เครื่องมือวัด บันทึกสัญญาณโดยใช้ไมโครคอมพิวเตอร์

PC - BASE STORAGE OSCILLOSCOPE

โดย

นายสายชล เมฆฉาย

นายสุรศักดิ์ กิจมงคลชัย

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาอุตสาหกรรมศาสตรบัณฑิต

สาขาเทคโนโลยีอิเลคทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2537

หัวข้อปริญญานิพนธ์ เครื่องมือวัด บันทึกสัญญาณโดยใช้ไมโครคอมพิวเตอร์

โดย

นาย สายชล เมฆฉาย เลขประจำตัว 35-103123

นาย สุรศักดิ์ กิจมงคลชัย เลขประจำตัว 35-103126

อาจารย์ที่ปรึกษา อ. ไพศาล สิทธิโยภาสกุล

ภาควิชา เทคนิคอุตสาหกรรม

ปีการศึกษา 2537

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง อนุมัติให้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาอุตสาหกรรมศาสตรบัณฑิต

คณะกรรมการสอบปริญญานิพนธ์

..... ประธานกรรมการ

( )

..... กรรมการ

( )

..... กรรมการ

( )

..... กรรมการ

( )

..... กรรมการ

( )

เครื่องมือวัด บันทึกสัญญาณโดยใช้ไมโครมิเตอร์

PC BASE STORAGE OSCILLOSCOPE

โดย 1. นายสายชล เมฆฉาย
2. นายสุรศักดิ์ กิจมงคลชัย

อาจารย์ที่ปรึกษา อ. ไพศาล สิทธิโยภาสกุล

บทคัดย่อ

ปริญญานิพนธ์นี้เป็นการประมวลผลรูปสัญญาณ (Signal Processing) โดยใช้เครื่องไมโครคอมพิวเตอร์ที่มีใช้งานกันอยู่ทั่ว ๆ ไป คือ IBM PC/XT/AT หรือเครื่องที่ใช้แทนกันได้ (Compatible) ในปัจจุบัน IBM PC มีราคาถูกลงมาก และมีประสิทธิภาพสูงขึ้นด้วย จึงเป็นผลให้เราสามารถนำ IBM PC นี้มาใช้ในงานประมวลผลสัญญาณทาง Digital ได้อย่างคุ้มค่า แต่เครื่องมือที่จำเป็นต้องใช้ในการเก็บรูปสัญญาณเพื่อนำมาให้ PC ประมวลผล (Storage Oscilloscope) นั้น ยังมีราคาแพงมาก และจำเป็นต้องสั่งซื้อจากต่างประเทศ ดังนั้นจึงทำให้เป็นที่มาของโครงงานนี้ เพื่อให้ได้เครื่องมือที่ใช้ในการประมวลผลสัญญาณ (Storage Oscilloscope) ที่มีราคาถูก และใช้แทนกันได้ในงานที่มีการประมวลสัญญาณที่ไม่ซับซ้อนมากนัก โดยความละเอียดของภาพที่ได้จาก IC Analog to Digital Convector เบอร์นี้เท่ากับ 128 level (7 bit) จึงคิดว่าโครงงานนี้จะเป็นประโยชน์ต่องานทางด้าน Signal Processing ต่อไป

ABSTRACT

This Project made for Signal Processing job by use microcomputer (IBM PC/XT/AT) Because in the present we can fine the IBM PC easily and now it has low cost but high efficiency. So that we can use for get data from Signal that we measure is too expensive cost and we must order from foreign. Thus this project occurred for become to low cost Signal Processing equipment by 128 level resolution. However, we think this project will be advantage on the Signal Processing job.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลงด้วยความเรียบร้อยด้วยการให้ความสนับสนุนจากผู้มีพระคุณดังต่อไปนี้

1. อาจารย์ไพศาล สิทธิโยภาสกุล

2. พี่ ๆ เพื่อน ๆ และน้อง ๆ ห้อง 3L.

3. เพื่อน ๆ ที่ บริษัท NS Electronics (Bangkok) จำกัด

4. พี่ ๆ และเพื่อน ๆ แผนก หหน. กอท. ฝรส. การไฟฟ้าฝ่ายผลิตฯ บางกรวย นนทบุรี

ทางกลุ่มผู้จัดทำขอขอบคุณมา ณ. ที่นี้ด้วย

อนึ่ง กลุ่มผู้จัดทำขอมอบคุณงามความดีทั้งหลายจากปริญญานิพนธ์ฉบับนี้แก่ บิดา มารดา ตลอดจนญาติมิตรที่ล่วงลับไปแล้วทุกท่าน

สารบัญ

บทที่ 1
บทนำ

ปัจจุบันเครื่อง Oscilloscope ได้แพร่หลายในวงการอิเล็กทรอนิกส์เป็นอย่างมาก ไม่ว่าจะเป็นงานวิเคราะห์ งานซ่อม หรือการเก็บข้อมูลบางอย่างไว้อ้างอิง จะยังมีอื่น ๆ อีกมากมาย ในปัจจุบันเครื่อง Oscilloscope รุ่นใหม่ที่เก็บสัญญาณหรือบันทึกข้อมูล พร้อมกับยังมี Function พิเศษต่าง ๆ มากมาย ซึ่งก็จะมีราคาสูงมากและในโรงงานอุตสาหกรรมได้นำเอาคอมพิวเตอร์เข้ามาควบคุมเครื่องจักรเพื่อลดการผิดพลาด ลดเวลาการสูญเสียต่าง ๆ และยังสามารถเพิ่มประสิทธิภาพการทำงานได้สูงขึ้น จากข้อดีของ Oscilloscope และคอมพิวเตอร์นั้น จึงเป็นที่เริ่มต้นของโครงการนี้ที่จะสร้างการ์ดใช้งานร่วมกับคอมพิวเตอร์ให้เป็น Oscilloscope แสดงผลหน้าจอที่คอมพิวเตอร์โดยได้ตัดบาง Function ของ Oscilloscope ออกไป เพราะบาง Function ไม่ได้ถูกใช้งานนำไปบันทึกสัญญาณและวิเคราะห์ เช่น นำไปบันทึกสัญญาณของเครื่องตรวจสอบ IC โดยการตรวจสอบสัญญาณบางสัญญาณว่าเครื่องตรวจสอบ IC สามารถผลิตสัญญาณโดยที่ยังอยู่ใน spec เพื่อที่จะได้ผลผลิตที่มีคุณภาพหรือป้องกันของเสีย หรือนำไปใช้ทำ SPC (Statistic Process Control) ทำให้เกิดประโยชน์สูงสุดในการใช้คอมพิวเตอร์ ส่วนเครื่อง Oscilloscope ก็นำไปใช้งานอย่างอื่น ๆ ซึ่งก็จะเป็นการลดค่าใช้จ่ายในการที่จะซื้อ Oscilloscope หลาย ๆ ตัวมาใช้งานซึ่งจะไม่คุ้ม

ฉะนั้น จึงได้มีการสร้างเครื่องมือชนิดหนึ่งขึ้นมา เรียกว่า PC BASE STORAGE Oscilloscope เพื่อแก้ไขปัญหาในจุดนี้ และมีข้อดีคือใช้งานร่วมกับเครื่องคอมพิวเตอร์ทั่วไปที่มีใช้กันอย่างแพร่หลาย เพียงแต่นำ CARD ของ HARDWARE เข้าไปเสียบ SLOT ของเครื่องคอมพิวเตอร์ และมี SOFTWARE เป็นตัวควบคุมให้คอมพิวเตอร์ทำหน้าที่เป็น Digital Storage Oscilloscope โดยแสดงผลออกทางจอภาพ เครื่องพิมพ์ พร้อมทั้งบันทึกข้อมูลลงบนแผ่น Diskette เช่นเดียวกับเครื่องที่มีขายตามท้องตลาดทั่วไป

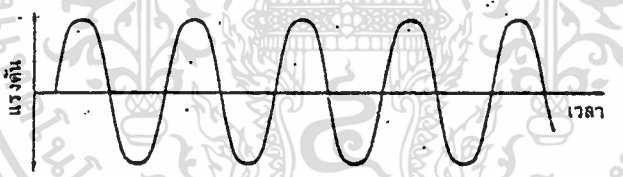
บทที่ 2

ทฤษฎีการทำงานของออสซิลโลสโคป

2.1 ออสซิลโลสโคป

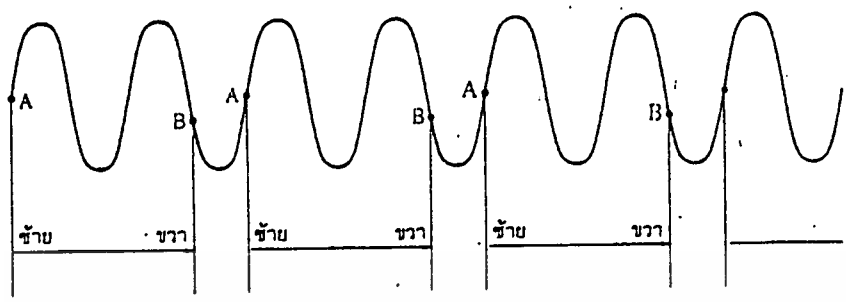
2.1.1 หลักการทำงานพื้นฐานของออสซิลโลสโคป

ออสซิโลสโคปสามารถแสดงภาพรูปคลื่นต่างๆได้โดยการผสมผสานระหว่างสามองค์ประกอบสำคัญ ซึ่งได้แก่ เวลา (แกน X) แรงดัน(แกน Y) และความสว่าง(แกน Z) ต่อไปนี้เราจะกล่าวถึงการผสมผสานขององค์ประกอบทั้งสาม โดยใช้รูปที่ 1. ประกอบการอธิบาย รูปที่ 2. แสดงให้เห็นถึงรูปคลื่นซายน์ซึ่งเป็นผลจากการทำงานขององค์ประกอบทั้งสามดังกล่าว เราจะเริ่มพิจารณาจากเพลตเบี่ยงเบนแนวราบ ซึ่งมีแรงดันที่เพิ่มขึ้นเป็นเชิงเส้นป้อนอยู่ แรงดันที่ป้อนแก่เพลตเบี่ยงเบนในแนวราบในลักษณะนี้จะเบี่ยงเบนให้ลำอิเล็คตรอนเคลื่อนที่จากซ้ายไปขวา ซึ่งเรามักเรียกกันว่า *กวาดภาพ*



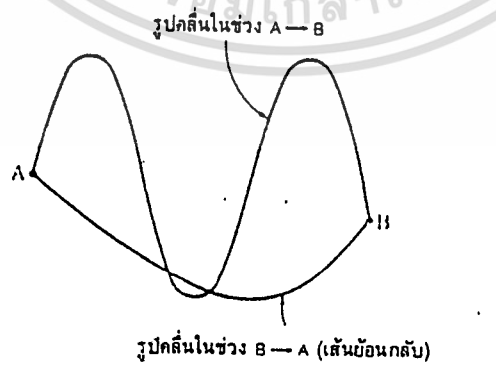
รูปที่ 1 ความสัมพันธ์ระหว่างเวลากับแรงดันในกรณีรูปคลื่นซายน์

แต่เนื่องจากจอภาพที่ใช้มีขนาดจำกัดที่แน่นอน เราไม่สามารถปล่อยให้มีการกวาดภาพจากซ้ายไปขวาโดยไม่มีกำหนดเวลา ดังนั้นจึงมักมีการออกแบบให้การกวาดภาพจากซ้ายไปขวาสิ้นสุดลงที่จุดใดจุดหนึ่ง (ดูรูปที่ 2 ประกอบ) แล้วให้มีการเริ่มต้นกวาดภาพจากซ้ายไปขวาใหม่ ซ้ำซากกันเช่นนี้อยู่เรื่อยๆ เมื่อใช้วิธีเช่นนี้เราสามารถแสดงรูปภาพรูปคลื่นสัญญาณซ้ำบนจอภาพได้ซ้ำแล้วซ้ำอีก

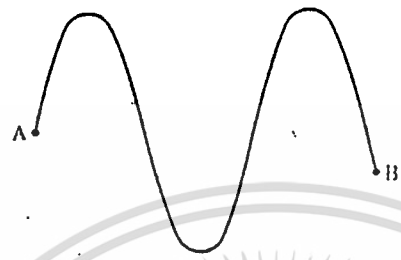


รูปที่ 2 การกำหนดช่วงของการกวาดภาพ
เพื่อสามารถแสดงรูปคลื่นบนจอภาพที่มีขนาดจำกัดแน่นอน

อย่างไรก็ตามวิธีการเช่นนี้ก็ยังมีปัญหา กล่าวคือในขณะที่การกวาดภาพย้อนกลับมายังจุดขวามือนั้น เป็นการกวาดภาพจากขวาไปซ้าย ทำให้มีเส้นจาก B ไป A ทับซ้อนบนช่วง A ไป B ในรูปที่ 2 ปรากฏการณ์เช่นนี้ได้แสดงให้เห็นชัดเชนขึ้นในรูปที่ 3 ด้วยเหตุนี้จึงจำเป็นต้องทำให้การกวาดภาพจาก B ไป A ไม่ปรากฏให้เห็นบนจอภาพ ทั้งนี้โดยการปิดกั้นลำอิเล็กตรอนไม่ให้เคลื่อนไปยังจอภาพในจังหวะที่การกวาดภาพมาถึงตำแหน่งทางขวามือ กล่าวอีกนัยหนึ่งก็คือทำให้ความสว่างจอภาพเป็นศูนย์ในช่วง B ไป A ถ้าหากทำได้เช่นนี้รูปคลื่นที่ได้ก็จะเป็นดังรูปที่ 4

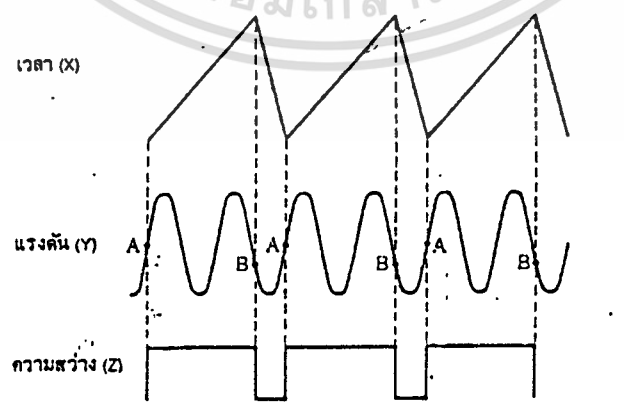


รูปที่ 3. แสดงการทับซ้อนของการกวาดภาพจาก A ไป B และย้อนกลับจาก B ไป A



รูปที่ 4 รูปคลื่นเช่นเดียวกับ รูปที่ 2.3 เพียงแต่กำจัดเส้นย้อนกลับให้หายไป

รูปที่ 5 เป็นการแสดงการเปลี่ยนแปลงตามเวลาขององค์ประกอบทั้งสามกล่าวคือ การเปลี่ยนแปลงแรงดันรูปคลื่น (X) ดังแสดงในรูปมักเรียกกันว่า คลื่นฟันเลื่อย เพราะมีลักษณะคล้ายฟันของใบเลื่อย รูปคลื่นนี้จะป้อนให้กับแผ่นควบคุมในแนวราบของหลอดคาโทดเรย์ ส่วนแผ่นควบคุมในแนวตั้งนั้นจะมีแรงดันรูปคลื่น (Y) ป้อนอยู่ และที่กริดควบคุมซึ่งทำหน้าที่ควบคุมความสว่างก็จะมีแรงดันรูปคลื่น (Z) ป้อนให้ ผลที่ได้ปรากฏบนจอเรืองแสงก็จะเป็นรูปคลื่นดังแสดงไว้ในรูปที่ 6



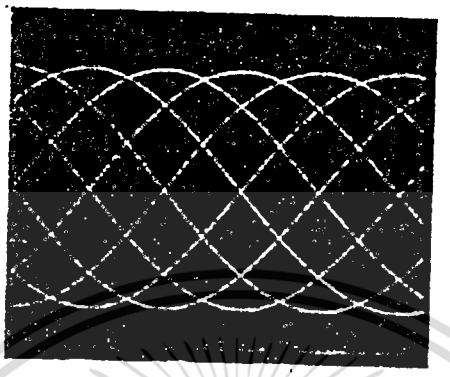
รูปที่ 5 ความสัมพันธ์ขององค์ประกอบทั้งสามดังกล่าวคือ
เวลา แรงดันและความสว่าง โดยเขียนบนแกนเวลา

อย่างไรก็ตาม องค์ประกอบทั้งสามดังกล่าวจำเป็นจะต้องทำงานประสานจังหวะกันได้เป็นอย่างดี จึงจะได้ภาพตามรูป 6 บนจอของออสซิโลสโคปหากว่าการทำงานขององค์ประกอบทั้งสามนี้ไม่ประสานกัน รูปคลื่นที่ได้จะซ้อนทับกันดังรูปที่ 7 ด้วยเหตุนี้จึงเห็นได้ว่าจุดหรือเวลาเริ่มต้นในการกวาดภาพนั้นจะมีความสำคัญ อย่างยิ่ง

เนื่องจากการกวาดภาพจะเกิดขึ้นซ้ำกันภายในช่วง A ไป B ซึ่งมีค่าคงที่แน่นอน ดังนั้นหากการกวาดภาพไม่ได้เริ่มต้นตรงจุดที่แรงดันมีค่าคงที่เท่ากันรูปที่ปรากฏบนจอก็จะเป็นรูปคลื่นที่ทับซ้อนกัน โดยแต่ละรูปคลื่นที่จะมีค่าแรงดันจุดเริ่มต้นที่แตกต่างกัน (ดูรูปที่ 7) แต่ถ้าหากสามารถปรับจังหวะการกวาดภาพให้เริ่มที่ค่าแรงดันคงที่ค่าหนึ่งแล้ว รูปคลื่นที่ได้จะซ้อนทับกันจนเห็นเป็นรูปเดียวกัน ดังรูปที่ 6 การทำให้การกวาดภาพเริ่มต้นที่ค่าแรงดันค่าหนึ่งและยังเป็นผลให้รูปคลื่นที่ได้ซ้อนทับกับรูปคลื่นเก่าเช่นนี้เรียกว่า การประสานจังหวะหรือซิงโครไนซ์

(Synchronize)

รูปที่ 6



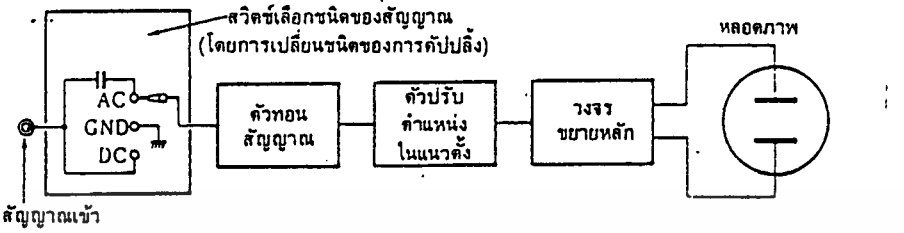
รูปที่ 7

2.1.2 การทำงานของแกนแนวตั้ง

องค์ประกอบพื้นฐานและหน้าที่การทำงานของส่วนต่างๆ

รูปที่ 8 แสดงองค์ประกอบพื้นฐานของแกนแนวตั้ง ในการอธิบายหน้าที่การทำงานของส่วนต่างๆต่อไปนี้จะอ้างอิงถึงรูป 8 นี้เป็นเกณฑ์

สัญญาณด้านเข้าจะผ่านสวิตช์หรือปุ่มเลือกชนิดของสัญญาณปุ่มเลือกนี้สามารถปรับได้สามตำแหน่งคือ AC, GND และ DC ในกรณีที่ต้องการเลือกดูสัญญาณไฟสลับเท่านั้นก็ปรับปุ่มนี้มาที่ "AC" แต่ถ้าหากต้องการเลือกดูสัญญาณที่มีส่วนของไฟตรงปนอยู่ด้วยก็ปรับปุ่มไปที่ " DC " เมื่อหมุนปุ่มมาที่ตำแหน่ง " GND " (หรือกราวนด์ - GROUND) วงจรด้านเข้าของออสซิโลสโคปจะไม่ต่อเข้าสัญญาณ หากจะต่อลงดิน (หรือกราวนด์) ซึ่งเท่ากับว่าสัญญาณขนาด 0 (ศูนย์) โวลต์ต่อกับสโคป การปรับตำแหน่งมาที่ GND เช่นนี้มีจุดมุ่งหมายเพื่อจะดูระดับแรงดันอ้างอิง (ในกรณีที่นี้แรงดัน 0 โวลต์)



รูปที่ 8 แสดงองค์ประกอบพื้นฐานของแกนแนวตั้ง

เมื่อปุ่มสวิตช์เลือกสัญญาณอยู่ที่ตำแหน่ง AC สัญญาณจะผ่านวงจรกรองผ่านสูง (HIGH PASS FILTER) ที่มีค่าความถี่คัตออฟ (CUT-OFF FREQUENCY) ระหว่าง 2-10 เฮิรตซ์ ทั้งนี้แล้วแต่โครงสร้างของวงจรกรองซึ่งอาจแตกต่างไปบ้างในแต่และรุ่นแต่ละยี่ห้อ แต่เมื่อปุ่มสวิตช์มาอยู่ที่ตำแหน่ง DC สัญญาณด้านเข้าจะต่อตรงเข้าสู่ตัวทอนสัญญาณ สัญญาณที่ผ่านสวิตช์เลือกสัญญาณจะผ่านต่อไปยังตัวทอนสัญญาณการที่ต้องมีวงจรทอนสัญญาณก็เพราะว่าสัญญาณที่ออสซิโลสโคปสามารถวัดและดูได้นั้น มีขนาดตั้งแต่ไม่กี่มิลลิโวลต์ เนื่องจากวงจรขยายสัญญาณภายในสโคปมีอัตราขยายคงที่ซึ่งออกแบบให้เหมาะกับสัญญาณเข้าขนาดเล็ก ดังนั้น เมื่อมีสัญญาณเข้าขนาดใหญ่จึงจำเป็นต้องทอนสัญญาณให้มีขนาดเล็กลง คุณสมบัติจำเป็นสองประการของตัวทอนสัญญาณได้แก่

ประการแรก เพื่อให้สามารถวัดค่าแรงดันในเชิงปริมาณได้ ปริมาณที่ถูกทอนสัญญาณจะต้องมีการปรับเทียบอย่างถูกต้องและแม่นยำ

ประการที่สอง จะต้องสามารถทอนสัญญาณให้มีขนาดเท่าใดก็ได้ตามต้องการ

ด้วยเหตุดังกล่าวข้างต้นนี้ ตัวทอนสัญญาณจึงประกอบด้วยส่วนตัวทอนสัญญาณที่ได้รับการปรับเทียบแล้วกับส่วนตัวทอนสัญญาณที่สามารถแปรค่าได้อย่างต่อเนื่อง ส่วนตัวทอนสัญญาณที่ได้รับการปรับเทียบแล้วเรียกกันว่า *ตัวทอนสัญญาณเป็นขั้น* (STEP ATTENUATOR) หน่วยของตัวทอนสัญญาณนี้มักไม่ค่อยใช้แสดงเป็นปริมาณการทอนสัญญาณ แต่แสดงเป็นสัดส่วนของแรงดันต่อ 1 ช่องสเกลบนหลอดภาพหรือเป็นโวลต์ต่อช่วง (V/DIV) นั่นเอง ขั้นการทอนสัญญาณที่มักพบเห็นกันมากที่สุดจะเป็นแบบขั้น 1-2-5 ซึ่งเป็นระบบที่มีการทอนสัญญาณลงขั้นละ - 6 เดซิเบล ดังนั้นปุ่มปรับในระบบนี้จึงเป็น 1 มิลลิโวลต์ต่อช่อง 2 มิลลิโวลต์ต่อช่อง 5 มิลลิโวลต์ต่อช่อง 10

มิลลิโวลต์ต่อช่อง เป็นต้น นอกจากระบบนี้ยังมีแบบขั้น 1-3 (ทอนสัญญาณลงขั้นละ - 10 เดซิเบล) หรือแบบขั้น 1-10 (ทอนสัญญาณลงขั้นละ - 20 เดซิเบล)

ส่วนตัวทอนสัญญาณที่แปรค่าอย่างต่อเนื่องได้เรียกกันว่า ตัวทอนสัญญาณแปรค่าได้ (VARIABLE ATTENUATION) ตัวทอนสัญญาณส่วนนี้จะได้รับการออกแบบให้สามารถครอบคลุมปริมาณการทอนสัญญาณซึ่งอยู่ระหว่างตัวทอนสัญญาณแบบขั้น เมื่อเราใช้ตัวทอนสัญญาณทั้งสองส่วนนี้ประกอบด้วยกันจะทำให้สามารถแสดงสัญญาณเข้าให้มีขนาดใดๆตามต้องการได้

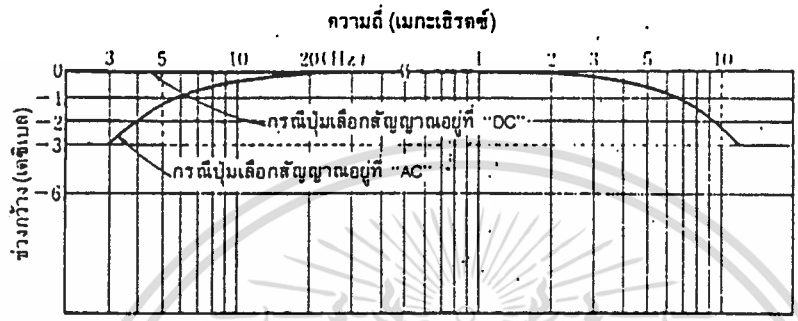
ตัวแปรตำแหน่งในแนวตั้งที่อยู่ถัดมานั้น ทำหน้าที่ปรับแต่งรูปคลื่นสัญญาณให้เลื่อนขึ้นหรือลงบนจอหลอดภาพ สัญญาณเข้าที่ผ่านออกจากตัวปรับตำแหน่งนี้จะได้รับการขยายให้มีขนาดใหญ่ขึ้นบ้าง แต่ยังไม่ใหญ่พอที่จะเบี่ยงเบนลำอิเล็กตรอนภายในหลอดภาพได้ ดังนั้นสัญญาณจากตัวปรับตำแหน่งจึงต้องผ่านวงจรขยายหลักเพื่อขยายให้มีขนาดใหญ่เพียงพอ แล้วจึงป้อนต่อไปยังแผ่นเบี่ยงเบนของหลอดภาพอีกต่อหนึ่ง

แถบความถี่ของแกนแนวตั้งและช่วงพลวัต

แถบความถี่ของแกนแนวตั้ง นับเป็นค่าที่มีความสำคัญซึ่งอาจสะท้อนให้เห็นถึงสมรรถนะทั้งหมดของออสซิลโลสโคปได้ ดังนั้นจึงควรทำความเข้าใจถึงความหมายของคำนี้ให้ดี เพื่อจะได้ใช้ออสซิลโลสโคปเต็มขีดความสามารถของเครื่อง

ความไวของแกนแนวตั้งของออสซิลโลสโคปนั้น จะมีการปรับเทียบกับแรงดันไฟตรงเอาไว้ อย่างไรก็ตามเมื่อค่าความถี่สัญญาณสูงขึ้น วงจรขยายสัญญาณภายในสโคปจะแสดงคุณสมบัติของวงจรกรองผ่านต่ำ (LOW PASS FILTER) ออกมา กล่าวคืออัตราขยายสัญญาณของวงจรจะค่อย ๆ ลดลงเมื่อความถี่สัญญาณเพิ่มสูงขึ้น ด้วยเหตุนี้ช่วงกว้าง (แอมพลิจูด หรือ AMPLITUDE) ของสัญญาณที่ปรากฏบนจอภาพหลอดภาพจะมีขนาดเล็กกว่าช่วงกว้างที่ควรจะเป็น ความแตกต่างนี้จะเพิ่มขึ้นเมื่อค่าความถี่เพิ่มขึ้น เมื่อช่วงกว้างของสัญญาณที่ปรากฏบนจอหลอดภาพมีขนาดเล็กกว่าช่วงกว้างที่ควรจะเป็นถึง 3 เดซิเบล เราจะเรียกช่วงความถี่จนถึงความถี่จุดนั้นว่า *แถบความถี่* (BANDWIDTH FREQUENCY) ของออสซิลโลสโคป กล่าวคือ ถ้าหากเราใช้ออสซิลโลสโคปที่มีแถบความถี่ 10 เมกะเฮิรตซ์ สังเกตดูรูปคลื่นซายน์ขนาดความถี่ 10 เมกะเฮิรตซ์ เราจะพบว่าช่วงกว้างของรูปคลื่นสัญญาณจะเล็กกว่าความเป็นจริงอยู่ 3 เดซิเบล อย่างไรก็ตาม การลดทอนสัญญาณดังกล่าวไม่ได้แม่นยำขนาดนั้น โดยปกติค่าความถี่ที่สัญญาณถูกลดทอนลง 3 เดซิเบล มักจะมีค่าสูงกว่าค่าตามพิกัดอยู่บ้าง รูปที่ 9 แสดงลักษณะสมบัติเชิงความถี่ของออสซิลโลสโคป 10 เมกะเฮิรตซ์ เอาไว้ ลักษณะสมบัติเช่นว่านี้จะแตกต่างไปบ้างตามรุ่นหรือโมเดลของเครื่อง แม้แต่เครื่องรุ่นเดียวกันก็อาจแตกต่างกันได้ อย่างไรก็ตามอาจกล่าวได้ว่าถ้าต้องการวัดโดยมีความแม่นยำภายในขอบเขต -3 เปอร์เซ็นต์แล้ว ช่วงความถี่สูงสุดของสัญญาณที่จะวัดค่าได้ควรมีค่าประมาณ 1/3 ของแถบความถี่ของออสซิลโลสโคป

หรือกล่าวกับกันก็คือ ถ้าหากต้องการวัดค่าเชิงปริมาณของสัญญาณความถี่ f ให้ได้แม่นยำ จะต้องใช้ออสซิลโลสโคปที่มีแถบช่วงความถี่มากกว่า 3f จึงจะดี



รูปที่ 9 ลักษณะสมบัติเชิงความถี่ออสซิลโลสโคป 10 เมกะเฮิรตซ์

ในกรณีที่ปุ่มเลือกสัญญาณอยู่ที่ตำแหน่ง AC ปรากฏการณ์ทำนองเดียวกันจะเกิดขึ้นในช่วงความถี่ต่ำ ในกรณีเช่นนี้ลักษณะสมบัติเชิงความถี่ของวงจรจะคล้ายกับวงจรกรอง CR กล่าวคือเป็น

$$1/\sqrt{1+[fc/f]2} \qquad \text{----------(1)}$$

จากสมการนี้ ช่วงความถี่ซึ่งทำให้ค่าดังกล่าวลดเป็น 97 เปอร์เซ็นต์ จะได้

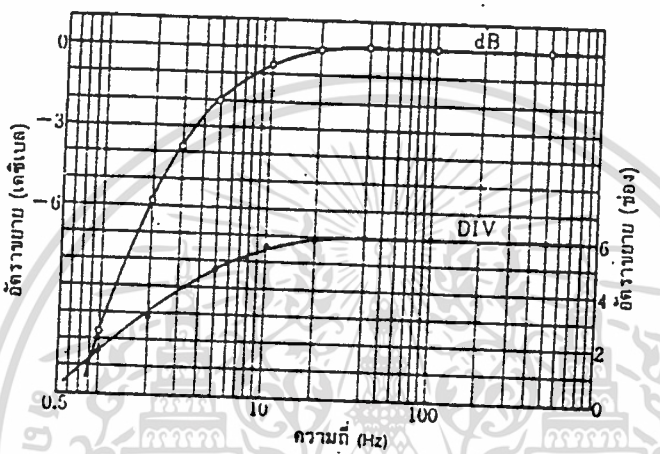
$$1/\sqrt{1+[fc/f]2} = 0.97 \qquad \text{----------(2)}$$

ค่าพิกัดแถบความถี่ของออสซิลโลสโคปจะมีการระบุไว้เป็นสองกรณี คือ เมื่อมีการต่อเชื่อมสัญญาณแบบ AC กับแบบ DC ในกรณีที่เป็นแบบ AC หัวข้อส่วนนี้อาจระบุเป็น

AC : 5 Hz ~ 10 MHz (-3 dB)

" 5 Hz " ในข้อระบุจำเพาะข้างต้น หมายความว่า 5 เฮิรตซ์ คือ ความถี่ซึ่งขนาดสัญญาณมีขนาดลดลง -3 เดซิเบล ดังนั้นถ้าหากต้องการใช้ออสซิลโลสโคปตัวนี้ทำการวัดค่าขนาดสัญญาณให้มี

ความแม่นยำภายใน -3 เปอร์เซ็นต์ จะต้องใช้วัดสัญญาณที่มีความถี่ 5 เฮิรตซ์ x 4 เท่า หรือ 20 เฮิรตซ์ หรือมากกว่านั้น



รูป 10 ลักษณะสมบัติเชิงความถี่ของการต่อเชื่อมแบบ AC

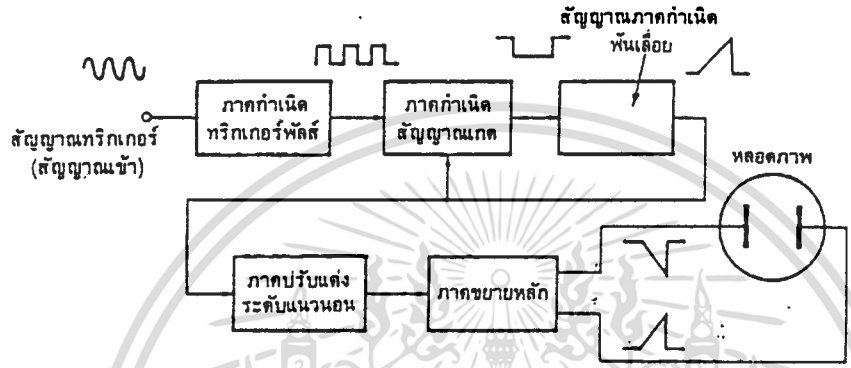
2.1.3 การทำงานของแกนแนวนอน

การกวาดภาพ

ที่แผ่นเบี่ยงเบนในแนวนอนของหลอดภาพออสซิลโลสโคป จะมี *"สัญญาณฟันเลื่อย"* ป้อนอยู่ สัญญาณฟันเลื่อยนี้จะถูกทริกเกอร์ (TRIGGER) หรือ กระตุ้นให้เข้าจังหวะกับสัญญาณเข้า นอกเหนือจากนี้ การทำงานในแกนแนวนอนหรือแกนเวลาจะต้องเป็นไปอย่างถูกต้องแม่นยำ ซึ่งทำให้มีความจำเป็นต้องให้กำเนิดสัญญาณฟันเลื่อยที่มีการแปรค่าเชิงเส้นอย่างแม่นยำเพื่อใช้วาดภาพ

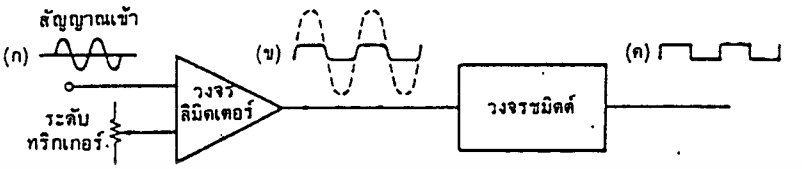
โครงสร้างพื้นฐานของแกนแนวนอนจะเป็นดังแสดงในรูปที่ 11 จากรูปจะเห็นว่าสัญญาณเข้าทำให้เกิดทริกเกอร์พัลส์ (TRIGGER PULSE) ทริกเกอร์พัลส์ที่เกิดขึ้นจะกระตุ้นให้มีการกวาดภาพ

ซึ่งระบบเช่นนี้เรียกกันว่า *"วิธีทริกเกอร์เพื่อกวาดภาพ"* ต่อไปนี้เราจะกล่าวถึงการทำงานของแต่ละส่วนโดยละเอียด ดังนี้

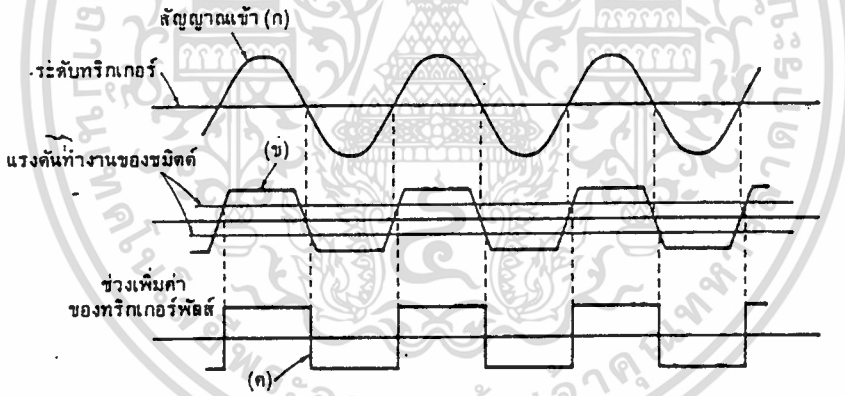


รูปที่ 11 โครงสร้างพื้นฐานของแกนแนวนอน

ก่อนอื่นจะกล่าวถึงภาคกำเนิดทริกเกอร์พัลส์ ซึ่งเรียกกันว่าวงจรลิมิตเตอร์ (LIMITTER) วงจรส่วนนี้จะประกอบขึ้นจากวงจรเปรียบเทียบแรงดันและวงจรชมิตต์ (SCHMITT) ทำหน้าที่แต่งรูปคลื่นสัญญาณ การทำงานโดยละเอียดของวงจรภาคนี้อาจดูได้จากรูปที่ 12 เมื่อสัญญาณเข้าผ่านมาที่วงจรลิมิตเตอร์ สัญญาณที่ผ่านออกมาจะถูกตัดส่วนบนและส่วนล่างออก (ดูรูปประกอบ) ตามค่าที่กำหนดเอาไว้ ค่าแรงดันที่กำหนดขอบเขตในการตัดสัญญาณ เราเรียกว่า ระดับการทริกเกอร์ (TRIGGER LEVEL) ระดับดังกล่าวนี้มีบทบาทสำคัญในการกำหนดตำแหน่งเริ่มต้นในการกวาดภาพ



รูปที่ 12 โครงสร้างของภาคกำเนิดทริกเกอร์พัลส์

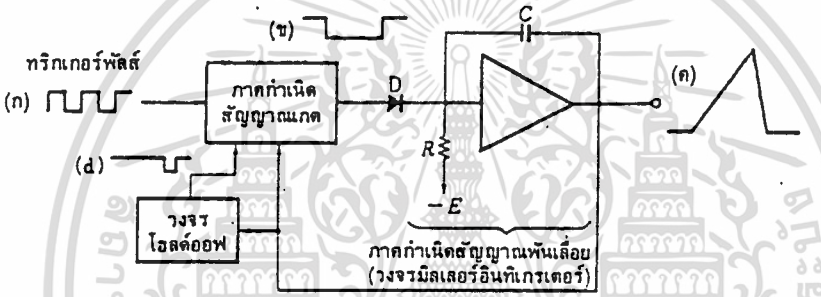


รูปที่ 13 รูปคลื่นสัญญาณในส่วนต่าง ๆ ของภาคกำเนิดทริกเกอร์พัลส์ แสดงการเปลี่ยนแปลงตามเวลา และความสัมพันธ์ระหว่างกัน

อย่างไรก็ตาม สัญญาณออกที่ได้จากวงจรลิมิตเตอร์ยังไม่เหมาะสมที่จะใช้เป็นทริกเกอร์พัลส์ จึงต้องใช้วงจรชมิตต์เพื่อแต่งรูปสัญญาณให้เป็นสัญญาณสี่เหลี่ยม (SQUARE WAVE) ที่มีการเพิ่มค่าอย่างฉับพลัน วงจรชมิตต์นี้จะมีลักษณะสมบัติฮีสเตอริซิส (HYSTERESIS) อยู่ด้วย ทั้งนี้เพื่อป้องกันสัญญาณทริกเกอร์ไม่ให้ทำงานผิดพลาดเนื่องจากสัญญาณรบกวน ดังนั้นจึงช่วยทำให้ตำแหน่งเริ่มต้นในการกวาดภาพไม่กระทบกระเทือนด้วยสัญญาณรบกวน

รูปที่ 13 แสดงการเปลี่ยนแปลงตามเวลาของรูปคลื่นสัญญาณส่วนต่าง ๆจากรูปจะเห็นว่าช่วงเพิ่มค่าของทริกเกอร์พัลส์นั้นจะล่าช้ากว่าระดับการทริกเกอร์อยู่บ้างเล็กน้อย ทั้งนี้เป็นผลจากแรงดันทำงาน (THRESHOLD VOLTAGE) ของวงจรชมิตต์ซึ่งมีลักษณะสมบัติฮีสเตอริซิส อย่างไรก็ตามส่วนล่าช้านี้ไม่มีผลในทางการใช้งานแต่อย่างใด

เมื่อได้ทริกเกอร์พัลส์แล้ว ต่อไปสัญญาณส่วนนี้จะไปสร้างสัญญาณเกตเพื่อกวาดภาพ วงจรส่วนนี้จะต่อเข้าเป็นลูป (LOOP) กับภาคกำเนิดสัญญาณฟันเลื่อย วงจรที่ช่วยทำให้ได้สัญญาณฟันเลื่อยที่มีเชิงเส้นดี มักใช้วงจรมิลเลอร์อินทิเกรเตอร์ (MILLER INTEGRATOR) ดังแสดงไว้ในรูปที่ 14

