN ทั้งหมดจะ OUT ออกไปยัง IC DECODER DM74LS154 LOC C3 จะOUT LOGIC LOW ที่ขา 2 ส่วนขาอื่นๆจะเป็น HIGH ทั้งหมด LOGIC LOW นี้จะทำให้ OUTPUT ของ NAND GATE ที่ LOC E6 ขา3



CS SIGNAL

เป็น HIGH ซึ่งจะทำให้สัญญาณ DI SIGNAL OUT ออกมาที่ชุด DRIVER และจ่ายออกไปยัง DUT ที่ตำแหน่งที่ 1 สัญญาณ CS จะเกิดจากการเปลี่ยน สเตทของ LOGICใน SOFTWARE

ในทำนองเดียวกัน สัญญาณ DI 1 ก็จะ OUTออกดังรูป



DI SIGNAL

7.3 DUT BOARD

เป็น BOARD ใช้ใส่ UNIT ที่จะทำการ TEST

การทดสอบวงจร

RUN PROGRAM โดยให้ทำการ PROGRAM หมดทั้ง BOARD แล้วใช้ OSCILLOSCOPE ก็จะได้ TIMING PROGRAM ตามที่กำหนดไว้ใน DATA SHEET ของ DEVICE EEPROM

รายการอุปกรณ์

1 รายการอุปกรณ์ของวงจร INTERFACE BOARD

INTEGRATED CIRCUIT

A1 DM74LS244N

A2 DM74LS244N

B1 8255A

C1 8255A

D1 DM74LS244N

D2 DM74LS244N

E1 DM74LS244N

E2 DM74LS244N

F1 DM74LS138N

F2 8255A

G1 DM74LS244N

G2 DM74LS244N

2 รายการอุปกรณ์ของวงจร LOGIC CONTROL BOARD

INTEGRATED CIRCUIT

A1 DM74LS138N

A3 DM74LS374N

A4 DM74LS374N

B1 DM74LS154N

B2 DM74LS154N

B3 DM74LS154N

C1 DM74LS154N

C2 DM74LS154N

C3 DM74LS154N

E1-E6 DM74LS00N

F1-F6 DM74LS00N

G1-G2 DM74LS366AN

G3-G5 DS75361N

H1 DM74LS151N

H2 DM74LS244N

I1 DM74LS366AN

I2 DM74LS244N

J1 DM74LS164N

J2 DM74LS240N

K1 DM74LS165N

K2 DS3686N

RESISTOR

R1 10K 1/4 5%

R2 10K 1/4 5%

R3 10K 1/4 5%

7.3 DATA SHEETS

National Semiconductor

NMC93C56/C66 2048-Bit/4096-Bit Serial Electrically Erasable Programmable Memories

General Description

The NMC93C56/NMC93C66 are 2048/4096 bits of CMOS electrically erasable memory divided into 128/256 16-bit registers. They are fabricated using National Semiconductor's floating-gate CMOS process for high speed and low power. They operate from a single 5V supply since V_{PP} is generated on-board. The serial organization allow the NMC93C56/66 to be packaged in an 8-pin DIP or 14-pin SO package to save board space.

The memories feature a serial interface with the instruction, address, and write data, input on the Data-In (DI) pin. All read data and device status come out on the Data-Out (DO) pin. A low-to-high transition of shift clock (SK) shifts all data in and out. This serial interface is MICROWIRE™ compatible for simple interface to standard microcontrollers and microprocessors. There are 7 instructions: Read, Erase/Write Enable, Erase, Erase All, Write, Write All, and Erase/Write Disable. The NMC93C56/66 do not require an erase cycle prior to the Write and Write All instructions. The Erase and Erase All instructions are available to maintain complete read and programming capability with the NMOS NMC9346. All programming cycles are completely self-timed for simplified operation. The busy status is available on the DO pin to indicate the completion of a programming cycle. EEPROMs are shipped in the erased state where all bits are logical 1's.

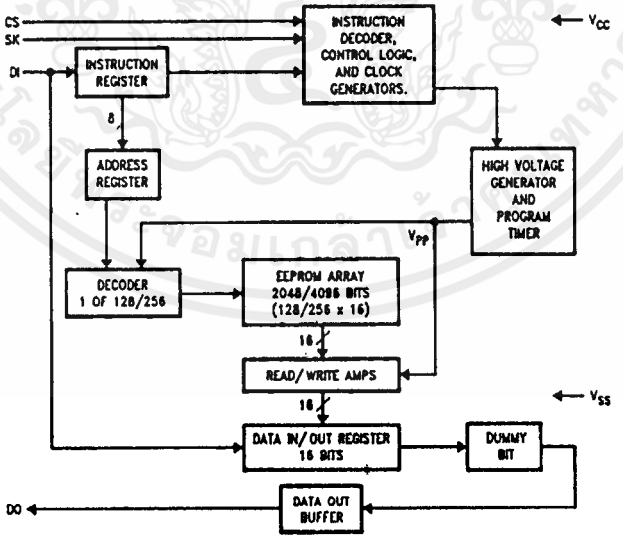
Compatibility with Other Devices

These memories are pin compatible to National Semiconductor's NMOS EEPROMs, NMC9306 and NMC9346 and CMOS EEPROMs NMC93C06/46. The NMC93C56/66 are both pin and function compatible with the NMC93C06/46, 256/1024-bit EEPROM with the one exception that the NMC93C56/66 require 2 additional address bits.

Features

- Typical active current 400 μA; Typical standby current 25 μA
- Reliable CMOS floating gate technology
- 5V only operation in all modes
- MICROWIRE compatible serial I/O
- Self-timed programming cycle
- Device status signal during programming mode
- Sequential register read
- Over 40 years data retention
- Designed for 100,000 write cycles

Block Diagram



TL/D/9617-1

Absolute Maximum Ratings (Note 1)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Ambient Storage Temperature	−65°C to +150°C
All Input or Output Voltages with Respect to Ground	+6.5V to −0.3V
Lead Temp. (Soldering, 10 sec.)	+300°C
ESD Rating	2000V

Operating Conditions

Ambient Operating Temperature	
NMC93C56/NMC93C66	0°C to +10°C
NMC93C56E/NMC93C66E	−40°C to +85°C
NMC93C56M/NMC93C66M (Mil. Temp.)	−55°C to +125°C
Positive Power Supply	4.5V to 5.5V

DC and AC Electrical Characteristics V_{CC} = 5V ± 10% (unless otherwise specified)

Symbol	Parameter	Part Number	Conditions	Min	Max	Units
I_{CC1}	Operating Current CMOS Input Levels	NMC93C56/NMC93C66 NMC93C56E/NMC93C66E NMC93C56M/NMC93C66M*	CS = V_{IH}, SK = 1 MHz SK = 0.5 MHz SK = 0.5 MHz		2 2 2	mA
I_{CC2}	Operating Current TTL Input Levels	NMC93C56/NMC93C66 NMC93C56E/NMC93C66E NMC93C56M/NMC93C66M	CS = V_{IH}, SK = 1 MHz SK = 0.5 MHz SK = 0.5 MHz		3 3 4	mA
I_{CC3}	Standby Current	NMC93C56/NMC93C66 NMC93C56E/NMC93C66E NMC93C56M/NMC93C66M	CS = 0V		50 100 100	µA
I_{IL}	Input Leakage	NMC93C56/NMC93C66 NMC93C56E/NMC93C66E NMC93C56M/NMC93C66M	V_{IN} = 0V to V_{CC}	−2.5 −10 −10	2.5 10 10	µA µA
I_{OL}	Output Leakage	NMC93C56/NMC93C66 NMC93C56E/NMC93C66E NMC93C56M/NMC93C66M	V_{IN} = 0V to V_{CC}	−2.5 −10 −10	2.5 10 10	µA µA
V_{IL} V_{IH}	Input Low Voltage Input High Voltage			−0.1 2	0.8 V_{CC} + 1	V V
V_{OL1}	Output Low Voltage	NMC93CS56/NMC93C566 NMC93CS56E/NMC93C566E NMC93CS56M/NMC93C566M	I_{OL} = 2.1 mA I_{OL} = 2.1 mA I_{OL} = 1.8 mA		0.4 0.4 0.4	V
V_{OH1}	Output High Voltage		I_{OH} = 400 µA	2.4		V
V_{OL2} V_{OH2}	Output Low Voltage Output High Voltage		I_{OL} = 10 µA I_{OH} = −10 µA	 V_{CC} − 0.2	0.2	V V
f_{SK}	SK Clock Frequency	NMC93C56/NMC93C66 NMC93C56E/NMC93C66E NMC93C56M/NMC93C66M		0 0 0	1 0.5 0.5	MHz
t_{SKH}	SK High Time	NMC93C56/NMC93C66 NMC93C56E/NMC93C66E NMC93C56M/NMC93C66M	(Note 2) (Note 3) (Note 3)	250 500 500		ns
t_{SKL}	SK Low Time	NMC93C56/NMC93C66 NMC93C56E/NMC93C66E NMC93C56M/NMC93C66M	(Note 2) (Note 3) (Note 3)	250 500 500		ns
t_{CS}	Minimum CS Low Time	NMC93C56/NMC93C66 NMC93C56E/NMC93C66E NMC93C56M/NMC93C66M	(Note 4) (Note 5) (Note 5)	250 500 500		ns
t_{CSS}	CS Setup Time	NMC93C56/NMC93C66 NMC93C56E/NMC93C66E NMC93C56M/NMC93C66M	Relative to SK	50 100 100		ns

*Note: Throughout this table "M" refers to temperature range (−55°C to +125°C), not package type.