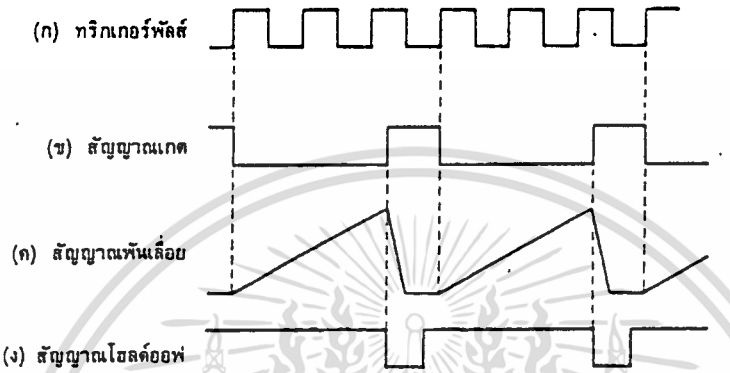


รูปที่ 14 ภาคกำเนิดสัญญาณฟันเลื่อย

เมื่อมีสัญญาณทริกเกอร์พัลส์เข้ามาที่ภาคกำเนิดสัญญาณเกต สัญญาณออกของวงจรเกตจะกลายเป็น "0" ทำให้ไดโอด D อยู่ในสภาพเปิดหรือไม่นำไฟฟ้า พร้อมกันนั้นวงจรอินทิเกรเตอร์จะเริ่มเก็บประจุด้วยค่าคงตัวเวลา (TIME CONSTANT) เป็น CR ขณะเดียวกันวงจรเข้าของภาคกำเนิดสัญญาณเกตก็จะเปิดและไม่รับทริกเกอร์พัลส์เข้ามาอีก ด้วยเหตุนี้สัญญาณออกของวงจรมิลเลอร์จะค่อย ๆ เพิ่มสูงขึ้นจนถึงระดับแรงดันที่ทำให้จุดบนจอภาพเคลื่อนมาถึงด้านขวาสุด แรงดันส่วนนี้จะป้อนกลับมาที่ภาคกำเนิดสัญญาณเกตและกลับค่าสัญญาณออกของวงจรภาคนี้ ดังนั้นไดโอด D จึงเปลี่ยนกลับเป็นปิดหรือนำไฟฟ้า ประจุที่ถูกเก็บอยู่ที่ตัวเก็บประจุ C ของวงจรมิลเลอร์จะถูกคายออกอย่างฉับพลัน จนกระทั่งแรงดันลดลงเท่ากับแรงดันที่ทำให้จุดบนจอภาพเคลื่อนมาอยู่ทางซ้ายสุดของจอภาพ

เมื่อการคายประจุหมดสิ้นลงแล้ว วงจรขาเข้าของภาคกำเนิดสัญญาณเกตก็จะเปิดออกเพื่อเตรียมรับสัญญาณทริกเกอร์พัลส์ใหม่อีกครั้ง วงจรโฮลด์ออฟ (HOLD-OFF) จะทำหน้าที่ในช่วงนี้ กล่าวคือ กำหนดช่วงเวลาตั้งแต่เริ่มมีการคายประจุจนกระทั่งเปิดวงจรขาเข้าของภาคกำเนิดสัญญาณเกต

วงจรนี้จะควบคุมไม่ให้ทริกเกอร์พัลส์เข้าสู่ภาคกำเนินสัญญาณพัลส์ได้จนกว่าการคายประจุเสร็จสิ้นสมบูรณ์ รูปที่ 15 แสดงรูปคลื่นสัญญาณส่วนต่าง ๆ ตามแกนเวลา รวมทั้งความสัมพันธ์ระหว่างรูปคลื่นเหล่านี้



รูปที่ 15 รูปคลื่นสัญญาณในส่วนต่าง ๆ ของภาคกำเนิดสัญญาณฟันเลื่อย แสดงการเปลี่ยนแปลงตามเวลาและความสัมพันธ์ระหว่างกัน

สัญญาณฟันเลื่อยที่ได้ออกมาจะถูกปรับแต่งตำแหน่งในแนวนอนต่อจากนั้นจึงเข้าสู่ภาคขยายหลัก (MAIN AMPLIFIER) เพื่อขยายให้มีแรงดันเพียงพอสำหรับเบี่ยงเบนลำอิเล็กตรอนได้ สัญญาณส่วนนี้จะถูกส่งต่อไปยังแผ่นเบี่ยงเบนแนวนอนของจอหลอดภาพอีกต่อหนึ่ง

ถ้าหากมีการเปลี่ยนแปลงอัตราขยายของภาคขยายหลักนี้ รูปคลื่นจะถูกขยายใหญ่ขึ้น เราเรียกวงจรส่วนนี้ว่า *ภาคขยายการกวาดภาพ* ซึ่งมีผลเช่นเดียวกับการทำให้เวลากวาดภาพเร็วขึ้นนั่นเอง (ความถี่สัญญาณสูงขึ้น)

นอกเหนือจากนี้ แกนเวลาหรือแกนนอน ยังมีความสำคัญในแง่ต่อไปนี้

ประการแรก แกนนี้ใช้เป็นบรรทัดฐานของเวลา ดังนั้นความเร็วในการกวาดภาพจึงต้องทำการปรับเทียบให้ถูกต้อง

ประการที่สอง แกนเวลานี้จะต้องสามารถปรับเปลี่ยนความเร็วการกวาดภาพสำหรับสัญญาณที่มีความถี่ต่ำไปจนถึงสัญญาณที่มีความถี่สูงได้ด้วย เพื่อช่วยให้การอ่านความเร็วการกวาดภาพเป็นไปได้โดยสะดวก ออสซิลโลสโคปที่ใช้จึงมักแสดงค่าเวลาที่ใช้ในการกวาดภาพได้หนึ่งช่องบนจอหลอดภาพหรือวินาทีต่อช่วง (SEC/DIV) และค่านี้จะสามารถปรับเปลี่ยนได้จากค่า 1-2-5 เช่นเดียวกับกรณีแกนแนวตั้ง นอกจากนี้ยังมีปุ่มปรับเพื่อเปลี่ยนค่าเวลาการกวาดภาพอย่างต่อเนื่อง ซึ่งก็ให้โวลุม (VOLUME) เช่นเดียวกับกรณีแกนแนวตั้ง

วิธีการต่าง ๆ ในการกวาดภาพ

1. กวาดภาพอัตโนมัติ ในระบบการทริกเกอร์กวาดภาพนั้น จะต้องมีทริกเกอร์พัลส์ช่วยกระตุ้นให้เริ่มการกวาดภาพ แต่ถ้าหากไม่มีสัญญาณขาเข้าหรือการปรับตั้งระดับทริกเกอร์เป็นไปไม่เหมาะสม ก็จะไม่มีสัญญาณทริกเกอร์พัลส์เกิดขึ้น ทำให้การกวาดภาพเริ่มไม่ได้ ยังผลให้ไม่มีรูปคลื่นปรากฏบนจอหลอดภาพ นอกเหนือจากสาเหตุดังกล่าวแล้ว การไม่มีรูปคลื่นปรากฏบนจอหลอดภาพยังมีสาเหตุอื่น ๆ อีก เช่น การปรับตำแหน่งในแนวตั้งหรือแนวนอนไม่เหมาะสม ปรับปุ่มลดทอนสัญญาณมากเกินไป จนทำให้สัญญาณทริกเกอร์มีขนาดเล็กซึ่งไม่อาจกระตุ้นให้วงจรกำเนิดสัญญาณทริกเกอร์พัลส์ทำงานได้เหล่านี้เป็นต้น ในเงื่อนไขต่าง ๆ ข้างต้น ถ้าหากเราสามารถทำให้มีเส้นสว่างปรากฏขึ้นบนจอหลอดภาพ (โดยที่การกวาดภาพอาจไม่เข้าจังหวะกับสัญญาณเข้าก็ได้) ก็อาจช่วยในการใช้งานออสซิลโลสโคปได้มาก จากการที่สามารถค้นหารูปคลื่นเพื่อพิจารณาดูขนาดโดยประมาณของรูปคลื่นสัญญาณได้ จะทำให้เราสามารถปรับตั้งระดับทริกเกอร์ใหม่จนรูปคลื่นสัญญาณหยุดนิ่งได้ ซึ่งจะอำนวยความสะดวกในการวัดค่าในลำดับต่อ ๆ ไป

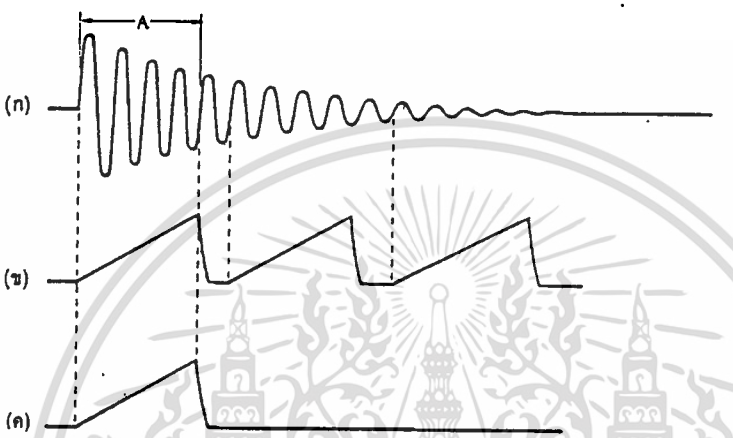
หรือในกรณีที่ปุ่มเลือกการต่อเชื่อมสัญญาณอยู่ที่ GND เพื่อปรับตั้งตำแหน่งของระดับแรงดัน 0 โวลต์ ถ้าหากเราทำให้มีการกวาดภาพเพื่อให้เกิดเส้นว่างบนจอภาพได้ก็จะสะดวกในการใช้งานเช่นกัน

การกวาดภาพโดยอิสระซึ่งไม่จำเป็นต้องเข้าจังหวะกับสัญญาณเข้าดังกล่าวเราเรียกว่า*การกวาดภาพแบบอิสระ* (FREE-RUNNING SWEEP) แต่ยังมีการกวาดภาพอีกวิธีซึ่งผสมผสานการกวาดภาพแบบอิสระกับการกวาดภาพแบบทริกเกอร์หรือแบบเข้าจังหวะ เราเรียกว่า *การกวาดภาพแบบอัตโนมัติ* (AUTO-FREE-RUNNING SWEEP) หรือเรียกย่อ ๆ ได้ว่า AUTO ระบบการกวาดภาพเช่นนี้ จะทำการ กวาดภาพแบบอิสระเมื่อไม่มีสัญญาณทริกเกอร์พัลส์อยู่ แต่ถ้ามีสัญญาณทริกเกอร์พัลส์ก็จะปรับเปลี่ยนเป็นการกวาดภาพแบบทริกเกอร์โดยอัตโนมัติ

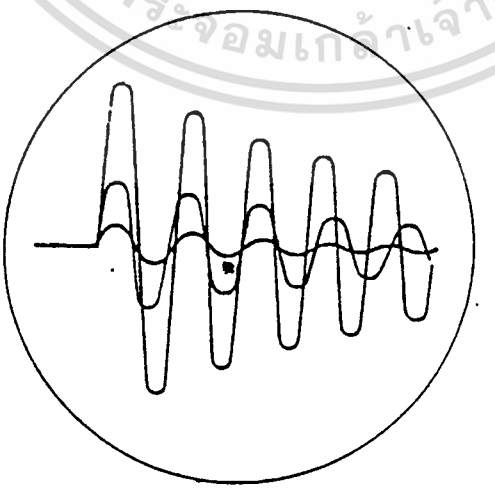
ระบบกวาดภาพแบบอัตโนมัตินี้ ทำงานได้โดยการตรวจสอบว่ามีสัญญาณทริกเกอร์พัลส์หรือไม่ แต่ถ้าหากความถี่ของทริกเกอร์พัลส์ต่ำ การตรวจสอบก็ทำไม่ได้ ช่วงความถี่ดังกล่าวนี้เราเรียกกันว่า *ความถี่จำกัดของการกวาดภาพแบบอัตโนมัติ* ซึ่งมักจะอยู่ในช่าง 30 ~ 50 *เฮิรตซ์*

การพัฒนาระบบกวาดภาพแบบอัตโนมัติดังกล่าวนี้นับเป็นก้าวกระโดดที่สำคัญซึ่งทำให้การใช้งานออสซิลโลสโคปเป็นไปได้สะดวกขึ้น สำหรับระบบการกวาดภาพแบบทริกเกอร์นั้น แม้จะไม่สามารถกวาดภาพแบบอิสระได้ แต่ก็ไม่มีปัญหาด้านความถี่จำกัดดังเช่นกรณีอัตโนมัติ ดังนั้นแบบทริกเกอร์จึงมีข้อได้เปรียบในการดูสัญญาณความถี่ต่ำ ๆ การกวาดภาพแบบหลังมักเรียกกันว่า *แบบปกติ* (NORMAL SWEEP หรือ NORM)

2. การกวาดภาพเดี่ยว แกนเวลาของออสซิลโลสโคปนั้นจำทการกวาดภาพของสัญญาณที่เกิดอย่างมีจังหวะช่วงเวลา (PERIODICAL) โดยให้จังหวะการกวาดภาพสัมพันธ์กับของสัญญาณนั้น ๆ ทำให้เกิดรูปคลื่นสัญญาณปรากฏบนจอหลอดภาพดูประหนึ่งสัญญาณหยุดนิ่งอยู่กับที่



รูปที่ 16 รูปคลื่นที่เปลี่ยนแปลงขนาดกับตำแหน่งการเกิดทริกเกอร์พัลส์



รูปที่ 17 สัญญาณจากรูปที่ 16 (ก) เมื่อใช้วิธีกวาดภาพตามแบบปกติ

แต่เมื่อเราพยายามให้ออสซิลโลสโคปแสดงรูปคลื่นสัญญาณดังรูปที่ 16 (ก) บนจอหลอดภาพโดยวิธีปกติทั่วไป รูปคลื่นที่ได้จะเป็นดังรูปที่ 17 คือ มีรูปคลื่นจำนวนมากซ้อนกันอยู่ ทำให้ยากแก่การอ่านค่าและสังเกตดู ที่เป็นเช่นนี้ก็เพราะมีสัญญาณฟันเลื่อยขนาดต่าง ๆ เกิดขึ้นซึ่งแตกต่างตามระดับการทริกเกอร์ ดังรูปที่ 16 (ข) แต่ถ้าหากสามารถทำให้สัญญาณฟันเลื่อยเกิดขึ้นเพียงครั้งเดียวดังรูป 16 (ค) รูปคลื่นที่ปรากฏบนจอหลอดภาพจะมีเฉพาะ ส่วน A ของรูปที่ 16 (ก) ทำให้สามารถหลีกเลี่ยงภาพซ้ำซ้อนดังแสดงในรูปที่ 17 ได้ การกวาดภาพที่เกิดขึ้นโดยอาศัยทริกเกอร์พัลส์ที่เกิดขึ้นในครั้งแรกเพียงครั้งเดียวเช่นนี้เราเรียกว่า *การกวาดภาพเดี่ยว* (SINGLE SWEEP)

การกวาดภาพแบบเดี่ยวจะให้รูปคลื่นปรากฏบนจอหลอดภาพเพียงชั่วระยะเวลาสั้น ๆ และเนื่องจากเวลาคงความสว่างของสารเรืองแสงที่ใช้ฉาบจอหลอดภาพสั้นมาก ดังนั้นเราจึงแทบจะสังเกตดูด้วยตาเปล่าไม่ได้ วิธีสังเกตดูที่ใช้กันโดยทั่วไปจึงอาศัยการถ่ายรูป ซึ่งจะกล่าวดังในรายละเอียดต่อไป

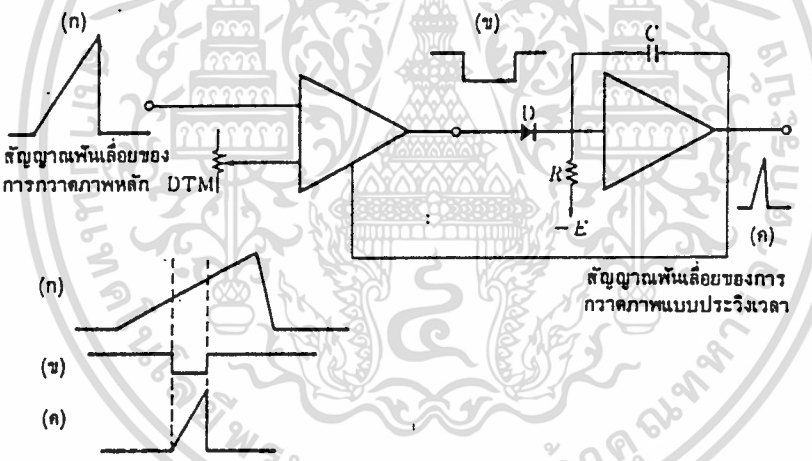
3. **การกวาดภาพแบบประวิงเวลา** การกวาดภาพแบบนี้จะช่วยในการขยายรูปคลื่นสัญญาณเฉพาะส่วน ซึ่งจะกล่าวถึงต่อไป การขยายรูปคลื่นสัญญาณดังกล่าวหมายถึงการขยายแกนเวลา กล่าวอีกแง่หนึ่งก็คือการเพิ่มความเร็วการกวาดภาพให้สูงขึ้นเฉพาะบางส่วนของรูปคลื่นสัญญาณ ด้วยวิธีการเช่นนี้ เราจึงอาจนำรูปคลื่นที่อยู่บนแกนเวลาในช่วงสั้น ๆ ขยายใหญ่ออกเพื่อดูในรายละเอียดได้

ยกตัวอย่าง เช่น รูปที่ 18 (ก) ถ้าหากต้องการขยายรูปคลื่นในส่วน A ออกไปในแนวนอนโดยการเพิ่มเวลาการกวาดภาพออกไปอีกจะได้ดังรูปที่ 18 (ค) ซึ่งรูปคลื่นส่วนที่ต้องการสังเกตดูจะหลุดออกนอกจอหลอดภาพ ในกรณีเช่นว่านี้ สิ่งที่น่าจะเป็นไปได้ก็คือ การให้เริ่มกวาดภาพจากตำแหน่งจุด A ในรูปที่ 18 (ก) ซึ่งเมื่อขยายให้ใหญ่ขึ้นก็จะได้รูปคลื่นดังรูปที่ 18 (ง) ที่เรายังสามารถสังเกตดูได้ หลักการของการกวาดภาพแบบประวิงเวลาก็อาศัยแนวคิดเช่นนี้นี่เอง

รูปที่ 18 การใช้กลไกกวาดภาพแบบประวิงเวลาเพื่อสังเกตรูปคลื่นเฉพาะส่วน

วิธีการทำงานของระบบการกวาดภาพนี้ก็คือ การประวิงเวลากวาดภาพให้ช้าลงกว่าเวลาปกติอยู่ T_D และโดยการปรับขนาดความเร็วการกวาดภาพโดยอิสระจะทำให้รูปคลื่นส่วน A ถูกขยายใหญ่ขึ้นดังรูปที่ 18(ง) วิธีการกวาดภาพเช่นนี้เราเรียกว่า การกวาดภาพแบบประวิงเวลา (DELAYED SWEEP) โดยเทียบกับวิธีทั่วไปซึ่งเรียกว่า *การกวาดภาพหลัก* โดยทั่วไปที่แผงควบคุมของออสซิลโลสโคปจะกำกับว่า "A SWEEP" สำหรับการกวาดภาพแบบประวิงเวลา

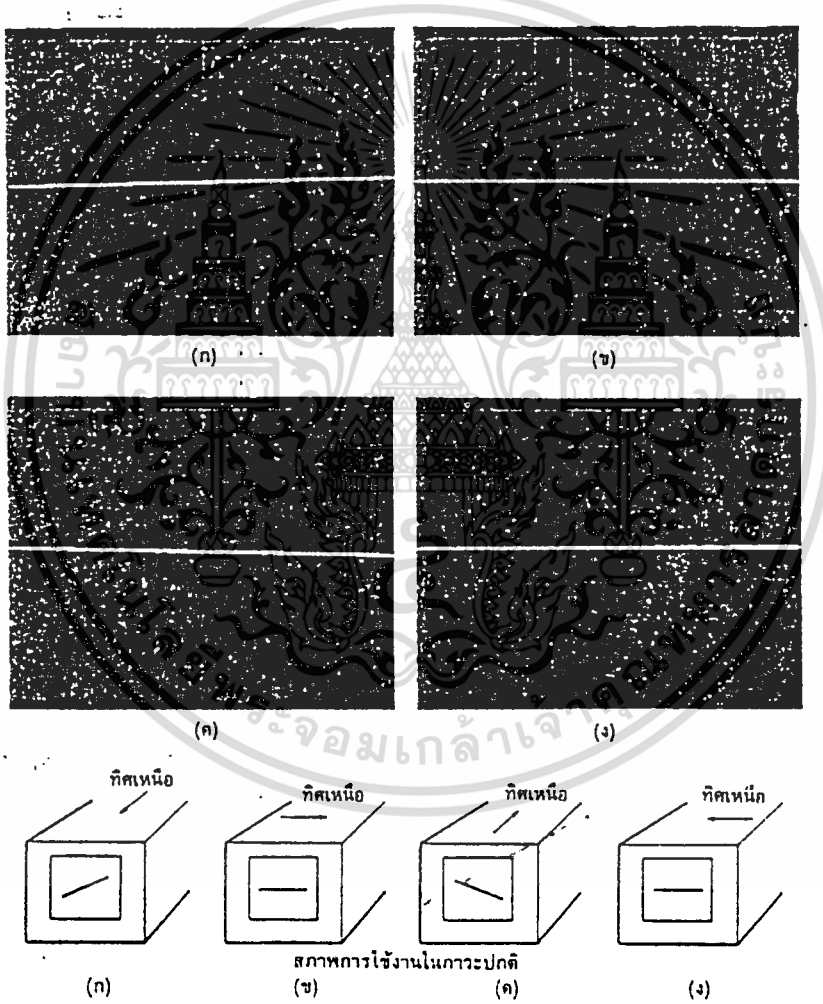
สำหรับสัญญาณเกตที่ใช้ในการกวาดภาพแบบประวิงเวลานั้น จะมีสัญญาณฟันเลื่อยจากการกวาดภาพหลักป้อนเข้าวงจรเปรียบเทียบค่า (COM-PARATOR) เมื่อสัญญาณมีขนาดใหญ่ถึงจุดหนึ่ง (ดูรูปที่ 19 ประกอบ) ก็จะได้สัญญาณออกที่เป็นสัญญาณเกตสำหรับการกวาดภาพแบบประวิงเวลาออกมาปุ่มซึ่งทำหน้าที่ควบคุมตำแหน่งในการเปรียบเทียบค่าของวงจรเปรียบเทียบค่าเรียกว่าปุ่ม DTM (DELAY TIME MULTIPLIER)



รูปที่ 19 วิธีสร้างสัญญาณแกตสำหรับกวาดภาพแบบประวิงเวลา

2.1.4 การปรับความลาดเอียงของเส้นสว่าง

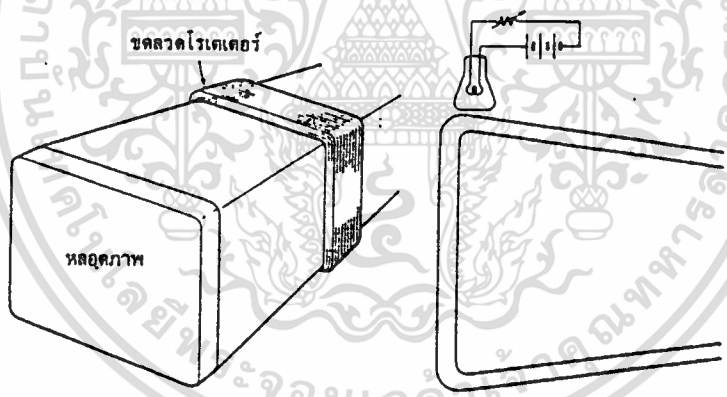
สำอิเล็กตรอนนั้นจะถูกหักเหได้โดยสนามไฟฟ้าและสนามแม่เหล็ก โดยเฉพาะอย่างยิ่งผลของสนามแม่เหล็กที่เกิดจากธรรมชาติอันได้แก่ขั้วแม่เหล็กโลกนั้น มีอิทธิพลต่อออสซิลโลสโคปไม่น้อยเลย



รูปที่ 20

รูปที่ 20 นั้น แสดงให้เห็นว่าทิศทางการวางออสซิลโลสโคปทำให้ผลของสนามแม่เหล็กโลกมีต่อเส้นสว่างได้อย่างไรบ้าง จากภาพจะเห็นได้ว่าเมื่อเราวางออสซิลโลสโคปไว้ในแนวตะวันออก/ตะวันตก แล้วหมุนตัวเครื่องไป 180 องศา เส้นสว่างจะเลื่อนขึ้นหรือลงเล็กน้อยดังรูปที่ 20 (ข) และรูปที่ 20 (ง) แต่จะไม่มีปัญหาในแง่การใช้งานแต่ประการใด แต่ถ้าหากเราหมุนเครื่องไป 90 องศาหรือหมุนเครื่องไป 180 องศา จากแนวเหนือ/ใต้ดังรูปที่ 20 (ก) และรูปที่ 20 (ค) เส้นสว่างจะลาดเอียงไปอันจะส่งผลเสียต่อการวัดได้ ซึ่งจำเป็นต้องแก้ไขให้ถูกต้อง

ให้ออสซิลโลสโคปนั้น จะมีกลไกซึ่งแก้ไขการลาดเอียงของเส้นสว่างอันเป็นผลมาจากสนามแม่เหล็กโลก วิธีแก้ดังกล่าวมีแสดงในรูปที่ 21 (ก) โดยติดตั้งขดลวดไว้ที่ตัวหลอดภาพแล้วปล่อยให้มีกระแสไหลผ่านตามความเหมาะสม เพื่อให้เกิดสนามแม่เหล็กซึ่งจะแก้ปัญหาการลาดเอียงของเส้นสว่างได้



(ก) วิธีการแก้การลาดเอียงของเส้นสว่างอันเนื่องจากสนามแม่เหล็กโลก

(ข) การส่องสว่างสเกลโดยใช้หลอดไฟ

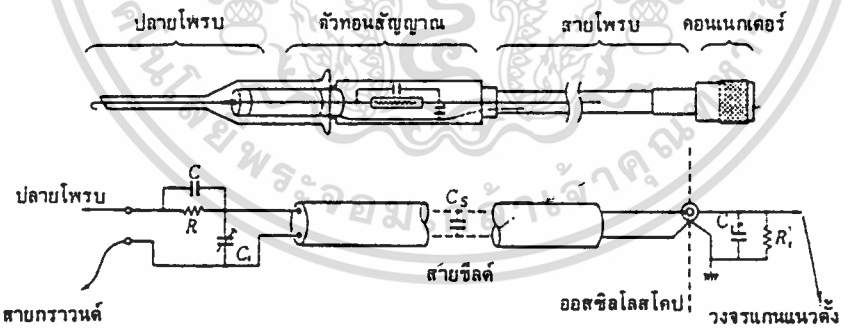
รูปที่ 21

อย่างไรก็ตามมาตรการแก้ไขดังกล่าวเหมาะสำหรับสนามแม่เหล็กโลก ซึ่งมีขนาดไม่ใหญ่นัก ในกรณีที่ต้องใช้ออสซิลโลสโคปในเงื่อนไขใช้งานซึ่งมีสนามแม่เหล็กเข้มมาก จึงควรใช้การกำบังหรือชีลด์ (SHIELD) ทางแม่เหล็กเพื่อขจัดอิทธิพลของสนามแม่เหล็ก แต่วิธีดังกล่าวนี้มีราคาแพง

2.1.5 การใช้งานโพรบ

โพรบ (PROBE) เป็นเครื่องมือที่ใช้ต่อโยงภาควงจรขาเข้าของออสซิลโลสโคปเข้ากับเป้าหมายที่ต้องการวัด โพรบมีอยู่มากมายตามแต่จุดประสงค์การใช้งาน อย่างไรก็ตามสามารถจำแนกอย่างคร่าว ๆ ได้เป็นโพรบแรงดันและโพรบกระแส ในที่นี้เราจะอธิบายเกี่ยวกับโพรบแรงดัน โดยเฉพาะอย่างยิ่ง คือ โพรบลดทอนสัญญาณ (ATTENUATOR PROBE) ซึ่งมีใช้งานโดยทั่วไป

ถ้าหากเราใช้ลวดตัวนำเพียงเส้นเดียวต่อเชื่อมระหว่างภาควงจรขาเข้าของออสซิลโลสโคปกับเป้าหมายที่ต้องการวัด อินดักแตนซ์ของลวดตัวนำจะทำให้สัญญาณความถี่สูง ๆ เกิดอาการ "วิ่งไม่หยุด" บนจอภาพ แต่หากเราใช้สายชีลด์ช่วย จะพบว่าอาการวิ่งไม่หยุดดังกล่าวจะลดลง แต่จะเกิดผลอีกด้านก็คือ สัญญาณความถี่สูงจะถูกลดทอนขนาดลงมาก ทั้งนี้เป็นผลจากค่าเก็บประจุ (CAPACITANCE) อันเกิดจากสายชีลด์กับอิมพีแดนซ์ของตัวที่จะวัดค่าซึ่งประกอบกันขึ้นเป็นวงจรกรองผ่านต่ำขึ้น โดยทั่วไปค่าเก็บประจุขาเข้าของออสซิลโลสโคปจะมีขนาดเล็กมากคือประมาณ 30 พิโคฟารัด (pF) แต่ถ้าความถี่สัญญาณที่จะวัดสูงจนทำให้เกิดปัญหาของค่าเก็บประจุจากสายชีลด์ดังกล่าวข้างต้น เราก็ไม่อาจใช้ประโยชน์จากคุณลักษณะเช่นนี้ของสโคปได้ เพื่อแก้ปัญหาสายชีลด์ที่ใช้ต่อเชื่อมระหว่างสโคปกับสิ่งที่จะวัด จึงมีการคิดค้นโพรบลดทอนสัญญาณขึ้น



รูปที่ 22 โครงสร้างของโพรบลดทอนสัญญาณ

โครงสร้างของโพรบดังกล่าวจะเป็นดังรูป 22 C_i กับ R_i เป็นค่าความเก็บประจุและความต้านทานด้านเข้าของสโคป สำหรับโพรบนั้นจะประกอบด้วย C กับ R ของวงจรลดทอนสัญญาณและทริมเมอร์คอนเดนเซอร์ C_t รวมทั้งค่าเก็บประจุของสายชีลด์ค่า C_s ความสัมพันธ์ของค่าต่าง ๆ เหล่านี้ จะเป็นดังนี้

$$CR = (C_t + C_s + C_i)\, R_i \qquad \text{----------(3)}$$

โดย C_t คือค่าทริมเมอร์คอนเดนเซอร์ที่ปรับเปลี่ยนค่าได้

ความต้านทานด้านเข้าของออสซิลโลสโคปมีค่าโดยประมาณ 1 เมกะโอห์ม ส่วนความต้านทาน R ของโพรบ 10:1 มีค่า 9 เมกะโอห์ม ค่า C ของโพรบนั้นอาจแตกต่างกันบ้างตามชนิดของโพรบ ในที่นี้จะใช้ค่า 12 พิโคปารัด (pF) ซึ่งมีใช้กันโดยทั่วไป ดังนี้นจากสมการ (3) จะได้ว่า

$$CR = 12 \times 9/1 = 108 \quad C_t + C_s + C_i \ (pF) \qquad \text{----------(4)}$$

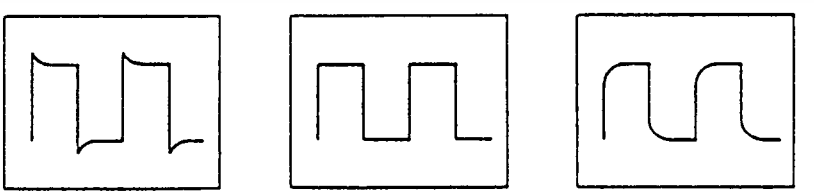
ถ้าสามารถปรับค่า C ที่โพรบให้สมการข้างต้นเป็นตริง โพรบก็จะสามารถลดทอนสัญญาณในอัตราส่วน 10:1 ได้ ในกรณีเช่นนี้ค่าเก็บประจุความต้านทานขาเข้าเมื่อมองจากปลายโพรบจะเป็นดังนี้

ค่าเก็บประจุขาเข้า : $C// (C_t + C_s + C_i) = 12// 108 = 10.8\ (pF)$ ----------(5)

ค่าความต้านทานขาเข้า : $R + R_i = 9 + 1 = 10\ M\Omega$ ----------(6)

ด้วยเหตุนี้ หากใช้โพรบลดทอนสัญญาณเช่นนี้ ตัวต่อเชื่อมระหว่างภาควงจรเข้าของสโคปกับวัตถุที่ต้องการวัดจะมีอิมพีแดนซ์สูงมาก ทำให้การวัดค่าในวงจรความถี่สูงเป็นไปได้ดีขึ้น

อย่างไรก็ตาม เนื่องจากค่าเก็บประจุขาเข้า (C_i) จะแตกต่างกันตามรุ่นของออสซิลโลสโคป ดังนั้นเพื่อที่สมการที่ 3 จะเป็นจริง จำเป็นจะต้องทำการปรับโพรบให้สอดคล้องกับออสซิลโลสโคปที่ใช้ วิธีการที่ใช้กันอย่างแพร่หลาย

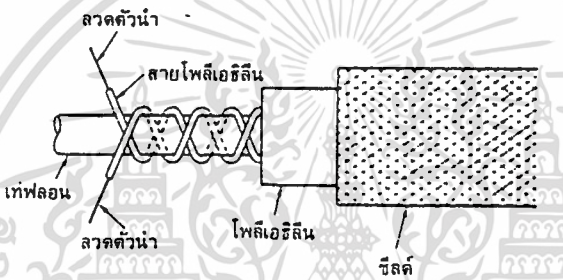


รูปที่ 23 การปรับตั้ง C_t โดยใช้รูปคลื่นสี่เหลี่ยม

และมีประสิทธิภาพดีก็โดยใช้สัญญาณสี่เหลี่ยมความถี่ 1 กิโลเฮิรตซ์เพื่อใช้ในการปรับแต่ง สัญญาณสี่เหลี่ยม 1 กิโลเฮิรตซ์ดังกล่าวสามารถดึงมาใช้ได้จากขั้วที่แผงหน้าปัดของออสซิสโลสโคปทั่วไป โดยการปรับตั้งค่า C_t จะทำให้สัญญาณสี่เหลี่ยมมีรูปร่างเปลี่ยนแปลงไปดังที่แสดงไว้ในรูปที่ 23

2.1.6 การหน่วงสัญญาณ

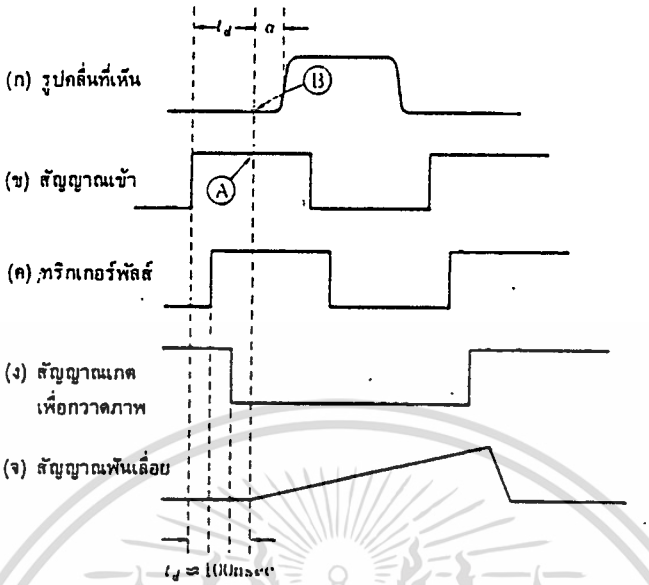
ในออสซิลโลสโคปที่มีแถบความถี่สูงกว่า 30 เมกะเฮิรตซ์มักจะมีการใช้วงจรหน่วงสัญญาณเพิ่มขึ้น องค์ประกอบสำคัญของวงจรคือ สายหน่วงเวลาหรือดีเลย์ลายน์ (DELAY LINE) สายหน่วงเวลาหรือสายเคเบิลดังกล่าวจะมีโครงสร้างดังรูปที่ 24โดยมีลวดตัวนำคู่หนึ่งพันไขว้สลับกลับทิศกัน



รูปที่ 24 โครงสร้างภายในของสายหน่วงเวลา

เมื่อใส่สายเคเบิลดังกล่าวในวงจรแกนแนวตั้ง สัญญาณขาเข้าจะถูกหน่วงให้ถึงแผ่นเบี่ยงเบนของหลอดภาพช้าไปเป็นเวลาที่แน่นอนค่าหนึ่ง การหน่วงสัญญาณดังกล่าวนี้นับว่าเป็นประโยชน์อย่างมากในการวัดคูพัลส์ความเร็วสูง โดยเฉพาะช่วงเริ่มและช่วงท้ายสัญญาณ

รูปที่ 25 แสดงให้เห็นความจำเป็นของการต้องมีวงจรหน่วงเวลาจากรูปจะเห็นว่าการ กวาดภาพโดยสัญญาณฟันเลื่อย (จ) เริ่มขึ้นภายหลังจากที่สัญญาณเข้า (ข) ป้อนให้กับแผ่นเบี่ยงเบนของหลอดภาพประมาณ 100 นาโนวินาที ทั้งนี้เป็นผลเนื่องจากเกิดความล่าช้าของสัญญาณในวงจรต่าง ๆ ขึ้น ดังแสดงเปรียบเทียบระหว่างสัญญาณเข้า (ข) ทริกเกอร์พัลส์ (ค) สัญญาณเกตเพื่อกวาดภาพ (ง) และสัญญาณฟันเลื่อย (จ) เราจะมองเห็นเฉพาะส่วนทางขวาของจุด A เท่านั้นบนหลอดภาพ แต่ถ้าหากเราสามารถหน่วงเวลาให้สัญญาณเข้ามาถึงช้าลงเป็นเวลา t_d =100 นาโนวินาที + α ภาพที่เราเห็นบนจอหลอดภาพจะเป็นดังรูป (ก) คือเริ่มตั้งแต่จุด B หรือก่อนสัญญาณเริ่มมีเล็กน้อย



รูปที่ 25

เวลาหน่วงหรือดีเลย์ไทม์ (DELAY TIME หรือ t_d) t_dจะมีค่าแตกต่างไปตามรุ่นของเครื่องซึ่งมักอยู่ในระหว่าง 100-200 นาโนวินาที กรณีของออสซิลโลสโคปทั่วไปนั้น มักมีวงจรหน่วงเวลาที่สามารถทำให้เห็นช่วงก่อนสัญญาณเข้า (หรือ ในรูปที่ 25) ประมาณไม่กี่สิบนาโนวินาที ถ้าหากจำเป็นต้องวัดดูช่วงก่อนสัญญาณเข้าที่มีค่าสูงกว่านี้ ก็คงจำเป็นต้องอาศัยการกวาดภาพแบบประวิงเวลา (ดูในหัวข้อวิธีการต่าง ๆ ในการกวาดภาพ) อย่างไรก็ตามในกรณีพัลส์ที่มีการเกิดซ้ำค่อนข้างช้า การใช้วิธีกวาดภาพแบบประวิงเวลาจะได้รูปคลื่นที่สว่างน้อยลง ซึ่งหากให้วิธีการหน่วงเวลาสัญญาณอาจจะดีกว่า

บทที่ 3

การอินเตอร์เฟสกับ IBM/PC

3.1 สัญญาณต่าง ๆ บน สล็อตของ IBM/PC

ภายใน IBM/PC ได้ออกแบบให้สามารถที่จะเพิ่มเติมวงจรอินเทอร์เฟสเข้าไปในภายหลังได้ โดยผ่านทางสล็อตที่อยู่บนเมนบอร์ด (Main Board) สำหรับสล็อตบนเมนบอร์ดนี้จะมีจำนวน 5 สล็อต (สำหรับใน IBM PC/XT จะมี 8 สล็อต จะกล่าวถึงในภายหลัง) ซึ่งแต่ละสล็อตจะมีจำนวนขาทั้งสิ้น 62 ขา แบ่งออกเป็น 2 ข้าง ๆ ละ 31 ขา ส่วนการเรียกตำแหน่งขาของสล็อตเหล่านี้จะขึ้นอยู่กับว่าขานั้นอยู่ข้างใด (ซ้ายหรือขวา) ของสล็อต โดยขาที่อยู่ทางด้านซ้ายของสล็อตจะเรียกโดยใช้อักษร "B" นำหน้าเลขตำแหน่งของขา เช่น ขา B16 ก็คือขาทางด้านซ้ายของสล็อตขาที่ 16 (นับจากทางด้านท้ายของเครื่อง)

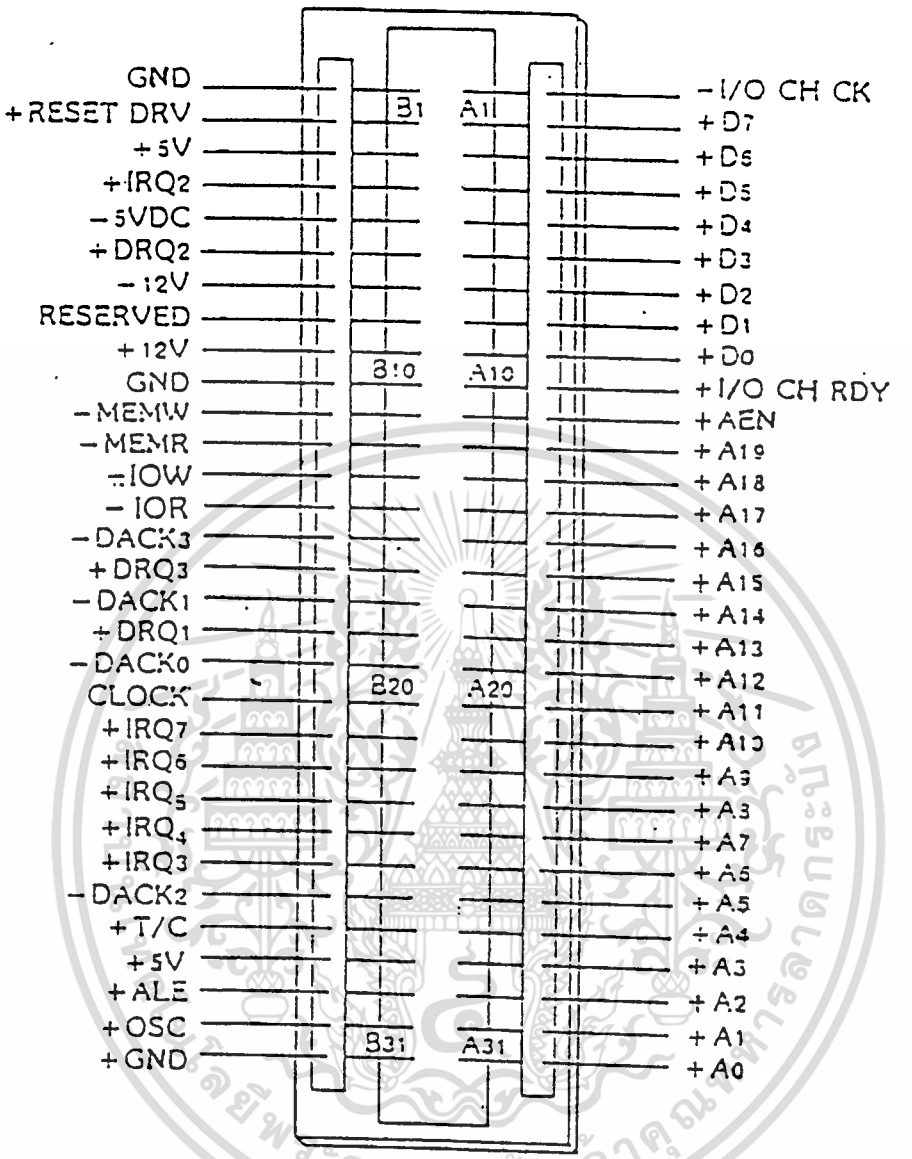
แต่ละขาของสล็อตเหล่านี้จะเชื่อมต่อกับเส้นสัญญาณต่าง ๆ บนเมนบอร์ด ทำให้การสร้างวงจรอินเทอร์เฟสกับ IBM/PC สามารถทำได้โดยสะดวก ซึ่งเส้นสัญญาณที่เชื่อมต่อกับขาของสล็อตเหล่านี้จะประกอบไปด้วย เส้นสัญญาณของบัสแอดเดรส (Address Bus), บัสข้อมูล (Data Bus), บัสควบคุมสำหรับการเขียน/อ่านข้อมูลจากหน่วยความจำ หรือพอร์ท I/O, เส้นสัญญาณสำหรับการขออินเทอร์รัพท์ของวงจรอินเทอร์เฟส, เส้นสัญญาณสำหรับการขอ DMA, สัญญาณฐานเวลา (Timing Signal) ต่าง ๆที่ใช้ในระบบ, เส้นสัญญาณแสดงการรีเฟรชหน่วยความจำ และสัญญาณสำหรับการตรวจสอบความผิดพลาด (I/O CHCK)