DC and AC Electrical Characteristics V_{CC} = 5V ±10% (unless otherwise specified) (Continued)

Symbol	Parameter	Part Number	Conditions	Min	Max	Units
t_{DIS}	DI Setup Time	NMC93C56/NMC93C66 NMC93C56E/NMC93C66E NMC93C56M/NMC93C66M	Relative to SK	100 200 200		ns
t_{CSH}	CS Hold Time		Relative to SK	0		ns
t_{DIH}	DI Hold Time	NMC93C56/NMC93C66 NMC93C56E/NMC93C66E NMC93C56M/NMC93C66M	Relative to SK	100 200 200		ns
t_{PD1}	Output Delay to "1"	NMC93C56/NMC93C66 NMC93C56E/NMC93C66E NMC93C56M/NMC93C66M	AC Test		500 1000 1000	ns
t_{PD0}	Output Delay to "0"	NMC93C56/NMC93C66 NMC93C56E/NMC93C66E NMC93C56M/NMC93C66M	AC Test		500 1000 1000	ns
t_{SV}	CS to Status Valid	NMC93C56/NMC93C66 NMC93C56E/NMC93C66E NMC93C56M/NMC93C66M	AC Test		500 1000 1000	ns
t_{DF}	CS to DO in TRI-STATE®	NMC93C56/NMC93C66 NMC93C56E/NMC93C66E NMC93C56M/NMC93C66M	AC Test CS = V_{IL}		100 200 200	ns
t_{WP}	Write Cycle Time				10	ms

Note 1: Stress above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of the specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Note 2: The SK frequency specification for Commercial parts specifies a minimum SK clock period of 1 μs, therefore in an SK clock cycle t_{SKH} + t_{SKL} must be greater than or equal to 1 μs. For example if t_{SKL} = 250 ns then the minimum t_{SKH} = 750 ns in order to meet the SK frequency specification.

Note 3: The SK frequency specification for Extended Temperature and Military parts specifies a minimum SK clock period of 2 μs, therefore in an SK clock cycle t_{SKH} + t_{SKL} must be greater than or equal to 2 μs. For example, if the t_{SKL} = 500 ns then the minimum t_{SKH} = 1.5 μs in order to meet the SK frequency specification.

Note 4: For Commercial parts CS must be brought low for a minimum of 250 ns (t_{CS}) between consecutive instruction cycles.

Note 5: For Extended Temperature and Military parts CS must be brought low for a minimum of 500 ns (t_{CS}) between consecutive instruction cycles.

Note 6: This parameter is periodically sampled and not 100% tested.

Capacitance (Note 6)

T_A = 25°C f = 1 MHz

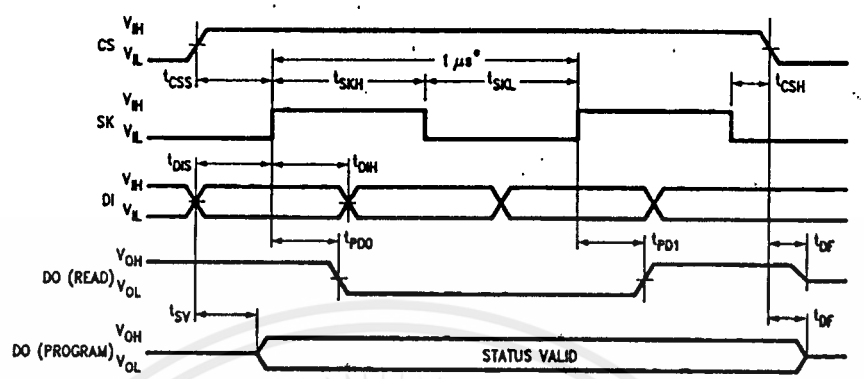
Symbol	Test	Typ	Max	Units
C_{OUT}	Output Capacitance		5	pF
C_{IN}	Input Capacitance		5	pF

AC Test Conditions

Output Load	1 TTL Gate and C_L = 100 pF
Input Pulse Levels	0.4V to 2.4V
Timing Measurement Reference Level	
Input	1V and 2V
Output	0.8V and 2V

Timing Diagrams

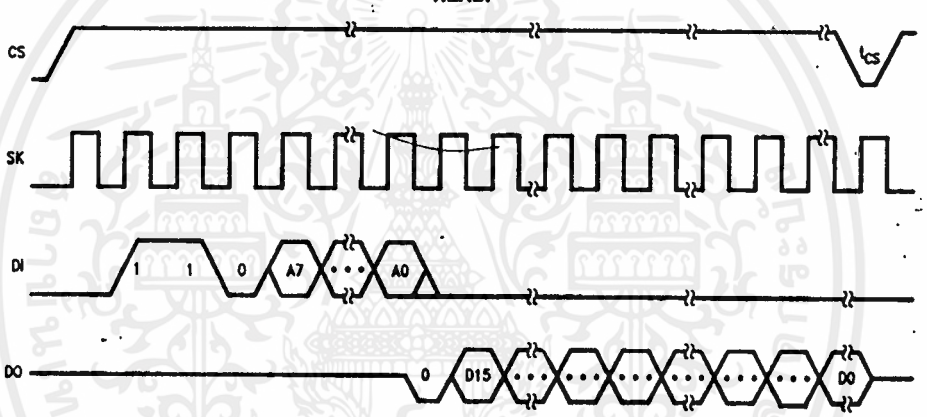
Synchronous Data Timing



TL/D/9617–4

*This is the minimum SK period (Note 2).

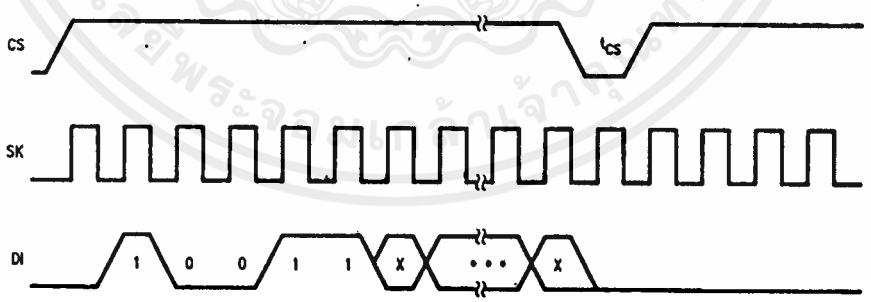
READ:



TL/D/9617–5

*Address bit A_7 becomes a "don't care" for NMC93C56.

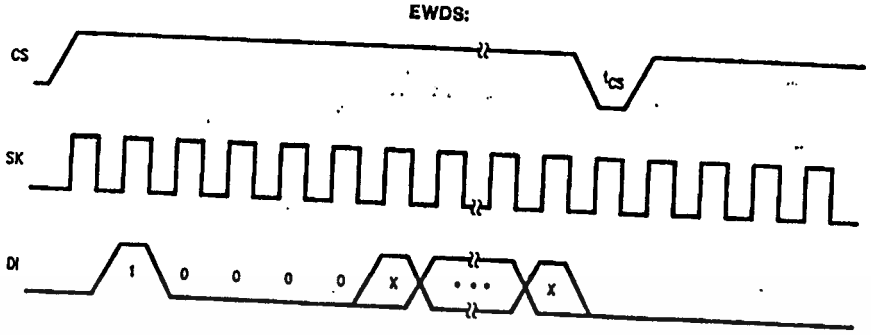
EWEN:



TL/D/9617–6

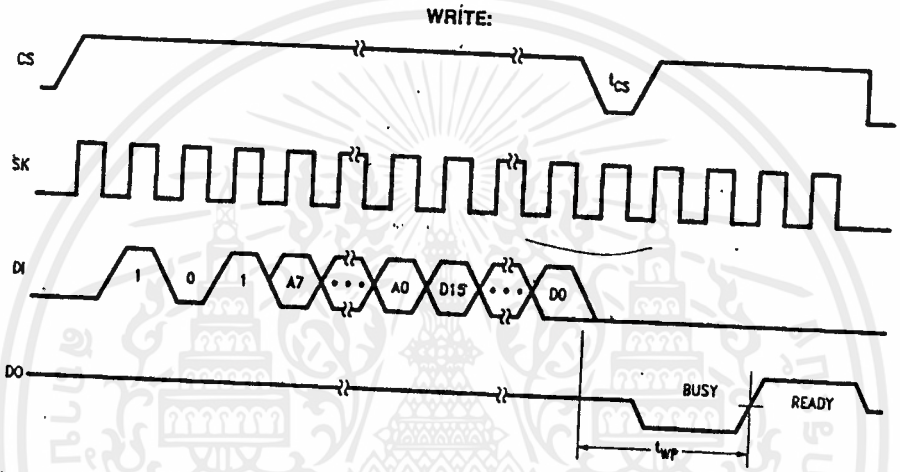
Timing Diagrams (Continued)

EWDS:



TL/D/9617–7

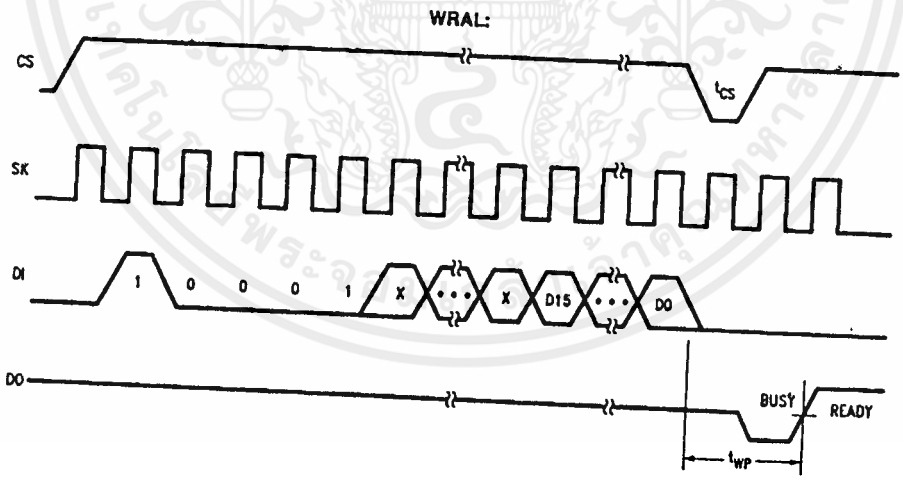
WRITE:



*Address bit A_7 becomes a "don't care" for NMC93C56.