นอกจากเส้นสัญญาณเหล่านี้แล้ว สล็อตบนเมนบอร์ดยังเชื่อมต่อกับแหล่งจ่ายไฟต่าง ๆ ที่ใช้ในระบบอีกด้วย คือ +5Vdc, +12Vdc และ -12Vdc

<u>รายละเอียดเกี่ยวกับสัญญาณต่าง</u> ๆ

OSC (Oscillator ; ขา B30) :

ขานี้เป็นเอาท์พุทที่เชื่อมต่อกับสัญญาณคล็อกที่มีค่าความถี่สูงสุดบนเมนบอร์ด คือ 14.31818 MHz ซึ่งมีคาบเวลาประมาณ 70 nanosec. และมี Duty Cycle (ช่วงเวลาใน 1 คาบที่สัญญาณคล็อกมีลอจิกเป็น "1" หารด้วยคาบเวลาทั้งหมด) ประมาณ 50% สัญญาณคล็อกอื่น ๆ ของระบบ เช่น คล็อกที่ป้อนให้กับ 8088 หรือ ชิพซัพพอร์ทต่าง ๆ นั้นจะถูกสร้างขึ้นโดยการหารสัญญาณคล็อกนี้ อย่างไรก็ตามสิ่งหนึ่งที่จะต้องคำนึงถึงในการใช้งานสัญญาณ OSC ก็คือ สัญญาณนี้จะไม่ Synchronize กับสัญญาณอื่น ๆ บนบัสของระบบ ดังนั้นจึงไม่ควรที่จะนำสัญญาณจากขา OSC นี้ไปใช้เป็นสัญญาณคล็อกสำหรับวงจรภายนอกอื่น ๆ ที่ทำงานร่วมกับระบบ



รูปที่ 26 ขาสัญญาณต่างๆ ของ Slot PC

CLK (Clock; ขา B20) :

ขาสัญญาณนี้เป็นเอาท์พุท ซึ่งต่อกับสัญญาณคล็อกที่ถูกสร้างขึ้นโดยการหารสัญญาณ OSC ด้วย 3 ทำให้ความถี่ประมาณ 4.77 MHz (14.31818 MHz/3) หรือ มีช่วงเวลาใน 1 คาบ (ช่วงเวลาของคล็อก 1 ลูก) เท่ากับ 210 nanosec. (1/4.77 MHz) สำหรับค่า Duty Cycle ของสัญญาณนี้จะมีค่าประมาณ 1/3 คือ ใน 1 คาบจะมีช่วงเวลาที่เป็นลอจิก "1" เท่ากับ 1/3 ของคาบเวลาทั้งหมด หรือประมาณ 70 nanosec. และช่วงเวลาที่เป็นลอจิก "0" เท่ากับ 2/3 ของคาบเวลาทั้งหมด หรือประมาณ 140 nanosec. สัญญาณนี้เป็นสัญญาณที่ถูกใช้เป็นคล็อกของระบบ

RESET DRV (ขา B2) :

ขาสัญญาณนี้เป็นเอาท์พุท ซึ่งจะแอคทีฟ (ลอจิก "1") ในช่วงที่เราเริ่มจ่ายไฟให้กับระบบ และจะยังคงแอคทีฟไปจนกว่าระบบต่าง ๆ ภายใน IBM/PC จะพร้อมที่จะทำงานได้ จากนั้นสัญญาณนี้ก็จะเปลี่ยนกลับเป็นลอจิก "0" นอกจากนี้ในระหว่างการทำงานของ IBM/PC ถ้าระดับแรงดันของแหล่งจ่ายไฟตกลง สัญญาณนี้ก็จะถูกทำให้แอคทีฟเช่นกัน โดยทั่วไปแล้วสัญญาณนี้จะถูกนำไปใช้ในการรีเซ็ทวงจรอินเทอร์เฟสหรืออุปกรณ์ I/O ต่าง ๆในช่วงที่เริ่มจ่ายไฟให้กับระบบ ซึ่งจะเป็นการทำให้วงจรหรืออุปกรณ์เหล่านั้นถูกปรับให้อยู่ในสภาวะที่แน่นอน ก่อนที่จะเริ่มต้นการทำงานในระบบ (สภาวะนี้เป็นสภาวะที่เราทราบ และต้องการให้วงจรทำงานในขณะที่ระบบถูกรีเซ็ท)

AO-A19 (Address Bus; ขา A31-A12) :

ขาสัญญาณทั้ง 20 ขานี้เป็นเอาท์พุท ซึ่งใช้สำหรับกำหนดแอดเดรสของหน่วยความจำ หรืออุปกรณ์ I/O ที่ 8088 ต้องการติดต่อด้วย โดยที่สัญญาณ AO จะมีนัยสำคัญต่ำสุด (Least Significant Bit) และ A19 นี้ จะถูกกำหนดโดย 8088 ในระหว่างขบวนการอ่าน/เขียนข้อมูลลงในหน่วยความจำหรืออุปกรณ์ I/o แต่ในช่วงของขบวนการ DMA นั้น DMA - Controller จะเป็นผู้กำหนดค่าแอดเดรสบนบัสแอดเดรสเอง (ในระหว่างนี้ 8088 จะถูกตัดออกจากระบบ)

จะเห็นได้ว่าจำนวนเส้นแอดเดรสนี้จะมีอยู่ 20 เส้น ซึ่งสามารถที่จะอ้างแอดเดรสของหน่วยความจำได้ถึง 1Mbyte แต่อย่างไรก็ตามจะมีแอดเดรสบางแอดเดรสที่ถูกใช้งาน โดย IBM/PC อยู่ก่อนแล้ว คือแอดเดรสของหน่วยความจำ RAM บนเมนบอร์ดที่ถูกใช้โดยระบบ จำนวน 64Kbyte (สำหรับ IBM PC/XT จะเป็นจำนวน 256Kbyte) และแอดเดรสสำหรับหน่วยความจำ ROM อีก 48Kbyte ซึ่งถูกจัดในช่วงของแอดเดรสบนสุดใน 1 Mbyte คือ OFCOOH จนถึง OFFFFFH (สำหรับ IBM PC/XT จะเป็น 64Kbyte)

สำหรับการอ้างแอดเดรสของพอร์ท I/O นั้น จะใช้เส้นแอดเดรสเพียง 16 เส้น คือ AO-A15 ซึ่งจะทำให้อ้างแอดเดรสของพอร์ทได้ 64K พอร์ท โดยผ่านทางชุดคำสั่ง IN และ OUT ส่วนเส้นแอดเดรสที่เหลือคือ A16-A19 นั้นจะไม่ถูกใช้งาน อย่างไรก็ตามภายใน IBM/PC จะใช้เส้นแอดเดรสในการอ้างแอดเดรสของพอร์ทเพียง 10 เส้น คือจาก A0-A9 และค่าแอดเดรสที่ใช้งานจะต้องอยู่ในช่วง 0200H จนถึง 03FFH เท่านั้น

D0-D7 (Data Bus; ขา A9-A2) :

ขาสัญญาณนี้จะเป็นแบบ Bi-Directional ซึ่งต่อกับบัสข้อมูลของระบบ เพื่อทำหน้าที่ในการส่งผ่านข้อมูลระหว่างพอร์ท I/O กับ IBM/PC โดยบิท DO จะมีนัยสำคัญต่ำสุดและบิท D7 จะมีนัยสำคัญสูงสุด

สำหรับในบัสไซเคิลของการเขียนข้อมูลที่สร้างขึ้นโดย 8088 นั้น ข้อมูลจะถูกส่งออกมาบนบัสข้อมูล ก่อนที่สัญญาณ $\overline{IOW}$ (ในกรณีที่ต้องการส่งข้อมูลให้กับพอร์ท) หรือ $\overline{MEMW}$ (ในกรณีที่ต้องการส่งข้อมูลให้กับหน่วยความจำ) จะเปลี่ยนจากลอจิก "0" เป็นลอจิก "1" (ขอบขาขึ้น) ซึ่งโดยทั่วไปขอบขาขึ้นของสัญญาณ $\overline{IOW}$ หรือ $\overline{MEMW}$ นี้ จะถูกใช้เพื่อสั่งให้พอร์ท I/O หรือหน่วยความจำที่มีแอดเดรสตรงกับค่าแอดเดรสบนบัสแอดเดรสนั้นรับข้อมูลไปเก็บไว้

สำหรับในบัสไซเคิลของการอ่านข้อมูลที่สร้างขึ้นโดย 8088 นั้น พอร์ท I/O หรือหน่วยความจำที่ถูกอ้างถึงจะต้องส่งข้อมูลออกมาบนบัสข้อมูล ก่อนที่สัญญาณ $\overline{IOR}$ (ในกรณีที่ต้องการอ่านข้อมูลจากพอร์ท) หรือ $\overline{MEMR}$ (ในกรณีที่ต้องการอ่านข้อมูลจากหน่วยความจำ) จะเปลี่ยนจากลอจิก "0" เป็นลอจิก "1" (ขอบขาขึ้น)

ALE (Address Latch Enable ; ขา B28) :

ขาสัญญาณนี้เป็นสัญญาณเอาท์พุทที่ 8288 Bus Controller สร้างขึ้นเพื่อใช้สำหรับแสดงการเริ่มต้นของบัสไซเคิล และแสดงให้อุปกรณ์ภายนอกทราบว่าแอดเดรสที่ 8088 ต้องการจะติดต่อด้วยนั้นถูกส่งออกมาบนบัสแอดเดรสแล้ว โดยที่สัญญาณ ALE นี้จะเปลี่ยนจากลอจิก "1" เป็น "0" เมื่อค่าแอดเดรสที่ถูกต้องถูกส่งออกมาบนบัสข้อมูลเรียบร้อยแล้ว ดังนั้นขอบขาลงของสัญญาณ ALE นี้จะถูกใช้ในการแลทช์ค่าแอดเดรสจากบัสแอดเดรส/ข้อมูล (address/Data Bus; AD0-AD7) ของ 8088 ทำให้สามารถแยกค่าแอดเดรส (A0-A19) และข้อมูล (A0-A7) ออกจากกันได้ อย่างไรก็ตามสัญญาณ ALE จะแอดทีฟเฉพาะในบัสไซเคิลที่สร้างขึ้นโดย 8088 เท่านั้น โดยจะไม่แอดทีฟในระหว่างขอบวนการ DMA

$\overline{\text{I/O CHCK}}$ (I/O Channel Check; ขา A1) :

ขาสัญญาณนี้เป็นอินพุทที่ใช้ในการแสดงความผิดพลาดเกี่ยวกับพาริตี้ ที่เกิดขึ้นในการทำงานของวงจรอินเทอร์เฟสหรืออุปกรณ์ I/O เมื่อขาสัญญาณนี้ได้รับลอจิก "0" จะทำให้ 8088 ถูกอินเทอร์รัพท์แบบ Non-Maskable (NMI) อย่างไรก็ตามเราสามารถที่จะกำหนดให้วงจรภายในของ IBM/PC ทำการขออินเทอร์รัพท์ (เมื่อได้รับสัญญาณ $\overline{\text{I/O CHCK}}$) หรือไม่ก็ได้ โดยการกำหนดลอจิกของบิทข้อมูลของพอร์ทที่ควบคุมการขออินเทอร์รัพท์แบบ NMI คือบิท D7 ของพอร์ท OOAOH ถูกเซ็ทเป็น "1" ก็จะทำให้วงจรภายนอกขออินเทอร์รัพท์แบบ NMI ได้ (Enable) แต่ถ้าบิท D7 ของพอร์ท OOAOH ถูกเซ็ทเป็น "0" ก็จะเป็นการดิสเอเบิล (Disable) การขออินเทอร์รัพท์แบบ NMI ดังนี้

Enable : ใช้คำสั่ง OUT ส่งข้อมูล 80H ไปยังพอร์ท OOAOH

Disable : ใช้คำสั่ง OUT ส่งข้อมูล OOH ไปยังพอร์ท OOAOH

และเนื่องจากยังมีอุปกรณ์อื่นที่สามารถของอินเทอร์รัพท์แบบ NMI ได้อีก ดังนั้นซอฟท์แวร์ที่ใช้งานจะต้องสามารถตรวจสอบว่าการขออินเทอร์รัพท์นั้นเกิดขึ้นจากแหล่งใดได้ด้วย

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

ขาสัญญาณนี้เป็นอินพุทที่ใช้เพิ่มช่วงเวลาในบัสไซเคิลในกรณีที่เกิดขึ้นนั้น ไม่สามารถทำงานทันตามช่วงเวลาปกติของบัสไซเคิลนั้น ๆ ได้ (ช่วงเวลาของบัสไซเคิลที่เกี่ยวกับหน่วยความจำใช้ช่วงเวลาเท่ากับช่วงเวลาของคล็อก 4 ลูก หรือ 840 nanosec. ในขณะที่บัสไซเคิลที่เกี่ยวกับ I/O จะใช้ช่วงเวลาเท่ากับช่วงเวลาของคล็อก 5 ลูกหรือ 1.05 usec.)

เมื่ออุปกรณ์ I/O หรือหน่วยความจำต้องการที่จะเพิ่มช่วงเวลาในบัสไซเคิลให้นานขึ้นอีกนั้น จะสามารถทำได้โดยการป้อนลอจิก "0" ให้กับขา I/O CHRDY ในช่วงเวลาที่ I/O หรือ หน่วยความจำที่ถูกกำหนดนั้น ได้รับสัญญาณจากการดีโค้ดแอดเดรส และสัญญาณ $\overline{\text{MEMR}}$, $\overline{\text{MEMW}}$, $\overline{\text{IOR}}$ หรือ IOW แอดทีฟ

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

ขาสัญญาณทั้ง 6 นี้เป็นขาอินพุทที่ใช้สำหรับการขออินเทอร์รัพท์จาก 8088 โดยสัญญาณเหล่านี้จะต่อเข้ากับ 8259A บนเมนบอร์ดโดยตรง โปรแกรมในส่วน BIOS ของ IBM/PC จะทำการโปรแกรม 8259A ให้ IRQ2 มีลำดับความสำคัญสูงสุด (Highest Priority) และ IRQ7 มีลำดับความสำคัญต่ำสุด ในกรณีที่มีการขออินเทอร์รัพท์เกิดขึ้นคือ ระดับลอจิกที่ขา IRQ ขาใดขาหนึ่งถูกเปลี่ยนจากลอจิก "0" เป็นลอจิก "1) (ขอบขาขึ้น) 8259A ก็จะทำการส่งสัญญาณ INT ให้กับ 8088 เพื่อทำการขออินเทอร์รัพท์

สิ่งสำคัญในการขออินเทอร์รัพท์โดยผ่านทาง IRQ2-IRQ7 นี้ ก็คืออุปกรณ์ที่ทำการขออินเทอร์รัพท์โดยผ่านทาง IRQ ขาใดก็จะต้องรักษาระดับสัญญาณที่ขา IRQ นั้น ให้แอดทีฟ (ลอจิก "1") อยู่จนกว่าจะได้รับสัญญาณ INTA (Interrupt Acknowledge) จาก 8088 เสียก่อน ถ้าไม่เช่นนั้นการขออินเทอร์รัพท์จะถูกยกเลิก และอินเทอร์รัพท์ Level 7 (IRQ7) ก็จะถูกสร้างขึ้นโดยอัตโนมัติ ไม่ว่าการขออินเทอร์รัพท์จะถูกยกเลิกนั้นจะเป็นการขออินเทอร์รัพท์ใน Level หรือขาใด

แต่อย่างไรก็ตามสัญญาณ INTA นี้จะไม่ถูกต่อออกมาที่ขาของสล็อตด้วย ดังนั้นโปรแกรมที่ทำการตอบสนองต่อการขออินเทอร์รัพท์ (Interrupt Service Routine) จะต้องทำการรีเซ็ทสัญญาณ IRQ เอง โดยใช้คำสั่ง OUT ไปยังพอร์ท I/O ที่เกี่ยวข้อง

$\overline{\text{IOR}}$ (I/O Read; ขา B14) :

ขาสัญญาณนี้เป็นเอาท์พุทแอดทีฟที่ลอจิก "0" ที่สร้างขึ้นโดย 8288 Bus Controller เพื่อใช้ในการแสดงว่าบัสไซเคิลที่เกิดขึ้นนี้ เป็นบัสไซเคิลของการอ่านข้อมูลจากพอร์ท I/O เพื่อให้พอร์ท I/O ที่

มีแอดเดรสตรงกับแอดเดรสบนบัสแอดเดรสนั้นส่งข้อมูลออกมาบนบัสข้อมูล โดยข้อมูลจะต้องถูกส่งออกมาบนบัสข้อมูลก่อนขอบขาขึ้นของสัญญาณ $\overline{\text{I OR}}$ ประมาณ 30 nanosec. เพื่อให้มั่นใจได้ว่า 8088 สามารถรับข้อมูลได้ถูกต้อง สำหรับในขบวนการ DMA 8237A-5 DMA Controller จะทำการสร้างสัญญาณ $\overline{\text{IOR}}$ เอง โดยที่ค่าแอดเดรสที่อยู่บนบัสแอดเดรสจะเป็นค่าแอดเดรสของหน่วยความจำ (แทนที่จะเป็นแอดเดรสของพอร์ท I/O) ที่พอร์ท I/O ที่ขอ DMA ต้องการจะนำข้อมูลไปเก็บ การที่พอร์ทใดจะส่งข้อมูลออกมาบนบัสข้อมูลนั้น จะอาศัยสัญญาณ DACK จาก DMA controller เป็นตัวกำหนด เช่นกรณีที่สัญญาณ DACK1 แอคทีฟก็จะแสดงว่าพอร์ท I/O ที่จะต้องส่งข้อมูลออกมาบนบัสข้อมูลก็คือพอร์ท I/O ที่ขอ DMA ผ่านทางแชนแนลที่ 1 (DRQ1) เป็นต้น

$\overline{\text{IOW}}$ (I/O Write; ขา B13) :

ขาสัญญาณนี้เป็นเอาท์พุทแอคทีฟที่ลอจิก "0" ซึ่งถูกสร้างขึ้นโดย 8288 Bus Controller เพื่อใช้แสดงว่าบัสไซเคิลที่เกิดขึ้นนี้เป็นบัสไซเคิลของการเขียนข้อมูลลงบนพอร์ท I/O เพื่อให้พอร์ท I/O ที่มีแอดเดรสตรงกับแอดเดรสบนบัสแอดเดรสนั้น รับข้อมูลที่อยู่บนบัสข้อมูลไปเก็บไว้ อย่างไรก็ตาม เนื่องจากในช่วงเวลาที่สัญญาณ $\overline{\text{IOW}}$ นี้แอคทีฟ (ลอจิก "0") นั้นข้อมูลบนบัสข้อมูลอาจจะยังไม่สมบูรณ์ ดังนั้นในการออกแบบจึงควรใช้ขอบขาขึ้นของสัญญาณ $\overline{\text{IOW}}$ แทนขอบขาลงในการทำให้พอร์ท I/O ที่เกี่ยวข้องรับข้อมูลไปเก็บไว้ เพื่อให้ข้อมูลบนบัสข้อมูลสมบูรณ์เสียก่อน สำหรับในขบวนการ DMA นั้น DMA-Controller จะทำการสร้างสัญญาณ $\overline{\text{IOW}}$ เอง โดยที่ค่าแอดเดรสที่อยู่บนบัสแอดเดรสจะเป็นค่าแอดเดรสของหน่วยความจำที่พอร์ท I/O ที่ขอ DMA ต้องการจะอ่านข้อมูล

$\overline{\text{MEMW}}$ (Memory Write; ขา B11) :

ขานี้เป็นเอาท์พุทแอคทีฟที่ลอจิก "0" ซึ่ง 8288 Bus Controller สร้างขึ้นในระหว่างบัสไซเคิลในการเขียนข้อมูลลงในหน่วยความจำของ 8288 สัญญาณ $\overline{\text{MEMW}}$ นี้จะถูกส่งออกมาเพื่อให้หน่วยความจำที่แอดเดรสตรงกับค่าแอดเดรสตรงกับค่าแอดเดรสบนบัสแอดเดรสนั้น ทำการรับข้อมูลที่อยู่บนบัสข้อมูลไปเก็บไว้ โดยทั่วไปหน่วยความจำจะรับข้อมูลในช่วงขอบขาขึ้นของสัญญาณ MEMW

สำหรับในระหว่างขบวนการ DMA นั้น 8237A-5 DMA-Controller จะทำการควบคุมบัสต่าง ๆ ของระบบแทน 8088 และสัญญาณ $\overline{\text{MEMW}}$ จะถูกใช้ในบัสไซเคิลของการเขียนข้อมูลลงในหน่วยความจำ (ข้อมูลถูกส่งจากอุปกรณ์ I/O ไปให้กับหน่วยความจำ)

$\overline{\text{MEMR}}$ (Memory Read; ขา B12):

ขานี้เป็นเอาท์พุทจาก 8288 ซึ่งสัญญาณนี้จะแอคทีฟ (ลอจิก "0") ในระหว่าง Bus Cycly ของการอ่านข้อมูลจากหน่วยความจำของ 8088 เพื่อให้หน่วยความจำที่มีแอดเดรสตรงกับค่าแอดเดรสบนบัสแอดเดรสนั้น ทำการส่งข้อมูลออกมาบนบัสข้อมูล โดยหน่วยความจำนั้นจะต้องส่งข้อมูลออก

มาในช่วงเวลา 30 nanosec. ก่อนที่สัญญาณ $\overline{MEMR}$ จะกลับเป็นลอจิก "1" ทั้งนี้ก็เพื่อให้ 8088 ได้รับข้อมูลที่ถูกต้อง

สำหรับในระหว่างขบวนการ DMA นั้น DMA-Controller จะทำการควบคุมบัสต่างๆ ของระบบแทน 8088 และสัญญาณ $\overline{MEMR}$ จะถูกใช้ในบัสไซเคิลของการอ่านข้อมูลจากหน่วยความจำ (ข้อมูลถูกส่งจากหน่วยความจำไปให้กับอุปกรณ์ I/O)

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

ขาสัญญาณทั้งสามนี้เป็นสัญญาณอินพุทแอคทีฟที่ลอจิก "1" ซึ่งอุปกรณ์ภายนอกสามารถใช้ในการขอ DMA จากระบบ โดยการป้อนระดับสัญญาณลอจิก "1" ให้กับขา DRQ ขาใดขาหนึ่ง (ขา DRQ ทั้งสามนี้จะต่อเข้ากับ DRQ1-DRQ3 ของ 8237A-5)

เมื่อ 8237A-5 ได้รับสัญญาณนี้แล้วก็จะตรวจสอบว่ามีการขอ DMA ในแชนเนลที่มีลำดับความสำคัญ (Priority) สูงกว่าหรือไม่ ถ้าไม่มีก็จะทำการขอ DMA จาก 8088 และตอบรับการขอ DMA จากอุปกรณ์ภายนอก (สัญญาณ $\overline{DACK}$ ของแชนแนลที่ขอ DMA จะแอคทีฟ) แต่ถ้ามี 8237A-5 ก็จะทำการขอ DMA ให้กับแชนแนลที่มีลำดับความสำคัญสูงกว่าก่อนแล้วจึงทำการขอ DMA ให้กับแชนแนลที่มีลำดับความสำคัญต่ำกว่า ภายใน ROM BIOS ของ IBM/PC จะโปรแกรม 8237A-5 ให้ DRQ1 มีลำดับความสำคัญสูงสุดและ DRQ3 มีลำดับความสำคัญสูงสุดและ DRQ3 มีลำดับความสำคัญต่ำสุด ดังนั้นถ้ามีการขอ DMA ของอุปกรณ์ภายนอกผ่านทางแชนแนลที่ 1 ก่อน จากนั้นเมื่อเสร็จจากขบวนการ DMA ของแชนแนลที่ 1 แล้ว จึงจะทำการขอ DMA ให้กับแชนแนลที่ 2

อย่างไรก็ตาม 8237A-5 ยังมีแชนแนลสำหรับการขอ DMA อยู่อีก 1 แชนแนล คือ แชนแนลที่ 0 (DRQO) ซึ่งในความเป็นจริงแล้วแชนแนลนี้จะมีลำดับความสำคัญที่สูงกว่าแชนแนลที่ 1 แต่จะไม่ถูกต่อออกมายังขาของสล็อต เนื่องจาก IBM/PC จะใช้แชนแนลที่ 0 นี้ในการรีเฟรชหน่วยความจำที่เป็น Dynamic RAM

ในการขอ DMA นั้นสัญญาณ DRQ นี้ จะต้องแอคทีฟอยู่ในช่วงระยะเวลาหนึ่งเท่านั้นถ้าสัญญาณนี้แอคทีฟอยู่นานเกินไป จำทำให้เกิดขบวนการ DMA ขึ้นมากกว่า 1 ขบวนการได้ สำหรับวงจรที่ขอ DMA โดยทั่วไปแล้วจะใช้สัญญาณตอบรับการขอ DMA หรือสัญญาณ $\overline{DACK}$ ของแชนแนลที่ขอ DMA นั้น ในการรีเซ็ทสัญญาณ DRQ เช่นอุปกรณ์ภายนอกที่ขอ DMA ผ่านทางแชนแนลที่ 1 (DRQ1) ก็จะคอยตรวจสอบการตอบรับการขอ DMA จากสัญญาณ $\overline{DACK}$ ของแชนแนลที่ 1 ($\overline{DACK1}$) เมื่อได้รับสัญญาณจาก $\overline{DACK1}$ แล้วก็จะรีเซ็ทสัญญาณ DRQ1 (เปลี่ยนจากลอจิก "1" เป็น "0")

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

สัญญาณทั้ง 4 นี้เป็นเอาท์พุทแอคทีฟที่ลอจิก "0" ซึ่ง 8237A-5 สร้างขึ้นเพื่อแสดงให้วงจรภายนอกที่ขอ DMA ทราบว่าการขอ DMA นั้นได้รับการตอบสนองแล้ว และ 8237A-5 จะเข้าสู่ขบวนการ DMA เพื่อให้การส่งผ่านข้อมูลระหว่างอุปกรณ์ I/O ที่ขอ DMA กับหน่วยความจำเกิดขึ้นได้โดยตรง (คือไม่ต้อยผ่าน 8088) โดยสัญญาณ $\overline{\text{DACK}}$ นี้จะแอคทีฟในแชนแนลใดก็ขึ้นอยู่กับว่าขบวนการ DMA ที่จะเกิดขึ้นนั้น เป็นการตอบสนองต่อการขอ DMA ในแชนแนลใด เช่นถ้าขบวนการ DMA ที่จะเกิดขึ้นเป็นการตอบสนองต่อการขอ DMA ในแชนแนลที่ 2 (DRQ2) สัญญษร $\overline{\text{DACK2}}$ ก็จะแอคทีฟ เป็นต้น

ดังที่ได้กล่าวแล้วว่าสัญญาณ DRQ0 นั้น จะไม่ถูกต่อออกมายังขาของสล็อต ดังนั้นวงจรอินเทอร์เฟสจึงไม่สามารถจะขอ DMA ผ่านทางแชนแนล 0 ได้ แต่สัญญาณ $\overline{\text{DACK0}}$ จะถูกต่อออกมายังสล็อตด้วย (ขา B19), ทั้งนี้ก็เพื่อที่จะแสดงให้วงจรอินเทอร์เฟสต่างๆ ทราบว่าขบวนการ DMA ที่เกิดขึ้นในช่วงเวลาที่ $\overline{\text{DACK0}}$ แอคทีฟนั้น เป็นขบวนการที่ใช้สำหรับการรีเฟรชหน่วยความจำที่เป็น Dynamic RAM ซึ่งวงจรอินเทอร์เฟสที่ใช้หน่วยความจำประเภทนี้สามารถจะนำไปใช้ในการรีเฟรช Dynamic RAM ที่อยู่ในวงจรได้

โดยที่การรีเฟรชหน่วยความจำนั้นจะเกิดขึ้นในทุกๆ 15.12 usec. หรือ ทุกๆ 72 คล็อก ดังนั้นสัญญาณ $\overline{\text{DACK0}}$ นี้ก็จะแอคทีฟในทุกๆ 15.12 usec. ด้วย

AEN (Address Enable; ขา A11) :

สัญญาณนี้เป็นเอาท์พุทที่ใช้ในการแสดงว่าบัสไซเคิลที่เกิดขึ้นในช่วงเวลาที่สัญญาณ AEN แอคทีฟ (ลิจิก "1") นั้น เป็นบัสไซเคิลของขบวนการ DMA

สำหรับบนเมนบอร์ดของ IBM/PC นั้น จะใช้สัญญาณนี้ในการดิสเอเบิล (Disable) 8288 Brs Controller และจะใช้ดิสเอเบิลพอร์ท I/O ต่างๆที่ไม่เกี่ยวข้องกับขบวนการ DMA ที่เกิดขึ้นนี้ ที่จำเป็นต้องทำเช่นนี้ก็เพราะในระหว่างขบวนการ DMA นั้น 8237A-5 จะส่งแอดเดรสของหน่วยความจำออกมาบนบัสแอดเดรส และจะทำให้สัญญาณ $\overline{\text{IOR}}$ หรือ $\overline{\text{IOW}}$ แอคทีฟด้วย ดังนั้นถ้าไม่ทำการดิสเอเบิลพอร์ท I/O ที่ไม่เกี่ยวข้องไว้ ก็อาจจะทำให้พอร์ท I/O ที่มีแอดเดรสตรงกับค่าแอดเดรสบนบัสแอดเดรส (ซึ่งเป็นแอดเดรสของหน่วยความจำ) นั้น ทำการอ่านหรือส่งข้อมูลออกมาบนบัสข้อมูลทำให้เกิดความผิดพลาดขึ้นได้

T/C (Terminal count; ขา B27) :

สัญญาณนี้ถูกสร้างขึ้นจากการนำเอาสัญญาณเอาท์พุทที่ขา EOP ของ 8237A-5 มากลับลอจิก (โดยใช้เกท Inverter) ทำให้สัญญาณ T/C นี้แอคทีฟที่ลอจิก "1"

สำหรับสัญญาณนี้จะแอคทีฟเมื่อจำนวนไบท์ในการส่งผ่านข้อมูลของขบวนการ DMA ในแชนแนลใดแชนแนลหนึ่ง ครบตามจำนวนที่กำหนดไว้ (ดูรายละเอียดในบทที่ 7 "การจัด DMA ของระบบ") โดยทั่วไปแล้วสัญญาณที่จะถูกใช้ในการสิ้นสุดขบวนการ DMA ที่ทำการส่งผ่านข้อมูลเป็นบล็อก เนื่องจากสัญญาณนี้จะแอคทีฟโดยไม่แสดงว่าเป็นสัญญาณของแชนแนลใด ดังนั้นจึงต้องทำการนำสัญญาณ T/C นี้ผ่านเกท Inverter แล้วนำไป OR กับสัญญาณ DACK เพื่อให้สามารถทราบได้ว่าสัญญาณ T/C ที่เกิดขึ้นนั้นเป็นสัญญาณของแชนแนลใด สำหรับในแชนแนลที่ 0 นั้น สัญญาณ T/C จะแอคทีฟในช่วงเวลาที่คงที่คือ ทุก ๆ 990.804 millisec. ซึ่งก็คือช่วงเวลาที่ใช้ในการรีเฟรชหน่วยความจำขนาด 64Kbyte นั่นเอง

บัสของแหล่งจ่ายไฟของระบบ

+5Vdc (ขา B3 และ B29) :

ขาทั้งสองนี้ต่อกับแหล่งจ่ายไฟ DC +5V ของระบบ โดยจะมีค่าความเที่ยงตรง (Regulated) $\pm$ 5% คืออยู่ในช่วง +4.75 ถึง +5.25 Vdc

+12Vdc (ขา B9) :

ขานี้จะต่อกับแหล่งจ่ายไฟ DC +12V ของระบบ โดยจะมีค่าความเที่ยงตรง (Regulated) $\pm$ 5% คืออยู่ในช่วง +11.4 ถึง +12.6 Vdc

-5Vdc (ขา B5) :

ขานี้จะต่อกับแหล่งจ่ายไป DC -5V ของระบบ โดยจะมีค่าความเที่ยงตรง (Regulated) $\pm$10% คืออยู่ในช่วง -5.5 ถึง -4.5 Vdc

-12Vdc (ขา B7) :

ขานี้จะต่อกับแหล่งจ่ายไฟ DC -12V ของระบบ โดยจะมีค่าความเที่ยงตรง (Regulated) $\pm$ 10% คืออยู่ในช่วง -13.2 ถึง -10.8 Vdc

GND (ขา B1, B10 และ B31) :

ขาทั้งสามนี้จะต่อเข้ากับกราวด์ (Ground) ของระบบ

การจัดสัญญาณบนสล็อตของ IBM PC/XT

สำหรับใน IBM PC/XT นั้นจะมีสล็อตสำหรับเชื่อมต่อกับวงจรภายนอกได้มากขึ้น คือ ใน IBM PC/XT จะทำการเพิ่มจำนวนสล็อตบนเมนบอร์ดขึ้นเป็น 8 สล็อต จากเดิมที่มีอยู่เพียง 5 สล็อตบน IBM PC โดยการจัดสัญญาณต่างๆในทั้ง 8 สล็อตจะยังคงเหมือนกับใน IBM PC เพียงแต่สัญญาณต่าง ๆ ที่จะถูกส่งออกมายังขาของสล็อตที่ 8 นั้น จะถูกต่อผ่านวงจรขับกระแส (Buffer) ก่อน และในสล็อตที่ 8 นี้ขา B8 จะถูกใช้งานด้วย โดยจะถูกใช้เป็นขา $\overline{\text{CARD SLCTD}}$ (หรือ Card Selected) ซึ่งขาสัญญาณนี้จะเป็นสัญญาณอินพุทจากวงจรภายนอกที่เสียบอยู่บนสล็อตที่ 8 เพื่อให้วงจรบนเมนบอร์ดทราบว่าคาร์ดที่อยู่บนสล็อตนี้ถูกเลือกใช้งานอยู่ ซึ่งจะทำให้ Driver บนเมนบอร์ดทำการอ่านหรือส่งข้อมูลไปยังสล็อตที่ 8

3.2 การจัดโครงสร้างแอดเดรสของระบบบัส

ในการจัดโครงสร้างระบบ พีซีเอที จัดโครงสร้างบัสสล็อตด้วยสัญญาณเดิมเมื่อเทียบกับเอ็กซ์ที และเพิ่มสล็อตเล็กขยายบัสส่วนเกินออกไป การจัดระบบจึงเพิ่มขยายต่อจากเดิมทั้งสิ้นในเกือบทุกส่วน ลองพิจารณาส่วนของการจัดแอดเดรสของระบบพีซีเอทีที่มีซีพียู 80286 80286 มีสายบัสแอดเดรสเป็นจำนวน 24 เส้น ซึ่งต่อกับหน่วยความจำจริงได้ถึง 16 เมกะไบต์ ไอบีเอ็มเอทีได้จัดโครงสร้างหน่วยความจำของระบบไว้ดังตารางที่ 1

แอดเดรส	หน่วยความจำ	ฟังก์ชั่น
000000-07FFFF	หน่วยความจำระบบ 512 กิโลไบต์	หน่วยความจำระบบ
080000-09FFFF	128 กิโลไบต์	หน่วยความจำขยายเพิ่มอีก จนครบ 640 กิโลไบต์
0A0000-OBFFFF	128 กิโลไบต์ วิดีโอ	สงวนไว้สำหรับการแสดงผล
0C0000-0DFFFF	128 กิโลไบต์ หน่วยความจำ ROM สำหรับขยายระบบ I/O	สงวนไว้สำหรับรอมบนบอร์ดอะแดปเตอร์ I/O
0E0000-OEFFFF	สงวนไว้ 64 กิโลไบต์สำหรับระบบ	ที่ซ้ำกับแอดเดรส FE0000
OF0000-0FFFFF	สงวนไว้ 64 กิโลไบต์ ไบออสรอม	ที่ซ้ำกับแอดเดรส FF0000
100000-FDFFFF	15 เมกะไบต์	ส่วนขยายหน่วยความจำ
FE0000-FFFFFF	128 กิโลไบต์	ซ้ำกับแอดเดรส FE000

ตารางที่ 1 โครงสร้างหน่วยความจำของระบบ

จะเห็นว่ามีแอดเดรสบางส่วนที่ซ้ำกัน ทั้งนี้เพราะเมื่อระบบที่ออกแบบมาใช้หน่วยความจำเต็มที่ เช่น ในโอเอส 2 จะมีการย้ายไบออสไปไว้ในแอดเดรสสุดท้าย ในส่วน FE0000-FFFFFF ได้

อนึ่ง ระบบพีซีเอทีมีโครงสร้างที่ปรับปรุงมาจากเอ็กซ์ทีด้วยนั้น ถ้าหากได้ทำความเข้าใจจากระบบเอ็กซ์ที่มาก่อนจะทำให้เข้าใจเอทีได้ดียิ่งขึ้น สำหรับบทต่อไปจะเป็นการกล่าวถึงการเชื่อมต่อแรม และรอม การจัดการดีเอ็มเอ และควบคุมอินเตอร์รัพต์

บทที่ 4

8254 Programable Peripheral Interface

8254 เป็นชิพซัพพอร์ทที่ทำหน้าที่ในการสร้างฐานเวลา ภายใน 8254 จะประกอบด้วยแชนแนลที่ทำงานเป็นอิสระต่อกันอยู่ 3 แชนแนล ซึ่งการที่แชนแนลทั้ง 3 ของ 8254 จะทำงานตามที่เราต้องการได้นั้น เราจะต้องทำการโปรแกรมแชนแนลเหล่านี้เสียก่อน

สำหรับการโปรแกรม 8254 นั้นจะทำได้ในลักษณะเดียวกับชิพซัพพอร์ทอื่น ๆ คือ ใช้คำสั่ง OUT ส่งข้อมูล (คำสั่ง) ให้กับรีจิสเตอร์ต่าง ๆ ของ 8254 ซึ่งรีจิสเตอร์ของ 8254 จะแบ่งออกได้เป็น 2 ประเภทคือ รีจิสเตอร์ Mode Control และ รีจิสเอตร์ Counter

รีจิสเตอร์ Counter

ภายใน 8254 จะมีรีจิสเตอร์ Counter ซึ่งเป็นรีจิสเตอร์ขนาด 16 บิทอยู่ 3 ตัว โดยข้อมูลในรีจิสเตอร์แต่ละตัวจะควบคุมช่วงเวลาของเอาท์พุทในแต่ละแชนแนล (รีจิสเตอร์ Counter ทั้ง 3 นี้จะทำงานเป็นอิสระต่อกัน)

เมื่อ 8254 เริ่มต้นการทำงานในแชนแนลใดแล้ว ข้อมูลในรีจิสเตอร์ Counter ของแชนแนลนั้นจะถูกลดค่าลง 1,2 หรือ 3 (ขึ้นอยู่กับโหมดการทำงาน) เมื่อแชนแนลนั้นได้รับคล็อกแต่ละลูก ข้อมูลในรีจิสเตอร์ counter จะถูกลดลงจนมีค่าเป็น "0" (Terminal Count) จากนั้นจึงจะเกิดการเปลี่ยนแปลงทางด้านเอาท์พุทและการทำงานของแชนแนลนั้น สำหรับการลดค่าของรีจิสเตอร์ Counter นี้ อาจจะลดค่าลงแบบ BCD หรือ Binary ก็ได้ ขึ้นอยู่กับการเซ็ทข้อมูลในบิท DO ของรีจิสเตอร์ Mode Control

เนื่องจากรีจิสเตอร์ Counter เป็นรีจิเตอร์ขนด 16 บิท แต่บัสข้อมูลของ 8254 มีขนาดเพียง 8 บิท ดังนั้นการเขียนหรืออ่านข้อมูลจากรีจิสเตอร์ Counter จึงต้องทำครั้งละ 8 บิทด้วย โดยข้อมูลในรีจิสเตอร์ Counter จะถูกแบ่งออกเป็น 2 ส่วนคือ 8 บิทบน (Most Significant Bit ; D15-D8) และ 8 บิทล่าง (Least Significant Bit; D7-D0) ส่วนการที่จะกำหนดว่าข้อมูลที่ส่งให้กับรีจิสเตอร์ Counter นี้เป็นข้อมูลสำหรับ 8 บิทบนหรือ 8 บิทล่างนั้น เราจะกำหนดได้โดยการเซ็ทค่าในบิท D5 และ D4 ของรีจิสเตอร์ Mode Control นอกจากนี้เรายังสามารถใช้ข้อมูลในบิททั้งสองนี้ในการกำหนดว่าการโปรแกรมรีจิสเตอร์ Counter นี้จะทำการโปรแกรมเฉพาะไบท์ใดไบท์หนึ่ง (เฉพาะ 8 บิทบนหรือ 8 บิทล่าง) หรือ โปรแกรมทั้ง 8 บิทบนและ 8 บิทล่างได้อีกด้วย (ดูรายละเอียดใน "รีจิสเตอร์ Mode Control")