TL/D/9617–8

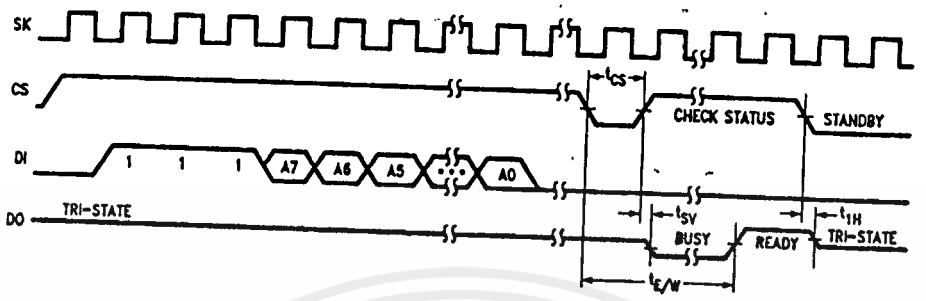
WRAL:



TL/D/9617–9

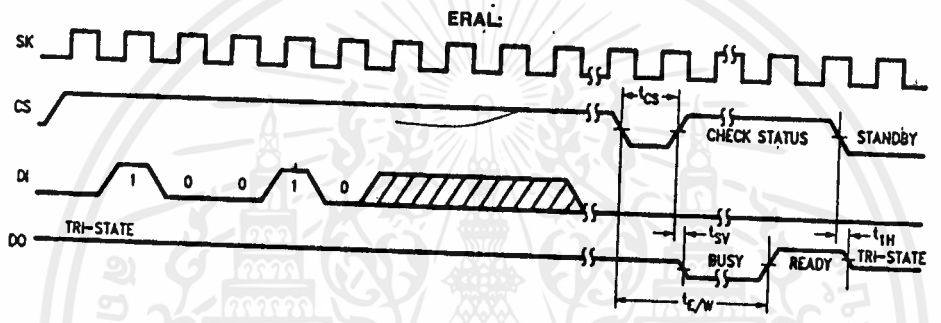
Timing Diagrams (Continued)

ERASE:



TL/D/9617–10

ERAL:



TL/D/9617–11

National Semiconductor

NMC93CS56/CS66 2048-Bit/4096-Bit Serial Electrically Erasable Programmable Memories

General Description

The NMC93CS56/NMC93CS66 are 2048/4096 bits of read/write memory divided into 128/256 registers of 16 bits each. N registers (N ≤ 128 or N ≤ 256) can be protected against data modification by programming into a special on-chip register, called the memory "protect register", the address of the first register to be protected. This address can be "locked" into the device, so that these registers can be permanently protected. Thereafter, all attempts to alter data in a register whose address is equal to or greater than the address stored in the "protect register" will be aborted.

The "read" instruction loads the address of the first register to be read into an 8-bit address pointer. Then the data is clocked out serially on the "DO" pin and automatically cycles to the next register to produce a serial data stream. In this way the entire memory can be read in one continuous data stream or as registers of varying length from 16 to 2048/4096 bits. Thus, the NMC93CS56/NMC93CS66 can be viewed as a non-volatile shift register.

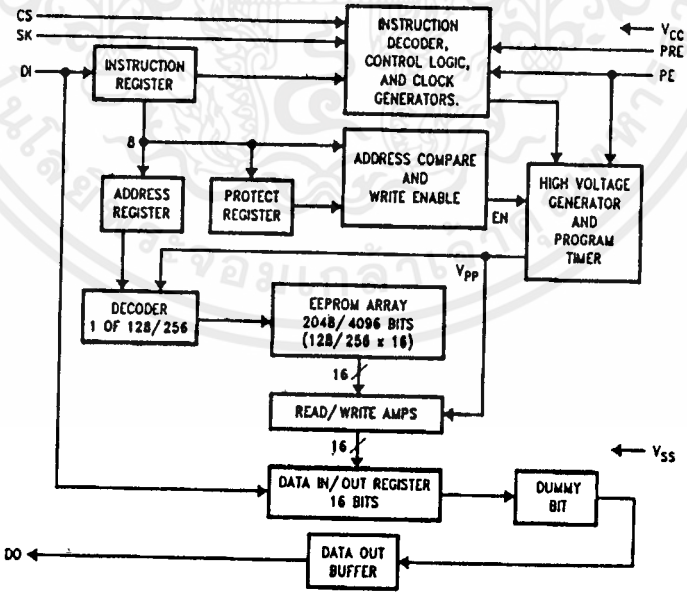
The "write" cycle is completely self-timed. No separate erase cycle is required before write. The "write" cycle is only enabled when pin 6 (program enable) is held "high". If the address of the register to be written is less than the address in the "protect register" then the data is written 16 bits at a time into one of the 128/256 data registers. If "CS" is brought "high" following the initiation of a "write" cycle, the "DO" pin indicates the ready/busy status of the chip.

National Semiconductor's EEPROMs are designed and tested for applications requiring extended endurance. Refer to device operation for further endurance information. Data retention is specified to be greater than 40 years.

Features

- Write protection in user defined section of memory
- Typical active current 400 μA; Typical standby current 25 μA
- Reliable CMOS floating gate technology
- 5 volt only operation in all modes
- MICROWIRE compatible serial I/O
- Self-timed programming cycle
- Device status signal during programming mode
- Sequential register read
- Over 40 years data retention
- Designed for 100,000 write cycles

Block Diagram

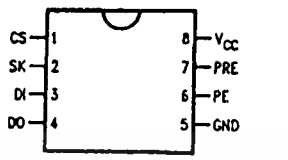


TL/D/9209-3

Connection Diagrams

PIN OUT:

Dual-In-Line Package (N)



TL/D/9209–1

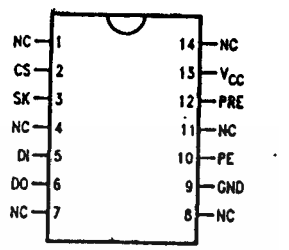
Top View

See NS Package Number N08E

Pin Names	
CS	Chip Select
SK	Serial Data Clock
DI	Serial Data Input
DO	Serial Data Output
GND	Ground
PE	Program Enable
PRE	Protect Register Enable
V_{CC}	Power Supply

PIN OUT:

SO Package (M)



TL/D/9209–2

Top View

See NS Package Number M14A

Ordering Information

Commercial Temp. Range (0°C to +70°C)

Order Number
NMC93CS56N/NMC93CS66N
NMC93CS56M/NMC93CS66M

Extended Temp. Range (−40°C to +85°C)

Order Number
NMC93CS56EN/NMC93CS66EN
NMC93CS56EM/NMC93CS66EM

Military Temp. Range (−55°C to +125°C)

Order Number
NMC93CS56MN/NMC93CS66MN
NMC93CS56MM/NMC93CS66MM

Absolute Maximum Ratings (Note 1)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Ambient Storage Temperature	−65°C to +150°C
All Input or Output Voltages with Respect to Ground	+6.5V to −0.3V
Lead Temp. (Soldering, 10 sec.)	+300°C
ESD rating	2000V

Operating Conditions

Ambient Storage Temperature	
NMC93CS56/MNC93CS66	0°C to +70°C
NMC93CS56E/NMC93CS66E	−40°C to +85°C
NMC93CS56M/NMC93CS66M (Mil. Temp.),	−55°C to +125°C
Positive Power Supply	4.5V to 5.5V

DC and AC Electrical Characteristics V_{CC} = 5V ±10% (unless otherwise specified)

Symbol	Parameter	Part Number	Conditions	Min	Max	Units
I_{CC1}	Operating Current CMOS Input Levels	NMC93CS56/NMC93CS66 NMC93CS56E/NMC93CS66E NMC93CS56M/NMC93CS66M*	CS = V_{IH}, SK = 1 MHz SK = 0.5 MHz SK = 0.5 MHz		2 2 2	mA
I_{CC2}	Operating Current TTL Input Levels	NMC93CS56/NMC93CS66 NMC93CS56E/NMC93CS66E NMC93CS56M/NMC93CS66M	CS = V_{IH}, SK = 1 MHz SK = 0.5 MHz SK = 0.5 MHz		3 3 4	mA
I_{CC3}	Standby Current	NMC93CS56/NMC93CS66 NMC93CS56E/NMC93CS66E NMC93CS56M/NMC93CS66M	CS = 0V		50 100 100	µA
I_{IL}	Input Leakage	NMC93CS56/NMC93CS66 NMC93CS56E/NMC93CS66E NMC93CS56M/NMC93CS66M	V_{IN} = 0V to V_{CC}	−2.5 −10 −10	2.5 10 10	µA µA
I_{OL}	Output Leakage	NMC93CS56/NMC93CS66 NMC93CS56E/NMC93CS66E NMC93CS56M/NMC93CS66M	V_{OUT} = 0V to V_{CC}	−2.5 −10 −10	2.5 10 10	µA µA
V_{IL} V_{IH}	Input Low Voltage Input High Voltage			−0.1 2	0.8 V_{CC} + 1	V V
V_{OL1}	Output Low Voltage	NMC93CS56/NMC93CS66 NMC93CS56E/NMC93CS66E NMC93CS56M/NMC93CS66M	I_{OL} = 2.1 mA I_{OL} = 2.1 mA I_{OL} = 1.8 mA		0.4 0.4 0.4	V
V_{OH1}	Output High Voltage		I_{OH} = −400 µA	2.4		V
V_{OL2} V_{OH2}	Output Low Voltage Output High Voltage		I_{OL} = 10 µA I_{OH} = −10 µA	 V_{CC} − 0.2	0.2	V V
f_{SK}	SK Clock Frequency	NMC93CS56/NMC93CS66 NMC93CS56E/NMC93CS66E NMC93CS56M/NMC93CS66M		0 0 0	1 0.5 0.5	MHz
t_{SKH}	SK High Time	NMC93CS56/NMC93CS66 NMC93CS56E/NMC93CS66E NMC93CS56M/NMC93CS66M	(Note 2) (Note 3) (Note 3)	250 500 500		ns
t_{SKL}	SK Low Time	NMC93CS56/NMC93CS66 NMC93CS56E/NMC93CS66E NMC93CS56M/NMC93CS66M	(Note 2) (Note 3) (Note 3)	250 500 500		ns
t_{CS}	Minimum CS Low Time	NMC93CS56/NMC93CS66 NMC93CS56E/NMC93CS66E NMC93CS56M/NMC93CS66M	(Note 4) (Note 5) (Note 5)	250 500 500		ns
t_{CSS}	CS Setup Time	NMC93CS56/NMC93CS66 NMC93CS56E/NMC93CS66E NMC93CS56M/NMC93CS66M	Relative to SK	50 100 100		ns
t_{PRES}	PRE Setup Time	NMC93CS56/NMC93CS66 NMC93CS56E/NMC93CS66E NMC93CS56M/NMC93CS66M	Relative to SK	50 100 100		ns

*Thruout this table "M" refers to temperature range (−55°C to +125°C) not package.

DC and AC Electrical Characteristics

V_{CC} = 5V ±10% (unless otherwise specified) (Continued)

Symbol	Parameter	Part Number	Conditions	Min	Max	Units
t_{PES}	PE Setup Time	NMC93CS56/NMC93CS66 NMC93CS56E/NMC93CS66E NMC93CS56M/NMC93CS66M	Relative to SK	50 100 100		ns
t_{DIS}	DI Setup Time	NMC93CS56/NMC93CS66 NMC93CS56E/NMC93CS66E NMC93CS56M/NMC93CS66M	Relative to SK	100 200 200		ns
t_{CSH}	CS Hold Time		Relative to SK	0		ns
t_{PEH}	PE Hold Time	NMC93CS56/NMC93CS66 NMC93CS56E/NMC93CS66E NMC93CS56M/NMC93CS66M	Relative to CS Relative to CS Relative to CS	250 500 500		ns
t_{PREH}	PRE Hold Time		Relative to SK	0		ns
t_{DIH}	DI Hold Time	NMC93CS56/NMC93CS66 NMC93CS56E/NMC93CS66E NMC93CS56M/NMC93CS66M	Relative to SK	100 200 200		ns
t_{PD1}	Output Delay to "1"	NMC93CS56/NMC93CS66 NMC93CS56E/NMC93CS66E NMC93CS56M/NMC93CS66M	AC Test		500 1000 1000	ns
t_{PD0}	Output Delay to "0"	NMC93CS56/NMC93CS66 NMC93CS56E/NMC93CS66E NMC93CS56M/NMC93CS66M	AC Test		500 1000 1000	ns
t_{SV}	CS to Status Valid	NMC93CS56/NMC93CS66 NMC93CS56E/NMC93CS66E NMC93CS56M/NMC93CS66M	AC Test		500 1000 1000	ns
t_{DF}	CS to DO in TRI-STATE®	NMC93CS56/NMC93CS66 NMC93CS56E/NMC93CS66E NMC93CS56M/NMC93CS66M	AC Test CS = V_{IL}		100 200 200	ns
t_{WP}	Write Cycle Time				10	ms

Note 1: Stress above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of the specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Note 2: The SK frequency specification for Commercial parts specifies a minimum SK clock period of 1 microsecond, therefore in an SK clock cycle t_{SKH} + t_{SKL} must be greater than or equal to 1 microsecond. For example if t_{SKL} = 250 ns then the minimum t_{SKH} = 750 ns in order to meet the SK frequency specification.

Note 3: The SK frequency specification for Extended Temperature and Military parts specifies a minimum SK clock period of 2 microseconds, therefore in an SK clock cycle t_{SKH} + t_{SKL} must be greater than or equal to 2 microseconds. For example, if t_{SKL} = 500 ns then the minimum t_{SKH} = 1.5 microseconds in order to meet the SK frequency specification.

Note 4: For Commercial parts CS must be brought low for a minimum of 250 ns (t_{CS}) between consecutive instruction cycles.

Note 5: For Extended Temperature and Military parts CS must be brought low for a minimum of 500 ns (t_{CS}) between consecutive instruction cycles.

Note 6: This parameter is periodically sampled and not 100% tested.

Capacitance (Note 6)

T_A = 25°C, f = 1MHz

Symbol	Test	Typ	Max	Units
C_{OUT}	Output Capacitance		5	pF
C_{IN}	Input Capacitance		5	pF

AC Test Conditions

Output Load	1 TTL Gate and C_L = 100 pF
Input Pulse Levels	0.4V to 2.4V
Timing Measurement Reference Level	
Input	1V and 2V
Output	0.8V and 2V

Functional Description

The NMC93CS56 and NMC93CS66 have 10 instructions as described below. Note that the MSB of any instruction is a "1" and is viewed as a start bit in the interface sequence. The next 10-bits carry the op code and the 8-bit address for register selection.

Read (READ):

The Read (READ) instruction outputs serial data on the DO pin. After a READ instruction is received, the instruction and address are decoded, followed by data transfer from the selected memory register into a 16-bit serial-out shift register. A dummy bit (logical 0) precedes the 16-bit data output string. Output data changes are initiated by a low to high transition of the SK clock. In the NONVOLATILE SHIFT-REGISTER mode of operation, the memory automatically cycles to the next register after each 16 data bits are clocked out. The dummy-bit is suppressed in this mode and a continuous string of data is obtained.

Write Enable (WEN):

When V_{CC} is applied to the part, it powers up in the Write Disable (WDS) state. Therefore, all programming modes must be preceded by a Write Enable (WEN) instruction. Once a Write Enable instruction is executed programming remains enabled until a Write Disable (WDS) instruction is executed or V_{CC} is removed from the part.

Write (WRITE):

The Write (WRITE) instruction is followed by 16 bits of data to be written into the specified address. After the last bit of data is put on the data-in (DI) pin, CS must be brought low before the next rising edge of the SK clock. This falling edge of CS initiates the self-timed programming cycle. The PE pin **MUST** be held "high" while loading the WRITE instruction, however, after loading the WRITE instruction the PE pin becomes a "don't care". The DO pin indicates the READY/BUSY status of the chip if CS is brought high after a minimum of 250 ns (t_{CS}). DO = logical 0 indicates that programming is still in progress. DO = logical 1 indicates that the register at the address specified in the instruction has been written with the data pattern specified in the instruction and the part is ready for another instruction.

Write All (WRALL):

The Write All (WRALL) instruction is valid only when the "protect register" has been cleared by executing a PRCLEAR instruction. The WRALL instruction will simultaneously program all registers with the data pattern specified in the instruction. Like the WRITE instruction, the PE pin **MUST** be held "high" while loading the WRALL instruction, however, after loading the WRITE instruction the PE pin becomes a "don't care". As in the WRITE mode, the DO pin indicates the READY/BUSY status of the chip if CS is brought high after a minimum of 250 ns (t_{CS}).

Write Disable (WDS):

To protect against accidental data disturb, the Write Disable (WDS) instruction disables all programming modes and should follow all programming operations. Execution of a READ instruction is independent of both the WEN and WDS instructions.

Protect Register Read (PRREAD):

The Protect Register Read (PRREAD) instruction outputs the address stored in the "protect register" on the DO pin. The PRE pin **MUST** be held "high" while loading the instruction. Following the PRREAD instruction the 8-bit address stored in the memory Protect Register is transferred to the serial out shift register. As in the READ mode, a dummy bit (logical 0) precedes the 8-bit address string.

Protect Register Enable (PREN):

The Protect Register Enable (PREN) instruction is used to enable the PRCLEAR, PRWRITE, and PRDS modes. Before

Instruction Set for the NMC93CS56 and NMC93CS66

Instruction	SB	Op Code	Address	Data	PRE	PE	Comments
READ	1	10	A7–A0		0	X	Reads data stored in memory, starting at specified address.
WEN	1	00	11XXXXXX		0	1	Write enable must precede all programming modes.
WRITE	1	01	A7–A0	D15–D0	0	1	Writes register if address is unprotected.
WRALL	1	00	01XXXXXX	D15–D0	0	1	Writes all registers. Valid only when Protect Register is cleared.
WDS	1	00	00XXXXXX		0	X	Disables all programming instructions.
PRREAD	1	10	XXXXXXXX		1	X	Reads address stored in Protect Register.
PREN	1	00	11XXXXXX		1	1	Must immediately precede PRCLEAR, PRWRITE, and PRDS instructions.
PRCLEAR	1	11	11111111		1	1	Clears the "protect register" so that no registers are protected from WRITE.
PRWRITE	1	01	A7–A0		1	1	Programs address into Protect Register. Thereafter, memory addresses ≥ the address in Protect Register are protected from WRITE.
PRDS	1	00	00000000		1	1	One time only instruction after which the address in the Protect Register cannot be altered.

Functional Description (Continued)

the PREN mode can be entered, the part must be in the Write Enable (WEN) mode. Both the PRE and PE pins **MUST** be held "high" while loading the instruction.

Note that a PREN instruction must **immediately** precede a PRCLEAR, PRWRITE, or PRDS instruction.

Protect Register Clear (PRCLEAR):

The Protect Register Clear (PRCLEAR) instruction clears the address stored in the Protect Register and, therefore, enables all registers for the WRITE and WRALL instruction. The PRE and PE pins must be held "high" while loading the instruction, however, after loading the PRCLEAR instruction the PRE and PE pins become "don't care". Note that a PREN instruction must **immediately** precede a PRCLEAR instruction.

Protect Register Write (PRWRITE):

The Protect Register Write (PRWRITE) instruction is used to write into the Protect Register the address of the first register to be protected. After the PRWRITE instruction is executed, all memory registers whose addresses are greater than or equal to the address specified in the Protect Register are protected from the WRITE operation. Note that before executing a PRWRITE instruction the Protect Register must first be cleared by executing a PRCLEAR operation and that the PRE and PE pins must be held "high" while loading the instruction, however, after loading the PRWRITE instruction the PRE and PE pins become 'don't care'. Note that a PREN instruction must *immediately* precede a PRWRITE instruction.