อีกสิ่งหนึ่งที่จะต้องคำนึงถึงก็คือ ในการโปรแกรมรีจิสเตอร์ Counter นั้น รีจิสเตอร์ Counter จะรับข้อมูลที่เราส่งไปให้ก็ต่อเมื่อสัญญาณที่ขา CLK ถูกเปลี่ยนจากลอจิก "0" เป็น "1" (ขอบขาขึ้น)

และเปลี่ยนจากลอจิก "1" กลับเป็นลอจิก "0" (ขอบขาลง) อีกครั้งเสียก่อน (จะกล่าวถึงการโปรแกรมรีจิสเตอร์นี้ในภายหลัง

วิธีในการโปรแกรมรีจิสเตอร์ Mode Control นี้จะทำได้โดยใช้คำสั่ง OUT ส่งข้อมูลไปยังพอร์ทของ 8254 ที่มีแอดเดรส AO และ A1 เป็น "1" ทั้งคู่ (สำหรับการโปรแกรม 8254 จะกล่าวโดยละเอียดในภายหลัง) สำหรับหน้าที่ของบิทต่าง ๆ ในรีจิสเตอร์ Mode Control นั้น จะแสดงได้ดังนี้

บิท	D7	D6	D5	D4	D3	D2	D1	D0
หน้าที่	SC1	SC0	RL1	RL0	M2	M1	MO	BCD

ตารางที่ 2 แสดง Bit Assignment ของ 8254

บิท D7-D6 (SC1,SCO) : ข้อมูลในบิททั้งสองนี้จะใช้สำหรับเลือกแชนแนลที่ต้องการ ซึ่งจะแสดงได้ดังนี้ (เนื่องจาก 8254 มีเพียง 3 แชนแนล ดังนั้นกรณีที่ข้อมูลในบิท SC1 และ SCO เป็น "1" ทั้งคู่จึงไม่มีผลในการเลือกแชนแนลใด ๆ)

SC1	SC0	แชนแนลที่ถูกเลือก
0	0	0
0	1	1
1	0	2
1	1	-

ตารางที่ 3 แสดง Bit Assignment ของ 8254

บิท D5-D4 (RL1,RL0) : ใช้ในการกำหนดไบท์ (8 บิทบนหรือ 8 บิทล่าง) ของข้อมูลที่ต้องการจะอ่านจากรีจิสเตอร์ Counter ซึ่งเป็นรีจิสเตอร์ขนาด 16 บิท และใช้ในการกำหนดว่าการโปรแกรมรีจิสเตอร์ Counter นี้จะทำการโปรแกรมเฉพาะไบท์ใดไบท์หนึ่งหรือโปรแกรมทั้ง2 ไบท์ นอกจากนี้ยังใช้ในการแลทช์ (Latch) ค่าในรีจิสเตอร์ Counter ได้อีกด้วย (ดูรายละเอียดในหัวข้อ "การโปรแกรมและอ่านข้อมูลจากรีจิสเตอร์ของ 8254") สำหรับผลจากการเซ็ทข้อมูลภายในบิท D5 และ D4 นั้นจะแสดงได้ดังนี้

RL1	RLO	
0	0	ทำการแลทช์ค่าในรีจิสเตอร์-เคาน์เตอร์
0	1	อ่าน/เขียนเฉพาะข้อมูลเฉพาะใน 8 บิทล่าง (least-significant Bit)
1	0	อ่าน/เขียนเฉพาะข้อมูลเฉพาะใน 8 บิทบน (Most-Significant Bit)
1	1	อ่าน/เขียนเฉพาะข้อมูลทั้ง 16 บิท โดยเริ่มจาก 8 บิทล่างก่อน จากนั้นจึงอ่าน/เขียนข้อมูลใน 8 บิทบน

ตารางที่ 4 แสดง Bit Assignment ของ 8254

บิท D3-D1 (M2-M0) : ข้อมูลในบิทนี้ใช้สำหรับเลือกโหมดการทำงานให้กับแชนแนลที่เรากำหนดในบิท SC1 และ SC0

สำหรับโหมดการทำงานของแชนแนลต่าง ๆ ใน 8254 จะมี 6 โหมด แต่เราใช้เฉพาะ Mode 3 เพียงโหมดเดียวเท่านั้น

M2	M1	MO	โหมดการทำงาน
0	0	0	โหมด 0 : Interrupt on Terminal Count
0	0	1	โหมด 1 : Programmable One-Shot
0	1	0	โหมด 2 : Rate Generator
0	1	1	โหมด 3 : Square Wave Generator
1	0	0	โหมด 4 : Software Triggered Strobe
1	0	1	โหมด 5 : Hardware Triggered Strobe

ตารางที่ 5 แสดง Bit Assignment ของ 8254

โหมด 3 : Square Wave Rate Generator

การทำงานในโหมดนี้จะมีลักษณะคล้ายกับในโหมด 2 คือ เป็นการหารความถี่ของสัญญาณคล็อกที่ป้อนให้กับแชนแนลที่ทำงานในโหมดนี้ด้วย N (ค่า Counter ที่โหลดให้กับรีจิสเตอร์ Counter) แต่จุดที่แตกต่างกันก็คือ ช่วงเวลาที่เอาท์พุทเป็นลอจิก "1" และ ลอจิก "0" ใน 1 คาบจะเท่ากัน (หรือเทียบเท่ากัน) ในขณะที่โหมด 2 นั้น ช่วงเวลาที่เอาท์พุทเป็นลอจิก "0" จะนานเท่ากับช่วงเวลาของ

ที่มีความถี่เท่ากับ ความถี่ของสัญญาณคล็อกที่ป้อนให้กับแชนแนลนั้นหารด้วย N โดยมีช่วงเวลาที่เป็นลอจิก "1" เท่ากับช่วงเวลาของสัญญาณคล็อก (N+1)/2 ลูก และช่วงเวลาที่เป็นลอจิก "0" เท่ากับช่วงเวลาของคล้อก (N-2)/2 ลูก ดังที่ได้กล่าวมาแล้ว ตัวอย่างเช่น ถ้าสัญยาณคล็อกที่ป้อนให้กับแชนแนลที่ทำงานในโหมด 3 มีความถี่ 1.19318 MHz หรือมีคาบเวลา (ช่วงเวลาของคล้อก 1 ลูก) ประมาณ 838 nanosec. ถ้าเราทำการโหลดค่า 04H (เลขคู่) ให้กับรีจิสเตอร์ Counter ก็จะทำให้เอาท์พุทที่ได้จากแชนแนลนี้มีช่วงเวลาที่เป็นลอจิก "1" และ "0" เท่ากัน คือ เท่ากับช่วงเลาของคล็อก 4/2=2 ลูก หรือประมาณ 838 x 2 = 1.676 usec. แต่ถ้าเราทำการโหลดค่า 05H (เลขคี่) ให้กับรีจิสเตอร์ Counter แล้วก็จะทำให้เอาท์พุทที่ได้จากแชนแนลนี้มีช่วงเวลาที่เป็นลอจิก "1" เท่ากับช่วงเวลาของคล็อก (5+1)/2 = 3 ลูก หรือ ประมาณ 838x3 = 2.514 usec. และช่วงเวลาที่เป็นลอจิก "0" เท่ากับช่วงเวลาของคล็อก (5-2)/2 = 2 ลูก หรือประมาณ 838x2 = 1.676 usec. สำหรับไดอะแกรมข้างล่างนี้ จะแสดง Timing Diagram ในกรณีของตัวอย่างที่ได้กล่าวถึงในทั้งสองกรณี

รูปที่ 27 ไดอะแกรมเวลาการทำงานในโหมด 3 ของ 8254

บิท DO: ข้อมูลในบิทนี้จะใช้สำหรับกำหนดว่าการลดค่าของข้อมูลในรีจิสเตอร์ Counter นั้นจะทำการลดในลักษณะของ BCD หรือ Binary โดยถ้าข้อมูลในบิทนี้ถูกเซ็ทเป็น "1" ค่าของข้อมูลในรีจิสเตอร์ Counter จะถูกลดลงในลักษณะของ BCD (Binary Coded Decimal) และถ้าข้อมูลในบิทนี้ถูกเซ็ทเป็น "0" ค่าของข้อมูลในรีจิสเตอร์ Counter ก็จะถูกลดลงในลักษณะของ Binary

การโปรแกรมรีจิสเตอร์ของ 8254

ในการโปรแกรมแชนแนลทั้ง 3 ของ 8254 เพื่อกำหนดโหมดการทำงาน, จำนวนไบท์ของข้อมูลที่ต้องการจะส่งให้กับรีจิสเตอร์ Counter, ช่วงเวลาของเอาท์พุท และรูปแบบการลดค่าของข้อมูลในรีจิสเตอร์ Counter (BCD หรือ Binary) ของแต่ละแชนแนลเราจะต้องทำการโปรแกรมรีจิสเตอร์ของแชนแนลนั้น 2 รีจิสเตอร์ Mode Control และรีจิสเตอร์ counter เสียก่อน

รีจิสเตอร์ Mode Control

คล็อกเพียง 1 ลูกเท่านั้นไม่ว่าค่าของ Counter จะมีค่าเปลี่ยนแปลงไป อย่างไรก็ตามการทำงานของแชนแนลที่ทำงานในโหมด 3 แบ่งออกได้เป็น 2 ลักษณะตามค่าของ Counter ที่โหลดให้กับรีจิสเตอร์ Counter (ในที่นี้จะแทนค่า Counter ด้วยตัวแปร N ดังนี้)

1. ถ้า N เป็นตัวเลขคู่ :

เอาท์พุทจะมีช่วงเวลาที่เป็นลอจิก "1" นานเท่ากับช่วงเวลาของคล็อก N/2 ลูก และจะเป็นลอจิก "0" อยู่นานเท่ากับช่วงเวลาของคล็อก N/2 ลูกเช่นกัน ซึ่งก็คือใน 1 คาบจะมีช่วงเวลานานเท่ากับช่วง เวลาของคล็อก N ลูก โดยมีค่า Duty Cycle (ช่วงเวลาที่เป็นลอจิก "1" หารด้วยช่วงเวลาใน 1 คาบ) เท่ากับ 50% นั่นเอง

สำหรับการทำงานของแชนแนลที่อยู่ในโหมด 3 และค่า N ที่โหลดให้กับรีจิสเตอร์ Counter เป็นเลขคู่นี้ จะอธิบายโดยย่อได้ดังนี้ คือ หลังจากเริ่มการทำงานเอาท์พุทของแชนแนลนี้เป็นลอจิก "1" จากนั้นทุกครั้งที่คล็อกที่ป้อนให้กับแชนแนลนี้เปลี่ยนระดับลอจิกจาก "1" เป็น "0" (ขอบขาลง) ค่าในรีจิสเตอร์ counter จะถูกลดค่าลง 2 ซึ่งการลดค่าในรีจิสเตอร์ Counter นี้ จะดำเนินไปจนกระทั่งรีจิสเตอร์ Counter มีข้อมูลเป็น "0" และรีจิสเตอร์ Counter จะถูกเปลี่ยนข้อมูลเป็นค่า Counter เริ่มต้นที่เราส่งให้ก่อนที่จะเริ่มต้นการทำงาน (ในที่นี้คือค่า N) โดยอัตโนมัติ จากนั้นเมื่อสัญญาณคล็อกเปลี่ยนจากลอจิก "1" เป็น "0" (ขอบขาลง) ค่าในรีจิสเตอร์ Counter ก็จะถูกลดลง 2 อีกจนกระทั่งค่าในรีจิสเตอร์ Counter เป็น "0" เอาท์พุทจึงจะกลับเป็นลอจิก "1" อีกครั้ง ค่า N ก็จะถูกโหลดให้กับรีจิสเตอร์ Counter อีกโดยอัตโนมัติ จากนั้นจึงทำงานตามขั้นตอนต่าง ๆตามที่ได้กล่าวมานั้นต่อไป

2. ถ้า N เป็นเลขคี่ :

ในกรณีนี้ช่วงเวลา 1 คาบของเอาท์พุทจะมีช่วงเวลาที่เป็นลอจิก "1" อยู่นานเท่ากับช่วงเวลาของคล็อก (N+1)/2 ลูก และจะมีช่วงเวลาที่เป็นลอจิก "0" นานเท่ากับ ช่วงเวลาของคล็อก (N-2)/2 ลูก สำหรับการทำงานของแชนแนลที่ถูกโปรแกรมให้ทำงานในโหมด 3 และ ค่า N ที่โหลดให้กับรีจิสเตอร์ Counter เป็นเลขคี่นั้น จะมีดังนี้คือ ขณะที่เริ่มต้นการทำงานในโหมด 3 เอาท์พุทจะมีระดับลอจิกเป็น "1" และ เมื่อได้รับคล็อกลูกแรกข้อมูลในรีจิสเตอร์ Counter จะถูกลดลง "1" จากนั้นเมื่อแชนแนลนี้ได้รับคล็อกลูกต่อไป ข้อมูลในรีจิสเตอร์ Counter ก็จะถูกลดลงครั้งละ "2" จนกระทั่งข้อมูลในรีจิสเตอร์ Counter เป็น "0" ในช่วงนี้เอาท์พุทจะถูกเปลี่ยนระดับลอจิกเป็น "0" และข้อมูลในรีจิสเตอร์ counter จะถูกโหลดด้วยค่าเริ่มต้น (หลังจากที่รีจิสเตอร์ counter ถูกโหลดด้วยค่าเริ่มต้น) ข้อมูลในรีจิสเตอร์ Counter จะถูกลดลง "3" และเมื่อแชนแนลนี้ได้รับคล็อกลูกต่อไป ข้อมูลในรีจิสเตอร์ Counter ก็จะถูกลดลงครั้งละ "2" จนกระทั่งค่าในรีจิสเตอร์ counter เป็น "0" แชนแนลนี้ก็จะเริ่มต้นการทำงานตามขั้นตอนแรกใหม่ต่อไป (เอาท์พุทเปลี่ยนกลับเป็นลอจิก "1" และโหลดค่า N ให้กับรีจิสเตอร์ Counter) ด้วยวิธีการนี้จึงทำให้เอาท์พุทที่ได้จากแชนแนลที่ทำงานในโหมด 3 เป็นสัญญาณคล็อก

ในการโปรแกรมรีจิสเตอร์ Mode Control ของแชนแนลทั้ง 3 ของ 8254 นั้น เราไม่จำเป็นต้องโปรมแกรมเรียงตามลำดับ กล่าวคือ ไม่จำเป็นต้องโปรแกรมรีจิสเตอร์ Mode Control ของแชนแนลที่ 0 ก่อน จากนั้นจึงโปรแกรมรีจิสเตอร์แชนแนลที่ 1 แล้วจึงตามด้วยแชนแนลที่ 2 แต่เราอาจจะโปรแกรมแชนแนลใดก่อนก็ได้ เช่น เราอาจจะโปรแกรมรีจิสเตอร์ Mode Control ของแชนแนลที่ 1 ก็ได้ (การจะเลือกโปรแกรมรีจิสเตอร์ Mode Control ของแชนแนลใดนั้น เราสามารถจะกำหนดได้โดยการเซ็ทบิท SCO และ SC1 ของรีจิสเตอร์ Mode Control ให้ตรงกับแชนแนลที่ต้องการ)

หลังจากการโปรแกรมรีจิสเตอร์ Mode Control ในแชนแนลใดแล้ว ถ้าเราต้องการโปรแกรมรีจิสเตอร์ Counter ของแชนแนบนั้นก็สามารถจะทำได้ หรือ อาจจะทำการโปรแกรมรีจิสเตอร์ Mode Control ของแชนแนลอื่น ๆ จนครบทั้ง 3 แชนแนลก่อนก็ได้ สำหรับลำดับของการโปรแกรมรีจิสเตอร์ Counter นั้น จะมีลักษณะเดียวกับการโปรแกรมรีจิสเตอร์ Mode Control คือเราจะเลือกโปรแกรมรีจิสเตอร์ Counter ของแชนแนบใดก่อนก็ได้ โดยไม่ขึ้นกับลำดับของการโปรแกรมรีจิสเตอร์ Mode Control เช่นเราอาจจะโปรแกรมรีจิสเตอร์ Mode Control ของแชนแนล 1 ก่อนแชนแนล 2 แต่โปรแกรมรีจิสเตอร์ Counter ของแชนแนล 2 ก่อนแชนแนล 1 ก็ได้

อย่างไรก็ตาม การโปรแกรมรีจิสเตอร์ Mode Control และรีจิสเตอร์ Counter นั้นยังมีข้อแตกต่างกันอยู่บ้าง แต่ก่อนที่จะกล่าวถึงข้อแตกต่างนี้จะอธิบายถึงการจัดแอดเดรสของ 8254 เสียก่อน สำหรับ 8254 นี้จะมีขาที่ใช้สำหรับต่อกับบัสแอดเดรสของระบบอยู่ 2 ขา คือ ขา A1 และ A0 ซึ่งจะทำให้ 8254 ใช้อดเดรสทั้งสิ้น 4 แอดเดรส คือ แอดเดรสที่มีบิท A1 และ A0 เป็น "00", "01" และ "11" ตามลำดับ สำหรับใน IBM/PC นั้น 0041H (บิท A1, เป็น "0" : บิท A0 เป็น "1"), 0042H (บิท A1 เป็น "1" ; บิท A0 เป็น "0") และ 0043H (บิท A1 เป็น "1" ; บิทA0 เป็น "1") แอดเดรสทั้ง 4 ของ 8254 นี้จะถูกใช้ในการโปรแกรมรีจิสเตอร์ Mode Control และ รีจิสเตอร์ Counter ซึ่งจะทำได้โดยการใช้คำสั่ง OUT ส่งข้อมูลไปยังแอดเดรสต่าง ๆ เหล่านี้ ในตารางข้างล่างจะแสดงถึงการใช้งานแอดเดรสทั้ง 4 ในการโปรแกรมรีจิสเตอร์ Mode Control และรีจิสเตอร์ Counter

แอดเดรส (ฐาน 16)	A1	A0	รีจิสเตอร์ที่ถูกโปรแกรม
300H	0	0	รีจิสเตอร์ Counter แชนแนล 0
301H	0	1	รีจิสเตอร์ Counter แชนแนล 1
302H	1	0	รีจิสเตอร์ Counter แชนแนล 2
303H	1	1	รีจิสเตอร์ Mode Control

ตารางที่ 8 Channel Selected Program

จากตารางข้างต้นนั้น เราจะเห็นข้อแตกต่างในการโปรแกรมรีจิสเตอร์ Mode Counterol และ รีจิสเตอร์ Counter ได้อย่างชัดเจน กล่าวคือ การโปรแกรมรีจิสเตอร์ Mode Control ของทั้ง 3 แชนแนล จะทำโดยการส่งข้อมูลผ่านไปยังแอดเดรสของ 8254 เพียงแอดเดรสเดียวคือแอดเดรสที่มีบิท A1 และ A0 เป็น "1" ทั้งคู่ (ใน IBM/PC คือ แอดเดรส 0043H และทำการเลือกแชนแนลโดยเซ็ทข้อมูลในบิท SC1 และ SC0 ให้ตรงกับแชนแนลที่ต้องการ แต่การโปรแกรมรีจิสเตอร์ Counter ของทั้ง 3 แชนแนล จะทำได้โดยการส่งข้อมูลไปยังแอดเดรสสำหรับรีจิสเตอร์เหล่านั้นโดยตรง คือ แอดเดรสที่มีบิท A1 และ A0 เป็น "00" (ใน IBM/PC คือ แอดเดรส 0040H) สำหรับแชนแนล 0, "01" (ใน IBM/PC คือ แอดเดรส 0041 H) สำหรับแชนแนล 1, "10" (ใน IBM/PC คือแอดเดรส 0042H) สำหรับแชนแนล 2

การอ่านข้อมูลจากรีจิสเตอร์ Counter ของ 8254

ในระหว่างการทำงานของแต่ละแชนแนลนั้น เราอาจจะมีความจำเป็นต้องอ่านข้อมูล (ค่า Counter) จากรีจิสเตอร์ counter ของแชนแนลที่กำลังทำงานอยู่นั้นด้วย ซึ่งการอ่านข้อมูลในระหว่างนี้อาจจะทำให้เกิดความผิดพลาดของข้อมูลที่ 8254 ส่งออกมาได้ ถ้าหากว่าข้อมูลในรีจิสเตอร์ Counter กำลังถูกลดค่าอยู่ ดังนั้นเราจึงจำเป็นต้องใช้วิธีการบางอย่างเข้าช่วย เพื่อป้องกันปัญหาที่อาจจะเกิดขึ้นได้ ซึ่งสามารถจะแบ่งออกได้เป็น 2 วิธีดังนี้

1. หยุดการนับของแชนแนลที่ต้องการจะอ่านข้อมูลนั้นไว้ชั่วคราว โดยการควบคุมสัญญาณที่ป้อนให้กับขา Gate ของแชนแนลนั้น (ทำให้เป็นลอจิก "0") หรือโดยการควบคุมวงจรภายนอกไม่ให้ทำการส่งสัญญาณคล็อกให้กับแชนแนลนั้นได้ จากนั้นเราก็สามารถจะทำการอ่านข้อมูลจากรีจิสเตอร์ Counter ได้ สำหรับวิธีการอ่านข้อมูลจากรีจิสเตอร์ Counter นั้น เราจะทำได้โดยการใช้คำสั่ง IN อ่านข้อมูลจากอดเดรสของ 8254 ซึ่งมีอยู่ 4 แอดเดรสดังที่ได้กล่าวมาแล้ว โดยแอดเดรสทั้ง 4 นี้จะใช้ในการอ่านรีจิสเตอร์ Counter ของแชนแนลต่าง ๆ ดังตาราง

บิท A1	บิท A0	แอดเดรสใน IBM/PC	รีจิสเตอร์ที่ถูกอ่าน
0	0	0040	รีจิสเตอร์-เคาน์เตอร์ แชนแนล 0
0	1	0041	รีจิสเตอร์-เคาน์เตอร์ แชนแนล 1
1	0	0042	รีจิสเตอร์-เคาน์เตอร์ แชนแนล 2
1	1	00043	------

ตารางที่ 9 IBM/PC Channel Selected

ในกรณีที่แชนแนลที่เราต้องการจะอ่านค่าจากรีจิสเตอร์ Counter นั้น ถูกโปรแกรมให้การอ่านข้อมูลจากรีจิสเตอร์ Counter ต้องอ่านทีละ 2 ไบท์แล้ว (ข้อมูลในบิท RL1 และ RL0 ของรีจิสเตอร์ Mode Control ถูกเซ็ทเป็น "1" ทั้งคู่) เราจะต้องทำการอ่านข้อมูลโดยการอ่านครั้งแรกเป็นการอ่านไบท์ที่เป็น 8 บิทล่าง (D7-D0; LSB) และการอ่านครั้งที่สองเป็นการอ่านไบท์ที่เป็น 8 บิทบน (D15-D8; MSB) ของรีจิสเตอร์ Counter (การอ่านทั้ง 2 ครั้งทำได้โดยการใช้คำสั่ง IN อ่านข้อมูลจากแอดเดรสของ 8254 ที่ตรงกับรีจิสเตอร์ Counter ของแชนแนลที่ต้องการ) ซึ่งในกรณีเช่นนี้ถ้าเรายังอ่านข้อมูลไม่ครบทั้ง 2 ไบท์แล้ว เราจะไม่สามารถใช้คำสั่ง OUT ในการโปรแกรมรีจิสเตอร์ Counter ของแชนแนลนี้ได้

2. ในกรณีที่เราต้องการจะอ่านข้อมูลจากรีจิสเตอร์ Counter โดยไม่จำเป็นต้องการทำงานของแชนแนลนั้นไว้ ก็จะทำได้โดยการแลทช์ค่าของข้อมูลจากรีจิสเตอร์ Counter ไว้ก่อนที่จะทำการอ่านข้อมูลจากรีจิสเตอร์ Counter นั้น สำหรับการแลทช์ค่า Counter นี้ จะทำได้โดยการใช้คำสั่ง OUT ส่งข้อมูลที่มีบิท D5 และ D4 (บิท RL1 และ RLO เป็น "00" ในฐานสอง) ให้กับรีจิสเตอร์ Mode Control ของแชนแนลที่ต้องการ (การกำหนดแชนแนลจะทำได้โดยกำหนดทางบิท SC1 และ SCO ดังที่ได้กล่าวในตอนต้น) ซึ่งจะทำให้ค่า Counter ในรีจิสเตอร์ Counter ถูกแลทช์เก็บเอาไว้ จากนั้นเมื่อเราทำการอ่านค่า counter จากรีจิสเตอร์ Counter นี้ 8254 ก็จะส่งข้อมูลที่แลทช์ไว้ออกมายังบัสข้อมูลของระบบ

จะเห็นได้ว่าการใช้วิธีหลังนี้จะทำให้เราสามารถอ่านข้อมูลจากรีจิสเตอร์ Counter ได้ โดยไม่จำเป็นต้องหยุดการทำงานของแชนแนลนั้นไว้ก่อน สำหรับวิธีในการอ่านข้อมูลจากรีจิสเตอร์ Counter นั้จะเหมือนกับในกรณีแรกทุกประการ

สำหรับตารางที่ 10 นี้จะแสดงการจัดเรียงบิทของรีจิสเตอร์ Mode Control ในกรณีที่ใช้สำหรับการแลช์ค่า Counter ของรีจิสเตอร์ Counter (การโปรแกรมรีจิสเตอร์ Mode Control ในลักษณะนี้จะไม่มีผลกับโหมดการทำงานที่ได้เซ็ทไว้ คือ บิท D3-D0 จะไม่มีผลหรือหน้าที่ใด ๆ)

A0,A1 = 11

D7	D6	D5	D4	D3	D2	D1	D0
SC1	SC0	0	0	X	X	X	X

SCJ,SCO - Specity counter to be latched.

D5,D4 - 00 designates counter latching operation.

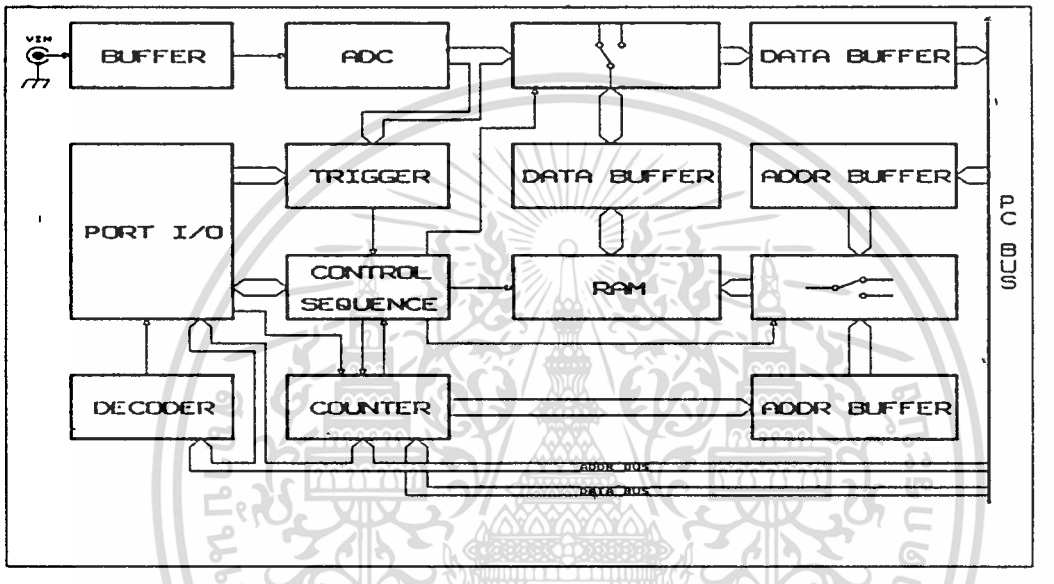
X - Don't care.

ตารางที่ 10

บทที่ 6

การทำงานของ ฮาร์ดแวร์

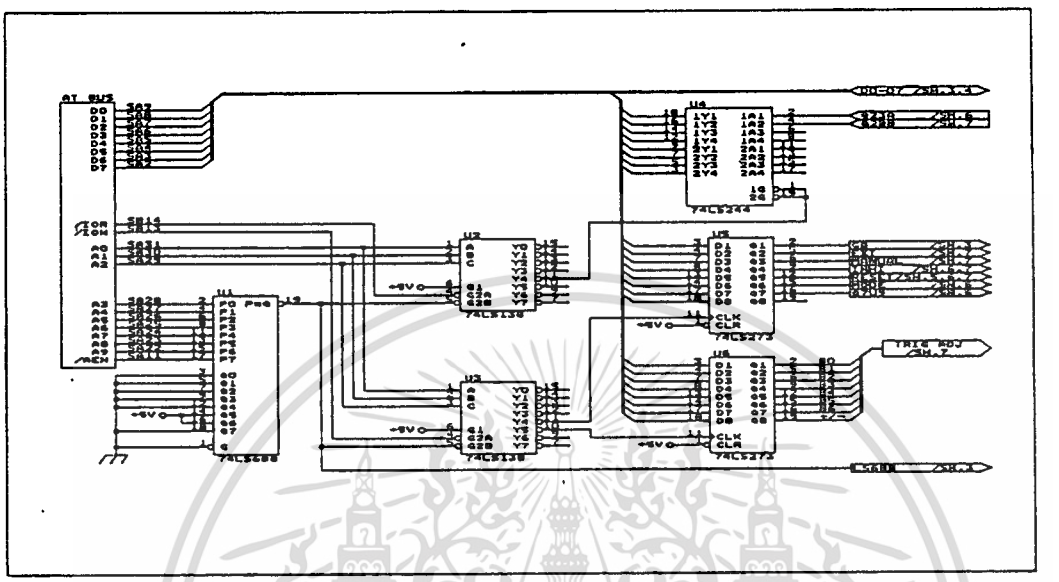
6.1 Block Diagram



รูปที่ 29 แสดง Block Diagram ของ ADC Card

จาก Block Diagram สัญญาณ I/P ผ่านเข้ามาทาง Buffer เพื่อป้องกัน A/D เสียหาย ในกรณีที่อินพุทมี Voltage มากเกินไป ผ่านเข้ามาทาง A/D และ A/D จะทำการ Converse สัญญาณ Analog ให้เป็น สัญญาณ Digital ไปเข้า Data Buffer แต่อีกส่วนหนึ่งผ่านไปยัง Block ของ Triger ในส่วน Triger จะทำการ Compare กับ Vtrig ที่เราตั้งเอาไว้โดยผ่าย Port I/O ถ้าสัญญาณที่ตั้ง Vtrig เท่ากับ Data ของ ADC ที่ Converse แล้ว จะส่งสัญญาณไปวงจร Control Sequence วรจรนี้จะส่งสัญญาณไป Select ให้ Data ของ A/D ต่อกับ RAM และ Address ของ RAM ต่อกับ Buffer และส่งสัญญาณไปที่ RAM เพื่อให้ RAM อยู่ใน Mode write พร้อมกับส่งสัญญาณให้ Counter ทำการนับ และ Write ข้อมูลลง RAM เมื่อ Counter นับจนครบ 32k (เท่ากับ Memory ของ RAM) Counter จะส่งสัญญาณไห้วงจร Control ผ่านไปยัง I/O Port จากนั้น I/O Port จะ write ข้อมูลเพื่อให้วงจร Control Sequence ทำหน้าที่ Select จาก Data ของ ADC ที่ต่อกับ RAM ไปเป็น DATA ของ RAM เข้ากับ PC ส่วน Address ก็จะสลับจาก Address ของ RAM ที่ต่อกับ Counter นั้นไปต่อ Address RAM เข้ากับ PC ไปแสดงผลที่หน้าจอ

6.2 วงจร Port Decoder



รูปที่ 30 วงจร Port Decoder

วงจรนี้เป็นวงจรที่ใช้ในการ Decode Port เพื่อที่เราจะได้ควบคุม Hardware (Analog to Digital Card) ได้อย่างถูกต้องตรงตามความต้องการ โดยจากวงจรเราใช้ U1 (74LS688) ซึ่งเป็น IC 8 bit Comparator ในการ Compare Address ของ AT BUS (ที่ต่อเข้ากับ P0-P7 ของ U1) กับค่า Logic ที่เราตั้งไว้ที่ Q0-Q7 โดยเมื่อ P กับ Q เท่ากัน Bit ต่อ Bit U1 ก็จะให้ Output P=Q Active Low ออกมาเพื่อไป Select U2 และ U3 (74LS138) ซึ่งเป็น Decoder และ U12 ซึ่งเป็น IC ที่ใช้สำหรับหาร Clock

AEN	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
P7	P6	P5	P4	P3	P2	P1	P0			
Q7	Q6	Q5	Q4	Q3	Q2	Q1	Q0			
0	1	1	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	1
0	1	1	0	0	0	0	0	0	1	0
0	1	1	0	0	0	0	0	0	1	1
0	1	1	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	0	1	0	1
0	1	1	0	0	0	0	0	1	1	0
0	1	1	0	0	0	0	0	1	1	1

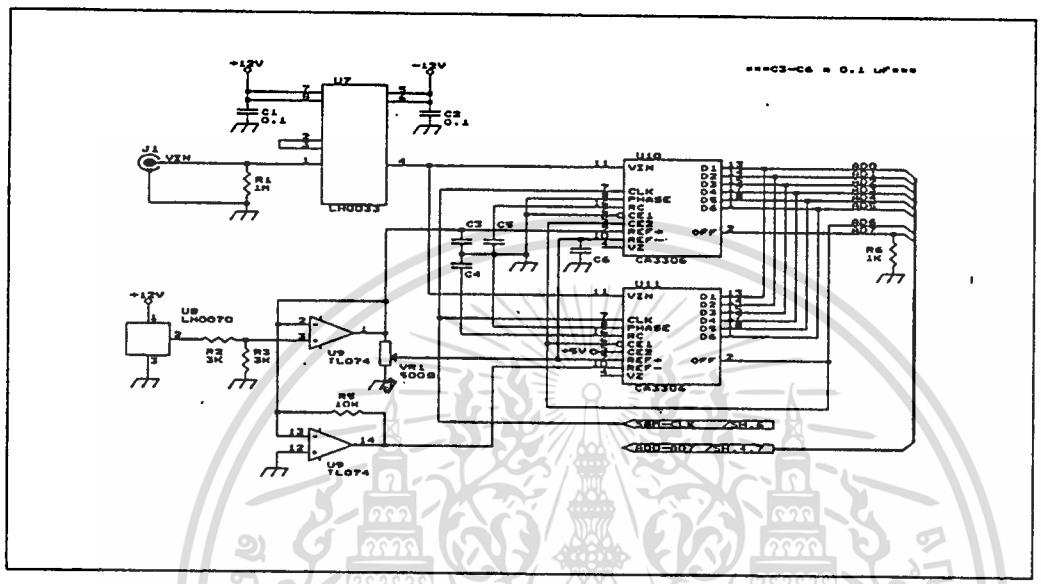
ตารางที่ 11 Port Decoder

จากวงจรซึ่งสามารถเขียนออกมาเป็นตารางในตารางที่ 11 จะเห็นว่าวงจรนี้สามาถ Decode Port ที่อยู่ในช่วง 300H-307H รวม 8 Port ซึ่งแต่ละ Port ก็จะมีหน้าที่ต่าง ๆกันออกไปคือ Port #300 H-303H ใช้ในการ Program U12 (8254) ซึ่งเป็นตัวหาร Clock Port #304H ใช้ในการ Read Status และ Control Sequence การทำงานของ Hardware Port #305H ใช้ในการปรับจุด Trig ในการเริ่มเก็บสัญญาณ

Port No.	Used for
300H	Used for 8254 Programming
301H	Used for 8254 Programming
302H	Used for 8254 Programming
303H	Used for 8254 Programming
304H	Used for read status and Sequence Control
305H	Used for trigger
306H	not used
307H	not used

ตารางที่ 12 Port Assignment

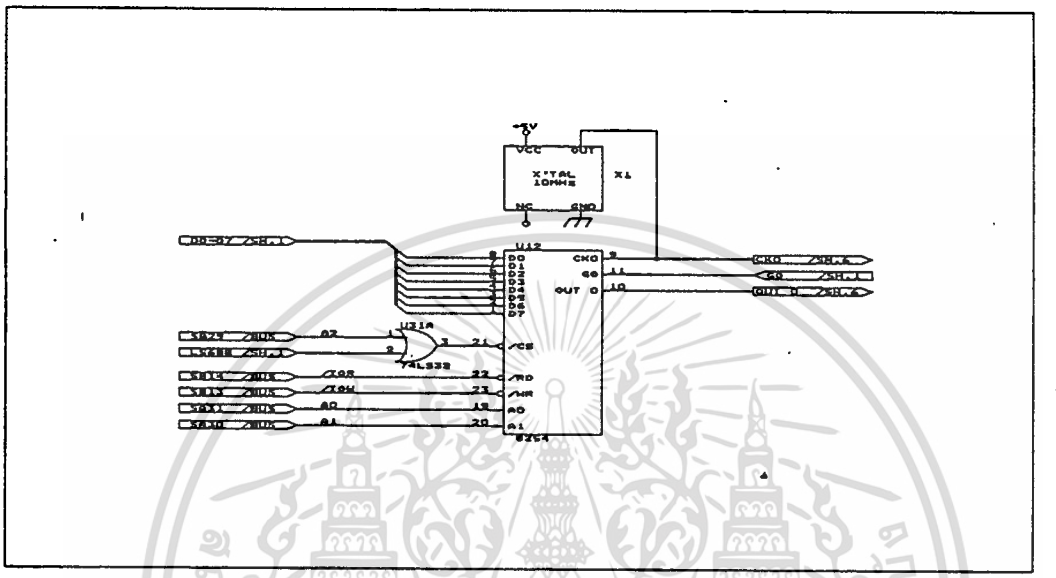
6.3 วงจร Analog to Digital Converter



รูปที่ 31 วงจร Analog to Digital Converter

วงจรนี้จะรับสัญญาณ Analog เข้ามาทาง J1 ผ่าน Wide Band Buffer U7 (LH0033) ซึ่งทำหน้าที่เป็น Buffer ของสัญญาณ Vin ที่จะป้อนให้กับ IC Analog to Digital Converter U10, U11 (CA3306) ซึ่งเป็น ADC แบบ 6 Bit ต่ออนุกรมกันอยู่ ซึ่ง U10, U11 จะได้รับแรงดันอ้างอิงจาก U8 (LH0070) ซึ่งจะให้ Output ที่ขาสอง เป็น +10 โวลต์ และถูก Divider โดย R2 และ R3 ให้เหลือเพียง 5 โวลต์ เพื่อป้อนให้กับขาสองของ U9A และขา 13 ของ U9B เพื่อทำเป็นแรงดันอ้างอิงทั้งบวกและลบให้กับ U10 และ U11 และจากวงจรจะเห็นได้ว่า U10 และ U11 จะทำการ Sampling ตามสัญญาณ SAM-CLOCK ทำให้ได้ AD0-AD7 ออกมาที่ Output

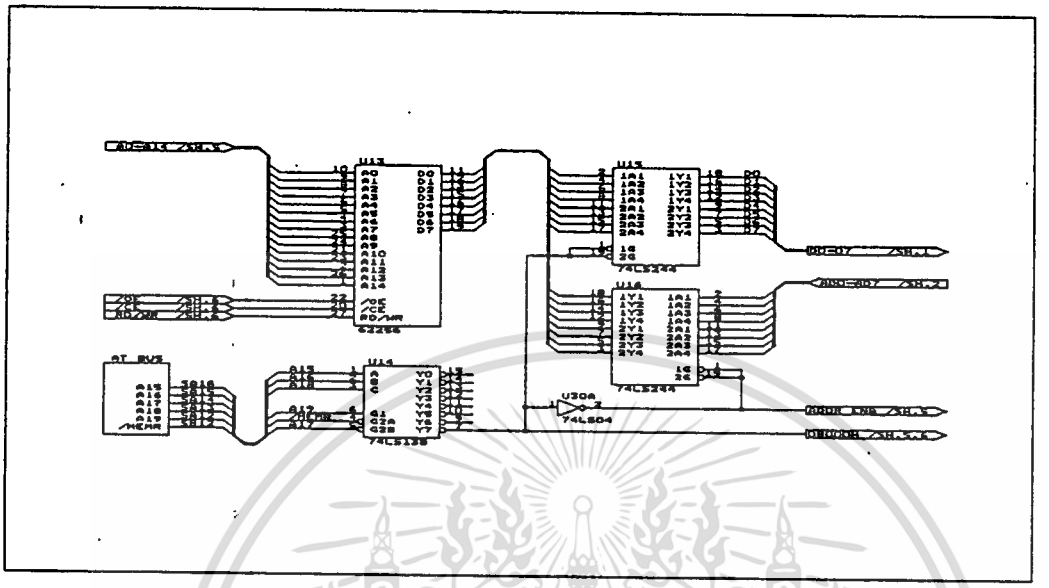
6.4 วงจร Clock Generator



รูปที่ 32 วงจร Clock Generator

วงจรนี้จะใช้ U12 (8254) เป็นตัวหาร Clock ที่ได้จาก Cystal 10 MHz (X1) เพื่อให้ได้ความถี่ของ Clock ตาม Time Base ที่ต้องการ จากวงจรจะเห็นได้ว่าเราใช้ A2 และ 74LS688 มา OR กัน เพื่อทำสัญญาณ /CS ให้แก่ U12 โดยจะสังเกตได้จากตารางที่ 11 จะเห็นได้ว่าขณะที่ A2 และ 74 LS688 เป็น 0 ทั้งคู่ นั่นหมายถึง ได้มีการเลือก Port # 300H-303H ซึ่งเป็น Port ที่สงวนเอาไว้ใช้สำหรับ U12 โดยเฉพาะ ซึ่งในการใช้งาน U12 นี้ จะต้องประกอบไปด้วยสัญญาณ /IOR , /IOW, A0 และ A1 ซึ่งกระบวนการต่าง ๆ นี้ได้กล่าวเอาไว้แล้วในเรื่องของการใช้งาน IC #8254 ข้างต้น