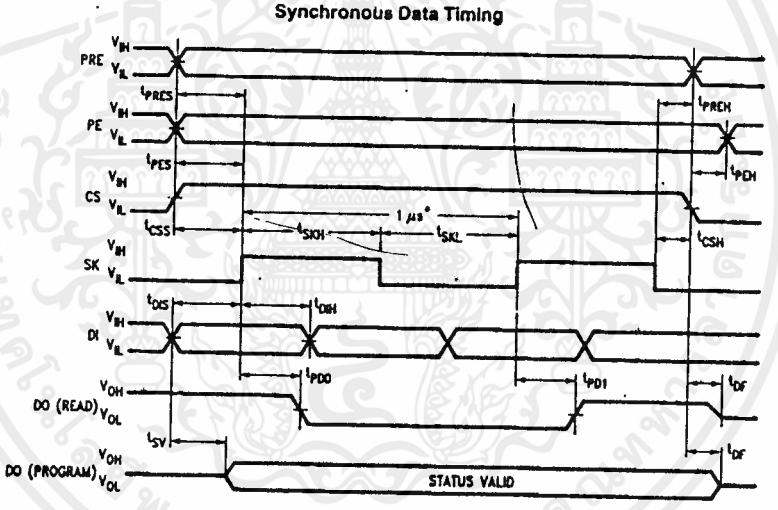
Protect Register Disable (PRDS):

The Protect Register Disable (PRDS) instruction is a **one time only** instruction which renders the Protect Register unalterable in the future. Therefore, the specified registers become **PERMANENTLY** protected against data changes. As in the PRWRITE instruction the PRE and PE pins must be held "high" while loading the instruction, and after loading the PRDS instruction the PRE and PE pins become "don't care".

Note that a PREN instruction must **immediately** precede a PRDS instruction.

Timing Diagrams

Synchronous Data Timing

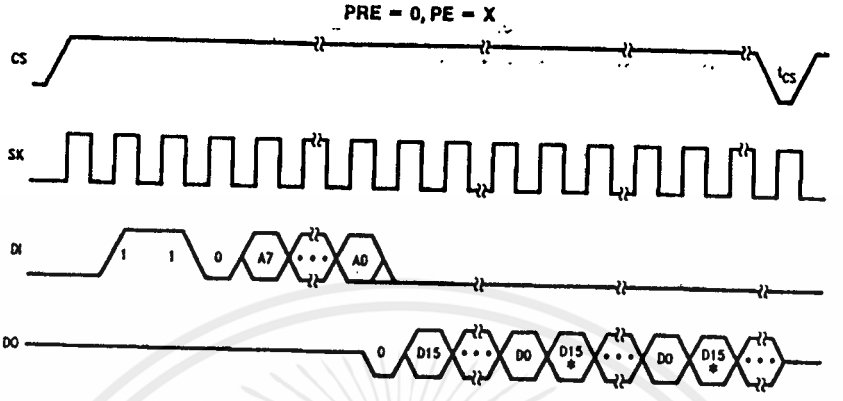


*This is the minimum SK period (See Note 2).

TL/D/9209–4

Timing Diagrams (Continued)

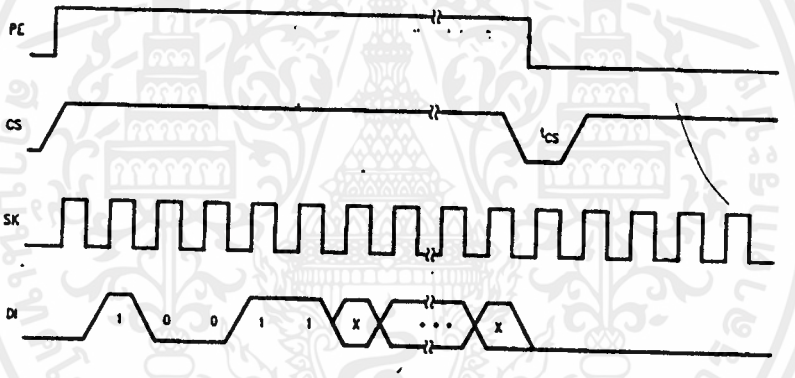
READ:*
PRE = 0, PE = X



TL/D/9209–5

*Address bit A_7 becomes a "don't care" for NMC93CS56.
*The memory automatically cycles to the next register.

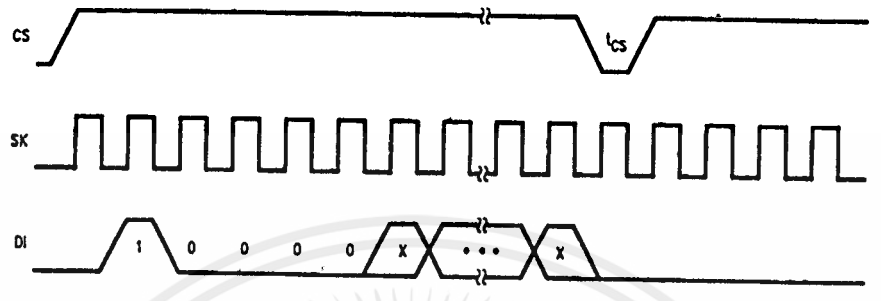
WEN:
PRE = 0, D0 = TRI-STATE



TL/D/9209–6

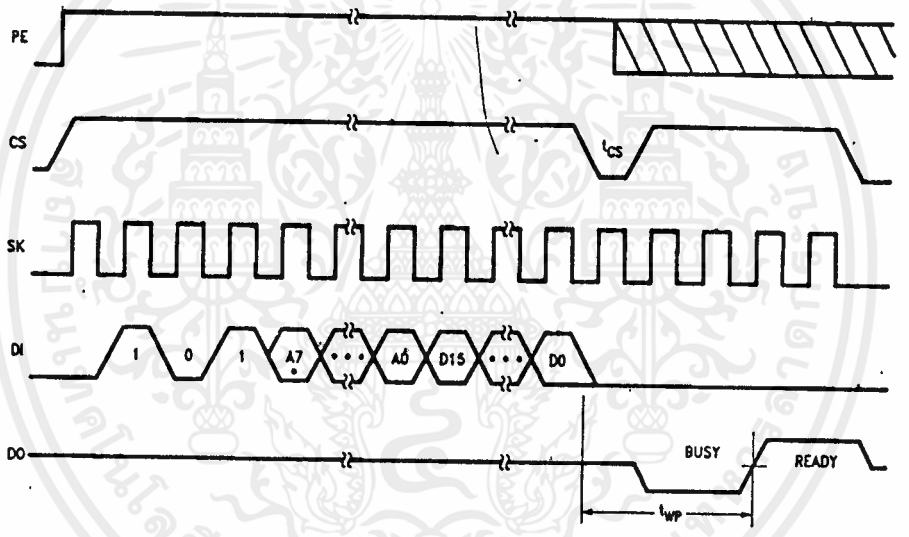
Timing Diagrams (Continued)

WDS:
PRE = 0, PE = X, DO = TRI-STATE



TL/D/9209–1

WRITE:*
PRE = 0

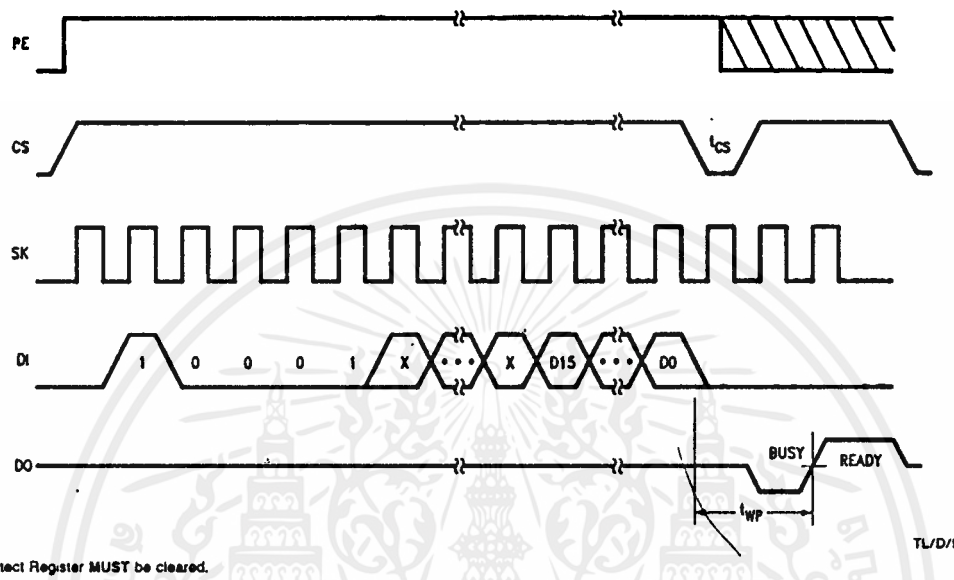


TL/D/9209–

*Address bit A_7 becomes a "don't care" for NMC93CS56.

Timing Diagrams (Continued)

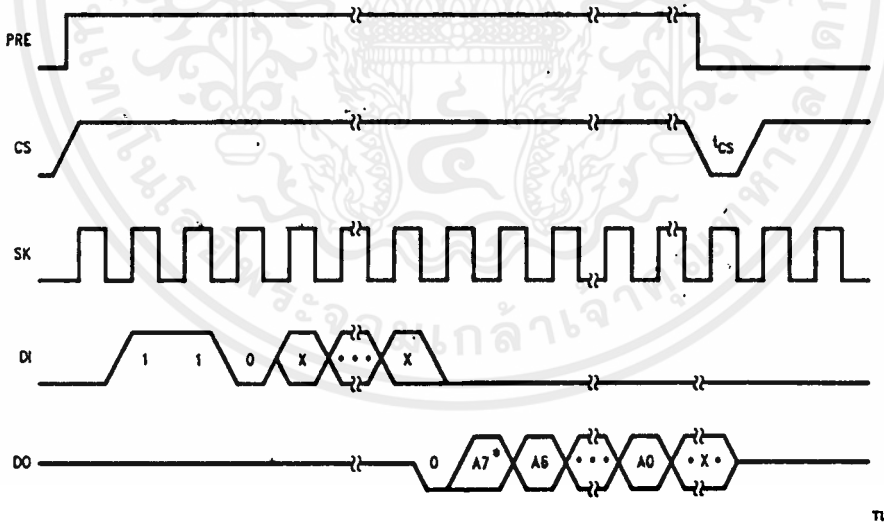
WRALL:*
PRE = 0



TL/D/9209–9

*Protect Register MUST be cleared.

PRREAD:*
PE = X

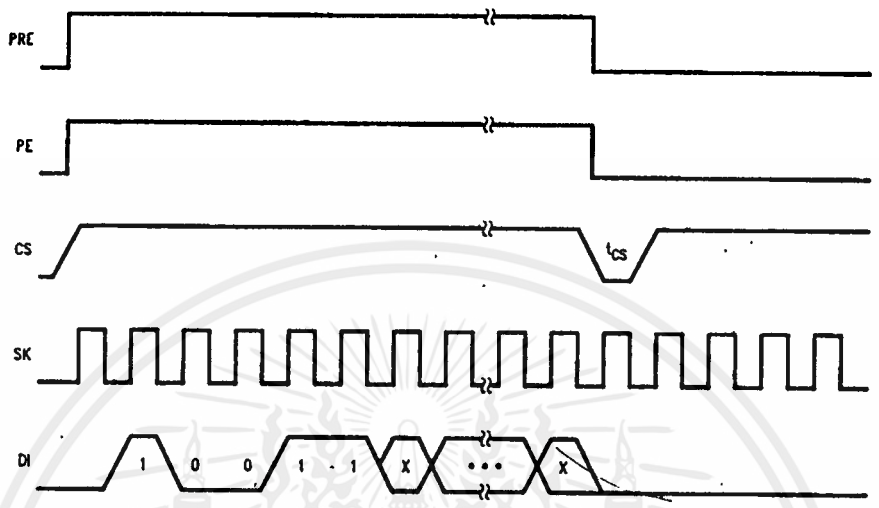


TL/D/9209–10

*Address bit A_7 becomes a "don't care" for NMC93CS56.

Timing Diagrams (Continued)

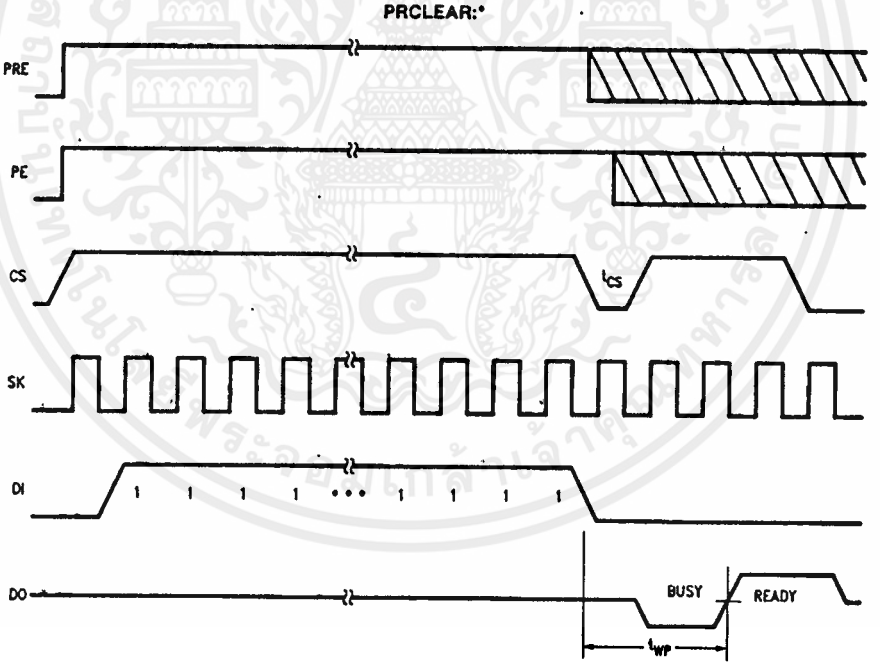
PREN:*
DO = TRI-STATE



TL/D/9209-1

*A WEN cycle must precede a PREN cycle.

PRCLEAR:*

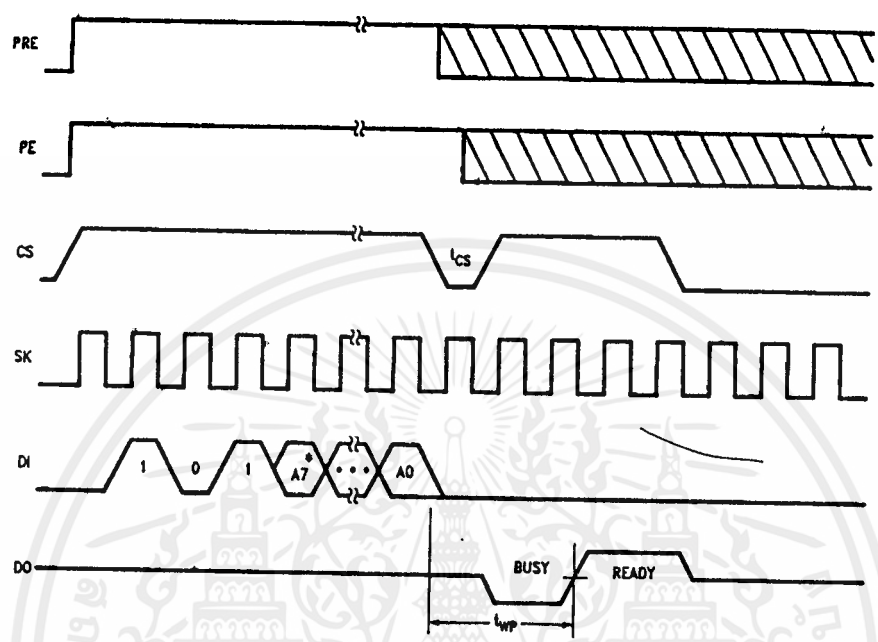


TL/D/9209-1

*A PREN cycle must immediately precede a PRCLEAR cycle.

Timing Diagrams (Continued)

PRWRITE:†

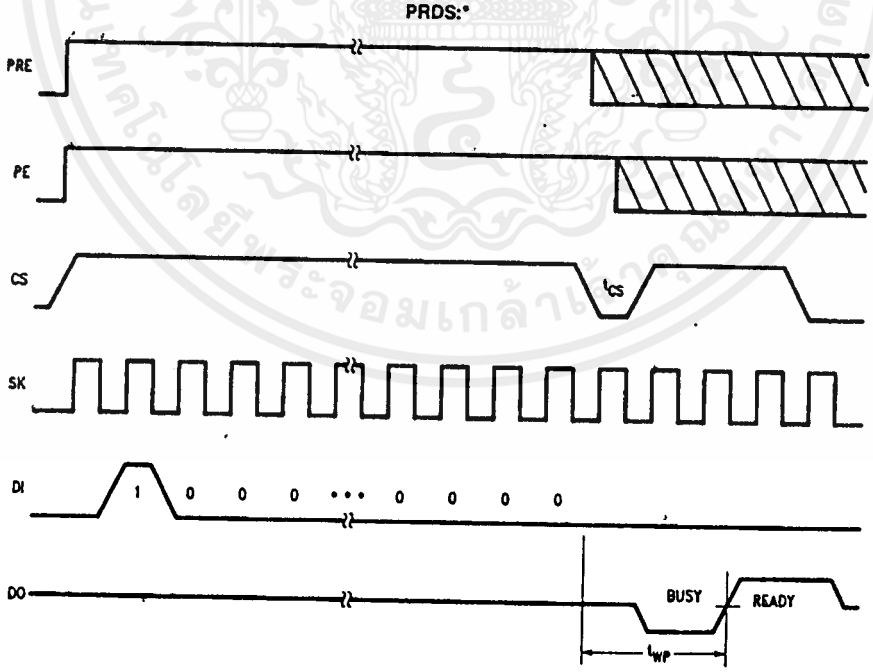


TL/D/9208–13

*Address bit A_7 becomes a "don't care" for NMC93CS56.

†Protect Register MUST be cleared before a PRWRITE cycle. A PREN cycle must immediately precede a PRWRITE cycle.

PRDS:*



TL/D/9208–14

*ONE TIME ONLY instruction. A PREN cycle must immediately precede a PRDS cycle.

National Semiconductor

54LS164/DM54LS164/DM74LS164
8-Bit Serial In/Parallel Out Shift Registers

General Description

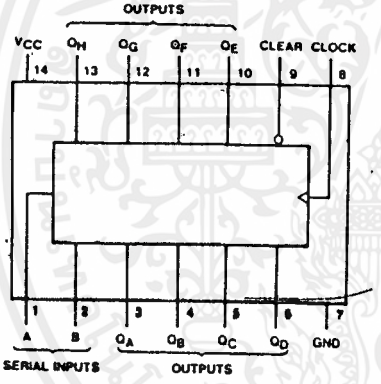
These 8-bit shift registers feature gated serial inputs and an asynchronous clear. A low logic level at either input inhibits entry of the new data, and resets the first flip-flop to the low level at the next clock pulse, thus providing complete control over incoming data. A high logic level on either input enables the other input, which will then determine the state of the first flip-flop. Data at the serial inputs may be changed while the clock is high or low, but only information meeting the setup and hold time requirements will be entered. Clocking occurs on the low-to-high level transition of the clock input. All inputs are diode-clamped to minimize transmission-line effects.

Features

- Gated (enable/disable) serial inputs
- Fully buffered clock and serial inputs
- Asynchronous clear
- Typical clock frequency 36 MHz
- Typical power dissipation 80 mW
- Alternate Military/Aerospace device (54LS164) is available. Contact a National Semiconductor Sales Office/ Distributor for specifications.

Connection Diagram

Dual-In-Line Package



TL/F/6398–1

Order Number 54LS164DMQB, 54LS164FMQB, 54LS164LMQB, DM54LS164J, DM54LS164W, DM74LS164M or DM74LS164N
See NS Package Number E20A, J14A, M14A, N14A or W14B

Function Table

Inputs				Outputs		...	
Clear	Clock	A	B	Q_A	Q_B	...	Q_H
L	X	X	X	L	L	...	L
H	L	X	X	Q_{A0}	Q_{B0}	...	Q_{H0}
H	↑	H	H	H	Q_{An}	...	Q_{Gn}
H	↑	L	X	L	Q_{An}	...	Q_{Gn}
H	↑	X	L	L	Q_{An}	...	Q_{Gn}

H = High t evel (steady state), L = Low Level (steady state)

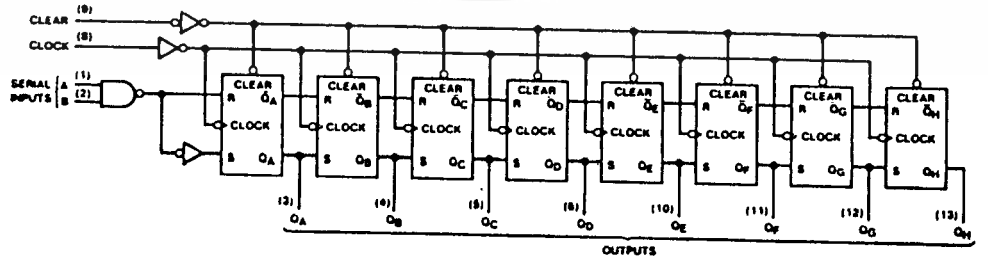
X = Don't Care (any input, including transitions)

↑ = Transition from low to high level

Q_{A0}, Q_{B0}, Q_{H0} = The level of Q_A, Q_B, or Q_H, respectively, before the indicated steady-state input conditions were established.