6.5 วงจร RAM และ Memory Decoder



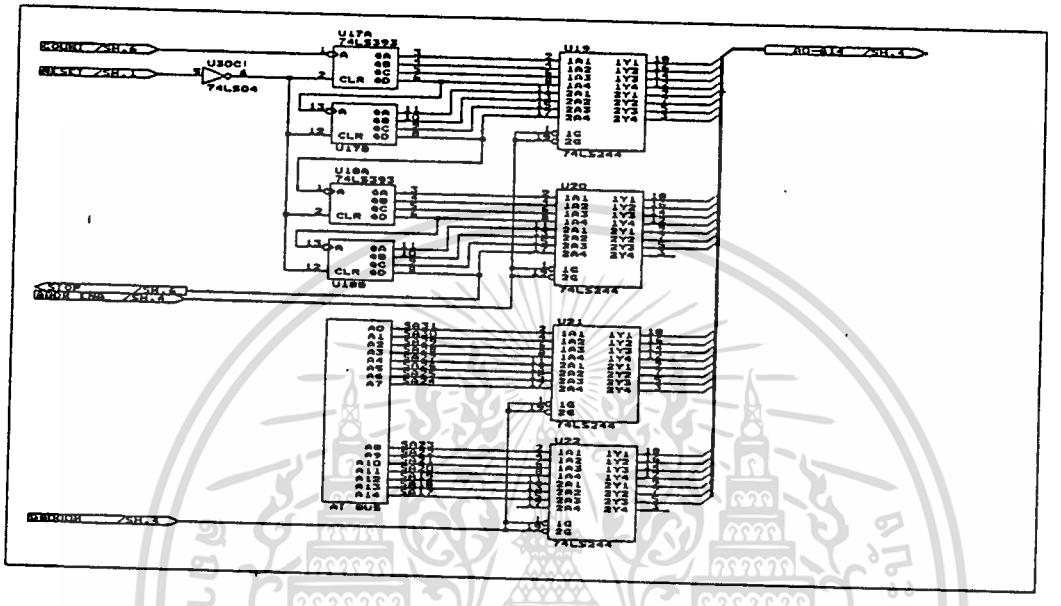
รูปที่ 33 วงจร RAM and Memory Decoder

ในส่วนของ RAM เราได้ใช้ RAM ขนาด 32KB เบอร์ 62256 (U13) ซึ่งได้รับสัญญาณควบคุมต่าง ๆจาก U24 ของวงจร Controller และรับ DATA มาจาก ADC โดยผ่าน Buffer U15 (74LS244) ในส่วนของวงจรนี้เราใช้ U14 (74LS138) ทำหน้าที่เป็นตัวถอดรหัส ซึ่งเราจะถอดรหัสในช่วงส่วนที่ว่างของด้านบนของหน่วยความจำใน IBM/PC หรือ Compatible โดยความจำส่วนนี้จะอยู่เหนือ 640 KB (Coventional Memory) ขึ้นไป โดยเราจะใช้ช่วง D8000H-DFFFFH (32 KB) และเพื่อให้ดูง่ายขึ้นจากวงจรเราจะเขียนเป็นตารางที่ 13 ได้ดังนี้

MEMR	A19	A18	A17	A16	A15	A14	A13	A12	A11-A8	A7-A4	A3-A0
0	1	1	0	1	1	0	0	0	0000	0000	0000
		(D)				(8)			(0)	(0)	(0)
⇓		⇓				⇓			⇓	⇓	⇓
0	1	1	0	1	1	1	1	1	1111	1111	1111
		(D)				(F)			(F)	(F)	(F)

ตารางที่ 13 Memory Decoder

6.6 วงจร Address Counter

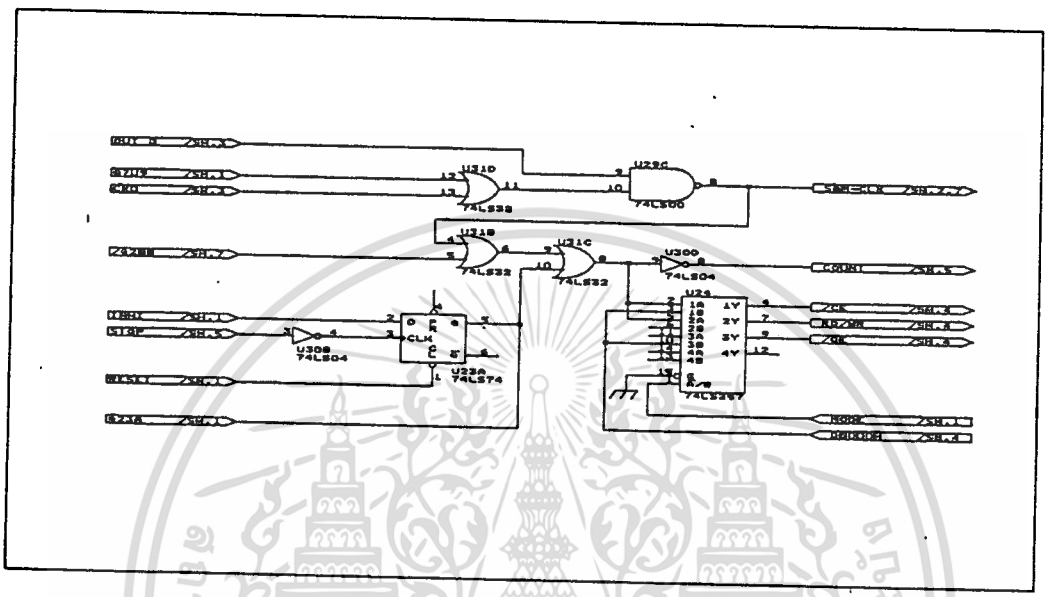


รูปที่ 34 วงจร Address Counter

จากวงจรเราใช้ U17 และ U18 (74LS393) เป็น Counter ทำการนับ Address ให้กับ RAM ตอนเก็บข้อมูลที่ได้จากการ Sampling ของ ADC โดย U17 และ U18 นี้ ได้ Clock ในการนับมาจากวงจร Contoriller ซึ่งก็คือสัญญาณ Count นั่นเอง และเมื่อการ Count เต็ม 32 KB แล้วก็จะส่งสัญญาณ STOP ไปสั่งให้หยุดการ Sampling

ส่วนในการ Tranfer Data ไปที่ Memory ของเครื่อง IBM PC นั้น เราใช้ Address ของ AT BUS มาเป็นตัว Count โดยผ่านทาง Buffer U21 และ U22 (74LS244) โดยจะได้รับการ Enarble จากสัญญาณ D8000H ซึ่งได้มาจากวงจร Memory Decoder

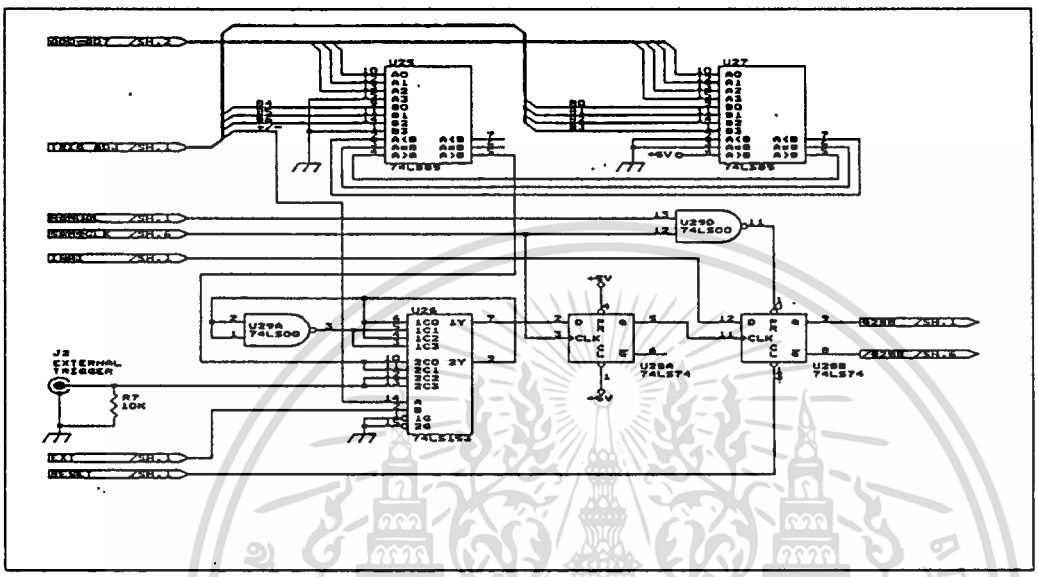
6.7 วงจร Seqence Controller



รูปที่ 35 วงจร Sequence Controller

วงจรส่วนนี้จะทำหน้าที่เป็นตัวควบคุมการ READ การ WRITE ของ RAM และยังเป็นส่วนควบคุมทำให้ Count หรือหยุด Count และเป็นวงจรที่จ่าย Clock ให้แก่ A/D

6.8 วงจร Trigger



รูปที่ 36 วงจร Trigger

วงจรส่วนนี้ทำหน้าที่ในการกำหนดจุด Trig ของสัญญาณว่าจะให้ Trig ที่จุดใด โดยมี 74LS85 เป็นตัว Compartor ระหว่าง Data ของ A/D และ Data ที่ Write ผ่าน Port I/O ในวงจร Port Decoder

บทที่ 7

การทำงานของ PC Oscilloscope

7.1 หลักการทำงานของ Software

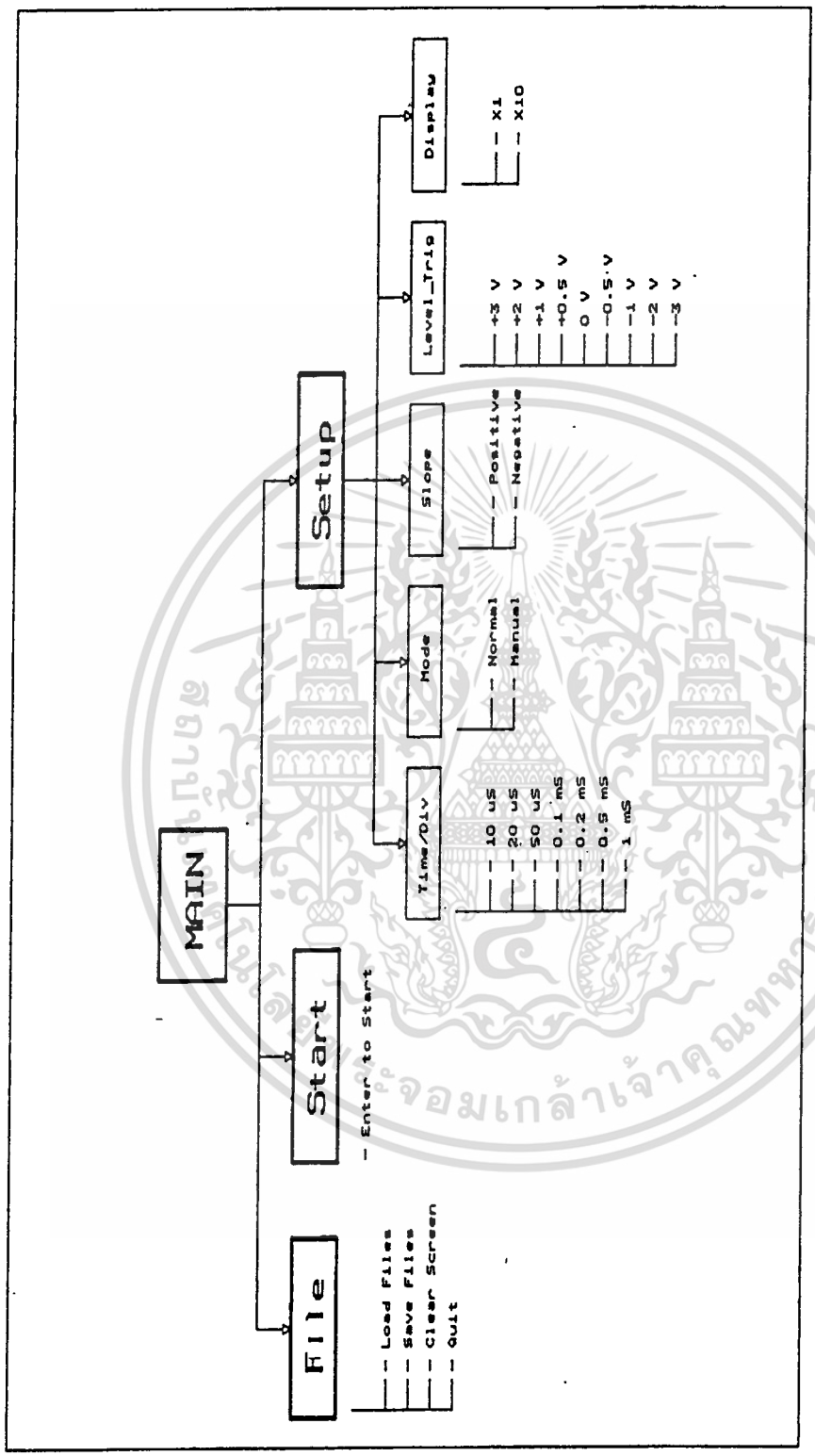
7.1.1 Flowchart ของ Program

จาก Flowchart ของ Program จะแบ่งออกเป็น 3 ส่วนใหญ่ๆดังนี้

1. File : ในส่วนนี้จะทำหน้าที่จัดการเกี่ยวกับแฟ้มข้อมูลคือ
   - Load File ทำหน้าที่อ่านข้อมูลรูปสัญญาณจากแผ่น Disk เพื่อมาแสดงผลที่หน้าจอ
   - Save File ทำหน้าที่เก็บข้อมูลรูปสัญญาณลงบนแผ่น Disk
   - Clear Screen ทำหน้าที่ Clear window ของการแสดงผลรูปสัญญาณ
   - Quit ทำหน้าที่ยกเลิกการทำงานของโปรมแกรม และออกจากโปรมแกรม

2. Start:
   - Enter To Start ทำหน้าที่เริ่มเก็บรูปสัญญาณลงใน RAM

3. Setup: ทำหน้าที่เกี่ยวกับการ Set Up Parameter ของโปรมแกรมดังนี้
   - TIME/DIV ทำหน้าที่เปลี่ยนค่า TIME/DIV (10uS ถึง 1mS)
   - Mode ทำหน้าที่เปลี่ยน Mode การทำงานของโปรมแกรมว่าจะให้เป็น Normal หรือ Manual
   - Slope ทำหน้าที่ Set Slope ของการ Trig ใน Mode Normal ว่าเป็นการ Trig บวกหรือ Trig ลบ
   - Level-Trig ทำหน้าที่ Set Level ในการเริ่มเก็บรูปสัญญาณ
   - X1/X10 ทำหน้าที่ขยาย Scale เมื่อต้องการใช้ Function คูณ 10



Flowchart Program

7.1.2 Flowchart การทำงานของ Hardware

จากรูป Flow Chart Hardware สามารถอธิบายได้ดังนี้

Start : คือการเริ่ม Initail Program และ Initial Hardware

Program 8254 : เป็นการ Program Mode การทำงานของ 8254 ให้อยู่ใน Mode 3 เพื่อใช้ในการหาร Clock จาก Crytal

Set RAM : เป็นการ Set RAM ให้อยู่ใน Write Mode เพื่อเตรียมรับข้อมูลที่ได้จากการ Sampling ของ ADC

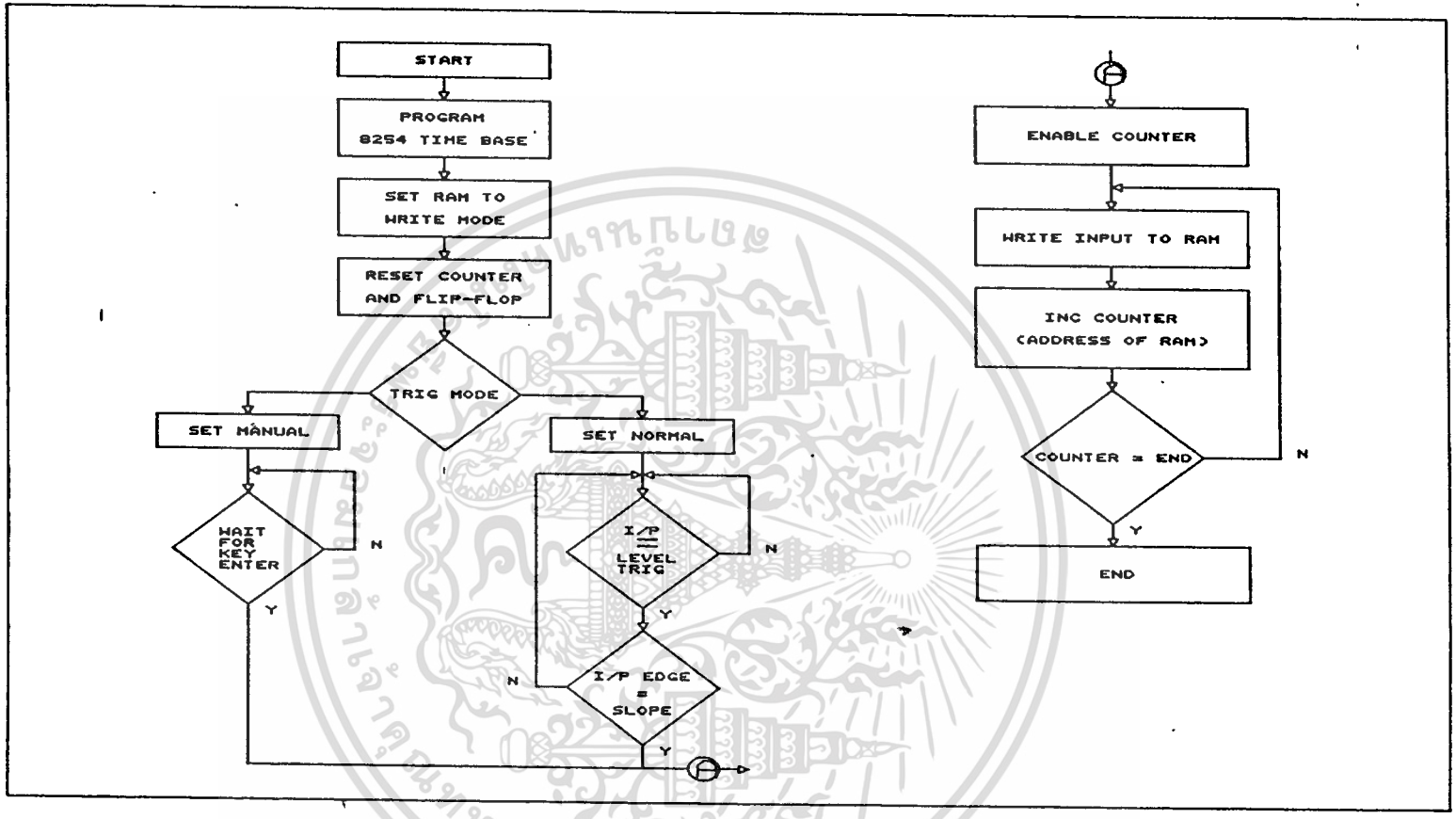
Reset Counter And Flip-flop : เป็นการ Initial Counter และ Flip-Flop ให้อยู่ในสภาวะเริ่มต้นของการทำงาน

Trig Mode : เป็นการ Set ว่าเป็น Mode Normal หรือ Manual

- ถ้าเป็น Mode Manual ก็ให้รอ Key Enter เพื่อเริ่มเก็บข้อมูลลง RAM
- ถ้าเป็น Mode Normal ก็ให้ ตรวจสอบจากการ Sampling เทียบกับค่าที่ Out ออกมาทาง Port ถ้าเท่ากันก็ให้ Check Slope ถ้าตรงกับค่าที่ตั้งเอาไว้ก็ให้เริ่มเก็บข้อมูลลง RAM

Enable Counter : คือเมื่อกด Enter ใน Mode Normal หรือได้ Condition ตามต้องการใน Mode Normal แล้วก็จะมีการ Enable Counter ให้มีการ Count Address ให้กับ RAM

Write Input To RAM และ Increase Counter : Function นี้จะถูกทำไปเรื่อยๆจน RAM เต็ม 32 KB โดยมีการ Check ที่ A14 เมื่อเต็มแล้วก็เป็นการจบการทำงาน



Flowchart การทำงานของ Hardware

7.2 Specification

โครงงานนี้เป็นการออกแบบ Card แปลงสัญญาณอะนาลอกเป็นดิจิตอลพร้อมทั้งซอฟท์แวร์ซึ่งเขียนขึ้นโดยใช้โปรแกรมคอมพิวเตอร์ภาษาซี เพื่อให้สามารถใช้แทนการทำงานของออสซิลโลสโคปได้บางส่วน คุณสมบัติของ Card นี้มีดังต่อไปนี้

1. รับสัญญาณอินพุทสูงสุด 5 Vp-p
2. รับสัญญาณได้ 1 แชนแนล
3. สามารถปรับความถี่ต่อช่วงจากซอฟท์แวร์ คือ 10uS, 20uS, 50uS, .1mS, .2mS, .5mS, 1mS
4. มีหน่วยความจำภายนอกแบบสเตติกแรม 32Kbyte
5. สามารถเก็บข้อมูลลงหน่วยความจำถาวร เช่น แผ่นฟลอปปี้ดิสค์หรือฮาร์ดดิสค์
6. สามรถปรับ Vtrig Hardware ได้ตั้งแต่ +5V ถึง -5V, Software Setup ได้ตั้งแต่ -3V, -2V, -1V, -0.5V, 0V, 0.5V, 1V, 2V, 3 V
7. สามารถแสดงผลได้ทั้งจอ EGA, VGA และ จอโมโนโครม
8. พิมพ์ออกทางเครื่องพิมพ์ได้

7.3 การทดลองและผลการทดลอง

จากการทดลองเราได้ป้อนความถี่ชนิดต่าง ๆ คือ Sine Wave, Square Wave และ Sawtooth ที่ความถี่ต่าง ๆ กัน ดังผลการทดลองต่อไปนี้

รูปที่ 39 สัญญาณ Sine Wave ที่ความถี่ 10 KHz Display X1

รูปที่ 40 สัญญาณ Sine Wave ที่ความถี่ 100 KHz Display X1

รูปที่ 41 สัญญาณ Sawtooth ที่ความถี่ 10 KHz Display X1

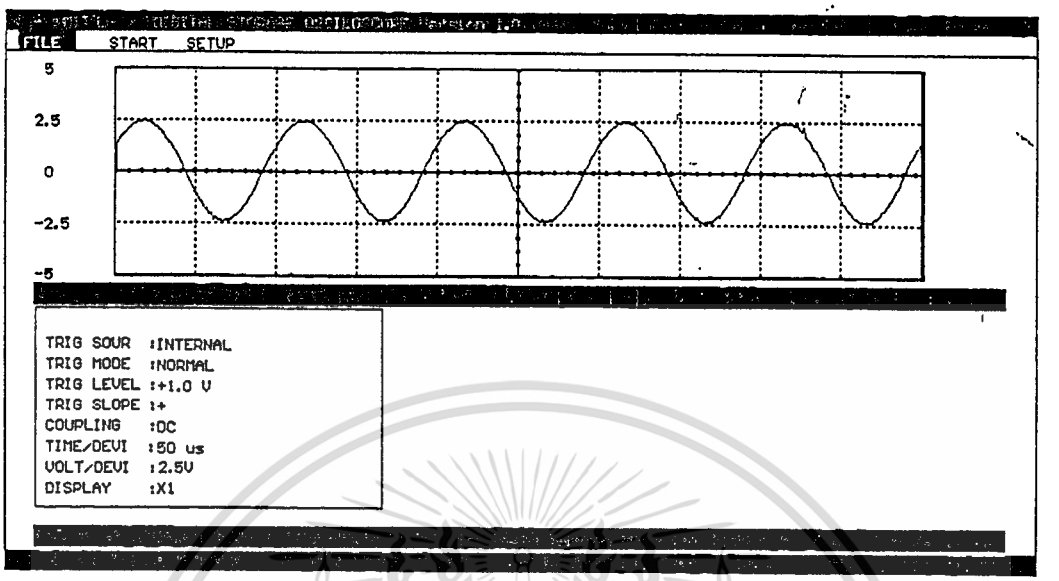
รูปที่ 42 สัญญาณ Sawtooth ที่ความถี่ 50 KHz Display X1

รูปที่ 43 สัญญาณ Square Wave ที่ความถี่ 20 KHz Display X1

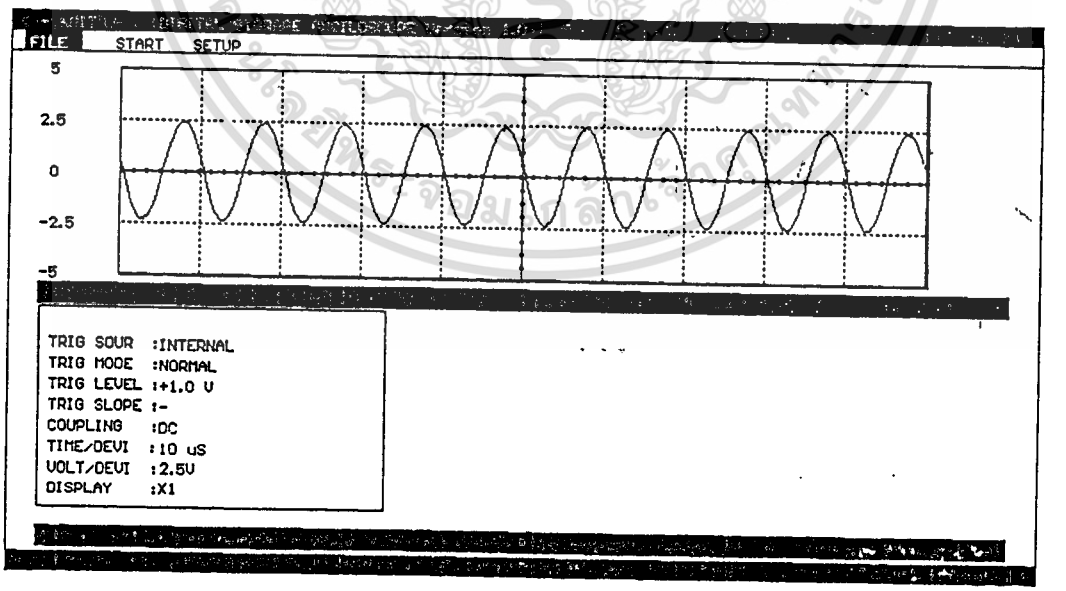
รูปที่ 44 สัญญาณ Square Wave ที่ความถี่ 50 KHz Display X1

รูปที่ 45 สัญญาณ Sine Wave ที่ความถี่ 600 KHz Display X1

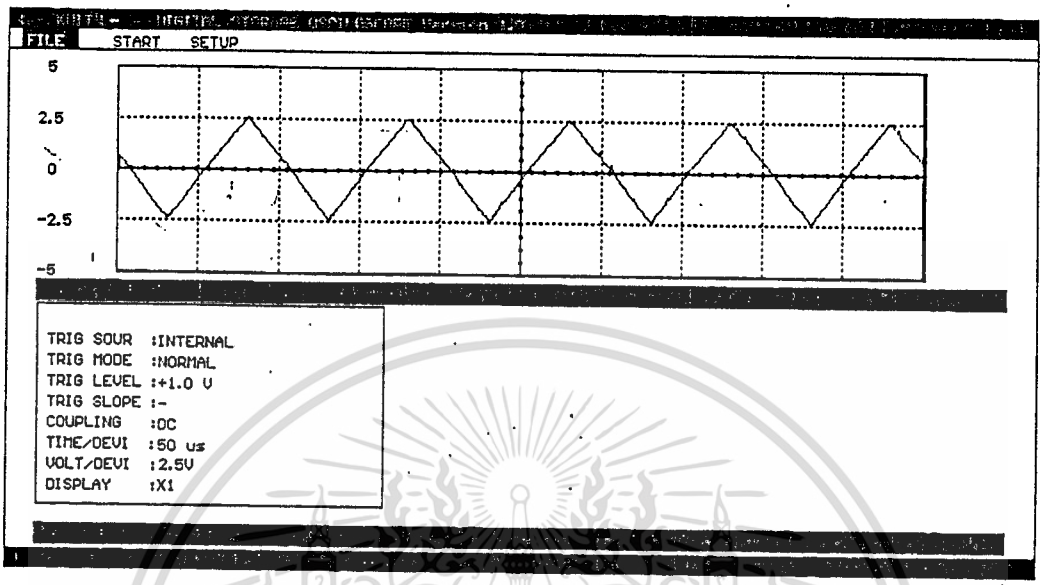
รูปที่ 46 สัญญาณ Sine Wave ที่ความถี่ 600 KHz Display X10



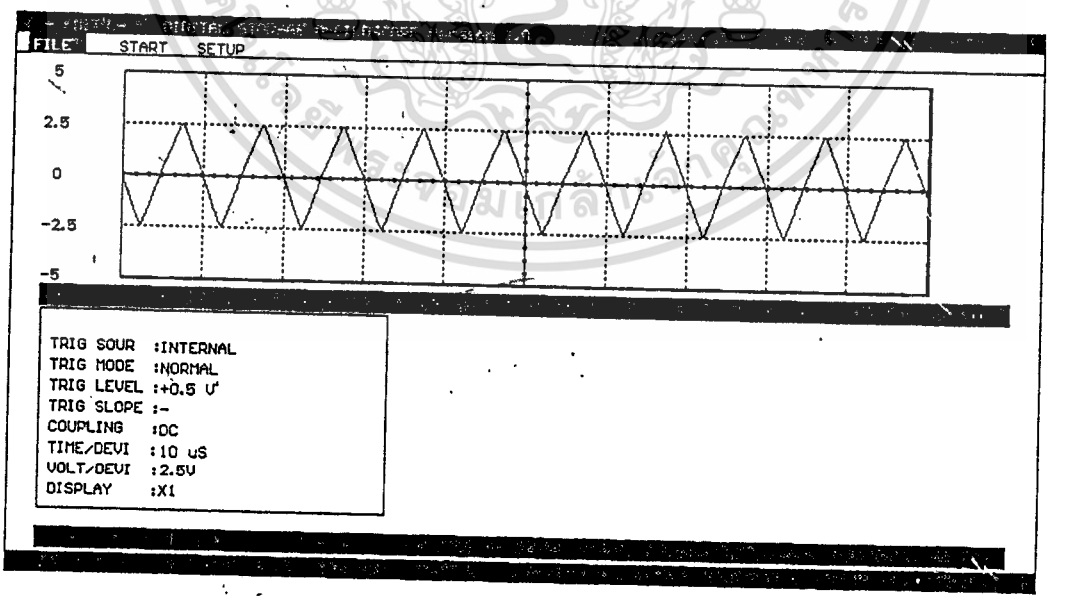
รูปที่ 39 สัญญาณ Sine Wave ที่ความถี่ 10 KHz Display X1



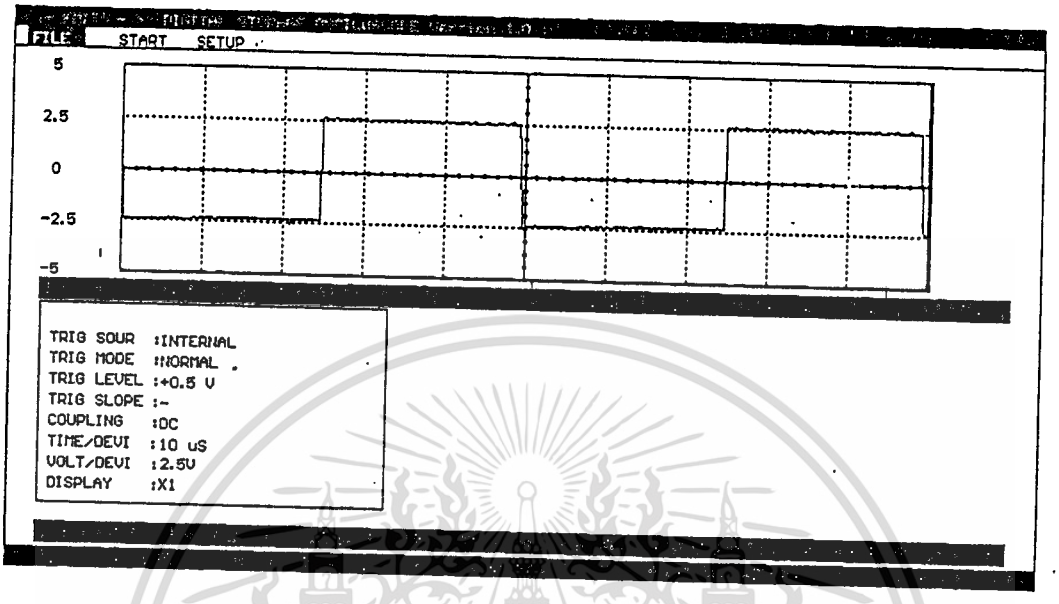
รูปที่ 40 สัญญาณ Sine Wave ที่ความถี่ 100 KHz Display X1



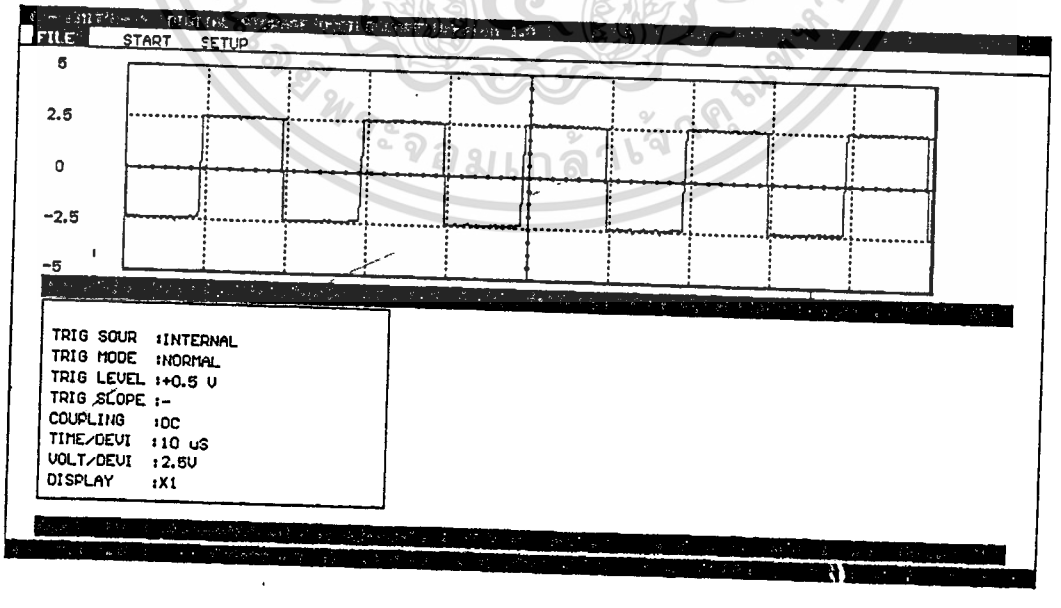
รูปที่ 41 สัญญาณ Sawtooth ที่ความถี่ 10 KHz Display X1



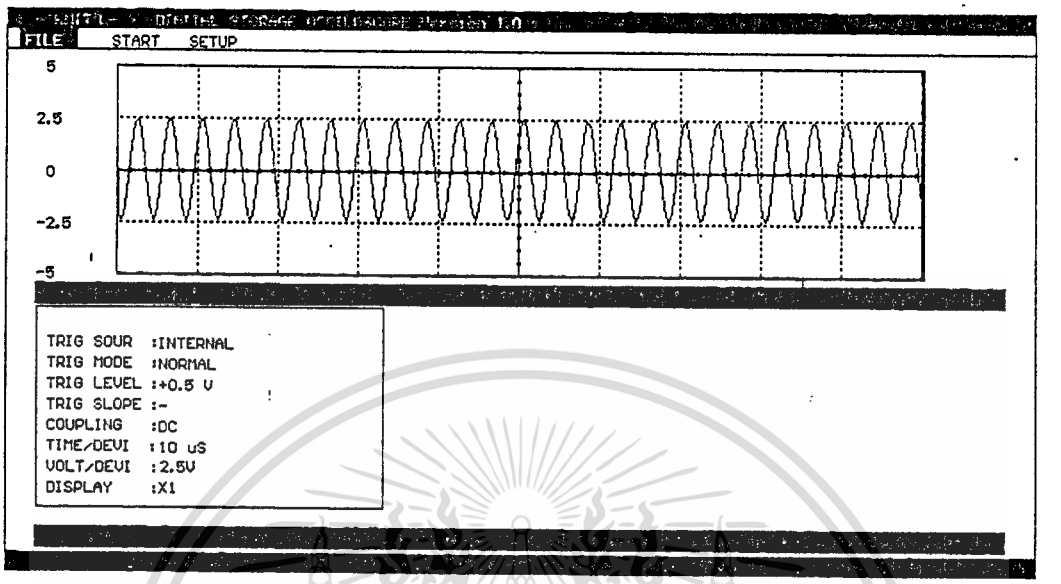
รูปที่ 42 สัญญาณ Sawtooth ที่ความถี่ 50 KHz Display X1



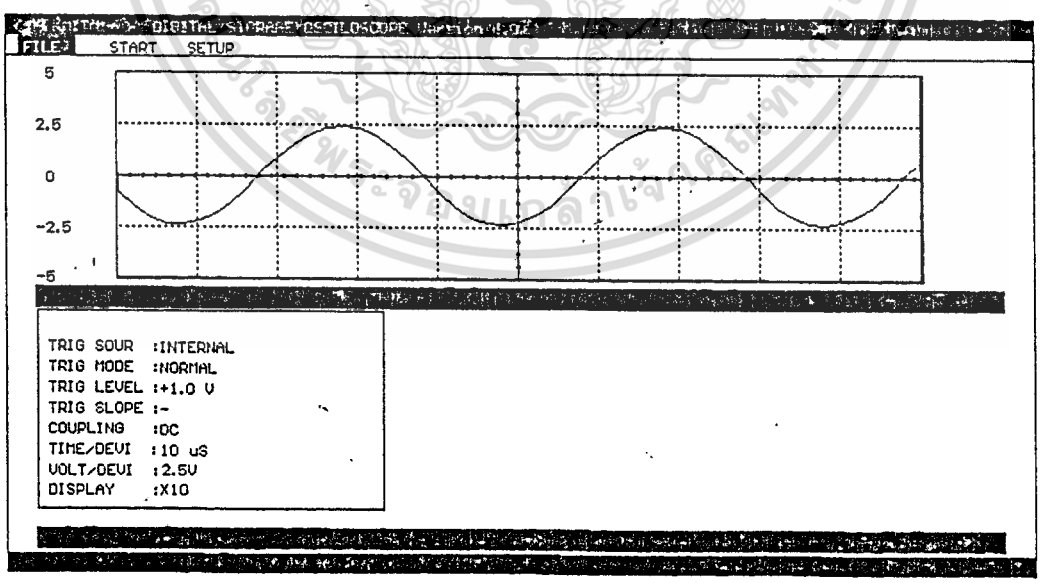
รูปที่ 43 สัญญาณ Square Wave ที่ความถี่ 20 KHz Display X1



รูปที่ 44 สัญญาณ Square Wave ที่ความถี่ 50 KHz Display X1



รูปที่ 45 สัญญาณ Sine Wave ที่ความถี่ 600 KHz Display X1



รูปที่ 46 สัญญาณ Sine Wave ที่ความถี่ 600 KHz Display X10

7.4 ปัญหาที่พบในการทำโครงงาน

1. อุปกรณ์หายาก คือ IC Flash A/D เบอร์ CA3300 ทำให้เสียเวลาในการหาซื้อ

2. การอ่านค่า ADC ในบางครั้งยังผิดพลาดอยู่ เนื่องจากระบบการเดินสายไม่เป็นระเบียบจึงเกิดการรบกวนกัน

3. การออกแบบเราใช้แผ่นวงจรเอนกประสงค์ทำให้เกิดการผิดพลาดได้ง่าย

4. ต้องใช้เวลาเขียนโปรแกรมมากเนื่องจากต้องศึกษาการเขียนโปรมแกรมภาษา C เพราะผู้ทำไม่เคยมีความรู้โปรแกรมภาษา C มาก่อน

7.5 การแก้ปัญหา

ในส่วนของ Hardward ปัญหาส่วนใหญ่เกิดจากการเดินสายที่ไม่เป็นระเบียบเนื่องจากใช้แผ่นวงจรพิมพ์แบบเอนกประสงค์ ทำให้เกิดการรบกวนกันขึ้น จึงแก้ปัญหาดังนี้

1. ต่อตัวเก็บประจุคร่อม IC ทุกตัวเพื่อป้องกันสัญญาณรบกวนจากแหล่งจ่ายไฟ

2. ในเครื่องคอมพิวเตอร์มีการ ออสซิลเลท ของอุปกรณ์บางตัวซึ่งจะมารบกวนสัญญาณอะนาลอกทางอินพุทของเราได้ ดังนั้นที่จุดต่อสัญญาณอินพุทที่เข้ามาวงจรเราได้ใช้สาย SHIELD เพื่อจะลดสัญญาณรบกวนให้น้อยที่สุดเท่าที่จะทำได้

ในส่วนของ Software แก้ปัญหาโดย

1. พยายามค้นคว้าหาตำราหรือเอกสารที่เกี่ยวข้องทางเทคนิคการเขียนโปรมแกรมภาษา C

2. พยายามแก้ปัญหาหรือเทคนิคต่างๆในการเขียนโปรมแกรมจากผู้รู้

บทที่ 8

บทสรุปและวิจารณ์

8.1 สรุปผลการทดลอง

จากการทดลองสามารถวัดสัญญาณรูปคลื่นได้ตั้งแต่ 0 ถึง +/- 5V ความถี่ จนถึง ซึ่งถ้าเกินความถี่ รูปที่ Plot ออกมาจะดูไม่รู้เรื่อง ต้องใช้ Function X10 สัญญาณที่ Plot ออกมาทางหน้าจอคอมพิวเตอร์นั้นเกือบเหมือนกับสัญญาณจากแหล่งจ่ายสัญญาณเลยทีเดียว และยิ่งถ้าเป็นความถี่ต่ำแล้วนั้นรูปที่ได้ออกมาแทบจะไม่มีความผิดเพี้ยนแม้แต่นิดเดียว PC SCOPE นี้สามารถนำไปใช้ได้ค่อนข้างหลากหลายในการวัดสัญญาณไม่ว่าจะเป็น Saw Tooth, Sine Wave หรือ Square Wave หรือสัญญาณอื่นๆที่มีความถี่ไม่สูงนัก PC SCOPE นี้สามารถเก็บข้อมูลไว้ใน Buffer และสามารถสั่งให้ Plot ออกมาทางหน้าจอได้ทันที วงจรที่เราสร้างขึ้นมานี้จะเป็นประโยชน์อย่างมากในการศึกษาถึงการ Interface ของ PC กับอุปกรณ์ภายนอก และยังเป็นการนำเอาทฤษฎีที่ได้เรียนมาประยุกต์ใช้ให้เกิดประโยชน์ขึ้นด้วย

8.2 ประโยชน์ที่ได้จากโครงงาน

1. Hardware ที่สร้างขึ้นสามารถใช้ร่วมกับคอมพิวเตอร์ PC ที่เรามีใช้กันอยู่อย่างแพร่หลาย ทำให้สะดวกต่อการใช้งานเป็นอย่างยิ่ง

2. Card นี้สามารถใช้งานแทน Digital Storage Osilloscope ได้ในระดับหนึ่งในโรงงานอุตสาหกรรมอิเล็กทรอนิคส์

3. สามารถนำไปวิเคราะสัญญาณต่างๆได้ในระดับหนึ่ง

4. ได้รับความรู้เป็นอย่างมากในการทำโครงงานนี้ทั้งในส่วนของ Hardware และ Software

5. สามารถนำเครื่องที่สร้างขึ้นใช้ในการเรียนการสอน

8.3 แนวทางในการพัฒนา

1. อาจเพิ่มชุด Mutiplex เพื่อสะดวกในการวัดสัญญาณหลายๆ จุดโดยไม่ต้องคอยย้าย Probe

2. พัฒนาให้ใช้กับ Mouse หรือ Keybord แบบ Touch Screen เพื่อความสะดวกในการใช้งานมากขึ้นของผู้ปฏิบัติงาน

3. พัฒนา Software ให้มี Display ที่สวยงามน่าใช้งาน

4. พัฒนาในส่วนของ ADC ให้มี Resulution สูงขึ้นและ Sampling Rate สูงขึ้นเพื่อ Accuracy ในการวัดจะได้สูงขึ้น

ภาคผนวก

- Source Program ภาษา C
- Block Diagram and Schematic Diagram
- Data Sheet ของ CA3300, 8254 และ 62256

```
#include <dos.h>
#include <math.h>
#include <conio.h>
#include <stdlib.h>
#include <alloc.h>
#include <stdarg.h>
#include <graphics.h>
#include <fcntl.h>
#include <io.h>
#include <bios.h>

void Initg (void);
void Main_Window (void);
void Main_Win1 (void);
void Read_kbd (void);
void Enter (void);
void Plot_screen(void);
void File_Ser(void);
void Get_Name(void);
void Display(void);
void Setup(void);

int     x=8,y=13,x1,y1,count=0,idx=0,a,acpage=0,vipage=0,res1;
char    menuf=0;
char    count1 []={0,3,0,0,0,3,0,3,0,2,0,1};
int     GraphDriver,GraphMode;
int     maxx,maxy,handle,stept=1,stept1=0,stepf=1,tg_index=2;
char    ascii,scancode,in;
char    image[400],image0[1000],image1[4800],image2[2000];
char    image3[4800];
```

```
char     image4[1000];/*image5[4000],image6[2400]*/
char     image8[3600],imagek[1000];
unsigned size;
void     far *image9,*image10,*image11,*image12,*image7,*image14;
char     buffer[1000];
char     *point[]={image1,image2,image3};
char     fname[]={"TEST.SCP  "};
int       trig=77,offset_time=0,run_mode=0;
int      timediv[]={1,4,20,100,400,1000,4000};
char     *ctimediv[]={"10 uS","20 us","50 us","0.1mS","0.2 mS";
char     "0.5 mS","1 mS"};
char     *fdiv[]={"","250KHZ","125KHZ","62KHZ","","","","31KHZ"};
char     *t_mod[]={"NORMAL","MANUAL"};
char     *t_sour[]={"INTERNAL","EXTERNAL"};
char      *t_slope[]={"-","+"};
char      *t_coup[]={"DC","AC"};
int      cfdiv[]={1,2,4,8,16,};
int       vt[]={102,90,77,70,64,57,51,38,25};
char     *VTG[]={"+3.0 V","+2.0 V","+1.0 V","+0.5 V"," 0.0 V",
                 "-0.5 V","-1.0 V","-2.0 V","-3.0 V"};
char      FLAG;
```

```
#define TIME_PORT          0x300
#define COMMAND_PORT       0x303
#define LEVELTRIG          0x305
#define CONTROL            0x304
```

```
#define NEGATIVE        0x0
#define POSITIVE        0x01
#define DC              0
#define AC              1

#define PORT_STATUS     0x304
#define INTERNAL        0xfd
#define EXTERNAL        0x02
#define NORMAL          0xfb
#define MANUAL          0x04
#define AUTO            0xA2
#define STOP            0x01
#define START           0x02
char DATA_CTR;
int     SLOPE=NEGATIVE,VTRIG=25,SOUR_TRIG=INTERNAL,VOLT_DIV=1;
int     TIME_DEV=0,TRIG_MODE=NORMAL,CONTROL_MODE,COUPLING=DC;
freq_div[]={2,4,10,20,40,100,200};

ex_run()
{
int y,y1,i,trig,data,page;

start_run();
}


set_trig()
{
        if(SLOPE==NEGATIVE)
                outportb(LEVELTRIG,trig|0x80);
```

```
        if(SLOPE==POSITIVE)
                outportb(LEVELTRIG,trig&0x7f);
        if(SOUR_TRIG==INTERNAL)
                CONTROL_MODE&=INTERNAL;
        else
                CONTROL_MODE|=EXTERNAL;

        if(TRIG_MODE==NORMAL)

                CONTROL_MODE&=NORMAL;
        else
                CONTROL_MODE|=MANUAL;
}



start_run()
{
        int status;

        set_trig();
        prg8254(TIME_DEV);

        CONTROL_MODE|=0x49;    /* ON MODE & SELECT 8243 SAMPLING  */
        CONTROL_MODE&=0xcf;    /* SET RAM TO WRITE MODE           */
        if(run_mode==0)
                CONTROL_MODE&=0xfb;          /* SET NORMAL MODE   */
        else
                CONTROL_MODE|=0x04;          /* SET RUN MODE      */
```

```
outportb(CONTROL,CONTROL_MODE);    /* RESET STAT              */
CONTROL_MODE|=0x10;
outportb(CONTROL,CONTROL_MODE);    /* MEASURE STAT            */

for(;;)

{
        status=inportb(PORT_STATUS);

        status&=0x01;
        if(status==0x01) break;
        if(bioskey(1)=1)
            {
            Read_kbd ();
            if (scancode==1)
                {
                FLAG=STOP;
                break;
                }
            if(scancode==77)
                if (offset_time<4000)
                        offset_time++;
            if(scancode==75)
                    if(offset_time=0)
                            offset_time--;
        }

}

}
```

```
prg8254(int n)
{
int devider=0;
int b[2];
  outportb(COMMAND_PORT,0x73);
  devider=freq_div[n];
  b[0]= devider &0x00ff;
  b[1]= devider >8;
  outportb(TIME_PORT,b[0]);
  outportb(TIME_PORT,b[1]);
  }
/*
  volt_step(int v)
  {

   int v_buff1,v_buff2;
   v=v*100;
   v_buff1=5000/128;
   v_buff2=v/v_buff1;
   return(v_buff2);
   }
        */
 /*
   Main program
              */


   void main ()
   {
```

```
prg8254(TIME_DEV);
Initg ();
show_status();
Main_Window ();
closegraph ();
}

void Initg (void)
{
 int a, g_driver,g_mode;

   g_driver=9;
   initgraph (&g_driver,1,""); /* Initialized */
   maxx=getmaxx ();                    /* Get MAX_X   */
   maxy=getmaxy ();                    /* Get MAX_Y   */
   setactivepage(1);
   settextstyle (SMALL_FONT,HORIZ_DIR,4);
   setfillstyle (SOLID_FILL,11);
   bar (0,0,200,10);
   getimage (0,0,150,10,imagek);
   size=imagesize (0,0,131,130);
   image7=malloc(size);

   if (image7==NULL)
        exit (1);
   size=imagesize(0,0,300,200);
   image14=malloc(size);
   if (image14==NULL)
        exit(1);
```

```
setfillstyle (SOLID_FILL,14);
bar (0,0,131,59);
getimage (0,0,131,59,image7);
setfillstyle (SOLID_FILL,7);
bar (0,0,131,11);
getimage (0,0,118,10,image0);
bar (0,0,102,11);
getimage (0,0,89,10,image);
cleardevice ();

setcolor (6);                    /* MENU 1 */
rectangle (0,0,129,57);
rectangle (0,0,130,58);
rectangle (0,0,131,58);
setcolor (6);
outtextxy (0,5,  " LOAD FILES   ");
outtextxy (0,17, " SAVE FILES   ");
outtextxy (0,29, " CLEAR SCREEN");
outtextxy (0,41, " QUIT         ");
putimage (0,0,image7,1);
getimage (0,0,131,58,image1);
cleardevice ();
setcolor (6);                    /* MENU 2 */
rectangle (0,0,129,21);
rectangle (0,0,130,21);
rectangle (0,0,131,21);
setcolor (6);
outtextxy (0,5, " ENTER TO STRAT");
putimage (0,0,image7,1);
getimage (0,0,131,22,image2);
```

```
cleardevice ();
/*************************/

setcolor (6);                /* MENU 3 */
rectangle (0,0,129,57);
rectangle (0,0,130,57);
rectangle (0,0,131,58);
setcolor (6);
outtextxy (0,5,  " TIME/DIV   ");
outtextxy (0,17, " MODE       ");
outtextxy (0,29, " SLOPE      ");
outtextxy (0,41, " LEVEL_TRIG ");
putimage (0,0,image,1);
getimage (0,0,131,58,image);
cleardevice ();

setcolor (6);     /* MENU 5 */
rectangle (0,0,129,45);
rectangle (0,0,130,45);
rectangle (0,0,131,46);
cleardevice ();
setcolor (15);        /* SCOPE BOX MONITOR     .....1...*/
setfillstyle (SOLID_FILL,0);
bar(0,0,500,200);
rectangle (0,0,500,128);
rectangle (0,0,501,129);
rectangle (0,0,502,129);
line (0,32+32,500,32+32);
line (250,0,250,128);
setlinestyle (USERBIT_LINE,0x0303,THICK_WIDTH);
```

```
line (0,32+32,500,32+32);
line (250,0,250,128);

setlinestyle (DOTTED_LINE,0,NORM_WIDTH);
x1=50;

for (a=9;a>0;a--,x1+=50)
    line (x1,0,x1,128);
y1=32;
for (a=3;a>0;a--,y1+=32)
        line(0,y1,500,y1);
size=imagesize (0,0,502,129);
image9=malloc(size);
if (image9==NULL)
       {
       closegraph ();
       exit (1);
       }
getimage (0,0,502,129,image9);
setlinestyle (SOLID_LINE,0,NORM_WIDTH);
cleardevice ();
setcolor (15);              /* SCOPE BOX MONITOR  .....1.... */
setfillstyle (SOLID_FILL,1);
bar (0,0,600,32);
rectangle (0,0,600,12);
rectangle (0,0,601,13);
rectangle (0,0,602,13);
size=imagesize (0,0,602,13);
image10=malloc (size);
if (image10==NULL)
```

```
        {
        closegraph ();
        exit (1);
        }
getimage (0,0,602,13,image10);

setfillstyle (SOLID_FILL,15);
bar (0,0,512,120);
rectangle (0,0,512,120);
rectangle (0,0,513,121);
rectangle (0,0,514,121);
setlinestyle (SOLID_LINE,0,NORM_WIDTH);
cleardevice ();
setfillstyle(SOLID_FILL,15);
bar (0,0,35,11);
getimage (0,0,55,11,image);
putimage (0,0,image,1);

bar (0,0,maxx,10);                    /* GET IMAGE FOR MENU */
getimage (0,0,maxx,10,image8);
cleardevice ();
setcolor(15);
outtextxy(0,0, " TRIG SOUR  :        ");
outtextxy(0,13," TRIG MODE  :        ");
outtextxy(0,26," TRIG LEVEL :        ");
outtextxy(0,39," TRIG SLOPE :        ");
outtextxy(0,52," COUPLING   :        ");
outtextxy(0,65," TIME/DEVI  :        ");
outtextxy(0,78," VOLT/DEVI  :2.5V    ");
size=imagesize (0,0,200,95);
```

```
image12=malloc (size);
if (image12==NULL)
        {
        closegraph ();
        exit (1);
        }

getimage(0,0,200,95,image12);

cleardevice();
setactivepage(0);

}

/*
  DISPLAY MENU ON SCREEN WITH 2 PAGE
  AND MAIN MENU TO ACCRESS FUNCTION  */

void Main_Window (void)
{
  Main_Win1 ();
  show_status();
  setvisualpage(0);
  setactivepage(1);
  Main_Win1();
  show_status();
  setactivepage(0);
  getimage (x,0,x+131,y+100,image7);
  putimage (x,y,image,1);
```

```
x1=x+5;
y1=y+25;

/* set loop display bar menu */

for (;;)
    {
    Read_kbd ();              /* Get Key Board */
    switch (scancode)
        {
        case 1:               /*Esc Key Press */
        if (menuf==1)
            {
            menuf=0;
            putimage (x,0,image,0);
            putimage (x,y,image,1);
            }
        break;

        case 28 :             /* Enter */
        if (menuf==0)
            {
            menuf=1;
            putimage (x,y+20,point[count],0);
            putimage (x1,y1,image0,1);
            }
        else
        Enter ();

        break;
```

```
case 77:          /*  Ringth Key Press */
if (count==2)
break;
putimage (x,0,image7,0);
count++;
x+=52;
getimage (x,0,x+131,y+100,image7);
putimage(x,y,image,1);
idx+=2;
x1=x+5;
y1=y+25+(count1[idx]*12);

if (menuf==1)
        {
        putimage (x,y+20,point[count],0);
        putimage (x1,y1,image0,1);
        }

break;

case  75:          /* Left Key Press */
if (count==0)
break;
putimage (x,0,image7,0);
count--;
x-=52;
getimage (x,0,x+131,y+100,image7);
putimage (x,y,image,1);
idx-=2;
x1=x+5;
```

```
y1=y+25+(count1[idx]*12);
if (menuf==1)
        {
        putimage (x,y+20,point[count],0);
        putimage (x1,y1,image0,1);
        }

break;
case  80:        /* Down Key Press */
if (menuf==0)
    break;
if (count1[idx]==count1[idx+1])
        {
        putimage (x1,y1,image,1);
        y1=y+25;
        count1[idx]=0;
        putimage (x1,y1,image,1);
        }
else
        {
        putimage (x1,y1,image,1);
        y1+=12;
        count1[idx]++;
        putimage (x1,y1,image,1);
        }

break;

case  72:        /* Up Key Press */
```

```
            if (menuf==0)
                    break;

            if (count1[idx]==0)
                    {
                    putimage (x1,y1,image,1);
                    y1=y+25+(count1[idx+1]*12);
                    putimage (x1,y1,image,1);
                    }
            else
                    {
                    putimage (x1,y1,image,1);
                    y1-=12;
                    count1[idx]--;
                    putimage (x1,y1,image,1);
                    }
            break;

            }/* END LOOP CASE */

    } /* END LOOP Key_Borad Recive */

} /* End Function */

/*
  Draw menu on sceen with active page
                                        */
void Main_Win1 (void)
```

```
{
 int xloc,yloc;
 if (GraphDriver==7)
 { setfillstyle (INTERLEAVE_FILL,0);
   bar (0,28,maxx,maxy-15);  }


 else
 { setfillstyle (SOLID_FILL,0);
   bar (0,26,maxx,maxy-13);  }
 setcolor (6);
 outtextxy (0,1," < - KMITL- >
 DIGITAL STORAGE OSCILOSCOPE Version 1.0");
 putimage (0,1,image8,1);
 setcolor (15);
 outtextxy (0,14,"  FILE    START   SETUP     ");
 setcolor (14);
 putimage(0,maxy-11,image8,1);
 setcolor (15);
 rectangle (0,0,maxx,maxy);
 rectangle (0,12,maxx,25);
 rectangle (0,maxy-12,maxx,maxy);


 putimage ((maxx-502)/2,35,image9,0);
 putimage ((maxx-602)/2,35+134,image10,0);
 putimage ((maxx-602)/2,35+154+132,image10,0);
 setcolor (15);
 outtextxy ((maxx-600)/2,30," 5");
 outtextxy ((maxx-600)/2,30+32,"2.5");
```

```
    outtextxy ((maxx-600)/2,30+64," 0");
    outtextxy ((maxx-600)/2,30+96,"-2.5");
    outtextxy ((maxx-600)/2,30+127,"-5");

    outtextxy((maxx-602)/2,35+134+1,"DIGITAL STORAGE OSCILOSCOPE");
 }

show_status()
{
    int v1,v2,xloc,yloc;
    xloc=(maxx-610)/2;
    yloc=30+159;
    setcolor(15);

    rectangle(xloc+5,yloc-5,xloc+220,yloc+120);
    putimage(20,200,image12,0);
    yloc+=12;
    xloc+=75;
    gprintf(&xloc,&yloc," %s",t_sour[0]);
    yloc+=2;
    gprintf(&xloc,&yloc," %s",t_mod[run_mode]);
    v1=VTRIG/10;
    v2=fmod(VTRIG,10);
    yloc+=2;
    gprintf(&xloc,&yloc," %s",VTG[tg_index]);
    yloc+=2;
    if(SLOPE==NEGATIVE)
            gprintf(&xloc,&yloc," -");
    else
            gprintf(&xloc,&yloc," +");
```

```
    yloc+=2;
    gprintf(&xloc,&yloc," %s",t_coup[COUPLING]);
    yloc+=2;
    gprintf(&xloc,&yloc," %s",ctimediv[TIME_DEV]);
    /*
    yloc+=8;
    gprintf(&xloc,&yloc," COUPLING   :");
    setcolor (15);
*/
}
/*
    Read Keyboard function use bios function 16h.
    Inline assembly.
                */

    void Read_kbd (void)
    {
        union REGS reg;
        reg.h.ah=0;
        int86(0x16,&reg,&reg);
        scancode=reg.h.ah;
        ascii=reg.h.al;
    }

/*
    Enter key press entry the other functions
                                        */

    void Enter (void)
```

```
{
/* int  a,b,c;*/
   switch (count)

   {
    case  1:
      putimage (x,0,image7,0);
      FLAG=START;
      for(;;){
      ex_run();
      Plot_screen ();
      if(FLAG==STOP|| run_mode==1)
         break;
      }
      putimage (x,y,image,1);
      putimage (x,y+20,point[count],0);
      putimage (x1,y1,image0,1);
      break;

    case  0:          /* FILE SERVICE */
      File_Ser ();
      break;
    case  2:
      Setup();
      break;

   }
}

/*
```

```
    Function to read data from A/D Convertor
    Display on screen both EGA/VGA &HERCULIS Card.
                                                    */
void Plot_screen (void)
{
    unsigned  char b,ex;
    unsigned  int  a,c,d,e;
    float  max,min;

    setfillstyle (SOLID_FILL,11);
    if (GraphDriver==7)
        setcolor (0);
    else
      setcolor (15);
        acpage=acpage ^1;          /* Set Active Page */
        setactivepage (acpage);
        putimage ((maxx-502)/2,35,image9,0);
        d=(maxx-500)/2;
        moveto (d,35+64);          /* Plot On Screen */
        a=offset_time+1;
        max=-100;
        min=100;
        outportb(CONTROL,CONTROL_MODE|=0x20);
        for (e=1;e<=500;e++,d++,a+=1 /*stept */)
                {
                c=35+128-peekb(0xd800,a);
                lineto (d,c);
                }

        vipage=vipage^1;        /* Set Visual Page */
```

```
        setvisualpage (vipage);
        getimage (x,0,x+131,y+100,image7);
        setactivepage (acpage);
        setcolor (15);
}

/*  Files service routine as save,load,clear,quit
                                                    */
void File_Ser (void)

{
  int a,c,d,e;
  switch (count1[idx])
{

  case 0:
  Get_Name ();          /* Get Key */
  if (scancode==1)     /* Esc Check */
    break;

  handle=_open(fname,O_RDWR);      /* Load File */
  _read(handle,buffer,560);
  _close(handle);
  putimage (x,0,image7,0);       /* Restore Screen */
  if (GraphDriver==7)            /*Detect COLOR */
    setcolor (0);
  else
    setcolor(15);

  d=(maxx-500)/2;
```

```
  moveto (d,35+64);                /* Plot On Screen */
  a=0;
  for (e=1;e<500;e++,d++)
  {
   c=35+128-buffer[e];
   lineto (d,c);      }

  getimage (x,0,x+131,y+100,image7);    /* Save Screen Image */
  putimage (x,y,image,1);
  putimage (x,y+20,point[count],0);
  putimage (x1,y1,image0,1);
  break;

case 1:
  Get_Name ();              /* Get Key          */
  if (scancode==1)          /* Esc Key Check */
    break;

  for(e=0;e<1000;e++)
   {buffer[e]=peekb(0xd800,e);
   }
  handle=_creat(fname,FA_ARCH);      /* Save File */
  _write(handle,buffer,60);

  _close(handle);
  break;

case 2:

  putimage (x,0,image7,0);
```

```
    putimage ((maxx-502)/2,35,image,0);
    getimage (x,0,x+131,y+100,image);
    putimage (x,y,image,1);
    putimage (x,y+20,point[count],0);
    putimage (x1,y1,image,1);
    break;

  case 3:

    closegraph ();
    printf("KMIT L.\n\n");
    exit(0);
  }
}
/*
    Get input from Keyboard press & display in graphics mode
                                                        */
    void Get_Name (void)
  {
    int num,x,y,keyf;

    setviewport (100,50,400,200,1);      /* Adj View Port */
    getimage (0,0,299,100,image14);
    setcolor(0);
    setfillstyle (SOLID_FILL,11);        /* Fill Menu     */
    bar (0,0,290,30);
    outtextxy (0,11, " Enter name :");
    rectangle (0,0,290,30);
    rectangle (0,0,291,31);
    rectangle (0,0,292,31);
```

```
setcolor (0);
num=0;
x=13*6;
y=11;
keyf=0;

setcolor(0);
outtextxy (13*6,11,"_");

for (;;)                    /* Loop Key Check */
{
Read_kbd();

if (scancode==1)          /* Esc Key Press */
break;

if (scancode==28)         /* Enter Key Press */
break;

if (ascii==0)             /* Allow Key Press */
 continue;

if (scancode==14)         /* Back Space Key Press */
{ if (num!=0)
  { putimage (x,y,imagek,0);
  num--;
  fname[num]=0x5f;
  fname[num+1]=0;
  outtextxy (13*6,11,fname);
```

```
  continue; }}

else                    /* Save Data To Buffer */
{ if (num==20)
  continue;
  fname[num]=ascii;
  num++;
  fname[num]=0x5f;
  fname[num+1]=0;
  putimage (x,y,imagek,0);
  outtextxy (13*6,11,fname);
  keyf=1;  }
}

if (keyf=0)       /* No Key Press */
  fname[num]=0;

putimage (0,0,image14,0);
setviewport (0,0,maxx,maxy,1);
}

/*
  DISPLAY BUFFER ON SCREEN FUNCTION.
                                        */
void Display (void)

{
 int a,c,d,e;
 switch (count1[idx])
 {
```

```
case 1:             /* Clear Buffer */

    for (e+0;e<=500;e++)
       buffer[e]=0;
    putimage (x,0,image7,0);
    putimage ((maxx-502)/2,35,image9,0);
    getimage (x,0,x+131,y+100,image7);
    putimage (x,y,image,1);
    putimage (x,y+20,point[count],0);
    putimage (x1,y1,image0,1);
    break;

case 0:                 /* Display Buffer */

    if (GraphDriver==7)     /* Detect Color */
      setcolor (0);
    else
      setcolor (1);
    putimage (x,0,image7,0);
    putimage ((maxx-502)/2,35,image9,0);

    d=(maxx-500)/2;
    moveto (d,35+64);           /* Plot On Screen */
    a=0;
    for (e=1;e<=500;e++,d++,a++)
    {
       buffer[a]=64;
     c=35+128-buffer[a];
     lineto (d,c);
       }
```

```
            getimage (x,0,x+131,y+100,image7); /* Save Screen Image*/
            putimage(x,y,image,1);
            putimage (x,y+20,point[count],0);
            putimage (x1,y1,image0,1);
            break;
        }
    }

    /* Setup function to setup system
*/
void Setup (void)

{
    static int x=5,y=28;
    static int a=0;
    switch(count1[idx])
            {
        case 1:
                run_mode=run_mode^1;
                break;
            case 2:

            if(SLOPE==NEGATIVE)
                SLOPE=POSITIVE;
            else
                SLOPE=NEGATIVE;
            break;

            case 3:
```

```
setviewport (250,50,371,300,1);
getimage (0,0,121,150,image14);
setcolor(0);
setfillstyle (SOLID_FILL,11);
bar (0,0,100,150);

rectangle (0,0,100,150);
rectangle (0,0,101,169);
rectangle (0,0,102,169);
rectangle (0,0,102,22);

outtextxy (0,10-2,"  TRIG LEVEL");
outtextxy (0,30-2,"  +3.0 V ");
outtextxy (0,42-2,"  +2.0 V ");
outtextxy (0,54-2,"  +1.0 V ");
outtextxy (0,66-2,"  +0.5 V ");
outtextxy (0,78-2,"   0.0 V ");
outtextxy (0,90-2,"  -0.5 V ");
outtextxy (0,102-2,"  -1.0 V");
outtextxy (0,114-2,"  -2.0 V ");
outtextxy (0,126-2,"  -3.0 V");
if (a>8);
        {
        x=5;
        y=28;
        a=0;
        }
```

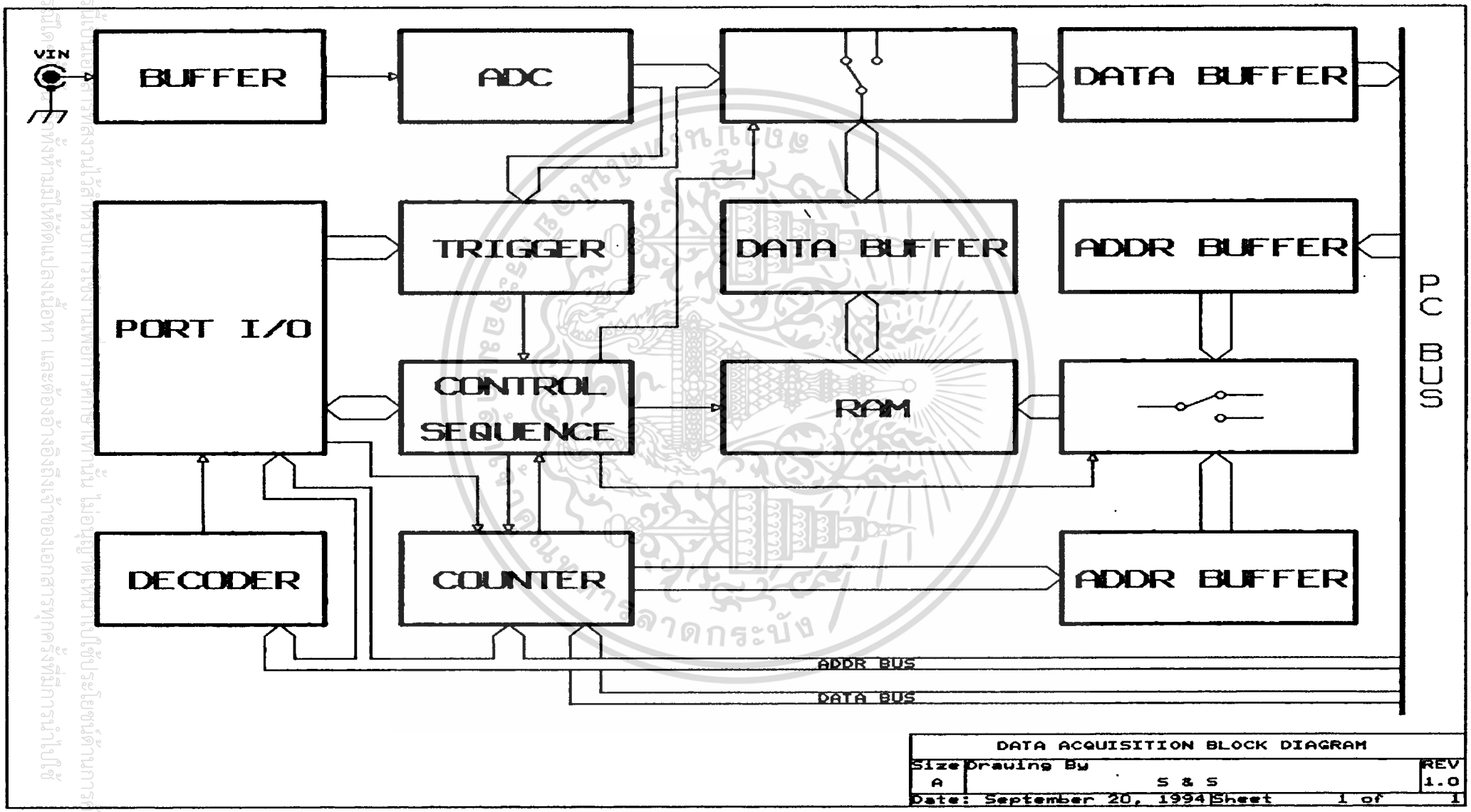
```
putimage(x,y,image4,1);


for (;;)
        {
        Read_kbd ();
        switch (scancode)
                {
                case 72:    /* Up Key Press */
                if(a=0)
                        {
                        a--;
                        putimage (x,y,image4,1);
                        y-=12;
                        putimage (x,y,image4,1);
                        }
                break;

                case 80:      /* Down Key Press */
                if (a!=8)
                        {
                        a++;
                        putimage (x,y,image4,1);
                        y+=12;
                        putimage (x,y,image4,1);
                        }
                break;


                case 28:
                trig=vt[a];
                tg_index=a;
```

```
                break;

                case 1:
                break;

                }

        if (scancode==1)
                break;
        if (scancode==28)
                break;
        }

putimage (0,0,image14,0);
setviewport (0,0,maxx,maxy,1);

break;

case 0:
setviewport (250,50,371,200,1);
getimage (0,0,121,150,image14);
setcolor(0);
setfillstyle (SOLID_FILL,11);
bar (0,0,100,120);
rectangle (0,0,100,120);
rectangle (0,0,101,119);
rectangle (0,0,101,119);
rectangle (0,0,101,22);
outtextxy (0,10-2," TIME/DIV ");
```

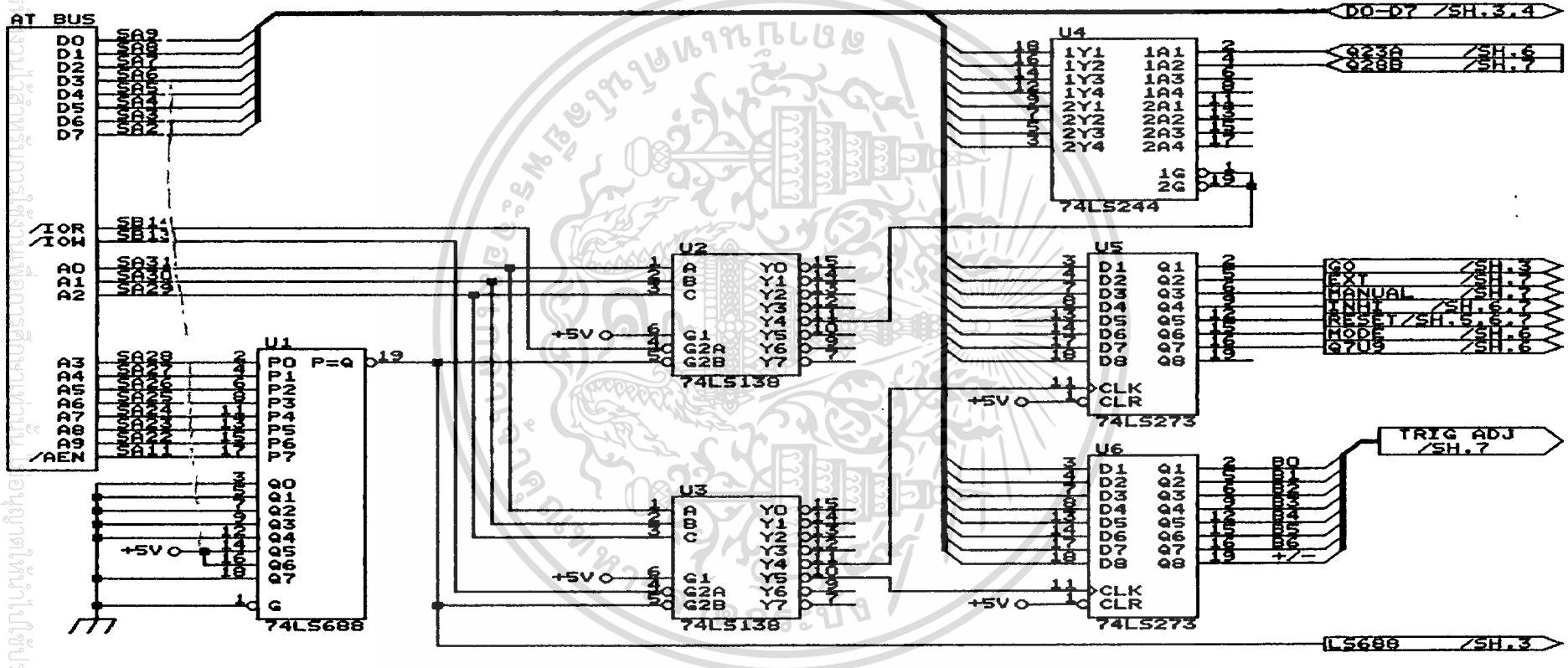
```
outtextxy (0,30-2," 10-us ");
outtextxy (0,42-2," 20-us ");
outtextxy (0,54-2," 50-us ");
outtextxy (0,66-2," .1-ms ");
outtextxy (0,78-2," .2-ms ");
outtextxy (0,90-2," .5-ms ");
outtextxy (0,102-2," 1-ms");
if (a>6);
        {
        x=5;
        y=28;
        a=0;
        }

putimage(x,y,image4,1);

for (;;)
        {
        Read_kbd ();
        switch (scancode)
                {
                case 72:    /* Up Key Press */
                if(a!=0)
                        {
                        a--;
                        putimage (x,y,image4,1);
                        y-=12;
                        putimage (x,y,image4,1);
                        }
                break;
```

```
                case 80:      /* Down Key Press */
                if (a!6)
                        {
                        a++;
                        putimage (x,y,image4,1);
                        y+=12;
                        putimage (x,y,image4,1);
                        }
                break;

                case 28:
                TIME_DEV=a;
                break;

                case 1:
                break;

                }

            if (scancode==1)
                break;
            if (scancode==28)
                break;
            }
        putimage (0,0,image14,0);
        setviewport (0,0,maxx,maxy,1);
```

```
            }

        acpage=acpage^1;
        show_status();
        acpage=acpage^1;
        show_status();
}


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

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

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

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

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

}
```

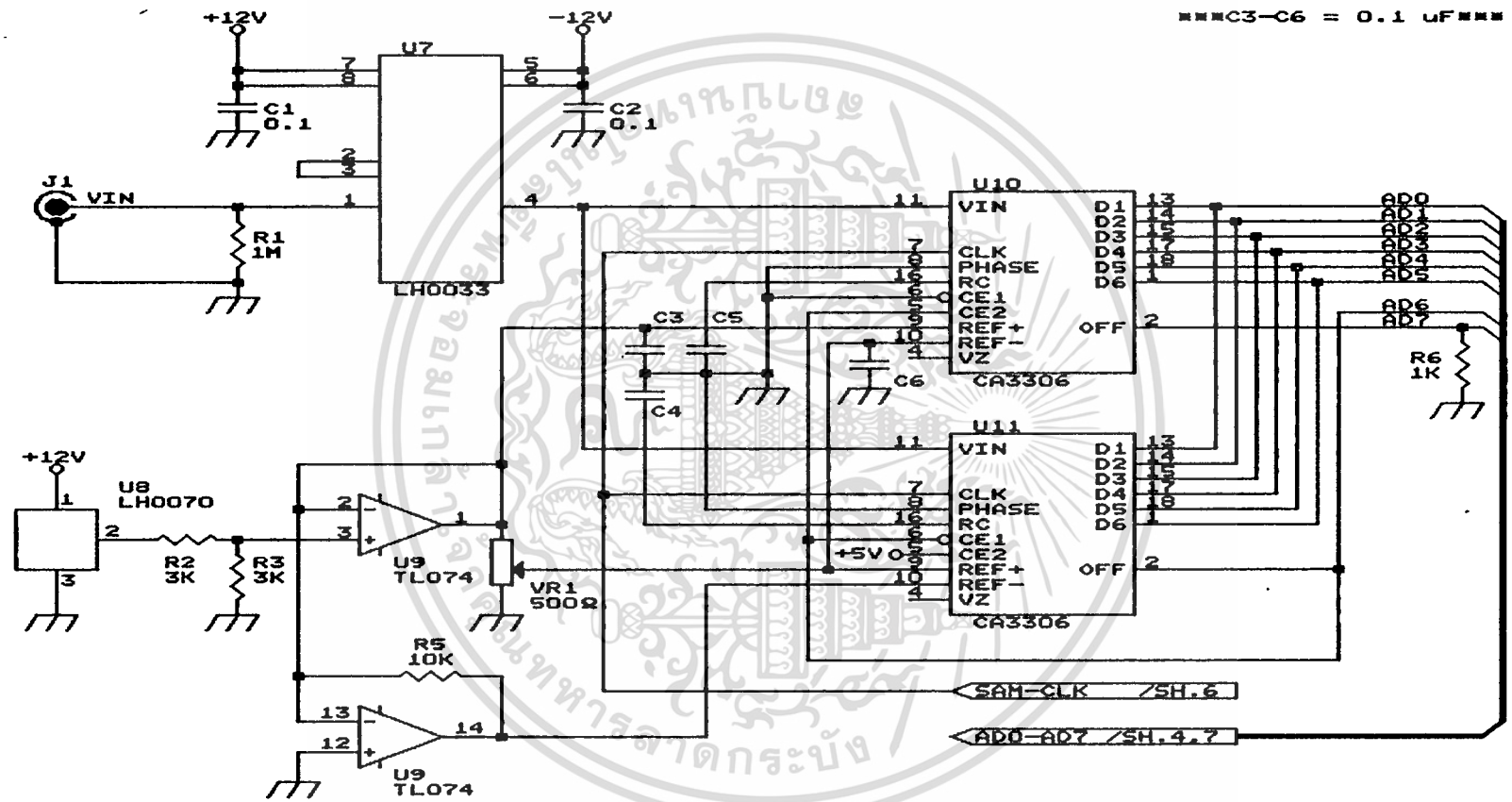


DATA ACQUISITION BLOCK DIAGRAM		
Size A	Drawing By S & S	REV 1.0
Date: September 20, 1994	Sheet 1 of 1	