Q_{An}, Q_{Gn} = The level of Q_A or Q_G before the most recent ↑ transition of the clock; indicates a one-bit shift.

Logic Diagram



TL/F/6398–2

Absolute Maximum Ratings (Note)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Supply Voltage	7V
Input Voltage	7V
Operating Free Air Temperature Range	
DM54LS and 54LS	−55°C to +125°C
DM74LS	0°C to +70°C
Storage Temperature Range	−65°C to +150°C

Note: *The "Absolute Maximum Ratings" are those values beyond which the safety of the device cannot be guaranteed. The device should not be operated at these limits. The parametric values defined in the "Electrical Characteristics" table are not guaranteed at the absolute maximum ratings. The "Recommended Operating Conditions" tables will define the conditions for actual device operation.*

Recommended Operating Conditions

Symbol	Parameter		DM54LS164			DM74LS164			Units
			Min	Nom	Max	Min	Nom	Max	
V_{CC}	Supply Voltage		4.5	5	5.5	4.75	5	5.25	V
V_{IH}	High Level Input Voltage		2			2			V
V_{IL}	Low Level Input Voltage				0.7			0.8	V
I_{OH}	High Level Output Current				−0.4			−0.4	mA
I_{OL}	Low Level Output Current				4			8	mA
f_{CLK}	Clock Frequency (Note 4)		0		25	0		25	MHz
t_W	Pulse Width (Note 4)	Clock	20			20			ns
		Clear	20			20			
t_{SU}	Data Setup Time (Note 4)		17			17			ns
t_H	Data Hold Time (Note 4)		5			5			ns
t_{REL}	Clear Release Time (Note 4)		30			30			ns
T_A	Free Air Operating Temperature		−55		125	0		70	°C

Electrical Characteristics over recommended operating free air temperature range (unless otherwise noted)

Symbol	Parameter	Conditions		Min	Typ (Note 1)	Max	Units
V_I	Input Clamp Voltage	V_{CC} = Min, I_I = −18 mA				−1.5	V
V_{OH}	High Level Output Voltage	V_{CC} = Min, I_{OH} = Max, V_{IL} = Max, V_{IH} = Min	DM54	2.5	3.4		V
			DM74	2.7	3.4		
V_{OL}	Low Level Output Voltage	V_{CC} = Min, I_{OL} = Max, V_{IL} = Max, V_{IH} = Min	DM54		0.25	0.4	V
			DM74		0.35	0.5	
		I_{OL} = 4 mA, V_{CC} = Min	DM74		0.25	0.4	
I_I	Input Current @ Max Input Voltage	V_{CC} = Max, V_I = 7V				0.1	mA
I_{IH}	High Level Input Current	V_{CC} = Max, V_I = 2.7V				20	µA
I_{IL}	Low Level Input Current	V_{CC} = Max, V_I = 0.4V				−0.4	mA
I_{OS}	Short Circuit Output Current	V_{CC} = Max (Note 2)	DM54	−20		−100	mA
			DM74	−20		−100	
I_{CC}	Supply Current	V_{CC} = Max (Note 3)			16	27	mA

Note 1: All typicals are at V_{CC} = 5V, T_A = 25°C.

Note 2: Not more than one output should be shorted at a time, and the duration should not exceed one second.

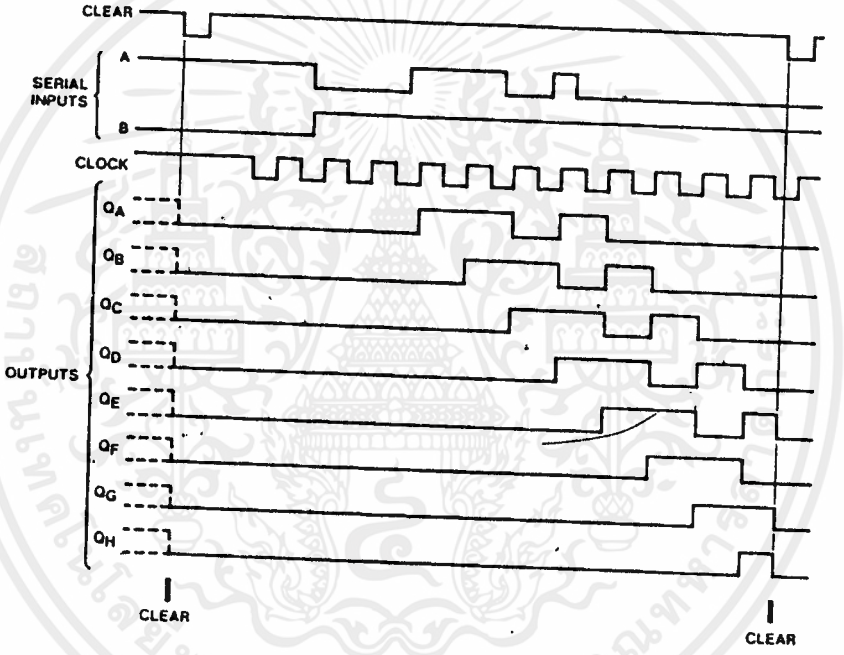
Note 3: I_{CC} is measured with all outputs open, the SERIAL input grounded, the CLOCK input at 2.4V, and a momentary ground, then 4.5V, applied to the CLEAR input.

Note 4: T_A = 25°C and V_{CC} = 5V.

Switching Characteristics at V_{CC} = 5V and T_A = 25°C (See Section 1 for Test Waveforms and Output Load)

Symbol	Parameter	From (Input) To (Output)	R_L = 2 kΩ				Units
			C_L = 15 pF		C_L = 50 pF		
			Min	Max	Min	Max	
f_{MAX}	Maximum Clock Frequency		25				MHz
t_{PLH}	Propagation Delay Time Low to High Level Output	Clock to Output		27		30	ns
t_{PHL}	Propagation Delay Time High to Low Level Output	Clock to Output		32		40	ns
t_{PHL}	Propagation Delay Time High to Low Level Output	Clear to Output		36		45	ns

Timing Diagram

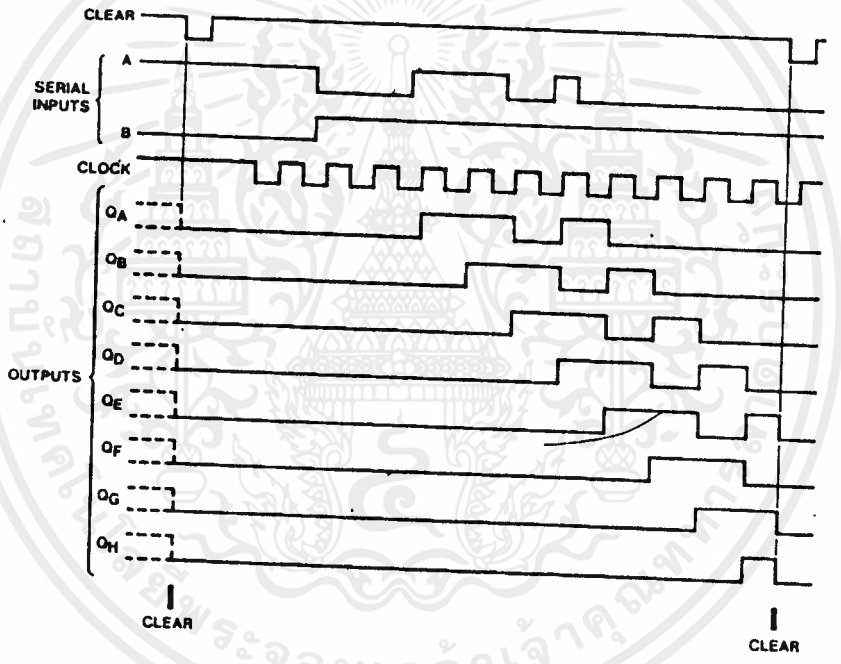


TL/F/6386-

Switching Characteristics at V_{CC} = 5V and T_A = 25°C (See Section 1 for Test Waveforms and Output Load)

Symbol	Parameter	From (Input) To (Output)	R_L = 2 kΩ				Units
			C_L = 15 pF		C_L = 50 pF		
			Min	Max	Min	Max	
f_{MAX}	Maximum Clock Frequency		25				MHz
t_{PLH}	Propagation Delay Time Low to High Level Output	Clock to Output		27		30	ns
t_{PHL}	Propagation Delay Time High to Low Level Output	Clock to Output		32		40	ns
t_{PHL}	Propagation Delay Time High to Low Level Output	Clear to Output		36		45	ns

Timing Diagram



TL/F/6300-1

National Semiconductor

54LS165/DM74LS165 8-Bit Parallel In/Serial Output Shift Registers

General Description

This device is an 8-bit serial shift register which shifts data in the direction of Q_A toward Q_H when clocked. Parallel-in access is made available by eight individual direct data inputs, which are enabled by a low level at the shift/load input. These registers also feature gated clock inputs and complementary outputs from the eighth bit.

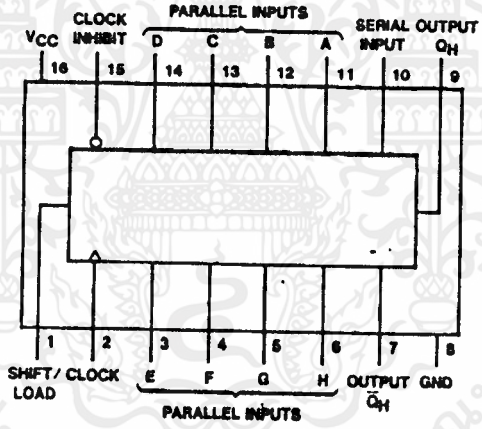
Clocking is accomplished through a 2-input NOR gate, permitting one input to be used as a clock-inhibit function. Holding either of the clock inputs high inhibits clocking, and holding either clock input low with the load input high enables the other clock input. The clock-inhibit input should be changed to the high level only while the clock input is high. Parallel loading is inhibited as long as the load input is high. Data at the parallel inputs are loaded directly into the register on a high-to-low transition of the shift/load input, regardless of the logic levels on the clock, clock inhibit, or serial inputs.

Features

- Complementary outputs
- Direct overriding (data) inputs
- Gated clock inputs
- Parallel-to-serial data conversion
- Typical frequency 35 MHz
- Typical power dissipation 105 mW

Connection Diagram

Dual-In-Line Package



TL/F/6399–1

Order Number 54LS165DMQB, 54LS165FMQB, DM74LS165WM or DM74LS165N
See NS Package Number J16A, M16B, N16E or W16A

Function Table

Inputs					Internal Outputs		Output
Shift/ Load	Clock Inhibit	Clock	Serial	Parallel A...H	Q_A	Q_B	Q_H
L	X	X	X	a...h	a	b	h
H	L	L	X	X	Q_{A0}	Q_{B0}	Q_{H0}
H	L	↑	H	X	H	Q_{An}	Q_{Gn}
H	L	↑	L	X	L	Q_{An}	Q_{Gn}
H	H	X	X	X	Q_{A0}	Q_{B0}	Q_{H0}

H = High Level (steady state), L = Low Level (steady state)

X = Don't Care (any input, including transitions)

↑ = Transition from low-to-high level

a...h = The level of steady-state input at inputs A through H, respectively.

Q_{A0}, Q_{B0}, Q_{H0} = The level of Q_A, Q_B, or Q_H, respectively, before the indicated steady-state input conditions were established.

Q_{An}, Q_{Gn} = The level of Q_A or Q_G, respectively, before the most recent ↑ transition of the clock.

Absolute Maximum Ratings (Note)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Supply Voltage	7V
Input Voltage	7V
Operating Free Air Temperature Range	
54LS	−55°C to +125°C
DM74LS	0°C to +70°C
Storage Temperature Range	−65°C to +150°C

Note: The "Absolute Maximum Ratings" are those values beyond which the safety of the device cannot be guaranteed. The device should not be operated at these limits. The parametric values defined in the "Electrical Characteristics" table are not guaranteed at the absolute maximum ratings. The "Recommended Operating Conditions" table will define the conditions for actual device operation.

Recommended Operating Conditions

Symbol	Parameter		54LS165			DM74LS165			Units
			Min	Nom	Max	Min	Nom	Max	
V_{CC}	Supply Voltage		4.5	5	5.5	4.75	5	5.25	V
V_{IH}	High Level Input Voltage		2			2			V
V_{IL}	Low Level Input Voltage				0.7			0.8	V
I_{OH}	High Level Output Current				−0.4			−0.4	mA
I_{OL}	Low Level Output Current				4			8	mA
f_{CLK}	Clock Frequency (Note 1)				30	0		25	MHz
f_{CLK}	Clock Frequency (Note 2)					0		20	MHz
t_W	Pulse Width (Note 2)	Clock	18			25			ns
		Load	15			15			
t_{SU}	Setup Time (Note 6)	Parallel	10			10			ns
		Serial	10			20			
		Enable	10			30			
		Shift	10			45			
t_H	Hold Time (Note 6)		5			0			ns
T_A	Free Air Operating Temperature		−55		125	0		70	°C

Electrical Characteristics over recommended operating free air temperature range (unless otherwise noted)

Symbol	Parameter	Conditions		Min	Typ (Note 3)	Max	Units
V_I	Input Clamp Voltage	V_{CC} = Min, I_I = −18 mA				−1.5	V
V_{OH}	High Level Output Voltage	V_{CC} = Min, I_{OH} = Max, V_{IL} = Max, V_{IH} = Min	54LS	2.5			V
			DM74	2.7	3.4		
V_{OL}	Low Level Output Voltage	V_{CC} = Min, I_{OL} = Max, V_{IL} = Max, V_{IH} = Min	54LS			0.4	V
			DM74		0.35	0.5	
		I_{OL} = 4 mA, V_{CC} = Min			0.25	0.4	
I_I	Input Current @ Max Input Voltage	V_{CC} = Max, V_I = 7V (DM74), V_I = 10V (54LS)	Shift/Load			0.3	mA
			Others			0.1	
I_{IH}	High Level Input Current	V_{CC} = Max, V_I = 2.7V	Shift/Load			60	µA
			Others			20	
I_{IL}	Low Level Input Current	V_{CC} = Max, V_I = 0.4V	Shift/Load			−1.2	mA
			Others			−0.4	
I_{OS}	Short Circuit Output Current	V_{CC} = Max (Note 4)	54LS	−20		−100	mA
			DM74	−20		−100	
I_{CC}	Supply Current	V_{CC} = Max (Note 5)			21	36	mA

Note 1: C_L = 15 pF, R_L = 2 kΩ, T_A = 25°C and V_{CC} = 5V

Note 2: C_L = 50 pF, R_L = 2 kΩ, T_A = 25°C and V_{CC} = 5V

Note 3: All typicals are at V_{CC} = 5V, T_A = 25° C.

Note 4: Not more than one output should be shorted at a time, and the duration should not exceed one second.

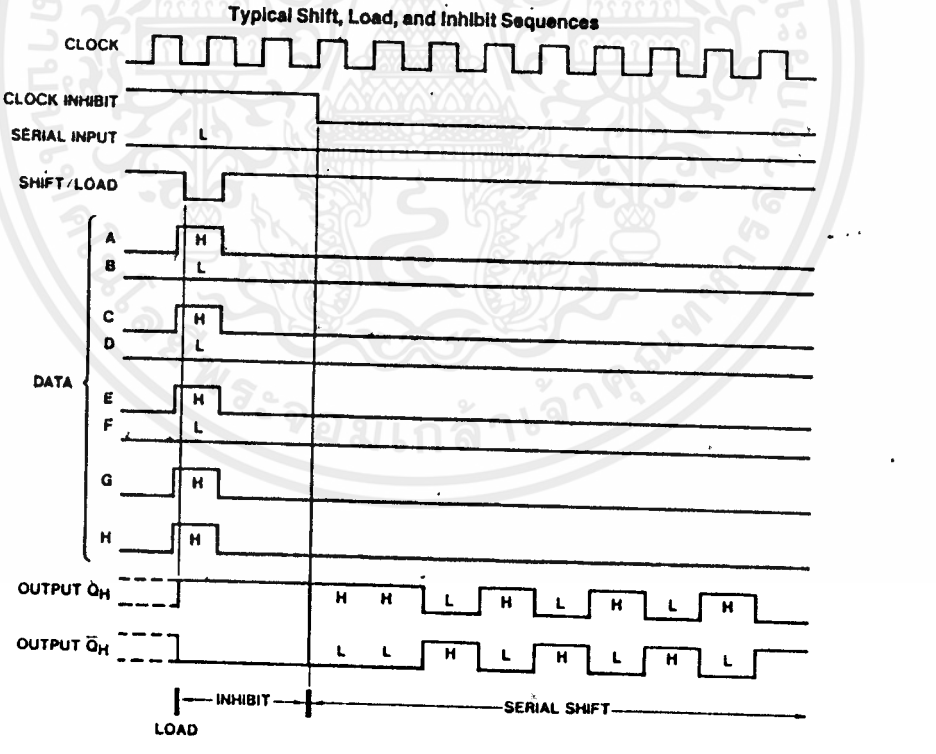
Note 5: With all outputs open, clock inhibit and shift/load at 4.5V, and a clock pulse applied to the CLOCK input, I_{CC} is measured first with the parallel inputs at 4.5V, then again grounded.

Note 6: T_A = 25°C and V_{CC} = 5V.

Switching Characteristics at V_{CC} = 5V and T_A = 25°C (See Section 1 for Test Waveforms and Output Load)

Symbol	Parameter	From (Input) To (Output)	54LS C_L = 15 pF		DM74LS R_L = 2 kΩ C_L = 50 pF		Units
			Min	Max	Min	Max	
f_{MAX}	Maximum Clock Frequency		25		20		MHz
t_{PLH}	Propagation Delay Time Low to High Level Output	Load to Any Q		30		37	ns
t_{PHL}	Propagation Delay Time High to Low Level Output	Load to Any Q		30		42	ns
t_{PLH}	Propagation Delay Time Low to High Level Output	Clock to Any Q		30		42	ns
t_{PHL}	Propagation Delay Time High to Low Level Output	Clock to Any Q		30		47	ns
t_{PLH}	Propagation Delay Time Low to High Level Output	H to Q_H		20		27	ns
t_{PHL}	Propagation Delay Time High to Low Level Output	H to Q_H		30		37	ns
t_{PLH}	Propagation Delay Time Low to High Level Output	H to $\overline{Q}_H$		30		32	ns
t_{PHL}	Propagation Delay Time High to Low Level Output	H to $\overline{Q}_H$		25		32	ns

Timing Diagram



TL/F/6398–3

กิตติกรรมประกาศ

คณะผู้จัดทำขอขอบพระคุณ อาจารย์ สุพรรณ กุลพาณิชย์
และคณะอาจารย์ภาควิชาเทคโนโลยีการวัดคุมทางอุตสาหกรรม สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหาร ลาดกระบัง ที่ได้ให้คำแนะนำ และคำปรึกษาจนโครงงานนี้ มีความสำเร็จระดับหนึ่ง

และคณะผู้จัดทำหวังเป็นอย่างยิ่งว่า โครงงานนี้จะเป็นประโยชน์ และช่วยในการค้นคว้าสำหรับนักศึกษาที่จะศึกษาในอนาคต

คณะผู้จัดทำ
18 มีนาคม 2536