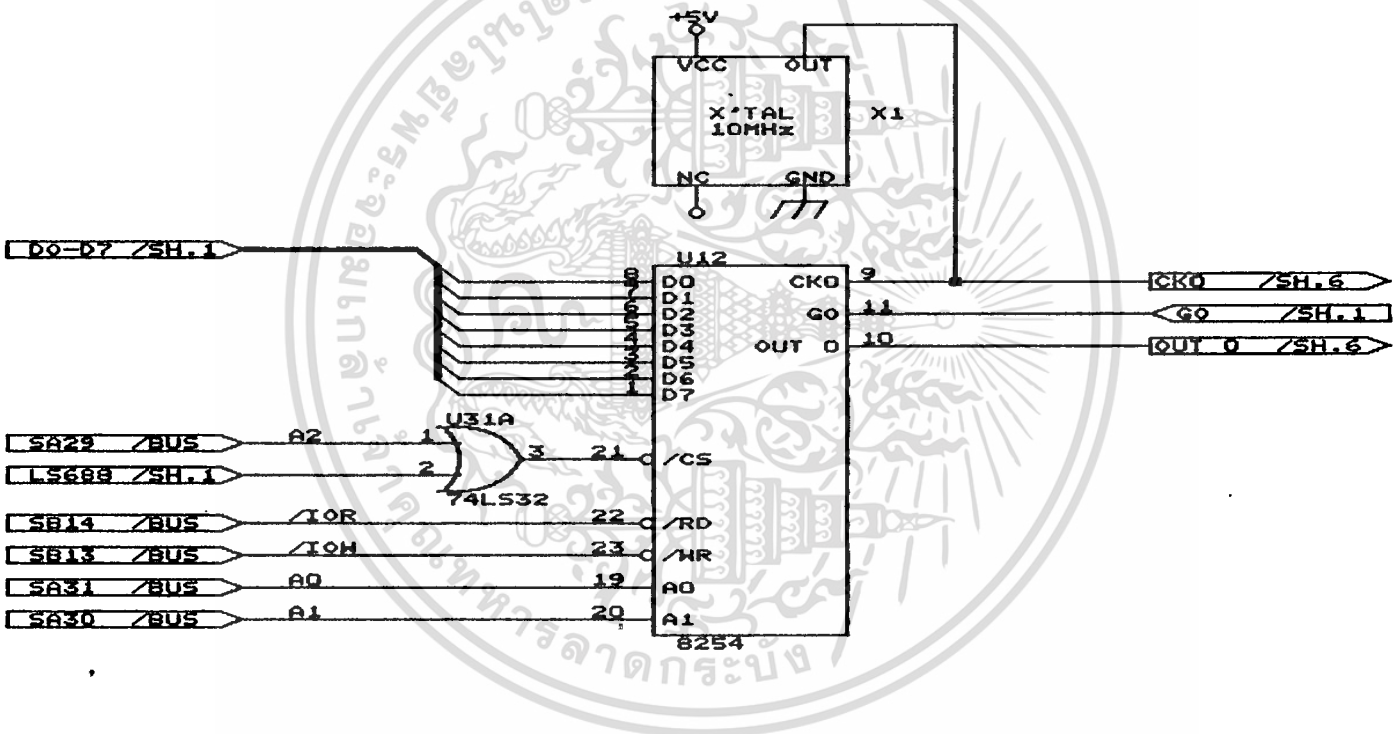
PORT DECODER CIRCUIT

Size	Drawing By	REV
A	S & S	1.0
Date: October 16, 1994	Sheet 1 of 7	

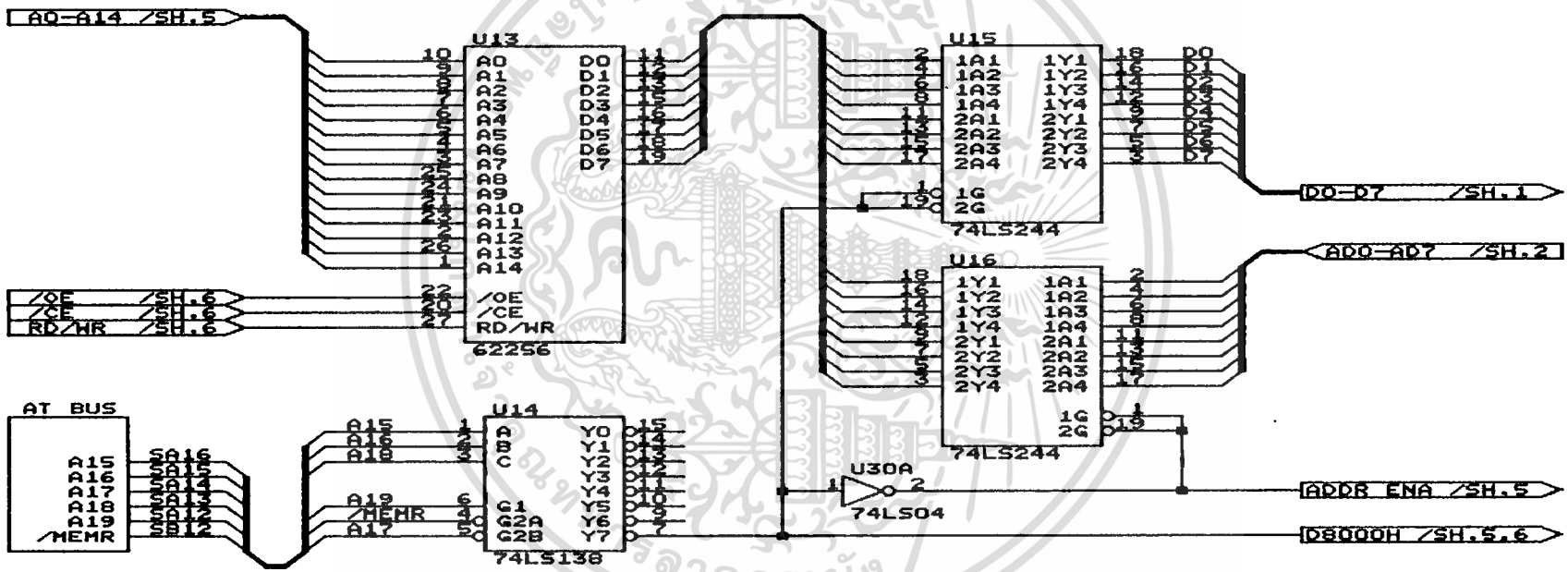


ANALOG TO DIGITAL CIRCUIT

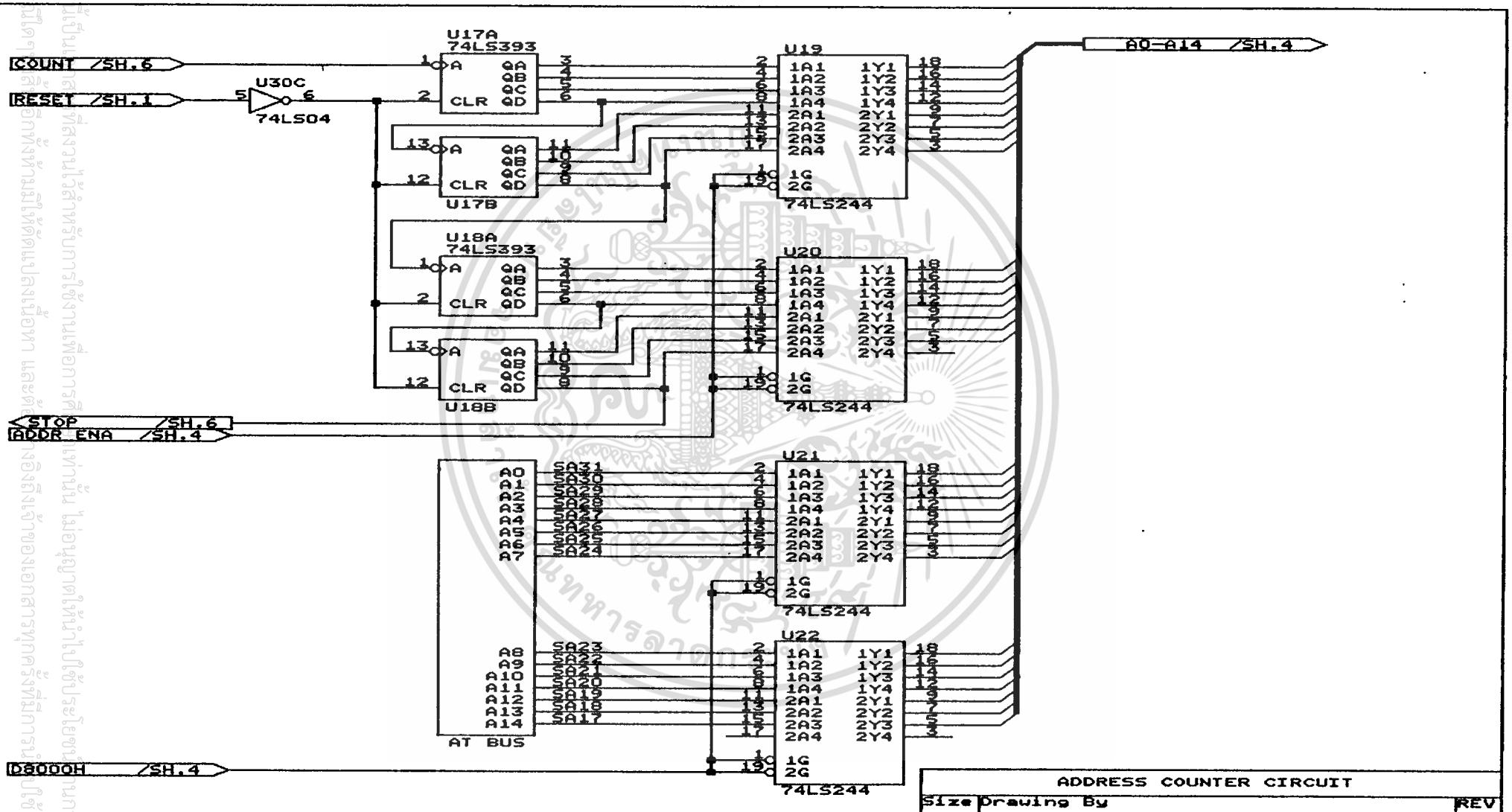
Size	Drawing By	REV
A	S & S	1.0
Date: October 16, 1994	Sheet 2 of 7	

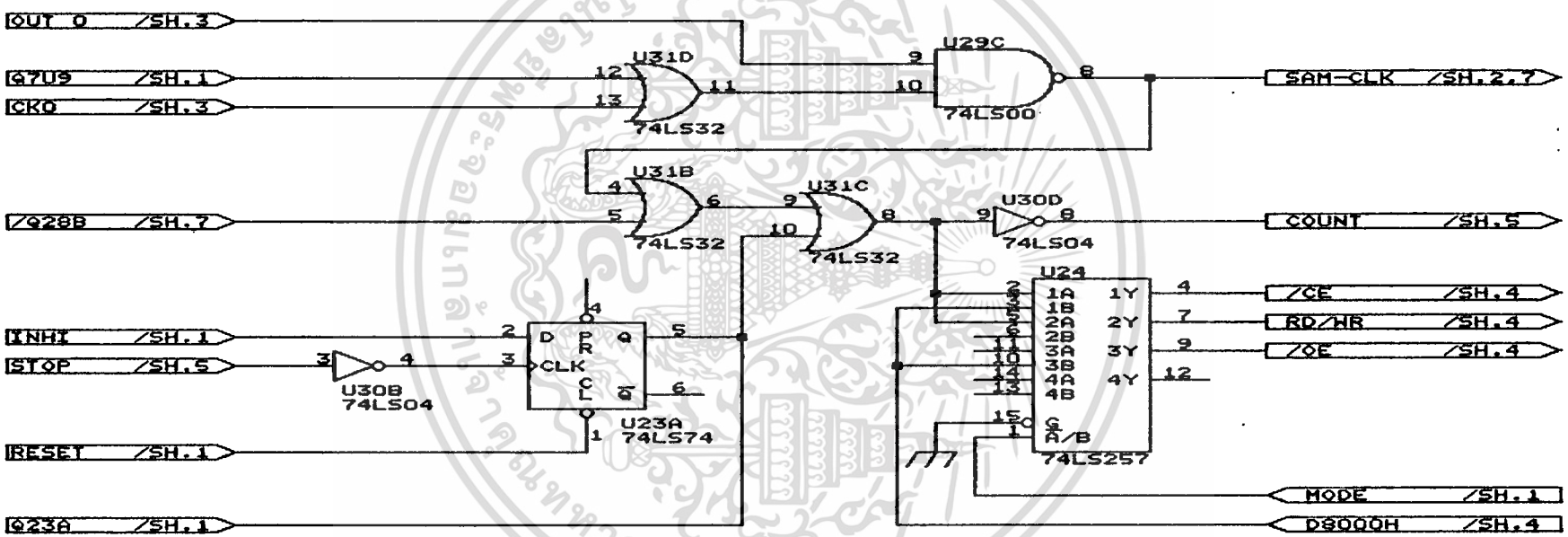


CLOCK CIRCUIT		
Size A	Drawing By S & S	REV 1.0
Date: September 21, 1994	Sheet 3 of 7	

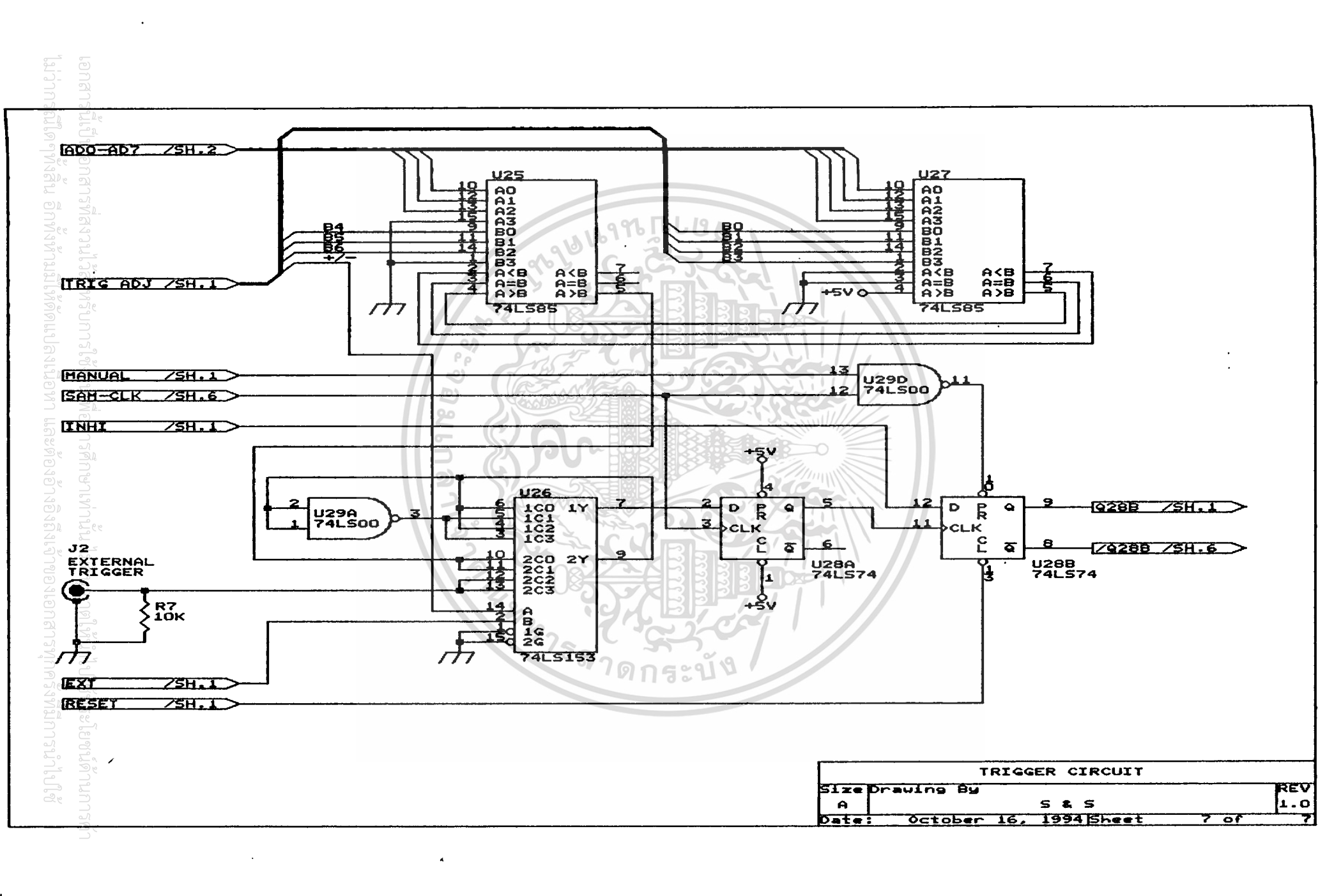


RAM AND MEMORY DECODER CIRCUIT		
Size A	Drawing By S & S	REV 1.0
Date: September 21, 1994	Sheet 4 of 7	


COUNT /SH.6
RESET /SH.1
U30C
74LS04
U17A
74LS393
U17B
U18A
74LS393
U18B
U19
74LS244
U20
74LS244
U21
74LS244
U22
74LS244
A0-A14 /SH.4
STOP /SH.6
ADDR ENA /SH.4
AT BUS
D8000H /SH.4
ADDRESS COUNTER CIRCUIT
Size A
Drawing By S & S
REV 1.0
Date: October 16, 1994
Sheet 5 of 7



CONTROLLER CIRCUIT		
Size A	Drawing By S & S	REV 1.0
Date: October 16, 1994	Sheet 6 of 7	


AD0-AD7 /SH.2
TRIG ADJ /SH.1
MANUAL /SH.1
SAM-CLK /SH.6
INHI /SH.1
EXT /SH.1
RESET /SH.1
B4
B5
B6
+/-
U25
74LS85
U27
74LS85
+5V
U29D
74LS00
U29A
74LS00
U26
74LS153
1C0
1C1
1C2
1C3
1Y
2C0
2C1
2C2
2C3
2Y
A
B
1G
2G
U28A
74LS74
U28B
74LS74
Q28B /SH.1
/Q28B /SH.6
J2
EXTERNAL
TRIGGER
R7
10K
TRIGGER CIRCUIT
Size
A
Drawing By
S & S
REV
1.0
Date: October 16, 1994
Sheet 7 of 7

Product Preview

CA3306, CA3306A, CA3306C

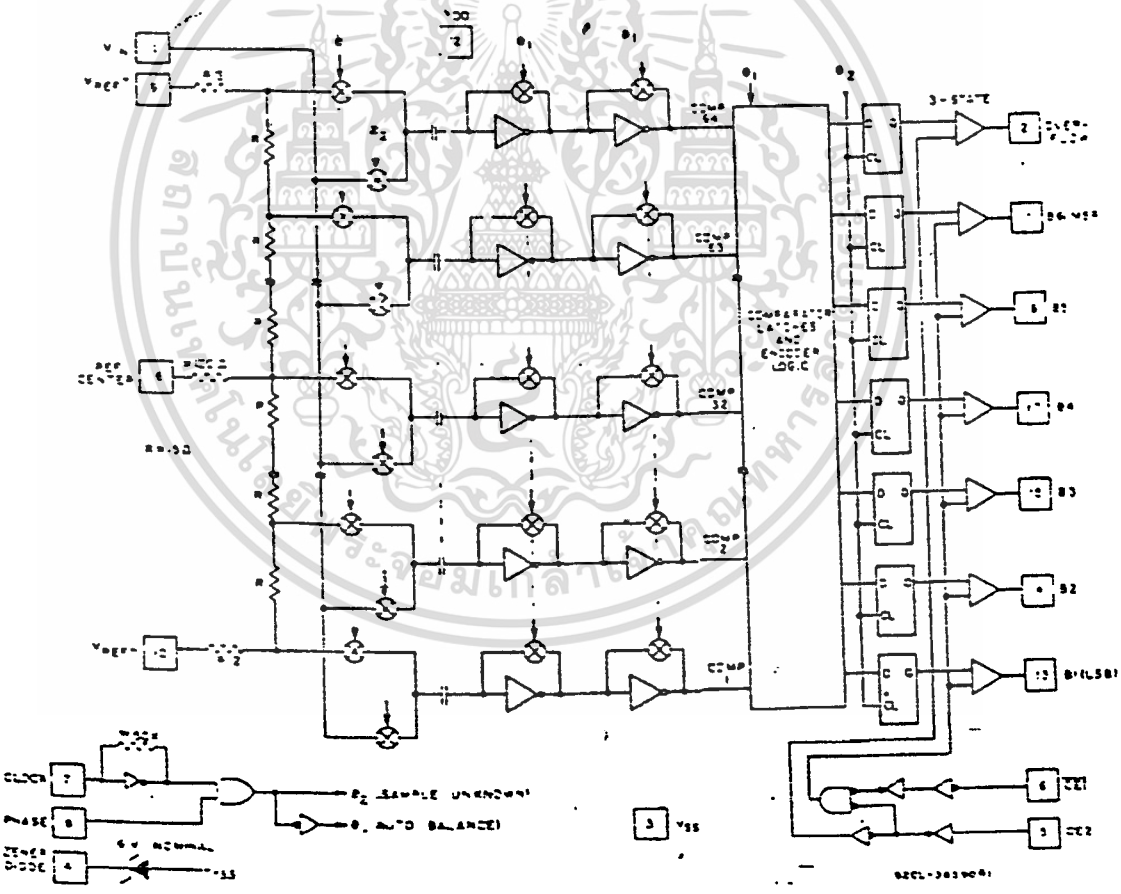
CMOS High-Speed 6-Bit Flash A/D Converter

Features:

- *Improved pin-for-pin retrofits for CA3300*
- *CMOS SOS low power*
- *Flash (Parallel) conversion technique*
- *15 MSPS conversion rate at 5 V*
- *1/4 LSB accuracy*
- *Single 3 to 6 V supply*
- *6 latched-bit outputs plus overflow*
- *May be stacked for higher resolution*
- *May be paralleled for double speed*

The CA3306 family members are pin-for-pin retrofits for the CA3300 (File 1316), but offering improved speed and linearity. All functions of the CA3300 are carried over: the ability to stack devices for higher resolution, parallel devices for doubled speed, and the availability of a built-in zener reference. Accurate digitizing at video speeds is now possible with only a single 5 volt supply (8 volts required for CA3300), and a tighter linearity is guaranteed at a lower reference (full scale) range.

The CA3306-series devices are supplied in 18-lead dual-in-line plastic packages (E suffix) and in 18-lead dual-in-line ceramic packages (D suffix).



Block Diagram of the CA3306

Preview Data only

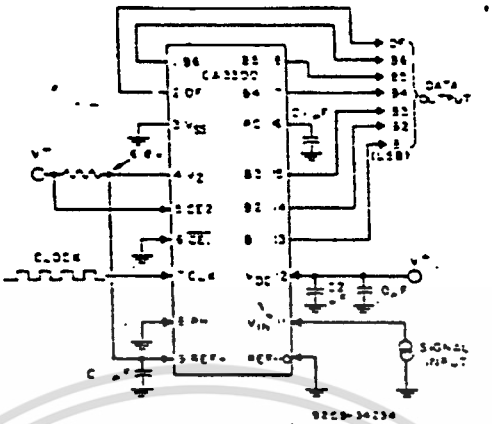


Fig. 12 - Typical CA3300 6-bit configuration 15-MHz sampling rate.

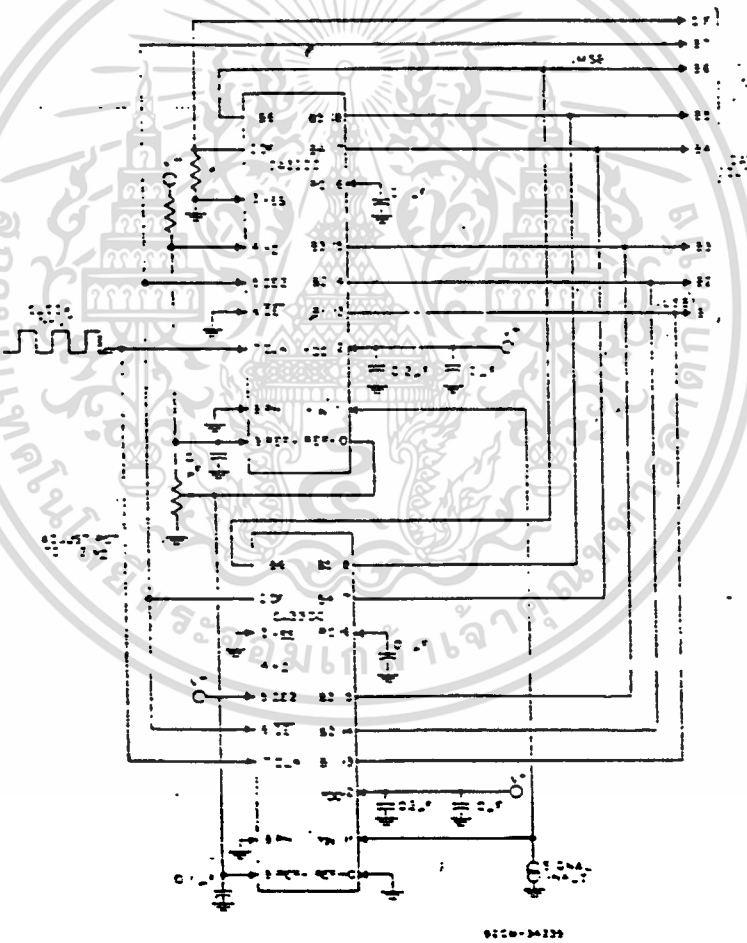


Fig. 13 - Typical CA3300 7-bit resolution configuration 15-MHz sampling rate

National Semiconductor Corporation

ADVANCED INFORMATION
April 1987

NMC61256N/NMC61256N-L 32,768 x 8-Bit Static RAM

General Description

The NMC61256N/NMC61256N-L is a 32,768 by 8-bit, new generation static RAM. It is fabricated with National's proprietary microCMOS double-polysilicon technology which combines high performance and high density with low power consumption and excellent reliability.

The NMC61256N/NMC61256N-L operates with a single 5V power supply with ±10% tolerance. Additional battery back-up operation is available (L version) for data retention down to 2V, with low standby current.

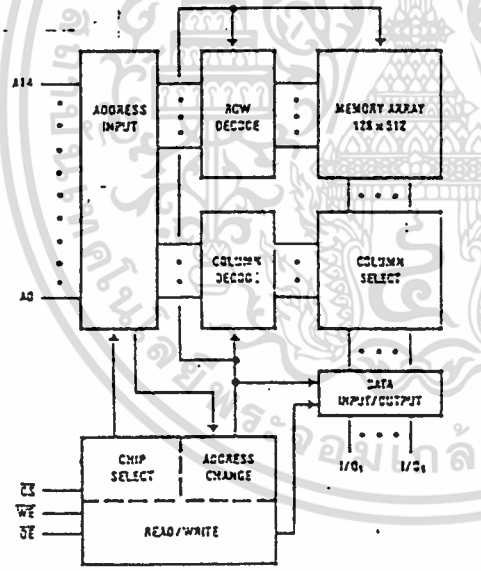
Packaging is in standard 28-pin plastic DIP.

In addition to the inputs and outputs being TTL compatible, the outputs are also CMOS compatible, in that capacitive loads are driven to V_{CC} or V_{SS}.

Features

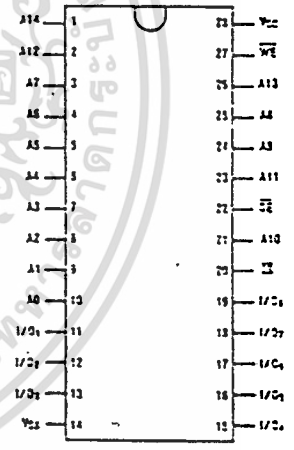
- Single power supply: 5V ±10%
- Fast access time 70 ns/100 ns/120 ns max
- Equal access and cycle times
- Completely static RAM: no clock or timing strobe required
- Low standby power and low power operation
  Standby: 50 µW, typical
  Operation: 10 mW/MHz, typical
- Battery back-up operation available (L version) with data retention supply voltage: 2V–5.5V
- Common data input and output, TRI-STATE® output
- TTL compatible: all inputs and outputs
- CMOS compatible: outputs drive capacitive loads to V_{CC} or V_{SS}
- Standard 28-pin package configuration

Block and Connection Diagrams



TL/D/5807–1

Dual-In-Line Package



TL/D/5807–2

Top View

Order Number NMC61256N or NMC61256N-L
See NS Package Number N28A

Order Number / Parameter	NMC61256N-70L	NMC61256N-70	NMC61256N-100L	NMC61256N-100	NMC61256N-120L	NMC61256N-120
Access Time (ns)	70	70	100	100	120	120
I_{CC} Standby, CMOS	500 µA	2 mA	500 µA	2 mA	500 µA	2 mA

Absolute Maximum Ratings

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

Voltage on Any Pin Relative to V_{SS}	−0.6V to +7V
Storage Temperature, T_{STG}	−55°C to +125°C
Temperature Under Bias, T_{BIAS}	−10°C to +85°C
Power Dissipation, P_D	1.0W
Current Through Any Pin	100 mA

ESD rating to be determined.

Recommended DC Operating Conditions

	Min	Max	Units
V_{CC} Supply Voltage	4.5	5.5	V
V_{SS} Supply Voltage	0	0	V
V_{IH}, Input High Voltage (Logic 1)			
TTL	2.2	6.0	V
CMOS	$V_{CC}-0.2$	$V_{CC}+0.2$	V
V_{IL}, Input Low Voltage (Logic 0)			
TTL	−0.3	0.8	V
CMOS	−0.3	0.2	V
T_{OPR}, Operating Temp	0	70	°C

DC Electrical Characteristics at recommended operating conditions

Symbol		Parameter	Conditions	Min	Max	Units
I_{LI}		Input Leakage Current	$V_{IN} = V_{SS}$ to V_{CC}	−2	2	µA
I_{LO}		Output Leakage Current	$\overline{CS}$ or $\overline{CE} = V_{IH}$ $V_{I/O} = V_{SS}$ to V_{CC}	−2	2	µA
I_{CC}		Active Quiescent Current, TTL	All Inputs at TTL Levels $\overline{CS} = V_{IL}$ TTL, $I_{I/O} = 0$ mA		25	mA
I_{CC}	Std.	Active Quiescent Current, CMOS	All Inputs at CMOS Levels $\overline{CS} = V_{IL}$ CMOS, $I_{I/O} = 0$ mA		2	mA
	L				500	µA
I_{CC1}		Average Operating Current, TTL	$T_{RC} = T_{RC}$ Min $\overline{CS} = V_{IL}$ TTL, $I_{I/O} = 0$ mA All Inputs at TTL Levels		50	mA
		Average Operating Current, CMOS	$T_{RC} = T_{RC}$ Min $\overline{CS} = V_{IL}$ TTL, $I_{I/O} = 0$ mA All Inputs at CMOS Levels		30	mA
I_{SB}	Std.	Standby Power Supply Current	$\overline{CS} = V_{IH}$ TTL $I_{I/O} = 0$ mA		4	mA
	L				2	mA
I_{SB1}	Std.	Standby Power Supply Current	$\overline{CS} = V_{IH}$ CMOS		2	mA
	L				500	µA
V_{OL}		Output Low Voltage, TTL	$I_{OL} = 8$ mA		0.4	V
		Output Low Voltage, CMOS	$I_{OL} = 2.0$ µA	−0.2	0.2	V
V_{OH}		Output High Voltage, TTL	$I_{OH} = -4$ mA	2.4		V
		Output High Voltage, CMOS	$I_{OH} = 2.0$ µA	$V_{CC}-0.2$	$V_{CC}+0.2$	V

Capacitance

Symbol	Parameter	Conditions	Max	Units
C_{IN}	Input Capacitance	$V_{IN} = 0V$ (Note 5)	5	pF
$C_{I/O}$	Input/Output Capacitance	$V_{I/O} = 0V$ (Note 5)	10	pF

Truth Table

Mode	$\overline{WE}$	$\overline{CS}$	$\overline{CE}$	I/O	Current
Not Selected (Power Down)	•	H	•	Hi-Z	I_{SB}, I_{SB1}
Output Disabled	H	L	H	Hi-Z	I_{CC}, I_{CC1}
Read	H	L	L	D_{OUT}	I_{CC}, I_{CC1}
Write	L	L	•	D_{IN}	I_{CC}, I_{CC1}

• = Don't care (H or L), H = Logic HIGH Level, L = Logic LOW Level

AC Electrical Characteristics* (Note 1)

Symbol	Parameter	NMC61256N/NMC61256N-L						Units
		-70		-100		-120		
		Min	Max	Min	Max	Min	Max	
READ CYCLE (Note 4)								
t_{RC}	Read Cycle Time	70		100		120		ns
t_{AA}	Address Access Time		70		100		120	ns
t_{CO}	Chip Selection ($\overline{CS}$) to Output Valid		70		100		120	ns
t_{OE}	Output Enable ($\overline{OE}$) to Output Valid		30		50		60	ns
t_{LZ}	Chip Selection ($\overline{CS}$) to Output Active (Note 11)	15		15		15		ns
t_{OLZ}	Output Enable ($\overline{OE}$) to Output Active (Note 11)	5		5		5		ns
t_{HZ}	Chip Deselection (CS) to Output in Hi-Z (Notes 2 and 3)	0	30	0	35	0	40	ns
t_{OHZ}	Output Disable ($\overline{OE}$) to Output in Hi-Z (Notes 2 and 3)	0	25	0	35	0	40	ns
t_{OHA}	Output Hold from Address Change	5		10		10		ns
WRITE CYCLE								
t_{WC}	Write Cycle Time	70		100		120		ns
t_{CW}	Chip Selection ($\overline{CS}$) to End of Write (Note 10)	60		80		85		ns
t_{AS}	Address Setup Time (Note 7)	0		0		0		ns
t_{AW}	Address Valid to End of Write	60		80		85		ns
t_{WP}	Write Pulse Width (Note 6)	40		60		70		ns
t_{WR}	Write Recovery Time from $\overline{CS}$ (Note 8)	0		0		0		ns
t_{WHZ}	Beginning of Write to Output in Hi-Z (Note 9)	0	25	0	35	0	40	ns
t_{DW}	Data Valid to Write Time Overlap	30		35		40		ns
t_{DH}	Data Hold from End of Write	0		0		0		ns
t_{OHZ}	Output Disable ($\overline{OE}$) to Output in Hi-Z	0	25	0	35	0	40	ns
t_{OW}	Output Active from End of Write	0		0		0		ns

*Applies to Standard and L Versions.

Note 1: AC test conditions T_A = 0°C to +70°C, V_{CC} = 5V ±10%.

Note 2: t_{HZ} and t_{OHZ} are defined as the time at which the outputs achieve the open circuit condition and are determined as:
High to TRI-STATE, measured V_{OH} (DC) − 0.10V
Low to TRI-STATE, measured V_{OL} (DC) + 0.10V

Note 3: At any given temperature and voltage condition, $t_{HZ\ MAX}$ is less than $t_{LZ\ MIN}$ both for a given device and from device to device (guaranteed not tested).

Note 4: $\overline{WE}$ is high for read cycle.

Note 5: T_A = 25°C, f = 1.0 MHz. This parameter is sampled and not 100% tested.

Note 6: A write occurs during the overlap (t_{WP}) of a low $\overline{CS}$ and a low $\overline{WE}$.

Note 7: t_{AS} is measured from the address changes to the beginning of the write.

Note 8: t_{WR} is measured from the earliest of $\overline{CS}$ or $\overline{WE}$ going high to the end of the write cycle.

Note 9: If $\overline{CS}$ is low during this period, I/O pins are in the output state. At this time, the data input signals of opposite phase to the outputs must not be applied.

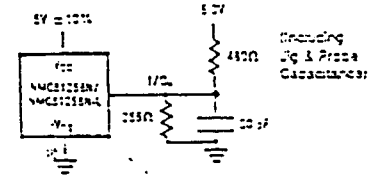
Note 10: If the $\overline{CS}$ low transition occurs simultaneously with the $\overline{WE}$ low transition or after the $\overline{WE}$ transition, the outputs will remain in a Hi-Z state.

Note 11: Output active level is defined as steady state TRI-STATE level ±0.1V.

AC Test Conditions

Input pulse levels:	V_{IH} = 3.0V, V_{IL} = 0.0V
Input rise and fall times:	5 ns
All input timing reference levels:	1.5V
Output timing reference levels:	V_{OH} = 2.0V, V_{OL} = 0.8V

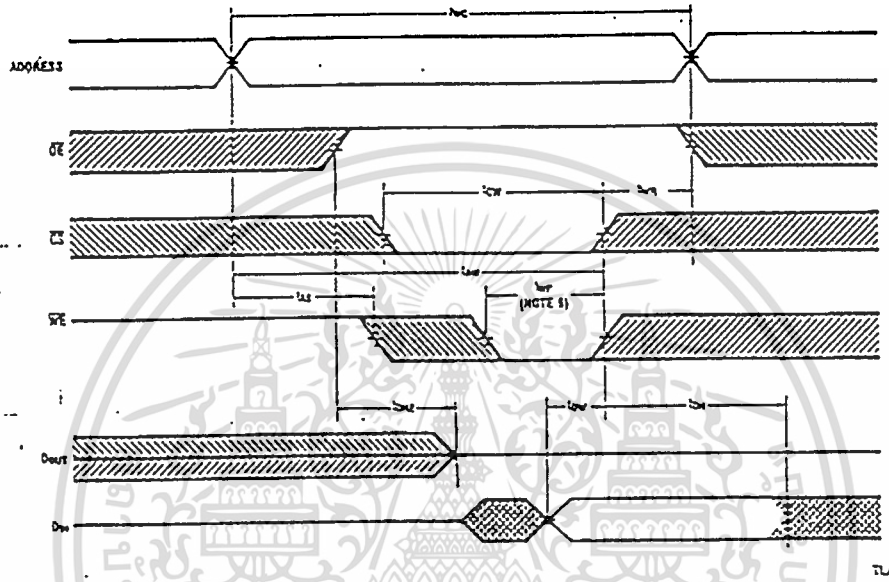
AC Test Load



TL/D/9807-3

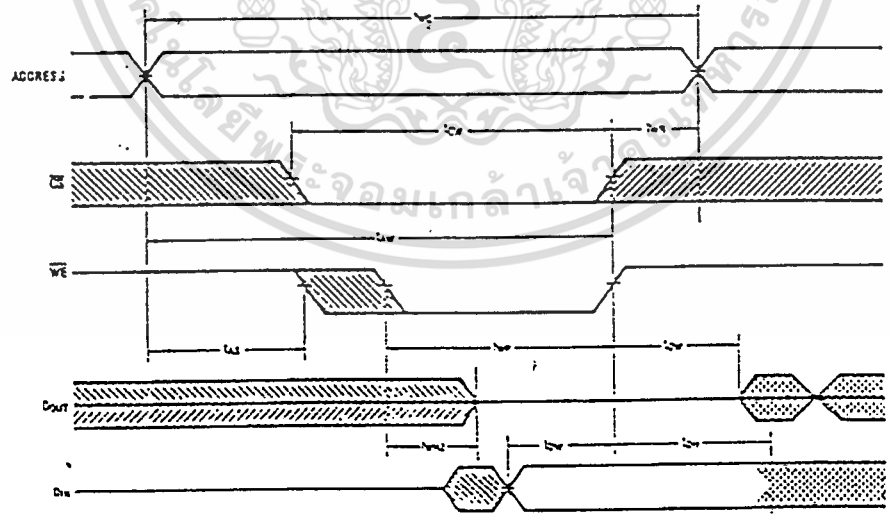
Timing Waveforms

Write Cycle 1 ($\overline{OE}$ Clocked)



TL/D/5807–4

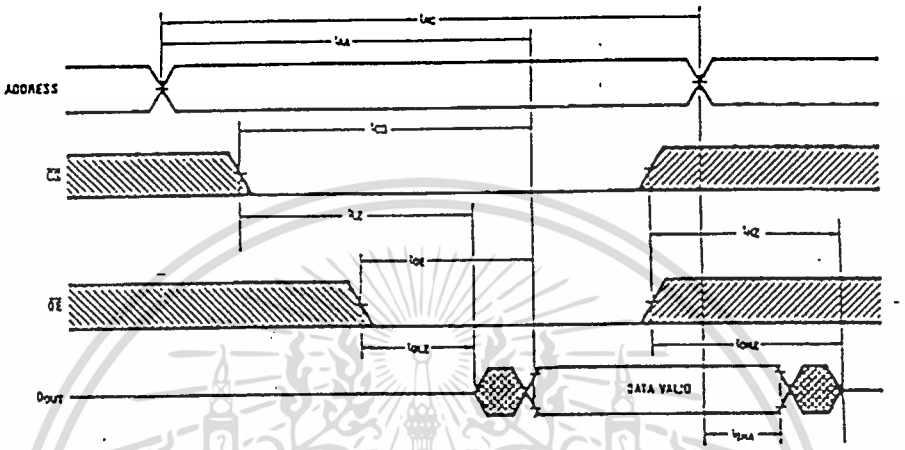
Write Cycle 2 ($\overline{OE}$ Low Fixed)



TL/D/5807–5

Timing Waveforms (Continued)

Read Cycle ($\overline{WE}$ = V_{IH})

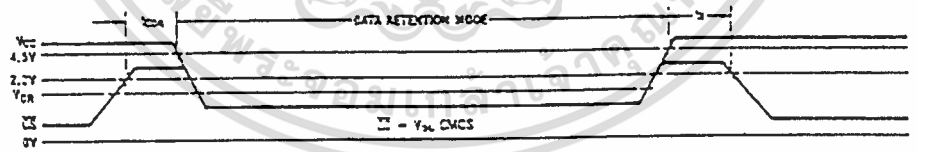


TL/D/8807–6

Low V_{CC} Data Retention (L Version)

Symbol	Parameter	Conditions	Min	Max	Units
V_{DR}	V_{CC} for Data Retention	$\overline{CS}$ > V_{IH}, CMOS	2.0	5.5	V
I_{CCDR}	Data Retention Current	V_{CC} = 2V $\overline{CS}$ > V_{IH}, CMOS		200	µA
t_{CDR}	Chip Deselect to Data Retention Time	See Retention Waveform	0		ns
t_R	Operation Recovery Time	See Retention Waveform	t_{RC}		ns

Low V_{CC} Data Retention Waveform



TL/D/8807–7

CMOS™

CA82C54

PROGRAMMABLE INTERVAL TIMER

- A high performance device featuring pin and functional compatibility with the industry standard 8254
- Supports 8086/88 and 80186/188 microprocessors
- High Speed: *zero wait state* 10 MHz and 8 MHz versions available
- Low power CMOS implementation
- TTL input/output compatibility
- Compatible with 8080/85 , 8086/88, 80286/386 and 68000 μP families
- Fully static operation
- Three independent 16 bit counters
- Six programmable counter modes
- Status read-back command
- Binary or BCD counting

The CA82C54 is a counter/timer device that includes complete pin and functional compatibility with the industry standard 8254. Designed for fast 10 MHz operation, it has three independently programmable 16 bit counters and six programmable counter modes. Counting can be performed in both binary and BCD formats.

The CA82C54 offers a very flexible, hardware solution to the generation of accurate time delays in microprocessor systems. A general purpose, multi-timing element, it can be used to implement event counters, elapsed time indicators, waveform generators plus a host of other functions.

The low power consumption of the CA82C54 makes it ideally suited to portable systems or those with low power standby modes.

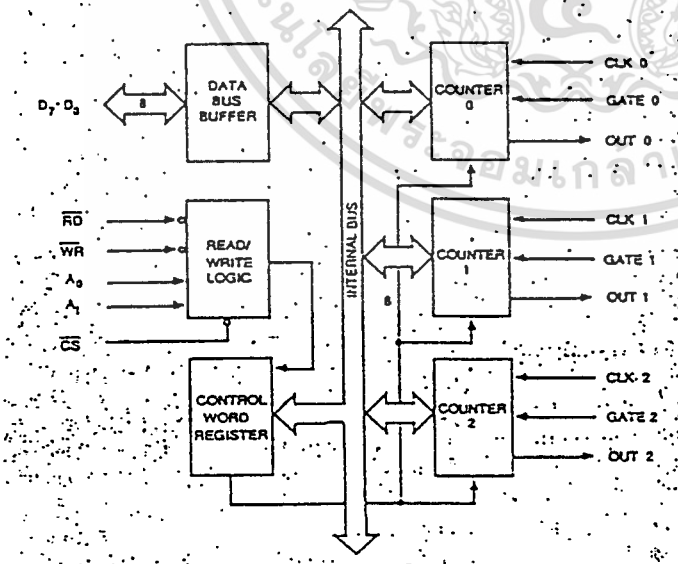


Figure 1 : CA82C54 BLOCK DIAGRAM

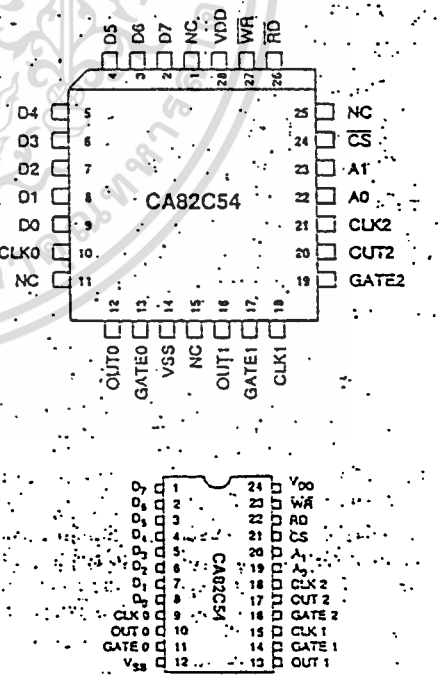


Figure 2 : PLCC and DIP PIN CONFIGURATIONS

Symbol	Pins		Type	Name and Function
	PLCC	DPIP		
A_0, A_1	22, 23	19, 20	I	**Address:** These two address pins are used to select the *Control Word Register* (for read or write operations), or one of the three Counters. They are normally connected to the system address bus. A_1 A_0 Selects: 0 0 Counter 0 0 1 Counter 1 1 0 Counter 2 1 1 Control Word Register
CLK 0	10	9	I	**Clock 0:** Clock input of Counter 0.
CLK 1	18	15	I	**Clock 1:** Clock input of Counter 1.
CLK 2	21	18	I	**Clock 2:** Clock input of Counter 2.
$\overline{CS}$	24	21	I	**Chip Select:** Active LOW control signal to enable the CA82C54 to respond to RD and WR signals. If CS is not LOW, RD and WR are ignored.
D_7 - D_0	2 - 9	1 - 8	I/O	**Data:** Bi-directional 3-state data bus lines, connected to system data bus.
GATE 0	13	11	I	**Gate 0:** Gate input of Counter 0.
GATE 1	17	14	I	**Gate 1:** Gate input of Counter 1.
GATE 2	19	16	I	**Gate 2:** Gate input of Counter 2.
OUT 0	12	10	O	**Output 0:** Output of Counter 0.
OUT 1	16	13	O	**Output 1:** Output of Counter 1.
OUT 2	20	17	O	**Output 2:** Output of Counter 2.
$\overline{RD}$	26	22	I	**Read Control:** Active LOW control signal used to enable the CA82C54 for read operations by the CPU.
V_{CC}	28	24	-	**Power:** 5 v ± 10% DC Supply
V_{SS}	14	12	-	**Ground:** 0 v
$\overline{WR}$	27	23	I	**Write Control:** Active LOW control signal used to enable the CA82C54 to be written to by the CPU.

FUNCTIONAL DESCRIPTION

The CA82C54 is a versatile programmable interval timer/counter designed for use in high speed 8, 16 and 32-bit microprocessor systems. It provides a means of generating accurate time delays in hardware that is fully software configurable. It can be treated as an array of I/O ports, with minimal software overhead.

The internal structure of the CA82C54 is illustrated in the block diagram of Figure 1. Major functional blocks include a data bus buffer, read/write logic, control word register, and three programmable counters.

Data Bus Buffer Block

The 8-bit, 3-state data bus buffer provides controllable, bi-directional interface between the CA82C54 and the microprocessor system bus.

Read/Write Logic Block

The read/write logic block generates internal control signals for the different functional blocks using address and control information obtained from the system. The active LOW signals; $\overline{CS}$, $\overline{RD}$ and $\overline{WR}$ are used to select the CA82C54

for operation, read a counter, and write to a counter (or the control word register) respectively. $\overline{CS}$ *must* be LOW for $\overline{RD}$ or $\overline{WR}$ to be recognized. Note that $\overline{RD}$ and $\overline{WR}$ *must NOT* be active at the same time.

The inputs A_0 and A_1 are used to select the control word register, or one of the three counters that is to be written to or read from (see Table 1). A_0 and A_1 connect directly to the corresponding signals of the microprocessor address bus, while $\overline{CS}$ is derived from the address bus using either a linear select method, or an address-decoder device.

Control Word Register

The control word register is a write only register that is selected by the read/write logic block when A_0 and A_1 = 1. When $\overline{CS}$ and $\overline{WR}$ are LOW, data is written into the CA82C54 control word register from the CPU via the data bus buffer. Control word data is interpreted as a number of different commands which are used to program the various device functions. For example, status information is available with the Read-Back Command. These are discussed further in the section on programming.

Counter Blocks

The CA82C54 contains three identical, independent counter blocks. Each counter provides the same functions, but can be programmed to operate in different modes relative to each other. A typical CA82C54 counter is illustrated in Figure 3, and contains the following functional elements: control logic, counter, output latches, count registers and status register.

The *Control Logic* provides the interface between the counter proper, the program instructions contained in the control word register and the external signals CLK n, GATE n and OUT n. It also keeps the status register information current, controls the access of OL and CR to the internal data bus, and the loading of CE from the CR registers.

The *Counter* proper (shown in the Figure 3 as CE, for counting element) is a 16-bit presettable synchronous down counter.

The *Output Latches* (shown as OL_M and OL_L) provide a mechanism whereby the CPU can read the *current* contents of the CE. These two 8-bit latches (M for most significant byte and L for least significant byte) together form a 16-bit latch capable of holding the complete contents of the CE. Note that this arrangement is also convenient for communicating 16-bit values over the 8-bit internal data bus.

During normal operation, the contents of OL track with the contents of CE. When a Counter Latch Command is issued by the CPU to a particular counter, its OL latches the current value of CE so that it can be read by the CPU (the CE cannot be read directly). OL then returns to tracking with CE. Note that only one latch (OL_M followed by OL_L) at a time is enabled by the counter's control logic.

The *Count Registers* (shown as CR_M and CR_L) behave as input latches to the CE, and provide a mechanism whereby the initial count value can be downloaded from the CPU to the CE. Similar in operation to OL, CR is controlled by the counter control logic. When a two byte initial count is to be downloaded, it is transferred one byte at a time across the internal CA82C54 data bus to the appropriate register (CR_M if the most significant byte, CR_L otherwise). CE is loaded by transferring both bytes simultaneously from CR. Note that CR is the interface between CE and the data bus, since CE cannot be accessed directly.

Both CR_M and CR_L are cleared automatically when the counter is programmed and a new initial count is to be written. Thus, regardless of the counter's previous programming, both CR bytes will be initialized to a known zero state. This is important in the case where one byte counts are programmed (either most significant or least significant byte), so that the unused byte is always zero, and won't corrupt the inital count value loaded into CE.

The *Status Register and Latch* is used to hold the current contents of the control word register and the status of the output and null count flag (see section on Programming). The contents of the status register must be latched to become available to the data bus, where they can be read by the CPU.

Note that the Control Word Register is also shown in the Counter block diagram. While not a part of the counter proper, its contents determine the functional operation of the counter, including mode selections programmed.

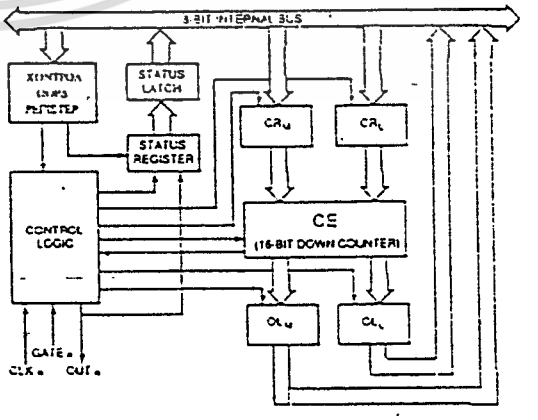


Figure 3 : BLOCK DIAGRAM of a COUNTER

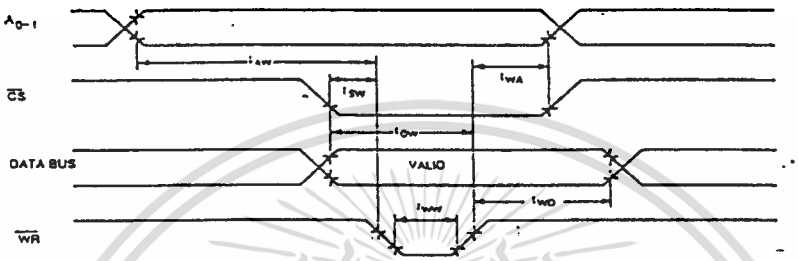
Table 2 :. AC CHARACTERISTICS (T_A = –40° to +85°C, V_{DD} = 5V ± 10%, V_{SS} = 0V) Bus Parameters[1]

Symbol	Parameter	Test Conditions	Limits (8 MHz)		Limits (10 MHz)		Units
			Min	Max	Min	Max	
t_{AD}	Data delay from address		-	220	-	185	ns
t_{AR}	Address stable before $\overline{RD}$↓		45	-	30	-	ns
t_{AW}	Address stable before $\overline{WR}$↓		0	-	0	-	ns
t_{CL}	CLK setup for count latch		–40	45	–40	40	ns
t_{CLK}	Clock period		125	DC	100	DC	ns
t_{DF}	$\overline{RD}$↑ to data floating		5	90	5	65	ns
t_{DW}	Data setup time before $\overline{WR}$↑		120	-	95	-	ns
t_F	Clock fall time		-	25	-	25	ns
t_{GH}	Gate hold time after CLK↑	Note 2	50	-	50	-	ns
t_{GL}	Gate width low		50	-	50	-	ns
t_{GS}	Gate setup time to CLK↑		50	-	40	-	ns
t_{GW}	Gate width high		50	-	50	-	ns
t_{OD}	Output delay from CLK↓		-	150	-	100	ns
t_{ODG}	Output delay from Gate↓		-	120	-	100	ns
t_{PWH}	High pulse width	Note 3	60	-	30	-	ns
t_{PWL}	Low pulse width	Note 3	60	-	50	-	ns
t_R	Clock rise time		-	25	-	25	ns
t_{RA}	Address hold time after $\overline{RD}$↑		0	-	0	-	ns
t_{RD}	Data delay from $\overline{RD}$↓		-	120	-	85	ns
t_{RR}	$\overline{RD}$ pulse width		150	-	95	-	ns
t_{RV}	Command recovery time		200	-	165	-	ns
t_{SR}	CS stable before $\overline{RD}$↓		0	-	0	-	ns
t_{SW}	CS stable before $\overline{WR}$↓		0	-	0	-	ns
t_{WA}	Address hold time $\overline{WR}$↑		0	-	0	-	ns
t_{WC}	CLK delay for loading		0	55	0	55	ns
t_{WD}	Data hold time after $\overline{WR}$↑		0	-	0	-	ns
t_{WG}	Gate delay for sampling		–5	50	–5	40	ns
t_{WO}	OUT delay from *Mode Write*		-	260	-	240	ns
t_{WW}	$\overline{WR}$ pulse width		150	-	95	-	ns

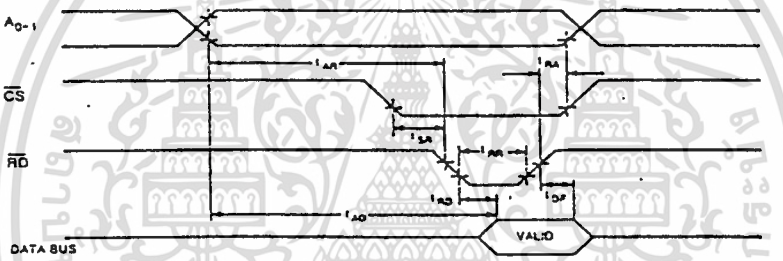
Notes:
1. AC timings measured at V_{CH} = 2.0 V, V_{CL} = 0.8 V
2. In Modes 1 and 5 triggers are sampled on each rising clock edge. A second trigger within 120 ns of the rising clock edge may not be detected. (70 ns for CA82C54–10).
3. Low-going glitches that violate t_{PWH}, t_{PWL} may cause errors requiring counter reprogramming.

Figure 4 : TIMING DIAGRAMS

a) Write Timing

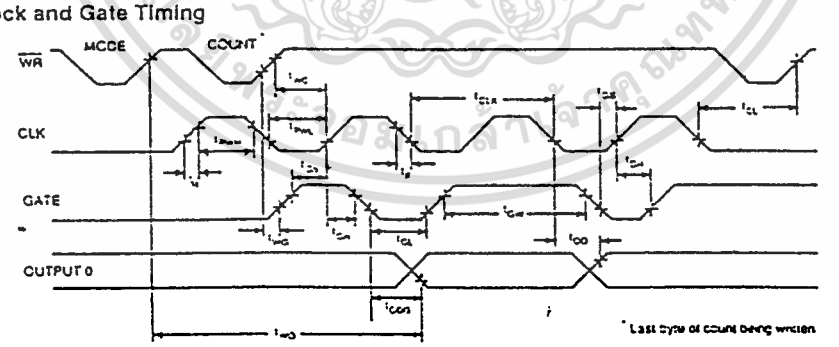


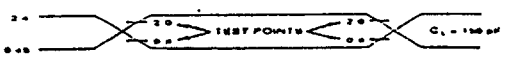
b) Read Timing



c) Recovery Timing

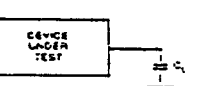
d) Clock and Gate Timing





All input signals must switch between 0.45V and 2.4V.
All timing measurements are made at 0.8V and 2.0V.

Figure 5: AC TESTING I/O WAVEFORM



C_L = 150 pF
C_L includes jig capacitance

Figure 6 : AC TESTING LOADING CIRCUIT

Table 3 : DC CHARACTERISTICS (T_A = –40°C to +85°C, V_{DD} = 5V ± 10%, V_{SS} = 0V)

Symbol	Parameter	Test Conditions	Limits Min	Limits Max	Units
I_{DD}	V_{DD} Supply Current	CLK Freq = 5 MHz, CA82C54–5 CLK Freq = 8 MHz, CA82C54–8 CLK Freq = 10 MHz, CA82C54–10	-	20	mA
I_{DDSB}	V_{DD} Standby Supply Current	CLK Freq = DC, $\overline{CS}$ = HIGH All inputs/data bus = HIGH All outputs floating	-	10	µA
I_{DDSB1}	V_{DD} Standby Supply Current	CLK Freq = DC, $\overline{CS}$ = HIGH All other inputs, outputs, I/O plus floating	-	150	µA
I_{IL}	Input Load Current	V_{IN} = V_{DD} to 0 V	-	± 20	µA
I_{OFL}	Output Float Leakage	V_{OUT} = V_{DD} to 0.45V	-	± 10	µA
V_{IH}	Input High Voltage		2.0	V_{DD} + 0.5V	V
V_{IL}	Input Low Voltage		–0.5	0.8	V
V_{OH}	Output High Voltage	I_{OH} = -400 µA	V_{DD} – 0.4V	-	V
		I_{OH} = -2.5 mA	3.0	-	V
V_{OL}	Output Low Voltage	I_{OL} = 2.5 mA	-	0.4	V

Table 4 : CAPACITANCE (T_A = 25 °C, V_{DD} = GND = 0V)

Symbol	Parameter	Test Conditions	Limits Min	Limits Max	Units
C_{IN}	Input Capacitance	FREQ = 1 MHz	-	10	pF
$C_{I/O}$	I/O Capacitance	Unmeasured pins	-	20	pF
C_{OUT}	Output Capacitance	returned to V_{SS}	-	20	pF

Table 5 : RECOMMENDED OPERATING CONDITIONS

Operating Voltage Range		+4.0 to +6.0 Volts
Operating Temperature Range	Commercial	0°C to +70°C
	Industrial	–40°C to +85°C
	Military	–55°C to +125°C

Table 6 : ABSOLUTE MAXIMUM RATINGS

Ambient Temperature Under Bias	0°C to 70°C
Storage Temperature	–65°C to +150°C
Voltage on Any Pin with Respect to Ground	–0.5 V to +7 V
Power Dissipation	1 Watt

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

PROGRAMMING

When installed in microprocessor systems the CA82C54 Programmable Interval Timer appears to the system software as an array of peripheral I/O ports, namely: three counters, and a control register for MODE programming.

After power-up, the state of the CA82C54 is undefined. Thus, the mode, the count value and the output of all the counters are undefined. The subsequent operation of each counter is determined when it is programmed. *and each counter must be programmed before it can be used.* Unused counters need not be programmed.

Counters are programmed by writing a Control Word and then an initial count value for the counter in question. All Control Words are written into the Control Word Register, which is selected when A_1, A_0 = 11. The Control Word itself specifies which counter is being programmed. It is illustrated in Figure 7.

By contrast, initial counts are written into the counters, not the Control Word Register. A_1, A_0 inputs are used to select the counter to be written into. The format of the initial count is determined by the Control Word used.

A number of commands are available to the user which allow access to the programmable features of the CA82C54. These are described in detail below.

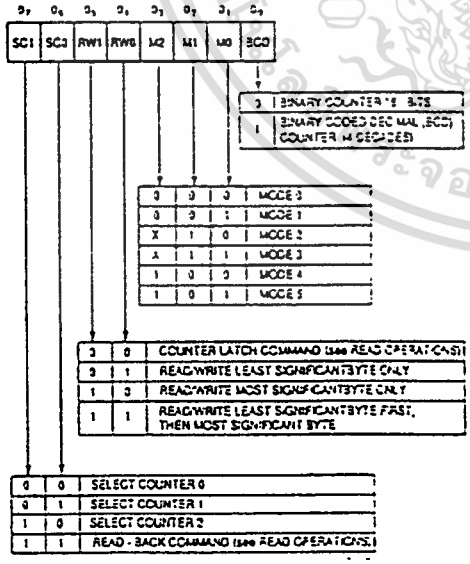


Figure 7 : CONTROL WORD FORMAT

Write Operations

Programming for the CA82C54 is very flexible, and requires that only two conventions be followed:

- For each Counter, the Control Word must be written before the initial count is written.
- The initial count must follow the count format specified in the Control Word (least significant byte only, most significant byte only, or least significant byte and then most significant byte).

Since the Control Word Register and the three Counters have separate addresses (selected by the A_1, A_0 inputs), and each Control Word specifies the Counter it applies to (SC0, SC1 bits), no special instruction sequence is required. Any programming sequence that follows the conventions above is acceptable.

A new initial count may be written to a Counter at any time without affecting the Counter's programmed Mode in any way. Counting will be affected as described in the Mode definitions. The new count must follow the programmed count format.

If a Counter is programmed to read/write 2-byte counts, the following precaution applies: a program must not transfer control (between writing the first and second byte) to another routine which also writes into the same Counter. Otherwise, the Counter will be loaded with an incorrect count.

Example	Step	Operation	A_1	A_0
1	1	Control Word-Counter 0	1	1
	2	LSB of count-Counter 0	0	0
	3	MSB of count-Counter 0	0	0
	4	Control Word-Counter 1	1	1
	5	LSB of count-Counter 1	0	1
	6	MSB of count-Counter 1	0	1
	7	Control Word-Counter 2	1	1
	8	LSB of count-Counter 2	1	0
	9	MSB of count-Counter 2	1	0
2	1	Control Word-Counter 1	1	1
	2	Control Word-Counter 0	1	1
	3	LSB of count-Counter 1	0	1
	4	Control Word-Counter 2	1	1
	5	LSB of count-Counter 0	0	0
	6	MSB of count-Counter 1	0	1
	7	LSB of count-Counter 2	1	0
	8	MSB of count-Counter 0	0	0
	9	MSB of count-Counter 2	1	0

Note: In both examples, all counters are programmed to read/write 2-byte counts. These are two of many programming sequences.

Figure 8 : SAMPLE PROGRAMMING SEQUENCES

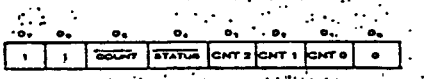
The Read–Back Command may also be used to latch status information of selected counter(s) by setting bit $D_4 = 0$ (STATUS). Status must be latched to be read; as counter status is obtained by a read from that counter.

The counter status format is shown in Figure 11. Bits D_5 through D_0 contain the counter's programmed Mode exactly as written in the last Mode Control Word. OUTPUT bit D_7 contains the current state of the OUT pin. This allows the user to monitor counter output via software, thus eliminating some hardware from a system.

NULL COUNT bit D_6 indicates when the last count written to the Counter Register (CR) has been loaded into the counting element (CE). The exact time this happens depends on the Mode of the counter and is described in the Mode Definitions. Until the count is loaded into the counting element (CE), it can't be read from the counter. If the count is latched or read before this time, the count value will not reflect the new count just written. The operation of Null Count is shown in Figure 12.

If multiple status latch operations of the counter(s) are performed without reading the status, all but the first are ignored. That is, the status that will be read is the status of the counter at the time the first status Read–Back Command was issued.

Both count and status of the selected counter(s) may be latched simultaneously by setting both COUNT and STATUS bits $D_5, D_4 = 0$. This is functionally the same as issuing two separate Read–Back Commands at once, and the above discussions apply here also. Specifically, if multiple count and/or status Read–Back Commands are issued to the same counter(s) without any intervening reads, all but the first are ignored. This is illustrated in Table 7.



Base Conditions:
- $A_1, A_0 = 1\ 1$
- $\overline{CS} = 0$
- $\overline{RD} = 1$
- $\overline{WR} = 0$

D_5: 0 = Latch count of selected counter(s)
D_4: 0 = Latch status of selected counter(s)
D_3: 1 = Select Counter 2
D_2: 1 = Select Counter 1
D_1: 1 = Select Counter 0

Figure 10 : READ–BACK COMMAND FORMAT

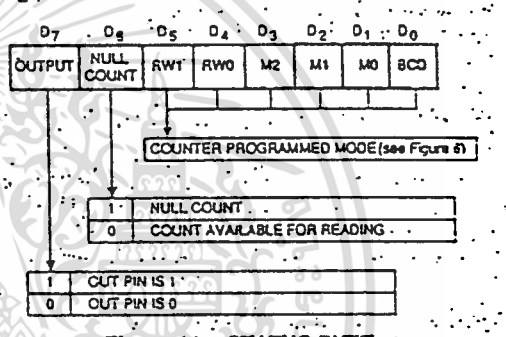


Figure 11 : STATUS BYTE

	This Action:	Causes:
A	Write to the control word register[1]	Null Count = 1
B	Write to the count register (CR)[2]	Null Count = 1
C	New count is loaded into CE (CR → CE)	Null Count = 0

Notes:
1. Only the counter specified by the control word will have its null count set = 1. Null counts of other counters are unaffected.
2. If the counter is programmed for two-byte counts (least significant byte then most significant byte), null count goes to 1 when second byte is written.

Figure 12 : NULL COUNT OPERATION

Table 7 : READ–BACK COMMAND EXAMPLE

Command Description	Result	Command Word							
		D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Read back count and status of Counter 0	Count and status latched for Counter 0	1	1	0	0	0	0	1	0
Read back status of Counter 1	Status latched for Counter 1	1	1	1	0	0	1	0	0
Read back status of Counters 2, 1	Status latched for Counter 2 *only*	1	1	1	0	1	1	0	0
Read back count of Counter 2	Count latched for Counter 2	1	1	0	1	1	0	0	0
Read back count and status of Counter 1	Count latched for Counter 1, *but not status*	1	1	0	0	0	1	0	0
Read back status of Counter 1	*Command ignored*, status already	1	1	1	0	0	0	1	0

OPERATIONAL DESCRIPTION

The following operations are common to all modes.

Control Word: When a Control Word is written to a Counter, all Control Logic is Reset, and OUT is initialized to a known state. No CLK pulses are needed.

Gate: The GATE input is always sampled on the rising edge [illegible] the GATE input is level

Counter: New counts are loaded, with the largest possible initial count being zero (0), equivalent to 2^{16} for binary counting and 10^4 for BCD counting, as in Table 8.

Counters decremented on the falling edge of CLK do not stop when they reach zero. In Modes 0, 1, 4, and 5 the Counters *wrap around* to the highest count (either FFFF

Read Operations

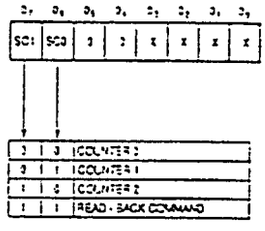
It is often desirable to read the value of a counter without disturbing the count in progress, a procedure easily accomplished in the CA82C54.

There are three possible methods for reading the counters. The first is through the Read–Back Command. The second is a simple read operation of the counter, which is selected with the A_1, A_0 inputs. The only requirement is that the CLK input of the selected counter must be inhibited by using either the GATE input or external logic; or the count must first be latched. Otherwise, the count may be in process of changing when it is read, giving an undefined result. The third method involves a special software command called the Counter Latch Command, described in more detail below.

Counter Latch Command

The Counter Latch Command, like a Control Word, is written to the Control Word Register, which is selected when A_1, A_0 = 11. Also like a Control Word, the SC0, SC1 bits select one of the three Counters, but two other bits, D_5 and D_4, distinguish this command from a Control Word.

The selected counter's output latch (OL) latches the count at the time the Counter Latch Command is received. This count is held in the latch until it is read by the CPU (or until the counter is reprogrammed). The count is then unlatched automatically and the OL returns to *following* the counting element (CE). This allows reading the contents of the Counters *on the fly* without affecting counting in progress. Multiple Counter Latch Commands may be used to latch more than onecounter. Each latched Counter's OL holds its count until it is read. Counter Latch Commands do not affect the programmed Mode of the counter in any way.



Base Conditions:
- A_1, A_0 = 1 1
- RD = 1
- CS = 0
- WR = 0

SC1, SC0 - specify counter to be latched

D_5, D_4 - 00 designates Counter Latch Command

X - don't care

Note: Don't care bits (X) should be 0 to insure compatibility with future Newbridge Microsystem products.

Figure 9 : COUNTER LATCH COMMAND FORMAT

If a counter is latched and then, some time later, latched again before the count is read, the second Counter Latch Command is ignored. The count read will be the count at the time the first Counter Latch Command was issued.

With either method, the count must be read according to the programmed format; specifically, if the counter is programmed for two byte counts, two bytes must be read. The two bytes do not have to be read one right after the other; read or write or programming operations of other counters may be inserted between them.

Another feature of the CA82C54 is that reads and writes of the same counter may be interleaved; for example, if the the counter is programmed for two byte counts, the following sequence is valid:

1. Read least significant byte.
2. Write new least significant byte.
3. Read most significant byte.
4. Write new most significant byte.

If a counter is programmed to read and write two-byte counts, the following precaution applies: a program must not transfer control (between reading first and second byte) to another routine which also reads from that same counter. Otherwise, an incorrect count will be read.

Read–Back Command

The Read–Back Command allows the user to check the count value, the programmed Mode and the current state of the OUT pin and Null count flag of the selected counter(s).

The command is written into the Control Word Register and has the format shown in Figure 10. The command applies to the counters selected by setting their corresponding bits D_3, D_2, D_1 = 1.

The Read–Back Command may be used to latch multiple counter output latches (OL) by setting the COUNT bit D_5 = 0 and selecting the desired counter(s). This single command is functionally equivalent to several counter latch commands, one for each counter latched. Each counter's latched count is held until it is read (or the counter is reprogrammed). That counter is automatically unlatched when read, but other counters remain latched until they are read. If multiple countRead–Back Commands are issued to the same counter without reading the count, all but the first are ignored. That is, the count which will be read is the count at the time the first Read–Back Command was issued.

MODE DEFINITIONS

The following terms are useful in describing the operation of the CA82C54.

- CLK pulse: A rising edge, followed by a falling edge, of a Counter's CLK input.
- Trigger: A rising edge of a Counter's GATE input.
- Counter loading: Transfer of a count from the CR to the CE (see Functional Description)

Mode 0 : Interrupt on Terminal Count

Mode 0 is typically used for event counting. After the Control Word is written, OUT is set *low*, and remains low until the Counter reaches zero. OUT then goes *high* and remains high until a new count or a new Mode 0 Control Word is written into the Counter.

GATE = 1 enables counting; GATE = 0 disables counting. GATE has no effect on OUT.

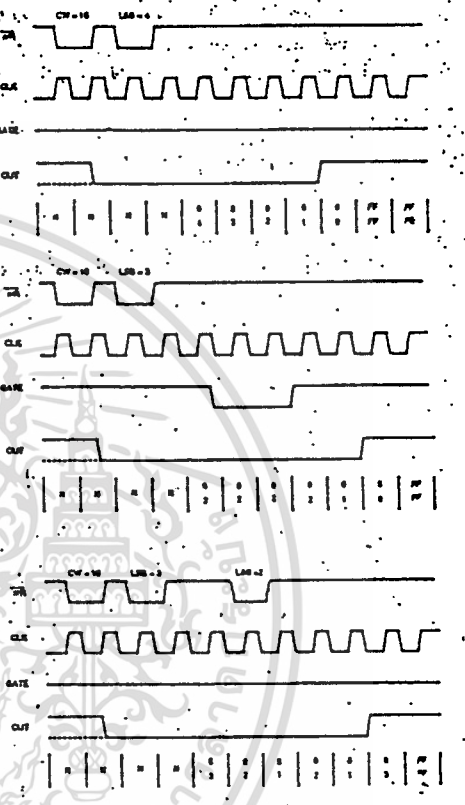
After a Control Word and initial count are written to a Counter, the initial count is loaded on the next CLK pulse. Since this CLK pulse does not decrement the count, OUT does not go high until N + 1 CLK pulses after the initial count is written (where N is the initial count value).

If a new count is written to the Counter, it is loaded on the next CLK pulse and counting continues from the new count. If a two-byte count is written, the following happens:

1. Writing the first byte disables counting. OUT is set *low* immediately (no clock pulse required).
2. Writing the second byte allows the new count to be loaded on the next CLK pulse.

This allows the counting sequence to be synchronized by software. Again, OUT does not go high until N + 1 CLK pulses after the new count of N is written.

If an initial count is written while GATE = 0, it will still be loaded on the next CLK pulse. When GATE goes high, OUT will go high N CLK pulses later. A CLK pulse is not required to load the Counter as this has already been done.



Notes: These conventions apply to all mode timing diagrams:

1. Counters are programmed for binary (not BCD) counting and for reading/writing least significant byte (LSB) only.
2. The counter is always selected (CS always low).
3. CW stands for *Control Word*; CW = 10 means a control word of 10, hex is written to the counter.
4. LSB is the *Least Significant Byte* of count.
5. Numbers below diagrams are count values.
   The lower number is the least significant byte.
   The upper number is the most significant byte.
   Since the counter is programmed to read/write LSB only, the most significant byte cannot be read.
   N stands for an undefined count.
   Vertical lines show transitions between count values.

Figure 13 : MODE 0 TIMING

Mode 1 : Hardware Retriggerable One-Shot

OUT is initially *high*. To begin the one-shot pulse, OUT goes *low* on the CLK pulse following a trigger and remains low until the Counter reaches zero. OUT then goes *high* and remains high until the CLK pulse following the next trigger.

After a Control Word and initial count have been written, the Counter is armed. A trigger causes the Counter to be loaded and OUT to be set *low* on the next CLK pulse, starting the one-shot pulse. An initial count of N results in a one-shot pulse N CLK cycles long. Since the one-shot is retriggerable, OUT remains low for N CLK pulses after any trigger. The one-shot pulse can be repeated without rewriting the same count into the counter. GATE has no effect on OUT.

If a new count is written to the Counter during a one-shot pulse, the current one-shot is not affected *unless* the Counter is retriggered. In this case, the new count is loaded into the Counter and the one-shot pulse continues for the duration of the count.

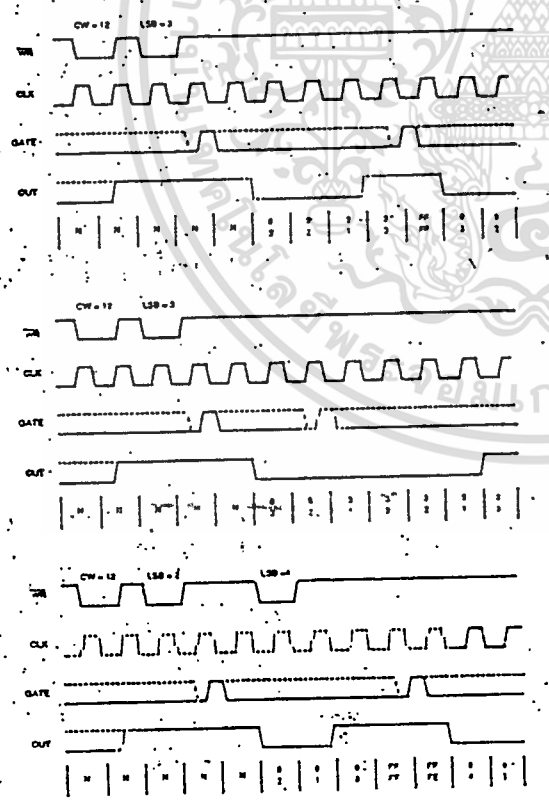


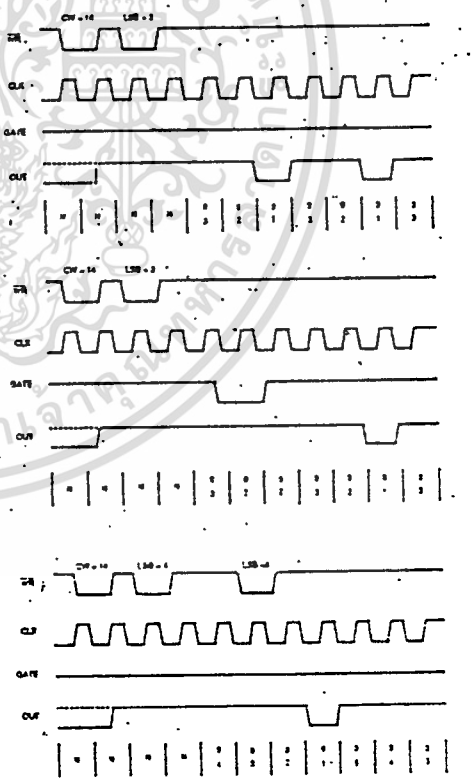
Figure 14 : MODE 1 TIMING

Mode 2 : Rate Generator

This mode functions like a divide-by-N counter and is typically used for generating Real Time Clock Interrupts. OUT is initially *high*. When the initial count has decremented to 1, OUT goes *low* for one CLK pulse, then *high* again. The Counter reloads the initial count and the process is repeated. Mode 2 is periodic, with the same sequence repeated indefinitely. For an initial count of N, the sequence repeats every N CLK cycles.

GATE = 1 enables counting; GATE = 0 disables counting. If GATE goes low during an output pulse, OUT is set *high* immediately. A trigger reloads the initial count into the Counter on the next CLK pulse; OUT goes low N CLK pulses after the trigger. Thus the GATE input can be used to synchronize the Counter.

After a Control Word and initial count have been written, the Counter is loaded on the next CLK pulse. OUT goes *low* N CLK pulses after the initial count is written, which allows the Counter to be synchronized by software.



Note: A GATE transition should *not* occur one clock cycle prior to reaching the terminal count (TC).

Figure 15 : MODE 2 TIMING

Writing a new count while counting does not affect the current counting sequence. If a trigger is received after a new count is written, but before the end of the current period, the Counter is loaded with the new count on the next CLK pulse and counting continues from the new count. Else, the new count is loaded at the end of the current counting cycle. In Mode 2, a count of 1 is illegal.

Mode 3 : Square Wave Mode

Mode 3 is typically used for Baud rate generation, and is similar to Mode 2 except for the duty cycle of OUT. OUT is initially *high*. When half the initial count has expired, OUT goes *low* for the remainder of the count. Mode 3 is also periodic; with the sequence above repeated indefinitely. An initial count of N results in a square wave with a period of N CLK cycles.

GATE = 1 enables counting; GATE = 0 disables counting. If GATE goes *low* while OUT is low, OUT is set *high* immediately (no CLK pulse is needed). A trigger reloads the Counter with the initial count on the next CLK pulse. Thus the GATE input can be used to synchronize the Counter.

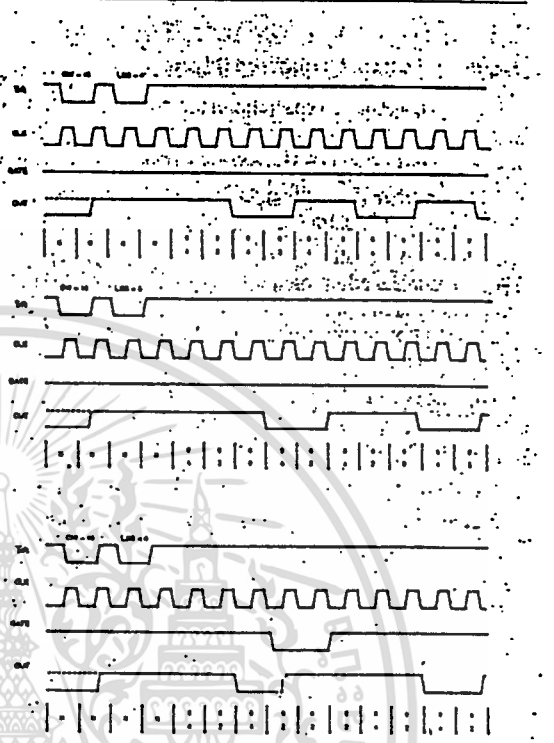
The Counter is loaded on the next CLK pulse after a Control Word and initial count have been written. This allows the Counter to be synchronized by software.

Writing a new count while counting does not affect the current counting sequence. If a trigger is received after writing a new count but before the end of the current half-cycle of the square wave, the Counter is loaded with the new count on the next CLK pulse and counting continues from the new count. Otherwise, the new count is loaded at the end of the current half-cycle.

Mode 3 is implemented as follows according to whether the initial count value is even or odd:

Even counts: OUT is initially *high*. The initial count is loaded on one CLK pulse and then decremented by two on succeeding CLK pulses. When the count expires, OUT goes *low* and the Counter is reloaded with the initial count. The above process is repeated indefinitely.

Odd counts: OUT is initially *high*. The initial count minus one (to give an even number) is loaded on one CLK pulse and then decremented by two on succeeding CLK pulses. One CLK pulse *after* the count expires, OUT goes *low* and the Counter is reloaded with the initial count minus one. Succeeding CLK pulses decrement the count by two. When the count expires, OUT goes *high* again and the Counter is reloaded with the initial count minus one. The above process is repeated indefinitely. So for odd counts, OUT is high for (N + 1)/2 counts and low for (N - 1)/2 counts.



Note: A GATE transition should *not* occur one clock prior to terminal count.

Figure 16 : MODE 3 TIMING

Mode 4 : Software Triggered Strobe

OUT is initially *high.* When the initial count expires, OUT goes low for one CLK pulse and then goes *high* again. Counting sequence is *triggered* by writing initial count.

GATE = 1 enables counting; GATE = 0 disables counting. GATE has no effect on OUT.

The Counter is loaded on the next CLK pulse after a Control Word and initial count have been written. This CLK pulse does not decrement the count, so for an initial count of N, OUT does not strobe low until N + 1 CLK pulses after the initial count is written.

If a new count is written during counting, it is loaded on the next CLK pulse and counting continues from the new count. If a two-byte count is written, the following events occur:

1. Writing the first byte has no effect on counting.
2. Writing the second byte allows the new count to be loaded on the next CLK pulse.

This allows the sequence to be *retriggered* by software. OUT strobes low N + 1 CLK pulses after the new count of N is written.

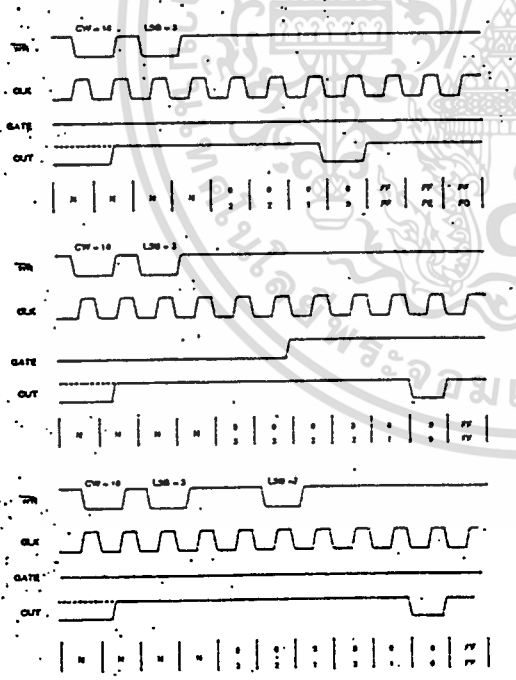


Figure 17 : MODE 4 TIMING

Mode 5 : Retriggerable Hardware Triggered Strobe

OUT is initially *high.* Counting is triggered by a rising edge of GATE. When the initial count has expired, OUT goes *low* for one CLK pulse; then goes *high* again.

After a Control Word and initial count has been written, the counter is loaded on the first CLK pulse following a trigger. This CLK pulse does not decrement the count, so, given an initial count of N, OUT does not strobe low until N + 1 CLK pulses after a trigger.

A trigger causes the Counter to be loaded with the initial count on the next CLK pulse. The counting sequence is retriggerable, so OUT will not go *low* until N + 1 CLK pulses after any trigger. GATE has no effect on OUT.

If a new count is written during counting, the current counting sequence will not be affected. If a trigger occurs after the new count is written, but before the current count expires, the Counter will be loaded with the new count on the next CLK pulse and counting will continue from there.

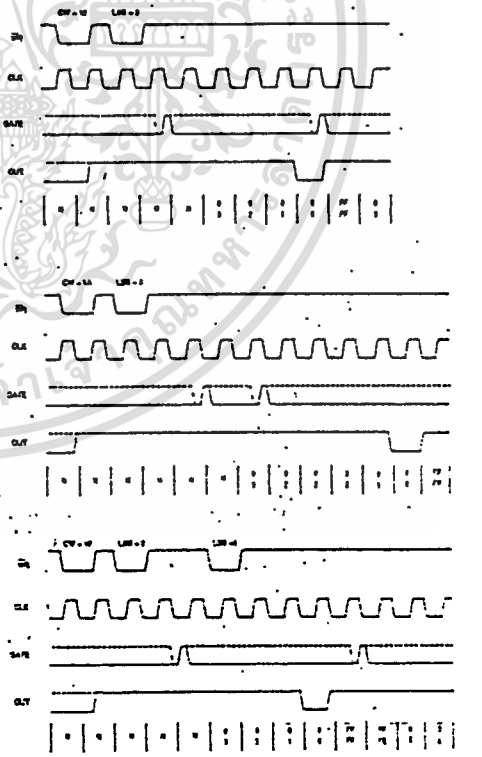


Figure 18 : MODE 5 TIMING

บรรณานุกรม

- ชานินทร์ ถาวรสาสนวงศ์, ทินกร ดุ๊ก, "การอินเตอร์เฟส IBM PC", บริษัท ฟิสิกส์เซ็นเตอร์การพิมพ์ จำกัด
- จิติ หนูแก้ว, "เทคนิคการเชื่อมต่อ IBM PC กับอุปกรณ์ภายนอกเพื่อประยุกต์ใช้งานต่าง ๆ", บริษัท ซีเอ็ดยูเคชั่น จำกัด
- ยืน ภู่วรวรรณ, "เทคโนโลยีฮาร์ดแวร์ IBM PC", บริษัท ซีเอ็ดยูเคชั่น จำกัด
- สมพัฒน์ รุ่งตะวันเรืองศรี, "เรียนรู้คอมพิวเตอร์กราฟิกส์ 2 มิติด้วยภาษา C", บริษัท ซีเอ็ดยูเคชั่น จำกัด
- มัณฑนา ปราการสมุทร, "การเขียนชุดคำสั่งภาษา C", บริษัท ดวงกมลสมัย จำกัด
- มนตรี พจนารถลาวัฒย์, "การเขียนโปรแกรมคอมพิวเตอร์ด้วยเทอร์โบซี", บริษัท ซีเอ็ดยูเคชั่น จำกัด
- Ben Ezzel, "Graphics Programming in Turbo C 2.0", Addision-Wesley, 1989
- Hameg Instrument, "Oscilloscope HM205-3 Manual", Hameg Gmbh, Frankfurt, Germany